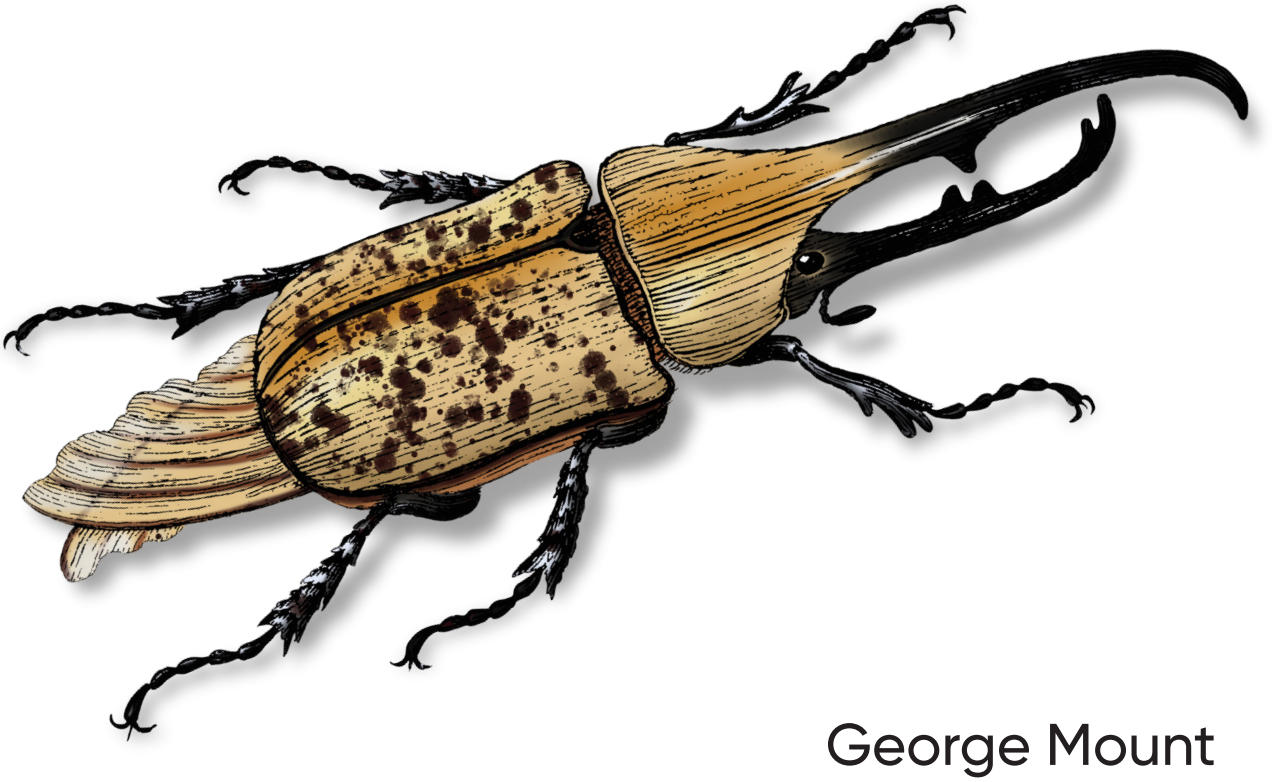
O'REILLY®

Modern Data Analytics in Excel

Using Power Query, Power Pivot, and More for Enhanced Data Analytics



George Mount

O'REILLY®

Modern Data Analytics in Excel

If you haven't modernized your data cleaning and reporting processes in Microsoft Excel, you're missing out on big productivity gains. And if you're looking to conduct rigorous data analysis, more can be done in Excel than you think. This practical book provides an introduction to the modern Excel suite of features along with other powerful tools for analytics.

George Mount of Stringfest Analytics shows business analysts, data analysts, and business intelligence specialists how to make bigger gains right from your spreadsheets by using Excel's latest features. You'll learn how to build repeatable data cleaning workflows with Power Query and design relational data models straight from your workbook with Power Pivot. You'll also explore new features for analytics, including dynamic array functions, AI-powered insights, and Python integration.

Learn how to build reports and analyses that were previously difficult or impossible to do in Excel. This book shows you how.

- **Build repeatable data cleaning processes for Excel with Power Query**
- **Create relational data models and analysis measures with Power Pivot**
- **Pull data quickly with dynamic array functions**
- **Use AI to uncover patterns and trends from inside Excel**
- **Integrate Python functionality with Excel for automated analysis and reporting**

"As someone who used Excel occasionally for work, I found *Modern Data Analytics* easy to follow and packed with practical tips. George Mount's straightforward approach is helpful for seasoned analysts and casual users alike."

—Meghan Finley
Technical writer and editor

George Mount is the founder of Stringfest Analytics, a consulting firm specializing in analytics professional development. He has worked with leading bootcamps, learning platforms, and practice organizations to help individuals excel at analytics. George is the recipient of Microsoft's Most Valuable Professional (MVP) award for exceptional technical expertise and community advocacy in Excel, and he's the author of *Advancing into Analytics: From Excel to Python and R* (O'Reilly, 2021). He resides in Cleveland, Ohio.

DATA SCIENCE

linkedin.com/company/oreilly-media
youtube.com/oreillymedia

US $59.99 CAN $74.99
ISBN: 978-1-098-14882-9

9 781098 148829 55999

Modern Data Analytics in Excel

Using Power Query, Power Pivot, and More for Enhanced Data Analytics

George Mount

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Modern Data Analytics in Excel
by George Mount

Copyright © 2024 Candid World Consulting, LLC. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

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

Acquisitions Editor: Michelle Smith
Development Editor: Sara Hunter
Production Editor: Christopher Faucher
Copyeditor: Penelope Perkins
Proofreader: Helena Stirling
Indexer: BIM Creatives, LLC
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

May 2024: First Edition

Revision History for the First Edition
2024-04-26: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098148829* for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Modern Data Analytics in Excel*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

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

978-1-098-14882-9

[LSI]

Table of Contents

Preface ix

Part I. Data Cleaning and Transformation with Power Query

1. Tables: The Portal to Modern Excel 3
Creating and Referring to Table Headers 3
Viewing the Table Footers 6
Naming Excel Tables 8
Formatting Excel Tables 9
Updating Table Ranges 9
Organizing Data for Analytics 10
Conclusion 11
Exercises 11

2. First Steps in Excel Power Query 13
What Is Power Query? 13
Power Query as Excel Myth Buster 13
"Excel Is Not Reproducible" 13
"Excel Does Not Have a True null" 14
"Excel Can't Process More Than 1,048,576 Rows" 15
Power Query as Excel's ETL Tool 15
Extract 15
Transform 17
Load 18
A Tour of the Power Query Editor 18
The Ribbon Menu 19
Queries 21

The Imported Data 22
Exiting the Power Query Editor 24
Returning to the Power Query Editor 26
Data Profiling in Power Query 26
What Is Data Profiling? 27
Exploring the Data Preview Options 27
Overriding the Thousand-Row Limit 31
Closing Out of Data Profiling 31
Conclusion 32
Exercises 32

3. Transforming Rows in Power Query. 33
Removing the Missing Values 34
Refreshing the Query 37
Splitting Data into Rows 39
Filling in Headers and Cell Values 42
Replacing Column Headers 42
Filling Down Blank Rows 43
Conclusion 44
Exercises 44

4. Transforming Columns in Power Query. 45
Changing Column Case 45
Delimiting by Column 47
Changing Data Types 47
Deleting Columns 48
Working with Dates 48
Creating Custom Columns 49
Loading & Inspecting the Data 51
Calculated Columns Versus Measures 52
Reshaping Data 53
Conclusion 54
Exercises 55

5. Merging and Appending Data in Power Query. 57
Appending Multiple Sources 57
Connecting to External Excel Workbooks 58
Appending the Queries 61
Understanding Relational Joins 62
Left Outer Join: Think VLOOKUP() 64
Inner Join: Only the Matches 68
Managing Your Queries 70

Grouping Your Queries 70
Viewing Query Dependencies 71
Conclusion 72
Exercises 73

Part II. Data Modeling and Analysis with Power Pivot

6. First Steps in Power Pivot. 77
What Is Power Pivot? 77
Why Power Pivot? 77
Power Pivot and the Data Model 80
Loading the Power Pivot Add-in 81
A Brief Tour of the Power Pivot Add-In 83
Data Model 83
Calculations 83
Tables 84
Relationships 84
Settings 84
Conclusion 84
Exercises 85

7. Creating Relational Models in Power Pivot. 87
Connecting Data to Power Pivot 87
Creating Relationships 88
Identifying Fact and Dimension Tables 92
Arranging the Diagram View 93
Editing the Relationships 94
Loading the Results to Excel 95
Understanding Cardinality 99
One-to-One Cardinality 100
One-to-Many Relationships 101
Many-to-Many Relationships 101
Why Does Cardinality Matter? 102
Understanding Filter Direction 103
Filtering orders with users 104
Filtering users with orders 105
Filter Direction and Cardinality 106
From Design to Practice in Power Pivot 106
Creating Columns in Power Pivot 106
Calculating in Power Query Versus Power Pivot 106
Example: Calculating Profit Margin 107

Recoding Column Values with SWITCH() 109
Creating and Managing Hierarchies 111
Creating a Hierarchy in Power Pivot 111
Using Hierarchies in the PivotTable 112
Loading the Data Model to Power BI 113
Power BI as the Third Piece of "Modern Excel" 113
Importing the Data Model to Power BI 114
Viewing the Data in Power BI 116
Conclusion 118
Exercises 118

8. Creating Measures and KPIs in Power Pivot. 119
Creating DAX Measures 119
Creating Implicit Measures 119
Creating Explicit Measures 122
Creating KPIs 128
Adjusting Icon Styles 130
Adding the KPI to the PivotTable 131
Conclusion 132
Exercises 132

9. Intermediate DAX for Power Pivot. 133
CALCULATE() and the Importance of Filter Context 134
CALCULATE() with One Criterion 135
CALCULATE() with Multiple Criteria 136
AND Conditions 136
OR Conditions 137
CALCULATE() with ALL() 138
Time Intelligence Functions 140
Adding a Calendar Table 140
Creating Basic Time Intelligence Measures 143
Conclusion 149
Exercises 149

Part III. The Excel Data Analytics Toolkit

10. Introducing Dynamic Array Functions. 153
Dynamic Array Functions Explained 153
What Is an Array in Excel? 154
Array References 154
Array Formulas 156

An Overview of Dynamic Array Functions 158
Finding Distinct and Unique Values with UNIQUE() 158
Finding Unique Versus Distinct Values 159
Using the Spill Operator 160
Filtering Records with FILTER() 160
Adding a Header Column 162
Filtering by Multiple Criteria 162
Sorting Records with SORTBY() 163
Sorting by Multiple Criteria 164
Sorting by Another Column Without Printing It 164
Creating Modern Lookups with XLOOKUP() 165
XLOOKUP() Versus VLOOKUP() 165
A Basic XLOOKUP() 166
XLOOKUP() and Error Handling 167
XLOOKUP() and Looking Up to the Left 168
Other Dynamic Array Functions 168
Dynamic Arrays and Modern Excel 169
Conclusion 169
Exercises 170

11. Augmented Analytics and the Future of Excel. 171
The Growing Complexity of Data and Analytics 171
Excel and the Legacy of Self-Service BI 172
Excel for Augmented Analytics 173
Using Analyze Data for AI Powered Insights 173
Building Statistical Models with XLMiner 179
Reading Data from an Image 181
Sentiment Analysis with Azure Machine Learning 184
Conclusion 188
Exercises 188

12. Python with Excel. 189
Reader Prerequisites 190
The Role of Python in Modern Excel 190
A Growing Stack Requires Glue 190
Network Effects Mean Faster Development Time 191
Bring Modern Development to Excel 192
Using Python and Excel Together with pandas and openpyxl 193
Other Python Packages for Excel 194
Demonstration of Excel Automation with pandas and openpyxl 195
Cleaning Up the Data in pandas 196
Summarizing Findings with openpyxl 200

Adding a Styled Data Source 204
Conclusion 206
Exercises 207

13. Conclusion and Next Steps. 209
Exploring Excel's Other Features 209
LET() and LAMBDA() 210
Power Automate, Office Scripts, and Excel Online 210
Continued Exploration of Power Query and Power Pivot 211
Power Query and M 211
Power Pivot and DAX 212
Power BI for Dashboards and Reports 213
Azure and Cloud Computing 213
Python Programming 214
Large Language Models and Prompt Engineering 214
Parting Words 215

Index. 217

Preface

Welcome to the Excel revolution. By updating how you think about and use Excel, you can unlock significant productivity gains and use your data more powerfully. This book introduces the "modern Excel" suite of features and other powerful analytics tools.

Learning Objective

By the end of this book, you should be able to use modern Excel tools for data cleaning, analysis, reporting, and advanced analytics. In particular, you'll clean and transform data with Power Query, create relational models in Power Pivot to build sophisticated analyses, and explore the Excel analytics toolkit to further automate and enhance your work.

Prerequisites

To meet these objectives, this book makes some technical and technological assumptions.

Technical Requirements

To make the most of this book, it is recommended that you have a Windows computer with the Microsoft 365 version of Excel for desktop. The features covered in this book are relatively new and may not be available in older Excel versions. Please note that many of these tools are still being developed for Mac, and compatibility may vary. Due to the fast-paced nature of Excel's development, it is difficult to provide a precise list of what's available for each version.

Chapter 7 of the book briefly explains how to load a Data Model from Excel into Power BI. It assumes that, as a Microsoft 365 for Windows user, you already have the free version of Power BI Desktop installed on your computer. Chapter 12 delves into

the integration of Python with Excel, guiding you through the process of downloading Python for free. All subsequent tasks and exercises within the book are designed to be completed exclusively within Excel, eliminating the need for external programs. However, you will configure a few Excel add-ins as part of the process.

Technological Requirements

This book is designed for intermediate Excel users eager to discover modern features with which they might not be familiar. To fully benefit from it, you should already be acquainted with the following Excel topics:

- Working with absolute-, relative-, and mixed-cell references
- Building conditional logic and conditional aggregation functions (`IF()` statements, `SUMIF()`/`SUMIFS()`, and so forth)
- Combining data sources (`VLOOKUP()`, `INDEX()`/`MATCH()`, or other lookup functions)
- Sorting, filtering, and aggregating data with PivotTables
- Basic plotting (bar charts, line charts, and so forth)

If you would like more practice with these topics before continuing, I recommend *Microsoft Excel 365 Bible* by Michael Alexander and Dick Kusleika (Wiley, 2022).

In Part III of the book, you will explore advanced concepts in statistics, programming, and related areas. Don't be discouraged if these topics appear challenging at first. There are ample resources to assist you in gaining proficiency, and I will provide helpful references when necessary. The primary objective of this book is to demonstrate the vast possibilities that Excel offers.

If you prefer to enhance your knowledge first before delving into these topics, I recommend reading my book *Advancing into Analytics: From Excel to Python and R* (O'Reilly, 2021). It offers comprehensive insights and guidance on advanced analytics techniques, Python programming, and various other topics relevant to modern data analytics in Excel.

How I Got Here

My journey to the data world started with Excel during the early 2010s, before data science and AI had fully taken the world by storm. At that time, Excel often felt like a closed system. If you desired to perform advanced analytics, it was commonly advised to switch to Python or R. For self-service relational data models, Access was recommended. Many of the complex analyses and automations I aimed to accomplish involved cumbersome VBA modules and unwieldy array formulas, making the user experience less than ideal.

For a while, it appeared that Excel might eventually succumb to obsolescence. However, today's Excel, bolstered by various features and applications, has undergone a remarkable transformation.

What Is "Modern Analytics"? Why Excel?

Modern analytics refers to the use of advanced tools and techniques to prepare and analyze data, ranging from simple retrospective analyses to predictive modeling and artificial intelligence. In the evolving landscape of data-driven decision making, it's essential to have tools that are versatile and interoperable, enabling users to perform a wide range of analytics activities.

Previously, Excel fell short in meeting these requirements. However, Excel has undergone significant transformation over the past decade, making it a true powerhouse for modern data analytics.

This book aims to dispel common misconceptions held by technical professionals about Excel and to demonstrate its capabilities in the modern analytics realm. By showcasing features such as Power Pivot, Power Query, and other tools, this book challenges the belief that Excel is limited to basic formulas and functions. It emphasizes that today's Excel has evolved into a robust platform capable of handling complex data analytics tasks.

Ultimately, this book showcases Excel as a powerful and versatile tool for modern analytics. It seeks to debunk myths, guiding technical professionals and managers to fully exploit Excel's potential for effective data analysis and decision making. In doing so, it enables users to harness Excel as a crucial component of the contemporary analytics toolkit, providing insights and driving success in our data-driven world.

Modern Excel and Interoperability

Modern analytics emphasizes interoperability, so it's not surprising that many tools showcased in this book are also prevalent elsewhere in the analyst's toolkit. Notably, Power Query and Power Pivot, discussed in Part I and Part II respectively, are also available in Power BI, Microsoft's business intelligence and reporting tool. Python can also be utilized in Power BI. These tools can be combined in various ways, and as you master one, you're likely to encounter it in a different context. This book primarily focuses on Excel, but it's helpful to understand how these elements fit into the broader modern analytics toolkit.

Book Overview

To meet the learning objective and scope of this book, I've divided the content into three parts.

Part I, Data Cleaning and Transformation with Power Query

Part I focuses on Power Query for data cleaning in Excel, and how it can be used as an extract, transform, load (ETL) tool. You'll get a tour of the Power Query Editor, learning about data profiling and various transformation techniques such as filtering, splitting, aggregating, and merging data.

Part II, Data Modeling and Analysis with Power Pivot

Part II introduces Power Pivot for Excel, focusing on its use for reporting. You will learn how to define relationships, build a Data Model, and enhance it with calculated columns, key performance indicators (KPIs), and more—primarily using the Data Analysis Expressions (DAX) language.

Part III, The Excel Data Analytics Toolkit

Part III of the book explores several exciting new features for data analysis in Excel. You will learn about dynamic array functions, which enable quick and flexible spreadsheet calculations. Additionally, the book provides a primer on predictive analytics and AI, discussing their potential applications in Excel and offering a glimpse into the program's future. The book concludes with an advanced topic: building an automated workbook using Python. You will learn how to effectively leverage Python and Excel together to enhance your analytical capabilities.

End-of-Chapter Exercises

When I read books, I tend to skip over the exercises at the end of the chapter because I feel keeping the momentum of my reading is more valuable. *Don't be like me!*

At the end of most chapters, I offer opportunities to apply what you've learned through practice. Exercises and their solutions are located in the *exercises* folder within the accompanying repository, organized into subfolders by chapter number. I encourage you to complete these drills and then compare your responses with the provided solutions. By doing so, you will not only enhance your understanding of the material, but also set a positive example for me.

This Is Not a Laundry List

Excel's rapid development pace and the abundance of new tools can be overwhelming. To avoid losing focus and making the book unwieldy, I have carefully selected a specific set of topics with broad potential and usefulness for intermediate Excel users, drawing from my years of experience as an Excel consultant and trainer.

If your favorite or most impactful feature for modern analytics in Excel is not covered in this book, I encourage you to share your perspective as a valued member of the community. The realm of data analytics in Excel goes beyond the boundaries of a single book, and the Excel community is eager to learn from your insights and experiences.

Are you ready to embark on a tour of modern Excel? I'll meet you in Chapter 1.

Conventions Used in This Book

The following typographical conventions are used in this book:

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

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

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/stringfestdata/modern-analytics-excel-book*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Modern Data Analytics in Excel* by George Mount (O'Reilly). Copyright 2024 Candid World Consulting, LLC, 978-1-098-14882-9."

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

O'Reilly Online Learning

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

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

How to Contact Us

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

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-889-8969 (in the United States or Canada)
707-827-7019 (international or local)
707-829-0104 (fax)
support@oreilly.com
https://www.oreilly.com/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/modern-data-analytics-excel*.

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

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

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

Acknowledgments

One of the most fascinating aspects of writing a book, especially the acknowledgments, is that it captures a moment in your life and highlights the people who are significant at that time.

Many of these names can be found in the acknowledgments to my previous book. I am especially grateful to the acquisitions team at O'Reilly, Michelle Smith and Jon Hassell, for giving me the green light to write another book. My friend and fellow O'Reilly author, Tobias Zwingmann, whose work I have mutually reviewed over the years, provided an exceptionally helpful technical review for this project. Additionally, my parents, Jonathan and Angela Mount, have been unwavering in their support, more than I could ever ask for. It's uncertain how many mothers wish their children to become Excel authors, but mine has been incredibly supportive.

I also had the opportunity to deepen my acquaintance with some individuals through this project. I extend my thanks to Alan Murray, Joseph Stec, and Meghan Finley for their invaluable additional technical reviews. Meghan, in particular, has not only brought her impressive technical editing experience to the book but has also been an incredible support as my girlfriend throughout the writing process. (As any author will tell you, writing a book inevitably becomes a family affair.) Additionally, I am grateful to Jeff Stevens, Laura Szepesi, and Mark Depow for their feedback on the manuscript.

Moreover, I owe a debt of gratitude to the editorial team at O'Reilly, who guided me through the extensive process of writing a technical book. A special thanks to Sara Hunter for being an invaluable editorial sounding board as I embarked on writing my second book.

Lastly, I would like to express my appreciation to the entire Excel community for being such a welcoming and inspiring group. This spreadsheet program has opened up more opportunities and introduced me to more incredible people than I could have ever imagined. I hope this book contributes in some small way to your own remarkable journey with Excel.

PART I

Data Cleaning and Transformation with Power Query

CHAPTER 1

Tables: The Portal to Modern Excel

Excel boasts an extensive array of analytical tools, which can make it challenging to determine the best starting point. However, a fundamental step is mastering the Excel table. This chapter delves into the essential elements of Excel tables, acting as a conduit to Power Query, Power Pivot, and additional tools highlighted in this book. It further underscores the significance of organizing data within a table meticulously. To engage with this chapter's content, navigate to *ch_01.xlsx* in the *ch_01* folder located within the companion repository of the book.

Creating and Referring to Table Headers

A dataset without column headers is practically useless, as it lacks meaningful context for interpreting what each column measures. Unfortunately, it's not uncommon to encounter datasets that break this cardinal rule. Excel tables act as a valuable reminder that the quality of a dataset hinges on the presence of clear and informative headers.

In the `start` worksheet of *ch_01.xlsx*, you will come across data in columns `A:F` without corresponding headers, which are currently located in columns `H:M`. This design is far less than optimal. To adjust it, click anywhere within the primary data source and proceed from the ribbon to Insert → Table → OK, as illustrated in Figure 1-1. Alternatively, you can press Ctrl+T or Ctrl+L from within the data source to launch the same Create Table dialog box.

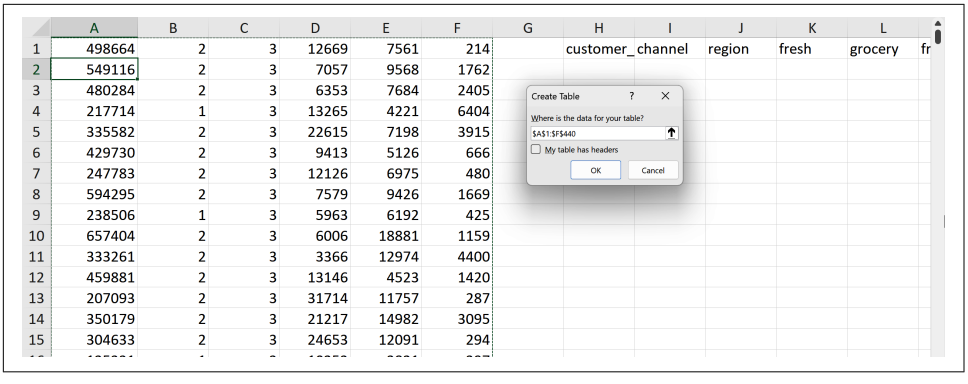


Figure 1-1. Converting the data source into a table

The Create Table dialog box automatically prompts you to specify if your data includes headers. Currently, it does not. In the absence of headers, the dataset is automatically assigned a series of header columns named `Column1`, `Column2`, and so forth.

From here, you can cut and paste the headers from columns `H:M` into the main table to clarify what is being measured in each column, such as in Figure 1-2.

	A	B	C	D	E	F
1	customer_id	channel	region	fresh	grocery	frozen
2	498664	2	3	12669	7561	214
3	549116	2	3	7057	9568	1762
4	480284	2	3	6353	7684	2405
5	217714	1	3	13265	4221	6404
6	335582	2	3	22615	7198	3915
7	429730	2	3	9413	5126	666
8	247783	2	3	12126	6975	480
9	594295	2	3	7579	9426	1669
10	238506	1	3	5963	6192	425
11	657404	2	3	6006	18881	1159
12	333261	2	3	3366	12974	4400

Figure 1-2. Excel table with headers

Header columns in Excel tables occupy a unique role in the dataset. While part of the table, they function as metadata rather than data itself. Excel tables provide the ability to programmatically distinguish between headers and data, unlike classic Excel formulas.

To see this difference in action, head to a blank cell in your worksheet and enter the equals sign. Point to cells `A1:F1` as your reference, and you'll notice that the formula becomes `Table1[#Headers]`.

You can also utilize this reference in other functions. For example, you can use `UPPER()` to dynamically convert the case of all the headers, such as in Figure 1-3.

	A	B	C	D	E	F	G
1							Formula used:
2	customer_id	channel	region	fresh	grocery	frozen	=Table1[#Headers]
3	CUSTOMER_ID	CHANNEL	REGION	FRESH	GROCERY	FROZEN	=UPPER(Table1[#Headers])
4							
5	customer_id	channel	region	fresh	grocery	frozen	
6	498664	2	3	12669	7561	214	
7	549116	2	3	7057	9568	1762	
8	480284	2	3	6353	7684	2405	
9	217714	1	3	13265	4221	6404	
10	335582	2	3	22615	7198	3915	
11	429730	2	3	9413	5126	666	
12	247783	2	3	12126	6975	480	

Figure 1-3. Excel header reference formulas

Viewing the Table Footers

Just as every story has a beginning, middle, and end, every Excel table comprises headers, data, and footers. However, footers need to be manually enabled. To do this, click anywhere in the table, navigate to Table Design on the ribbon, and select Total Row in the Table Style Options group, as in Figure 1-4.

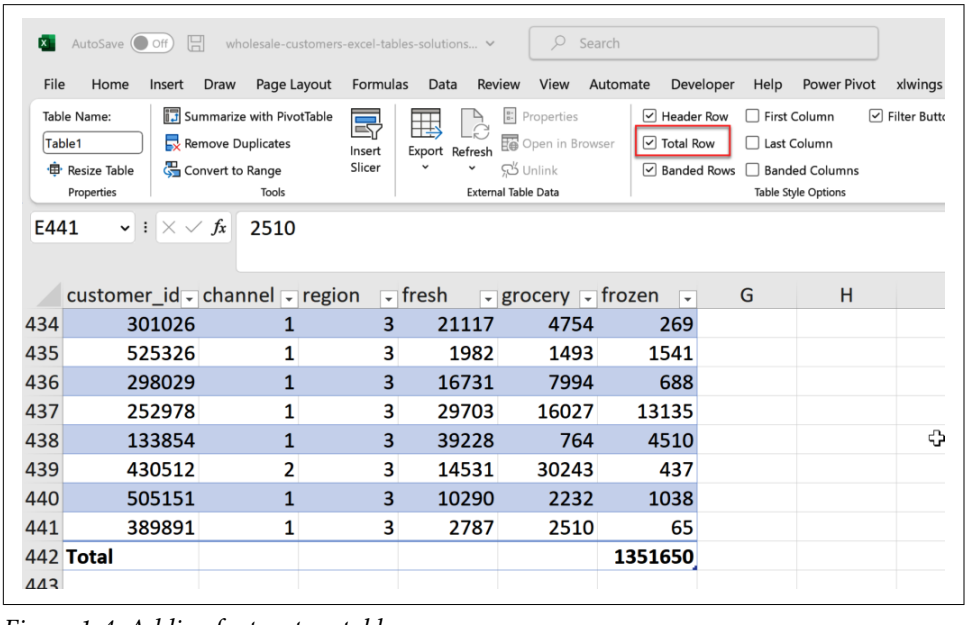
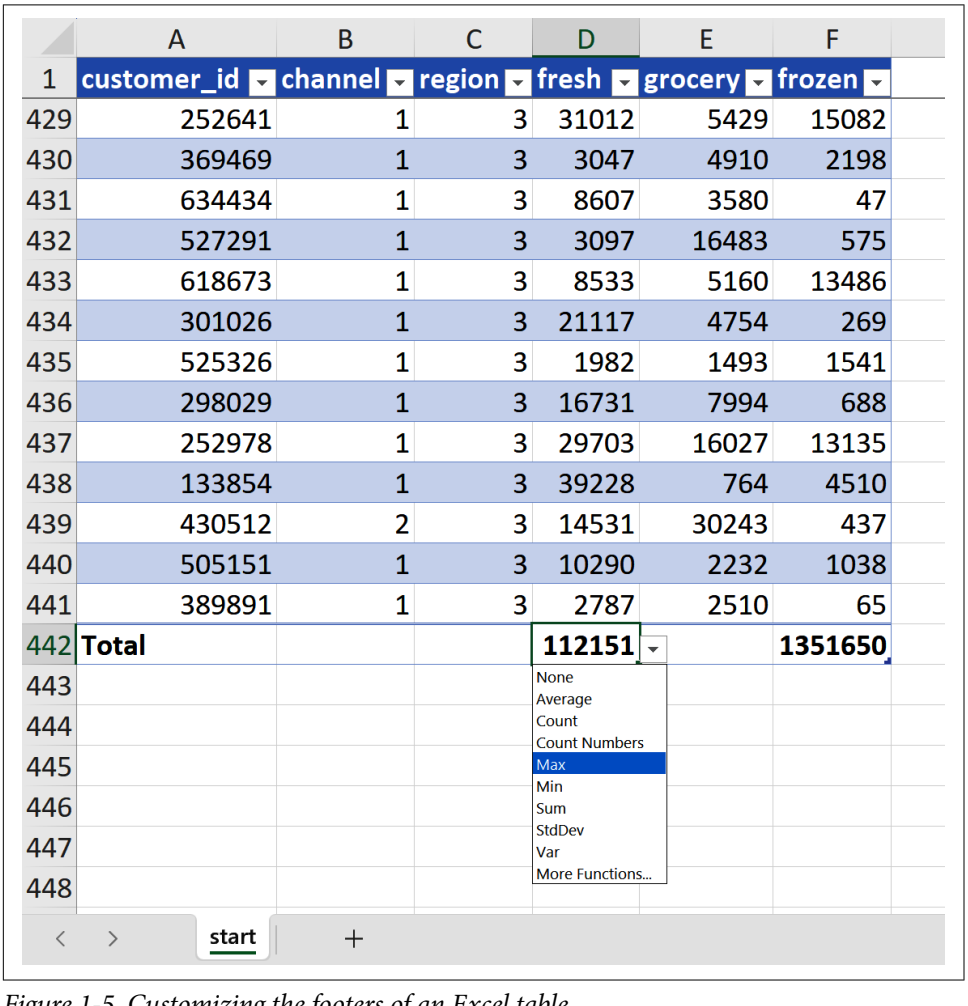


Figure 1-4. Adding footers to a table

By default, the Total Row in a table will calculate the sum of the last column in your data; in this case, `frozen`. However, you can customize this by clicking the drop-down menu on any column's footer. For instance, you can find the maximum sales amount of the `fresh` category, as in Figure 1-5.



	A	B	C	D	E	F
1	customer_id	channel	region	fresh	grocery	frozen
429	252641	1	3	31012	5429	15082
430	369469	1	3	3047	4910	2198
431	634434	1	3	8607	3580	47
432	527291	1	3	3097	16483	575
433	618673	1	3	8533	5160	13486
434	301026	1	3	21117	4754	269
435	525326	1	3	1982	1493	1541
436	298029	1	3	16731	7994	688
437	252978	1	3	29703	16027	13135
438	133854	1	3	39228	764	4510
439	430512	2	3	14531	30243	437
440	505151	1	3	10290	2232	1038
441	389891	1	3	2787	2510	65
442	Total			112151		1351650
443				None		
444				Average		
445				Count		
446				Count Numbers		
447				Max		
448				Min		
				Sum		
				StdDev		
				Var		
				More Functions...		

start

Figure 1-5. Customizing the footers of an Excel table

Table 1-1 summarizes the key formula references for the major components of Excel tables, assuming the table is named `Table1`.

Table 1-1. Summary of Excel table formula references

Formula	What it refers to
`=Table1[#Headers]`	Table headers
`=Table1`	Table data
`=Table1[#Totals]`	Table footers
`=Table1[#All]`	Table headers, data, and footers

As you progress in your Excel table skills, you'll discover additional helpful formula references that rely on the fundamental structure of headers, body, and footers.

Naming Excel Tables

Excel tables offer the advantage of enforcing the use of named ranges, which promotes a more structured approach to working with data. Although referring to `Table1` is an improvement over using cell coordinates like `A1:F22`, it is better to choose a descriptive name that reflects what the data represents.

To accomplish this, go to the Formulas tab on the ribbon, select Name Manager in the Defined Names group, and choose Edit for the `Table1` name. Change the name to `sales`, and then click OK. Figure 1-6 shows what your Name Manager should look like after making this change.

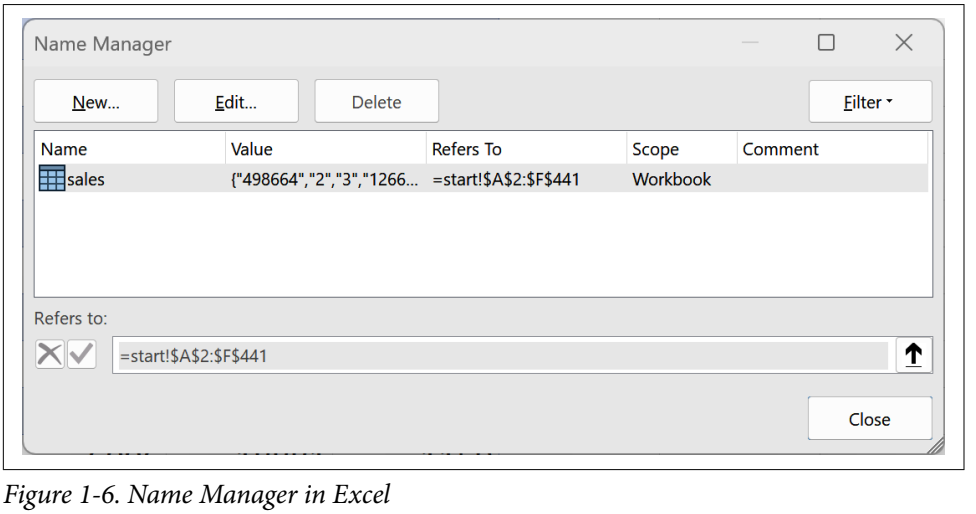


Figure 1-6. Name Manager in Excel

Once you close the Name Manager, you'll notice that all references to `Table1` have been automatically updated to reflect the new name: `sales`.

Formatting Excel Tables

As an Excel user, you know the importance of presenting data in an appealing format. Tables can be a game changer, instantly enhancing the visual appeal of your worksheet. With tables, you can easily add banded rows, colored headers, and more. To customize the look and feel of your table, click anywhere inside your table and head to Table Design in the ribbon. Take a look at Figure 1-7 for various options, such as changing table colors or toggling Banded Rows on and off.

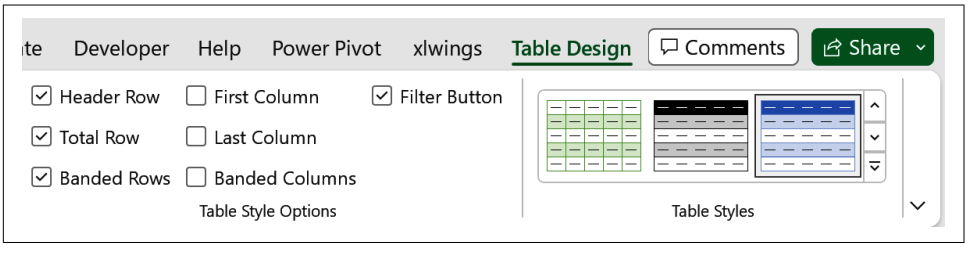


Figure 1-7. Table Design customization options

Updating Table Ranges

With Excel tables, the issue of totals becoming incorrect when data is added or removed is effectively resolved. Thanks to the use of structured references, formulas adapt seamlessly to changes in the data, ensuring accuracy. Furthermore, the total at the bottom of the table is automatically updated to reflect these changes, and it can be easily excluded from external references, maintaining the integrity of your calculations.

Calculate the sum of the `fresh` column using the structured formula `=SUM(sales[fresh])`. Microsoft's IntelliSense facilitates this process by allowing you to complete names efficiently as you type. Experiment with adding or removing rows, or modifying the `fresh` data in the `sales` table. You'll observe that the function to calculate total `fresh` sales updates dynamically and maintains consistent accuracy.

Referring to data by name instead of cell location minimizes potential formula issues arising from changing the table's size and placement. Tables also become crucial in preventing problems like missing data in a PivotTable when new rows are added.

Organizing Data for Analytics

While tables are valuable, an even more significant aspect of ensuring effortless and accurate data analysis lies in storing data in the appropriate shape.

Examine the `sales` table as an example. When attempting to create a PivotTable to calculate total sales by region, the format in which the data is stored presents a challenge. Ideally, all sales information should be consolidated into a single column. However, in the current setup, there is a distinct sales column for each department: `fresh`, `grocery`, and `frozen`. Excel does not recognize that these columns all represent the same metric, namely sales.

The reason this and many other datasets get difficult to analyze is that they are not stored in a format conducive to analysis. The rules of tidy data offer a solution. While Hadley Wickham offers three rules in his 2014 paper by the same name, this book focuses on the first: *each variable forms a column.*

The `sales` dataset violates the rule of tidy data by having multiple entries for the same variable, `field`, across different departments within each row. A helpful rule of thumb is that if multiple columns are measuring the same thing, the data is likely not tidy. By transforming the data into a tidy format, analysis becomes significantly simpler.

In Figure 1-8, you can see a comparison of the dataset before and after the transformation, highlighting the improved tidiness and ease of analysis. In Chapter 4, you will learn how to perform this fundamental transformation on a dataset with just a few clicks. In the meantime, you can explore the `sales-tidy` worksheet available in *ch01_solutions.xlsx*, which has already been transformed. Take a look to see firsthand how much simpler it is to obtain total sales by region now.

Before:

customer_id	channel	region	fresh	grocery	frozen
498664	2	3	12669	7561	214
549116	2	3	7057	9568	1762
480284	2	3	6353	7684	2405
217714	1	3	13265	4221	6404
335582	2	3	22615	7198	3915
429730	2	3	9413	5126	666
247783	2	3	12126	6975	480
594295	2	3	7579	9426	1669
238506	1	3	5963	6192	425
657404	2	3	6006	18881	1159
333261	2	3	3366	12974	4400
459881	2	3	13146	4523	1420
207093	2	3	31714	11757	287
350179	2	3	21217	14982	3095
304633	2	3	24653	12091	294
125231	1	3	10253	3821	397
126155	2	3	1020	12121	134

After:

customer_id	channel	region	department	sales
498664	2	3	fresh	12669
498664	2	3	grocery	7561
498664	2	3	frozen	214
549116	2	3	fresh	7057
549116	2	3	grocery	9568
549116	2	3	frozen	1762
480284	2	3	fresh	6353
480284	2	3	grocery	7684
480284	2	3	frozen	2405
217714	1	3	fresh	13265
217714	1	3	grocery	4221
217714	1	3	frozen	6404
335582	2	3	fresh	22615
335582	2	3	grocery	7198
335582	2	3	frozen	3915
429730	2	3	fresh	9413
429730	2	3	grocery	5126

Figure 1-8. Wholesale customers, before and after tidying

Conclusion

This chapter has laid the groundwork for utilizing Excel tables effectively. For an in-depth exploration of maximizing the potential of tables, including the application of structured references to formulate calculated columns, refer to *Excel Tables: A Complete Guide for Creating, Using, and Automating Lists and Tables* by Zack Barresse and Kevin Jones (Holy Macro! Books, 2014). Additionally, this chapter delved into the meticulous organization of data, a fundamental aspect of any successful data analysis project in Excel. Chapter 2 offers an introduction to simplifying data transformation with Power Query.

Exercises

To create, analyze, and manipulate data in Excel tables, follow the exercises using the `penguins` dataset located in *ch_01_exercises.xlsx* in the *exercises\ch_01_exercises* folder in the book's companion repository:

1. Convert the data to a table named `penguins`.
2. Utilize a formula reference to capitalize each column header.
3. Generate a new column called `bill_ratio` by dividing `bill_length_mm` by `bill_depth_mm`.
4. Include a total row to calculate the average `body_mass_g`.
5. Remove the banded row styling from the table.

For the solutions, refer to the *ch_01_exercise_solutions.xlsx* file located in the same folder.

CHAPTER 2

First Steps in Excel Power Query

In Chapter 1, tables were introduced as the gateway to modern Excel analytics. The following chapters of Part I delve deeper into the modern Excel toolkit, particularly Power Query. This tool addresses many of Excel's traditional limitations and offers a user-friendly, low-code environment.

What Is Power Query?

Power Query is a data connection technology that enables users to easily connect to, combine, and refine data from a wide range of sources from within Excel. Initially introduced as an add-in, it has become a core feature of modern Excel, significantly simplifying the data import and cleaning process. Power Query offers a user-friendly interface for performing complex data manipulations, such as merging tables, transforming data formats, and aggregating information, without requiring advanced programming skills.

Power Query as Excel Myth Buster

Analysts act as myth busters, challenging assumptions and revealing truths. Even in the domain of Excel, a legendary business tool, myths can arise. However, many of these myths no longer hold true. Power Query has successfully debunked numerous claims, positioning itself as the ultimate myth buster for Excel. The next section will refute common claims about Excel using Power Query.

"Excel Is Not Reproducible"

It's a common scenario: you find yourself under pressure, facing imminent deadlines and an intrusive manager, while attempting to revise a report from the previous week. The original creator of the report is unavailable, leaving you perplexed about how it

was assembled. The workbook appears to be a jumbled mess of deleted columns and altered values, making it difficult to decipher the necessary actions to take.

Reproducibility in computation allows the user to consistently achieve the same results by using identical inputs and processes. A workbook falls short of this goal when error-prone steps, complex calculations, or other elements that introduce uncertainty render consistent outcomes unattainable each time the file is opened.

The lack of reproducibility in traditional Excel has become a significant point of criticism for the software. This concern has led many technical professionals to be cautious about using Excel altogether, fearing that a single deleted column or a hard-coded cell could compromise the integrity of their results.

The decision to completely abandon Excel based on its past limitations is misguided. Excel now offers a solution for reproducibility through Power Query. With Power Query, users can create a duplicate of the source data, apply consistent transformation steps, and document each action in the Applied Steps list. This approach ensures reproducibility and eliminates the need to track data cleanup actions, addressing the concerns previously associated with Excel's lack of reproducibility.

VBA and Reproducibility

Experienced Excel users may question the assertion that classic Excel lacks robust reproducibility features because it includes Visual Basic for Applications (VBA). While VBA does provide a comprehensive scripting language that allows for auditing and debugging Excel results, its steep learning curve has limited its usability among most Excel users. Furthermore, Microsoft has made minimal updates to VBA, instead introducing alternative languages like Python, which will be explored in Chapter 12.

Irrespective of the ongoing debates regarding the best scripting language for Excel, ensuring a reproducible workflow should not be restricted to those capable of coding. It should be accessible to users with diverse technical backgrounds, and this is precisely what Power Query accomplishes.

"Excel Does Not Have a True null"

In relational databases, the concept of a missing or `null` value, representing unknown or unspecified data, is well known. However, Excel lacks a reserved keyword for `null` values, resulting in storage and handling challenges. Users may adopt different approaches for representing missing values in Excel, such as leaving them blank or hard-coding values like `NA`. This discrepancy makes it challenging to identify genuinely unknown values versus those that are actually equal to zero or intentionally left blank.

To overcome this limitation, Power Query introduces a dedicated `null` value to represent missing data. This enhancement facilitates precise profiling, removal, and replacement of missing values, ensuring accuracy and reproducibility.

"Excel Can't Process More Than 1,048,576 Rows"

Another frequently used argument against Excel is its perceived limitation when dealing with "big data." Detractors assert that with its worksheet maximum of approximately one million rows, Excel falls short in the era of massive datasets.

The solution lies, once again, in Power Query, which can easily import and process millions of rows and more. While it is true that Excel itself cannot handle more than a million rows, Power Query empowers users to aggregate and summarize the data within its editor before loading the results into an Excel worksheet.

For a compelling demonstration of surpassing the alleged million-row limit in Excel, check out this post from analytics consultant Orlando Mézquita on analyzing fifty million rows using Excel Power Query.

Power Query as Excel's ETL Tool

In the world of tech, many terms may seem impossibly complex at first glance. They may even be disguised by confusing acronyms. Upon closer examination, however, these concepts reveal their simplicity.

One such term is *ETL*, which stands for "extract, transform, load." Database administrators and data engineers often passionately discuss their "ETL pipelines" and "ETL software." It may give the impression that only certified data geeks can handle these tasks.

Power Query democratizes the ETL process by integrating it directly into an Excel spreadsheet. Don't let tech purists intimidate you! The essence of ETL is inherent in its name and can be accomplished using Excel.

The following sections present a step-by-step overview of this process. To follow along, open *ch_02.xlsx* from the book's resources in the *ch_02* folder.

Extract

The initial step in ETL is to "extract" the data from an external source. Power Query has the ability to connect to a diverse range of data sources, extending beyond Excel workbooks. Here are some examples of the sources it can connect to:

- Text and CSV files
- Relational databases like Oracle, Microsoft SQL Server, or SQLite

- SharePoint
- XML, HTML, and web data

However, for this demonstration, the data is conveniently located within the Excel workbook itself.

To begin, extract data from the `sales` table in the `sales` worksheet of *ch_02.xlsx*. Click anywhere inside the table, then go to Data → Get & Transform Data → From Table/Range, as shown in Figure 2-1.

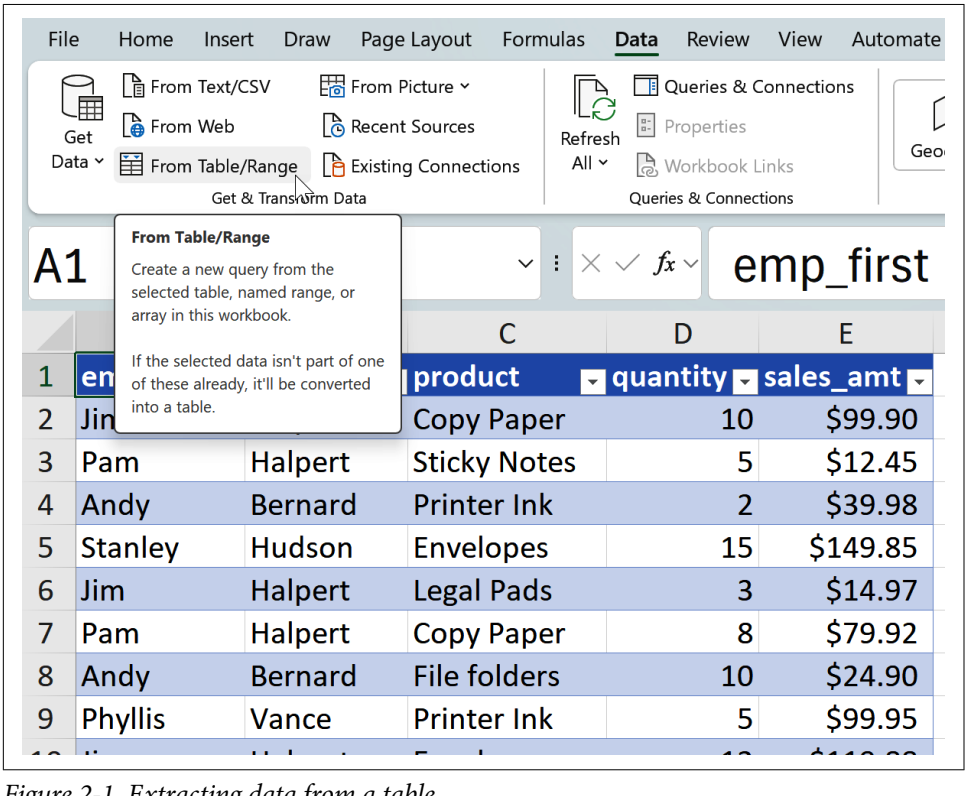


Figure 2-1. Extracting data from a table

The requirement of having this data in a table for utilizing Power Query is in part why we dedicated Chapter 1 to discussing tables. Tables serve as an essential gateway to accessing modern Excel tools.

While it may seem odd to "connect to and extract" data that already exists in your workbook, this approach is justified by Power Query's ability to preserve the raw data in its original form. Even when the raw data is within the same workbook as the analysis, it is advisable to extract a subset of the data (the "E" in ETL) to proceed with any analysis.

Transform

The next step is to connect to this data and perform the necessary transformations (the "T" in ETL).

Data transformation encompasses various tasks necessary to render data usable, such as:

- Sorting or filtering rows
- Adding, dropping, renaming, or calculating columns
- Merging or reshaping data sources

When you load your table into Power Query, the Power Query Editor pops up, offering numerous options for data cleaning and transformation. It can be quite overwhelming, especially if you're used to the classic Excel environment. However, we'll tackle this program step by step throughout the rest of Part I.

You will learn how to perform several data cleaning tasks later in the book. For now, a straightforward data transformation can be achieved by adding an index column. To do this, go to Add Column on the Power Query ribbon, then Index Column → From 1, as shown in Figure 2-2.

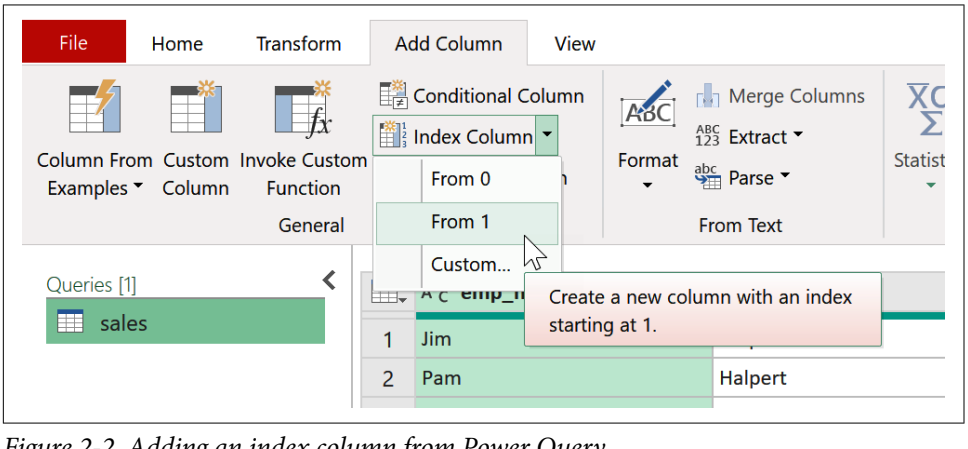


Figure 2-2. Adding an index column from Power Query

Load

Finally, in the Power Query Editor, navigate to the Home tab and choose Close & Load. This action will load the slightly transformed data extract into another Excel table and on a new worksheet, as seen in Figure 2-3.

	A	B	C	D	E	F
1	emp_first	emp_last	product	quantity	sales_amt	Index
2	Jim	Halpert	Copy Paper	10	99.9	1
3	Pam	Halpert	Sticky Notes	5	12.45	2
4	Andy	Bernard	Printer Ink	2	39.98	3
5	Stanley	Hudson	Envelopes	15	149.85	4
6	Jim	Halpert	Legal Pads	3	14.97	5
7	Pam	Halpert	Copy Paper	8	79.92	6
8	Andy	Bernard	File folders	10	24.9	7
9	Phyllis	Vance	Printer Ink	5	99.95	8
10	Jim	Halpert	Envelopes	12	119.88	9
11	Pam	Halpert	Legal Pads	7	17.43	10
12	Andy	Bernard	Copy Paper	4	39.96	11
13	Jim	Halpert	Printer Ink	8	79.92	12
14	Phyllis	Vance	Envelopes	15	74.85	13
15	Andy	Bernard	Legal Pads	3	59.97	14
16	Stanley	Hudson	Rubber Bands	60	14.94	15
17						

Figure 2-3. Loading data from Power Query to an Excel table

Congratulations on completing an entire ETL job:

- You extracted the raw data from an Excel table.
- You transformed the data using the Power Query Editor.
- You loaded the results back into Excel.

A Tour of the Power Query Editor

Now that you've walked through a very simple example of the entire Power Query ETL process, let's take a closer look at the Power Query Editor. For this demonstration, open the `penguins` worksheet, also found in *ch_02.xlsx*.

To get started, load the `penguins` table into Power Query. If you need a refresher on how to do this, you can refer back to the previous section. The resulting Power Query Editor should look like Figure 2-4.

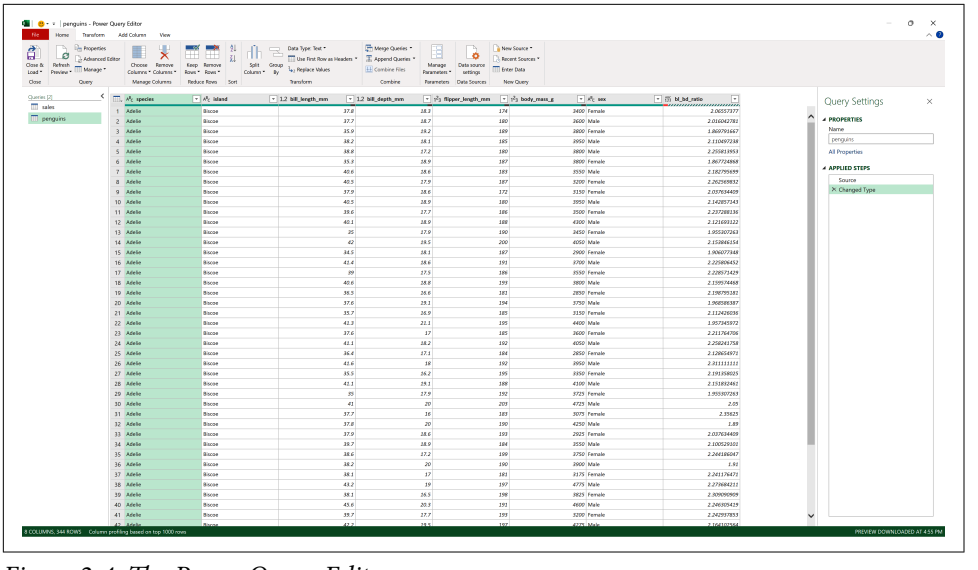


Figure 2-4. The Power Query Editor

Take a moment to delve deeper and appreciate the distinctive environment you're in. It bears a resemblance to Excel with its ribbon interface, yet it operates as an independent, standalone program. The subsequent section will explore its various elements.

The Ribbon Menu

At the top of the interface, you will notice a ribbon menu closely modeled after Excel's familiar interface, as shown in Figure 2-5.

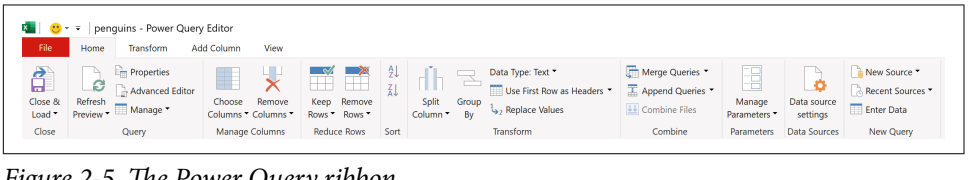


Figure 2-5. The Power Query ribbon

You should see four tabs on the ribbon: Home, Transform, Add Column, and View:

Home

Just like with regular Excel, the Home tab consists of the most fundamental operations in Power Query, such as selecting rows, deleting columns, and more. However, the Home tab in Power Query focuses on essential data transformation and cleaning tasks, unlike regular Excel's formatting features.

Transform

The Transform tab offers additional options for data cleaning and transformation. You'll have the opportunity to try many of these features in the following chapters.

Add Column

This tab is dedicated to creating new columns through a variety of methods. In "Transform" on page 17, you used this tab to add an index column to the data. In Chapter 4, you'll use it to create calculated columns.

View

The View tab customizes the display of the Power Query Editor. To begin, click on Formula Bar in the Layout group. This action will reveal a formula bar above your dataset, similar to Excel, as shown in Figure 2-6.

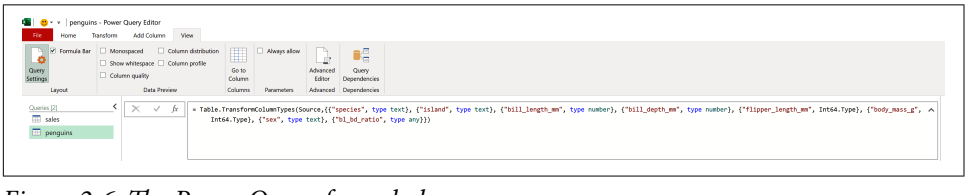


Figure 2-6. The Power Query formula bar

The formula displayed in the Power Query formula bar may appear different from typical Excel functions because it is written in the M programming language, which is specifically designed for Power Query. As you adjust your query through the Power Query Editor's point-and-click interface, the underlying M code is altered accordingly. This allows for debugging, customization, or sharing of results.

Although the presence of the formula bar might imply that mastering a complex programming language is crucial for leveraging Power Query's capabilities, this is not necessarily true. The majority of everyday tasks can be efficiently completed using the Home tab and other features, bypassing the need for M coding. This book focuses exclusively on these point-and-click options, making the formula bar redundant for our purposes.

This book won't use the formula bar in the Power Query Editor for future demonstrations. You can hide it by unchecking Formula Bar in the Layout group on the View tab.

To explore writing your own M code in Power Query, start by checking out the Advanced Editor. Access it by going to the View tab in the Power Query Editor and clicking on Advanced Editor in the Advanced group. This window displays your entire query in a single, comprehensive view.

Queries

Shift your attention from the ribbon to the list of queries on the upper left of the Editor. Here, you'll find the imported data sources, which you can toggle between to view. While both sources reside within this workbook, keep in mind that Power Query supports a wide range of data sources, including *.csv* files, databases, web pages, and more.

To perform actions on a specific query, right-click on its name, such as `penguins`, and a menu with various options will appear, as shown in Figure 2-7. These options include renaming the query, deleting it, and more.

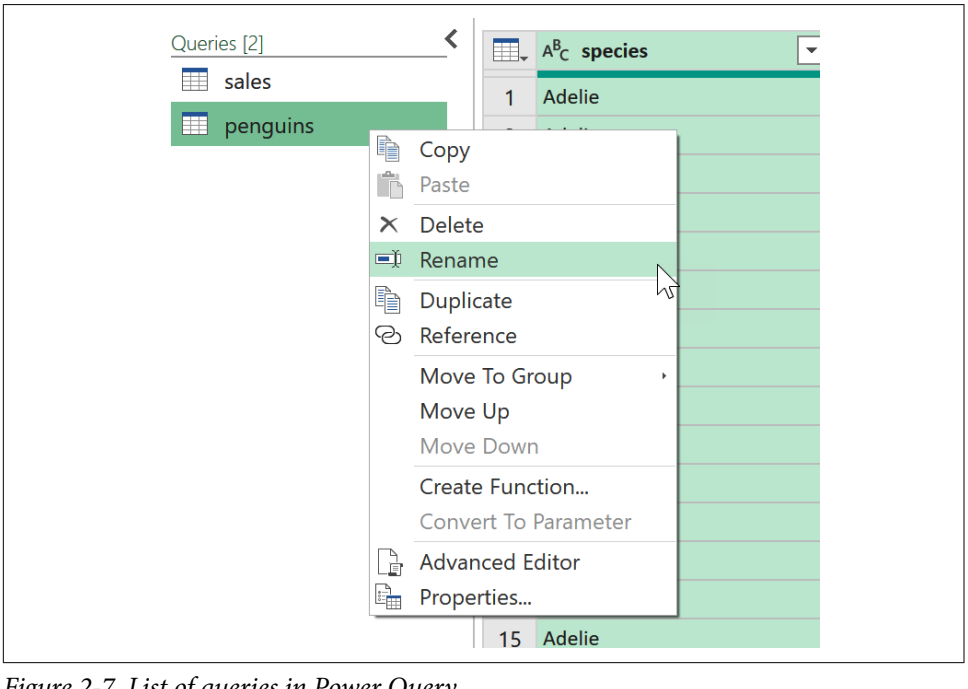


Figure 2-7. List of queries in Power Query

Power Query offers a plethora of right-click options, so don't hesitate to explore and try them out.

The Imported Data

Now, shift your focus to the component occupying most of the Editor's space: the data itself. Unlike in Excel, where you have the freedom to manipulate the data by, for example, hiding columns or inserting formulas, Power Query imposes editing restrictions.

To paraphrase lyrics of a song by The Police, Power Query diligently observes and tracks every step and movement you make with your query. Randomly hard-coding formulas or hiding columns at will is not allowed. All actions must be executed programmatically within Power Query's framework.

Consider the straightforward task of deleting a column. To remove `island` from the `penguins` dataset, simply right-click on the column label and choose "Remove," as shown in Figure 2-8.

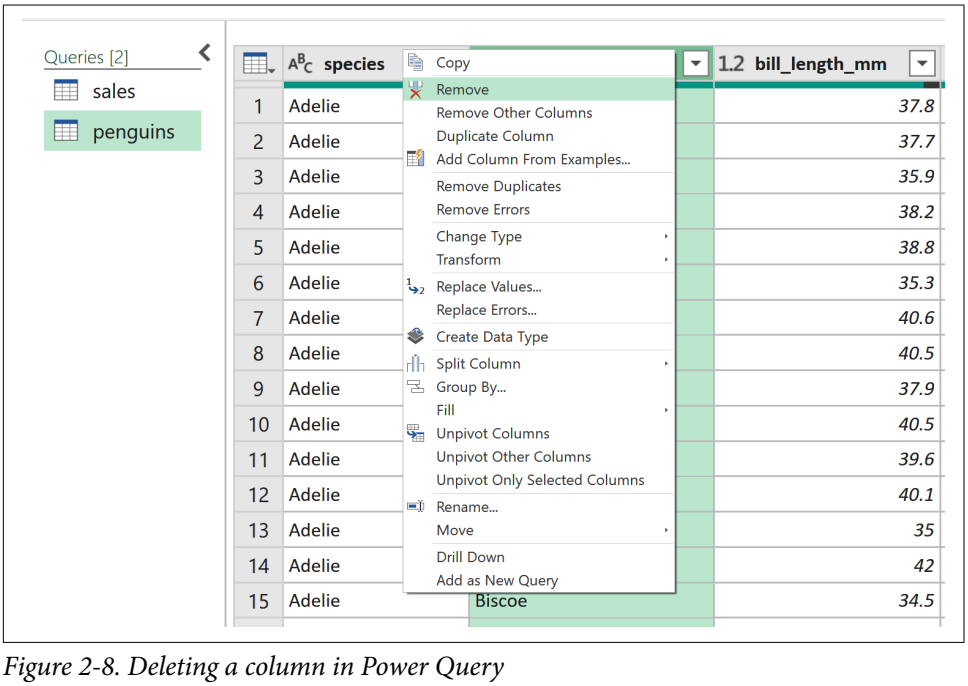


Figure 2-8. Deleting a column in Power Query

The column has been permanently removed from the dataset…or has it? To understand how data changes work in Power Query, take a look at the Applied Steps list to the right of the data, as shown in Figure 2-9.

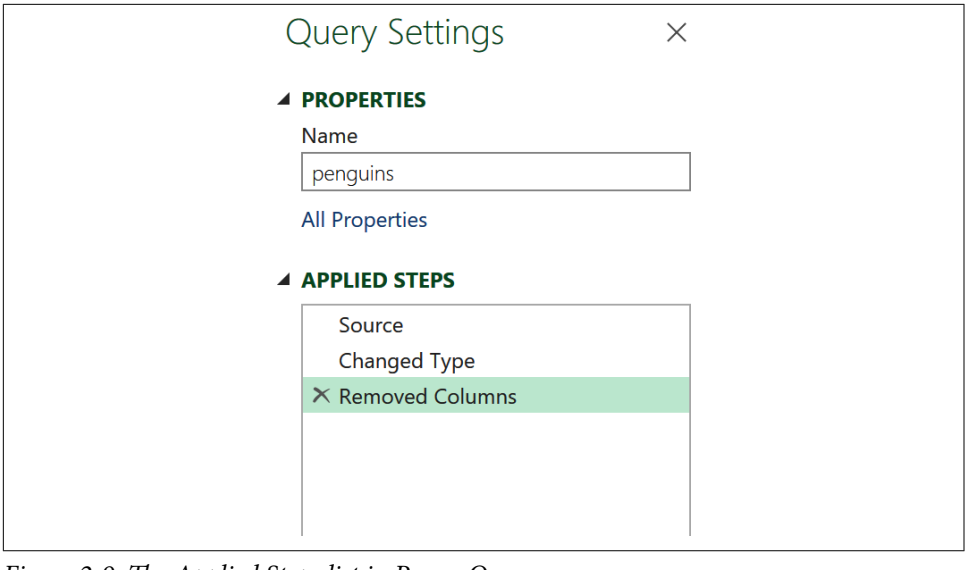


Figure 2-9. The Applied Steps list in Power Query

Power Query meticulously records every action, including deletions, in the Applied Steps list, displayed alongside your data. This ensures transparency and traceability of all operations.

To be more precise, this deletion is documented as the third step labeled Removed Columns in the Applied Steps section. The initial step is connecting to the data, referred to as the Source. The second step, Changed Type, involves setting data types for the table. Unlike Excel, Power Query requires every value within a column to be of the same type. This book will largely rely on Power Query's automatic data type casting. You can learn more about Power Query data types in Microsoft's official documentation.

By clicking on any of the steps in the Applied Steps list, you can revisit the data as it appeared at that specific point in time. For instance, if you click on Changed Type (the step prior to the column removal), the `island` column will reappear in your Editor view.

To reinforce your learning from earlier in this chapter, add an index column starting at 1 to the dataset with the Changed Type step still highlighted. You will receive a message to confirm that you wish to insert an intermediate step into the query, as in Figure 2-10.

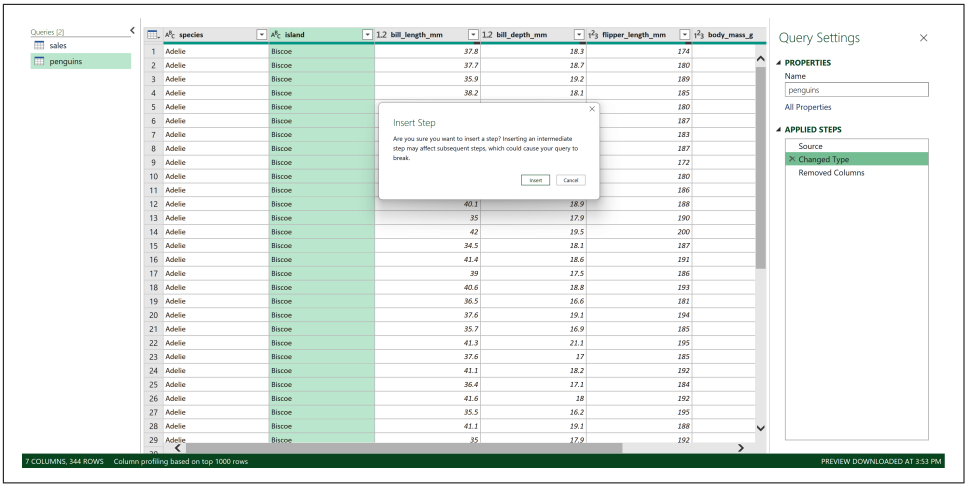


Figure 2-10. Inserting an intermediate step into Power Query

After clicking Insert, you will notice that Added Index is positioned before Removed Columns in the Applied Steps list, even though it was added at a later point in time. This allows for effortless modifications to the query as requirements evolve or new steps are added to the workflow.

You can perform various actions on these steps, such as deleting or renaming them. Suppose you wish to reintroduce the `island` column that was previously deleted in the query. You have two options: either click the X icon located to the left of the Removed Columns query step, or right-click on that same step to access a menu that lets you delete, rename, and reorder steps, among other tasks.

Although the Applied Steps list offers flexibility, it lacks a cherished feature of classic Excel: the ability to undo actions. Once a step is deleted, there is no built-in option to undo the deletion. Because most of these steps can be easily replicated, it is often just as convenient to manually repeat the process instead of relying on an undo button.

Exiting the Power Query Editor

After creating the desired query in the editor, it's time to exit Power Query and return to the regular Excel workbook. Earlier in this chapter, you saw that clicking directly on Close & Load on the Home tab of the Power Query Editor loads the result of the query to an Excel table.

There are other load options, which you can view by clicking the drop-down button next to Close & Load, then selecting Close & Load To. You should now see the dialog in Figure 2-11.

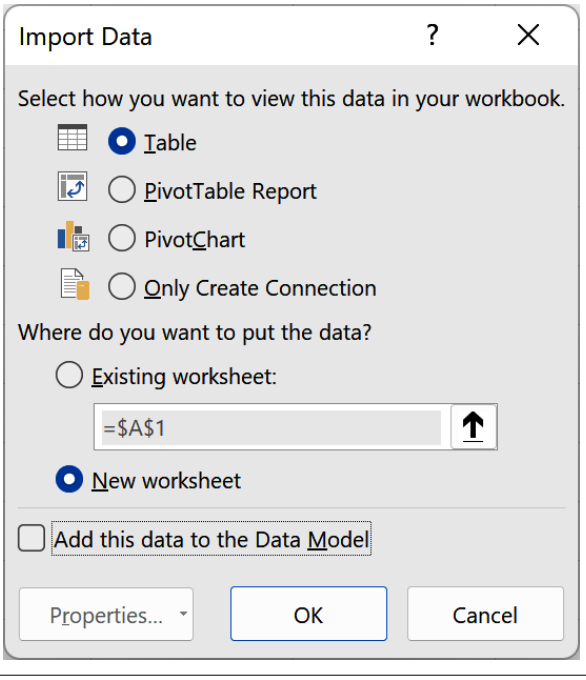


Figure 2-11. Power Query loading options

Your first decision is whether to load the data into a table, a PivotTable (referred to here as a PivotTable Report), a PivotChart, or to choose the connection-only option. When loading to connection only, the results of your query will not be loaded into Excel, but the query itself will remain accessible in the Power Query Editor.

If you choose to load the data into your workbook, you can either place it in a new worksheet or an existing one.

Additionally, you have the option to add the data to the Data Model. This enables you to construct a relational data model and leverage advanced reporting capabilities in your workbook. In Part II of this book, you'll be introduced to data models, Power Pivot, and DAX.

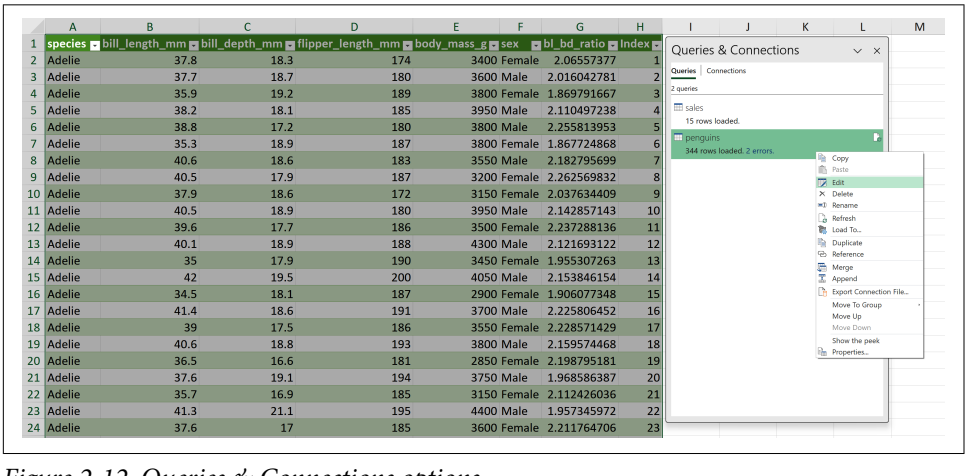
Choose whichever option you'd like for this practice exercise, then click OK.

Returning to the Power Query Editor

To resume transforming your data in Power Query or to change the way you loaded your query, head to the Data tab on the Excel ribbon, then select Queries & Connections. Look for your `penguins` query in the pane that appears on the right side of the window.

You'll see that Power Query is reporting two errors in this file; these will be identified and dealt with soon.

Right-click on `penguins` and you'll see several options for working with this query. To change how your query is loaded to Power Query, choose Load To. To go back to the Power Query Editor, choose Edit, as shown in Figure 2-12.



	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	bl_bd_ratio	Index
2	Adelie	37.8	18.3	174	3400	Female	2.06557377	1
3	Adelie	37.7	18.7	180	3600	Male	2.016042781	2
4	Adelie	35.9	19.2	189	3800	Female	1.869791667	3
5	Adelie	38.2	18.1	185	3950	Male	2.110497238	4
6	Adelie	38.8	17.2	180	3800	Male	2.255813953	5
7	Adelie	35.3	18.9	187	3800	Female	1.867724868	6
8	Adelie	40.6	18.6	183	3550	Male	2.182795699	7
9	Adelie	40.5	17.9	187	3200	Female	2.262569832	8
10	Adelie	37.9	18.6	172	3150	Female	2.037634409	9
11	Adelie	40.5	18.9	180	3950	Male	2.142857143	10
12	Adelie	39.6	17.7	186	3500	Female	2.237288136	11
13	Adelie	40.1	18.9	188	4300	Male	2.121693122	12
14	Adelie	35	17.9	190	3450	Female	1.955307263	13
15	Adelie	42	19.5	200	4050	Male	2.153846154	14
16	Adelie	34.5	18.1	187	2900	Female	1.906077348	15
17	Adelie	41.4	18.6	191	3700	Male	2.225806452	16
18	Adelie	39	17.5	186	3550	Female	2.228571429	17
19	Adelie	40.6	18.8	193	3800	Male	2.159574468	18
20	Adelie	36.5	16.6	181	2850	Female	2.198795181	19
21	Adelie	37.6	19.1	194	3750	Male	1.968586387	20
22	Adelie	35.7	16.9	185	3150	Female	2.112426036	21
23	Adelie	41.3	21.1	195	4400	Male	1.957345972	22
24	Adelie	37.6	17	185	3600	Female	2.211764706	23

Figure 2-12. Queries & Connections options

Data Profiling in Power Query

So far, we've highlighted Power Query as a robust ETL tool designed to streamline the data cleaning process. Yet, embarking on the task of cleaning a dataset without a clear understanding of what exactly makes it dirty is an impractical approach.

To mitigate this challenge, Power Query is equipped with an array of data profiling techniques. This segment will explore the integral role of data profiling within Power Query, underscoring its importance in the optimization of data quality.

What Is Data Profiling?

Data profiling offers insights into data characteristics, such as missing values, value frequencies, and summary statistics. This knowledge supports informed decision making and efficient data transformations. During the profiling process, consider the following questions:

- How accurate is the data?
- Are there evident issues?
- Are the purpose and measurement of every variable and observation clear?
- Is all the necessary data available? Are there gaps?
- In Excel-based data, are there formula errors affecting results?
- Has the data been transcribed accurately?

Answering these questions allows analysts to gauge the data's health and reliability, pinpoint potential problems, and decide on data cleaning, transformation, and analysis strategies.

Exploring the Data Preview Options

Data profiling is somewhat tucked away in the Power Query Editor. To access it, navigate to the View tab on the ribbon and locate the Data Preview group, as shown in Figure 2-13. You can explore the functionality of each of the five options by toggling them on one by one.

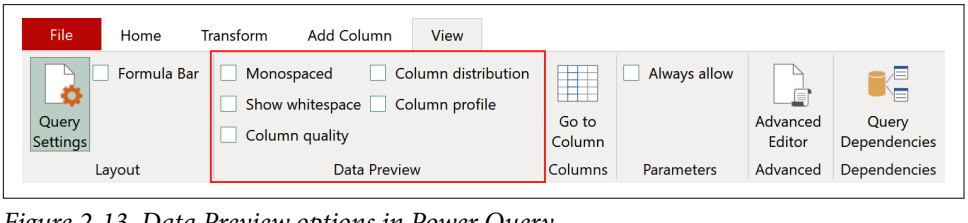


Figure 2-13. Data Preview options in Power Query

"Monospaced" and "Show whitespace"

The first two options modify the data's appearance within the Power Query Editor:

- "Monospaced" will render the data as fixed-width text.
- "Show whitespace" will reveal any leading or trailing spaces in the data

While these options are useful, particularly for identifying text that requires trimming, the true data profiling potential lies with subsequent options.

The "Column quality" and "Column distribution"

Next, select the following two options. A box will display above each column, presenting valuable insights into the data, such as the percentage of valid, error, and empty values. It also presents a visualization of the distribution of values in the column. These options provide a comprehensive overview of the data's quality and distribution, aiding in effective analysis and decision making.

What is a valid cell? When Excel refers to "valid" data, it simply means that the value is not empty and does not contain any errors. It's important to note that this definition of "valid" does not take into account the logical correctness or meaningfulness of the data. As a result, Power Query may consider nonsensical data to be "valid." For example, take row `71` of the `sex` column, as seen in Figure 2-14.

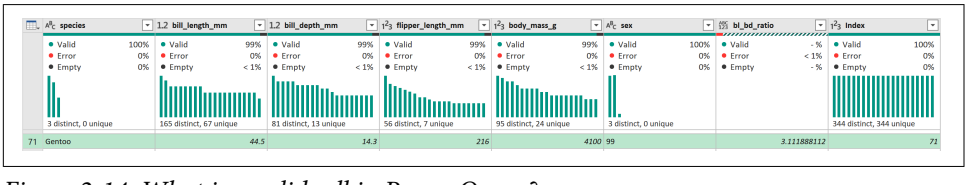


Figure 2-14. What is a valid cell in Power Query?

Missing values. Clearly, the entry of `99` is not a valid value for the `sex` column. It seems to be the result of a transcription error, where these cells were mistakenly filled with `99` instead of being left missing or `null`. To observe a genuine missing value in Power Query, head to row 296, as shown in Figure 2-15.

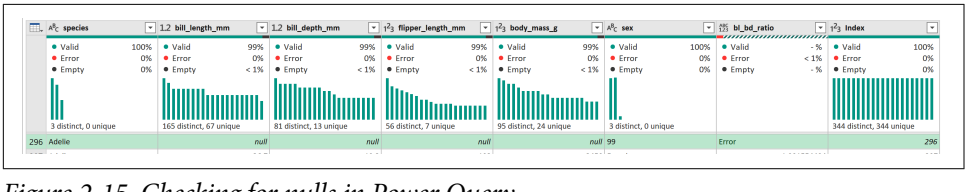


Figure 2-15. Checking for nulls in Power Query

Here you will notice several entries marked as `null`, which is the correct way to indicate an empty value in Power Query. Currently, the Column quality display shows fewer than 1% of cells categorized as "Empty" for each of these columns. Keep in mind, however, that these figures do not include incorrectly transcribed missing values.

Cell errors. To gain an understanding of the Error category, remain on row 296 and focus on the `bl_bd_ratio` column. This particular column was calculated in Excel by dividing the values in the `bill_length_mm` column by those in the `bill_depth_mm` column. However, in this specific row, the denominator of the formula was left empty,

resulting in an error. By clicking on the whitespace next to Error, you can verify that it results from a `#DIV/0` error, as shown in Figure 2-16.

Figure 2-16. Cell errors in Power Query data profiling

There are two of these calculation errors found in this column, resulting in the error message seen in Figure 2-16. This error could be resolved in a number of ways, including writing a base Excel formula that accounts for division by zero errors, or perhaps by filtering out the rows that are resulting in errors. You will learn how to filter rows and perform other operations on rows in Chapter 3.

The "Column distribution" option displays a visualization of the data distribution, along with some other information, as seen in Figure 2-17.

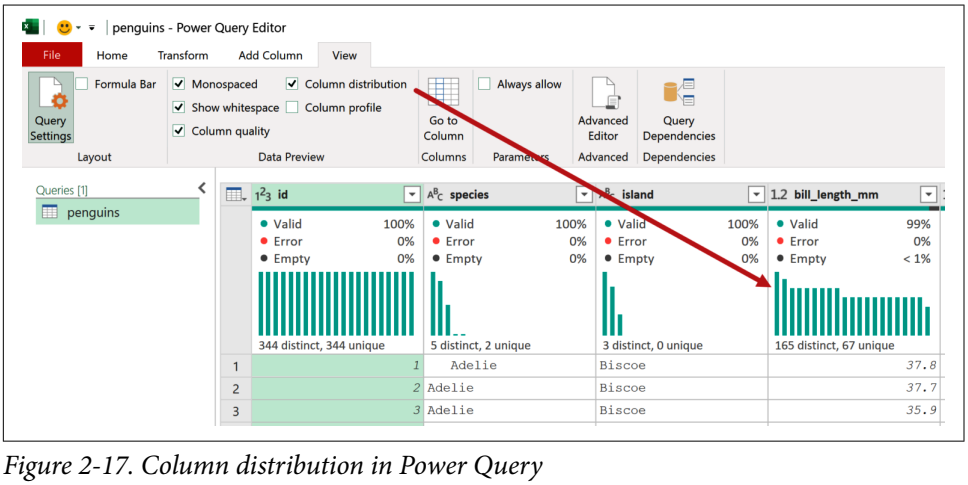


Figure 2-17. Column distribution in Power Query

However, these features are all available in the “Column profile” option, so we’ll focus there next.

Column profile. Finally, select the “Column profile” check box, choose a specific column, and navigate to the “Column profile” output beneath the dataset for detailed insights. For instance, consider the `species` column, a qualitative variable. This selection will provide a detailed breakdown of the values within the column, including a visualization that depicts the frequency of observations for each value, as seen in Figure 2-18.

Figure 2-18. Profiling a qualitative variable

For quantitative variables like `bill_depth_mm`, “Column profile” output will display additional measures such as the mean and standard deviation, as seen in Figure 2-19.

If you’re unfamiliar with qualitative and quantitative variables, think of *qualitative variables* as asking “What kind?” and *quantitative variables* as asking “How much?” or “How many?” For a deeper exploration of these variable types, refer to my book *Advancing into Analytics: From Excel to Python and R*. In Part II of this book, you’ll learn about dimensions and measures, which are similar concepts to qualitative and quantitative variables.

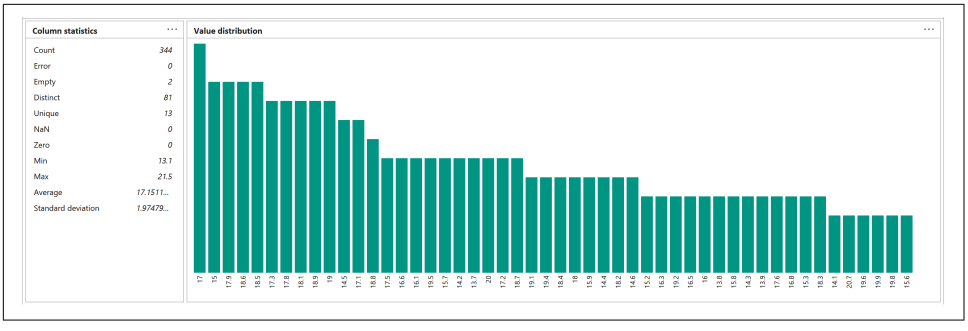


Figure 2-19. Column profiling in Power Query

Overriding the Thousand-Row Limit

If you are working with a dataset of greater than one thousand rows, make sure to include all observations in Power Query's data profiling. To do this, click toward the bottom of the editor and select "Column profiling based on entire data set," as seen in Figure 2-20.

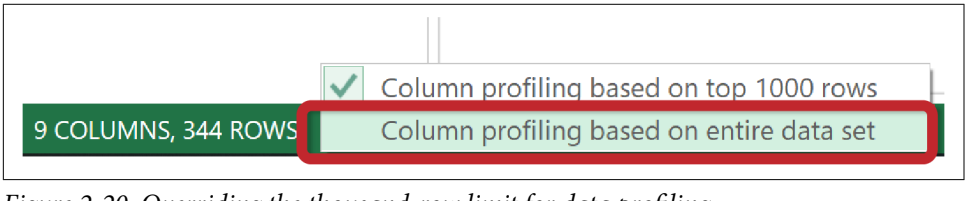


Figure 2-20. Overriding the thousand-row limit for data profiling

With Power Query's data profiling capabilities you were able to:

- Quickly spot incorrectly formatted cells
- Determine which columns contain missing values
- Visualize the distribution of each variable

Closing Out of Data Profiling

When you are finished profiling your data in Power Query, you can simply click Close & Load to return to Excel without making any changes to your query. Remember, data profiling is about checking the data *first* before making any changes to it. For the rest of the book, to cut down on clutter, we will keep the Data Preview options turned off.

Conclusion

This chapter debunked common myths associated with Excel by using Power Query as an ETL tool. It also explored both the Power Query Editor and the data profiling process. You are now ready to transform data using Power Query, which will be the focus of the remainder of Part I.

Exercises

For this chapter's exercises, explore a dataset of computer prices with Power Query. Open *ch_02_exercises.xlsx* in the *exercises\ch_02_exercises* folder in the book's companion repository. Perform the following:

1. Load the data into Power Query as a table. Name the query `computers`.
2. Add an index column starting from 1 to the data.
3. Rename the previous step in the Applied Steps list as "Added unique identifier."
4. Drag and drop the `Index` column so that it is the first column in the dataset.
5. Use Power Query's data profiling features to address the following questions. Be sure to adjust column profiling to work on the entire dataset.
   - What is the range of prices for the computers in the dataset?
   - What is the average amount of RAM in the dataset?
   - Are there any missing values in this dataset? If so, where?
6. Load the query results into an Excel PivotTable.

The solutions are available in *ch_02_exercise_solutions.xlsx* in the same folder.

CHAPTER 3

Transforming Rows in Power Query

Chapter 2 served as an introduction to Power Query's myth-busting capabilities as an ETL tool for Excel. In this and upcoming chapters of Part I, you'll have the chance to get hands-on practice with common data transformation tasks. The focus of this chapter is on rows.

Data cleaning often involves row manipulation tasks such as sorting, filtering, and removing duplicates. Traditional Excel offers interface-guided methods for these tasks, but they can be cumbersome and hard to replicate. Power Query offers a solution by enabling an auditable and repeatable data cleaning process without coding. To follow the demonstrations in this chapter, please access *ch_03.xlsx* in the *ch_03* folder of the book's repository.

In the `signups` worksheet of this workbook, your organization's party planning committee has been gathering RSVPs and wants the final list to be sorted alphabetically, with duplicates, blanks, and misprints eliminated. The committee is weary of manually sorting and removing unnecessary rows whenever new data is added. They desire a workbook that can be easily refreshed and reused as more individuals register or as new parties are scheduled.

Load this data into Power Query, naming the query `signups`. Capture all relevant rows in column `A` and confirm that your table includes headers before proceeding.

Removing the Missing Values

As mentioned in Chapter 2, Power Query provides a dedicated `null` value to represent missing values. The `signups` data contains three blank values, which may cause confusion. To eliminate them, navigate to the Home tab on the ribbon and select Remove Rows → Remove Blank Rows, as in Figure 3-1.

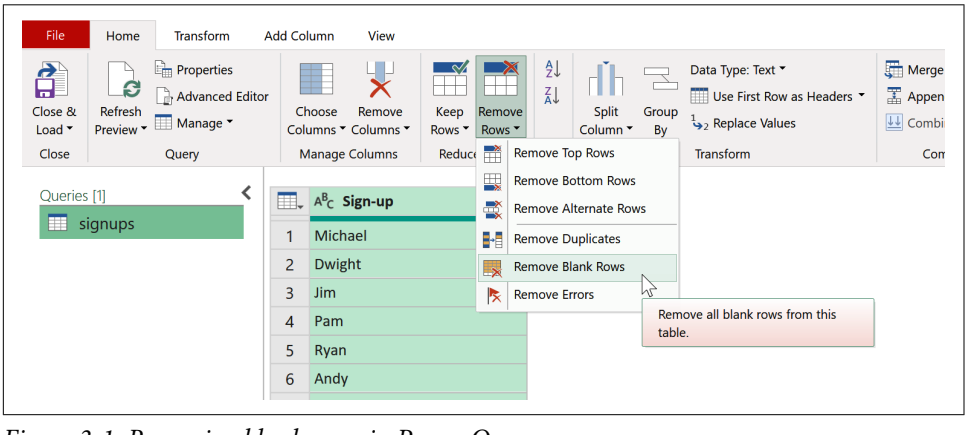


Figure 3-1. Removing blank rows in Power Query

Next, sort the list alphabetically. To do this, click the drop-down button next to the `Sign-up` column, where you'll find the sort and filter options, similar to basic Excel, as in Figure 3-2. Select Sort Ascending to sort the list alphabetically.

You may have noticed that *Phyllis* is entered multiple times in this dataset. To remove duplicates, head back to the Home tab, then select Remove Rows → Remove Duplicates.

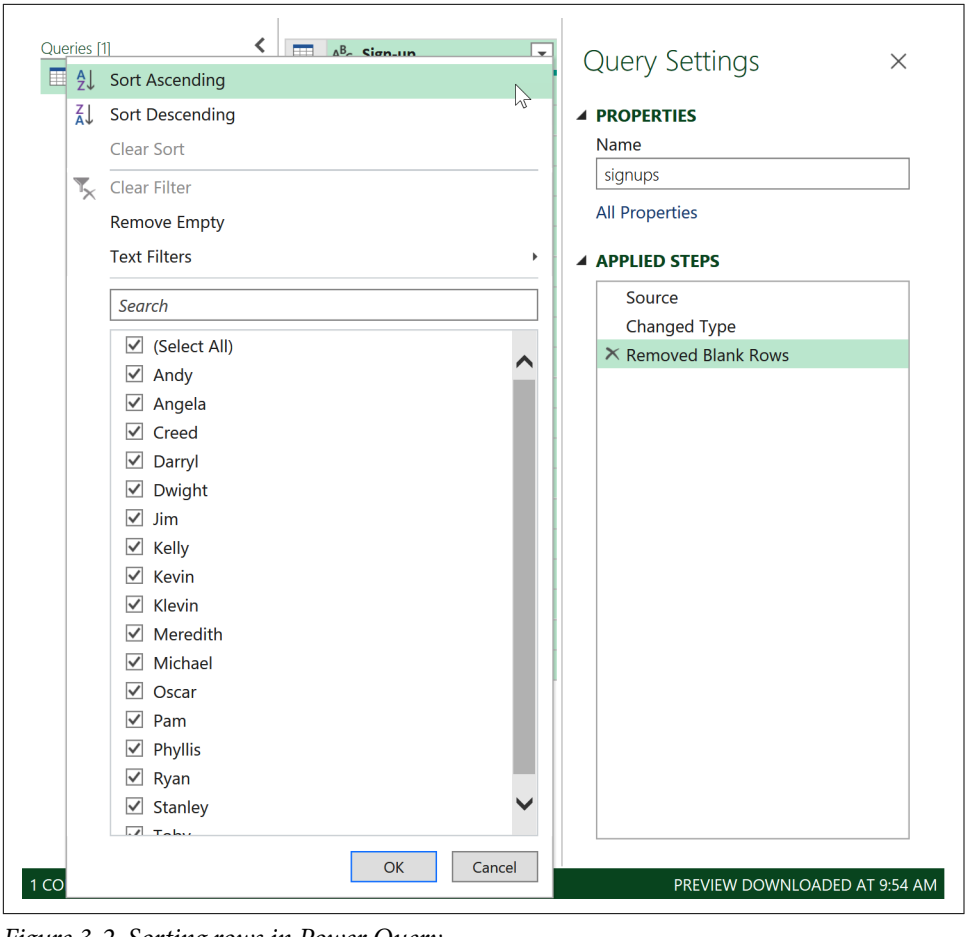


Figure 3-2. Sorting rows in Power Query

The list is mostly clean, except for a typo: "Klevin" in row 9. This error wouldn't be caught by simply removing blanks or duplicates, underscoring the importance of domain knowledge in data management. Power Query aids in standard cleaning tasks, but some scenarios require a deeper understanding of the data. The final step involves correcting this typo by filtering it out of the dataset, as shown in Figure 3-3.

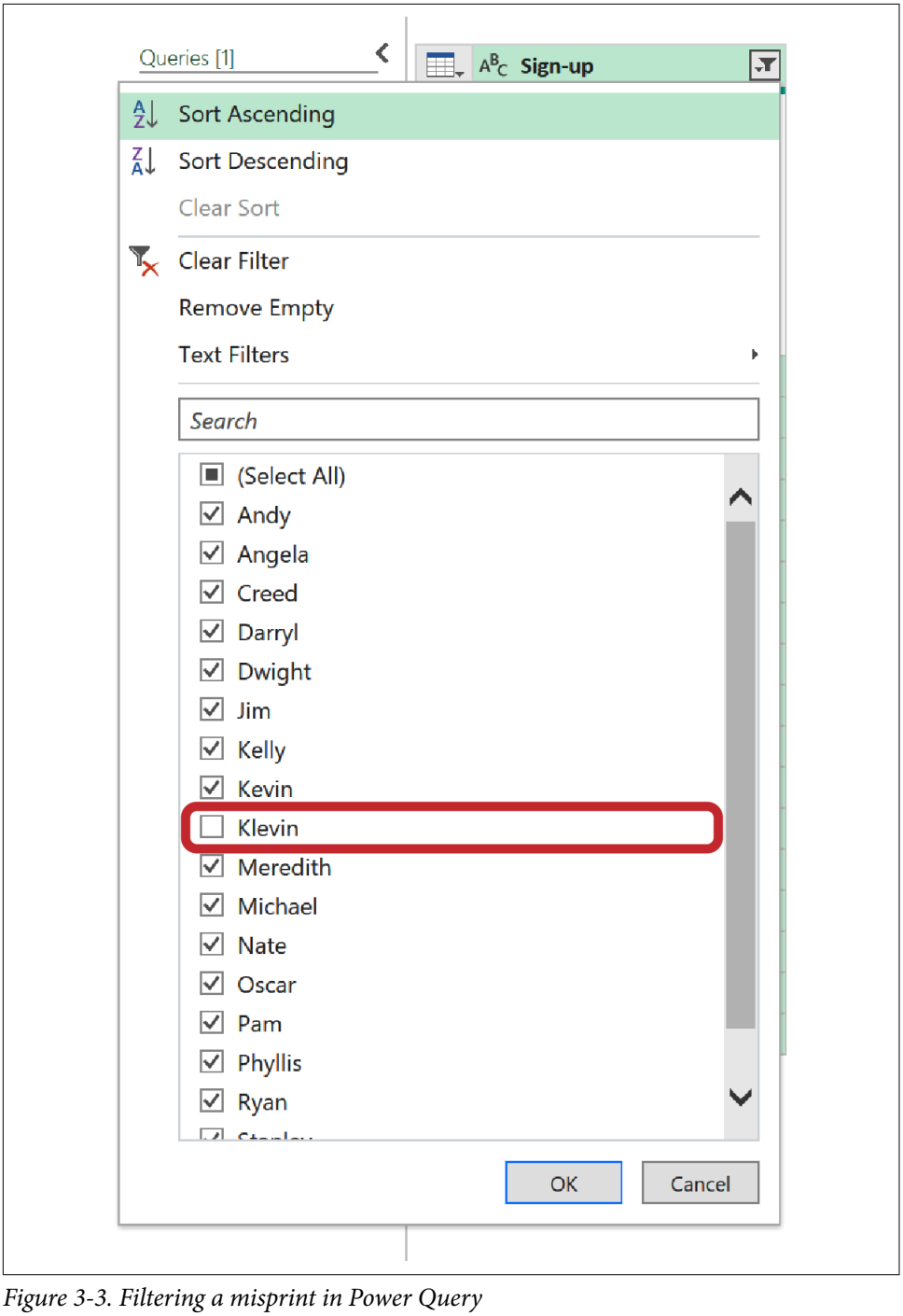


Figure 3-3. Filtering a misprint in Power Query

Refreshing the Query

Great job following these steps to clean the party planning data. To make the results more accessible, load the cleaned dataset into Excel: from the Home tab, click on Close & Load.

Organizing Extracts and Loads in the Workbook

Storing input data in the same workbook as the transformed version has a drawback: it becomes challenging to differentiate between the two. Although Excel attempts to assist by assigning a default blue color to the original table and green to the transformed one, confusion can still arise due to their similar worksheet names. To avoid this confusion, consider appending a suffix like "-out" to the Power Query output worksheet or adhering to another naming convention.

Power Query offers more than just an escape from the tedious and error-prone point-and-click data cleaning processes in Excel; its true advantage lies in its ability to refresh your work with a single click. To see this in action, add two rows to your original `signups` table. For example, I'll insert one blank row and a signup from Nate.

To rerun your query, navigate to the Power Query output table, right-click it, and select Refresh, as shown in Figure 3-4.

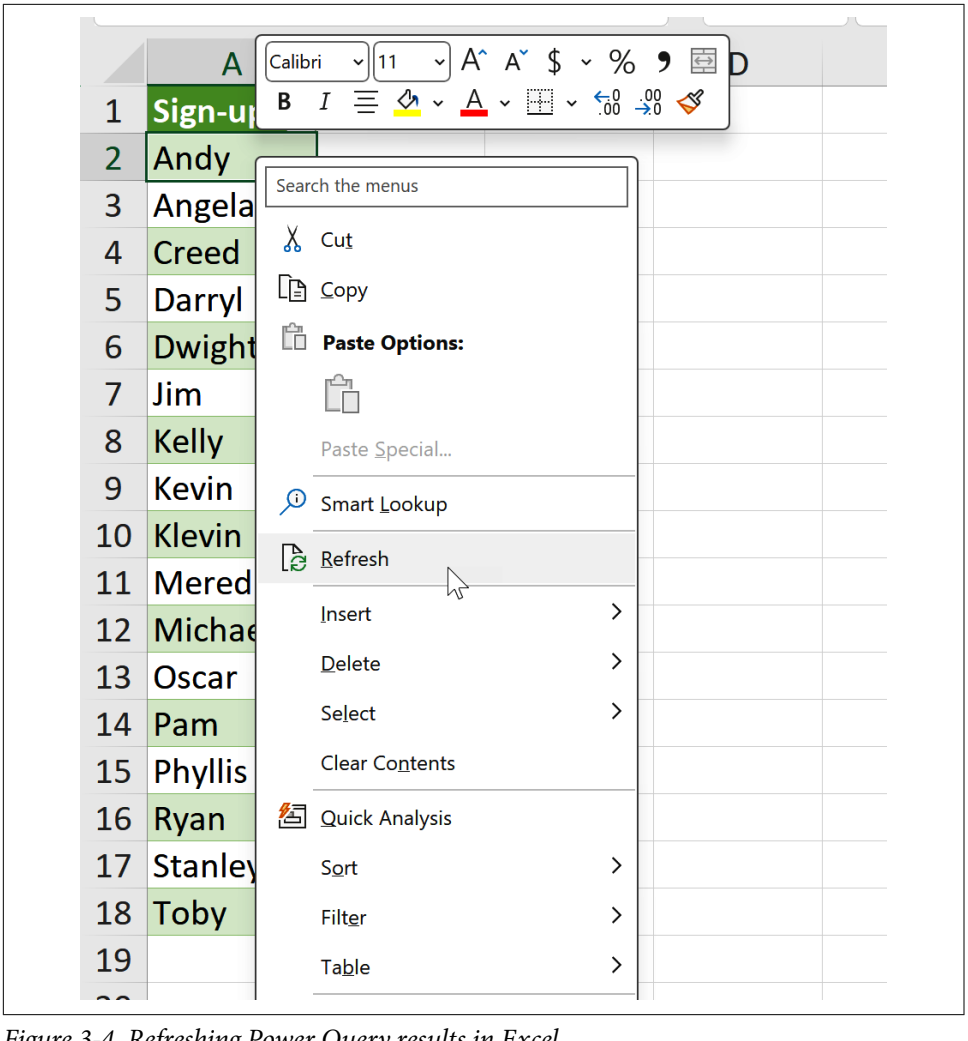


Figure 3-4. Refreshing Power Query results in Excel

The table will be automatically updated, applying all steps to the refreshed data. This workbook now features a one-click, reproducible data cleaning process that can be used for any future signup lists.

Splitting Data into Rows

Have you ever encountered a situation where you have a list of items separated by commas in Excel, and you wished to break them into individual cells? Take a look at the example displayed in Figure 3-5. You can find this data in the `roster` worksheet of *ch_03.xlsx*.

	A	B
1	Department	Signups
2	Sales	Jim, Dwight, Phyllis, Andy
3	Accounting	Kevin, Oscar
4	Warehouse	Daryl, Nate
5	Annex	Toby, Ryan, Kelly
6		

Figure 3-5. Cleaning a roster of names

This dataset contains signups for a project roster categorized by department and name. Our objective is to conveniently sort and filter this data based on name and department. In classic Excel, one approach you might try would be to use the Text to Columns feature, resulting in a messy, unsatisfactory outcome like Figure 3-6.

	A	B	C	D	E
1	Department	Signups	Column1	Column2	Column3
2	Sales	Jim	Dwight	Phyllis	Andy
3	Accounting	Kevin	Oscar		
4	Warehouse	Daryl	Nate		
5	Annex	Toby	Ryan	Kelly	
6					

Figure 3-6. RSVPs split into columns using Text to Columns

Power Query provides a convenient solution for splitting the data to achieve the desired outcome.

To start, import the `roster` data into Power Query and name the query `roster`. Select the `Signups` column in the dataset. In the Power Query Editor, go to the Home tab and click on the Split Column option. From the drop-down menu, select By Delimiter, as seen in Figure 3-7, to proceed.

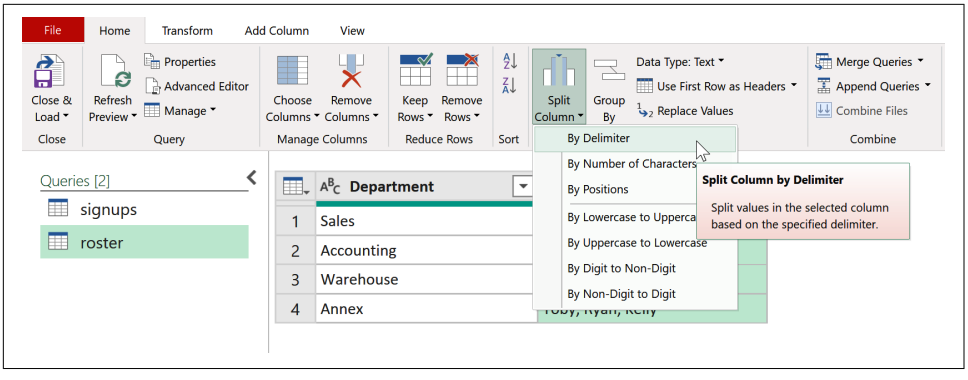


Figure 3-7. Splitting by delimiter in Power Query

The term *delimiter* refers to the character separating each item in the data. In this case, the delimiter is a comma, and Power Query will likely detect it automatically. If it doesn't, choose Comma from the drop-down list. Next, click on Advanced Options. Here you will find the hidden option to split text into rows rather than the standard columns, as seen in Figure 3-8. Click on Rows and then select OK.

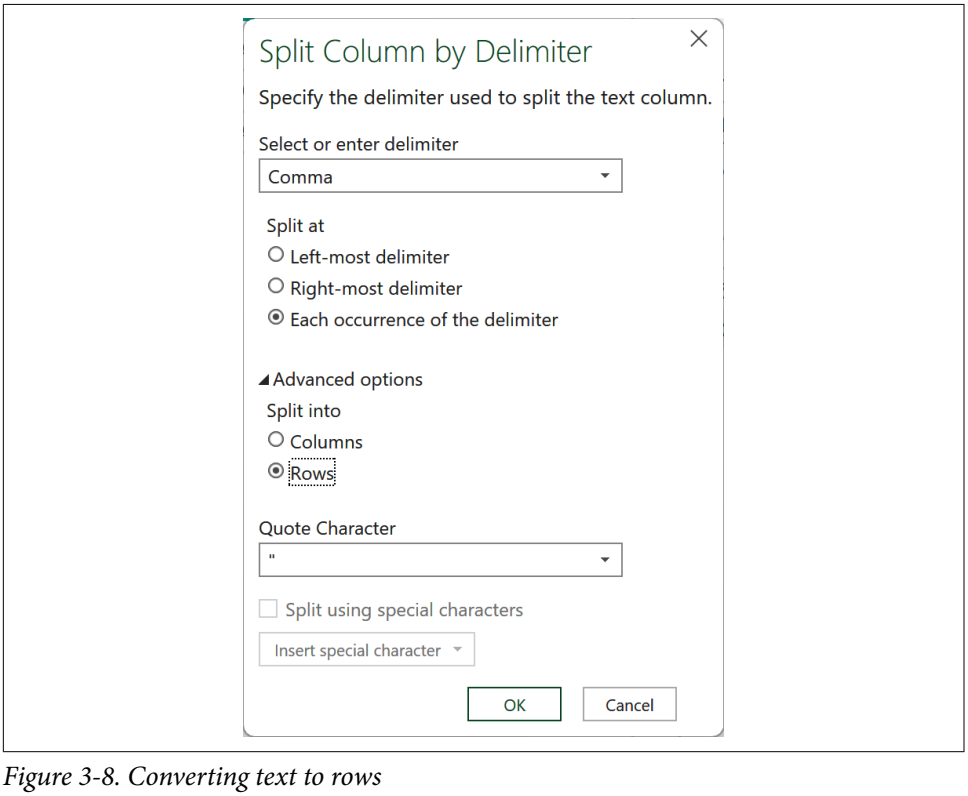


Figure 3-8. Converting text to rows

This query is nearly ready to load, but there is one final task to complete. To find out what, navigate to the View tab in the Power Query Editor ribbon. In the Data Preview group, ensure that "Show whitespace" is checked. For a refresher on Power Query's data previewing and profiling features, refer back to Chapter 2.

Extra spaces, left over from when the roster was delimited by commas, are now seen in the `Signups` column. To remove them, right-click on the column header, then select Transform → Trim, as seen in Figure 3-9.

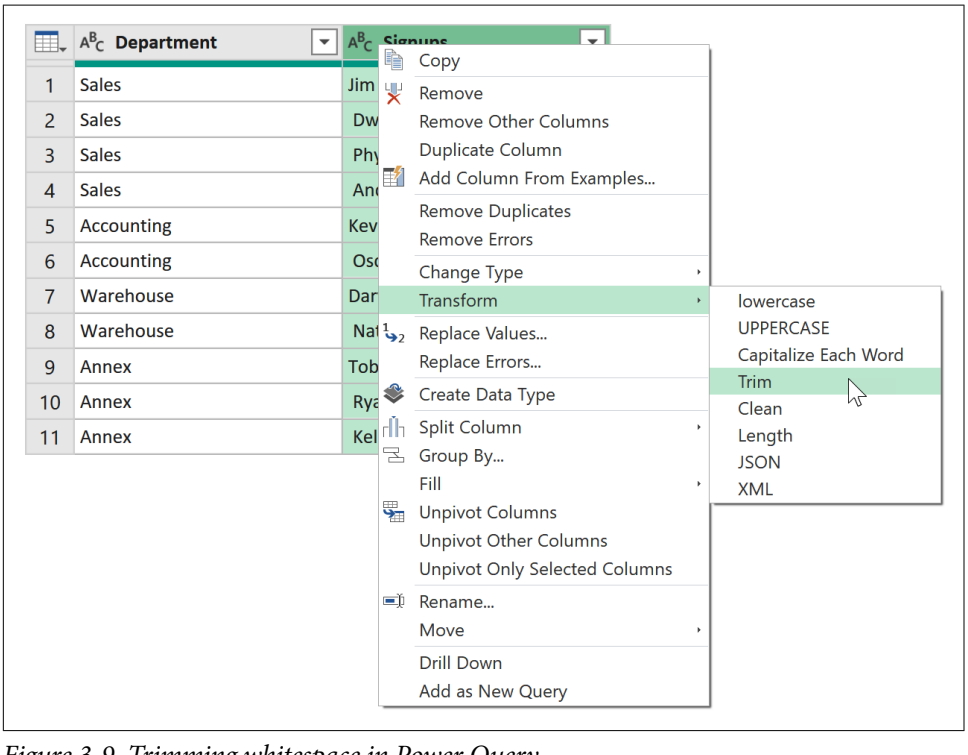


Figure 3-9. Trimming whitespace in Power Query

You can now Close & Load the results to a table, as shown in Figure 3-10:

	A	B
1	Department	Signups
2	Sales	Jim
3	Sales	Dwight
4	Sales	Phyllis
5	Sales	Andy
6	Accounting	Kevin
7	Accounting	Oscar
8	Warehouse	Daryl
9	Warehouse	Nate
10	Annex	Toby
11	Annex	Ryan
12	Annex	Kelly
13		

Figure 3-10. RSVP data split into rows

Filling in Headers and Cell Values

You may encounter instances where parts of a dataset are erroneously marked as `null` or otherwise missing by mistake. This can be attributed to formatting issues in an external system or poor data storage practices.

This section demonstrates how Power Query can assist in fixing both missing headers and missing values. For this next demonstration, refer to the `sales` worksheet in *ch_03.xlsx*.

Replacing Column Headers

Enterprise resource planning (ERP) extracts often contain an additional row filled with irrelevant information. In this particular dataset, the first row is marked with `###` in each column, while the actual column headers can be found in row 2. Instead of manually addressing the issue on a weekly basis by deleting the unnecessary row, it is possible to automate the cleanup with Power Query.

After loading the data into Power Query and naming the query `sales`, navigate to the Home tab and within the Transform group, select Use First Row as Headers, as shown in Figure 3-11.

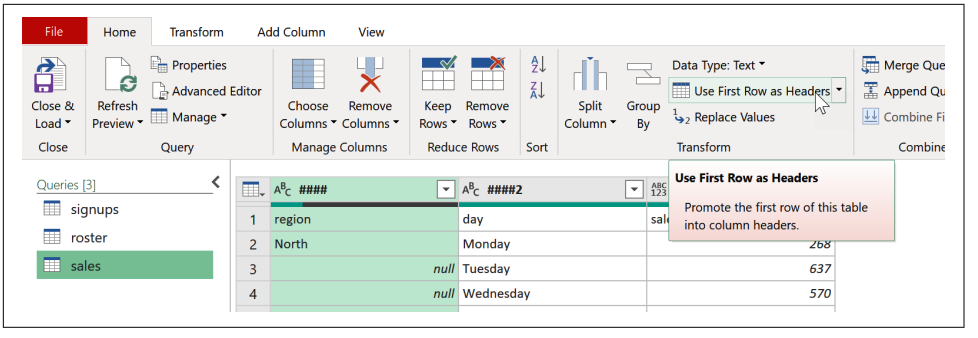


Figure 3-11. Using the first row as headers in Power Query

Filling Down Blank Rows

Now that the column headers have been resolved, it's time to address the issue of mistakenly blank rows. It seems that the ERP system fails to repeat the `region` label across every value for each category, which can pose difficulties when using PivotTables or other features to analyze the data. To fix this, with the `region` column selected, head to the Transform tab and within the Any Column group, choose Fill → Down. This can be observed in Figure 3-12.

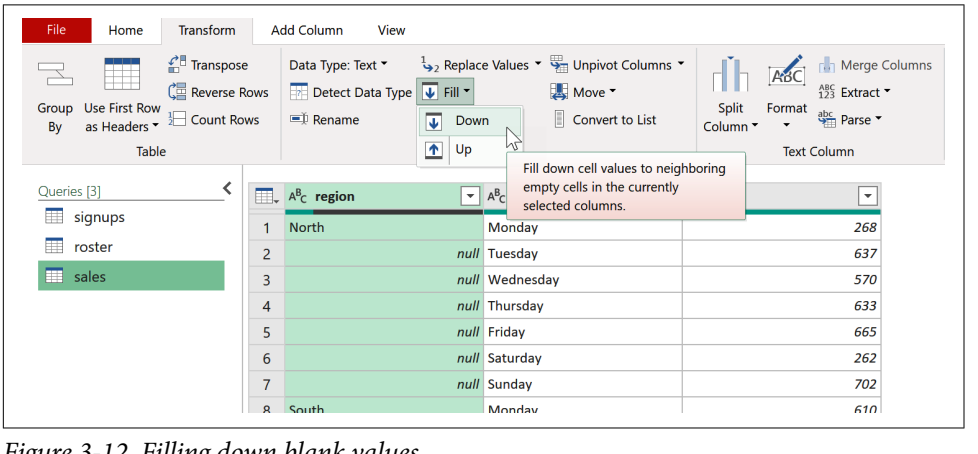


Figure 3-12. Filling down blank values

The data has been successfully cleaned. You can now close and load the results.

Conclusion

Power Query excels as a tool for cleaning data rows, offering simplified and efficient processes for tasks like sorting, filtering, eliminating duplicates, and managing missing values. Chapter 4 will extend this focus, shifting attention to the transformation of columns.

Exercises

Open *ch_03_exercises.xlsx* in the *exercises\ch_03_exercises* folder in the book's companion repository. It contains two worksheets. Use Power Query to manipulate and analyze the data.

On the *states* worksheet:

1. Remove the `United States` row from the data.
2. Fill down blanks on the `region` and `division` columns.
3. Sort by `population` from high to low.
4. Load results into a PivotTable.

On the *midwest_cities* worksheet, load this data into a table where each city is in its own row.

You can find a completed version of this file, *ch_03_exercise_solutions.xlsx*, in the same folder.

CHAPTER 4

Transforming Columns in Power Query

Chapter 3 focused on getting familiar with operating on rows; the focus of this chapter shifts to columns. This chapter includes various techniques like transforming string case, reformatting columns, creating calculated fields, and more. To follow this chapter's demonstrations, refer to *ch_04.xlsx* in the *ch_04* folder of the book's repository. Go ahead and load the `rentals` table into Power Query.

Changing Column Case

Power Query streamlines the process of converting text columns between lowercase, uppercase, and "proper" case (in which each word is capitalized). To test this capability, press the Ctrl key and select the `Title` and `Artist Name` columns simultaneously. Next, right-click on one of the columns, navigate to Transform → Capitalize Each Word, as shown in Figure 4-1.

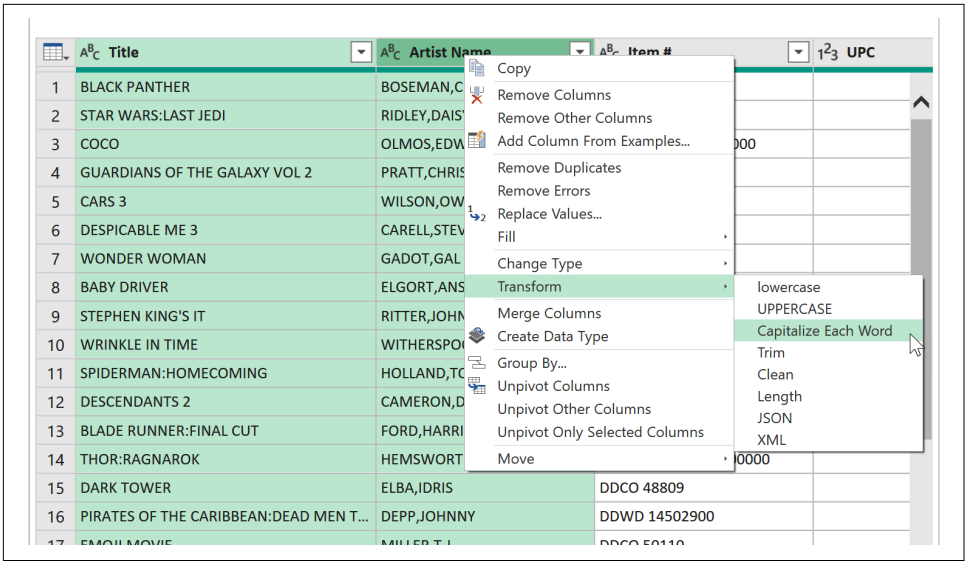


Figure 4-1. Changing text case in Power Query

Notice that `Title` and `Artist Name` lack spaces after colons and commas. To address this, with both columns still selected, right-click on either column and choose Replace Values. In the Replace Values dialog box, search for "`:`" and replace it with a colon followed by a space, as shown in Figure 4-2.

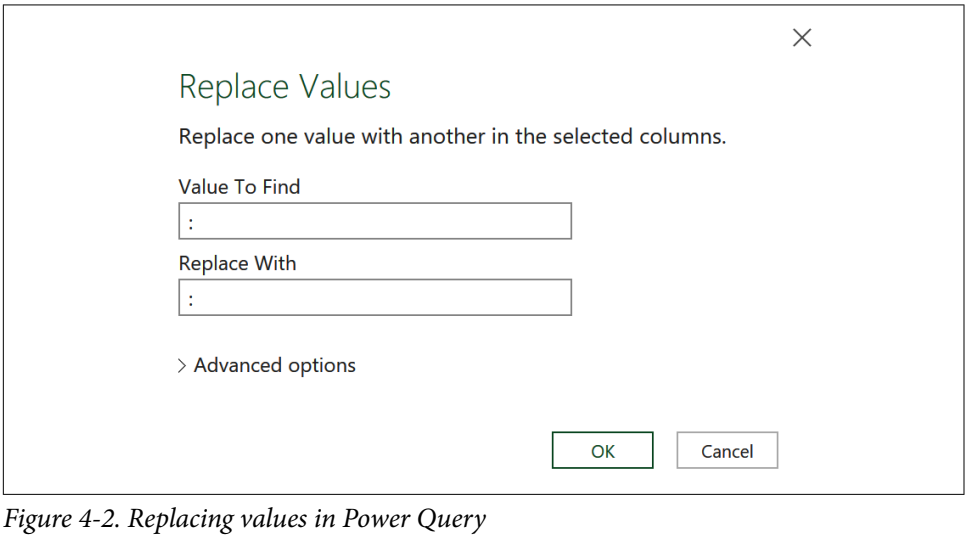


Figure 4-2. Replacing values in Power Query

Next, apply the same process to commas: replace them with a comma followed by a space.

As demonstrated in Chapter 3, Power Query captures every step you perform on the data in the Applied Steps list. This feature greatly facilitates the auditing of text changes compared to the conventional find and replace process.

Delimiting by Column

In Chapter 3, you learned how to split comma-delimit text into rows. Now it's time to do the same with columns. Right-click on the `Item #` column and split it into two by selecting Split Column → By Delimiter. In the dialog box, select Space from the drop-down and click OK. Once again, this process offers improved user-friendliness and a broader range of features compared to the traditional Text to Columns function.

The delimited columns are initially labeled as `Item #.1` and `Item #.2`. To rename them, simply double-click on the column headers in the Editor. As with all modifications in Power Query, these alterations are recorded via Applied Steps, allowing for effortless reversal or adjustment as needed.

Changing Data Types

In Power Query, each column is assigned a specific data type, which defines the operations that can be performed on it. When importing a dataset, Power Query automatically tries to determine the most appropriate data type for each column. However, there are situations where this automatic detection can be enhanced or adjusted.

For example, take the `UPC` column. By default, it is assigned the Whole Number data type. However, since we don't anticipate conducting significant mathematical operations on this column, it's more suitable to store it as text. To do this, click the number icon next to the `UPC` column and change its data type to Text, as seen in Figure 4-3.

Proceed with the following data type changes:

- Convert the `ISBN 13` column to Text.
- Convert the `Retail` column to Currency.

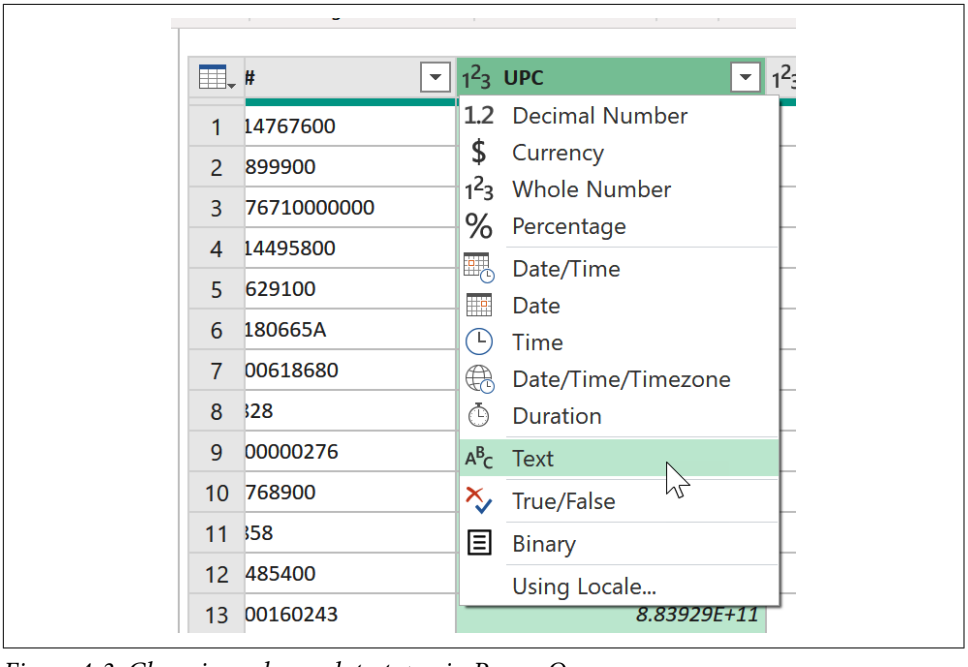


Figure 4-3. Changing column data types in Power Query

Deleting Columns

Removing unnecessary columns from a dataset makes it easier to process and analyze. Select the `BTkey` column and press the Delete key to remove it from your query. If you decide to include this column later, you can easily retrieve it through the Applied Steps list, as explained in Chapter 2.

Working with Dates

Power Query offers an extensive array of sophisticated methods for managing, transforming, and formatting dates. It facilitates the modification of date types, allowing users to extract components such as the month number and day name, and then store these components in the most appropriate data types.

To explore this functionality, let's apply it to the `Release Date` column in a few different ways. Begin by creating copies of this column: right-click on the column and choose "Duplicate Column." Perform this operation two more times to generate a total of three duplicate date columns.

Right-click the first duplicated `Release Date` column and navigate to Transform → Year → Year, as in Figure 4-4. The column will be reformatted and its type changed to display only the year instead of the complete date.

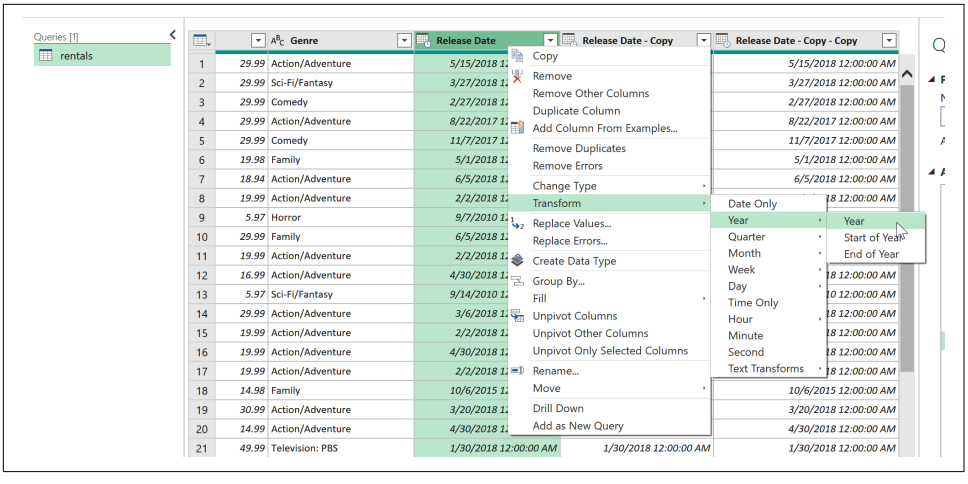


Figure 4-4. Transforming date columns in Power Query

Extract the month and day numbers from the next two columns. Double-click the column headers and rename them as `Year`, `Month`, and `Day`, respectively, to reflect the reformatted data. Close and load your results to an Excel table.

Great job on successfully executing a series of column-oriented data manipulations in Power Query. You're all set to load this query into Excel.

Creating Custom Columns

Adding a *calculated column* is a common task in data cleaning. Whether it's a profit margin, date duration, or something else, Power Query handles this process through its M programming language.

For this next demonstration, head to the `teams` worksheet of *ch_04.xlsx*. This dataset includes season records for every Major League Baseball team since 2000. Our objective is to create a new column that calculates the win record for each team during the season. This calculation is done by dividing the number of wins by the total number of wins and losses combined.

The first step, of course, is to load the data into Power Query. Then from the ribbon of the Editor, head to Add Column → Custom Column. Name your custom column `Wpct` and define it using the following formula:

```
[W] / ([W] + [L])
```

The M programming language of Power Query follows a syntax resembling Excel tables, where column references are enclosed in single square brackets. Take advantage of Microsoft's IntelliSense to hit tab and automatically complete the code as you type these references. Additionally, you have the option to double-click on the desired

columns from the "Available columns" list, which will insert them into the formula area.

If everything is correct, a green checkmark will appear at the bottom of the dialog box, indicating that no syntax errors have been detected, as shown in Figure 4-5.

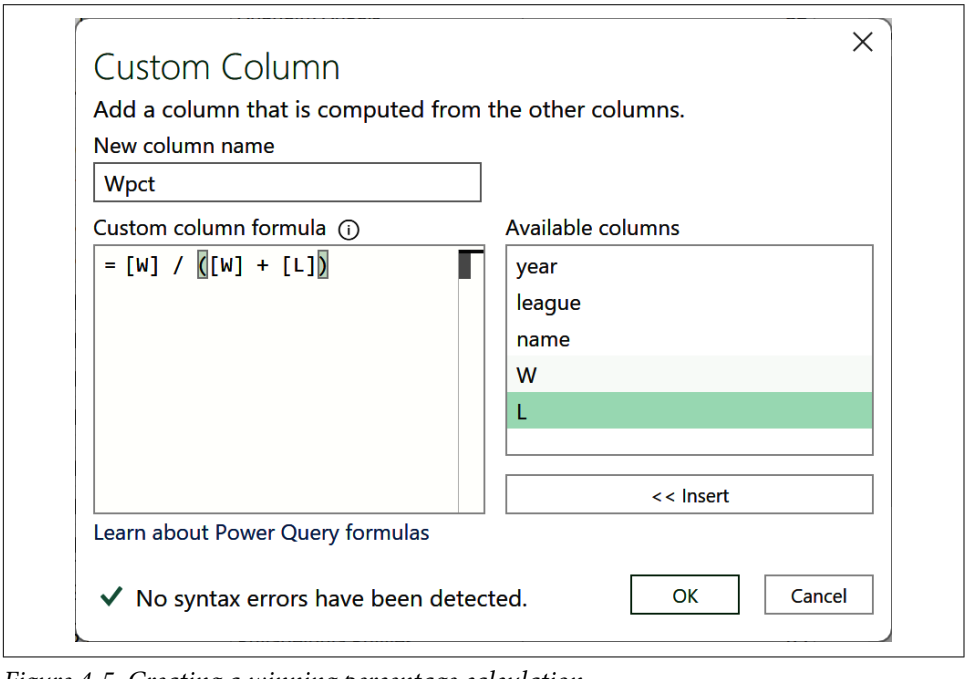
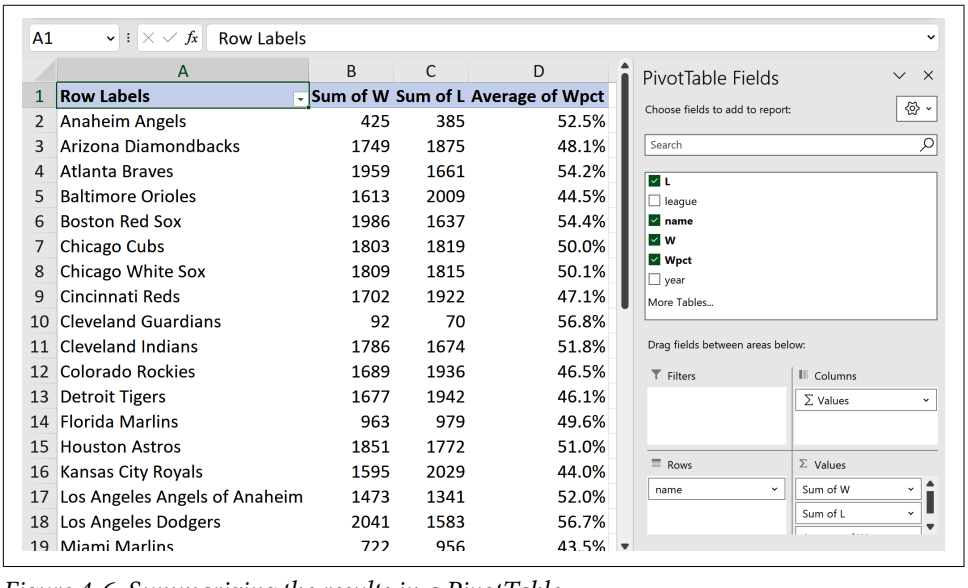


Figure 4-5. Creating a winning percentage calculation

Once you have created this column, go ahead and change its data type in Power Query to Percentage.

Loading & Inspecting the Data

Our new column is calculated and ready to work with. On the Power Query Editor ribbon, head to Home → Close & Load → Close & Load To, then select PivotTable Report and OK. From there, you can analyze the data, such as calculating the average `Wpct` for each team name, as shown in Figure 4-6.



Row Labels	Sum of W	Sum of L	Average of Wpct
Anaheim Angels	425	385	52.5%
Arizona Diamondbacks	1749	1875	48.1%
Atlanta Braves	1959	1661	54.2%
Baltimore Orioles	1613	2009	44.5%
Boston Red Sox	1986	1637	54.4%
Chicago Cubs	1803	1819	50.0%
Chicago White Sox	1809	1815	50.1%
Cincinnati Reds	1702	1922	47.1%
Cleveland Guardians	92	70	56.8%
Cleveland Indians	1786	1674	51.8%
Colorado Rockies	1689	1936	46.5%
Detroit Tigers	1677	1942	46.1%
Florida Marlins	963	979	49.6%
Houston Astros	1851	1772	51.0%
Kansas City Royals	1595	2029	44.0%
Los Angeles Angels of Anaheim	1473	1341	52.0%
Los Angeles Dodgers	2041	1583	56.7%
Miami Marlins	722	956	43.5%

Figure 4-6. Summarizing the results in a PivotTable

Power Query Data Types Versus Excel Cell Formatting

Keep in mind that altering the data type of a column in Power Query only modifies how it's stored in Power Query itself; it doesn't change the data's front-end formatting in Excel. For instance, if you format a column as a percentage in Power Query such as `Wpct` in the previous example, you'll still need to apply percentage formatting in Excel itself. In Part III, we'll explore how to consistently change the data's presentation through data formatting in Power Pivot.

Calculated Columns Versus Measures

It is important to note that the average `Wpct` displayed in the PivotTable is a simple, unweighted average of the season's winning percentages. This means that seasons with a lower number of games—such as the pandemic-affected 2020 season—have a disproportionate impact on the calculation. To verify this, compare the `Average of Wpct` value in the PivotTable with our own Excel calculation, as shown in Figure 4-7.

	A	B	C	D	E	F	G
1	**Row Labels**	**Sum of W**	**Sum of L**	**Average of Wpct**	Average Wpct (Measure)		
2	Anaheim Angels	425	385	52.47%	52.47%	=B2 / (B2 + C2)	
3	Arizona Diamondbacks	1749	1875	48.08%	48.26%		
4	Atlanta Braves	1959	1661	54.23%	54.12%		
5	Baltimore Orioles	1613	2009	44.45%	44.53%		
6	Boston Red Sox	1986	1637	54.41%	54.82%		
7	Chicago Cubs	1803	1819	49.97%	49.78%		
8	Chicago White Sox	1809	1815	50.14%	49.92%		
9	Cincinnati Reds	1702	1922	47.09%	46.96%		
10	Cleveland Guardians	92	70	56.79%	56.79%		
11	Cleveland Indians	1786	1674	51.81%	51.62%		
12	Colorado Rockies	1689	1936	46.50%	46.59%		
13	Detroit Tigers	1677	1942	46.14%	46.34%		
14	Florida Marlins	963	979	49.59%	49.59%		
15	Houston Astros	1851	1772	51.02%	51.09%		
16	Kansas City Royals	1595	2029	43.99%	44.01%		
17	Los Angeles Angels of Anaheim	1473	1341	52.03%	52.35%		
18	Los Angeles Dodgers	2041	1583	56.74%	56.32%		

Figure 4-7. Apparent PivotTable miscalculation

To address this issue, one approach involves using dynamic measures for real-time aggregation and calculations tailored to the analysis context. This is achieved with tools such as Power Pivot's Data Model and the DAX language, discussed in Part II of this book.

This doesn't mean that calculated columns in Power Query should be avoided altogether. They are simple to create and computationally efficient. Nevertheless, if there is a possibility that these columns might lead to misleading aggregations, it is advisable to opt for a DAX measure instead.

Reshaping Data

In Chapter 1, you were introduced to the concept of "tidy" data, where every variable is stored in one and only one column. You may recall the data in the `sales` worksheet as an example of untidy data. Fortunately, Power Query resolves this critical data storage problem. To begin, navigate to the familiar `sales` worksheet of the *ch_04.xlsx* workbook. Load this table into Power Query to initiate the data transformation process.

The goal is to "unpivot" or "melt" the all sales columns into one column named `sales`, along with the labels for those sales in one column called `department`. To do so, hold down the Ctrl key and select the first three variables: `customer_id`, `channel`, and `region`. Right-click and select Unpivot Other Columns, as shown in Figure 4-8.

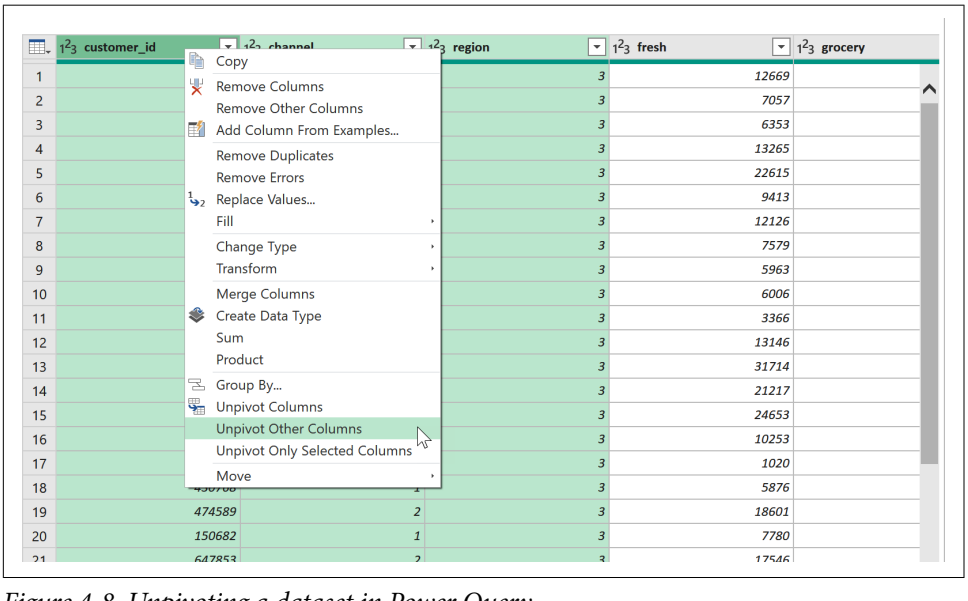
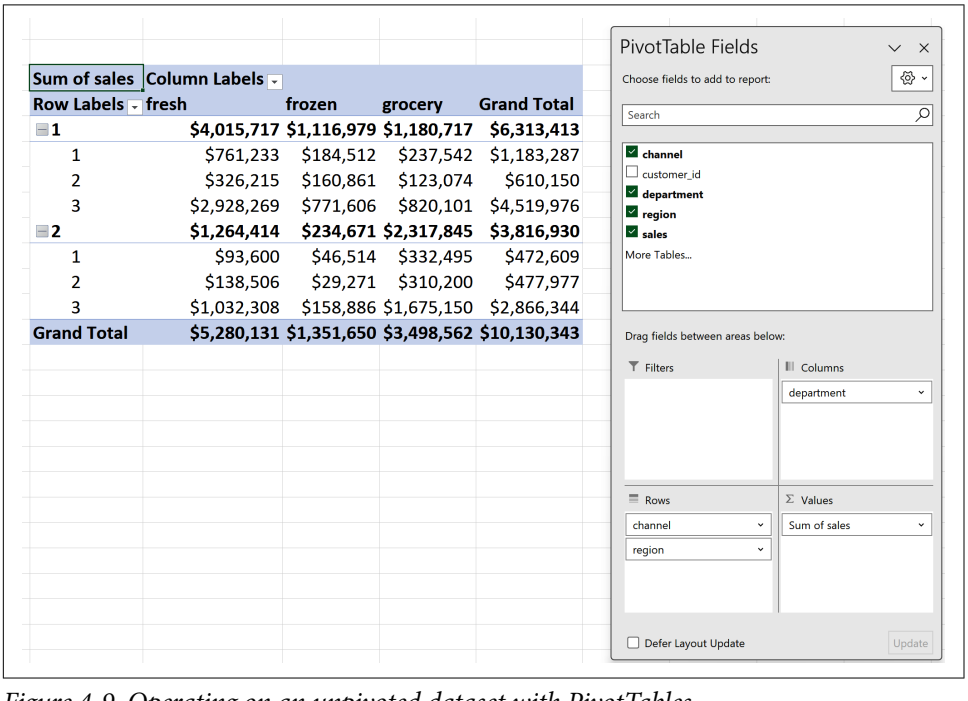


Figure 4-8. Unpivoting a dataset in Power Query

By default, the two resulting unpivoted columns will be called `Attribute` and `Value`. Rename them `department` and `sales`, respectively. You can now load the query to a PivotTable and analyze sales by channel and region. The results and benefits of creating a PivotTable based on this reshaped data are seen in Figure 4-9.



Sum of sales	Column Labels			
Row Labels	fresh	frozen	grocery	Grand Total
1	$4,015,717	$1,116,979	$1,180,717	$6,313,413
1	$761,233	$184,512	$237,542	$1,183,287
2	$326,215	$160,861	$123,074	$610,150
3	$2,928,269	$771,606	$820,101	$4,519,976
2	$1,264,414	$234,671	$2,317,845	$3,816,930
1	$93,600	$46,514	$332,495	$472,609
2	$138,506	$29,271	$310,200	$477,977
3	$1,032,308	$158,886	$1,675,150	$2,866,344
Grand Total	$5,280,131	$1,351,650	$3,498,562	$10,130,343

Figure 4-9. Operating on an unpivoted dataset with PivotTables

Conclusion

This chapter explored different ways to manipulate columns in Power Query. Chapter 5 takes a step further by working with multiple datasets in a single query. You'll learn how to merge and append data sources, as well as how to connect to external sources like *.csv* files.

Exercises

Practice transforming columns in Power Query with the *ch_04_exercises.xlsx* file found in the *exercises\ch_04_exercises* folder in the book's companion repository. Perform the following transformations to this work orders dataset:

1. Transform the `date` column to a month format, such as changing *1/1/2023* to *January*.
2. Transform the `owner` column to proper case.
3. Split the `location` column into two separate columns: `zip` and `state`.
4. Reshape the dataset so that `subscription_cost`, `support_cost`, and `services_cost` are consolidated into two columns: `category` and `cost`.
5. Introduce a new column named `tax` that calculates 7% of the values in the `cost` column.
6. Convert the `zip` variable to the Text data type, and update both `cost` and `tax` columns to Currency.
7. Load the results to a table.

For the solutions to these transformations, refer to *ch_04_solutions.xlsx* in the same folder.

CHAPTER 5

Merging and Appending Data in Power Query

So far in Part I, you have learned various operations to transform the rows and columns of a single table using Power Query. However, data often comes from multiple tables, including sources outside of Excel. In this chapter, you will discover how to merge multiple files into a single dataset.

Because this chapter focuses on connecting to external files rather than tables inside the same workbook, start following along by opening a new workbook.

Appending Multiple Sources

Data often arrives in formats that require vertically stacking files together. For example, Figure 5-1 illustrates a common scenario where sales data is presented in separate tables for January, February, and March. In these instances, it's helpful to merge them into a single source. This enables the computation of total sales for quarter 1, for instance.

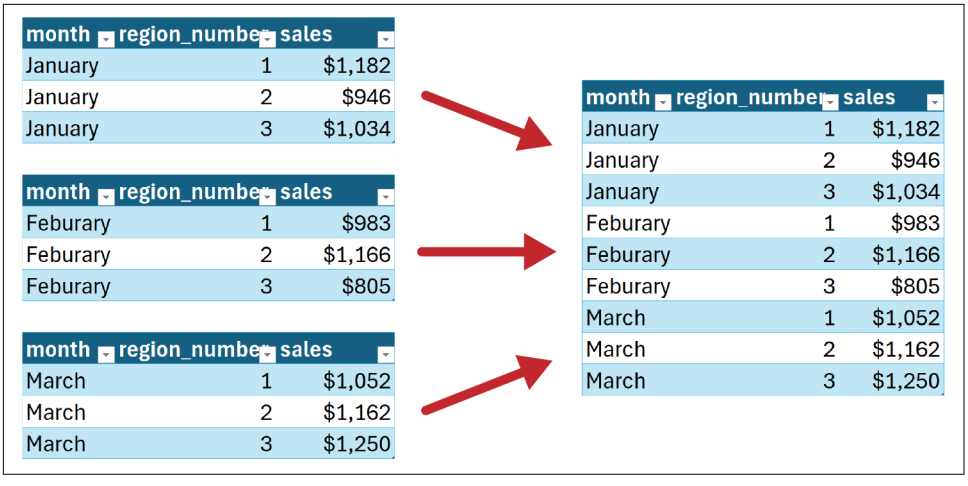


Figure 5-1. A simple example of datasets to append

The append operation in Power Query facilitates this process.

Connecting to External Excel Workbooks

So far, this book has used Power Query for the manipulation of data sources within a workbook. However, the utility of Power Query extends significantly beyond this scope. It facilitates integration with a multitude of data sources, notably external Excel files and *.csv* files, which will be a primary focus of this chapter. The *ch_05* folder in the book's companion repository features datasets sourced from sports journalist Sean Lahman's Major League Baseball database, through the conclusion of the 2022 season.

The files *people_born_in_usa.xlsx* and *people_born_outside_usa.xlsx* consist of information about Major League Baseball individuals who were born inside and outside of the United States, respectively. The goal is to vertically append these two files into a single table using Power Query.

To begin the process, navigate to the Data tab on the ribbon and select Get Data → From File → From Excel Workbook, as shown in Figure 5-2.

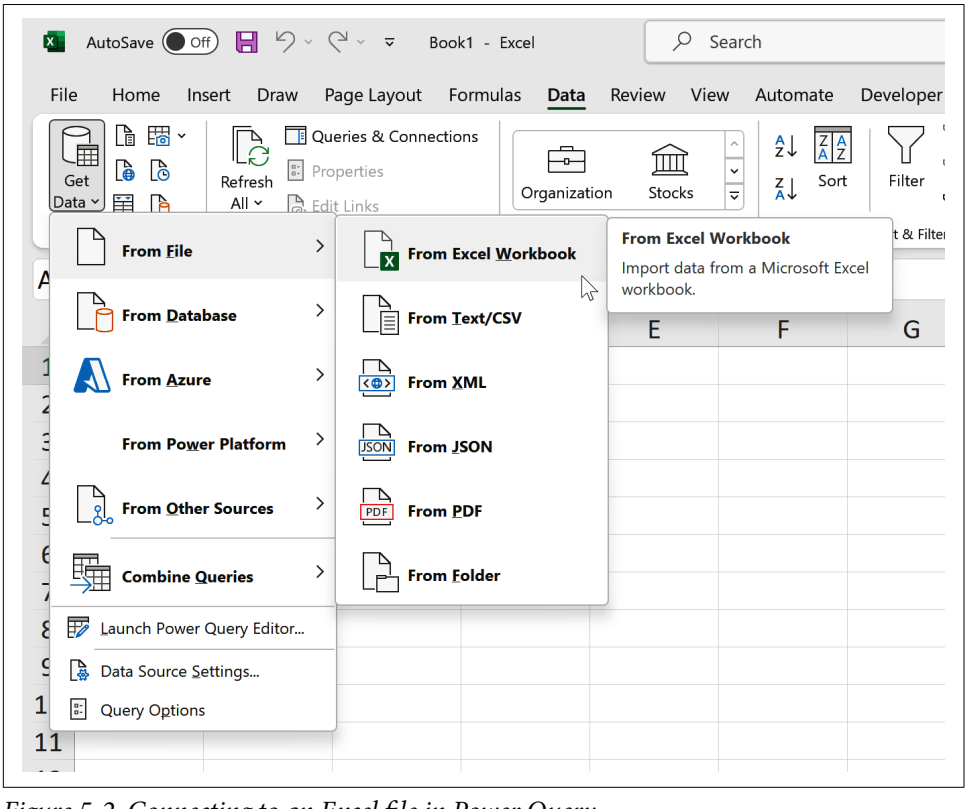


Figure 5-2. Connecting to an Excel file in Power Query

Start by connecting to *people_born_in_usa.xlsx*. Keep in mind that an Excel workbook can contain multiple worksheets, named ranges, tables, and more. That means you need to select exactly what entity of the workbook you want to load into Power Query. In this case, we want the `people_born_in_usa` table, so click on that option underneath the search bar in the Navigator dialog box, as shown in Figure 5-3.

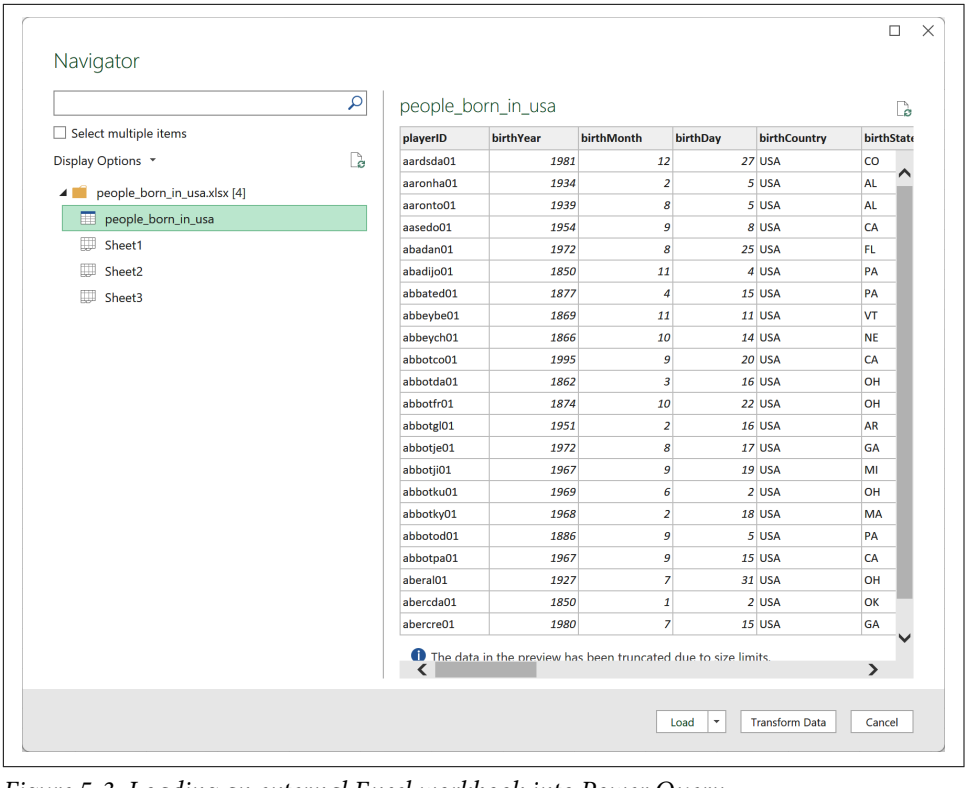


Figure 5-3. Loading an external Excel workbook into Power Query

Before loading data into your workbook, you can clean or transform it by accessing the Power Query Editor. This is done by selecting the Transform Data button. However, I will proceed to load the data directly into the workbook for now. If there's a need to transform the data later, I can always return to Power Query to make the necessary adjustments.

To proceed, click the drop-down button next to Load, and then select Load To in the Navigator dialog box and choose the Only Create Connection option. Since the goal is to append this file to another one for combined analysis later, there's no need to load the data into a separate Excel table at this stage.

Next, repeat these steps to load *people_born_outside_usa.xlsx*. Again, load the query as Only Create Connection. You have now loaded both files as connections only to Power Query.

On the ribbon, while the Data tab is still selected, go to Queries & Connections. Here, you'll find both `people_born_in_usa` and `people_born_outside_usa` listed as connections. Right-click on `people_born_in_usa` and choose Edit to open the Power Query Editor, as seen in Figure 5-4.

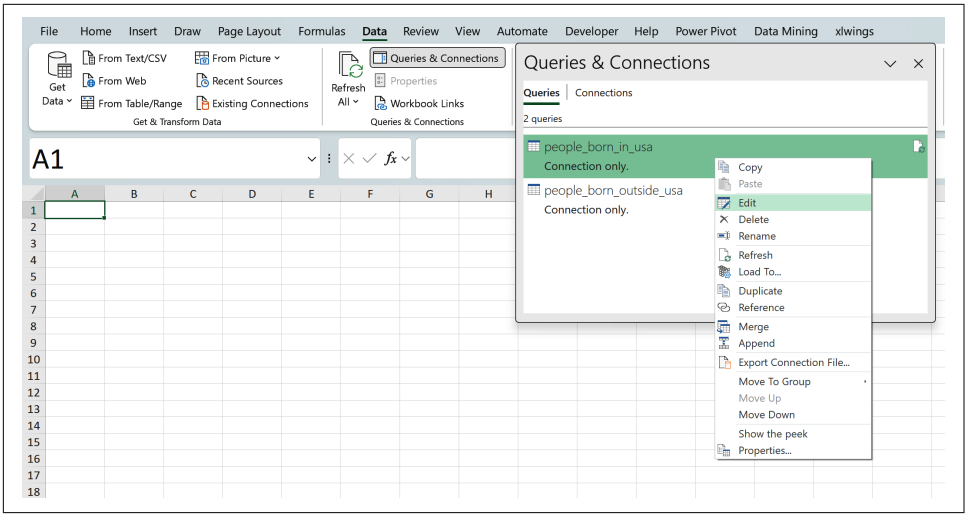


Figure 5-4. Viewing queries in the Queries & Connections Pane

Appending the Queries

Next, head to Home on the Power Query Editor ribbon and click the Append Queries dropdown in the Combine group. Select "Append Queries as New" to proceed, as seen in Figure 5-5.

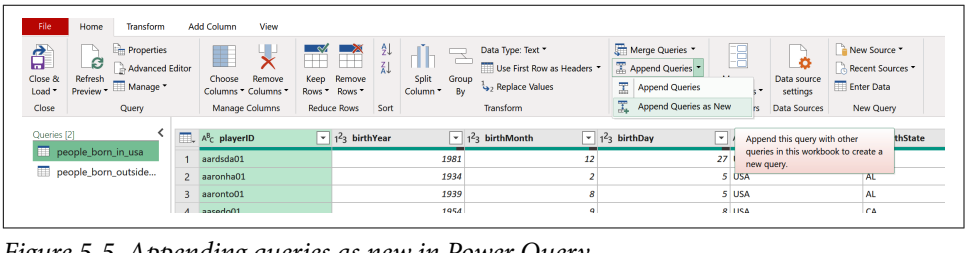


Figure 5-5. Appending queries as new in Power Query

Append Queries combines data from multiple tables into an existing query, enlarging it, whereas "Append Queries as New" combines them into a new query, keeping original tables unchanged.

The queries you are seeking to append should have a consistent data structure, including the same number of columns, column names, and data types. Otherwise, you may need to perform some data transformation steps to align the structure before appending.

You will now see an Append dialog box asking which tables to combine into a single table. Choose `people_born_in_usa` and `people_born_outside_usa`, as seen in Figure 5-6.

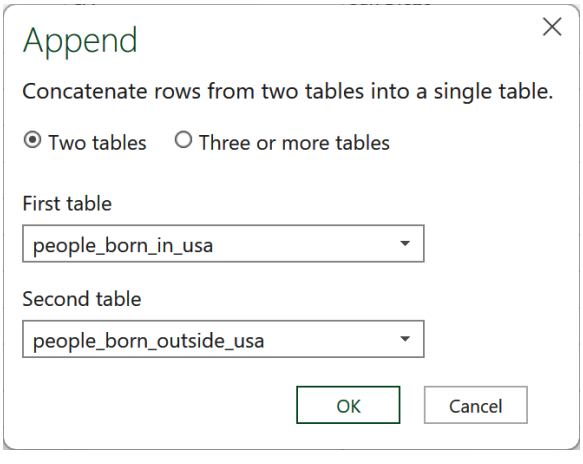


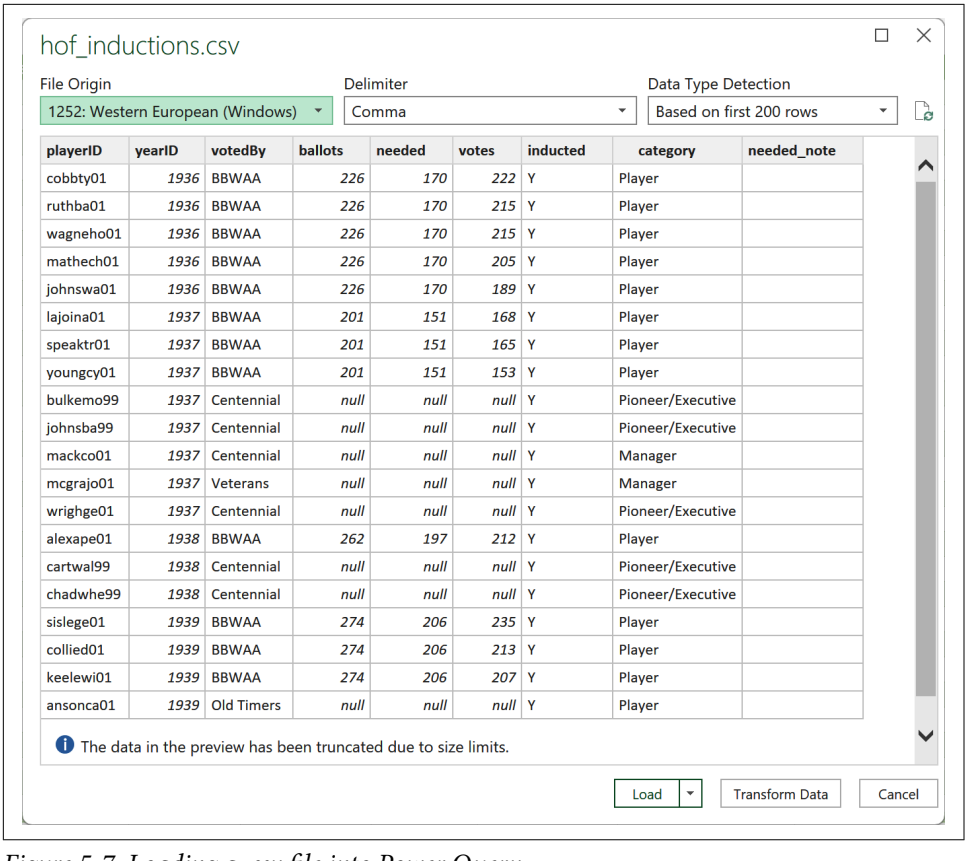
Figure 5-6. Appending two tables in Power Query

Well done! You appended the two tables to create a new query named `Append1`. Rename it to `people_append` for clarity. Close and load your results to an Excel table. The resulting query will have 20,370 rows, representing the combined total of rows from both tables. You can confirm this count using Power Query's data profiling features as covered in Chapter 2.

Understanding Relational Joins

After appending all individual-level records into one table, the next step involves linking it with other tables to gain further insights. The original Lahman database comprises various tables at the individual record, including batting records, All-Star game appearances, and others. By utilizing the `playerID` column, these tables can be interconnected efficiently.

To try this out in your current workbook, use Power Query to connect to the dataset named *hof_inductions.csv* located in the same folder. This dataset comprises information on inductees into the Baseball Hall of Fame. To proceed, navigate in Excel to Data → Get Data → From File → From Text/CSV. Then, locate and select *hof_inductions.csv*. Because a *.csv* file does not support multiple sheets or ranges like Excel does, it will display the data immediately, as shown in Figure 5-7.



playerID	yearID	votedBy	ballots	needed	votes	inducted	category	needed_note
cobbty01	1936	BBWAA	226	170	222	Y	Player	
ruthba01	1936	BBWAA	226	170	215	Y	Player	
wagneho01	1936	BBWAA	226	170	215	Y	Player	
mathech01	1936	BBWAA	226	170	205	Y	Player	
johnswa01	1936	BBWAA	226	170	189	Y	Player	
lajoina01	1937	BBWAA	201	151	168	Y	Player	
speaktr01	1937	BBWAA	201	151	165	Y	Player	
youngcy01	1937	BBWAA	201	151	153	Y	Player	
bulkemo99	1937	Centennial	null	null	null	Y	Pioneer/Executive	
johnsba99	1937	Centennial	null	null	null	Y	Pioneer/Executive	
mackco01	1937	Centennial	null	null	null	Y	Manager	
mcgrajo01	1937	Veterans	null	null	null	Y	Manager	
wrighge01	1937	Centennial	null	null	null	Y	Pioneer/Executive	
alexape01	1938	BBWAA	262	197	212	Y	Player	
cartwal99	1938	Centennial	null	null	null	Y	Pioneer/Executive	
chadwhe99	1938	Centennial	null	null	null	Y	Pioneer/Executive	
sislege01	1939	BBWAA	274	206	235	Y	Player	
collied01	1939	BBWAA	274	206	213	Y	Player	
keelewi01	1939	BBWAA	274	206	207	Y	Player	
ansonca01	1939	Old Timers	null	null	null	Y	Player	

Figure 5-7. Loading a .csv file into Power Query

Load `hof_inductions` as a connection only into your workbook.

With all relevant data loaded into Power Query, it's time to find a way to consolidate the information found in the `people_append` table with that found in the `hof_inductions` table by way of the shared `playerID` column.

One option to do this might be to use an Excel lookup function like `VLOOKUP()` to retrieve the corresponding name records for each value of `playerID`. I like to call `VLOOKUP()` the "duct tape of Excel" because of this ability to append additional columns to a dataset.

But if `VLOOKUP()` is duct tape, then the relational join is a full-on welder. I say this because `VLOOKUP()` is primarily designed for single-condition lookups within Excel's environment. Furthermore, it doesn't have a systematic approach to handle missing values, which can lead to data inconsistencies. It can also lead to slow and unreliable workbooks, as each `VLOOKUP()` formula must be recalculated every time the workbook calculates.

The new `XLOOKUP()` function in Excel is designed as a modern upgrade to `VLOOKUP()`, addressing several of its limitations. However, it too does not overcome all the issues when compared to relational joins in Power Query. For more information on the `XLOOKUP()` function, refer to Chapter 10.

Power Query provides a more comprehensive solution. It allows for merging data based on multiple conditions, handles larger datasets more efficiently, offers a systematic way to address missing values, records transformation steps to ensure data integrity, and can pull data from a range of sources.

This method is also more computationally efficient, as the merge is created just once and only re-evaluated when the query gets refreshed. The resulting merge table is a flat, formula-free object that is much easier to use. This makes Power Query a more dynamic tool for intricate data integration and transformation.

The following sections explore two of the most common relational join types: the left outer join and the inner join.

Left Outer Join: Think VLOOKUP()

The *left outer join* retains all records from the first merged table and searches for matching values in the second. If no matches are found, a `null` result is returned. This join type is very similar to `VLOOKUP()`, with one notable difference being the use of `null` to indicate missing values, where `VLOOKUP()` would return `#N/A`.

An example of the result of a left outer join on a small dataset is seen in Figure 5-8.

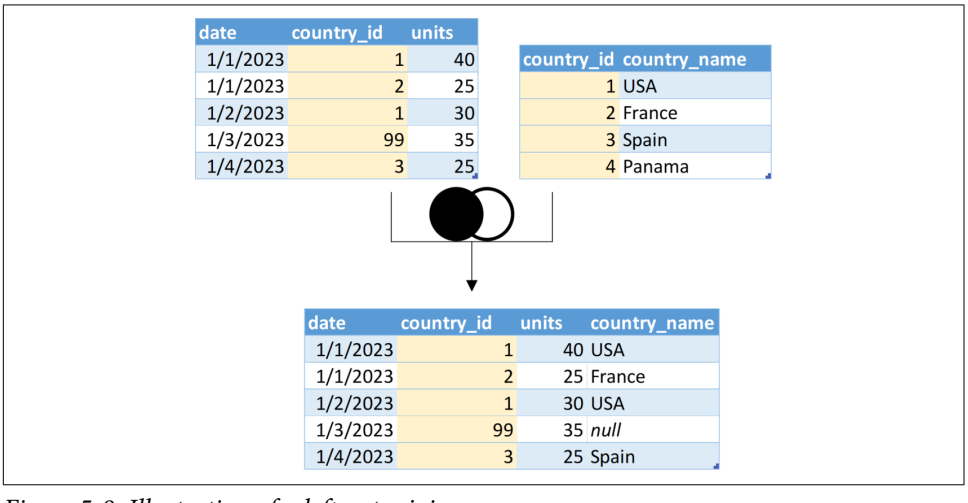


Figure 5-8. Illustration of a left outer join

To get started on the join, head back to the Power Query Editor and select the `people_append` table. Next, head to the Combine group on the ribbon and select Merge Queries as New, as shown in Figure 5-9:

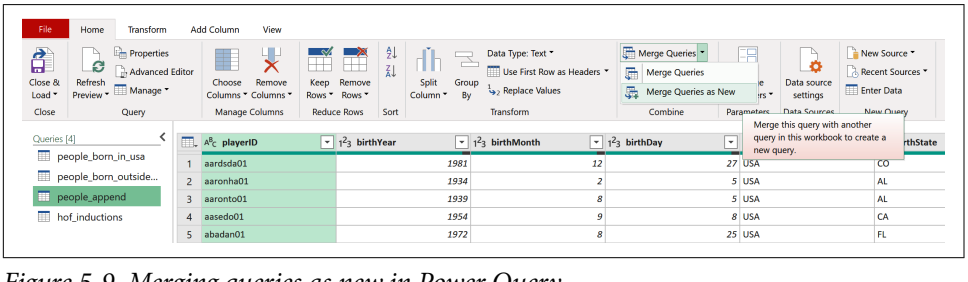
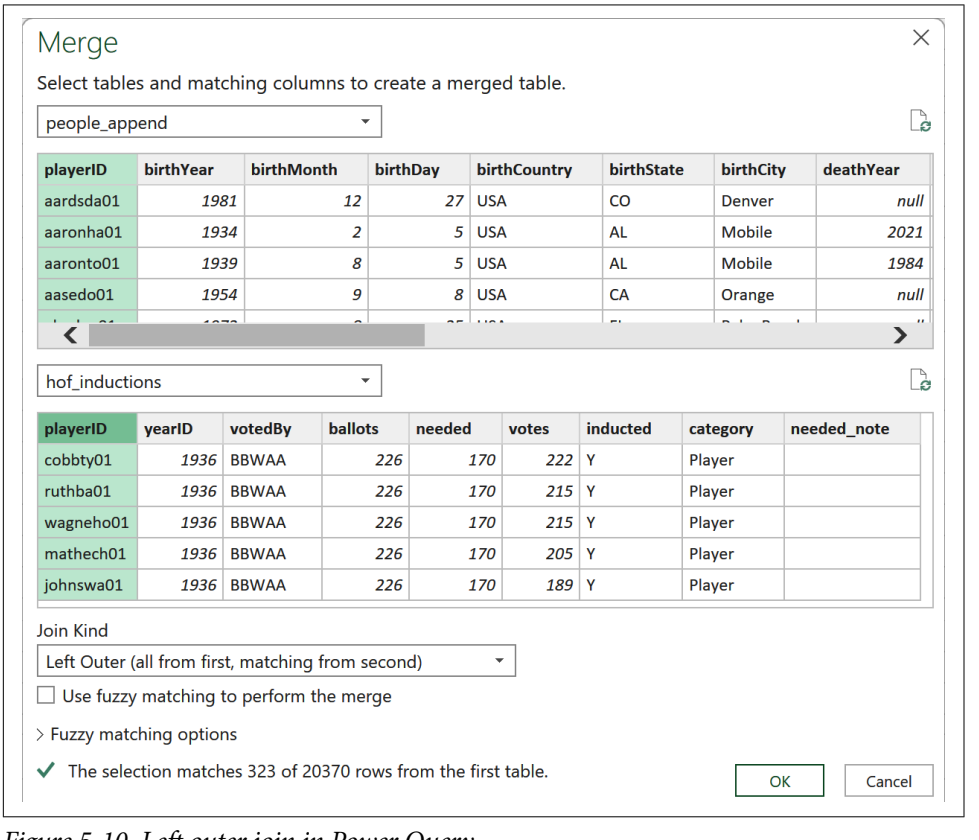


Figure 5-9. Merging queries as new in Power Query

To complete the join, select `hof_inductions` from the second drop-down menu of the Merge dialog box. Click on `playerID` in both tables to identify it as the column to merge on. Finally, confirm Left Outer as the desired Join Kind in the third drop-down menu.

Your Merge dialog should look like Figure 5-10.



playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear
aardsda01	1981	12	27	USA	CO	Denver	null
aaronha01	1934	2	5	USA	AL	Mobile	2021
aaronto01	1939	8	5	USA	AL	Mobile	1984
aasedo01	1954	9	8	USA	CA	Orange	null

playerID	yearID	votedBy	ballots	needed	votes	inducted	category	needed_note
cobbty01	1936	BBWAA	226	170	222	Y	Player	
ruthba01	1936	BBWAA	226	170	215	Y	Player	
wagneho01	1936	BBWAA	226	170	215	Y	Player	
mathech01	1936	BBWAA	226	170	205	Y	Player	
johnswa01	1936	BBWAA	226	170	189	Y	Player	

Figure 5-10. Left outer join in Power Query

Click OK, and you will see the merge results in a query named `Merge1`. Go ahead and double-click that name to rename the query `people_left`.

Scroll to the right of your `people_left` dataset. The data in our query looks a little unusual right now, specifically the `hof_inductions` column, which is set to Table in every row of the data. This represents a nested table containing all the matched rows from the second table for a given row in the first table.

Go ahead and click the button next to the `hof_inductions` header and then OK to expand the nested data, as shown in Figure 5-11.

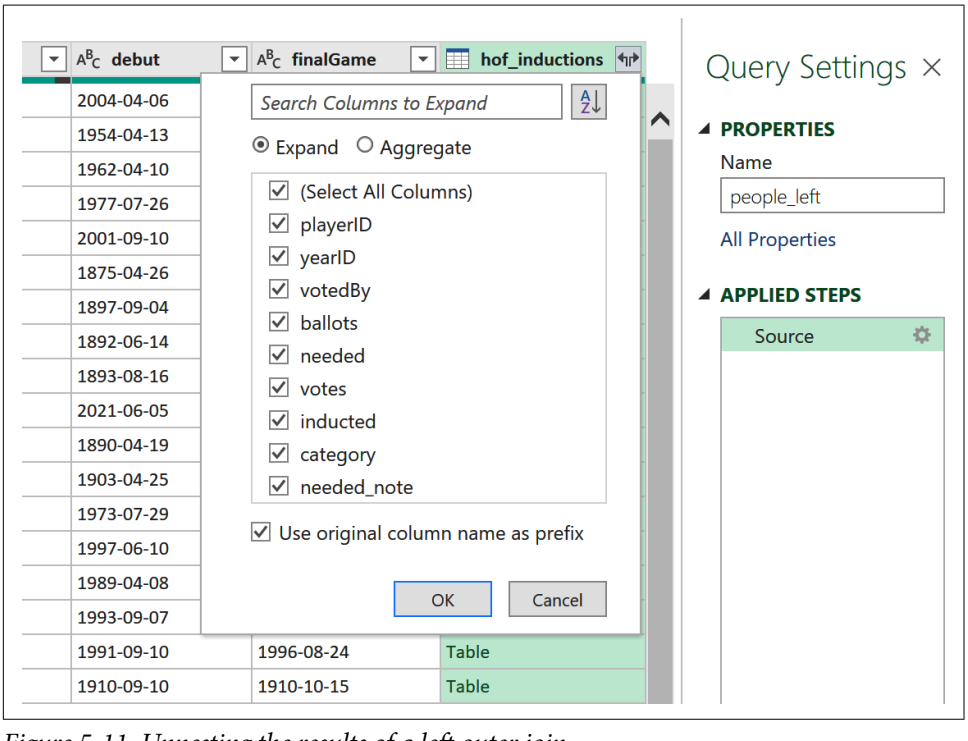


Figure 5-11. Unnesting the results of a left outer join

From this menu, you can select as many columns as you wish from the matching records in the `hof_inductions` table. You also have the option to prefix these columns with the name of their source table. For simplicity, we'll use the default approach, loading all columns with the prefix. However, for more streamlined queries in your actual work, you'll likely prefer to reduce the number of selected columns.

Load the results into a table in Excel.

The `people_left` table, like the original `people_append` table, has 20,370 records. This is because a left outer join includes all records from the `people` table, whether there's a matching record on the left or not. The results of the join are similar to those of `VLOOKUP()`, pulling related Hall of Fame records for each player. The advantage? It fetches all records from the `hof_inductions` table in one go and doesn't error out on unmatched records.

Inner Join: Only the Matches

By contrast, the *inner join* only keeps records in the resulting table where a match is found in both tables, as shown in Figure 5-12.

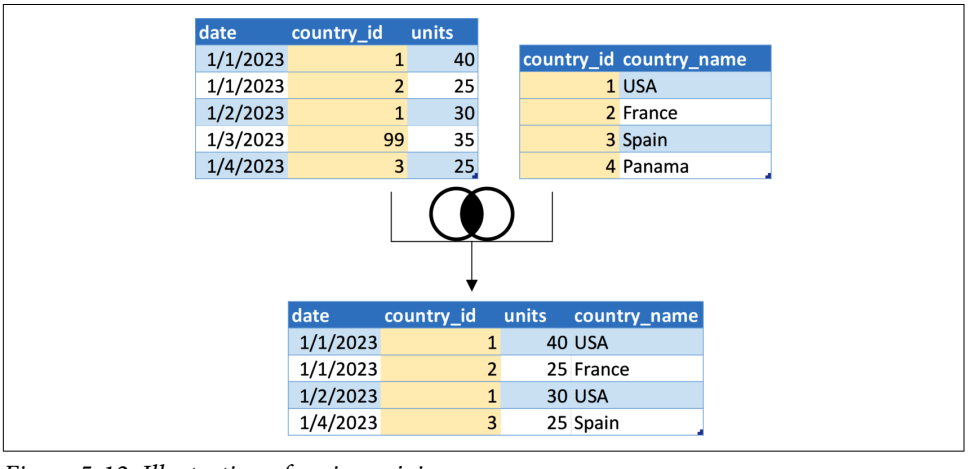


Figure 5-12. Illustration of an inner join

Following the logic of a left outer join, the `country_id` of 4 in the right table will not appear in the resulting table, as there is no corresponding match in the left table. Similarly, the `country_id` of 99 in the left table will also be excluded from the result for the same reason: it lacks a counterpart in the right table. For an entry to be included in the results, there must be a match in both tables.

This approach is useful for keeping only complete records and eliminating data with potential integrity issues. Applying these rules, the inner join will produce fewer rows than the left outer join.

Following the same steps as above, perform the inner join in Power Query. Select `people_append` inside the Editor, then Home → Merge Queries → Merge Queries as New. Your merge dialog box should look like Figure 5-13.

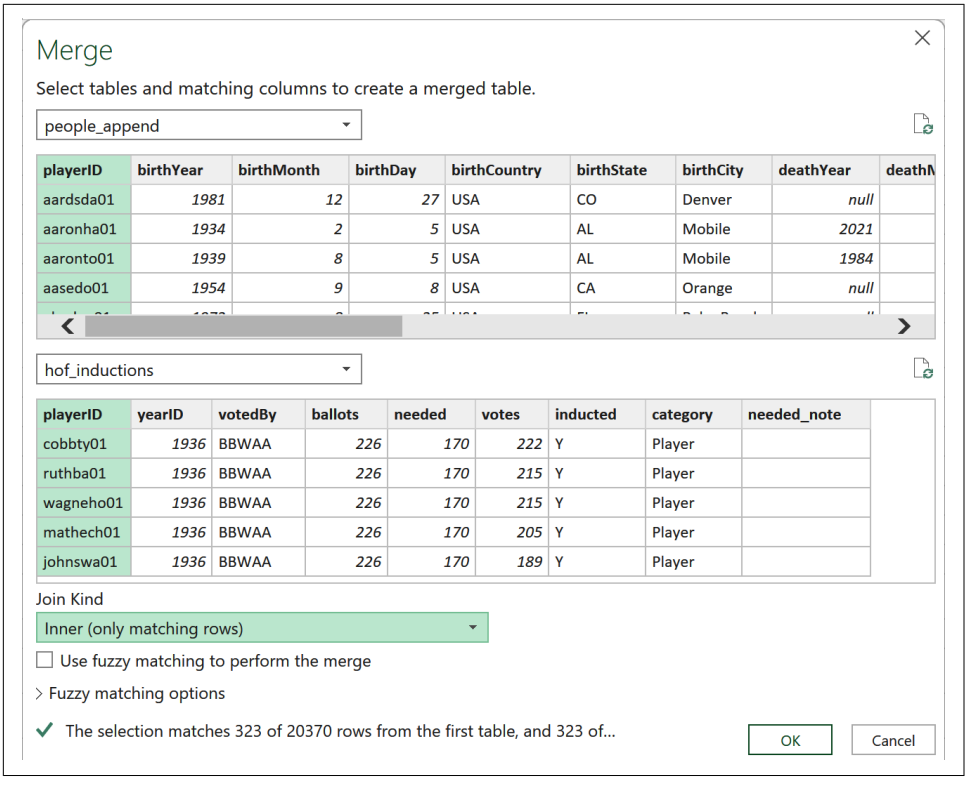


Figure 5-13. Merge settings to perform an inner join

You can expand the matching columns from the nested table in the same manner as with the left outer join, then rename the query `people_inner`. Load the resulting query into a table. This table consists of only 323 records.

The distinction is quite straightforward: an inner join only returns records that have a corresponding match in both tables. Not all `playerID` instances appear in the `hof_inductions` table, because not every player has been inducted into the Hall of Fame, resulting in the absence of their `playerID`s in the merged table.

Managing Your Queries

Great job on loading and merging data from various sources and formats! As you delve deeper into Power Query, it's likely that you'll accumulate multiple queries within your workbook. Managing and comprehending how these queries interact and function together will become crucial.

Grouping Your Queries

Grouping queries in Power Query enhances organization and streamlines maintenance by categorizing related queries. This approach makes managing complex Excel projects more manageable. By grouping your queries, you clearly distinguish between root queries and their dependencies, such as appends and merges, that build on those foundational queries.

To try it out, head back to the Power Query Editor.

Under the Queries list on the left side of the Editor window, hold down Ctrl and select the source queries: `people_born_in_usa`, `people_born_outside_usa`, and `hof_inductions`. Right-click, then select Move To Group → New Group as in Figure 5-14.

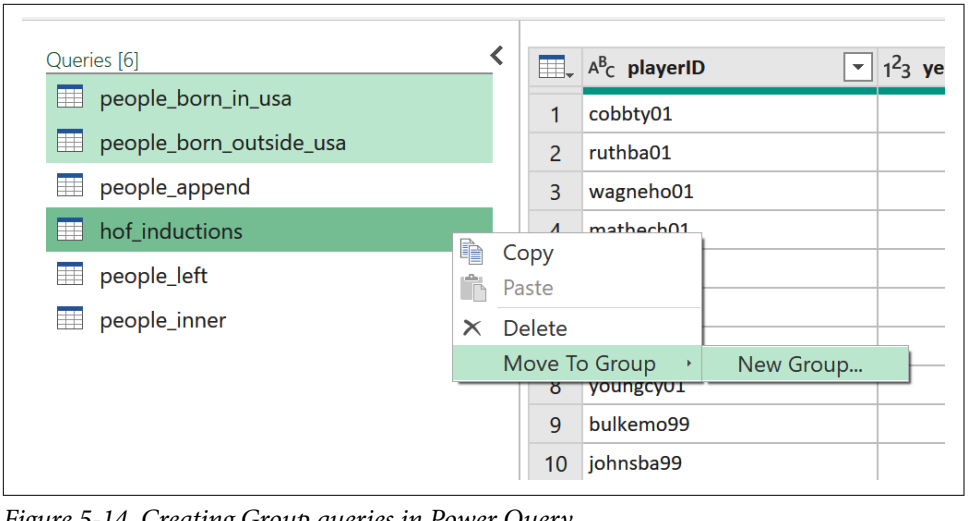


Figure 5-14. Creating Group queries in Power Query

When the New Group pop-up appears, call this group Sources. Click OK. You'll now see these three sources grouped into a folder in the Queries list. `hof_append`, `hof_left`, and `hof_inner` have also automatically been moved into a group called Other Queries, as shown in Figure 5-15.

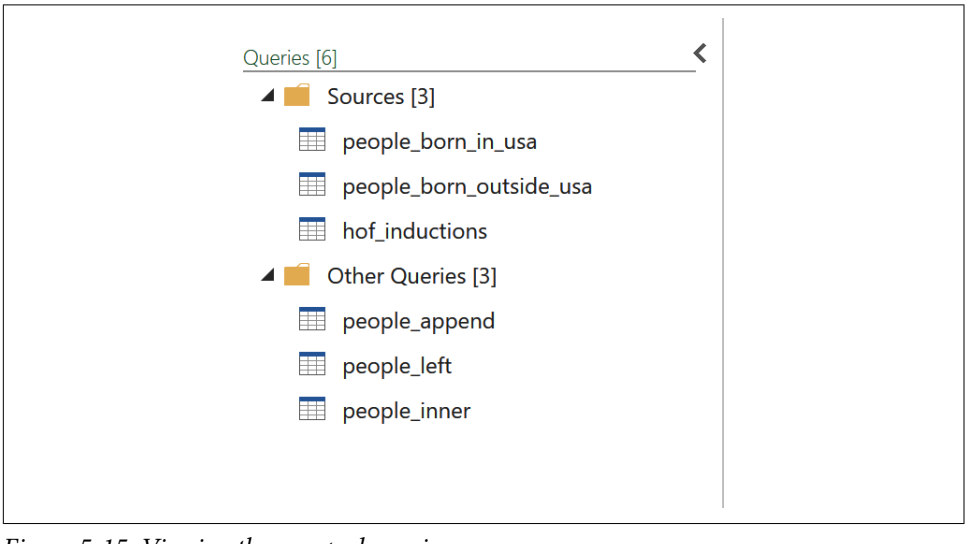


Figure 5-15. Viewing the grouped queries

Viewing Query Dependencies

View Query Dependencies displays how queries are interconnected, which helps identify the impact of changes and manage dependencies effectively, ensuring data integrity and reducing errors in complex projects. To check it out, go to View on the ribbon, then select Query Dependencies. It should look something like Figure 5-16.

Here, you can see which queries are derived directly from raw data sources, such as *.csv* files, the locations of these files, which sources are involved in merges or appends, which are loaded into the workbook, and more.

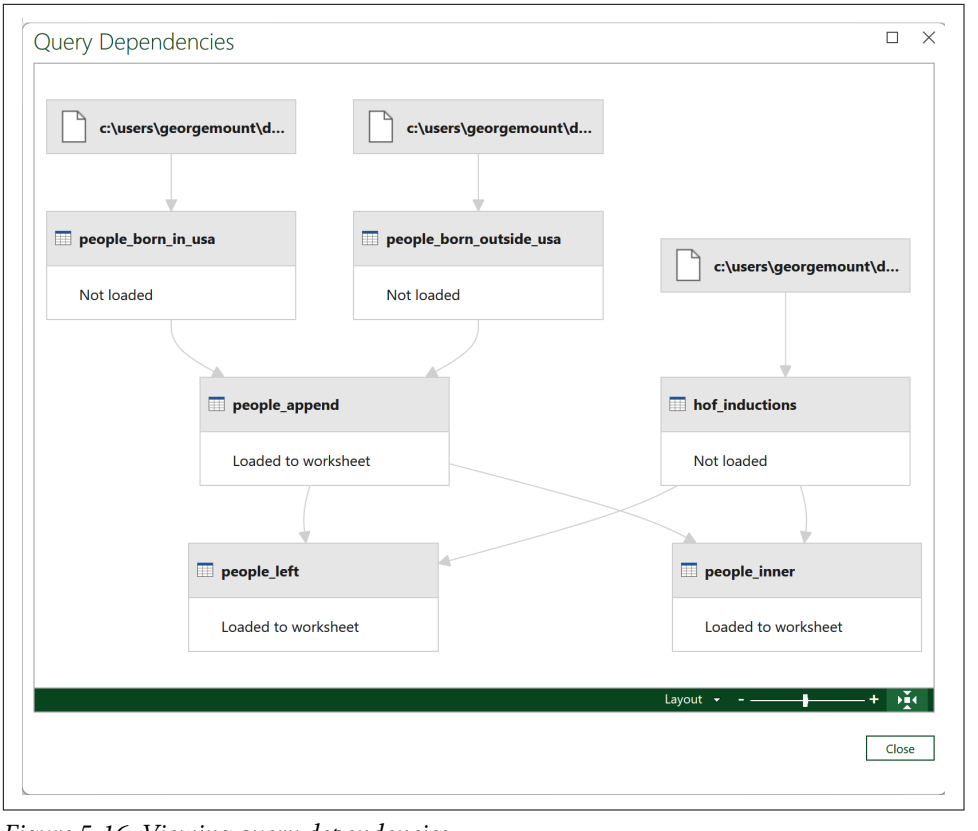


Figure 5-16. Viewing query dependencies

When you are finished viewing the Query Dependencies diagram, click Close.

Conclusion

This chapter highlighted the process of merging and appending data in Power Query and the importance of understanding the results. The tools allow for integration of various files, such as Excel workbooks and *.csv* files, into a unified dataset. By efficiently joining tables, data analysis becomes more streamlined.

Other join types exist within Power Query, such as the outer join that returns matching rows from both tables. For a detailed overview of join techniques, consult this Microsoft Learn article.

The overarching theme of Part I has been the indispensability of clean data for effective data analysis. Having established a strong foundation in data cleaning, the next phase of data analysis encompasses modeling and reporting using Power Pivot. This is the focus of Part II.

Exercises

Practice combining data sources into a single query using the files found in the *exercises\ch_05_exercises* folder in the book's companion repository. These files provide information on all outbound flights from the three major airports in New York City during the year 2013.

1. Append the *ewr-flights.csv*, *jfk-flights.csv*, and *lga-flights.csv* files, consisting of the flight records from Newark Liberty, John F. Kennedy, and LaGuardia airports, respectively. Name this query `flights`. (Hint: Select "Three or more tables" from the Append menu in Power Query to expedite this process.)
2. Merge this query with *planes.xlsx* using a left outer join, and then an inner join. Call the queries `flights_left` and `flights_inner`, respectively. How many records are returned for each? (Hint: Merge the tables based on `tailnum`).

You can find the solutions in *ch_05_solutions.xlsx* in the same folder.

PART II

Data Modeling and Analysis with Power Pivot

CHAPTER 6

First Steps in Power Pivot

Part I of this book concentrated on using Power Query for extracting data from diverse sources and transforming the outcomes into actionable datasets. Power Query does not operate as a standalone data analysis tool; instead, it serves as an intermediary to refine the data prior to analysis.

To continue on the analytics journey, Part II will focus on Power Pivot, a tool designed specifically for data analysis. With Power Pivot, users can establish relationships between data sources and generate advanced measures, enabling streamlined data analysis and reporting.

What Is Power Pivot?

Power Pivot is a relational data modeling and analysis tool directly integrated into Excel. It enables you to establish relationships across multiple tables and construct dashboards and reports based on PivotTables from this data model. Power Pivot offers a wide array of tools for creating robust analyses, significantly enhancing Excel's capabilities for business intelligence and reporting.

Why Power Pivot?

To understand the importance of Power Pivot in Excel data analysis, open *ch_06.xlsx* from the *ch_06* folder in the book's companion repository. Please note that this chapter does not include a solution file, as all steps are already completed for you.

In the `sales` worksheet, there are three tables that contain data related to sales, locations, and products. Suppose you wish to assign the correct product and branch names to each sales transaction for enhanced clarity.

In Excel, there are several ways to accomplish this. A popular method is to use the `VLOOKUP()` function to pull values from one table into another, as shown in Figure 6-1.

G2 =VLOOKUP([@[branch_id]], branches_lookup, 2, FALSE)

trans_id	trans_date	branch_id	product_id	quantity	total_price	branch_name	product_name	product_price
1	5/1/2023	1	1	10	$99.90	Scranton	Copy Paper	$9.99
2	5/2/2023	1	2	5	$12.45	Scranton	Sticky Notes	$2.49
3	5/3/2023	2	1	20	$199.80	Stamford	Copy Paper	$9.99
4	5/4/2023	3	3	2	$39.98	Nashua	Printer Ink	$19.99
5	5/5/2023	1	2	15	$149.85	Scranton	Sticky Notes	$2.49
6	5/5/2023	2	5	3	$14.97	Stamford	Legal Pads	$4.99
7	5/6/2023	2	2	10	$24.90	Stamford	Sticky Notes	$2.49
8	5/7/2023	1	4	8	$55.92	Scranton	Envelopes	$6.99
9	5/8/2023	3	3	5	$99.95	Nashua	Printer Ink	$19.99
10	5/8/2023	3	1	12	$119.88	Nashua	Copy Paper	$9.99
11	5/9/2023	1	2	7	$17.43	Scranton	Sticky Notes	$2.49
12	5/10/2023	2	4	3	$20.97	Stamford	Envelopes	$6.99
13	5/10/2023	1	5	10	$49.90	Scranton	Legal Pads	$4.99
14	5/11/2023	2	1	4	$79.96	Stamford	Copy Paper	$9.99
15	5/12/2023	3	2	6	$14.94	Nashua	Sticky Notes	$2.49
16	5/12/2023	1	4	5	$34.95	Scranton	Envelopes	$6.99
17	5/13/2023	2	1	8	$79.92	Stamford	Copy Paper	$9.99
18	5/14/2023	3	5	15	$74.85	Nashua	Legal Pads	$4.99
19	5/15/2023	1	3	3	$59.97	Scranton	Printer Ink	$19.99
20	5/15/2023	2	4	10	$69.90	Stamford	Envelopes	$6.99

branch_id	branch_name
1	Scranton
2	Stamford
3	Nashua

product_id	product_name	product_price
1	Copy Paper	$9.99
2	Sticky Notes	$2.49
3	Printer Ink	$19.99
4	Envelopes	$6.99
5	Legal Pads	$4.99

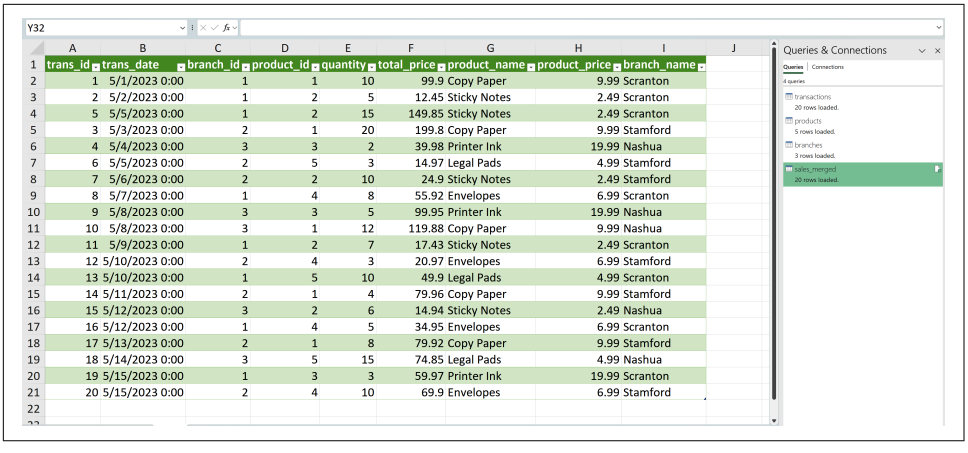
Figure 6-1. Consolidating data sources with `VLOOKUP()`

Although the `VLOOKUP()` function is frequently used, it comes with limitations. As pointed out in Chapter 5, lookup function outputs are static, merely augmenting existing tables instead of creating a fresh data source. Scaling this method becomes cumbersome when crafting lookup statements column by column.

Using lookup functions means Excel has to retain and sift through all the lookup data in its memory. As the volume of data grows and more lookups are executed, workbooks can become sluggish or even freeze. I dub these oversized and cumbersome Excel datasets "Frankentables."

In Chapter 5, you were introduced to a more efficient method of merging data sources using Power Query. If you were to use this method, as seen in Figure 6-2, you'd get a new table without formulas, but matching the dimensions of the earlier results achieved through lookups (assuming a left outer join was applied).

Power Query is more versatile and efficient compared to lookup functions, but it might not always be the optimal choice for every task. Just like `VLOOKUP()`, it consolidates each data point into a flat table, leading to larger file sizes and record duplication. Remember, Power Query's primary role is data *cleaning*, not data analysis. It lacks the functionality to create advanced measures like year-to-date calculations or dynamic aggregations.



trans_id	trans_date	branch_id	product_id	quantity	total_price	product_name	product_price	branch_name
1	5/1/2023 0:00	1	1	10	99.9	Copy Paper	9.99	Scranton
2	5/2/2023 0:00	1	2	5	12.45	Sticky Notes	2.49	Scranton
5	5/5/2023 0:00	1	2	15	149.85	Sticky Notes	2.49	Scranton
3	5/3/2023 0:00	2	1	20	199.8	Copy Paper	9.99	Stamford
4	5/4/2023 0:00	3	3	2	39.98	Printer Ink	19.99	Nashua
6	5/5/2023 0:00	2	5	3	14.97	Legal Pads	4.99	Stamford
7	5/6/2023 0:00	2	2	10	24.9	Sticky Notes	2.49	Stamford
8	5/7/2023 0:00	1	4	8	55.92	Envelopes	6.99	Scranton
9	5/8/2023 0:00	3	3	5	99.95	Printer Ink	19.99	Nashua
10	5/8/2023 0:00	3	1	12	119.88	Copy Paper	9.99	Nashua
11	5/9/2023 0:00	1	2	7	17.43	Sticky Notes	2.49	Scranton
12	5/10/2023 0:00	2	4	3	20.97	Envelopes	6.99	Stamford
13	5/10/2023 0:00	1	5	10	49.9	Legal Pads	4.99	Scranton
14	5/11/2023 0:00	2	1	4	79.96	Copy Paper	9.99	Stamford
15	5/12/2023 0:00	3	2	6	14.94	Sticky Notes	2.49	Nashua
16	5/12/2023 0:00	1	4	5	34.95	Envelopes	6.99	Scranton
17	5/13/2023 0:00	2	1	8	79.92	Copy Paper	9.99	Stamford
18	5/14/2023 0:00	3	5	15	74.85	Legal Pads	4.99	Nashua
19	5/15/2023 0:00	1	3	3	59.97	Printer Ink	19.99	Scranton
20	5/15/2023 0:00	2	4	10	69.9	Envelopes	6.99	Stamford

Figure 6-2. Consolidating data sources with Power Query merges

For a more powerful, efficient analysis experience, it's best to combine these data sources by establishing a relational data model using Power Pivot.

Why Merge in Power Query at All?

If you're confused that Chapter 5 touted the superiority of Power Query over lookup functions for combining data, only to see Power Query itself get upstaged by Power Pivot, there's a reason for this seeming overengineering of solution alternatives.

For specific tasks, such as one-off analyses or dealing with nonrelational data sources, Power Query offers a direct and efficient approach. Additionally, Power Query adeptly handles variations in data granularity. For instance, if you're working with monthly sales figures and daily foot traffic data, Power Query can aggregate the daily figures into monthly totals, ensuring consistency in data granularity across datasets.

However, while Power Query is invaluable for certain preparatory tasks and simplifications, it may not always be the optimal choice for more sophisticated analyses. This is where Power Pivot comes into play, offering advanced relational modeling capabilities. Particularly for large datasets, Power Pivot's ability to create intertable relationships enhances both the efficiency and flexibility of the data model.

Both tools have distinct strengths, and their combined use can cater to a wide spectrum of data processing needs.

Table 6-1 summarizes the pros and cons for each method of combining data sources.

Table 6-1. Comparison of combining data sources with XLOOKUP(), Power Query, and Power Pivot

Tool	Pros	Cons
XLOOKUP()	• Easy to understand • Available in native Excel	• Limited output flexibility • Columns looked up one at a time • Memory-intensive
Power Query	• More control over output • Easier to audit and maintain	• Relational joins can be confusing • Additional overhead in loading to Power Query
Power Pivot	• Can create complex data models • Built-in calculation and aggregation functions	• Complex to set up data model • Steep learning curve • Relational modeling unfamiliar to many Excel users

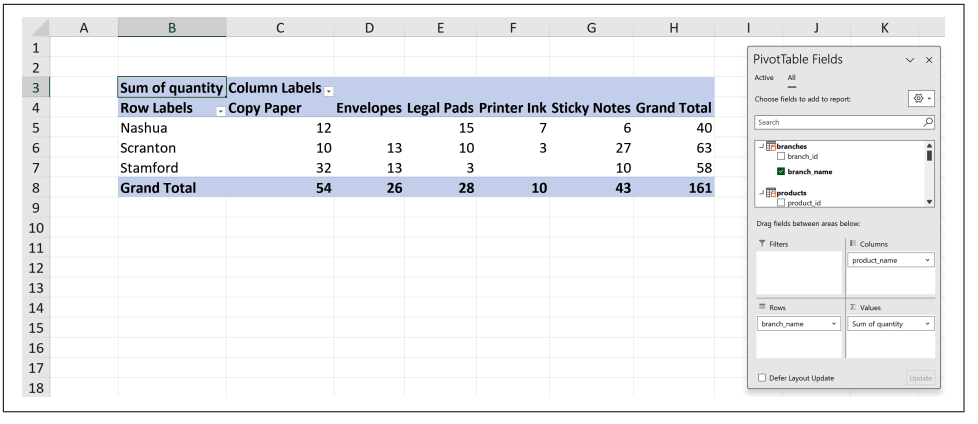
Power Pivot and the Data Model

Power Pivot operates within a Data Model, where relationships are established and managed. This approach permits the creation of a PivotTable from multiple sources without physically merging them.

Making use of the DAX formula language, Power Pivot handles intricate calculations on the data model, spanning time intelligence, rankings, percentiles, and more.

A major advantage lies in Power Pivot's ability to efficiently manage numerous data sources. It doesn't necessitate storing a memory-intensive "Frankentable" and computes DAX measures as needed. However, mastering Power Pivot can prove challenging due to its steep learning curve, particularly when dealing with data sources not consolidated into a single table.

On the `sales_pp` worksheet of *ch_06.xlsx*, I have created a Data Model consisting of the three sales data sources and loaded the results to a PivotTable. Now I am able to analyze and calculate based across all related tables, as shown in Figure 6-3.



Sum of quantity	Column Labels					
Row Labels	Copy Paper	Envelopes	Legal Pads	Printer Ink	Sticky Notes	Grand Total
Nashua	12		15	7	6	40
Scranton	10	13	10	3	27	63
Stamford	32	13	3		10	58
Grand Total	**54**	**26**	**28**	**10**	**43**	**161**

Figure 6-3. Consolidating data sources with Power Pivot relationships

Duplicate Table Names in the Power Pivot PivotTable

In the Power Pivot examples featured in this book, each table name is displayed twice in the resulting PivotTable: once with an orange cylinder icon and once without it. Always select the tables accompanied by the icon, as they are directly connected to the Data Model and include any measures added. Should you import your table into the Data Model from external sources, rather than using a table within the workbook, this issue of duplicate tables would be resolved.

To adjust these table relationships or add features like calculated columns or measures, it's necessary to load the Power Pivot add-in.

Loading the Power Pivot Add-in

To access Power Pivot, go to the File tab on the ribbon, then navigate to Options → Add-ins. In the Add-ins window, select COM Add-ins in the Manage dropdown and click Go, as shown in Figure 6-4.

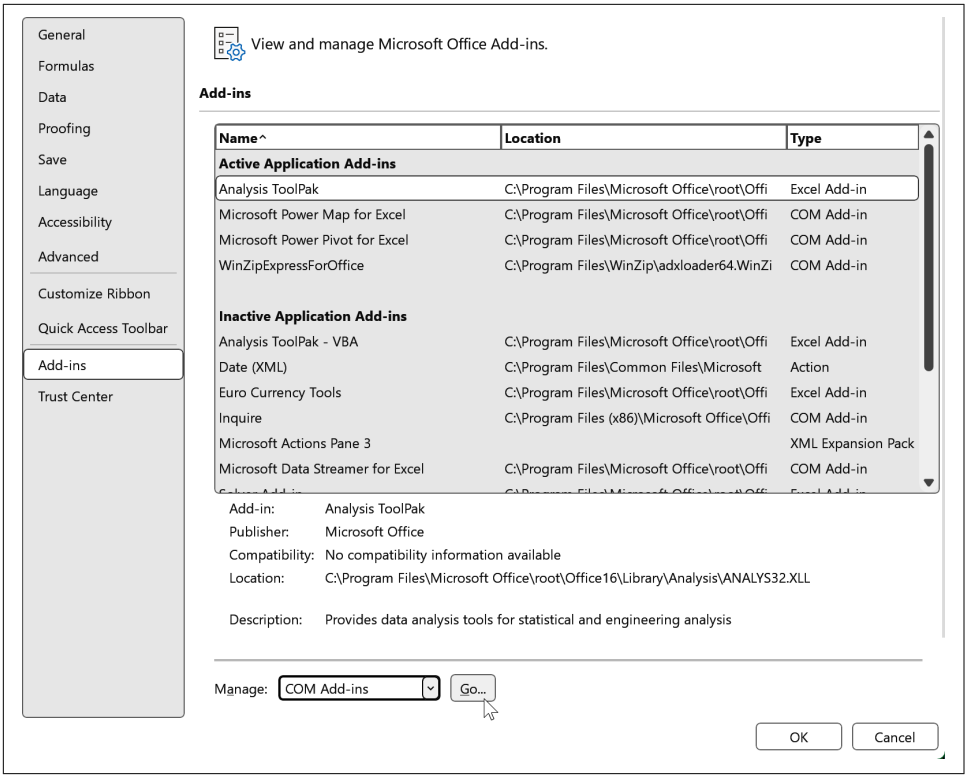


Figure 6-4. Loading the Power Pivot add-in

In the COM Add-ins dialog box, choose Microsoft Power Pivot for Excel and click OK, as shown in Figure 6-5. Now you're ready to use Power Pivot.

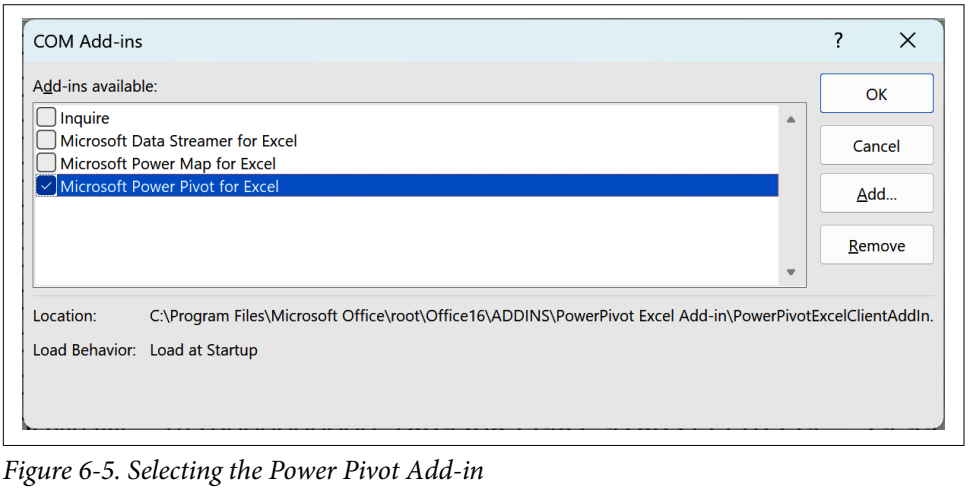


Figure 6-5. Selecting the Power Pivot Add-in

Check the ribbon for the new Power Pivot tab, as shown in Figure 6-6.

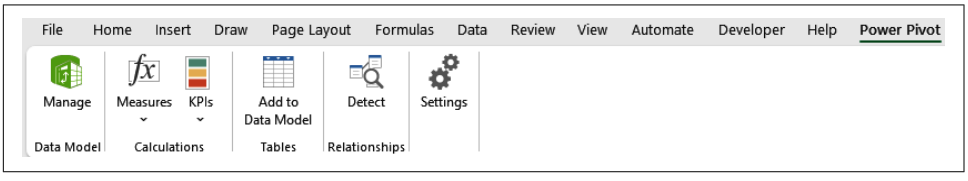


Figure 6-6. The Power Pivot add-in on the ribbon

A Brief Tour of the Power Pivot Add-In

The Power Pivot tab in the ribbon offers various options for creating and maintaining your Data Model along with related features. Let's look at each of these options, providing a high-level overview.

Data Model

Selecting Manage in Power Pivot opens a dedicated Power Pivot interface, showcasing the tables within your Data Model. It allows you to visualize the relationships among these tables and provides other functionalities. Take some time to explore this editor, and simply close it once you are finished. As you progress through the subsequent chapters of Part II, you'll gain additional experience working within this interface.

Calculations

In the Calculations group of the Power Pivot tab, you can create calculated measures and key performance indicators (KPIs), which we'll cover in more detail in later chapters of Part II:

Measures
: Measures in Power Pivot use the DAX language to execute calculations, aggregate data, and conduct advanced data manipulation and analysis. They aggregate values; compute totals, averages, and percentages; and are crucial to advanced Excel analytics.

KPIs
: A KPI is a measurable value that illustrates how effectively a company or organization is achieving its primary business goals. KPIs are essential for assessing success in meeting targets, and they play a crucial role in monitoring progress and steering decision-making processes. Power Pivot enables users to create and display KPIs within PivotTable-based dashboards and reports.

Tables

This section of Power Pivot allows you to import a workbook table into the Data Model. However, it is advisable to import data through Power Query, as discussed in Chapter 7. Power Query provides connections to a wide range of data sources, such as external workbooks and *.csv* files, covered in Part I, and it also enables data cleaning before creating the Data Model.

Relationships

This feature enables the automatic identification and creation of relationships between tables within the Data Model. Although it's a valuable tool worth exploring after mastering the basics covered in this book, it's crucial to have sufficient understanding to discern when a Data Model has been accurately constructed. For this reason, our focus will be on manually creating relationships, rather than relying on this automated option.

Settings

These settings enhance Data Model calculation performance and shed light on potential issues. Their in-depth use is beyond this book's scope.

Conclusion

This chapter unveiled Power Pivot's ability to streamline data from multiple sources without combining it into one table, positioning it as a solution to combat "Frankentables," much like Power Query dispels common Excel myths. While Power Pivot's features can be daunting, especially for traditional Excel users, its capabilities are unparalleled.

Subsequent chapters of Part II will delve deeper into Power Pivot, examining the intricacies of Data Model creation and analysis. Power Pivot simplifies the process of uncovering insights, making informed decisions, and crafting sophisticated analyses in Excel.

Exercises

To check your understanding of the concepts covered in this chapter, answer the following review questions:

1. What is the purpose of the Power Pivot add-in, and what can it enable you to do?
2. Explain the role of the Data Model in Power Pivot and its significance in data analysis.
3. What is the basic role of DAX measures and key performance indicators in Power Pivot?
4. Compare Power Query joins with Power Pivot relationships in terms of combining data sources.
5. What are the drawbacks of using lookup functions like `VLOOKUP()` or `XLOOKUP()` to merge tables in Excel?

Example answers to these questions can be found in the *exercises\ch_06_exercises* folder in the book's companion repository.

CHAPTER 7

Creating Relational Models in Power Pivot

Chapter 6 introduced the fundamentals of Power Pivot as an effective tool for data analysis and reporting, especially when working with data from multiple sources. This chapter offers a demonstration on how to use Power Pivot for relational data modeling.

Connecting Data to Power Pivot

As highlighted in Chapter 6, the Data Model serves as the foundation of Power Pivot, facilitating the creation and management of table relationships for effective data computation and analysis. Power Pivot streamlines this task with an intuitive drag-and-drop interface. This chapter delves deeper into the Data Model, using the *ch_07.xlsx* file. This file contains a retail sales dataset often referenced within the analytics community and is found in the *ch_07* folder among the book's resources.

In the example from Chapter 6, the Data Model was predefined. In this chapter, we'll need to define it manually.

Although direct connections to data sources via Power Pivot are feasible, it's recommended to channel data through Power Query first. This approach provides a convenient platform to establish any recurring data cleaning procedures on these tables when necessary.

To get started, import the `orders` table into Power Query using the Data → From Table/Range steps. Bypass any data transformation steps and proceed to Close & Load → Close & Load To on the Home tab.

To load this query to Power Pivot, choose to create the data as a connection only, and then select "Add this data to the Data Model," as shown in Figure 7-1.

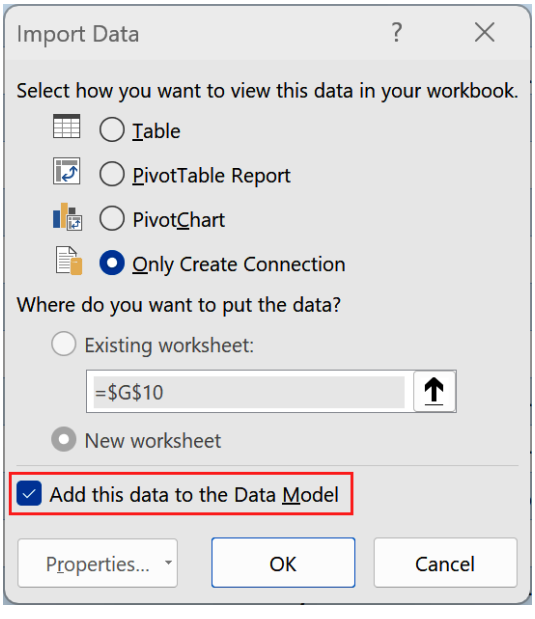


Figure 7-1. Loading a query from Power Query to Power Pivot

By following these steps, the query becomes accessible in Power Pivot for data modeling, but not in a separate worksheet for examination. It's important to note that the primary purpose of Power Pivot is to link and relate this table to others, construct DAX measures, and so on. Loading the data into the workbook as is would obscure these functionalities.

Repeat the process for both the `returns` and `users` tables to ensure that you have three connection-only queries incorporated into your workbook and added to the Data Model.

Creating Relationships

In Power Pivot, table relationships streamline data analysis, allowing for sophisticated models without the traditional, laborious data merging. This approach enhances consistency, reduces redundancy, and simplifies dataset management. Through these connections, users unlock dynamic and interactive exploration, boosting Excel's analytical power.

To establish relationships among the `orders`, `returns`, and `users` tables, navigate to the Power Pivot tab on the ribbon. Next, select Manage and choose Diagram View under the View group. Within Diagram View, the three tables and their respective column names will be displayed, as illustrated in Figure 7-2.

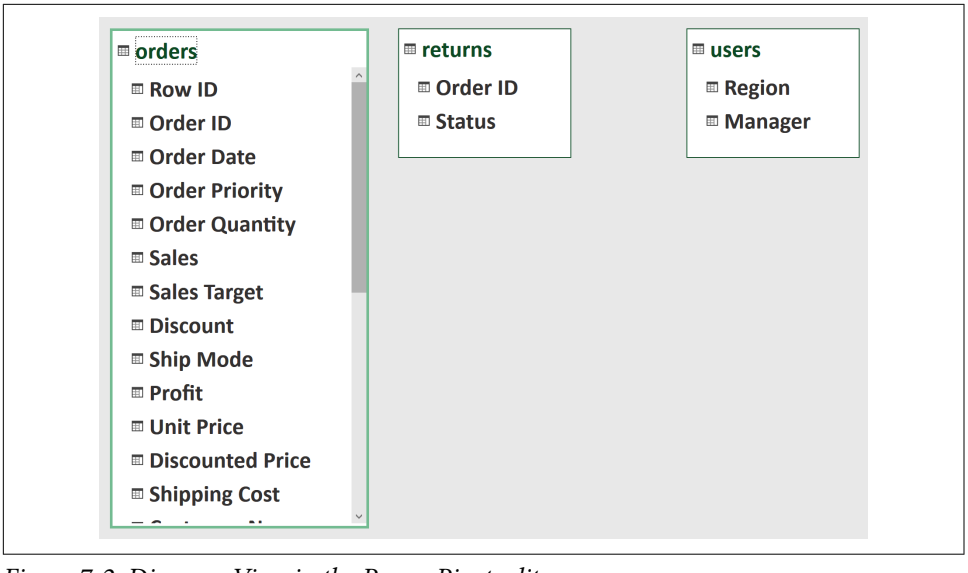


Figure 7-2. Diagram View in the Power Pivot editor

It's OK if your tables are not displayed in the exact order shown in Figure 7-2. We will establish the relationships between these tables, which will function regardless of their arrangement in Diagram View. Once we understand the content of the tables through these relationships, we'll rearrange them visually into a more coherent and efficient layout.

To create the first relationship, start by selecting the `orders` table. Then, go to the Design tab in Power Pivot and click on Create Relationship under the Relationships group, as shown in Figure 7-3.

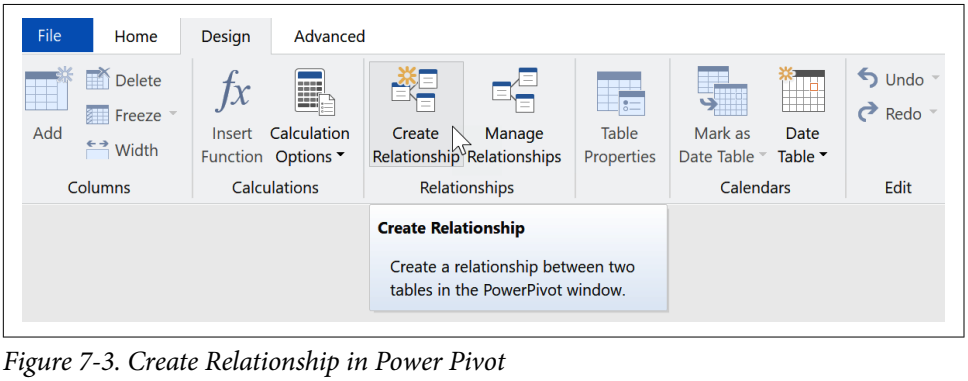


Figure 7-3. Create Relationship in Power Pivot

To establish a relationship between the `orders` and `returns` tables in Power Pivot, choose the `returns` table from the drop-down menu. Highlight the `Order ID` column in both tables. Complete the process by clicking OK, as shown in Figure 7-4.

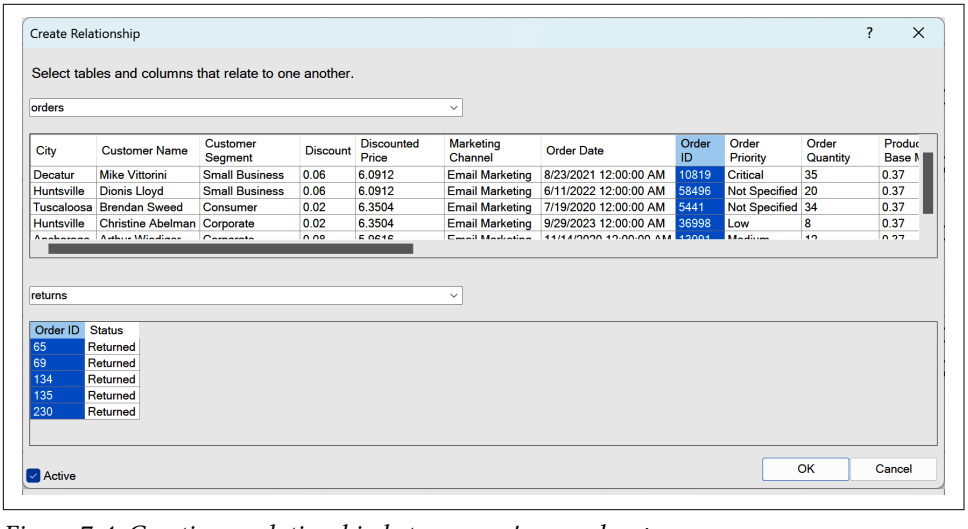


Figure 7-4. Creating a relationship between `orders` and `returns`

Much like the `VLOOKUP()` function, a data relationship is anchored on common columns between tables. Here, the shared column is `Order ID`. Upon setting this relationship and clicking OK, a line appears connecting the two tables, as shown in Figure 7-5.

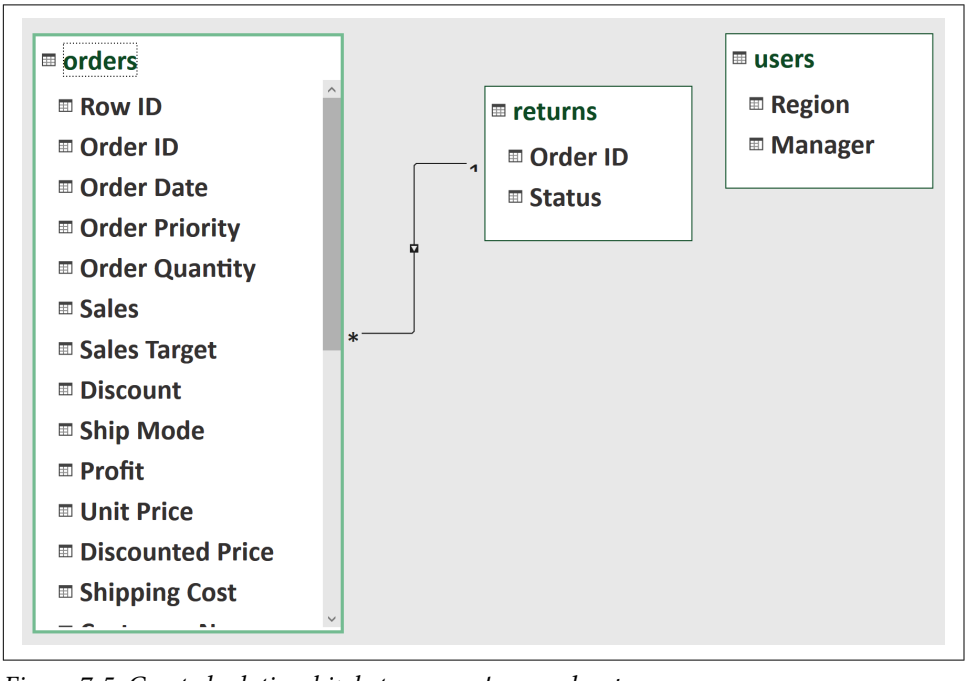


Figure 7-5. Created relationship between `orders` and `returns`

To establish the final relationship in the Data Model and connect all three tables, you can use the `Region` field present in both `orders` and `users`. Rather than using the Create Relationship option, a more efficient approach is to drag and drop the `Region` field between the two tables. This action will create the desired relationship, as illustrated in Figure 7-6.

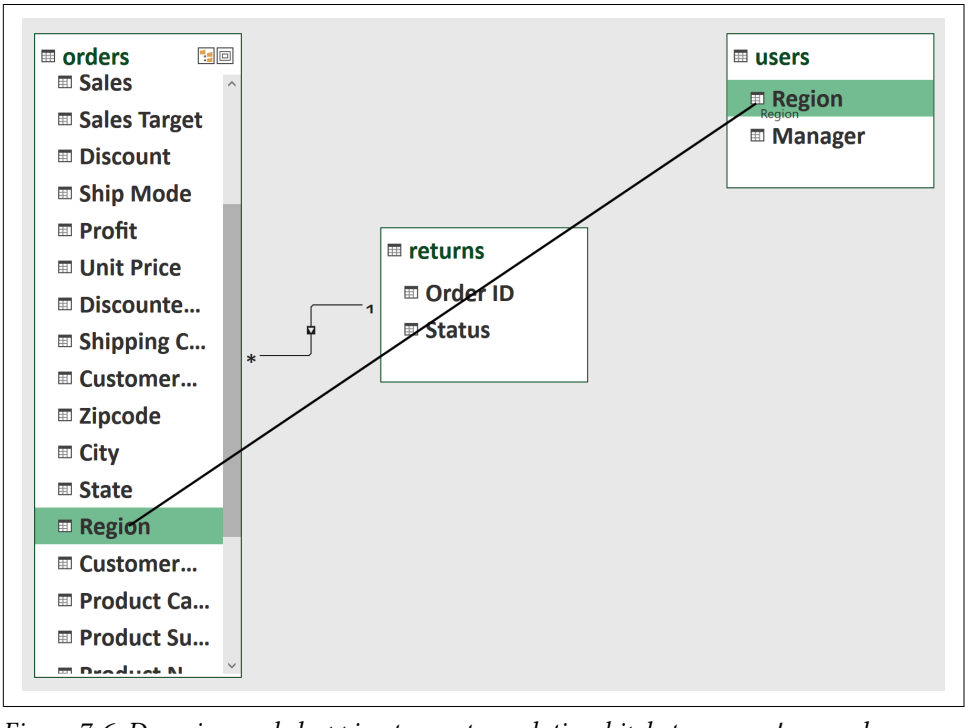


Figure 7-6. Dragging-and-dropping to create a relationship between `orders` and `users`

Identifying Fact and Dimension Tables

Once a comprehensive Data Model is created, a good next step is to identify its fact tables and dimension tables. *Fact tables* typically contain quantitative data that is suitable for calculations, such as averages, minimums, and maximums. On the other hand, *dimension tables* contain descriptive data that provides context to the measurements in the fact tables.

For example, the `orders` table consists of several measurable quantities, such as sales, profits, and units sold that can be summed, averaged, and so forth. These represent the core metrics of the business or process that you're analyzing. The presence these quantitative data indicate that this is a fact table.

Fact tables often lack contextual information that is crucial for data interpretation. In the case of the `orders` table, it may be useful to know the manager associated with each region of the company. The `users` table acts as a dimension table because it offers descriptive context; specifically, which manager oversees each region. Dimension tables play a vital role in slicing, dicing, and gaining a deeper understanding of the data.

Arranging the Diagram View

In real-world scenarios, it's common to encounter a dozen or more tables within the Data Model. Properly organizing the diagram is crucial for enabling users to understand the data effectively.

One useful technique is to position the fact table at the center of the diagram and surround it with dimension tables. This visual arrangement assists in comprehending the relationships and dependencies between the tables. To achieve this with the current data, click and drag the `returns` and `users` tables so that they are on either side of the `orders` table, as shown in Figure 7-7.

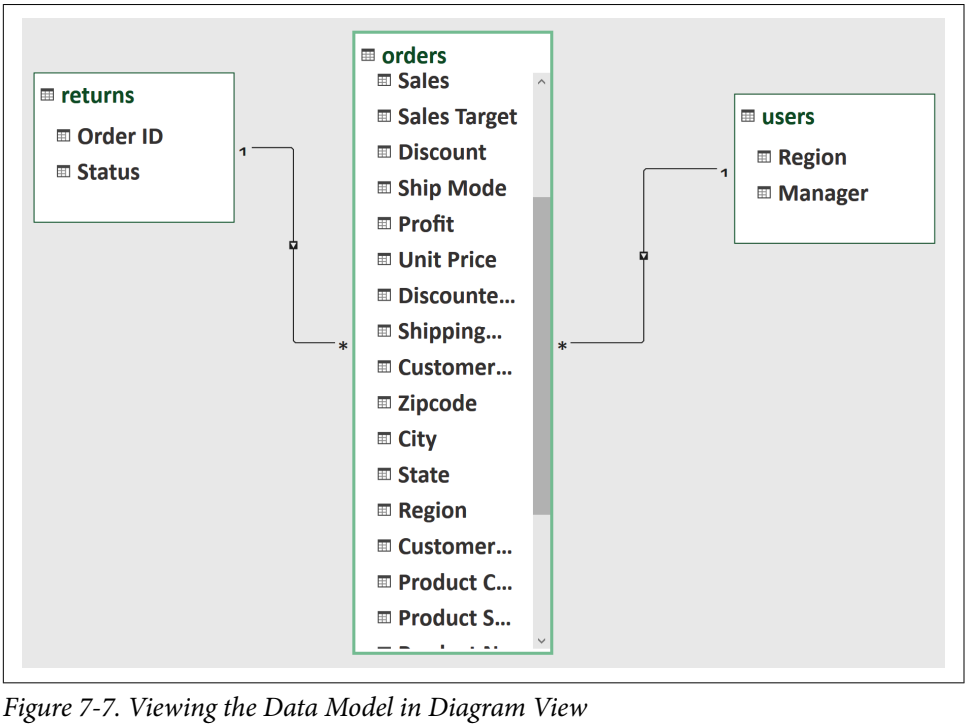


Figure 7-7. Viewing the Data Model in Diagram View

In a Data Model, when a fact table is central and flanked by dimension tables, as illustrated in Figure 7-7, this arrangement is termed a *star schema*. The star schema is a foundational concept in data model design. Its moniker is derived from the schema's visual representation, where the fact table anchors the center and the dimension tables emanate outward, mimicking the rays of a star.

Editing the Relationships

To modify defined relationships in Power Pivot, you have several options. First, you can right-click on any relationship line in Diagram View and select Edit Relationship to revisit the dialog where you can modify the related tables and columns. Second, you can temporarily disable or delete the relationship by right-clicking on the relationship lines and selecting from the menu.

Third, you can manage all relationships within the Data Model from a centralized location. To do this, navigate to the Design tab on the ribbon and click on Manage Relationships. This will provide a comprehensive view of all relationships within your Data Model, as depicted in Figure 7-8, enabling you to make necessary adjustments.

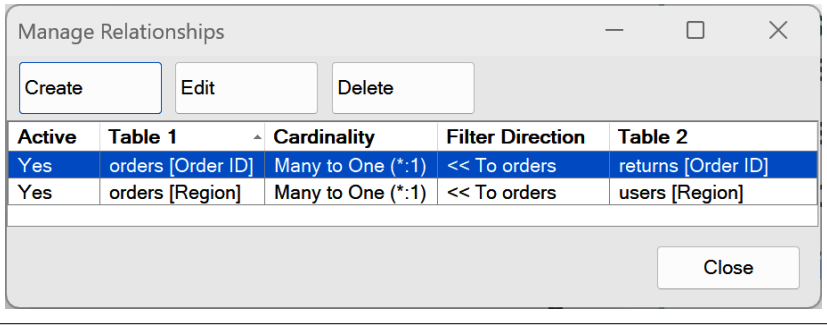


Figure 7-8. Managing relationships in the Data Model

The Manage Relationships dialog box—shown in Figure 7-8—provides insights into the cardinality and filter direction of each relationship. These ideas will be delved into later in this chapter.

Loading the Results to Excel

Once the Data Model is established, the next step is to transfer the results to Excel. This can be achieved by navigating to the Home tab in the Power Pivot editor and then selecting an option from the PivotTable drop-down menu, as illustrated in Figure 7-9.

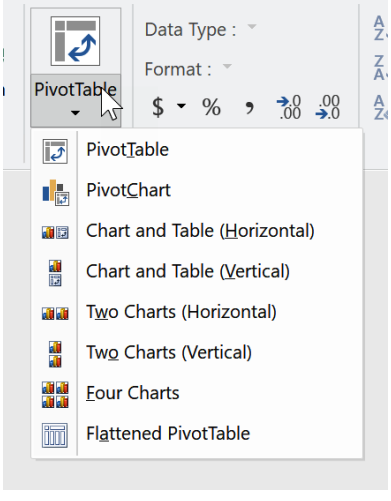


Figure 7-9. Power Pivot load options

Power Pivot offers various configuration options for loading a PivotTable into a workbook. These options often involve both PivotTables and PivotCharts, as Power Pivot is commonly used to create basic dashboards and reports. The final option, Flattened PivotTable, removes all subtotals and displays the data in a tabular, unnested format.

Select the PivotTable option. Click OK on the Create PivotTable dialog box to insert it into a new worksheet. You should now have something similar to Figure 7-10.

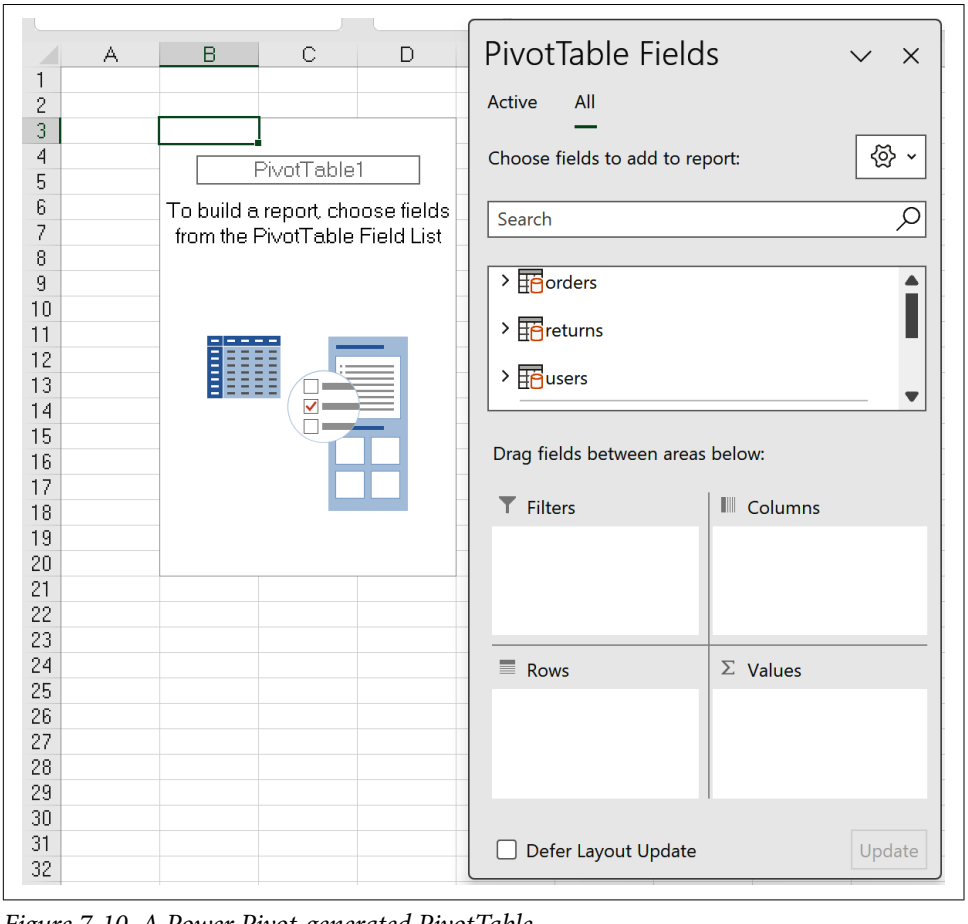


Figure 7-10. A Power Pivot-generated PivotTable

Now, drag the `Region` field from the `users` table to the Rows section of the PivotTable. Next, place `Sum of sales` from the `orders` table to the Values section. The Data Model will instantly use the relationships between these tables, specifically the shared `Region` value, to accurately perform the calculation. You can see the results in Figure 7-11.

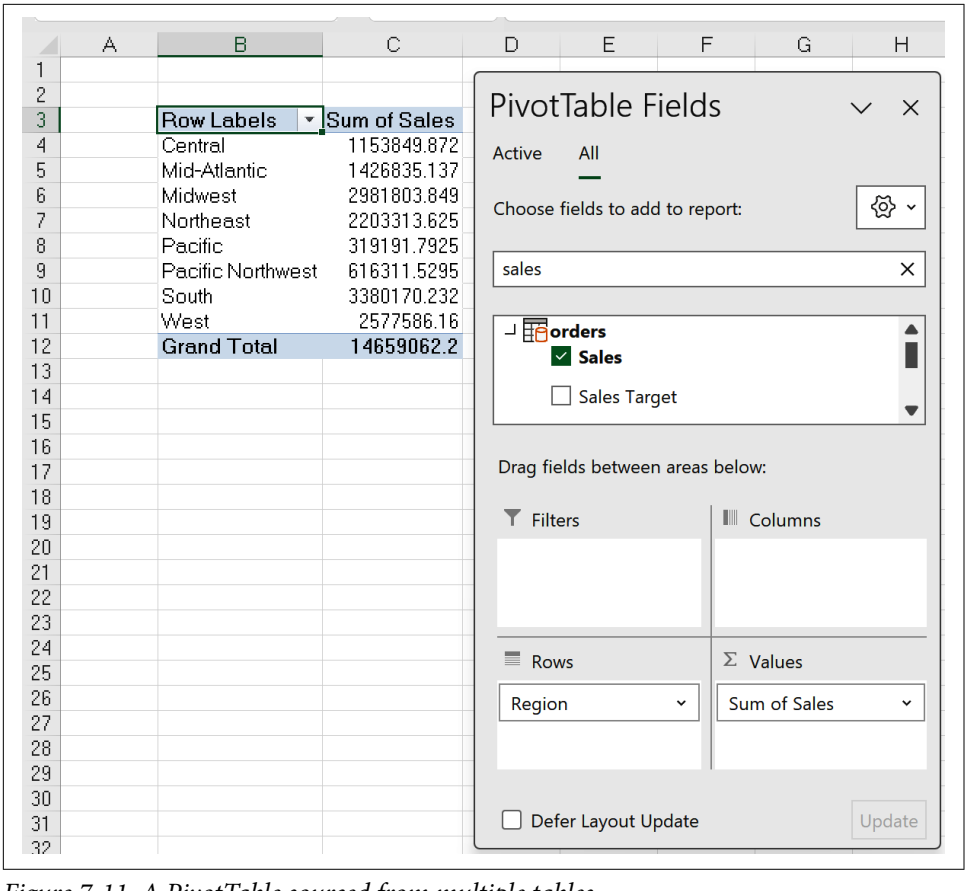


Figure 7-11. A PivotTable sourced from multiple tables

The results show that the `Sales` field is rounded to three decimal points, which is unusual, and the formatting doesn't clearly show currency, making it hard to read. It's possible to adjust these numbers in the worksheet or PivotTable, but a lasting fix is to format the data in the Power Pivot Data Model. To do this, go back to the Power Pivot tab on the ribbon and click on Manage.

On the Power Pivot Home tab, click on Data View in the View group. Select the `orders` worksheet tab at the bottom of Data View to access and operate on this table. Then, click the `Sales` column and adjust its format to Currency, also enabling the thousands separator, as seen in Figure 7-12.

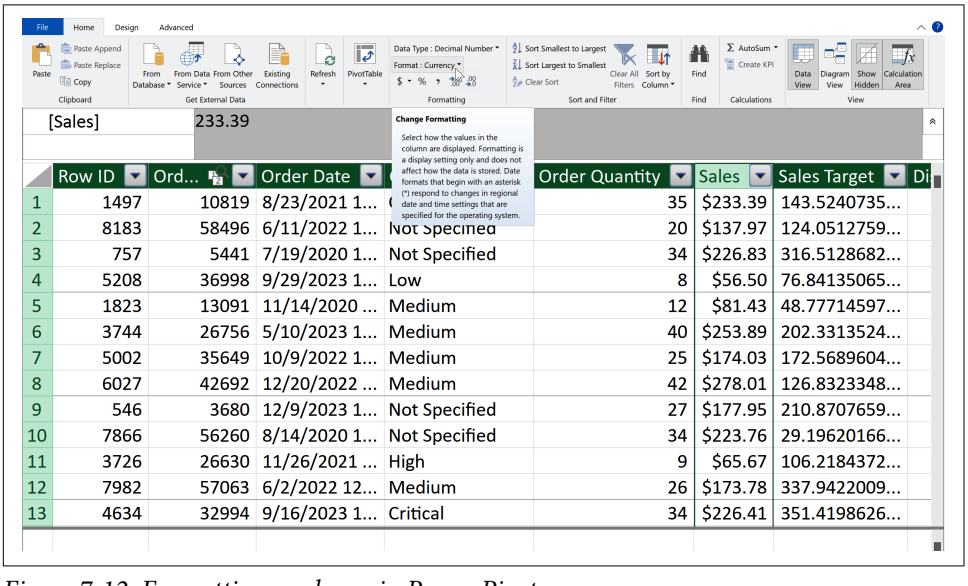


Figure 7-12. Formatting a column in Power Pivot

Feel free to make any other formatting adjustments to your source data while you are here.

After exiting Power Pivot, the enhancements will be clearly visible in the PivotTable, as illustrated in Figure 7-13.

This book will continue to make changes to the formatting of columns in Power Pivot without necessarily explicitly walking through the instructions.

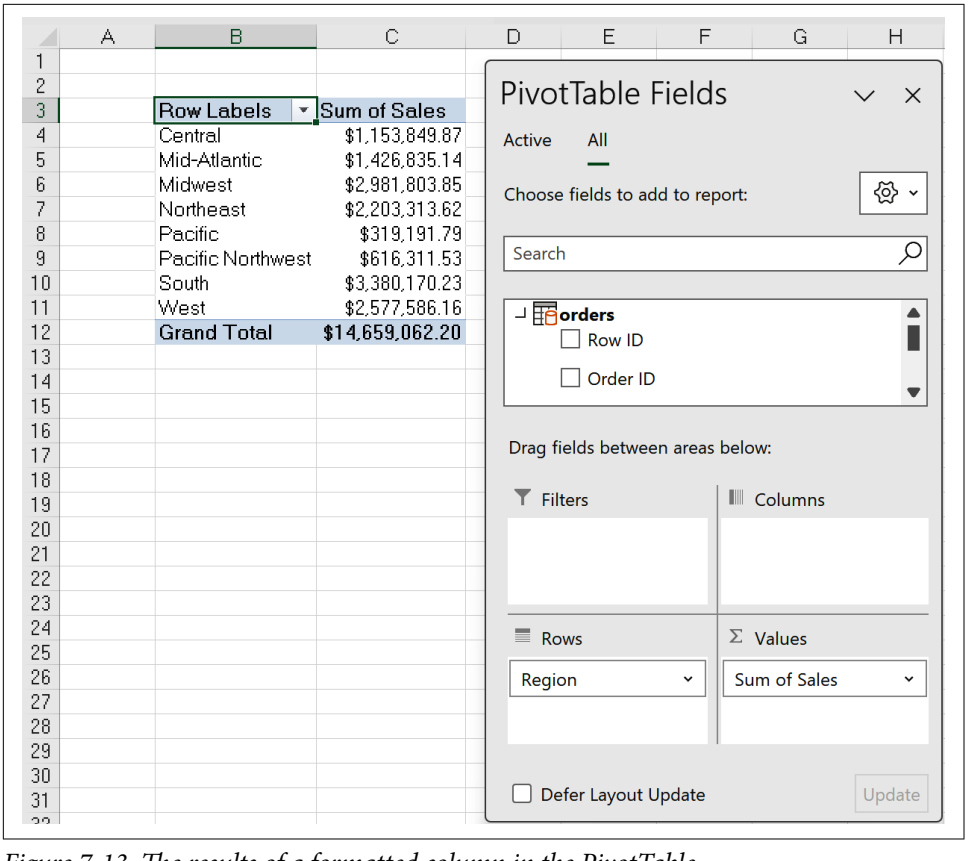


Figure 7-13. The results of a formatted column in the PivotTable

Understanding Cardinality

Challenges in Power Pivot results often emerge when numbers don't aggregate as expected or when specific fields become unusable due to issues in the Data Model's relationships. Such problems typically stem from an incomplete understanding of the Data Model's structure and its cardinality. Let's delve deeper into these elements.

The earlier section highlighted the significance of shared fields for creating relationships in Power Pivot. The count of unique records in each table plays a pivotal role in determining how relationships function within the Data Model. *Cardinality* pertains to the quantity of entries in one table that correlate to entries in another table.

One-to-One Cardinality

A *one-to-one* relationship represents the most straightforward form of cardinality, wherein each entry in one table corresponds uniquely to a single entry in another.

Consider a scenario where the Data Model comprises two tables: `product_details` and `supplier_details`, as illustrated in Figure 7-14.

`product_details`

Product ID	Product Name	Category	Sub-Category	Price ($)
P001	Adjustable Desk	Furniture	Tables	200.00
P002	Executive Chair	Furniture	Chairs	120.00
P003	Ballpoint Pen (Blue)	Office Supplies	Pens	1.00
P004	Printer Paper (500 sheets)	Office Supplies	Paper	5.00
P005	Monitor 24"	Technology	Monitors	150.00

`supplier_details`

Product ID	Supplier ID	Supplier Name	Address	Contact Number
P001	S001	FurniFix Inc.	123 Furniture St, NY	(123) 456-7890
P002	S002	ChairCrafters Ltd.	456 Chair Lane, LA	(234) 567-8901
P003	S003	PenMasters Corp.	789 Pen Ave, SF	(345) 678-9012
P004	S004	PaperStacks	101 Paper Rd, TX	(456) 789-0123
P005	S005	TechBrite	202 Tech Blvd, MI	(567) 890-1234

Figure 7-14. Example of a one-to-one relationship

In Figure 7-14, each record is identified by a unique `Product ID`, forming the basis of the relationship between the two tables.

Although such a structure can be useful, it's often not the most efficient. Merging the tables can minimize redundancy, decrease maintenance efforts, and enhance performance. Power Pivot, as a data modeling tool in Excel, doesn't provide an option for one-to-one cardinality, underscoring its limited application in real-world data models. Instead, Power Pivot is fine-tuned for one-to-many relationships.

One-to-Many Relationships

A *one-to-many* relationship indicates that several records in one table correspond to a single record in another table. Consider the example in Figure 7-15.

customers

Customer ID	Customer Name	Location
C001	Alice Smith	New York, NY
C002	Bob Johnson	Los Angeles, CA
C003	Charlie Brown	Chicago, IL

orders

Order ID	Product	Order Date	Amount ($)	Customer ID
O001	Adjustable Desk	2023-01-05	200.00	C001
O002	Ballpoint Pen (Blue)	2023-01-10	1.00	C001
O003	Executive Chair	2023-01-15	120.00	C002
O004	Monitor 24"	2023-01-20	150.00	C002
O005	Printer Paper (500 sheets)	2023-01-25	5.00	C003

Figure 7-15. Example of a one-to-many relationship

In this model, a customer from one table can have multiple related records in another table, such as orders. By storing related records in separate tables and connecting them via a single item per group, this method reduces data redundancy, streamlines updates and queries, and ensures data integrity. This efficient approach is key to building scalable, maintainable databases that accurately capture the complexities of business operations.

Many-to-Many Relationships

In scenarios where entities from two different tables have the capability to form multiple connections, a *many-to-many* relationship is present. Tools like Power Pivot do not directly accommodate these relationships. The common approach to managing such relationships is through the use of a *bridge* or *junction* table.

Consider tracking customer purchases across multiple promotions in a retail setting, such as in Figure 7-16.

customers

Customer ID	Customer Name
C101	Emily White
C102	Daniel Green
C103	Laura Blue

promotions

Promotion ID	Promotion Name	Date
P101	Summer Sale	2023-06-15
P102	Black Friday	2023-11-24
P103	New Year Bonanza	2024-01-01

Figure 7-16. Example of a many-to-many relationship

In this example, each customer and event is listed once, indicating that a customer may have made multiple purchases per promotion. To manage this complexity, we introduce a bridge table that maps which customers participated in which promotions, as shown in Figure 7-17.

customer-promotion (Bridge table)

Association ID	Customer ID	Promotion ID
A201	C101	P101
A202	C101	P102
A203	C102	P102
A204	C103	P103

Figure 7-17. Example of a many-to-many bridge table

This table simplifies the many-to-many relationship, representing each customer's engagement in specific promotions.

Why Does Cardinality Matter?

Cardinality plays a pivotal role in data modeling, ensuring data accuracy and consistency. In a one-to-many relationship, it's essential to confirm that each "one" entity corresponds uniquely to a "many" entity, and vice versa.

Though Power Pivot doesn't differentiate between one-to-one and one-to-many relationships, mastering this notion enhances data model performance in Power BI, which does cater to these distinctions. For a deeper dive into Power BI relationships, refer to Microsoft's documentation.

Grasping various cardinalities, such as one-to-one, one-to-many, and many-to-many, is crucial across all data modeling tools, not just Power Pivot. While Power Pivot emphasizes one-to-many relationships, understanding all cardinalities ensures organized data, preserved integrity, and smooth tool integration. This insight is invaluable

for troubleshooting and effective communication with data peers. In short, a thorough understanding of these principles offers adaptability in diverse data landscapes.

Understanding Filter Direction

As a relational data model, Power Pivot streamlines data analysis across multiple tables by leveraging common fields. Adjusting filters on these fields affects related tables, embodying the concept of *filter direction*, which is intrinsically tied to cardinality.

Within this workbook's Diagram View, the connection between the `users` and `orders` tables through the `Region` field is evident. A closer look at the line symbolizing this relationship reveals a small arrow pointing from `users` to `orders`, as shown in Figure 7-18.

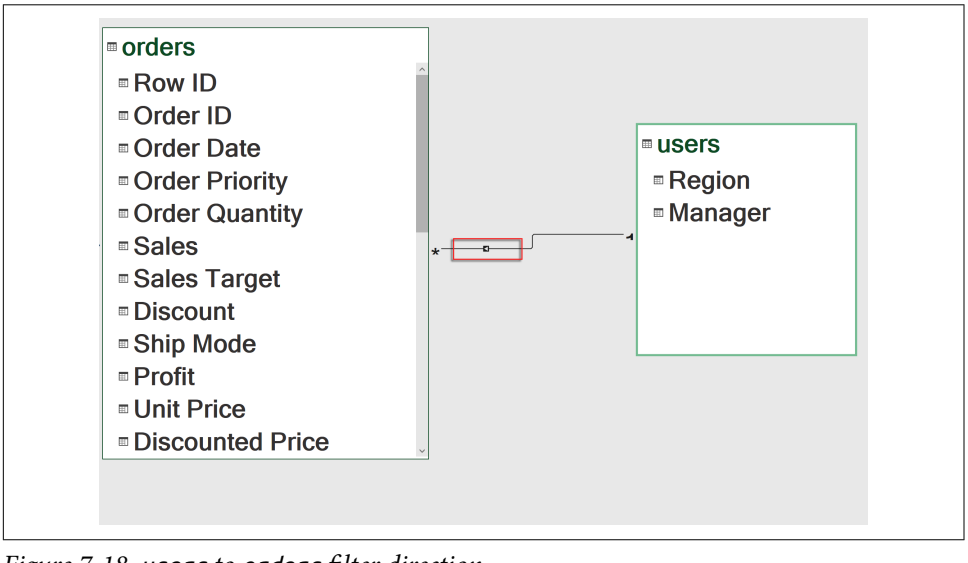


Figure 7-18. `users` to `orders` filter direction

The asterisk seen in Figure 7-18 indicates the "many" side of a one-to-many relationship between tables. This visual representation offers a quick glance at the relationship's nature and cardinality between the tables.

The arrow indicates the flow of filter effects from one table to the other. Applying filters to the left table influences the right table, but the reverse doesn't hold true.

Filtering orders with users

To understand the influence of filtering `orders` via the `users` table, begin by inserting a PivotTable from the Data Model into the workbook. Add the `Region` field from the `users` table to the Filters area and the `Sum of Sales` field from the `orders` table to the Values area. When adjusting the `Region` field, for instance by selecting Central, the PivotTable will showcase the sum of sales for the Central region, as demonstrated in Figure 7-19.

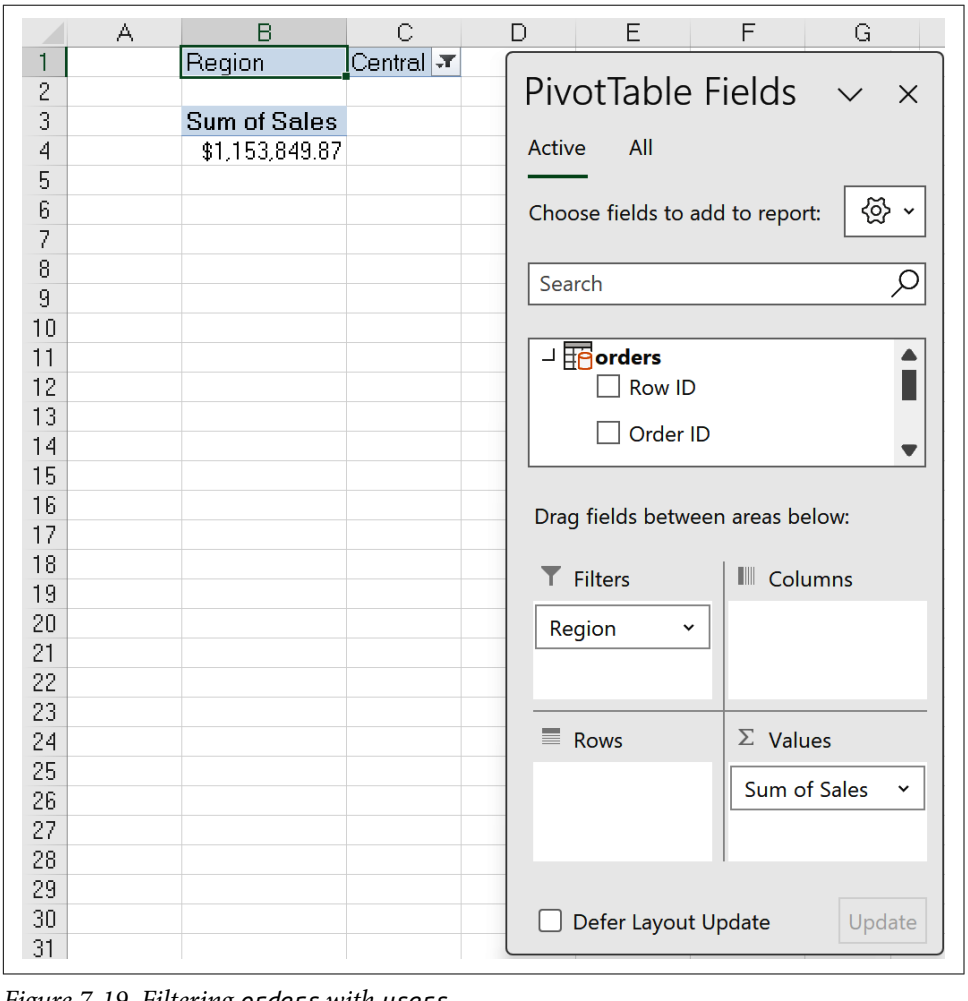


Figure 7-19. Filtering `orders` with `users`

The technical term for this phenomenon is that the filter is "propagated" from the `users` table to the `orders` table. This is the expected behavior when creating a filter, and the one you're probably used to.

Filtering users with orders

Now, consider a PivotTable using the `Region` field from the `orders` table in the Filters area and the `Manager` field from the `users` table in the Rows area.

Upon filtering for the Central region, something interesting happens: the data remains intact with no records omitted. This is depicted in Figure 7-20.

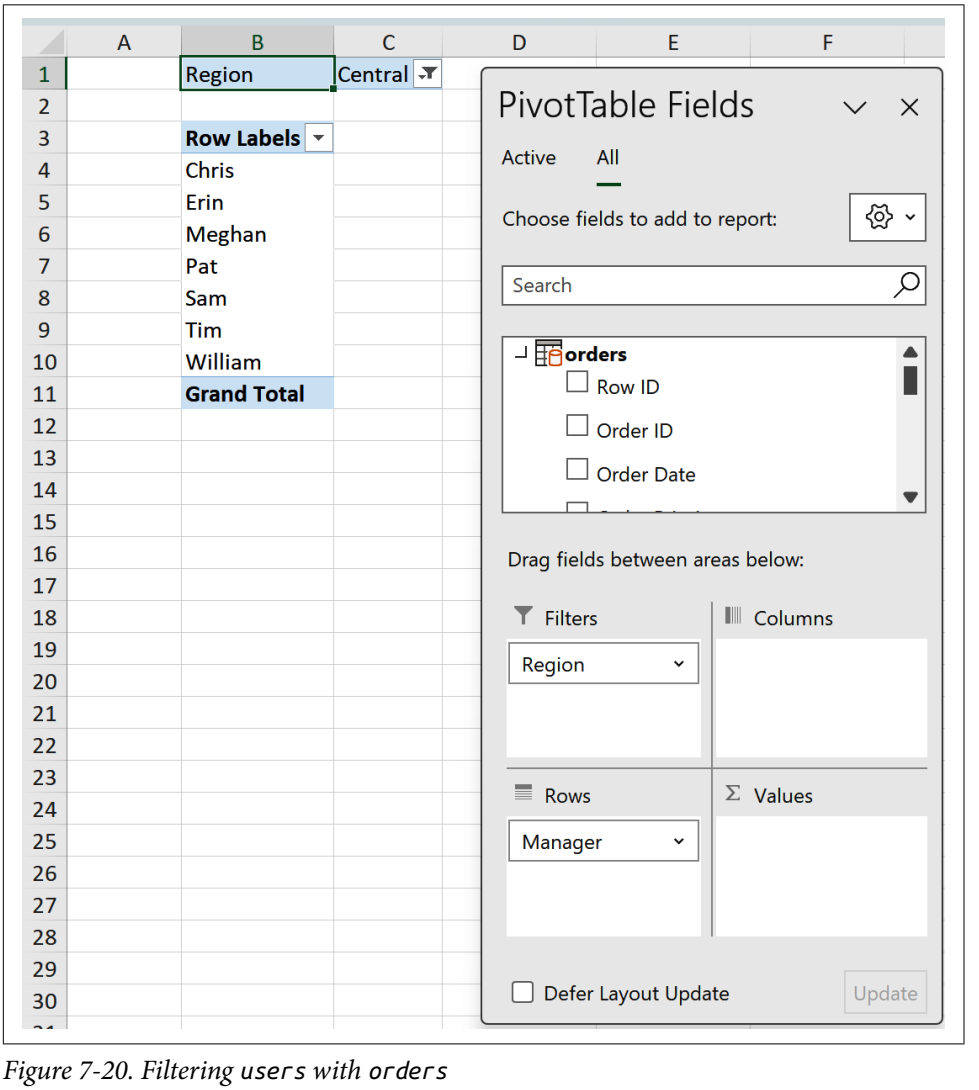


Figure 7-20. Filtering `users` with `orders`

This brings up the question: Why doesn't the filter applied to the `orders` table affect the `users` table? Shouldn't Chris be the only manager returned in this PivotTable,

given that he is the only manager in the Central region? The answer lies in the principle of filter direction.

Filter Direction and Cardinality

In Power Pivot, filter direction depends on the type of relationship. In a one-to-many relationship, filters move from the "one" side to the "many" side. For example, the `users` table can affect the `orders` table, but not vice versa. This approach enhances performance because filtering from the side with fewer records to the side with more records is more efficient.

> **Changing Filter Direction in Power Pivot**
>
> In Power Pivot, you cannot directly change the filter direction, as it is determined by the cardinality of the table relationship. However, if you have a specific requirement to indirectly alter the filter direction, you can explore the `CROSSFILTER()` function in DAX, which is beyond the scope of this book. You can learn more about it on Microsoft Learn.

From Design to Practice in Power Pivot

Jazz guitarist Irving Ashby once likened rhythm guitar to vanilla in a cake: "You can't taste it, but you know when it's been left out." Filter direction in your Data Model mirrors this sentiment. Typically, it works quietly in the background, but when something's amiss, its absence becomes glaringly evident.

With a foundational understanding of Data Model aspects like filter direction and cardinality, we can now delve into advanced features. Exploring calculated columns and hierarchies will further refine the Data Model, adding layers of flexibility and enhanced functionality.

Creating Columns in Power Pivot

In Chapter 4, you were introduced to creating calculated columns in Excel Power Query. Now, let's delve into when and how to perform the same task in Power Pivot, while also considering the advantages and disadvantages of each approach.

Calculating in Power Query Versus Power Pivot

Power Query and Power Pivot are distinct tools with complementary roles, and both can derive calculated columns. To determine which to use, consider the following:

- Use Power Query for data cleaning and transformation during the prep stage. It's ideal for one-time tasks like merging fields or changing data types, optimizing your model by simplifying data before loading it into Power Pivot.
- Use Power Pivot for advanced analyses, like dynamic calculations or building relationships between tables. These operations are done after data loading, enhancing reports and dashboards. However, excessive use may increase file size and slow performance.

By adhering to these guidelines, you can maximize the capabilities of both Power Query and Power Pivot, ensuring optimal calculated column creation relative to your data's state and processing requirements.

While these rules of thumb are helpful, the best way to decide whether to create calculated columns in Power Query or Power Pivot is to experiment with both tools and see which one works best for your needs.

Example: Calculating Profit Margin

Return to the Power Pivot editor. In Data View, select the `orders` table.

Create a calculated column called `Profit margin`. Scroll to the right until you reach the end of the table where it says "Add Column." Click inside Add Column to name the column `Profit margin`, then add the profit margin formula (`=orders[Profit]/orders[Sales]`) as shown in Figure 7-21.

[Profit mar... fx =orders[Profit] / orders[Sales]

	tegory	Product Sub-Category	Product Name	Supplier	Product Container	Marketing Channel	Product Base Margin	Ship Date	Profit margin	Add Column
1	plies	Paper	Xerox 1905	Xerox	Small Box	Email Marketing	0.37	8/23/2021 ...	-0.845751746...	
2	plies	Paper	Xerox 1997	Xerox	Small Box	Email Marketing	0.37	6/13/2022 ...	-0.898311227...	
3	plies	Paper	Xerox 21	Xerox	Small Box	Email Marketing	0.37	7/21/2020 ...	-0.405810518...	
4	plies	Paper	Xerox 1995	Xerox	Small Box	Email Marketing	0.37	10/1/2023 ...	-0.303716814...	
5	plies	Paper	Xerox 214	Xerox	Small Box	Email Marketing	0.37	11/15/202...	-0.548446518...	
6	plies	Paper	Xerox 1894	Xerox	Small Box	Email Marketing	0.37	5/12/2023 ...	-0.408720311...	
7	plies	Paper	Xerox 1994	Xerox	Small Box	Email Marketing	0.37	10/11/202...	-0.243463770...	
8	plies	Paper	Xerox 227	Xerox	Small Box	Email Marketing	0.37	12/21/202...	-0.727707636...	
9	plies	Paper	Xerox 2	Xerox	Small Box	Email Marketing	0.37	12/11/202...	-0.347007586...	
10	plies	Paper	Xerox 216	Xerox	Small Box	Email Marketing	0.37	8/15/2020 ...	-0.624150875...	
11	plies	Paper	Xerox 210	Xerox	Small Box	Email Marketing	0.37	11/28/202...	-0.536013400...	
12	plies	Paper	Xerox 220	Xerox	Small Box	Email Marketing	0.37	6/2/2022 1...	-0.553918747...	
13	plies	Paper	Xerox 227	Xerox	Small Box	Email Marketing	0.37	9/16/2023 ...	-0.728280552...	
14	plies	Paper	Xerox 224	Xerox	Small Box	Email Marketing	0.37	9/16/2021 ...	-0.716479200...	
15	plies	Paper	Xerox 213	Xerox	Small Box	Email Marketing	0.37	8/11/2023 ...	-0.660714285...	
16	plies	Paper	Xerox 207	Xerox	Small Box	Email Marketing	0.37	9/15/2023 ...	-0.407959356...	
17	plies	Paper	Xerox 226	Xerox	Small Box	Email Marketing	0.37	11/24/202...	-0.338816940...	
18	plies	Paper	Xerox 226	Xerox	Small Box	Email Marketing	0.37	3/7/2022 1...	-0.329385001...	
19	plies	Paper	Xerox 210	Xerox	Small Box	Email Marketing	0.37	9/10/2023 ...	-0.501557273...	
20	plies	Paper	Xerox 212	Xerox	Small Box	Email Marketing	0.37	11/25/202...	-0.749319812...	
21	plies	Paper	Xerox 1905	Xerox	Small Box	Email Marketing	0.37	5/25/2023 ...	-0.582962492...	

Figure 7-21. Creating a profit margin calculated column

Your calculated column should read like this:

```
orders[Profit] / orders[Sales]
```

Note that, in contrast to Excel tables, you must manually type in references to other columns within the table instead of selecting them with a mouse click or keyboard stroke.

This completes your first time using the DAX programming language for managing your Data Model in Power Pivot. You'll observe that the way individual columns are referenced bears a striking resemblance to the structured column references of basic Excel tables. You can format the new column as a percentage while still in Data View.

To validate this calculation, load the Data Model into a new PivotTable. Drag `Customer Segment` to the Rows section and `Average of Profit Margin` to the Values section. To cross-check accuracy, add `Sum of Profit` and `Sum of Sales` to the Values section as well.

Upon manually calculating the profit margin as shown in Figure 7-22, you may spot a discrepancy compared to the PivotTable values.

F4 =C4 / D4

	A	B	C	D	E	F
1						
2						
3		**Row Labels**	**Sum of Profit**	**Sum of Sales**	**Average of Profit margin**	Profit margin cross-check
4		Consumer	$279,502.16	$3,018,373.67	-12.05%	9.26%
5		Corporate	$594,847.82	$5,409,916.05	-14.50%	11.00%
6		Home Office	$299,057.96	$3,467,291.80	-12.04%	8.63%
7		Small Business	$312,636.58	$2,763,480.67	-13.77%	11.31%
8		**Grand Total**	**$1,486,044.52**	**$14,659,062.20**	**-13.28%**	10.14%
9						
10						

Figure 7-22. Checking the profit margin calculation

The issue arises because the `Profit margin` calculated column calculates a basic average of individual profit margins, without factoring in the aggregation of total profits and total sales. For an accurate profit margin calculation, dynamic and on-the-fly computations are needed which can't be accomplished solely through calculated columns. Instead, you'll need to use DAX measures, which will be extensively covered in Chapters 8 and 9.

For now, it is important to remember that calculated columns in Power Pivot should not be used when there is a possibility of aggregating the results. This issue is similar to working with calculated columns in Power Query, where they can become distorted when aggregated.

There are scenarios where calculated columns in the Data Model are indeed the appropriate choice. One such example is the use of `SWITCH()`, which the following section explores.

Recoding Column Values with SWITCH()

The `SWITCH()` function proves highly valuable for applying conditional logic to reassign values. Given that each row is evaluated independently and results generally aren't aggregated, it's more appropriate to save the `SWITCH()` outcomes as calculated columns, rather than measures.

To demonstrate, imagine you want to assign the numbers 1, 2, 3, and 4 to the segments `Consumer`, `Corporate`, `Home Office`, and `Small Business`, respectively. In cases where a match isn't found, you'd prefer to recode the value as `Unknown`. Begin by adding a new calculated column named `Segment number` to the `orders` table in Power Pivot, as depicted in Figure 7-23.

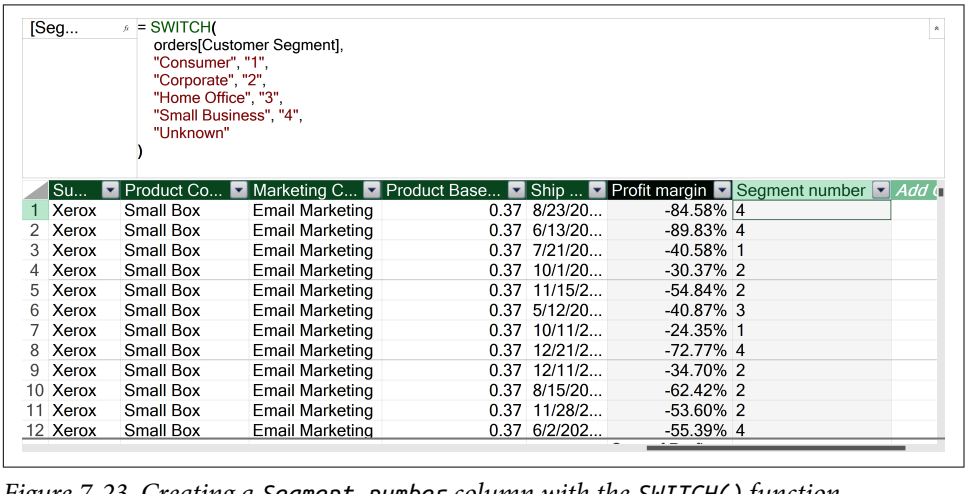


Figure 7-23. Creating a `Segment number` column with the `SWITCH()` function

Keep in mind that all values within a column of a Data Model table must share the same data type. Because `Segment number` includes the string `Unknown`, it's essential to transform the other values (1, 2, 3, 4) into strings to maintain consistency.

Load your updated Data Model to a new PivotTable, or refresh an existing one, to use this new column in your analysis. For instance, Figure 7-24 summarizes sales by the recoded segment numbers rather than the original category.

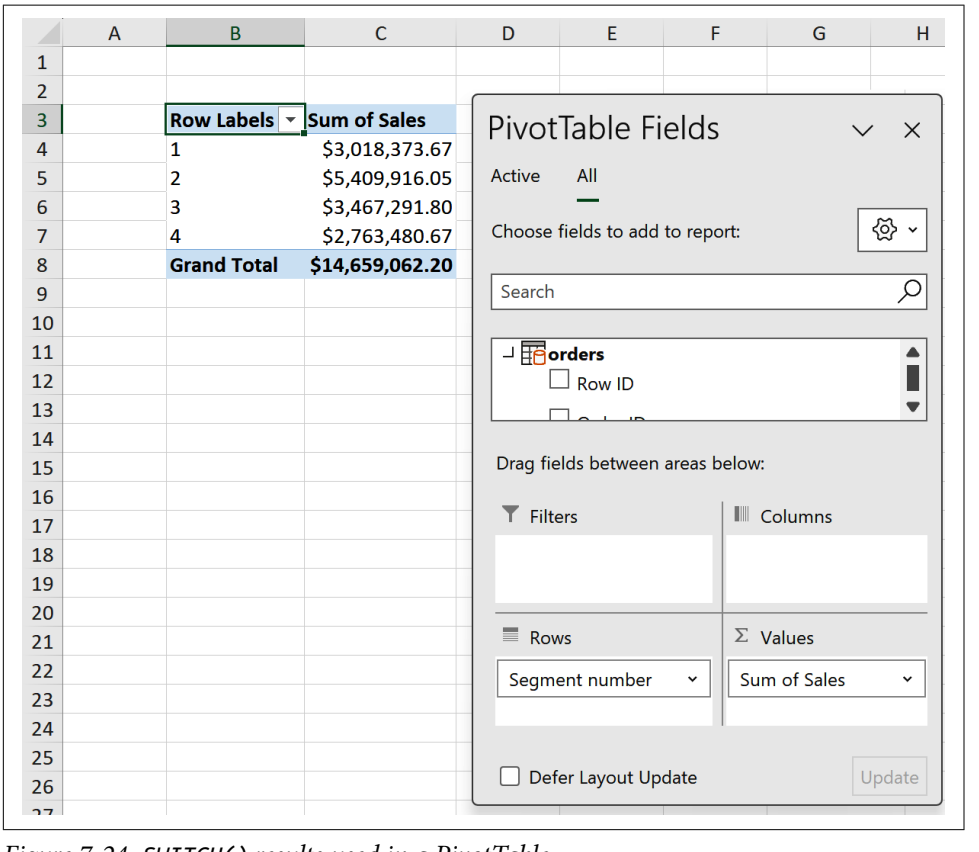


Figure 7-24. `SWITCH()` results used in a PivotTable

SWITCH() Versus Conditional Columns

The `SWITCH()` function available in Power Pivot offers a more efficient and comprehensible method for handling numerous comparisons within a single formula than does conditional columns. While conditional columns in Power Query fulfill a similar role, the `SWITCH()` function simplifies intricate decision making by evaluating an examined expression. It then returns the relevant outcome based on the first matching value. This approach eradicates the necessity for constructing and nesting multiple `IF()` statements, which can become unwieldy and less legible, especially with larger datasets and more intricate conditions.

Additionally, the `SWITCH()` function is optimized for peak performance within the Power Pivot engine. This translates to quicker processing times compared to evaluating nested `IF()` statements. Although performance considerations can vary according to the specific scenario, the `SWITCH()` function generally presents a streamlined and efficient solution for managing numerous comparisons in Power Pivot calculations.

Creating and Managing Hierarchies

Hierarchies are pivotal in numerous facets of our lives. Take my location while penning this book as an example: Cleveland, Ohio, United States. This can be organized into a hierarchical structure, starting with the broadest category (country: United States), followed by a more specific category (state: Ohio), and culminating in the most specific location (city: Cleveland). Integrating such hierarchical structures into the Data Model simplifies the process of data analysis and exploration, enabling a more efficient examination at varying levels of detail.

Creating a Hierarchy in Power Pivot

Let's create a product-based hierarchy in our Data Model consisting of `Product Category`, `Product Sub-Category` and `Product Name`. To do this, navigate to Diagram View. Select the dimensions in the desired hierarchical order (i.e., `Product Category` at the top) while holding the Ctrl key. Once selected, right-click and choose Create Hierarchy. Assign a name for the hierarchy, such as `Product Hierarchy`, as shown in Figure 7-25.

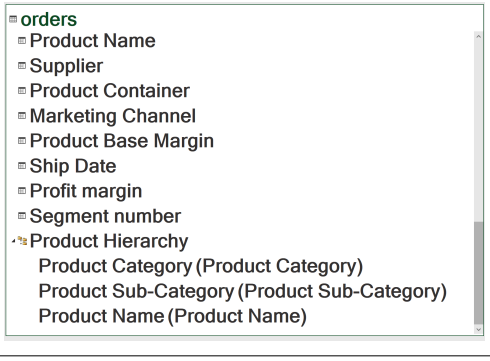


Figure 7-25. Hierarchies seen in Diagram View

In Diagram View, you can easily add, modify, or delete hierarchies within the Data Model as needed. For now, load the data into a PivotTable to see it in action.

Using Hierarchies in the PivotTable

After closing Power Pivot, return to your PivotTable. Place `Product Hierarchy` in the Rows section and `Sum of Sales` in the Values section. You will observe that the three dimensions within the hierarchy cannot be used individually in the PivotTable; they can only be used as part of the hierarchy.

Now, you can click on the small plus sign to the left of any product category to drill down into that category at the subcategory level and, eventually, to the individual product name level, as shown in Figure 7-26.

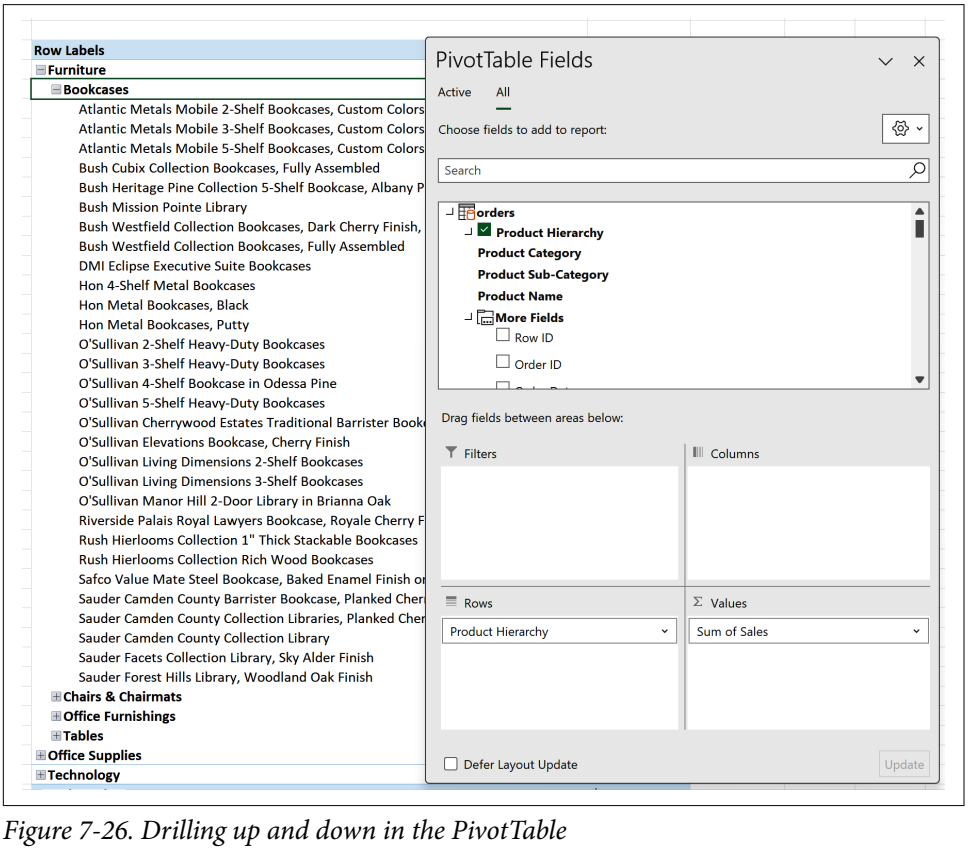


Figure 7-26. Drilling up and down in the PivotTable

Toggle the buttons back to a minus sign to navigate upwards in the hierarchy. In the Active Field group on the PivotTable Analyze tab, you'll find additional features that simplify hierarchy manipulation, such as the option to expand or collapse the entire hierarchy simultaneously.

Before incorporating hierarchies into your data, it is crucial to consider the impact of data quality inconsistencies. In cases where a single subcategory is inconsistently mapped to multiple categories, the hierarchy may lose its meaning for analysis. It is also important to note that less experienced Excel users may face initial challenges when working with hierarchies.

Loading the Data Model to Power BI

Up to this point, you have mastered the basic elements of creating Data Models, including valuable features such as calculated columns and hierarchies. In Chapters 8 and 9, we'll explore the creation of DAX measures and the use of tools like KPIs to improve data analysis and reporting. Before we move forward, let's briefly explore an alternative method for analyzing and visualizing your Data Model: Power BI. We will examine how Power BI operates and the benefits it provides.

Power BI as the Third Piece of "Modern Excel"

So far this book has primarily focused on Power Query and Power Pivot for data cleaning and data analysis, respectively. A third component of this stack, used for data visualization, was Power View, which was ultimately discontinued. Initially developed for Excel, Power View enabled the creation of interactive dashboards and reports. However, as time went on, the concept of Power View was integrated into Power BI, and newer versions of Excel no longer include it or include very limited functionality.

Microsoft's decision to shift focus from Power View in Excel to Power BI was driven by several factors. Power BI offers advanced data visualization capabilities that enable users to build interactive dashboards and reports using a wide range of data sources. This transition also aligns with Microsoft's cloud-oriented strategy, as Power BI operates largely as a cloud-based platform that enables collaboration and data accessibility from any location. By emphasizing Power BI, Microsoft provides users with a more comprehensive, modern, and integrated solution for business intelligence to meet evolving needs.

While Power BI is renowned for its ability to generate interactive dashboards, some analysts may find it less familiar, which can introduce challenges when they are building and sharing their work. Starting with Excel to build a Data Model remains a practical choice due to its widespread familiarity among professionals. As projects grow in complexity, and there's a demand for more advanced dashboards, transitioning from Excel to Power BI becomes a viable path. This section will explore the strategies for making this transition smoothly.

Importing the Data Model to Power BI

As this is not a book on Power BI, the main objective here is simply to load the Data Model into Power BI and preview it. To accomplish this, ensure that you have Power BI Desktop, a free application, installed. You can find instructions on how to install it in Microsoft's official documentation. If you wish to explore Power BI in more depth, check out Jeremey Arnold's book, *Learning Microsoft Power BI: Transforming Data into Insights* (O'Reilly, 2022).

To see how easy it is to transfer your Power Pivot work into Power BI, practice loading the *ch_07_solutions.xlsx* file or try loading the workbook you've been working on throughout this chapter.

With this workbook closed in Excel, open Power BI Desktop and create a new report. From the Power BI Desktop ribbon, head to File → Import → Power Query, Power Pivot, Power View, as shown in Figure 7-27.

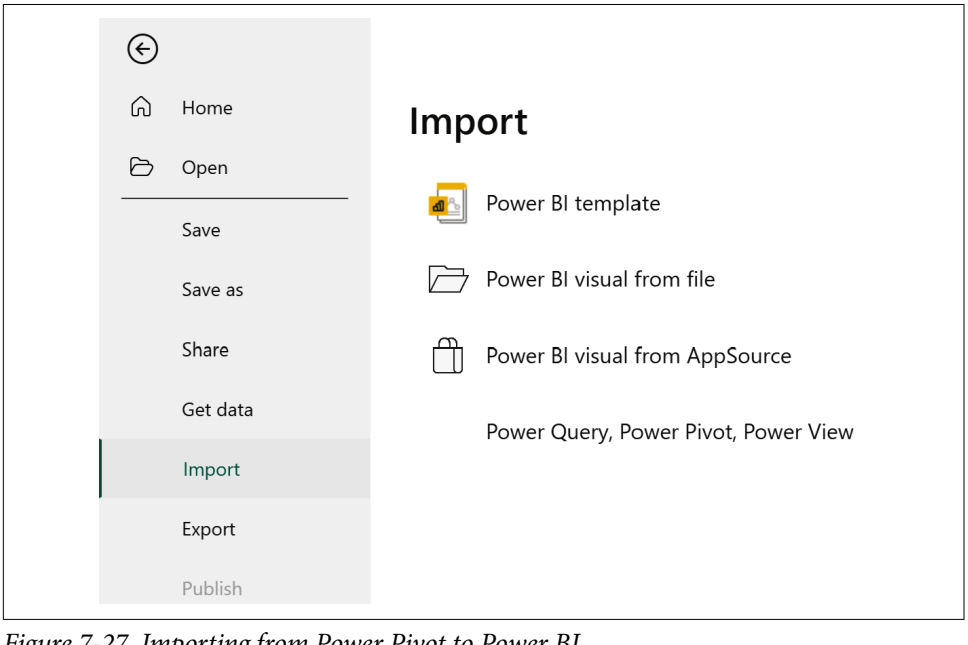


Figure 7-27. Importing from Power Pivot to Power BI

From here, browse to *ch_07_solutions.xlsx*, and select it. You might encounter a warning such as in Figure 7-28, indicating that Power BI will make its best effort to import the data. Click on the Start button to proceed with the import.

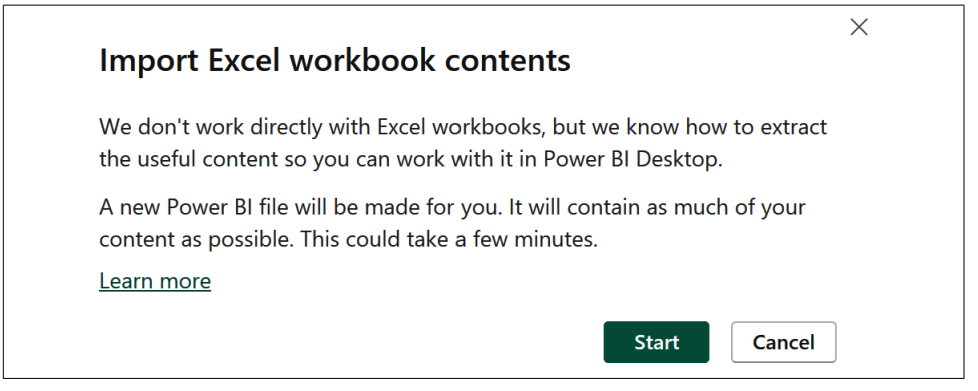


Figure 7-28. Import Excel workbooks warning

You can choose between copying the Excel data or keeping a live connection. Linking to the Excel workbook offers convenience for making changes to the data, but it comes with slower performance in Power BI. To keep things simple, I will make a copy of the data instead of maintaining the connection.

You should now receive a message confirming that Power BI has successfully imported your workbook, including its queries, Data Model relationships, and any KPIs or measures that were created. You might receive a message indicating that, due to the large size of one of the objects in the import, it was too extensive to be copied, and therefore, a live connection was used instead.

Viewing the Data in Power BI

To verify the Data Model has been imported correctly into Power BI, navigate to Model View by clicking the third small icon on the left side of the screen. In Model View, which is similar to Diagram View in Power Pivot, we can confirm that the relations between tables were properly defined, as seen in Figure 7-29.

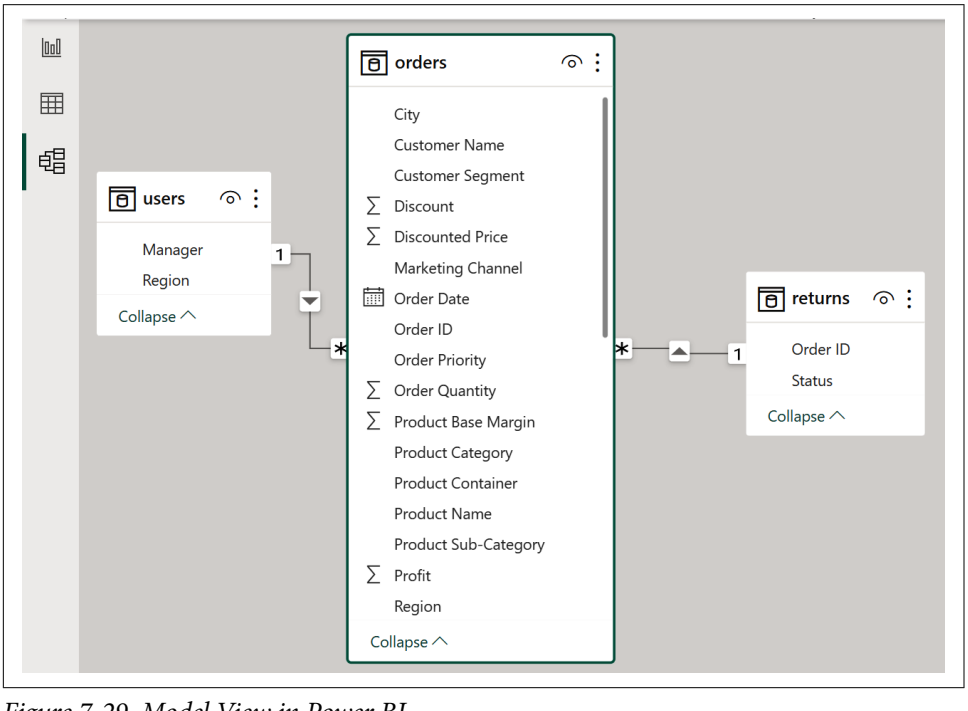


Figure 7-29. Model View in Power BI

Keep scrolling down the `orders` table and you will also see that the hierarchy and calculated columns you created in Power Pivot have been ported to Power BI.

You can view these calculated columns through the Table View, which is accessible by clicking the small spreadsheet icon above the Model View icon. This feature is akin to Power Pivot's Data View, enabling you to toggle between data sources. The calculated columns for `Profit Margin` and `Segment Number`, along with their formulas, have been successfully imported, as shown in Figure 7-30.

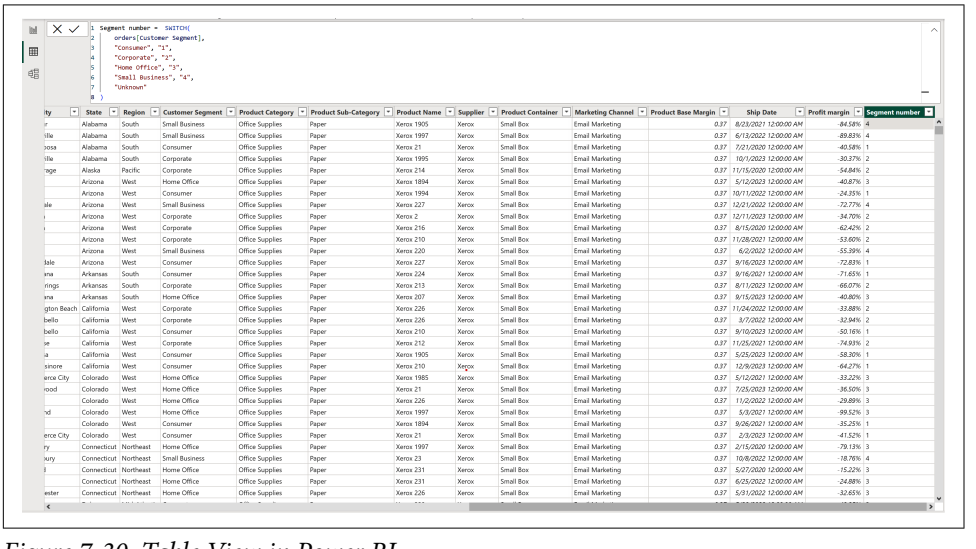


Figure 7-30. Table View in Power BI

The formula editor in Power BI is noticeably more advanced compared to the one in Power Pivot. This reflects the broader capabilities of Power BI, which offers numerous opportunities for creating advanced dashboards and reports that would be challenging to build in Excel alone.

Power BI has become Microsoft's modern platform for developing dashboards and reports. However, Excel still retains its legacy as a quick and user-friendly tool, allowing for more flexible and exploratory data modeling and analysis. In the end, Power BI and Excel complement each other and serve different purposes as part of the same team.

Your Power BI report can now be saved. This Power BI file has been saved on your behalf in the *ch_07* folder, titled *ch_07_solutions.pbix*.

Conclusion

This chapter provided a practical, hands-on approach to constructing basic Data Models and exploring essential features in Power Pivot. Upcoming chapters in Part II will delve deeper, exploring its data analysis and reporting capabilities.

Exercises

For this exercise, open *ch_07_exercises.xlsx* found in the *exercises\ch_07_exercises* folder in the book's companion repository. This workbook consists of three tables: `batting`, `people`, and `hof`. Perform the following:

1. Load the tables into Power Pivot via Power Query and establish relationships in the Power Pivot Data Model.
2. Identify fact and dimension tables in the Data Model and organize the model in Diagram View accordingly.
3. What is the cardinality of the relationships between these tables?
4. Use the `SWITCH()` function to generate an `is_player` column in the `hof` table. Assign Yes if the `category` column indicates Player, otherwise assign No.
5. Create a hierarchy among the `birthCountry`, `birthState`, and `birthCity` fields in the `people` table.
6. Load the Data Model results into an Excel PivotTable. Count the number of players. You can do this by totaling the number of `playerID`s that have Yes in the `is_player` column.

You can find the solutions in *ch_07_exercise_solutions.xlsx* in the same folder.

CHAPTER 8

Creating Measures and KPIs in Power Pivot

In Chapter 7, the fundamentals of Power Pivot and the Data Model were introduced, including relationships, hierarchies, and calculated columns. With the Data Model in place, this chapter delves into creating DAX measures and KPIs to aid end users in data interpretation.

For demonstrations, refer to *ch_08.xlsx* in the *ch_08* folder of the book's companion repository. This chapter uses the same retail sales dataset from Chapter 7, with the Data Model predefined in the provided exercise file.

Creating DAX Measures

In Chapter 7, the attempt to add a `profit margin` column to the `orders` table led to an unsatisfactory result. For aggregating and recalculating results across different categories and time periods, DAX measures are necessary. In Power Pivot, measures can be created in two ways: implicitly and explicitly. To gain hands-on experience with these methods, proceed by inserting a PivotTable from the Data Model.

Creating Implicit Measures

To aggregate data, like finding the total order quantity by region, one would typically drag the fields directly into the PivotTable, as shown in Figure 8-1.

To adjust the aggregation to determine the *average* number of units sold by region, navigate to the dropdown on "Sum of Order Quantity" in the PivotTable. Then, proceed to "Value Field Settings" and, within the "Summarize value field by" section, switch from Sum to Average.

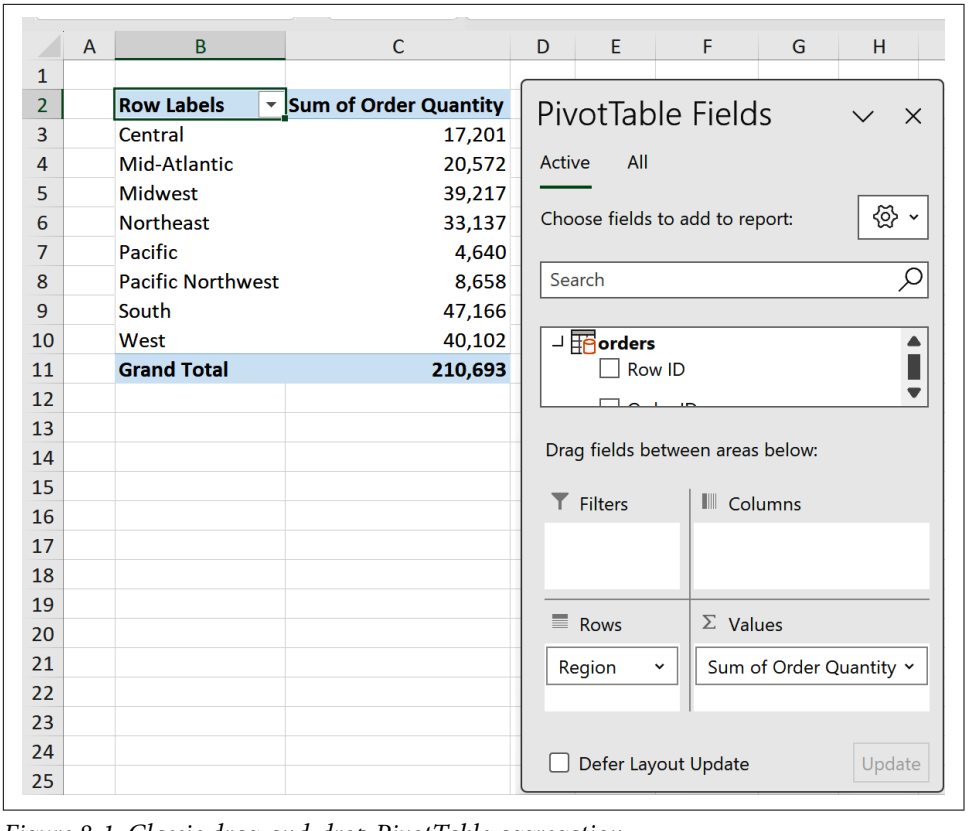


Figure 8-1. Classic drag-and-drop PivotTable aggregation

To see how the Data Model manages these PivotTable calculations, navigate to the Power Pivot tab on the ribbon and select Manage. Choose Diagram View from the View group on the Home tab. Then, on the Advanced tab, activate Show Implicit Measures. This will add two measures to the bottom of the `orders` table, as illustrated in Figure 8-2.

The measures created earlier via the PivotTable are termed *implicit measures*. Power Pivot automatically generates and stores these measures. They offer convenience for swift data exploration and analysis without the need to develop intricate calculations.

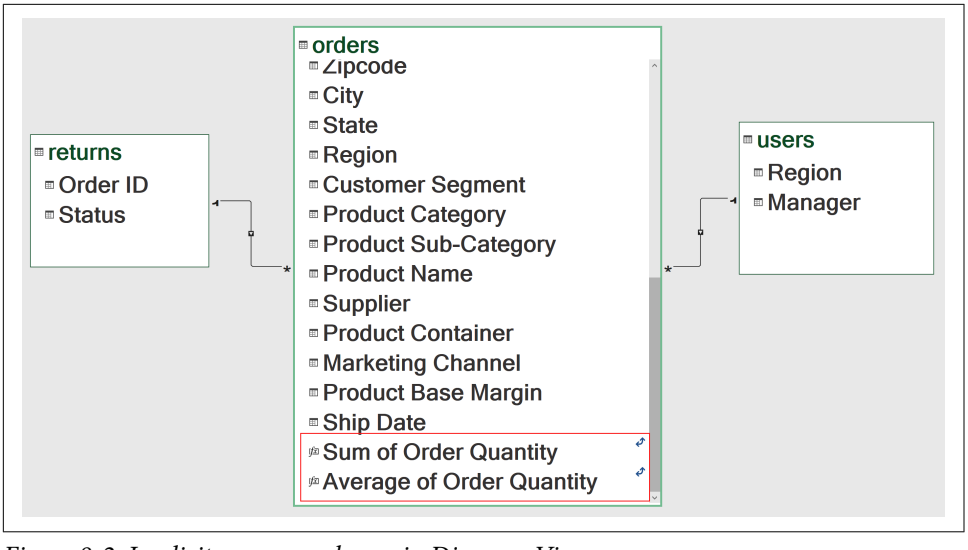


Figure 8-2. Implicit measures shown in Diagram View

Yet, implicit measures present challenges in customization and reusability throughout the Data Model. Simply aggregating an existing field won't suffice to establish a new derived measure, like average sales per unit sold. Creating a standalone, explicit measure combining two fields (sales and order quantity) becomes essential. Furthermore, the concealed nature of implicit measures complicates their management and organization. To address these issues, you can remove the implicit measures by following these steps: click on one of the measures, press and hold down the Ctrl key, click on the other measure, then right-click and select the Delete button, as shown in Figure 8-3.

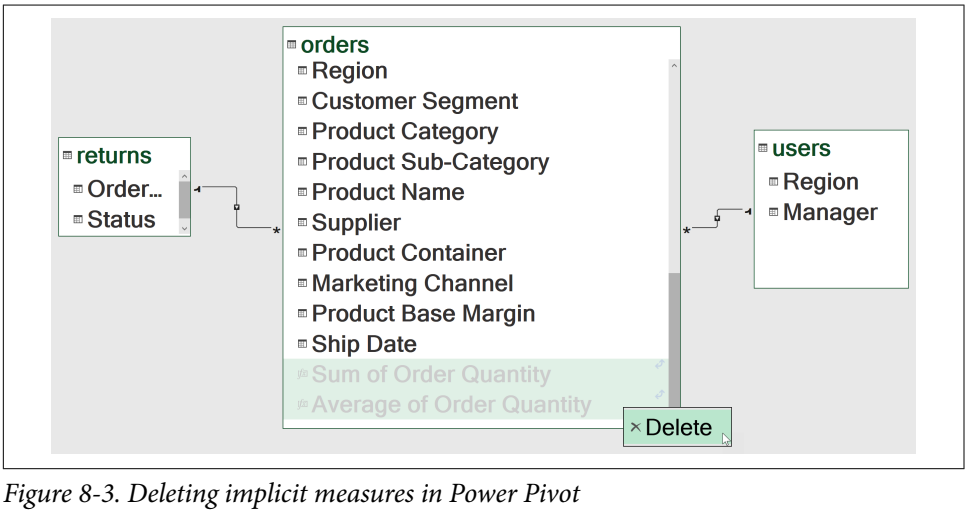


Figure 8-3. Deleting implicit measures in Power Pivot

Creating Explicit Measures

Instead of creating a DAX measure implicitly through PivotTable values, one can explicitly craft it using Power Pivot's Measures option. Exit the Power Pivot editor, navigate to the Power Pivot tab on the ribbon, and select Measures → New Measure, as shown in Figure 8-4.

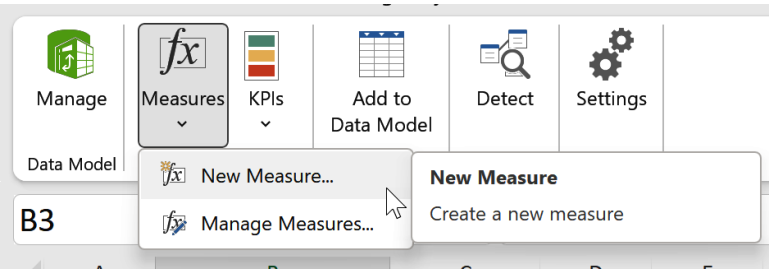


Figure 8-4. Creating a new measure in Power Pivot

Begin by establishing a `Total sales` measure that aggregates the `sales` column from the `orders` table. This DAX calculation will be similar to the structured table references discussed in Chapter 1. You can also continue to take advantage of Microsoft's IntelliSense for autocompleting the spelling of your functions, tables, and other elements used in building measures. Link this measure to the `orders` table by indicating the table name, and then format it as currency with 2 decimal places.

Conclude by selecting the "Check formula" button to validate the measure. Upon success, the message "No errors in formula" should appear, as shown in Figure 8-5.

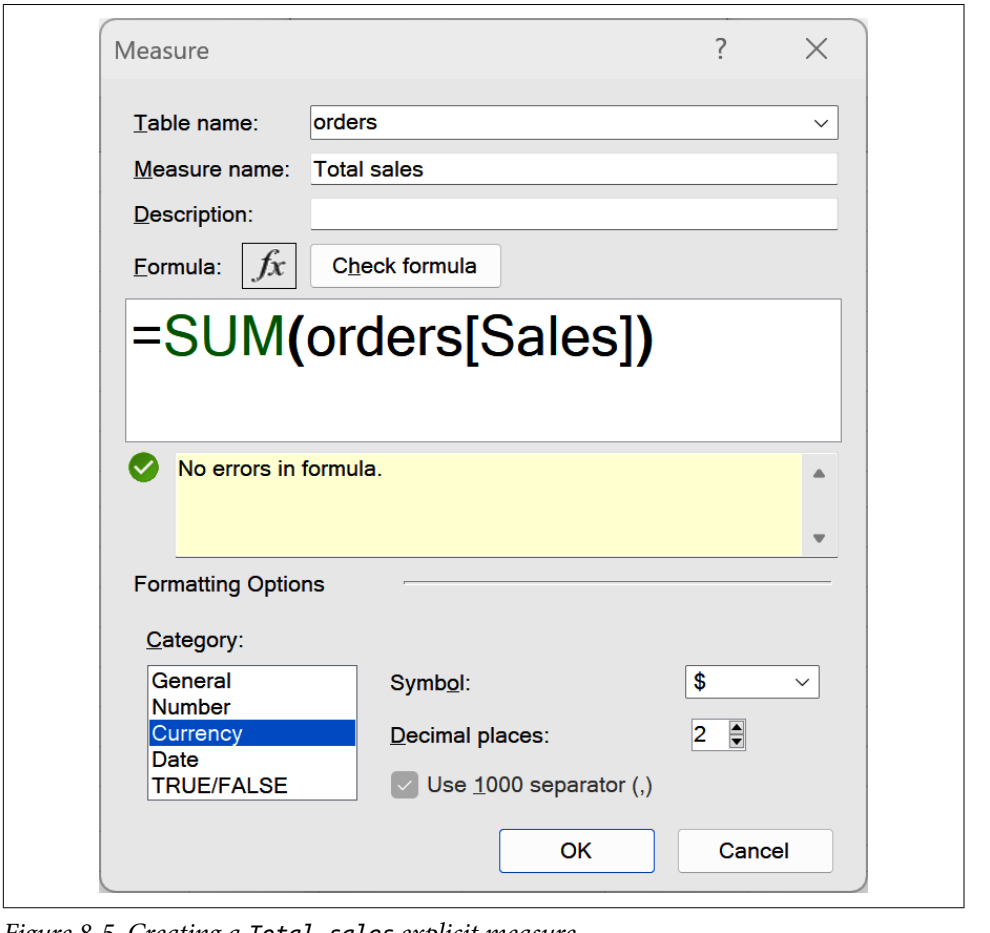


Figure 8-5. Creating a `Total sales` explicit measure

After clicking OK, the measure becomes available in the PivotTable fields under the `orders` table, distinguished by an *fx* symbol. Position `Region` in Rows and the newly created `Total sales` measure in Values, as illustrated in Figure 8-6.

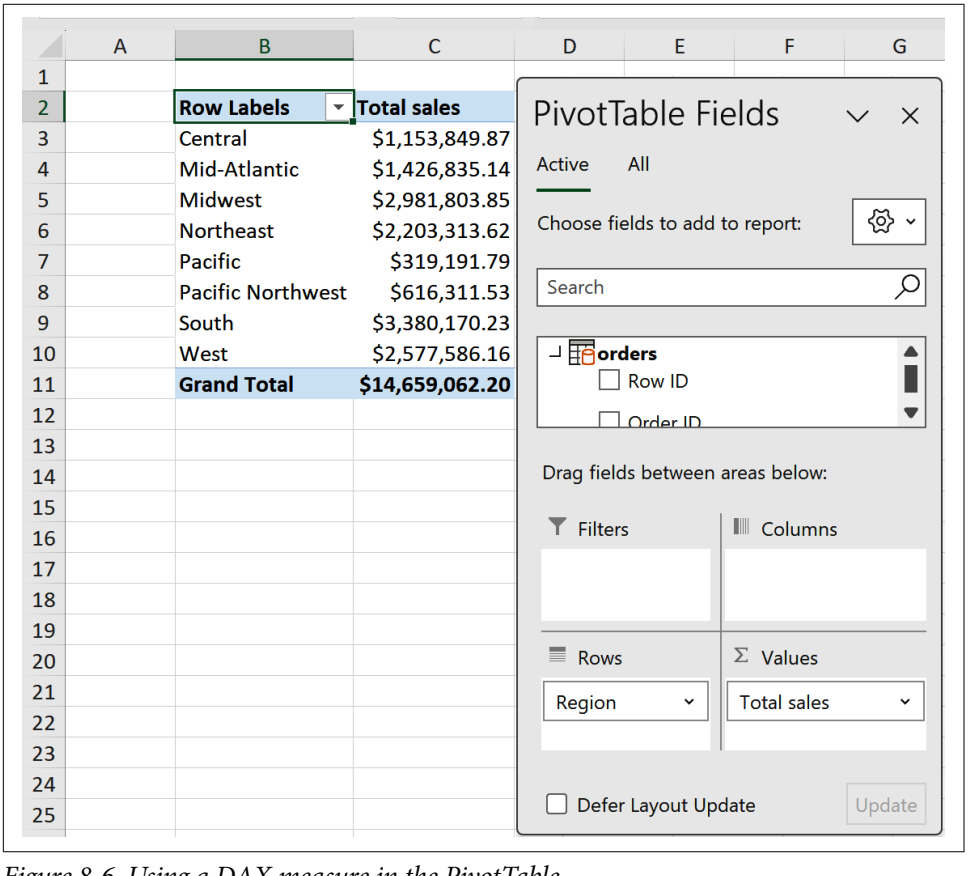


Figure 8-6. Using a DAX measure in the PivotTable

The explicit measure's aggregation type can't be changed, a limitation that does not apply to implicit measures. To modify how the measure is calculated, navigate back to the Power Pivot tab in the ribbon, select Measures → Manage Measures, and then click Edit on your `Total sales` measure.

Continue by formulating a measure named `Total profits`. Your measure should resemble Figure 8-7.

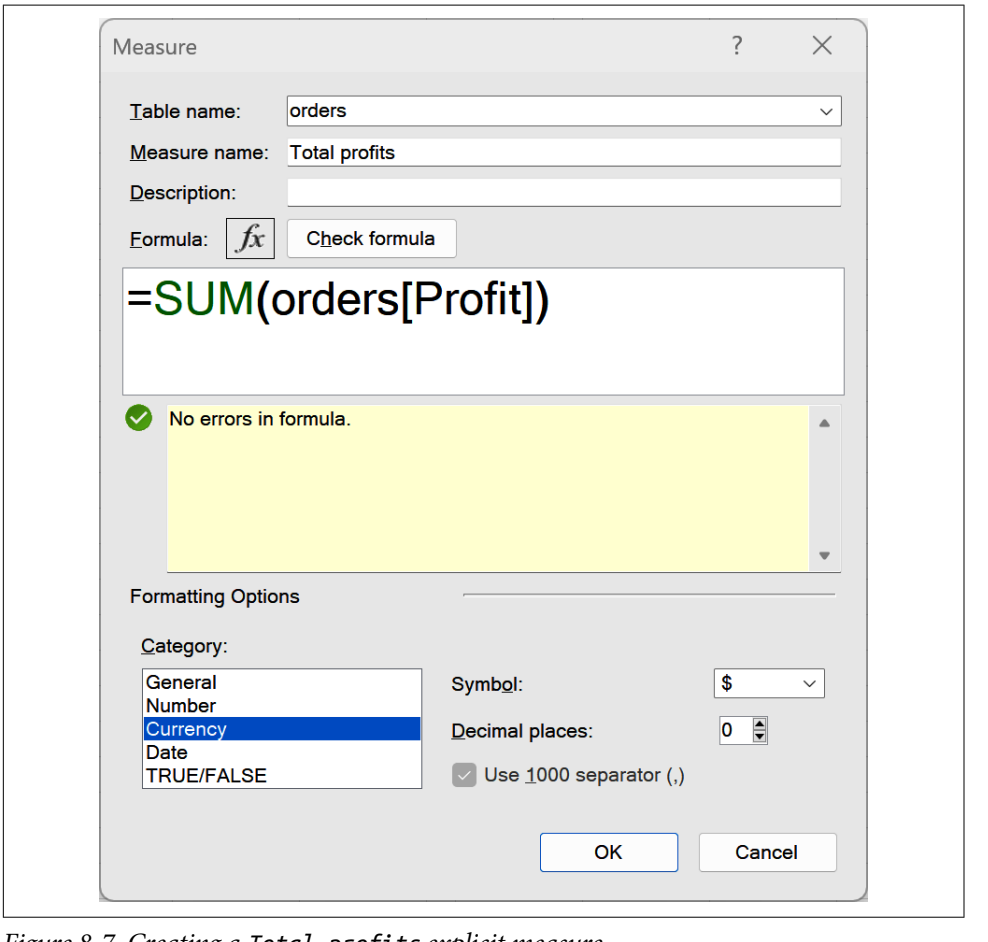


Figure 8-7. Creating a `Total profits` explicit measure

Calculated measures shine when used as inputs for other measures, facilitating advanced calculations that surpass the scope of implicit measures. For instance, the profit margin can be determined using the `Total profits` and `Total sales` measures, as depicted in Figure 8-8.

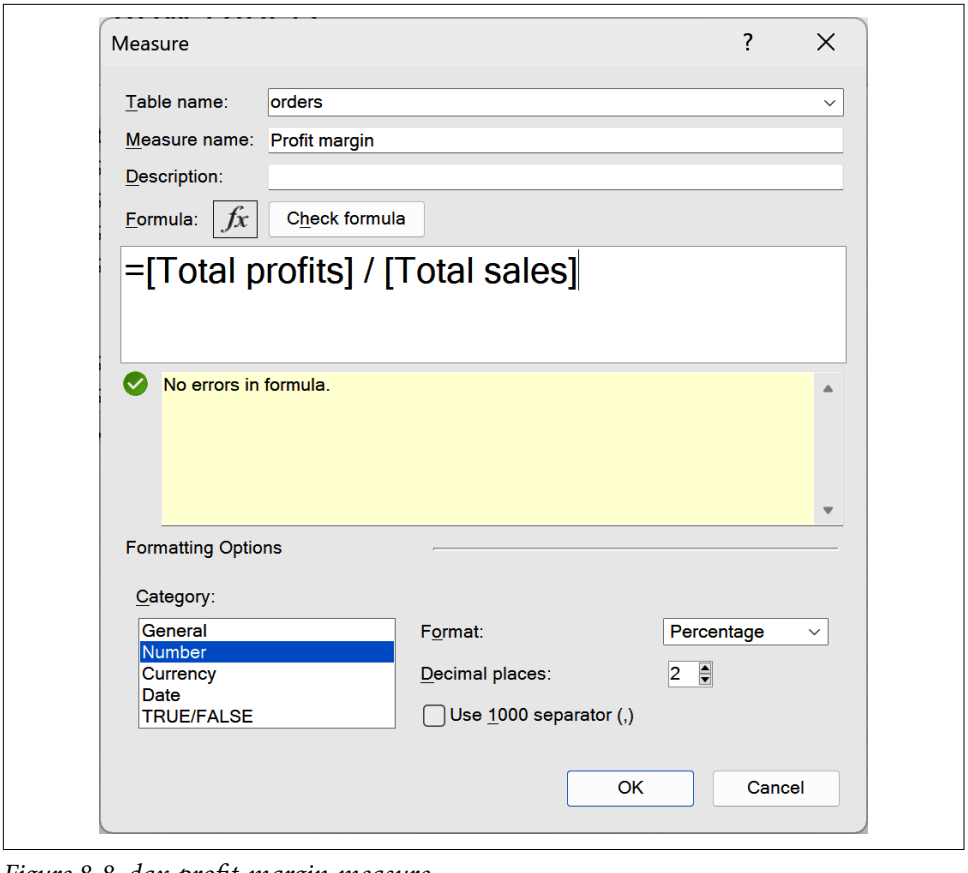


Figure 8-8. dax-profit-margin-measure

Add `Total sales`, `Total profits`, and `Profit margin` to the Values section of the PivotTable, and place `Region` in the Rows. Double-check that profit margin is calculating correctly using formulas. Unlike the approach in Chapter 7 using a calculated column, the calculations are now accurate, as demonstrated in Figure 8-9.

Implicit measures might be convenient, but explicit measures grant transparency, customization, and the capacity for advanced calculations. Investing the additional effort is beneficial, and it's recommended that you render all Power Pivot measures explicit, irrespective of their simplicity.

F3 =D3 / C3

	A	B	C	D	E	F
1						
2		**Row Labels**	**Total sales**	**Total profits**	**Profit margin**	Profit margin cross-check
3		Central	$1,153,849.87	$136,082.79	11.79%	11.79%
4		Mid-Atlantic	$1,426,835.14	$111,184.39	7.79%	7.79%
5		Midwest	$2,981,803.85	$302,000.86	10.13%	10.13%
6		Northeast	$2,203,313.62	$170,015.15	7.72%	7.72%
7		Pacific	$319,191.79	$50,208.54	15.73%	15.73%
8		Pacific Northwest	$616,311.53	$73,374.13	11.91%	11.91%
9		South	$3,380,170.23	$397,294.47	11.75%	11.75%
10		West	$2,577,586.16	$245,884.19	9.54%	9.54%
11		**Grand Total**	**$14,659,062.20**	**$1,486,044.52**	**10.14%**	10.14%

Figure 8-9. Double-checking the `Profit margin` measure

Table 8-1 compares implicit and explicit measures.

Table 8-1. Comparison of implicit versus explicit measures

Implicit measures	Explicit measures
Automatically generated by Power Pivot based on data fields	User-defined calculations
Quick and easy to create, requiring minimal effort	Require more time and technical expertise to create
Ideal for quick data exploration	Tailored to specific business needs
May not accurately capture desired metric or KPI	Accurate and specific
Less customizable and flexible	More customizable and flexible
Suitable for simple analysis	Suitable for complex analysis

DAX measures, created explicitly, allow for a broad spectrum of complex analyses. Indeed, Chapter 9 unveils methods that would be difficult or even impossible to achieve with Excel alone.

However, prior to exploring those advanced topics, it's crucial to secure some immediate victories that help users understand and leverage their data more effectively through Power Pivot. The core of data analytics is to ease data interpretation and decision making. Thus, this chapter concludes with a discussion on KPIs.

Creating KPIs

KPIs are essential for tracking business performance and achieving goals. In Excel Power Pivot, KPIs can provide valuable insights into your data analysis. The goal for this section is to create a KPI that compares overall sales to their sales target.

To do this, Power Pivot requires that both these figures be created as explicit measures. You have already created a `Total sales` measure; now use the same logic to create a `Total sales target` measure, as shown in Figure 8-10.

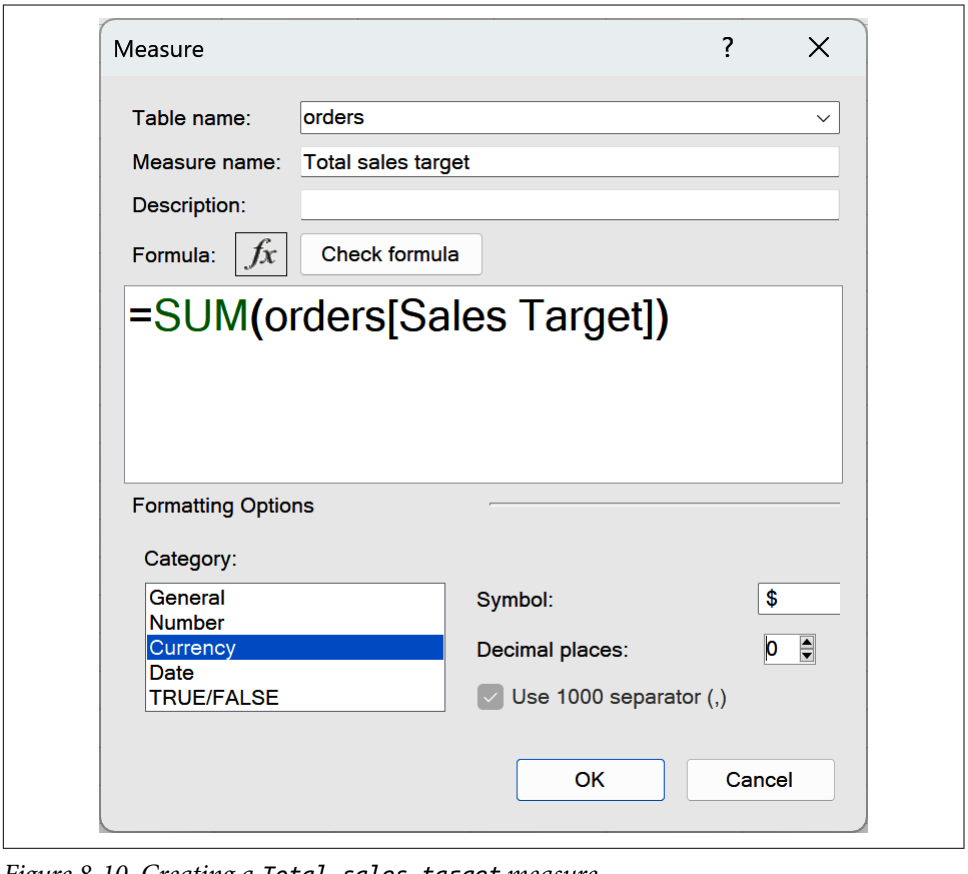


Figure 8-10. Creating a `Total sales target` measure

To get started building the KPI, head to Power Pivot on the ribbon, then select KPIs → New KPI. Set the KPI base field to `Total sales` and the target value to `Total sales target`, as shown in Figure 8-11.

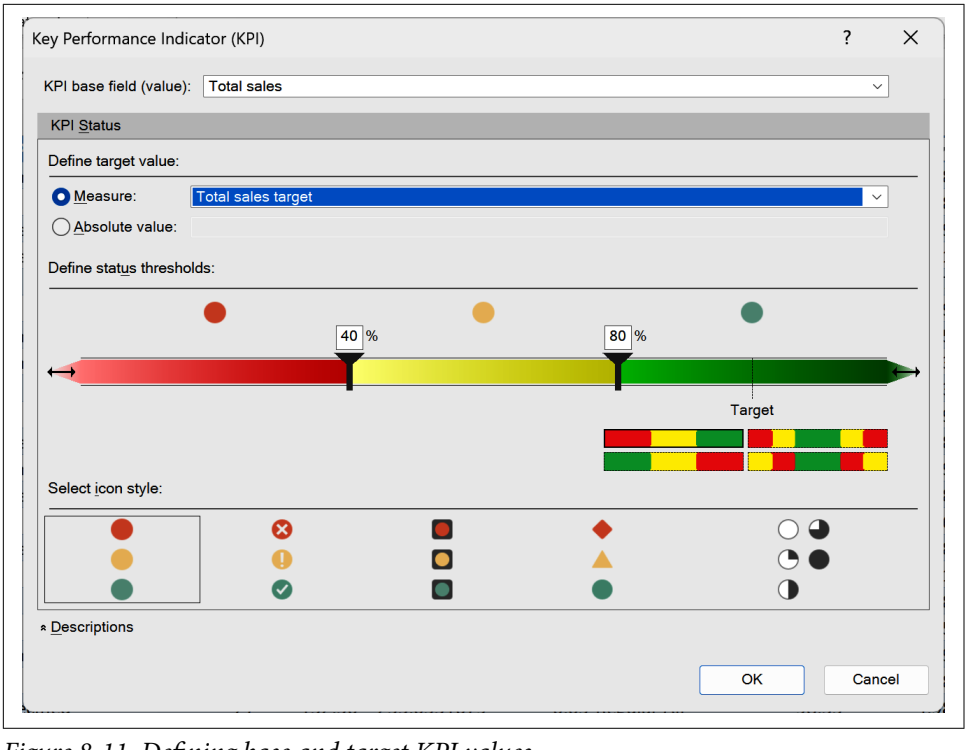


Figure 8-11. Defining base and target KPI values

This setup will compare actual sales to their target levels.

Next, set the status thresholds to provide context to the target value by defining acceptable ranges for performance. These thresholds categorize results into bands like "good," "satisfactory," or "poor," allowing users to quickly gauge performance relative to the target with more nuance than a simple binary hit-or-miss assessment.

Set a three-tier threshold to see which values are exceeding, meeting, and lagging expectations:

- A percent-to-sales target less than 90% will be marked in red.
- A percent-to-sales target percentage between 90% and 100% will be marked in yellow.
- A percent-to-sales target of 100% or greater will be marked in green.

Click and drag the thresholds in your KPI menu to match these rules, as shown in Figure 8-12.

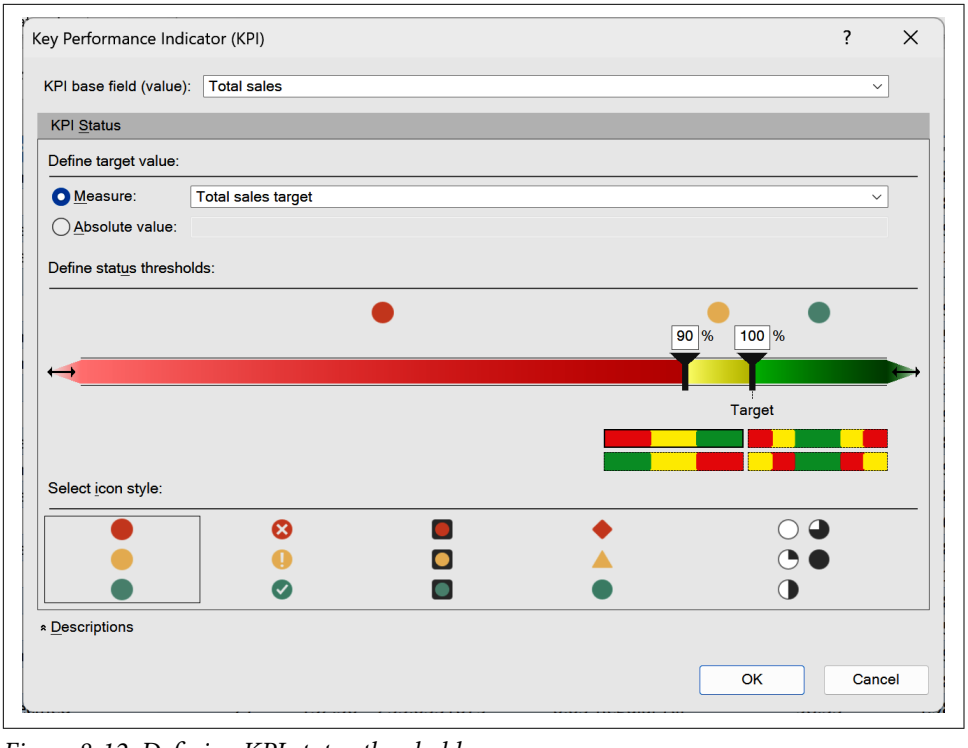


Figure 8-12. Defining KPI status thresholds

Adjusting Icon Styles

Next, you'll discover various options to customize the appearance and design of your KPIs, which could be beneficial in specific scenarios. However, it's important to note that using red, green, and yellow colors in data visualization is discouraged, as it can lead to confusion and misinterpretation for individuals with color vision differences.

Unfortunately, Power Pivot does not offer the capability to alter its color scheme, which is a notable shortcoming of the tool. This limitation may prompt users to consider migrating more complex dashboards and reports to more sophisticated BI platforms, such as Tableau or Power BI. These platforms provide a higher degree of flexibility in terms of color customization and other visualization features, catering to a more tailored and visually appealing presentation of data.

A red-yellow-green color palette is generally not recommended for data visualization as it can be challenging for individuals with color blindness to interpret the data accurately. Regrettably, Power Pivot does not allow for modifications to these colors, underscoring the necessity of considering more versatile solutions for comprehensive dashboards and reports.

Adding the KPI to the PivotTable

With your KPI now set up, go ahead and click OK. Next, insert a new PivotTable from the Data Model or use one already found in your workbook. Place `Region` in the Rows and `Customer Segment` in the Columns.

Toward the bottom of the `orders` group in the PivotTable field list, you should see a traffic light icon labeled `Total sales`. Select the dropdown and place the three fields inside into the PivotTable, as shown in Figure 8-13.

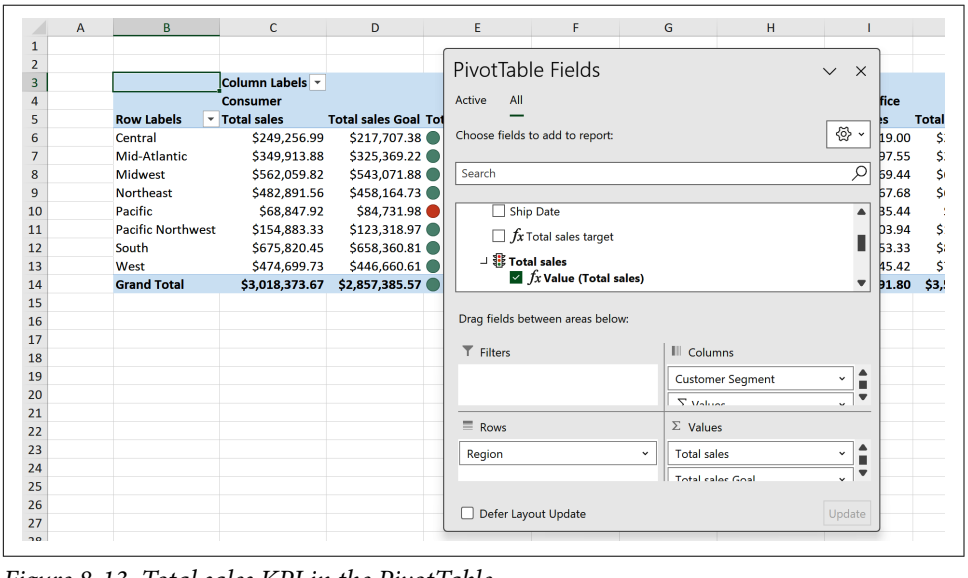


Figure 8-13. Total sales KPI in the PivotTable

If your `Total sales Goal` is not formatted properly, you can do so via Value Field Settings in the PivotTable.

The KPI is structured to function within the PivotTable as follows: it first displays the actual sales figure, followed by the set sales goal, which is based on the `Sales target` column. This display is enhanced by a visual indicator, providing an immediate understanding of whether the sales figures have met, surpassed, or fallen short of the target goal.

If you don't like the thresholds and want to fine tune them, you can always go back to Power Pivot on the ribbon, then select KPIs → Manage KPIs, click on your KPI, and click Edit.

KPIs and explicit measures-based PivotTables represent just the starting point for the extensive data reporting and visualization capabilities offered by Excel and Power Pivot. To gain a more thorough understanding of leveraging your Data Model's

results to construct complete dashboards enriched with additional functionalities such as slicers, conditional formatting, and more, consider exploring *Data Modeling with Microsoft Excel* by Bernard Obeng Boateng (Packt, 2023).

Conclusion

This chapter explored initial steps toward creating robust reports and analyses using Power Pivot, emphasizing the difference between implicit and explicit DAX measures. The chapter also introduced the concept of KPIs in Power Pivot, highlighting their usefulness in creating actionable reports, while noting some of their limitations.

Chapter 9 concludes Part II by delving into more advanced techniques and capabilities of DAX to create PivotTable-based analyses that would be otherwise difficult or impossible to build.

Exercises

For the exercises in this chapter, continue developing the Data Model you established in the exercises from Chapter 8. You have the option to proceed with your own workbook or begin anew with the *ch_08_exercises.xlsx* file, located in the *exercises \ch_08_exercises* folder in the the book's companion repository. Execute these tasks:

1. Create a PivotTable to present the total number of home runs (`HR`) by birth state (`birthState`) using an implicit measure.
2. Remove the implicit measure created in step 1 and establish a new explicit measure named `hr_total` that computes the sum of home runs, formatted as a whole number in thousands. Add this measure to the PivotTable.
3. Generate another explicit measure named `hr_pct` that calculates the percentage of total home runs (`HR`) out of total at bats (`AB`) from the batting table. Format the result as a percentage. Feel free to create an additional total at bats measure to assist with this.
4. Develop a KPI based on the metric `hr_pct` aiming for a target absolute value of 1. Use the following status thresholds:
   - Less than 2%: Red status
   - Between 2% and 3%: Yellow status
   - Greater than 3%: Green status
5. Apply the KPI to a PivotTable that displays `teamID` along the Rows and `yearID` along the Columns.

Refer to *ch_08_solutions.xlsx* in the same folder for solutions.

CHAPTER 9

Intermediate DAX for Power Pivot

In Chapter 8, you explored basic DAX measures for reporting. Now, in the final chapter of Part II, we dive into intermediate DAX tasks, enhancing PivotTable reporting in Excel.

To participate in the demonstrations, please open the *ch_09.xlsx* file, which is located in the *ch_09* folder of the book's companion repository. We will use the same retail sales dataset as in previous chapters.

This Excel workbook includes a PivotTable linked to the Data Model, and it features a predefined measure named `Total sales`. This measure calculates the sum of the `Sales` column from the `orders` table, and it will be used in various demonstrations that follow.

CALCULATE() and the Importance of Filter Context

In traditional PivotTables, all values adhere to the main filter. For instance, in Figure 9-1, if you filter by `Ship Mode` for "Express Air," you can't see total sales simultaneously. You either get overall sales or just Express Air sales, but not both.

	A	B	C	D
1				
2		Ship Mode	Express Air	
3				
4		**Total sales**		
5		$1,172,357.55		
6				

Figure 9-1. `Total sales` is now evaluated in a filter context

In eloquent terms, every value in a PivotTable adheres to its "filter context." However, the `CALCULATE()` function liberates measures from this constraint, enabling them to work within an modified filter context. This revolutionizes PivotTable capabilities.

For as powerful as the `CALCULATE()` function is, its syntax is rather simple, as shown in Table 9-1.

Table 9-1. The parameters of `CALCULATE()` explained

Parameter	Data type	Description
`expression`	Any valid DAX expression	The expression to be evaluated or calculated. This can be a measure, a column, or another function.
`[filter1], [filter2], [...]`	Column, Table, or Boolean expression	Optional. The filter or filters to be applied to the expression. This can be a single column, a table, or a Boolean expression.

CALCULATE() with One Criterion

Begin by crafting a measure named `Total express air sales`. This measure should filter `Total sales` to orders that have `Ship Mode` set to "Express Air," as shown in Figure 9-2.

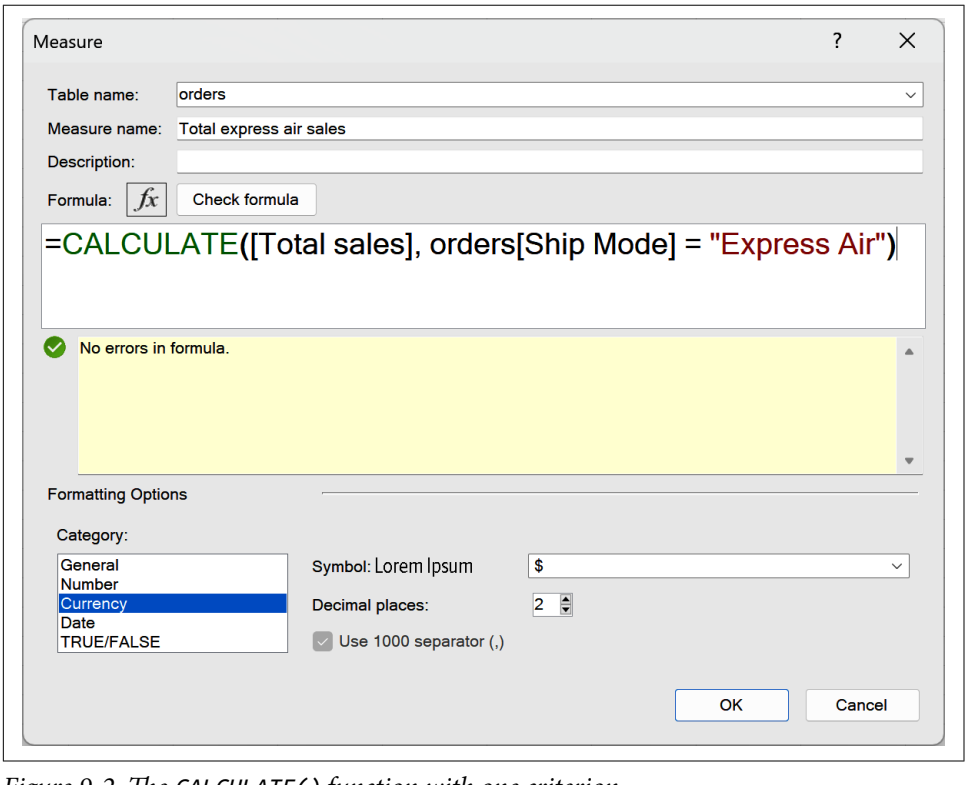
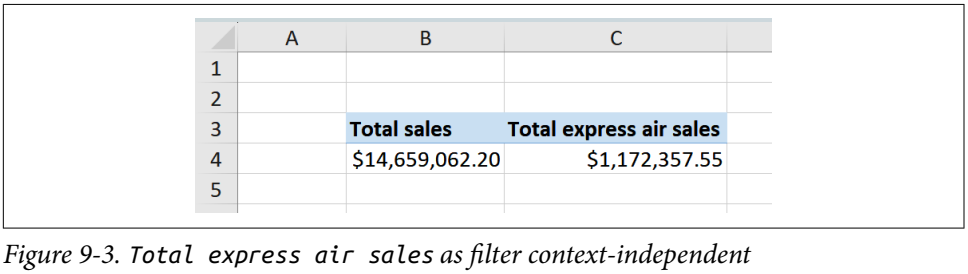


Figure 9-2. The `CALCULATE()` function with one criterion

Once you've crafted the measure, add it to the PivotTable next to `Total sales` and remove `Ship Mode` from the PivotTable. This lets you see both overall and Express Air sales concurrently in the PivotTable, as depicted in Figure 9-3.



	A	B	C
1			
2			
3		Total sales	Total express air sales
4		$14,659,062.20	$1,172,357.55
5			

Figure 9-3. `Total express air sales` as filter context-independent

You've transformed how your PivotTable evaluates individual data points. This is a groundbreaking advancement in PivotTable capabilities.

CALCULATE() with Multiple Criteria

It's also possible to modify the filter context using various conditions with `CALCULATE()`. This section will explore how to incorporate and/or criteria into our functions.

AND Conditions

Given that high-priority orders are most vulnerable to air freight disruptions, it might be helpful to review sales where both `Order Priority` is High and `Ship Mode` is Express Air.

Incorporating a second `AND` condition to a `CALCULATE()` measure merely involves adding an additional filter parameter, as illustrated in Figure 9-4.

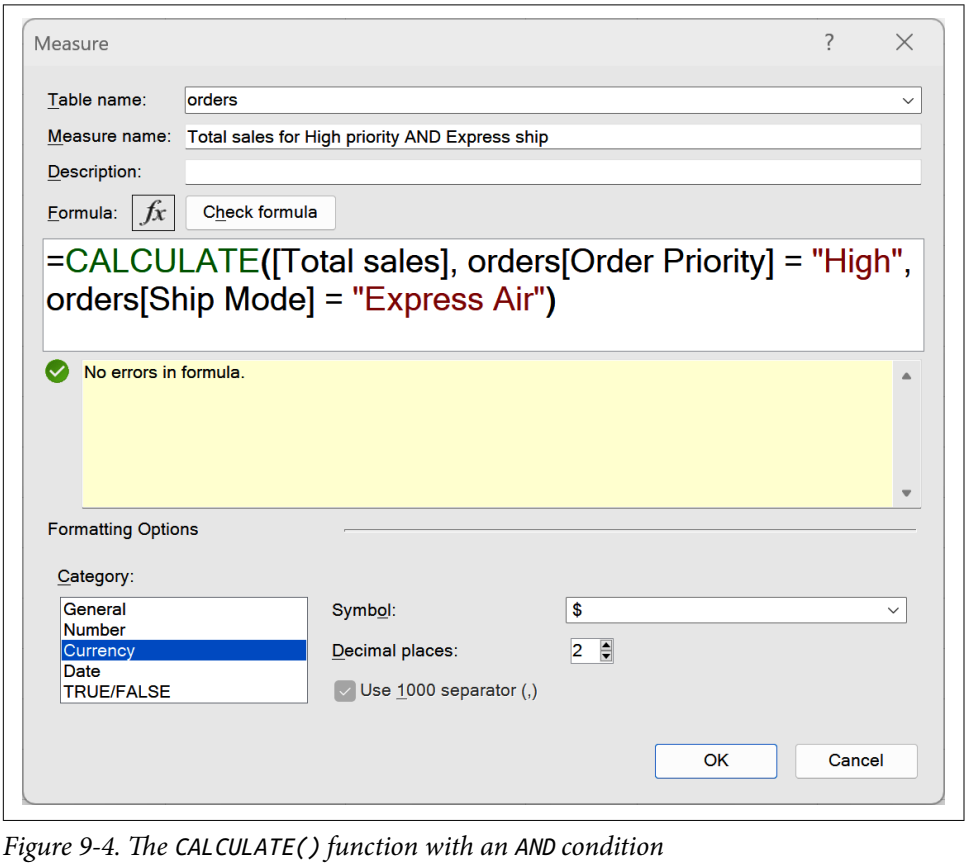


Figure 9-4. The `CALCULATE()` function with an `AND` condition

OR Conditions

With conditional logic, always be aware of result sensitivity. Even minor alterations in criteria can produce drastically different results.

For example, validate the sales amount by filtering orders where `Order Priority` is High *or* `Ship Mode` is Express Air. In `CALCULATE()`, combine the conditions using two `|` symbols, as shown in Figure 9-5.

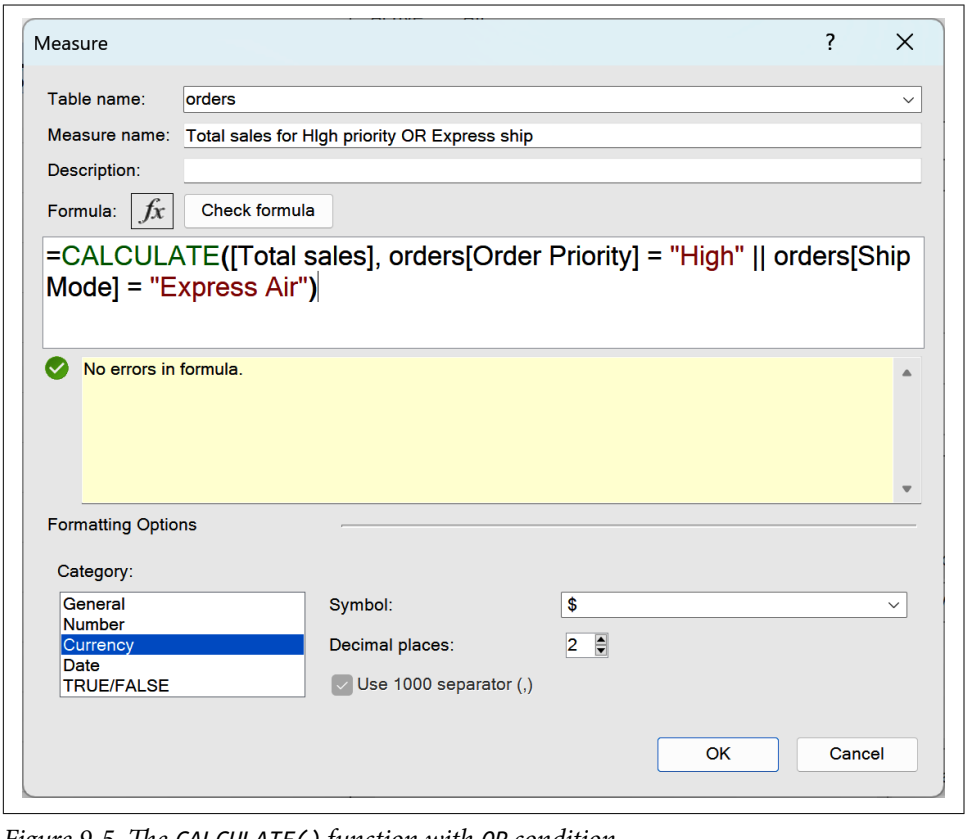


Figure 9-5. The `CALCULATE()` function with `OR` condition

CALCULATE() with ALL()

`CALCULATE()` can add to the filter context, but when paired with `ALL()`, it explicitly *clears all filter context* for a value. To understand this difference, consider the PivotTable in Figure 9-6. Both the `Total sales` measure and the filter context-altered `Total express air sales` fluctuate based on the overarching `Product Category` filter.

	A	B	C	D	E	F
1		Product Category	Office Supplies			
2						
3		**Row Labels**	**Total sales**	**Total express air sales**	**Total sales for High priority AND Express ship**	**Total sales for High priority OR Express ship**
4		Central	$293,175.13	$29,472.47	$1,175.20	$85,720.11
5		Mid-Atlantic	$354,770.45	$39,211.78	$3,347.29	$103,395.10
6		Midwest	$822,744.58	$111,774.44	$32,535.80	$253,007.40
7		Northeast	$570,737.26	$69,083.95	$11,675.41	$177,060.79
8		Pacific	$64,565.18	$11,739.70		$24,596.95
9		Pacific Northwest	$161,470.81	$14,112.92	$1,941.95	$35,228.42
10		South	$786,499.83	$125,652.99	$26,255.53	$370,723.24
11		West	$628,879.49	$71,732.89	$16,345.86	$182,583.44
12		**Grand Total**	**$3,682,842.73**	**$472,781.14**	**$93,277.04**	**$1,232,315.45**
13						

Figure 9-6. Results of `CALCULATE()` are affected by the PivotTable filter

To compute *all* base values irrespective of the overarching filter context, pair the `ALL()` function with `CALCULATE()`. For instance, I'll craft a measure named `All total sales`, as shown in Figure 9-7.

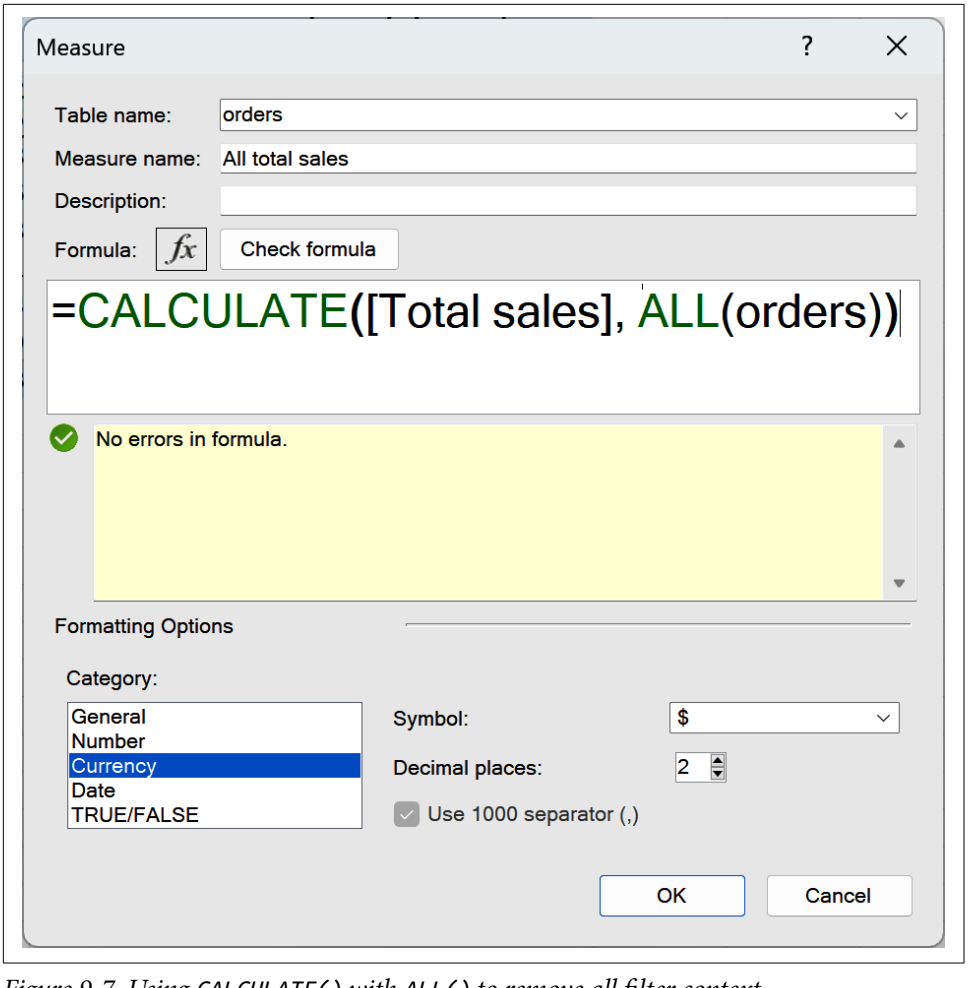


Figure 9-7. Using `CALCULATE()` with `ALL()` to remove all filter context

By applying `ALL(orders)` as the filter criteria, every record in the table is considered for calculation, overriding any other existing filter context in the PivotTable. The distinction is seen in Figure 9-8.

This is handy for contrasting specific sales data combinations with the overarching total, regardless of applied filters.

	A	B	C	D
1				
2		Product Category	(Multiple Items)	
3				
4		**Row Labels**	**Total sales**	**All total sales**
5		Central	$725,994.94	$14,659,062.20
6		Mid-Atlantic	$857,744.48	$14,659,062.20
7		Midwest	$1,844,183.47	$14,659,062.20
8		Northeast	$1,235,775.95	$14,659,062.20
9		Pacific	$175,674.06	$14,659,062.20
10		Pacific Northwest	$407,247.28	$14,659,062.20
11		South	$2,009,764.91	$14,659,062.20
12		West	$1,543,525.47	$14,659,062.20
13		**Grand Total**	**$8,799,910.55**	**$14,659,062.20**
14				

Figure 9-8. `All total sales` in the PivotTable

The `CALCULATE()` function holds significance akin to lookup functions and PivotTables in classic Excel. It signifies a leap to advanced capabilities. For deeper insights on filter context and the `CALCULATE()` function, refer to *The Definitive Guide to DAX*, second edition, by Alberto Ferrari and Marco Russo (Microsoft Press, 2019).

Time Intelligence Functions

> *At the heart of quantitative reasoning is a single question: Compared to what?*
>
> —Edward Tufte

In the business world, analyzing trends is crucial. Analysts examine current performance against historical data and assess monthly and yearly metrics. While traditional Excel methods can be unwieldy for this purpose, Power Pivot offers a streamlined approach.

Power Pivot introduces *time intelligence* capabilities, providing functions that facilitate analysis of time-based data, such as year-to-date totals and month-over-month growth. This eliminates the complexity of intricate formulas, simplifying trend analysis in Excel.

Adding a Calendar Table

To use time intelligence in Power Pivot effectively, begin by adding a calendar table. This provides a consistent and comprehensive date-time structure, enhancing data analysis accuracy and enabling more complex time-based calculations and

comparisons. In your Data Model, navigate to the Power Pivot tab, select Manage, go to the Design tab, and select Date Table → New, as illustrated in Figure 9-9.

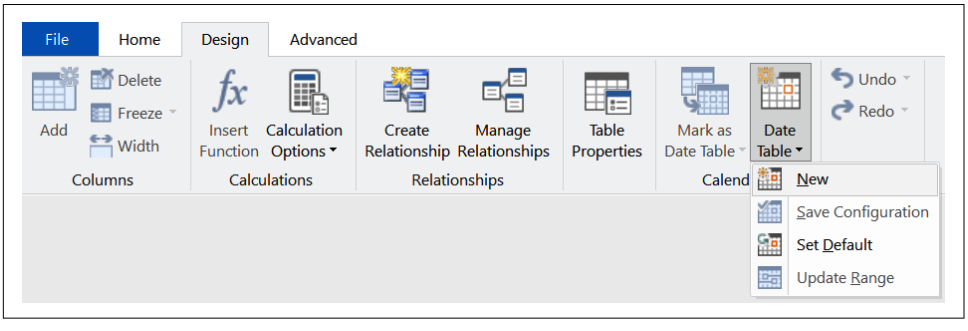


Figure 9-9. Adding a Date Table to the Data Model

You should now see a calendar table in your Data Model. Establish a relationship between the `Date` column of this table and the `Order Date` column in the `orders` table. Your Data Model should resemble Figure 9-10.

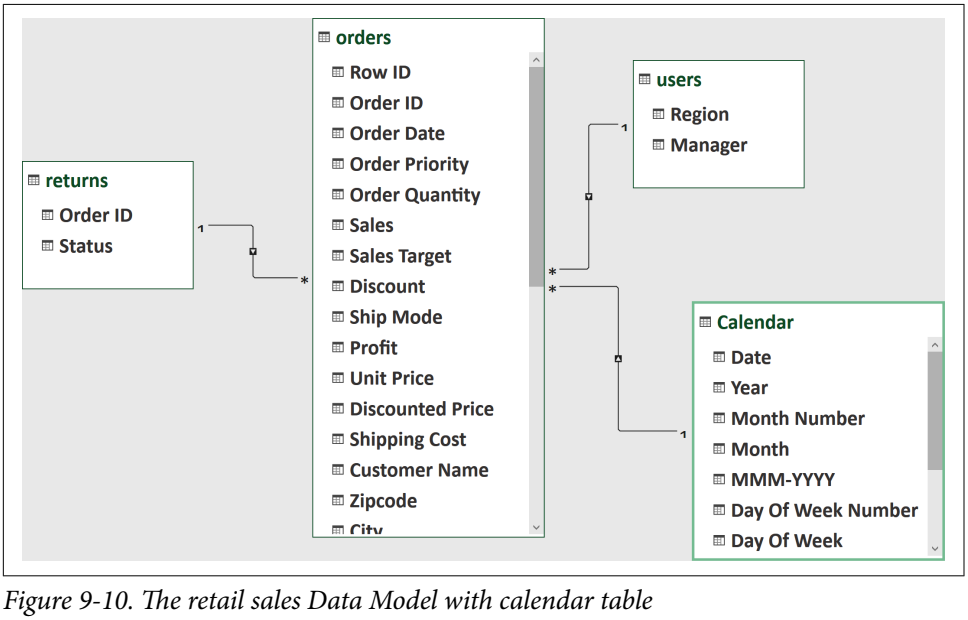


Figure 9-10. The retail sales Data Model with calendar table

Now you can leverage the various date measures from the calendar table in your Data Model, all set in relation to the `Order Date` in the `orders` table.

Changing Date Reference with USERELATIONSHIP()

Only one relationship between the calendar table and the `orders` table can be active at once. This means that if you want to align your analysis with the `Ship date` instead of the `Order Date` (for example, to gain insights into logistics or customer satisfaction), you'll need to shift the date relationship using the `USERELATIONSHIP()` function paired with `CALCULATE()` to modify which calendar basis is used in a particular measure.

To ensure the calendar table is functioning properly, insert a new PivotTable into your workbook. Drag `Date Hierarchy` from the calendar table into Rows and `Total sales` from `orders` into Values, as shown in Figure 9-11.

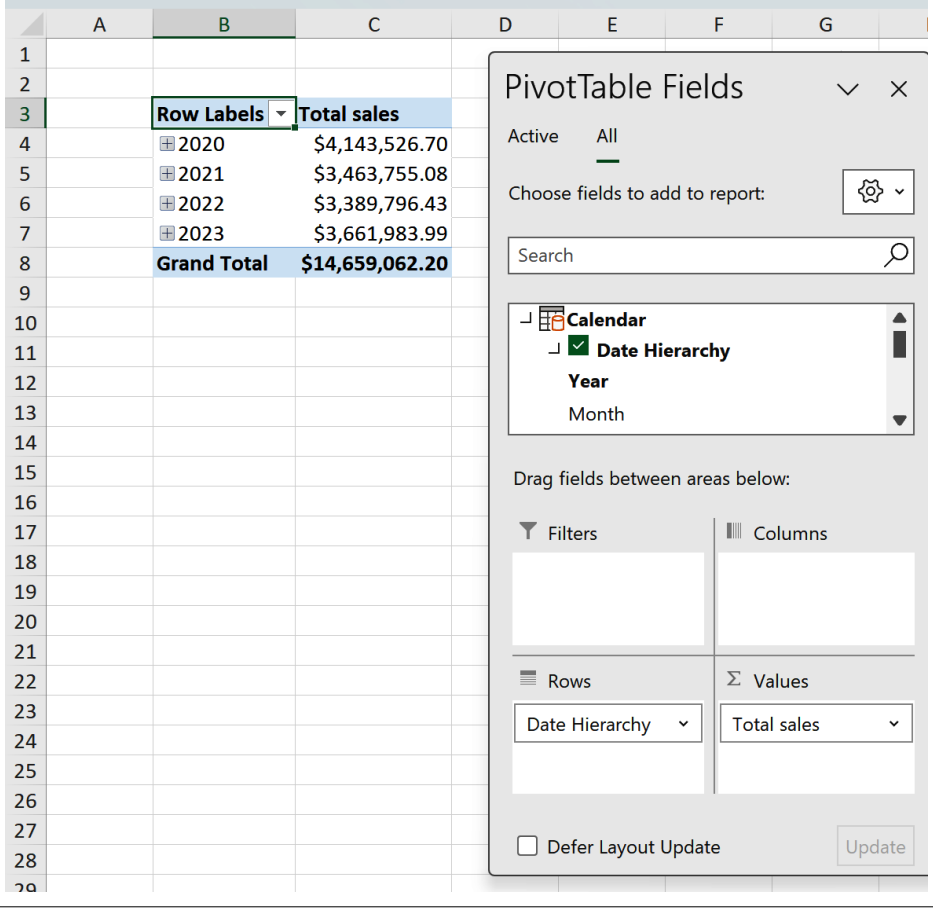


Figure 9-11. Using the calendar table in the PivotTable

Creating Basic Time Intelligence Measures

DAX offers an abundance of time intelligence functions, allowing you to retrieve previous periods, periods-to-date, and beyond. For instance, to calculate a year-to-date sales measure, use the `TOTALYTD()` formula, as shown in Figure 9-12.

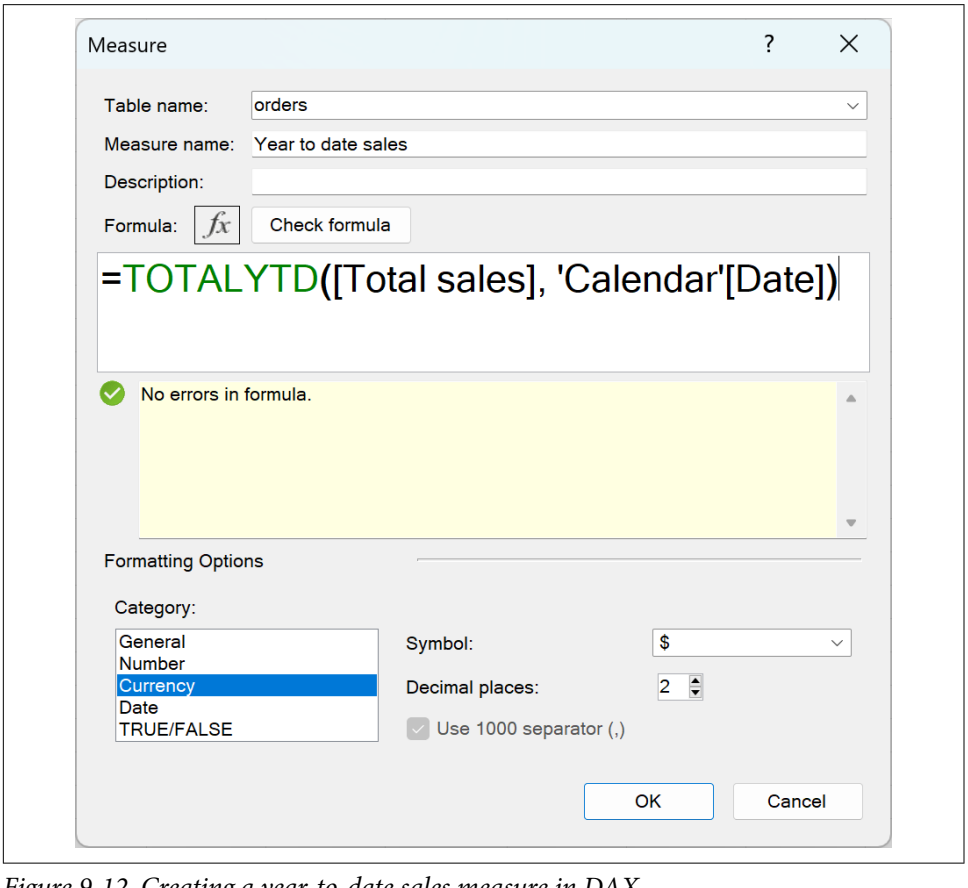


Figure 9-12. Creating a year-to-date sales measure in DAX

To verify this measure's accuracy, add it to a PivotTable along with `Date Hierarchy` in the Rows. Expand the data for the year 2020 by clicking the + sign. You should observe the measure incrementally increasing with each month, as depicted in Figure 9-13.

	A	B	C	D
1				
2				
3		**Row Labels**	**Total sales**	**Year to date sales**
4		**2020**		
5		January	$506,492.34	$506,492.34
6		February	$338,766.52	$845,258.86
7		March	$410,095.71	$1,255,354.57
8		April	$361,379.00	$1,616,733.57
9		May	$232,891.66	$1,849,625.23
10		June	$285,060.40	$2,134,685.63
11		July	$355,061.09	$2,489,746.71
12		August	$332,038.38	$2,821,785.10
13		September	$311,597.75	$3,133,382.85
14		October	$350,730.38	$3,484,113.23
15		November	$246,465.57	$3,730,578.80
16		December	$412,947.91	$4,143,526.70
17		**2021**	**$3,463,755.08**	**$3,463,755.08**
18		**2022**	**$3,389,796.43**	**$3,389,796.43**
19		**2023**	**$3,661,983.99**	**$3,661,983.99**
20		**Grand Total**	**$14,659,062.20**	**$3,661,983.99**
21				

Figure 9-13. Displaying year-to-date sales in a PivotTable

Next, to determine sales from the previous year's period, combine the familiar `CALCULATE()` function with the `SAMEPERIODLASTYEAR()` function, as demonstrated in Figure 9-14.

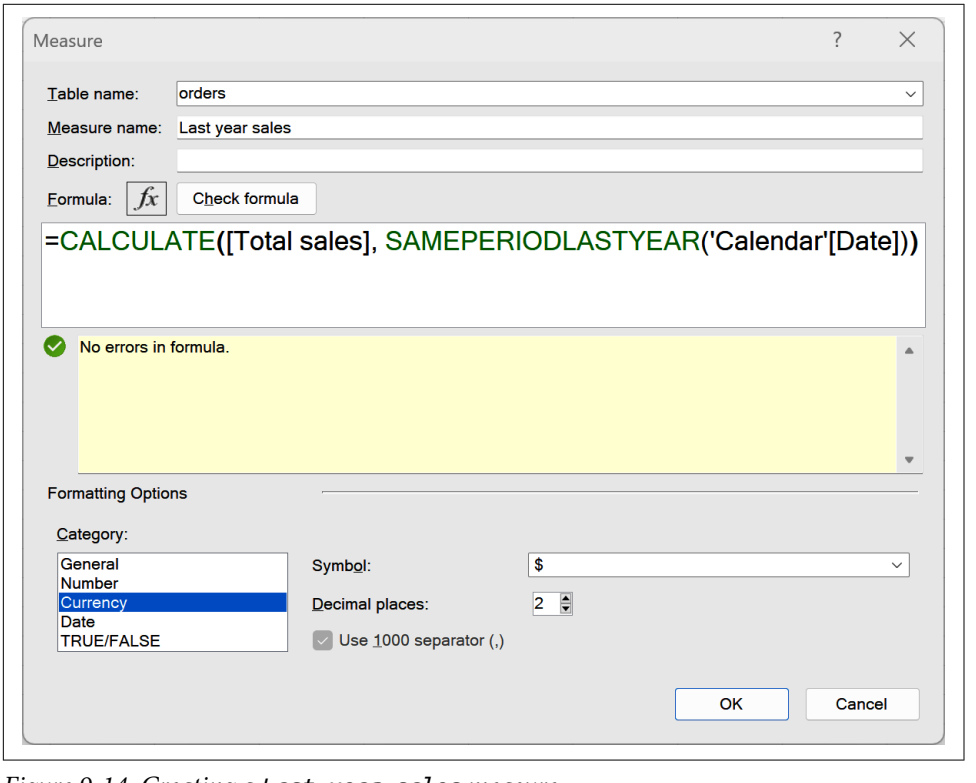


Figure 9-14. Creating a `Last year sales` measure

To ensure accuracy, preview the data in the PivotTable along with `Total sales`. You should see that the last year's sales values for 2021 the total sales for 2020, as shown in Figure 9-15.

Row Labels	Total sales	Last year sales
2020		
January	$506,492.34	
February	$338,766.52	
March	$410,095.71	
April	$361,379.00	
May	$232,891.66	
June	$285,060.40	
July	$355,061.09	
August	$332,038.38	
September	$311,597.75	
October	$350,730.38	
November	$246,465.57	
December	$412,947.91	
2021		
January	$329,835.43	$506,492.34
February	$262,774.37	$338,766.52
March	$232,099.30	$410,095.71
April	$237,092.18	$361,379.00
May	$301,493.40	$232,891.66
June	$258,956.58	$285,060.40
July	$229,323.67	$355,061.09
August	$202,630.03	$332,038.38
September	$404,141.93	$311,597.75
October	$360,294.92	$350,730.38
November	$296,435.95	$246,465.57
December	$348,677.33	$412,947.91
2022	$3,389,796.43	$3,463,755.08
2023	$3,661,983.99	$3,389,796.43
Grand Total	$14,659,062.20	$10,997,078.21

Figure 9-15. Comparing this year's to last year's sales

Lastly, establish a `Last year YTD sales` measure to contrast this year's year-to-date sales with the previous year's.

To accomplish this, you can combine `CALCULATE()` with both the `DATESYTD()` and `DATEADD()` functions. This setup will retrieve all year-to-date dates, adjusted backward by one year using the calendar table. The outcome is illustrated in Figure 9-16.

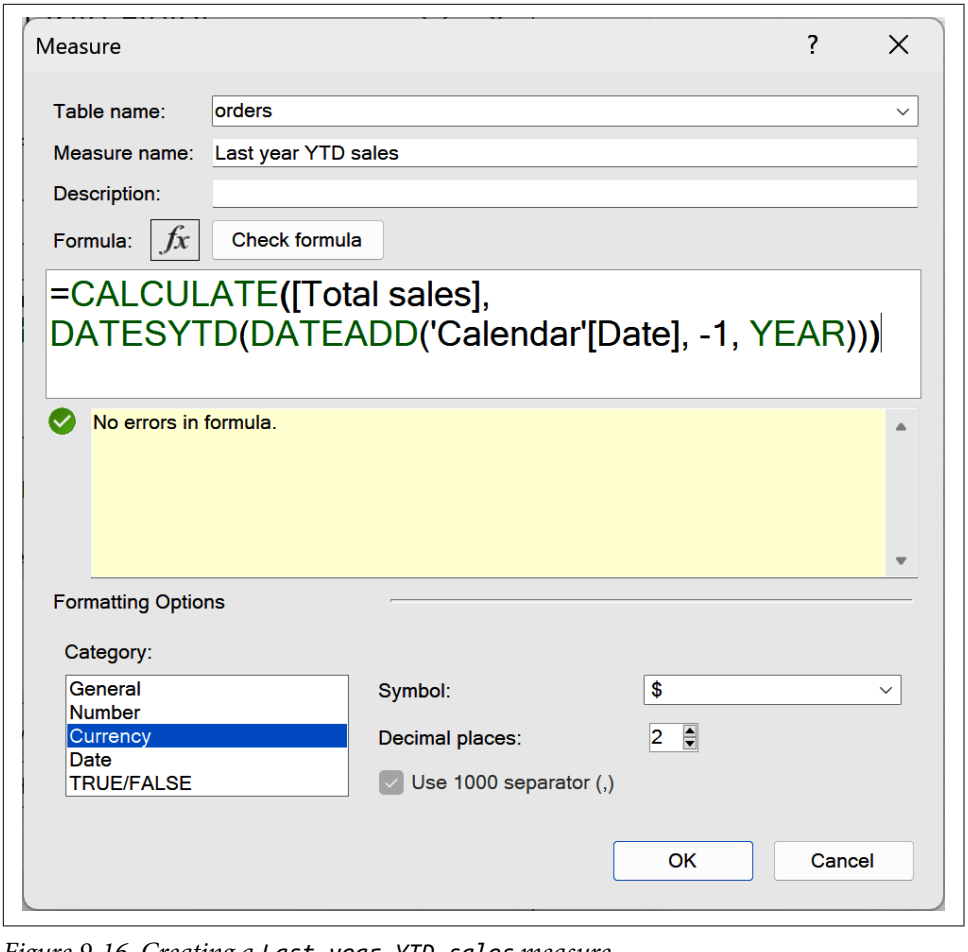


Figure 9-16. Creating a `Last year YTD sales` measure

You can now compare year-to-date trends for both the current and last year, as shown in Figure 9-17.

Row Labels	Year to date sales	Last year YTD sales
2020		
January	$506,492.34	
February	$845,258.86	
March	$1,255,354.57	
April	$1,616,733.57	
May	$1,849,625.23	
June	$2,134,685.63	
July	$2,489,746.71	
August	$2,821,785.10	
September	$3,133,382.85	
October	$3,484,113.23	
November	$3,730,578.80	
December	$4,143,526.70	
2021		
January	$329,835.43	$506,492.34
February	$592,609.80	$845,258.86
March	$824,709.10	$1,255,354.57
April	$1,061,801.28	$1,616,733.57
May	$1,363,294.67	$1,849,625.23
June	$1,622,251.25	$2,134,685.63
July	$1,851,574.91	$2,489,746.71
August	$2,054,204.95	$2,821,785.10
September	$2,458,346.88	$3,133,382.85
October	$2,818,641.80	$3,484,113.23
November	$3,115,077.75	$3,730,578.80
December	$3,463,755.08	$4,143,526.70
2022	**$3,389,796.43**	**$3,463,755.08**
2023	**$3,661,983.99**	**$3,389,796.43**
Grand Total	**$3,661,983.99**	**$3,389,796.43**

Figure 9-17. Comparing this year and last year year-to-date sales

Conclusion

DAX and Excel always offer something new to learn, and you can combine functions in countless ways. This chapter might be ending, but there's more to discover with Power Pivot in Excel.

So far, the book has explored Power Query and Power Pivot for data cleaning and modeling, respectively. But there's more data analytics in Excel. Part III touches on other features to help make your Excel projects even more dynamic and insightful.

Exercises

To practice building intermediate DAX measures, use the bike shop sales dataset found in *ch_09_exercises.csv* in the *exercises\ch_09_exercises* folder in the book's companion repository. Despite the book's focus on using Power Pivot to establish relationships between multiple tables in the same workbook, it can still be helpful to use with a single *.csv* file.

Load the data into Power Pivot via Power Query and create the following measures:

1. `accessories_rev`: Returns the total revenue when `product_category` is set to Accessories.
2. `accessories_rev_aus`: Returns the total revenue when both `product_category` is Accessories and `country` is Australia.
3. `aov_all`: Calculates the total revenue divided by the total order quantity across the entire dataset, irrespective of applied filters.
4. `profit_margin_ytd`: Returns the year-to-date profit margin.
5. `profit_margin_ly_ytd`: Returns the year-to-date profit margin for the previous year.

Feel free to create intermediary measures that aid in constructing the desired measures. For instance, when seeking year-to-date sales totals, it is helpful to first create a measure for total sales.

Ensure that your measures function as intended by testing them within a PivotTable. For example, if you're constructing a filter context-independent measure, apply a filter to observe its behavior. When developing a year-to-date measure, aggregate the measure by a date to verify its appropriate response. Additionally, even though we are working with a single table, it is still beneficial to include a calendar table for date-related operations.

Refer to *ch_09_solutions.xlsx* in the same folder for solutions.

PART III

The Excel Data Analytics Toolkit

CHAPTER 10

Introducing Dynamic Array Functions

Thus far, this book has covered building measures for Power Pivot with DAX and touched on M code in Power Query to a lesser extent. However, it hasn't examined the traditional workbook formulas and functions that have long been the foundation of Excel. Even this area of the program, which may have seemed neglected in favor of other flashy developments like Power Pivot and Power Query, has undergone significant improvements, becoming more powerful and capable.

This chapter introduces dynamic array functions, shedding light on their capabilities. You will learn how to sort, filter, and join datasets, among other tasks, using the familiar environment of the Excel formula bar.

Formulas Versus Functions in Excel

In Excel, formulas and functions have distinct meanings. A *formula* is a mathematical expression that manipulates data using operators, cell references, and constants; for example, `=B1 * B2`. In contrast, a *function* is a predefined formula providing extensive data analysis and manipulation capabilities; for example, `TRIMMEAN(B1:B3)`.

Formulas can include functions, blurring the distinction. This chapter aims to respect these as separate concepts, while recognizing their interdependence.

Dynamic Array Functions Explained

Dynamic array functions have impressive capabilities, and it's tempting to start experimenting with them right away. However, it's crucial to understand what makes these functions special and how they differ from traditional Excel approaches. The following section explores the path from arrays to array references and finally to dynamic array functions.

What Is an Array in Excel?

To follow this demonstration, open *ch_10.xlsx* in the *ch_10* folder of the book's resources. Go to the `array-references` worksheet.

First and foremost, an *array* in Excel refers to a collection of values. For instance, a basic array could consist of the numbers 3, 4, and 7 placed in cells `A2:C2`, as shown in Figure 10-1.

	A	B	C
1	Array:		
2	3	4	7

Figure 10-1. A basic Excel array

Arrays Versus Ranges in Excel

`A2:C2` is an example of a range, yet we're referring to it as an array. What's the difference? The *range* is the set of cells and their physical location, whereas the array is the data in those cells. You can read more about the difference on Microsoft's Tech Community Forum.

Array References

With an understanding of what constitutes an array in Excel, this next section explores various approaches to build array references.

Static array references

To create a tradititional Excel array reference, enter `=A2:C2` into cell `E2` and press Ctrl+Shift+Enter to indicate that you are referencing an array of values rather than a single value, as in Figure 10-2.

C2 {=A2:C2}

	A	B	C	D	E	F	G	H
1	Array:				Static array reference (Ctrl + Shift + Enter):			
2	3	4	7		3			
3								

Figure 10-2. A basic Excel array reference

You will see the resulting formula enclosed within curly brackets `{}`, but not all three values will be displayed in the result. This occurs because, in Excel, each cell is designed to hold only a single data point, not three, as you might be attempting to achieve here. To distribute data from an array across multiple cells, select the range `E2:G2` and enter the same reference, as shown in Figure 10-3.

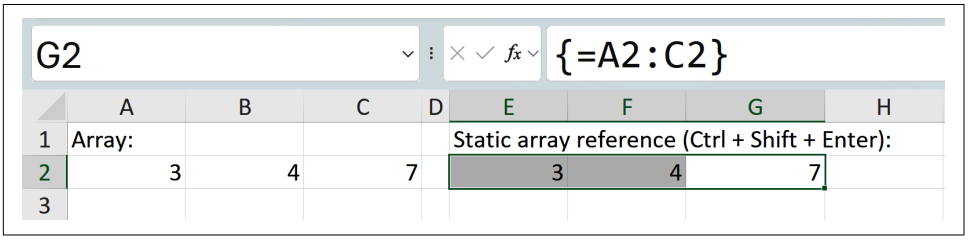


Figure 10-3. An improved Excel array reference

Excel's conventional approach to handling arrays comes with limitations. The process of writing and managing references using Ctrl+Shift+Enter can be arduous and lacks automatic adjustment.

Consider a scenario where cells are inserted or deleted between `A2:C2`—in such cases, the array reference fails to resize itself. Consequently, these array references could be described as *static* because they do not dynamically adapt to changes in the spreadsheet's structure or cell count.

Dynamic array references

Dynamic arrays were introduced in Excel in 2018 to overcome the limitations associated with traditional static arrays. Now, to refer to `A2:C2`, you simply need to type **`=A2:C2`** and press Enter, such as in cell `E5` of Figure 10-4.

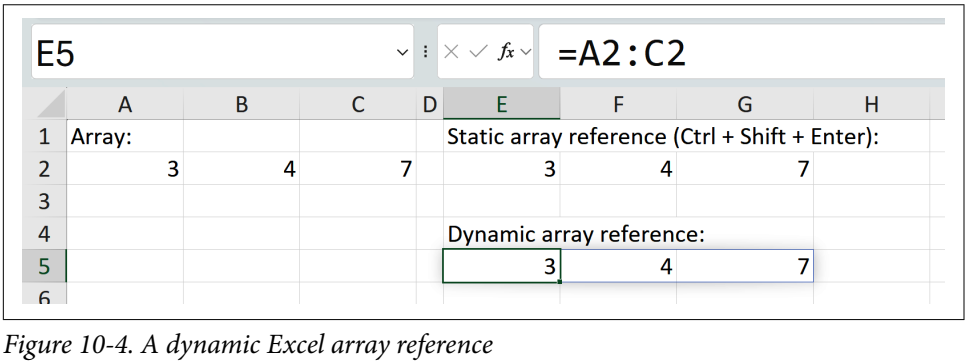


Figure 10-4. A dynamic Excel array reference

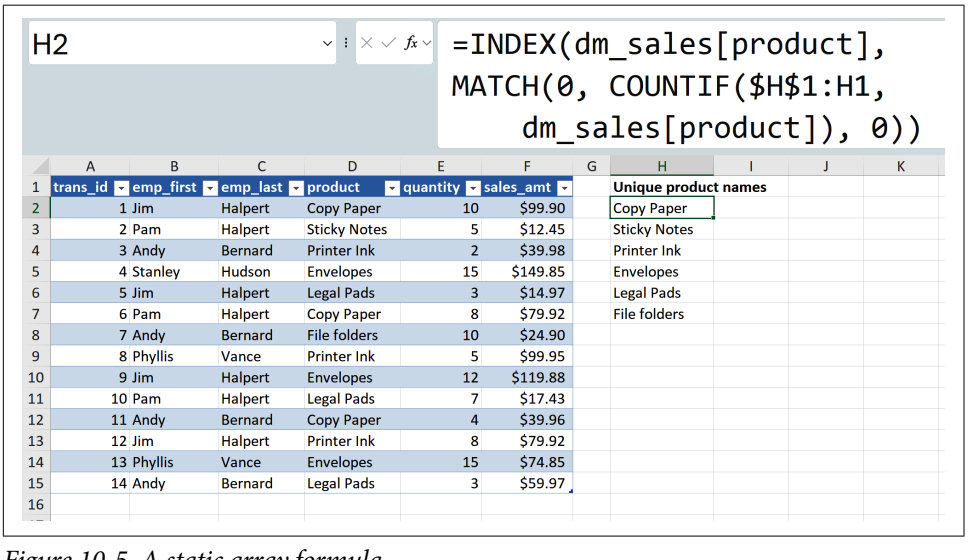
Using this reference, Excel intelligently recognizes the number of cells in the array. This means that if you insert or delete cells between `A2` and `C2`, the dynamic array reference will automatically resize to accommodate the changes. This dynamic adjustment saves time and effort by removing the need to manually update or modify the references whenever the data layout is altered.

Array Formulas

Having contrasted the behavior of array references in classic and modern Excel, the next section delves into the impact on functions using these references.

Static array formulas

Consider the following example, which uses a static array formula to list the unique products sold in a transactions dataset, as seen in Figure 10-5. You can follow along with this example in the `array-functions` worksheet of *ch_10.xlsx*.



trans_id	emp_first	emp_last	product	quantity	sales_amt
1	Jim	Halpert	Copy Paper	10	$99.90
2	Pam	Halpert	Sticky Notes	5	$12.45
3	Andy	Bernard	Printer Ink	2	$39.98
4	Stanley	Hudson	Envelopes	15	$149.85
5	Jim	Halpert	Legal Pads	3	$14.97
6	Pam	Halpert	Copy Paper	8	$79.92
7	Andy	Bernard	File folders	10	$24.90
8	Phyllis	Vance	Printer Ink	5	$99.95
9	Jim	Halpert	Envelopes	12	$119.88
10	Pam	Halpert	Legal Pads	7	$17.43
11	Andy	Bernard	Copy Paper	4	$39.96
12	Jim	Halpert	Printer Ink	8	$79.92
13	Phyllis	Vance	Envelopes	15	$74.85
14	Andy	Bernard	Legal Pads	3	$59.97

Figure 10-5. A static array formula

Don't worry too much about the formula's inner workings; we'll explore a more sensible alternative shortly. For now, observe how a static array can be challenging when it comes to determining the number of output pieces it should return.

Similar to array references, this approach lacks automatic adjustment to list the correct number of unique values, making it cumbersome and unintuitive. For example, if more transactions are added to the `dm_sales` table, any additional unique values, such as Rubber bands in this case, are not reflected in the array function, as seen in Figure 10-6.

H2

```
=INDEX(dm_sales[product],
MATCH(0, COUNTIF($H$1:H1,
    dm_sales[product]), 0))
```

	A	B	C	D	E	F	G	H	I	J	K
1	trans_id	emp_first	emp_last	product	quantity	sales_amt		Unique product names			
2	1	Jim	Halpert	Copy Paper	10	$99.90		Copy Paper			
3	2	Pam	Halpert	Sticky Notes	5	$12.45		Sticky Notes			
4	3	Andy	Bernard	Printer Ink	2	$39.98		Printer Ink			
5	4	Stanley	Hudson	Envelopes	15	$149.85		Envelopes			
6	5	Jim	Halpert	Legal Pads	3	$14.97		Legal Pads			
7	6	Pam	Halpert	Copy Paper	8	$79.92		File folders			
8	7	Andy	Bernard	File folders	10	$24.90					
9	8	Phyllis	Vance	Printer Ink	5	$99.95					
10	9	Jim	Halpert	Envelopes	12	$119.88					
11	10	Pam	Halpert	Legal Pads	7	$17.43					
12	11	Andy	Bernard	Copy Paper	4	$39.96					
13	12	Jim	Halpert	Printer Ink	8	$79.92					
14	13	Phyllis	Vance	Envelopes	15	$74.85					
15	14	Andy	Bernard	Legal Pads	3	$59.97					
16	15	Stanley	Hudson	Rubber bands	60	$14.94					
17											

Figure 10-6. Rubber bands (row 16) are not listed as a unique product

To see that extra value, you'll need to extend your array formula in column `H` down one more row.

Dynamic array functions

Dynamic array functions, on the other hand, perform exceptionally well in this scenario. There is even a `UNIQUE()` function specifically designed to handle this task, as seen in Figure 10-7.

If you see a `#SPILL` error after using the `UNIQUE()` function, make sure you have empty cells below it. This error happens when the results of the function spill over to non-empty adjacent cells.

Dynamic array functions offer a significant advancement over traditional array formulas by instantly updating output cells in response to changes in inputs. This dynamic behavior eliminates the need for manual recalculations or refreshing formulas, providing a seamless and efficient workflow.

I2 =UNIQUE(dm_sales[product])

	A	B	C	D	E	F	G	H	I	J
1	trans_id	emp_first	emp_last	product	quantity	sales_amt			Unique product names	
2	1	Jim	Halpert	Copy Paper	10	99.9			Copy Paper	
3	2	Pam	Beesly	Sticky Notes	5	12.45			Sticky Notes	
4	3	Andy	Bernard	Printer Ink	2	39.98			Printer Ink	
5	4	Stanley	Hudson	Envelopes	15	149.85			Envelopes	
6	5	Jim	Halpert	Legal Pads	3	14.97			Legal Pads	
7	6	Pam	Beesly	Copy Paper	8	79.92			Rubber bands	
8	7	Andy	Bernard	Sticky Notes	10	24.9				
9	8	Phyllis	Vance	Printer Ink	5	99.95				
10	9	Jim	Halpert	Envelopes	12	119.88				
11	10	Pam	Beesly	Legal Pads	7	17.43				
12	11	Andy	Bernard	Copy Paper	4	$39.96				
13	12	Jim	Halpert	Printer Ink	8	$79.92				
14	13	Phyllis	Vance	Envelopes	15	$74.85				
15	14	Andy	Bernard	Legal Pads	3	$59.97				
16	15	Stanley	Hudson	Rubber bands	60	$14.94				
17										

Figure 10-7. Finding unique values with the `UNIQUE()` function

An Overview of Dynamic Array Functions

With the capabilities of dynamic array functions in mind, it's time to explore a few examples. The first will expand on the `UNIQUE()` function discussed earlier.

Finding Distinct and Unique Values with UNIQUE()

In the previous example, the `UNIQUE()` dynamic array function was used to generate a list of unique product names. To explore this function and dataset further, continue using the `dm_sales` table in the *ch_10.xlsx* workbook.

Parameters and Arguments in Excel

Parameters and arguments in Excel are often used interchangeably, but they have distinct meanings. *Parameters* are variables within a function that act as placeholders for values, while *arguments* are the actual values or references used in a formula. For instance, the `ABS()` function takes a parameter called `number`, and when using the function like `ABS(-10)`, `-10` is the argument.

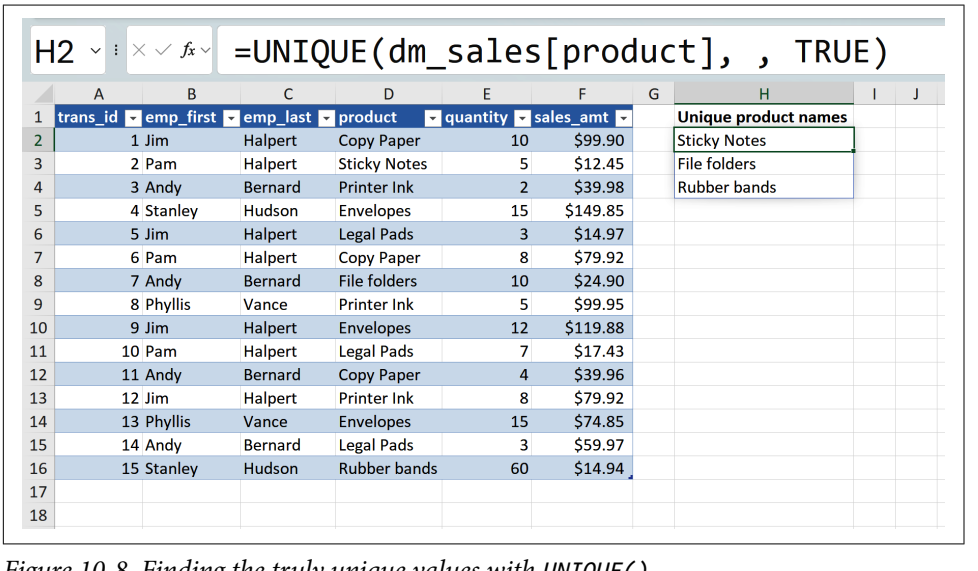
The `UNIQUE()` function has three parameters, two of which are optional. To understand how they work, refer to Table 10-1.

Table 10-1. `UNIQUE()` parameters

Parameter	Description
`range`	Required argument that specifies the data range or array from which you want to extract unique values.
`[by_col]`	Optional argument that determines whether the unique values should be extracted by column or row. By default, unique values are extracted by row. If you set this parameter to `TRUE`, unique values will be extracted by row. If you set this parameter to TRUE, unique values will be extracted by column.
`[exactly_once]`	Optional argument that specifies whether only values that appear exactly once should be considered unique. By default, all unique values, regardless of their frequency, are extracted. If you set this parameter to `TRUE`, only the values that appear exactly once will be extracted.

Finding Unique Versus Distinct Values

In database terminology, *unique values* refer to those appearing only once within a given range. This makes the name of the `UNIQUE()` function somewhat misleading. This function actually identifies *distinct* values, meaning those found once or more, rather than strictly unique ones. However, it's possible to obtain genuinely unique values by setting the third argument of the function to `TRUE`, as shown in Figure 10-8.



trans_id	emp_first	emp_last	product	quantity	sales_amt
1	Jim	Halpert	Copy Paper	10	$99.90
2	Pam	Halpert	Sticky Notes	5	$12.45
3	Andy	Bernard	Printer Ink	2	$39.98
4	Stanley	Hudson	Envelopes	15	$149.85
5	Jim	Halpert	Legal Pads	3	$14.97
6	Pam	Halpert	Copy Paper	8	$79.92
7	Andy	Bernard	File folders	10	$24.90
8	Phyllis	Vance	Printer Ink	5	$99.95
9	Jim	Halpert	Envelopes	12	$119.88
10	Pam	Halpert	Legal Pads	7	$17.43
11	Andy	Bernard	Copy Paper	4	$39.96
12	Jim	Halpert	Printer Ink	8	$79.92
13	Phyllis	Vance	Envelopes	15	$74.85
14	Andy	Bernard	Legal Pads	3	$59.97
15	Stanley	Hudson	Rubber bands	60	$14.94

Figure 10-8. Finding the truly unique values with `UNIQUE()`

Using the Spill Operator

In Excel, it is common to create additional calculations on top of existing ones, such as aggregating the results of a calculated column. The *spill operator*, denoted by the hash symbol (#), streamlines the aggregation process for dynamic array functions. Like dynamic arrays themselves, it automatically expands the output range to accommodate the data, eliminating the manual entry of array formulas and resizing of ranges. This feature enhances efficiency and conciseness when constructing formulas for aggregating data in Excel.

The number of unique values generated by the formula displayed in Figure 10-8 can be determined using the `COUNTA()` function. When selecting the range `D2:D7`, Excel automatically refers to it using the spill range indicated by the `#` operator, as in Figure 10-9.

I6 =COUNTA(H2#)

	A	B	C	D	E	F	G	H	I	J
1	trans_id	emp_first	emp_last	product	quantity	sales_amt		Unique product names		
2	1	Jim	Halpert	Copy Paper	10	$99.90		Sticky Notes		
3	2	Pam	Halpert	Sticky Notes	5	$12.45		File folders		
4	3	Andy	Bernard	Printer Ink	2	$39.98		Rubber bands		
5	4	Stanley	Hudson	Envelopes	15	$149.85				
6	5	Jim	Halpert	Legal Pads	3	$14.97		Count of unique values:	3	
7	6	Pam	Halpert	Copy Paper	8	$79.92				
8	7	Andy	Bernard	File folders	10	$24.90				
9	8	Phyllis	Vance	Printer Ink	5	$99.95				
10	9	Jim	Halpert	Envelopes	12	$119.88				
11	10	Pam	Halpert	Legal Pads	7	$17.43				
12	11	Andy	Bernard	Copy Paper	4	$39.96				
13	12	Jim	Halpert	Printer Ink	8	$79.92				
14	13	Phyllis	Vance	Envelopes	15	$74.85				
15	14	Andy	Bernard	Legal Pads	3	$59.97				
16	15	Stanley	Hudson	Rubber bands	60	$14.94				
17										

Figure 10-9. Aggregating a dynamic array with the spill operator

Although not extensively covered in this chapter, spill operators provide significant benefits for constructing dependent drop-down lists, dynamic charts, and various other features.

Filtering Records with FILTER()

Excel's traditional dropdowns for filtering data are intuitive, but have limitations. For instance, once they are applied, it's no longer possible to view the raw data in its entirety. It would be better to create a separate copy of the data and then apply filters to that copy, similar to how Power Query approaches data cleaning. Additionally,

defining complex filter logic rules across multiple columns can be tedious as well as repetitive.

To address these limitations, Excel introduced the `FILTER()` dynamic array function. `FILTER()` has three parameters, which are explained in detail in Table 10-2.

Table 10-2. `FILTER()` parameters

Parameter	Description
`array`	A required argument specifying the range or array of data that you want to filter.
`include`	A required argument specifying the filtering criteria or condition. It defines the values that should be included in the filtered result. This can be a logical expression, a value to match, or a formula that evaluates to `TRUE` or `FALSE` for each element in the array.
`[if_empty]`	An optional argument specifying the value to return if the filtered result is empty. By default, if no values meet the filtering criteria, the function returns an array with a `#CALC` error values.

To achieve the desired result, the `dm_sales` table needs to be filtered so that only the records with `product` set to Sticky Notes are returned, as shown in Figure 10-10.

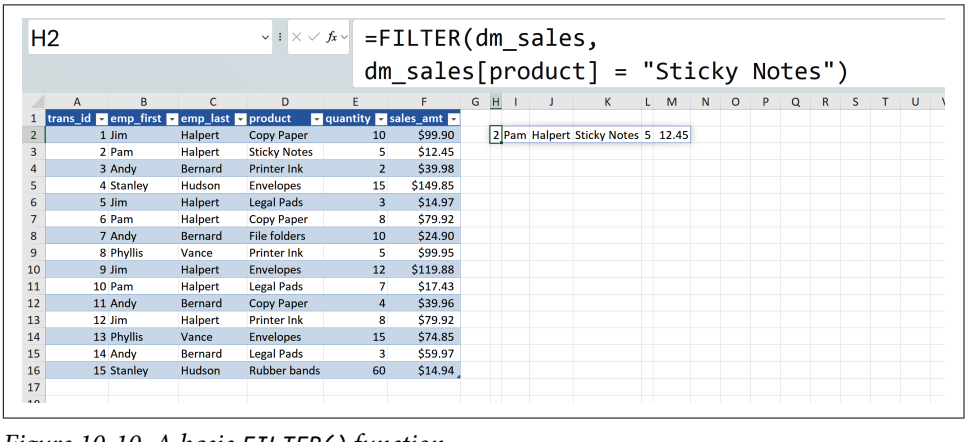
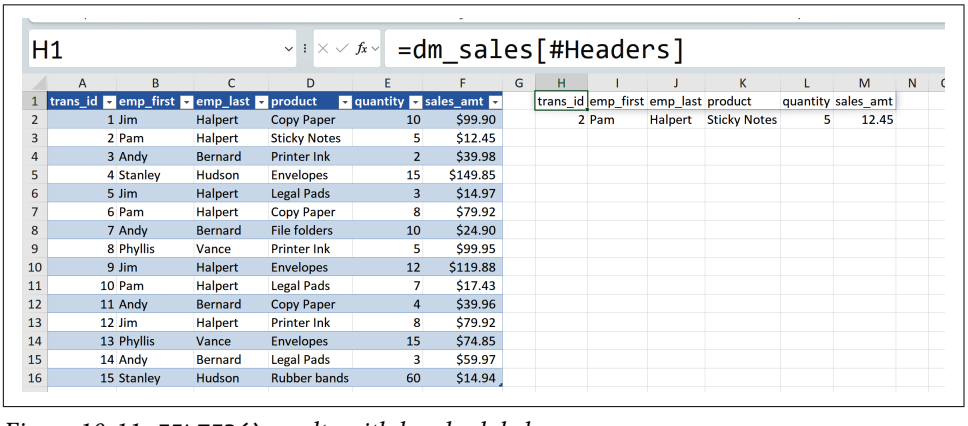


Figure 10-10. A basic `FILTER()` function

The `FILTER()` function is case insensitive by default. In the previous example, both "Sticky Notes" and "sticky notes" would yield the same result. To perform a case-sensitive filter, combine the `FILTER()` function with `EXACT()`; for example: `=FILTER(dm_sales, EXACT(dm_sales[product], "Sticky Notes"))`.

Adding a Header Column

The `FILTER()` function is already useful, but it lacks one crucial feature— it only returns matching rows, not the data's header columns. To include these headers, it's necessary to use a dynamic table header reference. For a quick recap on structured references, refer to Chapter 1. Include this reference above your filter output for dynamic header labels, as shown in Figure 10-11.



trans_id	emp_first	emp_last	product	quantity	sales_amt
1	Jim	Halpert	Copy Paper	10	$99.90
2	Pam	Halpert	Sticky Notes	5	$12.45
3	Andy	Bernard	Printer Ink	2	$39.98
4	Stanley	Hudson	Envelopes	15	$149.85
5	Jim	Halpert	Legal Pads	3	$14.97
6	Pam	Halpert	Copy Paper	8	$79.92
7	Andy	Bernard	File folders	10	$24.90
8	Phyllis	Vance	Printer Ink	5	$99.95
9	Jim	Halpert	Envelopes	12	$119.88
10	Pam	Halpert	Legal Pads	7	$17.43
11	Andy	Bernard	Copy Paper	4	$39.96
12	Jim	Halpert	Printer Ink	8	$79.92
13	Phyllis	Vance	Envelopes	15	$74.85
14	Andy	Bernard	Legal Pads	3	$59.97
15	Stanley	Hudson	Rubber bands	60	$14.94

trans_id	emp_first	emp_last	product	quantity	sales_amt
2	Pam	Halpert	Sticky Notes	5	12.45

Figure 10-11. `FILTER()` results with header labels

Applying the `FILTER()` function or other dynamic array functions to an Excel table does not include the table headers in the results.

Filtering by Multiple Criteria

The `FILTER()` function distinguishes itself from traditional drop-down menu filtering methods through its ability to use formulas to filter data. This capability unlocks a realm of powerful opportunities, while still preserving an intuitive and straightforward approach for establishing and understanding criteria.

To incorporate multiple criteria into the `FILTER()` function, employ the `*` symbol for `AND` statements and the `+` symbol for `OR` statements.

AND criteria

To search for records where the `product` is "Copy Paper" *and* the quantity is greater than 5, you can combine the criteria by multiplying them within separate sets of parentheses:

```
=FILTER(dm_sales, (dm_sales[product] = "Copy Paper") *
(dm_sales[quantity] > 5))
```

OR criteria

If you'd rather find records that meet *either* of this conditions, replace the `*` symbol with the `+` symbol to create an `OR` statement:

```
=FILTER(dm_sales, (dm_sales[product] = "Copy Paper") +
(dm_sales[quantity] > 5))
```

Nested AND/OR criteria

To create a filter function with nested `AND` or `OR` statements, group the statements using parentheses. This filter function includes records where the `sales_amt` is at least $100, or `quantity` is at least 10 *and* `product` is "Envelopes":

```
=FILTER(dm_sales,
(dm_sales[sales_amt] >= 100) +
((dm_sales[quantity] >= 10) * (dm_sales[product] = "Envelopes")))
```

You can continue adding and adjusting multiple criteria to `FILTER()` functions by following these guidelines.

Sorting Records with SORTBY()

`SORTBY()` is a dynamic array function that enables sorting of records based on multiple criteria simultaneously, featuring syntax similar to that of `SUMIFS()`. Table 10-3 walks through the parameters.

Table 10-3. `SORTBY()` parameters

Parameter	Description
`array`	A required argument specifying the array or range to sort.
`by_array1`	A required argument specifying the array or range to sort on.
`[sort_order1]`	An optional argument specifying how to sort the results. 1 sorts ascendingly, -1 sorts descendingly. The default is ascending.
`[by_array2]`	An optional argument specifying the array or range to sort on.
`[sort_order2]`	An optional argument specifying how to sort the results of `by_array2`. 1 sorts ascendingly, -1 sorts descendingly. The default is ascending.

To take an example, the dataset can be sorted in descending order based on `sales_amt` using the `SORTBY()` function, as shown in Figure 10-12.

H2 =SORTBY(dm_sales, dm_sales[sales_amt], -1)

trans_id	emp_first	emp_last	product	quantity	sales_amt
1	Jim	Halpert	Copy Paper	10	$99.90
2	Pam	Beesly	Sticky Notes	5	$12.45
3	Andy	Bernard	Printer Ink	2	$39.98
4	Stanley	Hudson	Envelopes	15	$149.85
5	Jim	Halpert	Legal Pads	3	$14.97
6	Pam	Beesly	Copy Paper	8	$79.92
7	Andy	Bernard	Sticky Notes	10	$24.90
8	Phyllis	Vance	Printer Ink	5	$99.95
9	Jim	Halpert	Envelopes	12	$119.88
10	Pam	Beesly	Legal Pads	7	$17.43
11	Andy	Bernard	Copy Paper	4	$39.96
12	Jim	Halpert	Printer Ink	8	$79.92
13	Phyllis	Vance	Envelopes	15	$74.85
14	Andy	Bernard	Legal Pads	3	$59.97
15	Stanley	Hudson	Rubber bands	60	$14.94

trans_id	emp_first	emp_last	product	quantity	sales_amt
4	Stanley	Hudson	Envelopes	15	149.85
9	Jim	Halpert	Envelopes	12	119.88
8	Phyllis	Vance	Printer Ink	5	99.95
1	Jim	Halpert	Copy Paper	10	99.9
6	Pam	Beesly	Copy Paper	8	79.92
12	Jim	Halpert	Printer Ink	8	79.92
13	Phyllis	Vance	Envelopes	15	74.85
14	Andy	Bernard	Legal Pads	3	59.97
3	Andy	Bernard	Printer Ink	2	39.98
11	Andy	Bernard	Copy Paper	4	39.96
7	Andy	Bernard	Sticky Notes	10	24.9
10	Pam	Beesly	Legal Pads	7	17.43
5	Jim	Halpert	Legal Pads	3	14.97
15	Stanley	Hudson	Rubber bands	60	14.94
2	Pam	Beesly	Sticky Notes	5	12.45

Figure 10-12. Sorting an Excel table with `SORTBY()`

Sorting by Multiple Criteria

`SORTBY()` enables sorting the data based on multiple criteria, with the flexibility to specify an ascending or descending order for each. For example, the data can be sorted in descending order by `emp_last` and in ascending order by `product`:

```
=SORTBY(dm_sales, dm_sales[emp_last], -1, dm_sales[product], 1)
```

You can extend this pattern to sort the dataset by even more criteria in the desired order.

Sorting by Another Column Without Printing It

`SORTBY()` can even sort a range by another range, even without including the original sort range in the results.

For example, say you want to obtain a list of transaction IDs sorted in descending order by sales. Instead of using the entire `dm_sales` table as the first argument, only select the `trans_id` column. The subsequent steps should be familiar. The result will consist of a single column, as shown in Figure 10-13.

H2

```
=SORTBY(dm_sales[trans_id],
dm_sales[sales_amt], -1)
```

	A	B	C	D	E	F	G	H	I	J
1	trans_id	emp_first	emp_last	product	quantity	sales_amt				
2	1	Jim	Halpert	Copy Paper	10	$99.90		4		
3	2	Pam	Halpert	Sticky Notes	5	$12.45		9		
4	3	Andy	Bernard	Printer Ink	2	$39.98		8		
5	4	Stanley	Hudson	Envelopes	15	$149.85		1		
6	5	Jim	Halpert	Legal Pads	3	$14.97		6		
7	6	Pam	Halpert	Copy Paper	8	$79.92		12		
8	7	Andy	Bernard	File folders	10	$24.90		13		
9	8	Phyllis	Vance	Printer Ink	5	$99.95		14		
10	9	Jim	Halpert	Envelopes	12	$119.88		3		
11	10	Pam	Halpert	Legal Pads	7	$17.43		11		
12	11	Andy	Bernard	Copy Paper	4	$39.96		7		
13	12	Jim	Halpert	Printer Ink	8	$79.92		10		
14	13	Phyllis	Vance	Envelopes	15	$74.85		5		
15	14	Andy	Bernard	Legal Pads	3	$59.97		15		
16	15	Stanley	Hudson	Rubber bands	60	$14.94		2		

Figure 10-13. Displaying `SORTBY()` results in one column

Creating Modern Lookups with XLOOKUP()

Up to this point, the examples demonstrating dynamic array functions have used a single table. Yet, it's often the case that data originates from several tables, requiring consolidation. Although Power Query and Power Pivot offer their own techniques for combining data from diverse sources, the immediate, dynamic, and interactive qualities of Excel formulas remain advantageous for tasks such as constructing models based on user input, performing real-time data analysis, and more.

The `XLOOKUP()` function presents a versatile alternative to the conventional `VLOOKUP()` function by exploiting the capabilities of dynamic arrays.

To explore this function further, refer to the `xlookup` worksheet in *ch_10.xlsx*. This worksheet contains three distinct tables associated with office supply sales.

XLOOKUP() Versus VLOOKUP()

`XLOOKUP()` offers a familiar experience for users accustomed to `VLOOKUP()` for retrieving data from one table and transferring it to another based on a shared lookup value. However, it introduces a range of additional search methods that are more versatile and sophisticated. For an overview of the key differences between `VLOOKUP()` and `XLOOKUP()`, refer to Table 10-4.

Table 10-4. `VLOOKUP()` versus `XLOOKUP()`

Feature	`VLOOKUP()`	`XLOOKUP()`
Search direction	Can only search vertically	Can search both vertically and horizontally
Return direction	Can only return values to the right of the lookup value	Can return values from columns to the left and right of the lookup value
Error handling	Returns #N/A if the value is not found	Can specify default value for unmatched items and handle errors

`XLOOKUP()` has six parameters in total, as shown in Table 10-5.

Table 10-5. `XLOOKUP()` parameters

Parameter	Description
`lookup_value`	Required argument specifying the value to search for in the `lookup_array`.
`lookup_array`	Required argument specifying the range or array to search for the `lookup_value`.
`return_array`	Required argument specifying the range or array from which to retrieve the data.
[`if_not_found`]	Optional argument specifying the value to return if the `lookup_value` is not found.
[`match_mode`]	Optional argument specifying the method for matching the `lookup_value`.
[`search_mode`]	Optional argument specifying the search behavior for finding the `lookup_value`.

This demonstration highlights the first four parameters of `XLOOKUP()`. For a more in-depth overview, check out Chapter 12 of Alan Murray's *Advanced Excel Formulas: Unleashing Brilliance with Excel Formulas* (Apress, 2022).

A Basic XLOOKUP()

Begin with a straightforward example: the `transactions` table includes a `product_id` that requires matching with its corresponding `product_name`. Here, the `product_id` serves as the lookup array and `product_name` serves as the return array, as shown in Figure 10-14.

XLOOKUP =XLOOKUP([@[product_id]], products[product_id], products[product_name])

trans_id	trans_date	branch_id	product_id	quantity	total_price	product_name
1	5/1/2023	1	1	10	$99.90	uct_name])
2	5/2/2023	1	2	5	$12.45	Sticky Notes
3	5/3/2023	2	1	20	$199.80	Copy Paper
4	5/4/2023	3	3	2	$39.98	Printer Ink
5	5/5/2023	1	99	15	$149.85	#N/A
6	5/5/2023	2	5	3	$14.97	Legal Pads
7	5/6/2023	2	2	10	$24.90	Sticky Notes
8	5/7/2023	1	4	8	$55.92	Envelopes
9	5/8/2023	3	3	5	$99.95	Printer Ink
10	5/8/2023	3	1	12	$119.88	Copy Paper
11	5/9/2023	1	2	7	$17.43	Sticky Notes
12	5/10/2023	2	4	3	$20.97	Envelopes
13	5/10/2023	1	5	10	$49.90	Legal Pads
14	5/11/2023	2	99	4	$79.96	#N/A
15	5/12/2023	3	2	6	$14.94	Sticky Notes
16	5/12/2023	1	4	5	$34.95	Envelopes
17	5/13/2023	2	1	8	$79.92	Copy Paper
18	5/14/2023	3	5	15	$74.85	Legal Pads
19	5/15/2023	1	3	3	$59.97	Printer Ink
20	5/15/2023	2	4	10	$69.90	Envelopes

branch_name	branch_id
Scranton	1
Stamford	2
Nashua	3

product_id	product_name	product_price
1	Copy Paper	$9.99
2	Sticky Notes	$2.49
3	Printer Ink	$19.99
4	Envelopes	$6.99
5	Legal Pads	$4.99

Figure 10-14. A basic `XLOOKUP()`

XLOOKUP() and Error Handling

Lookups for the `product_id` of *99* generate an error. Using `#N/A` as the result for a missing match is problematic. It can introduce calculation errors and create confusion for users who may not understand why the `#N/A` is being returned.

To customize the error message in the `XLOOKUP()` statement, specify the fourth, optional parameter. In this specific case, you discovered that products assigned to the number *99* should be labeled as "Other." The results are demonstrated in Figure 10-15.

XLOOKUP =XLOOKUP([@[product_id]], products[product_id], products[product_name], "Other")

trans_id	trans_date	branch_id	product_id	quantity	total_price	product_name
1	5/1/2023	1	1	10	$99.90	"Other")
2	5/2/2023	1	2	5	$12.45	Sticky Notes
3	5/3/2023	2	1	20	$199.80	Copy Paper
4	5/4/2023	3	3	2	$39.98	Printer Ink
5	5/5/2023	1	99	15	$149.85	Other
6	5/5/2023	2	5	3	$14.97	Legal Pads
7	5/6/2023	2	2	10	$24.90	Sticky Notes
8	5/7/2023	1	4	8	$55.92	Envelopes
9	5/8/2023	3	3	5	$99.95	Printer Ink
10	5/8/2023	3	1	12	$119.88	Copy Paper
11	5/9/2023	1	2	7	$17.43	Sticky Notes
12	5/10/2023	2	4	3	$20.97	Envelopes
13	5/10/2023	1	5	10	$49.90	Legal Pads
14	5/11/2023	2	99	4	$79.96	Other
15	5/12/2023	3	2	6	$14.94	Sticky Notes
16	5/12/2023	1	4	5	$34.95	Envelopes
17	5/13/2023	2	1	8	$79.92	Copy Paper
18	5/14/2023	3	5	15	$74.85	Legal Pads
19	5/15/2023	1	3	3	$59.97	Printer Ink
20	5/15/2023	2	4	10	$69.90	Envelopes

branch_name	branch_id
Scranton	1
Stamford	2
Nashua	3

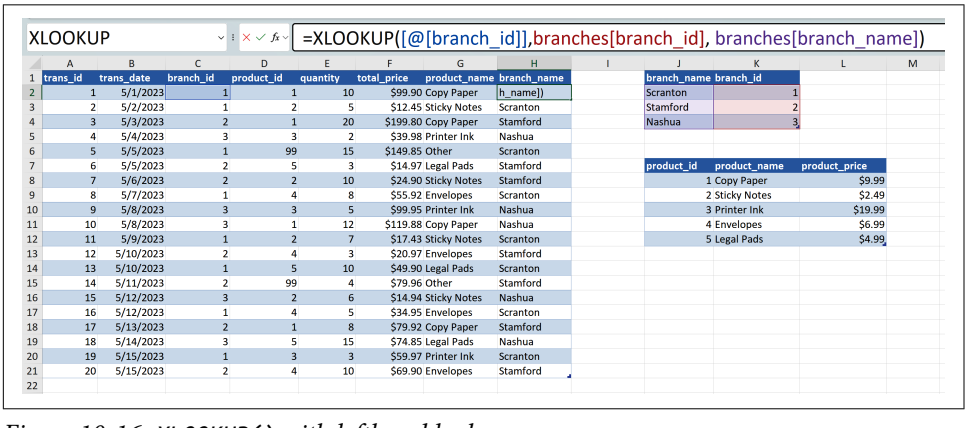
product_id	product_name	product_price
1	Copy Paper	$9.99
2	Sticky Notes	$2.49
3	Printer Ink	$19.99
4	Envelopes	$6.99
5	Legal Pads	$4.99

Figure 10-15. `XLOOKUP()` with error handling

After looking up product names in the table, it's time to do the same for branch names.

XLOOKUP() and Looking Up to the Left

A common criticism of `VLOOKUP()` is its inability to search to the left of the lookup array, unless helper functions are used. On the other hand, `XLOOKUP()` can search through values in any Excel range, including a table column to the left of the lookup value. An example of this is shown in Figure 10-16.



trans_id	trans_date	branch_id	product_id	quantity	total_price	product_name	branch_name
1	5/1/2023	1	1	10	$99.90	Copy Paper	h_name])
2	5/2/2023	1	2	5	$12.45	Sticky Notes	Scranton
3	5/3/2023	2	1	20	$199.80	Copy Paper	Stamford
4	5/4/2023	3	3	2	$39.98	Printer Ink	Nashua
5	5/5/2023	1	99	15	$149.85	Other	Scranton
6	5/5/2023	2	5	3	$14.97	Legal Pads	Stamford
7	5/6/2023	2	2	10	$24.90	Sticky Notes	Stamford
8	5/7/2023	1	4	8	$55.92	Envelopes	Scranton
9	5/8/2023	3	3	5	$99.95	Printer Ink	Nashua
10	5/8/2023	3	1	12	$119.88	Copy Paper	Nashua
11	5/9/2023	1	2	7	$17.43	Sticky Notes	Scranton
12	5/10/2023	2	4	3	$20.97	Envelopes	Stamford
13	5/10/2023	1	5	10	$49.90	Legal Pads	Scranton
14	5/11/2023	2	99	4	$79.96	Other	Stamford
15	5/12/2023	3	2	6	$14.94	Sticky Notes	Nashua
16	5/12/2023	1	4	5	$34.95	Envelopes	Scranton
17	5/13/2023	2	1	8	$79.92	Copy Paper	Stamford
18	5/14/2023	3	5	15	$74.85	Legal Pads	Nashua
19	5/15/2023	1	3	3	$59.97	Printer Ink	Scranton
20	5/15/2023	2	4	10	$69.90	Envelopes	Stamford

branch_name	branch_id
Scranton	1
Stamford	2
Nashua	3

product_id	product_name	product_price
1	Copy Paper	$9.99
2	Sticky Notes	$2.49
3	Printer Ink	$19.99
4	Envelopes	$6.99
5	Legal Pads	$4.99

Figure 10-16. `XLOOKUP()` with lefthand lookup

Thanks to its versatility in searching both vertically and horizontally, retrieving values from columns on either side of the matched lookup value, and handling errors within its formula, `XLOOKUP()` has emerged as the preferred formula for data retrieval in Excel.

Other Dynamic Array Functions

The dynamic array functions showcased here were among Excel's initial offerings, with subsequent additions expanding the range of capabilities.

For instance, the `RANDARRAY()` function produces an array of random numbers, enabling you to specify the number of rows and columns. This facilitates the generation of dynamic arrays filled with random values, ideal for simulations. Similarly, the `SEQUENCE()` function creates a sequence of numbers within an array, using a specified starting number, increment, and array size. This is particularly beneficial for generating linearly spaced values or time steps in simulations and dynamic models.

Many other dynamic array functions are geared towards text manipulation, including `VSTACK()` for vertical array combination and `TEXTSPLIT()` for splitting text using a specified delimiter. To explore a comprehensive list of dynamic array functions and access tutorials, visit Exceljet.com's article on the topic.

Dynamic Arrays and Modern Excel

Dynamic array functions may appear to be a step backward in Excel, given the availability of tools like Power Query and Power Pivot. Why willingly return to the days of delicate, formula-driven workbooks when these advanced features exist? This attitude overlooks the value that dynamic arrays bring to the modern Excel analytics stack. Here's why they are an important component:

Simplicity
: Dynamic array functions streamline data manipulation and analysis by enabling calculations within a single formula, enhancing comprehension and maintainability. This stands in contrast to the complex and multistep process involved in constructing and managing data cleaning tasks in Power Query or Data Models in Power Pivot.

Familiarity
: Dynamic array functions distinguish themselves from many of the other tools discussed in this book through their integration within the familiar Excel environment. Unlike add-ins that require installation or separate editors, dynamic array formulas are readily available within Excel itself. They are easily accessible, making adoption significantly smoother for the typical user.

Real-time updates
: Dynamic array functions offer the advantage of automatic result updates whenever the underlying data changes. This eliminates the need for manual formula recalculations or connection refreshes, and enables real-time analysis and insights. This functionality proves especially beneficial in dynamic scenarios where data is continuously evolving, such as real-time dashboards or financial models.

Conclusion

The chapter introduced dynamic array functions, revitalizing traditional, sometimes-clunky Excel references and formulas. These features now hold a vital place alongside Power Query and Power Pivot in the Excel analytics toolkit.

While dynamic array functions offer simplicity and low overhead, the subsequent chapters in Part III delve into more advanced tools. These tools require additional setup but offer intricate analytics insights that surpass what can be achieved through formulas alone. Throughout these chapters, you'll learn how to incorporate artificial intelligence, machine learning techniques, and advanced automation features using Python into your Excel workflow.

Exercises

To practice dynamic array functions, open the *ch_10_exercises.xlsx* file found in the *exercises\ch_10_exercises* folder in the book's companion repository. This workbook includes two datasets: `vehicles` and `common`. Complete the following exercises:

1. Find the distinct and truly unique values in the `make` column of the `vehicles` dataset. How many are there of each?
2. Display only the vehicles with city mileage greater than 30.
3. Display only the vehicles where either the city mileage is greater than 30, or where both cylinders are less than 6 and fuel is Regular.
4. Sort the `vehicles` dataset in descending order based on the highway mileage.
5. Sort just the `model` column of the `common` dataset based on the `years` column, descending.
6. Add the `years` column from the `common` dataset to the `vehicles` dataset. Return `Not reported` if a match is not found.

The solutions can be found in *ch_10_solutions.xlsx* in the same folder.

CHAPTER 11

Augmented Analytics and the Future of Excel

As the world of data analytics grows vaster and more complex, what role will Excel serve? Will it become obsolete in an AI-powered data ecosystem? This chapter explores the emergence of augmented analytics and the role Excel will play in this transformation, along with some current use cases.

Before we dive into the intriguing world of predictive analytics, AI, and the transformative changes unfolding across business, including within our spreadsheets, it's important to acknowledge the dynamic nature of this field. New product developments are emerging constantly. Even well-established tools, such as ChatGPT and Microsoft Copilot, frequently undergo significant updates and changes. This chapter aims to focus on understanding the fundamental and more stable aspects of Excel. My intention is not to provide an exhaustive overview of the latest advancements but to offer a glimpse into what augmented analytics in Excel entails and to equip you with the skills needed to navigate this area, regardless of how the features and tools evolve.

The Growing Complexity of Data and Analytics

In 2017, International Data Corporation (IDC), a market intelligence firm, predicted a remarkable tenfold increase in the amount of data in existence from 2016 to 2025, totaling 163 zettabytes or a trillion gigabytes.

With the overall quantity of data on the rise, it's also expanding in variety. According to AI services provider Taiger, 80% of digital data was unstructured by 2020—a figure that has likely grown with the emergence of generative natural language processing products like ChatGPT. Unstructured data is information that doesn't follow a

specific format or structure, making it difficult to organize and analyze in traditional databases or spreadsheets like Excel. Examples of unstructured data include text, images, videos, and social media posts, all of which require more complex processing techniques to extract useful information.

In addition, real-time data has gained significant importance, with IDC estimating that streaming data will comprise 30% of all data by 2025.

This explosion of data, characterized by research advisory firm Gartner as volume, velocity, and variety, has necessitated the use of advanced analytics methods. Data science has helped organizations uncover relationships and insights in the data using a variety of computational and statistical methods, while machine learning and artificial intelligence enable computers to learn and simulate human intelligence. These techniques have allowed businesses to automate decision-making processes, identify trends in real time, and create personalized experiences.

This revolution is here to stay: 94% of business leaders responded to a survey by Deloitte that AI is critical to success over the next five years, and the Bureau of Labor Statistics has projected a 36% increase in the number of data scientists employed over the next decade, from 113,000 in 2021.

Excel and the Legacy of Self-Service BI

Self-service business intelligence (BI), enabled by tools like Excel, has revolutionized decision making for businesses. It enables individual users to access and analyze data independently, without relying on assistance from information technology (IT) staff. However, self-service BI is limited in scope and sophistication. Data must be structured in a way that is compatible with Excel, limiting the analysis to descriptive and diagnostic analytics. This means that Excel is not equipped to handle advanced algorithms and machine learning models required for predictive or prescriptive analytics.

To make more strategic decisions, businesses need to complement self-service BI with more advanced analytics tools and techniques like data mining, machine learning, and AI.

Excel for Augmented Analytics

Augmented analytics is an approach that utilizes AI and machine learning technologies to enhance data analytics processes. It automates the insights generation process by sifting through large datasets, identifying trends, patterns, and anomalies without requiring manual intervention. This method significantly improves the efficiency and accuracy of data analysis and self-service BI, making it easier for businesses and individuals to make informed decisions based on data insights.

In the remainder of this chapter, you'll get a hands-on look at some of the use cases for augmented analytics in Excel as they exist today. First, you'll learn how to make the most of Analyze Data for AI-powered insights. Then, you'll build a basic predictive model using XLMiner. Finally, you'll use optical character recognition and Azure Machine Learning to perform sentiment analysis. These examples aim to expand your perception of Excel's capabilities and solidify its promising future in the realm of augmented analytics.

Using Analyze Data for AI Powered Insights

The Analyze Data feature in Excel is an augmented analytics product that uses AI to derive meaningful insights more efficiently. That said, AI is not a complete substitute for genuine expertise. To fully leverage Excel's potential with AI-generated insights through Analyze Data, properly structured data is essential.

The starter file for this demo can be found in the *ch_11* folder of the book's resources as *ch_11.xlsx*.

Analyze Data is ready to use in Excel with no downloads required. Simply position your cursor over the somewhat familiar `wholesale_customers` table located in the first worksheet of your workbook, and navigate to Home → Analyze Data to begin. Immediately, you will be provided with a variety of fascinating AI-generated insights, as shown in Figure 11-1. Select any insight to have it incorporated into your workbook directly.

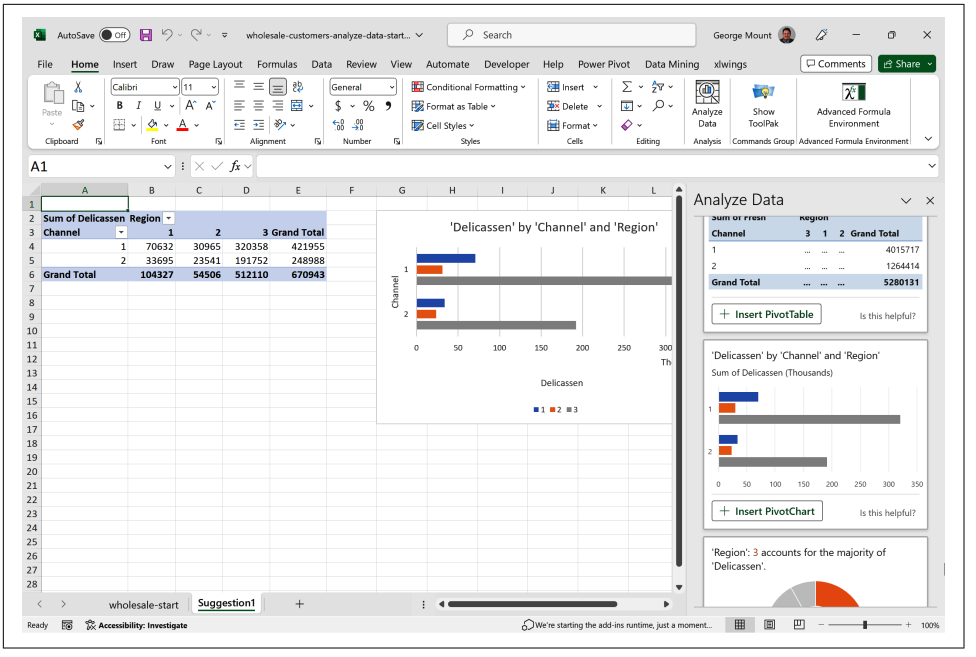


Figure 11-1. Inserting an Analyze Data insight

Due to the probabilistic nature of AI-generated insights, like those produced by Analyze Data, the results you receive may differ from what is shown in this book. This underscores the importance of having a solid understanding of your domain and data when using AI, as you will need to interpret and navigate complex and dynamic results.

The power of Analyze Data becomes more apparent with its natural language querying. For instance, imagine you're in a meeting with colleagues and need to swiftly retrieve the total sales for the grocery department. Instead of spending time manually calculating the answer, you can directly pose the question to Analyze Data and obtain the desired information right away, as shown in Figure 11-2.

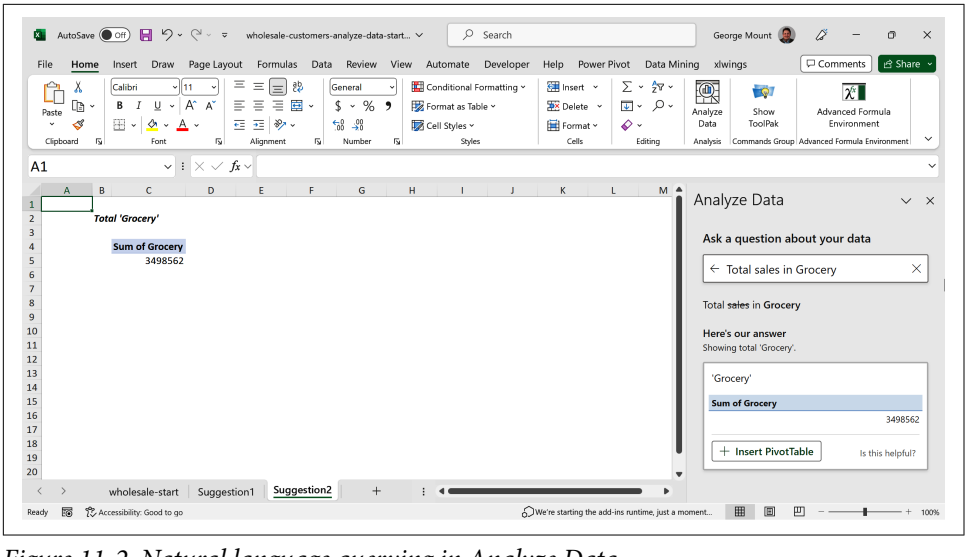


Figure 11-2. Natural language querying in Analyze Data

While querying this dataset is undoubtedly impressive, it does have certain limitations, primarily related to the data's layout. For instance, if you attempt to ask Analyze Data for the total sales by region, you will instead receive the sum of all region numbers, as in Figure 11-3.

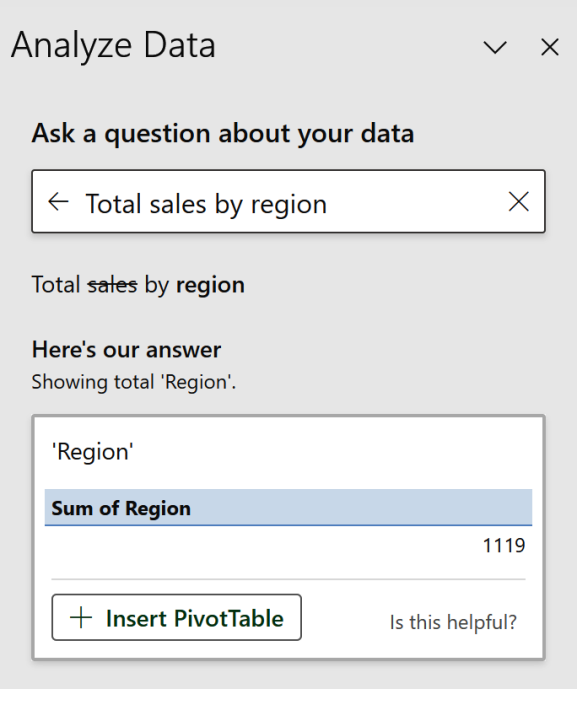
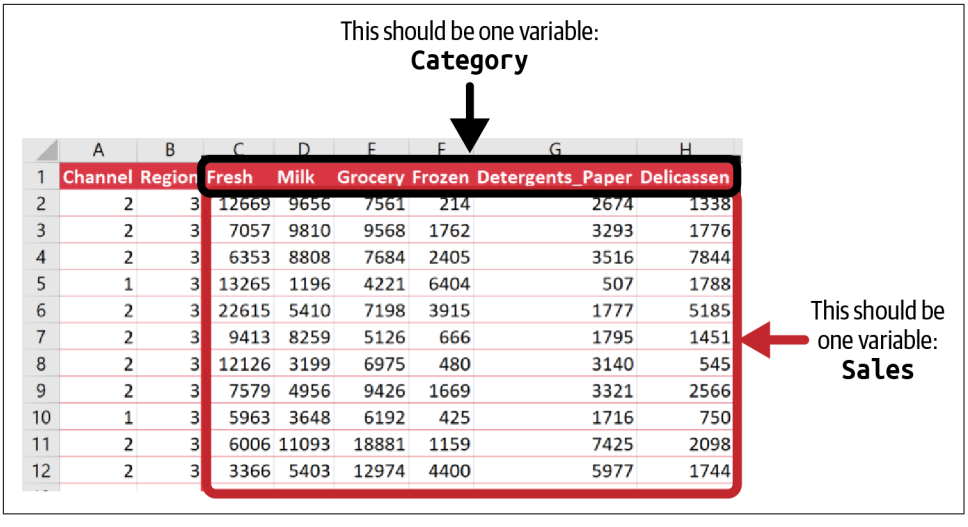


Figure 11-3. Analyze Data struggles with improperly constructed data

Analyze Data isn't sure what to do because the data is presented in an improper format. Instead of consolidating all sales figures in one column, they are spread across multiple columns. As a result, Analyze Data cannot determine which columns contain the relevant sales figures that need to be summed up. The gist of this formatting error is illustrated in Figure 11-4.



	A	B	C	D	E	F	G	H
1	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
2	2	3	12669	9656	7561	214	2674	1338
3	2	3	7057	9810	9568	1762	3293	1776
4	2	3	6353	8808	7684	2405	3516	7844
5	1	3	13265	1196	4221	6404	507	1788
6	2	3	22615	5410	7198	3915	1777	5185
7	2	3	9413	8259	5126	666	1795	1451
8	2	3	12126	3199	6975	480	3140	545
9	2	3	7579	4956	9426	1669	3321	2566
10	1	3	5963	3648	6192	425	1716	750
11	2	3	6006	11093	18881	1159	7425	2098
12	2	3	3366	5403	12974	4400	5977	1744

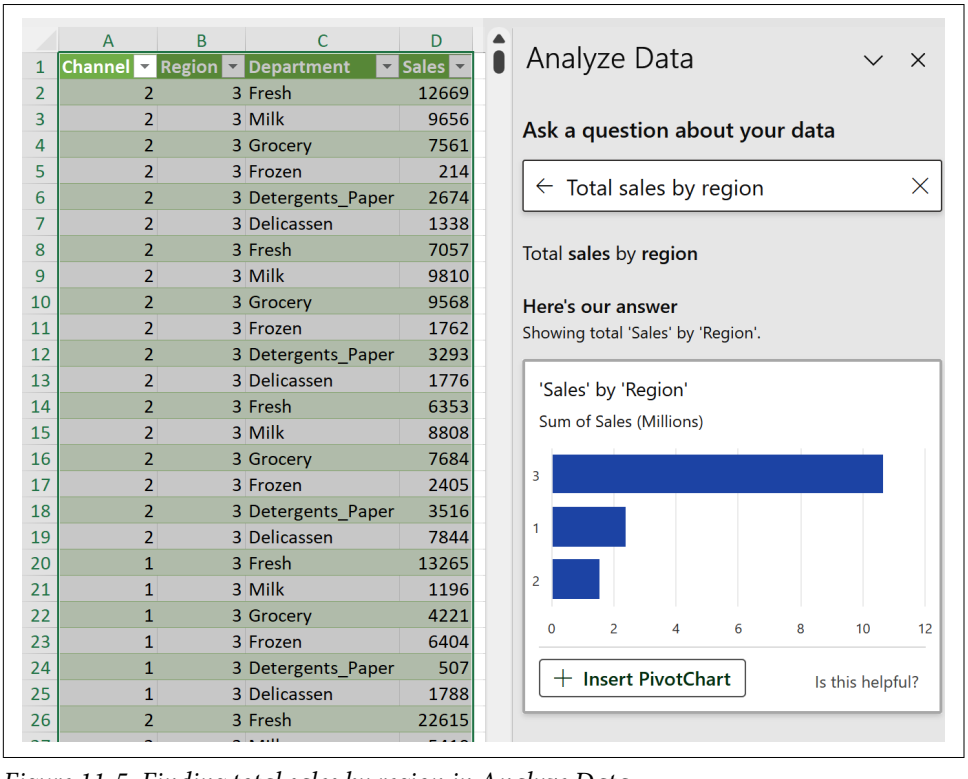
Figure 11-4. How to tidy this dataset for better insights

Storing data in an unclean or "untidy" format is a significant obstacle in analytics. You may have encountered this issue in your own work, without putting your finger on exactly what was wrong. Developing a conceptual understanding of dirty data allows you to identify issues early on in your project, saving significant amounts of time later on. To delve deeper into the theory of tidy data and learn how to handle it effectively, refer back to Chapter 1.

For AI to unleash its full potential in uncovering insights, data must be in a machine-readable, tidy format where each variable resides in a single column. To address this, we will use Power Query to unpivot the columns from `Fresh` to `Delicassen`[1] and rename them `Department` and `Sales`. Make sure to load the results of your query into an Excel table. For a refresher on how to unpivot and load this dataset in Power Query, refer back to Chapter 4.

1 This is the spelling used in the original dataset.

With the data in a tidy format, querying the data to find total sales by region is a breeze, as shown in Figure 11-5.



	A	B	C	D
1	Channel	Region	Department	Sales
2	2	3	Fresh	12669
3	2	3	Milk	9656
4	2	3	Grocery	7561
5	2	3	Frozen	214
6	2	3	Detergents_Paper	2674
7	2	3	Delicassen	1338
8	2	3	Fresh	7057
9	2	3	Milk	9810
10	2	3	Grocery	9568
11	2	3	Frozen	1762
12	2	3	Detergents_Paper	3293
13	2	3	Delicassen	1776
14	2	3	Fresh	6353
15	2	3	Milk	8808
16	2	3	Grocery	7684
17	2	3	Frozen	2405
18	2	3	Detergents_Paper	3516
19	2	3	Delicassen	7844
20	1	3	Fresh	13265
21	1	3	Milk	1196
22	1	3	Grocery	4221
23	1	3	Frozen	6404
24	1	3	Detergents_Paper	507
25	1	3	Delicassen	1788
26	2	3	Fresh	22615

Figure 11-5. Finding total sales by region in Analyze Data

What other insights can you glean with the power of Analyze Data? To access the solution for this and the other demonstrations in this chapter, download the *ch_11_solutions.xlsx* file from the same folder in the book's companion repository.

Analyze Data is a powerful augmented analytics tool that uses AI to efficiently derive valuable insights. However, it is crucial to have properly structured data for optimal results. By understanding the concept of tidy data and resolving formatting issues, users can fully leverage the potential of AI in uncovering meaningful insights.

Building Statistical Models with XLMiner

The XLMiner add-in for Excel enhances analytics by offering key data analysis and modeling tools. It allows users to access advanced analytics within Excel, improving the augmented analytics experience. The demo's starter dataset is found in the `housing` worksheet of *ch_11.xlsx*.

To get started, head to the ribbon then Insert File → Add-ins → Get Add-ins. From the Office Add-ins dialog, search for XLMiner and click Add, as shown in Figure 11-6.

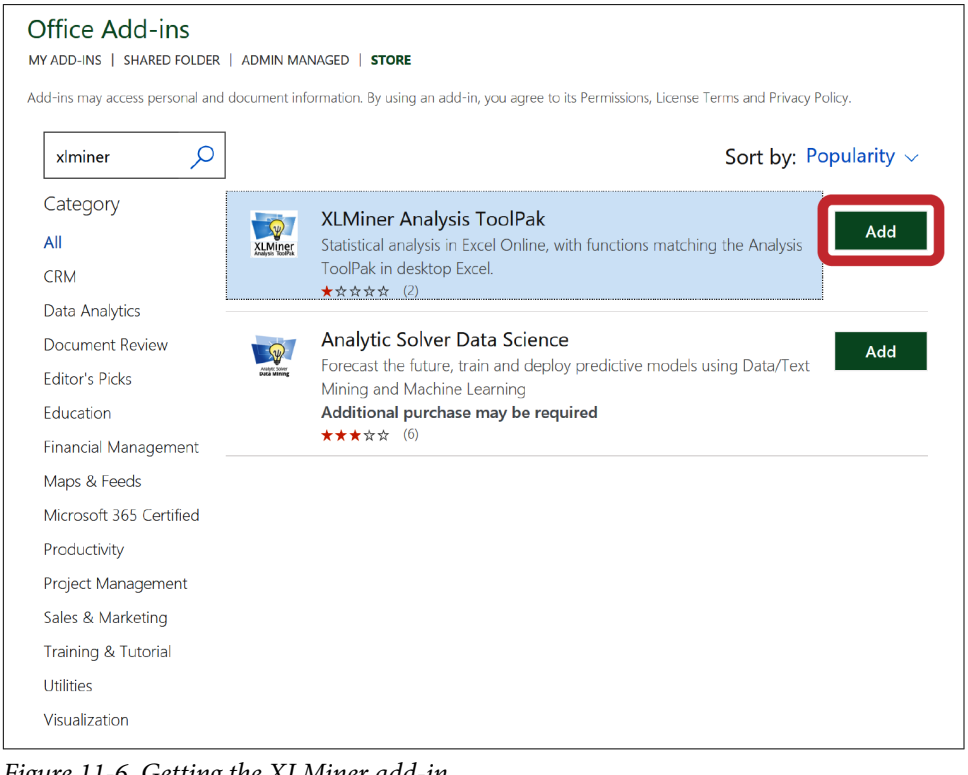


Figure 11-6. Getting the XLMiner add-in

Agree to the Terms & Conditions, click OK, and you should see the XLMiner add-in on the right side of your worksheet. As you're seeing, XLMiner comes with plenty of statistical tools and techniques. Let's focus on the "mother of all models," linear regression.

We will use `price` as the dependent variable and `lotsize`, `airco`, and `prefarea` as the independent variables. Head to the Linear Regression section of the XLMiner add-in, fill it out as shown in Figure 11-7, and then click OK.

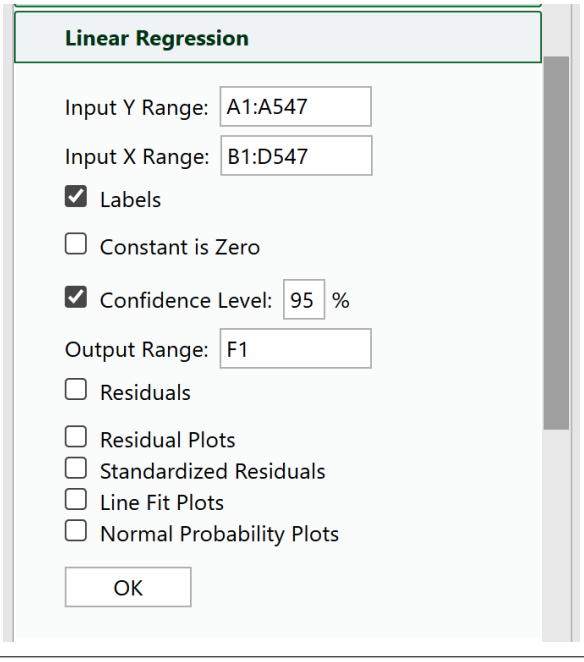


Figure 11-7. Setting up a linear regression in XLMiner

Navigating the drag-and-drop feature for naming input ranges in XLMiner can be somewhat challenging, and it often requires manually entering cell locations.

Before beginning to construct models and make predictions, it is crucial to conduct a thorough examination of the dataset to ensure that it aligns with the assumptions of the chosen model. While Python and R offer more extensive tools for analysis and testing, hands-on interaction with the data in Excel can prove to be advantageous as well. XLMiner acts as a meeting point, melding the straightforward data manipulation capabilities of Excel with fundamental aspects of the advanced analytical rigor typically exclusive to specialized data science tools.

You should see the output in Figure 11-8 from XLMiner after running the regression.

	E	F	G	H	I	J	K	L	M	N
6		Adjusted R Square	0.436467707							
7		Standard Error	20045.37142							
8		Observations	546							
9										
10		ANOVA								
11			df	SS	MS	F	Significance F			
12		Regression	3	1.70818E+11	56939339259	141.7046846	0			
13		Residual	542	2.17785E+11	401816915.2					
14		Total	545	3.88603E+11						
15										
16			Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95%	Upper 95%
17		Intercept	32770.49675	2216.879895	14.78226079	8.65278E-42	28415.76783	37125.22568	28415.76783	37125.22568
18		lotsize	5.116814473	0.415952537	12.3014383	7.6259E-31	4.299737937	5.933891009	4.299737937	5.933891009
19		airco	19437.2651	1895.231911	10.25587686	1.12388E-22	15714.36553	23160.16468	15714.36553	23160.16468
20		prefarea	12112.04184	2087.892931	5.80108379	1.12101E-08	8010.688509	16213.39516	8010.688509	16213.39516

Figure 11-8. Results of linear regression in XLMiner

Here you have typical regression diagnostics, such as coefficient p-values, R-square and more. If you'd like to learn more about interpreting these, check out my book *Advancing into Analytics: From Excel to Python and R* (O'Reilly, 2021).

XLMiner enhances Excel's data analysis capabilities by offering an accessible platform for statistical modeling, appealing to users with varying levels of expertise. Despite its user-friendly interface and seamless integration with Excel, XLMiner does not qualify as a comprehensive augmented analytics tool. This shortfall is primarily due to its inability to support real-time model deployment, lack of continuous learning for models to adapt over time, and insufficient support for advanced modeling techniques, such as neural networks.

Moreover, XLMiner's limited AI integration restricts its capacity to automate data analysis processes comprehensively. For tackling more sophisticated analytical tasks, users may need to explore more advanced tools available in R or Python ecosystems.

Reading Data from an Image

Analysts may face situations where their data is only available in printouts or other analog formats. To bypass the slow and error-prone manual data entry process, Excel provides a feature that allows for the direct conversion of text from images into a workbook.

Converting scanned paper documents into editable computer text files through a process known as optical character recognition (OCR) is not a novel concept. OCR technology has been in existence since the 1970s and has undergone significant advancements since then. Today, it is widely available in various programs, including Excel.

For this demonstration, we have customer reviews that exist only as printed copies. Our goal is to import them into Excel for sentiment analysis. You can find the file named *scanned_reviews.png* in the *ch_11* folder of the book's resources.

To get started, open a new Excel workbook and select Data → Get & Transform Data → From Picture → Picture from File from the ribbon. From here, you can navigate to and select the *scanned_reviews.png* exercise file. Import the file and you should see a Data from Picture menu to the right of your workbook, as shown in Figure 11-9.

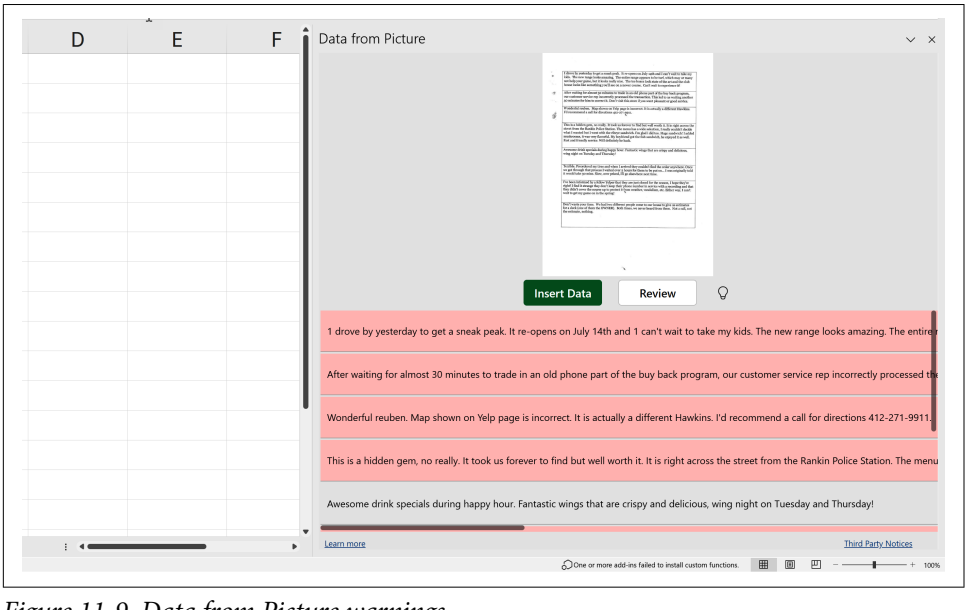


Figure 11-9. Data from Picture warnings

Excel's OCR feature converts images to text—seemingly magical, but it does make mistakes. Leveraging AI features, Excel can also predict where these mistakes might occur.

In this case, Excel has flagged all but one record as likely containing an error. You can click Review to scan and double-check each of them, then make any adjustments to the data. For example, the first entry starts with the letter 1 when it should be the pronoun I, as shown in Figure 11-10.

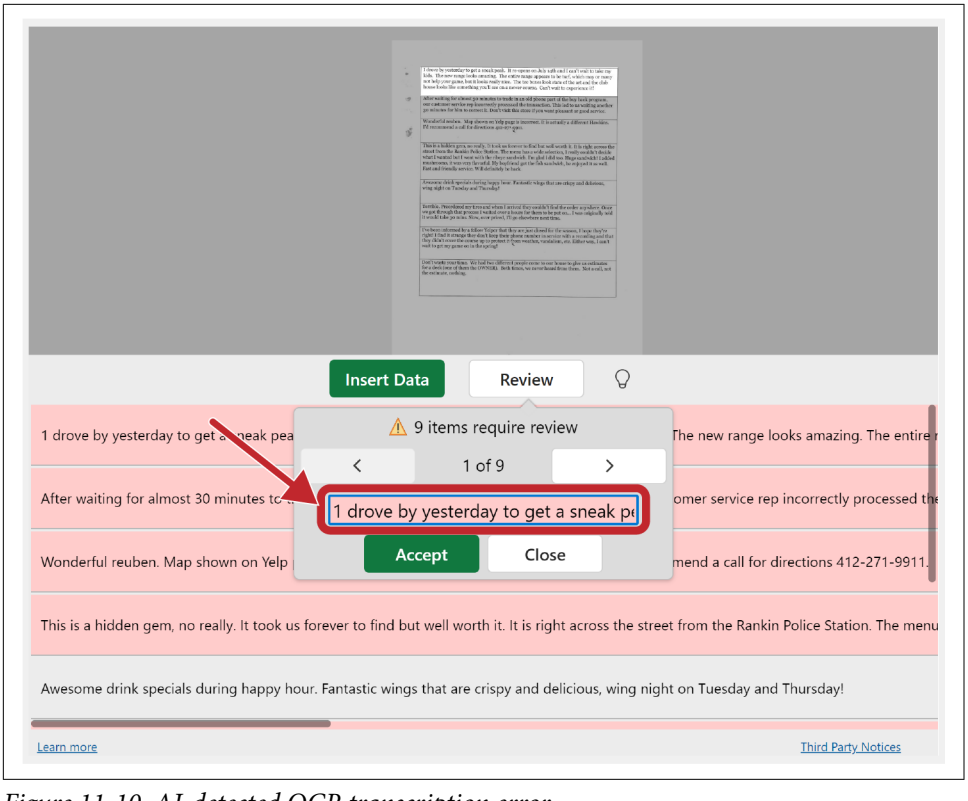


Figure 11-10. AI-detected OCR transcription error

After reviewing and identifying any potential erroneous entries, click the Insert Data button to transfer the results to Excel.

Excel's AI does a decent job predicting when text is likely to contain an error, but it's not perfect. For example, it might detect an error in an entry when none exists—known as a *false positive* in statistics. On the flipside, it might approve an entry that actually does have errors—a *false negative*.

Balancing the identification of potential false positives and false negatives represents a significant challenge in statistics and machine learning. For the time being, we'll rely on Excel's judgment, but as you progress in your analytics journey, you might encounter situations where making your own decisions is preferable.

Inserting unstructured data like text into Excel may pose certain challenges, as Excel is not inherently designed for such data. To maintain organization, it is recommended that you allocate a separate cell for each review and manually adjust accordingly. For instance, rows 6 and 7 can be combined to form a single review.

Although OCR technology has existed for some time, its integration into Excel proves exceptionally convenient. This feature is particularly beneficial for managing financial statements or similar numerical documents that one might wish to analyze in Excel.

The findings from this demonstration will lay the groundwork for our next one.

Sentiment Analysis with Azure Machine Learning

While Excel has traditionally been regarded as a tool suitable for working with small, structured datasets, the introduction of features related to AI and ML has blurred these conventional limitations. This highlights the immense potential of augmented analytics within Excel. A prime example of this is the ability to employ Excel for *sentiment analysis*, enabling the assessment of sentiment in a collection of text reviews.

Sentiment analysis is a data analysis tool that uses machine learning algorithms to decipher emotions and opinions in unstructured data. It often categorizes text as positive, negative, or neutral, enabling businesses to improve customer satisfaction and address concerns based on overall sentiment towards a brand, product, or service.

Manually assessing a few reviews isn't a problem, but it becomes challenging with thousands or more. Continuing with our analysis of the series of reviews imported from the image in the previous section, our current objective is to classify the sentiment of each review as positive, negative, or neutral. To automate this task, we will leverage Azure's text analytics features.

The first step involves integrating the Azure Machine Learning add-in into Excel. To do this, navigate to File → Get Add-ins. In the Office Add-ins dialog, search for "Azure Machine Learning" and click Add, followed by Continue, to install it. Upon completion, the Azure Machine Learning add-in should be visible on the right side of your Excel window. Proceed by selecting the second option, "Text Sentiment Analysis (Excel Add-in Solver)," as illustrated in Figure 11-11.

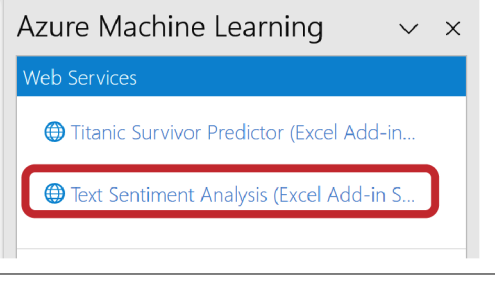


Figure 11-11. Selecting Azure sentiment analysis

Azure requires the input data for sentiment analysis to adhere to a specific format or *schema*. Here is an example where structuring data in a machine-friendly format is crucial for AI to work effectively.

For this particular task, we need to create three column headers in the workbook: `tweet_text`, `Sentiment`, and `Score`. These column names should exactly match what is found under the View Schema section of the Azure Machine Learning add-in.

The first column header, `tweet_text`, is where we place the restaurant reviews that were imported in the previous step. Despite its name, this column header can handle full reviews, not just tweets. The `Sentiment` and `Score` columns will be populated by Azure's sentiment analysis add-in, as shown in Figure 11-12.

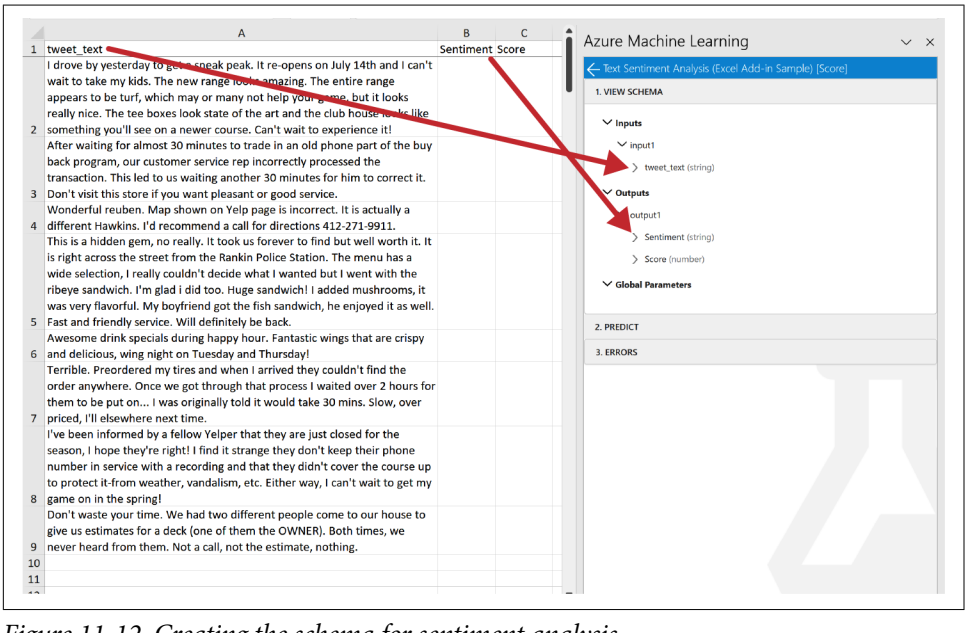


Figure 11-12. Creating the schema for sentiment analysis

To configure the input for sentiment analysis, head to the Predict section of the add-in and define the input area. This should cover cells `A1:A9`, which includes the header. Make sure to select the "My data has headers" option.

For the output area, the results of the sentiment analysis will be displayed starting from cell `B1`. Set this cell to confirm that the results will be populated in that location. After confirming your inputs resemble those of Figure 11-13, click Predict to generate the sentiment analysis.

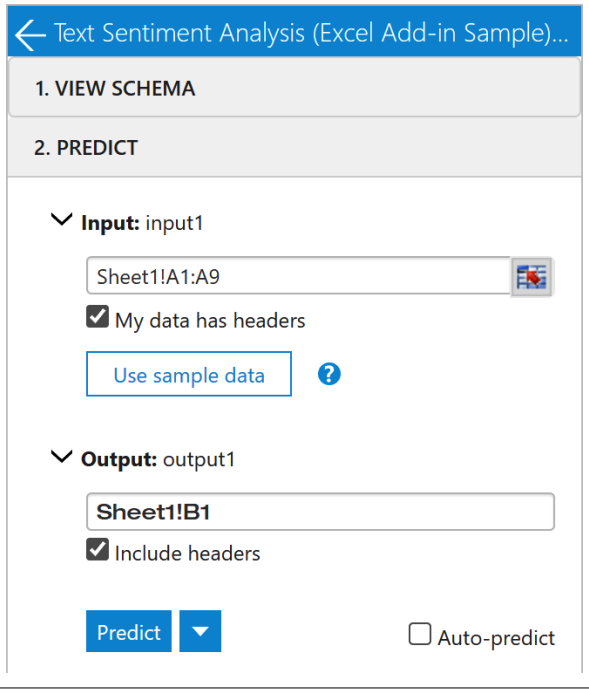


Figure 11-13. Defining the sentiment analysis inputs and outputs

After clicking on the Predict button, you should see columns `B` and `C` populated with results. Unfortunately, this process can be glitchy at times. If you encounter any issues, double-check the schema and inputs or try restarting Excel.

As anticipated, Azure has successfully classified each review as either positive, negative, or neutral, and the results are recorded in the `Sentiment` column. The `Score` column contains a numerical value ranging from 0 to 1, representing the sentiment score generated by Azure. A higher score indicates a more positive sentiment. These scores are subsequently categorized into negative, neutral, and positive groups.

If you're wondering how Azure generates these scores, it's through a complex machine learning model that only Azure can explain. Automated tools like this are convenient, but they often lack transparency and explainability. Although undeniably powerful, these tools are not infallible. For instance, in the sentiment analysis results seen in Figure 11-14, rows 3 and 5 were labeled as neutral and negative, respectively, despite being negative and positive reviews upon reading.

	A	B	C
1	tweet_text	Sentiment	Score
2	I drove by yesterday to get a sneak peak. It re-opens on July 14th and I can't wait to take my kids. The new range looks amazing. The entire range appears to be turf, which may or many not help your game, but it looks really nice. The tee boxes look state of the art and the club house looks like something you'll see on a newer course. Can't wait to experience it!	positive	9.05E-01
3	After waiting for almost 30 minutes to trade in an old phone part of the buy back program, our customer service rep incorrectly processed the transaction. This led to us waiting another 30 minutes for him to correct it. Don't visit this store if you want pleasant or good service.	neutral	0.57314
4	Wonderful reuben. Map shown on Yelp page is incorrect. It is actually a different Hawkins. I'd recommend a call for directions 412-271-9911.	positive	8.55E-01
5	This is a hidden gem, no really. It took us forever to find but well worth it. It is right across the street from the Rankin Police Station. The menu has a wide selection, I really couldn't decide what I wanted but I went with the ribeye sandwich. I'm glad i did too. Huge sandwich! I added mushrooms, it was very flavorful. My boyfriend got the fish sandwich, he enjoyed it as well. Fast and friendly service. Will definitely be back.	negative	2.97E-02
6	Awesome drink specials during happy hour. Fantastic wings that are crispy and delicious, wing night on Tuesday and Thursday!	positive	0.953895

Figure 11-14. Mislabeled reviews from sentiment analysis

The moral of the story is to leverage the full potential of AI, but also to exercise critical thinking and not rely solely on it. While AI possesses artificial intelligence, you possess the power of genuine human judgment and intuition. By combining the strengths of both, you can make more informed decisions.

Unstructured data is widely recognized as challenging to work with, but AI is well-suited to handle such data. Despite Excel's primary focus on structured data, there is a growing trend towards utilizing it for unstructured data, including text and images. However, just as Analyze Data performs optimally with a specific data layout, Azure's sentiment analysis add-in depends on an exact schema to effectively interpret the inputs and outputs of unstructured data.

Sentiment analysis is only the beginning. The upcoming integration of GPT-powered language modeling in Excel, such as Copilot, represents a significant leap forward in this regard. This integration promises to enhance Excel's capabilities and enable users to leverage powerful language modeling capabilities within their Excel workflows.

Conclusion

In conclusion, predictive analytics and AI are powerful tools that can help you gain deeper insights into your data and predict future outcomes. Excel has evolved to incorporate these tools, allowing users to leverage them for everything from image recognition to creating reports and analyses. By using Analyze Data for AI-powered insights, building predictive models with XLMiner, and integrating Excel with Azure Machine Learning, you can unlock the full potential of predictive analytics and AI in Excel.

Exercises

Practice using Excel's augmented analytics and AI features with these exercises and the datasets in the *exercises\ch_011_exercises* folder in the book's companion repository. Open *ch_11_exercises.xlsx* to start:

1. Perform sentiment analysis on a dataset of movie reviews located in the `imdb` worksheet using the Azure Machine Learning add-in. Afterwards, apply the XLMiner add-in to generate descriptive statistics for the obtained scores.
2. Import the *life_expectancy.png* image into Excel. Use the Analyze Data feature to produce a line chart that illustrates the average life expectancy over time. It might be necessary to adjust the data format to achieve this.

You can find the solutions to these exercises in *ch_11_solutions.xlsx* in the same folder.

CHAPTER 12

Python with Excel

Up to this point, this book has primarily focused on tools designed specifically for the Microsoft ecosystem, such as Power Pivot and Power Query. However, it concludes here with a discussion of an important programming language that has risen in popularity and become compatible with virtually every computer application imaginable, Excel included. Welcome to an introductory exploration of how Python can enhance your Excel experience.

This chapter is intentionally placed at the end of the book because I am aware of the apprehension it may cause among typical Excel users. Nevertheless, if you have reached this stage and are interested in advancing Excel's capabilities in the realm of modern analytics, I strongly recommend giving Python a try.

And it's not just my opinion that you should consider: Microsoft itself has endorsed the synergy between Python and Excel by developing an official native Python application within Excel, significantly broadening the scope of what you, as an analyst, can accomplish by leveraging these two powerful tools together.

However, this represents just one of the newer, specialized applications of Python with Excel, and it doesn't fully capture the breadth of what Python can offer Excel users. In this chapter, we'll explore the broader conceptual relationship between the two, accompanied by some examples. If you find this chapter intriguing, I encourage you to delve deeper into the native integration of Python into Excel.

The examples provided in this chapter do not utilize the native Python integration in Excel. Instead, they showcase alternative methods for combining these tools to achieve automation that surpasses the capabilities of the current Python-Excel integration.

Reader Prerequisites

Although this chapter can be completed without prior knowledge of Python, familiarity with concepts such as lists, indexing, loops, and the pandas and seaborn packages will greatly enhance your understanding.

If you'd like to learn more about Python before reading, I recommend starting with my book *Advancing into Analytics: From Excel to Python and R*, for a basic introduction to Python for Excel users. To delve even deeper into the subject, check out Felix Zumstein's *Python for Excel: A Modern Environment for Automation and Data Analysis* (O'Reilly, 2021).

This chapter is primarily a hands-on demonstration of Python coding. To gain the most from it, I encourage you to participate actively using your own computer. All that's required is a completely free version of Python, for which I recommend downloading the Anaconda distribution.

The Role of Python in Modern Excel

Prospective learners often find Python, an Excel-adjacent tool, intimidating. Many Excel users believe it should be the last tool on their learning list as it's not a Microsoft product and requires acquiring a new language proficiency.

Python may not be the best choice for every Excel user, but it's worth serious consideration for those looking to build complex automations, version-controlled projects, and other advanced development products. Let's explore the role of Python in modern analytics and its relationship with modern Excel.

A Growing Stack Requires Glue

When I first started as an analyst, my toolkit began and ended with Excel. Data management, reporting, dashboards—all were housed under the familiar green-and-white interface of Excel.

A few years on, the landscape has dramatically changed with the introduction of Power BI, Office Scripts, Jupyter Notebooks, and even Python integration within Excel. This expansion reflects broader technological shifts: moving from a single, all-encompassing application to a network of specialized, interconnected tools.

Navigating this diverse ecosystem requires a "conductor" or "glue" to seamlessly integrate various components. Whether it's transferring data between platforms, visualizing it in a new way, or deploying a cloud-based machine learning model to a user's dashboard, Python stands out as an exemplary choice. Its versatility spans from crafting simple scripts to developing complex, enterprise-level solutions, making it compatible with a wide range of operating systems and programming languages.

Microsoft has celebrated Python's role as a versatile "glue" language, incorporating its use across Azure, Power BI, SQL Server, and more. Python's popularity among developers and organizations alike has led to a thriving community of users and an abundance of resources.

Network Effects Mean Faster Development Time

"Everybody's doing it" isn't usually a good reason to engage in something, but in the case of programming languages it may hold merit.

Network effect, the concept that the value of something increases with its user base, applies to programming languages. As more programmers join, code sharing expands, providing a larger codebase for usage and further development, creating a virtuous cycle.

Python's versatility as a neutral "glue" language has led to its adoption in various professions, including database management, web development, and data analytics. This implies that regardless of the direction your Excel project takes or the tools you require, there's a high likelihood of finding collaborators who "speak" Python.

For example, imagine developing an inventory tracker or a similar tool using Excel, only to find that it has become too intricate for a basic workbook or so popular that there's a demand to transform it into a standalone web application. This transition typically represents a considerable challenge. However, if the original programming was done in Python, this shift can be notably more efficient and faster.

Python's versatility and widespread support in the web development realm facilitate smoother integration with various web technologies and platforms. Consequently, the time needed to evolve your Excel-based solution into a fully operational web application is greatly minimized. Essentially, starting with Python positions you advantageously and provides a robust foundation for future growth or adaptation, thereby optimizing the development trajectory as your project expands.

Bring Modern Development to Excel

Python enables modern Excel developers to implement best practices in software development, including unit testing, version control, and package development.

Unit testing

Unit testing involves testing individual components or units of software to verify that each one functions correctly on its own. It helps developers identify and fix bugs early in the development process, ensuring the reliability and performance of the final product.

Many programming languages offer *unit testing* features to guarantee that code operates as intended. However, Excel does not natively support this capability. While there are alternative tools available, Python stands out as an excellent option for unit testing because of its extensive network effects and the wealth of packages it offers. Automated unit testing improves reliability and decreases the chance of errors, which is particularly beneficial for Excel workbooks used by individuals with varying levels of technical expertise.

Version control

A *version control system* tracks changes in a repository, enabling users to view contributions, revert to previous versions, and so on. If you've ever struggled to differentiate between multiple workbooks like *budget-model-final.xlsx* and *budget-model-FINAL-final.xlsx*, you can appreciate the usefulness of version control.

Although Excel offers limited version control features, like viewing version history in OneDrive and using the Spreadsheet Inquire add-in, it falls short compared to the extensive features available when transitioning code production to Python.

Package development and distribution

If you're looking for an immediate reason to embrace Python for your daily analytical tasks, allow me to highlight a key advantage: *packages*.

While developing my own tools is something I enjoy, I also believe in the value of leveraging existing solutions when they meet my requirements. Python's robust capabilities for creating and distributing packages, especially via the Python Package Index, open up a universe of tools that are difficult to match with Excel add-ins or VBA modules. The vast majority of these tools are open source and freely available.

Whether your goal is to collect data from an application programming interface (API), analyze images, or simply generate descriptive statistics, the wide availability of Python packages makes a compelling case for investing time in learning Python. Notably, some of these packages are even designed to integrate smoothly with Excel.

Using Python and Excel Together with pandas and openpyxl

With Python's role in modern Excel in mind, let's explore how the two can work together. Two key packages to facilitate this integration are pandas and openpyxl. Let's consider the two in turn.

Why pandas for Excel?

If you're working with any kind of tabular data in Python, you won't get far without pandas. This package lets you, among other operations:

- Sort and filter rows
- Add, remove, and transform columns
- Aggregate and reshape a table
- Merge or append multiple tables

This is Python's equivalent to Power Query, enabling you to create reusable data cleaning and transformation workflows. And just like Power Query, pandas can effortlessly import data from diverse sources, including Excel, and even export the results of your analysis back to Excel.

The limitations of working with pandas for Excel

That said, pandas has limited features for deeply interacting with Excel workbooks. For example, it cannot help with the following tasks:

- Advanced formatting options for cells, such as applying specific styles or conditional formatting
- Support for executing Excel macros or VBA code within workbooks
- Direct access to Excel-specific features like data validation, charts, PivotTables, or formulas
- The ability to manipulate worksheets, such as modifying or deleting data

Fortunately, several packages exist to provide these more advanced Python/Excel features, most notably openpyxl.

What openpyxl contributes

openpyxl (pronounced *open pie Excel*) is a Python package providing functionality for working with Excel files, specifically the *.xlsx* file format. It allows users to read, write, and modify Excel spreadsheets programmatically. openpyxl integrates smoothly with pandas, allowing users to clean data using pandas and add additional functionality to the workbook using openpyxl.

Although openpyxl has limitations and cannot cover every Excel use case, it remains the single best Python package to get started with automating Excel tasks.

How to use openpyxl with pandas

Let's take a typical use case for automating a routine Excel business report where an analyst needs to generate monthly sales reports from multiple Excel worksheets. For these and other tasks, the basic workflow for using pandas with openpyxl is like so:

1. Read the data: use pandas to extract data from a variety of sources into tabular DataFrames.
2. Clean and analyze the data: use pandas to clean and manipulate the data, perform calculations, apply filters, handle missing values, and derive relevant insights.
3. Generate the report: use openpyxl to create a new Excel workbook or select an existing one. Populate the workbook with the consolidated data, apply conditional formatting, create charts, and incorporate any required visual elements.
4. Save the report: save the updated Excel workbook using openpyxl, specifying the desired filename and location.
5. Distribute and automate the report: send the generated report to the intended recipients through email, file sharing platforms, or any preferred method.

Other Python Packages for Excel

Powerful as it is for Excel tasks, especially when combined with pandas, openpyxl has limitations. Thankfully, other packages are available to handle specific use cases. Here are some other packages to be aware of:

XlsxWriter
: Similar to openpyxl, XlsxWriter can be used to write data, add formatting, and create charts to Excel files in the *.xlsx* format. This package is optimized for performance, particularly when working with large datasets. That said, as the name implies, XlsxWriter can only handle writing data to Excel, while openpyxl can both read and write.

xlwings
: This package enables the automation of Excel tasks, including interacting with Excel workbooks, running VBA macros, and accessing Excel's COM (Component Object Model) API on Windows. It provides complete two-way communication between Excel and Python in a way that openpyxl does not. On the other hand, this package requires a more complex development environment, with many features only available on Windows.

PyXLL

This is a paid library that enables users to write Excel add-ins using Python. Instead of automating Excel workbooks, PyXLL allows developers to build standalone applications for data science, financial trading, and other purposes. This enables users to engage with applications developed in Python directly within Excel, without the need to execute any Python code or understand the underlying Python mechanics.

Many other Python packages exist for Excel-related tasks, each with unique strengths and weaknesses.

Demonstration of Excel Automation with pandas and openpyxl

Time to stop discussing and start building! In this section we'll automate production of a small report from Python using pandas, openpyxl, and more.

First, we'll use pandas to perform complex data cleaning and analysis tasks that are difficult to achieve in Excel. After that, we'll create an overview worksheet consisting of a brief data summary and two charts, one from native Excel and one from Python. Finally, we'll load the entire dataset to a new worksheet and format the results.

A completed version of this script is available as *ch_12.ipynb* in the *ch_12* folder of the book's companion repository. If you're not sure how to open, navigate, or interact with this file, check out Part 3 of *Advancing into Analytics: From Excel to Python and R* as a primer to Python and Jupyter Notebooks.

Let's import the relevant modules and dataset to get started:

```
In [1]: # Data manipulation and visualization
        import pandas as pd
        import seaborn as sns

       # Excel file manipulation
       from openpyxl import Workbook
       from openpyxl.styles import PatternFill
       from openpyxl.chart import BarChart, Reference
       from openpyxl.drawing.image import Image
       from openpyxl.utils import get_column_letter
       from openpyxl.utils.dataframe import dataframe_to_rows
       from openpyxl.worksheet.table import Table, TableStyleInfo
```

The pandas library is capable of importing data from a variety of formats, including Excel workbooks, through the use of the `read_excel()` function. Let's import the *contestants.xlsx* file and name the resulting DataFrame `contestants`:

```
In [2]: contestants = pd.read_excel('data/contestants.xlsx')
```

Cleaning Up the Data in pandas

A pandas DataFrame can contain thousands, or even millions, of rows, making it impractical and computationally inefficient to print them all at every step of the analysis. Nevertheless, visually inspecting the data is essential for understanding its contents—a benefit well-known to Excel users. To quickly review the data and ensure it meets our expectations, we can use the `head()` method, which displays the first five rows:

```
In [3]: contestants.head()

Out[3]:

  EMAIL                          PRE POST SEX    EDUCATION  STUDY_HOURS
0 smehaffey0@creativecommons.org 485 494  Male   Bachelor's 20.0
1 dbateman1@hao12@.com           462 458  Female Bachelor's 14.8
2 bbenham2@xrea.com              477 483  Female Bachelor's 22.2
3 mwison@@g.co                   480 488  Female Bachelor's 21.3
4 jagostini4@wordpress.org       495 494  Female NaN        26.2
```

Based on this data preview, we identified a few issues that need to be addressed. First, it appears that some of the emails contain an invalid format. We also have a value in the `EDUCATION` column called `NaN`, which doesn't seem to belong. We can address these and other issues in the dataset in ways that would be difficult or impossible to do with Excel's features.

Working with the metadata

A good data analysis and transformation program should be equally proficient in handling both data and metadata, such as column header names. In this regard, pandas stands out as a particularly suitable tool.

Currently, our DataFrame has column names in uppercase. To make typing column names easier, I prefer using lowercase names. Fortunately, in pandas, we can accomplish this with a single line of code:

```
In [4]: contestants.columns = contestants.columns.str.lower()
        contestants.head()

Out[4]:

  email                          pre post sex    education  study_hours
0 smehaffey0@creativecommons.org 485 494  Male   Bachelor's 20.0
1 dbateman1@hao12@.com           462 458  Female Bachelor's 14.8
2 bbenham2@xrea.com              477 483  Female Bachelor's 22.2
3 mwison@@g.co                   480 488  Female Bachelor's 21.3
4 jagostini4@wordpress.org       495 494  Female NaN        26.2
```

Pattern matching/regular expressions

The `email` column in this DataFrame lists the email addresses of each contest participant. Our objective is to eliminate any rows containing invalid email addresses from this column.

To achieve this, we can employ text pattern matching, which is facilitated by a toolkit known as *regular expressions*. Although Power Query provides fundamental text manipulation features, such as converting text case, it does not support searching for specific text patterns—a capability that Python offers.

Crafting and validating regular expressions can be complex; however, numerous online resources such as ChatGPT are available to aid in this process. This is the regular expression we will use:

```
In [5]: # Define a regular expression pattern for valid email addresses
        email_pattern = r'^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}$'
```

Next, we can use the `str.contains()` method to keep only the records that match the pattern:

```
In [6]: full_emails = contestants[contestants['email'].str.contains(email_pattern)]
```

To confirm how many rows have been filtered out, we can compare the `shape` attribute of the two DataFrames:

```
In [7]: # Dimensions of original DataFrame
        contestants.shape

Out[7]: (100, 6)

In [8]: # Dimensions of DataFrame with valid emails ONLY
        full_emails.shape

Out[8]: (82, 6)
```

Refining our selection to include only participants with valid email addresses reduces the number from 100 to 82 contestants.

Analyzing missing values

The `info()` method offers a comprehensive overview of the DataFrame's dimensions and additional properties:

```
In [9]: full_emails.info()

    <class 'pandas.core.frame.DataFrame'>
    Index: 82 entries, 0 to 99
    Data columns (total 6 columns):
     #   Column       Non-Null Count  Dtype
    ---  ------       --------------  -----
     0   email        82 non-null     object
```

```
 1   pre          82 non-null     int64
 2   post         82 non-null     int64
 3   sex          82 non-null     object
 4   education    81 non-null     object
 5   study_hours  82 non-null     float64
dtypes: float64(1), int64(2), object(3)
memory usage: 4.5+ KB
```

In many computer programs, including Power Query, the term `null` signifies a missing or undefined value. Within pandas DataFrames, this concept is represented as `NaN`, an abbreviation for "Not a Number."

Although basic Excel does not offer an exact counterpart to `null`, Power Query (as explained in Chapter 2) incorporates this value, significantly enhancing data management and inspection processes. Nonetheless, handling these missing values programmatically in Power Query, such as eliminating them across all columns, can pose challenges. This task is simplified with pandas.

For instance, if one wishes to identify which columns contain the highest percentage of missing values, pandas facilitates this analysis effortlessly:

```
In [10]: full_emails.isnull().mean().sort_values(ascending=False)

Out[10]:

    education      0.012195
    email          0.000000
    pre            0.000000
    post           0.000000
    sex            0.000000
    study_hours    0.000000
    dtype: float64
```

Because there are so few missing values and only in one column, we will simply drop any row that has a missing observation in any column:

```
In [11]: complete_cases = full_emails.dropna()
```

To confirm that all missing observations have been cleared from the DataFrame, we can use the `info()` method again:

```
In [12]: complete_cases.info()

    <class 'pandas.core.frame.DataFrame'>
    Index: 81 entries, 0 to 99
    Data columns (total 6 columns):
     #   Column       Non-Null Count  Dtype
    ---  ------       --------------  -----
     0   email        81 non-null     object
     1   pre          81 non-null     int64
     2   post         81 non-null     int64
     3   sex          81 non-null     object
```

```
 4   education    81 non-null     object
 5   study_hours  81 non-null     float64
dtypes: float64(1), int64(2), object(3)
memory usage: 4.4+ KB
```

Creating a percentile

Using pandas, we'll create a percentile rank for the `post` column and confirm its validity by running descriptive statistics with `describe()`:

```
In [13]: complete_cases['post_pct'] = complete_cases['post'].rank(pct=True)
         complete_cases['post_pct'].describe()
```

```
Out[13]:

         count    81.000000
         mean      0.506173
         std       0.290352
         min       0.012346
         25%       0.265432
         50%       0.506173
         75%       0.759259
         max       1.000000
         Name: post_pct, dtype: float64
```

Creating a percentile column in Excel is straightforward, but validating results is easier in pandas with its statistical functions, methods for handling missing values, and more.

Let's confirm that our dataset has been cleaned and transformed using pandas:

```
In [14]: complete_cases.describe()
```

```
Out[14]:

      pre        post       study_hours post_pct
count 81.000000  81.000000  81.000000   81.000000
mean  480.506173 481.012346 23.445679   0.506173
std   20.626514  23.037737  8.178142    0.290352
min   409.000000 398.000000 0.000000    0.012346
25%   470.000000 467.000000 18.700000   0.265432
50%   484.000000 483.000000 22.600000   0.506173
75%   494.000000 497.000000 29.000000   0.759259
max   521.000000 540.000000 42.800000   1.000000
```

Now let's use openpyxl to create a styled summary report.

Summarizing Findings with openpyxl

Now that we've properly prepared the data using various techniques in pandas, we'll use openpyxl to create a summary in Excel. This will include both key figures and labels, as well as data visualizations.

Creating a summary worksheet

To get started building an Excel workbook with openpyxl, we'll declare variables representing workbook and worksheet objects:

```
In [14]: # Create a new workbook and select the worksheet
         wb = Workbook()

         # Assign the active worksheet to ws
         ws = wb.active
```

From there, we can populate any cell of the active sheet using its alphanumeric reference. I am going to insert and label the average pre and post scores in cells `A1:B2`:

```
In [16]: ws['A1'] = "Average pre score"

         # Round output to two decimals
         ws['B1'] = round(complete_cases['pre'].mean(), 2)
         ws['A2'] = "Average post score"
         ws['B2'] = round(complete_cases['post'].mean(), 2)
```

Inserting data into the workbook in this manner merely constitutes a basic data dump; it does not impact how the data is displayed in Excel. Based on my experience with data formatting, I anticipate that the labels in column A will need additional width. I'll proceed to adjust this now through the `width` property of the worksheet.

```
In [17]: ws.column_dimensions['A'].width = 16
```

Later in this chapter, we'll cover achieving autofit-like adjustments for column widths. But for now, let's shift our focus to adding charts to this summary.

There are two methods for generating Excel charts using Python. One approach involves scripting the creation of an Excel chart directly from Python code, while the other involves crafting a plot in Python and then inserting it into an Excel workbook. Both strategies have their advantages and disadvantages, which will be explored in turn.

Inserting charts

There are two methods for generating Excel charts using Python. One approach involves scripting the creation of an Excel chart directly from Python code, while the other involves crafting a plot in Python and then inserting it into an Excel workbook. Both strategies have their advantages and disadvantages, which will be explored in turn.

Option A: Create a native Excel plot. Excel's data visualization features are popular because they are easy to use and effective for basic visualization tasks. Let's explore how to create native Excel charts from Python using openpyxl.

To begin, we need to specify the type of Excel chart we want to create and identify the location of the data for the chart within the worksheet:

```
In [18]: # Create a bar chart object
         chart = BarChart()

         # Define the data range
         data = Reference(ws, min_col=2, min_row=1, max_col=2, max_row=2)
```

Next, we'll add this data source to the chart and label the chart's title and axes:

```
In [19]: # Add data to the chart
         chart.add_data(data)

         # Set chart title, axis labels
         chart.title = "Score Comparison"
         chart.x_axis.title = "Score Type"
         chart.y_axis.title = "Score Value"
```

Let's further customize this chart. We'll set category labels to reflect the data in the first column and also eliminate the chart legend:

```
In [20]: # Set category names
         categories = Reference(ws, min_col=1, min_row=1, max_row=2)
         chart.set_categories(categories)

         # Remove the legend
         chart.legend = None
```

With the chart fully defined and styled, it's time to insert it into the worksheet:

```
In [21]: # Add the chart to a specific location on the worksheet
         ws.add_chart(chart, "D1")
```

Option B: Insert a Python image. Python offers advantages in data visualization compared to Excel, as it provides more diverse visualization options and allows for easier customization of plots. For example, Excel lacks a built-in solution for analyzing relationships between multiple variables simultaneously. However, the seaborn data visualization package offers the `pairplot()` function, which provides a quick and convenient way to explore such relationships.

The following block visualizes these relationships across the selected variables of contestants. You can see the results in Figure 12-1:

```
In [22]: sns.pairplot(contestants[['pre', 'post', 'study_hours']])
```

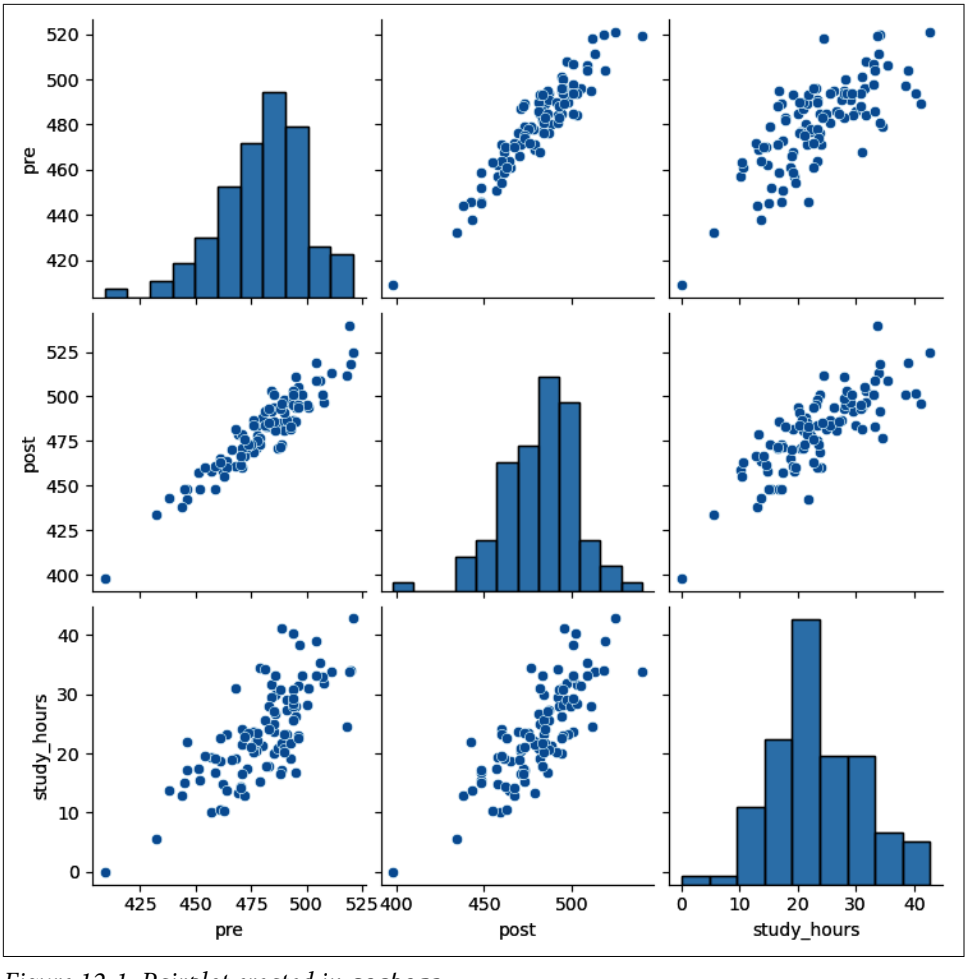


Figure 12-1. Pairplot created in `seaborn`

Not only does Python include a number of chart types that are difficult to build in Excel, they are easy to customize as well. For example, I'd like to see this visualization broken down by `sex`, which can be done by passing it to the `hue` parameter. I'm going to save the results of this plot (seen in Figure 12-2) as `sns_plot` so I can refer to it later:

```
In [23]: sns_plot = sns.pairplot(contestants[['pre', 'post',
             'study_hours', 'sex']], hue='sex')
```

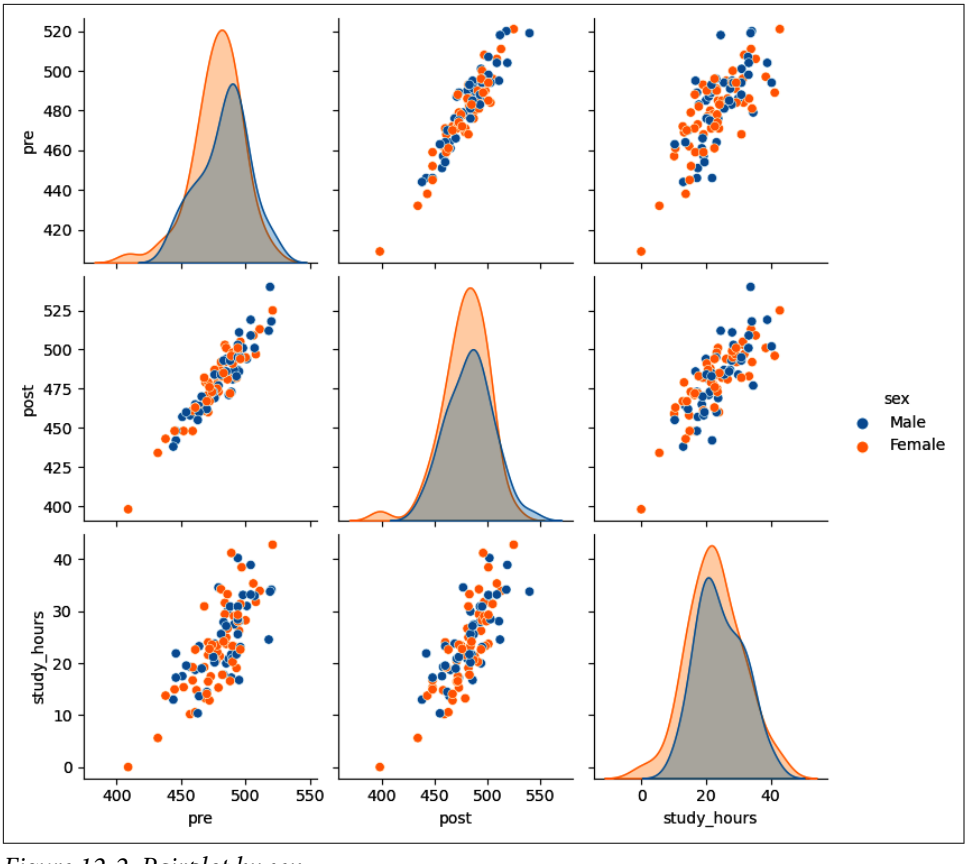


Figure 12-2. Pairplot by sex

Next, let's place a static image of this pairplot into the workbook. This requires first saving the image to disk, and then specifying where to place it into the workbook:

```
In [21]: # Save pairplot to disk as an image
         sns_plot.savefig('pairplot.png')

         # Load saved image into the worksheet
         image = Image('pairplot.png')
         ws.add_image(image, 'A20')
```

One of the aspects I particularly appreciate about Python plots is their ease of customization and the simplicity with which one can iterate between various types of plots. This kind of trial-and-error approach tends to be more challenging in Excel, due to its limited selection of chart options and the considerable effort required to enhance their visual appeal.

However, it's important to note that Python plots imported into Excel are essentially static images. If the underlying data changes, these charts will not update

automatically in the manner of native Excel charts. Additionally, imported Python plots lack interactive features, such as the tooltips that appear when you hover over elements in standard Excel charts.

The recent integration of Python with Excel offers the ability to create Python plots that possess some level of interactivity and can update in response to changes in the source data. For an excellent demonstration of this feature, check out this blog post by Excel MVP Mynda Treacy.

Excel versus Python charts

A summary of the pros and cons of these two methods is shown in Table 12-1.

Table 12-1. Pros and cons of Python versus Excel charts

	Pros	Cons
Building a native Excel plot	• Plot will update with changes in Excel data. • User can interact with and customize plot. • Plot can integrate with other Excel features like formulas and PivotTables.	• Limited number of Excel plot types exist. • It can be difficult to customize or iterate on Excel charts.
Inserting an image of a Python plot	• Access to several powerful plotting libraries such as `matplotlib` and `seaborn`. • Plot is easily audited and reproduced through the source code.	• The plot is a static image and lacks interactivity. • Updating or refreshing the chart from Excel is not possible.

The choice between methods depends on different factors, such data refresh needs and the availability of specific chart types in Excel. That said, the flexibility and range of options available in itself highlight Python's powerful capabilities for working with Excel.

Adding a Styled Data Source

Now that our summary worksheet has been created, we will create a second, styled worksheet consisting of the `complete_cases` DataFrame. The first step is to define this new worksheet:

```
In [25]: ws2 = wb.create_sheet(title='data')
```

Next, we'll iterate through each row of `complete_cases` and individually insert the rows into the worksheet:

```
In [26]: for row in dataframe_to_row(complete_cases, index=False, header=True):
             ws2.append(row)
```

Inserting the DataFrame into the worksheet is a start, but the resulting data may be challenging for users to read and manipulate. Let's make a few improvements.

Formatting percentages

By default, the `post_pct` column will be formatted as decimals instead of more readable percentages. To address this, we need to specify the location of this column in the worksheet and reformat it.

I will use the `get_loc()` method to find the index position of the `post_pct` column in the DataFrame, adding 1 to the results to account for Excel's 1-based indexing versus Python's 0-based indexing. The `get_column_letter()` function will then convert this index number into Excel's alphabetical column referencing:

```
In [27]: post_pct_loc = complete_cases.columns.get_loc('post_pct') + 1
         post_pct_col = get_column_letter(post_pct_loc)
         post_pct_col

Out[27]: 'J'
```

With the proper column identified, I will apply the desired number formatting to each row:

```
In [28]: number_format = '0.0%'

         for cell in ws2[post_pct_col]:
             cell.number_format = number_format
```

Converting to a table. As discussed in Chapter 1, Excel tables hold a number of benefits for data storage and analysis. We can convert this dataset into a table with the following code:

```
In [29]: # Specify desired table formatting
         style = TableStyleInfo(name='TableStyleMedium9', showRowStripes=True)

         # Name and identify range of table
         table = Table(displayName='contestants',
             ref='A1:' + get_column_letter(ws2.max_column) +
                str(ws2.max_row))

         # Apply styling and insert in worksheet
         table.tableStyleInfo = style
         ws2.add_table(table)
```

Applying conditional formatting. To enhance readability for end users, we can apply conditional formatting to the worksheet. The following code will apply a green background fill to participants above the 90th percentile and a yellow background fill to participants above the 70th percentile:

```
In [30]: # Define conditional formatting style
         green_fill = PatternFill(start_color="B9E8A2",
             end_color="B9E8A2", fill_type="solid")
         yellow_fill = PatternFill(start_color="FFF9D4",
             end_color="FFF9D4", fill_type="solid")
```

```
# Loop through data table and conditionally apply formatting
for row in ws2.iter_rows(min_row=2, min_col=1,
       max_col=len(complete_cases.columns)):
    # Convert index to 0-based indexing
    post_pct = row[post_pct_loc - 1].value
    if post_pct > .9:
       for cell in row:
           cell.fill = green_fill
    elif post_pct > .7:
        for cell in row:
           cell.fill = yellow_fill
```

Auto-fitting column widths. Although openpyxl lacks an autofit feature to automatically resize worksheet columns, we can achieve a similar outcome with the following code. It finds the widest row in each column of the worksheet, then adds sufficient padding to adjust the width of that column accordingly:

```
In [31]: for column in ws2.columns:
         max_length = 0
         column_letter = column[0].column_letter
         for cell in column:
             try:
                if len(str(cell.value)) > max_length:
                max_length = len(cell.value)
            except:
                pass
         adjusted_width = (max_length + 2) * 1.2
         ws2.column_dimensions[column_letter].width = adjusted_width
```

After completing the workbook, we can save the results to *ch12-output.xlsx*:

```
In [32]: wb.save('output/ch12-output.xlsx')
```

Conclusion

This chapter delved into the significant role of Python in enhancing modern Excel, underscoring its versatility as a "glue" language for development and its capability to augment Excel's functionalities. Through a practical demonstration, it illustrated how Python can automate Excel tasks, introducing features that are challenging or unattainable within the spreadsheet program alone.

As Microsoft continues to integrate Python into its data analytics suite, the synergy between Python and Excel is set to evolve. Nevertheless, this chapter lays down a robust foundation for leveraging Python and Excel in tandem, unlocking their full potential.

Exercises

To create a concise summary report of the *websites.xlsx* file in the *exercises\ch_12_exercises* folder in the book's companion repository, start by using the *ch_12_exercises.ipynb* Jupyter Notebook provided as a foundation. Complete the missing sections of this Notebook to reach the solution, which is available in the same folder named *ch_12_exercise_solutions.ipynb*.

For guidance on writing the code accurately, refer to the examples provided in this chapter. You are encouraged to enhance your work by integrating additional automated features.

CHAPTER 13

Conclusion and Next Steps

In the Preface, I stated the following learning objective:

> By the end of this book, you should be able to *use modern Excel tools for data cleaning, analysis, reporting, and advanced analytics*.

I sincerely hope you feel that this objective has been met and that you are now confident in advancing into further areas of analytics. As we conclude this stage of your journey in modern analytics, I would like to introduce some topics to further enrich and expand your understanding.

Exploring Excel's Other Features

I previously acknowledged in the Preface that this book cannot encompass every remarkable feature of modern analytics in Excel. Nevertheless, I aim to offer a list of noteworthy features and resources for your independent exploration. These "honorable mentions" will further deepen your understanding of the subject.

There are undoubtedly more tools to consider, and new ones continue to emerge. If you discover others that merit attention, please take the time to learn about them and share your findings with me and the community. After all, it requires a collective effort and more than a single book to harness the full capabilities of Excel.

LET() and LAMBDA()

The `LET()` and `LAMBDA()` functions significantly enhance efficiency, readability, and flexibility in Excel. Here's a concise introduction to both:

`LET()`
: The `LET()` function enables users to assign values to variables within a formula, improving readability and simplifying complex calculations. By defining variables at the start, users can reference them easily, streamlining the understanding and modification of intricate formulas.

`LAMBDA()`
: The `LAMBDA()` function enables users to create custom functions in Excel. This feature supports code modularity and minimizes redundancy by encapsulating complex operations for reuse across multiple formulas. Custom functions can be written to meet specific analytics needs, boosting productivity and enabling the development of sophisticated, specialized models and reports.

In essence, `LET()` and `LAMBDA()` equip Excel users with advanced tools to refine their analytical processes, enhance formula clarity, and expand the versatility and adaptability of their spreadsheets. For a detailed exploration of these functions, consider Chapter 15 of *Advanced Excel Formulas: Unleashing Brilliance with Excel Formulas*.

Power Automate, Office Scripts, and Excel Online

The evolution of analytics and automation within Excel is significantly enriched by integrating Power Automate, Office Scripts, and Excel Online, each contributing uniquely to streamline workflows and enhance productivity.

Power Automate stands out as a pivotal tool, automating a broad spectrum of tasks across Excel and other applications. Its capabilities extend from simple data entry to complex business processes. Particularly noteworthy is its ability to work with Power Query to automate data transformation.

Beyond data transformation, Power Automate's utility includes automating report generation and implementing notification systems. This versatility allows for the automation of routine tasks, freeing users to focus on more strategic activities.

The integration of Office Scripts with Excel Online further amplifies the power of Power Automate. Office Scripts, accessible in Excel Online, offers a web-based scripting language to record and automate repetitive Excel tasks. When combined with Power Automate, it enables these scripts to be executed automatically in response to specific triggers or events. For example, a script designed to adjust data within a workbook can be set to run automatically whenever a new file is uploaded to a designated SharePoint folder.

Additionally, Power Automate's interaction with Excel Online opens up opportunities for real-time collaboration and data processing. It facilitates operations like creating and updating workbooks, as well as extracting data from Excel files stored in the cloud, allowing for seamless collaboration among multiple users on Excel Online. This integration ensures that workflows are not only automated but also scalable and conducive to collaborative efforts.

In essence, the synergy between Power Automate, Office Scripts, and Excel Online empowers Excel users with advanced capabilities in automation, collaboration, and efficiency. This combination unlocks new potential in managing and analyzing data, making it an indispensable asset for modern Excel users looking to optimize their workflows and productivity. While detailed guides or books specifically covering this trio might not be readily available, Microsoft's documentation offers valuable insights and examples to get started.

Continued Exploration of Power Query and Power Pivot

A considerable part of this book is dedicated to exploring Power Query and Power Pivot in Excel, tools essential for data cleaning and analysis. While basic functions can be performed with minimal coding or technical knowledge, acquiring a more profound comprehension of these tools and understanding the foundational concepts significantly enhances their utility.

Power Query and M

Delving into intermediate Power Query and M concepts like parameters, custom functions, query optimization, automating refresh, and managing queries offers numerous advantages.

Parameters and custom functions provide flexibility and customization in data cleaning and analysis workflows. *Parameters* allow you to define values that can be used to control various aspects of your queries, such as filter conditions or connection settings, making your queries more dynamic and adaptable to changes. For instance, you can create a parameter to specify a date range for your data retrieval, enabling easy updates without altering the query's core logic.

Custom functions, on the other hand, let you create reusable pieces of code for operations that aren't directly available in Power Query's standard functionality or for complex transformations that you need to apply to multiple data sets. By defining a custom function, you can encapsulate a series of steps into a single, callable entity, streamlining your data processing tasks and ensuring consistency across your queries.

Query optimization is crucial as datasets grow in size and complexity. Optimizing queries ensures efficient data processing and reduces processing time. Learning query

optimization techniques helps streamline queries, eliminate unnecessary transformations, and improve overall performance. This knowledge is valuable for handling large datasets and complex transformations, increasing productivity and analysis speed.

Automating refresh and *managing queries* are also essential for maintaining up-to-date and reliable data analysis. Automating the refresh process ensures regular updates without manual intervention, saving time and effort. Effective query management allows for organization, monitoring, and troubleshooting of data transformations and connections. This proficiency ensures data integrity, reliability, and accurate insights for informed decision making.

In addition to these concepts that offer flexibility, efficiency, and enhanced data reliability, it's important to have a clear objective in mind when working with Power Query, just like any other project. For most projects, the goal is to create user-friendly data sources that seamlessly integrate with the Data Model in Power Pivot. Understanding data modeling principles such as normalization and schema design becomes crucial for using Power Query to its full potential as a data transformation tool. This approach acts as a bridge to grasping data modeling, which encompasses both Power Query and Power Pivot.

Power Pivot and DAX

To fully leverage the capabilities of Power Pivot in Excel, it is essential to understand and apply intermediate DAX and data modeling concepts. By grasping these concepts, users can make the most of Power Pivot and effectively address complex business questions and rules.

Before incorporating measures into the Data Model, it is crucial to ensure the proper design of the Data Model itself. This involves familiarizing oneself with concepts such as star schema (introduced briefly in Chapter 7), snowflake schema, and third normal form. These concepts play a significant role in data modeling and efficient data storage.

As you delve into more sophisticated DAX measures, optimizing code efficiency and readability becomes important. One effective technique for achieving this is using DAX variables. By storing intermediate results within a measure, DAX variables improve code clarity and performance.

For those seeking in-depth knowledge and guidance on advanced Power Pivot and DAX techniques, I recommend checking out Matt Allington's book, *Supercharge Excel: When You Learn to Write DAX for Power Pivot* (Holy Macro! Books, 2018). This resource offers valuable insights and practical guidance for expanding your skills in these areas and unlocking the full potential of Power Pivot in Excel.

Power BI for Dashboards and Reports

In Chapter 7 you were introduced to loading your work from Power Pivot to Power BI and were given a brief primer on how Excel and Power BI are conceptualized to work together. Further expanding your skill set to include Power BI offers a multitude of benefits that enhance your capabilities in data analysis, visualization, and reporting. Given your familiarity with Power Query and Power Pivot/DAX techniques from having read this book, transitioning to Power BI can be a natural and beneficial step forward.

One key draw for Excel users is Power BI's superior data visualization and dashboard features. Unlike Excel's general versatility, Power BI specializes in creating interactive reports and dashboards that are easily shareable and accessible on various devices, facilitating collaboration and access to insights. Moreover, Power BI's real-time analytics allows for immediate decision making based on the most current data, an essential feature for those dealing with dynamic data or needing constant metric monitoring.

For those beginning with Power BI, it's critical to understand the Power BI service ecosystem, encompassing Power BI Desktop for reporting and the Power BI service for distribution. Mastery of interactive visualization creation, advanced data modeling beyond Excel's scope, and DAX for intricate analyses and calculations is key. Additionally, learning how Excel integrates with Power BI enhances the ability to work fluidly across both platforms, improving data analysis and reporting efficiency and effectiveness.

Azure and Cloud Computing

Azure offers compelling advantages for modern analytics users in Excel. Excel users can enhance their analytics workflows with Azure's cloud-based infrastructure and services. Azure provides integrations like Power BI for interactive reporting and visualization, Azure Machine Learning for predictive models, and Azure Cognitive Services for analyzing unstructured data. You already briefly explored Azure's capabilities in Chapter 11 during sentiment analysis.

This integration combines Excel's powerful features with Azure's advanced capabilities, unlocking new possibilities for data analysis, machine learning, and data-driven decision making. For a comprehensive introduction to Azure, I recommend Jonah Andersson's *Learning Microsoft Azure: Cloud Computing and Development Fundamentals* (O'Reilly, 2023). For a hands-on exploration of bringing machine learning and AI to Power BI using Azure, Tobias Zwingmann's *AI-Powered Business Intelligence: Improving Forecasts and Decision Making with Machine Learning* (O'Reilly, 2022) provides valuable insights and practical guidance.

Python Programming

Learning Python is highly valuable for modern analytics users in Excel, whether you aim to automate spreadsheet production (as introduced in Chapter 12) or optimize machine learning models in Azure. Python is widely recognized as a leading programming language for AI development, prominently utilized in frameworks like TensorFlow, PyTorch, and Keras.

With the continuous advancements in AI, new tools and frameworks emerge, underscoring the importance of a strong Python foundation for developers. By acquiring Python skills, you position yourself for success in the rapidly evolving field of AI and related domains. Moreover, Python's versatility extends beyond AI and encompasses diverse applications, such as web development, data analysis, and automation.

To embark on your Python journey, I recommend starting with Al Sweigart's book, *Automate the Boring Stuff with Python: Practical Programming for Total Beginners*, second edition (No Starch Press, 2019). It serves as an excellent resource for beginners, providing a practical approach to learning Python and applying it to real-world tasks.

Large Language Models and Prompt Engineering

Chapter 11 introduced the concept of natural language querying in Excel through the Analyze Data feature, which marks the beginning of integrating AI tools like Copilot for enhanced data analysis. As Copilot continues to evolve, mastering AI-powered tools becomes increasingly critical.

For those involved in modern analytics with Excel, understanding large language models (LLMs) and prompt engineering is essential. LLMs, such as OpenAI's generative pre-trained transformer (GPT), are adept at understanding and generating human-like text, making them indispensable for analyzing unstructured data, deriving insights, and producing detailed reports.

GPT, a key LLM, includes various implementations like ChatGPT, designed for conversational interactions. To leverage conversational AI and ChatGPT fully, one must learn prompt engineering basics. This involves creating effective prompts that clearly convey analytical goals to the model, leading to precise and valuable responses. Copilot, built on the GPT model, benefits significantly from proficiency in these techniques.

Excel users will find that learning to craft questions, structure prompts, and include pertinent context can enhance their data analysis processes. These skills open new avenues for discovering insights and supporting informed decision making.

Parting Words

Science fiction author William Gibson famously stated, "The future is already here; it's just not evenly distributed." This concept resonates deeply with the modern experience of using Excel. With its extensive range of features and infinite possibilities for discovery, it's understandable to feel overwhelmed. Moreover, with rapid advancements in technology, there's a common fear of being left behind. However, by embracing a step-by-step approach and acknowledging that perfect mastery is not required, you can unlock more potential in Excel than you initially believed possible and remain competitive in the current business landscape.

Reflect on your achievements thus far with this book; you have every reason to be proud. However, do not dwell on these accomplishments for too long. There is much more to discover, and it won't be long before you realize that this book merely scratches the surface. As you conclude this chapter and this book, embrace the challenge: venture forth, continue learning, and advance your journey into modern analytics with Excel.

Index

Symbols

operator, 160
symbols, 42
* symbol, 162
+ symbol, 163
| (pipe) symbol, 137

A

ABS() function, 158
Advanced Editor, in Power Query, 21
Advanced Excel Formulas (Murray), 166, 210
Advancing into Analytics (Mount), 181, 190, 195
AI (artificial intelligence), 171-172, 214
  (see also Analyze Data feature; augmented analytics)
AI-Powered Business Intelligence (Zwingmann), 213
ALL() function, 138-140
Allington, Matt, 212
Anaconda, 190
Analyze Data feature, in Excel, 173-178
Andersson, Jonah, 213
Applied Steps list, 14, 22
arguments, versus parameters, 158
Arnold, Jeremey, 114
array formulas, 156-157
array parameter
  of FILTER() function, 161
  of SORTBY() function, 163
array references, 154-156
  dynamic, 155
  static, 154
arrays
  defined, 154
  ranges versus, 154
augmented analytics, 171-187
  Analyze Data feature, 173-178
  growing complexity of data and analytics, 171
  OCR technology, 181-184
  self-service BI, 172
  sentiment analysis, 184-187
  XLMiner add-in, 179-181
Automate the Boring Stuff with Python (Sweigart), 214
Available columns list, in Power Query, 50
Azure Machine Learning add-in, 184-187
Azure, cloud computing with, 213

B

big data, 15
blank values, removing, 34-35
bridge tables, 101
business intelligence (BI), self-service, 172
business performance, tracking (see KPIs)
by_array1 parameter, of SORTBY() function, 163
[by_array2] parameter, of SORTBY() function, 163
[by_col] parameter, of UNIQUE() function, 158

C

CALCULATE() function, 134-140
  combined with ALL(), 138-140
  combined with DATESYTD() and DATEADD() functions, 147

combined with SAMEPERIODLAST-YEAR() function, 145
combined with USERELATIONSHIP() function, 142
filter context and, 134
with multiple criteria, 136-137
AND conditions, 136
OR conditions, 137
with one criterion, 135-136
calculated columns
in Power Pivot
profit margin example, 107-109
recoding column values, 109-111
when to use, 106
in Power Query, 49-52
Calculations group, in Power Pivot, 83
cardinality, 99-103
filter direction and, 106
importance of, 102
many-to-many, 101
one-to-many, 101
one-to-one, 100
case changes, in Power Query columns, 45-47
cell errors, 28-30
ChatGPT, 214
Close & Load option, Power Query Editor, 24
cloud computing, with Azure, 213
code efficiency and readability, optimizing, 212
collaboration, 211
colors, in visualizations, 130
Column distribution option, in Power Query, 29
columns, in Power Pivot
calculating in Power Query versus, 106
calculating profit margin, 107-109
creating, 106-110
formatting, 98
recoding column values, 109
columns, in Power Query, 45-55
calculating in Power Pivot versus, 106
changing case, 45-47
changing data types, 47
custom
calculated versus measures, 52
creating, 49-52
loading and inspecting data, 51
dates, 48-49
deleting, 22, 48
delimiting by, 47
replacing headers, 42
reshaping data, 53-54
commas, items separated by
splitting into columns, 47
splitting into rows, 39-40
connection-only loading option, 25
Copilot, 214
COUNTA() function, 160
Create Table dialog box, 3
CROSSFILTER() function, 106
.csv files, loading into Power Query, 63
custom columns, in Power Query, 49-52
custom functions, 210, 211

D

data mining, 172
Data Model, 80-81
cardinality in, 99-103
importance of design and understanding of, 212
loading to Power BI, 113-117
managing relationships within, 94
viewing in Diagram View, 93
Data Modeling with Microsoft Excel (Obeng Boateng), 132
data profiling, 26-31
cell errors, 28-30
closing out of, 31
defined, 27
detailed insights, 30
fixed-width text, 27
large datasets, 31
leading or trailing spaces, 27
missing values, 28
Power Query Editor, 27-31
valid data, 28
data scientists, 172
data sources, in Power Query, 21
appending data from multiple sources, 57-62
appending queries, 61-62
connecting to external workbooks, 58-60
data types, in Power Query
changing, 47
Excel cell formatting versus, 51
setting, 23
data validity, 28
DataFrame, in pandas
analyzing missing values in, 197

metadata, 196
pattern matching in, 197
reviewing data in, 196
Date Table, 141
DATEADD() function, 147
dates, in Power Query, 48-49
DATESYTD() function, 147
DAX formula language, 80, 83
CALCULATE() function, 134-140
CROSSFILTER() function, 106
measures, 83, 119-127, 212
explicit, 122-127
implicit, 119-121
time intelligence functions, 140-147
Definitive Guide to DAX, The (Ferrari and Russo), 140
delimiters, items separated by
splitting into columns, 47
splitting into rows, 39-40
describe() method, 199
Diagram View, in Power Pivot Editor, 89, 93
dimension tables, 92-94
arranging the Diagram View, 93
editing relationships, 94
distinct values
finding, 158
unique values versus, 159
domain knowledge, 35
duplicates, removing from Power Query rows, 34
dynamic array functions, 153-169
arrays and array functions versus, 153-157
FILTER() function, 160-163
modern Excel and, 169
SORTBY() function, 163-164
spill operator, 160
static array formulas versus, 157
UNIQUE() function, 158-159
XLOOKUP() function, 165-168
dynamic array references, 155

E

empty values, in Power Query, 28
ETL process, 15-18
extracting data, 15-16
loading data, 18
transforming data, 17
EXACT() function, 161
[exactly_once] parameter, of UNIQUE() function, 158
Excel
augmented analytics, 171-187
cell formatting versus Power Query data types, 51
dynamic array functions, 153-169
Excel Online, 211
LAMBDA() function, 210
LET() function, 210
Office Scripts, 210
Power Automate, 210
Python and, 189-206
explicit measures
creating, 122-127
implicit measures versus, 126
expression parameter, of CALCULATE() function, 134

F

fact tables, 92-94
arranging the Diagram View, 93
editing relationships, 94
Ferrari, Alberto, 140
filter direction, 103-106
cardinality and, 106
changing, 106
filtering orders with users, 104
filtering users with orders, 105
filter parameter, of CALCULATE() function, 134, 136
FILTER() function, 160-163
filtering by multiple criteria, 162-163
AND criteria, 162
nested AND/OR criteria, 163
OR criteria, 163
header columns and, 162
finding and replacing, in Power Query, 46
fixed-width text, 27
footers, viewing, 6-8
formula bar, Power Query Editor, 20
formulas
assigning values to variables in, 210
errors in, 28, 123
formula references, 8
functions versus, 153
functions
custom, 210, 211
dynamic array functions, 153-169

formulas versus, 153
time intelligence functions, 140-147
fx symbol, 124

G

get_column_letter() function, 205
get_loc() method, 205
Gibson, William, 215
goals, business (see KPIs)
GPT (generative pre-trained transformer), 214

H

head() method, 196
headers
column, replacing, 42
table, creating and referring to, 3-5
hierarchies, 111-113
creating, 111
using in PivotTables, 112
hue parameter, 202

I

IF() statements, 110
[if_empty] parameter, of FILTER() function, 161
[if_not_found] parameter, of XLOOKUP() function, 166
implicit measures
creating, 119-121
explicit measures versus, 126
include parameter, of FILTER() function, 161
index columns, adding from Power Query, 17, 23
info() method, 197
inner join, 68-69

J

junction tables, 101

K

KPIs (key performance indicators), 83, 128-132
adding to PivotTables, 131
adjusting icon styles, 130
creating, 128-129
status thresholds of, 129, 131

L

LAMBDA() function, 210
large language models (LLMs), 214
leading spaces, 27
Learning Microsoft Azure (Andersson), 213
Learning Microsoft Power BI (Arnold), 114
left outer join, 64-67
LET() function, 210
linear regression, in XLMiner, 180
loading options, in Power Query, 25
lookup_array parameter, of XLOOKUP() function, 166
lookup_value parameter, of XLOOKUP() function, 166

M

M programming language, 20, 49, 211
machine learning, 172, 184-187
Manage Relationships dialog box, in Power Pivot, 94
many-to-many relationships, 101
[match_mode] parameter, of XLOOKUP() function, 166
measures
calculated columns versus, 52
DAX, 83, 119-127, 212
Measures option, in Power Pivot, 122
melting columns (see unpivoting columns)
metadata
in pandas, 196
in tables, 5
Mézquita, Orlando, 15
missing values, 28
analyzing with pandas, 197
removing, 34-35
Model View, in Power BI, 116
multiple data sources, appending data from, 57-62
appending queries, 61-62
connecting to external workbooks, 58-60
Murray, Alan, 166

N

#N/A errors, 167
Name Manager, 8
NaN (Not a Number), 198
natural language querying, 175
network effect, 191

notification systems, 210
null values, 14, 28, 34
    items erroneously marked with, 42
    in left outer joins, 64
number function, of ABS() function, 158
numbers, changing to Text data type, 47

O

Obeng Boateng, Bernard, 132
OCR (optical character recognition), 181-184
Office Scripts, 210
one-to-many relationships, 101
one-to-one relationships, 100
openpyxl package, 193-194
    functions of, 193
    summarizing findings, 200-206
        adding styled data sources, 204-206
        auto-fitting column widths, 206
        conditional formatting, 205
        converting to tables, 205
        creating native Excel plots, 201
        creating worksheets, 200
        formatting percentages, 205
        inserting Python images, 201-204
    using with pandas, 194
optical character recognition (OCR), 181-184

P

package development and distribution, 192
pairplot() function, 201
pandas package, 193
    data cleaning, 196-199
        analyzing missing values, 197
        creating percentiles, 199
        metadata, 196
        pattern matching, 197
    functions of, 193
    limitations of, 193
    using openpyxl with, 194
parameters
    arguments versus, 158
    Power Query and M, 211
pattern matching, 197
performance of business, tracking (see KPIs)
pipe (|) symbol, 137
PivotTables
    adding KPIs to, 131
    generating from Power Pivot, 96
    unpivoted datasets in, 54
    using hierarchies in, 112
Power Automate, 210
Power BI
    cardinality and, 102
    dashboards and reports, 213
    importance of, 113
    loading Data Model to, 113-117
    Power Pivot visuals versus, 130
    viewing data in, 116-117
Power Pivot, 77-85
    CALCULATE() function, 134-140
        with ALL(), 138-140
        filter context and, 134
        with multiple criteria, 136-137
        with one criterion, 135-136
    columns in
        calculating in Power Query versus, 106
        calculating profit margin, 107-109
        creating, 106-110
        formatting, 98
        recoding column values, 109
    Data Model, 80-81
    DAX measures, 119-127, 212
        explicit measures, 122-127
        implicit measures, 119-121
    defined, 77
    interface, 83-84
        Calculations group, 83
        Data Model group, 83
        Relationships group, 84
        settings management, 84
        Tables group, 84
    KPIs, 128-132
        adding to PivotTables, 131
        adjusting icon styles, 130
        creating, 128-129
    loading, 81-83
    Power Query versus, 79
    reasons for using, 77-79
    relational models, 87-118
        cardinality, 99-103
        connecting to data sources, 87-88
        creating columns, 106-110
        creating relationships, 88-92
        fact and dimension tables, 92-94
        filter direction, 103-106
        hierarchies, 111-113
        loading Data Model to Power BI, 113-117

loading results to Excel, 95-98
time intelligence functions, 140-147
adding calendar tables, 140-142
creating, 143-147
Power Query, 13-32
appending data from multiple sources, 57-62
appending queries, 61-62
connecting to external workbooks, 58-60
pros and cons of using Power Query for, 80
channeling data through before connecting to Power Pivot, 87
columns in, 45-55
calculating in Power Pivot versus, 106
changing case, 45-47
changing data types, 47
custom, 49-52
dates, 48-49
deleting, 22, 48
delimiting by, 47
replacing headers, 42
reshaping data, 53-54
data profiling, 26-31
closing out of, 31
defined, 27
Power Query Editor, 27-31
defined, 13
ETL process, 15-18
loading options, 25
M and, 211
myths debunked by, 13-15
big data limitations, 15
null values, 14
reproducibility, 13
Power Pivot versus, 79
queries, 70-72
appending, 61-62
grouping, 70
loading from Power Query to Power Pivot, 88
refreshing, 37-38
viewing dependencies, 71
refreshing results in Excel, 38
relational joins, 62-69
inner join, 68-69
left outer join, 64-67
replacing values in, 46
rows in, 33-44
filling down blank, 43
fixing typos, 35
refreshing queries, 37-38
removing duplicates, 34
removing missing values, 34-35
sorting, 34
splitting data into, 39-42
Power Query Editor, 18-26
Column profiling based on entire data set option, 31
Data Preview options, 27-30
Column distribution checkbox, 28-30
Column profile checkbox, 30
Column quality checkbox, 28-30
Monospaced checkbox, 27
Show whitespace checkbox, 27
exiting, 24-25
imported data, 22-24
queries list, 21
returning to, 26
ribbon menu, 19-21
Add Column tab, 20
Home tab, 19
Transform tab, 20
View tab, 20
Power View, 113
(see also Power BI)
prompt engineering, 214
Python, 189-206
books about, 190
downloading, 190
Excel versus Python charts, 204
learning about, 190
openpyxl
overview of, 193-194
summarizing findings, 200-206
pandas
data cleaning, 196-199
overview of, 193
programming, 214
role of in modern Excel, 190-192
as “glue” language, 190
network effect, 191
package development and distribution, 192
unit testing, 192
version control, 192
Python for Excel (Zumstein), 190
PyXLL package, 195

Q

qualitative variables, 30
quantitative variables, 30
queries, 70-72
  appending, 61-62
  grouping, 70
  management of, 212
  natural language querying, 175
  optimization of, 211
  refreshing, 37-38
  viewing dependencies, 71
Queries & Connections options, in Power Query, 26, 61

R

RANDARRAY() function, 168
range parameter, of UNIQUE() function, 158
ranges, versus arrays, 154
real-time data, 172
refresh process, in Power Query
  automating, 212
  refreshing results in Excel, 38
regular expressions, 197
relational joins, 62-69
  inner join, 68-69
  left outer join, 64-67
relational models, 87-118
  cardinality, 99-103
  columns
    calculating in Power Query versus in Power Pivot, 106
    calculating profit margin, 107-109
    creating, 106-110
    recoding column values, 109
  connecting to data sources, 87-88
  creating relationships, 88-92
  fact and dimension tables, 92-94
  filter direction, 103-106
  hierarchies, 111-113
  loading Data Model to Power BI, 113
  loading results to Excel, 95-98
Relationships group, in Power Pivot, 84
report generation, automating, 210
reproducibility
  Excel, 13
  VBA, 14
return_array parameter, of XLOOKUP() function, 166
rows, in Power Query, 33-44
  filling down blank, 43
  fixing typos, 35
  refreshing queries, 37-38
  removing duplicates, 34
  removing missing values, 34-35
  sorting, 34
  splitting data into, 39-42
Russo, Marco, 140

S

SAMEPERIODLASTYEAR() function, 145
schema, for sentiment analysis, 185
scripting, 210
seaborn data visualization package, 201
[search_mode] parameter, of XLOOKUP() function, 166
sentiment analysis, 184-187
SEQUENCE() function, 168
sorting
  rows, in Power Query, 34
  SORTBY() function, 163-164
    sorting by another column without printing, 164
    sorting by multiple criteria, 164
[sort_order1] parameter, of SORTBY() function, 163
[sort_order2] parameter, of SORTBY() function, 163
spaces in data, 27
#SPILL errors, 157
spill operator (#), 160
star schema, 94
static array formulas, 156
static array references, 154
streaming data, 172
structured references, 9
Supercharge Excel (Allington), 212
Sweigart, Al, 214
SWITCH() function
  conditional columns versus, 110
  recoding column values, 109

T

Table View, in Power BI, 117
Tableau, 130
tables, 3-11
  cardinality and, 99-103
  fact and dimension tables, 92-94
  footers, 6-8

formatting, 9
headers, 3-5
naming, 8
organizing data for analytics, 10
updating ranges, 9
Tables group, in Power Pivot, 84
text
changing case of, 45-47
changing from Whole Number data type to, 47
converting images to, 181-184
Text to Columns feature, in Excel, 39
TEXTSPLIT() function, 168
tidy data rules, 10, 53, 177
time intelligence functions, 140-147
adding calendar tables, 140-142
creating, 143-147
total row, in tables, 6
TOTALYTD() formula, 143
trailing spaces, 27
Transform Data button, in Power Query, 60
transformations
columns, 45-55
ETL process, 17
rows, 33-44
Treacy, Mynda, 204
Tufte, Edward, 140
typos, fixing in Power Query, 35

U

unique values
distinct values versus, 159
finding, 158
UNIQUE() function, 157-159
finding distinct and unique values with, 158
finding unique versus distinct values, 159
unit testing, 192
unpivoting columns, in Power Query, 53-54
unstructured data, 171
updates, regular, 212
UPPER() function, 5
USERELATIONSHIP() function, 142

V

variables
assigning values to, 210
optimizing code efficiency and readability using, 212
qualitative, 30
quantitative, 30
VBA (Visual Basic for Applications), 14
version control, 192
VLOOKUP() function, 63, 78, 165
VSTACK() function, 168

W

whitespace, 27, 41
Whole Number data type, changing to text, 47
Wickham, Hadley, 10
width property, 200
workbooks, Excel
connecting to with Power Query, 58-60
importing to Power BI, 114

X

XLMiner add-in, 179-181
XLOOKUP() function, 64, 165-168
basic lookups, 166
error handling and, 167
looking up to the left, 168
pros and cons of combining data sources using, 80
VLOOKUP() versus, 165
XlsxWriter package, 194
xlwings package, 194

Z

Zumstein, Felix, 190
Zwingmann, Tobias, 213

About the Author

George Mount is the founder and CEO of Stringfest Analytics, a consulting firm specializing in analytics training. George speaks regularly on this topic and maintains a blog at *stringfestanalytics.com*.

In addition to being the author of *Advancing into Analytics: From Excel to Python and R* (O'Reilly, 2021), he has been recognized as a Microsoft Most Valuable Professional (MVP) for his contributions to the community and technical expertise in Excel.

George holds a bachelor's degree in economics from Hillsdale College, as well as master's degrees in finance and information systems from Case Western Reserve University. He currently resides in Cleveland, Ohio.

Colophon

The animal on the cover of *Modern Data Analytics in Excel* is a Hercules beetle (*Dynastes hercules*), a species of rhinoceros beetle found in the rainforests of Central and South America, as well as some Caribbean islands.

These titans of the insect world are the longest beetles on Earth, with males reaching up to 7 inches in length, including their horns, used for grappling with other males for dominance (females lack horns entirely). Hercules beetles come in shades of olive green or brownish yellow, sometimes with an iridescent sheen, and have small black spots scattered across their bodies. These colors can shift based on humidity.

Despite their impressive size, Hercules beetles are generally harmless insects. They feed on rotting fruit and tree sap, and the impressive strength of their horns, which allows them to lift objects hundreds of times their own weight, comes in handy when searching for food or burrowing in the forest floor.

Hercules beetles are not currently considered an endangered species. However, habitat loss due to deforestation is a threat to their populations. Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on an antique line engraving from *Goldsmith's Natural History*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.

O'REILLY®

Learn from experts.
Become one yourself.

Books | Live online courses
Instant answers | Virtual events
Videos | Interactive learning

Get started at oreilly.com.

©2023 O'Reilly Media, Inc. O'Reilly is a registered trademark of O'Reilly Media, Inc. 175-7431575