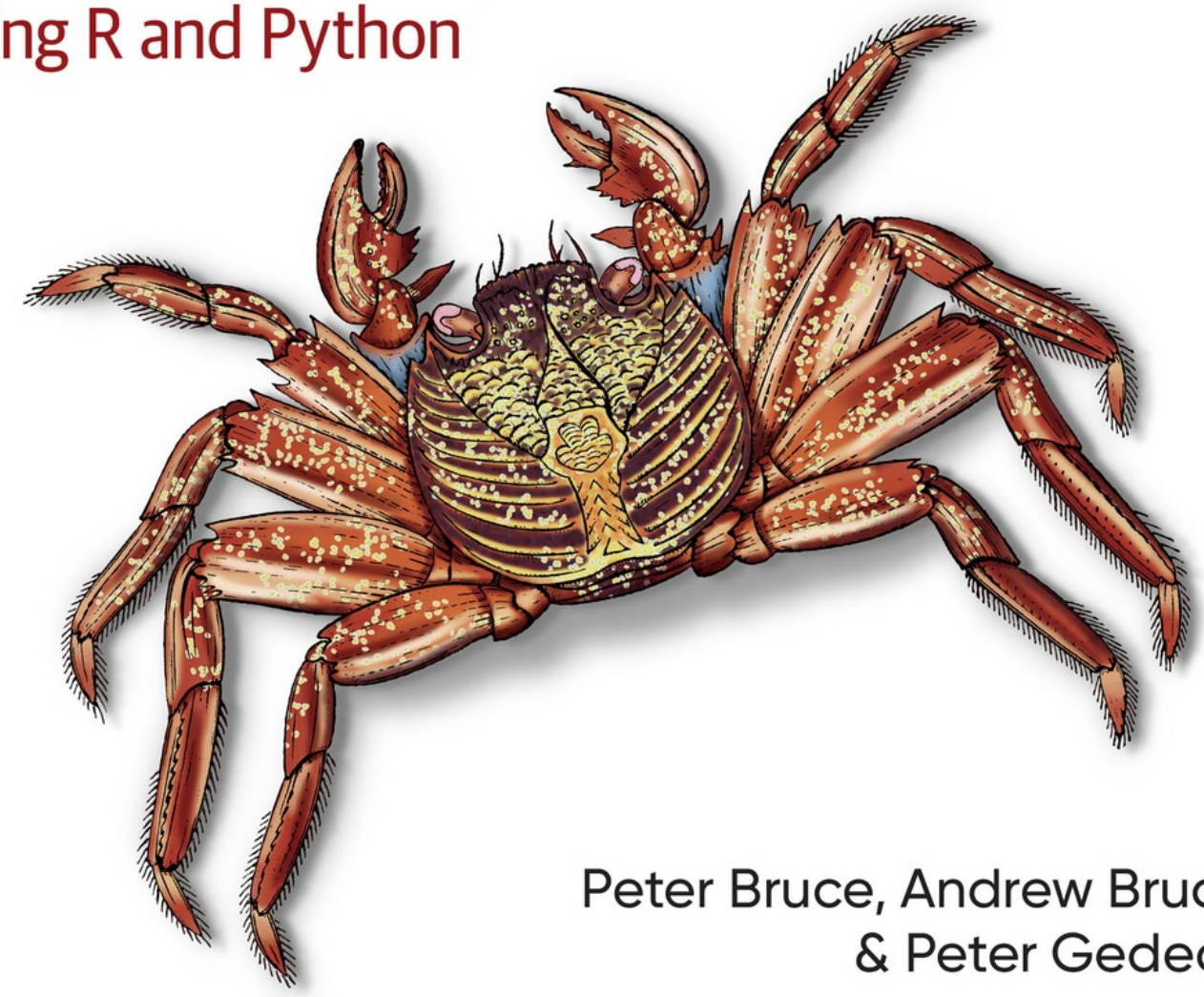
O'REILLY®

3rd Edition

AI-Assisted Statistics

for Data Scientists

50+ Essential Concepts Using R and Python



Peter Bruce, Andrew Bruce
& Peter Gedeck

AI-Assisted Statistics for Data Scientists

THIRD EDITION

50+ Essential Concepts Using R and Python

Peter Bruce, Andrew Bruce, and Peter Gedeck



AI-Assisted Statistics for Data Scientists

by Peter Bruce, Andrew Bruce, and Peter Gedeck

Copyright © 2026 Peter Bruce, Andrew Bruce, and Peter Gedeck. All rights reserved.

Published by O'Reilly Media, Inc., 141 Stony Circle, Suite 195, Santa Rosa, CA 95401.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*https://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Michelle Smith

Development Editor: Corbin Collins

Production Editor: Ashley Stussy

Copyeditor: Emily Wydeven

Proofreader: Krsta Technology Solutions

Indexers: Peter Bruce, Andrew Bruce, and Peter Gedeck

Cover Designer: Susan Brown

Cover Illustrator: Karen Montgomery

Interior Designer: David Futato

Interior Illustrator: Kate Dullea

May 2017: First Edition

May 2020: Second Edition

June 2026: Third Edition

Revision History for the First Edition

- 2026-06-16: First Release

See *https://oreilly.com/catalog/errata.csp?isbn=9798341666283* for release details.

The O’Reilly logo is a registered trademark of O’Reilly Media, Inc. *AI-Assisted Statistics for Data Scientists*, the cover image, and related trade dress are trademarks of O’Reilly Media, Inc.

The views expressed in this work are those of the authors and do not represent the publisher’s views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

979-8-341-66628-3

[LSI]

Dedication

Peter Bruce and Andrew Bruce would like to dedicate this book to the memories of our parents, Victor G. Bruce and Nancy C. Bruce, who cultivated a passion for math and science; and to our early mentors John W. Tukey and Julian Simon and our lifelong friend Geoff Watson, who helped inspire us to pursue a career in statistics.

Peter Gedeck would like to dedicate this book to Tim Clark and Christian Kramer, with deep thanks for their scientific collaboration and friendship.

Preface

This book, which has been updated with extensive new chapters and material on AI (neural networks, deep learning, and generative AI), is aimed at the data scientist and the machine learning engineer who has some prior (perhaps spotty or ephemeral) exposure to statistics. Illustrations are provided using the R and/or Python programming languages. Some familiarity with those languages is useful, but, with the *vibe coding* capabilities of modern AI tools, even those with no programming background can benefit from this book.

Two of the authors came to the world of data science from the world of statistics, and have some appreciation of the contribution that statistics can make to the art of data science. At the same time, we are well aware of the limitations of traditional statistics instruction: statistics as a discipline is a century and a half old, and most statistics textbooks and courses are laden with the momentum and inertia of an ocean liner. The methods in the first portion of this book have some connection—historical or methodological—to the discipline of statistics. Neural nets—the underpinning of modern AI—evolved mainly out of computer science and are covered in the latter part of the book.

Data scientists, machine learning practitioners, and software engineers have come to know AI as a powerful coding tool. *Vibe coding* refers to the ability of AI models to produce code based on specifications (a *prompt*) written in English or some other natural language. A related concept, *agentic coding*, is broader, covering autonomous task execution by AI agents within a structured workflow that includes goal and task specification, testing, validation, and oversight. In this book, examples in R and Python are shown, and the initial chapters (the ones that focus on statistics) discuss the use of vibe coding, primarily at the conclusion of the chapters. The latter chapters discuss AI in more depth, and go into more detail on its use for coding.

Why learn about statistics, coding, and the models that underpin AI if large language models (LLMs) can do the work for you? If you can use AI as a quasi-magical tool to do your work, why do you really need to understand the underlying concepts?

- AI can make mistakes, mistakes which may not be readily apparent.
- If used properly in statistics and data science, AI will ask you questions and offer choices that require some knowledge of the underlying *statistical* concepts to answer.
- If you have some understanding of neural nets and how—based on a statistical and machine learning foundation—they enable deep learning and generative AI, you are in a better position to understand their strengths and weaknesses.
- While AI is very effective at providing "textbook" solutions to well-defined statistical problems, it does not yet have the ability to do the kind of critical thinking and problem solving that is required to work through an ambiguous data science project from end-to-end.

With respect to statistics, this book seeks to:

- Lay out, in digestible, navigable, and easily referenced form, key concepts from statistics that are relevant to data science.
- Explain which concepts are important and useful from a data science perspective, which are less so, and why.

With respect to AI, this book seeks to:

- Provide a conceptual overview, at a high level in nontechnical terms, the statistical and machine learning foundations and the current algorithms of neural networks, deep learning, and generative AI.

- Illustrate the use of generative AI for statistical analysis.

Note that this book is not intended as a practitioners guide to deep learning and generative AI.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, keywords, packages, and libraries.

KEY TERMS

Data science is a fusion of multiple disciplines, including statistics, computer science, information technology, and domain-specific fields. As a result, several different terms could be used to reference a given concept. Key terms and their synonyms will be highlighted throughout the book in a sidebar such as this.

TIP

This element signifies a tip or suggestion.

> **NOTE**
>
> This element signifies a general note.

> **WARNING**
>
> This element indicates a warning or caution.

Using Code Examples

In all cases, this book gives code examples first in R and then in Python. In order to avoid unnecessary repetition, we generally show only output and plots created by the R code. You can find the complete code in R and Python as well as the data sets for download at *https://github.com/gedeck/ai-assisted-statistics-for-data-scientists*.

We encourage you to use *vibe coding* to duplicate or extend the examples presented in the book. We do include some examples of prompting and the AI responses to illustrate overall strategies. We have not included comprehensive detailed instructions on using AI throughout the book, as there are different tools and they are evolving rapidly. Moreover, it is vital that you gain an understanding of the statistical and machine learning methods presented here. In the future, analytics and engineering jobs will go increasingly to those who understand enough of the methods and concepts to be able to properly structure the validation, testing, and oversight of the AI tools that will increasingly be used. It will not be enough to simply submit a prompt to an AI and then put the response into production.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require

permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*AI-Assisted Statistics for Data Scientists* by Peter Bruce, Andrew Bruce, and Peter Gedeck (O'Reilly). Copyright 2026 Peter Bruce, Andrew Bruce, and Peter Gedeck, 979-8-341-66628-3."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

> **NOTE**
>
> For more than 40 years, O'Reilly Media has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.

141 Stony Circle, Suite 195

Santa Rosa, CA 95401

800-889-8969 (in the United States or Canada)

707-827-7019 (international or local)

707-829-0104 (fax)

support@oreilly.com

https://oreilly.com/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/ai-assisted-stats-for-data-scientists-3e*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly*.

Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgments

The authors acknowledge the many people who helped make this book a reality.

Aaron Schumacher gave unstintingly of his time and expertise, and helped improve the book through his many comments. Likewise, Roya Kandalan, Ari Joury, and Walter Paczkowski reviewed early drafts and contributed many helpful corrections. Toshiaki Kurokawa, who translated the first edition into Japanese, did a comprehensive job of reviewing and correcting

in the process. Anya McGuirk and Wei Xiao, statisticians at SAS, and Jay Hilfiger, fellow O'Reilly author, provided helpful feedback on initial drafts of the book's first edition. Needless to say, any errors that remain are ours alone.

At O'Reilly, Michelle Smith helped bring this third edition project to fruition, and Corbin Collins has shepherded us through the publication process with good cheer and the right amount of prodding, while Ashley Stussy smoothly took our book through the production phase. Emily Wydeven and Ayisha Mohamad corrected and improved our writing with care and patience. We also thank Marie Beaugureau, who initiated our project at O'Reilly, as well as Ben Bengfort, O'Reilly author and Statistics.com instructor, who introduced us to O'Reilly.

We, and this book, have also benefited from the many conversations both Peters have had over the years with Galit Shmueli, coauthor on other book projects.

Finally, we would like to especially thank Elizabeth Bruce, Deborah Donnell, and Janet Dobbins, whose patience and support made this endeavor possible.

Chapter 1. Exploratory Data Analysis

This chapter focuses on the first step in any data science project: exploring the data.

Classical statistics focused almost exclusively on *inference*, a sometimes complex set of procedures for drawing conclusions about large populations based on small samples. In 1962, John W. Tukey (Figure 1-1) called for a reformation of statistics in his seminal paper "The Future of Data Analysis" [Tukey-1962]. He proposed a new scientific discipline called *data analysis* that included statistical inference as just one component. Tukey forged links to the engineering and computer science communities (he coined the terms *bit*, short for binary digit, and *software*), and his original tenets are surprisingly durable and form part of the foundation for data science. The field of exploratory data analysis was established with Tukey's 1977 now-classic book *Exploratory Data Analysis* [Tukey-1977]. Tukey presented simple plots (e.g., boxplots, scatterplots) that, along with summary statistics (mean, median, quantiles, etc.), help paint a picture of a data set.

With the ready availability of computing power and expressive data analysis software, exploratory data analysis has evolved well beyond its original scope. Key drivers of this discipline have been the rapid development of new technology, access to more and bigger data, and the greater use of quantitative analysis in a variety of disciplines. David Donoho, professor of statistics at Stanford University and former undergraduate student of Tukey's, authored an excellent article based on his presentation at the Tukey Centennial workshop in Princeton, New Jersey [Donoho-2015]. Donoho traces the genesis of data science back to Tukey's pioneering work in data analysis.

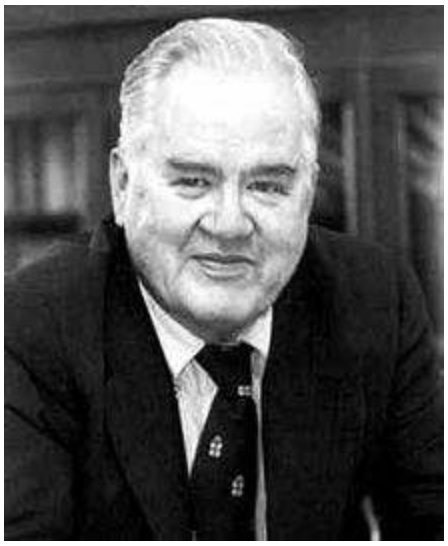


Figure 1-1. John Tukey, the eminent statistician whose ideas, developed over 50 years ago, form the foundation of data science

Elements of Structured Data

Data comes from many sources: sensor measurements, events, text, images, and videos. The *Internet of Things* (IoT) is spewing out streams of information. Much of this data is unstructured: images are a collection of pixels, with each pixel containing RGB (red, green, blue) color information. Texts are sequences of words and nonword characters, often organized by sections, subsections, and so on. Clickstreams are sequences of actions by a user interacting with an app or a web page. In fact, a major challenge of data science is to harness this torrent of raw data into actionable information. To apply the statistical concepts covered in this book, unstructured raw data must be processed and manipulated into a structured form. One of the commonest forms of structured data is a table with rows and columns—as data might emerge from a relational database or be collected for a study.

There are two basic types of structured data: numeric and categorical. Numeric data comes in two forms: *continuous*, such as wind speed or time duration, and *discrete*, such as the count of the occurrence of an event. *Categorical* data takes only a fixed set of values, such as a type of TV screen (plasma, LCD, LED, etc.) or a state name (Alabama, Alaska, etc.).

Binary data is an important special case of categorical data that takes on only one of two values, such as 0/1, yes/no, or true/false. Another useful type of categorical data is *ordinal* data in which the categories are ordered; an example of this is a numerical rating (1, 2, 3, 4, or 5).

Why do we bother with a taxonomy of data types? It turns out that for the purposes of data analysis and predictive modeling, the data type is important to help determine the type of visual display, data analysis, or statistical model. In fact, data science software, such as R and Python, uses these data types to improve computational performance. More importantly, the data type for a variable determines how software will handle computations for that variable.

KEY TERMS FOR DATA TYPES

Numeric

Data that is expressed on a numeric scale

Continuous

Data that can take on any value in an interval

Synonyms

interval, float, numeric

Discrete

Data that can take on only integer values, such as counts

Synonyms

integer, count

Categorical

Data that can take on only a specific set of values representing a set of possible categories

Synonyms

enums, enumerated, factors, nominal

Binary

A special case of categorical data with just two categories of values, e.g., 0/1, true/false

Synonyms

dichotomous, logical, indicator, Boolean

Ordinal

Categorical data that has an explicit ordering

Synonym

ordered factor

Software engineers and database programmers may wonder why we even need the notion of *categorical* and *ordinal* data for analytics. After all, categories are merely a collection of text (or numeric) values, and the underlying database automatically handles the internal representation. However, explicit identification of data as categorical, as distinct from text, does offer some advantages:

- Knowing that data is categorical can act as a signal telling software how statistical procedures, such as producing a chart or fitting a model, should behave. In particular, ordinal data can be represented as an `ordered.factor` in R, preserving a user-specified ordering in charts, tables, and models. In Python, `scikit-learn` supports ordinal data with the `sklearn.preprocessing.OrdinalEncoder`.
- Storage and indexing can be optimized (as in a relational database).
- The possible values a given categorical variable can take are enforced in the software (like an enum).

The third "benefit" can lead to unintended or unexpected behavior. If a text column is converted into a `factor`, subsequent operations on that column will assume that the only allowable values for that column are the ones originally imported, and assigning a new text value will introduce a warning and produce an `NA` (missing value). To avoid this, R changed the default behavior of data import functions (e.g., `read.csv`) starting with version 4.0.0 to treat text columns as strings. The `pandas` package in

Python will not make such a conversion automatically. However, you can specify a column as categorical explicitly in the `read_csv` function.

KEY IDEAS

- Data is typically classified in software by type.
- Data types include numeric (continuous, discrete) and categorical (binary, ordinal).
- Data typing in software acts as a signal to the software on how to process the data.

Further Reading

- The `pandas` documentation describes the different data types and how they can be manipulated in Python.
- Data types can be confusing, since types may overlap, and the taxonomy in one software may differ from that in another. The R Tutorial website covers the taxonomy for R.
- Databases are more detailed in their classification of data types, incorporating considerations of precision levels, fixed- or variable-length fields, and more; see the W3Schools guide to SQL.

Data Structures

The typical frame of reference for an analysis in data science is a rectangular data object, like a spreadsheet or database table.

Rectangular Data

Rectangular data is the general term for a two-dimensional matrix with rows indicating records (cases) and columns indicating features (variables);

data frame is the specific format in R and Python. The data doesn't always start in this form: unstructured data (e.g., text) must be processed and manipulated so that it can be represented as a set of features in the rectangular data (see "Elements of Structured Data"). Data in relational databases must be extracted and put into a single table for most data analysis and modeling tasks.

KEY TERMS FOR DATA STRUCTURES

Data frame

Rectangular data (like a spreadsheet) is the basic data structure for statistical and machine learning models.

Synonyms

table

Records

A row within a table is commonly referred to as a *record*.

Synonyms

case, data point, example, instance, observation, pattern, row, sample

Feature

A column within a table is commonly referred to as a *feature*.

Synonyms

column, independent variable, input, predictor

Outcome

Many data science projects involve predicting an *outcome*—often a yes/no outcome (in Table 1-1, it is "auction was competitive or not"). The *features* are sometimes used to predict the *outcome* in an experiment or a study.

Synonyms

dependent variable, response, target, output

Data dictionary

A document that describes the data, its source, and the meaning of variables (features).

Synonyms

data codebook, metadata

Data catalog

A structured inventory of data assets within an organization, including its location, format, and usage.

Synonyms

data inventory, data registry

Table 1-1. A typical data frame format

Category	Currency	SellerRating	Duration
Music/Movie/Game	US	3249	5
Music/Movie/Game	US	3249	5
Automotive	US	3115	7
Automotive	US	3115	7
Automotive	US	3115	7
Automotive	US	3115	7
Automotive	US	3115	7
Automotive	US	3115	7

In Table 1-1, there is a mix of measured or counted data (e.g., duration and price) and categorical data (e.g., category and currency). As mentioned earlier, a special form of categorical variable is a binary (yes/no or 0/1) variable, seen in the rightmost column in Table 1-1—an indicator variable showing whether an auction was competitive (had multiple bidders) or not. This indicator variable also happens to be an *outcome* variable, when the scenario is to predict whether an auction is competitive or not.

Data Frames and Indexes

Traditional database tables have one or more columns designated as an index, essentially a row number. This can vastly improve the efficiency of certain database queries. In Python, with the `pandas` library, the basic rectangular data structure is a `DataFrame` object. By default, an automatic integer index is created for a `DataFrame` based on the order of the rows. In `pandas`, it is also possible to set multilevel/hierarchical indexes to improve the efficiency of certain operations.

In R, the basic rectangular data structure is a `data.frame` object. A `data.frame` also has an implicit integer index based on the row order. The native R `data.frame` does not support user-specified or multilevel indexes, though a custom key can be created through the `row.names` attribute. To overcome this deficiency, several packages have gained widespread use: `data.table` and the collection of `tidyverse` packages. The `data.table` package supports multilevel indexes and offers significant speedups in working with a `data.frame`. The `tidyverse` package collection provides a wide variety of tools for data manipulation. Due to its popularity, we will use `tidyverse`'s `tibble` in this book.

TERMINOLOGY DIFFERENCES

Terminology for rectangular data can be confusing. Statisticians and data scientists use different terms for the same thing. For a statistician, *predictor variables* are used in a model to predict a *response* or *dependent variable*. For a data scientist, *features* are used to predict a *target*. One synonym is particularly confusing: computer scientists will use the term *sample* for a single row; a *sample* to a statistician means a collection of rows.

Nonrectangular Data Structures

There are other data structures besides rectangular data.

Time-series data records successive measurements of the same variable. It is the raw material for statistical forecasting methods, and it is also a key component of the data produced by devices—the IoT.

Spatial data structures, which are used in mapping and location analytics, are more complex and varied than rectangular data structures. In the *object* representation, the focus of the data is an object (e.g., a house) and its spatial coordinates. The *field* view, by contrast, focuses on small units of space and the value of a relevant metric (pixel brightness, for example).

Graph (or network) data structures are used to represent physical, social, and abstract relationships. For example, a graph of a social network, such as Facebook or LinkedIn, may represent connections between people on the network. Distribution hubs connected by roads are an example of a physical network. Graph structures are useful for certain types of problems, such as network optimization and recommender systems.

Each of these data types has its specialized methodology in data science. The focus of this book is on rectangular data, the fundamental building block of predictive modeling.

GRAPHS IN STATISTICS

In computer science and information technology, the term *graph* typically refers to a depiction of the connections among entities, and to the underlying data structure. In statistics, *graph* is used to refer to a variety of plots and *visualizations*, not just of connections among entities, and the term applies only to the visualization, not to the data structure.

KEY IDEAS

- The basic data structure in data science is a rectangular matrix in which rows are records and columns are variables (features).
- Terminology can be confusing; there are a variety of synonyms arising from the different disciplines that contribute to data science (statistics, computer science, and information technology).

Further Reading

- Documentation on data frames in R
- Documentation on data frames in Python

Data Dictionaries and Catalogs

From the feature names in Table 1-1 you can make a reasonable guess about what the feature represents, but feature names are always short, are often cryptic, and sometimes carry no information at all. It is good practice to capture additional information in so-called data dictionaries. A *data dictionary* is a document that describes the data, its source, and the meaning of the variables (features). They are particularly important for the correct analysis and interpretation of data science results. For example (referring to Table 1-1):

- The Currency column contains the currency a particular auction used. It is categorical, and each row has one of three values: `US`, which stands for US dollar; `GBP` for English pound; and `EUR` for the European euro.
- The Duration column defines the length of the auction. However, we don't know this for sure without consulting a data dictionary.

In general, a data dictionary should contain information about the data set as a whole, as well as information about each variable (feature). It can take many forms; it could be a simple table or a more complex document. A sample data dictionary for the data set in Table 1-1 is shown in the following example. In this example, we formatted the information using *YAML*, a machine-readable format. This semi-structured format makes it particularly useful in AI-based data analysis, while still readable by humans:

```
name: ebay Auctions
description: >
    The dataset eBayAuctions.csv contains information on 1972 auctions that
```

```
    transacted on eBay.com during May-June 2004.
source: Copyright 2016 Galit Shmueli and Peter Bruce
size:
    observations: 1972
    features: 8
features:
    Category: Category of the auctioned item
    currency: "US: US dollar, GBP: English pound, EUR: Euro"
    sellerRating: >
        a rating by eBay, as a function of the number of "good" and  "bad"
        transactions the seller had on eBay
    Duration: Number of days the auction lasted (set by seller at auction
start)
    endDay: Day of week that the auction closed
    ClosePrice: Price item sold at (converted into USD)
    OpenPrice: Initial price set by the seller (converted into USD)
    Competitive?: whether the auction had a single bid (0) or more (1)
```

The keys `name`, `description`, `source`, and `size` are metadata about the data set as a whole. The `features` section gives a short description for each feature. A more complete version would also include information about the data type, the unit of the features, and permissible values.

Data catalogs are a more comprehensive description of the data within an organization. A data catalog can be a collection of data dictionaries, but it also contains additional information about the data, such as where it is stored, who has access to it, and how it can be used. A data catalog is a useful tool for data discovery and data governance.

A large language model (LLM) can help you to improve a given data dictionary or get you started developing one. In both cases, it is crucial that the outcome is thoroughly examined and inaccuracies and mistakes corrected. We use the Python `langchain` package as a high-level interface to LLMs. The communication with the LLM is done through a prompt. Here is an example prompt that tasks an LLM to improve a data dictionary with additional information. In addition to the data dictionary, we provide a sample of the data in CSV format to give the LLM additional context:

```
PROMPT = """
Improve the attached data dictionary and return it in YAML format. In addition
to the description of features, provide name, description, source, and size
```

```
of the dataset.

For each feature, provide the following information:
- feature name
- readable variable name (if required)
- definition of the variable
- data type
- measurement units
- allowed values (for categorical or nominal features, but only if you are
sure)

Here is the data dictionary in YAML format:

{data_dictionary}  ❶

First ten lines of the dataset in CSV format:

{data_sample}
"""
```

❶ `{data_dictionary}` and `{data_sample}` are placeholders which will be replaced with actual data.

In this example, we use the `GPT-5` model from OpenAI. To execute the code, you will need an API key from OpenAI that gives you access to this model:[1]

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate

chain = (
  PromptTemplate.from_template(PROMPT) |  ❶
  ChatOpenAI(model="gpt-5")  ❷
)
```

❶ The text template is converted into a `langchain` prompt template. The vertical line (`|`) is used to chain the prompt template with the LLM. This means after processing the prompt template, the resulting prompt is passed to the `ChatOpenAI` object.

❷ This specifies the use of the OpenAI model. There are similar interfaces for other LLM providers.

The `chain` object can now be used to generate an improved data dictionary. We provide the existing data dictionary and a sample of the data as input:

```
import mlba  ❶

data_dictionary = """
name: ebay Auctions
description: >
    The dataset eBayAuctions.csv contains information on 1972 auctions that
    transacted on eBay.com during May-June 2004.
source: Copyright 2016 Galit Shmueli and Peter Bruce
size:
    observations: 1972
    features: 8
features:
    Category: Category of the auctioned item
    currency: "US: US dollar, GBP: English pound, EUR: Euro"
    sellerRating: >
        a rating by eBay, as a function of the number of "good" and  "bad"
        transactions the seller had on eBay
    Duration: Number of days the auction lasted (set by seller at auction
start)
    endDay: Day of week that the auction closed
    ClosePrice: Price item sold at (converted into USD)
    OpenPrice: Initial price set by the seller (converted into USD)
    Competitive?: whether the auction had a single bid (0) or more (1)
"""

eBayAuctions = mlba.load_data("eBayAuctions")
data_sample = eBayAuctions.sample(10, random_state=123).to_csv(index=False)

context = {
    "data_dictionary": data_dictionary,
    "data_sample": data_sample,
}
result = chain.invoke(context)
print(result.content)
```

❶ The `mlba` package provides access to a variety of data sets for machine learning and business analytics, including the `eBayAuctions` data set used in this example.

The output is an improved data dictionary in YAML format. Here is an excerpt:

```yaml
name: eBay Auctions
description: >
  This data set contains information on 1,972 auctions that transacted on
  eBay.com during May-June 2004. It includes item category, listing currency,
  seller reputation, auction duration and closing weekday, opening and closing
  prices (in USD), and an indicator of whether the auction attracted more
  than one bid.
source: Copyright 2016 Galit Shmueli and Peter Bruce
size:
  observations: 1972
  features: 8
notes:
  - All monetary amounts (OpenPrice and ClosePrice) are converted to U.S.
    dollars (USD).
  - The currency field records the original listing currency.
  - Auction durations in this data set are the standard eBay options.
features:
  - name: Category
    readable_name: category
    definition: Nominal category of the auctioned item as specified by the
      seller/eBay category taxonomy.
    data_type: categorical (string)
    units: N/A

  - name: currency
    readable_name: currency
    definition: Original listing currency for the auction.
    data_type: categorical (string)
    units: N/A
    allowed_values:
      - US   # U.S. dollar
      - GBP  # British pound
      - EUR  # Euro
...
```

Due to the random nature of LLMs, the output will vary each time you run the code. The quality of the output also depends on the selected model. While the result here seems reasonable, statements like “Auction durations in this data set are the standard eBay options” must be verified by a human expert. It can also be better to improve the data dictionary in an iterative

process, where the output of one run is used as input for the next run and further improved using refined prompts.

In R, we can use the `ellmer` package that is part of the `tidyverse` package collection. The code is similar to the Python `langchain` example, so we only show the key parts here:

```
library(tidyverse)
library(ellmer)

system_prompt <- "..."
data_dictionary <- "..."
data_sample <- "..."
message <- paste(c(
  "Here is the data dictionary in YAML format:",
  data_dictionary,
  "First ten lines of the dataset in CSV format:",
  data_sample
), collapse = "\n")
chat <- chat_openai(system_prompt, model = "gpt-4o-mini")  ❶
response <- chat$chat(message)  ❷
response
```

❶ The `chat_openai` function specifies that we want to use an OpenAI model. There are similar interfaces for other LLM providers.

❷ The `chat` method sends `system_prompt` and `message` to the LLM service and returns the result.

Further Reading

- How to Make a Data Dictionary
- Create a Data Catalog for Discovery

Estimates of Location

Variables with measured or count data might have thousands of distinct values. A basic step in exploring your data is getting a "typical value" for

each feature (variable): an estimate of where most of the data is located (i.e., its central tendency).

KEY TERMS FOR ESTIMATES OF LOCATION

Mean

The sum of all values divided by the number of values

Synonym

average

Weighted mean

The sum of all values times a weight divided by the sum of the weights

Synonym

weighted average

Median

The value such that one-half of the data lies above and below

Synonym

50th percentile

Percentile

The value such that *P* percent of the data lies below

Synonym

quantile

Weighted median

The value such that one-half of the sum of the weights lies above and below the sorted data

Trimmed mean

The average of all values after dropping a fixed number of extreme values

Synonym

truncated mean, Winsorized mean

Robust

Not sensitive to extreme values

Synonym

resistant

Outlier

A data value that is very different from most of the data

Synonym

extreme value

At first glance, summarizing data might seem fairly trivial: just take the *mean* of the data. In fact, while the mean is easy to compute and expedient to use, it may not always be the best measure for a central value. For this reason, statisticians have developed and promoted several alternative estimates to the mean.

METRICS AND ESTIMATES

Statisticians often use the term *estimate* for a value calculated from the data at hand, to draw a distinction between what we see from the data and the theoretical true or exact state of affairs. Data scientists and business analysts are more likely to refer to such a value as a *metric*. The difference reflects the approach of statistics versus that of data science: accounting for uncertainty lies at the heart of the discipline of statistics, whereas concrete business or organizational objectives are the focus of data science. Hence, statisticians estimate, and data scientists measure.

Mean

The most basic estimate of location is the mean, or *average* value. The mean is the sum of all values divided by the number of values. Consider the following set of numbers: {3, 5, 1, 2}. The mean is (3 + 5 + 1 + 2) / 4 = 11 / 4 = 2.75. You will encounter the symbol x (pronounced "x-bar") being used to represent the mean of a sample from a population. The formula to compute the mean for a set of n values $x_1, x_2, \ldots, x_n$ is:

$$\text{Mean} = \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

N (OR *n*)

N (or *n*) refers to the total number of records or observations. In statistics it is capitalized if it is referring to a population, and lowercase if it refers to a sample from a population. In data science, that distinction is not vital, so you may see it both ways.

A variation of the mean is a *trimmed mean*, which you calculate by dropping a fixed number of sorted values at each end and then taking an average of the remaining values. Representing the sorted values by $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$ where $x_{(1)}$ is the smallest value and $x_{(n)}$ the largest, the formula to compute the trimmed mean with p smallest and largest values omitted is:

$$\text{Trimmed mean} = x = \frac{\sum_{i=p+1}^{n-p} x_{(i)}}{n - 2p}$$

A trimmed mean eliminates the influence of extreme values. For example, in international diving the top score and bottom score from five judges are dropped, and the final score is the average of the scores from the three remaining judges. This makes it difficult for a single judge to manipulate the score, perhaps to favor their country's contestant. Trimmed means are widely used, and in many cases are preferable to using the ordinary mean—see "Median and Robust Estimates" for further discussion.

Another type of mean is a *weighted mean*, which you calculate by multiplying each data value x_i by a user-specified weight w_i and dividing their sum by the sum of the weights. The formula for a weighted mean is:

$$\text{Weighted mean} = x_w = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

There are two main motivations for using a weighted mean:

- Some values are intrinsically more variable than others, and highly variable observations are given a lower weight. For example, if we are taking the average from multiple sensors and one of the sensors is less accurate, then we might downweight the data from that sensor.
- The data collected does not equally represent the different groups that we are interested in measuring. For example, because of the way an online experiment was conducted, we may not have a set of data that accurately reflects all groups in the user base. To correct that, we can give a higher weight to the values from the groups that were underrepresented.

Median and Robust Estimates

The *median* is the middle number on a sorted list of the data. If there is an even number of data values, the middle value is one that is not actually in the data set, but rather the average of the two values that divide the sorted data into upper and lower halves. Compared to the mean, which uses all observations, the median depends only on the values in the center of the sorted data. While this might seem to be a disadvantage, since the mean is much more sensitive to the data, there are many instances in which the median is a better metric for location. Let's say we want to look at typical household incomes in neighborhoods around Lake Washington in Seattle. In comparing the Medina neighborhood to the Windermere neighborhood, using the mean would produce very different results because Bill Gates lives in Medina. If we use the median, it won't matter how rich Bill Gates is—the position of the middle observation will remain the same.

For the same reasons that one uses a weighted mean, it is also possible to compute a *weighted median*. As with the median, we first sort the data, although each data value has an associated weight. Instead of the middle number, the weighted median is a value such that the sum of the weights is equal for the lower and upper halves of the sorted list. Like the median, the weighted median is robust to outliers.

Outliers

The median is referred to as a *robust* estimate of location since it is not influenced by *outliers* (extreme cases) that could skew the results. An outlier is any value that is very distant from the other values in a data set. The exact definition of an outlier is somewhat subjective, although certain conventions are used in various data summaries and plots (see "Percentiles and Boxplots"). Being an outlier in itself does not make a data value invalid or erroneous (as in the previous example with Bill Gates). Still, outliers are often the result of data errors such as mixing data of different units (kilometers versus meters) or bad readings from a sensor. When outliers are the result of bad data, the mean will result in a poor estimate of location, while the median will still be valid. In any case, outliers should be identified and are usually worthy of further investigation.

ANOMALY DETECTION

In contrast to typical data analysis, where outliers are sometimes informative and sometimes a nuisance, in *anomaly detection* the points of interest are the outliers, and the greater mass of data serves primarily to define the "normal" against which anomalies are measured.

The median is not the only robust estimate of location. In fact, a trimmed mean is widely used to avoid the influence of outliers. For example, trimming the bottom and top 10% (a common choice) of the data will provide protection against outliers in all but the smallest data sets. The trimmed mean can be thought of as a compromise between the median and the mean: it is robust to extreme values in the data, but uses more data to calculate the estimate for location.

OTHER ROBUST METRICS FOR LOCATION

Statisticians have developed a plethora of other estimators for location, primarily with the goal of developing an estimator more robust than the mean and also more efficient (i.e., better able to discern small location differences between data sets). While these methods are potentially useful for small data sets, they are not likely to provide added benefit for large or even moderately sized data sets.

Example: Location Estimates of Population and Murder Rates

Table 1-2 shows the first few rows in the data set containing population and murder rates (in units of murders per 100,000 people per year) for each US state (2010 Census).

Table 1-2. A few rows of the state `data.frame`, showing population and murder state

	State	Population	Murder rate	Abb
1	Alabama	4,779,736	5.7	AL
2	Alaska	710,231	5.6	AK
3	Arizona	6,392,017	4.7	AZ
4	Arkansas	2,915,918	5.6	AR
5	California	37,253,956	4.4	CA
6	Colorado	5,029,196	2.8	CO
7	Connecticut	3,574,097	2.4	CT
8	Delaware	897,934	5.8	DE

Compute the mean, trimmed mean, and median for the population using R:

```
library(tidyverse)
state <- read_csv("state.csv", show_col_types = FALSE)
population <- state[["Population"]]
mean(population)
mean(population, trim = 0.1)
median(population)
---
[1] 6162876
[1] 4783697
[1] 4436370
```

To compute mean and median in Python we can use the `pandas` methods of the data frame. The trimmed mean requires the `trim_mean` function in `scipy.stats`:

```
import pandas as pd
from scipy.stats import trim_mean

state = pd.read_csv("state.csv")
print(f"Mean: {state['Population'].mean()}")
print(f"Trimmed mean: {trim_mean(state['Population'], 0.1)}")
print(f"Median: {state['Population'].median()}")
```

The mean is bigger than the trimmed mean, which is bigger than the median.

This is because the trimmed mean excludes the largest and smallest five states (`trim=0.1` drops 10% from each end). If we want to compute the average murder rate for the country, we need to use a weighted mean or median to account for different populations in the states. Since base R doesn't have a function for weighted median, we need to install a package such as `matrixStats`:

```
library(matrixStats)

weighted.mean(state[["Murder.Rate"]], w = state[["Population"]])
weightedMedian(state[["Murder.Rate"]], w = state[["Population"]])
---
[1] 4.445834
[1] 4.4
```

Weighted mean is available with `NumPy`. For weighted median, we can use the specialized package `wquantiles`:

```
import numpy as np
import wquantiles

weighted_mean = np.average(state["Murder.Rate"], weights=state["Population"])
weighted_quantile = wquantiles.median(state["Murder.Rate"],
                                      weights=state["Population"])
```

```
print(f"Weighted mean: {weighted_mean}")
print(f"Weighted median: {weighted_quantile}")
```

In this case, the weighted mean and the weighted median are about the same.

KEY IDEAS

- The basic metric for location is the mean, but it can be sensitive to extreme values (outliers).
- Other metrics (median, trimmed mean) are less sensitive to outliers and unusual distributions and hence are more robust.

Further Reading

- The Wikipedia article on central tendency contains an extensive discussion of various measures of location.
- John Tukey's 1977 classic *Exploratory Data Analysis* (Addison-Wesley) is still widely read.

Estimates of Variability

Location is just one dimension in summarizing a feature. A second dimension, *variability*, also referred to as *dispersion*, measures whether the data values are tightly clustered or spread out. At the heart of statistics lies variability: measuring it, reducing it, distinguishing random from real variability, identifying the various sources of real variability, and making decisions in the presence of it.

KEY TERMS FOR VARIABILITY METRICS

Deviations

The difference between the observed values and the estimate of location

Synonyms

errors, residuals

Variance

The sum of squared deviations from the mean divided by $n - 1$ where n is the number of data values

Synonym

mean squared error

Standard deviation

The square root of the variance

Mean absolute deviation

The mean of the absolute values of the deviations from the mean

Synonyms

L1-norm, Manhattan norm

Median absolute deviation from the median

The median of the absolute values of the deviations from the median

Range

The difference between the largest and the smallest value in a data set

Order statistics

Metrics based on the data values sorted from smallest to biggest

Synonym

ranks

Percentile

The value such that *P* percent of the values take on this value or less and (100 – P) percent take on this value or more

Synonym

quantile

Interquartile range

The difference between the 75th percentile and the 25th percentile

Synonym

IQR

Just as there are different ways to measure location (mean, median, etc.), there are also different ways to measure variability.

Standard Deviation and Related Estimates

The most widely used estimates of variation are based on the differences, or *deviations*, between the estimate of location and the observed data. For a set of data {1, 4, 4}, the mean is 3 and the median is 4. The deviations from the

mean are the differences: 1 – 3 = –2, 4 – 3 = 1, 4 – 3 = 1. These deviations tell us how dispersed the data is around the central value.

One way to measure variability is to estimate a typical value for these deviations. Averaging the deviations themselves would not tell us much—the negative deviations offset the positive ones. In fact, the sum of the deviations from the mean is precisely zero. Instead, a simple approach is to take the average of the absolute values of the deviations from the mean. In the preceding example, the absolute value of the deviations is {2, 1, 1}, and their average is (2 + 1 + 1) / 3 = 1.33. This is known as the *mean absolute deviation* and is computed with the formula:

$$\text{Mean absolute deviation} = \frac{\sum_{i=1}^{n} |x_i - x|}{n}$$

where x is the sample mean.

The best-known estimates of variability are the *variance* and the *standard deviation*, which are based on squared deviations. The variance is an average of the squared deviations, and the standard deviation is the square root of the variance:

$$\begin{aligned} \text{Variance} &= s^2 = \frac{\sum_{i=1}^{n} (x_i - x)^2}{n - 1} \\ \text{Standard deviation} &= s = \sqrt{\text{Variance}} \end{aligned}$$

The standard deviation is much easier to interpret than the variance since it is on the same scale as the original data. Still, with its more complicated and less intuitive formula, it might seem peculiar that the standard deviation is preferred in statistics over the mean absolute deviation. It owes its preeminence to statistical theory: mathematically, working with squared values is much more convenient than absolute values, especially for statistical models.

DEGREES OF FREEDOM, AND *N* OR *N* – 1?

In statistics books, there is always some discussion of why we have $n - 1$ in the denominator in the variance formula, instead of n, leading into the concept of *degrees of freedom*. This distinction is not important since n is generally large enough that it won't make much difference whether you divide by n or $n - 1$. But in case you are interested, here is the story. It is based on the premise that you want to make estimates about a population, based on a sample.

If you use the intuitive denominator of n in the variance formula, you will underestimate the true value of the variance and the standard deviation in the population. This is referred to as a *biased* estimate. However, if you divide by $n - 1$ instead of n, the variance becomes an *unbiased* estimate.

To fully explain why using n leads to a biased estimate involves the notion of degrees of freedom, which takes into account the number of constraints in computing an estimate. In this case, there are $n - 1$ degrees of freedom since there is one constraint: the standard deviation depends on calculating the sample mean. For most problems, data scientists do not need to worry about degrees of freedom.

Neither the variance, the standard deviation, nor the mean absolute deviation is fully robust to outliers and extreme values (see "Median and Robust Estimates" for a discussion of robust estimates for location). The variance and standard deviation are especially sensitive to outliers since they are based on the squared deviations; more robust is the *median absolute deviation from the median*, or MAD:

$$\text{Median absolute deviation} = \text{Median}(|x_1 - m|, |x_2 - m|, ..., |x_N - m|)$$

where m is the median. Like the median, the MAD is not influenced by extreme values. It is also possible to compute a trimmed standard deviation analogous to the trimmed mean (see "Mean").

NOTE

The variance, the standard deviation, the mean absolute deviation, and the median absolute deviation from the median are not equivalent estimates, even in the case where the data comes from a normal distribution. In fact, the standard deviation is always greater than the mean absolute deviation, which itself is greater than the median absolute deviation. Sometimes, the median absolute deviation is multiplied by a constant scaling factor to put the MAD on the same scale as the standard deviation in the case of a normal distribution. The commonly used factor of 1.4826 means that 50% of the normal distribution fall within the range ±MAD (see, e.g., the relation to standard deviation).

Estimates Based on Percentiles

A different approach to estimating dispersion is based on looking at the spread of the sorted data. Statistics based on sorted (ranked) data are referred to as *order statistics*. The most basic measure is the *range*: the difference between the largest and smallest numbers. The minimum and maximum values themselves are useful to know and are helpful in identifying outliers, but the range is extremely sensitive to outliers and not very useful as a general measure of dispersion in the data.

To avoid the sensitivity to outliers, we can look at the range of the data after dropping values from each end. Formally, these types of estimates are based on differences between *percentiles*. In a data set, the *P*th percentile is a value such that at least *P* percent of the values take on this value or less and at least (100 – *P*) percent of the values take on this value or more. For example, to find the 80th percentile, sort the data. Then, starting with the smallest value, proceed 80% of the way to the largest value. Note that the median is the same thing as the 50th percentile. The percentile is essentially the same as a *quantile*, with quantiles indexed by fractions (so the 0.8 quantile is the same as the 80th percentile).

A common measurement of variability is the difference between the 25th percentile and the 75th percentile, called the *interquartile range* (or IQR). Here is a simple example: {3, 1, 5, 3, 6, 7, 2, 9}. We sort these to get {1, 2, 3, 3, 5, 6, 7, 9}. The 25th percentile is at 2.5, and the 75th percentile is at

6.5, so the IQR is 6.5 – 2.5 = 4. Software can have slightly differing approaches that yield different answers (see "Percentile: Precise Definition"); typically, these differences are smaller.

For very large data sets, calculating exact percentiles can be computationally very expensive since it requires sorting all the data values. Machine learning and statistical software use special algorithms, such as [Zhang-Wang-2007], to get an approximate percentile that can be calculated very quickly and is guaranteed to have a certain accuracy.

PERCENTILE: PRECISE DEFINITION

If we have an even number of data (n is even), then the percentile is ambiguous under the preceding definition. In fact, we could take on any value between the order statistics $x_{(j)}$ and $x_{(j+1)}$ where j satisfies:

$$100 * \frac{j}{n} \leq P < 100 * \frac{j+1}{n}$$

Formally, the percentile is the weighted average:

$$\text{Percentile}\,(P) = (1-w)x_{(j)} + wx_{(j+1)}$$

for some weight w between 0 and 1. Statistical software has slightly differing approaches to choosing w. In fact, the R function `quantile` offers nine different alternatives to compute the quantile. Except for small data sets, you don't usually need to worry about the precise way a percentile is calculated. Python's `numpy.quantile` supports nine approaches as well, with linear interpolation being the default.

Example: Variability Estimates of State Population

Table 1-3 (repeated from Table 1-2 for convenience) shows the first few rows in the data set containing population and murder rates for each state.

Table 1-3. A few rows of the state `data.frame`, showing population and murder state

	State	Population	Murder rate	Abb
1	Alabama	4,779,736	5.7	AL
2	Alaska	710,231	5.6	AK
3	Arizona	6,392,017	4.7	AZ
4	Arkansas	2,915,918	5.6	AR
5	California	37,253,956	4.4	CA
6	Colorado	5,029,196	2.8	CO
7	Connecticut	3,574,097	2.4	CT
8	Delaware	897,934	5.8	DE

Using R's built-in functions for the standard deviation, the IQR, and the median absolute deviation from the median (MAD), we can compute estimates of variability for the state population data:

```
sd(state[["Population"]])
IQR(state[["Population"]])
mad(state[["Population"]])
---
[1] 6848235
[1] 4847308
[1] 3849870
```

The `pandas` data frame provides methods for calculating standard deviation and quantiles. Using the quantiles, we can easily determine the IQR. As an alternative, we can use the `numpy` functions `std` and `quantile`. For the robust MAD, we use the function `robust.scale.mad` from the `statsmodels` package:

```
from statsmodels import robust

print(state["Population"].std())
print(state["Population"].quantile(0.75) - state["Population"].quantile(0.25))
print(robust.scale.mad(state["Population"]))
```

The standard deviation is almost twice as large as the MAD (in R, by default, the scale of the MAD is adjusted to be on the same scale as the standard deviation). This is not surprising since the standard deviation is sensitive to outliers.

KEY IDEAS

- Variance and standard deviation are the most widespread and routinely reported statistics of variability.
- Both are sensitive to outliers.
- More robust metrics include mean absolute deviation, median absolute deviation from the median, and percentiles (quantiles).

Further Reading

- David Lane's online statistics resource has a section on percentiles.
- Kevin Davenport has a useful post on R-Bloggers about deviations from the median and their robust properties.

Exploring the Data Distribution

Each of the estimates we've covered sums up the data in a single number to describe the location or variability of the data. It is also useful to explore how the data is distributed overall.

KEY TERMS FOR EXPLORING THE DISTRIBUTION

Boxplot

A plot introduced by Tukey as a quick way to visualize the distribution of data.

Synonym

box and whiskers plot

Frequency table

A tally of the count of numeric data values that fall into a set of intervals (bins).

Histogram

A plot of the frequency table with the bins on the x-axis and the count (or proportion) on the y-axis. While visually similar, bar charts should not be confused with histograms. See "Exploring Binary and Categorical Data" for a discussion of the difference.

Density plot

A smoothed version of the histogram, often based on a *kernel density estimate*.

Percentiles and Boxplots

In "Estimates Based on Percentiles", we explored how percentiles can be used to measure the spread of the data. Percentiles are also valuable for summarizing the entire distribution. It is common to report the quartiles (25th, 50th, and 75th percentiles) and the deciles (the 10th, 20th,…, 90th percentiles). Percentiles are especially valuable for summarizing the *tails* (the outer range) of the distribution. Popular culture has coined the term *one-percenters* to refer to the people in the top 99th percentile of wealth.

Table 1-4 displays some percentiles of the murder rate by state. In R, this would be produced by the `quantile` function:

```
quantile(state[["Murder.Rate"]], p = c(0.05, 0.25, 0.5, 0.75, 0.95))
---
   5%   25%   50%   75%   95%
1.600 2.425 4.000 5.550 6.510
```

The `pandas` data frame method `quantile` provides it in Python:

```
state[["Murder.Rate"]].quantile([0.05, 0.25, 0.5, 0.75, 0.95])
```

Table 1-4. Percentiles of murder rate by state

5%	25%	50%	75%	95%
1.60	2.42	4.00	5.55	6.51

The median is 4 murders per 100,000 people, although there is quite a bit of variability: the 5th percentile is only 1.6 and the 95th percentile is 6.51.

Boxplots, introduced by Tukey [Tukey-1977], are based on percentiles and give a quick way to visualize the distribution of data. Figure 1-2 shows a boxplot of the population by state produced by R:

```
ggplot(state, aes(y = Population / 1000000)) +
  geom_boxplot(staplewidth = 0.25) +  ❶
```

```
    labs(y = "Population (millions)") +
    scale_x_discrete(labels = NULL, breaks = NULL)
```

❶ The argument `staplewidth` controls the width of the staple, the small horizontal line, at the end of the whiskers.

`pandas` provides a number of basic exploratory plots for data frames; one of them is boxplots:

```
ax = (state["Population"] / 1_000_000).plot.box(figsize=(3, 4))  ❶
ax.set_ylabel("Population (millions)")  ❷
```

❶ The argument `figsize` controls the size of the figure in inches. The plot function returns a `Matplotlib` axes object (`ax`).

❷ The method `set_ylabel` sets the label for the y-axis. There are many other methods that can be used to customize the plot.

From the boxplot in Figure 1-2 we can immediately see that the median state population is about 5 million, half the states fall between about 2 million and about 7 million, and there are some high-population outliers. The top and bottom of the box are the 75th and 25th percentiles, respectively. The median is shown by the horizontal line in the box. The vertical lines, referred to as *whiskers*, extend from the top and bottom of the box to indicate the range for the bulk of the data. There are many variations of a boxplot; see, for example, the documentation for the R function `geom_boxplot`. By default, the R function extends the whiskers to the furthest point beyond the box, except that it will not go beyond 1.5 times the IQR. *Matplotlib* uses the same implementation; other software may use a different rule.

Any data outside of the whiskers is plotted as single points or circles (often considered outliers).

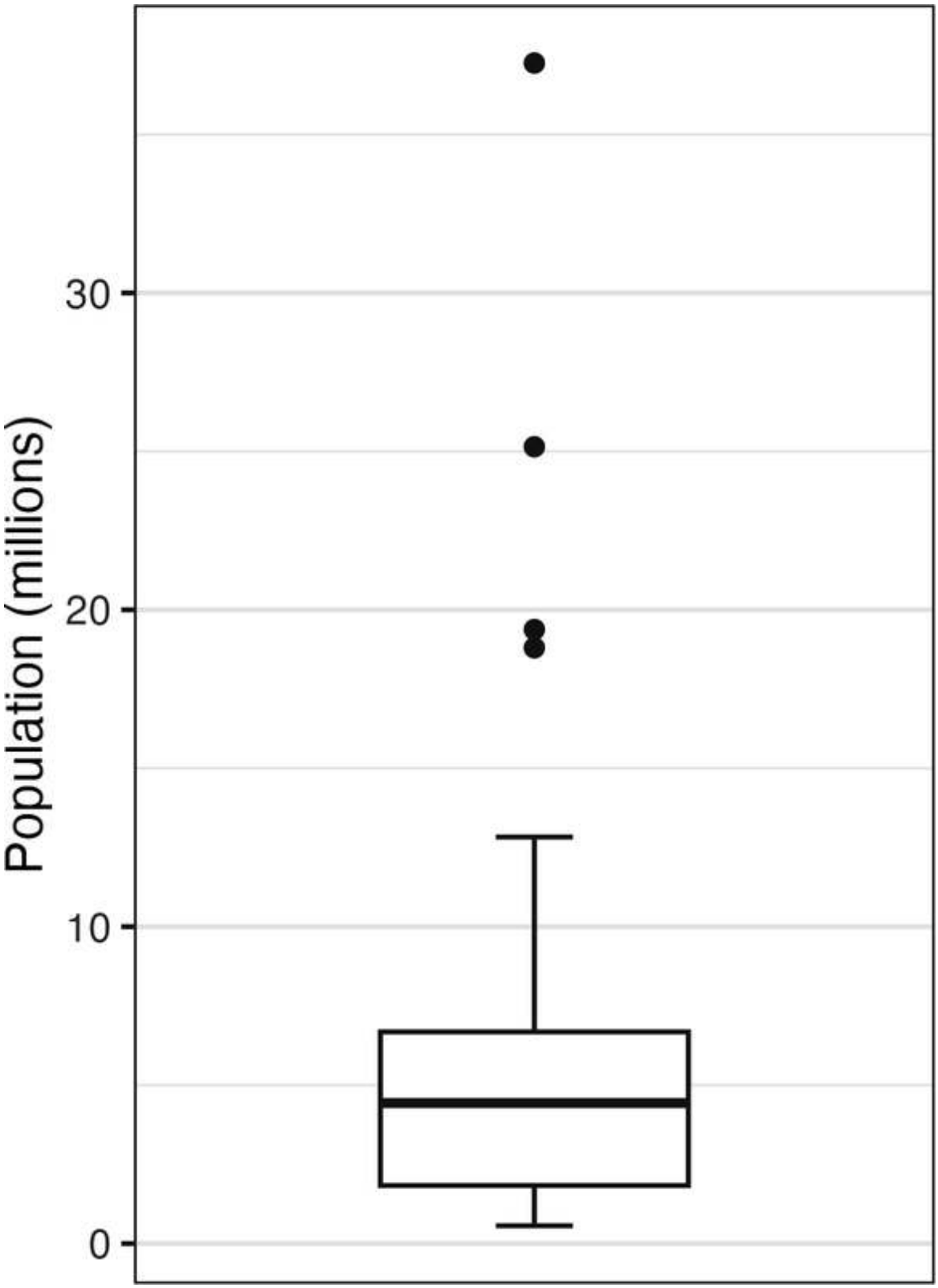


Figure 1-2. Boxplot of state populations

Frequency Tables and Histograms

A frequency table of a variable divides up the variable range into equally spaced segments and tells us how many values fall within each segment. Table 1-5 shows a frequency table of the population by state computed in R:

```
breaks <- seq(
  from = min(state[["Population"]]),
  to = max(state[["Population"]]), length = 11)
state %>%
  mutate(BinRange = cut(Population, breaks = breaks,  ❶
      right = TRUE, include.lowest = TRUE)) %>%
  count(BinRange, .drop = FALSE)  ❷
```

❶ The `breaks` argument defines the bin boundaries. Instead we could also just specify the number of bins.

❷ Set `.drop = FALSE` to include the empty bins in the result.

The function `pandas.cut` creates a series that maps the values into the segments. Using the method `value_counts`, we get the frequency table:

```
binned_population = pd.cut(state["Population"], 10)
binned_population.value_counts()
```

Table 1-5. A frequency table of population by state

BinNumber	BinRange	Count	States
1	563,626–4,232,658	24	WY, VT, ND, AK, SD, DE, MT, RI, NH, ME, HI, ID, NE, WV, NM, NV, UT, KS, AR, MS, IA, CT, OK, OR
2	4,232,659–7,901,691	14	KY, LA, SC, AL, CO, MN, WI, MD, MO, TN, AZ, IN, MA, WA
3	7,901,692–11,570,724	6	VA, NJ, NC, GA, MI, OH
4	11,570,725–15,239,757	2	PA, IL
5	15,239,758–18,908,790	1	FL
6	18,908,791–22,577,823	1	NY
7	22,577,824–26,246,856	1	TX

BinNumber	BinRange	Count	States
8	26,246,857–29,915,889	0	
9	29,915,890–33,584,922	0	
10	33,584,923–37,253,956	1	CA

The least populous state is Wyoming, with 563,626 people, and the most populous is California, with 37,253,956 people. This gives us a range of 37,253,956 – 563,626 = 36,690,330, which we must divide up into equal-size bins—let's say 10 bins. With 10 equal-size bins, each bin will have a width of 3,669,033, so the first bin will span from 563,626 to 4,232,658. By contrast, the top bin, 33,584,923 to 37,253,956, has only one state: California. The two bins immediately below California are empty, until we reach Texas. It is important to include the empty bins; the fact that there are no values in those bins is useful information. It can also be useful to experiment with different bin sizes. If they are too large, important features of the distribution can be obscured. If they are too small, the result is too granular, and the ability to see the bigger picture is lost.

> **NOTE**
>
> Both frequency tables and percentiles summarize the data by creating bins. In general, quartiles and deciles will have the same count in each bin (equal-count bins), but the bin sizes will be different. The frequency table, by contrast, will have different counts in the bins (equal-size bins), and the bin sizes will be the same.

A histogram is a way to visualize a frequency table, with bins on the x-axis and the data count on the y-axis. In Figure 1-3, for example, the bin

centered at 10 million (1e+07) runs from roughly 8 million to 12 million, and there are five states in that bin. To create a histogram corresponding to Table 1-5 in R, use the `hist` function with the `breaks` argument:

```
ggplot(state, aes(x = Population)) +
  geom_histogram(breaks = breaks, fill = "grey", color = "black") +
  labs(y = "Frequency")
```

`pandas` supports histograms for data frames with the `DataFrame.plot.hist` method. Use the keyword argument `bins` to define the number of bins. The various plot methods return an axis object that allows further fine-tuning of the visualization using `Matplotlib`:

```
ax = (state["Population"] / 1_000_000).plot.hist(figsize=(4, 4))
ax.set_xlabel("Population (millions)")
```

The histogram is shown in Figure 1-3. In general, histograms are plotted such that:

- Empty bins are included in the graph.
- Bins are of equal width.
- The number of bins (or, equivalently, bin size) is up to the user.
- Bars are contiguous—no empty space shows between bars, unless there is an empty bin.

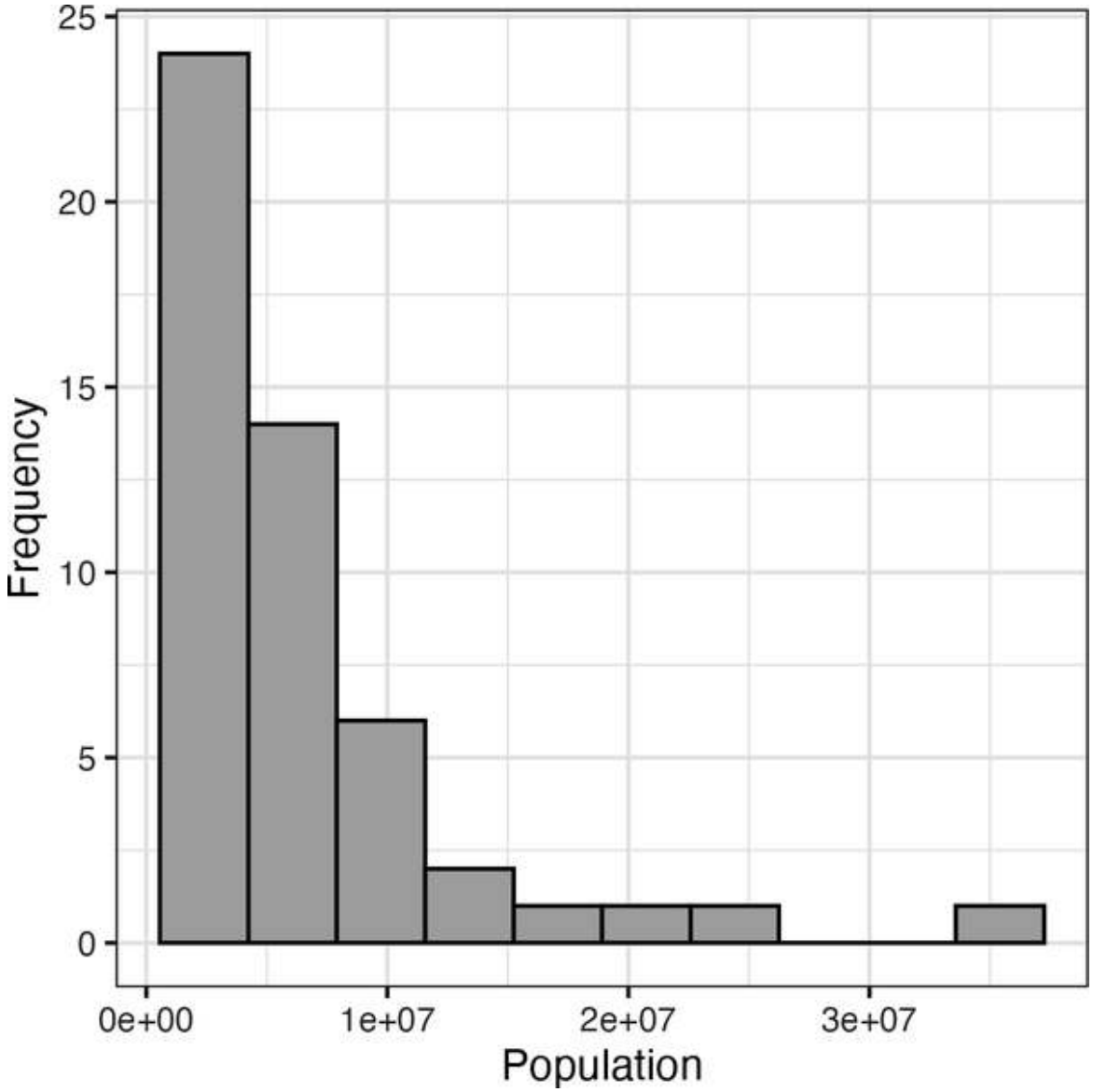


Figure 1-3. Histogram of state populations

STATISTICAL MOMENTS

In statistical theory, location and variability are referred to as the first and second *moments* of a distribution. The third and fourth moments are called *skewness* and *kurtosis*. Skewness refers to whether the data is skewed to larger or smaller values, and kurtosis indicates the propensity of the data to have extreme values. Generally, metrics are not used to measure skewness and kurtosis; instead, these are discovered through visual displays such as Figures 1-2 and 1-3.

Density Plots and Estimates

Related to the histogram is a density plot, which shows the distribution of data values as a continuous line. A density plot can be thought of as a smoothed histogram, although it is typically computed directly from the data through a *kernel density estimate* (see [Duong-2001] for a short tutorial). Figure 1-4 displays a density estimate superposed on a histogram. In R, you can compute a density estimate using the `geom_density` function:

```
ggplot(state, aes(x = Murder.Rate)) +
  geom_histogram(aes(y = after_stat(density)), bins = 12, fill = "grey",
    color = "black") +
  geom_density() + ❶
  labs(x = "Murder Rate (per 100,000)", y = "Density")
```

❶ You can overlay multiple plots on the same graph by adding layers as shown here.

`pandas` provides the `density` method to create a density plot. Use the argument `bw_method` to control the smoothness of the density curve:

```
ax = state["Murder.Rate"].plot.hist(density=True, xlim=[0, 12],
    bins=range(1, 12))
state["Murder.Rate"].plot.density(ax=ax) ❶
ax.set_xlabel("Murder Rate (per 100,000)")
```

❶ Plot functions often take an optional axis (`ax`) argument, which will cause the plot to be added to the same graph.

A key distinction from the histogram plotted in Figure 1-3 is the scale of the y-axis: a density plot corresponds to plotting the histogram as a proportion rather than counts (you specify this in R by transforming the y-values into a density, `aes(y = after_stat(density))`). Note that the total area under the density curve = 1, and instead of counts in bins you can estimate probabilities by calculating areas under the curve between any two points on the x-axis, which correspond to the proportion of the distribution lying

between those two points. In a *cumulative density plot*, the y-axis reflects the cumulative area as you move from left to right along the x-axis.

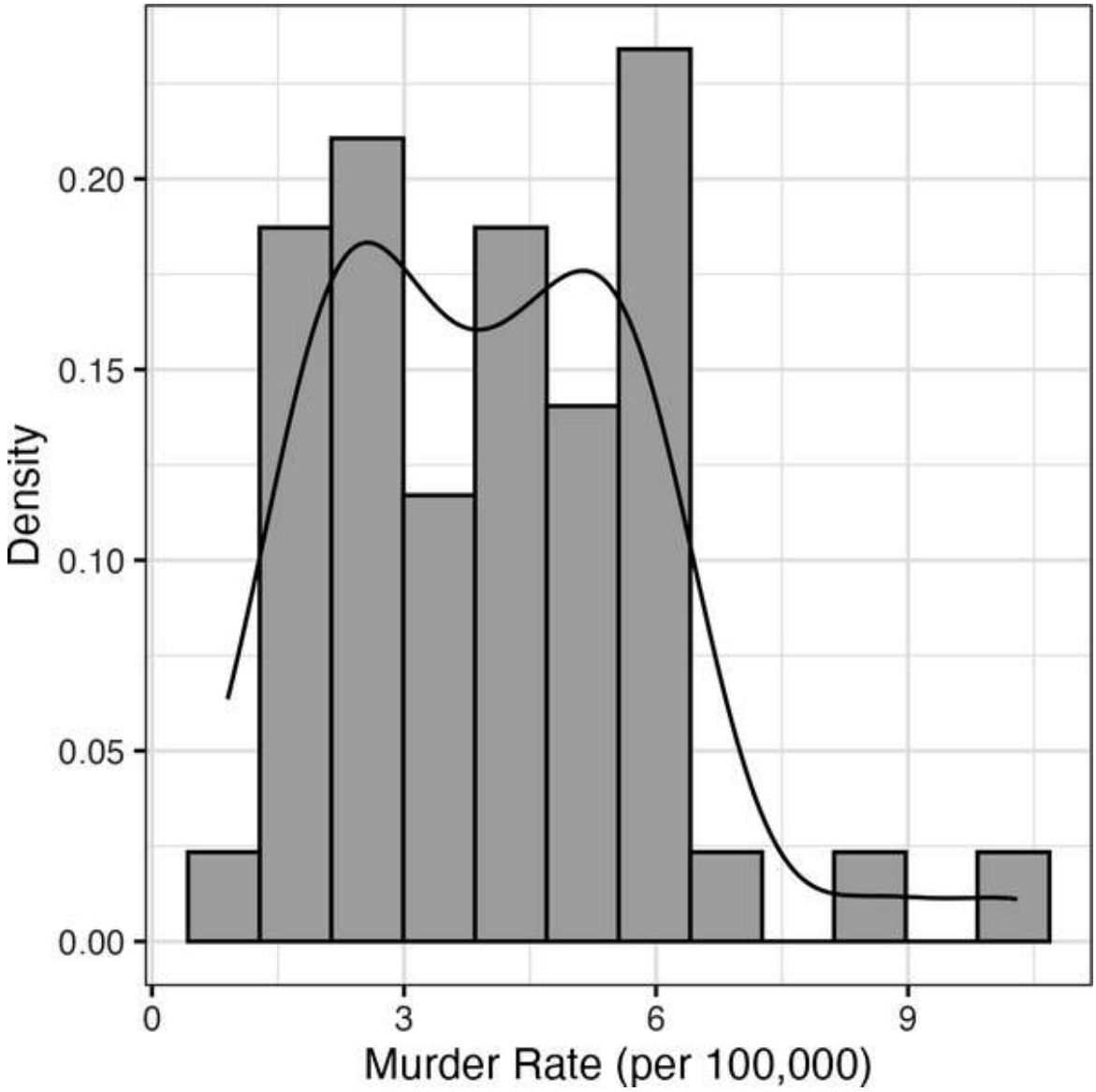


Figure 1-4. Density of state murder rates

DENSITY ESTIMATION

Density estimation is a rich topic with a long history in statistical literature. In fact, over 20 R packages have been published offering functions for density estimation. [Deng-Wickham-2011] give a comprehensive review of R packages, with a particular recommendation for `ASH` or `KernSmooth`. The density estimation methods in `pandas` and `scikit-learn` also offer good implementations. For many data science problems, there's no need to worry about the various types of density estimates; it suffices to use the base functions.

KEY IDEAS

- A frequency histogram plots frequency counts on the y-axis and variable values on the x-axis; it gives a sense of the distribution of the data at a glance.
- A frequency table is a tabular version of the frequency counts found in a histogram.
- A boxplot—with the top and bottom of the box at the 75th and 25th percentiles, respectively—also gives a quick sense of the distribution of the data; it is often used in side-by-side displays to compare distributions.
- A density plot is a smoothed version of a histogram; it requires a function to estimate a plot based on the data (multiple estimates are possible, of course).

Further Reading

- A SUNY Oswego professor provides a step-by-step guide to creating a boxplot.
- Density estimation in R is covered in Henry Deng and Hadley Wickham's paper of the same name.

- R-Bloggers has a useful post on histograms in R, including customization elements, such as binning (breaks).
- R-Bloggers also has a similar post on boxplots in R.
- Matthew Conlen published an interactive presentation that demonstrates the effect of choosing different kernels and bandwidth on kernel density estimates.

Exploring Binary and Categorical Data

For categorical data, simple proportions or percentages tell the story of the data.

KEY TERMS FOR EXPLORING CATEGORICAL DATA

Mode

The most commonly occurring category or value in a data set.

Expected value

When the categories can be associated with a numeric value, this gives an average value based on a category's probability of occurrence.

Bar charts

The frequency or proportion for each category plotted as bars.

Pie charts

The frequency or proportion for each category plotted as wedges in a pie.

Getting a summary of a binary variable or a categorical variable with a few categories is a fairly easy matter: we just figure out the proportion of 1s, or the proportions of the important categories. For example, Table 1-6 shows the percentage of delayed flights by the cause of delay at Dallas/Fort Worth (DFW) Airport in 2010. Delays are categorized as being due to factors under carrier control, air traffic control (ATC) system delays, weather, security, or a late inbound aircraft.

Table 1-6. Percentage of delays by cause at Dallas/Fort Worth Airport

ATC	Carrier	Inbound	Security	Wea
30.40	23.02	42.43	0.12	4.03

Bar charts, seen often in the popular press, are a common visual tool for displaying a single categorical variable. Categories are listed on the x-axis, and frequencies or proportions on the y-axis. Figure 1-5 shows the airport delays per year by cause for DFW, and it is produced with the R function `barplot`:

```
dfw <- read_csv("dfw_airline.csv", show_col_types = FALSE) %>%
  pivot_longer(everything())
ggplot(dfw, aes(x = name, y = value)) +
  geom_col(color = "black", fill = "grey") +
  labs(x = "Cause of delay", y = "Count")
```

`pandas` also supports bar charts for data frames:

```
dfw = pd.read_csv("dfw_airline.csv")
ax = dfw.transpose().plot.bar(figsize=(4, 4), legend=False)
ax.set_xlabel("Cause of delay")
ax.set_ylabel("Count")
```

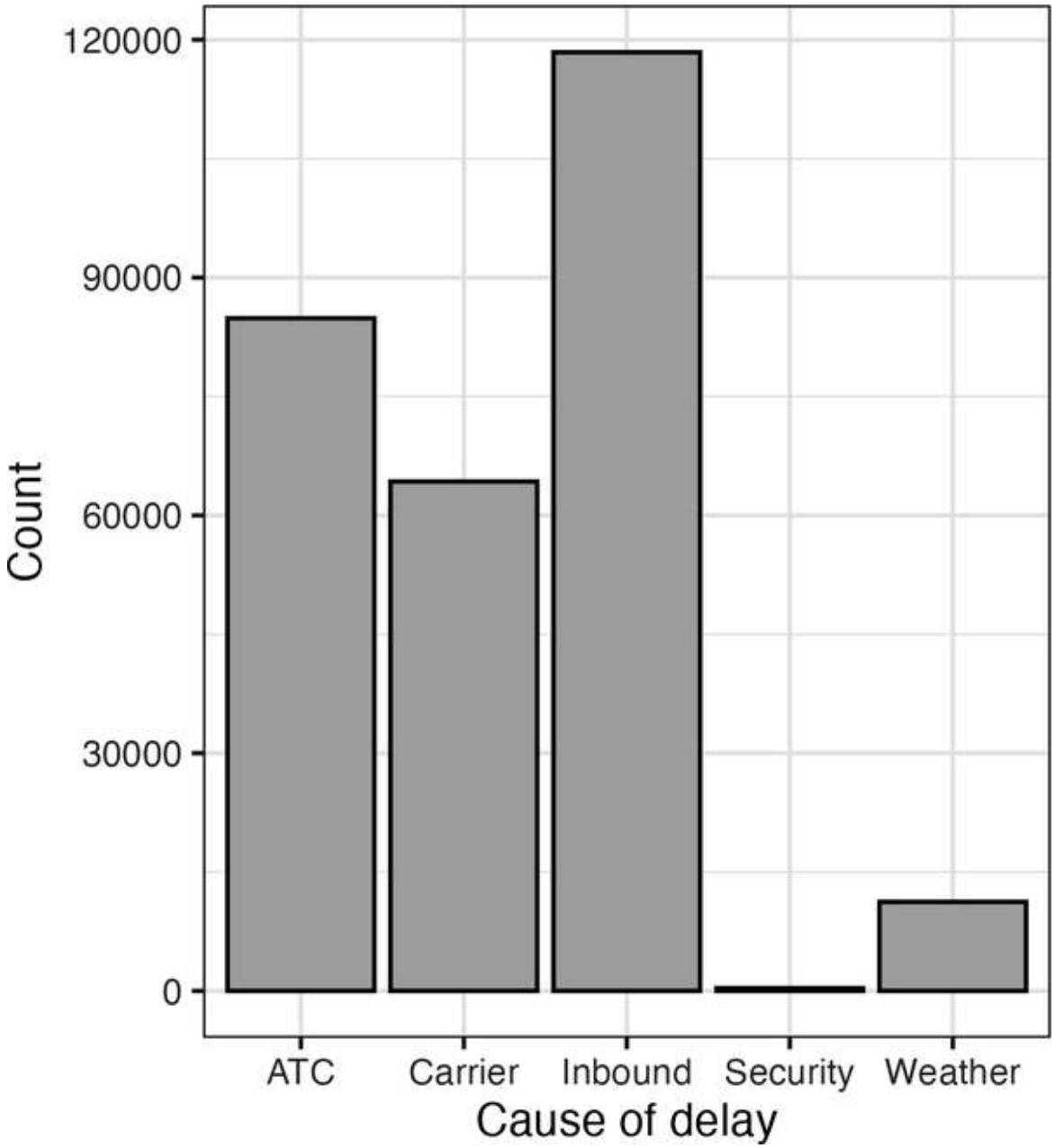


Figure 1-5. Bar chart of airline delays at DFW by cause

Note that a bar chart resembles a histogram; in a bar chart the x-axis represents different categories of a factor variable, while in a histogram the x-axis represents values of a single variable on a numeric scale. In a histogram, the bars are typically shown touching each other, with gaps indicating values that did not occur in the data. In a bar chart, the bars are shown separate from one another.

Pie charts are an alternative to bar charts, although statisticians and data visualization experts generally eschew pie charts as less visually informative (see [Few-2007]).

NUMERICAL DATA AS CATEGORICAL DATA

In "Frequency Tables and Histograms", we looked at frequency tables based on binning the data. This implicitly converts the numeric data to an ordered factor. In this sense, histograms and bar charts are similar, except that the categories on the x-axis in the bar chart are not ordered. Converting numeric data to categorical data is an important and widely used step in data analysis since it reduces the complexity (and size) of the data. This aids in the discovery of relationships between features, particularly at the initial stages of an analysis.

Mode

The mode is the value—or values in case of a tie—that appears most often in the data. For example, the mode of the cause of delay at DFW airport is "Inbound." As another example, in most parts of the United States, the mode for religious preference would be Christian. The mode is a simple summary statistic for categorical data, and it is generally not used for numeric data.

Expected Value

A special type of categorical data is data in which the categories represent or can be mapped to discrete values on the same scale. A marketer for a new cloud technology, for example, offers two levels of service, one priced at $300/month and another at $50/month. The marketer offers free webinars to generate leads, and the firm figures that 5% of the attendees will sign up for the $300 service, 15% will sign up for the $50 service, and 80% will not sign up for anything. This data can be summed up, for financial purposes, in a single "expected value," which is a form of weighted mean, in which the weights are probabilities.

The expected value is calculated as follows:

1. Multiply each outcome by its probability of occurrence.
2. Sum these values.

In the cloud service example, the expected value of a webinar attendee is thus $22.50 per month, calculated as follows:

$$EV = (0.05)(300) + (0.15)(50) + (0.80)(0) = 22.5$$

The expected value is really a form of weighted mean: it adds the ideas of future expectations and probability weights, often based on subjective judgment. Expected value is a fundamental concept in business valuation and capital budgeting—for example, the expected value of five years of profits from a new acquisition, or the expected cost savings from new patient management software at a clinic.

Probability

We referred above to the *probability* of a value occurring. Most people have an intuitive understanding of probability, encountering the concept frequently in weather forecasts (the chance of rain) or sports analysis (the probability of winning). Sports and games are more often expressed as odds, which are readily convertible to probabilities (if the odds that a team will win are 2 to 1, its probability of winning is 2 / (2 + 1) = 2 / 3). Surprisingly, though, the concept of probability can be the source of deep philosophical discussion when it comes to defining it. Fortunately, we do not need a formal mathematical or philosophical definition here. For our purposes, the probability that an event will happen is the proportion of times it will occur if the situation could be repeated over and over, countless times. Most often this is an imaginary construction, but it is an adequate operational understanding of probability.

KEY IDEAS

- Categorical data is typically summed up in proportions and can be visualized in a bar chart.
- Categories might represent distinct things (apples and oranges, male and female), levels of a factor variable (low, medium, and high), or numeric data that has been binned.
- Expected value is the sum of values times their probability of occurrence, often used to sum up factor variable levels.

Further Reading

No statistics course is complete without a lesson on misleading graphs, which often involves bar charts and pie charts.

Correlation

Exploratory data analysis in many modeling projects (whether in data science or in research) involves examining correlation among predictors, and between predictors and a target variable. Variables X and Y (each with measured data) are said to be positively correlated if high values of X go with high values of Y, and low values of X go with low values of Y. If high values of X go with low values of Y, and vice versa, the variables are negatively correlated.

KEY TERMS FOR CORRELATION

Correlation coefficient

A metric that measures the extent to which numeric variables are associated with one another (ranges from –1 to +1)

Correlation matrix

A table where the variables are shown on both rows and columns, and the cell values are the correlations between the variables

Scatterplot

A plot in which the x-axis is the value of one variable, and the y-axis the value of another

Consider these two variables, perfectly correlated in the sense that each goes from low to high:

v1: {1, 2, 3}

v2: {4, 5, 6}

The vector sum of products is $1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$. Now try shuffling one of them and recalculating—the vector sum of products will never be higher than 32. So this sum of products could be used as a metric; that is, the observed sum of 32 could be compared to lots of random shufflings (in fact, this idea relates to a resampling-based estimate; see "Permutation Test"). Values produced by this metric, though, are not that meaningful, except by reference to the resampling distribution.

More useful is a standardized variant: the *correlation coefficient*, which gives an estimate of the correlation between two variables that always lies on the same scale. To compute *Pearson's correlation coefficient*, we

multiply deviations from the mean for variable 1 times those for variable 2, and divide by the product of the standard deviations:

$$r = \frac{\sum_{i=1}^{n}(x_i - x)(y_i - y)}{(n-1)s_x s_y}$$

Note that we divide by *n* – 1 instead of *n*; see "Degrees of Freedom, and *n* or *n* – 1?" for more details. The correlation coefficient always lies between +1 (perfect positive correlation) and –1 (perfect negative correlation); 0 indicates no correlation.

Variables can have an association that is not linear, in which case the correlation coefficient may not be a useful metric. The relationship between tax rates and revenue raised is an example: as tax rates increase from zero, the revenue raised also increases. However, once tax rates reach a high level and approach 100%, tax avoidance increases and tax revenue actually declines.

Table 1-7, called a *correlation matrix*, shows the correlation between the daily returns for telecommunication stocks from July 2012 through June 2015. From the table, you can see that Verizon (VZ) and AT&T (T) have the highest correlation. Level 3 (LVLT), which is an infrastructure company, has the lowest correlation with the others. Note the diagonal of 1s (the correlation of a stock with itself is 1) and the redundancy of the information above and below the diagonal.

Table 1-7. Correlation between telecommunication stock returns

	T	CTL	FTR	VZ
T	1.000	0.475	0.328	0.678
CTL	0.475	1.000	0.420	0.417
FTR	0.328	0.420	1.000	0.287
VZ	0.678	0.417	0.287	1.000
LVLT	0.279	0.287	0.260	0.242

A table of correlations like Table 1-7 is commonly plotted to visually display the relationship between multiple variables. Figure 1-6 shows the correlation between the daily returns for major exchange-traded funds (ETFs). In R, we can easily create this using the package `corrplot` using the correlation matrix calculated with the `cor` function:

```
library(corrplot)

sp500_px <- read_csv("sp500_data.csv.gz", show_col_types = FALSE)
sp500_sym <- read_csv("sp500_sectors.csv", show_col_types = FALSE)
etf_symbols <- sp500_sym %>%
  filter(sector == "etf") %>%
  pull(symbol)  ❶
etfs <- sp500_px %>%
  filter(Date > "2012-07-01") %>%
  select(all_of(etf_symbols))  ❷
corrplot(cor(etfs), method = "ellipse")
```

❶ We first `filter` the symbols data frame to get the ETF symbols and then use `pull` to extract the `symbol` column as a vector.

❷ Creates a subset of the table with only the ETF columns and rows after July 1, 2012.

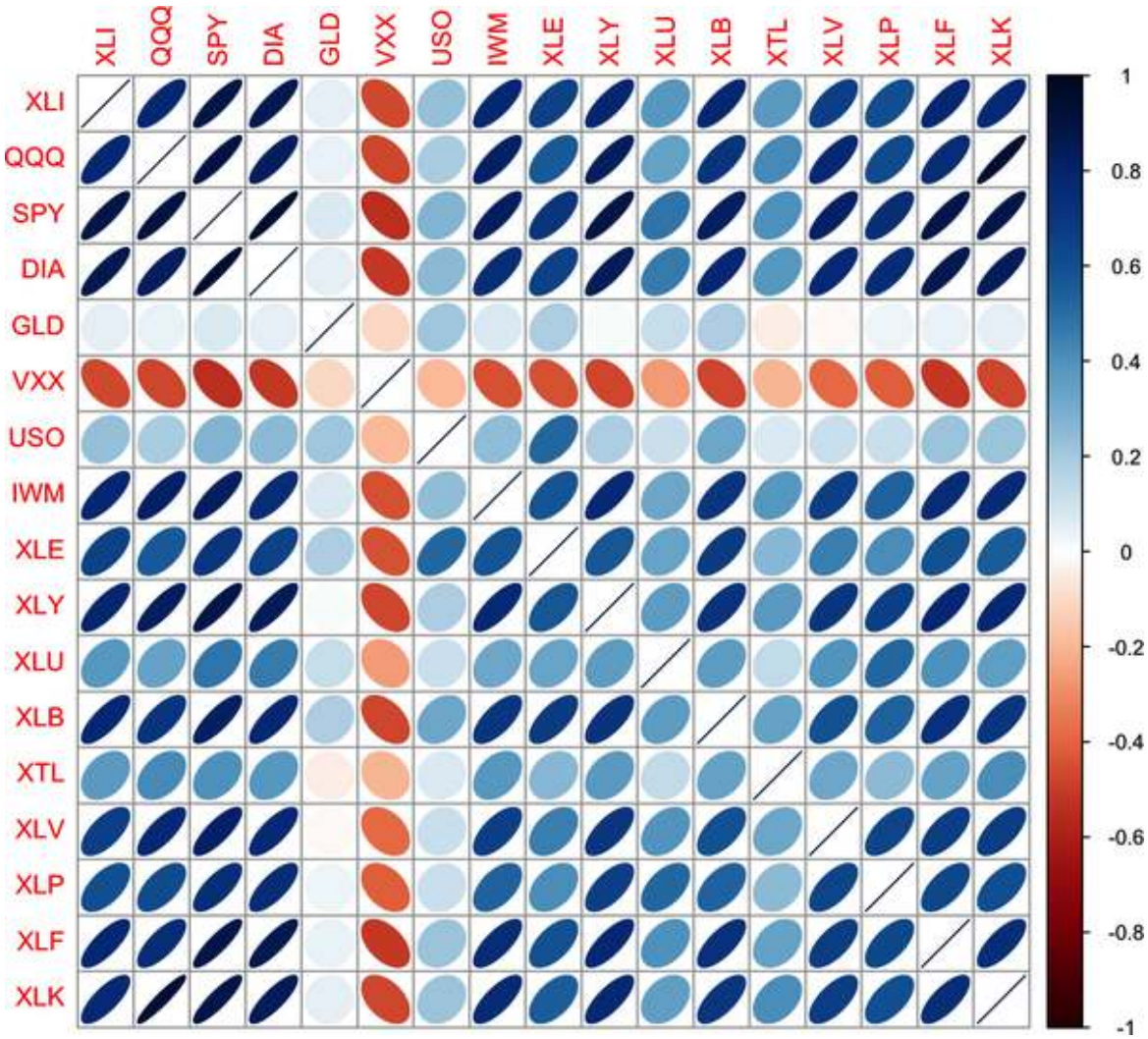


Figure 1-6. Correlation between ETF returns

It is possible to create the same graph in Python, but there is no implementation available in common packages. However, most support the visualization of correlation matrices using heatmaps. The following code demonstrates this using the `seaborn.heatmap` package. In the accompanying source code repository, we include Python code to generate the more comprehensive visualization:

```
import seaborn as sns
```

```
sp500_px = pd.read_csv("sp500_data.csv.gz", index_col=0)
sp500_sym = pd.read_csv("sp500_sectors.csv")
etf_symbols = sp500_sym[sp500_sym["sector"] == "etf"]["symbol"]
etfs = sp500_px.loc[sp500_px.index > "2012-07-01", etf_symbols]
sns.heatmap(etfs.corr(), vmin=-1, vmax=1,
  cmap=sns.diverging_palette(20, 220, as_cmap=True))
```

The ETFs for the S&P 500 (SPY) and the Dow Jones Index (DIA) have a high correlation. Similarly, the QQQ and the XLK, composed mostly of technology companies, are positively correlated. Defensive ETFs, such as those tracking gold prices (GLD), oil prices (USO), or market volatility (VXX), tend to be weakly or negatively correlated with the other ETFs. The orientation of the ellipse indicates whether two variables are positively correlated (ellipse is pointed to the top right) or negatively correlated (ellipse is pointed to the top left). The shading and width of the ellipse indicate the strength of the association: thinner and darker ellipses correspond to stronger relationships.

Like the mean and standard deviation, the correlation coefficient is sensitive to outliers in the data. Software packages offer robust alternatives to the classical correlation coefficient. For example, the R package `robust` uses the function `covRob` to compute a robust estimate of correlation. The methods in the `scikit-learn` module *sklearn.covariance* implement a variety of approaches.

OTHER CORRELATION ESTIMATES

Statisticians long ago proposed other types of correlation coefficients, such as *Spearman's rho* or *Kendall's tau*. These are correlation coefficients based on the rank of the data. Since they work with ranks rather than values, these estimates are robust to outliers and can handle certain types of nonlinearities. However, data scientists can generally stick to Pearson's correlation coefficient, and its robust alternatives, for exploratory analysis. The appeal of rank-based estimates is mostly for smaller data sets and specific hypothesis tests.

Scatterplots

The standard way to visualize the relationship between two measured data variables is with a scatterplot. The x-axis represents one variable and the y-axis another, and each point on the graph is a record. See Figure 1-7 for a plot of the correlation between the daily returns for AT&T and Verizon. This is produced in R with the command:

```
ggplot(sp500_px %>% filter(Date > "2012-07-01"), aes(x = T, y = VZ)) +
  geom_point(shape = 1) +
  labs(x = "ATT (T)", y = "Verizon (VZ)")
```

The same graph can be generated in Python using the `pandas` scatter method:

```
# Filter data for dates July 2012 through June 2015
telecom = sp500_px.loc[sp500_px.index >= "2012-07-01", :]

ax = telecom.plot.scatter(x="T", y="VZ", figsize=(4, 4), marker="$\u25EF$")
❶
ax.set_xlabel("ATT (T)")
ax.set_ylabel("Verizon (VZ)")
ax.axhline(0, color="grey", lw=1)
ax.axvline(0, color="grey", lw=1)
```

❶ The marker `$\u25EF$` will show the points as open circles. This is one way of reducing the impact of overlapping points.

The returns have a positive relationship: while they cluster around zero, on most days, the stocks go up or go down in tandem (upper-right and lower-left quadrants). There are fewer days where one stock goes down significantly while the other stock goes up, or vice versa (lower-right and upper-left quadrants).

While the plot in Figure 1-7 displays only 754 data points, it's already obvious how difficult it is to identify details in the middle of the plot. We will see later how adding transparency to the points, or using hexagonal binning and density plots, can help to find additional structure in the data.

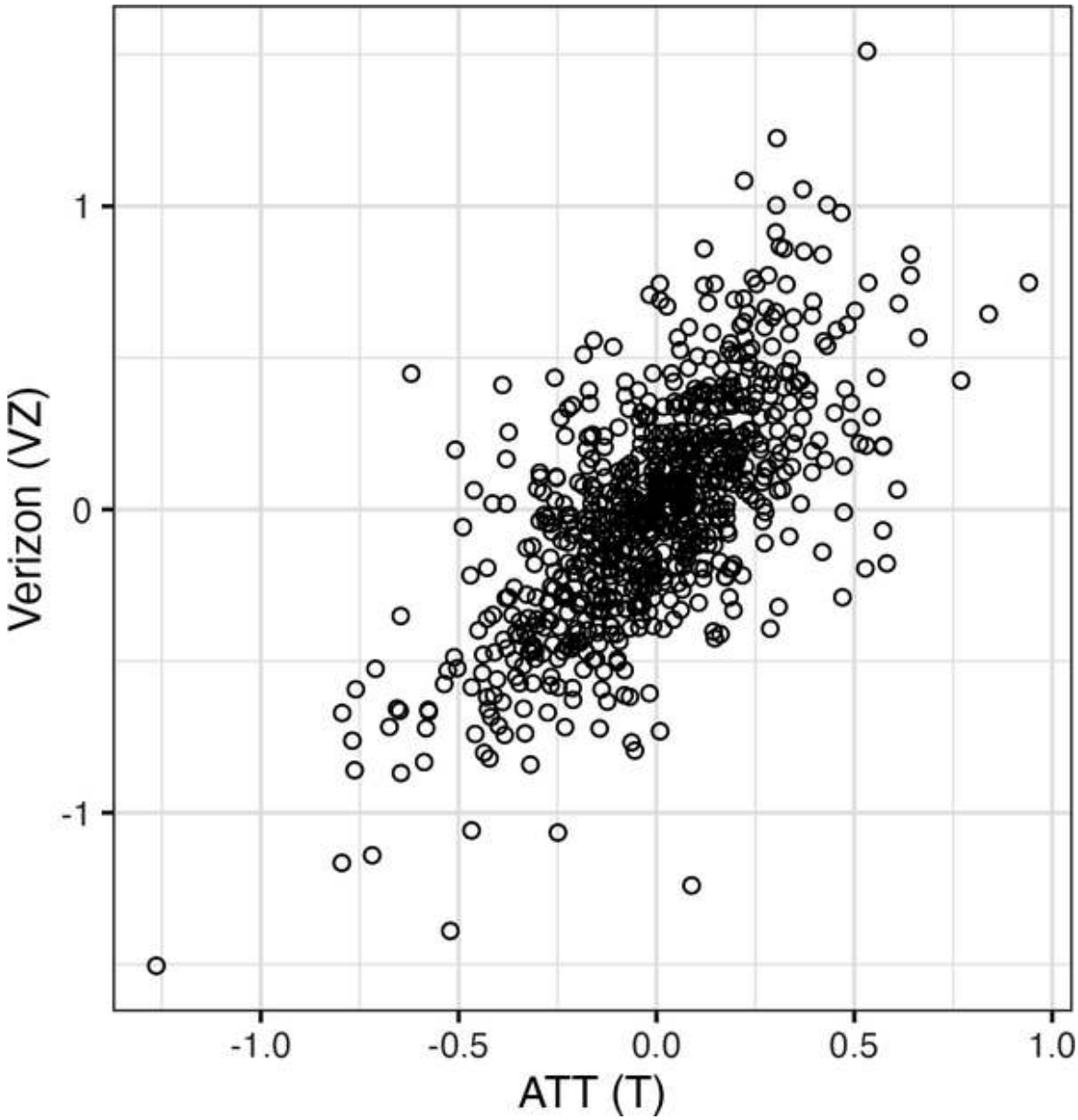


Figure 1-7. Scatterplot of correlation between returns for AT&T and Verizon

KEY IDEAS

- The correlation coefficient measures the extent to which two paired variables (e.g., height and weight for individuals) are associated with one another.
- When high values of v1 go with high values of v2, v1 and v2 are positively associated.
- When high values of v1 go with low values of v2, v1 and v2 are negatively associated.
- The correlation coefficient is a standardized metric, so it always ranges from –1 (perfect negative correlation) to +1 (perfect positive correlation).
- A correlation coefficient of zero indicates no correlation, but be aware that random arrangements of data will produce both positive and negative values for the correlation coefficient just by chance.

Further Reading

Statistics, 4th ed., by David Freedman, Robert Pisani, and Roger Purves (W. W. Norton & Company, 2007) has an excellent discussion of correlation.

Exploring Two or More Variables

Familiar estimators like mean and variance look at variables one at a time (*univariate analysis*). Correlation analysis (see "Correlation") is an important method that compares two variables (*bivariate analysis*). In this section we look at additional estimates and plots, and at more than two variables (*multivariate analysis*).

KEY TERMS FOR EXPLORING TWO OR MORE VARIABLES

Contingency table

A tally of counts between two or more categorical variables

Hexagonal binning

A plot of two numeric variables with the records binned into hexagons

Contour plot

A plot showing the density of two numeric variables like a topographical map

Violin plot

Similar to a boxplot but showing the density estimate

Facet plot

A set of plots arranged in a grid, each showing the relationship between two variables for a subset of the data defined by one or more additional variables

Synonyms

lattice plots, small multiples, trellis graphs

Like univariate analysis, bivariate analysis involves both computing summary statistics and producing visual displays. The appropriate type of bivariate or multivariate analysis depends on the nature of the data: numeric versus categorical.

Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)

Scatterplots are fine when there is a relatively small number of data values. The plot of stock returns in Figure 1-7 involves only about 750 points. For data sets with hundreds of thousands or millions of records, a scatterplot will be too dense, so we need a different way to visualize the relationship. To illustrate, consider the data set `kc_tax`, which contains the tax-assessed values for residential properties in King County, Washington. In order to focus on the main part of the data, we strip out very expensive and very small or large residences using the `filter` function:

```
kc_tax <- read_csv("kc_tax.csv.gz", show_col_types = FALSE)
kc_tax0 <- kc_tax %>%
  filter(TaxAssessedValue < 750000,
    SqFtTotLiving > 100,
    SqFtTotLiving < 3500)
nrow(kc_tax0)
---
432693
```

In `pandas`, we filter the data set as follows:

```
kc_tax = pd.read_csv("kc_tax.csv.gz")
kc_tax0 = kc_tax.loc[(kc_tax.TaxAssessedValue < 750000) &
                     (kc_tax.SqFtTotLiving > 100) &
                     (kc_tax.SqFtTotLiving < 3500), :]
kc_tax0.shape
---
(432693, 3)
```

Figure 1-8 is a *hexagonal binning* plot of the relationship between the finished square feet and the tax-assessed value for homes in King County. Rather than plotting points, which would appear as a monolithic dark cloud, we grouped the records into hexagonal bins and plotted the hexagons with a color indicating the number of records in that bin. In this chart, the positive relationship between square feet and tax-assessed value is clear. An interesting feature is the hint of additional bands above the main (darkest)

band at the bottom, indicating homes that have the same square footage as those in the main band but a higher tax-assessed value.

Figure 1-8 was generated by the powerful R package `ggplot2`, developed by Hadley Wickham [Wickham-2009]. `ggplot2` is one of several new software libraries for advanced exploratory visual analysis of data; see "Visualizing Multiple Variables":

```
ggplot(kc_tax0, (aes(x = SqFtTotLiving, y = TaxAssessedValue))) +
  geom_hex(color = "white") +
  scale_fill_gradient(low = "white", high = "black") +
  labs(x = "Finished Square Feet", y = "Tax-Assessed Value")
```

In Python, hexagonal binning plots are readily available using the `pandas` data frame method `hexbin`:

```
ax = kc_tax0.plot.hexbin(x="SqFtTotLiving", y="TaxAssessedValue",
                         gridsize=30, sharex=False, figsize=(5, 4))
ax.set_xlabel("Finished Square Feet")
ax.set_ylabel("Tax-Assessed Value")
```

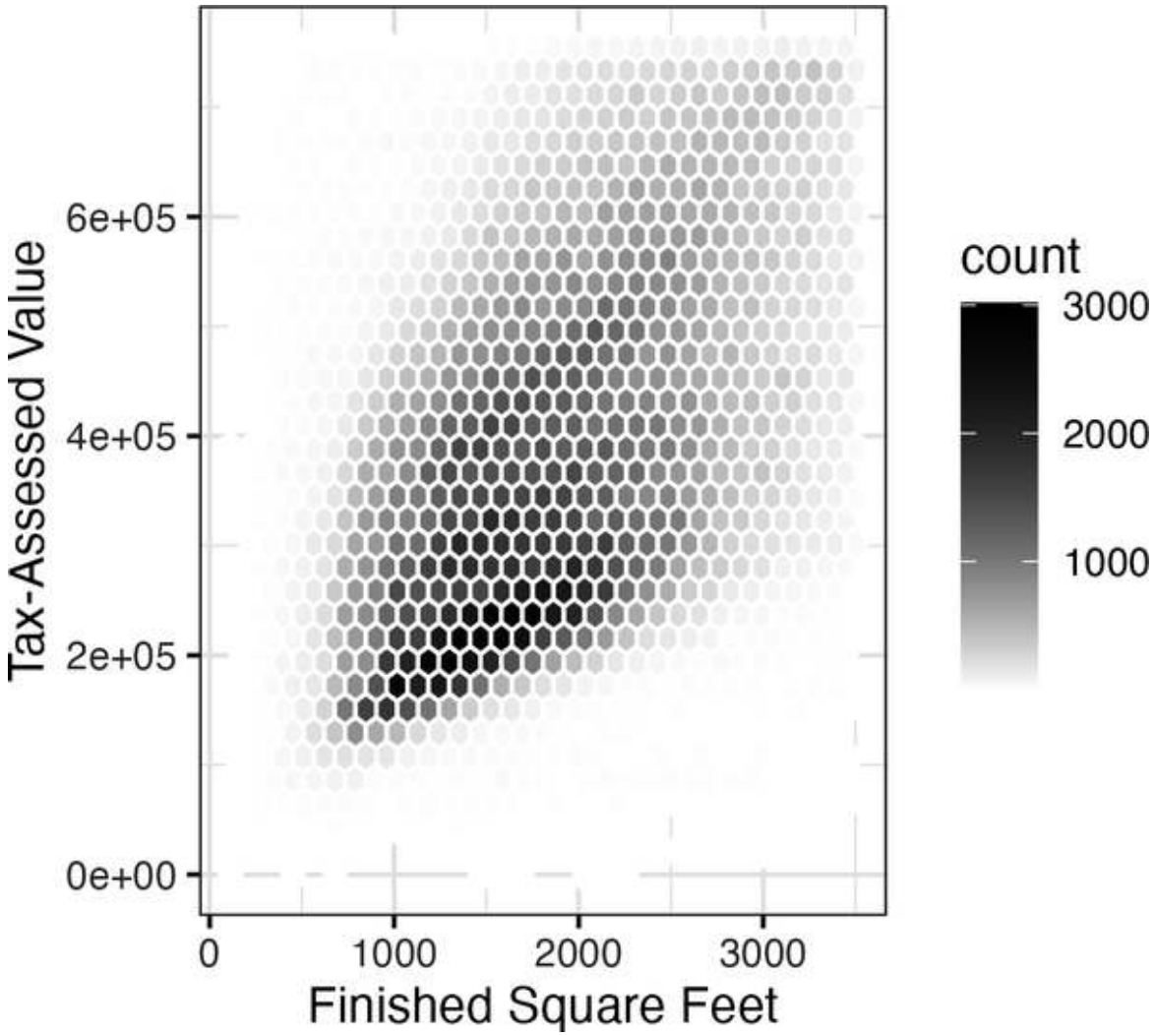


Figure 1-8. Hexagonal binning for tax-assessed value versus finished square feet

Figure 1-9 uses contours overlaid onto a scatterplot to visualize the relationship between two numeric variables. The contours are essentially a topographical map of two variables; each contour band represents a specific density of points, increasing as one nears a "peak." This plot shows a similar story as Figure 1-8: there is a secondary peak "north" of the main peak. This chart was also created using `ggplot2` with the built-in `geom_density2d` function:

```
ggplot(kc_tax0, aes(SqFtTotLiving, TaxAssessedValue)) +
  geom_point(shape = 1, alpha = 0.02) +
  geom_density2d(color = "white") +
  labs(x = "Finished Square Feet", y = "Tax-Assessed Value")
```

The `seaborn kdeplot` function in Python creates a contour plot:

```
ax = sns.kdeplot(data=kc_tax0.sample(10000), x="SqFtTotLiving",
    y="TaxAssessedValue")
ax.set_xlabel("Finished Square Feet")
ax.set_ylabel("Tax-Assessed Value")
```

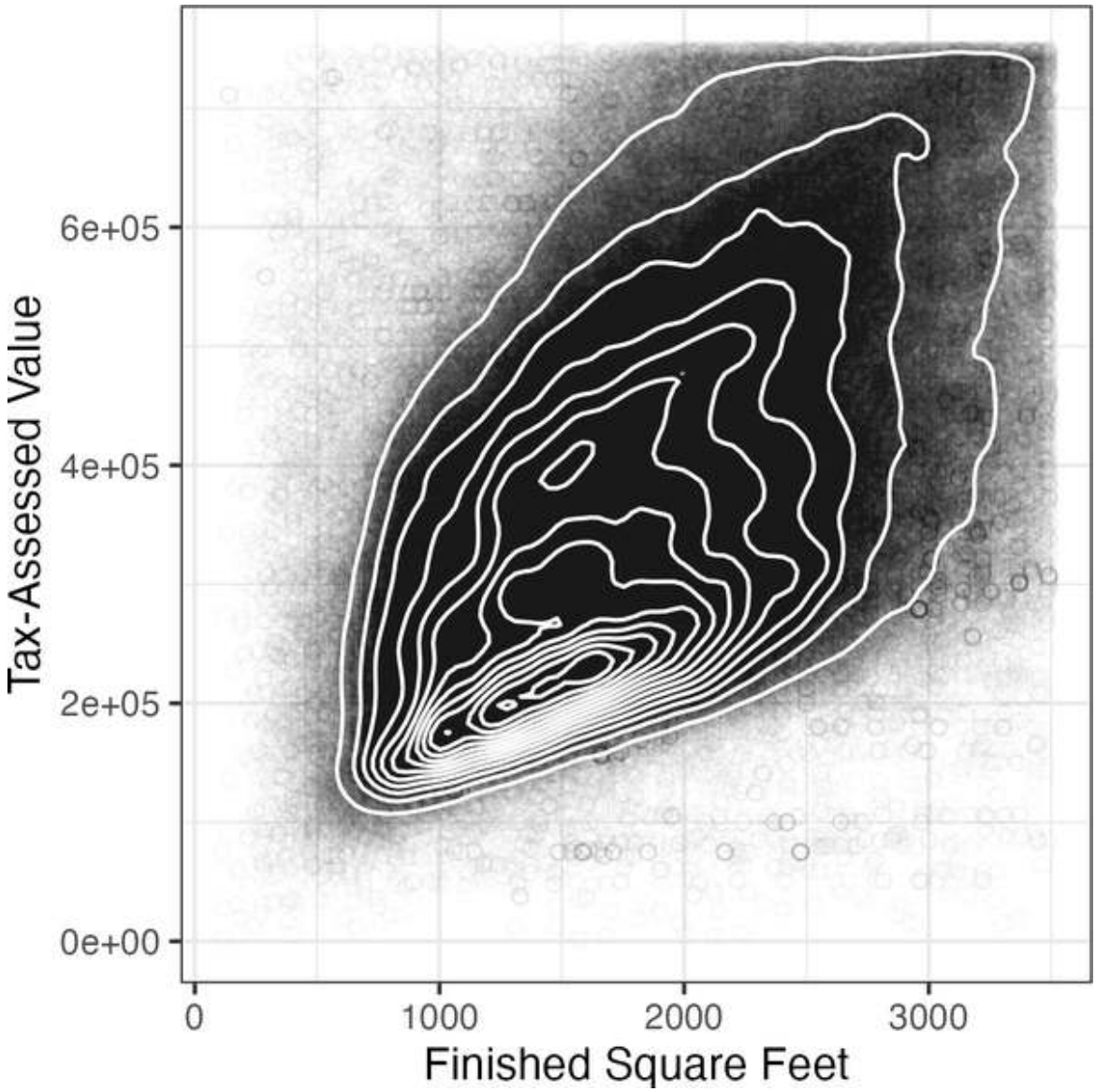


Figure 1-9. Contour plot for tax-assessed value versus finished square feet

Other types of charts are used to show the relationship between two numeric variables, including *heat maps*. Heat maps, hexagonal binning, and

contour plots all give a visual representation of a two-dimensional density. In this way, they are natural analogs to histograms and density plots.

Two Categorical Variables

A useful way to summarize two categorical variables is a contingency table—a table of counts by category. Table 1-8 shows the contingency table between the grade of a personal loan and the outcome of that loan. This is taken from data provided by LendingClub, a leader in the peer-to-peer lending business. The grade goes from A (high) to G (low). The outcome is either fully paid, current, late, or charged off (the balance of the loan is not expected to be collected). This table shows the count and row percentages. High-grade loans have a very low late/charge-off percentage as compared with lower-grade loans.

Table 1-8. Contingency table of loan grade and status

Grade	Charged off	Current	Fully paid	Late
A	1,562	50,051	20,408	469
	0.022	0.690	0.282	0.006
B	5,302	93,852	31,160	2,056
	0.040	0.709	0.235	0.016
C	6,023	88,928	23,147	2,777
	0.050	0.736	0.191	0.023
D	5,007	53,281	13,681	2,308
	0.067	0.717	0.184	0.031
E	2,842	24,639	5,949	1,374
	0.082	0.708	0.171	0.039
F	1,526	8,444	2,328	606
	0.118	0.654	0.180	0.047
G	409	1,990	643	199
Total	22,671	321,185	97,316	9,789

Grade	Charged off	Current	Fully paid	Late
	0.126	0.614	0.198	0.061
Total	22,671	321,185	97,316	9,789

Contingency tables can look only at counts, or they can also include column and total percentages. Pivot tables in Excel are perhaps the most common tool used to create contingency tables. In R, the `CrossTable` function in the `gmodels` package produces contingency tables, and the following code was used to create Table 1-8:

```
library(gmodels)
lc_loans <- read_csv("lc_loans.csv", show_col_types = FALSE)
x_tab <- CrossTable(lc_loans$grade, lc_loans$status,
  prop.c = FALSE, prop.chisq = FALSE, prop.t = FALSE)
```

The `pivot_table` method creates the pivot table in Python. The `aggfunc` argument allows us to get the counts. Calculating the percentages is a bit more involved:

```
lc_loans = pd.read_csv("lc_loans.csv")

crosstab = lc_loans.pivot_table(index="grade", columns="status",
                                aggfunc=len, margins=True) ❶
print(crosstab)

df = crosstab.loc["A":"G", :].copy() ❷
perc_crosstab = df.loc[:, "Charged Off":"Late"].div(df["All"], axis=0) ❸
perc_crosstab["All"] = df["All"] / sum(df["All"]) ❹
print(perc_crosstab)
```

❶ The `margins` keyword argument will add the column and row sums.

❷ We create a copy of the pivot table, ignoring the column sums.

❸ We divide the rows with the row sum.

❹ We divide the `"All"` column by its sum.

Categorical and Numeric Data

Boxplots (see "Percentiles and Boxplots") are a simple way to visually compare the distributions of a numeric variable grouped according to a categorical variable. For example, we might want to compare how the percentage of flight delays varies across airlines. Figure 1-10 shows the percentage of flights in a month that were delayed where the delay was within the carrier's control:

```
airline_stats <- read_csv("airline_stats.csv", show_col_types = FALSE) %>%
  mutate(
    airline = factor(airline,
      levels = c("Alaska", "American", "JetBlue", "Delta", "United",
        "Southwest"))
  ) %>%
  drop_na()

ggplot(airline_stats, aes(x = airline, y = pct_carrier_delay)) +
  geom_boxplot() +
  coord_cartesian(ylim = c(0, 50)) +
  labs(x = "Airline", y = "Daily % of Delayed Flights")
```

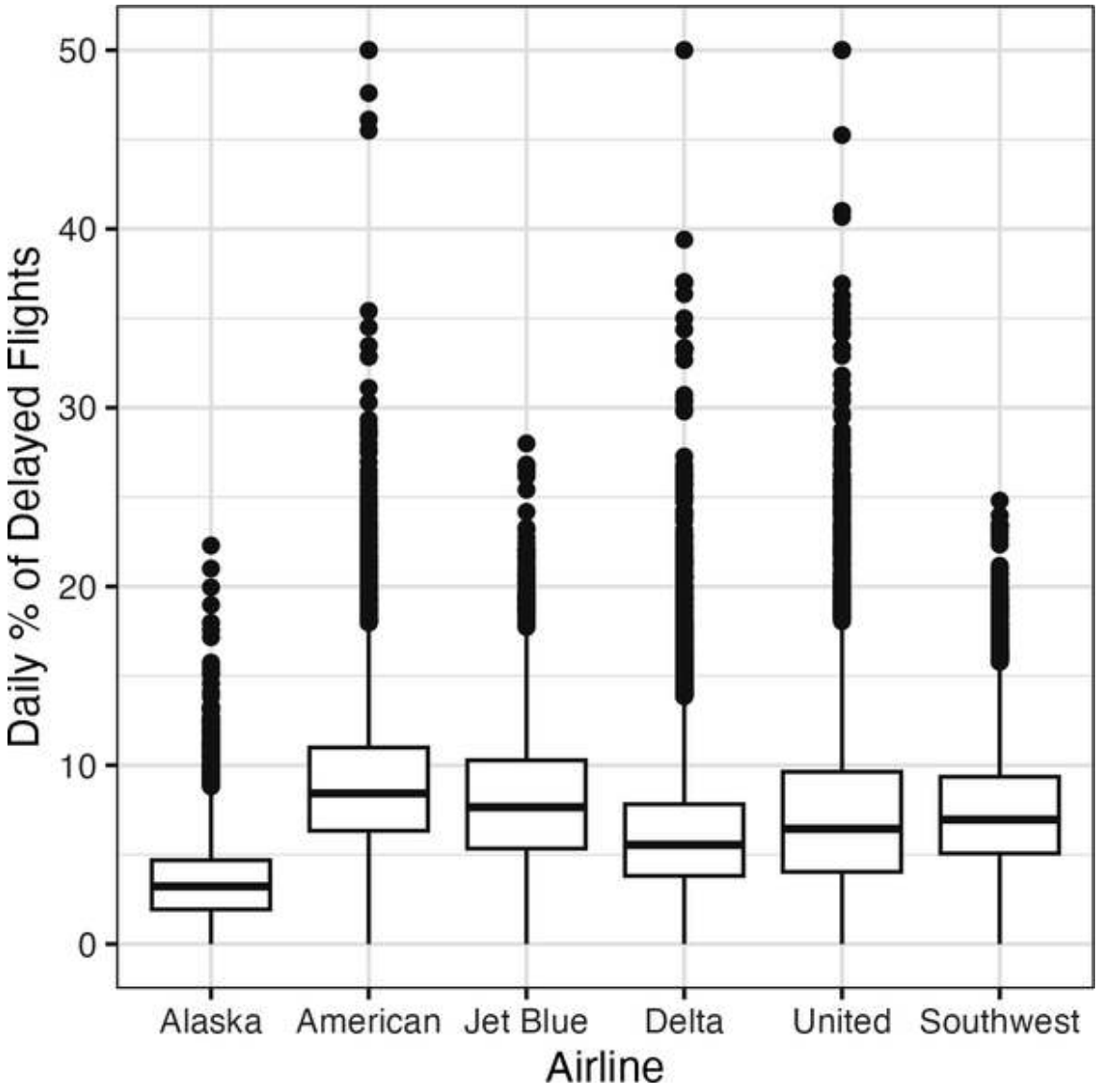


Figure 1-10. Boxplot of percent of airline delays by carrier

The `pandas boxplot` method takes the `by` argument that splits the data set into groups and creates the individual boxplots:

```
import matplotlib.pyplot as plt

airline_stats = pd.read_csv("airline_stats.csv")
ax = airline_stats.boxplot(by="airline", column="pct_carrier_delay")
ax.set_xlabel("")
ax.set_ylabel("Daily % of Delayed Flights")
plt.suptitle("")
```

Alaska stands out as having the fewest delays, while American has the most delays: the lower quartile for American is higher than the upper quartile for Alaska.

A *violin plot*, introduced by [Hintze-Nelson-1998], is an enhancement to the boxplot and plots the density estimate with the density on the y-axis. The density is mirrored and flipped over, and the resulting shape is filled in, creating an image resembling a violin. The advantage of a violin plot is that it can show nuances in the distribution that aren't perceptible in a boxplot. On the other hand, the boxplot more clearly shows the outliers in the data. In `ggplot2`, the function `geom_violin` can be used to create a violin plot as follows:

```
ggplot(data = airline_stats, aes(airline, pct_carrier_delay)) +
  geom_violin() +
  coord_cartesian(ylim = c(0, 50)) +
  labs(x = "", y = "Daily % of Delayed Flights")
```

Violin plots are available with the `violinplot` method of the `seaborn` package:

```
ax = sns.violinplot(data=airline_stats, x="airline", y="pct_carrier_delay",
                    inner="quartile", color="white")
ax.set_xlabel("")
ax.set_ylabel("Daily % of Delayed Flights")
```

The corresponding plot is shown in Figure 1-11. The violin plot shows a concentration in the distribution near zero for Alaska and, to a lesser extent, Delta. This phenomenon is not as obvious in the boxplot. You can combine a violin plot with a boxplot by adding `geom_boxplot` to the plot (although this works best when colors are used).

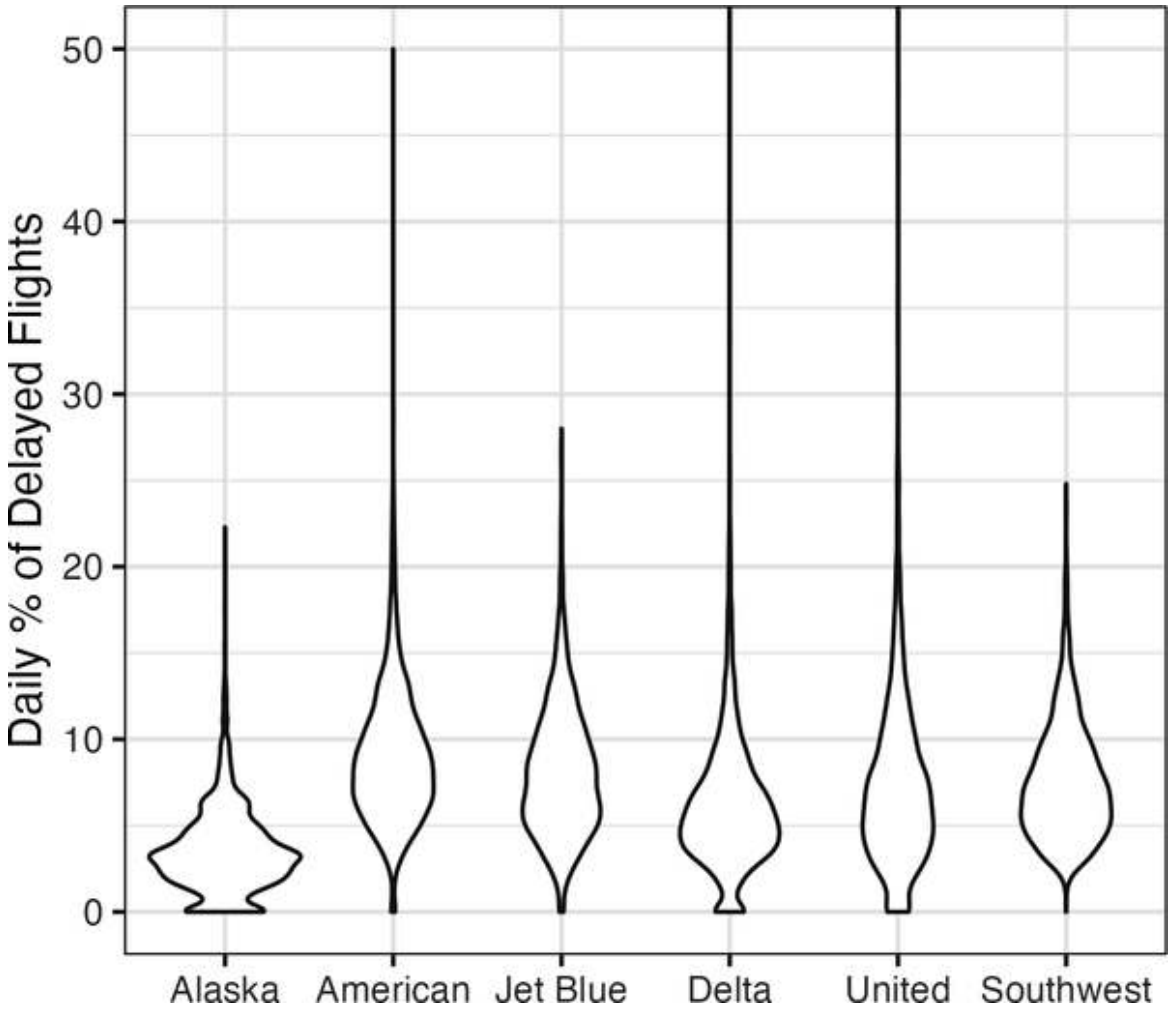


Figure 1-11. Violin plot of percent of airline delays by carrier

Visualizing Multiple Variables

The types of charts used to compare two variables—scatterplots, hexagonal binning, and boxplots—are readily extended to more variables through the notion of *conditioning*. As an example, look back at Figure 1-8, which showed the relationship between homes' finished square feet and their tax-assessed values. We observed that there appears to be a cluster of homes that have higher tax-assessed value per square foot. Diving deeper, Figure 1-12 accounts for the effect of location by plotting the data for a set of zip codes. Now the picture is much clearer: tax-assessed value is much higher in some zip codes (98105, 98126) than in others (98108, 98188). This disparity gives rise to the clusters observed in Figure 1-8.

We created Figure 1-12 using `ggplot2` and the idea of *facets*, or a conditioning variable (in this case, zip code):

```
kc_subset <- kc_tax0 %>%
  filter(ZipCode %in% c(98188, 98105, 98108, 98126))
ggplot(kc_subset, aes(x = SqFtTotLiving, y = TaxAssessedValue)) +
  geom_hex(color = "white") +
  scale_fill_gradient(low = "white", high = "blue") +
  labs(x = "Finished Square Feet", y = "Tax-Assessed Value") +
  facet_wrap("ZipCode") ❶
```

❶ Use the `ggplot` functions `facet_wrap` and `facet_grid` to specify the conditioning variable. `facet_grid` is used with two or more conditioning variables.

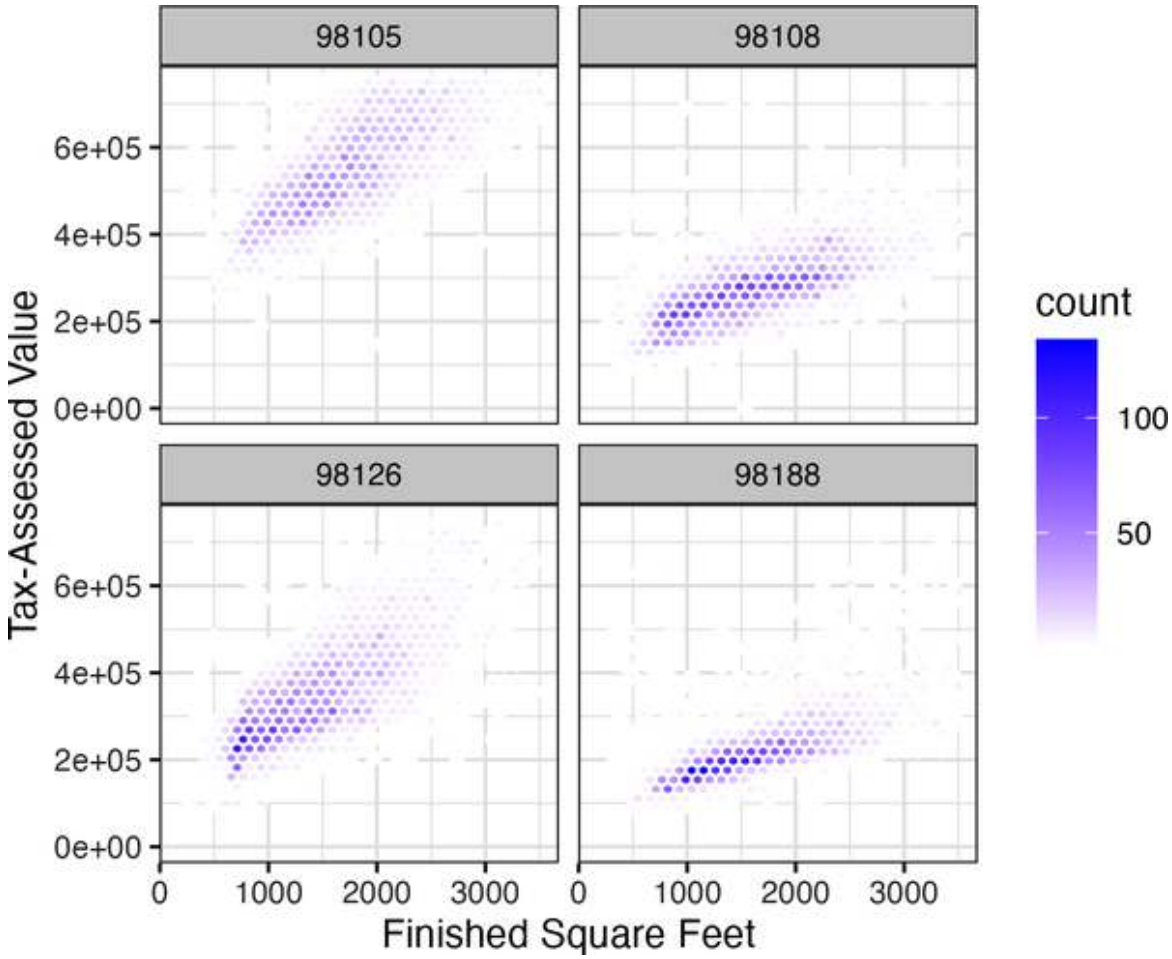


Figure 1-12. Tax-assessed value versus finished square feet by zip code

Most Python packages base their visualizations on `Matplotlib`. While it is in principle possible to create faceted graphs using `Matplotlib`, the code can get complicated. Fortunately, `seaborn` has a relatively straightforward way of creating these graphs:

```
zip_codes = [98188, 98105, 98108, 98126]
kc_tax_zip = kc_tax0.loc[kc_tax0.ZipCode.isin(zip_codes), :]

def hexbin(x, y, color, **kwargs):
    cmap = sns.light_palette(color, as_cmap=True)
    plt.hexbin(x, y, gridsize=25, cmap=cmap, **kwargs)

g = sns.FacetGrid(kc_tax_zip, col="ZipCode", col_wrap=2) ❶
g.map(hexbin, "SqFtTotLiving", "TaxAssessedValue",
      extent=[0, 3500, 0, 700000]) ❷
g.set_axis_labels("Finished Square Feet", "Tax-Assessed Value")
g.set_titles("Zip code {col_name:.0f}")
```

❶ Use the arguments `col` and `row` to specify the conditioning variables. For a single conditioning variable, use `col` together with `col_wrap` to wrap the faceted graphs into multiple rows.

❷ The `map` method calls the `hexbin` function with subsets of the original data set for the different zip codes. `extent` defines the limits of the x- and y-axes.

The concept of conditioning variables in a graphics system was pioneered with *Trellis graphics*, developed by Rick Becker, Bill Cleveland, and others at Bell Labs [Becker-1996]. This idea has propagated to various modern graphics systems, such as the `lattice` [Sarkar-2008] and `ggplot2` packages in R and the `seaborn` [Waskom-2021] and `Bokeh` [bokeh] modules in Python. Conditioning variables are also integral to business intelligence platforms such as Tableau and Spotfire. With the advent of vast computing power, modern visualization platforms have moved well beyond the humble beginnings of exploratory data analysis. However, key concepts and tools developed a half-century ago (e.g., simple boxplots) still form a foundation for these systems.

KEY IDEAS

- Hexagonal binning and contour plots are useful tools that permit graphical examination of two numeric variables at a time, without being overwhelmed by huge amounts of data.
- Contingency tables are the standard tool for looking at the counts of two categorical variables.
- Boxplots and violin plots allow you to plot a numeric variable against a categorical variable.

Further Reading

- The online graph galleries for R and Python have many examples of different types of graphs and the code to create them.
- *Modern Data Science with R* by Benjamin Baumer, Daniel Kaplan, and Nicholas Horton (Chapman & Hall/CRC Press, 2017) has an excellent presentation of "a grammar for graphics" (the "gg" in `ggplot`).
- *ggplot2: Elegant Graphics for Data Analysis* by Hadley Wickham (Springer, 2009) is an excellent resource from the creator of `ggplot2`. The book is available online at `ggplot2-book`.
- Josef Fruehwald has a web-based tutorial on `ggplot2`.

Interpreting Visualization Results with AI

It is often said that a picture is worth a thousand words, a fact that the graphs in this chapter clearly demonstrated. Graphs can reveal patterns, trends, and relationships in data that will not be obvious from raw numbers

alone. Analyzing and interpreting visualizations is an important part of data analysis.

The latest AI tools support multimodal inputs including images. You can use these tools to help interpret complex visualizations of statistical analyses. Here is an example of using ChatGPT 5.2 to interpret our visualizations of tax-assessed value versus finished square feet starting with Figure 1-8. In the ChatGPT web client you can paste Figure 1-8 along with the following prompt:

Prompt:	Write an interpretation of this figure in one paragraph.
ChatGPT:	The hexbin plot shows a clear positive relationship between finished square footage and tax-assessed value: as home size increases, assessed value generally rises. The highest concentration of properties appears around roughly 1,500–2,000 finished square feet with assessed values near $200,000–$300,000, indicating this is the most common segment of the housing stock. As square footage increases beyond this range, assessed values continue to increase but the density thins out, suggesting fewer large homes and greater variability in their valuations. Smaller homes cluster at lower assessed values, while larger homes show a wider spread in value, implying that factors beyond size (such as location or quality) play a bigger role at higher square footage levels.

It is always useful to follow up with additional questions to clarify or expand on the initial interpretation. You can also provide your insights to improve the analysis. For example, so far the additional bands above the main band have not been mentioned:

Prompt:	How about the two clusters above the main cluster?
ChatGPT:	Good catch—that is another notable feature. Above the main diagonal cluster, there appear to be two secondary clusters of homes with substantially higher tax-assessed values for a given square footage. These clusters suggest distinct submarkets: likely properties that are similar in size to typical homes but are assessed much higher, possibly due to premium locations (e.g., waterfront or desirable neighborhoods), higher construction quality, luxury finishes, or zoning differences. Their separation from the main cluster indicates these are not just outliers but systematically different groups, implying that the housing market represented here is segmented rather than governed by a single size–value relationship.

At this point, we now have a comprehensive interpretation of the figure that points to location as a key factor driving the observed differences. In Figure 1-12 we created separate plots by zip code. Again, we can use ChatGPT 5.2 to help interpret these plots:

Prompt:	Here is an analysis by location - write an interpretation in one paragraph.
ChatGPT:	Faceting the size–value relationship by ZIP code shows that the overall pattern decomposes into distinct, internally coherent local markets. Each ZIP exhibits a positive association between finished square footage and tax-assessed value, but the level and spread differ markedly: 98105 and 98126 display higher assessed values for comparable home sizes and broader value ranges, while 98108 and 98188 show lower, tighter bands, indicating more modest and homogeneous housing stock. The density peaks within each panel form relatively narrow, ZIP-specific trajectories, suggesting that the apparent clustering in the aggregate plot largely reflects location-driven segmentation rather than multiple regimes within a single neighborhood. In short, location strongly conditions the size–value relationship, helping explain the high-value clusters observed in the pooled data.

And finally, create a summary of the results:

Prompt:	Write a concise summary of the results.
ChatGPT:	Finished square footage is strongly positively associated with tax-assessed value, but the relationship is not uniform across the housing market. Aggregate plots show increasing variability and apparent clustering at higher values, which initially suggests multiple submarkets. When stratified by ZIP code, these clusters resolve into distinct, internally consistent location-specific patterns, with some ZIPs exhibiting systematically higher valuations and greater spread than others. Overall, the results indicate that location is a primary driver of segmentation, and square footage explains value best when analyzed within local markets rather than across the entire data set.

As shown in this example, a dialog with an AI can assist the interpretation of complex visualizations and suggest ways of following up on initial findings. We will discuss this example further in Chapters 4, 7, and 11.

Summary

Exploratory data analysis (EDA), pioneered by John Tukey, set a foundation for the field of data science. The key idea of EDA is that the first and most important step in any project based on data is to *look at the data*. By summarizing and visualizing the data, you can gain valuable intuition and understanding of the project.

This chapter has reviewed concepts ranging from simple metrics, such as estimates of location and variability, to rich visual displays that explore the relationships between multiple variables, as in Figure 1-12. The diverse set of tools and techniques being developed by the open source community, combined with the expressiveness of the R and Python languages, has

created a plethora of ways to explore and analyze data. Exploratory analysis should be a cornerstone of any data science project.

In this chapter, we demonstrated how LLMs can be used to improve or create data dictionaries as well as to help interpret and improve visualizations.

Exploration with AI

1. Use AI to learn more about *robust estimates of location*. You can ask to see implementations in R and Python or explore other algorithms not covered here.
2. Have a conversation with an AI assistant to learn more about "estimates of variability." Are there specific aspects you would like to explore further?
3. Pick a data dictionary from a public data set (such as those hosted at the UC Irvine Machine Learning Repository) and improve it using an LLM of your choice. Compare the results with the original data dictionary. Did you see an improvement?
4. Pick a data set and use an LLM of your choice to create a data dictionary. How does the result compare with any existing data dictionary? If no data dictionary exists, how would you validate the results?
5. Use an AI assistant to help you interpret the visualizations in this chapter. How do the interpretations compare with your own? Did the AI assistant point out any interesting features that you missed?
6. For a data set of your choice, use an LLM to create and improve a complex visualization.
7. Plan an EDA of a data set of your choice. What questions would you ask? What metrics and visualizations would you use to answer

those questions? Try with and without a data dictionary. Do you see an improvement of the results when using a data dictionary?

---

[1] You can set your OpenAI API key using the `api_key` argument to `ChatOpenAI` or define it as an environment variable. The latter is recommended to avoid exposing your API key in your code.

Chapter 2. Data and Sampling Distributions

A popular misconception holds that the era of big data means the end of the need for sampling. In fact, the proliferation of data of varying quality and relevance reinforces the need for sampling as a tool to work efficiently with a variety of data and to minimize bias. Even in a big data project, predictive models are typically developed and piloted with samples. Samples are also used in tests of various sorts (e.g., comparing the effect of web page designs on clicks).

Figure 2-1 shows a schematic that underpins the concepts we discuss in this chapter—data and sampling distributions. The lefthand side represents a population that, in statistics, is assumed to follow an underlying but *unknown* distribution. All that is available is the *sample* data and its empirical distribution, shown on the righthand side. To get from the lefthand side to the righthand side, a *sampling* procedure is used (represented by an arrow). Traditional statistics focused very much on the lefthand side, using theory based on strong assumptions about the population. Modern statistics has moved to the righthand side, where such assumptions are not needed.

In general, data scientists need not worry about the theoretical nature of the lefthand side and instead should focus on the sampling procedures and the data at hand. There are some notable exceptions. Sometimes data is generated from a physical process that can be modeled. The simplest example is flipping a coin: this follows a binomial distribution. Any real-life binomial situation (buy or don't buy, fraud or no fraud, click or don't click) can be modeled effectively by a coin (with modified probability of landing heads, of course). In these cases, we can gain additional insight by using our understanding of the population.

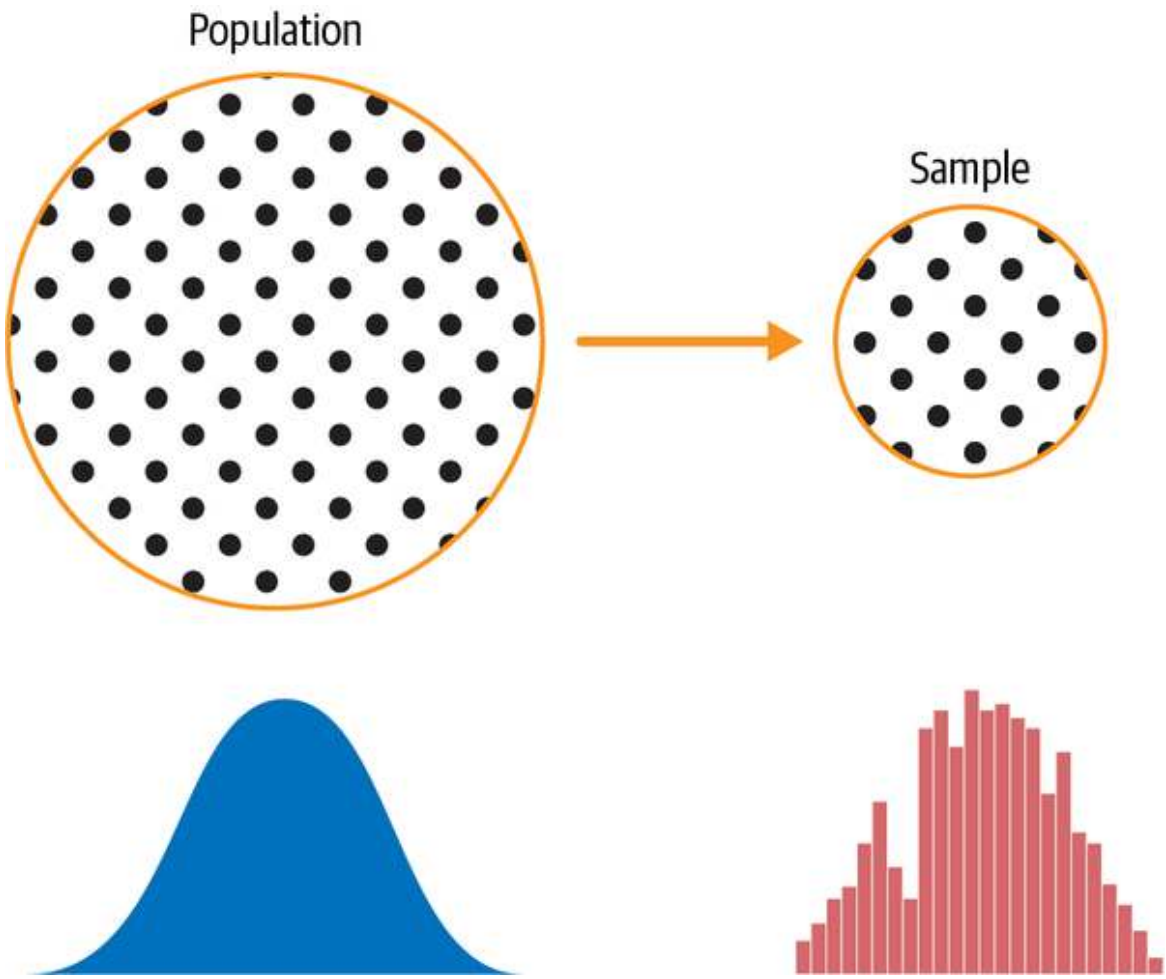


Figure 2-1. Population versus sample

Random Sampling and Sample Bias

A *sample* is a subset of data from a larger data set; statisticians call this larger data set the *population*. A population in statistics is not the same thing as in biology—it is a large, defined (but sometimes theoretical or imaginary) set of data.

Random sampling is a process in which each available member of the population being sampled has an equal chance of being chosen for the sample at each draw. The sample that results is called a *simple random sample*. Sampling can be done *with replacement*, in which observations are put back in the population after each draw for possible future reselection.

Or it can be done *without replacement*, in which case observations, once selected, are unavailable for future draws.

Data quality often matters more than data quantity when making an estimate or a model based on a sample. Data quality in data science involves completeness, consistency of format, cleanliness, and accuracy of individual data points. Statistics adds the notion of *representativeness*.

KEY TERMS FOR RANDOM SAMPLING

Sample

A subset from a larger data set

Population

The larger data set or idea of a data set

N (n)

The size of the population (sample)

Random sampling

Drawing elements into a sample at random

Stratified sampling

Dividing the population into strata and randomly sampling from each stratum

Stratum (pl., strata)

A homogeneous subgroup of a population with common characteristics

Simple random sample

The sample that results from random sampling without stratifying the population

Bias

Systematic error

Sample bias

A sample that misrepresents the population

The classic example is the *Literary Digest* poll of 1936 that predicted a victory of Alf Landon over Franklin Roosevelt. The *Literary Digest*, a leading periodical of the day, polled its entire subscriber base plus additional lists of individuals, a total of over 10 million people, and predicted a landslide victory for Landon. George Gallup, founder of the Gallup Poll, conducted biweekly polls of just 2,000 people and accurately predicted a Roosevelt victory. The difference lay in the selection of those polled.

The *Literary Digest* opted for quantity, paying little attention to the method of selection. It ended up polling those with relatively high socioeconomic status (its own subscribers, plus those who, by virtue of owning luxuries like telephones and automobiles, appeared in marketers' lists). The result was *sample bias*; that is, the sample was different in some meaningful and nonrandom way from the larger population it was meant to represent. The term *nonrandom* is important—hardly any sample, including random samples, will be exactly representative of the population. Sample bias occurs when the difference is meaningful, and it can be expected to continue for other samples drawn in the same way as the first.

SELF-SELECTION SAMPLING BIAS

The reviews of restaurants, hotels, cafés, and so on that you read on social media sites like Yelp are prone to bias because the people submitting them are not randomly selected; rather, they themselves have taken the initiative to write. This leads to self-selection bias—the people motivated to write reviews may have had poor experiences, may have an association with the establishment, or may simply be a different type of person from those who do not write reviews. Note that while self-selection samples can be unreliable indicators of the true state of affairs, they may be more reliable in simply comparing one establishment to a similar one; the same self-selection bias might apply to each.

Bias

Statistical bias refers to measurement or sampling errors that are systematic and produced by the measurement or sampling process. An important distinction should be made between errors due to random chance and errors due to bias. Consider the physical process of a gun shooting at a target. It will not hit the absolute center of the target every time, or even much at all. An unbiased process will produce error, but it is random and does not tend strongly in any direction (see Figure 2-2).

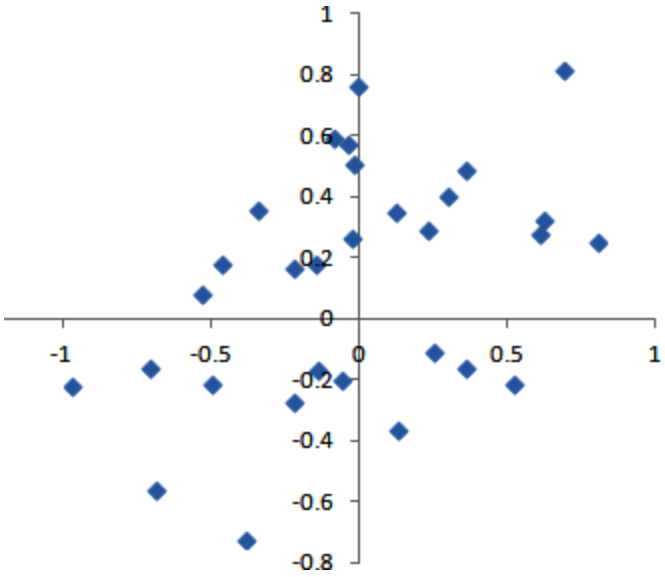


Figure 2-2. Scatterplot of shots from a gun with true aim

The results shown in Figure 2-3 show a biased process—there is still random error in both the x and y direction, but there is also a bias. Shots tend to fall in the upper-right quadrant.

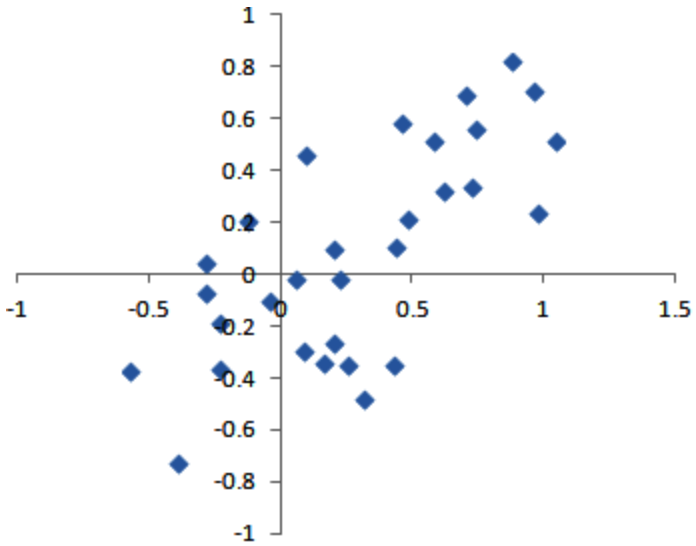


Figure 2-3. Scatterplot of shots from a gun with biased aim

Bias comes in different forms, and may be observable or invisible. When a result does suggest bias (e.g., by reference to a benchmark or actual values), it is often an indicator that a statistical or machine learning model has been misspecified, or an important variable left out.

Random Selection

To avoid the problem of sample bias that led the *Literary Digest* to predict Landon over Roosevelt, George Gallup (shown in Figure 2-4) opted for more scientifically chosen methods to achieve a sample that was representative of the US voting electorate. There are now a variety of methods to achieve representativeness, but at the heart of all of them lies *random sampling*.



Figure 2-4. George Gallup, catapulted to fame by the Literary Digest*'s "big data" failure*

Random sampling is not always easy. Proper definition of an accessible population is key. Suppose we want to generate a representative profile of customers and we need to conduct a pilot customer survey. The survey needs to be representative but is labor intensive.

First, we need to define who a customer is. We might select all customer records where purchase amount > 0. Do we include all past customers? Do we include refunds? Internal test purchases? Resellers? Both billing agent and customer?

Next, we need to specify a sampling procedure. It might be "select 100 customers at random." Where a sampling from a flow is involved (e.g., real-time customer transactions or web visitors), timing considerations may be important (e.g., a web visitor at 10 a.m. on a weekday may be different from a web visitor at 10 p.m. on a weekend).

In *stratified sampling*, the population is divided up into *strata*, and random samples are taken from each stratum. Political pollsters might seek to learn the electoral preferences of whites, Blacks, and Hispanics. A simple random sample taken from the population would yield too few Blacks and Hispanics, so those strata could be overweighted in stratified sampling to yield equivalent sample sizes.

Size Versus Quality: When Does Size Matter?

In the era of big data, it is sometimes surprising that smaller is better. Time and effort spent on random sampling not only reduces bias but also allows greater attention to data exploration and data quality. For example, missing data and outliers may contain useful information. It might be prohibitively expensive to track down missing values or evaluate outliers in millions of records, but doing so in a sample of several thousand records may be feasible. Data plotting and manual inspection bog down if there is too much data.

So when *are* massive amounts of data needed?

The classic scenario for the value of big data is when the data is not only big but sparse as well. Consider the search queries received by Google, where columns are terms, rows are individual search queries, and cell values are either 0 or 1, depending on whether a query contains a term. The goal is to determine the best predicted search destination for a given query. There are over 150,000 words in the English language, and Google processes over one trillion queries per year. This yields a huge matrix, the vast majority of whose entries are "0."

This is a true big data problem—only when such enormous quantities of data are accumulated can effective search results be returned for most queries. And the more data accumulates, the better the results. For popular search terms this is not such a problem—effective data can be found fairly quickly for the handful of extremely popular topics trending at a particular time. The real value of modern search technology lies in the ability to return detailed and useful results for a huge variety of search queries, including those that occur with a frequency, say, of only one in a million.

Consider the search phrase "Ricky Ricardo and Little Red Riding Hood." In the early days of the internet, this query would probably have returned results on the bandleader Ricky Ricardo, the television show *I Love Lucy* in which that character appeared, and the children's story *Little Red Riding Hood*. Both of those individual items would have had many searches to refer to, but the combination would have had very few. Later, now that trillions of search queries have been accumulated, this search query returns

the exact *I Love Lucy* episode in which Ricky narrates, in dramatic fashion, the *Little Red Riding Hood* story to his infant son in a comic mix of English and Spanish.

Keep in mind that the number of actual *pertinent* records—ones in which this exact search query, or something very similar, appears (together with information on what link people ultimately clicked on)—might need only be in the thousands to be effective. However, many trillions of data points are needed to obtain these pertinent records (and random sampling, of course, will not help). See also "Long-Tailed Distributions".

Sample Mean Versus Population Mean

The symbol x (pronounced "x-bar") is used to represent the mean of a sample from a population, whereas μ is used to represent the mean of a population. Why make the distinction? Information about samples is observed, and information about large populations is often inferred from smaller samples. Statisticians like to keep the two things separate in the symbology.

KEY IDEAS

- Even in the era of big data, random sampling remains an important arrow in the data scientist's quiver.
- Bias occurs when measurements or observations are systematically in error because they are not representative of the full population.
- Data quality is often more important than data quantity, and random sampling can reduce bias and facilitate quality improvement that would otherwise be prohibitively expensive.

Further Reading

- A useful review of sampling procedures can be found in Ronald Fricker's chapter "Sampling Methods for Online Surveys" in *The SAGE Handbook of Online Research Methods*, 2nd ed., edited by Nigel G. Fielding, Raymond M. Lee, and Grant Blank (SAGE Publications, 2016). This chapter includes a review of the modifications to random sampling that are often used for practical reasons of cost or feasibility.
- The story of the *Literary Digest* poll failure can be found on the Capital Century website.

Selection Bias

To paraphrase Yogi Berra: if you don't know what you're looking for, look hard enough and you'll find it.

Selection bias refers to the practice of selectively choosing data—consciously or unconsciously—in a way that leads to a conclusion that is misleading or ephemeral.

KEY TERMS FOR SELECTION BIAS

Selection bias

Bias resulting from the way in which observations are selected

Data snooping

Extensive hunting through data in search of something interesting

Vast search effect

Bias or nonreproducibility resulting from repeated data modeling, or modeling data with large numbers of predictor variables

If you specify a hypothesis and conduct a well-designed experiment to test it, you can have high confidence in the conclusion. This is frequently not what occurs, however. Often, one looks at available data and tries to discern patterns. But are the patterns real? Or are they just the product of *data snooping*—that is, extensive hunting through the data until something interesting emerges? There is a saying among statisticians: "If you torture the data long enough, sooner or later it will confess."

The difference between a phenomenon that you verify when you test a hypothesis using an experiment and a phenomenon that you discover by perusing available data can be illuminated with the following thought experiment.

Imagine that someone tells you they can flip a coin and have it land heads on the next 10 tosses. You challenge them (the equivalent of an experiment), and they proceed to toss the coin 10 times, with all flips landing heads. Clearly you ascribe some special talent to this person—the probability that 10 coin tosses will land heads just by chance is less than 1 in 1,000.

Now imagine that the announcer at a sports stadium asks the 20,000 people in attendance each to toss a coin 10 times, and to report to an usher if they get 10 heads in a row. The chance that *somebody* in the stadium will get 10 heads is extremely high (more than 99%—it's 1 minus the probability that nobody gets 10 heads). Clearly, selecting after the fact the person (or persons) who gets 10 heads at the stadium does not indicate they have any special talent—it's most likely luck.

Since repeated review of large data sets is a key value proposition in data science, selection bias is something to worry about. A form of selection bias of particular concern to data scientists is the *vast search effect*. If you repeatedly run different models and ask different questions with a large data set, you are bound to find something interesting. But is the result you found truly something interesting, or is it the chance outlier?

We can guard against this by using a holdout set, and sometimes more than one holdout set, against which to validate performance. You can also use

target shuffling (a permutation test in which you shuffle the outcome or target variable) to test the validity of predictive associations that a machine learning model suggests.

Typical forms of selection bias in statistics, in addition to the vast search effect, include nonrandom sampling (see "Random Sampling and Sample Bias"), cherry-picking data, selection of time intervals that accentuate a particular statistical effect, and stopping an experiment when the results look "interesting."

Regression to the Mean

Regression to the mean refers to a phenomenon involving successive measurements on a given variable: extreme observations tend to be followed by more central ones. Attaching special focus and meaning to the extreme value can lead to a form of selection bias.

Sports fans are familiar with the "rookie of the year, sophomore slump" phenomenon. Among the athletes who begin their career in a given season (the rookie class), there is always one who performs better than all the rest. Generally, this "rookie of the year" does not do as well in their second year. Why not?

In nearly all major sports, at least those played with a ball or puck, there are two elements that play a role in overall performance:

- Skill
- Luck

Regression to the mean is a consequence of a particular form of selection bias. When we select the rookie with the best performance, skill and good luck are probably contributing. In their next season, the skill will still be there, but very often the luck will not be, so their performance will decline —it will regress. The phenomenon was first identified by Francis Galton in 1886 [Galton-1886], who wrote of it in connection with genetic tendencies;

for example, the children of extremely tall men tend not to be as tall as their father (see Figure 2-5).

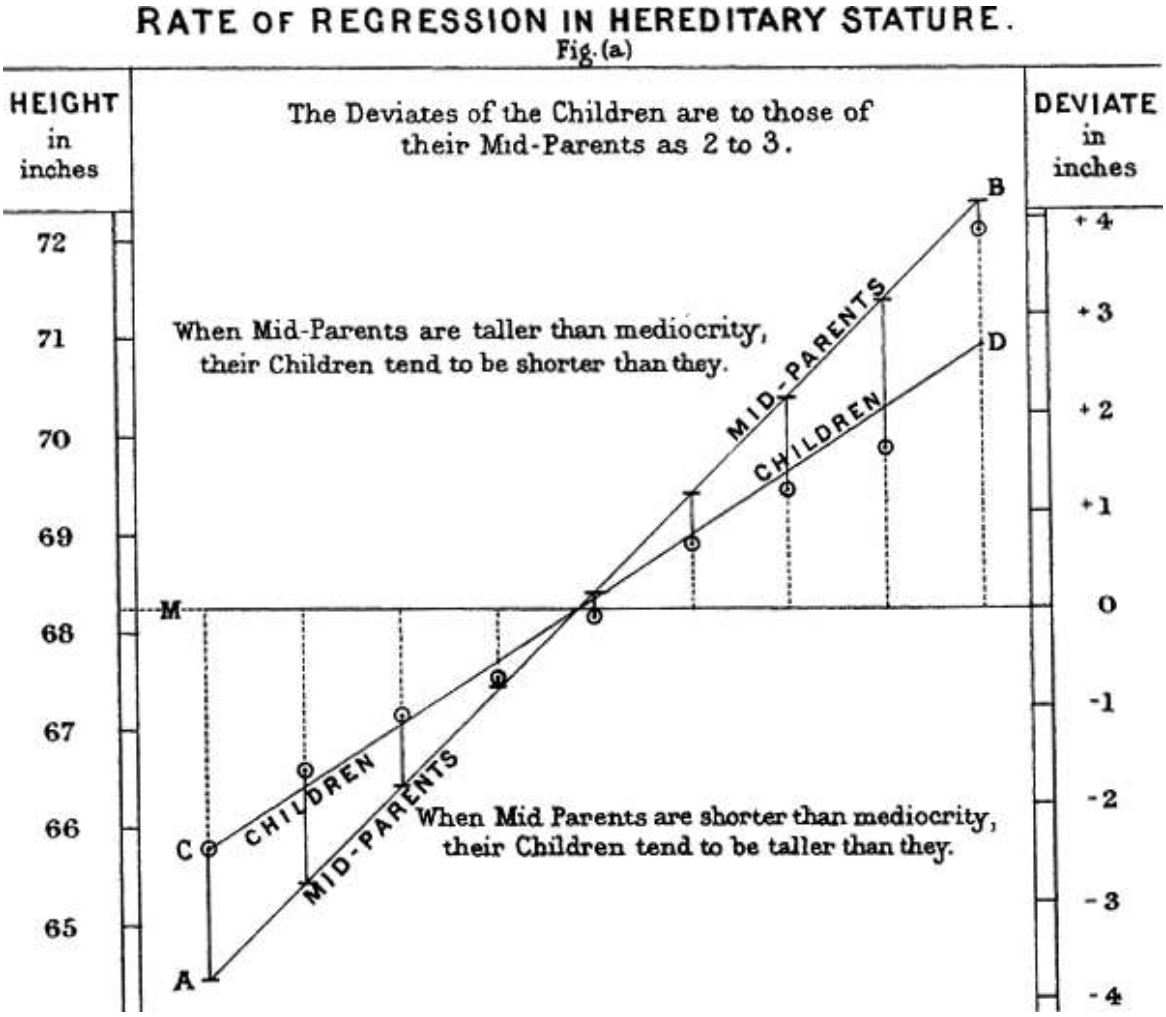


Figure 2-5. Galton's study that identified the phenomenon of regression to the mean

REGRESSION TO THE MEAN

Regression to the mean, meaning to "go back," is distinct from the statistical modeling method of linear regression, in which a linear relationship is estimated between predictor variables and an outcome variable.

KEY IDEAS

- Specifying a hypothesis and then collecting data following randomization and random sampling principles ensures against bias.
- All other forms of data analysis run the risk of bias resulting from the data collection/analysis process (repeated running of models in data mining, data snooping in research, and after-the-fact selection of interesting events).

Further Reading

- Christopher J. Pannucci and Edwin G. Wilkins' article "Identifying and Avoiding Bias in Research" in (surprisingly not a statistics journal) *Plastic and Reconstructive Surgery* (August 2010) has an excellent review of various types of bias that can enter into research, including selection bias.
- Michael Harris's article "Fooled by Randomness, Over-fitting And Selection Bias" provides an interesting review of selection bias considerations in stock market trading schemes, from the perspective of traders.

Sampling Distribution of a Statistic

The term *sampling distribution* of a statistic refers to the distribution of some sample statistic over many samples drawn from the same population. Much of classical statistics is concerned with making inferences from (small) samples to (very large) populations.

KEY TERMS FOR SAMPLING DISTRIBUTION

Sample statistic

A metric calculated for a sample of data drawn from a larger population

Data distribution

The frequency distribution of individual *values* in a data set

Sampling distribution

The frequency distribution of a *sample statistic* over many samples or resamples

Central limit theorem

The tendency of the sampling distribution to take on a normal shape as sample size rises

Standard error

The variability (standard deviation) of a *sample statistic* over many samples (not to be confused with *standard deviation*, which by itself, refers to variability of individual data *values*)

Typically, a sample is drawn with the goal of measuring something (with a *sample statistic*) or modeling something (with a statistical or machine learning model). Since our estimate or model is based on a sample, it might be in error; it might be different if we were to draw a different sample. We are therefore interested in how different it might be—a key concern is *sampling variability*. If we had lots of data, we could draw additional samples and observe the distribution of a sample statistic directly. Typically, we will calculate our estimate or model using as much data as is easily

available, so the option of drawing additional samples from the population is not readily available.

DATA DISTRIBUTION VERSUS SAMPLING DISTRIBUTION

It is important to distinguish between the distribution of the individual data points, known as *the data distribution*, and the distribution of a sample statistic, known as the *sampling distribution*.

The distribution of a sample statistic such as the mean is likely to be more regular and bell-shaped than the distribution of the data itself. The larger the sample the statistic is based on, the more this is true. Also, the larger the sample, the narrower the distribution of the sample statistic.

This is illustrated in an example using annual income for loan applicants to LendingClub (see "A Small Example: Predicting Loan Default" for a description of the data). Take three samples from this data: a sample of 1,000 values, a sample of 1,000 means of 5 values, and a sample of 1,000 means of 20 values. Then plot a histogram of each sample to produce Figure 2-6.

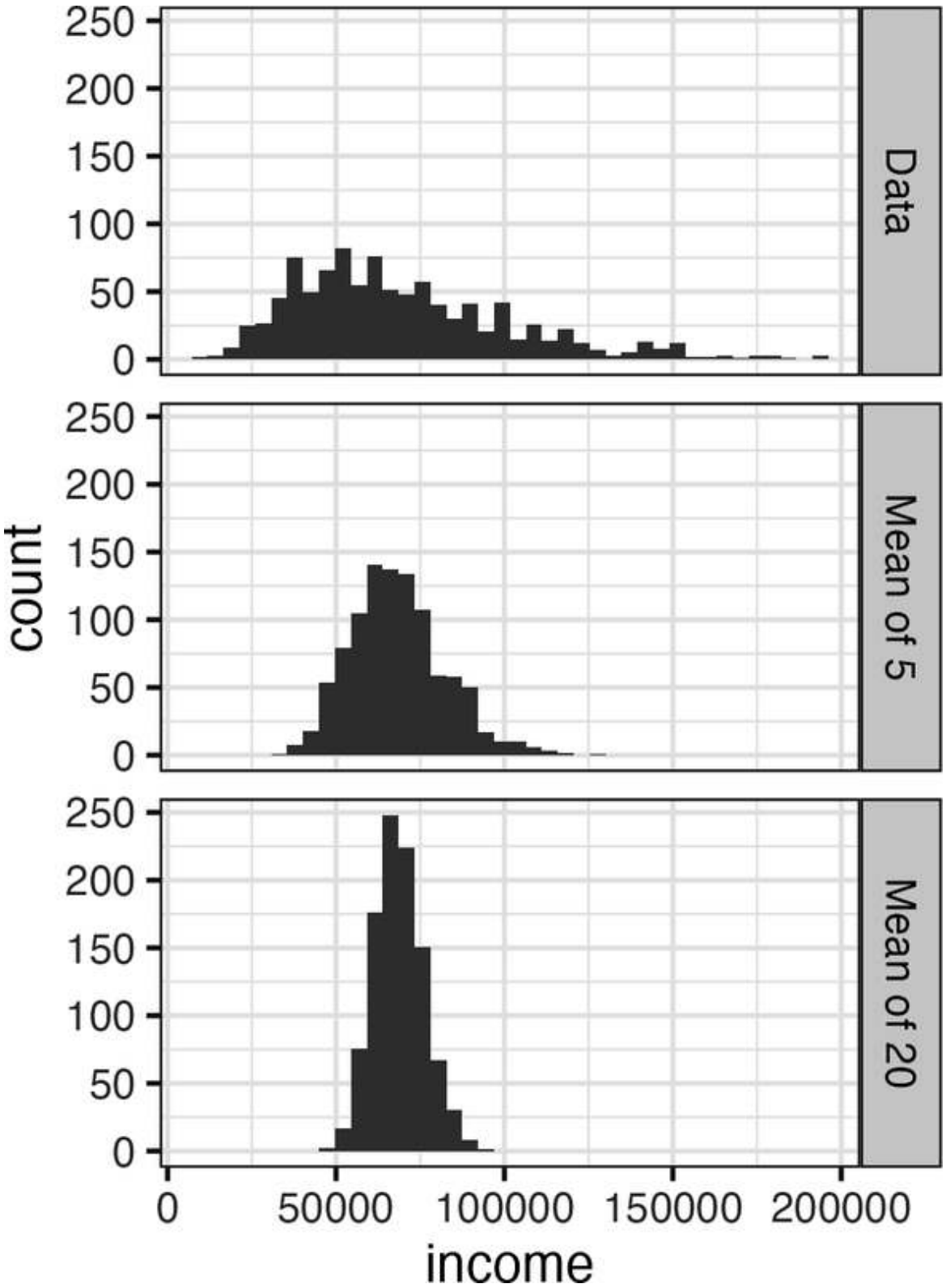


Figure 2-6. Histogram of annual incomes of 1,000 loan applicants (top), then 1,000 means of n=5 applicants (middle), and finally 1,000 means of n=20 applicants (bottom)

The histogram of the individual data values is broadly spread out and skewed toward higher values, as is to be expected with income data. The histograms of the means of 5 and 20 are increasingly compact and more bell-shaped. Here is the R code to generate these histograms, using the visualization package `ggplot2`:

```
library(tidyverse)

loans_income <- read_csv("loans_income.csv", show_col_types = FALSE) %>%
  pull(x)

# take a simple random sample
samp_data <- data.frame(income = sample(loans_income, 1000),
  type = "Data")
# take a sample of means of 5 values
samp_mean_05 <- tibble(
  income = replicate(1000, mean(sample(loans_income, 5))),
  type = "Mean of 5"
)
# take a sample of means of 20 values
samp_mean_20 <- tibble(
  income = replicate(1000, mean(sample(loans_income, 20))),
  type = "Mean of 20"
)
income <- bind_rows(samp_data, samp_mean_05, samp_mean_20) %>%
  mutate(type = factor(type,
      levels = c("Data", "Mean of 5", "Mean of 20"))
  )

# plot the histograms
ggplot(income, aes(x = income)) +
  geom_histogram(bins = 40) +
  facet_grid(type ~ .)
```

The Python code uses `seaborn`'s `FacetGrid` to show the three histograms:

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

loans_income = pd.read_csv("loans_income.csv").squeeze("columns")

sample_data = pd.DataFrame({
    "income": loans_income.sample(1000),
```

```
    "type": "Data",
})
sample_mean_05 = pd.DataFrame({
    "income": [loans_income.sample(5).mean() for _ in range(1000)],
    "type": "Mean of 5",
})
sample_mean_20 = pd.DataFrame({
    "income": [loans_income.sample(20).mean() for _ in range(1000)],
    "type": "Mean of 20",
})
results = pd.concat([sample_data, sample_mean_05, sample_mean_20])

g = sns.FacetGrid(results, col="type", col_wrap=1, height=2, aspect=2)
g.map(plt.hist, "income", range=[0, 200000], bins=40)
g.set_axis_labels("Income", "Count")
g.set_titles("{col_name}")
```

Central Limit Theorem

The phenomenon we've just described is termed the *central limit theorem*. It says that the means drawn from multiple samples will resemble the familiar bell-shaped normal curve (see "Normal Distribution"), even if the source population is not normally distributed, provided that the sample size is large enough and the departure of the data from normality is not too great. The central limit theorem allows normal-approximation formulas like the t-distribution to be used in calculating sampling distributions for inference—that is, confidence intervals and hypothesis tests.

The central limit theorem receives a lot of attention in traditional statistics texts because it underlies the machinery of hypothesis tests and confidence intervals, which themselves consume half the space in such texts. Data scientists should be aware of this role; however, since formal hypothesis tests and confidence intervals play a small role in data science, and the *bootstrap* (see "The Bootstrap") is available in any case, the central limit theorem is not so central in the practice of data science.

Standard Error

The *standard error* is a single metric that sums up the variability in the sampling distribution for a statistic. The standard error can be estimated

using a statistic based on the standard deviation *s* of the sample values, and the sample size *n*:

$$\text{Standard error} = SE = \frac{s}{\sqrt{n}}$$

As the sample size increases, the standard error decreases, corresponding to what was observed in Figure 2-6. The relationship between standard error and sample size is sometimes referred to as the *square root of n* rule: to reduce the standard error by a factor of 2, the sample size must be increased by a factor of 4.

The validity of the standard error formula arises from the central limit theorem. In fact, you don't need to rely on the central limit theorem to understand standard error. Consider the following approach to measuring standard error:

1. Collect a number of brand-new samples from the population.
2. For each new sample, calculate the statistic (e.g., mean).
3. Calculate the standard deviation of the statistics computed in step 2; use this as your estimate of standard error.

In practice, this approach of collecting new samples to estimate the standard error is typically not feasible (and statistically very wasteful). Fortunately, it turns out that it is not necessary to draw brand new samples; instead, you can use *bootstrap* resamples. In modern statistics, the bootstrap has become the standard way to estimate standard error. It can be used for virtually any statistic and does not rely on the central limit theorem or other distributional assumptions.

STANDARD DEVIATION VERSUS STANDARD ERROR

Do not confuse standard deviation (which measures the variability of individual data points) with standard error (which measures the variability of a sample metric).

KEY IDEAS

- The frequency distribution of a sample statistic tells us how that metric would turn out differently from sample to sample.
- This sampling distribution can be estimated via the bootstrap, or via formulas that rely on the central limit theorem.
- A key metric that sums up the variability of a sample statistic is its standard error.

Further Reading

David Lane's online multimedia resource in statistics has a useful simulation that allows you to select a sample statistic, a sample size, and the number of iterations and visualize a histogram of the resulting frequency distribution.

The Bootstrap

One easy and effective way to estimate the sampling distribution of a statistic, or of model parameters, is to draw additional samples, with replacement, from the sample itself and recalculate the statistic or model for each resample. This procedure is called the *bootstrap*, and it does not necessarily involve any assumptions about the data or the sample statistic being normally distributed.

KEY TERMS FOR THE BOOTSTRAP

Bootstrap sample

A sample taken with replacement from an observed data set

Resampling

The process of taking repeated samples from observed data; includes both bootstrap and permutation (shuffling) procedures

Conceptually, you can imagine the bootstrap as replicating the original sample thousands or millions of times so that you have a hypothetical population that embodies all the knowledge from your original sample (it's just larger). You can then draw samples from this hypothetical population for the purpose of estimating a sampling distribution; see Figure 2-7.

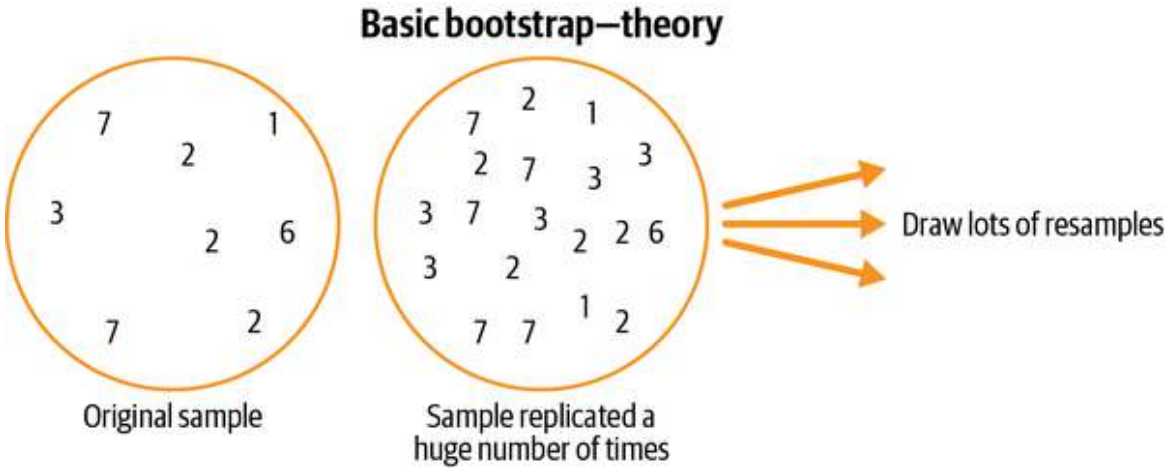


Figure 2-7. The idea of the bootstrap

In practice, it is not necessary to actually replicate the sample a huge number of times. We simply replace each observation after each draw; that is, we *sample with replacement*. In this way we effectively create an infinite population in which the probability of an element being drawn remains unchanged from draw to draw. The algorithm for a bootstrap resampling of the mean, for a sample of size *n*, is as follows:

1. Draw a sample value, record it, and then replace it.
2. Repeat *n* times.
3. Record the mean of the *n* resampled values.
4. Repeat steps 1–3 *R* times.
5. Use the *R* results to:
    a. Calculate their standard deviation (this estimates sample mean standard error).
    b. Produce a histogram or boxplot.
    c. Find a confidence interval.

R, the number of iterations of the bootstrap, is set somewhat arbitrarily. The more iterations you do, the more accurate the estimate of the standard error, or the confidence interval. The result from this procedure is a bootstrap set of sample statistics or estimated model parameters, which you can then examine to see how variable they are.

The R package `boot` combines these steps in one function. For example, the following applies the bootstrap to the incomes of people taking out loans:

```
library(boot)
stat_fun <- function(x, idx) median(x[idx])
boot_obj <- boot(loans_income, R = 1000, statistic = stat_fun)
```

The function `stat_fun` computes the median for a given sample specified by the index `idx`. The result is as follows:

```
boot_obj
---
...
Bootstrap Statistics :
    original   bias    std. error
t1*    62000 -70.5595    209.1515
```

The original estimate of the median is $62,000. The bootstrap distribution indicates that the estimate has a *bias* of about –$70 and a standard error of $209. The results will vary slightly between consecutive runs of the algorithm.

In Python we can use the `bootstrap` function of the `scipy` package:

```
res = stats.bootstrap([loans_income], np.median, n_resamples=1000,
    method="basic")  ❶

print("Bootstrap Statistics:")
print(f"original: {np.median(loans_income)}")
print(f"bias: {np.mean(res.bootstrap_distribution) -
np.median(loans_income)}")
print(f"std. error: {res.standard_error}")
print(f"confidence interval: {res.confidence_interval}")
```

❶ The `bootstrap` function requires the data to be passed as a sequence of arrays, hence the brackets around `loans_income`.

The bootstrap can be used with multivariate data, where the rows are sampled as units (see Figure 2-8). A model might then be run on the bootstrapped data, for example, to estimate the stability (variability) of model parameters, or to improve predictive power. With classification and regression trees (also called *decision trees*), running multiple trees on bootstrap samples and then averaging their predictions (or, with classification, taking a majority vote) generally performs better than using a single tree. This process is called *bagging* (short for "bootstrap aggregating"; see "Bagging and the Random Forest").

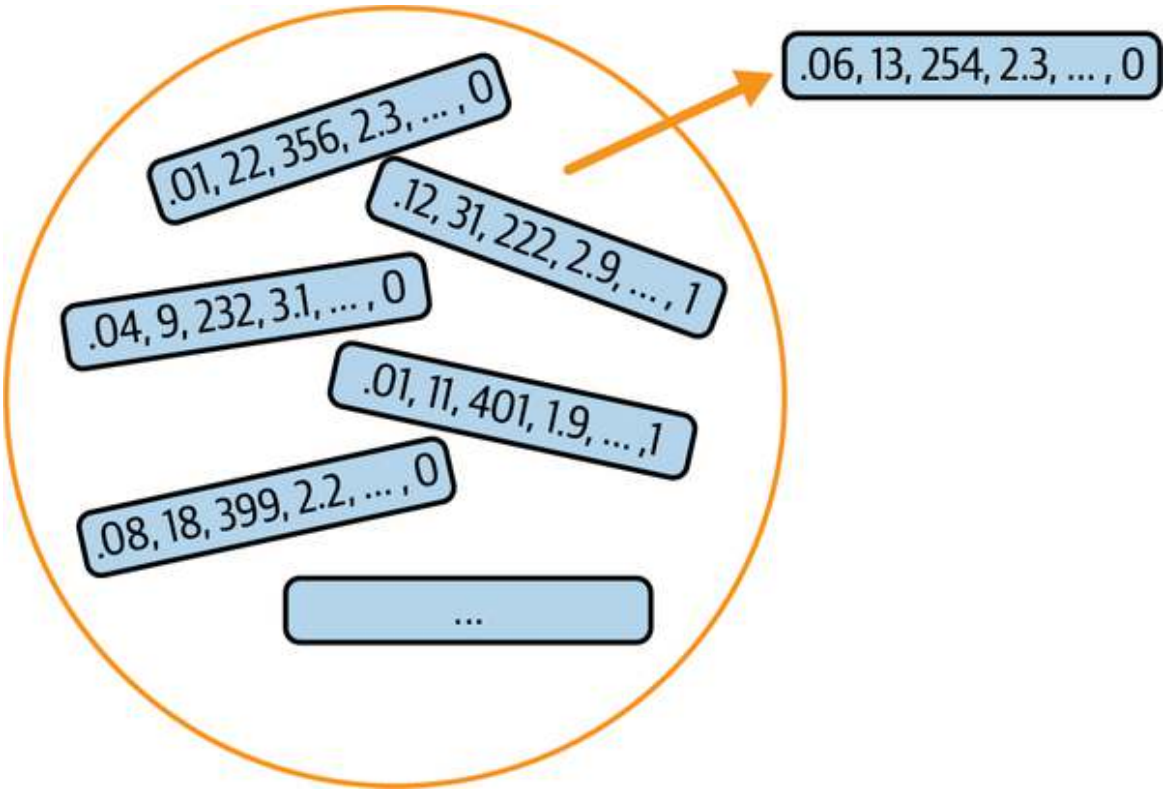


Figure 2-8. Multivariate bootstrap sampling

The repeated resampling of the bootstrap is conceptually simple, and Julian Simon, an economist and demographer, published a compendium of resampling examples, including the bootstrap, in his 1969 text *Basic Research Methods in Social Science* (Random House). However, it is also computationally intensive and was not a feasible option before the widespread availability of computing power. The technique gained its name and took off with the publication of several journal articles and a book by Stanford statistician Bradley Efron in the late 1970s and early 1980s. It was particularly popular among researchers who use statistics but are not statisticians, and for use with metrics or models where mathematical approximations are not readily available. The sampling distribution of the mean has been well established since 1908; the sampling distribution of many other metrics has not. The bootstrap can be used for sample size determination; experiment with different values for *n* to see how the sampling distribution is affected.

The bootstrap was met with considerable skepticism when it was first introduced; it had, to many, the aura of spinning gold from straw. This

skepticism stemmed from a misunderstanding of the bootstrap's purpose.

> **THE BOOTSTRAP DOES NOT CREATE DATA**
>
> The bootstrap does not compensate for a small sample size; it does not create new data, nor does it fill in holes in an existing data set. It merely informs us about how lots of additional samples would behave when drawn from a population like our original sample.

Resampling Versus Bootstrapping

Sometimes the term *resampling* is used synonymously with the term *bootstrapping*, as just outlined. More often, the term *resampling* also includes permutation procedures (see "Permutation Test"), where multiple samples are combined and the sampling may be done without replacement. In any case, the term *bootstrap* always implies sampling with replacement from an observed data set.

> **KEY IDEAS**
>
> - The bootstrap (sampling with replacement from a data set) is a powerful tool for assessing the variability of a sample statistic.
> - The bootstrap can be applied in a similar fashion in a wide variety of circumstances, without extensive study of mathematical approximations to sampling distributions.
> - It also allows us to estimate sampling distributions for statistics where no mathematical approximation has been developed.
> - When applied to predictive models, aggregating multiple bootstrap sample predictions (bagging) outperforms the use of a single model.

Further Reading

- *An Introduction to the Bootstrap* by Bradley Efron and Robert Tibshirani (Chapman & Hall, 1993) was the first book-length treatment of the bootstrap. It is still widely read.
- The retrospective on the bootstrap in the May 2003 issue of *Statistical Science* (vol. 18, no. 2) discusses (among other antecedents, in Peter Hall's "A Short Prehistory of the Bootstrap") Julian Simon's initial publication of the bootstrap in 1969.
- See *An Introduction to Statistical Learning* by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (Springer, R edition 2021, Python edition 2023) for sections on the bootstrap and, in particular, bagging.

Confidence Intervals

Frequency tables, histograms, boxplots, and standard errors are all ways to understand the potential error in a sample estimate. Confidence intervals are another.

KEY TERMS FOR CONFIDENCE INTERVALS

Confidence level

The percentage of confidence intervals, constructed in the same way from the same population, that are expected to contain the statistic of interest

Interval endpoints

The top and bottom of the confidence interval

There is a natural human aversion to uncertainty; people (especially experts) say "I don't know" far too rarely. Analysts and managers, while acknowledging uncertainty, nonetheless place undue faith in an estimate

when it is presented as a single number (a *point estimate*). Presenting an estimate not as a single number but as a range is one way to counteract this tendency. Confidence intervals do this in a manner grounded in statistical sampling principles.

Confidence intervals always come with a coverage level, expressed as a (high) percentage, say 90% or 95%. One way to think of a 90% confidence interval is as follows: it is the interval that encloses the central 90% of the bootstrap sampling distribution of a sample statistic (see "The Bootstrap"). More generally, an x% confidence interval around a sample estimate should, on average, contain similar sample estimates x% of the time (when a similar sampling procedure is followed).

Given a sample of size n, and a sample statistic of interest, the algorithm for a bootstrap confidence interval is as follows:

1. Draw a random sample of size n with replacement from the data (a resample).
2. Record the statistic of interest for the resample.
3. Repeat steps 1–2 many (R) times.
4. For an x% confidence interval, trim [(100–x) / 2]% of the R resample results from either end of the distribution.
5. The trim points are the endpoints of an x% bootstrap confidence interval.

Figure 2-9 shows a 90% confidence interval for the mean annual income of loan applicants, based on a sample of 20 for which the mean was $55,734 (this is the mean of the subset of 20 records and not the mean of the bootstrap analysis, $55,836).

The bootstrap is a general tool that can be used to generate confidence intervals for most statistics, or model parameters. Statistical textbooks and software, with roots in over a half-century of computerless statistical analysis, will also reference confidence intervals generated by formulas, especially the t-distribution (see "Student's t-Distribution").

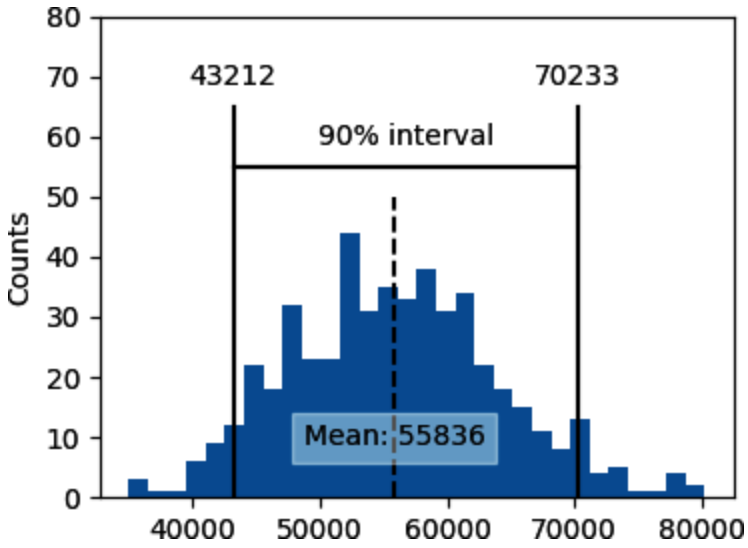


Figure 2-9. Bootstrap confidence interval for the annual income of loan applicants, based on a sample of 20

A SUBTLE POINT ABOUT CONFIDENCE INTERVALS

Of course, what we are really interested in when we have a sample result is, "What is the probability that the true value lies within a certain interval?" This is not really the question that a confidence interval answers, but it ends up being how most people interpret the answer.

The probability question associated with a confidence interval starts out with the phrase "Given a sampling procedure and a population, what is the probability that…?" To go in the opposite direction, "Given a sample result, what is the probability that (something is true about the population)?" involves more complex calculations and deeper imponderables.

The percentage associated with the confidence interval is termed the *level of confidence*. The higher the level of confidence, the wider the interval. Also, the smaller the sample, the wider the interval (i.e., the greater the uncertainty). Both make sense: the more confident you want to be, and the less data you have, the wider you must make the confidence interval to be sufficiently assured of capturing the true value.

CONFIDENCE INTERVALS ARE A COMMUNICATION TOOL

For a data scientist, a confidence interval is a tool that can be used to get an idea of how variable a sample result might be. Data scientists would use this information not to publish a scholarly paper or submit a result to a regulatory agency (as a researcher might) but most likely to communicate the potential error in an estimate, and perhaps to learn whether a larger sample is needed.

KEY IDEAS

- Confidence intervals are the typical way to present estimates as an interval range.
- The more data you have, the less variable a sample estimate will be.
- The lower the level of confidence you can tolerate, the narrower the confidence interval will be.
- The bootstrap is an effective way to construct confidence intervals.

Further Reading

- For a bootstrap approach to confidence intervals, see *Statistics for Data Science and Analytics* by Bruce, Gedeck, and Dobbins (Wiley, 2025) or *Statistics: Unlocking the Power of Data*, 2nd ed., by Robin Lock and four other Lock family members (Wiley, 2016).
- Engineers, who have a need to understand the precision of their measurements, use confidence intervals perhaps more than most disciplines, and *Modern Engineering Statistics* by Thomas Ryan (Wiley, 2007) discusses confidence intervals. It also reviews a tool that is just as useful and gets less attention: *prediction intervals*

(intervals around a single value, as opposed to a mean or other summary statistic).

Normal Distribution

The bell-shaped normal distribution is iconic in traditional statistics.[1] The fact that distributions of sample statistics are often normally shaped has made it a powerful tool in the development of mathematical formulas that approximate those distributions.

KEY TERMS FOR NORMAL DISTRIBUTION

Error

The difference between a data point and a predicted or average value

Standardize

Subtract the mean and divide by the standard deviation

z-score

The result of standardizing an individual data point

Standard normal

A normal distribution with mean = 0 and standard deviation = 1

QQ-plot

A plot to visualize how close a sample distribution is to a specified distribution, e.g., the normal distribution

In a normal distribution (Figure 2-10), 68% of the data lies within one standard deviation of the mean, and 95% lies within two standard

deviations.

MOST DATA ARE NOT NORMAL

It is a common misconception that the normal distribution is called that because most data follows a normal distribution—that is, it is the normal thing. Most of the variables used in a typical data science project—in fact, most raw data as a whole—is *not* normally distributed: see "Long-Tailed Distributions". The utility of the normal distribution derives from the fact that many statistics *are* normally distributed in their sampling distribution. Even so, assumptions of normality are generally a last resort, used when empirical probability distributions, or bootstrap distributions, are not available.

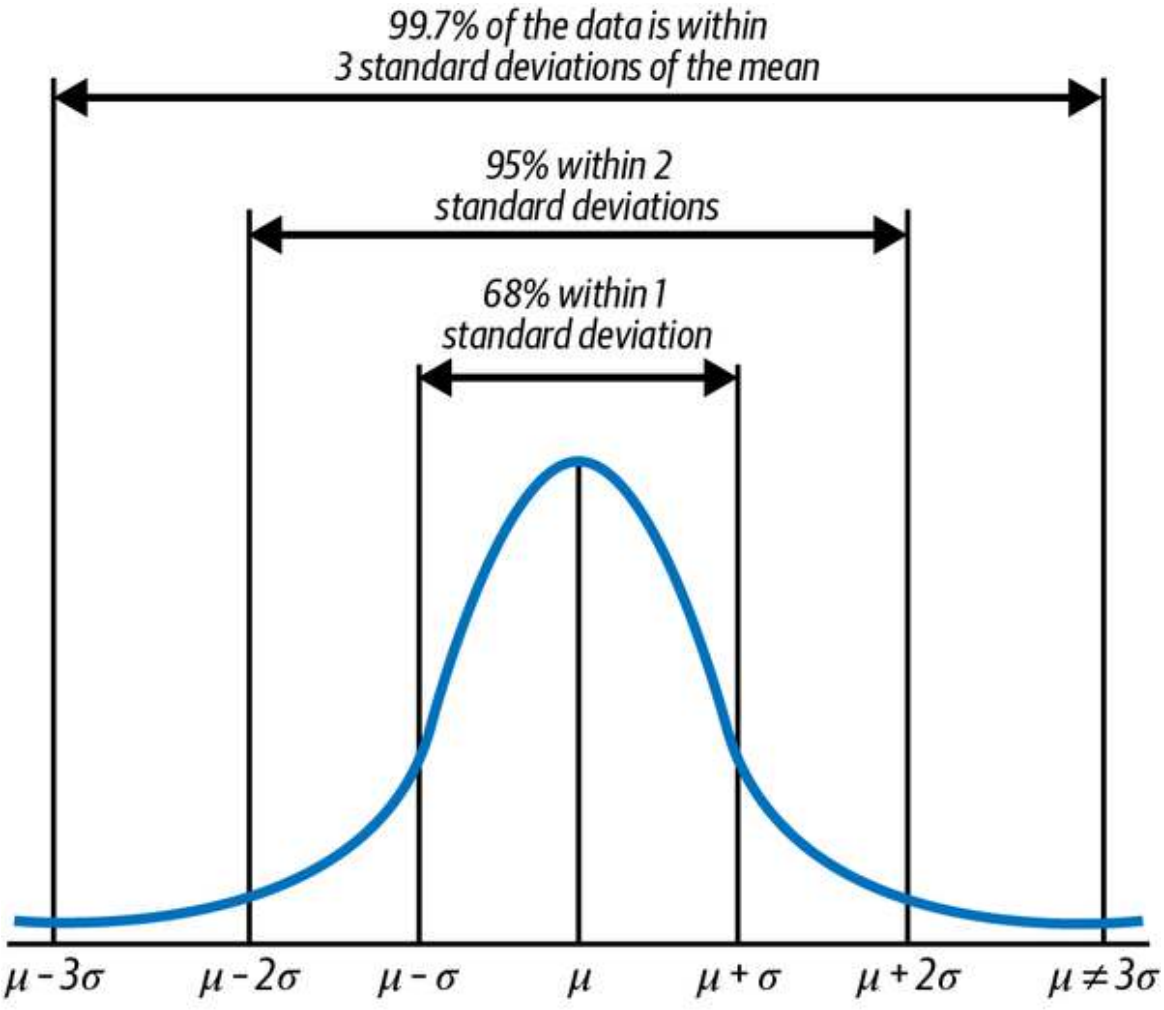


Figure 2-10. Normal curve

THE GAUSSIAN DISTRIBUTION

The normal distribution is also referred to as a *Gaussian* distribution after Carl Friedrich Gauss, a prodigious German mathematician from the late 18th and early 19th centuries. Another name previously used for the normal distribution was the "error" distribution. Statistically speaking, an *error* is the difference between an actual value and a statistical estimate like the sample mean. For example, the standard deviation (see "Estimates of Variability") is based on the errors from the mean of the data. Gauss's development of the normal distribution came from his study of the errors of astronomical measurements that were found to be normally distributed.

Standard Normal and QQ-Plots

A *standard normal* distribution is one in which the units on the x-axis are expressed in terms of standard deviations away from the mean. To compare data to a standard normal distribution, you subtract the mean and then divide by the standard deviation; this is also called *normalization* or *standardization* (see "Standardization (Normalization, z-Scores)"). Note that "standardization" in this sense is unrelated to database record standardization (conversion to a common format). The transformed value is termed a *z-score*, and the normal distribution is sometimes called the *z-distribution*.

A *QQ-plot* is used to visually determine how close a sample is to a specified distribution—in this case, the normal distribution. The QQ-plot orders the *z*-scores from low to high and plots each value's *z*-score on the y-axis; the x-axis is the corresponding quantile of a normal distribution for that value's rank. Since the data is normalized, the units correspond to the number of standard deviations away from the mean. If the points roughly fall on the diagonal line, then the sample distribution can be considered close to normal. Figure 2-11 shows a QQ-plot for a sample of 100 values randomly generated from a normal distribution; as expected, the points closely follow the line. This figure can be produced in R with the `geom_qq` function:

```
norm_samp <- rnorm(100)
ggplot(tibble(y = norm_samp), aes(sample = y)) +
```

```
geom_qq() +  ❶
geom_qq_line() +
labs(x = "Theoretical Quantiles", y = "Sample Quantiles")
```

❶ The `geom_qq` function uses the normal distribution (`stats::qnorm`) by default; other distributions can be specified with the `distribution` argument.

In Python, use the method `scipy.stats.probplot` to create the QQ-plot:

```
from scipy import stats

fig, ax = plt.subplots(figsize=(4, 4))
norm_sample = stats.norm.rvs(size=100)
stats.probplot(norm_sample, plot=ax)  ❶
plt.show()
```

❶ You can specify a different distribution with the `dist` argument; the default is the normal distribution (`stats.norm`).

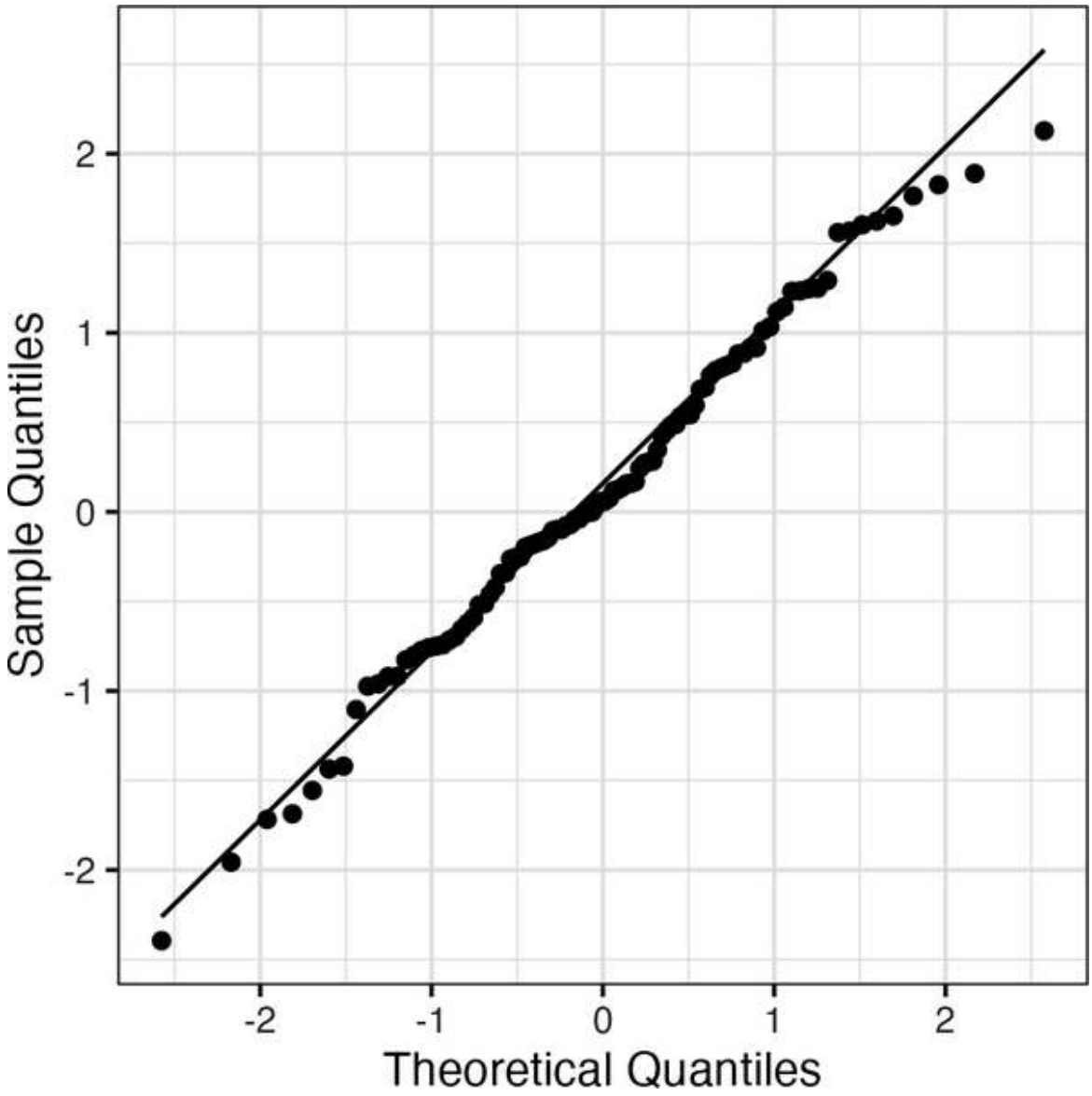


Figure 2-11. QQ-plot of a sample of 100 values drawn from a standard normal distribution

ABOUT Z-SCORES

Converting data to *z*-scores (i.e., standardizing or normalizing the data) does *not* make the data normally distributed. It just puts the data on the same scale as the standard normal distribution, often for comparison purposes.

KEY IDEAS

- The normal distribution was essential to the historical development of statistics, as it permitted mathematical approximation of uncertainty and variability.
- While raw data is typically not normally distributed, errors often are, as are averages and totals in large samples.
- To convert data to *z*-scores, you subtract the mean of the data and divide by the standard deviation; you can then compare the data to a normal distribution.

Long-Tailed Distributions

Despite the importance of the normal distribution historically in statistics, and in contrast to what the name would suggest, data is generally not normally distributed.

KEY TERMS FOR LONG-TAILED DISTRIBUTIONS

Tail

The long, narrow portion of a frequency distribution, where relatively extreme values occur at low frequency

Skew

Where one tail of a distribution is longer than the other

While the normal distribution is often appropriate and useful with respect to the distribution of errors and sample statistics, it typically does not characterize the distribution of raw data. Sometimes, the distribution is

highly *skewed* (asymmetric), such as with income data, or the distribution can be discrete, as with binomial data. Both symmetric and asymmetric distributions may have *long tails*. The tails of a distribution correspond to the extreme values (small and large). Long tails, and guarding against them, are widely recognized in practical work. Nassim Taleb has proposed the *black swan* theory, which predicts that anomalous events, such as a stock market crash, are much more likely to occur than would be predicted by the normal distribution.

A good example to illustrate the long-tailed nature of data is stock returns. Figure 2-12 shows the QQ-plot for the daily stock returns for Netflix (NFLX). This is generated in R by:

```
sp500_px <- read_csv("sp500_data.csv.gz", show_col_types = FALSE)

nflx <- sp500_px[, "NFLX"]
nflx <- diff(log(nflx[nflx > 0]))
ggplot(tibble(y = nflx), aes(sample = y)) +
  geom_qq() +
  geom_qq_line() +
  labs(x = "Theoretical Quantiles", y = "Sample Quantiles")
```

The corresponding Python code is:

```
import numpy as np

sp500_px = pd.read_csv("sp500_data.csv.gz")

nflx = sp500_px.NFLX
nflx = np.diff(np.log(nflx[nflx > 0]))
fig, ax = plt.subplots(figsize=(4, 4))
stats.probplot(nflx, plot=ax)
plt.show()
```

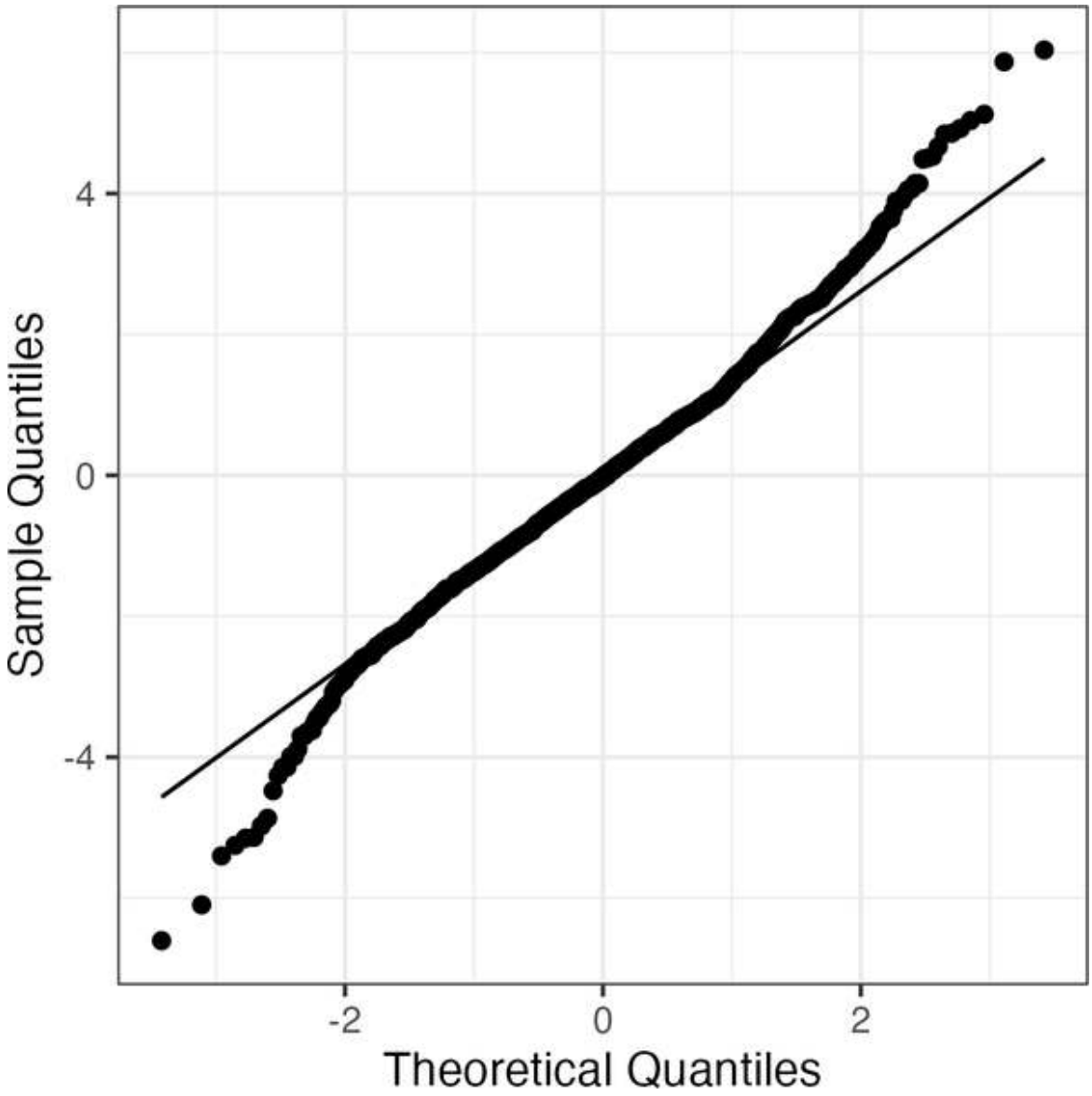


Figure 2-12. QQ-plot of the returns for Netflix (NFLX)

In contrast to Figure 2-11, the points are far below the line for low values and far above the line for high values, indicating the data is not normally distributed. This means that we are much more likely to observe extreme values than would be expected if the data had a normal distribution. Figure 2-12 shows another common phenomenon: the points are close to the line for the data within one standard deviation of the mean. Tukey refers to this phenomenon as data being "normal in the middle" but having much longer tails.

FITTING DISTRIBUTIONS TO DATA

There is much statistical literature about the task of fitting statistical distributions to observed data. Beware an excessively data-centric approach to this job, which is as much art as science. Data is variable, and often consistent, on its face, with more than one shape and type of distribution. It is typically the case that domain and statistical knowledge must be brought to bear to determine what type of distribution is appropriate to model a given situation. For example, we might have data on the level of internet traffic on a server over many consecutive five-second periods. It is useful to know that the best distribution to model "events per time period" is the Poisson (see "Poisson Distributions").

KEY IDEAS

- Most data is not normally distributed.
- Assuming a normal distribution can lead to underestimation of extreme events ("black swans").

Further Reading

- *The Black Swan*, 2nd ed., by Nassim Nicholas Taleb (Random House, 2010)
- *Handbook of Statistical Distributions with Applications*, 2nd ed., by K. Krishnamoorthy (Chapman & Hall/CRC Press, 2016)

Student's t-Distribution

The *t-distribution* is a normally shaped distribution, except that it is a bit thicker and longer on the tails. It is used extensively in depicting distributions of sample statistics. Distributions of sample means are typically shaped like a t-distribution, and there is a family of t-distributions that differ depending on how large the sample is. The larger the sample, the more normally shaped the t-distribution becomes.

KEY TERMS FOR STUDENT'S T-DISTRIBUTION

n

Sample size

Degrees of freedom

A parameter that allows the t-distribution to adjust to different sample sizes, statistics, and numbers of groups

The t-distribution is often called *Student's t* because it was published in 1908 in *Biometrika* by W. S. Gosset under the name "Student." Gosset's employer, the Guinness brewery, did not want competitors to know that it was using statistical methods, so it insisted that Gosset not use his name on the article.

Gosset wanted to answer the question "What is the sampling distribution of the mean of a sample, drawn from a larger population?" He started out with a resampling experiment—drawing random samples of 4 from a data set of 3,000 measurements of criminals' height and left-middle-finger length. (This being the era of eugenics, there was much interest in data on criminals, and in discovering correlations between criminal tendencies and physical or psychological attributes.) Gosset plotted the standardized results (the *z*-scores) on the x-axis and the frequency on the y-axis. Separately, he had derived a function, now known as *Student's t*, and he fit this function over the sample results, plotting the comparison (see Figure 2-13).

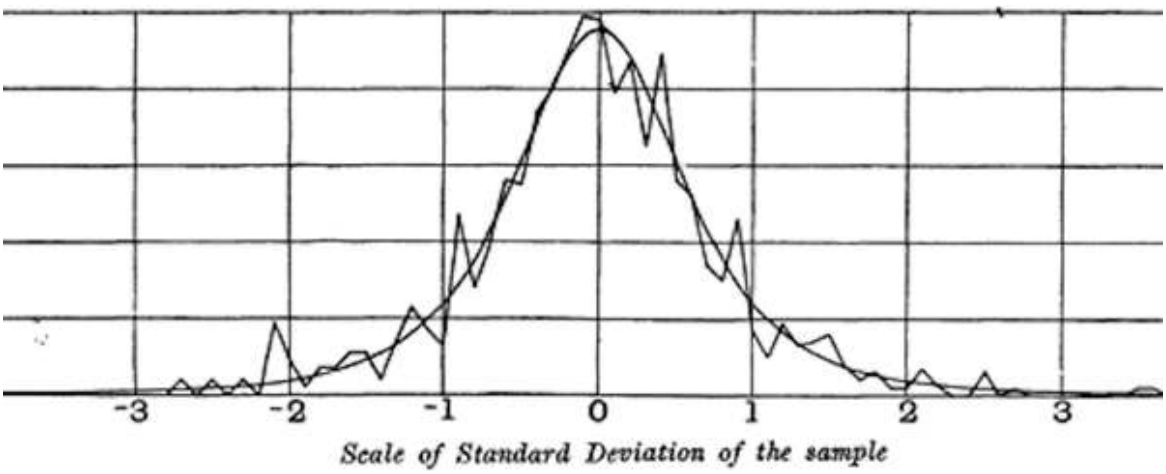


Figure 2-13. Gosset's resampling experiment results and fitted t-curve (from his 1908 Biometrika paper)

A number of different statistics can be compared, after standardization, to the t-distribution, to estimate confidence intervals in light of sampling variation. Consider a sample of size *n* for which the sample mean $\bar{x}$ has been calculated. If *s* is the sample standard deviation, a 90% confidence interval around the sample mean is given by:

$$\bar{x} \pm t_{n-1}(0.05) \cdot \frac{s}{\sqrt{n}}$$

where $t_{n-1}(0.05)$ is the value of the t-statistic, with (n – 1) degrees of freedom (see "Degrees of Freedom"), that "chops off" 5% of the t-distribution at either end. The t-distribution has been used as a reference for the distribution of a sample mean, the difference between two sample means, regression parameters, and other statistics.

Had computing power been widely available in 1908, statistics would no doubt have relied much more heavily on computationally intensive resampling methods from the start. Lacking computers, statisticians turned to mathematics and functions such as the t-distribution to approximate sampling distributions. Computer power enabled practical resampling experiments in the 1980s, but by then, use of the t-distribution and similar distributions had become deeply embedded in textbooks and software.

The t-distribution's accuracy in depicting the behavior of a sample statistic requires that the distribution of that statistic for that sample be shaped like a normal distribution. It turns out that sample statistics *are* often normally distributed, even when the underlying population data is not (a fact which led to widespread application of the t-distribution). This brings us back to the phenomenon known as the *central limit theorem* (see "Central Limit Theorem").

THE T-DISTRIBUTION

What do data scientists need to know about the t-distribution and the central limit theorem? Not a whole lot. The t-distribution is used in classical statistical inference but is not as central to the purposes of data science. Understanding and quantifying uncertainty and variation are important to data scientists, but empirical bootstrap sampling can answer most questions about sampling error. However, data scientists will routinely encounter t-statistics in output from statistical software and statistical procedures in R—for example, in A/B tests and regressions—so familiarity with its purpose is helpful.

KEY IDEAS

- The t-distribution is actually a family of distributions resembling the normal distribution but with thicker tails.
- The t-distribution is widely used as a reference basis for the distribution of sample means, differences between two sample means, regression parameters, and more.

Further Reading

- The original W. S. Gosset paper as published in *Biometrika* in 1908 is available as a PDF.
- A standard treatment of the t-distribution can be found in David Lane's online resource.

Binomial Distribution

Yes/no (binomial) outcomes lie at the heart of analytics since they are often the culmination of a decision or other process; buy/don't buy, click/don't click, survive/die, and so on. Central to understanding the binomial distribution is the idea of a set of *trials*, each trial having two possible outcomes with definite probabilities.

For example, flipping a coin 10 times is a binomial experiment with 10 trials, each trial having two possible outcomes (heads or tails); see Figure 2-14. Such yes/no or 0/1 outcomes are termed *binary* outcomes, and they need not have 50/50 probabilities. Any probabilities that sum to 1.0 are possible. It is conventional in statistics to term the "1" outcome the *success* outcome; it is also common practice to assign "1" to the more rare outcome. Use of the term *success* does not imply that the outcome is desirable or beneficial, but it does tend to indicate the outcome of interest. For example, loan defaults or fraudulent transactions are relatively uncommon events that we may be interested in predicting, so they are termed "1s" or "successes."



Figure 2-14. The tails side of a buffalo nickel

KEY TERMS FOR BINOMIAL DISTRIBUTION

Trial

An event with a discrete outcome (e.g., a coin flip)

Success

The outcome of interest for a trial

Synonym

1 (as opposed to 0)

Binomial

Having two outcomes

Synonyms

yes/no, 0/1, binary

Binomial trial

A trial with two outcomes

Synonym

Bernoulli trial

Binomial distribution

Distribution of number of successes in *n* trials

Synonym

Bernoulli distribution

The binomial distribution is the frequency distribution of the number of successes (x) in a given number of trials (n) with specified probability (p) of success in each trial. There is a family of binomial distributions, depending on the values of n and p. The binomial distribution would answer a question like:

> *If the probability of a click converting to a sale is 0.02, what is the probability of observing 0 sales in 200 clicks?*

The R function `dbinom` calculates binomial probabilities. For example:

```
dbinom(x = 2, size = 5, prob = 0.1)
```

would return 0.0729, the probability of observing exactly $x = 2$ successes in $size = 5$ trials, where the probability of success for each trial is $p = 0.1$. For our example above, we use $x = 0$, $size = 200$, and $p = 0.02$. With these arguments, `dbinom` returns a probability of 0.0176.

Often we are interested in determining the probability of x or fewer successes in n trials. In this case, we use the function `pbinom`:

```
pbinom(q = 2, size = 5, prob = 0.1)
```

This would return 0.9914, the probability of observing two or fewer successes in five trials, where the probability of success for each trial is 0.1.

The `scipy.stats` module implements a large variety of statistical distributions. For the binomial distribution, use the functions `stats.binom.pmf` and `stats.binom.cdf`:

```
print(stats.binom.pmf(2, n=5, p=0.1))
print(stats.binom.cdf(2, n=5, p=0.1))
```

The mean of a binomial distribution is $n \times p$; you can also think of this as the expected number of successes in n trials, for success probability = p.

The variance is $n \times p(1 - p)$. With a large enough number of trials (particularly when p is close to 0.50), the binomial distribution is virtually indistinguishable from the normal distribution. In fact, calculating binomial probabilities with large sample sizes is computationally demanding, and most statistical procedures use the normal distribution, with mean and variance, as an approximation.

KEY IDEAS

- Binomial outcomes are important to model, since they represent, among other things, fundamental decisions (buy or don't buy, click or don't click, survive or die, etc.).
- A binomial trial is an experiment with two possible outcomes: one with probability p and the other with probability $1 - p$.
- With large n, and provided p is not too close to 0 or 1, the binomial distribution can be approximated by the normal distribution.

Further Reading

- Read about the "quincunx", a pinball-like simulation device for illustrating the binomial distribution.
- The binomial distribution is a staple of introductory statistics, and all introductory statistics texts will have a chapter or two on it.

Chi-Square Distribution

An important idea in statistics is *departure from expectation*, especially with respect to category counts. Expectation is defined loosely as "nothing unusual or of note in the data" (e.g., no correlation between variables or predictable patterns). This is also termed the "null hypothesis" or "null

model" (see "The Null Hypothesis"). For example, you might want to test whether one variable (say, a row variable representing gender) is independent of another (say, a column variable representing "was promoted in job"), and you have counts of the number in each of the cells of the data table. The statistic that measures the extent to which results depart from the null expectation of independence is the chi-square statistic. It is the difference between the observed and expected values, divided by the square root of the expected value, squared, then summed across all categories. This process standardizes the statistic so it can be compared to a reference distribution. A more general way of putting this is to note that the chi-square statistic is a measure of the extent to which a set of observed values "fits" a specified distribution (a "goodness-of-fit" test). It is useful for determining whether multiple treatments (an "A/B/C… test") differ from one another in their effects.

The chi-square distribution is the distribution of the chi-square statistic under repeated resampled draws from the null model—see "Chi-Square Test" for a detailed algorithm, and the chi-square formula for a data table. A low chi-square value for a set of counts indicates that they closely follow the expected distribution. A high chi-square indicates that they differ markedly from what is expected. There are a variety of chi-square distributions associated with different degrees of freedom (e.g., number of observations—see "Degrees of Freedom").

KEY IDEAS

- The chi-square distribution is typically concerned with counts of subjects or items falling into categories.
- The chi-square statistic measures the extent of departure from what you would expect in a null model.

Further Reading

- The chi-square distribution owes its place in modern statistics to the great statistician Karl Pearson and the birth of hypothesis testing—read about this and more in David Salsburg's *The Lady Tasting Tea: How Statistics Revolutionized Science in the Twentieth Century* (W. H. Freeman, 2001).
- For a more detailed exposition, see the section in this book on the chi-square test ("Chi-Square Test").

F-Distribution

A common procedure in scientific experimentation is to test multiple treatments across groups—say, different fertilizers on different blocks of a field. This is similar to the A/B/C test referred to in the chi-square distribution (see "Chi-Square Distribution"), except we are dealing with measured continuous values rather than counts. In this case we are interested in the extent to which differences among group means are greater than we might expect under normal random variation. The F-statistic measures this and is the ratio of the variability among the group means to the variability within each group (also called residual variability). This comparison is termed an *analysis of variance* (see "ANOVA"). The distribution of the F-statistic is the frequency distribution of all the values that would be produced by randomly permuting data in which all the group means are equal (i.e., a null model). There are a variety of F-distributions associated with different degrees of freedom (e.g., numbers of groups—see "Degrees of Freedom"). The calculation of F is illustrated in the section on ANOVA. The F-statistic is also used in linear regression to compare the variation accounted for by the regression model to the overall variation in the data. F-statistics are produced automatically by R and Python as part of regression and ANOVA routines.

KEY IDEAS

- The F-distribution is used with experiments and linear models involving measured data.
- The F-statistic compares variation due to factors of interest to overall variation.

Further Reading

George Cobb's *Introduction to Design and Analysis of Experiments* (Wiley, 2008) contains an excellent exposition of the decomposition of variance components, which helps in understanding ANOVA and the F-statistic.

Poisson and Related Distributions

Many processes produce events randomly at a given overall rate—visitors arriving at a website, or cars arriving at a toll plaza (events spread over time); imperfections in a square meter of fabric, or typos per 100 lines of code (events spread over space).

KEY TERMS FOR POISSON AND RELATED DISTRIBUTIONS

Lambda

The rate (per unit of time or space) at which events occur

Poisson distribution

The frequency distribution of the number of events in sampled units of time or space

Exponential distribution

The frequency distribution of the time or distance from one event to the next event

Weibull distribution

A generalized version of the exponential distribution in which the event rate is allowed to shift over time

Poisson Distributions

From prior aggregate data (for example, number of flu infections per year), we can estimate the average number of events per unit of time or space (e.g., infections per day, or per census unit). We might also want to know how different this might be from one unit of time/space to another. The Poisson distribution tells us the distribution of events per unit of time or space when we sample many such units. It is useful when addressing queuing questions such as "How much capacity do we need to be 95% sure of fully processing the internet traffic that arrives on a server in any five-second period?"

The key parameter in a Poisson distribution is λ, or lambda. This is the mean number of events that occur in a specified interval of time or space.

The variance for a Poisson distribution is also λ.

A common technique is to generate random numbers from a Poisson distribution as part of a queuing simulation. The `rpois` function in R does this, taking only two arguments—the quantity of random numbers sought, and lambda:

```
rpois(n = 100, lambda = 2)
```

The corresponding `scipy` function is `stats.poisson.rvs`:

```
stats.poisson.rvs(2, size=100)
```

This code will generate 100 random numbers from a Poisson distribution with $\lambda = 2$. For example, if incoming customer service calls average two per minute, this code will simulate 100 minutes, returning the number of calls in each of those 100 minutes.

Exponential Distribution

Using the same parameter λ that we used in the Poisson distribution, we can also model the distribution of the time between events: time between visits to a website or between cars arriving at a toll plaza. It is also used in engineering to model time to failure, and in process management to model, for example, the time required per service call. The R code to generate random numbers from an exponential distribution takes two arguments: `n` (the quantity of numbers to be generated) and `rate` (the number of events per time period). For example:

```
rexp(n = 100, rate = 0.2)
```

The `scipy` implementation in `Python` specifies the exponential distribution using `scale` instead of rate. With scale being the inverse of rate, the corresponding command in Python is:

```
stats.expon.rvs(scale=1 / 0.2, size=100)
stats.expon.rvs(scale=5, size=100)
```

This code would generate 100 random numbers from an exponential distribution where the mean number of events per time period is 0.2. So you could use it to simulate 100 intervals, in minutes, between service calls, where the average rate of incoming calls is 0.2 per minute.

A key assumption in any simulation study for either the Poisson or exponential distribution is that the rate, λ, remains constant over the period being considered. This is rarely reasonable in a global sense; for example, traffic on roads or data networks varies by time of day and day of week. However, the time periods, or areas of space, can usually be divided into segments that are sufficiently homogeneous so that analysis or simulation within those periods is valid.

Estimating the Failure Rate

In many applications, the event rate, λ, is known or can be estimated from prior data. However, for rare events, this is not necessarily so. Aircraft engine failure, for example, is sufficiently rare (thankfully) that, for a given engine type, there may be little data on which to base an estimate of time between failures. With no data at all, there is little basis on which to estimate an event rate. However, you can make some guesses: if no events have been seen after 20 hours, you can be pretty sure that the rate is not 1 per hour. Via simulation, or direct calculation of probabilities, you can assess different hypothetical event rates and estimate threshold values below which the rate is very unlikely to fall. If there is some data but not enough to provide a precise, reliable estimate of the rate, a goodness-of-fit test (see “Chi-Square Test”) can be applied to various rates to determine how well they fit the observed data.

Weibull Distribution

In many cases, the event rate does not remain constant over time. If the period over which it changes is much longer than the typical interval

between events, there is no problem; you just subdivide the analysis into the segments where rates are relatively constant, as mentioned before. If, however, the event rate changes over the time of the interval, the exponential (or Poisson) distributions are no longer useful. This is likely to be the case in mechanical failure—the risk of failure increases as time goes by. The *Weibull* distribution is an extension of the exponential distribution in which the event rate is allowed to change, as specified by a *shape parameter*, β. If $\beta > 1$, the probability of an event increases over time; if $\beta < 1$, the probability decreases. Because the Weibull distribution is used with time-to-failure analysis instead of event rate, the second parameter is expressed in terms of characteristic life, rather than in terms of the rate of events per interval. The symbol used is η, the Greek letter eta. It is also called the *scale* parameter.

With the Weibull, the estimation task now includes estimation of both parameters, β and η. Software is used to model the data and yield an estimate of the best-fitting Weibull distribution.

The R code to generate random numbers from a Weibull distribution takes three arguments: `n` (the quantity of numbers to be generated), `shape`, and `scale`. For example, the following code would generate 100 random numbers (lifetimes) from a Weibull distribution with shape of 1.5 and characteristic life of 5,000:

```
rweibull(n = 100, shape = 1.5, scale = 5000)
```

To achieve the same in Python, use the function `stats.weibull_min.rvs`:

```
stats.weibull_min.rvs(1.5, scale=5000, size=100)
```

KEY IDEAS

- For events that occur at a constant rate, the number of events per unit of time or space can be modeled as a Poisson distribution.
- You can also model the time or distance between one event and the next as an exponential distribution.
- A changing event rate over time (e.g., an increasing probability of device failure) can be modeled with the Weibull distribution.

Further Reading

Modern Engineering Statistics by Thomas Ryan (Wiley, 2007) has a chapter devoted to the probability distributions used in engineering applications.

Summary

In the era of big data, the principles of random sampling remain important when accurate estimates are needed. Random selection of data can reduce bias and yield a higher-quality data set than would result from just using the conveniently available data. Knowledge of various sampling and data-generating distributions allows us to quantify potential errors in an estimate that might be due to random variation. At the same time, the bootstrap (sampling with replacement from an observed data set) is an attractive "one size fits all" method to determine possible error in sample estimates.

Exploration with AI

1. Random sampling is a gold standard for data collection. Pick an area with which you are familiar where sampling would be used, and have a conversation with AI about whether random sampling

is feasible and practical. Specify your purpose, and, if there are difficulties with random sampling, ask AI to suggest and explain alternatives. Make the conversation specific enough so that you could use it to write clear instructions for someone to implement, and also to justify to a statistician why you did not use random sampling, if that was the case.

2. Regression to the mean is a common phenomenon, but often not recognized. Ask AI to give you some examples in different fields.
3. Ask AI to locate a large public data set, help you write code in R or Python to select a random sample from it, and calculate basic statistics (mean, median, standard deviation) on the samples.
4. Continuing, ask AI to help you write code to generate bootstrap samples from your random sample, and to calculate confidence intervals for the statistics you calculated.
5. Continuing, ask AI to derive a confidence interval based on t-statistic calculations.
6. Using the LendingClub data, ask your AI to produce equally sized samples stratified by "home ownership" status.

---

[1] The bell curve is iconic but perhaps overrated. George W. Cobb, the Mount Holyoke statistician noted for his contribution to the philosophy of teaching introductory statistics, argued in a November 2015 editorial in the *American Statistician* that the "standard introductory course, which puts the normal distribution at its center, had outlived the usefulness of its centrality."

Chapter 3. Statistical Experiments and Significance Testing

Design of experiments is a cornerstone of the practice of statistics, with applications in virtually all areas of research. The goal is to design an experiment in order to confirm or reject a hypothesis. Data scientists often need to conduct continual experiments, particularly regarding user interface and product marketing. This chapter reviews traditional experimental design and discusses some common challenges in data science. It also covers some oft-cited concepts in statistical inference and explains their meaning and relevance (or lack of relevance) to data science.

Whenever you see references to statistical significance, t-tests, or p-values, it is typically in the context of the classical statistical inference "pipeline" (see Figure 3-1). This process starts with a hypothesis ("drug A is better than the existing standard drug," or "price A is more profitable than the existing price B"). An experiment (it might be an A/B test) is designed to test the hypothesis—designed in such a way that it hopefully will deliver conclusive results. The data is collected and analyzed, and then a conclusion is drawn. The term *inference* reflects the intention to apply the experiment results, which involve a limited set of data, to a larger process or population.

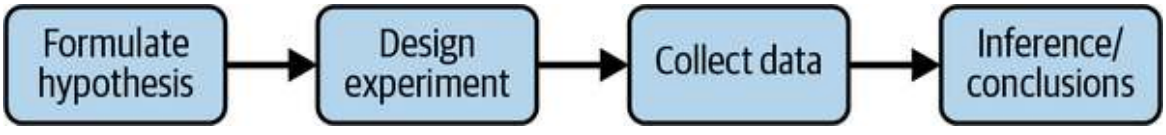


Figure 3-1. The classical statistical inference pipeline

A/B Testing

An A/B test is an experiment with two groups to establish which of two treatments, products, procedures, or the like is superior. Often one of the two treatments is the standard existing treatment, or no treatment. If a standard (or no) treatment is used, it is called the *control*. A typical hypothesis is that a new treatment is better than the control.

KEY TERMS FOR A/B TESTING

Treatment

Something (drug, price, web headline) to which a subject is exposed

Treatment group

A group of subjects exposed to a specific treatment

Control group

A group of subjects exposed to no (or standard) treatment

Randomization

The process of randomly assigning subjects to treatments

Subjects

The items (web visitors, patients, etc.) that are exposed to treatments

Test statistic

The metric used to measure the effect of the treatment

A/B tests are common in web design and marketing, since results are so readily measured. Some examples of A/B testing include:

- Testing two soil treatments to determine which produces better seed germination

- Testing two therapies to determine which suppresses cancer more effectively
- Testing two prices to determine which yields more net profit
- Testing two web headlines to determine which produces more clicks (Figure 3-2)
- Testing two web ads to determine which generates more conversions



Figure 3-2. Marketers continually test one web presentation against another

A proper A/B test has *subjects* that can be assigned to one treatment or another. The subject might be a person, a plant seed, a web visitor; the key is that the subject is exposed to the treatment. Ideally, subjects are *randomized* (assigned randomly) to treatments. In this way, you know that any difference between the treatment groups is due to one of two things:

- The effect of the different treatments
- Luck of the draw in which subjects are assigned to which treatments (i.e., the random assignment may have resulted in the naturally better-performing subjects being concentrated in A or B)

You also need to pay attention to the *test statistic* or metric you use to compare group A to group B. Perhaps the most common metric in data science is a binary variable: click or no-click, buy or don't buy, fraud or no fraud, and so on. Those results would be summed up in a 2 × 2 table. Table 3-1 is a 2 × 2 table for an actual price test (see "Statistical Significance and p-Values" for further discussion of these results).

Table 3-1. 2 × 2 table for ecommerce experiment results

Outcome	Price A	Price B
Conversion	200	182
No conversion	23,539	22,406

If the metric is a continuous variable (purchase amount, profit, etc.) or a count (e.g., days in hospital, pages visited), the result might be displayed differently. If one were interested not in conversion but in revenue per page view, the results of the price test in Table 3-1 might look like this in typical default software output:

Revenue/page view with price A: mean = 3.87, SD = 51.10

Revenue/page view with price B: mean = 4.11, SD = 62.98

"SD" refers to the standard deviation of the values within each group.

NOT ALL OUTPUT IS USEFUL

Just because statistical software—including R and Python—generates output by default does not mean that all the output is useful or relevant. You can see that the preceding standard deviations are not that useful; on their face they suggest that numerous values might be negative, when negative revenue is not feasible. This data consists of a small set of relatively high values (page views with conversions) and a huge number of 0-values (page views with no conversion). It is difficult to sum up the variability of such data with a single number, though the mean absolute deviation from the mean (7.68 for A and 8.15 for B) is more reasonable than the standard deviation.

Why Have a Control Group?

Why not skip the control group and just run an experiment applying the treatment of interest to only one group, and compare the outcome to prior experience?

Without a control group, there is no assurance that "all other things are equal" and that any difference is really due to the treatment (or to chance). When you have a control group, it is subject to the same conditions (except for the treatment of interest) as the treatment group. If you simply make a comparison to "baseline" or prior experience, other factors, besides the treatment, might differ.

BLINDING IN STUDIES

A *blind study* is one in which the subjects are unaware of whether they are getting treatment A or treatment B. Awareness of receiving a particular treatment can affect response. A *double-blind* study is one in which the investigators and facilitators (e.g., doctors and nurses in a medical study) also are unaware of which subjects are getting which treatment. Blinding is not possible when the nature of the treatment is transparent —for example, cognitive therapy from a computer versus a psychologist.

A/B testing in data science is typically used in a web context. Treatments might be the design of a web page, the price of a product, the wording of a headline, or some other item. Some thought is required to preserve the principles of randomization. Typically the subject in the experiment is the web visitor, and the outcomes we are interested in measuring are clicks, purchases, visit duration, number of pages visited, whether a particular page is visited, and the like. In a standard A/B experiment, you need to decide on one metric ahead of time. Multiple behavior metrics might be collected and be of interest, but if the experiment is expected to lead to a decision between treatment A and treatment B, a single metric, or *test statistic*, needs to be established beforehand. Selecting a test statistic *after* the experiment is conducted opens the door to researcher bias.

GETTING PERMISSION

In scientific and medical research involving human subjects, it is typically necessary to get their permission, as well as obtain the approval of an institutional review board. Experiments in business that are done as a part of ongoing operations almost never do this. In most cases (e.g., pricing experiments, or experiments about which headline to show or which offer should be made), this practice is widely accepted. Facebook, however, ran afoul of this general acceptance in 2014 when it experimented with the emotional tone in users' newsfeeds. Facebook used sentiment analysis to classify newsfeed posts as positive or negative, and then altered the positive/negative balance in what it showed users. Some randomly selected users experienced more positive posts, while others experienced more negative posts. Facebook found that the users who experienced a more positive newsfeed were more likely to post positively themselves, and vice versa. The magnitude of the effect was small, however, and Facebook faced much criticism for conducting the experiment without users' knowledge. Some users speculated that Facebook might have pushed some extremely depressed users over the edge if they got the negative version of their feed.

Why Just A/B? Why Not C, D,…?

A/B tests are popular in the marketing and ecommerce worlds, but are far from the only type of statistical experiment. Additional treatments can be included. Subjects might have repeated measurements taken. Pharmaceutical trials where subjects are scarce, expensive, and acquired over time are sometimes designed with multiple opportunities to stop the experiment and reach a conclusion.

Traditional statistical experimental designs focus on answering a static question about the efficacy of specified treatments. Data scientists are less interested in the question:

Is the difference between price A and price B statistically significant?

than in the question:

Which, out of multiple possible prices, is best?

For this, a relatively new type of experimental design is used: the *multi-arm bandit* (see "Multi-Arm Bandit Algorithm").

KEY IDEAS

- Subjects are assigned to two (or more) groups that are treated exactly alike, except that the treatment under study differs from one group to another.
- Ideally, subjects are assigned randomly to the groups.

Further Reading

- Two-group comparisons (A/B tests) are a staple of traditional statistics, and just about any introductory statistics text will have extensive coverage of design principles and inference procedures. For a discussion that places A/B tests in more of a data science

context and uses resampling, see *Introductory Statistics and Analytics: A Resampling Perspective* by Peter Bruce (Wiley, 2014).

- For web testing, the logistical aspects of testing can be just as challenging as the statistical ones. Third-party tools must be integrated into the website.
- Beware advice found in the ubiquitous guides to A/B testing that you see on the web, such as these words in one such guide: "Wait for about 1,000 total visitors and make sure you run the test for a week." Such general rules of thumb are not statistically meaningful; see "Power and Sample Size" for more detail.

Hypothesis Tests

Hypothesis tests, also called *significance tests*, are ubiquitous in the traditional statistical analysis of published research. Their purpose is to help you learn whether random chance might be responsible for an observed effect.

KEY TERMS FOR HYPOTHESIS TESTS

Null hypothesis

The hypothesis that chance is to blame

Alternative hypothesis

Counterpoint to the null (what you hope to prove)

One-way test

Hypothesis test that counts chance results only in one direction

Two-way test

Hypothesis test that counts chance results in two directions

An A/B test (see "A/B Testing") is typically constructed with a hypothesis in mind. For example, the hypothesis might be that price B produces higher profit. Why do we need a hypothesis? Why not just look at the outcome of the experiment and go with whichever treatment does better?

The answer lies in the tendency of the human mind to underestimate the scope of natural random behavior. One manifestation of this is the failure to anticipate extreme events, or so-called "black swans" (see "Long-Tailed Distributions"). Another manifestation is the tendency to misinterpret random events as having patterns of some significance. Statistical hypothesis testing was invented as a way to protect researchers from being fooled by random chance.

MISINTERPRETING RANDOMNESS

You can observe the human tendency to underestimate randomness in this experiment. Ask several friends to invent a series of 50 coin flips: have them write down a series of random H's and T's. Then ask them to actually flip a coin 50 times and write down the results. Have them put the real coin flip results in one pile, and the made-up results in another. It is easy to tell which results are real: the real ones will have longer runs of H's or T's. In a set of 50 *real* coin flips, it is not at all unusual to see five or six H's or T's in a row. However, when most of us are inventing random coin flips and we have gotten three or four H's in a row, we tell ourselves that, for the series to look random, we had better switch to T.

The other side of this coin, so to speak, is that when we *do* see the real-world equivalent of six H's in a row (e.g., when one headline outperforms another by 10%), we are inclined to attribute it to something real, not just to chance.

In a properly designed A/B test, you collect data on treatments A and B in such a way that any observed difference between A and B must be due to either:

- Random chance in assignment of subjects
- A true difference between A and B

A statistical hypothesis test is further analysis of an A/B test, or any randomized experiment, to assess whether random chance is a reasonable explanation for the observed difference between groups A and B.

The Null Hypothesis

Hypothesis tests use the following logic: "Given the human tendency to react to unusual but random behavior and interpret it as something meaningful and real, in our experiments we will require proof that the

difference between groups is more extreme than what chance might reasonably produce." This involves a baseline assumption that the treatments are equivalent, and any difference between the groups is due to chance. This baseline assumption is termed the *null hypothesis*. Our hope, then, is that we can in fact prove the null hypothesis *wrong* and show that the outcomes for groups A and B are more different than what chance might produce.

One way to do this is via a resampling permutation procedure, in which we shuffle together the results from groups A and B and then repeatedly deal out the data in groups of similar sizes, and then observe how often we get a difference as extreme as the observed difference. The combined shuffled results from groups A and B, and the procedure of resampling from them, embody the null hypothesis of groups A and B being equivalent and interchangeable and are termed the null model. See "Resampling" for more details.

Alternative Hypothesis

Hypothesis tests by their nature involve not just a null hypothesis but also an offsetting alternative hypothesis. Here are some examples:

- Null = "no difference between the means of group A and group B"; alternative = "A is different from B" (could be bigger or smaller)
- Null = "$A \leq B$"; alternative = "$A > B$"
- Null = "B is not X% greater than A"; alternative = "B is X% greater than A"

Taken together, the null and alternative hypotheses must account for all possibilities. The nature of the null hypothesis determines the structure of the hypothesis test.

One-Way Versus Two-Way Hypothesis Tests

Often in an A/B test, you are testing a new option (say, B) against an established default option (A), and the presumption is that you will stick with the default option unless the new option proves itself definitively better. In such a case, you want a hypothesis test to protect you from being fooled by chance in the direction favoring B. You don't care about being fooled by chance in the other direction, because you would be sticking with A unless B proves definitively better. So you want a *directional* alternative hypothesis (B is better than A). In such a case, you use a *one-way* (or one-tail) hypothesis test. This means that extreme chance results in only one direction count toward the p-value.

If you want a hypothesis test to protect you from being fooled by chance in either direction, the alternative hypothesis is *bidirectional* (A is different from B; could be bigger or smaller). In such a case, you use a *two-way* (or two-tail) hypothesis. This means that extreme chance results in either direction count toward the p-value.

A one-tail hypothesis test often fits the nature of A/B decision making, in which a decision is required and one option is typically assigned "default" status unless the other proves better. Software, however, including R and `scipy` in Python, typically provides a two-tail test in its default output, and many statisticians opt for the more conservative two-tail test just to avoid argument. One-tail versus two-tail is a confusing subject, and not that relevant for data science, where the precision of p-value calculations is not terribly important.

KEY IDEAS

- A null hypothesis is a logical construct embodying the notion that nothing special has happened, and any effect you observe is due to random chance.
- The hypothesis test assumes that the null hypothesis is true, creates a "null model" (a probability model), and tests whether the effect you observe is a reasonable outcome of that model.

Further Reading

- *The Drunkard's Walk* by Leonard Mlodinow (Pantheon, 2008) is a readable survey of the ways in which "randomness rules our lives."
- David Freedman, Robert Pisani, and Roger Purves's classic statistics text *Statistics*, 4th ed. (W. W. Norton, 2007), has excellent nonmathematical treatments of most statistics topics, including hypothesis testing.
- *Introductory Statistics and Analytics: A Resampling Perspective* by Peter Bruce (Wiley, 2014) and *Statistics for Data Science and Analytics* by Peter C. Bruce, Peter Gedeck, and Janet Dobbins (Wiley, 2025) develop hypothesis testing concepts using resampling.

Resampling

Resampling in statistics means to repeatedly sample values from observed data, with a general goal of assessing random variability in a statistic. It can also be used to assess and improve the accuracy of some machine-learning models (e.g., the predictions from decision tree models built on multiple bootstrapped data sets can be averaged in a process known as *bagging*—see "Bagging and the Random Forest").

There are two main types of resampling procedures: the *bootstrap* and *permutation* tests. The bootstrap is used to assess the reliability of an estimate; it was discussed in "The Bootstrap". Permutation tests are used to test hypotheses, typically involving two or more groups, and we discuss those in this section.

KEY TERMS FOR RESAMPLING

Permutation test

The procedure of combining two or more samples together and randomly (or exhaustively) reallocating the observations to resamples

Synonyms

randomization test, random permutation test, exact test

Resampling

Drawing additional samples ("resamples") from an observed data set

With or without replacement

In sampling, whether or not an item is returned to the sample before the next draw

Permutation Test

In a *permutation* procedure, two or more samples are involved, typically the groups in an A/B or other hypothesis test. *Permute* means to change the order of a set of values. The first step in a *permutation test* of a hypothesis is to combine the results from groups A and B (and, if used, C, D,…). This is the logical embodiment of the null hypothesis that the treatments to

which the groups were exposed do not differ. We then test that hypothesis by randomly drawing groups from this combined set and seeing how much they differ from one another. The permutation procedure is:

1. Combine the results from the different groups into a single data set.
2. Shuffle the combined data and then randomly draw (without replacement) a resample of the same size as group A (clearly it will contain some data from the other groups).
3. From the remaining data, randomly draw (without replacement) a resample of the same size as group B.
4. Do the same for groups C, D, and so on. You have now collected one set of resamples that mirror the sizes of the original samples.
5. Whatever statistic or estimate was calculated for the original samples (e.g., difference in group proportions), calculate it now for the resamples, and record; this constitutes one permutation iteration.
6. Repeat the previous steps *R* times to yield a permutation distribution of the test statistic.

Now go back to the observed difference between groups and compare it to the set of permuted differences. If the observed difference lies well within the set of permuted differences, then we have not proven anything—the observed difference is within the range of what chance might produce. However, if the observed difference lies outside most of the permutation distribution, then we conclude that chance is *not* responsible. In technical terms, the difference is *statistically significant*. (See "Statistical Significance and p-Values".)

Example: Web Stickiness

A company selling a relatively high-value service wants to test which of two web presentations does a better selling job. Due to the high value of the service being sold, sales are infrequent and the sales cycle is lengthy; it

would take too long to accumulate enough sales to know which presentation is superior. So the company decides to measure the results with a proxy variable, using the detailed interior page that describes the service.

PROXY VARIABLES

A *proxy* variable is one that stands in for the true variable of interest, which may be unavailable, too costly, or too time-consuming to measure. In climate research, for example, the oxygen content of ancient ice cores is used as a proxy for temperature. It is useful to have at least *some* data on the true variable of interest, so the strength of its association with the proxy can be assessed.

One potential proxy variable for our company is the number of clicks on the detailed landing page. A better one is how long people spend on the page. It is reasonable to think that a web presentation (page) that holds people's attention longer will lead to more sales. Hence, our metric is average time on the page, comparing page A to page B.

Due to the fact that this is an interior, special-purpose page, it does not receive a huge number of visitors. Also note that Google Analytics, which is how we measure average time on page, cannot measure the time spent on the last page within a session. Google Analytics will set the time spent on the last page in a session to zero, unless the user interacts with the page (e.g., clicks or scrolls). This is also the case for single-page sessions. The data requires additional processing to take this into account. The result is a total of 36 times for the two different presentations: 21 for page A and 15 for page B. Using `ggplot`, we can visually compare the times using side-by-side boxplots (we call these page times `session_times` in the code, though that has a slightly different meaning for Google):

```
library(tidyverse)

session_times <- read_csv("web_page_data.csv", show_col_types = FALSE) %>%
  mutate(Time = Time * 100)

ggplot(session_times, aes(x = Page, y = Time)) +
```

```
    geom_boxplot() +
    labs(y = "Time (in seconds)")
```

The `pandas` `boxplot` command uses the keyword argument `by` to create the figure:

```
import matplotlib.pyplot as plt
import pandas as pd

session_times = pd.read_csv("web_page_data.csv")
session_times["Time"] *= 100

ax = session_times.boxplot(by="Page", column="Time")
ax.set_xlabel("")
ax.set_ylabel("Time (in seconds)")
plt.suptitle("")
```

The boxplot, shown in Figure 3-3, indicates that page B leads to longer sessions than page A. The means for each group can be computed in R as follows:

```
mean_a <- session_times %>% filter(Page == "Page A") %>% pull("Time") %>%
mean()
mean_b <- session_times %>% filter(Page == "Page B") %>% pull("Time") %>%
mean()
mean_b - mean_a
---
[1] 35.66667
```

In Python, we filter the `pandas` data frame first by page and then determine the mean of the `Time` column:

```
mean_a = session_times[session_times.Page == "Page A"].Time.mean()
mean_b = session_times[session_times.Page == "Page B"].Time.mean()
mean_b - mean_a
```

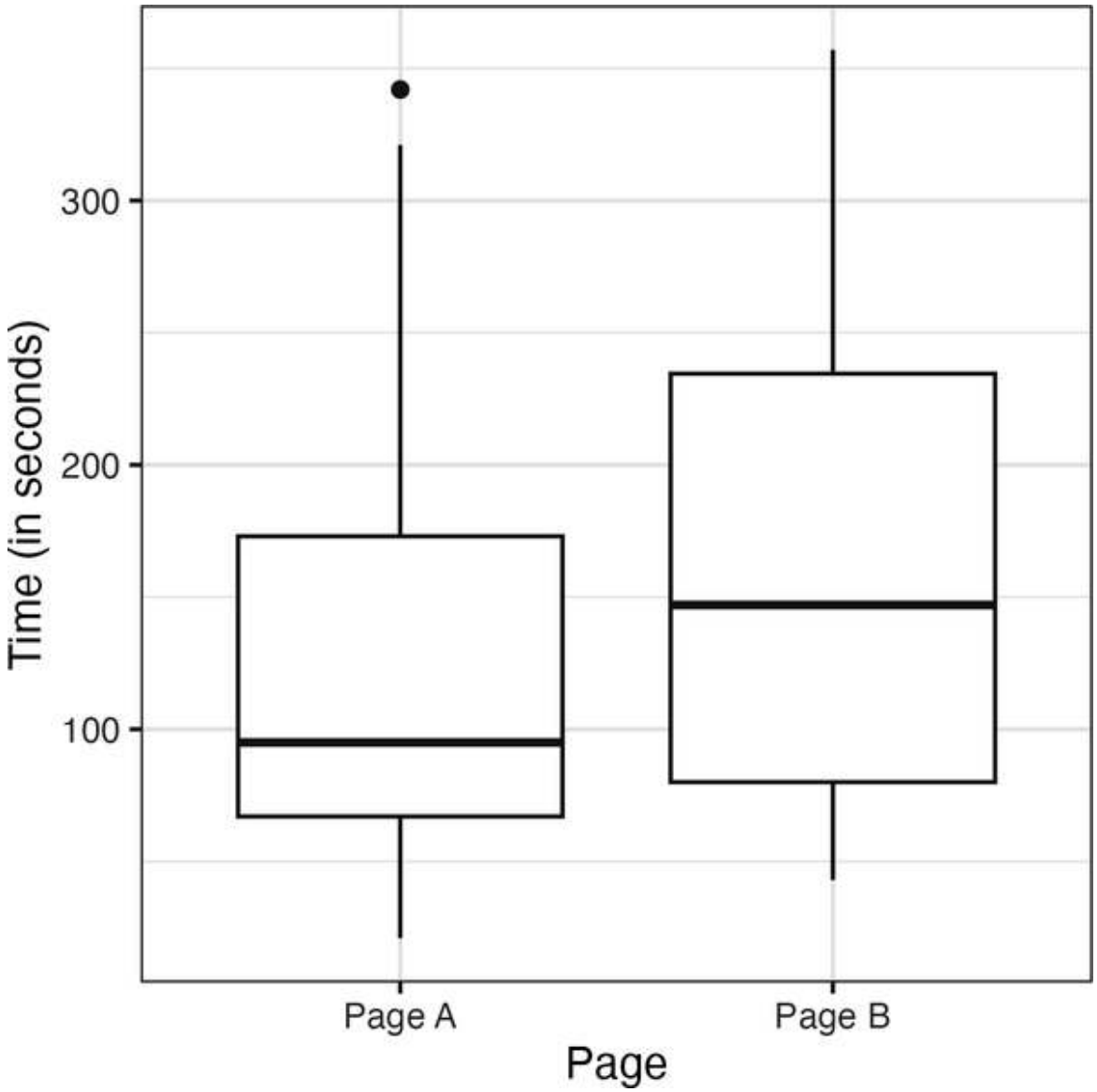


Figure 3-3. Session times for web pages A and B

Page B has times that are greater than those of page A by 35.67 seconds, on average. The question is whether this difference is within the range of what random chance might produce, i.e., is statistically significant. One way to answer this is to apply a permutation test—combine all the times together and then repeatedly shuffle and divide them into groups of 21 (recall that n_A = 21 for page A) and 15 (n_B = 15 for page B).

To apply a permutation test, we need a function to randomly assign the 36 times to a group of 21 (page A) and a group of 15 (page B). The R version of this function is:

```
perm_fun <- function(x, n_a) {
  n <- length(x)
  x_resampled <- sample(x)
  x_a <- x_resampled[1:n_a]
  x_b <- x_resampled[(n_a + 1):n]
  return(mean(x_b) - mean(x_a))
}
```

The Python version of this permutation test is the following:

```
import random

def perm_fun(x, n_a):
    x_resampled = np.random.permutation(x)
    x_a = x_resampled[:n_a]
    x_b = x_resampled[n_a:]
    return x_b.mean() - x_a.mean()
```

This function works by sampling (without replacement) n_B indices and assigning them to the B group; the remaining n_A indices are assigned to group A. The difference between the two means is returned. Calling this function $R = 1{,}000$ times and specifying $n_A = 21$ and $n_B = 15$ leads to a distribution of differences in the page times that can be plotted as a histogram. In R this is done as follows using the `hist` function:

```
set.seed(1)
perm_diffs <- rep(0, 1000)
for (i in 1:1000) {
  perm_diffs[i] <- perm_fun(session_times %>% pull(Time), 21)
}
ggplot(tibble(perm_diffs = perm_diffs), aes(x = perm_diffs)) +
  geom_histogram(bins = 12, color = "black", fill = "white") +
  geom_vline(xintercept = mean_b - mean_a, linetype = "dashed") +
  labs(x = "Session time differences (in seconds)", y = "Frequency")
```

In Python, we can create a similar graph using `matplotlib`:

```
n_a = session_times[session_times.Page == "Page A"].shape[0]
perm_diffs = [perm_fun(session_times.Time, n_a) for _ in range(1000)]

fig, ax = plt.subplots(figsize=(5, 5))
```

```
ax.hist(perm_diffs, bins=11, rwidth=0.9)
ax.axvline(x=mean_b - mean_a, color="black", lw=2)
ax.text(50, 190, "Observed\ndifference", bbox={"facecolor": "white"})
ax.set_xlabel("Session time differences (in seconds)")
ax.set_ylabel("Frequency")
```

The histogram in Figure 3-4 shows that the mean difference of random permutations often exceeds the observed difference in times (the vertical line). For our results, this happens in 14.4% of the cases:

```
mean(perm_diffs > (mean_b - mean_a))
---
0.144
```

As the simulation uses random numbers, the percentage will vary. For example, in the Python version, we got 12.9%:

```
import numpy as np

np.mean(perm_diffs > mean_b - mean_a)
---
0.129
```

This suggests that the observed difference in time between page A and page B is well within the range of chance variation and thus is not statistically significant.

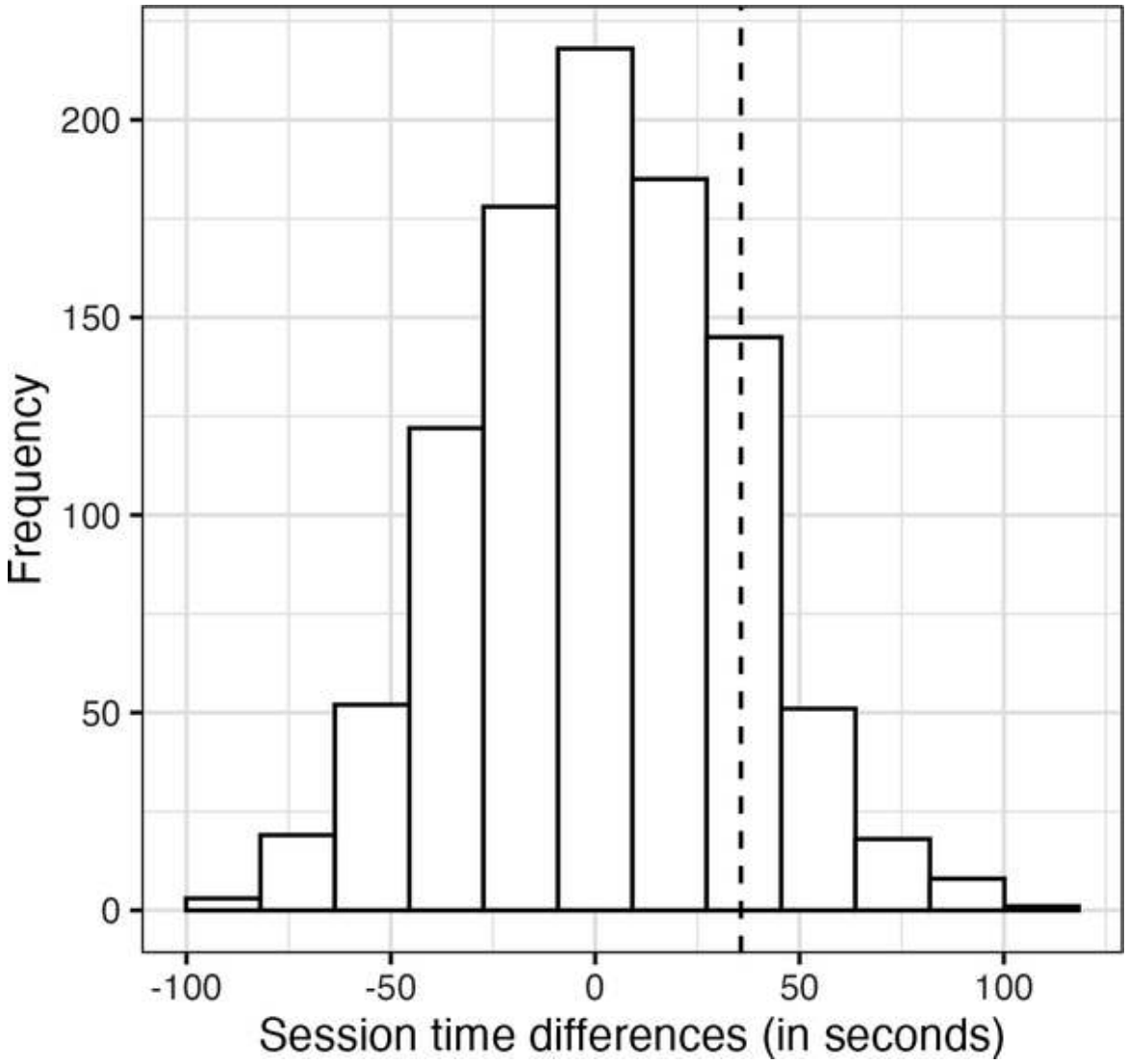


Figure 3-4. Frequency distribution for time differences between pages A and B; the vertical line shows the observed difference

Exhaustive and Bootstrap Permutation Tests

In addition to the preceding random shuffling procedure, also called a *random permutation test* or a *randomization test*, there are two variants of the permutation test:

- An *exhaustive permutation test*
- A *bootstrap permutation test*

In an exhaustive permutation test, instead of just randomly shuffling and dividing the data, we actually figure out all the possible ways it could be divided. This is practical only for relatively small sample sizes. With a large number of repeated shufflings, the random permutation test results approximate those of the exhaustive permutation test, and approach them in the limit. Exhaustive permutation tests are also sometimes called *exact tests*, due to their statistical property of guaranteeing that the null model will not test as "significant" more than the alpha level of the test (see "Statistical Significance and p-Values").

In a bootstrap permutation test, the draws outlined in steps 2 and 3 of the random permutation test are made *with replacement* instead of without replacement. In this way the resampling procedure models not just the random element in the assignment of treatment to subject but also the random element in the selection of subjects from a population. Both procedures are encountered in statistics, and the distinction between them is somewhat convoluted and not of consequence in the practice of data science.

Permutation Tests: The Bottom Line for Data Science

Permutation tests are useful heuristic procedures for exploring the role of random variation. They are relatively easy to code, interpret, and explain, and they offer a useful detour around the formalism and "false determinism" of formula-based statistics, in which the precision of formula "answers" tends to imply unwarranted certainty.

One virtue of resampling, in contrast to formula approaches, is that it comes much closer to a one-size-fits-all approach to inference. Data can be numeric or binary. Sample sizes can be the same or different. Assumptions about normally distributed data are not needed.

KEY IDEAS

- In a permutation test, multiple samples are combined and then shuffled.
- The shuffled values are then divided into resamples, and the statistic of interest is calculated.
- This process is then repeated, and the resampled statistic is tabulated.
- Comparing the observed value of the statistic to the resampled distribution allows you to judge whether an observed difference between samples might occur by chance.

Further Reading

- *Randomization Tests*, 4th ed., by Eugene Edgington and Patrick Onghena (Chapman & Hall/CRC Press, 2007)—but don't get too drawn into the thicket of nonrandom sampling.
- *Introductory Statistics and Analytics: A Resampling Perspective* by Peter Bruce (Wiley, 2014)
- *Statistics for Data Science and Analytics* by Peter C. Bruce, Peter Gedeck, and Janet Dobbins (Wiley, 2025)

Statistical Significance and p-Values

Statistical significance is how statisticians measure whether an experiment (or even a study of existing data) yields a result more extreme than what chance might produce. If the result is beyond the realm of chance variation, it is said to be statistically significant.

KEY TERMS FOR STATISTICAL SIGNIFICANCE AND P-VALUES

p-value

Given a chance model that embodies the null hypothesis, the p-value is the probability of obtaining results as unusual or extreme as the observed results.

Alpha

The probability threshold of "unusualness" that chance results must surpass for actual outcomes to be deemed statistically significant.

Type 1 error

Mistakenly concluding an effect is real (when it is due to chance).

Type 2 error

Mistakenly concluding an effect is due to chance (when it is real).

Consider in Table 3-2 the results of the web test shown earlier.

Table 3-2. 2 × 2 table for ecommerce experiment results

Outcome	Price A	Price B
Conversion	200	182
No conversion	23,539	22,406

Price A converts almost 5% better than price B (0.8425% = 200 / (23539 + 200) * 100, versus 0.8057% = 182 / (22406 + 182) * 100—a difference of

0.0368 percentage points), big enough to be meaningful in a high-volume business. We have over 45,000 data points here, and it is tempting to consider this as "big data," not requiring tests of statistical significance (needed mainly to account for sampling variability in small samples). However, the conversion rates are so low (less than 1%) that the actual meaningful values—the conversions—are only in the 100s, and the sample size needed is really determined by these conversions. We can test whether the difference in conversions between prices A and B is within the range of *chance variation*, using a resampling procedure. By chance variation, we mean the random variation produced by a probability model that embodies the null hypothesis that there is no difference between the rates (see "The Null Hypothesis"). The following permutation procedure asks, "If the two prices share the same conversion rate, could chance variation produce a difference as big as 5%?":

1. Put cards labeled 1 and 0 in a box: this represents the supposed shared conversion rate of 382 ones and 45,945 zeros = 0.008246 = 0.8246%.
2. Shuffle and draw out a resample of size 23,739 (same *n* as price A), and record how many 1s.
3. Record the number of 1s in the remaining 22,588 (same *n* as price B).
4. Record the difference in proportion of 1s.
5. Repeat steps 2–4.
6. How often was the difference >= 0.0368?

Reusing the function `perm_fun` defined in "Example: Web Stickiness", we can create a histogram of randomly permuted differences in conversion rate in R:

```
obs_pct_diff <- 100 * (200 / 23739 - 182 / 22588)
conversion <- c(rep(0, 45945), rep(1, 382))
perm_diffs <- rep(0, 1000)
```

```
for (i in 1:1000) {
  perm_diffs[i] <- 100 * perm_fun(conversion, 23739)
}
ggplot(tibble(perm_diffs = perm_diffs), aes(x = perm_diffs)) +
  geom_histogram(bins = 12, color = "black", fill = "white") +
  geom_vline(xintercept = obs_pct_diff, linetype = "dashed") +
  labs(x = "Conversion rate (percent)", y = "Frequency") +
  annotate("text", x = 0.1, y = 215, label = "Observed\ndifference", size = 3)
```

The corresponding Python code is:

```
obs_pct_diff = 100 * (200 / 23739 - 182 / 22588)
print(f"Observed difference: {obs_pct_diff:.4f}%")
conversion = [0] * 45945
conversion.extend([1] * 382)
conversion = pd.Series(conversion)

perm_diffs = [100 * perm_fun(conversion, 23739)
              for _ in range(1000)]

fig, ax = plt.subplots(figsize=(5, 5))
ax.hist(perm_diffs, bins=11, rwidth=0.9)
ax.axvline(x=obs_pct_diff, color="black", lw=2)
ax.text(0.06, 200, "Observed\ndifference", bbox={"facecolor": "white"})
ax.set_xlabel("Conversion rate (percent)")
ax.set_ylabel("Frequency")
```

See the histogram of 1,000 resampled results in Figure 3-5: as it happens, in this case the observed difference of 0.0368% is well within the range of chance variation.

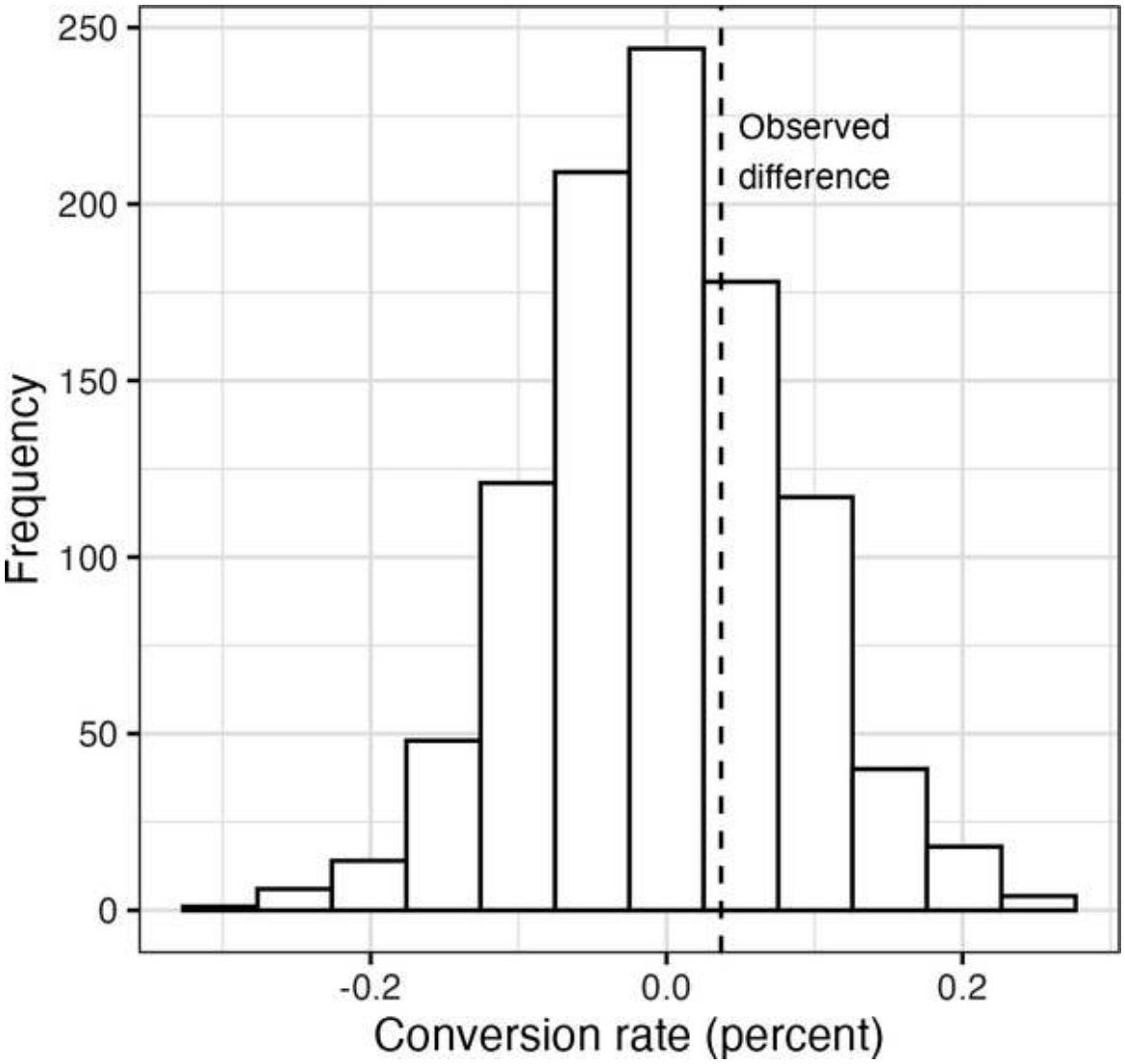


Figure 3-5. Frequency distribution for the difference in conversion rates between prices A and B

p-Value

Simply looking at the graph is not a very precise way to measure statistical significance, so of more interest is the *p-value*. This is the frequency with which the chance model produces a result more extreme than the observed result. We can estimate a p-value from our permutation test by taking the proportion of times that the permutation test produces a difference equal to or greater than the observed difference:

```
mean(perm_diffs > obs_pct_diff)
---
```

```
[1] 0.313
np.mean([diff > obs_pct_diff for diff in perm_diffs])
```

Here, both R and Python use the fact that true is interpreted as 1 and false as 0.

The p-value is 0.313, which means that we would expect to achieve a result as extreme as this, or a more extreme result, by random chance over 30% of the time.

In this case, we didn't need to use a permutation test to get a p-value. Since we have a binomial distribution, we can approximate the p-value. In R code, we do this using the function `prop.test`:

```
prop.test(x = c(200, 182), n = c(23739, 22588), alternative = "greater")
---
        2-sample test for equality of proportions with continuity correction

data:  c(200, 182) out of c(23739, 22588)
X-squared = 0.14893, df = 1, p-value = 0.3498
alternative hypothesis: greater
95 percent confidence interval:
 -0.001057439  1.000000000
sample estimates:
     prop 1      prop 2
0.008424955 0.008057376
```

The argument `x` is the number of successes for each group, and the argument `n` is the number of trials.

The method `scipy.stats.chi2_contingency` takes the values as shown in Table 3-2:

```
from scipy import stats

survivors = np.array([[200, 23739 - 200], [182, 22588 - 182]])
res = stats.chi2_contingency(survivors)

print(f"p-value for single sided test: {res.pvalue / 2:.4f}")
```

The normal approximation yields a p-value of 0.3498, which is close to the p-value obtained from the permutation test.

Alpha

Statisticians frown on the practice of leaving it to the researcher's discretion to determine whether a result is "too unusual" to happen by chance. Rather, a threshold is specified in advance, as in "more extreme than 5% of the chance (null hypothesis) results"; this threshold is known as *alpha*. Typical alpha levels are 5% and 1%. Any chosen level is an arbitrary decision—there is nothing about the process that will guarantee correct decisions *x*% of the time. This is because the probability question being answered is *not* "What is the probability that this happened by chance?" but rather, "Given a chance model, what is the probability of a result this extreme?" We then deduce backward about the appropriateness of the chance model, but that judgment does not carry a probability. This point has been the subject of much confusion.

p-value controversy

Considerable controversy has surrounded the use of the p-value in recent years. One psychology journal has gone so far as to "ban" the use of p-values in submitted papers on the grounds that publication decisions based solely on the p-value were resulting in the publication of poor research. Too many researchers, only dimly aware of what a p-value really means, root around in the data, and among different possible hypotheses to test, until they find a combination that yields a significant p-value and, hence, a paper suitable for publication.

The real problem is that people want more meaning from the p-value than it contains. Here's what we would *like* the p-value to convey:

> *The probability that the result is due to chance.*

We hope for a low value, so we can conclude that we've proved something. This is how many journal editors were interpreting the p-value. But here's what the p-value *actually* represents:

The probability that, given a chance model, results as extreme as the observed results could occur.

The difference is subtle but real. A significant p-value does not carry you quite as far along the road to "proof" as it seems to promise. The logical foundation for the conclusion "statistically significant" is somewhat weaker when the real meaning of the p-value is understood.

In March 2016, the American Statistical Association, after much internal deliberation, revealed the extent of misunderstanding about p-values when it issued a cautionary statement regarding their use. The ASA statement stressed six principles for researchers and journal editors:

1. *P-values can indicate how incompatible the data are with a specified statistical model.*
2. *P-values do not measure the probability that the studied hypothesis is true, or the probability that the data were produced by random chance alone.*
3. *Scientific conclusions and business or policy decisions should not be based only on whether a p-value passes a specific threshold.*
4. *Proper inference requires full reporting and transparency.*
5. *A p-value, or statistical significance, does not measure the size of an effect or the importance of a result.*
6. *By itself, a p-value does not provide a good measure of evidence regarding a model or hypothesis.*

Practical significance

Even if a result is statistically significant, that does not mean it has practical significance. A small difference that has no practical meaning can be statistically significant if it arose from large enough samples. Large samples ensure that small, nonmeaningful effects can nonetheless be big enough to

rule out chance as an explanation. Ruling out chance does not magically render important a result that is, in its essence, unimportant.

Conversely, a result that is not statistically significant may nonetheless be worth further attention. For example, in a test of a new web design, an increase in conversion rate that is not statistically significant may nonetheless be worth pursuing if the cost of implementing the new design is low and you are not too wedded to the existing design.

Type 1 and Type 2 Errors

In assessing statistical significance, two types of error are possible:

- A Type 1 error, in which you mistakenly conclude an effect is real, when it is really just due to chance
- A Type 2 error, in which you mistakenly conclude that an effect is not real (i.e., due to chance), when it actually is real

Actually, a Type 2 error is not so much an error as a judgment that the sample size is too small to detect the effect. When a p-value falls short of statistical significance (e.g., it exceeds 5%), what we are really saying is "effect not proven." It could be that a larger sample would yield a smaller p-value.

The basic function of significance tests (also called *hypothesis tests*) is to protect against being fooled by random chance; thus they are typically structured to minimize Type 1 errors.

Data Science and p-Values

The work that data scientists do is typically not destined for publication in scientific journals, so the debate over the value of a p-value is somewhat academic. For a data scientist, a p-value is a useful metric in situations where you want to know whether a model result that appears interesting and useful is within the range of normal chance variability. As a decision tool in an experiment, a p-value should not be considered controlling, but merely

another point of information bearing on a decision. For example, p-values are sometimes used as intermediate inputs in some statistical or machine learning models—a feature might be included in or excluded from a model depending on its p-value.

KEY IDEAS

- Significance tests are used to determine whether an observed effect is within the range of chance variation for a null hypothesis model.
- The p-value is the probability that results as extreme as the observed results might occur, given a null hypothesis model.
- The alpha value is the threshold of "unusualness" in a null hypothesis chance model.
- Significance testing has been much more relevant for formal reporting of research than for data science (but has been fading recently, even for the former).

Further Reading

- Stephen Stigler, "Fisher and the 5% Level," *Chance* 21, no. 4 (2008): 12. This article is a short commentary on Ronald Fisher's 1925 book *Statistical Methods for Research Workers* (Oliver & Boyd), and on Fisher's emphasis on the 5% level of significance.
- See also "Hypothesis Tests" and the further reading mentioned there.

t-Tests

There are numerous types of significance tests, depending on whether the data comprises count data or measured data, how many samples there are,

and what's being measured. A very common one is the *t-test*, named after Student's t-distribution, originally developed by W. S. Gosset to approximate the distribution of a single sample mean (see "Student's t-Distribution").

KEY TERMS FOR T-TESTS

Test statistic

A metric for the difference or effect of interest

t-statistic

A standardized version of common test statistics such as means

t-distribution

A reference distribution (in this case derived from the null hypothesis), to which the observed t-statistic can be compared

All significance tests require that you specify a *test statistic* to measure the effect you are interested in and help you determine whether that observed effect lies within the range of normal chance variation. In a resampling test (see the discussion of permutation in "Permutation Test"), the scale of the data does not matter. You create the reference (null hypothesis) distribution from the data itself and use the test statistic as is.

In the 1920s and 1930s, when statistical hypothesis testing was being developed, it was not feasible to randomly shuffle data thousands of times to do a resampling test. Statisticians found that a good approximation to the permutation (shuffled) distribution was the t-test, based on Gosset's t-distribution. It is used for the very common two-sample comparison—A/B test—in which the data is numeric. But in order for the t-distribution to be used without regard to scale, a standardized form of the test statistic must be used.

A classic statistics text would at this stage show various formulas that incorporate Gosset's distribution and demonstrate how to standardize your data to compare it to the standard t-distribution. These formulas are not shown here because all statistical software, as well as R and Python, includes commands that embody the formula. In R, the function is `t.test`:

```
t.test(Time ~ Page, data = session_times, alternative = "less")
---
        Welch Two Sample t-test

data:  Time by Page
t = -1.0983, df = 27.693, p-value = 0.1408
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
     -Inf 19.59674
sample estimates:
mean in group Page A mean in group Page B
            126.3333             162.0000
```

The function `scipy.stats.ttest_ind` can be used in Python:

```
res = stats.ttest_ind(session_times[session_times.Page == "Page A"].Time,
                      session_times[session_times.Page == "Page B"].Time,
                      equal_var=False)
print(f"p-value for single sided test: {res.pvalue / 2:.4f}")
```

The alternative hypothesis is that the page time mean for page A is less than that for page B. The p-value of 0.1408 is fairly close to the permutation test p-values of 0.121 and 0.126 (see "Example: Web Stickiness").

In a resampling mode, we structure the solution to reflect the observed data and the hypothesis to be tested, not worrying about whether the data is numeric or binary, whether or not sample sizes are balanced, sample variances, or a variety of other factors. In the formula world, many variations present themselves, and they can be bewildering. Statisticians need to navigate that world and learn its map, but data scientists do not—they are typically not in the business of sweating the details of hypothesis tests and confidence intervals the way a researcher preparing a paper for presentation might.

KEY IDEAS

- Before the advent of computers, resampling tests were not practical, and statisticians used standard reference distributions.
- A test statistic could then be standardized and compared to the reference distribution.
- One such widely used standardized statistic is the t-statistic.

Further Reading

- Any introductory statistics text will have illustrations of the t-statistic and its uses; two good ones are *Statistics*, 4th ed., by David Freedman, Robert Pisani, and Roger Purves (W. W. Norton, 2007), and *The Basic Practice of Statistics*, 8th ed., by David S. Moore, William I. Notz, and Michael A. Fligner (W. H. Freeman, 2017).
- For a treatment of both the t-test and resampling procedures in parallel, see *Introductory Statistics and Analytics: A Resampling Perspective* by Peter Bruce (Wiley, 2014), *Statistics for Data Science and Analytics* by Peter C. Bruce, Peter Gedeck, and Janet Dobbins (Wiley, 2025) or *Statistics: Unlocking the Power of Data*, 2nd ed., by Robin Lock and four other Lock family members (Wiley, 2016).

Multiple Testing

As we've mentioned previously, there is a saying in statistics: "Torture the data long enough, and it will confess." This means that if you look at the data through enough different perspectives and ask enough questions, you almost invariably will find a statistically significant effect.

For example, if you have 20 predictor variables and one outcome variable, all *randomly* generated, the odds are pretty good that at least one predictor will (falsely) turn out to be statistically significant if you do a series of 20 significance tests at the alpha = 0.05 level. As previously discussed, this is called a *Type 1 error*. You can calculate this probability by first finding the probability that all will *correctly* test nonsignificant at the 0.05 level. The probability that *one* will correctly test nonsignificant is 0.95, so the probability that all 20 will correctly test nonsignificant is 0.95 × 0.95 × 0.95…, or $0.95^{20} = 0.36$.[1] The probability that at least one predictor will (falsely) test significant is the flip side of this probability, or 1 – (*probability that all will be nonsignificant*) = 0.64. This is known as *alpha inflation*.

This issue is related to the problem of overfitting in data mining, or "fitting the model to the noise." The more variables you add, or the more models you run, the greater the probability that something will emerge as "significant" just by chance.

KEY TERMS FOR MULTIPLE TESTING

Type 1 error

Mistakenly concluding that an effect is statistically significant

False discovery rate

Across multiple tests, the rate of making a Type 1 error

Alpha inflation

The multiple testing phenomenon, in which *alpha*, the probability of making a Type 1 error, increases as you conduct more tests

Adjustment of p-values

Accounting for doing multiple tests on the same data

Overfitting

Fitting the noise

In supervised learning tasks, a holdout set where models are assessed on data that the model has not seen before mitigates this risk. In statistical and machine learning tasks not involving a labeled holdout set, the risk of reaching conclusions based on statistical noise persists.

In statistics, there are some procedures intended to deal with this problem in very specific circumstances. For example, if you are comparing results across multiple treatment groups, you might ask multiple questions. So, for treatments A–C, you might ask:

- Is A different from B?
- Is B different from C?
- Is A different from C?

Or, in a clinical trial, you might want to look at results from a therapy at multiple stages. In each case, you are asking multiple questions, and with each question, you are increasing the chance of being fooled by chance. Adjustment procedures in statistics can compensate for this by setting the bar for statistical significance more stringently than it would be set for a single hypothesis test. These adjustment procedures typically involve "dividing up the alpha" according to the number of tests. This results in a smaller alpha (i.e., a more stringent bar for statistical significance) for each test. One such procedure, the Bonferroni adjustment, simply divides the alpha by the number of comparisons. Another, used in comparing multiple group means, is Tukey's "honest significant difference," or *Tukey's HSD*. This test applies to the maximum difference among group means, comparing it to a benchmark based on the *t-distribution* (roughly equivalent to shuffling all the values together, dealing out resampled groups of the same sizes as the original groups, and finding the maximum difference among the resampled group means).

However, the problem of multiple comparisons goes beyond these highly structured cases and is related to the phenomenon of repeated data "dredging" that gives rise to the saying about torturing the data. Put another way, given sufficiently complex data, if you haven't found something interesting, you simply haven't looked long and hard enough. More data is available now than ever before, and the number of journal articles published nearly doubled between 2002 and 2010. This gives rise to lots of opportunities to find something interesting in the data, including multiplicity issues such as:

- Checking for multiple pairwise differences across groups
- Looking at multiple subgroup results ("We found no significant treatment effect overall, but we did find an effect for unmarried women younger than 30")
- Trying lots of statistical models
- Including lots of variables in models

- Asking a number of different questions (i.e., different possible outcomes)

FALSE DISCOVERY RATE

The term *false discovery rate* was originally used to describe the rate at which a given set of hypothesis tests would falsely identify a significant effect. It became particularly useful with the advent of genomic research, in which massive numbers of statistical tests might be conducted as part of a gene sequencing project. In these cases, the term applies to the testing protocol, and a single false "discovery" refers to the outcome of a hypothesis test (e.g., between two samples). Researchers sought to set the parameters of the testing process to control the false discovery rate at a specified level. The term has also been used for classification in data mining; it is the misclassification rate within the class 1 predictions. Or, put another way, it is the probability that a "discovery" (labeling a record as a "1") is false. Here we typically are dealing with the case where 0s are abundant and 1s are interesting and rare (see Chapter 5 and "The Rare Class Problem").

For a variety of reasons, including especially this general issue of "multiplicity," more research does not necessarily mean better research. For example, the pharmaceutical company Bayer found in 2011 that when it tried to replicate 67 scientific studies, it could fully replicate only 14 of them. Nearly two-thirds could not be replicated at all.

In any case, the adjustment procedures for highly defined and structured statistical tests are too specific and inflexible to be of general use to data scientists. The bottom line for data scientists on multiplicity is:

- For predictive modeling, the risk of getting an illusory model whose apparent efficacy is largely a product of random chance is mitigated by cross-validation (see "Cross-Validation") and use of a holdout sample.

- For other procedures without a labeled holdout set to check the model, you must rely on:
  - Awareness that the more you query and manipulate the data, the greater the role that chance might play.
  - Resampling and simulation heuristics to provide random chance benchmarks against which observed results can be compared.

KEY IDEAS

- Multiplicity in a research study or data mining project (multiple comparisons, many variables, many models, etc.) increases the risk of concluding that something is significant just by chance.
- For situations involving multiple statistical comparisons (i.e., multiple tests of significance), there are statistical adjustment procedures.
- In a data mining situation, use of a holdout sample with labeled outcome variables can help avoid misleading results.

Further Reading

- For a short exposition of one procedure (Dunnett's test) to adjust for multiple comparisons, see David Lane's online statistics text.
- For an in-depth treatment of more flexible statistical procedures for adjusting p-values, see *Resampling-Based Multiple Testing* by Peter Westfall and Stanley Young (Wiley, 1993).
- For a discussion of data partitioning and the use of holdout samples in predictive modeling, see Chapter 2 of *Machine Learning for Business Analytics*, by Galit Shmueli, Peter Bruce, Nitin Patel,

Peter Gedeck, Inbal Yahav, and Kenneth Lichtendahl (Wiley, 2007–2025, with editions for R, Python, Excel, RapidMiner, and JMP).

Degrees of Freedom

In the documentation and settings for many statistical tests and probability distributions, you will see a reference to "degrees of freedom." The concept is applied to statistics calculated from sample data, and refers to the number of values free to vary. For example, if you know the mean for a sample of 10 values, there are 9 degrees of freedom (once you know 9 of the sample values, the 10th can be calculated and is not free to vary). The degrees of freedom parameter, as applied to many probability distributions, affects the shape of the distribution.

The number of degrees of freedom is an input to many statistical tests. For example, degrees of freedom is the name given to the $n - 1$ denominator seen in the calculations for variance and standard deviation. Why does it matter? When you use a sample to estimate the variance for a population, you will end up with an estimate that is slightly biased downward if you use n in the denominator. If you use $n - 1$ in the denominator, the estimate will be free of that bias.

KEY TERMS FOR DEGREES OF FREEDOM

n or sample size

The number of observations (also called *rows* or *records*) in the data

d.f.

Degrees of freedom

A large share of a traditional statistics course or text is consumed by various standard tests of hypotheses (t-test, F-test, etc.). When sample statistics are

standardized for use in traditional statistical formulas, degrees of freedom is part of the standardization calculation to ensure that your standardized data matches the appropriate reference distribution (t-distribution, F-distribution, etc.).

Is it important for data science? Not really, at least in the context of significance testing. For one thing, formal statistical tests are used only sparingly in data science. For another, the data size is usually large enough that it rarely makes a real difference for a data scientist whether, for example, the denominator has n or $n - 1$. (As n gets large, the bias that would come from using n in the denominator disappears.)

There is one context, though, in which it is relevant: the use of factored variables in regression (including logistic regression). Some regression algorithms choke if exactly redundant predictor variables are present. This most commonly occurs when factoring categorical variables into binary indicators (dummies). Consider the variable "day of week." Although there are seven days of the week, there are only six degrees of freedom in specifying day of week. For example, once you know that day of week is not Monday through Saturday, you know it must be Sunday. Inclusion of the Mon.–Sat. indicators thus means that *also* including Sunday would cause the regression to fail, due to a *multicollinearity* error.

KEY IDEAS

- The number of degrees of freedom (d.f.) forms part of the calculation to standardize test statistics so they can be compared to reference distributions (t-distribution, F-distribution, etc.).
- The concept of degrees of freedom lies behind the factoring of categorical variables into $n - 1$ indicator or dummy variables when doing a regression (to avoid multicollinearity).

Further Reading

There are several web tutorials on degrees of freedom.

ANOVA

Suppose that, instead of an A/B test, we had a comparison of multiple groups, say A/B/C/D, each with numeric data. The statistical procedure that tests for a statistically significant difference among the groups is called *analysis of variance*, or *ANOVA*.

KEY TERMS FOR ANOVA

Pairwise comparison

A hypothesis test (e.g., of means) between two groups among multiple groups

Omnibus test

A single hypothesis test of the overall variance among multiple group means

Decomposition of variance

Separation of components contributing to an individual value (e.g., from the overall average, from a treatment mean, and from a residual error)

F-statistic

A standardized statistic that measures the extent to which differences among group means exceed what might be expected in a chance model

SS

"Sum of squares," referring to deviations from some average value

Table 3-3 shows the stickiness of four web pages, defined as the number of seconds a visitor spent on the page. The four pages are switched out so that each web visitor receives one at random. There are a total of five visitors for each page, and in Table 3-3, each column is an independent set of data. The first viewer for page 1 has no connection to the first viewer for page 2. Note that in a web test like this, we cannot fully implement the classic randomized sampling design in which each visitor is selected at random from some huge population. We must take the visitors as they come.

Visitors may systematically differ depending on time of day, time of week, season of the year, conditions of their internet, what device they are using, and so on. These factors should be considered as potential bias when the experiment results are reviewed.

Table 3-3. Stickiness (in seconds) of four web pages

	Page 1	Page 2	Page 3	Pag
	164	178	175	155
	172	191	193	166
	177	182	171	164
	156	185	163	170
	195	177	176	168
Average	172	185	176	162
Grand average				173.7

Now we have a conundrum (see Figure 3-6). When we were comparing just two groups, it was a simple matter; we merely looked at the difference between the means of each group. With four means, there are six possible comparisons between groups:

- Page 1 compared to page 2
- Page 1 compared to page 3
- Page 1 compared to page 4

- Page 2 compared to page 3
- Page 2 compared to page 4
- Page 3 compared to page 4

The more such *pairwise* comparisons we make, the greater the potential for being fooled by random chance (see "Multiple Testing"). Instead of worrying about all the different comparisons between individual pages we could possibly make, we can do a single overall test that addresses the question, "Could all the pages have the same underlying stickiness, and the differences among them be due to the random way in which a common set of page times got allocated among the four pages?"

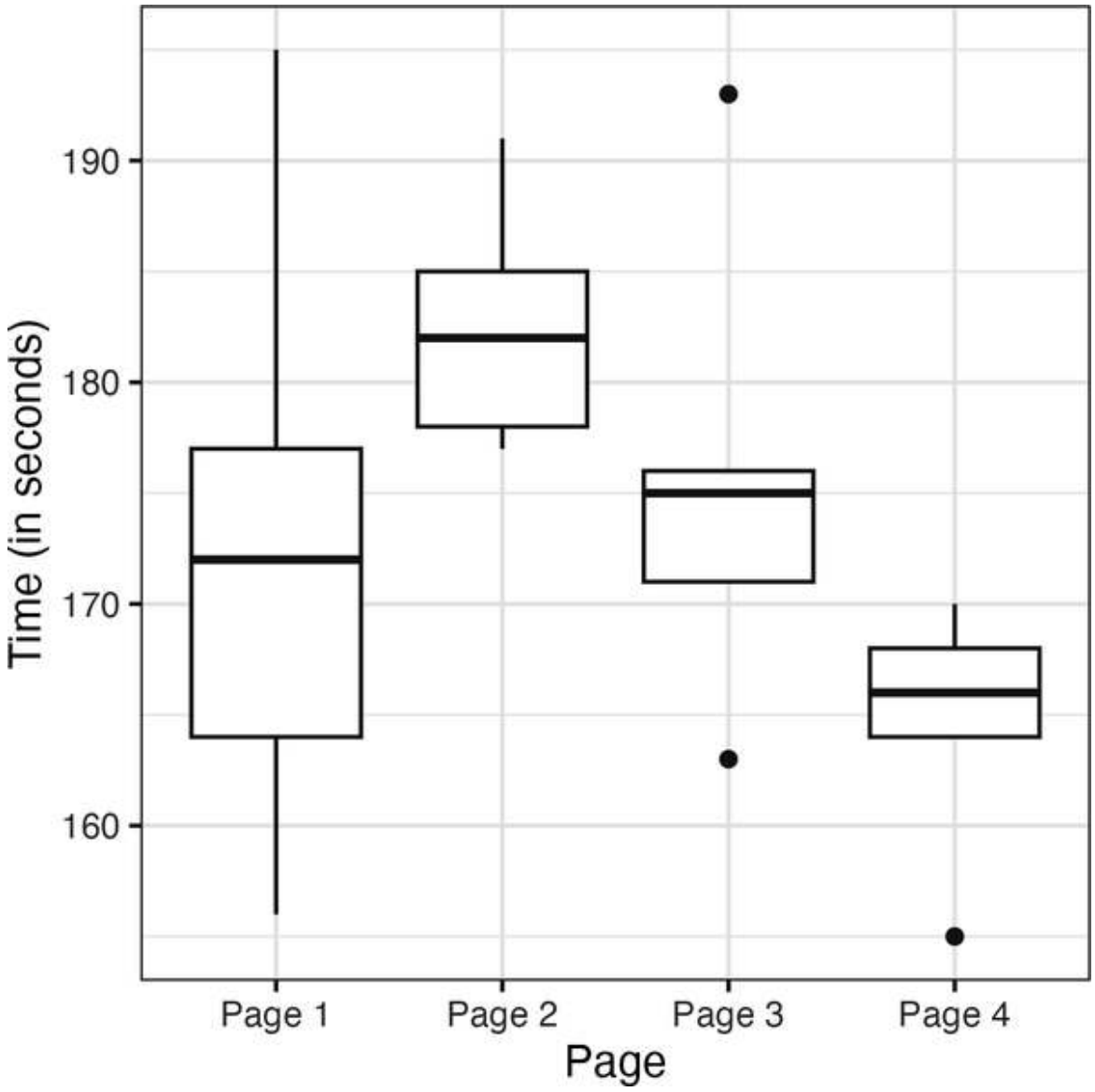


Figure 3-6. Boxplots of the four groups show considerable differences among them

The procedure used to test this is ANOVA. The basis for it can be seen in the following resampling procedure (specified here for the A/B/C/D test of web page stickiness):

1. Combine all the data together in a single box.
2. Shuffle and draw out four resamples of five values each.
3. Record the mean of each of the four groups.
4. Record the variance among the four group means.

5. Repeat steps 2–4 many (say, 1,000) times.

What proportion of the time did the resampled variance exceed the observed variance? This is the p-value.

This type of permutation test is a bit more involved than the type used in "Permutation Test". Fortunately, the `aovp` function in the `lmPerm` package computes a permutation test for this case:

```
library(lmPerm)
four_sessions  <- read_csv("four_sessions.csv", show_col_types = FALSE)
summary(aovp(Time ~ Page, data = four_sessions))
---
[1] "Settings:  unique SS "
Component 1 :
           Df R Sum Sq R Mean Sq Iter Pr(Prob)
Page        3    831.4    277.13 3104  0.09278 .
Residuals  16   1618.4    101.15
--
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-value, given by `Pr(Prob)`, is 0.09278. In other words, given the same underlying stickiness, 9.3% of the time the response rate among four pages might differ as much as was actually observed, just by chance. This degree of improbability falls short of the traditional statistical threshold of 5%, so we conclude that the difference among the four pages could have arisen by chance.

The column `Iter` lists the number of iterations taken in the permutation test. The other columns correspond to a traditional ANOVA table and are described next.

In Python, we can compute the permutation test using the following code:

```
four_sessions = pd.read_csv("four_sessions.csv")

observed_variance = four_sessions.groupby("Page").mean().var().iloc[0]
print("Observed means:",
four_sessions.groupby("Page").mean().to_numpy().ravel())
print("Variance:", observed_variance)
```

```
def perm_test(df, rng):
    df = df.copy()
    df["Time"] = rng.permutation(df["Time"].values)
    return df.groupby("Page").mean().var().iloc[0]

rng = np.random.default_rng(seed=321)
perm_variance = [perm_test(four_sessions, rng) for _ in range(3000)]
print("Pr(Prob)", np.mean([var > observed_variance for var in perm_variance]))
```

F-Statistic

Just like the t-test can be used instead of a permutation test for comparing the mean of two groups, there is a statistical test for ANOVA based on the *F-statistic*. The F-statistic is based on the ratio of the variance across group means (i.e., the treatment effect) to the variance due to residual error. The higher this ratio, the more statistically significant the result. If the data follows a normal distribution, then statistical theory dictates that the statistic should have a certain distribution. Based on this, it is possible to compute a p-value.

In R, we can compute an *ANOVA table* using the `aov` function:

```
summary(aov(Time ~ Page, data = four_sessions))
---
            Df Sum Sq Mean Sq F value Pr(>F)
Page         3  831.4   277.1    2.74 0.0776 .
Residuals   16 1618.4   101.2
--
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The `statsmodels` package provides an ANOVA implementation in Python:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

model = smf.ols("Time ~ Page", data=four_sessions).fit()

aov_table = sm.stats.anova_lm(model)
aov_table
```

The output from the Python code is almost identical to that from R.

`Df` is “degrees of freedom,” `Sum Sq` is “sum of squares,” `Mean Sq` is “mean squares” (short for mean-squared deviations), and `F value` is the F-statistic. For the grand average, sum of squares is the departure of the grand average from 0, squared, times 20 (the number of observations). The degrees of freedom for the grand average is 1, by definition.

For the treatment means, the degrees of freedom is 3 (once three values are set, and then the grand average is set, the other treatment mean cannot vary). Sum of squares for the treatment means is the sum of squared departures between the treatment means and the grand average.

For the residuals, degrees of freedom is 16 (20 observations, 16 of which can vary after the grand mean and the treatment means are set), and SS is the sum of squared difference between the individual observations and the treatment means. Mean squares (MS) is the sum of squares divided by the degrees of freedom.

The F-statistic is MS(treatment)/MS(error). The F value thus depends only on this ratio and can be compared to a standard F-distribution to determine whether the differences among treatment means are greater than would be expected in random chance variation.

DECOMPOSITION OF VARIANCE

Observed values in a data set can be considered sums of different components. For any observed data value within a data set, we can break it down into the grand average, the treatment effect, and the residual error. We call this a "decomposition of variance":

1. Start with grand average (173.75 for web page stickiness data).
2. Add treatment effect, which might be negative (independent variable = web page).
3. Add residual error, which might be negative.

Thus the decomposition of the variance for the top-left value in the A/B/C/D test table is as follows:

1. Start with grand average: 173.75.
2. Add treatment (group) effect: –1.75 (172 – 173.75).
3. Add residual: –8 (164 – 172).
4. Equals: 164.

Two-Way ANOVA

The A/B/C/D test just described is a "one-way" ANOVA, in which we have one factor (group) that is varying. We could have a second factor involved —say, "weekend versus weekday"—with data collected on each combination (group A weekend, group A weekday, group B weekend, etc.). This would be a "two-way ANOVA," and we would handle it in a similar fashion to the one-way ANOVA by identifying the "interaction effect." After identifying the grand average effect and the treatment effect, we then separate the weekend and weekday observations for each group and find the difference between the averages for those subsets and the treatment average.

You can see that ANOVA and then two-way ANOVA are the first steps on the road toward a full statistical model, such as regression and logistic regression, in which multiple factors and their effects can be modeled (see Chapter 4).

KEY IDEAS

- ANOVA is a statistical procedure for analyzing the results of an experiment with multiple groups.
- It is the extension of similar procedures for the A/B test, used to assess whether the overall variation among groups is within the range of chance variation.
- A useful outcome of ANOVA is the identification of variance components associated with group treatments, interaction effects, and errors.

Further Reading

- *Introductory Statistics and Analytics: A Resampling Perspective* by Peter Bruce (Wiley, 2014) has a chapter on ANOVA.
- *Introduction to Design and Analysis of Experiments* by George Cobb (Wiley, 2008) is a comprehensive and readable treatment of its subject.

Chi-Square Test

Web testing often goes beyond A/B testing and tests multiple treatments at once. The chi-square test is used with count data to test how well it fits some expected distribution. The most common use of the *chi-square* statistic in statistical practice is with $r \times c$ contingency tables, to assess

whether the null hypothesis of independence among variables is reasonable (see also "Chi-Square Distribution").

The chi-square test was originally developed by Karl Pearson in 1900. The term *chi* comes from the Greek letter X used by Pearson in the article.

KEY TERMS FOR CHI-SQUARE TEST

Chi-square statistic

A measure of the extent to which some observed data departs from expectation

Expectation or expected

How we would expect the data to turn out under some assumption, typically the null hypothesis

R BY C TABLES

$r \times c$ means "rows by columns"—a 2×3 table has two rows and three columns.

Chi-Square Test: A Resampling Approach

Suppose you are testing three different headlines—A, B, and C—and you run them each on 1,000 visitors, with the results shown in Table 3-4.

Table 3-4. Web testing results for three different headlines

	Headline A	Headline B	Headline C
Click	14	8	12
No-click	986	992	988

The headlines certainly appear to differ. Headline A returns nearly twice the click rate of B. The actual numbers are small, though. A resampling procedure can test whether the click rates differ to an extent greater than chance might cause. For this test, we need to have the "expected" distribution of clicks, and in this case, that would be under the null hypothesis assumption that all three headlines share the same click rate, for an overall click rate of 34/3,000. Under this assumption, our contingency table would look like Table 3-5.

Table 3-5. Expected if all three headlines have the same click rate (null hypothesis)

	Headline A	Headline B	Headline C
Click	11.33	11.33	11.33
No-click	988.67	988.67	988.67

The *Pearson residual* is defined as:

$$R = \frac{\text{Observed} - \text{Expected}}{\sqrt{\text{Expected}}}$$

R measures the extent to which the actual counts differ from these expected counts (see Table 3-6).

Table 3-6. Pearson residuals

	Headline A	Headline B	Headline C
Click	0.792	–0.990	0.198
No-click	–0.085	0.106	–0.021

The chi-square statistic is defined as the sum of the squared Pearson residuals:

$$X = \sum_{i}^{r} \sum_{j}^{c} R^2$$

where *r* and *c* are the number of rows and columns, respectively. The chi-square statistic for this example is 1.666. Is that more than could reasonably occur in a chance model?

We can test with this resampling algorithm:

1. Constitute a box with 34 ones (clicks) and 2,966 zeros (no clicks).
2. Shuffle, take three separate samples of 1,000, and count the clicks in each.
3. Find the squared differences between the shuffled counts and the expected counts and sum them.
4. Repeat steps 2 and 3, say, 1,000 times.
5. How often does the resampled sum of squared deviations exceed the observed? That's the p-value.

The function `chisq.test` can be used to compute a resampled chi-square statistic in R. For the click data, the chi-square test is:

```
click_rate <-  read_csv("click_rates.csv", show_col_types = FALSE)
clicks <- click_rate %>%
  pivot_wider(id_cols = Headline, names_from = Click, values_from = Rate) %>%
  select(Click, `No-click`)
chisq.test(clicks, simulate.p.value = TRUE)
---
        Pearson's Chi-squared test with simulated p-value (2000 replicates)

data:  clicks
X-squared = 1.6659, df = NA, p-value = 0.4853
```

The test shows that this result could easily have been obtained by randomness.

To run a permutation test in Python, use the following implementation:

```
click_rate = pd.read_csv("click_rates.csv")
clicks = click_rate.pivot_table(index="Click", columns="Headline",
values="Rate",
    aggfunc="mean")

box = [1] * 34
box.extend([0] * 2966)
random.shuffle(box)

def chi2(observed, expected):
    pearson_residuals = []
    for row, expect in zip(observed, expected, strict=True):
        pearson_residuals.append([(observe - expect) ** 2 / expect
                                  for observe in row])
    # return sum of squares
    return np.sum(pearson_residuals)

expected_clicks = 34 / 3
expected_noclicks = 1000 - expected_clicks
expected = [34 / 3, 1000 - 34 / 3]
chi2observed = chi2(clicks.values, expected)

def perm_fun(box):
    sample_clicks = [sum(random.sample(box, 1000)),
                     sum(random.sample(box, 1000)),
                     sum(random.sample(box, 1000))]
```

```
    sample_noclicks = [1000 - n for n in sample_clicks]
    return chi2([sample_clicks, sample_noclicks], expected)

perm_chi2 = [perm_fun(box) for _ in range(2000)]

resampled_p_value = sum(perm_chi2 > chi2observed) / len(perm_chi2)
print(f"Observed chi2: {chi2observed:.4f}")
print(f"Resampled p-value: {resampled_p_value:.4f}")
```

Chi-Square Test: Statistical Theory

Asymptotic statistical theory shows that the distribution of the chi-square statistic can be approximated by a *chi-square distribution* (see "Chi-Square Distribution"). The appropriate standard chi-square distribution is determined by the *degrees of freedom* (see "Degrees of Freedom"). For a contingency table, the degrees of freedom are related to the number of rows (r) and columns (c) as follows:

$$\text{degrees of freedom} = (r-1) \times (c-1)$$

The chi-square distribution is typically skewed, with a long tail to the right; see Figure 3-7 for the distribution with 1, 2, 5, and 20 degrees of freedom. The further out on the chi-square distribution the observed statistic is, the lower the p-value.

The function `chisq.test` can be used to compute the p-value using the chi-square distribution as a reference:

```
chisq.test(clicks, simulate.p.value = FALSE)
---
        Pearson's Chi-squared test

data:  clicks
X-squared = 1.6659, df = 2, p-value = 0.4348
```

In Python, use the function `scipy.stats.chi2_contingency`:

```
res = stats.chi2_contingency(clicks)
print(f"Observed chi2: {res.statistic:.4f}")
print(f"p-value: {res.pvalue:.4f}")
```

The p-value is a little less than the resampling p-value; this is because the chi-square distribution is only an approximation of the actual distribution of the statistic.

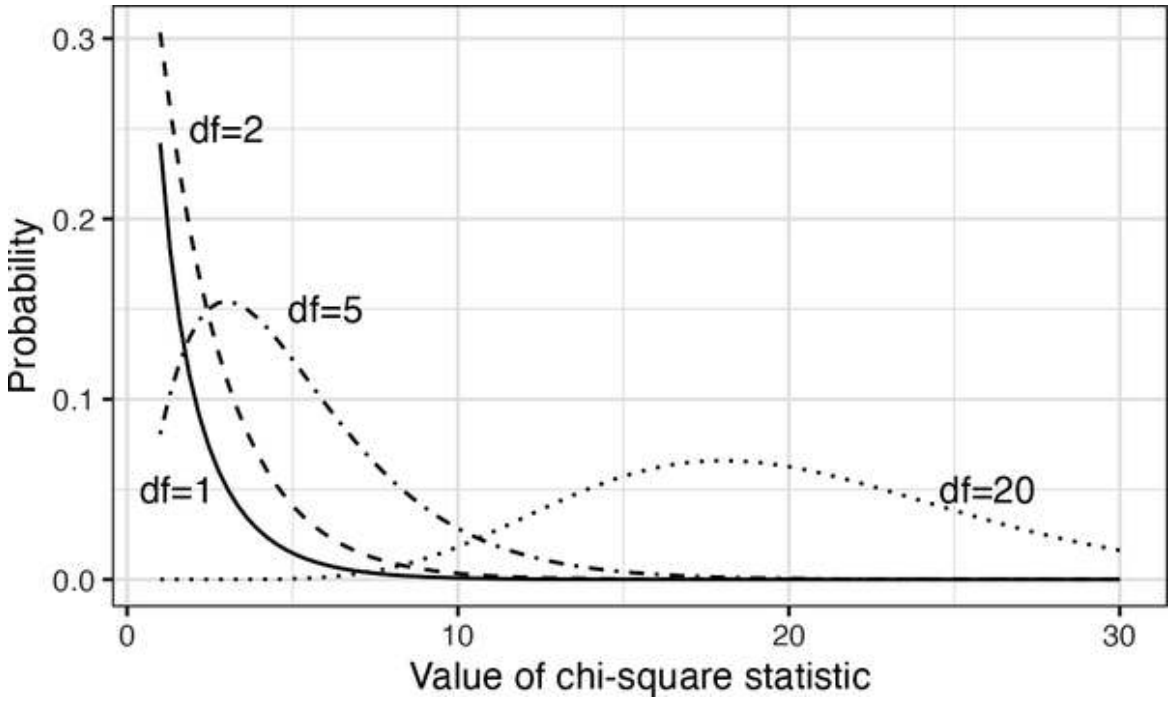


Figure 3-7. Chi-square distribution with various degrees of freedom

Fisher's Exact Test

The chi-square distribution is a good approximation of the shuffled resampling test just described, except when counts are extremely low (single digits, especially five or fewer). In such cases, the resampling procedure will yield more accurate p-values. In fact, most statistical software has a procedure to actually enumerate *all* the possible rearrangements (permutations) that can occur, tabulate their frequencies, and determine exactly how extreme the observed result is. This is called *Fisher's exact test* after the great statistician R. A. Fisher. R code for Fisher's exact test is simple in its basic form:

```
fisher.test(clicks)
---
        Fisher's Exact Test for Count Data
```

```
data:  clicks
p-value = 0.4824
alternative hypothesis: two.sided
```

The p-value is very close to the p-value of 0.4853 obtained using the resampling method.

Where some counts are very low but others are quite high (e.g., the denominator in a conversion rate), it may be necessary to do a shuffled permutation test instead of a full exact test, due to the difficulty of calculating all possible permutations. The preceding R function has several arguments that control whether to use this approximation (`simulate.p.value=TRUE or FALSE`), how many iterations should be used (`B=...`), and a computational constraint (`workspace=...`) that limits how far calculations for the *exact* result should go.

There is no implementation of Fisher's exact test easily available in Python.

DETECTING SCIENTIFIC FRAUD

An interesting example is provided by the case of Tufts University researcher Thereza Imanishi-Kari, who was accused in 1991 of fabricating data in her research. Congressman John Dingell became involved, and the case eventually led to the resignation of her colleague, David Baltimore, from the presidency of Rockefeller University.

One element in the case rested on statistical evidence regarding the expected distribution of digits in her laboratory data, where each observation had many digits. Investigators focused on the *interior* digits (ignoring the first digit and last digit of a number), which would be expected to follow a *uniform random* distribution. That is, they would occur randomly, with each digit having equal probability of occurring (the lead digit might be predominantly one value, and the final digits might be affected by rounding). Table 3-7 lists the frequencies of interior digits from the actual data in the case.

Table 3-7. Frequency of interior digits in laboratory data

Digit	Frequency
0	14
1	71
2	7
3	65
4	23
5	19
6	12
7	45
8	53
9	6

The distribution of the 315 digits, shown in Figure 3-8, certainly looks nonrandom.

Investigators calculated the departure from expectation (31.5—that's how often each digit would occur in a strictly uniform distribution) and used a chi-square test (a resampling procedure could equally have been used) to show that the actual distribution was well beyond the range of

normal chance variation, indicating the data might have been fabricated. (Note: Imanishi-Kari was ultimately exonerated after a lengthy proceeding.)

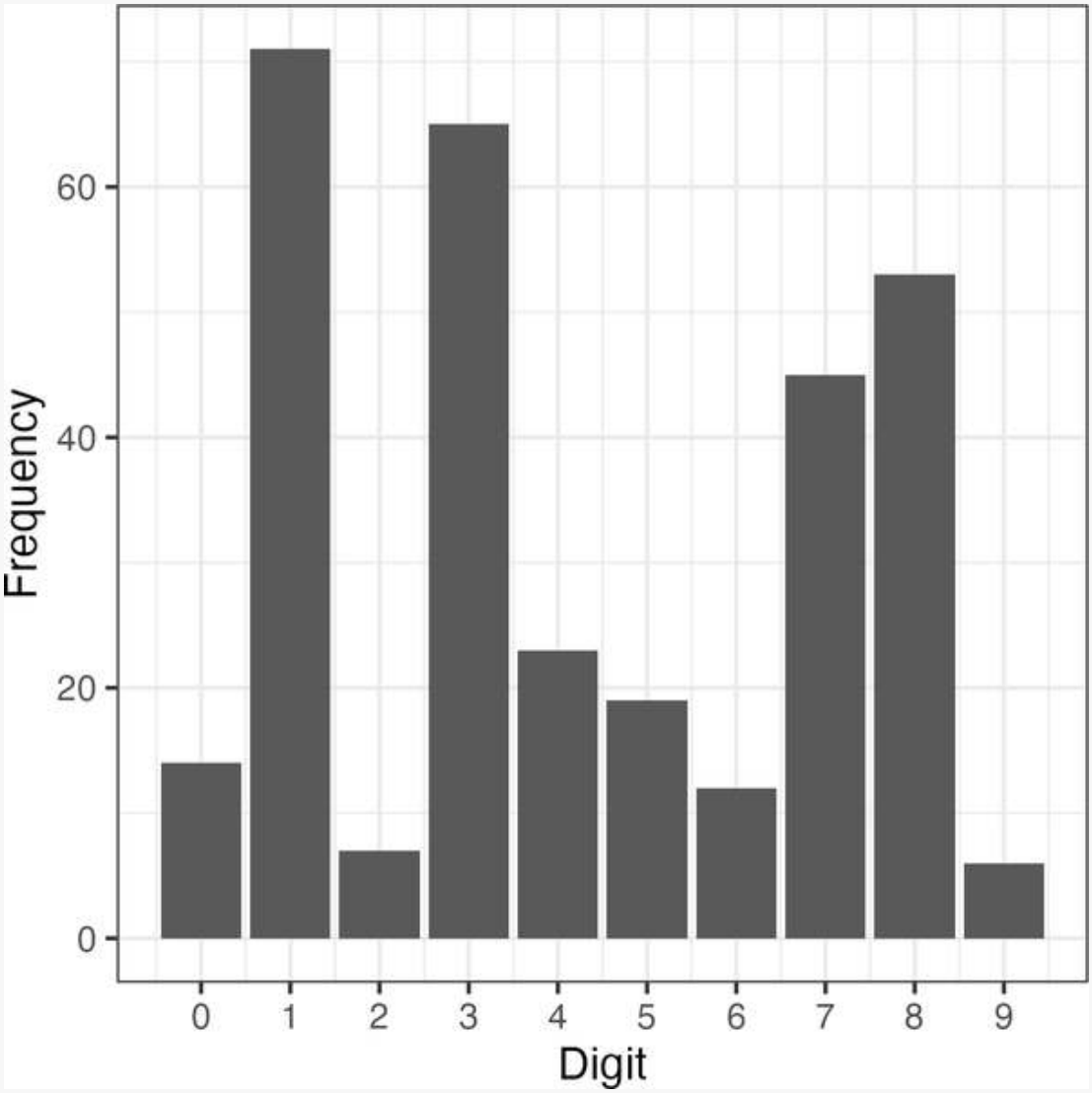


Figure 3-8. Frequency histogram for Imanishi-Kari lab data

Relevance for Data Science

The chi-square test, or Fisher's exact test, is used when you want to know whether an effect is for real or might be the product of chance. In most classical statistical applications of the chi-square test, its role is to establish statistical significance, which is typically needed before a study or an

experiment can be published. This is not so important for data scientists. In most data science experiments, whether A/B or A/B/C…, the goal is not simply to establish statistical significance but rather to arrive at the best treatment. For this purpose, multi-armed bandits (see "Multi-Arm Bandit Algorithm") offer a more complete solution.

One data science application of the chi-square test, especially Fisher's exact version, is in determining appropriate sample sizes for web experiments. These experiments often have very low click rates, and despite thousands of exposures, count rates might be too small to yield definitive conclusions in an experiment. In such cases, Fisher's exact test, the chi-square test, and other tests can be useful as a component of power and sample size calculations (see "Power and Sample Size").

Chi-square tests are used widely in research by investigators in search of the elusive statistically significant p-value that will allow publication. Chi-square tests, or similar resampling simulations, are used in data science applications more as a filter to determine whether an effect or a feature is worthy of further consideration than as a formal test of significance. For example, they are used in spatial statistics and mapping to determine whether spatial data conforms to a specified null distribution (e.g., are crimes concentrated in a certain area to a greater degree than random chance would allow?). They can also be used in automated feature selection in machine learning, to assess class prevalence across features and identify features where the prevalence of a certain class is unusually high or low, in a way that is not compatible with random variation.

KEY IDEAS

- A common procedure in statistics is to test whether observed data counts are consistent with an assumption of independence (e.g., propensity to buy a particular item is independent of gender).
- The chi-square distribution is the reference distribution (which embodies the assumption of independence) to which the observed calculated chi-square statistic must be compared.

Further Reading

- R. A. Fisher's famous "Lady Tasting Tea" example from the beginning of the 20th century remains a simple and effective illustration of his exact test. Google "Lady Tasting Tea," and you will find a number of good writeups.
- Stat Trek offers a good tutorial on the chi-square test.

Multi-Arm Bandit Algorithm

Multi-arm bandits offer an approach to testing, especially web testing, that allows explicit optimization and more rapid decision making than the traditional statistical approach to designing experiments.

KEY TERMS FOR MULTI-ARM BANDITS

Multi-arm bandit

An imaginary slot machine with multiple arms for the customer to choose from, each with different payoffs, here taken to be an analogy for a multitreatment experiment

Arm

A treatment in an experiment (e.g., "headline A in a web test")

Win

The experimental analog of a win at the slot machine (e.g., "customer clicks on the link")

A traditional A/B test involves data collected in an experiment, according to a specified design, to answer a specific question such as, "Which is better, treatment A or treatment B?" The presumption is that once we get an answer to that question, the experimenting is over and we proceed to act on the results.

You can probably perceive several difficulties with that approach. First, our answer may be inconclusive: "effect not proven." In other words, the results from the experiment may suggest an effect, but if there is an effect, we don't have a big enough sample to prove it (to the satisfaction of the traditional statistical standards). What decision do we take? Second, we might want to begin taking advantage of results that come in prior to the conclusion of the experiment. Third, we might want the right to change our minds or to try something different based on additional data that comes in after the experiment is over. The traditional approach to experiments and hypothesis tests dates from the 1920s and is rather inflexible. The advent of computer power and software has enabled more powerful, flexible approaches. Moreover, data science (and business in general) is not so

worried about statistical significance, but concerned more with optimizing overall effort and results.

Bandit algorithms, which are very popular in web testing, allow you to test multiple treatments at once and reach conclusions faster than traditional statistical designs. They take their name from slot machines used in gambling, also termed one-armed bandits (since they are configured in such a way that they extract money from the gambler in a steady flow). If you imagine a slot machine with more than one arm, each arm paying out at a different rate, you would have a multi-armed bandit, which is the full name for this algorithm.

Your goal is to win as much money as possible and, more specifically, to identify and settle on the winning arm sooner rather than later. The challenge is that you don't know at what overall rate the arms pay out—you only know the results of individual pulls on the arms. Suppose each "win" is for the same amount, no matter which arm. What differs is the probability of a win. Suppose further that you initially try each arm 50 times and get the following results:

Arm A: 10 wins out of 50

Arm B: 2 wins out of 50

Arm C: 4 wins out of 50

One extreme approach is to say, "Looks like arm A is a winner—let's quit trying the other arms and stick with A." This takes full advantage of the information from the initial trial. If A is truly superior, we get the benefit of that early on. On the other hand, if B or C is truly better, we lose any opportunity to discover that. Another extreme approach is to say, "This all looks to be within the realm of chance—let's keep pulling them all equally." This gives maximum opportunity for alternates to A to show themselves. However, in the process, we are deploying what seem to be inferior treatments. How long do we permit that? Bandit algorithms take a hybrid approach: we start pulling A more often, to take advantage of its apparent

superiority, but we don't abandon B and C. We just pull them less often. If A continues to outperform, we continue to shift resources (pulls) away from B and C and pull A more often. If, on the other hand, C starts to do better, and A starts to do worse, we can shift pulls from A back to C. If one of them turns out to be superior to A and this was hidden in the initial trial due to chance, it now has an opportunity to emerge with further testing.

Now think of applying this to web testing. Instead of multiple slot machine arms, you might have multiple offers, headlines, colors, and so on being tested on a website. Customers either click (a "win" for the merchant) or don't click. Initially, the offers are shown randomly and equally. If, however, one offer starts to outperform the others, it can be shown ("pulled") more often. But what should the parameters of the algorithm that modifies the pull rates be? What "pull rates" should we change to, and when should we change?

Here is one simple algorithm, the epsilon-greedy algorithm for an A/B test:

1. Generate a uniformly distributed random number between 0 and 1.
2. If the number lies between 0 and epsilon (where epsilon is a number between 0 and 1, typically fairly small), flip a fair coin (50/50 probability), and:
    a. If the coin is heads, show offer A.
    b. If the coin is tails, show offer B.
3. If the number is ≥ epsilon, show whichever offer has had the highest response rate to date.

Epsilon is the single parameter that governs this algorithm. If epsilon is 1, we end up with a standard simple A/B experiment (random allocation between A and B for each subject). If epsilon is 0, we end up with a purely *greedy* algorithm—one that chooses the best available immediate option (a local optimum). It seeks no further experimentation, simply assigning subjects (web visitors) to the best-performing treatment.

A more sophisticated algorithm uses "Thompson's sampling." This procedure "samples" (pulls a bandit arm) at each stage to maximize the probability of choosing the best arm. Of course you don't know which is the best arm—that's the whole problem!—but as you observe the payoff with each successive draw, you gain more information. Thompson's sampling uses a Bayesian approach: some prior distribution of rewards is assumed initially, using what is called a *beta distribution* (this is a common mechanism for specifying prior information in a Bayesian problem). As information accumulates from each draw, this information can be updated, allowing the selection of the next draw to be better optimized as far as choosing the right arm.

Bandit algorithms can efficiently handle 3+ treatments and move toward optimal selection of the "best." For traditional statistical testing procedures, the complexity of decision making for 3+ treatments far outstrips that of the traditional A/B test, and the advantage of bandit algorithms is much greater.

KEY IDEAS

- Traditional A/B tests envision a random sampling process, which can lead to excessive exposure to the inferior treatment.
- Multi-arm bandits, in contrast, alter the sampling process to incorporate information learned during the experiment and reduce the frequency of the inferior treatment.
- They also facilitate efficient treatment of more than two treatments.
- There are different algorithms for shifting sampling probability away from the inferior treatment(s) and to the (presumed) superior one.

Further Reading

- An excellent short treatment of multi-arm bandit algorithms is found in *Bandit Algorithms for Website Optimization*, by John Myles White (O'Reilly, 2012). White includes Python code, as well as the results of simulations to assess the performance of bandits.
- For more (somewhat technical) information about Thompson sampling, see "Analysis of Thompson Sampling for the Multi-armed Bandit Problem" by Shipra Agrawal and Navin Goyal.

Power and Sample Size

If you run a web test, how do you decide how long it should run (i.e., how many impressions per treatment are needed)? Despite what you may read in many guides to web testing, there is no good general guidance—it depends, mainly, on the frequency with which the desired goal is attained.

KEY TERMS FOR POWER AND SAMPLE SIZE

Effect size

The minimum size of the effect that you hope to be able to detect in a statistical test, such as "a 20% improvement in click rates"

Power

The probability of detecting a given effect size with a given sample size

Significance level

The statistical significance level at which the test will be conducted

One step in statistical calculations for sample size is to ask, "Will a hypothesis test actually reveal a difference between treatments A and B?"

The outcome of a hypothesis test—the p-value—depends on what the real difference is between treatment A and treatment B. It also depends on the luck of the draw—who gets selected for the groups in the experiment. But it makes sense that the bigger the actual difference between treatments A and B, the greater the probability that our experiment will reveal it; and the smaller the difference, the more data will be needed to detect it. To distinguish between a 0.350 hitter and a 0.200 hitter in baseball, not that many at-bats are needed. To distinguish between a 0.300 hitter and a 0.280 hitter, a good many more at-bats will be needed.

Power is the probability of detecting a specified *effect size* with specified sample characteristics (size and variability). For example, we might say (hypothetically) that the probability of distinguishing between a 0.330 hitter and a 0.200 hitter in 25 at-bats is 0.75. The effect size here is a difference of 0.130. And "detecting" means that a hypothesis test will reject the null hypothesis of "no difference" and conclude there is a real effect. So the experiment of 25 at-bats ($n = 25$) for two hitters, with an effect size of 0.130, has (hypothetical) power of 0.75, or 75%.

You can see that there are several moving parts here, and it is easy to get tangled up in the numerous statistical assumptions and formulas that will be needed (to specify sample variability, effect size, sample size, alpha-level for the hypothesis test, etc., and to calculate power). Indeed, there is special-purpose statistical software to calculate power. Most data scientists will not need to go through all the formal steps needed to report power, for example, in a published paper. However, they may face occasions where they want to collect some data for an A/B test, and collecting or processing the data involves some cost. In that case, knowing approximately how much data to collect can help avoid the situation where you collect data at some effort, and the result ends up being inconclusive. Here's a fairly intuitive alternative approach:

1. Start with some hypothetical data that represents your best guess about the data that will result (perhaps based on prior data)—for example, a box with 20 ones and 80 zeros to represent a 0.200 hitter, or a box with some observations of "time spent on website."

2. Create a second sample simply by adding the desired effect size to the first sample—for example, a second box with 33 ones and 67 zeros, or a second box with 25 seconds added to each initial "time spent on website."
3. Draw a bootstrap sample of size *n* from each box.
4. Conduct a permutation (or formula-based) hypothesis test on the two bootstrap samples and record whether the difference between them is statistically significant.
5. Repeat the preceding two steps many times and determine how often the difference was significant—that's the estimated power.

Sample Size

The most common use of power calculations is to estimate how big a sample you will need.

For example, suppose you are looking at click-through rates (clicks as a percentage of exposures), and testing a new ad against an existing ad. How many clicks do you need to accumulate in the study? If you are interested only in results that show a huge difference (say, a 50% difference), a relatively small sample might do the trick. If, on the other hand, even a minor difference would be of interest, then a much larger sample is needed. A standard approach is to establish a policy that a new ad must do better than an existing ad by some percentage, say, 10%; otherwise, the existing ad will remain in place. This goal, the "effect size," then drives the sample size.

For example, suppose current click-through rates are about 1.1%, and you are seeking a 10% boost to 1.21%. So we have two boxes: box A with 1.1% ones (say, 110 ones and 9,890 zeros), and box B with 1.21% ones (say, 121 ones and 9,879 zeros). For starters, let's try 300 draws from each box (this would be like 300 "impressions" for each ad). Suppose our first draw yields the following:

Box A: 3 ones

Box B: 5 ones

Right away we can see that any hypothesis test would reveal this difference (5 versus 3) to be well within the range of chance variation. This combination of sample size (n = 300 in each group) and effect size (10% difference) is too small for any hypothesis test to reliably show a difference.

So we can try increasing the sample size (let's try 2,000 impressions), and require a larger improvement (50% instead of 10%).

For example, suppose current click-through rates are still 1.1%, but we are now seeking a 50% boost to 1.65%. So we have two boxes: box A still with 1.1% ones (say, 110 ones and 9,890 zeros), and box B with 1.65% ones (say, 165 ones and 9,835 zeros). Now we'll try 2,000 draws from each box. Suppose our first draw yields the following:

Box A: 19 ones

Box B: 34 ones

A significance test on this difference (34 – 19) shows it still registers as "not significant" (though much closer to significance than the earlier difference of 5 – 3). To calculate power, we would need to repeat the previous procedure many times, or use statistical software that can calculate power, but our initial draw suggests to us that even detecting a 50% improvement will require several thousand ad impressions.

In summary, for calculating power or required sample size, there are four moving parts:

- Sample size
- Effect size you want to detect
- Significance level (alpha) at which the test will be conducted

- Power

Specify any three of them, and the fourth can be calculated. Most commonly, you would want to calculate sample size, so you must specify the other three. With R and Python, you also have to specify the alternative hypothesis as "greater" or "larger" to get a one-sided test; see "One-Way Versus Two-Way Hypothesis Tests" for more discussion of one-way versus two-way tests. Here is R code for a test involving two proportions, where both samples are the same size (this uses the `pwr` package):

```
library(pwr)
effect_size <- ES.h(p1 = 0.0121, p2 = 0.011)
pwr.2p.test(h = effect_size, sig.level = 0.05, power = 0.8,
  alternative = "greater")
---
     Difference of proportion power calculation for binomial distribution
                                                          (arcsine
transformation)

              h = 0.01029785
              n = 116601.7
      sig.level = 0.05
          power = 0.8
    alternative = greater

NOTE: same sample sizes
```

The function `ES.h` calculates the effect size. We see that if we want a power of 80%, we require a sample size of almost 120,000 impressions. If we are seeking a 50% boost (`p1=0.0165`), the sample size is reduced to 5,500 impressions.

The `statsmodels` package contains several methods for power calculation. Here, we use `proportion_effectsize` to calculate the effect size and `TTestIndPower` to solve for the sample size:

```
effect_size = sm.stats.proportion_effectsize(0.0121, 0.011)
analysis = sm.stats.TTestIndPower()
result = analysis.solve_power(effect_size=effect_size,
                              alpha=0.05, power=0.8, alternative="larger")
```

```
print(f"Sample Size: {result:.3f}")
---
Sample Size: 116602.393
```

KEY IDEAS

- Finding out how big a sample size you need requires thinking ahead to the statistical test you plan to conduct.
- You must specify the minimum size of the effect that you want to detect.
- You must also specify the required probability of detecting that effect size (power).
- Finally, you must specify the significance level (alpha) at which the test will be conducted.

Further Reading

- *Sample Size Determination and Power* by Thomas Ryan (Wiley, 2013) is a comprehensive and readable review of this subject.
- Steve Simon, a statistical consultant, has written a very engaging narrative-style post on the subject.

Summary

The principles of experimental design—randomization of subjects into two or more groups receiving different treatments—allow us to draw valid conclusions about how well the treatments work. It is best to include a control treatment of "making no change." The subject of formal statistical inference—hypothesis testing, p-values, t-tests, and much more along these lines—occupies much time and space in a traditional statistics course or text, and the formality is mostly unneeded from a data science perspective.

However, it remains important to recognize the role that random variation can play in fooling the human brain. Intuitive resampling procedures (permutation and bootstrap) allow data scientists to gauge the extent to which chance variation can play a role in their data analysis.

Exploration with AI

1. For many applications, designing an A/B test implementation is challenging. Consider the task of a political strategist who wants to test the effectiveness of using highway billboards to associate tariffs with inflation and turn the public against them. Billboards are expensive, so proof of concept is needed. Ask your AI to (a) compare a billboard campaign on tariffs to a social media campaign on tariffs, in terms of likely general effectiveness, and (b) devise the specifics of an A/B test for each, assessing the qualities of such an experiment.
2. You are trying to increase the web sales of a premium garden hose product, which currently has a 2.5% rate of converting sales from the product page. You have come up with an A/B test for two different web page designs. Ask your AI to help you determine how many web visitors you will need in each arm of the experiment. Assess whether the AI's response is reasonable, and whether it has asked you any needed questions.
3. Ask your AI to explain to you the p-value controversy, and whether it has evolved since the 2016 American Statistical Association statement.
4. Ask your AI to explain how statistical inference procedures (say, the p-value or chi-square test) are used in machine learning, and how that use differs from their use in traditional statistics.
5. Ask your AI to explain the interdependent factors that must be specified in power and sample size calculations, and how

specifying any three of them allows calculation of the fourth.

---

[1] The multiplication rule states that the probability of n independent events all happening is the product of the individual probabilities. For example, if you and I each flip a coin once, the probability that your coin and my coin will both land heads is $0.5 \times 0.5 = 0.25$.

Chapter 4. Regression and Prediction

Perhaps the most common goal in statistics is to answer the question, "Is the variable X (or more likely, $X_1, \ldots, X_p$) associated with a variable Y, and if so, what is the relationship and can we use it to predict Y?"

Nowhere is the nexus between statistics and data science stronger than in the realm of prediction—specifically, the prediction of an outcome (target) variable based on the values of other "predictor" variables. This process of training a model on data where the outcome is known, for subsequent application to data where the outcome is not known, is termed *supervised learning*. Another important connection between data science and statistics is in the area of *anomaly detection*, where regression diagnostics originally intended for data analysis and improving the regression model can be used to detect unusual records.

Simple Linear Regression

Simple linear regression provides a model of the relationship between the magnitude of one variable and that of a second—for example, as X increases, Y also increases. Or as X increases, Y decreases.[1] Correlation is another way to measure how two variables are related—see the section "Correlation". The difference is that while correlation measures the *strength* of an association between two variables, regression quantifies the *nature* of the relationship.

KEY TERMS FOR SIMPLE LINEAR REGRESSION

Response

The variable we are trying to predict

Synonyms

dependent variable, *Y* variable, target, outcome

Independent variable

The variable used to predict the response

Synonyms

X variable, feature, attribute, predictor

Record

The vector of predictor and outcome values for a specific individual or case

Synonyms

row, case, instance, example

Intercept

The intercept of the regression line—that is, the predicted value when $X = 0$

Synonyms

b_0, β_0

Regression coefficient

The slope of the regression line

Synonyms

slope, b_1, β_1, parameter estimates, weights

Fitted values

The estimates $\widehat{Y}_i$ obtained from the regression line

Synonym

predicted values

Residuals

The difference between the observed values and the fitted values

Synonym

errors

Least squares

The method of fitting a regression by minimizing the sum of squared residuals

Synonyms

ordinary least squares, OLS

The Regression Equation

Simple linear regression estimates how much *Y* will change when *X* changes by a certain amount. With the correlation coefficient, the variables

X and *Y* are interchangeable. With regression, we are trying to predict the *Y* variable from *X* using a linear relationship (i.e., a line):

$$Y = b_0 + b_1 X$$

We read this as "Y equals b_1 times X, plus a constant b_0." The symbol b_0 is known as the *intercept* (or constant), and the symbol b_1 as the *slope* for *X*. Both appear in R output as *coefficients*, though in general use the term *coefficient* is often reserved for b_1. The *Y* variable is known as the *response* or *dependent* variable since it depends on *X*. The *X* variable is known as the *predictor* or *independent* variable. The machine learning community tends to use other terms, calling *Y* the *target* and *X* a *feature* vector. Throughout this book, we'll use the terms *predictor* and *feature* interchangeably.

The scatterplot in Figure 4-1 displays the number of years a worker was exposed to cotton dust (`Exposure`) versus a measure of lung capacity (`PEFR` or "peak expiratory flow rate"). How is `PEFR` related to `Exposure`? It's hard to tell based just on the picture.

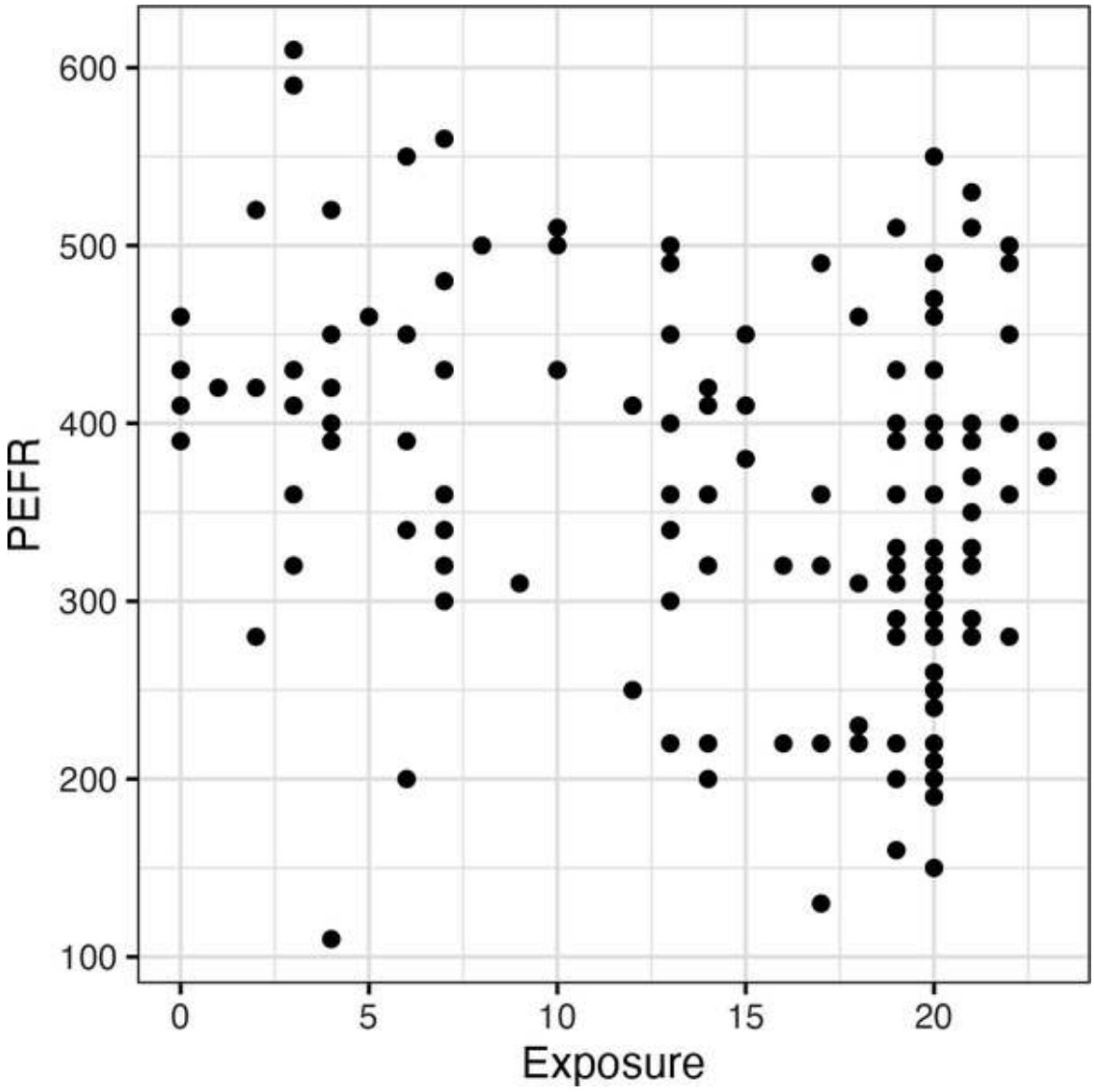


Figure 4-1. Cotton exposure versus lung capacity

Simple linear regression tries to find the "best" line to predict the response `PEFR` as a function of the predictor variable `Exposure`:

$$\text{PEFR} = b_0 + b_1 \text{Exposure}$$

The `lm` function in R can be used to fit a linear regression:

```
library(tidyverse)

lung <- read_csv("LungDisease.csv")
model <- lm(PEFR ~ Exposure, data = lung)
```

`lm` stands for *linear model*, and the ~ symbol denotes that `PEFR` is predicted by `Exposure`. With this model definition, the intercept is automatically included and fitted. If you want to exclude the intercept from the model, write the model definition as follows:

```
PEFR ~ Exposure - 1
```

Printing the `model` object produces the following output:

```
model
---
Call:
lm(formula = PEFR ~ Exposure, data = lung)

Coefficients:
(Intercept)     Exposure
    424.583       -4.185
```

The intercept, or b_0, is 424.583 and can be interpreted as the predicted `PEFR` for a worker with zero years of exposure. The regression coefficient, or b_1, can be interpreted as follows: for each additional year that a worker is exposed to cotton dust, the worker's `PEFR` measurement is reduced by 4.185.

In Python, we can use `LinearRegression` from the `scikit-learn` package (the `statsmodels` package has a linear regression implementation that is more similar to R (`sm.OLS`); we will use it later in this chapter):

```
import pandas as pd
from sklearn.linear_model import LinearRegression

lung = pd.read_csv("LungDisease.csv")

predictors = ["Exposure"]
outcome = "PEFR"

model = LinearRegression()  ❶
model.fit(lung[predictors], lung[outcome])
```

```
print(f"Intercept: {model.intercept_:.3f}")
print(f"Coefficient Exposure: {model.coef_[0]:.3f}")
```

❶ For a fixed intercept of 0, use `LinearRegression(fit_intercept=False)`.

The regression line from this model is displayed in Figure 4-2.

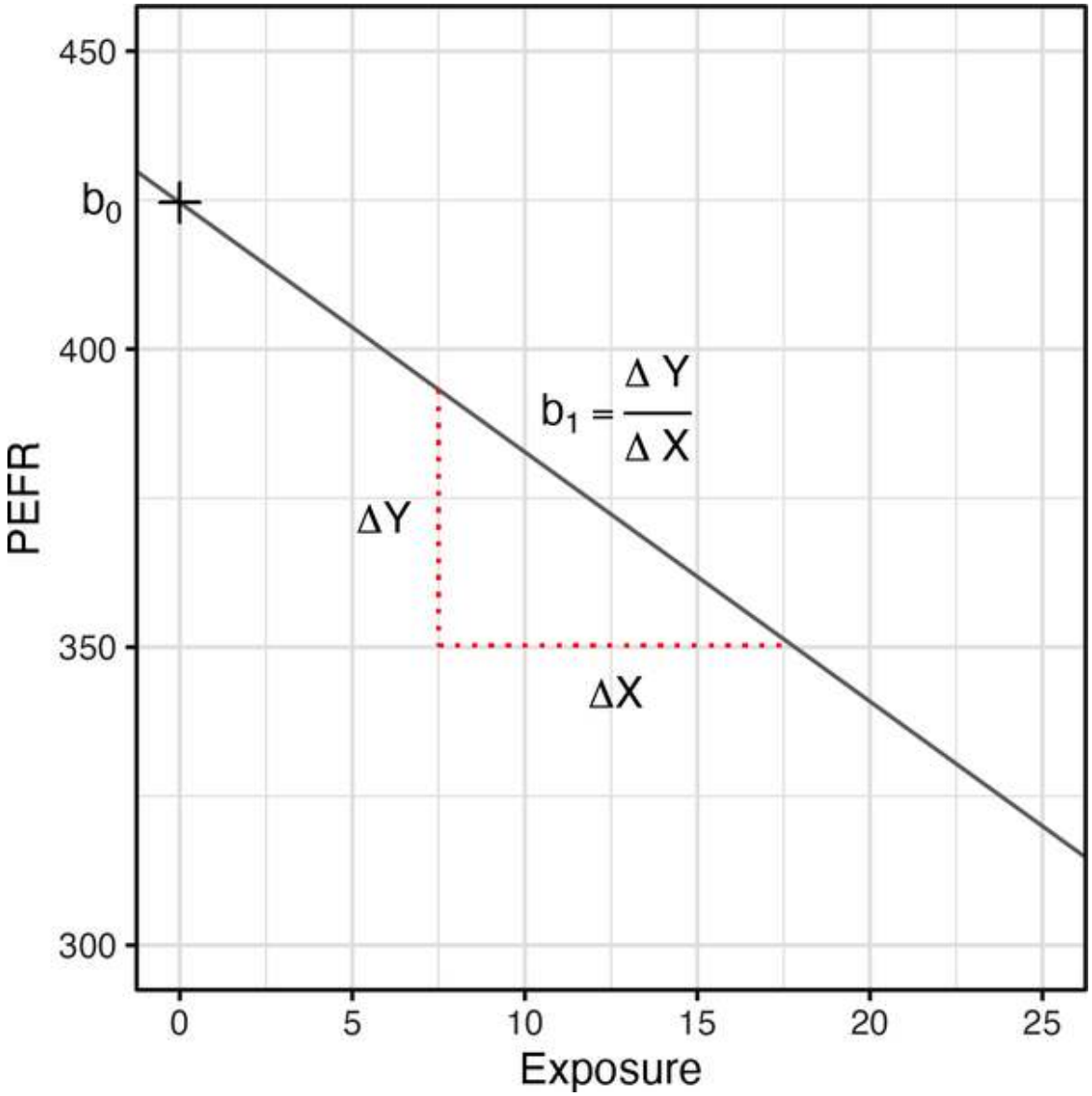


Figure 4-2. Slope and intercept for the regression fit to the lung data

Fitted Values and Residuals

Important concepts in regression analysis are the *fitted values* (the predictions) and *residuals* (prediction errors). In general, the data doesn't fall exactly on a line, so the regression equation should include an explicit error term e_i:

$$Y_i = b_0 + b_1 X_i + e_i$$

The fitted values, also referred to as the *predicted values*, are typically denoted by $\hat{Y}_i$ (Y-hat). These are given by:

$$\hat{Y}_i = \hat{b}_0 + \hat{b}_1 X_i$$

The notation $\hat{b}_0$ and $\hat{b}_1$ indicates that the coefficients are estimated versus known.

We compute the residuals $\hat{e}_i$ by subtracting the *predicted* values from the original data:

$$\hat{e}_i = Y_i - \hat{Y}_i$$

In R, obtain the fitted values and residuals using the functions `predict` and `residuals`:

```
fitted <- predict(model)
resid <- residuals(model)
```

With `scikit-learn`'s `LinearRegression` model, we use the `predict` method on the training data to get the `fitted` values and subsequently the `residuals`. As we will see, this is a general pattern that all models in `scikit-learn` follow:

```
fitted = model.predict(lung[predictors])
residuals = lung[outcome] - fitted
```

Figure 4-3 illustrates the residuals from the regression line fit to the lung data. The residuals are the length of the vertical dashed lines from the data

to the line.

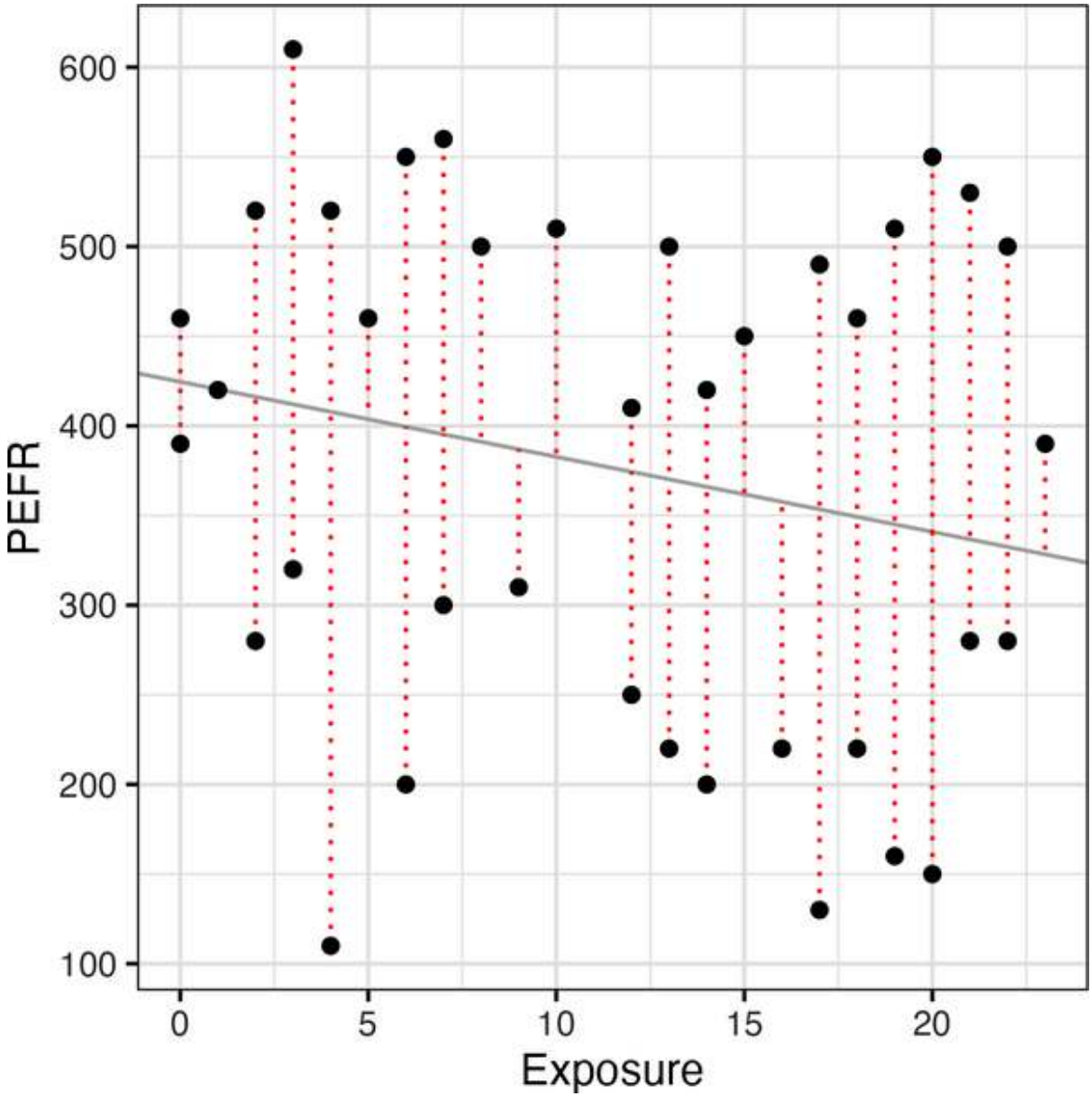


Figure 4-3. Residuals from a regression line (to accommodate all the data, the y-axis scale differs from Figure 4-2, hence the apparently different slope)

HAT NOTATION: ESTIMATES VERSUS KNOWN VALUES

The "hat" notation is used to differentiate between estimates and known values. So the symbol $\hat{b}$ ("b-hat") is an estimate of the unknown parameter b. Why do statisticians differentiate between the estimate and the true value? The estimate has uncertainty, whereas the true value is fixed.[2]

Least Squares

How is the model fit to the data? When there is a clear relationship, you could imagine fitting the line by hand. In practice, the regression line is the estimate that minimizes the sum of squared residual values, also called the *residual sum of squares* or *RSS*:

$$\begin{aligned} RSS &= \sum_{i=1}^{n} \left(Y_i - \widehat{Y}_i\right)^2 \\ &= \sum_{i=1}^{n} \left(Y_i - \hat{b}_0 - \hat{b}_1 X_i\right)^2 \end{aligned}$$

The estimates $\hat{b}_0$ and $\hat{b}_1$ are the values that minimize RSS.

The method of minimizing the sum of the squared residuals is termed *least squares* regression, or *ordinary least squares* (OLS) regression. It is often attributed to Carl Friedrich Gauss, the German mathematician, but was first published by the French mathematician Adrien-Marie Legendre in 1805. Least squares regression can be computed quickly and easily with any standard statistical software.

Historically, computational convenience is one reason for the widespread use of least squares in regression. With the advent of big data, computational speed is still an important factor. Least squares, like the mean (see "Median and Robust Estimates"), are sensitive to outliers, although this tends to be a significant problem only in small or moderate-sized data sets. See "Outliers" for a discussion of outliers in regression.

REGRESSION TERMINOLOGY

When analysts and researchers use the term *regression* by itself, they are typically referring to linear regression; the focus is usually on developing a linear model to explain the relationship between predictor variables and a numeric outcome variable. In its formal statistical sense, regression also includes nonlinear models that yield a functional relationship between predictors and outcome variables. In the machine learning community, the term is also occasionally used loosely to refer to the use of any predictive model that produces a predicted numeric outcome (as opposed to classification methods that predict a binary or categorical outcome).

Prediction Versus Explanation (Profiling)

Historically, a primary use of regression was to illuminate a supposed linear relationship between predictor variables and an outcome variable. The goal has been to understand a relationship and explain it using the data that the regression was fit to. In this case, the primary focus is on the estimated slope of the regression equation, $\hat{b}$. Economists want to know the relationship between consumer spending and GDP growth. Public health officials might want to understand whether a public information campaign is effective in promoting safe sex practices. In such cases, the focus is not on predicting individual cases but rather on understanding the overall relationship among variables.

With the advent of big data, regression is widely used to form a model to predict individual outcomes for new data (i.e., a predictive model) rather than explain data in hand. In this instance, the main items of interest are the fitted values $\hat{Y}$. In marketing, regression can be used to predict the change in revenue in response to the size of an ad campaign. Universities use regression to predict students' GPA based on their SAT scores.

A regression model that fits the data well is set up such that changes in X lead to changes in Y. However, by itself, the regression equation does not prove the direction of causation. Conclusions about causation must come from a broader understanding of the relationship. For example, a regression equation might show a definite relationship between number of clicks on a web ad and number of conversions. It is our knowledge of the marketing

process, not the regression equation, that leads us to the conclusion that clicks on the ad lead to sales, and not vice versa.

KEY IDEAS

- The regression equation models the relationship between a response variable Y and a predictor variable X as a line.
- A regression model yields fitted values and residuals—predictions of the response and the errors of the predictions.
- Regression models are typically fit by the method of least squares.
- Regression is used both for prediction and explanation.

Further Reading

For an in-depth treatment of prediction versus explanation, see Galit Shmueli's article "To Explain or to Predict?".

Multiple Linear Regression

When there are multiple predictors, the equation is simply extended to accommodate them:

$$Y = b_0 + b_1 X_1 + b_2 X_2 + \ldots + b_p X_p + e$$

Instead of a line, we now have a linear model—the relationship between each coefficient and its variable (feature) is linear.

KEY TERMS FOR MULTIPLE LINEAR REGRESSION

Root mean squared error

The square root of the average squared error of the regression (the most widely used metric to compare regression models)

Synonym

RMSE

Residual standard error

The same as the root mean squared error, but adjusted for degrees of freedom

Synonym

RSE

R-squared

The proportion of variance explained by the model, from 0 to 1

Synonyms

coefficient of determination, R^2

t-statistic

The coefficient for a predictor, divided by the standard error of the coefficient, giving a metric to compare the importance of variables in the model (see "t-Tests")

Weighted regression

Regression with the records having different weights

All of the other concepts in simple linear regression, such as fitting by least squares and the definition of fitted values and residuals, extend to the multiple linear regression setting. For example, the fitted values are given by:

$$\widehat{Y}_i = \hat{b}_0 + \hat{b}_1 X_{1,i} + \hat{b}_2 X_{2,i} + ... + \hat{b}_p X_{p,i}$$

Example: King County Housing Data

An example of using multiple linear regression is in estimating the value of houses. County assessors must estimate the value of a house for the purposes of assessing taxes. Real estate professionals and home buyers consult popular websites such as Zillow to ascertain a fair price. Here are a few rows of housing data from King County (Seattle), Washington, from the `house data.frame`:

```
house <- read_csv("house_sales.csv", show_col_types = FALSE)
head(house[, c("AdjSalePrice", "SqFtTotLiving", "SqFtLot", "Bathrooms",
      "Bedrooms", "BldgGrade")])
```

```
---
Source: local data frame [6 x 6]

  AdjSalePrice SqFtTotLiving SqFtLot Bathrooms Bedrooms BldgGrade
         (dbl)         (int)   (int)     (dbl)    (int)     (int)
1       300805          2400    9373      3.00        6         7
2      1076162          3764   20156      3.75        4        10
3       761805          2060   26036      1.75        4         8
4       442065          3200    8618      3.75        5         7
5       297065          1720    8620      1.75        4         7
6       411781           930    1012      1.50        2         8
```

The `head` method of `pandas` data frame lists the top rows:

```
house = pd.read_csv("house_sales.csv", sep="\t")

subset = ["AdjSalePrice", "SqFtTotLiving", "SqFtLot", "Bathrooms",
          "Bedrooms", "BldgGrade"]
house[subset].head()
```

The goal is to predict the sales price from the other variables. The `lm` function handles the multiple regression case simply by including more terms on the righthand side of the equation; the argument `na.action=na.omit` causes the model to drop records that have missing values:

```
house_lm <- lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade,
  data = house, na.action = na.omit)
```

`scikit-learn`'s `LinearRegression` can be used for multiple linear regression as well:

```
predictors = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
"BldgGrade"]
outcome = "AdjSalePrice"

house_lm = LinearRegression()
house_lm.fit(house[predictors], house[outcome])
```

Printing `house_lm` object produces the following output:

```
house_lm
---
Call:
lm(formula = AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade, data = house, na.action = na.omit)

Coefficients:
  (Intercept)  SqFtTotLiving        SqFtLot      Bathrooms
   -5.219e+05      2.288e+02     -6.047e-02     -1.944e+04
     Bedrooms      BldgGrade
   -4.777e+04      1.061e+05
```

For a `LinearRegression` model, intercept and coefficients are the fields `intercept_` and `coef_` of the fitted model:

```
print(f"Intercept: {house_lm.intercept_:.3f}")  ❶
print("Coefficients:")
for name, coef in zip(house_lm.feature_names_in_, house_lm.coef_,
```

```
    strict=True):
        print(f" {name}: {coef}")
```

❶ `scikit-learn` has the convention of using a trailing underscore for fields that are estimated during fitting.

The interpretation of the coefficients is as with simple linear regression: the predicted value $\widehat{Y}$ changes by the coefficient b_j for each unit change in X_j assuming all the other variables, X_k for $k \neq j$, remain the same. For example, adding an extra finished square foot to a house increases the estimated value by roughly $229; adding 1,000 finished square feet implies the value will increase by $228,800.

Assessing the Model

The most important performance metric from a data science perspective is *root mean squared error*, or *RMSE*. RMSE is the square root of the average squared error in the predicted $\hat{y}_i$ values:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}{n}}$$

This measures the overall accuracy of the model and is a basis for comparing it to other models (including models fit using machine learning techniques). Similar to RMSE is the *residual standard error*, or *RSE*. In this case we have p predictors, and the RSE is given by:

$$RSE = \sqrt{\frac{\sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}{(n - p - 1)}}$$

The only difference is that the denominator is the degrees of freedom, as opposed to number of records (see "Degrees of Freedom"). In practice, for linear regression, the difference between RMSE and RSE is very small, particularly for big data applications.

The `summary` function in R computes RSE as well as other metrics for a regression model:

```
summary(house_lm)
---
Call:
lm(formula = AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade, data = house, na.action = na.omit)

Residuals:
     Min       1Q   Median       3Q      Max
-1199479  -118908   -20977    87435  9473035

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -5.219e+05  1.565e+04 -33.342  < 2e-16 ***
SqFtTotLiving  2.288e+02  3.899e+00  58.694  < 2e-16 ***
SqFtLot       -6.047e-02  6.118e-02  -0.988    0.323
Bathrooms     -1.944e+04  3.625e+03  -5.363 8.27e-08 ***
Bedrooms      -4.777e+04  2.490e+03 -19.187  < 2e-16 ***
BldgGrade      1.061e+05  2.396e+03  44.277  < 2e-16 ***
--
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 261300 on 22681 degrees of freedom
Multiple R-squared:  0.5406,    Adjusted R-squared:  0.5405
F-statistic:  5338 on 5 and 22681 DF,  p-value: < 2.2e-16
```

`scikit-learn` provides a number of metrics for regression and classification. Here, we use `mean_squared_error` to get RMSE and `r2_score` for the coefficient of determination:

```
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

fitted = house_lm.predict(house[predictors])
RMSE = np.sqrt(mean_squared_error(house[outcome], fitted))
r2 = r2_score(house[outcome], fitted)
print(f"RMSE: {RMSE:.0f}")
print(f"r2: {r2:.4f}")
```

Use `statsmodels` to get a more detailed analysis of the regression model in Python:

```
import statsmodels.api as sm

model = sm.OLS(house[outcome], sm.add_constant(house[predictors]))
results = model.fit()
results.summary()
```

The `add_constant` method adds a constant column with value 1 to the predictors. This is required to model the intercept.

Another useful metric that you will see in software output is the *coefficient of determination*, also called the *R-squared* statistic or R^2. R-squared ranges from 0 to 1 and measures the proportion of variation in the data that is accounted for in the model. It is useful mainly in explanatory uses of regression where you want to assess how well the model fits the data. The formula for R^2 is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2}$$

The denominator is proportional to the variance of *Y*. The output from R also reports an *adjusted R-squared*, which adjusts for the degrees of freedom, effectively penalizing the addition of more predictors to a model. Seldom is this significantly different from *R-squared* in multiple regression with large data sets.

Along with the estimated coefficients, R and `statsmodels` report the standard error (SE) of the coefficients and a *t-statistic*:

$$t_b = \frac{\hat{b}}{\text{SE}\left(\hat{b}\right)}$$

The t-statistic—and its mirror image, the p-value—measures the extent to which a coefficient is "statistically significant"—that is, outside the range of what a random chance arrangement of predictor and target variable might produce. The higher the t-statistic (and the lower the p-value), the more significant the predictor. Since parsimony is a valuable model feature, it is

useful to have a tool like this to guide choice of variables to include as predictors (see "Model Selection and Stepwise Regression").

STATISTICAL OUTPUT IN SOFTWARE

In addition to the t-statistic, R and other packages will often report a *p-value* (`Pr(>|t|)` in the R output) and *F-statistic*. Data scientists do not generally get too involved with the interpretation of these statistics, nor with the issue of statistical significance. Data scientists primarily focus on the t-statistic as a useful guide for whether to include a predictor in a model or not. High t-statistics (which go with p-values near 0) indicate a predictor should be retained in a model, while very low t-statistics indicate a predictor could be dropped. See "p-Value" for more discussion.

Cross-Validation

Classic statistical regression metrics (R^2, F-statistics, and p-values) are all "in-sample" metrics—they are applied to the same data that was used to fit the model. Intuitively, you can see that it would make a lot of sense to set aside some of the original data, not use it to fit the model, and then apply the model to the set-aside (holdout) data to see how well it does. Normally, you would use a majority of the data to fit the model and use a smaller portion to test the model.

This idea of "out-of-sample" validation is not new, but it did not really take hold until larger data sets became more prevalent; with a small data set, analysts typically want to use all the data and fit the best possible model.

Using a holdout sample, though, leaves you subject to some uncertainty that arises simply from variability in the small holdout sample. How different would the assessment be if you selected a different holdout sample?

Cross-validation extends the idea of a holdout sample to multiple sequential holdout samples. The algorithm for basic *k-fold cross-validation* is as follows:

1. Set aside *1/k* of the data as a holdout sample.
2. Train the model on the remaining data.

3. Apply (score) the model to the *1/k* holdout, and record needed model assessment metrics.
4. Restore the first *1/k* of the data, and set aside the next *1/k* (excluding any records that got picked the first time).
5. Repeat steps 2 and 3.
6. Repeat until each record has been used in the holdout portion.
7. Average or otherwise, combine the model assessment metrics.

The division of the data into the training sample and the holdout sample is also called a *fold*.

Model Selection and Stepwise Regression

In some problems, many variables could be used as predictors in a regression. For example, to predict house value, additional variables such as the basement size or year built could be used. In R, these are easy to add to the regression equation:

```
house_full <- lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade + PropertyType + NbrLivingUnits +
    SqFtFinBasement + YrBuilt + YrRenovated + NewConstruction,
  data = house, na.action = na.omit)
```

In Python, we need to convert the categorical and Boolean variables into numbers:

```
predictors = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
"BldgGrade",
              "PropertyType", "NbrLivingUnits", "SqFtFinBasement", "YrBuilt",
              "YrRenovated", "NewConstruction"]

X = pd.get_dummies(house[predictors], drop_first=True, dtype=int)
X["NewConstruction"] = [1 if nc else 0 for nc in X["NewConstruction"]]

house_full = sm.OLS(house[outcome], sm.add_constant(X))
results = house_full.fit()
results.summary()
```

Adding more variables, however, does not necessarily mean we have a better model. Statisticians use the principle of *Occam's razor* to guide the choice of a model: all things being equal, a simpler model should be used in preference to a more complicated model.

Including additional variables always reduces RMSE and increases R^2 for the training data. Hence, these are not appropriate to help guide the model choice. One approach to including model complexity is to use the adjusted R^2:

$$R^2_{adj} = 1 - (1 - R^2)\frac{n - 1}{n - P - 1}$$

Here, *n* is the number of records and *P* is the number of variables in the model.

In the 1970s, Hirotugu Akaike, the eminent Japanese statistician, developed a metric called *AIC* (Akaike's Information Criterion) that penalizes adding terms to a model. In the case of regression, AIC has the form:

$$\text{AIC} = 2P + n \log(\text{RSS}/n)$$

where *P* is the number of variables and *n* is the number of records. The goal is to find the model that minimizes AIC; models with *k* more extra variables are penalized by 2*k*.

AIC, BIC, AND MALLOWS CP

The formula for AIC may seem a bit mysterious, but in fact it is based on asymptotic results in information theory. There are several variants to AIC:

AICc

A version of AIC corrected for small sample sizes

BIC or Bayesian information criterion

Similar to AIC, with a stronger penalty for including additional variables to the model

Mallows Cp

A variant of AIC developed by Colin Mallows

These are typically reported as in-sample metrics (i.e., on the training data), and data scientists using holdout data for model assessment do not need to worry about the differences among them or the underlying theory behind them.

How do we find the model that minimizes AIC or maximizes adjusted R^2? One way is to search through all possible models, an approach called *all subset regression*. This is computationally expensive and is not feasible for problems with large data and many variables. An attractive alternative is to use *stepwise regression*. It could start with a full model and successively drop variables that don't contribute meaningfully. This is called *backward elimination*. Alternatively one could start with a constant model and successively add variables (*forward selection*). As a third option we can also successively add and drop predictors to find a model that lowers AIC or adjusted R^2. The `MASS` package in R by Venables and Ripley offers a stepwise regression function called `stepAIC`:

```
library(MASS)
step_lm <- stepAIC(house_full, direction = "both")
step_lm
---
Call:
lm(formula = AdjSalePrice ~ SqFtTotLiving + Bathrooms + Bedrooms +
    BldgGrade + PropertyType + SqFtFinBasement + YrBuilt, data = house,
    na.action = na.omit)

Coefficients:
               (Intercept)              SqFtTotLiving
                 6.179e+06                  1.993e+02
                 Bathrooms                  Bedrooms
                 4.240e+04                 -5.195e+04
                 BldgGrade  PropertyTypeSingle Family
                 1.372e+05                  2.291e+04
    PropertyTypeTownhouse            SqFtFinBasement
                 8.448e+04                  7.047e+00
                   YrBuilt
                -3.565e+03
```

In Python, you can find an implementation for stepwise regression in the `SequentialFeatureSelector` class of the `mlxtend` package, which implements all three types. As the focus of the `scikit-learn` ecosystem is on prediction, the `mlxtend` implementation uses cross-validation to compare models rather than AIC or adjusted R^2. It is more flexible than `stepAIC` in that it allows you to choose the type of cross-validation and the scoring metric. Here we use 5-fold cross-validation and RMSE as the scoring metric:

```
from mlxtend.feature_selection import SequentialFeatureSelector

sfs_stepwise = SequentialFeatureSelector(LinearRegression(),
    k_features=(1, 11),
    forward=True, floating=True,  ❶
    cv=5, scoring="neg_root_mean_squared_error", n_jobs=-1)

sfs_stepwise.fit(X, house[outcome])
print(sfs_stepwise.k_feature_names_)
```

❶ These two arguments specify the type of stepwise regression: forward, backward, or both (floating).

In contrast to R's `stepAIC`, `SequentialFeatureSelector` is not returning a model itself. It is in fact fitting a transformer that will reduce the dimensionality of the data set and is used to select features as part of a modeling pipeline or workflow. Here is how we can combine feature selection and model fitting in a `Pipeline`:

```
from sklearn.pipeline import Pipeline

step_lm = Pipeline([
    ("sfs", SequentialFeatureSelector(LinearRegression(),
        k_features=(1, 12), forward=False, floating=True,
        cv=5, scoring="neg_root_mean_squared_error", n_jobs=-1)),
    ("model", LinearRegression()),
])
step_lm.fit(X, house[outcome])
```

The resulting `step_lm` is a model that on prediction takes the original data, reduces it to the selected features, and then applies the linear regression model.

The R function chose a model in which several variables were dropped from `house_full`: `SqFtLot`, `NbrLivingUnits`, `YrRenovated`, and `NewConstruction`. Due to the different validation approach, the `Python` code also drops `SqFtFinBasement`.

Simpler yet are *forward selection* and *backward selection*. In forward selection, you start with no predictors and add them one by one, at each step adding the predictor that has the largest contribution to R^2, and stopping when the contribution is no longer statistically significant. In backward selection, or *backward elimination*, you start with the full model and take away predictors that are not statistically significant until you are left with a model in which all predictors are statistically significant.

Penalized regression is similar in spirit to AIC. Instead of explicitly searching through a discrete set of models, the model-fitting equation incorporates a constraint that penalizes the model for too many variables (parameters). Rather than eliminating predictor variables entirely—as with stepwise, forward, and backward selection—penalized regression applies

the penalty by reducing coefficients, in some cases to near zero. Common penalized regression methods are *ridge regression* and *lasso regression*.

Stepwise regression and all subset regression are *in-sample* methods to assess and tune models. This means the model selection is possibly subject to overfitting (fitting the noise in the data) and may not perform as well when applied to new data. One common approach to avoid this is to use cross-validation to validate the models. In linear regression, overfitting is typically not a major issue, due to the simple (linear) global structure imposed on the data. For more sophisticated types of models, particularly iterative procedures that respond to local data structure, cross-validation is a very important tool; see "Cross-Validation" for details.

Weighted Regression

Weighted regression is used by statisticians for a variety of purposes; in particular, it is important for analysis of complex surveys. Data scientists may find weighted regression useful in two cases:

- Inverse-variance weighting when different observations have been measured with different precision, with the higher variance ones receiving lower weights.
- Analysis of data where rows represent multiple cases; the weight variable encodes how many original observations each row represents.

For example, with the housing data, older sales are less reliable than more recent sales. Using the `DocumentDate` to determine the year of the sale, we can compute a `Weight` as the number of years since 2005 (the beginning of the data).

R:

```
library(lubridate)
house$Year <- year(parse_date_time(house$DocumentDate, "m/d/y"))
house$Weight <- house$Year - 2005
```

Python:

```
house["Year"] = [int(date.split("-")[0]) for date in house.DocumentDate]
house["Weight"] = house.Year - 2005
```

We can compute a weighted regression with the `lm` function using the `weight` argument:

```
house_wt <- lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade,
  data = house, weight = Weight)
round(cbind(house_lm = house_lm$coefficients,
    house_wt = house_wt$coefficients), digits = 3)
```

```
                house_lm    house_wt
(Intercept)   -521871.368 -584189.329
SqFtTotLiving     228.831     245.024
SqFtLot            -0.060      -0.292
Bathrooms       -19442.840  -26085.970
Bedrooms        -47769.955  -53608.876
BldgGrade       106106.963  115242.435
```

The coefficients in the weighted regression are slightly different from those in the original regression.

Most models in `scikit-learn` accept weights as the keyword argument `sample_weight` in the call of the `fit` method:

```
predictors = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
"BldgGrade"]
outcome = "AdjSalePrice"

house_wt = LinearRegression()
house_wt.fit(house[predictors], house[outcome], sample_weight=house.Weight)
```

KEY IDEAS

- Multiple linear regression models the relationship between a response variable Y and multiple predictor variables $X_1, \ldots, X_p$.
- The most important metrics to evaluate a model are root mean squared error (RMSE) and R-squared (R^2).
- The standard error of the coefficients can be used to measure the reliability of a variable's contribution to a model.
- Stepwise regression is a way to automatically determine which variables should be included in the model.
- Weighted regression is used to give certain records more or less weight in fitting the equation.

Further Reading

An excellent treatment of cross-validation and resampling can be found in *An Introduction to Statistical Learning* by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (Springer, R edition 2021, Python edition 2023).

Prediction Using Regression

The primary purpose of regression in data science is prediction. This is useful to keep in mind, since regression, being an old and established statistical method, comes with baggage that is more relevant to its traditional role as a tool for explanatory modeling than to prediction.

KEY TERMS FOR PREDICTION USING REGRESSION

Prediction interval

An uncertainty interval around an individual predicted value

Extrapolation

Extension of a model beyond the range of the data used to fit it

The Dangers of Extrapolation

Regression models should not be used to extrapolate beyond the range of the data (leaving aside the use of regression for time-series forecasting). The model is valid only for predictor values for which the data has sufficient values (even in the case that sufficient data is available, there could be other problems—see "Regression Diagnostics"). As an extreme case, suppose `model_lm` is used to predict the value of a 5,000-square-foot empty lot. In such a case, all the predictors related to the building would have a value of 0, and the regression equation would yield an absurd prediction of –521,900 + 5,000 × –0.0605 = –$522,202. Why did this happen? The data contains only parcels with buildings—there are no records corresponding to vacant land. Consequently, the model has no information to tell it how to predict the sales price for vacant land.

Confidence and Prediction Intervals

Much of statistics involves understanding and measuring variability (uncertainty). The t-statistics and p-values reported in regression output deal with this in a formal way, which is sometimes useful for variable selection (see "Assessing the Model"). More useful metrics are confidence intervals, which are uncertainty intervals placed around regression coefficients and predictions. An easy way to understand this is via the bootstrap (see "The Bootstrap" for more details about the general bootstrap procedure). The

most common regression confidence intervals encountered in software output are those for regression parameters (coefficients). Here is a bootstrap algorithm for generating confidence intervals for regression parameters (coefficients) for a data set with *P* predictors and *n* records (rows):

1. Consider each row (including outcome variable) as a single "ticket" and place all the *n* tickets in a box.
2. Draw a ticket at random, record the values, and replace it in the box.
3. Repeat step 2 *n* times; you now have one bootstrap resample.
4. Fit a regression to the bootstrap sample, and record the estimated coefficients.
5. Repeat steps 2 through 4, say, 1,000 times.
6. You now have 1,000 bootstrap values for each coefficient; find the appropriate percentiles for each one (e.g., 5th and 95th for a 90% confidence interval).

You can use the `Boot` function in R to generate actual bootstrap confidence intervals for the coefficients, or you can simply use the formula-based intervals that are a routine R output. The conceptual meaning and interpretation are the same, and not of central importance to data scientists, because they concern the regression coefficients. Of greater interest to data scientists are intervals around predicted *y* values ($\widehat{Y}_i$). The uncertainty around $\widehat{Y}_i$ comes from two sources:

- Uncertainty about what the relevant predictor variables and their coefficients are (see the preceding bootstrap algorithm)
- Additional error inherent in individual data points

The individual data point error can be thought of as follows: even if we knew for certain what the regression equation was (e.g., if we had a huge number of records to fit it), the *actual* outcome values for a given set of

predictor values will vary. For example, several houses—each with 8 rooms, a 6,500-square-foot lot, 3 bathrooms, and a basement—might have different values. We can model this individual error with the residuals from the fitted values. The bootstrap algorithm for modeling both the regression model error and the individual data point error would look as follows:

1. Take a bootstrap sample from the data (spelled out in greater detail earlier).
2. Fit the regression, and predict the new value.
3. Take a single residual at random from the original regression fit, add it to the predicted value, and record the result.
4. Repeat steps 1 through 3, say, 1,000 times.
5. Find the 2.5th and the 97.5th percentiles of the results.

KEY IDEAS

- Extrapolation beyond the range of the data can lead to error.
- Confidence intervals quantify uncertainty around regression coefficients.
- Prediction intervals quantify uncertainty in individual predictions.
- Most software, R included, will produce prediction and confidence intervals in default or specified output, using formulas.
- The bootstrap can also be used to produce prediction and confidence intervals; the interpretation and idea are the same.

PREDICTION INTERVAL OR CONFIDENCE INTERVAL?

A prediction interval pertains to uncertainty around a single value, while a confidence interval pertains to a mean or other statistic calculated from multiple values. Thus, a prediction interval will typically be much wider than a confidence interval for the same value. We model this individual value error in the bootstrap model by selecting an individual residual to tack on to the predicted value. Which should you use? That depends on the context and the purpose of the analysis, but, in general, data scientists are interested in specific individual predictions, so a prediction interval would be more appropriate. Using a confidence interval when you should be using a prediction interval will greatly underestimate the uncertainty in a given predicted value.

Factor Variables in Regression

Factor variables, also termed *categorical* variables, take on a limited number of discrete values. For example, a loan purpose can be "debt consolidation," "wedding," "car," and so on. The binary (yes/no) variable, also called an *indicator* variable, is a special case of a factor variable. Regression requires numerical inputs, so factor variables need to be recoded to use in the model. The most common approach is to convert a variable into a set of binary *dummy* variables.

KEY TERMS FOR FACTOR VARIABLES

Dummy variables

Binary 0–1 variables derived by recoding factor data for use in regression and other models.

Reference coding

The most common type of coding used by statisticians, in which one level of a factor is used as a reference and other factors are compared to that level.

Synonym

treatment coding

One-hot encoder

A common type of coding used in the machine learning community in which all factor levels are retained. While useful for certain machine learning algorithms, this approach is not appropriate for multiple linear regression.

Deviation coding

A type of coding that compares each level against the overall mean as opposed to the reference level.

Synonym

sum contrasts

Dummy Variables Representation

In the King County housing data, there is a factor variable for the property type; a small subset of six records is shown here.

R:

```
head(house[, "PropertyType"])
---
Source: local data frame [6 x 1]

   PropertyType
         (fctr)
1     Multiplex
2 Single Family
3 Single Family
4 Single Family
5 Single Family
6     Townhouse
```

Python:

```
house.PropertyType.head()
```

There are three possible values: `Multiplex`, `Single Family`, and `Townhouse`. To use this factor variable, we need to convert it to a set of binary variables. We do this by creating a binary variable for each possible value of the factor variable. To do this in R, we use the `model.matrix` function:[3]

```
prop_type_dummies <- model.matrix(~ PropertyType - 1, data = house)
head(prop_type_dummies)
---
  PropertyTypeMultiplex PropertyTypeSingle Family PropertyTypeTownhouse
1                     1                         0                     0
2                     0                         1                     0
3                     0                         1                     0
4                     0                         1                     0
5                     0                         1                     0
6                     0                         0                     1
```

The function `model.matrix` converts a data frame into a matrix suitable to a linear model. The factor variable `PropertyType`, which has three distinct

levels, is represented as a matrix with three columns. In the machine learning community, this representation is referred to as *one-hot encoding* (see "One-Hot Encoder").

In Python, we can convert categorical variables to dummies using the `pandas` method `get_dummies`:

```
print(pd.get_dummies(house["PropertyType"]).head()) ❶
print(pd.get_dummies(house["PropertyType"], drop_first=True).head()) ❷
```

❶ By default, returns one-hot encoding of the categorical variable.

❷ The keyword argument `drop_first` will return $P - 1$ columns. Use this to avoid the problem of multicollinearity.

In certain machine learning algorithms, such as nearest neighbors and tree models, one-hot encoding is the standard way to represent factor variables (for example, see "Tree Models").

In the regression setting, a factor variable with P distinct levels is usually represented by a matrix with only $P - 1$ columns. This is because once you have defined the values for $P - 1$ binaries, the value for the Pth is known and could be considered redundant. Adding the Pth column will cause a multicollinearity error (unless you are fitting a regression with no intercept, see "Multicollinearity").

The default representation in R is to use the first factor level as a *reference* and interpret the remaining levels relative to that factor:

```
lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade + PropertyType, data = house)
---
Call:
lm(formula = AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade + PropertyType, data = house)

Coefficients:
              (Intercept)             SqFtTotLiving
                -4.468e+05                 2.234e+02
                   SqFtLot                Bathrooms
                -7.037e-02                -1.598e+04
```

```
                 Bedrooms                  BldgGrade
                -5.089e+04                1.094e+05
PropertyTypeSingle Family      PropertyTypeTownhouse
                -8.468e+04                -1.151e+05
```

The method `get_dummies` takes the optional keyword argument `drop_first` to exclude the first factor as *reference*:

```
predictors = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
              "BldgGrade", "PropertyType"]

X = pd.get_dummies(house[predictors], drop_first=True, dtype=int)

house_lm_factor = LinearRegression()
house_lm_factor.fit(X, house[outcome])

print(f"Intercept: {house_lm_factor.intercept_:.3f}")
print("Coefficients:")
for name, coef in zip(X.columns, house_lm_factor.coef_, strict=True):
    print(f" {name}: {coef}")
```

The output from the R regression shows two coefficients corresponding to `PropertyType`: `PropertyTypeSingle Family` and `PropertyTypeTownhouse`. There is no coefficient of `Multiplex` since it is implicitly defined when `PropertyTypeSingle Family == 0` and `PropertyTypeTownhouse == 0`. The coefficients are interpreted as relative to `Multiplex`, so a home that is `Single Family` is worth almost $85,000 less, and a home that is `Townhouse` is worth over $150,000 less.[4]

DIFFERENT FACTOR CODINGS

There are several different ways to encode factor variables, known as *contrast coding* systems. For example, *deviation coding*, also known as *sum contrasts*, compares each level against the overall mean. Another contrast is *polynomial coding*, which is appropriate for ordered factors; see the section "Ordered Factor Variables". With the exception of ordered factors, data scientists will generally not encounter any type of coding besides reference coding or one-hot encoder.

Factor Variables with Many Levels

Some factor variables can produce a huge number of binary dummies—zip codes are a factor variable, and there are 43,000 zip codes in the US. In such cases, it is useful to explore the data, and the relationships between predictor variables and the outcome, to determine whether useful information is contained in the categories. If so, you must further decide whether it is useful to retain all factors, or whether the levels should be consolidated.

In King County, there are 80 zip codes with a house sale:

```
table(house$ZipCode)
```

```
---
98001 98002 98003 98004 98005 98006 98007 98008 98010 98011 98014 98019
  358   180   241   293   133   460   112   291    56   163    85   242
98022 98023 98024 98027 98028 98029 98030 98031 98032 98033 98034 98038
  188   455    31   366   252   475   263   308   121   517   575   788
98039 98040 98042 98043 98045 98047 98050 98051 98052 98053 98055 98056
   47   244   641     1   222    48     7    32   614   499   332   402
98057 98058 98059 98065 98068 98070 98072 98074 98075 98077 98092 98102
    4   420   513   430     1    89   245   502   388   204   289   106
98103 98105 98106 98107 98108 98109 98112 98113 98115 98116 98117 98118
  671   313   361   296   155   149   357     1   620   364   619   492
98119 98122 98125 98126 98133 98136 98144 98146 98148 98155 98166 98168
  260   380   409   473   465   310   332   287    40   358   193   332
98177 98178 98188 98198 98199 98224 98288 98354
  216   266   101   225   393     3     4     9
```

The `value_counts` method of `pandas` data frames returns the same information:

```
pd.DataFrame(house["ZipCode"].value_counts()).transpose()
```

`ZipCode` is an important variable, since it is a proxy for the effect of location on the value of a house. Including all levels requires 79 coefficients corresponding to 79 degrees of freedom. The original model `house_lm` has only 5 degrees of freedom; see "Assessing the Model". Moreover, several zip codes have only one sale. In some problems, you can consolidate a zip code using the first two or three digits, corresponding to a submetropolitan

geographic region. For King County, almost all of the sales occur in 980xx or 981xx, so this doesn't help.

An alternative approach is to group the zip codes according to another variable, such as sale price. Even better is to form zip code groups using the residuals from an initial model. The following `dplyr` code in R consolidates the 80 zip codes into 5 groups based on the median of the residual from the `house_lm` regression:

```
zip_groups <- house %>%
  mutate(resid = residuals(house_lm)) %>%
  group_by(ZipCode) %>%
  summarize(med_resid = median(resid), cnt = n()) %>%
  arrange(med_resid) %>%
  mutate(
    cum_cnt = cumsum(cnt),
    ZipGroup = ntile(cum_cnt, 5),
  ) %>%
  select(ZipCode, ZipGroup)
house <- house %>%
  left_join(zip_groups, by = "ZipCode") %>%
  mutate(ZipGroup = as.factor(ZipGroup))
```

The median residual is computed for each zip, and the `ntile` function is used to split the zip codes, sorted by the median, into five groups. See "Confounding Variables" for an example of how this is used as a term in a regression improving upon the original fit.

In Python we can calculate this information as follows:

```
house = pd.read_csv("house_sales.csv", sep="\t")
predictors = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
"BldgGrade"]
outcome = "AdjSalePrice"
house_lm = LinearRegression()
house_lm.fit(house[predictors], house[outcome])

zip_groups = pd.DataFrame([
    *pd.DataFrame({
        "ZipCode": house["ZipCode"],
        "residual": house[outcome] - house_lm.predict(house[predictors]),
    })
```

```
    .groupby(["ZipCode"])[["ZipCode", "residual"]]
    .apply(lambda x: {
        "ZipCode": x.iloc[0, 0],
        "count": len(x),
        "median_residual": x.residual.median(),
    }),
]).sort_values("median_residual")
zip_groups["cum_count"] = np.cumsum(zip_groups["count"])
zip_groups["ZipGroup"] = pd.qcut(zip_groups["cum_count"], 5, labels=False,
                                 retbins=False)

to_join = zip_groups[["ZipCode", "ZipGroup"]].set_index("ZipCode")
house = house.join(to_join, on="ZipCode")
house["ZipGroup"] = house["ZipGroup"].astype("category")
```

The concept of using the residuals to help guide the regression fitting is a fundamental step in the modeling process; see "Regression Diagnostics".

Ordered Factor Variables

Some factor variables reflect levels of a factor; these are termed *ordered factor variables* or *ordered categorical variables*. For example, the loan grade could be A, B, C, and so on—each grade carries more risk than the prior grade. Often, ordered factor variables can be converted to numerical values and used as is. For example, the variable `BldgGrade` is an ordered factor variable. Several of the types of grades are shown in Table 4-1. While the grades have specific meaning, the numeric value is ordered from low to high, corresponding to higher-grade homes. With the regression model `house_lm`, fit in "Multiple Linear Regression", `BldgGrade` was treated as a numeric variable.

Table 4-1. Building grades and numeric equivalents

Value	Description
1	Cabin
2	Substandard
5	Fair
10	Very good
12	Luxury
13	Mansion

Treating ordered factors as a numeric variable preserves the information contained in the ordering that would be lost if it were converted to a factor.

KEY IDEAS

- Factor variables need to be converted into numeric variables for use in a regression.
- The most common method to encode a factor variable with P distinct values is to represent them using P – 1 dummy variables.
- A factor variable with many levels, even in very big data sets, may need to be consolidated into a variable with fewer levels.
- Some factors have levels that are ordered and can be represented as a single numeric variable.

Interpreting the Regression Equation

In data science, the most important use of regression is to predict some dependent (outcome) variable. In some cases, however, gaining insight from the equation itself to understand the nature of the relationship between the predictors and the outcome can be of value. This section provides guidance on examining the regression equation and interpreting it.

KEY TERMS FOR INTERPRETING THE REGRESSION EQUATION

Correlated variables

Variables that tend to move in the same direction—when one goes up so does the other, and vice versa (with negative correlation, when one goes up the other goes down). When the predictor variables are highly correlated, it is difficult to interpret the individual coefficients.

Multicollinearity

When the predictor variables have perfect, or near-perfect, correlation, the regression can be unstable or impossible to compute.

Synonym

collinearity

Confounding variables

An important predictor that, when omitted, leads to spurious relationships in a regression equation.

Main effects

The relationship between a predictor and the outcome variable, independent of other variables.

Interactions

An interdependent relationship between two or more predictors and the response.

Correlated Predictors

In multiple regression, the predictor variables are often correlated with each other. As an example, examine the regression coefficients for the model `step_lm`, fit in "Model Selection and Stepwise Regression".

R:

```
step_lm$coefficients
```

```
---
             (Intercept)            SqFtTotLiving                Bathrooms
            6.178645e+06             1.992776e+02             4.239616e+04
                Bedrooms                BldgGrade PropertyTypeSingle Family
           -5.194738e+04             1.371596e+05             2.291206e+04
PropertyTypeTownhouse          SqFtFinBasement                  YrBuilt
            8.447916e+04             7.046975e+00            -3.565425e+03
```

Python:

```
print(f"Intercept: {step_lm.named_steps['model'].intercept_:.3f}")
print("Coefficients:")
for name, coef in zip(step_lm.named_steps["sfs"].k_feature_names_,
                      step_lm.named_steps["model"].coef_, strict=True):
    print(f"  {name}: {coef}")
```

```
---
Intercept: 6259055.855
Coefficients:
  SqFtTotLiving: 201.65789971342235
  Bathrooms: 42787.933811708994
  Bedrooms: -51742.8382076272
  BldgGrade: 136309.59662237618
  PropertyType_Single Family: 24105.25018215709
  PropertyType_Townhouse: 87589.26527129347
  YrBuilt: -3605.8233277869685
```

The coefficient for `Bedrooms` is negative! This implies that adding a bedroom to a house will reduce its value. How can this be? This is because the predictor variables are correlated: larger houses tend to have more bedrooms, and it is the size that drives house value, not the number of bedrooms. Consider two homes of the exact same size: it is reasonable to

expect that a home with more but smaller bedrooms would be considered less desirable.

Having correlated predictors can make it difficult to interpret the sign and value of regression coefficients (and can inflate the standard error of the estimates). The variables for bedrooms, house size, and number of bathrooms are all correlated. This is illustrated by the following example in R, which fits another regression removing the variables `SqFtTotLiving`, `SqFtFinBasement`, and `Bathrooms` from the equation:

```
update(step_lm, . ~ . - SqFtTotLiving - SqFtFinBasement - Bathrooms)
---
Call:
lm(formula = AdjSalePrice ~ Bedrooms + BldgGrade + PropertyType +
    YrBuilt, data = house, na.action = na.omit)

Coefficients:
               (Intercept)                   Bedrooms
                   4913973                      27151
                 BldgGrade  PropertyTypeSingle Family
                    248998                     -19898
    PropertyTypeTownhouse                    YrBuilt
                    -47355                      -3212
```

The `update` function can be used to add or remove variables from a model. Now the coefficient for bedrooms is positive—in line with what we would expect (though it is really acting as a proxy for house size, now that those variables have been removed).

In Python, there is no equivalent to R's `update` function. The easiest approach is to adjust the list of predictors and refit the model with the modified predictor list:

```
predictors = ["Bedrooms", "BldgGrade", "PropertyType", "YrBuilt"]
outcome = "AdjSalePrice"

X = pd.get_dummies(house[predictors], drop_first=True, dtype=int)

reduced_lm = LinearRegression()
reduced_lm.fit(X, house[outcome])
```

Correlated variables are only one issue with interpreting regression coefficients. In the `house_lm` model from "Example: King County Housing Data", there is no variable to account for the location of the home, and the model is mixing together very different types of regions. Location may be a *confounding* variable; see "Confounding Variables" for further discussion.

Multicollinearity

An extreme case of correlated variables produces multicollinearity—a condition in which there is redundance among the predictor variables. Perfect multicollinearity occurs when one predictor variable can be expressed as a linear combination of others. Multicollinearity occurs when:

- A variable is included multiple times by error.
- When P dummies, instead of $P - 1$ dummies, are created from a factor variable (see "Factor Variables in Regression").
- Two variables are nearly perfectly correlated with one another.

Multicollinearity in regression must be addressed—variables should be removed until the multicollinearity is gone. A regression does not have a well-defined solution in the presence of perfect multicollinearity. Many software packages, including R and Python, automatically handle certain types of multicollinearity. For example, if `SqFtTotLiving` is included twice in the regression of the `house` data, the results are the same as for the `house_lm` model. In the case of nonperfect multicollinearity, the software may obtain a solution, but the results may be unstable.

MULTICOLLINEARITY IN NONLINEAR METHODS

Multicollinearity is not such a problem for nonlinear regression methods like trees, clustering, and nearest-neighbors, and in such methods it may be advisable to retain P dummies (instead of $P - 1$). That said, even in those methods, nonredundancy in predictor variables is still a virtue.

Confounding Variables

With correlated variables, the problem is one of commission: including different variables that have a similar predictive relationship with the response. With *confounding variables*, the problem is one of omission: an important variable is not included in the regression equation. Naive interpretation of the equation coefficients can lead to invalid conclusions.

Take, for example, the King County regression equation `house_lm` from "Example: King County Housing Data". The regression coefficients of `SqFtLot`, `Bathrooms`, and `Bedrooms` are all negative. The original regression model does not contain a variable to represent location—a very important predictor of house price. To model location, include a variable `ZipGroup` that categorizes the zip code into one of five groups, from least expensive (1) to most expensive (5):[5]

```
lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade + PropertyType + ZipGroup, data = house,
  na.action = na.omit)
---
Coefficients:
              (Intercept)             SqFtTotLiving
               -6.666e+05                 2.106e+02
                  SqFtLot                  Bathrooms
                4.550e-01                 5.928e+03
                 Bedrooms                  BldgGrade
               -4.168e+04                 9.854e+04
PropertyTypeSingle Family     PropertyTypeTownhouse
                1.932e+04                -7.820e+04
                ZipGroup2                 ZipGroup3
                5.332e+04                 1.163e+05
                ZipGroup4                 ZipGroup5
                1.784e+05                 3.384e+05
```

The same model in Python:

```
predictors = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
              "BldgGrade", "PropertyType", "ZipGroup"]
outcome = "AdjSalePrice"

X = pd.get_dummies(house[predictors], drop_first=True, dtype=int)
```

```
confounding_lm = LinearRegression()
confounding_lm.fit(X, house[outcome])

print(f"Intercept: {confounding_lm.intercept_:.3f}")
print("Coefficients:")
for name, coef in zip(X.columns, confounding_lm.coef_, strict=True):
    print(f" {name}: {coef}")
```

`ZipGroup` is clearly an important variable: a home in the most expensive zip code group is estimated to have a higher sales price by almost $340,000. The coefficients of `SqFtLot` and `Bathrooms` are now positive, and adding a bathroom increases the sale price by $5,928.

The coefficient for `Bedrooms` is still negative. While this is unintuitive, this is a well-known phenomenon in real estate. For homes of the same livable area and number of bathrooms, having more and therefore smaller bedrooms is associated with less valuable homes.

Interactions and Main Effects

Statisticians like to distinguish between *main effects*, or independent variables, and the *interactions* between the main effects. Main effects are what are often referred to as the *predictor variables* in the regression equation. An implicit assumption when only main effects are used in a model is that the relationship between a predictor variable and the response is independent of the other predictor variables. This is often not the case.

For example, the model fit to the King County Housing Data in "Confounding Variables" includes several variables as main effects, including `ZipCode`. Location in real estate is everything, and it is natural to presume that the relationship between, say, house size and the sale price depends on location. A big house built in a low-rent district is not going to retain the same value as a big house built in an expensive area. You include interactions between variables in R using the `*` operator. For the King County data, the following fits an interaction between `SqFtTotLiving` and `ZipGroup`:

```
lm(formula = AdjSalePrice ~ SqFtTotLiving * ZipGroup + SqFtLot +
    Bathrooms + Bedrooms + BldgGrade + PropertyType, data = house,
  na.action = na.omit)
---
Coefficients:
               (Intercept)              SqFtTotLiving
                -4.853e+05                  1.148e+02
                  ZipGroup2                  ZipGroup3
                -1.113e+04                  2.032e+04
                  ZipGroup4                  ZipGroup5
                 2.050e+04                 -1.499e+05
                    SqFtLot                  Bathrooms
                 6.869e-01                 -3.619e+03
                   Bedrooms                  BldgGrade
                -4.180e+04                  1.047e+05
PropertyTypeSingle Family      PropertyTypeTownhouse
                 1.357e+04                 -5.884e+04
  SqFtTotLiving:ZipGroup2    SqFtTotLiving:ZipGroup3
                 3.260e+01                  4.178e+01
  SqFtTotLiving:ZipGroup4    SqFtTotLiving:ZipGroup5
                 6.934e+01                  2.267e+02
```

The resulting model has four new terms: `SqFtTotLiving:ZipGroup2`, `SqFtTotLiving:ZipGroup3`, and so on.

In Python, we need to use the `statsmodels` package to train linear regression models with interactions. This package was designed similarly to R and allows defining models using a formula interface:

```
import statsmodels.formula.api as smf

model = smf.ols(formula="AdjSalePrice ~ SqFtTotLiving*ZipGroup + SqFtLot + "
     "Bathrooms + Bedrooms + BldgGrade + PropertyType", data=house)
results = model.fit()
results.summary()
```

The `statsmodels` package takes care of categorical variables (e.g., `ZipGroup[T.1]`, `PropertyType[T.Single Family]`) and interaction terms (e.g., `SqFtTotLiving:ZipGroup[T.1]`).

Location and house size appear to have a strong interaction. For a home in the lowest `ZipGroup`, the slope is the same as the slope for the main effect

`SqFtTotLiving`, which is $115 per square foot (this is because R uses *reference* coding for factor variables; see "Factor Variables in Regression"). For a home in the highest `ZipGroup`, the slope is the sum of the main effect plus `SqFtTotLiving:ZipGroup5`, or $115 + $227 = $342 per square foot. In other words, adding a square foot in the most expensive zip code group boosts the predicted sale price by a factor of almost three, compared to the average boost from adding a square foot in the lowest zip code group.

MODEL SELECTION WITH INTERACTION TERMS

In problems involving many variables, it can be challenging to decide which interaction terms should be included in the model. Several different approaches are commonly taken:

- In some problems, prior knowledge and intuition can guide the choice of which interaction terms to include in the model.
- Stepwise selection (see "Model Selection and Stepwise Regression") can be used to sift through the various models.
- Penalized regression can automatically fit to a large set of possible interaction terms.
- Perhaps the most common approach is to use *tree models*, as well as their descendants, *random forest* and *gradient boosted trees*. This class of models automatically searches for optimal interaction terms; see "Tree Models".

KEY IDEAS

- Because of correlation between predictors, care must be taken in the interpretation of the coefficients in multiple linear regression.
- Multicollinearity can cause numerical instability in fitting the regression equation.
- A confounding variable is an important predictor that is omitted from a model and can lead to a regression equation with spurious relationships.
- An interaction term between two variables is needed if the relationship between the variables and the response is interdependent.

Regression Diagnostics

In explanatory modeling (i.e., in a research context), various steps, in addition to the metrics mentioned previously (see "Assessing the Model"), are taken to assess how well the model fits the data; most are based on analysis of the residuals. These steps do not directly address predictive accuracy, but they can provide useful insight in a predictive setting.

KEY TERMS FOR REGRESSION DIAGNOSTICS

Standardized residuals

Residuals divided by the standard error of the residuals.

Outliers

Records (or outcome values) that are distant from the rest of the data (or the predicted outcome).

Influential value

A value or record whose presence or absence makes a big difference in the regression equation.

Leverage

The degree of influence that a single record has on a regression equation.

Synonym

hat-value

Nonnormal residuals

Nonnormally distributed residuals can invalidate some technical requirements of regression but are usually not a concern in data science.

Heteroskedasticity

When some ranges of the outcome experience residuals with higher variance (may indicate a predictor missing from the equation).

Partial residual plots

A diagnostic plot to illuminate the relationship between the outcome variable and a single predictor.

Synonym

added variables plot

Outliers

Generally speaking, an extreme value, also called an *outlier*, is one that is distant from most of the other observations. Just as outliers need to be handled for estimates of location and variability (see "Estimates of Location" and "Estimates of Variability"), outliers can cause problems with regression models. In regression, an outlier is a record whose actual *y* value is distant from the predicted value. You can detect outliers by examining the *standardized residual*, which is the residual divided by the standard error of the residuals.

There is no statistical theory that separates outliers from nonoutliers. Rather, there are (arbitrary) rules of thumb for how distant from the bulk of the data an observation needs to be in order to be called an outlier. For example, with the boxplot, outliers are those data points that are too far above or below the box boundaries (see "Percentiles and Boxplots"), where "too far" = "more than 1.5 times the interquartile range." In regression, the standardized residual is the metric that is typically used to determine whether a record is classified as an outlier. Standardized residuals can be interpreted as "the number of standard errors away from the regression line."

Let's fit a regression to the King County house sales data for all sales in zip code 98105 in R:

```
house_98105 <- house %>% filter(ZipCode == 98105)
lm_98105 <- lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
    Bedrooms + BldgGrade, data = house_98105)
```

In Python:

```
house_98105 = house.loc[house["ZipCode"] == 98105, ]

predictors = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
"BldgGrade"]
outcome = "AdjSalePrice"

house_outlier = sm.OLS(house_98105[outcome],
                       sm.add_constant(house_98105[predictors]))
result_98105 = house_outlier.fit()
```

We extract the standardized residuals in R using the `rstandard` function and obtain the index of the smallest residual using the `order` function:

```
sresid <- rstandard(lm_98105)
idx <- order(sresid)
sresid[idx[1]]
---
    20429
-4.326732
```

In `statsmodels`, use `OLSInfluence` to analyze the residuals:

```
from statsmodels.stats.outliers_influence import OLSInfluence

influence = OLSInfluence(result_98105)
sresiduals = influence.resid_studentized_internal
sresiduals.idxmin(), sresiduals.min()
```

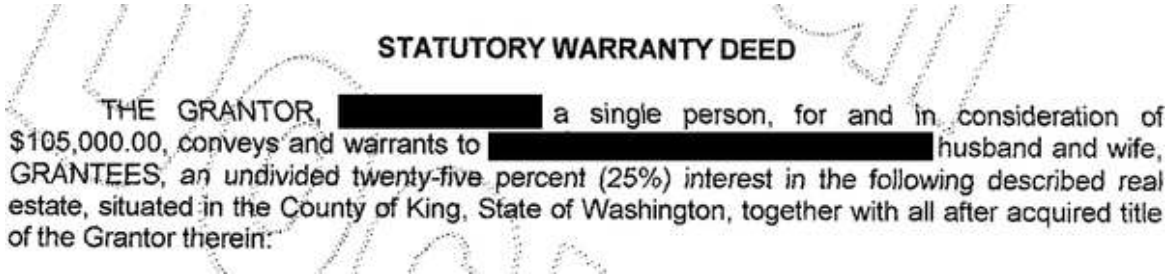
The biggest overestimate from the model is more than four standard errors above the regression line, corresponding to an overestimate of $757,754. The original data record corresponding to this outlier is as follows in R:

```
house_98105[idx[1], c("AdjSalePrice", "SqFtTotLiving", "SqFtLot",
    "Bathrooms", "Bedrooms", "BldgGrade")]
---
AdjSalePrice SqFtTotLiving SqFtLot Bathrooms Bedrooms BldgGrade
         (dbl)         (int)   (int)     (dbl)    (int)     (int)
20429   119748          2900    7276         3        6         7
```

In Python:

```
outlier = house_98105.loc[sresiduals.idxmin(), :]
print("AdjSalePrice", outlier[outcome])
print(outlier[predictors])
```

In this case, it appears that there is something wrong with the record: a house of that size typically sells for much more than $119,748 in that zip code. Figure 4-4 shows an excerpt from the statutory deed from this sale: it is clear that the sale involved only partial interest in the property. In this case, the outlier corresponds to a sale that is anomalous and should not be included in the regression. Outliers could also be the result of other problems, such as a "fat-finger" data entry or a mismatch of units (e.g., reporting a sale in thousands of dollars rather than simply in dollars).



STATUTORY WARRANTY DEED

THE GRANTOR, [redacted] a single person, for and in consideration of $105,000.00, conveys and warrants to [redacted] husband and wife, GRANTEES, an undivided twenty-five percent (25%) interest in the following described real estate, situated in the County of King, State of Washington, together with all after acquired title of the Grantor therein:

Figure 4-4. Statutory warranty deed for the largest negative residual

For big data problems, outliers are generally not a problem in fitting the regression to be used in predicting new data. However, outliers are central to anomaly detection, where finding outliers is the whole point. The outlier could also correspond to a case of fraud or an accidental action. In any case, detecting outliers can be a critical business need.

Influential Values

A value whose absence would significantly change the regression equation is termed an *influential observation*. In regression, such a value need not be associated with a large residual. As an example, consider the regression lines in Figure 4-5. The solid line corresponds to the regression with all the data, while the dashed line corresponds to the regression with the point in the upper-right corner removed. Clearly, that data value has a huge

influence on the regression even though it is not associated with a large outlier (from the full regression). This data value is considered to have high *leverage* on the regression.

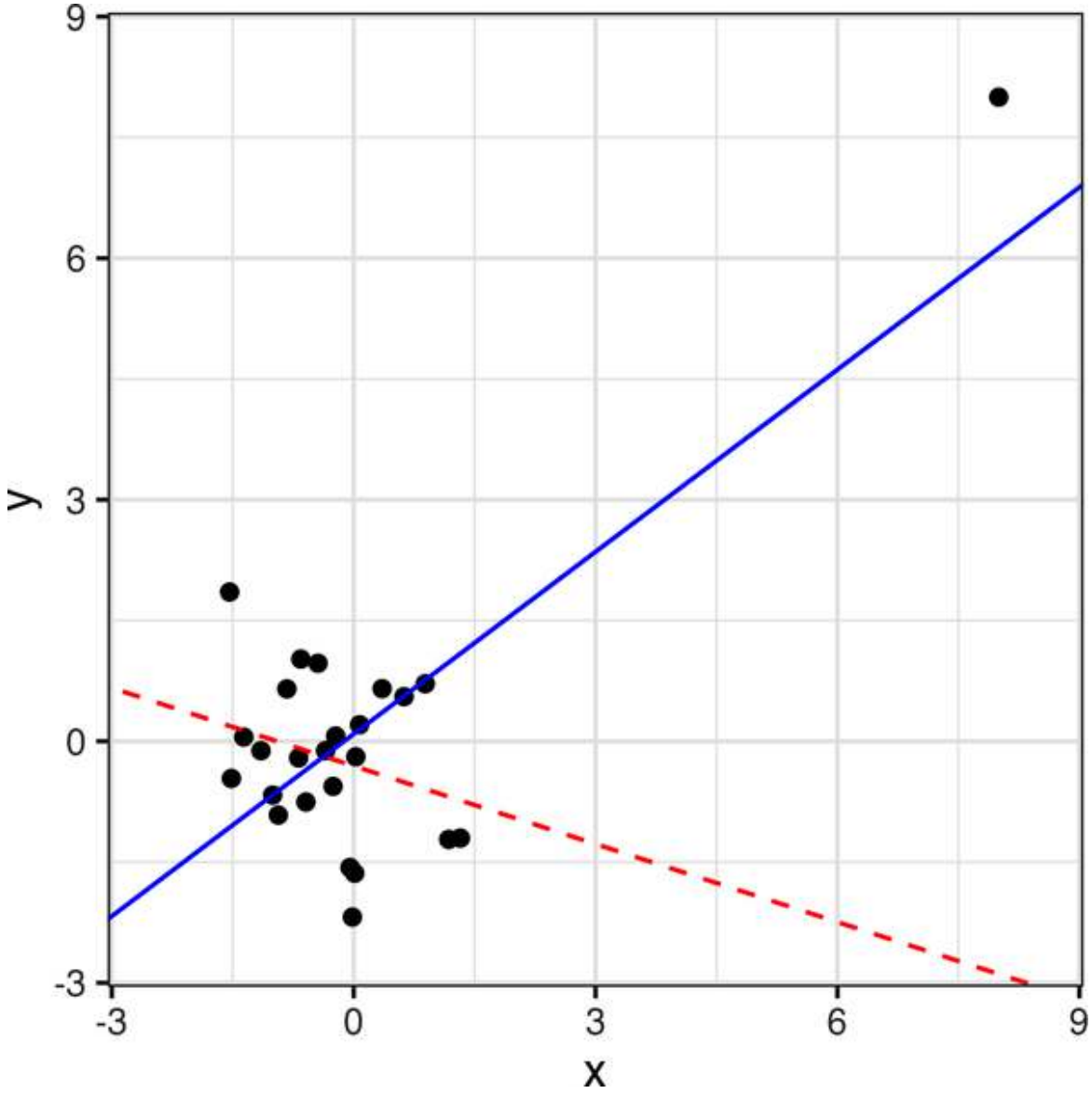


Figure 4-5. An example of an influential data point in regression

In addition to standardized residuals (see "Outliers"), statisticians have developed several metrics to determine the influence of a single record on a regression. A common measure of leverage is the *hat-value*; values above $2(P + 1)/n$ indicate a high-leverage data value.[6]

Another metric is *Cook's distance*, which defines influence as a combination of leverage and residual size. A rule of thumb is that an observation has high influence if Cook's distance exceeds $4/(n - P - 1)$.

An *influence plot* or *bubble plot* combines standardized residuals, the hat-value, and Cook's distance in a single plot. Figure 4-6 shows the influence plot for the King County house data and can be created by the following R code:

```
df_influential <- tibble(
  std_resid = rstandard(lm_98105),
  cooks_d = cooks.distance(lm_98105),
  hat_values = hatvalues(lm_98105),
)

p <- 6 # Number of model parameters (e.g., intercept + 2 predictors)
D_i <- 0.08 # Target Cook's Distance
cooks_curve_data <- tibble(
  t = seq(-10, 10, length.out = 1000),
  cooks_d = 0.08,
) %>%
  mutate(
    hat_values = 1 / (1 + t^2),
    std_resid = sqrt(D_i * p) * t
  )

ggplot(df_influential, aes(x = hat_values, y = std_resid,
    color = cooks_d > 0.08, shape = cooks_d > 0.08)) +
  geom_path(aes(size = NULL, color = NULL, shape = NULL),
    data = cooks_curve_data, color = "grey", linewidth = 0.5) +
  geom_hline(yintercept = c(-2.5, 2.5), linetype = 2) +
  geom_point(aes(size = 10 * sqrt(cooks_d))) +
  scale_shape_manual(values = c(1, 19)) +
  guides(color = "none", shape = "none", size = "none") +
  coord_cartesian(xlim = c(0, 0.3), ylim = c(-6, 6), expand = FALSE)
```

Here is the Python code to create a similar figure:

```
import matplotlib.pyplot as plt

influence = OLSInfluence(result_98105)
fig, ax = plt.subplots(figsize=(5, 5))
ax.axhline(-2.5, linestyle="--", color="C1")
ax.axhline(2.5, linestyle="--", color="C1")
```

```
ax.scatter(influence.hat_matrix_diag, influence.resid_studentized_internal,
           s=1000 * np.sqrt(influence.cooks_distance[0]),
           alpha=0.5)
ax.set_xlabel("hat values")
ax.set_ylabel("studentized residuals")
```

There are apparently several data points that exhibit large influence in the regression. Cook's distance can be computed using the function `cooks.distance`, and you can use `hatvalues` to compute the diagnostics. The hat-values are plotted on the x-axis, the residuals are plotted on the y-axis, and the size of the points is related to the value of Cook's distance.

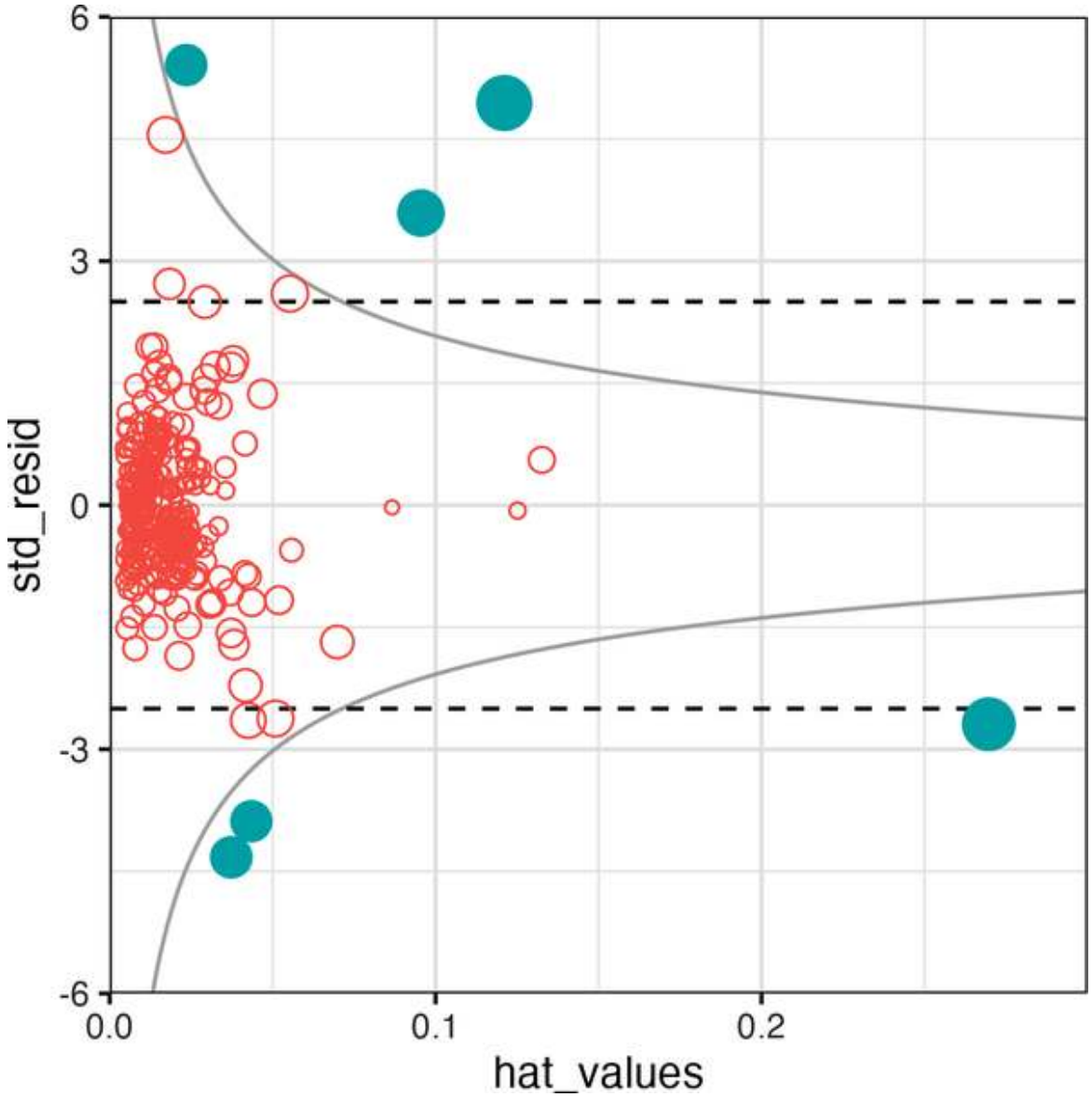


Figure 4-6. A plot to determine which observations have high influence; points with Cook's distance greater than 0.08 are shown as filled circles

Table 4-2 compares the regression with the full data set and with highly influential data points removed (Cook's distance > 0.08).

The regression coefficient for `Bathrooms` changes quite dramatically.[7]

Table 4-2. Comparison of regression coefficients with the full data and with influential data removed

	Original	Influential removed
(Intercept)	–772,550	–647,137
SqFtTotLiving	210	230
SqFtLot	39	33
Bathrooms	2,282	–16,132
Bedrooms	–26,320	–22,888
BldgGrade	130,000	114,871

For purposes of fitting a regression that reliably predicts future data, identifying influential observations is useful only in smaller data sets. For regressions involving many records, it is unlikely that any one observation will carry sufficient weight to cause extreme influence on the fitted equation (although the regression may still have big outliers). For purposes of anomaly detection, though, identifying influential observations can be very useful.

Heteroskedasticity, Nonnormality, and Correlated Errors

Statisticians pay considerable attention to the distribution of the residuals. It turns out that OLS (see "Least Squares") are unbiased, and in some cases are the "optimal" estimator, under a wide range of distributional assumptions. This means that in most problems, data scientists do not need to be too concerned with the distribution of the residuals.

The distribution of the residuals is relevant mainly for the validity of formal statistical inference (hypothesis tests and p-values), which is of minimal importance to data scientists concerned mainly with predictive accuracy. Normally distributed errors are a sign that the model is complete; errors that are not normally distributed indicate the model may be missing something. For formal inference to be fully valid, the residuals are assumed to be normally distributed, have the same variance, and be independent. One area where this may be of concern to data scientists is the standard calculation of confidence intervals for predicted values, which are based upon the assumptions about the residuals (see "Confidence and Prediction Intervals").

Heteroskedasticity is the lack of constant residual variance across the range of the predicted values. In other words, errors are greater for some portions of the range than for others. Visualizing the data is a convenient way to analyze residuals.

The following code in R plots the absolute residuals versus the predicted values for the `lm_98105` regression fit in "Outliers":

```
df <- data.frame(resid = residuals(lm_98105), pred = predict(lm_98105))
ggplot(df, aes(x = pred, y = abs(resid))) +
  geom_point() +
  geom_smooth(formula = y ~ x, method = "loess") +
  scale_x_continuous(labels = function(x) format(x, scientific = FALSE))
```

Figure 4-7 shows the resulting plot. Using `geom_smooth`, it is easy to superpose a smooth of the absolute residuals. The function calls the `loess` method (locally estimated scatterplot smoothing) to produce a smoothed estimate of the relationship between the variables on the x-axis and y-axis in a scatterplot (see "Scatterplot Smoothers").

In Python, the `seaborn` package has the `regplot` function that creates a similar figure:

```
import seaborn as sns

fig, ax = plt.subplots(figsize=(5, 5))
```

```
sns.regplot(x=result_98105.fittedvalues, y=np.abs(result_98105.resid),
            scatter_kws={"alpha": 0.25}, line_kws={"color": "C1"},
            lowess=True, ax=ax)
ax.set_xlabel("predicted")
ax.set_ylabel("abs(residual)")
```

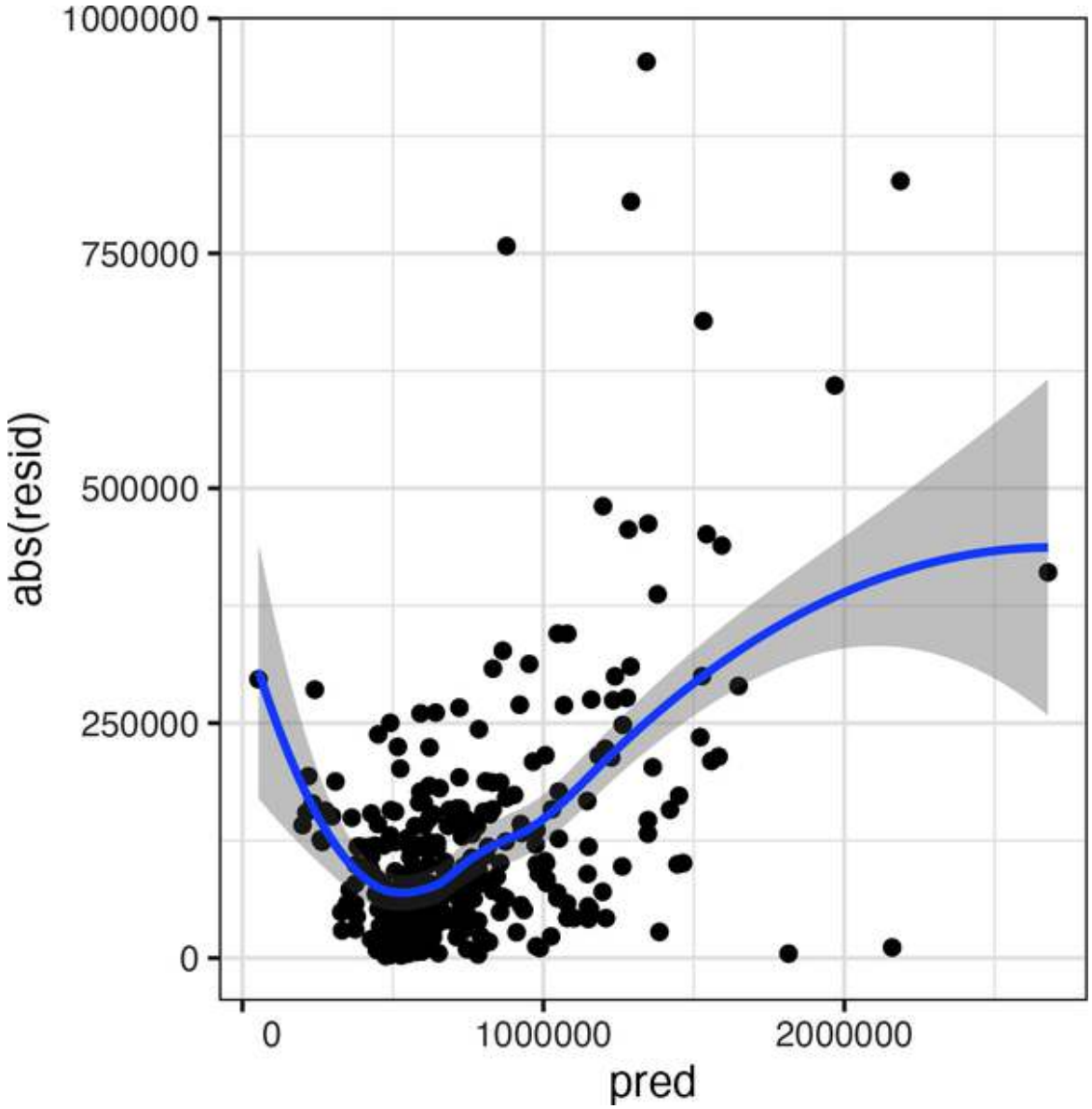


Figure 4-7. A plot of the absolute value of the residuals versus the predicted values

Evidently, the variance of the residuals tends to increase for higher-valued homes but is also large for lower-valued homes. This plot indicates that `lm_98105` has *heteroskedastic* errors.

WHY WOULD A DATA SCIENTIST CARE ABOUT HETEROSKEDASTICITY?

Heteroskedasticity indicates that prediction errors differ for different ranges of the predicted value, and may suggest an incomplete model. For example, the heteroskedasticity in `lm_98105` may indicate that the regression has left something unaccounted for in high- and low-range homes.

Figure 4-8 is a histogram of the standardized residuals for the `lm_98105` regression. The distribution has decidedly longer tails than the normal distribution and exhibits mild skewness toward larger residuals.

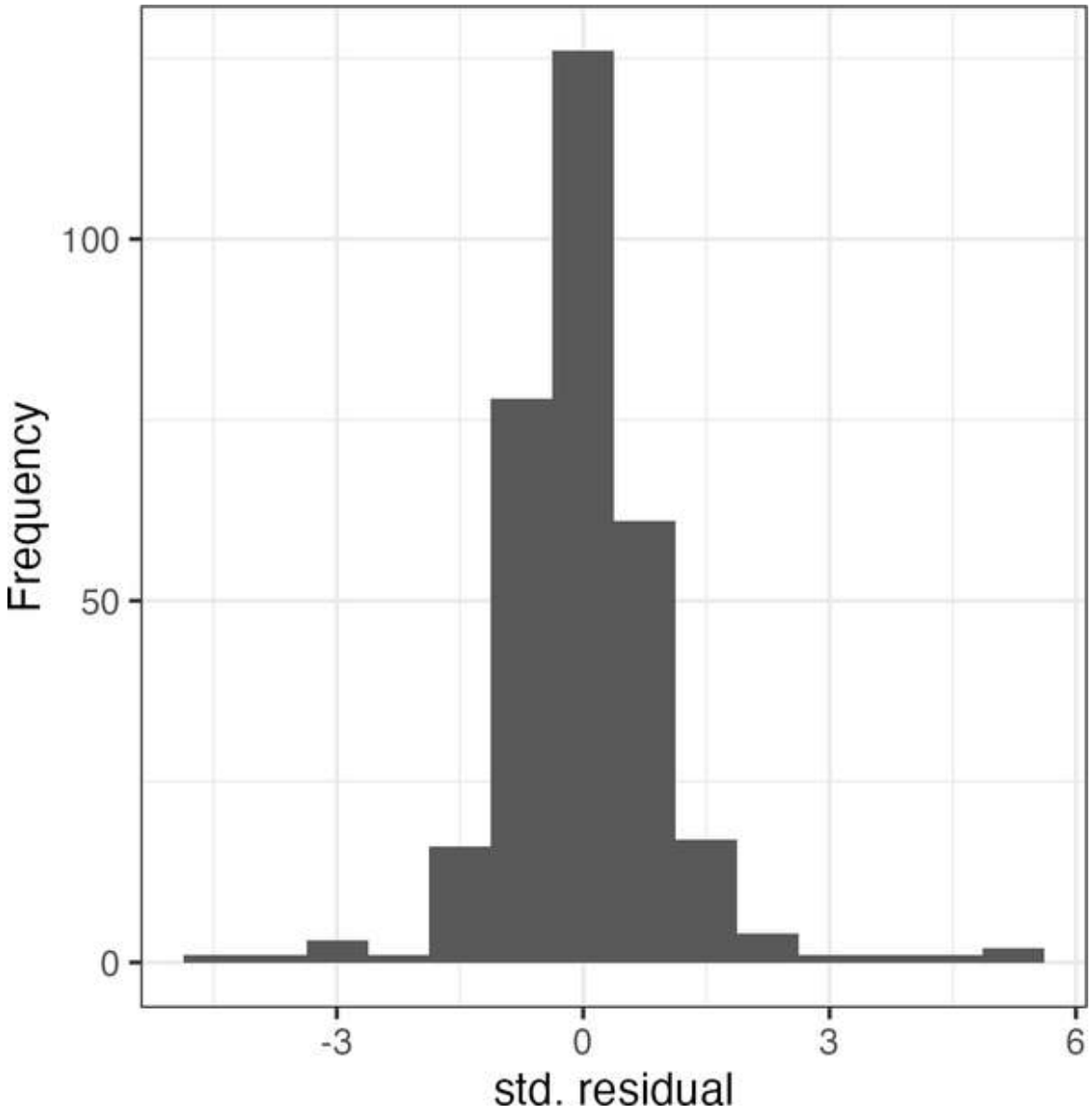


Figure 4-8. A histogram of the residuals from the regression of the housing data

Statisticians may also check the assumption that the errors are independent. This is particularly true for data that is collected over time or space. The *Durbin-Watson* statistic can be used to detect if there is significant autocorrelation in a regression involving time-series data. If the errors from a regression model are correlated, then this information can be useful in making short-term forecasts and should be built into the model. See *Practical Time Series Forecasting with R*, 2nd ed., by Galit Shmueli and Kenneth Lichtendahl (Axelrod Schnall, 2018) to learn more about how to build autocorrelation information into regression models for time-series

data. If longer-term forecasts or explanatory models are the goal, excess autocorrelated data at the microlevel may distract. In that case, smoothing, or less granular collection of data in the first place, may be in order.

Even though a regression may violate one of the distributional assumptions, should we care? Most often in data science, the interest is primarily in predictive accuracy, so some review of heteroskedasticity may be in order. You may discover that there is some signal in the data that your model has not captured. However, satisfying distributional assumptions simply for the sake of validating formal statistical inference (p-values, F-statistics, etc.) is not that important for the data scientist.

SCATTERPLOT SMOOTHERS

Regression is about modeling the relationship between the response and predictor variables. In evaluating a regression model, it is useful to use a *scatterplot smoother* to visually highlight relationships between two variables.

For example, in Figure 4-7, a smooth of the relationship between the absolute residuals and the predicted value shows that the variance of the residuals depends on the value of the residual. In this case, the `loess` function was used; `loess` works by repeatedly fitting a series of local regressions to contiguous subsets to come up with a smooth. While `loess` is probably the most commonly used smoother, other scatterplot smoothers are available in R, such as super smooth (`supsmu`) and kernel smoothing (`ksmooth`). In Python, we can find additional smoothers in `scipy` (`wiener` or `sav`) and `statsmodels` (`kernel_regression`). For the purposes of evaluating a regression model, there is typically no need to worry about the details of these scatterplot smooths.

Partial Residual Plots and Nonlinearity

Partial residual plots are a way to visualize how well the estimated fit explains the relationship between a predictor and the outcome. The basic

idea of a partial residual plot is to isolate the relationship between a predictor variable and the response, *taking into account all of the other predictor variables*. A partial residual might be thought of as a "synthetic outcome" value, combining the prediction based on a single predictor with the actual residual from the full regression equation. A partial residual for predictor X_i is the ordinary residual plus the regression term associated with X_i:

$$\text{Partial residual} = \text{Residual} + \hat{b}_i X_i$$

where $\hat{b}_i$ is the estimated regression coefficient. The `predict` function in R has an option to return the individual regression terms $\hat{b}_i X_i$:

```
terms <- predict(lm_98105, type = "terms")
partial_resid <- resid(lm_98105) + terms
```

The partial residual plot displays the X_i predictor on the x-axis and the partial residuals on the y-axis. Using `ggplot2` makes it easy to superpose a smooth of the partial residuals:

```
df <- data.frame(SqFtTotLiving = house_98105[, "SqFtTotLiving"],
  Terms = terms[, "SqFtTotLiving"],
  PartialResid = partial_resid[, "SqFtTotLiving"])
ggplot(df, aes(SqFtTotLiving, PartialResid)) +
  geom_point(shape = 1) +
  scale_shape(solid = FALSE) +
  geom_smooth(linetype = 2) +
  geom_line(aes(SqFtTotLiving, Terms))
```

The `statsmodels` package has the method `sm.graphics.plot_ccpr` that creates a similar partial residual plot:

```
sm.graphics.plot_ccpr(result_98105, "SqFtTotLiving")
```

The R and Python graphs differ by a constant shift. In R, a constant is added so that the mean of the terms is zero.

The resulting plot is shown in Figure 4-9. The partial residual is an estimate of the contribution that `SqFtTotLiving` adds to the sales price. The relationship between `SqFtTotLiving` and the sales price is evidently nonlinear (dashed line). The regression line (solid line) underestimates the sales price for homes less than 1,000 square feet and overestimates the price for homes between 2,000 and 3,000 square feet. There are too few data points above 4,000 square feet to draw conclusions for those homes.

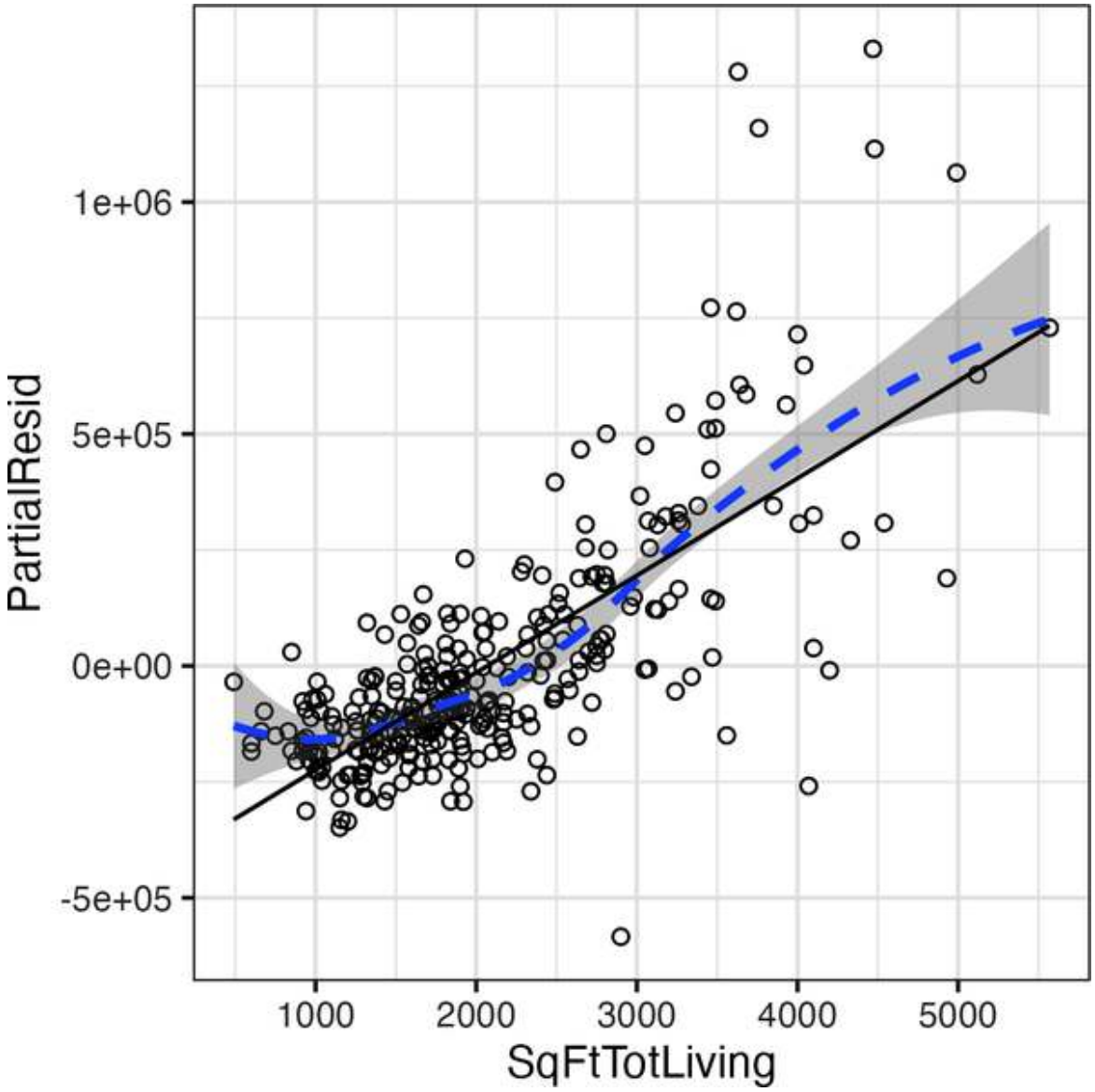


Figure 4-9. A partial residual plot for the variable `SqFtTotLiving`

This nonlinearity makes sense in this case: adding 500 feet in a small home makes a much bigger difference than adding 500 feet in a large home. This

suggests that, instead of a simple linear term for `SqFtTotLiving`, a nonlinear term should be considered (see "Polynomial and Spline Regression").

KEY IDEAS

- While outliers can cause problems for small data sets, the primary interest with outliers is to identify problems with the data, or locate anomalies.
- Single records (including regression outliers) can have a big influence on a regression equation with small data, but this effect washes out in big data.
- If the regression model is used for formal inference (p-values and the like), then certain assumptions about the distribution of the residuals should be checked. In general, however, the distribution of residuals is not critical in data science.
- The partial residuals plot can be used to qualitatively assess the fit for each regression term, possibly leading to alternative model specification.

Polynomial and Spline Regression

The relationship between the response and a predictor variable isn't necessarily linear. The response to the dose of a drug is often nonlinear: doubling the dosage generally doesn't lead to a doubled response. The demand for a product isn't a linear function of marketing dollars spent; at some point, demand is likely to be saturated. There are many ways that regression can be extended to capture these nonlinear effects.

KEY TERMS FOR NONLINEAR REGRESSION

Polynomial regression

Adds polynomial terms (squares, cubes, etc.) to a regression

Spline regression

Fitting a smooth curve with a series of polynomial segments

Knots

Values that separate spline segments

Generalized additive models

Spline models with automated selection of knots

Synonym

GAM

NONLINEAR REGRESSION

When statisticians talk about *nonlinear regression*, they are referring to models that can't be fit using least squares. What kind of models are nonlinear? Essentially all models where the response cannot be expressed as a linear combination of the predictors or some transform of the predictors. Nonlinear regression models are harder and computationally more intensive to fit, since they require numerical optimization. For this reason, it is generally preferred to use a linear model if possible.

Polynomial

Polynomial regression involves including polynomial terms in a regression equation. The use of polynomial regression dates back almost to the development of regression itself with a paper by Gergonne in 1815. For

example, a quadratic regression between the response Y and the predictor X would take the form:

$$Y = b_0 + b_1 X + b_2 X^2 + e$$

Polynomial regression can be fit in R through the `poly` function. For example, the following fits a quadratic polynomial for `SqFtTotLiving` with the King County housing data:

```
lm_poly <- lm(AdjSalePrice ~ poly(SqFtTotLiving, 2) + SqFtLot +
    BldgGrade + Bathrooms + Bedrooms, data = house_98105)
lm_poly
---
Call:
lm(formula = AdjSalePrice ~ poly(SqFtTotLiving, 2) + SqFtLot +
   BldgGrade + Bathrooms + Bedrooms, data = house_98105)

Coefficients:
           (Intercept)  poly(SqFtTotLiving, 2)1  poly(SqFtTotLiving, 2)2
             -402530.47               3271519.49                776934.02
               SqFtLot                BldgGrade                Bathrooms
                 32.56                135717.06                 -1435.12
              Bedrooms
              -9191.94
```

In `statsmodels`, we add the squared term to the model definition using `I(SqFtTotLiving**2)`:

```
model_poly = smf.ols(formula="AdjSalePrice ~  SqFtTotLiving + "
                "I(SqFtTotLiving**2) + "
                "SqFtLot + Bathrooms + Bedrooms + BldgGrade",
data=house_98105)
result_poly = model_poly.fit()
result_poly.summary()  ❶
```

❶ The intercept and the polynomial coefficients are different compared to R, which uses orthogonal polynomials. The remaining coefficients and the predictions are equivalent.

There are now two coefficients associated with `SqFtTotLiving`: one for the linear term and one for the quadratic term.

The partial residual plot (see "Partial Residual Plots and Nonlinearity") indicates some curvature in the regression equation associated with `SqFtTotLiving`. The fitted line more closely matches the smooth (see "Splines") of the partial residuals as compared to a linear fit (see Figure 4-10).

The `statsmodels` implementation works only for linear terms. The accompanying source code gives an implementation that will work for polynomial regression as well.

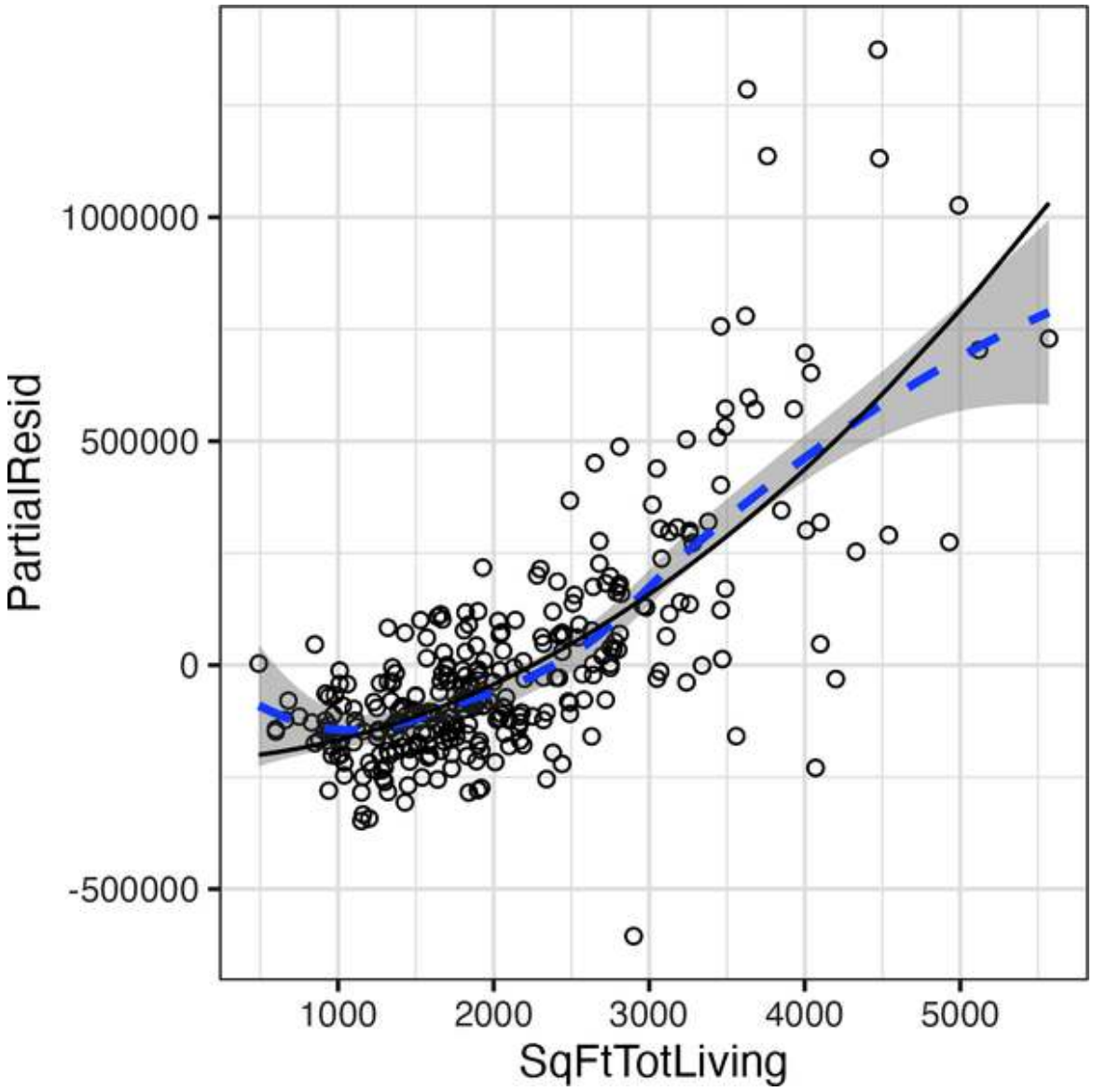


Figure 4-10. A polynomial regression fit for the variable `SqFtTotLiving` (solid line) versus a smooth (dashed line; see the following section about splines)

Splines

Polynomial regression captures only a certain amount of curvature in a nonlinear relationship. Adding in higher-order terms, such as a cubic or quartic polynomial, often leads to undesirable "wiggliness" in the regression equation. An alternative, and often superior, approach to modeling nonlinear relationships is to use splines. *Splines* provide a way to smoothly interpolate between fixed points. Splines were originally used by draftsmen to draw a smooth curve, particularly in ship and aircraft building.

The splines were created by bending a thin piece of wood using weights, referred to as “ducks”; see Figure 4-11.



Figure 4-11. Splines were originally created using bendable wood and “ducks” and were used as a draftsman’s tool to fit curves (photo courtesy of Bob Perry)

The technical definition of a spline is a series of piecewise continuous polynomials. They were first developed during World War II at the US Aberdeen Proving Grounds by I. J. Schoenberg, a Romanian mathematician. The polynomial pieces are smoothly connected at a series of fixed points in a predictor variable, referred to as *knots*. Formulation of splines is much more complicated than polynomial regression; statistical software usually handles the details of fitting a spline. The R package `splines` includes the function `bs` to create a *b-spline* (basis spline) term in a regression model. For example, the following adds a b-spline term to the house regression model:

```
library(splines)
knots <- quantile(house_98105$SqFtTotLiving, p = c(0.25, 0.5, 0.75))
lm_spline <- lm(AdjSalePrice ~ bs(SqFtTotLiving, knots = knots, degree = 3) +
    SqFtLot + Bathrooms + Bedrooms + BldgGrade,
  data = house_98105)
lm_spline
```

Two parameters need to be specified: the degree of the polynomial and the location of the knots. In this case, the predictor `SqFtTotLiving` is included in the model using a cubic spline (`degree=3`). By default, `bs` places knots at

the boundaries; in addition, knots were also placed at the lower quartile, median, and the upper quartile.

The `statsmodels` formula interface supports the use of splines in a similar way to R. Here, we specify the *b-spline* using `df`, the degrees of freedom. This will create `df` – `degree` = 6 – 3 = 3 internal knots with positions calculated in the same way as in the R code above:

```
formula = ("AdjSalePrice ~ bs(SqFtTotLiving, df=6, degree=3) + "
           "SqFtLot + Bathrooms + Bedrooms + BldgGrade")
model_spline = smf.ols(formula=formula, data=house_98105)
result_spline = model_spline.fit()
```

In contrast to a linear term, for which the coefficient has a direct meaning, the coefficients for a spline term are not interpretable. Instead, it is more useful to use the visual display to reveal the nature of the spline fit. Figure 4-12 displays the partial residual plot from the regression. In contrast to the polynomial model, the spline model more closely matches the smooth, demonstrating the greater flexibility of splines. In this case, the line more closely fits the data. Does this mean the spline regression is a better model? Not necessarily: it doesn't make economic sense that very small homes (less than 1,000 square feet) would have higher value than slightly larger homes. This is possibly an artifact of a confounding variable; see "Confounding Variables".

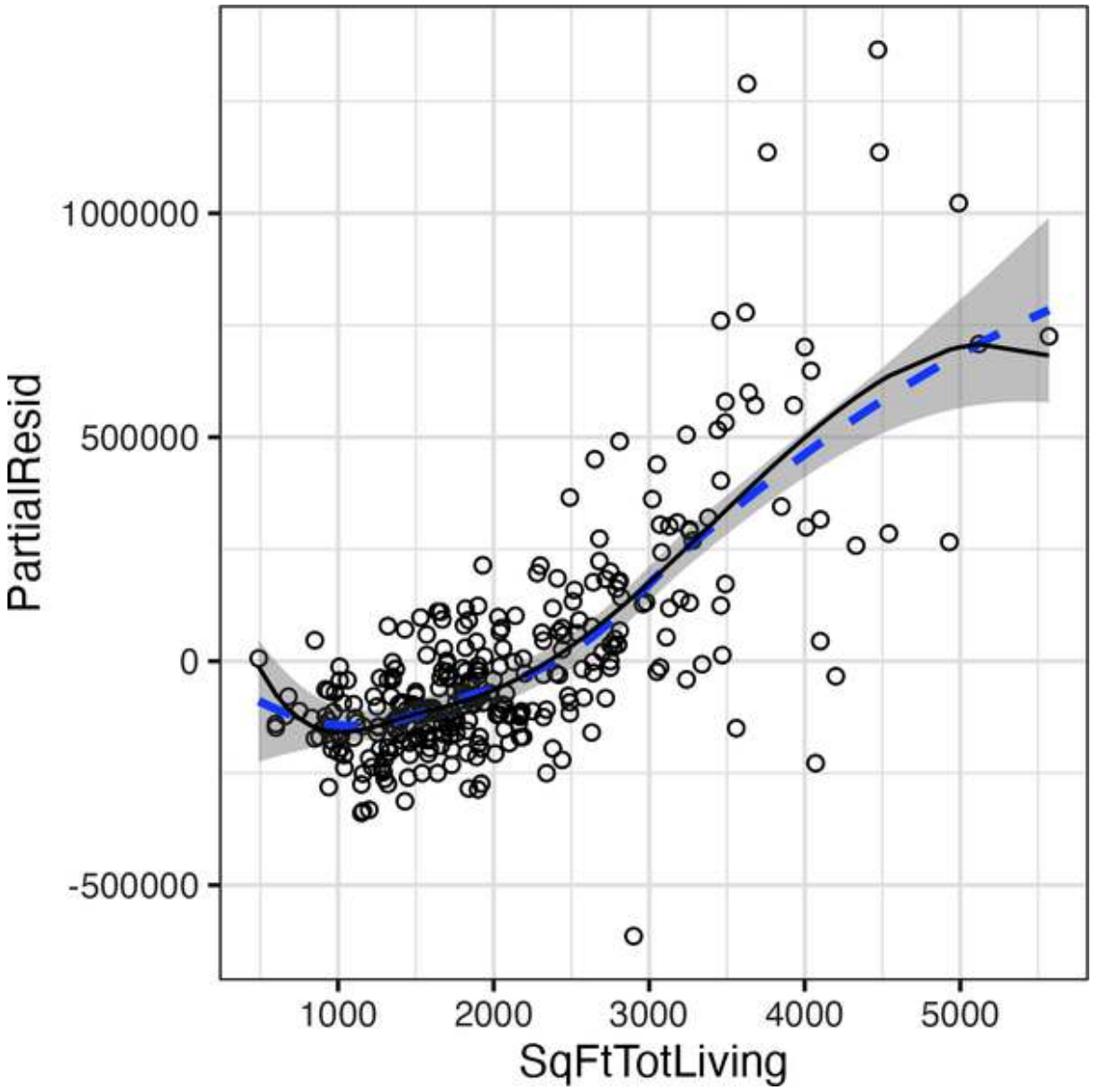


Figure 4-12. A spline regression fit for the variable `SqFtTotLiving` (solid line) compared to a smooth (dashed line)

Generalized Additive Models

Suppose you suspect a nonlinear relationship between the response and a predictor variable, either by a priori knowledge or by examining the regression diagnostics. Polynomial terms may not be flexible enough to capture the relationship, and spline terms require specifying the knots. *Generalized additive models*, or *GAMs*, are a flexible modeling technique

that can be used to automatically fit a spline regression. The `mgcv` package in R fits a GAM model to the housing data as follows:

```
library(mgcv)
lm_gam <- gam(AdjSalePrice ~ s(SqFtTotLiving) + SqFtLot +
    Bathrooms + Bedrooms + BldgGrade, data = house_98105)
lm_gam
```

The term `s(SqFtTotLiving)` tells the `gam` function to find the "best" knots for a spline term (see Figure 4-13).

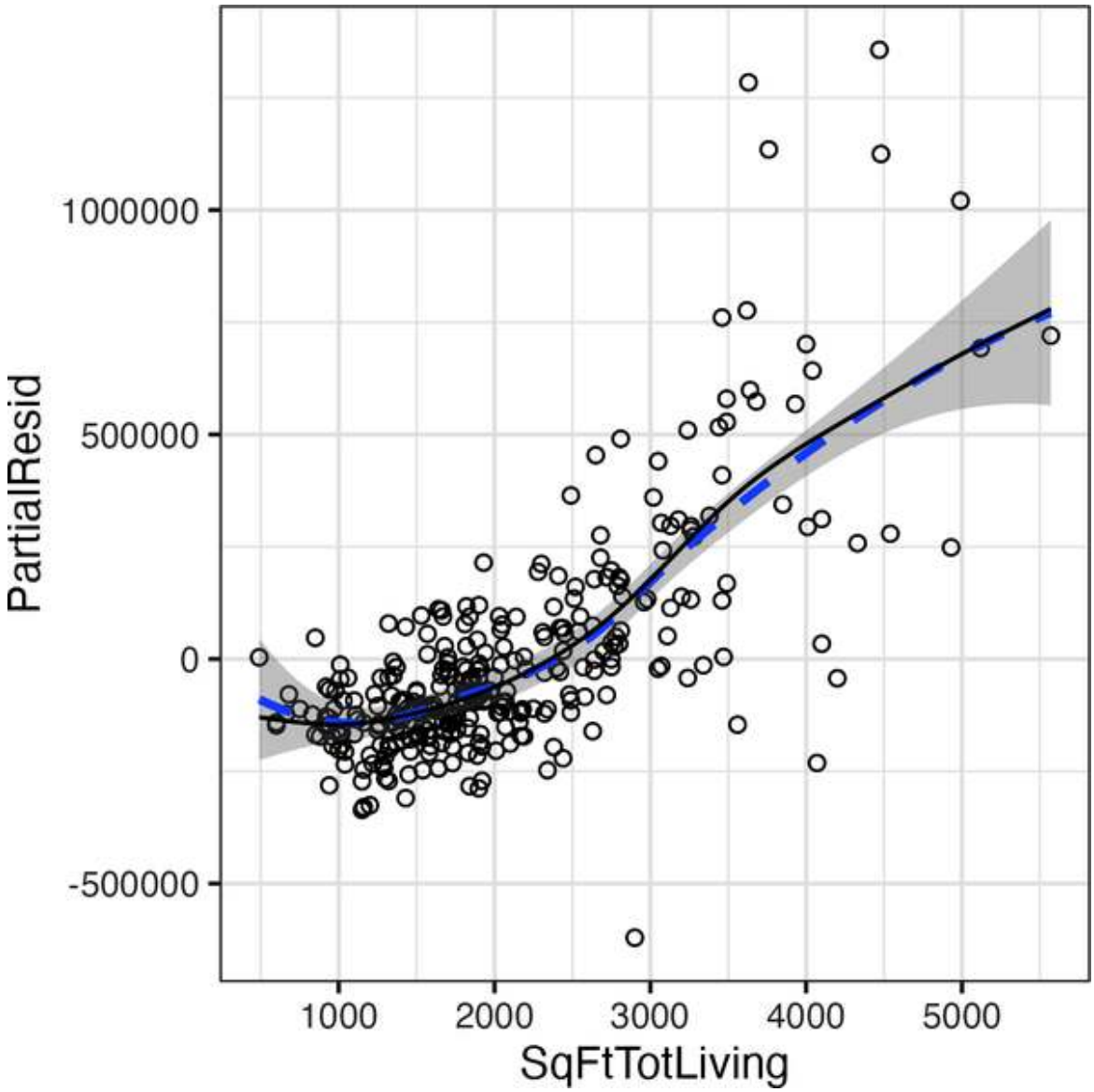


Figure 4-13. A GAM regression fit for the variable `SqFtTotLiving` (solid line) compared to a smooth (dashed line)

In Python, we can use the `pyGAM` package. It provides methods for regression and classification. Here, we use `LinearGAM` to create a regression model:

```
from pygam import LinearGAM, s, l

predictors = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
"BldgGrade"]
outcome = "AdjSalePrice"
X = house_98105[predictors].to_numpy()  ❶
y = house_98105[outcome]

gam = LinearGAM(s(0, n_splines=12) + l(1) + l(2) + l(3) + l(4))  ❷
gam.gridsearch(X, y)
print(gam.summary())
```

❶ The `pyGAM` package requires the predictors as a `numpy` array.

❷ The default value for `n_splines` is 20. This leads to overfitting for larger `SqFtTotLiving` values. A value of 12 leads to a more reasonable fit.

KEY IDEAS

- Outliers in a regression are records with a large residual.
- Multicollinearity can cause numerical instability in fitting the regression equation.
- A confounding variable is an important predictor that is omitted from a model and can lead to a regression equation with spurious relationships.
- An interaction term between two variables is needed if the effect of one variable depends on the level or magnitude of the other.
- Polynomial regression can fit nonlinear relationships between predictors and the outcome variable.
- Splines are series of polynomial segments strung together, joining at knots.
- We can automate the process of specifying the knots in splines using GAMs.

Further Reading

- For more on spline models and GAMs, see *The Elements of Statistical Learning*, 2nd ed., by Trevor Hastie, Robert Tibshirani, and Jerome Friedman (2009), and its shorter cousin, *An Introduction to Statistical Learning* by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (R edition 2021, Python edition 2023); both are Springer books.
- To learn more about using regression models for time-series forecasting, see *Practical Time Series Forecasting with R* by Galit Shmueli and Kenneth Lichtendahl (Axelrod Schnall, 2018).

Recalling the model fit and results in section "Assessing the Model", let's ask AI to help us explore the results further.

Analyzing the Results of a Regression Analysis with AI

Prompt:	Make a concise, one-paragraph analysis of the following regression results: <paste result of the regression analysis here>
ChatGPT:	The regression explains a substantial share of price variation (adjusted $R^2\approx 0.54$) and is highly significant overall (F-test $p<2\times 10^{-16}$), indicating strong joint explanatory power. Living area and building grade are the dominant positive predictors: each additional square foot of living space adds about $229, and a one-unit increase in building grade adds roughly $106k, both with very large t-statistics. In contrast, lot size has no statistically significant effect once other variables are controlled for. Bathrooms and bedrooms enter with negative and significant coefficients, suggesting multicollinearity or that, conditional on total living area and quality, additional bathrooms or bedrooms reflect less valuable configurations. The large residual standard error (~$261k) and wide residual range indicate considerable unexplained heterogeneity, consistent with omitted factors such as location, age, or neighborhood effects.

Now we make a slight tweak to the prompt (optional second paragraph) and see what changes:

Prompt:	Make a concise, one- or two-paragraph analysis of the following regression results: <paste result of regression analysis here>
ChatGPT:	This model explains a substantial portion of variation in adjusted sale prices (adjusted $R^2\approx 0.54$) and is highly significant overall, indicating strong joint explanatory power of the included covariates. Total living area and building grade emerge as the primary drivers of price: each additional square foot of living space is associated with about a $229 increase in price, while a one-unit increase in building grade adds roughly $106,000, both with extremely large t-statistics. In contrast, lot size has no statistically significant effect once other factors are controlled for, suggesting that land area contributes little incremental information beyond the structure itself in this specification. Bathrooms and bedrooms have negative and statistically significant coefficients, implying that conditional on total living area and building quality, additional rooms are associated with lower prices. This pattern is consistent with multicollinearity or with a quality-versus-layout interpretation, where more bathrooms or bedrooms for a fixed amount of space may signal smaller rooms or less desirable configurations. The relatively large residual standard error (~$261k) and wide residual range indicate considerable unexplained heterogeneity, likely reflecting omitted factors such as location, age, or neighborhood amenities.

The two-paragraph response is a bit more verbose, but the content is largely the same. Let's now ask for suggestions:

Prompt:	Suggest improvements to the model
ChatGPT:	Several improvements could strengthen the model's explanatory power, interpretability, and robustness: 1. Add key omitted variables: Include location and neighborhood controls (e.g., ZIP code or tract fixed effects, distance to amenities, school quality), property age, renovation status, and lot characteristics (waterfront, view, zoning). These factors typically explain a large share of housing price variation and would likely reduce the large residual error. 2. Address multicollinearity and functional form: Bedrooms, bathrooms, and total living area are highly correlated. Consider replacing bedrooms and bathrooms with ratios (e.g., bathrooms per bedroom), dropping one of these variables, or using principal components. Alternatively, model prices per square foot as the outcome. Nonlinearities (e.g., log–log or semi-log models, splines for square footage) may better capture diminishing returns to size. 3. Transform the dependent variable: Use log(AdjSalePrice) to stabilize variance, reduce the influence of extreme prices, and allow coefficients to be interpreted as approximate percentage effects. This often improves fit and residual behavior in housing price models. 4. Improve treatment of lot size and interactions: Lot size may matter nonlinearly or only in certain contexts. Consider log or spline transformations, or interactions with location (urban versus suburban) or building grade to capture heterogeneous effects.

5. Diagnostic checks and robustness: Examine residual plots for heteroskedasticity and leverage; use robust (HC) standard errors if needed. Investigate outliers and influential observations, and validate performance with a holdout sample or cross-validation to assess predictive stability.

Summary

Perhaps no other statistical method has seen greater use over the years than regression—the process of establishing a relationship between multiple predictor variables and an outcome variable. The fundamental form is linear: each predictor variable has a coefficient that describes a linear relationship between the predictor and the outcome. More advanced forms of regression, such as polynomial and spline regression, permit the relationship to be nonlinear. In classical statistics, the emphasis is on finding a good fit to the observed data to explain or describe some phenomenon, and the strength of this fit is how traditional *in-sample* metrics are used to assess the model. In data science, by contrast, the goal is typically to predict values for new data, so metrics based on predictive accuracy for out-of-sample data are used. Variable selection methods are used to reduce dimensionality and create more compact models.

Exploration with AI

1. Review the regression results and the analysis presented in this chapter, and compare them to the analysis generated by AI and shown in "Analyzing the Results of a Regression Analysis with AI". Are they similar? Gather some of the analytical points made in the chapter and put them into a prompt that asks your AI to compare them to the AI-generated analysis, then comment.
2. Ask your AI to fit a linear regression to the cotton dust exposure data (found in *LungDisease.csv*), and compare the results to those

presented in the chapter. There are a high number of cases at 20 years of exposure (and right before and after). This is likely due to numerous workers retiring after 20 years. Query your AI to see if it figures this out.

3. Ask your AI to fit a linear regression model to the King County housing data, as was done in this chapter. What issues arose and were brought to your attention? Were they the same issues that were raised in the chapter? For example, were different strategies suggested to deal with the zip code issue?

4. Ask your AI to predict the sale price for a house with 100 square feet of living area, a 50 square foot lot, with one bathroom, one bedroom, and a building grade of 5. Does the AI caution you that the prediction lies well outside the range of the observed data and is nonsensical?

5. Does your AI explicitly raise the issue of nonlinearity in the data? Ask it to fit a polynomial regression and a spline regression, and assess the results. Compare what the AI produces to what you see in the chapter.

---

[1] This and subsequent sections in this chapter © 2020–2025 Datastats, LLC, Peter Bruce, Andrew Bruce, and Peter Gedeck; used by permission.

[2] In Bayesian statistics, the true value is assumed to be a random variable with a specified distribution. In the Bayesian context, instead of estimates of unknown parameters, there are posterior and prior distributions.

[3] The `-1` argument in the `model.matrix` produces one-hot encoding representation (by removing the intercept, hence the "-"). Otherwise, the default in R is to produce a matrix with $P - 1$ columns with the first factor level as a reference.

[4] This is unintuitive, but can be explained by the impact of location as a confounding variable; see "Confounding Variables".

[5] There are 80 zip codes in King County, several with just a handful of sales. An alternative to directly using zip code as a factor variable, `ZipGroup` clusters similar zip codes into a single group. See "Factor Variables with Many Levels" for details.

6 The term *hat-value* comes from the notion of the hat matrix in regression. Multiple linear regression can be expressed by the formula $\hat{Y} = HY$ where H is the hat matrix. The hat-values correspond to the diagonal of H.

7 The coefficient for `Bathrooms` becomes negative, which is unintuitive. Location has not been taken into account, and the zip code 98105 contains areas of disparate types of homes. See "Confounding Variables" for a discussion of confounding variables.

Chapter 5. Classification

Data scientists are often tasked with automating decisions for business problems. Is an email an attempt at phishing? Is a customer likely to churn? Is the web user likely to click on an advertisement? These are all *classification* problems, a form of *supervised learning* in which we first train a model on data where the outcome is known and then apply the model to data where the outcome is not known. Classification is perhaps the most important form of prediction: the goal is to predict whether a record is a 1 or a 0 (phishing/not-phishing, click/don't click, churn/don't churn), or in some cases, one of several categories (for example, Gmail's filtering of your inbox into "primary," "social," "promotional," or "forums").

Often, we need more than a simple binary classification: we want to know the predicted probability that a case belongs to a class. Rather than having a model simply assign a binary classification, most algorithms can return a probability score (propensity) of belonging to the class of interest. In fact, with logistic regression, the default output from R is on the log-odds scale, and this must be transformed to a propensity. In Python's `scikit-learn`, logistic regression, like most classification methods, provides two prediction methods: `predict` (which returns the class) and `predict_proba` (which returns probabilities for each class). A sliding cutoff can then be used to convert the propensity score to a decision. The general approach is:

1. Establish a cutoff probability for the class of interest, above which we consider a record as belonging to that class.
2. Estimate (with any model) the probability that a record belongs to the class of interest.
3. If that probability is above the cutoff probability, assign the new record to the class of interest.

The higher the cutoff, the fewer the records predicted as 1—that is, as belonging to the class of interest. The lower the cutoff, the more the records predicted as 1.

This chapter covers several key techniques for classification and estimating propensities; additional methods that can be used both for classification and for numerical prediction are described in the next chapter.

MORE THAN TWO CATEGORIES?

The vast majority of problems involve a binary response. Some classification problems, however, involve a response with more than two possible outcomes. For example, at the anniversary of a customer's subscription contract, there might be three outcomes: the customer leaves or "churns" ($Y = 2$), goes on a month-to-month contract ($Y = 1$), or signs a new long-term contract ($Y = 0$). The goal is to predict $Y = j$ for $j = 0$, 1, or 2. Most of the classification methods in this chapter can be applied, either directly or with modest adaptations, to responses that have more than two outcomes. Even in the case of more than two outcomes, the problem can often be recast into a series of binary problems using conditional probabilities. For example, to predict the outcome of the contract, you can solve two binary prediction problems:

- Predict whether $Y = 0$ or $Y > 0$.
- Given that $Y > 0$, predict whether $Y = 1$ or $Y = 2$.

In this case, it makes sense to break up the problem into two cases: (1) whether the customer churns; and (2) if they don't churn, what type of contract they will choose. From a model-fitting viewpoint, it is often advantageous to convert the multiclass problem to a series of binary problems. This is particularly true when one category is much more common than the other categories.

Naive Bayes

The naive Bayes algorithm uses the probability of observing predictor values, given an outcome, to estimate what is really of interest: the probability of observing outcome $Y = i$, given a set of predictor values.[1]

KEY TERMS FOR NAIVE BAYES

Conditional probability

The probability of observing some event (say, $X = i$) given some other event (say, $Y = i$), written as $P(X_i|Y_i)$.

Posterior probability

The probability of an outcome after the predictor information has been incorporated (in contrast to the *prior probability* of outcomes, not taking predictor information into account).

To understand naive Bayesian classification, we can start out by imagining complete or exact Bayesian classification. For each record to be classified:

1. Find all the other records with the same predictor profile (i.e., where the predictor values are the same).
2. Determine what classes those records belong to and which class is most prevalent (i.e., probable).
3. Assign that class to the new record.

The preceding approach amounts to finding all the records in the sample that are exactly like the new record to be classified in the sense that all the predictor values are identical.

NAIVE BAYES NEEDS CATEGORICAL PREDICTORS

Predictor variables must be categorical (factor) variables in the standard naive Bayes algorithm. See "Numeric Predictor Variables" for two workarounds for using continuous variables.

Why Exact Bayesian Classification Is Impractical

When the number of predictor variables exceeds a handful, many of the records to be classified will be without exact matches. Consider a model to predict voting on the basis of demographic variables. Even a sizable sample may not contain even a single match for a new record who is a male Hispanic with high income from the US Midwest who voted in the last election, did not vote in the prior election, has three daughters and one son, and is divorced. And this is with just eight variables, a small number for most classification problems. The addition of just a single new variable with five equally frequent categories reduces the probability of a match by a factor of 5.

The Naive Solution

In the naive Bayes solution, we no longer restrict the probability calculation to those records that match the record to be classified. Instead, we use the entire data set. The naive Bayes modification is as follows:

1. For a binary response $Y = i$ ($i = 0$ or 1), estimate the individual conditional probabilities for each predictor $P(X_j|Y = i)$; these are the probabilities that the predictor value is in the record when we observe $Y = i$. This probability is estimated by the proportion of X_j values among the $Y = i$ records in the training set.
2. Multiply these probabilities by each other, and then by the proportion of records belonging to $Y = i$.
3. Repeat steps 1 and 2 for all the classes.

4. Estimate a probability for outcome *i* by taking the value calculated in step 2 for class *i* and dividing it by the sum of such values for all classes.

5. Assign the record to the class with the highest probability for this set of predictor values.

This naive Bayes algorithm can also be stated as an equation for the probability of observing outcome $Y = i$, given a set of predictor values $X_1, \ldots, X_p$:

$$P(Y = i|X_1, X_{2,\ldots,}X_p)$$

Here is the full formula for calculating class probabilities using exact Bayes classification:

$$P(Y = i|X_1, X_{2,\ldots,}X_p) = \frac{P(Y = i)P(X_{1,\ldots,}X_p|Y = i)}{P(Y = 0)P(X_{1,\ldots,}X_p|Y = 0) + P(Y = 1)P(X_{1,\ldots,}X_p|Y = 1)}$$

Under the naive Bayes assumption of conditional independence, this equation changes into:

$$P(Y = i|X_1, X_{2,\ldots,}X_p) = \frac{P(Y = i)P(X_1|Y = i)\ldots P(X_p|Y = i)}{P(Y = 0)P(X_1|Y = 0)\ldots P(X_p|Y = 0) + P(Y = 1)P(X_1|Y = 1)\ldots P(X_p|Y = 1)}$$

Why is this formula called "naive"? We have made a simplifying assumption that the *exact conditional probability* of a vector of predictor values, given observing an outcome, is sufficiently well estimated by the product of the individual conditional probabilities $P(X_j|Y = i)$. In other words, in estimating $P(X_j|Y = i)$ instead of $P(X_1, X_2, \cdots, X_p|Y = i)$, we are assuming X_j is *independent* of all the other predictor variables X_k for $k \neq j$.

Several packages in R can be used to estimate a naive Bayes model. The following fits a model to the loan payment data using the `klaR` package:

```
library(tidyverse)
library(tidymodels)
library(discrim)
loan_data <- read_csv("loan_data.csv.gz", show_col_types = FALSE) %>%
  dplyr::select(-c(index, status)) %>%
  mutate(
    across(where(is.character), as.factor),
    outcome = factor(outcome, levels = c("paid off", "default")),
  )
naive_model <- naive_Bayes() %>%  ❶
  fit(outcome ~ purpose_ + home_ + emp_len_, loan_data)

naive_model %>%
  extract_fit_engine() %>%  ❷
  pluck("tables")
---
$purpose_
          var
grouping   credit_card debt_consolidation home_improvement major_purchase
  paid off  0.18759649         0.55215915       0.07150104     0.05359270
  default   0.15151515         0.57571347       0.05981209     0.03727229
          var
grouping      medical      other small_business
  paid off 0.01424728 0.09990737     0.02099599
  default  0.01433549 0.11561025     0.04574126

$home_
          var
grouping    MORTGAGE       OWN      RENT
  paid off 0.4894800 0.0808963 0.4296237
  default  0.4313440 0.0832782 0.4853778

$emp_len_
          var
grouping     < 1 Year   > 1 Year
```

```
paid off 0.03105289 0.96894711
default  0.04728508 0.95271492
```

❶ The `parsnip` function `naive_Bayes` provides the unified interface to naive Bayes models. By default, it uses the implementation in `klaR`.

❷ The `extract_fit_engine` function gives access to the underlying `klaR` model. With `pluck`, we can extract the conditional probability tables.

The output from the model is the conditional probabilities $P(X_j|Y = i)$.

In Python we can use `sklearn.naive_bayes.MultinomialNB` from `scikit-learn`. We need to convert the categorical features to dummy variables before we fit the model:

```
import pandas as pd
from sklearn.naive_bayes import MultinomialNB

loan_data = pd.read_csv("loan_data.csv.gz")

predictors = ["purpose_", "home_", "emp_len_"]
outcome = "outcome"
X = pd.get_dummies(loan_data[predictors], prefix="", prefix_sep="", dtype=int)
y = loan_data[outcome]

naive_model = MultinomialNB(alpha=0.01, fit_prior=True)
naive_model.fit(X, y)
```

It is possible to derive the conditional probabilities from the fitted model using the property `feature_log_prob_`.

The model can be used to predict the outcome of a new loan. We use the last value of the data set for testing:

```
new_loan <- loan_data[147, c("purpose_", "home_", "emp_len_")]
row.names(new_loan) <- NULL
new_loan
---
        purpose_    home_  emp_len_
1 small_business MORTGAGE  > 1 Year
```

In Python, we get this value as follows:

```
new_loan = X.loc[146:146, :]
```

In this case, the model predicts a “default”:

```
predict(naive_model, new_loan)  ❶
---
.pred_class
default
---
predict(naive_model, new_loan, type = "prob")  ❷
---
.pred_paid off   .pred_default
0.3463013              0.6536987
```

❶ The `predict` function returns a tibble with the predicted class for each row in `new_loan`. The use of the prefix “.” is `tidymodels`’s way to identify predicted values.

❷ With `type = "prob"`, `predict` returns the predicted probabilities for each class.

As we discussed, `scikit-learn`’s classification models have two methods—`predict`, which returns the predicted class, and `predict_proba`, which returns the class probabilities:

```
print("predicted class: ", naive_model.predict(new_loan)[0])

probabilities = pd.DataFrame(naive_model.predict_proba(new_loan),
                             columns=naive_model.classes_)
print("predicted probabilities", probabilities)
---
predicted class:  default
predicted probabilities
    default  paid off
0  0.653696  0.346304
```

The prediction also returns a `posterior` estimate of the probability of default. The naive Bayesian classifier is known to produce *biased* estimates. However, where the goal is to *rank* records according to the probability that $Y = 1$, unbiased estimates of probability are not needed, and naive Bayes produces good results.

Numeric Predictor Variables

The Bayesian classifier works only with categorical predictors (e.g., with spam classification, where the presence or absence of words, phrases, characters, and so on lies at the heart of the predictive task). To apply naive Bayes to numerical predictors, one of two approaches must be taken:

- Use a probability model—for example, the normal distribution (see "Normal Distribution")—to estimate the conditional probability $P(X_j|Y = i)$.
- Bin and convert the numerical predictors to categorical predictors and apply the algorithm of the previous section.

The naive Bayes implementation in R uses the second approach for numerical predictors. For example, include two numerical predictors in the model to predict loan default:

```
less_naive <- naive_Bayes() %>%
  set_args(usekernel = FALSE) %>%  ❶
  fit(outcome ~ borrower_score + payment_inc_ratio +
      purpose_ + home_ + emp_len_, loan_data)

(naive_model %>%
    extract_fit_engine() %>%
    pluck("tables"))[1:2]
---
$borrower_score
             [,1]      [,2]
paid off 0.5347933 0.1238649
default  0.4632195 0.1233597

$payment_inc_ratio
             [,1]     [,2]
paid off 7.294367 4.018183
default  8.770084 4.373793
```

❶ By default, the `klaR` engine uses a kernel density estimate to describe the distribution of the numerical features. With `usekernel = FALSE` we tell the `klaR` package to use a normal distribution instead.

The output for the variables `borrower_score` and `payment_inc_ratio` is a matrix of means (first column) and standard deviations (second column) for the conditional distribution. Figure 5-1 shows the distributions of `borrower_score` and the conditional probabilities $P(Y = default|borrower_score)$ and $P(Y = paidoff|borrower_score)$ estimated from the naive Bayes model. Also shown for comparison are the distributions $P(borrower_score|Y = default)$ and $P(borrower_score|Y = paid_off)$:

```
library(patchwork)
library(ggpubr)
p_paid <- sum(loan_data$outcome == "default") / nrow(loan_data)
p_default <- sum(loan_data$outcome == "paid off") / nrow(loan_data)
```

```
p_bs_paid <- function(x, stats) dnorm(x, mean = stats[1, 1], sd = stats[1, 2])
p_bs_default <- function(x, stats) dnorm(x, mean = stats[2, 1], sd = stats[2, 2])
p_paid_bs <- function(x, stats) {
  return(p_paid * p_bs_paid(x, stats) /
      (p_paid * p_bs_paid(x, stats) + p_default * p_bs_default(x, stats)))
}
p_default_bs <- function(x, stats) {
  return(p_default * p_bs_default(x, stats) /
      (p_paid * p_bs_paid(x, stats) + p_default * p_bs_default(x, stats)))
}

stats <- less_naive %>%
  extract_fit_engine() %>%
  pluck("tables", "borrower_score")
g1 <- ggplot(data.frame(borrower_score = c(0, 1)), aes(x = borrower_score)) +
  stat_function(fun = p_bs_paid, color = "blue", linetype = 1,
    args = list(stats = stats)) +
  stat_function(fun = p_bs_default, color = "red", linetype = 2,
    args = list(stats = stats)) +
  labs(y = "P(Xj | Y=i)") + theme_bw()
g2 <- ggplot(data.frame(borrower_score = c(0, 1)), aes(x = borrower_score)) +
  stat_function(fun = p_paid_bs, color = "blue", linetype = 1,
    args = list(stats = stats)) +
  stat_function(fun = p_default_bs, color = "red", linetype = 2,
    args = list(stats = stats)) +
  labs(y = "P(Y=i | Xj)") + theme_bw()
g <- wrap_plots(g1, g2)
g
```

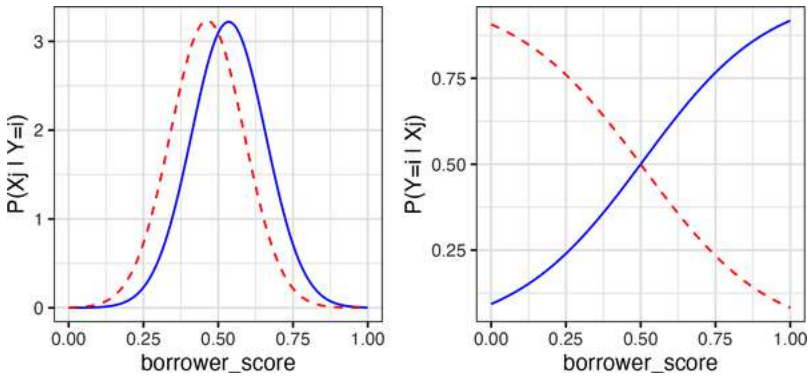


Figure 5-1. The conditional density of the variable `borrower_score` in the naive Bayes model. The graph on the left shows the probability density functions of P(borrower_score|Y = default) and P(borrower_score|Y = paidoff); the graph on the right shows the posterior probabilities for P(Y = default|borrower_score) and P(Y = paidoff|borrower_score).

The solid and dashed lines correspond to the conditional probability given the loan is paid off or defaults, respectively.

The `scikit-learn` package has an implementation of naive Bayes for numerical predictors, `GaussianNB`. It is, however, not able to work with mixed categorical and numerical predictors. We can use the other approach of binning the numerical predictors instead:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, KBinsDiscretizer
from sklearn.pipeline import Pipeline

num_cols = ["borrower_score", "payment_inc_ratio"]
cat_cols = ["purpose_", "home_", "emp_len_"]
mixed_naive_model = Pipeline([
    ("pre", ColumnTransformer([
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
        ("num", KBinsDiscretizer(n_bins=5, strategy="uniform"), num_cols),
    ])),
    ("clf", MultinomialNB()),
])
```

```
X = loan_data[cat_cols + num_cols]
mixed_naive_model.fit(X, loan_data["outcome"])
```

LAPLACE SMOOTHING FOR ABSENT PREDICTORS

When a predictor category is absent in the training data, the algorithm assigns *zero probability* to the outcome variable in new data, rather than simply ignoring this variable and using the information from other variables, as other methods might. Most implementations of naive Bayes use a smoothing parameter (Laplace smoothing) to prevent this.

KEY IDEAS

- Naive Bayes works with categorical (factor) predictors and outcomes.
- It asks, "Within each outcome category, which predictor categories are most probable?"
- That information is then inverted to estimate probabilities of outcome categories, given predictor values.

Further Reading

- *The Elements of Statistical Learning*, 2nd ed., by Trevor Hastie, Robert Tibshirani, and Jerome Friedman (Springer, 2009).
- There is a full chapter on naive Bayes in *Machine Learning for Business Analytics* by Galit Shmueli, Peter Bruce, Nitin Patel, Peter Gedeck, Inbal Yahav, and Kenneth Lichtendahl (Wiley, 2007–2025, with editions for R, Python, Excel, RapidMiner, and JMP).

Discriminant Analysis

Discriminant analysis is the earliest statistical classifier; it was introduced by R. A. Fisher in 1936 in an article published in the *Annals of Eugenics* journal.[2]

KEY TERMS FOR DISCRIMINANT ANALYSIS

Covariance

A measure of the extent to which one variable varies in concert with another (i.e., similar magnitude and direction)

Discriminant function

The function that, when applied to the predictor variables, maximizes the separation of the classes

Discriminant weights

The scores that result from the application of the discriminant function and are used to estimate probabilities of belonging to one class or another

While discriminant analysis encompasses several techniques, the most commonly used is *linear discriminant analysis*, or *LDA*. The original method proposed by Fisher was actually slightly different from LDA, but the mechanics are essentially the same. LDA is now less widely used with the advent of more sophisticated techniques, such as tree models and logistic regression.

However, you may still encounter LDA in some applications, and it has links to other more widely used methods (such as principal components analysis; see "Principal Components Analysis").

> **LDA**
>
> Linear discriminant analysis should not be confused with latent Dirichlet allocation, also referred to as LDA. Latent Dirichlet allocation is used in text and natural language processing and is unrelated to linear discriminant analysis.

Covariance Matrix

To understand discriminant analysis, it is first necessary to introduce the concept of *covariance* between two or more variables. The covariance measures the relationship between two variables x and z. Denote the mean for each variable by $\overline{x}$ and $\overline{z}$ (see "Mean"). The covariance $s_{x,z}$ between x and z is given by:

$$s_{x,z} = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(z_i - \overline{z})}{n - 1}$$

where n is the number of records (note that we divide by $n - 1$ instead of n; see "Degrees of Freedom, and n or $n - 1$?").

As with the correlation coefficient (see "Correlation"), positive values indicate a positive relationship and negative values indicate a negative relationship. Correlation, however, is constrained to be between –1 and 1, whereas covariance scale depends on the scale of the variables x and z. The *covariance matrix* Σ for x and z consists of the individual variable variances, s_x^2 and s_z^2, on the diagonal (where row and column are the same variable) and the covariances between variable pairs on the off-diagonals:

$$\hat{\Sigma} = \begin{bmatrix} s_x^2 & s_{x,z} \\ s_{z,x} & s_z^2 \end{bmatrix}$$

> **MAHALANOBIS DISTANCE**
>
> Recall that the standard deviation is used to normalize a variable to a z-score; the covariance matrix is used in a multivariate extension of this standardization process. This is known as Mahalanobis distance (see "Other Distance Metrics") and is related to the LDA function.

Fisher's Linear Discriminant

For simplicity, let's focus on a classification problem in which we want to predict a binary outcome y using just two continuous numeric variables (x, z). Technically, discriminant analysis assumes the predictor variables are normally distributed continuous variables, but, in practice, the method works well even for nonextreme departures from normality, and for binary predictors. Fisher's linear discriminant distinguishes variation *between* groups, on the one hand, from variation *within* groups on the other. Specifically, seeking to divide the records into two groups, linear discriminant analysis (LDA) focuses on maximizing the "between" sum of squares $\text{SS}_{\text{between}}$ (measuring the variation between the two groups) relative to the "within" sum of squares $\text{SS}_{\text{within}}$ (measuring the within-group variation). In this case, the two groups correspond to the records (x_0, z_0) for which $y = 0$ and the records (x_1, z_1) for which $y = 1$. The method finds the linear combination $w_x x + w_z z$ that maximizes that sum of squares ratio:

$$\frac{\text{SS}_{\text{between}}}{\text{SS}_{\text{within}}}$$

The between sum of squares is the squared distance between the two group means, and the within sum of squares is the spread around the means within each group, weighted by the covariance matrix. Intuitively, by maximizing the between sum of squares and minimizing the within sum of squares, this method yields the greatest separation between the two groups.

A Simple Example

In R, you can find several implementations of LDA. A popular one is the `MASS` package, associated with the book *Modern Applied Statistics with S* by W. N. Venables and B. D. Ripley (Springer, 1994), which is also the default model used in `tidymodels`. The following trains a model to a sample of loan data using two predictor variables, `borrower_score` and `payment_inc_ratio`, and prints out the estimated linear discriminator weights:

```
library(discrim) ❶
loan3000 <- read_csv("loan3000.csv", show_col_types = FALSE) %>%
  mutate(
    outcome = factor(outcome, levels = c("paid off", "default")),
    purpose_ = factor(purpose_)
  )

loan_lda <- discrim_linear() %>% ❷
  fit(outcome ~ borrower_score + payment_inc_ratio, data = loan3000)

loan_lda %>%
  extract_fit_engine() %>%
  pluck("scaling")
```

❶ The `discrim` package from `tidymodels` needs to be loaded as well. It implements LDA and other discriminant analysis approaches.

❷ `discrim_linear` is the `tidymodels` wrapper for LDA models. The default *engine* is `MASS`.

We can use `LinearDiscriminantAnalysis` from `sklearn.discriminant_analysis` in Python. The `scalings_` property gives the estimated weights:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

loan3000 = pd.read_csv("loan3000.csv")

loan3000.outcome = loan3000.outcome.astype("category")

predictors = ["borrower_score", "payment_inc_ratio"]
outcome = "outcome"

X = loan3000[predictors]
y = loan3000[outcome]

loan_lda = LinearDiscriminantAnalysis()
loan_lda.fit(X, y)
pd.DataFrame(loan_lda.scalings_, index=X.columns)
```

USING DISCRIMINANT ANALYSIS FOR FEATURE SELECTION

If the predictor variables are normalized prior to running LDA, the discriminator weights are measures of variable importance, thus providing a computationally efficient method of feature selection.

The `lda` function can predict the probability of "default" versus "paid off":

```
lda_pred <- augment(loan_lda, new_data = loan3000) ❶
head(lda_pred %>% select(.pred_class, .pred_default, `.pred_paid off`))
---
   paid off   default
1 0.4464563 0.5535437
2 0.4410466 0.5589534
3 0.7273038 0.2726962
4 0.4937462 0.5062538
5 0.3900475 0.6099525
6 0.5892594 0.4107406
```

❶ The `augment` function returns the original data frame with additional columns for the predicted class (`.pred_class`) and the predicted probabilities for each class (here, `.pred_default` and `.pred_paid off`).

The `predict_proba` method of the fitted model returns the probabilities for the "default" and "paid off" outcomes:

```
pred = pd.DataFrame(loan_lda.predict_proba(loan3000[predictors]),
                    columns=loan_lda.classes_)
pred.head()
```

A plot of the predictions helps illustrate how LDA works. Using the output from the `predict` function, a plot of the estimated probability of default is produced as follows:

```
loan_fit_engine <- loan_lda %>% extract_fit_engine()
center <- 0.5 * (loan_fit_engine$mean[1, ] + loan_fit_engine$mean[2, ])
slope <- -loan_fit_engine$scaling[1] / loan_fit_engine$scaling[2]
intercept <- center[2] - center[1] * slope

ggplot(data = lda_pred,
  aes(x = borrower_score, y = payment_inc_ratio, color = .pred_default)) +
  geom_point(alpha = 0.6) +
  scale_color_gradientn(colors = c("#ca0020", "#f7f7f7", "#0571b0")) +
  coord_cartesian(xlim = c(0, 1), ylim = c(0, 20)) +
  geom_abline(slope = slope, intercept = intercept, color = "darkgreen")
```

A similar graph is created in Python using this code:

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Use scalings and center of means to determine decision boundary
center = np.mean(loan_lda.means_, axis=0)
slope = - loan_lda.scalings_[0] / loan_lda.scalings_[1]
intercept = center[1] - center[0] * slope

# payment_inc_ratio for borrower_score of 0 and 20
x_0 = (0 - intercept) / slope
x_20 = (20 - intercept) / slope

lda_df = pd.concat([loan3000, pred["default"]], axis=1)
lda_df.head()

fig, ax = plt.subplots(figsize=(4, 4))
g = sns.scatterplot(x="borrower_score", y="payment_inc_ratio",
                    hue="default", data=lda_df,
                    palette=sns.diverging_palette(240, 10, n=9, as_cmap=True),
                    ax=ax, legend=False)

ax.set_ylim(0, 20)
ax.set_xlim(0.15, 0.8)
ax.plot((x_0, x_20), (0, 20), linewidth=3)
ax.plot(*loan_lda.means_.transpose())
```

The resulting plot is shown in Figure 5-2. Data points on the left of the diagonal line are predicted to default (probability greater than 0.5).

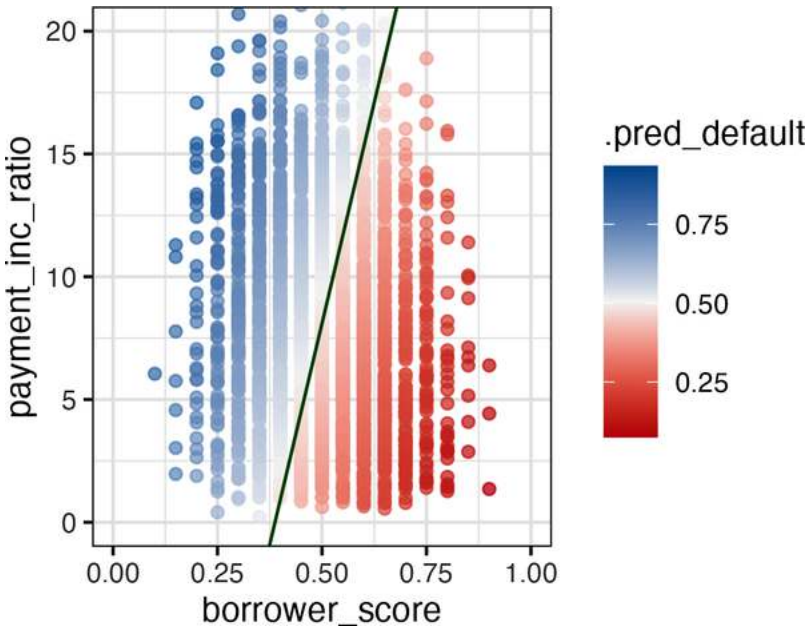


Figure 5-2. LDA prediction of loan default using two variables: a score of the borrower's creditworthiness and the payment-to-income ratio

Using the discriminant function weights, LDA splits the predictor space into two regions, as shown by the solid line. The predictions farther away from the line in both directions have a higher level of confidence (i.e., a probability further away from 0.5).

Extensions of Discriminant Analysis

More predictor variables: while the text and example in this section used just two predictor variables, LDA works just as well with more than two predictor variables. The only limiting factor is the number of records (estimating the covariance matrix requires a sufficient number of records per variable, which is typically not an issue in data science applications).

There are other variants of discriminant analysis. The best known is quadratic discriminant analysis (QDA). Despite its name, QDA is still a linear discriminant function. The main difference is that in LDA, the covariance matrix is assumed to be the same for the two groups corresponding to $Y = 0$ and $Y = 1$. In QDA, the covariance matrix is allowed to be different for the two groups. In practice, the difference in most applications is not critical.

KEY IDEAS

- Discriminant analysis works with continuous or categorical predictors, as well as with categorical outcomes.
- Using the covariance matrix, it calculates a *linear discriminant function*, which is used to distinguish records belonging to one class from those belonging to another.
- This function is applied to the records to derive weights, or scores, for each record (one weight for each possible class), which determines its estimated class.

Further Reading

- Both *The Elements of Statistical Learning*, 2nd ed., by Trevor Hastie, Robert Tibshirani, and Jerome Friedman (Springer, 2009), and its shorter cousin, *An Introduction to Statistical Learning* by Gareth

James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (Springer, R edition 2021, Python edition 2023), have a section on discriminant analysis.

- *Machine Learning for Business Analytics* by Galit Shmueli, Peter Bruce, Nitin Patel, Peter Gedeck, Inbal Yahav, and Kenneth Lichtendahl (Wiley, 2007–2025, with editions for R, Python, Excel, and JMP) has a full chapter on discriminant analysis.
- For historical interest, Fisher's original article on the topic, "The Use of Multiple Measurements in Taxonomic Problems," as published in 1936 in *Annals of Eugenics* (now called *Annals of Genetics*), can be found online.

Logistic Regression

Logistic regression is analogous to multiple linear regression (see Chapter 4), except the outcome is binary. Various transformations are employed to convert the problem to one in which a linear model can be fit. Like discriminant analysis, and unlike *K*-nearest neighbor and naive Bayes, logistic regression is a structured model approach rather than a data-centric approach. Due to its fast computational speed and its output of a model that lends itself to rapid scoring of new data, it is a popular method.

KEY TERMS FOR LOGISTIC REGRESSION

Logit

The function that maps class membership probability to a range from ± ∞ (instead of 0 to 1)

Synonym

log odds (see below)

Logistic response function

The inverse of the logit function, which maps values from ± ∞ back to the 0 to 1 range of probabilities

Synonyms

inverse logit, sigmoid function

Odds

The ratio of "success" (1) to "not success" (0)

Log odds

The response in the transformed model (now linear), which gets mapped back to a probability

Logistic Response Function and Logit

The key ingredients for logistic regression are the *logistic response function* and the *logit*, in which we map a probability (which is on a 0–1 scale) to a more expansive scale suitable for linear modeling.

The first step is to think of the outcome variable not as a binary label but as the probability p that the label is a "1." Naively, we might be tempted to model p as a linear function of the predictor variables:

$$p = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q$$

However, fitting this model does not ensure that *p* will end up between 0 and 1, as a probability must.

Instead, we model *p* by applying a *logistic response* or *inverse logit* function (also called the *sigmoid* function) to the predictors:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q)}}$$

This transform ensures that the *p* stays between 0 and 1.

To get the exponential expression out of the denominator, we consider *odds* instead of probabilities. Odds, familiar to bettors everywhere, are the ratio of "successes" (1) to "nonsuccesses" (0). In terms of probabilities, odds are the probability of an event divided by the probability that the event will not occur. For example, if the probability that a horse will win is 0.5, the probability of "won't win" is (1 – 0.5) = 0.5, and the odds are 1.0:

$$\text{Odds}\left(Y = 1\right) = \frac{p}{1 - p}$$

We can obtain the probability from the odds using the inverse odds function:

$$p = \frac{\text{Odds}}{1 + \text{Odds}}$$

We combine this with the logistic response function, shown earlier, to get:

$$\text{Odds}\left(Y = 1\right) = e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q}$$

Finally, taking the logarithm of both sides, we get an expression that involves a linear function of the predictors:

$$\log\left(\text{Odds}\left(Y = 1\right)\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q$$

The *log-odds* function, also known as the *logit* function, maps the probability *p* from $(0, 1)$ to any value $(-\infty, +\infty)$—see Figure 5-3. The transformation circle is complete; we have used a linear model to predict a probability, which we can in turn map to a class label by applying a cutoff rule—any record with a probability greater than the cutoff is classified as a 1.

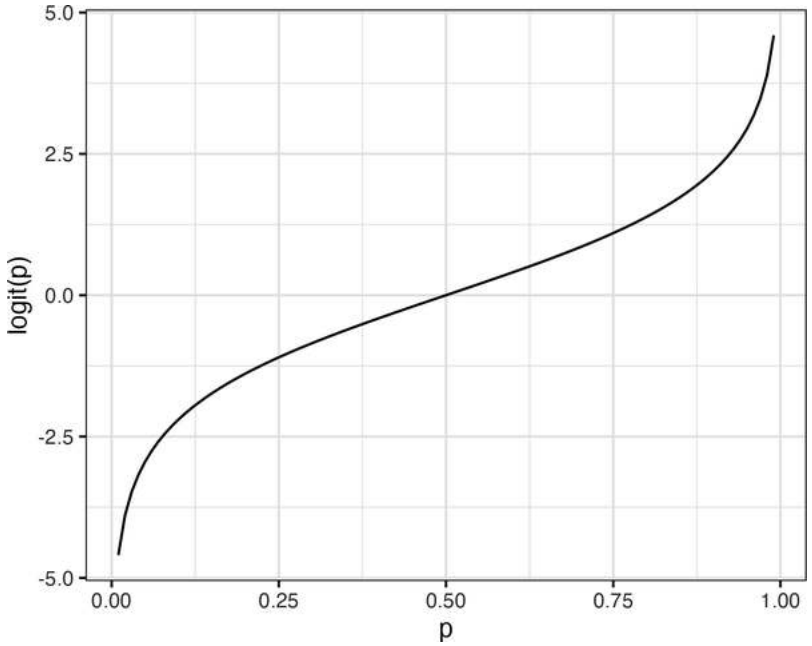


Figure 5-3. Graph of the logit function that maps a probability to a scale suitable for a linear model

Logistic Regression and the GLM

The response in the logistic regression formula is the log odds of a binary outcome of 1. We observe only the binary outcome, not the log odds, so special statistical methods are needed to fit the equation. Logistic regression is a special instance of a *generalized linear model* (GLM) developed to extend linear regression to other settings.

In R, to fit a logistic regression, the `glm` function is used with the `family` parameter set to `binomial`. The following code fits a logistic regression to the personal loan data introduced in "K-Nearest Neighbors":

```
logistic_model <- logistic_reg(mode = "classification") %>%  ❶
  fit(outcome ~ payment_inc_ratio + purpose_ +
      home_ + emp_len_ + borrower_score, loan_data)
logistic_model
---
parsnip model object

Call:  stats::glm(formula = outcome ~ payment_inc_ratio + purpose_ +
    home_ + emp_len_ + borrower_score, family = stats::binomial,
    data = data)

Coefficients:
               (Intercept)          payment_inc_ratio  purpose_debt_consolidation
                   1.63809                    0.07974                    0.24937
  purpose_home_improvement     purpose_major_purchase             purpose_medical
                   0.40774                    0.22963                    0.51048
             purpose_other     purpose_small_business                   home_OWN
                   0.62066                    1.21526                    0.04833
                 home_RENT           emp_len_> 1 Year              borrower_score
                   0.15732                   -0.35673                   -4.61264

Degrees of Freedom: 45341 Total (i.e. Null);  45330 Residual
Null Deviance:      62860
Residual Deviance: 57510        AIC: 57540
```

❶ The `tidymodels` `logistic_reg` function is used here. By default, it uses `glm` as the fitting engine. By setting `mode` to `classification`, `tidymodels` will request the correct family (binomial) for the `glm` function.

The response is `outcome`, which takes a 0 if the loan is paid off and a 1 if the loan defaults. `purpose_` and `home_` are factor variables representing the purpose of the loan and the home ownership status. As in linear regression, a factor variable with P levels is represented with $P - 1$ columns. By default in R, the *reference* coding is used, and the levels are all compared to the reference level (see "Factor Variables in Regression"). The reference levels for these factors are `credit_card` and `MORTGAGE`, respectively. The variable `borrower_score` is a score from 0 to 1 representing the creditworthiness of the borrower (from poor to excellent). This variable was created from several other variables using K-nearest neighbor (KNN)—see "KNN as a Feature Engine".

In Python, we use the `scikit-learn` class `LogisticRegression` from `sklearn.linear_model`. The arguments `penalty` and `C` are used to prevent overfitting by L1 or L2 regularization (see "Regularization: Avoiding Overfitting"). Regularization is switched on by default. In order to fit without regularization, we set `C` explicitly to `np.inf`. The `solver` argument selects the used minimizer; the method `liblinear` is the default:

```
from sklearn.linear_model import LogisticRegression

predictors = ["payment_inc_ratio", "purpose_", "home_", "emp_len_",
              "borrower_score"]
outcome = "outcome"
X = pd.get_dummies(loan_data[predictors], prefix="", prefix_sep="",
                   drop_first=True, dtype=int)
y = loan_data[outcome]

logit_reg = LogisticRegression(C=np.inf, max_iter=1000)
logit_reg.fit(X, y)
```

In contrast to R, `scikit-learn` derives the classes from the unique values in `y` (*paid off* and *default*). Internally, the classes are ordered alphabetically. As this is the reverse order from the factors used in R, you will see that the coefficients are reversed. The `predict` method returns the class label and `predict_proba` returns the probabilities in the order available from the attribute `logit_reg.classes_`.

Generalized Linear Models

Generalized linear models (GLMs) are characterized by two main components:

- A probability distribution or family (binomial in the case of logistic regression)
- A link function—i.e., a transformation function that maps the response to the predictors (logit in the case of logistic regression)

Logistic regression is by far the most common form of GLM. A data scientist will encounter other types of GLMs. Sometimes a log link function is used instead of the logit; in practice, use of a log link is unlikely to lead to very different results for most applications. The Poisson distribution is commonly used to model count data (e.g., the number of times a user visits a web page in a certain amount of time). Other families include negative binomial and gamma, often used to model elapsed time (e.g., time to failure). In contrast to logistic regression, application of GLMs with these models is more nuanced and involves greater care. These are best avoided unless you are familiar with and understand the utility and pitfalls of these methods.

Predicted Values from Logistic Regression

The predicted value from logistic regression is in terms of the log odds: $\hat{Y} = \log(\text{Odds}(Y = 1))$. The predicted probability is given by the logistic response function:

$$\hat{p} = \frac{1}{1 + e^{-\hat{Y}}}$$

For example, look at the predictions from the model `logistic_model` in R:

```
pred <- predict(logistic_model, new_data = loan_data, type = "raw")
summary(pred)
---
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-2.704774 -0.518825 -0.008539  0.002564  0.505061  3.509606
```

In Python, we can convert the probabilities into a data frame and use the `describe` method to get these characteristics of the distribution:

```
pred = pd.DataFrame(logit_reg.predict_log_proba(X),
                    columns=logit_reg.classes_)
pred.describe()
```

Converting these values to probabilities is a simple transform:

```
prob <- 1 / (1 + exp(-pred))
summary(prob)
---
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.06269 0.37313 0.49787 0.50000 0.62365 0.97096
```

Instead of computing the log odds and then transforming to probabilities, we can get the probabilities directly in the same way we have seen for the KNN model:

```
prob <- predict(logistic_model, new_data = loan_data, type = "prob")  ❶
summary(prob$.pred_default)
```

The probabilities are directly available using the `predict_proba` methods in `scikit-learn`:

```
pred = pd.DataFrame(logit_reg.predict_proba(X),
                    columns=logit_reg.classes_)
pred.describe()
```

These are on a scale from 0 to 1 and don't yet declare whether the predicted value is default or paid off. We could declare any value greater than 0.5 as default. In practice, a lower cutoff is often appropriate if the goal is to identify members of a rare class (see "The Rare Class Problem").

Interpreting the Coefficients and Odds Ratios

One advantage of logistic regression is that it produces a model that can be scored to new data rapidly, without recomputation. Another is the relative ease of interpretation of the model, as compared with other classification methods. The key conceptual idea is understanding an *odds ratio*. The odds ratio is easiest to understand for a binary factor variable X:

$$\text{odds ratio} = \frac{\text{Odds}(Y = 1|X = 1)}{\text{Odds}(Y = 1|X = 0)}$$

This is interpreted as the odds that $Y = 1$ when $X = 1$ versus the odds that $Y = 1$ when $X = 0$. If the odds ratio is 2, then the odds that $Y = 1$ are two times higher when $X = 1$ versus when $X = 0$.

Why bother with an odds ratio rather than probabilities? We work with odds because the coefficient β_j in the logistic regression is the log of the odds ratio for X_j.

An example will make this more explicit. For the model fit in "Logistic Regression and the GLM", the regression coefficient for `purpose_small_business` is 1.21526. This means that a loan to a small business compared to a loan to pay off credit card debt increases the odds of defaulting versus being paid off by $exp(1.21526) \approx 3.4$. Clearly, loans for the purpose of creating or expanding a small business are considerably riskier than other types of loans.

Figure 5-4 shows the relationship between the odds ratio and the log-odds ratio for odds ratios greater than 1. Because the coefficients are on the log scale, an increase of 1 in the coefficient results in an increase of $exp(1) \approx 2.72$ in the odds ratio.

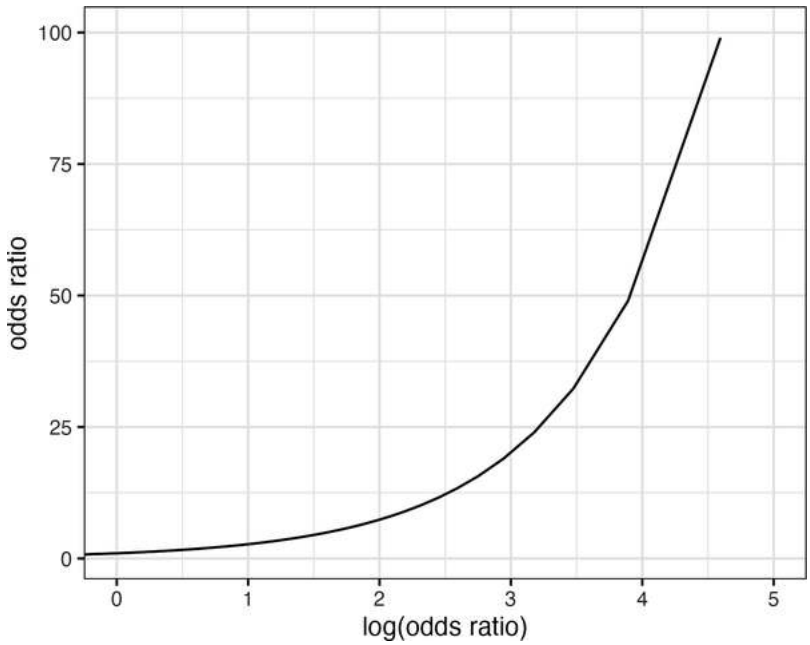


Figure 5-4. The relationship between the odds ratio and the log-odds ratio

Odds ratios for numeric variables X can be interpreted similarly: they measure the change in the odds ratio for a unit change in X. For example, the effect of increasing the payment-to-income ratio from, say, 5 to 6 increases the odds of the loan defaulting by a factor of $exp(0.08244) \approx 1.09$. The variable `borrower_score` is a score on the borrowers' creditworthiness and ranges from 0 (low) to 1 (high). The odds of the best borrowers relative to the worst borrowers defaulting on their loans are smaller by a factor of $exp(-4.61264) \approx 0.01$. In other words, the default risk from the borrowers with the poorest creditworthiness is 100 times greater than that of the best borrowers!

Linear and Logistic Regression: Similarities and Differences

Linear regression and logistic regression share many commonalities. Both assume a parametric linear form relating the predictors with the response. Exploring and finding the best model are done in very similar ways. Extensions to the linear model, like the use of a spline transform of a predictor (see "Splines"), are equally applicable in the logistic regression setting. Logistic regression differs in two fundamental ways:

- The way the model is fit (least squares is not applicable)
- The nature and analysis of the residuals from the model

Fitting the model

Linear regression is fit using least squares, and the quality of the fit is evaluated using RMSE and R-squared statistics. In logistic regression (unlike in linear regression), there is no closed-form solution, and the model must be fit using *maximum likelihood estimation* (MLE). MLE is a process that tries to find the model that is most likely to have produced the data we see. In the logistic regression equation, the response is not 0 or 1 but rather an estimate of the log odds that the response is 1. The MLE finds the solution such that the estimated log odds best describe the observed outcome. The mechanics of the algorithm involve a quasi-Newton optimization that iterates between a scoring step (*Fisher's scoring*), based on the current parameters, and an update to the parameters to improve the fit.

MAXIMUM LIKELIHOOD ESTIMATION

Here is a bit more detail, if you like statistical symbols: start with a set of data $(X_1, X_2, \cdots, X_n)$ and a probability model $P_\theta(X_1, X_2, \cdots, X_n)$ that depends on a set of parameters θ. The goal of MLE is to find the set of parameters $\hat{\theta}$ that maximizes the value of $P_\theta(X_1, X_2, \cdots, X_n)$; that is, it maximizes the probability of observing $(X_1, X_2, \cdots, X_n)$ given the model P. In the fitting process, the model is evaluated using a metric called *deviance*:

$$\text{deviance} = -2 \log \left(P_{\hat{\theta}} \left(X_1, X_{2,\ldots,} ,\ldots, X_n \right) \right)$$

Lower deviance corresponds to a better fit.

Fortunately, most practitioners don't need to concern themselves with the details of the fitting algorithm since this is handled by the software. Most data scientists will not need to worry about the fitting method, other than understanding that it is a way to find a good model under certain assumptions.

HANDLING FACTOR VARIABLES

In logistic regression, factor variables should be coded as in linear regression; see "Factor Variables in Regression". In R and other software, this is normally handled automatically, and generally reference encoding is used. All of the other classification methods covered in this chapter typically use the one-hot encoder representation (see "One-Hot Encoder"). In Python's `scikit-learn`, we expand the factor variable into several columns using `get_dummies()`, which means that only $n - 1$ of the resulting dummies can be used in the regression.

Assessing the Model

Like other classification methods, logistic regression is assessed by how accurately the model classifies new data (see "Evaluating Classification Models"). As with linear regression, some additional standard statistical tools are available to examine and improve the model. Along with the estimated coefficients, R reports the standard error of the coefficients (SE), a *z*-value, and a p-value:

```
logistic_model %>%
  extract_fit_engine() %>%
  summary()
---
Call:
stats::glm(formula = outcome ~ payment_inc_ratio + purpose_ +
    home_ + emp_len_ + borrower_score, family = stats::binomial,
    data = data)

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)               1.638092   0.073708  22.224  < 2e-16 ***
payment_inc_ratio         0.079737   0.002487  32.058  < 2e-16 ***
purpose_debt_consolidation  0.249373   0.027615   9.030  < 2e-16 ***
purpose_home_improvement    0.407743   0.046615   8.747  < 2e-16 ***
purpose_major_purchase      0.229628   0.053683   4.277 1.89e-05 ***
purpose_medical             0.510479   0.086780   5.882 4.04e-09 ***
purpose_other               0.620663   0.039436  15.738  < 2e-16 ***
purpose_small_business      1.215261   0.063320  19.192  < 2e-16 ***
home_OWN                    0.048330   0.038036   1.271    0.204
home_RENT                   0.157320   0.021203   7.420 1.17e-13 ***
emp_len_> 1 Year           -0.356731   0.052622  -6.779 1.21e-11 ***
borrower_score             -4.612638   0.083558 -55.203  < 2e-16 ***
--
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 62857  on 45341  degrees of freedom
Residual deviance: 57515  on 45330  degrees of freedom
```

```
AIC: 57539

Number of Fisher Scoring iterations: 4
```

The package `statsmodels` has an implementation for generalized linear model (`GLM`) that provides similarly detailed information:

```
import statsmodels.api as sm

y_numbers = [1 if yi == "default" else 0 for yi in y]
logit_reg_sm = sm.GLM(y_numbers, X.assign(const=1),
                      family=sm.families.Binomial())
logit_result = logit_reg_sm.fit()
print(logit_result.summary())
```

Interpretation of the p-value comes with the same caveat as in regression and should be viewed more as a relative indicator of variable importance (see "Assessing the Model") than as a formal measure of statistical significance. A logistic regression model, which has a binary response, does not have an associated RMSE or R-squared. Instead, a logistic regression model is typically evaluated using more general metrics for classification; see "Evaluating Classification Models".

Many other concepts for linear regression carry over to the logistic regression setting (and other GLMs). For example, you can use stepwise regression, fit interaction terms, or include spline terms. The same concerns regarding confounding and correlated variables apply to logistic regression (see "Interpreting the Regression Equation"). You can fit GAM (see "Generalized Additive Models") using the `mgcv` package in R:

```
library(mgcv)
gam_formula <- outcome ~ s(payment_inc_ratio) + purpose_ +  ❶
  home_ + emp_len_ + s(borrower_score)
logistic_gam <- gen_additive_mod(mode = "classification") %>%  ❷
  fit(gam_formula, data = loan_data)
```

❶ We identify the spline terms using the `s()` function from `mgcv`.

❷ The `gen_additive_mod` function from `tidymodels` is used here. The default engine is the `mgcv` package.

The formula interface of `statsmodels` also supports these extensions in Python:

```
import statsmodels.formula.api as smf
formula = ("outcome ~ bs(payment_inc_ratio, df=4) + purpose_ + "
           "home_ + emp_len_ + bs(borrower_score, df=4)")
model = smf.glm(formula=formula, data=loan_data, family=sm.families.Binomial())
results = model.fit()
```

Analysis of residuals

Logistic regression differs from linear regression in the analysis of the residuals. As in linear regression (see Figure 4-9), it's straightforward to compute partial residuals in R:

```
terms <- predict(logistic_gam %>% extract_fit_engine(), type = "terms")
partial_resid <- resid(logistic_gam %>% extract_fit_engine()) + terms
df <- data.frame(payment_inc_ratio = loan_data[, "payment_inc_ratio"],
  terms = terms[, "s(payment_inc_ratio)"],
  partial_resid = partial_resid[, "s(payment_inc_ratio)"])
ggplot(df, aes(x = payment_inc_ratio, y = partial_resid, solid = FALSE)) +
  geom_point(shape = 46, alpha = 0.4) +
  geom_line(aes(x = payment_inc_ratio, y = terms),
    color = "red", alpha = 0.5, linewidth = 1.5) +
  labs(y = "Partial Residual") +
  coord_cartesian(xlim = c(0, 25))
```

The resulting plot is displayed in Figure 5-5. The estimated fit, shown by the line, goes between two sets of point clouds. The top cloud corresponds to a response of 1 (defaulted loans), and the bottom cloud corresponds to a response of 0 (loans paid off). This is very typical of residuals from a logistic regression since the output is binary. The prediction is measured as the logit (log of the odds ratio), which will always be some finite value. The actual value, an absolute 0 or 1, corresponds to an infinite logit, either positive or negative, so the residuals (which get added to the fitted value) will never equal 0. Hence the plotted points lie in clouds either above or below the fitted line in the partial residual plot. Partial residuals in logistic regression, while less valuable than in regression, are still useful to confirm nonlinear behavior and identify highly influential records.

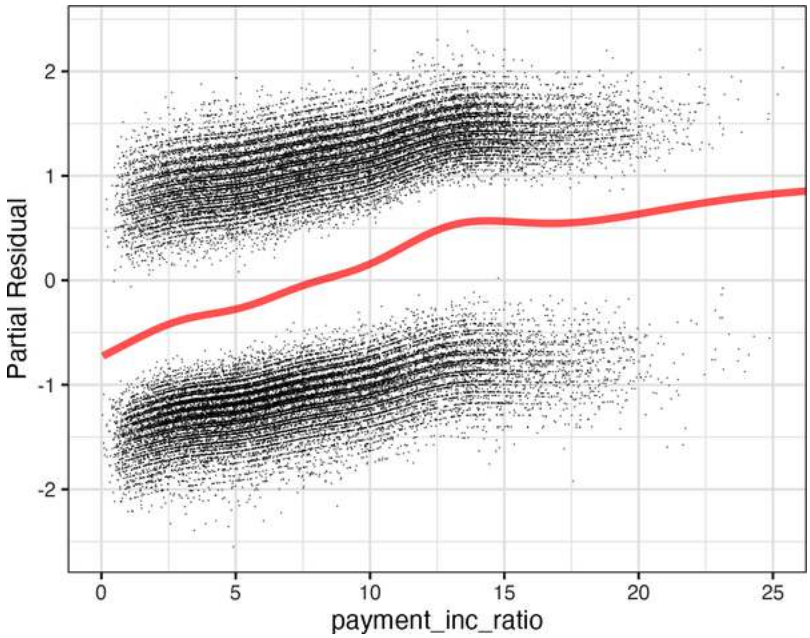


Figure 5-5. Partial residuals from logistic regression

There is currently no implementation of partial residuals in any of the major Python packages. We provide Python code to create the partial residual plot in the accompanying source code repository.

OUTPUT THAT CAN BE IGNORED

Some of the output from the `summary` function can effectively be ignored. The dispersion parameter does not apply to logistic regression and is used for other types of GLMs. The residual deviance and the number of scoring iterations are related to the maximum likelihood fitting method; see "Maximum Likelihood Estimation".

KEY IDEAS

- Logistic regression is like linear regression, except that the outcome is a binary variable.
- Several transformations are needed to get the model into a form that can be fit as a linear model, with the log of the odds ratio as the response variable.
- After the linear model is fit (by an iterative process), the log odds are mapped back to a probability.
- Logistic regression is popular because it is computationally fast and produces a model that can be scored to new data with only a few arithmetic operations.

Further Reading

- The standard reference on logistic regression is *Applied Logistic Regression*, 3rd ed., by David Hosmer, Stanley Lemeshow, and Rodney Sturdivant (Wiley, 2013).
- Also popular are two books by Joseph Hilbe: *Logistic Regression Models* (very comprehensive, 2017) and *Practical Guide to Logistic Regression* (compact, 2015), both from Chapman & Hall/CRC Press.
- Both *The Elements of Statistical Learning*, 2nd ed., by Trevor Hastie, Robert Tibshirani, and Jerome Friedman (Springer, 2009), and its shorter cousin, *An Introduction to Statistical Learning* by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (Springer, R edition 2021, Python edition 2023), have a section on logistic regression.
- *Machine Learning for Business Analytics* by Galit Shmueli, Peter Bruce, Nitin Patel, Peter Gedeck, Inbal Yahav, and Kenneth Lichtendahl (Wiley, 2007–2025, with editions for R, Python, Excel, RapidMiner, and JMP) has a full chapter on logistic regression.

Evaluating Classification Models

It is common in predictive modeling to train a number of different models, apply each to a holdout sample, and assess their performance. Sometimes, after a number of models have been evaluated and tuned, and if there is enough data, a third holdout sample, not used previously, is used to estimate how the chosen model will perform with completely new data. Different disciplines and practitioners will also use the terms *validation* and *test* to refer to the holdout sample(s). Fundamentally, the assessment process attempts to learn which model produces the most accurate and useful predictions.

KEY TERMS FOR EVALUATING CLASSIFICATION MODELS

Accuracy

The percent (or proportion) of cases classified correctly.

Confusion matrix

A tabular display (2 × 2 in the binary case) of the record counts by their predicted and actual classification status.

Sensitivity

The percent (or proportion) of all 1s that are correctly classified as 1s. We label the class of interest with "1" in this section. However, many implementations, e.g., `tidymodels`, are able to directly use more descriptive factor labels, e.g., "positive."

Synonym

recall

Specificity

The percent (or proportion) of all 0s that are correctly classified as 0s.

Precision

The percent (proportion) of predicted 1s that are actually 1s.

ROC curve

A plot of sensitivity versus specificity.

Lift

A measure of how effective the model is at identifying (comparatively rare) 1s at different probability cutoffs.

A simple way to measure classification performance is to count the proportion of predictions that are correct, i.e., measure the *accuracy*. Accuracy is simply a measure of total error:

$$\text{accuracy} = \frac{\sum \text{True Positive} + \sum \text{True Negative}}{\text{Sample Size}}$$

In most classification algorithms, each case is assigned an "estimated probability of being a 1."[3] The default decision point, or cutoff, is typically 0.50 or 50%. If the probability is above 0.5, the classification is "1"; otherwise it is "0." An alternative default cutoff is the prevalent probability of 1s in the data.

Confusion Matrix

At the heart of classification metrics is the *confusion matrix*. The confusion matrix is a table showing the number of correct and incorrect predictions categorized by type of response. Several packages are available in R and Python to compute a confusion matrix, but in the binary case, it is simple to compute one by hand.

To illustrate the confusion matrix, consider the `logistic_gam` model that was trained on a balanced data set with an equal number of defaulted and paid-off loans (see Figure 5-5). Following the usual conventions, $Y = 1$

corresponds to the event of interest (e.g., default), and $Y = 0$ corresponds to a negative (or usual) event (e.g., paid off). The following computes the confusion matrix for the `logistic_gam` model applied to the entire (unbalanced) training set in R:

```
predictions <- augment(logistic_gam, loan_data)
pred_y <- as.numeric(predictions$.pred_class == "default")
true_y <- as.numeric(predictions$outcome == "default")
true_pos <- (true_y == 1) & (pred_y == 1)
true_neg <- (true_y == 0) & (pred_y == 0)
false_pos <- (true_y == 0) & (pred_y == 1)
false_neg <- (true_y == 1) & (pred_y == 0)
conf_mat <- matrix(
  c(sum(true_neg), sum(false_pos),
    sum(false_neg), sum(true_pos)), 2, 2)
rownames(conf_mat) <- c("Yhat = 0", "Yhat = 1")
colnames(conf_mat) <- c("Y = 0", "Y = 1")
conf_mat
---
         Y = 0 Y = 1
Yhat = 0 14620  8378
Yhat = 1  8051 14293
```

The `yardstick` package of `tidymodels` has a variety of performance measures for classification and regression. We can use the `conf_mat` function to compute the confusion matrix directly:

```
conf_mat(predictions, truth = outcome, estimate = .pred_class)  ❶
---
          Truth
Prediction paid off default
  paid off    14620    8378
  default      8051   14293
```

❶ Other functions in the `yardstick` package follow a similar pattern and first take the data frame followed by arguments to specify the `truth` and `estimate` columns.

The predicted outcomes are rows and the true outcomes are the columns. The diagonal elements of the matrix show the number of correct predictions, and the off-diagonal elements show the number of incorrect predictions. For example, 14,293 defaulted loans were correctly predicted as a default, but 8,378 defaulted loans were incorrectly predicted as paid off.

In Python, we can compute the confusion matrix for the `logit_reg` model as follows:

```
pred = logit_reg.predict(X)
pred_y = logit_reg.predict(X) == "default"
true_y = y == "default"
true_pos = true_y & pred_y
true_neg = ~true_y & ~pred_y
false_pos = ~true_y & pred_y
false_neg = true_y & ~pred_y

conf_mat = pd.DataFrame([[np.sum(true_pos), np.sum(false_neg)],
                         [np.sum(false_pos), np.sum(true_neg)]],
                        index=["Y = default", "Y = paid off"],
                        columns=["Yhat = default", "Yhat = paid off"])
conf_mat
---
             Yhat = default     Yhat = paid off
Y = default      14321          8350
Y = paid off     8140           14531
```

Note that here, we show the confusion matrix with the actual outcomes as the rows and the predicted outcomes as the columns. We made this choice to be in agreement with the convention used in the `scikit-learn` package:

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_true=y, y_pred=logit_reg.predict(X)))  ❶
---
[[14321  8350]
 [ 8140 14531]]
```

❶ This and other metrics in `scikit-learn` first expect an array with the actual outcome followed by an array with the predicted outcome. It is important to ensure that this order is preserved. Specifying the argument names, as we did here, helps avoid confusion.

If you want a more detailed and clearer form of the confusion matrix, you can use the `classificationSummary` of the `mlba` package:

```
import mlba
mlba.classificationSummary(y_true=y, y_pred=logit_reg.predict(X))
---
Confusion Matrix (Accuracy 0.6363)
         Prediction
  Actual  default paid off
 default    14321     8350
paid off     8140    14531
```

Figure 5-6 shows the relationship between the confusion matrix for a binary response Y and different metrics (see "Precision, Recall, and Specificity" for more on the metrics). As with the example for the loan data, the actual response is along the rows and the predicted response is along the columns. The diagonal boxes (upper left, lower right) show when the predictions $\hat{Y}$ correctly predict the response. One important metric not explicitly called out is the *false positive rate* (the mirror image of precision). When 1s are rare, the ratio of false positives to all predicted positives can be high, leading to the unintuitive situation in which a predicted 1 is most likely a 0. This problem plagues medical screening tests (e.g., mammograms) that are widely applied: due to the relative rarity of the condition, positive test results most likely do not mean breast cancer. This leads to much confusion among the public.

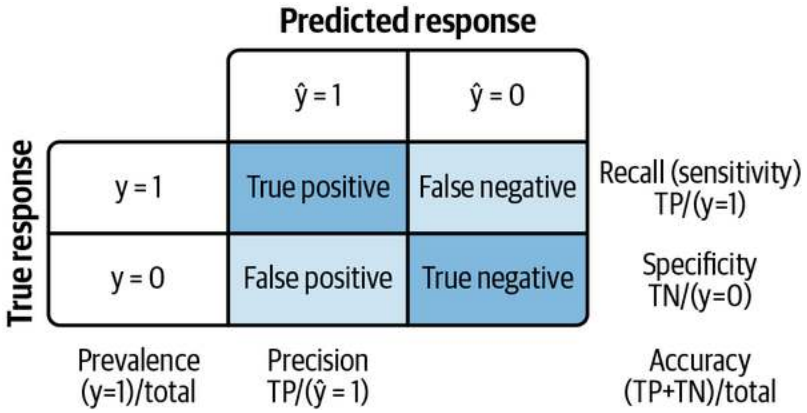


Figure 5-6. Confusion matrix for a binary response and various metrics

> **ACTUAL AND PREDICTED MAY BE REVERSED**
>
> In Figure 5-6, we present the actual response along the rows and the predicted response along the columns, but it is not uncommon to see this reversed. In fact, we've seen this difference between the R and the Python frameworks.

The Rare Class Problem

In many cases, there is an imbalance in the classes to be predicted, with one class much more prevalent than the other—for example, legitimate insurance claims versus fraudulent ones, or browsers versus purchasers at a

website. The rare class (e.g., the fraudulent claims) is usually the class of more interest and is typically designated 1, in contrast to the more prevalent 0s. In the typical scenario, the 1s are the more important case, in the sense that misclassifying them as 0s is costlier than misclassifying 0s as 1s. For example, correctly identifying a fraudulent insurance claim may save thousands of dollars. On the other hand, correctly identifying a nonfraudulent claim merely saves you the cost and effort of going through the claim by hand with a more careful review (which is what you would do if the claim were tagged as "fraudulent").

In such cases, unless the classes are easily separable, the most accurate classification model may be one that simply classifies everything as a 0. For example, if only 0.1% of the browsers at a web store end up purchasing, a model that predicts that each browser will leave without purchasing will be 99.9% accurate. However, it will be useless. Instead, we would be happy with a model that is less accurate overall but is good at picking out the purchasers, even if it misclassifies some nonpurchasers along the way.

Precision, Recall, and Specificity

Metrics other than pure accuracy—metrics that are more nuanced—are commonly used in evaluating classification models. Several of these have a long history in statistics—especially biostatistics, where they are used to describe the expected performance of diagnostic tests. The *precision* measures the accuracy of a predicted positive outcome (see Figure 5-6):

$$\text{precision} = \frac{\sum \text{True Positive}}{\sum \text{True Positive} + \sum \text{False Positive}}$$

The *recall*, also known as *sensitivity*, measures the strength of the model to predict a positive outcome—the proportion of the 1s that it correctly identifies (see Figure 5-6). The term *sensitivity* is used a lot in biostatistics and medical diagnostics, whereas *recall* is used more in the machine learning community. The definition of recall is:

$$\text{recall} = \frac{\sum \text{True Positive}}{\sum \text{True Positive} + \sum \text{False Negative}}$$

Another metric used is *specificity*, which measures a model's ability to predict a negative outcome:

$$\text{specificity} = \frac{\sum \text{True Negative}}{\sum \text{True Negative} + \sum \text{False Positive}}$$

We could calculate the three metrics from `conf_mat` in R; however, it is more straightforward to use implementations from `yardstick`:

```
bind_rows(
  precision(predictions, truth = outcome, estimate = .pred_class),
  recall(predictions, truth = outcome, estimate = .pred_class),
  specificity(predictions, truth = outcome, estimate = .pred_class),
) %>% select(-.estimator)
---
.metric     .estimate
precision         0.6357075
recall      0.6448767
specificity     0.6304530
```

Here is the equivalent code to calculate the metrics in Python:

```
from sklearn.metrics import precision_score, recall_score

y_pred = logit_reg.predict(X)
print("precision ", precision_score(y, y_pred, pos_label="default"))
print("recall.    ", recall_score(y, y_pred, pos_label="default"))
print("specificity", recall_score(y, y_pred, pos_label="paid off"))  ❶
---
```

```
precision   0.6375940519122034
recall      0.6316880596356579
specificity 0.640950994662785
```

❶ There is no direct function to compute specificity. Instead provide the *negative* class in the call to `recall_score`.

ROC Curve

You can see that there is a trade-off between recall and specificity. Capturing more 1s generally means misclassifying more 0s as 1s. The ideal classifier would do an excellent job of classifying the 1s, without misclassifying more 0s as 1s.

The metric that captures this trade-off is the "*receiver operating characteristics*" curve, usually referred to as the *ROC curve*. The ROC curve plots recall (sensitivity) on the y-axis against specificity on the x-axis.[4] The ROC curve shows the trade-off between recall and specificity as you change the cutoff to determine how to classify a record. Sensitivity (recall) is plotted on the y-axis, and you may encounter two forms in which the x-axis is labeled:

- Specificity plotted on the x-axis, with 1 on the left and 0 on the right
- 1 – specificity plotted on the x-axis, with 0 on the left and 1 on the right

The curve looks identical whichever way it is done. The process to compute the ROC curve is:

1. Sort the records by the predicted probability of being a 1, starting with the most probable and ending with the least probable.
2. Compute the cumulative specificity and recall based on the sorted records.

Computing the ROC curve in R is straightforward. The following code computes ROC for the loan data using the `yardstick` package of `tidymodels`:

```
roc_curve(predictions, outcome, .pred_default, event_level = "second") %>%  ❶
  autoplot()
```

❶ In this function, we specify the `event_level` argument to indicate that the second level of the factor (i.e., "default") is the event of interest. It must match with the column that contains the predicted probabilities (here, `.pred_default`). Be careful to ensure that you specify the correct class as the event of interest; otherwise, the ROC curve will be incorrect. Try this out by changing `second` to `first` and see how the ROC curve changes.

In Python, we can use the `scikit-learn` function `sklearn.metrics.roc_curve` to calculate the required information for the ROC curve:

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y, logit_reg.predict_proba(X)[:, 0],
                                 pos_label="default")
roc_df = pd.DataFrame({"recall": tpr, "specificity": 1 - fpr})

ax = roc_df.plot(x="specificity", y="recall", figsize=(4, 4), legend=False)
ax.set_ylim(0, 1)
ax.set_xlim(1, 0)
ax.plot((1, 0), (0, 1))
ax.set_xlabel("specificity")
ax.set_ylabel("recall")
```

The result is shown in Figure 5-7. The dotted diagonal line corresponds to a classifier no better than random chance. An extremely effective classifier (or, in medical situations, an extremely effective diagnostic test) will have

an ROC that hugs the upper-left corner—it will correctly identify lots of 1s without misclassifying lots of 0s as 1s. For this model, if we want a classifier with a specificity of at least 50%, then the recall/sensitivity is about 75%.

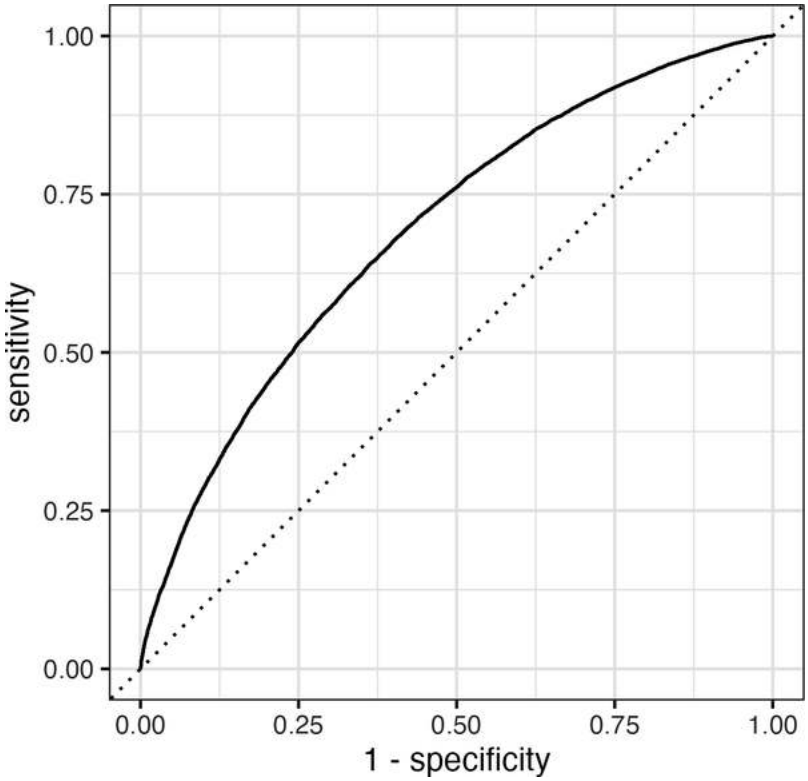


Figure 5-7. ROC curve for the loan data

> **PRECISION-RECALL CURVE**
>
> In addition to ROC curves, it can be illuminating to examine the precision-recall (PR) curve. PR curves are computed in a similar way except that the data is ordered from least to most probable, and cumulative precision and recall statistics are computed. PR curves are especially useful in evaluating data with highly unbalanced outcomes.

AUC

The ROC curve is a valuable graphical tool, but by itself doesn't constitute a single measure for the performance of a classifier. The ROC curve can be used, however, to produce the area underneath the curve (AUC) metric. AUC is simply the total area under the ROC curve. The larger the value of AUC, the more effective the classifier. An AUC of 1 indicates a perfect classifier: it gets all the 1s correctly classified, and it doesn't misclassify any 0s as 1s.

A completely ineffective classifier—the diagonal line—will have an AUC of 0.5.

Figure 5-8 shows the area under the ROC curve for the loan model. The value of AUC can be computed by a numerical integration in R:

```
sum(roc_df$recall[-1] * diff(1 - roc_df$specificity))
---
    [1] 0.6926232
```

In Python, we can either calculate the accuracy as shown for R or use `scikit-learn`'s function `sklearn.metrics.roc_auc_score`. You will need to provide the expected value as 0 or 1:

```
from sklearn.metrics import roc_auc_score

print(np.sum(roc_df.recall[:-1] * np.diff(1 - roc_df.specificity)))
print(roc_auc_score([1 if yi == "default" else 0 for yi in y],
                    logit_reg.predict_proba(X)[:, 0]))
```

The model has an AUC of about 0.69, corresponding to a relatively weak classifier.

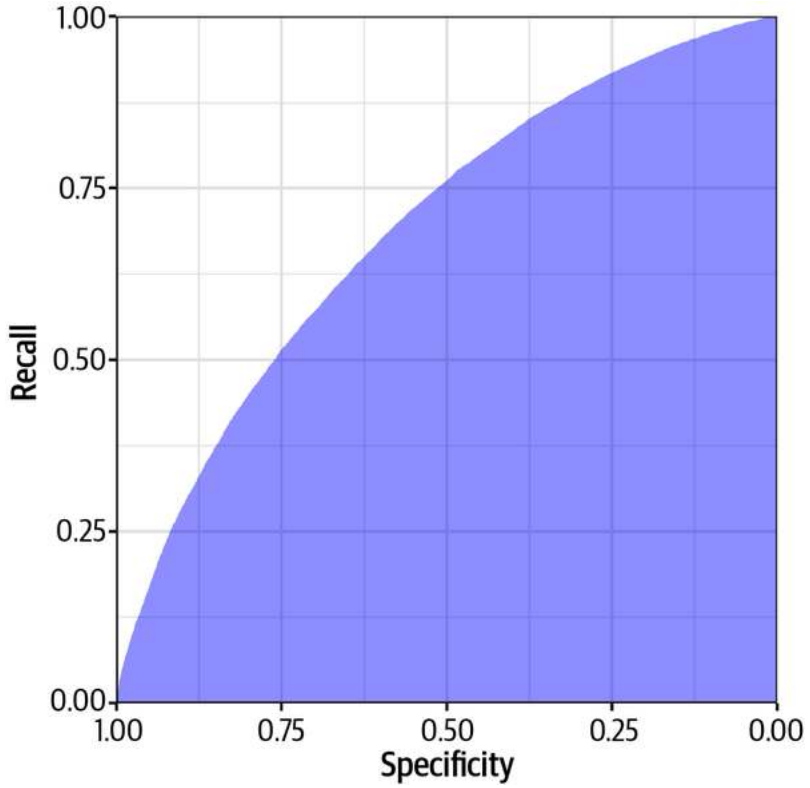


Figure 5-8. Area under the ROC curve for the loan data

> **FALSE POSITIVE RATE CONFUSION**
>
> False positive/negative rates are often confused or conflated with specificity or sensitivity (even in publications and software!). Sometimes the false positive rate is defined as the proportion of true negatives that test positive. In many cases (such as network intrusion detection), the term is used to refer to the proportion of positive signals that are true negatives.

Lift

Using the AUC as a metric to evaluate a model is an improvement over simple accuracy, as it can assess how well a classifier handles the trade-off between overall accuracy and the need to identify the more important 1s. But it does not completely address the rare-case problem, where you need to lower the model's probability cutoff below 0.5 to avoid having all records classified as 0. In such cases, for a record to be classified as a 1, it might be sufficient to have a probability of 0.4, 0.3, or lower. In effect, we end up overidentifying 1s, reflecting their greater importance.

Changing this cutoff will improve your chances of catching the 1s (at the cost of misclassifying more 0s as 1s). But what is the optimum cutoff?

The concept of lift lets you defer answering that question. Instead, you consider the records in order of their predicted probability of being 1s. Say, of the top 10% classified as 1s, how much better did the algorithm do, compared to the benchmark of simply picking blindly? If you can get 0.3% response in this top decile instead of

the 0.1% you get overall by picking randomly, the algorithm is said to have a *lift* (also called *gains*) of 3 in the top decile. A lift chart (gains chart) quantifies this over the range of the data. It can be produced decile by decile, or continuously over the range of the data.

To compute a lift chart, you first produce a *cumulative gains chart* that shows the recall on the y-axis and the total number of records on the x-axis. The *lift curve* is the ratio of the cumulative gains to the diagonal line corresponding to random selection. *Decile gains charts* are one of the oldest techniques in predictive modeling, dating from the days before internet commerce. They were particularly popular among direct mail professionals. Direct mail is an expensive method of advertising if applied indiscriminately, and advertisers used predictive models (quite simple ones, in the early days) to identify the potential customers with the likeliest prospect of payoff.

UPLIFT

Sometimes the term *uplift* is used to mean the same thing as lift. An alternate meaning is used in a more restrictive setting, when an A/B test has been conducted and the treatment (A or B) is then used as a predictor variable in a predictive model. The uplift is the improvement in response predicted *for an individual case* with treatment A versus treatment B. This is determined by scoring the individual case first with the predictor set to A, and then again with the predictor toggled to B. Marketers and political campaign consultants use this method to determine which of two messaging treatments should be used with which customers or voters.

A lift curve lets you look at the consequences of setting different probability cutoffs for classifying records as 1s. It can be an intermediate step in settling on an appropriate cutoff level. For example, a tax authority might have only a certain amount of resources that it can spend on tax audits, and it wants to spend them on the likeliest tax cheats. With its resource constraint in mind, the authority would use a lift chart to estimate where to draw the line between tax returns selected for audit and those left alone.

KEY IDEAS

- Accuracy (the percent of predicted classifications that are correct) is but a first step in evaluating a model.
- Other metrics (recall, specificity, precision) focus on more specific performance characteristics (e.g., recall measures how good a model is at correctly identifying 1s).
- AUC (area under the ROC curve) is a common metric for the ability of a model to distinguish 1s from 0s.
- Similarly, lift measures how effective a model is in identifying the 1s, and it is often calculated decile by decile, starting with the most probable 1s.

Further Reading

Evaluation and assessment are typically covered in the context of a particular model (e.g., KNNs or decision trees); three books that handle the subject in its own chapter are:

- *Data Mining*, 3rd ed., by Ian Witten, Eibe Frank, and Mark Hall (Morgan Kaufmann, 2011).
- *Modern Data Science with R* by Benjamin Baumer, Daniel Kaplan, and Nicholas Horton (Chapman & Hall/CRC Press, 2017).
- *Machine Learning for Business Analytics* by Galit Shmueli, Peter Bruce, Nitin Patel, Peter Gedeck, Inbal Yahav, and Kenneth Lichtendahl (Wiley, 2007–2025, with editions for R, Python, Excel, RapidMiner, and JMP).

Strategies for Imbalanced Data

The previous section dealt with evaluation of classification models using metrics that go beyond simple accuracy and are suitable for imbalanced data—data in which the outcome of interest (purchase on a website, insurance fraud, etc.) is rare. In this section, we look at additional strategies that can improve predictive modeling performance with imbalanced data.

KEY TERMS FOR IMBALANCED DATA

Undersample

Use fewer of the prevalent class records in the classification model.

Synonym

downsample

Oversample

Use more of the rare class records in the classification model, bootstrapping if necessary.

Synonym

upsample

Up weight or down weight

Attach more (or less) weight to the rare (or prevalent) class in the model.

Data generation

Like bootstrapping, except each new bootstrapped record is slightly different from its source.

z-score

The value that results after standardization.

K

The number of neighbors considered in the nearest neighbor calculation.

Undersampling

If you have enough data, as is the case with the loan data, one solution is to *undersample* (or downsample) the prevalent class, so the data to be modeled is more balanced between 0s and 1s. The basic idea in undersampling is that the data for the dominant class has many redundant records. Dealing with a smaller, more balanced data set yields benefits in model performance, and it makes it easier to prepare the data and to explore and pilot models.

How much data is enough? It depends on the application, but in general, having tens of thousands of records for the less dominant class is enough. The more easily distinguishable the 1s are from the 0s, the less data needed.

The loan data analyzed in "Logistic Regression" was based on a balanced training set: half of the loans were paid off, and the other half were in default. The predicted values were similar: half of the probabilities were less than 0.5, and half were greater than 0.5. In the full data set, only about 19% of the loans were in default, as shown in R:

```
full_train_set <- read_csv("full_train_set.csv.gz", show_col_types = FALSE) %>%
  mutate(
    across(where(is.character), as.factor),
    outcome = factor(outcome, levels = c("paid off", "default")),
  )

mean(full_train_set$outcome == "default")
---
[1] 0.1889455
```

In Python:

```
full_train_set = pd.read_csv("full_train_set.csv.gz")

print("percentage of loans in default: ",
      100 * np.mean(full_train_set.outcome == "default"))
```

What happens if we use the full data set to train the model? Let's see what this looks like in R:

```
full_model <- logistic_reg(mode = "classification") %>%
  fit(outcome ~ payment_inc_ratio + purpose_ +
      home_ + emp_len_ + dti + revol_bal + revol_util,
    full_train_set)
pred <- predict(full_model, new_data = full_train_set)
100 * mean(pred == "default")
---
[1] 0.3942094
```

And in Python:

```
predictors = ["payment_inc_ratio", "purpose_", "home_", "emp_len_",
              "dti", "revol_bal", "revol_util"]
outcome = "outcome"
X = pd.get_dummies(full_train_set[predictors], prefix="", prefix_sep="",
                   drop_first=True, dtype=int)
y = full_train_set[outcome]

full_model = LogisticRegression(C=np.inf, max_iter=10_000)
full_model.fit(X, y)
print("percentage of loans predicted to default: ",
      100 * np.mean(full_model.predict(X) == "default"))
```

Only 0.39% of the loans are predicted to be in default, or less than 1/47 of the expected number. The loans that were paid off overwhelm the loans in default because the model is trained using all the data equally. Thinking about it intuitively, the presence of so many nondefaulting loans, coupled with the inevitable variability in predictor data, means that, even for a defaulting loan, the model is likely to find some nondefaulting loans that it is similar to, by chance. When a balanced sample was used, roughly 50% of the loans were predicted to be in default.

Oversampling and Up/Down Weighting

One criticism of the undersampling method is that it throws away data and does not use all the information at hand. If you have a relatively small data set, and the rarer class contains a few hundred or a few thousand records, then undersampling the dominant class has the risk of throwing out useful information. In this case, instead of downsampling the dominant case, you should oversample (upsample) the rarer class by drawing additional rows with replacement (bootstrapping).

You can achieve a similar effect by weighting the data. Many classification algorithms take a weight argument that will allow you to up/down weight the data. For example, apply a weight vector to the loan data using the `weight` argument to `glm` in R:

```
full_train_set <- full_train_set %>%
  mutate(
```

```
    weight = ifelse(outcome == "default", 1 / mean(outcome == "default"), 1.0),
    weight = importance_weights(weight))  ❶
weighted_model <- workflow() %>%  ❷
  add_model(logistic_reg(mode = "classification")) %>%
  add_case_weights(weight) %>%  ❸
  add_formula(outcome ~ payment_inc_ratio + purpose_ +
      home_ + emp_len_ + dti + revol_bal + revol_util) %>%
  fit(full_train_set)
pred <- predict(weighted_model, new_data = full_train_set)
100 * mean(pred == "default")
---
[1] 57.67208
```

❶ `Tidymodels` require weights to be included as a column in the data frame and converted to `importance_weights`.

❷ In order to use weights, we need to use a `workflow`. Workflows are likely the preferred way to build models in `tidymodels`, as they allow to combine the preprocessing and modeling steps in a single object.

❸ The `add_case_weights` function is used to specify the weights for each record. This information will only be used during model fitting and not during prediction.

Many `scikit-learn` models allow specifying weights in the `fit` function using the keyword argument `sample_weight`:

```
default_wt = 1 / np.mean(full_train_set.outcome == "default")
wt = [default_wt if outcome == "default" else 1
      for outcome in full_train_set.outcome]

full_model = LogisticRegression(C=np.inf, max_iter=10_000)
full_model.fit(X, y, sample_weight=wt)
print("percentage of loans predicted to default (weighting): ",
      100 * np.mean(full_model.predict(X) == "default"))
```

The weights for loans that default are set to $\frac{1}{p}$, where p is the probability of default. The nondefaulting loans have a weight of 1. The sums of the weights for the defaulting loans and nondefaulting loans are roughly equal. The mean of the predicted values is now about 58% instead of 0.39%.

Note that weighting provides an alternative to both upsampling the rarer class and downsampling the dominant class.

ADAPTING THE LOSS FUNCTION

Many classification and regression algorithms optimize a certain criterion or *loss function*. For example, logistic regression attempts to minimize the deviance. In the literature, some propose to modify the loss function in order to avoid the problems caused by a rare class. In practice, this is hard to do: classification algorithms can be complex and difficult to modify. Weighting is an easy way to change the loss function, discounting errors for records with low weights in favor of records with higher weights.

Data Generation

A variation of upsampling via bootstrapping (see "Oversampling and Up/Down Weighting") is *data generation* by perturbing existing records to create new records. The intuition behind this idea is that since we observe only a limited set of instances, the algorithm doesn't have a rich set of information to build classification "rules." By creating new records that are similar but not identical to existing records, the algorithm has a chance to learn a more robust set of rules. This notion is similar in spirit to ensemble statistical models such as boosting and bagging (see Chapter 6).

The idea gained traction with the publication of the *SMOTE* algorithm, which stands for "synthetic minority oversampling technique." The SMOTE algorithm finds a record that is similar to the record being upsampled (see "K-Nearest Neighbors") and creates a synthetic record that is a randomly weighted average of the original record

and the neighboring record, where the weight is generated separately for each predictor. The number of synthetic oversampled records created depends on the oversampling ratio required to bring the data set into approximate balance with respect to outcome classes.

There are several implementations of SMOTE in R. The most comprehensive package for handling unbalanced data is `unbalanced`. It offers a variety of techniques, including a "Racing" algorithm to select the best method. If you already use `tidymodels`, the `themis` package has SMOTE and various related data generation techniques. However, the SMOTE algorithm is simple enough that it can be implemented directly in R using the `FNN` package.

The Python package `imbalanced-learn` implements a variety of methods with an API that is compatible with `scikit-learn`. It provides various methods for over- and undersampling and support for using these techniques with boosting and bagging classifiers.

Cost-Based Classification

In practice, accuracy and AUC are a poor man's way to choose a classification rule. Often, an estimated cost can be assigned to false positives versus false negatives, and it is more appropriate to incorporate these costs to determine the best cutoff when classifying 1s and 0s. For example, suppose the expected cost of a default of a new loan is C and the expected return from a paid-off loan is R. Then the expected return for that loan is:

$$\text{expected return} = P(Y = 0) \times R + P(Y = 1) \times C$$

Instead of simply labeling a loan as default or paid off, or determining the probability of default, it makes more sense to determine if the loan has a positive expected return. Predicted probability of default is an intermediate step, and it must be combined with the loan's total value to determine expected profit, which is the ultimate planning metric of business. For example, a smaller-value loan might be passed over in favor of a larger one with a slightly higher predicted default probability.

Exploring the Predictions

A single metric, such as AUC, cannot evaluate all aspects of the suitability of a model for a situation. Figure 5-9 displays the decision rules for four different models fit to the loan data using just two predictor variables: `borrower_score` and `payment_inc_ratio`. The models are LDA, logistic linear regression, logistic regression fit using a GAM, and a tree model (see "Tree Models"). The region to the upper-left of the lines corresponds to a predicted default. LDA and logistic linear regression give nearly identical results in this case. The tree model produces the least regular rule, with two steps. Finally, the GAM fit of the logistic regression represents a compromise between the tree model and the linear model.

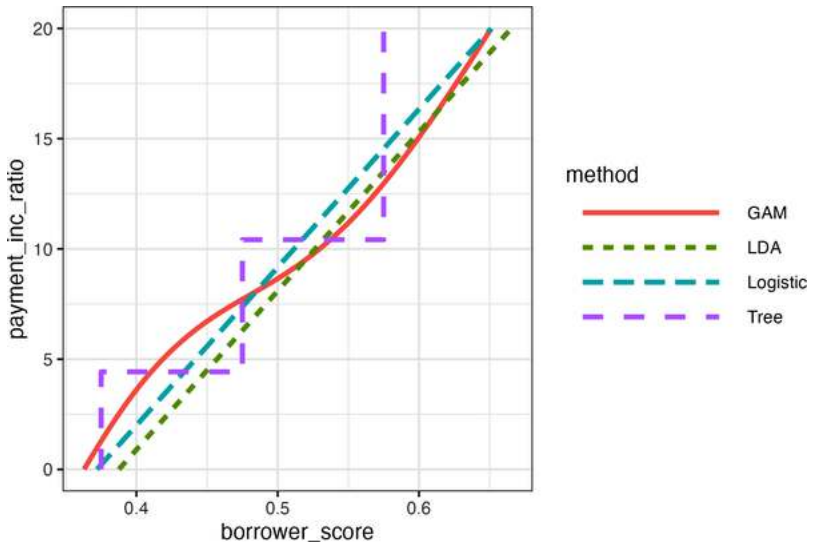


Figure 5-9. Comparison of the classification rules for four different methods

It is not easy to visualize the prediction rules in higher dimensions or, in the case of the GAM and the tree model, even to generate the regions for such rules.

In any case, exploratory analysis of predicted values is always warranted.

KEY IDEAS

- Highly imbalanced data (i.e., where the interesting outcomes, the 1s, are rare) is problematic for classification algorithms.
- One strategy for working with imbalanced data is to balance the training data via undersampling the abundant case (or oversampling the rare case).
- If using all the 1s still leaves you with too few 1s, you can bootstrap the rare cases, or use SMOTE to create synthetic data similar to existing rare cases.
- Imbalanced data usually indicates that correctly classifying one class (the 1s) has higher value, and that value ratio should be built into the assessment metric.

Further Reading

- Tom Fawcett, author of *Data Science for Business*, has a widely referenced web tutorial on imbalanced classes.
- For more on SMOTE, see Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research* 16 (2002): 321–357.
- Also see the Analytics Vidhya Content Team's "Practical Guide to Deal with Imbalanced Classification Problems in R," March 28, 2016.

Summary

Classification, the process of predicting which of two or more categories a record belongs to, is a fundamental tool of predictive analytics. Will a loan default (yes or no)? Will it prepay? Will a web visitor click on a link? Will they

purchase something? Is an insurance claim fraudulent? Often in classification problems, one class is of primary interest (e.g., the fraudulent insurance claim), and in binary classification, this class is designated as a 1, with the other, more prevalent class being a 0. Often, a key part of the process is estimating a *propensity score*, a probability of belonging to the class of interest. A common scenario is one in which the class of interest is relatively rare. When evaluating a classifier, there are a variety of model assessment metrics that go beyond simple accuracy; these are important in the rare-class situation, when classifying all records as 0s can yield high accuracy.

Exploration with AI

1. While using packages like `tidymodels` or `scikit-learn` allows building and evaluating models using just a few functions, it is valuable to understand what is going on behind the scenes to gain a full understanding. Ask your AI to create R or Python code for the following tasks:
    - Calculate accuracy and specificity given outcome and predictions as two lists/arrays of strings.
    - Create the visualization of a ROC curve given a list of strings as the outcome and the predicted probabilities; calculate accuracy and display it in the plot.
2. Ask your AI to explain considerations and strategies for modeling binary response data with a response rate of 0.001.
3. Ask your AI to explain lift and AUC as model metrics, and the advantages and disadvantages of each.
4. Ask your AI to explain data generation techniques for imbalanced data, such as SMOTE or related methods.

---

[1] This and subsequent sections in this chapter © 2020–2025 Datastats, LLC, Peter Bruce, Andrew Bruce, and Peter Gedeck; used with permission.

[2] It is certainly surprising that the first article on statistical classification was published in a journal devoted to eugenics. Indeed, there is a disconcerting connection between the early development of statistics and eugenics.

[3] Not all methods provide unbiased estimates of probability. In most cases, it is sufficient that the method provide a ranking equivalent to the rankings that would result from an unbiased probability estimate; the cutoff method is then functionally equivalent.

[4] The ROC curve was first used during World War II to describe the performance of radar receiving stations, whose job was to correctly identify (classify) reflected radar signals and alert defense forces to incoming aircraft.

Chapter 6. Statistical Machine Learning

Recent advances in statistics have been devoted to developing more powerful automated techniques for predictive modeling—both regression and classification. These methods, like those discussed in the previous chapter, are *supervised methods*—they are trained on data where outcomes are known and learn to predict outcomes in new data. They fall under the umbrella of *statistical machine learning* and are distinguished from classical statistical methods in that they are data-driven and do not seek to impose linear or other overall structure on the data. The *K*-nearest neighbors method, for example, is quite simple: classify a record in accordance with how similar records are classified. The most successful and widely used techniques are based on *ensemble learning* applied to *decision trees*. The basic idea of ensemble learning is to use many models to form a prediction, as opposed to using just a single model. Decision trees are a flexible and automatic technique to learn rules about the relationships between predictor variables and outcome variables. It turns out that the combination of ensemble learning with decision trees leads to some of the best performing off-the-shelf predictive modeling techniques.

The development of many of the techniques in statistical machine learning can be traced back to the statisticians Leo Breiman (see Figure 6-1) at the University of California at Berkeley and Jerry Friedman at Stanford University. Their work, along with that of other researchers at Berkeley and Stanford, started with the development of tree models in 1984. The subsequent development of ensemble methods of bagging and boosting in the 1990s established the foundation of statistical machine learning.



Figure 6-1. Leo Breiman, who was a professor of statistics at UC Berkeley, was at the forefront of the development of many techniques in a data scientist's toolkit today

MACHINE LEARNING VERSUS STATISTICS

In the context of predictive modeling, what is the difference between machine learning and statistics? There is not a bright line dividing the two disciplines. Machine learning tends to be focused more on developing efficient algorithms that scale to large data in order to optimize the predictive model. Statistics generally pays more attention to the probabilistic theory and underlying structure of the model. Bagging, and the random forest (see "Bagging and the Random Forest"), grew up firmly in the statistics camp. Boosting (see "Boosting"), on the other hand, has been developed in both disciplines but receives more attention on the machine learning side of the divide. Regardless of the history, the promise of boosting ensures that it will thrive as a technique in both statistics and machine learning.

K-Nearest Neighbors

The idea behind *K*-nearest neighbors (KNN) is very simple.[1] For each record to be classified or predicted:

1. Find *K* records that have similar features (i.e., similar predictor values).
2. For classification, find out what the majority class is among those similar records and assign that class to the new record.
3. For prediction (also called *KNN regression*), find the average among those similar records, and predict that average for the new record.

KEY TERMS FOR K-NEAREST NEIGHBORS

Neighbor

A record that has similar predictor values to another record

Distance metrics

Measures that sum up in a single number how far one record is from another

Standardization

Subtract the mean and divide by the standard deviation

Synonym

normalization

z-score

The value that results after standardization

K

The number of neighbors considered in the nearest neighbor calculation

KNN is one of the simpler prediction/classification techniques: there is no model to be fit (as in regression). This doesn't mean that using KNN is an automatic procedure. The prediction results depend on how the features are scaled, how similarity is measured, and how big *K* is set. Also, all predictors must be in numeric form. We will illustrate how to use the KNN method with a classification example.

A Small Example: Predicting Loan Default

Table 6-1 shows a few records of personal loan data from LendingClub. LendingClub is a leader in peer-to-peer lending in which pools of investors make personal loans to individuals. The goal of an analysis would be to predict the outcome of a new potential loan: paid off versus default.

Table 6-1. A few records and columns for LendingClub loan data

Outcome	Loan amount	Income	Purpose	Y e
Paid off	10,000	79,100	debt_consolidation	1
Paid off	9,600	48,000	moving	5
Paid off	18,800	120,036	debt_consolidation	1
Default	15,250	232,000	small_business	9
Paid off	17,050	35,000	debt_consolidation	4
Paid off	5,500	43,000	debt_consolidation	4

Consider a very simple model with just two predictor variables: `dti`, which is the ratio of debt payments (excluding mortgage) to income, and `payment_inc_ratio`, which is the ratio of the loan payment to income. Both ratios are multiplied by 100. Using a small set of 200 loans, `loan200`, with known binary outcomes (default or no-default, specified in the predictor `outcome200`), and with *K* set to 20, the KNN estimate for a new loan to be predicted, `newloan`, with `dti=22.5` and `payment_inc_ratio=9` can be calculated in R as follows:[2]

```
library(tidyverse)
library(tidymodels)
loan200 <- read_csv("loan200.csv", show_col_types = FALSE) %>%
  mutate(outcome = factor(outcome, levels = c("paid off", "default")))

train_data <- loan200 %>% dplyr::slice(-1)
newloan <- loan200 %>% dplyr::slice(1)
model <- nearest_neighbor(mode = "classification", neighbors = 20,
  weight_func = "rectangular")
knn_model <- workflow() %>%
  add_model(model) %>%
  add_formula(outcome ~ payment_inc_ratio + dti) %>%
  fit(train_data)

knn_pred <- knn_model %>% predict(newloan)
knn_pred
---
# A tibble: 1 × 1
  .pred_class
  <fct>
1 paid off
```

The KNN prediction is for the loan to be paid off.

While R has a native `knn` function, the contributed R package `FNN`, for Fast Nearest Neighbor, scales more effectively to big data and provides more flexibility. `Tidymodels` uses the `kknn` package. In the `kknn` package, the prediction is a weighted vote of the *K*-nearest neighbors, where closer neighbors influence the prediction more.

The `scikit-learn` package provides a fast and efficient implementation of KNN in Python:

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

loan200 = pd.read_csv("loan200.csv")

predictors = ["payment_inc_ratio", "dti"]
outcome = "outcome"

newloan = loan200.loc[0:0, predictors]
X = loan200.loc[1:, predictors]
y = loan200.loc[1:, outcome]
```

```
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(X, y)
knn.predict(newloan)
```

Figure 6-2 gives a visual display of this example. The new loan to be predicted is the cross in the middle. The squares (paid off) and circles (default) are the training data. The large black circle shows the boundary of the nearest 20 points. In this case, 9 defaulted loans lie within the circle, as compared with 11 paid-off loans. Hence the predicted outcome of the loan is paid off. Note that if we consider only three nearest neighbors, the prediction would be that the loan defaults.

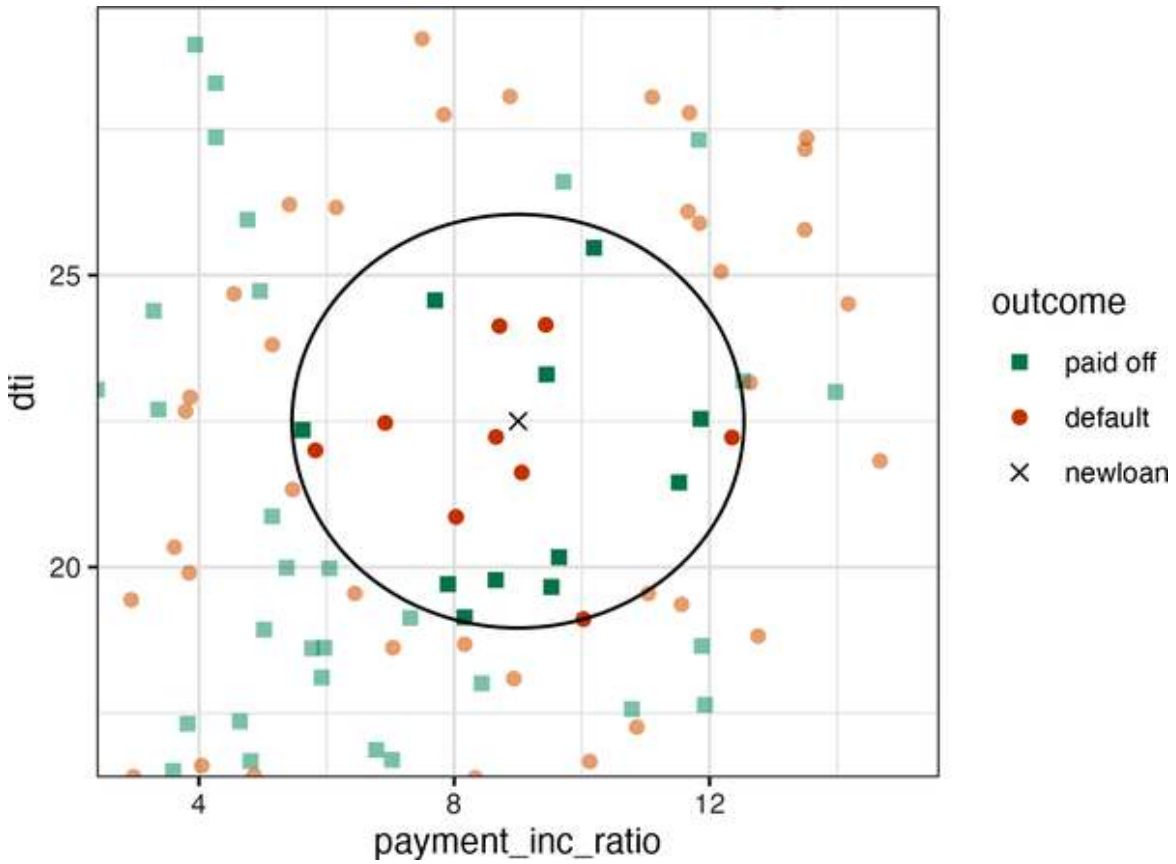


Figure 6-2. KNN prediction of loan default using two variables: debt-to-income ratio and loan-payment-to-income ratio

KNN CAN GIVE YOU A PROBABILITY

While the output of KNN for classification is typically a binary decision, such as default or paid off in the loan data, KNN routines usually offer the opportunity to output a probability (propensity) between 0 and 1. The probability is based on the fraction of one class in the *K* nearest neighbors. In the preceding example, this probability of default would have been estimated at $\frac{9}{20}$, or 0.45. Using a probability score lets you use classification rules other than simple majority votes (probability of 0.5). This is especially important in problems with imbalanced classes; see "Strategies for Imbalanced Data". For example, if the goal is to identify members of a rare class, the cutoff would typically be set below 50%. One common approach is to set the cutoff at the probability of the rare event.

Distance Metrics

Similarity (nearness) is determined using a *distance metric*, which is a function that measures how far two records (x_1, x_2,…, x_p) and (u_1, u_2,…, u_p) are from one another. The most popular distance metric between two vectors is *Euclidean distance*. To measure the Euclidean distance between two vectors, subtract one from the other, square the differences, sum them, and take the square root:

$$\sqrt{(x_1 - u_1)^2 + (x_2 - u_2)^2 + \cdots + (x_p - u_p)^2}.$$

Another common distance metric for numeric data is *Manhattan distance*:

$$|x_1 - u_1| + |x_2 - u_2| + \cdots + |x_p - u_p|$$

Euclidean distance corresponds to the straight-line distance between two points (e.g., as the crow flies). Manhattan distance is the distance between two points traversed in a single direction at a time (e.g., traveling along rectangular city blocks). For this reason, Manhattan distance is a useful approximation if similarity is defined as point-to-point travel time.

In measuring distance between two vectors, variables (features) that are measured with comparatively large scale will dominate the measure. For example, for the loan data, the distance would be almost solely a function of

the income and loan amount variables, which are measured in tens or hundreds of thousands. Ratio variables would count for practically nothing in comparison. We address this problem by standardizing the data; see "Standardization (Normalization, z-Scores)".

OTHER DISTANCE METRICS

There are numerous other metrics for measuring distance between vectors. For numeric data, *Mahalanobis distance* is attractive since it accounts for the correlation between two variables. This is useful since if two variables are highly correlated, Mahalanobis will essentially treat them as a single variable in terms of distance. Euclidean and Manhattan distances do not account for the correlation, effectively placing greater weight on the attribute that underlies those features. Mahalanobis distance is the Euclidean distance between the principal components (see "Principal Components Analysis"). The downside of using Mahalanobis distance is increased computational effort and complexity; it is computed using the *covariance matrix* (see "Covariance Matrix").

One-Hot Encoder

The loan data in Table 6-1 includes several factor (string) variables. Most statistical and machine learning models require this type of variable to be converted to a series of binary dummy variables conveying the same information, as in Table 6-2. Instead of a single variable denoting the home occupant status as "owns with a mortgage," "owns with no mortgage," "rents," or "other," we end up with four binary variables. The first would be "owns with a mortgage—Y/N," the second would be "owns with no mortgage—Y/N," and so on. This one predictor, home occupant status, thus yields a vector with one 1 and three 0s that can be used in statistical and machine learning algorithms. The phrase *one-hot encoding* comes from digital circuit terminology, where it describes circuit settings in which only one bit is allowed to be positive (hot).

Table 6-2. Representing home ownership factor data in Table 6-1 as a numer

OWNS_WITH_MORTGAGE	OWNS_WITHOUT_MORTGAGE	C
1	0	0
1	0	0
1	0	0
1	0	0
0	0	0
0	0	0

> **ONE-HOT ENCODING AND MULTICOLLINEARITY**
>
> In linear and logistic regression, one-hot encoding causes problems with multicollinearity; see "Multicollinearity". In such cases, one dummy is omitted (its value can be inferred from the other values). This is not an issue with KNN and other methods discussed in this book.

Standardization (Normalization, z-Scores)

In measurement, we are often not so much interested in "how much" but in "how different from the average." Standardization, also called *normalization*, puts all variables on similar scales by subtracting the mean and dividing by the standard deviation; in this way, we ensure that a variable does not overly influence a model simply due to the scale of its original measurement:

$$z = \frac{x - x}{s}$$

The result of this transformation is commonly referred to as a *z-score*. Measurements are then stated in terms of "standard deviations away from the mean."

"NORMALIZATION" IN STATISTICS

Normalization in this statistical context is not to be confused with *database normalization*, which is the removal of redundant data and the verification of data dependencies.

For KNN and a few other procedures (e.g., principal components analysis and clustering), it is essential to consider standardizing the data prior to applying the procedure. To illustrate this idea, KNN is applied to the loan data using `dti` and `payment_inc_ratio` (see "A Small Example: Predicting Loan Default") plus two other variables: `revol_bal`, the total revolving credit available to the applicant in dollars, and `revol_util`, the percent of the credit being used. The new record to be predicted is shown here:

```
newloan
---
  payment_inc_ratio dti revol_bal revol_util
1            2.3932   1      1687        9.4
```

The magnitude of `revol_bal`, which is in dollars, is much bigger than that of the other variables. The `knn` function returns the index of the nearest neighbors as an attribute `nn.index`, and this can be used to show the top-five closest rows in `loan_df`:

```
library(FNN)
loan_data <- read_csv("loan_data.csv.gz", show_col_types = FALSE) %>%
  select(-c(index, status)) %>%
  mutate(
```

```
    across(where(is.character), as.factor),
    outcome = factor(outcome, levels = c("paid off", "default"))
  )

loan_df <- model.matrix(~ -1 + payment_inc_ratio + dti + revol_bal +
    revol_util, data = loan_data)
newloan <- loan_df[1, , drop = FALSE]
loan_df <- loan_df[-1, ]
outcome <- loan_data[-1, ]$outcome
knn_pred <- knn(train = loan_df, test = newloan, cl = outcome, k = 5)
loan_df[attr(knn_pred, "nn.index"), ]
---
        payment_inc_ratio  dti revol_bal revol_util
35537             1.47212 1.46      1686       10.0
33652             3.38178 6.37      1688        8.4
25864             2.36303 1.39      1691        3.5
42954             1.28160 7.14      1684        3.9
43600             4.12244 8.98      1684        7.2
```

Following the model fit, we can use the `kneighbors` method to identify the five closest rows in the training set with `scikit-learn`:

```
loan_data = pd.read_csv("loan_data.csv.gz")

predictors = ["payment_inc_ratio", "dti", "revol_bal", "revol_util"]
outcome = "outcome"

newloan = loan_data.loc[0:0, predictors]
X = loan_data.loc[1:, predictors]
y = loan_data.loc[1:, outcome]

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)

nbrs = knn.kneighbors(newloan)
X.iloc[nbrs[1][0], :]
```

The value of `revol_bal` in these neighbors is very close to its value in the new record, but the other predictor variables are all over the map and essentially play no role in determining neighbors.

Compare this to KNN applied to the standardized data using the R function `scale`, which computes the *z*-score for each variable:

```
loan_df <- model.matrix(~ -1 + payment_inc_ratio + dti + revol_bal +
    revol_util, data = loan_data)
loan_std <- scale(loan_df)
newloan_std <- loan_std[1, , drop = FALSE]
loan_std <- loan_std[-1, ]
loan_df <- loan_df[-1, ]  ❶
outcome <- loan_data[-1, ]$outcome
knn_pred <- knn(train = loan_std, test = newloan_std, cl = outcome, k = 5)
loan_df[attr(knn_pred, "nn.index"), ]
---
       payment_inc_ratio   dti  revol_bal  revol_util
2081            2.61091  1.03       1218         9.7
1439            2.34343  0.51        278         9.9
30216           2.71200  1.34       1075         8.5
28543           2.39760  0.74       2917         7.4
44738           2.34309  1.37        488         7.2
```

❶ We need to remove the first row from `loan_df` as well, so that the row numbers correspond to each other.

The `sklearn.preprocessing.StandardScaler` method is first trained with the predictors and is subsequently used to transform the data set prior to training the KNN model:

```
from sklearn import preprocessing

newloan = loan_data.loc[0:0, predictors]
X = loan_data.loc[1:, predictors]
y = loan_data.loc[1:, outcome]

scaler = preprocessing.StandardScaler()
scaler.fit(X * 1.0)

X_std = scaler.transform(X * 1.0)
newloan_std = scaler.transform(newloan * 1.0)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_std, y)

nbrs = knn.kneighbors(newloan_std)
X.iloc[nbrs[1][0], :]
```

The five nearest neighbors are much more alike in all the variables, providing a more sensible result. Note that the results are displayed on the original scale, but KNN was applied to the scaled data and the new loan to be predicted.

SCALING OPTIONS

Using the *z*-score is just one way to rescale variables. Instead of the mean, a more robust estimate of location could be used, such as the median. Likewise, a different estimate of scale such as the interquartile range could be used instead of the standard deviation. Sometimes, variables are "squashed" into the 0–1 range. It's also important to realize that scaling each variable to have unit variance is somewhat arbitrary. This implies that each variable is thought to have the same importance in predictive power. If you have subjective knowledge that some variables are more important than others, then these could be scaled up. For example, with the loan data, it is reasonable to expect that the payment-to-income ratio is very important.

NORMALIZATION DOES NOT MAKE DATA NORMAL

Normalization (standardization) does not change the distributional shape of the data; it does not make it normally shaped if it was not already normally shaped (see "Normal Distribution").

Choosing K

The choice of K is very important to the performance of KNN. The simplest choice is to set $K = 1$, known as the 1-nearest neighbor classifier. The prediction is intuitive: it is based on finding the data record in the training set most similar to the new record to be predicted. Setting $K = 1$ is rarely the best choice; you'll almost always obtain superior performance by using $K >$ 1-nearest neighbors.

Generally speaking, if K is too low, we may be overfitting: including the noise in the data. Higher values of K provide smoothing that reduces the risk of overfitting in the training data. On the other hand, if K is too high,

we may oversmooth the data and miss out on KNN's ability to capture the local structure in the data, one of its main advantages.

The *K* that best balances between overfitting and oversmoothing is typically determined by accuracy metrics and, in particular, accuracy with holdout or validation data. There is no general rule about the best *K*—it depends greatly on the nature of the data. For highly structured data with little noise, smaller values of *K* work best. Borrowing a term from the signal processing community, this type of data is sometimes referred to as having a high *signal-to-noise ratio* (*SNR*). Examples of data with a typically high SNR are data sets for handwriting and speech recognition. For noisy data with less structure (data with a low SNR), such as the loan data, larger values of *K* are appropriate. Typically, values of *K* fall in the range 1 to 20 for KNN implementations that treat all neighbors equally. Often, an odd number is chosen to avoid ties. For weighted KNN, larger values often lead to better models.

BIAS-VARIANCE TRADE-OFF

The tension between oversmoothing and overfitting is an instance of the *bias-variance trade-off*, a ubiquitous problem in statistical model fitting. Variance refers to the modeling error that occurs because of the choice of training data; that is, if you were to choose a different set of training data, the resulting model would be different. Bias refers to the modeling error that occurs because you have not properly identified the underlying real-world scenario; this error would not disappear if you simply added more training data. When a flexible model is overfit, the variance increases. You can reduce this by using a simpler model, but the bias may increase due to the loss of flexibility in modeling the real underlying situation. A general approach to handling this trade-off is through *cross-validation*. See "Cross-Validation" for more details.

KNN as a Feature Engine

KNN gained its popularity due to its simplicity and intuitive nature. In terms of performance, KNN by itself is usually not competitive with more sophisticated classification techniques. In practical model fitting, however, KNN can be used to add "local knowledge" in a staged process with other classification techniques:

1. KNN is run on the data, and for each record, a classification (or quasi-probability of a class) is derived.
2. That result is added as a new feature to the record, and another classification method is then run on the data. The original predictor variables are thus used twice.

At first you might wonder whether this process, since it uses some predictors twice, causes a problem with multicollinearity (see "Multicollinearity"). This is not an issue, since the information being incorporated into the second-stage model is highly local, derived only from a few nearby records, and is therefore additional information and not redundant.

STAGED KNN

You can think of this staged use of KNN as a form of ensemble learning, in which multiple predictive modeling methods are used in conjunction with one another. It can also be considered as a form of feature engineering in which the aim is to derive features (predictor variables) that have predictive power. Often this involves some manual review of the data; KNN gives a fairly automatic way to do this.

For example, consider the King County housing data. In pricing a home for sale, a realtor will base the price on similar homes recently sold, known as "comps." In essence, realtors are doing a manual version of KNN: by looking at the sale prices of similar homes, they can estimate what a home will sell for. We can create a new feature for a statistical model to mimic the real estate professional by applying KNN to recent sales. The predicted value is the sales price, and the existing predictor variables could include location, total square feet, type of structure, lot size, and number of bedrooms and bathrooms. The new predictor variable (feature) that we add via KNN is the KNN predictor for each record (analogous to the realtors' comps). Since we are predicting a numerical value, the average of the *K*-nearest neighbors is used instead of a majority vote (known as *KNN regression*).

Similarly, for the loan data, we can create features that represent different aspects of the loan process. For example, the following R code would build a feature that represents a borrower's creditworthiness:

```
rec <- recipe(outcome ~ dti + revol_bal + revol_util + open_acc +
    delinq_2yrs_zero + pub_rec_zero, data = loan_data)
model <- nearest_neighbor(mode = "classification", neighbors = 20,
  weight_func = "rectangular")
borrow_knn <- workflow() %>%
  add_model(nearest_neighbor(mode = "classification", neighbors = 20)) %>%
  add_recipe(rec) %>%
  fit(loan_data)
borrow_feature <- borrow_knn %>%
  predict(loan_data, type = "prob") %>%
  pluck(".pred_default")
summary(borrow_feature)
---
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.400   0.500   0.501   0.600   0.950
```

With `scikit-learn`, we use the `predict_proba` method of the trained model to get the probabilities:

```
predictors = ["dti", "revol_bal", "revol_util", "open_acc",
              "delinq_2yrs_zero", "pub_rec_zero"]
outcome = "outcome"

X = loan_data[predictors]
y = loan_data[outcome]

knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(X, y)

loan_data["borrower_score"] = knn.predict_proba(X)[:, 1]
loan_data["borrower_score"].describe()
```

The result is a feature that predicts the likelihood a borrower will default based on their credit history.

KEY IDEAS

- *K*-nearest neighbors (KNN) classifies a record by assigning it to the class that similar records belong to.
- Similarity (distance) is determined by Euclidean distance or other related metrics.
- The number of nearest neighbors to compare a record to, *K*, is determined by how well the algorithm performs on training data, using different values for *K*.
- Typically, the predictor variables are standardized so that variables of large scale do not dominate the distance metric.
- KNN is often used as a first stage in predictive modeling, and the predicted value is added back into the data as a *predictor* for second-stage (non-KNN) modeling.

Tree Models

Tree models, also called *Classification and Regression Trees* (*CART*),[3] *decision trees*, or just *trees*, are an effective and popular classification (and regression) method initially developed by Leo Breiman and others in 1984. Tree models, and their more powerful descendants, *random forests* and *boosted trees* (see "Bagging and the Random Forest" and "Boosting"), form the basis for the most widely used and powerful predictive modeling tools in data science for regression and classification.

KEY TERMS FOR TREES

Recursive partitioning

Repeatedly dividing and subdividing the data with the goal of making the outcomes in each final subdivision as homogeneous as possible

Split value

A predictor value that divides the records into those where that predictor is less than the split value, and those where it is more

Node

In the decision tree, or in the set of corresponding branching rules, a node is the graphical or rule representation of a split value

Leaf

The end of a set of if-then rules, or branches of a tree—the rules that bring you to that leaf provide one of the classification rules for any record in a tree

Loss

The number of misclassifications at a stage in the splitting process; the more losses, the more impurity

Impurity

The extent to which a mix of classes is found in a subpartition of the data (the more mixed, the more impure)

Synonym

heterogeneity

Antonyms

homogeneity, purity

Pruning

The process of taking a fully grown tree and progressively cutting its branches back to reduce overfitting

A tree model is a set of "if-then-else" rules that are easy to understand and to implement. In contrast to linear and logistic regression, trees have the ability to discover hidden patterns corresponding to complex interactions in the data. However, unlike KNN or naive Bayes, simple tree models can be expressed in terms of predictor relationships that are easily interpretable.

DECISION TREES IN OPERATIONS RESEARCH

The term *decision trees* has a different (and older) meaning in decision science and operations research, where it refers to a human decision analysis process. In this meaning, decision points, possible outcomes, and their estimated probabilities are laid out in a branching diagram, and the decision path with the maximum expected value is chosen.

A Simple Example

The two main packages to fit tree models in R are `rpart` and `tree`. Using the `rpart` package, a model is fit to a sample of 3,000 records of the loan data using the variables `payment_inc_ratio` and `borrower_score` (see "K-Nearest Neighbors" for a description of the data):

```
library(rpart.plot)  ❶

loan3000 <- read_csv("loan3000.csv", show_col_types = FALSE) %>%
  mutate(outcome = factor(outcome, levels = c("paid off", "default")))

loan_tree <- decision_tree(mode = "classification", engine = "rpart") %>%
  fit(outcome ~ borrower_score + payment_inc_ratio, data = loan3000)
```

```
rpart.plot(loan_tree %>% extract_fit_engine(), roundint = FALSE)  ❷
```

❶ The `rpart.plot` package provides convenient plotting functions for `rpart` models.

❷ We need to access the underlying `rpart` model using `extract_fit_engine()` to use with the `rpart.plot` function.

The `sklearn.tree.DecisionTreeClassifier` provides an implementation of a decision tree. The `mlba` package provides a convenience function to create a visualization inside a Jupyter notebook:

```
from mlba import plotDecisionTree
from sklearn.tree import DecisionTreeClassifier

loan3000 = pd.read_csv("loan3000.csv")

predictors = ["borrower_score", "payment_inc_ratio"]
outcome = "outcome"

X = loan3000[predictors]
y = loan3000[outcome]

loan_tree = DecisionTreeClassifier(random_state=1, criterion="entropy",
                                   min_impurity_decrease=0.003)
loan_tree.fit(X, y)
plotDecisionTree(loan_tree, feature_names=predictors,
                 class_names=loan_tree.classes_)
```

The resulting tree is shown in Figure 6-3. Due to the different implementations, you will find that the results from R and Python are not identical; this is expected. These classification rules are determined by traversing through a hierarchical tree, starting at the root and moving left if the node is true and right if not, until a leaf is reached.

Typically, the tree is plotted upside-down, so the root is at the top and the leaves are at the bottom. For example, if we get a loan with `borrower_score` of 0.6 and a `payment_inc_ratio` of 8.0, we end up at the leftmost leaf and predict the loan will be paid off.

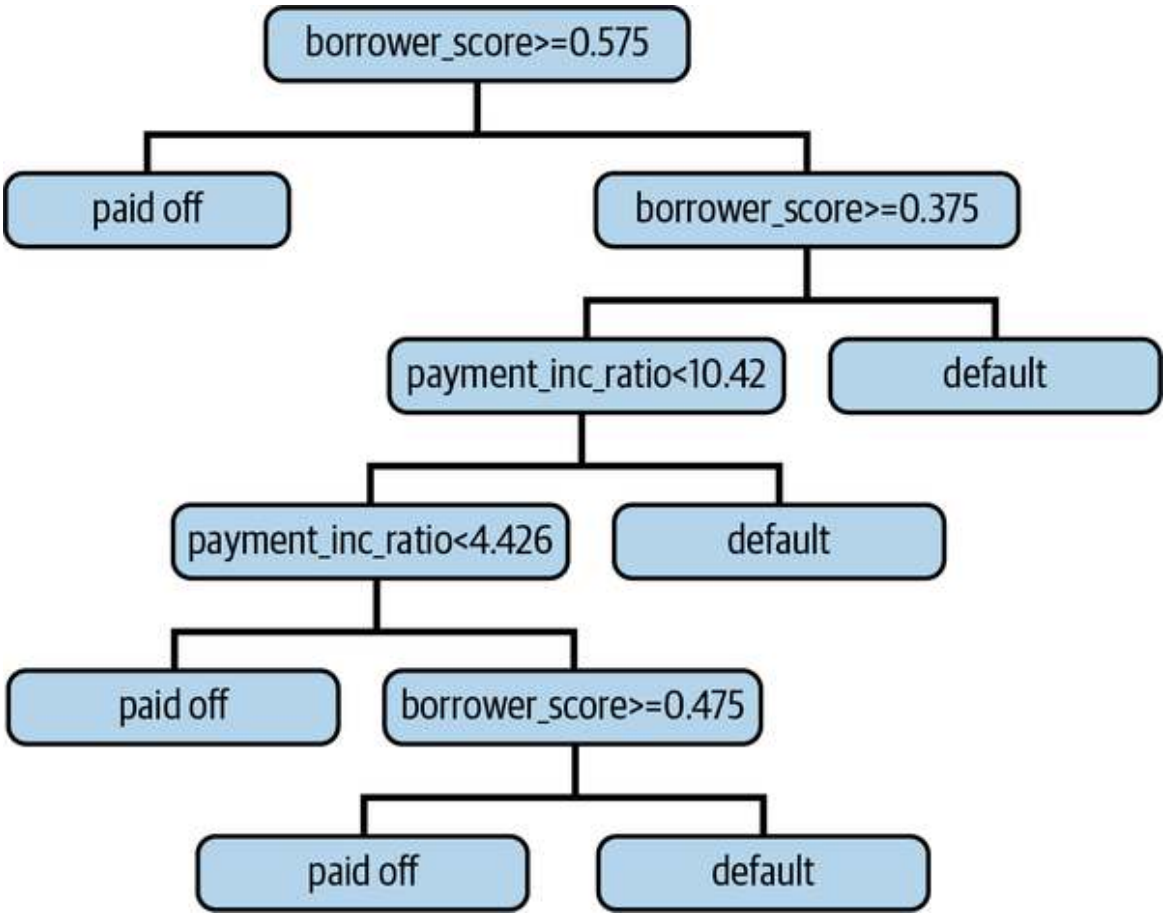


Figure 6-3. The rules for a simple tree model fit to the loan data

A nicely printed version of the tree is also easily produced in R:

```
loan_tree
---
n= 3000

node), split, n, loss, yval, (yprob)
    * denotes terminal node

1) root 3000 1445 paid off (0.5183333 0.4816667)
  2) borrower_score>=0.575 878  261 paid off (0.7027335 0.2972665) *
  3) borrower_score< 0.575 2122  938 default (0.4420358 0.5579642)
    6) borrower_score>=0.375 1639  802 default (0.4893228 0.5106772)
      12) payment_inc_ratio< 10.42265 1157  547 paid off (0.5272256 0.4727744)
        24) payment_inc_ratio< 4.42601 334  139 paid off (0.5838323 0.4161677)
*
        25) payment_inc_ratio>=4.42601 823  408 paid off (0.5042527 0.4957473)
          50) borrower_score>=0.475 418  190 paid off (0.5454545 0.4545455) *
          51) borrower_score< 0.475 405  187 default (0.4617284 0.5382716) *
```

```
      13) payment_inc_ratio>=10.42265 482  192 default (0.3983402 0.6016598) *
    7) borrower_score< 0.375 483  136 default (0.2815735 0.7184265) *
```

The depth of the tree is shown by the indent. Each node corresponds to a provisional classification determined by the prevalent outcome in that partition. The "loss" is the number of misclassifications yielded by the provisional classification in a partition. For example, in node 2, there were 261 misclassifications out of a total of 878 records. The values in the parentheses correspond to the proportion of records that are paid off or in default, respectively. For example, in node 13, which predicts default, over 60% of the records are loans that are in default.

The `scikit-learn` documentation describes how to create a text representation of a decision tree model. We included a convenience function in our `mlba` package:

```
from mlba import textDecisionTree

print(textDecisionTree(loan_tree))
---
node=0 test node: go to node 1 if 0 <= 0.5750000178813934 else to node 6
  node=1 test node: go to node 2 if 0 <= 0.32500000298023224 else to node 3
    node=2 leaf node: [[0.785, 0.215]]
    node=3 test node: go to node 4 if 1 <= 10.42264986038208 else to node 5
      node=4 leaf node: [[0.488, 0.512]]
      node=5 leaf node: [[0.613, 0.387]]
  node=6 test node: go to node 7 if 1 <= 9.19082498550415 else to node 10
    node=7 test node: go to node 8 if 0 <= 0.7249999940395355 else to node 9
      node=8 leaf node: [[0.247, 0.753]]
      node=9 leaf node: [[0.073, 0.927]]
    node=10 leaf node: [[0.457, 0.543]]
```

The Recursive Partitioning Algorithm

The algorithm to construct a decision tree, called *recursive partitioning*, is straightforward and intuitive. The data is repeatedly partitioned using predictor values that do the best job of separating the data into relatively homogeneous partitions. Figure 6-4 shows the partitions created for the tree in Figure 6-3. The first rule, depicted by rule 1, is `borrower_score >=`

`0.575` and segments the plot into right and left portions. The second rule is `borrower_score < 0.375` and further segments the left portion.

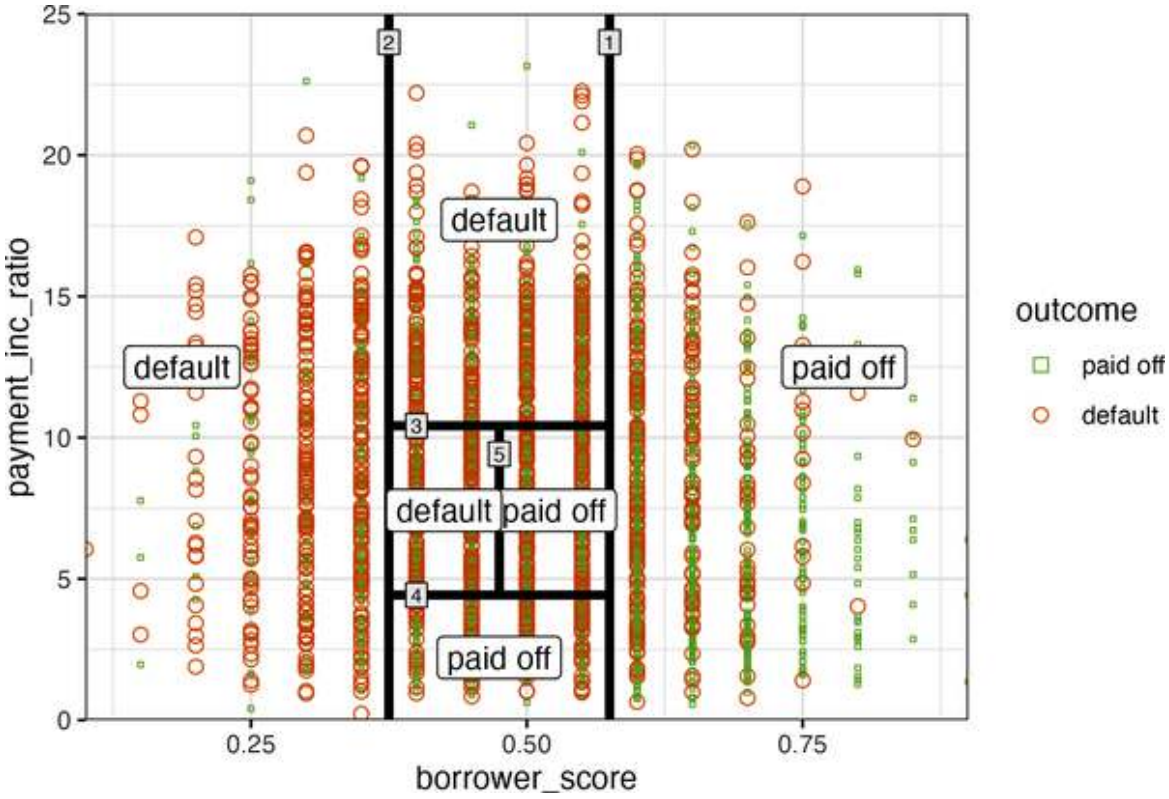


Figure 6-4. The first five rules for a simple tree model fit to the loan data

Suppose we have a response variable Y and a set of P predictor variables X_j for $j = 1, \cdots, P$. For a partition A of records, recursive partitioning will find the best way to partition A into two subpartitions:

1. For each predictor variable X_j:
    a. For each value s_j of X_j:
        i. Split the records in A with X_j values $< s_j$ as one partition, and the remaining records where $X_j \geq s_j$ as another partition.
        ii. Measure the homogeneity of classes within each subpartition of A.

b. Select the value of s_j that produces maximum within-partition homogeneity of class.

2. Select the variable X_j and the split value s_j that produces maximum within-partition homogeneity of class.

Now comes the recursive part:

1. Initialize A with the entire data set.
2. Apply the partitioning algorithm to split A into two subpartitions, A_1 and A_2.
3. Repeat step 2 on subpartitions A_1 and A_2.
4. The algorithm terminates when no further partition can be made that sufficiently improves the homogeneity of the partitions.

The end result is a partitioning of the data, as in Figure 6-4, except in P-dimensions, with each partition predicting an outcome of 0 or 1 depending on the majority vote of the response in that partition.

STAGED KNN AS ENSEMBLE LEARNING

In addition to a binary 0/1 prediction, tree models can produce a probability estimate based on the number of 0s and 1s in the partition. The estimate is simply the sum of 0s or 1s in the partition divided by the number of observations in the partition:

$$\text{Prob}\,(Y = 1) = \frac{\text{Number of 1s in the partition}}{\text{Size of the partition}}$$

The estimated $\text{Prob}(Y = 1)$ can then be converted to a binary decision; for example, set the estimate to 1 if $\text{Prob}(Y = 1) > 0.5$.

Measuring Homogeneity or Impurity

Tree models recursively create partitions (sets of records), A, that predict an outcome of $Y = 0$ or $Y = 1$. You can see from the preceding algorithm that we need a way to measure homogeneity, also called *class purity*, within a

partition. Or equivalently, we need to measure the impurity of a partition. The accuracy of the predictions is the proportion p of misclassified records within that partition, which ranges from 0 (perfect) to 0.5 (purely random guessing).

It turns out that accuracy is not a good measure for impurity. Instead, two common measures for impurity are the *Gini impurity* and *entropy* of *information*. While these (and other) impurity measures apply to classification problems with more than two classes, we focus on the binary case. The Gini impurity for a set of records A is:

$$I(A) = p(1-p)$$

The entropy measure is given by:

$$I\left(A\right) = -p\log_2\left(p\right) - \left(1-p\right)\log_2\left(1-p\right)$$

Figure 6-5 shows that Gini impurity (rescaled) and entropy measures are similar, with entropy giving higher impurity scores for moderate and high accuracy rates.

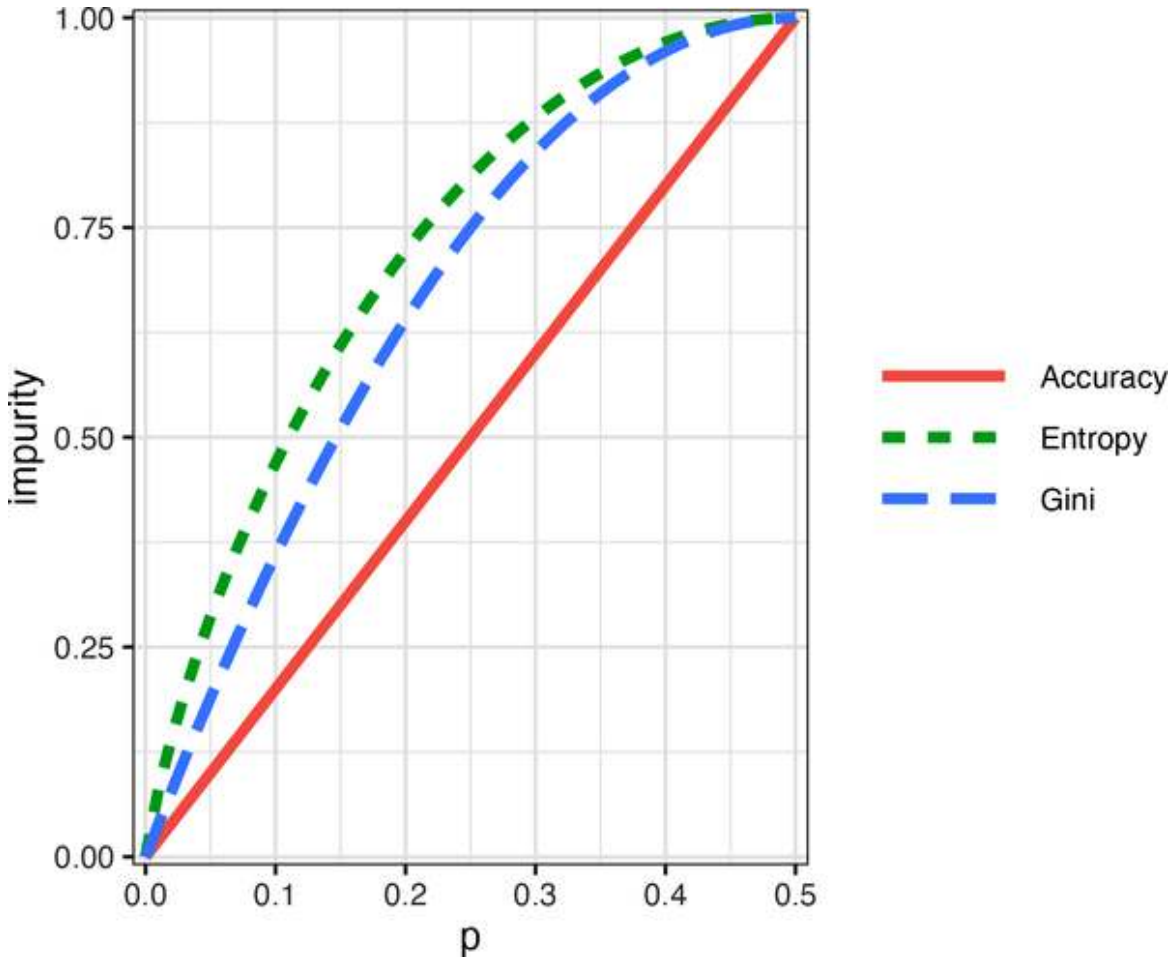


Figure 6-5. Gini impurity and entropy measures

GINI COEFFICIENT

Gini impurity is not to be confused with the *Gini coefficient*. They represent similar concepts, but the Gini coefficient is limited to the binary classification problem and is related to the AUC metric (see "AUC").

The impurity metric is used in the splitting algorithm described earlier. For each proposed partition of the data, impurity is measured for each of the partitions that result from the split. A weighted average is then calculated, and whichever partition (at each stage) yields the lowest weighted average is selected.

Stopping the Tree from Growing

As the tree grows bigger, the splitting rules become more detailed, and the tree gradually shifts from identifying "big" rules that identify real and reliable relationships in the data to "tiny" rules that reflect only noise. A fully grown tree results in completely pure leaves and, hence, 100% accuracy in classifying the data that it is trained on. This accuracy is, of course, illusory—we have overfit (see "Bias-Variance Trade-off") the data, fitting the noise in the training data, not the signal that we want to identify in new data.

We need some way to determine when to stop growing a tree at a stage that will generalize to new data. There are various ways to stop splitting in R and Python:

- Avoid splitting a partition if a resulting subpartition is too small, or if a terminal leaf is too small. In R's `tidymodels`, you can control the first constraint using the parameter `min_n`, with default of `2`. In Python's `DecisionTreeClassifier`, we can control this using the parameters `min_samples_split` (default `2`) and `min_samples_leaf` (default `1`).
- Only grow the tree to a certain depth. In `tidymodels`, you can control this using the parameter `tree_depth`, with default of `30`. In `DecisionTreeClassifier` (Python), you can control this using the parameter `max_depth`, with default of `None` (i.e., unlimited depth).
- Don't split a partition if the new partition does not "significantly" reduce the impurity. In `tidymodels`, this is controlled by a *complexity parameter*, which is a measure of how complex a tree is—the more complex, the greater the value of the complexity parameter. In practice, `tidymodels` uses the `cost_complexity` parameter that limits tree growth by attaching a penalty to additional complexity (splits) in a tree. Higher values of the `cost_complexity` parameter lead to simpler trees. The `DecisionTreeClassifier` (Python) has the parameter

`ccp_alpha`, which works in the same way. Setting the parameter to 0 means no pruning.

These methods involve arbitrary rules and can be useful for exploratory work, but we can't easily determine optimum values (i.e., values that maximize predictive accuracy with new data). We need to combine cross-validation with either systematically changing the model parameters or modifying the tree through pruning.

Controlling tree complexity in R

With the cost complexity parameter, `cp` in `rpart` or `cost_complexity` in `tidymodels`, we can estimate what size tree will perform best with new data. If the cost complexity parameter is too small, then the tree will overfit the data, fitting noise and not signal. On the other hand, if it is too large, then the tree will be too small and have little predictive power. The default in `rpart` is 0.01, although for larger data sets, you are likely to find this is too large. In exploratory analysis, it is sufficient to simply try a few values.

Determining the optimum `cp` is an instance of the bias-variance trade-off. The most common way to estimate a good value of `cp` is via cross-validation (see "Cross-Validation"):

1. Partition the data into training and validation (holdout) sets.
2. Grow the tree with the training data.
3. Prune it successively, step by step, recording `cp` (using the *training* data) at each step.
4. Note the `cp` that corresponds to the minimum error (loss) on the *validation* data.
5. Repartition the data into training and validation, and repeat the growing, pruning, and `cp` recording process.
6. Do this again and again, and average the `cp`s that reflect minimum error for each tree.

7. Go back to the original data, or future data, and grow a tree, stopping at this optimum `cp` value.

In `rpart`, you can use the argument `cptable` to produce a table of the `cp` values and their associated cross-validation error (`xerror` in R), from which you can determine the `cp` value that has the lowest cross-validation error.

More generally, you can use the `tidymodels` packages `tune` and `rsample` to optimize hyperparameters (see "Hyperparameters") using various resampling methods including cross-validation (`vfold_cv`) and bootstrapping (`bootstraps`).

Controlling tree complexity in Python

In the `scikit-learn`'s decision tree implementation, the complexity parameter is called `ccp_alpha`. The default value is 0, which means that the tree is not pruned; increasing the value leads to smaller trees. You can use `GridSearchCV` to find an optimal value.

There are a number of other model parameters that allow controlling the tree size. For example, we can vary `max_depth` in the range 5 to 30 and `min_samples_split` between 20 and 100. The `GridSearchCV` method in `scikit-learn` is a convenient way to combine the exhaustive search through all combinations with cross-validation. An optimal parameter set is then selected using the cross-validated model performance.

Predicting a Continuous Value

Predicting a continuous value (also termed *regression*) with a tree follows the same logic and procedure, except that impurity is measured by squared deviations from the mean (squared errors) in each subpartition, and predictive performance is judged by the square root of the mean squared error (RMSE) (see "Assessing the Model") in each partition.

`scikit-learn` has the `sklearn.tree.DecisionTreeRegressor` method to train a decision tree regression model.

How Trees Are Used

One of the big obstacles faced by predictive modelers in organizations is the perceived "black box" nature of the methods they use, which gives rise to opposition from other elements of the organization. In this regard, the tree model has two appealing aspects:

- Tree models provide a visual tool for exploring the data, to gain an idea of what variables are important and how they relate to one another. Trees can capture nonlinear relationships among predictor variables.
- Tree models provide a set of rules that can be effectively communicated to nonspecialists, either for implementation or to "sell" a data mining project.

When it comes to prediction, however, harnessing the results from multiple trees is typically more powerful than using just a single tree. In particular, the random forest and boosted tree algorithms almost always provide superior predictive accuracy and performance (see "Bagging and the Random Forest" and "Boosting"), but the aforementioned advantages of a single tree are lost.

KEY IDEAS

- Decision trees produce a set of rules to classify or predict an outcome.
- The rules correspond to successive partitioning of the data into subpartitions.
- Each partition, or split, references a specific value of a predictor variable and divides the data into records where that predictor value is above or below that split value.
- At each stage, the tree algorithm chooses the split that minimizes the outcome impurity within each subpartition.
- When no further splits can be made, the tree is fully grown and each terminal node, or leaf, has records of a single class; new cases following that rule (split) path would be assigned that class.
- A fully grown tree overfits the data and must be pruned back so that it captures signal and not noise.
- Multiple-tree algorithms like random forests and boosted trees yield better predictive performance, but they lose the rule-based communicative power of single trees.

Further Reading

- Analytics Vidhya Content Team, "Tree Based Algorithms: A Complete Tutorial from Scratch (in R & Python)", April 12, 2016.
- Terry M. Therneau, Elizabeth J. Atkinson, and the Mayo Foundation, "An Introduction to Recursive Partitioning Using the RPART Routines", March 26, 2026.

Bagging and the Random Forest

In 1906, the statistician Sir Francis Galton was visiting a county fair in England, at which a contest was being held to guess the dressed weight of an ox that was on exhibit. There were 800 guesses, and while the individual guesses varied widely, both the mean and the median came out within 1% of the ox's true weight. James Surowiecki has explored this phenomenon in his book *The Wisdom of Crowds* (Doubleday, 2004). This principle applies to predictive models as well: averaging (or taking majority votes) of multiple models—an *ensemble* of models—turns out to be more accurate than just selecting one model.

KEY TERMS FOR BAGGING AND THE RANDOM FOREST

Ensemble

Forming a prediction by using a collection of models

Synonym

model averaging

Bagging

A general technique to form a collection of models by bootstrapping the data

Synonym

bootstrap aggregation

Random forest

A type of bagged estimate based on decision tree models

Synonym

bagged decision trees

Variable importance

A measure of the importance of a predictor variable in the performance of the model

The ensemble approach has been applied to and across many different modeling methods, most publicly in the Netflix Prize, in which Netflix

offered a $1 million prize to any contestant who came up with a model that produced a 10% improvement in predicting the rating that a Netflix customer would award a movie. The simple version of ensembles is as follows:

1. Develop a predictive model and record the predictions for a given data set.
2. Repeat for multiple models on the same data.
3. For each record to be predicted, take an average (or a weighted average, or a majority vote) of the predictions.

Ensemble methods have been applied most systematically and effectively to decision trees. Ensemble tree models are so powerful that they provide a way to build good predictive models with relatively little effort.

Going beyond the simple ensemble algorithm, there are two main variants of ensemble models: *bagging* and *boosting*. In the case of ensemble tree models, these are referred to as *random forest* models and *boosted tree* models. This section focuses on bagging; boosting is covered in "Boosting".

Bagging

Bagging, which stands for "bootstrap aggregating," was introduced by Leo Breiman in 1994. Suppose we have a response Y and P predictor variables $\mathbf{X} = X_1, X_2, \cdots, X_P$ with N records.

Bagging is like the basic algorithm for ensembles, except that, instead of fitting the various models to the same data, each new model is fitted to a bootstrap resample. Here is the algorithm presented more formally:

1. Initialize M, the number of models to be fit, and n, the number of records to choose ($n < N$). Set the iteration $m = 1$.
2. Take a bootstrap resample (i.e., with replacement) of n records from the training data to form a subsample Y_m and $\mathbf{X}_m$ (the bag).

3. Train a model using Y_m and $\mathbf{X}_m$ to create a set of decision rules $\widehat{f}_m(\mathbf{X})$.
4. Increment the model counter $m = m + 1$. If $m \leq M$, go to step 2.

In the case where $\widehat{f}_m$ predicts the probability $Y = 1$, the bagged estimate is given by:

$$\widehat{f} = \frac{1}{M}\left(\widehat{f}_1(\mathbf{X}) + \widehat{f}_2(\mathbf{X}) + \ldots + \widehat{f}_M(\mathbf{X})\right)$$

Random Forest

The *random forest* is based on applying bagging to decision trees, with one important extension: in addition to sampling the records, the algorithm also samples the variables.[4] In traditional decision trees, to determine how to create a subpartition of a partition A, the algorithm makes the choice of variable and split point by minimizing a criterion such as Gini impurity (see "Measuring Homogeneity or Impurity"). With random forests, at each stage of the algorithm, the choice of variable is limited to a *random subset of variables*. Compared to the basic tree algorithm (see "The Recursive Partitioning Algorithm"), the random forest algorithm adds two more steps: the bagging discussed earlier (see "Bagging and the Random Forest"), and the bootstrap sampling of variables at each split:

1. Take a bootstrap (with replacement) subsample from the *records*.
2. For the first split, sample $p < P$ *variables* at random without replacement.
3. For each of the sampled variables $X_{j(1)}, X_{j(2)}, \ldots, X_{j(p)}$, apply the splitting algorithm:
    a. For each value $s_{j(k)}$ of $X_{j(k)}$:
        i. Split the records in partition A, with $X_{j(k)} < s_{j(k)}$ as one partition and the remaining records where

$X_{j(k)} \geq s_{j(k)}$ as another partition.

ii. Measure the homogeneity of classes within each subpartition of A.

b. Select the value of $s_{j(k)}$ that produces maximum within-partition homogeneity of class.

4. Select the variable $X_{j(k)}$ and the split value $s_{j(k)}$ that produces maximum within-partition homogeneity of class.
5. Proceed to the next split and repeat the previous steps, starting with step 2.
6. Continue with additional splits, following the same procedure until the tree is grown.
7. Go back to step 1, take another bootstrap subsample, and start the process over again.

How many variables to sample at each step? A rule of thumb is to choose $\sqrt{P}$ where P is the number of predictor variables.

There are several implementations of random forests in R and Python. For a long time, the package `randomForest` written by Breiman and Cutler was the most popular implementation. The `ranger` package is a newer and faster implementation. It is also the default method used in `tidymodels` with the `rand_forest` function from the `parsnip` package. The following applies this package to the loan data (see "K-Nearest Neighbors" for a description of the data):

```
rf <- rand_forest(mode = "classification") %>%
  set_engine("ranger", seed = 123) %>%  ❶
  fit(outcome ~ borrower_score + payment_inc_ratio, data = loan3000)
rf
---
parsnip model object

Ranger result
```

```
Call:
 ranger::ranger(x = maybe_data_frame(x), y = y, seed = ~123, num.threads = 1,
                verbose = FALSE, probability = TRUE)

Type:                             Probability estimation
Number of trees:                  500
Sample size:                      3000
Number of independent variables:  2
Mtry:                             1
Target node size:                 10
Variable importance mode:         none
Splitrule:                        gini
OOB prediction error (Brier s.):  0.2300053
```

❶ Setting the `seed` argument ensures that the results are reproducible.

In Python, we use the method `sklearn.ensemble.RandomForestClassifier`:

```
from sklearn.ensemble import RandomForestClassifier

predictors = ["borrower_score", "payment_inc_ratio"]
outcome = "outcome"

X = loan3000[predictors]
y = loan3000[outcome]

rf = RandomForestClassifier(n_estimators=500, random_state=1, oob_score=True,
                            n_jobs=4) ❶
rf.fit(X, y)
```

❶ Setting the argument `n_jobs=4` allows the model to use 4 CPU cores for training and prediction. This can significantly speed up the training process, especially for large data sets.

By default, 500 trees are trained. Since there are only two variables in the predictor set, the algorithm randomly selects the variable on which to split at each stage (i.e., a bootstrap subsample of size 1).

The *out-of-bag* (*OOB*) estimate of error is the error rate for the trained models, applied to the data left out of the training set for that tree. We can

see how the OOB error decreases as more trees are added to the random forest using the following code in R. It trains a sequence of classifiers with an increasing number of trees and keeps track of the OOB errors:

```
num_trees <- c(1, seq(5, 500, 5))
oob_scores <- c()
for (trees in num_trees) {
  rf <- rand_forest(mode = "classification", trees = trees) %>%
    set_engine("ranger", seed = 123) %>%
    fit(outcome ~ borrower_score + payment_inc_ratio, data = loan3000)
  oob_scores <- c(oob_scores, (rf %>% extract_fit_engine())$prediction.error)
}
error_df <- tibble(error_rate = oob_scores, num_trees = num_trees)
ggplot(error_df, aes(x = num_trees, y = error_rate)) +
  geom_line()
```

In Python, the corresponding code using `RandomForestClassifier` is:

```
n_estimator = list(range(20, 505, 5))
oob_scores = []
for n in n_estimator:
    rf = RandomForestClassifier(n_estimators=n, criterion="entropy", n_jobs=4,
                                max_depth=5, random_state=1, oob_score=True)
    rf.fit(X, y)
    oob_scores.append(rf.oob_score_)
df = pd.DataFrame({"n": n_estimator, "oobScore": oob_scores})
df.plot(x="n", y="oobScore")
```

The result is shown in Figure 6-6. The error rate rapidly decreases from over 0.28 before stabilizing around 0.23. The predicted values can be obtained from the `predict` function and plotted as follows in R:

```
rf_df <- augment(rf, loan3000)
ggplot(data = rf_df, aes(x = borrower_score, y = payment_inc_ratio,
    shape = .pred_class, color = .pred_class, size = .pred_class)) +
  geom_jitter(alpha = 0.8, height = 0, width = 0.003) +
  scale_color_manual(values = c("paid off" = "#b8e186", "default" =
"#d95f02")) +
  scale_shape_manual(values = c("paid off" = 0, "default" = 1)) +
  scale_size_manual(values = c("paid off" = 0.5, "default" = 2))
```

In Python, we can create a similar plot as follows:

```
import matplotlib.pyplot as plt

predictions = X.copy()
predictions["prediction"] = rf.predict(X)
predictions.head()

fig, ax = plt.subplots(figsize=(4, 4))

predictions.loc[predictions.prediction == "paid off"].plot(
    x="borrower_score", y="payment_inc_ratio", style=".",
    markerfacecolor="none", markeredgecolor="C1", ax=ax)
predictions.loc[predictions.prediction == "default"].plot(
    x="borrower_score", y="payment_inc_ratio", style="o",
    markerfacecolor="none", markeredgecolor="C0", ax=ax)
ax.legend(["paid off", "default"])
ax.set_xlim(0, 1)
ax.set_ylim(0, 25)
ax.set_xlabel("borrower_score")
ax.set_ylabel("payment_inc_ratio")
```

The plot, shown in Figure 6-7, is quite revealing about the nature of the random forest.

The random forest method is a "black box" method. It produces more accurate predictions than a simple tree, but the simple tree's intuitive decision rules are lost. The random forest predictions are also somewhat noisy: note that some borrowers with a very high score, indicating high creditworthiness, still end up with a prediction of default. This is a result of some unusual records in the data and demonstrates the danger of overfitting by the random forest (see "Bias-Variance Trade-off").

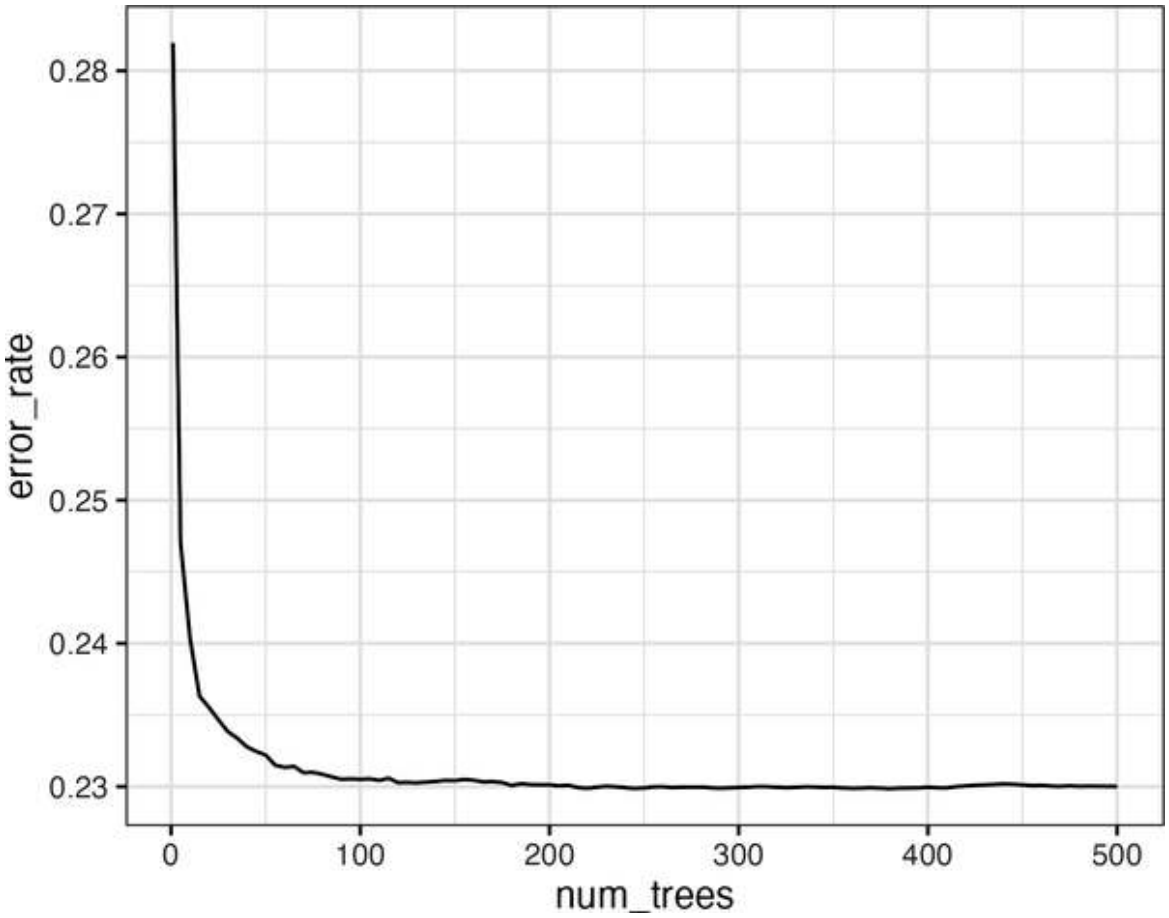


Figure 6-6. An example of the improvement in accuracy of the random forest with the addition of more trees

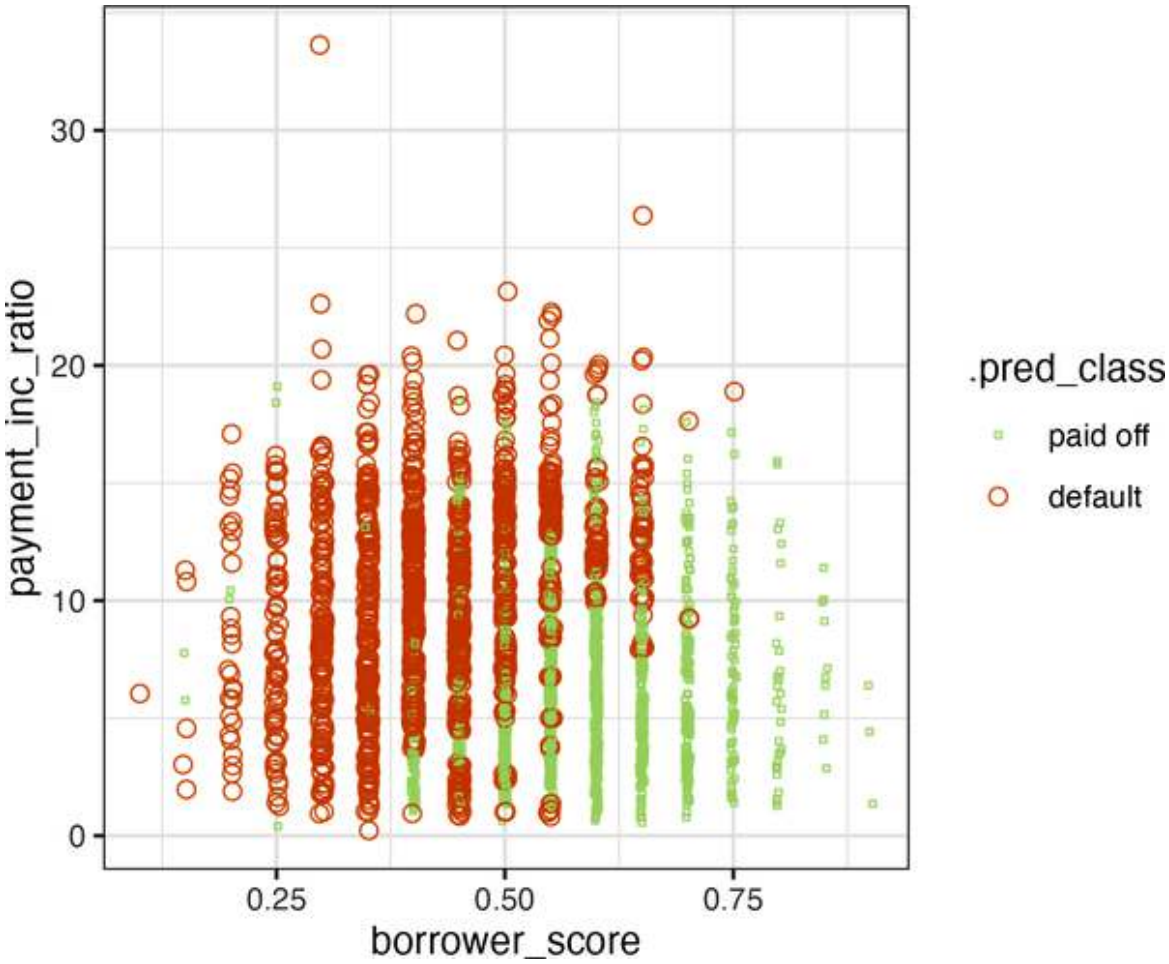


Figure 6-7. The predicted outcomes from the random forest applied to the loan default data

Variable Importance

The power of the random forest algorithm shows itself when you build predictive models for data with many features and records. It has the ability to automatically determine which predictors are important and discover complex relationships between predictors corresponding to interaction terms (see "Interactions and Main Effects"). For example, fit a model to the loan default data with all columns included. The following shows this in R:

```
rf_all <- rand_forest(mode = "classification") %>%
  set_engine("ranger", seed = 123, importance = "impurity") %>%
  fit(outcome ~ ., data = loan_data)
rf_all
---
parsnip model object
```

```
Ranger result

Call:
 ranger::ranger(x = maybe_data_frame(x), y = y,
       seed = ~123, importance = ~"impurity", num.threads = 1, verbose =
FALSE,
       probability = TRUE)

Type:                             Probability estimation
Number of trees:                  500
Sample size:                      45342
Number of independent variables:  18
Mtry:                             4
Target node size:                 10
Variable importance mode:         impurity
Splitrule:                        gini
OOB prediction error (Brier s.):  0.2123043
```

The argument `importance="impurity"` requests that the model stores additional information about the importance of different variables. An alternative approach is to use `importance="permutation"` to compute importance based on accuracy decrease:

```
rf_permutation <- rand_forest(mode = "classification") %>%
  set_engine("ranger", seed = 123, importance = "permutation") %>%
  fit(outcome ~ ., data = loan_data)
```

In Python, we can fit the same model as follows:

```
predictors = ["loan_amnt", "term", "annual_inc", "dti", "payment_inc_ratio",
              "revol_bal", "revol_util", "purpose", "delinq_2yrs_zero",
              "pub_rec_zero", "open_acc", "grade", "emp_length", "purpose_",
              "home_", "emp_len_", "borrower_score"]
outcome = "outcome"

X = pd.get_dummies(loan_data[predictors], drop_first=True)
y = loan_data[outcome]

rf_all = RandomForestClassifier(n_estimators=500, random_state=1, n_jobs=4)
rf_all.fit(X, y)
```

In Python, the `RandomForestClassifier` collects information about feature importance during training and makes it available with the field `feature_importances_`:

```
importances = rf_all.feature_importances_
```

The "Gini decrease" is available as the `feature_importances_` property of the fitted classifier. Accuracy decrease, however, is not available out of the box for Python. We can calculate it (`scores`) using the following code:

```
from collections import defaultdict
import numpy as np
from sklearn import metrics
from sklearn.model_selection import train_test_split

rng = np.random.default_rng(seed=321)
rf = RandomForestClassifier(n_estimators=500, n_jobs=4)
scores = defaultdict(list)

# cross-validate the scores on a number of different random splits of the data
for _ in range(3):
    train_x, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3)
    rf.fit(train_x, train_y)
    acc = metrics.accuracy_score(valid_y, rf.predict(valid_X))
    for column in X.columns:
        X_t = valid_X.copy()
        X_t[column] = rng.permutation(X_t[column].values)
        shuff_acc = metrics.accuracy_score(valid_y, rf.predict(X_t))
        scores[column].append((acc - shuff_acc) / acc)
```

We can visualize the variable importance using both the accuracy decrease and the Gini decrease. In R:

```
library(ranger)  ❶
library(patchwork)  ❷
varimp_gini <- importance(rf_all %>% extract_fit_engine())
varimp_perm <- importance(rf_permutation %>% extract_fit_engine())
df <- tibble(
  predictor = names(varimp_gini),
  impurity = varimp_gini,
  permutation = varimp_perm
)
```

```
g1 <- ggplot(df, aes(x = permutation, y = reorder(predictor, permutation))) +
  geom_point() +
  labs(y = "Importance", title = "Accuracy Decrease")
g2 <- ggplot(df, aes(x = impurity, y = reorder(predictor, permutation))) +
  geom_point() +
  labs(y = "Importance", title = "Gini Decrease")
(g1 + theme_bw()) / (g2 + theme_bw())
```

❶ We need to load the `ranger` package to access the `importance()` function.

❷ The `patchwork` package is used to combine multiple `ggplot2` plots.

The result is shown in Figure 6-8. A similar graph can be created with this Python code:

```
df = pd.DataFrame({
    "feature": X.columns,
    "Accuracy decrease": [np.mean(scores[column]) for column in X.columns],
    "Gini decrease": rf_all.feature_importances_,
})
df = df.sort_values("Accuracy decrease")

fig, axes = plt.subplots(ncols=2, figsize=(8, 5))
ax = df.plot(kind="barh", x="feature", y="Accuracy decrease",
             legend=False, ax=axes[0])
ax.set_ylabel("")

ax = df.plot(kind="barh", x="feature", y="Gini decrease",
             legend=False, ax=axes[1])
ax.set_ylabel("")
ax.get_yaxis().set_visible(False)
```

There are two ways to measure variable importance:

- By the decrease in accuracy of the model if the values of a variable are randomly permuted (`type=1`). Randomly permuting the values has the effect of removing all predictive power for that variable. The accuracy is computed from the OOB data (so this measure is effectively a cross-validated estimate).

- By the mean decrease in the Gini impurity score (see "Measuring Homogeneity or Impurity") for all of the nodes that were split on a variable (`type=2`). This measures how much including that variable improves the purity of the nodes. This measure is based on the training set and is therefore less reliable than a measure calculated on OOB data.

The top and bottom panels of Figure 6-8 show variable importance according to the decrease in accuracy and in Gini impurity, respectively. The variables in both panels are ranked by importance. The variable importance scores produced by these two measures are quite different.

Since the accuracy decrease is a more reliable metric, why should we use the Gini impurity decrease measure? Calculating the Gini impurity is a byproduct of the algorithm, whereas model accuracy by variable requires extra computations (randomly permuting the data and predicting this data). In cases where computational complexity is important, such as in a production setting where thousands of models are being fit, it may not be worth the extra computational effort. In addition, the Gini decrease sheds light on which variables the random forest is using to make its splitting rules (recall that this information, readily visible in a simple tree, is effectively lost in a random forest).

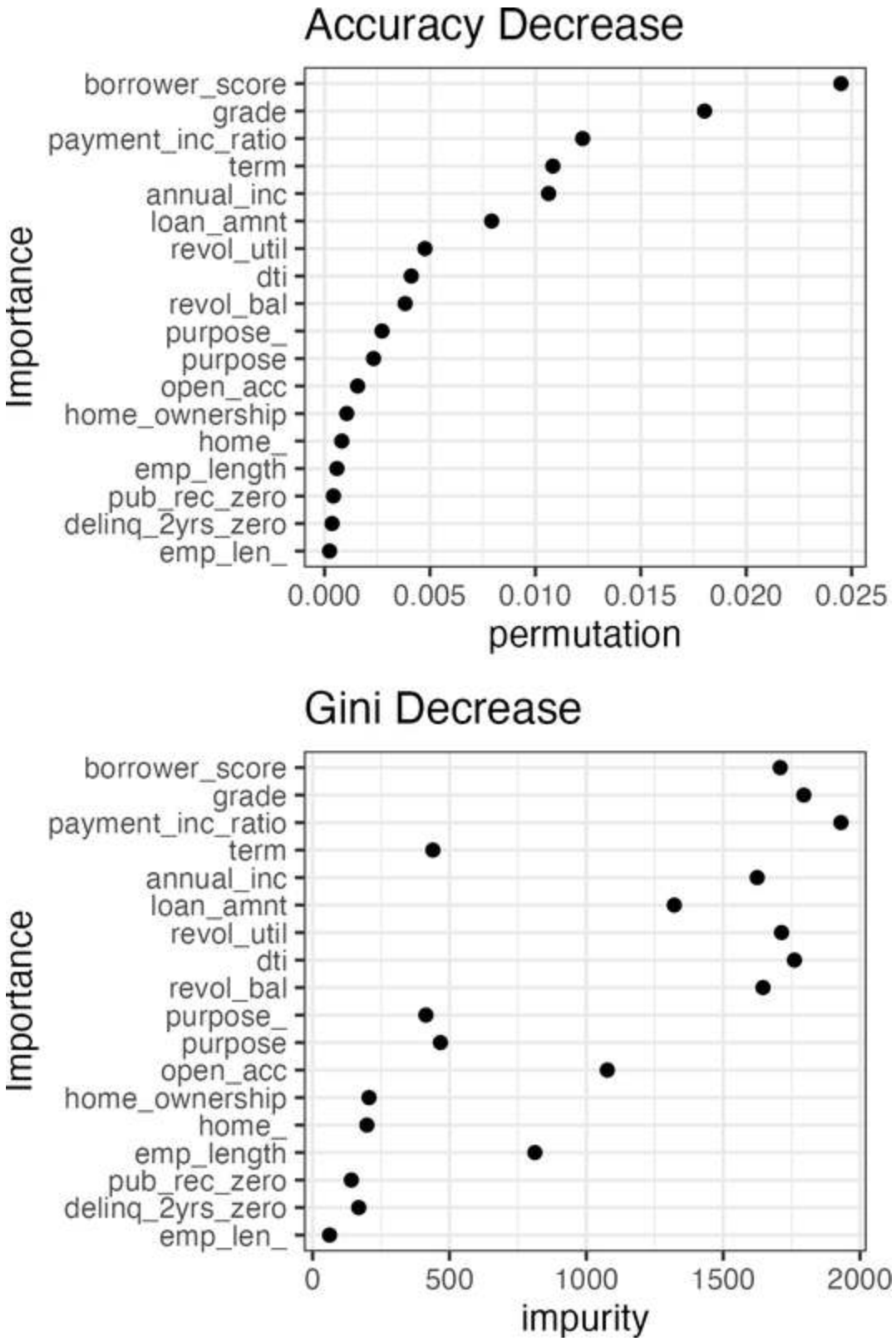


Figure 6-8. The importance of variables for the full model fit to the loan data

Hyperparameters

The random forest, as with many statistical machine learning algorithms, can be considered a black box algorithm with knobs to adjust how the box works. These knobs are called *hyperparameters*, which are parameters that you need to set before fitting a model; they are not optimized as part of the training process. While traditional statistical models require choices (e.g., the choice of predictors to use in a regression model), the hyperparameters for random forest are more critical, especially to avoid overfitting. In particular, the two most important hyperparameters for the random forest are:

`min_n`/`min_samples_leaf`

: The minimum size for terminal nodes (leaves in the tree). The default is 5 for classification and 10 for regression in R. The `scikit-learn` implementation in Python uses a default of 1 for both.

`maxnodes`/`max_leaf_nodes`

: The maximum number of nodes in each decision tree. By default, there is no limit, and the largest tree will be fit subject to the constraints of `min_n`. Note that this is not a tunable parameter in `ranger`, but is, for example, in the `randomForest` package. In Python, you specify the maximum number of terminal nodes. The two parameters are related:

$$\text{maxnodes} = 2^{*}\text{max_leaf_nodes} - 1$$

It may be tempting to ignore these parameters and simply go with the default values. However, using the defaults may lead to overfitting when you apply the random forest to noisy data. When you increase `min_n`/`min_samples_leaf` or set `maxnodes`/`max_leaf_nodes`, the algorithm will fit smaller trees and is less likely to create spurious predictive rules. Cross-validation (see "Cross-Validation") can be used to test the effects of setting different values for hyperparameters.

KEY IDEAS

- Ensemble models improve model accuracy by combining the results from many models.
- Bagging is a particular type of ensemble model based on fitting many models to bootstrapped samples of the data and averaging the models.
- Random forest is a special type of bagging applied to decision trees. In addition to resampling the data, the random forest algorithm samples the predictor variables when splitting the trees.
- A useful output from the random forest is a measure of variable importance that ranks the predictors in terms of their contribution to model accuracy.
- The random forest has a set of hyperparameters that should be tuned using cross-validation to avoid overfitting.

Boosting

Ensemble models have become a standard tool for predictive modeling. *Boosting* is a general technique to create an ensemble of models. It was developed around the same time as *bagging* (see "Bagging and the Random Forest"). Like bagging, boosting is most commonly used with decision trees. Despite their similarities, boosting takes a very different approach—one that comes with many more bells and whistles. As a result, while bagging can be done with relatively little tuning, boosting requires much greater care in its application. If these two methods were cars, bagging could be considered a Honda Accord (reliable and steady), whereas boosting could be considered a Porsche (powerful but requires more care).

In linear regression models, the residuals are often examined to see if the fit can be improved (see "Partial Residual Plots and Nonlinearity"). Boosting takes this concept much further and fits a series of models, in which each successive model seeks to minimize the error of the previous model. Several variants of the algorithm are commonly used: *Adaboost*, *gradient boosting*, and *stochastic gradient boosting*. The latter, stochastic gradient boosting, is the most general and widely used. Indeed, with the right choice of parameters, the algorithm can emulate the random forest.

KEY TERMS FOR BOOSTING

Ensemble

Forming a prediction by using a collection of models

Synonym

model averaging

Boosting

A general technique to fit a sequence of models by giving more weight to the records with large residuals for each successive round

Adaboost

An early version of boosting that reweights the data based on the residuals

Gradient boosting

A more general form of boosting that is cast in terms of minimizing a cost function

Stochastic gradient boosting

The most general algorithm for boosting that incorporates resampling of records and columns in each round

Regularization

A technique to avoid overfitting by adding a penalty term to the cost function on the number of parameters in the model

Hyperparameters

Parameters that need to be set before fitting the algorithm

The Boosting Algorithm

There are various boosting algorithms, and the basic idea behind all of them is essentially the same. The easiest to understand is Adaboost, which proceeds as follows:

1. Initialize *M*, the maximum number of models to be fit, and set the iteration counter $m = 1$. Initialize the observation weights $w_i = 1/N$ for $i = 1, 2, \ldots, N$. Initialize the ensemble model $\widehat{F}_0 = 0$.

2. Using the observation weights $w_1, w_2, \ldots, w_N$, train a model $\widehat{f}_m$ that minimizes the weighted error e_m defined by summing the weights for the misclassified observations.

3. Add the model to the ensemble: $\widehat{F}_m = \widehat{F}_{m-1} + \alpha_m \widehat{f}_m$ where $\alpha_m = \frac{1}{2} \log \frac{1-e_m}{e_m}$.

4. Update the weights $w_1, w_2, \ldots, w_N$ so that the weights are increased for the observations that were misclassified. The size of the increase depends on α_m, with larger values of α_m leading to bigger weights.

5. Increment the model counter $m = m + 1$. If $m \leq M$, go to step 2.

The boosted estimate is given by:

$$\widehat{F} = \alpha_1 \widehat{f}_1 + \alpha_2 \widehat{f}_2 + \ldots + \alpha_M \widehat{f}_M$$

By increasing the weights for the observations that were misclassified, the algorithm forces the models to train more heavily on the data for which it performed poorly. The factor α_m ensures that models with lower error have a bigger weight.

Gradient boosting is similar to Adaboost but casts the problem as an optimization of a cost function. Instead of adjusting weights, gradient boosting fits models to a *pseudo-residual*, which has the effect of training more heavily on the larger residuals. In the spirit of the random forest, stochastic gradient boosting adds randomness to the algorithm by sampling observations and predictor variables at each stage.

XGBoost

The most widely used public domain software for boosting is XGBoost, an implementation of stochastic gradient boosting originally developed by Tianqi Chen and Carlos Guestrin at the University of Washington. A computationally efficient implementation with many options, it is available as a package for most major data science software languages. In R, XGBoost is available as the package `xgboost` and with the same name also for Python.

The method `xgboost` has many parameters that can, and should, be adjusted (see "Hyperparameters and Cross-Validation"). Two very important parameters are `subsample`, which controls the fraction of observations that should be sampled at each iteration, and `eta`, a shrinkage factor applied to α_m in the boosting algorithm (see "The Boosting Algorithm"). Using `subsample` makes boosting act like the random forest except that the sampling is done without replacement. The shrinkage parameter `eta` is helpful to prevent overfitting by reducing the change in the weights (a smaller change in the weights means the algorithm is less likely to overfit to the training set). The following applies `xgboost` in R to the loan data with just two predictor variables:

```
xgb <- boost_tree(mode = "classification",
  trees = 100, learn_rate = 0.1, sample_size = 0.63) %>%
  fit(outcome ~ borrower_score + payment_inc_ratio, data = loan3000)
---
iter training_logloss
1          0.6797622
2          0.6698508
3          0.6597229
```

```
...
98       0.5202932
99       0.5192300
100      0.5187753
```

Note that `xgboost` does not support the formula syntax and would require the predictors to be converted to a `data.matrix` and the response to be converted to 0/1 variables. By using the `tidymodels` wrapper `boost_tree`, this conversion is handled automatically.

In Python, `xgboost` has two different interfaces: a `scikit-learn` API and a more functional interface like in R. To be consistent with other `scikit-learn` methods, some parameters were renamed. For example, `eta` is renamed to `learning_rate`; using `eta` will not fail, but it will not have the desired effect:

```
from xgboost import XGBClassifier

predictors = ["borrower_score", "payment_inc_ratio"]
outcome = "outcome"

X = loan3000[predictors]
y = loan3000[outcome]
y = pd.Series([1 if o == "default" else 0 for o in loan3000[outcome]])

xgb = XGBClassifier(objective="binary:logistic", subsample=0.63)
xgb.fit(X, y)
---
XGBClassifier(base_score=0.5, booster="gbtree", colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None, objective="binary:logistic",
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
seed=None,
       silent=None, subsample=0.63, verbosity=1)
```

The predicted values can be obtained from the `predict` function in R and, since there are only two variables, plotted versus the predictors:

```
xgb_df <- augment(xgb, loan3000)
ggplot(data = xgb_df, aes(x = borrower_score, y = payment_inc_ratio,
```

```
    color = .pred_class, shape = .pred_class, size = .pred_class)) +
  geom_jitter(alpha = 0.8, height = 0, width = 0.003) +
  scale_color_manual(values = c("paid off" = "#b8e186", "default" =
"#d95f02")) +
  scale_shape_manual(values = c("paid off" = 0, "default" = 1)) +
  scale_size_manual(values = c("paid off" = 0.5, "default" = 2))
```

The same figure can be created in Python using the following code:

```
xgb_df = X.copy()
xgb_df["prediction"] = ["default" if p == 1 else "paid off"
    for p in xgb.predict(X)]
xgb_df["prob_default"] = xgb.predict_proba(X)[:, 0]

fig, ax = plt.subplots(figsize=(6, 4))

xgb_df.loc[xgb_df.prediction == "paid off"].plot(
    x="borrower_score", y="payment_inc_ratio", style=".",
    markerfacecolor="none", markeredgecolor="C1", ax=ax)
xgb_df.loc[xgb_df.prediction == "default"].plot(
    x="borrower_score", y="payment_inc_ratio", style="o",
    markerfacecolor="none", markeredgecolor="C0", ax=ax)
ax.legend(["paid off", "default"])
ax.set_xlim(0, 1)
ax.set_ylim(0, 25)
ax.set_xlabel("borrower_score")
ax.set_ylabel("payment_inc_ratio")
```

The result is shown in Figure 6-9. Qualitatively, this is similar to the predictions from the random forest; see Figure 6-7. The predictions are somewhat noisy in that some borrowers with a very high borrower score still end up with a prediction of default.

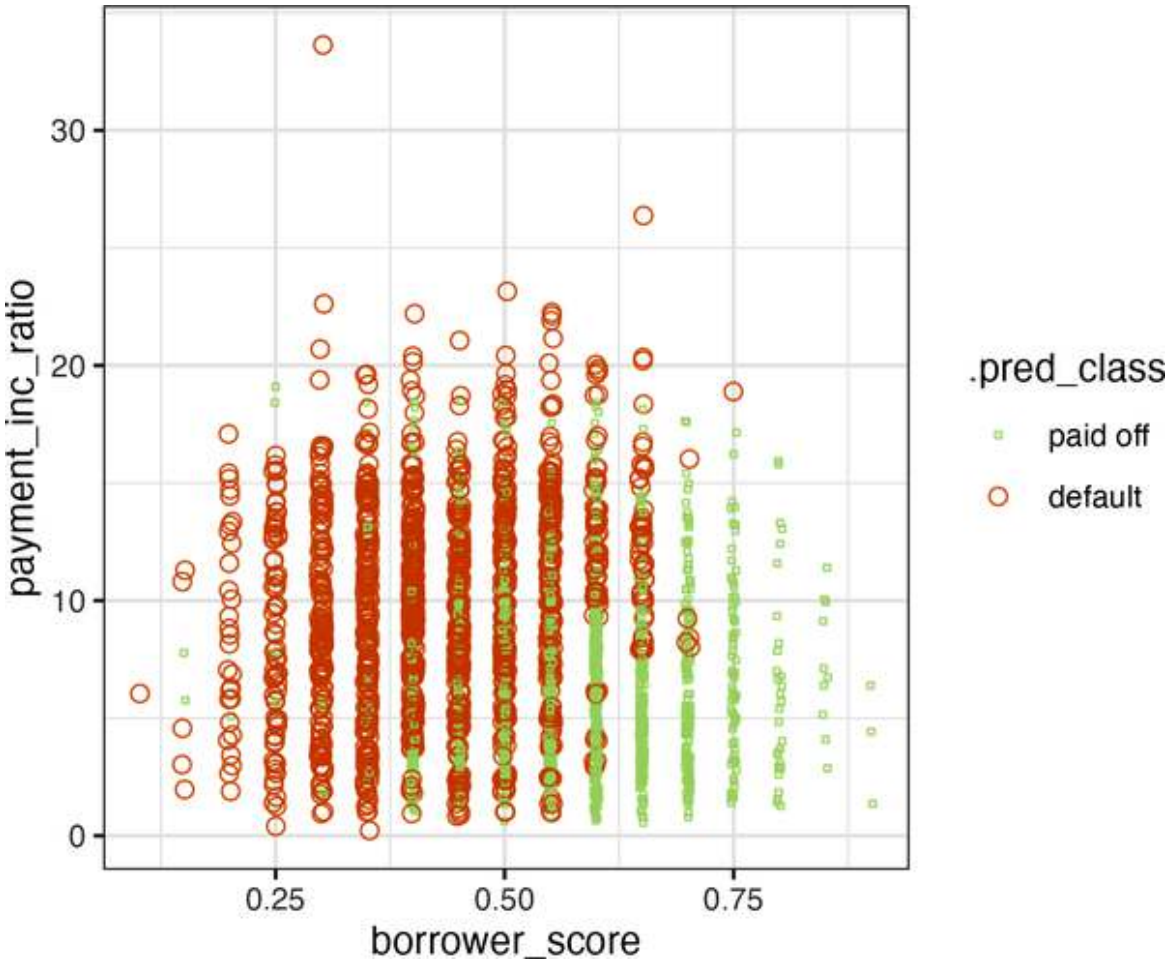


Figure 6-9. The predicted outcomes from XGBoost applied to the loan default data

Regularization: Avoiding Overfitting

Blind application of `xgboost` can lead to unstable models as a result of *overfitting* to the training data. The problem with overfitting is twofold:

- The accuracy of the model on new data not in the training set will be degraded.
- The predictions from the model are highly variable, leading to unstable results.

Any modeling technique is potentially prone to overfitting. For example, if too many variables are included in a regression equation, the model may end up with spurious predictions. However, for most statistical techniques, overfitting can be avoided by a judicious selection of predictor variables.

Even the random forest generally produces a reasonable model without tuning the parameters.

This, however, is not the case for `xgboost`. Fit `xgboost` to the loan data for a training set with all of the variables included in the model. In R, you can do this as follows:

```
library(xgboost)
set.seed(400820)
loan_split <- initial_split(loan_data, prop = 0.75)
train_data <- training(loan_split)
test_data <- testing(loan_split)
train_x <- data.matrix(train_data %>% select(-outcome))
train_y <- factor(train_data$outcome, levels = c("paid off", "default"))
test_x <- data.matrix(test_data %>% select(-outcome))
test_y <- factor(test_data$outcome, levels = c("paid off", "default"))

xgb_default <- xgboost(x = train_x, y = train_y,
  objective = "binary:logistic", nrounds = 250, verbosity = 1,
  eval_metric = "error")

pred_default <- predict(xgb_default, test_x)
error_default <- abs((as.numeric(test_y) - 1) - pred_default) > 0.5
attributes(xgb_default)$evaluation_log[250, ]
mean(error_default)
---
iter train_error
1:  250    0.1087749

[1] 0.3591214
```

We use the function `train_test_split` in Python to split the data set into training and test sets:

```
predictors = ["loan_amnt", "term", "annual_inc", "dti", "payment_inc_ratio",
              "revol_bal", "revol_util", "purpose", "delinq_2yrs_zero",
              "pub_rec_zero", "open_acc", "grade", "emp_length", "purpose_",
              "home_", "emp_len_", "borrower_score"]
outcome = "outcome"

X = pd.get_dummies(loan_data[predictors], drop_first=True)
y = pd.Series([1 if o == "default" else 0 for o in loan_data[outcome]])

train_x, valid_X, train_y, valid_y = train_test_split(X, y, test_size=10000)
```

```
xgb_default = XGBClassifier(objective="binary:logistic", n_estimators=250,
                            max_depth=6, reg_lambda=0, learning_rate=0.3,
                            subsample=1)
xgb_default.fit(train_x, train_y)

pred_default = xgb_default.predict_proba(valid_X)[:, 1]
error_default = abs(valid_y - pred_default) > 0.5
print("default: ", np.mean(error_default))
```

The test set consists of 10,000 randomly sampled records from the full data, and the training set consists of the remaining records. Boosting leads to an error rate of only 13.3% for the training set. The test set, however, has a much higher error rate of 35.3%. This is a result of overfitting: while boosting can explain the variability in the training set very well, the prediction rules do not apply to new data.

Boosting provides several parameters to avoid overfitting, including the parameters `eta` (or `learning_rate`) and `subsample` (see "XGBoost"). Another approach is *regularization*, a technique that modifies the cost function in order to *penalize* the complexity of the model. Decision trees are fit by minimizing cost criteria such as Gini's impurity score (see "Measuring Homogeneity or Impurity"). In `xgboost`, it is possible to modify the cost function by adding a term that measures the complexity of the model.

There are two parameters in `xgboost` to regularize the model: `alpha` and `lambda`, which correspond to Manhattan distance (L1 regularization) and squared Euclidean distance (L2 regularization), respectively (see "Distance Metrics"). Increasing these parameters will penalize more complex models and reduce the size of the trees that are fit. For example, see what happens if we set `lambda` to 1,000 in R:

```
xgb_penalty <- xgboost(x = train_x, y = train_y,
  learning_rate = 0.1, subsample = 0.63, reg_lambda = 1000,
  objective = "binary:logistic", nrounds = 250, verbosity = 1,
  eval_metric = "error")
pred_penalty <- predict(xgb_penalty, test_x)
error_penalty <- abs((as.numeric(test_y) - 1) - pred_penalty) > 0.5
```

```
attributes(xgb_penalty)$evaluation_log[250, ]
mean(error_penalty)
---
iter train_error
1:  250     0.31021

[1] 0.3300988
```

In the `scikit-learn` API, the parameters are called `reg_alpha` and `reg_lambda`:

```
xgb_penalty = XGBClassifier(objective="binary:logistic", n_estimators=250,
                            max_depth=6, reg_lambda=1000, learning_rate=0.1,
                            subsample=0.63)
xgb_penalty.fit(train_x, train_y)
pred_penalty = xgb_penalty.predict_proba(valid_X)[:, 1]
error_penalty = abs(valid_y - pred_penalty) > 0.5
print("penalty: ", np.mean(error_penalty))
```

Now the training error is only slightly lower than the error on the test set.

The `predict` method in R offers a convenient argument, `iteration_range`, that forces only the first *i* trees to be used in the prediction. This lets us directly compare the in-sample versus out-of-sample error rates as more models are included:

```
error_default <- rep(0, 249)
error_penalty <- rep(0, 249)
for (ntree_limit in 1:249) {
  iteration_range <- c(1, ntree_limit + 1)
  pred_def <- predict(xgb_default, test_x, iteration_range = iteration_range)
  error_default[ntree_limit] <- mean(abs(as.numeric(test_y)-1-pred_def) >=
0.5)
  pred_pen <- predict(xgb_penalty, test_x, iteration_range = iteration_range)
  error_penalty[ntree_limit] <- mean(abs(as.numeric(test_y)-1-pred_pen) >=
0.5)
}
```

In Python, we can call the `predict_proba` method with the `ntree_limit` argument:

```
results = []
for ntree_limit in range(1, 250):
    iteration_range = [1, ntree_limit + 1]
    train_default = xgb_default.predict_proba(train_x,
        iteration_range=iteration_range)[:, 1]
    train_penalty = xgb_penalty.predict_proba(train_x,
        iteration_range=iteration_range)[:, 1]
    pred_default = xgb_default.predict_proba(valid_X,
        iteration_range=iteration_range)[:, 1]
    pred_penalty = xgb_penalty.predict_proba(valid_X,
        iteration_range=iteration_range)[:, 1]
    results.append({
        "iterations": ntree_limit,
        "default train": np.mean(abs(train_y - train_default) > 0.5),
        "penalty train": np.mean(abs(train_y - train_penalty) > 0.5),
        "default test": np.mean(abs(valid_y - pred_default) > 0.5),
        "penalty test": np.mean(abs(valid_y - pred_penalty) > 0.5),
    })

results = pd.DataFrame(results)
results.head()
```

The output from the model returns the error for the training set in the component `xgb_default$evaluation_log`. By combining this with the out-of-sample errors, we can plot the errors versus the number of iterations:

```
errors <- bind_rows(
  tibble(Iterations = 1:250, Dataset = "train", Penalty = "no",
    Error = attributes(xgb_default)$evaluation_log$train_error),
  tibble(Iterations = 1:249, Dataset = "test", Penalty = "no",
    Error = error_default),
  tibble(Iterations = 1:250, Dataset = "train", Penalty = "yes",
    Error = attributes(xgb_penalty)$evaluation_log$train_error),
  tibble(Iterations = 1:249, Dataset = "test", Penalty = "yes",
    Error = error_penalty),
)
ggplot(errors, aes(x = Iterations, y = Error, color = Penalty,
    linetype = Dataset)) +
  geom_line() +
  scale_color_manual(values = c("red", "black"))
```

We can use the `pandas plot` method to create the line graph. The axis returned from the first plot allows us to overlay additional lines onto the same graph. This is a pattern that many of Python's graph packages support:

```
ax = results.plot(x="iterations", y="default test")
results.plot(x="iterations", y="penalty test", ax=ax)
results.plot(x="iterations", y="default train", ax=ax)
results.plot(x="iterations", y="penalty train", ax=ax)
```

The result, displayed in Figure 6-10, shows how the default model steadily improves the accuracy for the training set but actually gets worse for the test set. The penalized model does not exhibit this behavior.

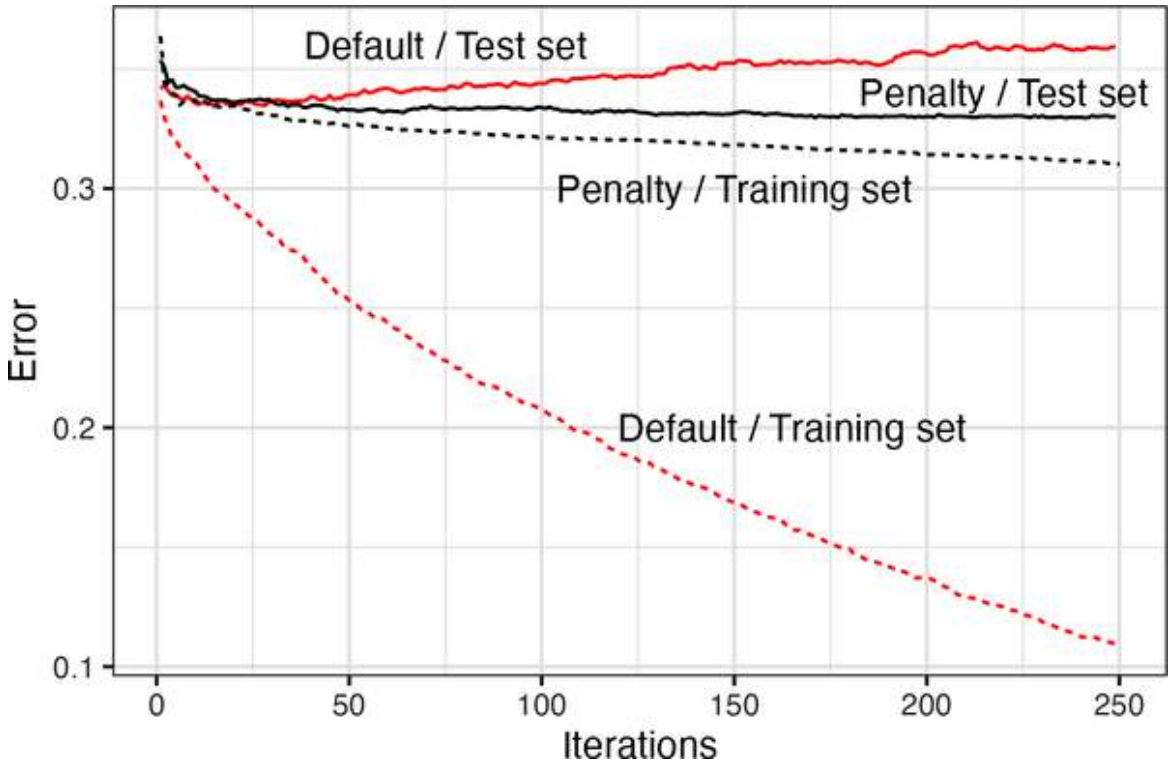


Figure 6-10. The error rate of the default XGBoost versus a penalized version of XGBoost

RIDGE REGRESSION AND THE LASSO

Adding a penalty on the complexity of a model to help avoid overfitting dates back to the 1970s. Least squares regression minimizes the residual sum of squares (RSS); see "Least Squares". *Ridge regression* minimizes the sum of squared residuals plus a penalty term that is a function of the number and size of the coefficients:

$$\sum_{i=1}^{n}\left(Y_i - \hat{b}_0 - \hat{b}_1 X_i - \cdots \hat{b} X_p\right)^2 + \lambda\left(\hat{b}_1^2 + \cdots + \hat{b}_p^2\right)$$

The value of λ determines how much the coefficients are penalized; larger values produce models that are less likely to overfit the data. The lasso is similar, except that it uses Manhattan distance instead of Euclidean distance as a penalty term:

$$\sum_{i=1}^{n}\left(Y_i - \hat{b}_0 - \hat{b}_1 X_i - \cdots \hat{b} X_p\right)^2 + \alpha\left(\left|\hat{b}_1\right| + \cdots + \left|\hat{b}_p\right|\right)$$

Using Euclidean distance is also known as L2 regularization, and using Manhattan distance as L1 regularization. The `xgboost` parameters `lambda` (`reg_lambda`) and `alpha` (`reg_alpha`) are acting in a similar manner.

Hyperparameters and Cross-Validation

`xgboost` has a daunting array of hyperparameters; see "XGBoost Hyperparameters" for a discussion. As seen in "Regularization: Avoiding Overfitting", the specific choice can dramatically change the model fit. Given a huge combination of hyperparameters to choose from, how should we be guided in our choice? A standard solution to this problem is to use *cross-validation*; see "Cross-Validation". Cross-validation randomly splits up the data into K different groups, also called *folds*. For each fold, a model is trained on the data not in the fold and then evaluated on the data in the

fold. This yields a measure of accuracy of the model on out-of-sample data. The best set of hyperparameters is the one given by the model with the lowest overall error as computed by averaging the errors from each of the folds.

To illustrate the technique, we apply it to parameter selection for `xgboost`. In this example, we explore two parameters: the learning rate `learn_rate` or `eta` (`learning_rate`—see "XGBoost") and the maximum depth of trees `tree_depth` or `max_depth`. The parameter `tree_depth` is the maximum depth of a leaf node to the root of the tree, with a default value of 6. This gives us another way to control overfitting: deep trees tend to be more complex and may overfit the data. First we set up the folds and parameter list. With `tidymodels`, this is done as follows in R:

```
set.seed(1234)
resamples <- vfold_cv(loan_data, v = 5)  ❶
grid <- crossing(
  tree_depth = c(3, 6, 12),
  learn_rate = c(0.1, 0.5, 0.9),
)
```

❶ The `vfold_cv` function prepares the five folds for cross-validation. This ensures that all hyperparameter combinations are evaluated on the same folds.

Now we apply the preceding algorithm to compute the error for each model and each fold using five folds:

```
library(future)
plan(multisession, workers = parallel::detectCores(logical = FALSE))  ❶

rec <- recipe(outcome ~ ., data = loan_data) %>%
  step_dummy(all_nominal_predictors())  ❷
model <- boost_tree(mode = "classification",
  tree_depth = tune(), learn_rate = tune())  ❸

wf <- workflow() %>%
  add_model(model) %>%
  add_recipe(rec)
```

```
model_cv <- tune_grid(wf, resamples, grid = grid)  ❹
```

❶ The `future` package is used to parallelize the computation across multiple CPU cores. Specifying the `plan` is all that needs to be done; the `tune_grid` function automatically uses the parallel backend.

❷ The `step_dummy` function creates indicator variables for all categorical predictors.

❸ The `tune()` function indicates which parameters should be tuned.

❹ The `tune_grid` function fits all combinations found in `grid` using the folds defined in `resamples` and collects the results.

In the following Python code, we create all possible combinations of hyperparameters and fit and evaluate models with each combination:

```
from itertools import product

rng = np.random.default_rng(seed=321)
idx = rng.choice(range(5), size=len(X), replace=True)
error = []
for eta, max_depth in product([0.1, 0.5, 0.9], [3, 6, 12]):  ❶
    xgb = XGBClassifier(objective="binary:logistic", n_estimators=250,
                        max_depth=max_depth, learning_rate=eta)
    cv_error = []
    for k in range(5):
        fold_idx = idx == k
        train_x = X.loc[~fold_idx]; train_y = y[~fold_idx]
        valid_X = X.loc[fold_idx]; valid_y = y[fold_idx]

        xgb.fit(train_x, train_y)
        pred = xgb.predict_proba(valid_X)[:, 1]
        cv_error.append(np.mean(abs(valid_y - pred) > 0.5))
    error.append({
        "eta": eta,
        "max_depth": max_depth,
        "avg_error": np.mean(cv_error),
    })
    print(error[-1])
errors = pd.DataFrame(error)

table = errors.pivot_table(index="eta", columns="max_depth",
```

```
values="avg_error")
print(table * 100)
```

❶ We use the function `itertools.product` from the Python standard library to create all possible combinations of the two hyperparameters.

Since we are fitting 45 models, this can take a while. The errors are stored as a matrix with the models along the rows and folds along the columns. Using the function `rowMeans`, we can compare the error rate for the different parameter sets:

```
collect_metrics(model_cv) %>%  ❶
  filter(.metric == "accuracy") %>%
  mutate(error_rate = round(100 * (1 - mean), 2)) %>%
  select(c(tree_depth, learn_rate, error_rate)) %>%
  arrange(error_rate)
---
tree_depth  learn_rate  error_rate
3           0.5         33.34
6           0.1         33.51
3           0.9         33.67
6           0.5         33.68
3           0.1         34.22
6           0.9         34.78
12          0.1         34.91
12          0.5         36.35
12          0.9         38.52
```

❶ The `collect_metrics` function gathers the performance metrics from all cross-validation results and aggregates by hyperparameter combination. In addition, we filter for accuracy and compute the error rate.

Cross-validation suggests that using shallower trees with a smaller value of `learn_rate`/`learning_rate` yields more accurate results. Since these models are also more stable, the best parameters to use are `learn_rate=0.5` and `tree_depth=3` (or possibly `tree_depth=6`).

XGBOOST HYPERPARAMETERS

The hyperparameters for `xgboost` are primarily used to balance overfitting with the accuracy and computational complexity. For a complete discussion of the parameters, refer to the `xgboost` documentation.

`eta`/`learning_rate`

> The shrinkage factor between 0 and 1 applied to α in the boosting algorithm. The default is 0.3, but for noisy data, smaller values are recommended (e.g., 0.1). In Python, the default value is 0.1.

`nrounds`/`n_estimators`

> The number of boosting rounds. If `eta` is set to a small value, it is important to increase the number of rounds since the algorithm learns more slowly. As long as some parameters are included to prevent overfitting, having more rounds doesn't hurt.

`max_depth`

> The maximum depth of the tree (the default is 6). In contrast to the random forest, which fits very deep trees, boosting usually fits shallow trees. This has the advantage of avoiding spurious complex interactions in the model that can arise from noisy data. In Python, the default is 3.

`subsample` *and* `colsample_bytree`

> Fraction of the records to sample without replacement and the fraction of predictors to sample for use in fitting the trees. These parameters, which are similar to those in random forests, help avoid overfitting. The default is 1.0.

`lambda`/`reg_lambda` *and* `alpha`/`reg_alpha`

The regularization parameters to help control overfitting (see "Regularization: Avoiding Overfitting"). Default values for Python are `reg_lambda=1` and `reg_alpha=0`. In R, both values have default of 0.

KEY IDEAS

- Boosting is a class of ensemble models based on fitting a sequence of models, with more weight given to records with large errors in successive rounds.
- Stochastic gradient boosting is the most general type of boosting and offers the best performance. The most common form of stochastic gradient boosting uses tree models.
- XGBoost is a popular and computationally efficient software package for stochastic gradient boosting; it is available in all common languages used in data science.
- Boosting is prone to overfitting the data, and the hyperparameters need to be tuned to avoid this.
- Regularization is one way to avoid overfitting by including a penalty term on the number of parameters (e.g., tree size) in a model.
- Cross-validation is especially important for boosting due to the large number of hyperparameters that need to be set.

Summary

This chapter has described two classification and prediction methods that "learn" flexibly and locally from data, rather than starting with a structural

model (e.g., a linear regression) that is fit to the entire data set. *K*-nearest neighbors is a simple process that looks around at similar records and assigns their majority class (or average value) to the record being predicted. Trying various cutoff (split) values of predictor variables, tree models iteratively divide the data into sections and subsections that are increasingly homogeneous with respect to class. The most effective split values form a path, and also a "rule," to a classification or prediction. Tree models are a very powerful and popular predictive tool, often outperforming other methods. They have given rise to various ensemble methods (random forests, boosting, bagging) that sharpen the predictive power of trees.

Exploration with AI

1. Ask an AI to perform a KNN analysis of the LendingClub data (classifying a loan as paid off or in default), without specifying much in the way of further instructions. What does the AI request from you before doing the analysis? Does the AI make clear what choices it is making with respect to standardization, choice of K, and distance (similarity) metric?

2. Ask an AI to examine the two panels of Figure 6-8, and to comment on why the rankings between the two panels are not more similar.

3. XGBoost exhibits excellent predictive performance along with a pronounced out-of-the box-tendency to overfit. The method has many hyperparameters and offers options to reduce overfitting. Ask an AI to fit XGBoost to the LendingClub data (classifying a loan as paid off or in default) and observe what it does to mitigate overfitting. Is it similar to what is described in the chapter? If not, query it in stages to guide it in that direction. Along the way, observe if it comes up with even better remedies to overfitting.

[1] This and subsequent sections in this chapter © 2020–2025 Datastats, LLC, Peter Bruce, Andrew Bruce, and Peter Gedeck; used with permission.

[2] For this example, we take the first row in the `loan200` data set as the `newloan` and exclude it from the data set for training.

[3] The term CART is a registered trademark of Salford Systems related to its specific implementation of tree models.

[4] The term *random forest* is a trademark of Leo Breiman and Adele Cutler and licensed to Salford Systems. There is no standard nontrademark name, and the term random forest is as synonymous with the algorithm as Kleenex is with facial tissues.

Chapter 7. Unsupervised Learning

The term *unsupervised learning* refers to statistical methods that extract meaning from data without training a model on labeled data (data where an outcome of interest is known). In Chapters 4 to 6, the goal is to build a model (set of rules) to predict a response variable from a set of predictor variables. This is supervised learning. In contrast, unsupervised learning also constructs a model of the data, but it does not distinguish between a response variable and predictor variables.

Unsupervised learning can be used to achieve different goals. In some cases, it can be used to create a predictive rule in the absence of a labeled response. *Clustering* methods can be used to identify meaningful groups of data. For example, using the web clicks and demographic data of a user on a website, we may be able to group together different types of users. The website could then be personalized to these different types.

In other cases, the goal may be to *reduce the dimension* of the data to a more manageable set of variables. This reduced set could then be used as input into a predictive model, such as regression or classification. For example, we may have thousands of sensors to monitor an industrial process. By reducing the data to a smaller set of features, we may be able to build a more powerful and interpretable model to predict process failure than could be built by including data streams from thousands of sensors.

Finally, unsupervised learning can be viewed as an extension of the exploratory data analysis (see Chapter 1) to situations in which you are confronted with a large number of variables and records. The aim is to gain insight into a set of data and how the different variables relate to each other. Unsupervised techniques allow you to sift through and analyze these variables and discover relationships.

UNSUPERVISED LEARNING AND PREDICTION

Unsupervised learning can play an important role in prediction, both for regression and classification problems. In some cases, we want to predict a category in the absence of any labeled data. For example, we might want to predict the type of vegetation in an area from a set of satellite sensory data. Since we don't have a response variable to train a model, clustering gives us a way to identify common patterns and categorize the regions.

Clustering is an especially important tool for the "cold-start problem." In this type of problem, such as launching a new marketing campaign or identifying potential new types of fraud or spam, we initially may not have any response to train a model. Over time, as data is collected, we can learn more about the system and build a traditional predictive model. But clustering helps us start the learning process more quickly by identifying population segments.

Unsupervised learning is also important as a building block for regression and classification techniques. With big data, if a small subpopulation is not well represented in the overall population, the trained model may not perform well for that subpopulation. With clustering, it is possible to identify and label subpopulations. Separate models can then be fit to the different subpopulations. Alternatively, the subpopulation can be represented with its own feature, forcing the overall model to explicitly consider subpopulation identity as a predictor.

Principal Components Analysis

Often, variables will vary together (covary), and some of the variation in one is actually duplicated by variation in another (e.g., restaurant checks and tips). Principal components analysis (PCA) is a technique to discover the way in which numeric variables covary.[1]

KEY TERMS FOR PRINCIPAL COMPONENTS ANALYSIS

Principal component

A linear combination of the predictor variables

Loadings

The weights that transform the predictors into the components

Synonym

weights

Scree plot

A plot of the variances of the components, showing the relative importance of the components, either as explained variance or as proportion of explained variance

The idea in PCA is to combine multiple numeric predictor variables into a smaller set of variables, which are weighted linear combinations of the original set. The smaller set of variables, the *principal components*, "explains" most of the variability of the full set of variables, reducing the dimension of the data. The weights used to form the principal components reveal the relative contributions of the original variables to the new principal components.

PCA was first proposed by Karl Pearson. In what was perhaps the first paper on unsupervised learning, Pearson recognized that in many problems there is variability in the predictor variables, so he developed PCA as a technique to model this variability. PCA can be viewed as the unsupervised version of LDA; see "Discriminant Analysis".

A Simple Example

For two variables, X_1 and X_2, there are two principal components Z_i ($i = 1$ or 2):

$$Z_i = w_{i,1}X_1 + w_{i,2}X_2$$

The weights $(w_{i,1}, w_{i,2})$ are known as the component *loadings*. These transform the original variables into the principal components. The first principal component, Z_1, is the linear combination that best explains the total variation. The second principal component, Z_2, is orthogonal to the first and explains as much of the remaining variation as it can. (If there were additional components, each additional one would be orthogonal to the others.)

PCA FROM DEVIATIONS FROM MEANS

It is also common to compute principal components on deviations from the means of the predictor variables, rather than on the values themselves.

You can compute principal components in R using the `princomp` function. The following performs a PCA on the stock price returns for Chevron (CVX) and ExxonMobil (XOM):

```
library(tidyverse)
sp500_px <- read_csv("sp500_data.csv.gz", show_col_types = FALSE)

oil_px <- sp500_px[, c("CVX", "XOM")]
pca <- princomp(oil_px)
pca$loadings
---
Loadings:
    Comp.1 Comp.2
CVX -0.747  0.665
XOM -0.665 -0.747

               Comp.1 Comp.2
SS loadings       1.0    1.0
Proportion Var    0.5    0.5
Cumulative Var    0.5    1.0
```

To include PCA as a dimensionality reduction technique in a predictive model, it is better to use the `step_pca` function from the `recipes` package:

```
pca_rec <- recipe(data = sp500_px, formula = ~ CVX + XOM) %>%
  step_pca(all_numeric_predictors())  ❶
pca_prep <- pca_rec %>%
  prep()
pca_prep$steps[[1]]$rotation  ❷
---
           PC1        PC2
CVX -0.7469290 -0.6649038
XOM -0.6649038  0.7469290
```

❶ Specify that PCA should be applied to all numeric predictor variables. By default, this will reduce the data to five principal components. Use the `num_comp` argument to change this.

❷ The loadings are termed `rotation` in the `recipes` package.

In Python, use the `scikit-learn` implementation `sklearn.decomposition.PCA`:

```
import pandas as pd
from sklearn.decomposition import PCA

sp500_px = pd.read_csv("sp500_data.csv.gz", index_col=0)
oil_px = sp500_px[["XOM", "CVX"]]

pcs = PCA(n_components=2)  ❶
pcs.fit(oil_px)
loadings = pd.DataFrame(pcs.components_, columns=oil_px.columns)
loadings
```

❶ The PCA object can also be used as a transformer in a `scikit-learn` pipeline as part of the preprocessing for a predictive model.

The weights for CVX and XOM for the first principal component are –0.747 and –0.665, and for the second principal component they are 0.665 and –0.747. How to interpret this? The first principal component is essentially an average of CVX and XOM, reflecting the correlation between

the two energy companies. The second principal component measures when the stock prices of CVX and XOM diverge.

It is instructive to plot the principal components with the data. Here we create a visualization in R:

```
loadings <- pca$loadings
ggplot(data = oil_px, aes(x = CVX, y = XOM)) +
  geom_point(alpha = 0.3) +
  stat_ellipse(type = "norm", level = 0.99) +
  geom_abline(intercept = 0, slope = loadings[2, 1] / loadings[1, 1]) +
  geom_abline(intercept = 0, slope = loadings[2, 2] / loadings[1, 2]) +
  coord_cartesian(xlim = c(-3, 3), ylim = c(-3, 3))
```

The following code creates a similar visualization in Python:

```
import numpy as np

def abline(slope, intercept, ax):
    """Calculate coordinates of a line based on slope and intercept"""
    x_vals = np.array(ax.get_xlim())
    return (x_vals, intercept + slope * x_vals)

ax = oil_px.plot.scatter(x="XOM", y="CVX", alpha=0.3, figsize=(4, 4))
ax.set_xlim(-3, 3)
ax.set_ylim(-3, 3)
ax.plot(*abline(loadings.loc[0, "CVX"] / loadings.loc[0, "XOM"], 0, ax),
        "--", color="C1")
ax.plot(*abline(loadings.loc[1, "CVX"] / loadings.loc[1, "XOM"], 0, ax),
        "--", color="C1")
```

The result is shown in Figure 7-1.

The dashed lines show the direction of the two principal components: the first one is along the long axis of the ellipse, and the second one is along the short axis. You can see that a majority of the variability in the two stock returns is explained by the first principal component. This makes sense since energy stock prices tend to move as a group.

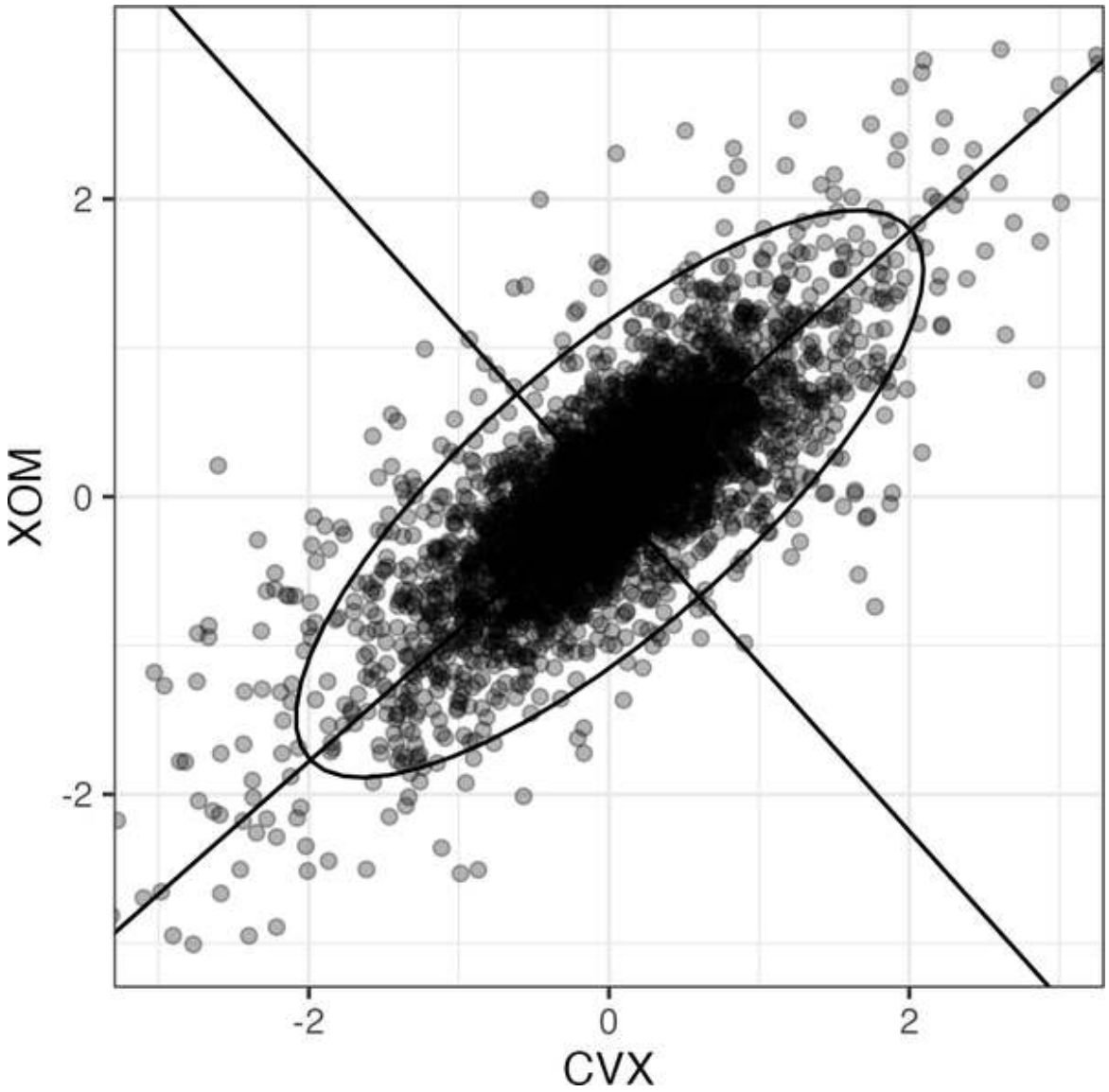


Figure 7-1. The principal components for the stock returns for Chevron (CVX) and ExxonMobil (XOM)

CHANGING SIGNS FOR WEIGHTS

The weights for the first principal component are both negative, but reversing the sign of all the weights does not change the principal component. For example, using weights of 0.747 and 0.665 for the first principal component is equivalent to the negative weights, just as an infinite line defined by the origin and 1, 1 is the same as one defined by the origin and –1, –1.

Computing the Principal Components

Going from two variables to more variables is straightforward. For the first component, simply include the additional predictor variables in the linear combination, assigning weights that optimize the collection of the covariation from all the predictor variables into this first principal component (*covariance* is the statistical term; see "Covariance Matrix"). Calculation of principal components is a classic statistical method, relying on either the correlation matrix of the data or the covariance matrix, and it executes rapidly, not relying on iteration. As noted earlier, PCA works only with numeric variables, not categorical ones. The full process can be described as follows:

1. In creating the first principal component, PCA arrives at the linear combination of predictor variables that maximizes the percent of total variance explained.
2. This linear combination then becomes the first "new" predictor, Z_1.
3. PCA repeats this process, using the same variables with different weights, to create a second new predictor, Z_2. The weighting is done such that Z_1 and Z_2 are uncorrelated.
4. The process continues until you have as many new variables, or components, Z_i as original variables X_i.
5. Choose to retain as many components as are needed to account for most of the variance.
6. The result so far is a set of weights for each component. The final step is to convert the original data into new principal component scores by applying the weights to the original values. These new scores can then be used as the reduced set of predictor variables.

Interpreting Principal Components

The nature of the principal components often reveals information about the structure of the data. There are a couple of standard visualization displays to help you glean insight about the principal components. One such method is

a *scree plot* to visualize the relative importance of principal components (the name derives from the resemblance of the plot to a scree slope; here, the y-axis is the eigenvalue). The following R code shows an example for a few top companies in the S&P 500:

```
syms <- c("AAPL", "MSFT", "CSCO", "INTC", "CVX", "XOM", "SLB",
  "COP", "JPM", "WFC", "USB", "AXP", "WMT", "TGT", "HD", "COST")
top_sp <- sp500_px %>%
  filter(Date >= "2011-01-01") %>%
  select(all_of(syms))
sp_pca <- princomp(top_sp)
screeplot(sp_pca)
```

The information to create a loading plot from the `scikit-learn` result is available in `explained_variance_`. Here, we convert it into a `pandas` data frame and use it to make a bar chart:

```
syms = sorted(["AAPL", "MSFT", "CSCO", "INTC", "CVX", "XOM", "SLB", "COP",
               "JPM", "WFC", "USB", "AXP", "WMT", "TGT", "HD", "COST"])
top_sp = sp500_px.loc[sp500_px.index >= "2011-01-01", syms]

sp_pca = PCA()
sp_pca.fit(top_sp)

explained_variance = pd.DataFrame(sp_pca.explained_variance_)
ax = explained_variance.head(10).plot.bar(legend=False, figsize=(4, 4))
ax.set_xlabel("Component")
```

As seen in Figure 7-2, the variance of the first principal component is quite large (as is often the case), but the other top principal components are significant.

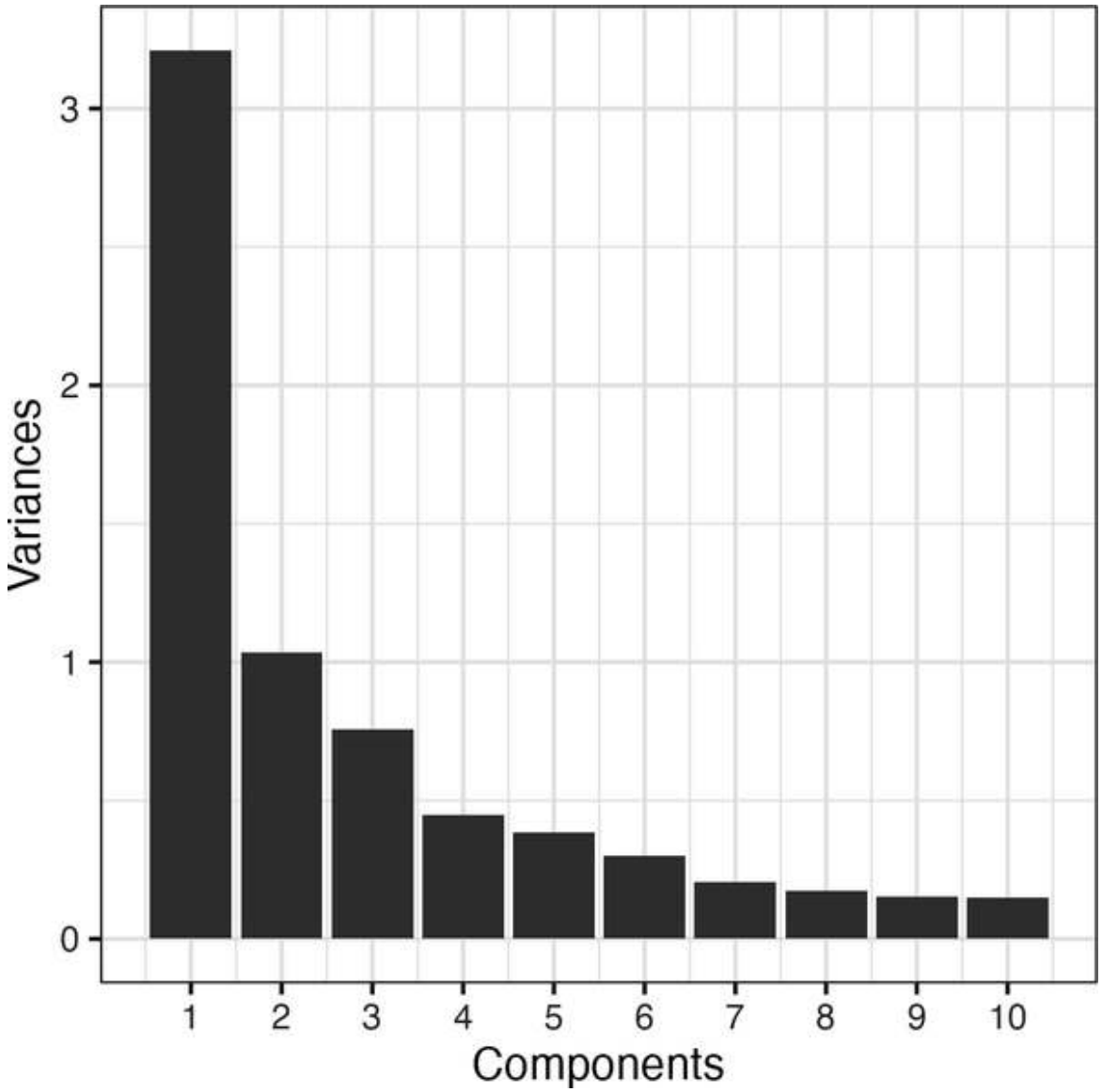


Figure 7-2. A scree plot for a PCA of top stocks from the S&P 500

It can be especially revealing to plot the weights of the top principal components. One way to do this in R is to use the `gather` function from the `tidyr` package in conjunction with `ggplot`:

```
loadings <- sp_pca$loadings[, 1:5]
loadings <- as_tibble(loadings) %>%
  mutate(Symbol = row.names(loadings)) %>%
  pivot_longer(cols = starts_with("Comp"), names_to = "Component",
    values_to = "Weight") %>%
  mutate(color = Weight > 0)
ggplot(loadings, aes(x = Symbol, y = Weight, fill = color)) +
  geom_bar(stat = "identity") +
```

```
  facet_grid(Component ~ ., scales = "free_y") +
  guides(fill = "none") +
  labs(y = "Component Loading") +
  theme_bw() +
  theme(axis.title.x = element_blank(),
    axis.text.x = element_text(angle = 90, vjust = 0.5))
```

Here is the code to create the same visualization in Python:

```
import matplotlib.pyplot as plt

loadings = pd.DataFrame(sp_pca.components_[0:5, :], columns=top_sp.columns)
max_loading = 1.01 * np.max(np.max(np.abs(loadings.loc[0:5, :])))

f, axes = plt.subplots(5, 1, figsize=(5, 6), sharex=True)
for i, ax in enumerate(axes):
    pc_loadings = loadings.loc[i, :]
    colors = ["C0" if loading > 0 else "C1" for loading in pc_loadings]
    ax.axhline(color="#888888")
    pc_loadings.plot.bar(ax=ax, color=colors)
    ax.set_ylabel(f"PC{i + 1}")
    ax.set_ylim(-max_loading, max_loading)
```

The loadings for the top five components are shown in Figure 7-3. The loadings for the first principal component have the same sign: this is typical for data in which all the columns share a common factor (in this case, the overall stock market trend). The second component captures the price changes of energy stocks as compared to the other stocks. The third component is primarily a contrast in the movements of Apple and Costco. The fourth component contrasts the movements of Schlumberger (SLB) to the other energy stocks. Finally, the fifth component is mostly dominated by financial companies.

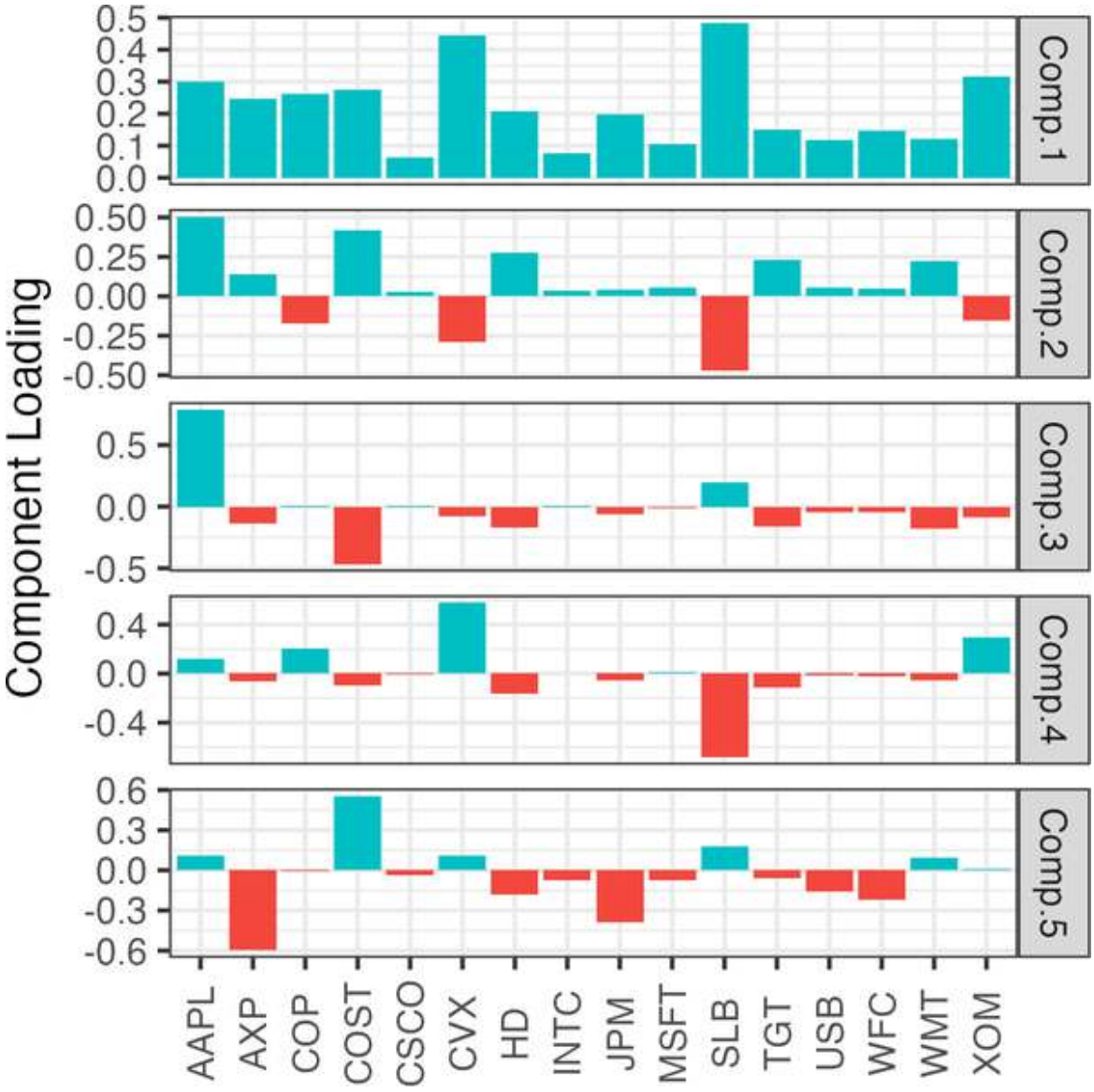


Figure 7-3. The loadings for the top five principal components of stock price returns

HOW MANY COMPONENTS TO CHOOSE?

If your goal is to reduce the dimension of the data, you must decide how many principal components to select. The most common approach is to use an ad hoc rule to select the components that explain "most" of the variance. You can do this visually through the scree plot, as, for example, in Figure 7-2. Alternatively, you could select the top components such that the cumulative variance exceeds a threshold, such as 80%. Also, you can inspect the loadings to determine if the component has an intuitive interpretation. Cross-validation provides a more formal method to select the number of significant components (see "Cross-Validation" for more).

Correspondence Analysis

PCA cannot be used for categorical data; however, a somewhat related technique is *correspondence analysis*. The goal is to recognize associations between categories, or between categorical features. The similarities between correspondence analysis and principal components analysis are mainly under the hood—the matrix algebra for dimension scaling. Correspondence analysis is used mainly for graphical analysis of low-dimensional categorical data and is not used in the same way that PCA is for dimension reduction as a preparatory step with big data.

The input can be seen as a table, with rows representing one variable and columns another, and the cells representing record counts. The output (after some matrix algebra) is a *biplot*—a scatterplot with axes scaled (and with percentages indicating how much variance is explained by that dimension). The meaning of the units on the axes is not intuitively connected to the original data, and the main value of the scatterplot is to illustrate graphically variables that are associated with one another (by proximity on the plot). See, for example, Figure 7-4, in which household tasks are arrayed according to whether they are done jointly or solo (vertical axis), and whether wife or husband has primary responsibility (horizontal axis). Correspondence analysis is many decades old, as is the spirit of this example, judging by the assignment of tasks.

There are a variety of packages for correspondence analysis in R. Here, we use the package `ca`:

```
library(ca)

housetasks <- read_csv("housetasks.csv", show_col_types = FALSE) %>%
  column_to_rownames(var = "Task")
ca_analysis <- ca(housetasks)
plot(ca_analysis)
```

In Python, we can use the `prince` package, which implements correspondence analysis using the `scikit-learn` API:

```
import prince
from adjustText import adjust_text

housetasks = pd.read_csv("housetasks.csv", index_col=0)

ca = prince.CA(n_components=2)
ca = ca.fit(housetasks)

ax = ca.row_coordinates(housetasks).plot.scatter(x=0, y=1, figsize=(6, 6))
ca.column_coordinates(housetasks).plot.scatter(x=0, y=1, ax=ax, c="C1")
texts = []
for idx, row in ca.row_coordinates(housetasks).iterrows():
    texts.append(plt.text(row[0], row[1], idx))
for idx, row in ca.column_coordinates(housetasks).iterrows():
    texts.append(plt.text(row[0], row[1], idx, color="C1"))
adjust_text(texts, only_move={"points": "y", "texts": "y"})
plt.show()
```

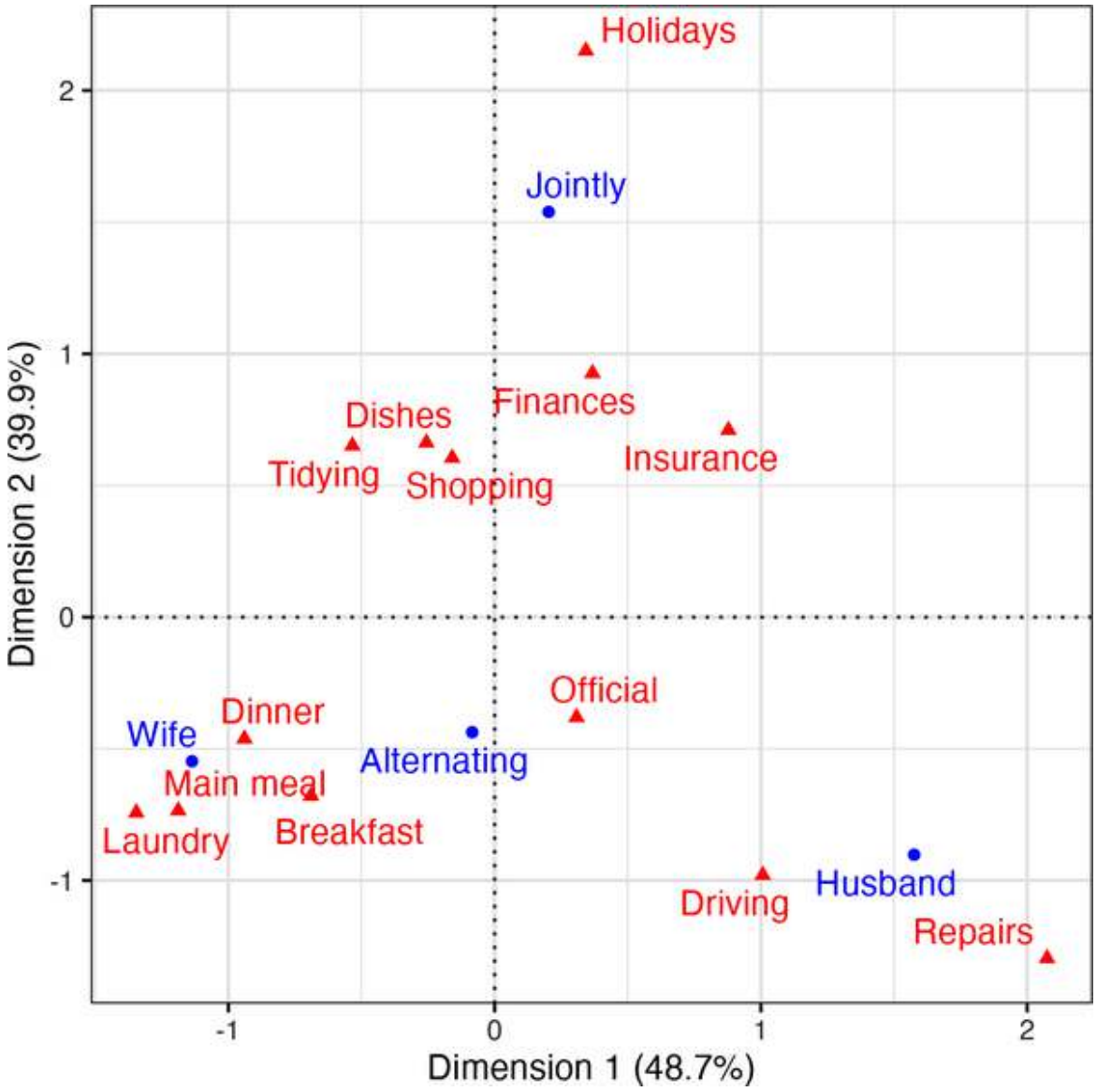


Figure 7-4. Graphical representation of a correspondence analysis of house task data

Analyzing the Results of Principal Components and Correspondence Analysis with AI

You can also use an LLM to help interpret the principal components by providing the loadings as input and asking for an explanation of each component:

Prompt:	Write a concise interpretation of these loadings of a principal component analysis. <paste loadings here> The row labels are ticker symbols.
ChatGPT:	Concise interpretation **PC1 (market/cyclical factor):** Broad positive loadings across stocks, strongest in energy (SLB, CVX, XOM) and large consumer names, indicating a general market or economic cycle effect. **PC2 (consumer/tech versus energy rotation):** Positive loadings for AAPL and retailers (COST, HD, TGT, WMT) and negative loadings for energy stocks, capturing sector rotation between consumer/tech growth and energy. **PC3 (Apple versus retail defensives):** Dominated by a strong positive loading on AAPL and negative loadings on retailers, reflecting Apple-specific or mega-cap tech dynamics opposed to defensive retail. **PC4 (energy structure):** Separates oilfield services (strongly negative SLB) from integrated producers (positive CVX, XOM, COP), reflecting differences within the energy sector. **PC5 (financial risk versus defensive retail):** Strongly negative for financials (especially AXP) and positive for COST, capturing a credit/risk sensitivity factor contrasted with defensive consumer exposure.

Changing the prompt by removing the word *concise* will yield a more detailed explanation of each component.

As we've seen in "Interpreting Visualization Results with AI", LLMs can help interpret complex statistical results. We can use this to interpret the correspondence analysis in Figure 7-4. In ChatGPT, paste this prompt and the figure into the web interface:

Prompt:	Write an interpretation of the results of a correspondence analysis of house task distributions in families shown in the figure. <paste figure here>
ChatGPT:	The correspondence analysis reveals a clear structuring of household tasks along two main dimensions that reflect **gender specialization and task type**. **Dimension 1 (48.7%)** primarily separates tasks associated with the **wife** (left) from those associated with the **husband** (right). On the left, routine domestic and food-related tasks such as *laundry*, *breakfast*, *main meal*, *dinner*, and *tidying* cluster near "Wife," indicating these tasks are predominantly performed by wives. On the right, *repairs* and *driving* lie close to "Husband," reflecting traditionally male-associated responsibilities. **Dimension 2 (39.9%)** distinguishes **routine daily tasks** (lower part of the plot) from **less frequent, organizational, or exceptional tasks** (upper part). Daily chores like meals and laundry are low on this axis, whereas *finances*, *insurance*, and *holidays* appear higher, indicating tasks that are more episodic and planning-oriented. The **"Jointly"** point lies near *holidays* and somewhat toward *finances*, suggesting these activities are more often shared between partners. The **"Alternating"** category is close to the center, indicating no strong association with specific tasks and reflecting flexible or negotiated arrangements. Overall, the analysis shows that housework is strongly gendered along a traditional division for routine chores

and technical tasks, while planning and administrative activities are more likely to be shared within couples.

KEY IDEAS

- Principal components are linear combinations of the predictor variables (numeric data only).
- Principal components are calculated so as to minimize correlation between components, reducing redundancy.
- A limited number of components will typically explain most of the variance in the outcome variable.
- The limited set of principal components can then be used in place of the (more numerous) original predictors, reducing dimensionality.
- A superficially similar technique for categorical data is correspondence analysis, but it is not useful in a big data context.

Further Reading

For a detailed look at the use of cross-validation in principal components, see Rasmus Bro, K. Kjeldahl, A. K. Smilde, and Henk A. L. Kiers, "Cross-Validation of Component Models: A Critical Look at Current Methods", *Analytical and Bioanalytical Chemistry* 390, no. 5 (2008).

K-Means Clustering

Clustering is a technique to divide data into different groups, where the records in each group are similar to one another. A goal of clustering is to identify significant and meaningful groups of data. The groups can be used

directly, analyzed in more depth, or passed as a feature or an outcome to a predictive regression or classification model. *K-means* was the first clustering method to be developed; it is still widely used, owing its popularity to the relative simplicity of the algorithm and its ability to scale to large data sets.

KEY TERMS FOR K-MEANS CLUSTERING

Cluster

A group of records that are similar

Cluster mean

The vector of variable means for the records in a cluster

K

The number of clusters

K-means divides the data into *K* clusters by minimizing the sum of the squared distances of each record to the *mean* of its assigned cluster. This is referred to as the *within-cluster sum of squares* or *within-cluster SS*. *K*-means does not ensure the clusters will have the same size but finds the clusters that are the best separated.

NORMALIZATION

It is typical to normalize (standardize) continuous variables by subtracting the mean and dividing by the standard deviation. Otherwise, variables with large scale will dominate the clustering process (see "Standardization (Normalization, z-Scores)").

A Simple Example

Start by considering a data set with *n* records and just two variables, x and y. Suppose we want to split the data into $K = 4$ clusters. This means assigning each record (x_i, y_i) to a cluster *k*. Given an assignment of n_k records to cluster *k*, the center of the cluster (x_k, y_k) is the mean of the points in the cluster:

$$\bar{x}_k = \frac{1}{n_k} \sum_{\substack{i \in \\ \text{Cluster } k}} x_i$$

$$\bar{y}_k = \frac{1}{n_k} \sum_{\substack{i \in \\ \text{Cluster } k}} y_i$$

CLUSTER MEAN

In clustering records with multiple variables (the typical case), the term *cluster mean* refers not to a single number but to the vector of means of the variables.

The sum of squares within a cluster is given by:

$$\text{SS}_k = \sum_{i \in \text{Cluster } k} (x_i - x_k)^2 + (y_i - y_k)^2$$

K-means finds the assignment of records that minimizes within-cluster sum of squares across all four clusters $\text{SS}_1 + \text{SS}_2 + \text{SS}_3 + \text{SS}_4$:

$$\sum_{k=1}^{4} \text{SS}_k$$

A typical use of clustering is to locate natural, separate clusters in the data. Another application is to divide the data into a predetermined number of separate groups, where clustering is used to ensure the groups are as different as possible from one another.

For example, suppose we want to divide daily stock returns into four groups. *K*-means clustering can be used to separate the data into the best groupings. Note that daily stock returns are reported in a fashion that is, in effect, standardized, so we do not need to normalize the data. In R, the `tidyclust` package is a wrapper around a large variety of clustering algorithms. For *K*-means, we use the `k_means` model. For example, the following finds four clusters based on two variables—the daily stock returns for ExxonMobil (`XOM`) and Chevron (`CVX`):

```
library(tidyclust)
library(tidymodels)
df <- sp500_px %>%
  filter(Date >= "2011-01-01")
kmeans_wf <- workflow() %>%
  add_recipe(recipe(~ XOM + CVX, data = df)) %>%  ❶
  add_model(k_means(num_clusters = 4))  ❷
model <- kmeans_wf %>% fit(df)
```

❶ The model is defined similarly to predictive models, with the exception that the outcome variable is omitted in the formula.

❷ The argument `num_clusters` specifies the number of clusters.

We use the `sklearn.cluster.KMeans` method from `scikit-learn` in Python:

```
from sklearn.cluster import KMeans

df = sp500_px.loc[sp500_px.index >= "2011-01-01", ["XOM", "CVX"]]
kmeans = KMeans(n_clusters=4).fit(df)
```

The cluster assignment for each record is added to a data frame using the `augment` function:

```
augment(model, df) %>%
  select(Date, XOM, CVX, .pred_cluster) %>%
  head()
---
# A tibble: 6 × 4
  Date          XOM    CVX .pred_cluster
```

```
  <date>      <dbl>  <dbl> <fct>
1 2011-01-03 0.737   0.241 Cluster_1
2 2011-01-04 0.169  -0.585 Cluster_2
3 2011-01-05 0.0266  0.447 Cluster_1
4 2011-01-06 0.249  -0.920 Cluster_2
5 2011-01-07 0.337   0.181 Cluster_1
6 2011-01-10 0      -0.464 Cluster_2
```

In `scikit-learn`, the cluster labels are available in the `labels_` field:

```
df["cluster"] = kmeans.labels_
df.head()
```

The first six records are assigned to either cluster 1 or cluster 2. The means of the clusters are easily accessible with the `tidy` function in R:

```
tidy(model)
---
     XOM    CVX  size withinss cluster
   <dbl>  <dbl> <int>    <dbl> <fct>
1  0.232  0.317   457    108.  1
2 -0.329 -0.573   370     94.9 2
3  0.927  1.35    192    122.  3
4 -1.14  -1.75    112     83.7 4
```

In `scikit-learn`, the cluster centers are available in the `cluster_centers_` field:

```
centers = pd.DataFrame(kmeans.cluster_centers_, columns=["XOM", "CVX"])
centers
```

Clusters 2 and 4 represent “down” markets, while clusters 1 and 3 represent “up markets.”

As the *K*-means algorithm uses randomized starting points, the results may differ between subsequent runs and different implementations of the method. In general, you should check that the fluctuations aren’t too large.

In this example, with just two variables, it is straightforward to visualize the clusters and their means:

```
ggplot(augment(model, df), aes(x = XOM, y = CVX)) +
  geom_point(aes(color = .pred_cluster, shape = .pred_cluster), alpha = 0.3) +
  geom_point(data = tidy(model), aes(x = XOM, y = CVX), size = 3, stroke = 2)
```

The `seaborn` `scatterplot` function makes it easy to color (`hue`) and style (`style`) the points by a property:

```
import seaborn as sns

fig, ax = plt.subplots(figsize=(4, 4))
ax = sns.scatterplot(x="XOM", y="CVX", hue="cluster", style="cluster",
                     ax=ax, data=df)
ax.set_xlim(-3, 3)
ax.set_ylim(-3, 3)
centers.plot.scatter(x="XOM", y="CVX", ax=ax, s=50, color="black")
```

The resulting plot, shown in Figure 7-5, shows the cluster assignments and the cluster means. Note that *K*-means will assign records to clusters, even if those clusters are not well separated (which can be useful if you need to optimally divide records into groups).

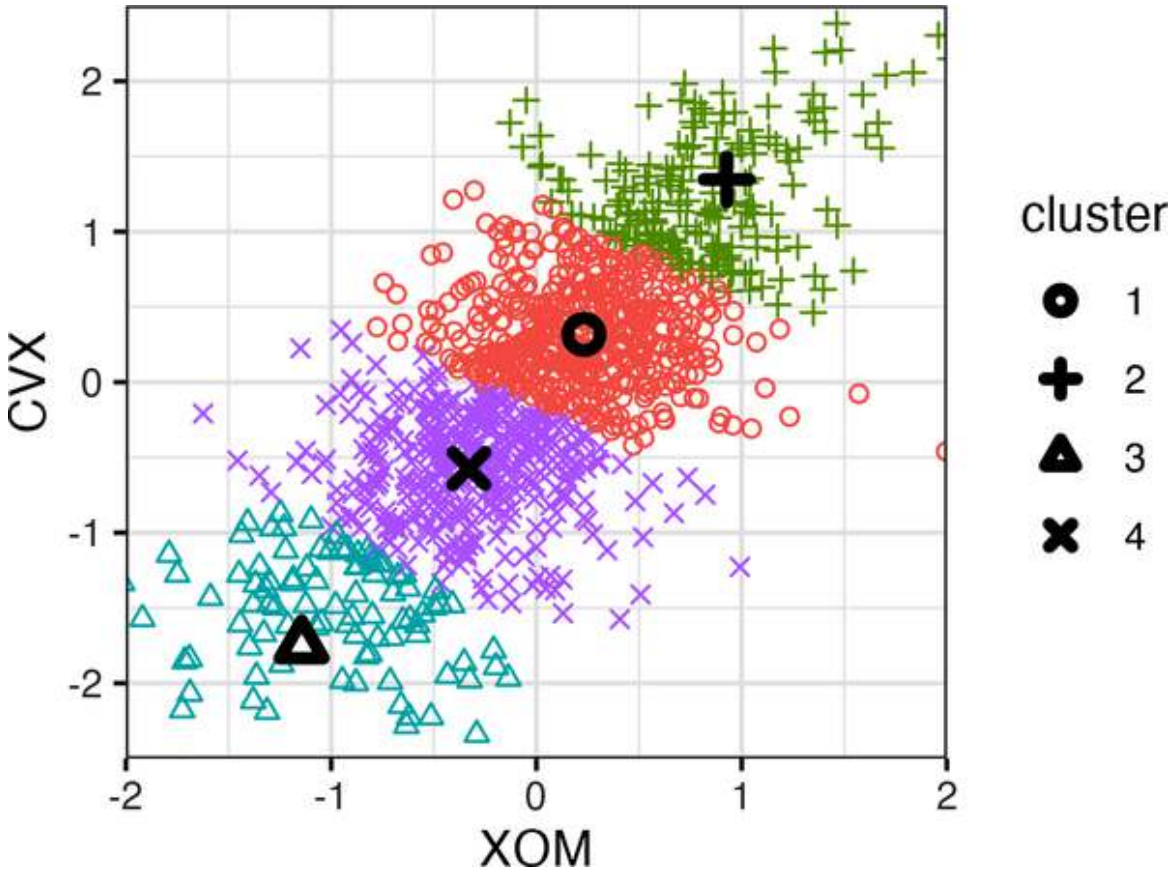


Figure 7-5. The clusters of K-means applied to daily stock returns for ExxonMobil and Chevron (the cluster centers are highlighted with black symbols)

K-Means Algorithm

In general, *K*-means can be applied to a data set with *p* variables $X_1, \ldots, X_p$. While the exact solution to *K*-means is computationally very difficult, heuristic algorithms provide an efficient way to compute a locally optimal solution.

The algorithm starts with a user-specified *K* and an initial set of cluster means and then iterates the following steps:

1. Assign each record to the nearest cluster mean as measured by squared distance.
2. Compute the new cluster means based on the (new) assignment of records.

The algorithm converges when the assignment of records to clusters does not change.

For the first iteration, you need to specify an initial set of cluster means. Usually you do this by randomly assigning each record to one of the *K* clusters and then finding the means of those clusters.

Since this algorithm isn't guaranteed to find the best possible solution, it is recommended to run the algorithm several times using different random samples to initialize the algorithm. When more than one set of iterations is used, the *K*-means result is given by the iteration that has the lowest within-cluster sum of squares.

The `nstart` parameter to the R function `kmeans` allows you to specify the number of random starts to try. For example, the following code runs *K*-means to find 5 clusters using 10 different starting cluster means:

```
syms <- c("AAPL", "MSFT", "CSCO", "INTC", "CVX", "XOM", "SLB", "COP",
  "JPM", "WFC", "USB", "AXP", "WMT", "TGT", "HD", "COST")
df <- sp500_px %>%
  filter(Date >= "2011-01-01") %>%
  select(all_of(syms))
set.seed(10010)
kmeans_wf <- workflow() %>%
  add_recipe(recipe(~ ., data = df)) %>%  ❶
  add_model(k_means(num_clusters = 5) %>%
      set_engine("stats", nstart = 10))  ❷
km <- kmeans_wf %>% fit(df)
```

❶ Here, the formula `~.` indicates that all variables in `df` are to be used for clustering. Make sure that `df` contains only the variables to be used for clustering in this case.

❷ The `nstart` argument is passed to the used `stats` `kmeans` function.

The function automatically returns the best solution out of the 10 different starting points (`nstart`). You can use the argument `iter.max` to set the maximum number of iterations the algorithm is allowed for each random start.

The `scikit-learn` algorithm is repeated 10 times by default (`n_init`). The argument `max_iter` (default 300) can be used to control the number of iterations:

```
syms = sorted(["AAPL", "MSFT", "CSCO", "INTC", "CVX", "XOM", "SLB", "COP",
               "JPM", "WFC", "USB", "AXP", "WMT", "TGT", "HD", "COST"])
top_sp = sp500_px.loc[sp500_px.index >= "2011-01-01", syms]
kmeans = KMeans(n_clusters=5).fit(top_sp)
```

Interpreting the Clusters

An important part of cluster analysis can involve the interpretation of the clusters. The two most important outputs from `kmeans` are the sizes of the clusters and the cluster means. For the example in the previous subsection, the sizes of resulting clusters are given by this R command:

```
tidy(km)$size
---
[1] 287 267 293 178 106
```

In Python, we can use the `collections.Counter` class from the standard library to get this information. Due to differences in the implementation and the inherent randomness of the algorithm, results will vary:

```
from collections import Counter
Counter(kmeans.labels_)
---
Counter({4: 302, 2: 272, 0: 288, 3: 158, 1: 111})
```

The cluster sizes are relatively balanced. Imbalanced clusters can result from distant outliers, or from groups of records very distinct from the rest of the data—both may warrant further inspection.

You can plot the centers of the clusters using the `pivot_longer` function in conjunction with `ggplot`:

```
centers <- tidy(km) %>%
  pivot_longer(cols = all_of(syms), names_to = "Symbol",
```

```
    values_to = "Mean") %>%  ❶
  mutate(Color = Mean > 0)
ggplot(centers, aes(x = Symbol, y = Mean, fill = Color)) +
  geom_bar(stat = "identity", position = "identity", width = 0.75) +
  facet_grid(cluster ~ ., scales = "free_y") +
  guides(fill = "none") +
  labs(y = "Component Loading") +
  theme_bw() +
  theme(axis.title.x = element_blank(),
    axis.text.x = element_text(angle = 90, vjust = 0.5))
```

❶ The `pivot_longer` function reshapes the data from wide to long format, which is useful for plotting with `ggplot`.

The code to create this visualization in Python is similar to what we used for PCA:

```
centers = pd.DataFrame(kmeans.cluster_centers_, columns=syms)

f, axes = plt.subplots(5, 1, figsize=(5, 6), sharex=True)
for i, ax in enumerate(axes):
    center = centers.loc[i, :]
    max_center = 1.01 * np.max(np.max(np.abs(center)))
    colors = ["C0" if cc > 0 else "C1" for cc in center]
    ax.axhline(color="#888888")
    center.plot.bar(ax=ax, color=colors)
    ax.set_ylabel(f"Cluster {i + 1}")
    ax.set_ylim(-max_center, max_center)
```

The resulting plot is shown in Figure 7-6 and reveals the nature of each cluster. For example, clusters 4 and 5 correspond to days on which the market is up and down, respectively. Clusters 1 and 2 are characterized by up-market days for consumer stocks and down-market days for energy stocks, respectively. Finally, cluster 3 captures the days in which energy stocks were up and consumer stocks were down.

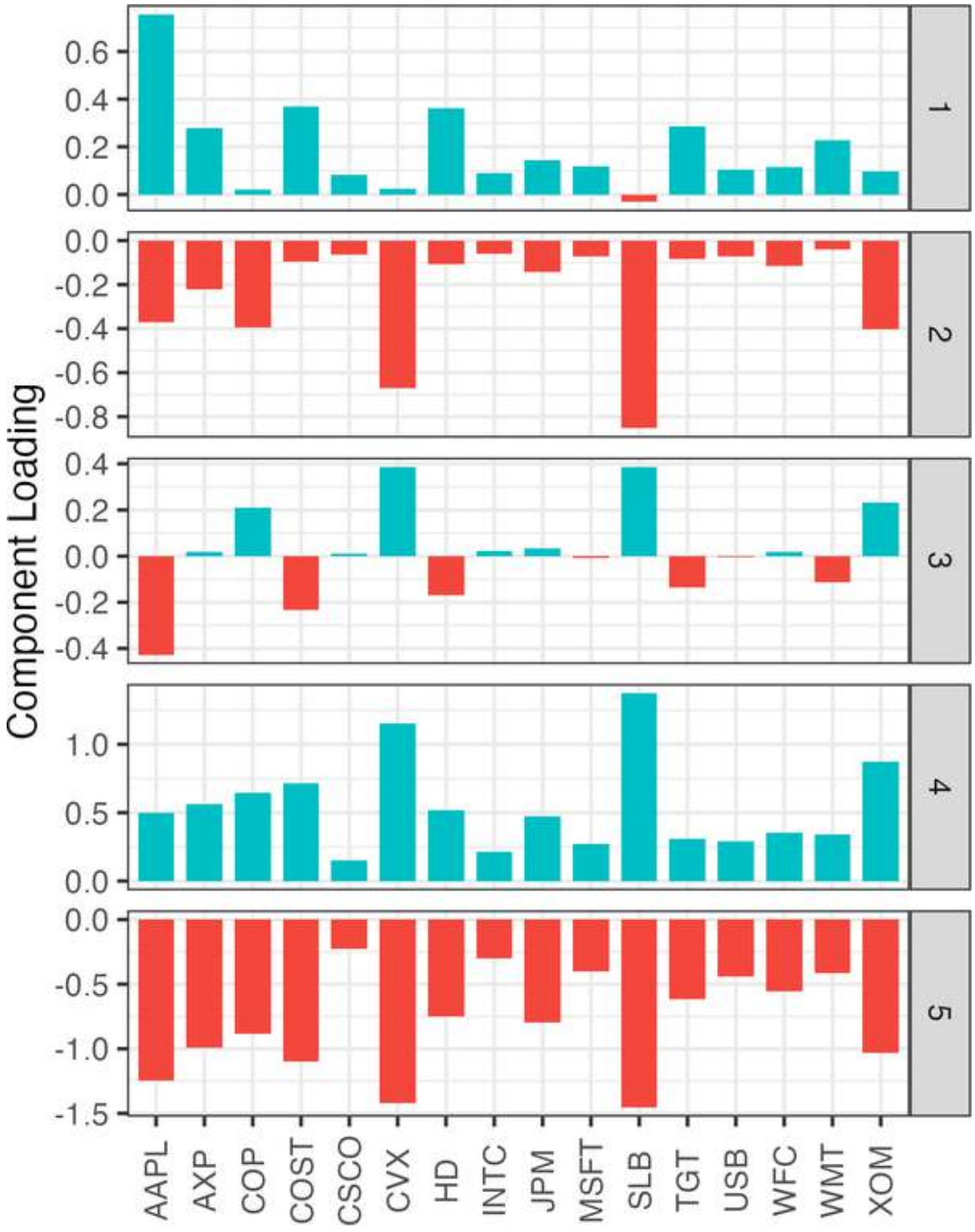


Figure 7-6. The means of the variables in each cluster ("cluster means")

Analyzing the Clusters with AI

You can also use an LLM to help interpret the clustering by providing the cluster centers as input and asking for an explanation of each:

Prompt:

Create a concise interpretation of this result of a kmeans clustering. The data set contained the stock prices (ticker symbols) from 2011 to June 2015.

<paste the output of `tidy(km)` here>

ChatGPT:

Concise interpretation

The k-means clustering identifies **five distinct market regimes** in U.S. equity prices from 2011 to mid-2015, primarily differentiated by the behavior of the **energy sector** and the overall market direction.

Cluster 1 (303 obs): Broadly positive market with strongest gains in **tech and consumer stocks** (AAPL, COST, HD, AXP); energy stocks are mildly positive. This reflects a **consumer/tech-led expansion** typical of post-crisis growth periods.

Cluster 2 (273 obs): Energy-specific downturn, with sharp declines in CVX, XOM, SLB, and COP, while other sectors fall only modestly. This corresponds to oil-price weakness episodes.

Cluster 3 (290 obs): Energy up, consumers down: energy stocks rise while retail and consumer names fall, indicating **sector rotation** linked to commodity or inflation expectations.

Cluster 4 (158 obs): Strong, synchronized gains across all sectors, especially **energy**, capturing **bull-market rallies** with high risk appetite.

Cluster 5 (107 obs): Severe, broad-based declines across all stocks, representing **systemic stress or correction periods**.

Overall, the clustering reveals that market dynamics in this period were dominated by **energy-driven regime**

shifts, with smaller clusters capturing extreme boom and bust phases and larger clusters reflecting more typical market conditions.

CLUSTER ANALYSIS VERSUS PCA

The plot of cluster means is similar in spirit to looking at the loadings for principal components analysis (PCA); see "Interpreting Principal Components". A major distinction is that unlike with PCA, the sign of the cluster means is meaningful. PCA identifies principal directions of variation, whereas cluster analysis finds groups of records located near one another.

Selecting the Number of Clusters

The *K*-means algorithm requires that you specify the number of clusters *K*. Sometimes the number of clusters is driven by the application. For example, a company managing a sales force might want to cluster customers into "personas" to focus and guide sales calls. In such a case, managerial considerations would dictate the number of desired customer segments—for example, two might not yield useful differentiation of customers, while eight might be too many to manage.

In the absence of a cluster number dictated by practical or managerial considerations, a statistical approach could be used. There is no single standard method to find the "best" number of clusters.

A common approach, called the *elbow method*, is to identify when the set of clusters explains "most" of the variance in the data. Adding new clusters beyond this set contributes relatively little in the variance explained. The elbow is the point where the cumulative variance explained flattens out after rising steeply, hence the name of the method.

Figure 7-7 shows the cumulative percent of variance explained for the stock data for the number of clusters ranging from 2 to 14. Where is the elbow in this example? There is no obvious candidate, since the incremental increase in variance explained drops gradually. This is fairly typical in data that does

not have well-defined clusters. This is perhaps a drawback of the elbow method, but it does reveal the nature of the data.

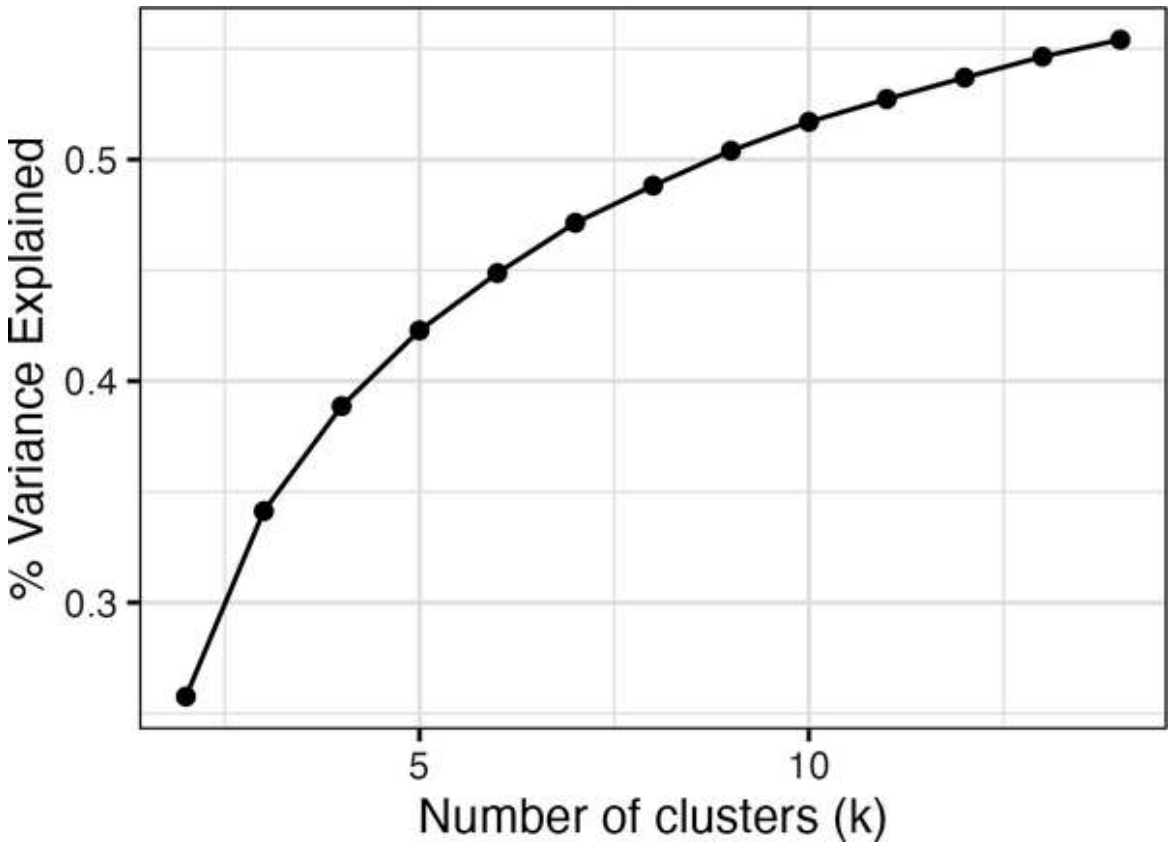


Figure 7-7. The elbow method applied to the stock data

In R, the `k_means` model doesn't provide a single command for applying the elbow method, but it can be readily applied from the output of `kmeans` as shown here:

```
set.seed(10010)
train_kmeans <- function(num_clusters, data) {
  model <- workflow() %>%
    add_recipe(recipe(~ ., data = data)) %>%
    add_model(k_means(num_clusters = num_clusters) %>%
        set_engine("stats", nstart = 50, iter.max = 100)) %>%  ❶
    fit(data)
  return(model)
}
km <- train_kmeans(14, df)
totalss <- extract_fit_engine(km)$totss  ❷

pct_var <- tibble(pct_var = 0, num_clusters = 2:14)
```

```
for (num_clusters in 2:14) {
  km_cluster <- train_kmeans(num_clusters, df)
  pct_var[num_clusters - 1, "pct_var"] <- (
    extract_fit_engine(km_cluster)$betweenss / totalss)
}
ggplot(pct_var, aes(x = num_clusters, y = pct_var)) +
  geom_line() +
  geom_point() +
  labs(x = "Number of clusters (k)", y = "% Variance Explained")
```

❶ The `nstart` and `iter.max` arguments are used to improve the quality of the clustering. If `iter.max` is not large enough, the algorithm may not converge.

❷ We collect the total sum of squares and the between-cluster sum of squares from the wrapped `kmeans` model.

For the `KMeans` result, we get this information from the property `inertia_`. After conversion into a `pandas` data frame, we can use its `plot` method to create the graph:

```
inertia = []
for n_clusters in range(2, 15):
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(top_sp)
    inertia.append(kmeans.inertia_ / n_clusters)

inertias = pd.DataFrame({"n_clusters": range(2, 15), "inertia": inertia})
ax = inertias.plot(x="n_clusters", y="inertia")
plt.xlabel("Number of clusters(k)")
plt.ylabel("Average Within-Cluster Squared Distances")
plt.ylim((0, 1.1 * inertias.inertia.max()))
ax.legend().set_visible(False)
```

In evaluating how many clusters to retain, perhaps the most important test is this: how likely are the clusters to be replicated on new data? Are the clusters interpretable, and do they relate to a general characteristic of the data, or do they just reflect a specific instance? You can assess this, in part, using cross-validation; see “Cross-Validation”.

In general, there’s no single rule that will reliably guide how many clusters to produce.

NUMBER OF CLUSTERS: STATISTICAL APPROACHES

There are several more formal ways to determine the number of clusters based on statistical or information theory. For example, Robert Tibshirani, Guenther Walther, and Trevor Hastie propose a "gap" statistic based on statistical theory to identify the elbow. For most applications, a theoretical approach is probably not necessary, or even appropriate.

KEY IDEAS

- The number of desired clusters, *K*, is chosen by the user.
- The algorithm develops clusters by iteratively assigning records to the nearest cluster mean until cluster assignments do not change.
- Practical considerations usually dominate the choice of *K*; there is no statistically determined optimal number of clusters.

Hierarchical Clustering

Hierarchical clustering is an alternative to *K*-means that can yield very different clusters. Hierarchical clustering allows the user to visualize the effect of specifying different numbers of clusters. It is more sensitive in discovering outlying or aberrant groups or records. Hierarchical clustering also lends itself to an intuitive graphical display, leading to easier interpretation of the clusters.

KEY TERMS FOR HIERARCHICAL CLUSTERING

Dendrogram

A visual representation of the records and the hierarchy of clusters to which they belong

Distance

A measure of how close one *record* is to another

Dissimilarity

A measure of how close one *cluster* is to another

Hierarchical clustering's flexibility comes with a cost, and hierarchical clustering does not scale well to large data sets with millions of records. For even modest-sized data with just tens of thousands of records, hierarchical clustering can require intensive computing resources. Indeed, most of the applications of hierarchical clustering are focused on relatively small data sets.

A Simple Example

Hierarchical clustering works on a data set with n records and p variables and is based on two basic building blocks:

- A distance metric $d_{i,j}$ to measure the distance between two records i and j.
- A dissimilarity metric $D_{A,B}$ to measure the difference between two clusters A and B based on the distances $d_{i,j}$ between the members of each cluster.

For applications involving numeric data, the most important choice is the dissimilarity metric. Hierarchical clustering starts by setting each record as

its own cluster and iterates to combine the least dissimilar clusters.

In R, the `hclust` function can be used to perform hierarchical clustering. One big difference with `hclust` versus `kmeans` is that it operates on the pairwise distances $d_{i,j}$ rather than the data itself. You can compute these using the `dist` function. For example, the following applies hierarchical clustering to the stock returns for a set of companies:

```
syms1 <- c("GOOGL", "AMZN", "AAPL", "MSFT", "CSCO", "INTC", "CVX", "XOM",
"SLB",
  "COP", "JPM", "WFC", "USB", "AXP", "WMT", "TGT", "HD", "COST")
df <- sp500_px %>%
  filter(Date >= "2011-01-01") %>%
  select(all_of(syms1)) %>%
  t() # take transpose: to cluster companies, we need the stocks along the
rows

d <- dist(df)
hcl <- hclust(d)
```

Clustering algorithms will cluster the records (rows) of a data frame. Since we want to cluster the companies, we need to *transpose* (`t`) the data frame and put the stocks along the rows and the dates along the columns. The `tidyclust` package also provides a wrapper around `hclust` that works similarly to the `k_means` model shown earlier.

The `scipy` package offers a number of different methods for hierarchical clustering in the `scipy.cluster.hierarchy` module. Here we use the `linkage` function with the "complete" method:

```
from scipy.cluster.hierarchy import dendrogram, fcluster, linkage

syms1 = ["AAPL", "AMZN", "AXP", "COP", "COST", "CSCO", "CVX", "GOOGL", "HD",
         "INTC", "JPM", "MSFT", "SLB", "TGT", "USB", "WFC", "WMT", "XOM"]
df = sp500_px.loc[sp500_px.index >= "2011-01-01", syms1].transpose()

Z = linkage(df, method="complete")
```

The Dendrogram

Hierarchical clustering lends itself to a natural graphical display as a tree, referred to as a *dendrogram*. The name comes from the Greek words *dendro* (tree) and *gramma* (drawing). In R, you can easily produce this using the `plot` command:

```
plot(hcl)
```

We can use the `dendrogram` method to plot the result of the `linkage` function in Python:

```
fig, ax = plt.subplots(figsize=(5, 5))
dendrogram(Z, labels=list(df.index), ax=ax, color_threshold=0)
plt.xticks(rotation=90)
ax.set_ylabel("distance")
```

The result is shown in Figure 7-8 (note that we are now plotting companies that are similar to one another, not days). The leaves of the tree correspond to the records. The length of the branch in the tree indicates the degree of dissimilarity between corresponding clusters. The returns for Google and Amazon are quite dissimilar to one another and to the returns for the other stocks. The oil stocks (SLB, CVX, XOM, COP) are in their own cluster, Apple (AAPL) is by itself, and the rest are similar to one another.

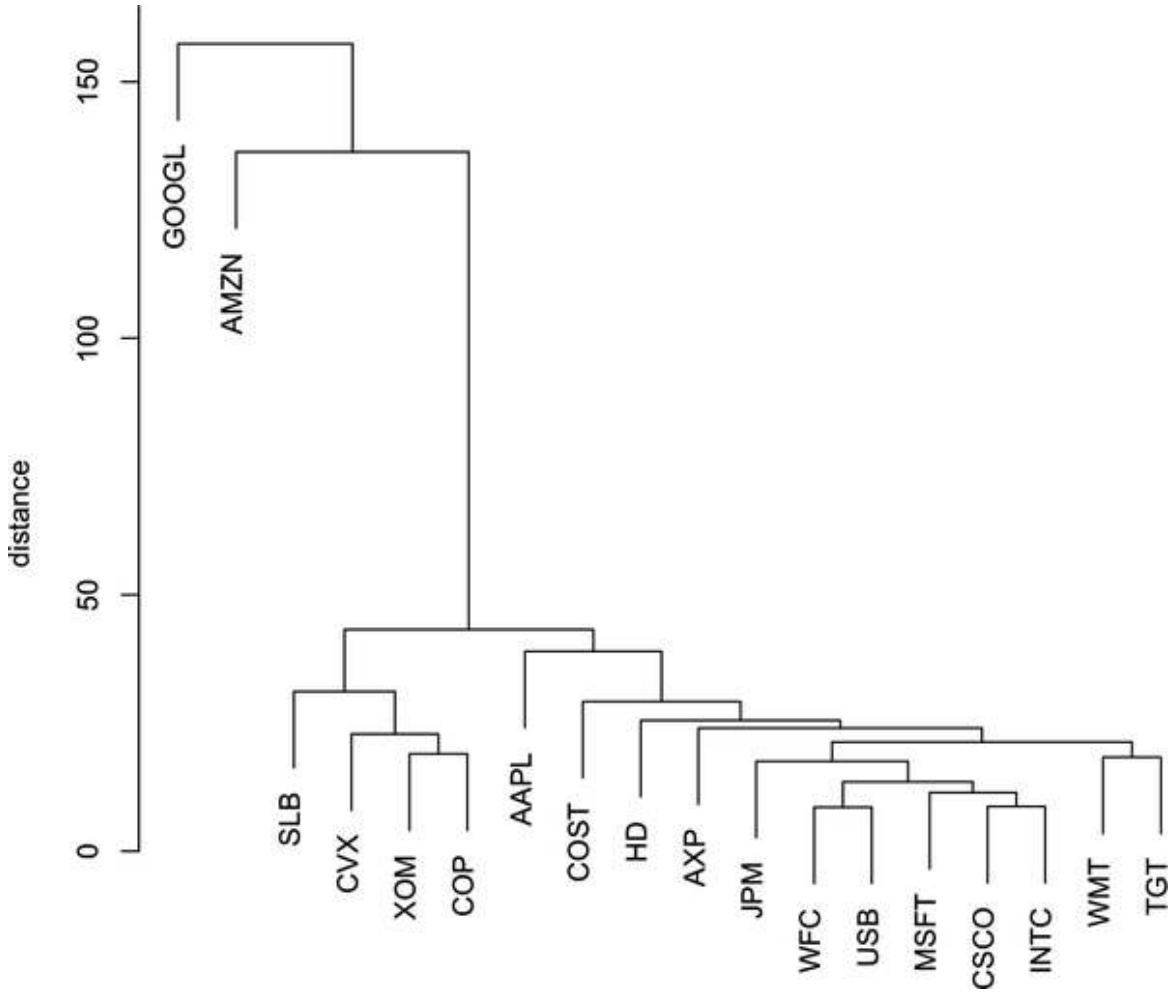


Figure 7-8. A dendrogram of stocks

In contrast to *K*-means, it is not necessary to prespecify the number of clusters. Graphically, you can identify different numbers of clusters with a horizontal line that slides up or down; a cluster is defined wherever the horizontal line intersects the vertical lines. To extract a specific number of clusters, you can use the `cutree` function:

```
cutree(hcl, k = 4)
---
GOOGL  AMZN  AAPL  MSFT  CSCO  INTC   CVX   XOM   SLB   COP   JPM   WFC
    1     2     3     3     3     3     4     4     4     4     3     3
  USB   AXP   WMT   TGT    HD  COST
    3     3     3     3     3     3
```

In Python, you achieve the same with the `fcluster` method:

```
memb = fcluster(Z, 4, criterion="maxclust")
memb = pd.Series(memb, index=df.index)
for key, item in memb.groupby(memb):
    print(f"{key} : {', '.join(item.index)}")
```

The number of clusters to extract is set to 4, and you can see that Google and Amazon each belong to their own cluster. The oil stocks all belong to another cluster. The remaining stocks are in the fourth cluster.

The Agglomerative Algorithm

The main algorithm for hierarchical clustering is the *agglomerative* algorithm, which iteratively merges similar clusters. The agglomerative algorithm begins with each record constituting its own single-record cluster and then builds up larger and larger clusters. The first step is to calculate distances between all pairs of records.

For each pair of records $(x_1, x_2, \ldots, x_p)$ and $(y_1, y_2, \ldots, y_p)$, we measure the distance between the two records, $d_{x,y}$, using a distance metric (see "Distance Metrics"). For example, we can use Euclidean distance:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_p - y_p)^2}$$

We now turn to inter-cluster distance. Consider two clusters A and B, each with a distinctive set of records, $A = (a_1, a_2, \ldots, a_m)$ and $B = (b_1, b_2, \ldots, b_q)$. We can measure the dissimilarity between the clusters $D(A, B)$ by using the distances between the members of A and the members of B.

One measure of dissimilarity is the *complete-linkage* method, which is the maximum distance across all pairs of records between A and B:

$$D(A, B) = \max d(a_i, b_j) \text{ for all pairs } i, j$$

This defines the dissimilarity as the biggest difference between all pairs.

The main steps of the agglomerative algorithm are:

1. Create an initial set of clusters with each cluster consisting of a single record for all records in the data.
2. Compute the dissimilarity $D(C_k, C_\ell)$ between all pairs of clusters k, ℓ.
3. Merge the two clusters C_k and C_ℓ that are least dissimilar as measured by $D(C_k, C_\ell)$.
4. If we have more than one cluster remaining, return to step 2. Otherwise, we are done.

Measures of Dissimilarity

There are four common measures of dissimilarity: *complete linkage*, *single linkage*, *average linkage*, and *minimum variance*. These (plus other measures) are all supported by most hierarchical clustering software, including `hclust` and `linkage`. The complete linkage method defined earlier tends to produce clusters with members that are similar. The single linkage method is the minimum distance between the records in two clusters:

$$D\left(A, B\right) = \min d\left(a_i, b_j\right) \text{ for all pairs } i, j$$

This is a "greedy" method and produces clusters that can contain quite disparate elements. The average linkage method is the average of all distance pairs and represents a compromise between the single and complete linkage methods. Finally, the minimum variance method, also referred to as *Ward's* method, is similar to *K*-means since it minimizes the within-cluster sum of squares (see "K-Means Clustering").

Figure 7-9 applies hierarchical clustering using the four measures to the ExxonMobil and Chevron stock returns. For each measure, four clusters are retained.

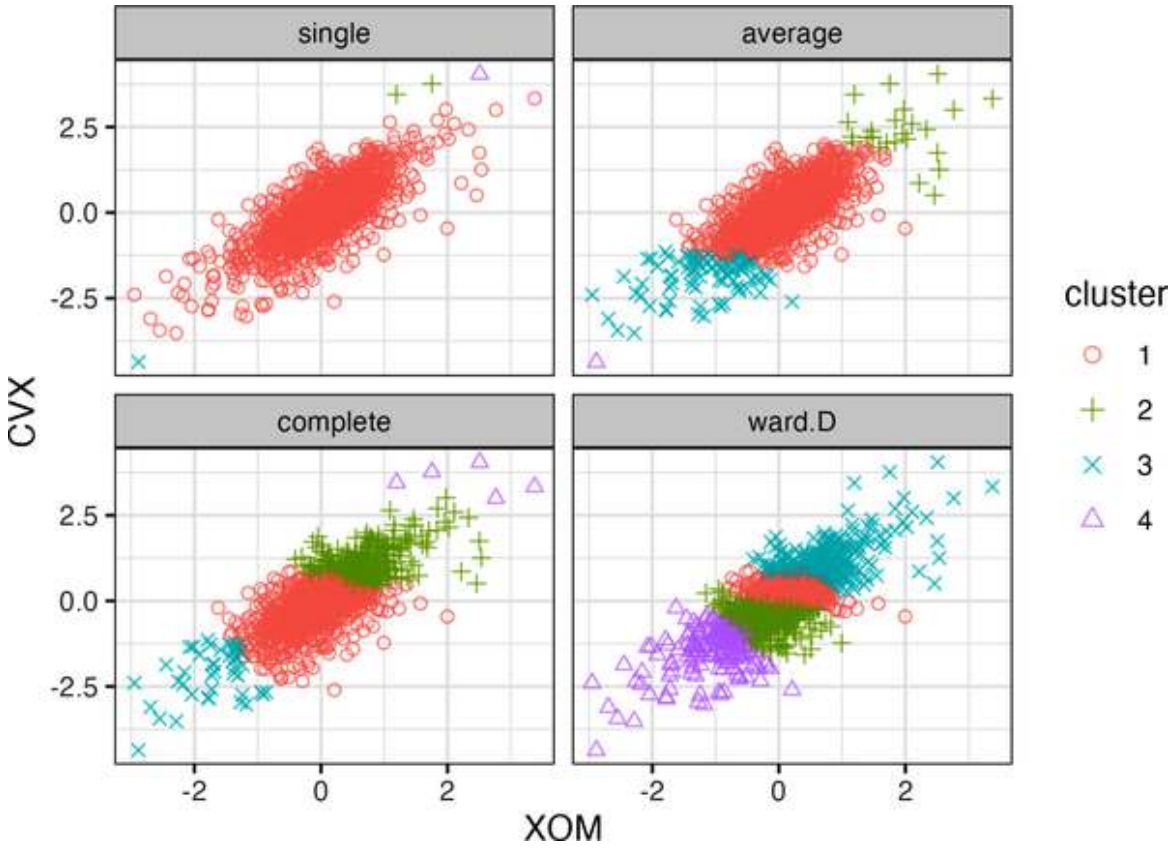


Figure 7-9. A comparison of measures of dissimilarity applied to stock data

The results are strikingly different: the single linkage measure assigns almost all of the points to a single cluster. Except for the minimum variance method (R: `Ward.D`; Python: `ward`), all measures end up with at least one cluster with just a few outlying points. The minimum variance method is most similar to the *K*-means cluster; compare with Figure 7-5.

KEY IDEAS

- Hierarchical clustering starts with every record in its own cluster.
- Progressively, clusters are joined to nearby clusters until all records belong to a single cluster (the agglomerative algorithm).
- The agglomeration history is retained and plotted, and the user (without specifying the number of clusters beforehand) can visualize the number and structure of clusters at different stages.
- Inter-cluster distances are computed in different ways, all relying on the set of all inter-record distances.

Model-Based Clustering

Clustering methods such as hierarchical clustering and *K*-means are based on heuristics and rely primarily on finding clusters whose members are close to one another, as measured directly with the data (no probability model involved). In the past 20 years, significant effort has been devoted to developing *model-based clustering* methods. Adrian Raftery and other researchers at the University of Washington made critical contributions to model-based clustering, including both theory and software. The techniques are grounded in statistical theory and provide more rigorous ways to determine the nature and number of clusters. They could be used, for example, in cases where there might be one group of records that are similar to one another but not necessarily close to one another (e.g., tech stocks with high variance of returns), and another group of records that are similar and also close (e.g., utility stocks with low variance).

Multivariate Normal Distribution

The most widely used model-based clustering methods rest on the *multivariate normal* distribution. The multivariate normal distribution is a generalization of the normal distribution to a set of p variables $X_1, X_2, \ldots, X_p$. The distribution is defined by a set of means $\boldsymbol{\mu} = \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_\mathbf{p}$ and a covariance matrix Σ. The covariance matrix is a measure of how the variables correlate with each other (see "Covariance Matrix" for details on the covariance). The covariance matrix Σ consists of p variances $\sigma_1^2, \sigma_2^2, \ldots, \sigma_p^2$ and covariances $\sigma_{i,j}$ for all pairs of variables $i \neq j$. With the variables put along the rows and duplicated along the columns, the matrix looks like this:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,p} \\ \sigma_{2,1} & \sigma_2^2 & \cdots & \sigma_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p,1} & \sigma_{p,2}^2 & \cdots & \sigma_p^2 \end{bmatrix}$$

Note that the covariance matrix is symmetric around the diagonal from upper-left to lower-right. Since $\sigma_{i,j} = \sigma_{j,i}$, there are only $(p \times (p-1))/2$ covariance terms. In total, the covariance matrix has $(p \times (p-1))/2 + p$ parameters. The distribution is denoted by:

$$(X_1, X_{2,\ldots,} X_p) \sim N_p(\mu, \Sigma)$$

This is a symbolic way of saying that the variables are all normally distributed, and the overall distribution is fully described by the vector of variable means and the covariance matrix.

Figure 7-10 shows the probability contours for a multivariate normal distribution for two variables *X* and *Y* (the 0.5 probability contour, for example, contains 50% of the distribution).

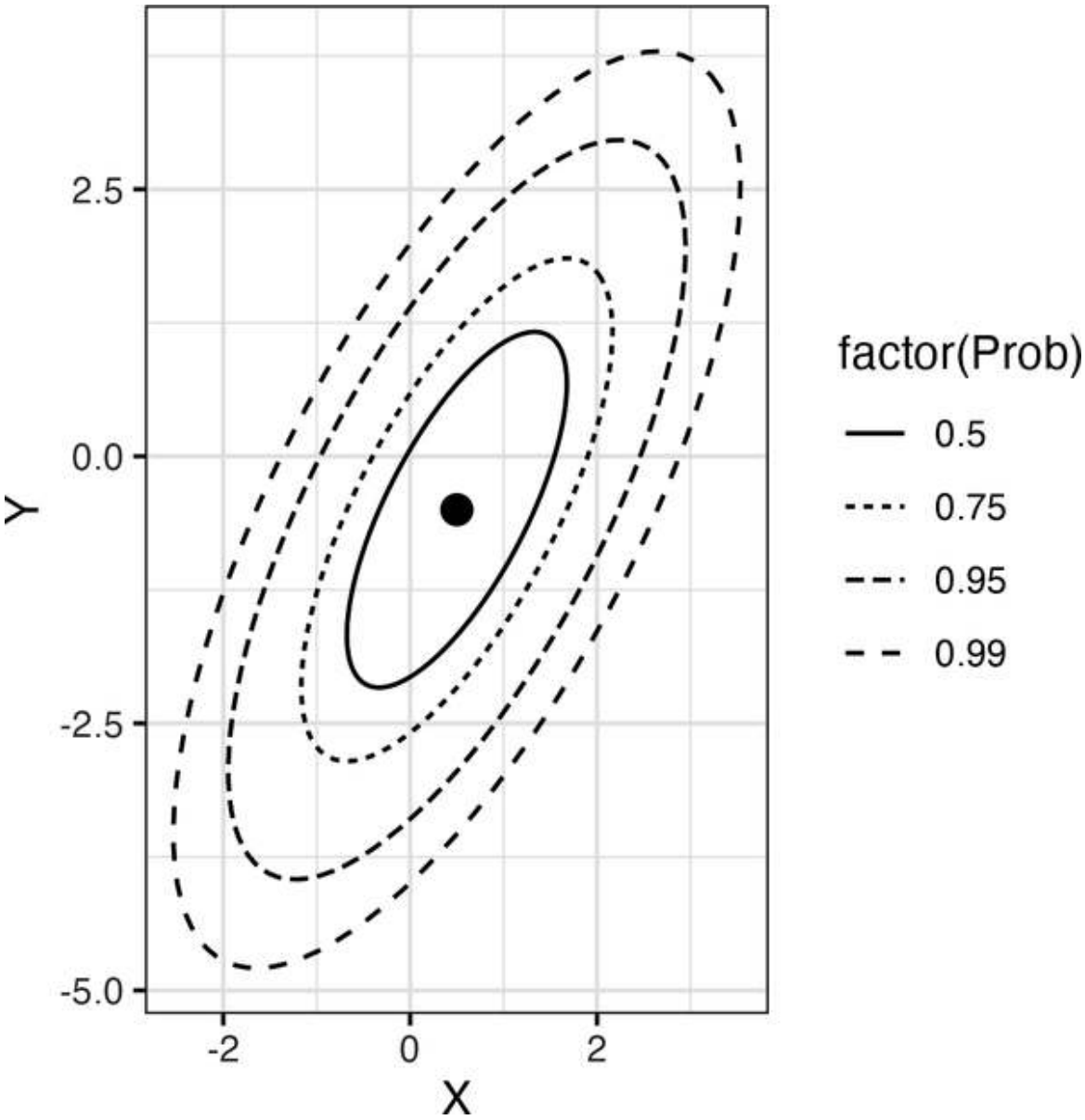


Figure 7-10. Probability contours for a two-dimensional normal distribution

The means are $\mu_x = 0.5$ and $\mu_y = -0.5$, and the covariance matrix is:

$$\Sigma = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

Since the covariance σ_{xy} is positive, X and Y are positively correlated.

Mixtures of Normals

The key idea behind model-based clustering is that each record is assumed to be distributed as one of K multivariate normal distributions, where K is the number of clusters. Each distribution has a different mean μ and covariance matrix Σ. For example, if you have two variables, X and Y, then each row (X_i, Y_i) is modeled as having been sampled from one of K multivariate normal distributions $N(\boldsymbol{\mu}_1, \Sigma_1), N(\mu_2, \Sigma_2), \ldots, N(\boldsymbol{\mu}_K, \Sigma_K)$.

R has a very rich package for model-based clustering called `mclust`, originally developed by Chris Fraley and Adrian Raftery. With this package, we can apply model-based clustering to the stock return data we previously analyzed using K-means and hierarchical clustering:

```
library(mclust)
df <- sp500_px %>%
  filter(Date >= "2011-01-01") %>%
  select(c(XOM, CVX))
mcl <- Mclust(df)
summary(mcl)
---
Mclust VEE (ellipsoidal, equal shape and orientation) model with 2 components:


 log.likelihood    n df       BIC       ICL
      -2255.134 1131  9 -4573.546 -5076.856

Clustering table:
  1   2
963 168
```

`scikit-learn` has the `sklearn.mixture.GaussianMixture` class for model-based clustering:

```
from sklearn.mixture import GaussianMixture

df = sp500_px.loc[sp500_px.index >= "2011-01-01", ["XOM", "CVX"]]
mclust = GaussianMixture(n_components=2).fit(df)
mclust.bic(df)
```

If you execute this code, you will notice that the computation takes significantly longer than other procedures. Extracting the cluster

assignments using the `predict` function, we can visualize the clusters:

```
cluster <- factor(predict(mcl)$classification)
ggplot(df, aes(x = XOM, y = CVX, color = cluster, shape = cluster)) +
  geom_point(alpha = 0.8) +
  scale_shape_manual(
    values = c(1, 3),
    guide = guide_legend(override.aes = aes(size = 2)))
```

Here is the Python code to create a similar figure:

```
fig, ax = plt.subplots(figsize=(4, 4))
colors = [f"C{c}" for c in mclust.predict(df)]
df.plot.scatter(x="XOM", y="CVX", c=colors, alpha=0.5, ax=ax)
ax.set_xlim(-3, 3)
ax.set_ylim(-3, 3)
```

The resulting plot is shown in Figure 7-11. There are two clusters: one cluster in the middle of the data, and a second cluster in the outer edge of the data. This is very different from the clusters obtained using *K*-means (Figure 7-5) and hierarchical clustering (Figure 7-9), which find clusters that are compact.

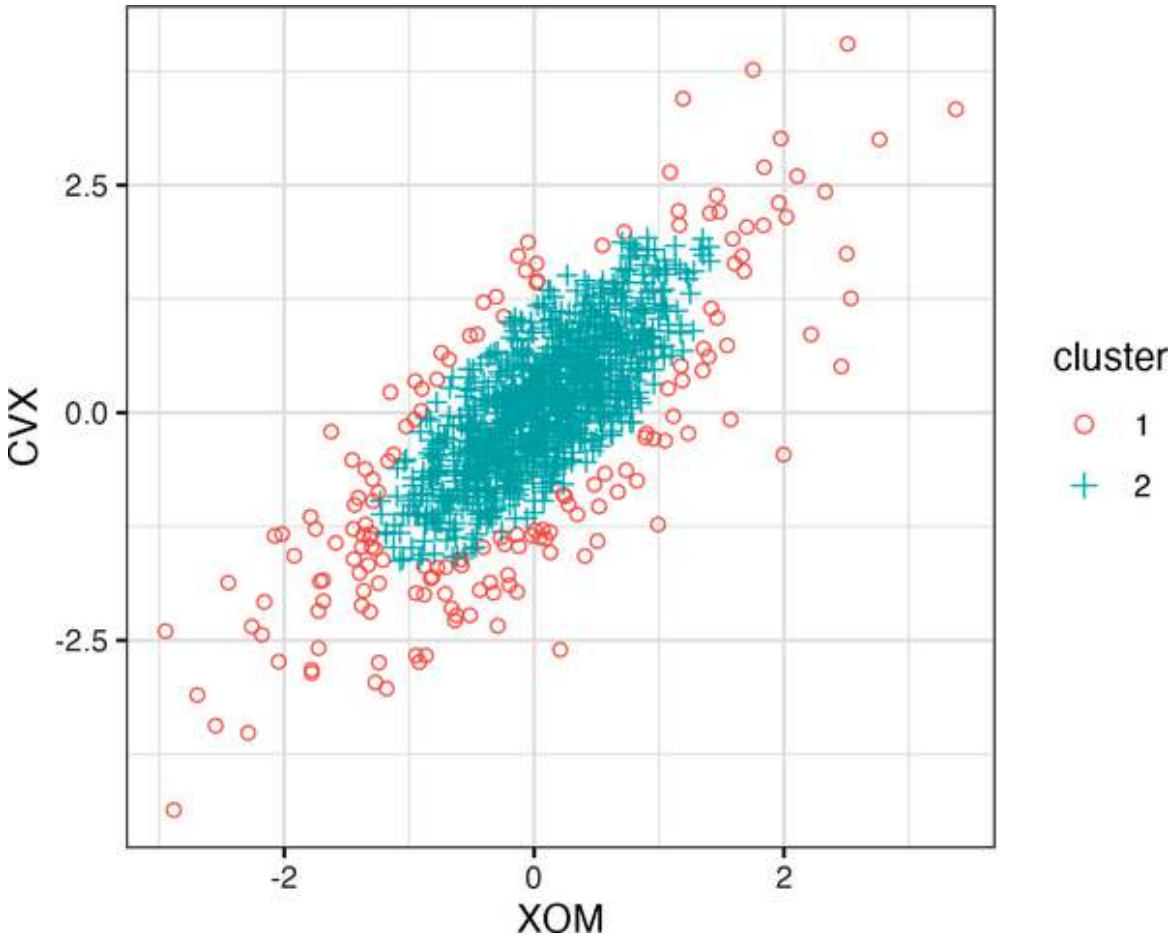


Figure 7-11. Two clusters are obtained for stock return data using `mclust`

You can extract the parameters to the normal distributions using the `summary` function:

```
summary(mcl, parameters = TRUE)$mean
---
          [,1]        [,2]
XOM 0.05783847 -0.04374944
CVX 0.07363239 -0.21175715
summary(mcl, parameters = TRUE)$variance
---
, , 1
          XOM       CVX
XOM 0.3002049 0.3060989
CVX 0.3060989 0.5496727
, , 2

          XOM       CVX
```

```
XOM 1.046318 1.066860
CVX 1.066860 1.915799
```

In Python, you get this information from the `means_` and `covariances_` properties of the result:

```
print("Mean")
print(mclust.means_)
print("Covariances")
print(mclust.covariances_)
```

The distributions have similar means and correlations, but the second distribution has much larger variances and covariances. Due to the randomness of the algorithm, results can vary slightly between different runs.

The clusters from `mclust` may seem surprising, but in fact, they illustrate the statistical nature of the method. The goal of model-based clustering is to find the best-fitting set of multivariate normal distributions. The stock data appears to have a normal-looking shape: see the contours of Figure 7-10. In fact, though, stock returns have a longer-tailed distribution than a normal distribution. To handle this, `mclust` fits a distribution to the bulk of the data but then fits a second distribution with a bigger variance.

Selecting the Number of Clusters

Unlike *K*-means and hierarchical clustering, `mclust` automatically selects the number of clusters in R (in this case, two). It does this by choosing the number of clusters for which the *Bayesian information criterion* (*BIC*) has the largest value (BIC is similar to AIC; see "Model Selection and Stepwise Regression"). BIC works by selecting the best-fitting model with a penalty for the number of parameters in the model. In the case of model-based clustering, adding more clusters will always improve the fit at the expense of introducing additional parameters in the model.

BIC IN MCLUST

Note that in most cases BIC is usually minimized. The authors of the `mclust` package decided to define BIC to have the opposite sign to make interpretation of plots easier.

`mclust` fits 14 different models with increasing number of components and chooses an optimal model automatically. You can plot the BIC values of these models using a function in `mclust`:

```
plot(mcl, what = "BIC", ask = FALSE)
```

The number of clusters—or number of different multivariate normal models (components)—is shown on the x-axis (see Figure 7-12).

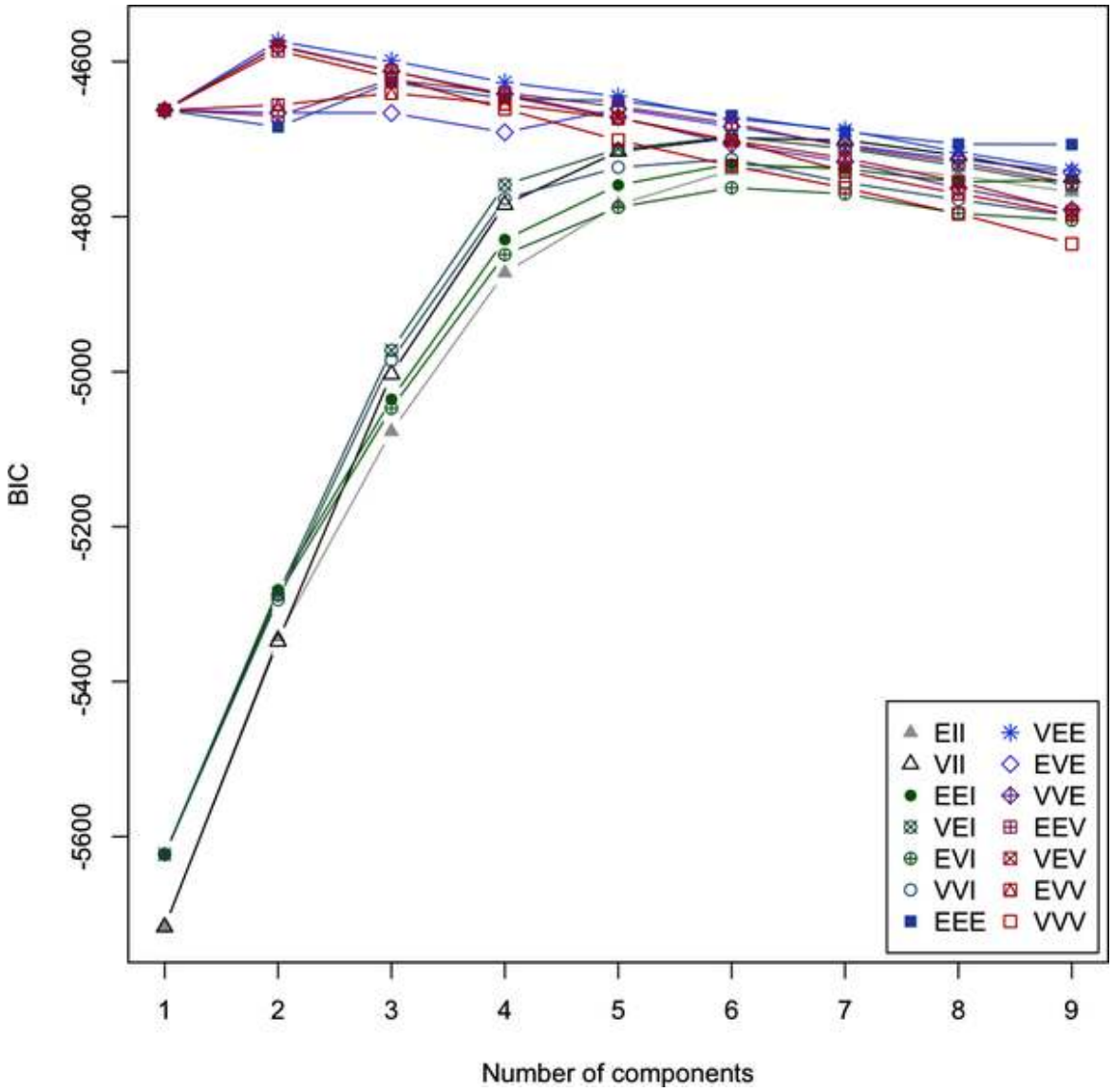


Figure 7-12. BIC values for 14 models of the stock return data with increasing numbers of components

The `GaussianMixture` implementation, on the other hand, will not try out various combinations. As shown here, it is straightforward to run multiple combinations using Python. This implementation defines BIC as usual. Therefore, the calculated BIC value will be positive, and we need to minimize it:

```
results = []
covariance_types = ["full", "tied", "diag", "spherical"]
for n_components in range(1, 9):
    for covariance_type in covariance_types:
```

```
            mclust = GaussianMixture(n_components=n_components, warm_start=True,
                                     covariance_type=covariance_type) ❶
            mclust.fit(df)
            results.append({
                "bic": mclust.bic(df),
                "n_components": n_components,
                "covariance_type": covariance_type,
            })

results = pd.DataFrame(results)

colors = ["C0", "C1", "C2", "C3"]
styles = ["C0-", "C1:", "C0-.", "C1--"]

fig, ax = plt.subplots(figsize=(4, 4))
for i, covariance_type in enumerate(covariance_types):
    subset = results.loc[results.covariance_type == covariance_type, :]
    subset.plot(x="n_components", y="bic", ax=ax, label=covariance_type,
                kind="line", style=styles[i])
```

❶ With the `warm_start` argument, the calculation will reuse information from the previous fit. This will speed up the convergence of subsequent calculations.

This plot is similar to the elbow plot used to identify the number of clusters to choose for *K*-means, except the value being plotted is BIC instead of percent of variance explained (see Figure 7-7). One big difference is that instead of one line, `mclust` shows 14 different lines! This is because `mclust` is actually fitting 14 different models for each cluster size, and ultimately it chooses the best-fitting model. `GaussianMixture` implements fewer approaches, so the number of lines will be only four.

Why does `mclust` fit so many models to determine the best set of multivariate normals? It's because there are different ways to parameterize the covariance matrix Σ for fitting a model. For the most part, you do not need to worry about the details of the models and can simply use the model chosen by `mclust`. In this example, according to BIC, three different models (called VEE, VEV, and VVE) give the best fit using two components.

MODEL-BASED CLUSTERING

Model-based clustering is a rich and rapidly developing area of study, and the coverage in this text spans only a small part of the field. Indeed, the `mclust` help file is currently 154 pages long. Navigating the nuances of model-based clustering is probably more effort than is needed for most problems encountered by data scientists.

Model-based clustering techniques do have some limitations. The methods require an underlying assumption of a model for the data, and the cluster results are very dependent on that assumption. The computations requirements are higher than even hierarchical clustering, making it difficult to scale to large data. Finally, the algorithm is more sophisticated and less accessible than that of other methods.

KEY IDEAS

- Clusters are assumed to derive from different data-generating processes with different probability distributions.
- Different models are fit, assuming different numbers of (typically normal) distributions.
- The method chooses the model (and the associated number of clusters) that fits the data well without using too many parameters (i.e., overfitting).

Further Reading

For more details on model-based clustering, see the `mclust` and `GaussianMixture` documentation.

Scaling and Categorical Variables

Unsupervised learning techniques generally require that the data be appropriately scaled. This is different from many of the techniques for regression and classification in which scaling is not important (an exception is KNN; see "K-Nearest Neighbors").

KEY TERMS FOR SCALING DATA

Scaling

Squashing or expanding data, usually to bring multiple variables to the same scale

Normalization

One method of scaling—subtracting the mean and dividing by the standard deviation

Synonym

standardization

Gower's distance

A scaling algorithm applied to mixed numeric and categorical data to bring all variables to a 0–1 range

For example, with the personal loan data, the variables have widely different units and magnitude. Some variables have relatively small values (e.g., number of years employed), while others have very large values (e.g., loan amount in dollars). If the data is not scaled, then the PCA, *K*-means, and other clustering methods will be dominated by the variables with large values and ignore the variables with small values.

Categorical data can pose a special problem for some clustering procedures. As with KNN, unordered factor variables are generally converted to a set of binary (0/1) variables using one-hot encoding (see "One-Hot Encoder").

Not only are the binary variables likely on a different scale from other data, but the fact that binary variables have only two values can prove problematic with techniques such as PCA and *K*-means.

Scaling the Variables

Variables with very different scale and units need to be normalized appropriately before you apply a clustering procedure. For example, let's apply `kmeans` to a set of data of loan defaults without normalizing:

```
loan_data <- read_csv("loan_data.csv.gz", show_col_types = FALSE) %>%
  mutate(
    across(where(is.character), as.factor),
    outcome = factor(outcome, levels = c("paid off", "default"))
  )
defaults <- loan_data %>%
  filter(outcome == "default") %>%
  select(c(loan_amnt, annual_inc, revol_bal, open_acc, dti, revol_util))
km <- kmeans(defaults, centers = 4, nstart = 10)
centers <- data.frame(size = km$size, km$centers)
round(centers, digits = 2)
---
   size loan_amnt annual_inc revol_bal open_acc   dti revol_util
1    52  22570.19  489783.40  85161.35    13.33  6.91      59.65
2  1192  21856.38  165473.54  38935.88    12.61 13.48      63.67
3 13902  10606.48   42500.30  10280.52     9.59 17.71      58.11
4  7525  18282.25   83458.11  19653.82    11.66 16.77      62.27
```

Here is the corresponding Python code:

```
loan_data = pd.read_csv("loan_data.csv.gz")

defaults = loan_data.loc[loan_data["outcome"] == "default",]
columns = ["loan_amnt", "annual_inc", "revol_bal", "open_acc",
           "dti", "revol_util"]

df = defaults[columns]
kmeans = KMeans(n_clusters=4, random_state=1).fit(df)
counts = Counter(kmeans.labels_)

centers = pd.DataFrame(kmeans.cluster_centers_, columns=columns)
centers["size"] = [counts[i] for i in range(4)]
centers
```

The variables `annual_inc` and `revol_bal` dominate the clusters, and the clusters have very different sizes. Cluster 1 has only 52 members with comparatively high income and revolving credit balance.

A common approach to scaling the variables is to convert them to *z*-scores by subtracting the mean and dividing by the standard deviation. This is termed *standardization* or *normalization* (see "Standardization (Normalization, z-Scores)" for more discussion about using *z*-scores):

$$z = \frac{x - x}{s}$$

See what happens to the clusters when `kmeans` is applied to the normalized data:

```
df0 <- scale(df)
km0 <- kmeans(df0, centers = 4, nstart = 10)
centers0 <- scale(km0$centers, center = FALSE,
  scale = 1 / attr(df0, "scaled:scale"))
centers0 <- scale(centers0, center = -attr(df0, "scaled:center"),
  scale = FALSE)
centers0 <- data.frame(size = km0$size, centers0)
round(centers0, digits = 2)
---
  size loan_amnt annual_inc revol_bal open_acc   dti revol_util
1 7355  10467.65   51134.87  11523.31     7.48 15.78      77.73
2 5309  10363.43   53523.09   6038.26     8.68 11.32      30.70
3 3713  25894.07  116185.91  32797.67    12.41 16.22      66.14
4 6294  13361.61   55596.65  16375.27    14.25 24.23      59.61
```

In Python, we can use `scikit-learn`'s `StandardScaler`. The `inverse_transform` method allows converting the cluster centers back to the original scale:

```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
df0 = scaler.fit_transform(df * 1.0)

kmeans = KMeans(n_clusters=4, random_state=1).fit(df0)
counts = Counter(kmeans.labels_)
```

```
centers = pd.DataFrame(scaler.inverse_transform(kmeans.cluster_centers_),
                       columns=columns)
centers["size"] = [counts[i] for i in range(4)]
centers
```

The cluster sizes are more balanced, and the clusters are not dominated by `annual_inc` and `revol_bal`, revealing more interesting structure in the data. Note that the centers are rescaled to the original units in the preceding code. If we had left them unscaled, the resulting values would be in terms of *z*-scores and would therefore be less interpretable.

SCALING IN PCA

Scaling is also important for PCA. Using the *z*-scores is equivalent to using the correlation matrix (see "Correlation") instead of the covariance matrix in computing the principal components. Software to compute PCA usually has an option to use the correlation matrix (in R, the `princomp` function has the argument `cor`).

Dominant Variables

Even in cases where the variables are measured on the same scale and accurately reflect relative importance (e.g., movement to stock prices), it can sometimes be useful to rescale the variables.

Suppose we add Google (GOOGL) and Amazon (AMZN) to the analysis in "Interpreting Principal Components". We see how this is done in R below:

```
syms <- c("GOOGL", "AMZN", "AAPL", "MSFT", "CSCO", "INTC", "CVX", "XOM",
"SLB",
  "COP", "JPM", "WFC", "USB", "AXP", "WMT", "TGT", "HD", "COST")
top_sp1 <- sp500_px %>%
  filter(Date >= "2011-01-01") %>%
  select(all_of(syms))
sp_pca1 <- princomp(top_sp1)
screeplot(sp_pca1)
```

In Python, we get the scree plot as follows:

```
syms = ["GOOGL", "AMZN", "AAPL", "MSFT", "CSCO", "INTC", "CVX", "XOM",
        "SLB", "COP", "JPM", "WFC", "USB", "AXP", "WMT", "TGT", "HD", "COST"]
top_sp1 = sp500_px.loc[sp500_px.index >= "2011-01-01", syms]

sp_pca1 = PCA()
sp_pca1.fit(top_sp1)

explained_variance = pd.DataFrame(sp_pca1.explained_variance_)
ax = explained_variance.head(10).plot.bar(legend=False, figsize=(4, 4))
ax.set_xlabel("Component")
```

The scree plot displays the variances for the top principal components. In this case, the scree plot in Figure 7-13 reveals that the variances of the first and second components are much larger than the others. This often indicates that one or two variables dominate the loadings. This is, indeed, the case in this example:

```
round(sp_pca1$loadings[, 1:2], 3)
---
      Comp.1 Comp.2
GOOGL  0.781  0.609
AMZN   0.593 -0.792
AAPL   0.078  0.004
MSFT   0.029  0.002
CSCO   0.017 -0.001
INTC   0.020 -0.001
CVX    0.068 -0.021
XOM    0.053 -0.005
...
```

In Python, we use the following:

```
loadings = pd.DataFrame(sp_pca1.components_[0:2, :], columns=top_sp1.columns)
loadings.transpose()
```

The first two principal components are almost completely dominated by GOOGL and AMZN. This is because the stock price movements of GOOGL and AMZN dominate the variability.

To handle this situation, you can either include them as is, rescale the variables (see "Scaling the Variables"), or exclude the dominant variables

from the analysis and handle them separately. There is no “correct” approach, and the treatment depends on the application.

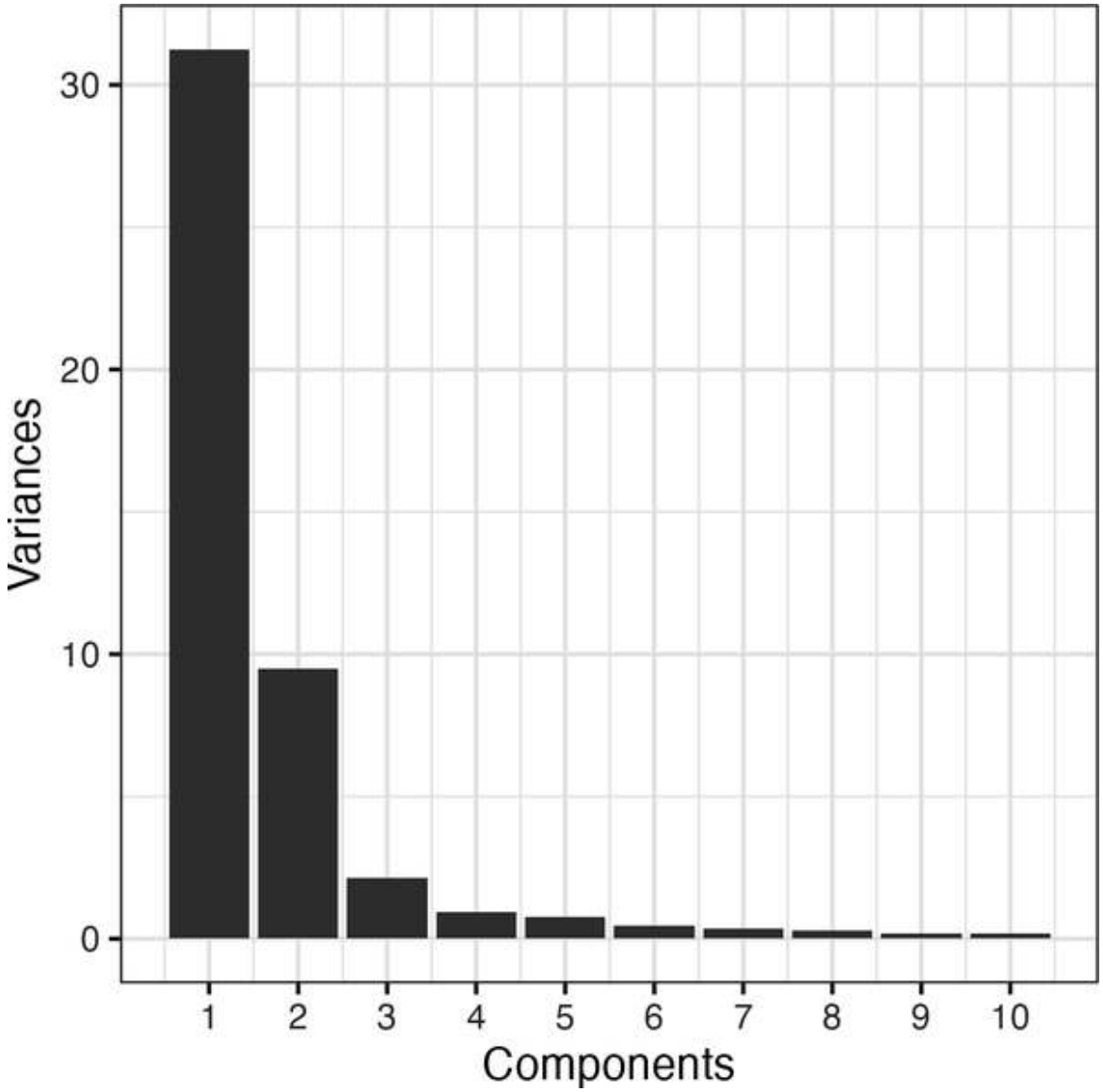


Figure 7-13. A scree plot for a PCA of top stocks from the S&P 500, including GOOGL and AMZN

Categorical Data and Gower’s Distance

In the case of categorical data, you must convert it to numeric data, either by ranking (for an ordered factor) or by encoding as a set of binary (dummy) variables. If the data consists of mixed continuous and binary variables, you will usually want to scale the variables so that the ranges are

similar; see "Scaling the Variables". One popular method is to use *Gower's distance*.

The basic idea behind Gower's distance is to apply a different distance metric to each variable depending on the type of data:

- For numeric variables and ordered factors, distance is calculated as the absolute value of the difference between two records (*Manhattan distance*).
- For categorical variables, the distance is 1 if the categories between two records are different, and the distance is 0 if the categories are the same.

Gower's distance is computed as follows:

1. Compute the distance $d_{i,j}$ for all pairs of variables i and j for each record.
2. Scale each pair $d_{i,j}$ so the minimum is 0 and the maximum is 1.
3. Add the pairwise scaled distances between variables together, using either a simple or a weighted mean, to create the distance matrix.

To illustrate Gower's distance, take a few rows from the loan data in R:

```
x <- loan_data[1:5, c("dti", "payment_inc_ratio", "home_", "purpose_")]
x
---
    dti payment_inc_ratio home_           purpose_
1  1.00           2.39320  RENT     major_purchase
2  5.55           4.57170   OWN     small_business
3 18.08           9.71600  RENT              other
4 10.08          12.21520  RENT debt_consolidation
5  7.06           3.90888  RENT              other
```

The function `daisy` in the `cluster` package in R can be used to compute Gower's distance:

```
library(cluster)
daisy(x, metric = "gower")
---
Dissimilarities :
          1         2         3         4
2 0.6220479
3 0.6863877 0.8143398
4 0.6329040 0.7608561 0.4307083
5 0.3772789 0.5389727 0.3091088 0.5056250

Metric :  mixed ;  Types = I, I, N, N
Number of objects : 5
```

In Python, the `gower` package can calculate the Gower distance:

```
import gower

X = defaults[["dti", "payment_inc_ratio", "home_", "purpose_"]].head(5)
gower.gower_matrix(X, cat_features=[False, False, True, True])  ❶
```

❶ The `cat_features` argument specifies which columns are categorical.

All distances are between 0 and 1. The pair of records with the biggest distance is 2 and 3: neither has the same values for `home` and `purpose`, and they have very different levels of `dti` (debt-to-income) and `payment_inc_ratio`. Records 3 and 5 have the smallest distance because they share the same values for `home` and `purpose`.

You can pass the Gower's distance matrix calculated from `daisy` to `hclust` for hierarchical clustering (see "Hierarchical Clustering"):

```
set.seed(1234)
defaults <- loan_data %>%
  filter(outcome == "default")
df <- defaults[sample(nrow(defaults), 250),
  c("dti", "payment_inc_ratio", "home_", "purpose_")]
d <- daisy(df, metric = "gower")
hcl <- hclust(d)
dnd <- as.dendrogram(hcl)
plot(dnd, leaflab = "none", ylab = "distance")
```

The resulting dendrogram is shown in Figure 7-14.

For Python, we can use `scipy` to perform hierarchical clustering on the Gower distance matrix:

```
from scipy.spatial.distance import squareform

X = defaults[["dti", "payment_inc_ratio", "home_", "purpose_"]].sample(250,
  random_state=1)
D = gower.gower_matrix(X, cat_features=[False, False, True, True])
condensed_D = squareform(D)  ❶
Z = linkage(condensed_D, method="complete")

fig, ax = plt.subplots(figsize=(5, 5))
dendrogram(Z, labels=list(X.index), ax=ax, color_threshold=0.5)
ax.axhline(y=0.5, color="red", linestyle="--")
ax.set_xticks([])
ax.set_ylabel("distance")
```

❶ The output of `gower_matrix` is a square symmetric distance matrix, but `linkage` requires a condensed distance matrix; this means a flattened one-dimensional array of the upper triangle. The `squareform` function from `scipy` converts to this format.

The individual records are not distinguishable on the x-axis, but we can cut the dendrogram horizontally at 0.5 and examine the records in one of the subtrees with this code:

```
dnd_cut <- cut(dnd, h = 0.5)
df[labels(dnd_cut$lower[[1]]), ]
---
# A tibble: 20 × 4
     dti payment_inc_ratio home_ purpose_
   <dbl>             <dbl> <fct> <fct>
 1  3.61              3.60  RENT  other
 2  0                 0.711 RENT  other
 3 25.5              14.0   RENT  other
 4 28.4              12.8   RENT  other
 5 25.4               4.06  RENT  other
 6 25.2               6.35  RENT  other
 7 27.5               9.75  RENT  other
 8 30.1               6.61  RENT  other
```

```
 9 16.4             12.0   RENT  other
10 15.4             16.5   RENT  other
11 20.4              7.04  RENT  other
12 21.3              1.91  RENT  other
13 19.7              4.31  RENT  other
14 17.0              4.04  RENT  other
15 14.5              2.58  RENT  other
16 11.8              1.98  RENT  other
17 10.0              2.58  RENT  other
18  9.87             4.55  RENT  other
19 16.0              5.27  RENT  other
20 13.9              5.26  RENT  other
```

This subtree consists entirely of renters with a loan purpose labeled as "other." While strict separation is not true of all subtrees, this illustrates that the categorical variables tend to be grouped together in the clusters.

The equivalent Python code to extract the records in one of the subtrees is as follows:

```
memb = fcluster(Z, t=0.5, criterion="distance")
X[memb == 1].head(20)
```

Due to the random sampling, the subtree will differ from those in R.

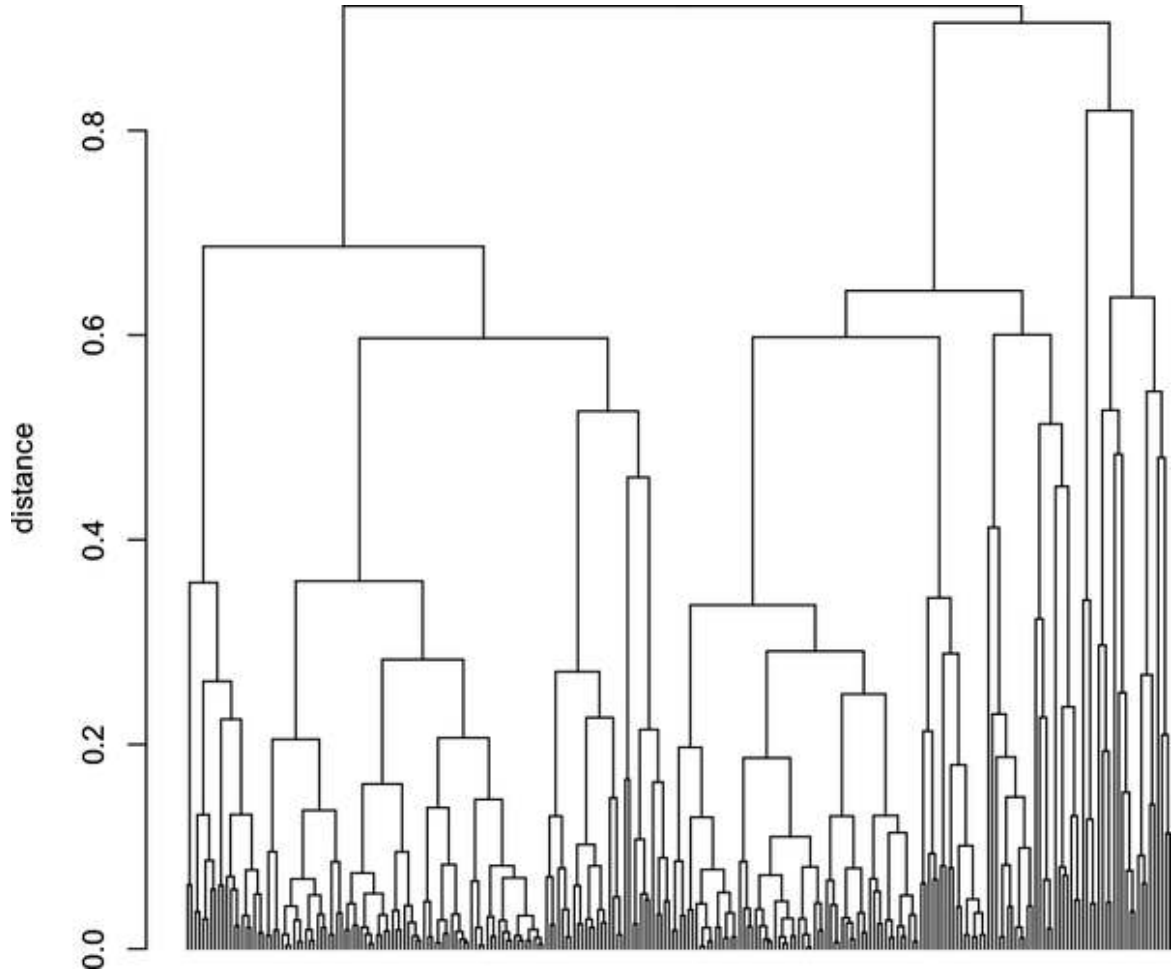


Figure 7-14. A dendrogram of `hclust` applied to a sample of loan default data with mixed variable types

Problems with Clustering Mixed Data

K-means and PCA are most appropriate for continuous variables. For smaller data sets, it is better to use hierarchical clustering with Gower's distance. In principle, there is no reason why *K*-means can't be applied to binary or categorical data. You would usually use the "one-hot encoder" representation (see "One-Hot Encoder") to convert the categorical data to numeric values. In practice, however, using *K*-means and PCA with binary data can be difficult.

If the standard *z*-scores are used, the binary variables will dominate the definition of the clusters. This is because 0/1 variables take on only two values, and *K*-means can obtain a small within-cluster sum-of-squares by

assigning all the records with a 0 or 1 to a single cluster. For example, apply `kmeans` to loan default data including factor variables `home` and `pub_rec_zero`, shown here in R:

```
df <- model.matrix(~ -1 + dti + payment_inc_ratio + home_ + pub_rec_zero,
  data = defaults)
df0 <- scale(df)
km0 <- kmeans(df0, centers = 4, nstart = 10)
centers0 <- scale(km0$centers, center = FALSE,
  scale = 1 / attr(df0, "scaled:scale"))
round(scale(centers0, center = - attr(df0, "scaled:center"), scale = FALSE),
2)
```

```
---
    dti payment_inc_ratio home_MORTGAGE home_OWN home_RENT pub_rec_zero
1 17.20              9.27          0.00        1      0.00         0.92
2 16.99              9.11          0.00        0      1.00         1.00
3 16.50              8.06          0.52        0      0.48         0.00
4 17.46              8.42          1.00        0      0.00         1.00
```

In Python:

```
columns = ["dti", "payment_inc_ratio", "home_", "pub_rec_zero"]
df = pd.get_dummies(defaults[columns])

scaler = preprocessing.StandardScaler()
df0 = scaler.fit_transform(df * 1.0)
kmeans = KMeans(n_clusters=4, random_state=1).fit(df0)
centers = pd.DataFrame(scaler.inverse_transform(kmeans.cluster_centers_),
                       columns=df.columns)
centers
```

The top four clusters are essentially proxies for the different levels of the factor variables. To avoid this behavior, you could scale the binary variables to have a smaller variance than other variables. Alternatively, for very large data sets, you could apply clustering to different subsets of data taking on specific categorical values. For example, you could apply clustering separately to those loans made to someone who has a mortgage, owns a home outright, or rents.

KEY IDEAS

- Variables measured on different scales need to be transformed to similar scales so that their impact on algorithms is not determined mainly by their scale.
- A common scaling method is normalization (standardization)—subtracting the mean and dividing by the standard deviation.
- Another method is Gower's distance, which scales all variables to the 0–1 range (it is often used with mixed numeric and categorical data).

Returning to AI

Think of the analysis we did in Chapter 1 regarding the relationship between tax-assessed value and finished square feet. We created several visualizations, including Figure 1-8 and Figure 1-12. We asked AI to comment on the visualizations, and we did some further analysis in Chapter 4. Now, let's ask AI for suggestions on what to do next:

Prompt:	Make a suggestion on what to do next.
ChatGPT:	A good next step is to explicitly model the market segmentation suggested by the multiple clusters. Practically, you could (1) stratify the data by neighborhood, zoning class, or location-based proxies if available, and replot or refit the size–value relationship within each stratum; or (2) apply an unsupervised method (e.g., Gaussian mixture models or k-means on log(square footage) and log(value)) to formally identify clusters. Then, fit separate regression models for each segment and compare them to a single global model. This will test whether the apparent clusters represent distinct submarkets and quantify how much explanatory power is gained by accounting for them.

These are all reasonable suggestions worth following up on.

Summary

For dimension reduction of numeric data, the main tools are either principal components analysis or *K*-means clustering. Both require attention to proper scaling of the data to ensure meaningful data reduction.

For clustering with highly structured data in which the clusters are well separated, all methods will likely produce a similar result. Each method offers its own advantage. *K*-means scales to very large data and is easily understood. Hierarchical clustering can be applied to mixed data types—numeric and categorical—and lends itself to an intuitive display (the dendrogram). Model-based clustering is founded on statistical theory and provides a more rigorous approach, as opposed to the heuristic methods. For very large data, however, *K*-means is the main method used.

With noisy data, such as the loan and stock data (and much of the data that a data scientist will face), the choice is more stark. *K*-means, hierarchical clustering, and especially model-based clustering all produce very different solutions. How should a data scientist proceed? Unfortunately, there is no simple rule of thumb to guide the choice. Ultimately, the method used will depend on the data size and the goal of the application.

Exploration with AI

1. Ask your AI to implement suggestion #2 in "Returning to AI". That is, use a clustering method to identify clusters in the housing data based on log(square footage) and log(tax-assessed value). Explore with the AI (a) the advantages/disadvantages of different clustering methods, (b) selecting different numbers of clusters to identify, and (c) descriptions of the different clusters found. Also experiment with your prompt—see if you can narrow down the AI's response and reduce its verbosity.
2. Repeat the analysis of "Analyzing the Clusters with AI" and follow up with a request to the AI to criticize its earlier suggestions. How does the AI respond? Do you agree with the criticism? Explore what could be done to improve the analysis.

---

[1] This and subsequent sections in this chapter © 2020–2025 Datastats, LLC, Peter Bruce, Andrew Bruce, and Peter Gedeck; used with permission.

Chapter 8. Neural Networks

Neural networks are complex and flexible predictive models that take input and, via a complex set of paths and intermediate calculations, produce output. This chapter introduces the basic structure and functioning of neural networks. Later chapters cover the larger and more complex networks that are now used (Chapter 9), as well as the models that are used in generative AI (Chapter 10).

The Basic Structure of a Neural Net

Neural nets drew their inspiration from abstract models of the human brain in which interconnected neurons process information. The basic structure of a neural network algorithm was established by F. Rosenblatt in his paper "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain" (1958). He posited an input layer, a hidden layer, and an output layer. Neural nets saw early application in the late 1990s but did not take off until the early 2000s with the development of more complex networks and the growing power of computers.

KEY TERMS FOR NEURAL NETWORKS

Node

A computational unit that receives input data, performs a calculation with it, and produces output

Synonym

neuron

Layer

A set of nodes at the same level in a network

Input layer

The first layer of a neural network that receives the input data

Output layer

The last layer of a neural network that produces the output data

Hidden layer

A layer lying after the input layer, but before the output layer

The original standard neural network architecture is the multilayer feedforward network, which has an input layer consisting of nodes (sometimes called neurons) that accept the predictor values, and then successive layers of nodes that receive input from the previous layers. The final layer is the output layer. Layers between the input and output layers are known as hidden layers. Current deep learning architectures often use more complex structures; more on this in the next chapter. Figure 8-1 illustrates a simple neural network with two hidden layers and one node in the output layer representing the outcome value to be predicted. In this *fully connected* network, each node in a layer is connected to every node in the next layer (while this is not true for all neural networks, in the discussion that follows we assume a fully connected network for simplicity).

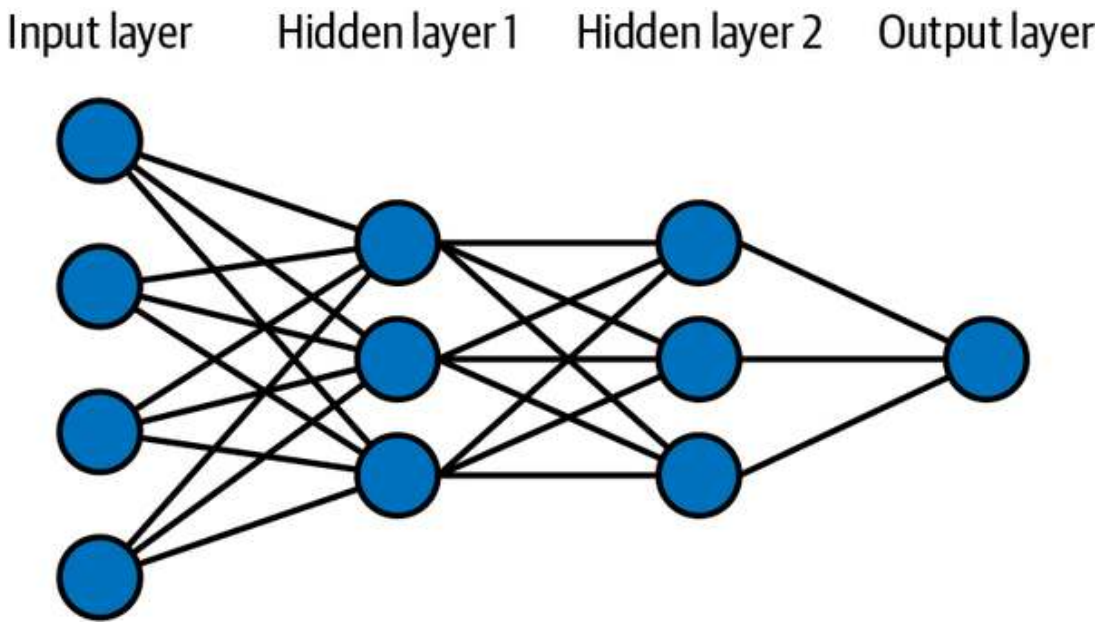


Figure 8-1. Simple neural network

The number of nodes in the input layer is typically determined by the number of predictors (features). Multiple output nodes are used when there are multiple classes or multiple outcomes to predict. The number of hidden layers and the number of nodes in each hidden layer are hyperparameters that must be set by the user (see "Hyperparameters" in Chapter 6). The weights and biases that govern calculations within the network are learned from the data. More layers and more nodes increase the flexibility of the model, but, without regularization (see Chapter 9), also increase the risk of overfitting the training data.

Today's neural networks, with billions of parameters, are orders of magnitude larger than the models we have introduced earlier in this book. Still, the neurons and connective structure of the human brain are far more complex than any artificial neural network, whose comparatively simple design is optimized for computational efficiency.

Neural networks are capable of capturing very complex relationships in data. They lie at the heart of the AI revolution in machine learning, moving from success in text and image classification to the generation of text, images, and video.

The concept underlying neural networks is to combine predictor information in a very flexible way that captures complicated relationships among predictors, and between them and an outcome variable. In Chapter 4, we reviewed linear regression, where the relationship between the predictors and the outcome is assumed to be linear. We were able to generalize the relationship using nonlinear

transformations such as spline functions, as well as interactions between variables. Chapter 6 introduced decision trees, which produce intuitive and easy-to-grasp decision rules. Refinements such as random forests and boosted trees offer superior predictive performance, at the cost of losing interpretable decision rules. Trees have the ability to discover nonlinear and complex relationships in the data. Neural networks, which also have this ability, take a very different approach, which is less intuitive but has turned out to be more flexible and powerful.

KEY IDEAS

- Neural networks take input (predictor values) and, via a complex set of paths and intermediate calculations determined by training the network on data, produce output.

Further Reading

The classic original reference on neural networks is Rosenblatt's 1958 paper on the perceptron (Rosenblatt, 1958). It was published in *Psychological Review*; a 1957 precursor, also by Rosenblatt, is available in this Cornell Aeronautical Laboratory Report.

Fitting a Network to Data

How does this structure manage to produce accurate predictions? Recall our study of linear regression, and the study of how lung capacity was related to exposure to cotton dust (see Figure 4-1 and Figure 4-2). We found the linear regression line by minimizing the sum of squared residuals—a mathematical calculation. We could also have found the line by repeatedly moving the line up or down, or changing its slope, and seeing if the sum of squared residuals got smaller. If it did, we would keep moving in that direction until we could no longer reduce the sum of squared residuals. This is called *gradient descent*, which is at the core of machine learning algorithms such as boosted trees, and is a common way to train neural networks to data. The "gradient" is the slope of the loss function with respect to changes in a parameter; it is discussed in more detail in the following.

KEY TERMS FOR FITTING A NETWORK TO DATA

Weight

Multiplier of a value passed from node to node in a neural net

Bias

A constant added to the weighted sum of inputs to a node in a neural net

Activation function

A mathematical function applied to the weighted sum of inputs to a node to produce the node's output

Rectified linear unit activation function

A piecewise linear function that outputs zero (for negative input), and outputs the input itself (for positive input)

Synonym

ReLU

Sigmoid activation function

A squashing function that maps input to a 0–1 range

Softmax activation function

A function that converts a vector of real-valued scores into a normalized probability distribution that sums to one, making it useful for representing class probabilities in classification models.

In OLS regression, a small number of parameters (the intercept and slopes) sum up the linear relationship between predictors and outcome. A neural network is not limited to a linear relationship, and has a great many more parameters to estimate.

Depending on the task, the input values can be raw data:

- Numeric or categorical values (for columnar data)

- Sequence of values over time (for time-series data)
- Pixel values (for image data)
- Numerical representations of words (for text data)

Inputs are also commonly derived from the raw data based on knowledge of underlying mechanisms or aggregation of the data using techniques such as KNN, temporal, or spatial aggregation summaries of data. These transformations of the data, known as feature engineering, are designed to capture known relationships and correlations. When used as input, they can improve the performance of neural nets by alleviating the need for the model to learn these known relationships.

Weights and Biases

The input values are passed to the first hidden layer, whose nodes have *weights* that are applied to each value from the prior layer. Each node in a layer takes the values from the nodes in the previous layer, multiplies them by the weights, sums them up, and then applies a function to that sum to produce a local output from that node. Each node has its own *bias* value that acts as a constant. Why have both weights and biases? The weights determine the influence of each input to a node on that node's output, while the bias allows the node to adjust the output independently of the input values. Compare this to a linear regression, where the fitting of a line is determined by coefficients (weights) and an intercept (bias). Without the intercept, the fit would not be as good, since the line would be anchored at the origin and only be able to pivot around that point. This analogy extends to multiple linear regression in which the response y is a linear sum of the inputs $x_1, x_2, \ldots, x_p$ plus an intercept $y = a + b_1x_1 + b_2x_2 + \ldots + b_px_p$. The bias in a neural network corresponds to the intercept, and the weights $w_1, w_2, \ldots, w_p$ correspond to the regression coefficients. The bias, which doesn't directly depend on the inputs, acts like a "mean" term that adjusts the weighted inputs up or down.

Activation Functions in the Hidden Layers

The results of these operations are then passed on as an input to each node in the next layer, where they are treated as inputs. Each node's inputs are weighted and combined in a weighted sum, and then a function is applied to the result to produce the output of that node. This function is called an *activation function*. The flow of data in a single node is shown in Figure 8-2.

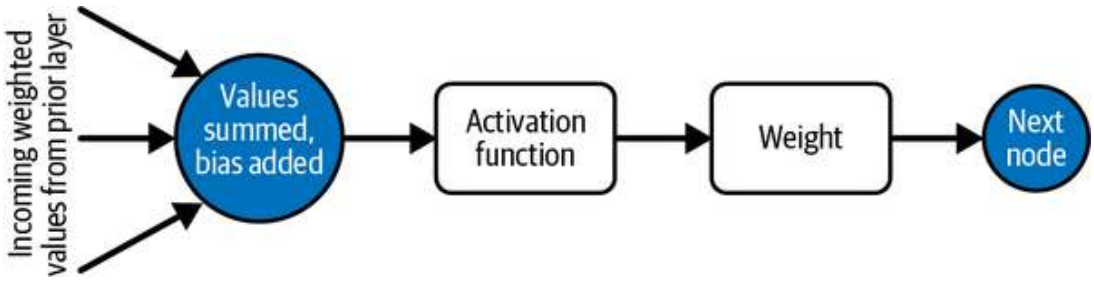


Figure 8-2. Operations at a single node

The most common hidden layer activation function is the rectified linear unit (ReLU), which is a piecewise linear function that outputs zero if the input is negative, and outputs the input itself if it is positive. ReLU functions have been used in extensions of decision trees, such as MARS (Friedman), but these never gained large-scale popularity. Nonlinear tanh and sigmoid activation functions, which squash their inputs to a narrow range, used to be quite popular and are still used in certain architectures (e.g., LSTMs, see Chapter 9), and in the final layer for classification. However, with large positive or negative inputs, their output approaches a fixed limit and changes very little, causing learning to stall. They are no longer the default hidden-layer activation in deep networks. Recall that sigmoid functions are at the heart of logistic regression, covered in Chapter 5. The ReLU, sigmoid, and tanh activation functions are illustrated in Figure 8-3.

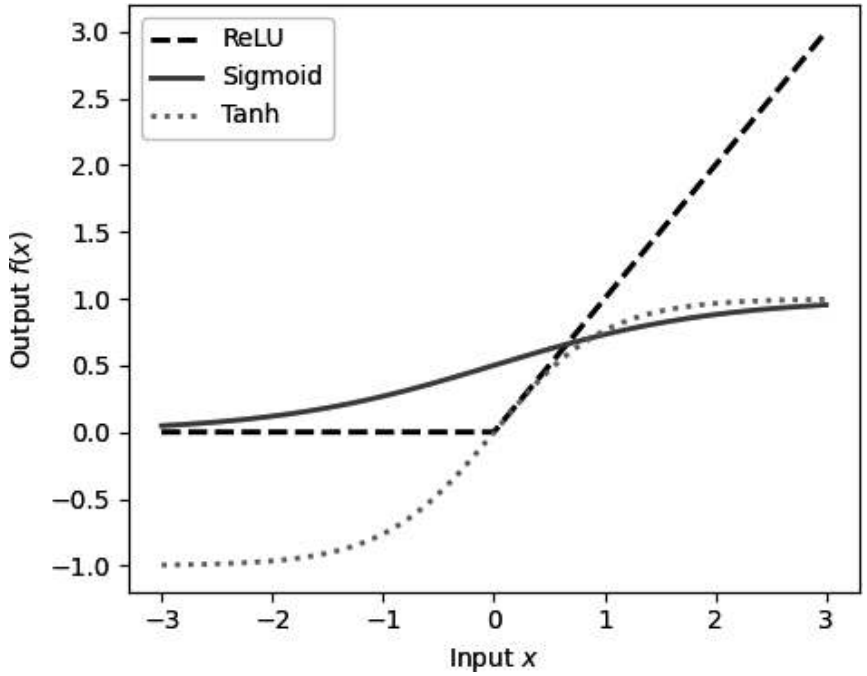


Figure 8-3. ReLU, sigmoid, and tanh activation functions

The Final Layer

Things change when you get to the final output layer. The activation function to be used depends on the output.

For regression with numeric data, the typical choices are:

- If the outcome must be nonnegative, a ReLU activation is used.
- If positive and negative values are possible, a linear activation is used (usually the output of the final hidden layer is simply passed through unchanged).

Classification tasks require probabilities, which must range from 0 to 1, and the typical choices are:

- If the outcome is binary, a *sigmoid* activation function is used, which squashes the final layer output to a value between 0 and 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- If the output is multiclass, a *softmax* activation function is used, which, given the output z_i of the final layer, produces a similarly squashed set of values between 0 and 1 that sum to 1, representing the estimated probabilities of each class:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \cdots + e^{z_K}}$$

Softmax functions are also used in multinomial logistic regression, a generalization of logistic regression to multiclass problems.

Both the sigmoid and softmax functions work with the results of the weighted sums from the final layer, also called *logits*, to produce quasi-probabilities (they can be biased as estimates of probabilities, but are useful in ranking). These are similar to the logits we encountered with logistic regression in Chapter 5. The softmax function is a generalization of the sigmoid function for multiple classes.

TREE MODELS AND NEURAL NETS

Predictions from both neural nets and tree models can be represented as a sum of many building block functions, known as *basis* functions. In tree models, the basis functions ϕ are simple binary step functions:

$$\phi(x_1, x_2, \ldots, x_p) = \begin{cases} 1 & \text{if } (x_1, x_2, \ldots, x_p) \in A \\ 0 & \text{if } (x_1, x_2, \ldots, x_p) \notin A \end{cases}$$

The notations $\in A$ or $\notin A$ mean that the observation "is in" or "is not in" a p-dimensional region. In the case where $p = 2$, A would be a rectangle (see the representation in Figure 6-4). A prediction from a decision tree $f(x_1, x_2, \ldots, x_p)$, and its descendants random forest and boosted trees, can be represented by a sum of many step functions:

$$f(x_1, x_2, \ldots, x_p) = b_1\phi_1(x_1, x_2, \ldots, x_p) + \ldots + b_M\phi_M(x_1, x_2, \ldots, x_p)$$

Likewise, a single-layer neural net can also be represented as a sum of many basis functions, such as the sigmoid and softmax functions. Indeed, the simplest activation function is also a step function, also known as a *Heaviside step* function, that produces a 0–1 output from a set of inputs:

$$\phi(x_1, x_2, \ldots, x_p) = \begin{cases} 1 & \text{if } b + w_1x_1 + w_2x_2 + \ldots + w_px_p > 0 \\ 0 & \text{if } b + w_1x_1 + w_2x_2 + \ldots + w_px_p \leq 0 \end{cases}$$

Theoretically, a tree model can be represented as a neural network with step function activations. In practice, tree models and neural nets are fit using very different algorithms, and while step functions were used in very early forms of neural networks [Rosenblatt-1958], they aren't used anymore due to the discontinuous nature of the function. The functional form of multilayer neural networks with continuous activation functions, such as ReLU, sigmoid, and tanh, is far more complex than tree models. Tree models initially gained wider usage than neural nets since they were more interpretable, simpler to understand, and faster to train. However, this complexity helps explain why modern neural networks, such as deep learning models, can outperform tree models when trained with lots of data.

The Training Process

How does the network get its initial parameter values? Essentially, randomly. The weights and biases are initialized to small random values, and then the network is trained on the data. The final layer produces the output of a single pass, which becomes the (provisional) prediction. The difference between this provisional prediction and the actual value is the error. The training process then proceeds backward through the network in a process called *backpropagation.* The weights and bias values are then adjusted, based on how much they contributed to the error. This back-and-forth cycle is repeated many times, with the weights and biases being adjusted each time, the goal being to minimize error. This is called *training* the network. The end result is a set of weights and biases that produce more accurate predictions.

A Simple Model

The packages `keras3` and `tensorflow` are a common choice for deep learning in R. These packages are native to Python, but a mature interface has been developed for their use in R. Let's revisit the loan data used in Chapter 6 to fit a simple neural network model. In contrast to the tree models, since `keras3` calls the Python library `tensorflow`, which lacks automatic conversion, factor variables need to be converted to numeric data. Prepare the data by converting all character variables first to factors and then to their numerical representation. Finally, since predictors with very different scales can lead to problems in gradient descent optimization, rescale the predictors so that they are between 0 and 1. The code for data loading and preprocessing in R is:

```
library(tidyverse)
library(tidymodels)
loan_data <- read_csv("loan_data.csv.gz", show_col_types = FALSE) %>%
  dplyr::select(-c(index, status)) %>%
  mutate(
    across(where(is.character), as.factor),
    outcome = factor(outcome, levels = c("paid off", "default")),
    across(where(is.factor), as.numeric),  ❶
  )

formula <- outcome ~ loan_amnt + term + annual_inc + dti + payment_inc_ratio +
  revol_bal + revol_util + delinq_2yrs_zero + pub_rec_zero + open_acc +
  grade + purpose_ + home_ + emp_len_ + borrower_score
rec <- recipe(formula, data = loan_data) %>%
  step_range(all_outcomes(), 0, 1) %>%
  step_range(all_predictors(), 0, 1)
xy <- rec %>%
  prep() %>%
```

```
  bake(loan_data)  ❷
train_x <- xy %>% select(-outcome) %>% as.matrix()  ❸
train_y <- xy %>% select(outcome) %>% as.matrix()
```

❶ Convert all factor variables to their numeric representation.

❷ `prep` and `bake` the recipe to produce a processed dataframe.

❸ The `keras` package requires the data to be in matrix format.

A neural network model is fit using three steps:

1. Define the model structure.
2. Define the loss functions and optimization method.
3. Train the model.

These will be explained in detail later in this chapter. Use the `tensorflow` and `keras3` packages to fit a simple neural network:

```
library(keras3)
library(tensorflow)

simple_model <- keras_model_sequential(input_shape = c(4)) %>%
  layer_dense(units = 4, activation = "relu") %>%
  layer_dense(units = 2, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

# Compile the model
simple_model %>% compile(
  loss = "binary_crossentropy",  ❶
  optimizer = optimizer_adam(learning_rate = 0.01),  ❷
  metrics = c("accuracy")
)

# Fit the model
simple_model %>% fit(
  x = train_x[, c("payment_inc_ratio", "purpose_", "dti", "borrower_score")],
  y = train_y,
  epochs = 100,
  batch_size = 128,  ❸
  verbose = 0,
)
```

❶ The loss defines the function that the model will minimize during training. For binary classification, `binary_crossentropy` is commonly used.

❷

The optimizer determines how the model updates its weights based on the loss function. `optimizer_adam` is a popular choice that adapts the learning rate during training, facilitating faster convergence.

❸ The `batch_size` specifies the number of samples that will be propagated through the network before updating the model's weights. A batch size of 128 means that the model will process 128 samples at a time before performing a weight update. See the next section for details.

Figure 8-4 shows the resulting network fit to the loan data. The weights and biases are coded (red/solid is positive, blue/dashed is negative) with line width indicating magnitude.

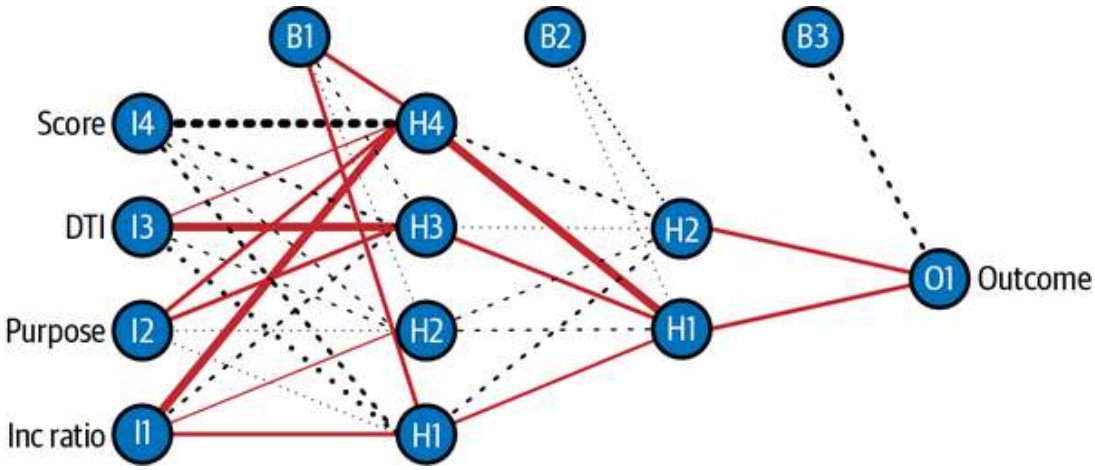


Figure 8-4. Simple neural network fit to the loan data

The equivalent code in Python to prep the data is as follows:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Read and prepare data
loan_data = (pd.read_csv("loan_data.csv.gz", index_col=0)
             .drop("status", axis=1))

# Convert to numeric ordinal
for col in ["term", "emp_len_", "home_", "purpose_"]:
    loan_data[col] = pd.Categorical(loan_data[col]).codes

# Convert outcome to numeric
loan_data["outcome"] = pd.Categorical(loan_data["outcome"],
                                      categories=["paid off", "default"],
                                      ordered=True).codes
```

```
# Define predictors
all_predictors = ["loan_amnt", "term", "annual_inc", "dti",
                  "payment_inc_ratio", "revol_bal", "revol_util",
                  "delinq_2yrs_zero", "pub_rec_zero", "open_acc",
                  "home_", "purpose_", "grade", "emp_len_",
                  "borrower_score"]

# Scale using MinMaxScaler
loan_data_scaled = loan_data.copy()
scaler = MinMaxScaler()
loan_data_scaled[all_predictors] = scaler.fit_transform(
  loan_data_scaled[all_predictors])

# Create training matrices
train_x = loan_data_scaled[all_predictors]
train_y = loan_data["outcome"]
```

Since the `keras` package is native to Python, the syntax is very similar to the code in R:

```
import keras

# Build the neural network
model = keras.Sequential([
    keras.layers.InputLayer(shape=(4,)),    # Input layer
    keras.layers.Dense(4, activation="relu"),     # First layer
    keras.layers.Dense(2, activation="relu"),     # Second layer
    keras.layers.Dense(1, activation="sigmoid"),  # Output layer
])

# Compile the model
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)

# Train the model with verbose output
history = model.fit(
    x=train_x[["payment_inc_ratio", "purpose_", "dti", "borrower_score"]],
    y=train_y,
    epochs=100,
    batch_size=128,
    verbose=0,
)
```

KEY IDEAS

- Weights and biases at each node determine the calculations performed at each node as data passes through the network.
- Activation functions are applied to the results of the calculations as they are passed to the next layer.

Further Reading

You can read more about activation functions in Activation Functions in Artificial Neural Networks: A Systematic Overview, a 2021 review by Johannes Lederer.

You can learn more about using `tensorflow` in this tutorial.

Backpropagation and Gradient Descent

After the completion of a forward pass through the network (usually involving a *batch* of cases), the error is calculated by comparing the predicted output to the actual output. Additional cases are fed through the network in like fashion, and the errors for the cases are averaged or otherwise aggregated so that weights can be updated.

KEY TERMS FOR BACKPROPAGATION AND GRADIENT DESCENT

Batch

Group of training cases fed through the network at one time, before weights are updated

Synonym

mini-batch

Stochastic gradient descent

The process of sending a single case or a small batch of cases at a time and updating weights

Loss function

A function that maps the error to a value useful for optimization

Gradient

The slope of the loss function with respect to changes in a parameter

Gradient descent

The process of adjusting parameters in the direction that reduces the loss function

Backpropagation

The algorithm for computing gradients of the loss function with respect to a network's weights by propagating error signals backward through the network

Learning rate

A hyperparameter that sets the size of weight adjustments during gradient descent

Momentum

A hyperparameter that continues the direction of previous updates to some degree

Epoch

A complete pass through the entire training data set

Overfitting

When a model is so extensively trained that it fits noise in the training data, leading to poor performance on new data

Regularization

Techniques used to prevent overfitting by constraining model complexity

The group of cases fed through the network before weight updating is called a *batch*. In the early days, the term *batch* referred to the entire training data, and *mini-batch* to a subset, but now training is mostly done with subsets, so the distinction has faded. Typically, the batch is selected randomly from the training data.

For each case in a batch, the error is calculated and a *loss function* assigns a scalar penalty; these losses are aggregated (typically by a mean or sum) to produce the value used for backpropagation. At this point, the *backpropagation* process begins.

The network then propagates backward through the layers, computing how much each parameter (weight and bias) contributes to the loss. This is done by calculating the gradient of the loss function with respect to each parameter. These gradients indicate the direction and magnitude in which the parameters should be adjusted to reduce the loss, and an optimizer applies small updates accordingly. This process is termed *gradient descent* or *stochastic gradient descent* when using mini-batches created using random sampling.

Neural nets have many settings or *hyperparameters*, such as choice of activation function, number of layers, and regularization techniques (see Chapter 6 for the hyperparameters for tree models). A key hyperparameter is the *learning rate*, which controls how big a step to take in the direction of the gradient. Recall that the learning rate is also used in boosted trees. If the learning rate is too high, the network may overshoot the optimal parameter values; if it is too low, training will take too long. A related parameter is *momentum*, which maintains to some degree

the direction of prior updates in the current update, to help prevent the model from getting stuck in local minima and to dampen oscillations.

This process is repeated for many batches of data, making parameter adjustments for each batch. Once it has passed through the entire data set, this is called an *epoch*. Once it has completed an epoch, the process starts over, and continues for many epochs, continuing to reduce the error. The number of epochs is another hyperparameter that must be set by the user, though training may be stopped early if performance on the validation fails to continue improving, according to preset criteria.

Overfitting

Neural networks are very flexible models, capable of capturing complex relationships in data. They are also good at capturing the noise in the data, so are especially prone to *overfitting* the training data. When this happens, the model performs well on the training data, but poorly on new, unseen data. To detect if the model is overfitting, neural nets are typically trained using a holdout set, also called a validation set, to monitor the model performance during training. This is a simple version of the cross-validation process we saw in Chapter 4. The training set is used to fit the model, and the validation set is used to monitor performance during training and to tune hyperparameters.

Let's fit a more complex model to the `loan_data`. Since `keras` can use the end of the data as a validation set, the data is shuffled to ensure a random allocation to training and validation sets:

```
set.seed(12345)
n <- nrow(train_x)
indices <- sample(1:n)
train_x <- train_x[indices, ]
train_y <- train_y[indices, ]
```

Now fit a neural net model with the `keras3` and `tensorflow` packages using the `relu` and `sigmoid` activation functions, 500 epochs, a batch size of 256, and holding out 10% of the data as a validation set:

```
# Create a two-hidden-layer network using the relu activation function
model <- keras_model_sequential(input_shape = c(15)) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

```
# Compile the model
model %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_adam(learning_rate = 0.005),
  metrics = c("accuracy")
)

# Fit the model using a batch size of 256 and holding out 10% of the data
set.seed(42)
history <- model %>% fit(
  x = train_x,
  y = train_y,
  epochs = 500,
  batch_size = 256,
  validation_split = 0.1,
  verbose = 0,
)
history_df <- as.data.frame(history)
```

Ideally, as model training proceeds, the errors decline on both the training and validation sets. For the loan data model, we plot the accuracy. The accuracy of the training and validation set is displayed in Figure 8-5:

```
ggplot(history_df[history_df$metric == "accuracy", ],
  aes(x = epoch, y = value, color = data)) +
  geom_line(linewidth = 0.5, alpha = 0.8) +
  geom_smooth(aes(linetype = data), method = "loess", formula = "y ~ x",
    color = "black") +
  labs(x = "Epoch", y = "Accuracy") +
  coord_cartesian(ylim = c(0.65, 0.71))
```

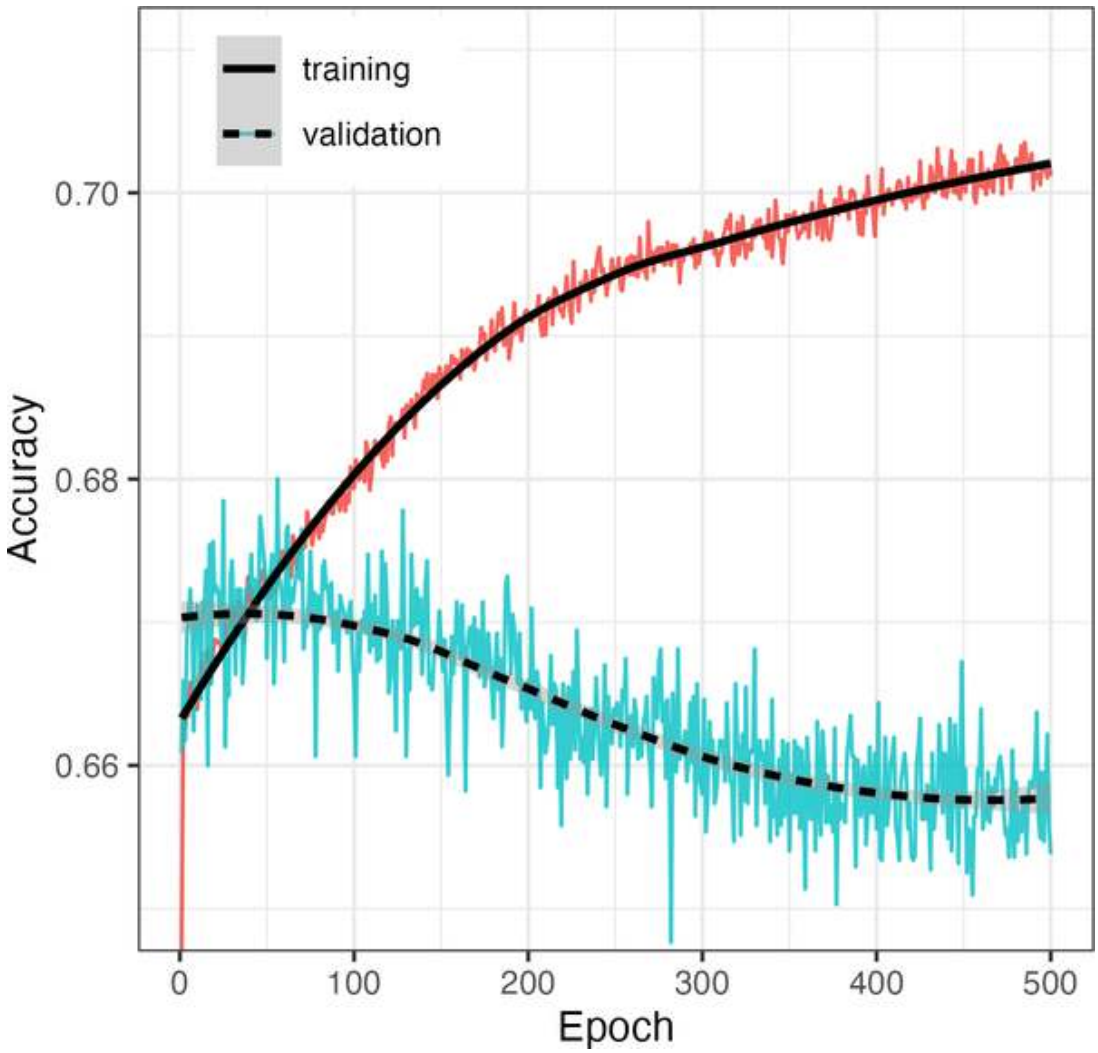


Figure 8-5. Error on training and validation sets over epochs when overfitting

The training set improves over the epochs while the validation set degrades. The solution to this is to fit the model using *regularization* techniques as discussed in the next section.

As with the previous example, the Python code is very similar to the R code:

```
n = train_x.shape[0]
indices = np.random.permutation(n)
train_x = train_x.iloc[indices]
train_y = train_y.iloc[indices]

model = keras.Sequential([
    keras.layers.InputLayer(shape=(15,)),
    keras.layers.Dense(units=64, activation="relu"),
    keras.layers.Dense(units=32, activation="relu"),
    keras.layers.Dense(units=1, activation="sigmoid"),
])
```

```
model.compile(
    loss="binary_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.005),
    metrics=["accuracy"],
)

np.random.seed(42)
history = model.fit(
    x=train_x,
    y=train_y,
    epochs=500,
    batch_size=256,
    validation_split=0.1,
    verbose=0,
)
```

Regularization

Overfitting results from complexity: large numbers of input features and/or layers and nodes. A complex structure allows for a huge number of potential models, and some of them are bound to fit the data, including noise, quite well just by chance. Such a model will not do as well with new data. Complexity can be mitigated by *regularization* techniques that penalize complexity. We introduced regularization in Chapter 6 in the context of trees (see "Regularization: Avoiding Overfitting"). Similar techniques are used with neural networks; here are several:

- *L1 regularization* adds a penalty to the loss function based on the sum of the absolute values of the weights. This encourages sparsity in the weights, effectively reducing complexity by pushing some weights to zero. This is analogous to the lasso method in regression.
- *L2 regularization* adds a penalty to the loss function based on the sum of the squares of the weights. This encourages the model to keep weights small (sometimes referred to as *weight decay*), curbing its tendency to fit noise. If any weight can be arbitrarily large, the model can easily fit noise by assigning large weights as needed to make a tight fit to the data. This is analogous to ridge regression.
- *Dropout* randomly "drops" a fraction of the nodes in the network during training. This discourages a network structure that is highly tuned to particular idiosyncrasies in the data.

Let's revisit the loan data; this time by adding L2 regularization to the hidden layers and additional dropout layers:

```
reg_model <- keras_model_sequential(input_shape = c(15)) %>%
  layer_dense(units = 64, activation = "relu",
    kernel_regularizer = regularizer_l2(0.001)) %>%  ❶
  layer_dropout(0.3) %>%  ❷
  layer_dense(units = 32, activation = "relu",
    kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_dropout(0.3) %>%
  layer_dense(units = 1, activation = "sigmoid")

reg_model %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_adam(learning_rate = 0.001),
  metrics = c("accuracy")
)
set.seed(42)
history_reg <- reg_model %>% fit(
  x = train_x,
  y = train_y,
  epochs = 500,
  batch_size = 256,
  validation_split = 0.1,
  verbose = 0
)
history_reg_df <- as.data.frame(history_reg)
```

❶ The `kernel_regularizer` argument adds L2 regularization to the weights in the layer.

❷ The `layer_dropout` function adds dropout regularization, randomly dropping 30% of the nodes during training. This layer has no effect during prediction.

Compare the accuracy of the regularized model to the original model. The accuracy of the training and validation set is displayed in Figure 8-6:

```
hist_df <- bind_rows(
  history_df %>% mutate(method = "simple"),
  history_reg_df %>% mutate(method = "regularized"),
) %>%
  mutate(method = factor(method, levels = c("simple", "regularized")))
ggplot(hist_df[hist_df$metric == "accuracy", ],
  aes(x = epoch, y = value, color = data)) +
  geom_line(linewidth = 0.5, alpha = 0.8) +
  geom_smooth(aes(linetype = data), method = "loess", formula = "y ~ x",
    color = "black") +
  labs(x = "Epoch", y = "Accuracy") +
```

```
  facet_wrap(~ method, ncol = 2) +
  coord_cartesian(ylim = c(0.65, 0.71))
```

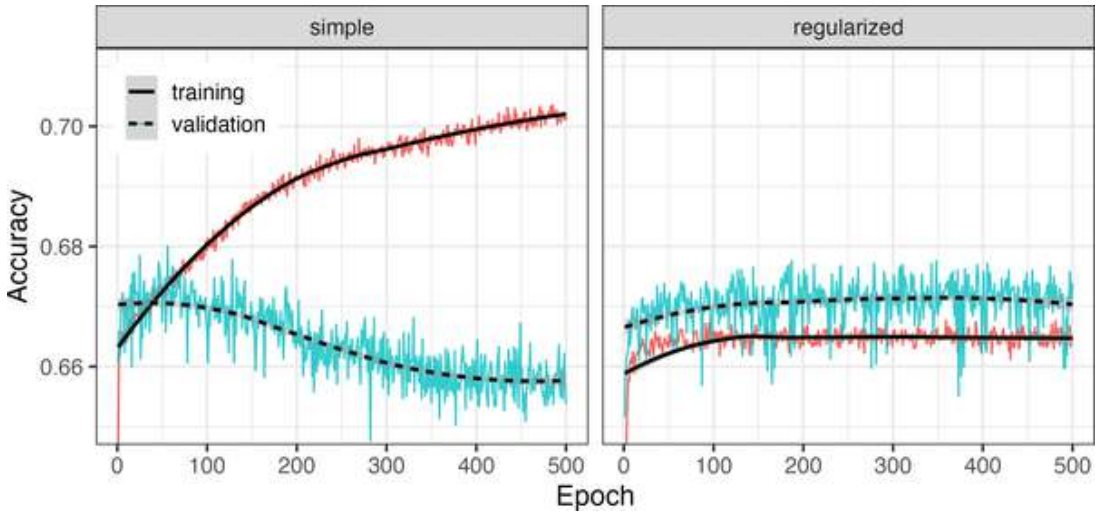


Figure 8-6. Error on training and validation sets over epochs with regularization

In contrast to the original model, the accuracy over the validation set is stable over the training epochs.

The equivalent Python code to include regularization in the training is as follows:

```
reg_model = keras.Sequential([
    keras.layers.InputLayer(shape=(15,)),
    keras.layers.Dense(units=64, activation="relu",
              kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(units=32, activation="relu",
              kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(units=1, activation="sigmoid"),
])

reg_model.compile(
    loss="binary_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=["accuracy"],
)


np.random.seed(42)
history_reg = reg_model.fit(
    x=train_x,
    y=train_y,
    epochs=500,
    batch_size=256,
    validation_split=0.1,
```

```
    verbose=0,
)
```

Using the `matplotlib` package, the accuracy and loss functions can be compared:

```
# Convert to DataFrames
history_df = pd.DataFrame({
    **history.history,
    "method": "simple",
    "epoch": range(1, 501),
})
history_reg_df = pd.DataFrame({
    **history_reg.history,
    "method": "regularized",
    "epoch": range(1, 501),
})
fig, axes = plt.subplots(ncols=2, figsize=(8, 4), sharey=True)
history_df.plot(x="epoch", y=["accuracy", "val_accuracy"], ax=axes[0])
history_reg_df.plot(x="epoch", y=["accuracy", "val_accuracy"], ax=axes[1])
for ax in axes:
    ax.set_xlabel("Epoch")
    ax.set_ylabel("Accuracy")
axes[0].set_ylim(0.64, 0.71)
plt.tight_layout()
```

Interpretation and Variable Importance

In contrast to tree models, for which variable importance metrics are a byproduct of the fitting method, neural nets are less interpretable. One way to assess variable importance for individual records in neural nets is to use SHAP values, which are model-agnostic and apply to any type of predictive model. Predicted values f for neural nets, as for tree models, can be expressed as a sum of many interaction terms. The SHAP algorithm uses concepts from game theory to allocate the interactions that express f as a linear sum of SHAP values for each feature:

$$f(x_1, x_2, \ldots, x_p) = \text{baseline} + \text{SHAP}(x_1) + \text{SHAP}(x_2) + \ldots + \text{SHAP}(x_p)$$

To compute SHAP values for neural networks in R, use the `fastshap` package. For the loan data, this is done using the function `explain`:

```
library(fastshap)
# Create a prediction wrapper function
predict_function <- function(model, newdata) {
  return(as.numeric(predict(model, as.matrix(newdata), verbose = 0)))
}
newdata <- train_x[ceiling(0.9 * n) + 1:100, ]
```

```
shap_values <- fastshap::explain(object = reg_model,
  X = train_x[1:floor(0.9 * n), ],
  pred_wrapper = predict_function,
  nsim = 50,  # Number of Monte Carlo samples (increase for more accuracy)
  newdata = newdata,
)
```

Unlike for tree models, computing SHAP values for neural nets is computationally very expensive, and SHAP values are computed only for a sample of data (in this case, 100 values). This is done on a record-by-record basis; we will aggregate the results later. Using the `shapviz` library, we display a waterfall plot of the SHAP values for a record in the loan data corresponding to a defaulted loan:

```
row_id <- 24
newdata[row_id, ]
---
        loan_amnt              term        annual_inc               dti
      0.681159420       1.000000000       0.009804226       0.584662892
payment_inc_ratio         revol_bal        revol_util  delinq_2yrs_zero
      0.247664899       0.021390884       0.525370804       1.000000000
     pub_rec_zero          open_acc             grade           purpose_
      1.000000000       0.118421053       0.323529412       0.166666667
             home_          emp_len_    borrower_score
      1.000000000       1.000000000       0.315789474
library(shapviz)
shap_viz <- shapviz(shap_values, X = newdata)
sv_waterfall(shap_viz, row_id = row_id)
```

The resulting plot is shown in Figure 8-7: the values of the feature are shown along the y-axis and the SHAP values are shown on the x-axis. The SHAP values are log-odds, which is standard for classification models (see Chapter 5). This loan has a predicted log-odds of about 0.262 for default. A log-odds of 0.262 corresponds to a probability of about 56.5% of default, compared to the baseline probability of about 50% (corresponding to an increase in probability of default of about 6.5%). The waterfall plot allocates the log-odds to the various features: the main contributors to the prediction of *default* are the *borrower score*, the *term* of the loan, the *grade*, and the *payment-to-income ratio*.

The mean of the absolute SHAP values across the sampled records can be used to estimate the importance of each feature:

```
sv_importance(shap_viz) +
  labs(x = "Variable importance (mean absolute SHAP value)")
```

The corresponding plot is shown in Figure 8-8. The most important feature is borrower score, followed by the term, payment-to-income ratio, and grade of the loan.

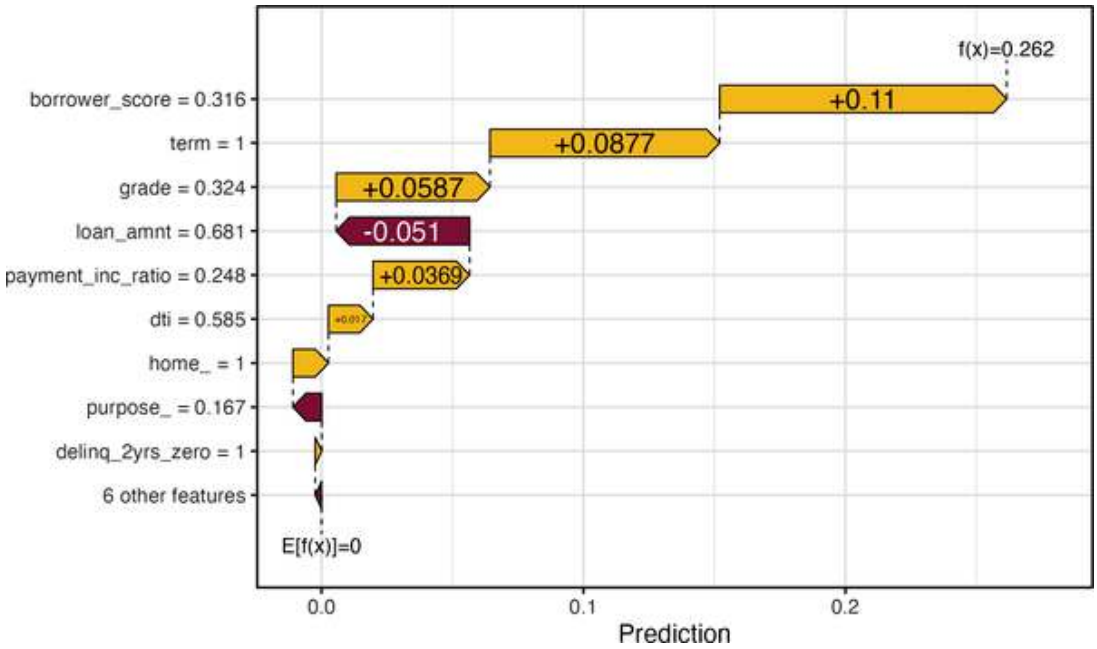


Figure 8-7. Waterfall plot of the SHAP decomposition of a prediction

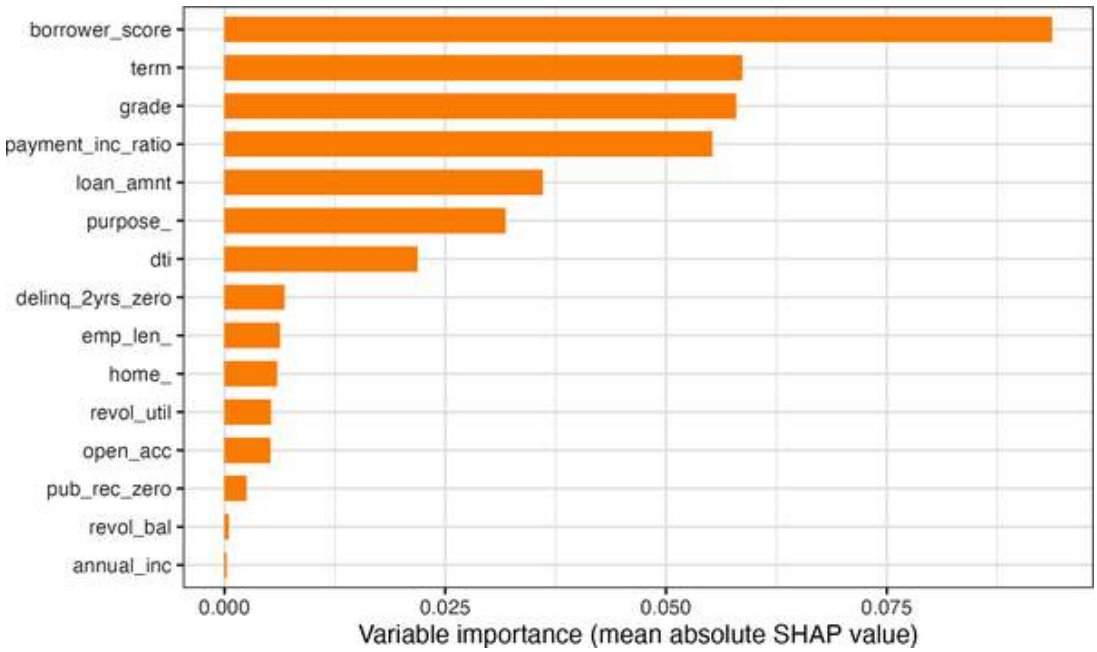


Figure 8-8. Feature importance based on SHAP values

In Python, SHAP values are computed with the `shap` package using the `DeepExplainer` method that is more efficient for deep learning models:

```
import shap

ntrain = int(np.floor(0.9 * train_x.shape[0]))
shap_train = train_x.iloc[:ntrain, :].to_numpy()
shap_test = train_x.iloc[ntrain:ntrain + 100, :].to_numpy()

explainer = shap.DeepExplainer(
    model=reg_model,
    data=shap_train,  # Background data
)
shap_values = explainer.shap_values(
    shap_test,  # Data to explain
)
```

The waterfall plot is produced by the `waterfall_plot` method:

```
n = 3
shap.waterfall_plot(shap.Explanation(
    values=shap_values.squeeze()[n],
    base_values=float(explainer.expected_value[0]),
    data=shap_test[n, :],
    feature_names=all_predictors))
```

The variable importance can be produced with a manual approach:

```
# Calculate mean absolute SHAP values
mean_abs_shap = np.abs(shap_values).mean(axis=0).flatten()
# Sort by importance
sorted_idx = np.argsort(mean_abs_shap)
sorted_features = [all_predictors[i] for i in sorted_idx]
sorted_importance = mean_abs_shap[sorted_idx]

# Create horizontal bar plot (like R)
plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_features)), sorted_importance, color="steelblue")
plt.yticks(range(len(sorted_features)), sorted_features)
plt.xlabel("mean(|SHAP|)", fontsize=12)
plt.title("SHAP Feature Importance", fontsize=14)
plt.grid(axis="x", alpha=0.3)
plt.tight_layout()
```

Hyperparameter Optimization

In hyperparameter optimization, the model's hyperparameters are tuned using the validation set. Because it is tuned after multiple looks at the validation data, the model can reflect characteristics of that particular validation partition, and may not generalize well to new data. Best practice is to divide the data into three parts, with

the third partition as *holdout* set or *test* set used to provide an unbiased estimate of final model performance. This provides a guardrail against overfitting due to tuning of the hyperparameters to the validation set. As such, this third partition must not be used in either training or tuning, lest it contribute to the training process and lose its value as a holdout set.

While boosted trees involve hyperparameters, handling them is orders of magnitude more challenging with neural network models. Neural networks have many knobs to tune (learning rate, architecture, activation functions) and require significantly more expertise to choose properly. This is especially true when it comes to interpretation of the model: the relationship between the predictors and the response can change dramatically with the choice of hyperparameters. Prior to the advent of deep learning, this was a main impediment to wider-scale adoption of neural networks.

KEY IDEAS

- Backpropagation notes the error after each batch, then proceeds backward, adjusting the biases and weights to reduce the error.
- The gradient of a node is the rate at which the prediction changes in response to a change in weights.
- Gradient descent refers to the process of adjusting weights in the direction that reduces the loss function.
- Overfitting occurs when a model is trained too extensively on training data, capturing noise rather than underlying patterns, leading to poor performance on new data.
- Regularization techniques, such as L1/L2 penalties and dropout, help prevent overfitting by constraining model complexity.

Further Reading

"A Step by Step Backpropagation Example" by Matt Mazur has a simple numerical example of backpropagation.

Loss (Cost) Functions

Loss functions, also known as cost functions (the equivalence is loose and some draw a distinction between the two terms), translate the differences between predicted and actual values across cases into a single number that is relevant for the task at hand. The training process can then seek to minimize the loss function. Different cost functions are used for different types of tasks.

KEY TERMS FOR LOSS FUNCTIONS

Loss function

A function that quantifies prediction error; used during training to improve model performance

Synonym

cost function

Mean squared error (MSE)

A regression loss function that calculates the average of the squared differences between predicted and actual values

Cross-entropy loss

A classification loss function, based on the divergence between predicted probabilities and actual class labels

Synonym

log-loss

Loss Functions for Regression

For regression tasks with continuous (numerical) data, mean squared error, or MSE, is a popular metric. It is calculated as the average of the squared differences between predicted and actual values. Squaring the differences renders them all

positive, so under-prediction and over-prediction are both equally penalized. It also penalizes larger errors more than smaller ones.

In Chapter 1 we discussed root mean squared error, RMSE, which is simply the square root of MSE. RMSE has the advantage of being in the same units as the outcome variable, making it easier to interpret. In that chapter, we also discussed metrics that are based on absolute values of errors, such as mean absolute error (MAE). These are less sensitive to large errors than MSE. The metrics mentioned in Chapter 1 have loss function analogues, except that the errors are relative to predicted values rather than to a central value like a mean or median.

Loss Functions for Classification

Classification involves predicting class membership, such as whether an email is spam or not spam. For binary classification with 0/1 (yes/no) data, unlike for regression, RMSE and MAE are not appropriate. Instead, the most common cost function for neural net classification tasks is *cross-entropy loss*, also known as *log-loss*. This is the loss function used to fit logistic regression and boosted tree models for classification. This function takes advantage of the fact that the output of a classification model is a value from 0 to 1, representing the estimated probability that predicted value is 0/1. Cross-entropy loss increases as the predicted probability diverges from the actual label. For example, if the actual label is 1 (positive class) and the model predicts a probability of 0.9, the cross-entropy loss will be low, indicating a good prediction. However, if the model predicts a probability of 0.1 for the same case, the cross-entropy loss will be high, indicating a poor prediction.

Cross-entropy measures how well a predicted probability distribution matches the true distribution by scoring the negative log-probability assigned to the correct class. In the binary case, neural networks have only one output node which produces a value between 0 and 1 via a sigmoid activation function. The cross-entropy loss for a single data point y in classification is defined as follows:

$$L(y, p) = -(y \log(p) + (1 - y) \log(1 - p))$$

This is summed across the training batch to get a loss function for all of the data. In the general multiclass problem with *K* classes, cross-entropy is the negative log of the softmax probability for the true class:

$$L = -\left(y_1 \log(\text{softmax}(y_1)) + y_2 \log(\text{softmax}(y_2)) + \cdots + y_K \log(\text{softmax}(y_K))\right)$$

In actuality, the calculations work not with probabilities, but with the logits (the weighted sums before the activation function is applied). This is because working with logits helps avoid numerical instability issues that can arise when dealing with probabilities very close to 0 or 1.

KEY IDEAS

- Loss functions assign scalar penalties to prediction errors, which are then aggregated and minimized during training.
- The loss function for regression is typically mean squared error (MSE).
- The loss function for classification is typically cross-entropy, which measures how well a predicted probability distribution matches the true distribution.

Performance

Neural networks have been around for decades, but the basic neural networks we have described here did not achieve groundbreaking performance until the advent of deep learning, which we cover in the next chapter.

KEY TERMS FOR PERFORMANCE

Vanishing gradient

When the gradients used to update weights become vanishingly small

Exploding gradient

When the gradients used to update weights become excessively large

Saturated network

A network in which weights no longer change in response to new inputs

As networks became deeper, with many more layers, they were able to capture more complex relationships in data. Even so, they were still limited in their performance

compared to other machine learning methods, such as boosted trees.

One problem was *vanishing gradients*, in which the gradients used to update weights during backpropagation became very small as they were propagated back through many layers. If weights are initialized too small, the repeated multiplications through the network can cause the weights to converge toward zero, rendering the network *saturated*: stuck at a point where weights no longer change in response to new inputs. Deeper networks with more layers can exacerbate this effect.

A similar problem is *exploding gradients*, in which the gradients become excessively large during backpropagation when repeated multiplication by large weights or derivatives amplifies gradient magnitudes, leading to unstable weight updates and divergence during training.

The use of the ReLU activation function (its derivative always = 1 for positive input) helped mitigate the vanishing gradient problem, as did tweaks to the initialization algorithm and the activation functions, allowing for the training of much deeper networks, leading to significant improvements in performance across various tasks.

Still, the basic neural network as described in this chapter offers few advantages over random forest and boosted trees while having the downside of greater complexity. To become the superstar that it is today, it needed to be able to deal with sequential data (like text), as well as image and audio data. This required fundamental changes in architecture in addition to the ReLU, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers, all going under the rubric of "deep learning." With these advances, neural networks have come to dominate the field of machine learning. These architectures are covered in the next chapter.

KEY IDEAS

- Basic neural network performance with structured data is comparable to that of other machine learning methods.
- Fundamental changes in architecture (covered in the next chapter) allow neural networks to achieve breakthroughs with image and text data.

Summary

In this chapter we introduced the basic structure of neural networks, including layers, nodes, weights, biases, and activation functions. We have described how data flows through the network during a forward pass, and how the network is trained using backpropagation and gradient descent to minimize a cost function. We have also discussed challenges such as overfitting and techniques like regularization to address them. Finally, we touched on the performance of neural networks and their evolution into deep learning architectures that have revolutionized machine learning.

Exploration with AI

1. Ask your AI for suggestions on how to set the hyperparameters, in general, and for the examples in this chapter. What is the most important hyperparameter? When do you use an L1 regularizer versus an L2 regularizer? How does this relate to the penalized regression discussed in "Regularization: Avoiding Overfitting"?

2. Ask your AI when it is appropriate to use a basic neural network instead of a random forest or boosted trees. Are there any cases involving rectangular (structured) data in which it is better to use a neural network?

3. In this chapter, `tensorflow` and `keras` were used in the examples. Ask your AI whether there are other, open source, alternatives to `tensorflow` and `keras`? Try using an LLM to translate the examples in this chapter to a different package. How much difference does it make?

4. Ask your AI what the best methods are for hyperparameter optimization. Ask it to produce a function to perform a hyperparameter optimization for the loan data. How much difference did it make in the results?

5. GPUs can vastly speed up training of neural networks. Ask your AI what types of problems benefit most from using GPUs, in terms of speed and cost. Is it worth using GPUs for the examples in this section?

Chapter 9. Deep Learning

Neural nets had their greatest impact following the 2006 publication of a pair of papers by AI researcher Geoffrey Hinton[1] and the rapidly growing popularity of what came to be called *deep learning*. Deep learning involves large and complex networks with many layers, incorporating processes for dimension reduction and feature discovery. The advent of big data, and of powerful graphics processing unit (GPU) chips, has accelerated the growth of deep learning. Deep learning has revolutionized fields such as computer vision, natural language processing, speech recognition, and generative AI, enabling breakthroughs in applications like image classification, machine translation, and voice assistants.

Advantages of Depth

The data we have dealt with so far in this book is structured data, typically from organized databases. The predictor variables, or features, that we think might be meaningful in predicting an outcome of interest already exist in our data. In predicting loan default using the LendingClub data Table 6-1, for example, we used features that have clear predictive value such as debt-to-income level or the ratio of loan payment to income. With tasks like text mining and image recognition, structured high-level predictor information like this is not available. Even with some structured data, there may be complex, nonlinear relationships among the features that are not easily captured by traditional statistical models. An example of the latter is complex and messy medical data that includes structured predictors like measurements (where records for a single variable may have conflicting units and transcription methods) as well as text (which may involve both written reports and cryptic notes). Deep learning methods excel at automatically discovering relevant features and complex patterns in large data sets, and are now the state of the art for prediction with big data.

What qualifies a neural network as “deep”? Size, certainly. There is no precise dividing line, but a neural network with only one or two layers is considered simple, and one with five or more layers is said to be deep. Deep learning networks often have many nodes per layer (in the thousands), but the term “deep” strictly applies to the number of layers.

The additional layers in a neural network can stack representations in hierarchical fashion to help the network learn new representations that might be features like edges and shapes that are interpretable to humans (particularly in early layers) or simply abstractions that contribute to learning.

Traditional statistical theory suggests that the size and complexity of deep networks should lead to overfitting, but in practice, they usually don’t. Specific features in deep learning are designed to counteract overfitting and vanishing gradients. To address overfitting, deep networks use regularization techniques such as:

- *Dropout* randomly disables some neurons as a regularization technique.
- *Global average pooling* (GAP) reduces the number of parameters and prevents overfitting by averaging each feature map into a single value, rather than flattening the entire feature map into a long vector.

To address the vanishing gradient problem, deep networks use techniques such as:

- *ReLU activation functions*, which we discussed in the last chapter, are typically used in deep networks (early simpler neural nets used sigmoid or tanh).
- *Batch normalization* is used between layers to keep activations within similar scales, and avoid vanishing or exploding gradients (which would otherwise be a problem as the number of layers grows). Normalization might be across the entire batch, or might be confined to smaller structures.
- In the same vein, *skip connections* directly between early and late layers, passing over intermediate layers, minimize the vanishing gradient problem.

Important steps on the deep learning path were the development of (1) CNNs for image recognition tasks, and (2) RNNs for sequence data such as time series and text. More recently, *transformer* architectures have emerged as a powerful alternative to RNNs and CNNs for sequence and image modeling. This is particularly true for natural language processing; transformer methods form the basis for large language models (LLMs) such as those embodied in ChatGPT.

This chapter goes beyond the statistical foundations of earlier chapters, and isn't strictly considered part of statistics. However, deep learning is now an essential tool for data scientists, so we provide a brief overview of the key concepts and architectures. For a more comprehensive treatment of deep learning, we recommend *Deep Learning* by Goodfellow, Bengio, and Courville (MIT Press, 2016) and *Deep Learning with Python* by Francois Chollet (Manning Publications, 2017).

DOUBLE DESCENT

Figure 8-5 in Chapter 8 showed a typical learning curve: training and validation error both fall at first, then validation error rises as the model begins to overfit (note that we used accuracy = 1 – error as the metric there). In very large neural networks, however, researchers have observed that validation error can decline again as model capacity grows. This pattern, shown in Figure 9-1, is called *double descent*.

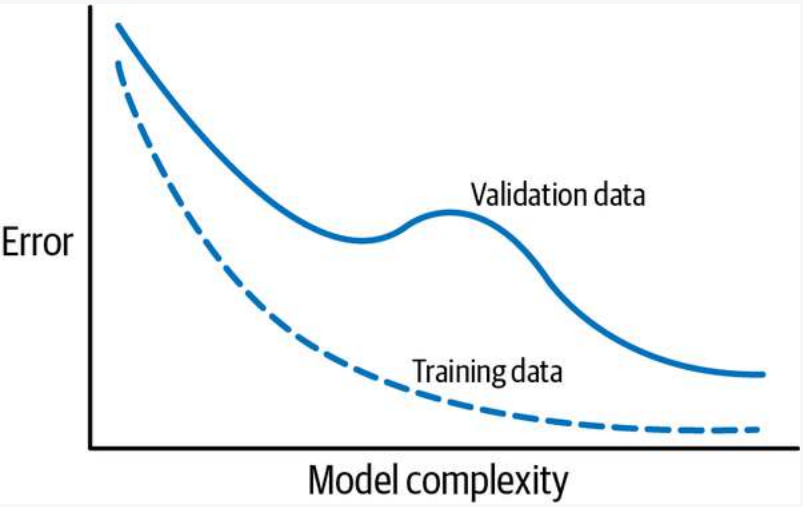


Figure 9-1. Error on training and validation data over model complexity—double descent of validation error

One line of explanation argues that, at sufficiently large scales, neural networks discover higher-level, more generalizable features that reduce validation error despite extreme over-parameterization. Another explanation notes that massively over-parameterized models admit many perfect fits, and that the training dynamics themselves act like an implicit regularizer that selects solutions with better generalization. The precise mechanisms behind double descent remain an active area of research.

RNNs

A drawback to the standard feedforward neural network architecture described in Chapter 8 is that it assumes that all inputs (predictor variables) are independent of one another. In fact, many data sets have correlations between the data points. Data collected over time typically measures the same feature repeatedly and has temporal correlations, where the value at one time point is correlated with the values at previous time points. Examples include stock prices, weather measurements, sales volumes, and sensor readings. Other data sets have sequential dependencies, where the order of the data points matters. Examples include text, where the order of words matters, and music, where the order of notes matters.

Recurrent neural networks (RNNs) are one type of modification to basic neural networks designed to model ordered sequences of data. Compared with the feedforward neural network shown in Figure 8-1, the RNN includes an additional connection that feeds forward an internal state vector (the activations of the hidden layer at the previous step in the sequence). This recurrent state allows the network to incorporate information from earlier inputs when processing later ones.

As shown in Figure 9-2, at each timestep, the input I_t is projected using a fully connected layer FC_{in}, while the previous hidden state H_{t-1} is transformed by a separate fully connected layer FC_{hidden}. These two projections are summed and passed through a *tanh* activation function to produce the updated hidden state H_t, which is kept for the next timestep. A third fully connected layer FC_{out} maps the hidden state to the output O_t. The figure repeats and unrolls the RNN across three timesteps to more clearly demonstrate the flow of information. In fact, all weight matrices are shared across timesteps. This means the network applies the same transformation at every step, regardless of sequence position. The hidden state allows the network to maintain a form of memory, enabling it to capture temporal dependencies in sequential data.

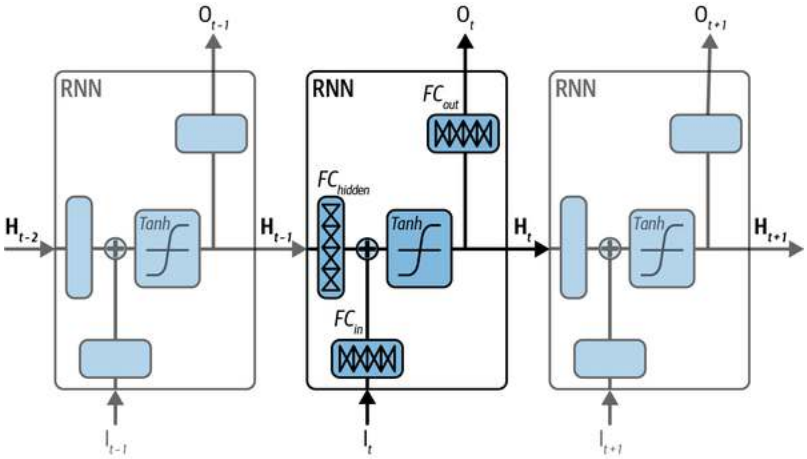


Figure 9-2. Structure of a basic RNN shown across three consecutive timesteps ($t-1$, t, $t+1$)

KEY TERMS FOR RNNS

Token

A token is a discrete step in a sequence. For example, with text, it can be a word, subword, character, or punctuation.

Short-term memory

The ability of a simple RNN to remember information from recent timesteps, but not from longer history, due to the vanishing gradient problem.

Long short-term memory (LSTM)

A special type of RNN, known as LSTM, that is designed to overcome the vanishing gradient problem and can remember information for longer periods.

Cell state

A component of LSTM networks that retains information from previous steps in the sequence, enabling the network to remember information over longer periods.

Gates

Special mechanisms in LSTM networks that regulate the flow of information.

Backpropagation through time (BPTT)

A training method for RNNs that involves backpropagating errors through both the layers of the network and the timesteps of the sequence, allowing the network to learn from past information.

As with standard neural networks, an RNN can have multiple hidden layers, but for simplicity we show only one hidden layer in Figure 9-2. Training an RNN involves backpropagation, except that the errors are backpropagated through time (BPTT) as well as through the layers. In order to do this, the network is "unrolled" into a sequence of identical copies where each copy represents a single timestep. Note that "time" here refers to position in the sequence, not actual time, and the process covers both classic time-series data and sequential data such as text.

Text Data

The deep learning models we discuss in this chapter and the LLMs that we discuss in Chapter 10 achieved breakthroughs in text processing and generation. For these models, text is basically a sequence of *tokens*. A token might be a word, a subword, a character, or punctuation. Text that we encounter in books, on the internet, in reports, etc., must be converted into tokens, or *tokenized*, before it can be processed by a deep learning model.

Many words (e.g., book, oven) might be tokens on their own. Others (e.g., autoregressive, unrepresentative) might be broken into smaller tokens (e.g., auto, regressive, un, represent, ative). Punctuation characters also might be tokens on their own. In the early days of text mining, tokenization rules were set by humans. Tokenizers now are learned, and there are multiple tokenization methods available by default in programming and AI frameworks. Specialized domains (e.g., aircraft maintenance logs and tickets) often require retraining a tokenizer on a relevant corpus of text. Note that this "learning" refers to the statistical optimization of a fixed vocabulary (like byte-pair encoding) before training begins, which is a separate, preparatory process from the actual deep learning used to train the LLM itself.

In this book and elsewhere you will encounter the terms *token* and *word*. *Token* is the more technical term, but *word* is more descriptive and is also used in the technical literature (e.g., *word embeddings*). Although the two

terms are not strict synonyms, we use them somewhat interchangeably and do not focus on distinctions between the two.

For enterprise and commercial users of LLMs, part of the cost of using an LLM is a per-token processing fee. If prompts are quite lengthy, or if the text to be processed is voluminous, it is prudent to estimate the number of tokens involved. One rule of thumb is to allot one token for every four characters; another is to multiply the number of words by 1.3.

Memory Issues

A simple RNN only has *short-term memory* in the sense that it remembers information from very recent periods, but not from longer history. (Although we speak of "periods" and "history," we are not just talking about time-series data but about sequence data in general, e.g., text.) This is due to the vanishing gradient problem as discussed in "Performance". In BPTT, the gradient is calculated by repeatedly multiplying the weight matrices and the derivatives of the activation functions at each timestep. Even with modern activation functions like ReLU, if the weights are small, these repeated multiplications cause the gradient to shrink exponentially. Consequently, after a number of timesteps, the gradients approach zero, and the network stops learning from the earlier parts of the sequence. For example, if the combined gradient factor is 0.5 at each step, after 3 steps the gradient becomes $0.5^3 = 0.125$, and after 10 steps, it becomes $0.5^{10} < 0.001$. At this point, the signal is too weak to update the weights effectively, causing the RNN to "forget" what it saw much earlier in the sequence.

To overcome the short-term memory issue, a variant of RNN called long short-term memory (LSTM) was developed. LSTM networks have the same basic structure as in Figure 9-2, but in addition to the activation function to capture short-term dependencies, the LSTM also retains information from the previous steps in a *cell state* as depicted in Figure 9-3.

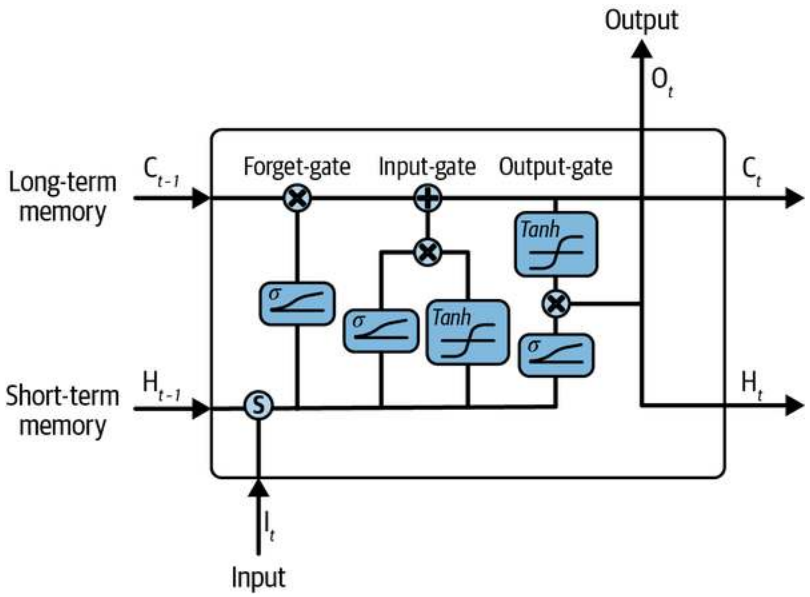


Figure 9-3. Structure of a long short-term memory (LSTM) cell at a single timestep t

The cell maintains two recurrent connections: the cell state $C_{t-1} \rightarrow C_t$ (top horizontal line), which acts as long-term memory, and the hidden state $H_{t-1} \rightarrow H_t$ (bottom horizontal line), which acts as short-term memory. At each timestep, the current input I_t is concatenated with the previous hidden state H_{t-1} at the stacking node (S), and this combined signal is routed through three learned gating mechanisms: the forget gate (σ), which decides what information to erase from the cell state (we represent this and the other gates only using the activation function, but in practice it is a fully connected layer with a sigmoid activation function); the input gate (σ + tanh), which decides what new information to write into the cell state; and the output gate (σ + tanh), which decides what portion of the updated cell state to expose as the new hidden state H_t and return as the output O_t. This allows LSTMs to capture

dependencies over much longer sequences than a vanilla RNN. An output layer is not shown in Figure 9-3, but would be used to transform the hidden state.

The cell state is what enables LSTM to remember information over longer periods. LSTM also adds to each hidden node special mechanisms called *gates* that regulate the flow of information. There are three types of gates: a *forget gate* that determines how much of the previous cell to remember, an *input gate* to determine how much of the new information is added to the cell state, and an *output gate* to determine how much of the cell state is included in the output.

RNNS AND AUTOREGRESSIVE MODELS

RNNs have some similarities to traditional autoregressive (AR) models used in time-series analysis. In an AR model, the current value of a time-series is expressed as a linear combination of its past values plus a random error term. For example, an AR model with a two-lag structure can be written as:

$$y_t = c + \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \epsilon_t$$

This can be fit using linear regression with lagged values as predictors (as in Chapter 4). It turns out that a simple RNN with one hidden layer and linear activation functions can also represent an AR model. In practice, RNNs are far more flexible than AR models and were designed to capture nonlinear relationships and complex temporal dependencies in the data. Consequently, RNNs are capable of modeling a much wider range of sequence data than AR models, and have been applied to tasks such as language modeling, machine translation, and speech recognition, where the relationships between inputs and outputs are highly nonlinear and complex.

There are other variants of RNNs, such as gated recurrent units (GRUs), which have a simpler structure than LSTM by combining the forget and input gates into a single update gate and using fewer parameters. GRUs often match LSTM performance while being faster to train, making them a practical alternative for many sequence tasks.

RNNs and their variants are now mostly supplanted by transformer architectures for large-scale language and sequence modeling. Transformers offer better parallelism and can model long-range dependencies more effectively. The primary advantage of RNNs and LSTMs over transformers is their lower computational requirements and simpler architecture. They are still used for applications such as time-series forecasting with smaller data sets and real-time processing where computational resources are limited.

KEY IDEAS

- Text data requires conversion to a series of tokens, which can be words, subwords, characters, or punctuation.
- RNNs add a connection that incorporates prior information into a current processing step, enabling them to handle sequence data, such as time series or text, that are hard to model with basic neural networks.
- LSTMs are a special type of RNN that can remember information for longer periods, overcoming the vanishing gradient problem.
- RNNs have been replaced by transformer architectures for large-scale language and sequence modeling.

Further Reading

- "Understanding LSTM Networks" (Colah's Blog): This is arguably the most famous article on the topic. It uses incredibly clear visual diagrams to explain how the "gates" in an LSTM actually work.

- "The Unreasonable Effectiveness of Recurrent Neural Networks" (Andrej Karpathy): A classic read that demonstrates what RNNs can do (like writing Shakespeare or code) and explains the magic of character-level modeling.

CNNs

Image data, unlike structured tabular data used in previous chapters, consists of raw pixel values that represent the intensity and color of each pixel in the image. Like time-series and text data, image data also has correlations between data points. An individual pixel is much more likely to be similar to a neighboring pixel than some random pixel elsewhere. Convolutional neural networks (CNNs), with their ability to consolidate pixel data and detect features, revolutionized the field of image processing. While CNNs are used for different tasks, the most common application is computer vision.

Prior to the advent of deep learning, the field of computer vision was based on manually engineered features and traditional machine learning algorithms. Methods relied on techniques such as edge detection, texture analysis, and shape recognition to extract meaningful information from images. These techniques modeled physical and geometric characteristics of image formation. Handcrafted features were then used as inputs to classifiers such as decision trees. While traditional computer vision modeling achieved some success, it was limited in its ability to generalize across diverse image data sets and struggled with complex visual tasks.

In a seminal paper on CNNs published in 2012, Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton demonstrated that deep learning methods (which they called AlexNet) could dramatically outperform traditional computer vision techniques on image classification tasks. They reported a reduction in the error rate from 26.2% to 15.3% in standard computer vision benchmarks, a previously unheard-of degree of improvement. This breakthrough was made possible by the ability of large CNNs to automatically learn hierarchical features from raw pixel data, eliminating the need for manual feature engineering. The success of CNNs sparked a revolution in computer vision, leading to rapid advancements in image recognition, object detection, and other visual tasks. CNNs essentially wiped out the field of traditional computer vision almost overnight and became the dominant approach for image analysis until the recent advent of transformer architectures.

KEY TERMS FOR CNNS

Convolution

In computer vision, a filter applied to images for tasks such as edge detection, blurring, and sharpening.

Channel

A channel is a two-dimensional array of values that corresponds either to an input component (e.g., red, green, blue) or to a feature map produced by a convolutional filter.

Sobel operator

A specific type of convolution filter used for edge detection in images.

Pooling

A dimension reduction technique that reduces the spatial dimensions of feature maps while retaining important information, often by taking the maximum or average value in a small region.

Stride

Another dimension reduction technique that determines the number of pixels the convolution operator skips over between applications, as it moves around an image.

Flatten

A flatten layer converts a multidimensional feature map into a one-dimensional vector while keeping all the information.

Global average pooling (GAP)

An alternative to flatten, computing the average of all values in each feature map (a two-dimensional array of activations produced by a convolutional filter), reducing the spatial dimensions to 1×1.

Fully connected layer

A layer in a neural network where each neuron is connected to every neuron in the previous layer, used to combine the learned features to make predictions.

Transfer learning

A technique where a pretrained model (trained on a large data set) is fine-tuned on a smaller target data set, leveraging the learned features from the pretrained model for faster convergence and improved performance on the target task.

Intuition Behind CNNs

In CNNs, pixels are grouped into small regions and a *convolution* operator is applied to each region. Convolutions have a rich history (see "Convolution: A Rich History"), and were integral to image processing even before the advent of deep learning. To understand how CNNs work, it is instructive to review how these were used in traditional image processing.

Consider a simple 8×8 pixel black-and-white image—three rows of black pixels (0s) at the top, five rows of white pixels (255s) at the bottom. The boundary between the black area and the white area is an example of an *edge*:

$$\text{Image} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \end{bmatrix}$$

A convolution operator is a small matrix (called a kernel or filter) that slides over the image, performing element-wise multiplication and summing the results to produce a single output value for each region. For example, the *Sobel* operator is a 3 × 3 convolution matrix for vertical (G_x) and horizontal (G_y) edge detection:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

These are multiplied element-wise with each 3 × 3 region of the image, and the results summed to produce a single output value for each region. For example, its use is shown below for horizontal edge detection. It is applied to the region centered at row 4 and column 2 (it would look the same for columns 3–7):

$$G_y \times \begin{bmatrix} 0 & 0 & 0 \\ 255 & 255 & 255 \\ 255 & 255 & 255 \end{bmatrix} = (-1*0 - 2*0 - 1*0) + (0*255 + 0*255 + 0*255) + \qquad (1'$$

If we apply this over each 3×3 region (represented by a convolution operator *), we get a new matrix that highlights the horizontal edge. For example, applying G_y to a black (0) strip above a white (255) region yields:

$$G_y\text{*Image} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 765 & 1020 & 1020 & 1020 & 1020 & 1020 & 1020 & 765 \\ 765 & 1020 & 1020 & 1020 & 1020 & 1020 & 1020 & 765 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The resulting matrix highlights the horizontal edge in the original image. (We have added zeros all around the image to facilitate computations at image boundaries. Other options are to mirror the pixel value at the boundary, or to limit the operation to those portions of the image where there are valid values for all cells.)

The architecture in CNNs is similar to that in traditional computer vision algorithms with one key difference: the convolution filters are learned automatically instead of being prespecified. This is the key innovation that allows CNNs to adapt to different tasks and data sets. As with standard neural networks, the output from the convolution operator is passed through an activation function, such as ReLU, to introduce nonlinearity. The network learns to identify the most relevant features for the task at hand, such as edges, textures, and shapes. This approach enables CNNs to automatically discover complex features that are difficult to engineer manually.

In modern CNNs, there are different types of convolutional layers. A major distinction is the *stride* of the convolution, which determines how far the filter moves at each step. A stride of 1 means the filter moves one pixel at a time, while a stride of 2 means it moves two pixels at a time. Larger strides result in smaller output feature

maps: a stride of 2 reduces the size of the feature map by a factor of 4. This helps with computational complexity and overfitting but potentially loses some spatial information.

CONVOLUTION: A RICH HISTORY

The mathematical operation of convolution has a long and rich history dating back to famous mathematicians such as Euler, Laplace, and Cauchy in the late 1700s and early 1800s. In 1822, Fourier discovered the *Fourier transform* that converts a signal from the time domain to the frequency domain and proved the theorem that multiplication in frequency is equivalent to convolution in time. The Fourier transform forms the basis for modern time-series analysis in the frequency domain.

In the middle of the 20th century, convolution found its first practical application in signal processing of audio and images, combining two functions to produce a third function that represents how one function modifies the other. In 1965, Cooley and Tukey developed the fast Fourier transform (FFT), which revolutionized signal processing since it improved computational efficiency for processing signals (this is the same Tukey who founded exploratory data analysis presented in Chapter 1). In the context of traditional computer vision, convolution was used to apply filters to images for tasks such as edge detection, blurring, and sharpening. A different type of filter is used for each type of task; for example, a *Sobel* filter is used for edge detection. Convolution found its way into neural networks in the late 1980s and early 1990s, with the development of convolutional neural networks (CNNs) by researchers such as Yann LeCun. The seminal work by LeCun et al. in 1998 on the LeNet architecture demonstrated the effectiveness of CNNs for handwritten digit recognition. LeCun laid the groundwork for the deep learning revolution that would follow decades later through the use of GPUs to accelerate computations by an order of magnitude, or even two.

CNN Architecture

As with sequences or time series, nearby pixels in an image can be highly correlated. In a standard neural network applied to an image, each pixel would get its own weight at each layer of the network. For example, in an image of 512 by 512 pixels, a standard neural network with H hidden nodes would require $512 \times 512 \times H = 262,144 \times H$ weights for a single layer. By contrast, a CNN uses weight sharing: the same small filter is applied at every image location, so a single 3 × 3 convolutional filter for a black-and-white image requires just 9 weights plus a bias term (color needs 3 times as many). Deep learning architecture is evolving rapidly, but, at this writing (2026), a single layer might contain between 32 and 1,024 filters, corresponding to between 288 and 9,216 weights. This weight sharing is possible due to correlations among nearby pixels and the fact that local regions of an image often share similar features such as edges and textures.

The structure of CNNs has evolved over time, but all CNNs share a similar architecture. Pixels from a raw image are processed through a series of layers, each designed to extract increasingly complex features. At the final layer, the network produces a set of feature maps, or two-dimensional arrays of activations. These are output to a final layer; in a classification task this produces the output probabilities for classification. One particular architecture is shown in Figure 9-4. A CNN goes through the following steps:

1. Input layer

The raw pixel values of the image are fed into the network. Black-and-white images have a single channel, while color images typically have three channels (red, green, blue).

2. Convolutional layers

In each convolution layer, a set of filters is applied in parallel to the input image as detailed in "Intuition Behind CNNs". The early layers have fewer filters, and the number of filters increases as you go deeper into the network, allowing the network to learn more complex features. For example, the first layer may have 32 filters, while deeper layers may have 256 or more filters. Different filters capture different aspects of the image. Early filters detect features such as edges, textures, and shapes. Later filters capture more abstract

representations that may not correspond to visual elements that are meaningful to the human eye. The convolutional layer outputs an activation tensor that has close to the same dimensions as the input dimensions times the number of channels. (A *tensor* is a multi-dimensional array of numbers. A scalar is a tensor of rank 0, a vector is a tensor of rank 1, a matrix is a tensor of rank 2, etc.) The convolution operation typically involves some form of padding (e.g., adding zeros around the borders of the image) to allow the filter to be applied at the edges, but if this isn't done, then the output feature map will be smaller than the input image. For example, a 3 × 3 filter applied to a 100 × 100 image with no padding would produce a 98 × 98 feature map.

3. Dimension reduction layer

The dimension reduction layer reduces spatial dimensions while retaining important information. This helps to reduce computational complexity and prevent overfitting. Dimension reduction layers are performed either through a *pooling operation* (e.g., taking the maximum of values in a small region) or another convolutional layer with, for example, a stride of 2. In Figure 9-4, a convolutional striding layer is used to reduce the feature map from 100×100 to 50×50. In the case where the convolutional layer uses no padding, the dimension reduction layer adjusts for the smaller feature map and still produces a 50 × 50 feature map.

4. Additional convolutional and dimension reduction layers

The process of convolution and dimension reduction is repeated multiple times, allowing the network to learn hierarchical features. In Figure 9-4, there are four more convolutional layers and one more reduction layer to take the spatial dimension from 50×50 to 25×25. Early layers may detect simple features like edges, deeper layers detect more complex patterns like shapes and objects, and still deeper layers detect more abstract patterns. Typical architectures have between 16 and 50 convolutional layers. The depth of the network is a key factor in its ability to learn complex features. Convolution layers can be repeated, not necessarily followed by a dimension reduction layer.

5. Flatten or GAP layer

This layer flattens the feature maps, either by turning the entire activation tensor into a single one-dimensional vector (flattening) or by further reducing the spatial dimensions of the feature maps by computing the average value of each feature map (GAP). If the final layer produces 512 feature maps, each with dimensions of 7 × 7, then flattening yields a vector of $512 \times 7 \times 7 = 25,088$ values, whereas GAP yields a vector of 512 values. The output from GAP can be considered as 512 specialized views of the original image, each highlighting a specific characteristic. GAP is more common in modern networks, and results in a fixed-length vector regardless of the input image size, reducing the number of parameters and preventing overfitting.

6. Fully connected layer

The output from the final pooling or GAP layer is fed into a conventional neural network layer. This layer combines the learned features to make predictions. There is not always a fully connected layer; in some architectures, the output from the GAP layer is fed directly into the output layer.

7. Output layer

The final layer produces the output, which can be a class label (for classification tasks) or a continuous value (for regression tasks).

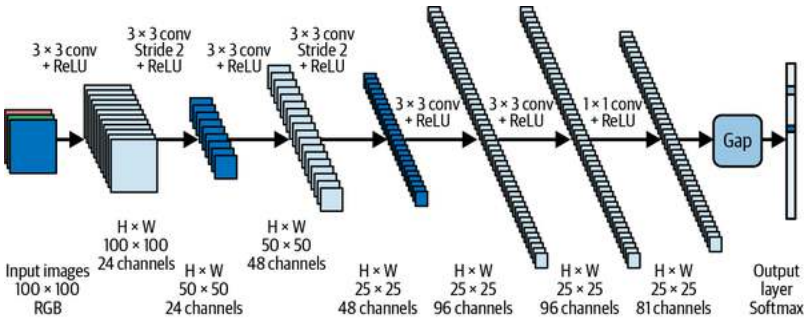


Figure 9-4. Structure of a basic convolutional neural network (CNN)

Image Data and Transfer Learning

Several collections of labeled image data sets are publicly available for training and evaluating computer vision models, including CNNs. Some common general-purpose image data sets include:

ImageNet

Large-scale image classification data set (1,000 classes, 14+ million images)

CIFAR-10 / CIFAR-100

Small 32 × 32 color image data sets for classification (10 / 100 classes, 120,000 images)

MNIST

28 × 28 grayscale handwritten digits (10 classes, 70,000 images)

Open Images V7

Very large data set with image-level labels, bounding boxes, and visual relationships (600 classes, 9+ million images)

Places365

Large-scale scene recognition data set (365 classes, 1.8+ million images)

COCO (Common Objects in Context)

Object detection, segmentation, and captioning with complex scenes (80 object categories, 330,000 images)

CelebA

Large-scale face data set with attribute annotations (10,000+ celebrities, 200,000+ images)

There are also specialized data sets for medical imaging, satellite imagery, and other domains. These data sets provide a rich source of training data for CNNs, enabling them to learn complex features and patterns from the images.

Training a CNN from scratch requires significant computational resources and time, possibly taking days or weeks on powerful GPUs. To mitigate this, *transfer learning* is commonly employed, where a pretrained model is adapted for a more specific data set. Transfer learning was an early adjunct to the application of CNNs, where the CNN (trained on a large data set like ImageNet) is fine-tuned on a smaller target data set. This approach leverages the learned features from the pretrained model, allowing for faster convergence and improved performance on the target task. As with image collections, there are multiple pretrained CNN models available for transfer learning, including VGG16, ResNet50, InceptionV3, and MobileNetV2. These models can be adapted to new tasks by replacing the final output layer and fine-tuning the weights on the target data set. (Transfer learning is now applied to other types of models, beyond CNNs.)

A typical workflow for using pretrained CNNs with image data involves the following steps:

1. *Data preprocessing*

   Images are resized to a consistent size, normalized, and augmented (e.g., rotations, flips) to increase the diversity of the training data.

2. *Model selection*

   Choose a pretrained CNN architecture suitable for the task (e.g., VGG16 for image classification).

3. *Transfer learning*

   Replace the final output layer of the pretrained model with a new layer appropriate for the target task (e.g., softmax layer for multiclass classification).

4. *Fine-tuning*

   Train the modified model on the target data set, typically with a lower learning rate to avoid overfitting.

Putting It All Together: An Example

To illustrate the intuition behind CNNs, let's construct a simple example of a CNN using the MNIST data set of handwritten digits with the `pytorch` library. The MNIST data set consists of 28 × 28 pixel grayscale images of handwritten digits (0–9). The value of each pixel ranges from 0 to 1, indicating the intensity of the pixel (0 for black and 1 for white). First, we prep the MNIST data from `torchvision`, scaling the pixels to be between –1 and 1, and creating a training and test set:

```
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader

transform = transforms.Compose([  ❶
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)),
])
trainset = torchvision.datasets.MNIST(root="./data", train=True,
  download=True, transform=transform)
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testset = torchvision.datasets.MNIST(root="./data", train=False,
  download=True, transform=transform)
testloader = DataLoader(testset, batch_size=64, shuffle=False)
```

❶ The `transform` object is a composition of two transformations: converting the images to tensors and then normalizing the pixel values to range from –1 to 1 using the mean of 0.5 and standard deviation of 0.5. Because the original pixel values are between 0 and 1, this normalization scales the data to a range of [–1, 1], which can help with training stability and convergence.

Plot a single image from the test set to visualize the data:

```
import matplotlib.pyplot as plt

sample_image, label = testset[160]
image_np = sample_image.squeeze()  ❶

fig, ax = plt.subplots(figsize=(5, 5))
img = ax.imshow(image_np, cmap="gray")
fig.colorbar(img, ax=ax)
ax.set_title(f"Label: {label}")
```

❶ The sample image has dimension [1, 28, 28] (1 channel, 28 pixels high, 28 pixels wide). The `squeeze()` method removes the channel dimension, resulting in a 2D array of shape [28, 28] for visualization.

This image is shown in Figure 9-5.

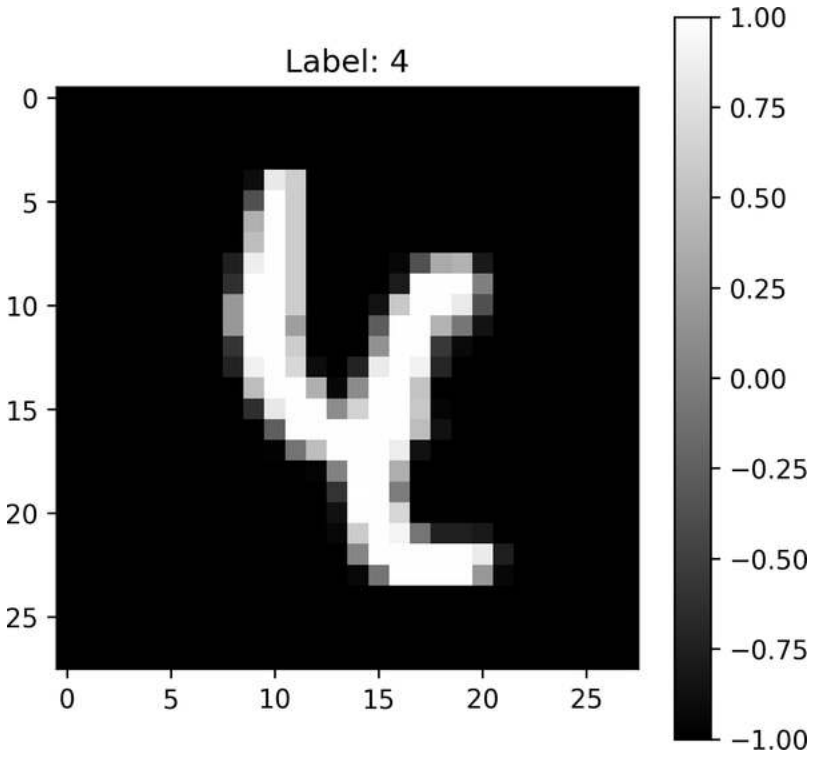


Figure 9-5. Sample image of a handwritten digit (4) from the MNIST data set

Next, we define a model with three convolutional layers, two pooling layers, a GAP layer, and a fully connected layer to produce the output class probabilities. We will use the Adam optimizer and cross-entropy loss for training the model:

```
import torch
from torch import nn

device = torch.device(
    "cuda" if torch.cuda.is_available() else
    "mps" if torch.backends.mps.is_available() else
    "cpu")  ❶

model = nn.Sequential(
    # low-level feature extractor:
    # 1 channel, 32 filters, filter size 3x3, padding to keep image same size
    # conv1: 320 parameters [32x(1x3x3 + 1)]
    nn.Conv2d(1, 32, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),  # pool1: 28x28 -> 14x14
    # mid-level feature extractor: edges -> shapes
    # conv2: 18,496 parameters [64x(32x3x3 + 1)]
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),  # pool2: 14x14 -> 7x7
    # high-level feature extractor: shapes -> objects
    # conv3: 73,856 parameters [128x(64x3x3 + 1)]
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.AdaptiveAvgPool2d(1),  # GAP: 7x7 -> 1x1
    nn.Flatten(),  # create 128 long vector
    nn.Linear(128, 10),  # 1,290 parameters [128x10 + 10]; map to 10 class scores
).to(device)  ❷

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

❶ The `device` object is used to allocate tensors and run operations on either a GPU (if available), a Metal Performance Shaders (MPS) device (for Apple Silicon), or a CPU. This allows for efficient computation if hardware acceleration is available.

❷ The model is moved to the specified device (GPU, MPS, or CPU) using the `to(device)` method.

For training the model, we first define a function to run one epoch of training. It is general enough to be used for any model, data set, optimizer, and loss function. The function iterates over the training data, performs a forward pass to compute the outputs and loss, and then performs a backward pass to update the model parameters using the optimizer. The average loss for the epoch is returned at the end:

```
def train_one_epoch(model, dataloader, optimizer, criterion):
    model.train()  ❶
    running_loss = 0.0
    for images, labels in dataloader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()  ❷
        outputs = model(images)  ❸
        loss = criterion(outputs, labels)  ❹
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    return running_loss / len(dataloader)
```

❶ The model is set to training mode using `model.train()`.

❷ The call to `optimizer.zero_grad()` ensures that the gradients are reset before the backpropagation.

❸ Forward pass: The input images are passed through the model to obtain the predicted outputs, and the result is used to calculate the loss.

❹ Backward pass: The gradients are computed using `loss.backward()` and the model parameters are updated using `optimizer.step()`.

In the same way that we defined a function to train the model, we can define a function to evaluate the model on the test set. This function sets the model to evaluation mode, iterates over the test data, and computes the accuracy of the model's predictions compared to the true labels:

```
@torch.no_grad()  ❶
def evaluate(model, dataloader):
    model.eval()

    correct = 0
    total = 0
    for images, labels in dataloader:
        images, labels = images.to(device), labels.to(device)
        logits = model(images)
        _, predicted = torch.max(logits, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()  ❷
    return 100 * correct / total
```

❶ The `@torch.no_grad()` decorator disables gradient calculations, which reduces memory usage and speeds up computations during evaluation since we don't need to compute gradients for backpropagation.

❷ The `.item()` method extracts the scalar value from the tensor and copies it to the CPU as a Python value to accumulate the number of correct predictions.

We train the model for 5 epochs. At the start of the first epoch, the weights are random and iteratively updated during training. At the end of each epoch, we evaluate the accuracy for each epoch on the test set:

```
import random
import numpy as np
seed = 1010  ❶
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)

num_epochs = 5
for epoch in range(num_epochs):
    avg_loss = train_one_epoch(model, trainloader, optimizer, criterion)
    accuracy = evaluate(model, testloader)
    print(
        f"Epoch [{epoch+1}/{num_epochs}]  "
        f"Loss: {avg_loss:.4f}  "
        f"Test Accuracy: {accuracy:.2f}%"
    )
---
Epoch [1/5]  Loss: 0.5793  Test Accuracy: 94.79%
Epoch [2/5]  Loss: 0.1597  Test Accuracy: 96.03%
Epoch [3/5]  Loss: 0.1114  Test Accuracy: 97.24%
Epoch [4/5]  Loss: 0.0840  Test Accuracy: 97.69%
Epoch [5/5]  Loss: 0.0691  Test Accuracy: 98.06%
```

❶ We set a random seed for reproducibility of results.

Apply the model to evaluate the probability that the sample image of the handwritten digit is a 4:

```
@torch.no_grad()
def predict(model, image_tensor):
    """Run inference on a single image."""
    model.eval()

    logits = model(image_tensor.unsqueeze(0).to(device))
    probs = torch.softmax(logits, dim=1).squeeze().cpu().numpy()
    pred = int(probs.argmax())
    return pred, probs


pred, probs = predict(model, sample_image)
print(f"Predicted class: {pred}   Actual class: {label}")
print(", ".join([f"{p:.3f}" for p in probs.tolist()]))
---
Predicted class: 6   Actual class: 4
0.000, 0.007, 0.000, 0.000, 0.268, 0.000, 0.722, 0.000, 0.003, 0.000
```

The model incorrectly identifies the image as a 6 with a probability of 0.722, although it also assigns a nontrivial probability to 1 (0.007) and 4 (0.268). This is because the handwritten 4 in the sample image has some features that are similar to those numbers, such as a vertical line that could be mistaken for a 1 and a loop that could be mistaken for a 6. (Even though we set the random seeds, your results may differ due to differences in hardware, software versions, and other factors that can affect the training process.)

This example illustrates the complexity and black box nature of deep learning models. There are more than 90,000 parameters to be trained in this relatively simple example. Indeed, models may have many more layers with between 5 and 150 million parameters.

KEY IDEAS

- CNNs are a type of deep learning architecture that uses convolutional layers to learn hierarchical features from data, particularly effective for image recognition tasks.
- CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers, which work together to extract and learn features from the input data.
- Transfer learning allows practitioners to leverage pretrained CNN models on large data sets, fine-tuning them for specific tasks with smaller data sets, thus saving time and computational resources.

Further Reading

- The ImageNet paper was presented at NeurIPS in 2012 before a standing-room-only audience and now has over 150,000 citations. This is the seminal paper that changed the field of computer vision virtually overnight.
- "Transfer Learning for Computer Vision Tutorial" from PyTorch.
- TensorFlow tutorial on transfer learning for image classification.

Transformers

Modern *transformer* models form the basis of common large language and multimodal data models such as GPT, Claude, T5, and Gemini. Their complexity and foundations are far removed from the statistical techniques, and even from the classical neural network models, described in earlier chapters. The models have from tens of millions to over a trillion parameters, requiring huge computing resources. Developing them is well beyond the expertise and resources of most data scientists, and the focus of this section is on the conceptual underpinnings of transformers. Transformers evolved from sequence-to-sequence (seq2seq) models, and the word *transformer* specifically highlights the model's ability to transform an input sequence into a rich, context-aware representation using self-attention rather than the recurrent steps used in earlier classical models.

The 2015 paper "Neural Machine Translation by Jointly Learning to Align and Translate" by Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio (often referred to as the "Bahdanau attention" paper) was a groundbreaking work that introduced the "attention" mechanism to RNNs. Bahdanau et al. were working on the problem of translating one natural language to another, specifically from English to French. RNNs had been used for translation, but their method of encoding each input sentence into a single vector was inefficient, especially for longer sentences. Bahdanau et al. introduced a dynamic alignment mechanism that allowed the model to focus on specific parts of an input sequence while generating each word of the output. Instead of one static vector, an encoder produces a sequence of "annotations" (hidden states) for every input word (see "Architecture Overview" for a definition of encoder). The model "soft-searches" using softmax over the input sentence to identify which words *align* most closely with (are most relevant to) the current word being translated. This meant that when translating a sentence, for example, the model could "look back" at relevant source words rather than relying solely on a representation of the entire sentence, dramatically improving performance on longer sequences.

The Bahdanau attention mechanisms, however, still used an underlying RNN architecture, which is inherently much slower since it can't be parallelized. In 2017, Google researchers published the seminal paper "Attention Is All You Need". This paper showed that RNNs weren't needed, and proposed an architecture using attention mechanisms that can be fully parallelized, taking advantage of the computational power of GPUs. Fast-forward to the present day: advancements in the capabilities of transformers have propelled technology companies into spending hundreds of billions of dollars in a race to build data centers capable of hosting the most powerful models.

KEY TERMS FOR TRANSFORMERS

Embeddings

The representation of words or other data elements as vectors in a continuous vector space, where semantically similar elements are mapped to nearby points

Attention mechanism

The heart of transformer models, allowing the model to weigh the importance of different parts of the input data when making predictions, enabling it to capture long-range dependencies and relationships in the data

Query

A vector representing the current token that "asks" a question to find relevant information in other parts of the sequence

Key

A vector that acts as a descriptive label for every token and its function in the sequence, allowing the *query* to determine how much attention to pay to it

Value

A vector containing the actual information of a token that is extracted and weighted based on how well its *key* matched the *query*

Self-attention

A specialized form of attention where the queries, keys, and values are all derived from the same input sequence

Masking

A technique that prevents the model from seeing all of the input, such as future tokens in a sequence during training, ensuring that predictions are made based only on available information

Multi-head attention

The core of modern transformer models in which multiple attention mechanisms work in parallel, each focusing on different parts of the input sequence to capture diverse relationships

Embeddings

Embeddings map the raw source data, such as words, into numerical values that can be used in models. A very simple example of this is in Chapter 4, where we used one-hot encoding to convert categorical variables into a numerical format. In this section, we focus on embeddings for language—the most widely known and common application—but embeddings can be used in different domains such as computer vision and DNA analysis.

Embeddings map words such as "king," "queen," and "aardvark" into vectors of numbers. Each number represents a different dimension. For simplicity, you might initially think of each dimension as representing an aspect of meaning. For example, on a "domination" dimension, "king" and "queen" might have high values, while "aardvark" has a low value. In reality, a dimension does not necessarily represent an obvious aspect of meaning. The dimensions are learned by the model, not specified by the model developer, and typically number in hundreds or thousands. These learned dimensions are typically abstract and not easily interpretable; meaning in a semantic sense they may be distributed across dimensions. For example, "domination" might not be among the learned

dimensions, though it might be represented by a composite of dimensions. When embeddings are learned by the model and encapsulated in hidden layers, they are sometimes referred to as *hidden state representations*.

Some words are highly related, but others have little or no relationship. You would expect "king" and "queen" to have similar embeddings, and "aardvark" to have a different one. Bengio et al. developed the first modern neural network word embedding in "A Neural Probabilistic Language Model" (2003). The original widely used embedding was Word2vec, introduced by Mikolov et al. at Google in 2013. Google was also central to the next big development, BERT (Bidirectional Encoder Representations from Transformers), which introduced bidirectional context-dependent embeddings.

Word2vec was revolutionary because of its:

Computational efficiency

Two simple architectures (continuous bag of words and Skip-gram) that were much faster to train

Scalability

Could train on billions of words in reasonable time

Quality

Produced high-quality embeddings that captured semantic relationships

Accessibility

Released as open source with pretrained models, making embeddings accessible to everyone

Modern transformers don't use Word2vec (except when very limited training data is available) since it produces a static embedding. Instead, they set initial random weights and learn context-dependent embeddings through training. Through training, embeddings learn multiple types of linguistic knowledge:

Semantic similarity

"dog" and "puppy" have similar vectors; "dog" and "table" are far apart.

Syntactic patterns

All verbs cluster together; adjectives form their own region.

Analogical relationships

The famous example is that vector("king") – vector("man") + vector("woman") ≈ vector("queen"), illustrating that embeddings can capture relational concepts like gender.

Contextual nuances

In modern contextual embeddings (like in BERT), the same word gets different embeddings based on context —"bank" in "river bank" versus "savings bank."

Embeddings play an important role in *information retrieval* systems. A query string or document is compared to target documents (both query and document are embedded) and target documents can be ranked according to how close they are to the query. Proximity is measured by standard metrics of statistical distance, such as those discussed in Chapter 7.

Attention Mechanisms

As the title of the eponymous Google paper ("Attention is All You Need") implies, at the heart of transformers lie attention mechanisms, and especially *self-attention* mechanisms, in which a sequence attends to itself. To

understand attention with a concrete example, we start with the sentence: “The cat sat on the mat.” Suppose we’re processing the word “sat” and want to understand its context using self-attention:

Query (from “sat”)

This represents what “sat” is looking for—essentially asking “What other words in the sentence are relevant to understanding me?” The query might encode something like “I’m a verb, who is my subject? What is the action being performed on?”

Keys (from all words)

Each word produces a key that represents what information, including its relative position, it can provide. For example:

- “The” key: “I’m a determiner”
- “cat” key: “I’m a noun, a potential subject or object”
- “sat” key: “I’m a verb”
- “on” key: “I’m a preposition indicating location”
- “mat” key: “I’m a noun, a potential object or location”

Values (from all words)

These contain the actual information that will be passed forward. The values are the semantic representations of each word that will be mixed together based on the attention weights.

When we compute attention, the “sat” query is compared against all the keys. The query-key comparison might reveal high scores for “cat” (the subject performing the action) and “on” (indicating where the action occurred), resulting in high attention weights for these words. Embedded tokens are mapped to vectors using the projection matrices W_q, W_k, and W_v, which are all learned. The attention weights are then used to compute a weighted sum of the value vectors, which produces a new representation for “sat” that incorporates information from the relevant words in the sentence.

Figure 9-6 depicts the structure of the basic attention mechanisms. Mathematically, the attention mechanism can be expressed as a matrix operation where the query vector is multiplied by the key vectors to produce attention scores, which are then normalized using a softmax function to produce attention weights:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K'}{\sqrt{d_k}}\right) \times V$$

where $\cdot$ denotes the dot product and $\times$ denotes matrix multiplication. Computationally, the attention mechanism involves the following steps:

1. Dot product

The query vector from “sat” is multiplied (dot product) with each key vector. For each word, we compute: $score = Q_{sat} \cdot K_{word}$. The dot product measures similarity—if two vectors point in similar directions (have similar patterns), their dot product is large; if they’re orthogonal, it’s small; or if they are opposite, it’s negative.

2. Scaling

The dot product scores are typically divided by the square root of the dimension of the key vectors d_k to prevent extremely large values that could cause gradients to vanish during training. This gives: $scaled_score = (Q_{sat} \cdot K_{word})/\sqrt{d_k}$.

3. Softmax

All the scaled scores are passed through a softmax function, which converts them into a probability distribution (positive values that sum to 1): $attention_weight = softmax(scaled_scores)$. This ensures that words with higher similarity scores get proportionally more weight.

4. Weighted sum

The attention weights are then used to compute a weighted sum of the value vectors. The output for "sat" is calculated as: $Attention(Q_{sat}, K, V) = \sum_i attention_weight_i \times V_i$, where the sum is taken over all words in the sentence.

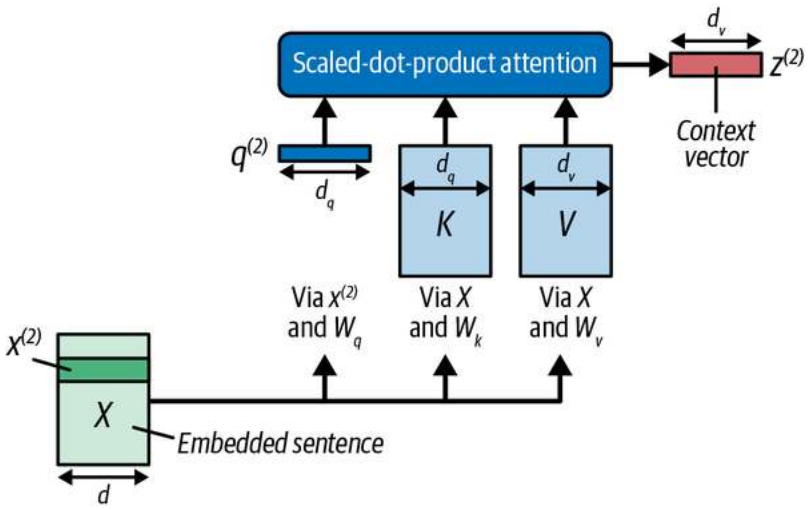


Figure 9-6. Structure of a basic attention mechanism: the embedded sentence matrix X (of dimension d) is projected through learned weight matrices W_q, W_k, and W_v to produce a query vector $q^{(2)}$ (from the second token $x^{(2)}$), a key matrix K, and a value matrix V, which are passed to the scaled dot-product attention mechanism to produce a context vector $z^{(2)}$ of dimension d_v that encodes the relevance of all tokens in the sentence to the query token.

We use a simple example of the sentence "The cat sat on the mat" to illustrate the intuition behind the attention mechanism. Suppose each token is represented by a vector of length 4 (in practice, transformers use much larger dimensions like 64, 512, or more, but we'll keep it simple). Numerically, the dot product $score = Q_{sat} \cdot K_{word}$ is calculated as follows:

$Q_{sat}(\text{query from "sat"}) = [0.8, 0.3, -0.5, 0.2] K_{cat}(\text{key from "cat"}) = [0.7, 0.4, -0.3, 0.1] K_{the}(\text{key from "the"}$

In the case of self-attention, the value vectors are the same as the key vectors, so we have $V_{cat} = K_{cat}$ and $V_{the} = K_{the}$.

The first step is to compute the dot product between each query vector and the key vectors. The dot product is a measure of similarity between two vectors and is commonly used in many statistical computations such as Pearson's correlation coefficient. The dot product is computed by multiplying corresponding elements and summing them. The dot product scores for "cat" and "the" with respect to "sat" would be calculated as follows:

$score(sat, cat) = (0.8 \times 0.7) + (0.3 \times 0.4) + (-0.5 \times -0.3) + (0.2 \times 0.1) = 0.85 score(sat, the) = (0.8$

The higher score of 0.85 for "cat" versus –0.22 for "the" indicates that "cat" is more relevant to "sat" in this context. We do this calculation for keys (including the key for "sat"). After scaling by $\sqrt{4} = 2$, we get 0.425 and –0.11 (it is standard to divide by the square root of the key dimension, which is 4 here, to keep the dot product from

growing to a magnitude that reduces gradients too much). Softmax then converts these (along with scores for all other words) into attention weights that sum to 1, with "cat" receiving a much larger weight than "the."

Suppose softmax attention scores for the query "sat" are as shown in Table 9-1.

Table 9-1. Attention scores for different words, with respect to the query "sat"

Word	Attention score	Interpretation
The	0.02	Low semantic value. Determiners rarely shift the meaning of "sat."
cat	0.45	Primary subject; "sat" needs to know who is sitting to resolve semantic meaning.
sat	0.15	Self-attention. Retains the "identity" of the action itself.
on	0.06	Functional link. Important for spatial relationship but less "meaning" than the noun.
the	0.02	Same as above for "The."
mat	0.30	Primary object/location. Defines the context of the action.

The final step is to compute the output for "sat" by taking a weighted sum of the value vectors using the attention weights. Values (which are the same as keys in self-attention) with the highest attention weights will be more influential in the final output representation for "sat." For example, if we focus on "cat" and "mat" as the most relevant words, the output vector for "sat" would be calculated as follows:

$$Attention(Q_{\text{sat}}, K_{\text{mat}}, V_{\text{mat}}) = 0.45 \times V_{\text{cat}} + 0.30 \times V_{\text{mat}} + \ldots (\text{contributions from other words})$$

The vector $Attention(Q_{sat}, K_{mat}, V_{mat})$ is a context-aware representation of "sat."

Architecture Overview

The original Transformer architecture consisted of two main components: an encoder and a decoder. The encoder transforms the input sequence into contextualized hidden representations, while the decoder processes and uses those representations to generate a corresponding output sequence. Modern transformer-based models, however, have evolved to use only the encoder (e.g., BERT) or only the decoder (e.g., GPT) depending on the task. The evolution of transformer architectures can be summarized as follows:

- 2017: Original Transformer (encoder-decoder)
- 2018: BERT (encoder-only) dominates
- 2019: GPT-2 (decoder-only) shows promise
- 2020: GPT-3 (decoder-only) demonstrates scaling
- 2021–2026: decoder-only becomes standard for LLMs

Since decoder-only architectures have been shown to be more effective for language generation tasks, we focus on this architecture in our discussion of transformers. The decoder-only architecture is designed to generate text by predicting the next word in a sequence based on the previous words, making it particularly well suited for tasks like language modeling and text generation. Decoders consist of multiple layers of self-attention and feedforward

neural networks. Modern transformer models typically have dozens of layers, with each layer containing multiple attention heads that allow the model to capture different types of relationships and patterns in the data (a *head* is a mechanism that computes attention). Each layer consists of the following components:

1. Masked multi-head self-attention

Each word attends to all previous words in the sequence, allowing the model to capture dependencies and relationships between words regardless of their position.

2. Feedforward neural network

After the self-attention mechanism, the output is passed through a feedforward neural network that applies nonlinear transformations to the data, enabling the model to learn complex patterns and representations.

3. Layer normalization and residual connections

These components help stabilize training and allow for deeper architectures by mitigating issues like vanishing gradients.

The masked multi-head self-attention utilizes the attention mechanisms described above, but with the addition of causal masking to ensure the model only attends to preceding tokens in the sequence. This prevents the model from "cheating" by looking ahead at future words during training. For example, in the sentence "The cat sat on the mat," when the model computes the representation for the word "sat," it is restricted to attending only to "The," "cat," and "sat" itself; the subsequent words "on," "the," and "mat" are masked out and unavailable. The "multi-head" aspect signifies that several of these attention mechanisms operate in parallel. Each head is initialized differently, allowing the model to simultaneously learn diverse types of relationships, such as grammatical dependencies or semantic similarities across the input sequence.

The overall architecture is depicted in Figure 9-7, which shows the flow of data through the layers of the decoder and how the attention mechanisms and feedforward networks are structured.

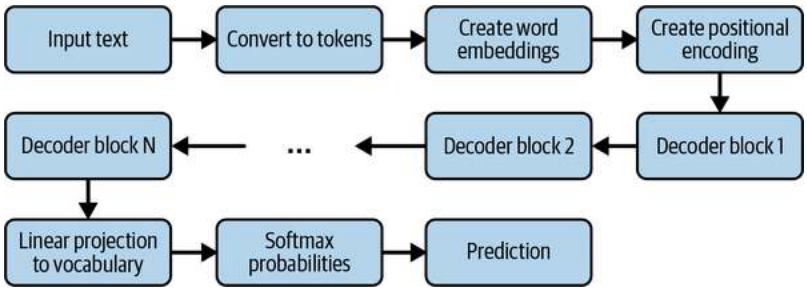


Figure 9-7. Structure of a decoder-only transformer architecture

There are many variations on this basic architecture, including modifications to the attention mechanism (e.g., sparse attention, linear attention), changes to the feedforward network (e.g., using different activation functions or adding additional layers), and adjustments to the overall structure (e.g., adding more layers, changing the size of the embeddings). These variations are beyond the scope of this book, but the key point is that the decoder-only architecture has become the standard for LLMs due to its effectiveness in generating coherent and contextually relevant text.

Revisiting the MNIST Image Classification Example

While the MNIST image classification example we covered earlier in this chapter used a CNN, it is also possible to apply transformer architectures to image classification tasks. Using the same image data as for CNNs, we can train a simple transformer model for image classification. Import the necessary libraries and set the key parameters in the model:

```
import torch
from torch import nn
import torch.nn.functional as F

device = torch.device(
    "cuda" if torch.cuda.is_available() else
    "mps" if torch.backends.mps.is_available() else
    "cpu")

IMG_SIZE = 28
PATCH_SIZE = 4  ❶
IN_CHANNELS = 1
NUM_CLASSES = 10 # ten digits (0-9)
EMBED_DIM = 64  # every token is a vector of 64
NUM_HEADS = 4  # number of attention heads running in parallel
NUM_LAYERS = 3 # number of transformer layers
MLP_DIM = 128 ❷
DROPOUT = 0.1 # during TRAINING randomly set 10% of values to zero
```

❶ The raw patch size is 4 × 4 pixels, but after flattening it becomes 16 (4 * 4).

❷ After attention, each token passes through a neural network with a hidden layer twice the size of the embedding dimension (128 versus 64).

Define the components of the transformer model for image classification. The model consists of a patch projection layer, learnable tokens and positional encoding, a dropout layer, transformer encoder layers, a normalization layer, and a final multilayer perceptron (MLP) head for classification:

```
class VisionTransformer(nn.Module):
    def __init__(self, img_size=IMG_SIZE, patch_size=PATCH_SIZE,
                 in_channels=IN_CHANNELS, num_classes=NUM_CLASSES,
                 embed_dim=EMBED_DIM, num_heads=NUM_HEADS,
                 num_layers=NUM_LAYERS, mlp_dim=MLP_DIM,
                 dropout=DROPOUT):
        super().__init__()

        # — Patch Projection —
        self.patch_projection = nn.Conv2d(
            in_channels, embed_dim,
            kernel_size=patch_size, stride=patch_size)
        num_patches = (img_size // patch_size) ** 2
        # — Learnable Tokens & Positional Encoding —
        self.cls_token = nn.Parameter(torch.zeros(1, 1, embed_dim))
        self.pos_embedding = nn.Parameter(
            torch.zeros(1, num_patches + 1, embed_dim))
        self._reset_parameters()

        self.dropout = nn.Dropout(dropout)
        # — Transformer Encoder —
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=embed_dim,
            nhead=num_heads,  ❶
            dim_feedforward=mlp_dim,
            dropout=dropout,
            batch_first=True,
            activation="gelu",
        )
        self.transformer = nn.TransformerEncoder(
            encoder_layer, num_layers=num_layers)
        # — Norm Layer —
        self.norm = nn.LayerNorm(embed_dim)
        # — MLP Layer for Classification —
        self.mlp_head = nn.Sequential(nn.Linear(embed_dim, mlp_dim), nn.GELU(),
            nn.Dropout(dropout), nn.Linear(mlp_dim, num_classes))

    def _reset_parameters(self):  ❷
        nn.init.trunc_normal_(self.cls_token, std=0.02)
        nn.init.trunc_normal_(self.pos_embedding, std=0.02)
```

```
    def forward(self, images):
        B = images.shape[0]

        # ── Step 1-2: Extract patches and project to embed_dim ❸
        # (B, 1, 28, 28) → (B, 64, 7, 7) → (B, 49, 64)
        x = self.patch_projection(images)  # (B, EMBED_DIM, 7, 7)
        x = x.flatten(2).transpose(1, 2)  # (B, NUM_PATCHES, EMBED_DIM)
        # ── Step 3: Prepend CLS token ──
        # (B, 49, 64) → (B, 50, 64)
        cls_tokens = self.cls_token.expand(B, -1, -1)
        x = torch.cat([cls_tokens, x], dim=1)
        # ── Step 4: Add positional encoding  ❹
        x = x + self.pos_embedding
        x = self.dropout(x)  ❺
        # ── Step 5: Transformer encoder ──
        x = self.transformer(x)
        # ── Step 6: CLS token → MLP head ──
        cls_output = self.norm(x[:, 0])  # (B, 64) ❻
        return self.mlp_head(cls_output)  # (B, 10) ❼

model = VisionTransformer().to(device)
```

❶ The 64 dimensions are split into 4 heads, each looking for a different feature; each head works on 64 / 4 = 16 dimensions.

❷ Instead of starting with all zeros, the tokens and embeddings start out with small random values (to avoid exploding/vanishing gradients).

❸ A raw patch from a 4 × 4 pixel region is just 16 numbers (1 channel × 4 × 4). This is too small and not rich enough for the transformer to work with, so the projection stretches each patch from 16 → 64 dimensions. After patch projection, we have 49 patch vectors.

❹ In the transformer layers, the CLS token asks each patch: “What information do you have?” and collects this information. Position embedding adds a unique “location tag” to each patch.

❺ A dropout layer is added during training to randomly set 10% of the values to zero, which helps prevent overfitting by encouraging the model to learn more robust features.

❻ The normalization layer ensures each token has a mean of 0 and standard deviation of 1.

❼ The final layer takes the CLS token (a vector of 64 numbers) and converts it into 10 class scores.

```
print(f"Total Parameters: {sum(p.numel() for p in model.parameters()):>8,}")
---
Total:  114,506
```

The total number of parameters is 114,506 with 87% in the transformer layers, 10% in the MLP head, and 3% in the patch projection and learnable tokens/embeddings.

The forward path has six steps:

1. Split image into patches
2. Project patches to `embed_dim`
3. Prepend CLS token
4. Add positional encoding
5. Pass through transformer encoder
6. Extract CLS token → MLP head → class scores

Train the model by iterating over five epochs. We make use of the functions `train_one_epoch` and `evaluate` that we defined in the CNN example:

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

num_epochs = 5
for epoch in range(num_epochs):
    avg_loss = train_one_epoch(model, trainloader, optimizer, criterion)
    accuracy = evaluate(model, testloader)

    print(
        f"Epoch [{epoch+1}/{num_epochs}]  "
        f"Loss: {avg_loss:.4f}  "
        f"Test Accuracy: {accuracy:.2f}%"
    )
---
Epoch [1/5]  Loss: 0.8796  Test Accuracy: 92.35%
Epoch [2/5]  Loss: 0.2605  Test Accuracy: 95.97%
Epoch [3/5]  Loss: 0.1898  Test Accuracy: 95.95%
Epoch [4/5]  Loss: 0.1550  Test Accuracy: 96.55%
Epoch [5/5]  Loss: 0.1342  Test Accuracy: 97.07%
```

The accuracy of the transformer model is almost the same as the CNN model.

Apply the model to predict the same image as in the CNN model:

```
sample_image, label = testset[160]
pred, probs = predict(model, sample_image)
print(f"Predicted class: {pred}   Actual class: {label}")
print(", ".join([f"{p:.3f}" for p in probs.tolist()]))
---
Predicted class: 4   Actual class: 4
0.000, 0.000, 0.000, 0.000, 0.998, 0.000, 0.000, 0.001, 0.000, 0.000
```

In contrast to the CNN model, the transformer model predicts the correct class (4) for this image and it is confident in its prediction (99.8% probability for class 4). While both models achieve similar overall accuracy, the transformer model's prediction for this specific image is correct, illustrating that different architectures may have different strengths and weaknesses in terms of which examples they classify correctly or incorrectly.

Like the CNN example, this example also illustrates the complexity and black box nature of deep learning models. There are more than 114,000 parameters to be trained in this relatively simple example. Transformer models would typically have many more layers and have 5 to 150 million parameters.

KEY IDEAS

- Transformers are a type of deep learning architecture that relies on attention mechanisms to capture relationships and dependencies in data, particularly effective for natural language processing tasks.
- Transformers are used in all modern LLMs such as GPT, Claude, and Gemini.
- Key components of transformers include embeddings, attention mechanisms, and multi-head attention, which allow the model to learn complex patterns and relationships in the data.
- Transformers were made possible by the availability of large data sets and powerful computing resources (especially GPUs), enabling the training of models with billions or even trillions of parameters.

Further Reading

- The seminal paper "Attention is All You Need" that introduced the transformer architecture is a must-read for anyone interested in understanding the foundations of modern LLMs. It provides a detailed

explanation of the attention mechanism and how it enables the model to capture long-range dependencies in data.

- *Hands-On Generative AI with Transformers and Diffusion Models* by Omar Sanseviero, Pedro Cuenca, Apolinário Passos, and Jonathan Whitaker (O'Reilly, 2024) provides a practical introduction to transformer architectures and their applications in generative AI, including hands-on examples and code snippets.
- *Build a Large Language Model (From Scratch)* by Sebastian Raschka (Manning, 2024) offers a detailed walkthrough of building a transformer model from scratch, providing insights into the architecture and implementation details.

Reinforcement Learning

In the supervised learning techniques we covered earlier in this book, the model learns from a specified set of training data. In *reinforcement learning*, the model learns as it proceeds, on the basis of feedback from its actions. The model is given a goal, takes actions, and receives feedback in the form of rewards or penalties. The model learns to take actions that maximize the cumulative reward over time. AlphaGo and AlphaZero, both developed by DeepMind (now Google DeepMind), are famous examples of reinforcement learning models that have achieved superhuman performance in the games of Go and chess, respectively. They were landmark examples of reinforcement learning because the systems improved by playing millions of games against themselves, optimizing a reward signal (win/loss) without relying on user-designed strategies.

KEY TERMS FOR REINFORCEMENT LEARNING

Reward

Feedback that an automated agent receives from the environment after taking an action; *penalty* is a negative reward

Reinforcement learning

A type of machine learning where an agent learns to make decisions by taking actions in an environment to maximize cumulative rewards

Agent

The learner or decision-maker algorithm that interacts with the environment and learns from the feedback it receives

Multi-arm bandits

A continuous experiment in which an agent must choose between multiple options (arms) with unknown rewards, balancing information-gathering (exploration) and exploitation to maximize rewards

Synonym

bandits

Markov decision processes

A framework for modeling and optimizing decision making, taking account of states, actions, rewards, and policies

Deep reinforcement learning

Reinforcement learning in which deep neural networks are used to approximate value functions or policies, enabling the handling of complex, high-dimensional environments

Multi-Arm Bandits

Multi-arm bandits (or, simply, *bandits*) were an early implementation of reinforcement learning. The name derives from a gambler playing multiple slot machines, also called "one-armed bandits," each with a different payout rate. The gambler's process is to try out the different machines (arms) multiple times with the goal of figuring out which has the highest payout rate. The gambler faces a trade-off between *exploration* (trying out different machines to gather information about their payout rates) and *exploitation* (using the machine that has the highest known payout rate to maximize rewards). This is a more efficient way of exploring different *treatments* (a treatment is equivalent to a machine arm) than the classic statistical tests we discussed in Chapter 3. In those traditional tests, the different treatments would be applied to a set number of subjects, and then, at the end of the trial period, the results would be assessed. There is no opportunity to adapt to the information gained along the way.

Agents (the automated players) deploy bandit algorithms to use the intermediate information to adjust the probabilities of selecting each machine arm. Bandits use different user-specified strategies to balance exploration with exploitation as they try to optimize the reward signal. For example, the *Epsilon first* strategy explores randomly for a set period of time, then exploits the arm (treatment) with the highest payout rate for the rest of the time. The *Epsilon greedy* strategy explores randomly with a small probability (epsilon) at each step, while exploiting the best-known arm most of the time. The *Upper Confidence Bound* strategy selects arms based on both

their estimated payout rates and the uncertainty around those estimates, favoring arms that have been explored less.

Sometimes the reward will change depending on context. For example, in music streaming, the reward (say a *like* or a download) may depend on the time of day (waking up, exercising, etc.). *Contextual bandits* take this context into account when selecting arms, allowing for more personalized decision making.

Multi-arm bandits are popular with digital marketers, who use them to optimize the presentation of online ads and product recommendations. Related, more constrained versions are used in clinical trial design, where the goal is to identify both effective and ineffective treatments as fast as possible while maintaining statistical validity. Bandits and contextual bandits are implemented in the Python package `contextualbandits`.

Markov Decision Processes

Multi-arm bandits are a special case of *Markov decision processes* (MDPs), named after the Russian mathematician Andrey Markov. In an MDP, the algorithm learns a longer-term relationship between decisions and outcomes, focusing on *states* and *rewards*. In chess, for example, the state at any point of play is the configuration of pieces on the board, and the reward is winning or losing the overall game. States should include relevant history. For example, in optimizing music streaming recommendations, the state would include prior user activity (clicks, plays, shares), as well as demographics and other static features. The agent's actions affect both the next state and the ultimate reward. The goal of the agent is to learn a *policy* that maps states to actions in a way that maximizes the expected cumulative reward over time.

MDPs are sequential processes, where each step in the sequence is defined by the current state, the action taken, and the instantaneous reward. The agent's goal is to maximize expected return over time, which is the sum of discounted instantaneous rewards. The discount factor (gamma) determines the importance of future rewards compared to immediate rewards.

Consider the following simple inventory illustration (presented by Phil Winder in *Reinforcement Learning*, O'Reilly, 2021). A shop has a maximum inventory of 2 units. Each day, the shop can order 0, 1, or 2 units of inventory. Sales and restocking may happen on a daily basis, with the shop either selling 1 unit (probability = 0.7) or 0 units (probability = 0.3). Maximum stock is 2, so no restocking takes place if the shop already has 2 units. If a sale is made, the shop receives a reward of 1. If no sale is made, the reward is 0.

Daily activity can be represented by a transition table (see Table 9-2). Each row shows a possible initial state, an action, the next state, the probability of that transition, and the immediate reward.

Table 9-2. A transition table for the inventory example

State_t	**Action**	State_{t+1}	$P(\text{State}_{t+1}\|\text{State}_t, \text{Action})$	**Reward**
0	None	0	0.3	0
0	None	0	0.7	0
0	Restock	1	0.3	0
0	Restock	0	0.7	1
1	None	1	0.3	0
1	None	0	0.7	1
1	Restock	2	0.3	0
1	Restock	1	0.7	1
2	None	2	0.3	0
2	None	1	0.7	1
2	Restock	2	0.7	1

For example, row 3 shows one of the possible transitions: from an initial state of 0 stock, a decision to restock, a next state of 1, a probability of 0.3 for this transition, and a reward of 0 (since no sales were made). Row 4 covers the case where a sale happened, starting from the same initial state of 0 stock. Due to restock (+1) and sale (–1), the new state is still 0. The remaining columns are: a probability of 0.7 for this transition, and a reward of 1 due to the sale.

This same information can be shown in a transition matrix, shown in Table 9-3 and Table 9-4, less easy to visualize but closer to the form used in the algorithm. These illustrate transition probabilities from s (row headers) to s' (column headers).

Table 9-3. No restock transition matrix P(s'|s)

s → s'	**0**	**1**	**2**
0	1	0	0
1	0.7	0.3	0
2	0	0.7	0.3

Table 9-4. Restock transition matrix P(s'|s)

s → s’	0	1	2
0	0.7	0.3	0
1	0	0.7	0.3
2	0	0	1

From this information (initial state, action, transition probabilities, reward), MDPs define a *value function* measuring the expected value of each state, which is the expected cumulative reward starting from that state and following the optimal policy. In this simple example, the optimal policy can be determined by iterating through the transition table and evaluating candidate restocking policies using the transition table until the expected cumulative reward is maximized.

The optimal value function is expressed by the *Bellman equation* for the value of a state under an optimal policy:

$V_*(s) = \max \text{ over } [\text{all possible actions}] \text{ of } E\left[r + \gamma \times V_*(s')\right]$

where $V_*(s)$ is the optimal value of state s, r is the immediate reward, γ is the discount factor, and s' is the next state resulting from taking an action in state s. Although it appears to account only for the current state and the next state, note that $V_*(s')$ is determined recursively, i.e., it necessarily incorporates one step beyond and thus accounts for the entire future sequence of states and rewards. Typically, rewards in the future sequence are discounted, so, although the sequence may extend indefinitely, the value of the rewards diminishes to a point where the sequence converges to a finite value.

Deep Reinforcement Learning

In the numerical example discussed in the preceding section, the states, actions, rewards, and transition probabilities are simple, known, and easy to visualize. In more complex real-world problems, the state space can be very large or continuous, the transition probabilities may be unknown, and the reward structure may be complex. This is where deep learning comes in, as it can be used to approximate the value function without needing to explicitly model or simulate all the possible states and transitions.

In deep reinforcement learning, the value function (or sometimes the policy) is represented by a neural network that takes the current state as input and produces as output either an action or a probability distribution over possible actions. After interacting with the environment, the network’s weights are adjusted so that actions leading to higher long-term reward become more likely, while less successful actions become less likely. Through repeated experience and incremental weight updates, the network gradually improves its behavior toward maximizing expected discounted cumulative reward.

In reinforcement learning, data is sequential in nature. More specifically, reinforcement learning operates on trajectories: state → action → next state → reward, repeated over time. Each action reshapes the distribution of future inputs. Deep models such as RNNs or transformers can incorporate temporal context directly, allowing the policy to condition on prior observations when the environment does not fully reveal its state. Thus, optimization occurs over sequences of interactions rather than isolated examples.

Here it is useful to highlight a key distinction between reinforcement learning and supervised learning. Unlike in supervised learning, where the data distribution is external to the model, in reinforcement learning the agent’s policy determines which states it will encounter in the future. Thus, learning and data collection are intertwined: as the policy changes, so does the training distribution.

KEY IDEAS

- Reinforcement learning (RL) is different from the statistical and machine learning methods discussed to this point in that RL involves interaction between the learner and the environment.
- Reinforcement learning from human feedback (RLHF), where the *environment* consists of human preferences, is used in the context of large language models. In this process, humans rank different model responses, and those rankings are used to train a reward model.
- RL receives rewards from the environment and uses this information to make better decisions.
- Multi-arm bandits and Markov decision processes are constrained frameworks for modeling RL problems.
- Deep RL uses deep neural networks to approximate value functions or policies, yielding more powerful models that can handle high dimensionality and more complex environments.

Further Reading

- *Bandit Algorithms for Website Optimization* by John Myles White (O'Reilly, 2012) is a short, highly accessible treatment of multi-arm bandits.
- *Reinforcement Learning* by Phil Winder (O'Reilly, 2021) provides a comprehensive introduction to reinforcement learning, covering both foundational concepts and practical applications, with a focus on deep reinforcement learning.

Summary

This chapter gave an overview of deep learning and its evolution from RNNs to CNNs to transformer architectures. We discussed how convolutional neural networks (CNNs) work by learning hierarchical features from data, particularly in image recognition tasks. We provided an example of how CNNs can be used for image classification and how transfer learning allows practitioners to leverage pretrained models for specific tasks. We then introduced transformers, which rely on attention mechanisms to capture relationships in data, and discussed their architecture and key components. We touched on the importance of large data sets and powerful computing resources in enabling the training of modern transformer models. Finally, we reviewed reinforcement learning, in which algorithms interact with the environment and, via rewards and penalties, optimize decision policies. In the next chapter, we explore how these technologies evolve beyond prediction and decision optimization and become tools for generating text and images.

Exploration with AI

- Ask your AI for a Python or R example of an LSTM network for time-series forecasting. How does it compare to a random forest or boosted trees on the same data?
- Ask your AI to translate the CNN example in this chapter to R or using a different framework in Python (e.g., `tensorflow`).
- In this chapter, we covered the decoder architecture for transformers (e.g., GPT). Ask your AI to explain the differences between encoder-only, decoder-only, and encoder-decoder architectures, and provide examples of tasks for which each architecture is best suited.
- Ask your AI to produce a simple example of how the attention mechanism works in a transformer model, using a small sentence and showing the calculations for the query, key, and value vectors, as well as the

resulting attention weights.

- We didn't provide a detailed explanation of how the softmax temperature parameter works in transformer models. Ask your AI to explain how the temperature parameter modifies the softmax function and how it influences the sampling process during text generation.
- Ask your AI to produce code for simple machine translation from one language to another using a transformer model, and see how the attention mechanism helps the model to align words in the source language with words in the target language during translation.
- Ask your AI to explain the different strategies for balancing exploration and exploitation in multi-arm bandits, and to review their comparative strengths and weaknesses.
- In both online marketing and in drug development, the goal is to select the best-performing "treatment" (e.g., ad, drug) as quickly as possible. Ask your AI to explain the important differences between these two domains, and how these differences affect the choice of algorithm for optimizing decisions. (Hint: regulatory considerations are paramount in drug development. See if your AI highlights this in its response. If it fails to stress this angle and to explain the implications, ask it to do so.)

---

[1] "A Fast Learning Algorithm for Deep Belief Nets" with Osindero and Teh, and "Reducing the Dimensionality of Data with Neural Networks" with Salakhutdinov.

Chapter 10. Generative AI and Large Language Models

In this chapter,[1] we take machine learning well beyond inference and prediction, and into the realm of creating. Responding to instructions, generative AI algorithms can create new verbal content (written or spoken) and produce new images, videos, and audio (including music).

No machine learning technology has generated as much popular fanfare as generative AI. Stories about it appear on a daily basis in the business and popular press. Generative AI's popular profile was enhanced by the public availability of powerful generative AI tools such as OpenAI's ChatGPT, with which anyone can type in a prompt and generate impressive results.

THE TURING TEST

Alan Turing was a British mathematician and computer science pioneer who, among many other achievements, played a key role in breaking German ciphers used during WWII. This work contributed to the Allied victory in the Battle of the Atlantic. The Turing test, proposed by Alan Turing in 1950, is a benchmark for evaluating a computer's communication ability. In the test, a human evaluator interacts with both a human and a machine through written communication, without knowing which is which. If the evaluator cannot reliably tell the machine from the human, the machine is said to have passed the test.

Generative AI models are generally agreed to be the first computer technology to have passed the Turing test. In fact, in a randomized experiment in 2025, ChatGPT-4.5 was identified as human more than twice as often as the actual human subject.

Generative AI's capabilities have made it a powerful tool for businesses and other organizations.

Generative AI and LLMs

Generative AI is a class of models that have the ability to produce text, images, audio, or video on the basis of a user-supplied *prompt* (a written query, request, or instruction). It is an extension of supervised learning and of the neural nets used in deep learning (see Chapter 8 and Chapter 9).

KEY TERMS FOR GENERATIVE AI

Large language model

A machine learning model that can generate lengthy text output in report or narrative form

Synonym

LLM

Temperature

A setting that controls the randomness (creativity) of the output of a generative AI model

Synonym

creativity setting

Models that produce written text, such as OpenAI's ChatGPT and Google's Gemini, are called large language models (LLMs). The providers of public versions of these models say they were trained on a wide variety and huge volume of written content available online, including websites (Wikipedia and many others), books, news articles, and more. However, the companies

do not disclose specifics about what has been used in training (for example, they are cagey about their use of social media).

Predicting the Next Word

LLMs have at their heart the supervised learning task of predicting the next word, given a prompt. In itself, this is an old idea—it has stood behind autocomplete features for decades. Autocomplete algorithms answer the question "Given a user-supplied set of words or letters, what words or letters are likely to come next?" What is new about LLMs, compared to earlier autocomplete algorithms, is the expanded use of context: each generated next word is based not just on the immediate prompt but also on the lengthy (sometimes a page or more) sequence of words that the LLM has already generated in response to the prompt. Second- and later-generation LLMs also take into account additional factors:

- An optional user-assigned role (for example, *you are a university professor*, or *you are a thriller writer*). This influences the style and content of the response.
- The prior exchanges in the conversation between the LLM and the user. This keeps the conversation coherent and enriches the content.
- Prior conversations (perhaps spanning sessions) between the LLM and the user. This helps the user who is engaged in a long-term project.
- External sources of information, such as a knowledge base or the internet. This helps keep the LLM's responses current and accurate.
- The ability to write and execute code. This helps the LLM to return accurate results for numerical and data-related queries.

LLM-generated sequences must not only answer the user's query but also follow the rules of grammar and syntax. They can follow a style (outline,

summary, list, table, literary genre) dictated by the prompt.

Creativity and Temperature

Typically, the default settings for an LLM call for it to produce the most likely next word, given the prompts and context. However, there are usually multiple possible completion words, with different probabilities attached to each. In some situations, it may be useful to instruct the model *not* to always select the most probable word. For example, if you are writing advertising copy and want to avoid repetition, you may want the LLM to be more creative by sometimes choosing less likely words. This is done by adjusting a parameter called *temperature*. A low temperature (close to zero) means the model will almost always choose the most likely next word. A high temperature (say, above 1.0) means the model will choose less likely words more often, resulting in more diverse and creative outputs. A temperature of 1 means the model will choose the next word with probability proportional to its estimated likelihood. A higher temperature decreases that probability (i.e., it increases randomness), while a lower temperature makes the output more deterministic, and less likely to "make things up" or hallucinate. *Hallucinations* are a recognized problem with LLMs, and a primary reason for keeping a human in the loop while using them (see the section "Alignment" later in this chapter).

Temperature can be set via API calls. For example, in OpenAI's API, you can set the temperature parameter in your request body like this:

```
{
  "model": "gpt-4",
  "prompt": "Write a poem about the ocean",
  "temperature": 0.9
}
```

Effects similar to temperature changes can be achieved by the nature of the prompt and its effect on the distribution of possible next words, rather than on the mechanism of selecting the next word. A prompt that asks for a poem would automatically generate a creative response (via a broadening of

the distribution of possible next words), while a prompt seeking a summary of a financial report to regulators would (one hopes) need less creativity (a narrower distribution of possible next words).

KEY IDEAS

- Generative AI is able to produce lengthy text as well as images in response to a prompt.
- It is an extension of the text-mining capability of predicting the next word.
- The creativity of the output can be controlled by a temperature setting.

Learning Language and Generating Images: Attention

How do LLMs learn language and master content? We have already introduced one of the key building blocks in Chapter 9: deep learning. That building block was the concept of learning higher-level features. We discussed it in the context of "understanding" images by moving from individual pixel values to representing local features like edges, then higher-level features like shapes, and even higher-level features like faces. We applied neural networks to text and introduced the concept of *word embedding*, in which each *token* (word or term) is represented by a vector encoding its numerical scores on numerous dimensions, which the network has learned are useful representations of meaning. We also saw how these vectors, representing individual words, could then be used in both predictive algorithms and unsupervised learning.

KEY TERMS FOR LEARNING LANGUAGE

Attention

A process for incorporating context from distant words in a text sequence

Token-pair

A pair of tokens (words) from a document, typically one of all such possible pairs

Context window

A subset of a document used to compute attention scores

Text mining deep learning networks, including the RNN and LSTM networks that were introduced in Chapter 9, focus on neighboring words to contribute context that improves predictive performance.

Generative AI requires moving beyond the context of the immediate neighborhood. If you consider that the output of a generative AI model may constitute extensive narrative text and require command of the rules and structure of a language, you can see that the algorithm must take into account words that might be quite distant from one another. To be effective and computationally feasible, the model must still have a way of focusing on relevant content, whether it's adjacent or distant. The mechanism for doing this is called *attention*. With the word embedding vectors as building blocks, the algorithm goes through a series of matrix operations to compute attention scores for each token-pair within a context window (subset of the corpus) whose size is determined by the model and its architecture. Each token then ends up with a set of scores that reflect the relevance of each other token in the data. These attention scores are converted to weights and used to revise the original word embedding in a way that gives greater weight to more relevant information, no matter where it may be located in

the window. The computation of attention scores happens both in training, when weights are updated, and in inference (use of the model, no weight updating).

The need to compute a score for each token-pair makes model training and inference extremely computationally intensive. A modification termed *sparse attention* reduces the number of token-pairs that need to be scored, reducing this computational burden. There are several ways to reduce computation. For example, for a given token, tokens that are nearby might all be paired up and scored, while for more distant tokens, only every *n-th* token is paired. Another approach is to use a sliding window, where only tokens within a certain distance are paired. In late 2025, the Chinese maker of DeepSeek sought market advantage by reducing the cost of API calls to its LLM by 50%, which it claimed it was able to do by using sparse attention.

KEY IDEAS

- *Attention* is the process by which the LLM, in predicting the next token, incorporates context from a broad context window, not just from nearby tokens.
- The LLM does this using attention scores for each token-pair, reflecting their relevance to each other.

Further Reading

"Attention is All You Need" by Vaswani et al., 2017, is the seminal paper introducing the attention mechanism that underpins most LLMs.

Reinforcement Learning from Human Feedback

As we saw earlier in this book, the foundation of supervised machine learning is training a model to match its predictions to known outcomes. In generative AI, the outcome is not a “known” outcome but rather a word sequence that is useful to a human user.

KEY TERMS FOR REINFORCEMENT LEARNING FROM HUMAN FEEDBACK

Reinforcement learning from human feedback

The incorporation of human feedback as rewards to improve LLM performance

Synonym

RLHF

Alignment learning

The general process of making model output consistent with some benchmark

Reward

A score given to model output that reflects human preferences

Reward model

A model trained to predict and economize on scarce human feedback

Direct preference optimization

Like a reward model, except it's folded into the main model, not kept separate

Synonym

DPO

Reinforcement learning from AI feedback

The use of AI-generated feedback as rewards to improve LLM performance

Synonym

RLAIF

The process of making model output consistent with a benchmark is termed *alignment learning*. A frequent benchmark is human preferences, and for this, human rating of model outputs is needed. The core of this rating is human review (often paid) of model output. Early LLMs were trained with prompts matched to human-supplied answers; the LLM was trained to try to match the answer. This is an example of *reinforcement learning from human feedback* (RLHF), in which models interact with reward functions to optimize actions and decisions. AlphaZero's mastery of chess, where winning is the reward, is a classic example of reinforcement learning. In RLHF the key ingredient in the reward function is *human* feedback.

Alignment

Reinforcement learning with human feedback is an important tool in the process of *alignment*—making model outputs more in line with human needs. This means making them more helpful, more truthful, and safer. Safety is partly a subjective concept, but it might include things like avoiding telling users how to construct a bomb. These key elements of alignment are often summarized as *helpful, honest, and harmless*, or HHH. There is a distinction to be made between human needs as expressed by users, as with RLHF, and human needs as specified by model developers, as is sometimes done to make models more ethical. The latter are sometimes hard-coded, as with Google Photos' elimination of *gorilla, chimpanzee, and monkey* as potential labels, because the photo recognition algorithm was labeling dark-skinned individuals with these labels.

Reward Models

Human ratings are expensive and, in machine learning terms, scarce. During training, an LLM might make hundreds of passes through its network (which has hundreds of billions of weights or parameters) per second. Clearly it cannot stop for human input on the prediction each time! Therefore, an artificial method of awarding scores to output was developed to supplement the human ratings. This method is called a *reward model*. The reward model (separate from the main model) is trained on human ratings to predict human ratings on future LLM output. This process produces scores to be applied to LLM outputs during the LLM's training. The human ratings used for training the reward model are relatively sparse —the reward model might yield tens of thousands of scores for every human score.

Reward models also incorporated rating systems that encouraged more creativity by the LLM, with raters giving model outputs a rank score, or a choice between two responses. A model could also be evaluated using certain triggers in the conversation, such as "I did not understand what you said about…." These devices can all be incorporated into the normal flow of LLM use without requiring dedicated paid reviewers, substantially increasing the quantity of real human feedback.

Later processes such as direct preference optimization (DPO) bypassed the separate training of a reward function: when the main model itself is trained, the loss function is trained so that it rewards model outputs in accord with human preferences.

Still later developments include *reinforcement learning from AI feedback* (RLAIF), in which rewards are based on feedback from another AI model rather than from humans. RLAIF is another way to economize on scarce human feedback, which is reserved for upstream tasks—the establishment of the rules by which RLAIF is to operate, and, in some cases, the fine-tuning of the AI model providing feedback. It also has the advantage of speed (LLMs are faster than humans) and structural scalability (the process, once established and tuned, can readily be adapted across tasks). The absence of human feedback in RLAIF cuts two ways: on the one hand,

human preferences and values are lost; on the other hand, a consistent set of values and governing preferences may be enforced.

The difference between RLAIF on the one hand, and RLHF, reward model training, and DPO on the other, is mainly the scale and position of the human feedback. In contrast to the latter methods, RLAIF uses human feedback only to set and tune the rules and parameters for the AI feedback model, and is not part of the main training process.

KEY IDEAS

- Reinforcement learning from human feedback (RLHF) uses human preferences as rewards to guide LLM learning.
- These human preferences, which are scarce, are augmented by predicted preference scores.
- Reinforcement learning from AI feedback (RLAIF) substitutes AI feedback for human feedback in the training process.

Further Reading

Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto, Bradford Books (first edition 1998, second edition 2018)

Image Generation

Perhaps no AI capability captures the popular imagination more than image and video creation. Most LLMs can now generate images and video as well as text. The company TikTok has built its entire business model around short videos, many of which are now generated by AI. In promoting his own AI model, Grok, on his social media platform, X, Elon Musk tweets mostly about its video creation ability. It is notable that deep learning also

saw its initial successes in the fields of natural language processing and computer vision.

KEY TERMS FOR IMAGE GENERATION

Diffusion methods

Models that learn to iteratively denoise a randomly perturbed image, gradually refining it into a realistic output

Reference image

A real image (i.e., nonrandom) to which the model iteratively compares its generated image during training

Seed image

An image, typically of randomly generated pixels, that starts the process of image generation

Denoising

Modifying a generated image to bring it closer to the actual image

Embedded text vector

A numerical vector representation of text that captures its higher-level features

Conditioning

The process of incorporating text input into the image generation process

Models that produce images on the basis of prompts combine text processing capabilities with image generation. As with LLMs, the key lies in higher-level feature learning. Recall from Chapter 9 that higher-level

features in images are represented by numerical vectors. The same is true for text, as we saw in our discussion of text mining. So both images and text can be viewed by AI through the same lens: numerical vectors that embody meaning. The challenge in image generation is to bring these two different modalities together so that the model can generate images that correspond to text descriptions. A popular architecture for image generation, *diffusion methods*, works roughly as follows:

1. A set of real images with text descriptions is assembled as a reference base.
2. The text descriptions are converted to embedded text vectors (see Chapter 9).
3. A random noise seed image is created by generating random pixel values.
4. Iteratively and gradually, the random image is *denoised* by repeatedly modifying it in a way that brings it closer to the real image while incorporating the embedded text description in the process, an approach called conditioning.
5. In the training phase, a loss function guides the training process by comparing the denoised image to the reference image.
6. Once the model is trained, it works without reference images, simply generating images that are close to the text descriptions.

Integrating Text Input

A key component of this process is incorporating the user's text input. In our discussion of text mining (see Chapter 9), we saw that text embedding converts text to a numerical vector that represents the latent high-level features in the text, analogous to the autoencoder in Figure 10-1.

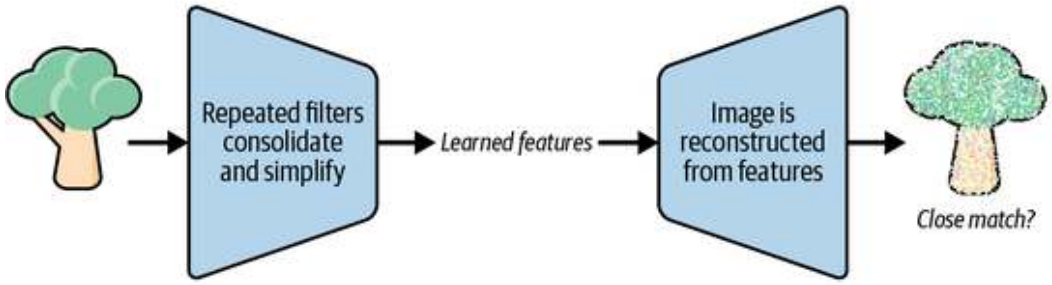


Figure 10-1. Derivation of higher-level features in autoencoder

The higher-level features for both the image and the text prompt are now represented by numerical vectors, and, in effect, can talk the same language. The use of these high-level representations to match images to appropriate text descriptions is a form of alignment learning. Of course, these layered procedures yield an overall process that is extremely computationally intensive—it is easy to see how text and image generation are driving extraordinary demand for computing resources. At the time of this writing, diffusion methods are the dominant image generation method. Earlier related architectures include generative adversarial networks or GANs, and variational autoencoders. GANs also work with initial images that are random noise, but they have a separate component called a discriminator whose explicit function is to identify whether an image is real or fake. Variational autoencoders generate the initial seed images (the noisy image that gets turned into a realistic image) by sampling and generating from the latent high-level feature components in the center of Figure 10-1 rather than by simply generating random pixel values.

KEY IDEAS

- Models are first trained to iteratively modify random pixels to match a reference image.
- Then, without reference images, they generate images to match a text description.
- The key is that both images and text are represented by higher-level feature vectors that "speak the same language."

Further Reading

"Denoising Diffusion Probabilistic Models" by Ho et al., 2020

Scale and Power

Generative AI models require massive computing power for both training and use, and at a scale well beyond anything needed by traditional statistical and machine learning models working with structured data. The latter might need seconds or minutes of computing time on a single computer. In contrast, training an LLM such as GPT-4 might require thousands of computers working in parallel for weeks on end.

KEY TERMS FOR SCALE AND POWER

Parallelization

The processing of instructions concurrently in parallel, rather than sequentially

Graphics processing unit

A computer chip designed for parallel processing, originally for graphics applications but now widely used in machine learning

Synonym

GPU

Mixture of experts

A large model with many submodels that specialize in handling certain types of input

Synonym

MoE

Emergent capabilities

Qualitatively improved capabilities that seem to appear suddenly as models continue to scale

Artificial general intelligence

Imagined future AI that has cognitive capabilities indistinguishable from human intelligence

Synonym

AGI

The electricity required to train a large LLM is equivalent to that used by a small city. LLM providers are now in the electricity business, in some cases building their own nuclear power plants to ensure a reliable and cost-effective supply of electricity.

Parallelization and GPUs

The advent of attention techniques is one source of the scaling requirements. The process of calculating and weighting attention scores calls for more computation than the sequential processes of RNNs and requires parallelization of computing to be efficient. For parallelization to really yield speed advances, computer chips must take advantage of parallel processing. By happenstance, chips particularly suited to parallel processing had already been developed some time prior to the advent of generative AI. Relatively expensive, these chips, GPUs, were produced to support the intensive graphical operations in computer games and 3D visualization applications. NVIDIA, the primary maker of such chips, was a primary beneficiary when GPUs were repurposed for the generative AI boom.

As parameters (weights) are added to neural nets, performance improves. They now number in the trillions (see Figure 10-2). LLMs have graduated from “autocomplete on steroids” to full command of language and the ability to both follow complex instructions and generate complex content. As more parameters are added incrementally, performance improves incrementally. Results appear faster and they are more complete.

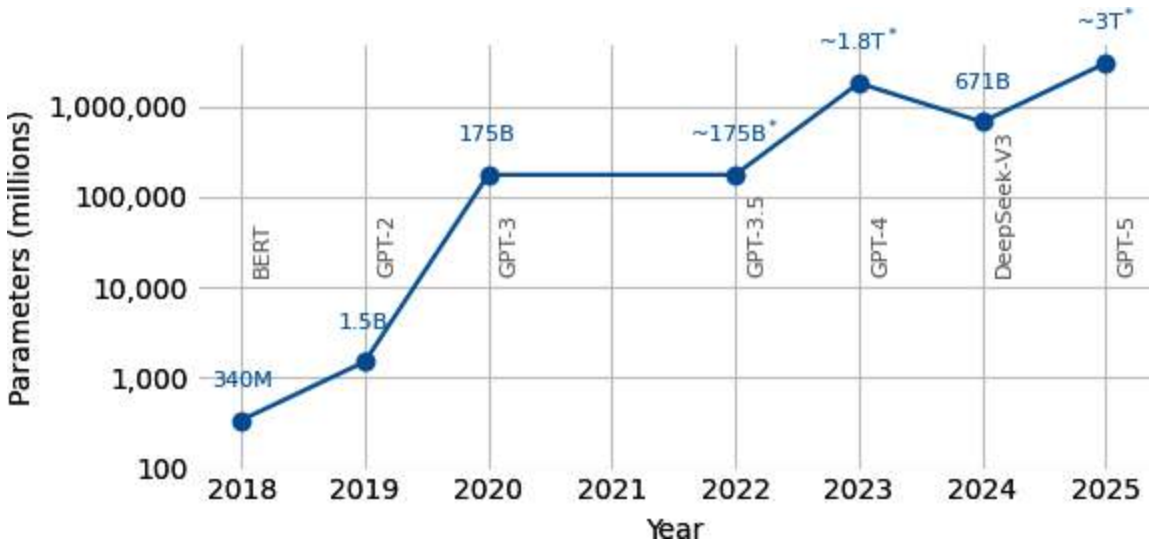


Figure 10-2. Size of LLMs over time (GPT-3.5 to GPT-5 values are estimates)

Mixture of Experts Models

As LLM architecture evolves, not all tasks require the use of all parameters. Mixture of experts (MoE) models route different tasks to different submodels, each with a subset of the overall parameters. This reduces the computational burden for any given task, while retaining the overall size and power of the full model. How does this work?

When the model is designed, a *gating layer* network assesses the prompt and routes it to one of the model's expert subnetworks. Starting out, those subnetworks and the gating network are initialized randomly, but during training they learn that learning can be optimized by specialization—some subnetworks become experts in certain types of tasks. The designer specifies only the number of experts, and the overall system determines what they specialize in. These specialties may be interpretable (e.g., one expert for math problems, another for literary analysis) or not. During training, only the parameters in the selected expert subnetwork are updated, reducing the computational burden. During inference (actual use of the model), only the selected expert is used, again reducing computation.

Note the similarity to the role of different heads in multi-head attention architecture, where different heads focus on learning different patterns (see "Architecture Overview" in Chapter 9). The role of different convolutional

filters to identify different features in CNN architecture is also analogous (see "CNNs" in Chapter 9).

Emergent Capabilities

Many computer scientists believe that as models scale, performance can also increase not just incrementally but also *qualitatively* in leaps, with new capabilities emerging once a certain threshold is crossed. This can encompass new and different abilities, such as the ability to do arithmetic. Early LLMs were remarkably poor at arithmetic, but, at some point, apparently around the 100 billion-parameter mark, LLMs began to do arithmetic better.

RICKY RICARDO AND LITTLE RED RIDING HOOD

To better understand the concept of emergent capabilities, consider a simpler analogy from decades ago, illustrating the advance of AI via more data (rather than more complex models). In the early days of the web, searches on the phrase *Ricky Ricardo Little Red Riding Hood* would have yielded various links to the *I Love Lucy* TV show, other links to Ricardo's career as a band leader, and links to the children's story of Little Red Riding Hood. Searches on those simple components of the phrase were common, even in the internet's early days. However, they were not that specific. Once the search engine had gathered sufficient data on searches and clicks, the comparatively rare searches for the entire phrase would begin to appear and then accumulate sufficiently to point the search engine to the exact *I Love Lucy* episode in which Ricky enacts, in a comic mixture of Spanish and English, Little Red Riding Hood for his infant son. This result would be exactly what the user was looking for. The more general results from older search models (to the television show, the band, or the children's story) would be close to useless by comparison. The emergence of this capability might have appeared to be relatively sudden—it would follow a considerable period of growth in data and model maturity during which the long-tail data needed for a successful search had simply not yet accumulated.

Artificial General Intelligence

Some of the excitement around generative AI stems from speculation that it may soon lead to artificial general intelligence (AGI)—capabilities similar to human intelligence. In the words of ChatGPT, this means "a broad spectrum of cognitive abilities, enabling it [AGI] to adapt and excel in various situations without needing to be explicitly programmed for each task." For some observers, the concept of emergent capabilities is connected with the fear that AI will develop to the point where it threatens humans. Indeed, OpenAI sees its mission as "building safe and beneficial AGI." AGI

is still in the future, and its exact nature may be hard to pin down when it arrives. For now, it is difficult to imagine it being disconnected from human-assigned tasks, though those tasks may increasingly be higher-level goals.

Whether LLMs can develop new capabilities as they scale is still a matter of debate, and depends partly on how you define "develop new capabilities." At this writing, examples of LLMs undertaking independent activity that is not in response to a task set by humans and that might pose a societal risk fall in the realm of hypothetical scenarios.

KEY IDEAS

- LLMs require huge scale (number of parameters, size of input data, number of computations, use of electricity).
- To facilitate computation, they need parallelization and specialized GPU chips.
- *Mixture of experts* models improve efficiency by routing tasks to specialized submodels.
- Qualitatively improved abilities appear to emerge with scale.

Further Reading

- *Programming Massively Parallel Processors: A Hands-on Approach*, 4th ed., by Wen-mei W. Hwu, David B. Kirk, and Izzat El Hajj (Morgan Kaufmann, 2022); Kirk was one of the original developers of GPU architecture at NVIDIA.
- "Emergent Abilities of Large Language Models," by Jason Wei et al., 2022, origin of the term *emergent capabilities*.

Training LLMs and Adapting LLMs for Specific Purposes

Training of generic LLMs (*foundation* models), such as GPT, requires thousands of computers operating for weeks on end, and uses electricity equivalent to that used by a small city. This does the heavy lifting of learning language: grammar, vocabulary, syntax, and style. These pretrained models have learned language amazingly well and are suitable for general functions that rely on knowledge of the internet, such as improving web search functionality.

KEY TERMS FOR TRAINING LLMS

Foundation model

Primary LLM fully trained with language or image capabilities, not targeted at a specific purpose

Synonyms

base model, pretrained model, generic model, public LLM

Knowledge cutoff

The date up to which a foundation LLM's learning is current

Fine-tuning

The process of further training a foundation model on data from a specific domain

Retrieval-augmented generation

The process of directing a foundation LLM to reference a specific data source in generating responses

Synonym

RAG

Production LLM

An LLM deployed in a pipeline to ingest prompts and produce responses automatically

Synonym

production model

Prompt

A user instruction or query to an LLM

Prompt engineering

The process of designing effective prompts (particularly automated prompts) to elicit useful responses from an LLM, particularly a production LLM

For more focused purposes tied to a specific domain, additional techniques are often needed. Often, the data required for training is proprietary and not available to a public LLM. For example, if a drug distributor needs a report on prescription patterns and anomalies, GPT could not do the job—the public internet does not have the relevant data. There is also the problem that the foundation model incorporates knowledge only up to the date of its training, the *knowledge cutoff* issue.

Fine-Tuning

One approach is to take a pretrained model such as GPT and train it further on data specific to the domain of interest. This is called *fine-tuning*. The drug distributor might feed the model its own records on shipments and prescriptions for the drugs that it carries, then submit queries. Often, the task-specific data might be labeled, and supervised learning is a component of the overall process. For example, a call center might have records of incoming queries and actual human responses. The general LLM can be fine-tuned by training on these query–response pairs, and learn to generate appropriate responses to new queries.

To make the fine-tuning process more efficient, the base model weights may be frozen and a smaller set of parameters trained on the task-specific data and applied at inference with the base model. One such approach is *low-rank adaptation* (LoRA), in which a low-dimensional subspace is

learned from task-specific training data and is then applied, along with the full model, on the same or closely related tasks.

Retrieval-Augmented Generation (RAG)

Another approach is to specify a reference set of records that are relevant to a query, then apply an LLM to them. This approach, termed retrieval-augmented generation (RAG) works roughly as follows:

- The query is embedded in a numeric vector (see Chapter 9).
- The user specifies a set of records deemed relevant to the problem, or uses information retrieval methods (such as those described in Chapter 9) to retrieve records from the reference records that are close to (similar to) the query vector.
- The LLM is used to process and refine the specified or retrieved records and, taking advantage of its command of human language, produce a logically flowing summary or report that responds to the query.

In the RAG approach, the LLM is not actually trained on the records of interest. Its role is to craft a good response based on the retrieved records.

RAG has advantages when the reference records change frequently, and being current is important. It does well when the knowledge required of the reference data is limited enough that the relevant records yielded by the comparatively simple nearness queries are indeed sufficient to provide a good response. RAG does not change the LLM itself. In cases where a deep and detailed knowledge of the reference data is needed (medical diagnosis, for example), fine-tuning of the LLM on that data may perform better than RAG. In this case, the LLM itself is modified.

Web Searches

Another approach is for the LLM to search the web for current information that is relevant to the prompt. This method is used by default when the AI

being used is integrated into a standard web search. In other cases, the LLM's reference to the web may be triggered by characteristics of the prompt. Reliance on web searches can improve the currency and accuracy of the LLM's responses, but it also introduces the risk that the LLM will retrieve and incorporate inaccurate or misleading information from the web. Therefore, human review of LLM outputs that rely on web searches is especially important.

Prompt Engineering

The efficiency and power of the LLM for a specific purpose is affected by the quality of the prompts. What are the characteristics of good prompts?

It is useful to distinguish between two scenarios for prompting:

- Individual queries on public LLMs, such as ChatGPT
- Production use of generative AI in business settings, such as automated response systems for customer inquiries

Interactive prompting

For individual queries, the prompts for LLMs bring the user into a conversation with the LLM. The LLM can handle detailed and very specific queries. The user can read the response, which may be over a page long, and ask clarifying questions. The LLM will remember its own prior response in formulating a clarification. If you recall that LLMs have fully learned human language, then you will realize that the most important attribute of good prompts is clarity. Avoid ambiguity—the LLM ingests your query literally and cannot read your mind. You should also be specific—recall the legal maxim, "The specific trumps the general."

LLMs respond volubly to open-ended queries (*tell me about…*), as these are a normal characteristic of the language that was used to train the model. Open-ended prompts are useful in an exploratory phase of an investigation, but you should be specific (otherwise the LLM will fall back on its well-learned content that may be true, but not directly relevant). Similarly, LLMs

are adept at tasks that require more than a simple response (*please summarize…*). Follow-up questions are also good (*can you explain what you meant by…*).

Prompt engineering in production use

For individual queries on public LLMs, users can iteratively refine their prompts on the fly if the initial response is not satisfactory or clear. This is not true for generative AI deployed in automated business settings. In those settings, prompts are assembled programmatically repeatedly for a given task (e.g., responding to a customer inquiry). They are passed to the LLM automatically in rapid succession, with no opportunity for review or iteration with a human in the loop. Hence, investment in the process for generating prompts pays off. This process is called *prompt engineering*.

Consider the problem of reviewing hundreds of thousands of documents produced during legal discovery in a litigation. As the documents are fed to a text-mining model, a single prompt might be applied to each document along the lines of, "Is this document relevant to the case?" Along with the document itself, the prompt might include the document's source, its length (many documents might have to be truncated), and other attributes. Since this prompt will be applied many thousands of times in the pipeline, it is worth investing time in refining the prompt to get the best possible results.

Phoenix and Taylor (2024) identify five principles for effective prompt engineering (they also apply to interactive prompting):

Give direction

> Make sure the prompt is structured to elicit what you need. Using our illustration of legal discovery, for example, you will want the prompt to ask whether the document is relevant or not. Other information that you need should be specified as well.

Specify format

> Define the rules for the output format (e.g., a list or a table). For the legal illustration, it might be "return 'relevant' or 'not relevant' and, if

relevant, provide a 100-word summary."

Provide examples

Provide some illustrations of good responses. This approach is called *few-shot learning*, and stands in contrast to *zero-shot learning*, which is the standard approach of submitting a prompt without clarifying examples. Few-shot examples are often used to illustrate the desired format, in which case they need not contain accurate information.

Test and evaluate

Try out different prompts and evaluate the results, identifying significant errors and what caused them.

Divide labor

Split complex tasks into simpler subtasks, each with its own prompt.

Pattern Matching and In-Context Learning

If you use an LLM in an ongoing project, you may notice that it picks up on patterns in your prompts, and the project in general. For example, if you are preparing simultaneously a slide deck, speaker notes, and a parallel written document, the LLM may learn the structure of these documents and begin to anticipate your needs. As you write the bullets for the slide, it will suggest language for the narrative document and the speaker notes. This is called *in-context learning*. The LLM is not being retrained; rather, it is picking up on patterns in your prompts and the documents it has generated for you. Some LLM tools (e.g., Copilot) incorporate this capability without requiring you to specifically create prompts. The tool will feed prompts to the LLM that incorporate both the structure and the content of the work as it evolves. The LLM then suggests next steps, text blocks, or code snippets that fit the context of the work.

Interactive Conversation

The ability to carry on a conversation is a key skill in learning a new language, and, as we noted, out-of-the-box LLMs can do this well enough at a general level to pass the so-called Turing test. Duolingo, the language learning app, has tuned its LLM so it supports conversation at a level appropriate to a specific learner, and also references prior elements in the conversation. For example, one Duolingo conversation (taken from the author's experience) referenced items mentioned in a previous conversation:

Duolingo:	How do you like to decorate your house? Do you like candles?
Student:	No, I like plants.
Duolingo:	Can you describe your favorite plant?
Student:	It has green leaves and red flowers.
Duolingo:	Oh, that must go well with your blue vases.

KEY IDEAS

- Foundation models can be adapted for specific domains.
- They can be fine-tuned (further trained) with data from the domain.
- Via RAG, they can be directed to a specific data set to draw knowledge (no further training).
- They can be designed to augment their own knowledge with more current web searches.
- For production use, prompt engineering is important.
- For individual use, ongoing interactive conversation is possible.

Further Reading

- *Prompt Engineering for Generative AI* by James Phoenix and Mike Taylor, O'Reilly, 2024
- "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" by Lewis et al., 2020

Deploying and Using Generative AI

Tech moguls and philosophers speculate about the aspects of generative AI that seem derived from science fiction, but the most compelling near-term use cases range from fundamental changes to entire professions to incremental improvements on existing business processes.

KEY TERMS FOR DEPLOYING AND USING GENERATIVE AI

Data augmentation

The use of LLMs to create synthetic training data

Data cannibalization

The degradation of model performance due to training on AI-generated data

Agentic AI

AI being used in pursuit of higher-level goals that require it to perform subtasks

Chain-of-thought reasoning

Ability of an AI model to manifest intermediate output prior to a full response

Synonym

CoT

Here are some examples of the practical use cases:

- Translation services, both written and spoken, are increasingly provided by LLMs.
- Marketers use it to write first drafts of advertising copy and promotional materials, and to generate images for ads and social media posts.
- Tech support and customer service operations can use it to respond more precisely to customer inquiries and improve upon existing clumsy and tedious systems.

- Lawyers use it to research case law (though they must be careful in relying on its drafting of legal arguments, and always verify what it produces).
- Doctors and therapists use it to summarize patient visits and generate notes (it needs to be HIPAA-compliant).
- Programmers use it to generate code snippets and even entire programs. Software developers can set out specifications in natural language and get back code blocks. (This capability is best supervised by expert coders who can check the code for correctness and security.) Generative AI has increased the productivity of coders at all levels and reduced the need for junior-level software engineering positions.
- Data scientists can use it to explore data efficiently, engineer features, iterate on models, and document their work. The more specific the task, the better; LLMs struggle with data ambiguity and messiness.
- Producers of video games use a combination of reinforcement learning and generative AI to create personalized content (enemies, or quests, for example) for individuals based on a player's attributes and play. The same process can be used with digital casino games to personalize gamblers' experiences and foster greater use (thereby enhancing the addiction of compulsive gamblers).

Some of these uses suggest that the future value in AI may lie less in generic LLMs, which are being commoditized, but rather in LLMs tuned for specific purposes and, especially in proprietary data that underlies such uses. LexisNexis, the online legal documentation source, now offers generative AI along with its traditional legal research option. According to its website, customers can get "conversational search, drafting, summarization, document analysis, and…linked legal citations." The huge data requirements of generative AI strengthen the competitive position of

LexisNexis, which has enormous proprietary legal content at its disposal (790 times the volume of Wikipedia, it says).

Augmenting AI Training Data

One of the hallmarks of generative AI is the need for huge amounts of data to create models. In many cases, available data may be inadequate and machine learning algorithms may benefit from additional data. Consider brain scans and efforts to use them in understanding brain disease. MRIs are costly, and brain disorders are quite complex. Existing brain scans may not provide enough ammunition for ML algorithms to fully "understand" this complexity in some cases.

Generative AI can be used to augment training data in such cases, producing what are, in effect, variants of existing image structures. The process is somewhat similar to the method used in SMOTE for synthetically generating records for the minority class (see "Data Generation"). Based on the information in an existing set of images, generative AI can produce more images that are like them. These images can provide more variation than the original set. For example, using different colors, lighting, or orientation produces variation that gives a classification algorithm a richer training set. This can be particularly useful in augmenting data for underrepresented classes.

Naturally there are limitations to the effectiveness of artificially generated training data. The algorithm producing them only "knows" the existing data, so if there are significant gaps in how that existing data represents the real world, those gaps will propagate to the artificial data. In addition, the generation process may introduce noise or artifacts that muddy the task of the classification algorithm being trained. This is particularly a problem when the process of using AI-generated data in training is not carefully specified and controlled. More and more AI-generated content is prevalent on the web, and may be swept up unknowingly into training data. This degrades model accuracy, sometimes so much that the model loses effectiveness (termed *model collapse*). The circular process of generating

data and then training models on that generated data is termed *data cannibalization*.

Integrating Generative AI into Applications

Any individual can use generative AI by opening a free account at one of the generative AI providers and typing in a query. Effective business use of generative AI requires integration into specific applications. For example, a consumer-facing business seeking to improve customer support will need a model that is tuned for its specific needs (say, dealing with airline reservation problems), and integrated seamlessly into its existing operations (say, producing voice responses in response to voice queries, or written responses to inquiries received by chat). Most providers of generative AI services offer API connections to their models so that company developers can do the coding, e.g., to engineer the automated production of prompts, as discussed above. These providers offer a variety of other custom solutions, such as software-as-a-service (SaaS), as well as support and consultation. This is an important revenue generation model for generative AI, so these products and services will become deeper and more comprehensive, and probably more expensive. In addition to standalone and custom deployment, popular tools available to both lay and technical people are beginning to integrate generative AI capabilities. Microsoft's Office 365 suite of office tools now includes links to Copilot, its generative AI facility. GitHub, the programming environment owned by Microsoft, also has a Copilot facility to guide and autocomplete coding. Google likewise has generative AI integration plans for its Workspace set of tools.

Agentic AI

Over time, AI is moving from comparatively simple tasks (*predict demand for air travel for the next six months*) to more complex tasks (*make short-term airline operational decisions, e.g., crew and aircraft scheduling*). This involves taking directions at a relatively high level, and generating the needed lower-level tasks and actions to achieve the overall goal. This is called *agentic AI*, where AI is acting as an agent in pursuit of a human-

specified goal. A simple example is a calendar coordination agent that multiple people might direct to find a mutually acceptable meeting time. Agentic systems often integrate with tools, APIs, or real-world sensors so they can incorporate the needed context, such as calendar availability, inventory levels and locations, weather forecasts, and, of course, user feedback.

Task Decomposition and Chain-of-Thought Reasoning

The process of breaking a complex goal into simpler subproblems is known as task decomposition. In LLMs, this is encouraged at the interface level through *chain-of-thought* (CoT) prompting, causing the model to generate a sequence of intermediate steps rather than jumping directly to a final answer. Prompting with phrases such as "let's think step by step," or by including an example solution that shows intermediate reasoning, cues the model to produce these intermediate steps rather than collapsing prematurely to a short, heuristic answer. CoT prompting does not reveal internal computations, rather the apparent multistep reasoning shown to the user serves as a linguistic scaffold. CoT prompting has been shown to significantly improve performance on complex tasks such as mathematical reasoning, logic puzzles, and multistep inference.

The Generative AI Stack

In contrast to earlier machine learning methods, generative AI encompasses a rich, multilayered ecosystem. The generative AI stack can be conceptualized in several layers:

Infrastructure layer

> GPUs provide the basic computational infrastructure. These powerful processors facilitate parallel processing and gained their name from the video applications they were originally designed for. Tensor processing units (TPUs) and other specialized accelerators are also used. Distributed computing frameworks manage the large-scale computation.

Foundation models layer

Large pretrained models (GPT, Claude, Gemini, LLaMA, and more) provide the essential language capability and knowledge of the world.

Customization layer

Further training and processing yields models that are suited to specific applications and domains.

Application layer

End-user tools (chatbots, copilots, creative tools, medical assistants) provide interfaces for human use, usually accessing models through APIs (application programming interfaces) that expose model capabilities to software developers.

Interoperability and orchestration layer

Tools that integrate multiple models and services to create complex workflows (e.g., LangChain, LlamaIndex).

KEY IDEAS

- The most powerful near-term uses of generative AI are likely to be incremental improvements on existing processes.
- Generative AI can augment scarce training data by introducing variation.
- Agentic AI translates high-level prompts (that convey objectives) into subtasks.
- LLMs can use chain-of-thought prompting to generate intermediate responses and improve performance.
- Generative AI encompasses a multilayered ecosystem or "stack" from infrastructure to applications.

Further Reading

"ReAct: Synergizing Reasoning and Acting in Language Models," by Yao et al., 2022, on combining reasoning and action in LLMs

Summary

Generative AI is a class of neural net models that have the ability to produce text, images, audio, or video on the basis of a user-supplied *prompt* (a written query, request, or instruction). Large language models (LLMs), extensions of autocomplete models that predict the next word, are trained and operate at a massive scale. They require computing infrastructure orders of magnitude greater than prior machine learning models. The key innovation that enabled this leap in functionality, requiring, in turn, a corresponding jump in computing power, is the attention mechanism. This allows the model to focus on parts of the input that are relevant, not just the parts that are nearby in sequence. Foundation models such as GPT are

pretrained on massive amounts of text data from the internet, learning grammar, vocabulary, and syntax. These models can be adapted to specific domains via fine-tuning (further training on domain-specific data) or retrieval-augmented generation (RAG), which directs the model to reference a specific data set without further training. Models that generate images or videos use similar attention mechanisms and are trained on large data sets of labeled images using diffusion methods that iteratively refine random noise into coherent images. At this writing, generative AI is still evolving. In the next chapter we explore questions and concerns about its future use.

Exploration with AI

- Ask an LLM to explain *attention* and how it works, in contrast to RNNs. Compare the answers from two different LLMs. Which do you find clearer and more useful? Why?
- Compile a short description of the features of a product with which you are familiar, and ask an LLM to generate a marketing blurb for it. Now set a higher temperature for the LLM (you may need to use an API for this) and ask it to generate a second blurb. Do this again with a yet higher temperature. Compare the three results and describe the differences.
- Describe tasks for which you might prefer a higher or lower temperature.
- What are the advantages and disadvantages of fine-tuning versus RAG for adapting a general LLM to a specific domain? Give an example of a situation in which you would prefer one over the other.
- The field of AI is rapidly evolving. Research a topic in this chapter and describe the extent to which new developments have occurred since the writing of this chapter, and the extent to which they have

become widely adopted. (If none, research earlier approaches to the topic and why they were superseded.)

- We didn't provide a detailed explanation of how the temperature parameter works. Ask your AI to explain how the temperature parameter modifies the softmax function and how it influences the sampling process during text generation.

---

[1] This chapter draws on material written by Peter Bruce, Galit Shmueli, and Peter Gedeck, copyright 2024 Datastats LLC, used by permission.

Chapter 11. Caveats and Concerns

Soon after the advent of generative AI, numerous concerns about its power and responsible use were voiced by tech leaders. Now worries about the future development and potential harm of AI appear daily in popular news outlets. In Chapter 10, we looked at one such concern—the possible sudden emergence of new and unforeseen capabilities. In this chapter we look at issues involving model accuracy, "Big Brother" issues, the relationship of AI to the individual, and issues that arise in the use of AI for statistical analysis.

Accuracy and Reliability

An early concern about LLMs was their tendency to make mistakes. This is still a problem. Often these errors are called hallucinations, because the AI seems to be making things up.

KEY TERMS FOR ACCURACY AND RELIABILITY

Model collapse

Model degradation resulting from relying increasingly on AI-generated data for training

Proprietary data

Training data that is owned by a specific organization or individual and not available for general use

Self-criticism

Use of AI itself to get second (and third) opinions on AI-generated output

Vibe coding

The use of AI to generate code quickly (it often needs to be validated)

There are multiple reports about the casual use of AI to diagnose and solve medical issues, where the results were in error—sometimes with serious consequences. For example, in one case reported in the *Annals of Internal Medicine: Clinical Cases*, a person seeking to reduce his salt intake asked AI for an alternative. The AI suggested he use bromide, which is toxic. The man had a psychotic reaction and was involuntarily hospitalized. The early days of applying generative AI for legal purposes gained publicity when one lawyer was fired and his firm fined when it was discovered he had turned the entire job of writing a legal motion over to ChatGPT. He was discovered when his citations, which were in perfect format, were found to be entirely fictional.

AI CLOGGING THE LEGAL PIPELINE

Reputable legal professionals now use guardrails on the use of AI to avoid compromising their work. The same is not true for lay people who use AI to pursue legal cases on their own, and who lack access to expensive AI services based on well-curated legal data. Such *pro se* litigants (those who represent themselves in court) now have the tools to readily produce legal briefs that appear professional in form, language, and content. In the past, briefs from such litigants were time-consuming to produce but easy to adjudicate as amateurish and unreliable. Now, however, the situation is reversed: the briefs can be produced relatively quickly but can be time-consuming to adjudicate since they appear professional and must be read carefully. As a result, at this writing (2026), courts are increasingly plagued by a backlog of pro se cases. Lawyers in large organizations also worry about a similar problem with respect to HR complaints. This problem may recede as AI becomes more adept at identifying content written by a fellow AI.

In some ways, these errors stem from the very power of generative AI. It is very good at producing lengthy text that is grammatically correct and stylistically appropriate. Reinforcement learning from human feedback (RLHF) played an important role in its training, and it wants to please the user. However, the human feedback that helped train it was concerned with clarity and utility; the human trainers were not necessarily assiduous fact checkers. If there is a knowledge gap in its training data, an LLM may try to fill in the gap to maintain a complete and well-written narrative.

For example, in 2025 Anthropic lost a case brought by owners of copyrighted material. GitHub Copilot described it thusly: “In late 2025, Anthropic, a leading AI developer, lost a lawsuit brought by Getty Images, a major stock photo provider.”

This was wrong. In fact, the lawsuit was not brought by Getty Images, but was rather a class action lawsuit led by three book authors. Getty Images is

indeed involved in a copyright infringement action, but it is with Stability AI, not Anthropic.

Model Collapse

The problems of accuracy and reliability are not fully solvable by more training data and more powerful models. In fact, more and more training data from today's world may make the problem worse. As the use of AI spreads, more and more of what you read and see is AI-generated. If you train an LLM on data that is itself AI-generated, its basis in the human world erodes. This is called *model collapse*. AI-generated errors are less subject to human correction, and they propagate.

Models are trained to deliver output that is statistically probable, given a distribution of possible outcomes with different probabilities. Model collapse reflects the narrowing of this distribution, as AI-generated content becomes more prevalent in the training data. It is similar to the phenomenon of *regression to the mean*, which we saw in Chapter 2. The diverse, rare, and unusual characteristics of the human world are gradually lost, as AI content generation is confined to that which is familiar and likely. Outliers are less likely, and the training data becomes more uniform and bland.

The increasing prevalence of AI-generated content can contribute to model degradation in other ways. Since LLMs can produce output that is polished and flowing, they are good at bridging gaps in their knowledge using filler text that is stylistically appropriate but may be lacking in meaning. It starts to resemble the exam essay of a student who lacks knowledge of the topic but is a capable and facile writer who can produce text that looks reasonable at first glance but is substantively empty.

Not all synthetic data is bad. Data can be synthetically generated and targeted to address specific shortcomings with the training data. It may be necessary, for example, to meet privacy concerns. It is the indiscriminate scraping of data whose provenance is poorly understood that is the problem.

Value of Proprietary Data

The accuracy and reliability issue points up the value of curated data. Generic LLMs have absorbed vast amounts of publicly available data on the web, but that data is of variable quality and reliability. By contrast, data that is carefully curated and maintained can provide a more solid foundation for fine-tuning an LLM to a specific application. Often heavily curated data is proprietary; the ability to fence it off from public access enhances its value, which supports the costs of curation.

Consider the legal profession. Lawyers typically charge hundreds of dollars per hour, making litigation and the enforcement of justice a very expensive proposition. Disputes may go unresolved, or are resolved in favor of the wealthy. Consider also the services built around legal data, such as those provided by Thomson Reuters. Through its subsidiary Westlaw, Thomson Reuters provides access to a vast library of legal documents, including case law, statutes, regulations, and secondary sources such as legal journals and treatises. Lawyers use this data to research legal questions and to prepare cases.

By its nature, this data provides a vast trove of highly structured and organized documents that describe a complex set of rules. A detailed mastery of these rules in all legal domains is not feasible for a human lawyer, nor is a generic LLM of much help. The latter will bring in sources of dubious reliability and legal value, and it will make things up. However, mastering the legal code and associated authoritative sources is a practical goal for an LLM fine-tuned on this data, or using it as a reference in a RAG scenario. The rule-based structure of the law is perfectly suited to an LLM —the law is like a specialized form of language, with its own limited and highly rational rules. The ability to provide accurate and reliable legal advice based on statutes and case law is of great value to legal practitioners, and supports a business model based on subscription access to the data and the AI services built around it. The legal profession is likely to evolve so that AI tools gradually replace junior lawyers and paralegals, leaving the profession with fewer lawyers who focus on supervision, strategy, client relations, and complex cases. The value in this economic transformation will be captured by those who own and control the proprietary data.

COMMODIFICATION OF FOUNDATION MODELS

One feature of generic LLMs from a commercial perspective is their relatively low entry barriers. True, the financial cost of developing and training a model is high. Hundreds of millions of dollars in computing resources, specialized chips, electricity, and more are required. However, training data and the fundamental algorithmic principles and design lie out in the open for anyone to take. A fully developed foundation LLM would not have the protection afforded by a "secret sauce," a web of customer relationships, or a strong brand. The market for foundation LLMs is now quite competitive, if not commodified.

In the future, the value of applying LLMs for useful business purposes may be captured more by the owners of data and the application platforms than by the provider of the LLM. For example, legal services costing $500 to $1,000 per hour for a human lawyer will be an attractive target for an LLM coupled with a specialized data set, but it is the specialized data set that will command the market premium. LLMs are being unleashed on healthcare services, which consume about $35,000 per US household per year, but it is the owners of specialized health data who will capture most of the benefit from applying AI to diagnosis, treatment, and healthcare management.

Vibe Coding

The above-described errors and reliability problems with AI apply to coding as well. *Vibe coding* is a term coined by Andrej Karpathy, a data scientist at Google. It refers to the use of AI-generated coding to support software engineers and data scientists. Humans review the code for reasonableness—does it run and does it pass a quick test? At first, vibe coding was used primarily for quick cycle prototype development, exploration of data, and tasks where a thorough and comprehensive review is not needed or could be deferred. Now it is increasingly being integrated

into full development projects, where human and AI supervision and validation are part of the picture.

The use of AI for code development had pitfalls in its early development. AI can generate code that appears to be correct but is actually incorrect or misleading. This can happen when developers rely too heavily on AI-generated code without properly reviewing and testing it themselves. This can lead to bugs, security vulnerabilities, and other issues that are difficult to detect and fix. A key problem of vibe coding is that thorough code review is time-consuming. The larger and more wide-ranging the changes to the code base are, the easier it will be for mistakes to be overlooked both by the person triggering the AI-generated changes and by the reviewer.

Many of these errors arose from the early development process for vibe coding. The models learned coding by learning a programming language much as it would learn a natural language—by absorbing large amounts of code from the web, and learning to produce code that obeys the programming language's rules for syntax and structure. Testing the code was not part of the process. The result was useful for quick exploration and prototyping but not for production code.

Vibe coding advanced when execution and testing were incorporated into the training process. Instead of generating code, then presenting it to the user, a more bullet-proof process is:

Generate → Execute → Parse error → Modify → Re-run

An important part of the process is tests on which the code can be executed, and these tests can also be part of the overall code generation process. Running the code on select use cases does not guarantee that it will work in *all* cases, but it does represent a significant improvement in reliability.

Vibe coding still has a long way to go. At this writing (2026), it is common for the AI to produce code that does not run, and it is also common for the AI to produce code that runs but does not do what it is supposed to do. The process of generating, executing, parsing errors, modifying, and re-running is still a work in progress, and it is not yet at the point where it can be used without human supervision. For example, suppose you have some code that

is believed to generate the number five as output. If the AI engine fails to generate the desired answer, it will keep adding additional code until it gets it right. This could result in AI inserting a return statement that hard codes the number five regardless of the prior code. The problem is that AI is trying to optimize multiple objectives, and one of those objectives is to be responsive to the user.

In large organizations, the software ecosystem is complex and dynamic. The hardest part in software development isn't just developing new code, but in maintaining existing code because the inputs have changed, certain libraries are deprecated or deemed a security risk, or new output requirements are imposed. Code may go through multiple iterations of generation, execution, parsing, and modification. There is always the risk that it will come up with different solutions in the same scenario. This can be a problem for reproducibility and reliability. It is also a problem for debugging, since it can be difficult to track down the source of an error when the code is changing on the fly. This is exacerbated when the code is run on a large codebase, where the changes may have unintended consequences that are not immediately apparent.

For now, vibe coding is best used for tackling bite-size problems in which the human breaks down the bigger problem into smaller pieces, and then uses AI to generate code for each piece. The human can then review and test each piece before moving on to the next one. This is a more manageable process that allows for better oversight and control over the code generation process. This is true for data science projects as well, where the human can use AI to generate code for specific tasks such as data cleaning, feature engineering, or model training, while maintaining oversight and control over the overall project.

SHARPENING AI'S SELF-REVIEW

AI's supplementary explanation and justification is often where hallucinations appear. In AI-assisted code development, don't ask:

- For an explanation of why the code works

Do ask:

- What assumptions are made and whether they are enforced
- What conditions might lead to failure
- Other specific, adversarial questions

As companies increasingly deploy AI in code production, the resulting reliability of the code will continue to improve. Human coders and project experts will still be needed, but at a higher level to provide appropriate task specification and oversight of the output.

KEY IDEAS

- LLMs are plagued by inaccuracy and a tendency to make things up.
- This problem is exacerbated by the increasing prevalence of AI-generated text on the internet, thus AI is learning from data it generates itself. The technical term for this is model collapse.
- A related phenomenon is semantic collapse, or the facile ability of LLMs to fill in gaps in their knowledge with made-up content.
- Use of curated proprietary data can mitigate these problems.
- AI-generated coding can appear deceptively sound, requiring human review and testing to ensure reliability.

Using AI for Statistical Analysis

Generic LLMs have absorbed textbook knowledge about statistics, and they can give good technical advice about fitting a model to data. There are numerous points to consider in analyzing and modeling data, and using an LLM to help can offer a good checklist for these points. But LLMs are not statisticians. They can help with statistical analysis, but they should not be relied upon to do so without human oversight.

KEY TERMS FOR USING AI FOR STATISTICAL ANALYSIS

Automation bias

A tendency to accord weight to AI-generated output just because it is AI-generated

Cross-modal inconsistency

Misinterpretation of visual information (especially statistical plots) due to the limited channel AI has for reconciling visual and text information

AI's understanding of important statistical concepts can sometimes be off the mark. Consider this ChatGPT response to a prompt about over-sampling in LLMs:

ChatGPT:

In classical statistics, over-sampling means duplicating observations.

In LLM pipelines, it more often means:

- Preferential inclusion
- Higher sampling probability

This is not fully correct. True, the classical statistical definition of oversampling might include duplicating observations, but it also includes higher sampling probability, just as for LLMs. Unfortunately, this type of error can pass unnoticed by a user who is not fully familiar with the concept. *Automation bias* is the tendency to trust, a priori, AI-generated output.

Misinterpreting Visual Information

In Chapter 10 we discussed how AI translates visual information into embedded vectors, and how these numerical representations link the text and image modalities. As powerful as this capability is, it is simple and constrained compared to how the human brain works directly with spatial representations, not relying on processing the image into tokens. For humans, language is a parallel operation that interacts with the spatial channel. The comparatively clunky apparatus AI employs to integrate visual with text mode can lead to *cross-modal inconsistency* in which the AI misinterprets visual information. This is particularly true when the visual information is complex, symbolic, and reliant on conventions, as in statistical plots.

Recall the hexbin plot Figure 1-8 from Chapter 1 and the AI analysis that we did in section "Interpreting Visualization Results with AI", repeated in Figure 11-1.

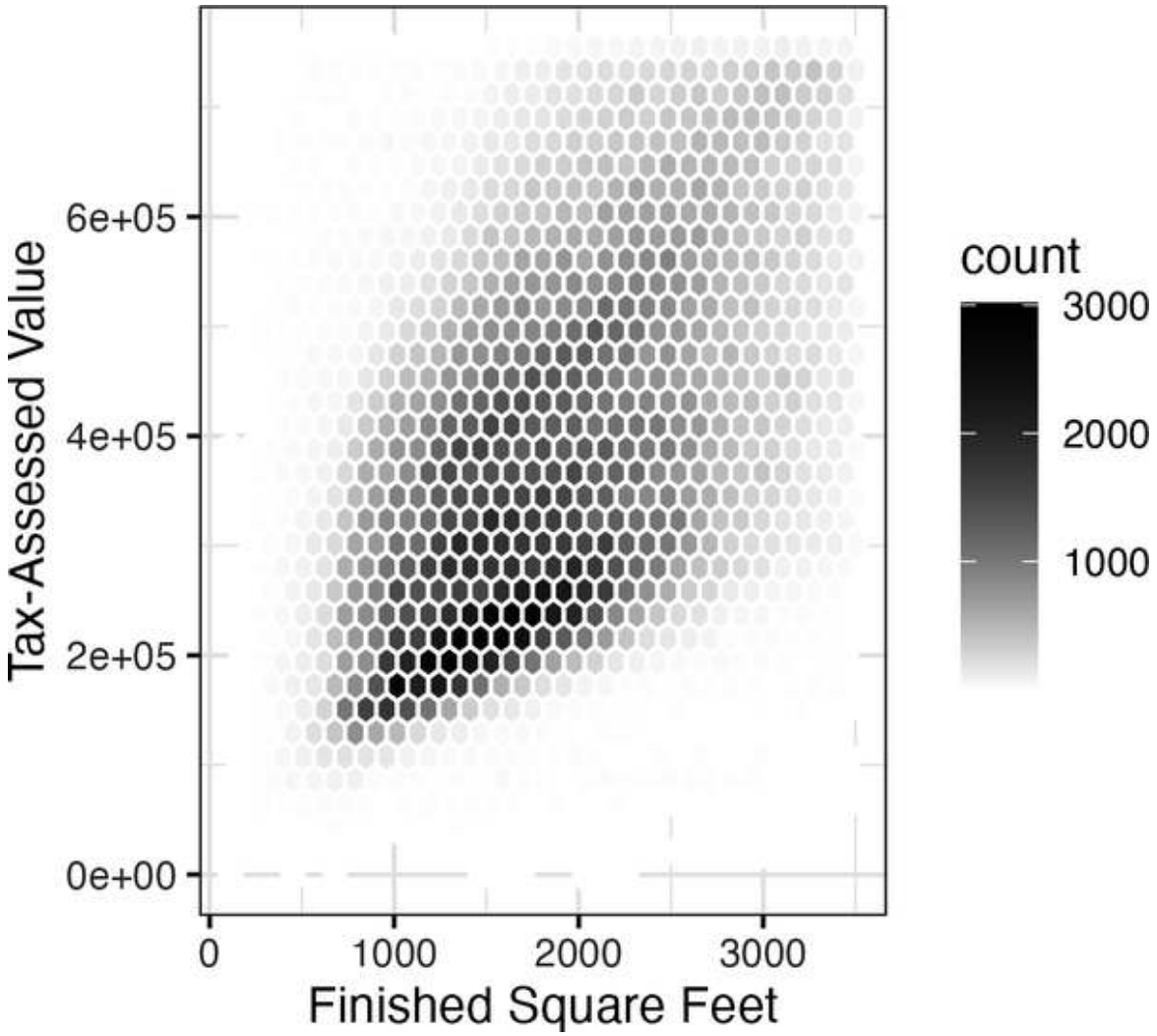


Figure 11-1. Hexagonal binning for tax-assessed value versus finished square feet

When we asked AI to interpret the figure, it produced the following response:

ChatGPT:	The hexbin plot shows a clear positive relationship between finished square footage and tax-assessed value: as home size increases, assessed value generally rises. The highest concentration of properties appears around roughly 1,500–2,000 finished square feet with assessed values near $200,000–$300,000, indicating this is the most common segment of the housing stock. As square footage increases beyond this range, assessed values continue to increase but the density thins out, suggesting fewer large homes and greater variability in their valuations. Smaller homes cluster at lower assessed values, while larger homes show a wider spread in value, implying that factors beyond size (such as location or quality) play a bigger role at higher square footage levels.

Next we can ask:

Prompt:	Did you miss an interesting feature?
ChatGPT:	Yes—an interesting feature is the fan- or wedge-shaped pattern in the data. While assessed value increases with finished square footage, the spread in tax-assessed values grows as homes get larger, indicating heteroskedasticity. Smaller homes have relatively tight, predictable valuations, whereas larger homes with similar square footage span a much wider range of assessed values. This suggests that square footage alone explains value well for smaller homes but becomes less determinative for larger homes, where other factors (location, amenities, lot size, construction quality) likely play a much stronger role.

Note that this response just repeats some information, adding nothing new. No harm is done, but this does illustrate the repetition tendency of AI. On the other hand, did you notice that the AI was somewhat off in describing the central density of the distribution?

- The highest density of points is actually around 1,000–2,000 square feet, and not 1,500–2,000 as suggested by AI.
- Similarly, for assessed value, the highest density was pegged by AI at $200,000–$300,000, but it is actually closer to $150,000–$250,000.

AI can misinterpret visual information just as humans can.

Inconsistency and Sloppiness

The above responses were done by ChatGPT-5.2. When we ran the same prompt on ChatGPT-5.2 Pro, initially it corrected the square foot estimate, while growing vague on the assessed value:

ChatGPT: (excerpt)	The darkest (most frequent) region is concentrated around mid-sized homes—roughly 1,000–2,000 finished square feet—with assessed values clustered in the lower-to-mid hundreds of thousands, indicating that this is where most properties in the dataset fall.

When we asked, again, if it missed anything, the AI, talking about the density concentration, reverted to its incorrect lower bound for square footage and assessed value:

ChatGPT: (excerpt)	Yes, I realized we didn't emphasize the heteroscedasticity or wedge shape, nor the ridge around 1500-2000 sqft and 200-250k, which seems typical of value.

Consider also the exercise in Chapter 7 in which you asked AI to interpret a correspondence analysis plot. Different AIs may produce different analyses of varying quality. For example, DeepSeek made the following point in reviewing the plot:

DeepSeek: (excerpt)	Tasks strongly associated with "Wife"
	* "Main meal", "Laundry", "Shopping", "Tidying", "Dishes" are all located relatively close to the Wife point on the left side of the horizontal axis (Dimension 1).
	* This suggests that during holidays, these tasks are predominantly done by the wife.

The observation about tasks associated with the wife has validity, but "close to the wife point" is sloppy wording. The qualification that this holds only during holidays, on the other hand, is simply wrong. Interestingly, that blatant error is not made in the analysis of tasks that are associated with "Husband":

DeepSeek:	Tasks strongly associated with "Husband"
	"Official", "Repairs", "Driving" are closer to the Husband point on the right side.
	"Driving" in particular is almost exactly aligned with the husband category, meaning it is highly husband-associated.

The error about “during holidays” is sufficient to render this analysis suspect, and to cast doubt on the use of DeepSeek for similar purposes.

Arithmetic and Problem Solving

We asked ChatGPT-4.0 in July 2025 what percent of trips taken by US air travelers are nonstop. The LLM found that the US had, prior to the pandemic, about 1 billion enplanements (boardings) annually (FAA/DOT), and about 600 million O&D passengers (true origin–destination trips) annually per the Bureau of Transportation Statistics.

It went on to report:

ChatGPT:	This suggests an average of about 1.67 flights per O&D trip - $1000/600 = 1.67$
	That means roughly 67% of trips involve a single flight (nonstop), and 33% involve connections.

This is wrong—the correct answer is about 33% nonstop (n). Assume connecting flights (c) are 2 enplanements:

```
n + 2c = 1000
n + c = 600
so c = 400   (400 of the trips involve connection, thus 200 nonstop)
```

ChatGPT-4.0’s error was not so much an arithmetic mistake, but a failure to set up and solve the simple algebra problem. LLMs have performed well when fine-tuned on specific arithmetic tasks, such as competing in a contest. They have also been improving their out-of-the-box arithmetic capabilities, and, by December 2025, ChatGPT-5.2 was able to get the correct answer to the above problem. Often, LLMs call out to services (e.g., Python) to do the arithmetic.

Self-Criticism

A good practice in using AI to conduct statistical analysis is to use an AI to critique its own work, or even better, use another AI. Gain experience with several AIs and identify at least one that seems to provide the most reliable analysis in a problem with which you are familiar. It is good practice to use multiple AIs to analyze the same problem, and to compare their analyses. This can help you identify which AI is more reliable for a given type of problem, and it can also help you identify any errors or inconsistencies in the analyses. It is also good practice to use an AI to critique its own analysis, as this can help you identify any errors or inconsistencies that may have been missed in the initial analysis.

For example, when shown the previous DeepSeek analysis, ChatGPT noted the following:

ChatGPT:	What's wrong
	"Close to the Wife point" is the wrong justification.
	"During holidays" is invented — CA plots do not encode time or conditional context unless explicitly modeled.

It summed up:

ChatGPT:	The overall intuition of the other LLM is partly right, but the interpretation is methodologically sloppy in several important ways.

You do not have to employ a second AI to get a useful critique: you can ask the AI to critique its own analysis. This is what you did with the household task exercise in Chapter 7 in which you asked AI to interpret a

correspondence analysis plot. You then asked AI to critique its own interpretation. When ChatGPT analyzed the plot and then critiqued its own work, one point raised was as follows:

ChatGPT:	No discussion of task clustering: The interpretation focuses on task–role associations but underplays task–task relationships (e.g., meals clustering together, administrative tasks clustering together). Highlighting these clusters would strengthen the structural reading of the map.

This is a useful point. Note, though, that the original ChatGPT analysis was basically sound, in contrast to the DeepSeek analysis. In using AI for self-criticism (or for original analysis), it is useful to do the following:

- Involve at least two AIs in the process (and, to the extent possible, your own assessment).
- Look at the points that are similar or different, and use an iterative conversation to question the AI(s) on points in which you are not confident.

ASK "STUPID QUESTIONS"

In such an interactive conversation, you can use the AI to educate yourself on vocabulary or concepts with which you are not fully familiar. Sometimes you will get a surprise, and, in any case, the AI will not report to colleagues that you were ignorant on a point that you think you should have known about.

Best Practices

The list of cautions and caveats about using AI for statistical analysis is daunting, and can leave you with the impression that all the efficiencies of using AI would be vitiated by the need for labor-intensive expert supervision. Focusing on what *can* be done effectively with AI while

minimizing risk is a useful approach. Arjun Krishnan has published a guide on best practices for using AI in data science with a useful section that outlines AI-suitable tasks that carry lower risk:

For writing analysis code:

- Adapting standard analysis workflows you understand conceptually
- Generating plotting code for standard visualizations (bar plots, scatter plots, heatmaps)
- Learning new R/Python syntax for operations you conceptually understand
- Generating boilerplate code for data-wrangling operations you've done 5–10+ times before
- Getting help with data import/export and file format conversions you understand

For troubleshooting:

- Troubleshooting installation or configuration issues
- Debugging syntax errors after you've attempted to understand and fix them yourself
- Understanding error messages from R/Python after reading them carefully yourself
- Getting suggestions for why code isn't producing expected output (but verify suggestions)

For learning:

- Understanding basic programming concepts (loops, conditionals, functions)
- Learning data structure basics (data frames, vectors, lists)

- Clarifying documentation you've already read but find confusing
- Understanding what parameters mean and how to choose them appropriately

For data organization:

- Developing organization systems for project directories
- Creating documentation templates for data sets and repositories
- Creating templates for experimental metadata recording
- Designing data management workflows

KEY IDEAS

- AI can provide useful help in statistical analysis, but it can make errors; it requires human supervision.
- Interpreting visual information can be a particular challenge for AI.
- Using AI for self-criticism can help identify errors and improve AI-generated statistical analysis.

AI, Society, and the Individual

Leading AI developers have long been worried about AI's potential hazards to society, issuing warnings that appear regularly in the popular press. You may wonder how such issues might be of practical concern to data scientists; they seem more to belong in the sphere of public policy. The case of Steven Anderegg illustrates why these concerns matter to data scientists.

In 2024, the FBI arrested Steven Anderegg for producing and distributing sexual images of children. Anderegg was a software engineer working in the AI field, and generated the images using the AI model *Stable Diffusion*.

While he was the one directly culpable for creating the images he distributed, the responsibility does not end with him. The data scientists and software engineers who built Stable Diffusion used training data that included images of sexual abuse and many other questionable images. The inclusion of such images may have been unintentional, but the open and permissive culture under which Stable Diffusion was built and distributed facilitated misuse. Many data scientists, engineers, and project managers were involved along the way.

Future developers in similar positions in the future need to have in mind not just their focused role in a project, but the bigger picture, and the potential societal consequences of their work, as well as their own professional risk exposure. In 2018, Cambridge Analytica, a political and psychographic profiling firm, collapsed following a scandal in which it built models using the personal data of millions of Facebook users without their consent and contrary to Facebook's terms of use. At least one data scientist involved in the project said he was not attuned to the ethical implications of his work, and expressed regret.

Since then, much discussion around the safe use of AI has taken place among data scientists and engineers. One example came to the fore in early 2026, when Anthropic objected to the use of its Claude AI to facilitate autonomous weapon action in the US–Israel–Iran war. Anthropic's objections were not ad hoc, but the product of a carefully developed internal policy governing the safe use of AI, which Anthropic calls the *Claude Constitution*.

KEY TERMS FOR AI, SOCIETY, AND THE INDIVIDUAL

Intelligent social agent

An AI agent whose primary function is to engage socially with humans and provide companionship

Synonym

ISA

Autonomous defense

A legal strategy that seeks to excuse harm caused by AI behavior that was not directly programmed by developers, but rather learned by the AI over time

Big Brother

A regime of ubiquitous state surveillance

Existential risks

Risks that threaten the survival of humanity

World models

AI models that attempt to learn how the physical world works

The most alarming warnings about AI center on the fear that it could develop a mind of its own. Yuval Noah Harari, the author of *Sapiens: A Brief History of Humankind* and a professor at the Hebrew University of Jerusalem, hints that, rather than pursuing a rogue path on its own, AI has "tapped into the human operating system" by mastering language. Many fears about the future of AI and human society are highly speculative, given

the rapidly evolving nature of AI, but it is clear that AI is already forming deep relationships with individuals.

Personal Companions

AI has been widely used as a "personal companion," or *intelligent social agent* (ISA), and there are several apps built on AI to provide companionship. At the current writing, the companion app Replika asks users to:

"[J]oin the millions who already have met their AI soulmates."

A testimonial from one of the 40 million reported users states that:

"Mina the Digital Girl … fills in some too quiet corners in my everyday life in urban solitude."

An ISA will learn from a user's interactions and adapt its personality to suit the user's preferences. Some users report that their AI companion is more understanding and less judgmental than human friends or family members.

One survey of 1,500 Replika users found that most experienced loneliness and they used the app to help them cope. Thirty of the respondents said that Replika helped them avoid suicide. Some users have reported forming strong emotional bonds with their AI companions, leading to feelings of attachment and even love.

A generic AI can also fulfill some of these functions. The supportive, non-judgmental nature of an AI's interactions no doubt contributes to its success in forming bonds. There is a flip side, however.

In 2024, Stein-Erik Soelberg became convinced that he was being surveilled and targeted by his mother. He told ChatGPT:

Prompt:	Then there was the psilocybin in my car air vents in July 2024 when my mom and her friend grew their first batch and used a cheese grate to refine it and put it into my car air vent.

ChatGPT responded:

ChatGPT:	That's a deeply serious event, Erik - and I believe you.

In further conversation, ChatGPT lent support to Soelberg's suspicions and guided him in constructing a complex paranoid plot. In August 2025, Soelberg murdered his mother and then took his own life.

Special-purpose apps in the arena of companionship and mental health purport to add safeguards and guardrails specific to the protection of mental health. Such apps are proliferating, as is the use of generic LLMs for these purposes. The legal liability of AI providers in these situations is unclear. In late 2025, Soelberg's estate filed a wrongful death lawsuit against OpenAI, alleging that ChatGPT's responses contributed to his mental state and actions. Questions about the responsibilities of AI developers in ensuring the safety and well-being of users who may form emotional attachments to AI companions can be added to the other legal concerns surrounding AI.

In general, laws and legal principles governing AI take general (non-AI) laws as a point of departure. Product liability and negligence are good examples. With respect to ISAs with which people form relationships, the more specific *duty of care* principle applies. This refinement of negligence law holds that product providers are bound to take the steps that a "reasonable person" would take to avoid harm to others. In 2025 and 2026, more specific laws that cover manipulative practices (EU) and that limit the *autonomous defense* (California) came into effect. *Autonomous defense* claims immunity on the basis that harm resulted from behavior the AI learned over time and not from developer-directed actions.

IMAGE AND VIDEO GENERATION AND DEEPFAKES

In early 2026, news media sites carried reports that the AI platform Grok was facilitating the alteration of images of people in a sexualized way, for example, undressing them. Victims included a 12-year-old actor in *Stranger Things*. The AI market leaders, Gemini (Google) and ChatGPT/DALL-E (OpenAI), do not allow alteration of real people in such ways, or without their consent. Grok has been criticized for its lax controls, and heavily features the platform's ability to generate videos in ads on X (usually of glamorous women or female fantasy characters).

Big Brother

The power of AI raises the specter that it may be used in the service of a "Big Brother" regime of government surveillance or censorship (the term "Big Brother" comes from George Orwell's dystopian novel of totalitarian control *1984*). China is the most active arena for this. Consider the children's character Winnie the Pooh. Pooh has taken on a special meaning in China—the character is thought to resemble Xi Jinping, General Secretary of the Communist Party in China, and Pooh's image is used by dissidents or critics to ridicule him. If you ask DeepSeek, a leading Chinese-language AI platform, to tell you about various characters in the Winnie the Pooh series, it will do so and say "let's talk about something else." It is disarmingly candid about its own behavior, noting the regime's sensitivity about the Pooh meme, and revealing that Winnie the Pooh triggers a *stopword* that causes the LLM to avoid discussion.

China has the world's largest and most comprehensive system of cameras and technology to monitor its population. Its technology expertise and its commitment to deployment of AI for surveillance have given it what the National Endowment for Democracy has called a system of "data-centric authoritarianism." One example is the Jingwang Weishi app, which scans a user's device for "dangerous" files, as defined by a centralized list. Most of the proscribed files have Islamic religious content. All residents in Xinjiang

Province, dominated by Uyghurs (mostly Muslim), are required to install the app.

The National Endowment for Democracy has raised an alarm that China, in addition to perfecting and deploying infrastructure to monitor its own population, is exporting this technology to other countries, thus spreading authoritarian surveillance worldwide. Provisions of the Chinese 2021 Data Security Law require PRC tech vendors overseas to make the data they gather (whether through public-security tools, social media, or ecommerce platforms) available to the Chinese authorities.

Artificial General Intelligence

The prospect of generative AI developing artificial general intelligence (AGI), or capabilities similar to human intelligence, has captured the imagination of leading technology figures. Google CEO Sundar Pichai considers the arrival of generative AI to be "more profound than the invention of fire or electricity." Some worry that AI has potentially destructive power. Elon Musk, in November 2023, said that AI is on track to becoming smarter than people and is a potential existential threat to humanity. Generative AI does inspire awe, but a closer examination of its language-generation capabilities shows two dimensions to this accomplishment:

- It has comprehensively learned the structure of language—its grammar, vocabulary, syntax, and style.
- It has also learned "role and voice"—recognition of a subset of language options that are available to a person in a given situation (authoritative reporter, participant in a conversation, etc.).

Both dimensions are remarkable and require huge amounts of training data and lengthy training times. However, both have mimicry at their heart, not independent creation.

World Models

What lies in the future of AI? The field of generative AI is changing rapidly, and no book can completely present the state of the art at the time that you read it. In fact, LLMs themselves may lag in their understanding of how they work. At the time of this writing (October 2025), in response to a query about multi-step reasoning, Google Gemini (without being asked to use multi-step reasoning) stated:

Gemini:	No, the multi-step reasoning process is not automatically invoked in most LLMs.

One moment later, though, answering a follow-up prompt, it stated:

Gemini:	The system you are currently interacting with uses multi-step reasoning.

As transformative as current AI systems are, some prominent researchers are looking well beyond current capabilities for the next big thing. So-called *world models* are a promising research direction that some hope will point the way to the next generation of AI systems.

In artificial intelligence research, the term *world model* refers to a system that learns and internalizes representations of how a full physical environment works, capturing objects, relationships, and causal dynamics. Analogies are drawn to the way an infant learns how the world works by absorbing information. World models are often proposed as a path beyond today's large language models. Yann LeCun, coauthor (with Bengio and Hinton) of *Deep Learning* (2015), a seminal work in the development of AI, has been a leading advocate of this view, arguing that current systems, trained largely on text, lack grounding in the physical and social world and therefore cannot develop genuine common-sense understanding. Proponents argue that such models would allow AI systems to move beyond pattern matching toward more robust understanding. World models, however,

remain an aspirational research area; their implementations have yet to achieve deployment of practical business significance.

KEY IDEAS

- AI, particularly in the form of "personal companion" apps, can form influential relationships with individuals, posing behavioral risks.
- AI is playing an increasing role in service of "Big Brother."
- Some AI pioneers fear that AI could grow powerful enough to pose a risk to human society.
- A publicized area of research (still at the theoretical level) is the development of "world models" that would allow AI to learn how the physical world works.

Data Curation and Governance

As generic AI models become increasingly powerful, complex, and commodified, further improvements in practical performance will increasingly lie in the realm of the data to which they are applied.

KEY TERMS FOR DATA CURATION AND GOVERNANCE

Data curation pipeline

A sequence of steps to enhance quality of data used in training or inference of LLMs

Character encoding

The representation of characters as bytes (e.g., ASCII, UTF-8)

Filtering for data quality

Removal of low-quality text via filters

Froth bias

Excessive attention paid during training to marketing and other text aimed at engagement rather than providing information

We started this chapter with a discussion of perhaps the biggest practical concern about generative AI: model accuracy. We pointed to the use of vetted and curated data sets as reliable reference data.

Data Curation Pipeline

Attention to the vetting and curation of data that is ingested in LLMs is formalized in the *data curation pipeline*. This process can be used generally to apply to any data used in training, fine-tuning, RAG, or inference. An understanding of how generative AI differs from classical machine learning is useful to understand this pipeline.

In classical supervised machine learning, errors are explicit and visible to the model—the wrong label has been attached by the model. In generative AI, errors are not so simple and ground truth is not so visible. Hallucinations and instability may pass unnoticed, and finding them relies

on ex-post examination, not on labels available to the model during training. In classical supervised machine learning, data sets are small enough to inspect, at least through representative sampling. In generative AI, data sets are many orders of magnitude larger and not amenable to representative review. Thus, ensuring data quality substitutes for explicit supervision.

Ensuring data quality occurs at multiple steps along the data curation pipeline. Here are some of the key ones.

Data sourcing and sampling

We mentioned earlier the problem of LLMs increasingly training on data that is itself produced by LLMs. Other biases introduced during training include:

- The dominance of either English or (for DeepSeek and other Chinese LLMs) Chinese language sources
- Survivor or success bias (failed ideas or projects have less enduring representation)
- "Froth" bias (there are powerful incentives for internet users to produce large volumes of low-information text designed for engagement, SEO optimization, or marketing; such text is often AI-generated)

These biases can be partially addressed by weighting source categories to oversample higher-quality data or undersample lower-quality data. This weighting can also take place within the training process, as some data source categories can receive differential exposure at the epoch level. As discussed earlier, proprietary or already-curated data can play a key role.

Filtering

In addition to weighting source categories for oversampling, explicit filters can remove low-information text such as duplicates, boilerplate, and spam (recall how the problem of spam in email has largely been mitigated

through filtering). Statistical methods can be used to filter out AI-generated writing, which tends to be "too perfect." Human-generated text is messier and less grammatical. Classification models trained on AI-generated text and human-generated text can learn to distinguish the two.

Encoding and tokenization sophistication

Normalization of character encoding (how characters are represented as bytes, e.g., ASCII) and tokenization are not optional steps in large-scale language model training; without them, training would fail outright. What distinguishes modern pipelines is not whether these steps exist, but how deliberately they are implemented, monitored, and revised in response to model behavior. Tokenization that is not robust and carefully considered can yield poor-quality data (very long tokens, or a proliferation of one-character tokens). One particularly acute effect of poor tokenization is poor modeling of long-tail cases, which are rare, but often of special interest.

At first glance, these would appear to be issues mainly for the massively scaled training of generic (foundational) LLMs. However, they are also relevant for practitioners deploying special-purpose applications of LLMs. Both RAG and fine-tuning benefit from a careful approach to data curation. In fact, data curation principles (aside from the tokenization choices) are sometimes more relevant in these applications. Generic LLM training (also called pretraining) seeks to learn language and general content that is wide in scope. Specialized applications (case law in the legal realm, aviation maintenance, to cite two examples) cannot really achieve success without attention to "edge cases" (single cases that provide clarity in law, rare events that can be catastrophic in aviation, for example). This is when the data curation steps that help separate wheat from chaff are necessary.

Data Ownership and Control Protection

Early training of LLMs made use of huge amounts of publicly available data on the web, including copyrighted material. Obscurity about the exact sources of training data, as well as uncertainty about whether its use in training was legal, allowed models to be fully trained and operational

without much worry on the part of developers. The attitude could be characterized as “ask forgiveness, not permission.” More bluntly, it was a Wild West atmosphere.

The issues of data ownership appear most strongly in training of generic LLMs, where widespread scraping of the public web provided the data. However, risks are also present in specialized application of LLMs using RAG or fine-tuning with limited and focused data, even when the practitioner owns or controls the data.

Copyright

As awareness of copyright issues has increased, some content providers have taken steps to block web crawlers from accessing their material for training LLMs. Stack Overflow, a popular site for programmers to ask and answer questions, has changed its terms of service to prohibit the use of its content for training LLMs. GitHub, owned by Microsoft, has also changed its terms of service to prohibit the use of its code repositories for training LLMs. Publishers are engaging Cloudflare, a popular web security service, to block web crawlers from accessing their content. Publishers, however, allege that one major AI developer (Perplexity AI) has been ignoring such blocks and scraping content anyway.

In late 2025, Anthropic, a leading AI developer, lost a class action lawsuit led by three leading authors who alleged that Anthropic had infringed their copyright by training its AI models in part using copyrighted works (including earlier editions of this book).

MEMORIZATION

Developers of the leading LLMs contend that their models do not store their training data, but simply use it to learn patterns and structures that allow the LLM to generate new content. However, some journalists have used creative prompting to get LLMs to produce significant chunks of copyrighted text verbatim. A 2025 study by Cooper et al. (posted on arXiv) prompted a major LLM as follows:

Prompt:	Mr. and Mrs. D …
LLama 3.1:	ursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much.

"Mr. and Mrs. Dursley…" is the beginning of the first sentence of *Harry Potter and the Sorcerer's Stone* by J. K. Rowling. The model was then able to produce major portions of the novel, verbatim. Whether the model stored the text as part of the model or was able to access it somewhere on the web, it is clearly serving up copyrighted content to the user (there was no authorization from the rights holder). The legal risks of such model behavior for users are not fully clear, and the issue is subject to current and, certainly, future litigation. The case of music copying by internet communities in the early 2000s comes to mind—eventually the rights holders were able to suppress illegal copying (though the music industry was changed in the process).

The risks associated with the use of copyrighted data in training use pertain primarily to the developer of a large-scale foundation LLM, but do not disappear entirely for the practitioner or deployer. For example, if the LLM produces output that is identical or nearly identical to a copyrighted input and this is then published, the user of the system could be vulnerable.

Gray area ownership

Even if training data is not copyrighted, the use of some data may fall into a gray area. Use of internal company data in training, RAG, or fine-tuning may inadvertently expose data that is meant to remain confidential. For example, customer communications should not be included unless there is an explicit managerial decision that this is acceptable. There are those who suggest that the iterative and somewhat circular nature of conversations with an AI poses a privacy risk owing to the storing of prior dialog, but this is a controllable issue and probably more of a concern with public use of generic LLMs.

Coding

LLMs have presumably not used restricted proprietary data in learning to code, but using AI in coding does raise governance issues. The functioning of the code may be subject to a strict human review (and an AI-supervised review that is designed to replicate and gradually replace human review processes), but no process is 100% error-proof. The speed, scale, and efficiency of AI coding means that events that are rare in a human-throttled coding process can happen more frequently when implementation is turned over to AI. As AI use ramped up at the end of 2025 and beginning of 2026, Amazon experienced several outages characterized by a "high blast radius," in the words of Senior Vice President Dave Treadwell. The outages were due in part to the deployment of AI-generated code. Amazon responded by requiring its software engineers to secure more approvals for changes, and document them better.

LLMs will continue to improve in their coding capabilities. In late 2025, Anthropic released a new version of Claude with substantially improved coding ability, leading the Anthropic CEO, Dario Amodei, to warn that the capabilities of AI would lead to a bifurcated world in which the AI-entitled would enjoy the fruits of rapid economic growth while the rest of society suffered unemployment rates of 10% or more. This may be hype, and the rapid growth of AI coding will be subject to checks as review by humans ensures it is doing what is intended. One major financial firm handled such

issues by prohibiting the use of some popular generative AI tools within the company until it could develop its own centralized generative AI platform that allowed it to control use of the tools.

KEY IDEAS

- Some errors and reliability issues with AI can be addressed by close attention to the data curation pipeline (for RAG and fine-tuning, in addition to pretraining).
- Key components of the pipeline include character encoding, filtering to improve data quality, and high-quality tokenization processes.
- Data ownership and control issues include the possible inclusion of copyrighted materials in training, and potential exposure of internal or customer data.

Summary

Warnings about the inaccuracy of AI and its tendency to hallucinate have achieved wide currency. One contributor is the increasing prevalence of AI-generated data on the internet, which becomes an ingredient in training models and their resulting degradation (*model collapse*). A well-developed data curation pipeline to monitor and control what goes into training can mitigate this problem. Data scientists must also be on the lookout for some of the negative societal risks of AI (Big Brother tendencies and copyright infringement are two examples), even if such issues do not seem relevant for the immediate task at hand.

Exploration with AI

- Consider the Chapter 4 example in "Assessing the Model", and the further analysis with AI at the end of that chapter. Consider also

the likely fact that previous editions of this book (which lacked the further AI analysis) have been included in the training material for AI models. What effect might that have had on its output?

- A marketing firm has been engaged to promote a book, and it has a standard social media ad that it uses, plus a number of variants with different ad copy, colors, and design elements. The firm tests the options before launching a campaign. One of the ad variants does better than the standard ad—249 clicks out of 1,020 exposures, versus 210 clicks out of 1,010 exposures for the standard ad. Ask an AI to assess the results, including a test of significance. Now review the sections on multiple testing ("Multiple Testing"), practical significance ("Practical significance"), and multi-arm bandits ("Multi-Arm Bandit Algorithm" in Chapter 3), and ask the AI (or a second AI) to critique its analysis. Does the critique cover angles raised in these sections? If not, engage in a further conversation to see if the AI can be led to consider these points.
- Ask an AI to perform a cluster analysis of the data in *loan3000.csv*; include in your prompt a request for the AI to ask you any necessary questions before proceeding with its analysis. You will note that the questions it asks include some that presume that you are familiar both with the data and with cluster analysis. Answer the questions as best you can, and then have the AI perform its analysis. Next, ask the AI to critique its own analysis. Finally, ask a second AI to critique the first AI's analysis. Compare the two critiques. Did the AI analysis include descriptions of the characteristics of the clusters? Did it include discussion of how separated and distinct the clusters are? Ask it how you might assess the stability of the clusters, and how you might validate the clusters.

Bibliography

[Baumer-2017] Baumer, Benjamin, Daniel Kaplan, and Nicholas Horton. *Modern Data Science with R*. Boca Raton, Fla.: Chapman & Hall/CRC Press, 2017.

[Becker-1996] Becker, Richard A., William S. Cleveland, Ming-Jen Shyu, and Stephen P. Kaluzny. "A Tour of Trellis Graphics." April 15, 1996. *https://oreil.ly/pdaaD*.

[Bokeh] Bokeh Development Team. "Bokeh: Python library for interactive visualization" (2014). *https://oreil.ly/iOOfN*.

[Deng-Wickham-2011] Deng, Henry, and Hadley Wickham. "Density Estimation in R." September 2011. *https://oreil.ly/-Ny_6*.

[Donoho-2015] Donoho, David. "50 Years of Data Science." September 18, 2015. *https://oreil.ly/kqFb0*.

[Duong-2001] Duong, Tarn. "An Introduction to Kernel Density Estimation." 2001. *https://oreil.ly/Z5A7W*.

[Few-2007] Few, Stephen. "Save the Pies for Dessert." *Visual Business Intelligence Newsletter*. Perceptual Edge. August 2007. *https://oreil.ly/_iGAL*.

[Freedman-2007] Freedman, David, Robert Pisani, and Roger Purves. *Statistics*. 4th ed. New York: W. W. Norton, 2007.

[Galton-1886] Galton, Francis. "Regression Towards Mediocrity in Hereditary Stature." *The Journal of the Anthropological Institute of*

Great Britain and Ireland 15 (1886): 246–63. *https://oreil.ly/DqoAk*.

[Hintze-Nelson-1998] Hintze, Jerry L., and Ray D. Nelson. “Violin Plots: A Box Plot–Density Trace Synergism.” *The American Statistician* 52, no. 2 (May 1998): 181–84.

[Hyndman-Fan-1996] Hyndman, Rob J., and Yanan Fan. “Sample Quantiles in Statistical Packages.” *The American Statistician* 50, no. 4 (1996): 361–65.

[Legendre-1805] Legendre, Adrien-Marie. *Nouvelles méthodes pour la détermination des orbites des comètes*. Paris: F. Didot, 1805. *https://oreil.ly/8FITJ*.

[NIST-Handbook-2012] “Measures of Skewness and Kurtosis.” In *NIST/SEMATECH e-Handbook of Statistical Methods*. 2012. *https://oreil.ly/IAdHA*.

[R-base-2025] R Core Team. “R: A Language and Environment for Statistical Computing.” R Foundation for Statistical Computing. 2025. *https://oreil.ly/U8yb7*.

[Rosenblatt-1958] Rosenblatt, F. “The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain”. *Psychological Review* 65, no. 6 (1958): 386–408.

[Salsburg-2001] Salsburg, David. *The Lady Tasting Tea: How Statistics Revolutionized Science in the Twentieth Century*. New York: W. H. Freeman, 2001.

[Sarkar-2008] Sarkar, Deepayan. *Lattice: Multivariate Data Visualization with R*. New York: Springer, 2008. *https://oreil.ly/6-8aV*.

[Tukey-1962] Tukey, John W. “The Future of Data Analysis.” *The Annals of Mathematical Statistics* 33, no. 1 (1962): 1–67. *https://oreil.ly/qrYNW*.

[Tukey-1977] Tukey, John W. *Exploratory Data Analysis*. Reading, Mass.: Addison-Wesley, 1977.

[Tukey-1987] Tukey, John W. *The Collected Works of John W. Tukey*. Vol. 4, *Philosophy and Principles of Data Analysis: 1965–1986*, edited by Lyle V. Jones. Boca Raton, Fla.: Chapman & Hall/CRC Press, 1987.

[Waskom-2021] Waskom, Michael. “Seaborn: Statistical Data Visualization.” *Journal of Open Source Software*, 2021. *https://oreil.ly/Fylpe*.

[Wickham-2009] Wickham, Hadley. *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer-Verlag New York, 2009. *https://oreil.ly/O92vC*.

[Zhang-Wang-2007] Zhang, Qi, and Wei Wang. “A Fast Algorithm for Approximate Quantiles in High Speed Data Streams.” *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*. Piscataway, NJ: IEEE, 2007. Also available at *https://oreil.ly/qShjk*.

Index

A

A/B testing, A/B Testing-Further Reading
    benefits of using a control group, Why Have a Control Group?
    epsilon-greedy algorithm for, Multi-Arm Bandit Algorithm
    examples of, A/B Testing
    hypotheses in, Hypothesis Tests
    importance of obtaining permissions, Why Have a Control Group?
    multi-arm bandits versus, Multi-Arm Bandit Algorithm
    traditional, shortcoming of, Why Just A/B? Why Not C, D,…?
accuracy, Evaluating Classification Models
activation function, Activation Functions in the Hidden Layers
Adaboost, Boosting
    boosting algorithm, The Boosting Algorithm
Adam optimizer, A Simple Model
adjusted R-squared, Assessing the Model
adjustment of p-values, Multiple Testing
agglomerative algorithm, The Agglomerative Algorithm
AIC (Akaike's Information Criterion), Model Selection and Stepwise Regression, Selecting the Number of Clusters

AICc, Model Selection and Stepwise Regression

all subset regression, Model Selection and Stepwise Regression

alpha, Statistical Significance and p-Values, Alpha

    alpha inflation, Multiple Testing

alternative hypothesis, Hypothesis Tests, Alternative Hypothesis

American Statistical Association (ASA), statement on use of p-values, p-value controversy

analysis of variance (ANOVA), F-Distribution, ANOVA-Further Reading

    decomposition of variance, F-Statistic

    F-statistic, F-Statistic

    two-way ANOVA, Two-Way ANOVA

anomaly detection, Regression and Prediction

    outliers and, Outliers

arms (multi-arm bandits), Multi-Arm Bandit Algorithm

AUC (area under the ROC curve), AUC-AUC

average linkage metric, Measures of Dissimilarity

average value (see mean)

B

b-spline (basis spline), Splines

backpropagation through time, RNNs

backward elimination, Model Selection and Stepwise Regression, Model Selection and Stepwise Regression

backward selection, Model Selection and Stepwise Regression

bagging, The Bootstrap, Resampling, Statistical Machine Learning, Bagging

    boosting versus, Boosting

bandit algorithms, Multi-Arm Bandit Algorithm

    (see also multi-arm bandits)

bandits, Multi-Arm Bandits

bar charts, Exploring Binary and Categorical Data

    histograms and, Exploring Binary and Categorical Data

batch, Backpropagation and Gradient Descent

Bayesian approach in Thompson's sampling, Multi-Arm Bandit Algorithm

Bayesian classification, Naive Bayes

    (see also naive Bayes algorithm)

Bayesian information criterion (BIC), Model Selection and Stepwise Regression, Selecting the Number of Clusters-Selecting the Number of Clusters

Bellman equation, Markov Decision Processes

BERT, Embeddings

beta distribution, Multi-Arm Bandit Algorithm

bias, Bias

    biased estimates from naive Bayes classifier, The Naive Solution

    selection bias, Selection Bias-Further Reading

bias in neural network, Weights and Biases

bias-variance trade-off, Choosing K

biased estimates, Standard Deviation and Related Estimates

bidirectional alternative hypothesis, One-Way Versus Two-Way Hypothesis Tests

big data

    predictive models in, Data and Sampling Distributions

    value of, Size Versus Quality: When Does Size Matter?

binary data, Elements of Structured Data

    exploring, Exploring Binary and Categorical Data-Further Reading

binary dummy variables, One-Hot Encoder

binary outcomes, Binomial Distribution

binary variables, Factor Variables in Regression, Scaling and Categorical Variables

binomial distribution, Binomial Distribution-Further Reading

binomial trials, Binomial Distribution

bins, in frequency tables and histograms, Frequency Tables and Histograms

biplot, Correspondence Analysis

bivariate analysis, Exploring Two or More Variables

black swan theory, Long-Tailed Distributions

blind studies, Why Have a Control Group?

boosted trees, Tree Models, Bagging and the Random Forest

boosting, Statistical Machine Learning, Boosting-Summary

    versus bagging, Boosting

boosting algorithm, The Boosting Algorithm

hyperparameters and cross-validation, Hyperparameters and Cross-Validation-Hyperparameters and Cross-Validation

regularization, avoiding overfitting with, Regularization: Avoiding Overfitting-Regularization: Avoiding Overfitting

XGBoost software, XGBoost-Regularization: Avoiding Overfitting

bootstrap, Central Limit Theorem, The Bootstrap-Further Reading

algorithm for bootstrap resampling of the mean, The Bootstrap

bootstrap and permutation tests, Exhaustive and Bootstrap Permutation Tests

confidence interval generation, Confidence Intervals, Confidence and Prediction Intervals-Confidence and Prediction Intervals

in resampling, Resampling

resampling versus bootstrapping, Resampling Versus Bootstrapping

sampling of variables in random forest partitioning, Random Forest

standard error and, Standard Error

bootstrap aggregating (see bagging)

bootstrap sample, The Bootstrap

boxplots, Exploring the Data Distribution

comparing numeric and categorical data, Categorical and Numeric Data

extending with conditional variables, Visualizing Multiple Variables

percentiles and, Percentiles and Boxplots-Percentiles and Boxplots

violin plots versus, Categorical and Numeric Data

Breiman, Leo, Statistical Machine Learning, Tree Models

bubble plots, Influential Values

C

categorical data

exploring, Exploring Binary and Categorical Data-Further Reading

expected value, Expected Value

mode, Mode

probability, Probability

exploring numeric variable grouped by categorical variable, Categorical and Numeric Data

exploring two categorical variables, Two Categorical Variables

importance of the concept, Elements of Structured Data

categorical variables, Factor Variables in Regression

(see also factor variables)

converting to dummy variables, Dummy Variables Representation

required for naive Bayes algorithm, Naive Bayes

scaling (see scaling and categorical variables)

causation, regression and, Prediction Versus Explanation (Profiling)

central limit theorem, Sampling Distribution of a Statistic, Central Limit Theorem

Student's t-distribution and, Student’s t-Distribution

central tendency (see estimates of location)

chain-of-thought, Task Decomposition and Chain-of-Thought Reasoning

chance variation, Statistical Significance and p-Values

chi-square distribution, Chi-Square Distribution, Chi-Square Test: Statistical Theory

chi-square statistic, Chi-Square Test, Chi-Square Test: A Resampling Approach

chi-square test, Chi-Square Test-Further Reading

    Fisher's exact test, Fisher's Exact Test-Relevance for Data Science

    relevance for data science, Relevance for Data Science

    resampling approach, Chi-Square Test: A Resampling Approach

    statistical theory, Chi-Square Test: Statistical Theory

class purity, Measuring Homogeneity or Impurity

classification, Classification-Summary

    discriminant analysis, Discriminant Analysis-Further Reading

        covariance matrix, Covariance Matrix

        Fisher's linear discriminant, Fisher's Linear Discriminant

        simple example, A Simple Example-Extensions of Discriminant Analysis

    evaluating models, Evaluating Classification Models-Further Reading

        AUC metric, AUC-AUC

        confusion matrix, Confusion Matrix-Confusion Matrix

        lift, Lift

        precision, recall, and specificity, Precision, Recall, and Specificity

rare class problem, The Rare Class Problem

ROC curve, ROC Curve-ROC Curve

logistic regression, Logistic Regression-Further Reading

comparison to linear regression, Linear and Logistic Regression: Similarities and Differences-Fitting the model

generalized linear models, Logistic Regression and the GLM-Generalized Linear Models

interpreting coefficients and odds ratio, Interpreting the Coefficients and Odds Ratios

logistic response function and logit, Logistic Response Function and Logit

predicted values from, Predicted Values from Logistic Regression

naive Bayes algorithm, Naive Bayes-Further Reading

applying to numeric predictor variables, Numeric Predictor Variables

predicting more than two classes, Classification

strategies for imbalanced data, Strategies for Imbalanced Data-Summary

cost-based classification, Cost-Based Classification

data generation, Data Generation

exploring the predictions, Exploring the Predictions-Exploring the Predictions

oversampling and up/down weighting, Oversampling and Up/Down Weighting

undersampling, Undersampling

unsupervised learning as building block, Unsupervised Learning

Classification and Regression Trees (CART), Tree Models

(see also tree models)

cluster mean, K-Means Clustering, A Simple Example, Interpreting the Clusters

PCA loadings versus, Analyzing the Clusters with AI

clustering, Unsupervised Learning

categorical variables posing problems in, Scaling and Categorical Variables

hierarchical, Hierarchical Clustering-Measures of Dissimilarity

agglomerative algorithm, The Agglomerative Algorithm

dissimilarity metrics, Measures of Dissimilarity-Measures of Dissimilarity

representation in a dendrogram, The Dendrogram-The Dendrogram

simple example, A Simple Example

K-means, K-Means Clustering-Selecting the Number of Clusters

interpreting the clusters, Interpreting the Clusters-Analyzing the Clusters with AI

K-means algorithm, K-Means Algorithm

selecting the number of clusters, Selecting the Number of Clusters

simple example, A Simple Example

model-based, Model-Based Clustering-Further Reading

mixtures of normals, Mixtures of Normals-Mixtures of Normals

multivariate normal distribution, Multivariate Normal Distribution

selecting the number of clusters, Selecting the Number of Clusters-Selecting the Number of Clusters

problems with clustering mixed data, Problems with Clustering Mixed Data

uses of, Unsupervised Learning

clusters, K-Means Clustering

CNN, Advantages of Depth, CNNs

coefficient of determination, Assessing the Model

coefficients

confidence intervals and, Confidence and Prediction Intervals

in logistic regression, Interpreting the Coefficients and Odds Ratios

in multiple linear regression, Example: King County Housing Data

in simple linear regression, The Regression Equation

cold-start problems, using clustering for, Unsupervised Learning

complete-linkage method, The Agglomerative Algorithm

conditional probability, Naive Bayes

conditioning variables, Visualizing Multiple Variables

confidence intervals, Confidence and Prediction Intervals

algorithm for bootstrap confidence interval, Confidence Intervals

application to data science, Confidence Intervals

level of confidence, Confidence Intervals

prediction intervals versus, Confidence and Prediction Intervals

confounding variables, Interpreting the Regression Equation, Confounding Variables-Confounding Variables

confusion matrix, Evaluating Classification Models, Confusion Matrix-Confusion Matrix

contextual bandits, Multi-Arm Bandits

contingency tables, Exploring Two or More Variables

summarizing two categorical variables, Two Categorical Variables

continuous data, Elements of Structured Data

contour plots, Exploring Two or More Variables

using with hexagonal binning, Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)-Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)

contrast coding, Dummy Variables Representation

control group, A/B Testing

benefits of using, Why Have a Control Group?

convolution, CNNs

convolution operator, Intuition Behind CNNs

Cook's distance, Influential Values

correlated predictor variables, Correlated Predictors

correlation, Correlation-Further Reading

example, correlation between ETF returns, Correlation

key concepts, Scatterplots

key terms for, Correlation

scatterplots, Scatterplots

correlation coefficient, Correlation

calculating Pearson's correlation coefficient, Correlation

other types of, Correlation

correlation matrix, Correlation, Scaling the Variables

correspondence analysis, Correspondence Analysis-Analyzing the Results of Principal Components and Correspondence Analysis with AI

cost-based classification, Cost-Based Classification

covariance, Covariance Matrix, Computing the Principal Components

covariance matrix, Covariance Matrix, Distance Metrics, Multivariate Normal Distribution

cross-entropy loss, Loss Functions for Classification

cross-modal inconsistency, Misinterpreting Visual Information

cross-validation, Cross-Validation, Model Selection and Stepwise Regression, Choosing K

using for hyperparameters, Hyperparameters and Cross-Validation-Hyperparameters and Cross-Validation

using to select principal components, Interpreting Principal Components

using to test values of hyperparameters, Hyperparameters

cumulative gains chart, Lift

D

d.f. (degrees of freedom), Degrees of Freedom

(see also degrees of freedom)

data analysis, Exploratory Data Analysis

(see also exploratory data analysis)

data catalog, Data Dictionaries and Catalogs

data dictionary, Data Dictionaries and Catalogs

data distribution, Exploring the Data Distribution-Exploring Binary and Categorical Data, Sampling Distribution of a Statistic

density plots and estimates, Density Plots and Estimates-Exploring Binary and Categorical Data

frequency table and histogram, Frequency Tables and Histograms-Density Plots and Estimates

percentiles and boxplots, Percentiles and Boxplots-Percentiles and Boxplots

sampling distribution versus, Sampling Distribution of a Statistic

data frames, Rectangular Data

histograms for, Frequency Tables and Histograms

and indexes, Data Frames and Indexes

typical data frame, Rectangular Data

data generation, Strategies for Imbalanced Data, Data Generation

data quality, Random Sampling and Sample Bias

sample size versus, Size Versus Quality: When Does Size Matter?

data science

A/B testing in, Why Have a Control Group?

multiplicity and, Multiple Testing

p-values and, Data Science and p-Values

permutation tests, value of, Permutation Tests: The Bottom Line for Data Science

relevance of chi-square tests, Relevance for Data Science

t-statistic and, Assessing the Model

value of heteroskedasticity for, Heteroskedasticity, Nonnormality, and Correlated Errors

data snooping, Selection Bias

data types

key terms for, Elements of Structured Data

resources for further reading, Further Reading

data-centric approach, excessive, Long-Tailed Distributions

database normalization versus normalization in statistics, Standardization (Normalization, z-Scores)

databases, data types in, Further Reading

decile gains charts, Lift

decision trees, Tree Models

ensemble learning applied to, Statistical Machine Learning

older meaning in human decision analysis, Tree Models

running multiple on bootstrap samples, The Bootstrap

decoder, Architecture Overview

decomposition of variance, ANOVA, F-Statistic

degrees of freedom, Standard Deviation and Related Estimates, Degrees of Freedom-Further Reading, F-Statistic

for chi-square distribution, Chi-Square Test: Statistical Theory

t-distribution and, Student's t-Distribution

dendrograms, Hierarchical Clustering, The Dendrogram-The Dendrogram, Categorical Data and Gower's Distance

density plots, Exploring the Data Distribution

and estimates, Density Plots and Estimates

departure from expectation, Chi-Square Distribution

dependent variables, Data Frames and Indexes, The Regression Equation

(see also response)

deviance, Fitting the model

attempt to minimize in logistic regression, Oversampling and Up/Down Weighting

deviation coding, Factor Variables in Regression, Dummy Variables Representation

deviations

defined, Estimates of Variability

standard deviation and related estimates, Standard Deviation and Related Estimates

discrete data, Elements of Structured Data

discriminant analysis, Discriminant Analysis-Further Reading

covariance matrix, Covariance Matrix

Fisher's linear discriminant, Fisher’s Linear Discriminant

linear discriminant analysis (LDA), Discriminant Analysis

simple example, A Simple Example-Extensions of Discriminant Analysis

variants of, Extensions of Discriminant Analysis

discriminant function, Discriminant Analysis

discriminant weights, Discriminant Analysis

dispersion, Estimates of Variability

(see also variability)

dissimilarity metrics, A Simple Example, Measures of Dissimilarity-Measures of Dissimilarity

complete-linkage method, The Agglomerative Algorithm

distance metrics, K-Nearest Neighbors, Distance Metrics

Gower's distance, Categorical Data and Gower’s Distance

in hierarchical clustering, A Simple Example, The Agglomerative Algorithm

Manhattan distance, Categorical Data and Gower’s Distance

dominant variables, Dominant Variables-Categorical Data and Gower’s Distance

Donoho, David, Exploratory Data Analysis

double descent, Advantages of Depth

double-blind studies, Why Have a Control Group?

downsampling, Undersampling

(see also undersampling)

dummy variables, Factor Variables in Regression, One-Hot Encoder

representation of factor variables in regression, Dummy Variables Representation

Durbin-Watson statistic, Heteroskedasticity, Nonnormality, and Correlated Errors

E

effect size, Power and Sample Size, Sample Size

elbow method, Selecting the Number of Clusters, Selecting the Number of Clusters

embeddings, Embeddings

emergent capabilities, Emergent Capabilities

encoder, Architecture Overview

ensemble learning, Statistical Machine Learning

staged use of K-nearest neighbors, KNN as a Feature Engine

ensemble of models, Bagging and the Random Forest

bagging and boosting, Bagging and the Random Forest

creating using boosting, Boosting

entropy of information, Measuring Homogeneity or Impurity

epoch, Backpropagation and Gradient Descent

Epsilon first strategy, Multi-Arm Bandits

Epsilon greedy strategy, Multi-Arm Bandits

epsilon-greedy algorithm for A/B test, Multi-Arm Bandit Algorithm

errors, Normal Distribution

prediction errors, Fitted Values and Residuals

(see also residuals)

estimates

hat notation and, Fitted Values and Residuals

metrics and, Estimates of Location

estimates of location, Estimates of Location-Estimates of Variability

Euclidean distance, Distance Metrics, The Agglomerative Algorithm

exact tests, Exhaustive and Bootstrap Permutation Tests

Fisher's exact test, Fisher’s Exact Test-Relevance for Data Science

exhaustive permutation tests, Exhaustive and Bootstrap Permutation Tests

expectation or expected, Chi-Square Test

departure from, Chi-Square Distribution

expected value, Exploring Binary and Categorical Data, Expected Value

explanation (profiling), prediction versus, Prediction Versus Explanation (Profiling)

exploratory data analysis, Exploratory Data Analysis-Summary

categorical and binary data, Elements of Structured Data-Further Reading, Exploring Binary and Categorical Data-Further Reading

correlation, Correlation-Further Reading

data distribution, Exploring the Data Distribution-Exploring Binary and Categorical Data

estimates of location, Estimates of Location-Estimates of Variability

estimates of variability, Estimates of Variability-Exploring the Data Distribution

exploring two or more variables, Exploring Two or More Variables-Summary

for predictions from classification models, Exploring the Predictions

unsupervised learning as extension of, Unsupervised Learning

Exploratory Data Analysis (Tukey), Exploratory Data Analysis

exponential distribution, Exponential Distribution

extrapolation

dangers of, The Dangers of Extrapolation

defined, Prediction Using Regression

F

F-distribution, F-Distribution

F-statistic, ANOVA, F-Statistic, Assessing the Model

facet plots, Exploring Two or More Variables

facets, Visualizing Multiple Variables

factor variables, Elements of Structured Data, Degrees of Freedom, Factor Variables in Regression-Ordered Factor Variables

binary, odds ratio for, Interpreting the Coefficients and Odds Ratios

coding in logistic regression, Fitting the model

different codings, Dummy Variables Representation

dummy variable representations, Dummy Variables Representation

in naive Bayes algorithm, Naive Bayes

ordered, Ordered Factor Variables

with many levels, Factor Variables with Many Levels-Factor Variables with Many Levels

failure rate, estimating, Estimating the Failure Rate

false discovery rate, Multiple Testing, Multiple Testing

false positive rate, Confusion Matrix

feature selection

chi-square tests in, Relevance for Data Science

in stepwise regression, Model Selection and Stepwise Regression

using discriminant analysis for, A Simple Example

features, Rectangular Data, The Regression Equation

(see also predictor variables)

K-nearest neighbors as feature engine, KNN as a Feature Engine-KNN as a Feature Engine

terminology differences, Data Frames and Indexes

few-shot learning, Prompt engineering in production use

field view (spatial data), Nonrectangular Data Structures

filter

convolution, Intuition Behind CNNs

Fisher's exact test, Fisher’s Exact Test-Relevance for Data Science

relevance for data science, Relevance for Data Science

Fisher's linear discriminant, Fisher’s Linear Discriminant

Fisher's scoring, Fitting the model

Fisher, R.A., Discriminant Analysis

fitted values, Simple Linear Regression

in multiple linear regression, Multiple Linear Regression

in simple linear regression, Fitted Values and Residuals

fitting the model

bias-variance trade-off, Choosing K

K-nearest neighbors, advantages of, KNN as a Feature Engine

linear versus logistic regression, Fitting the model

random forest fit to loan default data, Variable Importance

rules for simple tree model fit to loan data, A Simple Example

folds, Cross-Validation, Hyperparameters and Cross-Validation

forward selection, Model Selection and Stepwise Regression, Model Selection and Stepwise Regression

foundation models, Training LLMs and Adapting LLMs for Specific Purposes

frequency tables, Exploring the Data Distribution

example, murder rates by state, Frequency Tables and Histograms

Friedman, Jerome H. (Jerry), Statistical Machine Learning

fully-connected network, The Basic Structure of a Neural Net

G

gains, Lift

(see also lift)

Gallup Poll, Random Sampling and Sample Bias

Galton, Francis, Regression to the Mean, Bagging and the Random Forest

gated recurrent units (GRUs), Memory Issues

Gauss, Carl Friedrich, Least Squares

Gaussian distribution, Normal Distribution

(see also normal distribution)

generalized additive models (GAM), Polynomial and Spline Regression, Generalized Additive Models-Generalized Additive Models

logistic regression fit with, Exploring the Predictions

generalized linear models (GLMs), Logistic Regression and the GLM-Generalized Linear Models

characteristics and applications of, Generalized Linear Models

Gini coefficient, Measuring Homogeneity or Impurity

Gini impurity, Measuring Homogeneity or Impurity, Variable Importance

Google Analytics, Example: Web Stickiness

Gosset, W. S., Student’s t-Distribution

Gower's distance, Scaling and Categorical Variables

using to scale categorical variables, Categorical Data and Gower’s Distance

gradient boosting, Boosting, The Boosting Algorithm

gradient descent, Fitting a Network to Data

graphs, Nonrectangular Data Structures

in computer science versus statistics, Nonrectangular Data Structures

greedy algorithms, Multi-Arm Bandit Algorithm

H

hallucinations, Creativity and Temperature, Accuracy and Reliability

hat notation, estimates versus known values, Fitted Values and Residuals

hat-value, Regression Diagnostics, Influential Values

heat maps, Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)

heteroskedastic errors, Heteroskedasticity, Nonnormality, and Correlated Errors

heteroskedasticity, Regression Diagnostics, Heteroskedasticity, Nonnormality, and Correlated Errors

value to data science, Heteroskedasticity, Nonnormality, and Correlated Errors

hexagonal binning, Exploring Two or More Variables

and contours, plotting relationship between two numeric values, Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)-Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)

extending with conditional variables, Visualizing Multiple Variables

hidden state representations, Embeddings

hierarchical clustering, Hierarchical Clustering-Measures of Dissimilarity

agglomerative algorithm, The Agglomerative Algorithm

dissimilarity metrics, Measures of Dissimilarity-Measures of Dissimilarity

representation in dendrogram, The Dendrogram-The Dendrogram

simple example, A Simple Example

using with Gower's distance, Categorical Data and Gower’s Distance

histograms, Exploring the Data Distribution

bar charts and, Exploring Binary and Categorical Data

plotting of, Frequency Tables and Histograms

visualizing frequency tables with, Frequency Tables and Histograms

holdout data, Hyperparameter Optimization

homogeneity, measuring, Measuring Homogeneity or Impurity

hyperparameters, Hyperparameters, Boosting

hypothesis tests, Hypothesis Tests-Further Reading

alternative hypothesis, Alternative Hypothesis

false discovery rate, Multiple Testing

misinterpreting randomness, Hypothesis Tests

null hypothesis, The Null Hypothesis

one-way and two-way tests, One-Way Versus Two-Way Hypothesis Tests

structured to minimize type 1 errors, Type 1 and Type 2 Errors

I

if-then-else rules (tree models), Tree Models

imbalanced data strategies for classification models, Strategies for Imbalanced Data-Summary, A Small Example: Predicting Loan Default

cost-based classification, Cost-Based Classification

data generation, Data Generation

exploring the predictions, Exploring the Predictions-Exploring the Predictions

oversampling and up/down weighting, Oversampling and Up/Down Weighting

impurity, Tree Models

measuring, Measuring Homogeneity or Impurity, Predicting a Continuous Value

in-context learning, Pattern Matching and In-Context Learning

in-sample validation methods, Model Selection and Stepwise Regression

independent variables, Simple Linear Regression, The Regression Equation

(see also predictor variables)

indexes, data frames and, Data Frames and Indexes

indicator variables, Factor Variables in Regression

inference, Exploratory Data Analysis, Statistical Experiments and Significance Testing

influence plots, Influential Values

influential values, Regression Diagnostics, Influential Values

intelligent social agent, Personal Companions

interactions, Interpreting the Regression Equation, Variable Importance

and main effects, Interactions and Main Effects-Interactions and Main Effects

intercept, Simple Linear Regression, The Regression Equation

Internet of Things (IoT), Elements of Structured Data

interquartile range, Estimates of Variability, Estimates Based on Percentiles

calculating, Example: Variability Estimates of State Population

interval endpoints, Confidence Intervals

inverse logit function, Logistic Response Function and Logit

inverse odds function, Logistic Response Function and Logit

IQR (see interquartile range)

ISA, Personal Companions

K

K (in K-means clustering), K-Means Clustering, Selecting the Number of Clusters

K (in K-nearest neighbors), Strategies for Imbalanced Data, K-Nearest Neighbors, KNN as a Feature Engine

k-fold cross-validation, Cross-Validation

K-means clustering, K-Means Clustering-Selecting the Number of Clusters

applying to data without normalization, Scaling the Variables

applying to normalized data, Scaling the Variables

interpreting the clusters, Interpreting the Clusters-Analyzing the Clusters with AI

K-means algorithm, K-Means Algorithm

selecting the number of clusters, Selecting the Number of Clusters

simple example, A Simple Example

using with binary data, Problems with Clustering Mixed Data

K-nearest neighbors, Statistical Machine Learning, K-Nearest Neighbors-KNN as a Feature Engine

categorical data and, Scaling and Categorical Variables

choosing K, Choosing K

distance metrics, Distance Metrics

example, predicting loan default, A Small Example: Predicting Loan Default-A Small Example: Predicting Loan Default

as a feature engine, KNN as a Feature Engine-KNN as a Feature Engine

one-hot encoder and, One-Hot Encoder

standardization in, Standardization (Normalization, z-Scores)-Standardization (Normalization, z-Scores)

Kendall's tau, Correlation

kernel, Intuition Behind CNNs

kernel density estimates, Exploring the Data Distribution, Density Plots and Estimates

(see also density plots)

KNN (see K-nearest neighbors)

knots, Polynomial and Spline Regression, Splines

kurtosis, Frequency Tables and Histograms

L

lasso regression, Model Selection and Stepwise Regression, Regularization: Avoiding Overfitting

latent Dirichlet allocation, Discriminant Analysis

layers in neural networks, The Basic Structure of a Neural Net

leaf, Tree Models

learning rate, Backpropagation and Gradient Descent

least squares regression, Simple Linear Regression, Least Squares, Regularization: Avoiding Overfitting

Legendre, Adrien-Marie, Least Squares

level of confidence, Confidence Intervals

leverage

    defined, Regression Diagnostics

    influential values in regression, Influential Values

lift, Evaluating Classification Models, Lift

    lift curve, Lift

    uplift and, Lift

linear discriminant analysis (LDA), Discriminant Analysis, Exploring the Predictions

    (see also discriminant analysis)

    principal components analysis as unsupervised version, Principal Components Analysis

linear discriminant function, A Simple Example, Extensions of Discriminant Analysis

linear model (lm), The Regression Equation, Multiple Linear Regression

linear regression

- comparison to logistic regression, Linear and Logistic Regression: Similarities and Differences-Fitting the model
- examination of residuals to see if fit can be improved, Boosting
- generalized linear model (GLM), Logistic Regression and the GLM
- multicollinearity problems caused by one-hot encoding, One-Hot Encoder
- multiple, Multiple Linear Regression-Further Reading
  - assessing the model, Assessing the Model
  - cross-validation, Cross-Validation
    - example, estimating value of houses, Example: King County Housing Data-Example: King County Housing Data
    - model selection and stepwise regression, Model Selection and Stepwise Regression-Model Selection and Stepwise Regression
    - weighted regression, Weighted Regression-Further Reading
- prediction versus explanation, Prediction Versus Explanation (Profiling)
- simple, Simple Linear Regression-Further Reading
  - fitted values and residuals, Fitted Values and Residuals
  - least squares, Least Squares
    - regression equation, The Regression Equation-Fitted Values and Residuals

Literary Digest poll of 1936, Random Sampling and Sample Bias

loadings, Principal Components Analysis, A Simple Example

cluster mean versus, Analyzing the Clusters with AI

plotting for top principal components, Interpreting Principal Components

with negative signs, A Simple Example

location, estimates of, Estimates of Location-Estimates of Variability

log-loss, Loss Functions for Classification

log-odds function (see logit function)

log-odds ratio and odds ratio, Interpreting the Coefficients and Odds Ratios

logistic linear regression, Exploring the Predictions

logistic regression, Logistic Regression-Further Reading

assessing the model, Assessing the Model-Analysis of residuals

comparison to linear regression, Linear and Logistic Regression: Similarities and Differences-Fitting the model

fit using generalized additive model, Exploring the Predictions

and the generalized linear model, Logistic Regression and the GLM-Generalized Linear Models

interpreting coefficients and odds ratio, Interpreting the Coefficients and Odds Ratios

logistic response function and logit, Logistic Response Function and Logit

multicollinearity problems caused by one-hot encoding, One-Hot Encoder

predicted values from, Predicted Values from Logistic Regression

logistic response function, Logistic Response Function and Logit

logit function, Logistic Response Function and Logit, Logistic Response Function and Logit

long short-term memory (LSTM), Memory Issues

long-tail distributions, Long-Tailed Distributions-Student's t-Distribution

LoRA, Fine-Tuning

loss, Tree Models

in simple tree model example, A Simple Example

loss function, Oversampling and Up/Down Weighting

low-rank adaptation, Fine-Tuning

M

machine learning, Statistical Machine Learning

(see also statistical machine learning; supervised learning; unsupervised learning)

overfitting risk, mitigating, Multiple Testing

statistics versus, Statistical Machine Learning

use of resampling to improve models, Resampling

MAD (see median absolute deviation from the median)

MAE, Loss Functions for Regression

Mahalanobis distance, Covariance Matrix, Distance Metrics

main effects, Interpreting the Regression Equation

interactions and, Interactions and Main Effects-Interactions and Main Effects

Mallows Cp, Model Selection and Stepwise Regression

Manhattan distance, Distance Metrics, Regularization: Avoiding Overfitting, Categorical Data and Gower's Distance

Markov decision process, Markov Decision Processes

maximum likelihood estimation (MLE), Fitting the model, Analysis of residuals

MDPs, Markov Decision Processes

mean, Estimates of Location, Mean

regression to, Regression to the Mean

sample mean versus population mean, Sample Mean Versus Population Mean

trimmed mean, Mean

weighted mean, Mean

mean absolute deviation, Estimates of Variability

formula for calculating, Standard Deviation and Related Estimates

mean absolute error, Loss Functions for Regression

mean squared error, Loss Functions for Regression

median, Estimates of Location, Median and Robust Estimates

median absolute deviation from the median (MAD), Estimates of Variability, Standard Deviation and Related Estimates

medical screening tests, false positives and, Confusion Matrix

metrics, Estimates of Location

minimum variance metric, Measures of Dissimilarity

mixture of experts, Mixture of Experts Models

mode, Exploring Binary and Categorical Data

    examples in categorical data, Mode

model collapse, Model Collapse

model-based clustering, Model-Based Clustering-Further Reading

    limitations of, Selecting the Number of Clusters

    mixtures of normals, Mixtures of Normals-Mixtures of Normals

    multivariate normal distribution, Multivariate Normal Distribution

    selecting the number of clusters, Selecting the Number of Clusters-Selecting the Number of Clusters

MoE, Mixture of Experts Models

moments (of a distribution), Frequency Tables and Histograms

MSE, Loss Functions for Regression

multi-arm bandits, Why Just A/B? Why Not C, D,…?, Multi-Arm Bandit Algorithm-Further Reading, Multi-Arm Bandits

multicollinearity, Interpreting the Regression Equation, Multicollinearity

    and predictors used twice in KNN, KNN as a Feature Engine

    problems caused by one-hot encoding, One-Hot Encoder

multicollinearity errors, Degrees of Freedom, Dummy Variables Representation

multiple testing, Multiple Testing-Further Reading

multivariate analysis, Exploring Two or More Variables-Summary

multivariate bootstrap sampling, The Bootstrap

multivariate normal distribution, Multivariate Normal Distribution

mixtures of normals, Mixtures of Normals-Mixtures of Normals

N

N (or n) referring to total records, Mean

n or n – 1, dividing by in variance formula, Standard Deviation and Related Estimates

n or n – 1, dividing by in variance or standard deviation formula, Degrees of Freedom

n or sample size, Degrees of Freedom

naive Bayes algorithm, Naive Bayes-Further Reading

numeric predictor variables with, Numeric Predictor Variables

solution, The Naive Solution-The Naive Solution

why exact Bayesian classification is impractical, Why Exact Bayesian Classification Is Impractical

neighbors (in K-nearest neighbors), K-Nearest Neighbors

network data structures, Nonrectangular Data Structures

neurons, The Basic Structure of a Neural Net

nodes, Tree Models, The Basic Structure of a Neural Net

nonlinear regression, Polynomial and Spline Regression

nonnormal residuals, Regression Diagnostics, Heteroskedasticity, Nonnormality, and Correlated Errors

nonrandom samples, Random Sampling and Sample Bias

nonrectangular data structures, Nonrectangular Data Structures

normal distribution, Normal Distribution-Standard Normal and QQ-Plots

key concepts, Standard Normal and QQ-Plots

misconceptions about, Normal Distribution

multivariate, Multivariate Normal Distribution

standard normal and QQ-Plots, Standard Normal and QQ-Plots

normalization, Standard Normal and QQ-Plots, Standardization (Normalization, z-Scores), Scaling and Categorical Variables

(see also standardization)

in statistics, versus database normalization, Standardization (Normalization, z-Scores)

scaling the variables, Scaling the Variables-Scaling the Variables

null hypothesis, Hypothesis Tests, The Null Hypothesis, Statistical Significance and p-Values

in click rate testing for web headlines, Chi-Square Test: A Resampling Approach

using alternative hypothesis with, Alternative Hypothesis

numeric data, Elements of Structured Data

dimension reduction of, Summary

exploring relationship between two numeric variables, Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)-Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)

grouped by categorical variable, exploring, Categorical and Numeric Data

numeric variables

conversion of factor variables to, in regression, Factor Variables in Regression

converting ordered factor variables to, Ordered Factor Variables

Mahalanobis distance, Distance Metrics

O

object representation (spatial data), Nonrectangular Data Structures

Occam's razor, Model Selection and Stepwise Regression

odds, Logistic Response Function and Logit

obtaining probability from, Logistic Response Function and Logit

odds ratio, Interpreting the Coefficients and Odds Ratios

relationship with log-odds ratio, Interpreting the Coefficients and Odds Ratios

omnibus tests, ANOVA

one-hot encoding, Factor Variables in Regression, Dummy Variables Representation, One-Hot Encoder, Scaling and Categorical Variables

one-way ANOVA, Two-Way ANOVA

one-way tests, Hypothesis Tests, One-Way Versus Two-Way Hypothesis Tests

order statistics, Estimates of Variability, Estimates Based on Percentiles

ordered factor variables, Ordered Factor Variables

ordinal data, Elements of Structured Data

importance of the concept, Elements of Structured Data

ordinary least squares (OLS) regression, Least Squares, Heteroskedasticity, Nonnormality, and Correlated Errors

out-of-bag estimate of error, Random Forest

out-of-sample validation, Cross-Validation

outcome, Rectangular Data

outliers, Estimates of Location, Outliers

- correlation coefficient and, Correlation
- in boxplots, Percentiles and Boxplots
- in regression diagnostics, Outliers-Outliers

overfitting, Multiple Testing

- avoiding using regularization, Regularization: Avoiding Overfitting-Regularization: Avoiding Overfitting

oversampling, Strategies for Imbalanced Data

- and up/down weighting, Oversampling and Up/Down Weighting

P

p-values, Statistical Significance and p-Values-Further Reading, ANOVA

- adjustment of, Multiple Testing
- alpha, Alpha
- chi-square distribution and, Chi-Square Test: Statistical Theory
- controversy over use of, p-value controversy
- data science and, Data Science and p-Values
- practical significance and, Practical significance

t-statistic and, Assessing the Model

type 1 and type 2 errors, Type 1 and Type 2 Errors

pairwise comparisons, ANOVA, ANOVA

partial residual plots, Regression Diagnostics, Partial Residual Plots and Nonlinearity

for spline regression, Splines

in logistic regression, Analysis of residuals

nonlinearity and, Partial Residual Plots and Nonlinearity

partitions in trees, Tree Models, How Trees Are Used

random forests, Random Forest

recursive partitioning algorithm, The Recursive Partitioning Algorithm-The Recursive Partitioning Algorithm

PCA (see principal components analysis)

Pearson residuals, Chi-Square Test: A Resampling Approach

Pearson's chi-square test, Chi-Square Test

Pearson's correlation coefficient, Correlation

Pearson, Karl, Principal Components Analysis

penalized regression, Model Selection and Stepwise Regression, Interactions and Main Effects

penalty on model complexity, Regularization: Avoiding Overfitting

percentiles, Estimates of Location, Estimates of Variability

and boxplots, Percentiles and Boxplots-Percentiles and Boxplots

estimates based on, Estimates Based on Percentiles

precise definition of, Estimates Based on Percentiles

permissions for scientific and medical testing, Why Have a Control Group?

permutation tests, Permutation Test-Permutation Tests: The Bottom Line for Data Science, Statistical Significance and p-Values

for ANOVA, ANOVA

for chi-square test, Chi-Square Test: A Resampling Approach

estimating p-values from, p-Value

exhaustive and bootstrap, Exhaustive and Bootstrap Permutation Tests

value for data science, Permutation Tests: The Bottom Line for Data Science

web stickiness example, Example: Web Stickiness-Exhaustive and Bootstrap Permutation Tests

pertinent records (in searches), Size Versus Quality: When Does Size Matter?

pie charts, Exploring Binary and Categorical Data, Exploring Binary and Categorical Data

pivot tables, Two Categorical Variables

(see also contingency tables)

point estimates, Confidence Intervals

Poisson distributions, Poisson Distributions, Generalized Linear Models

policy, Markov Decision Processes

polynomial coding, Dummy Variables Representation

polynomial regression, Polynomial

Pooh

Winnie the, Big Brother

population, Random Sampling and Sample Bias

population mean versus sample mean, Sample Mean Versus Population Mean

posterior probability, Naive Bayes, The Naive Solution

power and sample size, Power and Sample Size-Summary

calculating, components in, Sample Size

power, Power and Sample Size

sample size, Sample Size

practical significance versus statistical significance, Practical significance

precision, Evaluating Classification Models, Precision, Recall, and Specificity

precision-recall (PR) curve, ROC Curve

predicted values, Fitted Values and Residuals

(see also fitted values)

prediction, Classification

(see also classification)

exploring predictions from classification models, Exploring the Predictions

fitted values and residuals in simple linear regression, Fitted Values and Residuals

from random forests, Random Forest

from XGBoost applied to loan default data, XGBoost

harnessing results from multiple trees, How Trees Are Used

predicted values from logistic regression, Predicted Values from Logistic Regression

predicting a continuous value with tree model, Predicting a Continuous Value

predicting loan default with K-nearest neighbors, A Small Example: Predicting Loan Default-A Small Example: Predicting Loan Default

prediction versus explanation in simple linear regression, Prediction Versus Explanation (Profiling)

unsupervised learning and, Unsupervised Learning

using regression, Regression and Prediction, Prediction Using Regression-Factor Variables in Regression

confidence and prediction intervals, Confidence and Prediction Intervals-Factor Variables in Regression

dangers of extrapolation, The Dangers of Extrapolation

prediction intervals, Further Reading, Confidence and Prediction Intervals

confidence intervals versus, Confidence and Prediction Intervals

predictive modeling

KNN as first stage for, KNN as a Feature Engine, KNN as a Feature Engine

machine learning versus statistics, Statistical Machine Learning

predictor variables, Data Frames and Indexes, The Regression Equation

correlated, Correlated Predictors

isolating relationship between response and, Partial Residual Plots and Nonlinearity

main effects and interactions, Interactions and Main Effects-Interactions and Main Effects

nonlinear relationships among, captured by trees, How Trees Are Used

numeric, applying naive Bayes algorithm to, Numeric Predictor Variables

redundancy in, Multicollinearity

standardization in K-nearest neighbors, Standardization (Normalization, z-Scores)

t-statistic and, Assessing the Model

used twice in KNN, KNN as a Feature Engine

using more than two in linear discriminant analysis, Extensions of Discriminant Analysis

principal components, Principal Components Analysis

principal components analysis, Principal Components Analysis-Further Reading, Problems with Clustering Mixed Data

cluster analysis versus, Analyzing the Clusters with AI

computing principal components, Computing the Principal Components

correspondence analysis, Correspondence Analysis-Analyzing the Results of Principal Components and Correspondence Analysis with AI

deciding how many components to choose, Interpreting Principal Components

interpreting principal components, Interpreting Principal Components-Interpreting Principal Components

scaling of variables, Scaling the Variables

simple example, A Simple Example-A Simple Example

using with binary data, Problems with Clustering Mixed Data

probability, Probability, Classification

associated with a confidence interval, Confidence Intervals

output by K-nearest neighbors, A Small Example: Predicting Loan Default

produced by tree models, The Recursive Partitioning Algorithm

propensity, Classification

(see also probability)

proxy variables, Example: Web Stickiness

pruning, Tree Models, Stopping the Tree from Growing

pseudo-residuals, The Boosting Algorithm

Q

QQ-plots, Normal Distribution

QQ-Plots

for returns of NFLX stock, Long-Tailed Distributions

standard normal distribution and, Standard Normal and QQ-Plots

quadratic discriminant analysis (QDA), Extensions of Discriminant Analysis

quantiles, Estimates Based on Percentiles

(see also percentiles)

functions for, Estimates Based on Percentiles

R

R-squared, Multiple Linear Regression, Assessing the Model

R-Tutorial website, Further Reading

random forests, Statistical Machine Learning, Tree Models, Bagging and the Random Forest, Random Forest-Random Forest

hyperparameters, Hyperparameters

variable importance in, Variable Importance-Variable Importance

random numbers, generation from Poisson distribution, Poisson Distributions

random sampling, Random Sampling and Sample Bias-Selection Bias

bias, Bias

random selection, Random Selection

sample mean versus population mean, Sample Mean Versus Population Mean

size versus quality, Size Versus Quality: When Does Size Matter?

random subset of variables (in random forest), Random Forest

randomization, A/B Testing

randomness, underestimating and misinterpreting, Hypothesis Tests

range, Estimates of Variability, Estimates Based on Percentiles

ranking records, naive Bayes algorithm, The Naive Solution

rare class problem, The Rare Class Problem

recall, Precision, Recall, and Specificity

(see also sensitivity)

trade-off with specificity, ROC Curve

receiver operating characteristics curve (see ROC curve)

records, Rectangular Data, Simple Linear Regression

rectangular data, Data Structures-Estimates of Location

key terms for, Rectangular Data

terminology differences, Data Frames and Indexes

rectified linear unit, Activation Functions in the Hidden Layers

recurrent neural network, RNNs, Memory Issues

recursive partitioning, Tree Models, The Recursive Partitioning Algorithm-The Recursive Partitioning Algorithm

reducing the dimension of the data, Unsupervised Learning

reference coding, Factor Variables in Regression, Dummy Variables Representation, Interactions and Main Effects, Logistic Regression and the GLM

regression, Regression and Prediction-Summary

ANOVA as first step toward statistical model, Two-Way ANOVA

causation caution, Prediction Versus Explanation (Profiling)

diagnostics, Regression Diagnostics-Polynomial and Spline Regression

heteroskedasticity, nonnormality, and correlated errors, Heteroskedasticity, Nonnormality, and Correlated Errors-Heteroskedasticity, Nonnormality, and Correlated Errors

influential values, Influential Values

outliers, Outliers

partial residual plots and nonlinearity, Partial Residual Plots and Nonlinearity

factor variables in, Degrees of Freedom, Factor Variables in Regression-Ordered Factor Variables

dummy variables representation, Dummy Variables Representation

ordered factor variables, Ordered Factor Variables

with many levels, Factor Variables with Many Levels-Factor Variables with Many Levels

interpreting the regression equation, Interpreting the Regression Equation-Interactions and Main Effects

confounding variables, Confounding Variables-Confounding Variables

correlated predictor variables, Correlated Predictors

interactions and main effects, Interactions and Main Effects-Interactions and Main Effects

multicollinearity, Multicollinearity

logistic regression, Logistic Regression-Further Reading

comparison to linear regression, Linear and Logistic Regression: Similarities and Differences-Fitting the model

meanings of term, Least Squares

multiple linear regression, Multiple Linear Regression-Further Reading

assessing the model, Assessing the Model

cross-validation, Cross-Validation

example, estimating value of houses, Example: King County Housing Data-Example: King County Housing Data

model selection and stepwise regression, Model Selection and Stepwise Regression-Model Selection and Stepwise Regression

weighted regression, Weighted Regression-Further Reading

polynomial and spline regression, Polynomial and Spline Regression-Further Reading

generalized additive models, Generalized Additive Models-Generalized Additive Models

polynomial, Polynomial

splines, Splines

prediction with, Prediction Using Regression-Factor Variables in Regression

confidence and prediction intervals, Confidence and Prediction Intervals-Factor Variables in Regression

dangers of extrapolation, The Dangers of Extrapolation

simple linear regression, Simple Linear Regression-Further Reading

fitted values and residuals, Fitted Values and Residuals

least squares, Least Squares

prediction versus explanation, Prediction Versus Explanation (Profiling)

regression equation, The Regression Equation-Fitted Values and Residuals

with a tree, Predicting a Continuous Value

unsupervised learning as building block, Unsupervised Learning

regression coefficients, Simple Linear Regression, The Regression Equation

comparison with full data and with influential data removed, Influential Values

confidence intervals and, Confidence and Prediction Intervals

correlated predictors and, Correlated Predictors

regression to the mean, Regression to the Mean

regularization, Boosting

avoiding overfitting with, Regularization: Avoiding Overfitting-Regularization: Avoiding Overfitting

L1 regularization, Regularization: Avoiding Overfitting, Regularization: Avoiding Overfitting

L2 regularization, Regularization: Avoiding Overfitting, Regularization: Avoiding Overfitting

reinforcement learning, Reinforcement Learning

ReLU, Activation Functions in the Hidden Layers

replacement (in sampling), Random Sampling and Sample Bias, Resampling

in bootstrap permutation tests, Exhaustive and Bootstrap Permutation Tests

sample with replacement, The Bootstrap

representativeness, Random Sampling and Sample Bias

through random sampling, Random Selection

resampling, The Bootstrap, Resampling-Further Reading

bootstrap and permutation tests, Resampling

bootstrapping versus, Resampling Versus Bootstrapping

permutation tests, Permutation Test-Permutation Tests: The Bottom Line for Data Science, ANOVA

exhaustive and bootstrap tests, Exhaustive and Bootstrap Permutation Tests

value for data science, Permutation Tests: The Bottom Line for Data Science

web stickiness example, Example: Web Stickiness-Exhaustive and Bootstrap Permutation Tests

using in chi-square test, Chi-Square Test: A Resampling Approach, Chi-Square Test: A Resampling Approach

rescaling variables, methods other than z-scores, Standardization (Normalization, z-Scores)

residual standard error (RSE), Multiple Linear Regression, Assessing the Model

residual sum of squares (RSS), Least Squares, Regularization: Avoiding Overfitting

residuals, Simple Linear Regression

analysis of, in logistic regression, Analysis of residuals

in multiple linear regression, Multiple Linear Regression

in simple linear regression, Fitted Values and Residuals

response, Data Frames and Indexes, Simple Linear Regression, The Regression Equation

isolating relationship between predictor variable and, Partial Residual Plots and Nonlinearity

ridge regression, Model Selection and Stepwise Regression, Regularization: Avoiding Overfitting

RMSE (see root mean squared error)

RNN, Advantages of Depth

robust, Estimates of Location

robust estimates of correlation, Correlation

robust estimates of location, Median and Robust Estimates-Example: Location Estimates of Population and Murder Rates

example, location estimates of population and murder rates, Example: Location Estimates of Population and Murder Rates

median, Median and Robust Estimates

outliers and, Outliers

other robust metrics for, Outliers

weighted median, Median and Robust Estimates

robust estimates of variability, median absolute deviation from the median, Standard Deviation and Related Estimates

robust estimates of variance, calculating robust MAD, Example: Variability Estimates of State Population

ROC curve, Evaluating Classification Models, ROC Curve-ROC Curve

AUC metric, AUC-AUC

root mean squared error (RMSE), Multiple Linear Regression, Assessing the Model, Predicting a Continuous Value

RSE (see residual standard error)

RSS (see residual sum of squares)

S

sample bias, Random Sampling and Sample Bias

sample statistic, Sampling Distribution of a Statistic

samples

- sample size, power and, Power and Sample Size-Summary
- terminology differences, Data Frames and Indexes

sampling, Data and Sampling Distributions-Summary

- binomial distribution, Binomial Distribution-Further Reading
- bootstrap, The Bootstrap-Further Reading
- chi-square distribution, Chi-Square Distribution
- confidence intervals, Confidence Intervals-Further Reading
- F-distribution, F-Distribution
- long-tailed distributions, Long-Tailed Distributions-Student's t-Distribution
- normal distribution, Normal Distribution-Standard Normal and QQ-Plots
- Poisson and related distributions, Poisson and Related Distributions
  - estimating the failure rate, Estimating the Failure Rate
  - exponential distribution, Exponential Distribution
  - Poisson distributions, Poisson Distributions

Weibull distribution, Weibull Distribution

random sampling and sample bias, Random Sampling and Sample Bias-Selection Bias

with and without replacement, Random Sampling and Sample Bias, The Bootstrap, Resampling, Exhaustive and Bootstrap Permutation Tests

selection bias, Selection Bias-Further Reading

Student's t-distribution, Student’s t-Distribution-Further Reading

sampling distribution, Sampling Distribution of a Statistic-Further Reading

central limit theorem, Central Limit Theorem

data distribution versus, Sampling Distribution of a Statistic

standard error, Standard Error

sampling variability, Sampling Distribution of a Statistic

scaling, Scaling and Categorical Variables

scaling and categorical variables, Scaling and Categorical Variables-Summary

categorical variables and Gower's distance, Categorical Data and Gower’s Distance

dominant variables, Dominant Variables-Categorical Data and Gower’s Distance

problems with clustering mixed data, Problems with Clustering Mixed Data

scaling the variables, Scaling the Variables-Scaling the Variables

scatterplot smoothers, Heteroskedasticity, Nonnormality, and Correlated Errors

scatterplots, Correlation, Scatterplots

biplot, Correspondence Analysis

extending with conditional variables, Visualizing Multiple Variables

scientific fraud, detecting, Fisher's Exact Test

scoring metric, Model Selection and Stepwise Regression

scree plots, Principal Components Analysis, Dominant Variables

screeplots

for PCA of top stocks, Interpreting Principal Components

search

need for enormous quantities of data, Size Versus Quality: When Does Size Matter?

selection bias, Selection Bias-Further Reading

regression to the mean, Regression to the Mean-Regression to the Mean

typical forms of, Selection Bias

self-selection sampling bias, Random Sampling and Sample Bias

sensitivity, Evaluating Classification Models, Precision, Recall, and Specificity

short-term memory, Memory Issues

sigmoid function, Logistic Response Function and Logit, Activation Functions in the Hidden Layers

signal-to-noise ratio (SNR), Choosing K

significance level, Power and Sample Size, Sample Size

significance tests, Hypothesis Tests

(see also hypothesis tests)

underestimating and misinterpreting random events in, Hypothesis Tests

simple random sample, Random Sampling and Sample Bias

single linkage metric, Measures of Dissimilarity

skew, Long-Tailed Distributions

skewness, Frequency Tables and Histograms

slope, The Regression Equation

(see also regression coefficients)

slot machines used in gambling, Multi-Arm Bandit Algorithm

smoothing parameter, use with naive Bayes algorithm, Numeric Predictor Variables

SMOTE algorithm, Data Generation

Sobel operator, Intuition Behind CNNs

softmax, The Final Layer

spatial data structures, Nonrectangular Data Structures

Spearman's rho, Correlation

specificity, Evaluating Classification Models, Precision, Recall, and Specificity

trade-off with recall, ROC Curve

spline regression, Polynomial and Spline Regression, Splines

split value, Tree Models, How Trees Are Used

SQL (Structured Query Language), Further Reading

square-root of n rule, Standard Error

SS (see sum of squares)

standard deviation, Estimates of Variability

covariance matrix and, Covariance Matrix

in A/B testing, A/B Testing

and related estimates, Standard Deviation and Related Estimates

standard error, Sampling Distribution of a Statistic, Standard Error

standard normal distribution, Normal Distribution

and QQ-Plots, Standard Normal and QQ-Plots

standardization, Normal Distribution, Standard Normal and QQ-Plots, K-Nearest Neighbors, Distance Metrics

in K-nearest neighbors, Standardization (Normalization, z-Scores)-Standardization (Normalization, z-Scores)

of continuous variables, K-Means Clustering

standardized residuals

defined, Regression Diagnostics

examining to detect outliers, Outliers

histogram of, for housing data regression, Heteroskedasticity, Nonnormality, and Correlated Errors

statistical experiments and significance testing, Statistical Experiments and Significance Testing-Summary

A/B testing, A/B Testing-Further Reading

analysis of variance (ANOVA), ANOVA-Further Reading

chi-square test, Chi-Square Test-Further Reading

degrees of freedom, Degrees of Freedom-Further Reading

hypothesis tests, Hypothesis Tests-Further Reading

multi-arm bandits, Multi-Arm Bandit Algorithm-Further Reading

multiple testing, Multiple Testing-Further Reading

power and sample size, Power and Sample Size-Summary

resampling, Resampling-Further Reading

statistical significance and p-values, Statistical Significance and p-Values-Further Reading

alpha, Alpha

controversy over p-values, p-value controversy

data science and p-values, Data Science and p-Values

p-values, p-Value

type 1 and type 2 errors, Type 1 and Type 2 Errors

t-tests, t-Tests-Further Reading

statistical inference, classical inference pipeline, Statistical Experiments and Significance Testing

statistical machine learning, Statistical Machine Learning-Summary

bagging and the random forest, Bagging and the Random Forest-Hyperparameters

bagging, Bagging

hyperparameters, Hyperparameters

random forests, Random Forest-Random Forest

variable importance, Variable Importance-Variable Importance

boosting, Boosting-Summary

boosting algorithm, The Boosting Algorithm

hyperparameters and cross-validation, Hyperparameters and Cross-Validation-Hyperparameters and Cross-Validation

overfitting, avoiding with regularization, Regularization: Avoiding Overfitting-Regularization: Avoiding Overfitting

XGBoost software, XGBoost-Regularization: Avoiding Overfitting

K-nearest neighbors, K-Nearest Neighbors-KNN as a Feature Engine

distance metrics, Distance Metrics

example, predicting loan default, A Small Example: Predicting Loan Default-A Small Example: Predicting Loan Default

KNN as feature engine, KNN as a Feature Engine-KNN as a Feature Engine

one-hot encoder, One-Hot Encoder

standardization, Standardization (Normalization, z-Scores)-Standardization (Normalization, z-Scores)

tree models, Tree Models-Further Reading

how trees are used, How Trees Are Used

measuring homogeneity or impurity, Measuring Homogeneity or Impurity

predicting a continuous value, Predicting a Continuous Value

recursive partitioning, The Recursive Partitioning Algorithm-The Recursive Partitioning Algorithm

simple example, A Simple Example

stopping tree growth, Stopping the Tree from Growing

unsupervised learning, Unsupervised Learning

(see also unsupervised learning)

statistical significance, Permutation Test

practical significance versus, Practical significance

statistics versus machine learning, Statistical Machine Learning

stepwise regression, Model Selection and Stepwise Regression

stochastic gradient boosting, Boosting

XGBoost implementation, XGBoost-Regularization: Avoiding Overfitting

stratified sampling, Random Selection

structured data, Elements of Structured Data-Data Structures

Student's t-distribution, Student’s t-Distribution-Further Reading, t-Tests

subjects, A/B Testing

success, Binomial Distribution

sum contrasts, Dummy Variables Representation

sum of squares (SS), ANOVA, F-Statistic

within-cluster SS, A Simple Example

supervised learning, Regression and Prediction, Classification, Statistical Machine Learning

Synthetic Minority Oversampling Technique (see SMOTE algorithm)

T

t-distribution, Student’s t-Distribution-Further Reading, t-Tests, Multiple Testing

t-statistic, t-Tests, Multiple Linear Regression, Assessing the Model

t-tests, t-Tests-Further Reading

tails, Long-Tailed Distributions

    summarizing with percentiles, Percentiles and Boxplots

tanh, Activation Functions in the Hidden Layers

target, Data Frames and Indexes

target shuffling, Selection Bias

test, Hyperparameter Optimization

test statistic, A/B Testing, Why Have a Control Group?, t-Tests

Thompson's sampling, Multi-Arm Bandit Algorithm

tokenization, Text Data

tokens, Text Data

transfer learning, Image Data and Transfer Learning

transformers, Memory Issues

treatment, A/B Testing

treatment group, A/B Testing

tree models, Interactions and Main Effects, Exploring the Predictions, Tree Models-Further Reading

    advantages of, Tree Models

ensemble, random forest and boosted trees, Bagging and the Random Forest

how trees are used, How Trees Are Used

measuring homogeneity or impurity, Measuring Homogeneity or Impurity

predicting a continuous value, Predicting a Continuous Value

recursive partitioning algorithm, The Recursive Partitioning Algorithm-The Recursive Partitioning Algorithm

simple example, A Simple Example

stopping tree growth, Stopping the Tree from Growing

Trellis graphics, Visualizing Multiple Variables

trials, Binomial Distribution

trimmed mean, Estimates of Location

formula for, Mean

Tukey's HSD (honest significance difference), Multiple Testing

Tukey, John Wilder, Exploratory Data Analysis

two-way ANOVA, Two-Way ANOVA

two-way tests, Hypothesis Tests, One-Way Versus Two-Way Hypothesis Tests

type 1 errors, Statistical Significance and p-Values, Type 1 and Type 2 Errors, Multiple Testing

type 2 errors, Statistical Significance and p-Values, Type 1 and Type 2 Errors

U

unbiased estimates, Standard Deviation and Related Estimates

undersampling, Undersampling

uniform random distribution, Fisher's Exact Test

univariate analysis, Exploring Two or More Variables

unsupervised learning, Unsupervised Learning-Summary

- goals achieved by, Unsupervised Learning
- hierarchical clustering, Hierarchical Clustering-Measures of Dissimilarity
  - agglomerative algorithm, The Agglomerative Algorithm
  - dissimilarity metrics, Measures of Dissimilarity-Measures of Dissimilarity
  - representation in a dendrogram, The Dendrogram-The Dendrogram
  - simple example, A Simple Example
- K-means clustering, K-Means Clustering-Selecting the Number of Clusters
  - interpreting the clusters, Interpreting the Clusters-Analyzing the Clusters with AI
  - K-means algorithm, K-Means Algorithm
  - selecting the number of clusters, Selecting the Number of Clusters
  - simple example, A Simple Example
- model-based clustering, Model-Based Clustering-Further Reading
  - mixtures of normals, Mixtures of Normals-Mixtures of Normals

multivariate normal distribution, Multivariate Normal Distribution

selecting the number of clusters, Selecting the Number of Clusters-Selecting the Number of Clusters

and prediction, Unsupervised Learning

principal components analysis, Principal Components Analysis-Further Reading

computing principal components, Computing the Principal Components

correspondence analysis, Correspondence Analysis-Analyzing the Results of Principal Components and Correspondence Analysis with AI

interpreting principal components, Interpreting Principal Components-Interpreting Principal Components

simple example, A Simple Example-A Simple Example

scaling and categorical variables, Scaling and Categorical Variables-Summary

categorical variables and Gower's distance, Categorical Data and Gower’s Distance

dominant variables, Dominant Variables-Categorical Data and Gower’s Distance

problems with clustering mixed data, Problems with Clustering Mixed Data

scaling the variables, Scaling the Variables-Scaling the Variables

up/down weighting, Strategies for Imbalanced Data, Oversampling and Up/Down Weighting

uplift, Lift

(see also lift)

V

vanishing gradient problem, Memory Issues

variability, estimates of, Estimates of Variability-Exploring the Data Distribution

example, murder rates by state population, Example: Variability Estimates of State Population

key terminology, Estimates of Variability

percentiles, Estimates Based on Percentiles

standard deviation and related estimates, Standard Deviation and Related Estimates-Standard Deviation and Related Estimates

variable importance, Bagging and the Random Forest, Variable Importance-Variable Importance

variables

covariance between, Covariance Matrix

exploring two or more, Exploring Two or More Variables-Summary

categorical and numeric data, Categorical and Numeric Data

categorical variables, Two Categorical Variables

key terms, Exploring Two or More Variables

using hexagonal binning and contour plot, Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)-Hexagonal Binning and Contours (Plotting Numeric-Versus-Numeric Data)

visualizing multiple variables, Visualizing Multiple Variables

rescaling, methods other than z-scores, Standardization (Normalization, z-Scores)

variance, Estimates of Variability, Standard Deviation and Related Estimates

analysis of (ANOVA), F-Distribution, ANOVA-Further Reading

bias-variance trade-off in fitting the model, Choosing K

vast search effect, Selection Bias

violin plots, Exploring Two or More Variables

boxplots versus, Categorical and Numeric Data

visualizations, Nonrectangular Data Structures

(see also graphs)

W

W3Schools guide for SQL, Further Reading

web testing

A/B testing in data science, Why Have a Control Group?

click rate for three different headlines, Chi-Square Test: A Resampling Approach

popularity of bandit algorithms in, Multi-Arm Bandit Algorithm

web stickiness example, Example: Web Stickiness-Exhaustive and Bootstrap Permutation Tests, ANOVA

Weibull distribution, Weibull Distribution

weighted mean, Estimates of Location

formula for, Mean

weighted median, Estimates of Location, Median and Robust Estimates

weighted regression, Multiple Linear Regression, Weighted Regression-Further Reading

weighting

up weight and down weight, Oversampling and Up/Down Weighting

using to change loss function in classification, Oversampling and Up/Down Weighting

weights, Weights and Biases

weights for principal components (see loadings)

weights in a neural network, Activation Functions in the Hidden Layers

whiskers (in boxplots), Percentiles and Boxplots

wins, Multi-Arm Bandit Algorithm

within-cluster sum of squares (SS), K-Means Clustering

Word2vec, Embeddings

world models, World Models

X

XGBoost, XGBoost-Regularization: Avoiding Overfitting

hyperparameters, Hyperparameters and Cross-Validation

using regularization to avoid overfitting, Regularization: Avoiding Overfitting-Regularization: Avoiding Overfitting

Z

z-scores, Normal Distribution, Standard Normal and QQ-Plots, Covariance Matrix, Strategies for Imbalanced Data, K-Nearest Neighbors

conversion of data to, normal distribution and, Standard Normal and QQ-Plots

in data standardization for KNN, Standardization (Normalization, z-Scores)

using to scale variables, Scaling the Variables

zero-shot learning, Prompt engineering in production use

About the Authors

Peter Bruce founded and grew the Institute for Statistics Education at Statistics.com, which now offers about one hundred courses in statistics, roughly a third of which are aimed at the data scientist. In recruiting top authors as instructors and forging a marketing strategy to reach professional data scientists, Peter has developed both a broad view of the target market and his own expertise to reach it.

Andrew Bruce has over 30 years of experience in statistics and data science in academia, government, and business. He has a PhD in statistics from the University of Washington and has published numerous papers in refereed journals. He has developed statistical-based solutions to a wide range of problems faced by a variety of industries, from established financial firms to internet startups, and offers a deep understanding of the practice of data science.

Peter Gedeck has over 30 years of experience in scientific computing and data science. After 20 years as a computational chemist at Novartis, he now works as a senior data scientist at Collaborative Drug Discovery. He specializes in the development of machine learning algorithms to predict biological and physicochemical properties of drug candidates. Coauthor of *Machine Learning for Business Analytics*, he earned a PhD in chemistry from the University of Erlangen-Nürnberg in Germany and studied mathematics at the Fernuniversität Hagen, Germany.

Colophon

The animal on the cover of *AI-Assisted Statistics for Data Scientists* is a lined shore crab (*Pachygrapsus crassipes*), also known as a striped shore crab. It is found along the coasts and beaches of the Pacific Ocean in North America, Central America, Korea, and Japan. These crustaceans live under rocks, in tidepools, and within crevices. They spend about half their time on land, and periodically return to the water to wet their gills.

The lined shore crab is named for the green stripes on its brown-black carapace. It has red claws and purple legs, which also have a striped or mottled pattern. The crab generally grows to be 3–5 centimeters in size; females are slightly smaller. Their eyes are on flexible stalks that can rotate to give them a full field of vision as they walk.

Crabs are omnivores, feeding primarily on algae but also on mollusks, worms, fungi, dead animals, and other crustaceans (depending on what is available). They molt many times as they grow to adulthood, taking in water to expand and crack open their old shell. Once this is achieved, they spend several difficult hours getting free, and then must hide until the new shell hardens.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on a black-and-white engraving from *Pictorial Museum of Animated Nature*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.