

EXPERT INSIGHT

In color

Learning Tableau 2025

**Leverage Tableau's newest features
to revolutionize your data storytelling
with AI-enhanced insights**

Sixth Edition



Joshua N. Milligan

<packt>

Learning Tableau 2025

Sixth Edition

Leverage Tableau's newest features to revolutionize your data storytelling with AI-enhanced insights

Joshua N. Milligan



Learning Tableau 2025

Sixth Edition

Copyright © 2025 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author nor Packt Publishing and its dealers and distributors will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Portfolio Director: Sunith Shetty

Relationship Lead: Apeksha Shetty

Project Manager: Shashank Desai

Content Engineers: Shrishti Pandey, Ayushi Bulani, David Sugarman

Technical Editor: Gaurav Gavas

Copy Editor: Safis Editing

Indexer: Tejal Soni

Proofreaders: Shrishti Pandey, Ayushi Bulani, David Sugarman

Production Designer: Aparna Bhagat

Growth Lead: Bhavesh Amin

First published: April 2015

Second edition: September 2016

Third edition: March 2019

Fourth edition: August 2020

Fifth edition: August 2022

Sixth edition: July 2025

Production reference: 1180725

Published by Packt Publishing Ltd.

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB, UK.

ISBN 978-1-83588-678-6

www.packtpub.com

To my dad and mom, Stuart and LaNita, who encouraged me to pursue my interests and allowed me to spend hours on the family computer. To my children, Caleb, Rachel, Daniel, and Grace, who are currently pursuing their varied interests and keeping me on my toes. To my wife, Kara, who is my best friend and partner in life, as she continues to love and support me in everything.

– Joshua Milligan

Contributors

About the author

Joshua N. Milligan is a Tableau Hall of Fame Visionary (formerly known as Zen Master). He is an Iron Viz finalist and author of every edition of Learning Tableau. Bringing a creative approach, he loves to share in-depth knowledge with the Tableau Community. His passion is training, mentoring, and helping people gain insights and make decisions through visual analytics using Tableau. He is a principal consultant at Resultant, where he has served clients in numerous industries since 2004.

I want to thank my colleagues at Resultant, the Tableau Community and DataFam, all the staff at Packt, along with the reviewers. I especially want to thank my wife, who has lovingly supported me in everything.

About the reviewers

Nick Katko is a seasoned data and analytics professional with over 18 years of experience in architecting, developing, and delivering actionable insights through visually compelling dashboards. After discovering the original Polaris paper (the Stanford technical report that became Tableau's foundation) in 2007, he hitched himself to the Tableau rocket ship and has been on a wild ride ever since. Earning multiple Tableau and business systems certifications throughout his career, Nick has partnered with various consultancies and many Fortune 500 organizations to modernize analytics platforms, lead enablement initiatives, and solve high-stakes data challenges. And yes, with his naked eye, he can tell if your viz is 1 pixel out of alignment.

Zach Bowders is an acclaimed data analyst, recognized as a five-term Tableau Visionary and Ambassador. With over 275 public visualizations, he has earned 12 Viz of the Day honors and two Vizzie Community Awards. A regular Tableau User Group speaker, he has presented at the Tableau Conference and served as a Tableau instructor for Emory University's "Women in Technology" non-profit program. He shares insights bi-weekly on the Data + Love podcast, exploring data visualization, analytics, and storytelling.

Prasann Prem is a Tableau Visionary, Social Ambassador, and senior BI engineer at Emirates NBD. A global speaker and mentor, he's known for simplifying data through impactful storytelling. He founded Tableau Buddy, a platform for guiding beginners via 1:1 mentorship, speaker forums, and a 4-month sandbox program. He also runs Vizdiner, a Tableau learning hub offering free and structured content. With 18k+ LinkedIn followers, Prasann empowers the global community with insights, tools, and mentorship to grow in data visualization.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



Table of Contents

Preface	xvii
Your Book Comes with Exclusive Perks - Here's How to Unlock Them	xxii
Chapter 1: Taking Off with Tableau	1
Connecting to data	2
Foundations for building visualizations	6
Measures and dimensions • 7	
Discrete and continuous fields • 8	
<i>Discrete fields • 8</i>	
<i>Continuous fields • 10</i>	
Visualizing data	11
Bar charts • 12	
<i>Iterations of bar charts for deeper analysis • 14</i>	
Line charts • 16	
<i>Iterations of line charts for deeper analysis • 19</i>	
Geographic visualizations • 21	
<i>Filled maps • 22</i>	
<i>Symbol maps • 24</i>	
<i>Density maps • 26</i>	
Using Show Me	28
Putting everything together in a dashboard	30
The dashboard interface • 31	
Building your dashboard • 33	
Summary	36

Chapter 2: Connecting to Data and Tableau	37
The Tableau paradigm	37
The Tableau paradigm in action • 38	
Connecting to data	42
Connecting to data in a file • 43	
Connecting to data in a database • 46	
Connecting to other data in the cloud • 48	
Shortcuts for connecting to data • 50	
Managing data source metadata	51
Working with extracts instead of live connections	54
Creating extracts • 55	
Using extracts • 57	
Performance • 59	
Portability and security • 60	
When to use an extract • 60	
Filtering data	61
Filtering discrete (blue) fields • 63	
Filtering continuous (green) fields • 65	
Filtering dates • 67	
Other filtering options • 68	
Tableau's order of operations • 69	
Summary	70
Chapter 3: Moving Beyond Basic Visualizations	73
Comparing values	74
Bar charts • 74	
Bar chart variations • 76	
<i>Bullet chart</i> • 77	
<i>Bar-in-bar chart</i> • 81	
<i>Highlighting categories of interest</i> • 85	
Visualizing dates and times	87
Date parts, date values, and exact dates • 87	
Variations of date and time visualizations • 91	
Gantt charts • 92	
Relating parts of the data to the whole	95
Stacked bars • 95	

Treemaps • 99	
Area charts • 101	
Pie charts • 103	
Visualizing distributions	104
Circle charts • 105	
<i>Jittering</i> • 106	
Box and whisker plots • 108	
Histograms • 109	
Visualizing multiple axes to compare different measures	114
Scatterplot • 114	
Dual-axis and combination charts • 115	
Going further with Viz Extensions	118
Setting up Viz Extensions • 118	
Visualizing flow with the Sankey Viz Extension • 119	
Summary	122
 Chapter 4: Starting an Adventure with Calculations	 123
Introduction to calculations	124
The four main types of calculations • 124	
Creating and editing calculations • 124	
Data types • 127	
Additional functions and operators • 128	
Understanding the data used in chapter examples • 129	
Row-level calculations	129
Concatenating strings • 129	
String manipulation and conditional logic • 130	
Planning for data variations • 133	
Aggregate calculations	134
Why the row-level versus aggregate difference matters • 137	
Parameters	139
Creating parameters • 139	
Practical examples of calculations and parameters	142
Fixing data issues • 142	
Extending the data • 143	
Enhancing user experience, analysis, and visualizations • 145	
Meeting business requirements • 147	
Ad hoc calculations • 148	

Performance considerations	149
Summary	151
Chapter 5: Leveraging Level of Detail Calculations	153
Overview of level of detail	153
Level of detail calculations	155
Level of detail syntax • 155	
Level of detail types • 155	
<i>Fixed</i> • 156	
<i>Include</i> • 156	
<i>Exclude</i> • 156	
An illustration of the difference level of detail can make • 156	
Examples of fixed level of detail calculations	158
Flagging members based on historical values • 159	
Latest balance for a member • 162	
Examples of include level of detail expressions	164
Average loans per member • 164	
<i>Alternative approaches</i> • 166	
Example of exclude level of detail calculations	167
Average credit score per loan type • 167	
Summary	169
Chapter 6: Diving Deep with Table Calculations	171
An overview of table calculations	172
Creating and editing table calculations	173
Quick table calculations	174
Fundamental table calculation concepts	176
Relative versus fixed • 176	
Scope and direction • 177	
Addressing and partitioning • 183	
Custom table calculations	187
Meta table functions • 187	
Lookup and previous value • 188	
Running functions • 191	
Window functions • 192	
Rank functions • 193	

Script functions • 193	
The Total function • 193	
Practical examples	195
Year-over-year growth • 195	
Dynamic titles with totals • 197	
Table calculation filtering (late filtering) • 199	
Summary	201
 Chapter 7: Making Visualizations that Look Great and Work Well	 203
Visualization considerations	203
Leveraging formatting in Tableau	204
Workbook-level formatting • 206	
Worksheet-level formatting • 207	
Field-level formatting • 210	
<i>Custom number formatting • 211</i>	
<i>Custom date formatting • 214</i>	
<i>Null formatting • 216</i>	
Additional formatting options • 220	
Adding value to visualizations	222
Tooltips • 225	
<i>Viz in Tooltip • 227</i>	
Summary	229
 Chapter 8: Telling a Data Story with Dashboards	 231
Introduction to dashboards	232
Dashboard objectives • 232	
Dashboard approaches • 233	
Designing dashboards in Tableau	234
Using objects • 234	
<i>Tiled versus floating • 236</i>	
<i>Manipulating objects on the dashboard • 236</i>	
A dashboard to understand profitability	237
Building the views • 238	
Creating the dashboard framework • 241	
Implementing actions to guide the story • 244	
Context filtering • 247	

Designing for different displays and devices	249
Interactivity with actions	252
Triggers of actions • 252	
Creating actions • 252	
Types of action • 253	
<i>Filter actions • 254</i>	
<i>Highlight actions • 257</i>	
<i>URL actions • 258</i>	
<i>Go to Sheet actions • 260</i>	
<i>Parameter actions • 260</i>	
<i>Set actions • 261</i>	
A scorecard dashboard	265
Stories	269
Summary	276
 Chapter 9: Visual Analytics: Trends, Clustering, Distributions, and Forecasting	 277
Trends	278
Customizing trend lines • 284	
Trend models • 286	
<i>Linear • 287</i>	
<i>Logarithmic • 287</i>	
<i>Exponential • 288</i>	
<i>Power • 288</i>	
<i>Polynomial • 289</i>	
Analyzing trend models • 289	
<i>Exporting statistical model details • 292</i>	
Data Guide	296
Clustering	300
Distributions	304
Forecasting	307
Summary	312
 Chapter 10: Advanced Visualizations	 313
Advanced visualizations—when and why to use them	314
Slope charts and bump charts	314
Waterfall charts	317

Step lines and jump lines	318
Sparklines	319
Dumbbell charts	321
Unit/symbol charts	322
Marimekko charts	325
Sunburst diagram with Viz Extensions	327
Animated visualizations	329
Enhancing analysis with animation • 329	
Enhancing data storytelling with animation • 331	
Summary	333
Chapter 11: Dynamic Dashboards	335
Show/Hide buttons	335
Dynamic Zone Visibility	342
Summary	346
Chapter 12: Exploring Mapping and Advanced Geospatial Features	347
Overview of Tableau maps	347
Rendering maps with Tableau	348
Customizing map layers • 349	
Customizing map options • 351	
Using geospatial data	352
Including latitude and longitude in your data • 352	
Importing definitions into Tableau's geographic database • 354	
Leveraging spatial objects • 354	
Leveraging spatial functions	356
MAKELINE() and MAKEPOINT() • 359	
DISTANCE() • 360	
BUFFER() • 361	
INTERSECTS() • 363	
Creating custom territories	364
Ad hoc custom territories • 364	
Field-defined custom territories • 366	
Map layers	367
Tableau mapping – tips and tricks	371
Plotting data on background images	372
Summary	376

Chapter 13: Understanding the Tableau Data Model, Joins, and Blends	377
Explanation of the sample data used in this chapter	377
Exploring the Tableau data model	379
<i>Creating a data model</i> • 379	
<i>Layers of the data model</i> • 384	
<i>Using the data model</i> • 385	
<i>The Data pane interface</i> • 385	
<i>Data model behaviors</i> • 386	
<i>Multi-fact analysis</i> • 390	
Using joins	391
<i>Types of joins</i> • 391	
<i>Joining tables</i> • 394	
<i>Other join considerations</i> • 397	
<i>Join calculations</i> • 397	
<i>Cross-database joins</i> • 398	
<i>The unintentional duplication of data</i> • 399	
Using blends	401
<i>Using blending to show progress toward a goal</i> • 403	
When to use a data model, joins, or blends	408
Summary	409
Chapter 14: Structuring Messy Data to Work Well in Tableau	411
Structuring data for Tableau	412
Well-structured data in Tableau • 413	
<i>Wide data</i> • 413	
<i>Tall data</i> • 414	
<i>Wide versus tall data in Tableau</i> • 415	
Star schemas • 418	
The four basic data transformations	421
Overview of transformations • 421	
Pivots (along with some simple data cleaning) • 421	
Unions • 427	
Joins • 432	
Aggregation • 436	
Overview of advanced fixes for data problems	442
Summary	443

Chapter 15: Taming Data with Tableau Prep	445
Getting ready to explore Tableau Prep Builder	446
Understanding the Tableau Prep Builder interface	446
Flowing with the fundamental paradigm	449
Connecting to data • 452	
Cleaning the data • 460	
<i>Unioning and merging mismatched fields, and removing unnecessary fields • 460</i>	
<i>Grouping and cleaning • 463</i>	
Calculations and aggregations in Tableau Prep • 469	
<i>Row-level calculations • 469</i>	
<i>Level of detail calculations • 471</i>	
<i>Aggregations • 474</i>	
Filtering data in Tableau Prep • 477	
Transforming the data for analysis • 479	
Using parameters in Tableau Prep Builder	488
Options for automating flows	492
Summary	492
Chapter 16: Extending Tableau: Extensions and Scripts	493
Tableau extensions	494
What is a dashboard extension? • 494	
Using a dashboard extension • 495	
Creating an extension • 502	
<i>Setting up your environment • 503</i>	
<i>The files and code that define the dashboard extension • 504</i>	
<i>The dashboard extension in action • 507</i>	
Tableau analytics extensions	509
Scripting with Python • 510	
<i>Setting up TabPy • 510</i>	
<i>Using a Python script in Tableau • 512</i>	
Table extensions	516
Adding a table extension • 517	
Summary	522

Chapter 17: AI-Assisted Analysis: Tableau Agent and Tableau Pulse	523
Tableau Agent	524
Tableau Pulse	530
Setting up the metric definitions • 530	
<i>Definition options • 533</i>	
<i>Insight options • 538</i>	
<i>Creating and interacting with metrics • 542</i>	
<i>Seeing and sharing metrics • 547</i>	
Summary	549
 Other Books You May Enjoy	 553
Index	557

Preface

When Tableau was first introduced, it was a dramatic paradigm shift away from tabular reports and extensive cycles of data integration that produced results long after relevant decisions could have been made. Tableau disrupted the paradigm for visually interacting with data, and it continues to offer a unique paradigm – one which makes it easy and intuitive (and fun!) to be hands-on with data, to receive instant visual feedback with every action, and to ask questions and uncover insights in a natural flow of thought and interaction. The result is an ever growing #datafam, a community that loves Tableau for its simplicity, beauty, and ability to make working with data fun!

Beyond the core paradigm, Tableau continues to innovate – incorporating new features and new paradigms to make the task of seeing, understanding, and acting upon data even easier and more enjoyable than ever! With data model and multi-fact analysis, viz extensions, AI integration (including Tableau Agent and Tableau Pulse), Tableau continues to push the boundaries of how analysts, business users, and decision makers interact with data.

We'll look at Tableau through the lens of understanding the underlying paradigm of how and why Tableau works, in the context of practical examples. Then we'll build on this solid foundation of understanding so that you will have the tools and skills to tackle even the toughest data challenges!

Who this book is for

This book is for anyone who needs to see and understand their data! From the business user to the hardcore data analyst to the CEO and executive decision makers, everyone needs to have the ability to ask and answer questions of data. Having a bit of background with data will definitely help, but you don't need to be confident with scripting, SQL, or database structures.

Whether you're new to Tableau, or have been using it for months or even years, with this book you'll gain a solid foundation of understanding Tableau, and the tools and skills to build toward advanced mastery of the tool.

What this book covers

Chapter 1, Taking Off with Tableau, introduces the foundational principles of Tableau. We'll walk through multiple examples in a logical progression that will introduce everything from the interface to connecting to data, building your first visualization, and even a dashboard. This chapter will give you a solid foundation for terminology and concepts that will be used throughout the book.

Chapter 2, Connecting to Data and Tableau, covers the foundational concepts involved in connecting to data, using several practical examples. It covers the various types of connections, file types, cloud-based and on-premises databases, and how to work with metadata.

Chapter 3, Moving Beyond Basic Visualizations, builds on the basic visualization principles covered in the first chapter to explore variations and extended possibilities. You will learn when and how to use a wide variety of visualizations to analyze and communicate data.

Chapter 4, Starting an Adventure with Calculations, introduces calculations and parameters, giving an overview of the major types of calculation and then detailed examples of row-level and aggregate calculations. It combines conceptual knowledge with practical examples, and concludes with performance considerations.

Chapter 5, Leveraging Level of Detail Calculations, takes an in-depth look at level of detail expressions and how to use them to solve complex data challenges. It not only gives an overview, but dives into examples of FIXED, INCLUDE, and EXCLUDE.

Chapter 6, Diving Deep with Table Calculations, gives you a strong foundation for understanding and using table calculations to solve a wide range of data challenges. It covers the concepts of scope and direction, addressing and partitioning, and walks through several in-depth practical examples.

Chapter 7, Making Visualizations That Look Great and Work Well, covers how to extend and alter the default formatting applied to visualizations by Tableau, to customize options such as font, color, lines, shading, annotations, and tooltips to effectively communicate a data story.

Chapter 8, Telling a Data Story with Dashboards, builds on concepts that were introduced in the first chapter and expanded on throughout. It walks through several practical examples of various kinds of dashboards to help you gain a solid understanding of what a dashboard is, how to build one and make it interactive, and how to use it to tell a compelling data story.

Chapter 9, Visual Analytics: Trends, Clustering, Distributions, and Forecasting, introduces the visual statistical analytics capabilities built into Tableau and supplies you with practical examples of how and when to leverage these capabilities. This includes adding and modifying trend models, leveraging clustering capabilities, using and modifying forecast models, and visualizing the distribution of data. This chapter also includes new details regarding the new features and interface present for Explain Data. You will not only understand how to employ statistical models but also evaluate their accuracy.

Chapter 10, Advanced Visualizations, builds upon the visualizations and techniques already covered, demonstrating how Tableau can be used to create any kind of visualization. A multitude of examples demonstrates a wide variety of advanced visualizations, from bump charts to Marimekko charts to animated visualizations.

Chapter 11, Dynamic Dashboards, builds your dashboard skills by demonstrating various techniques to show and hide content on a dashboard. The result is a truly dynamic user experience that enhances your ability to communicate data.

Chapter 12, Exploring Mapping and Advanced Geospatial Features, demonstrates everything about maps and geospatial visualization, from map basics to geospatial functions, custom territories, and plotting data on custom background images.

Chapter 13, Understanding the Tableau Data Model, Joins, and Blends, explores the major ways of relating data in Tableau, including joins, blends, and relationships using Tableau's data model. With practical examples and detailed descriptions, you will understand the difference between logical and physical layers and how to leverage relationships, joins, and blends to achieve great analytical results.

Chapter 14, Structuring Messy Data to Work Well in Tableau, acknowledges that real-world data is sometimes a mess, and gives you a foundation for understanding well-structured data and a toolset for dealing with data that isn't structured well in Tableau.

Chapter 15, Taming Data with Tableau Prep, explores the Tableau Prep Builder tool, including the overall paradigm and specific features. You will work through an extended practical example to understand how to leverage Tableau Prep's amazing ability to clean and structure data.

Chapter 16, Extending Tableau: Extensions and Scripts, explores the incredible potential of extending Tableau's functionality using new features and integrations in Tableau. The chapter explores dashboard extensions that give you the ability to seamlessly integrate code, backend systems, and new user interfaces directly into your dashboards. You'll also see examples of leveraging scripts that could be used for anything from data cleansing to advanced predictive models.

Chapter 17, AI Assisted Analysis: Tableau Agent and Tableau Pulse, concludes the book with a look at how to leverage AI in Tableau to assist in everything from analysis to decision making. This chapter covers Tableau Agent, a fully integrated AI assistant that helps with everything from generating calculations and visualizations to answering analytics questions. Additionally, the chapter dives into details on Tableau Pulse which integrates generative AI, predictive modeling, and automation, to give you quick and actionable insights into your most important metrics.

To get the most out of this book

This book does not assume specific database knowledge, but it will definitely help to have some basic familiarity with data itself. We'll cover the foundational principles first, and while it may be tempting to skip the first chapter, please don't! We'll lay a foundation of terminology and explore the paradigm that will be used throughout the remainder of the book.

You'll be able to follow along with many of the examples in the book using Tableau Desktop and Tableau Prep Builder (in *Chapter 15, Taming Data with Tableau Prep*).

Most examples can be completed with almost any recent version of Tableau. New features and interfaces will require version 2025.1 or later. To use Tableau Agent as described in *Chapter 17, AI Assisted Analysis: Tableau Agent and Tableau Pulse*, you will need to speak with a Tableau representative about a Tableau+ license or trial.

You may download and install the most recent versions from Tableau using these links:

Tableau Desktop: <https://www.tableau.com/products/desktop/download>

Tableau Prep Builder: <https://www.tableau.com/products/prep/download>

Please speak to a Tableau representative for specific licensing information. In most cases, you may install a 14-day trial of each product if you do not currently have a license.

Download the example code files

The code bundle for the book is hosted on GitHub at <https://github.com/PacktPublishing/Learning-Tableau-2025-6E>. We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

CodeInText: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. For example; “Connect to Hospital Visits.xlsx and generate an extract.”

A block of code is set as follows:

```
IF LEFT([Room], 1) = "1"
THEN "First Floor"
ELSEIF LEFT([Room], 1) = "2"
THEN "Second Floor"
END
```

Bold: Indicates a new term, an important word, or words that you see on the screen, for example, in menus or dialog boxes, also appear in the text like this. For example: “Select **Table Layout | Advanced** from the **Analysis** menu.”



Warnings or important notes appear like this.



Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email feedback@packtpub.com and mention the book's title in the subject of your message. If you have questions about any aspect of this book, please email us at questions@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you reported this to us. Please visit <http://www.packtpub.com/submit-errata>, click Submit Errata, and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packtpub.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit <http://authors.packtpub.com>.

Share Your Thoughts

Once you've read *Learning Tableau 2025_Sixth edition*, we'd love to hear your thoughts! Please click [here](#) to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Your Book Comes with Exclusive Perks - Here's How to Unlock Them

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search this book by name. Ensure it's the correct edition.

UNLOCK NOW



Note: Keep your purchase invoice ready before you start.



Figure 0.1: Next-Gen Reader, AI Assistant (Beta), and Free PDF access

Enhanced reading experience with our Next-gen Reader:

-  **Multi-device progress sync:** Learn from any device with seamless progress sync.
-  **Highlighting and notetaking:** Turn your reading into lasting knowledge.
-  **Bookmarking:** Revisit your most important learnings anytime.
-  **Dark mode:** Focus with minimal eye strain by switching to dark or sepia mode.

Learn smarter using our AI assistant (Beta):

- ✦ **Summarize it:** Summarize key sections or an entire chapter.
- ✦ **AI code explainers:** In the next-gen Packt Reader, click the **Explain** button above each code block for AI-powered code explanations.

Note: The AI assistant is part of next-gen Packt Reader and is still in beta.

Learn anytime, anywhere:

 Access your content offline with DRM-free PDF and ePub versions—compatible with your favorite e-readers.

Unlock Your Book's Exclusive Benefits

Your copy of this book comes with the following exclusive benefits:

 Next-gen Packt Reader

✦ AI assistant (beta)

 DRM-free PDF/ePub downloads

Use the following guide to unlock them if you haven't already. The process takes just a few minutes and needs to be done only once.

How to unlock these benefits in three easy steps

Step 1

Keep your purchase invoice for this book ready, as you'll need it in *Step 3*. If you received a physical invoice, scan it on your phone and have it ready as either a PDF, JPG, or PNG.

For more help on finding your invoice, visit <https://www.packtpub.com/unlock-benefits/help>.

Note: Did you buy this book directly from Packt? You don't need an invoice. After completing Step 2, you can jump straight to your exclusive content.

Step 2

Scan this QR code or go to [packtpub.com/unlock](https://www.packtpub.com/unlock).



On the page that opens (which will look similar to Figure 0.2 if you're on desktop), search for this book by name. Make sure you select the correct edition.

Discover and unlock your book's exclusive benefits

Bought a Packt book? Your purchase may come with free bonus benefits designed to maximise your learning. Discover and unlock them here

Discover Benefits Sign Up/In Upload Invoice

[Need Help?](#)

- 1. Discover your book's exclusive benefits**
Search by title or ISBN
[CONTINUE TO STEP 2](#)
- 2. Login or sign up for free**
- 3. Upload your invoice and unlock**

Figure 0.2: Packt unlock landing page on desktop

Step 3

Sign in to your Packt account or create a new one for free. Once you're logged in, upload your invoice. It can be in PDF, PNG, or JPG format and must be no larger than 10 MB. Follow the rest of the instructions on the screen to complete the process.

Need help?

If you get stuck and need help, visit <https://www.packtpub.com/unlock-benefits/help> for a detailed FAQ on how to find your invoices and more. The following QR code will take you to the help page directly:



Note: If you are still facing issues, reach out to customer@packt.com.

1

Taking Off with Tableau

Tableau is an amazing platform for seeing, understanding, and making key decisions based on your data! With it, you can carry out incredible data discovery, analysis, and storytelling. You'll accomplish these tasks and goals visually using an interface that is designed for the natural and seamless flow of thought and work.

You don't need to write complex scripts or queries to leverage the power of Tableau. Instead, you interact with your data in a visual environment where everything that you drag and drop is translated into the necessary queries for you and then displayed visually. In the latest versions of Tableau, you can even leverage AI to guide your analysis and automate some tasks. You'll be working in real time, so you will see results immediately, get answers as quickly as you can ask questions, and be able to iterate through potentially dozens of ways to visualize the data in order to find a key insight or tell a piece of the story.

This chapter introduces the foundational principles of Tableau. We'll walk through a series of examples that will introduce you to the basics of connecting to data, exploring and analyzing data visually, and finally, putting it all together in a fully interactive dashboard. These concepts will be developed far more extensively in subsequent chapters. However, *don't skip this chapter*, as it introduces key terminology and concepts, including the following:

- Connecting to data
- Foundations for building visualizations
- Visualizing data
- Using Show Me
- Putting everything together in a dashboard

Let's begin by looking at how you can connect Tableau to your data.

Connecting to data

Tableau connects to data stored in a wide variety of files, databases, and repositories. This includes **flat files** such as text files, Excel documents, and spatial files; **databases** such as SQL Server and Oracle; and **cloud-based data sources** such as Snowflake and Microsoft Azure. With very few exceptions, the process of analysis and creating visualizations will be the same, no matter what data source you use.

We'll examine data connections and related topics more extensively throughout the book. For example, we'll cover the following:

- Connecting to a wide variety of different types of data sources in *Chapter 2, Connecting to Data and Tableau*
- Using joins, blends, and object model connections in *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*
- Understanding the data structures that work well and how to deal with messy data in *Chapter 14, Structuring Messy Data to Work Well in Tableau*
- Leveraging the power and flexibility of Tableau Prep to cleanse and shape data for deeper analysis in *Chapter 15, Taming Data with Tableau Prep*

In this chapter, we'll connect to a text file derived from one of the sample datasets that ships with Tableau: `Superstore.csv`. **Superstore** is a fictional retail chain that sells various products to customers across the United States and the file contains a record for every line item of every order with details on the customer, location, item, sales amount, and revenue.



Please use the supplied `Superstore.csv` data file instead of the Tableau Superstore sample data, as there are differences that will change the results. Instructions for downloading all the samples can be found in the *Preface*.

The Chapter 1 workbooks, included with the code files bundle, already have connections to the file; however, for this example, we'll walk through the steps of creating a connection in a new workbook:

1. Open Tableau. You should see the home screen with a list of connection options on the left and, if applicable, thumbnail previews of recently edited workbooks in the center, along with sample workbooks at the bottom.
2. Under **Connect** and **To a File**, click on **Text File**.
3. In the **Open** dialog box, navigate to the `\Learning Tableau\Chapter 01` directory and select the `Superstore.csv` file.

You will now see the data connection screen, which allows you to visually create connections to data sources. We'll examine the features of this screen in detail in the *Connecting to data* section of *Chapter 2, Connecting to Data and Tableau*. For now, Tableau shows a list of fields and a preview of the file's data for the connection:

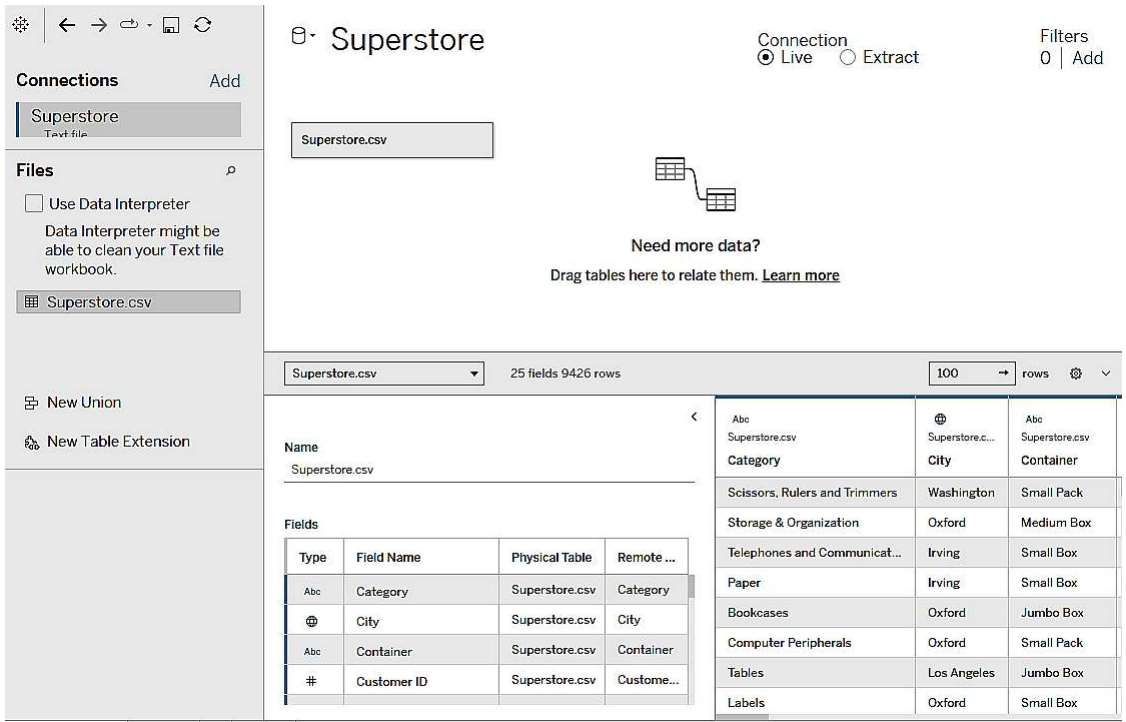


Figure 1.1: The data connection screen allows you to build a connection to your data

For this connection, no other configuration is required, so simply click on the **Sheet 1** tab at the bottom of the Tableau window to start visualizing the data! You should now see the main work area within Tableau, which looks like this:

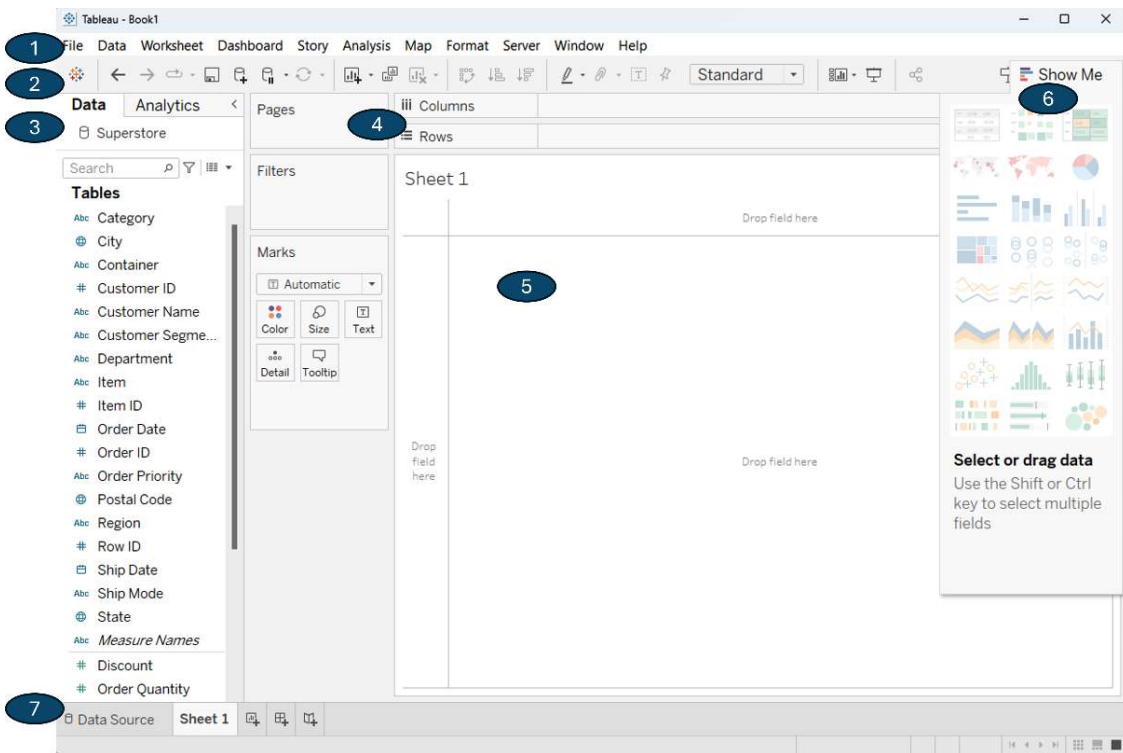


Figure 1.2: Elements of Tableau's primary interface, numbered, with descriptions to follow

We'll refer to the elements of the interface throughout the book using specific terminology, so take a moment to familiarize yourself with the terms used for the various components numbered in the preceding screenshot:

1. The **menu** contains various menu items for performing a wide range of functions.
2. The **toolbar** allows common functions such as undo, redo, save, add a data source, and so on.
3. The **Data** pane is active when the **Data** tab is selected and lists all tables and fields of the selected data source. The **Analytics** pane is active when the **Analytics** tab is selected and gives options for supplementing visualizations with visual analytics.
4. Various shelves, such as **Pages**, **Columns**, **Rows**, and **Filters**, serve as areas to drag and drop fields from the Data pane. The **Marks** card contains additional shelves, such as **Color**, **Size**, **Text**, **Detail**, and **Tooltip**. Tableau will visualize data based on the fields you drop onto the shelves.



Data fields in the **Data** pane are available to add to a view. Fields that have been dropped onto a shelf are called *in the view* or *active fields* because they play an active role in the way Tableau draws the visualization.

You may sometimes hear fields referred to as **pills** because of their appearance in the user interface.

5. The **canvas** or **view** is where Tableau will draw the data visualization. In addition to dropping fields on shelves, you may also drop fields directly onto the view. A **title** is located at the top of the canvas. By default, the title displays the name of the sheet. It can be edited or hidden.
6. **Show Me** is a feature that allows you to quickly iterate through various types of visualizations based on data fields of interest. We'll look at **Show Me** in the *Using Show Me* section.
7. The tabs at the bottom of the window give you options for editing the data source, as well as navigating between and adding any number of sheets, dashboards, or stories. Often, any tab (whether it is a sheet, a dashboard, or a story) is referred to generically as a **sheet** or **worksheet**.

A Tableau workbook is a collection of data sources, sheets, dashboards, and stories. All of this is saved as a single Tableau workbook file (.twb or .twbx). A workbook is organized into a collection of tabs, which can be seen on the bottom toolbar in Tableau Desktop:

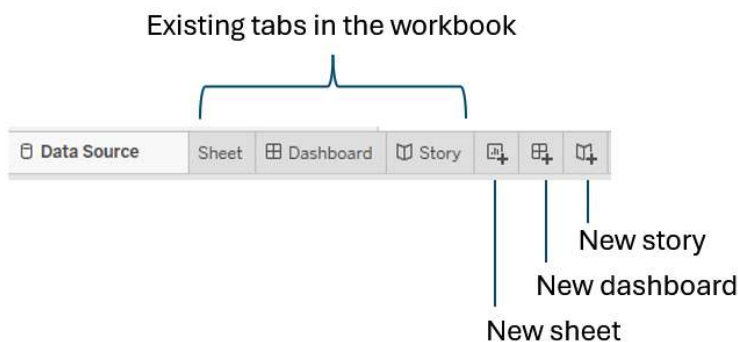


Figure 1.3: Tabs in a Tableau workbook

- A **sheet** is a single data visualization, such as a bar chart or a line graph. Since sheet is also a generic term for any tab, we'll often refer to a sheet as a **view** because it is a single view of the data.
- A **dashboard** is a presentation of any number of related views and other elements (such as text or images) arranged together as a cohesive whole to communicate a message to an audience. Dashboards are often designed to be interactive.
- A **story** is a collection of dashboards or single views that have been arranged to communicate a narrative from the data. Stories may also be interactive.

Along the bottom of the screen, you'll notice a few other items. As you work, a **status bar** at the bottom left will display important information and details about the view, selections, and the user. The controls on the bottom right allow you to navigate between sheets, dashboards, and stories, as well as viewing the tabs with **Show Filmstrip** or switching to a sheet sorter showing an interactive thumbnail of all sheets in the workbook.

Now that you have connected to the data in the text file, we'll explore some examples that lay the foundation for data visualization, and then move on to building some foundational visualization types. To prepare for this, please do the following:

1. From the menu, select **File | Exit**.
2. When prompted to save the changes, select **No**.
3. From the \learning Tableau\Chapter 01 directory, open the file **Chapter 01 Starter.twbx**. This file contains a connection to the Superstore data file and is designed to help you walk through the examples in this chapter.



The files for each chapter include a **Starter** workbook that allows you to work through the examples given in this book. If, at any time, you'd like to see the completed examples, open the **Complete** workbook for the chapter.

With the existing connection in the starter workbook, you are ready to start visualizing and analyzing it. As you begin to do so, you will take on the role of an analyst at the retail chain. You'll ask questions about the data, build visualizations to answer those questions, and ultimately design a dashboard to share the results. Let's start by laying some foundations for understanding how Tableau visualizes data.

Foundations for building visualizations

When you first connect to a data source such as the Superstore file, Tableau will display the data connection and the fields in the **Data** pane. Fields can be dragged from the **Data** pane onto the canvas area or onto various shelves, such as **Rows**, **Columns**, **Color**, or **Size**. As we'll see, the placement of the fields will result in different encodings of the data based on the type of field.

Measures and dimensions

The fields from the data source are visible in the **Data** pane and are divided into **Measures** and **Dimensions**, which are separated by a line:

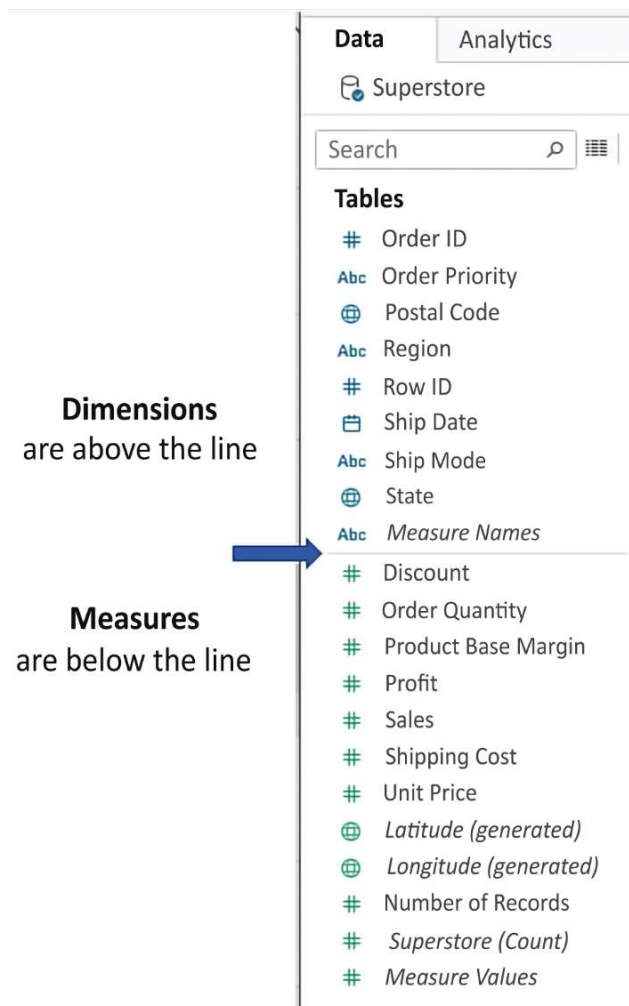


Figure 1.4: Dimensions are listed above the line and measures are listed below the line. In data sources with multiple data sources, the list of measures and dimensions is per table.

The difference between **Measures** and **Dimensions** is a fundamental concept to understand when using Tableau:

- **Measures** are values that are aggregated. For example, they are summed, averaged, or counted, or the result is the minimum or maximum value.
- **Dimensions** are values that determine the level of detail at which measures are aggregated. You can think of them as slicing the measures or creating groups into which the measures fit. The combination of dimensions used in the view defines the view's basic level of detail.

As an example (which you can view in the Chapter 01 Starter workbook on the **Measures and Dimensions** sheet), consider a view created using the Region and Sales fields from the Superstore connection:

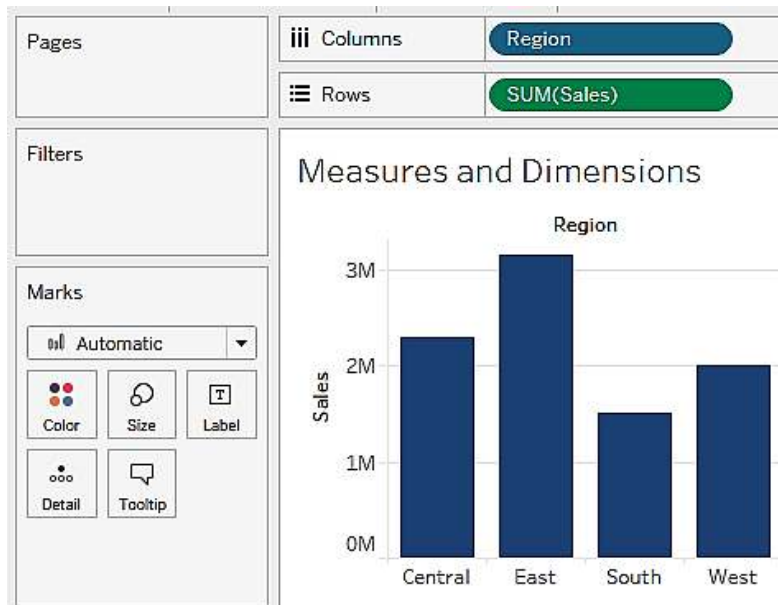


Figure 1.5: A bar chart demonstrating the use of measures and dimensions

The Sales field is used as a measure in this view. Specifically, it is being aggregated as a sum. When you use a field as a measure in the view, the aggregation type (for example, SUM, MIN, MAX, or AVG) will be shown on the active field. Note that, in the preceding example, the active field on Rows clearly indicates the sum aggregation of Sales: **SUM(Sales)**.

The Region field is a dimension with one of four values for each record of data: **Central**, **East**, **South**, or **West**. When the field is used as a dimension in the view, it slices the measure. So, instead of an overall sum of sales, the preceding view shows the sum of sales for each region.

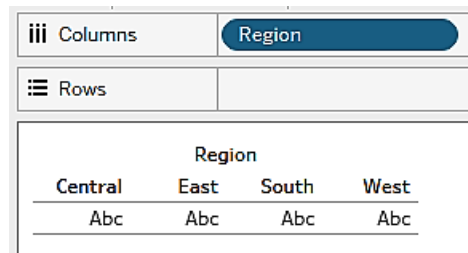
Discrete and continuous fields

Another important distinction to make with fields is whether a field is being used as **discrete** or **continuous**. Whether a field is discrete or continuous determines how Tableau visualizes it based on where it is used in the view. Tableau will give a visual indication of the default value for a field (the color of the icon in the **Data** pane) and how it is being used in the view (the color of the active field on a shelf). Discrete fields, such as Region in the previous example, are blue. Continuous fields, such as Sales, are green.

Discrete fields

Discrete (blue) fields have values that are shown as distinct and separate from one another. Discrete values can be reordered and still make sense. For example, you could easily rearrange the values of Region to be East, South, West, and Central, instead of the default order in Figure 1.6

When a discrete field is used on the **Rows** or **Columns** shelves, the field defines headers. Here, the discrete field **Region** defines the column headers:

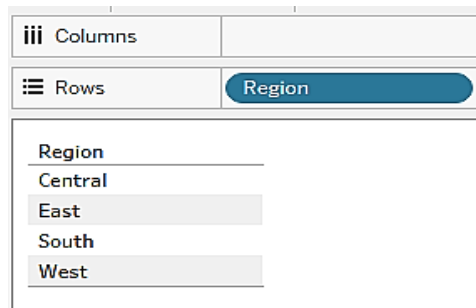


The screenshot shows the Tableau interface with 'Region' on the Columns shelf. The view displays a table with four columns: Central, East, South, and West. Each column contains the value 'Abc'.

Region			
Central	East	South	West
Abc	Abc	Abc	Abc

Figure 1.6: The discrete field on Columns defines column headers

Here, it defines the row headers:



The screenshot shows the Tableau interface with 'Region' on the Rows shelf. The view displays a list of regions: Central, East, South, and West. Each region is followed by a horizontal line, indicating a header for a row of data.

Region
Central
East
South
West

Figure 1.7: The discrete field on Rows defines row headers

When used for **Color**, a discrete field defines a discrete color palette in which each color aligns with a distinct value of the field:

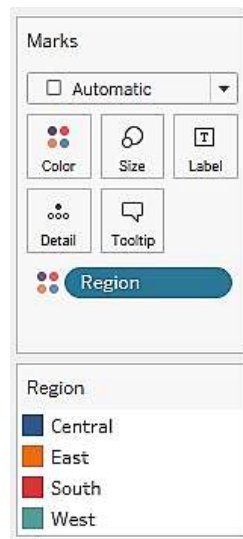


Figure 1.8: The discrete field on Color defines a discrete color palette

Continuous fields

Continuous (green) fields have values that flow from first to last as a continuum. Numeric and date fields are often (though not always) used as continuous fields in the view. The values of these fields have an order that it would make little sense to change.

When used on **Rows** or **Columns**, a continuous field defines an axis:

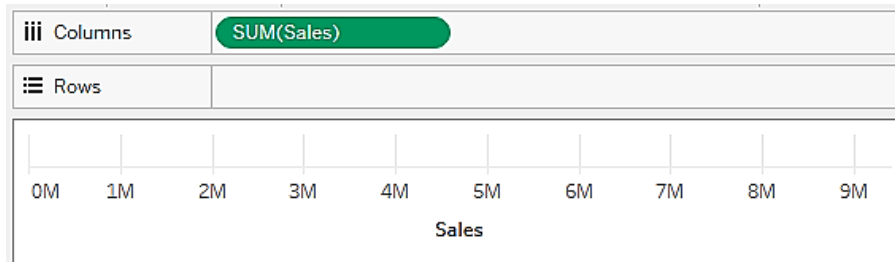


Figure 1.9: The continuous field on Columns (or Rows) defines an axis

When used for **Color**, a continuous field defines a gradient:

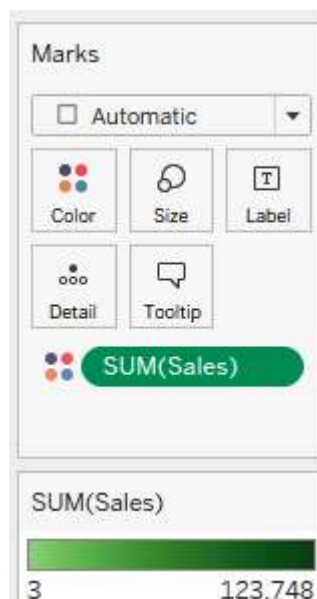


Figure 1.10: The continuous field on Color defines a gradient color palette

It is very important to note that continuous and discrete are different concepts from measure and dimension. While most dimensions are discrete by default, and most measures are continuous by default, it is possible to use any measure as a discrete field and some dimensions as continuous fields in the view, as shown here:

		Can be	
		Discrete	Continuous
A	Dimension	Yes	If Numeric or Date
	Measure	Yes	Yes

Figure 1.11: Measures and dimensions can be discrete or continuous



To change the default setting of a field, right-click on the field in the **Data** pane and select **Convert to Discrete** or **Convert to Continuous**.

To change how a field is used in the view, right-click on the field in the view and select **Discrete** or **Continuous**. Alternatively, you can drag and drop the fields between **Dimensions** and **Measures** in the **Data** pane, which will automatically convert the field.

In general, you can think of the differences between the types of fields as follows:

- Choosing between a dimension and measure tells Tableau *how to slice or aggregate* the data.
- Choosing between discrete and continuous tells Tableau *how to display* the data with a header or an axis and defines individual colors or a gradient.

As you work through the examples in this book, pay attention to the fields you are using to create the visualizations, whether they are dimensions or measures, and whether they are discrete or continuous. Experiment with changing fields in the view from continuous to discrete, and vice versa, to gain an understanding of the differences in the visualization. While the concepts of dimensions/measures and discrete/continuous may seem complex for now, with a little practice they will become second nature. And they form the foundation for a solid understanding of how and why Tableau works! Come back to them often. We'll put this understanding into practice as we turn our attention to visualizing data.

Visualizing data

A new connection to a data source is an invitation to explore and discover! At times, you may come to the data with very well-defined questions and a strong sense of what you expect to find. Other times, you will come to the data with general questions and very little idea of what you will find. Tableau empowers you to rapidly and iteratively explore the data, ask new questions, and make new discoveries.

The visualization examples in this section cover a few of the most foundational visualization types. As you work through the examples, keep in mind that the goal is not simply to learn how to create a specific chart. Rather, the examples are designed to help you think through the process of asking questions about the data and getting answers through iterations of visualization. Tableau is designed to make that process intuitive, rapid, and transparent.



Something that is far more important than memorizing the steps to create a specific chart type is understanding how and why to use Tableau to create a chart and being able to adjust your visualization to gain new insights as you ask new questions.

Bar charts

Bar charts visually represent data in a way that makes the comparison of values across different categories easy. The length of the bar is the primary means by which you will visually understand the data. You may also incorporate color, size, stacking, and order to communicate additional attributes and values.

Creating bar charts in Tableau is very easy. Simply drag and drop the measure you want to see onto either the **Rows** or **Columns** shelf and the dimension that defines the categories onto the opposing **Rows** or **Columns** shelf.

As an analyst for **Superstore**, you are ready to begin a discovery process focused on sales (especially the dollar value of sales). As you follow the examples, work your way through the sheets in the Chapter 01 Starter workbook. The Chapter 01 Complete workbook contains the complete examples so that you can compare your results at any time:

1. Click on the **Sales by Department** tab to view that sheet.
2. Drag and drop the **Sales** field from the **Data** pane onto the **Columns** shelf. You now have a bar chart with a single bar representing the sum of sales for all of the data in the data source.

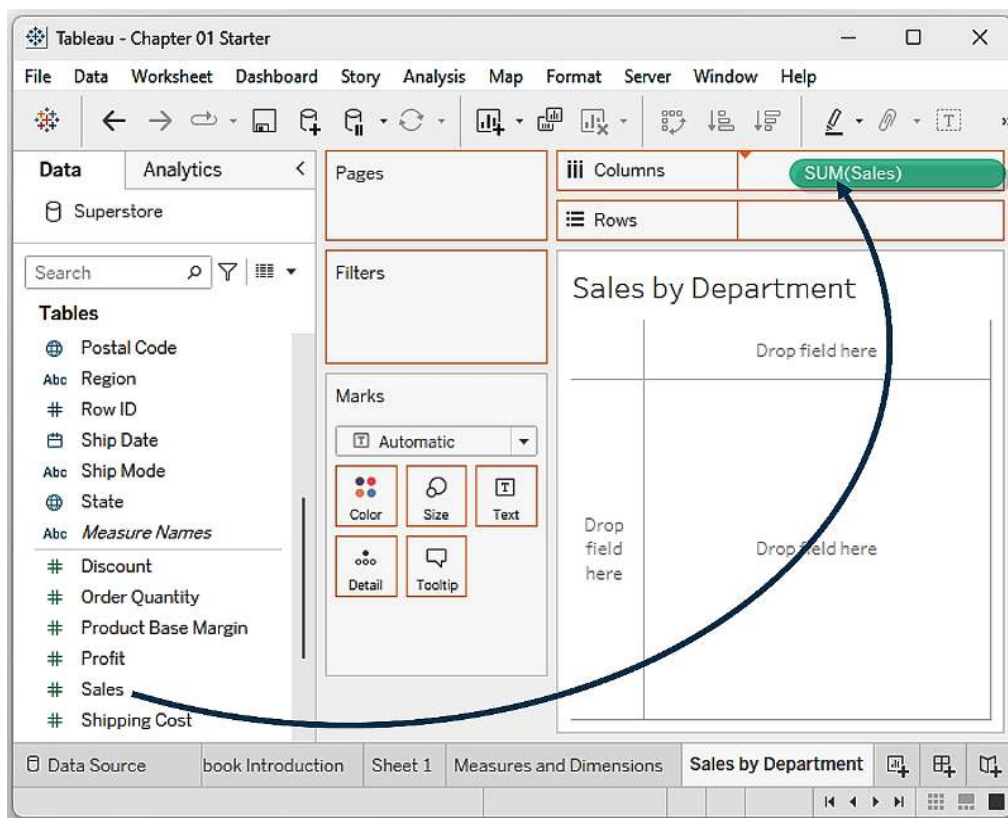


Figure 1.12: Drag and drop Sales from the Data pane to the Columns shelf

3. Drag and drop the Department field from the **Data** pane to the **Rows** shelf. This slices the data to give you three bars, each having a length that corresponds to the sum of sales for each department:

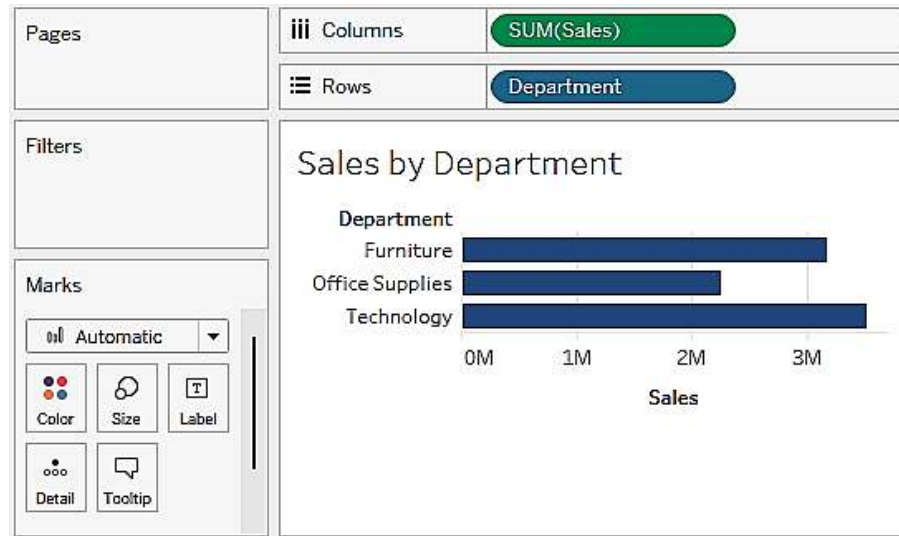


Figure 1.13: The view *Sales by Department* should look like this when you have completed the preceding steps

You now have a horizontal bar chart. This makes comparing the sales between the departments easy. The marks type drop-down menu on the **Marks** card is set to **Automatic** and indicates that Tableau has determined that bars are the best visualization given the fields you have placed in the view. As a dimension, Department slices the data. Being discrete, it defines row headers for each department in the data. As a measure, the Sales field is aggregated. Being continuous, it defines an axis. The mark type of bar causes individual bars for each department to be drawn from 0 to the value of the sum of sales for that department.



Typically, Tableau draws a mark (such as a bar, a circle, or a square) for every combination of dimensional values in the view. In this simple case, Tableau is drawing a single bar mark for each dimensional value (Furniture, Office Supplies, and Technology) of Department. The type of mark is indicated and can be changed in the drop-down menu on the **Marks** card. The number of marks drawn in the view can be observed on the lower-left status bar.

Tableau draws different marks in different ways; for example, bars are drawn from 0 (or the end of the previous bar, if stacked) along the axis. Circles and other shapes are drawn at locations defined by the value(s) of the field that is defining the axis. Take a moment to experiment with selecting different mark types from the drop-down menu on the **Marks** card. A solid grasp of how Tableau draws different mark types will help you to master the tool.

Iterations of bar charts for deeper analysis

Using the preceding bar chart, you can easily see that the **Technology** department has more total sales than either the **Furniture** or **Office Supplies** department. What if you want to further understand sales amounts for departments across various regions? Follow these two steps:

1. Navigate to the **Bar Chart (two levels)** sheet, where you will find an initial view that is identical to the one you created earlier.
2. Drag the **Region** field from the **Data** pane to the **Rows** shelf and drop it to the left of the **Department** field already in view.

You should now have a view that looks like this:

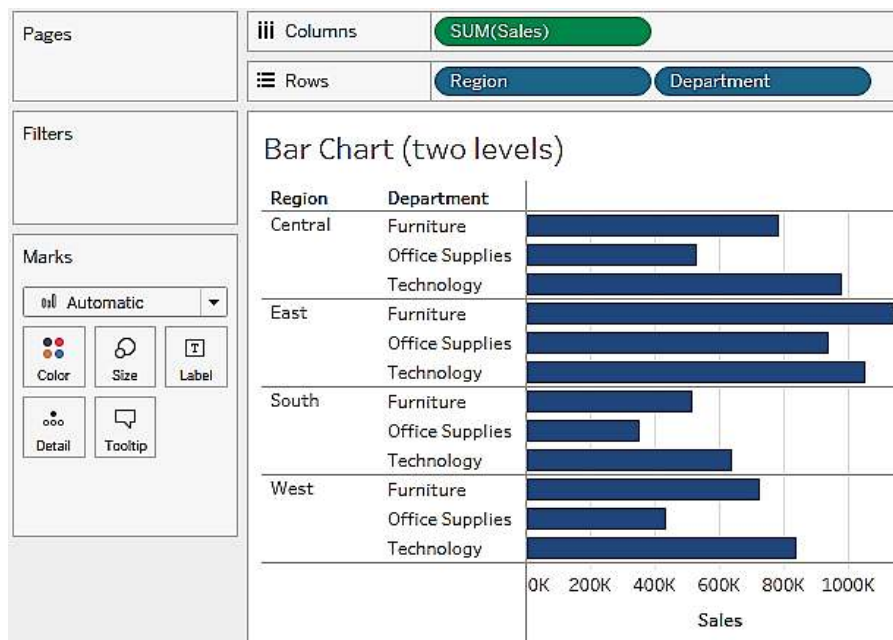


Figure 1.14: The view, *Bar Chart (two levels)*, should look like this when you have completed the preceding steps

You still have a horizontal bar chart, but now you've introduced **Region** as another dimension, which changes the level of detail in the view and further slices the aggregate of the sum of sales. By placing **Region** before **Department**, you can easily compare the sales of each department within a given region.

Now you are starting to make some discoveries. For example, the **Technology** department has the most sales in every region, except in the **East**, where **Furniture** had higher sales. **Office Supplies** has never had the highest sales in any region.

Consider an alternate view, using the same fields arranged differently:

1. Navigate to the **Bar Chart (stacked)** sheet, where you will find a view that is identical to the two-level bar chart.

2. Drag the Region field from the Rows shelf and drop it onto the Color shelf:

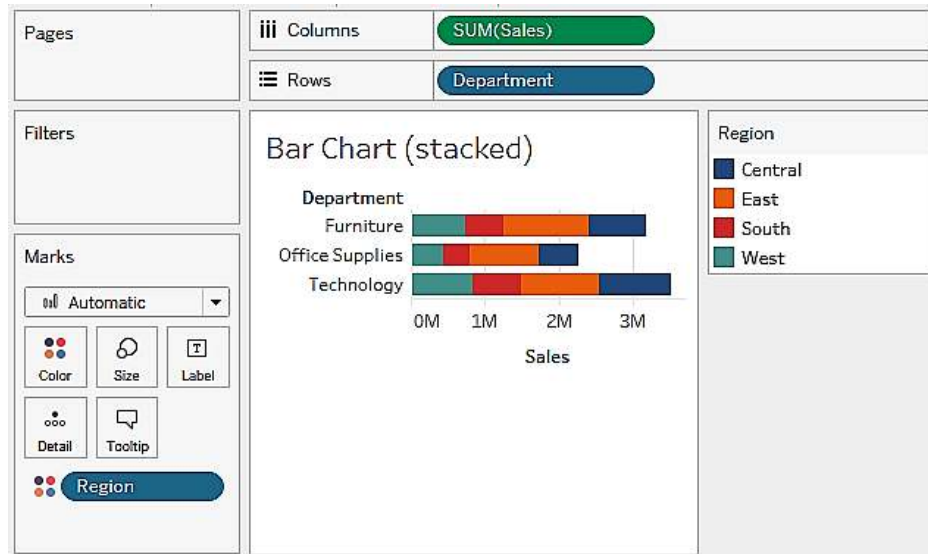


Figure 1.15: The Bar Chart (stacked) view should look like this

Instead of a **side-by-side bar chart**, you now have a **stacked bar chart**. Each segment of the bar is color-coded by the Region field. Additionally, a color legend has been added to the workspace. You haven't changed the level of detail in the view, so sales are still summed for every combination of Region and Department.

- The **view level of detail** is a key concept when working with Tableau. In most basic visualizations, the combination of values of all dimensions in the view defines the lowest level of detail for that view. All measures will be aggregated or sliced by the lowest level of detail. For simple views, the number of marks (indicated in the lower-left status bar) corresponds to the number of unique combinations of dimensional values. That is, there will be one mark for each combination of dimension values. Consider how the view level of detail changes based on the fields used in the view:
- If Department is the only field used as a dimension, you will have a view at the department level of detail, and all measures in the view will be aggregated per department.
- If Region is the only field used as a dimension, you will have a view at the region level of detail, and all measures in the view will be aggregated per region.
- If you use both Department and Region as dimensions in the view, you will have a view at the level of department and region. All measures will be aggregated per unique combination of department and region, and there will be one mark for each combination of department and region.

Stacked bars can be useful when you want to understand part-to-whole relationships. It is now easier to see what portion of the total sales of each department is made in each region. However, it is very difficult to compare sales for most of the regions across departments. For example, can you easily tell which department had the highest sales in the **East** region? It is difficult because, with the exception of the **West** region, every segment of the bar has a different starting point.

Now take some time to experiment with the bar chart to see what variations you can create:

1. Navigate to the **Bar Chart (experimentation)** sheet.
2. Try dragging the **Region** field from **Color** to the other various shelves on the **Marks** card, such as **Size**, **Label**, and **Detail**. Observe that in each case, the bars remain stacked but are redrawn based on the visual encoding defined by the **Region** field.
3. Use the **Swap** button on the toolbar to swap fields on **Rows** and **Columns**. This allows you to very easily change from a horizontal bar chart to a vertical bar chart (and vice versa):

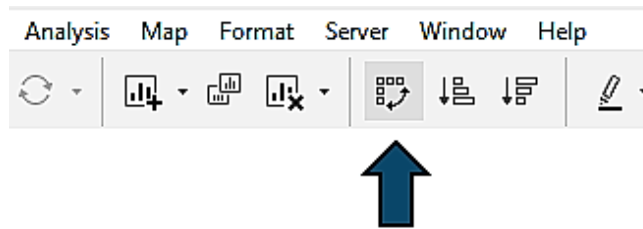


Figure 1.16: Swap Rows and Columns button

4. Drag and drop **Sales** from the **Data** pane on top of the **Region** field on the **Marks** card to replace it. Drag the **Sales** field to **Color** if necessary and notice how the color legend is a gradient for the continuous field.
5. Experiment further by dragging and dropping other fields onto various shelves. Note the behavior of Tableau for each action you take.
6. From the **File** menu, select **Save**.



In the event of a system crash, the **Autosave** feature should protect your work. The next time you open Tableau, you will be prompted to recover any previously open workbooks that had not been manually saved. You should still develop a habit of saving your work early and often and maintain appropriate backups.

As you continue to explore various iterations, you'll gain confidence with the flexibility available to visualize your data.

Line charts

Line charts connect related marks in a visualization to show movement or a relationship between those connected marks. The position of the marks and the lines that connect them are the primary means of communicating the data. Additionally, you can use size and color to communicate additional information.

The most common kind of line chart is a **time series**. A time series shows the movement of values over time. Creating one in Tableau requires only a date and a measure.

Continue your analysis of Superstore sales using the Chapter 01 Starter workbook you just saved:

1. Navigate to the **Sales over time** sheet.
2. Drag the **Sales** field from the **Data** pane to **Rows**. This gives you a single, vertical bar representing the sum of all sales in the data source.
3. To turn this into a time series, you must introduce a date. Drag the **Order Date** field from the **Data** pane to **Columns**. Tableau has a built-in date hierarchy, and the default level of **Year** has given you a line chart connecting four years. Notice that you can clearly see an increase in sales year after year:

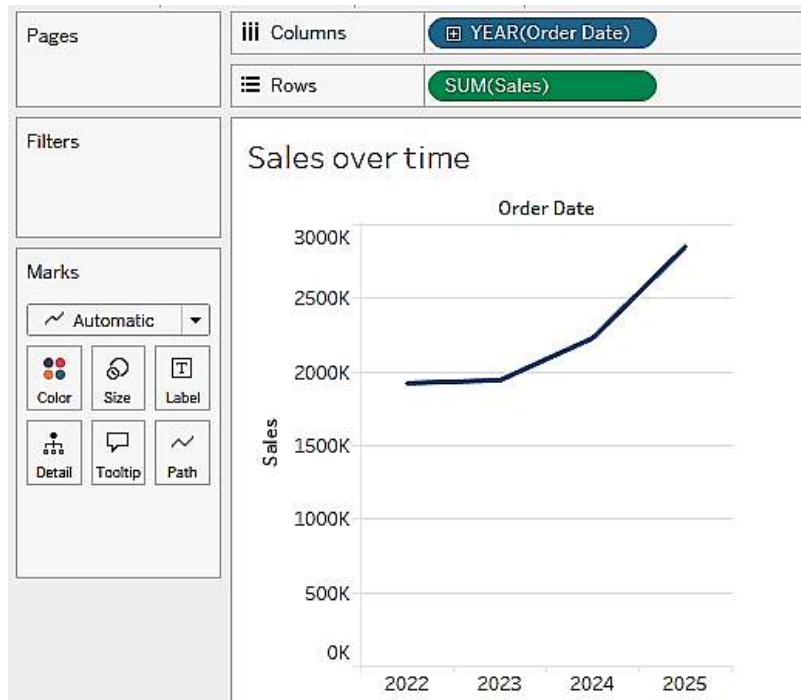


Figure 1.17: An interim step in creating the final line chart; this shows the sum of sales by year



Throughout the examples in this book, we'll often refer to using the drop-down menu on a field. You'll find this menu by right-clicking on a field or by selecting the drop-down indicator that appears as you hover over a field:



Figure 1.18: The drop-down indicator that appears on fields when you hover

 **Quick tip:** Need to see a high-resolution version of this image? Open this book in the next-gen Packt Reader or view it in the PDF/ePub copy.

 **The next-gen Packt Reader and a free PDF/ePub copy** of this book are included with your purchase. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



4. Use the drop-down menu on the **YEAR(Order Date)** field on **Columns** (or right-click on the field) and switch the date field to use **Quarter**. You may notice that **Quarter** is listed twice in the drop-down menu. We'll explore the various options for date parts, values, and hierarchies in the *Visualizing dates and times* section of *Chapter 3, Moving Beyond Basic Visualizations*. For now, select the second option:

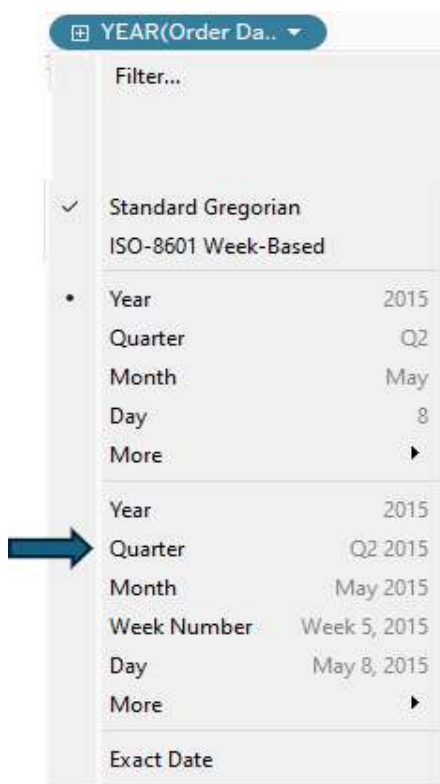


Figure 1.19: Select the second Quarter option in the drop-down menu

- Notice that the cyclical pattern is quite evident when looking at sales by quarter:

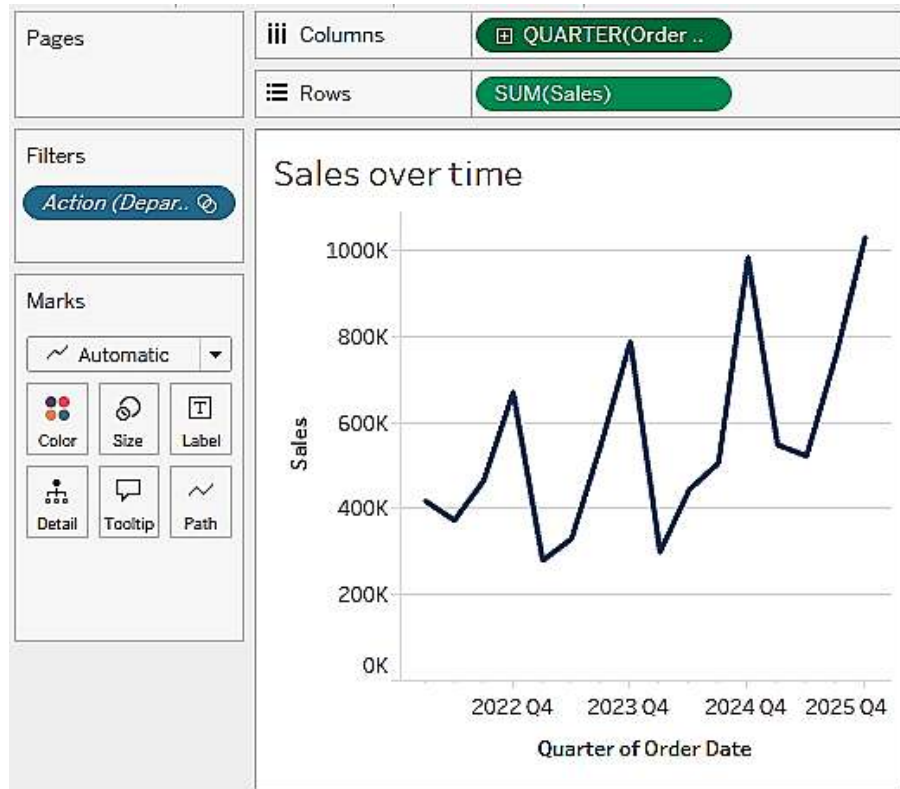


Figure 1.20: Your final view shows sales over each quarter for the last several years

Let's consider some variations of line charts that allow you to ask and answer even deeper questions.

Iterations of line charts for deeper analysis

Right now, you are looking at the overall sales over time. Let's do some analysis at a slightly deeper level:

- Navigate to the **Sales over time (overlapping lines)** sheet, where you will find a view that is identical to the one you just created.
- Drag the **Region** field from the **Data** pane to **Color**. Now you have a line per region. Each line is a different color, and a legend indicates which color matches each region. As in the stacked bars example, adding a dimension to **Color** splits the marks. However, unlike the bars, where the segments were stacked, the lines are not stacked. Instead, the lines are drawn at the exact value for the sum of sales for each region and quarter. This allows easy and accurate comparison. It is interesting to note that the cyclical pattern can be observed for each region:

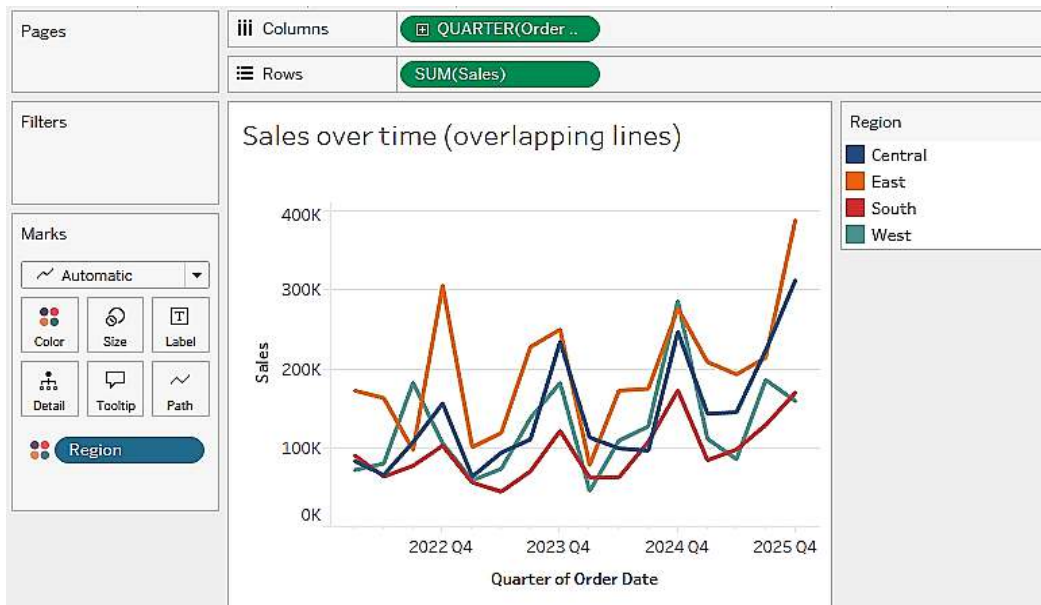


Figure 1.21: This line chart shows the sum of sales by quarter with different colored lines for each region

With only four regions, it's relatively easy to keep the lines separate. But what about dimensions that have even more distinct values? Let's consider that case in the following example:

1. Navigate to the **Sales over time (multiple rows)** sheet, where you will find a view that is identical to the one you just created.
2. Drag the **Category** field from the **Data** pane and drop it directly on top of the **Region** field currently on the **Marks** card. This replaces the **Region** field with **Category**. You now have 17 overlapping lines.



Often, you'll want to avoid more than two or three overlapping lines. But you might also consider using color or size to showcase an important line in the context of the others. Also, note that clicking on an item in the **Color** legend will highlight the associated line in the view. Highlighting is an effective way to pick out a single item and compare it to all the others.

3. Drag the **Category** field from **Color** on the **Marks** card and drop it into **Rows**. You now have a line chart for each category. This enables you to compare each product over time without an overwhelming overlap. You can easily see trends and patterns over time. This is the start of a sparklines visualization that will be developed more fully in *Chapter 10, Advanced Visualizations*:

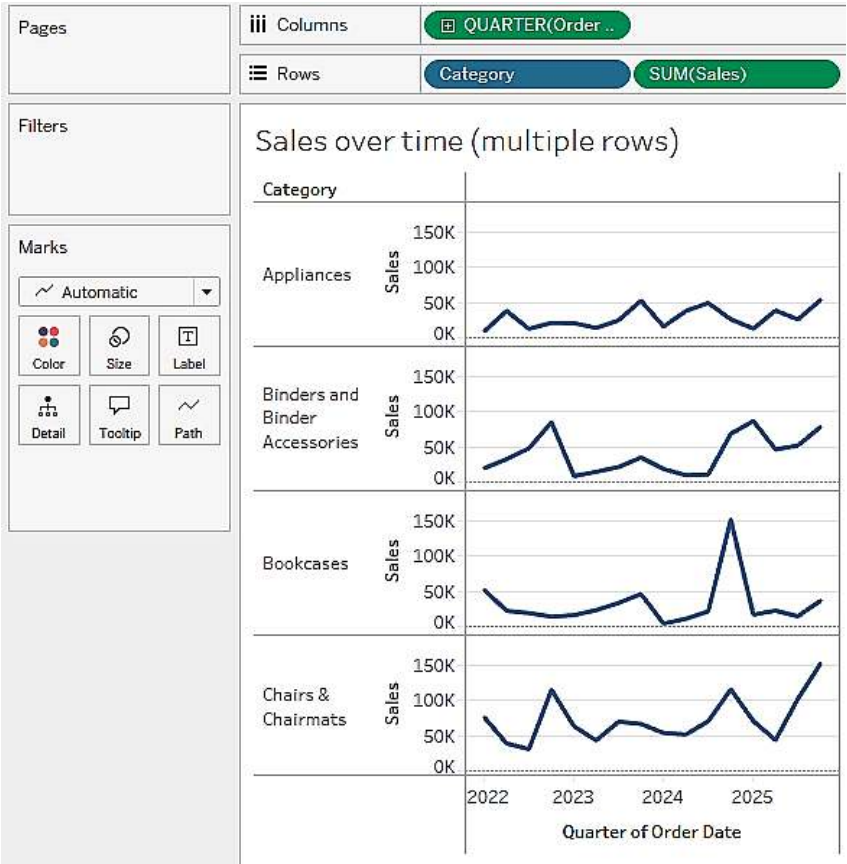


Figure 1.22: Your final view should be a series of line charts for each category

The variations in lines for each Category allow you to notice variations in the trends, extremes, and rate of change.

Geographic visualizations

In Tableau, the built-in geographic database recognizes geographic roles for fields such as Country, State, City, Airport, Congressional District, or Zip Code. Even if your data does not contain latitude and longitude values, you can simply use geographic fields to plot locations on a map. If your data *does* contain latitude and longitude fields, you may use those instead of the generated values.

Tableau will automatically assign geographic roles to some fields based on a field name and a sampling of values in the data. You can also assign or reassign geographic roles to any field by right-clicking on the field in the **Data** pane and using the **Geographic Role** option. This is also a good way to see what built-in geographic roles are available.

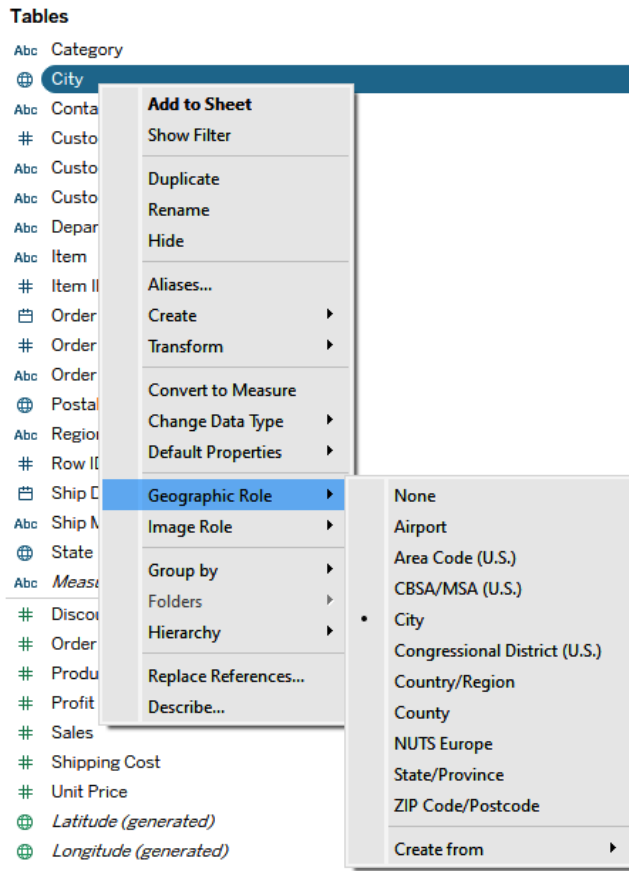


Figure 1.23: The drop-down menu of a field reveals an option to assign a geographic role

Geographic visualization is incredibly valuable when you need to understand where things happen and whether there are any spatial relationships within the data. Tableau offers several types of geographic visualization:

- Filled maps
- Symbol maps
- Density maps

Additionally, Tableau can read spatial files and geometries from some databases and render spatial objects, polygons, and more. We'll take a look at these and other geospatial capabilities in *Chapter 12, Exploring Mapping and Advanced Geospatial Features*. For now, we'll consider some foundational principles for geographic visualization.

Filled maps

Filled maps fill areas such as countries, states, or ZIP codes to show a location. The color that fills the area can be used to communicate measures such as average sales or population, as well as dimensions such as region. These maps are also called **choropleth maps**.

Let's say you want to understand sales for Superstore and see whether there are any patterns geographically.



If your regional settings are not set to the US, you may need to use the **Edit Locations** option to set the country to **United States**.

You might take an approach like the following:

1. Navigate to the **Sales by State** sheet.
2. Double-click on the **State** field in the **Data** pane. Tableau automatically creates a geographic visualization using the **Latitude (generated)**, **Longitude (generated)**, and **State** fields. The Automatic mark type is a circle, which is drawn in the geographic center of each state.
3. Drag the **Sales** field from the **Data** pane and drop it on the **Color** shelf on the **Marks** card. Based on the fields and shelves you've used, Tableau has switched the automatic mark type to **Map**:

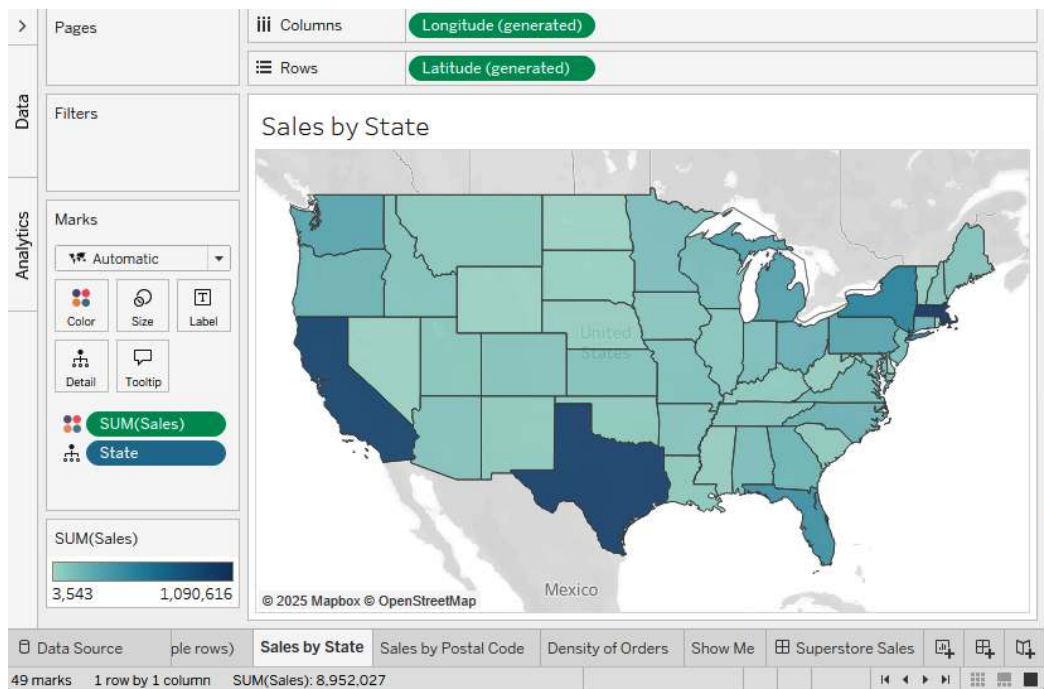


Figure 1.24: A filled map showing the sum of sales per state

The filled map fills each state with a single color to indicate the relative sum of sales for that state. The color legend, now visible in the view, gives the range of values and indicates that the state with the least sales had a total of 3,543 and the state with the most sales had a total of 1,090,616.

When you look at the number of marks displayed in the bottom status bar, you'll see that it is **49**. Careful examination reveals that the marks consist of the lower 48 states and Washington DC; Hawaii and Alaska are not shown. Tableau will only draw a geographic mark, such as a filled state, if it exists in the data and is not excluded by a filter.

Observe that the map displays Canada, Mexico, and other locations not included in the data. These are part of a background image retrieved from an online map service. The state marks are then drawn on top of the background image. We'll look at how you can customize the map in *Chapter 12, Exploring Mapping and Advanced Geospatial Features*.

Filled maps can work well in interactive dashboards and have quite a bit of aesthetic value. However, certain kinds of analyses are very difficult with filled maps. Unlike other visualization types, where size can be used to communicate facets of the data, the size of a filled geographic region only relates to the geographic size and can make comparisons difficult. For example, which state has the highest sales? You might be tempted to say Texas or California because the larger size influences your perception, but would you have guessed Massachusetts? Some locations may be small enough that they won't even show up compared to larger areas.

Use filled maps with caution and consider pairing them with other visualizations on dashboards for clear communication.

Symbol maps

With symbol maps, marks on the map are not drawn as filled regions; rather, marks are shapes or symbols placed at specific geographic locations. The size, color, and shape may also be used to encode additional dimensions and measures.

Continue your analysis of Superstore sales by following these steps:

1. Navigate to the **Sales by Postal Code** sheet.
2. Double-click on **Postal Code** in the **Data** pane. Tableau automatically adds **Postal Code** to **Detail** on the **Marks** card and **Longitude (generated)** and **Latitude (generated)** to **Columns** and **Rows**. The mark type is set to a circle by default, and a single circle is drawn for each postal code at the correct latitude and longitude.
3. Drag **Sales** from the **Data** pane to the **Size** shelf on the **Marks** card. This will resize each circle based on the total sales for that postal code.
4. Drag **Profit** from the **Data** pane to the **Color** shelf on the **Marks** card. This encodes the mark color to correspond to the sum of profit. You can now see the geographic location of profit and sales at the same time. This is useful because you will see some locations with high sales and low profit, which may require some action.

The final view should look similar to the following (note that some fine-tuned adjustments have been made to the size and color):

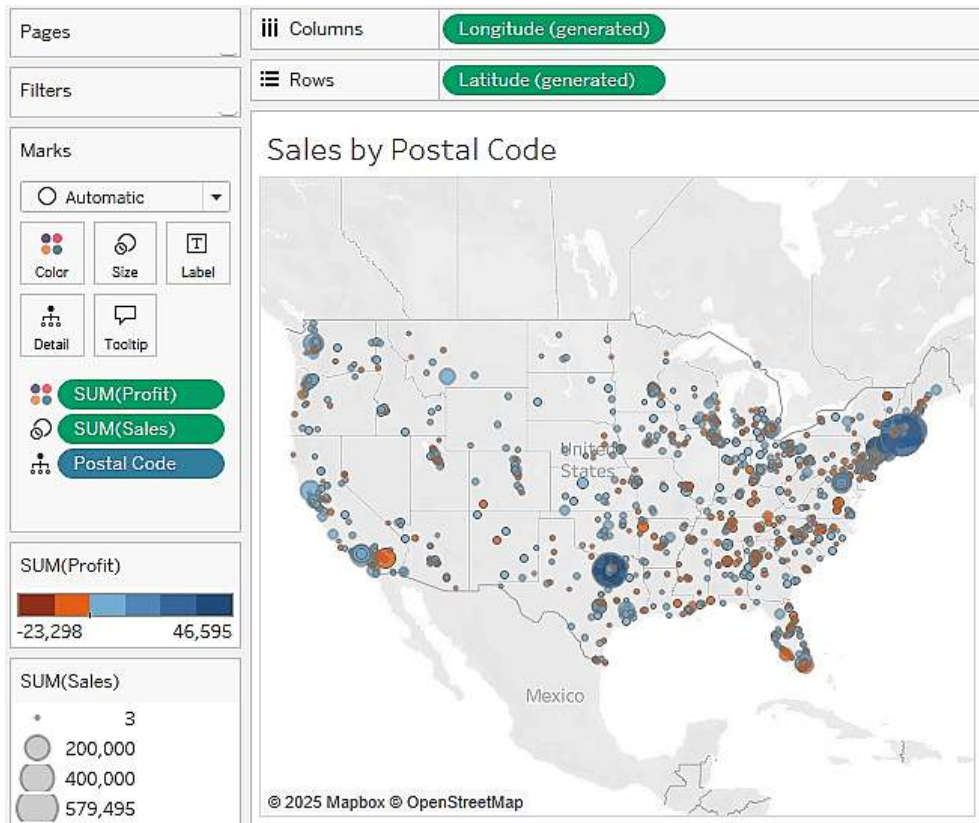


Figure 1.25: A symbol map showing the sum of profit (encoded with color) and the sum of sales (encoded with size) per postal code

Sometimes, you'll want to adjust the marks on a symbol map to make them more visible. Some options include the following:

- If the marks are overlapping, click on the **Color** shelf and set the transparency to somewhere between 50% and 75%. Additionally, add a dark border. This makes the marks stand out, and you can often better discern any overlapping marks.
- If marks are too small, click on the **Size** shelf and adjust the slider. You may also double-click on the **Size** legend and edit the details of how Tableau assigns size.
- If the marks are too faint, double-click on the **Color** legend and edit the details of how Tableau assigns color. This is especially useful when you are using a continuous field that defines a color gradient.

A combination of tweaking the size and using **Stepped Color** and **Use Full Color Range**, as shown here, produced the result for this example:

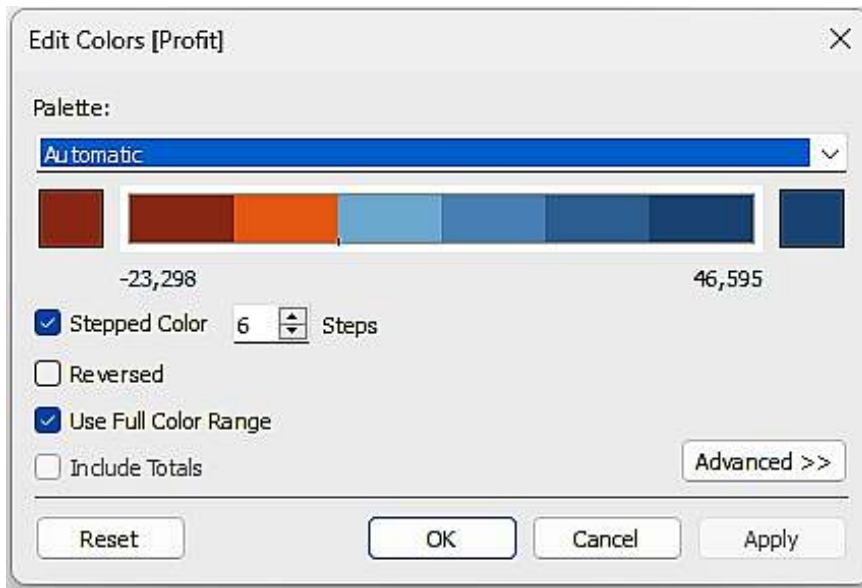


Figure 1.26: The Edit Colors dialog includes options for changing the number of steps, reversing, and using the full color range, including totals, as well as advanced options for adjusting the range and center point

Unlike filled maps, symbol maps allow you to use size to visually encode aspects of the data. Symbol maps also allow greater precision. In fact, if you have latitude and longitude in your data, you can very precisely plot marks at a street address level of detail. This type of visualization also allows you to map locations that do not have clearly defined boundaries.

Sometimes, when you manually select **Map** in the **Marks** card drop-down menu, you will get an error message indicating that filled maps are not supported at the level of detail in the view. In those cases, Tableau is rendering a geographic location that does not have built-in shapes.

Other than cases where filled maps are not possible, you will need to decide which type best meets your needs. We'll also consider the possibility of combining filled maps and symbol maps in a single view in later chapters.

Density maps

Density maps show the spread and concentration of values within a geographic area. Instead of individual points or symbols, the marks blend together, showing greater intensity in areas with a high concentration. You can control the **Color**, **Size**, and **Intensity**.

Let's say you want to understand the geographic concentration of orders. You might create a density map using the following steps:

1. Navigate to the **Density of Orders** sheet.
2. Double-click on the Postal Code field in the **Data** pane. Just as before, Tableau automatically creates a symbol map geographic visualization using the Latitude (generated), Longitude (generated), and State fields.
3. Using the drop-down menu on the **Marks** card, change the mark type from **Automatic** to **Density**. The individual circles now blend together, showing concentrations:

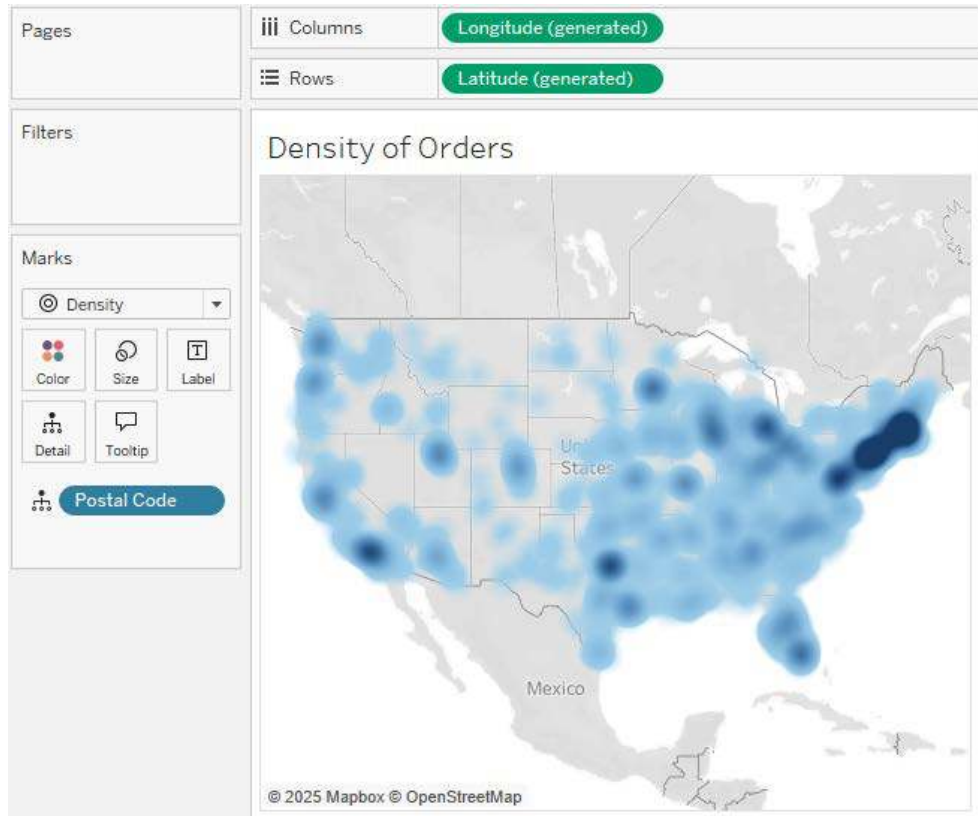


Figure 1.27: A density map showing concentration by postal code

Try experimenting with the **Color** and **Size** options. Clicking on **Color**, for example, reveals some options specific to the **Density** mark type:

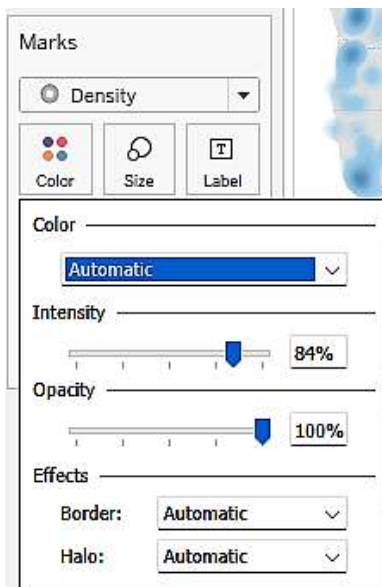


Figure 1.28: Options for adjusting the Color, Intensity, Opacity, and Effects for Density marks

Several color palettes are available that work well for density marks (the default ones work well with light color backgrounds, but there are others designed to work with dark color backgrounds). The **Intensity** slider allows you to determine how intensely the marks should be drawn based on concentrations. The **Opacity** slider lets you decide how transparent the marks should be.

This density map displays a high concentration of orders from the east coast. Sometimes, you'll see patterns that merely reflect population density. In such cases, your analysis may not be particularly meaningful. In this case, the concentration on the East Coast compared to the lack of density on the West Coast is intriguing.

With an understanding of how to use Tableau to visualize your data, let's turn our attention to a feature that generates some visualizations automatically, Show Me!

Using Show Me

Show Me is a powerful component of Tableau that arranges selected and active fields into the places required for the selected visualization type. The **Show Me** toolbar displays small thumbnail images of different types of visualizations, allowing you to create visualizations with a single click. Based on the fields you select in the **Data** pane and the fields that are already in view, **Show Me** will enable possible visualizations and highlight a recommended visualization.

It does this with visual analytics best practices built in, giving you the ability to quickly see a variety of visualizations that might answer your questions.

Explore the features of **Show Me** by following these steps:

1. Navigate to the **Show Me** sheet.
2. If the **Show Me** pane is not expanded, click on the **Show Me** button in the upper right of the toolbar to expand the pane.
3. Press and hold the **Ctrl** key while clicking on the Postal Code, State, and Profit fields in the **Data** pane to select each of those fields. With those fields highlighted, **Show Me** should look like this:



Figure 1.29: The Show Me interface

Notice that the **Show Me** window has enabled certain visualization types, such as **text tables**, **heat maps**, **symbol maps**, **filled maps**, and **bar charts**. These are the visualizations that are possible given the fields already in the view, in addition to any selected in the **Data** pane.

Show Me highlights the recommended visualization for the selected fields and gives a description of what fields are required as you hover over each visualization type. Symbol maps, for example, require one geographic dimension and 0 to 2 measures.

Other visualizations are grayed out, such as **lines**, **area charts**, and **histograms**. Show Me will not create these visualization types with the fields that are currently in the view and are selected in the **Data** pane. Hover over the grayed-out line charts option in Show Me. It indicates that line charts require one or more measures (which you have selected) but also require a date field (which you have not selected).

Tableau will draw line charts with fields other than dates. Show Me gives you options for what is typically considered good practice for visualizations. However, there may be times when you know that a line chart would represent your data better. Understanding how Tableau renders visualizations based on fields and shelves instead of always relying on Show Me will give you much greater flexibility in your visualizations and will allow you to rearrange things when Show Me doesn't give you the exact results you want. At the same time, you will need to cultivate an awareness of good visualization practices.

Show Me can be a powerful way in which to quickly iterate through different visualization types as you search for insights into the data. But as a data explorer, analyst, and storyteller, you should consider Show Me as a helpful guide that gives suggestions. You may know that a certain visualization type will answer your questions more effectively than the suggestions of Show Me. You also may have a plan for a visualization type that will work well as part of a dashboard but isn't even included in Show Me.

You will be well on your way to learning and mastering Tableau when you can use Show Me effectively but feel just as comfortable with building visualizations without it. Show Me is powerful for quickly iterating through visualizations as you look for insights and raise new questions. It is useful for starting with a standard visualization that you will further customize. It is wonderful as a teaching and learning tool.

However, be careful that it doesn't become a crutch, where you use it without understanding how visualizations are actually built from the data. Take the time to evaluate why certain visualizations are or are not possible. Pause to see what fields and shelves were used when you selected a certain visualization type.

End the example by experimenting with Show Me by clicking on various visualization types, looking for insights into the data that may be revealed based on the visualization type. For example, **circle views** and **box-and-whisker plots** show the distribution of postal codes for each state. Bar charts easily expose several postal codes with negative profits.

Now that you have become familiar with creating individual views of the data, let's turn our attention to putting it all together in a dashboard.

Putting everything together in a dashboard

Often, you'll need more than a single visualization to communicate the full story of the data. In these cases, Tableau makes it very easy for you to use multiple visualizations together on a dashboard. In Tableau, a **dashboard** is a collection of views, filters, parameters, images, and other objects that work together to communicate a data story. Dashboards are often interactive and allow end users to explore different facets of the data.

Dashboards serve a wide variety of purposes and can be tailored to suit a wide variety of audiences. Consider the following possible dashboards:

- A summary-level view of profit and sales to allow executives to take a quick glimpse at the current status of the company
- An interactive dashboard, allowing sales managers to drill into sales territories to identify threats or opportunities

- A dashboard allowing doctors to track patient re-admissions, diagnoses, and procedures to make better decisions about patient care
- A dashboard giving real-estate investors the ability to identify trends and make decisions about various rental properties
- An interactive dashboard for loan officers to make lending decisions based on portfolios broken down by credit ratings and geographic location

Considerations for different audiences and advanced techniques will be covered in detail in *Chapter 8, Telling a Data Story with Dashboards*.

The dashboard interface

You can create a new dashboard by clicking the **New Dashboard** button next to the tabs at the bottom of the Tableau window or by selecting **Dashboard | New Dashboard** from the menu.

When you create a new dashboard, the interface will be slightly different than it is when designing a single view. We'll start designing your first dashboard after a brief look at the interface. You might navigate to the **Superstore Sales** sheet and take a quick look at it yourself.

The dashboard window consists of several key components. Techniques for using these objects will be detailed in *Chapter 8, Telling a Data Story with Dashboards*. For now, focus on gaining some familiarity with the options that are available.

One thing you'll notice is that the left sidebar has been replaced with dashboard-specific content:

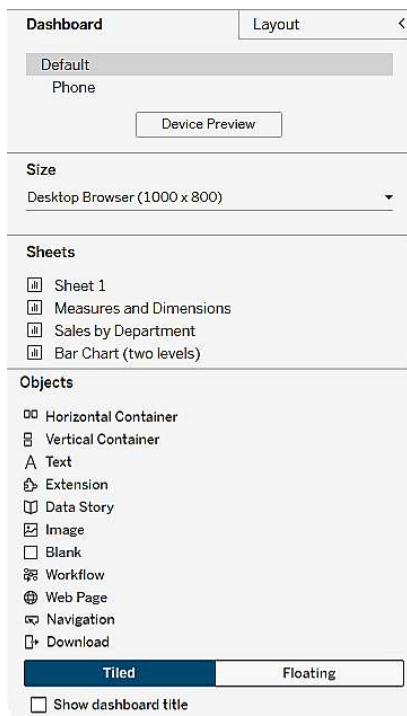


Figure 1.30: The sidebar for dashboards

The left sidebar contains two tabs:

- A **Dashboard** tab, for sizing options and adding sheets and objects to the dashboard
- A **Layout** tab, for adjusting the layout and formatting of various objects on the dashboard

The **Dashboard** pane contains options for previewing based on the target device along with several sections:

- A **Size** section, for dashboard sizing options
- A **Sheets** section, containing all sheets (views) available to place on the dashboard
- An **Objects** section with additional objects that can be added to the dashboard

You can add sheets and objects to a dashboard by dragging and dropping. As you drag the view, a light-gray shading will indicate the location of the sheet in the dashboard once it is dropped. You can also double-click on any sheet and it will be added automatically.

In addition to adding sheets, the following objects may be added to the dashboard:

- **Horizontal** and **Vertical** layout containers will give you finer control over the layout.
- **Text** allows you to add text labels and titles.
- An **Extension** gives you the ability to add controls and objects that you or a third party have developed for interacting with the dashboard and providing extended functionality. We'll cover extensions in *Chapter 16, Extending Tableau: Extensions and Scripts*.
- A **Data Story** (requires Tableau Server) provides natural language insights into your data.
- An **Image** allows you to embed images in your dashboard.
- A **Blank** object allows you to preserve blank space in a dashboard, or it can serve as a placeholder until additional content is designed.
- A **Workflow** object allows you to trigger actions in a Salesforce flow.
- A **Navigation** object allows the user to navigate between dashboards.
- A **Download** button allows end users to download data or export the dashboard as a PowerPoint, PDF, or image.

Using the toggle, you can select whether new objects will be added as **Tiled** or **Floating**. **Tiled** objects will snap into a tiled layout next to other tiled objects or within layout containers. **Floating** objects will float on top of the dashboard in successive layers. Benefits and scenarios for when you would use either tiled or floating design methods are covered in detail in *Chapter 8, Telling a Data Story with Dashboards*. In this chapter, we'll leave the default of **Tiled** selected, allowing views to easily “snap into” the dashboard.

When a worksheet is first added to a dashboard, any legends, filters, or parameters that were visible in the worksheet view will be added to the dashboard. If you wish to add them at a later point, select the sheet in the dashboard and click on the little drop-down caret on the upper-right side. Nearly every object has a drop-down caret, providing many options for fine-tuning the appearance of the object and controlling behavior.

Take note of the various controls that become visible for selected objects on the dashboard:

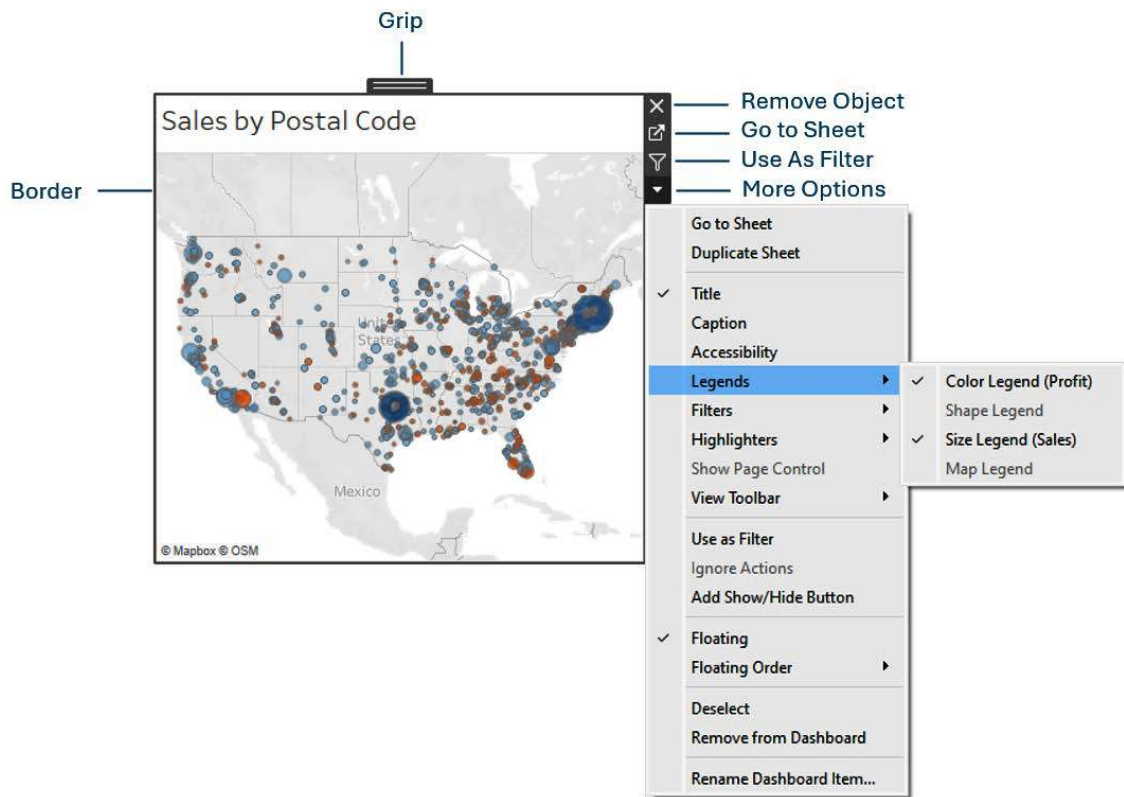


Figure 1.31: Various controls and UI elements become visible when selecting an object on a dashboard

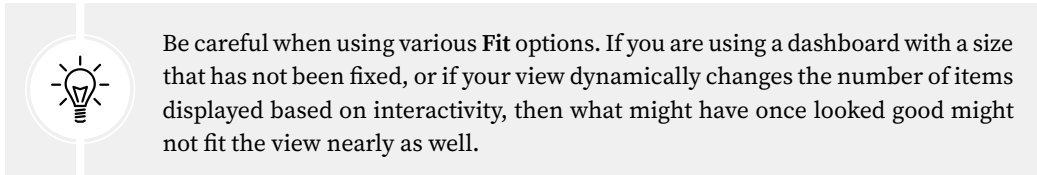
You can resize an object on the dashboard using the border. The **Grip**, labeled in Figure 1.31, allows you to move the object once it has been placed. We'll consider other options as we work with them.

Building your dashboard

With an overview of the interface, you are now ready to build a dashboard by following these steps:

1. Navigate to the **Superstore Sales** sheet. You should see a blank dashboard.
2. Successively double-click on each of the following sheets listed in the **Dashboard** section on the left: **Sales by Department**, **Sales over time**, and **Sales by Postal Code**. Notice that double-clicking on the object adds it to the layout of the dashboard.
3. Add a title to the dashboard by checking the **Show Dashboard title** at the lower left of the sidebar.
4. Select the **Sales by Department** sheet in the dashboard and click on the drop-down arrow to show the menu.

5. Select **Fit | Entire View**. The **Fit** options describe how the visualization should fill any available space.



6. Select the **Sales** size legend by clicking on it. Use the **X** option to remove the legend from the dashboard:

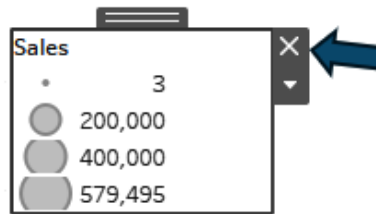


Figure 1.32: Select the legend by clicking on it, then click the X to remove it from the dashboard

7. Select the **Profit** color legend by clicking on it. Use the **Grip** to drag it and drop it under the map.
8. For each view (**Sales by Department**, **Sales by Postal Code**, and **Sales over time**), select the view by clicking on an empty area in the view. Then, click on the **Use as Filter** option to make that view an interactive filter for the dashboard:

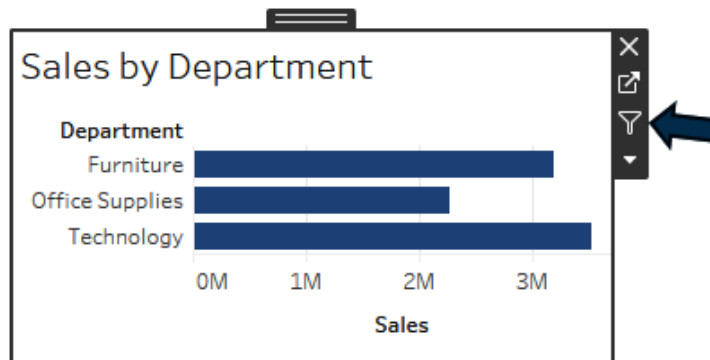
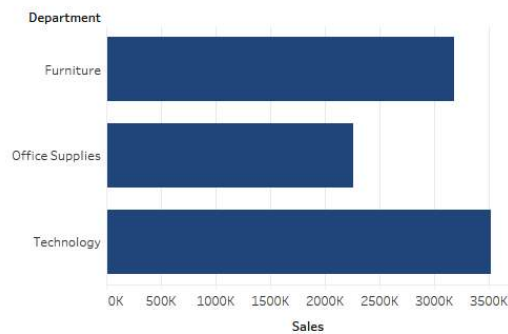


Figure 1.33: Click on the Use as Filter button to use a view as a filter in a dashboard

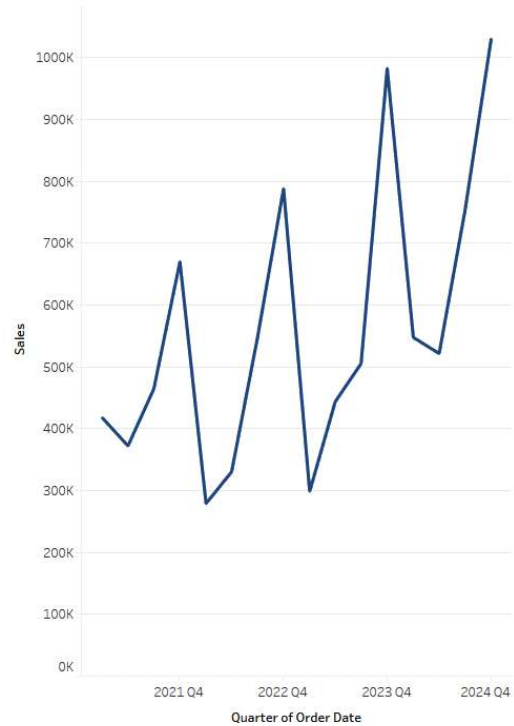
Your dashboard should look like this:

Superstore Sales

Sales by Department



Sales over time



Sales by Postal Code

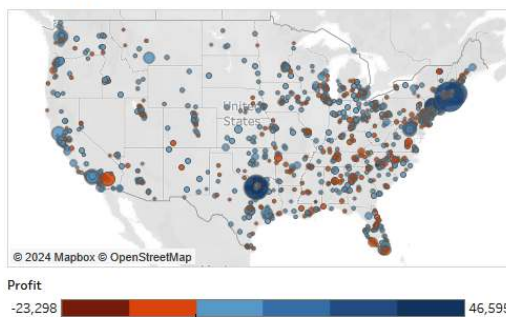


Figure 1.34: The final dashboard consisting of three views

- Take a moment to interact with your dashboard. Click on various marks, such as the bars, states, and points of the line. Notice that each selection filters the rest of the dashboard. Clicking on a selected mark will deselect it and clear the filter. Also, notice that selecting marks in multiple views causes filters to work together. For example, selecting the bar for **Furniture** in **Sales by Department** and the **2019 Q4** in **Sales over time** allows you to see all the ZIP codes that had furniture sales in the fourth quarter of 2019.

Congratulations! You have now created a dashboard that allows you to carry out interactive analysis!

As an analyst for the Superstore chain, your visualizations allowed you to explore and analyze the data. The dashboard you created can be shared with members of management, and it can be used as a tool to help them see and understand the data to make better decisions. When a manager selects the furniture department, it immediately becomes obvious that there are locations where sales are quite high, but the profit is actually very low. This may lead to decisions such as a change in marketing or a new sales focus for that location. Most likely, this will require additional analysis to determine the best course of action. In this case, Tableau will empower you to continue the cycle of discovery, analysis, and storytelling.

Summary

Tableau's visual environment allows a rapid and iterative process of exploring and analyzing data visually. You've taken your first steps toward understanding how to use the platform. You connected to data and then explored and analyzed the data using some key visualization types, such as bar charts, line charts, and geographic visualizations. Along the way, you focused on learning the techniques and understanding key concepts such as the difference between measures and dimensions and discrete and continuous fields. Finally, you put all of the pieces together to create a fully functional dashboard that allows an end user to understand your analysis and make discoveries of their own.

Congratulations! You've begun your journey with Tableau. The concepts we've covered are far more than theoretical. Dimensions and measures, discrete and continuous fields, sheets, dashboards, and stories – all the interface terminology; it all forms a solid foundation for understanding the Tableau paradigm. As you work through the rest of the book, come back to this chapter to review the concepts. They will become second nature and will help you every step of the way!

In the next chapter, we'll explore how Tableau works with data. You will be exposed to fundamental concepts and practical examples of how to connect to various data sources. Combined with the key concepts you just learned about building visualizations, you will be well equipped to move on to more advanced visualizations, deeper analysis, and telling fully interactive data stories.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



2

Connecting to Data and Tableau

Tableau offers the ability to connect to nearly any data source. It does this with a unique paradigm that leverages the power and efficiency of existing database engines (for example, Snowflake or SQL Server) or, alternatively, extracts the data locally.

We'll look at relating tables of data together using joins, blends, unions, and the object model in *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*. In this chapter, we'll focus on essential concepts of how Tableau connects to and works with data more broadly. We'll cover the following topics:

- The Tableau paradigm
- Connecting to data
- Managing data source metadata
- Working with extracts instead of live connections
- Filtering data

We'll start by gaining an understanding of the underlying paradigm of how Tableau works with data.

The Tableau paradigm

The unique and exciting experience of working with data in Tableau is a result of **Visual Query Language (VizQL)**.

VizQL was developed as a Stanford University research project, focusing on the natural ways that humans visually perceive the world and how that could be applied to data visualization. We naturally perceive differences in size, shape, spatial location, and color. VizQL allows Tableau to translate your actions, as you drag and drop fields of data in a visual environment into a query language that defines how the data encodes those visual elements. You will never need to read, write, or debug VizQL. As you drag and drop fields onto various shelves defining size, color, shape, and spatial location, Tableau will generate VizQL behind the scenes. This allows you to focus on visualizing data, not writing code!

One of the benefits of VizQL is that it provides a common way of describing how the arrangement of various fields in a view defines a query related to the data.

This common baseline can then be translated into numerous flavors of SQL, MDX, and **Tableau Query Language (TQL)**; used for extracted data). Tableau will automatically perform the translation of VizQL into a native query to be run by the source data engine.

In its simplest form, the Tableau paradigm of working with data looks like the following diagram:

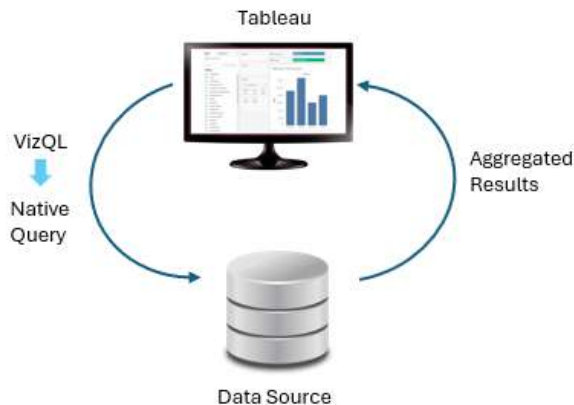


Figure 2.1: The basic Tableau paradigm for working with data

Let's look at how this paradigm works in a practical example.

The Tableau paradigm in action

Open the Chapter 02 Starter.twbx workbook, located in the \Learning Tableau\Chapter 02 directory, and navigate to the Tableau Paradigm sheet. That view was created by dropping the **Region** dimension on **Columns** and the **Sales** measure on **Rows**. Here is a screenshot:

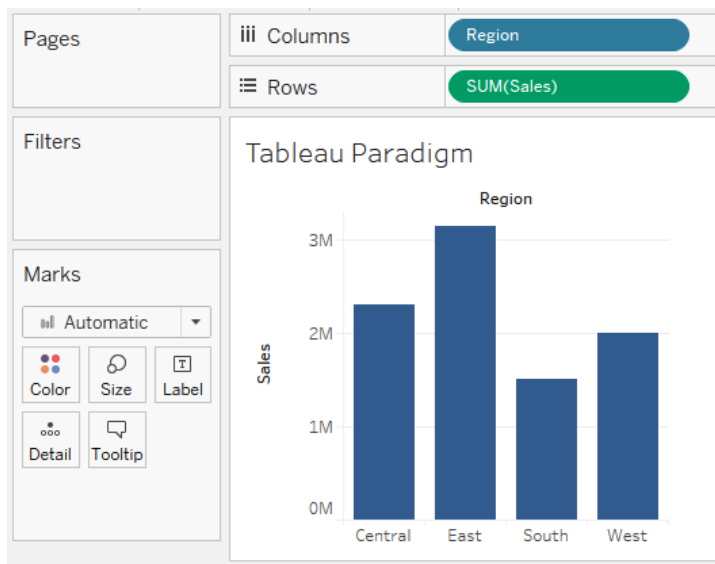


Figure 2.2: This bar chart is the result of a query that returned four aggregate rows of data

The view is defined by two fields. **Region** is the only dimension, which means it defines the level of detail in the view and slices the measure so that there will be a bar per region. **Sales** is used as a measure aggregated by summing each sale within each region. (Notice also that **Region** is discrete, resulting in column headers, while **Sales** is continuous, resulting in an axis.)

For the purpose of this example (although the principle is applicable to any data source), let's say you were connected live to a SQL Server database with the Superstore data stored in a table. When you first create what is shown in the preceding screenshot, Tableau generates a VizQL script, which is translated into a SQL script and sent to SQL Server. The SQL Server database engine evaluates the query and returns aggregated results to Tableau, which are then rendered visually.

The entire process would look something like the following diagram in Tableau's paradigm:

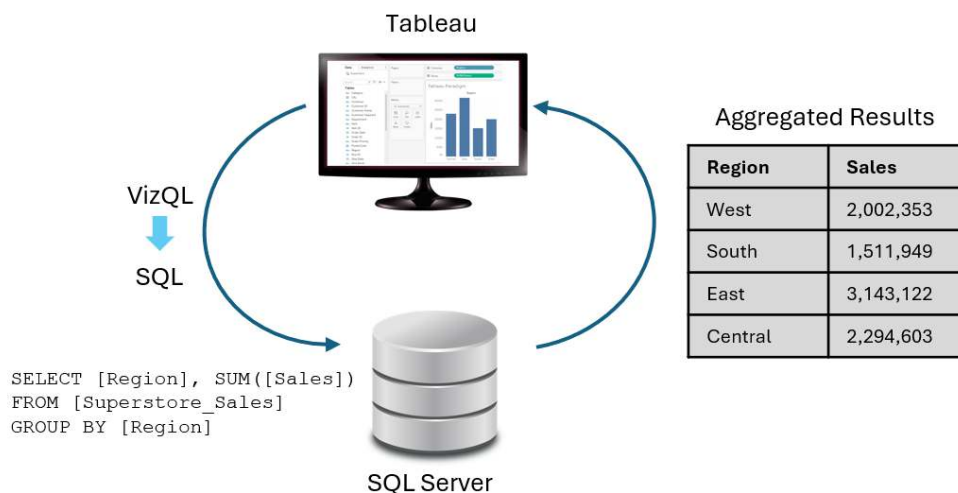


Figure 2.3: Tableau generated the bar chart in the previous image using a paradigm like this

There may have been hundreds, thousands, or even millions of rows of sales data in SQL Server. However, when SQL Server processes the query, it returns aggregate results. In this case, SQL Server returns only four aggregate rows of data to Tableau—one row for each region.

On occasion, a database administrator may want to find out what scripts are running against a certain database to debug performance issues or to determine more efficient indexing or data structures. Many databases supply profiling utilities or log execution of queries. In addition, you can find SQL or MDX generated by Tableau in the logs located in the `My Tableau Repository\Logs` directory. The community-supported Log Viewer makes it easier to traverse the logs and view live updates. It is available here: <https://github.com/tableau/tableau-log-viewer>.

You may also use Tableau's built-in **Performance Recorder** to locate the queries that have been executed. From the top menu, select **Help | Settings and Performance | Start Performance Recording**, then interact with a view, and, finally, stop the recording from the menu. Tableau will open a dashboard that will allow you to see tasks, performance, and queries that were executed during the recording session.

To see the aggregate data that Tableau used to draw the view, press *Ctrl* + *A* to select all the bars, and then right-click one of them and select **View Data...**:

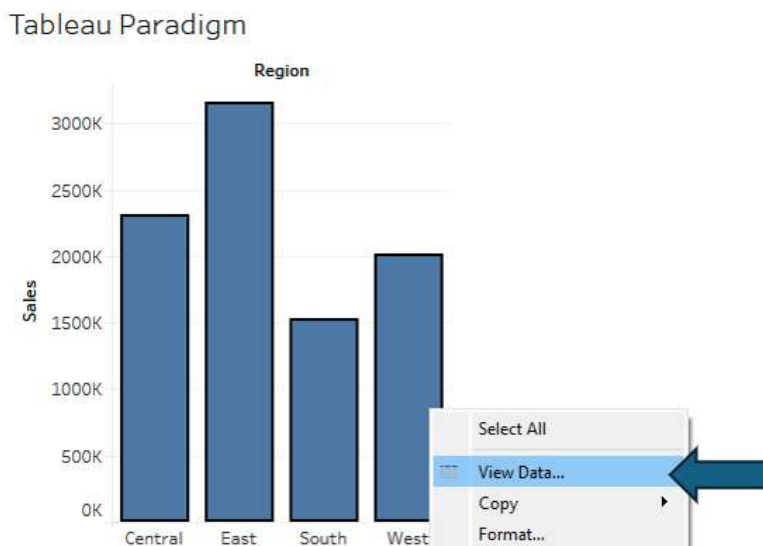


Figure 2.4: Use the View Data... tooltip option to see a summary or underlying data for a mark

This will reveal a **View Data** window:

View Data: Tableau Paradigm (4 marks)

Tabs

Summary

Full Data

Summary

4 rows 2 fields

Abc	#
Migrated Data	Migrated Data
Region	Sales
West	2,002,353
South	1,511,949
East	3,143,122
Central	2,294,603

→

ROWS

Figure 2.5: The Summary tab displays the aggregate data Tableau used to render each mark in the view

The **View Data** screen allows you to observe the data in the view. The **Summary** tab displays the aggregate-level data that was used to render the view. The **Sales** values here are the sum of sales for each region. When you click the **Full Data** tab, Tableau will query the data source to retrieve all the records that make up the aggregate records.

In this case, there are 9,426 underlying records, as indicated on the status bar in the following screenshot:

View Data: Tableau Paradigm (4 marks) ✕

Tabs < Summary Full Data	Full Data 9,426 rows 26 fields Show Fields Download				
	Abc Migrated Data	# Migrated Data	Abc Migrated Data	⊕ Migrated Data	Abc Migrated Data
	Region	Sales	Category	City	Container
	East	7	Scissors, Rulers and Trimmers	Washington	Small Pack
	East	312	Storage & Organization	Oxford	Medium Box
	Central	2,635	Telephones and Communicat...	Irving	Small Box
	Central	281	Paper	Irving	Small Box
10,000 → rows ⚙	East	2,983	Bookcases	Oxford	Jumbo Box
	East	2,247	Computer Peripherals	Oxford	Small Pack

Figure 2.6: The Full Data tab reveals the row-level data in the database

Tableau did not need 9,426 records to draw the view and did not request them from the data source until the **Full Data** tab was clicked.

Database engines are optimized to perform aggregations on data. Typically, these database engines are also located on powerful servers. Tableau leverages the optimization and power of the underlying data source. In this way, Tableau can visualize massive datasets with relatively little local processing of the data.

Additionally, Tableau will only query the data source when you make changes requiring a new query or a view refresh. Otherwise, for improved performance, it will use the aggregate results stored in a local cache, as illustrated here:

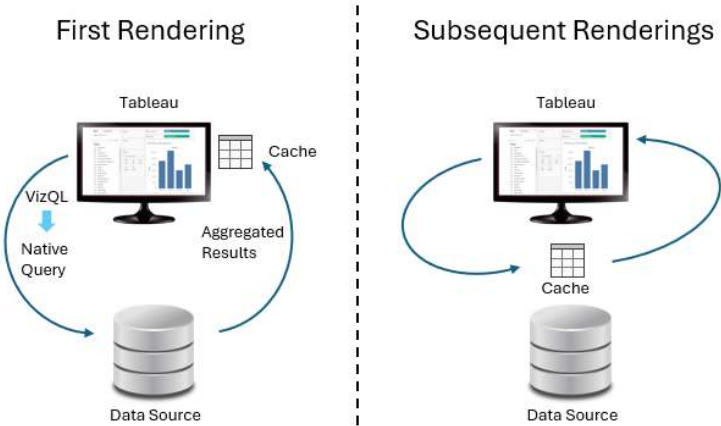


Figure 2.7: The first rendering with a given set of fields queries the data source directly. Subsequent renderings will query the cache, even if the same fields are re-arranged in the view

In the preceding example (shown in *Figure 2.2*), the query with **Region** as a dimension and the sum of **Sales** as a measure will only be issued once to the data source. When the four rows of aggregated results are returned, they are stored in the cache. After the initial rendering, if you were to move **Region** to another visual encoding shelf, such as **Color**, or **Sales** to a different visual encoding shelf, such as **Size**, then Tableau would retrieve the aggregated rows from the cache and simply re-render the view.

You can force Tableau to bypass the cache and refresh the data from a data source by pressing *F5* or selecting your data source from the **Data** menu and selecting **Refresh**. Do this any time you want a view to reflect the most recent changes in a live data source.

If you were to introduce new fields into the view that did not have cached results, Tableau would send a new query to the data source, retrieve the aggregated results, and add those results to the cache.

With an understanding of the overall Tableau paradigm, let's consider some specific examples of connecting to data.

Connecting to data

There is virtually no limit to the data that Tableau can visualize! Tableau can connect to files, relational databases, cloud-based data sources, and on-premises database engines. Almost every new version of Tableau adds new native connectors. Tableau continues to add native connectors for cloud-based data. The **web data connector** allows you to write a connector for any online data you wish to retrieve. The **Tableau Hyper API** allows you to programmatically read and write extracts of data, enabling you to access data from any source and write it to a native Tableau format. Additionally, for any database without a built-in connection, Tableau gives you the ability to use a generic **ODBC** connection.

You may have multiple data sources in the same workbook. Each source will show up under the **Data** tab on the left sidebar.

Although the terms “data connection” and “data source” are often used interchangeably, it is helpful to make a distinction. A **connection** technically refers to the connection made to data in a single location, such as tables in a single database or files of the same type in the same directory structure. A **data source** may contain more than one connection that can be joined together, such as a table in SQL Server joined to tables in a Snowflake database that are joined to an Excel table. You can think about it this way: a Tableau workbook may contain one or more data sources and each data source may contain one or more connections. We'll maintain this distinction throughout the book.

This section will focus on a few practical examples of connecting to various sources of data. There's no way to cover every possible type of connection but we will cover several that are representative of others. You may or may not have access to some of the databases or cloud services in the following examples. Don't worry if you aren't able to follow each example. Merely observe the similarities. Every time, you'll follow a similar pattern: point Tableau to the location of the data, enter the necessary credentials, and configure a few options.

Connecting to data in a file

File-based data includes all sources of data where the data is stored in a file. File-based data sources include the following:

- **Microsoft Excel:** An .xls, .xlsx, or .xlsm spreadsheet created in Excel. Multiple Excel sheets or sub-tables may be joined or unioned together in a single connection.
- **Text file:** A delimited text file, most commonly .txt, .csv, or .tab. Multiple text files in a single directory may be joined or unioned together in a single connection.
- **Spatial file:** A wide variety of spatial formats are supported, such as .kml, .shp, .tab, .mif, spatial JSON, and ESRI database files. These formats contain spatial objects that can be rendered by Tableau.
- **Extracts:** A .hyper or .tde file containing data that was extracted from an original source.
- **Microsoft Access:** An .mdb or .accdb database file created in Access.
- **Adobe PDF:** A .pdf file that may contain tables of data that can be parsed by Tableau.
- **Statistical file:** A .sav, .sas7bdat, .rda, or .rdata file generated by statistical tools, such as SAS or R.
- **JSON file:** A .json file that contains data in JSON format.
- **Local cube file:** A .cub file that contains multi-dimensional data. These files are typically exported from **Online Analytical Processing (OLAP)** databases.

In addition, you may also select a Tableau workbook (.twb or .twbx) to import connections saved in that workbook.

Follow this example to see a connection to an Excel file:

1. Navigate to the **Connect to Excel** sheet in the *Chapter 02 Starter.twbx* workbook.
2. From the menu, select **Data | New Data Source** and select **Microsoft Excel** from the list of possible connections.
3. In the open dialog, open the *Superstore.xlsx* file from the `\Learning Tableau\Chapter 02` directory. Tableau will open the **Data Source** screen. You should see the two sheets of the Excel document listed on the left.
4. Double-click the **Orders** sheet and then the **Returns** sheet. Tableau will display relationships on the lower left (number 8 in *Figure 2.8*). We'll cover relationships in depth in *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*. For now, accept the defaults by closing the dialog.

Your **Data Source** screen should look similar to the following screenshot:

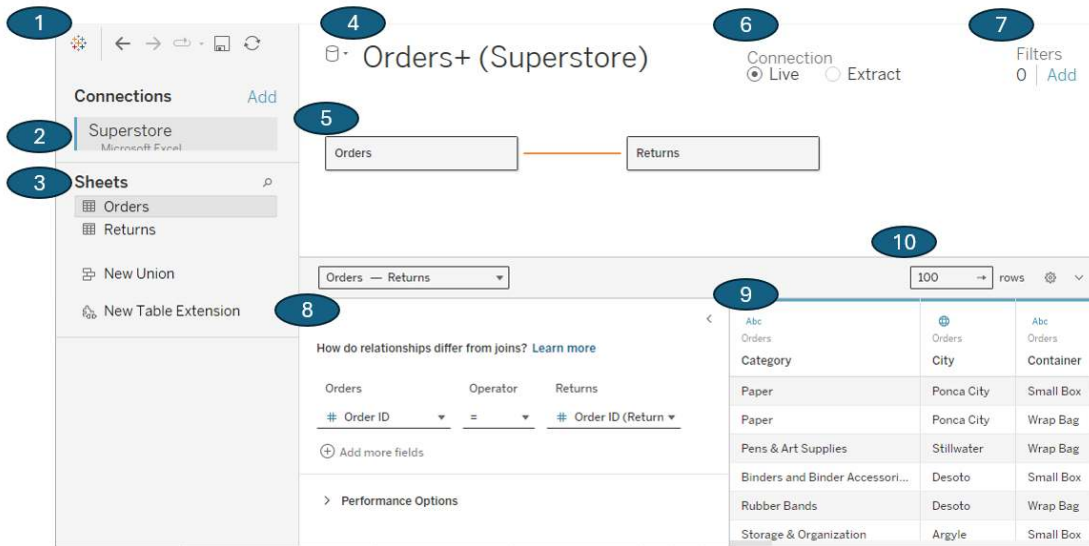


Figure 2.8: The Data Source screen with two objects (*Orders* and *Returns*)

Take some time to familiarize yourself with the **Data Source** screen interface, which has the following features (numbered in the preceding screenshot):

1. **Toolbar:** The toolbar has a few familiar controls, including undo, redo, and save. It also includes the option to refresh the current data source.
2. **Connections:** All the connections in the current data source. Click **Add** to add a new connection to the current data source. This allows you to join data across different connection types. Each connection will be color-coded so that you can distinguish what data is coming from which connection.
3. **Sheets (or Tables):** Lists all the tables of data available for a given connection. This includes sheets, sub-tables, and named ranges for Excel; tables, views, and stored procedures for relational databases; and other connection-dependent options, such as **New Union** or custom SQL.
4. **Data source name:** This is the name of the currently selected data source. You may select a different data source using the drop-down arrow next to the database icon. You may click the name of the data source to edit it.
5. **Data model canvas:** Drop sheets and tables from the left into this area to add them to the data model. You may add additional tables by dragging and dropping or double-clicking them. Each will be added as an object to the data model. You may also add tables as unions or double-click an object on the canvas to edit the underlying tables and joins. We'll cover the details extensively in *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*. For now, simply note that *Orders* and *Returns* are related together by the order ID.
6. **Live or Extract options:** For many data sources, you may choose whether you would like to have a live connection or an extracted connection. We'll look at these in further detail in the *Working with extracts instead of live connections* section later in this chapter.

7. **Data source filters:** You may add filters to the data source. These will be applied at the data source level, and thus to all views of the data using this data source in the workbook.
8. **Object dropdown:** Here, you can select from a list of objects that are present in your data connection, including individual tables and relationships. Based on your selections, you may see or even modify certain aspects of the object. Here, for example, the relationship between **Orders** and **Returns** has been selected. This reveals some options beneath that allow you to define the relationship.
9. **Preview pane/metadata view:** Depending on your selection in the options, this space displays either a preview of data or a list of all fields with additional metadata. Notice that these views give you a wide array of options, such as changing data types, hiding or renaming fields, and applying various data transformation functions. We'll consider some of these options in this and later chapters.
10. **Preview pane options:** These options allow you to specify whether you'd like to see a preview of the data or a list of metadata, and how you would like to preview the data (examples include alias values, hidden fields shown, and how many rows you'd like to preview).

Once you have created and configured your data source, you may select any tab in the Tableau workbook to start using the connection.

Conclude this exercise with the following steps:

1. Click the data source name to edit the text and rename the data source to **Orders and Returns**.
2. Navigate to **Connect to Excel**.
3. Use the **Orders and Returns** data source to create a bar chart showing **Returns (Count)** by **Return Reason**. Your view should look like the following screenshot:

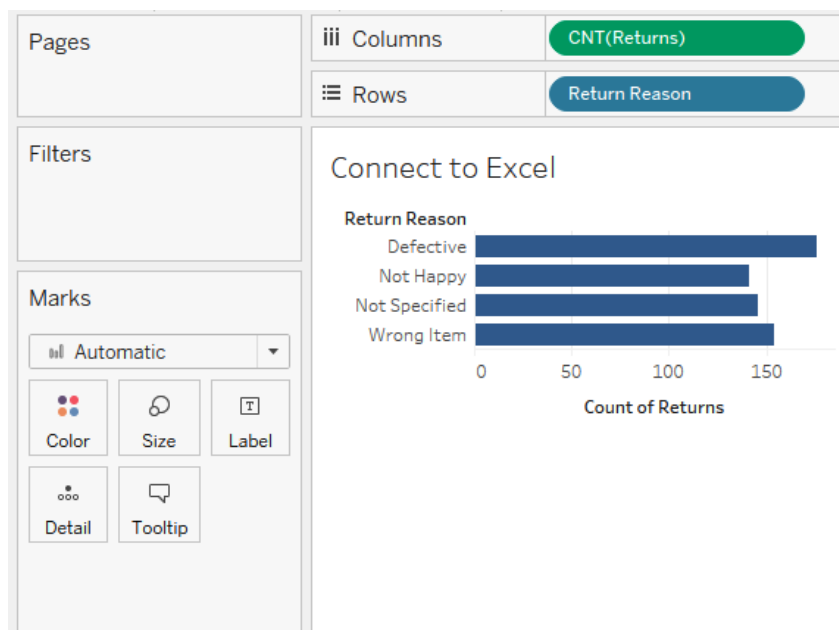


Figure 2.9: The number of returns by return reason

If you need to edit the connection at any time, you may use one of the following methods:

- Select **Data** from the menu, locate your connection, and then select **Edit Data Source...**
- Right-click any data source under the **Data** tab on the left sidebar and select **Edit Data Source...**
- Click the **Data Source** tab in the lower-left corner

Connecting to data in a database

Database servers, such as SQL Server, Snowflake, Vertica, and Oracle, host data on one or more server machines and use powerful database engines to store, aggregate, sort, and serve data based on queries from client applications. Some of these databases may be hosted on your network (often referred to as “on-premises” or “on-prem”) or may be distributed in the cloud. Tableau can leverage the capabilities of these servers to retrieve data for visualization and analysis. Alternatively, data can be extracted from these sources and stored in an extract.

As an example of connecting to a server data source, we’ll consider connecting to Snowflake. If you have access to a server-based data source, you may wish to create a new data source and explore the details. However, this specific example is not included in the workbook in this chapter.

As soon as the Snowflake connection is selected, the interface displays options for some initial configuration, as follows:

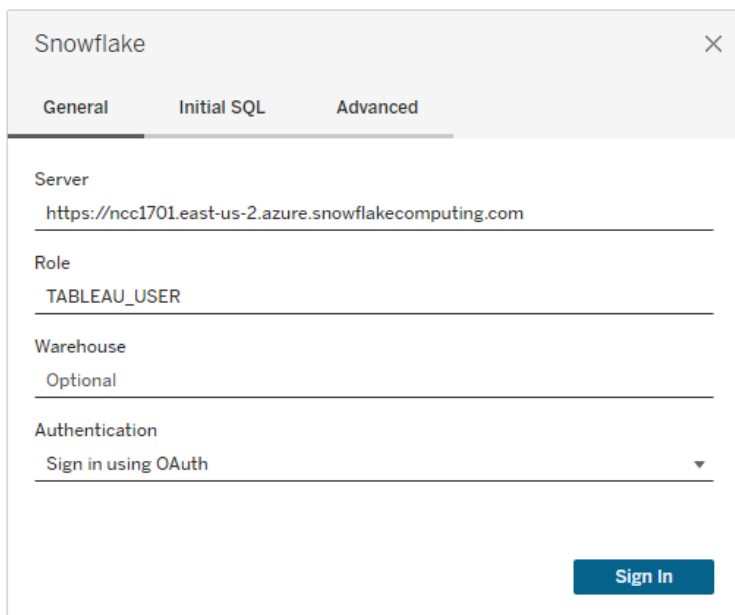
The image shows a 'Snowflake' connection editor dialog box. It has a title bar with a close button (X) in the top right corner. Below the title bar are three tabs: 'General', 'Initial SQL', and 'Advanced'. The 'General' tab is currently selected. The 'General' tab contains four labeled input fields: 'Server' with the value 'https://ncc1701.east-us-2.azure.snowflakecomputing.com', 'Role' with the value 'TABLEAU_USER', 'Warehouse' with the value 'Optional', and 'Authentication' with a dropdown menu showing 'Sign in using OAuth'. A blue 'Sign In' button is located at the bottom right of the dialog box.

Figure 2.10: The connection editor for Snowflake

A connection to Snowflake requires the **Server** name, as well as **Role**, **Warehouse**, and **Authentication** information.

Using the **Initial SQL** and **Advanced** tabs, you may specify SQL to run at the time of the initial connection, as well as setting advanced connection settings, such as custom driver parameters.

In order to maintain high standards of security, Tableau will not save a password as part of a data source connection in a workbook. This means that if you share a workbook using a live connection with someone else, they will need to have credentials to access the data. This also means that when you first open the workbook, you will need to re-enter your password for any connections requiring a password. (Options for embedding passwords or credentials do exist when publishing a workbook or data source to Tableau Server.)

Once you click the **Sign In** button, you will see a screen that is very similar to the connection screen you saw for Excel. The main difference is on the left, where you have an option for selecting a **Warehouse**, **Database**, **Schema**, and objects, as shown in the following screenshot:

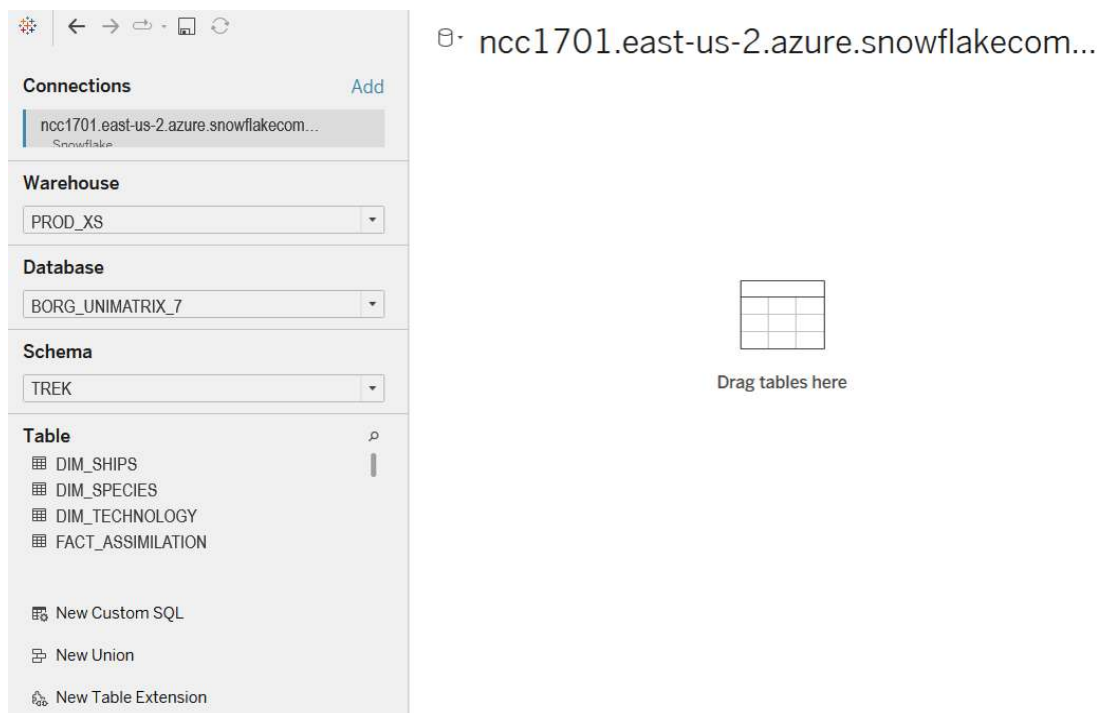


Figure 2.11: Once connected to a database, Tableau will display tables, views, and stored procedures as options to add to the object model

Once you've selected a warehouse, database, and schema, you will see the following as options on the screen:

- **Table:** This shows any data tables or views contained in the selected database.
- **New Custom SQL:** You may write your own custom SQL scripts and add them as tables. You may join these as you would any other table or view.
- **New Union:** You may union together tables in the database. Tableau will match fields based on name and data type, and you may additionally merge fields as needed.
- **New Table Extension:** Table extensions allow you to add a custom script (in a language such as Python). This gives you the ability to greatly enhance the data with a variety of transformations.

Once you have finished configuring the connection, click a tab for any sheet to begin visualizing the data.

Connecting to other data in the cloud

Cloud-based data sources vary widely. Some have much the same look and feel as on-premises database engines, such as **Amazon RDS**, **Microsoft Azure SQL Database**, and **Google Cloud SQL**, while others are cloud-based file storage solutions, such as **OneDrive**, **Google Drive**, **Dropbox**, and **Amazon S3**.

Here, we'll consider an example of Google Sheets, which allows users to create and maintain spreadsheets of data online. Sheets are saved in Google Drive and may be shared and collaborated on by many different users. Here, we'll walk through an example of connecting to a sheet that is shared via a link.

To follow the example, you'll need a free Google account. With your credentials, follow these steps:

1. Click the **Add new data source** button on the toolbar, as shown here:



Figure 2.12: The Add new data source button

2. Select **Google Drive** from the list of possible data sources and authenticate using your credentials.
3. On the **Connect to Google Drive** screen, under the **My Drive** tab, enter the following shared link into the search box (for convenience, it is included in the Chapter 02 Starter workbook in the **Connect to Google Drive** tab, and may be copied and pasted): https://docs.google.com/spreadsheets/d/1YUZ1UnryuCX7cwbfnPtGuzx0G4yth7i-m9Jrkce9_PE/edit?usp=sharing.

After searching for the link, Tableau will show the Superstore Google Sheets document, like this:



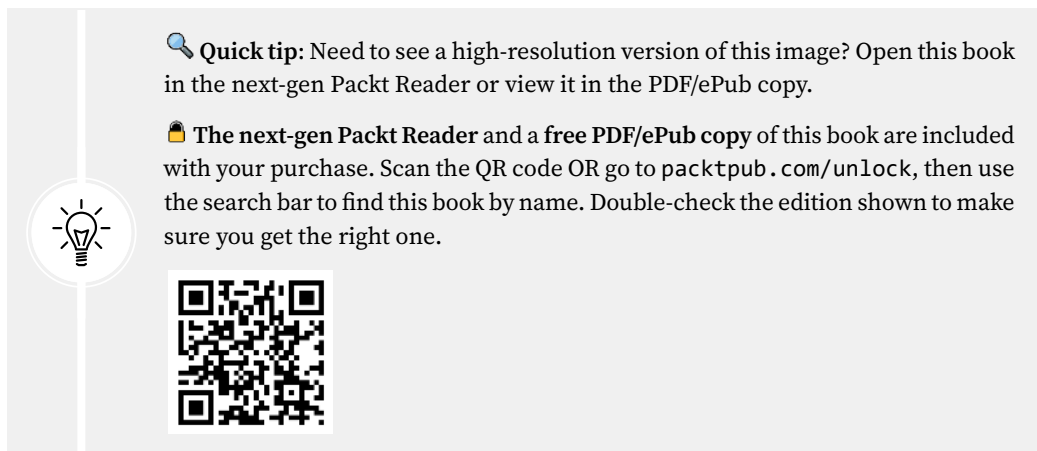
Figure 2.13: When connecting, you may browse the contents of Google Drive. In this example, paste the shared URL into the search box

Complete connecting to the Google Sheets document by following these steps:

1. Select the resulting Superstore sheet in the list and then click the **Connect** button. You should now see the **Data Source** screen.
2. Drag the **Orders** sheet onto the canvas.
3. Click the **Data Source** name to rename it Superstore (Google Sheets):



Figure 2.14: Renaming a data source



4. For the purpose of this example, switch the connection option from **Live** to **Extract**. When connecting to your own Google Sheets data, you may choose either **Live** or **Extract**:



Figure 2.15: Switch between Live and Extract, edit the extract options, and add filters

5. Click the tab for the **Connect to Google Sheets** sheet.

- Double-click State to create a geographic visualization. Add **Profit** to the **Color** and **Label**, which will result in a visualization like this:

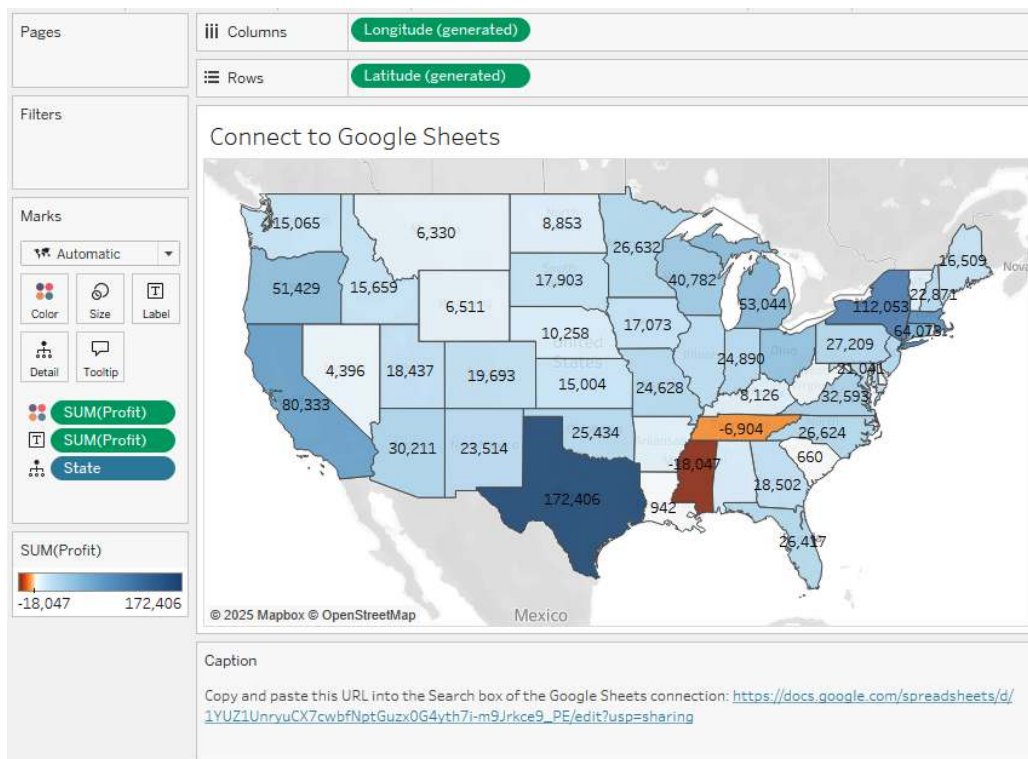


Figure 2.16: The filled map demonstrates the ability to connect to a cloud-based data source

If your location is outside the United States, you may need to change your regional settings for Tableau to properly show the states on the map. Use the menu and select **File | Workbook Locale | More**, and select **English (United States)**.

Now that we've seen a few specific examples of connecting to data, let's consider some shortcuts and how to manage our data sources.

Shortcuts for connecting to data

You can make certain connections very quickly. These options will allow you to begin analyzing more quickly:

- Paste data from the clipboard. If you have copied data in your system's clipboard from any source (for example, a spreadsheet, a table on a web page, or a text file), you can then paste the data directly into Tableau. This can be done using **Ctrl + V**, or **Data | Paste Data** from the menu. The data will be stored as a file and you will be alerted of its location when you save the workbook.

- Select **File | Open** from the menu. This will allow you to open any data file that Tableau supports, such as text files, Excel files, Access files (not available on macOS), spatial files, statistical files, JSON, and even offline cube (.cub) files.
- Drag and drop a file from Windows Explorer or Finder onto the Tableau workspace. Any valid file-based data source can be dropped onto the Tableau workspace, or even the Tableau short-cut on your desktop or taskbar.
- Duplicate an existing data source. You can duplicate an existing data source by right-clicking and selecting **Duplicate**.

These shortcuts provide a quick way of analyzing the data you need. Let's turn our attention to managing data sources.

Managing data source metadata

Data sources in Tableau store information about the connection(s). In addition to the connection itself (for example, database server name, database, and/or filenames), the data source also contains information about all the fields available (such as field name, data type, default format, comments, and aliases). Often, this *data about the data* is referred to as **metadata**.

Right-clicking a field in the data pane reveals a menu of metadata options. Some of these options will be demonstrated in a later exercise; others will be explained throughout the book. These are some of the options available via right-clicking:

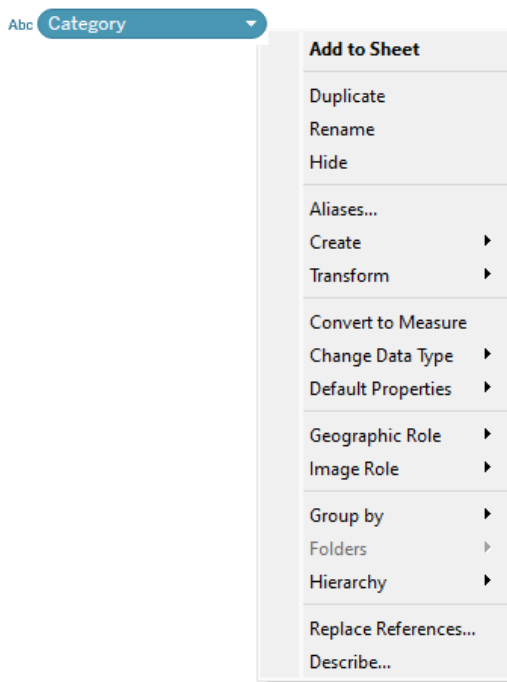


Figure 2.17: Options on the drop-down menu of a field in the Data pane

Although the options may vary based on which field you select, in general, you'll find options like these:

- Renaming the field.
- Hiding the field.
- Changing aliases for values of a non-date dimension.
- Creating calculated fields, groups, sets, bins, or parameters.
- Splitting the field.
- Changing the default use of a date or numeric field to either discrete or continuous.
- Redefining the field as a dimension or a measure.
- Changing the data type of the field.
- Assigning a geographic role to the field.
- Assigning an image role to the field.
- Changing defaults for how a field is displayed in a visualization, such as the default colors and shapes, number or date format, sort order (for dimensions), or type of aggregation (for measures).
- Adding a default comment for a field. Any comments you type in for a field will be shown as a tooltip when hovering over a field in the data pane, or shown as part of the description when **Describe...** is selected from the menu.
- Adding or removing the field from a hierarchy.

Metadata options that relate to the visual display of the field, such as default sort order or default number format, define the overall default for a field. However, you can override the defaults in any individual view by right-clicking the active field on the shelf and selecting the desired options.

To see how this works, use the filled map view of Profit by State that you created in the **Connect to Google Sheets** view. If you did not create this view, you may use the **Orders and Returns** data source, though the resulting view will be slightly different. With the filled map in front of you, follow these steps:

1. Right-click the **Profit** field in the data pane and select **Default Properties | Number Format...**. The resulting dialog gives you many options for numeric format.
2. Set the number format to **Currency (Custom)** with **0 Decimal places** and the **Display Units in Thousands (K)**, before clicking **OK**:

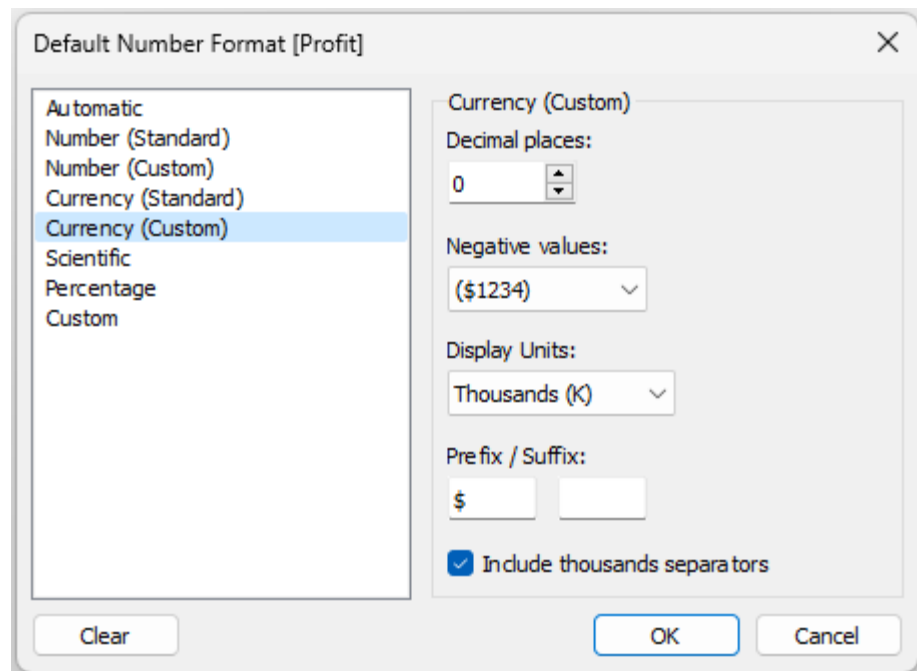


Figure 2.18: Editing the default number format of a field. The exact format of currency will depend on your regional settings

3. After clicking **OK**, you should notice that the labels on the map have updated to include currency notation.
4. Right-click the **Profit** field again and select **Default properties | Color....** The resulting dialog gives you an option to select and customize the default color encoding of the **Profit** field. Experiment with various palettes and settings. Notice that every time you click the **Apply** button, the visualization updates:

Diverging palettes (palettes that blend from one color to another) work particularly well for fields such as **Profit**, which can have negative and positive values. The default center of 0 allows you to easily tell what values are positive or negative based on the color shown.

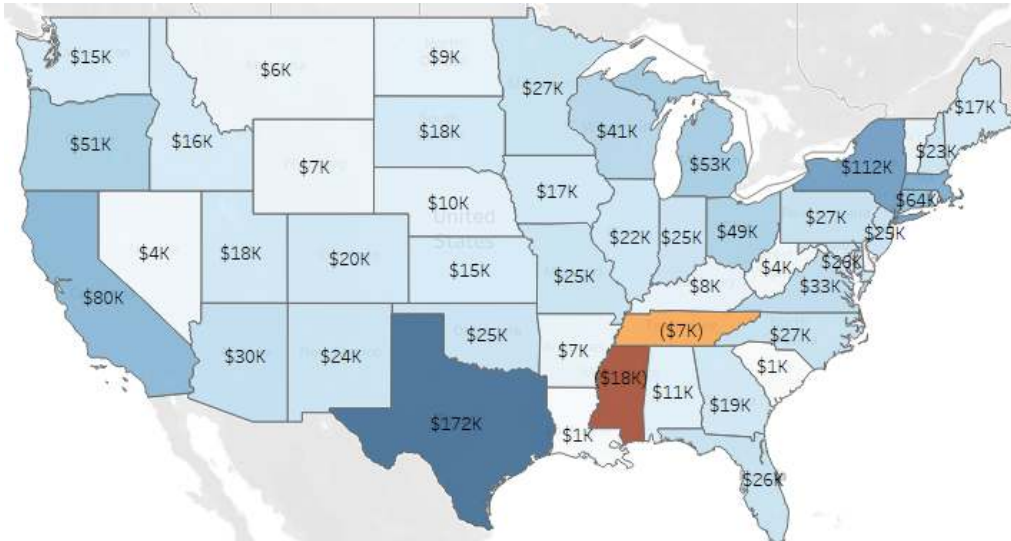


Figure 2.19: Customizing color

Because you have set the default format for the field at the data-source level, any additional views you create using **Profit** will include the default formatting you specified.

Consider using color-blind-safe colors in your visualizations. Orange and blue are usually considered a color-blind-safe alternative to red and green. Tableau also includes a discrete color-blind-safe palette. Additionally, consider adjusting the intensity of the colors, using labels or different visualizations to make your visualizations more accessible.

Next, let's take some time to consider the use of extracts as an alternative to live connections, and how to create, use, and manage them.

Working with extracts instead of live connections

Nearly all data sources allow the option of either connecting live or extracting the data. A few cloud-based data sources require an extract. Conversely, OLAP data sources cannot be extracted and require live connections.

Extracts extend the way in which Tableau works with data. Consider the following diagram:

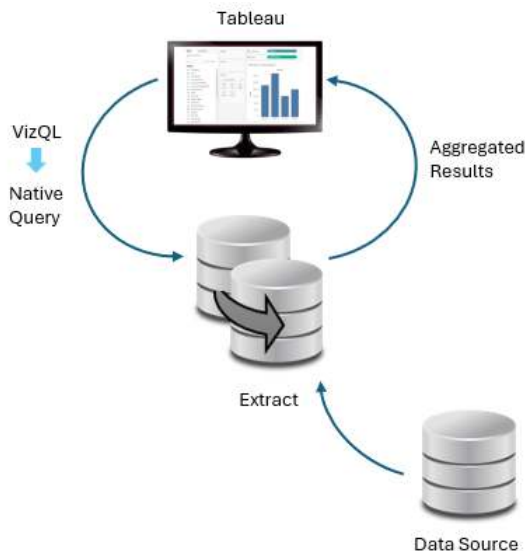


Figure 2.20: Data from the original data source is extracted into a self-contained snapshot of the data

When using a live connection, Tableau issues queries directly to the data source (or uses data in the cache, if possible). When you extract the data, Tableau pulls some or all the data from the original source and stores it in a Hyper extract file with the extension `.hyper`.

The fundamental paradigm of how Tableau works with data does not change, but you'll notice that Tableau is now querying and getting results from the extract. Data can be retrieved from the source again to refresh the extract. Thus, each extract is a snapshot of the data source at the time of the latest refresh. Extracts offer the benefit of being portable and extremely efficient.

Creating extracts

Extracts can be created in multiple ways, as follows:

- Select **Extract** on the **Data Source** screen, as follows. The **Edit** link will allow you to configure the extract:

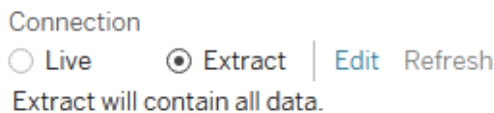


Figure 2.21: Select either Live or Extract for a connection and configure options for the extract by clicking Edit

- Select the data source from the **Data** menu, or right-click the data source on the data pane and select **Extract Data....** You will be given a chance to set configuration options for the extract, as demonstrated in the following screenshot:

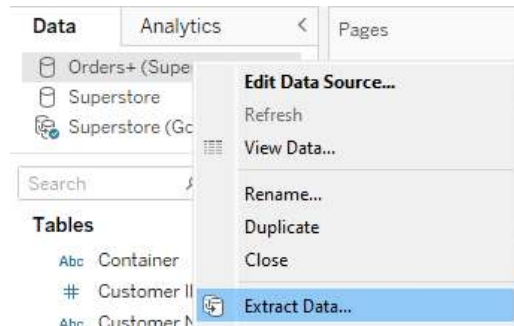


Figure 2.22: The Extract Data... option

- Developers may create an extract using the Tableau Hyper API. This API allows you to use Python, Java, C++, or C#/NET to programmatically read and write Hyper extracts. The details of this approach are beyond the scope of this book, but documentation is readily available on Tableau's website at <https://tableau.github.io/hyper-db/docs/>.
- Certain tools, such as **Alteryx** or **Tableau Prep**, can output Tableau extracts. We'll consider Tableau Prep output options in *Chapter 15, Taming Data with Tableau Prep*.

You'll have quite a few options for configuring an extract. To edit these options, select **Extract** and then **Edit...** on the **Data Source** screen or **Extract data...** from the context menu of a connection in the data pane. When you configure an extract, you will be prompted to select certain options, as shown here:

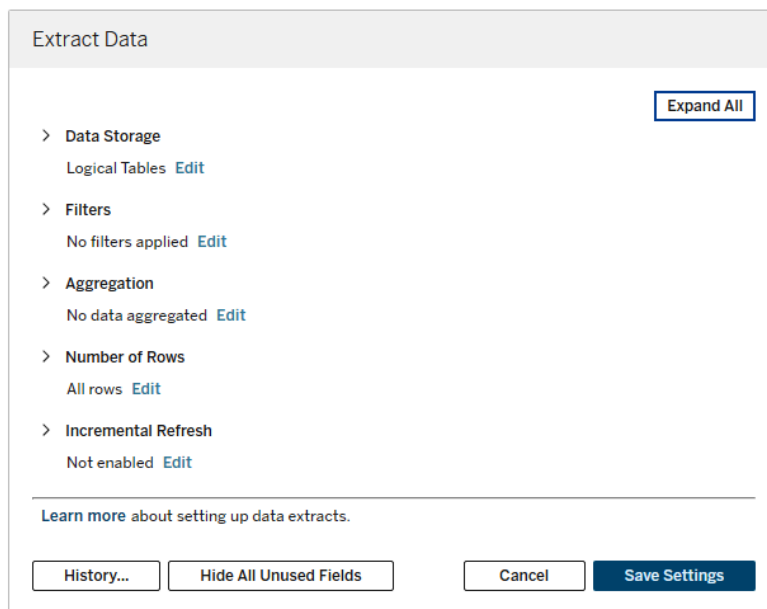


Figure 2.23: The Extract Data dialog gives quite a few options for how to configure the extract

You have a great deal of control when configuring an extract. Here are the various options, and the impact your choices will have on performance and flexibility:

- Depending on the data source and object model you’ve created, you may select between **Logical Tables** and **Physical Tables**. We’ll explore the details in *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*.
- You may optionally add extract **Filters**, which limit the extract to a subset of the original source.
- You may choose to create an **Aggregate** extract by expanding the option and selecting the option to aggregate data for visible dimensions and, optionally, to a specified date level, such as year or month.

Visible fields are those that are shown in the data pane. You may hide a field from the **Data Source** screen or from the data pane by right-clicking a field and selecting **Hide**. This option will be disabled if the field is used in any view in the workbook. Hidden fields are not available to be used in a view, and are not included in an extract as long as they are hidden prior to creating or optimizing the extract.

- You may select to limit the extract to a specified **Number of Rows**. This can allow for a smaller subset of data while you work to build visuals and dashboards.
- You may enable **Incremental Refresh**. By default, Tableau will replace the contents of an extract when refreshing. But with an incremental refresh, you may specify that a table should have rows added based on a field. New records where the field value is greater than the maximum value currently in the extract will be added. For example, you might select a table and the field **Row ID**. When the incremental refresh is run, any record with a row ID higher than the highest value of the previous extract refresh would be included in the next incremental refresh.

Incremental refreshes can be a great way of dealing with large volumes of data that grow over time. However, use incremental refreshes with care, because the incremental refresh will only add new rows of data based on the field you specify. You won’t get changes to existing rows, nor will rows be removed if they were deleted at the source. You will also miss any new rows if the value for the incremental field is less than the maximum value in the existing extract. If either of these scenarios is possible in your data source, you will likely want to avoid using incremental refreshes. Many times, when incremental refreshes are used, a less-frequent full refresh will be used to ensure data integrity. For example, daily incremental refreshes might be supplemented with a full weekly refresh.

Now that we’ve considered how to create and configure extracts, let’s turn our attention to using them.

Using extracts

Any data source that is using an extract will have a distinctive icon that indicates the data has been pulled from an original source into an extract, as shown in the following screenshot:

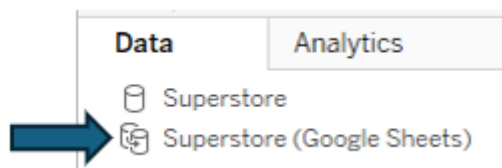


Figure 2.24: The icon next to a data source indicates whether it is extracted

The second data connection in *Figure 2.24* is extracted, while the first is not. After an extract has been created, you may choose to use the extract or not. When you right-click a data source (or **Data** from the menu and then the data source), you will see the following menu options:

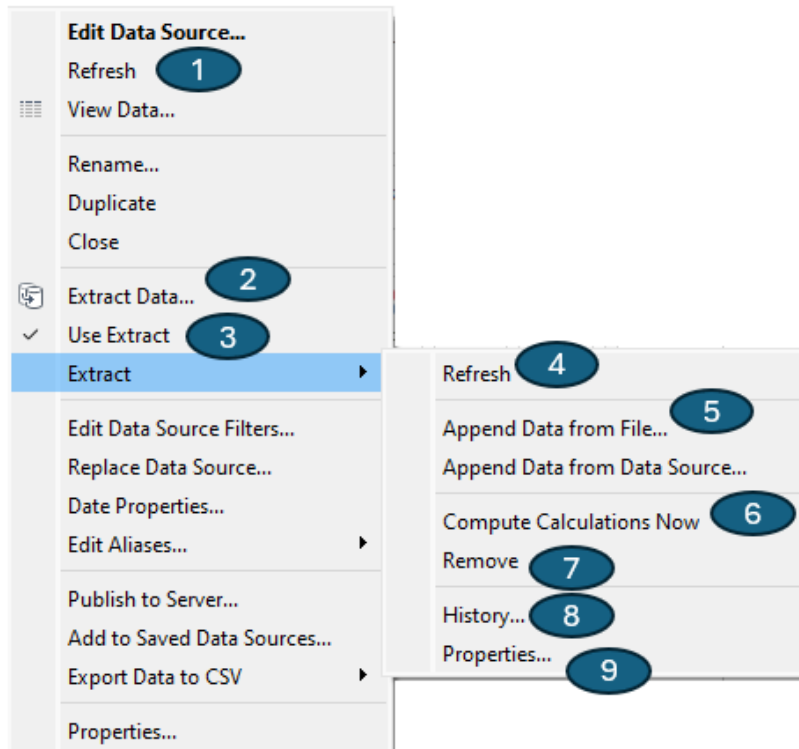


Figure 2.25: The context menu for a data connection in the Data pane with Extract options numbered

Let's cover them in more detail:

1. **Refresh:** The **Refresh** option under the data source simply tells Tableau to refresh the local cache of data. With a live data source, this would re-query the underlying data. With an extracted source, the cache is cleared and the extract is re-queried, but this **Refresh** option does not update the extract from the original source. To do that, use **Refresh** under the **Extract** sub-menu (see number 4 in this list).
2. **Extract Data...:** This creates a new extract from the data source (replacing an existing extract if it exists).
3. **Use Extract:** This option is enabled if there is an extract for a given data source. Unchecking the option will tell Tableau to use a live connection instead of the extract. The extract will not be removed and may be used again by checking this option at any time. If the original data source is not available in this workbook, Tableau will ask where to find it or to regenerate it.
4. **Refresh:** This **Refresh** option refreshes the extract with data from the original source. It does not optimize the extract for some changes you make (such as hiding fields or creating new calculations).

5. **Append Data from File... or Append Data from Data Source...:** These options allow you to append additional files or data sources to an existing extract, provided they have the same exact data structure as the original source. This adds rows to your existing extract; it will not add new columns.
6. **Compute Calculations Now:** This will restructure the extract, based on changes you've made since originally creating the extract, to make it as efficient as possible. Certain calculated fields may be *materialized* (that is, calculated once so that the resulting value can be stored) and newly hidden columns or deleted calculations will be removed from the extract.
7. **Remove:** This removes the definition of the extract, optionally deletes the extract file, and resumes a live connection to the original data source.
8. **History...:** This allows you to view the history of the extract and refreshes.
9. **Properties...:** This enables you to view the properties of the extract, such as the location, underlying source, filters, and row limits.

We considered how to create extracts, along with the performance ramifications of using extracts, in the *Working with extracts instead of live connections* section.

Performance

Depending on scale and volume, both Hyper extracts may perform faster than most traditional live database connections. The incredible performance of Tableau extracts is based on several factors, including the following:

- Extracts make use of a hybrid of **Online Transactional Processing (OLTP)** and **OLAP** models and the engine determines the optimal query.
- Extracts leverage columnar storage so that duplicate values in a field are not stored for every record. This makes extracts smaller and more efficient than data sources that store values for every record.
- Extracts are structured so they can be loaded quickly into memory without additional processing and moved between memory and disk storage, so the size is not limited to the amount of RAM available, but RAM is efficiently used to boost performance.
- Many calculated fields are materialized in the extract. The pre-calculated value stored in the extract can often be read faster than executing the calculation every time the query is executed. Hyper extracts extend this by potentially materializing many aggregations.

You may choose to use extracts to increase performance over traditional databases. To maximize your performance gain, consider the following actions:

- Prior to creating the extract, hide unused fields. If you have created all desired visualizations, you can click the **Hide Unused Fields** button on the **Extract** dialog to hide all fields not used in any view or calculation.
- If possible, use a subset of data from the original source. For example, if you have historical data for the last 10 years but only need the last 2 years for analysis, then filter the extract by the Date field.

- Optimize an extract after creating or editing calculated fields or deleting or hiding fields.
- Store extracts on solid-state drives.

Although performance is one major reason to consider using extracts, there are other factors, which we will now consider.

Portability and security

Let's say that your data is hosted on a database server accessible only from inside your office network. Normally, you'd have to be onsite or using a VPN to work with the data. Even cloud-based data sources require an internet connection. With an extract, you can take the data with you and work offline.

An extract file contains data extracted from the source. When you save a workbook, you may save it as a Tableau workbook (.twb) file or a Tableau packaged workbook (.twbx) file. Let's consider the difference:

- A Tableau workbook (.twb) contains definitions for all the connections, fields, visualizations, and dashboards, but does not contain any data or external files, such as images. A Tableau workbook can be edited in Tableau Desktop and published to Tableau Server. A .twb file is XML and can be viewed and even edited (at the user's discretion!) in an XML viewer or text editor.
- A Tableau packaged workbook (.twbx) contains everything in a .twb file but also includes extracts and external files that are packaged together in a single file with the workbook. A packaged workbook using extracts can be opened with Tableau Desktop or Tableau Reader, and published to Tableau Public or Tableau Online.

A packaged workbook file (.twbx) is really just a compressed .zip file. If you rename the extension from .twbx to .zip, you can access the contents as you would any other .zip file.

There are a couple of security considerations to keep in mind when using an extract. First, any security based on authentication with a database engine will not be effective after the extract is created. An extract does not require a username or password. All data in an extract can be read by anyone. Second, any data for visible (non-hidden) fields contained in an extract file (.hyper), or an extract contained in a packaged workbook (.twbx), can be accessed even if the data is not shown in the visualization. Be very careful to limit access to extracts or packaged workbooks containing sensitive or proprietary data.

When to use an extract

You should consider various factors when determining whether to use an extract. In some cases, you won't have an option (for example, OLAP requires a live connection and some cloud-based data sources require an extract). In other cases, you'll want to evaluate your options.

In general, use an extract when:

- You need better performance than you can get with the live connection.
- You need the data to be portable.

- You need to use functions that are not supported by the database data engine (some examples include `MEDIAN`, which is not supported with a live connection to some database engines, or `DATEPARSE`).
- You want to share a packaged workbook. This is especially true if you want to share a packaged workbook with someone who uses the free Tableau Reader, which can only read packaged workbooks with data extracted.

In general, do not use an extract when you have any of the following use cases:

- You have sensitive data that should not be accessible by certain users. You may hide sensitive fields prior to creating the extract, in which case they are no longer part of the extract.

Extracts on Tableau Server may be locked down with row-level security and user permissions can be used to prevent downloads of the data.

- You need to manage security based on login credentials and are not able to use Tableau Server or Tableau Cloud to prevent unauthorized access to the extract.
- You need to see changes in the source data updated in real time.
- The volume of data makes the time required to build the extract impractical. The number of records that can be extracted in a reasonable amount of time will depend on factors such as the data types of fields, the number of fields, the speed of the data source, and network bandwidth.

With an understanding of how to create, manage, and use extracts (and when not to use them), we'll turn our attention to the various ways of filtering data in Tableau.

Filtering data

Often, you will want to filter data in Tableau in order to perform an analysis on a subset of data, narrow your focus, or drill into details. Tableau offers multiple ways of filtering data.

While you may apply any number of filters to any given view, if you want to limit the scope of all of your analysis to a subset of data, you can filter the data at the source using one of the following techniques:

- **Data source filters** are applied before all other filters and are useful when you want to limit your analysis to a subset of data.
- **Extract filters** limit the data that is stored in an extract. Data source filters are often converted into extract filters if they are present when you extract the data.
- **Custom SQL filters** are built into the `WHERE` clause of connections using custom SQL statements. These can be dynamic with a live connection that embeds parameter values in the SQL statement. We'll examine parameters in *Chapter 4, Starting an Adventure with Calculations*.

If your filters apply only to specific views, you can apply filters to those views using one of the following techniques:

- Drag and drop fields from the data pane to the **Filters** shelf.

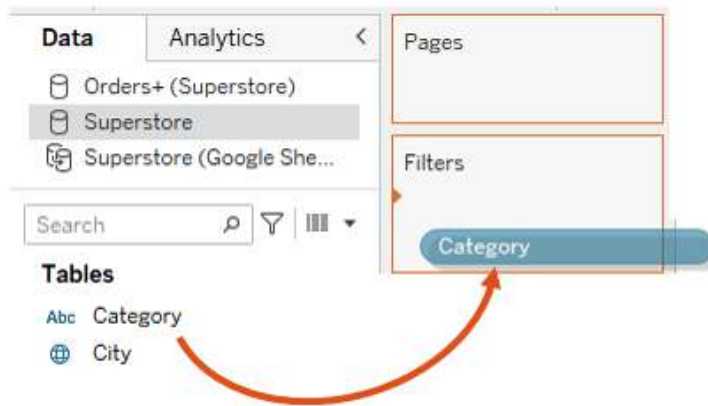


Figure 2.26: Dragging and dropping a field onto the Filters shelf

- Select one or more marks or headers in a view and then select **Keep Only** or **Exclude**, as shown here:

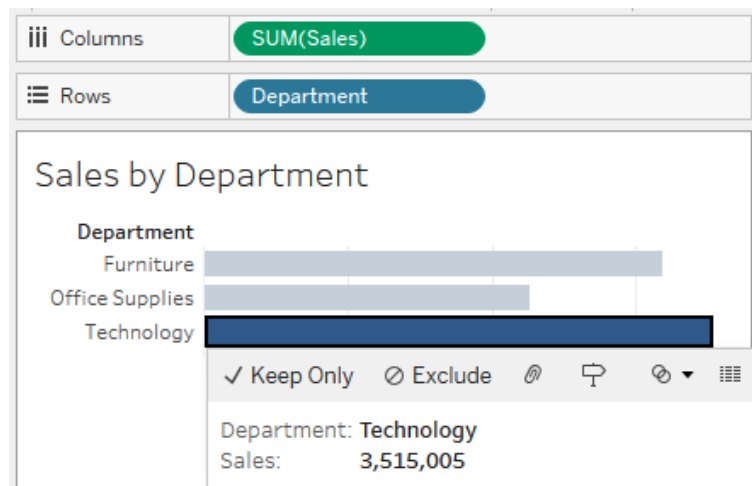


Figure 2.27: Based on the mark selection, you may keep only values that match or exclude such values

- Right-click on any field in the data pane or in the view and select **Show Filter**. The filter will be shown as a control (examples include a drop-down list and checkbox) to allow the end user of the view or dashboard the ability to change the filter.

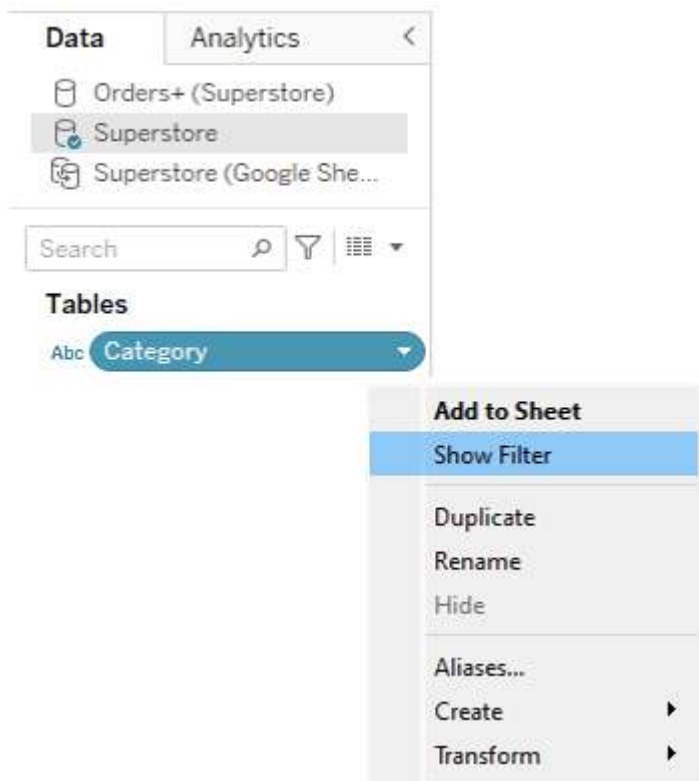


Figure 2.28: Right-click or use the drop-down menu to select *Show Filter*

- Use an action filter. We'll look more at filters and action filters as they relate to dashboards in *Chapter 8, Telling a Data Story with Dashboards*.

Each of these options adds one or more fields to the **Filters** shelf of a view. When you drop a field on the **Filters** shelf, you will be prompted with options to define the filter. The filter options will differ most noticeably based on whether the field is discrete or continuous. Whether a field is filtered as a dimension or as a measure will greatly impact how the filter is applied and the results.

Filtering discrete (blue) fields

When you filter using a discrete field, you will be given options for selecting individual values to keep or exclude. For example, when you drop the discrete **Department** dimension onto the **Filters** shelf, Tableau will give you the following options:

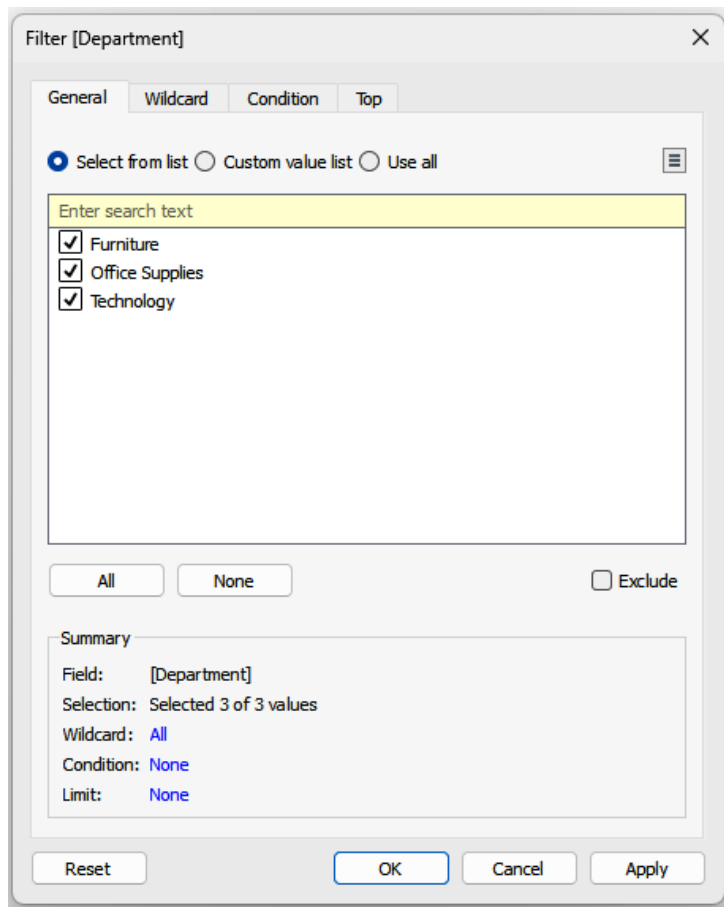


Figure 2.29: A filter for a discrete field will show options for including or excluding individual values

The **Filter** options include the **General**, **Wildcard**, **Condition**, and **Top** tabs. Your filter can include options from each tab. The **Summary** section on the **General** tab will show all options selected:

- The **General** tab allows you to select items from a list (you can use the custom list to add items manually if the dimension contains a large number of values that take a long time to load). You may use the **Exclude** option to exclude the selected items.
- The **Wildcard** tab allows you to match string values that contain, start with, end with, or exactly match a given value. For example, you might type John to filter a list of customers whose last names are Johnson or Johnston.
- The **Condition** tab allows you to specify conditions based on aggregations of other fields that meet conditions (for example, a condition to keep any **Department** where the sum of sales was greater than \$1,000,000). Additionally, you can write a custom calculation to form complex conditions. We'll cover calculations more in *Chapter 4, Starting an Adventure with Calculations*, and *Chapter 6, Diving Deep with Table Calculations*.
- The **Top** tab allows you to limit the filter to only the top or bottom items. For example, you might decide to keep only the top five items by the sum of sales.

Discrete numeric measures (except for calculated fields using table calculations) cannot be added to the **Filters** shelf. If you attempt to drag and drop a discrete measure to the Filters shelf, you will be prompted to use the field as a continuous measure or a discrete dimension.

Let's next consider how continuous filters are filtered.

Filtering continuous (green) fields

If you drop a continuous dimension onto the **Filters** shelf, you'll get a different set of options. Often, you will first be prompted as to how you want to filter the field, as follows:

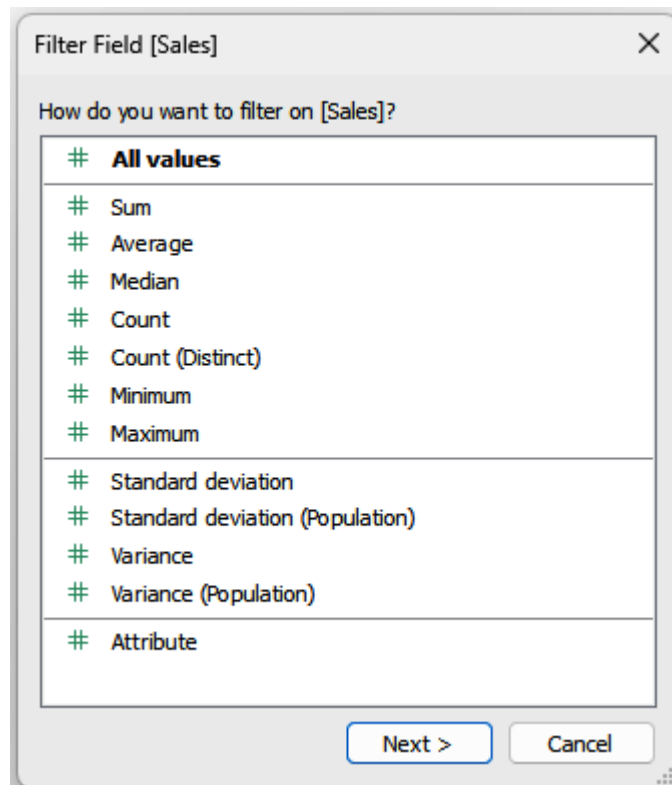


Figure 2.30: For numeric values, you'll often see options for aggregating the value as part of the filter

The options here are divided into two major categories:

- **All values:** The filter will be based on each individual value of the field, *row by row*. For example, an **All values** filter keeping only sales above \$100 will evaluate each record of underlying data and keep only individual sales above \$100.
- **Aggregation:** The filter will be based on the aggregation specified (for example, **Sum**, **Average**, **Minimum**, **Maximum**, **Standard deviation**, and **Variance**) and the aggregation will be performed at the level of detail of the view. For example, a filter keeping only the sum of sales above \$100,000 on a view at the level of items will keep only items that had at least \$100,000 in total sales.

Once you've made a selection (or if the selection wasn't applicable to the field selected), you will be given another interface for setting the actual filter, as follows:

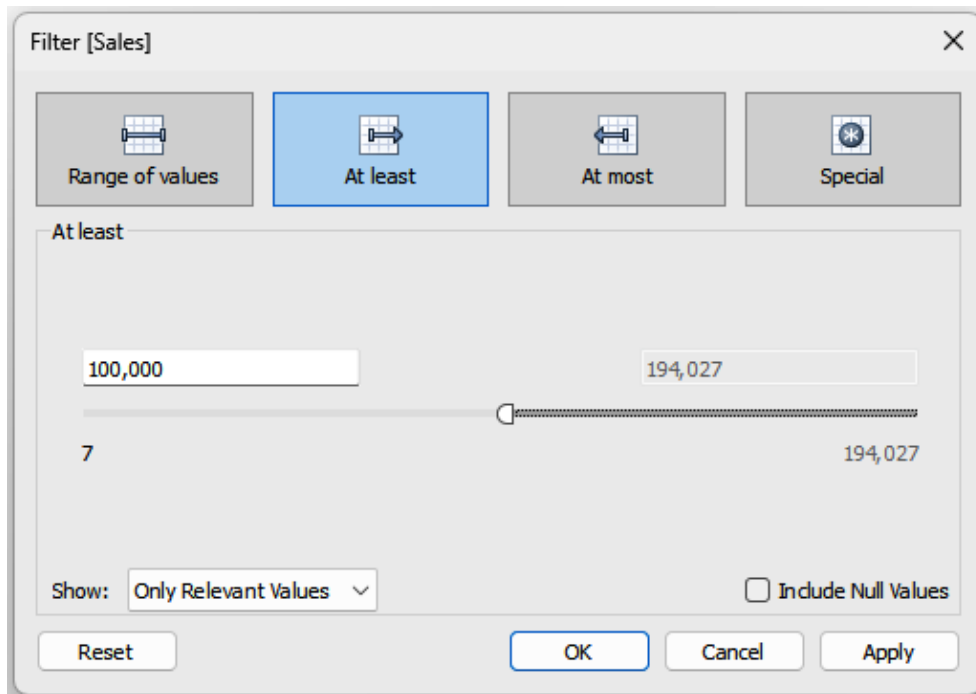


Figure 2.31: Filter options for Sales (as a SUM)

Here, you'll see options for filtering continuous values based on a range with a start, end, or both. The **Special** tab gives options for showing all values, NULL values, or non-NULL values.

From a user interface perspective, the most dramatic difference in filtering options comes from whether a field is discrete or continuous. However, you should always think about whether you are using the field as a **dimension filter** or a **measure filter** to understand what kind of results you will get based on the order of operations, which is discussed in the *Tableau's order of operations* section:

- Dimension filters will filter *detail rows of data*. For example, filtering out the central region will eliminate all rows for that region. You will not see any states for that region and your aggregate results, such as SUM(Sales), will not include any values from that region.
- Measure filters will filter *aggregate rows of data* at the level of detail defined by the dimensions included in your view. For example, if you filtered to include only where SUM(Sales) was greater than \$100,000 and your view included Region and Month, then the resulting view would include only values where the Region had more than \$100,000 in sales for the given month.

Other than filtering discrete and continuous fields, you'll also notice some different options for filtering dates, which we'll consider next.

Filtering dates

We'll take a look at the special way Tableau handles dates in the *Visualizing dates and times* section of Chapter 3, *Moving Beyond Basic Visualizations*. For now, consider the following options available when you drop an **Order Date** field onto the **Filters** shelf:

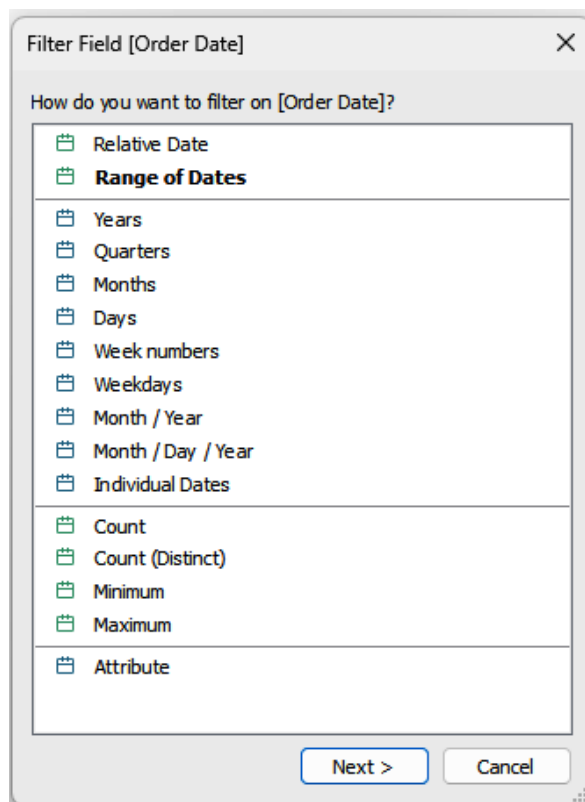


Figure 2.32: Initial filter options for a date field

The options here include the following:

- **Relative Date:** This option allows you to filter a date based on a specific date (for example, keeping the last three weeks from today, or the last six months from January 1).
- **Range of Dates:** This option allows you to filter a date based on a range with a starting date, ending date, or both.
- **Date part:** This option allows you to filter based on discrete parts of dates, such as **Years**, **Months**, or **Days**, or combinations of parts, such as **Month / Year**. Based on your selection, you will have various options for filtering and have the option of defaulting to the latest value in the data.
- **Individual Dates:** This option allows you to filter based on each value of the date field in the data.
- **Count and Count (Distinct):** These options allow you to filter based on the count or distinct count of date values in the data.

Depending on your selection, you will be given additional options for filtering.

Other filtering options

You will also want to be aware of the following options when it comes to filtering. Many are available via the drop-down menu of a field that is on the **Filters** shelf, as shown in the following image.

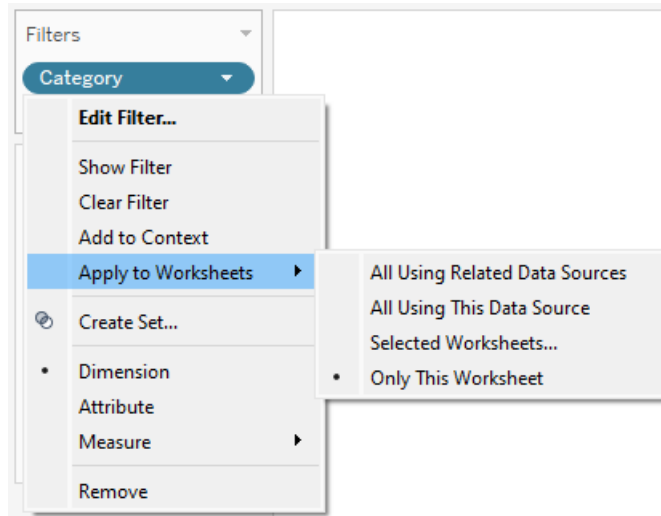


Figure 2.33: Drop-down menu options for a field on the Filters shelf

The figure illustrates where to access several key filter options in Tableau. When you right-click a field (like **Category**) on the Filters shelf, a menu appears with choices such as **Edit Filter**, **Add to Context**, and **Apply to Worksheets**. These options directly correspond to the filtering capabilities described in the following list:

- **Filters** may be added to the **context**. Context is illustrated in the *Tableau's order of operations* section, and we'll see various examples of using context throughout the book. The option for adding a field used as a filter is available via the drop-down menu on the filter control or the field on the **Filters** shelf.
- **Filters** may be set to show all values in the database, all values in the context, all values in a hierarchy, or only values that are relevant based on other filters. These options are available via the drop-down menu on the **Filter** control or the field on the **Filters** shelf.
- When using Tableau Server, you may define user filters that allow you to provide row-level security by filtering based on user credentials.
- By default, any field placed on the **Filters** shelf defines a filter that is specific to the current view. However, you may specify the scope by using the menu for the field on the **Filters** shelf. Select **Apply to** and choose one of the following options:
 - **All related data sources:** All data sources will be filtered by the value(s) specified. The relationships of fields are the same as blending (that is, the default by name and type match, or customized through the **Data | Edit Relationships...** menu option). All views using any of the related data sources will be affected by the filter. This option is sometimes referred to as **cross-data source filtering**.

- **Current data source:** The data source for that field will be filtered. Any views using that data source will be affected by the filter.
- **Selected worksheets:** Any worksheets selected that use the data source of the field will be affected by the filter.
- **Current worksheet:** Only the current view will be affected by the filter.

We'll see plenty of practical examples of filtering data throughout the book, many of which will make use of some of these options. We'll conclude the chapter with a brief look at Tableau's order of operations, where filters play a major role.

Tableau's order of operations

You'll recall the basic paradigm under which Tableau operates discussed at the beginning of this chapter. Throughout the cycle of querying the data and visualizing results, Tableau will apply various filters and calculations based on the way you've created the visualization. We have yet to cover calculations but we'll include them in the order of operations here because it is important to see the big picture. You may wish to return to this section after reading the chapters on various types of calculations to better understand how they fit.

There are many ways you might create a mental model of how Tableau orders the various filters and calculations it employs. Some have created stair or flow diagrams, which you can find via a quick Google search. Here, we'll consider a picture of ever-shrinking datasets, upon which Tableau might operate:

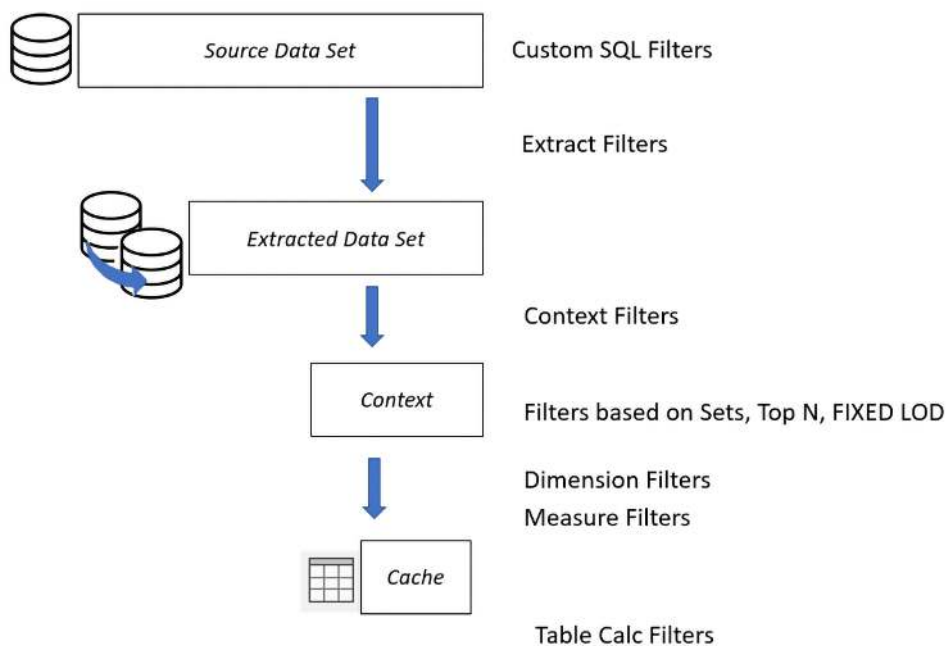


Figure 2.34: Order of operations

The original **Source Dataset** is your Excel file, Snowflake tables, or any original source to which you've connected. It contains the entirety of data that Tableau might ever be able to use. You might write custom SQL statements as part of your Tableau connection and any filters you write as part of the SQL statement will be applied before anything else.

Creating an extract is an optional choice you have for most datasets. If you do create an **Extracted Dataset**, then you may use extract filters to limit the data from the source that is stored in the extract.

As discussed previously, you may choose filters to add to **Context**. This further limits the dataset upon which Tableau operates. We'll consider sets and LOD calculations in future chapters. For now, just note that those and any filters based on them, along with **Top N** filters, operate within the context.

We've also seen how dimensions and measures may be used as filters. Note that Tableau filters dimensions first (because they might filter out rows that shouldn't be included in the aggregation of a measure filter) and then measure filters are applied.

The end result is the **Cache**—the aggregate table of results that Tableau needs for the visualization. We'll consider table calculations in *Chapter 6, Diving Deep with Table Calculations*. For now, simply note that table calculations are applied to the cache, so any filters based on table calculations are one of the last operations to be done by Tableau.

There are quite a few operations Tableau applies based on the cache, including forecasts, table calculations, sub-totals, grand totals, and trend and reference lines. Of these, table calculation filters only affect trend and reference lines.

You don't need to understand every aspect of Tableau's order of operations now. In fact, most of the time, you won't even think about it. However, you may wish to return to this section from time to time as you progress through the book and explore some of the capabilities of Tableau. Knowing the order in which Tableau operates will help you avoid some pitfalls and even come up with innovative solutions to difficult challenges!

Summary

This chapter covered key concepts of how Tableau works with data. Although you will not usually be concerned with what queries Tableau generates to query underlying data engines, having a solid understanding of Tableau's paradigm will greatly aid you as you analyze data.

We looked at multiple examples of different connections to different data sources, considered the benefits and potential drawbacks of using data extracts, considered how to manage metadata, and considered options for filtering data.

Working with data is fundamental to everything you do in Tableau. Understanding how to connect to various data sources, when to work with extracts, and how to customize metadata will be key as you begin deeper analysis and more complex visualizations, such as those covered in *Chapter 3, Moving Beyond Basic Visualizations*.

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search for this book by name.



Note: Keep your purchase invoice ready before you start.

3

Moving Beyond Basic Visualizations

You are now ready to set out on your adventure of creating more advanced visualizations! In this chapter, we will explore various types of advanced visualizations, and we'll explore new capabilities introduced in Tableau 2024, such as Viz Extensions, which simplify the creation of advanced visualization types.

Advanced does not necessarily mean *difficult* since Tableau makes many visualizations easy to create.

Advanced also does not necessarily mean *complex*. The goal is to communicate the data, not to obscure it in needless complexity.

Instead, these visualizations are advanced in the sense that you will need to understand when they should be used, why they are useful, and how to leverage the capabilities of Tableau to create them. Additionally, many of the examples we will look at will introduce some advanced techniques, such as calculations, to extend the usefulness of foundational visualizations. Many of these techniques will be developed fully in future chapters, so don't worry about trying to absorb every detail right now.

Most of the examples in this chapter are designed so that you can follow along. However, don't simply memorize a set of instructions. Instead, take the time to understand how the combinations of different field types you place on different shelves change the way headers, axes, and marks are rendered. Experiment and even deviate from the instructions occasionally to see what else is possible. You can always use Tableau's back button to go through the example again!

In this chapter, we will cover the following categories of visualizations:

- Comparing values
- Visualizing dates and times
- Visualizing distributions
- Visualizing multiple axes to compare different measures
- Going further with Viz Extensions

You may have noticed the lack of a spatial location or geographic category in the preceding list. Mapping was introduced in *Chapter 1, Taking Off with Tableau*, and we'll get to some advanced geographic capabilities in *Chapter 12, Exploring Mapping and Advanced Geospatial Features*.

You can recreate the examples that are found in this chapter by using the `Chapter 03 Starter.twbx` workbook, or you can even start from scratch by using a blank workbook and connecting to the `Hospital Visits.csv` file that's located in the `Learning Tableau/Chapter 03` folder. The completed examples can be found in the `Chapter 03 Complete.twbx` workbook.

We will begin by assessing which types of visualizations are effective for quantitative comparisons.

Comparing values

Often, you will want to compare the differences between measured values across different categories. You might find yourself asking the following questions:

- How many customers did each store serve?
- How much energy did each wind farm produce?
- How many patients did each doctor see?

In each case, you are looking to make a comparison (among stores, wind farms, or doctors) in terms of some quantitative measurement (the number of customers, units of electricity, and patients).

Let's look at some examples of visualizations that help answer questions like these.

Bar charts

Here is a simple bar chart, similar to the one we built in *Chapter 1, Taking Off with Tableau*:

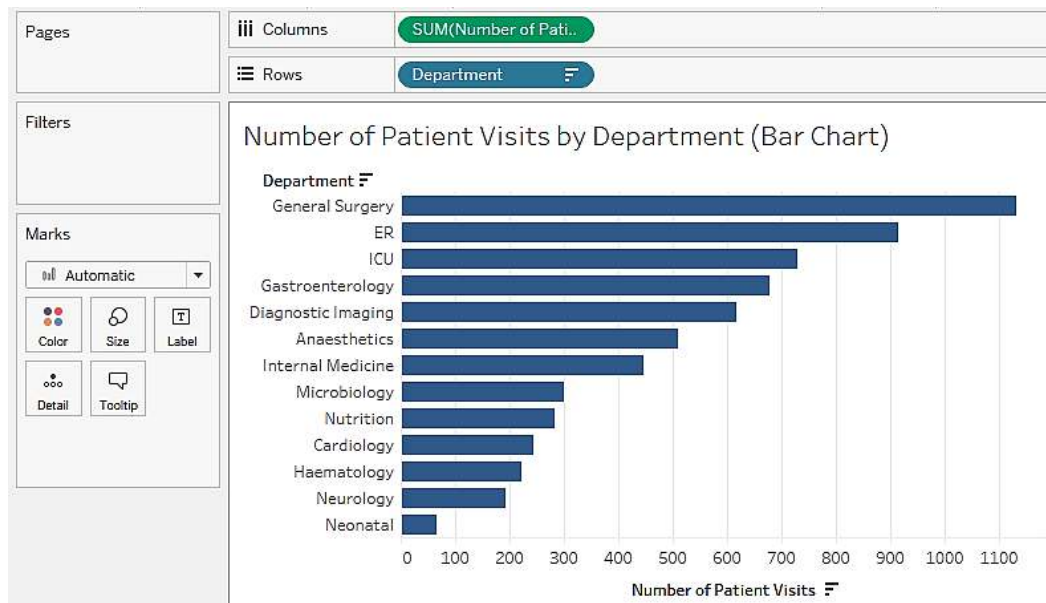


Figure 3.1: A bar chart showing the number of patient visits by department

This bar chart makes it easy to compare the number of patient visits between various departments in the hospital. As a dimension, **Department** slices the data according to each distinct value, such as **ER**, **ICU**, or **Cardiology**. It creates a header for these values because it is discrete (blue). As a measure, **Number of Patient Visits** gives the sum of patient visits for each department. Because it is a continuous (green) field, it defines an axis, and bars are rendered to visualize the value.

Notice that the bar chart is sorted with the department having the highest sum of patient visits at the top and the lowest at the bottom. Sorting a bar chart often adds a lot of value to the analysis because it makes it easier to make comparisons and see the rank order. For example, it is easy to see that the **Microbiology** department has had more patient visits than the **Nutrition** department. If the chart wasn't sorted, this may not have been as obvious.

You can sort a view in multiple ways, as follows:

- Click one of the sort icons on the toolbar: This results in an automatic sort of the dimension based on the measure that defined the axis. Changes in data or filtering that result in a new order will be reflected in the view:

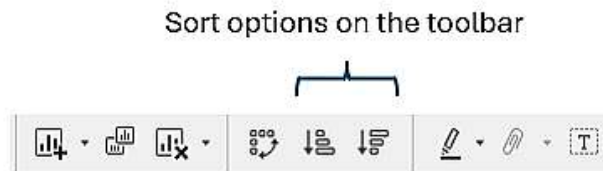


Figure 3.2: Toolbar sort icons

- Click the sort icon on the axis: The option icon will become visible when you hover over the axis and will then remain in place when you enable the sort. This will also result in automatic sorting:

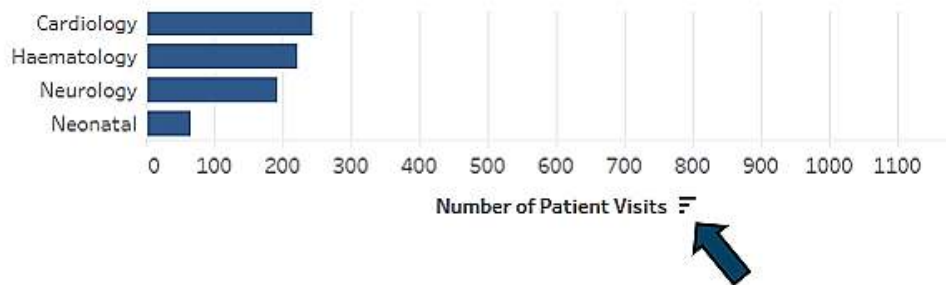


Figure 3.3: Axis sort icon

- Use the drop-down menu on the active dimension field and select **Sort...** to view and edit the sorting options. You can also select **Clear Sort** to remove any sorting:

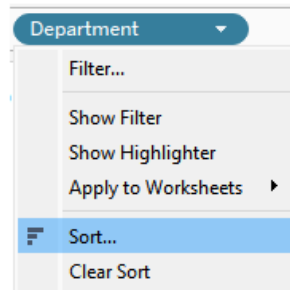


Figure 3.4: Sorting using the drop-down menu

- Use the drop-down menu on the field label for rows and select the desired sorting option:

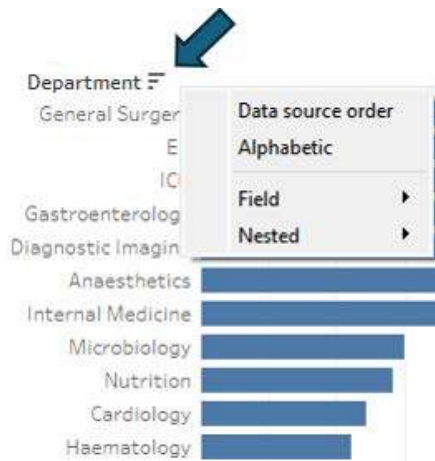


Figure 3.5: Sorting by the field label

- Drag and drop row headers to manually rearrange them. This results in a manual sort that does not get updated with data refreshes.

All of these sorting methods are specific to the view and will override any default sort you defined in the metadata.

Bar chart variations

A basic bar chart can be extended in many ways to accomplish various objectives. Consider the following variations:

- Bullet chart to show progress toward a goal, target, or threshold
- Bar-in-bar chart to show progress toward a target or compare two specific values within a category
- Highlighting categories of interest

Bullet chart

A **bullet graph** (sometimes also called a **bullet chart**) is a great way to visually compare a measure with a goal, target, or threshold. The bar indicates the measure value, while the line indicates the target. Tableau also defaults to shading to indicate 60% and 80% of the distance to the goal or threshold. The line and the shading are reference lines. We'll consider detailed examples of working with reference lines throughout the book, especially in *Chapter 9, Visual Analytics: Trends, Clustering, Distributions, and Forecasting*. Here is an example of a bullet chart:

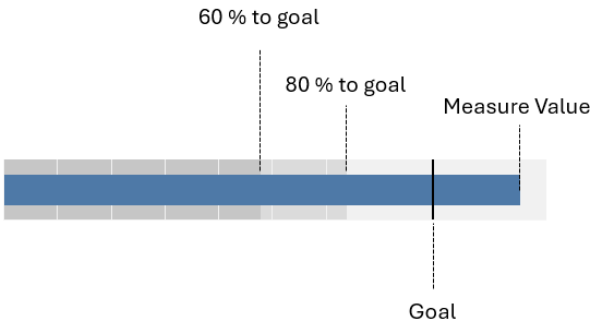


Figure 3.6: Parts of a bullet graph

In this scenario, we'll analyze how the hospital operated with respect to revenue goals. The hospital administration set the following revenue goals for 2025 and now wants to understand how each department actually performed:

Anaesthetics	\$300,000
Cardiology	\$5,000,000
Diagnostic Imaging	\$500,000
ER	\$6,000,000
Gastroenterology	\$900,000
General Surgery	\$8,000,000
Haematology	\$800,000
ICU	\$3,800,000
Internal Medicine	\$200,000
Microbiology	\$50,000
Neonatal	\$10,000
Neurology	\$3,000,000
Nutrition	\$10,000

Figure 3.7: Department goals are stored in a spreadsheet, as shown here

The patient visit and revenue data is contained in `Hospital Visits.csv`, and the revenue goals are in `Hospital Goals.csv`. These two data connections are related together in a **Data Model** in the Starter and Complete workbooks. We'll look more at the data model in *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*. For now, simply use the `Hospital Visits & Revenue` data source to complete the examples in this chapter.

We'll build a bullet graph using the Chapter 3 workbook, which contains the **Hospital Visits** and **Hospital Goals** spreadsheet data sources. We'll use these two data sources to visualize the relationship between actual and target minutes to service as you follow these steps:

1. Navigate to the **Revenue Per Department (Bullet Chart)** sheet.
2. Create a basic bar chart of the total **Revenue** per **Department**. You can accomplish this by dragging **Department** to **Rows** and **Revenue** to **Columns**.
3. Sort **Department** from highest to lowest.
4. Filter **Date of Admit** by **Year** and keep only 2025 data. At this point, your view should look like this:

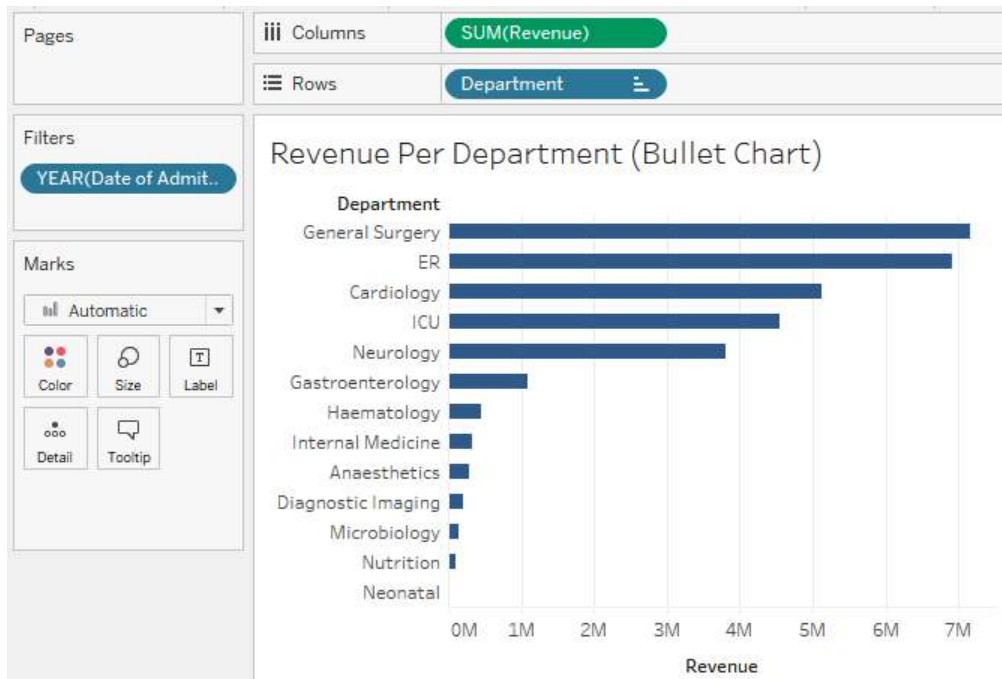


Figure 3.8: Interim steps in creating the bullet graph

5. In the **Data** pane, select the **Goal** field under the **Hospital Goals.csv** table.
6. Open **Show Me** and select the bullet graph option. At this point, Tableau will have created a bullet graph using the fields already in the view and the **Goal** field you selected in the data pane.

When you use **Show Me** to create a bullet chart, you may sometimes find that Tableau uses the fields in reverse order from what you intend (with the wrong measure defining the axis and bars, and the other defining the reference line). If this happens, simply right-click the axis and select **Swap Reference Line Fields**:

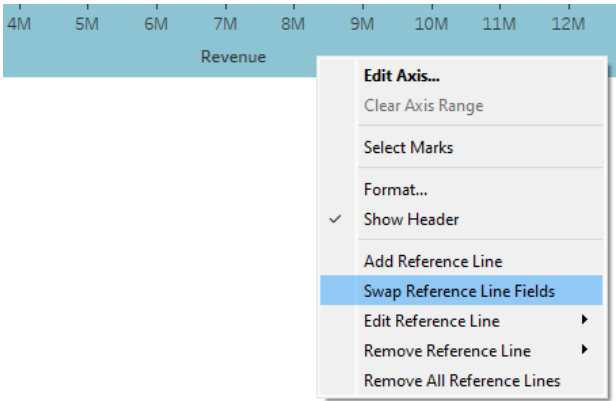


Figure 3.9: The Swap Reference Line Fields option

The completed bullet chart should look like the following:

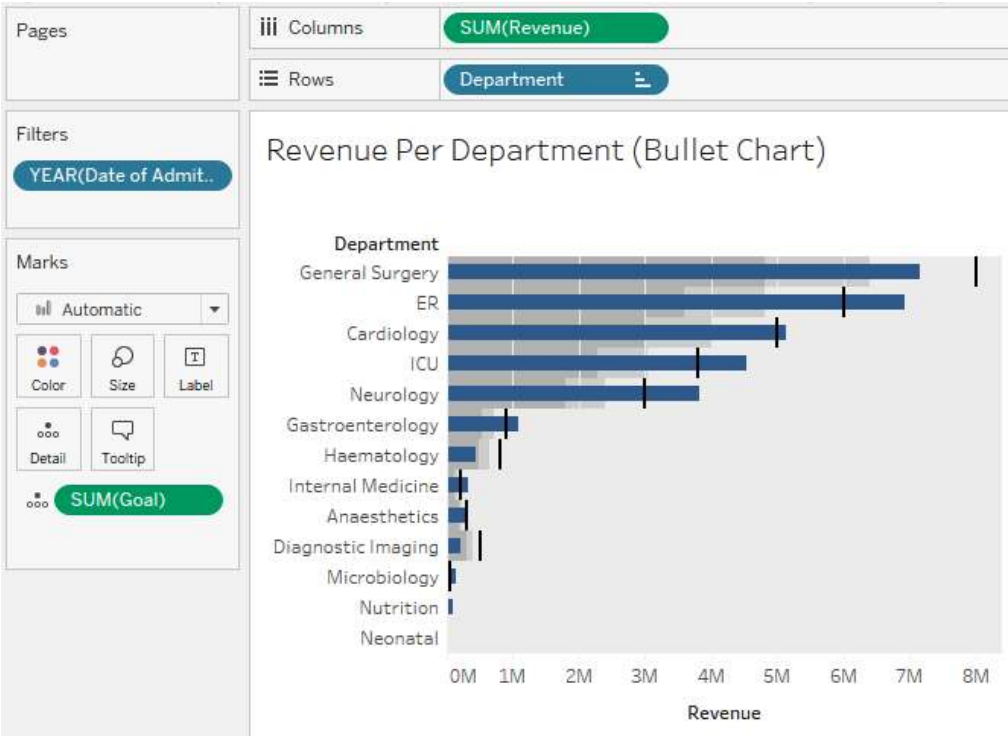


Figure 3.10: The complete bullet graph

The complete bullet graph allows us to see which departments have met their goals and which are behind. Next, let’s consider how we can highlight this even more.

Calling out thresholds

With bullet charts, it can be helpful to visually highlight the bars that fail to meet or exceed the threshold. We’ll look at calculations in depth in the next chapter, but for now, you can complete this

example with the following steps:

1. Use the drop-down arrow in the **Data** pane and select **Create Calculated Field...**:

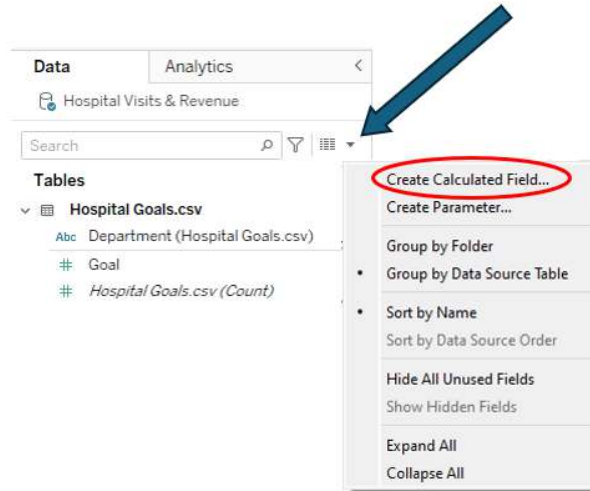


Figure 3.11: Creating a calculated field

2. Name the calculated field **Goal Met?** and add the following code:

```
SUM([Revenue]) >= SUM([Goal])
```

 **Quick tip:** Enhance your coding experience with the **AI Code Explainer** and **Quick Copy** features. Open this book in the next-gen Packt Reader. Click the **Copy** button

- (1) to quickly copy code into your coding environment, or click the **Explain** button
- (2) to get the AI assistant to explain a block of code to you.

```
function calculate(a, b) {  
  return {sum: a + b};  
};
```

Copy

Explain

1

2



 **The next-gen Packt Reader** is included for free with the purchase of this book. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



3. Click OK, drag the new **Goal Met?** field from the **Data** pane, and drop it on **Color**.

The calculation returns true when the Revenue value for a department is greater than the goal value, and false otherwise. With the calculated field on **Color**, it becomes very easy to see which departments have met the 2025 goals:

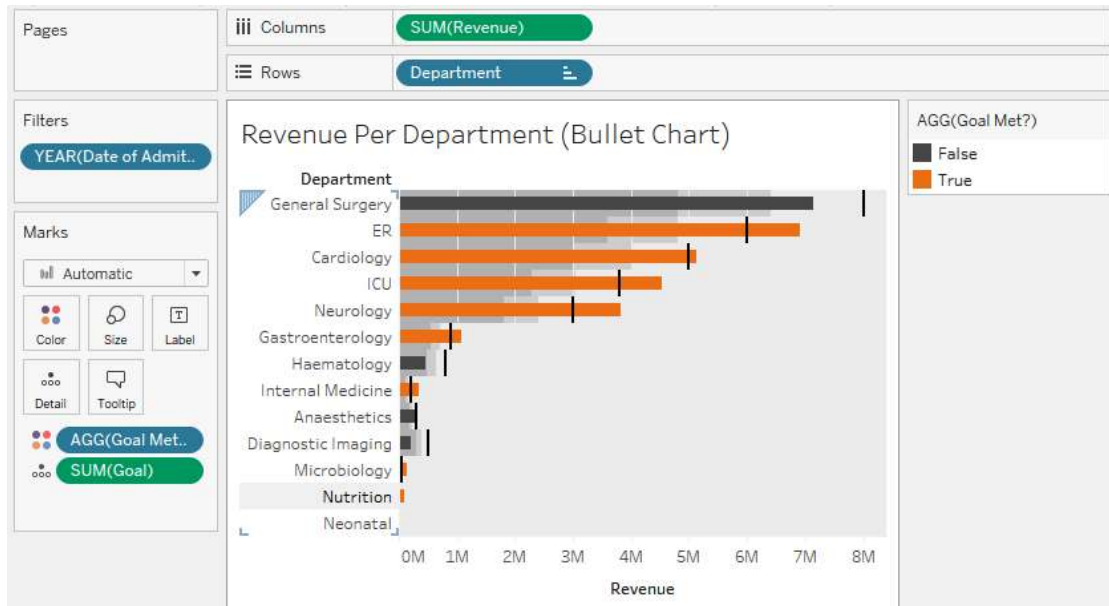


Figure 3.12: Departments that have met their goal are highlighted in this bullet chart

Color is one of the strongest ways to get attention with your visualizations. Use color with intent. Decide whether you want to highlight good or poor performance.

Bar-in-bar chart

Another way to show relationships between two values for each category is with a bar-in-bar chart. Like the bullet chart, the bar-in-bar chart can show progress toward a goal. It can also be used to compare any two values.

For example, you might compare revenue to a target, or you might compare the revenue for the current year to the previous year:

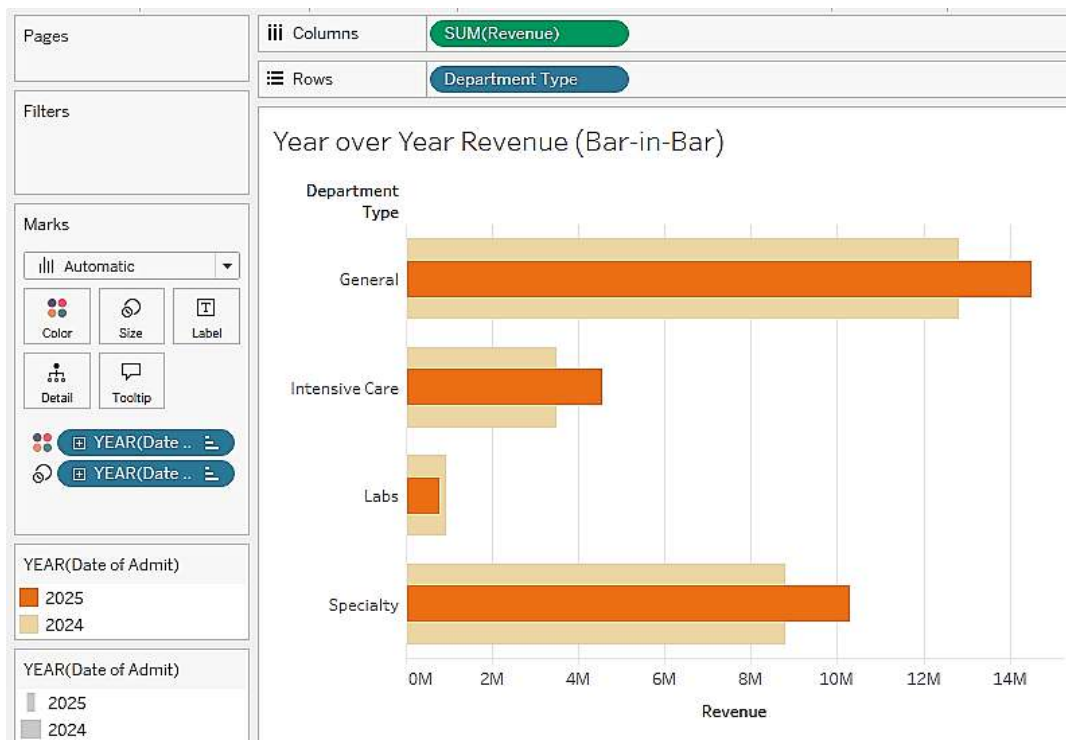


Figure 3.13: Bar-in-bar chart

To create this view, continue in the same workbook and follow these steps:

1. Navigate to the **Year over Year Revenue (Bar-in-Bar)** sheet.
2. Drag and drop **Revenue** onto the horizontal axis in the view (which gives the same results as dropping it onto the **Columns** shelf).
3. Drag and drop **Department Type** onto **Rows**.
4. Drag and drop **Date of Admit** onto **Color**. We'll discuss dates in more detail in the next section, but you'll notice that Tableau uses the year of the date to give you a stacked bar chart that looks like this:

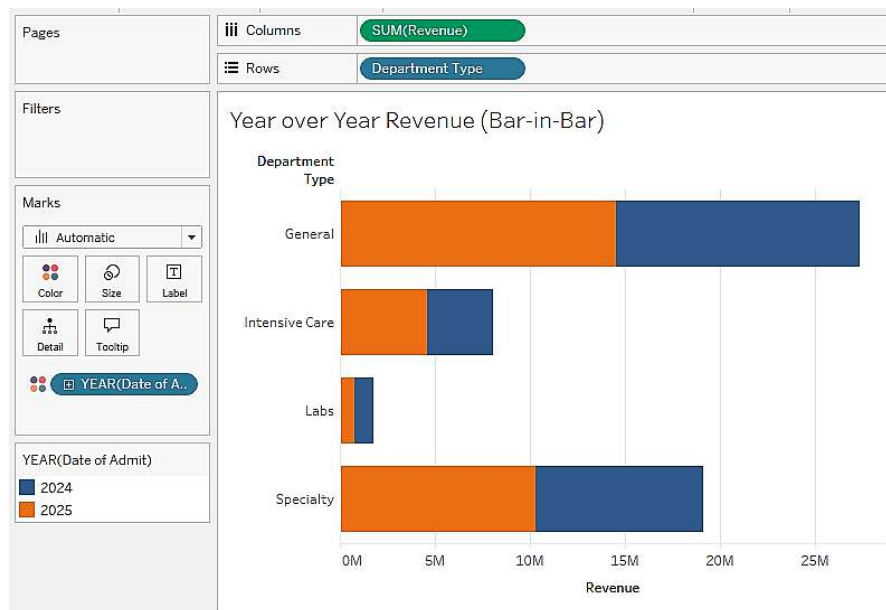


Figure 3.14: Interim steps in creating the bar-in-bar chart

- For a bar-in-bar chart, we do not want the marks to be stacked. To turn off stacking, use the main menu to select **Analysis | Stack Marks | Off**.
- All the bar segments now begin at 0, and some bars may completely obscure others. To see each bar, we'll need to adjust another visual element. In this case, hold down the *Ctrl* key while dragging the **YEAR(Date of Admit)** field that is currently on **Color** in the **Marks** card to **Size**.



After completing the previous step, a size legend should appear. The bars will be sized based on the year, and we will be able to see all of the segments that are available, even if they overlap.

- We want 2025 to be in front and 2024 to be in the background, so drag and drop 2025 within the **Size** legend to reorder the values so that 2024 comes after 2025:

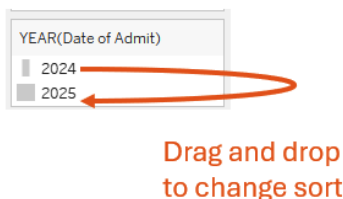


Figure 3.15: You can drag and drop items in legends to reorder them

8. Double-click the **Color** legend to edit the colors so that 2025 is emphasized. A darker orange or blue for 2025 with a light gray or lighter shade of the same color for 2024 would serve this purpose well (though you may find other color combinations that you prefer!).

At this point, your view should look like the bar-in-bar chart that was shown in *Figure 3.13* at the beginning of this section. You may wish to further enhance the visualization by doing the following:

- Adding a border to the bars. Accomplish this by clicking the **Color** shelf and using the **Border** option. (This optional step was not done in the example. You will need to decide the aesthetic that works best for you and your audience).
- Adjusting the size range to reduce the difference between the large and small extremes. Accomplish this by double-clicking the **Size** legend (or using the caret dropdown and selecting **Edit** from the menu).
- Adjusting the sizing of the view. Accomplish this by hovering over the canvas, just over the bottom border, until the mouse cursor changes to a sizing cursor, and then click and drag to resize the view. You may also want to adjust how the view fills the space. Use the dropdown on the toolbar and experiment with the options:



Figure 3.16: This dropdown determines how the current view is sized

- Hiding the size legend. You may decide that the size legend does not add anything to this particular view as the size was only used to allow overlapping bars to be seen. To hide any legend, use the drop-down arrow on the legend and select **Hide Card**:

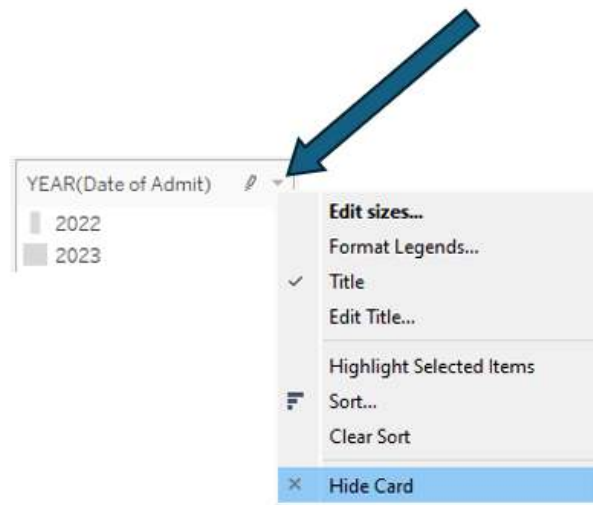


Figure 3.17: The Hide Card option for legends

The bar-in-bar chart gives you another way to compare values. Next, we'll consider a variation that allows us to highlight areas of interest.

Highlighting categories of interest

Let's say one of your primary responsibilities at the hospital is to monitor the number of patient visits to the **ICU** and **Neonatal** departments. You don't necessarily care about the details of other departments, but you do want to keep track of how your two departments compare with others. You might design something like this:

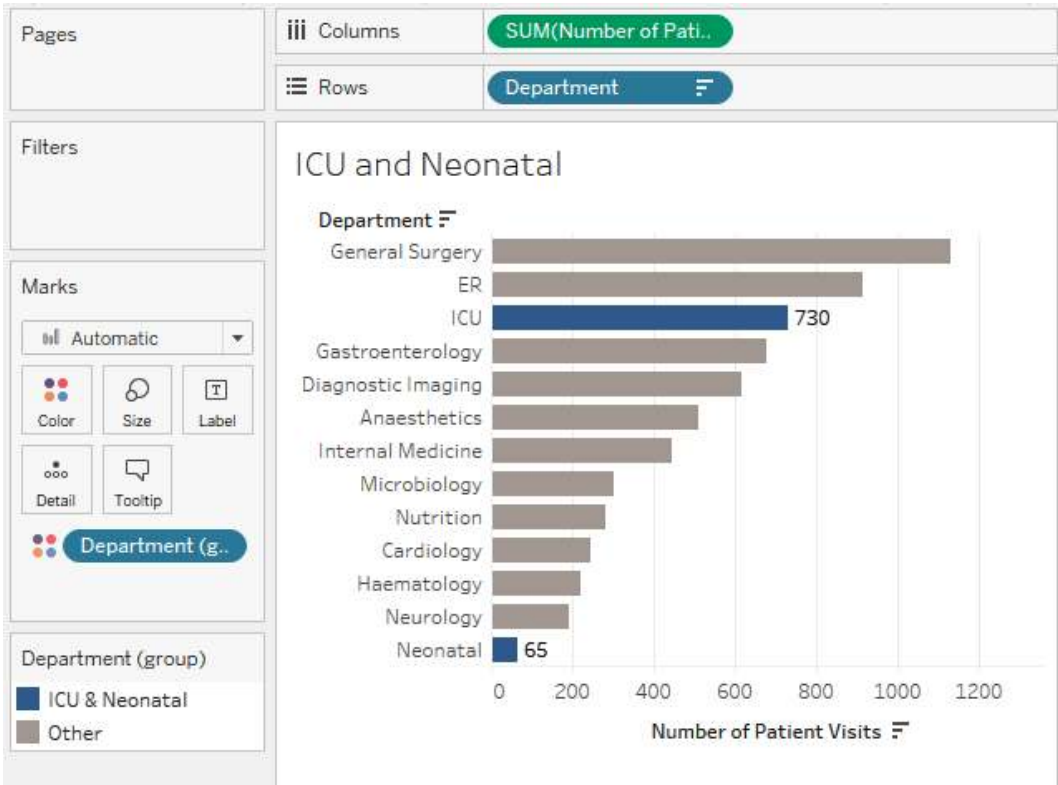


Figure 3.18: A bar chart with two bars highlighted via color

Now, as the data is refreshed over time, you will be able to immediately see how the two departments of interest to you compared to other departments. To create this view, follow these steps:

1. Navigate to the **ICU and Neonatal** sheet.
2. Place **Department** on Rows and **Number of Patient Visits** on Columns. Sort the bar chart in descending order.
3. Click on the bar in the view for **ICU**, and while holding down the *Ctrl* key, click the bar for **Neonatal**.

4. Hover the cursor over one of the selected bars for a few seconds, and from the menu that appears, click the **Create Group** button (which looks like a paperclip):

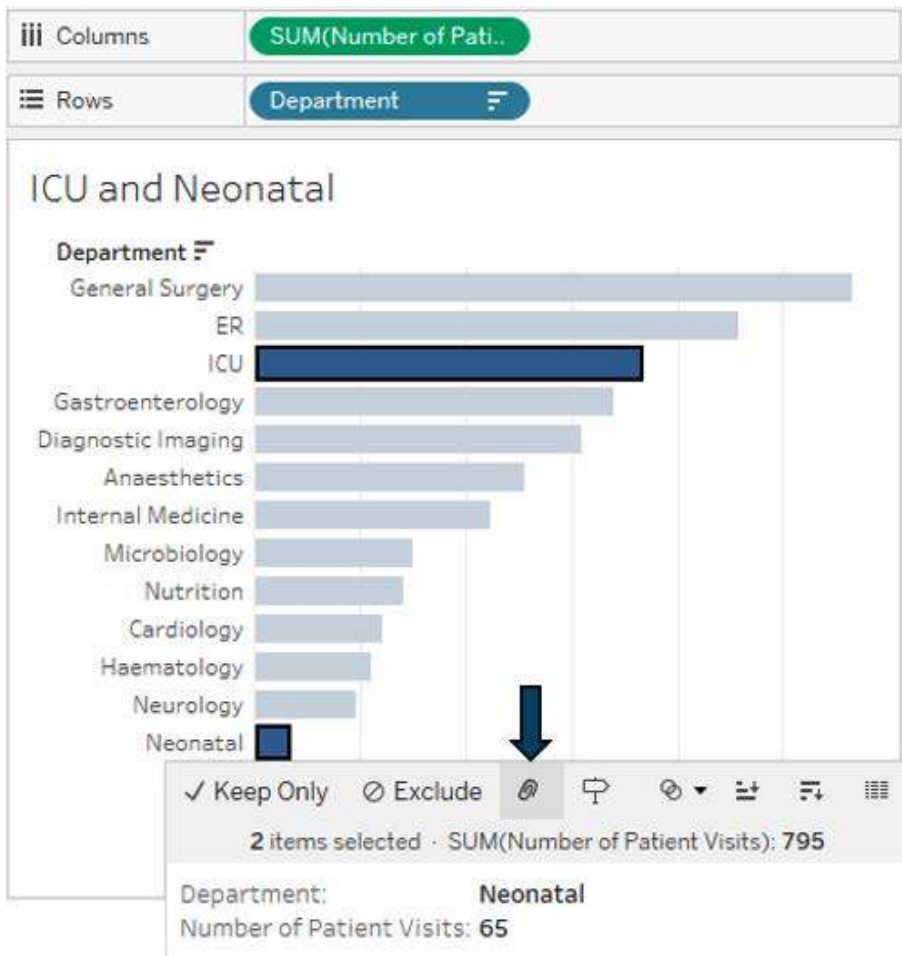


Figure 3.19: After Ctrl + clicking the two bars, use the paperclip icon to group them

This will create a group, which results in a new dimension, named **Department (group)**, in the left-hand data pane. Tableau automatically assigns this field to **Color**.



Ad hoc groups are powerful in Tableau. You can create groups in the view (as you did previously) or by using the menu for a dimension in the data pane and selecting **Create | Group**. You can use them as you would any other dimension.

5. To add a label only to the bars for those two departments, right-click each bar and select **Mark label | Always show**. The label for the mark will always be shown, even if other labels are turned off for the view or the label overlaps marks or other labels.

The color will make monitoring easy. The label will only be displayed for the two departments you selected and will update with the changing data.

Now that we've considered how bar charts can be used to compare values and have walked through several examples of variations, let's turn our attention to visualizing dates and times.

Visualizing dates and times

In your analysis, you will often want to understand when something happened. You'll ask questions like the following:

- When did we gain the most new customers?
- Is profit trending up or down?
- What times of day have the highest call volume?
- What kinds of seasonal trends do we see in sales?

Fortunately, Tableau makes this kind of visual discovery and analysis easy. In this section, we'll look at how Tableau works with dates and some various ways you might visualize time.

Date parts, date values, and exact dates

When you are connected to a flat file, relational, or extracted data source, Tableau provides a robust built-in date hierarchy for any date field.



Cubes/OLAP connections do not allow Tableau hierarchies. You will want to ensure that all the date hierarchies and date values you need are defined in the cube.

To see this in action, continue with the Chapter 3 workbook, navigate to the Built-in Date Hierarchy sheet, and create a view by dragging and dropping **Number of Patient Visits** to **Rows** and **Date of Admit** to **Columns**. The **YEAR(Date of Admit)** field on **Columns** will have a plus sign indicator, like this:

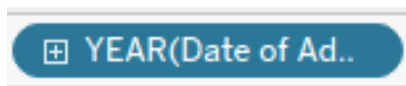


Figure 3.20: The field representing the YEAR part of the date hierarchy

You'll also find a plus or minus indicator as you hover over headers, like this:

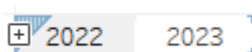


Figure 3.21: A plus icon on the column headers that could be used to expand the hierarchy

When you click it, the hierarchy expands by adding **QUARTER(Date of Admit)** to the right of **YEAR(Date of Admit)** on **Columns**, and the view is expanded to the new level of the hierarchy:

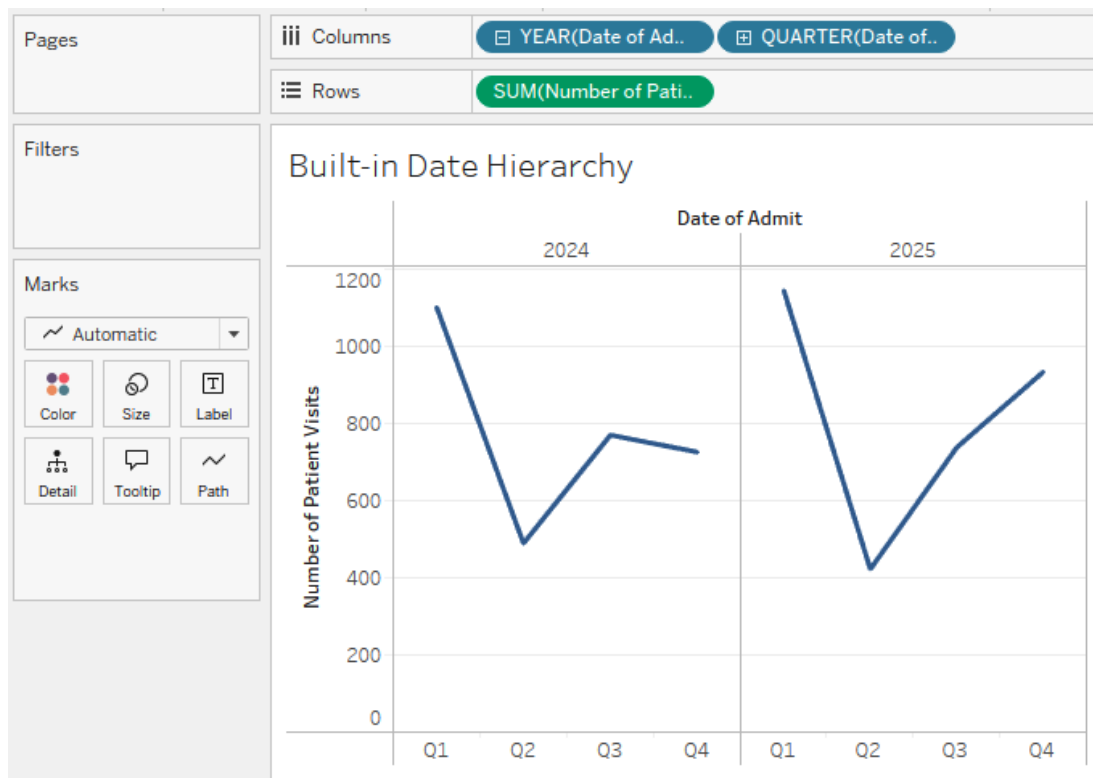


Figure 3.22: The expanded hierarchy with the year and quarter shown

The **YEAR(Date of Admit)** field now has a minus sign indicator that allows you to collapse the hierarchy back to the year level. The **QUARTER** field also has a plus sign, indicating that you can expand the hierarchy further. Starting with **Year**, the hierarchy flows as follows: **Year** | **Quarter** | **Month** | **Day**. When the field is a date and time, you can further drill down into **Hour** | **Minute** | **Second**. Any of the parts of the hierarchy can be moved within the view or removed from the view completely.

The hierarchy is made up of **Date Parts**, which is one of the three ways a date field can be used. When you right-click the date field in the view or by using the drop-down menu, you'll see multiple date options, as follows:

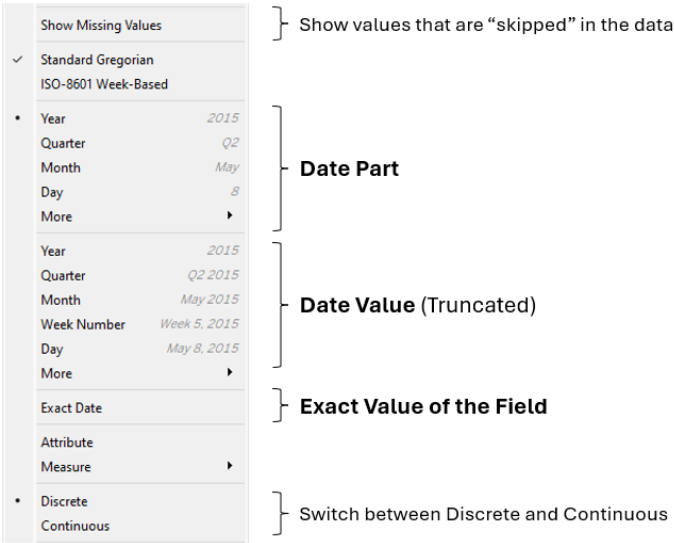


Figure 3.23: The drop-down menu on an active date field demonstrates the various aspects of dates in Tableau

The three main date types are evident, though not explicitly labeled, in the menu:

- Date part:** This field will represent a specific part of the date, such as the quarter or month. The part of the date is used by itself and without reference to any other part of the date. This means that the date of November 8, 1980, when used as a month date part, is simply **November** in the view. The **November** that's selected in the view here represents all of the Novembers in the dataset, while the number of patient visits is the total for both 2024 and 2025:

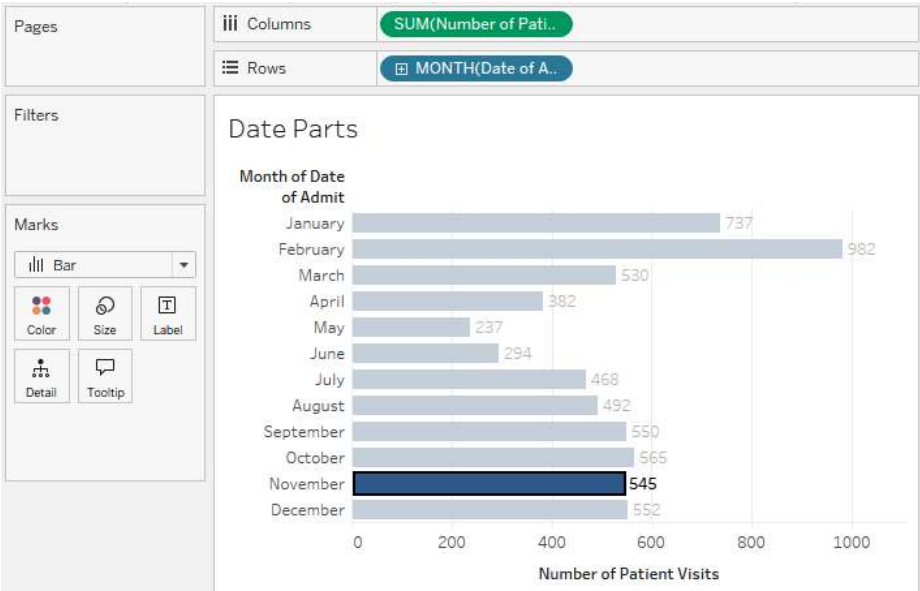


Figure 3.24: This view uses Month as a date value. The number of patient visits is the total for the month, without regard to the year

- Date value:** This field will represent a date value, but rolled up or truncated to the level you select. For example, if you select a date value of **Month**, then November 8, 2025, gets truncated to the month and year and is **November 2025**. You'll notice that **November 2024** and **November 2025** each have a separate value in the header and a distinct bar:

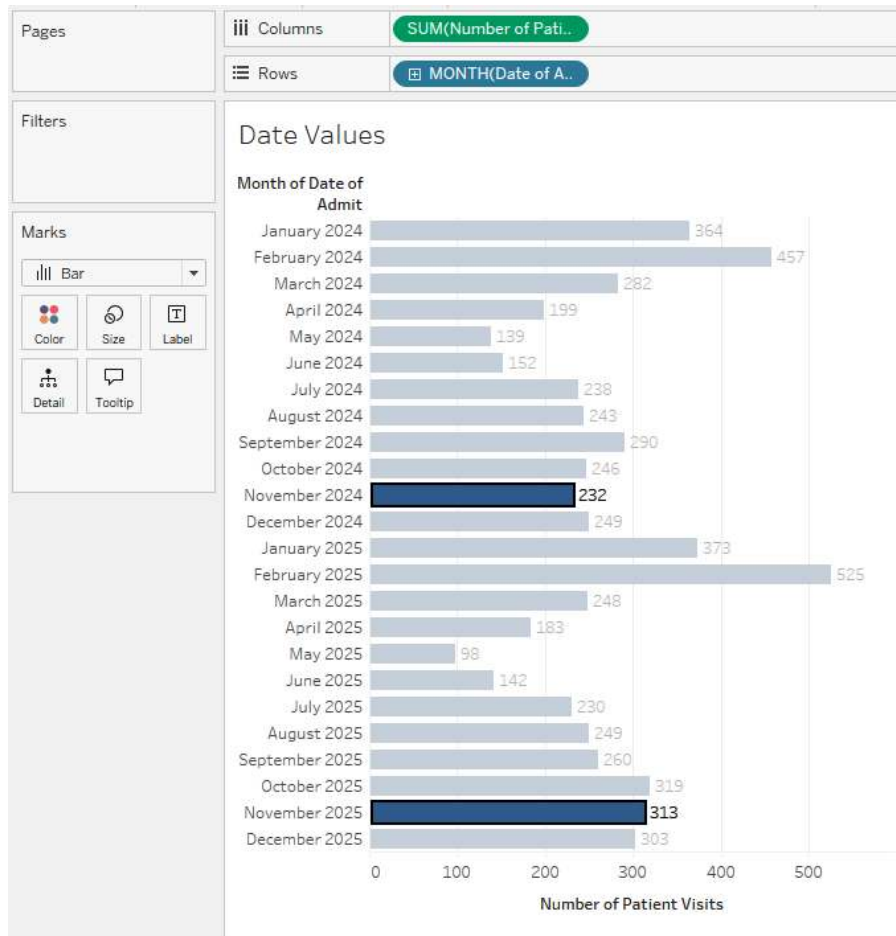


Figure 3.25: This view uses Month as a date value. The number of patient visits is the total for the month with regard to the year

- Exact date:** This field represents the exact date value (including time, if applicable) in the data. This means that November 8, 1980, 2:01 a.m. is treated as distinct from November 8, 1980, 3:08 p.m.

It is important to note that nearly any of these options can be used as discrete or continuous fields. Date parts are discrete by default. Date values and exact dates are continuous by default. However, you can switch between discrete and continuous as required to allow flexibility in the visualization.

For example, you must have an axis (requiring a continuous field) to create a reference line. Also, Tableau will only connect lines at the lowest level of row or column headers. Using a continuous date value instead of multiple discrete date parts will allow you to connect lines across multiple years, quarters, and months.



As a shortcut, you can right-click and then drag and drop a date field into the view to get a menu of options for how the date field should be used prior to the view being drawn.

Let’s consider some various ways we might visualize dates and times next.

Variations of date and time visualizations

The ability to use various parts and values of dates and even mix and match them gives you a lot of flexibility in creating unique and useful visualizations.

For example, using the month date part for columns and the year date part for color gives a time series that makes a visual year-over-year comparison quite easy. The year date part has been copied to the label so that the lines can be labeled:



Figure 3.26: The comparison of two years, month by month

This kind of view allows easy year-by-year comparison.



The following heat map is another example of using date parts on different shelves to achieve useful analysis. This kind of visualization can be quite useful when looking at patterns across different parts of time, such as hours in a day or weeks in a month. Here, we are looking at how many patients were admitted by month and day:

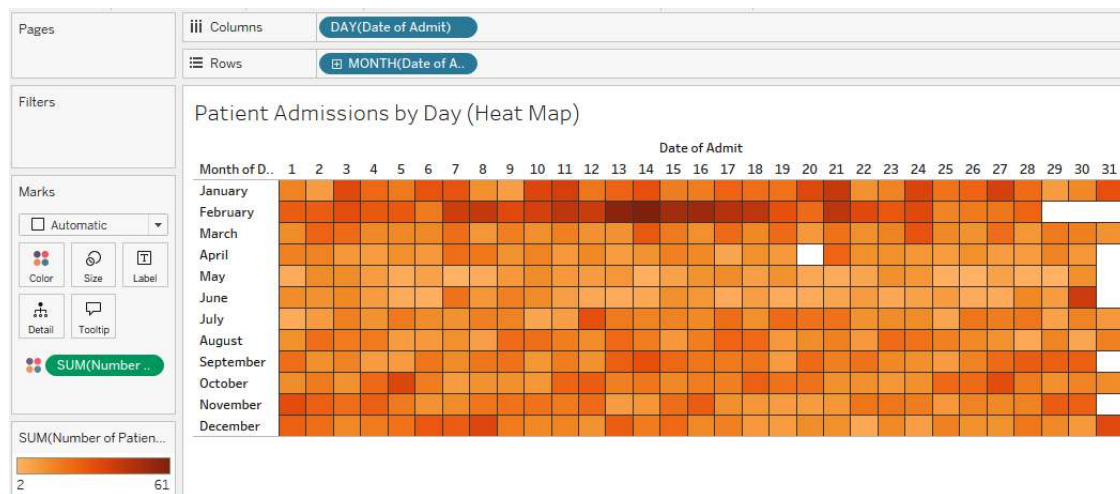


Figure 3.27: A heat map showing the intensity of patient visits by day and month

The year has not been included in the view, so this is an analysis of all years in the data and allows us to see whether there are any seasonal patterns or hotspots. We might notice patterns related to epidemics, doctors' schedules, or the timing of insurance benefits. Perhaps the increased intensity of patient admissions in February corresponds to the flu season.

Observe that placing a continuous field on the **Color** shelf resulted in Tableau completely filling each intersection of **Row** and **Column** with the shade of color that encoded the sum of patient visits. Clicking on the **Color** shelf gives us some fine-tuning options, including the option to add borders to marks. In this view, a black border has been added to help distinguish each cell.

Gantt charts

Gantt charts can be incredibly useful for understanding any series of events with a duration, especially if those events have some kind of relationship. Visually, they are very useful for determining whether certain events overlap, have a dependency, or take more or less time than other events.

For example (not included in the workbook), the following Gantt chart shows a series of processes that run when an application is started.

Some of these processes run in parallel, and some are clearly dependent on others. The Gantt chart makes these dependencies clear:

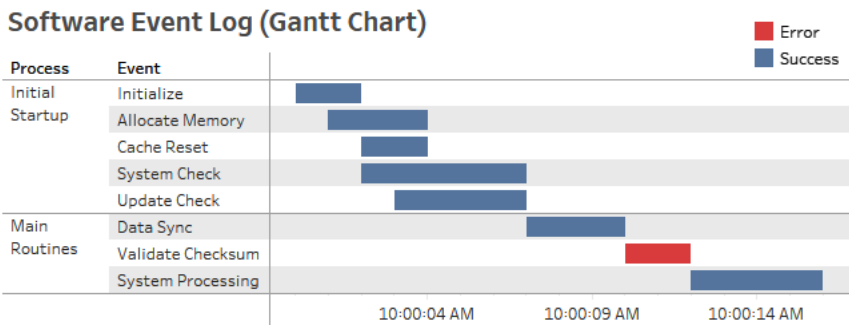


Figure 3.28: A Gantt chart showing the time each process started and how long each took

Gantt charts use the **Gantt mark** type on the **Marks** card dropdown. A Gantt bar mark starts at the value that was specified by the field on **Rows** that defines the axis. The length of the Gantt bar is then determined by the field on the **size** card, with positive values stretching to the right and negative values to the left.

At the hospital, you might want to see each patient’s visit to the **ER** in 2025 and understand how long each visit lasted, whether any patients returned to the hospital, and how much time there was between visits. The following steps give an example of how you might create a Gantt chart:

- 1. Place **Department** on **Filters** and keep only **ER**.
- 2. Place **Date of Admit** on **Filters**, select **Years** as the option for filtering, and keep only **2025**.
- 3. Place **Date of Admit** on **Columns** as a continuous **Exact Date** or as a **Day** value (not the **Day** part). Notice that Tableau’s automatic default for the mark type is Gantt bars:

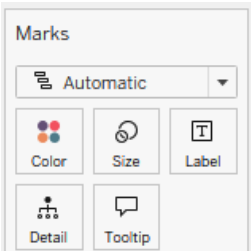


Figure 3.29: In this case, Gantt bars are the automatic mark type

- 4. Place **Doctor** and **Patient Name** on **Rows**. The result is a row for each patient grouped by each doctor. A Gantt bar represents a stay in the hospital.



In most cases, we’d also want to add a unique identifier to the view, such as **Patient ID**, to ensure that patients who happen to share the same name are distinguished in the visualization. This is not necessary with this dataset, as all the names happen to be unique, but it may be vitally important when you work with your data.

- The length of the Gantt bar is set by placing a field with a duration value on the **Size** shelf. There is no such field in this dataset. However, we have the **Date of Discharge**, and we can create a calculated field for the duration. We'll cover calculations in more detail in the next chapter. For now, select **Analysis** from the menu and click **Create Calculated Field....** Name the field **Days in the Hospital** and enter the following code:

```
DATEDIFF('day', [Date of Admit], [Date of Discharge])
```

- The new calculated field will appear among the other measures of the Hospital Visits in the data pane. Drag and drop the field onto the **Size** shelf. You now have a Gantt chart showing when patients were admitted and how long each visit lasted.



Consider sorting the **Patient Name** dimension in the view. For example, as shown in *Figure 3.30*, sorting by field or nested and selecting **Date of Admit** as the minimum would enable you to see patients who were admitted earlier toward the top and patients who were admitted later toward the bottom. It is based on the earliest (minimum) date of admission for the patient, even if they were admitted multiple times. Sorting can be a very useful technique for seeing patterns in the Gantt chart.

Your final view should look something like this:

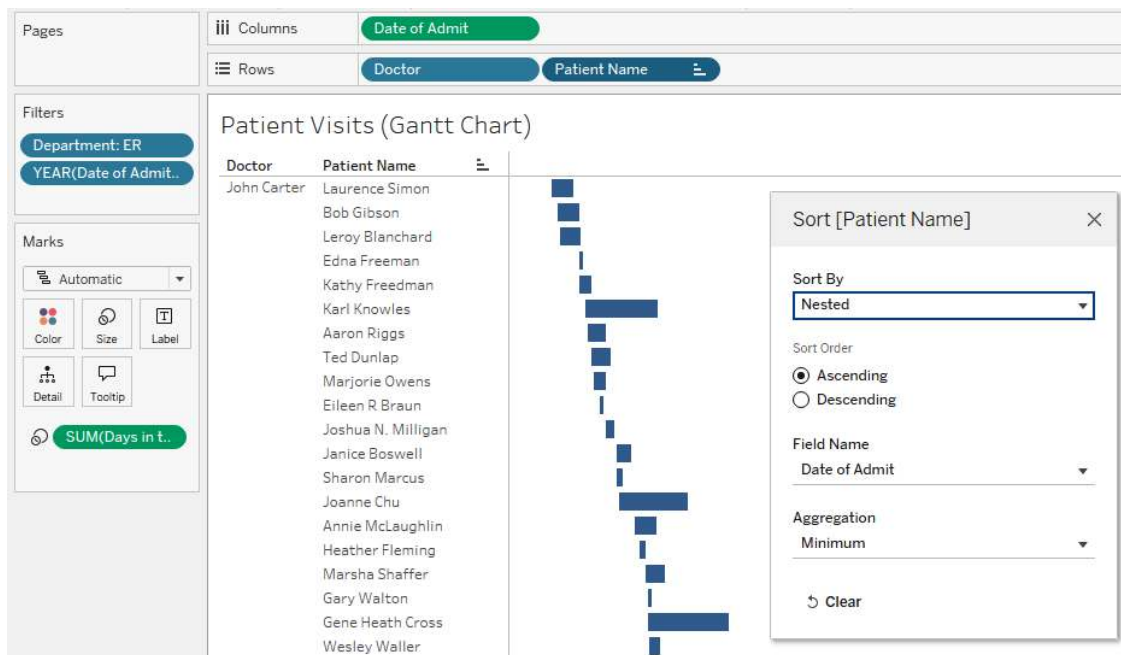


Figure 3.30: The final Gantt chart, showing each patient, when they were admitted, how long they stayed, and whether they ever returned. It has been sorted as suggested in the preceding tip

This type of chart can be very useful for finding patterns and relationships between entities over time.



Best Practice

When plotted on a date axis, the field defining the length of the Gantt bars always needs to be in terms of days. If you want to visualize events with durations that are measured in hours or seconds, avoid using the `day` argument for `DATEDIFF` because it computes whole days and loses precision in terms of hours and seconds.

Instead, calculate the difference in hours or seconds and then convert this back into days. The following code converts the number of seconds between a start and end date and then divides by 86,400 to convert the result into days, including fractional parts of the day:

```
DATEDIFF('second', [Start Date], [End Date]) / 86400.
```

With a good understanding of how Tableau works with dates and times, we've considered some different options for visualization. Let's turn our focus onto how to visualize part-to-whole relationships.

Relating parts of the data to the whole

As you explore and analyze data, you'll often want to understand how various parts add up to a whole. For example, you'll ask questions such as the following:

- How much does each electric generation method (wind, solar, coal, and nuclear) contribute to the total amount of energy produced?
- What percentage of total profit is made in each state?
- How much space does each file, subdirectory, and directory occupy on my hard disk?

These questions are asking about the relationship between the part (production method, state, and file/directory) and the whole (total energy, national sales, and hard disk). There are several types of visualizations and variations that can aid you in your analysis.

Let's now look at some visualization examples that will aid us as we consider how to show part-to-whole relationships.

Stacked bars

We looked at stacked bars in *Chapter 1, Taking Off with Tableau*, where we noted one significant drawback: it is difficult to compare values across most categories. Except for the leftmost (or bottom-most) bars, the other bar segments have different starting points, so lengths are much more difficult to compare. It doesn't mean stacked bars should never be used, but caution should be exercised to ensure clarity of communication.

Here, we are using stacked bars to visualize the makeup of the whole. We are less concerned with visually comparing across categories and more concerned with seeing the parts that make up a category.

For example, at the hospital, we might want to know what the patient population looks like within each type of department. Perhaps each patient was assigned a risk profile on admission.

We can visualize the number of visits broken down by risk profile as a stacked bar, like this:

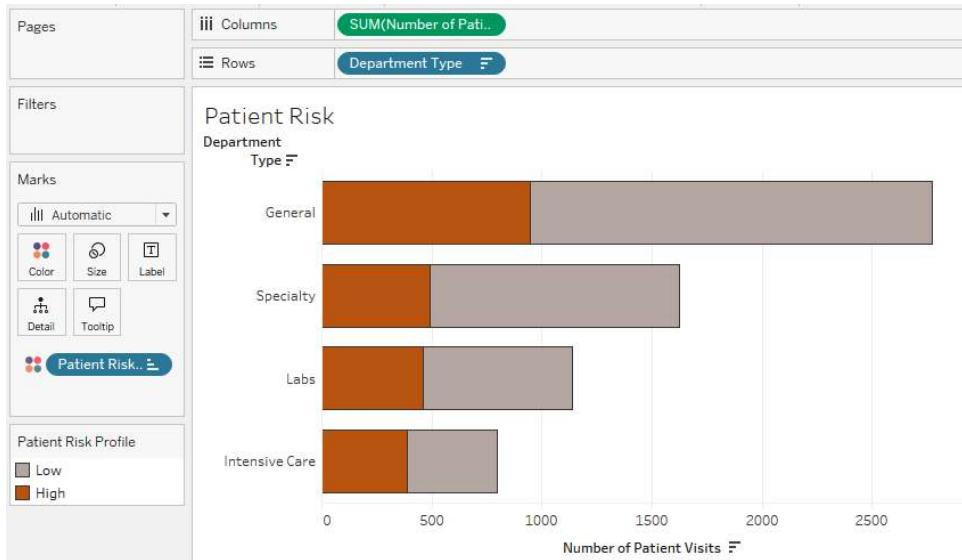


Figure 3.31: A stacked bar chart showing the total number of patients per department and the breakdown of low and high risk

This gives a decent view of the visits for each department type. We can tell that more people visit the general departments and that the number of high-risk patients for both **Specialty** and **Labs** is about the same. **Intensive Care** sees fewer high-risk patients and fewer patients overall. But this is only part of the story.

Consider a stacked bar that doesn't give the absolute value, but gives percentages for each type of department:

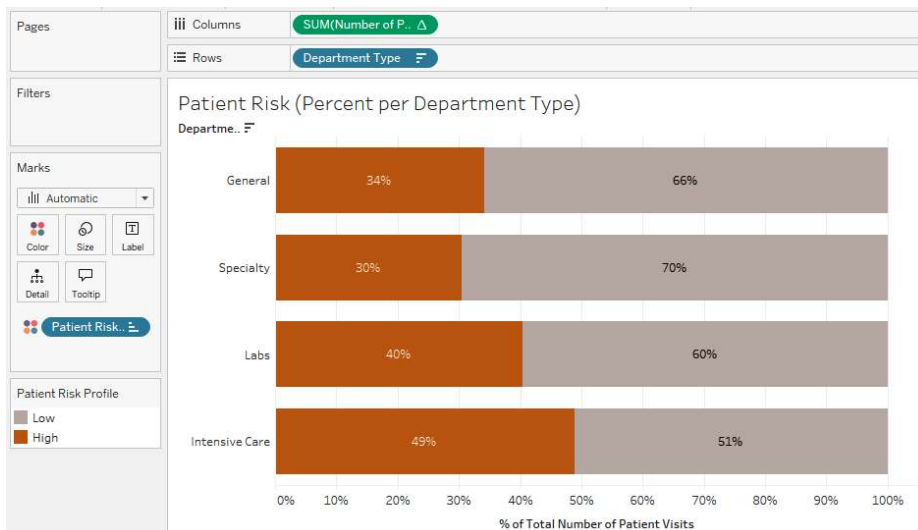


Figure 3.32: A stacked bar chart showing the relative number of high-risk and low-risk patients per department

Compare the previous two stacked bar charts. The fact that nearly 50% of patients in **Intensive Care** are considered **High** risk is evident in both charts. However, the second chart makes this immediately obvious.

None of the data has changed between the two charts, but the bars in the second chart represent the percentage of the total for each type of department. You can no longer compare the absolute values, but comparing the relative breakdown of each department type has been made much easier. Although there are fewer patients in **Intensive Care**, a much higher percentage of them are in the high-risk category.

Let's consider how the preceding charts can be created and even combined into a single visualization in Tableau. We'll use a special kind of calculation called a quick table calculation. We'll cover table calculations in extensive detail in *Chapter 6, Diving Deep with Table Calculations*. Here, simply follow these steps:

1. Create a stacked bar chart by placing **Department Type** on **Rows**, **Number of Patient Visits** on **Columns**, and **Patient Risk Profile** on **Color**. You'll now have a single stacked bar chart.
2. Sort the bar chart in descending order.
3. Duplicate the **Number of Patient Visits** field on **Columns** by holding down *Ctrl* while dragging the **Number of Patient Visits** field in the view to a new spot on **Columns**, immediately to the right of its current location. Alternatively, you can drag and drop the field from the data pane to **Columns**. At this point, you have two **Number of Patient Visits** axes, which, in effect, duplicate the stacked bar chart:

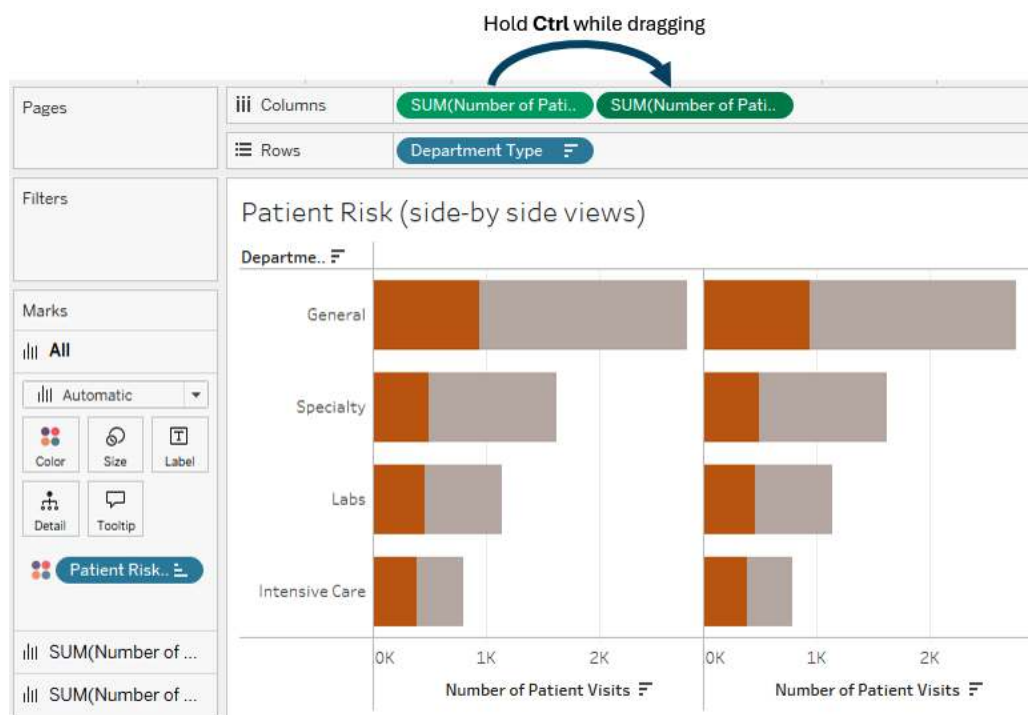


Figure 3.33: An interim step when creating the stacked bars

- Using the drop-down menu of the second **Number of Patient Visits** field, select **Quick Table Calculation | Percent of Total**. This table calculation runs a secondary calculation on the values that were returned from the data source to compute a percentage of the total. Here, you will need to further specify how that total should be computed.
- Using the same drop-down menu, select **Compute Using | Patient Risk Profile**. This tells Tableau to calculate the percentage of each **Patient Risk Profile** within a given department. This means that the values will add up to 100% for each department.
- Turn on labels by clicking the T (Text/label) button from the top-level toolbar. This turns on default labels for each mark:

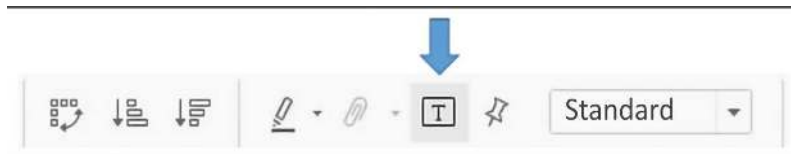


Figure 3.34: This toolbar button toggles text labels on/off

After following the preceding steps, your completed stacked bar charts should appear as follows:

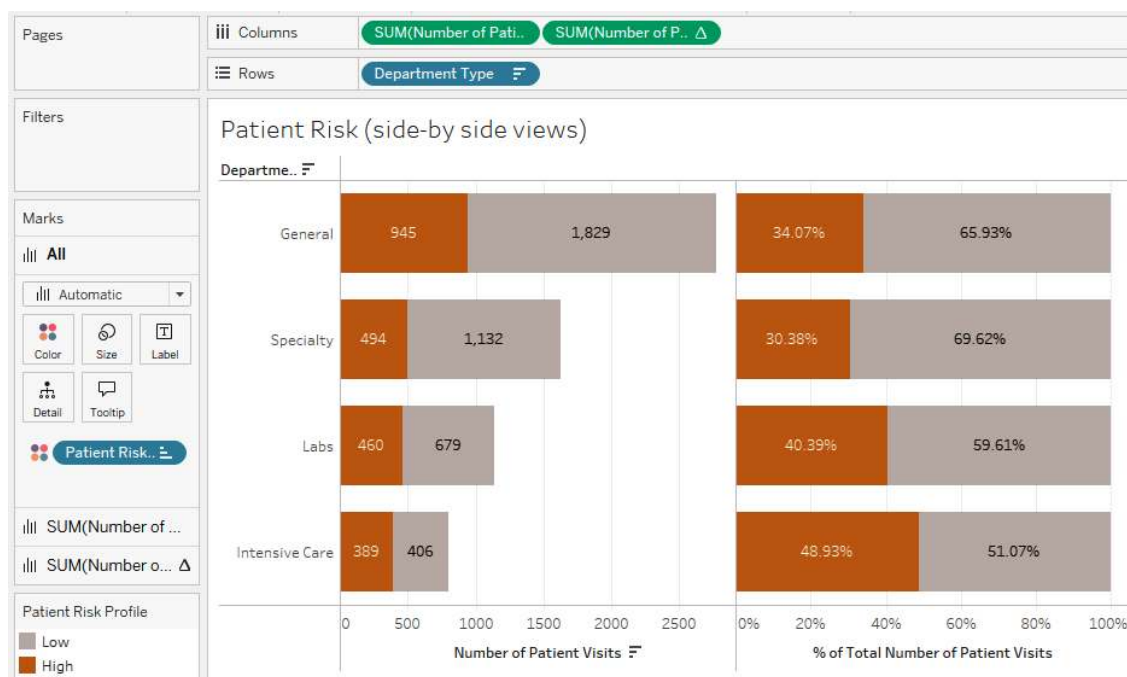


Figure 3.35: The final stacked bar view with absolute and relative values

Using both the absolute values and percentages in a single view can reveal significant details that might be obscured with only one of the charts.

Treemaps

Treemaps use a series of nested rectangles to display parts of the whole, especially within hierarchical relationships. Treemaps are particularly useful when you have hierarchies and dimensions with **high cardinality** (a high number of distinct values).

Here is an example of a treemap that shows the number of days spent in the hospital by patients. The largest rectangle sections show **Department Type**. Within them are departments, doctors, and patients:

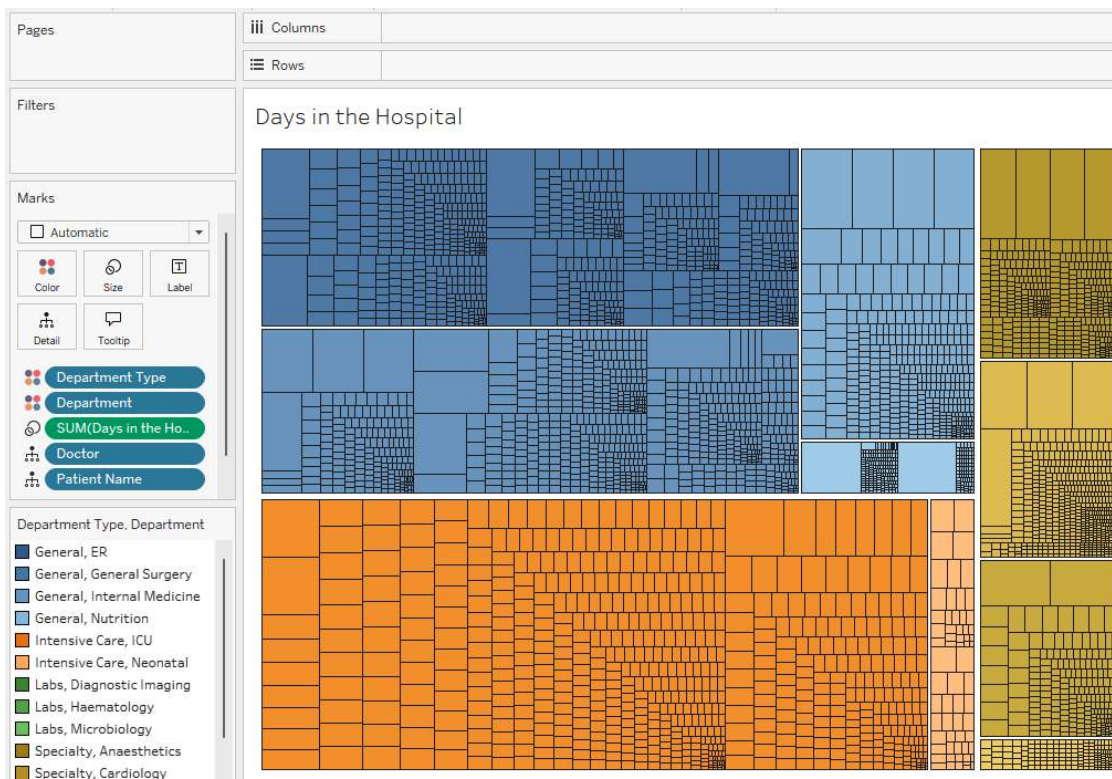


Figure 3.36: A treemap showing the part-to-whole relationship of Department Types/Departments/Doctors/Patients

To create a treemap, you simply need to place a measure on the **Size** shelf and a dimension on the **Detail** shelf. You can add additional dimensions to the level of detail to increase the detail of the view. Tableau will add borders of varying thickness to separate the levels of detail that are created by multiple dimensions. Note that in the preceding view, you can easily see the division of department types, then departments, then doctors, and finally individual patients. You can adjust the border of the lowest level by clicking the **Color** shelf.

The order of the dimensions on the **Marks** card defines the way the treemap groups the rectangles. Additionally, you can add dimensions to rows or columns to slice the treemap into multiple treemaps. The result is effectively a bar chart of treemaps:

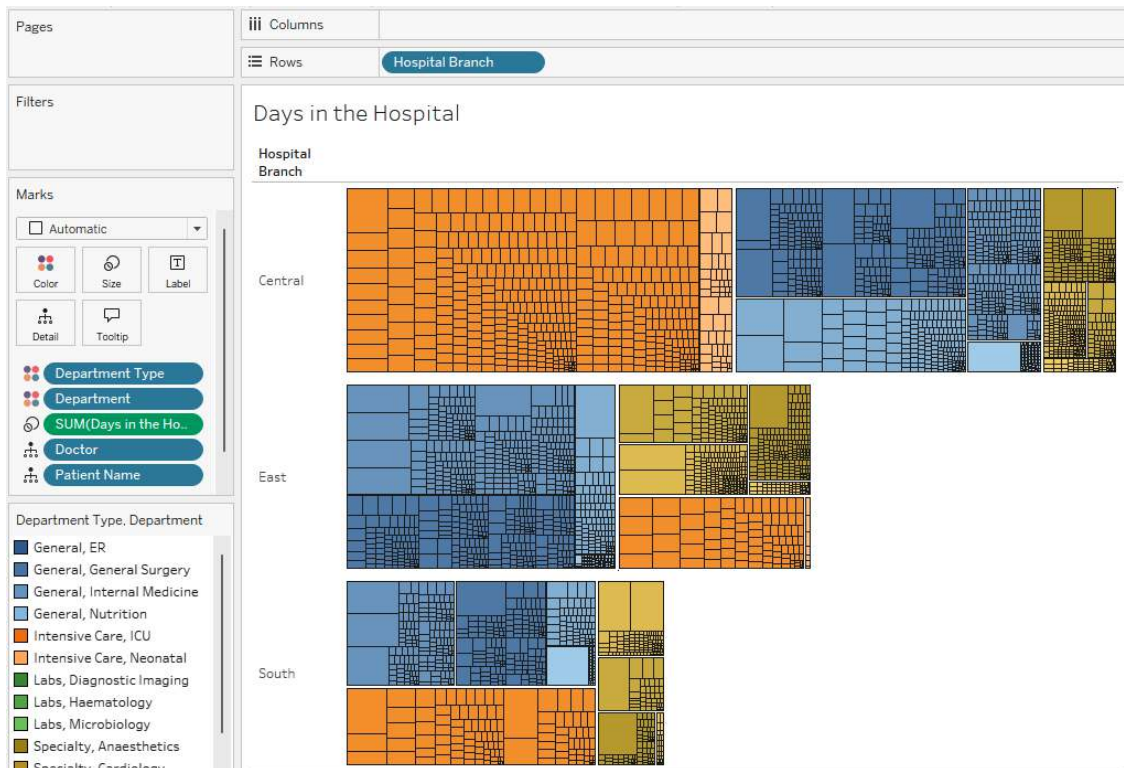


Figure 3.37: Adding a dimension to Rows has effectively made a bar chart of treemaps

The preceding treemap not only demonstrates the ability to have multiple rows (or columns) of treemaps—it also demonstrates the technique of placing multiple fields on the **Color** shelf. This can only be done with discrete fields. You can assign two or more colors by holding down the *Shift* key while dropping the second field on color. Alternatively, the icon or space to the left of each field on the **Marks** card can be clicked to change which shelf is used for the field:

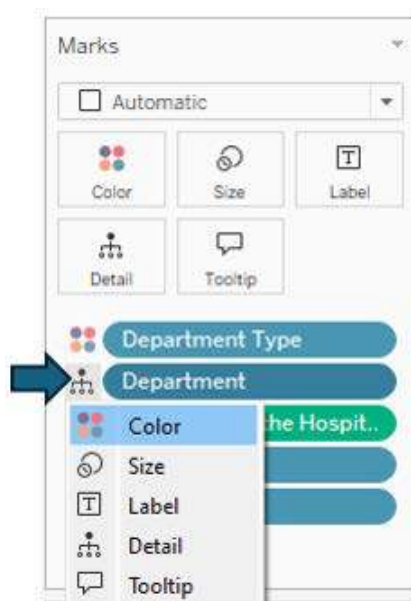


Figure 3.38: Clicking the icon next to a field on the Marks card allows you to change which shelf is used



Treemaps, along with packed bubbles, word clouds, and a few other chart types, are called **non-Cartesian** chart types. This means they are drawn without an x or y axis, and do not even require row or column headers. To create any of these chart types, do the following:

- Make sure that no continuous fields are used on **Rows** or **Columns**
- Use any field as a measure on **Size**
- Change the mark type based on the desired chart type: square for treemap, circle for packed bubbles, or text for word cloud (with the desired field on **Label**)

Area charts

Take a line chart and then fill in the area beneath the line. If there are multiple lines, then stack the filled areas on top of each other. That's how you might think of an area chart.

In fact, in Tableau, you may find it easier to create a line chart, like you've done previously, and then change the mark type on the **Marks** card to **Area**. Any dimensions on the **Color**, **Label**, or **Detail** shelves will create slices of the area that will be stacked on top of each other. The **Size** shelf is not applicable to an area chart.

As an example, consider a visualization of patient visits over time, segmented by hospital branch:

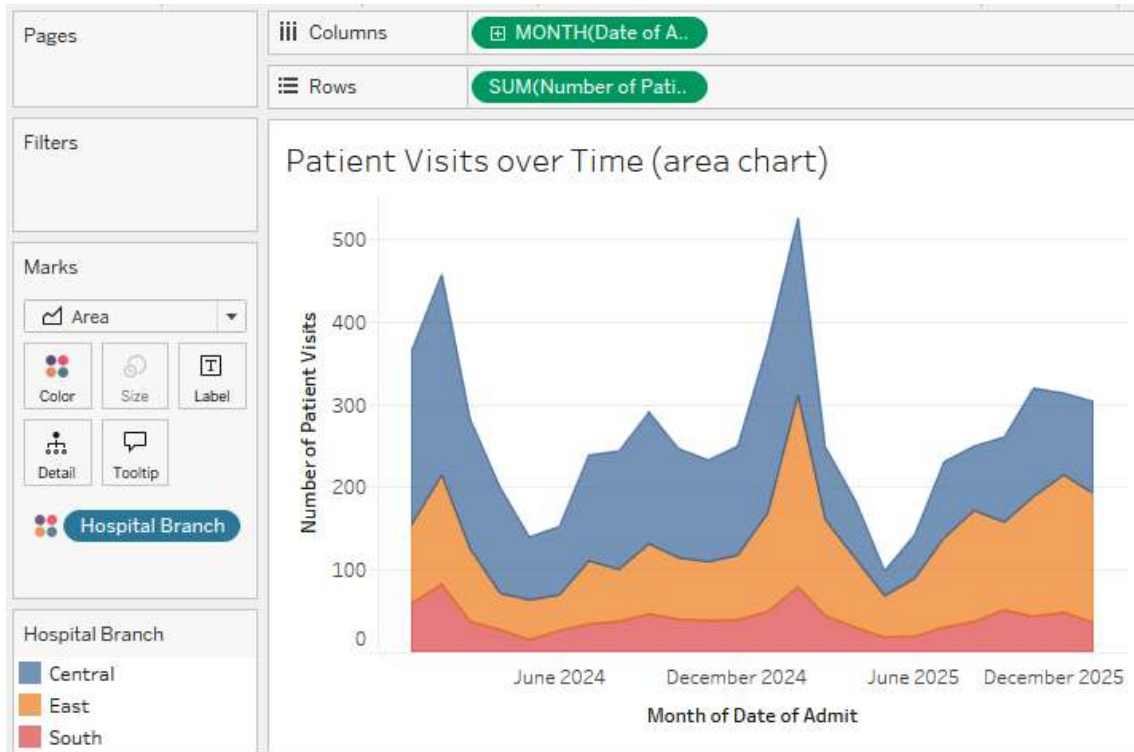


Figure 3.39: An area chart showing patient visits over time by hospital branch

Each band represents a different hospital branch location. In many ways, the view is aesthetically pleasing, and it does highlight some patterns in the data. However, it suffers from some of the same weaknesses as the stacked bar chart. Only the bottom band (**South**) can be read in terms of the values on the axis.

The other bands are stacked on top, and it becomes very difficult to compare. For example, it is obvious that there is a spike around February of each year. But is it in each band? Or is one of the lower bands pushing the higher bands up? Which band has the most significant spike?

Now, consider the following view:

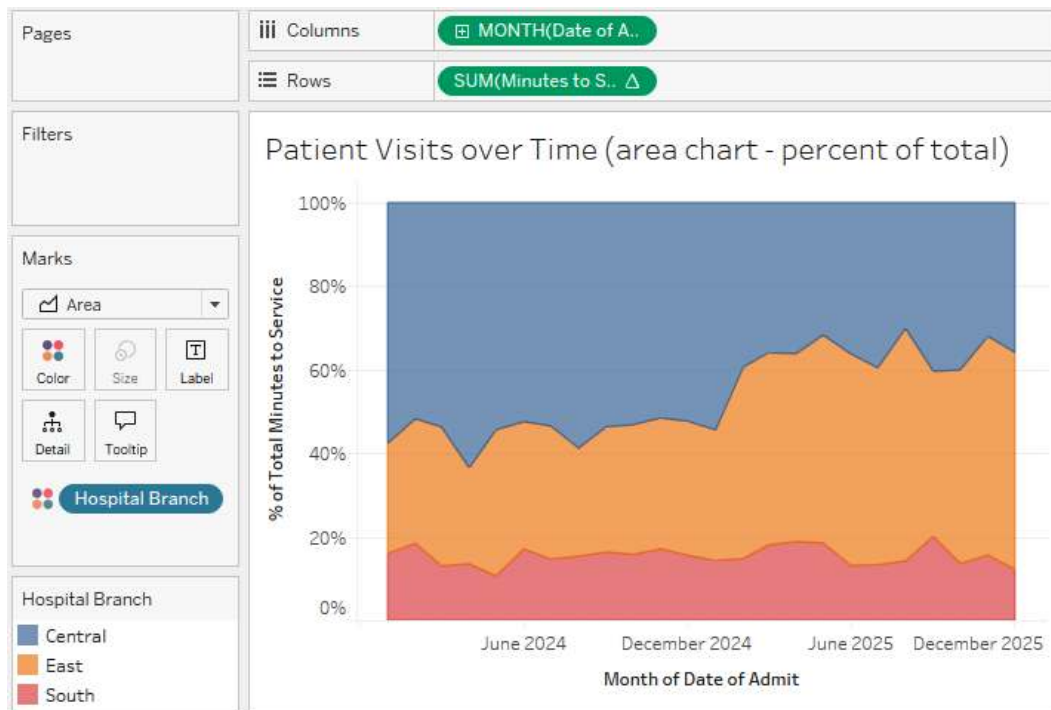


Figure 3.40: An area chart showing percentages instead of absolute values

This view uses a quick table calculation, like the stacked bars example. It is no longer possible to see the spikes, as in the first chart. However, it is much easier to see that there was a dramatic increase in the percentage of patients seen by the **East** branch (the middle band) around February 2024, and that the branch continued to see a significant number of patients through the end of the year.

It is important to understand what facets of the data story are emphasized (or hidden) by selecting a different chart type. You might even experiment in the Chapter 3 workbook by changing the first area chart to a line chart. You may notice that you can see the spikes as well as the absolute increase and decrease in patient visits per branch. Each chart type contributes to a certain aspect of the data story.



You can define the order in which the areas are stacked by changing the sort order of the dimensions on the shelves of the **Marks** card. Additionally, you can rearrange them by dragging and dropping them within **Color Legend** to further adjust the order.

Pie charts

Pie charts can also be used to show part-to-whole relationships. To create a pie chart in Tableau, change the mark type to **Pie**. This will give you an **Angle** shelf, which you can use to encode a measure. The dimension(s) you place on the **Marks** card (typically, on the **Color** shelf) will define the slices of the pie:

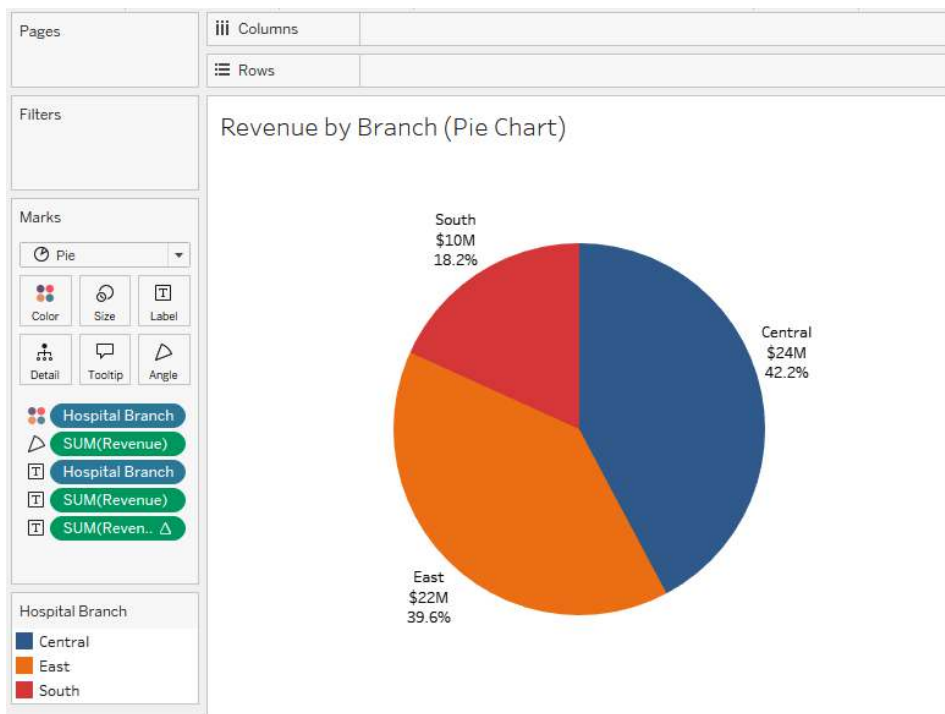


Figure 3.41: A pie chart showing total revenue broken down by branch

Observe that the preceding pie chart uses the sum of total revenue to define the angle of each slice; the higher the sum, the wider the slice. The **Hospital Branch** dimension is slicing the measure and defining slices of the pie. This view also demonstrates the ability to place multiple fields on the **Label** shelf. The second **SUM(Revenue)** field is the percentage of the total table calculation you saw previously. This allows you to see the absolute values of revenue, as well as a percentage of the whole.



Pie charts can work well with a few slices. In most cases, more than two or three become very difficult to see and understand. Also, as a good practice, sort the slices by sorting the dimension that defines the slices. In the preceding example, the **Hospital Branch** dimension was sorted by the SUM of revenue descending. This was done by using the drop-down menu option. This causes slices to be ordered from largest to smallest and allows anyone reading the chart to easily see which slices are larger, even when the size and angles are nearly identical.

With a good understanding of some techniques for visualizing part-to-whole relationships, let's move on to visualizing distributions.

Visualizing distributions

Often, simply understanding totals, sums, and even the breakdown of part-to-whole only gives a piece of the overall picture. Most of the time, you'll want to understand where individual items fall within a distribution of all similar items.

You might find yourself asking questions such as the following:

- How much does each customer spend at our stores and how does that compare to all other customers?
- How long do most of our patients stay in the hospital? Which patients fall outside the normal range?
- What's the average life expectancy of components in a machine, and which last more than average? Are there any components with extremely long or extremely short lives?
- How far above or below passing were students' test scores?

These questions have similarities. In each case, you seek an understanding of how individuals (customers, patients, components, and students) relate to the group. In each case, you most likely have a relatively high number of individuals. In data terms, you have a dimension (customer, patient, component, and student) representing a relatively large population of individuals and some measure (amount spent, length of stay, life expectancy, and test score) you'd like to compare. Using one or more of the following visualizations might be a good way to do this.

Circle charts

Circle charts are one way to visualize distributions. Consider the following view, which shows how each doctor compares to other doctors within the same type of department in terms of the average days their patients stay in the hospital:

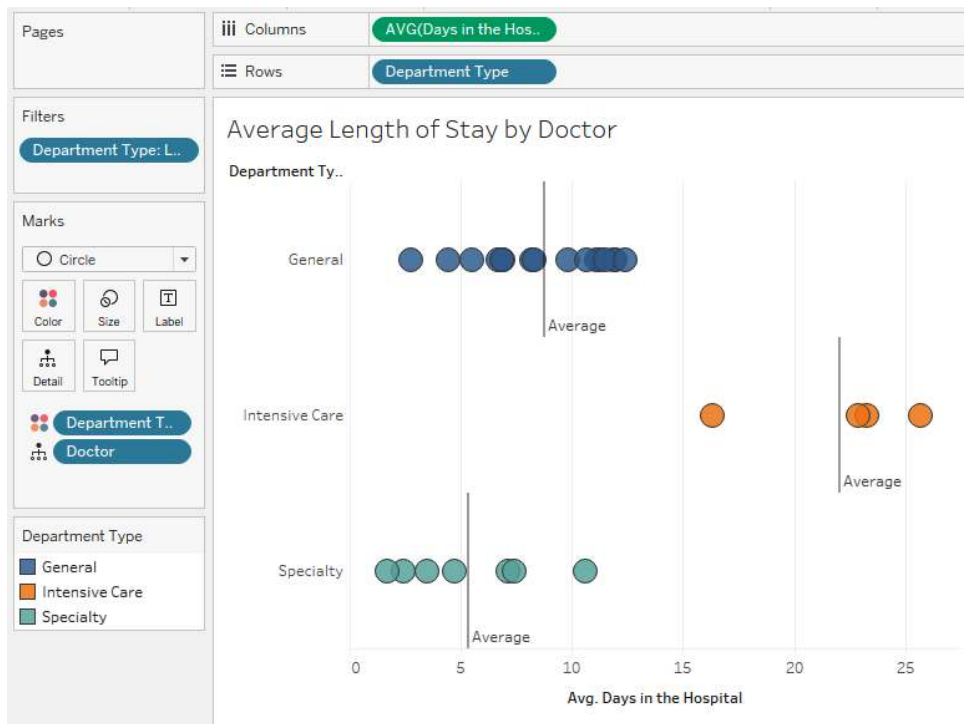


Figure 3.42: A circle chart showing the average length of stay for each doctor within each department type

Here, you can see which doctors have patients who stay in the hospital for longer or shorter durations than the average. It is also interesting to note that certain types of departments have longer average lengths of stay than others. This makes sense as each type of department has patients with different needs. It's probably not surprising that patients in **Intensive Care** tend to stay longer. Certain departments may have different goals or requirements. Being able to evaluate doctors within their type of department makes comparisons far more meaningful.

To create the preceding circle chart, you need to place the fields on the shelves that are shown and then simply change the mark type from **Automatic** (which was a bar mark) to **Circle**. **Department Type** defines the rows, and each circle is drawn at the level of **Doctor**, which is in the level of **Detail** on the **Marks** card.

Finally, to add the average lines, simply switch to the **Analytics** tab of the left-hand pane and drag **Average Line** to the view, specifically dropping it on the **Cell** option:



Figure 3.43: You can add reference lines and more by dragging from the Analytics tab to the view

You may also click one of the resulting average lines and select **Edit** to find fine-tuning options such as labeling, tooltips, and formatting.

Jittering

When using views such as circle plots or other similar visualization types, you'll often see that marks overlap, which can obscure parts of the true story. Do you know for certain, just by looking, how many doctors there are in **Intensive Care** who are above average? How many are below? Or could there be two or more circles exactly overlapping? One way of minimizing this is to click the **Color** shelf and add some transparency and a border to each circle. Another approach is a technique called **jittering**.



Jittering is a common technique in data visualization that involves adding a bit of intentional noise to a visualization to prevent overlap without harming the integrity of what is being communicated. Alan Eldridge and Steve Wexler are among those who have pioneered techniques for jittering in Tableau.

Various jittering techniques, such as using the `Index()` and `Random()` functions, can be found by searching for *jittering* on the Tableau forums or for *Tableau jittering* using a search engine.

Here is one approach that uses the `Index()` function, computed along **Doctor**, as a continuous field on **Rows**. Since `INDEX()` is continuous (green), it defines an axis and causes the circles to spread out vertically.

Now, you can more clearly see each individual mark and be more confident that the overlap is not obscuring the true picture of the data:



Figure 3.44: Here, `INDEX()` has been added as a continuous field on Rows (the table calculation is computed along Doctor)



In the preceding view, the vertical axis that was created by the **Index** field is hidden. You can hide an axis or header by using the drop-down menu of the field defining the axis or header and unchecking **Show Header**. Alternatively, you can right-click any axis or header in the view and select the same option.

You can use jittering techniques on many kinds of visualizations that involve plotting fixed points that could theoretically overlap, such as dot plots and scatterplots. Next, we will move on to another useful distribution visualization technique: box and whisker plots.

Box and whisker plots

Box and whisker plots (sometimes just called **box plots**) add additional statistical context to distributions. To understand a box and whisker plot, refer to the following diagram:

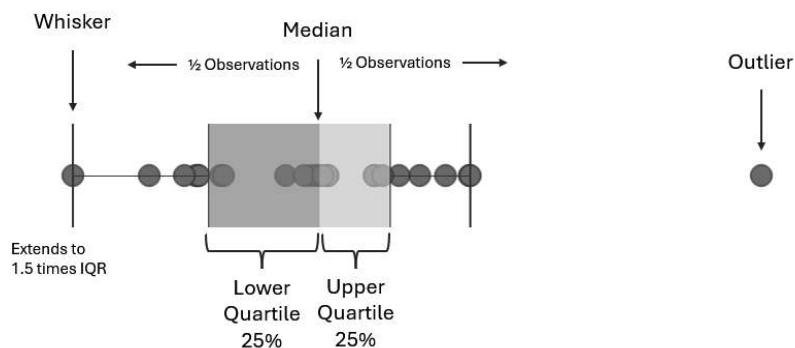


Figure 3.45: Explanation of a box and whisker plot

Here, the box plot has been added to a circle graph. The box is divided by the median, meaning that half of the values are above the median value, and half are below. The box also indicates the lower and upper quartiles, which each contain a quarter of the values. The span of the box makes up what is known as the **Interquartile Range (IQR)**. The whiskers extend to 1.5 times the IQR value (or the maximum extent of the data). Any marks beyond the whiskers are outliers.

To add box and whisker plots, use the **Analytics** tab on the left-hand sidebar and drag **Box Plot** to the view. Doing this to the circle chart we saw in Figure 3.42 yields the following chart:



Figure 3.46: A box plot applied to the previous circle chart

The box plots help us to see and compare the medians, the ranges of data, the concentration of values, and any outliers. You may edit box plots by clicking or right-clicking the box or whisker and selecting **Edit**. This will reveal multiple options, including how whiskers should be drawn, whether only outliers should be displayed, and other formatting possibilities.

Histograms

Another possibility for showing distributions is to use a histogram. A histogram looks similar to a bar chart, but the bars show a count of occurrences of a value. For example, standardized test auditors looking for evidence of grade tampering might construct a histogram of student test scores.

Typically, a distribution might look like the following example (this is not included in the workbook):

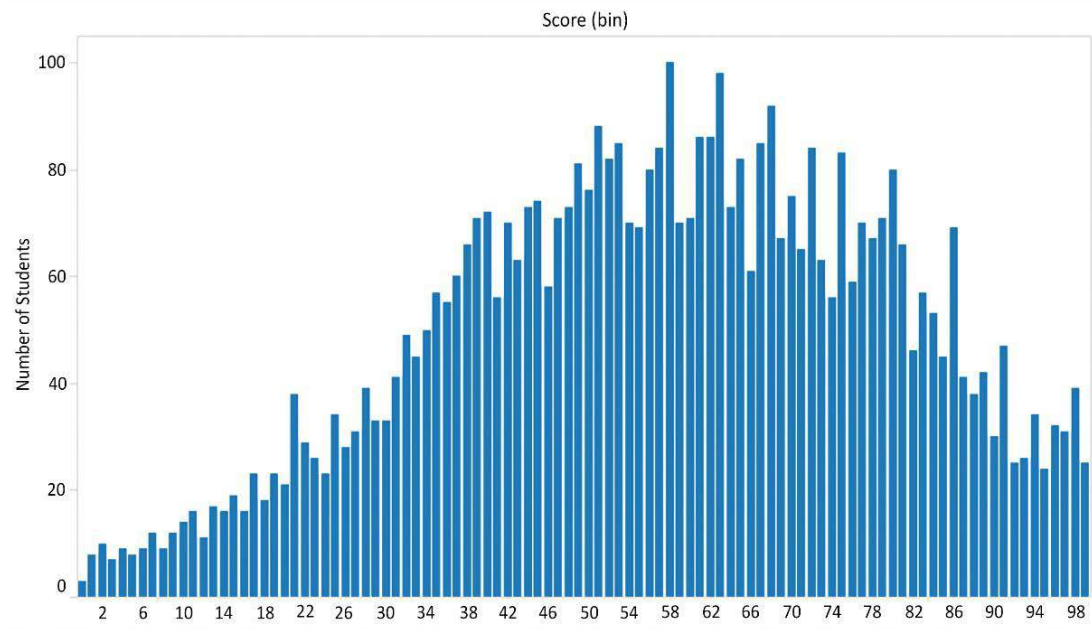


Figure 3.47: A histogram of test scores

The test scores are shown on the x axis, and the height of each bar shows the number of students who achieved that particular score. A typical distribution often has a recognizable bell curve. In this case, some students are doing poorly and some are doing extremely well, but most have scores somewhere in the middle.

What if auditors saw something like this?

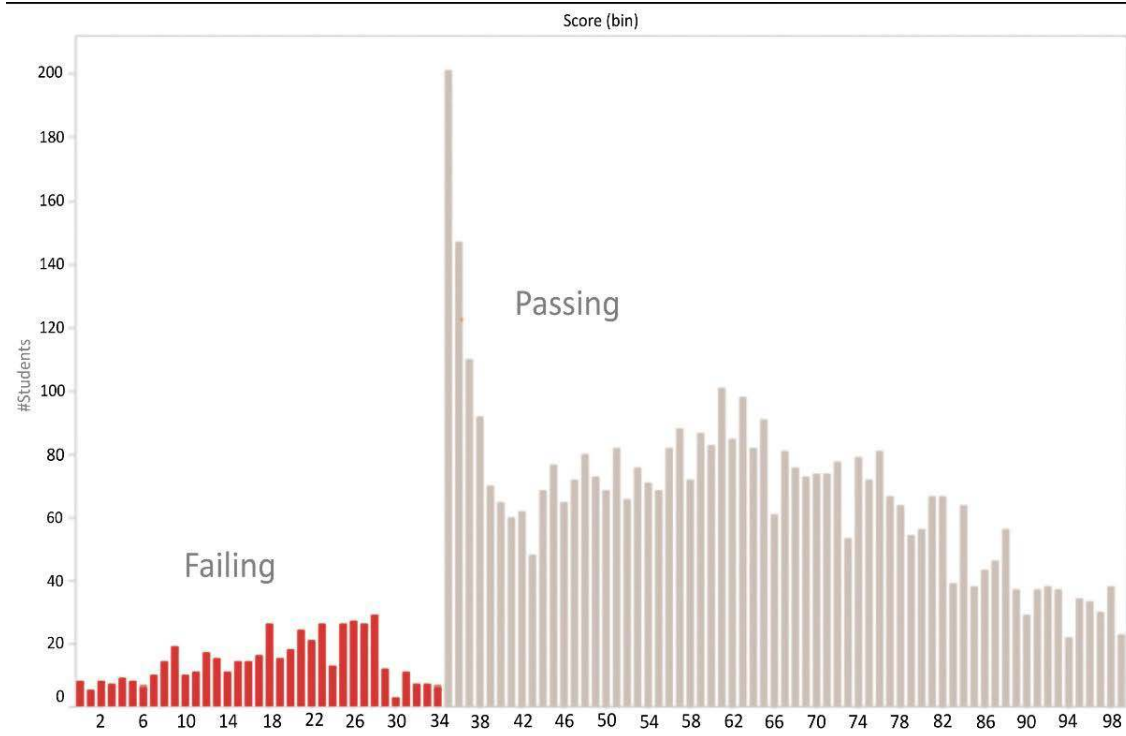


Figure 3.48: A histogram that does not have a typical bell curve, raising some questions

Something is clearly wrong. Perhaps graders have bumped up students who were just shy of passing to barely passing. It's also possible this may indicate bias in subjective grading instead of blatant tampering. We shouldn't jump to conclusions, but the pattern is not normal and requires investigation. Histograms are very useful in catching anomalies like this. We'll see more examples of deeper analysis in *Chapter 9, Visual Analytics: Trends, Clustering, Distributions, and Forecasting*.

Now that we've seen an example of histograms, let's shift our focus back to the hospital data and work through an example. What if you want to visualize the time it takes to begin patient treatment so that you can observe the patterns for different patient populations? You might want to create a histogram. In this example, we'll start with a blank view and follow these steps:

1. Click to select the **Minutes to Service** field under **Measures** in the data pane.
2. Expand **Show Me** if necessary and select the histogram.

Upon selecting the histogram, Tableau builds the chart by creating a new dimension, **Minutes to Service (bin)**, which is used in the view, along with a COUNT of **Minutes to Service** to render the view:

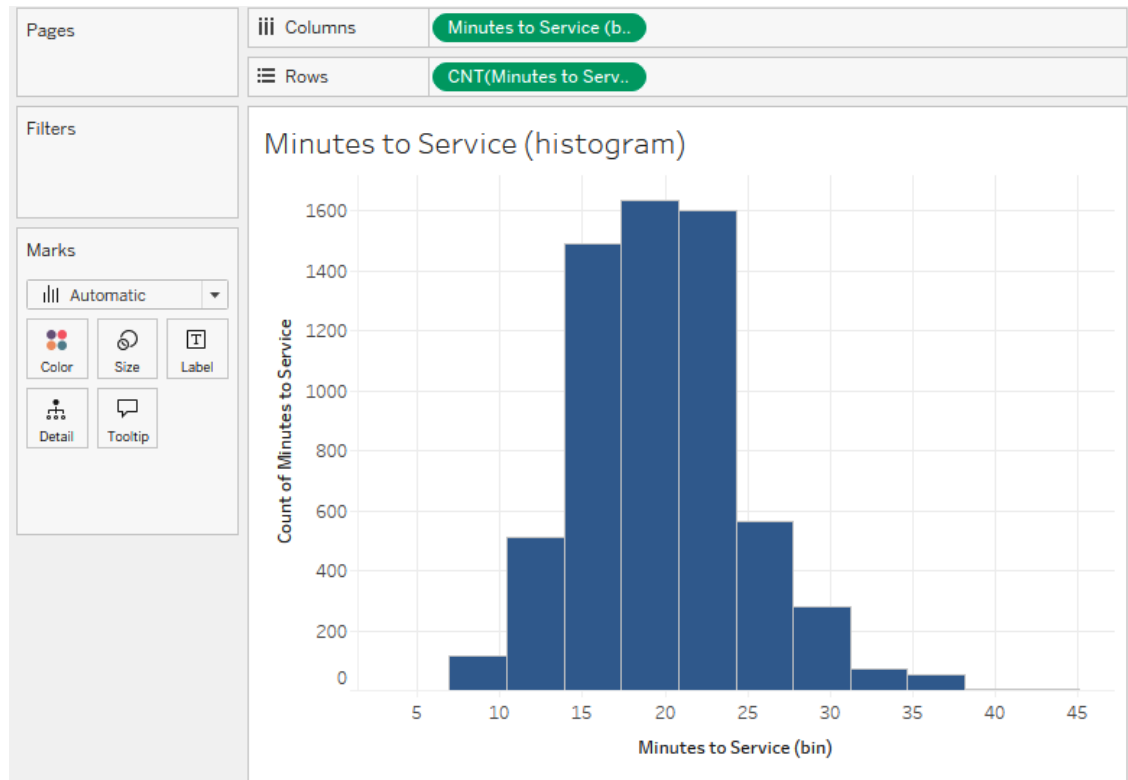


Figure 3.49: A histogram showing the distribution of patients according to minutes to service



Bins are ranges of measure values that can be used as dimensions to slice the data. You can think of bins as buckets. For example, you might look at test scores by 0–5%, 5–10%, and so on, or people’s ages by 0–10, 10–20, and so on. You can set the size, or range, of the bin when it is created and edit it at any point. Tableau will also suggest a size for the bin based on an algorithm that looks at the values that are present in the data. Tableau will use uniform bin sizes for all bins.

For this view, Tableau automatically set the bin size to 3.47 minutes, which is not very intuitive. Experiment with different values by right-clicking or using the drop-down menu in the **Minutes to Service (bin)** field in the data pane and selecting **Edit**. The resulting window gives some information and allows you to adjust the size of the bins:

Edit Bins [Minutes to Service]

New field name: Minutes to Service (bin)

Size of bins: 3.47

Suggest Bin Size

Range of Values:

Min: 4

Diff: 40

Max: 44

CntD: 40

OK

Cancel

Figure 3.50: Options for editing a bin

Here, for example, is the same histogram with each bin sized to 2 minutes:

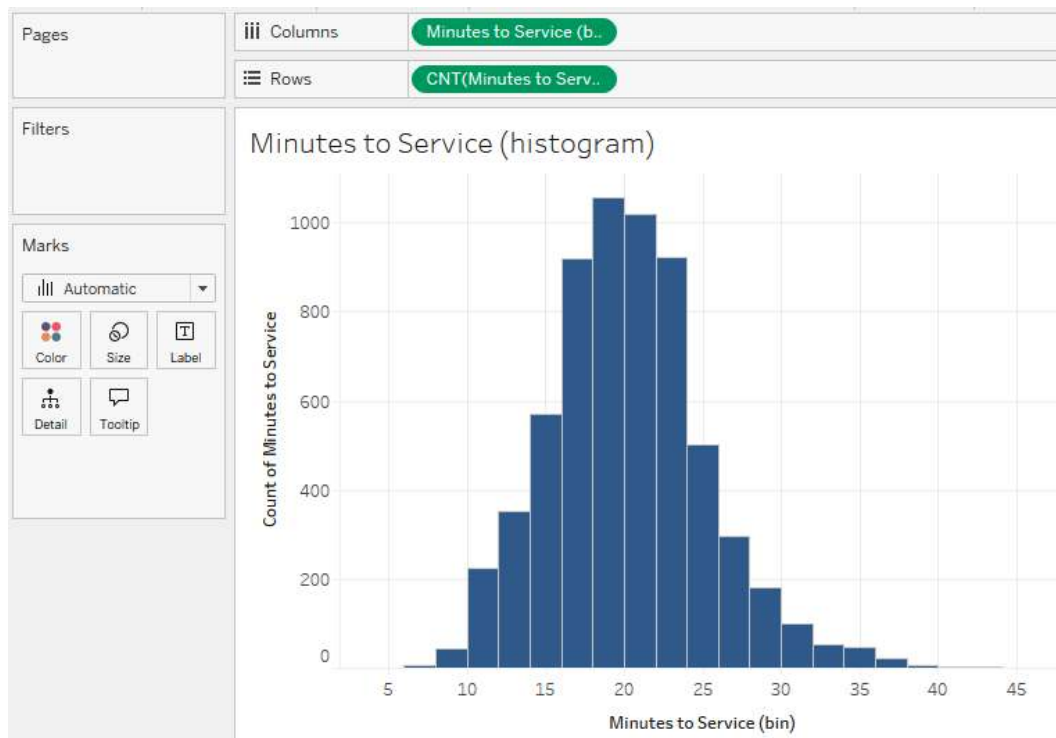


Figure 3.51: A histogram with a bin size of 2

You can see the curve, which peaks at just under 20 minutes and then tapers off with a few patients having to wait as long as 40 minutes.

You might pursue additional analysis, such as seeing how wait times vary for the majority of patients based on their risk profile, such as in this view:

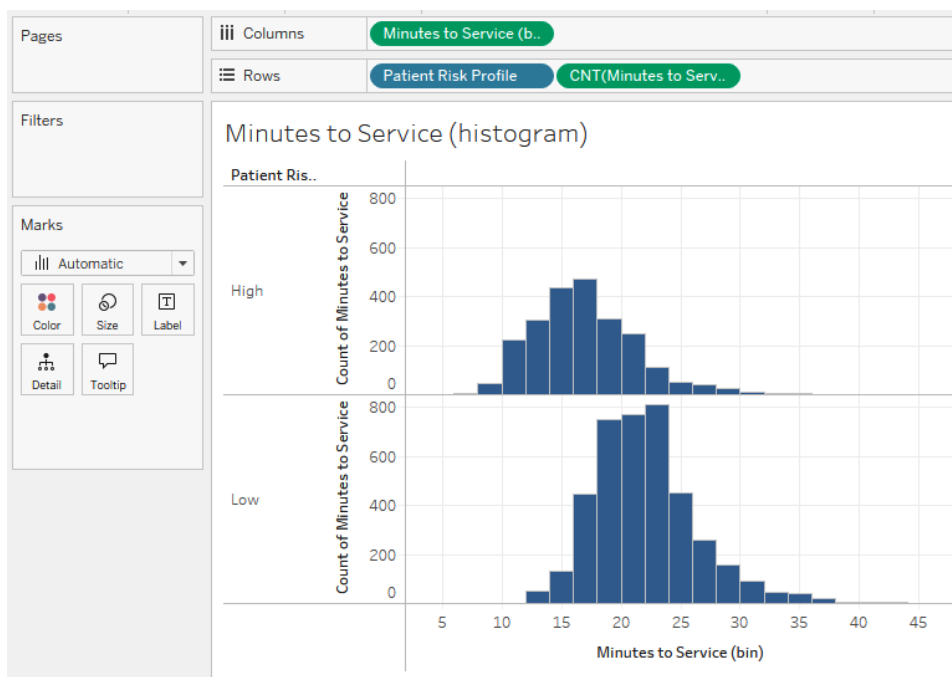


Figure 3.52: Patient risk profile creates two rows of histograms, showing that most high-risk patients receive faster care (as we would hope)



You can create new bins on your own by right-clicking a numeric field and selecting **Create | Bins**. You can edit the size of bins by selecting the **Edit** option for the bin field.

You'll also want to consider what you want to count for each bin and place that on **Rows**. When you used **Show Me**, Tableau placed the **COUNT** of **Minutes to Service** on **Rows**, which is just a count of every record where the value was not null. In this case, that's equivalent to a count of patient visits because the dataset contains one record per visit. However, if you wanted to count the number of unique patients, you might consider replacing the field in the view with `COUNTD([Patient ID])`.



Just like dates, when the bin field in the view is discrete, the drop-down menu includes an option for **Show Missing Values**. If you use a discrete bin field, you may wish to use this option to avoid distorting the visualization and to identify what values don't occur in the data.

We've seen how to visualize distributions with circle plots, histograms, and box plots. Let's turn our attention to using multiple axes to compare different measures.

Visualizing multiple axes to compare different measures

Often, you'll need to use more than one axis to compare different measures, understand the correlation, or analyze the same measure at different levels of detail. In these cases, you'll use visualizations with more than one axis.

Scatterplot

A scatterplot is an essential visualization type for understanding the relationship between two measures. Consider a scatterplot when you find yourself asking questions like the following:

- Does how much I spend on marketing really make a difference in sales?
- How much does power consumption go up with each degree of heating/cooling?
- Is there any correlation between hours of study and test performance?

Each of these questions seeks to understand the correlation (if any) between two measures. Scatterplots help you understand these relationships and see any outliers.

Consider the following scatterplot, which looks for a relationship between the total revenue for a department and the number of patient visits:

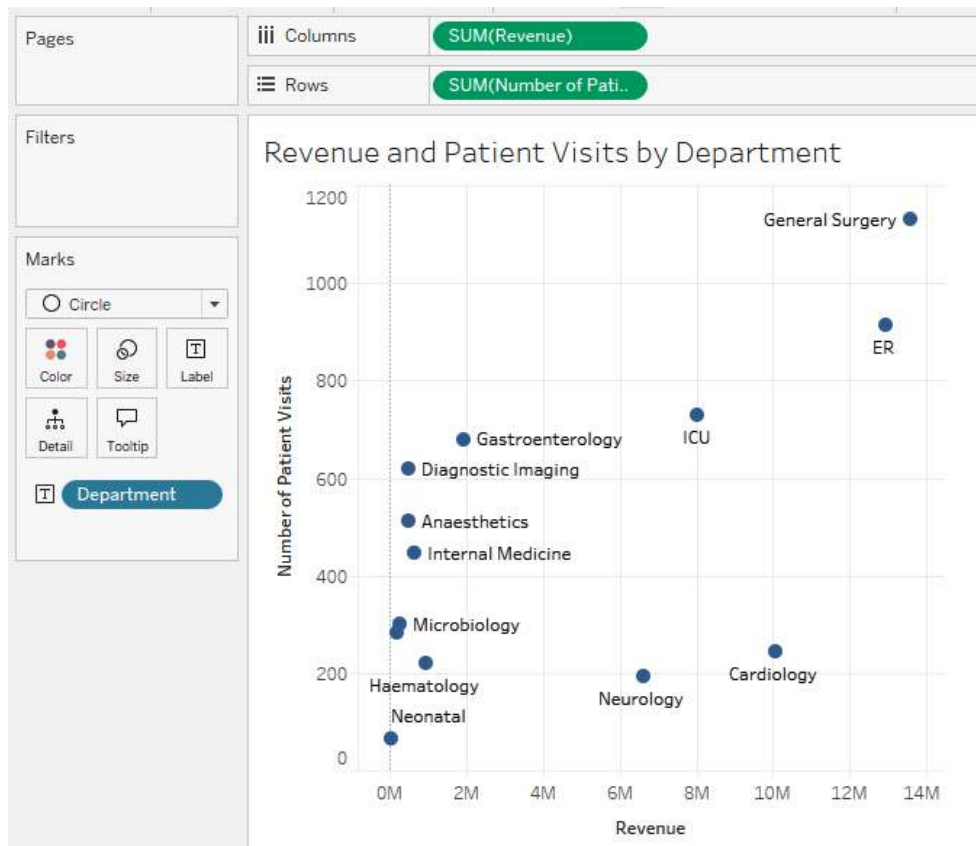


Figure 3.53: A scatterplot showing the correlation between Revenue and Number of Patient Visits

The **Department** dimension is on the **Text** shelf and defines the view level of detail. Each mark in the view represents the total **Revenue** and total **Number of Patient Visits** for the department.

As you would expect, most departments have higher revenue with a higher volume of patients. **Neurology** and **Cardiology** stand out as having high revenue despite a low volume of patients.



Look for ways to use **Size** and **Color** to encode additional data in scatterplots. For example, we might add **Department Type** to **Color** to see if departments of the same type show similar correlations. We might encode **Size** with the average length of stay to see if higher revenues can also be explained by patients who stay longer.

Let's consider a few other types of charts that use multiple axes.

Dual-axis and combination charts

One very important feature of Tableau is the ability to use a dual axis. Scatterplots use two axes, but they are *x* and *y*. You also observed in the stacked bar example that placing multiple continuous (green) fields next to each other on **Rows** or **Columns** results in multiple side-by-side axes. Dual axis, on the other hand, means that a view is using two axes that are opposite each other with a common pane.

Here is an example (not included in the workbook) of using a dual axis for **Sales** and **Profit**:

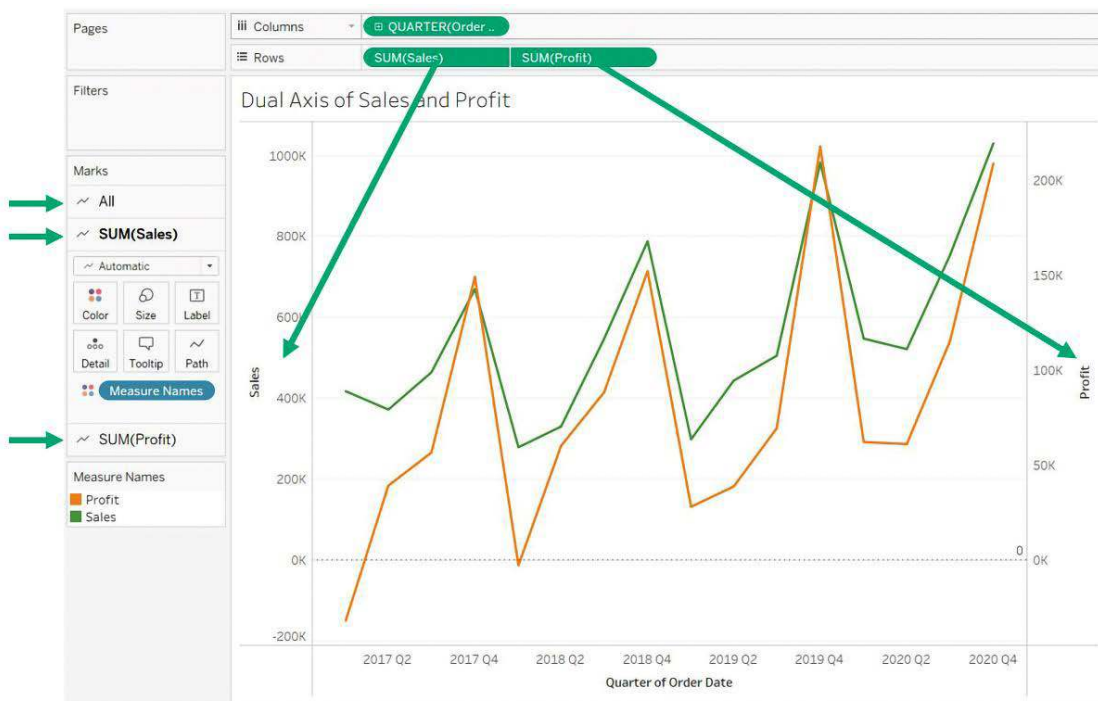


Figure 3.54: A dual-axis chart with an indication of which field defines which axis

There are several key features of the view, which are as follows:

- The **Sales** and **Profit** fields on **Rows** indicate that they have a dual axis by sharing a flattened side.
- The axes defined by **Sales** and **Profit** are on opposing sides of the view. Also, note that they are not synchronized, which, in many cases, can give a distorted view of the data. It would be great if profit was that close to total sales! But it's not. To synchronize the axes, right-click the right axis and select **Synchronize Axis**. If that option is grayed out, it is likely that one of the values is a whole number type and the other is a decimal type. You can change the data type of one of the fields by right-clicking on it in the data pane and selecting **Change Data Type | Number (Whole) or Number (Decimal)**.
- The **Marks** card is now an accordion-like control with an **All** section and a section for **Sales** and **Profit**. You can use this to customize marks for all measures or specifically customize marks for either **Sales** or **Profit**.

To create a dual axis, drag and drop two continuous (green) fields next to each other on **Rows** or **Columns**, then use the drop-down menu on the second, and select **Dual Axis**. Alternatively, you can drop the second field onto the canvas, opposite the existing axis.



Dual axes can be used with any continuous field that defines an axis. This includes numeric fields, date fields, and latitude or longitude fields that define a geographic visualization. In the case of latitude or longitude, simply copy one of the fields and place it immediately next to itself on the **Rows** or **Columns** shelf. Then, select **Dual Axis** by using the drop-down menu.

A **combination chart** extends the use of dual axes to overlay different mark types. This is possible because the **Marks** card will give options for editing all marks or customizing marks for each axis.



Multiple mark types are available any time two or more continuous fields are located beside each other on **Rows** or **Columns**.

As an example of a combination chart, consider the following visualization:

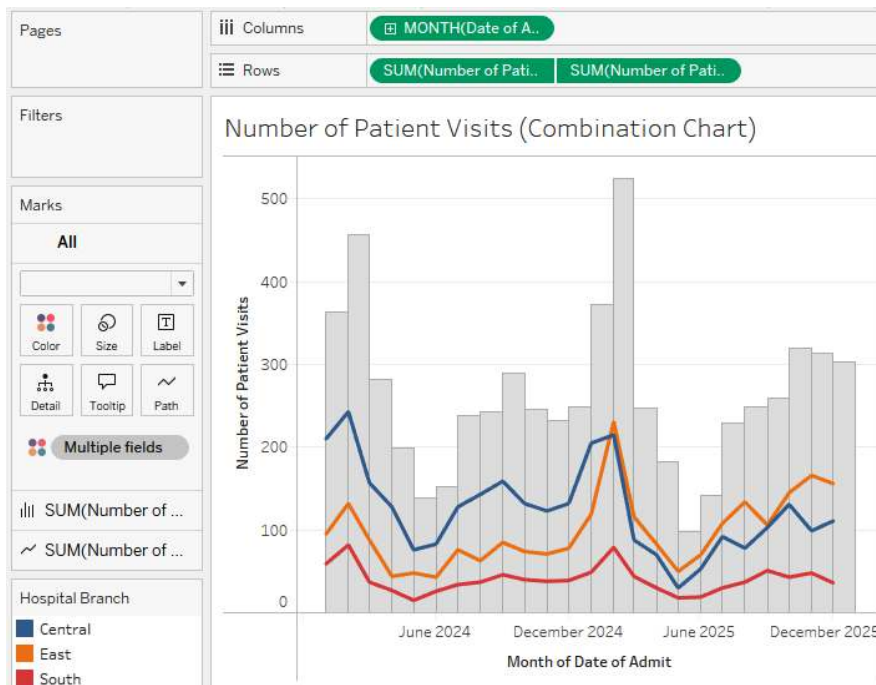


Figure 3.55: A combination chart that shows individual branch visits as lines and total visits as bars

This chart uses a combination of bars and lines to show the total number of patient visits over time (the bars) and the breakdown of patient visits by hospital branch over time (the lines). This kind of visualization can be quite effective at giving additional context to data.

There are several things to note about this view:

- The field on the **Color** shelf is listed as **Multiple Fields** and is gray on the **Marks** card. This indicates that different fields have been used for **Color** for each axis on **Marks**.
- The view demonstrates the ability to mix levels of detail in the same view. The bars are drawn at the highest level (patient visits for each month), while the lines have been drawn at a lower level (patient visits for each branch for each month).
- The view demonstrates the ability to use the same field (**Patient Visits**, in this case) multiple times on the same shelf (**Rows**, in this case).
- The second axis (the **Patient Visits** field on the right-hand side) has the header hidden to remove redundancy from the view. You can do this by unchecking **Show Header** from the drop-down menu on the field in the view or by right-clicking on the axis or header you wish to hide.

Dual-axis and combination charts open a wide range of possibilities for mixing mark types and levels of detail and are very useful for generating unique insights. We'll see a few more examples of these throughout the rest of this book. Be sure to experiment with this feature and let your imagination run wild with all that can be done. Next, we'll turn our attention to an exciting new feature in Tableau: Viz Extensions!

Going further with Viz Extensions

Beyond comparisons, timelines, part-to-whole relationships, and distributions there are other ways to visualize data. For example, you might want to explore relationships represented by the data. Network graphs, chord diagrams, and radial charts can provide insights into hierarchies and connections that are hard to see otherwise. If you want to see flow within your data, Sankey charts are a great option. These visualizations have been possible in Tableau but often required data manipulation or advanced techniques to achieve.

Setting up Viz Extensions

In the most recent versions of Tableau, you can access these visualizations (and many more) as **Viz Extensions**. Viz Extensions are web-based components that can be integrated into Tableau and allow users to easily integrate new chart types into Tableau's drag-and-drop paradigm. They can be used on dashboards and are capable of the same interactivity as any standard Tableau visualization.



Viz Extensions, available in Tableau 2024.2 and later, are written in JavaScript and use Tableau's Extensions API. Many are available directly from Tableau, while others are available from third-party developers. You can even develop your own using the API and any JavaScript visualization library of your choice.

Let's say you want to visualize the flow of patients once they have arrived at the ER in the hospital. You want to know the order of events and how many patients flow from one event to another. You could use bar charts or area charts to see the number of patients over time, but an excellent way to see the flow of patients is to use a Sankey chart. While Sankey is not a standard mark type, you'll notice the option to add a new **Viz Extension**.

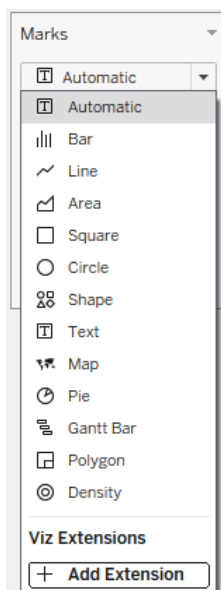


Figure 3.56: The Mark Type dropdown on the Marks card reveals the option to add a Viz Extension

When you click **Add Extension**, a dialog box will prompt you to find the extension you wish to add:

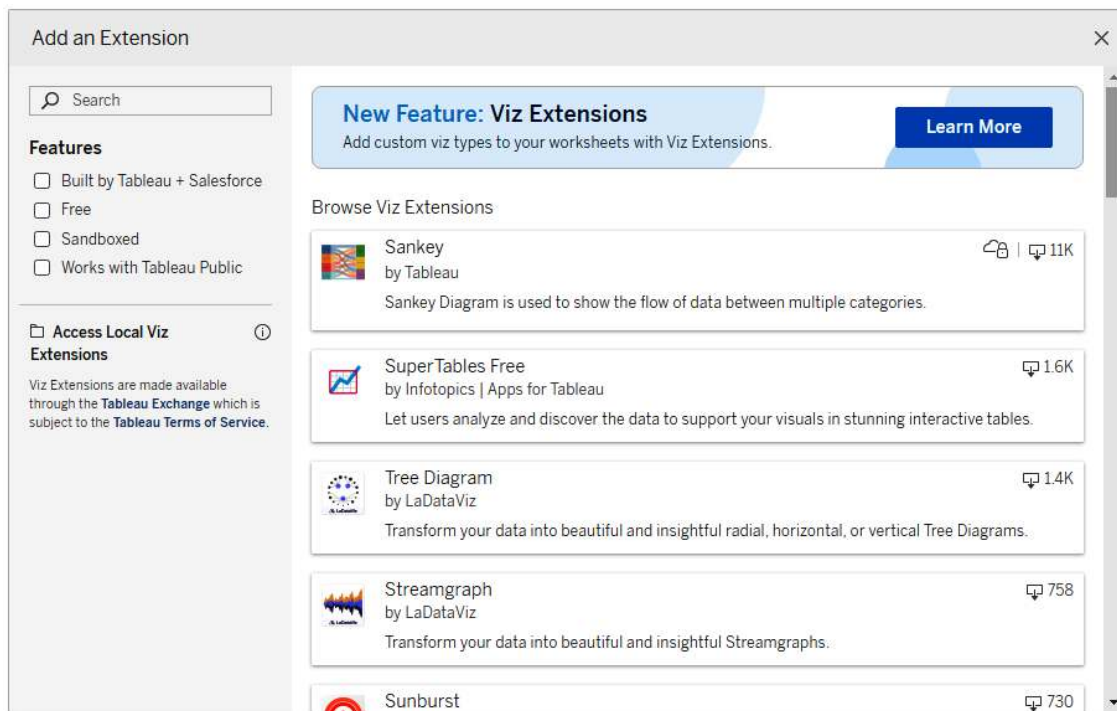


Figure 3.57: The Add an Extension dialog box gives you many options for finding an appropriate extension to visualize your data

You can search through an ever-growing library of extensions provided by Tableau as well as third-party developers. Many are free, while some are paid. In this example, we'll select **Sankey** to visualize the flow of patients through their hospital experience. After selecting **Sankey**, we'll be given some additional information about technical specifications and a detailed description.

Visualizing flow with the Sankey Viz Extension

Once you've reviewed the technical details and opened the extension, you'll see the extensions interface on the Marks card:

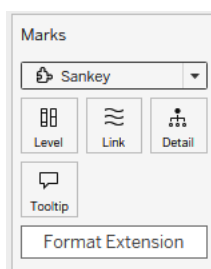


Figure 3.58: The Marks card for a Sankey includes drop locations for Level, Link, Detail, and Tooltip, along with a Format Extension button

The **Hospital Visits & Revenue** data source contains fields named **Event 1**, **Event 2**, **Event 3**, and **Event 4**. These fields represent a flow of events for each patient such as being admitted to the ER, having x-rays or labwork, and finally being discharged. With the Sankey Viz Extension selected, we'll visualize the flow of patients with these steps:

1. Drag **Event 1** to **Level** on the **Marks** card. This defines the first level or step in the Sankey visualization.

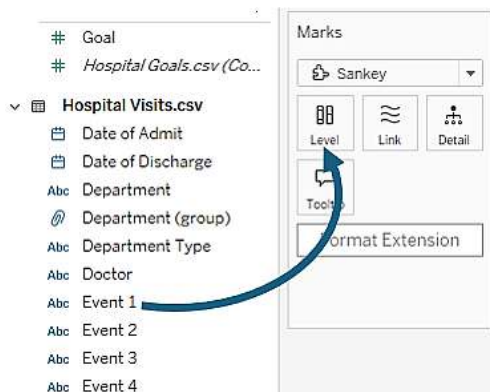


Figure 3.59: Dragging a field to Level creates a new step in the Sankey visualization

2. Successively drag and drop **Event 2**, **Event 3**, and **Event 4** onto **Level**. Notice that each successive dimension creates a new step in the diagram.

Your Sankey should now look something like this:

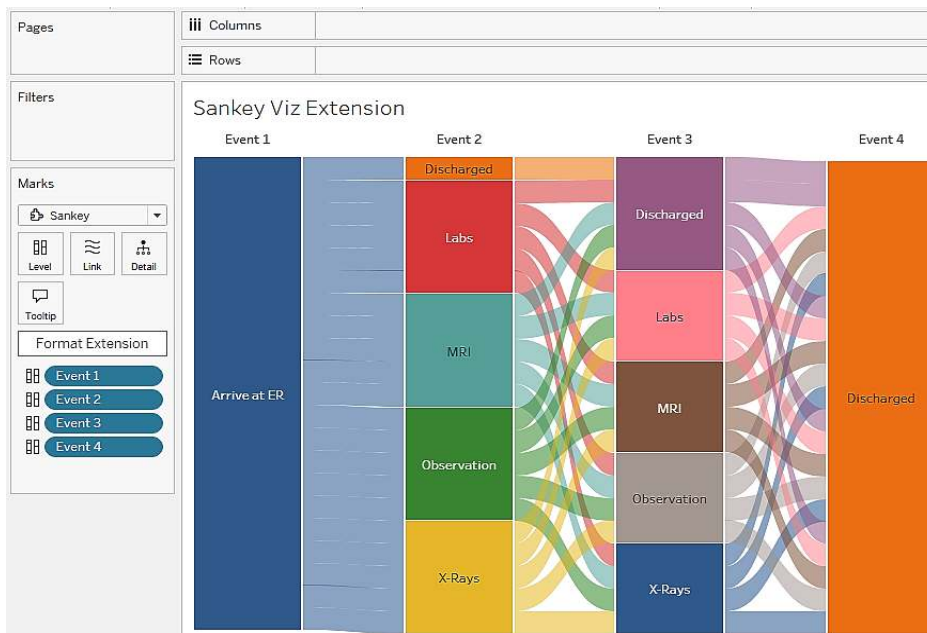


Figure 3.60: An interim step in creating the Sankey diagram in which all events are shown as levels or steps in the flow

Notice that all patients start by arriving at the ER and then progress through a flow of events that may be different for each patient. This diagram is not quite complete. We can see each successive step in the flow, but the links that show us the amount of flow (in this case, the number of patients) between levels have not been defined by the data. Let's complete the diagram.

3. Drag and drop **Number of Patient Visits** onto **Link** on the **Marks** card. This will size each link according to the number of patients moving from one level to the next.
4. Click on **Format Extension** and experiment with the various formatting options.
5. Ultimately, you will have a view something like this:

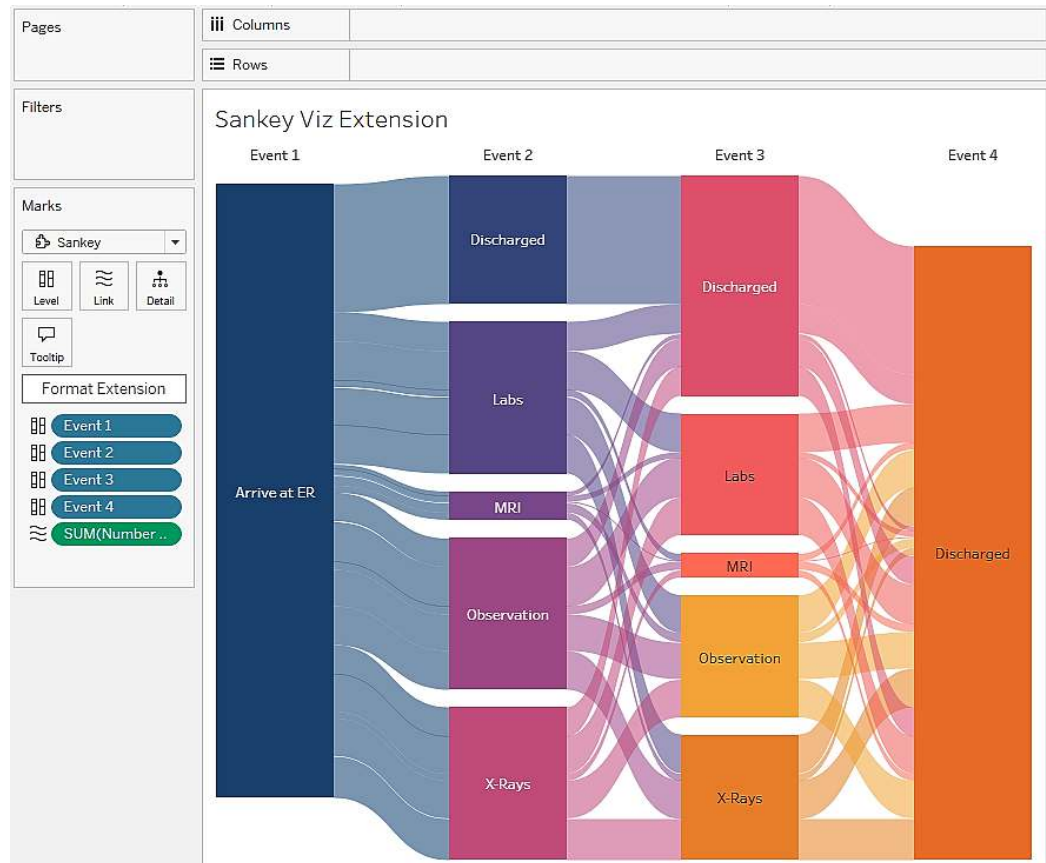


Figure 3.61: A complete Sankey showing the flow of patients from Arrival to Discharge

This type of visualization uncovers some interesting insights. For example, very few patients end up getting an MRI right after arriving at the ER. Some patients who are initially placed in observation end up getting labs, MRIs, or x-rays, while some who start with those diagnostics end up in observation.

As you use Viz Extensions, you'll find that they integrate into Tableau in much the same way other visualizations do. The Sankey we created has interactivity such as hover-over tooltips and highlighting upon selecting a specific link or level. You could use it in a dashboard, and it would be interactive with other visualizations and filters.

Ultimately, Viz Extensions open a world of possibilities for new and exciting ways to explore your data!

Summary

We've covered quite a bit of ground in this chapter! You should now have a good grasp of when to use certain types of visualizations. The types of questions you ask about data will often lead you to a certain type of view. You've explored how to create these various types and how to extend basic visualizations using a variety of advanced techniques, such as calculated fields, jittering, multiple mark types, dual axes, and Viz Extensions. Along the way, we've also covered some details on how dates work in Tableau.

Hopefully, the examples of using calculations in this chapter have whetted your appetite for learning more about creating calculated fields. The ability to create calculations in Tableau opens endless possibilities for extending the analysis of data, calculating results, customizing visualizations, and creating rich user interactivity. We'll dive deep into calculations in the next two chapters to see how they work and what amazing things they can do.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



4

Starting an Adventure with Calculations

We've already seen the amazing discovery, analysis, and data storytelling possible in Tableau by simply connecting to data and dragging and dropping fields. Now, we'll set off on an adventure with calculations.

Calculations significantly extend the possibilities for analysis, design, and interactivity in Tableau. We'll see how calculations can be used to address common issues with data, extend data by adding new dimensions and measures, and provide additional flexibility in interactivity.

As you work through this chapter, try to understand the key concepts behind how calculations work in Tableau. As usual, follow along with the examples, but feel free to explore and experiment. The goal is not to merely have a list of calculations you can copy but to gain knowledge of how calculations can be used to solve problems and add creative functionality to your visualizations and dashboards.

The first half of this chapter focuses on providing an overview of the various kinds of calculations and how they are created, while the second half provides quite a few practical examples. The topics we will study here are as follows:

- Introduction to calculations
- Row-level calculations
- Aggregate calculations
- Parameters
- Practical examples of calculations and parameters
- Performance considerations

We'll start with an introduction to the types of calculations in Tableau and then delve into some examples.

Introduction to calculations

In Tableau, a calculation is often referred to as a **Calculated Field** because, in most cases, when you create a calculation, it will show up as a new field in the data pane. Calculations consist of code that's made up of functions, operations, and references to other fields, parameters, constants, groups, or sets. This code returns a value. Sometimes, this result is per row of data, and sometimes it is done at an aggregate level. We'll consider the difference between Tableau's major calculation types next.

The four main types of calculations

Tableau has four major types of calculations:

- **Row-level calculations:** These calculations are performed for every row of underlying data.
- **Aggregate calculations:** These calculations are performed at an aggregate level, which is usually defined by the dimensions used in the view.
- **Level of detail calculations:** These special calculations are aggregations that are performed at a specified level of detail, with the results available at the row level.
- **Table calculations:** These calculations are performed on the table of aggregate data that has been returned by the data source to Tableau.

Understanding and recognizing the four main types of calculations will enable you to leverage the power and potential of calculations in Tableau.

In this chapter, we'll take a close look at two of the four main types of calculations in Tableau: row-level and aggregate calculations. We'll consider the final two types in *Chapter 5, Leveraging Level of Detail Calculations*, and *Chapter 6, Diving Deep with Table Calculations*.



As you think through using a calculation to solve a problem, always consider the type of calculation you might need. Look for tips throughout this chapter and the next two that will help you consider *why* a certain type of calculation was used.

Now we have examined the major calculation types in Tableau, we will see how they are created and edited.

Creating and editing calculations

There are multiple ways to create a calculated field in Tableau:

1. Select **Analysis | Create Calculated Field...** from the menu.

2. Use the drop-down menu next to **Dimensions** in the **Data** pane and select **Create Calculated Field...**:

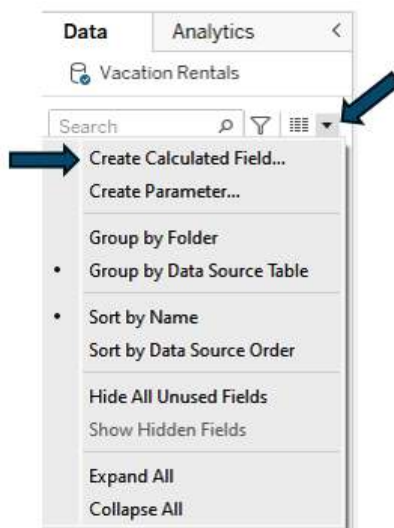


Figure 4.1: The Create Calculated Field... option

3. Right-click an empty area in the **Data** pane and select **Create Calculated Field...**.
4. Use the drop-down menu on a field, set, or parameter in the **Data** pane and select **Create | Calculated Field....** The calculation will begin as a reference to the field you selected.
5. Double-click an empty area on the **Rows**, **Columns**, or **Measure Values** shelves, or in the empty area on the **Marks** card, to create an ad hoc calculation (though this will not show the full calculation editor).
6. When you create a calculated field, it becomes part of the currently selected data source. To edit an existing calculated field, use the drop-down menu in the **Data** pane and select **Edit...**.

The interface for creating and editing calculations looks like this:

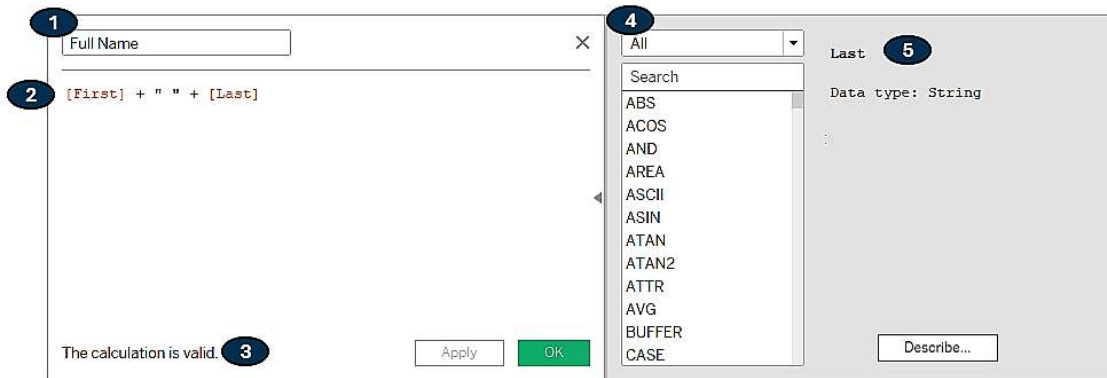


Figure 4.2: The creating and editing calculations interface

This window has several key features:

1. **Calculated field name:** Enter the name of the calculated field here. Once created, the calculated field will show up as a field in the **Data** pane with the name you entered in this text box.
2. **Code editor:** Enter code in this text area to perform the calculation. The editor includes auto-complete for recognized fields and functions. Additionally, you may drag and drop fields and text snippets to and from the code editor and the **Data** pane and view.
3. An **indicator** at the bottom of the editor will alert you to errors in your code. Additionally, if the calculation is used in views or other calculated fields, you will see a drop-down indicator that will let you see the dependencies. Click the **Apply** button to apply changes to the calculation throughout the workbook while leaving the calculation editor open. The **OK** button will save the code changes and close the editor. If you wish to discard any changes you've made, click the **X** button in the upper-right corner to cancel the changes.
4. The **functions list** contains all the functions that you can use in your code. Many of these functions will be used in examples or discussed in this chapter. Tableau groups various functions according to their overall use:
 - **Number:** Mathematical functions, such as rounding, absolute value, trig functions, square roots, and exponents.
 - **String:** Functions that are useful for string manipulation, such as getting a substring, finding a match within a string, replacing parts of a string, and converting a string value into uppercase or lowercase.
 - **Date:** Functions that are useful for working with dates, such as finding the difference between two dates, adding an interval to a date, getting the current date, and transforming strings with non-standard formats into dates.
 - **Type Conversion:** Functions that are useful for converting one type of field into another, such as converting strings into integers, integers into floating-point decimals, or strings into dates. We'll cover the major Tableau data types in the next section.
 - **Logical:** Decision-making functions, such as **if then else** logic or case statements.
 - **Aggregate:** Functions that are used for aggregating such as summing, getting the minimum or maximum values, or calculating standard deviations or variances.
 - **Pass Through** (only available when connected live to certain databases, such as SQL Server): These functions allow you to pass through raw SQL code to the underlying database and retrieve a returned value at either a row level or an aggregate level.
 - **User:** Functions that are used to obtain usernames and check whether the current user is a member of a group. These functions are often used in combination with logical functions to customize the user's experience or to implement user-based security when publishing to Tableau Server or Tableau Online.
 - **Table calculation:** These functions are different from the others. They operate on the aggregated data *after* it is returned from the underlying data source and just prior to the rendering of the view.
 - **Spatial:** These functions allow you to perform calculations with spatial data.



Selecting a function in the list or clicking a field, parameter, or function in the code will reveal details about the selection on the right. This is helpful when nesting other calculated fields in your code, when you want to see the code for that particular calculated field, or when you want to understand the syntax for a function.

With a good understanding of the interface, let's briefly look at some foundational concepts for calculations.

Data types

Fundamental to the concept of calculations are **data types**, which describe the kind of information stored by a field, parameter, or returned by a function. Tableau distinguishes seven types of data:

- **Number (decimal):** These are numeric values that include places after the decimal. Values such as 0.02, 100.377, or 3.14159 are decimal values.
- **Number (whole):** These are integer or whole numbers with no fractional values or places after the decimal. Values such as 5, 157, and 1,455,982 are whole numbers.
- **Date and Time:** These are dates along with times. Values such as November 8, 1980 12:04:33 PM are date and time types.
- **Date:** These are dates without times. Values such as July 17, 1979 are date types.
- **String:** These are a series of characters. A string may consist of a mixture of alphabetic characters, numeric characters, symbols, or special characters. They may even be blank (empty). Values such as Hello World, password123, and %\$@*! are all strings. In code, strings will be surrounded by single or double quotes.
- **Boolean:** This is a true or false value. The values TRUE and FALSE, and the expressions 1=1 (which evaluates as true) and 1=2 (which evaluates as false), are all Boolean types.
- **Spatial:** A complex value that describes a location, line, or shape as it relates to a spatial area.

Every field in Tableau has one of these data types and every function in Tableau returns one of these data types. Some functions expect input that matches some of these types, and you'll receive errors if you try to pass in the wrong type.

Some types can be converted into other types. For example, using some of the type conversion functions mentioned above, you could convert the string "2.81" into the decimal value 2.81. You could convert that decimal value into a whole number, but in that case, you'd lose the places after the decimal value and the whole number would simply be 2.

A data type is different from the format displayed. For example, you may choose to format a decimal as a percentage (for example, 0.2 could be shown as 20%), as currency (for example, 144.56 could be formatted as \$144.56), or even as a number with 0 decimals (for example, 2.81 would be rounded to 3).

Pay attention to the data types of fields and functions as we continue.

Additional functions and operators

Tableau supports numerous functions and operators. In addition to the functions that are listed on the calculation screen, Tableau supports the following operators, keywords, and syntax conventions:

Operator/Keyword	Description
AND	Logical and between two Boolean values or statements
OR	Logical or between two Boolean values or statements
NOT	Logical not to negate a Boolean value or statement
IN	A logical expression that tests for the equality of a value against a list of values and returns true if the value matches one of the items in the list
= or ==	Logical equals to test the equality of two statements or values (single or double equal signs are equivalent in Tableau's syntax)
+	Addition of numeric or date values or the concatenation of strings
-	Subtraction of numeric or date values
*	Multiplication of numeric values
/	Division of numeric values
^	Raise to a power with numeric values
()	Parentheses to define the order of operations or enclose function arguments
[]	Square brackets to enclose field names
{ }	Curly braces to enclose the level of detail calculations
//	Double slash to start a comment

Table 4.1: A table showing Tableau functions and operators



Field names that are a single word may optionally be enclosed in brackets when used in calculations. Field names with spaces, special characters, or from secondary data sources must be enclosed in brackets.

You'll see these operators and functions throughout the next few chapters, so familiarize yourself with their uses.



Additionally, there are some functions that are “undocumented,” but may still be used. For a list of such functions, see <https://www.flerlagetwins.com/2024/09/hidden-functions.html>

Now, let's consider the data that will guide us through some practical examples.

Understanding the data used in chapter examples

Before we get started with some examples, let's consider a sample dataset that will be used for the examples in this chapter. It's simple and small, which means we will be able to easily see how the calculations are being done.

This dataset is included as `Vacation Rentals.csv` in the `\Learning Tableau\Chapter 04` directory of this book's resources, and it is also included in the `Chapter 4 STARTER` workbook as a data source named `Vacation Rentals`:

Rental Property	First	Last	Start	End	Discount	Rent	Tax per Night
112-Asbury Atoll	Mary	Slessor	Dec 2	Dec 9	150	1,500	15
112-Asbury Atoll	Amy	Carmichael	Dec 9	Dec 15	0	1,500	15
155-Beach Breeze	Charles	Ryrie	Dec 2	Dec 9	130	1,300	10
155-Beach Breeze	Dwight	Pentecost	Dec 16	Dec 23	280	1,400	10
207-Beach Breeze	Lewis	Chafer	Dec 9	Dec 23	280	2,800	10
207-Beach Breeze	John	Walvoord	Dec 2	Dec 9	60	1,500	10

Table 4.2: A table showing the sample data

The dataset describes several vacation rental properties, the renters, the start and end dates of the rental period, the discount, rent, and tax per night. We'll use it throughout the rest of the chapter as we see some examples of calculations. Let's start with row-level calculations.

Row-level calculations

We'll walk through several examples of row-level calculations in this section. You'll find the completed calculations in the `Complete` workbook, but you might prefer to start from scratch in the `Starter` workbook. We won't necessarily cover creating a visualization for every example but try building some of your own as you work through the examples.

Concatenating strings

We'll start with a very simple example and then build up in complexity. In the `Chapter 04` workbook, create a new calculated field called `Full Name` with the following code:

```
[First] + " " + [Last]
```

This code concatenates the strings of **First** and **Last** with a space in between them. Your calculation editor should look something like the following:

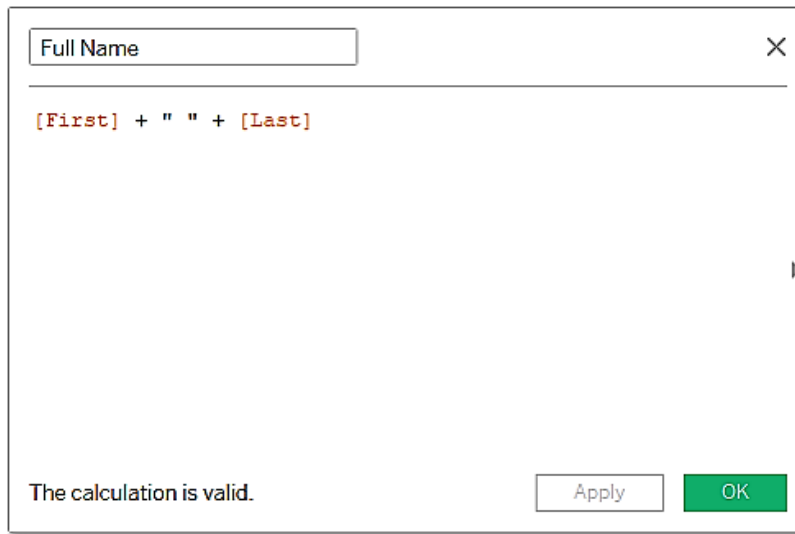


Figure 4.3: Creating the Full Name calculation in the editor

After clicking **OK**, you should notice a new **Full Name** field in the **Data** pane. The value for that field is calculated per row of data. That is, every row of data contains the full name of the renter.

String manipulation and conditional logic

Note that the **Rental Property** field contains values such as 112-Asbury Atoll or 207-Beach Breeze. Let's assume that the naming convention of the rental unit in the vacation rental data gives us the room number and the name of the building separated by a dash. For example, the unit named 207-Beach Breeze is room 207 of the Beach Breeze building.

Name the first Room, and add the following code:

```
SPLIT([Rental Property], "-", 1)
```

Then, create another calculated field named **Building** with the following code:

```
SPLIT([Rental Property], "-", 2)
```

Both of these functions use the `Split()` function, which splits a string into multiple values and keeps one of those values. This function takes three arguments: the **string**, the **delimiter** (a character or set of characters that separates values), and the **token number** (which value to keep from the split, that is, first, second, third, and so on). Using the - (dash) as the delimiter, **Room** is the first value and **Building** is the second.

Using the two calculated fields, create a bar chart of **Rent per Building & Room**, like this:

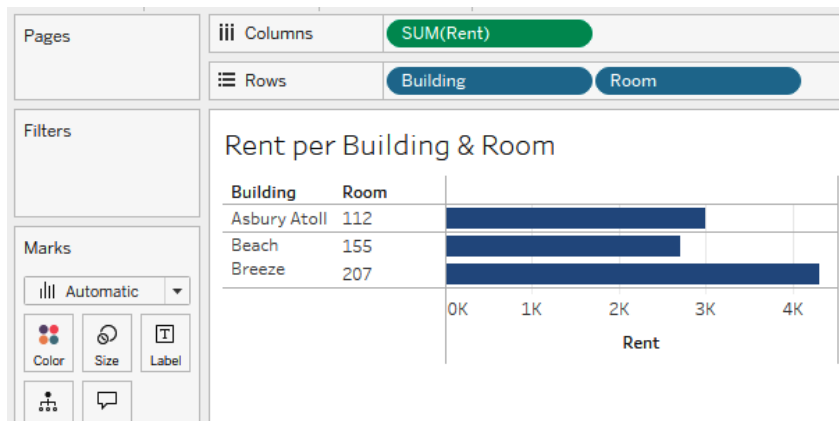


Figure 4.4: Using your calculated fields to build a view

The **Building** and **Room** fields show up in the **Data** pane as dimensions. The calculated dimensions can be used just like any other dimension. They can slice the data, define the level of detail, and group measures.

Row-level calculations are calculated at the row level, but you can choose to aggregate the results. For example, you could aggregate to find the highest **Room** number (MAX) or count the distinct number of **Buildings** (COUNTD). In fact, if the result of a row-level calculation is numeric, Tableau will often place the resulting field as a measure by default. But as we've seen, you can use a row-level field as either a dimension or a measure in the view.

Note that Tableau adds a small equals sign to the icon of the fields in the **Data** pane to indicate that they are calculated fields:

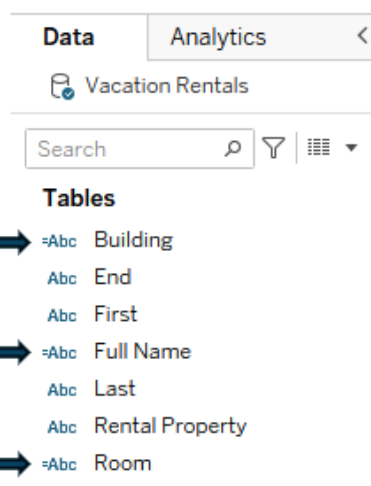


Figure 4.5: The small = sign indicates a field is a calculation

The code for both calculated fields is executed for every row of data and returns a row-level value. We can verify that the code is operating on the row level by examining the source data. Simply click on the **View Data** icon in the **Data** pane (it's next to the search box and filter icon in the preceding screenshot). Here, the new fields of **Building**, **Full Name**, and **Room**, along with the row-level values, can be clearly seen:

Building	End	First	Full Name	Last	Rental Property	Room	Start
Asbury Atoll	9-Dec	Mary	Mary Slessor	Slessor	112-Asbury Atoll	112	2-Dec
Asbury Atoll	15-Dec	Amy	Amy Carmichael	Carmichael	112-Asbury Atoll	112	9-Dec
Beach Breeze	9-Dec	Charles	Charles Ryrie	Ryrie	155-Beach Breeze	155	2-Dec
Beach Breeze	23-Dec	Dwight	Dwight Pentecost	Pentecost	155-Beach Breeze	155	16-Dec
Beach Breeze	23-Dec	Lewis	Lewis Chafer	Chafer	207-Beach Breeze	207	9-Dec
Beach Breeze	9-Dec	John	John Walvoord	Walvoord	207-Beach Breeze	207	2-Dec

Figure 4.6: Viewing the underlying data shows us the calculation is done per row of data



Tableau provides a shortcut for splitting a field. You can use the drop-down menu on a field in the **Data** pane and select **Transform | Split** or **Transform | Custom Split** (if you have a non-standard delimiter). The results are calculated fields similar to those you created previously but with some additional logic around determining data types. Transform functionality, such as split, is also available for fields in the **Preview** or **Metadata** views on the **Data** source screen.

We'll extend the example a bit more and assume you know that the floor of a room is indicated by its number. Rooms 100 through 199 are on the first floor, and 200 through 299 are on the second. You'd like to have that information available for analysis.

We could potentially add this attribute to the source data, but there are times when this may not be an option or may not be feasible. You may not have permission to change the source data or the source might be a spreadsheet that is automatically generated every day, and any changes would be overwritten.

Instead, you can create a row-level calculation in Tableau to extend the data. To do so, create a calculated field named **Floor** with the following code:

```
IF LEFT([Room], 1) = "1"
THEN "First Floor"
ELSEIF LEFT([Room], 1) = "2"
THEN "Second Floor"
END
```

This code uses the `LEFT()` function to return the leftmost character of the room. Thus, 112 gives a result of 1; 207 gives a result of 2.

The `IF THEN END` logic allows us to assign a result (either `First Floor` or `Second Floor`), depending on which case is true. Notice that you used the `Room` field in the calculation, which, in turn, was another calculation.



Using a reference to a calculated field within another calculation is referred to as **nesting**. The calculations that use other calculated fields are called **nested calculations**. There's no theoretical limit to how many levels of nesting you can use, but it may become hard to untangle the logic if you use too many levels of nesting.

Planning for data variations

As you write calculations, consider whether your calculation covers variations in the data that are not currently present.



A few good questions to ask yourself whenever you write a calculation in Tableau are:
What happens if the data changes? Does the calculation handle unexpected or invalid values? Have I covered every case?

For example, the preceding floor calculation only works if all the rooms are either 100- or 200-level rooms. What if there is a room, 306, on the third floor, or a room, 822, on the eighth floor?

To account for additional cases, we might simplify our calculation to the following:

```
LEFT([Room], 1)
```

This code simply returns the leftmost character of the room number. We'll get 3 for 306 and 8 for 822. But what if we have room numbers such as 1056 on the tenth floor and 1617 on the sixteenth? We'd have to consider other options, such as the following:

```
MID([Room], 0, LEN([Room]) - 2)
```

Although this is more complicated, the string functions return a substring that starts at the beginning of the string and ends just before the last two characters. That gives us floor 5 for 506, and floor 18 for 1856.

We've now considered some row-level calculation examples. Let's move on to the next major type of calculation in Tableau: aggregate-level calculations.

Aggregate calculations

We've already considered aggregations such as `sum`, `min`, and `max` in Tableau. Often, you'll use fields as simple aggregations in the view. But sometimes, you'll want to use aggregations in more complex calculations.

For example, you might be curious to explore the percentage of the rent that was discounted. There is no such field in the data. It could not really be stored in the source, because the value changes based on the level of detail present in the view (for example, the discount percentage for an individual unit will be different from the discount percentage per floor or per building). Rather, it must be calculated as an aggregate and recalculated as the level of detail changes.

Let's create a calculation named `Discount %` with the following code:

```
SUM([Discount]) / SUM([Rent])
```

This code indicates that the sum of `Discount` should be divided by the sum of `Rent`. This means that all the values of `Discount` will be added, and all the values of `Rent` will be added. Only after the sums are calculated will the division occur.



Once you've created the calculation, you'll notice that Tableau treats the new field as a **Measure** in the **Data** pane. Tableau will treat any calculation with a numeric result as a measure by default, but you can change *row-level calculations* to dimensions if desired. In this case, though, you are not even able to redefine the new field as a dimension. The reason for this is that Tableau will treat every *aggregate calculation* as a measure, no matter what data type is returned. This is because an aggregate calculation depends on dimensions to define the level of detail at which the calculation is performed. So, an aggregate calculation cannot be a dimension itself.

As the value of your calculation is a percentage, you will also likely want to define the format as a percentage. To do this, right-click the `Discount %` field, select **Default Properties | Number Format**, and select **Percentage**. You may adjust the number of decimal places that are displayed if desired.

Now, create a couple of views to see how the calculation returns different results, depending on the level of detail in the view. First, we'll build a view to look at each individual rental period:

1. Place **Building**, **Room**, **Full Name**, **Start**, and **End** on **Rows**.
2. In the **Data** pane, under **Measures**, double-click each of the following fields: **Rent**, **Discount**, and **Discount %**. Tableau will place each of these measures in the view by using **Measure Names** and **Measure Values**.
3. Rearrange the fields on the **Measure Values** shelf so that the order is **Rent**, **Discount**, and **Discount %**:

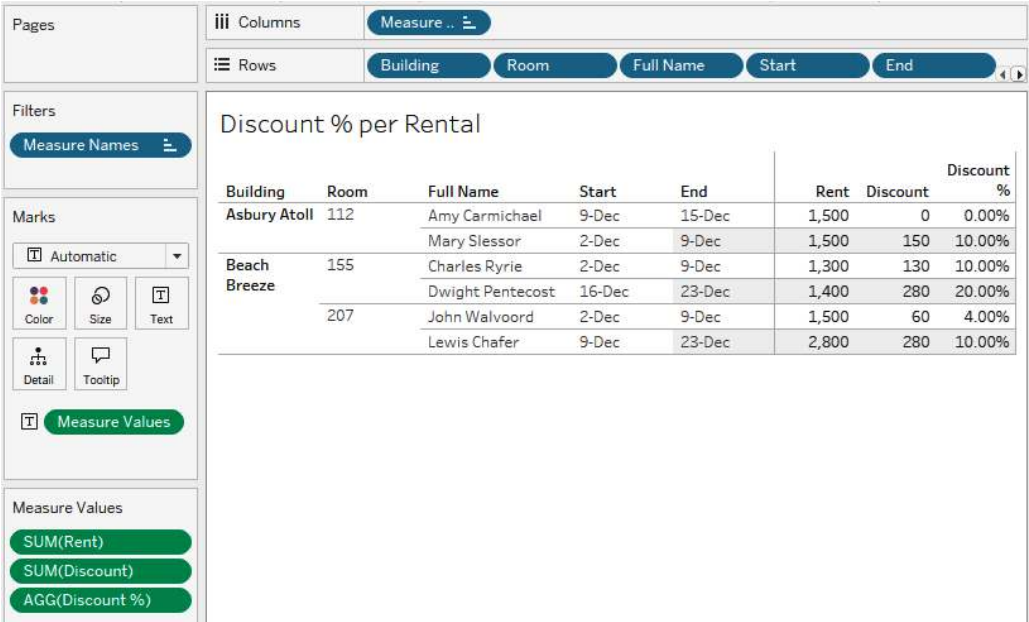


Figure 4.7: Illustrates the Discount % calculated at the level of Building, Room, Full Name, Start, and End

You can see the percentage given by way of a discount for each rental period. However, notice how the values change when you remove all fields except **Building** and **Room**:

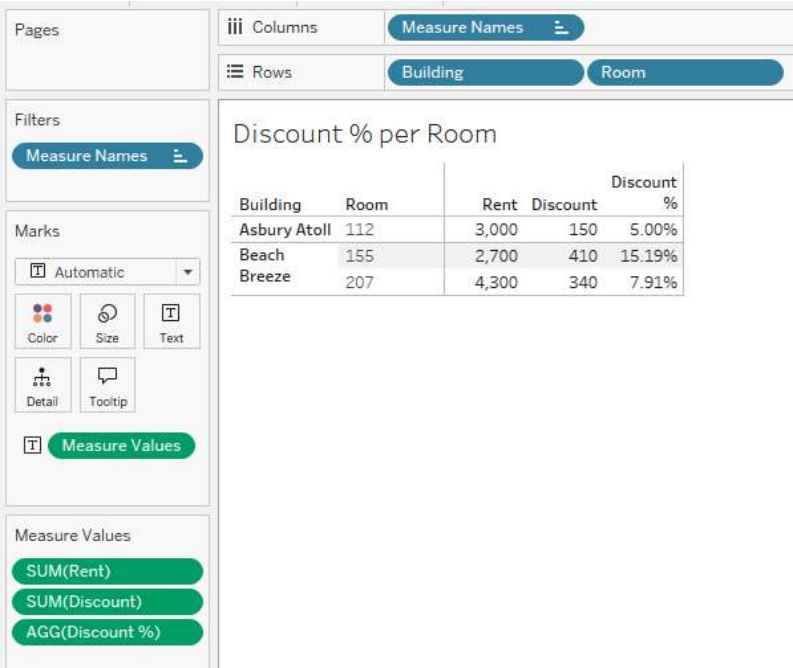


Figure 4.8: Illustrates the Discount % calculated at the level of Building and Room

Why did the values change? Because aggregations depend on what dimensions define the level of detail of the view. In the first case, **Building** and **Room**, **Full Name**, **Start**, and **End** defined the level of detail in the view. So, the calculation added up all the rent for each rental period and all the discounts for the rental period and then divided them. In the second case, **Building** and **Room** redefined the level of detail. So, the calculation added up all the prices for each building and room and all the discounts for each building and room and then divided them.



You may have noticed that as you double-clicked each measure, it was added to the pane of the view in a special way. **Measure Names** and **Measure Values** are special fields that appear in every data connection (toward the bottom of the **Data** pane). These serve as placeholders for multiple measures that share the same space in the view.

In the view you just created, for example, three measures all shared space in the pane. **Measure Values** on **Text** indicated that all values of measures on the **Measure Values** shelf should be displayed as text. The **Measure Names** field on **Columns** created a column for each measure, with the value of the name of that measure.

Notice that the values change again, as expected, if you look at the overall dataset without slicing by any dimensions:

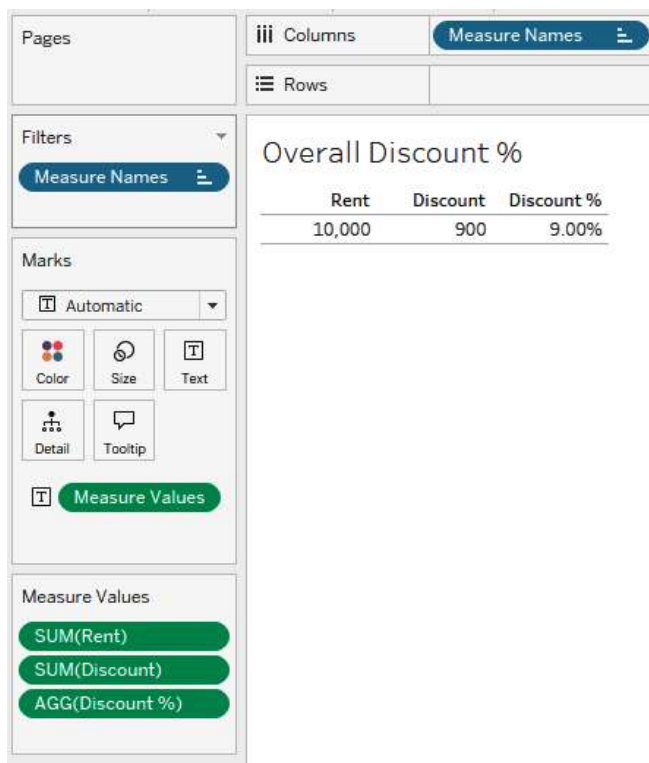


Figure 4.9: Illustrates the Discount % calculated at the highest possible level: the entire dataset

An easy way to get Tableau to implement **Measure Names/Measure Values** is to remember that they are used whenever you want to use *two or more measures in the same space* in a view. So, if you want to use two or more measures in the pane, drag the first to the pane and then the second. If you want to use two or more measures on the same axis, drag the first to the axis, and then drag and drop the second on the same spot.

Now that you have a basic understanding of row-level and aggregate calculations, let's consider why the distinction is important.

Why the row-level versus aggregate difference matters

Let's say you created a Discount % (row level) calculation with the following code:

```
[Discount]/[Rent]
```

The code differs from the aggregate calculation you created previously, which had the following code:

```
SUM([Discount])/SUM([Rent])
```

Here is the dramatic difference in results:

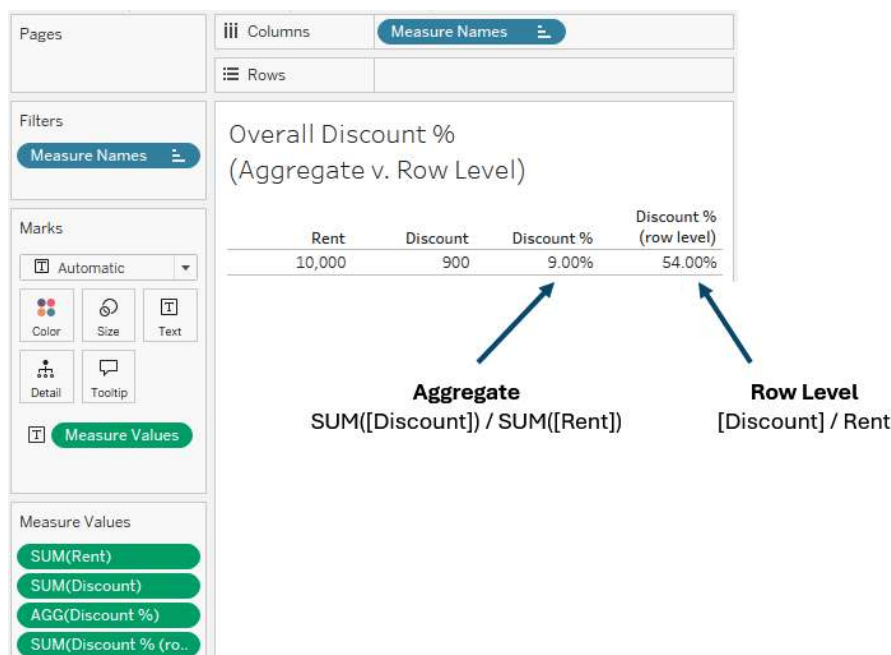


Figure 4.10: Illustrates the Discount % calculated as a row-level value and as an aggregate

Why is there such a difference in the results? It's a result of the way the calculations were performed.

Notice that Discount % (row level) appears on the **Measure Values** shelf as a SUM. That's because the calculation is a row-level calculation, so it gets calculated row by row and then aggregated as a measure after all the row-level values have been determined. The 54.00% value you see is actually a sum of percentages that were calculated in each record of underlying data.

In fact, the row-level calculation and the final aggregation are performed like this:

Rental Property	First	Discount	Rent	Discount %
112-Asbury Atoll	Mary	150	\$1,500	10%
112-Asbury Atoll	Amy	0	\$1,500	0%
155-Beach Breeze	Charles	130	\$1,300	10%
155-Beach Breeze	Dwight	280	\$1,400	20%
207-Beach Breeze	Lewis	280	\$2,800	10%
207-Beach Breeze	John	60	\$1,500	4%

Row Level
[Discount] / [Rent]
Results calculated for each row

Final Aggregation
Sum of row level results
54%

Figure 4.11: If each Discount % result is calculated at a row level and then aggregated, the result is wrong

Contrast that with the way the aggregate calculation is performed. Notice that the aggregation that's listed in the active field on the **Measure Values** shelf in the view is AGG, not SUM. This indicates that you have defined the aggregation in the calculation. Tableau does not aggregate the results further. Here is how the aggregate calculation is performed:

Discount	Rent	Discount %
150	\$1,500	
0	\$1,500	
130	\$1,300	
280	\$1,400	
280	\$2,800	
60	\$1,500	

Row Level
Not Calculated

Division of Sums:
 $SUM([Discount]) / SUM([Rent])$
9%

$SUM([Discount])$ **\$900**

$SUM([Rent])$ **\$10,000**

Figure 4.12: If the numerator and denominator are aggregated first, then the Discount % calculation is correct

It is vital to understand the difference between row-level and aggregate calculations to ensure you are getting the results you expect and need. In general, use row-level calculations when you are certain that you will use either the value as a dimension or that an aggregation of the row-level values will make sense. Use aggregate calculations if aggregations must be performed prior to other operations.

One of the most common error messages that's encountered while writing Tableau calculations is *Cannot mix aggregate and non-aggregate arguments with this function*. When you encounter this message, check your code to make sure you are not improperly mixing row-level fields and calculations with aggregate fields and calculations. For example, you cannot have something like `[Discount] / SUM([Rent])`.

This mixture of a row-level value (Discount) and the aggregation (SUM of Rent) is invalid. Because row-level and aggregate are two completely different levels, they cannot be mixed.

With the distinction between row-level and aggregate calculations clear, let's take an interlude and discuss parameters before building additional examples.

Parameters

Before moving on to some additional examples of row-level and aggregate calculations, let's take a little side trip to examine parameters, given that they can be used in incredible ways in calculations.

A **parameter** in Tableau is a placeholder for a single, global value such as a number, date, or string. Parameters may be shown as controls (such as sliders, drop-down lists, or type-in text boxes) to end users of dashboards or views, giving them the ability to change the current value of the parameter. Parameter values may even be changed with actions, as you'll see in *Chapter 8, Telling a Data Story with Dashboards*.

The value of a parameter is global, so if the value is changed, every view and calculation in the workbook that references the parameter will use the new value. Parameters provide another way to provide rich interactivity to the end users of your dashboards and visualizations.

Parameters can be used to allow anyone interacting with your view or dashboard to dynamically do many things, including the following:

- Alter the results of a calculation
- Change the size of the bins
- Change the number of top or bottom items in a top n filter or top n set
- Set the value of a reference line or band
- Pass values to a custom SQL statement that's used in a data source

Some of these are options we'll consider in later chapters.

Since parameters can be used in calculations, and since calculated fields can be used to define any aspect of a visualization (from filters to colors to rows and columns), the change in a parameter value can have dramatic results. We'll see some examples of this in the following sections.

Creating parameters

Creating a parameter is similar to creating a calculated field. There are multiple ways to create a parameter in Tableau:

- Use the drop-down menu next to **Dimensions** in the **Data** pane and select **Create Parameter**.
- Right-click an empty area in the **Data** pane and select **Create Parameter**.
- Use the drop-down menu on a field, set, or parameter already in the **Data** pane and select **Create | Parameter...**

In the last case, Tableau will create a parameter with a list of potential values based on the **domain** (distinct values) of the field. For fields in the **Data** pane that are **discrete** (blue) by default, Tableau will create a parameter with a list of values matching the discrete values of the field. For fields in the **Data** pane that are **continuous** (green), Tableau will create a parameter with a range set to the minimum and maximum values of the field that's present in the data.

When you first create a parameter (or subsequently edit an existing parameter), Tableau will present an interface like this:

The screenshot shows the 'Create Parameter' dialog box with the following numbered callouts:

- 1** Name: The title of the parameter.
- 2** Data type: A dropdown menu currently set to 'Float'.
- 3** Display format: A dropdown menu currently set to '1'.
- 4** Current value: A dropdown menu currently set to 'First Option'.
- 5** Value when workbook opens: A dropdown menu currently set to 'Current value'.
- 6** Allowable values: Radio buttons for 'All', 'List' (selected), and 'Range'.
- 7** Value table: A table with two columns, 'Value' and 'Display As'.

Value	Display As
1	First Option
2	Second Option
3	Third Option
Click to add	
- 8** Fixed/When workbook opens: Radio buttons for 'Fixed' (selected) and 'When workbook opens'. Below them is an 'Add values from' button.
- 9** Cancel/OK: Buttons at the bottom right, with a 'Remove Selected' button above them.

Figure 4.13: The Create Parameter interface numbered with corresponding descriptions below

The interface contains the following features:

1. **Name** will show as the default title for the parameter controls and will also be the reference in calculations. You can also add a **Comment** to describe the use of the parameter.
2. **Data type** defines what type of data is allowed for the value of the parameter. Options include integer, float (floating-point decimal), string, Boolean, date, or date with time.
3. **Display format** defines how the values will be displayed. For example, you might want to show an integer value as a dollar amount, a decimal as a percentage, or display a date in a specific format.
4. **Current value** defines what the initial default value of the parameter will be. Changing this value on this screen or on a dashboard or visualization where the parameter control is shown will change the current value.

5. **Value when workbook opens** allows you to optionally change the default value of the parameter when the workbook opens based on a calculated value.
6. The **Allowable values** option gives us the ability to restrict the scope of values that are permissible. There are three options for **Allowable values**:
 - **All** allows any input from the user that matches the data type of the parameter.
 - **List** allows us to define a list of values from which the user must select a single option. The list can be entered manually, pasted from the clipboard, or loaded from a dimension of the same data type.
 - **Range** allows us to define a range of possible values, including an optional upper and lower limit, as well as a step size. This can also be set from a field or another parameter.
7. In the example of the preceding screenshot, since we've selected **List** for **Allowable values**, we are given options to enter the list of possible values. In this example, a list of three items has been entered. Notice that the value must match the data type, but the display value can be any string value. You can drag and drop values in the list to reorder the list. If **Range** had been selected, the screen would instead show options for setting the **Minimum**, **Maximum**, and **Step Size** for the range.
8. Also specific to **List** are a couple of additional options for populating the list:
 - **Fixed**: You may manually enter the values, paste from the clipboard, or set them from the existing values of a field in the data. In any case, the list will be a static list and will not change even if the data is updated.
 - **When workbook opens** allows you to specify a field that will dynamically update the list based on the available values for that field when the workbook is first opened.
9. Click **OK** to save changes to the parameter or **Cancel** to revert.

When the parameter is created, it appears in the **Data** pane in the **Parameters** section. The drop-down menu for a parameter reveals an option, **Show Parameter Control**, which adds the parameter control to the view. The little drop-down caret at the upper right of the parameter control reveals a menu for customizing the appearance and behavior of the parameter control.

Here is the parameter control, shown as a single value list, for the parameter we created earlier:

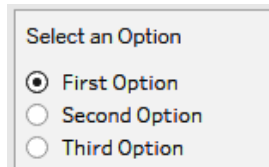


Figure 4.14: The parameter control shown as a single select radio button list

This control can be shown on any sheet or dashboard and allows the end user to select a single value. When the value is changed, any calculations, filters, sets, or bins that use the parameter will be re-evaluated, and any views that are affected will be redrawn.

Next, we'll consider some practical examples that use parameters in calculations.

Practical examples of calculations and parameters

Let's turn our attention to some practical examples of row-level and aggregate calculations. The goal is to learn about and understand some of what is possible with calculations. You will be able to build on these examples as you embark on your analysis and visualization journey.



A great place to find help and suggestions for calculations is the official Tableau forums at <https://community.tableau.com/s/explore-forums>.

Fixing data issues

Often, data is not entirely clean. That is, it has problems that need to be corrected before meaningful analysis can be performed. For example, dates may be incorrectly formatted, or fields may contain a mix of numeric values and character codes that need to be separated into multiple fields. Calculated fields can often be used to resolve these issues.



We'll consider using Tableau Prep—a tool designed to shape and cleanse data—in *Chapter 15, Taming Data with Tableau Prep*. Tableau Prep's calculation syntax is nearly identical, so many of the examples in this chapter will also be applicable in that context. Knowing how to address data issues in either Tableau Desktop or Tableau Prep will help you greatly.

We'll continue working with the Vacation Rentals data. You'll recall that the start and end dates looked something like this:

Start	End
Dec 2	Dec 9
Dec 9	Dec 15
Dec 16	Dec 23

Table 4.3: A table showing example dates

Without the year, Tableau does not recognize the Start or End fields as *dates*. Instead, Tableau recognizes them as *strings*. You might try using the drop-down menu on the fields in the **Data** pane to change the data type to **Date**. But without the year, Tableau will almost certainly parse them incorrectly, or at least incompletely. This is a case where we'll need to use a calculation to fix the issue.

Assuming, in this case, that you are confident the year should always be 2025, you might create calculated fields named `Start Date` and `End Date`.

Here is the code for getting the start date:

```
DATE([Start] + ", 2025")
```

And here is the code for getting the end date:

```
DATE([End] + ", 2025")
```

What these calculated fields do is concatenate the month and day with the year and then use the DATE() function to convert the string into a date value. Indeed, Tableau recognizes the resulting fields as dates (with all the features of a date field, such as built-in hierarchies). A quick check in Tableau reveals the expected results:

Columns

Rows

Rental Prop...RoomStartEndStart DateEnd Date

Rental Property	Room	Start	End	Start Date	End Date	
112-Asbury Atoll	112	2-Dec	9-Dec	12/2/2025	12/9/2025	Abc
		9-Dec	15-Dec	12/9/2025	12/15/2025	Abc
155-Beach Breeze	155	2-Dec	9-Dec	12/2/2025	12/9/2025	Abc
		16-Dec	23-Dec	12/16/2025	12/23/2025	Abc
207-Beach Breeze	207	2-Dec	9-Dec	12/2/2025	12/9/2025	Abc
		9-Dec	23-Dec	12/9/2025	12/23/2025	Abc

Figure 4.15: The corrected dates appear next to the string versions. All fields are discrete dimensions on Rows (the dates are exact dates)

Not only are we able to fix problems in the data, but we can also extend the data and our analysis using calculations. We'll consider this next.

Extending the data

Often, there will be dimensions or measures you'd like to have in your data, but that are not present in the source. Many times, you will be able to extend your dataset using calculated fields. We already considered an example of creating a field for the full name of the guest where we only had first and last name fields.

Another piece of data that might unlock some truly interesting analysis would be the length of each rental. We have the start and end dates, but not the length of time between those two dates. Fortunately, this is easy to calculate.

Create a calculated field named `Nights Rented` with the following code:

```
DATEDIFF('day', [Start Date], [End Date])
```

Tableau employs intelligent code completion. It will offer suggestions for functions and field names as you type in the code editor. Pressing the *Tab* key will autocomplete what you have started to type based on the current suggestion.

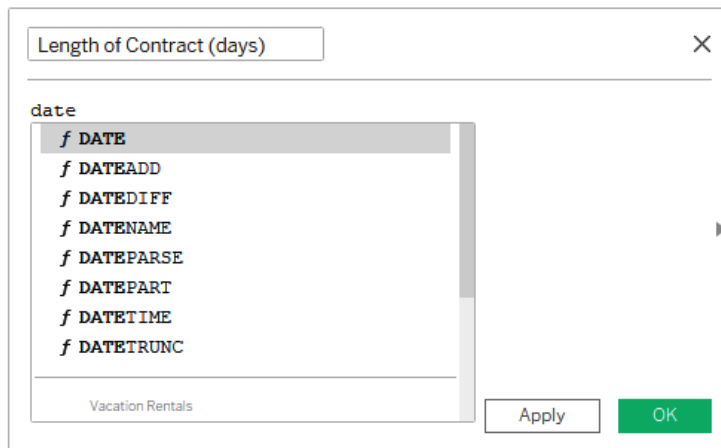


Figure 4.16: The intelligent code completion will suggest possible field names and functions

as you type

The DATEDIFF() function takes a date part description and a start and an end date and returns a numeric value for the difference between the two dates. We now have a new measure that wasn't available previously.

We can use this new measure in our visualizations, such as the Gantt chart of rentals, as follows:

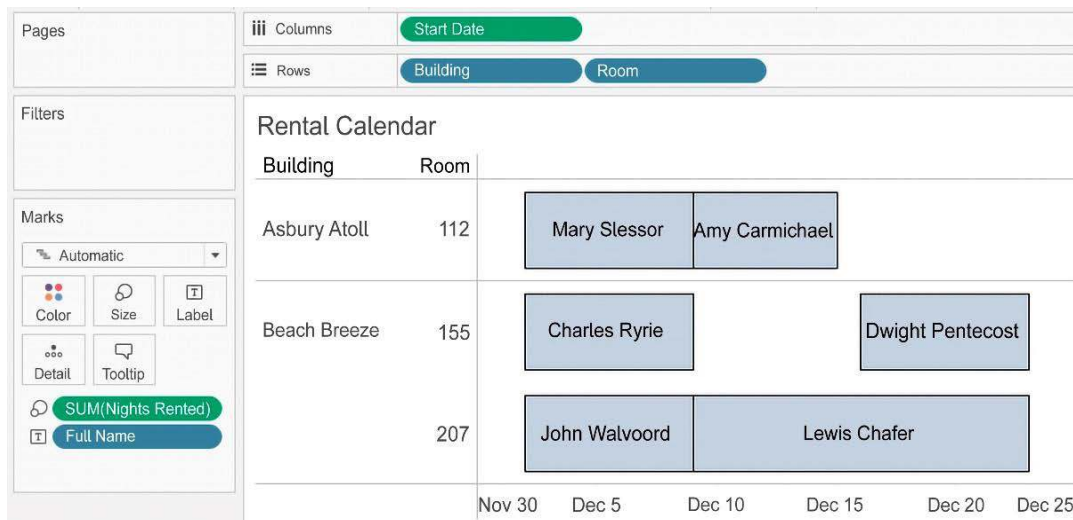


Figure 4.17: The calculated field allows us to create the Gantt chart



Remember that you can right-click and drag a field from the **Data** pane to a shelf to get options about how to use the field. In this case, right-clicking and dragging **Start Date** to **Columns** allows you to select to use it as a continuous field.

You'll find many ways to extend your data with calculations as you continue your journey in Tableau. And that will enable you to accomplish some amazing analyses and visualizations. We'll consider some examples next.

Enhancing user experience, analysis, and visualizations

Calculations and parameters can greatly enhance user experience, analysis, and visualizations.

Let's say we want to give the vacation condo manager the ability to do some what-if analysis. Every year, she offers a free night during the month of December. She wants to be able to see which renters would have received the free night, depending on which night she chooses.

To accomplish this, follow these steps:

1. If you have not done so, create a Gantt chart similar to what was shown earlier (following the field placement of the screenshot).
2. Create a parameter called **Free Night** with a data type of **Date** and a starting value of 12/12/2025. This will allow the manager to set and adjust the starting date for the promotional month. Show the parameter control by selecting **Show Parameter Control** from the drop-down menu on the **Free Night** parameter in the **Data** pane.
3. Now, add a reference line to the view to show the free night. Do this by switching to the **Analytics** tab in the left-hand sidebar. Drag **Reference Line** to the view and drop it on **Table**:

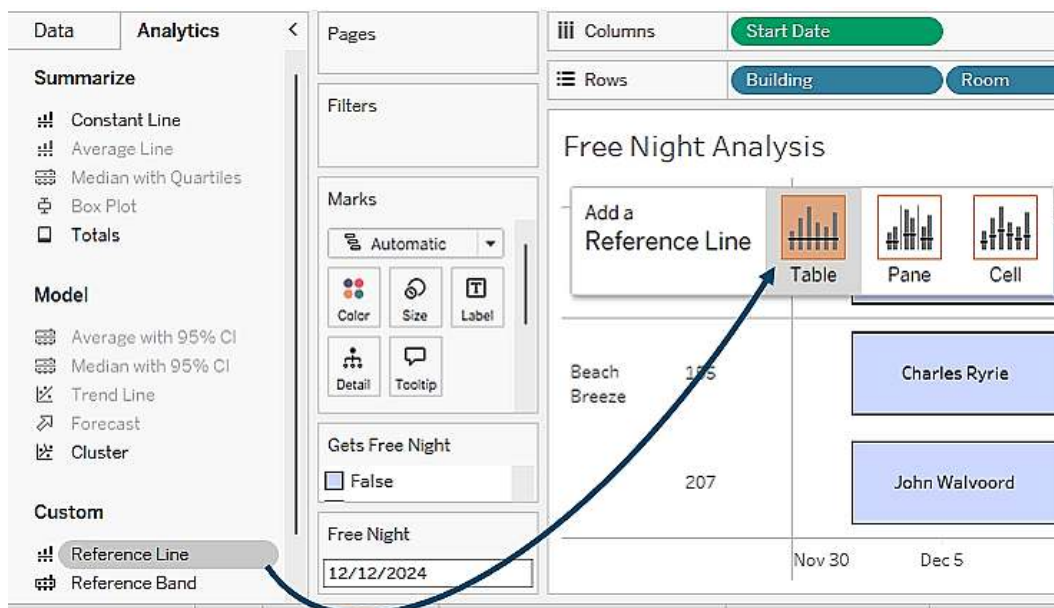


Figure 4.18: Add a reference line by switching to the Analytics pane and dragging the reference line onto the canvas

- In the resulting dialog box, set **Line Value** to **Free Night**. You may wish to set **Label** to **None**, or **Custom** with the text **Free Night**. You may also wish to adjust the formatting of the line:

Edit Reference Line, Band, or Box

Line Band Distribution Box Plot

Scope

☒ Entire Table ☐ Per Pane ☐ Per Cell

Line

Value: Free Night

Label: None

Tooltip: Automatic

Line only 95

Formatting

Line:

Fill Above: None

Fill Below: None

☒ Show recalculated line for highlighted or selected data points

OK

Figure 4.19: Use the Edit Reference Line dialog to adjust the formatting, labels, and tooltips

- Create a calculated field called **Gets Free Night** that returns a true or false value, depending on whether the free night falls within the rental period:

```
[Free Night] >= [Start Date]
AND
[Free Night] <= [End Date]
```

- Place this new calculated field on the **Color** shelf.

We now have a view that allows the apartment manager to change the date and see a dynamically changing view that makes it obvious which renters would have fallen within a given promotional period. Experiment by changing the value of the **Free Night** parameter to see how the view updates:

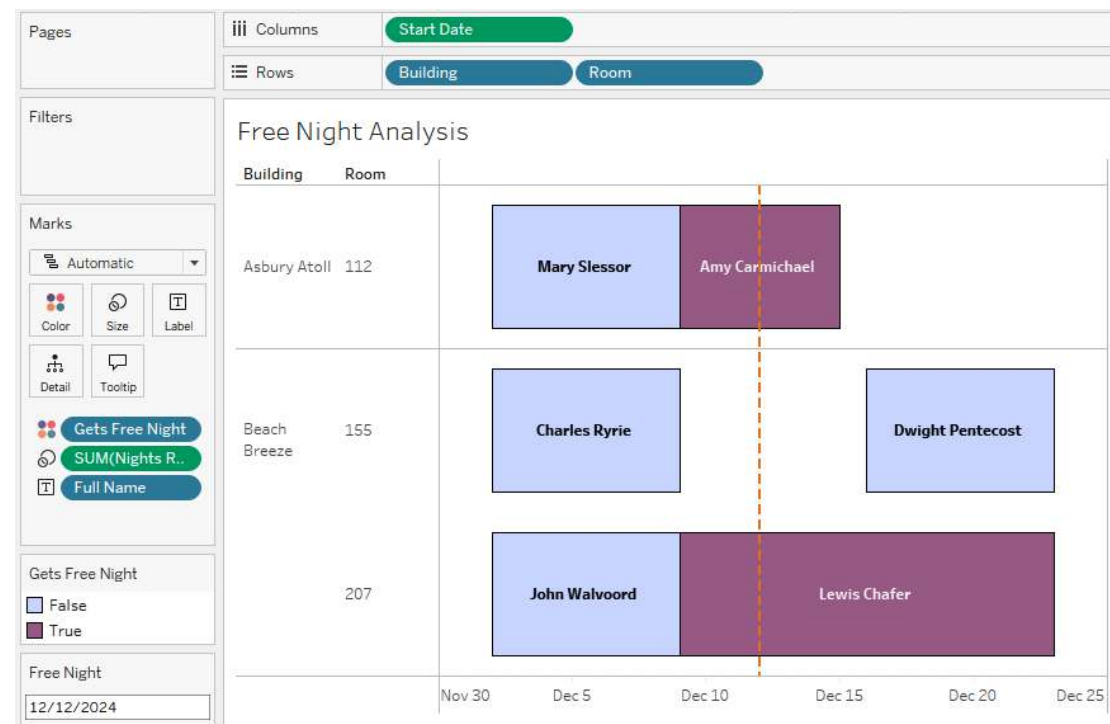


Figure 4.20: The reference line will move, and the affected individuals will be recalculated every time you change the Free Night parameter value

The preceding view shows the proposed free night as a dashed line and highlights which rental periods would receive a free night. The line position and Gantt mark colors will change as the apartment manager adjusts the **Free Night** parameter value.

In addition to extending your analysis, visualization, and user experience, you might also use calculations to add the required business logic. We'll consider that next.

Meeting business requirements

Sometimes, data doesn't exactly match what your organization wants. For example, the measures in the data may not be the exact metrics required to make key business decisions or dimension values may need to be grouped according to a certain set of rules. Although this kind of business logic is often applied as data is transformed or modeled prior to connecting to it with Tableau, you may find cases where you need to implement business logic on the fly.

In this example, consider that the measure *Rent* is simply the base rent and does not include the discount or taxes. Those are separate fields in the data. If you needed to analyze the total revenue, you'd need to calculate it. That calculation might look something like this:

```
[Rent] - [Discount] + ([Tax per Night] * [Nights Rented])
```


This formula takes the base Rent, subtracts the Discount, and then adds the Tax per Night multiplied by the Nights Rented. The parentheses aid readability but are not required because the multiplication operator, *, has higher precedence and is evaluated before the addition, +.

Up until now, we've created calculated fields that extend the data source. Sometimes, you just need a quick calculation to help in a single view. We'll conclude by looking at these quick ad hoc calculations.

Ad hoc calculations

Ad hoc calculations allow you to add calculated fields to shelves in a single view without adding fields to the **Data** pane.

Let's say you have a simple view that shows **Revenue per Guest**, like this:

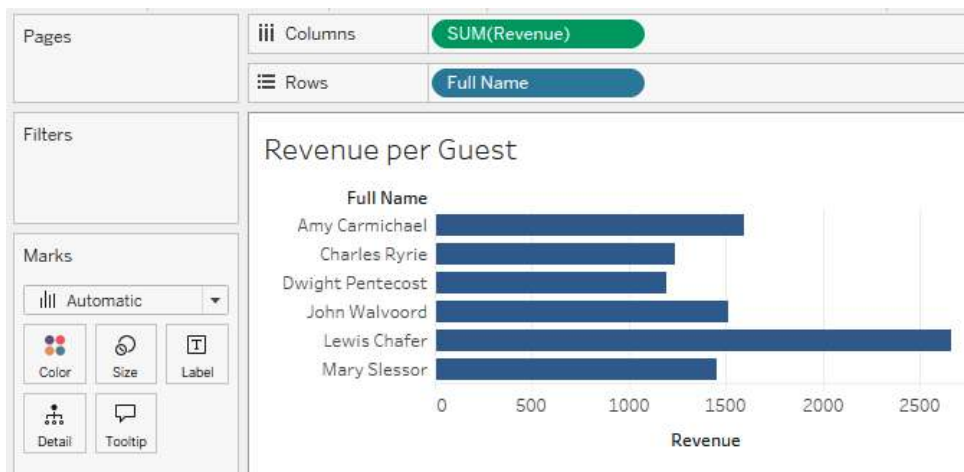


Figure 4.21: The revenue generated from each individual's stay

What if you wanted to quickly highlight any renters who had paid less than \$1,500? One option would be to create an ad hoc calculation. To do so, simply double-click on an empty area of the **Columns**, **Rows**, or **Measure Values** cards, or on the empty space of the **Marks** shelf, and then start typing the code for a calculation. In this example, we've double-clicked the empty space on the **Marks** shelf:

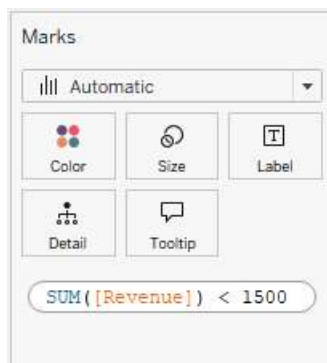


Figure 4.22: Creating an ad hoc calculation on the Marks card

Here, we've entered code that will return `True` if the sum of `Rent` is less than \$1,500 and `False` otherwise. Pressing *Enter* or clicking outside the text box will reveal a new ad hoc field that can be dragged and dropped anywhere within the view. Here, we've moved it to the **Color** shelf:

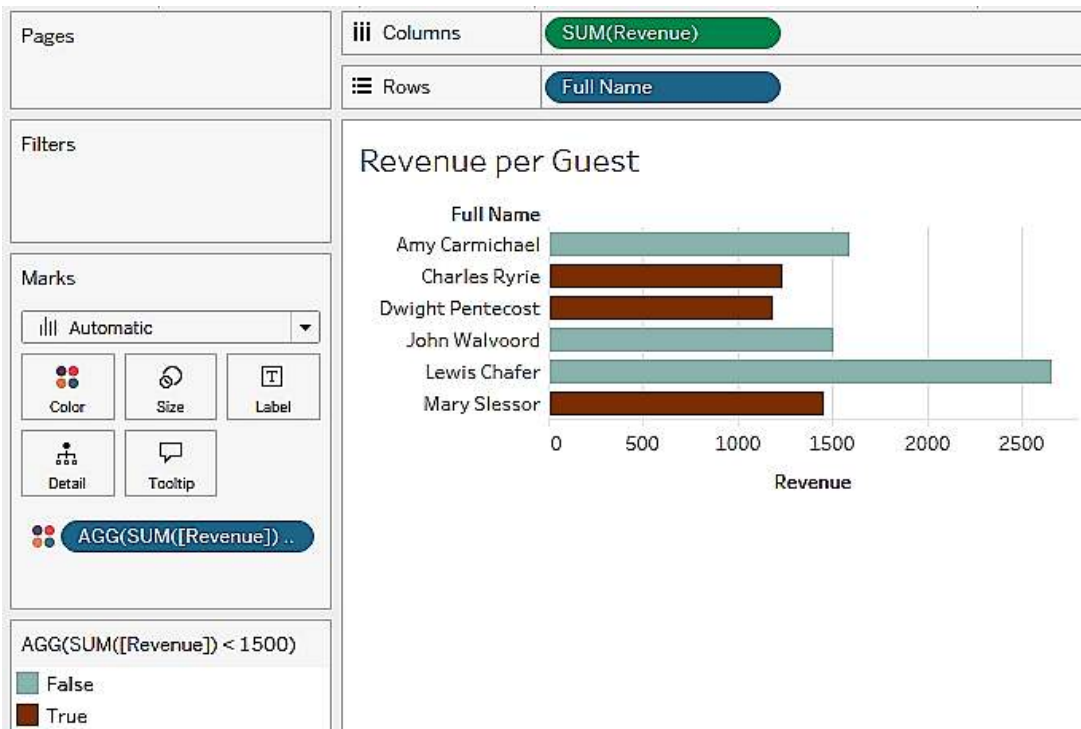


Figure 4.23: Using the ad hoc calculation on Color

The ad hoc field is only available within the view and does not show up in the **Data** pane. You can double-click the field to edit the code.



Dragging and dropping an ad hoc field into the **Data** pane transforms it into a regular calculated field that will be available for other views that are using that data source.

Having seen a number of practical examples, let's conclude by considering some ways to ensure good performance when using calculations.

Performance considerations

With small datasets and efficient databases, you may not notice inefficient calculations. With larger datasets, the efficiency of your calculations can start to make a difference to the speed at which a view is rendered.

Here are some tips for making your calculations as efficient as possible:

- Boolean and numeric calculations are much faster than string calculations. If possible, avoid string manipulation and use aliasing or formatting to provide user-friendly labels. For example, don't write the following code: `IF [value] == 1 THEN "Yes" ELSE "No" END`. Instead, simply write `[value] == 1`, and then edit the aliases of the field and set `True` to `Yes` and `False` to `No`.
- Look for ways to increase the efficiency of a calculation. If you find yourself writing a long `IF ELSEIF` statement with lots of conditions, see whether there are one or two conditions that you can check first to eliminate the checks of all the other conditions. For example, let's consider simplifying the following code:

```
//This is potentially less efficient...
IF [Type] = "Dog" AND [Age] < 1 THEN "Puppy"
ELSEIF [Type] = "Cat" AND [Age] < 1 THEN "Kitten"
END
//...than this code:
IF [Age] < 1 THEN
  IF [Type] = "Dog" THEN "Puppy"
  ELSEIF [Type] = "Cat" THEN "Kitten"
END
END
```

Notice how the check of type doesn't have to be done for any records where the age was equal to or greater than 1. That could be a very high percentage of records in the dataset.

- Use `IN()` instead of a series of `OR` statements. Not only is it more performant, it's more readable. Consider the differences between the following segments of code:

```
//This is potentially less efficient...
IF [Type] = "Dog" OR [Type] = "Cat" OR [Type] = "Pangolin"
THEN "Mammal"
ELSEIF [Type] = "Lizard" OR [Type] = "Snake" OR [Type] = "Turtle"
THEN "Reptile"
END

//...than this code (which is also more readable)
IF [Type] IN ("Dog", "Cat", "Pangolin")
THEN "Mammal"
ELSEIF [Type] IN ("Lizard ", " Snake ", " Turtle ")
THEN "Reptile"
END
```

- Row-level calculations have to be performed for every row of data. Try to minimize the complexity of row-level calculations. However, if that is not possible or doesn't solve a performance issue, consider the final option.

- When you create a data extract, certain row-level calculations are materialized. This means that the calculation is performed once when the extract is created, and the results are then stored in the extract. The end result is that the data engine does not have to execute the calculation over and over. Instead, the value is simply read from the extract. Calculations that use any user functions, parameters, or `TODAY()` or `NOW()` will not be materialized in an extract as the value necessarily changes according to the current user, parameter selection, and system time. Tableau's optimizer may also determine whether to materialize certain calculations that are more efficiently performed in memory rather than having to read the stored value.
- Use the performance recorder to analyze performance issues. To start the recorder, use the menu to select **Help | Settings and Performance | Start Performance Recording**. After interacting with the visualization or dashboard, use the same menu options but with **Stop Performance Recording**. A new instance of Tableau will open with data and a dashboard designed to help you analyze performance issues, including those with calculations.
- Similarly, you can view the Tableau logs to identify performance issues and potential resolutions. A log viewing tool is available at <https://github.com/tableau/tableau-log-viewer>.



When you use an extract to materialize row-level calculations, only the calculations that were created at the time of the extract are materialized. If you edit calculated fields or create new ones after creating the extract, you will need to optimize the extract (use the drop-down menu on the data source or select it from the **Data** menu, and then select **Extract | Optimize** or **Extract | Compute Calculations Now**).

As you continue to work with calculations, pay special attention to situations where you notice performance issues, and consider whether you can optimize your calculations for better results.

Summary

Calculations open amazing possibilities in Tableau. You are no longer confined to the fields in the source data. With calculations, you can extend the data by adding new dimensions and measures, fix bad or poorly formatted data, and enhance the user experience with parameters for user input and calculations that enhance visualizations.

The key to using calculated fields is understanding the four main types of calculations in Tableau. Row-level calculations are performed for every row of source data. These calculated fields can be used as dimensions or they can be aggregated as measures. Aggregate calculations are performed at the level of detail that's defined by the dimensions that are present in a view. They are especially helpful, and even necessary, when you first aggregate components of the calculation before performing additional operations.

In the next chapter, we'll explore the third of the four main types of calculations: **level of detail calculations**. This will greatly extend your ability to work with data and solve all kinds of interesting problems.

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search for this book by name.

Note: Keep your purchase invoice ready before you start.



5

Leveraging Level of Detail Calculations

After exploring row-level and aggregate calculations, it's time to focus on the third of the four main types of calculations: **level of detail calculations**.

Level of detail calculations (often referred to as **LOD calcs** or **LOD expressions**) enable you to perform aggregations at a specific level of detail, which can differ from the level of detail defined in the view. This capability allows you to conduct various analyses that would otherwise be quite challenging.

In this chapter, we'll cover the following:

- Overview of level of detail
- Level of detail calculations
- Examples of fixed level of detail calculations
- Examples of include level of detail calculations
- Examples of exclude level of detail calculations

Overview of level of detail

What does the term **level of detail** mean? A lot depends on the context in which the term is used. Within Tableau, we'll distinguish several levels of detail, each of which is vitally important to understand in order to properly analyze data:

- **Data level of detail:** Sometimes referred to as the **grain** of the data, this is the level of detail defined by a single record of the dataset. If you can articulate what one record of the data represents (for example, "Every record represents a single order" or "There is one record for every customer"), then you have a good understanding of the data level of detail. *Row-level calculations operate at this level.*

- **View level of detail:** As mentioned in *Chapter 4, Starting an Adventure with Calculations*, the combination of fields used as dimensions in the view defines the view level of detail. Typically, Tableau draws a single mark for each distinct combination of values present in the data for all the dimensions in the view. For example, if **Customer** and **Year** are the two dimensions in your view, Tableau will draw a mark (such as a bar or circle) for each **Customer/Year** combination present in the data after filters have been applied. *Aggregate calculations operate at this level.*
- **Calculated level of detail:** This is a separate level of detail defined by a calculation. As we'll see, you may use any number of dimensions to define the level of detail. *Level of detail calculations are used to define this level.*

Consider the following dataset, with a **data level of detail** of one record per customer:

Customer	State	Membership Date	Membership Level	Orders
Neil	Kansas	2009-05-05	Silver	1
Jeane	Kansas	2012-03-17	Gold	5
George	Oklahoma	2016-02-01	Gold	10
Wilma	Texas	2018-09-17	Silver	4

Table 5.1: A sample dataset of customer records

In this case, each record defines a single unique customer. If we were to perform a row-level calculation, such as `DATEDIFF('year', [Membership Date], TODAY())` to determine the number of years each customer has been a member, then the result would be calculated per record.

Now consider a view created from the data with a **view level of detail** of **State**:

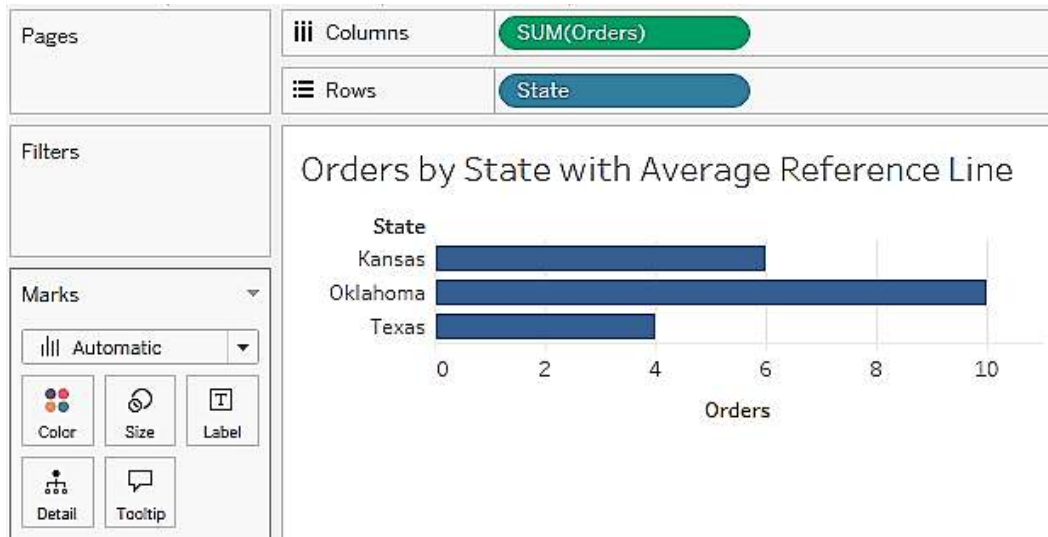


Figure 5.1: The view level of detail, which is State

As the only dimension in the view, **State** defines the view level of detail. There is one mark per state, and calculations and fields used as aggregates, such as **SUM(Orders)**, will be performed per state.

Based on that particular view, we might want to enhance our understanding by asking additional questions, such as the following:

- Which customer was the first member of each state in the view?
- How does the number of orders per state compare to the average number of orders for all states?
- Which membership level had the highest or lowest number of orders per state?

In each case, the question involves a level of detail that is different from the view (the minimum membership date per state compared to each individual customer, the average orders overall compared to orders per state, and the minimum or maximum number of orders per membership level per state). In some cases, it might make sense to build a new view to answer these questions. But sometimes we want to supplement an existing view or compare different levels of detail in the same view. Level of detail calculations provide a solution!

Level of detail calculations

Before getting into practical examples of using level of detail calculations, let's take a moment to understand the syntax and types of level of detail calculations.

Level of detail syntax

Level of detail calculations follow this basic pattern of syntax:

```
{FIXED|INCLUDE|EXCLUDE [Dim 1],[Dim 2] : AGG([Field])}
```

The definitions of the preceding declaration are as follows:

- **FIXED**, **INCLUDE**, and **EXCLUDE** are keywords that indicate the type of level of detail calculation. We'll consider the differences in detail in the following section.
- **Dim 1**, **Dim 2** (and as many dimensions that are needed) is a comma-separated list of dimension fields that defines the level of detail at which the calculation will be performed.
- **AGG** is the aggregate function you wish to perform (such as **SUM**, **AVG**, **MIN**, and **MAX**). If the field is already an aggregate calculation, you will not specify an additional aggregate function.
- **Field** is the value that will be aggregated as specified by the aggregation you choose.

Although we'll dive deeper into specifics in the next section, here is a real-world example of the syntax:

```
{FIXED [State] : MIN([Orders])}
```

Level of detail types

Three types of level of detail calculations are used in Tableau: **fixed**, **include**, and **exclude**.

Fixed

Fixed level of detail expressions work at the level of detail that's specified by the list of dimensions in the code, regardless of what dimensions are in the view. For example, the following code returns the average orders per state, regardless of what other dimensions are in the view:

```
{FIXED [State] : AVG([Orders])}
```

You may include as many dimensions as needed or none at all. The following code represents a fixed calculation of the average orders for the entire set of data from the data source:

```
{FIXED : AVG([Orders])}
```

Alternatively, you might write the calculation in the following way with identical results:

```
{AVG([Orders])}
```



A fixed level of detail expression with no dimensions specified is sometimes referred to as a *table-scoped fixed level of detail expression* because the aggregation defined in the calculation will be for the entire table.

Include

Include level of detail expressions aggregate at the level of detail that's determined by the dimensions in the view, *along with* the dimensions listed in the code. For example, the following code calculates the average orders at the level of detail that's defined by dimensions in the view, but includes the Membership Level dimension, even if Membership Level is not in the view:

```
{INCLUDE [Membership Level] : AVG([Orders])}
```

Exclude

Exclude level of detail expressions aggregate at the level of detail determined by the dimensions in the view, *excluding* any listed in the code. For example, the following code calculates the average number of orders at the level of detail defined in the view, but does not include the Customer dimension as part of the level of detail, even if Customer is in the view:

```
{EXCLUDE [Customer] : AVG([Orders])}
```

An illustration of the difference level of detail can make

As you analyze data, one thing you might often wonder is how slices of data relate to the overall picture. For example, you might wonder how the number of orders for each state in *Figure 5.1* relates to the overall average number of orders. One quick and easy option is to add an **Average Line** to the view from the **Analytics** tab by dragging and dropping like this:

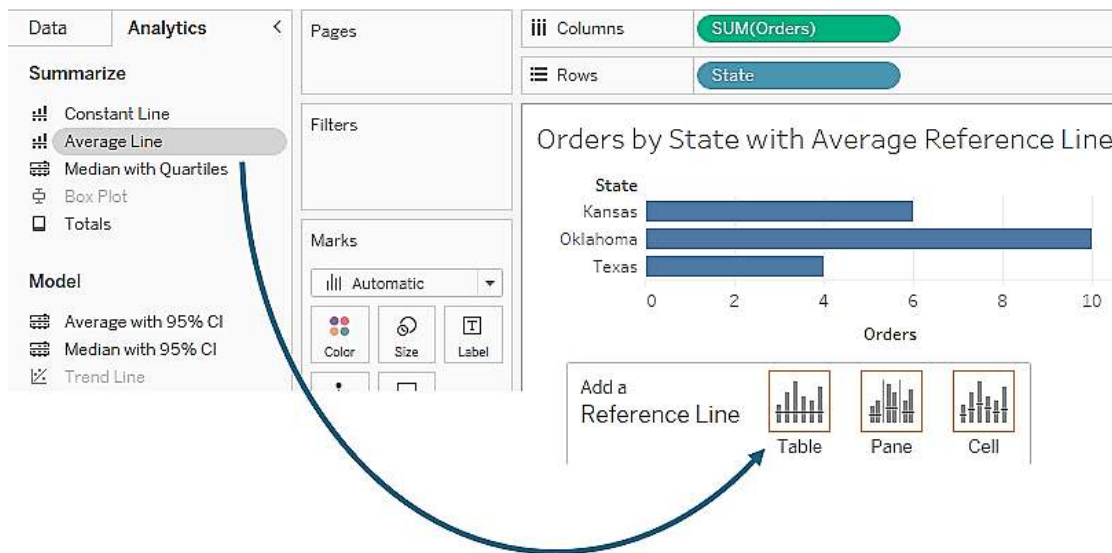


Figure 5.2: Adding an average line to the view

You'll end up with an average line that looks like this:

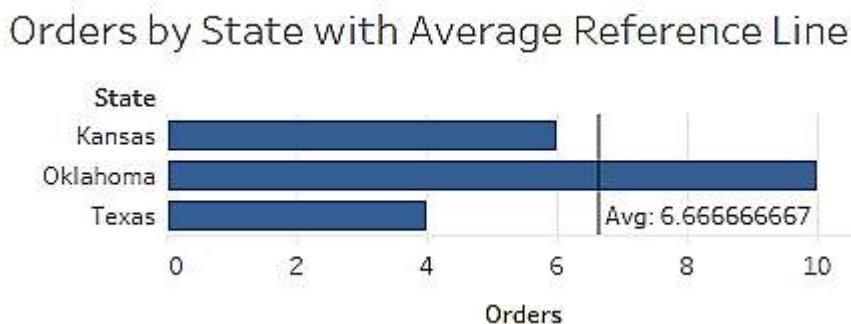


Figure 5.3: The overall average is reported as 6.66667. This is the average per state

But is 6.66667 truly the average number of orders overall? It turns out that it's not. It's actually the average of the sum of the number of orders for each state: $(6 + 10 + 4) / 3$. Many times, that average line (that is, the average of the total number of orders per state) is exactly what we want to compare when using aggregate numbers.

But sometimes, we might want to calculate the true overall average. To get the average number of orders present in the entire dataset, we might consider creating a calculation named Overall Average Number of Orders and using a fixed level of detail calculation like this:

```
{FIXED : AVG([Orders])}
```

Adding that calculated field to the **Detail** part of the **Marks** card and editing the reference line to use that field instead gives us a different result:

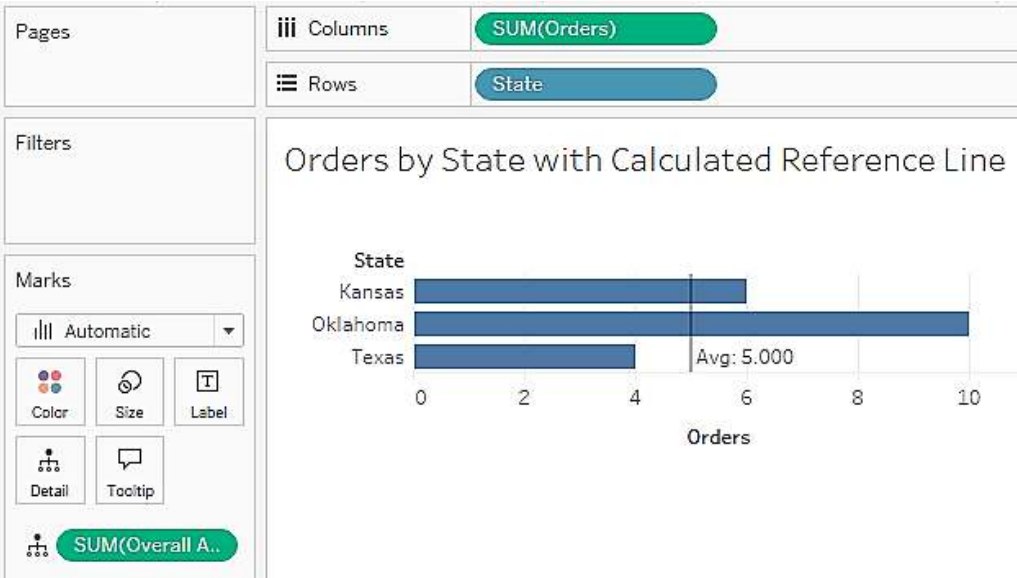


Figure 5.4: The true overall average number of orders per customer is 5

You’ll recall that the original dataset had four records, and a quick check validates the result:

$$(1 + 5 + 10 + 4) / 4 = 5$$

Now that we have examined how level of detail calculations make a real difference, let’s look at some practical examples.

Examples of fixed level of detail calculations

For practical examples of level of detail calculations, we’ll use the Chapter 05 Loans dataset contained in the Chapter 05 workbook. The actual dataset contains many more records and a few more columns, but here is a sample of the data it includes:

Date	Portfolio	Loan Type	Balance	Open Date	Member Name	Credit Score	Age	State
4/1/2024	Auto	Used Auto	15987	9/29/2022	Samuel Thomas	678	37	California
8/1/2024	Mortgage	1st Mortgage	96364	8/7/2017	Lloyd Kennedy	768	62	Wyoming
3/1/2024	Mortgage	HELOC	15123	4/2/2017	Inez Balling	651	66	Illinois
3/1/2024	Mortgage	1st Mortgage	418635	9/30/2019	Patrick Manley	766	60	Wyoming
5/1/2024	Credit Card	Gold Card	2292	10/15/2022	Edgar Wyllie	811	50	Ohio
4/1/2024	Auto	New Auto	21844	12/5/2021	Brian Small	808	48	Illinois
5/1/2024	Auto	New Auto Plus	1151	10/22/2022	Eric Cannon	660	44	Pennsylvania
3/1/2024	Credit Card	Standard Card	2090	8/13/2022	Anne Gaines	690	63	Nebraska
7/1/2024	Auto	New Auto Plus	336	7/27/2019	Ian Sharpless	798	50	Minnesota
4/1/2024	Auto	Used Auto	17023	2/29/2024	Diana Gibson	707	34	Massachusetts

Figure 5.5: Sample rows of the dataset used throughout this chapter

The dataset represents historical data for loans for members at a bank, credit union, or similar financial institution. Each record is a monthly snapshot of a loan and contains the date of the snapshot along with fields that describe the loan (**Portfolio**, **Loan Type**, **Balance**, and **Open Date**) and fields for the member (**Name**, **Credit Score**, **Age**, and **State**).

In Tableau, you can create a quick LOD in one of two ways. First, you can hold *Ctrl* while dragging a measure onto a dimension. This will use the default aggregation of the measure (usually SUM) and generate a fixed level of detail calculation. For example, holding *Ctrl* and dragging **Balance** onto **Portfolio** will create a new calculated field named **Balance (Portfolio)** with the code {FIXED [Portfolio] : SUM(Balance)}. Alternately, you can hold *Ctrl* and click on a measure and dimension, and then use the drop-down menu on either one to select **Create | LOD Calculation....**

As in previous chapters, the goal is to understand key concepts and some key patterns. The following are only a few examples of all the possibilities presented by level of detail calculations.

Flagging members based on historical values

Let’s say branch management has determined that any member who has ever had a credit score of less than 550 is considered to be at risk of loan default and eligible for special assistance. Consider the history of the following three individuals:

Member ID	Member Name	Loan Type	Date	Balance	Credit Score
158	Vicki Modzelewski	Used Auto	1/1/2024	10,615	712
			2/1/2024	10,441	712
			3/1/2024	10,285	699
			4/1/2024	10,108	699
			5/1/2024	9,891	699
			6/1/2024	9,736	717
			7/1/2024	9,556	717
479	Thomas Villareal	Used Auto	2/1/2024	7,407	526
			3/1/2024	7,191	526
			4/1/2024	6,984	563
			5/1/2024	6,771	563
			6/1/2024	6,551	563
			7/1/2024	6,334	591
			8/1/2024	6,115	591
576	Charles Reeves	Used Auto	1/1/2024	28,145	610
			2/1/2024	27,187	610
			3/1/2024	26,226	535
			4/1/2024	25,267	535
			5/1/2024	24,302	535
			6/1/2024	23,337	530
			7/1/2024	22,366	530

Figure 5.6: Credit scores for three individuals with scores under the 550 threshold indicated via arrows

Every month, a new snapshot of history is recorded. Loan balances often change along with the member’s credit score. Some members have never been at risk. The first member, Vicki, has 699 as her lowest recorded score and has never been at risk. However, both Charles and Thomas had periods in their histories where their credit scores fell below the threshold (indicated with arrows in the preceding screenshot).

A simple row-level calculation, such as `[Credit Score] < 550`, could identify each record where the monthly snapshot of credit score indicated a risk. But members whose scores fluctuated *below* and *above* the threshold would have records that were alternately TRUE or FALSE.

We want every record for a given member to be TRUE if *any* of the records for that member are below the threshold and FALSE if *none* of the records are below the threshold.

One solution is to use a level of detail calculation, which we’ll name `Member Ever at Risk?`, with the following code:

```
{FIXED [Member ID] : MIN([Credit Score])} < 550
```

This calculation determines the lowest credit score for each member and compares it to the risk threshold of 550.

The result is the same for each record for a given member, as you can see here:

FIXED Example 1: Member Ever at Risk?

Member ID	Member Name	Loan Type	Date	Member Ever at Risk?	Balance	Credit Score
158	Vicki Modzelewski	Used Auto	1/1/2024	False	10,615	712
			2/1/2024	False	10,441	712
			3/1/2024	False	10,285	699
			4/1/2024	False	10,108	699
			5/1/2024	False	9,891	699
			6/1/2024	False	9,736	717
			7/1/2024	False	9,556	717
479	Thomas Villareal	Used Auto	2/1/2024	True	7,407	526
			3/1/2024	True	7,191	526
			4/1/2024	True	6,984	563
			5/1/2024	True	6,771	563
			6/1/2024	True	6,551	563
			7/1/2024	True	6,334	591
			8/1/2024	True	6,115	591
576	Charles Reeves	Used Auto	1/1/2024	True	28,145	610
			2/1/2024	True	27,187	610
			3/1/2024	True	26,226	535
			4/1/2024	True	25,267	535
			5/1/2024	True	24,302	535
			6/1/2024	True	23,337	530
			7/1/2024	True	22,366	530

Figure 5.7: The Member Ever at Risk? field is True or False for all records of a given member

Notice that every record contains the result for the relevant member. This illustrates one key concept for fixed level of detail calculations: *while the calculation is an aggregation at a defined level of detail, the results are at the row level*. That is, the TRUE or FALSE value is calculated at the member level, but the results are available as row-level values for every record for that member.

This enables all kinds of analysis possibilities, such as:

- Filtering to include only at-risk members but retaining all records for their history. If you instead filtered based on individual credit scores, you'd lose all records for parts of the history where the credit score was above the threshold. Those records might be critical for your analysis.
- Correctly counting members as either at risk or not while avoiding counting them for both cases if the history fluctuated.
- Comparing members who were or were not at risk at other levels of detail. For example, this view shows the number of members who were at risk or not at risk by portfolio:



Figure 5.8: We can implement some brushing to show what proportion of members has ever been at risk

Fixed level of detail calculations are context sensitive. That is, they operate within a context, which is either 1) the entire dataset or 2) defined by **context filters** (filters where the drop-down **Add to Context** option has been selected). In this example, that means the values calculated for each member will not change if you don't use context filters. Consider Thomas in Figure 5.7, who will always be considered at risk, even if you applied a normal filter that kept only dates after March 2024. That is because the fixed level of detail calculation would work across the entire dataset and find at-risk values in January and February. If you were to add such a filter to context, the result could change. This behavior of fixed level of detail calculations can be leveraged to aid your analysis but can also cause unexpected behavior if not understood.

This is a single example of the kinds of analysis made simple with level of detail calculations. There's so much more we can do, and we'll see another example next!

Latest balance for a member

Many datasets contain a series of events or a history of transactions. You may find yourself asking questions such as:

- What diagnoses are common for a patient's first visit to the hospital?
- What was the last reported status of each computer on the network?
- How much did each customer spend on their last order?
- How much did the first trade of the week make compared to the last?

None of these questions is simply asking when the earliest or latest event happened. A simple MIN or MAX aggregation of the date would provide that answer. But with these questions, there is the added complexity of asking for additional detail about what happened on the earliest or latest dates. For these kinds of questions, level of detail calculations provide a way to arrive at answers.

Consider the following three members' data contained in the Chapter 05 Loans dataset:

Member ID	Member Name	Loan Number	Loan Type	Date	
14827	Kelly Wooldridge	1	New Auto Plus	1/1/2024	21,684
				2/1/2024	21,348
				3/1/2024	21,001
				4/1/2024	21,001
				5/1/2024	20,327
				6/1/2024	19,987
				7/1/2024	19,646
16024	Joseph Clark	1	Used Auto	2/1/2024	19,043
				3/1/2024	18,656
				4/1/2024	18,263
				5/1/2024	17,873
				6/1/2024	17,479
				7/1/2024	17,087
				8/1/2024	16,691
16070	Gerald Quinney	1	1st Mortgage	3/1/2024	144,138
				4/1/2024	140,943
				5/1/2024	137,737
				6/1/2024	134,520
				7/1/2024	131,293
		2	Used Auto	3/1/2024	6,809
				4/1/2024	6,636
				5/1/2024	6,460
				6/1/2024	6,285
				7/1/2024	6,107
				8/1/2024	5,929
				9/1/2024	5,749

Figure 5.9: The data for three selected members in the Loans dataset

Each has a history of balances for each of their loans. However, the most recent date of history differs for each loan. Kelly's most recent balance is given for July. Joseph's latest balance is for August. Gerald has two loans: the first has the most recent balance for July and the second has the most recent balance for September.

What if you want to identify only the records that represent the latest known balance for a member? You might consider using a fixed level of detail calculation called `Latest Date per Member/Loan` with code such as this:

```
{FIXED [Member ID],[Loan Number] : MAX([Date])} = [Date]
```

This determines the maximum date per member per loan and compares the result to each row-level date, returning `TRUE` for matches and `FALSE` otherwise.

Two dimensions have been used to define the level of detail in the previous calculation because a single member can have more than one loan. If you had a truly unique identifier for each loan, you could alternatively use that as the single dimension defining the level of detail. You will need to have a good understanding of your data to accurately leverage level of detail calculations.

You can see the results of the calculation here:

FIXED Example 2: Most Recent Balance

Member ID	Member Name	Loan Number	Loan Type	Date	Latest Date per Member/Loan	
14827	Kelly Wooldridge	1	New Auto Plus	1/1/2024	False	21,684
				2/1/2024	False	21,348
				3/1/2024	False	21,001
				4/1/2024	False	21,001
				5/1/2024	False	20,327
				6/1/2024	False	19,987
				7/1/2024	True	19,646
16024	Joseph Clark	1	Used Auto	2/1/2024	False	19,043
				3/1/2024	False	18,656
				4/1/2024	False	18,263
				5/1/2024	False	17,873
				6/1/2024	False	17,479
				7/1/2024	False	17,087
				8/1/2024	True	16,691
16070	Gerald Quinney	1	1st Mortgage	3/1/2024	False	144,138
				4/1/2024	False	140,943
				5/1/2024	False	137,737
				6/1/2024	False	134,520
				7/1/2024	True	131,293
		2	Used Auto	3/1/2024	False	6,809
				4/1/2024	False	6,636
				5/1/2024	False	6,460
				6/1/2024	False	6,285
				7/1/2024	False	6,107
				8/1/2024	False	5,929
				9/1/2024	True	5,749

Figure 5.10: The latest date per loan per person is indicated by a `True` value for the calculation

If you had wanted to determine the first record for each loan, you would have simply changed MAX to MIN in the code. You can use the level of detail calculation's row-level TRUE/FALSE result as a filter to keep only the latest records, or even as part of other calculations to accomplish an analysis, such as comparing, starting, and ending balances.

The technique demonstrated by this calculation has many applications. In cases where data *trickles in*, you can identify the most recent records. In cases where you have duplication of records, you can filter to keep only the first or last. You can identify a customer's first or last purchase. You can compare historic balances to the original balance and much, much more!

We've just seen how fixed level of detail calculations can be leveraged to answer some complex questions. Let's continue by examining include level of detail expressions.

Examples of include level of detail expressions

Include level of detail calculations can be very useful when you need to perform certain calculations at levels of detail that are lower (more detailed) than the view level of detail. Let's take a look at an example.

Average loans per member

Some members have a single loan. Some have two or three or possibly more. What if we wanted to see how many loans the average member has on a state-by-state basis? Let's consider how we might go about that.

We'll start with a sheet where the view level of detail is **State**:

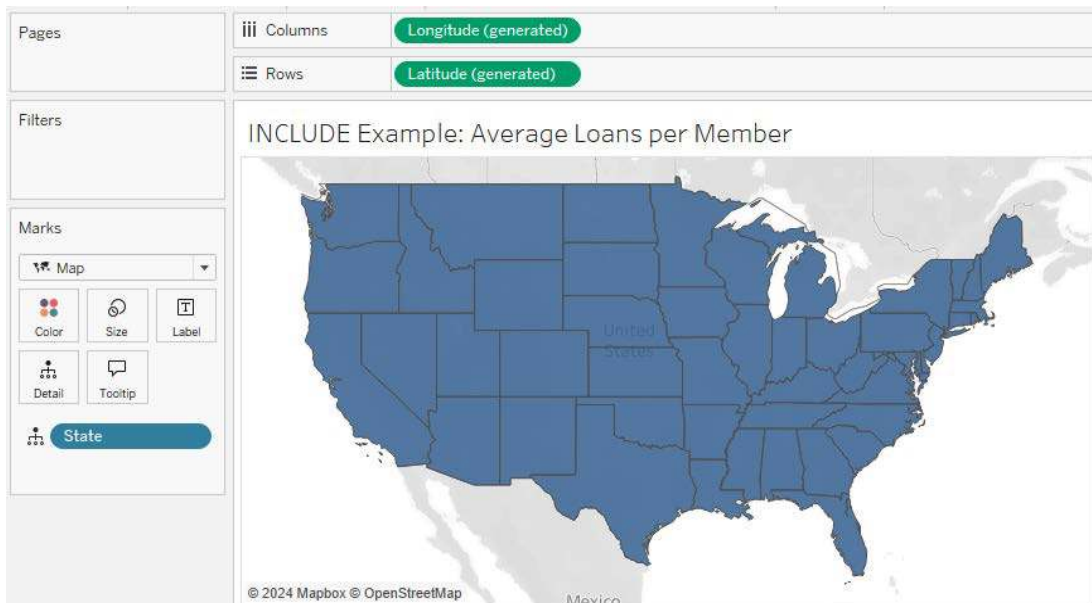


Figure 5.11: The starting place for the example—a filled map by state

It would be relatively easy to visualize the average credit score or average balance per state. But what if we want to visualize the average number of loans per member for each state? While there are several possible approaches to solving this kind of problem, here we'll consider using the following level of detail expression, named `Number of Loans per Member`:

```
{INCLUDE [Member ID] : COUNTD([Loan Number])}
```

This code returns a distinct count of loan numbers at a level of detail that includes the member ID along with all dimensions that define the view level of detail (state, in this case). When we add the calculation to the view, we'll need to decide how to aggregate it. In this case, we want the average number of loans per member, so we'll select **Measure | Average** from the dropdown on the field, revealing an interesting geographic pattern:

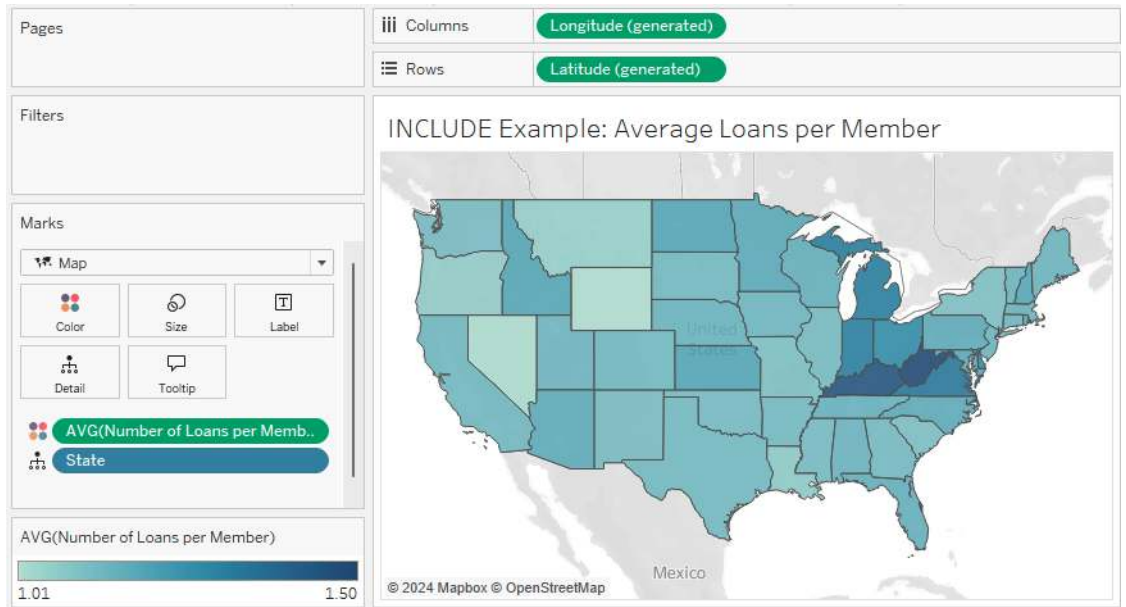


Figure 5.12: Using the include level of detail calculation to create a gradient of color to show average loans per member

As you think through how include level of detail calculations work, you might want to create a new sheet and work with the data in the form of a crosstab, prior to rearranging it visually:

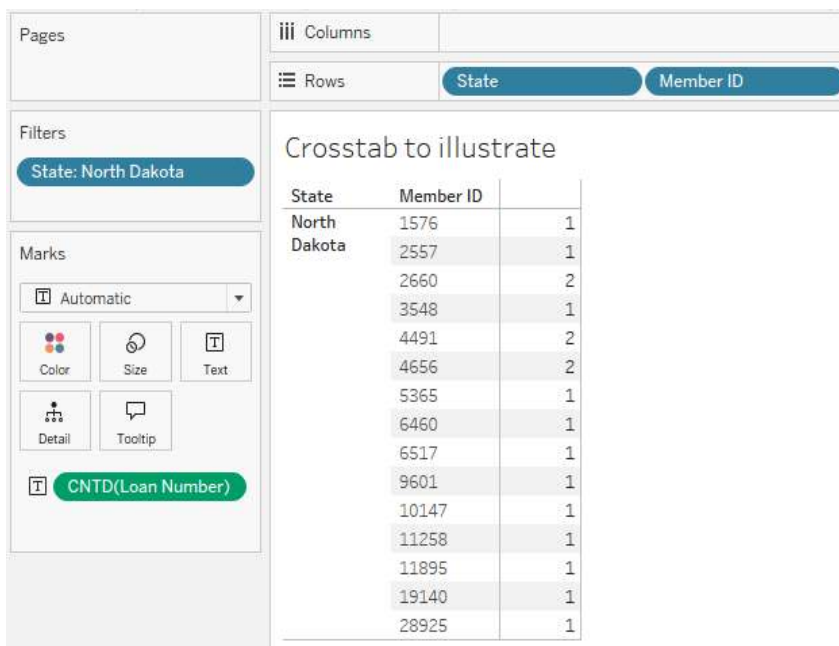


Figure 5.13: A crosstab helps illustrate how the distinct count of loans can be used as a basis for an average

State, the first dimension on **Rows**, comes from the view level of detail. **Member ID** has been included in the crosstab to simulate the dimension included in the level of detail expression. `COUNTD(Loan Number)` gives the number of loans per member. Averaging the values for all the members in the state gives the state average. A quick check for **North Dakota** gives us an average of 1.2 loans per member, which matches the map visualization exactly.

In this case, the include level of detail expression gives us a useful solution. There are some alternatives and it is helpful to consider some of these as you think through how you might solve similar issues. We'll consider those next.

Alternative approaches

It's worth noting that the above dataset actually allows you to use `MAX([Loan Number])` instead of `COUNTD([Loan Number])` as the number simply increments for each member based on how many loans they have. The highest number is identical to the number of loans for that member. In a significantly large dataset, the `MAX` calculation should perform better.

There are also a few other approaches to solving this problem, such as the calculation:

```
COUNTD(STR([Member ID]) + "_" + STR([Loan Number]))
/
COUNTD([Member ID])
```

This code takes the distinct count of loans and divides it by the distinct count of members. In order to count the distinct number of loans, the code creates a unique key by concatenating a string of a member ID and a loan number.

The aggregate calculation alternative has the advantage of working at any level of detail in your view. You may find either the level of detail or aggregate calculation to be easier to understand, and you will need to decide which best helps you to maintain a flow of thought as you tackle the problem at hand.

Another approach would be to use a fixed level of detail expression, such as:

```
{FIXED [State],[Member ID] : COUNTD([Loan Number])}
```

This calculation results in the same level of detail as the include expression and uses the same distinct count of loan number. It turns out that in this dataset, each member belongs to only a single state, so State wouldn't necessarily have to be included in the fixed level of detail expression. However, if you wanted to change the level of detail, you'd need to adjust the calculation, whereas, with the include expression, you'd only have to add or remove dimensions to or from the view.

With the include example, along with some alternatives, in mind, let's turn our attention to an example of exclude level of detail calculations.

Example of exclude level of detail calculations

Exclude level of detail calculations are useful when you want to perform certain calculations at higher (less detailed) levels than the view level of detail. The following example will demonstrate how we can leverage this functionality.

Average credit score per loan type

In this example, we'll answer the following question: how does the average credit score for a given loan type compare to the overall average for the entire portfolio?

Take the following view, which shows the average credit score per loan type (where loan types are grouped into portfolios):

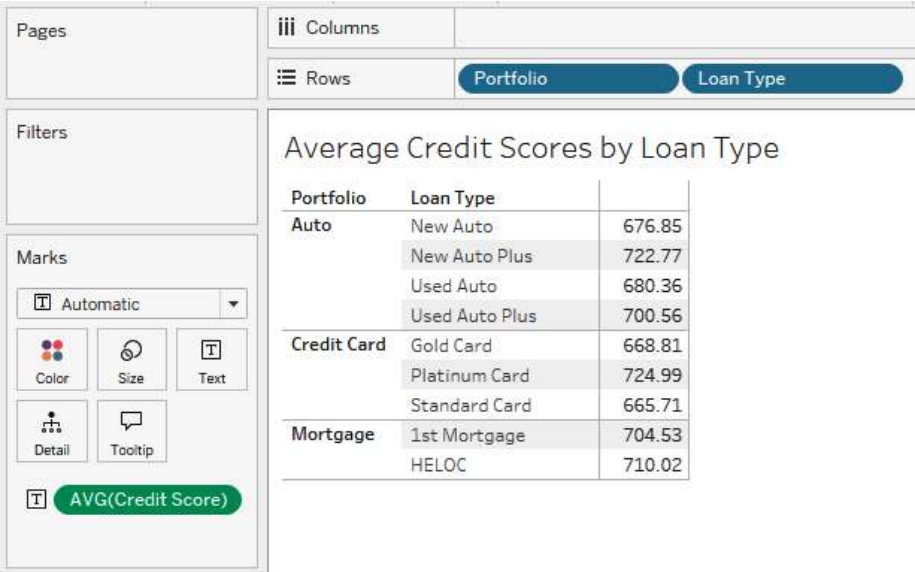


Figure 5.14: This crosstab shows the average credit score per loan type

What if we wanted to compare the average credit score of each loan type with the overall average credit score for the entire portfolio? We could accomplish this with an exclude level of detail calculation that looks like this:

```
{EXCLUDE [Loan Type] : AVG([Credit Score])}
```

This removes **Loan Type** from the level of detail and the average is calculated only per portfolio. This gives us results like the following:

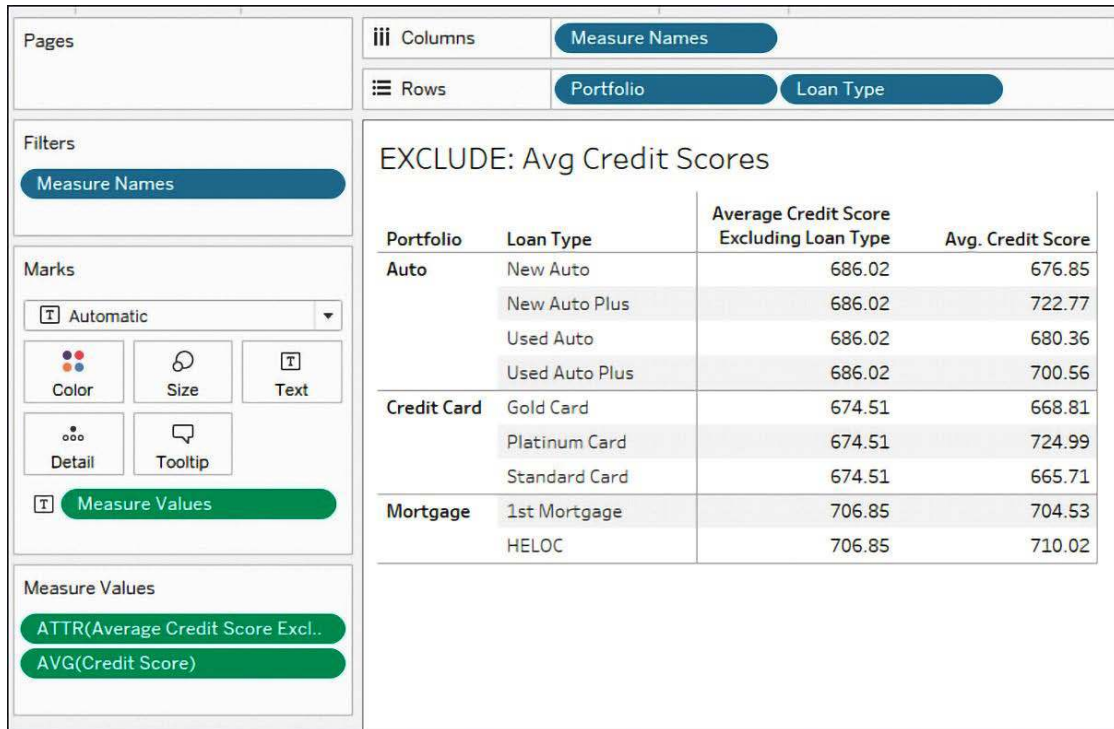


Figure 5.15: The exclude level of detail expression removes Loan Type so the average is only at the portfolio level

You'll notice that the same value for the average excluding loan type is repeated for each loan type. This is expected because the overall average is at the portfolio level and is not affected by the loan type. As is, this is perhaps not the most useful view. However, we can extend the calculation a bit to give us the difference between the overall portfolio average and the average of each loan type. The code would look like this:

```
AVG([Credit Score]) - AVG([Average Credit Score Excluding Loan Type])
```

This takes the average at the view level of detail (loan type and portfolio) and subtracts the average at the portfolio level to give us the difference between each loan type average and the overall portfolio average. We might rearrange the view to see the results visually, like this:

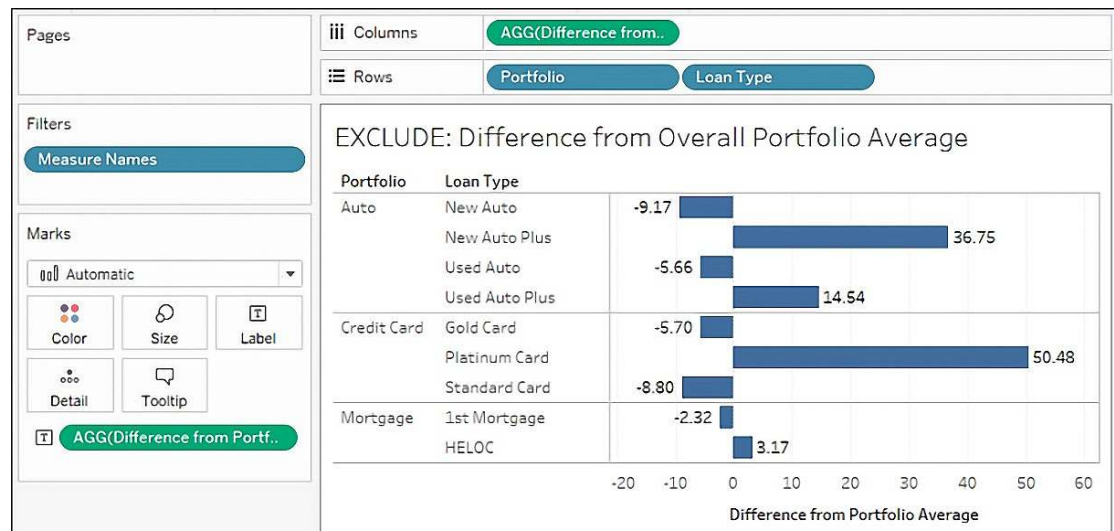


Figure 5.16: The final view shows the difference between the loan type average credit score and the overall portfolio average

Exclude level of detail expressions give us the ability to analyze differences between the view level of detail and higher levels of detail.

Summary

Level of detail expressions significantly enhance your calculation capabilities. They provide tools for working with data at various levels of detail. With fixed level of detail calculations, you can identify the first or last event in a series or whether a condition is ever true across entire subsets of data. With include expressions, you can work at lower levels of detail and then summarize those results in a view. With exclude expressions, you can work at higher levels of detail, greatly expanding analysis possibilities.

In the next chapter, we’ll explore the final main type of calculations: **table calculations**. These are some of the most powerful calculations in terms of their ability to solve problems, and they open up incredible possibilities for in-depth analysis. In practice, they range from very easy to exceptionally complex.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



6

Diving Deep with Table Calculations

Table calculations are a powerful feature in Tableau. They enable solutions that really couldn't be achieved any other way (short of writing a custom application or complex custom SQL scripts!). The features include the following:

- They make it possible to work with poorly structured data and still get quick results without waiting for fixes of the data at the source.
- They enable comparisons, allow for secondary aggregations, and perform calculations across multiple values within a visualization.
- They open up incredible possibilities for analysis and creative approaches to solving problems, highlighting insights, and improving the user experience.

Table calculations range in complexity, from incredibly easy to create (with just a couple of clicks) to extremely complex (requiring an understanding of **addressing**, **partitioning**, and **data densification**). We'll start off simple and move toward complexity in this chapter. The goal is to gain a solid foundation in creating and using table calculations, understanding how they work, and looking at some examples of how they can be used. We'll consider these topics:

- An overview of table calculations
- Creating and editing table calculations
- Quick table calculations
- Fundamental table calculation concepts
- Custom table calculations
- Practical examples

The examples in this chapter will return to the sample Superstore data that we used in the first chapter. To follow along with the examples, use the `Chapter 06 Starter.twbx` workbook.

An overview of table calculations

Table calculations are different from all other calculations in Tableau. Row-level, aggregate calculations, and **level of detail (LOD)** expressions, which we explored in the previous chapters, are performed as part of the query to the data source. If you were to examine the queries sent to the data source by Tableau, you'd find the code for your calculations translated into whatever implementation of SQL the data source used.



You may wish to refer to the order of operations discussed in *Chapter 2, Connecting to Data in Tableau*

Table calculations, on the other hand, are performed after the initial query. Here's a diagram that demonstrates how aggregated results are stored in Tableau's cache:

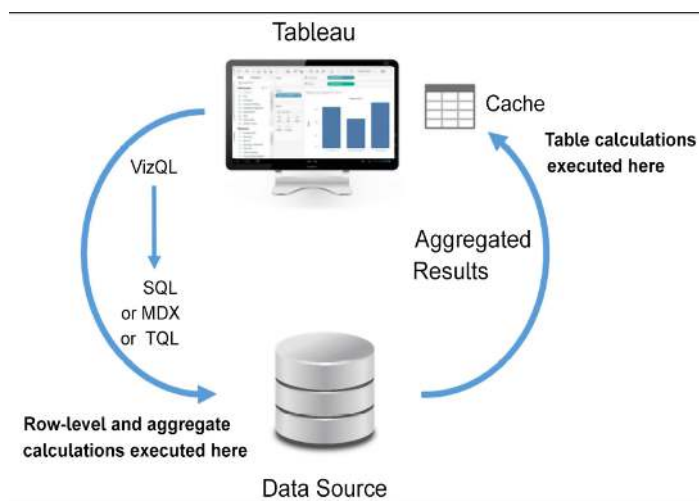


Figure 6.1: Table calculations are computed in Tableau's cache of aggregated data

Table calculations are performed on the aggregate table of data in Tableau's cache right before the data visualization is rendered. As we'll see, this is important to understand for multiple reasons, including the following:

- **Aggregation:** Table calculations operate on aggregate data. You cannot reference a field in a table calculation without referencing the field as an aggregate.
- **Filtering:** Regular filters will be applied before table calculations. This means that table calculations will only be applied to data returned from the source to the cache. You'll need to avoid filtering any data necessary for the table calculation.
- **Table calculation filtering** (sometimes called **late filtering**): Table calculations used as filters will be applied after the aggregate results are returned from the data source. The order of operations is important: row-level and aggregate filters are applied, aggregate data is returned to

the cache, and then the table calculation is applied as a filter that effectively hides data from the view. This allows some creative approaches to solving certain kinds of problems that we'll consider in some of the examples later in the chapter.

- **Performance:** If you are using a live connection to an enterprise database server, then row-level and aggregate-level calculations will be taking advantage of enterprise-level hardware. Table calculations are performed in the cache, which means they will be performed on whatever machine is running Tableau.



You will not likely need to be concerned if your table calculations are operating on a dozen or even hundreds of rows of aggregate data, or if you anticipate publishing to a powerful Tableau server. However, if you are getting back hundreds of thousands of rows of aggregate data on your local machine, then you'll need to consider the performance of your table calculations. At the same time, there are cases where table calculations might be used to avoid an expensive filter or calculation at the source.

With this overview of table calculations in mind, let's jump into understanding some options for creating table calculations.

Creating and editing table calculations

There are several ways to create table calculations in Tableau, including the following:

- Use the drop-down menu for any active field used as a numeric aggregate in the view, select **Quick Table Calculation**, and then the desired calculation type
- Use the drop-down menu for any active field that is used as a numeric aggregate in the view, select **Add Table Calculation**, then select the calculation type and adjust any desired settings
- Create a calculated field and use one or more table calculation functions to write your own custom table calculations

The first two options create a **quick table calculation**, which can be edited or removed using the drop-down menu on the field and by selecting **Edit Table Calculation...** or **Clear Table Calculation**. The third option creates a calculated field, which can be edited or deleted like any other calculated field.

A field on a shelf in the view that is using a table calculation, or which is a calculated field using table calculation functions, will have a delta symbol icon visible, as follows.

Following is a snippet of an active field without a table calculation:

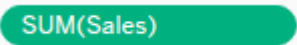


Figure 6.2: An active field without a table calculation applied

Following is the active field with a table calculation, with the delta symbol visible on the right side:



Figure 6.3: An active field with a table calculation applied includes the delta symbol

Most of the examples in this chapter will utilize text tables/cross-tab reports as these most closely match the actual aggregate table in the cache. This makes it easier to see how the table calculations are working.

Table calculations can be used in any type of visualization. However, when building a view that uses table calculations, especially more complex ones, try using a table with all dimensions on the **Rows** shelf and then adding table calculations as discrete values on **Rows** to the right of the dimensions. Once you have all the table calculations working as desired, you can rearrange the fields in the view to give you the appropriate visualization. This approach approximates the way table calculations are computed in the aggregate table in Tableau's cache and allows you to think through how the various dimensions and aggregations work together and impact your calculations.

We'll now move on from the concept of creating table calculations to some examples.

Quick table calculations

Quick table calculations are predefined table calculations that can be applied to fields used as measures in the view. These calculations include common and useful calculations such as **Running Total**, **Difference**, **Percent Difference**, **Percent of Total**, **Rank**, **Percentile**, **Moving Average**, **YTD Total** (year-to-date total), **Compound Growth Rate**, **Year over Year Growth**, and **YTD Growth**. You'll find applicable options in the drop-down list on a field used as a measure in the view, as shown in the following screenshot:

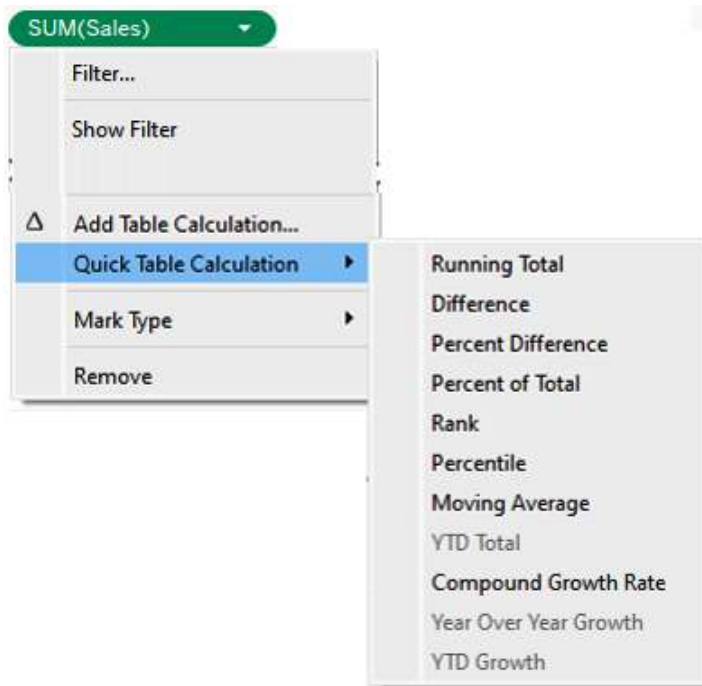


Figure 6.4: Using the dropdown, you can create a quick table calculation from an aggregate field in the view

Consider the following example using the sample Superstore Sales data:

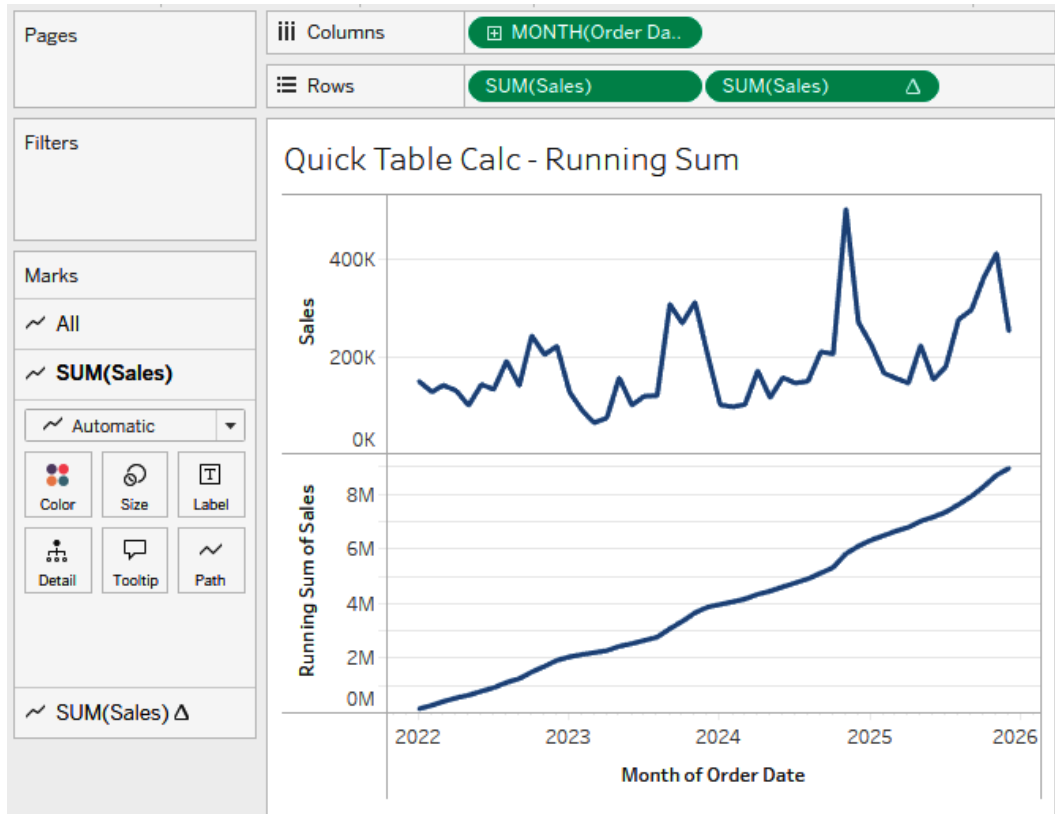


Figure 6.5: The first `SUM(Sales)` field is a normal aggregate. The second has a quick table calculation of Running Total applied

Here, **Sales** over time is shown. **Sales** has been placed on the **Rows** shelf twice and the second **SUM(Sales)** field has had the **running sum** quick table calculation applied. Using the quick table calculation meant it was unnecessary to write any code.

You can actually see the code that the quick table calculation uses by double-clicking the table calculation field in the view. This turns it into an **ad hoc calculation**. You can also drag an active field with a quick table calculation applied to the data pane, which will turn it into a calculated field that can be reused in other views.

The following table demonstrates some of the quick table calculations:

Some examples of Quick Table Calcs

Year of Order Date	Quarter of Order Date	Sales	Running Sum of Sales along Table (Down)	Difference in Sales from the Previous along Table (Down)	Rank of Sales along Table (Down)
2022	Q1	417,555	417,555		12
	Q2	372,289	789,844	-45,266	13
	Q3	464,319	1,254,163	92,030	10
	Q4	670,182	1,924,345	205,863	5
2023	Q1	279,148	2,203,493	-391,034	16
	Q2	330,269	2,533,762	51,121	14
	Q3	546,875	3,080,637	216,606	7
	Q4	788,255	3,868,892	241,380	3
2024	Q1	298,848	4,167,740	-489,407	15
	Q2	443,764	4,611,504	144,916	11
	Q3	505,453	5,116,957	61,689	9
	Q4	982,675	6,099,632	477,222	2
2025	Q1	547,656	6,647,288	-435,019	6
	Q2	521,650	7,168,938	-26,006	8
	Q3	752,933	7,921,871	231,283	4
	Q4	1,030,156	8,952,027	277,223	1

Figure 6.6: Sales in the first column is simply the SUM(Sales). The three additional columns show various table calculations applied (Running Sum, Difference, Rank)

Fundamental table calculation concepts

Although it is quite easy to create quick table calculations, it is essential to understand some fundamental concepts. We'll examine these concepts, starting with the difference between relative and fixed table calculations.

Relative versus fixed

It is important to understand that table calculations may be computed in one of the following two ways:

- **Relative:** Table calculations will be computed relative to the layout of the table. They might move across or down the table. Rearranging dimensions in a way that changes the table will change the table calculation results. As we'll see, the key to relative table calculations is **scope** and **direction**. When you set a table calculation to use a relative computation, it will continue to use the same relative scope and direction, even if you rearrange the view.

The term *relative* here is different from **Relative To**, which appears in the UI for some quick table calculations.

- **Fixed:** The table calculation will be computed using one or more dimensions. Rearranging those dimensions in the view will not change the computation of the table calculation. Here, the scope and direction remain fixed to one or more dimensions, no matter where they are moved within the view. When we talk about fixed table calculations, we'll focus on the concepts of **partitioning** and **addressing**.

You can see these concepts in the user interface. The following is the **Table Calculation** editor that appears when you select **Edit Table Calculation** from the menu of a **Table Calculation** field:

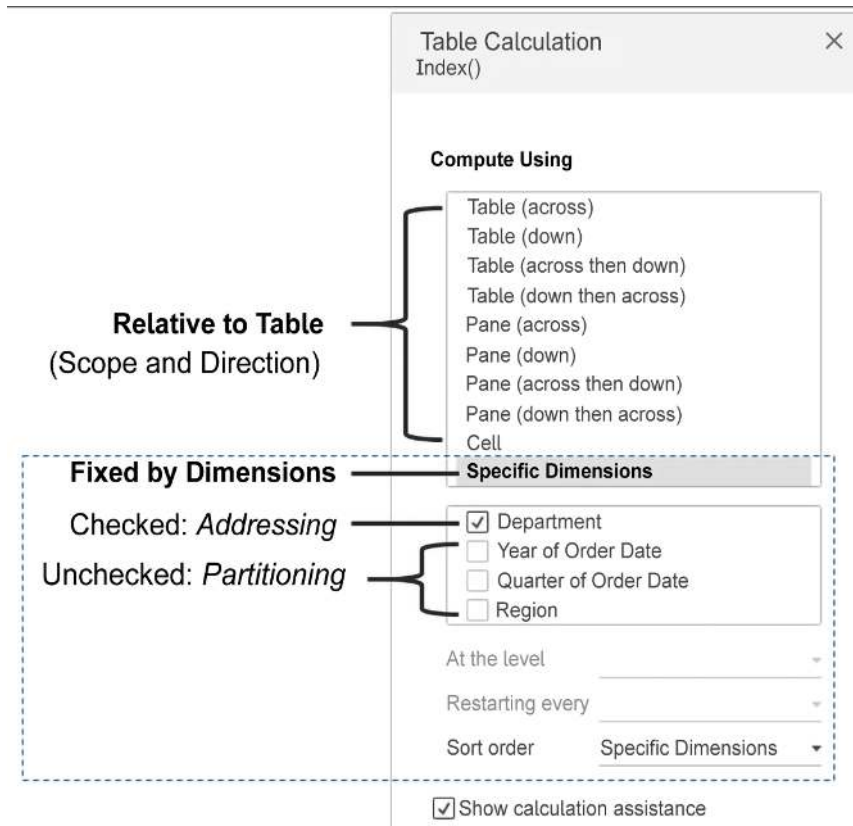


Figure 6.7: The Edit Table Calculation UI demonstrates the difference between Relative and Fixed table calculations

We'll explore the options and terms in more detail, but for now, notice the options that relate to specifying a table calculation that is computed relative to rows and columns, and options that specify a table calculation that is computed fixed to certain dimensions in the view.

Next, we'll look at **scope** and **direction**, which describe how relative table calculations operate.

Scope and direction

Scope and **direction** are terms that describe how a table calculation is computed relative to the table. Specifically, scope and direction refer to the following:

- **Scope:** The scope defines the boundaries within which a given table calculation can reference other values.
- **Direction:** The direction defines how the table calculation moves within the scope.

You've already seen table calculations being calculated in **Table (across)** (the running sum of sales over time in *Figure 6.5*) and **Table (down)** (in *Figure 6.6*). In these cases, the scope was the entire table and the direction was either across or down. For example, the running total calculation ran across the entire table, adding subsequent values as it moved from left to right.

To define the scope and direction for a table calculation, use the drop-down menu for the field in the view and select **Compute Using**. You will get a list of options that vary slightly depending on the location of dimensions in the view. The first of the options listed allows you to define the scope and direction relative to the table. After the option for **Cell**, you will see a list of dimensions present in the view. We'll look at those in the next section.

The options for scope and direction relative to the table are as follows:

- **Scope options:** Table, pane, and cells
- **Direction options:** Down, across, down then across, across then down

In order to understand these options, consider the following example:

The screenshot shows the Tableau interface with the following components:

- Columns:** Region, Department
- Rows:** YEAR(Order Date), QUARTER(Order Date)
- Filters:** Region, YEAR(Order Date)
- Marks:** SUM(Sales)

The main view is titled "Tables, Panes, Cells" and displays a table with the following data:

		Region / Department					
		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2021	Q1	103,094	21,517	47,770	41,391	6,221	24,258
	Q2	50,254	77,176	35,663	44,256	5,275	30,549
	Q3	33,016	34,014	30,213	32,794	30,200	119,668
	Q4	72,409	124,765	108,417	43,313	35,524	26,782
2022	Q1	55,241	26,176	19,534	20,296	9,793	28,652
	Q2	47,499	29,320	41,988	22,534	19,487	31,268
	Q3	102,001	44,762	81,264	46,923	15,995	74,876
	Q4	63,724	75,442	110,800	41,742	39,343	101,173
2023	Q1	33,938	20,638	23,858	14,296	17,108	13,947
	Q2	47,778	51,273	73,519	37,173	36,998	35,092
	Q3	83,456	59,039	32,302	34,504	16,305	76,077
	Q4	124,363	65,704	86,405	171,909	51,372	62,808
2024	Q1	51,050	74,600	83,149	53,471	24,272	33,619
	Q2	57,299	83,749	52,066	22,994	19,818	42,868
	Q3	92,243	51,459	70,612	56,791	62,900	66,427
	Q4	132,884	98,473	157,206	42,129	45,894	71,268

Figure 6.8: The difference between table, pane, and cell in the view

When it comes to the scope of table calculations, Tableau makes the following distinctions:

- The **table** is the entire set of aggregate data. The table is outlined in orange in *Figure 6.8*.
- The **pane** is a smaller section of the entire table. Technically, it is defined by the penultimate level of the table; that is, the next-to-last dimension on the **Rows** and/or **Columns** shelf defines

- the pane. In the preceding screenshot, you can see that the intersection of **Year** on rows and **Region** on columns defines the panes. One of the eight panes is outlined in blue in *Figure 6.8*.
- The **cell** is defined by the lowest level of the table. In this view, the intersection of one **Department** within a **Region** and one **Quarter** within a **Year** is a single cell. One of 96 cells is outlined in green in *Figure 6.8*.

The bounded areas in the preceding screenshot are defined by the scope. Scope (and, as we’ll see, also partition) defines windows within the data that contain various table calculations. Window functions, such as `WINDOW_SUM()` in particular, work within the confines of these windows.

In order to see how scope and direction work together, let’s work through a few examples. We’ll start by creating our own custom table calculations. Create a new calculated field named `Index` containing the code `Index()`.

`Index()` is a table calculation function that has an initial value of 1 and increments by one as it moves in a given direction and within a given scope. There are many practical uses for `Index()`, but we’ll use it here because it is easy to see how it is moving for a given scope and direction.

Create the table as shown in *Figure 6.8*, with `YEAR(Order Date)` and `QUARTER(Order Date)` on Rows and **Region** and **Department** on Columns. Instead of placing **Sales** in the view, add the newly created **Index** field to the Text shelf. Then, experiment using the drop-down menu on the **Index** field and select **Compute Using** to cycle through various scope and direction combinations. In the following examples, we’ve only kept the **East** and **West** regions and two years, **2023** and **2024**:

- **Table (across):** This is Tableau’s default when there are columns in the table. Notice in the following how **Index** increments across the entire table:



		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2023	Q1	1	2	3	4	5	6
	Q2	1	2	3	4	5	6
	Q3	1	2	3	4	5	6
	Q4	1	2	3	4	5	6
2024	Q1	1	2	3	4	5	6
	Q2	1	2	3	4	5	6
	Q3	1	2	3	4	5	6
	Q4	1	2	3	4	5	6

Figure 6.9: Table (across)

- **Table (down):** When using table (down), **Index** increments down the entire table:

		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2023	Q1	1	1	1	1	1	1
	Q2	2	2	2	2	2	2
	Q3	3	3	3	3	3	3
	Q4	4	4	4	4	4	4
2024	Q1	5	5	5	5	5	5
	Q2	6	6	6	6	6	6
	Q3	7	7	7	7	7	7
	Q4	8	8	8	8	8	8



Figure 6.10: Table (down)

- **Table (across then down):** This increments **Index** across the table, then steps down, continues to increment across, and repeats for the entire table:



		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2023	Q1	1	2	3	4	5	6
	Q2	7	8	9	10	11	12
	Q3	13	14	15	16	17	18
	Q4	19	20	21	22	23	24
2024	Q1	25	26	27	28	29	30
	Q2	31	32	33	34	35	36
	Q3	37	38	39	40	41	42
	Q4	43	44	45	46	47	48

Figure 6.11: Table (across then down)

- **Table (down then across):** This increments **Index** down the table, then steps across, continues to increment down, and repeats for the entire table:

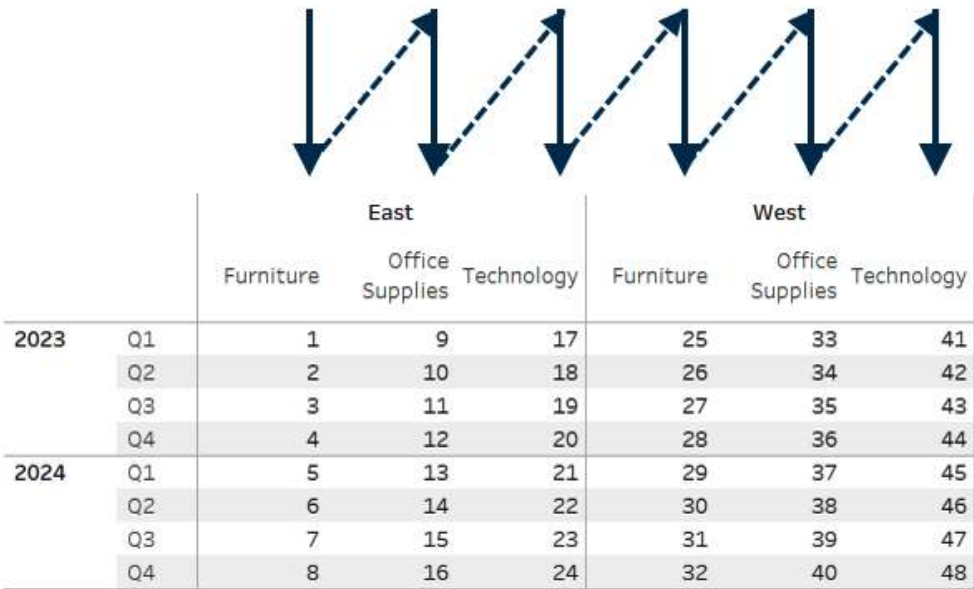


Figure 6.12: Table (down then across)

- **Pane (across):** This defines a boundary for **Index** and causes it to increment across until it reaches the pane boundary, at which point the indexing restarts:

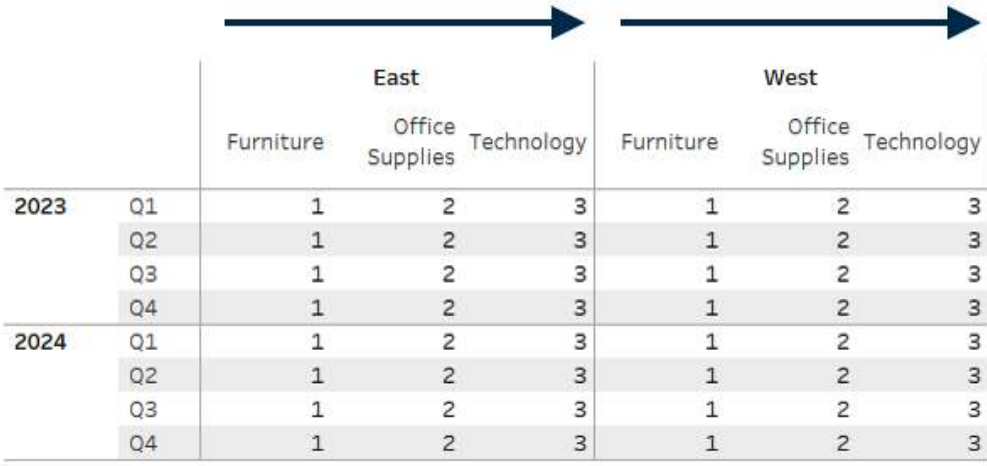


Figure 6.13: Pane (across)

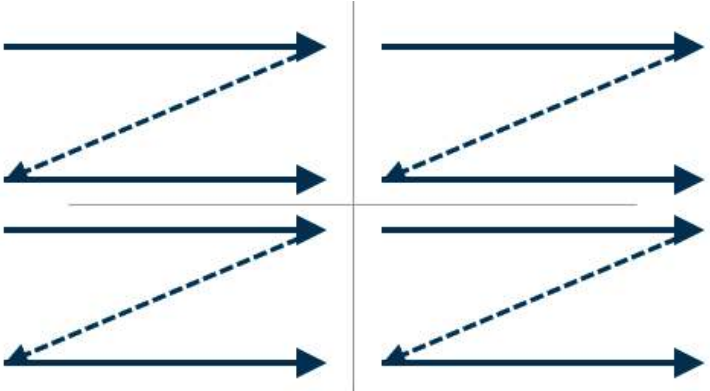
- **Pane (down):** This defines a boundary for **Index** and causes it to increment down until it reaches the pane boundary, at which point the indexing restarts:

		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2023	Q1	1	1	1	1	1	1
	Q2	2	2	2	2	2	2
	Q3	3	3	3	3	3	3
	Q4	4	4	4	4	4	4
2024	Q1	1	1	1	1	1	1
	Q2	2	2	2	2	2	2
	Q3	3	3	3	3	3	3
	Q4	4	4	4	4	4	4



Figure 6.14: Pane (down)

- **Pane (across then down):** This allows **Index** to increment across the pane and continue by stepping down. The pane defines the boundary here:



		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2023	Q1	1	2	3	1	2	3
	Q2	4	5	6	4	5	6
	Q3	7	8	9	7	8	9
	Q4	10	11	12	10	11	12
2024	Q1	1	2	3	1	2	3
	Q2	4	5	6	4	5	6
	Q3	7	8	9	7	8	9
	Q4	10	11	12	10	11	12

Figure 6.15: Pane (across then down)

- **Pane (down then across):** This allows **Index** to increment down the pane and continue by stepping across. The pane defines the boundary here:

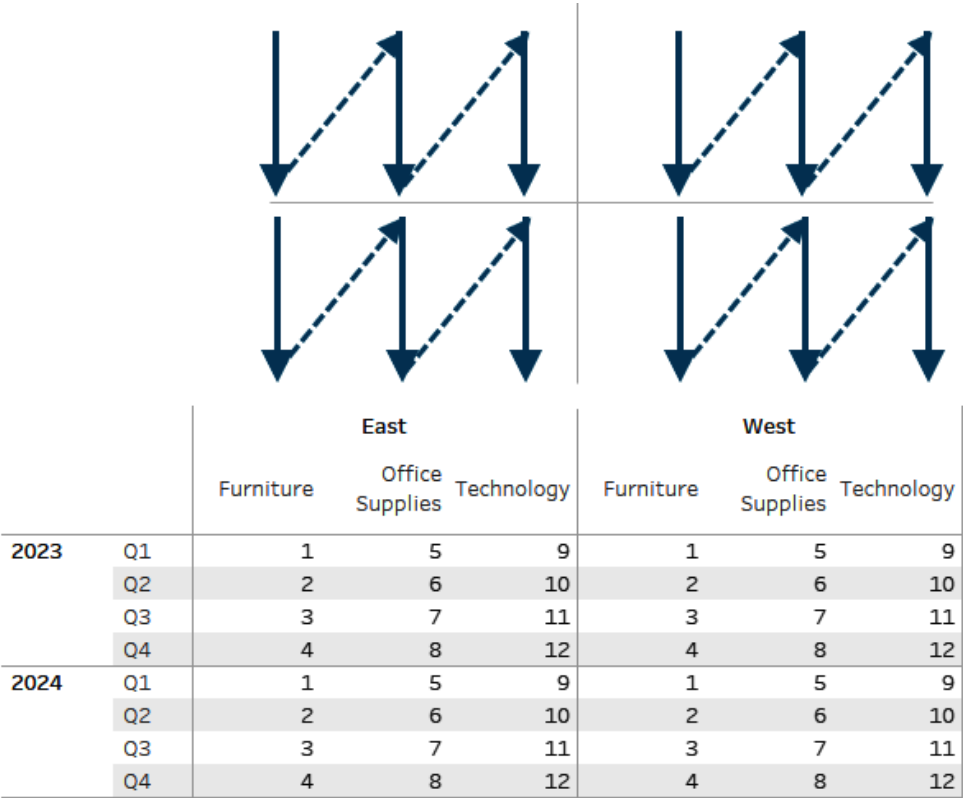


Figure 6.16: Pane (across then down)

You may use scope and direction with any table calculation. Consider how a running total or percentage difference would be calculated using the same movement and boundaries shown here. Keep experimenting with different options until you feel comfortable with how scope and direction work.

Scope and direction operate relative to the table, so you can rearrange fields, and the calculation will continue to work in the same scope and direction. For example, you could swap **Year of Order Date** with **Department** and still see **Index** calculated according to the scope and direction you defined.

Next, we’ll consider the corresponding concept for table calculations that are fixed to certain dimensions.

Addressing and partitioning

Addressing and **partitioning** are very similar to scope and direction but are typically used to describe how table calculations are computed with absolute reference to certain fields in the view. With addressing and partitioning, you define which dimensions in the view define the addressing (direction) and all others define the partitioning (scope).

Using addressing and partitioning gives you much finer control because your table calculations are no longer relative to the table layout, and you have many more options for fine-tuning the scope, direction, and order of the calculations.

To begin to understand how this works, let's consider a simple example. Using the preceding view, select **Edit table calculation** from the drop-down menu of the **Index** field on **Text**. In the resulting dialog box, check **Department** under **Specific Dimensions**.

The result of selecting **Department** is as follows:

The screenshot shows a Tableau interface with a table calculation view. The columns are Region (East, West) and Department (Furniture, Office Supplies, Technology). The rows are Year (2021-2024) and Quarter (Q1-Q4). The 'Index()' field is set to 'Department'. The 'Table Calculation' dialog box is open, showing the 'Compute Using' section with 'Specific Dimensions' selected. The 'Specific Dimensions' section has 'Department' checked, and 'Quarter of Order Date', 'Region', and 'Year of Order Date' are unchecked. The 'Sort order' is set to 'Specific Dimensions'.

		Region / Department					
		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2021	Q1	1	2	3	1	2	3
	Q2	1	2	3	1	2	3
	Q3	1	2	3	1	2	3
	Q4	1	2	3	1	2	3
2022	Q1	1	2	3	1	2	3
	Q2	1	2	3	1	2	3
	Q3	1	2	3	1	2	3
	Q4	1	2	3	1	2	3
2023	Q1	1	2	3	1	2	3
	Q2	1	2	3	1	2	3
	Q3	1	2	3	1	2	3
	Q4	1	2	3	1	2	3
2024	Q1	1	2	3	1	2	3
	Q2	1	2	3	1	2	3
	Q3	1	2	3	1	2	3
	Q4	1	2	3	1	2	3

Figure 6.17: Setting the table calculation to *Compute Using Specific Dimensions* uses addressing and partitioning

You'll notice that Tableau is computing **Index** along (in the direction of) the checked dimension, **Department**. In other words, you have used **Department** for addressing, so each new department increments the index. All other unchecked dimensions in the view are implicitly used for partitioning; that is, they define the scope or boundaries at which the index function must restart. As we saw with scope, these boundaries are sometimes referred to as a window.

The preceding view looks identical to what you would see if you set **Index** to compute using **Pane (across)**. However, there is a major difference. When you use **Pane (across)**, **Index** is always computed across the pane, even if you rearrange the dimensions in the view, remove some, or add others.

But when you compute using a dimension for addressing, the table calculation will always compute using that dimension. Removing that dimension will break the table calculation (the field will turn red with an exclamation mark) and you'll need to edit the table calculation via the drop-down menu to adjust the settings. If you rearrange dimensions in the view, **Index** will continue to be computed along the **Department** dimension.

Here, for example, is the result of clicking the **Swap Rows and Columns** button in the toolbar:

The screenshot shows the Tableau interface with a table calculation. The Columns shelf contains **YEAR(Order Date)** and **QUARTER(Order Date)**. The Rows shelf contains **Region** and **Department**. The Marks shelf has **Index()**. The view displays a table titled "Addressing and Partitioning" with columns for Region (East, West), Department (Furniture, Office Supplies, Technology), and Quarters (Q1, Q2, Q3, Q4) for 2021. The Index values are 1, 2, 3 for each department within a region.

The "Table Calculation" dialog box is open, showing the "Compute Using" section. The "Specific Dimensions" section is selected, and the "Department" checkbox is checked. The "At the level" dropdown is set to "Department", "Restarting every" is set to "Department", and "Sort order" is set to "Specific Dimensions". The "Show calculation assistance" checkbox is also checked.

		2021			
		Q1	Q2	Q3	Q4
East	Furniture	1	1	1	1
	Office Supplies	2	2	2	2
	Technology	3	3	3	3
West	Furniture	1	1	1	1
	Office Supplies	2	2	2	2
	Technology	3	3	3	3

Figure 6.18: Swap Rows and Columns does not change how this table calculation was computed as it is fixed to the dimensions rather than the table layout

Notice that **Index** continues to be computed along **Department** even though the entire orientation of the table has changed. To complete the following examples, we'll undo the swap of rows and columns to return our table to its original orientation.

Let's consider a few other examples of what happens when you add additional dimensions. For example, if you check **Quarter of Order Date**, you'll see Tableau highlight a partition defined by **Region** and **Year of Order Date**, with **Index** incrementing by the addressing fields of **Quarter of Order Date** and then **Department**:

Columns: Region, Department
Rows: YEAR(Order Date), QUARTER(Order Date)

Filters: Region, YEAR(Order Date)

Marks: Automatic, Color, Size, Text, Detail, Tooltip, Index()

Addressing and Partitioning

		Region / Department					
		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2023	Q1	1	5	9	1	5	9
	Q2	2	6	10	2	6	10
	Q3	3	7	11	3	7	11
	Q4	4	8	12	4	8	12
2024	Q1	1	5	9	1	5	9
	Q2	2	6	10	2	6	10
	Q3	3	7	11	3	7	11
	Q4	4	8	12	4	8	12

Table Calculation
 Index()

Compute Using

- Table (across)
- Table (down)
- Table (across then down)
- Table (down then across)
- Pane (across)
- Pane (down)
- Pane (across then down)
- Pane (down then across)
- Cell

Specific Dimensions

- ☒ Department
- ☒ Quarter of Order Date
- ☐ Region
- ☐ Year of Order Date

At the level: **Deepest**

Restarting every: **None**

Sort order: **Specific Dimensions**

☒ Show calculation assistance

Figure 6.19: Adding dimensions alters the table calculation's behavior. One of the resulting partitions is highlighted

If you were to select **Department** and **Year of Order Date** as the addressing of **Index**, you'd see a single partition defined by **Region** and **Quarter of Order Date**, like this:

Columns: Region, Department
Rows: YEAR(Order Date), QUARTER(Order Date)

Filters: Region, YEAR(Order Date)

Marks: Automatic, Color, Size, Text, Detail, Tooltip, Index()

Addressing and Partitioning

		Region / Department					
		East			West		
		Furniture	Office Supplies	Technology	Furniture	Office Supplies	Technology
2023	Q1	1	3	5	1	3	5
	Q2	1	3	5	1	3	5
	Q3	1	3	5	1	3	5
	Q4	1	3	5	1	3	5
2024	Q1	2	4	6	2	4	6
	Q2	2	4	6	2	4	6
	Q3	2	4	6	2	4	6
	Q4	2	4	6	2	4	6

Table Calculation
 Index()

Compute Using

- Table (across)
- Table (down)
- Table (across then down)
- Table (down then across)
- Pane (across)
- Pane (down)
- Pane (across then down)
- Pane (down then across)
- Cell

Specific Dimensions

- ☒ Department
- ☐ Quarter of Order Date
- ☐ Region
- ☒ Year of Order Date

At the level: **Deepest**

Restarting every: **None**

Sort order: **Specific Dimensions**

☒ Show calculation assistance

Figure 6.20: Changing the checked dimensions alters the table calculation's behavior. One of the resulting partitions is highlighted

You'll notice, in this view, that **Index** increments for every combination of **Year of Order Date** and **Department** within the partition of **Quarter of Order Date** and **Region**.

These are a few of the other things to consider when working with addressing and partitioning:

- You can specify the sort order. For example, if you wanted **Index** to increment according to the value of the sum of sales, you could use the drop-down list at the bottom of the table calculation editor to define a custom sort based on the sum of sales.
- The **At the Level** option in the edit table calculation dialog box allows you to specify a level at which the table calculations are performed. Most of the time, you'll leave this set at **Deepest** (which is the same as setting it to the bottom-most dimension), but occasionally, you might want to set it at a different level if you need to keep certain dimensions from defining the partition but need the table calculation to be applied at a higher level. You can also reorder the dimensions by dragging and dropping within the checkbox list of **Specific Dimensions**.
- The **Restarting Every** option effectively makes the selected field, and all dimensions in the addressing above that selected field, part of the partition, but allows you to maintain the fine-tuning of the ordering.
- Dimensions are the only kinds of fields that can be used in addressing; however, a discrete (blue) measure can be used to partition table calculations. To enable this, use the drop-down menu on the field and uncheck **Ignore in Table Calculations**.

Take some time to experiment with various options and become comfortable with how addressing and partitioning work. Next, we'll look at how to write our own custom table calculations.

Custom table calculations

Before we move on to some practical examples, let's briefly discuss how to write your own table calculations, instead of using quick table calculations. You can see a list of available table calculation functions by creating a new calculation and selecting **Table Calculation** from the drop-down list under **Functions**.

For each of the examples, we'll set **Compute Using** | **Category**. This means that **Department** will be the partition.

You can think of table calculations broken down into several categories. The table calculations explored in the following subsections can be combined and even nested just like other functions.

Meta table functions

These are the functions that give you information about partitioning and addressing. These functions also include **Index**, **First**, **Last**, and **Size**:

- **Index** gives an increment as it moves along the addressing within the partition
- **First** gives the offset from the first row in the partition, so the first row in each partition is 0, the next row is -1, then -2, and so on

- **Last** gives the offset to the last row in the partition, so the last row in each partition is 0, the next-to-last row is 1, then 2, and so on
- **Size** gives the size of the partition

The following screenshot illustrates the various functions:

Meta Table Calculations

Department	Category	Index along Category	First along Category	Last along Category	Size along Category
Furniture	Bookcases	1	0	3	4
	Chairs & Chairmats	2	-1	2	4
	Office Furnishings	3	-2	1	4
	Tables	4	-3	0	4
Office Supplies	Appliances	1	0	8	9
	Binders and Binder Accessories	2	-1	7	9
	Envelopes	3	-2	6	9
	Labels	4	-3	5	9
	Paper	5	-4	4	9
	Pens & Art Supplies	6	-5	3	9
	Rubber Bands	7	-6	2	9
	Scissors, Rulers and Trimmers	8	-7	1	9
	Storage & Organization	9	-8	0	9
Technology	Computer Peripherals	1	0	3	4
	Copiers and Fax	2	-1	2	4
	Office Machines	3	-2	1	4
	Telephones and Communication	4	-3	0	4

Figure 6.21: Meta table calculations

Index, **First**, and **Last** are all affected by scope/partition and direction/addressing, while **Size** will give the same result at each address of the partition, no matter what direction is specified.

Lookup and previous value

The first of these two functions gives you the ability to reference values in other rows, while the second gives you the ability to carry forward values.

Notice from the following screenshot that direction is very important for these two functions:

Lookup and Previous Value

Department	Category	Lookup	Previous Value
Furniture	Bookcases	Null	,Bookcases
	Chairs & Chairmats	Bookcases	,Bookcases,Chairs & Chairmats
	Office Furnishings	Chairs & Chairmats	,Bookcases,Chairs & Chairmats,Office Furnishings
	Tables	Office Furnishings	,Bookcases,Chairs & Chairmats,Office Furnishings,Tables
Office Supplies	Appliances	Null	,Appliances
	Binders and Binder Accessor...	Appliances	,Appliances,Binders and Binder Accessories
	Envelopes	Binders and Binder Acc...	,Appliances,Binders and Binder Accessories,Envelopes
	Labels	Envelopes	,Appliances,Binders and Binder Accessories,Envelopes,Labels
	Paper	Labels	,Appliances,Binders and Binder Accessories,Envelopes,Labels,Paper
	Pens & Art Supplies	Paper	,Appliances,Binders and Binder Accessories,Envelopes,Labels,Paper,Pens & Art Supplies
	Rubber Bands	Pens & Art Supplies	,Appliances,Binders and Binder Accessories,Envelopes,Labels,Paper,Pens & Art Supplies,Rubber Bands
Technology	Computer Peripherals	Null	,Computer Peripherals
	Copiers and Fax	Computer Peripherals	,Computer Peripherals,Copiers and Fax
	Office Machines	Copiers and Fax	,Computer Peripherals,Copiers and Fax,Office Machines
	Telephones and Communication	Office Machines	,Computer Peripherals,Copiers and Fax,Office Machines,Telephones and Communication

Figure 6.22: Lookup and Previous_Value functions (though Previous_Value includes some additional logic described below)

Both calculations are computed using the addressing of **Category** (so **Department** is the partition). Here, we’ve used the code `Lookup(ATTR([Category]), -1)`, which looks up the value of the category in the row offset by -1 from the current one. The first row in each partition gets a NULL result from the lookup (because there isn’t a row before it).

For `Previous_Value`, we used this code:

```
Previous_Value("") + "," + ATTR([Category])
```

 **Quick tip:** Enhance your coding experience with the **AI Code Explainer** and **Quick Copy** features. Open this book in the next-gen Packt Reader. Click the **Copy** button

(1) to quickly copy code into your coding environment, or click the **Explain** button

(2) to get the AI assistant to explain a block of code to you.



```
function calculate(a, b) {
  return {sum: a + b};
};
```

Copy

Explain

1

2

 **The next-gen Packt Reader** is included for free with the purchase of this book. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



Notice that in the first row of each partition, there is no previous value, so `Previous_Value()` simply returned what we specified as the default: an empty string. This was then concatenated together with a comma and the category in that row, giving us the value **Bookcases**.

 If you wished to avoid having the initial comma, you could write a slightly more complex calculation that checked the length of the previous value and only appended the comma if there was already a value, like this:

```
IF LEN(Previous_Value("")) == 0
THEN ATTR([Category])
ELSE Previous_Value("") + "," + ATTR([Category])
END
```

In the second row, **Bookcases** is the previous value, which gets concatenated with a comma and the category in that row, giving us the value **Bookcases, Chairs & Chairmats**, which becomes the previous value in the next row. The pattern continues throughout the partition and then restarts in the partition defined by the **Office Supplies** department.

Running functions

These functions run along direction/addressing and include `Running_Avg()`, `Running_Count()`, `Running_Sum()`, `Running_Min()`, and `Running_Max()`, as follows:

Running Functions

Department	Category	Sales	Running Sum of Sales along Category	Running Min of Sales along Category
Furniture	Bookcases	507,496	507,496	507,496
	Chairs & Chairmats	1,164,586	1,672,082	507,496
	Office Furnishings	444,634	2,116,716	444,634
	Tables	1,061,922	3,178,638	444,634
Office Supplies	Appliances	456,736	456,736	456,736
	Binders and Binder ..	638,583	1,095,319	456,736
	Envelopes	147,915	1,243,234	147,915
	Labels	23,446	1,266,680	23,446
	Paper	253,620	1,520,300	23,446
	Pens & Art Supplies	103,265	1,623,565	23,446
	Rubber Bands	8,670	1,632,235	8,670
	Scissors, Rulers and ..	40,432	1,672,667	8,670
	Storage & Organizat..	585,717	2,258,384	8,670
Technology	Computer Periphera..	490,851	490,851	490,851
	Copiers and Fax	661,215	1,152,066	490,851
	Office Machines	1,218,655	2,370,721	490,851
	Telephones and Com..	1,144,284	3,515,005	490,851

Figure 6.23: Running Functions

Notice that `Running_Sum(SUM[Sales])` continues to add the sum of sales to a running total for every row in the partition. `Running_Min(SUM[Sales])` keeps the value of the sum of sales if it is the smallest value it has encountered so far as it moves along the rows of the partition.

Window functions

These functions operate across all rows in the partition at once and essentially aggregate the aggregates. They include `Window_Sum`, `Window_Avg`, `Window_Max`, and `Window_Min`, among others, as shown in the following screenshot:

Window Functions

Department	Category	Sales	Window Sum along Category	Window Max along Category
Furniture	Bookcases	507,496	3,178,638	1,164,586
	Chairs & Chairmats	1,164,586	3,178,638	1,164,586
	Office Furnishings	444,634	3,178,638	1,164,586
	Tables	1,061,922	3,178,638	1,164,586
Office Supplies	Appliances	456,736	2,258,384	638,583
	Binders and Binder ..	638,583	2,258,384	638,583
	Envelopes	147,915	2,258,384	638,583
	Labels	23,446	2,258,384	638,583
	Paper	253,620	2,258,384	638,583
	Pens & Art Supplies	103,265	2,258,384	638,583
	Rubber Bands	8,670	2,258,384	638,583
	Scissors, Rulers and ..	40,432	2,258,384	638,583
	Storage & Organizat..	585,717	2,258,384	638,583
Technology	Computer Periphera..	490,851	3,515,005	1,218,655
	Copiers and Fax	661,215	3,515,005	1,218,655
	Office Machines	1,218,655	3,515,005	1,218,655
	Telephones and Com..	1,144,284	3,515,005	1,218,655

Figure 6.24: Examples of window functions

`Window_Sum(SUM([Sales]))` adds up the sums of sales within the entire window (in this case, for all categories within the department). `Window_Max(SUM([Sales]))` returns the maximum sum of sales within the window.

You may pass optional parameters to window functions to further limit the scope of the window. The window will always be limited to, at most, the partition.

Rank functions

These functions provide various ways to rank based on aggregate values. There are multiple variations of rank, which allow you to decide how to deal with ties (equal values) and how dense the ranking should be. The following screenshot simply uses the standard `RANK()` function:

Rank Functions

Department	Category	Sales	Rank along Category
Furniture	Bookcases	507,496	3
	Chairs & Chairmats	1,164,586	1
	Office Furnishings	444,634	4
	Tables	1,061,922	2
Office Supplies	Appliances	456,736	3
	Binders and Binder ..	638,583	1
	Envelopes	147,915	5
	Labels	23,446	8
	Paper	253,620	4
	Pens & Art Supplies	103,265	6
	Rubber Bands	8,670	9
	Scissors, Rulers and ..	40,432	7
	Storage & Organizat...	585,717	2
Technology	Computer Periphera...	490,851	4
	Copiers and Fax	661,215	3
	Office Machines	1,218,655	1
	Telephones and Com..	1,144,284	2

Figure 6.25: Example of `RANK()` function

The `Rank(SUM([Sales]))` calculation returns the rank of the sum of sales for categories within the department.

Script functions

These functions allow integration with the R analytics platform or Python, either of which can incorporate simple or complex scripts for everything from advanced statistics to predictive modeling. We'll consider various examples of this functionality in *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*.

The Total function

The `Total` function deserves its own category because it works a little differently from the others. Unlike the other functions, which work on the aggregate table in the cache, `Total` will re-query the underlying source for all the source data rows that make up a given partition. In most cases, this will yield the same result as a window function.

For example, `Total(SUM([Sales]))` gives the same result as `Window_Sum(SUM([Sales]))`, but `Total(AVG([Sales]))` will almost certainly give a different result from `Window_AVG(SUM([Sales]))` because `Total` gives you the actual average of underlying rows, while the `Window` function averages the sums, as can be seen here:

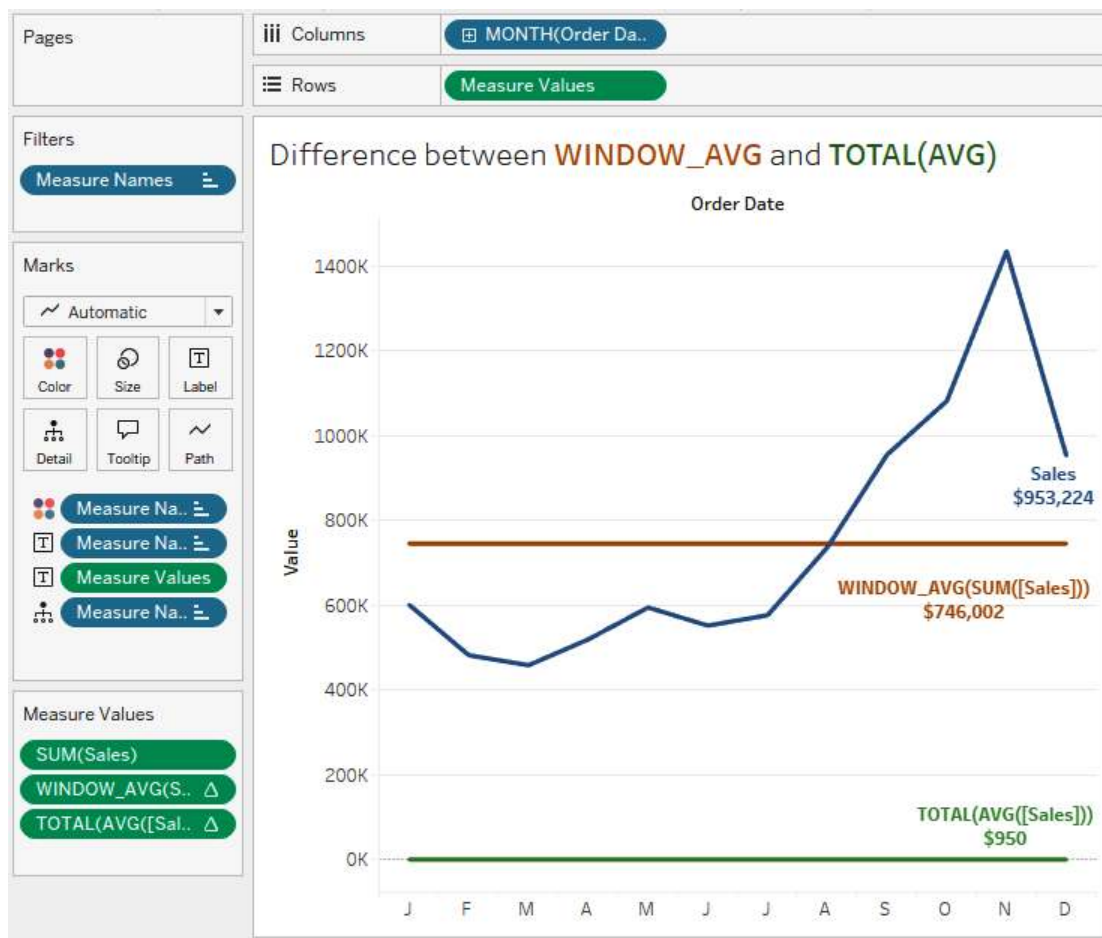


Figure 6.26: The difference between `WINDOW_AVG()` and `TOTAL(AVG)`

As you can see, `WINDOW_AVG()` gives you the average of the sums shown by the **Sales** line. `TOTAL(AVG())` gives the overall total average of the underlying data set (in other words, the average sales line is \$950).

In this section, we have looked at a number of table calculation functions. These will give you the building blocks to solve all kinds of practical problems and answer a great many questions. From ranking to year-over-year comparisons, you now have a foundation for success. Let's now move on to some practical examples.

Practical examples

Having looked at some of the essential concepts of table calculations, let’s consider some practical examples. We’ll look at several examples, although the practical use of table calculations is nearly endless. You can do everything from running sums and analyzing year-over-year growth to viewing percentage differences between categories, and much more.

Year-over-year growth

Often, you may want to compare year-over-year values. How much has our customer base grown over the last year? How did sales in each quarter compare to sales in the same quarter last year? These types of questions can be answered using **Year over Year Growth**.

Tableau exposes **Year over Year Growth** as one option in the quick table calculations. Here, for example, is a view that demonstrates **Sales** by **Quarter**, along with the percentage difference in sales for a quarter compared with the previous year:



Figure 6.27: Year-over-year growth of Sales

The second **Sum(Sales)** field has had the **Year over Year Growth** quick table calculation applied (and the **Mark** type changed to **Bar**). You’ll notice the **>4 nulls** indicator in the lower-right corner, alerting you to the fact that there are at least four null values (which makes sense as there is no 2021 with which to compare quarters in 2022).

If you filtered out 2022, the nulls that would appear in 2023 as table calculations can only operate on values present in the aggregated data in the cache. Any regular filters applied to the data are applied at the source and the excluded data never makes it to the cache.

As easy as it is to build a view like this example, take care because Tableau assumes each year in the view has the same number of quarters. For example, if the data for Q1 in 2022 was not present or filtered out, then the resulting view would not necessarily represent what you want. Consider the following, for example:

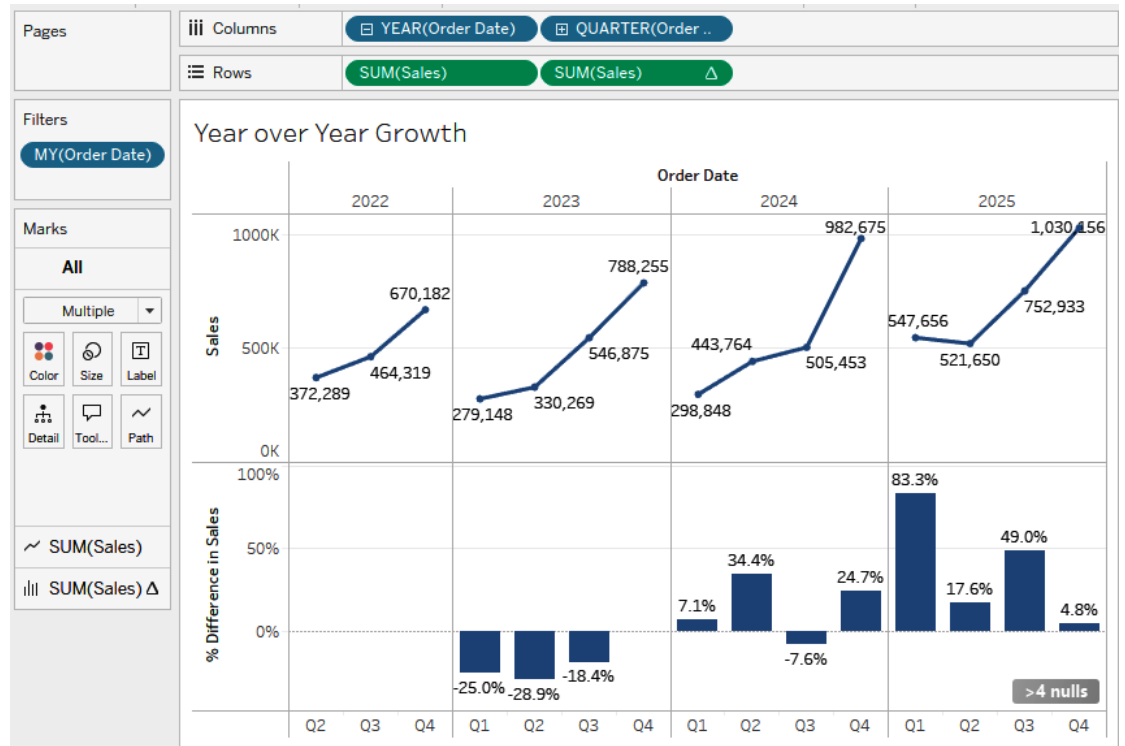


Figure 6.28: Year-over-year growth of Sales—but it doesn't work with Q1 missing in the first year

The problem here is that Tableau calculates the quick table calculation using the addressing of **Year** and **Quarter** and an *At the Level* of value of **Year of Order Date**. This works assuming all quarters are present. However, here, the first quarter in 2023 is matched with the first quarter present in 2022, which is really Q2. To solve this, you would need to edit the table calculation to only use **Year** for addressing. **Quarter** then becomes the partition and thus, comparisons are done for the correct quarter.

An additional issue arises for consideration: what if you don't want to show 2022 in the view? Filtering it out will cause issues for 2023. In this case, we'll look at **table calculation filtering**, or **late filtering**, later in this section. Another potential way to remove 2021 but keep access to its data values is to right-click the 2021 header in the view and select **Hide**.

Hide is a special command that simply keeps Tableau from rendering data, even when it is present in the cache. If you later decide you want to show 2021 after hiding it, you can use the menu for the **YEAR(Order**

Date) field and select **Show Hidden Data**. Alternatively, you can use the menu to select **Analysis | Reveal Hidden Data**.

You may also wish to hide the null indicator in the view. You can do this by right-clicking the indicator and selecting **Hide Indicator**. Clicking the indicator will reveal options to filter the data or display it as a default value (typically, 0).

Year-over-year growth (or any period over another) is a common analytical question that table calculations allow you to answer. Next, let’s consider another example of table calculations in practice.

Dynamic titles with totals

You’ve likely noticed the titles that are displayed for every view. There are also captions that are not shown unless you specifically turn them on (to do this, select **Worksheet | Show Caption** from the menu).

By default, the title displays the sheet name and captions are hidden, but you can show and modify them. At times, you might want to display totals that help your end users understand the broad context or immediately grasp the magnitude of the data.

Here, for example, is a view that allows the user to select one or more regions and then see **Sales Per State in each Region**:

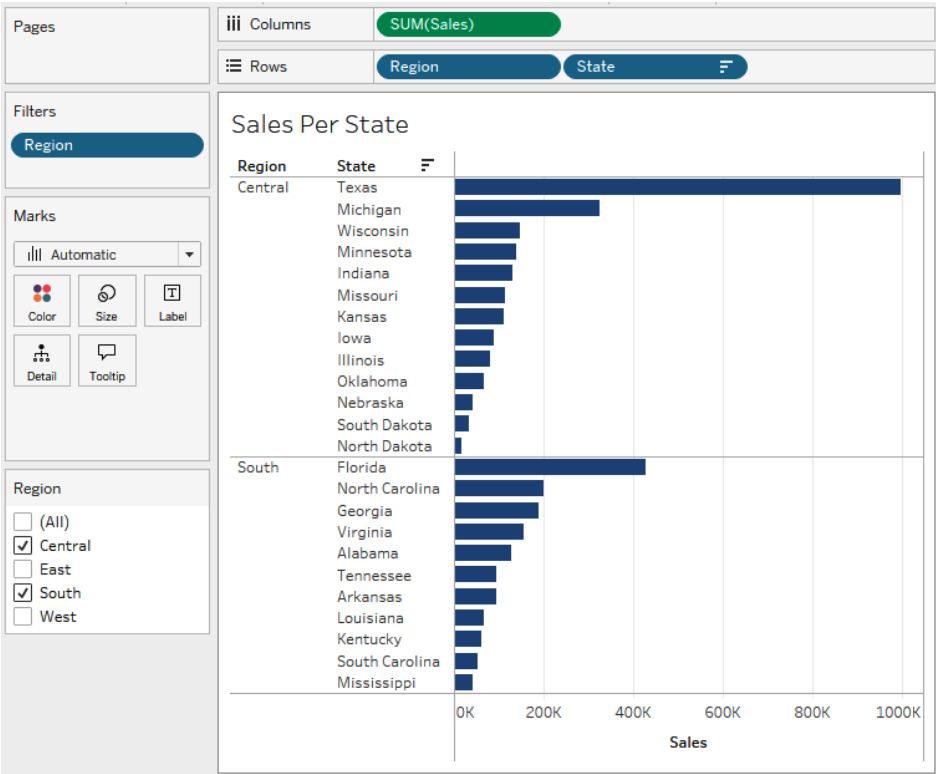


Figure 6.29: Sales Per State for two regions

It might be useful to show a changing number of states in the title as the user selects different regions. You might first think of using an aggregation on **State**, such as `Count Distinct`. However, if you try showing that in the title, you will always see the value 1. Why? Because the view level of detail is **State** and the distinct count of states per state is 1!

However, there are some options with table calculations that let you further aggregate aggregates. Or, you might think of determining the number of values in the table based on the size of the window. In fact, here are several possibilities:

- To get the total distinct count: `TOTAL(COUNTD([State]))`
- To get the sum within the window: `WINDOW_SUM(SUM(1))`
- To get the size of the window: `SIZE()`

You may recall that a window is defined as the boundaries determined by scope or partition. Whichever we choose, we want to define the window as the entire table. Either a relative computation of Table (down) or a fixed computation using all of the dimensions would accomplish this. Here is a view that illustrates a dynamic title and all three options in the caption:

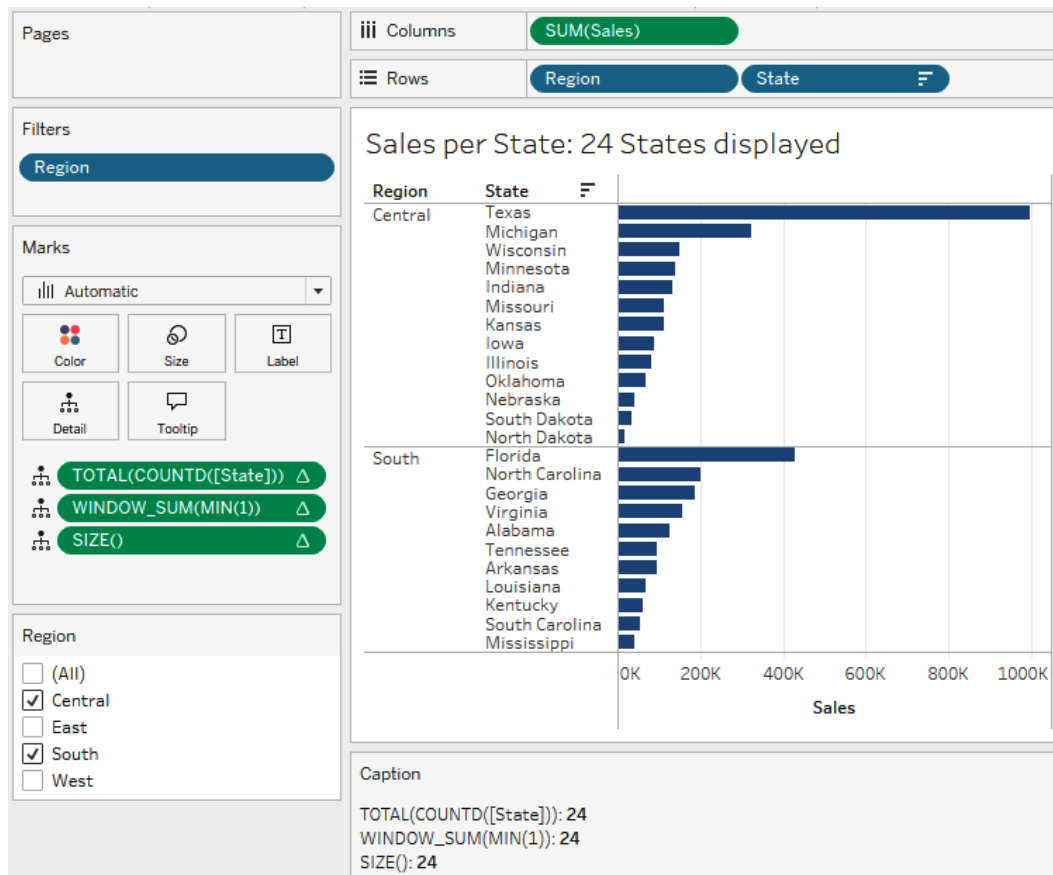


Figure 6.30: Various table calculations could be employed to achieve the total in the title

This example illustrated how you might use various table calculations to work at higher levels of detail, specifically counting all the states in the view. This technique will enable you to solve various analytical questions as you use Tableau. Let's now turn our attention to another technique that helps solve quite a few problems.

Table calculation filtering (late filtering)

Let's say you've built a view that allows you to see the percentage of total sales for each department. You have already used a quick table calculation on the **Sales** field to give you the percent of the total. You've also used **Department** as a filter. But this presents a problem.

Since table calculations are performed after the aggregate data is returned to the cache, the filter on **Department** has already been evaluated at the data source and the aggregate rows don't include any departments excluded by the filter. Thus, the percent of the total will always add up to 100%; that is, it is the percentage of the filtered total, as shown in the following screenshot:

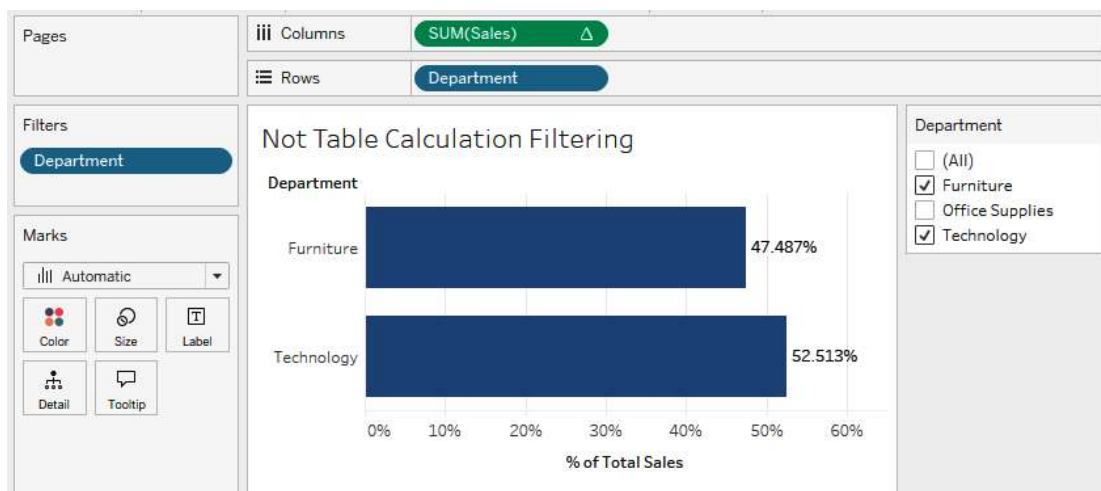


Figure 6.31: When Office Supplies is filtered out, the percentage table calculation adds up to 100% for the departments remaining in the view

What if you wanted to see the percentage of total sales for all departments, even if you want to exclude some from the display? One option is to use a table calculation as a filter.

You might create a calculated field called `Department (table calc filter)` with the code `LOOKUP(ATTR([Department]), 0)`. The `Lookup()` function makes this a table calculation, while `ATTR()` treats **Department** as an aggregation. The second argument, `0`, tells the lookup function not to look backward or forward. Thus, the calculation returns all values for **Department**, but as a table calculation result.

When you place that table calculation on the **Filters** shelf instead of the **Department** dimension, then the filter is not applied at the source. Instead, all the aggregate data is still stored in the cache and the table calculation filter merely *hides* it from the view. Other table calculations, such as **Percent of Total**, will still operate on all the data in the cache.

In this case, that allows **Percent of Total** to be calculated for all departments, even though the table calculation filter is hiding one or more, as shown in the following screenshot:

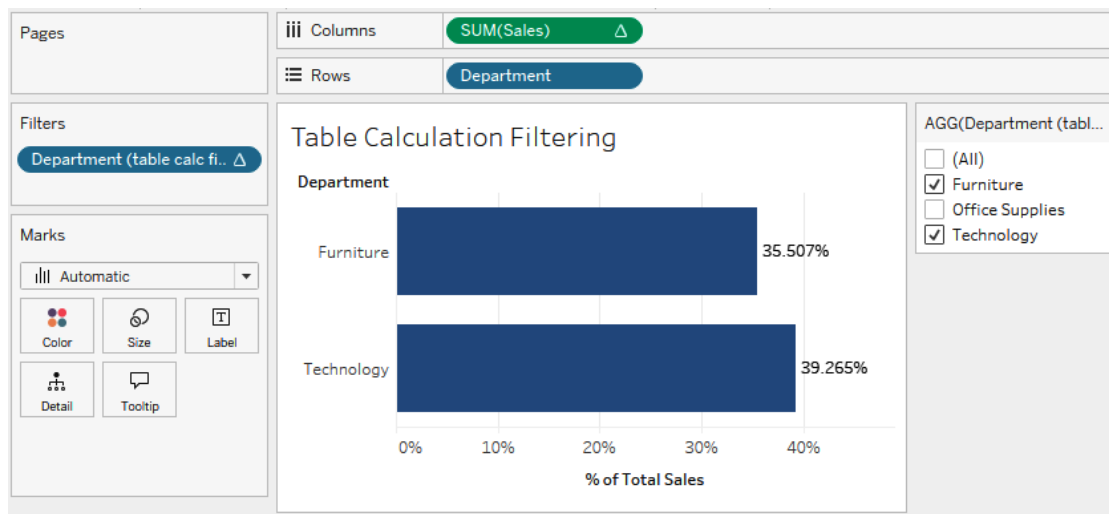


Figure 6.32: When a table calculation filter is used, all the aggregate data is available in the cache for the % of Total Sales to be calculated for all departments

You might have noticed the ATTR function used. Remember that table calculations require aggregate arguments. ATTR (which is short for attribute) is a special aggregation that returns the value of a field if there is only a single value of that field present for a given level of detail or a * if there is more than one value.

To understand this, experiment with a view having both **Department** and **Category** on rows. Using the drop-down menu on the active field in the view, change **Category** to **Attribute**. It will display as * because there is more than one category for each department. Then, undo and change **Department** to **Attribute**. This will display the department name because there is only one department per category.

In this example, we've seen how to effectively use table calculations as filters when we need other table calculations to operate on all the data in the cache.

Summary

We've covered a lot of concepts concerning table calculations in this chapter. You now have a foundation for using the simplicity of quick table calculations and leveraging the power of advanced table calculations. We've looked at the concepts of scope and direction as they apply to table calculations that operate relative to the row and column layout of the view. We've also considered the related concepts of addressing and partitioning as they relate to table calculations that have computations fixed to certain dimensions.

The practical examples we've covered barely scratch the surface of what is possible but should give you an idea of what can be achieved. The kinds of problems that can be solved and the diversity of questions that can be answered are almost limitless.

We'll turn our attention to some lighter topics in the next couple of chapters, such as formatting and design, but we'll certainly see another table calculation or two before we're finished!

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search for this book by name.



Note: Keep your purchase invoice ready before you start.

7

Making Visualizations that Look Great and Work Well

Tableau applies many good visual practices by default. For data discovery and initial analysis, you likely won't worry too much about changing many of these defaults. However, as you consider how to best communicate the data story you've uncovered, you'll want to consider how to leverage everything, from fonts and text to colors and design, so that you can communicate well with your audience.

Tableau's formatting options give you quite a bit of flexibility. Fonts, titles, captions, colors, row and column banding, labels, shading, annotations, and much more can all be customized to make your visualizations tell an impressive story.

This chapter will cover the following topics:

- Visualization considerations
- Leveraging formatting in Tableau
- Adding value to visualizations

As you think about why you should adjust a given visualization, there are several things to consider. We'll start with those considerations.

Visualization considerations

Tableau employs good practices for formatting and visualization from the time you start dropping fields onto shelves. You'll find that the discrete palettes use colors that are easy to distinguish, the fonts are pleasant, the grid lines are faint where appropriate, and the numbers and dates follow the default format settings defined in the metadata.

The default formatting is more than adequate for discovery and analysis. If your focus is analysis, you may not want to spend too much time fine-tuning the formatting until you are getting ready to share results. However, when you contemplate how you will communicate the data to others, you might consider how adjustments to the formatting can make a major difference to how well the data story is told.

Sometimes, you will have certain formatting preferences in mind or a set of corporate guidelines that dictate font and color selections. In these cases, you might set formatting options in a blank workbook and save it as a template.

Here are some of the things you should consider:

- **Audience:** Who is the audience and what are their needs?
- **Goal:** Is the goal to evoke an emotional response or to present facts for a logical decision? Are you highlighting an action that needs to be taken, or simply trying to generate interest in a topic?
- **Setting:** Consider the environment in which the data story will be communicated. Is it a formal business meeting where the format should reflect a high level of professionalism? Is it going to be shared informally on a blog?
- **Mode:** How will the visualizations be presented? Depending on the mode, you'll want to ensure that rows, columns, fonts, and marks are large enough for a projector or compact enough for an iPad. If you are publishing to Tableau Server, Tableau Cloud, or Tableau Public, then did you select fonts that are safe for the web? Will you need to use the device designer to create different versions of a dashboard?
- **Mood:** Certain colors, fonts, and layouts can set a mood. Does the data tell a story that should invoke a certain response from your audience? Is the data story somber or playful? The color red, for example, may connote danger or negative results or indicate that an action is required. However, you'll need to be sensitive to your audience and the specific context. Colors have different meanings in different cultures and different contexts. In some cultures, red might indicate joy or happiness. Also, red might not be a good choice to communicate negativity if it is the color of the corporate logo.
- **Consistency:** Generally, use the same fonts, colors, shapes, line thickness, and row banding throughout all visualizations. This is especially true when they will be seen together on a dashboard or even used in the same workbook. You may also consider how to remain consistent throughout the organization without being too rigid.

These considerations will inform your design and formatting decisions. As with everything else you do with Tableau, think of design as an iterative process. Seek feedback from your intended audience often and adjust your practices as necessary to make sure your communication is as clear and effective as possible. The entire goal of formatting is to more effectively communicate the data.

Leveraging formatting in Tableau

In this chapter, we will focus on worksheet-level formatting, as we've already covered metadata (data source-level formatting) in *Chapter 2, Connecting to Data and Tableau*, and we will cover dashboard and story formatting in *Chapter 8, Telling a Data Story with Dashboards*. However, it is beneficial to see the big picture of formatting in Tableau.

Tableau employs default formatting that includes default fonts, colors, shading, and alignment. Additionally, there are several levels of formatting you can customize, as shown in the following table:

Format Level	Description and How to Adjust	Example
Locale	Based on the locale of the system (or a manually adjusted locale) and sets the default format used for currency and dates. You can set the locale for a workbook from the menu. Select File Workbook Locale.	Setting the locale to English (United Kingdom) causes December 21 to be rendered as 21/12/2025 while English (United States) will render the date as 12/21/2025.
Data Source	The default formatting for a field, including number and date formats, colors, and shapes. We’ve already seen how you can set default formats for numeric and date fields. Other defaults, such as colors and shapes, can also be set using the Default Properties menu on the drop-down menu in the Data pane.	Setting the default number format of the Sales field to Currency (Custom) .
Workbook	The default format applied at a workbook level. This includes the default font for worksheets, tooltips, and titles. It also allows you to specify default formats for lines. Various global formatting options may be set. From the menu, select Format Workbook.	Making the default format of trend lines a light gray dotted line.
Story	Formatting applied to a story such as the story shading, title, navigator, and text objects. Navigate to a story tab and select Format Story (or Story Format) to edit formatting for story-specific elements. These include options for customizing the shading, title, navigator, and description.	Making the default shading for the navigator a light blue.
Dashboard	Formatting applied to a dashboard such as the overall shading, dashboard title, worksheet titles, and text objects. Dashboard-specific elements can be formatted. When viewing a dashboard, select Format Dashboard (or Dashboard Format).	Setting the default font of worksheet titles to Tableau bold, italic, and dark green.

Format Level	Description and How to Adjust	Example
Worksheet	All the various formatting options for a visualization, including fonts, alignment, shading, lines, banding, and field formatting. Worksheet formatting covers a wide range of possibilities and there are numerous interface options. We'll cover these in depth, but in general, select an option from the top Format menu or right-click an element you wish to format in a visualization and select Format.	Setting the alignment for a field to be right-aligned in the pane.

- In addition to the levels of formatting mentioned above, we'll see quite a few places that allow for **rich-text formatting**. Elements such as titles, captions, annotations, labels, and tooltips all contain text that can be formatted with varying fonts, colors, and alignment. This formatting is specific to the individual text element.
- We'll also consider some specifics related to **field-level formatting**. In addition to setting defaults in the metadata, we can define the way a field is formatted within a specific worksheet.

We'll start our investigation by examining workbook-level formatting.

Workbook-level formatting

Tableau allows you to set certain formatting defaults at a workbook level. To view the options and make changes to the defaults, click **Format | Workbook....** The left pane will now show formatting options for the workbook:

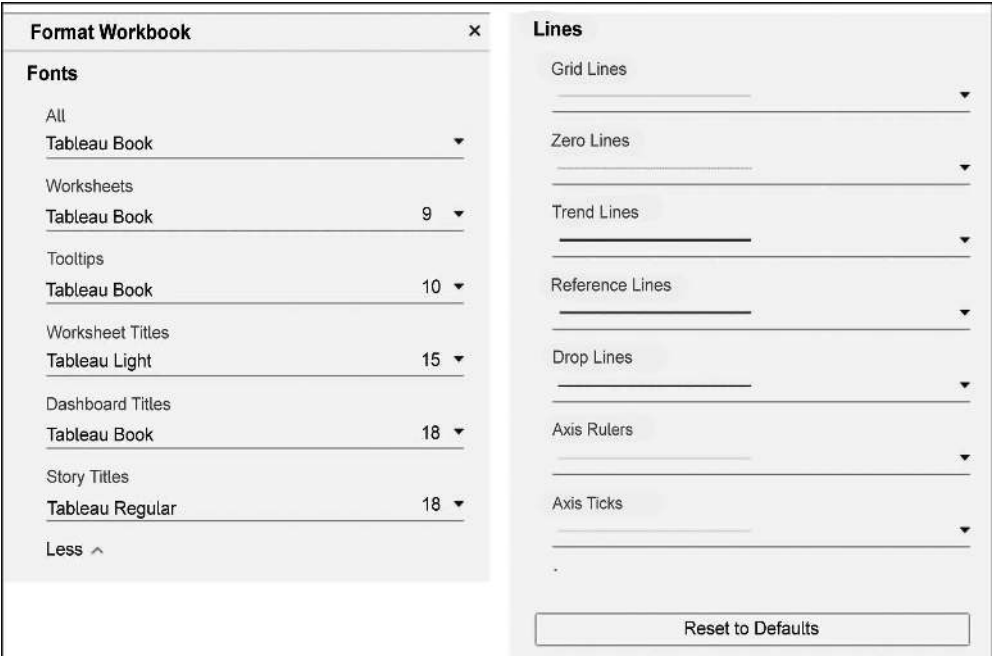


Figure 7.1: Workbook formatting options



Don't worry about making any formatting changes yet. We'll work through some practical examples later in the chapter. For now, just note that you have a wide range of options to customize the defaults for your entire workbook.

The options include the ability to change default **Fonts**, which apply to various parts of a view or dashboard, and default **Lines**, which apply to the various types of lines used in visualizations. Notice also the **Reset to Defaults** button, should you wish to revert to the default formatting. Any changes here will impact the whole workbook but may be overridden at other levels of formatting. For example, you might set the workbook font to be size 10 Arial but override that with size 12 Times New Roman on a specific worksheet.

Workbook formatting sets defaults that will apply broadly across all sheets in your workbook: worksheets, dashboards, and stories. As we continue, we'll see how you can apply formatting specifically to a specific sheet. We'll examine the broad range of worksheet-level formatting options next.

Worksheet-level formatting

You've already seen how to edit metadata in previous chapters, and we'll cover dashboards and stories in detail in future chapters. So, we'll shift our attention to worksheet-level formatting now.

Before we dive into the details of how to adjust settings, consider the following parts of a view related to formatting:

Formatting: Parts of the View

		Customer Segment				
1 Department	Category	4 Consumer	Corporate	Home Office	Small Business	6 Grand Total
3 Furniture	Bookcases	5 92,626	262,085	79,404	73,381	507,496
	Chairs & Chairmats	305,381	407,724	212,830	238,651	1,164,586
	Office Furnishings	69,528	115,506	197,188	62,412	444,634
	Tables	228,934	363,979	287,507	181,502	1,061,922
	a Total	a 696,469	1,149,294	776,929	555,946	3,178,638
Office Supplies	Appliances	63,813	167,941	124,757	100,225	456,736
	Binders and Binder Accessor...	103,625	225,160	148,472	161,326	638,583
	Envelopes	37,643	44,462	22,577	43,233	147,915
	Labels	3,713	7,929	5,411	6,393	23,446
	Paper	53,004	89,312	61,123	50,181	253,620
	Pens & Art Supplies	24,027	36,004	21,765	21,469	103,265
	Rubber Bands	1,710	2,197	2,294	2,469	8,670
	Scissors, Rulers and Trimme...	14,628	9,625	12,947	3,232	40,432
	Storage & Organization	121,719	154,918	179,151	129,929	585,717
	Total	423,882	737,548	578,497	518,457	2,258,384
Technology	Computer Peripherals	80,805	224,142	110,840	75,064	490,851
	Copiers and Fax	148,504	205,639	174,718	132,354	661,215
	Office Machines	260,011	516,513	245,019	197,112	1,218,655
	Telephones and Communicat...	225,571	436,295	282,962	199,456	1,144,284
b Grand Total		714,891	1,382,589	813,539	603,986	3,515,005
		1,835,242	3,269,431	2,168,965	1,678,389	8,952,027

Figure 7.2: Parts of a view that can be formatted using worksheet-level formatting

This view consists of the following parts, which can be formatted:

1. **Field labels for rows:** Field labels can be formatted from the menu (**Format | Field Labels...**) or by right-clicking them in the view and selecting **Format...**. Additionally, you can hide field labels from the menu (**Analysis | Table Layout** and then uncheck the option for showing field labels) or by right-clicking them in the view and selecting the option to hide. You can use the **Analysis | Table Layout** option on the top menu to show them again if needed.
2. **Field labels for columns:** These have the same options as labels for rows, but they may be formatted or shown/hidden independently from the row-field labels.
3. **Row headers:** These will follow the formatting of headers in general unless you specify different formatting for headers for rows only. Notice that subtotals and grand totals have headers. The subtotal and grand total headers are marked **a** and **b**, respectively.
4. **Column headers:** These will follow the formatting of headers in general unless you specify different formatting for headers for the columns only. Notice that subtotals and grand totals have headers. The grand total header marked in the preceding screenshot is a column header.
5. **Pane:** Many formatting options include the ability to format the pane differently than the headers.
6. **Grand Total (column) pane:** This is the pane for grand totals that can be formatted at a sheet or column level.
7. **Grand Total (row) pane:** This is the pane for grand totals that can be formatted at a sheet or row level.

Worksheet-level formatting is accomplished using the format window, which will appear on the left side, in place of the Data pane.

To view the format window, select **Format** from the menu and then **Font...**, **Alignment...**, **Shading...**, **Borders...**, or **Lines...**:

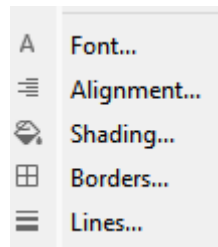


Figure 7.3: Formatting options for a worksheet

You can also right-click nearly any element in the view and select **Format**. This will open the format window specific to the context of the element you selected. Just be sure to verify that the title of the format window matches what you expect.

When you make a change, you should see the view update immediately to reflect your formatting. If you don't, you are likely working in the wrong tab of the formatting window, or you may have formatted something at a lower level (for example, **Rows**) that overrides changes made at a higher level (for example, **Sheet**).

You should now see the formatting window on the left, in this case, **Format Font**. It will look like this:

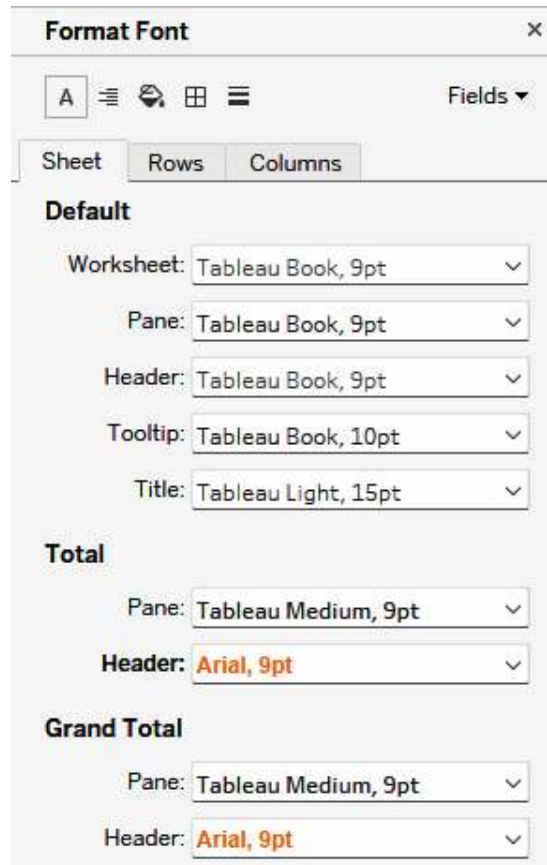


Figure 7.4: The Format Font pane

Notice these key aspects of the formatting window:

- The title (**Format Font**) of the window will give you the context for your formatting selections.
- The icons at the top match the selection options of the **Format** menu. This allows you to easily navigate through those options without returning to the menu each time.
- The three tabs, **Sheet**, **Rows**, and **Columns**, allow you to specify options at a sheet level and then override those options and defaults at a row and column level. For example, you could make the **Row** grand totals have different pane and header fonts from the **Column** grand totals (although this specific choice would likely be jarring and is not recommended!).
- The **Fields** dropdown in the upper-right corner allows you to fine-tune formatting at the field level.
- Any changes that you make will be previewed and result in a bold label to indicate that the formatting option has been changed from the default (notice how the font for **Header** under **Total** has been customized, resulting in the label text of **Header** being shown in bold).

The three options for clearing the format are as follows:

- **Clear Single Option:** In the format window, right-click the label or control of any single option you have changed and select **Clear** from the pop-up menu.
- **Clear All Current Options:** At the bottom of the format window, click the **Clear** button to clear all visible changes. This applies only to what you are currently seeing in the format window. For example, if you are viewing **Shading** on the **Rows** tab and click **Clear**, only the shading options on the **Rows** tab will be cleared.
- **Clear Sheet:** From the menu, select **Worksheet | Clear | Formatting**. You can also use the dropdown from the **Clear** item on the toolbar. This clears all custom formatting on the current worksheet.

The other format options (such as alignment and shading) all work very similarly to the Font option. There are only a few subtleties to mention:

- **Alignment** includes options for horizontal and vertical alignment, text direction, and text wrapping.
- **Shading** includes an option for row and column banding. The banding allows alternating patterns of shading that help to differentiate or group rows and columns. Light row banding is enabled by default for text tables, but it can be useful in other visualization types, such as horizontal bar charts as well. Row banding can be set to different levels that correspond to the number of discrete (blue) fields present on the **Rows** or **Columns** shelves.
- **Borders** refers to the borders drawn around cells, panes, and headers. It includes an option for row and column dividers. You can see in the view the dividers between the departments. By default, the level of the borders is set based on the next-to-last field in the rows or columns.
- **Lines** refers to lines that are drawn on visualizations using an axis. This includes grid lines, reference lines, zero lines, and axis rulers. You can access a more complete set of options for reference lines and drop lines from the **Format** option of the menu.

We have considered how to adjust formatting at the entire workbook level as well as for a given sheet. Keep these options in mind (we'll see some practical examples of how to apply this to your visualizations at the end of the chapter). For now, let's turn our attention to formatting at the field level.

Field-level formatting

In the upper-right corner of the format window is a little drop-down menu labeled **Fields**. Selecting this drop-down menu gives you a list of fields in the current view. Selecting a field from this list updates the format window with options appropriate for the field. Here, for example, is the window as it appears for the **SUM(Sales)** field:

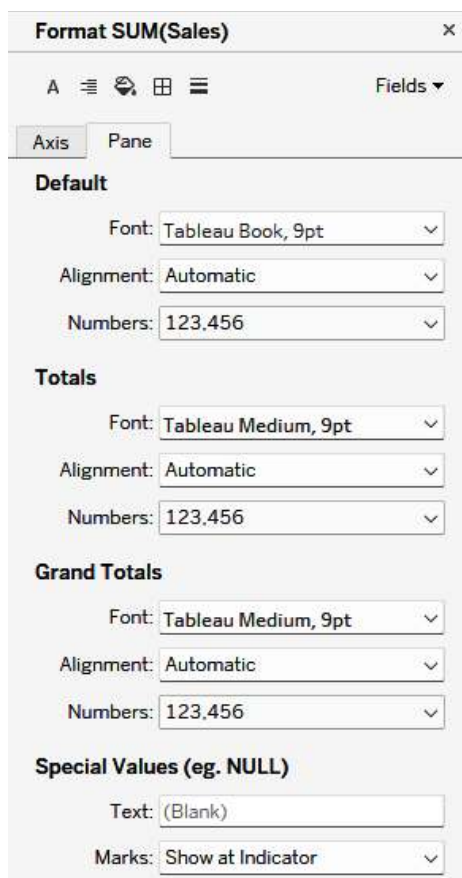


Figure 7.5: The Format pane for field-level formatting

The title of the format window (**Format SUM(Sales)**, in this example) alerts you to the field you are formatting. Selecting an icon for **Font**, **Alignment**, and so on from the top-left corner of the window will switch back to sheet-level formatting. However, you can switch between the tabs of **Axis** and **Pane**. These two tabs allow you to have different formatting for a field when it is used in the header or as an axis label versus how it is formatted in the pane of the view. The options for fields include **Font**, **Alignment**, **Shading**, and **Number** and **Date** formats. The last two options will override any default metadata formats.

You'll notice special options for formatting certain types of fields. Numbers allow a wide range of formatting options and even include the ability to provide custom formatting, which we'll consider next.

Custom number formatting

When you alter the format of a number, you can select from several standard formats, as well as a custom format. The custom format allows you to enter a format string that Tableau will use to format the number. Your format string may use combinations of the number sign (#), commas, negative signs, and parentheses, along with a literal string enclosed in quotation marks to indicate how the number should be displayed.

The format string allows up to three entries, separated by semicolons to represent positive, negative, and zero formats.

Here are some examples, assuming the positive number is 34,331.336 and the negative number is -8,156.7777:

Format String	Resulting Value
#; -#	34331 and -8157
#,###.##; (#,###.##)	34,331.34 and (8,156.78)
#,###.00000;-#,###.00000	34,331.33600 and -8,156.77770
"up " #,###;"down ",###;"same"	up 34,331 and down 8,157
#,###" ▲ "; #,###" ▼ "	34,331▲ and 8,157▼

Figure 7.6: Examples of format strings and resulting values

You can replicate these examples and experiment with other format strings using the **Custom Number Formatting** view in the Starter or Complete workbooks. For example, consider the formatting settings shown here:

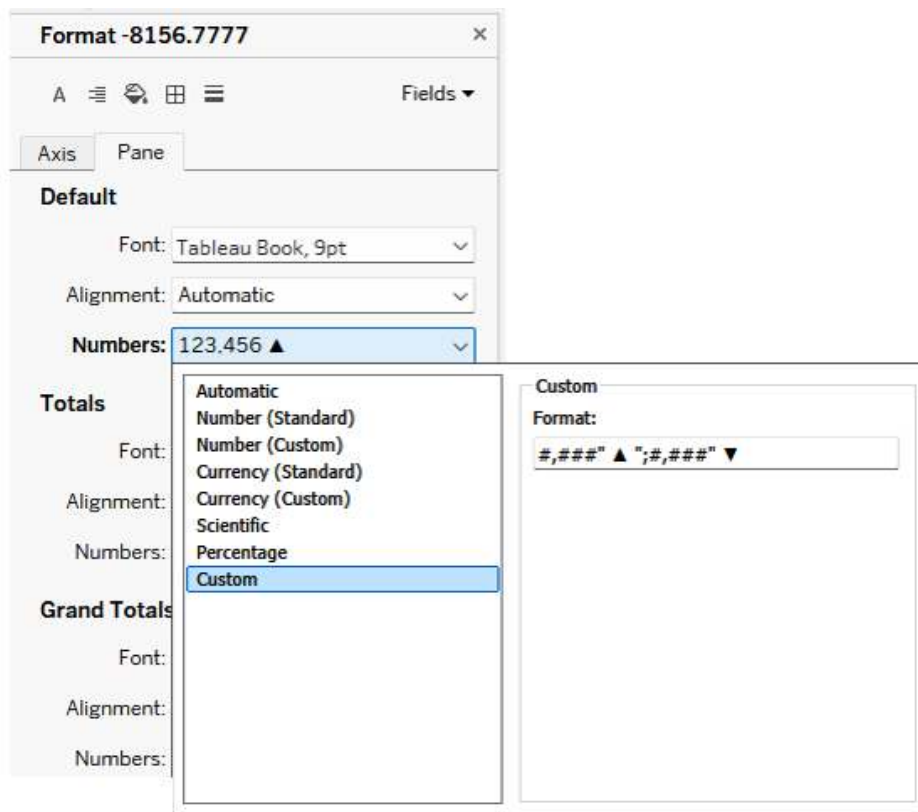


Figure 7.7: Setting a custom number string that shows an up or down indication corresponding to positive or negative values

The settings in *Figure 7.7* result in a view that looks like this:

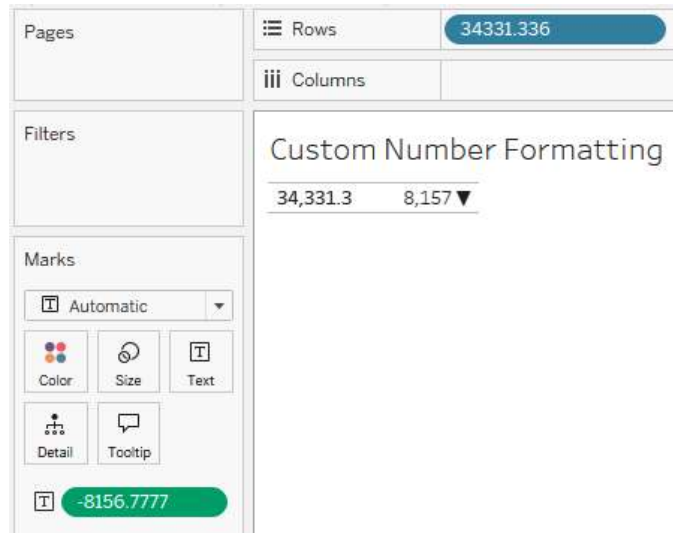


Figure 7.8: The results of the custom format string

There are several ways to insert Unicode characters and emojis:

- Search online and copy/paste.
- Use **Charmap** (Character Map) or an equivalent to find, copy, and paste Unicode characters.
- Hold down the *Alt* key and type + and the numeric value of the Unicode character on your number pad (Windows).
- Enable Unicode input and hold down on *option* while typing the numeric value of the Unicode character (macOS).

Notice how Tableau rounds the display of the number based on the format string. Always be aware that numbers you see as text, labels, or headers may have been rounded due to the formatting. For example, if you specify the format `#,###`, then a value such as `1500.03` would be shown rounded as `1,500`, but if you used the format string `#,###.##`, then you would see the precise value, `1,500.03`.

Also observe how you can mix format characters such as the number sign, commas, and decimal points with strings. The fourth example shown would provide a label where a value of zero would normally be displayed.

Finally, notice that the format string in *Figure 7.7* and the results in *Figure 7.8* makes use of a Unicode character. This gives a wide range of possibilities, such as displaying degrees or other units of measure. You may even use emojis! Unicode characters and emojis may be used throughout Tableau in textboxes, titles, field names and labels, aliases, and more.

Selecting a predefined format that is close to what you want and then switching to custom will allow you to start with a custom format string that is close to your desired format.

Let's turn our attention to a similar ability to apply custom format strings to dates.

Custom date formatting

The following table illustrates some possible formatting of the date value of 11/08/2018, 1:30 PM based on various format strings:

Format String	Resulting Value
m/d/yyyy	11/8/2018
dd/mm/yyyy	08/11/2018
"The date is" m/d/yyyy	The date is 11/8/2018
mmm d, yyyy	Nov 8, 2018
mmmm dd, yyyy	November 08 2018
mm/dd/yyyy h:mm AM/PM	11/08/2018 1:30PM
ttttt	1:30:28 PM
dddd, mmmm d, HH:MM:SS	Thursday, November 8, 13:30:28
ddd	Thu

Figure 7.9: Some possible date formatting examples

These are merely examples, and you may include as many literal strings as you'd like.



For a complete list of custom date format string options, check out https://onlinehelp.tableau.com/current/pro/desktop/en-us/dates_custom_date_formats.htm.

Notice how applying some custom date formatting improves the readability of the axis for a small timeline in this example:

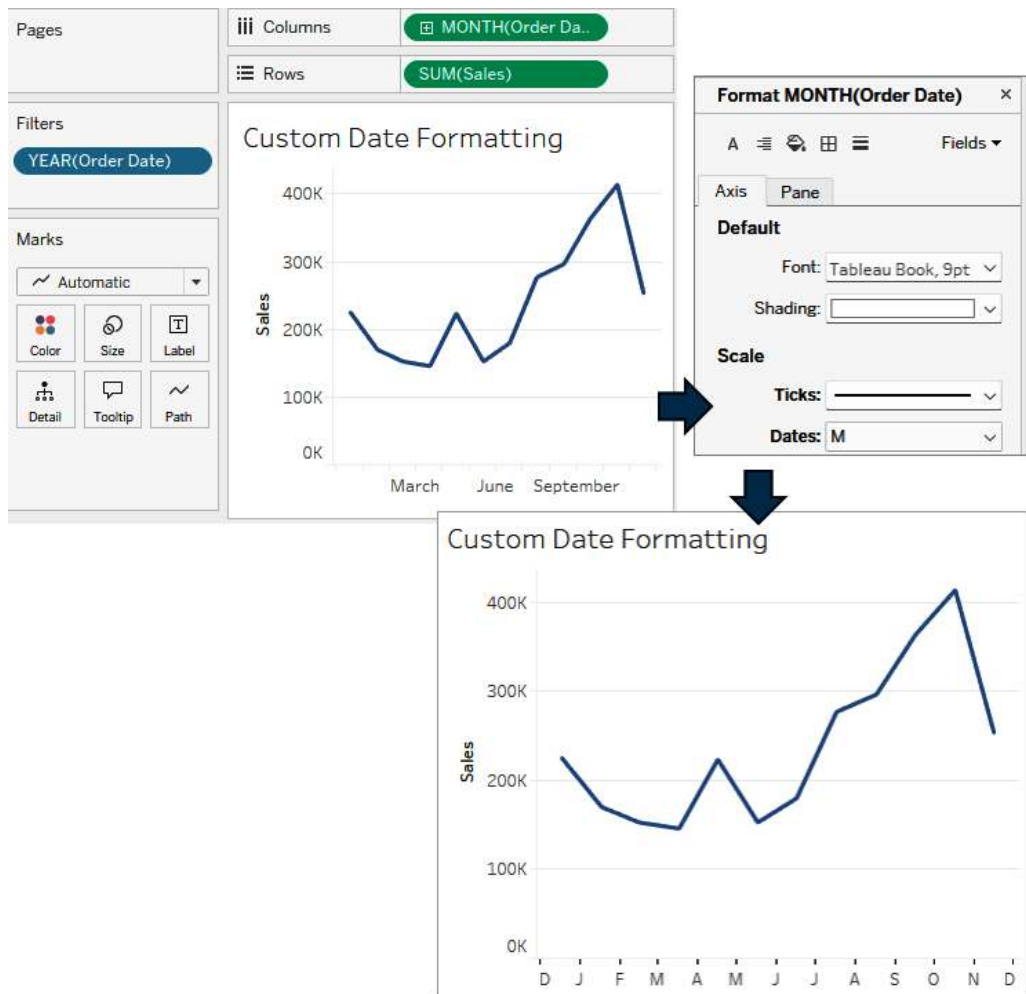


Figure 7.10: The custom format string used here is *M*, which results in a single letter for the month

Custom number and date formats are fine when you have values that need to be formatted. But what if there is no value? How can we format NULL values? Let's consider that next.

Null formatting

Another aspect of formatting a field is specially formatting **Null** values. When formatting a field, select the **Pane** tab and locate the **Special Values** section, as shown in the following screenshot:

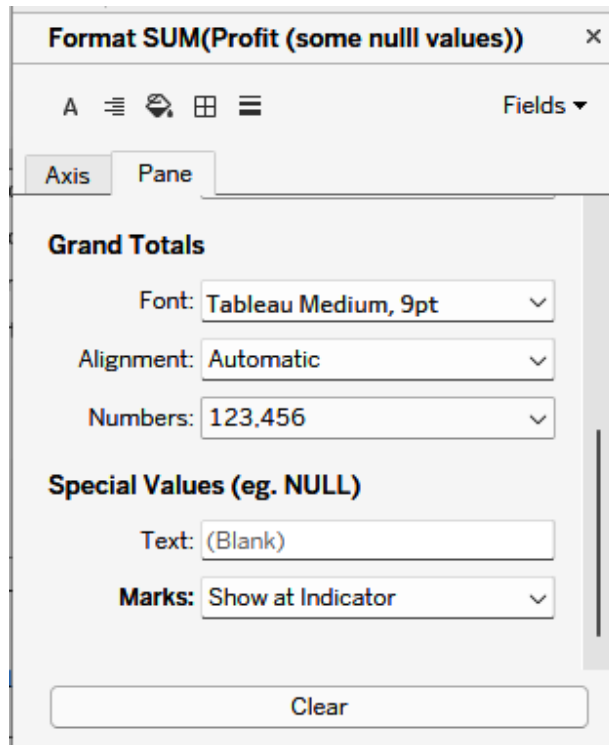


Figure 7.11: The Special Values options appear on the Format pane

Enter any text you would like to display in the pane (in the **Text** field) when the value of the field is null. You can also choose where marks should be displayed. The **Marks** drop-down menu gives multiple options that define where and how the marks for null values should be drawn when an axis is being used. You have the following options:

- **Show at Indicator** results in a small indicator with the number of null values in the lower right of the view. You can click the indicator for options to filter the null values or show them at the default value. You can right-click the indicator to hide it.
- **Show at Default Value** displays a mark at the default location (usually 0).
- **Hide (Connect Lines)** does not place a mark for null values but does connect lines between all non-null values.
- **Hide (Break Lines)** causes the line to break where there are gaps created by not showing the null values.

You can see these options in the following screenshots, with the location of two null values indicated by a gray band.

Show at Indicator reveals no marks in the gray band with the number of null values indicated in the lower-right corner:

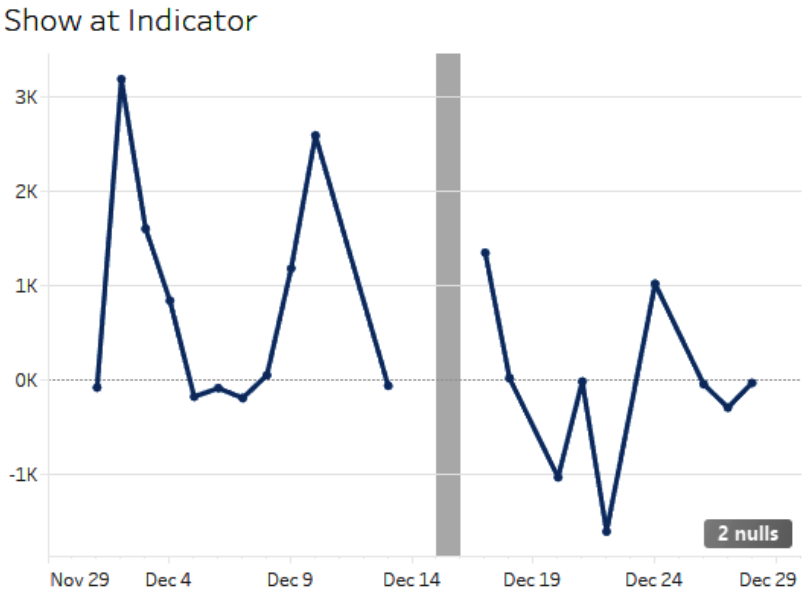


Figure 7.12: Show at Indicator

Show at Default Value places marks at 0 and connects the lines:

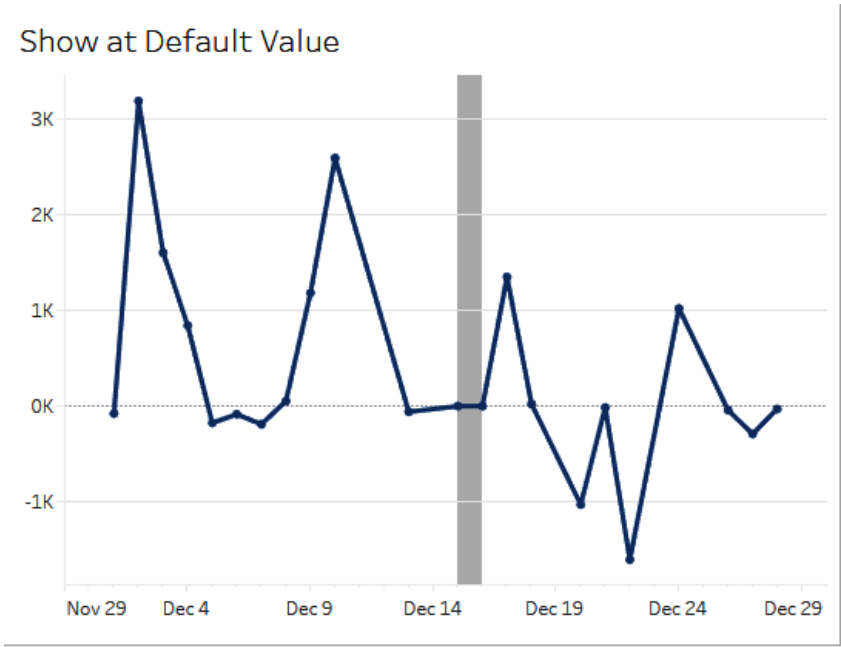


Figure 7.13: Show at Default Value

Hide (Connect Lines) removes marks for the missing values, but does connect the existing marks:

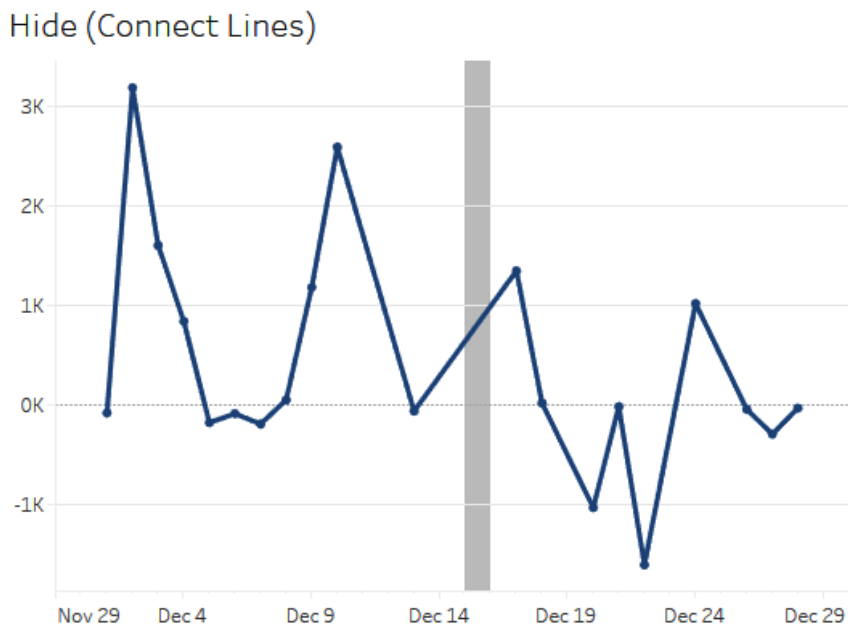


Figure 7.14: Hide (Connect Lines)

Hide (Break Lines) removes the marks for the missing values and does not connect the existing marks:

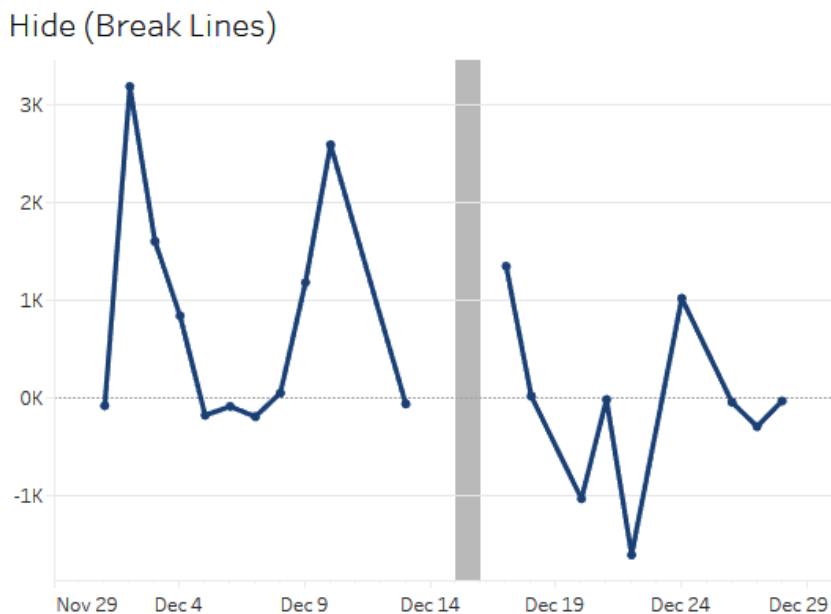


Figure 7.15: Hide (Break Lines)

Any of these options might have a good use, but consider how each communicates the information. Connecting the lines might help communicate a movement over time but may also minimize the existence of missing data. Breaking the lines might help highlight missing values but might take away from the overall message. You'll need to decide which option best meets your goals based on the considerations mentioned at the beginning of this chapter.

You'll notice that the preceding line charts have little circle markers at the location of each mark drawn in the view. When the mark type is a line, clicking on the **Color** shelf opens a menu that gives options for the markers. All mark types have standard options, such as color and transparency.

Some mark types support additional options such as border and/or halo, as shown here:

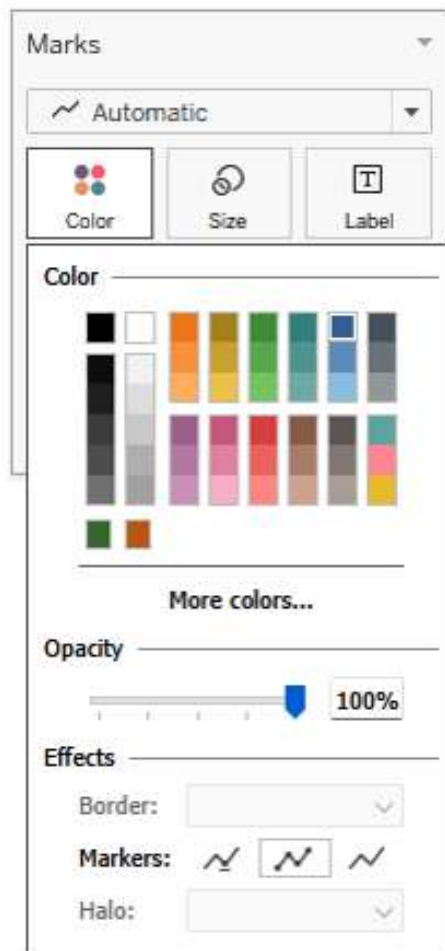


Figure 7.16: Adding Markers to lines

Knowing these options will help you as you think about how to communicate missing data, but always consider that another visualization type such as a bar chart might be even more effective in communicating missing values:

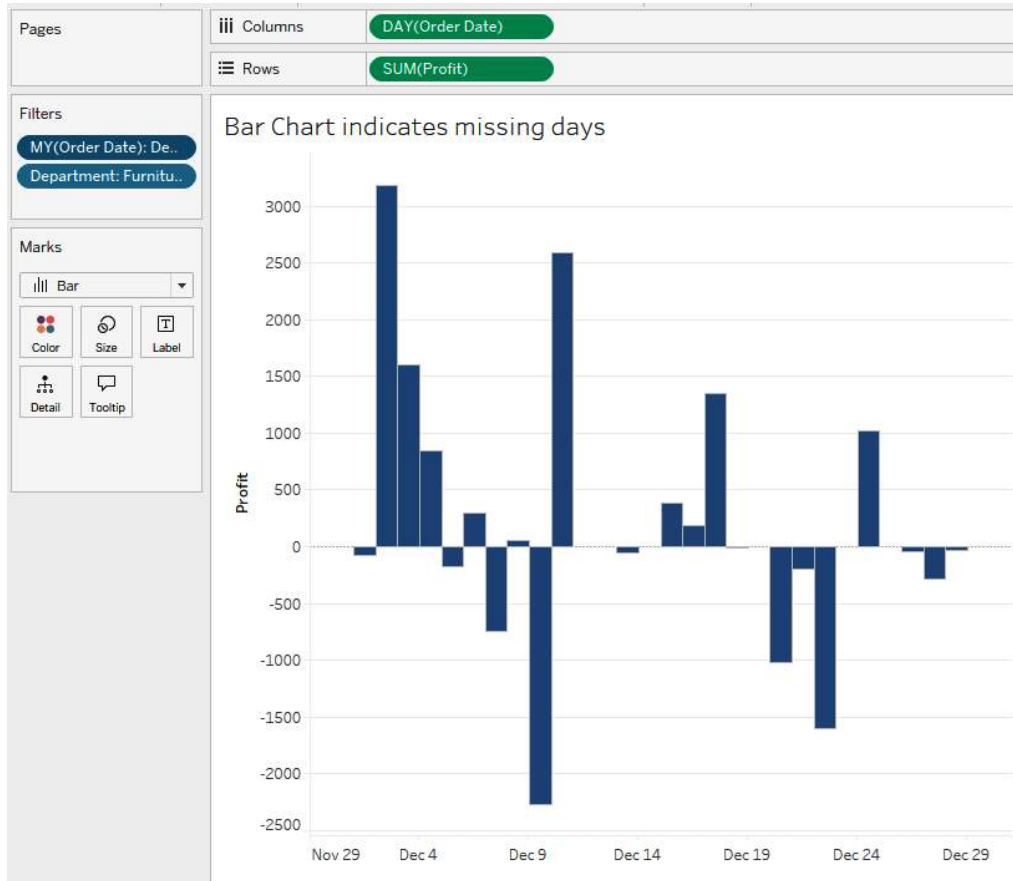


Figure 7.17: Bar charts are sometimes better than line charts for showing missing values

Knowing how to format null values gives you some options as you consider how to communicate the data. Let's take a look at a few additional options.

Additional formatting options

Additional formatting options can also be accessed from the formatting window. These options include the following:

- A myriad of options for **Reference Lines**
- Line and text options for **Drop Lines**
- Shading and border options for **Titles and Captions**
- Text, box, and line options for **Annotations**
- Font, shading, alignment, and separator options for **Field labels**

- Title and body options for **Legends**, **Quick Filters**, and **Parameters**
- Cell size and **Workbook** theme options

You'll find most of these relatively straightforward. A few options might not be as obvious:

- **Drop Lines**, which appear as lines drawn from the mark to the axis, can be enabled by right-clicking any blank area in the pane of the view with an axis and selecting **Drop Lines** | **Show Drop Lines**. Additional options can be accessed by using the same right-click menu and selecting **Edit Drop Lines**. Drop lines are only displayed in Tableau Desktop and Reader but are not currently available when a view is published to Tableau Server, Tableau Cloud, or Tableau Public.
- **Titles** and **Captions** can be shown or hidden for any view by selecting **Worksheet** from the menu and then selecting the desired options. In addition to standard formatting, which can be applied to titles and captions, the text of a title or caption can be edited and specifically formatted by double-clicking the title or caption, right-clicking the title or caption and selecting **Edit**, or by using the drop-down menu of the title or caption (or the drop-down menu of the view on a dashboard). The text of titles and captions can dynamically include the values of parameters, the values of any field in the view, and certain other data and worksheet-specific values.
- **Annotations** can be created by right-clicking a mark or space in the view, selecting **Annotate**, and then selecting one of the following three types of annotations:
 - **Mark** annotations are associated with a specific mark in the view. If that mark does not show (due to a filter or axis range), then neither will the annotation. Mark annotations can include a display of the values of any fields that define the mark or its location.
 - **Point** annotations are anchored to a specific point in the view. If the point is ever not visible in the view, the annotation will disappear. Point annotations can include a display of any field values that define the location of the point (for example, the coordinates of the axis).
 - **Area** annotations are contained within a rectangular area. The text of all annotations can dynamically include the values of parameters, and certain other data- and worksheet-specific values.
- You can copy formatting from one worksheet to another (within the same workbook or across workbooks) by selecting **Copy Formatting** from the **Format** menu while viewing the source worksheet (or selecting the **Copy Formatting** option from the right-click menu on the Source worksheet tab). Then, select **Paste Formatting** on the **Format** menu while viewing the target worksheet (or select the option from the right-click menu on the **Target** worksheet tab).

This option will apply any custom formatting present on the source sheet to the target. However, specific formatting applied during the editing of the text of titles, captions, labels, and tooltips is not copied to the target sheet.

We've now considered a lot of options for formatting the workbook, individual sheets, fields, numbers, dates, and null values. Now, let's consider how we can leverage some of these techniques to truly bring a better understanding of the data.

Adding value to visualizations

Now that we've considered how formatting works in Tableau, let's look at some ways in which formatting can add value to a visualization.

When you apply custom formatting, always ask yourself what the formatting adds to the understanding of the data. Is it making the visualization clearer and easier to understand? Or is it just adding clutter and noise?

In general, try a minimalistic approach. Remove everything from the visualization that isn't necessary. Emphasize important values, text, and marks, while de-emphasizing those that are only providing support or context.

Consider the following visualization, all using default formatting:

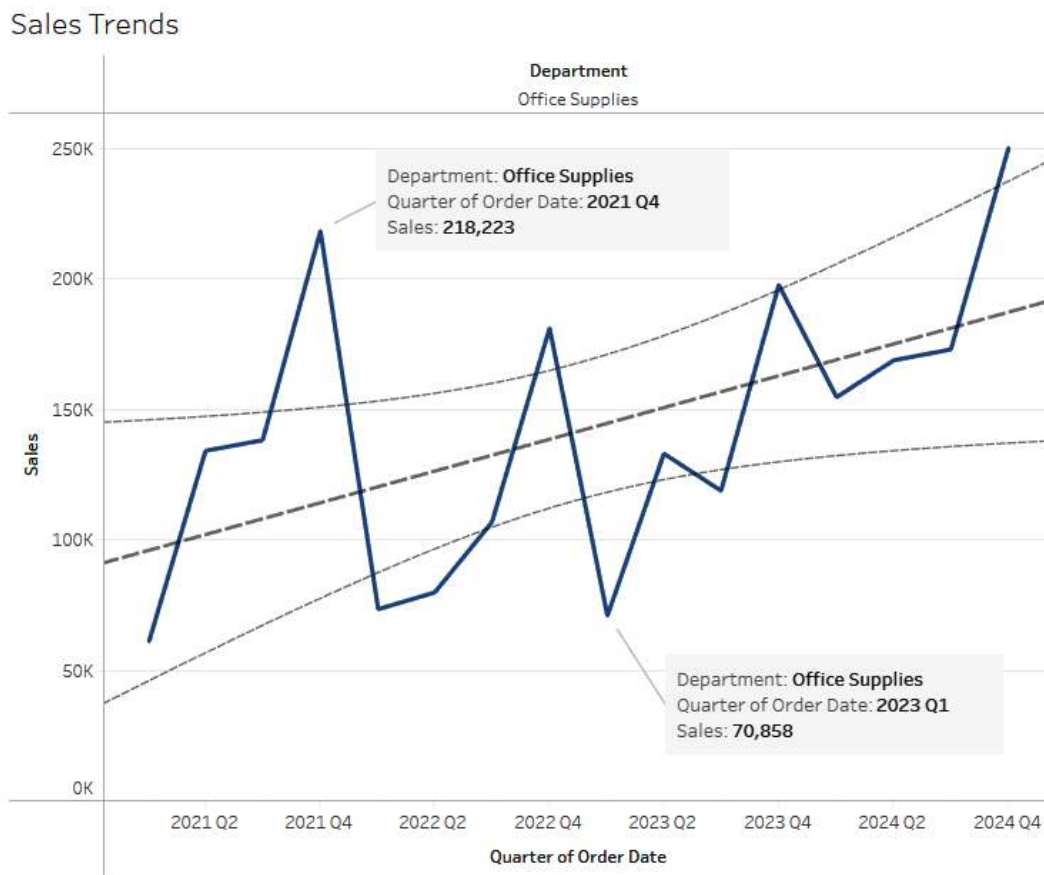


Figure 7.18: The default formatting is often great for data discovery and quick analysis but may be more cluttered than desired for clearly communicating and emphasizing the data story to others

The default format works fairly well, but compare that to this visualization:

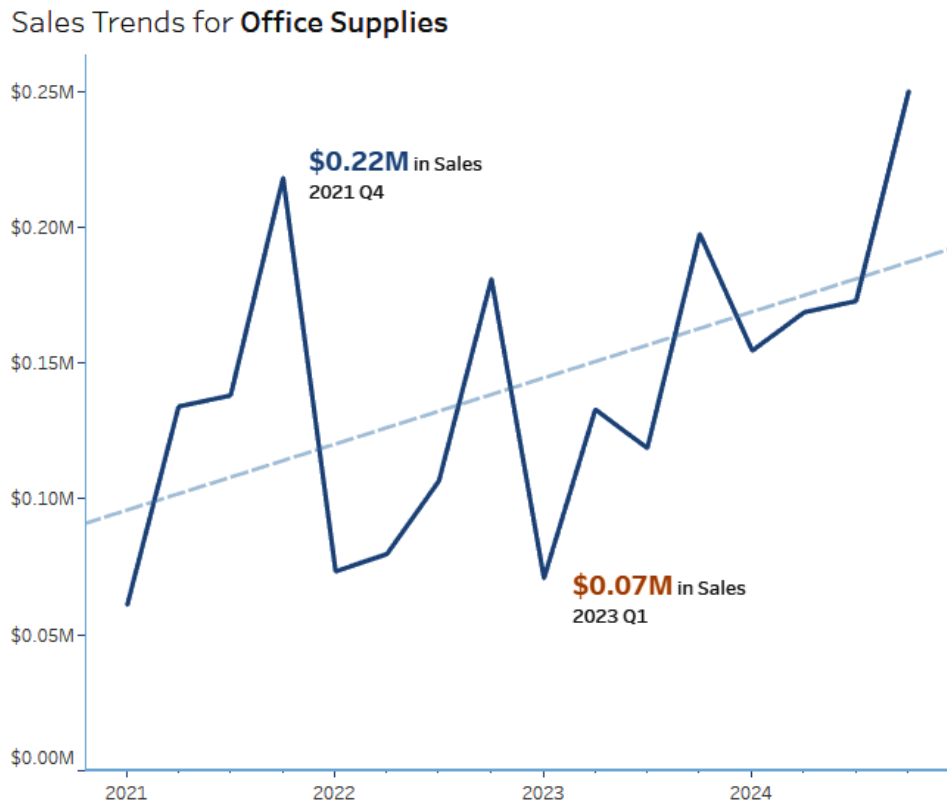


Figure 7.19: Formatting can make a visualization less cluttered and communicate data more effectively

Both of the preceding diagrams show sales by the quarter, filtered to the **Office Supplies** department. The first view uses the default formatting. The second view has some formatting adjustments, including the following:

- **Title** has been adjusted to include the department name.
- **Sales** has been formatted to be shown using a custom currency with two decimal places and units of millions. This is true for the axis and the annotations. Often, a high level of precision can clutter a visualization. The initial view of the data gives us the trend and enough detail to understand the order of magnitude. Tooltips or additional views can be used to reveal detail and increase precision.
- The axis labels have been removed by right-clicking the axis, selecting **Edit Axis**, and then clearing the text. The title of the view clearly indicates that you are looking at Sales. The values alone reveal the second axis to be by the quarter. If there are multiple dates in the data, you might need to specify which one is in use. Depending on your goals, you might consider hiding the axes completely.

- The gridlines on **Rows** have been removed. Gridlines can add value to a view, especially in views where being able to determine values is of high importance. However, they can also clutter and distract. You'll need to decide, based on the view itself and the story you are trying to tell, whether gridlines are helpful.
- The trend line has been formatted to match the color of the line, though it is lighter and thinner, to de-emphasize it. Additionally, the confidence bands have been removed. You'll have to decide whether they add context or clutter based on your needs and the audience.
- The lines, shading, and boxes have been removed from the annotations to reduce clutter.
- The size and color of the annotations have been altered to make them stand out. If the goal had been to simply highlight the minimum and maximum values on the line, labels might have been a better choice, as they can be set to display at only **Min/Max**. In this case, however, the lower number is actually the second-lowest point in the view.
- Axis rulers and ticks have been emphasized and colored to match the marks and reference line (axis rulers are available under the **Lines** option on the **Format** window).

Formatting can also be used to dramatically alter the appearance of a visualization. Consider the following chart:

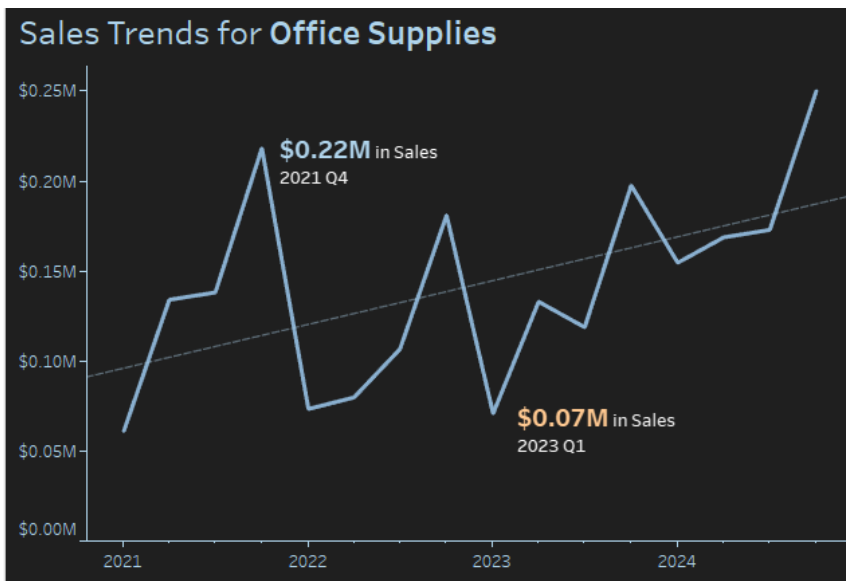


Figure 7.20: A dark background can be set by formatting the shading of a view

This visualization is nearly identical to the previous view. However, shading has been applied to the worksheet and the title. Additionally, fonts and mark colors were lightened as needed to show up well on a dark background. Some find this format more pleasing, especially on mobile devices. If the view is to be embedded into a website with a dark theme, this formatting may be very desirable. However, you may find some text more difficult to read on a dark background.

You'll want to consider your audience, the setting, and the mode of delivery as you consider whether such a format is the best for your situation.



Sequential color palettes (a single color gradient based on a continuous field) should be reversed when using a dark background. This is because the default of lighter (lower values) to darker (higher values) works well on a white background, where darker colors stand out and lighter colors fade into white. On a black background, lighter colors stand out more, and darker colors fade into black. You'll find the reverse option when you edit a color palette using the drop-down menu upon double-clicking the legend, or right-clicking the legend, selecting **Edit Colors...**, and checking **Reversed**.

Having seen the impact of formatting on the view, we'll now turn our attention to formatting for tooltips.

Tooltips

As they are not always visible, tooltips are an often overlooked aspect of visualizations. However, they add subtle professionalism. Consider the following default tooltip that displays when the end user hovers over one of the marks shown in the preceding screenshot:

Department:	Office Supplies
Quarter of Order Date:	2024 Q1
Department:	Office Supplies
Sales:	\$0.15M

Figure 7.21: Default tooltip



 **Quick tip:** Need to see a high-resolution version of this image? Open this book in the next-gen Packt Reader or view it in the PDF/ePub copy.

 **The next-gen Packt Reader and a free PDF/ePub copy** of this book are included with your purchase. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



Compare it to this tooltip:



Figure 7.22: Customized tooltip

The tooltip was edited by clicking **Tooltip** on the **Marks** card, which brought up an editor allowing rich text editing of the tooltip:

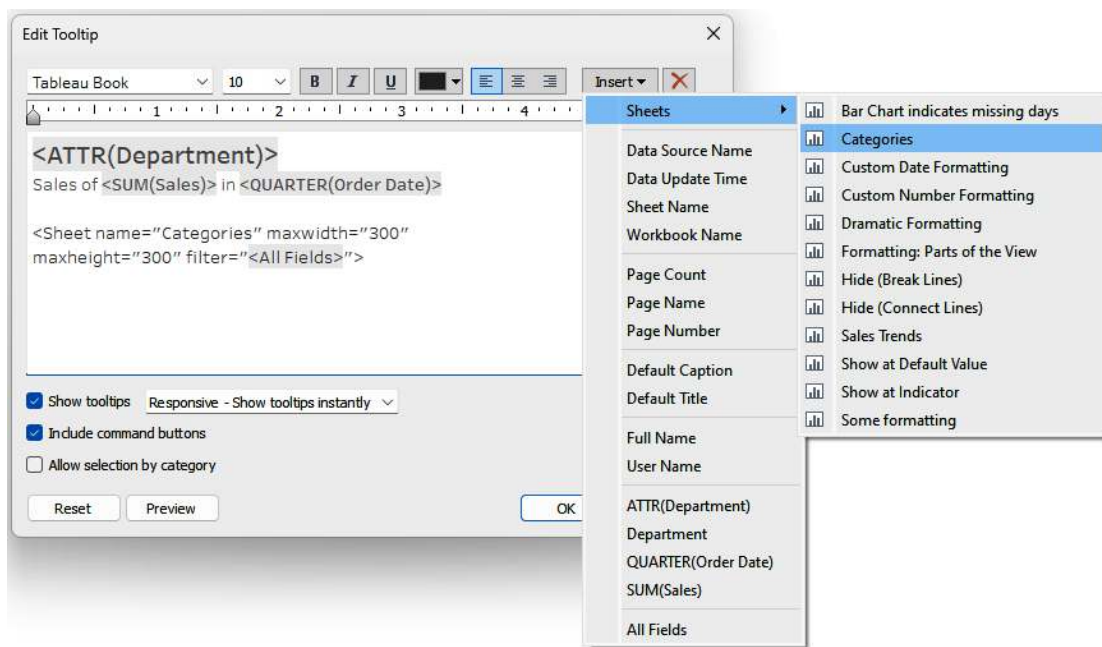


Figure 7.23: Tooltip editor

This editor is similar to those used for editing the text of labels, titles, captions, and annotations. You can input text and format it as desired. Additionally, the **Edit Tooltip** dialog has some additional functionality:

- The **Insert** drop-down menu in the upper-right corner allows you to insert sheets, fields, parameters, and other dynamic values. These special or dynamic text objects are enclosed as a tag in the text editor (for example, `<SUM(Sales)>`). We'll consider the special case of sheets in a moment.
- A **Show tooltips** checkbox option and a drop-down menu to indicate the style of the tooltip (**Responsive - Show tooltips instantly** or **On Hover**).
- An **Include command buttons** checkbox option. This is the default, and you can observe the command buttons in the first unedited tooltip in this section. The command buttons include options such as **Include**, **Exclude**, **Create Sets**, and so on. Many of these options are still available to the end user via a right-click, so removing them from the tooltip does not prevent the user from accessing them completely.
- An **Allow selection by category** checkbox option. When enabled, this feature allows users to click the values of dimensions shown in the tooltip and thus select all marks associated with that value in the view.

Consider unchecking **Show tooltips** for any view where they do not significantly and intentionally add value to the user experience.

Viz in Tooltip

Tableau allows you to embed visualizations in tooltips that are dynamically filtered as you hover over different marks. Often referred to as **Viz in Tooltip**, this greatly extends the interactivity available to end users, the ability to drill down to the details, and the ability to quickly see data in different ways.

In the preceding screenshot, the following tag was added to the tooltip by selecting **Insert | Sheets | Categories**:

```
<Sheet name="Categories" maxwidth="300" maxheight="300" filter="<All Fields>">
```

This tag, which you may edit by directly editing the text, tells Tableau to show the visualization in the **Categories** sheet as part of the tooltip. The maximum width and height are set to 300 pixels by default. The filter indicates which field(s) act(s) as a filter from the sheet to the **Viz in Tooltip**. By default, `<All Fields>` means that all dimensions in the view will act as filters. However, you may specify a list of fields to specifically filter by one or more dimensions that are present in the view (for example, `<Department>`, `<Category>`).

Notice the final view with the tooltip:

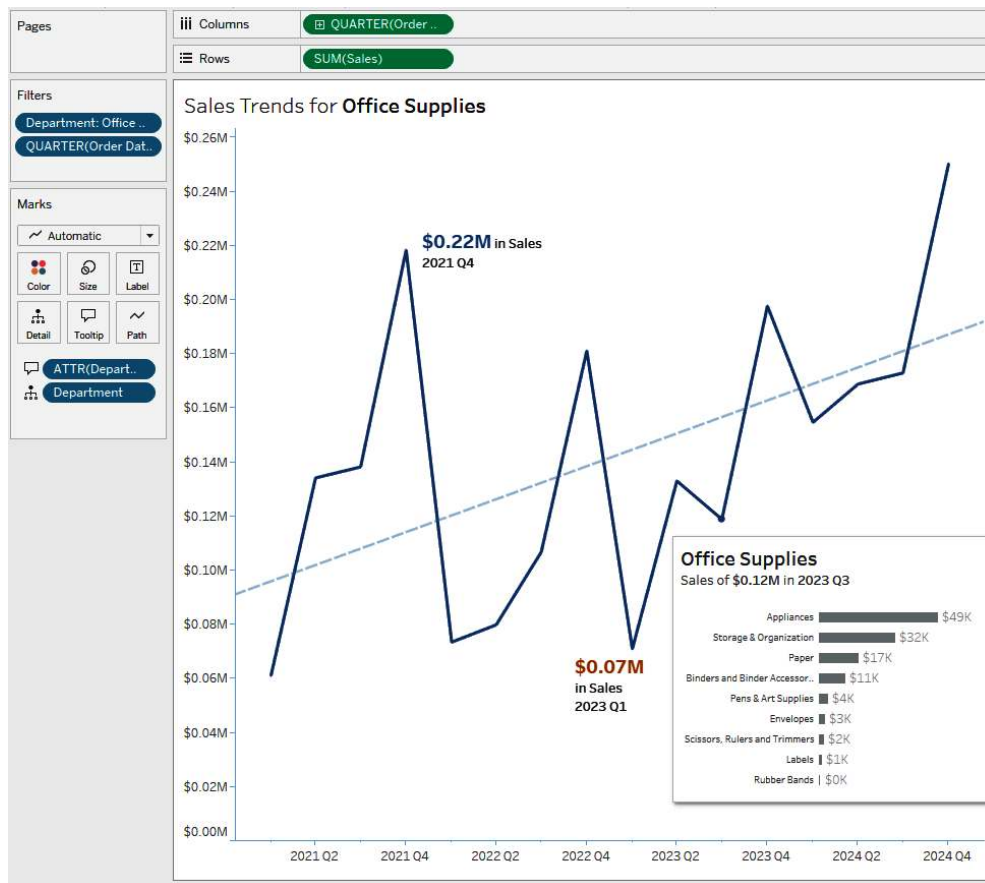


Figure 7.24: Viz in Tooltip

There are many possibilities with **Viz in Tooltip**. First, you can leverage the capability to drill down into details without using extra space on a dashboard and without navigating to a separate view. Second, you can show different aspects of the data (for example, geographic location as a tooltip for a time series). Finally, you might consider how to use **Viz in Tooltip** to allow the end user to see parts of the whole within a broader context.

There are a great many more valuable applications of this feature, but here are a few tips to wrap up our examination of **Viz in Tooltip**:

- You may have more than one Viz in a single tooltip.
- Setting the **Fit** option to **Entire View** for the sheet being used in a tooltip fits it to the maximum width and height.
- You may hide sheets used in tooltips by right-clicking the tab of the sheet and selecting **Hide**. To restore them, right-click the tab of the sheet being used in the tooltip and select **Unhide Sheets**.

Summary

The goal of formatting is to increase the effective communication of the data. Always consider the audience, setting, mode, mood, and consistency as you work through the iterative process of formatting. Focus on formatting that adds value to your visualization and avoids unnecessary clutter.

We covered quite a few options for formatting—from fonts, colors, lines, and more at the workbook level to formatting individual sheets and fields. We discussed how to customize formatting for numbers, dates, and null values and how to use these techniques to bring value to your visualizations.

With an understanding of how formatting works in Tableau, you'll be able to refine the visual elements that you created in discovery and analysis into highly effective visual communication of your data story.

In the next chapter, we'll take a deep dive into putting everything together in dashboards.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



8

Telling a Data Story with Dashboards

In data discovery and analysis, you will probably create numerous data visualizations. Each of these visualizations gives you a snapshot of a story within the data. Each insight into the data answers a question or two. At times, the discovery and analysis phase is enough for you to make a key decision and the cycle is complete. In other cases, you will need to bring the snapshots together to communicate a complete and compelling story to your intended audience.

Tableau allows you to bring together related data visualizations into a single dashboard. This dashboard could be a static view of various aspects of the data or a fully interactive environment, allowing users to dynamically filter, drill down, and interact with the data visualizations.

In this chapter, we'll take a look at most of these concepts within the context of several in-depth examples, where we'll walk through the dashboard design process, step by step. As before, don't worry about memorizing lists of instructions. Instead, focus on understanding why and how the components and aspects of dashboards work.

This chapter will cover the following topics:

- Introduction to dashboards
- Designing dashboards in Tableau
- A dashboard to understand profitability
- Designing for different displays and devices
- Interactivity with actions
- A scorecard dashboard
- Stories

For the examples in this chapter, we'll return to the Superstore Sales example data we used in the previous chapters. Go ahead and create a new workbook with a connection to that dataset, and we'll begin by introducing the key concepts of dashboards.

Introduction to dashboards

Before diving into some practical examples, let's take some time to understand what a dashboard is and why you might create one.

From a Tableau perspective, a **dashboard** is an arrangement of individual visualizations, along with other components, such as legends, filters, parameters, text, containers, images, extensions, buttons, and web objects, that are arranged on a single canvas. Ideally, the visualizations and components should work together to tell a complete and compelling data story. Dashboards are usually (but not always) interactive.

Dashboard objectives

The primary objective of a dashboard is to communicate data to a certain audience with an intended result. Often, we'll talk about *telling the data story*. That is, there is a narrative (or multiple narratives) contained within the data that can be communicated to others.

While you can tell a data story with a single visualization or even a series of complex dashboards, a single Tableau dashboard is the most common way to communicate a single story. Each dashboard seeks to tell a story by giving a clear picture of certain information. Before framing the story, you should understand what story the data tells. How you tell the story will depend on numerous factors, such as your audience, the way the audience will access the dashboard, and what response you want to elicit from your audience.

Stephen Few, one of the leading experts in the field of data visualization, defines a dashboard as “a visual display of the most important information that’s needed to achieve one or more objectives, consolidated and arranged on a single screen so the information can be monitored at a glance.”



This definition is drawn from Few's paper *Dashboard Confusion*, which can be read here: https://www.perceptualedge.com/articles/ie/dashboard_confusion.pdf.

This definition is helpful to consider because it places some key boundaries around the data story and the way we will seek to tell it in Tableau. In general, your data story should follow these guidelines:

- The data story should focus on the most important information. Anything that does not communicate or support the main story should be excluded. You may wish to include that information in other dashboards.
- The data story that you tell must meet your key objectives. Your objectives may range from giving information to providing an interface for further exploration to prompting your audience to act or make key decisions. Anything that doesn't support your objectives should be reserved for other dashboards.
- The data story should be easily accessible, and the main idea should be clear. Depending on your audience, you may wish to explicitly state your conclusions from the data, or you may want to guide your audience so that they can draw their own.

When you set out to build a dashboard, you'll want to carefully consider your objectives. Your discovery and analysis should have uncovered various insights into the data and its story. Now, it's your responsibility to package that discovery and analysis into a meaningful communication of the story to your particular audience in a way that meets your objectives and their needs. The way you handle this task is called your **approach**.

Dashboard approaches

There are numerous possible approaches to building dashboards based on your objectives. The following is by no means a comprehensive list:

- **Guided analysis:** You've done the analysis, made the discoveries, and thus have a deep understanding of the implications of the data story. Often, it can be helpful to design a dashboard that guides your audience through a similar process of making discoveries for themselves, so the need to act is clear. For example, you may have discovered wasteful spending in the marketing department, but the team may not be ready to accept your results unless they can see how the data led you to that conclusion.
- **Exploratory:** Many times, you do not know what story the data will tell when the data is refreshed in the next hour, next week, or next year. What may not be a significant aspect of the story today might be a major decision point in the future. In these cases, your goal is to provide your audience with an analytical tool that gives them the ability to explore and interact with various aspects of the data on their own. For example, today, customer satisfaction is high across all regions. However, your dashboard needs to give your organization the ability to continually track satisfaction over time, dynamically filter by product and price, and observe any correlations with factors such as quality and delivery time.
- **Scorecard/status snapshot:** There may be a wide agreement on **Key Performance Indicators (KPIs)** or metrics that indicate good and poor performance. You don't need to guide the audience through discovery or force them to explore. They just need a high-level summary and the ability to drill down into details to quickly find and fix problems and reward success. For example, you may have a dashboard that simply shows how many support tickets are still unresolved. The manager can pull up the dashboard on a mobile device and act immediately if necessary.



We'll consider Tableau Pulse Metrics as an alternative way to make KPIs accessible across your organization in *Chapter 17, AI Assisted Analysis: Tableau Agent and Tableau Pulse*.

- **Narrative:** This type of dashboard emphasizes a story. There may be aspects of exploration, guided analysis, or performance indication but, primarily, you are telling a single story using the data. For example, you may desire to tell the story of the spread of an invasive species of insect, including where it started, how quickly it spread, the results, and the efforts to contain it. Your dashboard should tell the story, using data, in a visual way.

We'll look at several in-depth examples to better understand a few of these different approaches and incorporate many of the skills we covered in previous chapters. First, we'll introduce some key aspects of designing dashboards in Tableau.

Your dashboard may have a hybrid approach. For example, you might have an exploratory dashboard that prominently displays some KPIs. However, be careful to not overload a dashboard. Trying to meet more than one or two objectives with any single dashboard will probably result in an overwhelming mess.

Designing dashboards in Tableau

No matter your objective or approach, the practical task of designing a dashboard in Tableau will look similar each time. In this section, we will go through some fundamental concepts.

Using objects

Dashboards are made up of objects that are arranged on a canvas.

You'll see a list of objects that can be added to a dashboard in the left-hand pane of a dashboard:

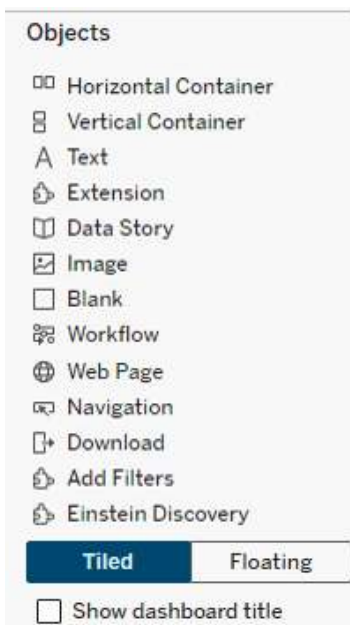


Figure 8.1: Objects available to add to a dashboard

The pane includes these objects:

- **Horizontal Container:** A layout container within which other objects will be arranged in a single row (horizontally).
- **Vertical Container:** A layout container within which other objects will be arranged in a single column (vertically).

- **Text:** An object that allows you to include richly formatted text in your dashboard.
- **Extension:** One of a growing number of tools developed by Tableau and third parties (or maybe even you!) that leverages the extensions API to provide extended functionality to dashboards. This could allow you to accomplish things such as gathering extensive usage data, dynamically updating parameters, incorporating visualizations from other platforms (such as D3), and much more. We'll consider writing our own extensions in depth in *Chapter 16, Extending Tableau: Extensions and Scripts*.
- **Data Story:** A natural language description and summary of data in a view on the dashboard. The analysis and resulting summary are auto-generated but may be configured and customized. This component requires Tableau Server or Tableau Cloud.
- **Image:** An image (for example, .gif, .png, or .jpeg) that can be positioned and sized on your dashboard. Optionally, you may set a URL for navigation when a user clicks the image.
- **Blank:** A blank object that can be used as a placeholder or to provide spacing options.
- **Workflow:** An object that allows you to automate the flow of data to a **Salesforce Flow** (a business automation application from Salesforce).
- **Web Page:** An object that allows you to embed web content in the dashboard. You can set the URL at design time. We'll also consider how to use actions to dynamically change the URL.
- **Navigation:** A navigation button allows you to define user navigation to other sheets and dashboards in the workbook.
- **Download:** A download object adds a download button (as an image or text) that gives the end user an easy option to export the dashboard or part of the dashboard as an image, crosstab, PDF, or PowerPoint.
- **Add Filters:** Provides a configurable control that allows the end user to show or hide various components of the dashboard.
- **Einstein Discovery:** Provides an AI/machine learning component to the web page that allows for the use of predictive models to make predictions and forecasts based on your data. This component requires additional Salesforce licensing. We'll consider some approaches to AI in Tableau in *Chapter 17, AI-Assisted Analysis: Tableau Agent and Tableau Pulse*.

In addition to the objects that you can add through the sidebar, there are other objects that may be applicable to a given dashboard:

- **Filters:** These will appear as controls for the end user so that they can select values to filter
- **Parameters:** Similar to filters, parameters will show up as controls for the end user to select a parameter option
- **Page controls:** These are controls that give the end user options for paging through the data and may be used to animate visualizations and dashboards
- **Legends:** These include color, size, and shape legends to help the end user understand various visualizations
- **Highlighters:** These allow the user to highlight various dimension values within views
- **Dashboard title:** A special text object that displays the name of the dashboard sheet by default

Tiled versus floating

An object on the dashboard is either tiled or floating. If it is a tiled object, it will snap into the dashboard or layout container where you drop it. If it is a floating object, it will float on the dashboard in layers. You can change the order of the layers for a floating object.

You'll notice the **Tiled** and **Floating** buttons directly beneath the **Objects** palette, as shown in *Figure 8.1*. These buttons define the default setting for objects that you place on the dashboards, but you can change whether any given object is tiled or floating.

Hold down the Shift key as you drag an object to quickly change it from tiled to floating, or from floating to tiled.

As you become experienced in designing dashboards, you'll probably develop a preference for designing using a predominately tiled approach or a predominately floating approach (you can mix tiled and floating objects on any dashboard). Many designers find that one design method or the other fits their style. Here are some considerations:

- **Precision:** Floating objects can be sized and positioned to exact pixel-perfection, while tiled objects will depend greatly upon their containers for position and size.
- **Speed:** Many designers find a tiled layout much faster to create as they don't have to worry about precision or layering.
- **Dynamic resizing:** Floating objects work well on a fixed-size dashboard, but a dashboard that dynamically resizes based on window size will shift floating objects, often into undesirable locations. Tiled objects move and resize more reliably (but not always perfectly!).
- **Flexibility:** Certain design techniques can be accomplished with one approach or the other. For example, transparent visualizations can be layered on top of background images using a floating technique. However, sheet swapping (which we'll look at in *Chapter 11, Dynamic Dashboards*) is often accomplished with a tiled approach.
- **Maintenance:** Changes to the layout of a floating dashboard might be harder and more tedious than doing so for tiled layouts.

Experiment with various design techniques, and feel free to develop your own style!

Manipulating objects on the dashboard

You may wish to manipulate an object once it is part of a dashboard. Every object has certain controls that become visible when you select it. We see these controls here:

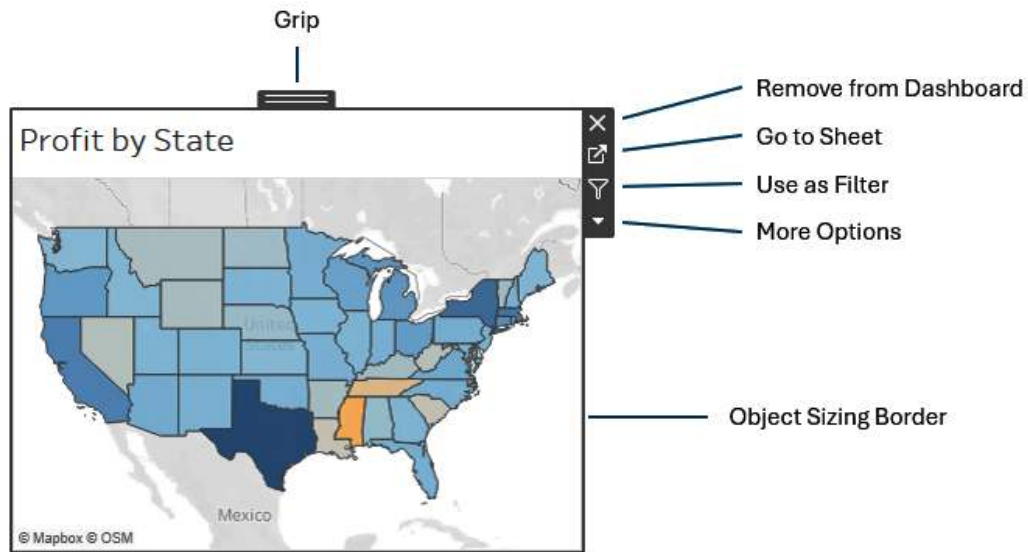


Figure 8.2: Various controls become accessible once you select a dashboard object

Let's see how we can use these controls:

- **Grip:** Click and drag the grip to move the object.
- **Remove from Dashboard:** Click this to remove the object from the dashboard.
- **Go to Sheet:** To edit a single visualization on a dashboard, use this button to navigate to the individual sheet.
- **Use as Filter:** Clicking here will enable the view to be used as a filter. Selecting a mark in the view will now filter other views on the dashboard. We'll look at the specifics of filter actions later in this chapter and how to fine-tune those actions.
- **More Options:** This drop-down arrow reveals a host of options for the object, including control display options for parameters and filters; showing or hiding titles or captions on views; adding legends, parameters, and filters to the dashboard; formatting, layout, and size options; and more.
- **Object Sizing Border:** Hovering over the border will cause your cursor to change to a sizing cursor. You can drag the border to adjust the size of the object.

You may notice different sizing behavior based on what type of container an object is inside and whether the object is tiled or floating.

In the first sections of this chapter, we have introduced the theoretical fundamentals of dashboards and some key elements of their design. Now, let's apply these concepts with a practical example.

A dashboard to understand profitability

Having covered some conceptual topics as well as practical matters related to dashboard design, we'll dive into an example.

For the most part, we'll accept Tableau's default formatting for this example. However, at times, you may notice formatting applied in screenshots and the complete workbook that is not specifically called out in the instructions. Feel free to employ some of the formatting capabilities you learned in the previous chapter to enhance your dashboard and give it your own individual style.

Let's say you've been tasked with helping management find which items are the least profitable. Management feels that most of the least profitable items should be eliminated from their inventory. However, since you've done your analysis, you've discovered that certain items, while not profitable overall, have made a profit at times in various locations. Your primary objective is to give management the ability to quickly see an analysis of the least profitable items to identify whether an item has always been unprofitable, answering the question, "Is the least profitable item always unprofitable?" This example will combine aspects of a guided analytics dashboard and an exploratory tool.

Building the views

Use the Superstore Sales dataset and follow these steps to build the individual views that will form the basis of the dashboard:

1. Create a bar chart showing profit by category. Sort the categories in descending order by the sum of profit.
2. Add the **Department** field to **Filters** and show a filter. To accomplish this, use the drop-down menu of the **Department** field in the data pane and select **Show Filter**.
3. Name the sheet **Overall Profit by Category**:



Figure 8.3: A bar chart showing the sum of Profit by Category with Department as a filter

4. Create another similar view showing profit by item. Sort the items in descending order by the sum of profit.
5. You'll notice that there are too many items to see at once. For your objectives on this dashboard, you can limit the items to only the top 10 least profitable ones. Add the **Item** field to the **Filters** shelf, select the **Top** tab, and adjust the settings to filter by field. Specify **Bottom** 10 by Sum(Profit):

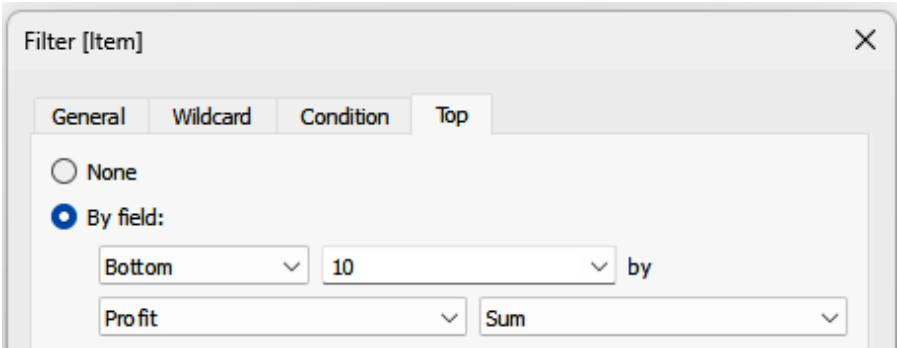


Figure 8.4: Use the Top tab to set the number of items to display

6. Rename the sheet Top 10 Least Profitable Items:

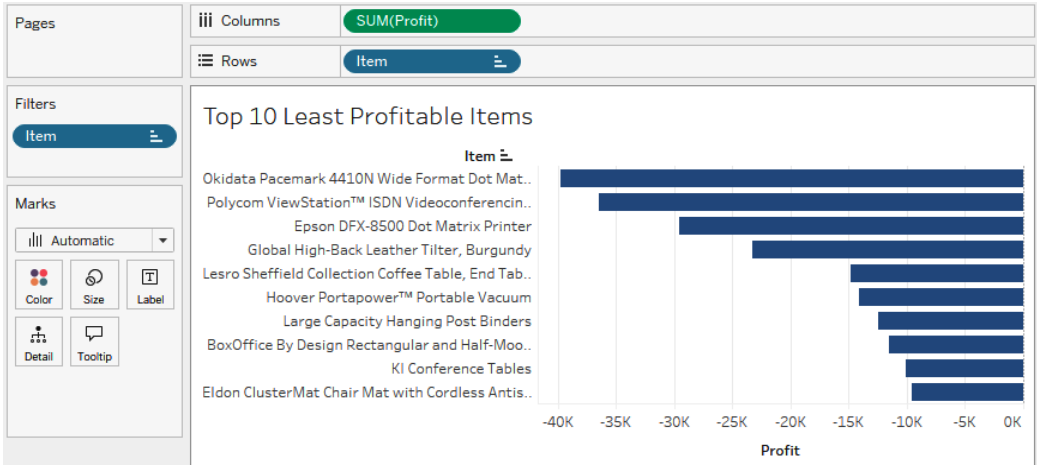


Figure 8.5: The resulting bar chart shows the top 10 least profitable items

7. Create another sheet that displays a filled map of profit by state. You can accomplish this rather quickly by double-clicking the **State** field in the data window and then dropping **Profit** on the **Color** shelf.

8. Rename the sheet to Profit by State:

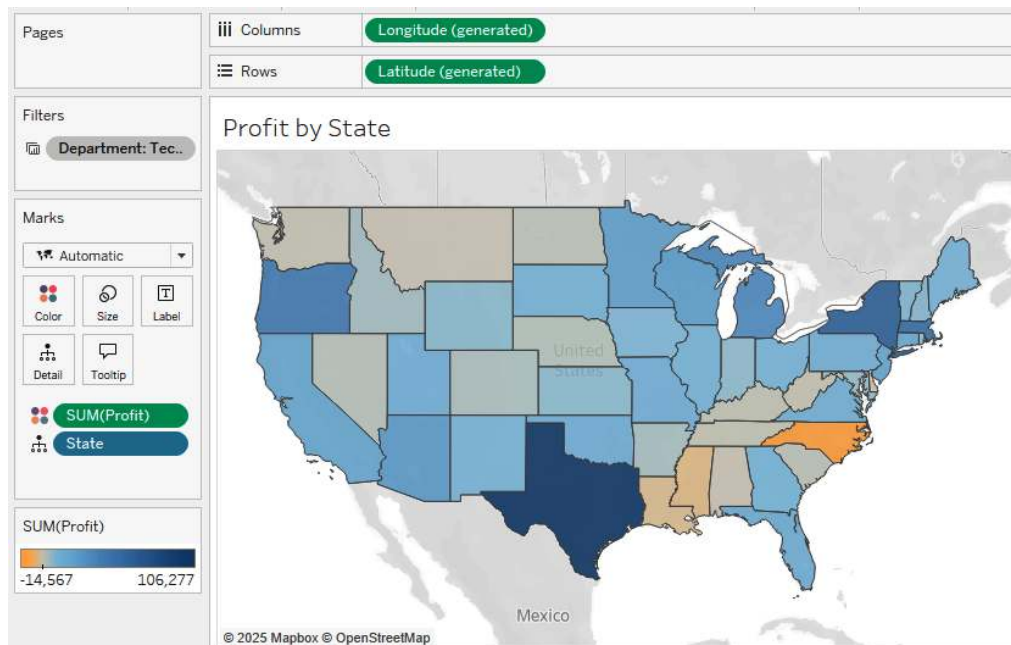


Figure 8.6: A filled map showing profit by state

9. Create one final sheet to show when profits were made or lost. Ensure that the **Order Date** field has been added as the **Quarter** date value and that it is continuous (green).
10. Add a linear trend line. To do this, switch to the **Analytics** tab of the left sidebar and drag **Trend Line** from **Model** to the view. Alternatively, right-click a blank area of the canvas of the view and select **Trend Lines** | **Show Trend Lines**. Edit the formatting to remove confidence bands and adjust the look and feel as desired.

11. Rename the sheet Profit Trend:

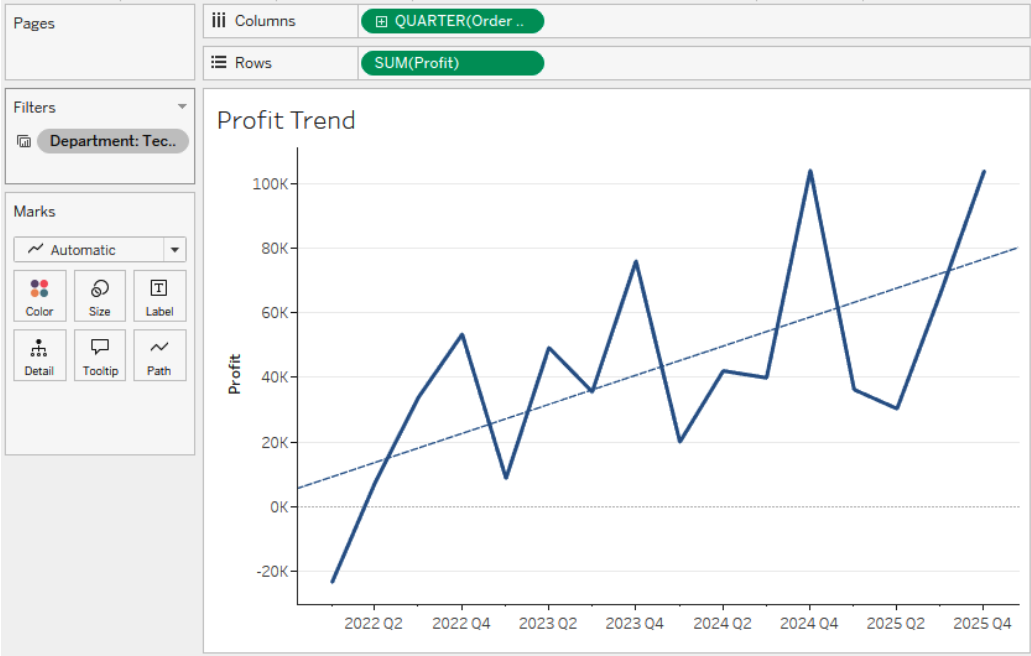


Figure 8.7: A line chart showing the trend of profit by quarter

Now that you’ve created the views that will make up the dashboard, let’s start to put the dashboard together!

Creating the dashboard framework

At this point, you have all the necessary views to achieve the objectives for your dashboard. Now, all that remains is to arrange them and enable the interactivity that’s required to effectively tell the story:

1. Create a new dashboard by clicking the **New Dashboard** tab to the right of all existing worksheet tabs or by selecting **Dashboard | New Dashboard** from the menu.
2. Rename the new dashboard **Is Least Profitable Always Unprofitable?**.
3. At the bottom of the left sidebar, check **Show dashboard title**.

4. Add the views to the dashboard by dragging them from the **Dashboard** pane in the left sidebar and dropping them into the dashboard canvas. Your arrangement should look something like this:

Is Least Profitable Always Unprofitable?

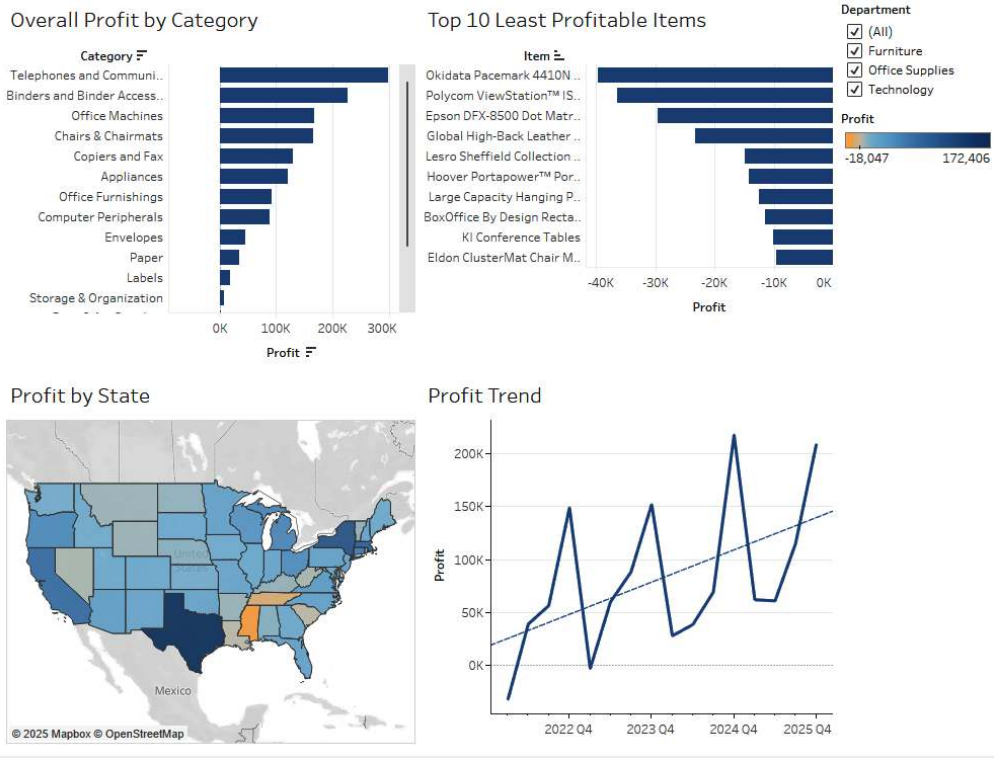


Figure 8.8: All views are placed on the dashboard

After adding views to the dashboard, you'll want to take some time to reposition and resize various elements and views.

5. Use the drop-down menu on the **Department** filter and change the control to **Single Value** (drop-down).

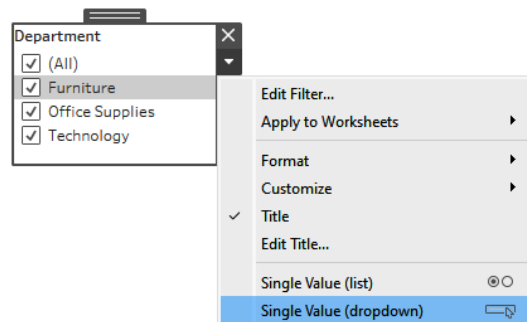


Figure 8.9: Changing the control

6. You'll notice that changing the value of the filter only changes the **Overall Profit by Category** view. You can adjust which views the filter applies to: from the drop-down menu, select **Apply to Worksheets | All Using This Data Source**.



Options for applying filters may be set using the dropdown on the filter control or on the field on the **Filters** shelf in the view. The options include the following:

- **All Using Related Data Sources:** The filter will be applied to all data sources where the field used for filtering is related between data sources. Relationships may be edited from **Data | Edit Relationships** on the main menu.
- **All Using This Data Source:** The filter will be applied to any view using the data source as the primary data source.
- **Selected Worksheets...:** The filter will be applied to the worksheets you select.
- **Only This Worksheet:** The filter will be applied only to the current worksheet.

Continue creating the dashboard with the following steps.

7. From the left sidebar, drag and drop a **Text** object above **Overall Profit by Category** and enter the following instructions:
 1. Select a Department from the drop-down
 2. Select a category below
 3. Select an Item below



You'll want to consider how best to give user instructions or directions in a dashboard. Here, you're being instructed to insert text even before the dashboard is created. In most real-world cases, you'll want to complete a dashboard and then determine how to best guide the end-users to use the dashboard.

8. Using the grip, move the **Department** filter immediately above the **Top 10 Least Profitable Items** view.
9. Size the text object to align the **Top 10** view with the overall view.
10. Move the **Profit** color legend below the **Profit by State** view.
11. Use the drop-down menu of **Overall Profit by Category** to **Fit | Entire View**. This will ensure that all of the categories are visible without the need for a scrollbar.
12. Additionally, fit **Top 10 Least Profitable Items** to **Entire View**.

At this point, your dashboard should look similar to the following:

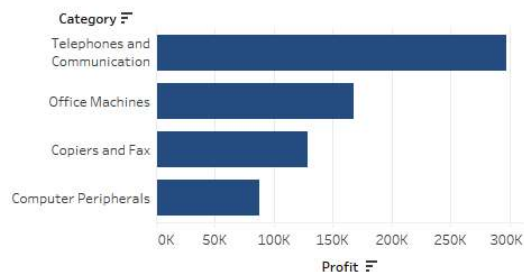
Is Least Profitable Always Unprofitable?

1. Select a Department from the dropdown
2. Select a category below
3. Select an Item below

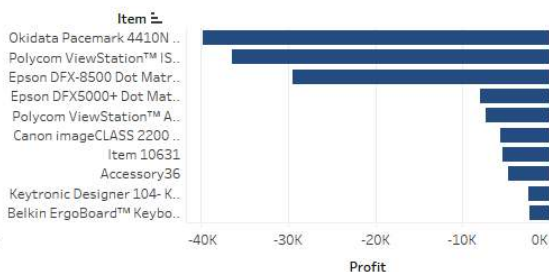
Department

Technology

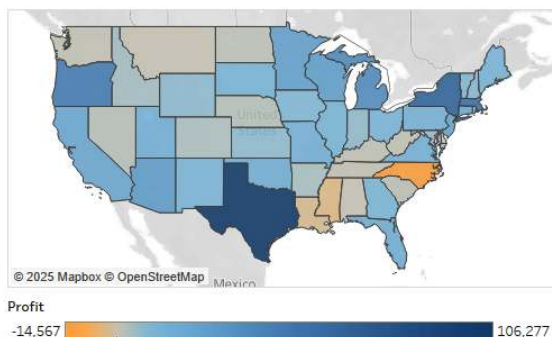
Overall Profit by Category



Top 10 Least Profitable Items



Profit by State



Profit Trend

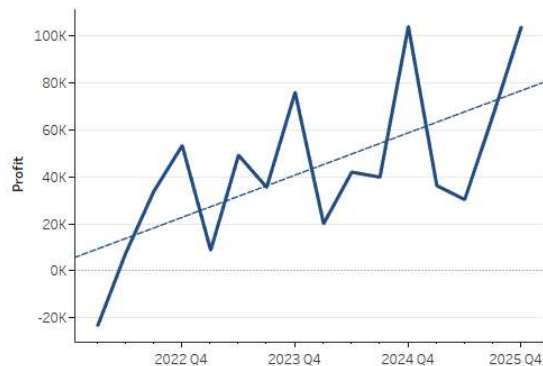


Figure 8.10: The polished dashboard with rearranged and resized objects

We now have a dashboard with all the views we want to include. As you've seen, it's easy to add views and objects and rearrange them as desired. Let's continue by examining how to drive a story with actions.

Implementing actions to guide the story

You now have a framework that will support the telling of the data story. Your audience will be able to locate the least profitable items within the context of a selected category. Then, the selection of an item will answer the question of whether it has always been unprofitable in every location. To enable this flow and meet your objectives, you'll often need to enable interactivity. In this case, we'll use actions. We'll conclude this example with some specific steps and then unpack the intricacies of actions later in the chapter:

1. Click the **Use as Filter** button on the **Overall Profit by Category** view. This will cause the view to be used as an interactive filter for the entire dashboard. That is, when the user selects a bar, all other views will be filtered based on the selection:

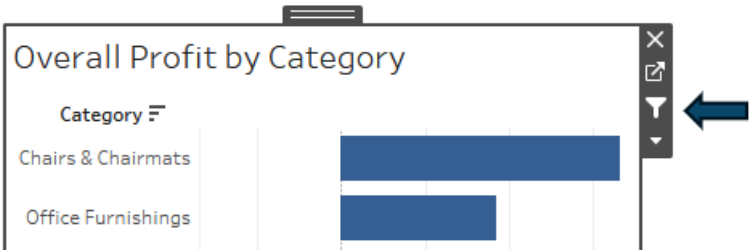


Figure 8.11: The Use As Filter control on the Profit by Category view

2. From the main menu, select **Dashboard | Actions**. In the dialog, switch to view actions only from **This Sheet**. You’ll see a list containing one action named **Filter 1 (generated)**. This is the action that was created when you selected **Use as Filter** previously:

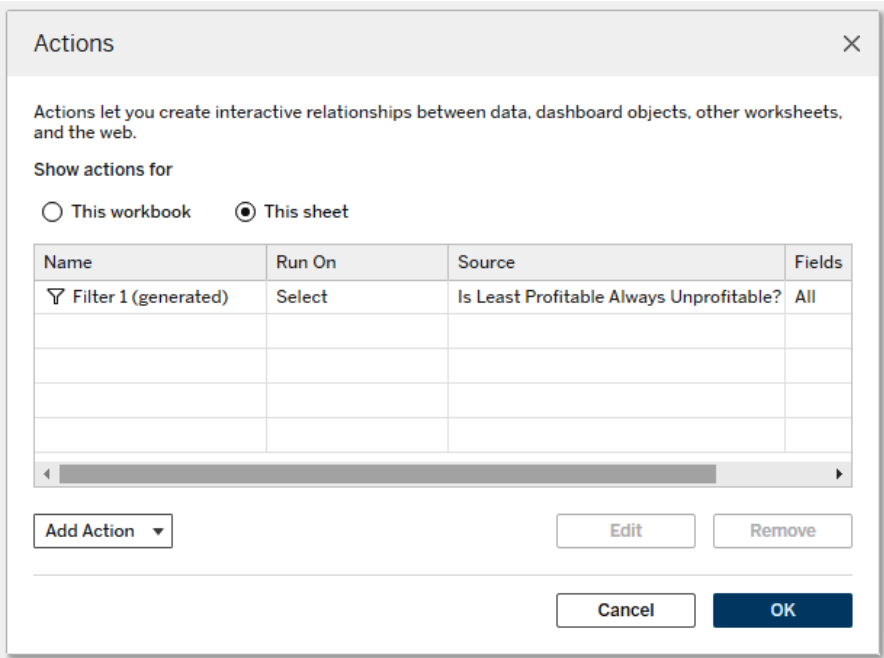


Figure 8.12: Filter 1 (generated) was created when the filter control was clicked

As a good practice, **Edit** the name of generated actions to something more meaningful. For example, you might name this action **Filter by Category**. This will make your actions far easier to maintain.

3. Click the **Add Action** button and select **Filter**. The resulting dialog gives you options for selecting the source and target, as well as additional options for the action.
4. Here, we want an action that filters everything except the **Overall Profit by Category** view when the user selects an item. In the **Add Filter Action** dialog, set **Source Sheets** to **Top 10**

Least Profitable Items, and **Target Sheets to Profit by State and Profit Trend**. Make sure that the action is set to run on **Select**. Name the filter **Filter by Item**.

Add Filter Action

Name

Filter by Item

Insert

Source Sheets

Is Least Profitable Always Unprofitabl...

Overall Profit by Category

Profit by State

Profit Trend

Top 10 Least Profitable Items

Run action on

Hover

Select

Menu

Single-select only

Target Sheets

Is Least Profitable Always Unprofitabl...

Overall Profit by Category

Profit by State

Profit Trend

Top 10 Least Profitable Items

Clearing the selection will

Keep filtered values

Show all values

Exclude all values

Filter

All fields

Selected fields

	Source Field	Target Data Source	Target Field
	Click to add		

Remove

Cancel

OK

Figure 8.13: The setup for the **Filter by Item** action

5. Click **OK** to save the settings on the **Add Filter Action** dialog and then click **OK** again on the **Actions** dialog.

You now have three filters (two are actions) that drive the dashboard:

- Selecting a department from the dropdown will filter the entire dashboard (it actually filters all views in the workbook as you set it to filter every view using the data source)
- Selecting a category (clicking a bar or header) will filter the entire dashboard to that selection
- Selecting an item (clicking a bar or header) will filter the **Profit by State** and **Profit Trend** dashboards

You can clear a selection in a view by clicking a blank area or by clicking the selected mark one more time. For example, if you click the bar for **Bookcases** to select it (and thus filter the rest of the dashboard), you may click the bar one more time to deselect it.

Experiment with the filters and actions to see how your dashboard functions.

Context filtering

You may have noticed that when you use the drop-down filter to select a single department or select a single category, you have fewer than 10 items in the **Top 10** view. For example, selecting **Furniture** from the **Department** filter and clicking on the bar for **Tables** results in only three items being shown. This is because the **Top Item** filter is evaluated at the same time as the action filter. There are only three items with the category of **Tables** that are also in the **Top 10**.

What if you want to see the top 10 items within the category of **Tables**? You can accomplish this using context filters.



Context filters are special filters in Tableau that are applied before other filters. Other filters are then applied within the “context” of the context filters. Conceptually, context filters result in a subset of data upon which other filters and calculations operate. In addition to **Top Filters**, **Computed Sets** and **Fixed Level of Detail** calculations are also computed within the context defined by context filters.

In this case, navigate to the **Top 10** sheet and add the **Department** filter and the newly added action (**Category**) filter to the context using the drop-down menu of the fields on the **Filters** shelf. Once added to the context, those fields will be gray on the **Filters** shelf. Now, you will see the top 10 items within the context of the selected department and category:

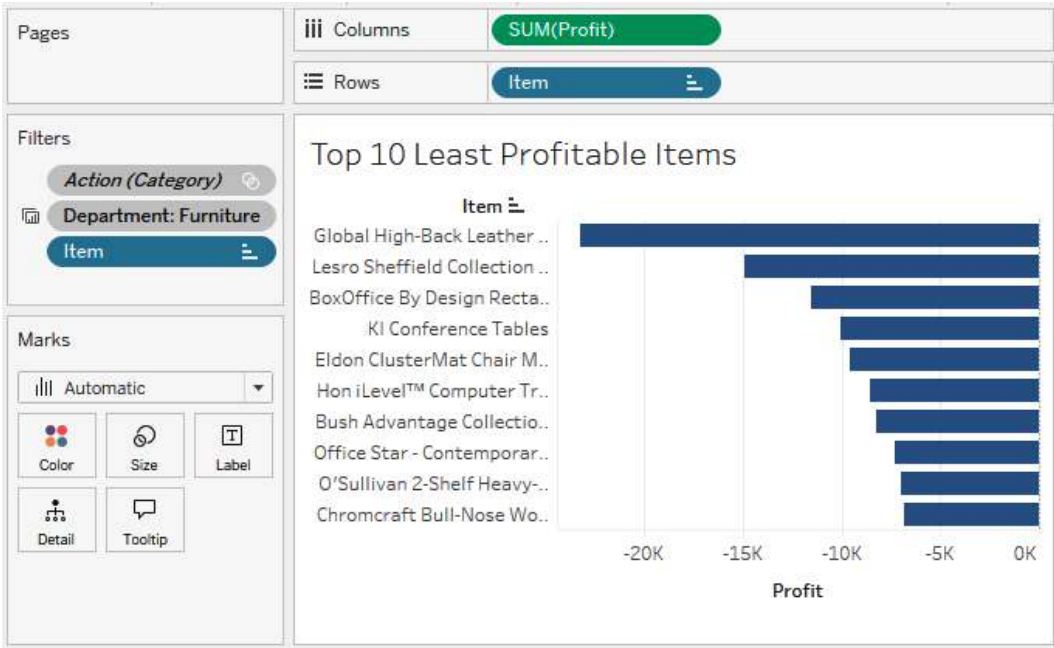


Figure 8.14: The 10 least profitable items will be within the context of the Action (Category) and Department filters

Notice that adding filters to the context causes the fields to be color-coded gray on the **Filters** shelf.

If you edit the action on the dashboard, the filter might be automatically updated and you may have to re-add it to the context.

Go ahead and step through the actions by selecting a couple of different categories and a couple of different items. Observe how the final dashboard meets your objectives by telling a story:

Is Least Profitable Always Unprofitable?

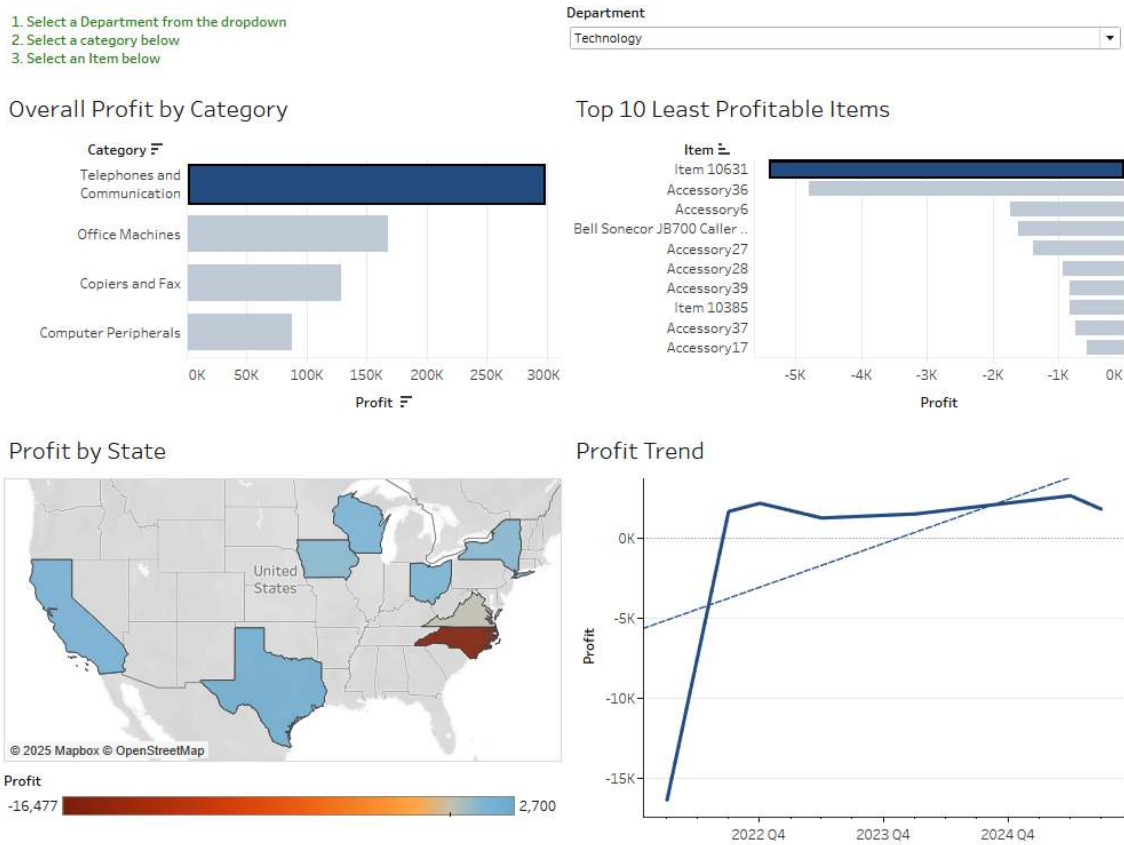


Figure 8.15: The final dashboard with filters triggered by selection

The user has selected **Technology** from the **Department** dropdown, **Telephones and Communications** from the **Category** list, and then **Item 10631**, which is the least profitable item within the category. This reveals the states where the item was sold (color-coded by profit) and a time series of profit for the item.

Should management remove item 10631 from the inventory? Not without first considering that the item only lost profit in one instance and that the trend is positive toward greater profitability. Granted, the original loss was a large loss, but this was also a long time ago and every subsequent sale of the item resulted in a profit. The results of your findings may lead to further analysis to determine what factors play a part in the profit and loss for the item and better decision-making by management.

When you look at the **Chapter 08 Completed** workbook, you'll only see a tab at the bottom for the dashboard. The individual views have been hidden. Hiding tabs for sheets that are used in dashboards or stories is a great way to keep your workbook clean and guide your audience away from looking at sheets that are meant to be seen in the context of a dashboard or story. To hide a sheet, right-click the tab and select **Hide Sheet**. To unhide a sheet, navigate to the dashboard or story using the sheet, right-click the sheet in the left-hand pane, and uncheck **Hide Sheet**. Additionally, you can hide or unhide all sheets that are used in a dashboard by right-clicking the **Dashboard** tab and selecting the appropriate option. Sheets that are used in tooltips may be hidden or unhidden in the same way.

You now have a fully interactive dashboard! You built the views, added them to the dashboard, and then created some meaningful calculations. Along the way, you learned about **top filters** and **context filters**. Now, let's discuss how to design dashboards for different displays and devices.

Designing for different displays and devices

When designing a dashboard, some of the first questions you'll often ask yourself are: *How will my audience view this dashboard? What kind of device will they use?* With the wide adoption of mobile devices, this latter question becomes very important because what looks great on a large flat-screen monitor doesn't always look great on a tablet or phone. You'll want to consider everything from the size of fonts to the layout of objects to ensure the best experience for the end-user.

The top of the **Dashboard** tab on the left sidebar reveals a button to preview the dashboard on various devices, as well as a dropdown for **Size** options:

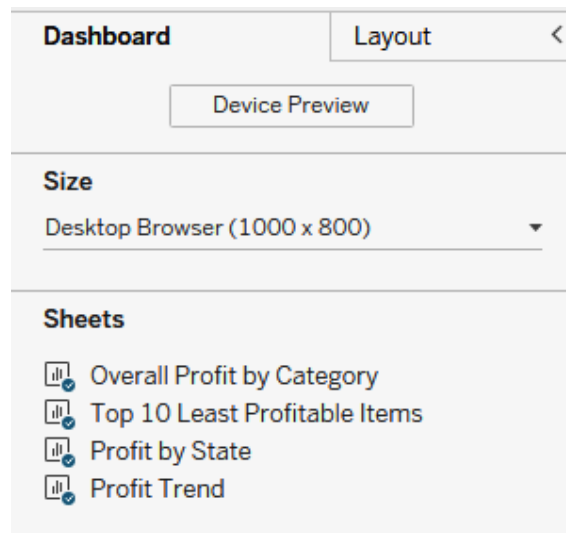


Figure 8.16: The Device Preview option allows you to design and preview your dashboard for other devices

Clicking the **Device Preview** button not only allows you to see how your dashboard will look with various device types (and even specific models) but also allows you to add a layout for each device type, which you can customize:



Figure 8.17: Customizable options for devices

You can not only see how your dashboard will appear on various devices and models but also how it will look based on the orientation of the device and whether the Tableau Mobile app is used (if available for the selected device).

Clicking the **Add Layout** button (that is, the **Add Tablet Layout** button in the preceding screenshot) will add a layout on the **Dashboard** tab on the left sidebar:

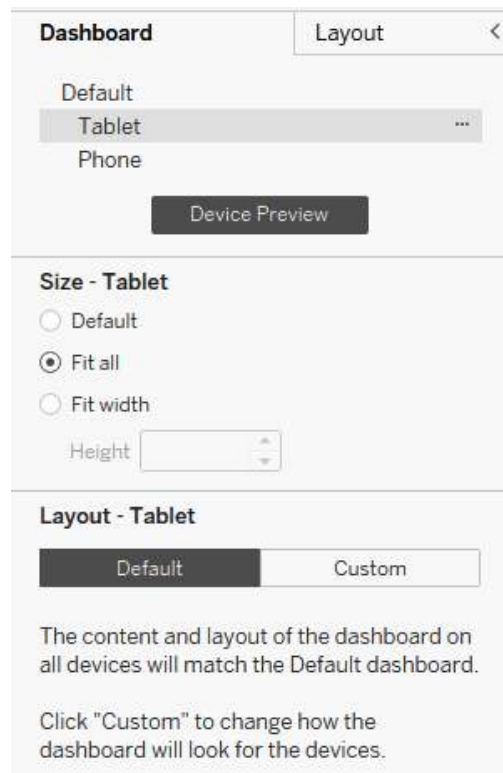


Figure 8.18: Each layout can be configured with various options

Each layout can have its own size and fit options, and the layout options allow you to switch from **Default** to **Custom**. This gives you the ability to rearrange the dashboard for any given layout. You may even remove views and objects for a certain layout. For example, you might simplify a dashboard to one or two views for a phone while leaving three or four in place for a desktop display.

The Chapter 08 Completed workbook contains an example of the profit analysis dashboard and has a couple of layout options. For example, here is that dashboard formatted for display on a phone in which the dashboard will fit according to the width of the phone and allow scrolling up and down:

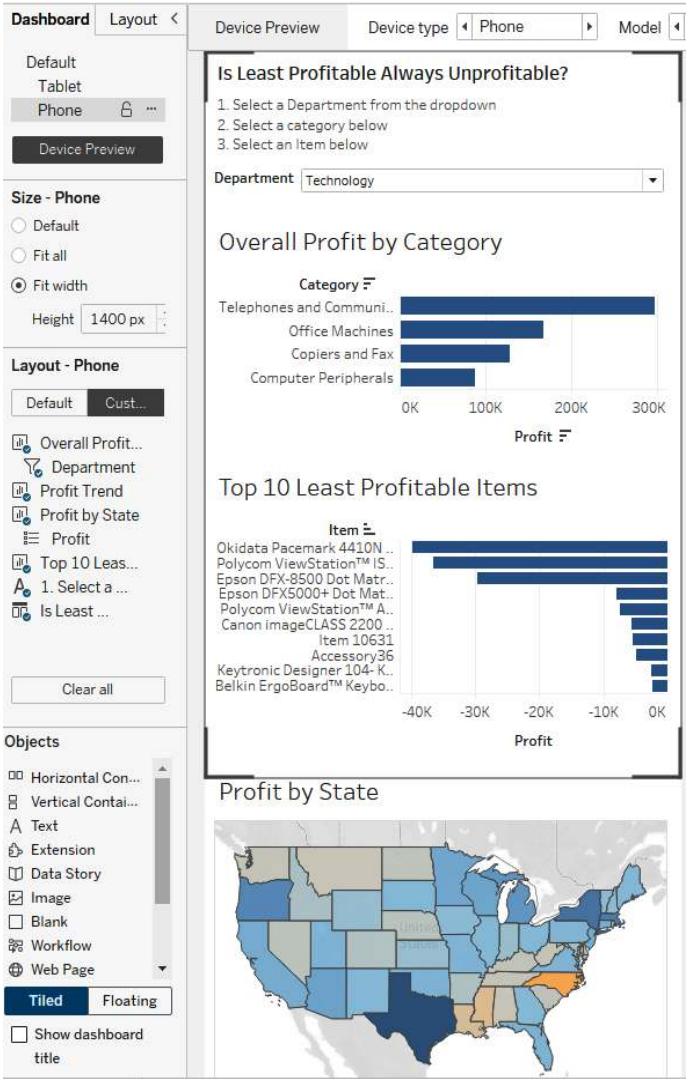


Figure 8.19: The phone layout of the dashboard

As you can see, the arrangement of the dashboard for the phone means that **Profit by State** and **Profit Trend** do not appear within the preview boundaries for a phone device. However, they are only a finger swipe away.

Interactivity with actions

As you build a dashboard, you'll often want to design with end user interactivity in mind. An **action** is a user-initiated event that triggers a defined response in a dashboard or worksheet. For example, you might want a click on a bar chart to filter other views in the dashboard or you might want to highlight all the marks in a scatterplot that match values on a chart as the user hovers over them. We'll see how to create these actions and more!

We'll start by considering what user interactivity can trigger an action.

Triggers of actions

As we'll see, when you create an action, you'll be able to specify what end user interaction triggers an event. Actions can be set to occur on any one of three possible triggers:

- **Hover:** The action is triggered when the user moves the mouse cursor over a mark (or taps a mark on a mobile device).
- **Select:** The action is triggered when the user selects a mark. Specifically, a user may select a mark by clicking or tapping, clicking and holding to draw a rectangle, radius, or lasso around multiple marks, or by clicking a row or column header. In this final case, all marks for that header are selected. A user may deselect by clicking the already selected mark, clicking an empty space in the view, or clicking the already selected header.
- **Menu:** The action is triggered when the user selects the menu option in a tooltip or header context menu.

With this in mind, let's discuss how to create specific actions.

Creating actions

Certain actions are automatically generated by Tableau based on shortcuts. For example, you might select **Use as Filter** from the drop-down menu of a view on a dashboard, which results in an automatically generated filter action. Enabling highlighting using the button on a discrete color legend or from the toolbar will automatically generate a highlight action:

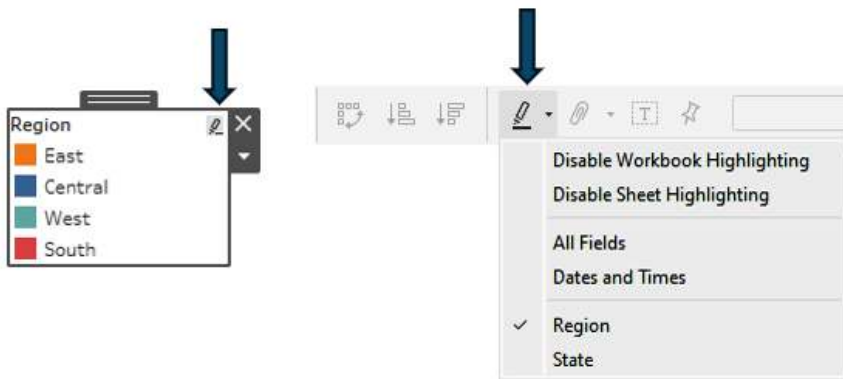


Figure 8.20: Options for enabling highlighting

Many times, you will create actions by selecting **Dashboard | Actions** from the menu. This will open a dialog where you can manage existing actions and create new ones:

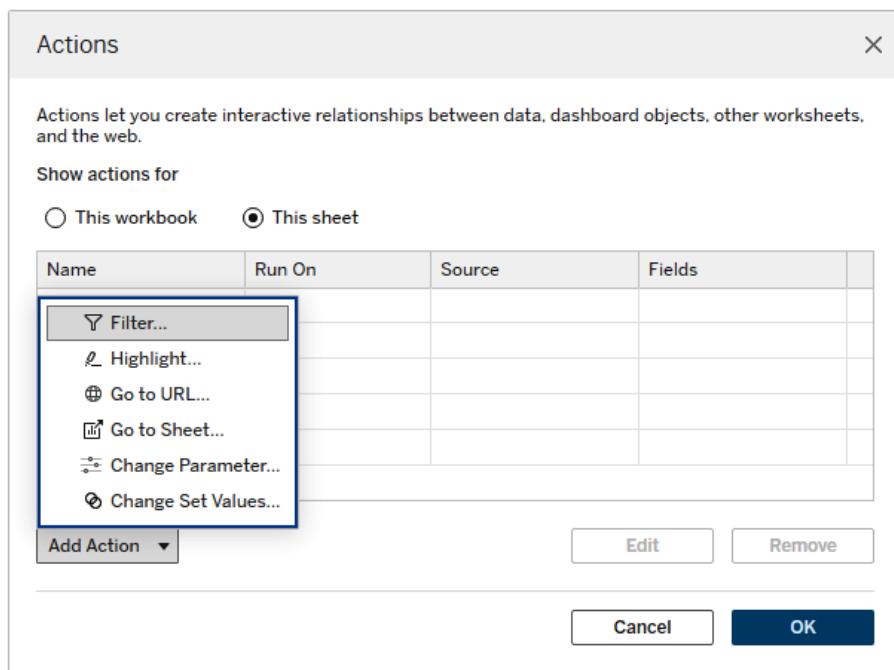


Figure 8.21: The Actions dialog allows you to manage existing actions and create new ones.

As you can see, clicking **Add Action** prompts you with options for multiple types of actions. We'll turn our attention to the specific types of actions next.

Types of action

Tableau supports six kinds of action:

- **Filter:** The user's action causes one or more filters to be applied to one or more views.
- **Highlight:** The user's action causes specific marks and headers to be highlighted in one or more views.
- **Go to URL:** The user's action causes a specific URL to be opened (either in a browser, in a new tab, or in an embedded web object).
- **Go to Sheet:** The user's action causes navigation to a selected sheet or dashboard.
- **Change Parameter:** The user's action changes the value of a parameter. This allows the user to visually interact with parameters in new and exciting ways!
- **Change Set Values:** The user's action defines a set. Sets may be used in calculations, in filters, and on shelves to define visual attributes of marks. This opens a lot of possibilities to allow complex and creative interactions.

Let's discuss the details of each type.

Filter actions

Filter actions are defined by one or more source sheets that pass one or more dimensional values as filters to target sheets upon an action. Remember that every mark on a sheet is defined by a unique intersection of dimensional values. When an action occurs involving one or more of those marks, the dimensional values that comprise the mark(s) can be passed as filters to one or more target sheets.

When you create or edit a filter action, you will see options like these:

Edit Filter Action

Name

Filter by Item Insert ▼

Source Sheets

Is Least Profitable Always Unprofitabl... ▼

☐ Overall Profit by Category
☐ Profit by State
☐ Profit Trend
☒ Top 10 Least Profitable Items

Run action on

☐ Hover
☒ Select
☐ Menu
☐ Single-select only

Target Sheets

Is Least Profitable Always Unprofitabl... ▼

☐ Overall Profit by Category
☒ Profit by State
☒ Profit Trend
☐ Top 10 Least Profitable Items

Clearing the selection will

☐ Keep filtered values
☒ Show all values
☐ Exclude all values

Filter

☒ All fields ☐ Selected fields

☐	Source Field	Target Data Source	Target Field
☐	Click to add ▼		

Remove

Cancel OK

Figure 8.22: Options for filter actions

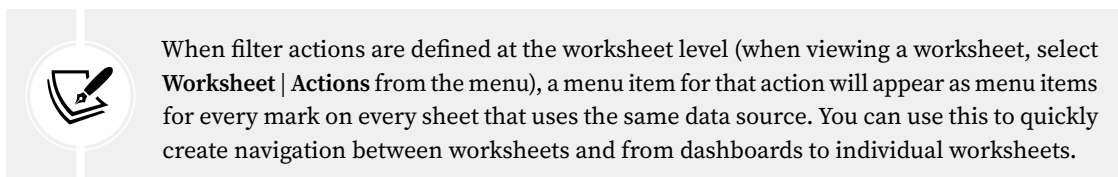
This screen allows you to do the following:

- Name the filter.
- Choose **Source** and **Target** sheets. The **Source** sheet is where the user will initiate the action and the **Target** sheet is where the response will be applied.
- Set the action that triggers the filter (hover, selection, or menu) and whether the selection of multiple marks or only a single mark initiates the action.

- Choose what happens when the selection is cleared.
- Specify which dimensions are used to pass filter values to the **Target** sheet(s).

Try to give your actions names that help you differentiate between multiple actions in the dashboard. Additionally, if your action is set to run on **Menu**, then the name you use will be shown as a link in the tooltip. Use the arrow to the right of the name to insert special field placeholders. These will be dynamically updated with the values of the fields for a mark when the user sees the menu option in a tooltip.

You may select as many source and target sheets as you desire. However, if you specify particular **Target filters** in the bottom section, the fields you select must be present in the source sheet (for example, on **Rows**, **Columns**, and **Detail**). You will receive a warning if a field is not available for one or more **Source** sheets and the action will not be triggered for those sheets. Most of the time, your source and target will be the same dashboard. Optionally, you can specify a different target sheet or dashboard, which will cause the action to navigate to the target in addition to filtering.



Consider the following example of a filter action that's triggered when a bar is selected in the source:

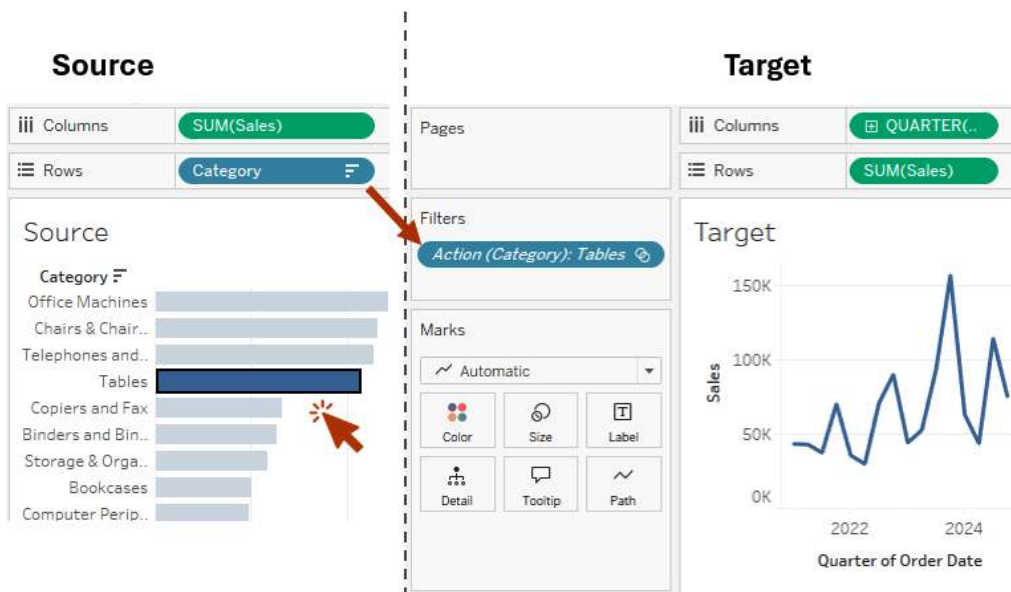


Figure 8.23: Clicking the bar for Tables passes Category as a filter to the Target sheet

Each bar mark in the source is defined by the **Category** dimension. When the bar for **Tables** is selected, a single filter is set on the target.

If the mark is defined by more than one dimension (for example, **Category** and **Region**), then the target sheet will still have a single filter with the combination of dimension values that had been selected.

In this example, the filter contains **Office Machines** and **West**, matching the dimensions that define the selected square:

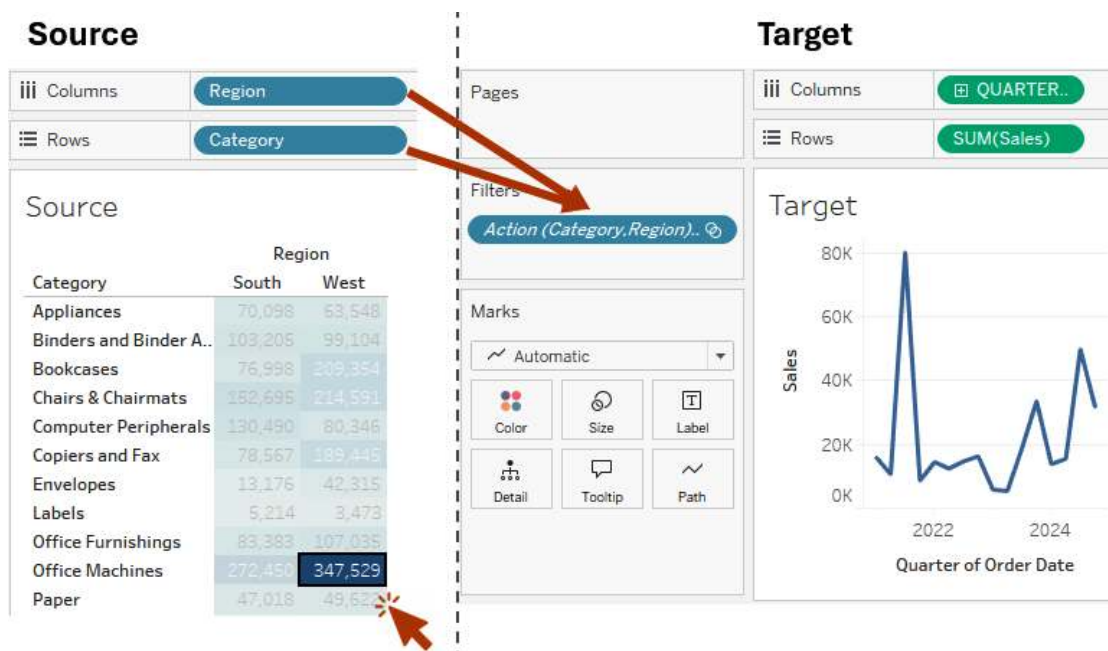


Figure 8.24: Clicking the square for the intersection of Office Machines and West passes both dimensional values as a single action filter to the target

By default, all dimensions present in the source view are used in a filter action. Optionally, you can specify which fields should be used. You can use the **Selected Fields** option in the **Edit Filter Actions** dialog to accomplish the following:

- Filter based on fewer dimensions. For example, if you only selected the **Region** field, then selecting the preceding square would only pass the West region as a filter to the target.
- Filter a target view using a different data source. The **Selected Fields** option allows you to map the source field to a target field (even if the target field has a different name, though the values must still match). For example, if the target used a data source where **East** was a possible value for a field named **Area**, you could map **Region** from the source to **Area** in the target.



The concept of Source and Target sheets also applies to Highlight and Go to Sheet actions, though neither of those action types apply a filter to the target.

Filter actions are very useful for narrowing focus, drilling into details, and seeing different aspects of a selected dimension. You'll find yourself using them often. Let's consider another type of action: highlight actions.

Highlight actions

This type of action does not filter target sheets. Instead, **highlight actions** cause marks that are defined, at least in part, by the selected dimensional value(s) to be highlighted in the target sheets. The options for highlight actions are very similar to filter actions, with the same configuration options for source and target sheets, and which events are able to trigger the action.

Consider a dashboard with three views and a highlight action based on the **Region** field. When the action is triggered for the **East** region, all marks defined by **East** are highlighted. The dimension(s) that are used for the highlight must be present in all views where you want the highlighting to be applied. Both the map and scatter plot have **Region** on the **Detail** part of the **Marks** card:

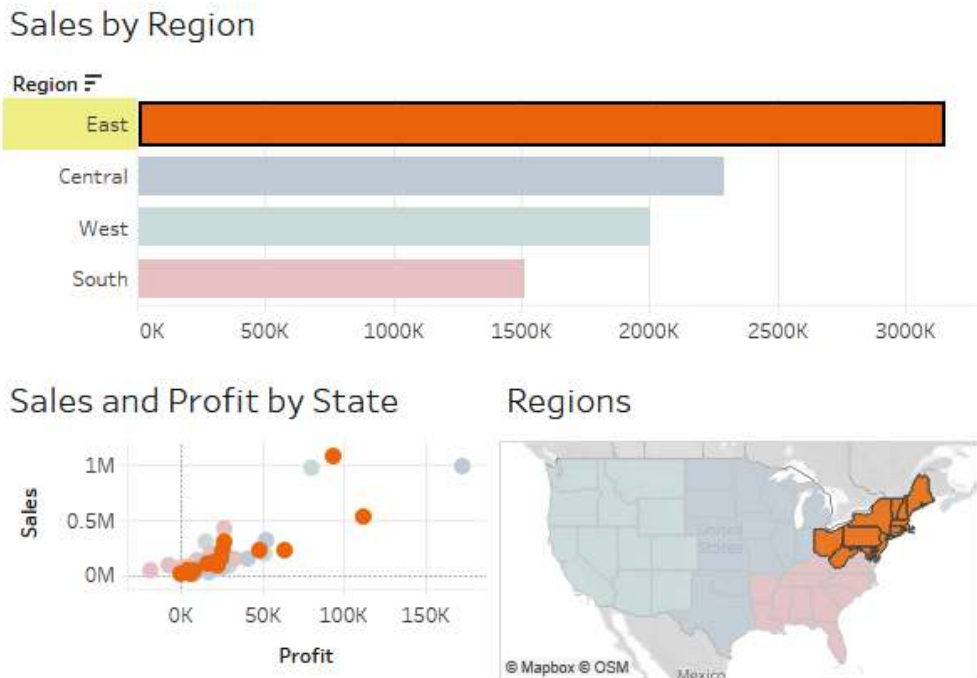


Figure 8.25: Clicking the bar for East has highlighted all other marks associated with that dimensional value

Highlighting is an effective way of showing part of the data in relation to the whole. Whereas filtering would eliminate anything that didn't match a given value, highlighting shows what matches in the context of everything else.

Highlighters (also called **data highlighters**) are shown as user controls (similar to filters and parameters) that cause highlighting based on user interaction. They can be applied to one or more views and will highlight the marks of the views. They do not create an action. To add highlighters, select any discrete (blue) field in the view and use the drop-down menu to click **Show Highlighter**. Alternatively, you can use the menu and select **Analysis | Highlighters**. On a dashboard, you can add a highlighter by using a view's drop-down menu and selecting **Highlighters**.

URL actions

URL actions allow you to dynamically generate a URL based on an action and open it within a web object in the dashboard or in a new browser window or tab. URL actions can be triggered by the same hover, menu, and select events as other actions. The name of the URL action differentiates it and will appear as the link when used as a menu.

The URL includes any hardcoded values you enter as well as placeholders that are accessible via the arrow to the right of the URL text box. These placeholders include fields and parameters. The values will be dynamically inserted into the URL string when the action is triggered based on the values for the fields that make up the selected mark(s) and current values for parameters.

If you have included a web object in the dashboard, the URL action will automatically use that as the target. Otherwise, the action opens a new browser window (using your default browser, when the dashboard is viewed on the desktop or the reader) or a new tab (when the dashboard is viewed in a web browser).

Some web pages have different behaviors when viewed in iframes. The browser object does not use iframes in Tableau Desktop or Tableau Reader but does when the dashboard is published to Tableau Server, Tableau Online, or Tableau Public. You will want to test URL actions based on how your dashboards will be viewed by your audience.

You may specify a target for the URL action when you create or edit the URL action:

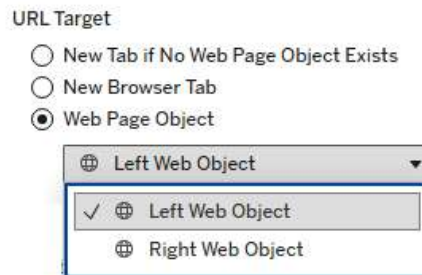


Figure 8.26: Options for a URL action

 **Quick tip:** Need to see a high-resolution version of this image? Open this book in the next-gen Packt Reader or view it in the PDF/ePub copy.

 **The next-gen Packt Reader and a free PDF/ePub copy** of this book are included with your purchase. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



Options include **New Browser Tab**, **Web Page Object** (you may select which object if you have more than one on the dashboard), and **New Tab if No Web Page Object Exists**. If you have more than one web page object on the dashboard, you may wish to give them meaningful names to make selection easier.

To accomplish rename an object on the dashboard, do one of the following:

- Use the dropdown on the dashboard object menu to select **Rename Dashboard Item....**
- Switch to the **Layout** tab on the left-hand pane and expand **Item hierarchy** until you locate the objects you wish to rename. Right-click the object and select **Rename Dashboard Item....**:

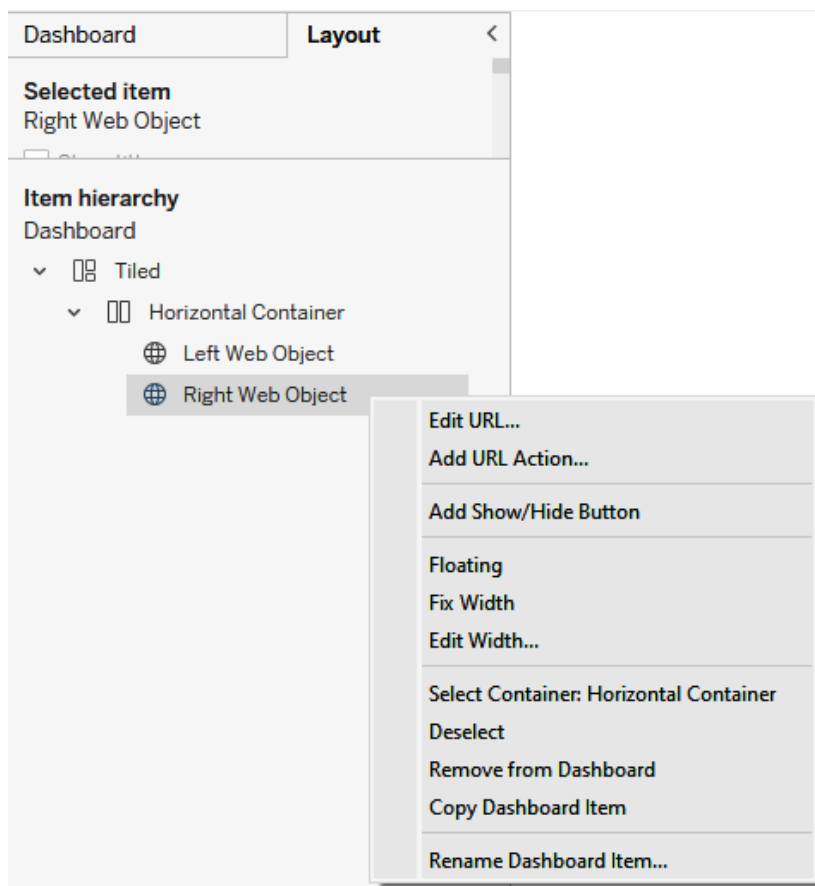


Figure 8.27: Dashboard objects can be renamed using the item hierarchy and the right-click context menu

After you have renamed the object, you will be able to more readily identify which one is the target of the URL action.

Next, we'll consider another type of action that accomplishes a specific result: navigation.

Go to Sheet actions

Go to Sheet actions (also called navigation actions) allow you to specify navigation to another dashboard or sheet in the workbook based on user interaction with one or more views. For example, clicking a bar in a bar chart might navigate to another dashboard. These are similar to filter actions that define another sheet as a target, but Go to Sheet actions do not set any filters:

Add Go to Sheet Action [X]

Name
GoToSheet1

Source Sheets

[+] Is Least Profitable Always Unprofitabl... ▼

- ☒ Overall Profit by Category
- ☒ Profit by State
- ☒ Profit Trend
- ☒ Top 10 Least Profitable Items

Run action on

☐ Hover

☒ Select

☐ Menu

☐ Single-select only

Target Sheets

[+] Profit Ratio Analysis ▼

Cancel OK

Figure 8.28: An example of a Go to Sheet action

Parameter actions

Parameter actions allow you to set the value of a parameter based on a user action. When you create a parameter action, you'll set options using a screen like this:

Edit Parameter Action

Name
Parameter1 Insert ▾

Source Sheets

- ☐ Parameter Example
- ☐ Profit over Time
- ☒ State Map

Run action on

- ☐ Hover
- ☒ Select
- ☐ Menu

Target Parameter
Abc Select a State ▾

Clearing the selection will

- ☐ Keep current value
- ☒ Set value to

No State Selected

Source Field Aggregation

Abc State (Sup... ▾ None ▾

Cancel OK

Figure 8.29: Options for a parameter action

As with other actions, you may select the sheets and user interactions (**Hover**, **Select**, **Menu**) that trigger the action. Additionally, you will set the target parameter and specify which field will set the value. You may use any field in the sheet and may also specify an aggregation for the field. You also have the option of keeping the filter when deselecting the mark that triggered the action or setting the parameter to a new value. This gives you options for reverting to a default value or clearing the value as best meets your needs.

We saw in *Chapter 4, Starting an Adventure with Calculations*, how parameters can be used to accomplish all kinds of results. You can use them in calculations, filters, and **Top N** sets, and you can use them to define reference lines and alter the visual aspects of any view. This opens a world of possibilities for driving an interactive experience using views and triggers.

Set actions

Set actions allow you to populate a set with one or more values for one or more dimensions. This is very powerful because sets can be used on any shelf to encode any visual element, can be used in calculations, and can be used as filters. They can be used in all of these ways—and in different ways—in different views. This gives you incredible flexibility in what can be accomplished with set actions.



You'll recall discussing sets in *Chapter 2, Connecting to Data in Tableau*. There we saw that you can create fixed or computed sets. As we'll see in the following example, set actions operate on fixed sets.

There are many things you can do with sets! Here, we'll explore an example of a bar chart representing sales by category. If you were to filter the bar chart by a given region, you would notice the lengths of the bars change. However, if instead of filtering, you only highlighted the segments of the bar that represented that region, you'd gain some insight into the overall context. This technique is known as **proportional brushing**.

You'll find this example of a set action in the *Chapter 08 Complete.twbx* workbook in the dashboard named *Sales by Region and Category (set actions)*, which looks like this:

Sales by Region and Category (set actions)

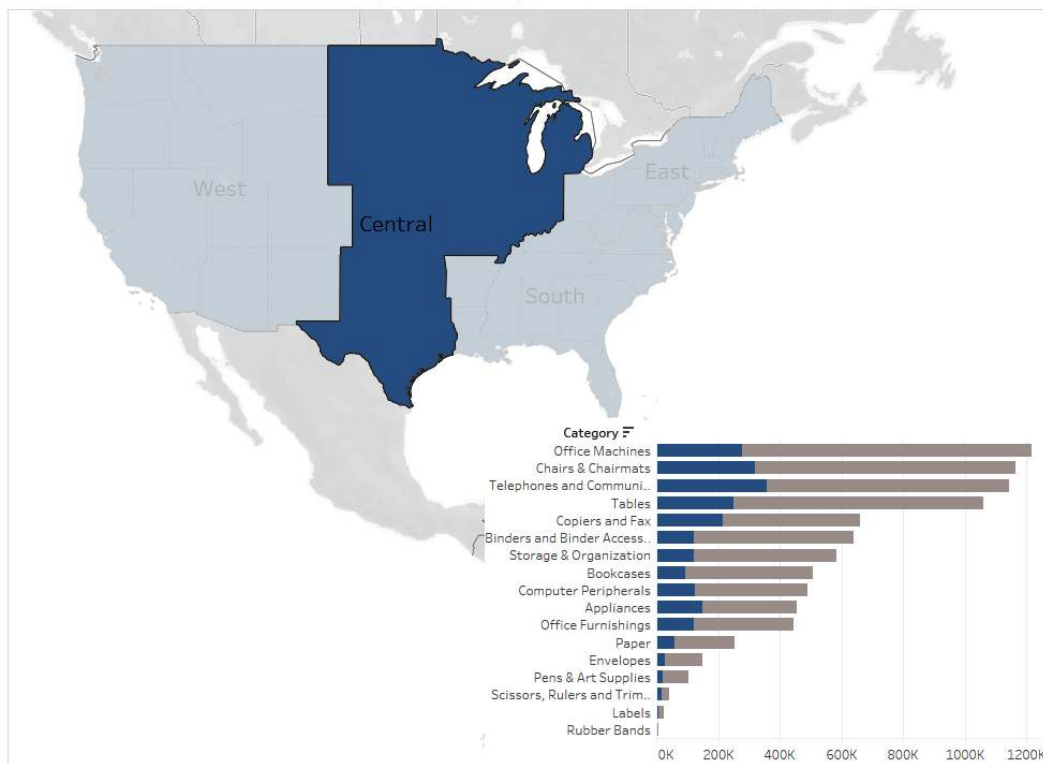


Figure 8.30: The set action allows brushing: highlighting the portion of the bars that belong to the selection

The dashboard consists of two views: a map and a bar chart. Selecting a region on the map triggers a set action that updates the bar chart. A filter action would filter the bar chart, causing the length of each bar to only show the value for the selected region.

Here, however, the set action is used to show the portion of the overall bar that belongs to that region while still retaining the full length of the bar for all regions.

To replicate this interactivity, follow these steps:

1. Use the drop-down menu on the **Region** field under **Dimensions** on the data pane to select **Create | Set**. Name the set **Region Set**.
2. In the resulting **Create Set** dialog, on the **General** tab, check one or more values. This creates a fixed set. In this example, it does not matter which, if any, values you select because you'll configure the set action to update the values momentarily.
3. Create a bar chart of **Sales by Category**.
4. Drag **Region Set** from the data pane and drop it on **Color**:

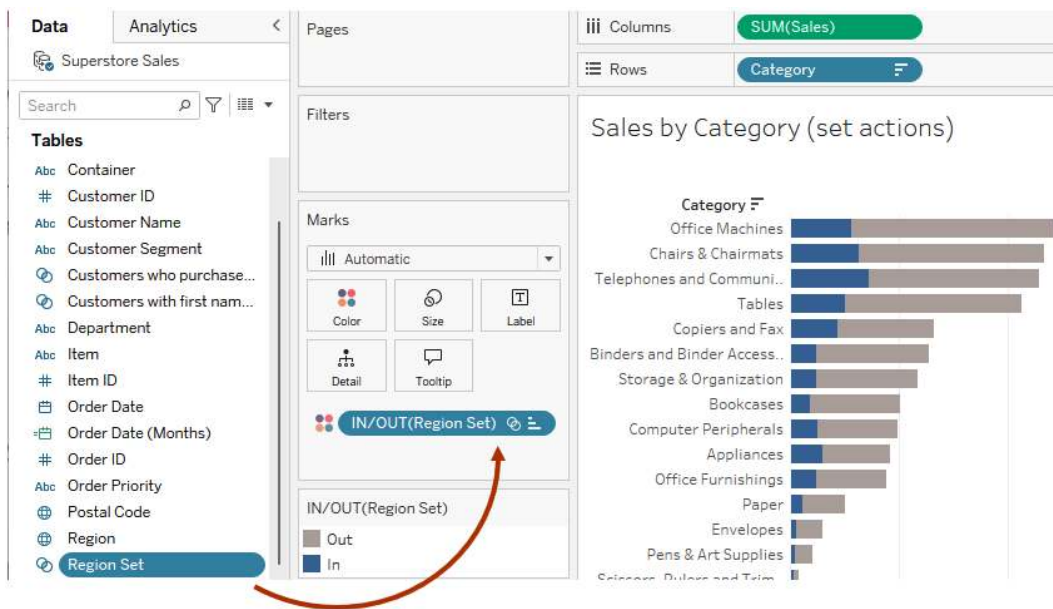


Figure 8.31: Drag and drop the set on Color to show the difference between In and Out of the set

You may use a set on shelves, just as you would any other field. Notice how the set gives two values, that is, **In** and **Out**, which define two colors. You may wish to adjust the colors to emphasize **In**.

You may use sets in calculated fields as well. For example, the code `[Region Set]` gives a Boolean true/false result for each record, indicating whether it is in the set.

Conclude the set action example by creating a region map, the dashboard, and implementing the set action:

1. Use the drop-down menu on **Region** to select **Geographic Role | Create from... | State**. This tells Tableau to treat **Region** as a geographic field based on its relationship with the geographic field **State**.

2. In a new, blank view, double-click the **Region** field to create the map view. Now that **Region** is recognized as a geographic field, Tableau will generate latitude, longitude, and the geometries that are necessary to render the shapes.
3. Add both the map and bar chart views to a single dashboard. You may position them however you'd like.
4. Add a set action by selecting **Dashboard | Actions** from the menu and then **Add Action | Change Set Values...** in the resulting dialog. The resulting dialog has many similar features to other action types:

Figure 8.32: Options for the set action

You'll notice options to give the action values for **Name**; **Run action on: Hover, Select, or Menu**; and options for **Clearing the selection**. Just like other action types, you may also specify **Source** sheets that trigger the action. **Target Set** allows you to specify which data source and which fixed set in that data source will have values updated based on the action. In this case, we want to update **Region Set** when a selection is made on the **Sales by Region (set actions)** view. We'll elect to remove all values from the set when the selection is cleared.

Once you have created the preceding action, your dashboard should function very similarly to the example that was shown at the beginning of this section. Selecting a region on the map highlights the portion of the bars that correspond to that region. This technique is known as **brushing** or **proportional brushing**.

This technique is only one of the hundreds of possible applications of set actions. Since sets can be used on any shelf and in calculations, updating the values via set actions gives you endless possibilities for user interaction and analytics.

Having looked at the various kinds of actions, let’s move on to discuss another example of a different kind of dashboard.

A scorecard dashboard

Now, we’ll consider another example dashboard that demonstrates slightly different objectives. Let’s say everyone in the organization has agreed upon a KPI of profit ratio. Furthermore, there is consensus that the cut-off point between an acceptable and poor profit ratio is 15%, but management would like to have the option to adjust the value dynamically to see whether other targets might be better.



We’ll discuss Tableau Pulse and Metrics as an alternate way to make KPIs widely accessible to decision-makers across the organization in *Chapter 17, AI-Assisted Analysis: Tableau Agent and Tableau Pulse*.

Consider the following dashboard:

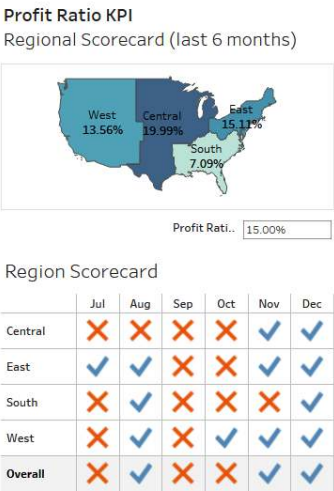


Figure 8.33: A simple profit KPI dashboard

This dashboard allows your audience to very quickly evaluate the performance of each region over the last six months. Executive management could very quickly pull this dashboard up on their mobile devices and take appropriate action as needed.

The dashboard provides interactivity with the KPI target parameter. Additional drill-down capability into other dashboards or views could be provided if desired. If this view were published on Tableau Server, it is not unreasonable to think that regional managers might subscribe to the view and receive a scheduled email containing an up-to-date image of this dashboard.

Let's consider how to create a similar dashboard:

1. Create a float type parameter named **Profit Ratio KPI Target** set to an initial .15, formatted as a percentage.
2. Create a calculation named **Profit Ratio** with the code `SUM([Profit]) / SUM([Sales])`. This is an aggregate calculation that will divide the profit total by the sum of sales at the level of detail defined in the view.
3. Create a second calculation named **KPI - Profit Ratio** with the following code:

```
IF [Profit Ratio] >= [Profit Ratio KPI Target]
THEN "Acceptable"
ELSE "Poor"
END
```

This code will compare the profit ratio to the parameterized cut-off value. Anything equal to or above the cut-off point will get a value of **Acceptable**, and everything below will get the value of **Poor**.

4. Create a new sheet named **Region Scorecard**. The view consists of **Region** on **Rows**, **Order Date** as a discrete date part on **Columns**, and the **KPI - Profit Ratio** field on both shape and color. You'll observe that the shapes have been edited to use checkmarks and Xs, and that the color palette is using colorblind-safe blue and orange.
5. Add column grand totals using the **Analytics** pane and format the grand totals with a custom label of **Overall**, with bold font and light gray shading.
6. Add **Order Date** as a filter and set it to the top 6 by field (**Order Date as Min**). This will dynamically filter the view to the last six months:

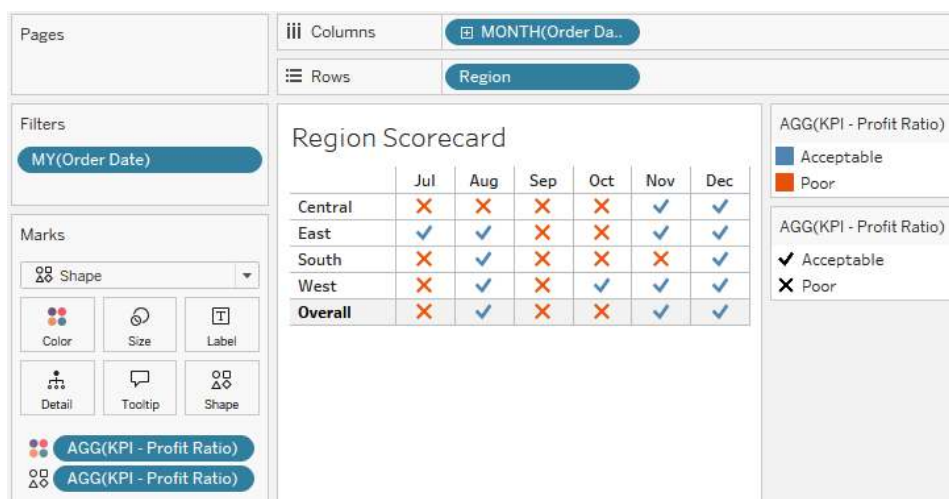


Figure 8.34: This view defines the Region scorecard showing Acceptable versus Poor results per region per month

7. Create another sheet named **Profit Ratio by Region**.
8. If you skipped the set actions example, use the drop-down menu on **Region** to select **Geographic Role | Create from... | State**. This tells Tableau to treat **Region** as a geographic field based on its relationship with the geographic field **State**.
9. Double-click the **Region** field in the data pane. Tableau will automatically generate a geographic visualization based on **Region**. We'll examine the creation of custom geographies in more detail in *Chapter 12, Exploring Mapping and Advanced Geospatial Features*.
10. Place **Profit Ratio** on **Color** and **Label**. You will also want to format **Profit Ratio** as a percentage. You may do so by formatting the field in this view specifically, or by setting the default number format for the field in the data pane (the latter is probably preferred as you will almost always want it to display as a percentage).
11. Additionally, add **Region** to **Label**. Rearrange the fields in the **Marks** card to reorder the label or click the **Label** shelf to edit the label text directly.
12. Apply the same filter to this view as you did to the **Region Scorecard** view. You may wish to navigate to the **Region Scorecard** sheet and use the dropdown on **Order Date** on the **Filters** shelf to apply the existing filter to multiple sheets:

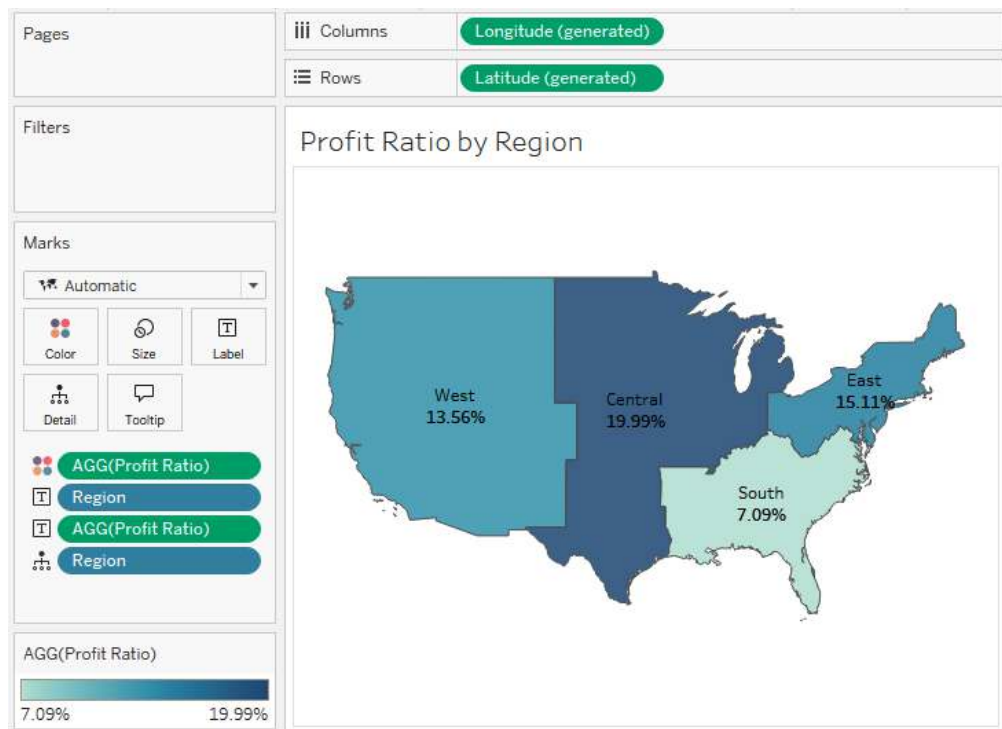


Figure 8.35: The filled map shows profit by region

13. Once both views have been created, you can arrange the views as a dashboard. The example in the Chapter 08 Complete workbook has a phone layout applied to it, as shown here:

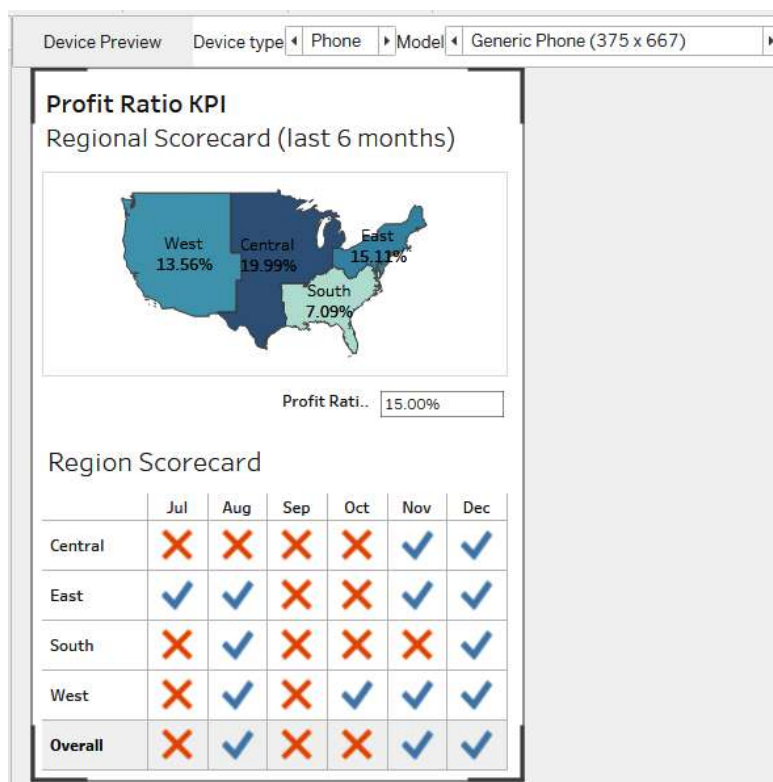


Figure 8.36: A phone layout for the KPI dashboard

Experiment with various layouts and positioning of the elements.



By default, all objects that are added to the dashboard are tiled. **Tiled** objects snap in place and appear beneath **floating** objects. Any object can be added to the dashboard as a floating object by switching the toggle under **New Objects** in the left window, or by holding Shift while dragging the objects to the dashboard.

Existing objects can be switched between floating and tiled by holding Shift while moving the object or using the drop-down caret menu. The drop-down caret menu also gives you options to adjust the floating order of objects. Additionally, floating objects can be resized and positioned with pixel precision by selecting the floating object and using the positioning and sizing controls in the lower left.

You can mix tiled and floating elements, but many dashboard authors prefer to build dashboards that are composed entirely of one or the other. This ensures consistency between different layouts and sizes of screens (especially if the dashboard is set to an **Automatic** or **Range** sizing option).

This example illustrates a dashboard used for quick lookup and rapid decision-making. What if we want to tell a more complex story, show the progression of detail, and maybe even present the data story in a specific order? Let's see how to use a feature of Tableau designed specifically for that purpose.

Stories

The **stories** feature allows you to tell a story using interactive snapshots of dashboards and views. The snapshots become points in a story. This allows you to construct a guided narrative or even an entire presentation.

Let's consider an example in which story points might be useful. The executive managers are pleased with the **Regional Scorecard** dashboard you developed previously. Now, they want you to make a presentation to the board and highlight some specific issues for the South region. With minimal effort, you can take your simple scorecard, add a few additional views, and tell an entire story:

1. First, we'll build a couple of additional views. Create a simple geographic view named **ProfitRatio KPI by State**. Make this a filled map with the **KPI – Profit Ratio** field, defining **Color**.
2. Add **Profit Ratio** to the **Detail** part of the **Marks** card so that it is available for later use:

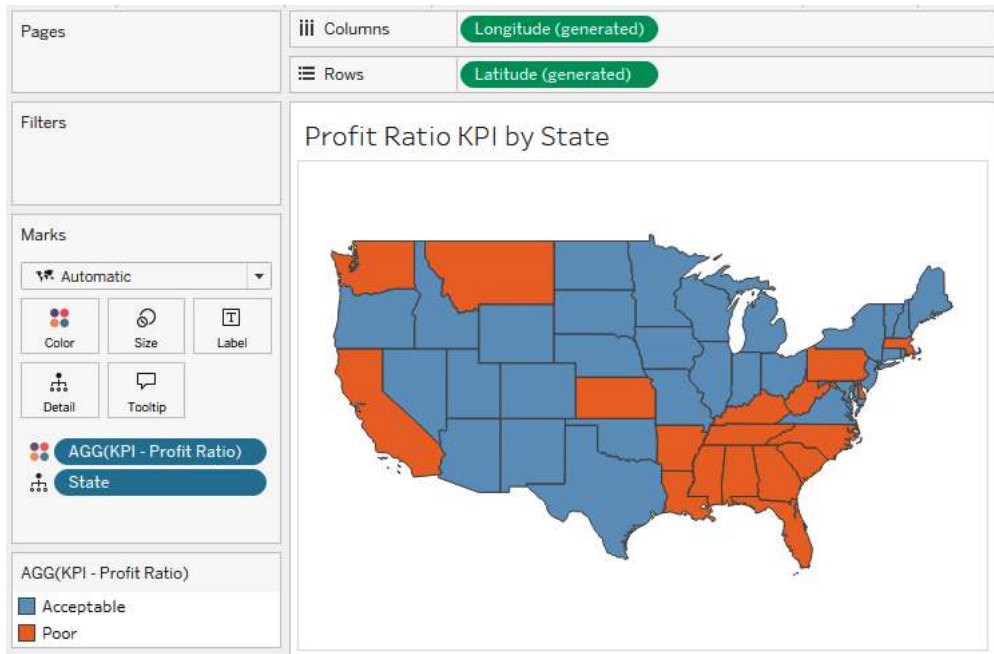


Figure 8.37: Profit Ratio KPI by State uses color encoding to show Acceptable versus Poor

3. Create one additional view named **Profit Ratio by Quarter**. Use **Order Date** as a continuous date value on **Columns** and **Profit Ratio** on **Rows**.

4. Set the mark type to bars. Add a reference line for the **Profit Ratio KPI Target** parameter value (you can right-click the **Profit Ratio** axis and set it to **Add Reference Line...**).
5. Add **KPI – Profit Ratio** to **Color**. You may also wish to click the **Color** shelf and add a border.
6. Go ahead and filter the view to the South region and use the drop-down menu to apply that filter to the **Profit Ratio KPI by State** view as well:

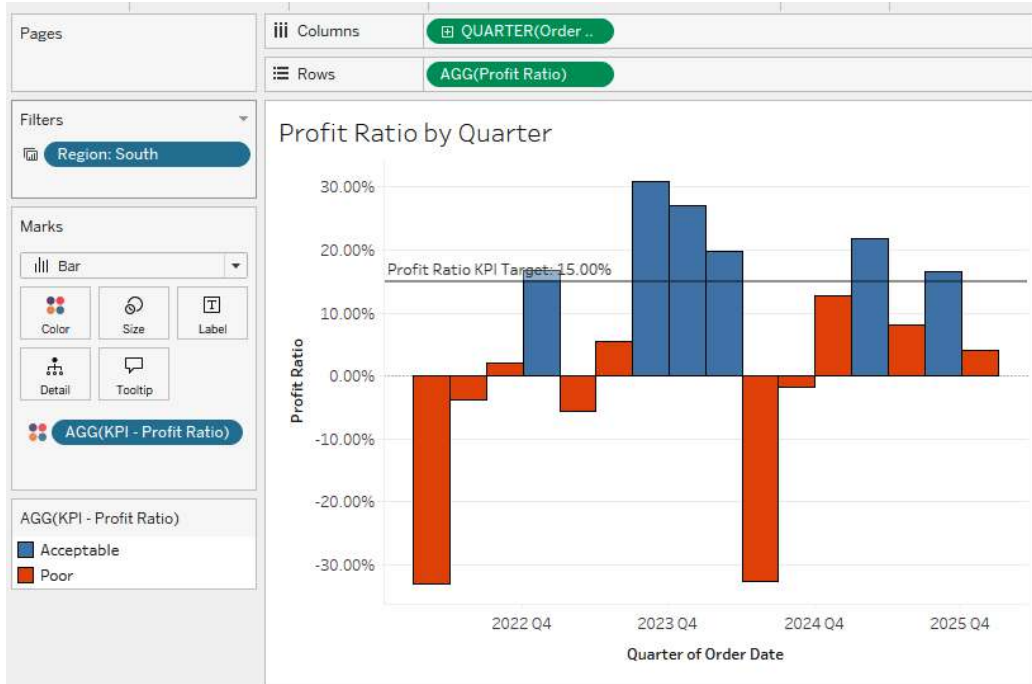


Figure 8.38: Profit Ratio by Quarter shows whether a given quarter was Acceptable or Poor based on the target

7. Create a new dashboard with the two new views arranged in the same way as what's shown in the following screenshot. Add the **Profit Ratio KPI Target** parameter and **Region** filter if they do not show.

8. Use the dropdown on **Profit Ratio KPI by State** to use that view as a filter:

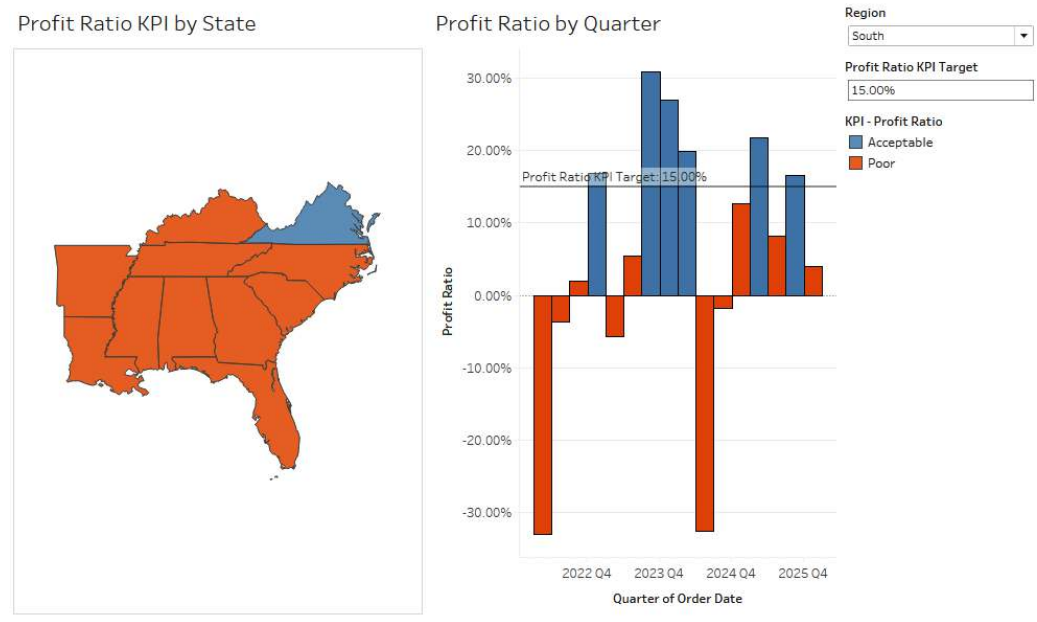


Figure 8.39: The Profit Ratio KPI by State and by Quarter views on the same dashboard

9. Create a new story by selecting **Story | New Story** from the menu, or by using the new story tab at the bottom next to the existing sheets:



The **Story** interface consists of a sidebar with all visible dashboards and views. At the top, you'll see the story title, which can be edited via a double-click. Each new point in the story will appear as a navigation box with text that can also be edited. Clicking on the box will give you access to the story point, where you can then add a single dashboard or view.

will present you with an option to update the existing story point to the current state of the dashboard. Alternatively, you can use the **Revert** button above the navigation box to return to the original state of the dashboard. Clicking **X** will remove the story point.

Each story point contains an entirely independent snapshot of a dashboard. Filter selections, parameter values, selections, and annotations will be remembered for a particular story point but will have no impact on other story points or any other sheet in the dashboard.

You may rearrange story points by dragging and dropping the navigation boxes.

1. We'll build the story by completing the following steps:
 1. Give the story the title **South Region Analysis**.
 2. Add the **Regional Scorecard** dashboard as the first story point. Select the **South** region on the map. Give the story point the following text: The South Region has not performed well the last 6 months:

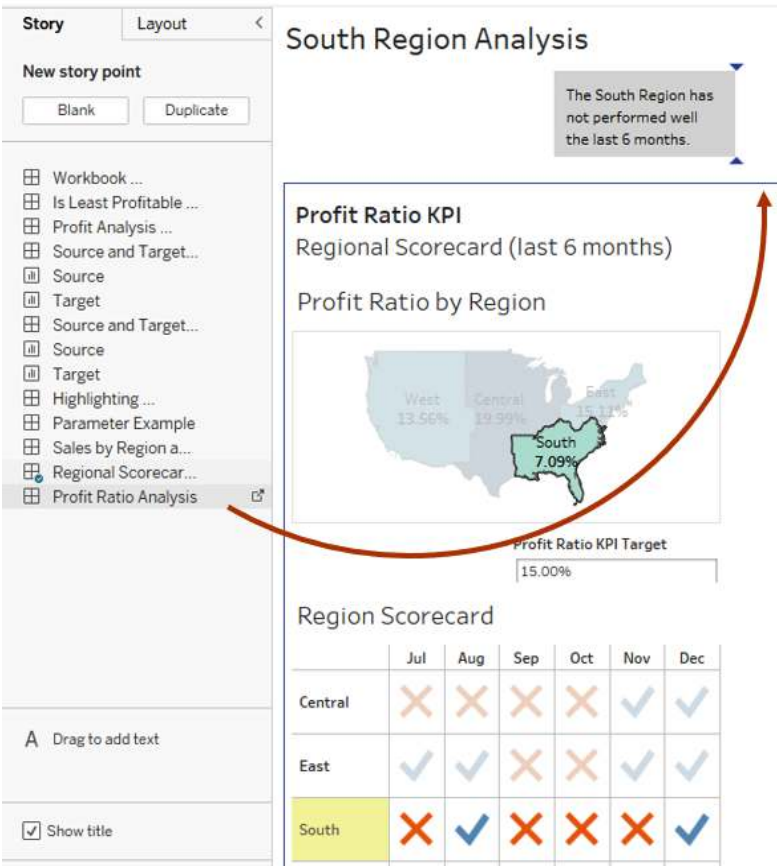


Figure 8.42: The first story point highlights the performance of the South region. A second story point can be created by dragging a dashboard into the navigator or by using the **Blank** or **Duplicate** buttons on the left sidebar

3. Drag the Profit Ratio Analysis dashboard into the story navigator as illustrated in Figure 8.41 to create a new story point.
4. Give this story point a caption of **Only one state has met the 15% target overall**.
5. Right-click **Virginia** on the map and select **Annotate | Mark**. Keep the state and profit ratio as part of the annotation:

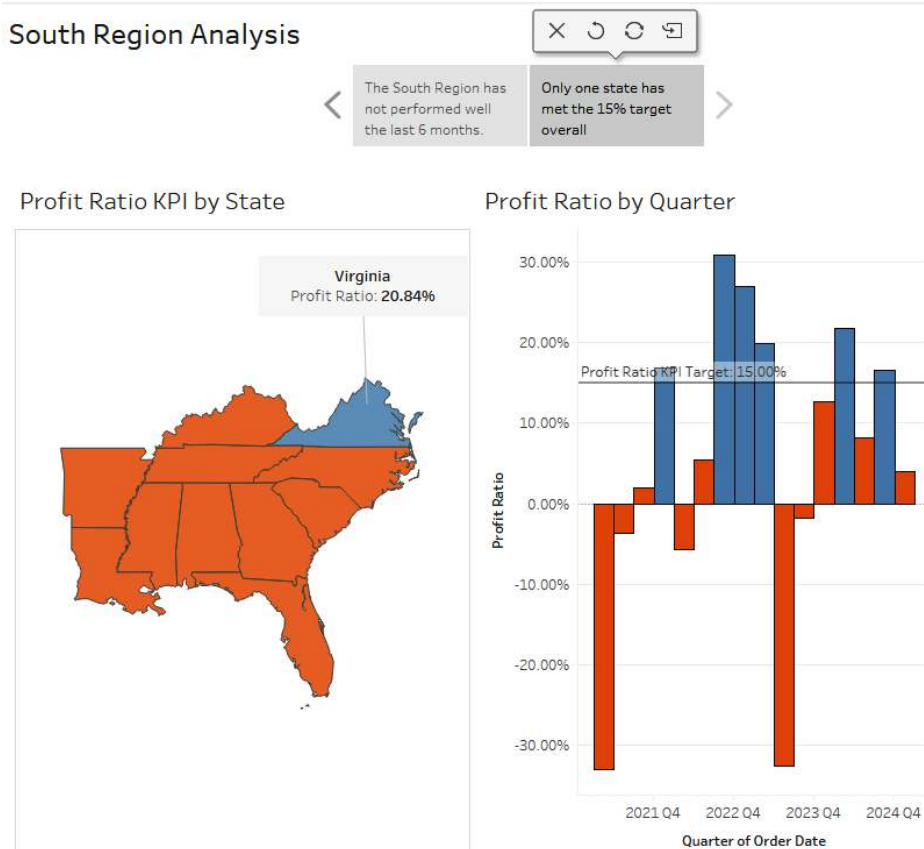


Figure 8.43: The second story point dives into the details

6. Click the **Duplicate** button on the left sidebar to copy the current story point. Give this new story point a caption of **3 states would meet a goal of 10%**. Set **Profit Ratio KPI Target** to **10.00%** and update the point.
7. Click the **Duplicate** button again and give the newly created point a caption of **Certain states have performed well historically**.
8. Right-click the annotation for Virginia, select **Remove** to delete it, and then add a similar annotation for **Louisiana**. Then, click **Louisiana** to select that state.
9. Make sure to click the **Update** button to capture the state of the dashboard.

In presentation mode, the buttons for adding, duplicating, updating, or removing story points are not shown.

Your final story should look similar to this:

South Region Analysis

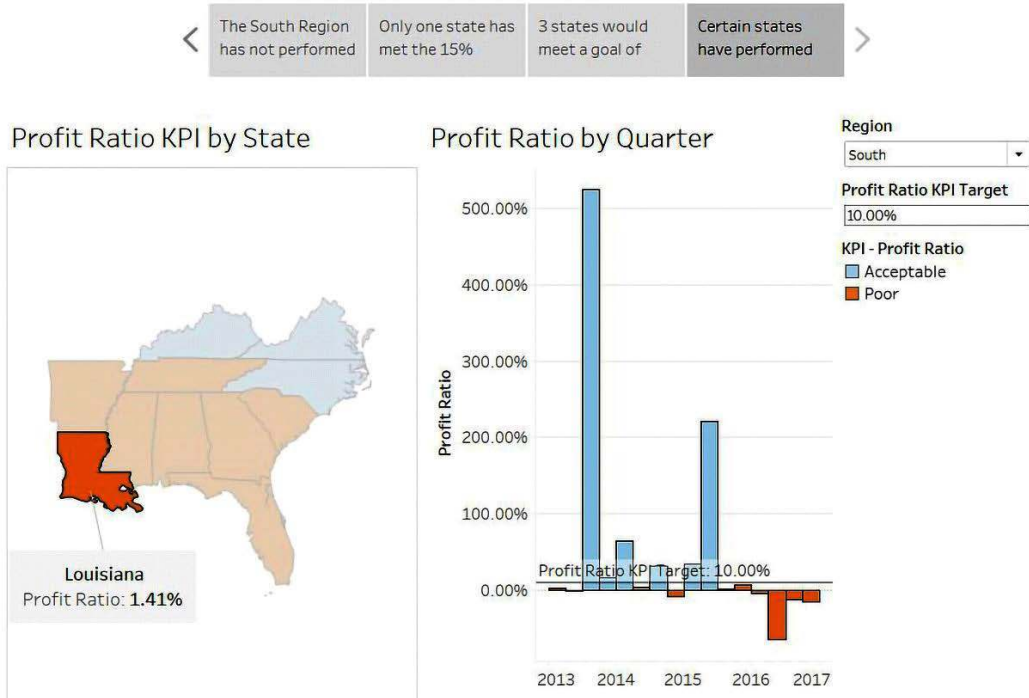


Figure 8.44: This story point highlights quarterly results for a single state

Take some time to walk through the presentation. Clicking navigation boxes will show that story point. You can fully interact with the dashboard on each story point. In this way, you can answer questions on the fly and dig into details, and then continue through the story.

A great way to learn about dashboard techniques (and data visualization techniques in general) is to subscribe to *Viz of the Day* (<http://www.tableau.com/public/community/viz-of-the-day>).

A new visualization, dashboard, or story is featured each day. When you see a design technique or visualization you want to understand, you can download the workbook and explore the various techniques that were used.

Stories are a great way to call out specific insights or action points in the overall data story.

Summary

When you are ready to share your discovery and analysis, you will likely use dashboards to relate the story to your audience. The way in which you tell the story will depend on your objectives, as well as your audience and the mode of delivery. Using a combination of views, objects, parameters, filters, and legends, you can create an incredible framework for telling a data story. Tableau allows you to specifically design layouts for different devices to ensure that your audience has the best experience possible. By introducing actions and interactivity, you can invite your audience to participate in the story. Story points will allow you to bring together many snapshots of dashboards and views to craft and present entire narratives.

In the next chapter, we'll turn our attention to some deeper analysis involving trends, distributions, forecasting, and clustering.

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search for this book by name.



Note: Keep your purchase invoice ready before you start.

9

Visual Analytics: Trends, Clustering, Distributions, and Forecasting

The rapid visual analysis that is possible using Tableau is incredibly useful for answering numerous questions and making key decisions, but it barely scratches the surface of the possible analytics. For example, a simple scatterplot can reveal outliers, but sometimes you want to understand the distribution or identify clusters of similar observations. A simple time series helps you to see the rise and fall of a measure over time, but sometimes you want to see the trend or make predictions for future values.

Tableau enables you to quickly enhance your data visualizations with statistical analysis. Built-in features such as trend models, clustering, distributions, and forecasting allow you to quickly add value to your visual analysis. Additionally, Tableau integrates with the R and Python platforms, which opens endless options for parsing and analyzing your data. We'll look more at these in *Chapter 16, Extending Tableau: Extensions and Scripts*.

This chapter will cover the built-in statistical models and analysis, including the following topics:

- Trends
- Data Guide
- Clustering
- Distributions
- Forecasting

We'll look at these concepts in the context of a few examples using some sample datasets. You can follow along with and reproduce these examples using the Chapter 9 workbook.

When analyzing data that changes over time, understanding the overall nature of change is vitally important. Seeing and understanding trends is where we'll begin.

Trends

World Population.xlsx is included in the Chapter 09 directory. It contains one record for each country for each year from 1960 to 2015, measuring population. Using this dataset, let's take a look at the historical trends of various countries. Create a view similar to the one shown in the following screenshot, which shows the change in population over time for **Afghanistan** and **Australia**. You'll notice that **Country Name** has been filtered to include only **Afghanistan** and **Australia** and the field has additionally been added to the **Color** and **Label** shelves:

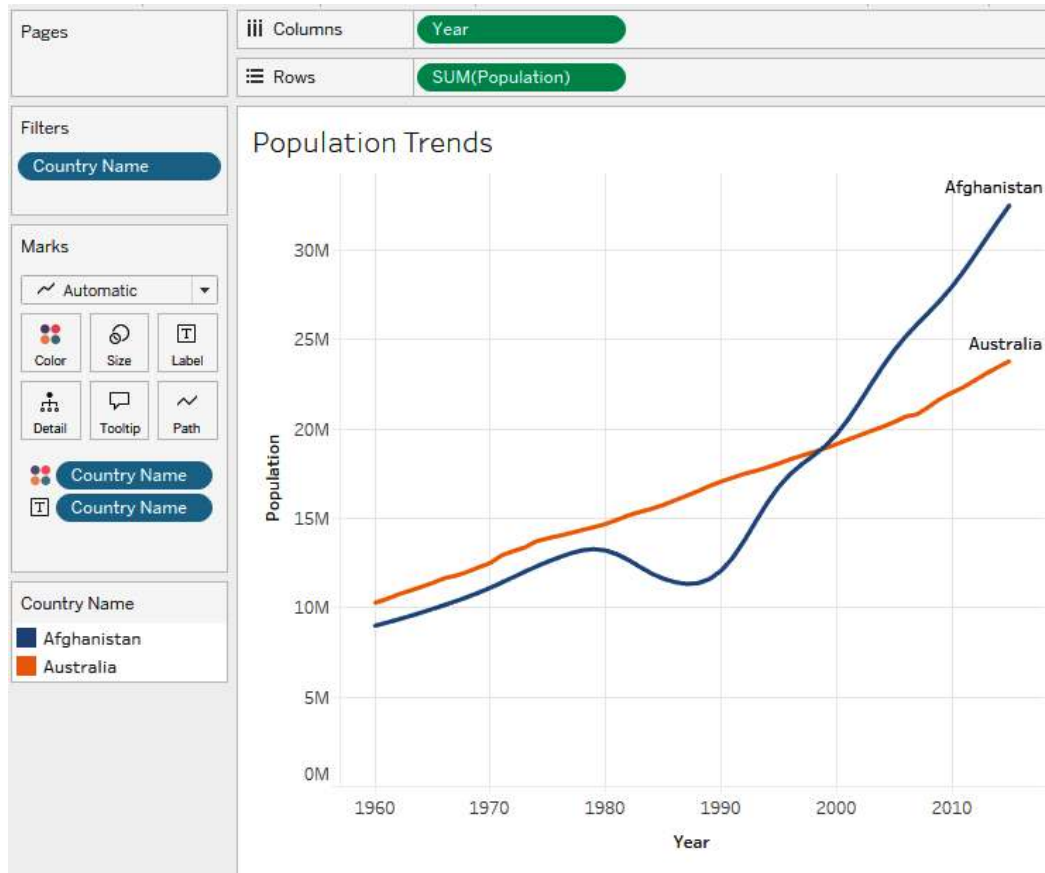


Figure 9.1: Population values for Afghanistan and Australia over time

From this visualization alone, you can make several interesting observations. The growth of the two countries' populations was fairly similar up to 1980. At that point, the population of Afghanistan went into decline until 1988, when it started to increase. At some point around 1996, the population of Afghanistan exceeded that of Australia. The gap has grown wider ever since.

While we have a sense of the two trends, they become even more obvious when we see the trend lines. Tableau offers several ways of adding trend lines:

1. From the menu, select **Analysis | Trend Lines | Show Trend Lines**.
2. Right-click on an empty area in the pane of the view and select **Show Trend Lines**.
3. Click on the **Analytics** tab on the left-hand sidebar to switch to the **Analytics** pane. Drag and drop **Trend Line** on the trend model of your choice (we'll use **Linear** in this example and discuss the others later in this chapter):

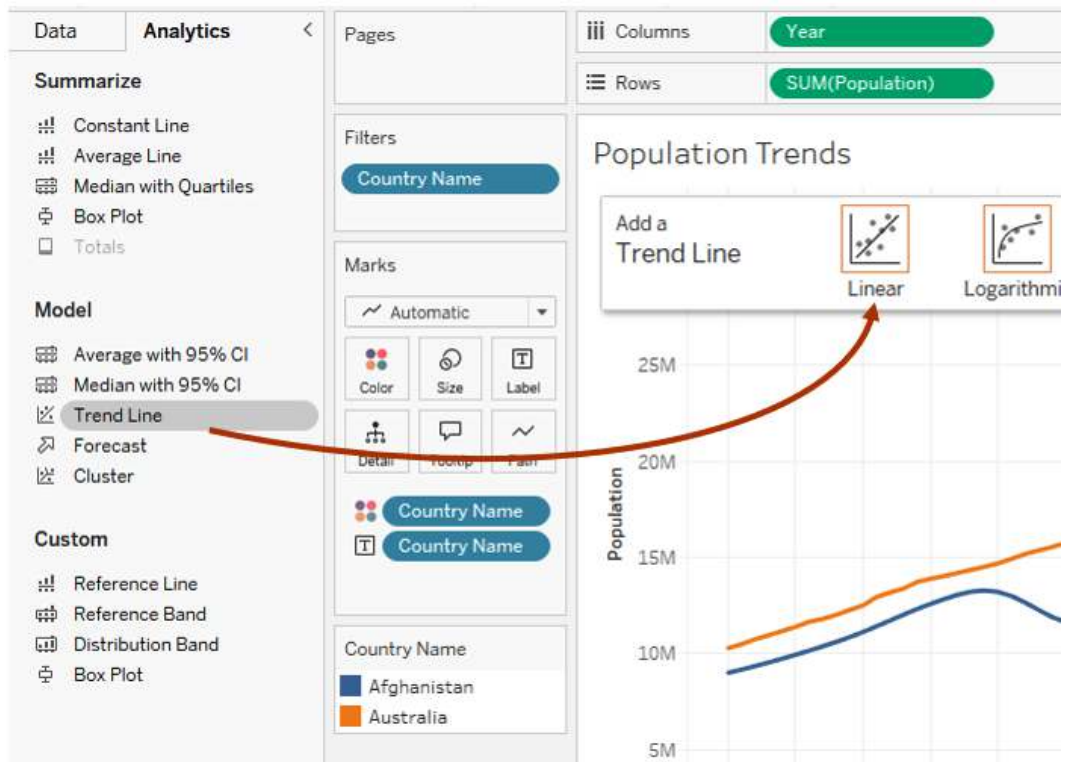


Figure 9.2: Adding a trend line to a view by dragging and dropping from the Analytics pane

Once you've added **Trend Line** to your view, you will see two trend lines: one for each country. We'll look at how we can customize the display shortly. For now, your view should look like this:

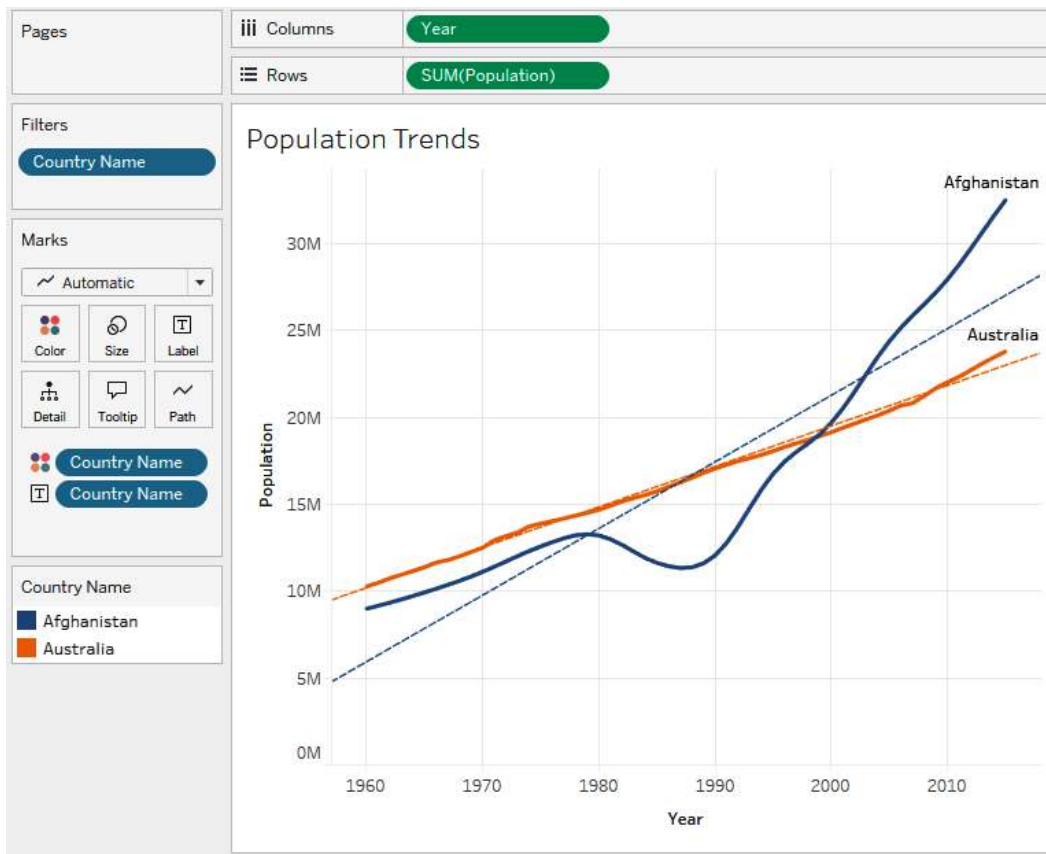


Figure 9.3: Each trend line shows the overall trend for the respective country

Trends are computed by Tableau after the query of the data source and are based on various elements in the view:

- **The two fields that define X and Y coordinates:** The fields on **Rows** and **Columns** that define the x and y axes describe coordinates, allowing Tableau to calculate various trend models. In order to show trend lines, you must use a continuous (green) field on both **Rows** and **Columns**. The only exception to this rule is that one may be a discrete (blue) **date** field. If you use a discrete **date** field to define headers, the other field must be a continuous field.
- **Additional fields that create multiple, distinct trend lines:** Discrete (blue) fields on the **Rows**, **Columns**, or **Color** shelves can be used as factors to split a single trend line into multiple, distinct trend lines.
- **The trend model selected:** We'll examine the differences in models in the next section.



Observe in *Figure 9.3* that there are two trend lines. Since **Country Name** is a discrete (blue) field on **Color**, it defines a trend line per color by default.

Earlier, we observed that the population of Afghanistan increased and decreased within various historical periods. Note that the trend lines are calculated along the entire date range. What if we want to see different trend lines for those time periods?

One option is to simply select the marks in the view for the time period of interest. Tableau will, by default, calculate a trend line for the current selection. Here, for example, the points for Afghanistan from 1980 to 1988 have been selected and a new trend is displayed:

Population Trends

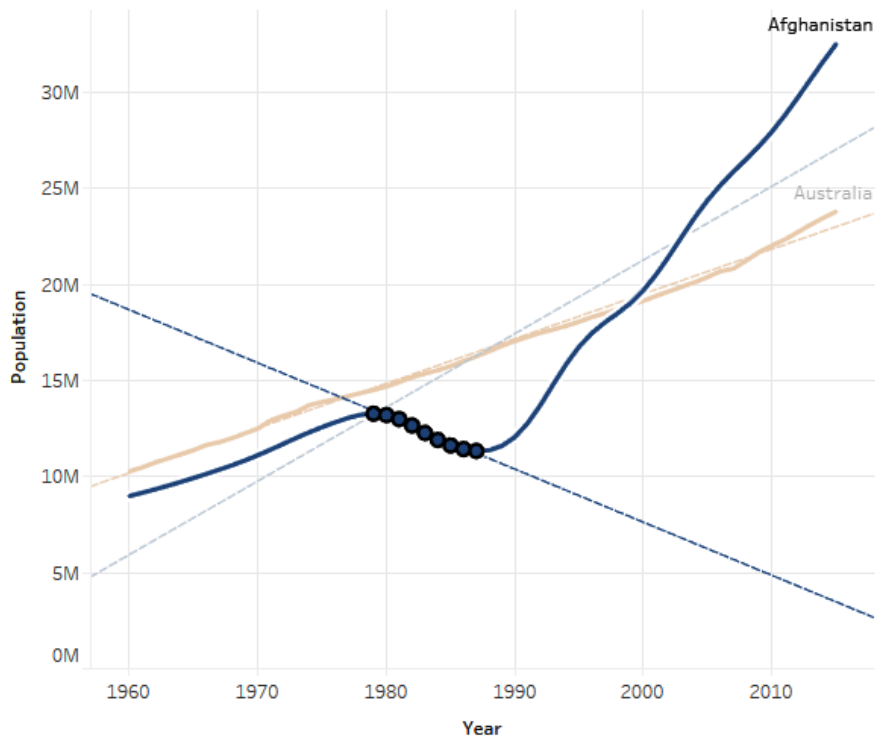


Figure 9.4: The default settings specify that trend lines will be drawn for selections

Another option is to instruct Tableau to draw distinct trend lines using a discrete field on **Rows**, **Columns**, or **Color**.

Go ahead and create a calculated field called **Period** that defines discrete values for the different historical periods using code like this:

```
IF [Year] <= 1979
  THEN "1960 to 1979"
ELSEIF [Year] <= 1988
  THEN "1980 to 1988"
ELSE "1988 to 2015"
END
```

 **Quick tip:** Enhance your coding experience with the **AI Code Explainer** and **Quick Copy** features. Open this book in the next-gen Packt Reader. Click the **Copy** button

(1) to quickly copy code into your coding environment, or click the **Explain** button

(2) to get the AI assistant to explain a block of code to you.



```
function calculate(a, b) {
  return {sum: a + b};
};
```

Copy

1

Explain

2

 **The next-gen Packt Reader** is included for free with the purchase of this book. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



When you place it on Columns, you'll get a header for each time period, which breaks the lines and causes separate trends to be shown for each time period. You'll also observe that Tableau keeps the full date range in the axis for each period. You can set an independent range by right-clicking on one of the date axes, selecting **Edit Axis**, and then checking the option for **Independent axis range for each row or column**:

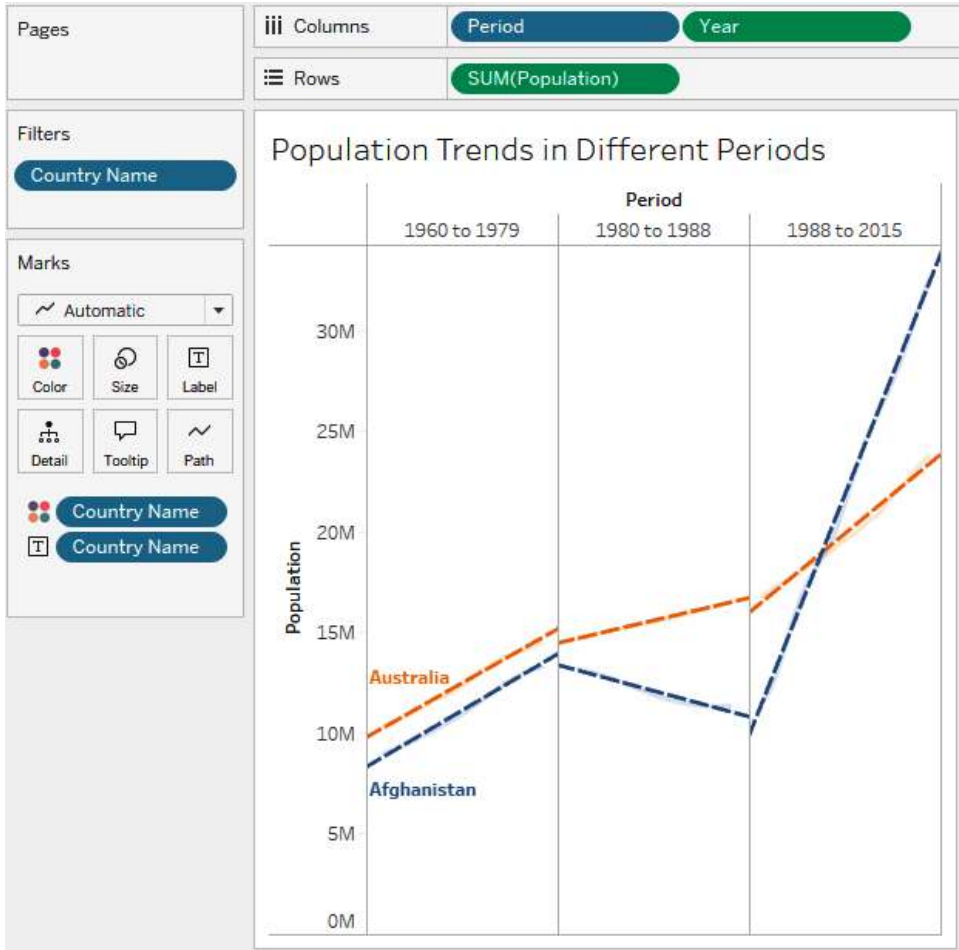


Figure 9.5: Here, the discrete dimension *Period* creates three separate time periods and a trend for each one

In this view, transparency has been applied to the **Color** of the marks to help the trend lines stand out. Additionally, the axis for **Year** was hidden (by unchecking the **Show Header** option on the field). Now, you can clearly see the difference in trends for different periods of time. Australia's trends only slightly change in each period. Afghanistan's trends vary considerably.

With an understanding of how to add trend lines to your visualization, let's dive a bit deeper to understand how to customize the trend lines and model.

Customizing trend lines

Let's examine another example that will allow us to consider various options for trend lines. Using the **Real Estate** data source, create a view like this one:

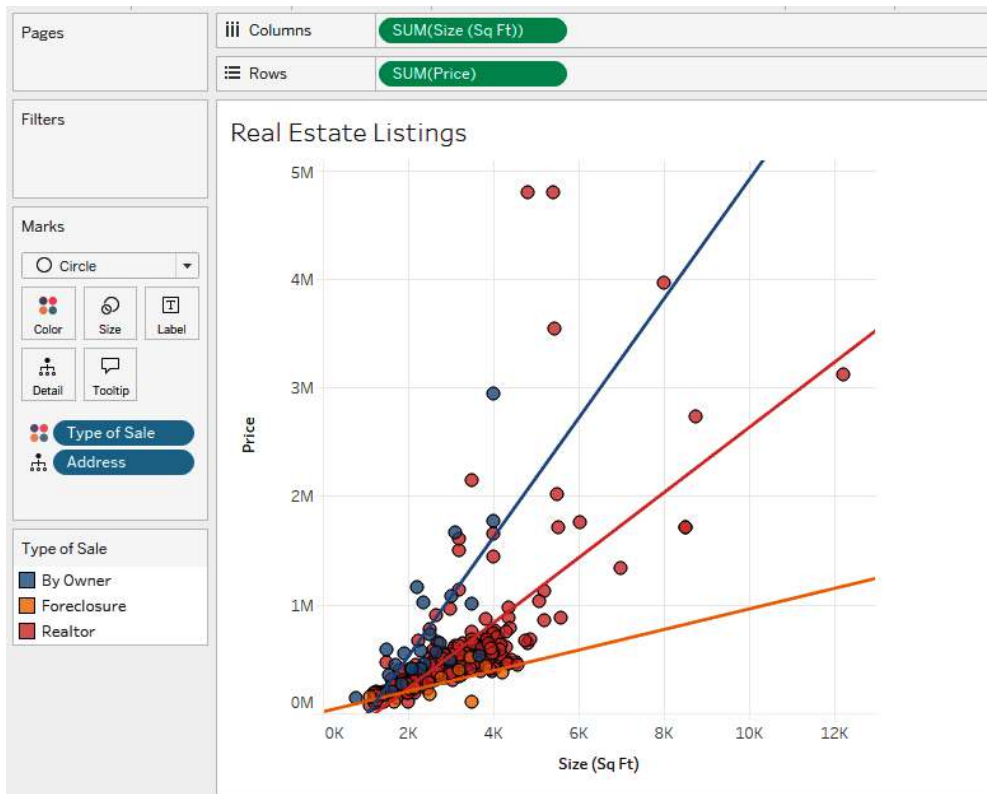


Figure 9.6: Trend lines on a scatterplot are often useful for better understanding correlation and outliers

Here, we've created a scatterplot with the sum of **Size (Sq Ft)** on **Columns** to define the x axis and the sum of **Price** on **Rows** to define the y axis. **Address** has been added to the **Detail** of the **Marks** card to define the level of aggregation. So, each mark on the scatterplot is a distinct address at a location defined by the size and price. **Type of Sale** has been placed on **Color**. Trend lines have been shown. As per Tableau's default settings, there are three trend lines: one per color. The confidence bands have been hidden.

Assuming good data and a good model, the trend lines demonstrate how much and how quickly **Price** is expected to rise in correlation with an increase in **Size** for each type of sale.



In this dataset, we have two fields, **Address** and **ID**, each of which defines a unique record. Adding one of those fields to the **Level of Detail** effectively disaggregates the data and allows us to plot a mark for each individual property. Sometimes, you may not have a dimension in the data that defines uniqueness. In those cases, you can disaggregate the data by unchecking **Aggregate Measures** from the **Analysis** menu.

Alternatively, you can use the drop-down menu on each of the measure fields on Rows and Columns to change them from measures to dimensions while keeping them continuous. As dimensions, each individual value will define a mark. Keeping them continuous will retain the axes required for trend lines.

Let's consider some of the options available for trend lines. You can edit trend lines by using the menu and selecting **Analysis | Trend Lines | Edit Trend Lines...** or by right-clicking on a trend line and then selecting **Edit Trend Lines....** When you do, you'll see a dialog box like this:

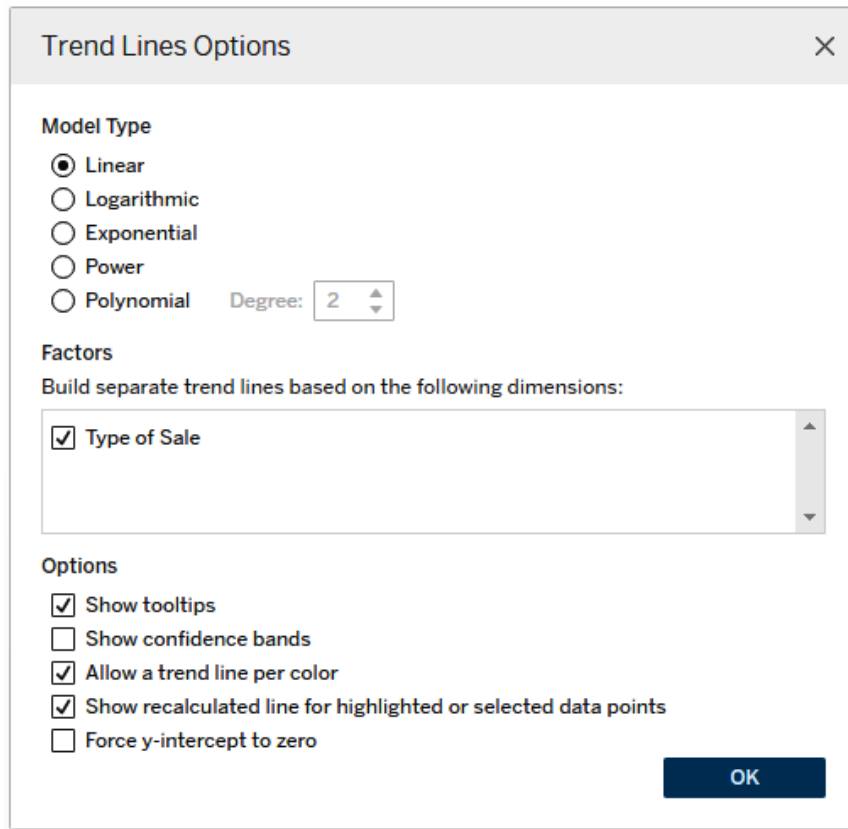


Figure 9.7: Tableau offers many options for configuring trend lines

Here, you have options for selecting a model type; selecting applicable fields as factors in the model; allowing discrete colors to define distinct trend lines; showing confidence bands; forcing the y -intercept to be zero; showing recalculated trends for selected marks; and showing tooltips for the trend line. We'll examine these options in further detail.



You should only force the y -intercept to be zero if you know that it must be zero. With this data, it is almost certainly not zero (that is, there are no houses in existence that are 0 square feet that are listed for \$0).

For now, experiment with the options. Notice how either removing the **Type of Sale** field as a factor or unchecking the **Allow a trend line per color** option results in a single trend line.

You can also see the result of excluding a field as a factor in the following view, where **Type of Sale** has been added to **Rows**:

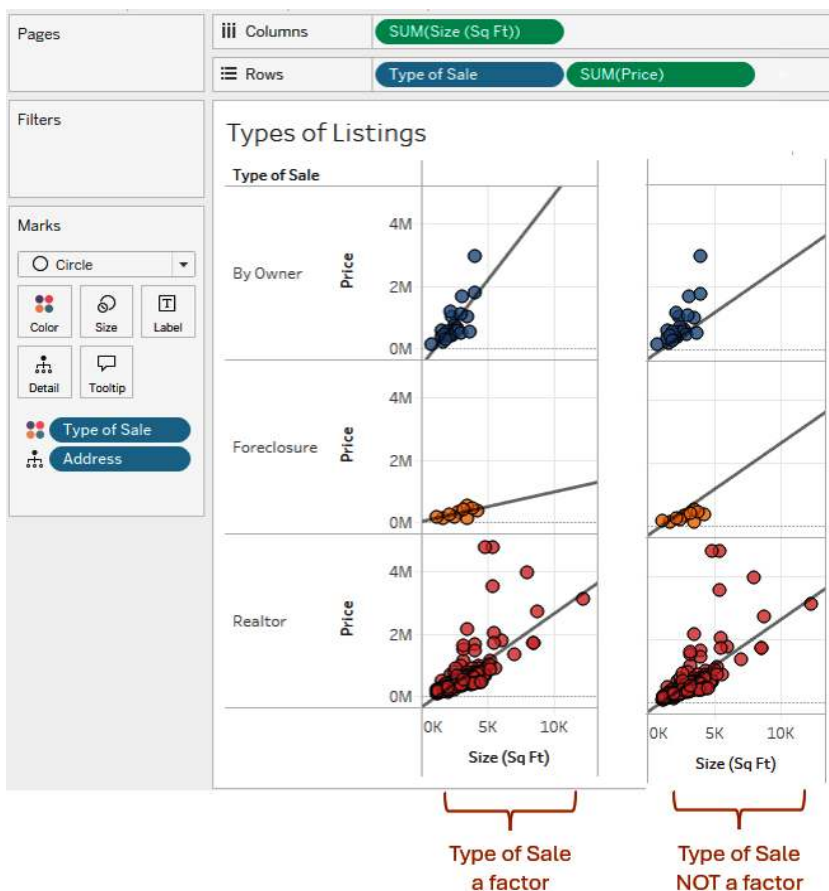


Figure 9.8: Including a field as a factor tells Tableau whether it contributes to the trend model

As represented in the left-hand portion of the preceding screenshot, **Type of Sale** is included as a factor. This results in a distinct trend line for each type of sale. When **Type of Sale** is excluded as a factor, the same trend line (which is the overall trend for all types) is drawn three times. This technique can be quite useful for comparing subsets of data to the overall trend.

Customizing the trend lines is only one aspect of using trends to understand the data. Also of significant importance is the trend model itself, which we'll consider customizing in the next section.

Trend models

Let's return to the original view and stick with a single trend line as we consider the trend models that are available in Tableau. The following models can be selected from the **Trend Line Options** window.

Linear

We'd use a linear model if we assumed that, as **Size** increases, **Price** will increase at a constant rate. No matter how much **Size** increases, we'd expect **Price** to increase so that new data points fall close to the straight line:

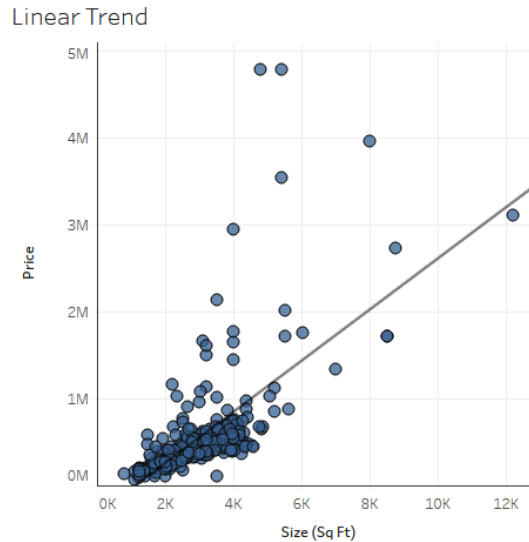


Figure 9.9: Linear trend

Logarithmic

We'd employ a logarithmic model if we expected the **law of diminishing returns** to be in effect—that is, the size can only increase so much before buyers will stop paying much more:

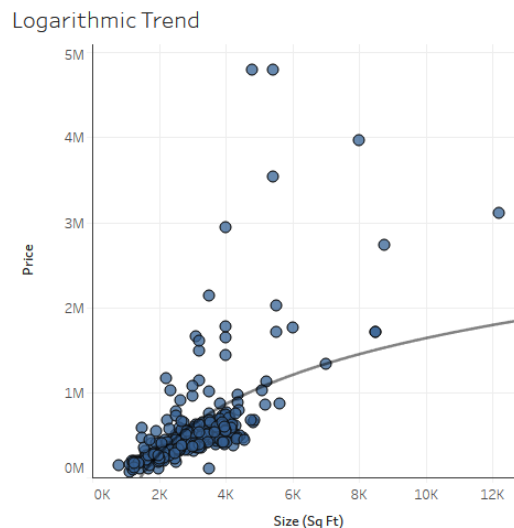


Figure 9.10: Logarithmic trend

Exponential

We'd use an exponential model to test the idea that each additional increase in size results in a dramatic (exponential!!) increase in price:

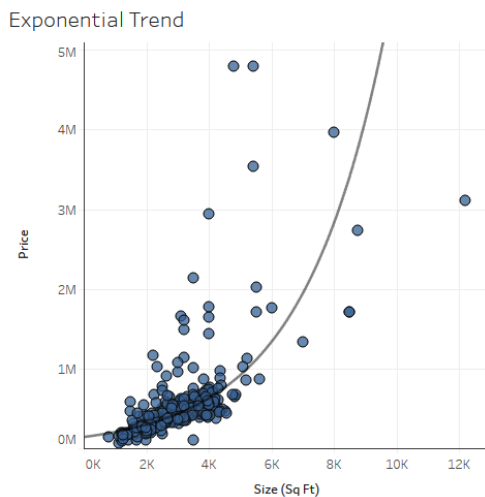


Figure 9.11: Exponential trend

Power

We'd employ a power trend model if we felt the relationship between size and price was non-linear and somewhere between a diminishing logarithmic trend and an explosive exponential trend. The curve would indicate that the price was a function of the size to a certain power. A power trend predicts certain events very well, such as the distance covered by the acceleration of a vehicle:

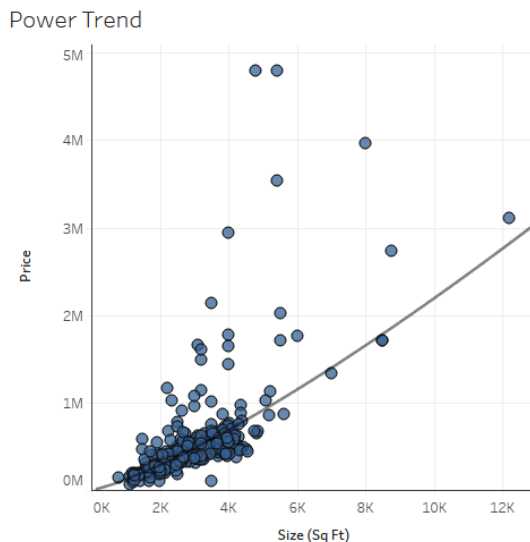


Figure 9.12: Power trend

Polynomial

We'd use this model if we felt the relationship between **Size** and **Price** was complex and followed more of an S-shaped curve, where initially increasing the size dramatically increased the price but, at some point, the price leveled off. You can set the degree of the polynomial model anywhere from 2 to 8. The trend line shown here is a 3rd-Degree polynomial:

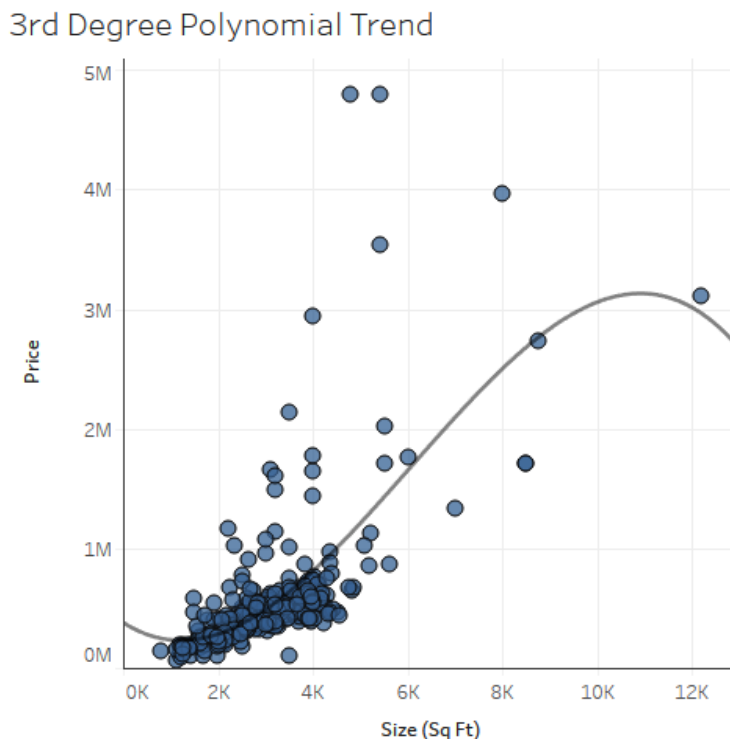


Figure 9.13: 3rd-Degree polynomial trend

You'll want to understand the basics of the trend models so that you can test and validate your assumptions of the data. Some of the trend models are clearly wrong for our data (though statistically indicated as valid, it is highly unlikely that prices will exponentially increase). A mixture of common sense along with an ever-increasing understanding of statistics will help you as you progress through your journey.

You may also wish to analyze your models for accuracy, and we'll turn there next.

Analyzing trend models

It can be useful to observe trend lines but, often, we want to understand whether the trend model we've selected is statistically meaningful. Fortunately, Tableau gives us some visibility into the trend models and calculations.

Simply hovering over a single trend line will reveal the formula as well as the **R-Squared** and **P-value** for that trend line:

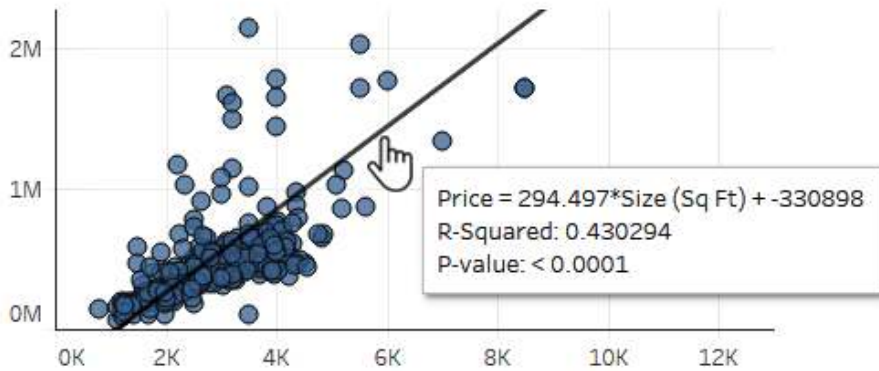


Figure 9.14: Tooltip displayed by hovering over the trend line



R-Squared gives a value that indicates how well the model fits. In general, the higher the R-Squared value, the better the fit.

If all the circle marks fell exactly on the line, you would see an R-Squared value of 1.

P-value is a statistical concept that describes the probability that the results of assuming no relationship between values (random chance) are at least as close as the results predicted by the trend model. A P-value of 5% (.05) would indicate a 5% chance of random chance describing the relationship between values at least as well as the trend model. This is why P-values of 5% or less are typically considered to indicate a significant trend model. A P-value higher than 5% often leads statisticians to question the correlation described by the trend model.

Additionally, you can see a much more detailed description of the trend model by selecting **Analysis | Trend Lines | Describe Trend Model...** from the menu or by using the similar menu from a right-click on the view's pane.

When you view the trend model, you'll see the **Describe Trend Model** window:

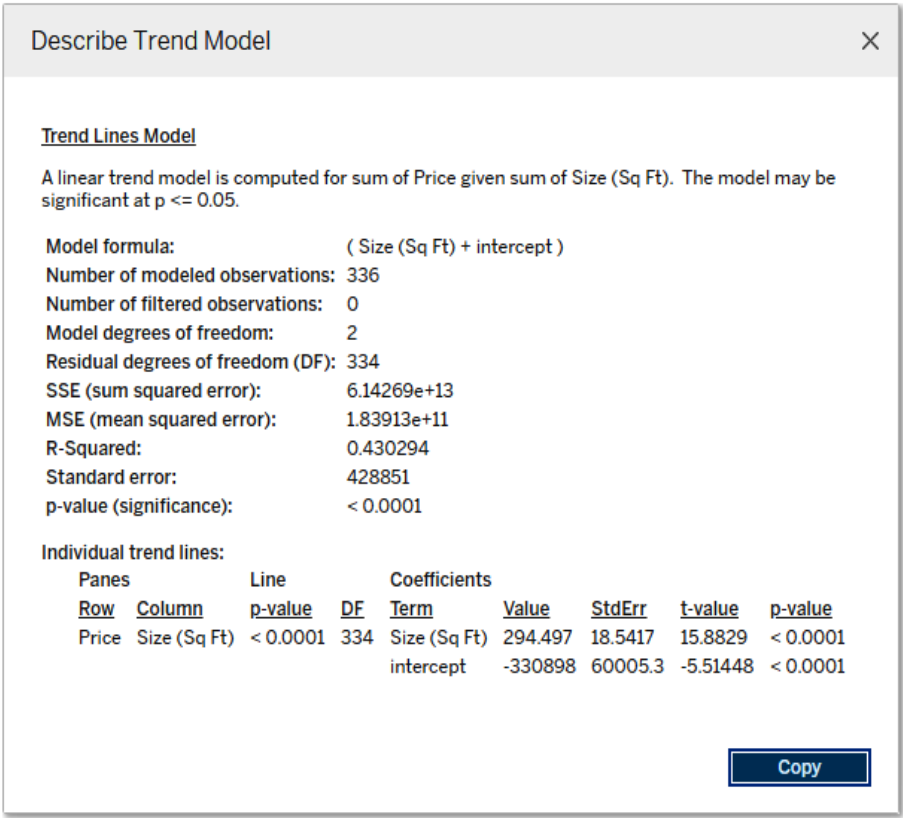


Figure 9.15: The Describe Trend Line window

The statistical information shown in the window includes the equation defining the trend model, the P-value, and the coefficients.



You can also get a trend model description in the worksheet description, which is available from the **Worksheet** menu or by pressing *Ctrl + E*. The worksheet description includes quite a bit of other useful summary information about the current view.

You'll recall, in a previous example, the option to define a field as a factor or to have separate trend lines per color. In those cases, a separate P-value will be calculated for each trend line, and you may find that the P-value is different for different lines in a visualization (for example, the lines in *Figure 9.6*). At times, you may even observe that the model is statistically significant overall, even though one or more trend lines may not be.

Additional summary statistical information can be displayed in Tableau Desktop for a given view by showing a summary. From the menu, select **Worksheet | Show Summary**. The information displayed in the summary can be expanded using the drop-down menu on the **Summary** card:

Summary	
Count:	336
SUM(Price)	
Sum:	183,703,400
Average:	546,736.31
Minimum:	68,400
Maximum:	4,790,100
Median:	410,500.00
Standard deviation:	567,324
First quartile:	289,425.00
Third quartile:	561,100.00
Skewness:	4.59
Excess Kurtosis:	25.74
SUM(Size (Sq Ft))	
Sum:	1,001,318
Average:	2,980.11
Minimum:	784
Maximum:	12,200
Median:	2,800.50
Standard deviation:	1,264
First quartile:	2,108.50
Third quartile:	3,627.00
Skewness:	2.27
Excess Kurtosis:	10.95

Figure 9.16: Summary information

The wealth of information available via tooltips and summaries will help you evaluate your trend models and understand the accuracy and details. But we can go even further, by exporting and analyzing statistical data for the trend models. We'll consider that next.

Exporting statistical model details

Tableau also gives you the ability to export data, including data related to trend models. This allows you to, more deeply, and even visually, analyze the trend model itself. Let's analyze the third-degree polynomial trend line of the real estate price and size scatterplot without any factors. To export data related to the current view, use the menu to select **Worksheet | Export | Data**. The data will be exported as a **Microsoft Access Database (.mdb)** file and you'll be prompted where to save the file.



The ability to export data to **Access** is limited to Windows only. If you're using a Mac, you won't have the option. You may wish to skip this section but don't worry if you aren't able to replicate the examples.

On the **Export Data to Access** screen, specify an access table name and select whether you wish to export data from the entire view or the current selection. You may also specify that Tableau should connect to the data. This will generate the data source and make it available with the specified name in the current workbook:

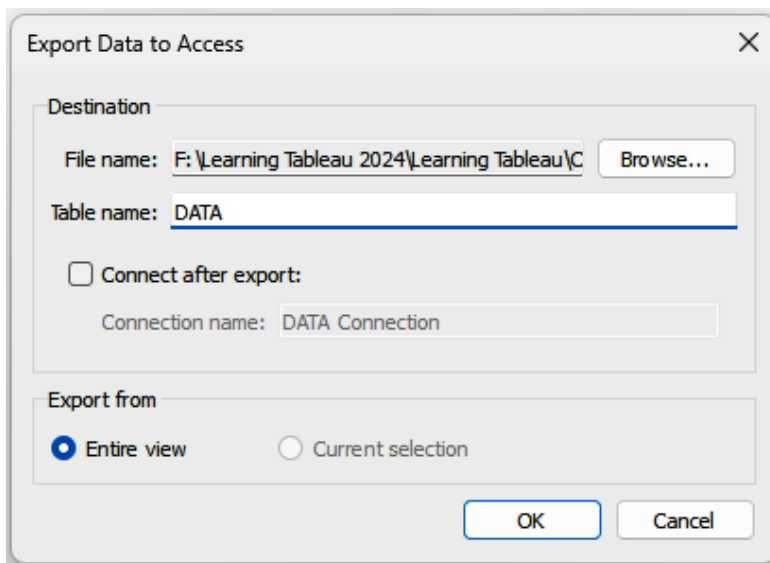


Figure 9.17: The *Export Data to Access* dialog box

The new data source connection will contain all the fields that were in the original view as well as additional fields related to the trend model.

This allows us to build a view like the following, using the residuals and predictions (you'll notice that Tableau has exported the fields with quite verbose names):

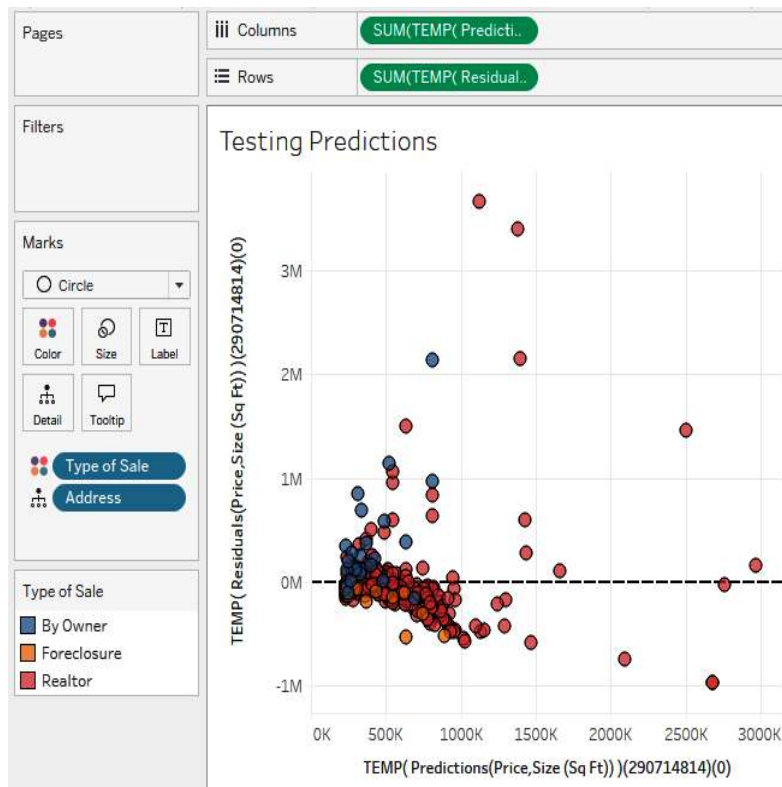


Figure 9.18: A view using residuals and predictions to test the model

A scatterplot of predictions (x axis) and residuals (y axis) allows you to see how far each mark was from the location predicted by the trend line. In effect, the trend line has been flattened out to zero and allows you to see each mark's vertical distance from zero.

It also allows you to see whether residuals are distributed evenly on either side of zero. An uneven distribution would likely indicate problems with the trend model.

You can include this new view along with the original on a dashboard to explore the trend model visually. Use the Highlight button on the toolbar to highlight the **Address** field:

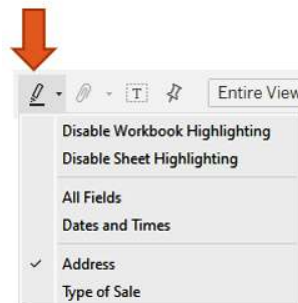
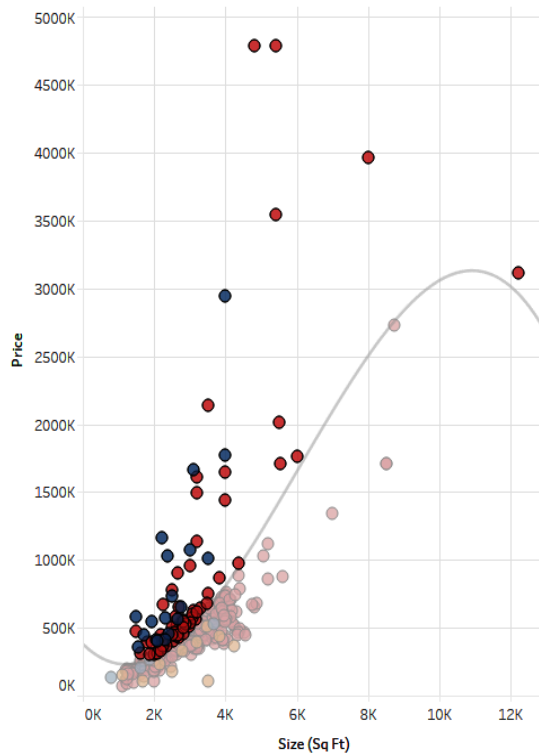


Figure 9.19: Using the Highlight button on the toolbar to highlight the Address field

With the highlight action defined, selecting marks in one view will allow you to see them in the other. You could extend this technique to export multiple trend models and dashboards to evaluate several trend models at the same time:

3rd Degree Polynomial



Testing Predictions

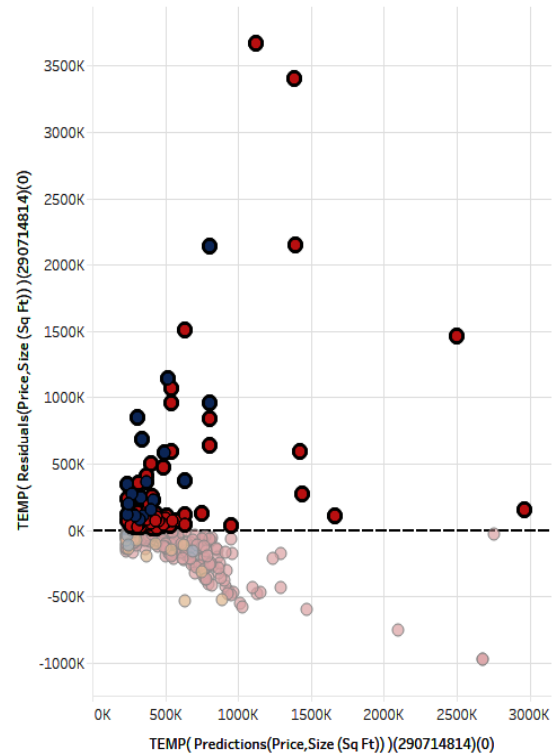


Figure 9.20: Placing the original view alongside the testing view allows you to see the relationship

Understanding and using trend models can be very useful in analyzing data. It helps you make predictions and see trends and patterns. One very important aspect of looking at trends is finding outliers and understanding what makes them exceptional.

Sometimes an outlier may be the result of bad or suspect data. Sometimes outliers truly don't matter to the overall analysis. But sometimes you might find something unexpected that prompts further investigation and leads to exciting discoveries. Tableau provides an excellent way to begin exploring marks that fall outside expected norms. We'll turn our attention next to a relatively new feature in Tableau: Data Guide.

Data Guide

With Data Guide, you can begin to uncover some of the reasons and explanations behind the data you see in your visualizations. Why are sales up so much at a specific point on the timeline? Why was profit down so much last month? Data Guide takes some of the tedious manual effort out of hunting for answers to questions like these.

To use Data Guide for a view, click the signpost icon next to **Show Me**:

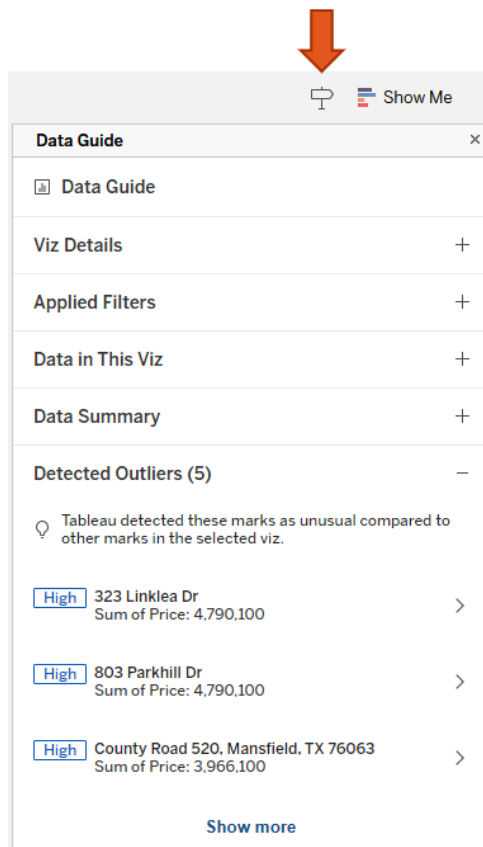


Figure 9.21: Expanding Data Guide using the signpost

The Data Guide pane reveals a lot of information and options. The sections include:

- **Viz Details** allows you to edit the viz description, add links for additional resources, and add accessibility text. All of this will be available to end users.
- **Applied Filters** describes the filters applied to the view.
- **Data in This Viz** gives an overview of the fields used in the view.
- **Data Summary** gives a summary of the viz, including the number of marks.
- **Detected Outliers** gives a list of outlier marks, allowing you to drill into details to understand what factors may have contributed to exceptionally high or low values.

Additionally, you may simply select a mark in a view that you'd like to understand and select the **Data Guide** option on the tooltip toolbar (it also looks like a signpost).

As an example, select the mark for 803 Parkhill Dr. Mansfield TX 76063 in the **Data Guide** sheet (shown in *Figure 9.22*) and then click the **Data Guide** icon on the tooltip, as shown here:

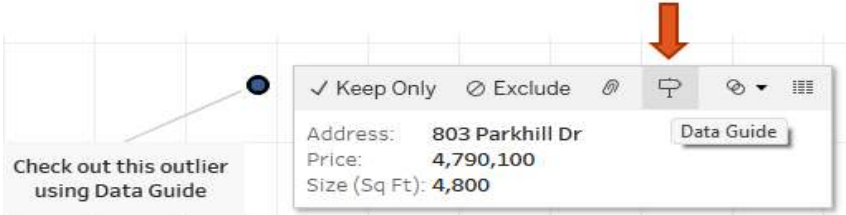


Figure 9.22: Using Data Guide to evaluate a mark in a viz

This opens the **Data Guide** pane, which looks like this:

Data Guide

803 Parkhill Dr

Data in This Mark

Real Estate

Address

803 Parkhill Dr

Price

4,790,100

Size (Sq Ft)

4,800

Applied Filters

Explanations

Explore underlying values for 803 Parkhill Dr

High

Sum of Price

4,790,100

High

Sum of Size (Sq Ft)

4,800

Other things to explore

What is unique about 803 Parkhill Dr?

Figure 9.23: The Explain Data pane

You'll notice that the **Sum of Price** and **Sum of Size (Sq Ft)** are both indicated as **High** compared to other marks. You may click each measure to get potential explanations for each value. For example, here is the result of clicking **Sum of Price** and then one of the identified contributing measures, **# Beds**:

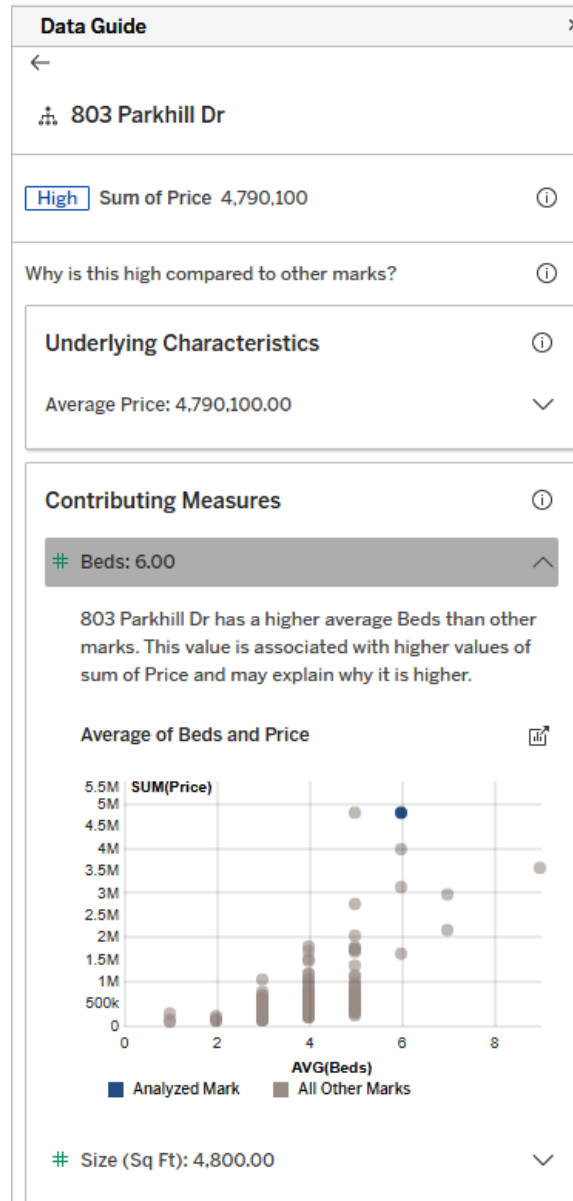


Figure 9.24: Exploring the contributing measures for values

Observing the results reveals one explanation for **Sum of Price**: a correlation with the number of **Beds**. The visualization reveals that the property does indeed have a high number of beds compared with other properties.

Another factor given to explore is the size. To account for this and any other factors, you'll need to explore the issue further to verify the actual cause(s) and to ensure the data is good. But it seems reasonable that having more bedrooms than most other properties may indeed increase the value of a given property.

In some cases, Data Guide will reveal issues with the data. For example, Data Guide does indicate 323 Linklea Dr as another outlier with a high price and high size. However, looking into the explanations and examining the **Underlying Characteristics** reveals an interesting find:

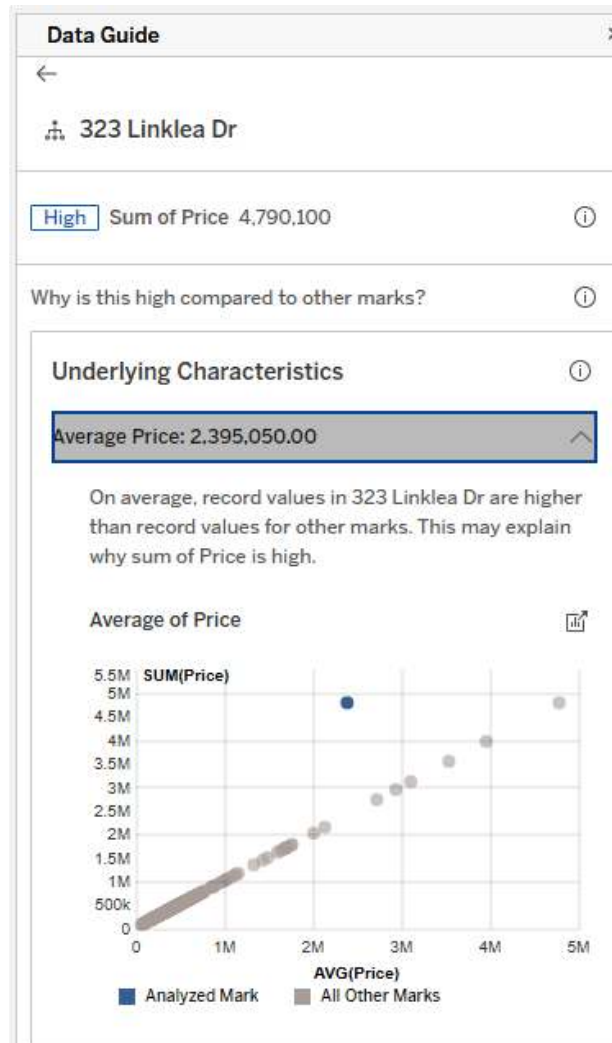


Figure 9.25: Exploring the underlying characteristics of a mark

The sum of Price and average of Price for every other property is equal, but 323 Linklea Dr has a sum that is twice its average. A bit of further exploration reveals the reason: there are duplicate records in the dataset. This reveals bad data as the reason behind this outlier.

Outlier analysis is made far easier with Data Guide. Much of the guesswork is removed and you can identify paths to explore much more quickly and efficiently. Next, we'll take a look at Tableau's capability for identifying complex relationships within data using clustering.

Clustering

Tableau gives you the ability to quickly perform clustering analysis in your visualizations. This allows you to find groups, or **clusters**, of individual data points that are similar based on any number of your choosing. This can be useful in many different industries and fields of study, as in the following examples:

- Marketing may find it useful to determine groups of customers related to each other based on spending amounts, frequency of purchases, or times and days of orders.
- Patient care directors in hospitals may benefit from understanding groups of patients related to each other based on diagnoses, medication, length of stay, and the number of readmissions.
- Immunologists may search for related strains of bacteria based on drug resistance or genetic markers.
- Renewable energy consultants may like to pinpoint clusters of windmills based on energy production and then correlate that with geographic location.



Tableau uses a standard **k-means clustering** algorithm that will yield consistent results every time the view is rendered. Tableau will automatically assign the number of clusters (k), but you have the option of adjusting the value as well as assigning any number of variables.

As we consider clustering, we'll turn once again to the real estate data to see whether we can find groupings of related houses on the market and then determine whether there's any geographic pattern based on the clusters we find.

Although you can add clusters to any visualization, we'll start with a scatterplot because it already allows us to see the relationship between two variables. That will give us some insight into how clustering works, and then we can add additional variables to see how the clusters are redefined.

Beginning with the basic scatterplot of **Address** by **Size** and **Price**, switch to the **Analytics** pane and drag **Cluster** to the view:

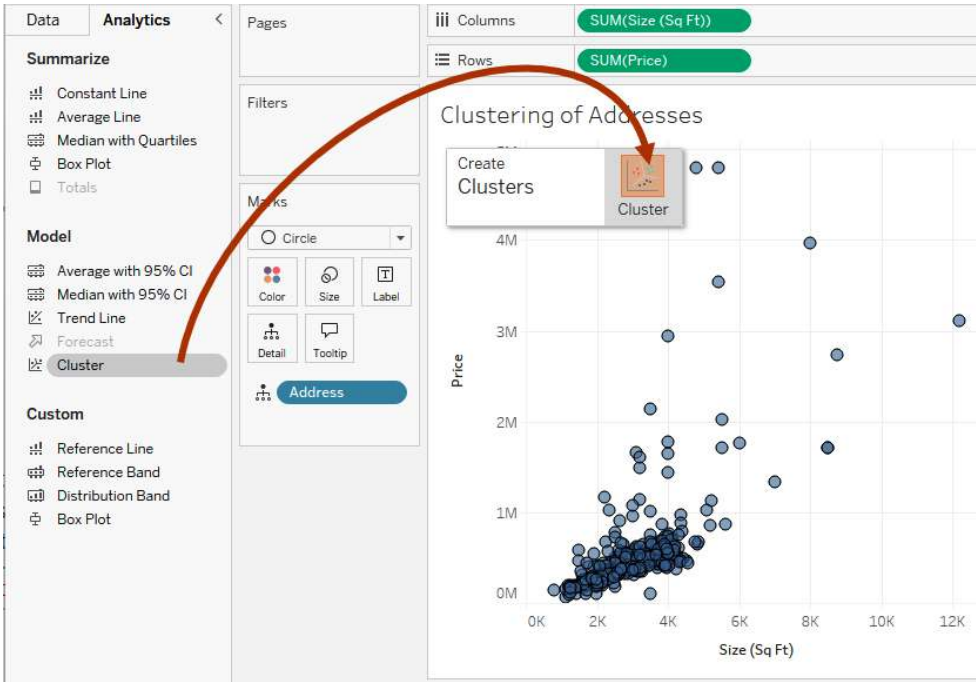


Figure 9.26: Adding clusters by dragging and dropping from the Analytics pane

When you drop **Cluster** onto the view, Tableau will generate a new **Clusters** field (automatically placed on **Color** here) and will display a **Clusters** window containing the fields used as variables and an option to change the number of clusters. **Variables** will contain the measures already in the view by default:



Figure 9.27: Clusters of individual addresses based on Price and Size

Variables are all the factors that the clustering algorithm uses to determine related data points. **Number of Clusters** determines how many groups the data is partitioned into. In the preceding view, you'll observe three clusters of houses:

- Those with a low price and a smaller size (blue).
- Those with an average price and size (orange).
- Those with a high price and a large size (gray).

Because the two variables used for the clusters are the same as those used for the scatterplot, it's relatively easy to see the boundaries of the clusters (you can imagine a couple of diagonal lines partitioning the data).

You can drag and drop nearly any field into and out of the **Variables** section (from the Data pane or the view) to add and remove variables. The clusters will automatically update as you do so. Experiment by adding **Beds** to the **Variables** list and observe that there's now some overlap between **Cluster 1** and **Cluster 2** because some larger homes only have two or three bedrooms while some smaller homes might have four or five. The number of bedrooms now helps define the clusters. Remove **Beds** and note that the clusters are immediately updated again.

Once you have meaningful clusters, you can materialize the clusters as groups in the data source by dragging them from the view and dropping them into the **Data** pane:

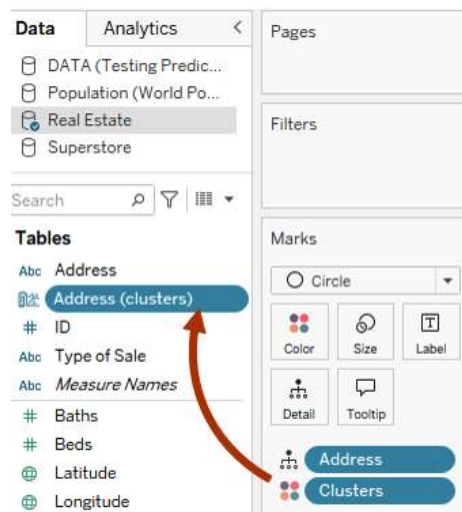


Figure 9.28: Materializing a cluster by dragging the Clusters field from the view to the Data pane

The cluster group will not be recalculated at render time. To recalculate the cluster group, use the dropdown on the field in the **Data** pane and select **Refit**.

Using a cluster group allows you to accomplish a lot, including the following:

- Cluster groups can be used across multiple visualizations and can be used in actions on dashboards.
- Cluster groups can be edited, and individual members moved between groups if desired.

- Cluster group names can be aliased, allowing more descriptive names than **Cluster 1** and **Cluster 2**.
- Cluster groups can be used in calculated fields, while clusters can't.

In the following example, a map of the properties has been color-coded by the **Address (clusters)** group in the previous view to help us see whether there's any geographic correlation to the clusters based on price and size.

While the clusters could have been created directly in this visualization, the group has some of the advantages mentioned:

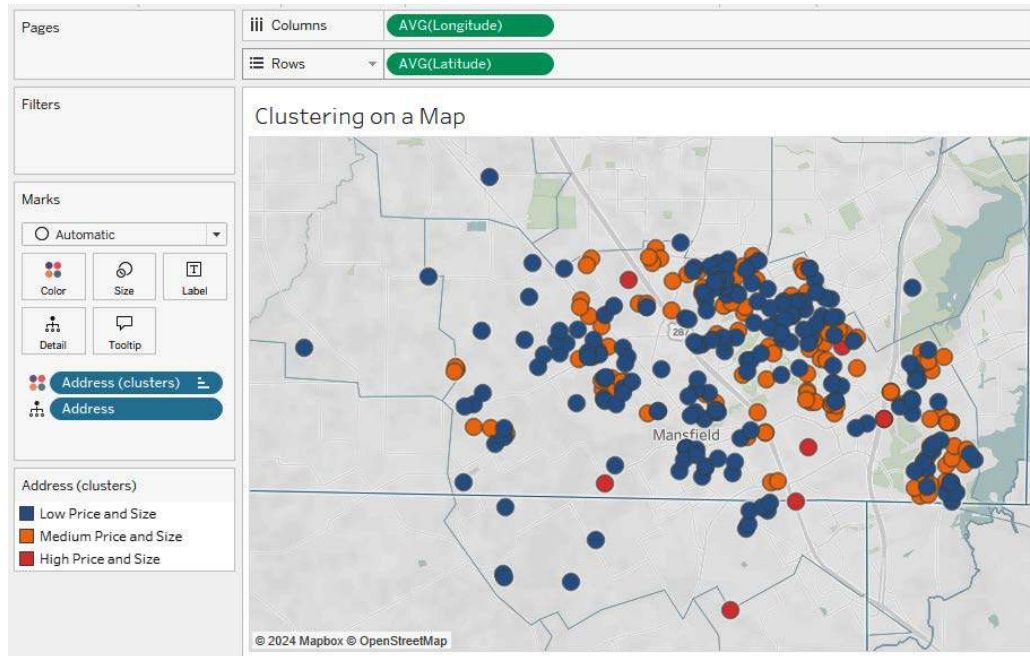


Figure 9.29: This view uses the clusters we identified to additionally understand any geospatial relationships

In the view here, each original cluster is now a group that has been aliased to give a better description of the cluster. You can use the drop-down menu for the Group field in the Data pane or right-click the item in the Color legend to edit aliases.



There are a lot of options for editing how maps appear. You can adjust the layers that are shown on maps to help provide additional context for the data you are plotting. From the top menu, select **Maps | Background Layers**. The layer options will show in the left-hand sidebar. The preceding map has had **Streets**, **Highways**, **County Borders**, and **Zip Code Boundaries** enabled to give each address a more precise context of location. The layers that are available to select will depend on the zoom level of the map. We'll cover these features in more detail in *Chapter 12, Exploring Mapping and Advanced Geospatial Features*.

In looking at the previous view, you do indeed find neighborhoods that are almost exclusively **Low Price and Size (Cluster 1)** and others that are almost exclusively **Mid-Range (Cluster 2)**. Consider how a real estate investor might use such a visualization to look for a low-priced house in a mid-range neighborhood.

In addition to the clustering of values, we might want to understand the distribution of values. We'll turn there next.

Distributions

Analyzing distributions can be quite useful. We've already seen that certain calculations are available for determining statistical information such as averages, percentiles, and standard deviations. Tableau also makes it easy to quickly visualize various distributions, including confidence intervals, percentiles, percentiles, quantiles, and standard deviations.

You may add any of these visual analytic features using the **Analytics** pane (alternately, you can right-click on an axis and select **Add Reference Line**). Just like reference lines and bands, distribution analytics can be applied within the scope of a table, pane, or cell. When you drag and drop the desired visual analytic, you'll have options for selecting the scope and the axis. In the following example, we've dragged and dropped **Distribution Band** from the **Analytics** pane onto the scope of **Pane** for the axis defined by **Sum(Price)**:

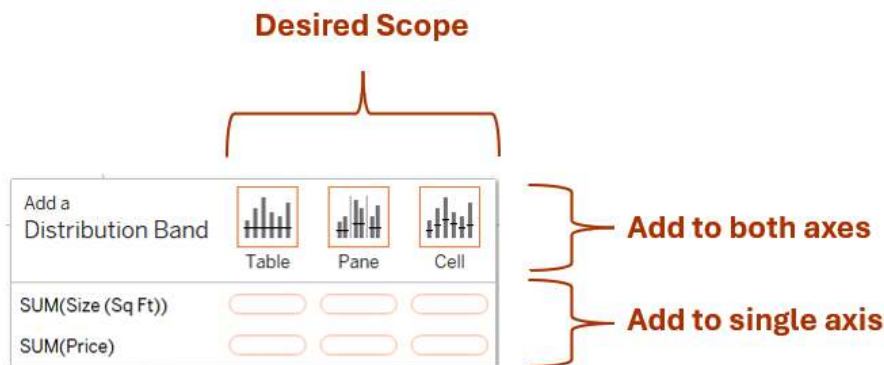


Figure 9.30: Defining the scope and axes as you add reference lines and distributions from the Analytics pane

Once you've selected the scope and axes, you'll be given options to change some settings. You may also edit lines, bands, distributions, and box plots by right-clicking the analytic feature in the view or by right-clicking the axis or the reference lines themselves.

As an example, let's take the scatterplot of addresses by price and size with **Type of Sale** on **Columns** in addition to **Color**:

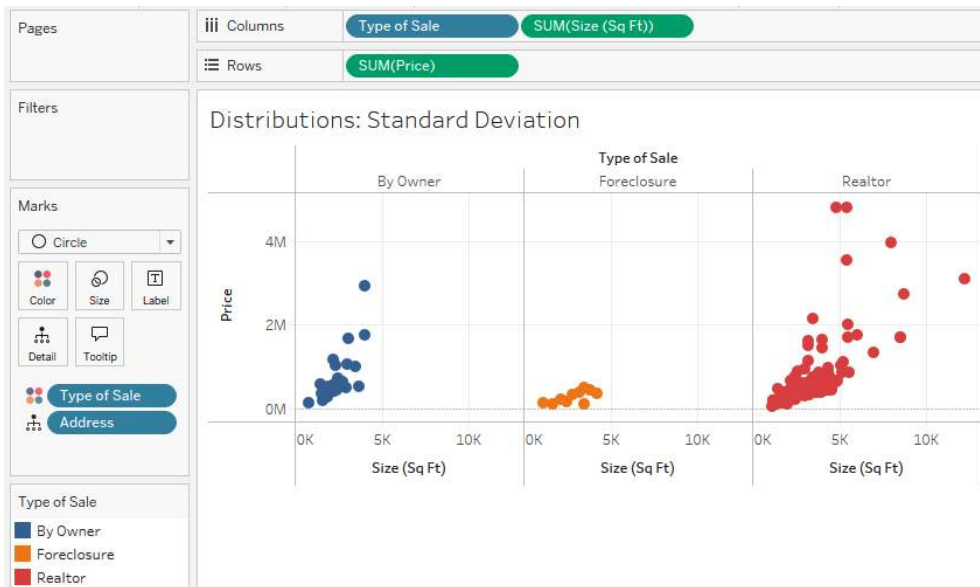


Figure 9.31: A scatterplot divided into three columns

Next, we drag and drop the **Distribution** band from the **Analytics** pane onto **Pane** only for the axis defined by **Price**. This brings up a dialog box to set the options:

Figure 9.32: The dialog box for adding or editing lines, bands, distributions, or box plots

Each specific **Distribution** option specified in the **Value** drop-down menu under **Computation** has unique settings. **Confidence Interval**, for example, allows you to specify a percent value for the interval. **Standard Deviation** allows you to enter a comma-delimited list of values that describe how many standard deviations there will be and at what intervals. The preceding settings reflect specifying standard deviations of -2, -1, 1, and 2. After adjusting the label and formatting as shown in the preceding screenshot, you should see results like this:

Distributions: Standard Deviation

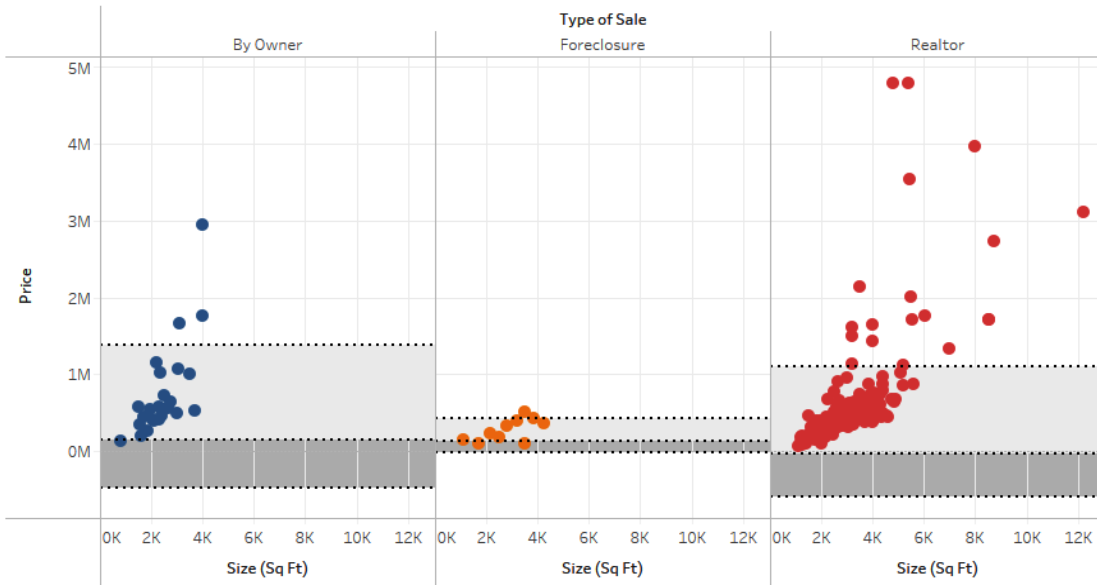


Figure 9.33: Two standard deviations of Price for each Type of Sale

Since you applied the standard deviations per pane, you get different bands for each type of sale. Each axis can support multiple distributions, reference lines, and bands. You could, for example, add an average line in the preceding view to help a viewer understand the center of the standard deviations.

On a scatterplot, using a distribution for each axis can yield a very useful way to analyze outliers.

Showing a single standard deviation for both **Area** and **Price** allows you to easily see properties that fall within norms for both, one, or neither (all other factors being equal, you might consider purchasing a house that was on the high end of size but within normal price limits!):

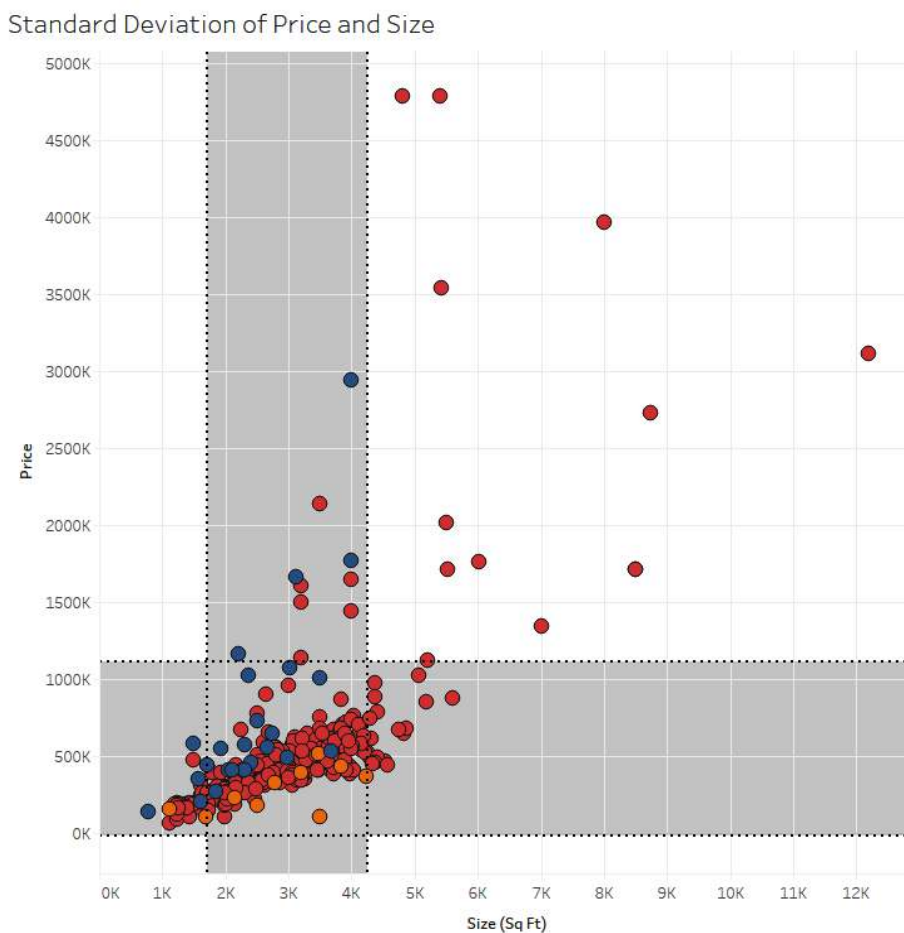


Figure 9.34: One standard deviation for both Price and Size of all houses

Distributions are a powerful tool for understanding outliers and concentrations of values. Next, we'll turn our attention to another useful tool: forecasting.

Forecasting

As we've seen, trend models make predictions. Given a good model, you expect additional data to follow the trend. When the trend is over time, you can get some idea of where future values may fall. However, predicting future values often requires a different type of model. Factors such as seasonality can make a difference not predicted by a trend alone. Tableau includes built-in forecasting models that can be used to predict and visualize future values.

To use forecasting, you'll need a view that includes a **Date** field or enough date parts for Tableau to reconstruct a date (for example, a **Year** and a **Month** field). Tableau also allows forecasting based on integers instead of dates. You may drag and drop a forecast from the **Analytics** pane, select **Analytics** | **Forecast** | **Show Forecast** from the menu, or right-click on the view's pane and select the option from the context menu.

Here, for example, is the view of the population growth over time for Afghanistan and Australia with forecasts shown:

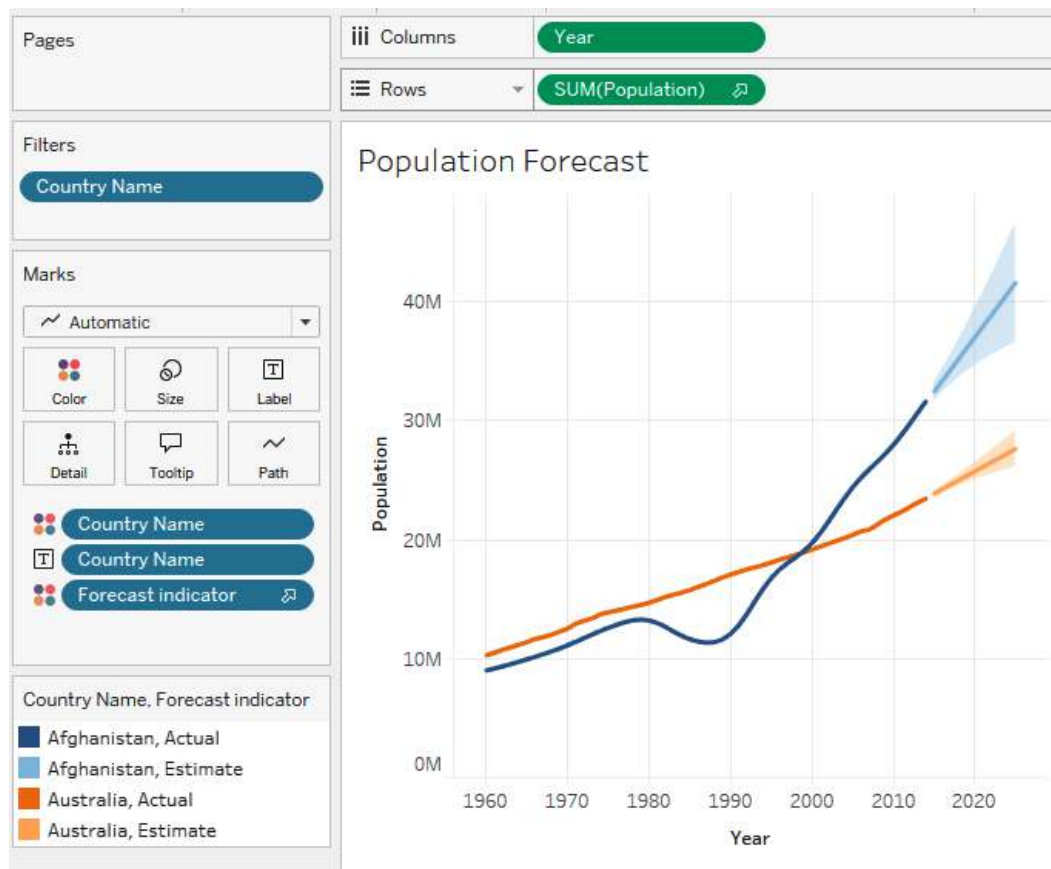


Figure 9.35: A forecast of the population for both Afghanistan and Australia

Note that, when you show the forecast, Tableau adds a Forecast icon to the SUM(Population) field on Rows to indicate that the measure is being forecast. Additionally, Tableau adds a new, special **Forecast indicator** field to Color so that forecast values are differentiated from actual values in the view.



You can move the **Forecast indicator** field or even copy it (hold the *Ctrl* key while dragging and dropping) to other shelves to further customize your view.

When you edit the forecast by selecting **Analysis | Forecast | Forecast Options...** from the menu or use the right-click context menu on the view, you'll be presented with various options for customizing the trend model, like this:

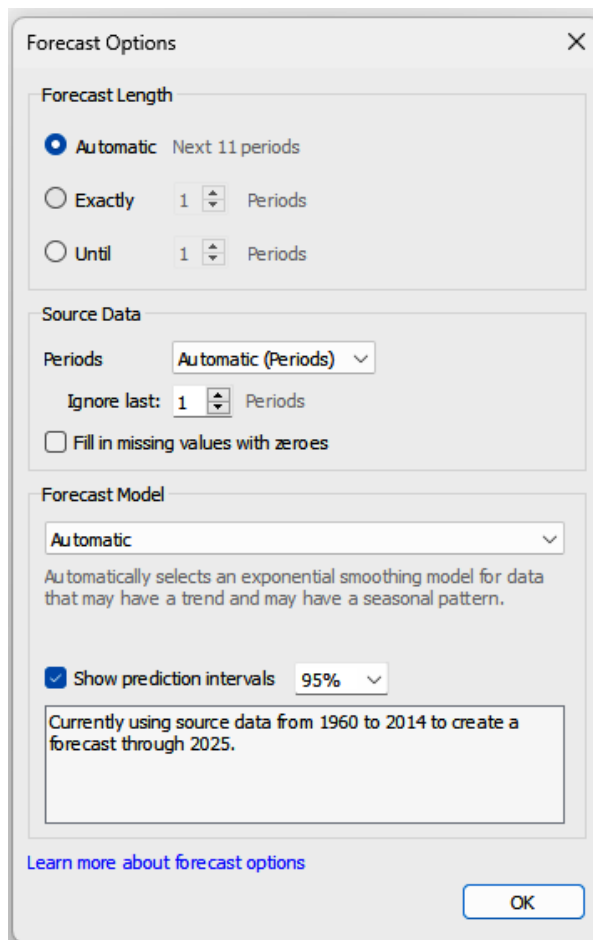


Figure 9.36: The Forecast Options dialog box

Here, you have options to set the length of the forecast, determine aggregations, customize the model, and set whether you wish to show prediction intervals. The forecast length is set to **Automatic** by default, but you can extend the forecast by a custom value.

The options under **Source Data** allow you to optionally specify a different grain of data for the model. For example, your view might show a measure by year, but you could allow Tableau to query the source data to retrieve values by month and use the finer grain to potentially achieve better forecasting results.



Tableau's ability to separately query the data source to obtain data at a finer grain for more precise results works well with relational data sources. However, **Online Analytical Processing (OLAP)** data sources aren't compatible with this approach, which is one reason why forecasting isn't available when working with cubes.

By default, the last value is excluded from the model. This is useful when you're working with data where the most recent time period is incomplete. For example, when records are added daily, the last (current) month isn't complete until the final records are added on the last day of the month. Prior to that last day, the incomplete time period might skew the model if it's not ignored.

The model itself can be set to automatic with or without seasonality or can be customized to set options for seasonality and trend. To understand the options, consider the following view of **Sales by Month** from the Superstore sample data:

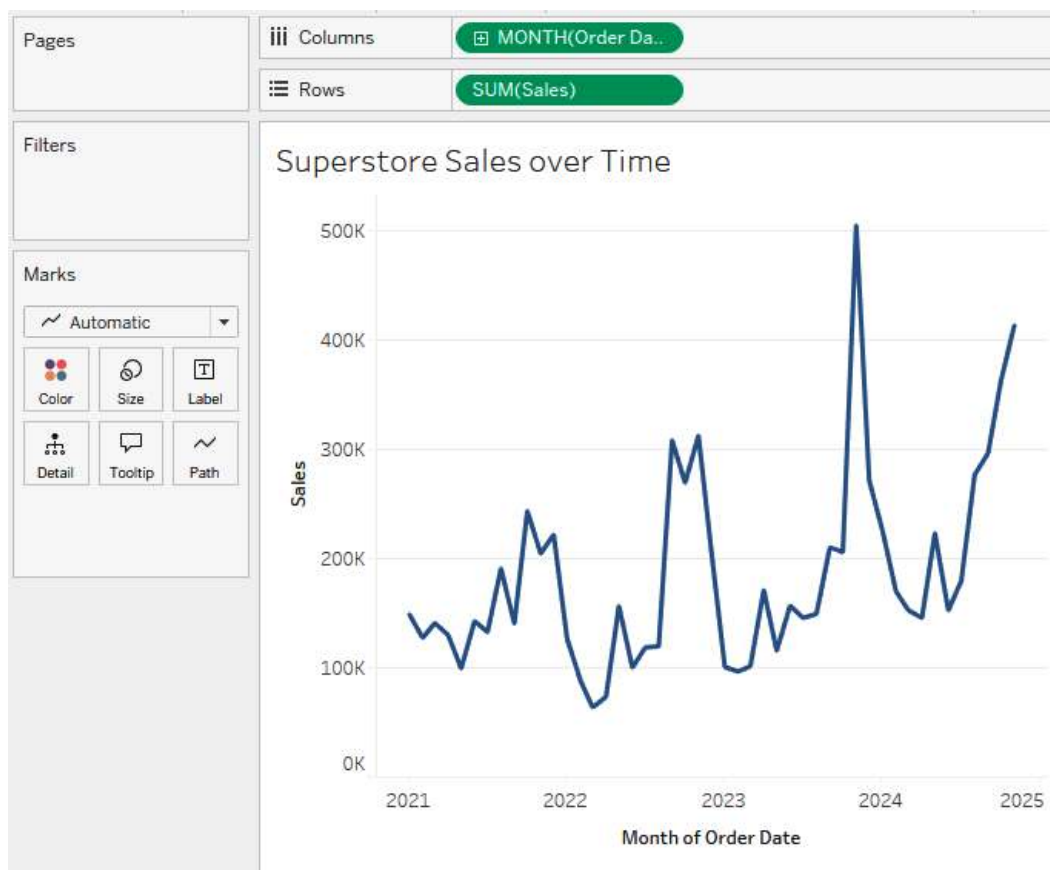


Figure 9.37: This time series shows a cyclical or seasonal pattern

The data displays a distinct cyclical or seasonal pattern. This is very typical for retail sales and other types of data. The following are the results of selecting various custom options:

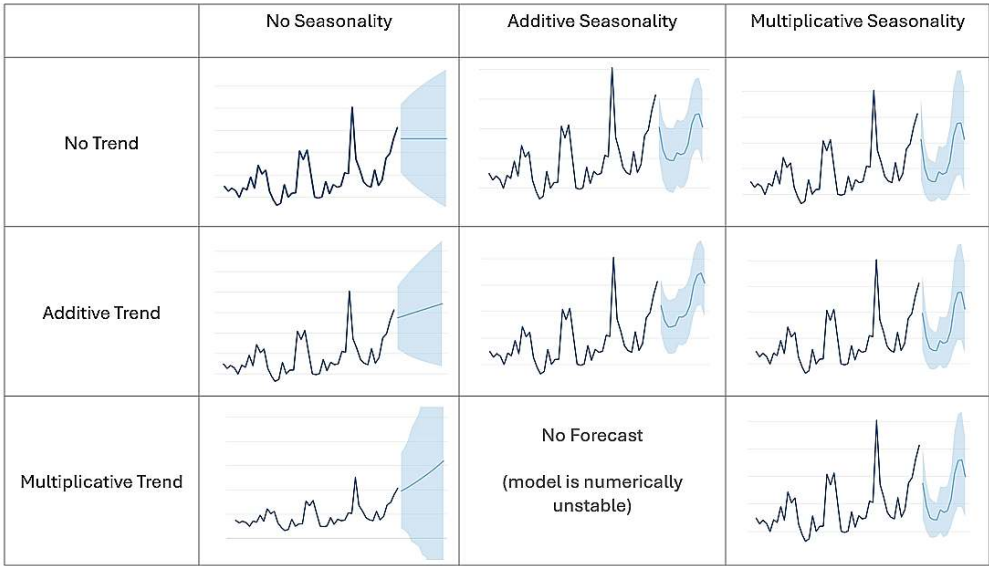


Figure 9.38: Selecting various forecast models will yield different results

Examining the differences above will help you understand the differences in the options. For example, note that no seasonality results in a straight line that does not fluctuate with seasons. Multiplicative trends result in sharper inclines and decreases, while multiplicative seasonality results in sharper variations.

Much like trends, forecast models and summary information can be accessed using the menu.

Selecting **Analysis | Forecast | Describe Forecast** will display a window with tabs for both the summary and details concerning the model:

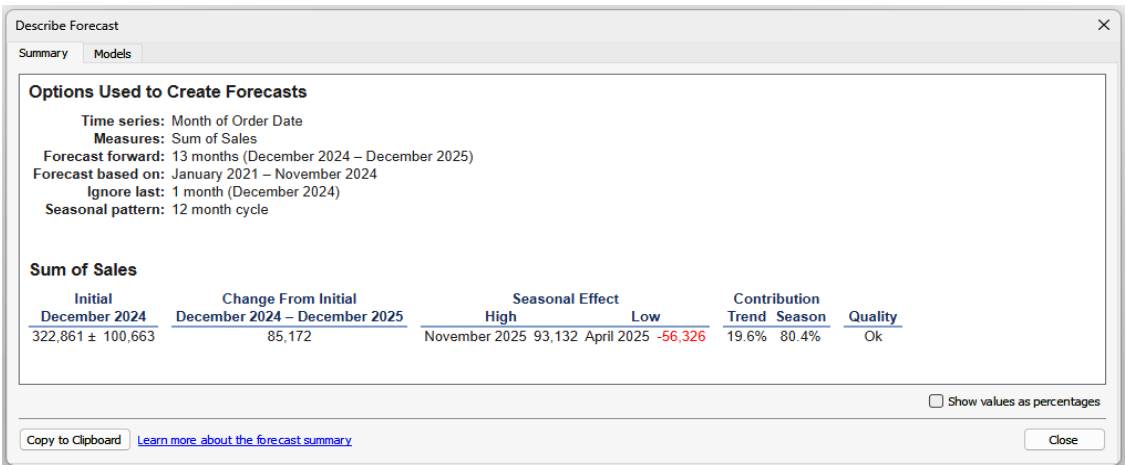


Figure 9.39: Tableau describes the forecast model

Clicking the **Learn more about the forecast summary** link at the bottom of the window will give you much more information on the forecast models used in Tableau.



Forecast models are only enabled given a certain set of conditions. If the option is disabled, ensure that you're connected to a relational database and not OLAP, that you're not using table calculations, and that you have at least five data points.

Summary

Tableau provides an extensive set of features for adding value to your analysis. Trend lines allow you to more precisely identify outliers, determine which values fall within the predictions of certain models, and even make predictions of where measurements are expected. Tableau gives extensive visibility into the trend models and even allows you to export data containing trend model predictions and residuals. Clusters enable you to find groups of related data points based on various factors. Distributions are useful for understanding the spread of values across a dataset. Forecasting allows a complex model of trends and seasonality to predict future results. Having a good understanding of these tools will give you the ability to clarify and validate your initial visual analyses. With this understanding, you'll be able to better leverage additional features, such as Data Guide, with practical application to the real-world problems you face.

Next, we'll consider some advanced visualization types that will expand the horizons of what you are able to accomplish with Tableau and the way in which you communicate data to others!

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



10

Advanced Visualizations

We've explored many different types of visualizations and considered which types of questions they best answer. For example, **bar charts** aid in comparing values; **line charts** can show changes and trends over time; **stacked bars** and **treemaps** help us see part-to-whole relationships; and **box plots** help us understand distributions and outliers. We've also seen how to enhance our understanding and data storytelling with calculations, annotations, formatting, and reference lines. With this knowledge as a foundation, we'll expand the possibilities of data analysis with some advanced visualizations.

These are only examples of Tableau's amazing flexibility and are meant to inspire you to think through new ways of seeing, understanding, and communicating your data. These are not designed as complex charts for the sake of complexity, but rather to spark creativity and interest to effectively communicate data.

We'll consider the following topics:

- Advanced visualizations—when and why to use them
- Slope charts and bump charts
- Waterfall charts
- Step lines and jump lines
- Sparklines
- Dumbbell charts
- Unit/symbol charts
- Marimekko charts
- Sunburst (using Viz Extensions)
- Animated visualizations

Advanced visualizations—when and why to use them

The visualization types we've seen up to this point will answer many, if not most, of the questions you have about your data. If you are asking questions of *when?*, then a time series is the most likely solution. If you are asking *how much?*, a bar chart gives a good, quick result. But there are times when you'll ask questions that are better answered with a different type of visualization. For example, movement or flow might be best represented with a Sankey diagram. *How many?* might be best answered with a unit or symbol chart.

Comparing changes in ranks or absolute values might be best accomplished with a slope or bump chart. The visualizations that follow are not what you will use as you first explore the data. But as you dive deeper into your analysis and want to know or communicate more, you might consider some of the options in this chapter.

Each of the visualizations in this chapter is created using the supplied Superstore data. Instead of providing step-by-step instructions, we'll point out specific advanced techniques used to create each chart type. The goal is not to memorize steps but to understand how to creatively leverage Tableau's features.

You can find completed examples in the [Chapter 10 Complete workbook](#), or test your growing Tableau skills by building everything from scratch using the [Chapter 10 Starter workbook](#).

Let's start our journey into advanced visualizations with slope and bump charts.

Slope charts and bump charts

A **slope chart** shows a change of values from one period or status to another. For example, here is a slope chart demonstrating the change in sales rank for each state in the **South** region from 2023 to 2024:

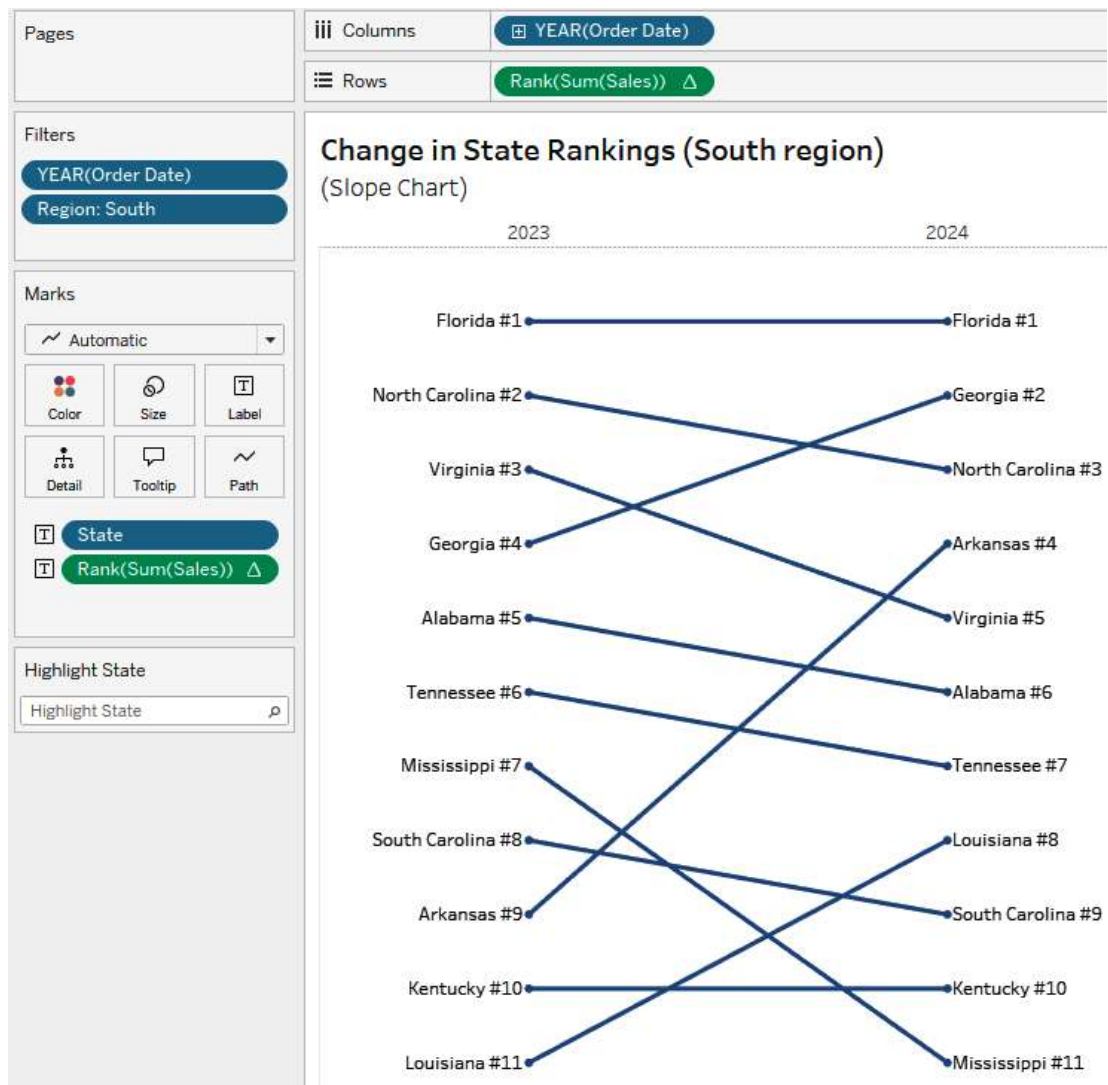


Figure 10.1: A slope chart is useful to compare the change of rank or absolute values from one period or status to another

Here are some features and techniques used to create the preceding slope chart:

- The table calculation **Rank(Sum(Sales))** is computed by (addressed by) **State**, meaning that each state is ranked within the partition of a single year.
- **Grid Lines** and **Zero Lines** for **Rows** have been set to **None**.
- The axis has been reversed (right-click the axis and select **Edit**, then check the option to reverse). This allows rank #1 to appear at the top and lower ranks to appear in descending order.
- The axis has been hidden (right-click the axis and uncheck **Show Header**).
- Labels have been edited (by clicking on **Label**) to show on both ends of the line, to center vertically, and to position the rank number next to the state name.

- The year column headers have been moved from the bottom of the view to the top (from the top menu, select **Analysis | Table Layout | Advanced** and uncheck the option to show the innermost level at the bottom).
- A **Highlighter** has been added (using the dropdown on the State field in the view, select **Show Highlighter**) to give the end user the ability to highlight one or more states.

Highlighters give the user the ability to highlight marks in a view by selecting values from the drop-down list or by typing (any match on any part of a value will highlight the mark; so, for example, typing **Carolina** would highlight **North Carolina** and **South Carolina** in the preceding view). Data highlighters can be shown for any field you use as discrete (blue) in the view and will function across multiple views in a dashboard as long as that same field is used in those views.

Slope charts can use absolute values (for example, the actual values of **Sales**) or relative values (for example, the rank of **Sales**, as shown in this example).

If you were to show more than two years of data to observe the change in rankings over multiple periods of time, the resulting visualization might be called a **bump chart** and look like this:

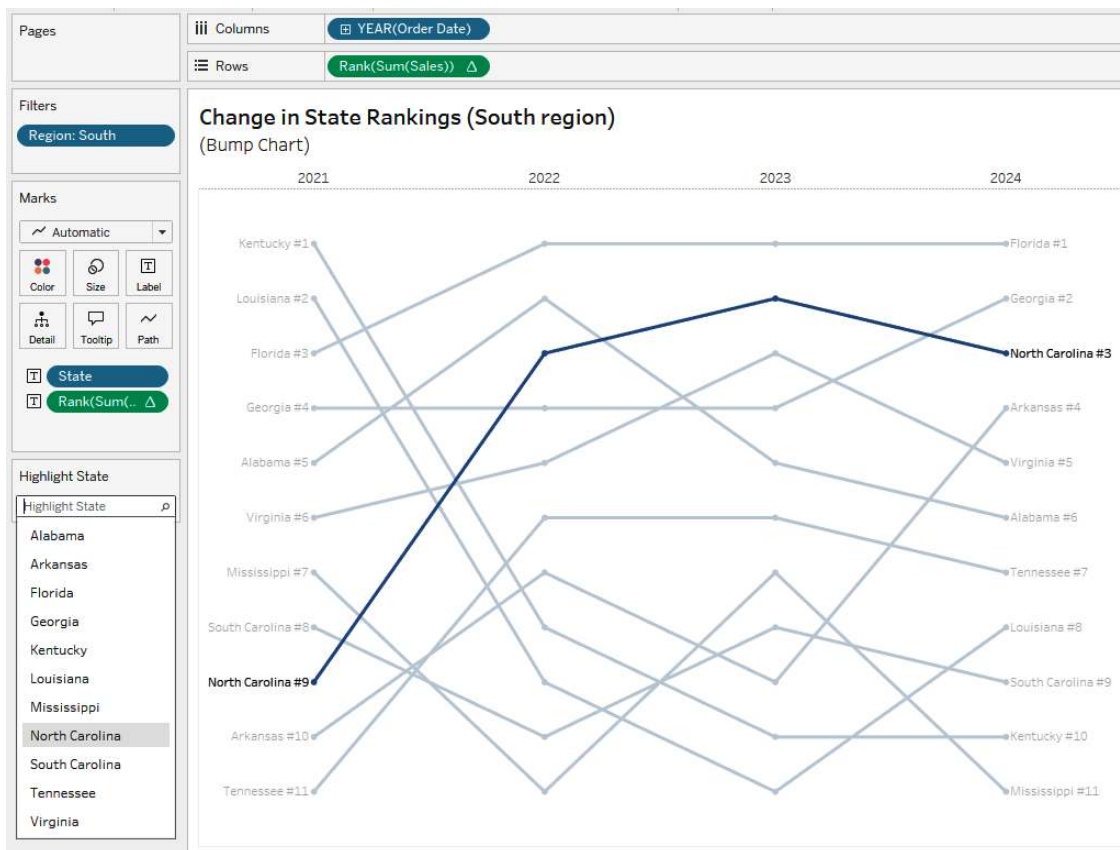


Figure 10.2: This bump chart shows the change in rank for each state over time and makes use of a highlighter

Slope charts are very useful when comparing ranks before and after or from one state to another. Bump charts extend this concept across more than two periods. Consider using either of these two charts when you want to understand changes in rank and make comparisons against that change.

Next, we'll consider a chart that helps us understand the build-up of parts to the whole.

Waterfall charts

A **waterfall chart** is useful when you want to show how parts successively build up to a whole. In the following screenshot, for example, a waterfall chart shows how profit builds up to a grand total across **Department** and **Category** of products. Sometimes profit is negative, so at that point, the waterfall chart takes a dip. Each positive and negative value builds up toward the total:

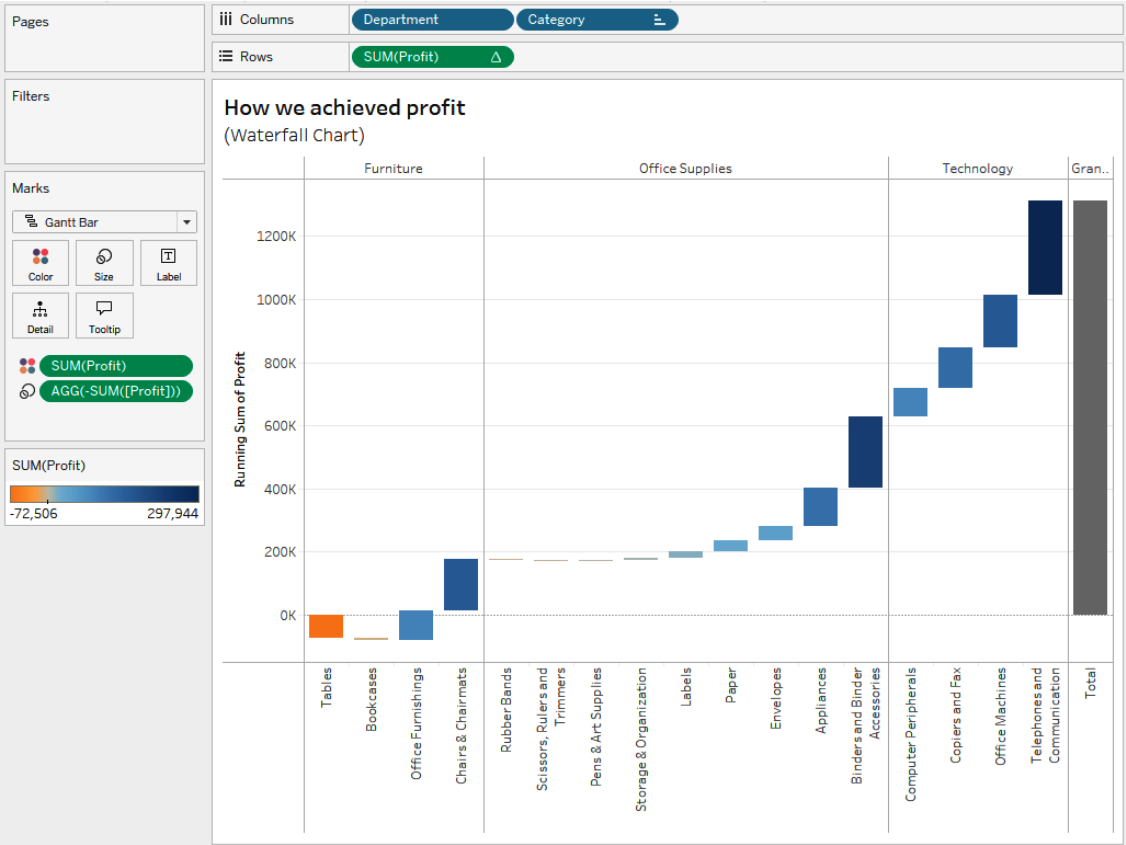


Figure 10.3: This waterfall chart shows how each category adds (or subtracts) profit to build toward the total

Here are the features and techniques used to build the chart:

- The SUM(Profit) field on Rows is a **Running Total** table calculation (created using a **Quick Table Calculation** from the drop-down menu) and is computed across the table.
- **Row Grand Totals** have been added to the view (dragged and dropped from the **Analytics** pane).

- The mark type is set to **Gantt Bar** and an ad hoc calculation is used with code: **SUM(Profit)** for the size. This may seem a bit odd at first, but it causes the Gantt bars to be drawn from the actual value and filled down when the profit is positive or filled up when the profit is negative.
- **Category** has been sorted by the sum of the profit in ascending order so that the waterfall chart builds slowly (or negatively) from left to right within each **Department**. You might want to experiment with the sort options to discover the impact on the presentation.

Waterfall charts will help you demonstrate a build-up or progression toward a total or whole value. Let's next consider step lines and jump lines to show discrete changes over time.

Step lines and jump lines

With a mark type of **Line**, click the **Path** shelf and you'll see three options for **Line Type**:

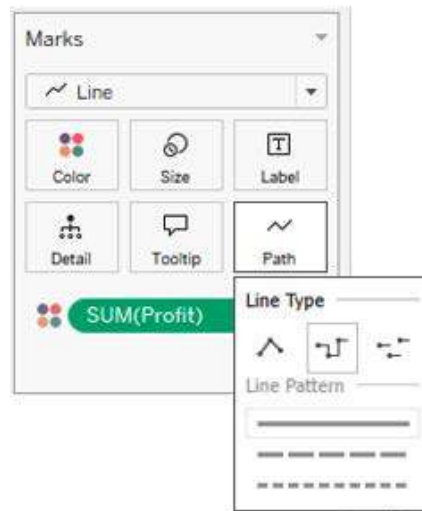


Figure 10.4: Change the type of Line by clicking Path on the Marks card

The three options are:

- **Linear:** Use angled lines to emphasize movement or transition between values. This is the default and every example of a line chart in this book so far has made use of this line type.
- **Step lines:** Remain connected but emphasize discrete steps of change. This is useful when you want to communicate that there is no transition between values or that the transition is a discrete step in value. For example, you might want to show the number of generators running over time. The change from 7 to 8 is a discrete change that might be best represented by a step line.
- **Jump lines:** These are not connected, and when a value changes, a new line starts. Jump lines are useful when you want to show values that indicate a certain state that may exist for a given period of time before jumping to another state. For example, you might wish to show the daily occupancy rate of a hotel over time. A jump line might help emphasize that each day is a new value.

In the following example, we've taken the build-up of profit that was previously demonstrated with a **waterfall chart** and used step lines to show each successive step of profit:

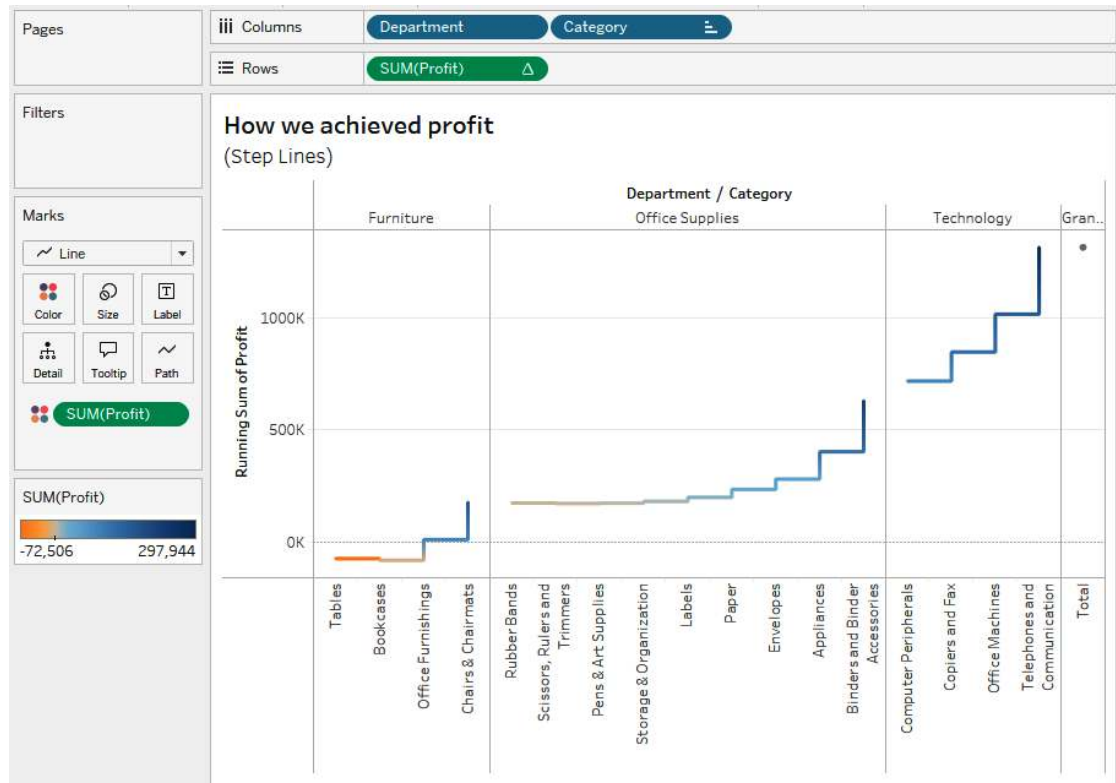


Figure 10.5: A step line chart emphasizes an abrupt change or discrete difference

Experiment with switching line types to see the visual impact and what each communicates about the data.

Let's turn our attention to sparklines as a way to show multiple trends at once.

Sparklines

Sparklines are visualizations that use multiple small line graphs that are designed to be read and compared quickly. The goal of sparklines is to give a visualization that can be understood at a glance. You aren't trying to communicate exact values, but rather give the audience the ability to quickly understand trends, movements, and patterns.

Among various uses of this type of visualization, you may have seen sparklines used in financial publications to compare the movement of stock prices. Recall that in *Chapter 1, Taking Off with Tableau*, we considered the initial start of a **sparklines** visualization as we looked at iterations of line charts. Here is a far more developed example:

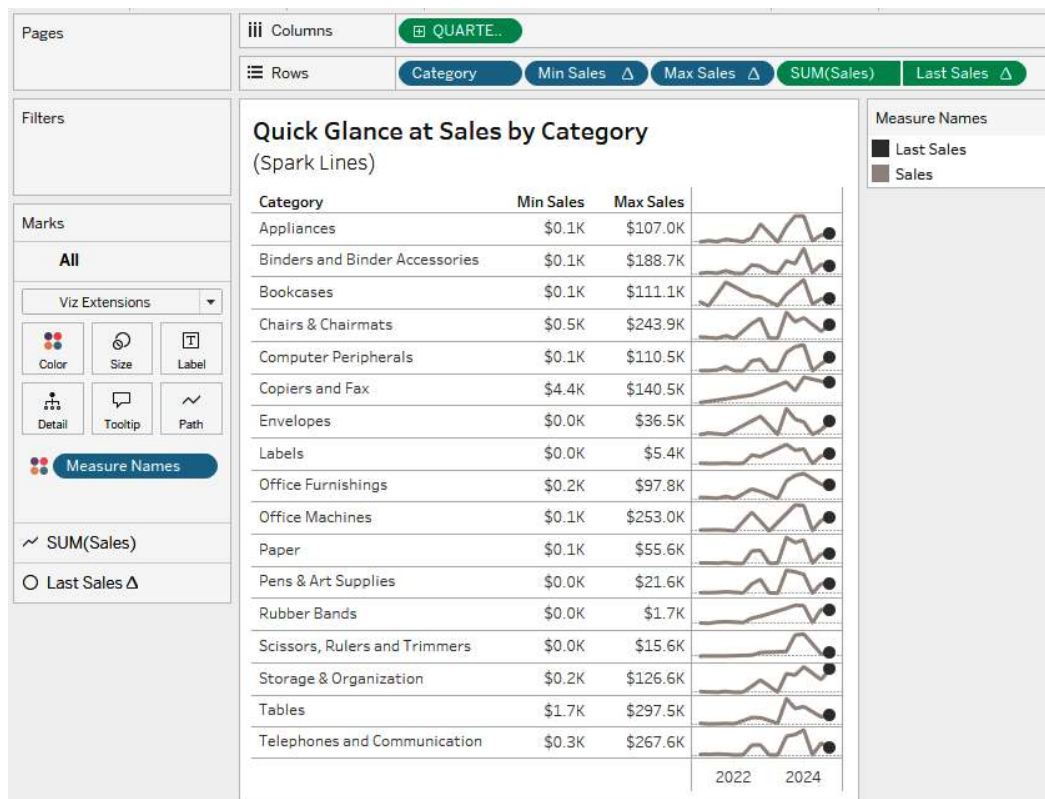


Figure 10.6: Sparklines give you a quick glance at the “shape” of change over time across multiple categories

You can build a chart like this by following these steps:

1. Start with a simple view of SUM(Sales) by Quarter of Order Date (as a date value) with Category on Rows.
2. Create two calculated fields to show the minimum and maximum quarterly sales values for each category. **Min Sales** has the code `WINDOW_MIN(SUM(Sales))` and **Max Sales** has the code `WINDOW_MAX(SUM(Sales))`. Add both to Rows as discrete (blue) fields.
3. Place the **Last Sales** calculation with the code `IF LAST() == 0 THEN SUM([Sales]) END` on Rows and use a synchronized dual axis with a circle mark type to emphasize the final value of sales for each timeline.
4. Edit the axis for SUM(Sales) to have independent axis ranges for each row or column and hide the axes. This allows the line movement to be emphasized. Remember: The goal is not to show the exact values, but to allow your audience to see the patterns and movement.
5. Hide grid lines for Rows.
6. Resize the view (compress the view horizontally and set it to **Fit Height**). This allows the sparklines to fit into a small space, facilitating a quick understanding of patterns and movement.

Sparklines can be used with all kinds of time series to reveal overall big-picture trends and behaviors at a glance.

Dumbbell charts

A **dumbbell chart** is a variation of the circle plot that compares two values for each slice of data, emphasizing the distance between the two values.

Here, for example, is a chart showing the **difference in profit between east and west regions** for each **Category of products**:

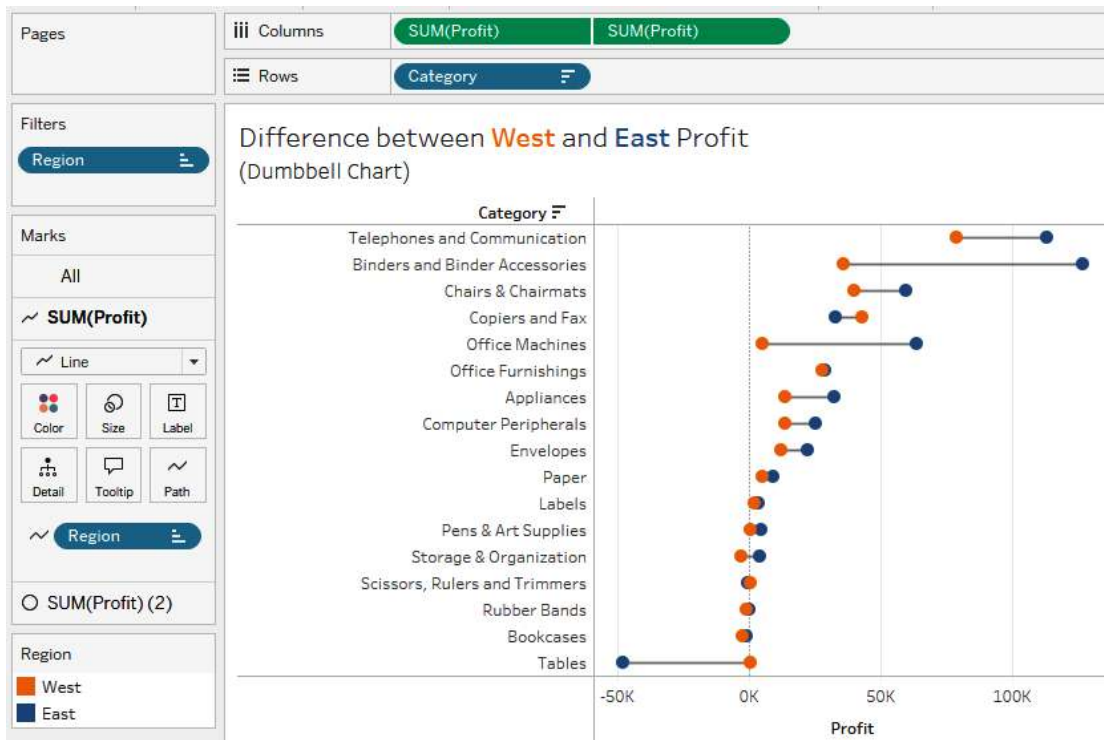


Figure 10.7: A dumbbell chart emphasizes the distance/difference between two values

This chart was built using the following features and techniques:

- A synchronized dual axis of **SUM(Profit)** has been used with one set to mark the type of **Circle** and the other set to **Line**
- **Category** has been sorted by **Profit** descending (the sort sums the profit for both the East and West regions)
- **Region** has been placed on the **Path** shelf for the line to tell Tableau to draw a line between the two **regions**

The **Path** shelf is available for the **Line** and **Polygon** mark types. When you place a field on the **Path** shelf, it tells Tableau the order to connect the points (following the sort order of the field placed on **Path**). Paths are often used with geographic visualizations to connect origins and destinations on routes but can be used with other visualization types. Tableau draws a single line between two values (in this case, East and West).

- **Region** is placed on **Color** for the circle mark type

Dumbbell charts are great at highlighting the disparity between values. Let's next consider how we can use unit/symbol charts to drive responses.

Unit/symbol charts

A **unit chart** can be used to show individual items, often using shapes or symbols to represent each individual. These charts can elicit a powerful emotional response because the representations of the data are less abstract and more easily identified as something real. For example, here is a chart showing how many customers had late shipments for each **Region**:

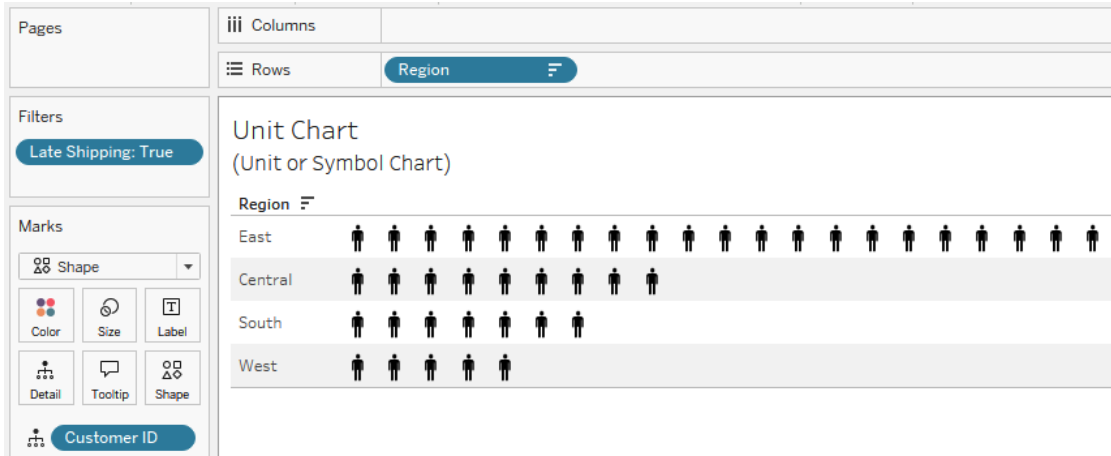


Figure 10.8: Each image represents a real person and is less abstract than circles or squares

The view was created with the following techniques:

1. The view is filtered where **Late Shipping** is **True**. **Late Shipping** is a calculated field that determines if it took more than 14 days to ship an order. The code is as follows:

```
DATEDIFF('day', [Order Date], [Ship Date]) > 21
```

2. **Region** has been sorted by the distinct count of **Customer ID** in descending order.
3. **Customer ID** has been placed on **Detail** so that there is a mark for each distinct customer.
4. The mark type has been changed to **Shape** and the shape has been changed to the included person shape in the **Gender** shape palette. To change shapes, click the **Shape** shelf and select the desired shape(s), as shown in the following screenshot:

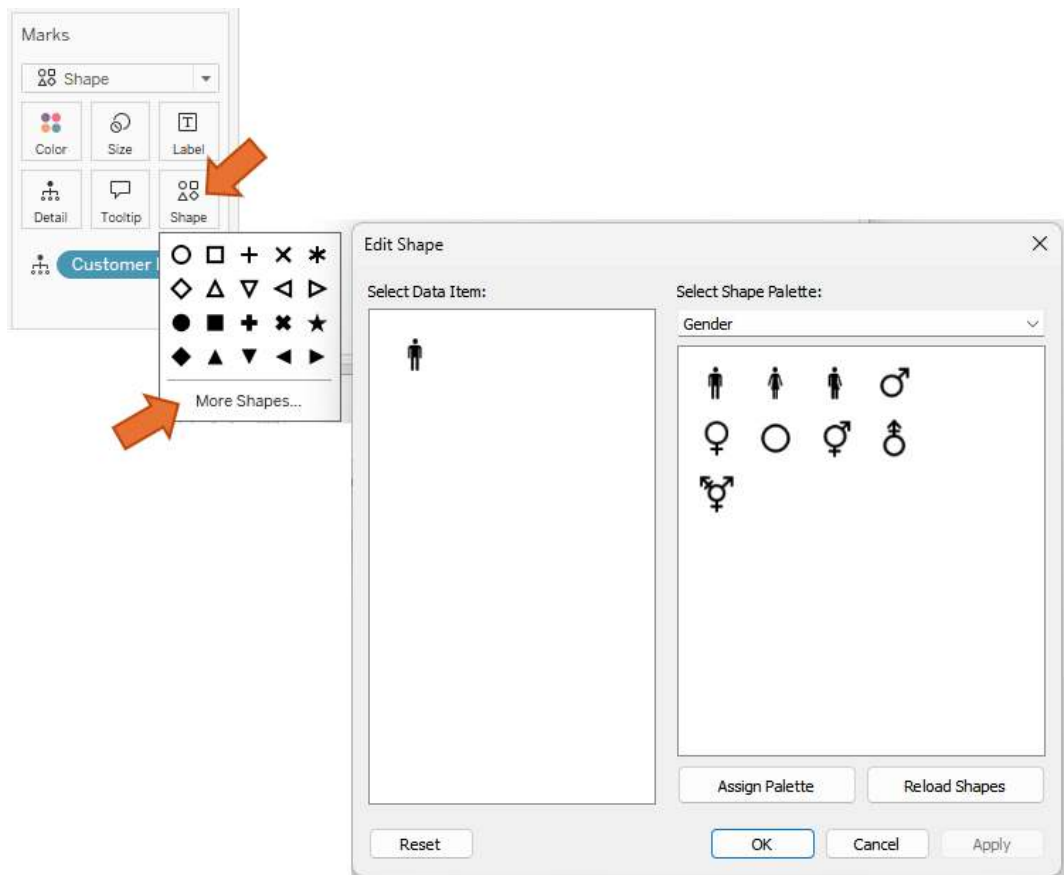


Figure 10.9: You can assign shapes to dimensional values using the Shape shelf

The preceding unit chart might elicit more of a response from regional managers than a standard bar chart when they are gently reminded that poor customer service impacts real people. Granted, the shapes are still abstract, but they more closely represent an actual person. You could also consider labeling the mark with the customer name or using other techniques to further engage your audience and bridge the gap between abstract concepts and real people.



Remember that normally in Tableau, a mark is drawn for each distinct intersection of dimensional values. So, it is rather difficult to draw, for example, 10 individual shapes for a single row of data that simply contains the value 10 for a field. This means that you will need to consider the shape of your data and include enough rows to draw the units you wish to represent.

Concrete shapes, in any type of visualization, can also dramatically reduce the amount of time it takes to comprehend the data. This is sometimes referred to as **reducing the cognitive load**. Contrast the amount of effort required to identify the departments in these two scatterplots:

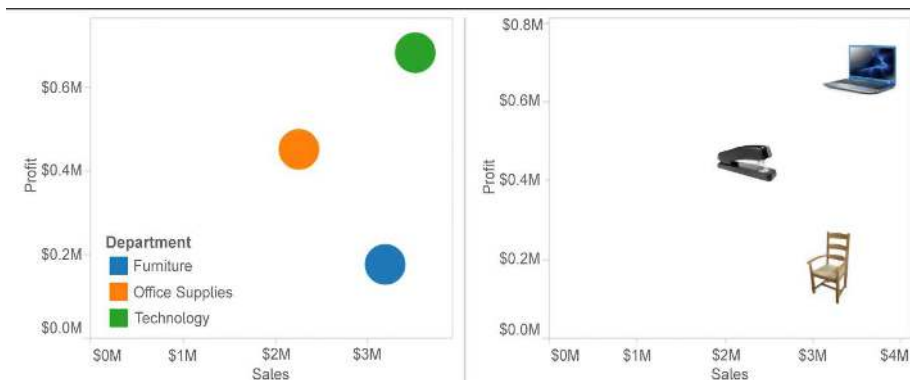


Figure 10.10: Notice the difference in “cognitive load” between the left chart and the right

Once you know the meaning of a shape, you no longer have to reference a legend. Placing a discrete field on the **Shape** shelf allows you to assign shapes to individual values of the field.

Shapes are images located in the My Tableau Repository\Shapes directory. You can include your own custom shapes in subfolders of that directory by adding folders and image files.

If your data contains (or you supplement it with a calculation) a field containing a URL to an image, you can define the field as an image that can be used in headers. In the data pane, use the dropdown of the field to select **Image Role | URL**. When you use the field on Rows or Columns, the image will be shown rather than text. Here, for example, is a line chart with department headers that have an image role:

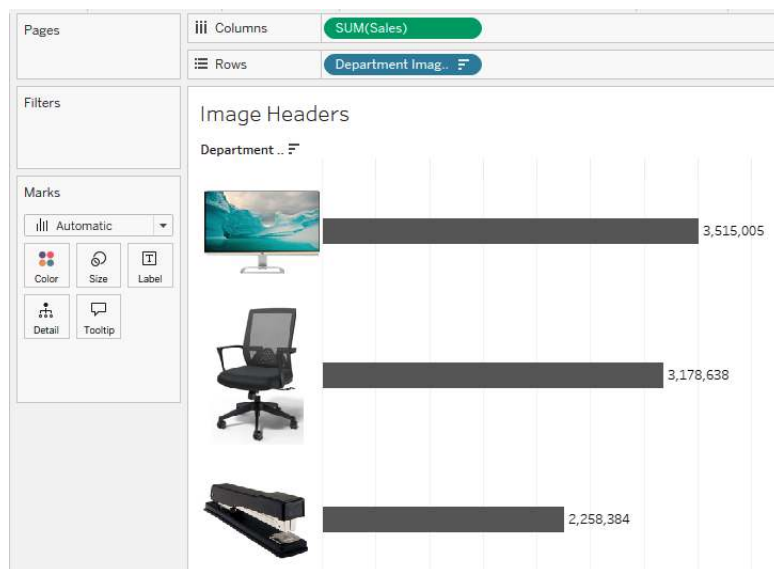


Figure 10.11: The Rows field defines the URL for images for each role. Instead of text, the image is shown in the header.

Consider using this feature for product catalogs, HR directories, or anywhere you can make the abstract more real. Always consider policies and ethics in your design. But where appropriate, images can drive empathetic decision-making.

Marimekko charts

A **Marimekko chart** (sometimes also called a **Mekko chart**) is similar to a vertically stacked bar chart, but additionally uses varying widths of bars to communicate additional information about the data. Here, for example, is a **Marimekko chart** showing the breakdown of sales by region and department.

The width of the bars communicates the total **Sales for Region**, while the height of each segment gives you the percentage of sales for the **Department** within the **Region**:

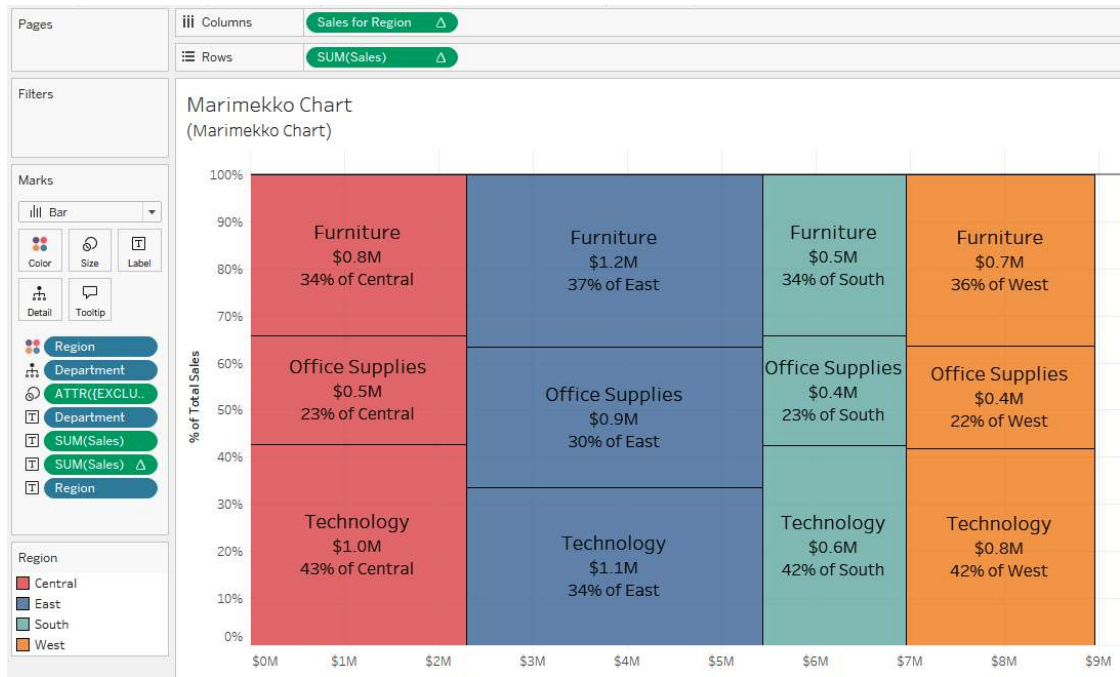


Figure 10.12: The amount of sales per Department is indicated by the height of each segment, while the width of each bar indicates the overall sales per Region

Creating a **Marimekko chart** in Tableau leverages the ability to fix the width of bars according to the axis' units.

Clicking the **Size** shelf when a continuous (green) field is on **Columns** (thus defining a horizontal axis) and the mark type is set to **Bar** reveals options for a fixed size. You can manually enter a **Size** and **Alignment** or drop a field on the **Size** shelf to vary the width of the bars.

Here are the steps required to create this kind of visualization:

1. The mark type has been specifically set to **Bar**.
2. **Region** and **Department** have been placed on **Color** and **Detail**, respectively. They are the only dimensions in the view, so they define the view's **level of detail**.
3. **Sales** has been placed on **Rows** and a **Percent of Total** quick table calculation has been applied. **Compute Using** (addressing) has been set to **Department** so that we get the percentage of sales for each department within the partition of the **Region**.
4. The calculated field **Sales for Region** calculates the *x*-axis location for the right-side position of each bar. The code is as follows:

```
IF FIRST() = 0
    THEN MIN({EXCLUDE [Department] : SUM(Sales)})
ELSEIF LOOKUP(MIN([Region]), -1) <> MIN([Region])
    THEN PREVIOUS_VALUE(0) + MIN({EXCLUDE [Department] : SUM(Sales)})
ELSE
    PREVIOUS_VALUE(0)
END
```

While this code may seem daunting at first, it follows a logical progression. Specifically, if this is the first bar segment, we'll want to know the sum of **Sales** for the entire region (which is why we exclude **Department** with an inline level-of-detail calculation). When the calculation moves to a new **Region**, we'll need to add the previous **Region** total to the new **Region** total. Otherwise, the calculation is for another segment in the same **Region**, so the regional total is the same as the previous segment. In this case, we'll set the **Compute Using** option to **Region** to enable the logical progression to work as expected.

Finally, a few additional adjustments were made to the view:

1. The field on **Size** is an ad hoc level-of-detail calculation with the code {EXCLUDE [Department] : SUM(Sales)}. As we mentioned earlier, this excludes **Department** and allows us to get the sum of sales at the **Region** level. This means that each bar is sized according to the total sales for the given **Region**.
2. Clicking on the **Size** shelf and switching to **Fixed** gives the option to set the alignment of the bars to the right. Since the preceding calculation gave the right position of the bar, we need to make certain that the bars are drawn from that starting point. This is illustrated in the following screenshot:

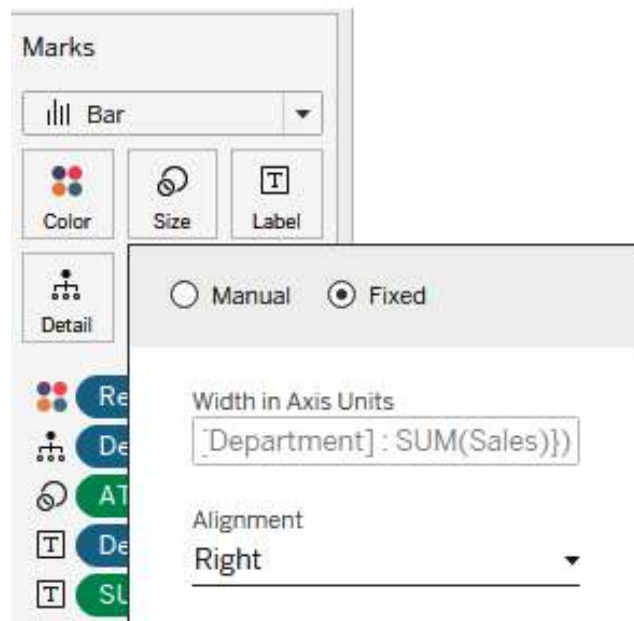


Figure 10.13: Setting the Size to Fixed and Alignment to Right

3. Various fields, such as **SUM(Sales)** as an absolute value and percentage, have been copied to the **Label** shelf so that each bar segment more clearly communicates meaning to the viewer.

To add labels to each **Region** column, you might consider creating a second view and placing both on a dashboard. Alternatively, you might use annotations.

In addition to allowing you to create Marimekko charts, the ability to control the size of bars in axis units opens all kinds of possibilities for creating additional visualizations, such as complex cascade charts or stepped area charts. The techniques are like those used here. You may also leverage the sizing feature with **continuous bins** (use the drop-down menu to change a bin field in the view to continuous from discrete).



For a more comprehensive discussion of Marimekko charts, along with approaches that work with sparse data, see Jonathan Drummey's blog post at <https://www.tableau.com/about/blog/2016/8/how-build-marimekko-chart-tableau-58153>.

Sunburst diagram with Viz Extensions

We looked at an example of Viz Extensions in *Chapter 3, Moving Beyond Basic Visualizations*. Here, we'll briefly consider another example that illustrates the power and flexibility of Tableau Viz Extensions to make it easy to create non-standard visualizations.



Third-party Viz Extensions are often hosted on third-party servers, and some require additional licensing for full-feature access. Detailed descriptions of where the extensions are hosted, what level of access it has to your data, and licensing information are available when you first add the extension. You'll want to consider your organization's policies before using these extensions.

You may host some Viz Extensions locally. As of the time of writing, only the Sankey and Tableau Table Viz Extensions from Tableau are **sandboxed**. This means they are self-contained with Tableau's environment.

Consider the following sunburst diagram and how it shows the breakdown and hierarchy of departments and categories:

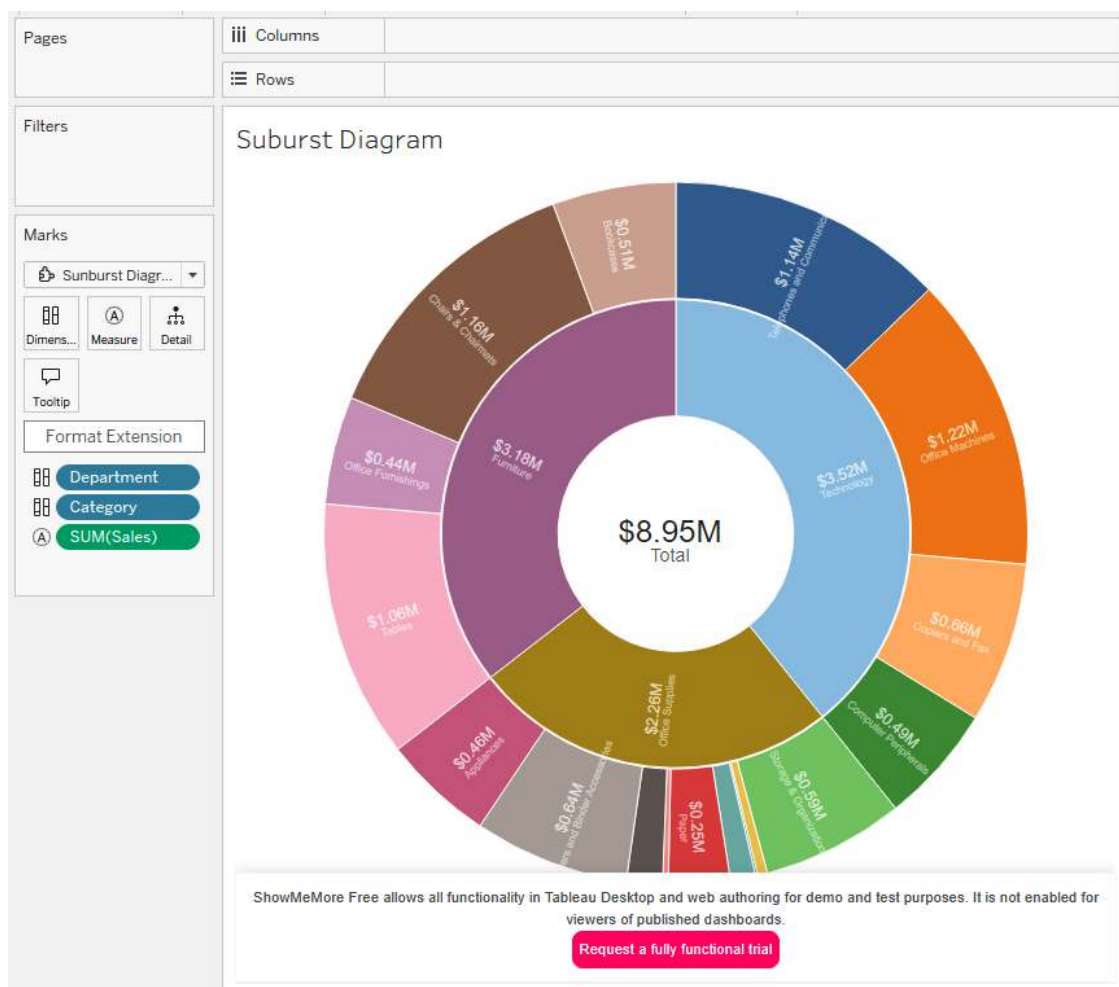


Figure 10.14: A sunburst diagram made possible with Viz Extensions

Department and **Category** are both used as **Dimensions** in the view, while **SUM(Sales)** is used as the **Measure**. The sunburst diagram shows the first dimension as the inner ring and expands outward with subsequent dimensions. Formatting options and information about Viz Extensions are available upon clicking **Format Extension**.

Take a few minutes to explore various Viz Extensions available through Tableau Exchange (easily accessible by clicking **Add Extension** from the drop-down menu on the **Marks** card). Sunburst diagrams, radial charts, chord diagrams, and many more are only a few clicks away!

In addition to the various types of visualizations we've already considered, we'll also look at some ways to leverage animation in your data storytelling. That's where we turn our attention next.

Animated visualizations

Previous versions of Tableau allowed rudimentary animation using the **Pages** shelf with playback controls. Newer versions of Tableau include **Mark Animation**, which means marks smoothly transition when you apply filters, sorting, or page changes. Consider leveraging animation to extend your analytical potential in a couple of ways:

- Use animations while exploring and analyzing your data. This allows you to gain analytical insights you might otherwise miss, such as seeing how far and in which direction marks in a scatterplot move as a filter changes.
- Use it strategically to enhance the data story. Animation can be used to capture interest, draw attention to important elements, or build suspense toward a conclusion.

We'll consider both approaches to animation in the following examples.

Enhancing analysis with animation

Many visualizations in Tableau employ animation for fluid movement that illustrates change as filters are applied. Marks either move or fade in/fade out depending on the filters applied. In many cases, this subtly allows you to comprehend the changes and relationships within the data more than an abrupt redrawing of a visualization.

Consider the following scatterplot included in the sample workbook:

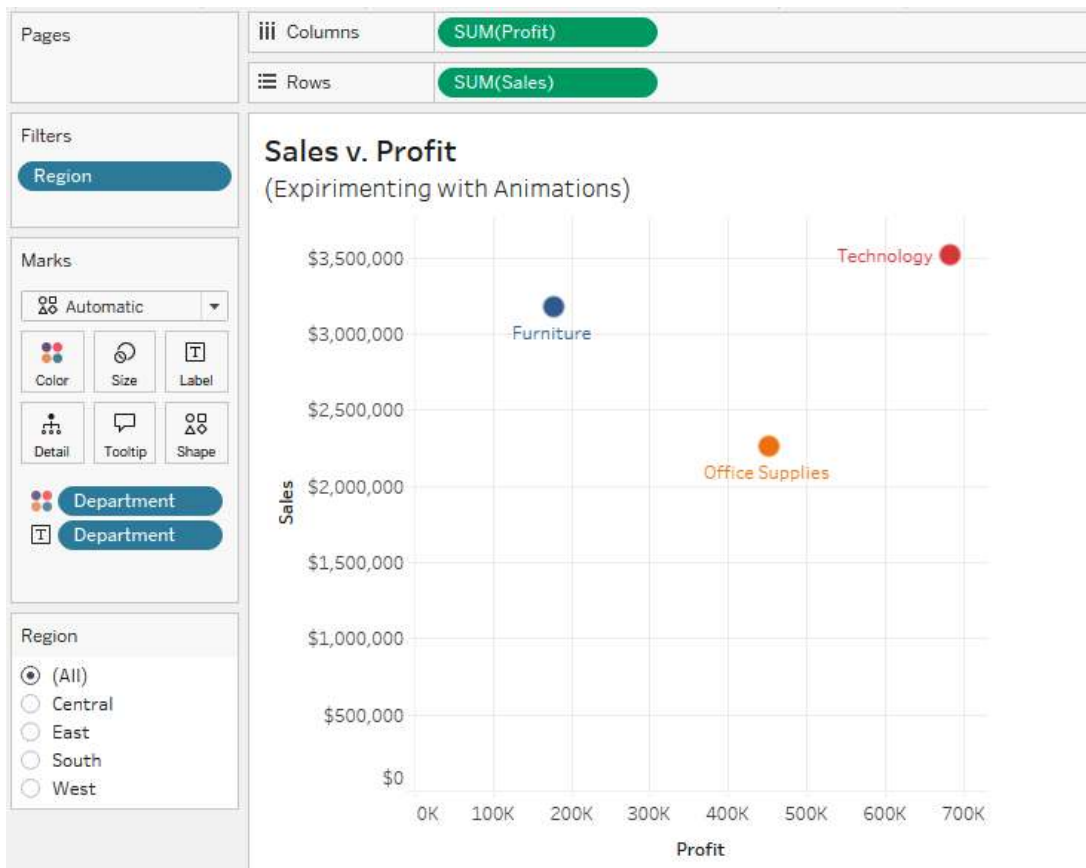


Figure 10.15: Sales and profit per Department

Notice the **Region** filter. Change the filter selection a few times in the Chapter 10 workbook. The animation has been turned off. You'll observe what occurs without animations: the circle marks are immediately redrawn at the new location determined by the filter. This works well, but there is a bit of a disconnect between the filter settings. As you switch between regions, notice the mental difficulty in keeping track of where a mark was versus where it is with the new selection. Did one region's mark increase in profit? Did it decrease in sales?

Now, turn on animations for the view. To do this, use the menu to select **Format | Animations....** The **Animations** format pane will show on the left. Use it to turn **On** the selected sheet:

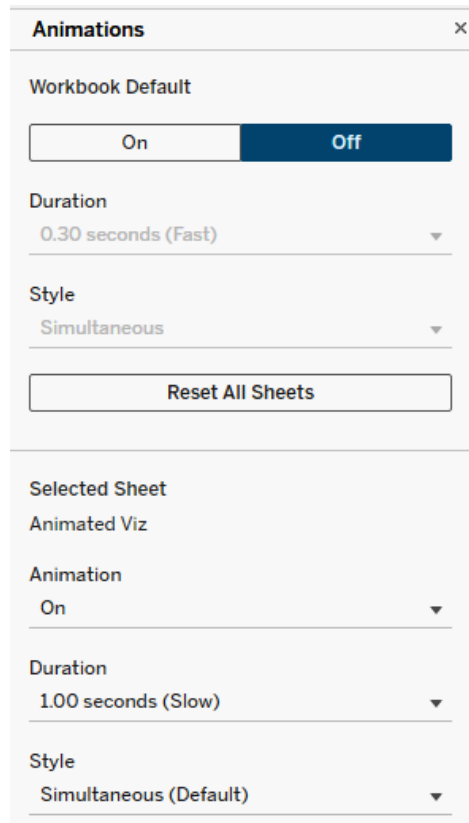


Figure 10.16: The Animations format pane gives various options for workbook and individual sheet animation settings

Experiment with various duration settings and change the filter value. Notice how much easier it is to see the change in sales and profit from region to region. This gives you the ability to notice changes more easily. You'll start to gain insight, even without spending a lot of cognitive effort, into the magnitude and direction of change. Animations provide a path to this analytical insight.

Enhancing data storytelling with animation

Beyond providing analytical insight as you perform your data discovery and analysis, you can also leverage animation to more effectively drive interest and highlight decision points, opportunities, or risks in your data stories.

As an example, consider this view in the Chapter 10 workbook:

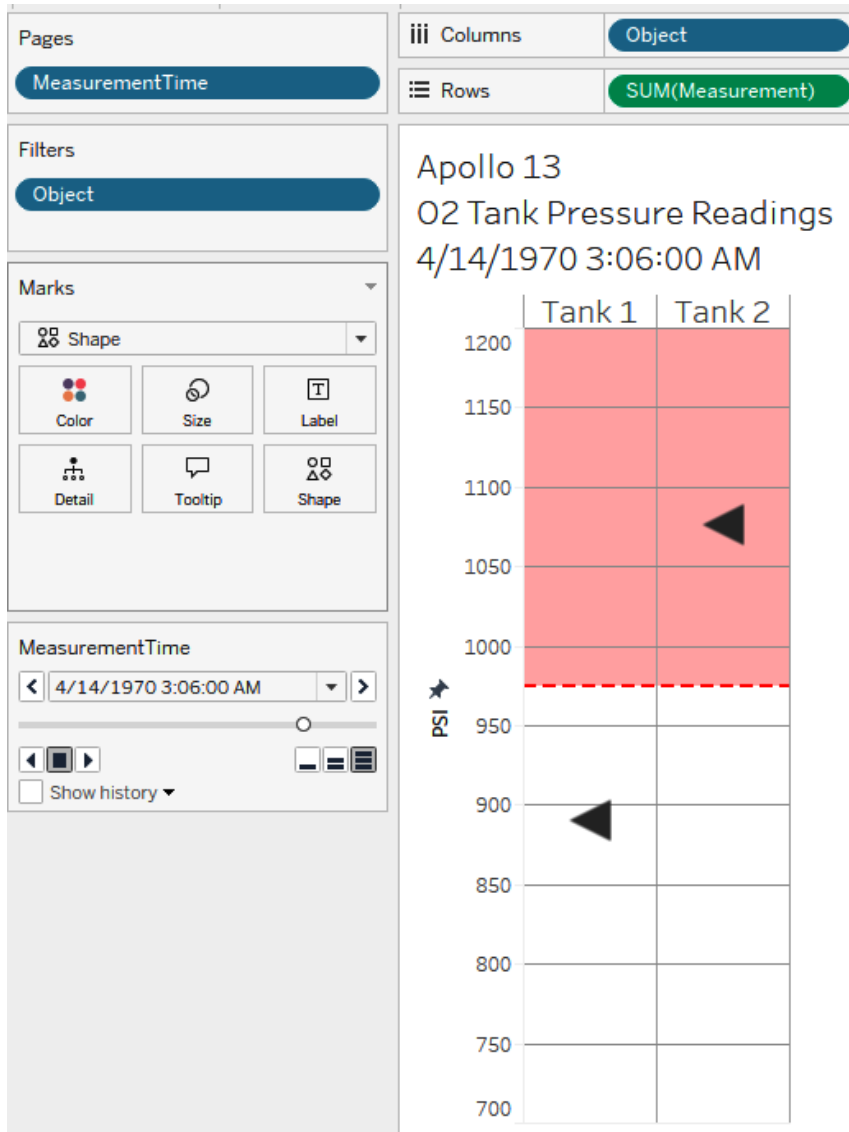


Figure 10.17: O2 Tank 1 and Tank 2 pressure readings over time during the Apollo 13 mission

The view tells a part of the story of Apollo 13 and the disaster that crippled the spacecraft. It does this by making use of both the **Pages** shelf and smooth animation. Experiment with the animation speed and playback controls in the Chapter 10 workbook. Consider how animation can be used to heighten awareness, drive interest, or even create suspense.

When you use multiple views on a dashboard, each having the same combination of fields on the **Pages** shelf, you can synchronize the playback controls (using the caret drop-down menu on the playback controls) to create a fully animated dashboard.

Animations can be shared with other users of Tableau Desktop and are displayed on Tableau Server, Tableau Online, and Tableau Public.

Summary

We've covered a wide variety of advanced visualization types in this chapter! We've considered slope and bump charts that show changes in rank or value, step and jump lines that show discretely changing values, and unit charts that help materialize abstract concepts.

There is no way to cover every possible visualization type. Instead, the aim was to demonstrate some of what can be accomplished and spark new ideas and creativity. As you experiment and iterate through new ways of looking at data, you'll become more confident in how to best communicate the stories contained in the data. Next, we'll return briefly to the topic of dashboards to see how some advanced techniques can make them truly dynamic.

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search for this book by name.



Note: Keep your purchase invoice ready before you start.

11

Dynamic Dashboards

We looked at dashboards previously, in *Chapter 8, Telling a Data Story with Dashboards*, and considered how to make them interactive using a variety of actions. Now, we'll expand on that concept to consider how to create truly dynamic dashboards—dashboards where objects can be shown and hidden. This enables you to do all kinds of amazing things, such as allow the user to select which type of visualization to see or to dynamically hide or show controls that are relevant or desired.

We'll consider the following types of dynamic interactivity:

- Show/Hide buttons
- Dynamic Zone Visibility

Let's start by discussing how to show and hide content on dashboards using Show/Hide buttons.

Show/Hide buttons

Show/Hide buttons allow you to show and hide layout containers (and everything in them!). You'll find a couple of examples in the *Chapter 11 Complete* workbook. The *Chapter 11 Starter* workbook has the dashboards, but not the Show/Hide buttons or containers. The images used in the second example are also in the *Chapter 11* directory.

To use the Show/Hide buttons, add a layout container to a dashboard as a **floating object** as shown in *Figure 11.1*:

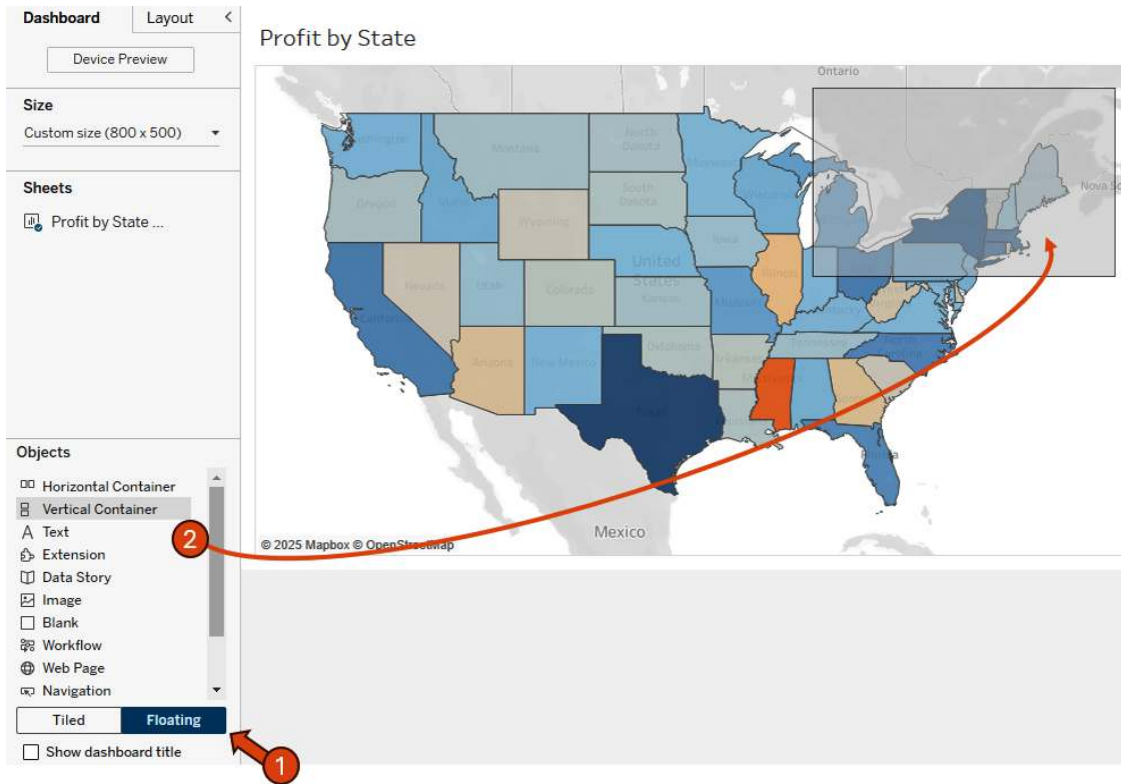


Figure 11.1: Add a floating layout container by selecting Floating and then drag and drop the layout container onto the dashboard

After you have added the layout container, use the dropdown to select **Add Show/Hide Button**, as shown here in *Figure 11.2*:

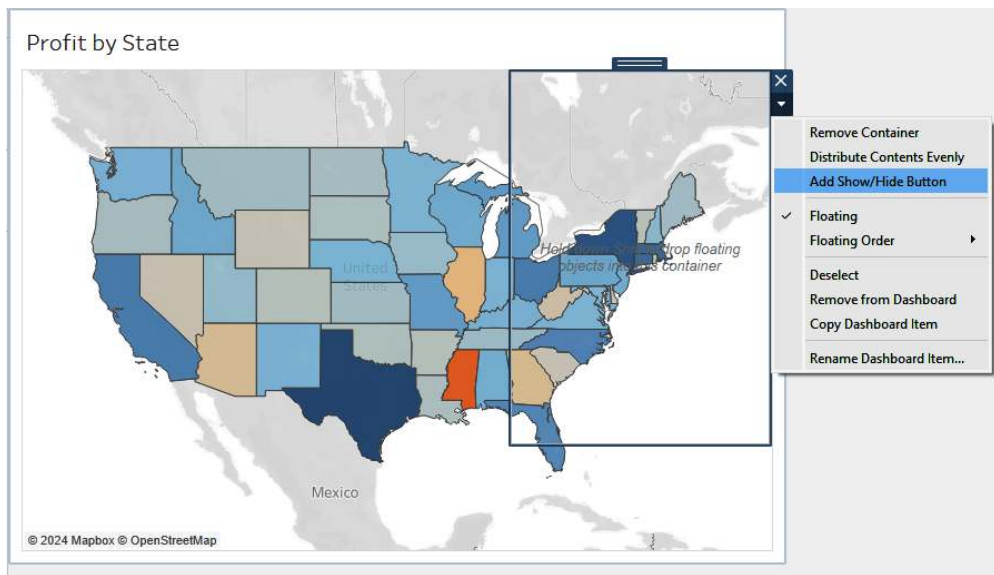


Figure 11.2: A dashboard with a single map view and a floating layout container

On this dashboard, we've added a vertical layout container floating over the map. When we select **Add Show/Hide Button**, Tableau adds a small button to the dashboard:



Figure 11.3: The Show/Hide button for the layout container

The button can be moved, resized, and formatted as desired. A Show/Hide button can be applied to any single layout container on the dashboard.



Use the **Rename Dashboard Item...** option on the layout container dropdown to make it easy to identify which layout container is the target of a Show/Hide button.

You may edit the button's appearance and actions using the button's dropdown to select **Edit Button...**:

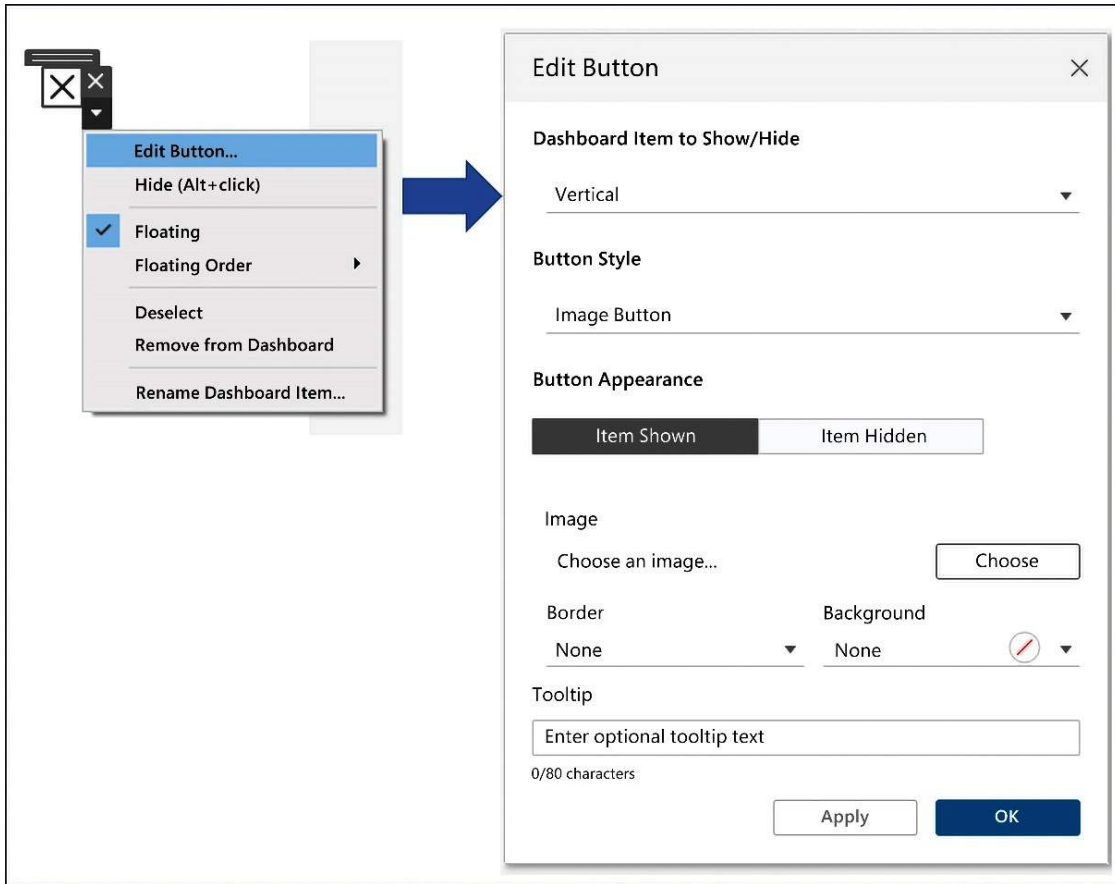


Figure 11.4: Selecting Edit Button... reveals many options for altering the button's behavior and appearance

In the resulting dialog box, you'll have options for selecting which layout container to show or hide, the style of the button (image or text), and which images or text to show when the container is shown or hidden.

In this example, the Show/Hide button uses the default **Image Button** style (as seen in *Figure 11.3*) and the layout container has been filled with a couple of filters and the map's legend.



You may include any dashboard objects in a container, including filters, legends, parameter and set controls, and even other views!

For its part, the container has been formatted with a border and a semi-transparent background so that the map is slightly visible beneath it. This was accomplished by selecting the container and then using the **Layout** tab of the **Dashboard** pane, as shown here:

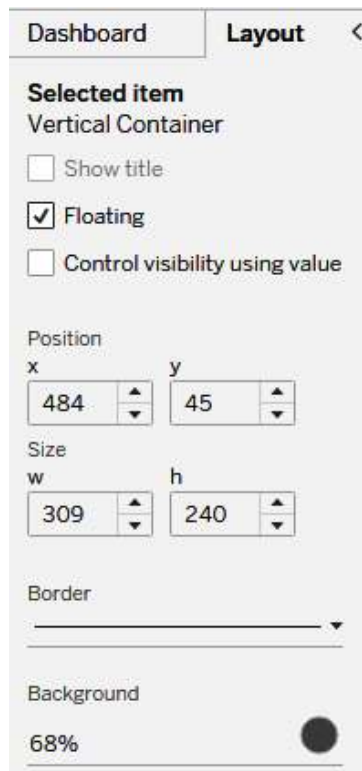


Figure 11.5: Use the Layout pane to adjust options for any selected dashboard object

The result, demonstrated in the Chapter 11 workbook, is a dashboard with a Show/Hide button that makes it clear to the user that additional options can be shown or hidden:

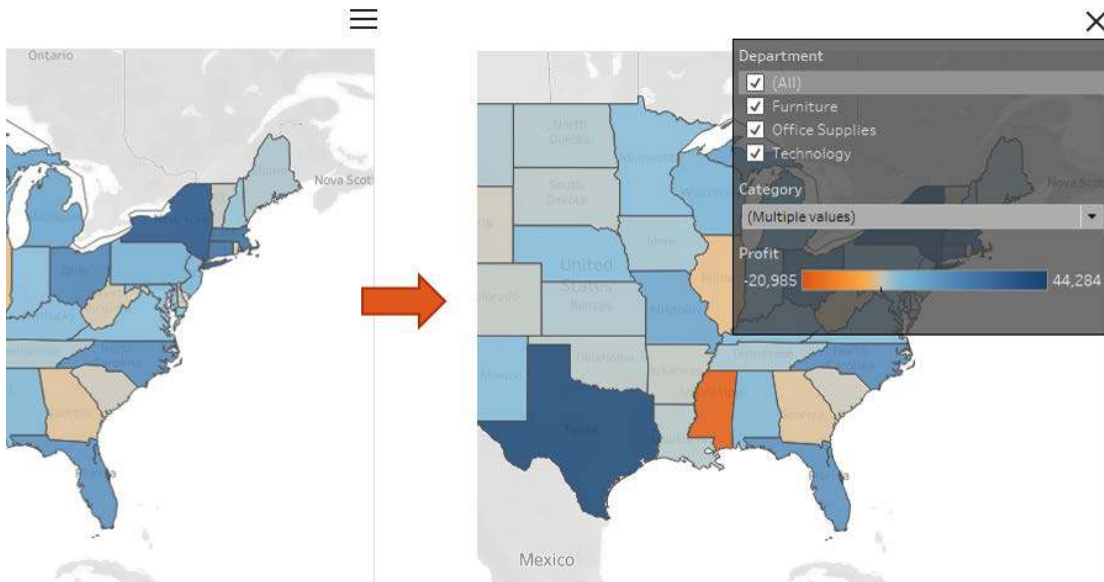


Figure 11.6: Here, you can see the layout container's hidden and shown states along with the changing image of the button

When working in Tableau Desktop, you can switch between **Design Mode** and **Presentation Mode** by pressing **F11** or using the Presentation Mode button on the toolbar.



Figure 11.7: Click the Presentation Mode button on the toolbar to enter presentation mode

Presentation Mode hides all design tools and menus and allows you to show the dashboard or test it as it would function when published. Buttons will function a bit differently depending on the mode. In **Design Mode**, clicking the button allows you to access various menus and controls associated with designing the button. If you want to actually trigger the button's action, use **Alt + Click** while in **Design Mode** or switch to **Presentation Mode** where a single click will trigger the button.

Show/Hide buttons give you a lot of flexibility in your dashboard design, from optimizing space and simplifying the display, to removing filters and other clutter from printed output.

Let's look at another example in the Chapter 11 Complete workbook that illustrates the possibility of showing and hiding instructions:

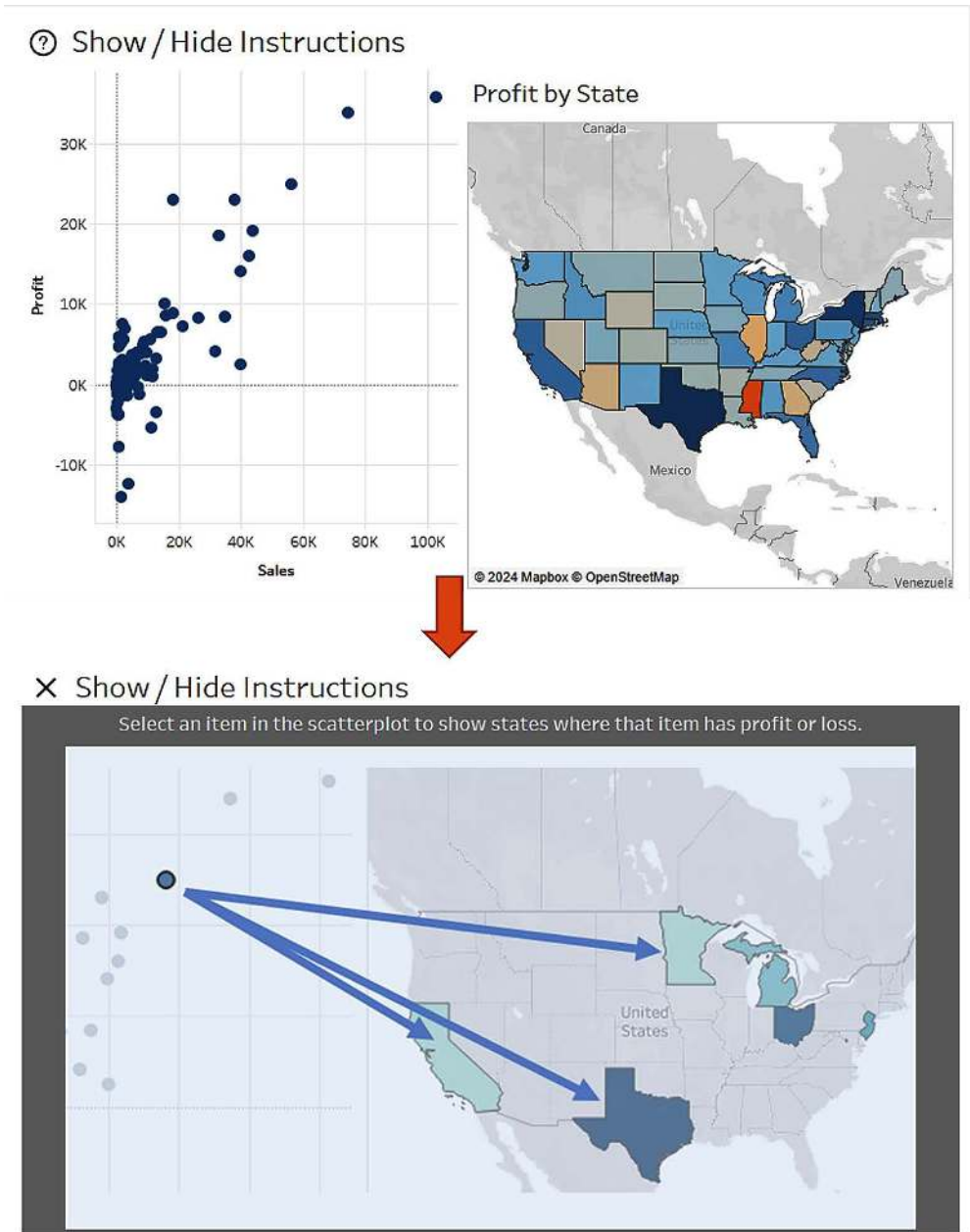


Figure 11.8: Clicking the question mark button reveals instructions for how to use the dashboard

In this case, the default button image has been replaced with a question mark icon. This is accomplished by editing the image as illustrated in Figure 11.3 and changing the image (questionmark.png is included in the Chapter 11 directory).

When the user clicks the button, a floating container appears over the top of the entire dashboard. It includes text as well as an annotated image to instruct the user on how to use the dashboard. In this simple example, it might be overkill to supply instructions in this way, when a simple text label would suffice. However, as you build more complex dashboards or have detailed analyses that require explanation, this is a great way to give users additional instructions and details that would otherwise clutter a dashboard.



Consider also using Show/Hide buttons to allow user interactivity that you do not want to include in PDFs or printed pages of the dashboards. Or you may simply want to get them out of the way for the end user of a dashboard – so they can get to them when needed but can focus on the data without clutter. Filters and parameters are incredibly useful, but you don't always want them to show up in a polished output.

Let's move on to discuss another method of dynamically showing, hiding, and even swapping out content on a dashboard.

Dynamic Zone Visibility

Dynamic Zone Visibility allows you to show and hide objects on a dashboard based on a parameter or field value. This technique allows you to build rich and dynamic dashboards.

Let's discuss a simple example of a dashboard that allows the user to switch a view between a bar chart and a map. We'll accomplish this with a parameter and then apply visibility based on the parameter to the applicable views.

Dynamic Zone Visibility is turned on by selecting an object on a dashboard and then checking the option on the Layout pane:

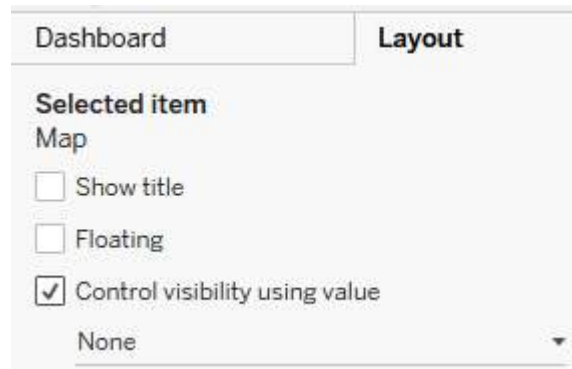


Figure 11.9: Turning on dynamic zone visibility

The drop-down option allows you to select a parameter or field that specifies whether the object is visible or not. The value of the parameter or field must be a Boolean. A true value will cause the object to be visible and a false value will cause the object to be invisible.

For our simple example, we have a view of profit by state named **Map** and a view of profit by department and category called **Bar Chart**. We'll start by creating a parameter that will give the end user the option of which they want to see:

Edit Parameter [Show Sheet]X

Name

Show Sheet

Properties

Data type

String

Display format

Map

Current value

Map

Value when workbook opens

Current value

Allowable values

☐ All

☒ List

☐ Range

Value	Display As
Bar Chart	Bar Chart
Map	Map
Click to add	

☒ Fixed

☐ When workbook opens

Add values from

Remove Selected

Cancel

OK

Figure 11.10: A parameter named *Show Sheet* that gives the options *Bar Chart* or *Map*

This parameter specifies the options as a **String** data type. In itself, it cannot control the visibility of an object. But we can create a couple of Boolean calculations:

- **Show Map** with the code [Show Sheet] = "Map"
- **Show Bar Chart** with the code [Show Sheet] = "Bar Chart"

This gives us everything we need to dynamically show and hide the objects in the dashboard. We'll start with the parameter, the Map, and the Bar Chart in a Horizontal Layout container, like this:

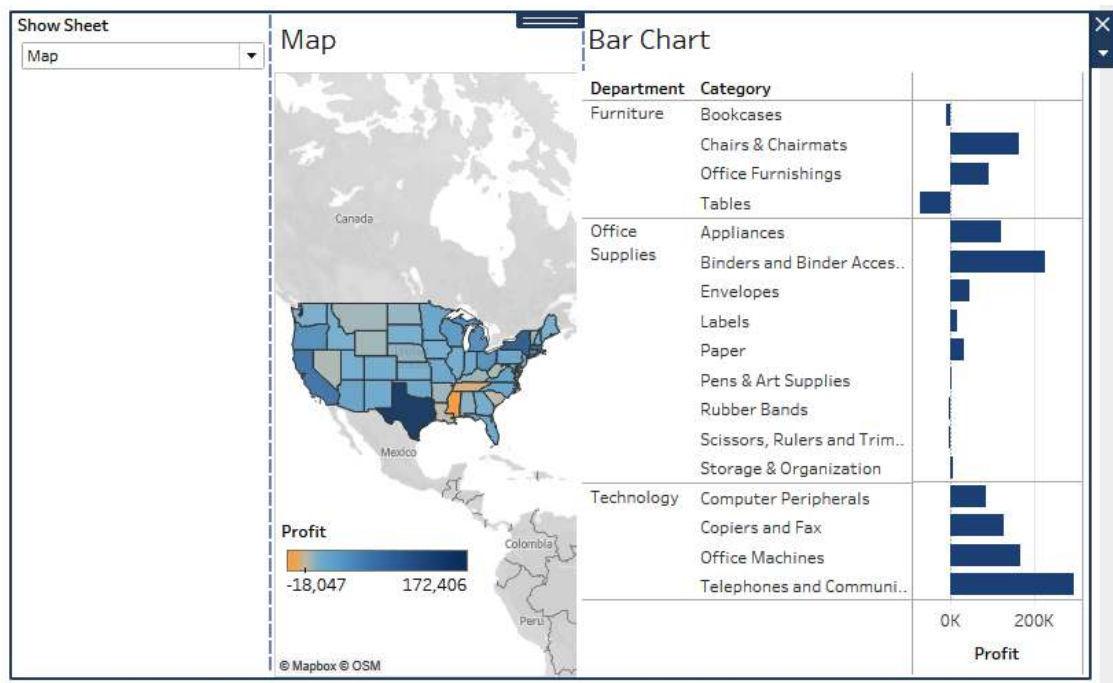


Figure 11.11: The parameter, Map, and Bar Chart are all in a single layout container



Dynamic Zone Visibility will work on containers, objects in containers, and floating objects. Here, placing the Map and Bar Chart in a single layout container will allow the displayed view to expand to fill the empty space left when one of the views is hidden.

We'll turn on Dynamic Zone Visibility by selecting each object and setting the appropriate field to control whether it is shown or not. Both the **Map** and the **Profit** legend should show when the Show Map calculation is true. So, we'll adjust the settings on the Layout pane as follows:

The figure shows the 'Layout' pane for a dashboard. It has two tabs: 'Dashboard' and 'Layout'. The 'Layout' tab is selected. Under the 'Selected item' section, 'Map' is chosen. The following settings are visible:

- ☒ Show title
- ☐ Floating
- ☒ Control visibility using value

Below these settings is a dropdown menu labeled 'Show Map'.

Figure 11.12: The Map and the Profit legend will show when Show Map is true



Quick tip: Need to see a high-resolution version of this image? Open this book in the next-gen Packt Reader or view it in the PDF/ePub copy.

The next-gen Packt Reader and a free PDF/ePub copy of this book are included with your purchase. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



Similarly, we'll select the bar chart and set it to show only when Show Bar Chart is true.

The result is a dynamic experience in which the end user can choose how they want to see the data:

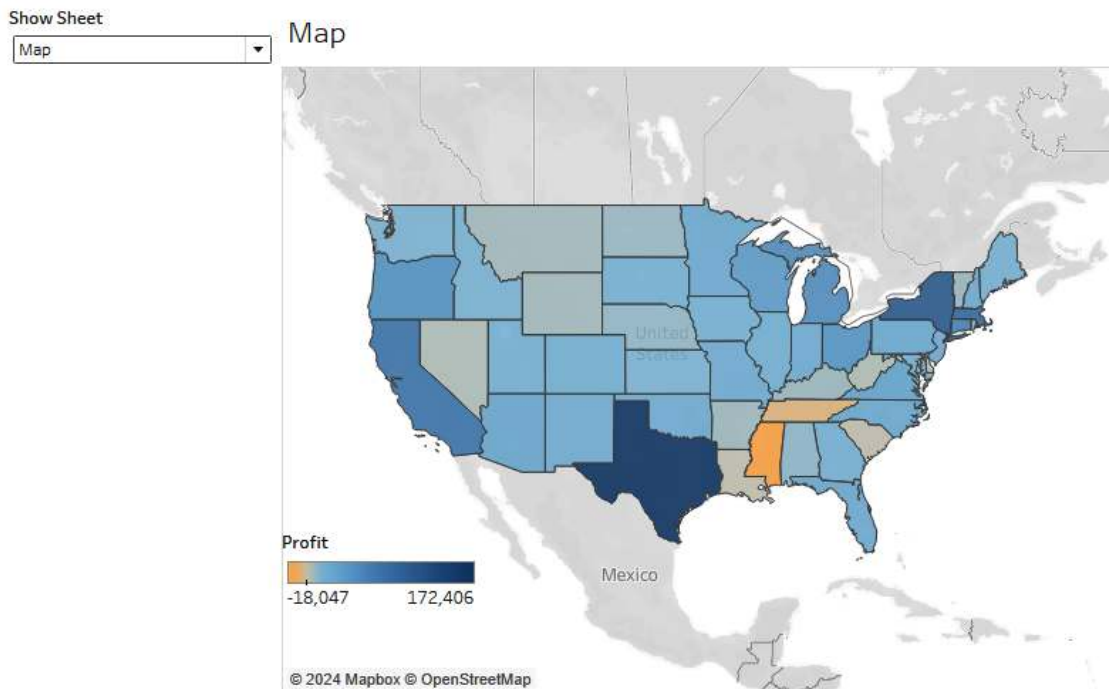


Figure 11.13: When the user selects Map, the Map and legend are visible, but the bar chart is not.

Be aware that views that are not currently visible can still add to the load time of a dashboard. **Sheet swapping**, sometimes also called **sheet selection**, is an older technique in which views are dynamically shown and hidden on a dashboard based on filtering. It avoids some of the performance issues, though is not as flexible as dynamic zone visibility. You can find examples of sheet swapping here: <https://vizpainter.com/sheet-selection-on-steroids/>.

Summary

Creating truly dynamic dashboards gives you incredible flexibility in your dashboard design. Show/Hide buttons give the end user the option to show or hide content on the dashboard. Dynamic Zone Visibility allows you to apply logic to control what objects are shown or hidden.

The techniques covered in this chapter will enable you to accomplish all sorts of amazing interactivity, from hiding and showing controls, instructions, and explanations of business rules and analysis to allowing users to determine visualization types and swap views in and out.

Next, we'll turn our attention to exploring a certain kind of data using some advanced techniques: geospatial data!

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



12

Exploring Mapping and Advanced Geospatial Features

Up until now, we've seen examples of maps and geospatial visualizations that leverage some of the basic functionality of Tableau. In this chapter, we'll embark on a journey to uncover the vast range of mapping and geospatial features available. From using the built-in geospatial database and supplementing it with additional data and spatial files to using advanced geospatial functions, we'll explore what's possible with Tableau.

As we've done previously, this chapter will approach the concepts through some practical examples. These examples will span various industries, including real estate, transportation, and healthcare. As always, these examples are broadly applicable, and you'll discover many ways to leverage your data and uncover answers in the spatial patterns you'll find.

In this chapter, we'll cover the following topics:

- Overview of Tableau maps
- Rendering maps with Tableau
- Using geospatial data
- Leveraging spatial functions
- Creating custom territories
- Map layers
- Tableau mapping - tips and tricks
- Plotting data on background images

Overview of Tableau maps

Tableau contains an internal geographic database that allows it to recognize common geographic elements and render a mark at a specific latitude and longitude on a map. In many cases, such as with a country or state, Tableau also contains internal shapefiles that allow it to render the mark as a complex vector shape in the proper location. Tableau also leverages your specific geospatial data, such

as latitude and longitude, shapefiles, and spatial objects. We'll consider some of those possibilities throughout this chapter. For now, we'll walk through some of the basics of how Tableau renders maps and some of the customizations and options available.

Rendering maps with Tableau

Consider the following screenshot (the Basic Map example in the Chapter 12 workbook), with certain elements numbered for reference:

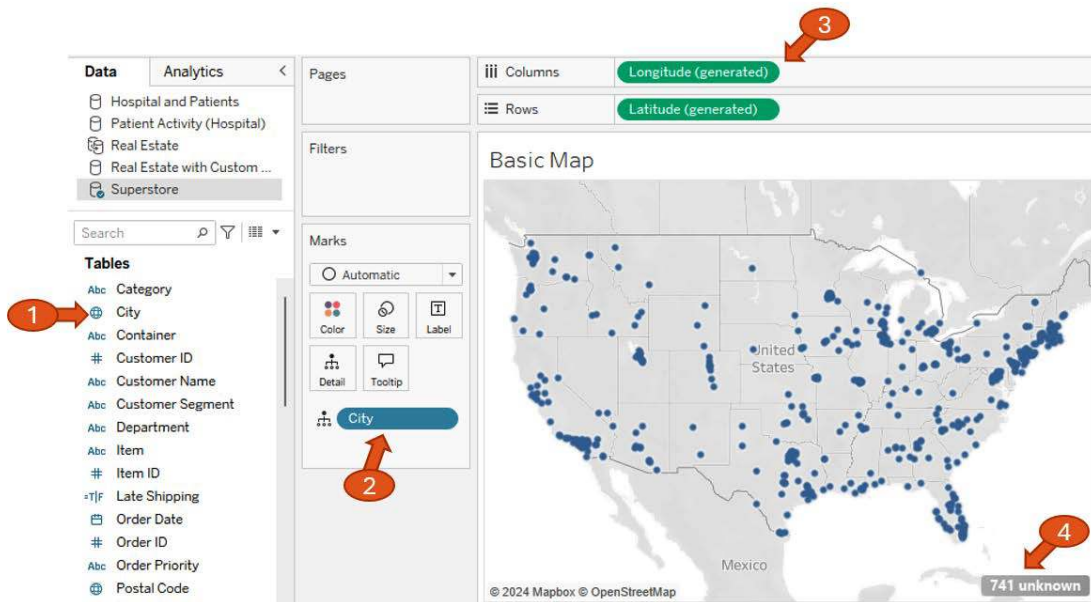


Figure 12.1: A basic geospatial rendering in Tableau

The numbers indicate some of the important aspects of Tableau's ability to render maps:

1. A geographic field in the data is indicated by a globe icon. Fields that Tableau recognizes will have this icon by default. You may assign a geographic role to any field by using the menu and selecting **Geographic Role**.
2. The geographic field in the view (in this case, on **Detail**) is required to render the map.
3. If Tableau is able to match the geographic field with its internal database, the **Latitude (generated)** and **Longitude (generated)** fields placed on **Rows** and **Columns** along with the geographic field(s) on the **Marks** card will render a map.
4. Values that don't find a match in Tableau's geographic database (for example, invalid spellings or abbreviations or locations not contained in the internal database) will result in an indicator that alerts you to the fact that there are unknown values.

You may right-click the unknown indicator to hide it, or click it for the following options:

- Edit locations (manually match location values to known values or latitude/longitude)
- Filter out unknown locations

- Plot at the default location. The default is a latitude and longitude of 0, a location that is sometimes humorously referred to as Null Island, located just off the west coast of Africa.

Tableau renders marks on the map similar to the way it would on a scatterplot (in fact, you might think of a map as a kind of scatterplot that employs some complex geometric transformations to project latitude and longitude). This means you can render circles, dots, and custom shapes on the map.

Under the marks, the map itself is a vector image retrieved from an online map service. We'll consider the details and how to customize the map layers and options next.

Customizing map layers

The map itself—the land and water, terrain, topography, streets, country and state borders, and more—are all part of a vector image that is retrieved from an online map service (an offline option is available from the menu; select **Map** | **Background Maps** | **Offline**).



If you will be using Tableau in an environment where the internet is not available (or publishing to a Tableau server that lacks an internet connection), select the **Offline** option. However, be aware that the offline version does not contain the detail or zoom levels available in the online options.

Marks are then rendered on top of that image. You already know how to use data, calculations, and parameters to adjust how marks are rendered, but Tableau gives you a lot of control over how maps are rendered.

Use the menu to explore various options by selecting **Map** | **Background Maps**. Here, for example, is a **Dark Map**:

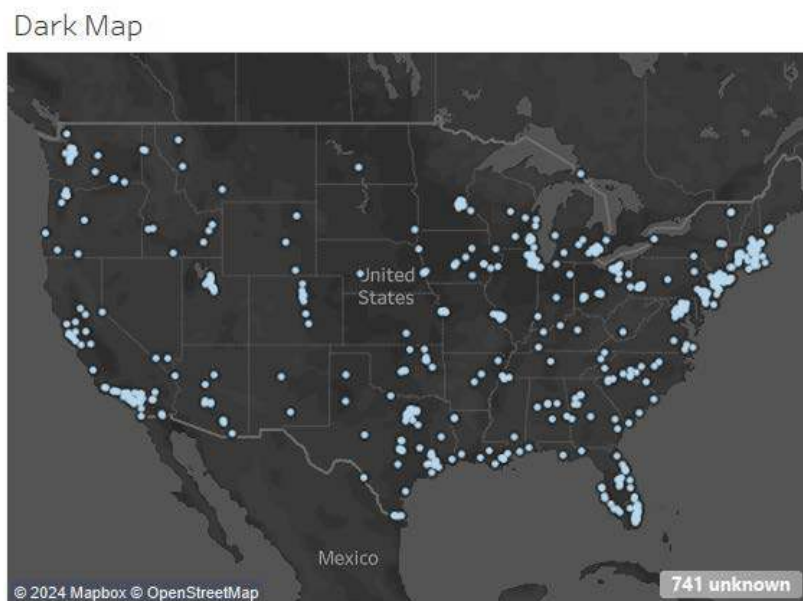


Figure 12.2: Dark Map is one of many options for map backgrounds

This map contains the exact same marks as the previous screenshot. It simply uses a different background. Other options include **Light**, **Streets**, **Satellite**, and more.

Additional layer options can be found by selecting **Map | Background Layers** from the menu. This opens a **Background Layers** pane that looks like this:

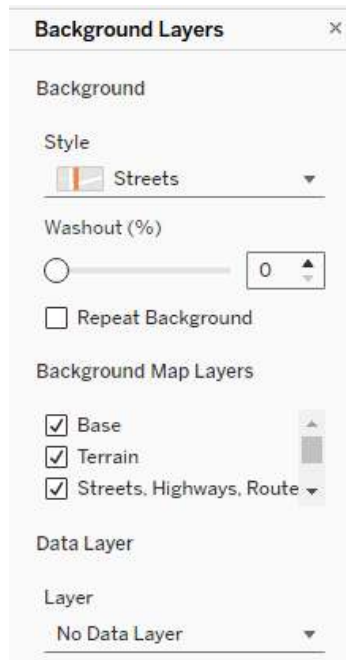


Figure 12.3: The Background Layers pane

The **Background Layers** pane gives you options for selecting a background, setting washout (which allows you to fade the background map, therefore making the marks stand out more), selecting features to display, and setting **Data Layer**. Various options may be disabled depending on the zoom level (for example, **Building Footprints** is not enabled unless you zoom in close enough on the map). **Data Layer** allows you to apply a filled map to the background based on various demographics. These demographics only appear as part of the image and are not interactive and the data is not exposed to either user interaction or calculation.

You may also use the menu options **Map | Background Maps | Manage Maps** to change which map service is used, allowing you to specify your own WMS server, a third party, or to use **Mapbox** maps. This allows you to customize the background layers of your map visualizations in any way you'd like.



The details of these capabilities are outside the scope of this book; however, you'll find excellent documentation from Tableau at https://help.tableau.com/current/pro/desktop/en-us/maps_mapsources_wms.htm.

Customizing map options

Additionally, you can customize the map options available to end users. Notice the controls that appear when you hover over the map:

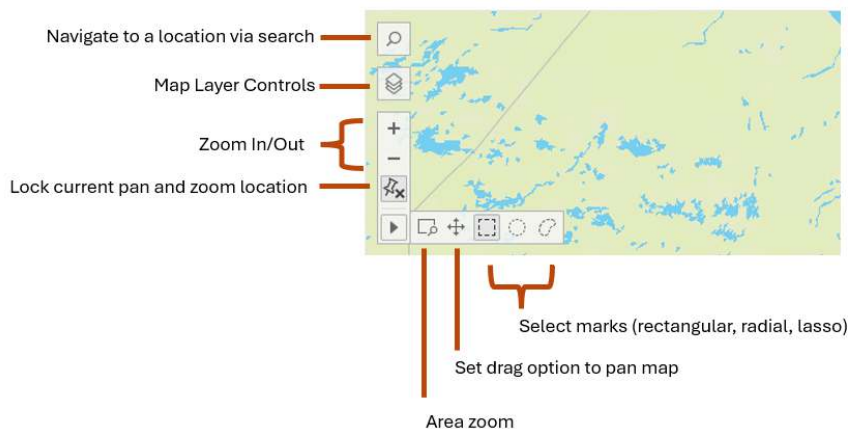


Figure 12.4: Available controls when customizing a map

These controls allow you to search the map, adjust layers (if applicable), zoom in and out, pin the map to the current location, and use various types of selections.



You can also use keyboard and mouse combinations to navigate the map. Use *Ctrl* + mouse wheel or *Shift* + *Ctrl* + mouse click to zoom. Click and hold or *Shift* + click to pan.

Additional options will appear when you select **Map | Map Options** from the top menu:

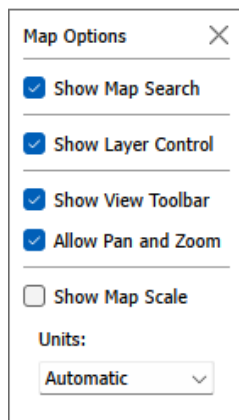


Figure 12.5: Map Options

These options give you the ability to set what map actions are allowed for the end user and whether to show a scale. Additionally, under **Units**, you can set the units displayed for the scale and radial selections. The options are **Automatic** (based on system configuration), **Metric** (meters and kilometers), and **U.S.** (feet and miles).

There are quite a few other geospatial capabilities packed into Tableau and we'll uncover some of them as we next explore how to leverage geospatial data.

Using geospatial data

We've seen that, with any data source, Tableau supplies **Latitude (generated)** and **Longitude (generated)** fields based on any fields it matches with its internal geographic database. Fields such as country, state, zip code, MSA, and congressional district are contained in Tableau's internal geography. As Tableau continues to add geographic capabilities, you'll want to consult the documentation to determine specifics on what the internal database contains.

However, you can also leverage specific geospatial data in your visualizations. We'll consider ways to use data that enable geospatial visualizations, including the following:

- Including Latitude and Longitude as values in your data
- Importing a .csv file containing definitions of Latitude and Longitude in Tableau's database
- Leveraging Tableau's ability to connect to various spatial files or databases that natively support spatial objects

We'll explore each of these options in the following section and then look at how to extend the data even further with geospatial functions in the next section.

Including latitude and longitude in your data

Including latitude and longitude in your data gives you a lot of flexibility in your visualizations (and calculations). For example, while Tableau has built-in geocoding for countries, states, and zip codes, it does not provide geocoding at an address level. Supplying latitude or longitude in your data gives you the ability to precisely position marks on the map.

You'll find the following example in the Chapter 12 workbook using the Real Estate data source:

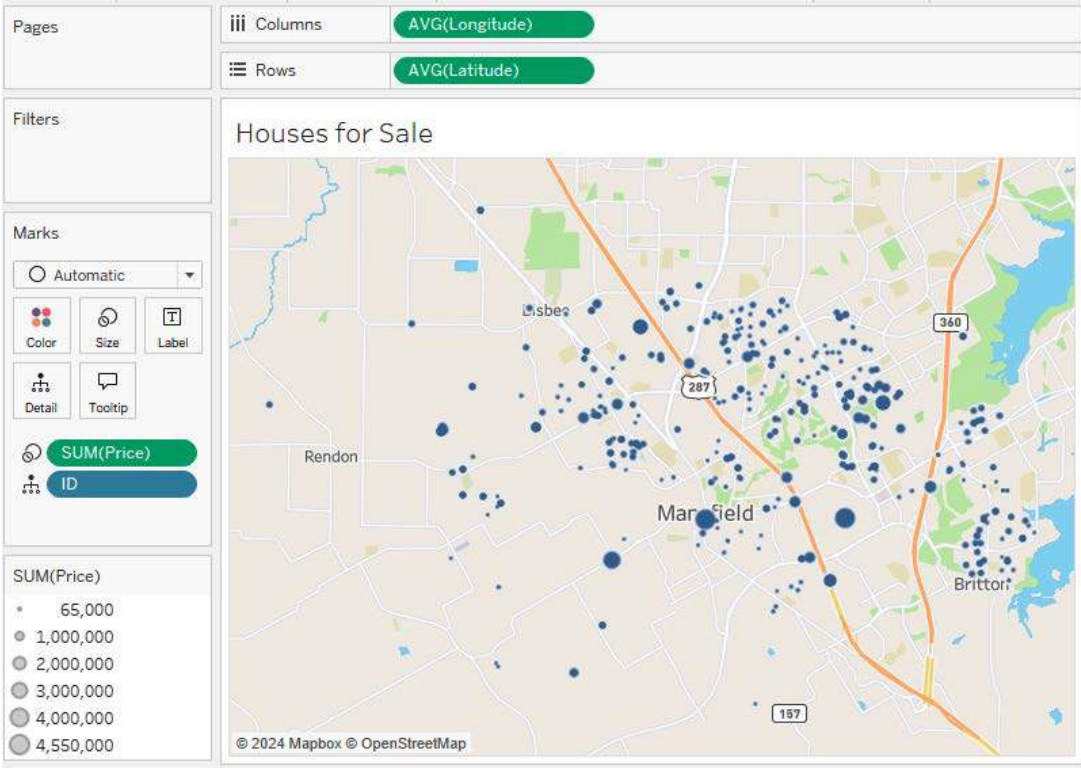


Figure 12.6: A map of houses for sale, sized by price

Here, each house could be mapped with a precise location and sized according to price. In order to help the viewer visually, the **Streets** background has been applied.



There are many free and commercial utilities that allow you to **geocode** addresses. That is, given an address, these tools will add latitude and longitude.

If you are not able to add the fields directly to your data source, you might consider using cross-database joins or data blending. Another alternative is to import latitude and longitude definitions directly into Tableau. We'll consider this option next.

Importing definitions into Tableau's geographic database

In order to import your own custom geographic definitions, from the menu, select **Map | Geocoding | Import Custom Geocoding....** The import dialog contains a link to documentation describing the option in further detail:

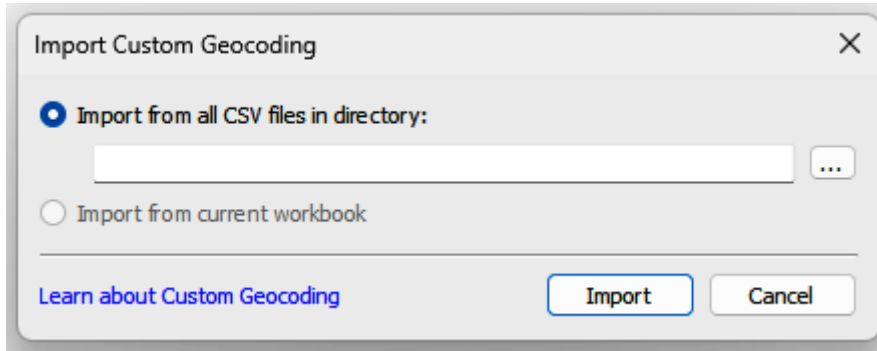


Figure 12.7: The Import Custom Geocoding dialog box

By importing a set of definitions, you can either:

- Add new geographic types
- Extend Tableau's built-in geographic types

Latitude and longitude define a single point. At times, you'll need to render shapes and lines with more geospatial complexity. For that, you'll want to consider some of the geospatial functions and spatial object support, which we'll look at next.

Leveraging spatial objects

Spatial objects define geographic areas that can be as simple as a point and as complex as multi-sided polygons. This allows you to render everything from custom trade areas to rivers, roads, and historic boundaries of counties and countries. Spatial objects can be stored in spatial files and are supported by some relational databases as well.

Tableau supports numerous spatial file formats, such as ESRI, MapInfo, KML, GeoJSON, and TopoJSON. Additionally, you may connect directly to ESRI databases as well as relational databases with geospatial support, such as ESRI, SQL Server, PostgreSQL + PostGIS, and Pivotal Greenplum + PostGIS. If you create an extract, the spatial objects will be included in the extract.

Many applications, such as Alteryx, Google Earth, and ArcGIS, can be used to generate spatial files. Spatial files are also readily available for download from numerous organizations. This gives you a lot of flexibility when it comes to geospatial analysis.

Here, for example, is a map of US railroads:



Figure 12.8: Map of US railroads



To replicate this example, download the shapefile (as a .zip archive) from the United States Census Bureau here: <https://catalog.data.gov/dataset/tiger-line-shapefile-2015-nation-u-s-rails-national-shapefile>.

Once you have downloaded and unzipped the files, connect to the `t1_2015_us_rails.shp` file. In the preview, you'll see records of data with ID fields and railway names. The **Geometry** field is the spatial object that defines the linear shape of the railroad segment:

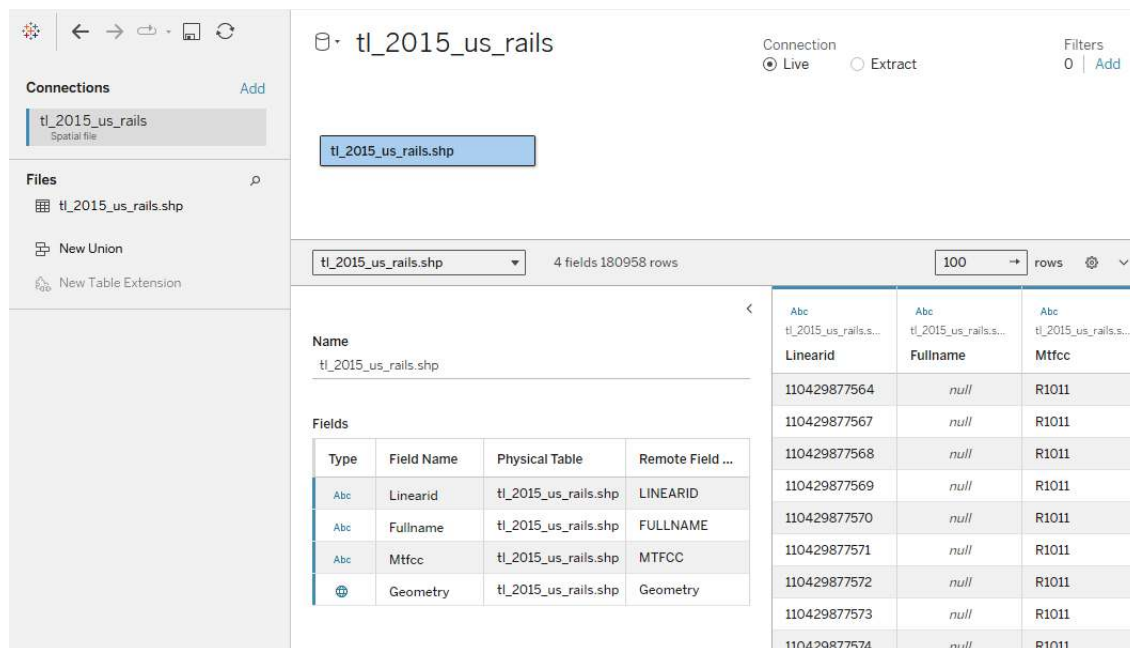


Figure 12.9: Map of US railroads preview

On a blank sheet, simply double-click the **Geometry** field. Tableau will include the geographic collection in the **Detail** section of the **Marks** card and introduce automatically generated latitude and longitude fields to complete the rendering. Experiment with including the **ID** field in the detail and with filtering based on **Fullname**.



Consider using cross-database joins to supplement existing data with custom spatial data. Additionally, Tableau supports spatial joins, which allow you to bring together data that is only related spatially, even if no other relationships exist.

Next, we'll take a look at leveraging some spatial functions and even a spatial join or two to extend your analytics.

Leveraging spatial functions

Tableau continues to add native support for spatial functions. At the time of writing, Tableau supports the following functions:

- **Area()** returns the total area of a polygon spatial object. You may specify the units of measurement as either meters, kilometers, miles, or feet.
- **Makeline()** returns a line spatial object given two points.
- **Makepoint()** returns a point spatial object given two coordinates.
- **Distance()** returns the distance between two points in the desired units of measurement.

- `Buffer()` creates a circle around a point with a radius of the given distance. As with the `Area` function, you may specify the units of measurement.
- `Intersection()` returns a spatial object that represents the intersection of two spatial objects.
- `Intersects()` returns a Boolean value indicating if two given spatial objects intersect.
- `Length()` returns the length of a linestring spatial object (a spatial object that represents a line or path). You may specify the units of measure.
- `Outline()` returns a linestring spatial object that is the outline of a given spatial object.
- `Shapetype()` returns a string that indicates the type of spatial object referenced.
- `Symdifference()` is similar to `Intersection()` but returns the area that is not part of the intersection. That is, it is the area contained in one or the other of two spatial objects, but not both.
- `Validate()` returns a Boolean value indicating whether a given spatial object is valid or not.

We'll explore a few of these functions using the `Hospital` and `Patients` dataset in the Chapter 12 workbook. The dataset reimagines real estate data as a hospital surrounded by patients, indicated in the following view by the difference in **Shape**, **Size**, and **Color**:

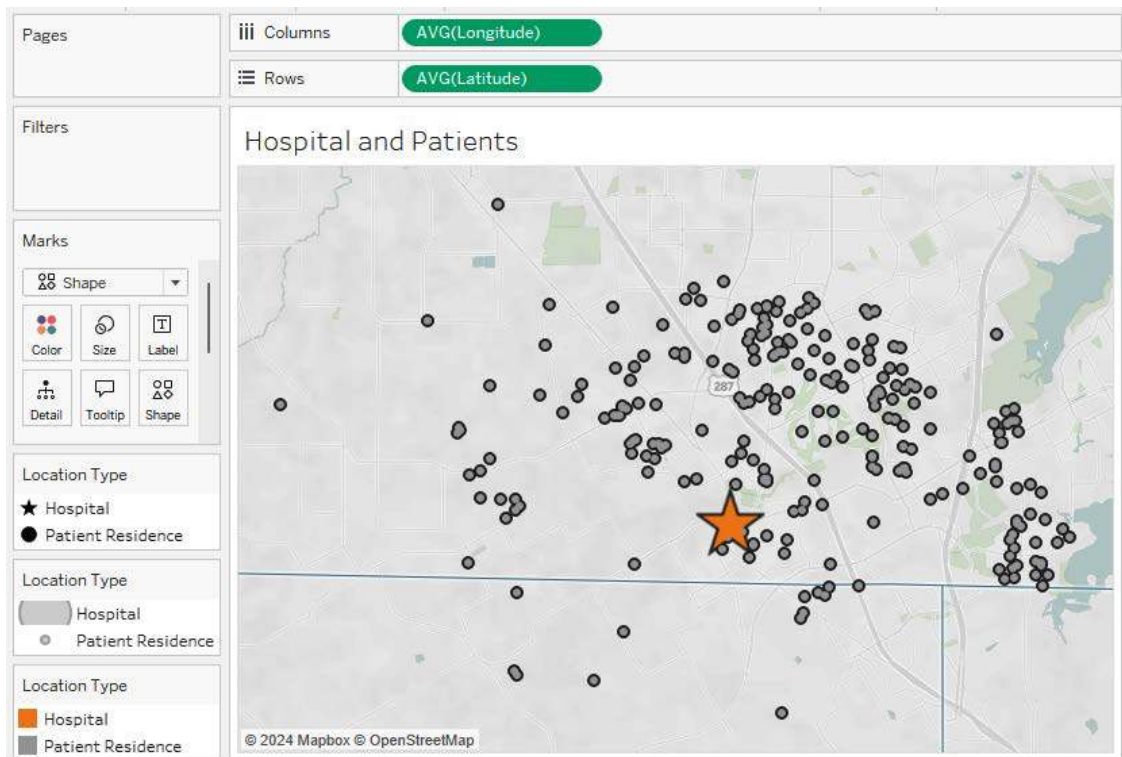


Figure 12.10: A hospital (represented by the star) surrounded by patients

There are numerous analytical questions we might ask. Let's focus on these:

- How far is each patient from the hospital?
- How many patients fall within a given radius?
- Which patients are outside the radius?

To start answering these questions, we'll create some calculated fields that give us the building blocks. In order to use multiple points in the same calculation, the latitude and longitude of the hospital will need to be included with each patient record. One way to achieve this is by using a couple of **FIXED Level of Detail (LOD)** expressions to return the values to each row.

We'll create a calculation called `Hospital Latitude` with the following code:

```
{FIXED : MIN(IF [Location Type] == "Hospital" THEN [Latitude] END)}
```

 **Quick tip:** Enhance your coding experience with the **AI Code Explainer** and **Quick Copy** features. Open this book in the next-gen Packt Reader. Click the **Copy** button

(1) to quickly copy code into your coding environment, or click the **Explain** button

(2) to get the AI assistant to explain a block of code to you.

```
function calculate(a, b) {  
  return {sum: a + b};  
};
```

Copy

Explain

1

2



 **The next-gen Packt Reader** is included for free with the purchase of this book. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



And a corresponding calculation called `Hospital Longitude` with the following code:

```
{FIXED : MIN(IF [Location Type] == "Hospital" THEN [Longitude] END)}
```

In each case, the latitude and longitude for the hospital are determined with the IF/THEN logic and returned as a row-level result by the FIXED LOD expression. This gives us the building blocks for a couple of additional calculations. We'll next consider a couple of examples, contained in the Chapter 12 workbook.

MAKELINE() and MAKEPOINT()

As we consider these two functions, we'll create a calculated field to draw a line between the hospital and each patient. We'll name our calculation **Line** and write this code:

```
MAKELINE(
    MAKEPOINT([Hospital Latitude], [Hospital Longitude]),
    MAKEPOINT([Latitude], [Longitude])
)
```

`MAKELINE()` requires two points, which can be created using the `MAKEPOINT()` function. That function requires a latitude and longitude. The first point is for the hospital and the second is the latitude and longitude for the patient.

As the function returns a spatial object, you'll notice the field has a geography icon:



Figure 12.11: Geography icon added to **Line** field

 **Quick tip:** Need to see a high-resolution version of this image? Open this book in the next-gen Packt Reader or view it in the PDF/ePub copy.

 **The next-gen Packt Reader and a free PDF/ePub copy** of this book are included with your purchase. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



On a new visualization, if you were to double-click the **Line** field, you'd immediately get a geographic visualization because that field defines a geospatial object. You'll notice the **COLLECT(Line)** field on **Detail**, and Tableau's **Longitude (generated)** and **Latitude (generated)** on **Columns** and **Rows**. The geospatial collection is drawn as a single object unless you split it apart by adding dimensions to the view.

In this case, each ID defines a separate line, so adding **ID** to **Detail** on the **Marks** card splits the geospatial object into separate lines:

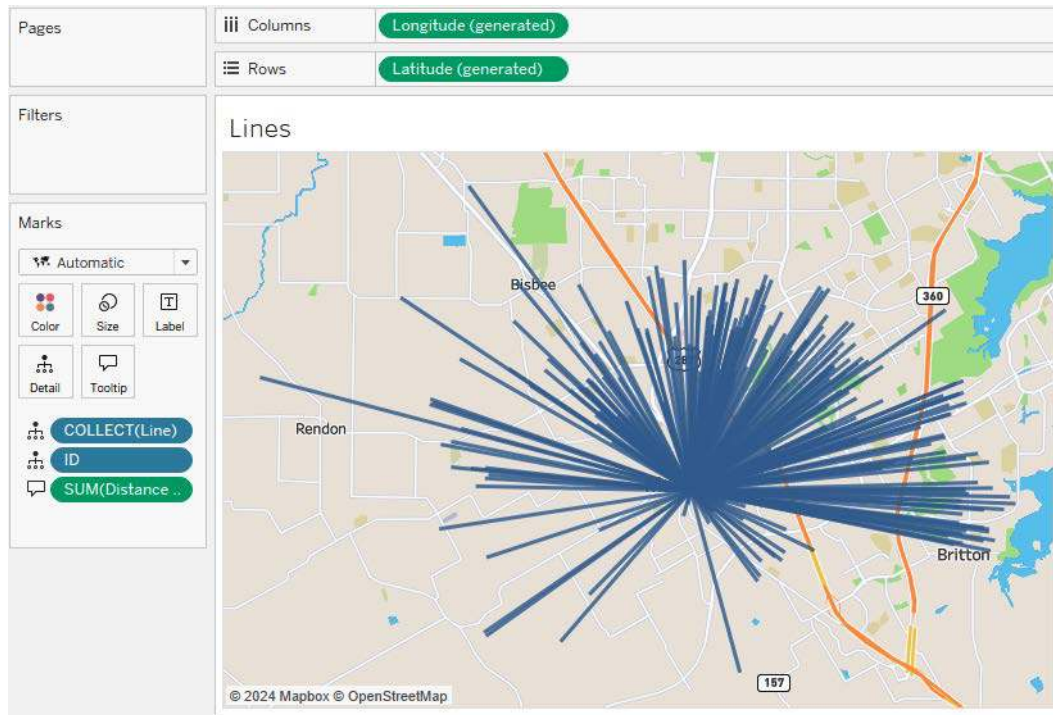


Figure 12.12: Each line originates at the hospital and is drawn to a patient

What if we wanted to know the distance covered by each line? We'll consider that next in an expanded example.

DISTANCE()

Distance can be a very important concept as we analyze our data. Knowing how far apart two geospatial points are can give us a lot of insight. The calculation itself is very similar to `MAKELINE()` and we might create a calculated field named `Distance to the Hospital` with the following code:

```
DISTANCE(
  MAKEPOINT([Hospital Latitude], [Hospital Longitude]),
  MAKEPOINT([Latitude], [Longitude]),
  'mi'
)
```

Similar to the `MAKELINE()` calculation, the `DISTANCE()` function requires a couple of points, but it also requires a unit of measurement.

Here, we've specified miles using the argument `'mi'`, but we could have alternately used `'km'` to specify kilometers.

We can place this calculation on **Tooltip** to see the distance covered by each line:

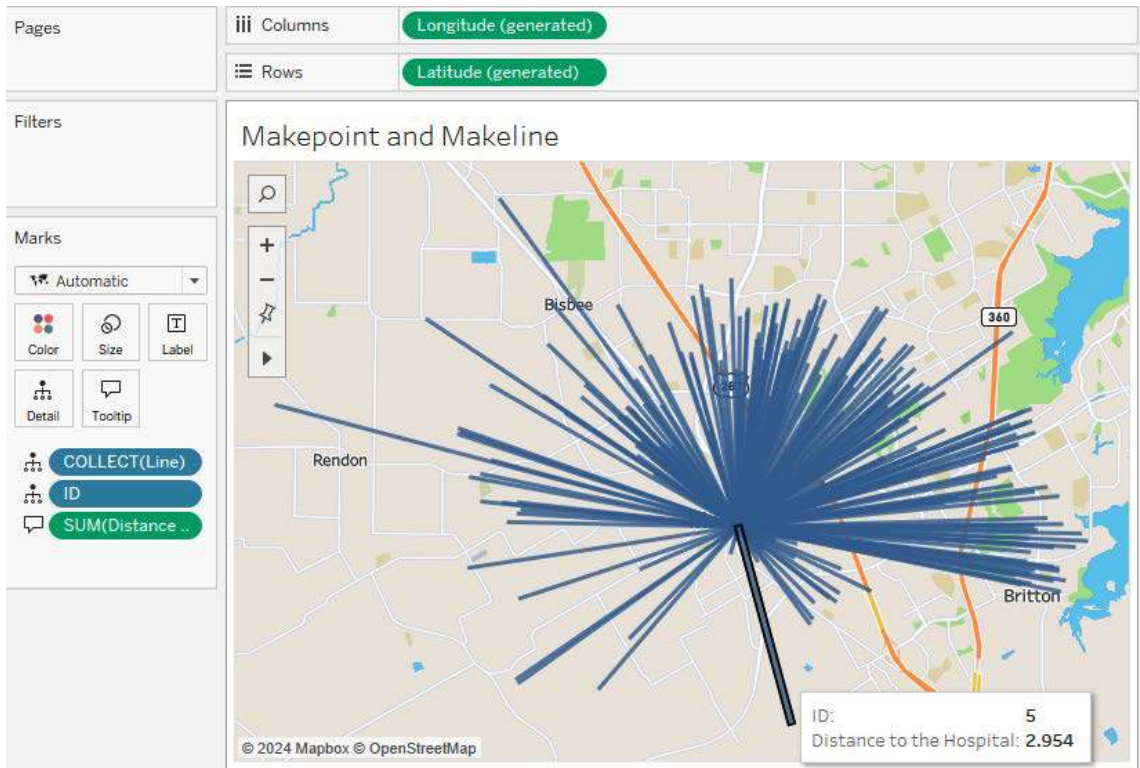


Figure 12.13: The tooltip now displays the distance from the hospital to the patient

This simple example could be greatly extended. Right now, we can tell that patient ID 5 is 2.954 miles away from the hospital when we hover over the line. We could improve the display by rounding down the distance to 2 decimal places or looking up the patient's name. We could greatly increase the analytical usefulness by using distance as a filter (to analyze patients that are over or under a certain threshold of distance) or using distance as a correlating factor in a more complex analysis.

We can accomplish some of this visually with `BUFFER()`, which we'll explore next!

BUFFER()

Buffer is similar to `DISTANCE()`. However, instead of calculating a distance between two points, the `BUFFER()` function allows you to specify a point, a distance, and a unit of measurement to draw a circle with a radius of the specified distance around the point.

For example, you might want to visualize which patients fall within a 3-mile radius of the hospital.

To do that, we'll create a calculated field named `Hospital Radius`, with the following code:

```
IF [Location Type] == "Hospital"
THEN BUFFER(MAKEPOINT([Latitude], [Longitude]), 3, 'mi')
END
```


This code first checks to make sure to perform the calculation only for the hospital record. The `BUFFER()` calculation itself uses the latitude and longitude to make a point and then specifies a 3-mile radius.

In order to visualize the radius along with the individual marks for each patient, we'll create a **dual-axis map**. A dual-axis map copies either the latitude or longitude fields on rows or columns and then uses the separate sections of the **Marks** card to render different geospatial objects. Here, for example, we'll plot the points for patients as circles and the radius using the **Automatic** mark type:

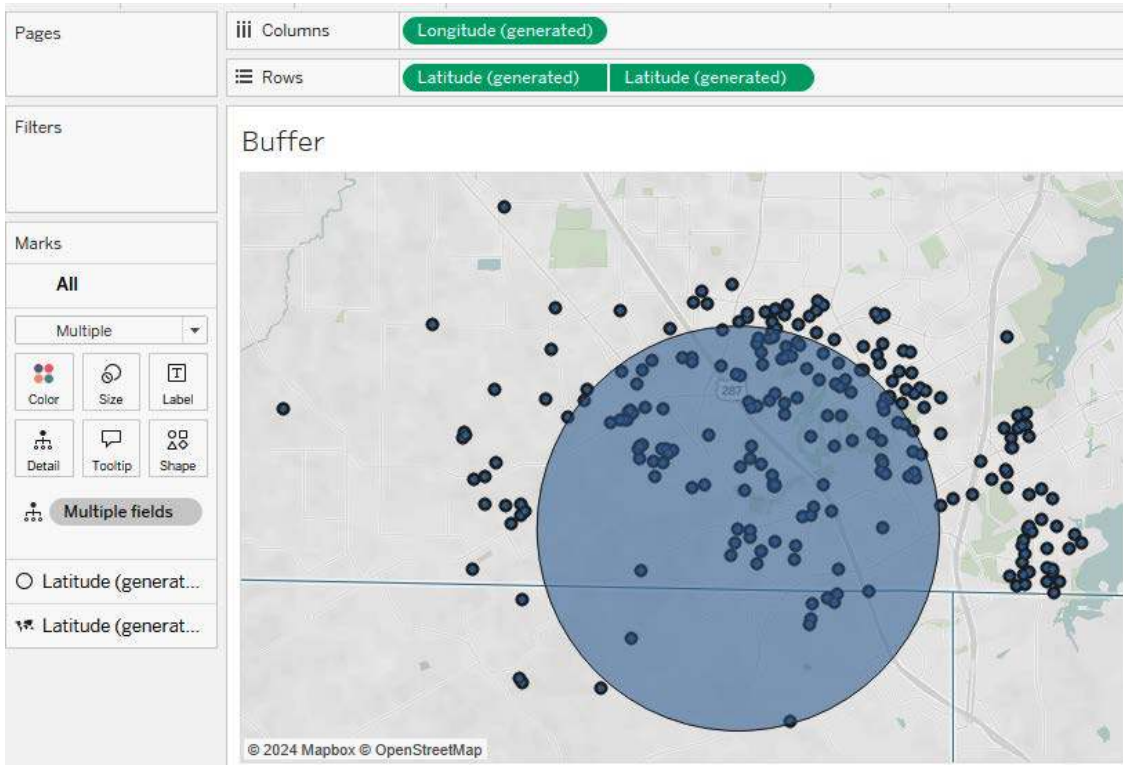


Figure 12.14: Patients who fall within a 3-mile radius of the hospital

Notice that we've used the generated **Latitude** and **Longitude** fields. These serve as placeholders for Tableau to visualize any spatial objects. In the first section of the **Marks** card, we include the **Latitude** and **Longitude** fields from the data. In the second, we included the **Hospital Radius** field. In both cases, the generated fields allow Tableau to use the geographic or spatial objects on the **Marks** card to define the visualization.

We've barely scratched the surface of what's possible with spatial functions. For example, you could parameterize the radius value to allow the end user to change the distance dynamically. You could use `MAKEPOINT()` and `BUFFER()` calculations as join calculations in your data source to bring together spatially related data. With this data, for example, you could **intersect join** the hospital record on `BUFFER()` to the patient records on `MAKEPOINT()` to specifically work with a dataset that includes or excludes patients within a certain radius. This greatly increases your analytic capabilities.

INTERSECTS()

INTERSECTS() takes two geometries and returns an output of true or false based on whether they overlap. Let's say you wanted to not only visually see which patients fell within a 3-mile radius from the hospital, but you wanted to filter to only those patients.

To do that, we'll first create a new version of the hospital radius geometry that copies the hospital radius to every record in the dataset. To accomplish this, we'll create a FIXED Level of Detail, like this:

```

BUFFER(
  MAKEPOINT(
    {FIXED : MAX(IF [Location Type] == 'Hospital' THEN [Latitude] END)},
    {FIXED : MAX(IF [Location Type] == 'Hospital' THEN [Longitude] END)}), 3, 'mi')

```

Next, we'll create another calculation that checks to see if the buffer intersects with the point for each patient. The code will be something like this:

```

INTERSECTS([Hospital Radius (FIXED)], MAKEPOINT([Latitude], [Longitude]))

```

Notice that the function requires two geometries. The first is the FIXED LOD of hospital radius we created. The second is a point defined by the latitude and longitude of the individual patient. A TRUE result indicates that the patient falls within the radius. We can use the field as a filter, on color, to split the view into multiple columns or rows, or nearly any way we'd like! Here we'll simply filter to keep patients that fall within the radius:

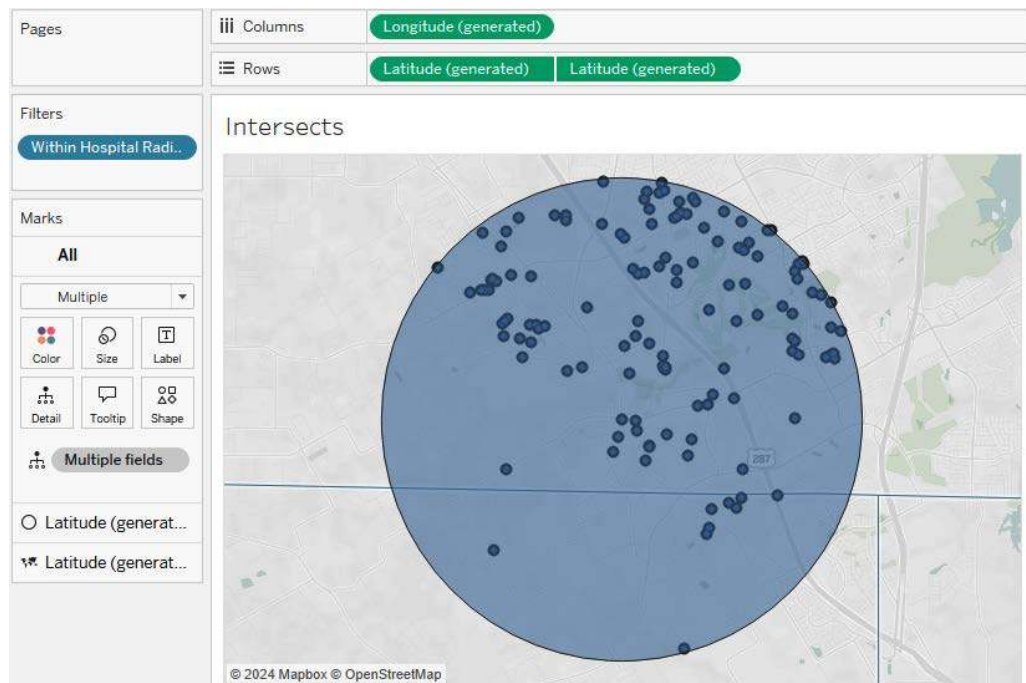


Figure 12.15: Keeping only patients who fall within a 3-mile radius of the hospital

Let's briefly consider a couple of other spatial functions worthy of mention.

Creating custom territories

Custom territories are geographic areas or regions that you create (or that the data defines) as opposed to those that are built in (such as country or area code). Tableau gives you two options for creating custom territories: **ad hoc custom territories** and **field-defined custom territories**. We'll explore these next.

Ad hoc custom territories

You can create custom territories in an ad hoc way by selecting and grouping marks on a map. Simply select one or more marks, hover over one, and then use the **Group** icon. Alternatively, right-click one of the marks to find the option. You can create custom territories by grouping by any dimension if you have latitude and longitude in the data or any geographic dimension if you are using Tableau's generated latitude and longitude.

Here, using the Real Estate with Custom Territories data source, we'll consider an example using zip code:

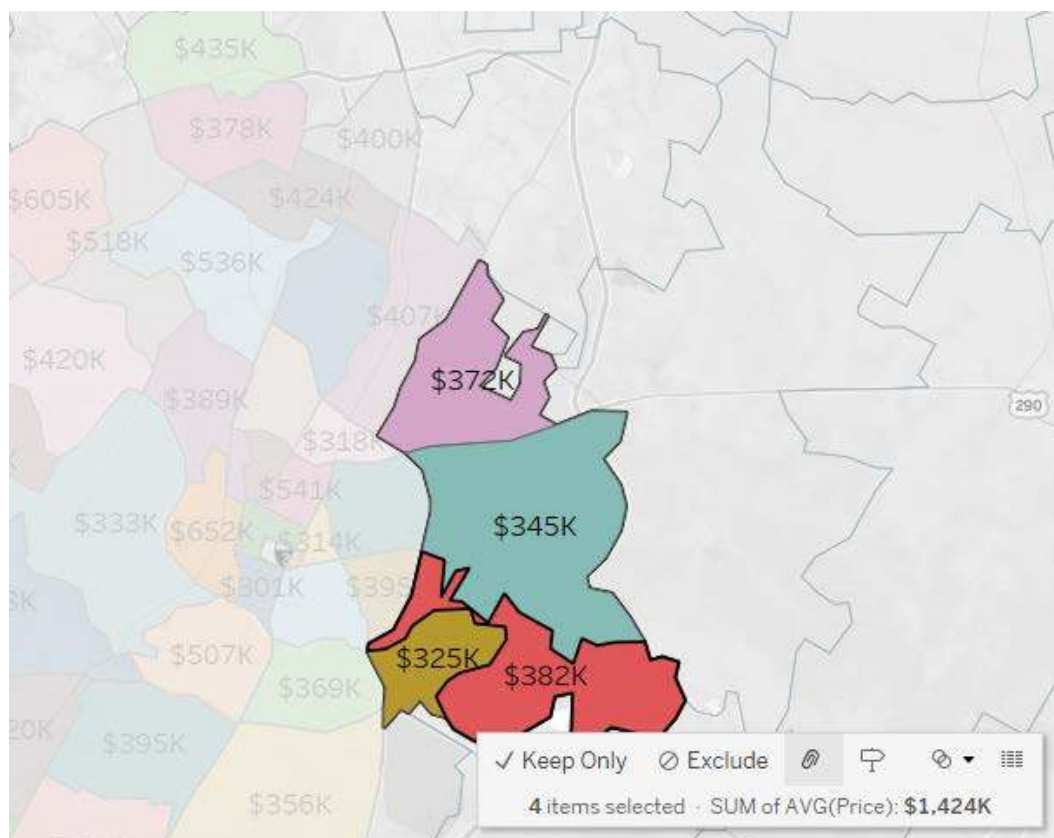


Figure 12.16: After selecting the filled regions to group as a new territory, use the paperclip icon to create the group

You'll notice that after selecting multiple zip codes and clicking the **Group** icon, Tableau creates a new field, **Zip Code (group)**. The new field has a paperclip and globe icon in the data pane, indicating it is a group and a geographic field:



Figure 12.17: A group and geographic field

Tableau automatically includes the group field on **Color**.

You may continue to select and group marks until you have all the custom territories you'd like. With Zip Code still part of the view level of detail, you will have a mark for each zip code (and any measure will be sliced by zip code). However, when you remove Zip Code from the view, leaving only the **Zip Code (group)** field, Tableau will draw the marks based on the new group:

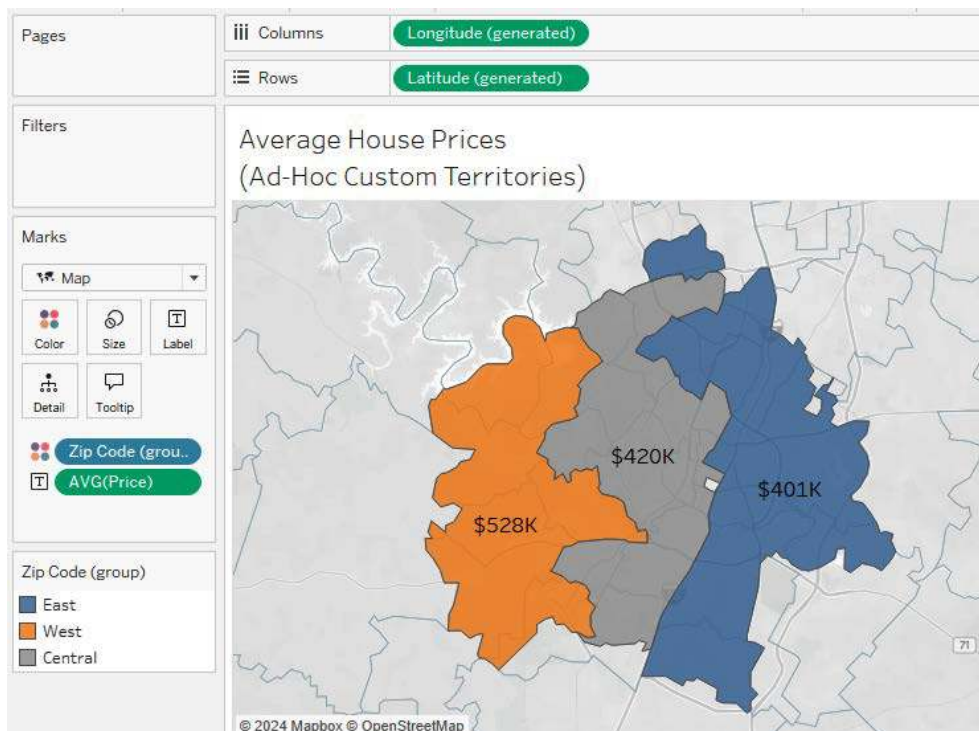


Figure 12.18: Grouping by Custom Territories

Here, the group field has been renamed **Custom Territories** and the group names have been aliased as **East**, **West**, and **Central**. We can see the average price of houses in each of the custom territories.



With a filled map, Tableau will connect all contiguous areas and still include disconnected areas as part of selections and highlighting. With a symbol map, Tableau will draw the mark in the geographic center of all grouped areas. Try switching between mark types to see the difference for yourself.

Sometimes the data itself defines the territories. In that case, we won't need to manually create the territories. Instead, we'll use the technique described next.

Field-defined custom territories

Sometimes your data includes the definition of custom territories. For example, let's say your data had a field named **Region** that already grouped zip codes into various regions. That is, every zip code was contained in only one region. You might not want to take the time to select marks and group them manually.

Instead, you can tell Tableau the relationship already exists in the data. In this example, you'd use the drop-down menu of the **Region** field in the data pane and select **Geographic Role | Create From... | Zip Code**. **Region** is now a geographic field that defines custom territories:

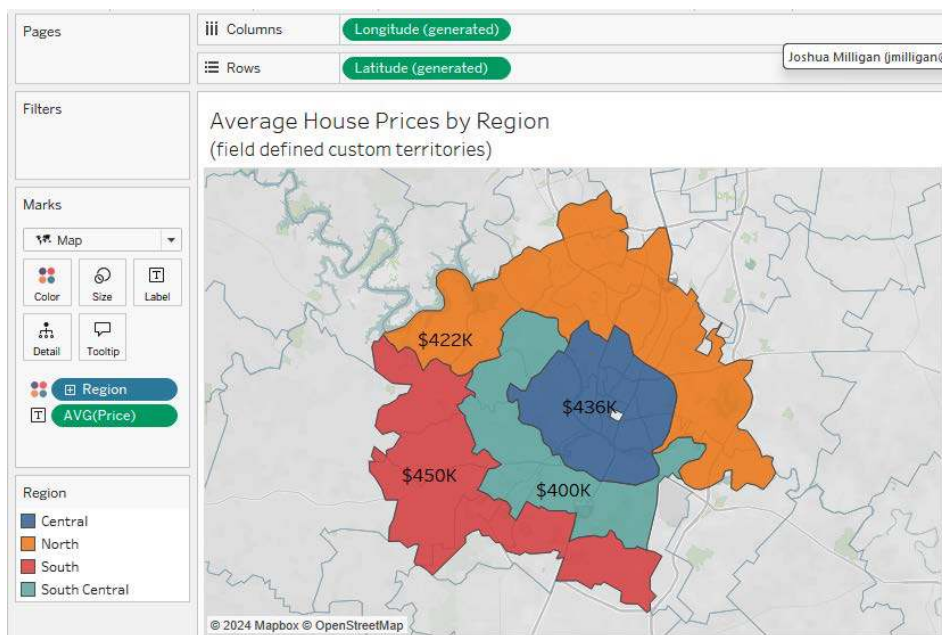


Figure 12.19: The custom regions here are defined by the **Region** field in the data

In this case, the regions have been defined by the **Region** field in the data. If the regions are later redefined and the data is updated, Tableau will display the new regions. Using field-defined custom regions gives us confidence that we won't need to manually update the definitions.



Use ad hoc custom territories to perform quick analysis, but consider field-defined custom territories for long-term solutions because you can then redefine the territories in the data without manually editing any groups in the Tableau data source.

Now that we've considered numerous mapping techniques, from geospatial functions to custom territories, let's consider how we might consolidate a lot of information—from different levels of detail—onto a single map using **Background Layers**.

Map layers

Map layers make it easy to bring together a wide range of geospatial data from your data source(s) into a single map. Consider, for example, the Real Estate with Custom Territories data source we used in the previous example. It contains numerous geographic data, including:

- City
- State
- Zip code
- Region (created as a field-defined custom territory in the previous example)
- Latitude and longitude (of the individual houses for sale)

Let's say we want to create a map that brings much of this together, showing the individual houses as circles sized by price, along with an indication of the zip codes and the custom regions we defined previously with a label indicating the average price for the region.

The main key to leveraging layers is to make certain that Tableau's special **Longitude (generated)** and **Latitude (generated)** are the fields used on **Columns** and **Rows**. Any geographic fields you add to a layer of the map will then be used as the geospatial data to render that specific layer.

To start, we'll create a simple map. We can begin with any of the geospatial fields, but **Zip Code** might be the easiest as a simple double-click on the **Zip Code** creates a geographic visualization. After changing the mark type to **Map**, you'll see a visualization similar to *Figure 12.20*:

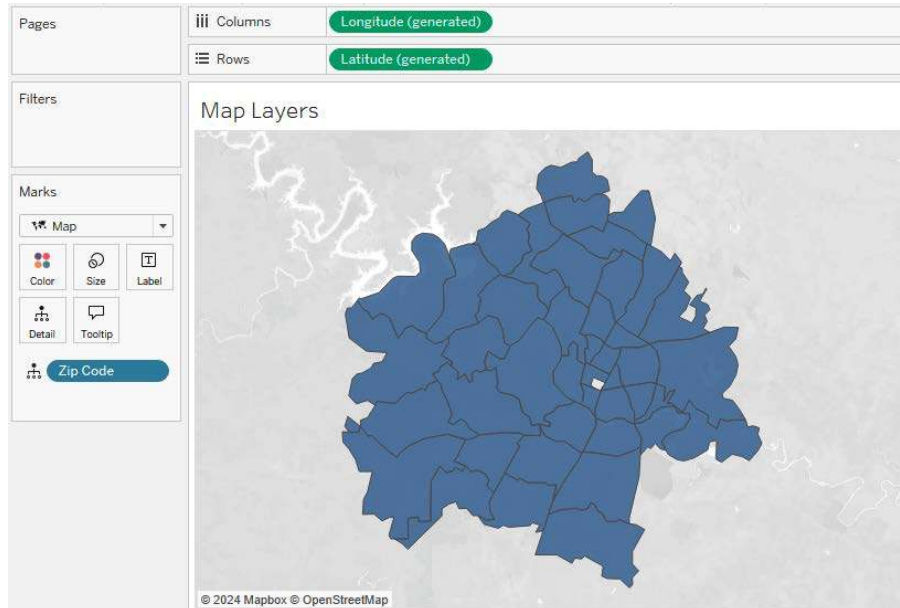


Figure 12.20: A filled map of zip codes

Next, we can take any other geospatial fields and drag and drop them as layers onto the map. For example, dragging **Region** shows an option for adding a **Marks Layer** to the map:

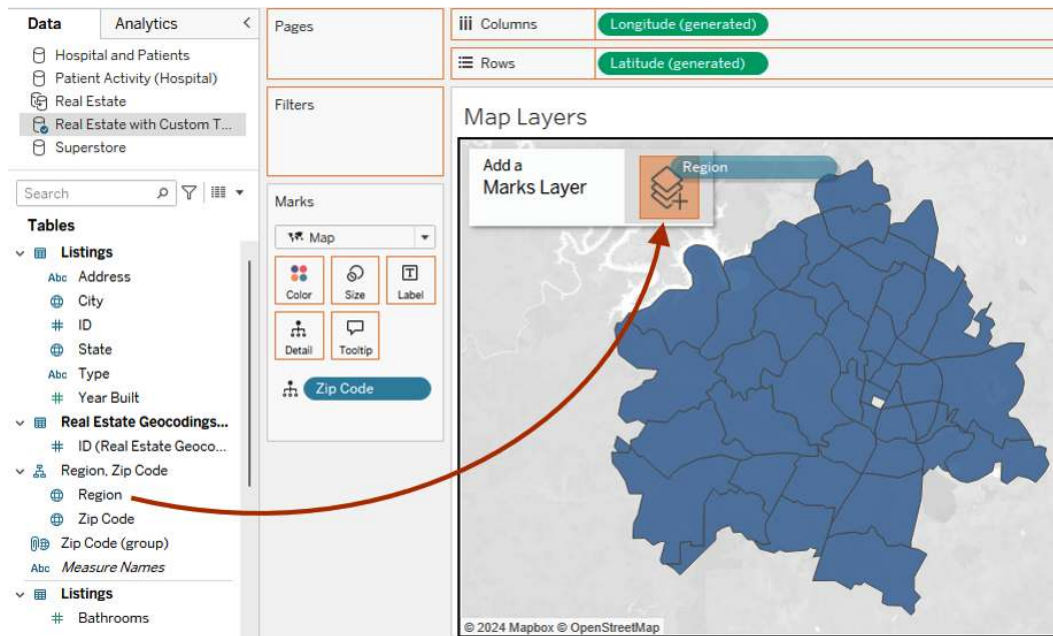


Figure 12.21: Dragging and dropping a geographic field onto the Add a Marks Layer interface adds a layer to the map

Notice that the **Marks** card now contains two layers, **Region** and **Zip Code**, as indicated in Figure 12.22:

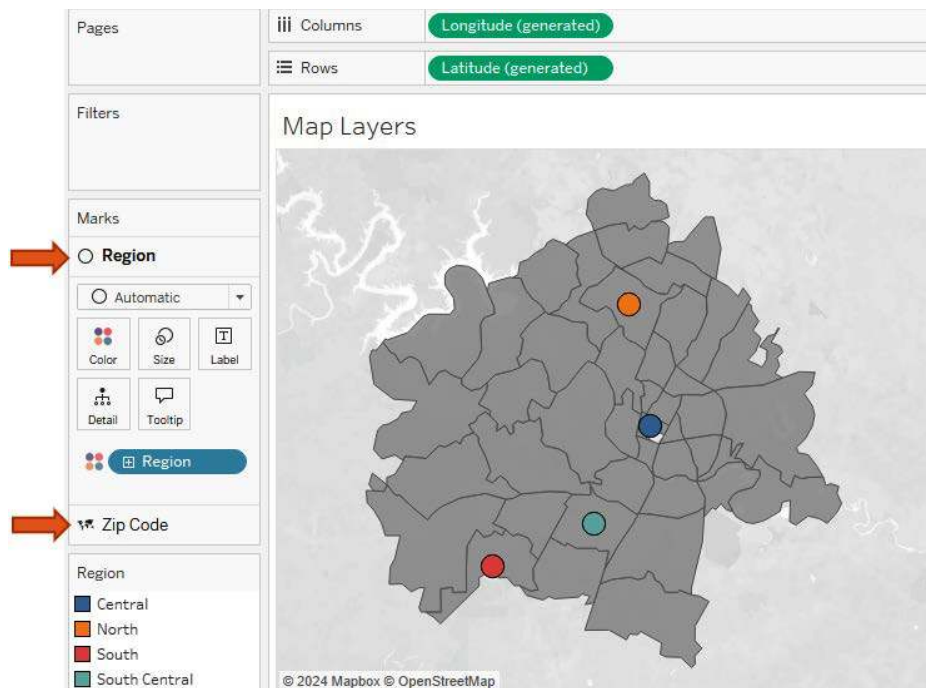


Figure 12.22: Region and Zip Code layers indicated as sections of the Marks card

Each layer can be customized; you can change the mark type, add new fields to add detail or adjust other visual characteristics of the marks for the layer, or even change the visibility of the layer using the eye icon that appears when hovering over the section.

You may also reorder layers by dragging and dropping the sections of the **Marks** card. Layers that are on top will have precedence for user interactivity. That is, marks on top of others may prevent users from hovering over or clicking on those marks and may even visually obscure them, depending on the shape, size, and opacity settings.

To complete this example, we'll follow a few additional steps:

1. Change the **Region** layer's mark type to **Map** and place a copy of **Region** on **Color**.
2. Add one more layer for the individual houses based on the **Latitude** and **Longitude** that we have in our dataset. We'll accomplish this by dragging and dropping either of those fields onto the **Add a Marks Layer** interface.



You won't see any marks displayed until we have both **Latitude** and **Longitude** in the detail of the **Marks** card for that layer, as both fields are required for Tableau to plot a point on the map.

3. By default, **Latitude** and **Longitude** are treated as averages, so we'll need to include a field like **ID** to show a point for each house rather than a single average point for all houses.

The final setup for the **Marks** card layers is shown here:

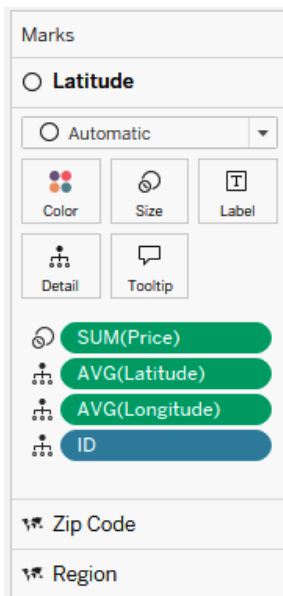


Figure 12.23: Latitude, Longitude, and ID are all required on the layer for Tableau to correctly plot each point



You may optionally rename layers to help you better manage and document your work. In this example, we might consider renaming the **Latitude** layer to **House** because that is ultimately what each latitude and longitude represents.

After a bit of additional fine-tuning and formatting, we might end up with the final visualization in *Figure 12.24*:

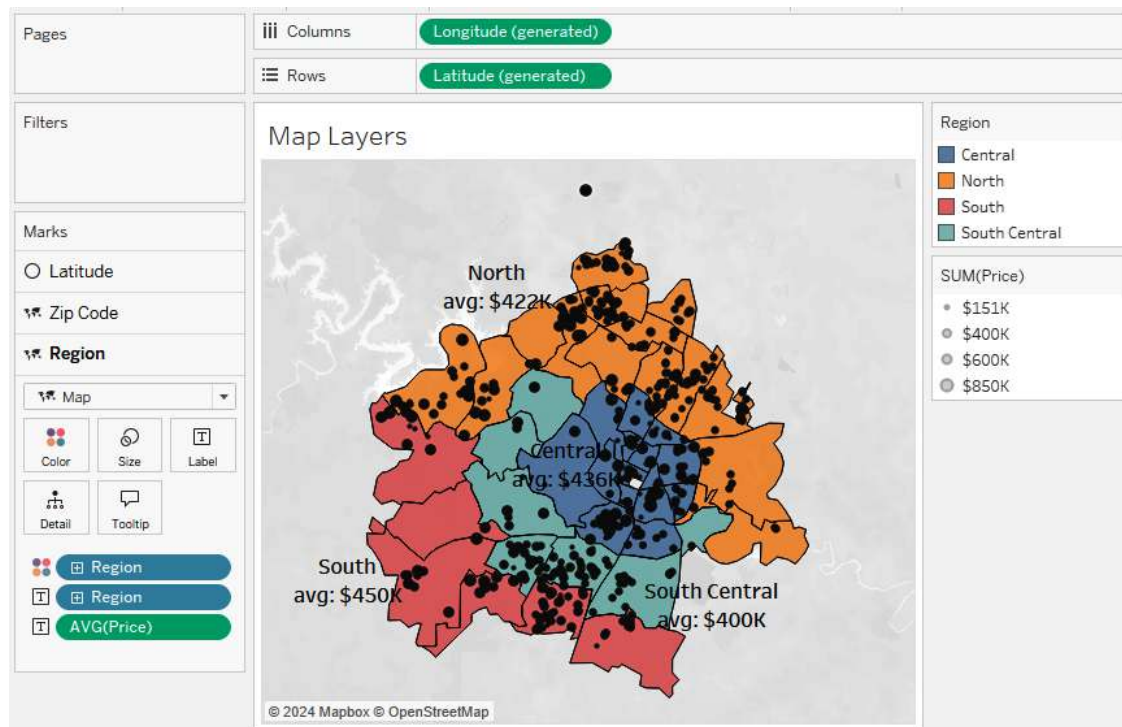


Figure 12.24: A map consisting of three layers



The end user can show and hide layers on a map in a dashboard using the **Layers** hover control (as shown in *Figure 12.5*) as long as you have left the **Show Layer Control** map option active.

With a good understanding of map layers, you are now ready to show numerous geographic details on a single map. Watch out that you don't overwhelm your audience, but the ability to bring together numerous details that are related geospatially is a powerful tool in your analysis and storytelling abilities. With that, we'll turn to a few final tips and tricks for mapping in Tableau.

Tableau mapping – tips and tricks

There are a few other tips to consider when working with geographic visualizations:

- Use the top menu to select **Map | Background Layers** for numerous options for what layers of background to show as part of the map.
- Other options for zooming include using the mouse wheel, double-clicking, *Shift + Alt* + clicking, and *Shift + Alt + Ctrl* + clicking.
- You can click and hold for a few seconds to switch to pan mode.
- You can show or hide the zoom controls and/or map search by right-clicking the map and selecting the appropriate option.
- Zoom controls can be shown on any visualization type that uses an axis.
- The pushpin on the zoom controls alternately returns the map to the best fit of visible data or locks the current zoom and location.
- While **Background Layers** will often be your go-to solution for layering multiple sets of geospatial data on a map, you might also create a **dual-axis map** by duplicating (*Ctrl* + drag/drop) either the **Longitude** on **Columns** or the **Latitude** on **Rows** and then using the field's drop-down menu to select **Dual Axis**. You can use this technique to combine multiple mark types on a single map:

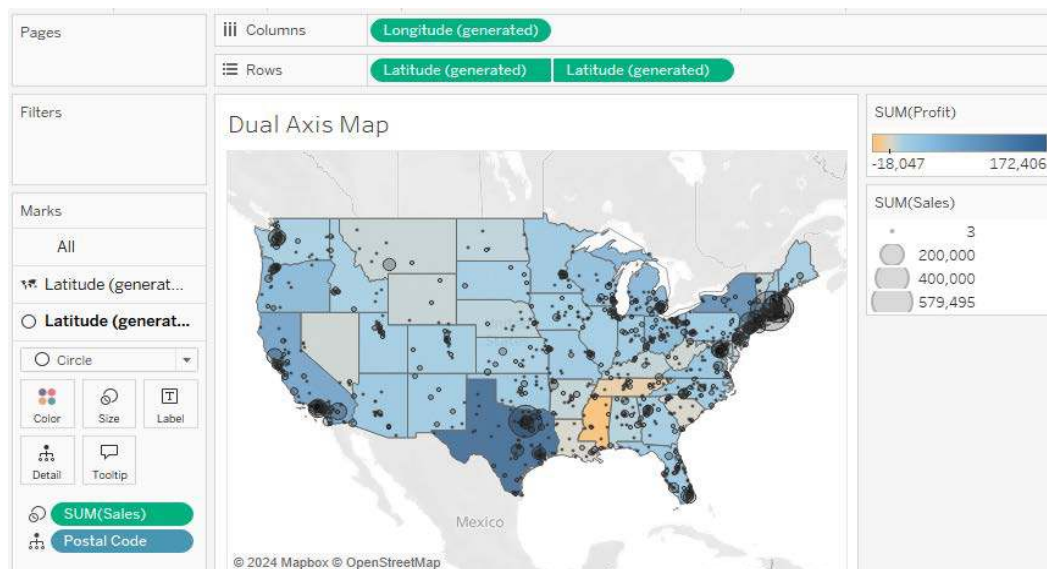


Figure 12.25: Dual Axis Map showing Profit at a state level and Sales at a Postal Code level

- You can use dual axes to display various levels of detail or to use different mark types. In this case, both are accomplished. The map leverages the dual axis to show **Profit** at a state level with a filled map and **Sales** at a **Postal Code** level with a circle.

- When using filled maps, consider setting **Washout** to 100% in the **Background Layers** window for decluttered maps. However, only filled shapes will show, so any missing states (or counties, countries, or others) will not be drawn:

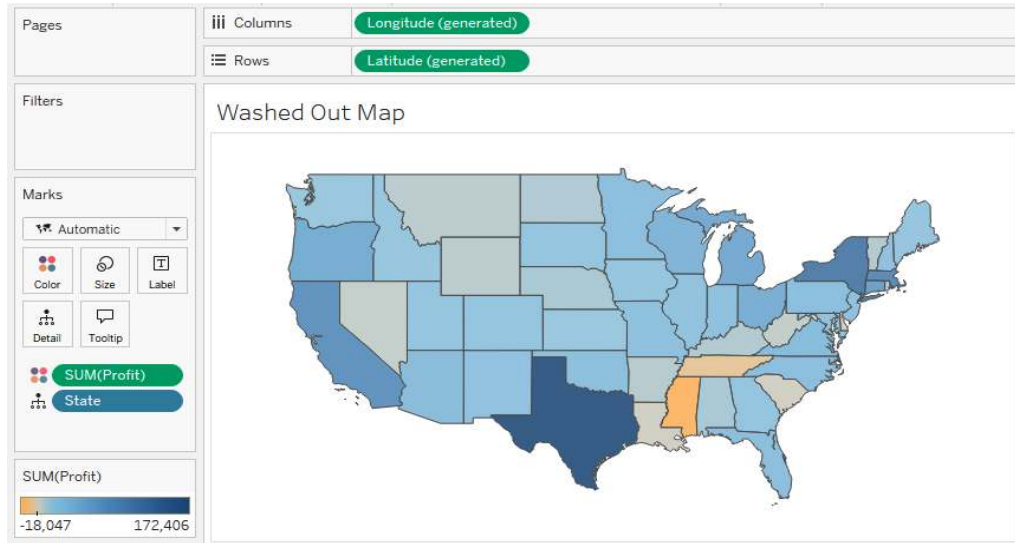


Figure 12.26: Washed Out Map

- You can change the source of the background map image tiles using the menu and selecting **Map | Background Maps**. This allows you to choose between **None**, **Offline** (which is useful when you don't have an internet connection but is limited in the detail that can be shown), or **Tableau** (the default).
- Additionally, from the same menu option, you can specify **Map Services...** to use a WMS server or Mapbox.

Next, we'll conclude this chapter by exploring how plotting your data onto background images can further enhance data visualization and presentation.

Plotting data on background images

Background images allow you to plot data on top of any image. Consider the possibilities! You could plot ticket sales by seat on an image of a stadium, room occupancy on the floor plan of an office building, the number of errors per piece of equipment on a network diagram, or meteor impacts on the surface of the moon.

In this example, we'll plot the number of patients per month in various rooms in a hospital. We'll use two images of floorplans for the ground floor and the second floor of the hospital. The data source is located in the Chapter 12 directory and is named `Hospital.xlsx`. It consists of two tabs: one for patient counts and another for room locations based on the x/y coordinates mapped to the images. We'll shortly consider how that works. You can view the completed example in the Chapter 12 Complete.twbx workbook or start from scratch using Chapter 12 Starter.twbx.

To specify a background image, use the top menu to select **Map | Background Images** and then click the data source to which the image applies—in this example, **Patient Activity (Hospital)**. On the **Background Images** screen, you can add one or more images.

Here, we'll start with **Hospital - Ground Floor.png**, located in the **Chapter 12** directory:

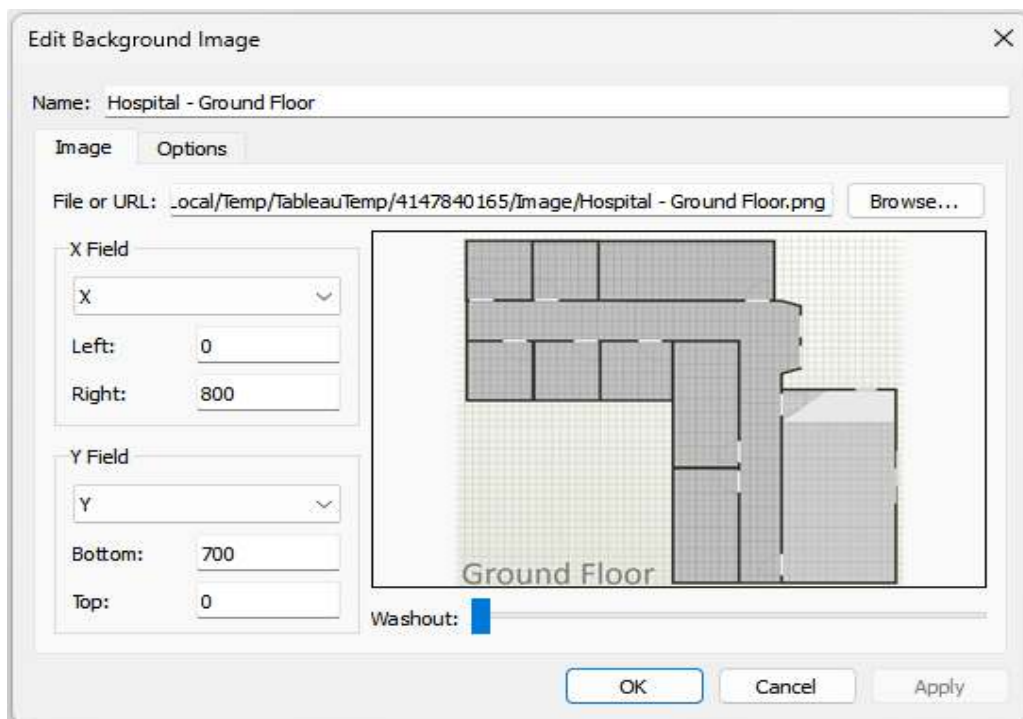


Figure 12.27: Edit Background Image pane

You'll notice that we mapped the fields **X** and **Y** (from the **Locations** tab) and specified **Right** at 800 and **Bottom** at 700. This is based on the size of the image in pixels.



You don't have to use pixels, but most of the time it makes it far easier to map the locations for the data. In this case, we have a tab of an Excel file with the locations already mapped to the *x* and *y* coordinates on the image (in pixels). With cross-database joins, you can create a simple text or Excel file containing mappings for your images and join them to an existing data source. You can map points manually (using a graphics application) or use one of many free online tools that allow you to quickly map coordinates on images.

As this image applies only to the ground floor, we'll only want to show it for data related to the ground floor, so, switching to the **Options** tab, we'll ensure that the condition is set based on the data. We'll also make sure to check **Always Show Entire Image**:

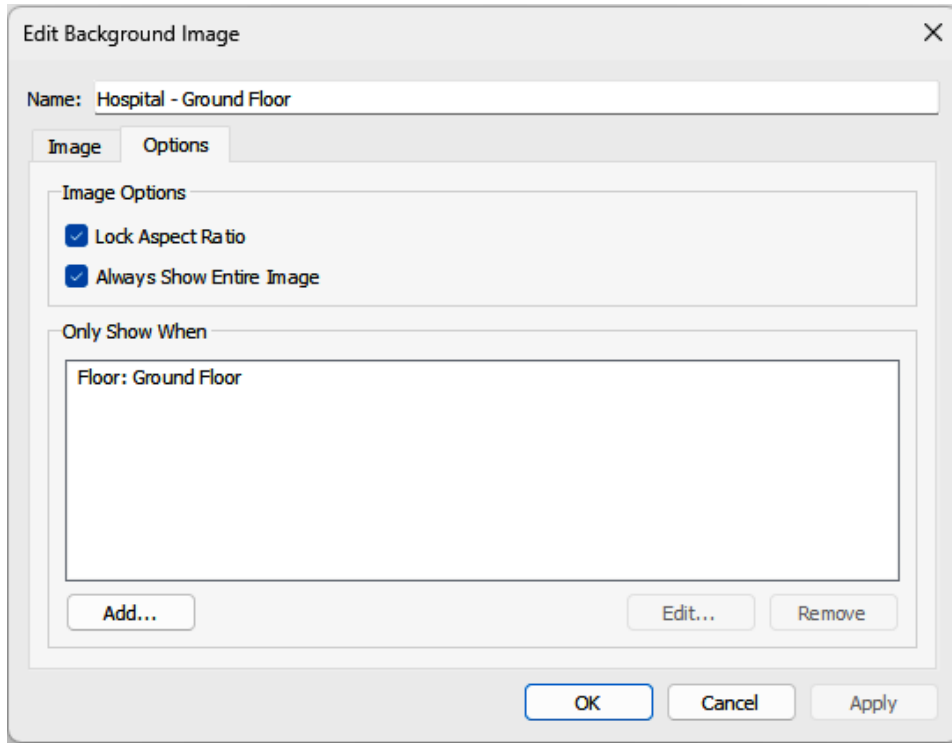


Figure 12.28: Edit Background Image pane

Next, repeating the preceding steps, we'll add the second image (`Hospital - 2nd Floor.png`) to the data source, ensuring it only shows for 2nd Floor.

Once we have the images defined and mapped, we're ready to build a visualization. The basic idea is to build a scatterplot using the X and Y fields for axes. But we have to ensure that X and Y are not summed because if they are added together for multiple records, then we no longer have a correct mapping to pixel locations. There are a couple of options:

- Use X and Y as continuous dimensions.
- Use MIN, MAX, or AVG instead of SUM, and ensure that **Location** is used to define the view level of detail.
- Additionally, images are measured from 0 at the top to Y at the bottom, but scatterplots start with 0 at the bottom and values increase upward. So, initially, you may see your background images appear upside-down. To get around this, we'll edit the y axis (right-click and select **Edit Axis**) and check the option for **Reversed**.

We also need to ensure that the **Floor** field is used in the view. This is necessary to tell Tableau which image should be displayed. At this point, we should be able to get a visualization like this:

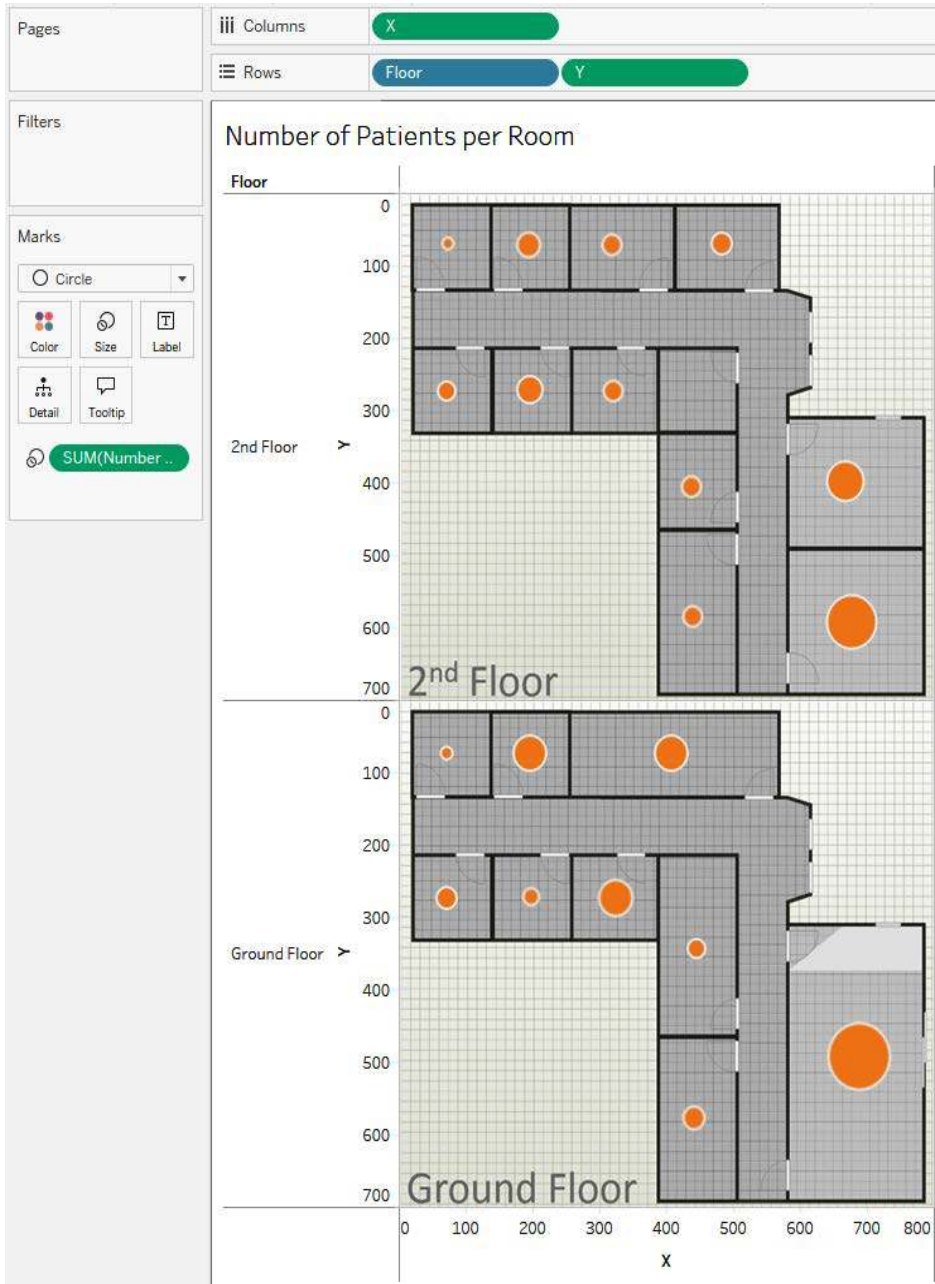


Figure 12.29: Plotting the number of patients per room on a floorplan image

Here, we've plotted circles with the size based on the number of patients in each room. We could clean up and modify the visualization in various ways:

- Hide the x and y axes (right-click the axis and uncheck **Show Header**)
- Hide the header for **Floor**, as the image already includes the label
- Add **Floor** to the **Filter** shelf so that the end user can choose to see one floor at a time

The ability to plot marks on background images opens a world of possibilities for communicating complex topics. Consider how you might show the number of hardware errors on a diagram of a computer network, the number of missed jump shots on a basketball court, or the distance between people in an office building.

All of this, and more, is possible!

Summary

We've covered a lot of ground in this chapter! The basics of visualizing maps are straightforward, but there is a lot of power and possibility behind the scenes. From using your own geospatial data to leveraging geospatial objects and functions and potentially layering a lot of information on a single map, you have a lot of analytical options. Creating custom territories and plotting data on background images expand your possibilities even further.

Next, we'll introduce some capabilities that greatly expand the functionality of Tableau. In fact, you'll even write your own code in Python, R, and JavaScript and see the power of AI to extend your analytics, dashboards, and interactivity to a whole new level!

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search for this book by name.



Note: Keep your purchase invoice ready before you start.

13

Understanding the Tableau Data Model, Joins, and Blends

In this chapter, we'll gain a deeper understanding of how to model and structure data with Tableau. We've seen the **Data Source** screen in previous chapters and briefly explored how to drag and drop tables to form relationships. Now, we'll explore some of Tableau's more complex features to gain a good understanding of how Tableau allows you to relate multiple tables together, either logically or physically.

We'll start with a broad overview of Tableau's new data model and then examine some details of different types of joins and blends. The data model and blending apply primarily to Tableau Desktop (and Server), but pay special attention to the discussion of joins, as a good understanding of join types will aid you greatly when we discuss Tableau preparation in *Chapter 15, Taming Data with Tableau Prep*.

In this chapter, we'll cover the following topics:

- Explanation of the sample data used in this chapter
- Exploring the Tableau data model
- Using joins
- Using blends
- When to use a data model, joins, or blends

We'll start by understanding the sample dataset included in the workbook for this chapter. This is so that you have a good foundation of knowledge before working through the examples.

Explanation of the sample data used in this chapter

For this chapter, we'll use a sample dataset of patient visits to the hospital. The data itself is contained in the Excel file `Hospital Visits.xlsx` in the `Learning Tableau\Chapter 13` directory. In this example, each tab of the Excel file represents tables of data. The principles described here are directly applicable to relational database tables, multiple files, or tables in any data source. The relationship between those tables is illustrated here:

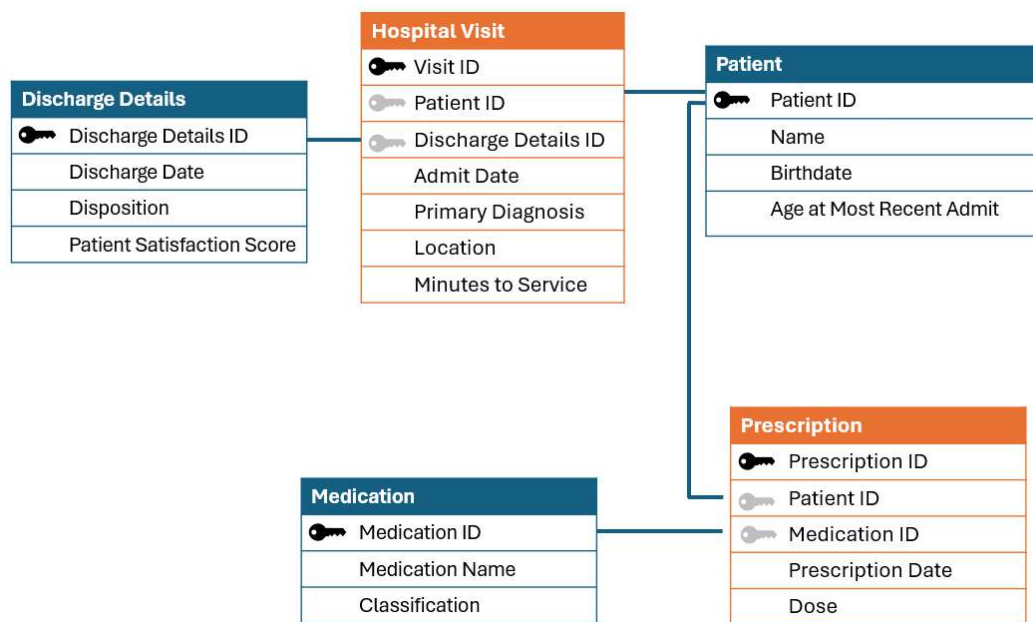


Figure 13.1: The five tabs of the Excel file illustrated as four tables with relationships

Excel does not explicitly define the relationships, but they are shown here as they might exist in a relational database using foreign key lookups.

The orange tables in the diagram represent base tables. In a Tableau data model, the **base tables** are the tables on which the rest of the model is built. They are the main tables that contain links to other tables. In a classic data warehouse, they would be called **fact tables** because they contain the “facts” or measures that will be aggregated (minutes to service and dose as well as a count of the number of records, which indicates the number of visits and the number of prescriptions). The blue tables are **dimension tables** because they contain attributes about each object (a discharge, a patient, a medication).

Here is a brief explanation of the individual tables and their relationships:

- **Hospital Visit:** This table records the admission and diagnosis of a single patient on a single visit to the hospital. It contains attributes, such as **Admit Type** and **Location**, and a measure of **Minutes to Service**. It has keys that link it to the patient table and to a table containing details about the patient’s discharge.
- **Prescription:** This table has a record for each prescription for patients. In addition to a date and a dose, there is a key that links it to a medication table.
- **Patient:** This table contains additional information for a single patient, such as their **Name** and **Birthdate**, and a measure of their **Age at Most Recent Admit**.

- **Discharge Details:** This table gives additional information for the discharge of a patient, such as the **Discharge Date** and the **Disposition** (the condition under which they were discharged and where they went after being discharged). It also contains a measure, **How did patient feel about care?** (1-10), that ranks the patient's feelings about the level of care, with 1 being the lowest and 10 being the highest.
- **Medication:** This table contains a record for each medication and includes attributes such as name and classification.

The tables relate to each other in different ways. Here are some details:

- **Hospital Visit to Patient:** Each visit is made by a single patient, so **Hospital Visit** will always have a **Patient ID** field that points to a single record in the **Patient** table. We will also find additional patients in the **Patient** table who do not have recorded visits. Perhaps they are historical records from a legacy system or the patient's interaction with the hospital occurred in a manner other than a visit.
- **Hospital Visit to Discharge Details:** Each visit may have a single discharge, but some patients may still be in the hospital. In a well-designed data structure, we should be able to count on a record in the **Discharge Details** table that indicates "still in the hospital." In our Excel data, however, there may or may not be a **Discharge Details ID**, meaning there won't always be a matching **Discharge Details** record for every **Hospital Visit**.
- **Prescription to Patient:** Some patients have one or more prescriptions. If we examined the prescription table, we would find that it does not have **Patient IDs** that match every patient in the **Patient** table. This is expected because not all patients have prescriptions. We will also find some Patient IDs repeated in the **Prescription** table. This is also expected because some patients have more than one prescription.

With a solid grasp of the sample data source, let's turn our attention to how we might build a data model in Tableau.

Exploring the Tableau data model

Every data source you create in Tableau will use the data model. Any time you relate two or more sets of data, you'll need to give some thought to the various ways you might relate them in the data model. We'll cover some of the basics of what a data model is and how to create it first.

Creating a data model

We briefly looked at the **Data Source** screen in *Chapter 2, Connecting to Data in Tableau*. Now, we'll take a deeper look at the concepts behind the interface. Feel free to follow along with the following example in the Chapter 13 Starter.twb workbook, or examine the end results in Chapter 13 Complete.twbx.

We'll start by creating a connection to the `Hospital Visits.xlsx` file in the Chapter 13 directory. The **Data Source** screen will look like this upon first connecting to the file:

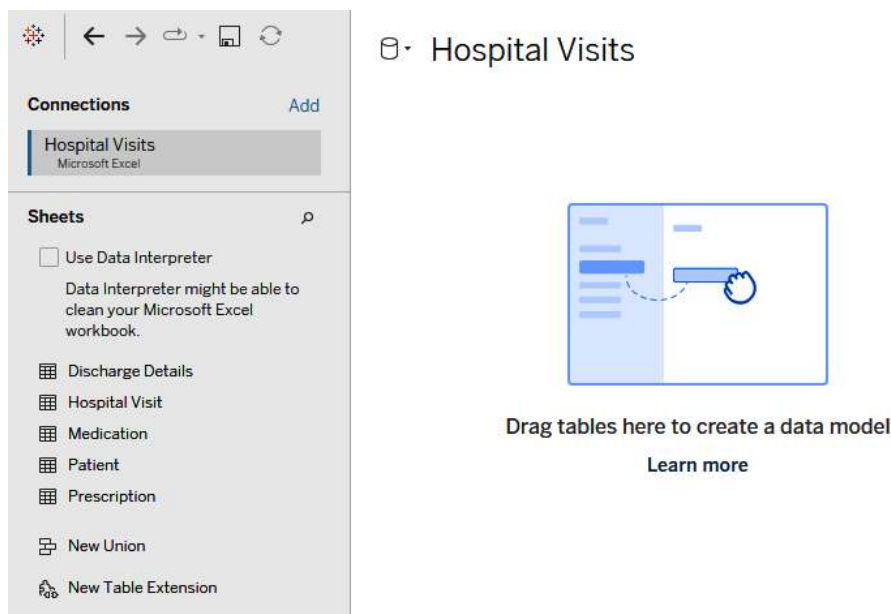


Figure 13.2: The Data Source screen lists the tabs in the Excel workbook and invites you to start a data model

We'll build the data model by dragging and dropping tables onto the canvas. It is important to start with our base tables, which we previously identified as **Hospital Visit** and **Prescription**.



In many cases, you'll have only a single base table. In this example, and often in real-world scenarios, you'll have multiple base tables. Using multiple base tables related through common dimensions is called **multi-fact analysis** in Tableau.

We'll start by dragging and dropping **Hospital Visit** onto the canvas. With each successive table we drag, Tableau will draw suggested relationships to a base table and provide a drop zone to add a **New Base Table** as seen here:

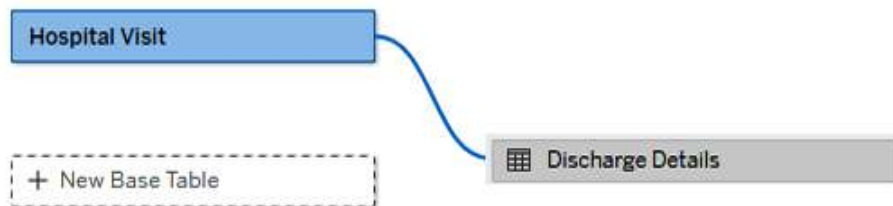


Figure 13.3: Dragging a new table onto the canvas shows the possible relationships that can be created



The connectors between tables are properly called **relationships** but are colloquially known as *noodles* due to their flexible look and feel in the interface.

Tableau will create the relationship based on fields that match in name and type if possible (or based on referential integrity defined in a relational database). You may always adjust the settings as needed. For our tables, we'll accept the default settings because the ID fields that indicate the correct relationship are identically named and implicitly typed the same.

After adding **Hospital Visit** as the base, add **Discharge Details** and **Patient** as related tables, accepting all defaults. Next, we'll add **Prescription** as a new base table by dropping it onto the base table as shown here:

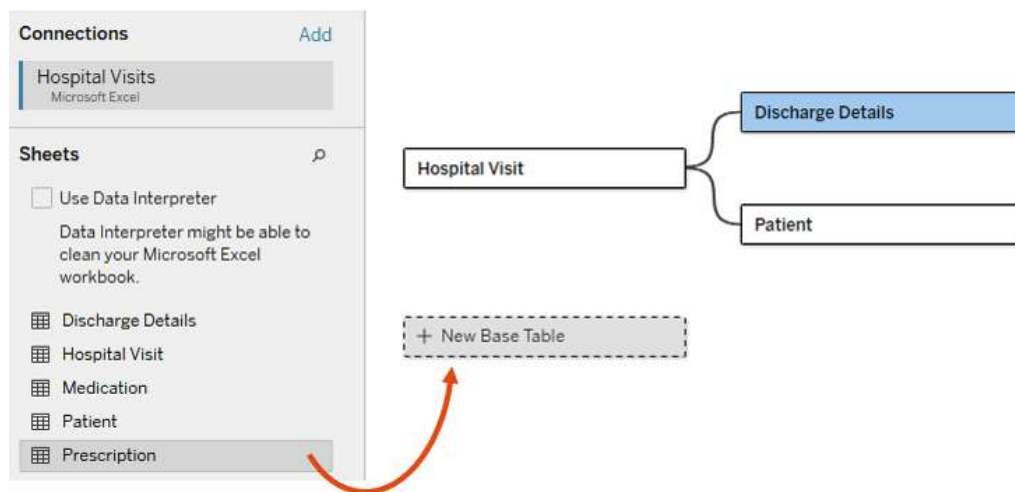


Figure 13.4: Adding a new base table

Hovering over the **Prescription** base table reveals a + indicator that allows us to drag and drop a new relationship to an existing table, like this:

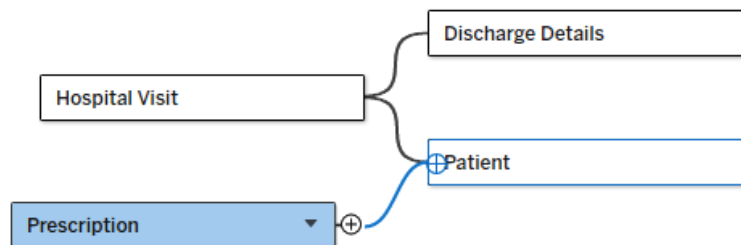


Figure 13.5: Adding a new relationship to an existing table

Again, we'll accept the default settings for the relationship. Finally, we'll add the **Medication** table with a relationship to **Prescription**. The final data model will look like this:

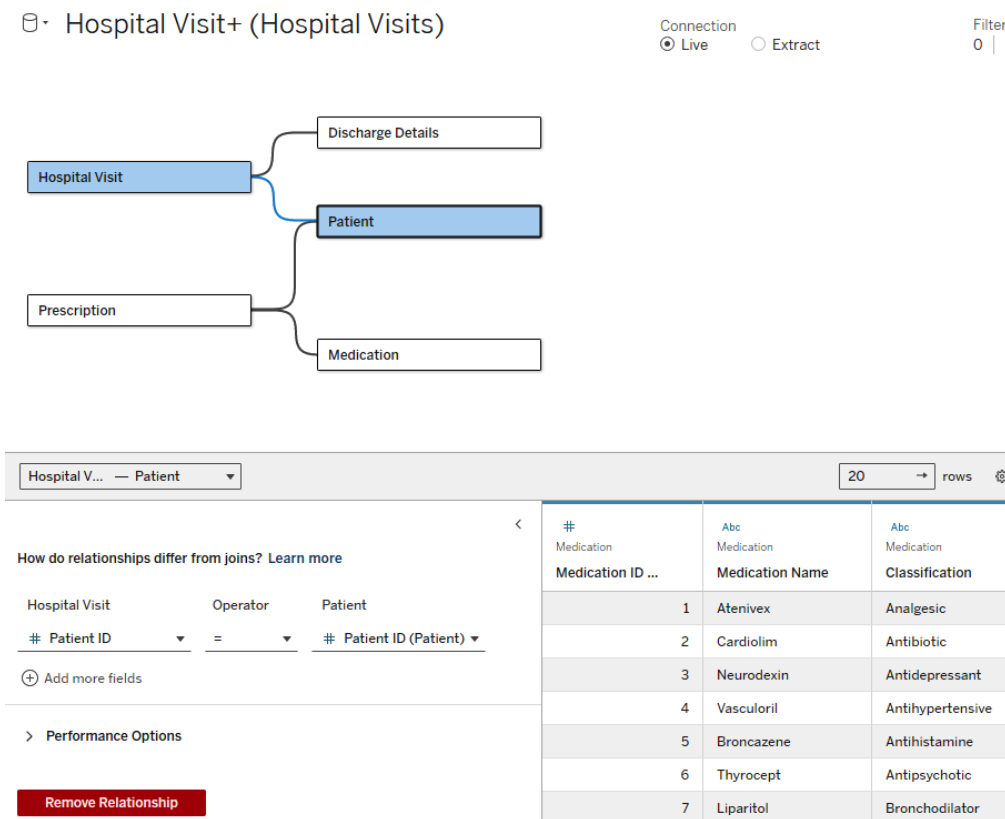


Figure 13.6: The final data model

Our data model consists of two base tables related through the common dimension **Patient** and each having a separate unshared dimension.

Selecting a connector between tables will allow you to edit the relationship in the bottom pane. In Figure 13.6, you'll notice the relationship options for the connector between **Hospital Visit** and **Patient**. Tableau automatically created our relationships because the ID fields had the same name and type between both tables. If necessary, you could manually edit the relationships to change which fields define the relationship.



A **relationship** simply defines which fields connect the tables together. It does not define exactly how the tables relate to each other. We'll discuss the concepts of join types (for example, left join or inner join) later in this chapter, but relationships are not restricted to a certain join type. Instead, Tableau will use the appropriate kind of join as well as the correct aggregations depending on which fields you use in your view. For the most part, you won't have to think about what Tableau is doing behind the scenes, but we'll examine some unique behaviors in the next section.

Also, notice the **Performance Options** drop-down menu in the relationship editor, as shown here:

The screenshot shows the Tableau relationship editor interface. At the top, there are three columns: 'Hospital Visit', 'Operator', and 'Patient'. Under 'Hospital Visit' is a dropdown menu showing '# Patient ID'. Under 'Operator' is an equals sign '='. Under 'Patient' is a dropdown menu showing '# Patient ID (Patient)'. Below these is a button with a plus icon and the text 'Add more fields'. A horizontal line separates this from the 'Performance Options' section. This section has a dropdown arrow and the title 'Performance Options'. Below the title is a paragraph: 'These settings help Tableau optimize queries during analysis. The default settings are recommended, if you aren't sure what to choose. [Learn more](#)'. Under this paragraph are two rows of settings. The first row is 'Cardinality' with two dropdown menus, both set to 'Many'. The second row is 'Referential Integrity' with two dropdown menus, both set to 'Some records match'. At the bottom of the section is a button labeled 'Revert to Default'.

Figure 13.7: The Performance Options section of the relationship editor

These performance options allow Tableau to generate more efficient queries if the nature of the relationship is known. If you do not know the exact nature of the relationship, it is best to leave the options in their default settings as an incorrect setting can lead to incorrect results.

There are two basic concepts covered by the performance options:

- **Cardinality:** This term indicates how many records in one table could potentially relate to the records of another table. For example, we know that one visit matches to only one patient. However, we also know that a patient could have many prescriptions.
- **Referential Integrity:** This term indicates whether we expect all records to find a match or whether some records could potentially be unmatched. For example, we know (from the description of the data in this chapter) that there are patients in the **Patient** table that will not have a match in the **Hospital Visit** table. We also know that some patients will not have discharge records as they are still in the hospital.

If Tableau is able to determine constraints from a relational database, those constraints will be used. Otherwise, Tableau will set the defaults to **Many** and **Some records match**. For the examples in this chapter, we do know the precise nature of the relationships (they are described in the previous section), but we'll accept the performance defaults as the dataset is small enough that there won't be any perceptible performance gain in modifying them.

With our initial data model created, let's take a moment to explore the two layers of the data model paradigm.

Layers of the data model

A data model consists of two layers:

- The **logical layer**: A semantic layer made up of logical tables or objects that are related. Each logical table might be made up of one or more physical tables.
- The **physical layer**: A layer made up of the physical tables that come from the underlying data source. These tables may be joined or unioned together with conventional joins or unions or created from custom SQL statements.

Consider the following screenshot of a canvas containing our five tables:

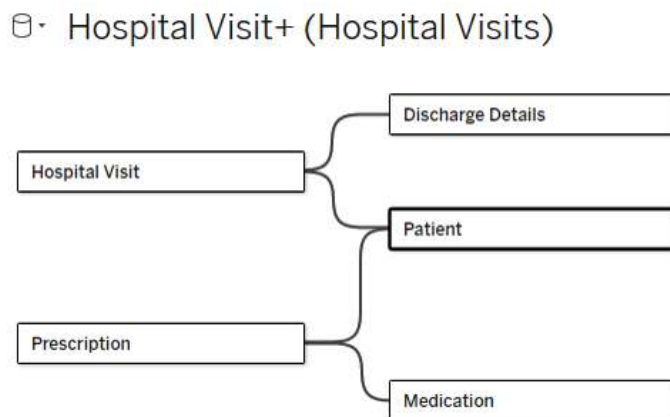


Figure 13.8: The logical layer of the data model

This initial canvas defines the **logical tables** of the data model. A logical table is a collection of data that defines a single structure or object that relates to other logical structures of data. Double-click on the **Hospital Visit** table on the canvas, and you'll see another layer beneath the logical layer:



Figure 13.9: The physical layer of the physical tables that make up Hospital Visit

This is the physical layer for the logical **Hospital Visit** table. This physical layer is made up of physical tables of data—potentially unioned or joined together. In this case, we are informed that **Hospital Visit is made of 1 table**. So, in this case, the logical layer of **Hospital Visit** is identical to the physical

layer underneath. In the *Using joins* section of this chapter, we'll explore examples of how we might extend the complexity of the physical layer with multiple tables while still treating the collection of tables as a single object.

Go ahead and close the physical layer of **Hospital Visit** with the X icon in the upper-right corner. Then navigate to the **Analysis** tab of the workbook for this chapter, and we'll explore how the data model works in practice.

Using the data model

For the most part, working with the data model will be relatively intuitive. There are a few **Data** pane interface features that will help you work with the data model. There are also a few data model behaviors you should learn to expect. Once you are comfortable with them, your analysis will exceed expectations!

The Data pane interface

With multiple tables related in a data model, you'll find the **Data** pane looks something like this:

The screenshot shows the 'Data' pane interface with a tree view of tables. Each table is expanded to show its dimensions (indicated by a blue cube icon) and measures (indicated by a green hash icon). The tables are: Discharge Details, Hospital Visit, Medication, Patient, and Prescription.

Tables	
▼	Discharge Details
	Discharge Date
	Disposition
	Discharge Details ID (Discharg...
	Patient Satisfaction Score
	Discharge Details (Count)
▼	Hospital Visit
	Admit Date
	Admit Type
	Location
	Patient ID
	Primary Diagnosis
	Visit ID
	Discharge Details ID
	Minutes to Service
	Hospital Visit (Count)
▼	Medication
	Classification
	Medication ID (Medication)
	Medication Name
	Medication (Count)
▼	Patient
	Birthdate
	M or F
	Name
	Patient ID (Patient)
	Age at Most Recent Admit
	Patient (Count)
▼	Prescription
	Dose
	Medication ID

Figure 13.10: The Data pane is organized by logical tables and shows a separation of dimensions and measures per table

You'll notice that the **Data** pane is organized by logical tables, with fields belonging to each table. Measures and dimensions are separated by a thin line rather than appearing in different sections as they did previously.

This makes it easier to find the fields relevant to your analysis and also helps you to understand the expected behavior of the data model. Each logical table also has its own number of records field that is named using the convention **Table Name (Count)**, which will appear at the end of the list of measures for that table.

With an overview of how the **Data** pane helps you work with the data model, let's look at some behaviors you can expect from the data model.

Data model behaviors

In the **Analysis** tab of the Starter workbook, experiment with creating different visualizations. Especially note dimensions, what values are shown, and how measures are aggregated. We'll walk through a few examples to illustrate (which you can replicate in the Starter workbook or examine in the Complete workbook).

First, notice that dragging **Name** from the **Patient** table to **Rows** reveals 10 patients. It turns out that not all of these patients have hospital visits, but when we use one or more dimensions from the same logical table, we see the full domain of values in Tableau. That is, we see all the patients, whether or not they visited the hospital. We can verify how many visits each patient had by adding the **Hospital Visit (Count)** field, resulting in the following view:

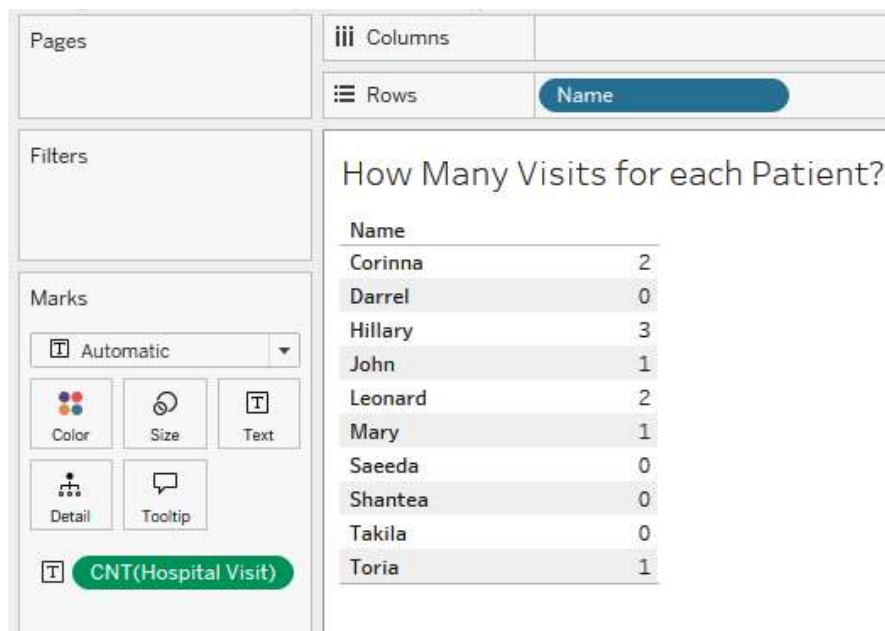


Figure 13.11: All patients are shown, even those with 0 visits

But if we add **Primary Diagnosis** to the table, notice that only 6 out of the 10 patients are shown:

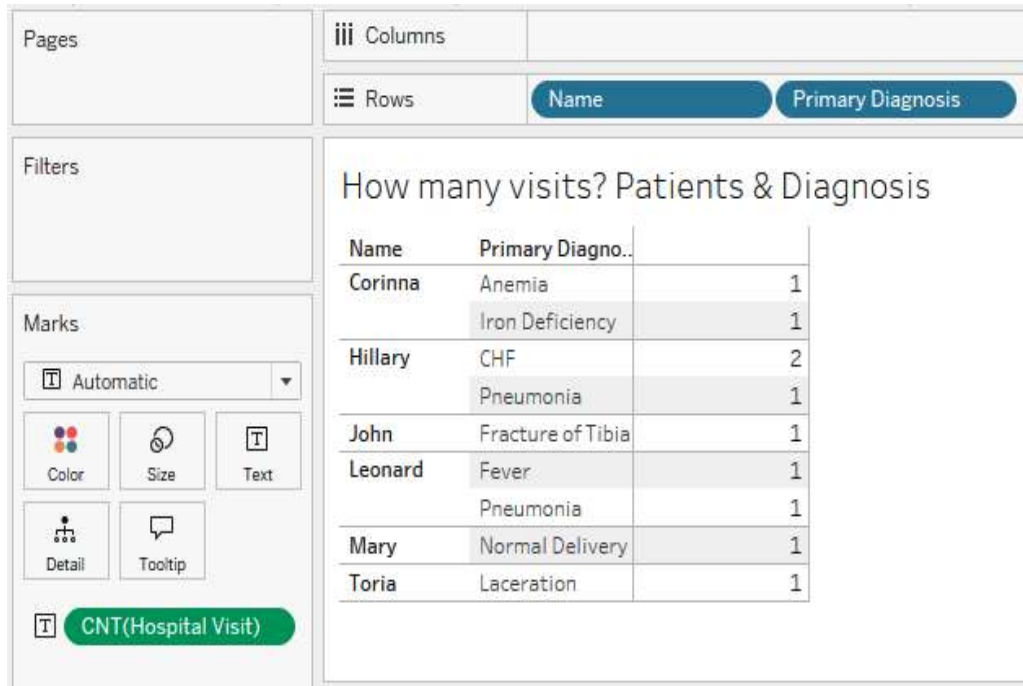


Figure 13.12: Only patients with visits are shown; most patients had a single visit with a given diagnosis, but one came in twice with the same diagnosis

This highlights another behavior: when you include dimensions from two or more tables, only matching values are shown. In essence, when you add **Name** and **Primary Diagnosis**, Tableau shows you patients who exist in both the **Patient** and **Hospital Visit** tables. This is great if you want to focus on only patients who have visited the hospital.

But what if you truly want to see all patients and a diagnosis where applicable? To accomplish that, simply add a measure from the table of the field where you want to see the full domain. In this case, we could add either the **Age at Most Recent Admit** or **Patient (Count)** measures, as both come from the **Patient** table. Doing so results in the following view:

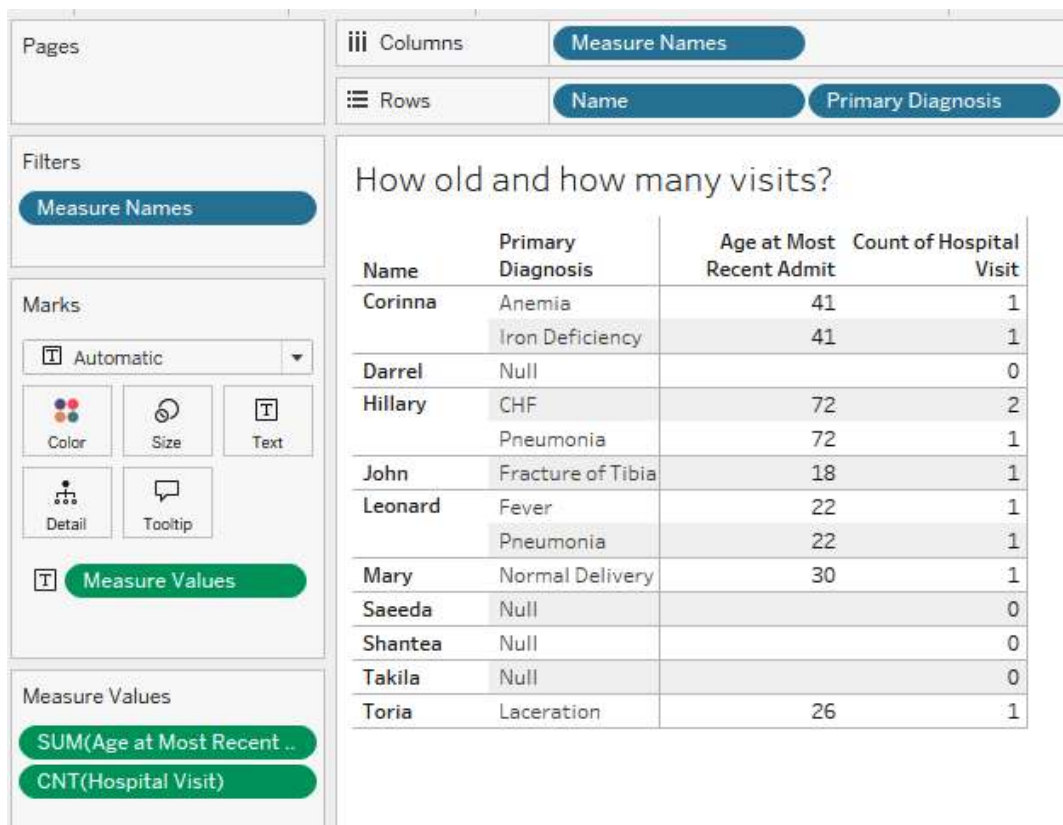


Figure 13.13: All patients are once again shown

Even though the **Age at Most Recent Admit** value is NULL for patients who have never been admitted, simply adding the measure to the view instructs Tableau to show all patients. This demonstrates a third behavior: including a measure from the same table as a dimension will force Tableau to show the full domain of values for that dimension.

Another basic principle of data model behavior is also displayed here. Notice that **Age at Most Recent Admit** is shown for each patient and each diagnosis. However, Tableau does not incorrectly duplicate the value in totals or subtotals.

If you were to add subtotals for each patient in the **Age at Most Recent Admit** and **Count of Hospital Visit** columns, as has been done in the following view, you'd see that Tableau has the correct values:

How old and how many visits? with subtotals

Name	Primary Diagnosis	Age at Most Recent Admit	Count of Hospital Visit
Corinna	Anemia	41.00	1.00
	Iron Deficiency	41.00	1.00
	Total	41.00	2.00
Darrel	Null		0.00
	Total		0.00
Hillary	CHF	72.00	2.00
	Pneumonia	72.00	1.00
	Total	72.00	3.00
John	Fracture of Tibia	18.00	1.00
	Total	18.00	1.00
Leonard	Fever	22.00	1.00
	Pneumonia	22.00	1.00
	Total	22.00	2.00

Figure 13.14: Tableau calculates the subtotals correctly, even though traditional join behavior would have duplicated the values

This final behavior of the data model can be stated as: aggregates are calculated at the level of detail defined by the logical table of the measure. This is similar to how you might use a **Level of Detail (LOD)** expression to avoid a LOD duplication, but you don't have to write the expression or break your flow of thought to solve the problem. The Tableau data model does the hard work for you!

Take some additional time to build out views and visualizations with the data model you've created, and review the following behaviors so you know what to expect and how to control the analysis you want to perform:

- When you use one or more dimensions from the same logical table, you'll see the full domain of values in Tableau
- When you include dimensions from two or more logical tables, only matching values are shown
- Including a measure from the same logical table as a dimension will force Tableau to show the full domain of values for that dimension (even when the previous behavior was in effect)
- Aggregates are calculated at the level of detail defined by the logical table of the measure

With just a bit of practice, you'll find that the behaviors feel natural, and you'll especially appreciate Tableau performing aggregations at the correct level of detail.



When you first create a new data model, it is helpful to run through a couple of quick checks similar to the examples in this section. That will help you gain familiarity with the data model, as well as helping you validate that the relationships are working as you expect.

Let's take a moment to consider how our data model allows us to bring together measures from multiple fact tables.

Multi-fact analysis

We previously noted that when you have two or more base tables you are engaging in **multi-fact analysis**. As we have both **Hospital Visit** and **Prescription** as base tables, we might consider the kinds of questions we can ask based on data from both tables. For example, we might wonder if the number of prescriptions a patient has can predict the number of visits they will make to the hospital. A scatterplot showing the correlation can be made using fields from the two base tables:

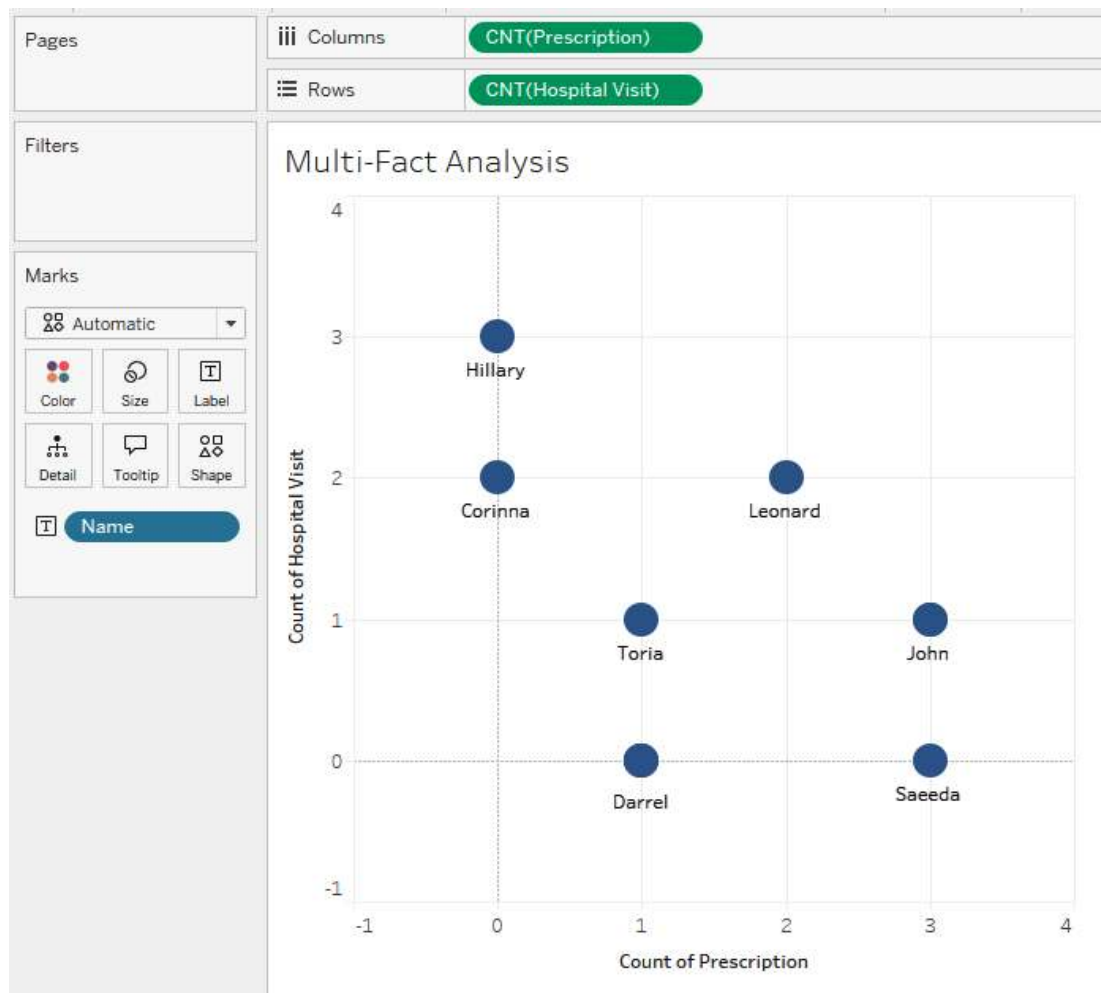


Figure 13.15: The number of prescriptions vs. the number of hospital visits by patients

We used the count of prescriptions from the **Prescription** table and the number of hospital visits from the **Hospital Visit** table. As **Patient** is a common dimension shared between the two, we can slice both measures by **Name**. The resulting scatterplot reveals that we don't have enough data for a statistically sound result. But the idea is interesting and a real-world scenario might yield surprising results!

The power and flexibility of using the Tableau data model to build relationships between tables give you an incredible tool for your analytics!

We'll now turn our focus to learning how to relate data in the physical layer using joins.

Using joins

A **join** at the physical level is a row-by-row matching of the data between tables. We'll look at some different types of joins and then consider how to leverage them in the physical layer of a data model.

Types of joins

In the physical layer, you may specify the following types of joins:

- **Inner:** Only records that match the join condition from both the table on the left and the table on the right will be kept. In the following example, only three matching rows are kept in the results:

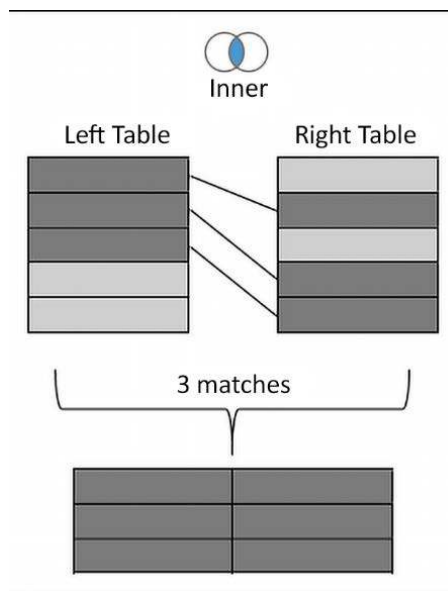


Figure 13.16: Inner join

- **Left:** All records from the table on the left will be kept. Matching records from the table on the right will have values in the resulting table, while unmatched records will contain NULL values for all fields from the table on the right.

In the following example, the five rows from the left table are kept, with NULL results for any values in the right table that were not matched:

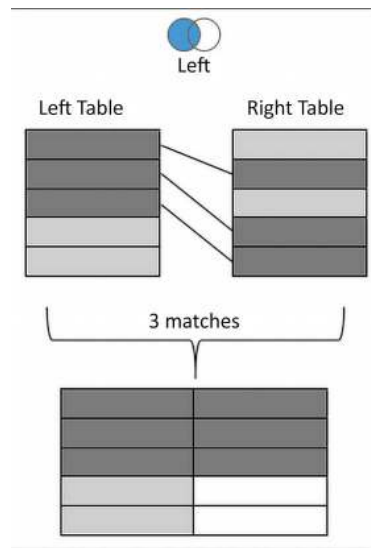


Figure 13.17: Left join

- Right:** All records from the table on the right will be kept. Matching records from the table on the left will result in values, while unmatched records will contain NULL values for all fields from the table on the left. Not every data source supports a right join. If it is not supported, the option will be disabled. In the following example, the five rows from the right table are kept, with NULL results for any values from the left table that were not matched:

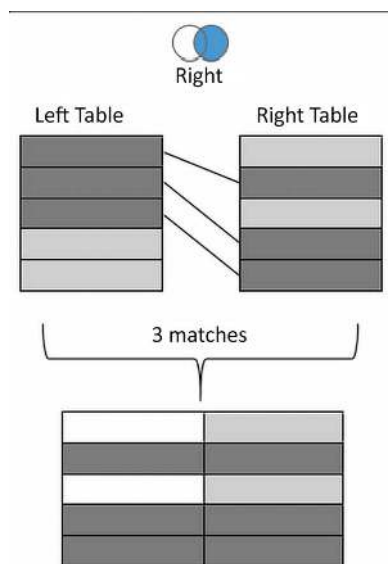


Figure 13.18: Right join

- **Full Outer:** All records from tables on both sides will be kept. Matching records will have values from the left and the right. Unmatched records will have NULL values where either the left or the right matching record was not found. Not every data source supports a full outer join.

If it is not supported, the option will be disabled. In the following example, all rows are kept from both sides with NULL values where matches were not found:

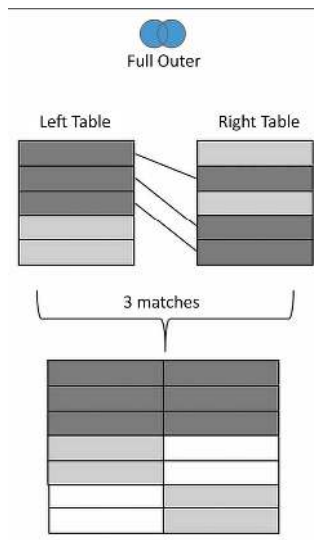


Figure 13.19: Full Outer join

- **Spatial:** This joins together records that match based on the **intersection** (overlap) of spatial objects (we discussed Tableau's spatial features in *Chapter 12, Exploring Mapping and Advanced Geospatial Features*). For example, a point based on the latitude and longitude might fall inside the complex shape defined by a shapefile. Records will be kept for any records where the spatial object in one table overlaps with the spatial object specified for the other:

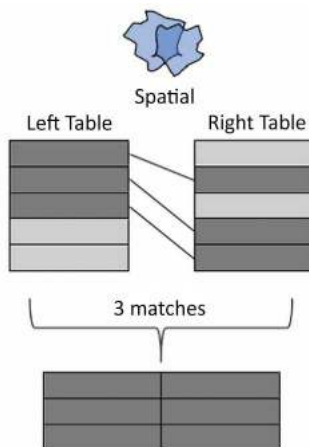


Figure 13.20: Spatial join

When you select spatial objects from the left and right tables, you'll need to specify **Intersects** as the operator between the fields to accomplish a spatial join, as shown in *Figure 13.21*:

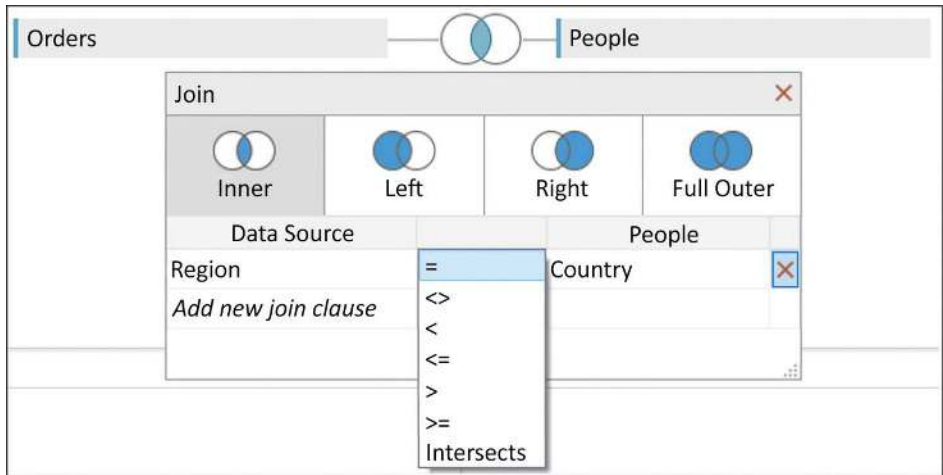


Figure 13.21: Assuming the two fields selected represent spatial objects, the Intersects option will be available

With a solid understanding of join types, let's consider how to use them in the physical layer of Tableau's data model.

Joining tables

Most databases have multiple tables of data that are related in some way. Additionally, you are able to join together tables of data across various data connections for many different data sources.

For our examples here, let's once again consider the tables in the hospital database, with a bit of simplification:

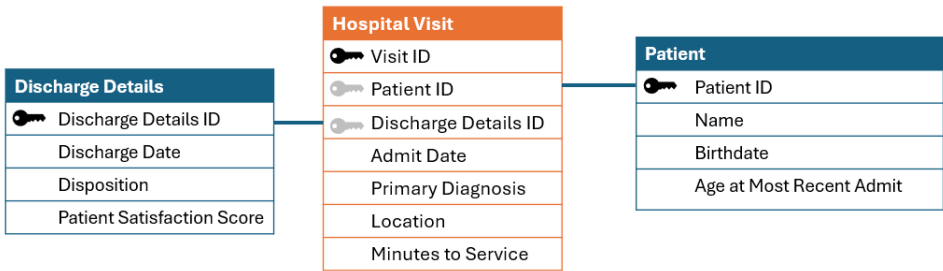


Figure 13.22: The primary Hospital Visit table with Patient and Discharge Details as they might exist in a relational database

Let's consider how we might build a data source using some joins in the physical layer. To follow along, create a new Excel data source in the Chapter 13 Starter.twbx workbook that references the Hospital Visits (Joins).xlsx file in the Chapter 13 directory. You may also examine the connection in the Chapter 13 Complete.twbx workbook.

Just as we did before, we'll start by dragging the **Hospital Visit** table onto the data source canvas such that we have a **Hospital Visit** object in the logical layer, like this:

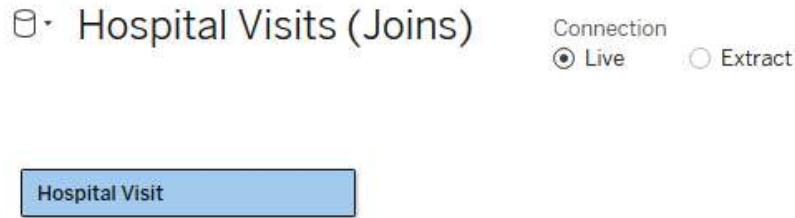


Figure 13.23: After dragging the table onto the canvas, the *Hospital Visit* object is created in the logical layer

At this point, the logical layer object simply contains a single physical table. But we'll extend that next. Double-click on the **Hospital Visit** object to expand the physical layer. It will look like this:

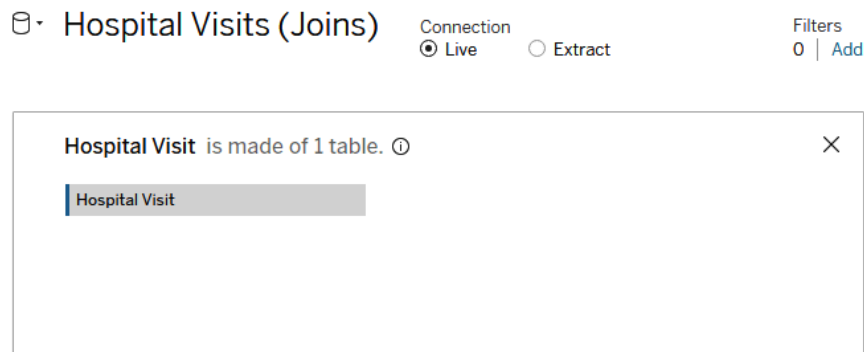


Figure 13.24: The physical layer, which currently consists of a single physical table

You can extend the physical model by adding additional tables. We'll do that here, by adding **Discharge Detail** and **Patient**. As we add them, Tableau will create joins between the tables. Here is what the joins look like in our physical layer:

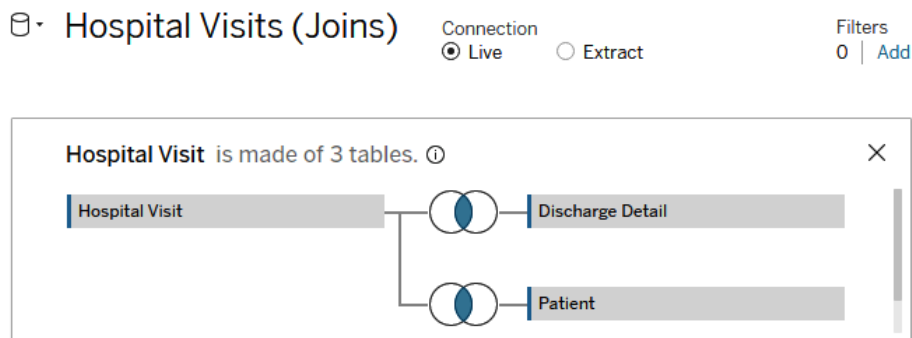


Figure 13.25: *Hospital Visit* joined to *Discharge Detail* and *Patient* in the physical layer

Clicking on one of the join icons will reveal an interface allowing you to configure the join. Here is what clicking on the join between **Hospital Visit** and **Patient** looks like:

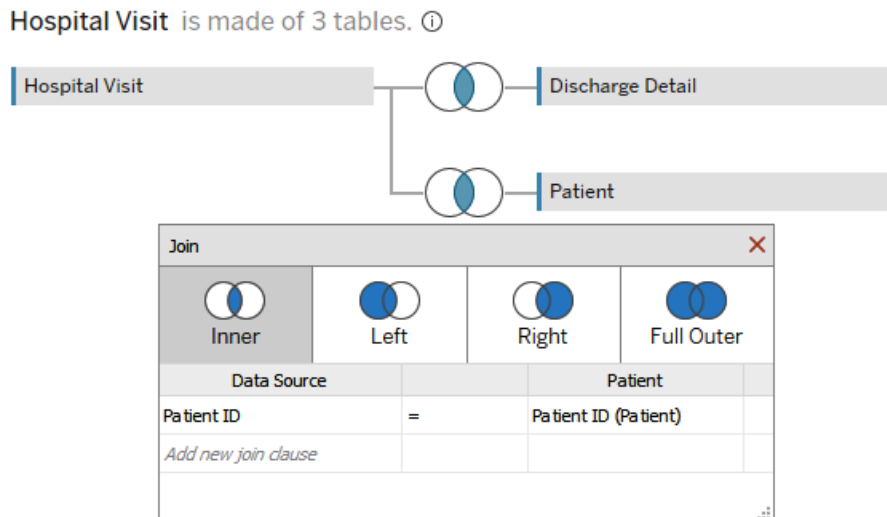


Figure 13.26: The interface to configure the join between *Hospital Visit* and *Patient*

The **Join** interface allows you to specify the join type (**Inner**, **Left**, **Right**, or **Full Outer**) and to specify one or more fields on which to join. Between the fields, you may select which kind of operator joins the fields. The default is equality (=; the fields must be equal), but you may also select inequality (<>; the fields must not be equal), less than (<), less than or equal to (<=), greater than (>), or greater than or equal to (>=). It is important to note that joined fields must be the same data type. The type of join and the field relationships that define the join will determine how many records are returned from the join. We'll take a look at the details in the next section.



Typically, you'll want to start by dragging the primary table onto the physical layer canvas. In this case, **Hospital Visit** contains keys to join additional tables. Additional tables should be dragged and dropped after the primary table.

For now, accept the fields that Tableau automatically detects as shared between the tables (**Discharge Details ID** for **Discharge Details** and **Patient ID** for **Patient**). Change the join to **Discharge Details** to a left join. This means that all hospital visits will be included, even if there has not yet been a discharge. Leave **Patient** as an inner join. This will return only records that are shared between the tables so that only patients with visits will be retained.

Ultimately, the physical layer for **Hospital Visit** will look like this:

Hospital Visit is made of 3 tables. ⓘ

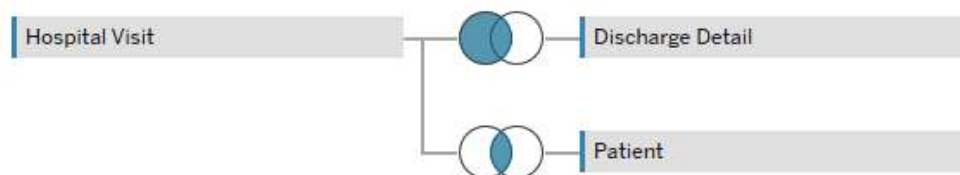


Figure 13.27: The physical layer is made up of three tables joined together

When you close the physical layer, you'll once again see the logical layer, which contains a single object: **Hospital Visit**. That object now contains a join icon, indicating that it is made up of joined physical tables. But it remains a single object in the logical layer of the data model and looks like this:



Figure 13.28: The logical layer contains a single object that is made up of three physical tables

All the joins create what you might think of as one flat table, which can be related together with other logical objects in the data model. Those objects, in turn, might each be made up of a single physical table or multiple physical tables joined together.

We'll wrap up our exploration of joins with a few additional details.

Other join considerations

We conclude this section with some further possibilities to leverage joins, as well as a caution regarding a potential problem that can arise from their use.

Join calculations

In the previous example, we noted that Tableau **joins** rows in one table to rows in another based on fields in the data. You may come across cases where you need to join based on values that are not present in the data but can be derived from the existing data. For example, imagine that there is a **Patient Profile** table that would add significant value to your dataset (it is not part of the sample files). However, it lacks a Patient ID and only has **First Name** and **Last Name** fields.

To join this to our **Patient** table, we would use a **join calculation**. This is a calculation that exists only for the purpose of joining tables together. To create a join calculation, use the drop-down list of fields in the **Join** dialog box and select the final option, **Create Join Calculation...**:

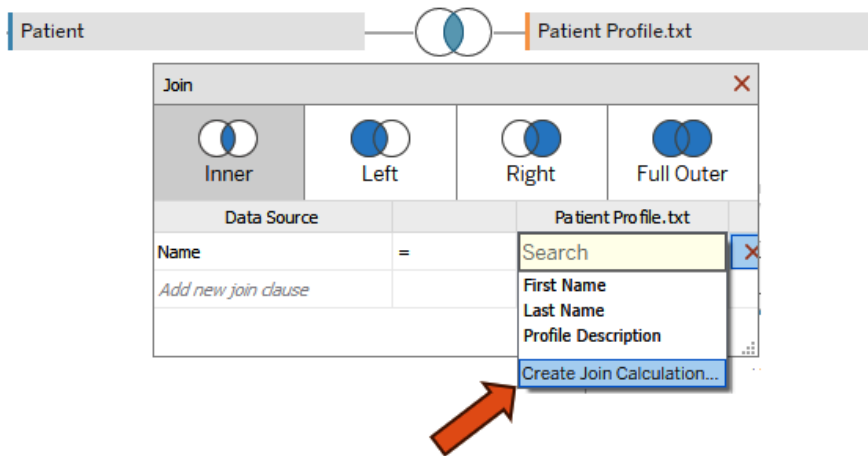


Figure 13.29: You can create a join calculation to aid in forming the correct joins

Selecting this option allows you to write row-level calculations that can be used in the join. This allows you to account for type mismatches by casting a field in one table to a type that matches a field in the other. You might also need to use row-level join calculations to adjust values so they match. For example, our join calculation might have code like `[First Name] + " " + [Last Name]` to return values that match with the **Name** field.



In general, try to avoid joining on text fields, especially in larger datasets, for performance reasons. For many database engines, joining on integers is far more efficient. Also, it is entirely possible for two separate people to share first and last names, so a real-world dataset that followed the structure in this example would be subject to false matches and errors.

If applicable to your data and situation, you may also leverage the geospatial functions mentioned in *Chapter 12, Exploring Mapping and Advanced Geospatial Features*, to create a spatial join between two sources, even when one or both lack specific spatial objects on which to join. For example, if you have **Latitude** and **Longitude**, you might create a join calculation with the code `MAKEPOINT([Latitude], [Longitude])` to find the intersection with another spatial object in another table.

Join calculations can also help when you are missing a field for a join. What if the data you want to join is in another database or file completely? In this scenario, we would consider cross-database joins.

Cross-database joins

With Tableau, you have the ability to join (at the row level) across multiple different data connections. Joining across different data connections is referred to as a **cross-database join**. For example, you can join SQL Server tables with text files or Excel files, or join tables in one database with tables in another, even if they are on a different server. This opens up all kinds of possibilities for supplementing your data or analyzing data from disparate sources.

Consider the hospital data. Though not part of the data included in the Chapter 13 sample data, it would not be uncommon for billing data to be in a separate system from patient care data. Let's say you had a file for patient billing that contained data you wanted to include in your analysis of hospital visits. You would be able to accomplish this as follows:



Figure 13.30: Joining tables or files based on separate data connections

1. Add a new connection to the text file.
2. Select the new **Patient Billing** connection.
3. Drag and drop the **Patient Billing.txt** file as a new table into the physical layer to create a new join.



Each data connection will be color-coded so that you can immediately identify the source of various tables in the designer.

You may use tables from any number of connections at the physical layer of any logical object. This gives you a lot of flexibility and power to create a single unified data model even when your data is scattered across multiple sources.

But whether you are joining data from a different data connection or tables from a single connection, you'll want to watch out! We'll consider some of the unintentional errors that can result from faulty joins next.

The unintentional duplication of data

We conclude with a warning about joins—if you are not careful, you could potentially end up with a few extra rows or many times the number of records you were expecting. Let's consider a theoretical example.

Let's say you have a Visit table like this:

Visit ID	Patient Name	Doctor ID
1	Kirk	1
2	Picard	2
3	Sisko	3

Table 13.1: A table showing patients' names

And a Doctor table like this:

Doctor ID	Doctor Name
1	McCoy
2	Crusher
3	Bashir
2	Pulaski

Table 13.2: A table showing doctors' names

Notice that the value 2 for Doctor ID occurs twice in the Doctor table. Joining the table on equality between the Doctor ID value will result in duplicate records, regardless of which join type is used. Such a join would result in the following dataset:

Visit ID	Patient Name	Doctor ID	Doctor Name
1	Kirk	1	McCoy
2	Picard	2	Crusher
3	Sisko	3	Bashir
2	Picard	2	Pulaski

Table 13.3: A table aligning patients and doctors

This will greatly impact your analysis. For example, if you were counting the number of rows to determine how many patient visits had occurred, you'd overcount. There are times when you may want to intentionally create duplicate records to aid in analysis; however, often, this will appear as an unintentional error.



In addition to the danger of unintentionally duplicating data and ending up with extra rows, there's also the possibility of losing rows where values you expected to match didn't match exactly. Get into the habit of verifying the row count of any data sources where you use joins.

A solid understanding of joins will not only help you as you leverage Tableau Desktop and Tableau Server, but it will also give you a solid foundation when we look at Tableau Prep in *Chapter 15, Taming Data with Tableau Prep*. For now, let's wrap up this chapter with a brief look at blends.

Using blends

Data blending allows you to use data from multiple data sources (not just multiple connections in a single source) in the same view. Often, these sources may be of different types. For example, you can blend data from Snowflake with data from Excel. You can blend Google Sheet data with a spatial file. Data blending also allows you to compare data at different levels of detail. Let's consider the basics and a simple example.



As Tableau's data model capabilities have matured, the cases where data blending is the best solution have diminished. It is still an important concept to understand but try to consider alternative solutions using the data model. We'll summarize cases when to use or not use various methods of relating data in the section *When to use a data model, joins, or blends* at the end of the chapter.

Data blending is done at an aggregate level and involves different queries sent to each data source, unlike joining, which is done at the row level and (conceptually) involves a single query to a single data source. A simple data blending process involves several steps, as shown in the following diagram:

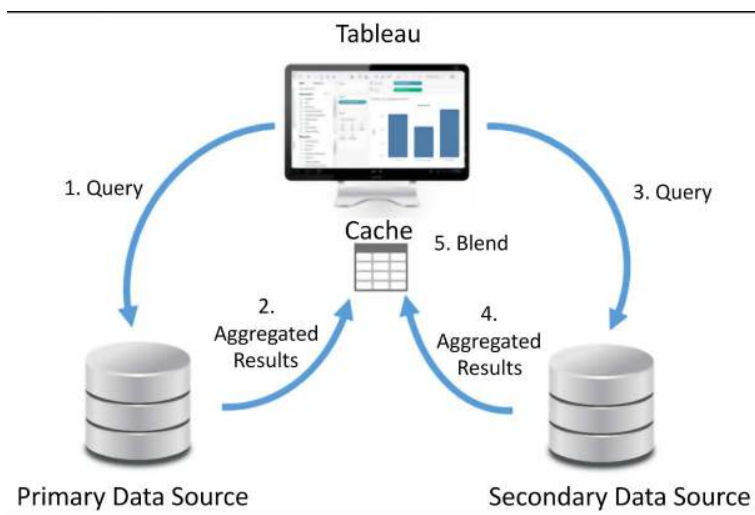


Figure 13.31: How Tableau accomplishes blending

We can see the following from the preceding diagram:

1. Tableau issues a query to the primary data source.
2. The underlying data engine returns aggregate results.
3. Tableau issues another query to the secondary data source. This query is filtered based on the set of values returned from the primary data source for dimensions that link the two data sources.
4. The underlying data engine returns aggregate results from the secondary data source.
5. The aggregated results from the primary data source and the aggregated results from the secondary data source are blended together in the cache.

It is important to note that data blending is different from joining. Joins are accomplished in a single query and results are matched row by row. Data blending occurs by issuing two separate queries and then blending together the aggregate results.

There can only be one primary source, but there can be as many secondary sources as you desire. Steps 3 and 4 are repeated for each secondary source. When all aggregated results have been returned, Tableau matches the aggregated rows based on linking fields.



When you have more than one data source in a Tableau workbook, whichever source you use first in a view becomes the primary source for that view.

Blending is view-specific. You can have one data source as the primary in one view and the same data source as the secondary in another. Any data source can be used in a blend, but OLAP cubes, such as in SQL Server Analysis Services, must be used as the primary source.

In many ways, blending is similar to creating a data model with two or more objects. In many cases, the data model will give you exactly what you need without using blending. However, you have some flexibility with blending because you can change which fields are related at a view level rather than at an object level.

Linking fields are dimensions that are used to match data blended between primary and secondary data sources. Linking fields define the level of detail for the secondary source. Linking fields are automatically assigned if fields match by name and type between data sources.

Otherwise, you can manually assign relationships between fields by selecting, from the menu **Data | Edit Blend Relationships**, as shown in *Figure 13.32*:

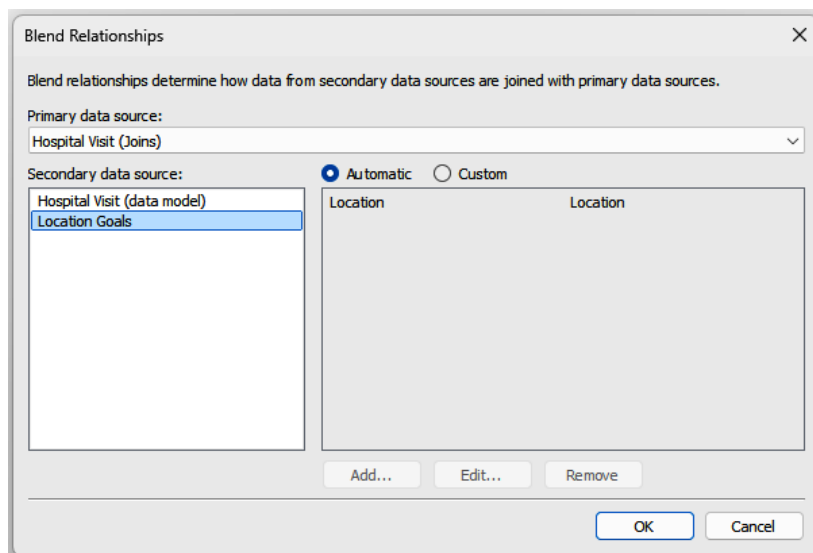


Figure 13.32: Defining blending relationships between data sources

The **Blend Relationships** window will display the relationships recognized between different data sources. You can switch from **Automatic** to **Custom** to define your own linking fields.

Linking fields can be activated or deactivated to blend in a view. Linking fields used in the view will usually be active by default, while other fields will not. You can, however, change whether a linking field is active or not by clicking on the link icon next to a linking field in the **Data** pane.



Additionally, use the **Edit Data Relationships** screen to define the fields that will be used for **cross-data source filters**. When you use the drop-down menu of a field on **Filters** in a view, and select **Apply to Worksheets | All Using Related Data Sources**, the filter works across data sources.

Let’s take this from the conceptual to the practical with an example.

Using blending to show progress toward a goal

Let’s look at a quick example of blending in action. Let’s say you have the following table representing the service goals of various locations throughout the hospital when it comes to serving patients:

Location	Avg. Minutes to Service Goal
Inpatient Surgery	30
Outpatient Surgery	40
ICU	30
OBGYN	25
Lab	120

Table 13.4: A table showing serving time goals for different hospital departments

This data is contained in a simple text file, named `Location Goals.txt`, in the *Chapter 14* directory. Both the starter and complete workbooks already contain a data source defined for the file.

We'll start by creating a simple bar chart from the **Hospital Visit (Joins)** data source you created previously, showing **Average Minutes to Service by Location** like so:

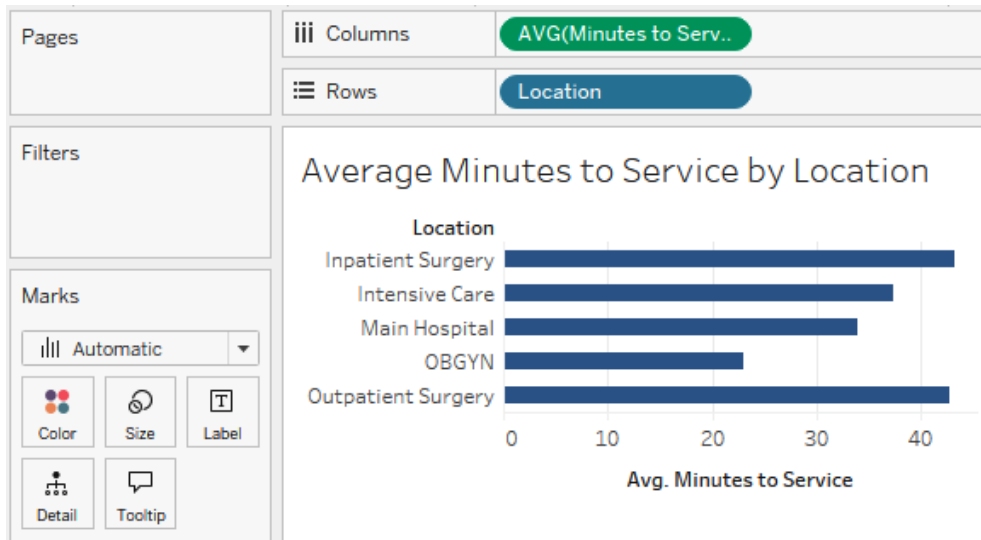


Figure 13.33: Average Minutes to Service by Location

Then, in the **Data** pane, we'll select the **Location Goals** data source. Observe the **Data** pane shown here:

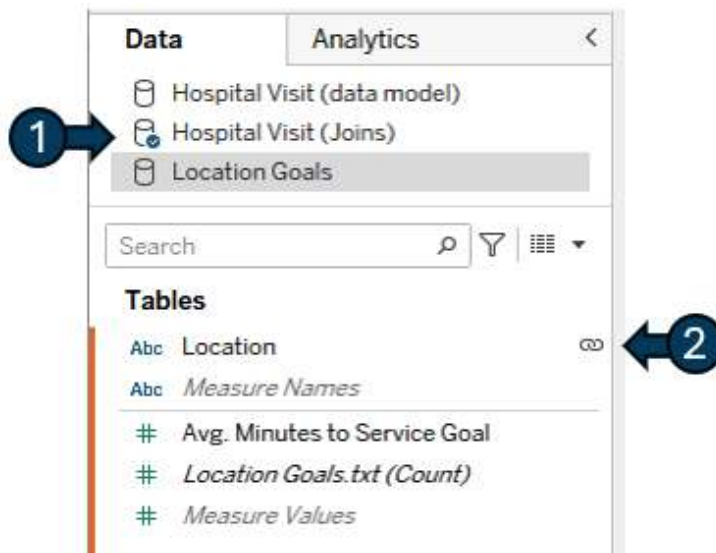


Figure 13.34: Hospital Visit (Joins) is shown as the Primary data source and Location in the Location Goals data source is indicated as a linking field

Notice the following features of the interface:

1. The blue checkmark on the **Hospital Visit (Joins)** data source indicates that the data source is primary.

2. Tableau recognizes **Location** as a linking field and indicates that it is active with a connected link icon.



You may click on the link icon to switch from active to inactive or vice versa to control the level of detail at which aggregations are done in the secondary source.

For now, click on **Avg. Minutes to Service Goal** in the **Data** pane and select **Bullet Graph** from **Show Me**, as indicated here:

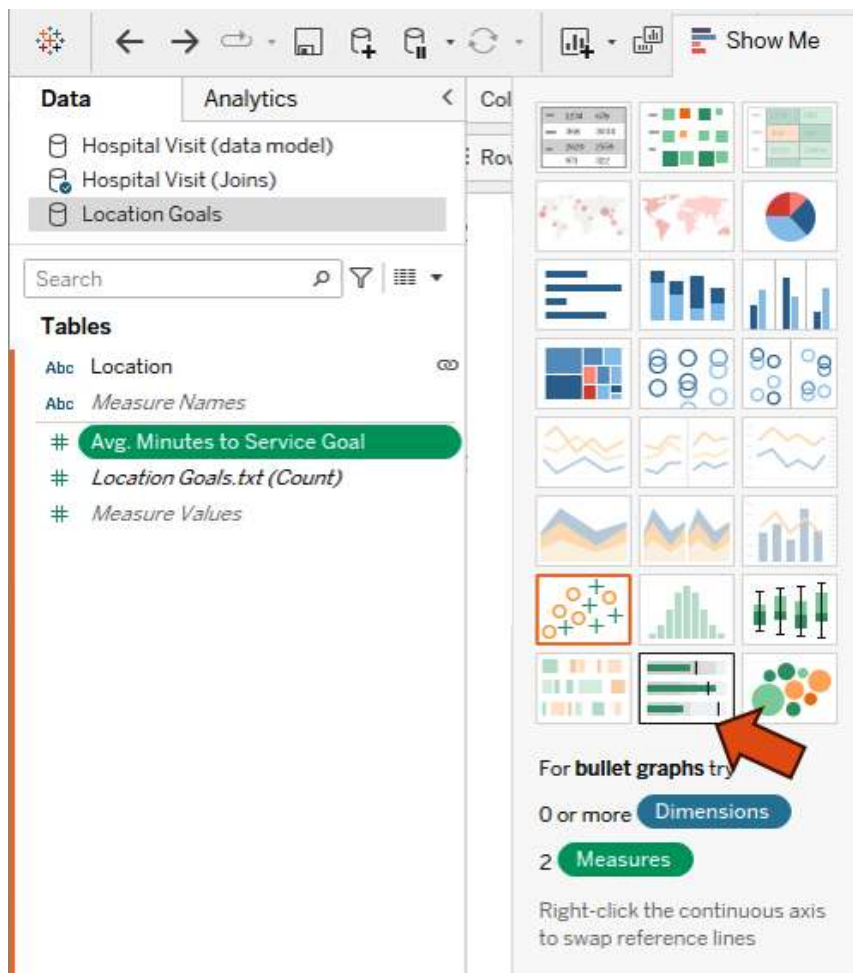


Figure 13.35: You may drag and drop fields from secondary sources into the view or use **Show Me**

You may have to right-click on the **Avg. Minutes to Service** axis in the view and select the **Swap Reference Line** fields to ensure the goal is the reference line and the bar is the actual metric.

Sometimes ShowMe will use the wrong fields for the axis and the reference lines of a bullet chart. If this occurs, you can right-click the axis and select **Swap Reference Line Fields** to switch the field used for the reference line and the field used for the axis:

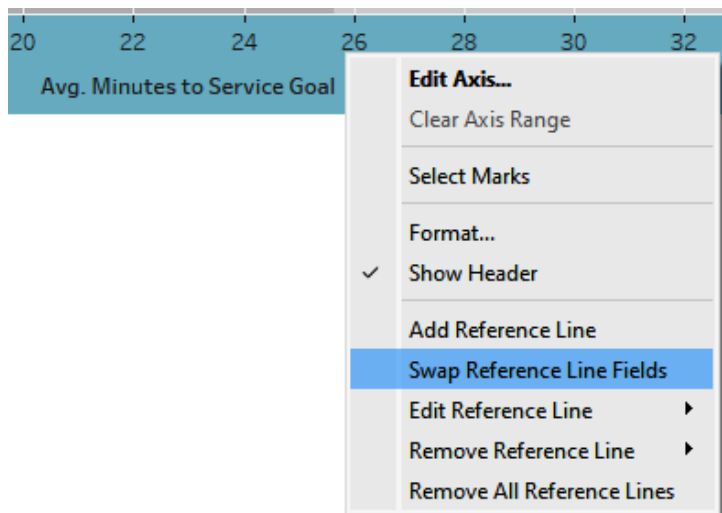


Figure 13.36: Swapping the reference line fields

Your final view should now look like this:

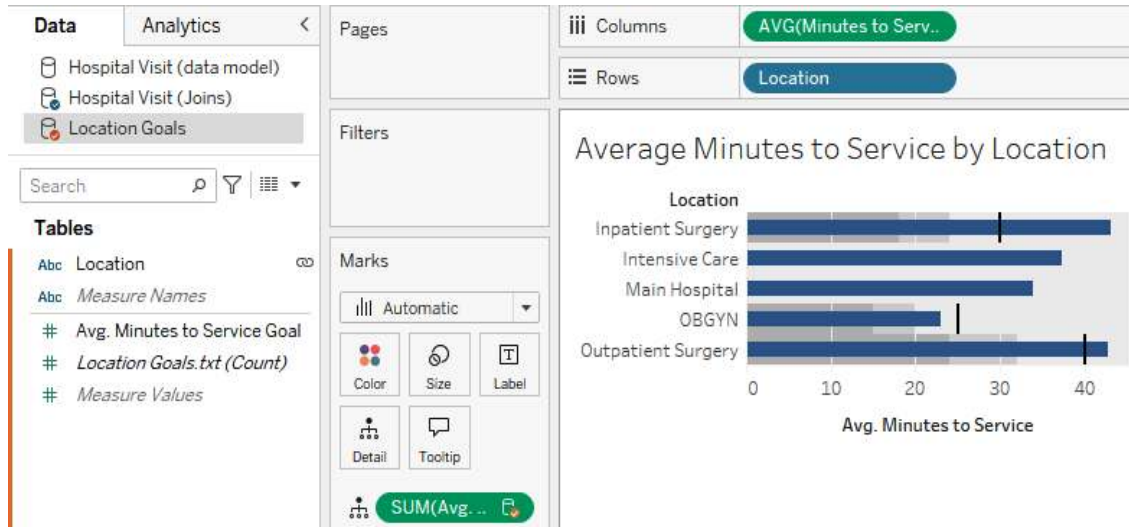


Figure 13.37: A view created from a primary source and a secondary source

Notice that both the **Hospital Visits (Joins)** data source and the **Location Goals** data source are used in this view. **Hospital Visit (Joins)** is the primary data source (indicated by a blue checkmark), while **Location Goals** is the secondary source (indicated by the orange checkmark). The **Avg. Minutes to Service Goal** field on **Detail** in the **Marks** card is secondary and also indicated by an icon with an orange checkmark.

You may also see that **Main Hospital** and **Intensive Care** do not have goals indicated in the view. Recall that the primary data source is used to determine the full list of values shown in the view. **Main Hospital** is in the primary source but does not have a match in the secondary source. It is shown in the view, but it does not have a secondary source value.

Intensive Care also does not have a secondary value. This is because the corresponding value in the secondary source is **ICU**. Values must match exactly between the primary and secondary sources for a blend to find matches. However, blends also take into account aliases.



An **alias** is an alternate value for a dimension value that will be used for display and data blending. Aliases for dimensions can be changed by right-clicking on row headers or using the menu on the field in the view or the **Data** pane and selecting the **Aliases** option.

We can change the alias of a field by right-clicking on the row header in the view and using the **Edit Alias...** option, as shown here:

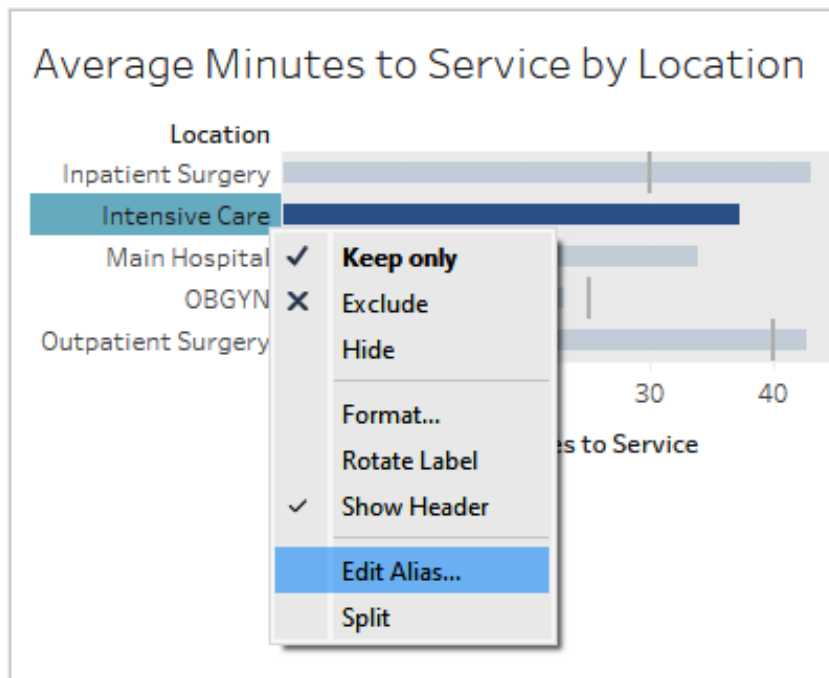


Figure 13.38: Using the **Edit Alias...** option

If we change the alias to ICU, a match is found in the secondary source and our view reflects the secondary value:

Average Minutes to Service by Location

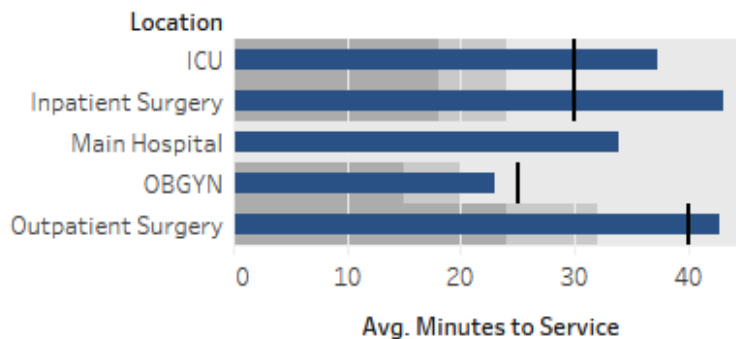


Figure 13.39: ICU now finds a match in the secondary source

Location Goals.txt also contains a goal for Lab. However, Lab is not a location in our primary data source and is, therefore, not shown in this view. If we were to create a new view and use **Location Goals** as the primary source, it would show.



You'll recall that, with the data model, we had approaches to show all the values from a logical object regardless of whether there were matches in other objects. With data blending, we will only ever see values from the primary data source. This is a limitation of data blending that should be considered as you decide the best approach for your analysis. Other limitations include unsupported aggregations (such as COUNTD) and not being able to blend between sources that have multiple logical objects in the data model.

We've covered quite a few options regarding how to relate data in this chapter. Let's just take a moment to consider when to use these different techniques.

When to use a data model, joins, or blends

In one sense, every data source you create using the latest versions of Tableau will use a data model. Even data sources using one physical table will have a corresponding object in the logical layer of a data model. But when should you relate tables using the data model, when should you join them together in the physical layer, and when should you employ blending?

Most of the time, there's no single right or wrong answer. However, here are some general guidelines to help you think through when it's appropriate to use a given approach.

In general, use a **data model** to relate tables:

- When joins would make correct aggregations impossible or require complex LOD expressions to get accurate results
- When joins would result in unwanted duplication of data
- When you need flexibility in showing full domains of dimensions versus only values that match across relationships
- When you are uncertain of a data source and don't know what type of join to use

In general, use **joins** at the physical level:

- When the results of joining two or more tables in a specific way clearly define a logical object that you want to use in the logical layer
- When you want to do a spatial join
- When you want to specifically control the type of join used in your analysis
- When the performance of the data model is less efficient than it would be with the use of joins

In general, use **blending** when:

- You need to relate data sources that cannot be joined or related using a data model (such as OLAP cubes)
- You need flexibility to “fix” matching using aliases
- You need flexibility to adjust which fields define the relationship differently in different views

As you grow in confidence while using each of these approaches, you'll be able to better determine which makes sense in a given circumstance.

Summary

You now have several techniques to turn to when you need to relate tables of data together. The data model gives a paradigm for relating logical tables of data together. It introduces a few behaviors when it comes to showing the full and partial domains of dimensional values, but it also greatly simplifies aggregations by taking into account the natural level of detail for the aggregation. In the physical layer, you have the option of joining together physical tables.

We covered the various types of joins and discussed possibilities for using join calculations and cross-database joins for ultimate flexibility. We briefly discussed how data blending works and saw a practical example. Finally, we examined a broad outline of when to turn to each approach. You now have a broad toolset to tackle data in different tables or even in different databases or files.

We'll expand our thoughts about data and data modeling a bit in the next chapter. There, we'll explore the data structures that work best for real-world data as we mold and shape it to meet our analytics goals!

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



14

Structuring Messy Data to Work Well in Tableau

So far, most of the examples we've looked at in this book assume that the data used is structured well and is fairly clean. Data in the real world isn't always so pretty. Maybe it's messy or it doesn't have a good structure. It may be missing values or have duplicate values, or it might have the wrong **Level of Detail (LOD)**.

How can you deal with this type of messy data? In the previous chapter, we considered how Tableau's data model can be used to relate data in different tables. We will consider Tableau Prep Builder as a robust way to clean and structure data in the next chapter. Much of the information in this chapter will be an essential foundation for working with Tableau Prep Builder. But it is also foundational for sound analytics.

For now, we'll focus on some of the basic data structures that work well in Tableau and some of the additional techniques you can use to get data into those structures. We'll keep our discussion limited to native Tableau features in this chapter, but much of what you learn will apply to Tableau Prep Builder in the next chapter. By the end of this chapter, you'll have a solid foundation of understanding of what constitutes a good data structure. Knowing which data structures work well with Tableau is key to understanding how you will be able to resolve certain issues.

In this chapter, we'll focus on some principles for structuring data to work well with Tableau, as well as some specific examples of how to address common data issues. This chapter will cover the following topics:

- Structuring data for Tableau
- The four basic data transformations
- Overview of advanced fixes for data problems

We'll start by discussing which data structures work well for Tableau.

Structuring data for Tableau

We've already seen that Tableau can connect to nearly any data source. Whether it's a built-in direct connection to a file or database or data obtained through a custom **Web Data Connector (WDC)** or through the Tableau data extract API to generate an extract, no data is off limits! However, there are certain structures that make data easier to work with in Tableau.

There are two key points to ensure a good data structure that works well with Tableau:

- Every record of a source data connection should be at a meaningful LOD
- Every measure contained in the source should match the LOD of the data source or possibly be at a higher LOD, but it should never be at a lower LOD

For example, let's say you have a table of test scores with one record per classroom in a school. Within the record, you may have three measures: the average GPA for the classroom, the number of students in the class, and the number of students at the entire school:

School	Classroom	Average GPA	Number of Students	Number of Students (School)
Pickaway Elementary	4th Grade	3.78	153	1,038
Pickaway Elementary	5th Grade	3.73	227	1,038
Pickaway Elementary	6th Grade	3.84	227	1,038
McCord Elementary	4th Grade	3.82	94	915
McCord Elementary	5th Grade	3.77	89	915
McCord Elementary	6th Grade	3.84	122	915

Table 14.1: Table of test scores

The first two measures (**Average GPA** and **Number of Students**) are at the same LOD as the individual record of data (per classroom in the school). **Number of Students (School)** is at a higher LOD (per school). As long as you are aware of this, you can do a careful analysis (or use Tableau's data model to relate a table of student count per school).

However, you would have a data structure issue if you tried to store each student's GPA in the class record. If the data were structured in a way that attempted to store all of the students' GPAs per grade level (for example, containing a column for each student or a single field containing a comma-separated list of student scores), we'd need to do some work to make the data more usable in Tableau.

Understanding the LOD of the source (often referred to as **granularity**) is vital. Every time you connect to a data source, the very first question you should ask and answer is: what does a single record represent? If, for example, you were to drag and drop the **Table (Count)** field into the view and observe 1,000 records, then you should be able to complete the statement I have 1,000 _____. It could be 1,000 students, 1,000 test scores, or 1,000 schools. Having a good grasp of the granularity of the data will help you to avoid poor analysis and allow you to determine if you even have the data that's necessary for your analysis.



Tip

A quick way to find the LOD of your data is to place the **Table (Count)** field on the **Text** shelf, and then try different dimensions (from the same logical object) on the **Rows** shelf. When all of the rows display a 1, and the total that's displayed in the lower-left status bar equals the number of records in the data, then that dimension (or combination of dimensions) uniquely identifies a record and defines the lowest LOD of your data.

With an understanding of the overarching principles regarding the granularity of data, let's move on and understand certain data structures that allow you to work seamlessly and efficiently in Tableau. Sometimes, it may be preferable to restructure the data at the source using tools such as Alteryx or Tableau Prep Builder. However, at times, restructuring the source data isn't possible or feasible. For example, you may not have write access to the database or it may be a cloud-based data source that has a predefined structure. We'll take a look at some options in Tableau for those cases. For now, let's consider what kinds of data structures work well with Tableau.

Well-structured data in Tableau

The two key points to a good structure that we mentioned in the previous section should result in a data structure where a single measure is contained in a single column. You may have multiple different measures, but any single measure should almost never be divided into multiple columns. Often, the difference is described as **wide data** versus **tall data**.

Wide data

Wide data is not typically a good structure for visual analysis in Tableau.

Wide data describes a structure in which a measure in a single row is spread over multiple columns. This data is often more **human-readable than tall data**. Wide data often results in fewer rows with more columns.

Here is an example of what wide data looks like in a table of population numbers:

Country Name	1960	1961	1962	1963
Afghanistan	8,774,440	8,953,544	9,141,783	9,339,507
Australia	10,276,477	10,483,000	10,742,000	10,950,000

Table 14.2: Table of population numbers

Notice that the LOD for this table is a row for every country. However, the single measure (population) is not stored in a single column. This data is wide because it has a single measure (population) that is being divided into multiple columns (a column for each year). The wide table violates the second key to a good structure since the measure is at a lower LOD than the individual record (per country per year, instead of just per country).

Tall data

Tall data is typically a good structure for visual analysis in Tableau.

Tall data describes a structure in which each distinct measure in a row is contained in a single column. Tall data often results in more rows and fewer columns.

Consider the following table, which represents the same data as earlier but in a tall structure:

Country Name	Year	Population
Afghanistan	1960	8,774,440
Afghanistan	1961	8,953,544
Afghanistan	1962	9,141,783
Afghanistan	1963	9,339,507
Afghanistan	1964	9,547,131
Australia	1960	10,276,477
Australia	1961	10,483,000
Australia	1962	10,742,000
Australia	1963	10,950,000
Australia	1964	11,167,000

Table 14.3: Table of population numbers in a tall structure

Now, we have more rows (a row for each year for each country). Individual years are no longer separate columns and population measurements are no longer spread across those columns. Instead, one single column gives us a dimension of **Year** and another single column gives us a measure of **Population**. The number of rows has increased, while the number of columns has decreased. Now, the measure of population is at the same LOD as the individual row, and so visual analysis in Tableau will be much easier.

Let's take a look at the difference this makes in practice.

Wide versus tall data in Tableau

You can easily see the difference between wide and tall data in Tableau. Here is what the **wide data** table looks like in the left **Data** window:

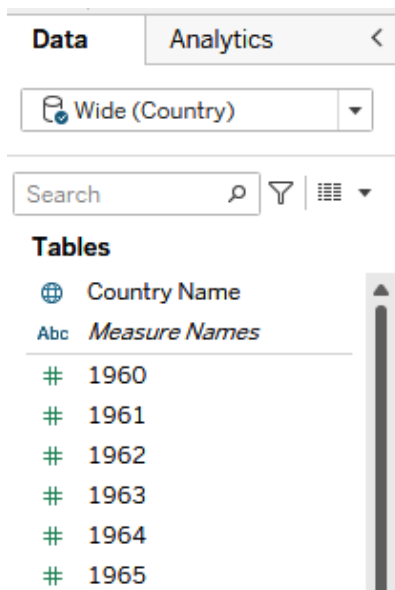


Figure 14.1: The wide data has a measure for every year

As we'd expect, Tableau treats each column in the table as a separate field. The wide structure of the data works against us. We end up with a separate measure for each year. If you wanted to plot a line graph of population per year, you would likely struggle. What dimension represents the date? What single measure can you use for the population?

This isn't to say that you can't use wide data in Tableau. For example, you might use **Measure Names**/**Measure Values** to plot all the **Year** measures in a single view, like this:

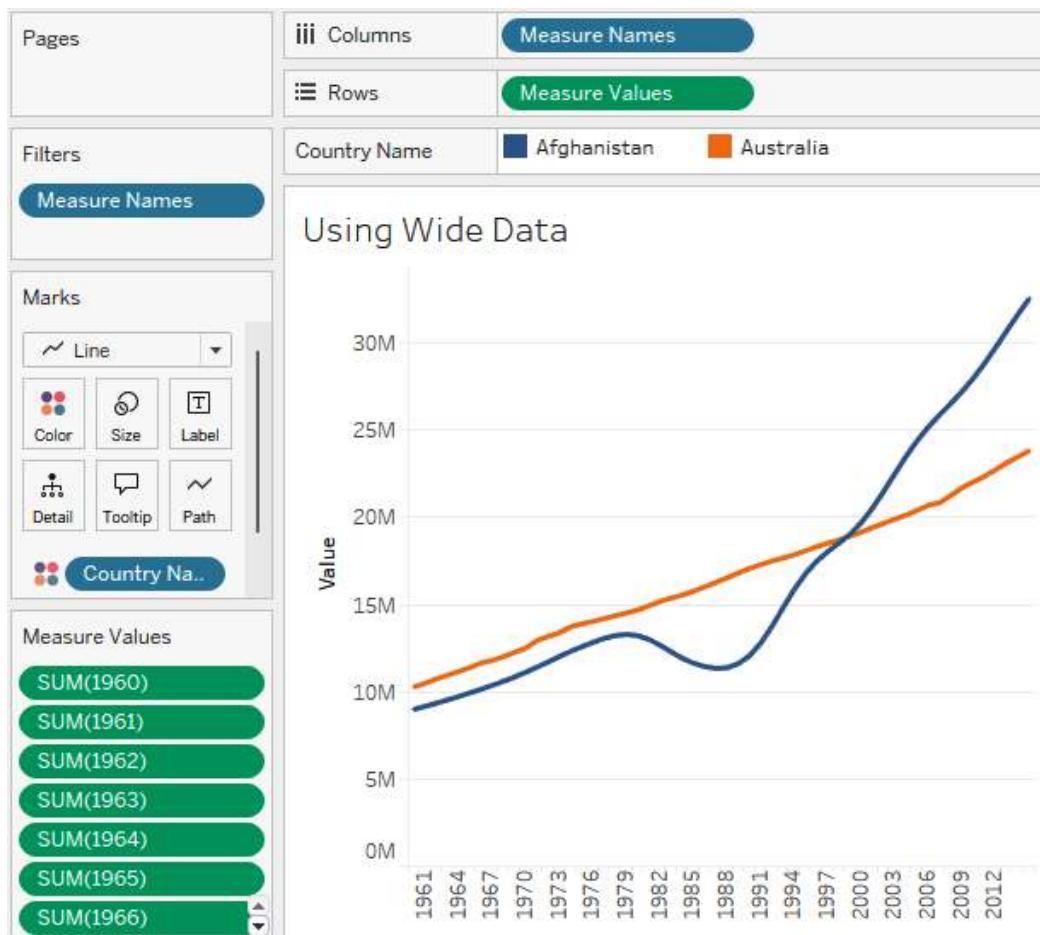


Figure 14.2: The wide data can still be used but in a complex and limited way

You'll notice that every **Year** field has been placed in the **Measure Values** shelf. The good news is that you can create visualizations from poorly structured data like this. The bad news is that views are often more difficult to create and certain advanced features may not be available.

The following limitations apply to the view in *Figure 14.2* based on the wide data structure:

- Because Tableau doesn't have a date dimension or integer, you cannot use forecasting
- Because Tableau doesn't have a date or continuous field in **Columns**, you cannot enable trend lines
- Because each measure is a separate field, you cannot use quick table calculations (such as running total, percent difference, and others)

- Determining things such as the average population across years will require a tedious custom calculation instead of simply changing the aggregation of a measure
- You don't have an axis for the date (just a series of headers for the measure names), so you won't be able to add reference lines

In contrast, the **tall data** looks like this in the **Data** pane:

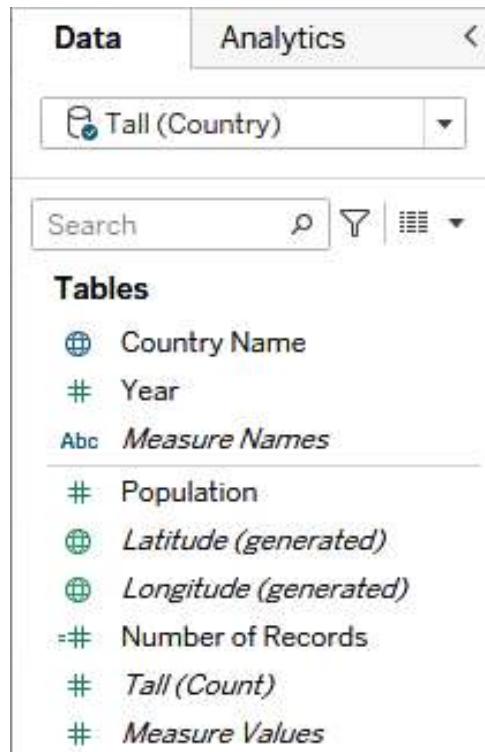


Figure 14.3: The tall data has a Year dimension and a single Population measure

This data source is much easier to work with. There's only one measure (**Population**) and a **Year** dimension to slice the measure. If you want a line chart of the population by year, you can simply drag and drop the **Population** and **Year** fields into **Columns** and **Rows**. Forecasting, trend lines, clustering, averages, standard deviations, and other advanced features will all work in the way you expect them to.

You can see that the resulting visualization is much easier to create in Tableau, using only three active fields:

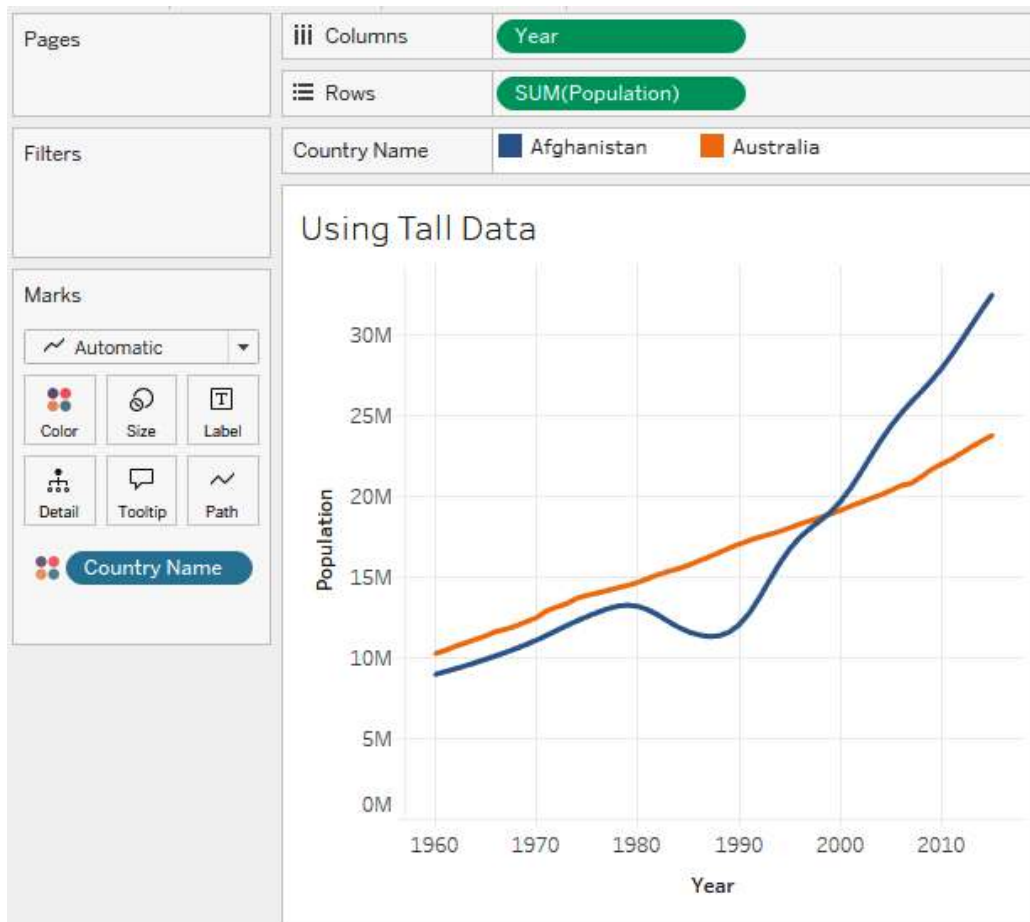


Figure 14.4: The view is much easier to create in Tableau with the tall data

Next, we'll consider another good structure for data that works well with Tableau.

Star schemas

Assuming they are well designed, **star schema** data models (typically used in data warehouses or data marts) work very well with Tableau because they have well-defined granularity, measures, and dimensions. Additionally, if they are implemented well, they can be extremely efficient to query. This allows for an ergonomic experience when using live connections in Tableau.

Star schemas are so named because they consist of a single fact table surrounded by related dimension tables, thus forming a star pattern. **Fact tables** contain measures at a meaningful granularity, while **dimension tables** contain attributes for various related entities.



Note

We saw in the previous chapter that the Tableau data model is conceptually similar to a star schema. A star schema makes it very easy to build a Tableau data model with the fact table(s) as your base table(s) and dimension tables as related tables.

The following diagram illustrates a simple star schema with a single fact table (**Hospital Visit**) and three dimension tables (**Patient**, **Primary Physician**, and **Discharge Details**):

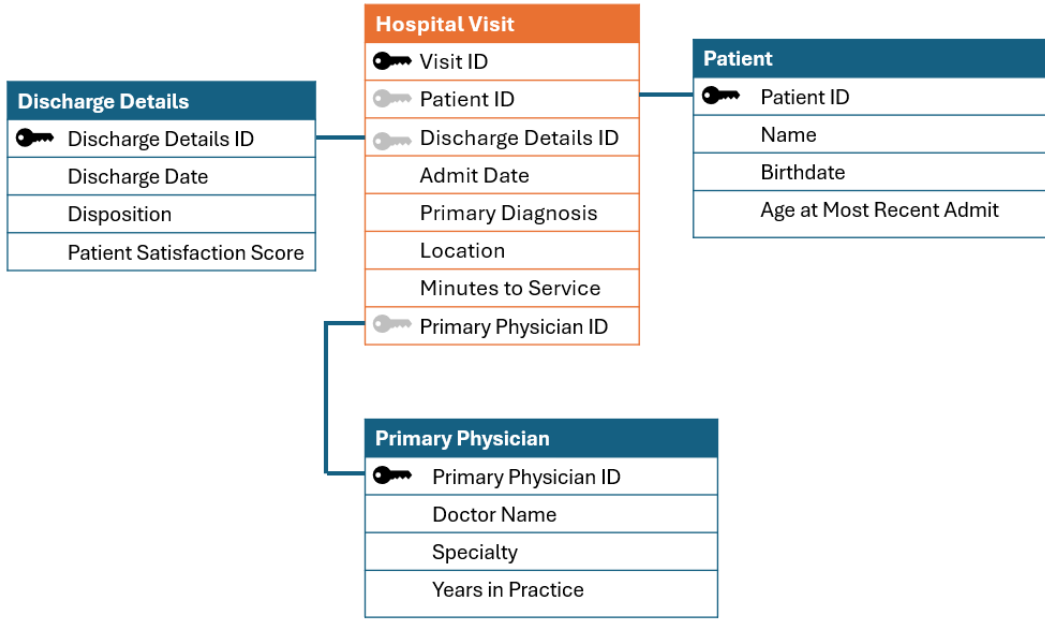


Figure 14.5: A simple star schema

Fact tables are joined to the related dimension using what is often called a **surrogate key** or **foreign key**, which references a single dimension record. The fact table defines the level of granularity and contains measures. In this case, **Hospital Visit** has a granularity of one record for each visit. In this simple example, each visit is for one patient who saw one primary physician and was discharged. The **Hospital Visit** table explicitly stores a measure of Visit Duration and implicitly defines another measure of Number of Visits (as the row count).

Note



Data modeling purists would point out that date values have been stored in the fact table (and even some of the dimensions). They would, instead, recommend having a date dimension table with extensive attributes for each date and only a surrogate (foreign) key stored in the fact table.

A date dimension can be very beneficial, especially if you want to have a common date shared among multiple base tables in a Tableau data model.

At the same time, Tableau's built-in date hierarchy and extensive date options make storing a date in the fact table a viable option. Consider using a date dimension if you want a common date in multi-fact analysis, need specific attributes of dates that are not available in Tableau (for example, indication of days that are corporate holidays), have complex fiscal years, or need to support legacy BI reporting tools.

A well-designed star schema allows the use of **inner joins** since every surrogate key should reference a single dimension record. In cases where dimension values are not known or not applicable, special dimension records are used. For example, a hospital visit that is not yet complete (the patient is still in the hospital) may reference a special record in the **Discharge Details** table marked as Not yet discharged.

You've already worked with a similar structure of data in the previous chapter, where you experienced the differences between relating tables in the logical layer and the physical layer of the data model. Feel free to return to *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*, to review the concepts.

Well-implemented star schemas are particularly attractive for use in **live connections** because Tableau can improve performance by implementing join culling. **Join culling** is Tableau's elimination of unnecessary joins in queries, which can, at times, be performance-intensive for the database engine.

For example, if you were to place the **Physician Name** on rows and the average of **Visit Duration** on columns to get a bar chart of average visit duration per physician, then joins to the **Treatment** and **Patient** tables may not be needed. Tableau will eliminate unnecessary joins as long as you are using a simple star schema with joins that are only from the central fact table and have referential integrity enabled in the source, or allow Tableau to assume referential integrity. For tables joined in the physical layer, select the data source connection from the data menu, or use the context menu from the data source connection and choose **Assume Referential Integrity**. For relationships in the logical layer of the data model, use the performance options for referential integrity for each applicable relationship.

Having considered some examples of good structures, let's turn our attention to some basic transformations that will help us to transform a poorly structured dataset into one with a good structure that is easier to work with in Tableau.

The four basic data transformations

In this section, we'll give you an overview of some basic transformations that can fundamentally change the structure of your data. We'll start with an overview and then look at some practical examples.

Overview of transformations

In Tableau (and Tableau Prep), there are four basic data transformations. The following definitions broadly apply to most databases and data transformation tools, but there are some details and terminology that are Tableau-specific:

- **Pivots:** This indicates the transformation of columns to rows or rows to columns. The resulting dataset will be narrower and taller with fewer columns and more rows (columns to rows) or wider and shorter with more columns and fewer rows (rows to columns).



Note

A columns-to-rows transformation, which is called a **pivot** in Tableau, is often called an **unpivot** or **transpose** in other tools.

- **Unions:** This indicates the appending of rows from one table of data to another, with the matching columns aligned. The resulting data structure is a single table containing rows from all unioned tables and columns that match between the original tables, along with unmatched columns containing NULL values for tables that did not have them.
- **Joins:** This indicates the row-by-row matching of two or more tables, resulting in a data structure that includes columns from all tables. The number of rows is based on the type of join and how many matches are found.
- **Aggregations:** This indicates the rolling up of a table to a higher LOD such that there will be a row for each unique set of values for all dimensions used for grouping, along with other values that are aggregated (such as a sum, min, max, or other aggregation).

In order to fully understand these definitions, we'll turn to some illustrations and practical examples.

Pivots (along with some simple data cleaning)

The Excel workbook *World Population Data.xlsx*, which is included in the Chapter 14 directory of the resources that are included with this book, is typical of many Excel documents.

Here is what it looks like:

	A	B	C	D	E	F	G	H
1	World Population Data							
3	<i>This is sample data only.</i>							
4	<i>Accuracy and completeness is not guaranteed.</i>							
5								
6	Country Name and Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964
7	Aruba (ABW)	Population, total	SP.POP.TOTL	54208	55435	56226	56697	57029
8	Andorra (AND)	Population, total	SP.POP.TOTL	13414	14376	15376	16410	17470
9	Afghanistan (AFG)	Population, total	SP.POP.TOTL	8774440	8953544	9141783	9339507	9547131
10	Angola (AGO)	Population, total	SP.POP.TOTL	4965988	5056688	5150076	5245015	5339893
11	Albania (ALB)	Population, total	SP.POP.TOTL	1608800	1659800	1711319	1762621	1814135
12	United Arab Emirates (ARE)	Population, total	SP.POP.TOTL	89608	97727	108774	121574	134411
13	Argentina (ARG)	Population, total	SP.POP.TOTL	20623998	20959241	21295290	21630854	21963952

Figure 14.6: The World Population Data Excel file

Excel documents such as this are often more human-readable but contain multiple issues when used for data analysis in Tableau. The issues in this particular document include the following:

- Excessive headers (titles, notes, and formatting) that are not part of the data
- Merged cells
- Country name and code in a single column
- Columns that are likely unnecessary (**Indicator Name** and **Indicator Code**)
- The data is wide, that is, there is a column for each year, and the population measure is spread across these columns within a single record

When we initially connect to the Excel document in Tableau, the connection screen will look similar to Figure 14.7, as follows:

Data (World Population Data)						
Data						
Data 57 fields 239 rows						
Table Details	Abc Data	Abc Data	Abc Data	# Data	# Data	# Data
	F1	F2	F3	F4	F5	F6
	World Population Data	null	null	null	null	null
	This is sample data only. Ac...	null	null	null	null	null
	Country Name and Code	Indicator Name	Indicator Code	1,960	1,961	1,962
	Aruba (ABW)	Population, total	SP.POPTOTL	54,208	55,435	56,226
	Andorra (AND)	Population, total	SP.POPTOTL	13,414	14,376	15,376
	Afghanistan (AFG)	Population, total	SP.POPTOTL	8,774,440	8,953,544	9,141,783
	Angola (AGO)	Population, total	SP.POPTOTL	4,965,988	5,056,688	5,150,076

Figure 14.7: World Population Data.xlsx on Tableau's Data Source page

The data preview reveals some of the issues resulting from the poor structure:

- Since the column headers were not in the first Excel row, Tableau gave the defaults of **F1**, **F2**, and so on to each column
- The title **World Population Data** and the note about sample data were interpreted as values in the **F1** column
- The actual column headers are treated as a row of data (the third row)

Fortunately, these issues can be addressed in the connection window. First, we can correct many of the excessive header issues by turning on the **Tableau Data Interpreter**, a component that specifically identifies and resolves common structural issues in Excel or Google Sheets documents. You'll find the **Use Data Interpreter** option located under **Sheets** in the left pane:

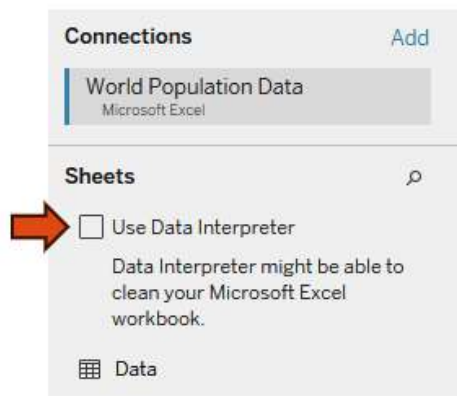


Figure 14.8: The Use Data Interpreter option

After checking the option, the data preview reveals much better results:

Data	Abc Data	Abc Data	# Data	# Data	# Data
Country Name and Code	Indicator Name	Indicator Code	1960	1961	1962
Aruba (ABW)	Population, total	SP.POPT.TOTL	54,208	55,435	56,226
Andorra (AND)	Population, total	SP.POPT.TOTL	13,414	14,376	15,376
Afghanistan (AFG)	Population, total	SP.POPT.TOTL	8,774,440	8,953,544	9,141,783
Angola (AGO)	Population, total	SP.POPT.TOTL	4,965,988	5,056,688	5,150,076
Albania (ALB)	Population, total	SP.POPT.TOTL	1,608,800	1,659,800	1,711,319

Figure 14.9: Tableau Data Interpreter fixes many of the common issues found in Excel (and similar) data sources



Tip

Clicking on the **Review the results...** link that appears under the checkbox will cause Tableau to generate a new Excel document that is color-coded to indicate how the data interpreter parsed the Excel document. Use this feature when you need to verify that Tableau has correctly interpreted the Excel document and retained the data you expected.

Observe the elimination of the excess headers and the correct names of the columns. However, a few additional issues will still need to be corrected.

First, we can hide the **Indicator Name** and **Indicator Code** columns if we feel they are not useful for our analysis. Clicking on the drop-down arrow on a column header reveals a menu of options.

Selecting **Hide** will remove the field from the connection and even prevent it from being stored in extracts:

Indicator Name	Indicator Code
Population, total	SP.POP.TOTL
Population, total	SP.POP.TOTL
Population, total	SP.POP.TOTL
Population, total	SP.POP.TOTL
Population, total	SP.POP.TOTL
Population, total	SP.POP.TOTL
Population, total	SP.POP.TOTL
Population, total	SP.POP.TOTL

Figure 14.10: You can hide fields from the Data Source screen

Second, we can use the **Split** option on the same menu to split the **Country Name and Code** column into two columns so that we can work with the name and code separately. In this case, the **Split** option on the menu works well and Tableau perfectly splits the data, even removing the parentheses from around the code. In cases where the split option does not initially work, try the **Custom Split...** option.



Tip

You may need to scroll to the right of all the fields to see the newly split fields.

We'll also use the **Rename** option to rename the split fields from Country Name and Code - Split 1 and Country Name and Code - Split 2 to **Country Name** and **Country Code**, respectively. Then, we'll **Hide** the original Country Name and Code field.

At this point, most of the data structure issues have been remedied. However, you'll recognize that the data is in a **wide** format. We have already seen the issues that we'll run into:

 Data	 Data	 Data	 Data	 Data	 Data
Country Name and Code	1960	1961	1962	1963	1964
Aruba (ABW)	54,208	55,435	56,226	56,697	57,029
Andorra (AND)	13,414	14,376	15,376	16,410	17,470
Afghanistan (AFG)	8,774,440	8,953,544	9,141,783	9,339,507	9,547,131
Angola (AGO)	4,965,988	5,056,688	5,150,076	5,245,015	5,339,893

Figure 14.11: After some cleaning, the data is still in an undesirable wide structure

Our final step is to **pivot** the year columns. This means that we'll reshape the data in such a way that every country will have a row for every year. Select all the year columns by clicking on the 1960 column, scrolling to the far right, and holding *Shift* while clicking on the 2013 column. Finally, use the drop-down menu on any of the year fields and select the **Pivot** option:

 Data	 Data	 Data	 Data	
2010	2011	2012	2013	
101,597	101,932	102,384	102,911	
77,907	77,865	78,360	79,218	
28,397,812	29,105,480	29,824,536	30,551,674	
19,549,124	20,180,490	20,820,525	21,471,618	



- Rename
- Copy Values
- Hide
- Create Calculated Field...
- Pivot**
- Merge Mismatched Fields

Figure 14.12: Using Pivot with a set of selected columns

The result is two columns (**Pivot field names** and **Pivot field values**) in place of all the year columns:

Data	Abc Pivot	# Pivot
Country Name and Code	Pivot Field Names	Pivot Field Values
Aruba (ABW)	1960	54,208
Aruba (ABW)	1961	55,435
Aruba (ABW)	1962	56,226
Aruba (ABW)	1963	56,697
Aruba (ABW)	1964	57,029

Figure 14.13: The result of the pivot

Rename the two new columns to **Year** and **Population**. Your dataset is now narrow and tall instead of wide and short.

Finally, notice that the icon on the **Year** column indicates that it is recognized by Tableau as a text field. Clicking on the icon will allow you to change the data type directly. In this case, selecting **Date** will result in NULL values, but changing the data type to **Number (whole)** will give you integer values that will work well in most cases:

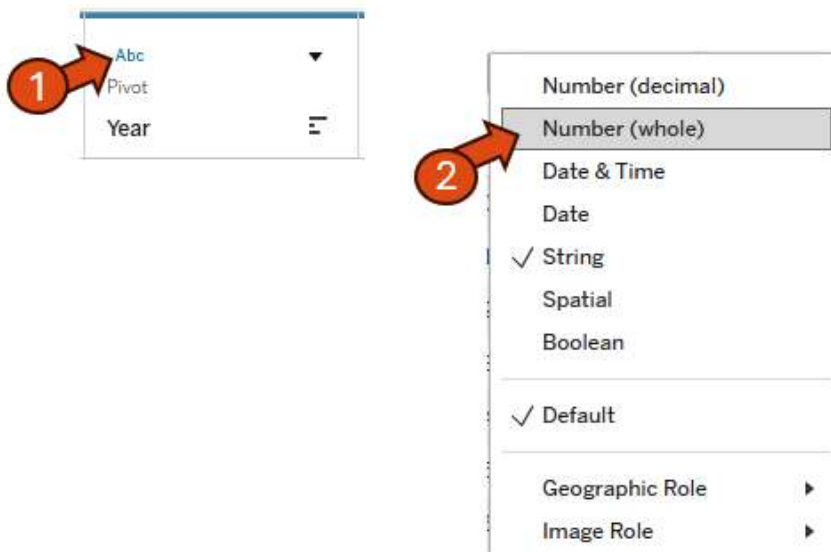


Figure 14.14: You can change the data types of the fields on the Data Source page



Tip

Alternatively, you could use the first drop-down menu in the **Year** field and select **Create Calculated Field....** This would allow you to create a calculated field, which you might name **Year (date)**, which parses the year string as a date using code like `DATE (DATEPARSE ("yyyy", [Year] ))`. This code will parse the string and then convert it into a simple date without a time. You can then hide the original **Year** field. You can hide any field, even if it is used in calculations, as long as it isn't used in a view. This leaves you with a very clean dataset.

The final cleaned, pivoted dataset is far easier to work with in Tableau than the original:

 Data	 Pivot	 Pivot	 Calculation	 Calculation
Country Name and Code	Year	Population	Country Name	CountryCode
Aruba (ABW)	1960	54,208	Aruba	ABW
Aruba (ABW)	1961	55,435	Aruba	ABW
Aruba (ABW)	1962	56,226	Aruba	ABW
Aruba (ABW)	1963	56,697	Aruba	ABW

Figure 14.15: The cleaned and pivoted dataset

The data interpreter, cleaning options, and ability to pivot data (columns to rows) in Tableau make working with many datasets far easier. Next, we'll take a look at unions.

Unions

Often, you may have multiple individual files or tables that, together, represent the entire set of data. For example, you might have a process that creates a new monthly data dump as a new text file in a certain directory. Or, you might have an Excel file where data for each department is contained in a separate sheet.

A **union** is a concatenation of data tables that brings together rows of each table into a single table. For example, consider the following three tables of data:

Name	Occupation	Bank account balance
Luke	Farmer	\$2,000
Leia	Princess	\$50,000
Han	Smuggler	-\$20,000

Table 14.4: Originals

Name	Occupation	Bank account balance
Watto	Junk Dealer	\$9,000
Darth Maul	Face Painter	\$10,000
Jar Jar	Sith Lord	-\$100,000

Table 14.5: Prequels

Name	Occupation	Bank account balance
Rey	Scavenger	\$600
Poe	Pilot	\$30,000
Kylo	Unemployed	\$0

Table 14.6: Sequels

A union of these tables would give a single table containing the rows of each individual table:

Name	Occupation	Bank account balance
Luke	Farmer	\$2,000
Leia	Princess	\$50,000
Han	Smuggler	-\$20,000
Watto	Junk Dealer	\$9,000
Darth Maul	Face Painter	\$10,000
Jar Jar	Sith Lord	-\$100,000
Rey	Scavenger	\$600
Poe	Pilot	\$30,000
Kylo	Unemployed	\$0

Table 14.7: Union of tables 14.4, 14.5, and 14.6

Tableau allows you to union together tables from file-based data sources, including the following:

- Text files (.csv, .txt, and other text file formats)
- Sheets (tabs) within Excel documents
- Subtables within an Excel sheet
- Multiple Excel documents
- Google Sheets
- Relational database tables

**Tip**

Use the **Data Interpreter** feature to find subtables in Excel or Google Sheets. They will show up as additional tables of data in the left sidebar.

To create a union in Tableau, follow these steps:

1. Create a new data source from the menu, toolbar, or **Data Source** screen, starting with one of the files you wish to be part of the union. Then, drag any additional files into the **Drag table to union** drop zone just beneath the existing table on the canvas (in either the logical or physical layer; although, the union technically exists in the physical layer):

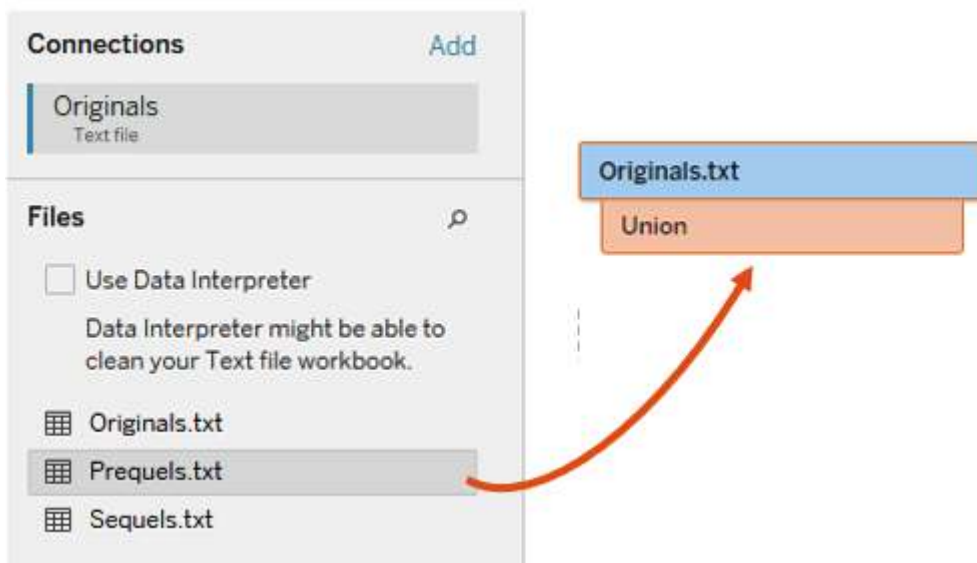


Figure 14.16: You may create unions by dragging and dropping tables or files directly under existing tables on the canvas

2. Once you've created a union, you can use the drop-down menu on the table in the designer to configure options for the union. Alternatively, you can drag the **New Union** object from the left sidebar into the designer to replace the existing table. This will reveal options to create and configure the union:

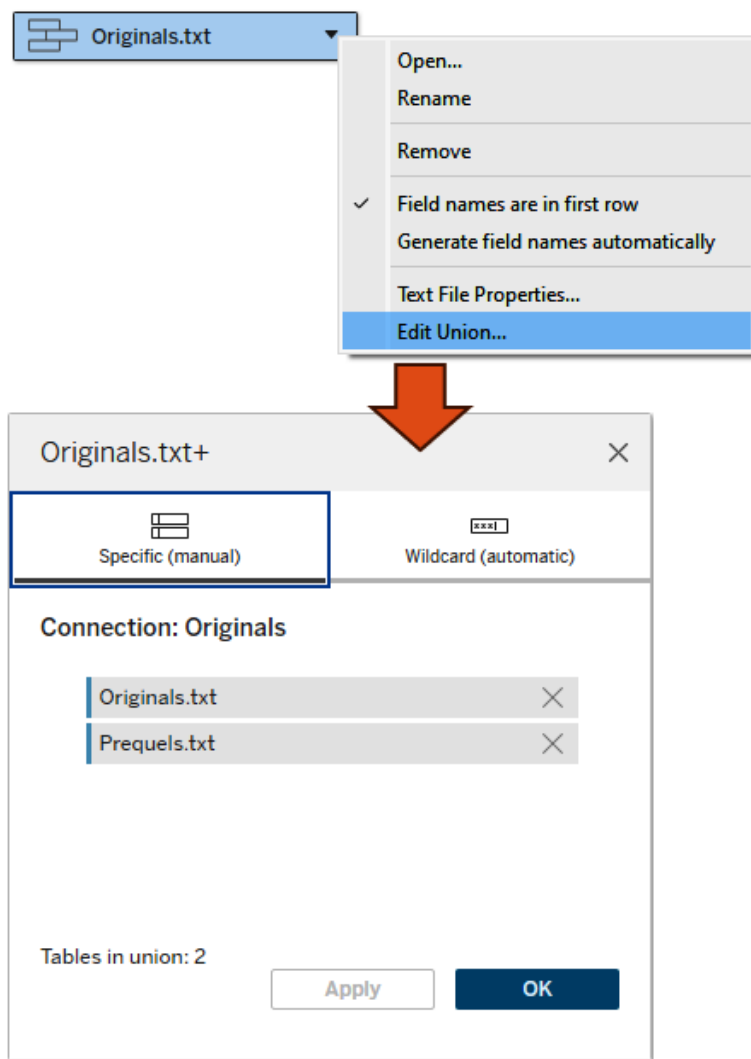


Figure 14.17: You may edit unions with these options

The **Specific (manual)** tab allows you to drag tables into and out of the union. The **Wildcard (automatic)** tab allows you to specify wildcards for filenames and sheets (for Excel and Google Sheets) that will automatically include files and sheets in the union based on a wildcard match. You might take some time to experiment with the Wildcard option now to see if you can get Tableau to union the same files automatically.

**Tip**

Use the **Wildcard (automatic)** feature if you anticipate additional files being added in the future. For example, if you have a directory where data files are dumped on a periodic basis, the wildcard feature will ensure that you don't have to manually edit the connection.

3. Once you have defined the union, you may use the resulting data source to visualize the data. Additionally, a union table may be joined with other tables in the designer window, giving you a lot of flexibility in working with data.

When you create a union, Tableau will include one or more new fields in your data source to help you to identify the file, sheet, and table where the data originated. **Path** will contain the file path (including the filename), **Sheet** will contain the sheet name (for Excel or Google Sheets), and **Table Name** will contain the subtable or text filename. You can use these fields to help you to identify data issues and also to extend your dataset as needed.

For example, if you had a directory of monthly data dump files, named 2025-01.txt, 2025-02.txt, 2025-03.txt, and so on, but no actual date field in the files, you could obtain the date using a calculated field with code such as the following:

```
DATEPARSE('yyyy-MM', [Table Name] )
```

In a union, Tableau will match the columns between tables by name. What happens when columns are not the same between tables or files? If you were to write a query to a database, you might expect a failed result as column names usually have to match exactly. However, Tableau allows you to union together files or tables with mismatched names.

Columns that exist in one file/table but not in others will appear as part of the union table, but values will be NULL in files/tables where the column does not exist. For example, if one of the files contained a column named **Job** instead of **Occupation**, the final union table would contain a column named **Job** and another named **Occupation**, with NULL values where the column did not exist. You can merge the mismatched columns by selecting the columns (holding *Ctrl* as you click each column header) and using the drop-down menu. This will **coalesce** (keep the first non-null of) the values per row of data in a single new column:

Abc Originals.txt+	# Originals.txt+	Abc Originals.txt+	
Occupation	Bank Account Balance	Job	
Farmer	2,000.00	null	
Princess	50,000.00	null	
Smuggler	-20,000.00	null	
Junk Dealer	9,000.00	null	
Face Painter	10,000.00	null	
Sith Lord	-100,000.00	null	
null	600.00	Scavenger	
null	30,000.00	Pilot	
null	0.00	Unemployed	

Figure 14.18: Use Merge Mismatched Fields to combine columns resulting from a union where the field names didn't match (this mismatch is not included in the example data)

You do not have to merge mismatched fields. At times, there will not be corresponding matches in all files or tables and that may be useful for your analysis.

Unions allow you to bring together multiple files or tables with roughly the same structure and **stack** them together so that you end up with all records from all tables/files. With pivots and unions explored, we've covered two of the four basic transformation types. We'll continue with an example of how joins can be used to restructure your data.

Joins

You'll recall that the concept of joins and the types of joins were previously discussed in *Chapter 13, Understanding the Tableau Data Model, Joins, and Blends*. While joins are quite useful in bringing together tables in the same database or even disparate data sources (data contained in different systems and formats), they can be used to solve other data issues, too, such as reshaping data to make it easier to meet your objectives in Tableau.

You can work through the following example in the Chapter 14 workbook, but the server database data source is simulated with a text file (`Hospital Patient.txt`).

Let’s say you have a table in a server database (such as SQL Server or Oracle) that contains one row per hospital patient and includes **Admit Date** and **Discharge Date** as separate columns for each patient:

Patient ID	Patient Name	Admit Date	Discharge Date
1	David	12/1/2024	12/20/2024
2	Solomon	12/3/2024	12/7/2024
3	Asa	12/5/2024	12/22/2024
4	Jehoshaphat	12/5/2024	12/6/2024
5	Joash	12/9/2024	12/16/2024
6	Amaziah	12/10/2024	12/14/2024
7	Uzziah	12/12/2024	12/24/2024
8	Jotham	12/16/2024	12/29/2024
9	Hezekiah	12/18/2024	12/22/2024
10	Josiah	12/22/2024	12/23/2024

Table 14.8: Table of patient visits

While this data structure works well for certain kinds of analyses, you would find it difficult to use if you wanted to visualize the number of patients in the hospital day by day for the month of December.

For one, which date field do you use for the axis? Even if you pivoted the table so that you had all of the dates in one field, you would find that you have gaps in the data. **Sparse data**, that is, data in which records do not exist for certain values, is quite common in certain real-world data sources. Specifically, in this case, you have a single record for each **Admit** or **Discharge** date, but no records for the days in between.

Sometimes, it might be an option to restructure the data at the source, but if the database is locked down, you may not have that option. You could also use Tableau’s ability to fill in gaps in the data (**data densification**) to solve the problem. However, that solution could be complex and, potentially, brittle or difficult to maintain.

An alternative is to use a join to create the rows for all dates. In this case, we'll leverage a cross-database join to bring in another source of data altogether. You might have a date dimension in your data warehouse or you might quickly create an Excel sheet with a list of dates you want to see, like this:

	A
1	Date
2	12/1/2024
3	12/2/2024
4	12/3/2024
5	12/4/2024
6	12/5/2024
27	12/26/2024
28	12/27/2024
29	12/28/2024
30	12/29/2024
31	12/30/2024
32	12/31/2024

Figure 14.19: An Excel file containing every date in December

The Excel file includes a record for each date. Our goal is to **cross-join** (join every row from one table with every row in another) the data between the database table and the Excel table. With this accomplished, you will have a row for every patient for every date.

Note



Joining every record in one dataset with every record in another dataset creates what is called a **Cartesian product**. The resulting dataset will have $N1 * N2$ rows (where $N1$ is the number of rows in the first dataset and $N2$ is the number of rows in the second). Take care in using this approach. It works well with smaller datasets. As you work with larger datasets, the Cartesian product may grow so large that this solution is untenable. In many cases, leveraging the data model and the approaches you learned in the previous chapter for showing all values of a dimension for a logical object might be a far better approach.

You'll often have specific fields in the various tables that will allow you to join the data together. In this case, however, we don't have any keys that define a join. The dates also do not give us a way to join all the data in a way that gives us the structure we want. To achieve the cross-join, we'll use a join calculation. A **join calculation** allows you to write a special calculated field specifically for use in joins.

In this case, we'll select **Create Join Calculation...** for both tables and enter the single, hard-coded value, that is, 1, for both the left and right sides:

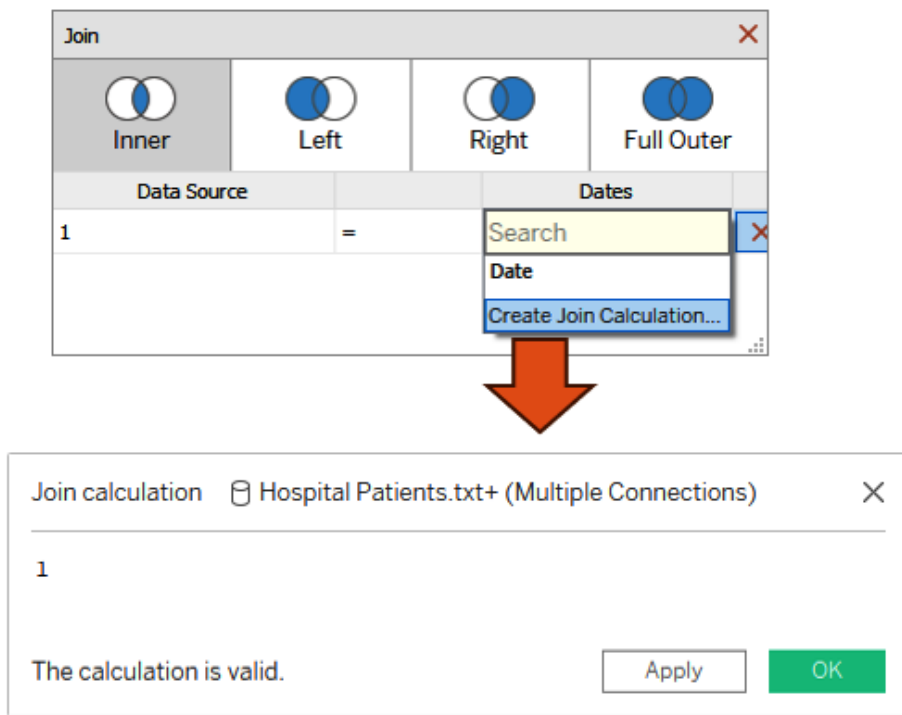


Figure 14.20: As we've seen, the join is created in the physical layer of the data model

Since 1 in every row on the left matches 1 in every row on the right, we get every row matching every row—a true cross-join.

Based on the join calculation, our new cross-joined dataset contains a record for every patient for every date, and we can now create a quick calculation to see whether a patient should be counted as part of the hospital population on any given date. The calculated field, named **Patients in Hospital**, has the following code:

```
IF [Admit Date] <= [Date] AND [Discharge Date] >= [Date]
THEN 1
ELSE 0
END
```


This allows us to easily visualize the flow of patients, and even potentially perform advanced analytics based on averages, trends, and even forecasting:

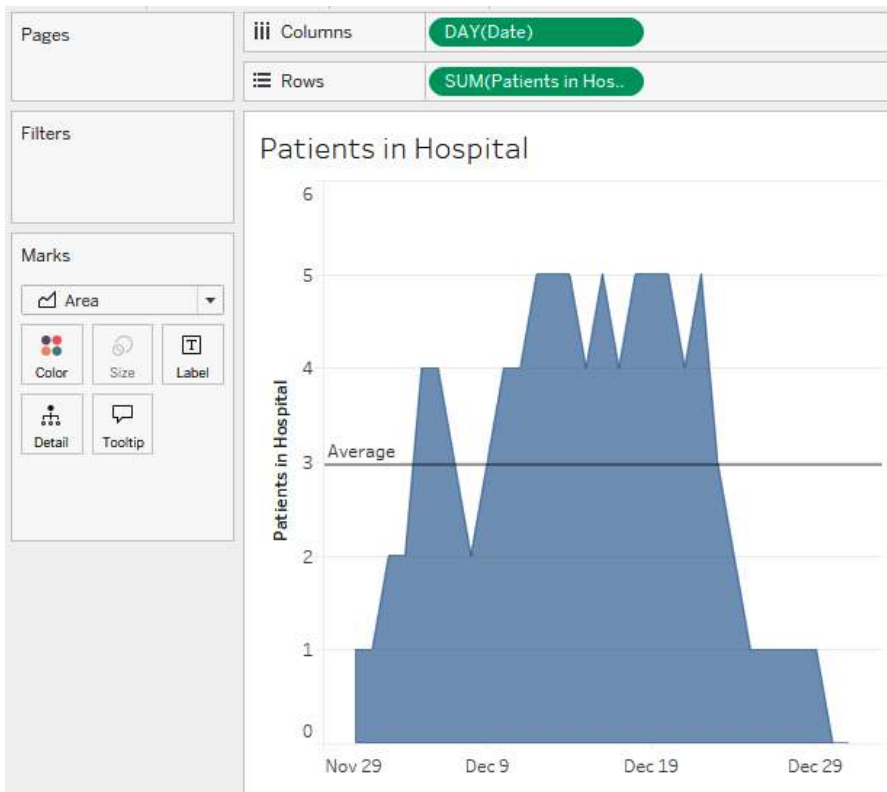


Figure 14.21: A visualization of the daily hospital population, made easy with some data restructuring

Ultimately, for a long-term solution, you might want to consider developing a server-based data source that gives the structure that's needed for the desired analysis. And, as mentioned previously, leveraging Tableau's data model would be a better solution in most cases. However, a join here allowed us to achieve the analysis quickly and come up with a result.

Having considered examples of pivots, unions, and joins, let's turn our focus to an example of the final major transformation type: aggregation.

Aggregation

Remember that the two key points to a good data structure are as follows:

- Having a LOD that is meaningful
- Having measures that match the LOD or that are possibly at higher LODs

Measures at lower levels tend to result in wide data and can make some analysis difficult or even impossible. Measures at higher LODs can, at times, be useful. As long as we are aware of how to handle them correctly, we can avoid some pitfalls.

Consider, for example, the following data (included as `Apartment Rent.xlsx` in the Chapter 14 directory), which gives us a single record each month per apartment:

Apartment	Month of Month	Rent Collected	Square Feet
A	January	0	900
	February	0	900
	March	0	900
	April	0	900
	May	0	900
	June	1,500	900
	July	1,500	900
	August	1,500	900
	September	1,500	900
	October	1,500	900
	November	1,500	900
	December	1,500	900
B	January	1,200	750
	February	1,200	750
	March	1,200	750
	April	1,200	750
	May	1,200	750
	June	1,200	750
	July	0	750
	August	0	750
	September	0	750
	October	0	750
	November	0	750
	December	0	750

Figure 14.22: The Apartment Rent data, which is poorly structured because the Square Feet measure is repeated for every month

The two measures are at different LODs:

- **Rent Collected** matches the LOD of the data where there is a record of how much rent was collected for each apartment for each month.
- **Square Feet**, on the other hand, does not change month to month. Rather, it is at the higher LOD, of **Apartment** only.

Note



Tableau’s data model would make this data very easy to work with if it were contained in two tables at the correct LOD. If it were in a relational database where we could use a custom SQL statement to create a couple of tables at the right LOD, we might consider that approach. In the next chapter, we’ll consider how Tableau Prep Builder could be used to easily solve this problem. For now, work through the details to gain some understanding of how to deal with aggregation issues if you are faced with a similar poor structure (and you’ll gain an immense appreciation for what Tableau Prep Builder and the Tableau data model can do!).

The difference in LODs can be observed when we remove the date from the view and look at everything at the **Apartment** level:

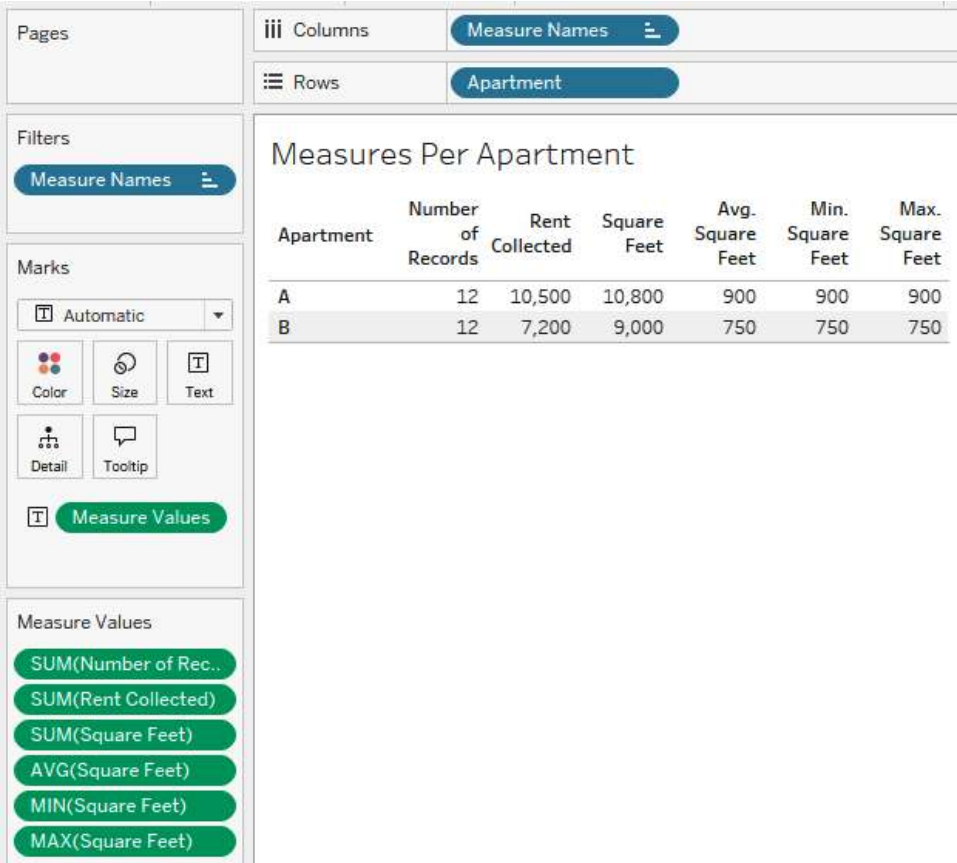


Figure 14.23: An illustration of how various aggregations might be right or wrong depending on the LOD

Notice that the **SUM(Rent Collected)** makes perfect sense. You can add up the rent collected per month and get a meaningful result per apartment. However, you cannot Sum **Square Feet** and get a meaningful result per apartment. Other aggregations, such as average, minimum, and maximum, do give the right results per apartment.

However, imagine that you were asked to come up with the ratio of total rent collected to square feet per apartment. You know it will be an aggregate calculation because you have to sum the rent that's collected prior to dividing. But which of the following is the correct calculation?

- `SUM([Rent Collected])/SUM([Square Feet])`
- `SUM([Rent Collected])/AVG([Square Feet])`
- `SUM([Rent Collected])/MIN([Square Feet])`
- `SUM([Rent Collected])/MAX([Square Feet])`

The first one is obviously wrong. We've already seen that square feet should not be added each month. Any of the final three would be correct if we ensure that **Apartment** continues to define the LOD of the view.

However, once we look at the view that has a different LOD (for example, the total for all apartments or the monthly total for multiple apartments), the calculations don't work. To understand why, consider what happens when we turn on the column grand totals (drag and drop **Totals** from the **Analytics** tab or select **Analysis | Totals | Show Column Grand Totals** from the top menu):

Measures Per Apartment

Apartment	Number of Records	Rent Collected	Square Feet	Avg. Square Feet	Min. Square Feet	Max. Square Feet
A	12	10,500	10,800	900	900	900
B	12	7,200	9,000	750	750	750
Grand Total	24	17,700	19,800	825	750	900

Figure 14.24: None of the aggregations work to give us a grand total

The problem here is that the **Grand Total** line is at the LOD of all apartments (for all months). What we really want as the **Grand Total** of square feet is $900 + 750 = 1,650$. However, here, the sum of square feet is the addition of square feet for all apartments for all months. The average won't work. The minimum finds the value 750 as the smallest measure for all apartments in the data. Likewise, the maximum picks 900 as the single largest value. Therefore, none of the proposed calculations would work at any LOD that does not include the individual apartment.

Tip



You can adjust how subtotals and grand totals are computed by clicking on the individual value and using the drop-down menu to select how the total is computed. Alternatively, right-click on the active measure field and select **Total Using**. You can change how all measures are totaled at once from the menu by selecting **Analysis | Totals | Total All Using**. Using this **two-pass total** technique could result in correct results in the preceding view, but it would not universally solve the problem. For example, if you wanted to show the price per square foot for each month, you'd have the same issue.

Fortunately, Tableau gives us the ability to work with different LODs in a view. Using LOD calculations, which we studied in *Chapter 5, Leveraging Level of Detail Calculations*, we can calculate the square feet per apartment.

Here, we'll use a fixed LOD calculation to keep the LOD fixed at the apartment level. We'll create a calculated field, named *Square Feet per Apartment*, with the following code:

```
{ INCLUDE [Apartment] : MIN([Square Feet]) }
```

The curly braces surround a LOD calculation and the keyword **INCLUDE** indicates that we want to include *Apartment* as part of the LOD for the calculation, even if it is not included in the view LOD. **MIN** is used in the preceding code, but **MAX** or **AVG** could have been used as well because all give the same result per apartment.

As you can see, the calculation returns the correct result in the view at the apartment level and at the grand total level, where Tableau includes **Apartment** to find 900 (the minimum for A) and 750 (the minimum for B) and then sums them to get 1,650:

Apartment	Number of Records	Rent Collected	Square Feet per Apartment
A	12	10,500	900
B	12	7,200	750
Grand Total	24	17,700	1,650

Figure 14.25: An LOD calculation gives us the correct result at all LODs

Now, we can use the LOD calculated field in another calculation to determine the desired results. We'll create a calculated field, named *Rent Collected per Square Foot*, with the following code:

```
SUM([Rent Collected])/SUM([Square Feet per Apartment])
```

When that field is added to the view and formatted to show decimals, the final outcome is correct:

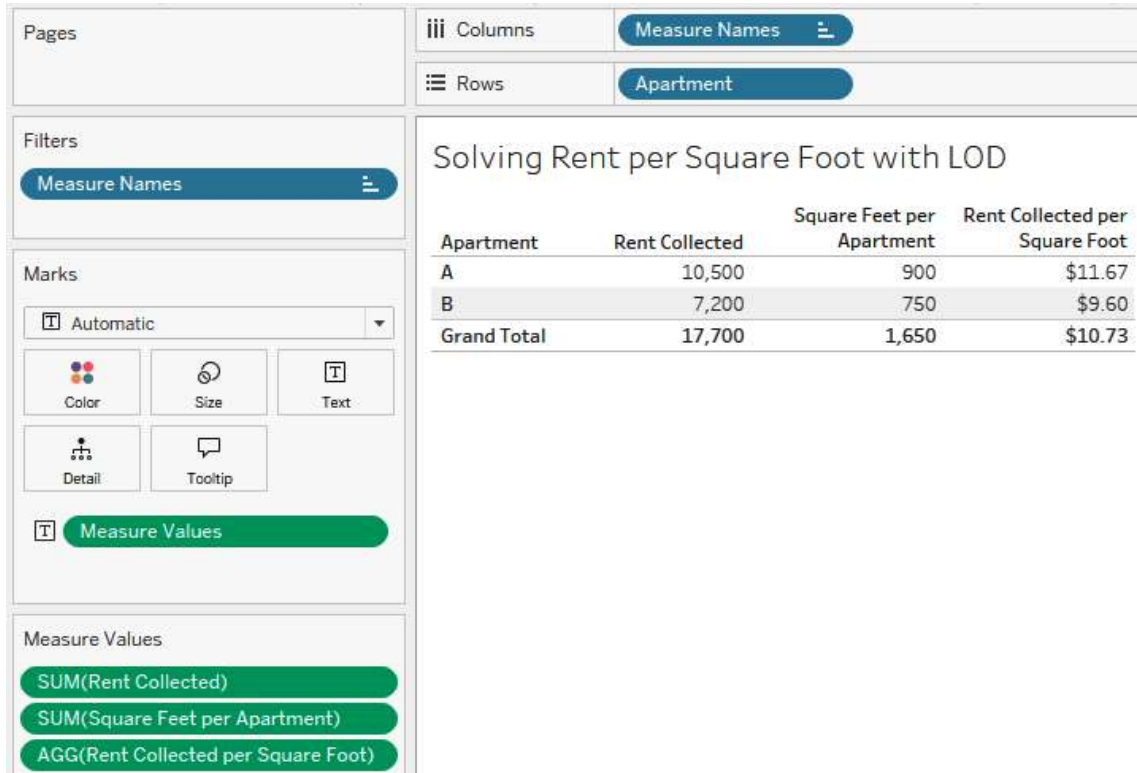


Figure 14.26: The LOD expression gives us the foundation for even more complexity, such as calculating the rent per area

Alternatively, instead of using `INCLUDE`, we could have used a `FIXED` LOD, which is always performed at the LOD of the dimension(s) following the `FIXED` keywords, regardless of what LOD is defined in the view. This would have told Tableau to always calculate the minimum square feet per apartment, regardless of what dimensions define the view LOD. While very useful, be aware that the `FIXED` LOD calculations are calculated for the entire context (either the entire dataset or the subset defined by the **context filters**). Using them without understanding this can yield unexpected results.

For now, we've learned how to handle some aggregation issues; however, in the next chapter, we'll explore how we can truly transform the data with aggregations to make problems like the previous exercise much easier. In the meantime, let's consider some alternative fixes for data problems.

Overview of advanced fixes for data problems

In addition to the techniques that we mentioned earlier in this chapter, there are some additional possibilities to deal with data structure issues.

It is outside the scope of this book to develop these concepts fully. However, with some familiarity of these approaches, you can broaden your ability to deal with challenges as they arise:

- **Custom SQL:** This can be used in the data connection to resolve some data problems. Custom SQL can be used to radically reshape the data that's retrieved from the source. Custom SQL is not an option for all data sources, but it is an option for many relational databases. Consider a custom SQL script that takes the wide table of country populations we mentioned earlier in this chapter and restructures it into a tall table:

```
SELECT [Country Name],[1960] AS Population, 1960 AS Year
FROM Countries
```

```
UNION ALL
```

```
SELECT [Country Name],[1961] AS Population, 1961 AS Year
FROM Countries
```

```
UNION ALL
```

```
SELECT [Country Name],[1962] AS Population, 1962 AS Year
FROM Countries
```

```
...
```

```
...
```

It might be a little tedious to set up, but it will make the data much easier to work with in Tableau! However, many data sources using complex custom SQL will need to be extracted for performance reasons.

- **Table calculations:** Table calculations can be used to solve a number of data challenges, from finding and eliminating duplicate records to working with multiple LODs. Since table calculations can work within partitions at higher LODs, you can use multiple table calculations and aggregate calculations together to mix LODs in a single view. A simple example of this is the **Percent of Total** table calculation, which compares an aggregate calculation at the LOD in the view with a total at a higher LOD.
- **Data blending:** Data blending can be used to solve numerous data structure issues. Because you can define the linking fields that are used, you can control the LOD of the blend.

- **Data scaffolding:** Data scaffolding extends the concept of data blending. With this approach, you construct a scaffold of various dimensional values to use as a primary source and then blend them into one or more secondary sources. In this way, you can control the structure and granularity of the primary source while still being able to leverage data that's contained in the secondary sources.
- **Data model:** Data blending is useful when you need to control the level of the relationship per view. If the relationship is better defined, the data model will give you incredible power to relate tables that are at different LODs and have confidence that aggregations will work correctly.

Summary

Up until this chapter, we looked at data that was, for the most part, well structured and easy to use. In this chapter, we considered what constitutes a good structure and ways to deal with poorly structured data. A good structure consists of data that has a meaningful LOD and that has measures that match that LOD. When measures are spread across multiple columns, we get data that is wide instead of tall.

We also spent some time understanding the basic types of transformation: pivots, unions, joins, and aggregations. Understanding these will be fundamental to solving data structure issues.

You also got some practical experience in applying various techniques to deal with data that has the wrong shape or has measures at the wrong LOD. Tableau gives us the power and flexibility to deal with some of these structural issues, but it is far preferable to fix a data structure at the source.

We'll expand that toolset quite a bit more in the next chapter as we look at Tableau Prep Builder. Tableau Prep gives you incredible power and sophistication, allowing you to bring together data from various sources, clean it, and structure it in any way you like!

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search for this book by name.



Note: Keep your purchase invoice ready before you start.

15

Taming Data with Tableau Prep

We considered some options for structuring data in Tableau Desktop in the previous chapter. Many of the concepts around well-structured data will apply here as we now turn our attention to another product from Tableau: **Tableau Prep**. Tableau Prep extends the Tableau platform with robust options for cleaning and structuring data for analysis in Tableau. In the same way that Tableau Desktop provides a hands-on, visual experience for visualizing and analyzing data, Tableau Prep provides a hands-on, visual experience for cleaning and shaping data.

In this chapter, we'll use the term Tableau Prep broadly to speak of the entire platform that Tableau has developed for data prep and sometimes as shorthand for Tableau Prep Builder, the client application that's used to connect to data, create data flows, and define output. Where needed for clarity, we'll use these specific names:

- **Tableau Prep Builder:** The client application that's used to design data flows, run them locally, and publish them.
- **Tableau Prep Conductor:** An add-on to Tableau Server that allows the scheduling and automation of published data flows. We won't focus much on Tableau Prep Conductor in this chapter as the primary aim is to understand how to build flows to transform and clean data. The interface for scheduling those flows with Conductor is relatively straightforward.

Tableau Prep is designed fundamentally for cleaning and shaping data. We'll cover a lot of ground in this chapter, but our goal is not to cover every possible feature—and indeed, we won't. Instead, we will seek to understand the underlying paradigm and flow of thought that will enable you to tackle a multitude of data challenges in Tableau Prep.

In this chapter, we'll work through a couple of practical examples as we explore the paradigm of Tableau Prep, understand the fundamental transformations, and see many of the features and functions of Tableau Prep.

We'll cover quite a few topics in this chapter, including the following:

- Getting ready to explore Tableau Prep
- Understanding the Tableau Prep Builder interface
- Flowing with the fundamental paradigm
- Using parameters in Tableau Prep Builder
- Options for automating flows

Let's start by understanding how to get started with Tableau Prep.

Getting ready to explore Tableau Prep Builder

Tableau Prep Builder is available for Windows and Mac. If you do not currently have Tableau Prep Builder installed on your machine, please take a moment to download the application from <https://www.tableau.com/products/prep/download>. Licenses for Tableau Prep Builder are included with Tableau Creator licensing. If you do not currently have a license, you may trial the application for 14 days. Please speak with your Tableau representative to confirm licensing and trial periods.

The examples in this chapter use files located in the \Learning Tableau\Chapter 15 directory. Specific instructions will guide you on when and how to use the various files.

Once you've downloaded and installed Tableau Prep Builder, you will be able to launch the application. Once you do, you'll find a welcome screen that we'll explain when we cover the interface in the next section.

Understanding the Tableau Prep Builder interface

You'll find a lot of similarities in the interfaces of Tableau Prep Builder and Tableau Desktop. The home screen of Tableau Prep Builder will look similar to this:

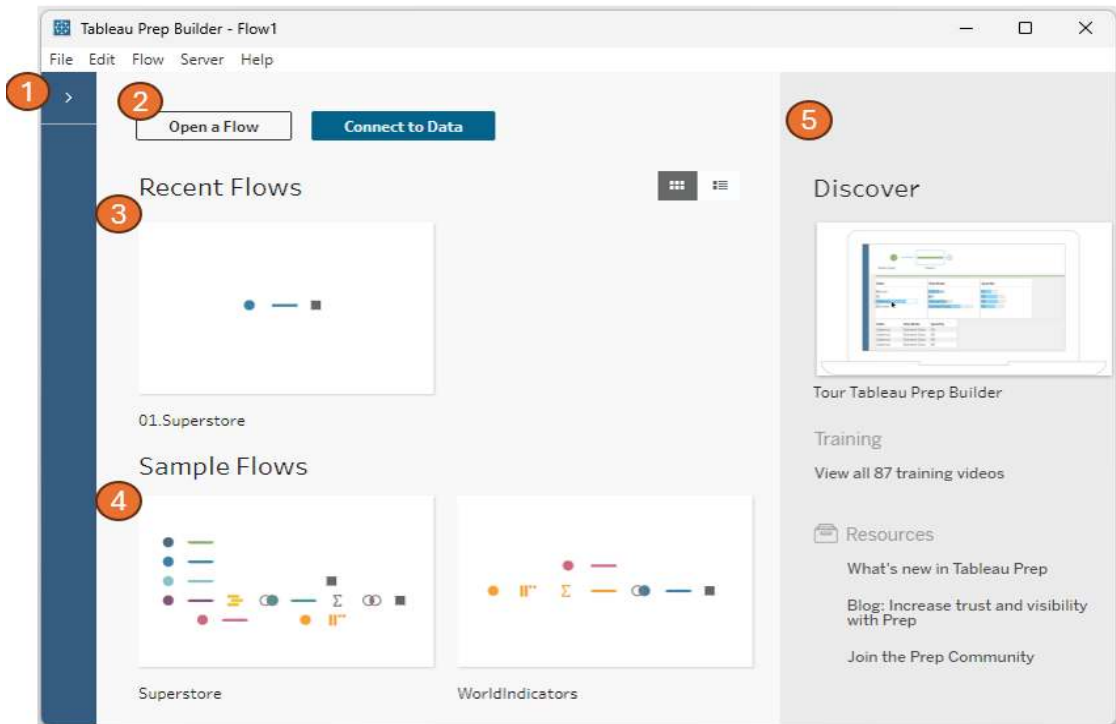


Figure 15.1: The Tableau Prep Builder welcome screen with numbering to identify key components of the UI

The following components have been numbered in *Figure 15.1*:

1. The menu includes options for opening files, editing and running flows, signing into Tableau Server, and various **Help** functions.
2. We'll define a flow in the next section. For now, think of a flow in terms of Tableau Prep's equivalent of a Tableau Desktop workbook.

Also notice the **Connections** pane to the left, immediately beneath the **File** menu. It is collapsed initially but will contain a list of data connections as you create them.

3. The two large buttons at the top give you the option to **Open a Flow**, which opens an existing Tableau Prep flow file, or **Connect to Data**, to start a new flow with an initial data connection.
4. **Recent Flows** shows the Tableau Prep data flows that you have recently saved. You may click on one of these to open the flow and edit or run it. A toggle button on the right allows you to switch between thumbnails and a list.
5. **Sample Flows** allows you to open some prebuilt examples.
6. The **Discover** pane gives you options for training and resources as you learn more about Tableau Prep. A notification to upgrade will also appear if there is a newer version available.

Once you have opened or started a new flow, the home screen will be replaced with a new interface, which will facilitate the designing and running of flows:

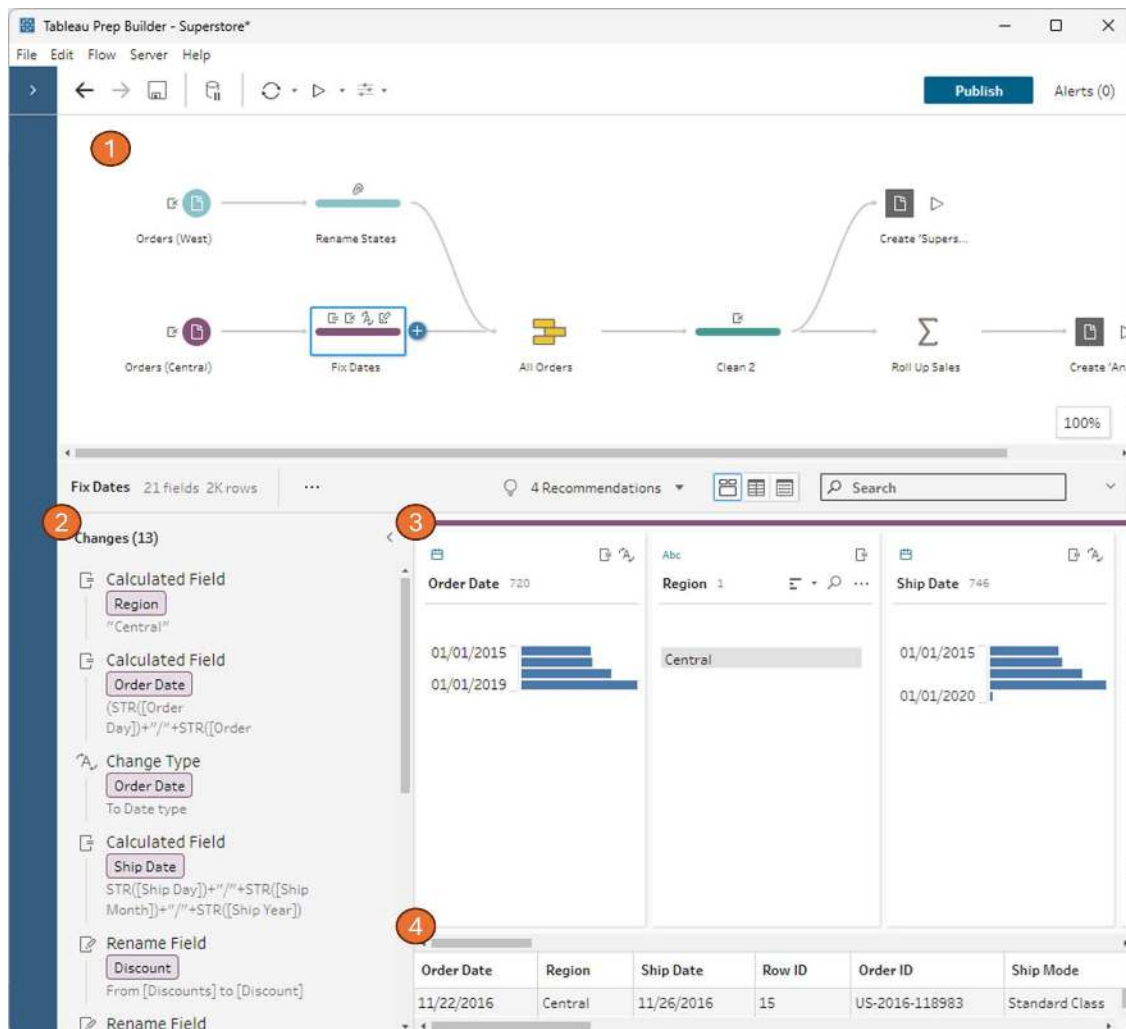


Figure 15.2: When designing a flow, you'll find an interface like this one. The major components are numbered and described as follows

This interface consists of the following, which are numbered in the preceding screenshot:

1. The flow pane, where you will logically build the flow of data with steps that will do anything from cleaning to calculation, to transformation and reshaping. Selecting any single step will reveal the interface on the bottom half of the screen. This interface will vary slightly, depending on the type of step you have selected.
2. The settings, or changes, pane lists settings for the step and also a list of all changes that are made in the step, from calculations to renaming or removing fields, to changing data types or grouping values. You can click on individual changes to edit them or see how they alter the data.

3. The profile pane gives you a profile of each field in the data as it exists for the selected step. You can see the type and distribution of values for each field. Clicking on a field will highlight the lineage in the flow pane and clicking one or more values of a field will highlight the related values of other fields.
4. The data grid shows individual records of data as they exist in that step. Selecting a change in the changes grid will show the data based on changes up to and including the selected change. Selecting a value in the profile pane will filter the data grid to only show records containing that value. For example, selecting the first row of the Order Date field in the profile pane will filter the data grid to show only records represented by that bar. This allows you to explore the data but doesn't alter the data until you perform a specific action that results in a change.

You will also notice the toolbar above the preview pane that allows you to undo or redo actions, refresh data, or run the flow. Additionally, there will be other options or controls that appear based on the type of step or field that's selected. We'll consider those details as we dive into the paradigm of Tableau Prep, and a practical example later in the chapter.

Flowing with the fundamental paradigm

The overall paradigm of Tableau Prep is a hands-on, visual experience of discovering, cleaning, and shaping data through a flow. A **flow** (sometimes also called a **data flow**) is a logical series of steps and changes that are applied to data from input(s) to output(s).

Here is an example of what a flow looks like in the flow pane of Tableau Prep:

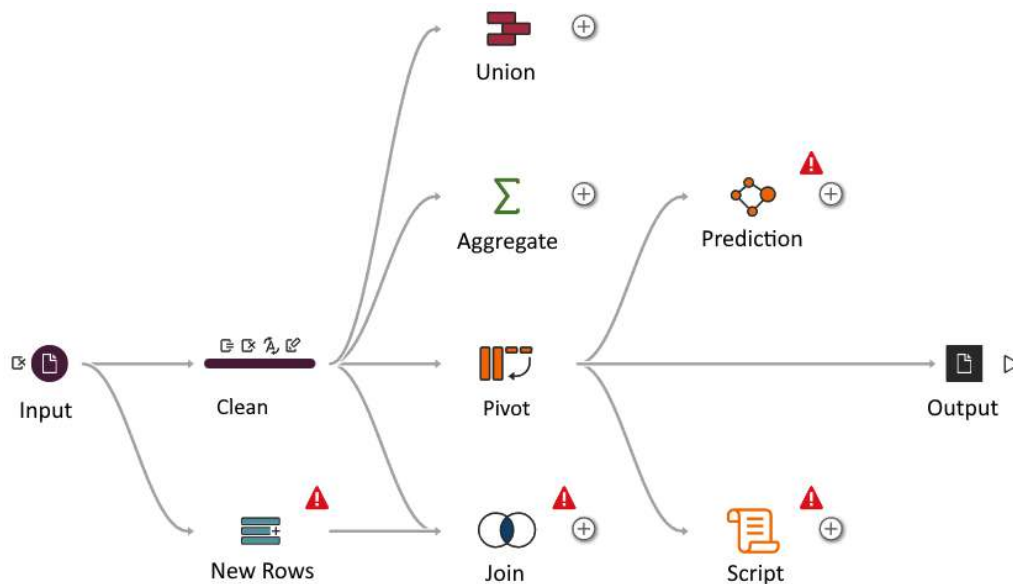


Figure 15.3: An example flow in Tableau Prep

The individual components of the flow are called **steps**, which are connected by lines that indicate the logical flow of data from left to right. The lines are sometimes called connectors or branches of

the flow. Notice that the **Pivot** step here has one line coming in from the left and three branches extending to the right. Any step can have multiple output branches that represent logical copies of the data at that point in the flow.

Note



One important thing to notice is that four of the step types represent the four major transformations of data we discussed in *Chapter 14, Structuring Messy Data to Work Well in Tableau*. The step types of **Pivot**, **Union**, **Join**, and **Aggregate** exactly match those transformations. You may wish to refresh your memory on the basic transformations in the previous chapter.

The **Clean** step allows various other operations involved in cleaning and calculating. The **Script** step and **Prediction** step allow you to extend your flow with advanced analytics or functions available in R or Python or by leveraging Einstein Discovery. We'll cover extending Tableau Desktop with scripts in *Chapter 16, Extending Tableau: Extensions and Scripts*. We won't spend much time on them in this chapter, but you'll find that integrating them in Tableau Prep is very similar to using them in Tableau Desktop.

Note



You'll notice that the **New Rows**, **Join**, **Script**, and **Prediction** steps, and a couple of others in the example shown in *Figure 15.3*, have an error icon, indicating that something has not been configured correctly in the flow. Hovering over the icon gives a tooltip description of the error. Often, selecting a step with an error icon may reveal details about the error in the changes pane or elsewhere in the configuration steps.

As we work through an example of a flow throughout this chapter, we'll examine each type of step more closely. For now, consider these preliminary definitions of the primary steps in Tableau Prep:

- **Input:** An input step starts the flow with data from a source such as a file, table, view, or custom SQL. It gives options for defining file delimiters, unions of multiple tables or files, and how much data to sample (for larger record sets).
- **Clean:** A clean step allows you to perform a wide variety of functions on the data, including calculations, filtering, adjusting data types, removing and merging fields, grouping and cleaning, and much more.
- **New Rows:** This step adds new rows to your dataset. This is useful if you want to fill in missing values. For example, you might use this if you have an initial dataset that has a date field, but the data is sparse and you want to ensure that there is at least one row per date.
- **Aggregate:** An aggregate step allows you to aggregate values (for example, MIN, MAX, SUM, AVG) at a level of detail you specify.
- **Join:** A join step allows you to bring together two branches of the flow and match data row by row based on the fields you select and the type of join.
- **Union:** A union step allows you to bring together two or more branches representing sets of data to be unioned together. You will have options for merging or removing mismatched fields.

- **Pivot:** A pivot step allows you to transform columns of data into rows or rows of data into columns. You'll have options to select the type of pivot as well as the fields themselves. The term *transpose* is sometimes also used to describe this operation.
- **Script:** A script step allows you to flow your data through an R or Python script, allowing you to use various functions to achieve complex analytics or transformations.
- **Prediction:** A prediction step allows you to use Salesforce Einstein Discovery to feed your data through a predictive model and receive predictions based on the data. This will require a Salesforce account and a previously designed predictive model.
- **Output:** The output step defines the ultimate destination for the cleaned and transformed data. This could be a text file (.csv), extract (.hyper), Excel (.xlsx), a published extracted data source to Tableau Server, a table in a database or cloud storage, or Salesforce CRM or Data Cloud. You'll have options to select and configure the type of output.

Right-clicking a step or connector reveals various options. You may also drag and drop steps onto other steps to reveal options such as joining or unioning the steps together. If you want to replace an early part of the flow to swap out an input step, you can right-click the connector, select **Remove**, and then drag the new input step over the desired next step in the flow to add it as the new input. Any changes you make in the middle of a flow may have an impact on later steps in the flow. If you change or remove a dependency, Tableau Prep will indicate broken steps with a warning or error icon on the step.

As you add and connect various steps to achieve your goals, you'll be defining a flow of data and a flow of logical steps and actions.

Note



In addition to using the term *flow* to refer to the steps and connections that define the logical flow and transformation of the data, we'll also use the term to refer to the file that Tableau Prep uses to store the definition of the steps and changes of a flow. Tableau Prep flow files have the .tfl (unpackaged flow) or .tflx (packaged flow) extension.

The paradigm of Tableau Prep goes far beyond the features and capabilities of any single step. As you build and modify flows, you'll receive instant feedback so that you can see the impact of each step and change. This makes it relatively easy (and fun!) to iteratively discover your data and make the necessary changes.

Note



When you are building flows, adding steps, making changes, and interacting with data, you are in **design mode**. Tableau Prep uses a combination of the Hyper engine's cache, along with direct queries of the database, to provide near-instant feedback as you make changes. When you run a flow, you are using **batch mode** or **execution mode**. Tableau Prep will run optimized queries and operations that may be slightly different than the queries that are run in design mode.

For the remainder of this chapter, we'll consider a specific example using some sample data files. This will aid in our discussion of the Tableau Prep paradigm and highlight some important features and considerations. The example will unfold organically, which will allow us to see how Tableau Prep gives you incredible flexibility to address data challenges as they arise and make changes as you discover new aspects of your data.

We'll put you in the role of an analyst at your organization, with the task of analyzing employee air travel. This will include ticket prices, airlines, and even a bit of geospatial analysis of the trips themselves. The data needs to be consolidated from multiple systems and will require some cleaning and shaping to enable the analysis.

To follow along, open Tableau Prep Builder, which will start on the home screen (there is no starter flow for this chapter). The sample data is in the Chapter 15 directory, along with the Complete flow if you want to check your work. The Complete (clean) flow contains a sample of how a flow might be self-documented—it will not match screenshots precisely.

Note



When you open the Complete flow file, you'll likely receive errors and warnings that input paths and output paths are not valid. This is expected because your machine will almost certainly have a different drive and directory structure than the one on which the examples were prepared. You'll also run into this behavior when you share flow files with others. To resolve the issues, simply work through the connections in the **Connections** pane (expanded in *Figure 15.4*) on the left to reconnect to the files and set output steps to the appropriate directories on your machine.

We'll start by connecting to some data!

Connecting to data

Connecting to data in Tableau Prep is very similar to connecting to data in Tableau Desktop. From the home screen, you may click either **Connect to Data** or the + button on the expanded **Connections** pane:

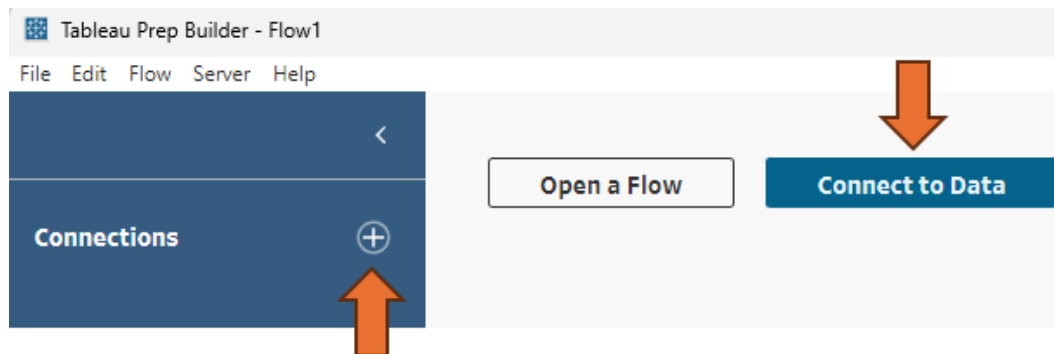


Figure 15.4: You can make a new data connection by clicking the + button or the Connect to Data button

Either UI element will bring up a list of data source types to select.



Note

As with Tableau Desktop, for file-based data sources, you may drag the file from **Windows Explorer** or **Finder** onto the Tableau Prep window to quickly create a connection.

Tableau Prep supports connections to dozens of file types and databases, and the list continues to grow. You'll recognize many of the same types of connection possibilities that exist in Tableau Desktop; however, Tableau Prep does not support all of the exact same connection types that are available in Tableau Desktop.

You may create as many connections as you like and the **Connections** pane will list each connection separately with any associated files, tables, views, stored procedures, or other options that are applicable to that data source. You will be able to use any combination of data sources in the flow.

For now, let's start our example with the following steps:

1. Starting from the window shown in *Figure 15.4*, click **Connect to Data**.
2. From the expanded list of possible connections that appears, select **Microsoft Excel**.
3. You'll see a main table called **Employee Flights** and a sub-table named **Employee Flights Table 1**. Drag the **Employee Flights** table to the flow canvas. An input step will be created, giving you a preview of the data and other options. The input preview pane will initially look like this:

💡 3 Recommendations

Data Preview

Field List

🔍 Search

11 fields

#	Abc		
Employee Airline T...	F2	F3	F4
null	null	null	null
null	null	null	07/08/1905
null	null	null	07/09/1905
null	null	null	07/10/1905
null	null	null	07/11/1905
null	Passenger Email	null	null
4,144	ekiefer@vizpainter.com	01/10/2019	02/06/2019
9,102	abarton@vizpainter.c...	09/10/2017	11/12/2017

Figure 15.5: The initial preview pane for the Excel data

At the top, you'll notice buttons that allow you to switch between **Data Preview** and **Field List**. The data is shown by default. Switching to **Field List** displays a grid of fields and options for

those fields. You'll notice that many of the fields in the **Employee Flights** table are named F2, F3, F4, and so on.

This is due to the format of the Excel file, which has merged cells and a summary sub-table. Continue the exercise with the following steps:

4. Check the **Use Data Interpreter** option in the **Connections** pane:

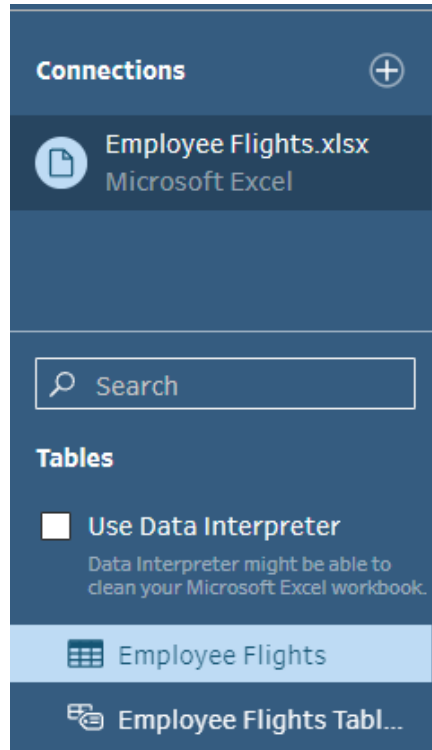


Figure 15.6: The Connections pane has an option to use Data Interpreter

Tableau Prep will correctly parse the file, as shown here:

Preview Employee Flights 11 fields					
	#	#	Abc		
1	Source Row Nu... ⓘ	Row ID	Passenger Email	Purchase Date	Travel Date
2	±	4,144	ekiefer@vizpainter.com	01/10/2019	02/06/2019
3	±	9,102	abarton@vizpainter.c...	09/10/2017	11/12/2017
4	±	2,194	achung@vizpainter.com	10/22/2018	12/17/2018
5	±	6,695	svernon@vizpainter.c...	06/07/2018	08/30/2018

Figure 15.7: Data Interpreter parses the file to fix common issues such as merged cells, empty headers, and sub-total lines

The **Source Row Number** field is added to flat file inputs, but then a **change** is automatically made to the input step to remove the field. You may remove the change and thus restore the field:



Figure 15.8: Switch to the Changes tab and remove the change to restore the field

Switching to **Field List** of an input step (as shown in Figure 15.5) causes Tableau Prep to display a grid of fields in the data. You may use the grid to uncheck any fields you do not wish to include, edit the **Type** of data by clicking the associated symbol (for example, change a string to a date), and edit **Field Name** itself by double-clicking the field name value.

Note



If Tableau Prep Builder detects that the data source contains a large number of records, it may turn on **Data Sampling**. Data Sampling uses a smaller subset of records for giving rapid feedback and profiling in design mode. However, it will use the full set of data when you run the entire flow in batch mode. You can control the data sampling options by clicking **Data Sample** on the input pane. While you can set the sample size for the source, subsequent steps, such as joins, that result in large numbers of records may turn on sampling that cannot be disabled. You'll receive an indicator of **Data Sampling** if it occurs anywhere in the flow.

Now, we'll continue to explore the data and fix some issues along the way. Continue with these steps:

1. Click the **View and clean data** zone to the right of the **Employee Flights** input step. This will extend the flow by adding a clean step called **Clean 1**:

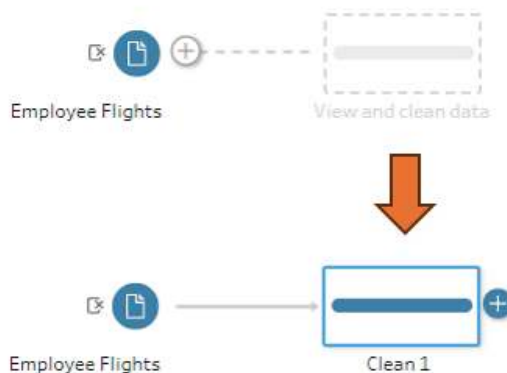


Figure 15.9: Adding an initial Clean Step

**Tip**

Notice the + button that appears to the right of a step. Clicking that button allows you to add a new step (with the type of your choosing) to the flow. Hovering over a connector between steps will cause a + button to appear, which allows you to insert a step into the flow.

2. With the **Clean 1** step selected, take a moment to explore the data using the profile pane. Observe how selecting individual values for fields in the profile pane highlights portions of related values for other fields. This can give you great insight into your data, such as seeing the different price ranges based on Ticket Type:

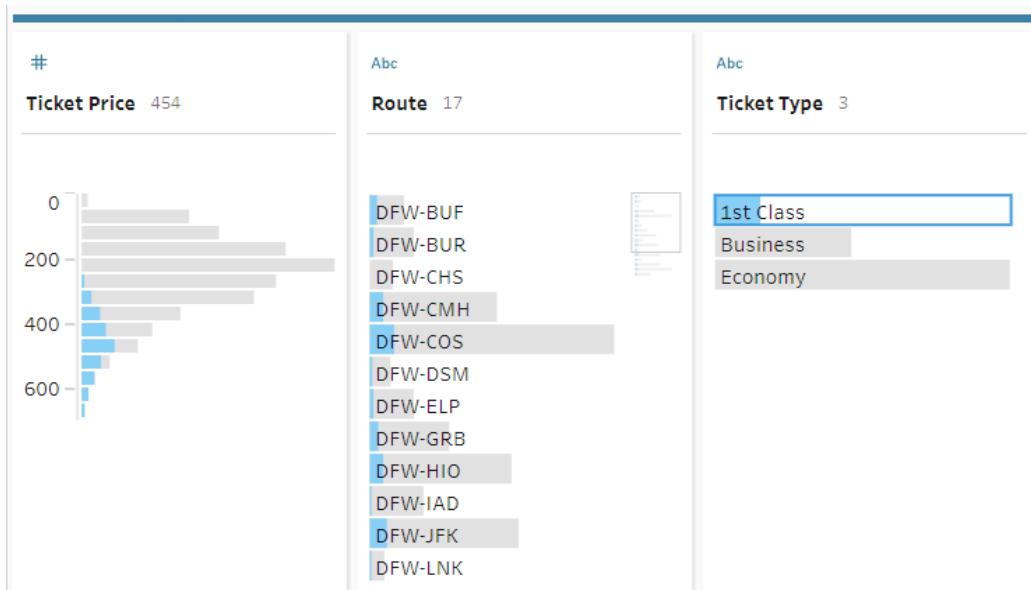


Figure 15.10: Selecting a value for a field in the profile pane highlights which values (and what proportion of those values) relate to the selected value

Highlighting the bar segments across fields in the profile pane, which results from selecting a field value, is called **brushing** or **proportional brushing**. You can also take action on selected values via the toolbar at the top of the profile pane or by right-clicking a field value. These actions include filtering values, editing values, or replacing them with NULL. However, before making any changes or cleaning any of the data, let's connect to some additional data.

It turns out that most of the airline ticket booking data is in one database that's represented by the Excel file, but another airline's booking data is stored in files that are periodically added to a directory. These files are in the \Learning Tableau\Chapter 15\ directory. The files are named with the convention Southwest YYYY.csv (where YYYY represents the year of travel).

We'll connect to all the existing files and ensure that we are prepared for additional future files:

1. Click the + icon on the **Connections** pane to add a new connection to a text file:

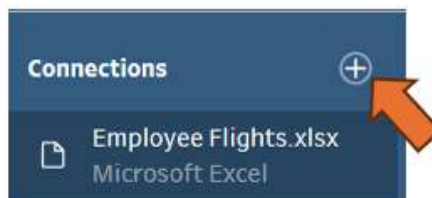


Figure 15.11: Adding a new connection

 **Quick tip:** Need to see a high-resolution version of this image? Open this book in the next-gen Packt Reader or view it in the PDF/ePub copy.

 **The next-gen Packt Reader and a free PDF/ePub copy** of this book are included with your purchase. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



2. Navigate to the \Learning Tableau\Chapter 15\ directory and select any of the Southwest YYYY.csv files to start the connection. Looking at the **Input** settings, you should see that Tableau Prep correctly identifies the field separators, field names, and types:

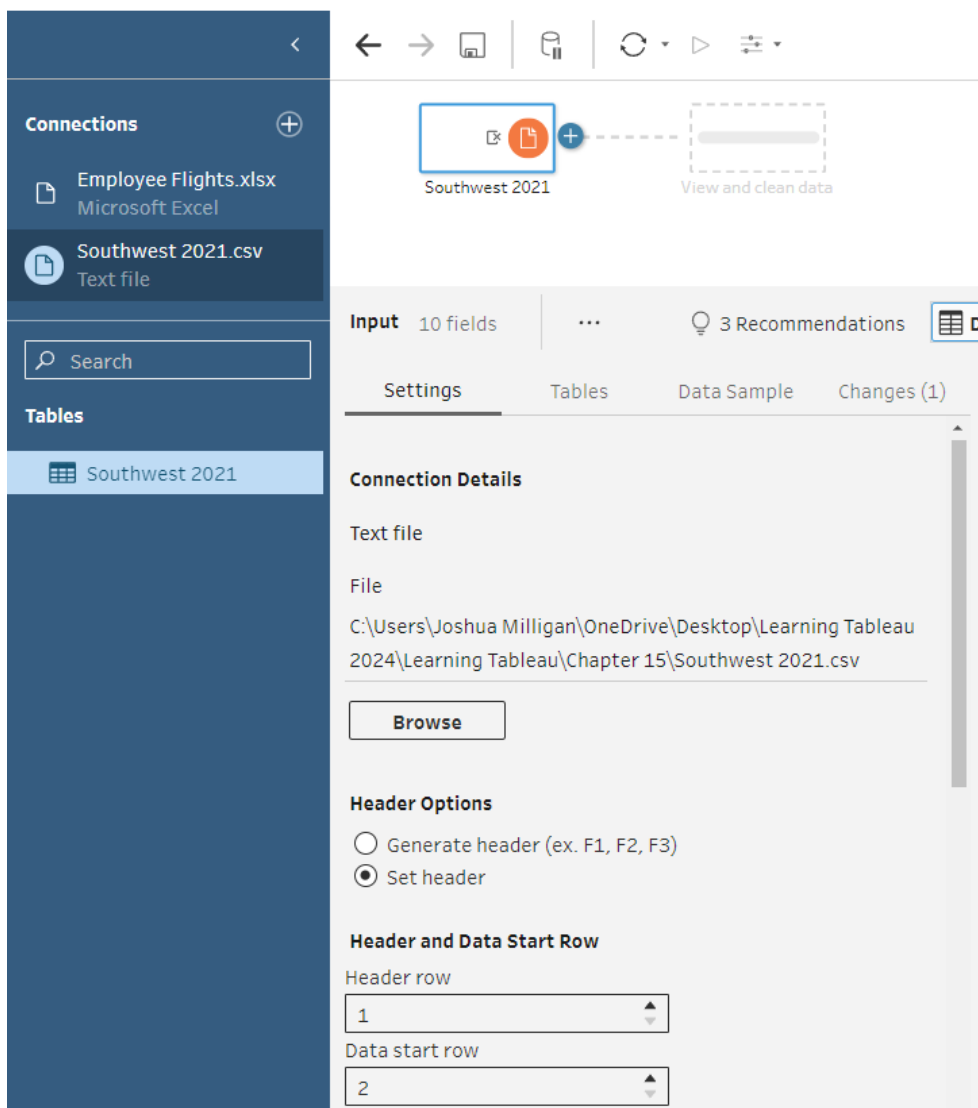


Figure 15.12: A text file includes options for headers, field separators, text qualifiers, character sets, header and data positions, and more. Notice also tabs such as Tables and Data Sample, giving other options for the text input

3. In the **Input** pane, select the **Tables** tab and switch from **Single table** to **Union multiple tables**. Add a file filter so that the filename matches **Southwest***. This tells Tableau Prep to union all of the text files in the directory that begin with **Southwest** together:

Input 10 fields Filter Values...

Settings **Tables** Data Sample Changes (1)

▼ **Source**

☐ Single table
☒ Union multiple tables

Search in
 Chapter 15 ▼

☐ Include subfolders

▼ **File Filters (1)** ⓘ

Include files matching all conditions

File name ⓘ
 Matches ▼

Southwest*

Included Tables (4)

File	Date Modified	Date Created	Size
Southwest 2021.csv	10/25/2024	10/25/2024	62,190 Bytes
Southwest 2022.csv	10/25/2024	10/25/2024	67,627 Bytes
Southwest 2023.csv	10/25/2024	10/25/2024	90,673 Bytes
Southwest 2024.csv	10/25/2024	10/25/2024	59,788 Bytes

Apply

Figure 15.13: All Southwest files will be included in this input

4. Click the **Apply** button (shown in Figure 15.13).
5. Add a new clean step after the **Southwest** input step in the flow pane. This step will be named **Clean 2** by default. Once again, explore the data, but don't take any action yet. You may notice a new field in the **Clean 2** step called **File Paths**, which indicates which file in the union is associated with each record.

With our input steps defined, let's move on to consider how to clean up some of the data to get it ready for analysis.

Cleaning the data

The process of building out the flow is quite iterative, and you'll often make discoveries about the data that will aid you in cleaning and transforming it. We'll break this example into sections for the sake of reference, but don't let this detract from the idea that building a flow should be a flow of thought. The example is meant to be seamless!

We'll look at quite a few possibilities for prepping the data in this section, including merging and grouping. Let's start with seeing how to union together branches in the flow.

Unioning and merging mismatched fields, and removing unnecessary fields

We know that we want to bring together the booking data for all the airlines, so we'll union together the two paths in the flow:

1. Drag the **Clean 2** step onto the **Clean 1** step and drop it onto the **Union** box that appears. This will create a new **Union** step with input connections from both of the clean steps:

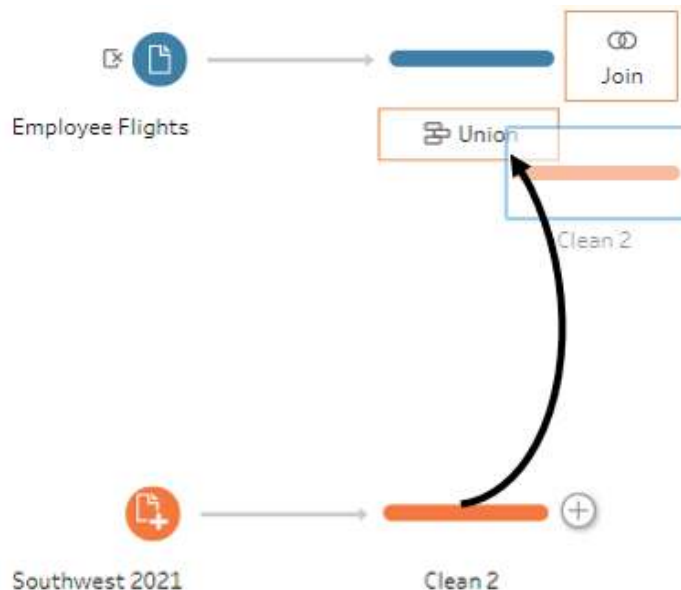


Figure 15.14: Dragging one step onto another in the flow reveals options for bringing the datasets together in the flow. Here, for example, there are options for creating a Union or Join

- The **Union** pane that shows up when the **Union** step is selected will show you the mismatched fields, indicate the associated input, and give you options for removing or merging the fields. For example, Fare Type and Ticket Type are named differently between the Excel file and the text files, but in fact represent the same data. Hold down the **Ctrl** key and select both fields. Then, select **Merge Fields** from the toolbar at the top of the pane or from the right-click menu:

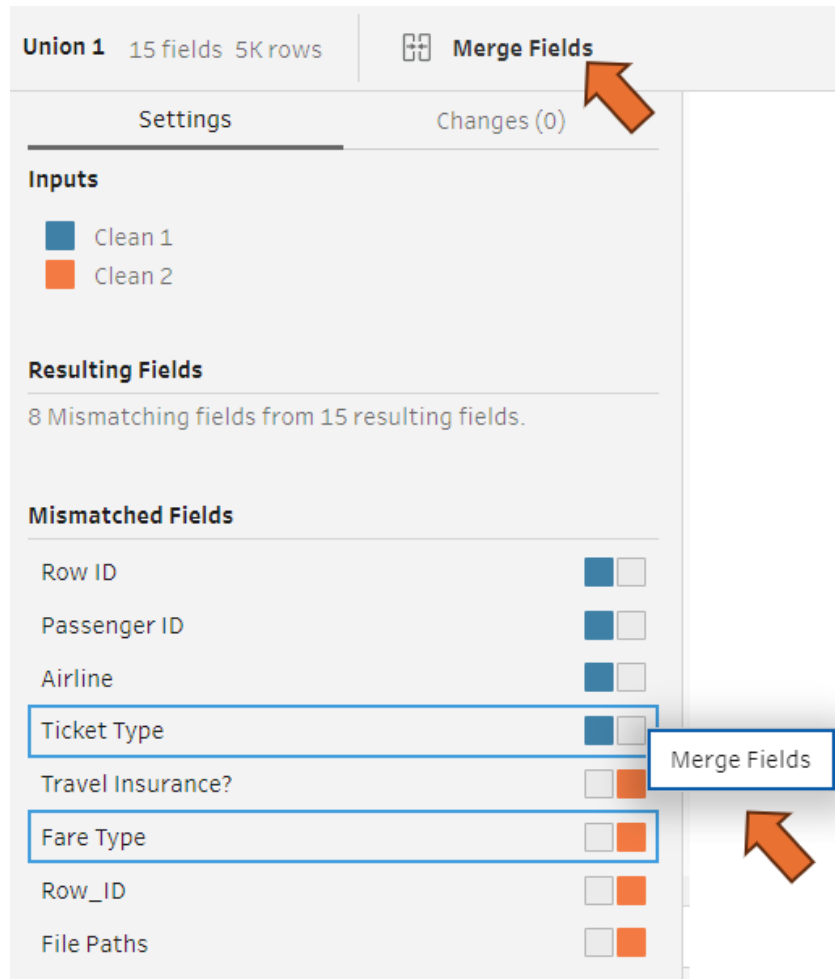


Figure 15.15: When you select a single field, Tableau Prep will highlight fields that are potentially the same between tables. Selecting both reveals the Merge Fields option at the top and a Merge Fields context menu option on right-click

- Also, merge Row ID and Row_ID.
- File Paths** applies only to the Southwest files, which were unioned together in the input step. While this auto-generated field can be very useful at times, it does not add anything to the data in this example. Select the field, then click the ellipsis menu button and select **Remove**.

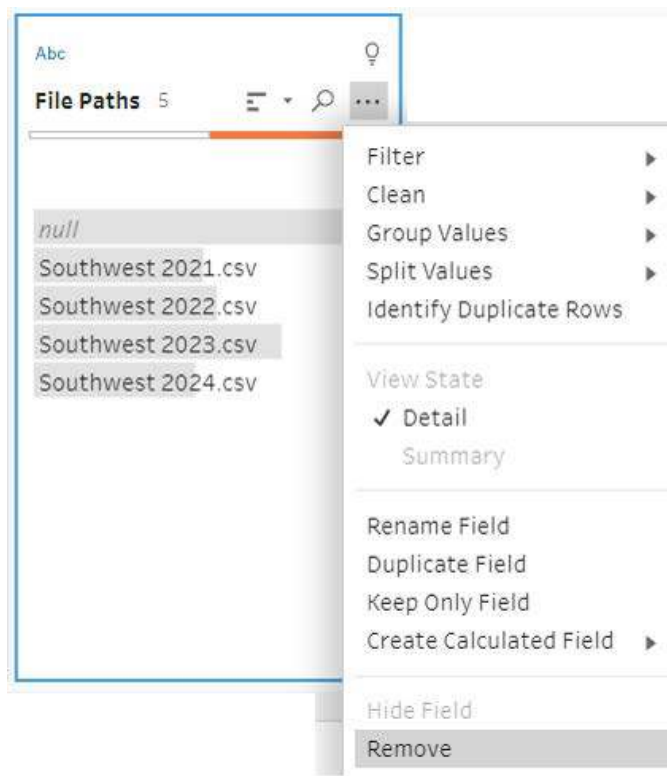


Figure 15.16: Removing an unneeded field

5. Similarly, Travel Insurance? and Passenger ID apply to only one of the inputs and will be of little use in our analysis. Remove those fields as well.
6. The single remaining mismatched field, Airline, is useful. Leave it for now, click the + icon on the **Union 1** step in the flow pane, and extend the flow by selecting **Clean Step**. At this point, your flow should look like this:

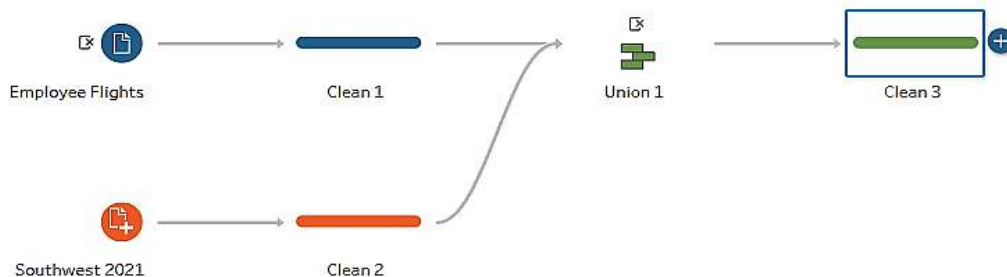


Figure 15.17: Your flow should look similar to this. You may notice variations in the exact location of steps or color (you can change a step's color by right-clicking the step and selecting *Edit Color*)

Note



There is an icon above the **Union 1** step in the flow, indicating changes that were made within this step. In this case, the changes are the removal of several of the fields. Each step with changes will have similar icons, which will reveal tooltip details when you hover over them and also allow you to interact with the changes. You can see a complete list of changes, edit them, reorder them, and remove them by clicking the step and opening the **Changes** pane. Depending on the step type, this is available by either expanding it or selecting the **Changes** tab.

Next, we'll continue building the flow and consider some options for grouping and cleaning.

Grouping and cleaning

Now, we'll spend some time cleaning up the data that came from both input sources. With the **Clean 3** step selected, use the **Profile** pane to examine the data and continue our flow. The first two fields indicate some issues that need to be addressed:

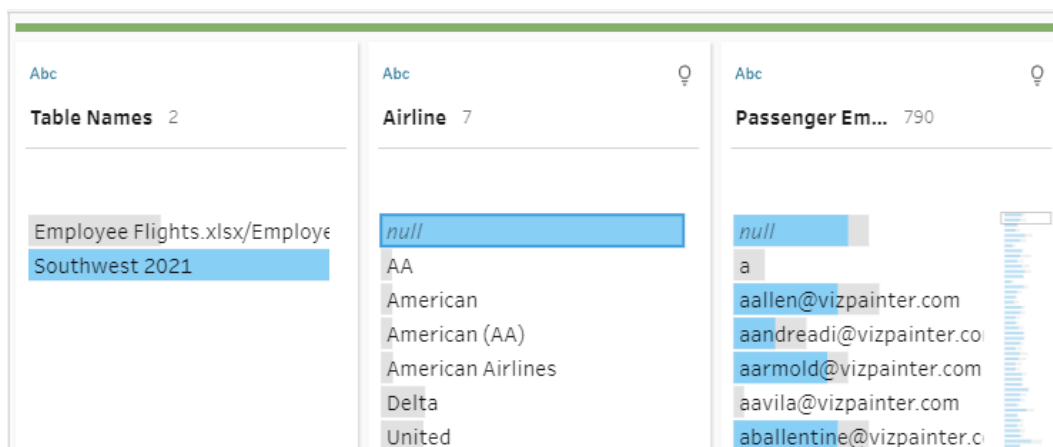


Figure 15.18: Every null value in the **Airline** field comes from the Southwest files. Fortunately, in this case, the source of the data indicates the airline

The **Table Names** field was generated by Tableau Prep as part of **Union 1** to indicate the source of the records. The **Airline** field came only from the Excel files. You can confirm this by selecting it in the profile pane and observing the highlighted path of the field in the flow pane:

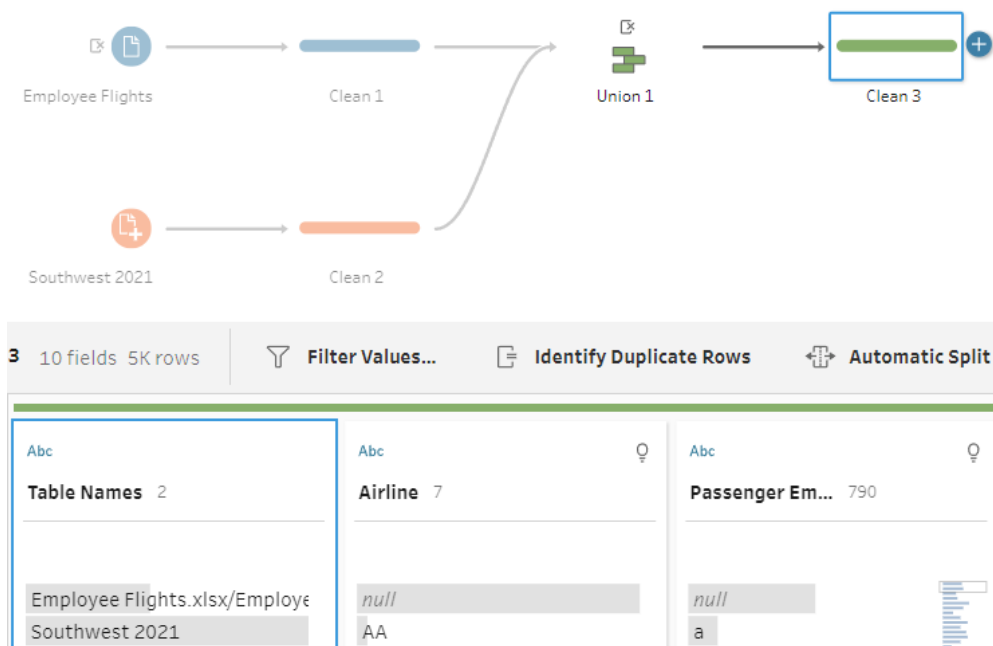


Figure 15.19: Selecting a field in the profile pane highlights the lineage of the field in the flow

Click the **null** value for **Airline** and observe the brushing as shown in Figure 15.19: this is proof that the null values in **Airline** all come from the Southwest files since those files did not contain a field to indicate the airline. We'll address the null values and do some additional cleanup:

1. Since we can see that all null values for **Airline** came from the Southwest files, we know the airline is Southwest. Double-click the null value and then type Southwest to replace null with the value you know represents the correct airline. Tableau Prep will indicate that a **Group** and **Replace** operation has occurred with a paperclip icon.



Note

In a real-world scenario, we would want to be a bit more careful. Even though all NULL values are currently only from the Southwest files, we might get NULL values from other airlines in the future. A calculation that explicitly checks the value of **Table Names** would be a more robust solution. We'll look at calculations in the next section.

2. We'll do an additional grouping to clean up the variations of **American**. Using the **Options** button on the **Airline** field, select **Group Values** | **Pronunciation**:

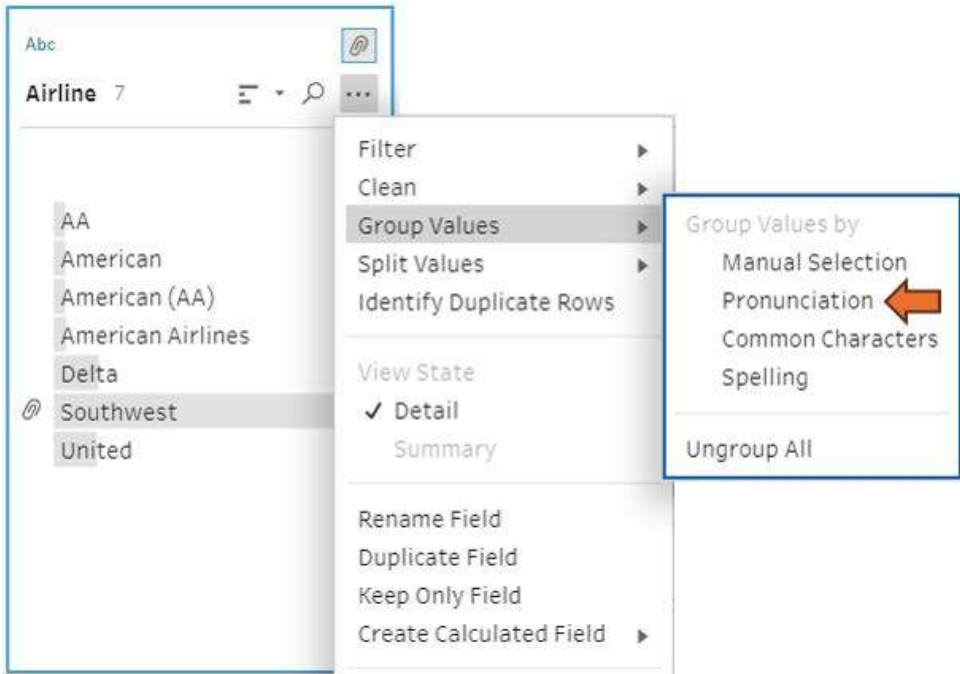


Figure 15.20: The ellipsis button on a field will reveal a plethora of options, from cleaning to filtering, to grouping, to creating calculations

Nearly all the variations are grouped into the **American** value. Only **AA** remains.

3. In the **Group Values by Pronunciation** pane that has appeared, select the **American Airlines** group and manually add **AA** by checking it in the list that appears to the right:

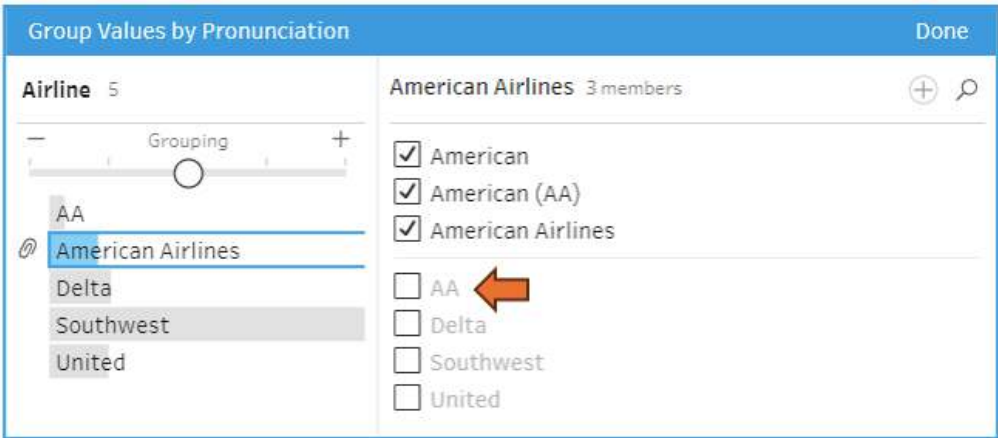


Figure 15.21: When grouping by pronunciation, you'll notice a slider allowing you control over the sensitivity of the grouping. You can also manually adjust groupings by selecting a field

4. Click **Done** on the **Group Values by Pronunciation** pane.
5. Next, select the **Table Names** field, which is no longer needed. Using either the toolbar option, the menu from a right-click for the field, or the Options button, select **Remove Field**.
6. Some fields in the **Profile** pane have a **Recommendations** icon (which resembles a lightbulb) in the upper-right corner. Click this icon on the **Passenger Email** field and then **Apply** the recommendation to assign a data role of **Email**:

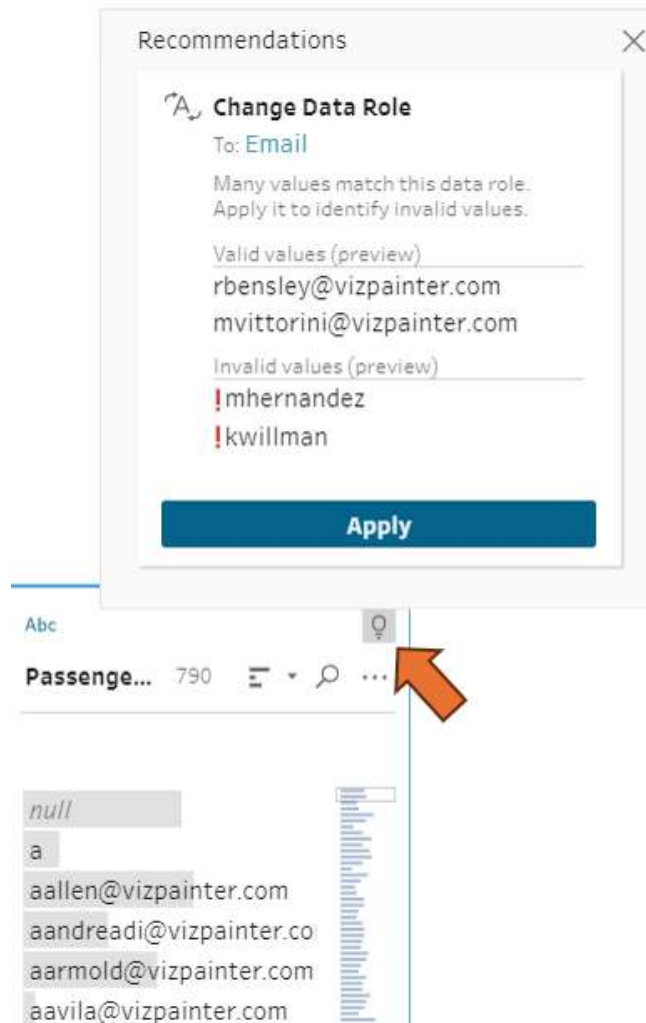


Figure 15.22: Recommendations will show when Tableau Prep has suggestions for cleaning a field

Data roles allow you to quickly identify valid or invalid values according to what pattern or domain of values is expected. Once you have assigned a data role, you may receive additional recommendations to either filter or replace invalid values.

After applying the recommendation, you'll see an indication in the **Profile** pane for invalid values (shown as a red exclamation mark in *Figure 15.22*). As you continue following the example, we'll consider some options for quickly dealing with those invalid values.

7. Click the **Recommendations** button on the **Passenger Email** field again. You'll see two new options presented. Click **Apply** on the **Group Values** option and replace invalid values with null:

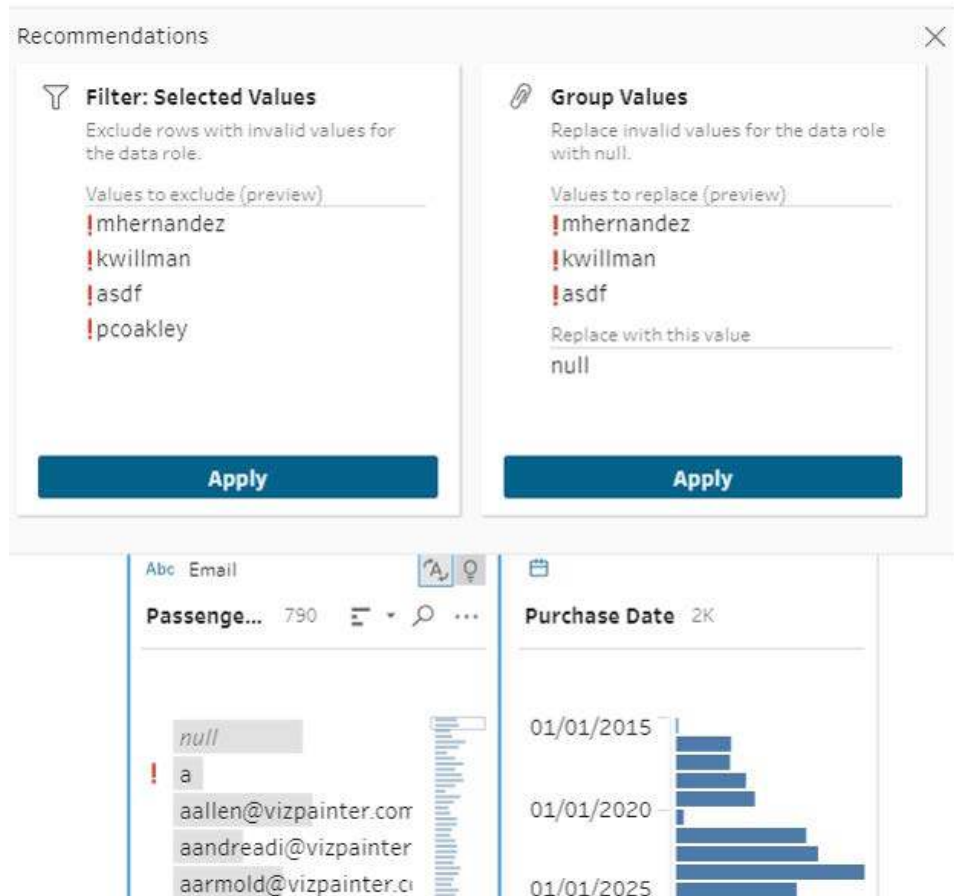


Figure 15.23: Here, Tableau Prep suggests either filtering out records with invalid values or replacing the invalid values with null. In this case, we don't want to filter out the entire record, but the invalid values themselves are useless and are best represented by null

8. Most of the remaining fields look fine, except for **Fare Type** (or possibly **Ticket Type**, depending on which name was kept when merging the fields previously). This field contains the values **1st Class** and **First Class**. Select both of these values by clicking each while holding down the *Ctrl* key and then group them together with the **First Class** value. Two interface options for grouping the values are indicated here:

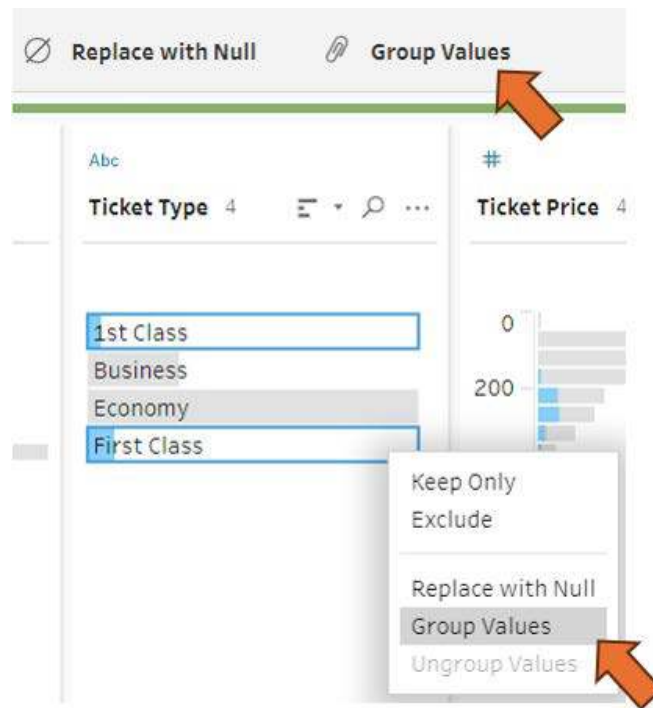


Figure 15.24: After selecting two or more values, you can group them together with the toolbar option or the right-click menu

9. At this point, we have a clean dataset that contains all our primary data. There's already a lot of analysis we could do. In fact, let's take a moment to preview the data. Right-click the **Clean 3** step and select **Preview in Tableau Desktop**:

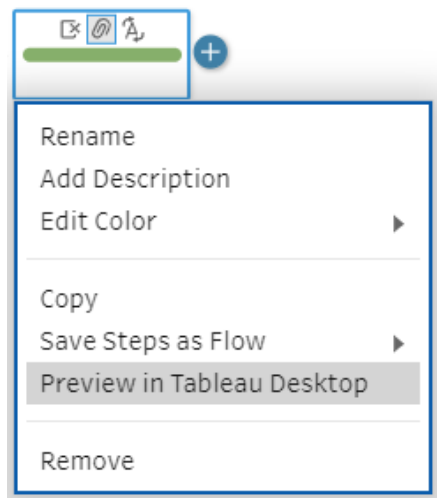


Figure 15.25: You may preview the data represented by any step in Tableau Desktop by selecting the option from the right-click menu for that step

A new data connection will be made and opened in Tableau Desktop. You can preview the data for any step in the flow. Take a few moments to explore the data in Tableau Desktop and then return to Tableau Prep. Now, we'll turn our attention to extending the dataset with some calculations, supplemental data, and a little restructuring.

Calculations and aggregations in Tableau Prep

Let's look at how to create calculations and some options for aggregations in Tableau Prep.

Row-level calculations

Calculations in Tableau Prep follow a syntax that's nearly identical to Tableau Desktop. However, you'll notice that aggregate calculations require the use of a **FIXED** level of detail function. This is because, outside of using a **FIXED** level of detail function, all aggregations are performed using an **Aggregate** step, which we'll consider shortly.

Calculations and aggregations can greatly extend our analytic capabilities and potentially increase performance by pre-calculating values based on complex logic. In our current example, there is an opportunity to analyze the length of time between ticket purchase and actual travel. We may also want to mark each record with an indicator of how frequently a passenger travels overall. Let's dive into these calculations as we continue our example with the following steps:

1. We'll start with a calculation to determine the length of time between the purchasing of tickets and the day of travel. Select the **Clean 3** step and then click **Create Calculated Field**. Name the calculation **Days from Purchase to Travel** and enter `DATEDIFF('day', [Purchase Date], [Travel Date])`.
2. Examine the results in the **Profile** pane. The new field should look like this:

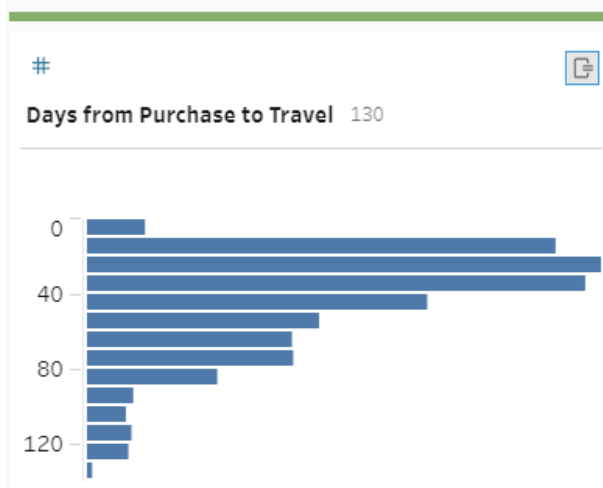


Figure 15.26: The calculated field shows up in the Profile pane

The default view here (as in many cases with numeric fields) is a summary binned histogram. You can change the view to see its details by selecting the ellipsis button at the upper right of the field and switching to **Detail**, which will show every value of the field:

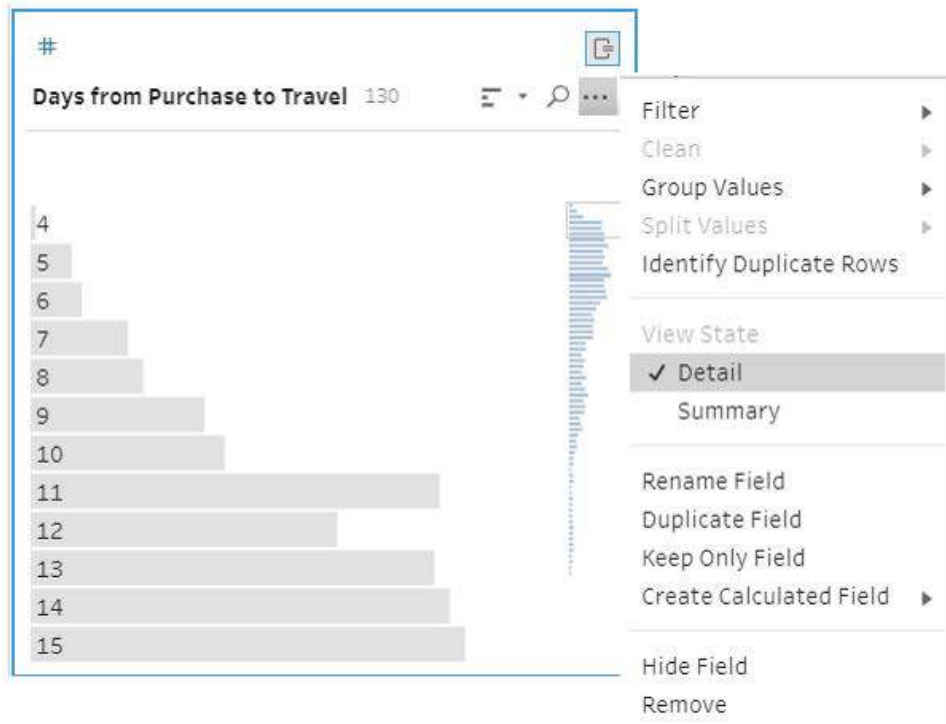


Figure 15.27: Numeric and date fields can be viewed in Summary or Detail

The shape of the data that's indicated by the default summary histogram is close to what we might have expected, with most people purchasing tickets closer (but not immediately prior) to the date of travel. There might be some opportunity for getting better deals by purchasing farther in advance, so identifying this pattern (and then exploring it more fully in Tableau Desktop) will be key for this kind of analysis.

Level of detail calculations

There are a few other types of analysis we may wish to pursue. Let's consider how we might create segments of passengers based on how often they travel.

We'll accomplish this using a **FIXED level of detail (LOD)** expression. We could create the calculation from scratch, matching the syntax we learned for Tableau Desktop to write the calculation like this:

```
{FIXED [Person] : COUNTD([Row_ID])}
```

 **Quick tip:** Enhance your coding experience with the **AI Code Explainer** and **Quick Copy** features. Open this book in the next-gen Packt Reader. Click the **Copy** button

(1) to quickly copy code into your coding environment, or click the **Explain** button

(2) to get the AI assistant to explain a block of code to you.



```
function calculate(a, b) {  
  return {sum: a + b};  
};
```

Copy**Explain**

1

2

 **The next-gen Packt Reader** is included for free with the purchase of this book. Scan the QR code OR go to packtpub.com/unlock, then use the search bar to find this book by name. Double-check the edition shown to make sure you get the right one.



The preceding calculation would count the distinct rows per person. Knowing that each row represents a trip, we could alternately use the code `{FIXED [Person] : SUM(1)}`, which would potentially be more performant, depending on the exact data source.

In this example, though, we'll use the interface to visually create the calculation:

1. Click the ellipsis button on the Person field and select **Create Calculated Field | Fixed LOD**:

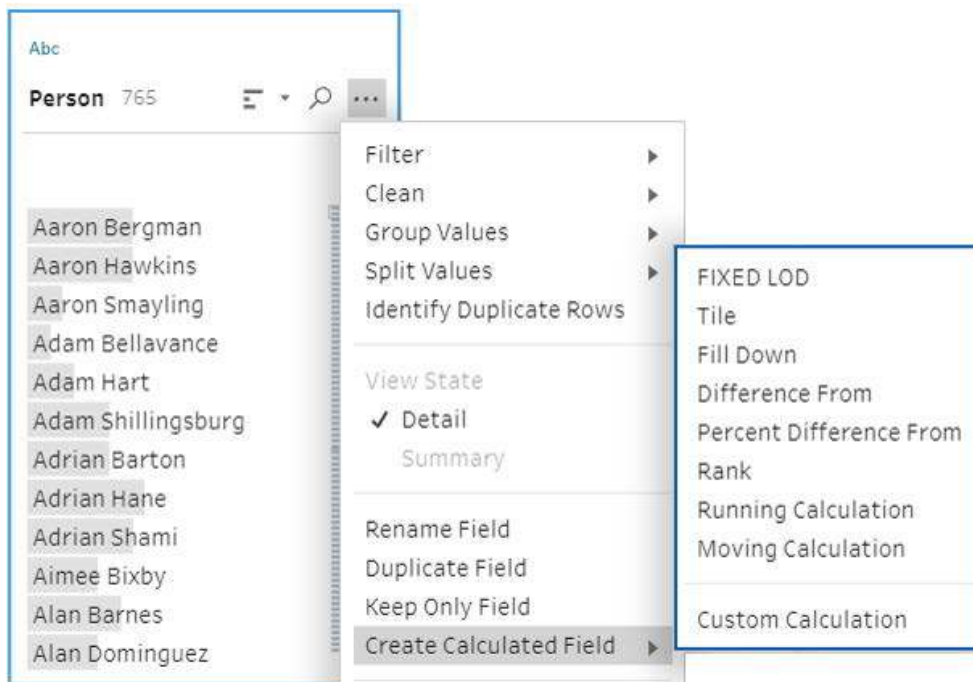


Figure 15.28: To create a Fixed LOD calculation, use the menu and select **Create Calculated Field | Fixed LOD**

In addition to **FIXED LOD**, you'll notice other options in the menu. These include the following: **Tile** groups values into equal segments – e.g. “quartiles.” **Fill Down** carries a value down row by row. This is useful in flat files where headings may not be repeated on every row. **Difference From** and **Percent Difference From** allow to you compute differences between rows of data (with groups and sorting that you define). **Rank** computes the rank for the selected field. **Running Calculation** allows you to compute running sums or averages. **Moving Calculation** computes sums or averages within a moving window of data.

- This will bring up a **Fixed LOD** pane allowing us to configure the LOD expression. The **Group by** field is already set to **Person** (as we started the calculation from that field), but we'll need to configure **Compute using** to perform the distinct count of rows and rename the field as **Trips per Person**, as shown here:

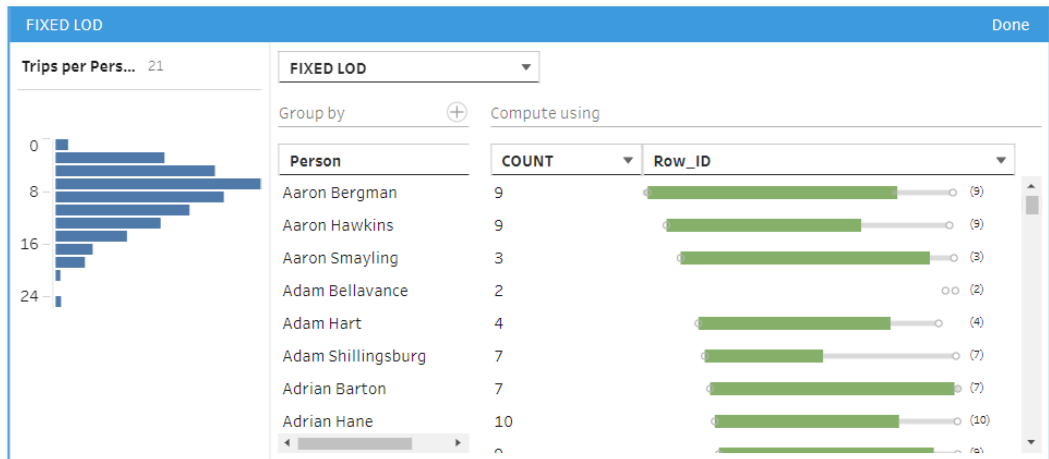


Figure 15.29: The Fixed LOD pane allows you to configure the LOD expression visually and get instant visual feedback concerning results

- Click **Done** when you have finished configuring the **Fixed LOD**.
- We'll use the **Trips per Person** field to create segments of customers. We'll accomplish this with another calculated field, so click **Create Calculated Field...** to bring up the code editor. Name the field **Frequency Segment** and enter the following code:

```
IF [Trips per Person] <= 2 THEN "Rarely"
ELSEIF [Trips per Person] <= 5 THEN "Occasionally"
ELSE "Frequently"
END
```

The code uses the **Trips per Person** field in an If Then Else construction to create three segments. You can see the correspondence between the fields in the preview pane:

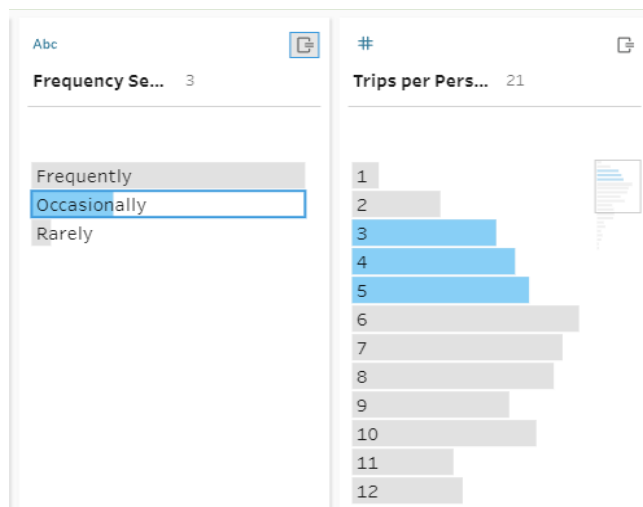


Figure 15.30: You can easily visualize how calculations relate to each other and other fields using the Profile pane

The Frequency Segment field can be used to accomplish all kinds of useful analysis. For example, you might want to understand whether frequent travelers typically get better ticket prices.

We've seen row-level and fixed LOD calculations and noted the option for **Rank**. Let's now turn our attention to aggregations.

Aggregations

Aggregations in Tableau Prep are accomplished using an aggregate step. We'll continue our flow with the idea that we want to better understand our frequency of travel segment:

1. Click the + symbol on **Clean 3** and add an **Aggregate** step. The new step will be named **Aggregate 1** by default:

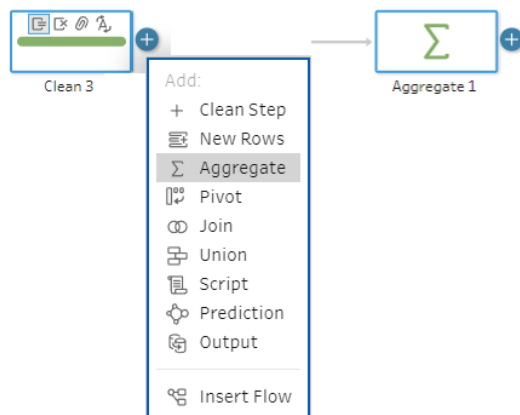


Figure 15.31: Adding an Aggregate step to the flow using the + symbol

2. Double-click the text **Aggregate 1** under the new step. This allows you to edit the name. Change the name from **Aggregate 1** to **Frequency Segment**.



Note

Give steps meaningful names to self-document the flow. This will greatly help you and others when you return to edit the flow in the future. Additionally, when you are editing the name of a step, the **Add a description** text will appear below the name, as shown in *Figure 15.32*.

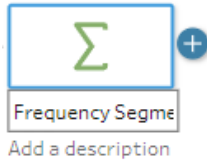


Figure 15.32: When editing the name of a step, you may also add a more verbose description to help document the step's purpose

Selecting the aggregate step reveals a pane with options for grouping and aggregating fields in the flow:

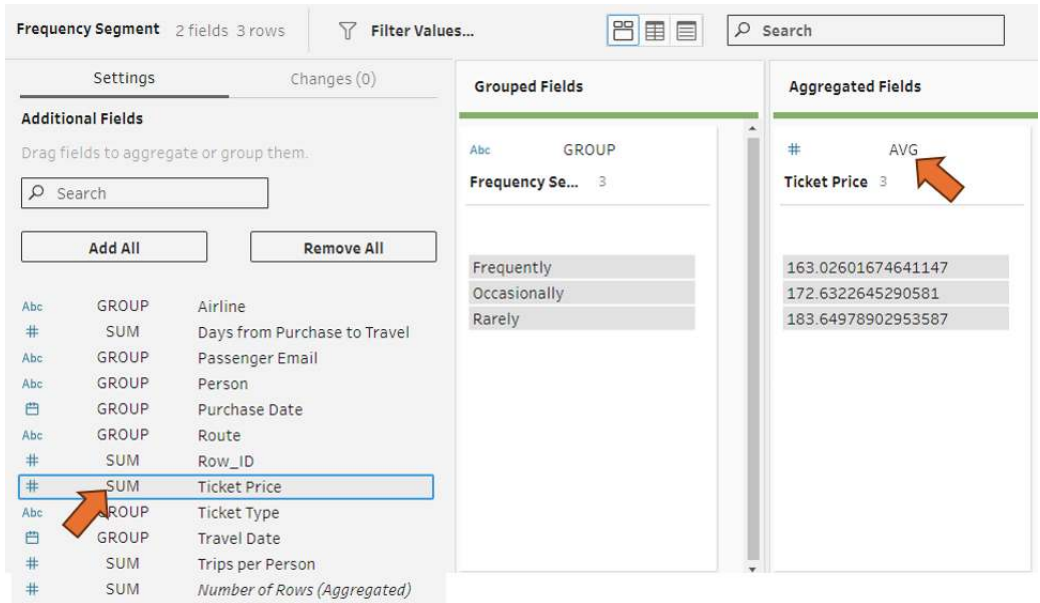


Figure 15.33: Grouping and aggregating within an aggregate step

You may drag and drop fields from the left to the **Grouped Fields** or **Aggregated Fields** sections and you may change the type of aggregation by clicking on the aggregation text on the left pane or in the Profile pane (indicated by arrows in *Figure 15.33*) and selecting a different aggregation from the resulting dropdown.

In *Figure 15.33*, notice that we've added **Frequency Segment** to **GROUP** and **Ticket Price** to **Aggregated Fields** as **AVG**. Notice also the **Number of Rows (Aggregated)** that appears at the bottom of the list of fields on the left. This is a special field that's available in the aggregation step.

1. Conclude the example by clicking the **+** icon that appears when you hover over the **Frequency Segment** aggregate step and adding an **Output** step:

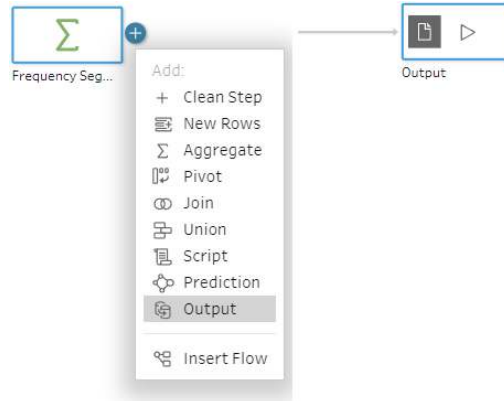


Figure 15.34: Adding an Output step to the flow using the + symbol

2. When you select the **Output** step, the **Output** pane shows options for saving the output and a preview of what the output will look like. In this case, we've configured the output to save to a **Comma Separated Values (.csv)** file named **Frequency Segment**:

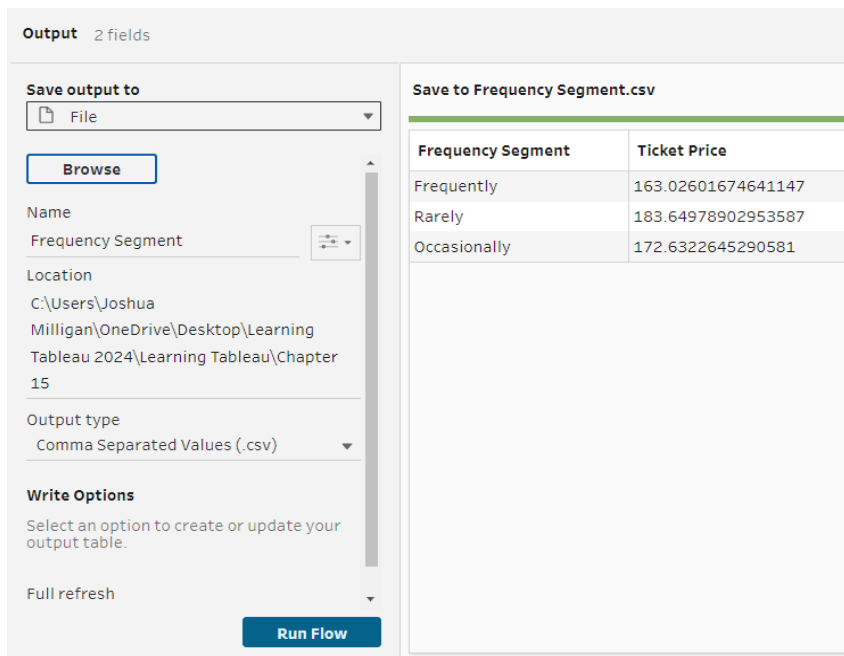


Figure 15.35: This output will contain exactly three rows of data

3. The **Output** pane also gives options for setting the output type, performing a full refresh of the data or appending to existing data, and running the flow.

We'll extend our flow in the next few sections to additionally output detailed data. The detailed data as well as the output file of aggregate data gives us some nice options for using Tableau's data model in Tableau Desktop to accomplish some complex analysis.

Let's continue by thinking about filtering data in Tableau Prep.

Filtering data in Tableau Prep

There are a couple of ways to filter data in Tableau Prep:

- Filter an input
- Filter within the flow

Filtering an input can be efficient because the query that's sent to the data source will return fewer records. To filter an input, select the input step and then click the **Filter Values...** button on the input pane:

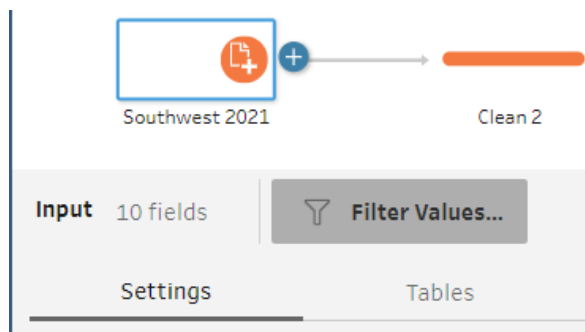


Figure 15.36: The **Filter Values...** option allows you to filter values on the input step. This could improve performance on large datasets or relational databases

The **Add Filter** dialog that pops up allows you to write a calculation with a Boolean (true/false) result. Only true values will be retained.

Filtering may also be done within a clean step anywhere in the flow. There are several ways to apply a filter:

- Select one or more values for a given field and then use the **Keep Only** or **Exclude** options.
- Use the **Option** button on a field to reveal multiple filter options based on the field type. For example, dates may be filtered by **Calculation...**, **Selected Values**, **Range of Dates**, **Relative Dates**, or **Null Values**:

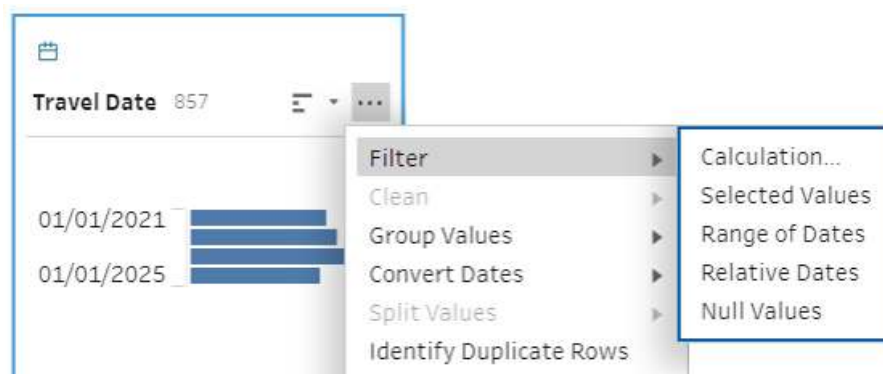


Figure 15.37: Filter options for a field include filtering by Calculation, Selected Values, Range of Dates, and Relative Dates, and keeping or excluding Null Values

- Select a field and then **Filter Values** from the toolbar. Similar to the way filters work in the input pane, you will be prompted to write code that returns true for records you wish to keep. If, for example, you wanted to keep records for trips scheduled on or after January 1, 2024, you could write code such as the following:

```
[Travel Date] >= MAKEDATE(2024, 1, 1)
```

While no filtering is required for the dataset in our example, you may wish to experiment with various filtering techniques.

At this point, your flow should look something like this:

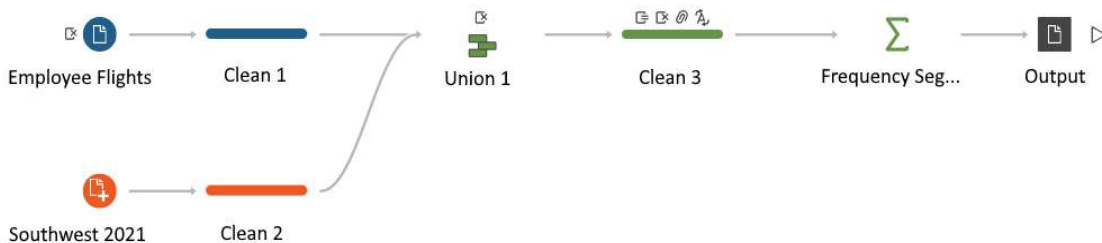


Figure 15.38: Your flow should look similar to this (exact placement and colors of steps may vary)

Let's conclude the Tableau Prep flow with some final transformations to make the data even easier to use in Tableau.

Transforming the data for analysis

Let's create a new branch in the flow to work once again with the detailed data. Click on the **Clean 3** step and examine the **Preview** pane. In particular, consider the **Route** field:

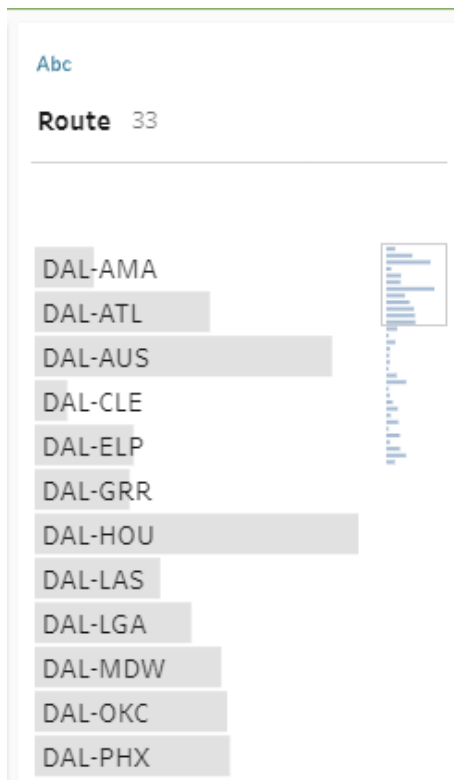


Figure 15.39: Route uses airport codes for origin and destination separated by a dash

Tableau Desktop (and Server) contains built-in geocoding for airport codes. But to accomplish our specific analysis goal (and open other possibilities for geospatial functions in Tableau Desktop), we'll supplement our data with our own geocoding. We'll also need to consider the shape of the data. Origins and destinations will be most useful split into separate fields, and if we want to connect them visually, we'll also want to consider splitting them into separate rows (a row for the origin and another row for the destination).

There are quite a few possibilities for visualizing this data. For example, we could keep the origin and destination on the same row and use a dual-axis map or map layers. If we want to connect origins with destinations with a line, we could keep them in the same row of data and use Tableau's `MAKELINE()` function. The example you'll follow here will direct you to split the data into separate rows.

If you are following along, here are the steps we'll take:

1. Use the + button that appears when you hover over the **Clean 3** step. Use that to add a new clean step, which will be automatically named **Clean 4**:

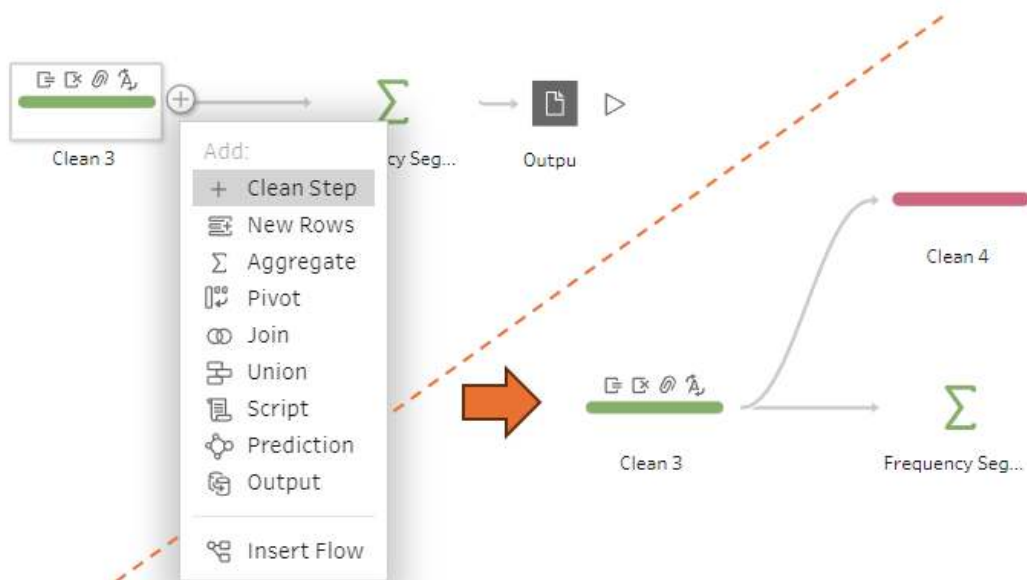


Figure 15.40: Adding a step that already has an output adds a new branch to the flow

2. In the **Clean 4** step, use the ellipsis button on the **Route** field and select **Split Values | Automatic Split**:

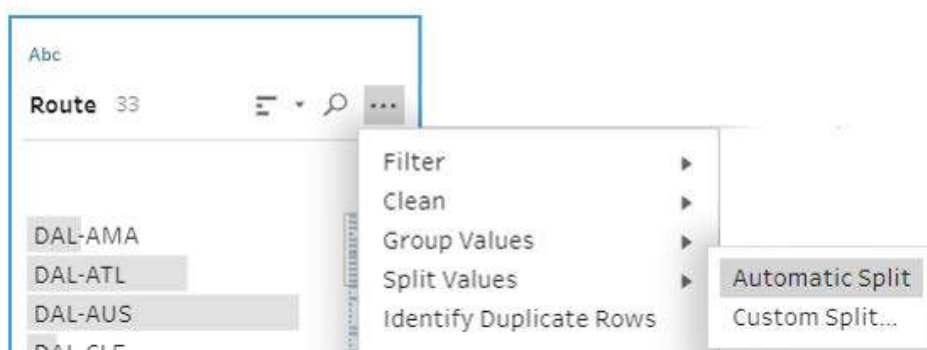


Figure 15.41: Split Values allows you to divide delimited strings into separate fields. Automatic Split attempts to determine the delimiter, while Custom Split... allows you greater options and flexibility

You'll now see two new fields added to the Clean step:

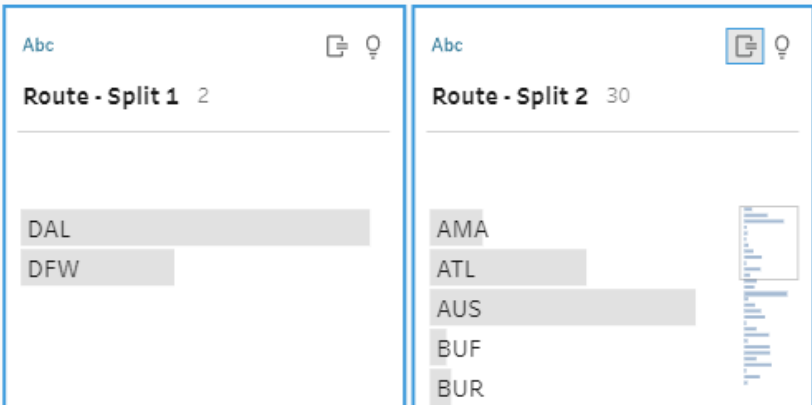


Figure 15.42: The results of the split will be new fields in the flow

3. **Route - Split 1** is the origin and **Route - Split 2** is the destination. Double-click the field name in the profile pane (or use the option from the ellipsis button) to rename the fields to **Origin** and **Destination**.
4. Locate the **US Airports.hyper** file in the *Chapter 16* directory. This file contains each Airport Code along with the Airport Name and Latitude and Longitude:

Airport Code 1K	Airport Name 1K	Latitude 1K	Longitude 1K
0V4	Aberdeen Regional Airpor	-33.84469986	-176.6459961
19S	Abilene Regional Airport	19.72139931	-174.2059937
23M	Abraham Lincoln Capital /	19.73880005	-171.7330017
2A5	Acadiana Regional Airpor	19.76009941	-170.4929962
2H0	Adak Airport	19.9205	-170.2200012
2K7	Addison Airport	20.00130081	-169.6620026
3AU	Adirondack Regional Airp	20.26530075	-168.095
4A7	Aiken Municipal Airport	20.78560066	-167.9219971
4U9	Ak-Chin Regional Airport	20.79560089	-166.5440063
52A	Akhiok Airport	20.89859962	-166.4660034
57C	Akiak Airport	20.96290016	-166.3390045
5B2	Akron Canton Regional Ai	21.1529007	-166.2709961

Figure 15.43: The hyper extract contains the data we'll need to supplement the flow with our own geospatial data

5. Make a connection to this file in Tableau Prep. You may choose to drag and drop the file onto the Tableau Prep canvas or use the **Add Connection** button from the interface. Tableau will automatically insert an input step named **Extract (Extract.Extract)**. Feel free to change the name of the input step to **Airport Codes**.

We'll want to join the **Airport Codes** into our flow to look up Latitude and Longitude, but before we do, we'll need to account for the fact that **Origin** and **Destination** in **Clean 4** are currently both on the same row of data. One option is to pivot the data.

- Use the + button on the **Clean 4** step to add a **Pivot** step:

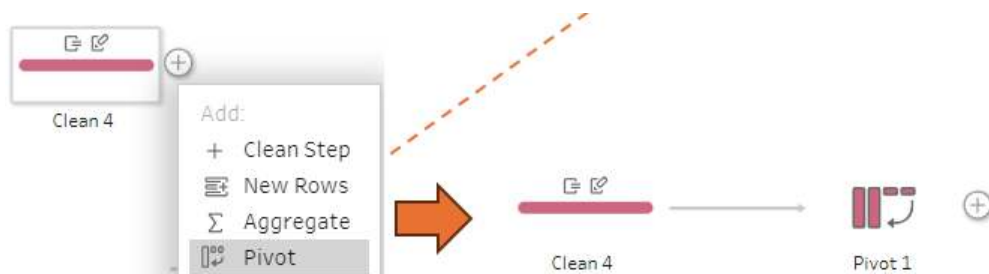


Figure 15.44: Adding a Pivot step from Clean 4

- The pivot pane gives you options for transforming rows into columns or columns into rows. We'll keep the default option of **Columns to Rows**. Drag both the **Origin** and **Destination** fields into the **Pivoted Fields** area of the pane:

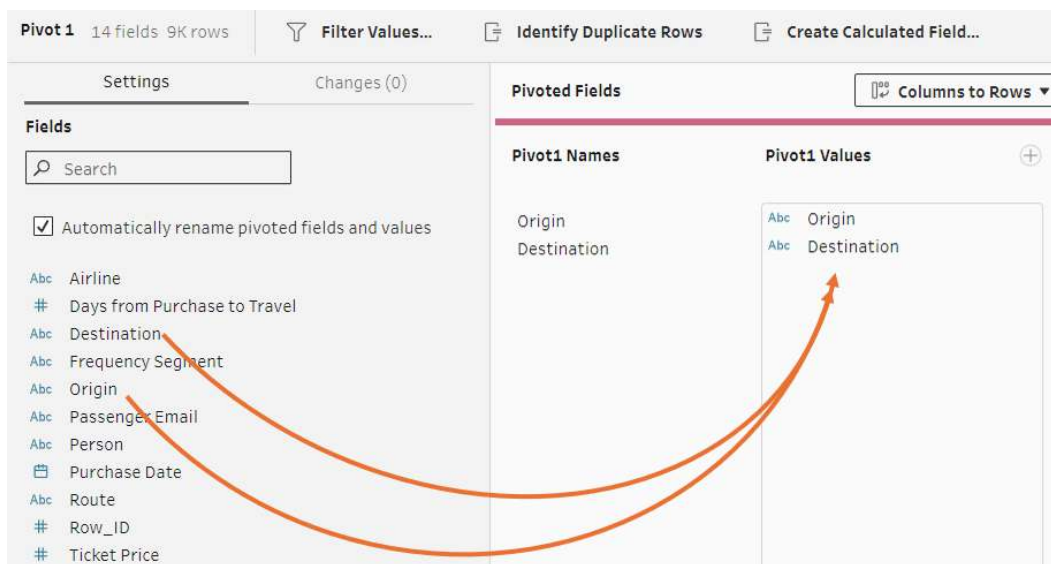


Figure 15.45: Pivot1 Names keeps values from the original column names, while Pivot1 Values contains the actual values from Origin and Destination

- 8. As a shortcut, instead of *steps 6 and 7*, you could have selected both the **Origin** and **Destination** fields in the **Clean 4** step, and selected **Pivot Columns to Rows**:

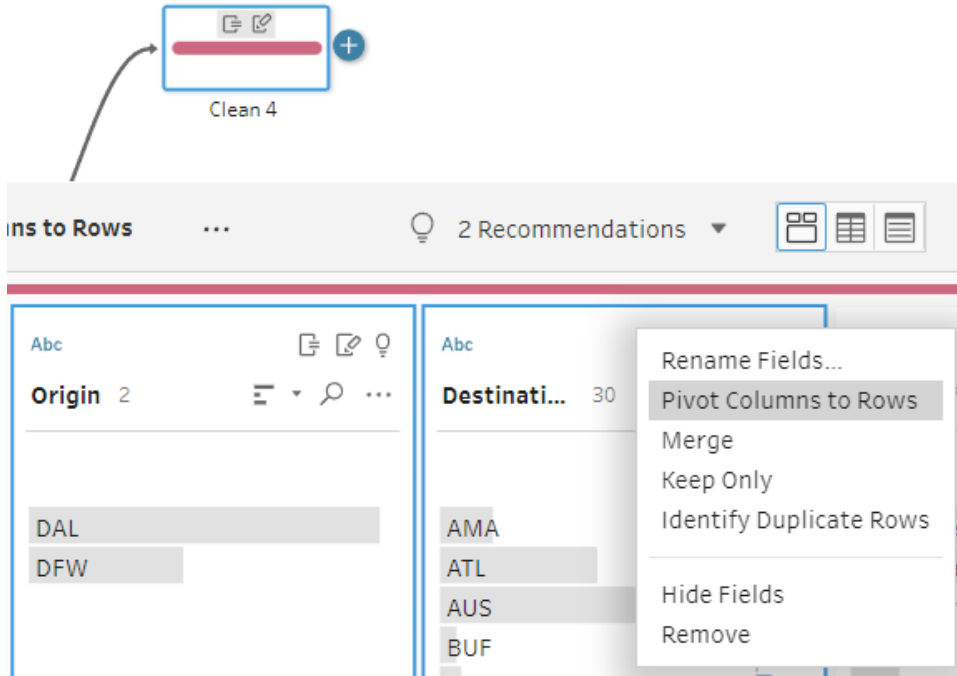


Figure 15.46: A shortcut for pivoting columns to rows

- 9. Double-click the text for **Pivot1 Values** and rename the field **Airport Code**. This field will contain all the airport codes for both the origin and destination records:

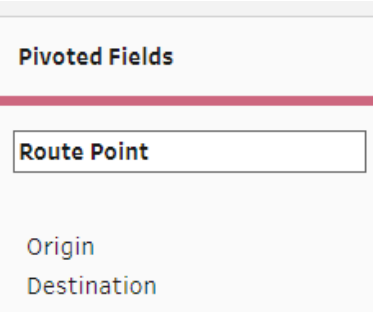


Figure 15.47: Renaming Pivot Values

10. Similarly, double-click the text for **Pivot1 Names** and rename the field **Route** to **Point**. This field will label each record as either **Origin** or **Destination**.

At this point, we have a dataset that contains a single record for each endpoint of the trip (either an origin or destination).

Note



Notice that the pivot resulted in duplicate data. What was once one row (origin and destination together) is now two rows. The record count has doubled, so we can no longer count the number of records to determine the number of trips. We also cannot SUM the cost of a ticket as it will double count the ticket. We'll need to use MIN/MAX/AVG or some kind of level of detail expression or filter to look at only origins or destinations. While many transformations allow us to accomplish certain goals, we have to be aware that they may introduce other complications.

The only location information we currently have in our main flow is the airport code. However, we already made a connection to `Airports.hyper` and renamed the input step **Airport Codes**.

11. Locate the **Airport Codes** input step and drag it over to the **Pivot 1** step. Drop it onto the **Join** area:

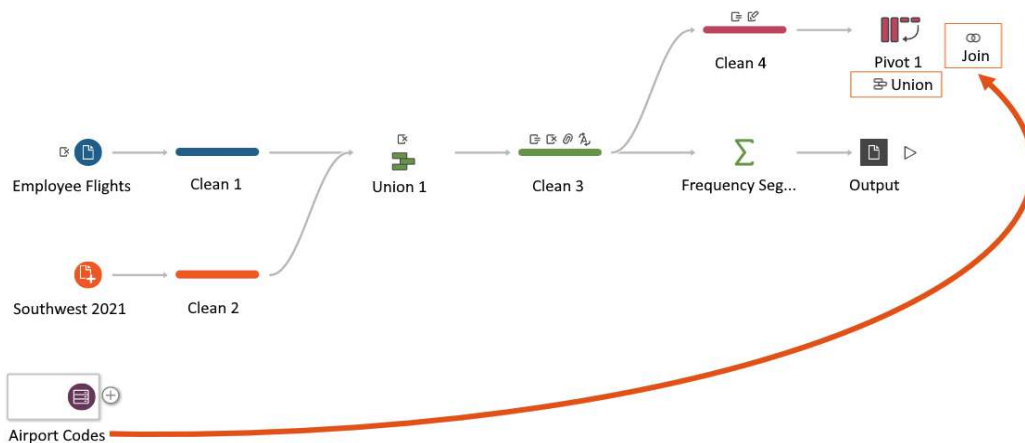


Figure 15.48: Dragging Airport Codes to the Join area of the Pivot step

After dropping the **Airport Codes** input step onto the **Join** area, a new join step will be created, named **Join 1**. Take a moment to examine the Join pane:

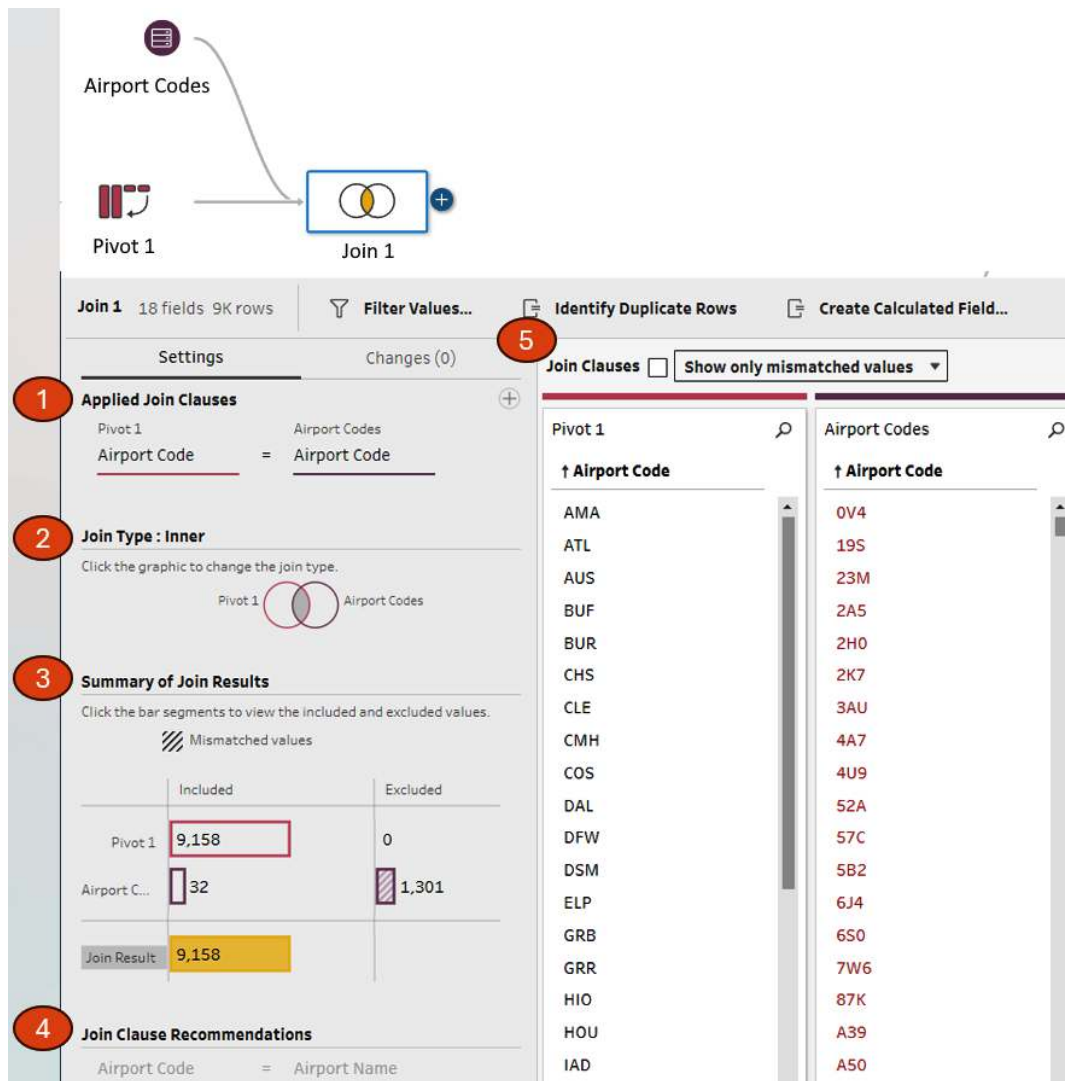


Figure 15.49: The join pane gives a lot of information and options for configuring the join and understanding the results. Important sections of the interface are numbered, with descriptions below

You'll notice the following features in Figure 15.49:

1. **Applied Join Clauses:** Here, you have the option to add conditions to the join clause, deciding which fields should be used as keys to define the join. You may add as many clauses as you need.
2. **Join Type:** Here, you may define the type of join (inner, left, left inner, right, right inner, outer). Accomplish this by clicking sections of the Venn diagram.
3. **Summary of Join Results:** The bar chart here indicates how many records come from each input of the flow and how many matched or did not match. You may click a bar segment to see filtered results in the data grid. Notice that the colors of the bars match the input steps and the join step as applicable.

4. **Join Clause Recommendations:** If applicable, Tableau Prep will display probable join clauses that you can add with a single click.
5. **Join Clauses:** Here, Tableau Prep displays the fields used in the join clauses and the corresponding values. Any unmatched values will have a red font color. You may edit values by double-clicking them. This enables you to fix individual mismatched values as needed.

We do not need to configure anything in this example. The default of an **Inner** join on the **Airport Code** fields works well. We can confirm that all 9,158 records from the **Pivot** step are kept. Only 32 records from the **Airport Codes** hyper file are actual matches (1,301 records didn't match). That is not concerning. It just means we had a lot of extra codes that could have possibly supplemented our data but weren't actually needed. Now, continuing from our previous example:

12. From **Join 1**, add a final output step and configure it to output to a .csv file named **Airline Travel.csv**.
13. Run the flow by using the run button at the top of the toolbar or by clicking the run button on the output step.

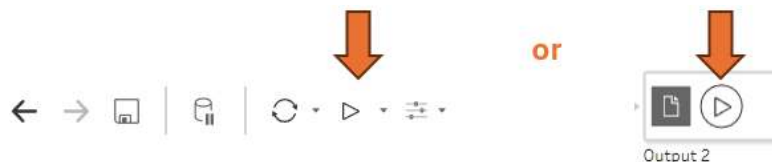


Figure 15.50: The toolbar allows you to run the flow for all outputs or a single output, while the button on the output step will only run the flow for that output

Your final flow will look something like this:

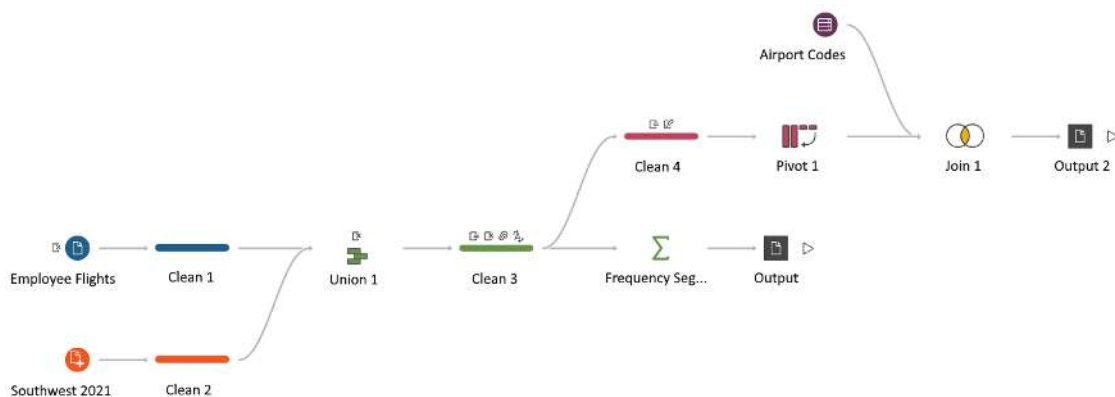


Figure 15.51: Your final flow will resemble this, but may be slightly different in appearance

The Chapter 15 Complete (clean).tflx file is a bit cleaned up with appropriate step labels and descriptions. As a good practice, try to rename your steps and include descriptions so that your flow is easier to understand.

This is what the cleaned version looks like:

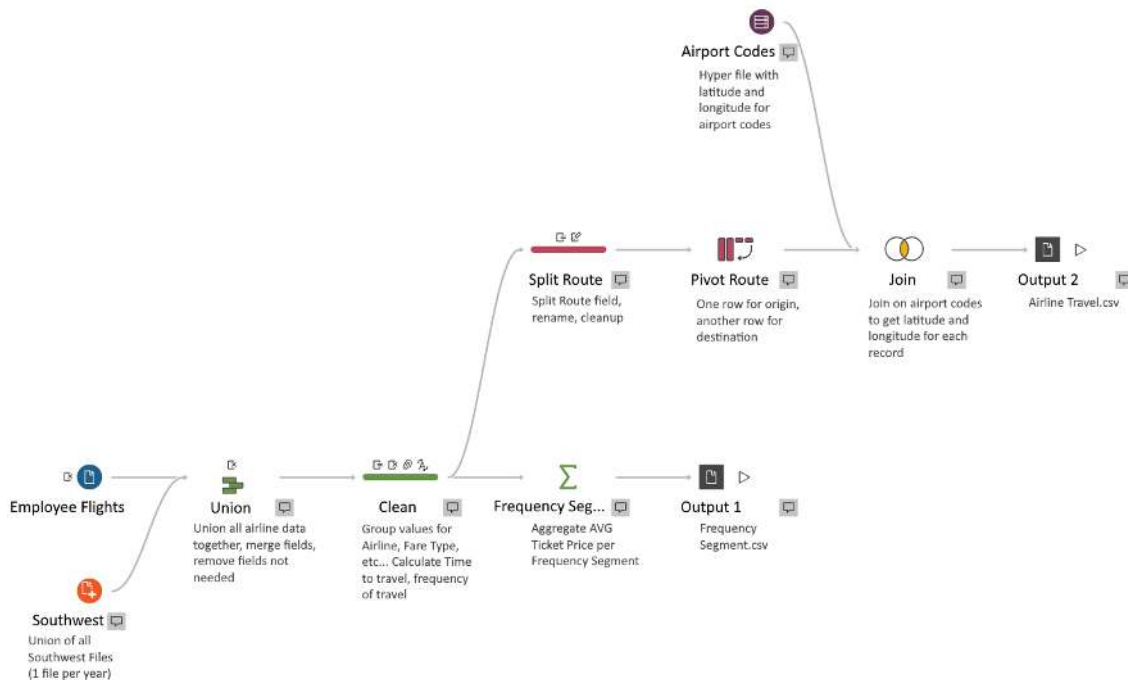


Figure 15.52: This flow is cleaned up and contains “self-documentation”

Once the flow has been executed, open the Airline Travel.twb workbook in the \Learning Tableau\ Chapter 15 directory to see how the data might be used and to explore it on your own:

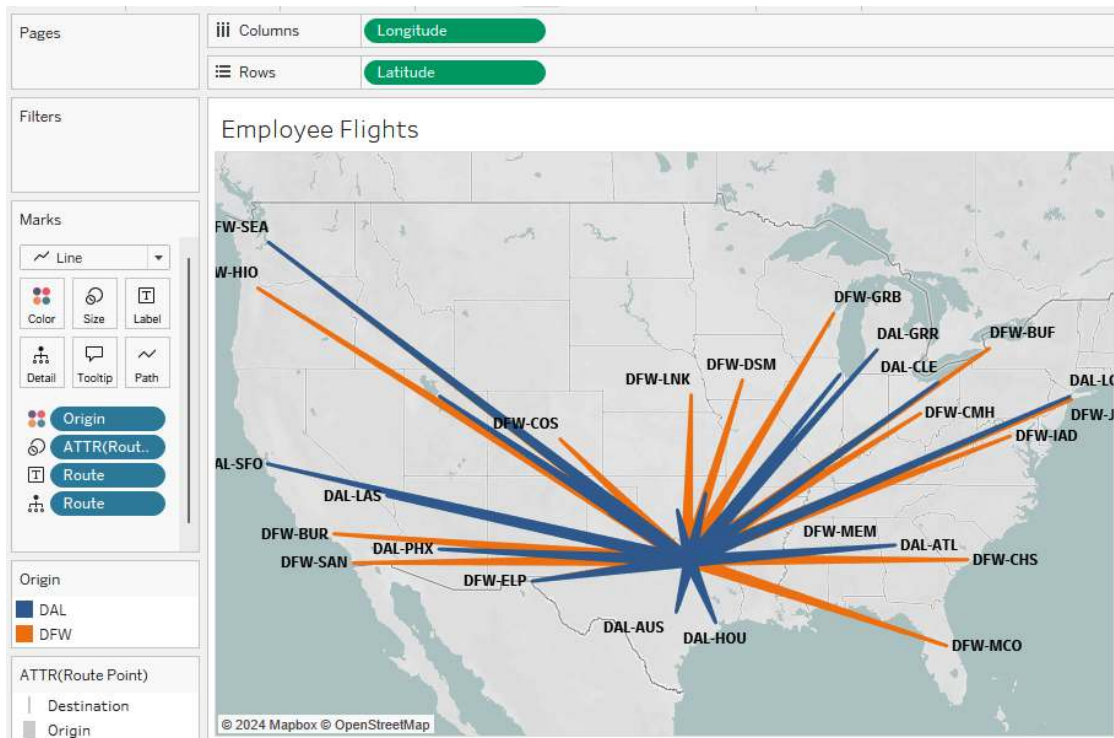


Figure 15.53: Exploring the data in the *Airline Travel.twb* workbook, which shows all flight destinations from Dallas Love Field (DAL) or Dallas Fort Worth (DFW)

Note



Unlike `.tde` or `.hyper` files, `.csv` files may be written to, even if they are open as a data source in Tableau Desktop. You will receive an error if you run a flow that attempts to overwrite a `.tde` or `.hyper` file that is in use. Additionally, you may rearrange the field order for a `.csv` file by dragging and dropping fields into the Profile pane of a clean step prior to the output.

With our data now prepped and ready for in-depth analysis, let's return to Tableau Prep Builder to consider how we might use parameters to control our flow.

Using parameters in Tableau Prep Builder

We've seen parameters in Tableau Desktop. Parameters in Tableau Prep are very similar and are capable of being used to control the input, output, and logic within a flow.

Let's say, for example, that you wanted to create a parameter that indicated whether the dataset was test or production data. Depending on the value, you might change aspects of the data in the flow or even output to a different location.

To create or manage parameters in Tableau Prep Builder, use the **Parameters** button on the toolbar:

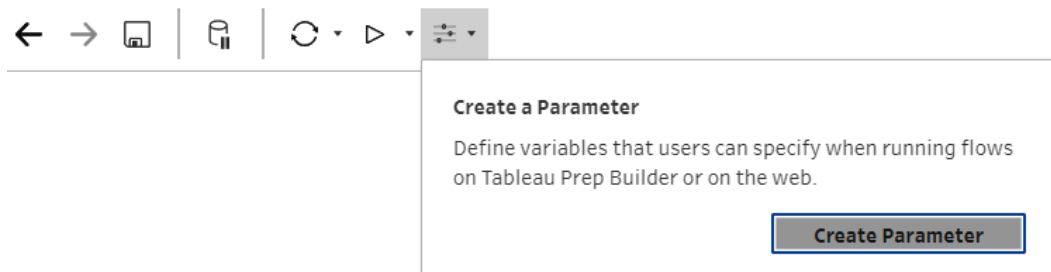


Figure 15.54: Clicking the Parameters button on the toolbar reveals the option to create a parameter or manage any existing parameters

Once you have clicked **Create Parameter**, you will see another window with all the options for your new parameter:

A screenshot of the "Create Parameter" dialog box. The dialog has a title bar with "Create Parameter" and a close button (X). Inside, there are several sections: "Name (required)" with a text input field containing "Test or Production"; "Description" with a text input field containing "Determines whether the flow runs for test data or production data"; "Data type" with a dropdown menu showing "String"; "Current value (required)" with a dropdown menu showing "Test"; "Allowable values" with two radio buttons, "All" and "List", where "List" is selected; "List of values" with a list containing "Test" (highlighted in blue), "Production", and "Click to add new value" (disabled); a "Delete" button with a trash icon; a checkbox "Require selection at run time" which is unchecked; and at the bottom, a "Delete Parameter" button with a trash icon, a "Cancel" button, and an "OK" button.

Figure 15.55: Creating a new parameter

You'll find much is very similar to creating a parameter in Tableau Desktop. You can give the parameter a name and a description, set **Data type** and **Current value**, and determine whether **All** entries are valid or whether you will define a **List** of values.



Tip

Checking **Require selection at run time** will force a selection whenever the flow is run. Consider using this option if the selection of the value will make a major impact on the flow and you want to minimize user error.

Once you have created a parameter, you may use it throughout the flow. As of this writing, you may use parameters in:

- **Input Steps:** As part of a filename, file path, database table name, or in a custom SQL statement
- **Various Flow Steps:** You may use a parameter as part of a filter calculation or calculated field
- **Output Steps:** As part of a filename, file path, database table name, Excel worksheet name, or a published data source name

In an output step connected to a database engine, you may also use a parameter value in the custom SQL that runs before or after the flow execution.

Here, for example, is a custom SQL statement that runs after the flow that inserts a new record containing a timestamp and either the value **Production** or **Test** into an `Execution_Log` table:

Save output to

Database and Cloud Storage ▼

Settings Custom SQL

Before After

Optional: Execute this SQL after writing flow data to database table.

Insert Parameter ▼

```
INSERT INTO Execution_Log
VALUES (GETDATE(),
<Test or Production>)|
```

Figure 15.56: Using a parameter in an Output step custom SQL statement

Consider another example of using our parameter to define the output file:

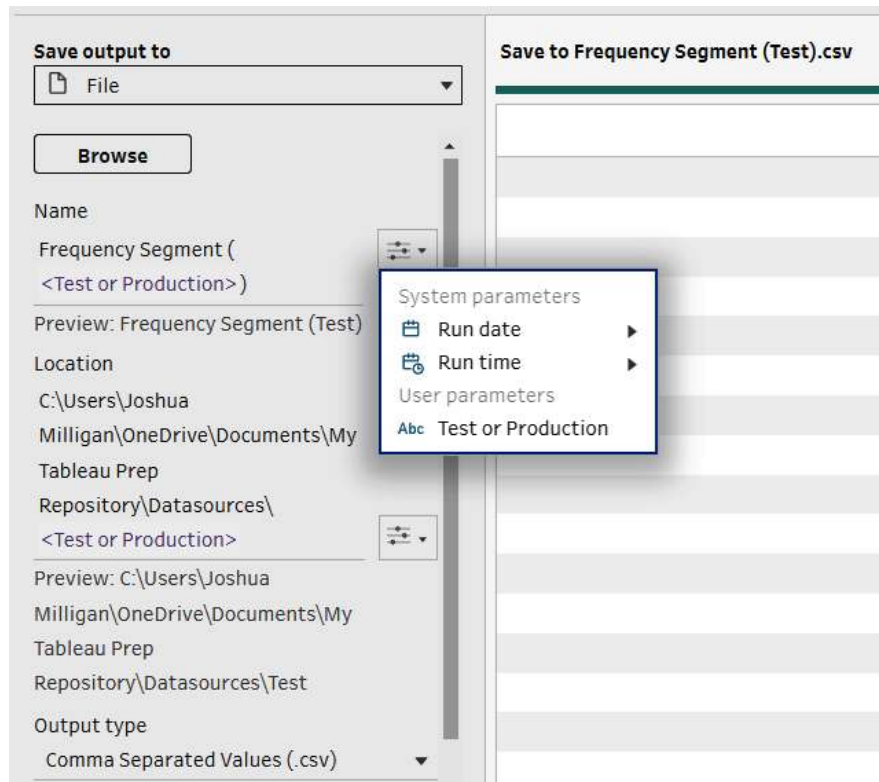


Figure 15.57: Using a parameter to define an output filename

You'll notice the parameter button in the output step interface that allows you to insert parameters into the settings. Here, we've defined the name of the output as `Frequency Segment (<Test or Production>)`, which will result in either `Frequency Segment (Test)` or `Frequency Segment (Production)`, depending on the parameter value. We've also used the parameter to dynamically select the location of the file. Notice also the options for inserting **Run date** or **Run time** (which are **System parameters**).

Consider the flexibility you have with parameters! You can create a flow that dynamically determines the source, applies different filter logic and calculations throughout the flow, and then outputs a different file, all based on a single selection of whether you want to run for test or production. And that's only one of an unlimited number of examples of how you might use parameters in your flow!

With a good foundation on parameters in Tableau Prep, let's wrap up by considering some options for automating Tableau Prep flows.

Options for automating flows

Tableau Prep Builder allows you to design and run flows using the application. Sometimes, data cleansing and prepping will be a one-time operation to support an ad hoc analysis. However, you will often want to run a flow subsequently to capture new or changed data and to cleanse and shape it according to the same pattern. In these cases, you'll want to consider some options for automating the flow:

- **Tableau Prep Builder** may be run via a command line. You may supply JSON files to define credentials for input or output data connections. This enables you to use scripting and scheduling utilities to run the flow without manually opening and running the Tableau Prep interface. Details on this option are available from Tableau Help: https://help.tableau.com/current/prep/en-us/prep_save_share.htm#refresh-output-files-from-the-command-line.
- **Tableau Prep Conductor**, an add-on to Tableau Server or Tableau Cloud (requiring a Data Management license), gives you the ability to publish entire flows from Tableau Prep Builder to Tableau Server or Tableau Cloud and then either run them on-demand or on a custom schedule. It also provides monitoring and troubleshooting capabilities.

Summary

Tableau Prep's innovative paradigm of hands-on data cleansing and shaping with instant feedback greatly extends the Tableau platform and gives you incredible control over your data. In this chapter, we considered the overall interface and how it allows you to iteratively and rapidly build out a logical flow to clean and shape data for the desired analysis or visualization.

Through a detailed, yet practical, example that was woven throughout this chapter, we explored every major transformation in Tableau Prep, from inputs to unions, joins, aggregates, and pivots, to outputs. Along the way, we also examined other transformations and capabilities, including calculations, splits, merges, and the grouping of values. This gives you a foundation for molding and shaping data in any way you need.

In the next chapter, we'll continue our journey with some thoughts on how to extend Tableau functionality through extensions and scripts!

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>



16

Extending Tableau: Extensions and Scripts

So far, most of what we've covered in this book has been focused on features of Tableau that work "out of the box." However, Tableau is also incredibly flexible, and you can extend the functionality greatly through extensions and custom scripts (such as Python or R). In the next chapter, we'll go even further by considering how to leverage AI and machine learning in your analytics. All of this will give you incredible power and flexibility to take your analysis and creativity to new levels!

As in other chapters, we'll cover both concepts and practical examples. Many of the more advanced examples will require setting up your environment to run external repositories, services, and platforms such as a web server or **SDK (Software Development Kit)**. It is beyond the scope of this book to provide in-depth instructions on how to set up these components, and the exact steps may change over time.

Instead, you will be directed to the appropriate resources and documentation to set up any additional resources required. The examples will include details that assume you have completed the required setup. Sections with examples requiring advanced setup will state the requirement up front so that you are able to decide whether you wish to follow the example or merely read the content.

In this chapter, we'll cover the following topics:

- Tableau extensions
- Tableau analytics extensions
- Table extensions

We'll start by covering dashboard extensions and then move on to analytics extensions and table extensions.

Tableau extensions

Tableau has two main types of extensions that leverage the Tableau Extensions API: **Viz Extensions** and **dashboard extensions**. Both allow developers to add additional capabilities to Tableau. You've already had some exposure to Viz Extensions. In this chapter, we'll focus primarily on using and even developing dashboard extensions. But note that Viz Extensions leverage the same development framework.

In *Chapter 8, Telling a Data Story with Dashboards*, we noted that the Dashboard Extension object was one of many objects that can be added to a dashboard. Now, we'll consider the details of what an extension is, how to use extensions, and even how to create a dashboard extension.

What is a dashboard extension?

A Tableau dashboard extension, also known simply as a **dashboard extension**, is a web application that integrates into a dashboard, providing extended capabilities beyond the built-in features of Tableau. Extensions can perform many functions on a dashboard, such as providing an interface for user input, rendering graphical feedback, and integrating external applications.

Whether you will simply be using extensions or are interested in developing your own, it is important to understand how dashboard extensions work. Obviously, if you will be developing extensions, you will need a deeper understanding! Take a moment to examine the following diagram, which illustrates the fundamental architecture around an extension:

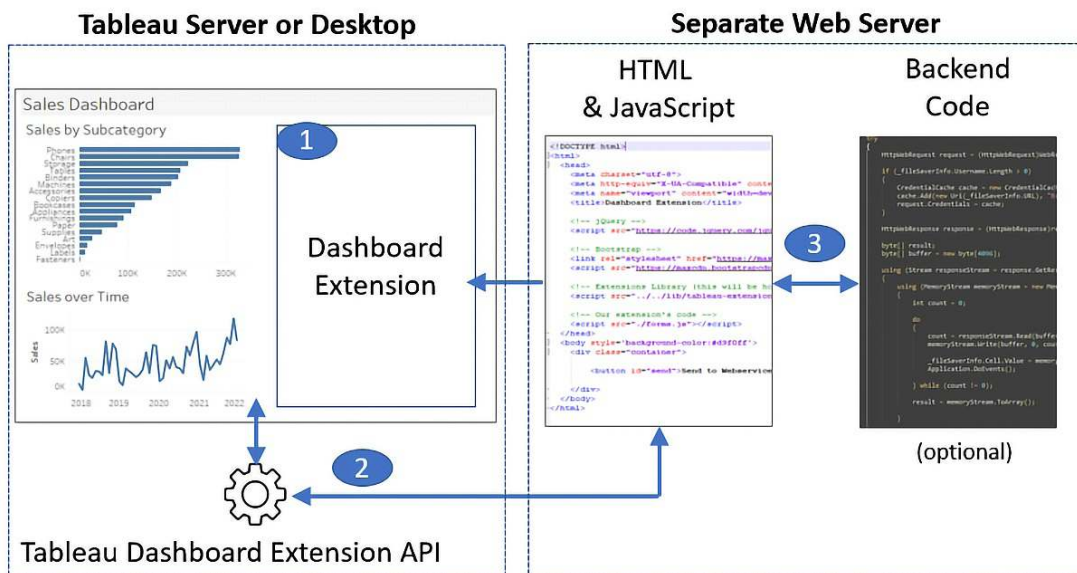


Figure 16.1: The basic architecture of a Tableau dashboard extension

We'll consider more of the technical details as we proceed through the next few sections, but in general, an extension is simply code such as HTML, JavaScript, and often other code (such as PHP or .NET) that can be executed on a separate web server. This code accomplishes several things (numbered in *Figure 16.1*):

1. A dashboard extension may be added to a dashboard using the Dashboard Extension object. It can be added as either a tiled or floating object. Often, the dashboard extension will provide a user interface. For example, you may have controls or input objects for the user to manipulate. Or you may provide the user with graphical feedback based on backend processing. Occasionally, there won't be an interface (for example, an extension that passively collects dashboard usage data), but in most cases, there will be some kind of user input or graphical output. A Dashboard Extension object is always required to use an extension. But in cases where the interface is not needed, it may be hidden from the end user by positioning it on the dashboard in such a way that it has no width or height or floats behind other objects.
2. The code will also make use of the **Tableau Dashboard Extensions API** – a programmatic interface for objects and actions on the dashboard. This enables the dashboard extension to do many things, such as read data from parameters and views on the dashboard, perform certain tasks when specific actions are triggered, or even programmatically change values on a dashboard.
3. The HTML renders any interface on the dashboard and JavaScript may be used alongside any backend code. Together, the JavaScript and backend code may access external systems and APIs, read or write to external databases, and more. This provides a wide range of possibilities for integrating Tableau into processes and applications across an organization. For example, a dashboard user might select bars in a bar chart representing errors and then enter corrected values in the Dashboard Extension object. The backend code might use these actions and data to update an external process to correct the errors or send an alert to the appropriate person.

With this foundation, let's move on to consider how to use and ultimately create dashboard extensions.

Using a dashboard extension

We'll work through a simple example to demonstrate how you can add, configure, and use an extension in a dashboard. Specifically, we will use the **Image Map Filter** extension (created by Tableau and Salesforce), which will allow us to use an image of a hospital floorplan as an interactive filter selection. You'll find the complete example in the *Chapter 16 Complete workbook*.



Tip

When you first open a workbook that uses dashboard extensions, you may receive a prompt to allow the extension. This is a security feature that allows you to disable an extension you do not trust.

To follow along with this example, use the Chapter 16 Starter workbook, locate the **Hospital Rooms** dashboard, and then drag and drop the **Extension** object onto the dashboard, as illustrated:

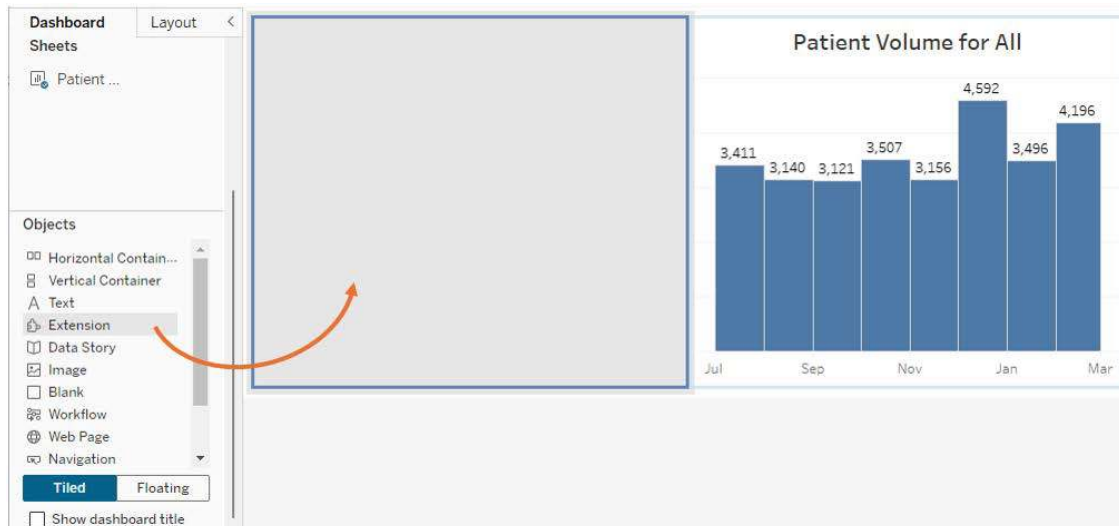


Figure 16.2: Drag and drop the Extension object onto the dashboard

Dashboard extension objects are just like other objects and may be added to a dashboard as either floating or tiled objects.

Note



Most dashboard extensions provide user interaction – either input, output, or both – and you will want to find an appropriate spot to show them in the flow of the dashboard. However, some dashboard extensions may operate in the background and you may not want them to ever be visible to the end user. If you wish to hide an extension (or any object, for that matter) from an end user, consider floating the object on the dashboard, switching to the **Layout** tab on the left, and setting **X**, **Y**, **Height**, and **Width** all to a value of 1.

Dropping the Extension object onto a dashboard will bring up the **Add an Extension** dialog, which looks like this:

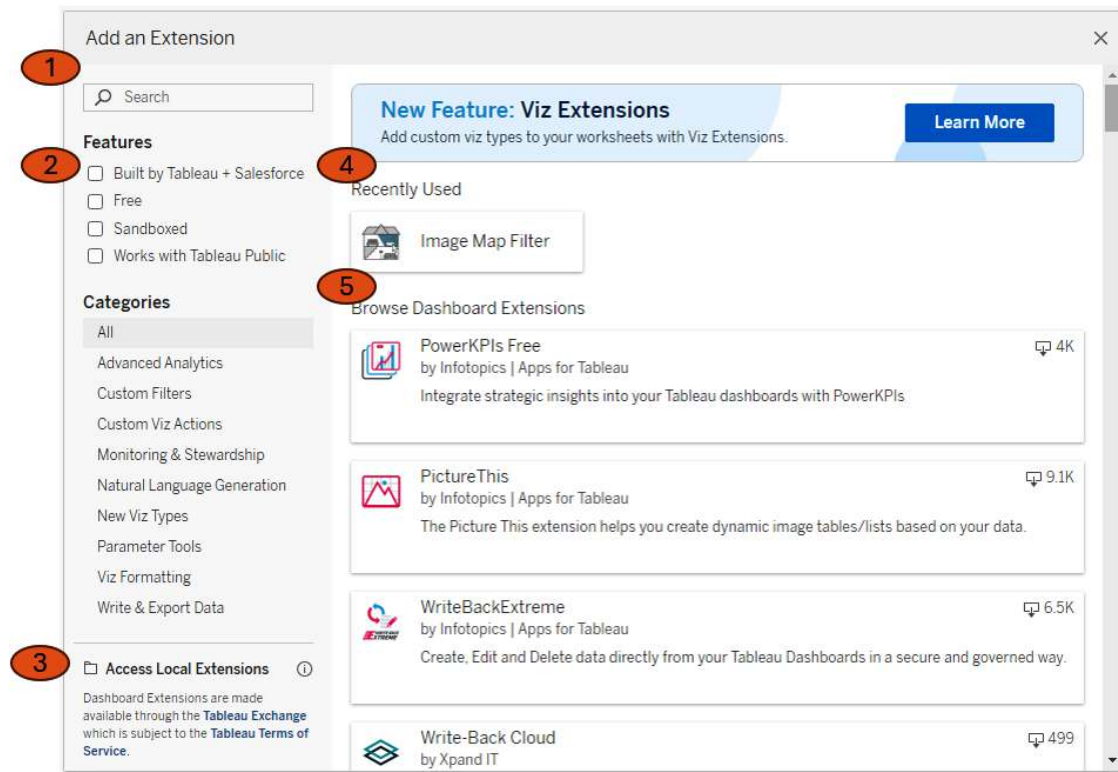


Figure 16.3: The Add an Extension dialog

The **Add an Extension** screen allows you to select which extension you wish to add to the dashboard. By default, you will see a list of extensions hosted on the **Tableau Exchange**, an online repository of extensions that have been created by Tableau (Salesforce) or third-party developers. The screen consists of various sections, numbered in *Figure 16.3* and explained here:

1. The left panel allows you to search for an extension by name or keyword(s). You may also narrow down the list by selecting one of the **Categories**.
2. You may further narrow down the list of extensions by checking one or more **Features**. This includes narrowing the focus to those created by Tableau (Salesforce), extensions that are free instead of paid, extensions that are sandboxed, or extensions that are allowed to work with Tableau Public.



Note

Salesforce acquired Tableau in 2019, which is why you often see the label **Tableau** (Salesforce) or **Tableau + Salesforce**.

3. If you wish to use your own extension or one that has not been published to Tableau Exchange, then click **Access Local Extensions**. We'll use this in the *Creating an extension* section later in this chapter.
4. If you have any **Recently Used** extensions, they will be shown here for quick access.
5. The list of extensions below **Browse Dashboard Extensions** shows you the name of each extension, the author, a brief description, and the number of uses. You may select an extension by clicking it.

To continue this example, check the **Built by Tableau + Salesforce** feature on the left and then locate and click the extension named **Image Map Filter** in the list. Upon selecting a feature, you'll be shown a much longer description and specifications, like this:

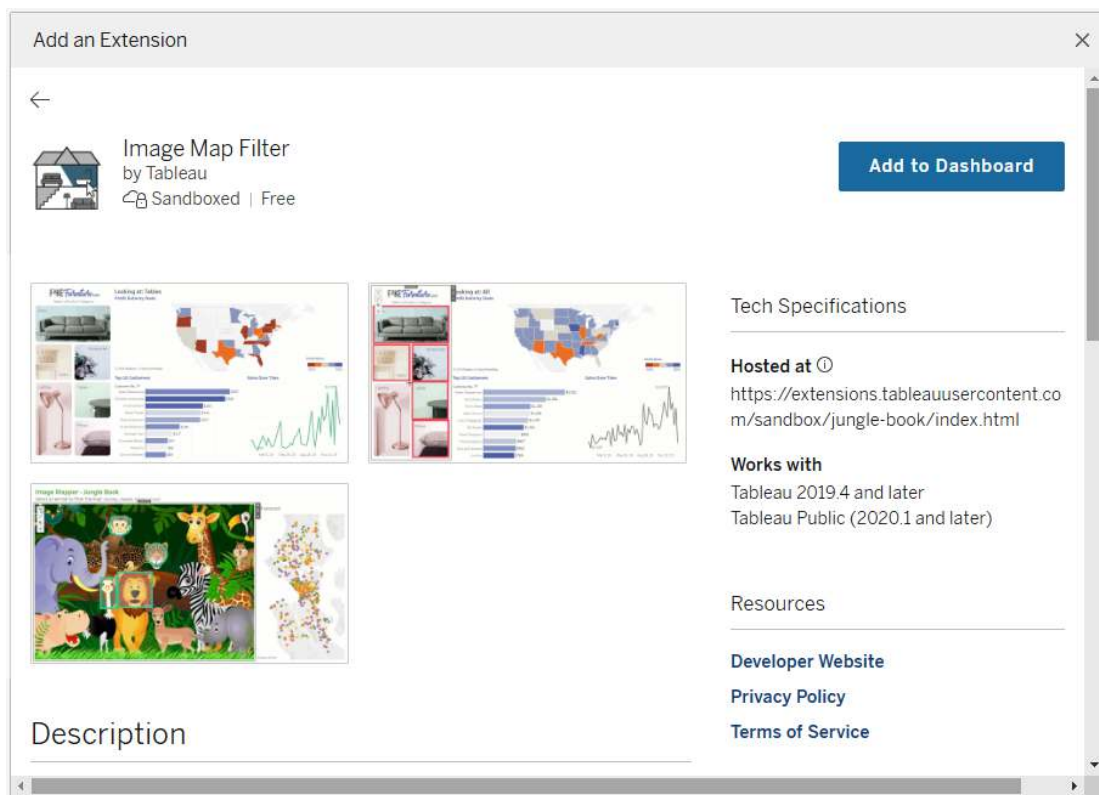


Figure 16.4: The description and specifications for the Image Map Filter dashboard extension

Take a moment to read the description and tech specifications. If you selected the wrong extension, or realize the selected extension does not meet your needs, you can close the window. Otherwise, click the **Add to Dashboard** button to add the extension to the dashboard.

Note



Always double-check the tech specifications to ensure compatibility with your version of Tableau, the URL where the extension is hosted, and for any security or legal documentation. Because dashboard extensions potentially have access to the data shown in your dashboard, make sure you trust the organization that developed the extension and that you don't have any legal or security concerns about adding the extension to your dashboard. Your organization's Tableau Server administrator has the option to enable or disable extensions on Tableau Server. You'll want to know your organization's policies prior to designing dashboards with extensions.

When you first add an extension to a dashboard, you'll often be prompted to configure the extension. In this example, the configuration screen should show by default. If it does not, or if you need to configure the extension later, you can use the caret dropdown of the extension object to access the configuration options. In this example, the configuration screen looks like this:

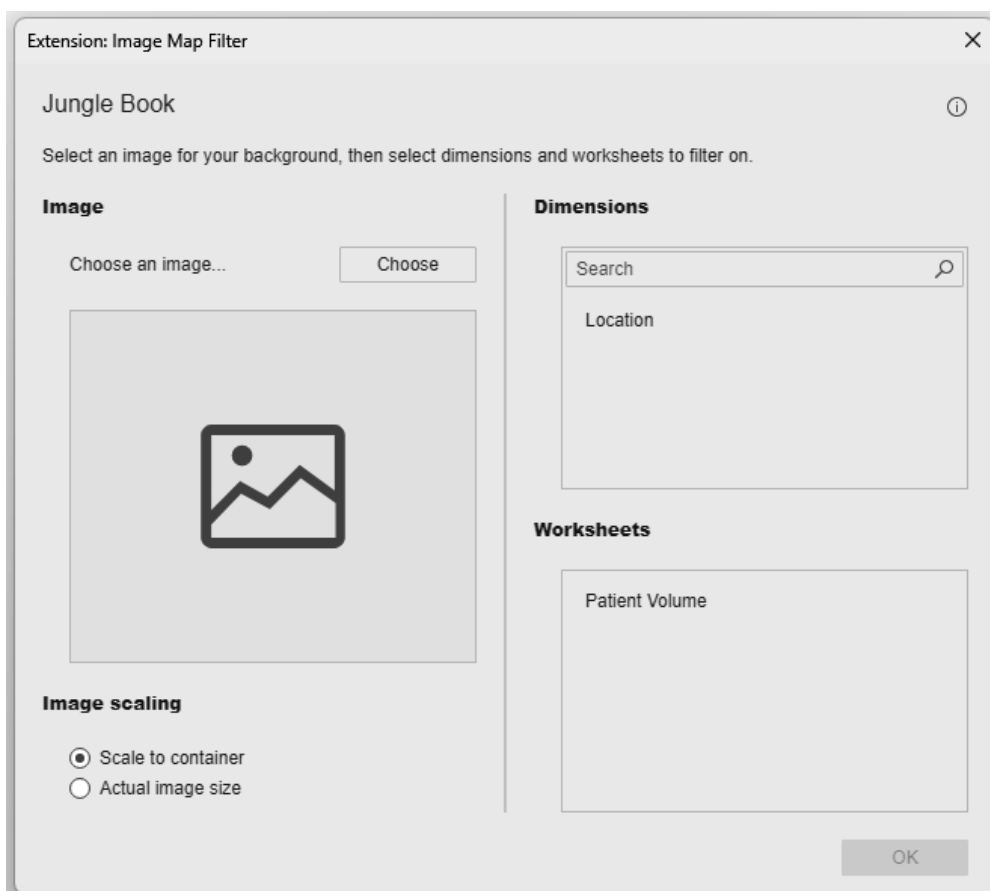


Figure 16.5: The configuration screen for the Image Map Filter dashboard extension

Each extension will have different configuration options, and some may not have any at all. In this case, perform the following steps to complete the configuration:

1. Under **Image**, click **Choose** and select `Hospital.png` located in the Chapter 16 directory.
2. Leave the **Scale to container** option selected. This will allow the image and the clickable hotspots to resize based on the container in the dashboard.
3. As **Patient Volume** is the only worksheet on the dashboard and it only contains a single dimension, **Location**, you will only need to select those items under **Dimensions** and **Worksheets**. Selecting these specifies that the image will filter the selected worksheet using the selected dimension. On more complex dashboards, you would need to take care to match up the correct sheets and dimensions.
4. Click **OK** to set the configuration options.

The dashboard will now look like this:



Figure 16.6: The dashboard with the configured dashboard extension

Notice the toolbar on the extension. This allows you to switch between filtering, drawing, selecting/editing, and moving. Continue the example with the following steps.

5. Using the drawing icon, select a rectangle shape.
6. Draw the square around an individual room.
7. Match the room up with the correct filter option.

8. Steps 5 through 7 are illustrated in *Figure 16.6*:

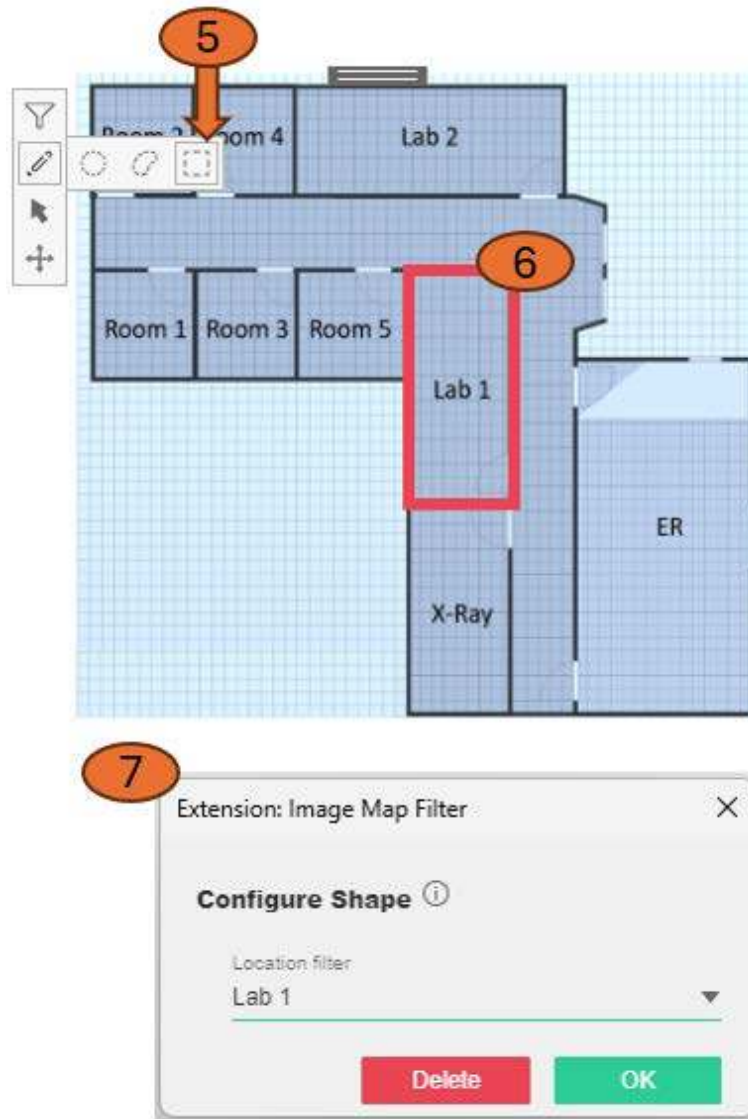


Figure 16.7: Drawing a shape around each room and matching them with the corresponding filter option

9. Repeat *steps 6 and 7* for each room until all rooms in the floor plan image have been matched with all the filter options.
10. Once all shapes are mapped, switch to filtering by clicking on the funnel icon:

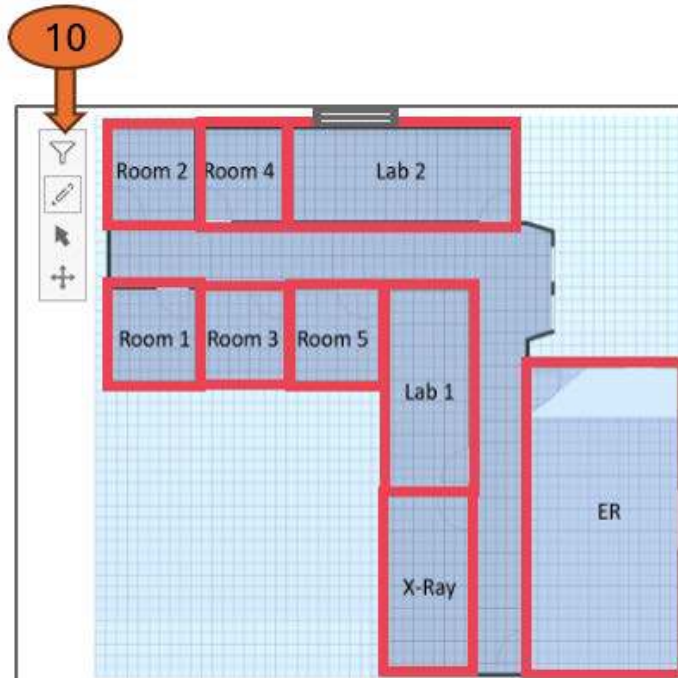


Figure 16.8: Switch to filtering once all shapes are mapped

11. Test the extension by clicking on various rooms. You should see the patient volume chart update appropriately. You have extended the capability of your dashboard and given your end users a new way of interacting with the dashboard that, in many ways, is more intuitive and fun than selecting a location from a dropdown.
12. New extensions are added often, so continue to explore and discover new ways of extending the functionality of your dashboards!

But what if you need unique capabilities that go beyond the extensions Tableau or third parties have created? You might consider creating your own extensions – and that’s what we’ll consider next!

Creating an extension

In some cases, you may have specific requirements that are not met by a third-party extension. In these cases, you may wish to create your own extensions. For this example, we’ll put together an extension that dynamically displays an image based on the value of parameters.



Completing this example will require downloading, installing, and configuring several components, which are detailed in the *Setting up your environment* section.

In *Chapter 10, Advanced Visualizations*, we saw the URL Image type, which allows us to use URLs in our data to show images in row or column headers. In this example, we'll create a dashboard extension that allows us to do something similar but extends the capabilities a bit by also allowing us to exactly control the image size.

As we've seen, using a Tableau dashboard extension will require some knowledge of HTML and JavaScript, as well as an understanding of the Tableau Extensions API. This example will provide a broad overview and you will be able to follow along at a high level even without expertise. In fact, all the code is already written as the goal is not to teach you the details of any coding. Instead, the goal is to show you all the pieces, allow you to put it all together, and offer explanations along the way. Of course, you may want to alter and extend the code as much as you want!

Setting up your environment

If you wish to follow along with the example, you will need to install various components and configure your environment to host and run custom dashboard extensions. If you choose not to configure your environment, you will not be able to use the custom extension. However, you may still wish to read through the remainder of this section to better understand how extensions work and how you might create one in the future.

Note



To replicate the example in this section and run a Tableau dashboard extension, you'll need to install several components. **This includes downloading, installing, and running a local web server**, as described in this section. It is highly recommended that you read through the online documentation completely and it is assumed that you will have installed and configured all the required components following that documentation, which is located here: https://tableau.github.io/extensions-api/docs/dashext/trex_getstarted.

You will additionally find some of the installation files here: <https://tableau.github.io/extensions-api/docs/installation>.

Using the link in the preceding information box, follow the online instructions to download and install the following components:

- **Git:** A version control system that will, among other things, allow you to install and start a local web server.
- **Node.js and npm:** A JavaScript runtime and package manager that will download, install, and run the local web server.

- **The Tableau Extensions API SDK:** A software development kit for the Tableau Extensions API that will provide you with the resources required for extension development. It will also contain the sample code that will give you useful examples.

**Tip**

You may have to run Git Bash as an administrator on some systems.

- Once you have installed and configured Git, Node.js, and the Tableau Extensions API, you will additionally need to copy the ShowImage directory and files from the Learning Tableau\Chapter 16 directory to the extensions-api-main\Samples\Dashboard directory where you installed the API SDK.

The files and code that define the dashboard extension

Look at the files contained in the extensions-api-main\Samples\ShowImage directory. We'll examine some key details surrounding these files, but for now, you'll see the following:

- **ShowImage.trex:** A manifest file that tells Tableau where the extension is hosted and also includes information such as a description, the author, and the organization that created the extension.
- **ShowImage.html:** An HTML file that, in this example, will display an image to the user. In general, the HTML file provides an interface to the user – either showing output or capturing input. The HTML file will contain a link to a .js (JavaScript) file.
- **ShowImage.js:** A JavaScript file that, in this example, will read values from parameters on the dashboard and dynamically interact with objects in the HTML file to display an image. In general, the JavaScript file will give you programmatic control via the Dashboard Extensions API (to access objects and values on the dashboard) and via the HTML file (to get input and show output to the user in the extension control).

**Tip**

You may wish to use a simple text editor (such as Notepad), a more advanced text editor (such as Notepad++), or even a development platform (such as Visual Studio) to view and edit code files.

Let's examine each of these files in a bit more detail to see what they contain and how they function.

The manifest file

We'll start with the manifest file, ShowImage.trex. The manifest file is an XML document that contains everything needed to tell Tableau where the extension is hosted and give users information such as a description and the author of the extension. Open the file in an editor of your choice if you wish to see the entire file. Here, we'll only examine especially relevant snippets.

First, notice the description and author tags:

```
<description>Show Image Based on Parameters</description>
<author name="Joshua N. Milligan" email="jmilligan@vizpainter.com"
organization="VizPainter" website="https://www.vizpainter.com"/>
```

These tags describe the author, email, organization, and website associated with the extension. You'll want to set the correct details for any extension you create.

Next, consider the source-location tag:

```
<source-location>
<url>http://localhost:8765/Samples/ShowImage/ShowImage.html</url>
</source-location>
```

The URL contained in this section of code is especially important as it points to the HTML file that will be hosted in the dashboard. In this case, the URL points to the local web server installed in our environment. Assuming you set up your environment according to the defaults, your location should be the same. Ultimately, you may decide to host your extension on a dedicated server in your organization or in the cloud. At that point, you will need to update the manifest to point to the new location.

The HTML file

In this example, the HTML document is named `ShowImage.html`. This example uses a very simple HTML document with some links to external JavaScript files and an image with a blank source, which will be updated via the JavaScript. Examine the full file in an editor and then consider the most important snippets:

```
<script src="../../lib/tableau.extensions.1.latest.js"></script>
```

This reference points to the latest version of the Extensions API. Assuming you configured and started the web server based on the location of your Tableau Extensions API download, your reference should not need to be altered.

Next, we have a reference to the JavaScript file we'll use for our code:

```
<script src="./ShowImage.js"></script>
```

This line references the `ShowImage.js` file, which contains all the code we'll need to update the image based on parameter values in the dashboard.

Finally, within the body of the HTML, there is one final tag:

```
<img id="imgImage" src='' />
```

This defines the image and initializes it with a blank source. We'll update this object in the JavaScript code based on parameter values in the dashboard.

The JavaScript file

Finally, let's consider some of the relevant sections of the JavaScript file, `ShowImage.js`. This is a very simple JavaScript file that makes use of the Tableau Extensions API to read parameter values and apply the appropriate values, based on the name of the parameter, to the image in the HTML document.

First, we have a function to update the image based on the parameter values:

```
function updateImage (p) {
  if (p.name=='ImageUrl'){
    document.getElementById("imgImage").src = p.currentValue.formattedValue;
  }
  else if (p.name=='ImageWidth'){
    document.getElementById("imgImage").style.width = p.currentValue.
formattedValue +      "px";
  }
}
```

This function takes in a parameter object, `p`, and then checks the name of the object. If the parameter's name is `ImageUrl`, then it sets the `src` attribute of the HTML image to the value of the parameter. If the parameter's name is `ImageWidth`, then it sets the `width` style to the parameter value with "px" (pixels) appended.

This example highlights the fact that many extensions will require certain naming conventions to be used in a dashboard. Specifically, this extension requires a parameter named `ImageUrl` to set the URL for the image. The extension will function without a second parameter named `ImageWidth`, but only a parameter with that exact name can be used to set the image width.

Next, we have a function that is called whenever a parameter value changes:

```
function onParameterChange (parameterChangeEvent) {
  parameterChangeEvent.getParameterAsync().then(function (param) {
    updateImage(param);
  });
}
```

We'll see next how the `onParameterChange` function is set as an event listener. For now, understand that the function is called whenever a value of a parameter changes in the dashboard. When a parameter's value changes, the previously examined `updateImage` function is called with the changed parameter as an argument.

Finally, notice the `document.ready` function, which will be called after the HTML document has loaded, giving us a chance to initialize everything we need:

```
$(document).ready(function () {
  tableau.extensions.initializeAsync().then(function () {
    tableau.extensions.dashboardContent.dashboard.getParametersAsync().
then(function (parameters) {
```

```
parameters.forEach(function (p) {  
    p.addEventListener(tableau.TableauEventType.ParameterChanged,  
        onParameterChange);  
    updateImage(p);  
});  
});  
});  
});
```

When the HTML document is loaded and ready, this block of code will be executed. First, the Tableau extensions initialization is called, after which `tableau.extensions.dashboardContent.dashboard.getParametersAsync()` gets an array of all parameters in the dashboard. For each of those parameters, the `onParameterChange` function is added as an event listener and the `updateImage` function is called, which will update either the image source or the width, depending on which parameter is passed in.

All of these functions work together such that when the extension is first loaded in the dashboard and whenever a parameter value changes, the image will be updated. Let's see it in action!

The dashboard extension in action

In the Chapter 16 Complete and Chapter 16 Starter workbooks, you'll find a **Learning Tableau Editions** tab that is ready for our ShowImage extension.



Tip

Make sure you have started the local web server, so the extension content is available for the Tableau dashboard.

Drag the **Extension** object into the blank area of the dashboard, and on the **Add an Extension** pop-up window, select **Access Local Extensions** in the lower-left corner:

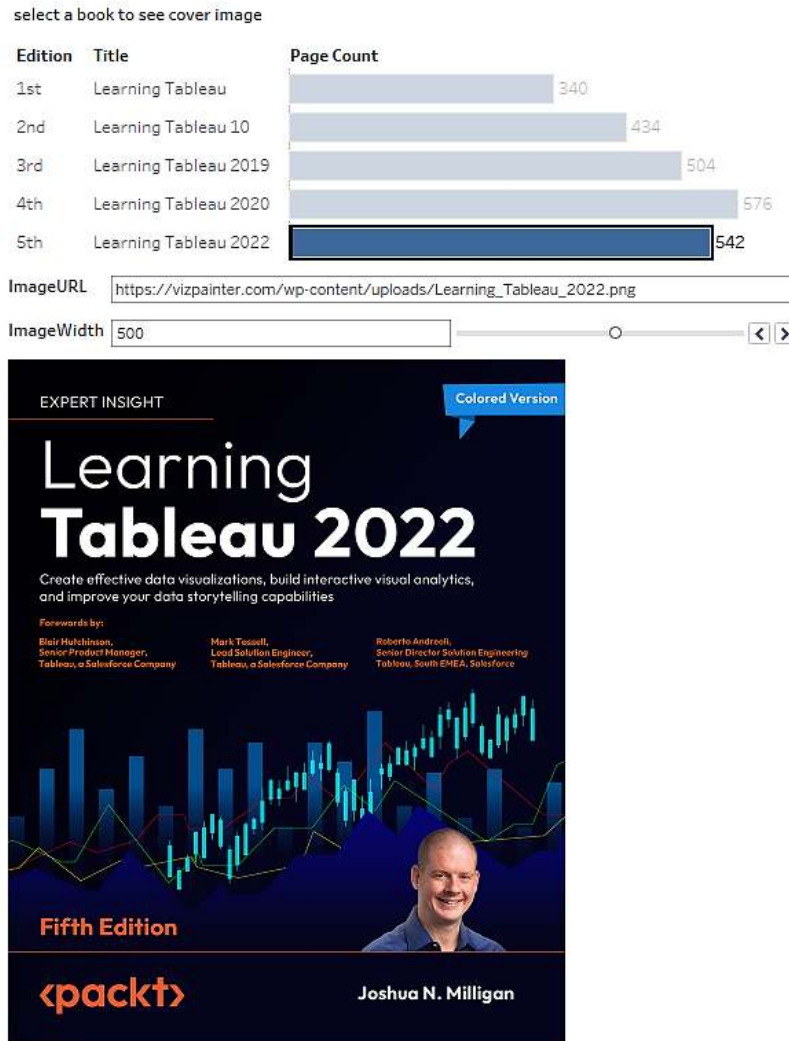


Figure 16.9: Adding a local extension

You will then be asked to locate the appropriate `.trex` manifest file. Browse to the `ShowImage.trex` file located in the Tableau Extensions API directory structure (`extensions-api-main\Samples\Dashboard\ShowImage\`). If you are prompted to **Allow the Extension**, select **OK**.

The ShowImage folder is included in the code files, but make sure you are working with the copy you made in the extensions-api directory structure. Reference the instructions in the *Setting up your environment* section if necessary.

At this point, the extension should be active and working. Test it by selecting one of the editions of *Learning Tableau* to see an image of the cover:



Prior to the extension working, you will need to make sure all necessary tools (such as the Tableau Extensions API SDK) are installed, your environment is configured, and the extension is pointing to the ShowImage.trex file based on your local directory structure.

Figure 16.10: The dashboard with the embedded ShowImage extension

As you examine the dashboard, you'll notice the two parameters that match in name with what is expected by the JavaScript file: **ImageURL** and **ImageWidth**. The view triggers a parameter action that updates the value of the parameter, and that in turn triggers the JavaScript function that updates the image.

In fact, any change of the parameter value will trigger the image update function, so you could type in the URL of an image or adjust the width slider manually, which would also cause the image to update.

You've just put together and used a simple extension! We've only scratched the surface of what's possible!

Dashboard extensions aren't the only possible way to extend Tableau's functionality. Next, we'll consider analytics extensions, which, like dashboard extensions, provide unlimited potential for what's possible.

Tableau analytics extensions

Tableau analytics extensions allow you to use a connection from Tableau to an external analytics service or platform to perform complex operations and use the results in calculations, visualizations, and dashboards. Tableau analytics extensions allow you to connect to analytics services, such as:

- TabPy: Tableau's Python server.
- Rserve: An open-source R server.
- Einstein Discovery: Salesforce's predictive/AI analytics engine.
- Analytics Extensions API: An API allowing you to connect to numerous analytics tools and platforms. This allows you to use connectors you create programmatically or prebuilt connectors. For example, you can connect to MATLAB using the Analytics Extensions API.

Much like dashboard extensions required a separate web server to host the extension, analytics extensions require a separate service or platform. For example, to run Python scripts in Tableau, you'll need to connect to an instance of TabPy. If you want to run R scripts, you'll need to have a connection to an Rserve server. In general, Tableau will execute analytics extensions as illustrated in the following diagram:

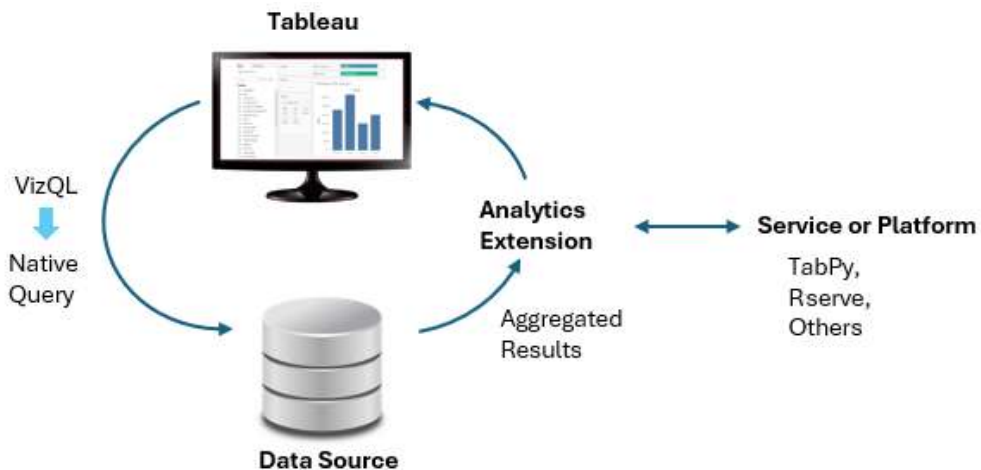


Figure 16.11: Analytics extensions are implemented based on aggregated results from the data source

In this section, we'll look at an example of using a Python script with TabPy to illustrate how to use analytics extensions. In the next section, we'll look at Einstein Discovery—an AI-powered analytics extension that integrates with CRM Analytics, which deserves special consideration.

Scripting with Python

Both R and Python are programming languages that allow the use of advanced analytics, including statistical functions, predictive models, and AI. Python in particular has gained strong popularity among data scientists, analysts, and others who work with data. In this section, we'll use a very simple Python script in Tableau. Of course, you'll be able to accomplish much more as you progress in your abilities!

Completing this example will require downloading, installing, and configuring Tableau's Python server, TabPy. The next section, *Setting up TabPy*, will point to online resources and documentation that will guide you through the process.

While this example highlights the ability to use Python scripts in Tableau, the process for using R scripts is very similar. Instead of installing TabPy, you would install Rserve (an R server). The connection process and the process of writing scripts mirrors the example given in this section.

We'll briefly look at setting up TabPy as the engine for processing the Tableau scripts and then implement a simple Python script to clean up some data in our dataset.

Setting up TabPy

For this example, we'll assume an installation on your local machine, but the TabPy service may be installed in a location that's accessible from Tableau Desktop (and Tableau Server, if you will be using it).

Note



Follow the online documentation to set up TabPy. The basic commands for installing the prerequisites and TabPy are listed in this section, but be aware that the steps may change in future versions and the online documentation is the most complete and authoritative source. You will find documentation on installing and configuring TabPy here: <https://tableau.github.io/TabPy/docs/server-install.html>.

Prior to installing TabPy, you will need to install Python, which is available from python.org.

If you installed Git as described in the *Setting up your environment* section of *Creating an extension*, you'll be able to issue the necessary commands to install and run TabPy from a Git Bash prompt. If you did not install Git previously, you'll need to do so to follow this example.

The basic commands to set up TabPy are:

- To install pip, a package installer for Python, issue this command:

```
python -m pip install --upgrade pip
```

- To install TabPy, issue this command:

```
pip install tabpy
```

- To run TabPy, issue this command:

```
tabpy
```

**Note**

Once TabPy is running, note the port number indicated in the Git command-line window after issuing the `tabpy` command.

From Tableau Desktop, use the menu to select **Help | Settings and Performance | Manage Analytics Extension Connection....**

A dialog box will ask you to **Select a connection type**:

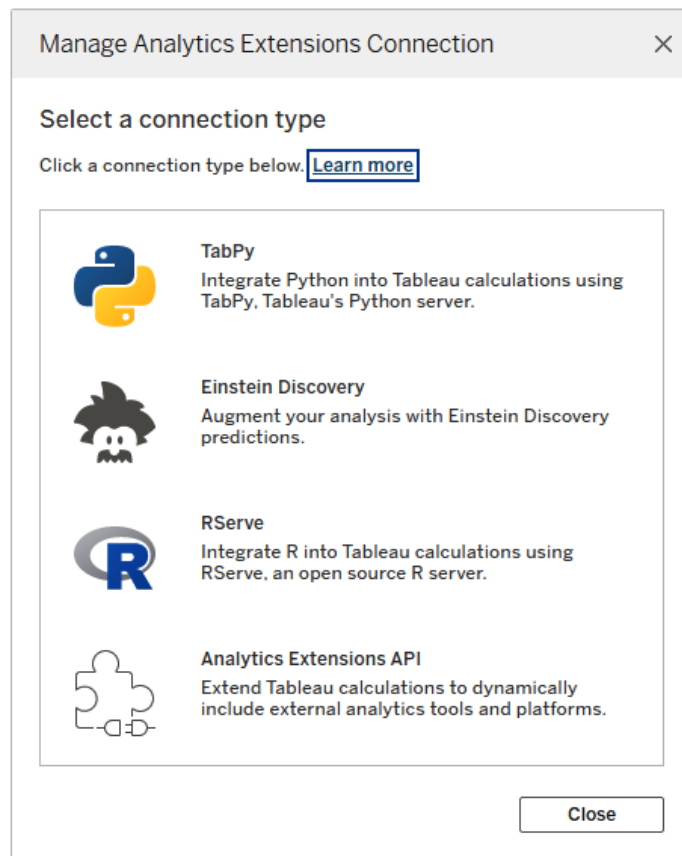
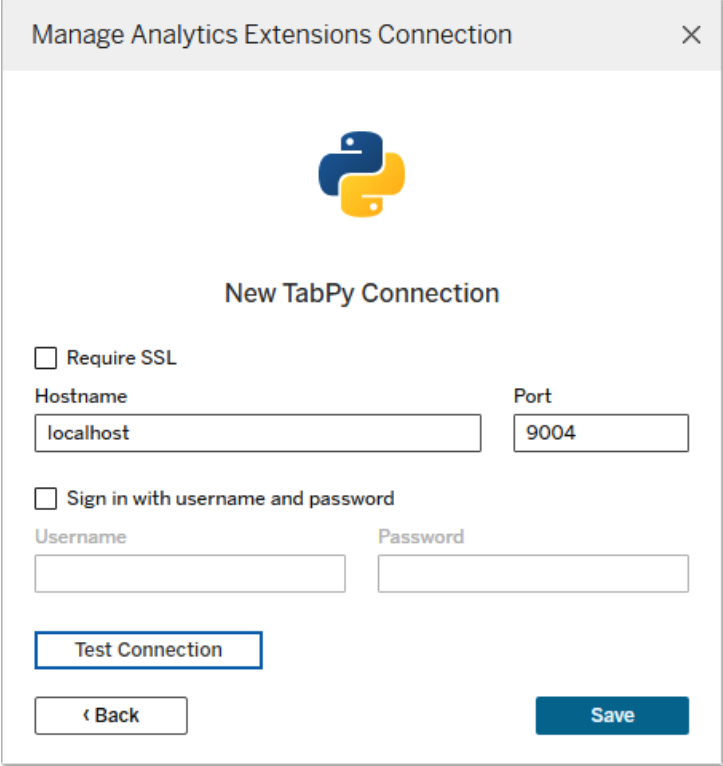


Figure 16.12: The Manage Analytics Extensions Connection dialog allows you to configure options for various analytics extension types

Select **TabPy** and then enter a **Hostname** of `localhost` and a **Port** of `9004` (or the port number on your system, if it differs):



The screenshot shows a dialog box titled "Manage Analytics Extensions Connection" with a close button (X) in the top right corner. Inside the dialog, there is a Python logo and the text "New TabPy Connection". Below this, there are two checkboxes: "Require SSL" (unchecked) and "Sign in with username and password" (unchecked). Under "Require SSL", there are two text input fields: "Hostname" with the value "localhost" and "Port" with the value "9004". Under "Sign in with username and password", there are two empty text input fields labeled "Username" and "Password". At the bottom, there are three buttons: "Test Connection" (highlighted with a blue border), "Back" (with a left arrow), and "Save" (in a dark blue box).

Figure 16.13: Setting up a new TabPy connection with hostname and port information

Once you have clicked **Test Connection** and verified that everything is working, click **Save**. You may close the dialog.

With TabPy installed and running and a connection established in Tableau Desktop, we're now ready to see a Python script in action!

Using a Python script in Tableau

To follow along with the example, navigate to the Episode Ratings sheet in the Chapter 16 Complete or Chapter 16 Starter workbook.

Note



When you first open a workbook containing a connection to an external service via analytics extensions, you'll be prompted with a security warning. As with dashboard extensions, you'll want to be aware that data may be transferred to the external service. Make certain you trust the workbook and its source.

In the workbook, you'll find a simple view of episodes of the original Star Trek series along with IMDb ratings:

Title	AVG(IMDb Rat..)
the cage	7.7
the changeling	7.7
the city on the edge of forever	9.2
the cloud minders	7.1
the conscience of the king	7.2
the corbomite maneuver	8.1
the deadly years	7.4
the devil in the dark	8.4
the doomsday machine	8.8
the empath	6.6
the enemy within	7.7
the enterprise incident	8.5
the galileo seven	7.7
the gamesters of triskelion	7.0
the immunity syndrome	7.6

Figure 16.14: Episodes of Star Trek along with IMDb ratings

One thing you may notice is that the episode titles are all lowercase. For a great user experience for those using visualizations or dashboards based on this data, it would be nice to display the titles in title case.

We'll use a Python script to transform the string to title case. It would be possible to write a complex calculation using regular expressions in Tableau to accomplish this task. However, the purpose of this example is to introduce you to writing a Python script and integrating it into your Tableau workflow. Beyond this simple example, there are many data cleansing and analytics tasks that Python is capable of handling that Tableau alone is not.

The Python script itself is contained in a single calculated field. If you are viewing the Complete workbook, find the calculated field named Episode and select **Edit**. If you are using the Starter workbook, create a calculated field named Episode. The contents of the calculation look like this:

```
SCRIPT_STR(
"
import re

exceptions = ['a', 'an', 'the', 'at', 'by', 'for', 'in', 'of', 'on', 'to',
'up', 'and', 'as', 'but', 'or', 'nor', 'with', 'is']
```

```

if _arg1 is None:
    return None

title = _arg1[0]

word_list = re.split(' ', title.lower())
capitalized_title = [word_list[0].capitalize()]

for word in word_list[1:]:
    capitalized_title.append(word if word in exceptions else word.capitalize())

return ' '.join(capitalized_title)

",

MIN([Title]) )

```

This calculation defines a Python script that is executed via TabPy to return a title case version of a string value that is passed into the script. We'll dissect the Python script itself, but first, let's note some important aspects of the `Script_` functions in Tableau:

- The calculated field uses the `SCRIPT_STR()` function. The `Script_` functions in Tableau allow you to pass a script to an external engine and return a result. The exact function you use depends on the data type of the result returned from the script. In this case, we expect a string result. In other cases, you may need to use `SCRIPT_BOOL`, `SCRIPT_INT`, or `SCRIPT_REAL` if you expect a Boolean, integer, or real number result, respectively.
- The `Script_` functions all take two or more arguments. The first argument is a string containing the script that will be executed via the analytics extension. The second, and optionally third, fourth, fifth, and so on, are values you wish to pass into your script. These values will be referenced in the script as `_arg1`, `_arg2`, `_arg3`, and so on. They will be passed in as objects, so you'll most often obtain the value of the object as an attribute using code such as `_arg2[0]`.
- The `Script_` functions are treated as table calculations in Tableau. This means all values you pass into the script will be aggregations at the level of detail defined by the view. This is why `MIN([Title])` is used rather than just `[Title]`. Furthermore, you may need to specify the scope and direction of the table calculation to ensure the values are passed into the script as you expect.

Now, let's turn our attention to the Python script itself to understand how it works. We'll walk line by line through the code, starting with the first:

```
import re
```

This line imports Python's regular expression library, which we will use to parse the string.

We'll also define a list of exception words that should not be capitalized in the middle of a title:

```
exceptions = ['a', 'an', 'the', 'at', 'by', 'for', 'in', 'of', 'on', 'to',  
             'up', 'and', 'as', 'but', 'or', 'nor', 'with', 'is']
```

The `exceptions` variable contains the list of exception words. The next lines exit the function if `Title` is null:

```
if _arg1 is None:  
    return None
```

You'll recall that we passed in `Title` as the first argument. If it is `Null`, the Python script will also return `Null`.

Next, we'll assign the value to a Python variable named `title`. This is primarily for convenience and readability:

```
title = _arg1[0]
```

Notice that we used `_arg1[0]` because we want the string value of the first (and, in this example, only) argument passed into the script.

Next, we split the title into each individual word and store it as a list in a variable named `word_list`:

```
word_list = re.split(' ', title.lower())
```

You'll notice that we use the regular expression library we imported in the first line of code to accomplish the split. We also use the `lower()` function to ensure that all words in the title are lowercase prior to implementing any capitalization logic.

Next, we define a variable that will ultimately contain the list of title case words that will be returned to Tableau:

```
capitalized_title = [word_list[0].capitalize()]
```

We initialize the list with the first word capitalized, because the first word of every title is capitalized without exception.

Next, we loop through the remaining words in the title and capitalize them so long as they are not an exception word:

```
for word in word_list[1:]:  
    capitalized_title.append(word if word in exceptions else word.capitalize())
```

Each correctly cased word is appended to the `capitalized_title` list.

Finally, after looping through all the words in the title, we return the title-cased list as a single string where all the words are joined back together with a space as a separator:

```
return ' '.join(capitalized_title)
```


The end result is that Tableau will pass in an episode title and the Python script, executed via an analytics extension to TabPy, will return the correct title case as a result:

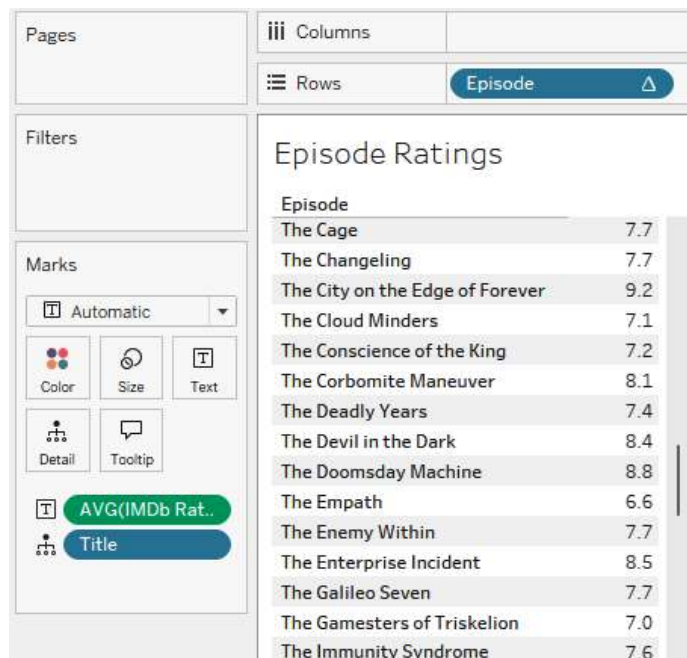


Figure 16.15: Episodes of Star Trek along with IMDb ratings

You'll notice that Episode is a table calculation indicated by the delta symbol on the field and that the original Title is still in the view (on Detail in the Marks card). Title must be in the view so that it can be passed into the Python script, and Episode is computed at the cell level so that each individual title is passed in.

Beyond this simple example, there are so many incredible things you can do with Python (and similarly R). As you learn and grow, you'll uncover a wealth of statistical tools, predictive libraries, and the ability to accomplish incredible feats of analytics.

We'll turn our attention to one final example using Python in table extensions.

Table extensions

We just walked through an example of creating a new field with a Python script. With table extensions, you can add entirely new tables to your data model and leverage the full power of Python to transform, cleanse, and analyze your data!

Table extensions allow you to add tables of data to your data model. Unlike most of the tables we've seen so far, which have come from files or databases, these files are created based on scripts.

The best way to see this is in action through an example. Table extensions can leverage the same analytics extensions we looked at previously (such as TabPy or Rserve). For the following example, you'll need to ensure you have TabPy set up and running as we did previously.

Adding a table extension

One thing you may notice about the Star Trek Episodes dataset is that it contains a **Review** of each episode. One thing we might want to do with freeform text is to perform sentiment analysis.



Note

Sentiment analysis processes natural language text and applies a score or determination as to whether the sentiment (feeling or emotion) behind the text is positive, neutral, or negative.

We will accomplish this sentiment analysis and supplement our data with a new sentiment score value for each record.

Start by editing the **Star Trek Episodes** data source:

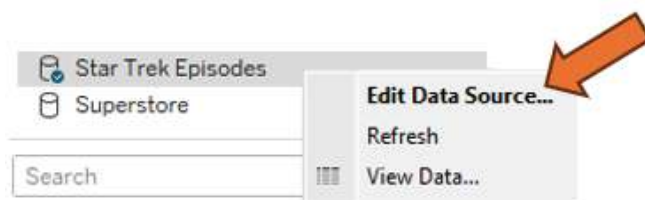


Figure 16.16: Edit the data source

For this example, we're not creating an entirely new table of data. Instead, we're going to extend the existing table. So, we'll first remove the existing table using the drop-down option on the table object in the pane:

☞ Star Trek Episodes

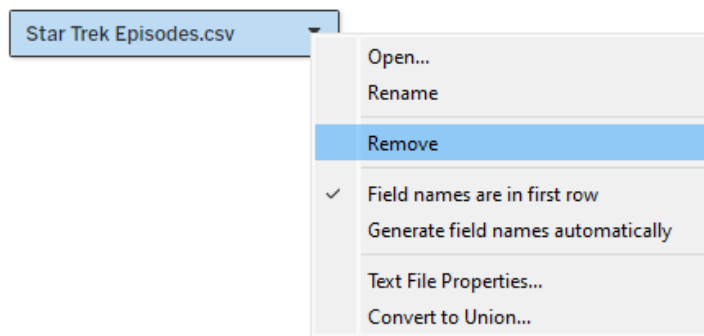


Figure 16.17: Remove the existing table

Next, we'll add a **New Table Extension** by dragging and dropping the object onto the blank pane:

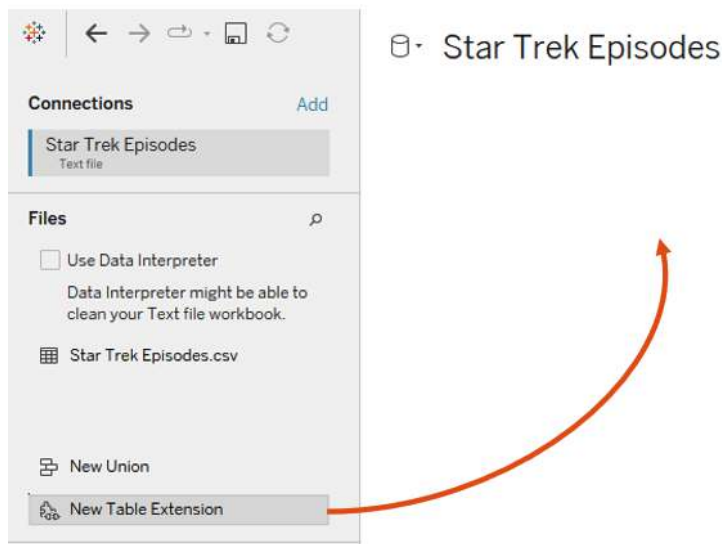


Figure 16.18: Adding a new table extension

That will give us a new interface for setting up the **Table Extension**:

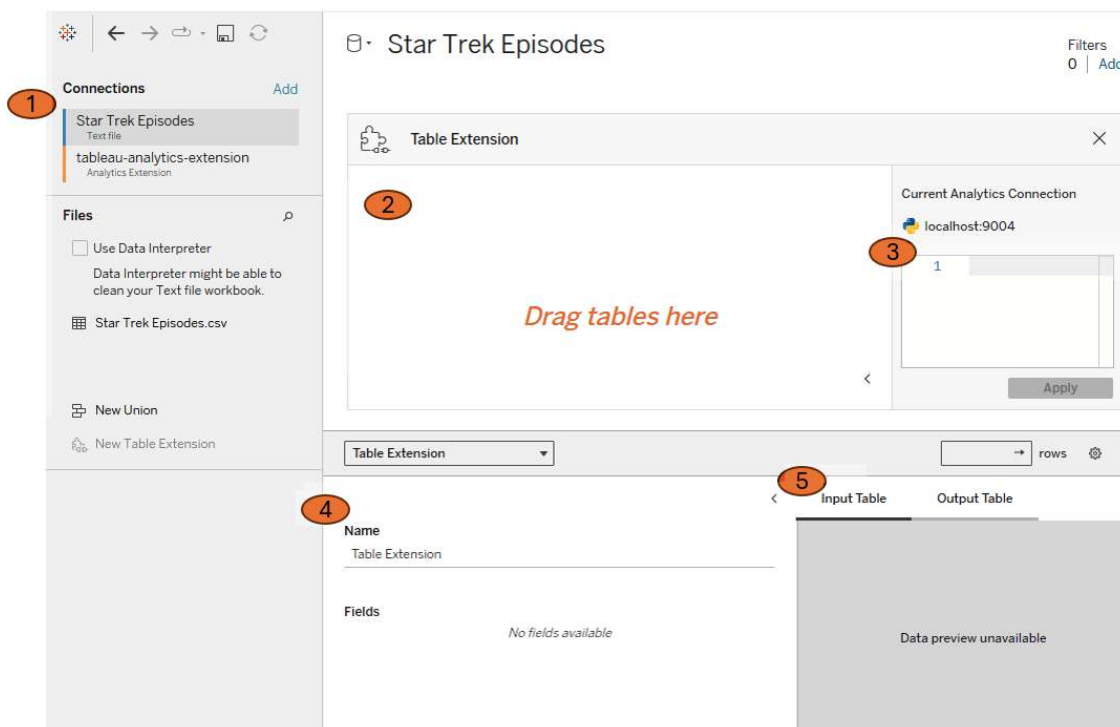


Figure 16.19: Components of the Table Extension interface

The **Table Extension** interface consists of the following:

1. **Connections**, which includes your original connection along with the analytics-extension connection.
2. The **Table Extension** drop pane, where you can drop tables from one or more connections. These tables will form the input table, which is the base dataset that you can modify using your script.
3. **Current Analytics Connection** gives you a place to write your script. It shows the current connection (in this case, TabPy running on localhost:9904).
4. A place to give your table extension a **Name**. This will become the name of the table defined by your extension. It will also show you the fields available for your script.
5. **Input Table** and **Output Table** are two tabs that allow you to see a preview of records from the input table, as well as the output returned by your script.

We'll start with the **Star Trek Episodes.csv** table as our input. Drag it onto the **Table Extension** pane:



Figure 16.20: Adding an input table to the Table Extension pane

Next, we'll include the following script in the analytics connection:

```
import pandas as pd
from textblob import TextBlob

def get_sentiment_score(review):
    analysis = TextBlob(review)
    return analysis.sentiment.polarity

df = pd.DataFrame(_arg1)
df['Sentiment Score'] = df['Review'].apply(get_sentiment_score)

return df.to_dict(orient="list")
```

There are several components to this script. First, we import a couple of Python libraries:

- **pandas** allows us to handle the dataset from the input table.
- **TextBlob** gives us an easy way to perform the sentiment analysis.

Next, we define a function called `get_sentiment_score()`, which takes in a single argument, **review**. The score assigned will be from -1 to 1, indicating negative, neutral, or positive along that range.

This line in the script takes the input table (passed into the script as a list object named `_arg1`) and transforms it into a DataFrame object:

```
df = pd.DataFrame(_arg1)
```

The next line adds a new column named **Sentiment Score** and assigns the value of the **Review** column with the `get_sentiment_score()` function applied:

```
df['Sentiment Score'] = df['Review'].apply(get_sentiment_score)
```

Finally, the script returns the table by translating the DataFrame back into a list:

```
return df.to_dict(orient="list")
```

Click on **Apply** under the script and then update the preview for the output table to see how the output contains all of the original columns of the original table along with the new Sentiment Score column:

The screenshot shows the Tableau interface with a script editor and a table preview. The script defines a function `get_sentiment_score` that uses `TextBlob` to analyze the sentiment of a review. It then applies this function to a DataFrame created from the input table, adding a new column 'Sentiment Score'. The preview shows the 'Output Table' with the original columns (IMDb Rating, Review, Title) and the new 'Sentiment Score' column.

#	IMDb Rating	Review	Title	Sentiment Score
Tableau_73_EOCECA92-12F7-4872-...	7.80000	I really enjoyed the premise o...	a piece of the action	-0.006250
Tableau_73_EOCECA92-12F7-4872-...	6.90000	While the episode tried to tac...	a private little war	0.300000
Tableau_73_EOCECA92-12F7-4872-...	8.10000	I enjoyed how the episode ex...	a taste of armageddon	0.125000
Tableau_73_EOCECA92-12F7-4872-...	8.20000	I thought the alien race we e...	all our yesterdays	0.125000
Tableau_73_EOCECA92-12F7-4872-...	8.70000	Spock's emotional journey re...	amok time	0.189286

Figure 16.21: Previewing the results of the table extension

When we switch back to the **Episode Rating** tab, we'll see the new Sentiment Score column in our data:

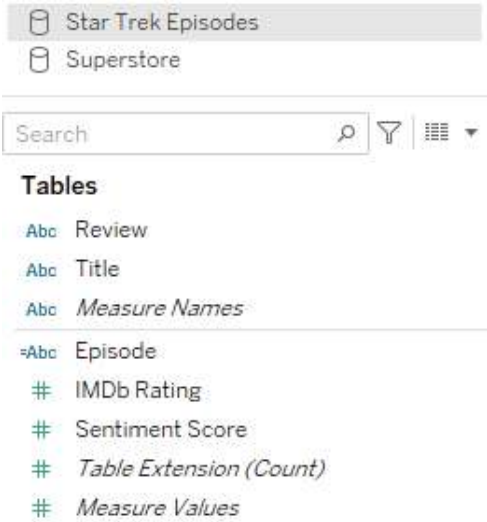


Figure 16.22: Sentiment Score is now part of the data

Notice that the **Sentiment Score** column is not a calculated field. Although it was created with script, it is a row-level value of the table and can be used as any other field. Here, for example, is a scatterplot showing the relationship between the IMDb rating and the sentiment score of the selected reviews:

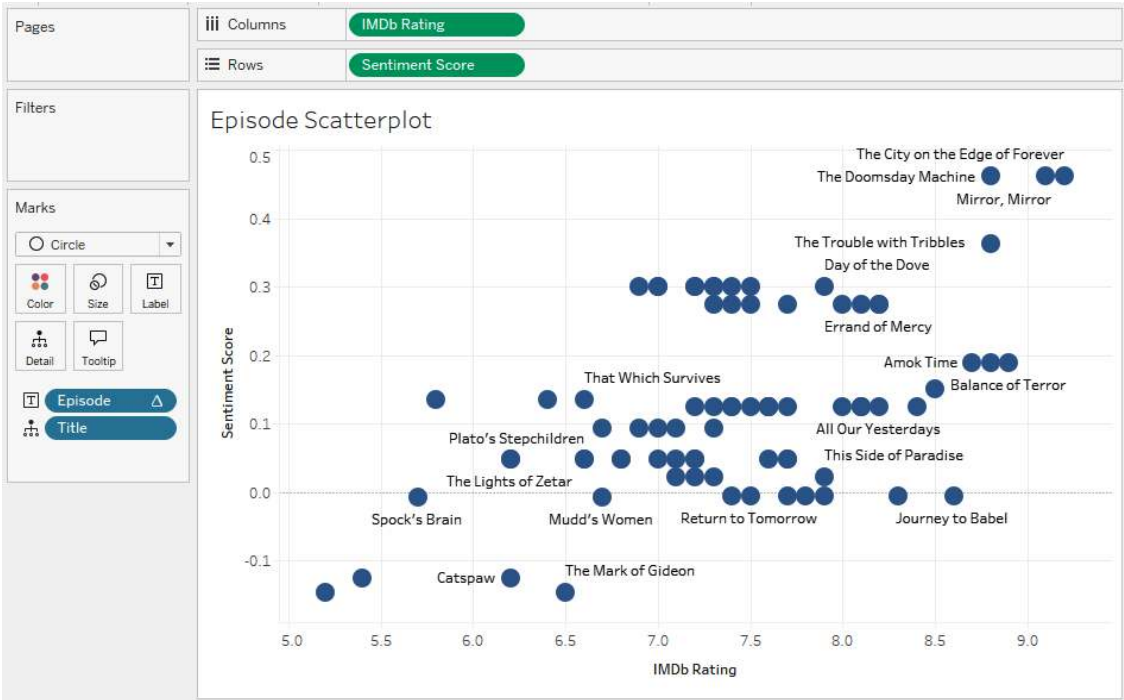


Figure 16.23: Scatterplot showing the relationship between Sentiment Score and IMDb Rating

Summary

The ability to extend Tableau is amazing! With dashboard extensions, you have an ever-growing library of Tableau/Salesforce and third-party extensions. These can extend the user interface, provide integration with backend systems and platforms, and enhance the user experience in many ways. We even looked at an example of writing your own dashboard extensions. The ability to write custom code and host it on your own servers gives you ultimate flexibility and control as you extend Tableau's capabilities.

In the next chapter, we'll turn our attention to AI and machine learning as we see how to leverage metrics, Tableau Agent, and Einstein Discovery and the ability to incorporate AI into your Tableau analyses.

Unlock this book's exclusive benefits now

Scan this QR code or go to packtpub.com/unlock, then search for this book by name.



Note: Keep your purchase invoice ready before you start.

17

AI-Assisted Analysis: Tableau Agent and Tableau Pulse

The past few years have seen the rapid development of AI technology. Breakthroughs in natural language processing, such as the rise of large language models, GPT, and sophisticated machine learning models, hold out the promise of automating and accelerating insight and action. Tableau has embraced the vision of integrating AI throughout the cycle of analytics in a way that gives analysts, business users, and decision-makers the ability to collaborate and more quickly reach data-driven decisions through the assistance of AI.

Tableau continues to innovate with AI and brings together multiple AI technologies. For example, **Tableau Next** is a new product offering from Tableau incorporating AI along with a common semantic layer that allows multiple ways of interacting with the data, sharing the analysis, and taking action through various workflows. Although it is beyond the scope of this book to cover Tableau Next, there are plenty of AI features and technologies to cover in the classic Tableau products.

Within Tableau Desktop and Tableau Cloud, AI innovations bring new ways of interacting with data. Alongside interactions familiar to long-time Tableau users, such as drag and drop Tableau has introduced new ways of interacting with AI, such as asking questions and receiving feedback in natural language. It is beyond the scope of this book to cover the entire platform (and it is rapidly evolving), but in this chapter, we will focus on two main components of the platform:

Tableau Agent (formerly Einstein Copilot): An AI assistant that enables natural language exploration of data, automation of certain tasks, and even the creation of some calculations.

Tableau Pulse: An interface built on top of Tableau that enables natural language understanding in your data and metrics and allows for guided drill-down into detailed insights. Tableau Pulse is generally available on Tableau Cloud.

Unlike most other chapters, this chapter does not have any sample workbooks (though sample data is included). Both Tableau Agent and Tableau Pulse run on Tableau Cloud. Tableau Pulse is generally available with a cloud license (either paid or trial), but Tableau Agent requires additional licensing. Additionally, specific examples would likely be difficult to replicate precisely as every session with generative AI may yield different results and flows of interaction. You may wish to simply read the chapter to gain an overview of AI in Tableau, or you may explore the features in this chapter using either a paid or trial license of the features.

**Note**

Please follow up with a Tableau representative for licensing information. Trial periods for Tableau products may be requested at <https://www.tableau.com/products/trial>.

In this chapter, we'll cover the two main AI components within the classic Tableau platform:

1. Tableau Agent
2. Tableau Pulse

**Note**

AI capabilities in Tableau are based on the **Einstein Trust Layer**, which guarantees privacy and security of your data. For more information, you may visit <https://www.salesforce.com/artificial-intelligence/trusted-ai/>.

We'll start by seeing how Tableau Agent can help us rapidly explore a dataset, automatically generate some visualizations, and aid us in the creation of calculations. Then, we'll explore Tableau Pulse and see the power of metrics.

Tableau Agent

Tableau Agent is much like other conversational AI assistants such as ChatGPT or Microsoft Copilot. What differentiates Tableau Agent is its awareness of the context and semantics of your datasets in addition to an interface in the Tableau environment. This gives you a powerful tool to explore and analyze your data.

As we noted, Tableau Agent is cloud based and requires additional licensing. As of the time of writing, Tableau Agent requires Tableau+, a premium-tier offering from Tableau. Please talk to a Tableau representative to understand your options for paid or trial licensing.

When you first create a new Tableau workbook on Tableau Cloud, you'll be prompted with the **Connect to Data** screen.

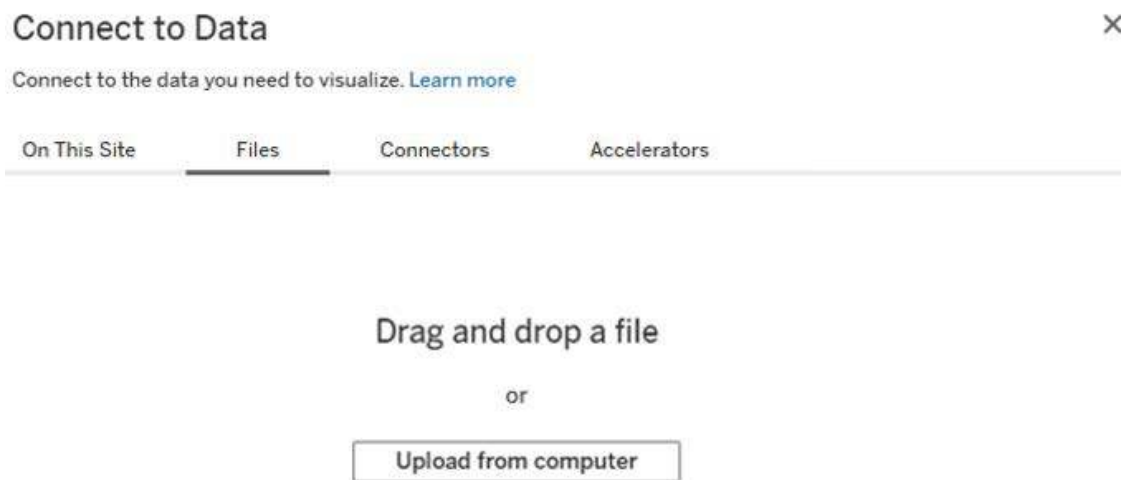


Figure 17.1: Uploading a file to Tableau Cloud for a new workbook

Here, we'll upload the `Superstore.csv` file located in the `\Learning Tableau\Chapter 16` directory of the sample files.

Once the data is ready, you'll find **Tableau Agent** in the top right, near **Show Me** and the data guide.

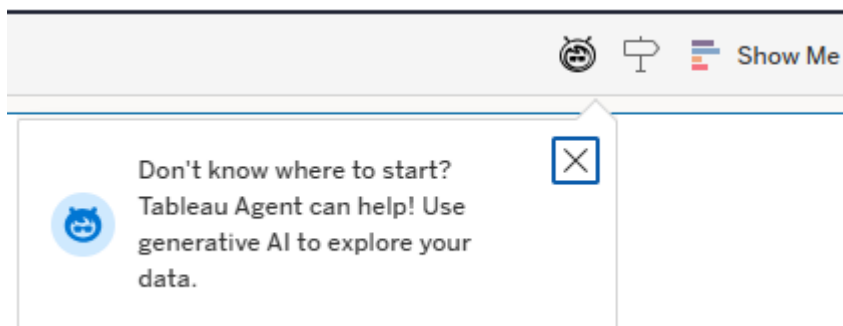


Figure 17.2: Tableau Agent

When you first click on the Tableau Agent icon, you'll receive some information along with a warning that generative AI can produce harmful or inaccurate results.



Note

Pay special attention to the warning as generative AI requires critical human thinking to evaluate results and suggestions. As helpful as AI is, *you* are critical in the process to ensure good, meaningful results.

Once you accept the disclaimer, Tableau Agent will offer you the opportunity to elicit suggestions via a button or enter a natural language prompt to get feedback.

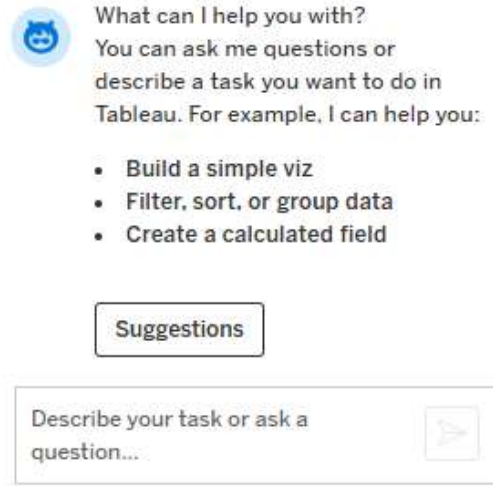


Figure 17.3: Initial interaction with Tableau Agent

We'll consider multiple ways of interacting with Tableau Agent. For now, let's examine what happens if we click the **Suggestions** button. In this case, Tableau has indexed our data and Tableau Agent uses this along with contextual and semantic awareness to provide some quick options for insight into the dataset:

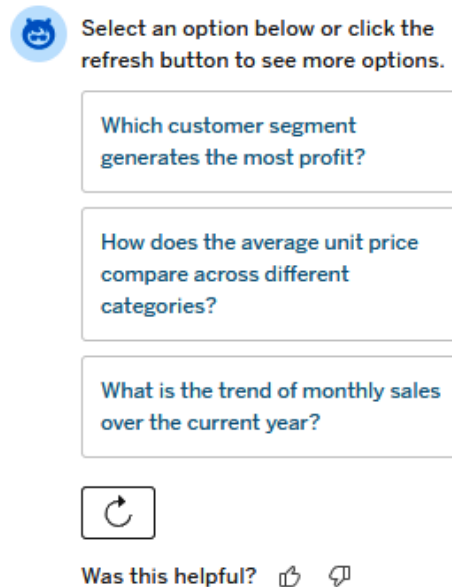


Figure 17.4: Tableau Agent provides contextually aware suggestions for exploring your data

The suggestions are based on your data and specific analysis you could pursue. Not only does this give you some potential avenues for analysis, but it also provides you with examples of the types of prompts that can elicit useful responses.



Note

Leverage Tableau Agent early in your data exploration to gain an understanding of your datasets and what types of initial analysis are possible.

Let’s say you wanted to pursue an analysis of sales over time. The final suggestion, **What is the trend of monthly sales over the current year?**, seems promising. Selecting that suggestion button leads Tableau Agent to build a viz!

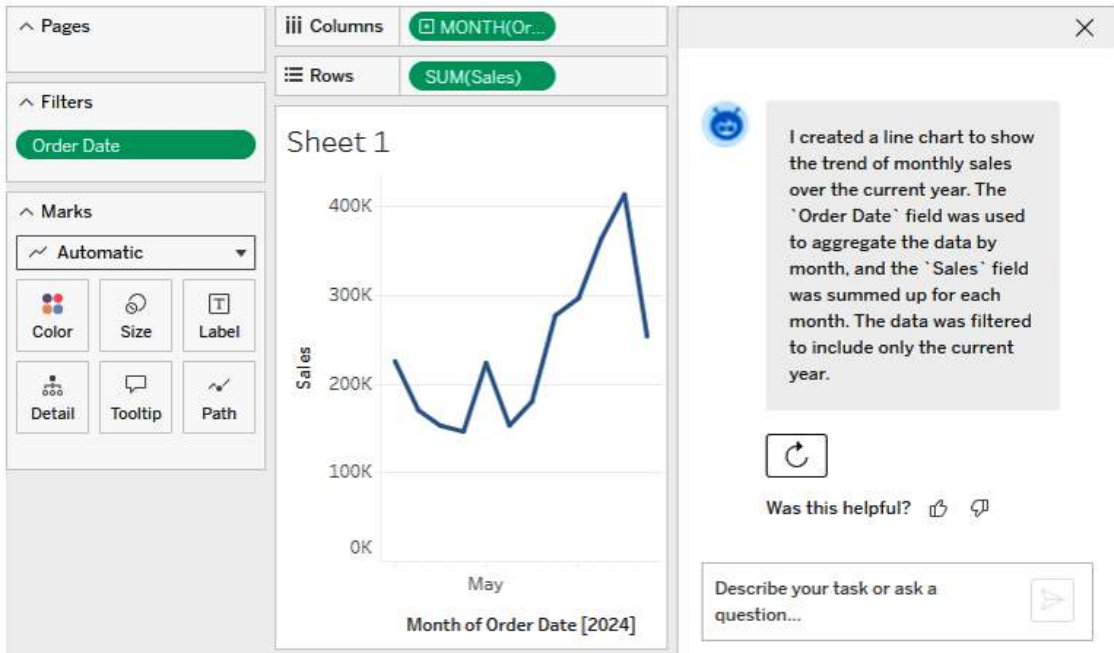


Figure 17.5: Tableau Agent is capable of building visualizations in response to prompts

Tableau Agent built a line chart to answer the question. It gives a natural language explanation of the chart and also describes how the view is filtered. Let’s say we were curious to see the top 3 states represented in this line chart. We might write a prompt similar to: Split the line by state and show the top 3 states. In response to this, Tableau Agent adjusts the visualization and filter and provides an additional response:

JM Split the line by state and show the top 3 states



I updated the viz to show the trend of monthly sales over the current year, split by state. Additionally, I filtered the data to show only the top 3 states with the highest sales.

Figure 17.6: Tableau Agent describes a modification to the view

Examining the resulting view reveals that Tableau Agent did indeed make the correct modifications:

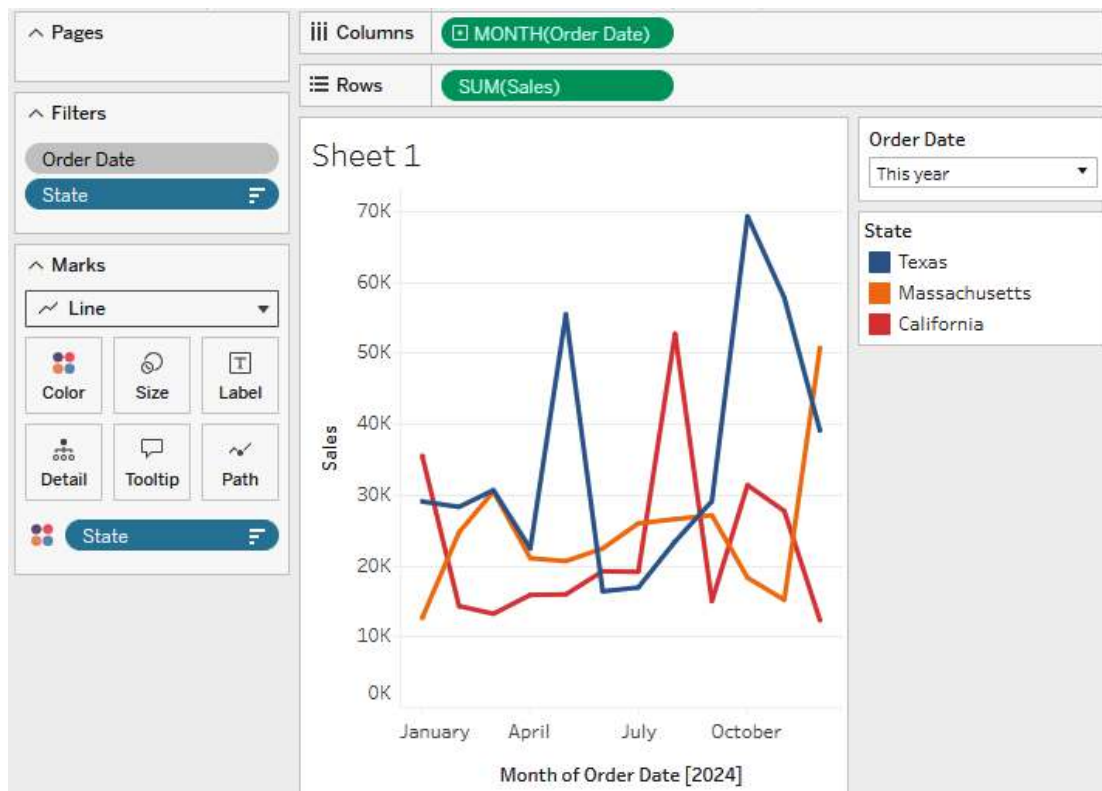


Figure 17.7: Tableau Agent describes a modification to the view

Not only did Tableau Agent add Sales to Color, but it also added a Top 3 filter and added the existing Order Date filter to the context (ensuring that the top 3 states are in the context of the date filter).

This initial analysis, assisted by Tableau Agent, may spark our understanding and imagination of how we approach the data. Seeing the sales by state may cause us to think about additional questions, such as, “How many customers are in each state?” Perhaps we’d like a calculation we could use in future analysis.

Tableau Agent not only provides general natural language feedback about our data and creates visualizations but is also capable of creating and editing calculations. We could prompt Tableau Agent to create a calculation that determines the number of customers per state. Tableau Agent creates a calculation and responds as follows:

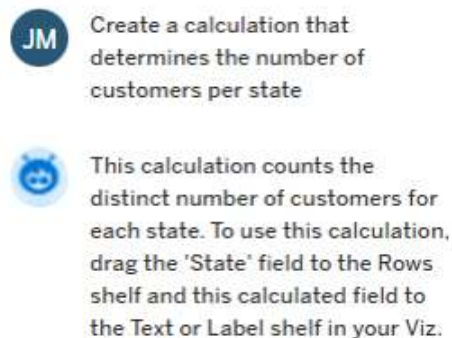


Figure 17.8: Tableau Agent describes a calculation it created

The resulting calculation does indeed calculate the number of distinct customers per state:



Figure 17.9: The calculation suggested by Tableau Agent

We could reuse this calculation in future analyses and visualizations. Note the Tableau Agent icon that appears in the upper-right corner of the calculation editor. Tableau Agent is always available to help in the context of a calculation.

Note



Be careful when using Tableau Agent to create calculations. You'll want to double-check that the calculation does what you want and that it is a good approach for your needs. For example, while the **FIXED** Level of Detail calculation does calculate the distinct number of customers per state, you could accomplish the same thing without any calculations in a simple view. On the other hand, the calculation may allow for more sophisticated analysis. This is where your knowledge and experience will guide you as you work with Tableau Agent to reach your goals.

Tableau Agent is an incredible assistant when it comes to exploring and analyzing your data. It gives you the ability to automate some tedious tasks, such as writing code. Use its suggestions and guidance along with your experience and curiosity to dig into the answers you seek!

Next, we'll take a look at Tableau Pulse and how it leverages multiple technologies to aid business users and executives in the flow of work.

Tableau Pulse

Tableau Pulse extends the Tableau platform, bringing generative AI, predictive analytics, and visualization together into a tool that allows easily digestible insights for executives and business users, along with the ability to drill into the drivers and causes of trends and predictions.

The heart of Tableau Pulse is the **metrics layer**, which allows an organization to define the **metrics** or **key performance indicators (KPIs)** that drive decision-making. Some examples of metrics include:

- The **number of new leads** for the sales team
- The **average wait time** for patients in the ER
- The **number of support tickets resolved**
- The **occupancy rate** of rooms in a hotel chain
- The **dollar value of products sold**

Each of these is a measurement that gives an indication of performance (good or bad). Each can be measured over time and compared with previous periods of time to indicate whether performance is improving or not. Each may have multiple factors that drive the values to be higher or lower.

In our exploration of Tableau Pulse, we'll consider a simple metric of Sales (a dollar amount of products sold) from the `Superstore.csv` file located in the `\Learning Tableau\Chapter 16` directory of the sample files. Although the metric is simple, we'll soon see the power of Tableau Pulse in uncovering insights and communicating the results in an easily accessible format for decision-makers.

Setting up the metric definitions

Definitions of each metric are kept in the metrics layer and individual metrics can be built on top of a single definition. For example, we might create a definition of **Sales** and then a couple of metrics such as **Month to Date Sales** or **Year to Date Sales**, which both use the same definition.

We'll start our exploration of Tableau Pulse by setting up a definition of **Sales**. After logging in to Tableau Cloud, we'll navigate to **Tableau Pulse** using the side panel:

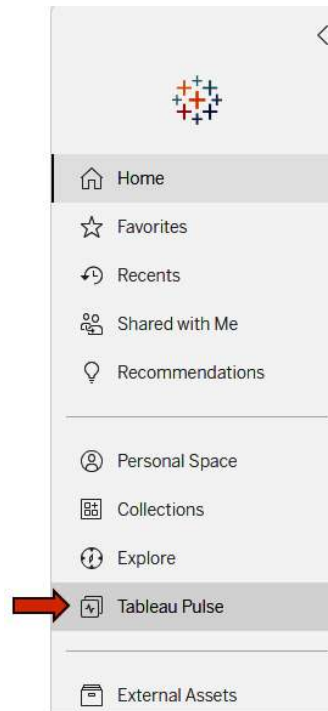


Figure 17.10: Tableau Pulse is available on Tableau Cloud through the navigation panel

You may be prompted with a disclaimer about AI before seeing the **Tableau Pulse** home screen:

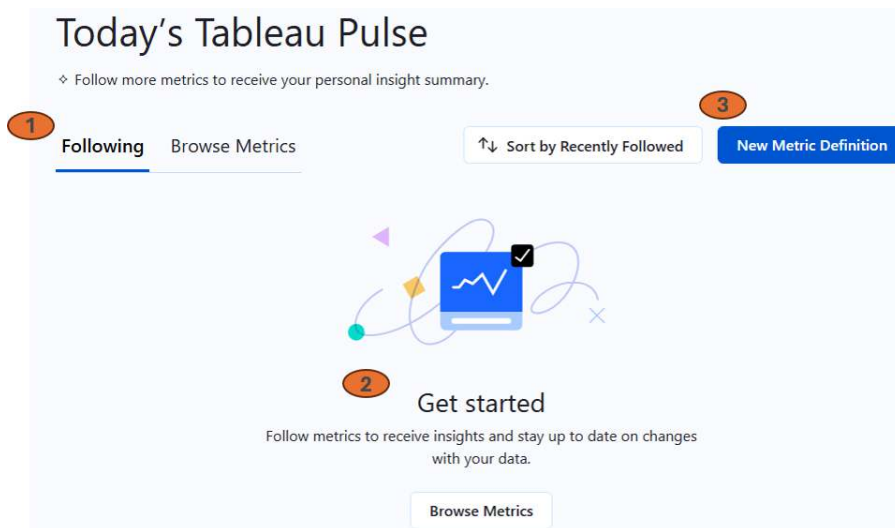


Figure 17.11: The Tableau Pulse home screen

The home screen consists of three main sections:

- 1. Tabs to show you metrics that you are following
- 2. A view of metrics you are following or an invitation to browse and start following metrics
- 3. The ability to define a **New Metric Definition**

We'll continue our journey by selecting **New Metric Definition** to create our definition. Upon clicking the button, we'll be prompted to select a published data source:

Select Data Source

Search

Warning	Type	Name	Connects To	Location
		Chess Openings	Data/Extracts/Extract.hy...	Chess
		Hymns	Data/Extracts/Extract.hy...	Music
		Sonatas	Data/Extracts/Extract.hy...	Music
		Black Holes	Astromony.csv	Astronomy
		Superstore	Superstore.csv	Superstore

Selected: "Superstore"

Connect

Figure 17.12: We'll select Superstore from the list of published data sources

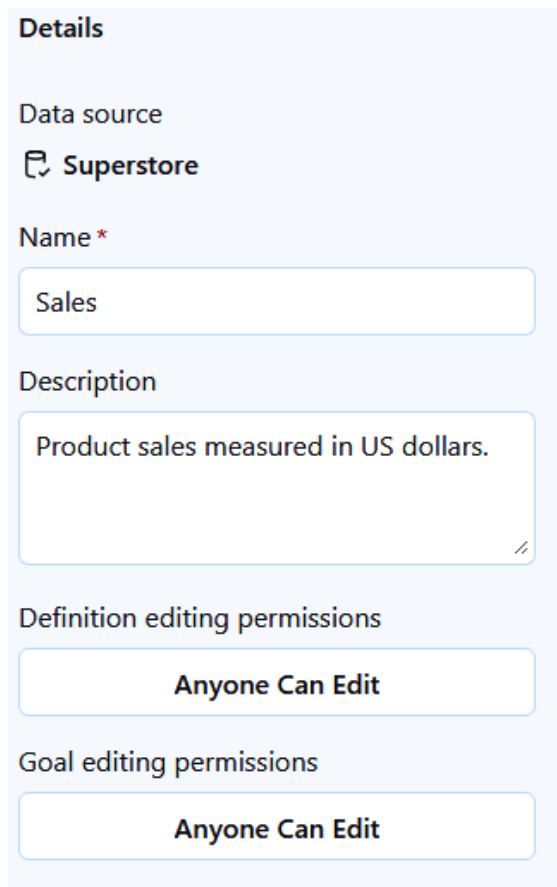
Note

You must use a published data source to define metrics. If you are following along with this example and have not already uploaded the `Superstore.csv` file, please do so, following the instructions in the *Tableau Agent* section of this chapter.

Once we have connected to the data source, we'll be able to configure our metric definition and the settings that drive insights into the metric. We'll start with the definition settings and then consider how to set up the drivers of insights.

Definition options

Let's first consider how to define our metric. We'll walk through the settings one section at a time. First, we find the ability to add a **Name** and **Description** and set up editing permissions:



Details

Data source

 **Superstore**

Name *

Sales

Description

Product sales measured in US dollars.

Definition editing permissions

Anyone Can Edit

Goal editing permissions

Anyone Can Edit

Figure 17.13: Name, description, and editing permissions for a metric definition

Giving a metric definition a clear name and definition will help users better understand the metrics that are based on the definition. You may optionally click the permission buttons to limit editing of either the definition or the goal of the metric to specific users and groups.

Next, we'll further define the metric based on the field in the data that defines our measure, the aggregation, and how time-based values should be treated:

The image shows the 'Value' configuration panel in Tableau, which is used to define a metric. The panel has a title 'Value' and a 'Clear Selections' link. It contains several sections, each with a numbered callout in an orange circle:

- 1** **Measure ***: A dropdown menu showing 'Sales' with a green grid icon.
- 2** **Aggregation**: A dropdown menu showing 'Sum'.
- 3** **Show sparkline values to date as ⓘ**: A dropdown menu showing 'Running total'.
- 4** **+ Add Definition Filter**: A button to add filters to the metric definition.
- 5** **Time dimension ***: A dropdown menu showing 'Order Date' with a blue calendar icon.
- 6** **> Advanced time settings**: A button to access more time-related options.

Figure 17.14: Configuring the definition of the metric value

In this section, we'll set up:

- The **Measure**, which is the field that primarily defines the measurement of our metric. In this case, we have selected **Sales**.
- The **Aggregation**, which defines how we want to summarize the metric. Sales makes sense to treat as a **Sum** because we will want to see how sales add up over time and other dimensions. In some cases, we might consider other aggregations, such as minimum, maximum, average, or median.
- Whether **to date** values (which will be shown as a **sparkline** in a metric visualization) should be a **Running total** or non-cumulative. In our case, sales do accumulate over time and a running total makes a lot of sense.
- If desired, we could limit the dataset analyzed by the metric definition by clicking the **Add Definition Filter**. In this example, we won't define any additional filters.

- We need to define a **Time dimension** as every dimension is measured over time, and the change in value over time will indicate performance. Here, we've selected **Order Date** as the best way to track sales over time.
- To define optional **Advanced time settings**, we'd expand that section of the definition editor.
- The expanded advanced settings are shown here:

Date offset ⓘ

0 days

Minimum time granularity

Day ▼

Time comparison ⓘ

Prior period ▼

Primary

Prior year ▼

Secondary

Create Advanced Definition

Figure 17.15: Configuring the advanced time settings of a metric definition

By default, every metric is measured to the current date. But you may have data that lags behind the current date. For example, you may have financial data that isn't final until a month is closed or time entry data that isn't relevant until the end of a week. The **Date offset** allows you to account for such cases and tell Tableau Pulse how much lag to allow for.



Tip

If you are following this example using the supplied dataset, you shouldn't have to define an offset as long as the date is within 2025. However, if you are working through this in 2026 or beyond, you'll see no data unless you define an offset.

The **Minimum time granularity** allows you to specify the smallest unit of time that is meaningful. Some metrics will fluctuate wildly over small periods of time. Sales, for example, may be up or down on any given day, and we may decide that a week is the minimum amount of time that should be used for comparisons.

You can optionally configure the **Primary** and **Secondary time comparison** to indicate the periods of time that should be compared to determine growth or decline.

As you adjust these settings, you will see a preview on the main screen:

Sales

Month to Date

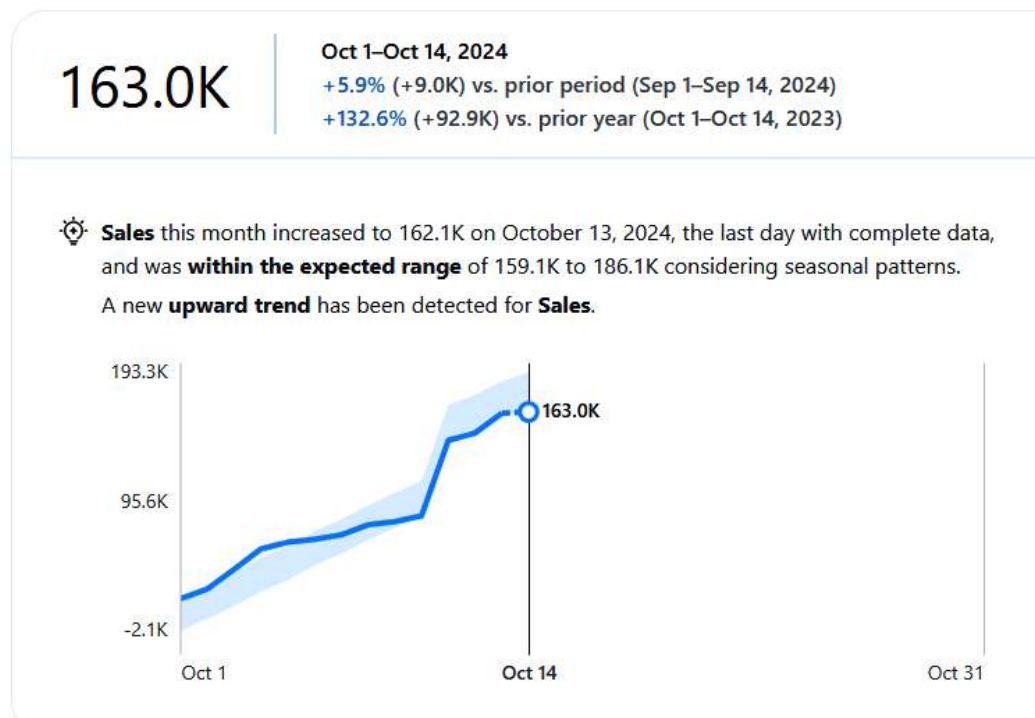


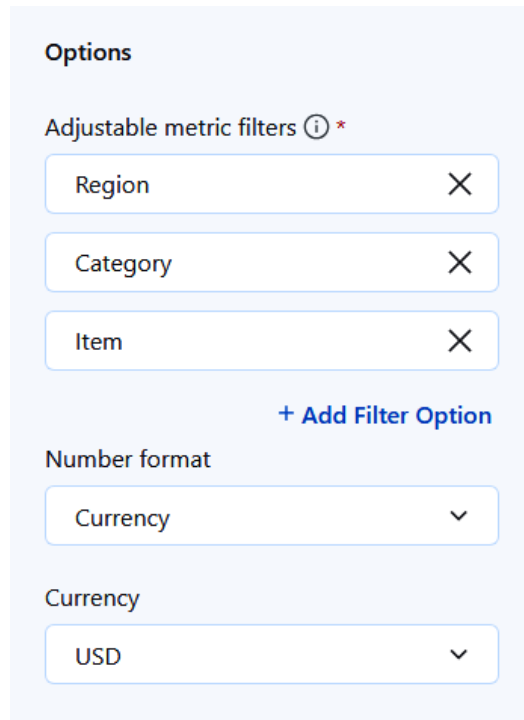
Figure 17.16: Configuring the advanced time settings of a metric definition

Notice the various features of the metric preview shown in *Figure 17.16*. **Sales** is shown **Month to Date** with a cumulative value of 163.0K. The exact timeframe is shown as October 1 through October 14 and the percent change for the primary and secondary time period comparisons is shown.

A natural language description of the Sales metric has been generated, describing the change and trend. A sparkline visualization completes the metric showing the running total of sales for the month to date timeframe.

This preview allows you to see a framework for a metric based on the definition.

We'll complete the definition with a few options shown here:



Options

Adjustable metric filters ⓘ *

Region ✕

Category ✕

Item ✕

[+ Add Filter Option](#)

Number format

Currency ▾

Currency

USD ▾

Figure 17.17: Options for filters and formatting

The final options allow you to add **Adjustable metric filters**. Unlike the **Definition Filter** we saw in *Figure 17.14* the adjustable metric filters allow you to add filters that end-users can change. These fields will also drive some of the insights and drill-down capabilities associated with the metric. In this example, we've added **Region**, **Category**, and **Item** as filters. You must have at least one filter, but can add additional filters as desired.

Finally, you may specify formatting for the metric. In this case, we've chosen **Currency** and **USD**. For certain metrics, you may supply text for the unit of measure.



Note

Just below the **Time comparison** settings, there is a **Create Advanced Definition** button (shown in *Figure 17.15*). Clicking this button will open an interface similar to a view canvas in Tableau, where you can drag and drop fields and see a visualization. The measure, time dimension, and filters you set up in the editor will be applied to the metric definition.

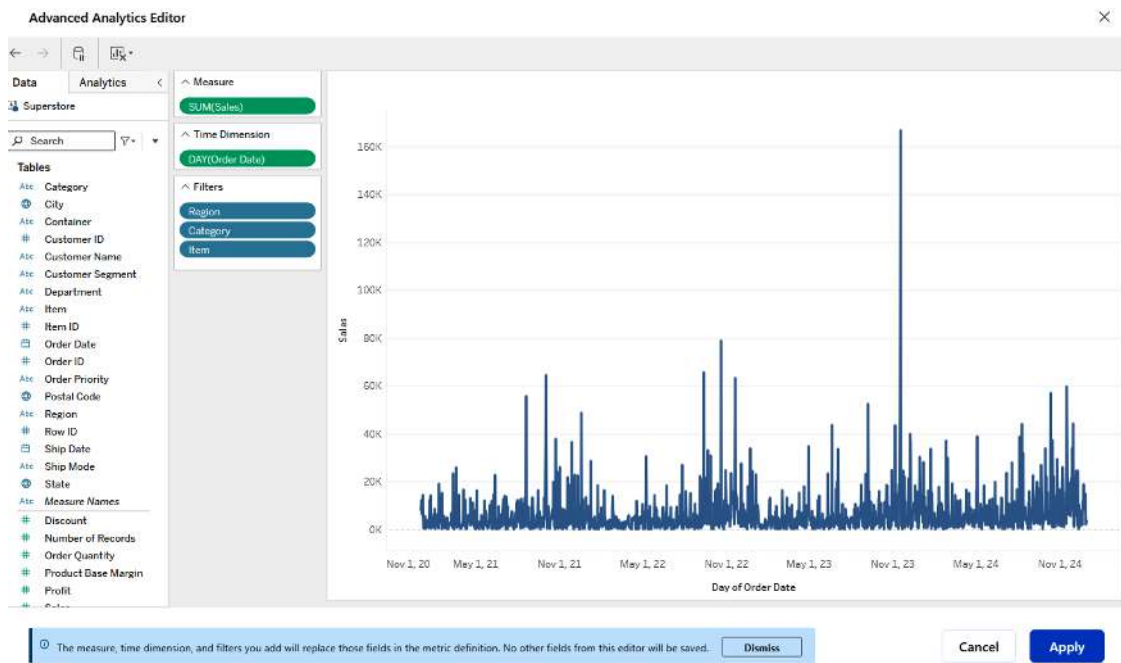


Figure 17.18: Advanced Analytics Editor for metric definitions

Prior to saving our metric definition, we'll need to set up the insights, which we'll consider next.

Insight options

In addition to defining the measure, time dimensions, and filters, we have a few additional options to adjust the metric so that advanced insights can be derived and generated as the data changes. This will help the AI know how to generate text that describes the metric and trends, as well as how to look for root causes within the data itself.

We'll examine each section of the insights section, which is accessed by selecting the **Insights** tab, as shown below:

Definition	Insights
Dimensions The fields you add as adjustable metric filters also control the insights generated.	
Region, Category, Item	
Value going up is	
Favorable	

Figure 17.19: Fields that control insights and setting what a change in value represents

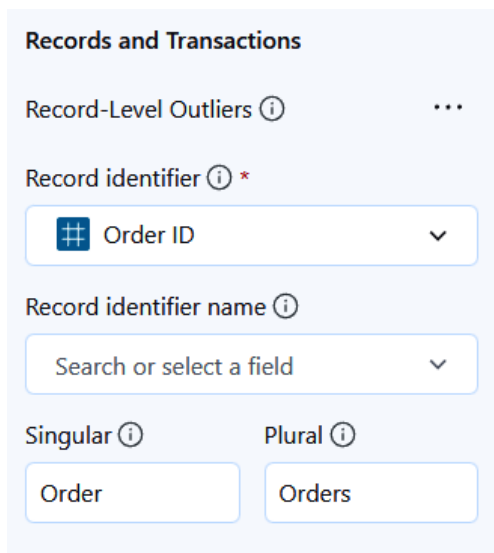
You'll notice that the fields we previously selected as adjustable metric filters show up here as fields that will control the insights that are generated. We'll see in detail the part they play.

You also have the ability to change what a change in the value represents. Specifically, you can set whether the **value going up is**:

- **Favorable** (an upward trend is good)
- **Neutral** (an upward trend is neither good nor bad)
- **Unfavorable** (an upward trend is bad)

You'll need to make the determination for each metric definition as to whether a higher value is good or bad. In our particular example, an upward movement in Sales is generally favorable. However, a higher average wait time for admittance into the ER is generally unfavorable.

Next, we'll set up individual insights, starting with records and transactions. Here, we must specify the record identifier and optionally a record name and what a singular record is called versus a plural group of records:



Records and Transactions

Record-Level Outliers ⓘ ...

Record identifier ⓘ *

 Order ID ▼

Record identifier name ⓘ

Search or select a field ▼

Singular ⓘ Plural ⓘ

Order Orders

Figure 17.20: Details for records and transactions

In this example, we've selected **Order ID** and we do not have a separate field that names each order. In some cases, we might have separate fields for the ID and name. For example, in a dataset with one record per customer, we might have a unique customer ID and also a customer name.

We've also identified that a single record is an **Order** and multiple records represent multiple **Orders**.

The remaining options for insights allow you to turn on or off various insights. You'll need to consider what insights will best serve the decision-makers and business users who are using the metric. Let's consider the types of insights and some examples.

You'll find the following types of insights:

- **Records and transactions:** These insights will focus on detecting outliers of individual records. For example, we might find individual sales order lines that are particularly large or fall outside the expected norms.
- **Trends and changes:** These insights will focus on the trend of the metric over time.
- **Contributions and breakdowns:** These insights will focus on what factors drive the change or move in an opposite direction from the change. There are also insights that focus on concentrated values and which factors contribute the least.

We've already considered the setup for records and transactions. As an example of a **trend change alert**, consider the following insight:



Figure 17.21: An insight into trends

Notice how Tableau Pulse identifies a previous trend and a new trend within the data and calls out the change.

For contributions and breakdowns, we'll find quite a few examples of insights. Here are a few:

💡 Compared to last month, **Sales increased** by \$41.5K. Out of the 17 members of **Category**, 9 increased. **Office Machines, Telephones and Communication, and Chairs & Chairmats increased the most.**

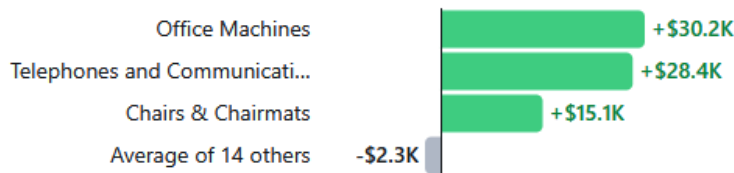


Figure 17.22: The top contributors to the increase in Sales based on Category

This insight focuses on values of **Category** that have most significantly contributed to the increase in sales. Notice the generated text description, which specifies the amount of change and how many categories changed (in which direction) and calls out the most significant.

You'll find other examples in the preview pane, such as this **bottom contributors** insight:

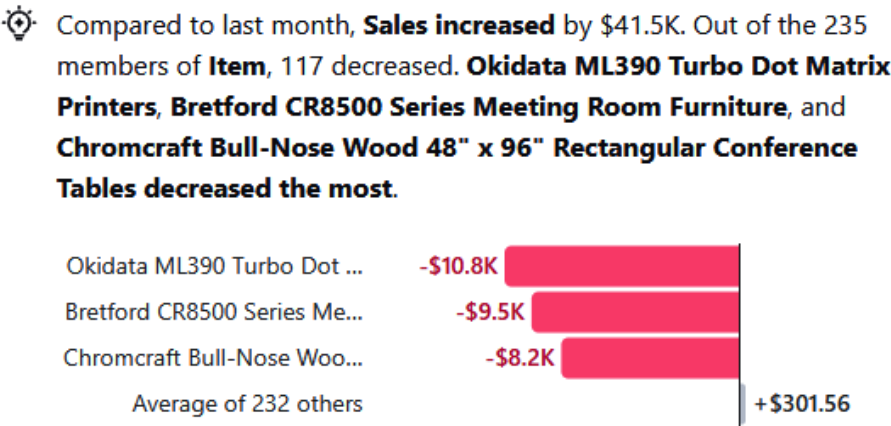


Figure 17.23: The greatest decrease in items in terms of Sales

This insight focuses on members of **Item** as the bottom contributors to **Sales**.

As an example of a **concentrated contribution alert**, consider the following:

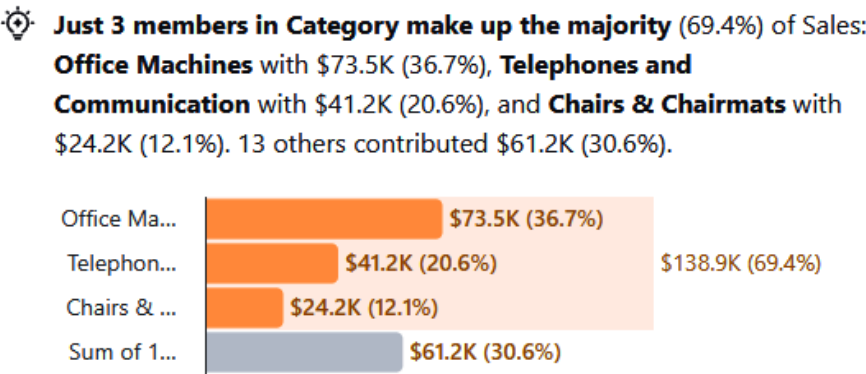


Figure 17.24: The greatest decrease in items in terms of Sales

You may find other types represented in the preview pane. Some types may not be applicable to your dataset or may only become applicable as new data arrives.

We'll complete our setup of the metric by clicking **Save Definition** in the lower-left pane:



Figure 17.25: Clicking **Save Definition** will save your metric definition and make it available for use in Tableau Pulse

Once your definition is saved, you (and others in your organization) may set up various metrics based on the definition and then use them in various ways. We'll briefly consider some options in the next section.

Creating and interacting with metrics

When you browse metrics in Tableau Pulse, you'll find an initial metric created based on the metric definition you saved:

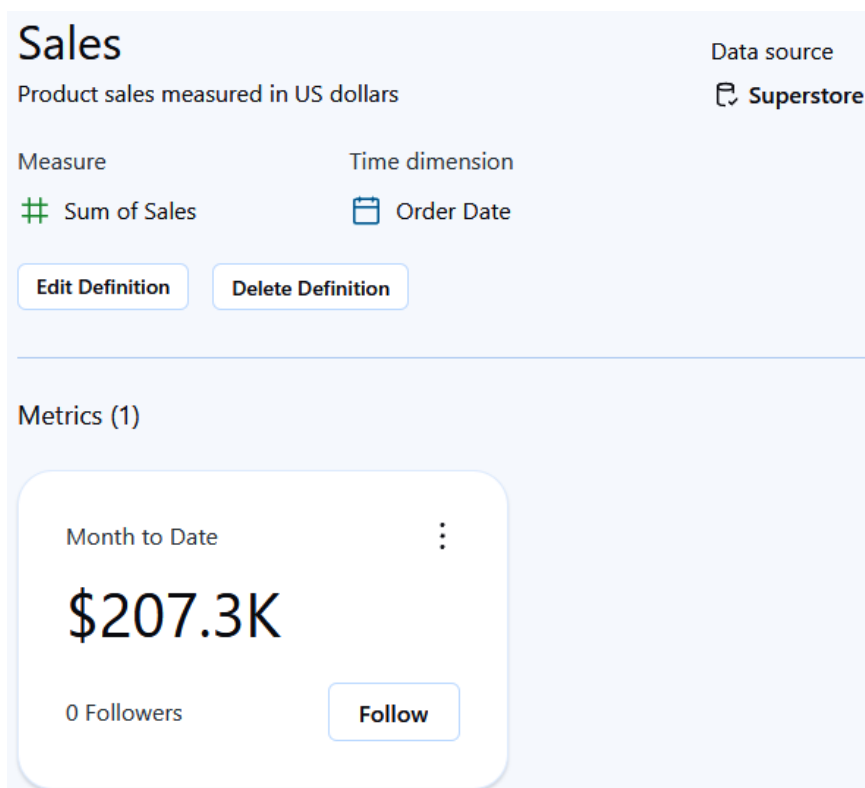


Figure 17.26: The **Sales** metric definition initially has a single metric, **Month to Date**

From here, you can add or edit the definition, follow the metric, or click the metric to interact with it and even create additional metrics.



Tip

When you follow a metric, it will show up in your personal insight summary. You may follow multiple metrics and see them show up on your Tableau Pulse home screen or in the Tableau mobile app.

If we want to interact with the metric, and potentially create different versions of the metric, we click the card. Doing so reveals the following interface:

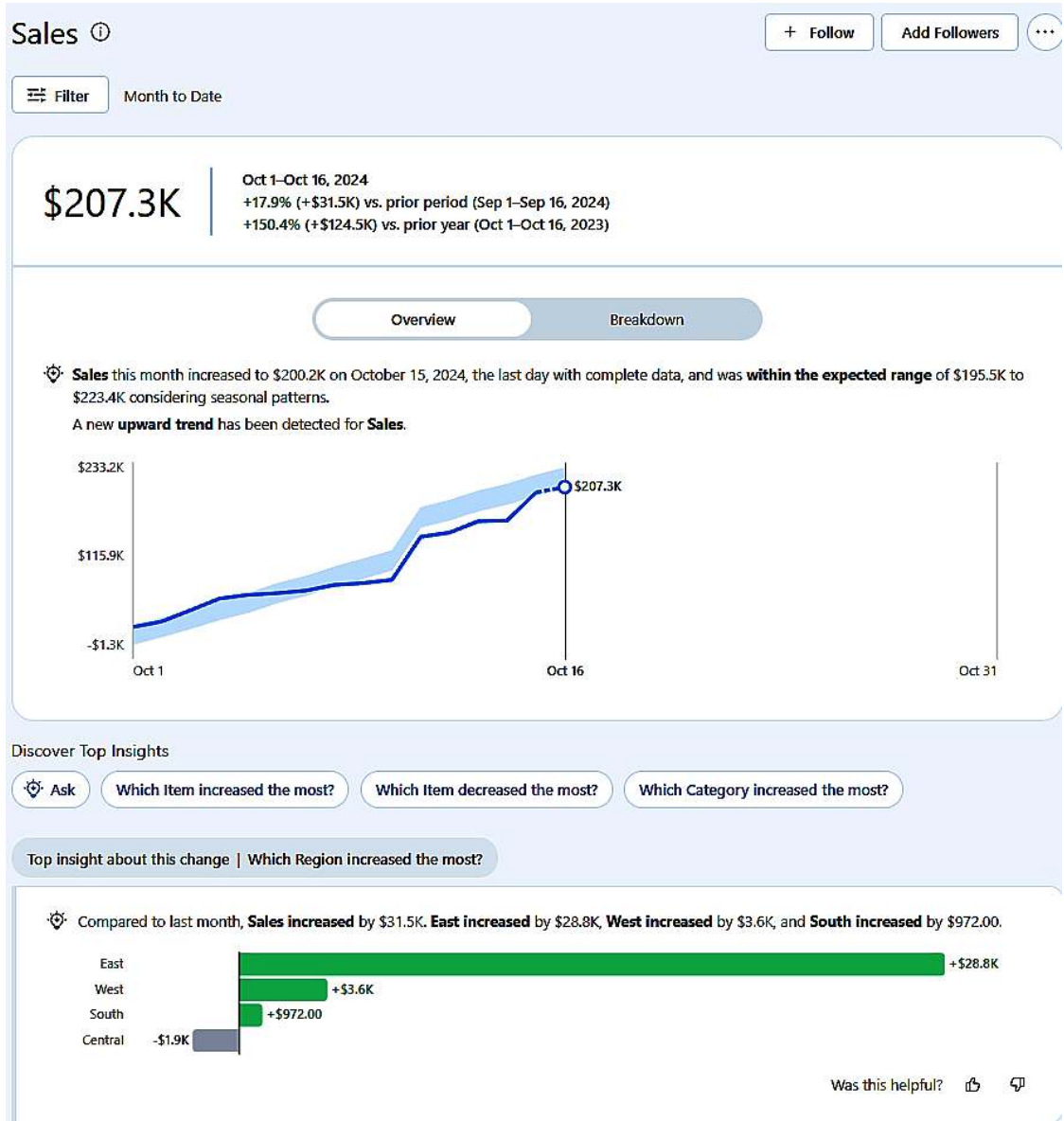


Figure 17.27: The Sales metric interface

The **Sales** metric displays the Month to Date number, a summary of changes from the previous period, a trendline of the month to date cumulative value, and the top insight. There are quite a few options for interaction:

- You may **Follow** or **Add Followers** to this particular metric using the buttons in the top-right corner of the metric screen
- Switching between **Overview** and **Breakdown** allows you to either see the month to date values overall or drill into the details of the breakdown by the dimensions you included as adjustable metric filters.

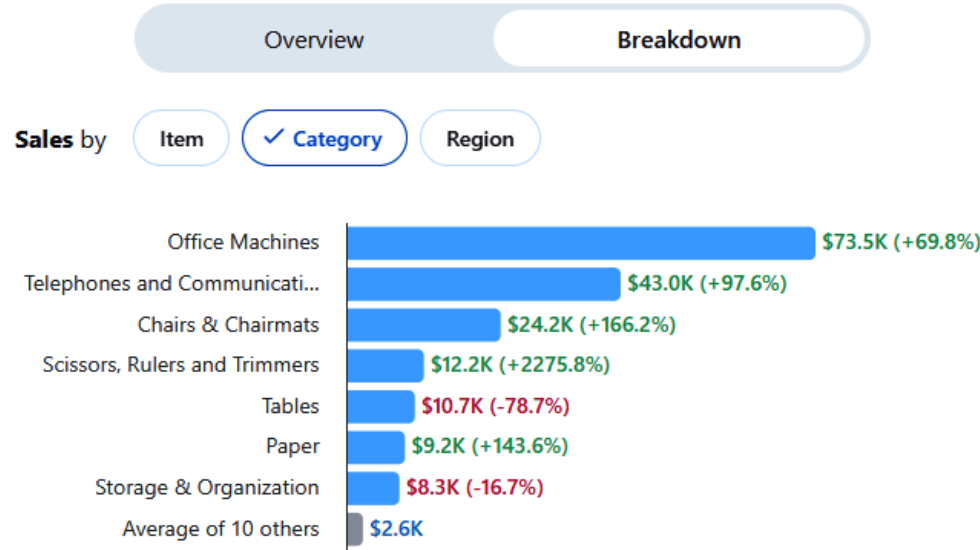


Figure 17.28: The breakdown of Sales by Category

Also shown are some buttons to prompt the insight generative AI to reveal more insights. There is also an **Ask** button that allows you to ask your own questions.



Figure 17.29: Quick insights into the metrics

As you can see, metrics are designed to give you quick insights and the ability to rapidly drill into details that help you understand the factors and causes of changes to your metric value. The AI-generated natural language descriptions help you easily understand the implications of each insight.

But what if you wanted to customize a metric? Let's say you were the manager of the Central Region and you were more concerned about year to date performance because you have a year-end goal of \$750K in Sales. In that case, you could select the Filter button at the top of the metric interface to set up your customized metric, like this:

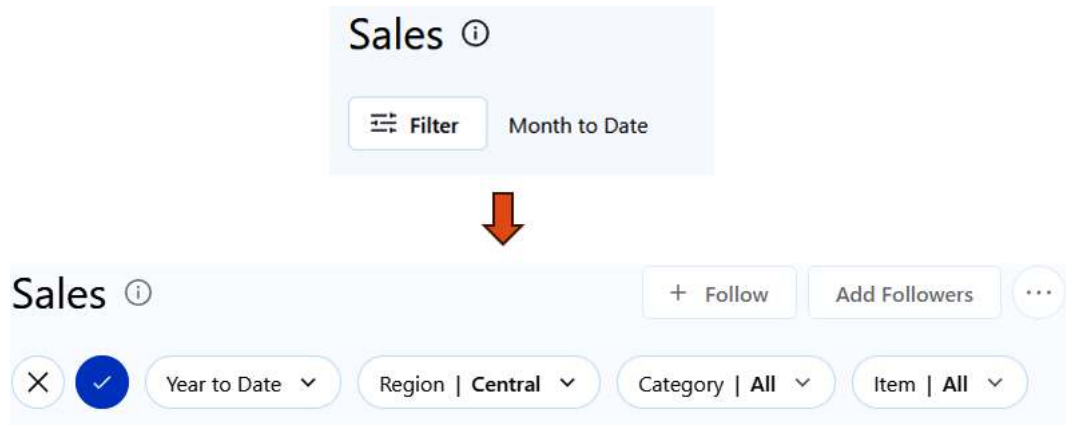


Figure 17.30: Customizing the Sales metric by filtering

After customizing the filters, you might also want to include your goal by selecting the ellipses button:

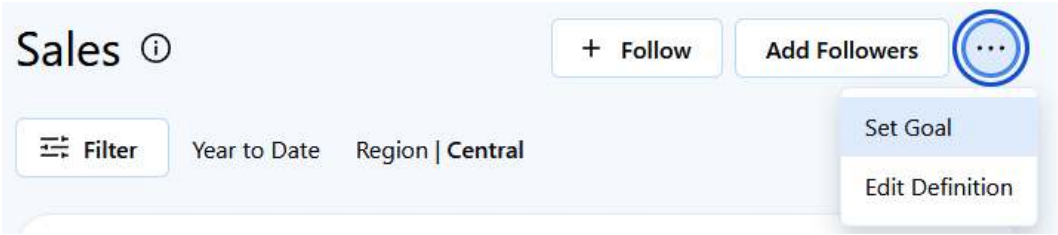
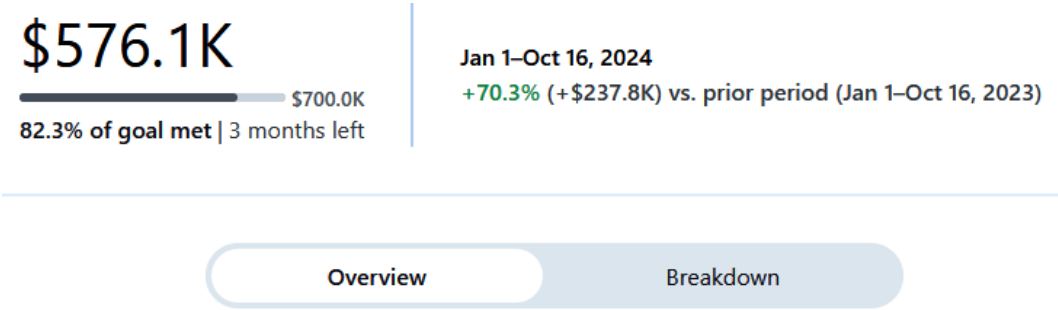


Figure 17.31: Adding a goal to the metric

Once you’ve entered your goal of 700,000, you may wish to **Follow** the new metric. The **Sales** metric definition now has two metrics built on top of it. You may notice that your new metric incorporates your goal into the visualizations and descriptions:



💡 **Sales** this year increased to \$511.8K in September 2024, the last month with complete data, and was **within the expected range** of \$464.2K to \$534.8K considering seasonal patterns. **Sales is rising at a steady rate.** The trend first began 45 months ago on January 1, 2021.

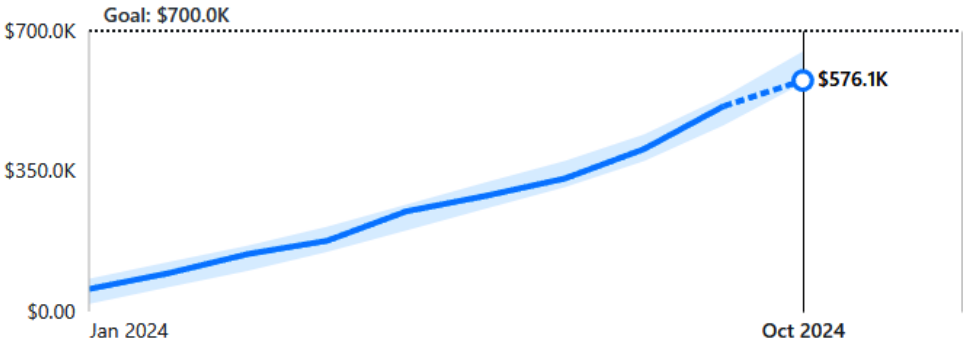


Figure 17.32: The new metric shows the goal and indicates how much of the goal has been achieved

Tableau Pulse gives you quite a bit of power and flexibility to create a single metric definition and then multiple metrics that use a common semantic layer but are contextualized for various users and decision-makers!

Before we close the chapter, let's take a quick look at a couple of additional ways of seeing and sharing your metrics.

Seeing and sharing metrics

In addition to working with metrics in a browser, you can interact with metrics in several other ways:

- **Use the Tableau mobile app to interact with metrics.** Tableau metrics are designed to be mobile friendly. Asking questions via button taps, switching between overviews and breakdowns, and drilling into insights are all very accessible.

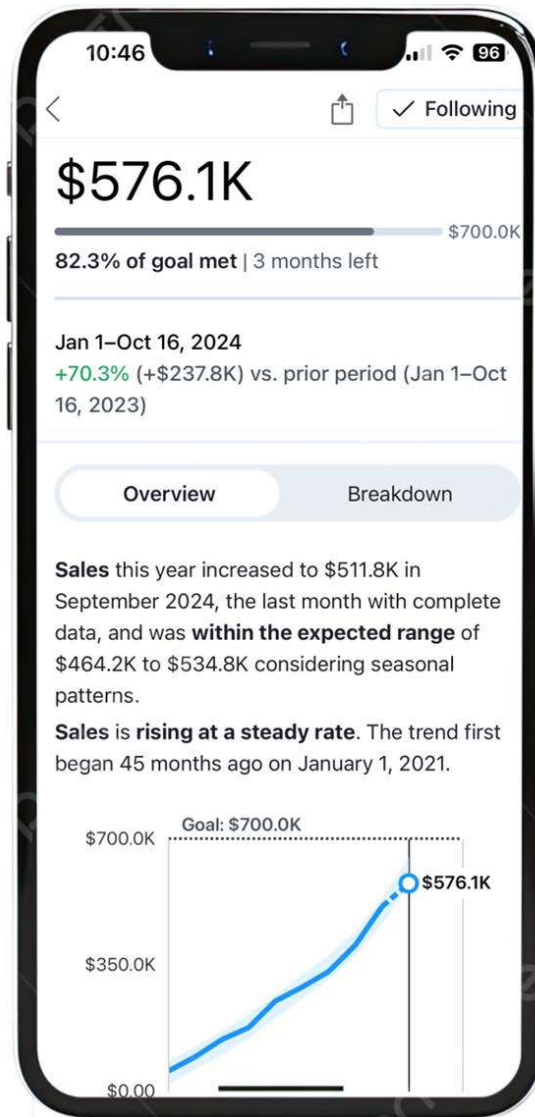


Figure 17.33: Interacting with metrics using the Tableau mobile app

- **Metrics may be embedded in Tableau dashboards.** Metrics can be embedded in any dashboard in a workbook on Tableau Cloud that uses the same published data source as the metric. This gives you a lot of possibilities for leveraging the power and simplicity of metrics, while including other context and analysis.

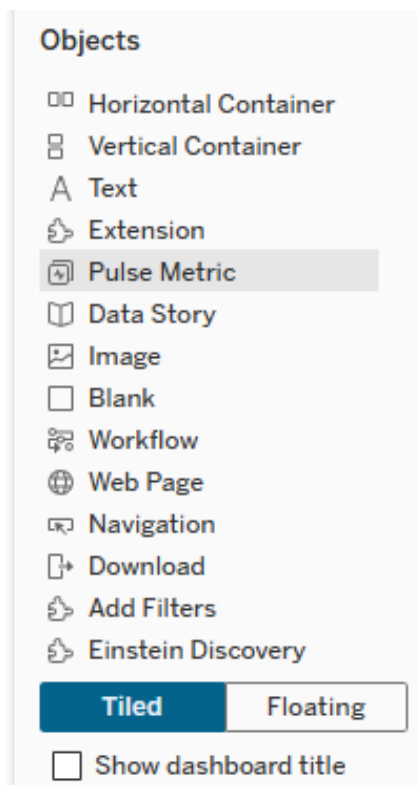


Figure 17.34: Pulse Metric appears in the list of objects for dashboards in workbooks on Tableau Cloud

Metrics can be embedded in messaging applications such as Slack and Teams. You'll need to install the appropriate plugins, but this provides a seamless way of interacting with your data in the natural flow of work and communication.

Summary

Tableau Agent and Tableau Pulse bring together multiple AI technologies to assist you in discovering insights and making key decisions faster than ever. Tableau Agent is your personal AI assistant that leverages a contextual awareness of your data to give you an interactive experience where you can ask questions and get answers in the form of natural language, visualizations, and calculations. Tableau Pulse gives you the ability to create a single metric definition and then derive multiple metrics based on the common semantic layer. This gives you and others in your organization the ability to quickly define metrics that deliver contextual and meaningful insights. You'll get natural language descriptions of trends, outliers, and causes and the ability to effortlessly drill into the details as you seek to understand the implications of your data.

While reading through the chapters of this book, we have observed that Tableau is an amazing platform for exploring, prepping, and cleaning your data as you create useful and meaningful visualizations and dashboards to understand and communicate key insights. Throughout this book, we've considered how to connect to data—whether file-based, in an on-premises database, or in the cloud. You've worked through examples of exploring and prepping data to clean it and structure it for analysis. We've covered numerous types of visualization and how they can uncover deep analytical insights. The four main types of calculations were explored in depth, giving you the tools to extend the data, analysis, and user interactivity. You've built dashboards and told stories with the data. You've expanded your horizons with extensions, scripts, and AI assistance!

You now have a solid foundation. At its core, the Tableau platform is intuitive and easy to use. As you dive deeper, the simplicity becomes increasingly beautiful. As you discover new ways of understanding your data, solving complex problems, asking new questions, and finding new answers in your datasets, your new Tableau skills will help you uncover, interpret, and share new insights hidden in your data.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

<https://packt.link/ds>





packtpub.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

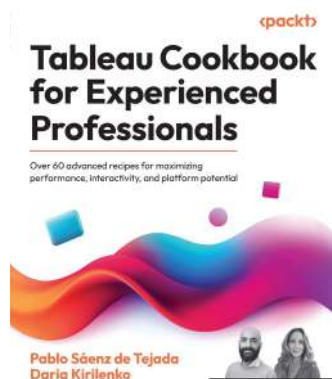
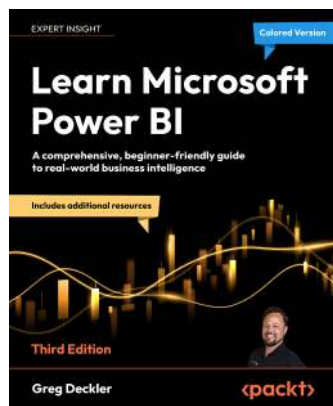


Tableau Cookbook for Experienced Professionals

Pablo Sáenz de Tejada, Daria Kirilenko

ISBN: 978-1-83546-974-3

- Build advanced, high-performing Tableau data models and tables
- Tackle data complexities with LOD expressions and table calculations
- Leverage zone visibility to build interactive and user-friendly dashboards
- Optimize workbook performance for improved user experience
- Analyze geospatial data with map layers, shapefiles, and spatial joins
- Develop a deeper understanding of Tableau's REST API capabilities
- Use Tableau's order of operations to troubleshoot calculations
- Adopt industry-standard best practices to secure corporate data assets



Learn Microsoft Power BI

Greg Deckler

ISBN: 978-1-83664-741-6

- Understand the fundamentals of Microsoft Fabric and Copilot
- Plan and manage your business intelligence (BI) projects
- Connect to and transform data using Power Query
- Create semantic models and DAX calculations optimized for analysis and reporting
- Uncover business insights and design professional reports
- Deploy and govern Power BI, with a focus on gateways and deployment pipelines

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share Your Thoughts

Now you've finished *Learning Tableau 2025_Sixth edition*, we'd love to hear your thoughts! If you purchased the book from Amazon, please [click here](#) to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Index

A

actions 252

- creating 252, 253
- filter actions 253-256
- Go to Sheet actions 253, 260
- highlight actions 253, 257
- parameter actions 253, 260, 261
- set actions 253, 261-265
- triggers 252
- URL actions 253-259

active fields 5

addressing 171, 183-187

ad hoc calculations 148, 149, 175

ad hoc custom territories 364, 365

Adobe PDF 43

advanced visualizations 314

- bump charts 317
- dumbbell chart 321, 322
- jump lines 318
- Marimekko chart 325-327
- slope charts 314-317
- sparklines 319, 320
- step lines 318
- unit chart 322-325
- waterfall charts 317, 318

aggregate calculations 124, 134-137

- versus row-level calculations 137, 138

aggregations 421, 436-441, 474-477

alias 407

Alteryx 56

Amazon RDS 48

Amazon S3 48

Analytics Extensions API 509

animated visualizations 329

animation

- analysis, enhancing 329-331
- data storytelling, enhancing 331, 332

area charts 29, 101-103

B

background images

- data, plotting 372-376

Background Layers pane 350

bar charts 12, 13, 74-76, 313

- categories of interest, highlighting 85-87
- iterations, for deeper analysis 14-16
- variations 76

bar-in-bar chart 81-85

base tables 378

batch mode 451

bins 111

Blend Relationships 403

blends

- need for 408
- using 401-403, 409
- using, to display goal progress 403-408

box plots 108, 109, 313

brushing 456

BUFFER() function 361, 362

bullet graph 77-79

thresholds, calling 79-81

bump charts 317

C

Calculated Field 124

calculated level of detail 154

calculations 123, 124

aggregate calculations 124

creating 124, 125

editing 125

level of detail calculations 124, 471-474

performance considerations 149, 150

reference link, for official Tableau forums 142

row-level calculations 124, 469, 470

table calculations 124

calculations and parameters, practical examples

ad hoc calculations 148, 149

analysis, enhancing 145, 146

data, extending 143-145

data issues, fixing 142, 143

user experience, enhancing 145, 146

visualizations, enhancing 145, 146

canvas 5

cardinality 383

Cartesian product 434

circle charts 105, 106

jittering 106, 107

cloud

data connection 48-50

cloud-based data sources 2

clustering analysis

performing 300-303

coalesce 431

combination chart 116, 117

conditional logic 130-132

connection 42

context 68

continuous fields 8, 10

Create Parameter interface 140, 141

creating and editing calculations interface

calculated field name 126

code editor 126

functions list 126

indicator 126

cross-database join 398, 399

cross-data source filtering 68

cross-join 434

custom SQL 442

custom SQL filters 61

custom table calculations 187

functions, running 191

lookup and previous value 188-190

meta table functions 187, 188

rank functions 193

script functions 193

Total function 193, 194

Window functions 192

custom territories 364

ad hoc custom territories 364, 365

creating 364

field-defined custom territories 366

D

Dashboard Confusion

reference link 232

dashboard extensions 494, 495

files 504

HTML file 505

JavaScript file 506

manifest file 504

- using 495-502
- working 507-509
- dashboard, profitability 237**
 - context filtering 247-249
 - framework, creating 241-243
 - story, driving with actions 244-246
 - views, building 238-240
- dashboards 5, 30, 232**
 - approaches 233
 - building 33-35
 - designing, for displays and devices 249-251
 - exploratory 233
 - floating 236
 - guided analysis 233
 - interface 31-33
 - narrative 233
 - objectives 232, 233
 - objects 234, 235
 - objects, manipulating 236, 237
 - scorecard/status snapshot 233
 - tiled 236
- data**
 - filtering 477, 478
 - in chapter examples 129
 - plotting, on background images 372-376
 - transforming 479-487
 - visualizing 11
- databases 2**
- data blending 401, 442**
- data cleaning 460, 463-469**
- data connection 2, 3, 42, 452-459**
 - in cloud 48-50
 - in database 46, 47
 - shortcuts 50, 51
 - to Excel file 43-46
- data densification 171, 433**
- data fields 5**
- data filtration**
 - continuous (green) fields, filtering 65, 66
 - dates, filtering 67
 - discrete (blue) fields, filtering 63-65
 - options 68
 - order of operations 69, 70
- data filtration techniques**
 - custom SQL filters 61
 - data source filters 61
 - extract filters 61
- data grouping 463-469**
- Data Guide 296**
 - pane 296
 - using 296-300
- data highlighters 257**
- data level of detail 153**
- data model 443**
 - need for 408
 - using, to relate tables 409
- Data pane 4**
- data problems**
 - advanced fixes 442, 443
- data scaffolding 443**
- data source 42, 377**
- data source filters 61**
- data source metadata**
 - managing 51-54
- Data Source screen 44**
 - connections 44
 - data model canvas 44
 - data source filters 45
 - data source name 44
 - live or extract options 44
 - object dropdown 45
 - preview pane/metadata view 45
 - preview pane options 45
 - sheets 44
 - toolbar 44

data storytelling

enhancing, with animation 331, 332

data structures 412, 413

star schema 418-420

tall data 414, 415

wide data 413, 414

wide data, versus tall data 415-418

data transformations 421

aggregation 436-441

joins 432-436

overview 421

pivot 421-427

union 427-432

data types 127

Boolean 127

date 127

date and time 127

number (decimal) 127

number (whole) 127

spatial 127

string 127

data variations

planning 133

date parts 87-89**dates and times**

variations, visualizing 91, 92

visualizing 87

date values 87, 90**delimiter 130****density maps 26, 28****design mode 451****dimension filter 7, 8, 66****dimension tables 378, 418****direction 177****discrete fields 8****DISTANCE() function 360, 361****distributions**

analyzing 304-307

visualizing 104

Dropbox 48**dual axis 115, 116****dual-axis map 362****dumbbell charts 321, 322****dynamic dashboards 335**

Dynamic Zone Visibility 342-345

Show/Hide buttons 335-342

Dynamic Zone Visibility 342-345**E****Einstein Discovery 509****Einstein Trust Layer**

reference link 524

exact dates 87, 90**Excel file**

data connection 43-46

exclude level of detail calculations 156

examples 167-169

execution mode 451**extract filters 61****extracts 43**

avoiding, scenarios 61

creating 55-57

performance 59, 60

portability and security 60

using 57-59

using, scenarios 60

working with 54, 55

F**fact tables 378, 418****field-defined custom territories 366**

field-level formatting 206, 210, 211

- custom date formatting 214, 215
- custom number formatting 211-213
- null formatting 216-220

file-based data sources

- Adobe PDF 43
- extracts 43
- JSON file 43
- local cube file 43
- Microsoft Access 43
- Microsoft Excel 43
- spatial file 43
- statistical file 43
- text file 43

filled maps 22-24**filter actions 253-256****filters 68****fixed 176****fixed level of detail expressions 156, 471**

- examples 158-164

flat files 2**floating objects 268****flow 449, 451****forecasting 307**

- using 307-312

foreign key 419**formatting**

- additional options 220, 221
- field-level formatting 210, 211
- leveraging 204-206
- workbook-level formatting 206, 207
- worksheet-level formatting 207-210

full outer join 393**functions**

- aggregate 126
- date 126
- logical 126
- number 126

pass through 126**reference link 128****spatial 126****string 126****table calculation 126****type conversion 126****user 126**

G

Gantt charts 92-95**geographic visualizations 21, 22****density maps 26-28****filled maps 22-24****symbol maps 24-26****geospatial data**

- custom geographic definitions, importing 354
- latitude data, implementing 352
- longitude data, implementing 352
- spatial objects, using 354-356
- using 352

Git 503**Google Cloud SQL 48****Google Drive 48****Go to Sheet actions 253, 260****granularity 413**

H

highlight actions 253, 257**highlighters 257****histograms 29, 109-113****HTML file 505**

I

Image Map Filter extension 495**include level of detail expressions 156**

- examples 164-167

inner joins 391, 420

Interquartile Range (IQR) 108

INTERSECTS() function 363, 364

J

JavaScript file 506

jittering 106, 107

join 391, 421, 432-436

calculations 397, 398

considerations 397

cross-database join 398, 399

need for 408

unintentional errors 399, 400

using 391

using, at physical level 409

join calculation 434

join culling 420

join types 391

full outer join 393

inner join 391

left join 391

right join 392

spatial join 393, 394

JSON file 43

jump lines 318

K

Key Performance Indicators (KPIs) 233, 530

k-means clustering algorithm 300

L

late filtering 172, 196

left join 391

level of detail 153, 156-158

calculated 154

data 153

exclude 156

fixed 156

include 156

syntax 155

view 154, 155

level of detail calculations 124, 153, 172, 358, 389, 411, 471-474

measures 437

line charts 16, 18, 313

iterations, for deeper analysis 19, 20

lines 29

live connections 420

local cube file 43

logical layer 384

log viewing tool

reference link 151

Lookup value 188-190

M

MAKELINE() function 359, 360

MAKEPOINT() function 359, 360

manifest file 504

map layers 367-370

Marimekko chart 325-327

reference link 327

measure filter 7, 8, 66

menu 4

metadata 51

meta table functions 187, 188

metric definitions

definition options 533-542

insight options 538, 539

metrics 530

creating 542-546

interacting with 542-546

sharing 547, 548

viewing 547, 548

metrics layer 530
Microsoft Access 43
Microsoft Azure SQL Database 48
Microsoft Excel 43
mismatched fields
 merging 460-462
multi-fact analysis 380, 390

N

nested calculations 133
nesting 133
Node.js 503
non-Cartesian chart types 101
npm 503

O

ODBC connection 42
OneDrive 48
Online Analytical Processing (OLAP) 43, 59
Online Transactional Processing (OLTP) 59
operators 128

P

packaged workbook 60
parameter actions 253, 260, 261
parameters 139
 creating 139
 using, in Tableau Prep Builder 488-491
partitioning 171, 183-187
physical layer 384
pie charts 103, 104
pills 5
pivots 421-427
previous value 188-190

primary interface 4, 5
proportional brushing 262, 456
P-value 290
Python 510
Python script
 using, in Tableau 512-516

Q

quick table calculation 174, 175

R

RANK() function 193
referential Integrity 383
relationships 381, 382
relative 176
rich-text formatting 206
right join 392
row-level calculations 124, 129, 469, 470
 versus aggregate calculations 137, 138
Rserve 509
R-Squared value 290

S

Sankey Viz Extension
 flow, visualizing 119-121
scatterplot 114, 115
scope 177
scorecard dashboard 265-268
script functions 193
sentiment analysis 517
set actions 253, 261-265
sheet 5
Show/Hide buttons 335-342
Show Me 28
 features 29
 using 29, 30

side-by-side bar chart 15
 slope chart 314-317
 sparklines 319, 320
 sparse data 433
 spatial file 43
 spatial functions
 BUFFER() function 361, 362
 DISTANCE() function 360, 361
 INTERSECTS() function 363, 364
 MAKELINE() function 359, 360
 MAKEPOINT() function 359, 360
 using 356-358
 spatial join 393, 394
 stacked bars 15, 95-98, 313
 star schema 418-420
 statistical file 43
 step lines 318
 steps 449, 450
 aggregate step 450
 clean step 450
 input step 450
 join step 450
 New Rows 450
 output step 451
 pivot step 451
 prediction step 451
 script step 451
 union step 450
 stories feature 5, 269-275
 Story interface 271
 strings 130
 concatenating 129
 manipulation 130-132
 sunburst diagram
 with Viz Extensions 327-329
 surrogate key 419
 symbol maps 24-26

T

Tableau Agent 523-530
 Tableau analytics extensions 509
 Tableau Dashboard Extensions API 495
 Tableau data model
 behaviors 386-389
 creating 379-383
 Data pane interface 385, 386
 exploring 379
 layers 384, 385
 logical layer 384
 multi-fact analysis 390, 391
 overview 377-379
 physical layer 384
 using 385
 Tableau Exchange 497
 Tableau extensions 494
 creating 502, 503
 dashboard extensions 494, 495
 Viz Extensions 494
 Tableau Extensions API SDK 504
 Tableau Hyper API 42
 Tableau maps
 map layers, customizing 349, 350
 map options, customizing 351, 352
 overview 347
 rendering 348
 tips and tricks 371, 372
 Tableau Next 523
 Tableau paradigm 37, 38
 working 38-42
 Tableau Prep 56, 445
 Tableau Prep Builder 445, 446, 492
 interface 446-449
 parameters, using 488-491
 URL 446
 Tableau Prep Conductor 445, 492

Tableau Pulse 523, 530

metric definitions, setting up 530-532

Tableau Query Language (TQL) 38**table calculations 124, 171, 442**

addressing and partitioning 183-187

creating 173, 174

editing 173, 174

filtering 196

fundamental concepts 176

overview 172, 173

practical examples 195-200

relative versus fixed 176, 177

scope and direction 177-183

table extensions 516

adding 517-521

tables

joining 394-397

TabPy 509

setting up 510, 512

tall data 414, 415

versus wide data 415-418

text file 43**third-party Viz Extensions 328****tiled objects 268****time series 16****token number 130****toolbar 4****tooltips 225, 227****Total function 193, 194****transpose 421****treemaps 99-101, 313****trend models 286**

analyzing 289-292

exponential model 288

linear model 287

logarithmic model 287

polynomial model 289

power trend model 288

statistical model details, exporting 292-295

trends 278-283

trend lines, customizing 284-286

two-pass total technique 440

U

union 421, 427-432**unioning 460-462****unit chart 322-325****unpivot 421****URL actions 253, 258, 259**

V

values

comparing 74

view level of detail 15, 154, 155**visible fields 57****visualization**

considerations 203, 204

default formatting 222-224

Visual Query Language (VizQL) 37**Viz Extensions 118, 494**

setting up 118, 119

Viz in Tooltip 227, 228

W

waterfall charts 317-319**Web Data Connector (WDC) 42, 412****Web Map Service (WMS) servers**

reference link 350

whisker plots 108, 109**wide data 413, 414**

versus tall data 415-418

Window functions 192**workbook 5, 60****workbook-level formatting 206, 207****worksheet-level formatting 207-210**