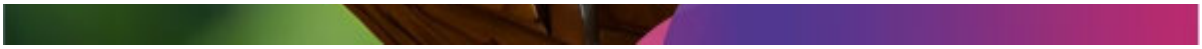
# Self-Service Analytics with Power BI

Learn how to build an end-to-end analytics solution in Power BI

Annu Roy
Rishiraj Deb
Gaurav Aroraa

bpb



# Self-Service Analytics with Power BI

## Learn how to build an end-to-end analytics solution in Power BI

Annu Roy

Rishiraj Deb

Gaurav Aroraa

www.bpbonline.com

First published: 2024

Published by BPB Online

WeWork

119 Marylebone Road

London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55518-200

www.bpbonline.com

# Foreword

Data, Analytics and Reporting have fascinated me ever since I started my career as a software developer in IT industry. It has been a transformational journey so far, wherein I worked on a wide spectrum of products and technologies and helped customers across industries with the best business intelligence solutions to meet their objectives. Being in industry for more than 19 years and a leader of Analytics and Cloud practice, I have had the opportunity to build capabilities in Analytics and Reporting space and drive strategic initiatives to evangelize business intelligence, maximize leverage of the best tools, and cultivate a culture for continuous innovation, capability enrichment, top talent development, process assurance, and quality.

Power BI is a suite of business intelligence, reporting and data visualization products and services that helps organizations to create a data-driven culture with business intelligence for all. It is a range of software services, apps, and connectors that work in conjunction to turn unrelated data sources into insights that are cohesive, interactive, and visually immersive. In any business, system generates a wide variety of data in the size of terabytes, petabytes or more, and business intelligence is used to analyze this data and create actionable insights. The organization’s success relies on these decisions so its critical that companies leverage the potential of business intelligence and reporting to make complex business decision and be successful in the competitive world.

Business intelligence is critical for making decisions in today's competitive market. Irrespective of the business size, to truly understand the business performance and monitoring critical metrics is essential. Another key aspect is the self-service that enables end-users to access, analyze, and interpret data without relying on extensive IT support or expertise. With the help of self-service Business Intelligence, business users can create their reports and visualizations using intuitive tools and user-friendly interfaces. Power BI is one such powerful tool that can generate insights into data and drive informed decisions. As per Gartner, Power BI is a powerful and flexible platform for self-service and enterprise business intelligence. One of the standout features of Power BI is its ability to connect to and visualize any data, enabling users to reveal insights, and create visually stimulating reports quickly and easily.

This book provides a good overview of why business intelligence reporting is critical to success of any business and how Power BI enables any organization to provide data insights to support decision making. It starts with "why" providing an insight into the importance of visualization and musically taking you through journey of how Power BI addresses different dimension of problems in this space. It provides a detailed view of key features of Power BI and the capabilities of both built-in and custom visuals. What I liked the most is the way authors have provided a simplified step-by-step journey into wide range of features and capabilities of Power BI. How it can be used to build customizable dashboards, drag-and-drop report authoring, data modelling, managing large data volume and more. The right mix of business scenarios, challenges, and pictorial view of how Power BI features can be used to solve the problems is commendable.

As a reader, I find this book relevant. Its easy to follow details makes it very helpful for me and anyone interested in Power BI. This book justifies the title and provides adequate information on Power BI with clear objectives and a step-by-step approach to enhance your knowledge and provide you with good sources of information.

Kumar

Dedicated to

My son Shikhar an ardent reader and budding writer

Roy

My daughter Irene Deb with all my love and best wishes

Deb

My beloved wife Shuuby & My Daughter Aarchi

Aroraa

About the Authors

Annu Roy is a Service Area Leader managing and collaborating with a team of technical experts to help clients adopt modern ways of working in their IT landscape. She is a PMP certified, senior Program Manager with experience in Complex Program Management. She has led diverse and distributed, multi-disciplinary teams of Application Leads, Architects, Business Analysts and technical resources located in different geographies, spanning Government, Healthcare and Finance industries. She is a certified Scrum Master, SAFe Agilist and ITIL V3 Expert. With more than 2 decades of experience in the IT Industry working with multiple big brands, she has a passion for mentoring students and professionals and believes that every individual deserves an opportunity to learn and grow.

Rishiraj Deb is a Senior Consultant and Subject Matter Expert, having vast experience in designing & delivering complex data analytics and business intelligence projects, helping businesses gain a better insight on their data. Rishiraj strongly believes enterprise data is an asset and is fascinated about creating data driven solutions using various data science and data analytics platforms and frameworks. Rishiraj has a master’s degree in computer science apart from having various professional certifications. Having worked for multiple big enterprises across industries, he is familiar with the real-life challenges businesses face today regarding utilizing their data and making the most out of it. Various concepts explained in the book should help address most of those challenges.

Gaurav Aroraa is a passionate technology leader and hands-on technologist, with a track record of driving technical innovation. Gaurav has delivered solutions for enterprises and start-ups operating in leadership, management, architectural, and development capacities. Gaurav has over 26 years of experience, collaborating with some of the most well-known technologies like Java, Microsoft, Angular, React, Python, PHP etc. pertaining to the domains Medical, Media, Construction, Gaming, Finance, ATM, Supply-chain, and so on. Gaurav has a doctorate in computer science (Machine Learning) from California Public University. Gaurav is a Microsoft MVP award recipient. He is a Mentor of Change with AIM NITI Aayog, Govt. of India, Business Coach with Business Blaster, Govt of NCT of Delhi. He is a lifetime member of the Computer Society of India (CSI), an advisory member and senior mentor at IndiaMentor. He has authored books across-the-technologies. Recently, Gaurav has recognized as a world record holder for writing books in exceptional technologies. His recent publications are:

Microservices by Examples Using .NET (BPB)

Data Analytics Principles, Tools, and Practices (BPB)

Enterprise Integration with MuleSoft (BPB)

## About the Reviewers

Pujarini Mohapatra is a seasoned professional with more than 15 years in Enterprise Software Services, Product Development & Management. With deep expertise in Enterprise Software and strategic product delivery, Pujarini has led large teams through dynamic business environments. She has excelled in managing complex programs, overseeing Product Development, and driving Customer Acquisition. Her domain spans Finance, Banking, Sales, Media & Robotic Process Automation.

She is currently working as a Senior Engineering Manager Lead at Microsoft, Power Platform Customer Advisory Team. In her previous role, she worked as an Associate Director at Novartis, Co-founder at InteGen IT Services, Wells Fargo, Tech Mahindra, and Microsoft, showcasing her impact in architecture, automation, and leadership.

Pujarini holds an MCA from ICFAI University and completed the General Management Program at the Indian School of Business. Her passion for innovation and strategic leadership define her journey.

Abhishek Mittal is a Data Engineering & Analytics professional who has more than 9 years of experience in business intelligence and data warehousing space. He delivers exceptional value to customers by designing high-quality solutions and leading their successful implementations.

His work entails architecting solutions for complex data problems for various clients across various business domains, managing technical scope and client expectations, and managing implementation of the solution.

He is a Microsoft Azure, Power BI, Power Platform and Snowflake certified professional and works as Senior Architect with Nagarro. He is also a Microsoft Certified Trainer. He is very passionate towards learning and exploring new skills. He is gregarious in nature and always believes in sharing knowledge and helping others.

Acknowledgements

Annu Roy: My heartfelt gratitude goes out to my colleagues and mentors, Dr. Gaurav Arora and Deepak Gupta, in this journey, for showing the path, and then guiding continuously towards the goal.

My inspiration to undertake anything challenging always comes from my mother, Ms. Krishna Roy, who brought up four of us and provided the best education to us despite not having formal education herself. I continue to draw my courage and belief to be able to undertake difficult tasks from her.

The most special encouragement and motivation comes to me from my son, who always inspires me to keep up with the digital generation.

Thanks to BPB for giving us this platform to share knowledge and grow ourselves in the process. I would also like to thank Rishiraj Deb and Gaurav Arora for their valuable contribution towards making this endeavour a success, and the technical experts, reviewers and editors who made every effort to help us produce a quality outcome.

Last but not the least, special gratitude to the readers for showing their interest in this topic. I wish you a pleasant learning experience while reading this book.

Rishiraj Deb: I would like to express my heartfelt gratitude to everyone who contributed to the creation of this book. Your unwavering support and encouragement have been a constant source of strength throughout this writing journey for me.

First and foremost, I want to thank the publisher for believing in the potential of this work and providing the platform to share my knowledge and thoughts with the world.

To my dear mother, Anita Deb, and brother, Biswapriya Deb, your belief in my abilities has fuelled my determination to bring this book to life.

Finally, my deepest appreciation goes to my loving wife, Deboshree Deb. Your understanding, patience, and continuous encouragement have been invaluable during the countless hours I spent writing this book.

To my co-authors, thank you for your collaboration and contributions. Your insights and discussions have enriched the ideas within these pages.

To all the readers, it is with utmost sincerity that I share this acknowledgement. I hope that the book proves to be useful to you, just as it has been shaped by my own experiences and learnings.

Thank you all for being a part of this incredible journey with me. Your support means the world.

Gaurav Aroraa: In life, it is hard to understand things when you do not find support. My family is one such support system, and I am the luckiest

to have them. I would like to thank my wife, Shuuby Arora, and my little angel, Aarchi Arora, who gave me permission to write and invest time in this book.

A special thanks to the BPB team. Also, a big thanks to Rishiraj Deb, Annu Roy, and Rajesh Kumar. It was a long journey of revisiting this book, with valuable participation and collaboration of reviewers, technical experts, and editors.

I would also like to acknowledge the valuable contribution of my colleagues and co-worker during many years working in the tech industry, who have taught me so much and provided valuable feedback on my work.

Finally, I would like to thank all the readers who have taken an interest in my books and for their support in making it a reality. Your encouragement has been invaluable.

# Preface

Data visualization and analysis is a critical task for enterprises that require a comprehensive understanding of the latest technologies and programming languages. Power BI is one of the tools that have become increasingly popular in the field of data analysis and visualization for enterprises.

This book is designed to provide a comprehensive guide to touch every shore of Data Analytics and Visualization with the help of Power BI. It covers a wide range of topics, including Data discovery (ETL capabilities etc.), Data Modeling, Report Distribution and management, covering business use-cases.

Throughout the book, you will learn about the key features of Power BI and various analytic tools. You will also learn about best practices for building enterprise level analytics with numerous practical examples to help you understand the concepts.

This book is intended for developers who are new in the field of analytics and want to excel with the power of these tools. It is also helpful for experienced developers who want to expand their knowledge of these technologies/tools and improve their skills in building robust and reliable applications for data visualization and analytics.

With this book, you will gain the knowledge and skills to become a proficient developer in the field of enterprise development using Power BI

and related tools.

Chapter 1: Getting Started with Power BI - elaborates the need of data visualization in today's data driven business world and explains why Power BI can be a tool of choice. The building blocks of the tool will be discussed here along with a quick overview of each of them. It would hand-hold the readers with the required installation processes.

Chapter 2: Data Discovery Using Power Query - discusses the features of Power Query and the capability of connecting to different data sources. The common data transformations would be discussed to shape the data as per business needs. Readers would get introduced to the M (Mashup) language and its uses in the context of Power Query.

Chapter 3: Data Modeling in Power BI - explains about modelling data in Power BI desktop. Power BI works best with a star schema instead of a big denormalized table. Here we elaborate the star schema design and its relevance to develop Power BI data models optimized for performance and usability. Also, different ways of implementing custom calculation using Data Analysis Expressions (DAX) would be discussed to enhance the model.

Chapter 4: Visualizing Data in Power BI - elaborates how to create visualizations in Power BI. Commonly used visuals will be discussed in detail along with different formatting options to impart the knowledge of creating a positive user experience. This chapter also discusses the default visuals and the visuals from the marketplace.

Chapter 5: Managing Reports in Power BI Service - gives special attention with explanation of Power BI reports, hosted on an Azure SaaS platform called Power BI Service. This chapter focuses on covering important topics from the service including workspace management, access control etc. along with data gateways and why to use them. This chapter also explains topics like dataflows and apps along with outlines of how to implement data security in Power BI.

Chapter 6: Working with Large Data Volumes - discusses concepts of big data reporting, which is a pressing need across many industries, thanks to the ever-growing volume of data. The chapter discusses a few of the key capabilities and features of Power BI for handling very large amounts of data. A few external tools that can be integrated easily with Power BI would be discussed here - which help in terms of source control, scripting & metadata deployment.

Chapter 7: DAX Reference Guide - provides a comprehensive guide of Data Analysis Expressions The chapter also explains the various commonly used DAX functions, using suitable examples. Readers can also refer to the expressions and customize them as per the business requirement.

Chapter 8: Use Case- Creating a Risk Report in Power BI - discusses how to make use of the concepts learnt with the help of a real-life business use-case.

Coloured Images

Please follow the link to download the

Coloured Images of the book:

https://rebrand.ly/k43yijh

We have code bundles from our rich catalogue of books and videos available at Check them out!

## Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at :

business@bpbonline.com for more details.

At you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

---

## Piracy

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

## If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

## Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

# CHAPTER 1

# Getting Started with Power BI

## Introduction

Introducing readers to the world of business intelligence, and one of the most popular tools in this domain which is Power BI. In this chapter, we will try to understand why we need Power BI in the first place and what different dimensions of problems it addresses. We will get familiarized with the tool and the components, see how to set up the environment as well as understand the cost factors involved with different licensing options, which should help us to create a solid foundation for the journey we begin with this book.

## Structure

In this chapter, we will discuss the following topics:

Importance of visualization

Introduction to Power BI

Building blocks of Power BI

Installing Power BI

Quick tour of Power BI components

Flow of work in Power BI

Power BI licensing details

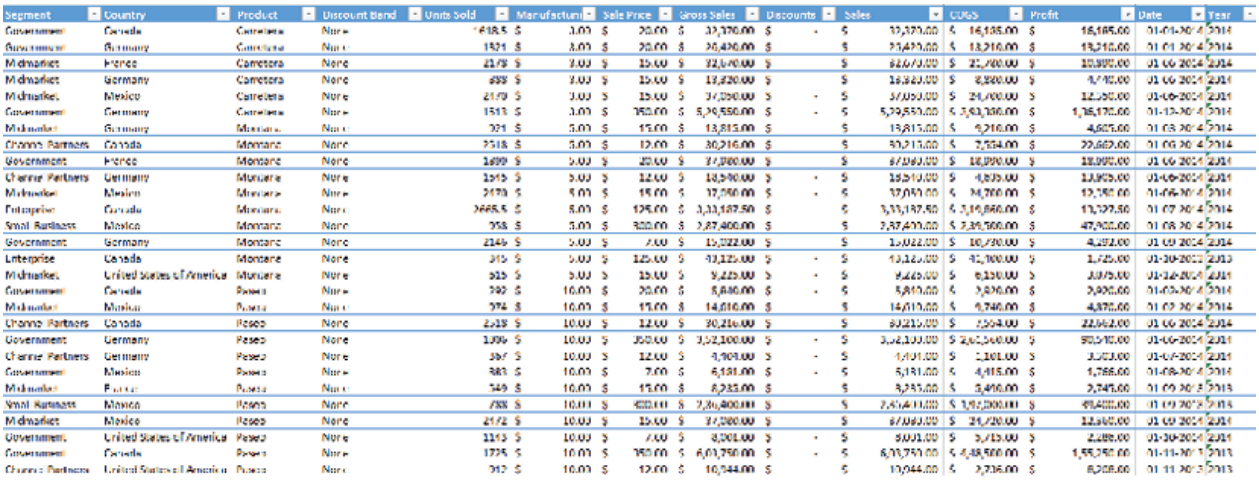
## Objectives

This chapter aims to make the readers understand the core concepts of the tool without covering individual topics in detail. By the end of the chapter, readers should be familiar with the common terminologies and have an understanding of the general working principles. This should help us to dive deep into individual components in the subsequent chapters of the book.

## Importance of visualization

Data is the new currency – maybe all of us who work with data are familiar with this phrase. However, it is not of much use if the data is not structured or modeled and cannot be visualized. Our minds remember and understand visuals in a much better way in comparison with flat tabular data, and hence, comes the need of creating visuals from raw data, which should be representative of the dataset on which they are built.

If we want to understand this concept further with an example, let us consider the small sample dataset as illustrated in Figure



| Segment | Country | Product | Discount Band | Units Sold | Manufacturing Price | Sale Price | Gross Sales | Discounts | Sales | COGS | Profit | Date | Year |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Government | Canada | Carretera | None | 1618.5 | $ 3.00 | $ 20.00 | $ 32,370.00 | $ - | $ 32,370.00 | $ 16,185.00 | $ 16,185.00 | 01-01-2014 | 2014 |
| Government | Germany | Carretera | None | 1321 | $ 3.00 | $ 20.00 | $ 26,420.00 | $ - | $ 26,420.00 | $ 13,210.00 | $ 13,210.00 | 01-01-2014 | 2014 |
| Midmarket | France | Carretera | None | 2178 | $ 3.00 | $ 15.00 | $ 32,670.00 | $ - | $ 32,670.00 | $ 21,780.00 | $ 10,890.00 | 01-06-2014 | 2014 |
| Midmarket | Germany | Carretera | None | 888 | $ 3.00 | $ 15.00 | $ 13,320.00 | $ - | $ 13,320.00 | $ 8,880.00 | $ 4,440.00 | 01-06-2014 | 2014 |
| Midmarket | Mexico | Carretera | None | 2470 | $ 3.00 | $ 15.00 | $ 37,050.00 | $ - | $ 37,050.00 | $ 24,700.00 | $ 12,350.00 | 01-06-2014 | 2014 |
| Government | Germany | Carretera | None | 1513 | $ 3.00 | $ 350.00 | $ 529,550.00 | $ - | $ 529,550.00 | $ 363,100.00 | $ 166,450.00 | 01-12-2014 | 2014 |
| Midmarket | Germany | Montana | None | 921 | $ 5.00 | $ 15.00 | $ 13,815.00 | $ - | $ 13,815.00 | $ 9,210.00 | $ 4,605.00 | 01-03-2014 | 2014 |
| Channel Partners | Canada | Montana | None | 2518 | $ 5.00 | $ 12.00 | $ 30,216.00 | $ - | $ 30,216.00 | $ 7,554.00 | $ 22,662.00 | 01-06-2014 | 2014 |
| Government | France | Montana | None | 1899 | $ 5.00 | $ 20.00 | $ 37,980.00 | $ - | $ 37,980.00 | $ 18,990.00 | $ 18,990.00 | 01-06-2014 | 2014 |
| Channel Partners | Germany | Montana | None | 1545 | $ 5.00 | $ 12.00 | $ 18,540.00 | $ - | $ 18,540.00 | $ 4,635.00 | $ 13,905.00 | 01-06-2014 | 2014 |
| Midmarket | Mexico | Montana | None | 2470 | $ 5.00 | $ 15.00 | $ 37,050.00 | $ - | $ 37,050.00 | $ 24,700.00 | $ 12,350.00 | 01-06-2014 | 2014 |
| Enterprise | Canada | Montana | None | 2665.5 | $ 5.00 | $ 125.00 | $ 333,187.50 | $ - | $ 333,187.50 | $ 319,860.00 | $ 13,327.50 | 01-07-2014 | 2014 |
| Small Business | Mexico | Montana | None | 958 | $ 5.00 | $ 300.00 | $ 287,400.00 | $ - | $ 287,400.00 | $ 239,500.00 | $ 47,900.00 | 01-08-2014 | 2014 |
| Government | Germany | Montana | None | 2146 | $ 5.00 | $ 7.00 | $ 15,022.00 | $ - | $ 15,022.00 | $ 10,730.00 | $ 4,292.00 | 01-09-2014 | 2014 |
| Enterprise | Canada | Montana | None | 345 | $ 5.00 | $ 125.00 | $ 43,125.00 | $ - | $ 43,125.00 | $ 41,400.00 | $ 1,725.00 | 01-10-2013 | 2013 |
| Midmarket | United States of America | Montana | None | 615 | $ 5.00 | $ 15.00 | $ 9,225.00 | $ - | $ 9,225.00 | $ 6,150.00 | $ 3,075.00 | 01-12-2013 | 2013 |
| Government | Canada | Paseo | None | 292 | $ 10.00 | $ 20.00 | $ 5,840.00 | $ - | $ 5,840.00 | $ 2,920.00 | $ 2,920.00 | 01-03-2014 | 2014 |
| Midmarket | Mexico | Paseo | None | 974 | $ 10.00 | $ 15.00 | $ 14,610.00 | $ - | $ 14,610.00 | $ 9,740.00 | $ 4,870.00 | 01-03-2014 | 2014 |
| Channel Partners | Canada | Paseo | None | 2518 | $ 10.00 | $ 12.00 | $ 30,216.00 | $ - | $ 30,216.00 | $ 7,554.00 | $ 22,662.00 | 01-06-2014 | 2014 |
| Government | Germany | Paseo | None | 1006 | $ 10.00 | $ 350.00 | $ 352,100.00 | $ - | $ 352,100.00 | $ 261,560.00 | $ 90,540.00 | 01-06-2014 | 2014 |
| Channel Partners | Germany | Paseo | None | 367 | $ 10.00 | $ 12.00 | $ 4,404.00 | $ - | $ 4,404.00 | $ 1,101.00 | $ 3,303.00 | 01-07-2014 | 2014 |
| Government | Mexico | Paseo | None | 883 | $ 10.00 | $ 7.00 | $ 6,181.00 | $ - | $ 6,181.00 | $ 4,415.00 | $ 1,766.00 | 01-08-2014 | 2014 |
| Midmarket | France | Paseo | None | 549 | $ 10.00 | $ 15.00 | $ 8,235.00 | $ - | $ 8,235.00 | $ 5,490.00 | $ 2,745.00 | 01-09-2013 | 2013 |
| Small Business | Mexico | Paseo | None | 788 | $ 10.00 | $ 300.00 | $ 236,400.00 | $ - | $ 236,400.00 | $ 157,200.00 | $ 79,200.00 | 01-09-2013 | 2013 |
| Midmarket | Mexico | Paseo | None | 2472 | $ 10.00 | $ 15.00 | $ 37,080.00 | $ - | $ 37,080.00 | $ 24,720.00 | $ 12,360.00 | 01-09-2014 | 2014 |
| Government | United States of America | Paseo | None | 1143 | $ 10.00 | $ 7.00 | $ 8,001.00 | $ - | $ 8,001.00 | $ 5,715.00 | $ 2,286.00 | 01-10-2014 | 2014 |
| Government | Canada | Paseo | None | 1725 | $ 10.00 | $ 350.00 | $ 603,750.00 | $ - | $ 603,750.00 | $ 448,500.00 | $ 155,250.00 | 01-11-2013 | 2013 |
| Channel Partners | United States of America | Paseo | None | 912 | $ 10.00 | $ 12.00 | $ 10,944.00 | $ - | $ 10,944.00 | $ 2,736.00 | $ 8,208.00 | 01-11-2013 | 2013 |

Figure Sample Raw Data

Now, let us understand the data first.

This is sales data for products across different segments, which includes information like manufacturing cost, sales price, number of units sold, gross

sales, total cost, profit, discount information in the case offered for any product, date of the transaction and the country where the transaction took place.

Looking at the data one can say it is not really intuitive in terms of deriving insights, isn't it?

Considering the fact that in a real-life business scenario, for potentially millions of records, someone cannot just take a glance at the data and make something out of it. It requires queries to pass to this dataset or table, to retrieve some meaningful information. Data visualization addresses this issue and enables users to gain meaningful insights from a plethora of data, in a quick, universal, and efficient way.

Now, let us create some visuals using the same records and see how that helps as shown in Figure

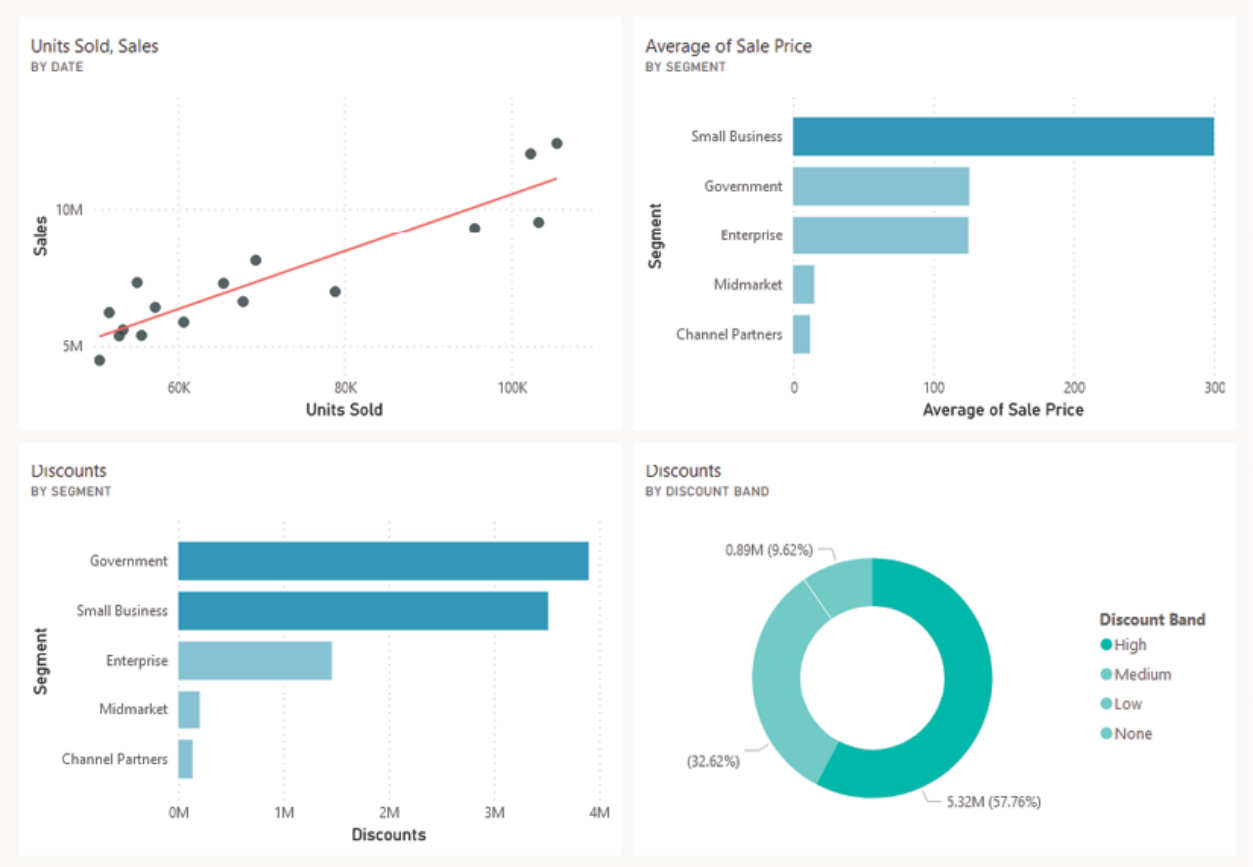



Figure Visualizing the sample raw data

Looking at the above visuals, one can instantly draw the following conclusions:

There is an obvious correlation between units sold and the sales amount.

The segment Small Business has significantly more Sale Price.

Government & Small Business segments also offered more discounts than others.

Cases where discounts have been offered, High accounts for the majority.

In today’s fast-paced world of ever-growing data, it would increasingly become difficult to analyse and gain insights from flat tabular data. Hence, effective visualization is becoming the pressing need for telling a story which is based on and can be substantiated by data, also helps businesses across industries to find actionable insights.

Though visualization is important, alone it cannot meet the diverse business intelligence needs of today’s enterprises. In the next section, we will see the common requirements of business intelligence projects and understand why Power BI can be a tool of choice.

## Introduction to Power BI

As we have seen now why visuals are important, that is not always the only outcome looked for in a report. Visualization is a very important aspect of analytical reports, where we analyse the existing data and try to find trends, patterns in it which otherwise is not intuitively available. Also, it plays an important part in telling a story which best describes the data we are working on and the key outcomes we can get out of it.

However, the requirement might be an operational kind of report to track the day-to-day operational activities of an organization. Probably, a table is all that would be required to be used as a visual in this case, what would matter more is the ability to integrate with the database or data warehouse that stores the data and the ability to process large data volumes.

For a wide range of business users, Excel has been the most popular tool for decades to analyse data and report. Maybe the requirement is to move the users from Excel-based report silos to a centralized online reporting framework. Often one of the biggest challenges with these kinds of scenarios is that people have invested so much time and effort in perfecting their Excel skills that they are reluctant to come out of their comfort zone. Here, it might be wiser not to force users into an abrupt change, rather we need to create something which closely mimics their existing ways of working so that they can self-serve and eventually move to a more robust BI (business intelligence) solution.

Power BI is a business intelligence tool that can meet all the requirements mentioned above, and many more! It excels in:

Integration: Power BI has hundreds of different data source connectors which help to integrate with various platforms for reporting.

Transformation: The data available in real-life business use cases is often far from what is ideal for a data model. Hence, it needs to be shaped and transformed to meet the modeling requirements. Power BI has a rich set of ready-to-use transformations using just a few button clicks, with the capability of easily customizing those transformations as and when required.

Data Modeling: It is easy to create different data models in Power BI; also one can easily enhance those models with custom calculations using an expression language (DAX) which is very similar to Excel formulas in terms of syntax.

Visualization: Power BI has a wide array of visuals readily available for the users once they install the tool. Additionally, there is a marketplace for certified free and paid visuals created by third parties which users can import in the development environment by matter of a click. If that is not enough, there are provisions for creating custom visuals using R or Python scripting!

Distribution: There is a Software as a Service (SaaS) platform built in Azure cloud, called Power BI which hosts all the published Power BI reports and provides different options for collaboration & management.

Security: Multiple layers of data security can be implemented in Power BI which includes access control on Power BI Service as well as role-based security for individual records of a table.

Support Framework: Being a Microsoft product, it has excellent integration with other Microsoft Power Platform components like Power Apps and Power Automate, which enables to create end-to-end solutions. Also, it has integration with multiple external tools which can be used for a range of activities starting from model optimization to big data processing.

Licensing Options: Power BI Desktop is a free software which anyone can download, install, and start learning which eliminates the need for expensive sandboxes. There are other user-based and capacity-based licensing options which can be adopted on a need-to-have basis.

Convenience of Use: It’s convenient to try out and learn, easy to get support from online communities and one can start value addition to the work in a matter of hours!

The increasing popularity and adoption of Power BI by every size of enterprise across industries is a testimonial of how useful the product can be in terms of automating routine project reports as well as deploying enterprise-scale BI solutions.

## Building blocks of Power BI

The basic elements or building blocks of Power BI content can be categorized as:

Datasets are the collection of data on top of which the Power BI contents are built.

These are visuals that are created based on the underlying dataset. They can be a table, a graph, a chart, a card, or any other visual representation of data.

Reports are a collection of visuals, in one or more pages, used to convey insights about the data in the dataset. Reports can be directly consumed by the users or can be distributed via Apps.

Dashboards are single pages which bring together visuals from one or more reports. Reports are usually function specific, for example, Sales Reports, Inventory Reports, and so on. while dashboards can provide quick overviews across the spectrum of reports.

Tiles are individual visuals of a dashboard. They can separately be interacted with and can be used to trigger data-based alerts.

A pictorial representation of Power BI building blocks can look like Figure

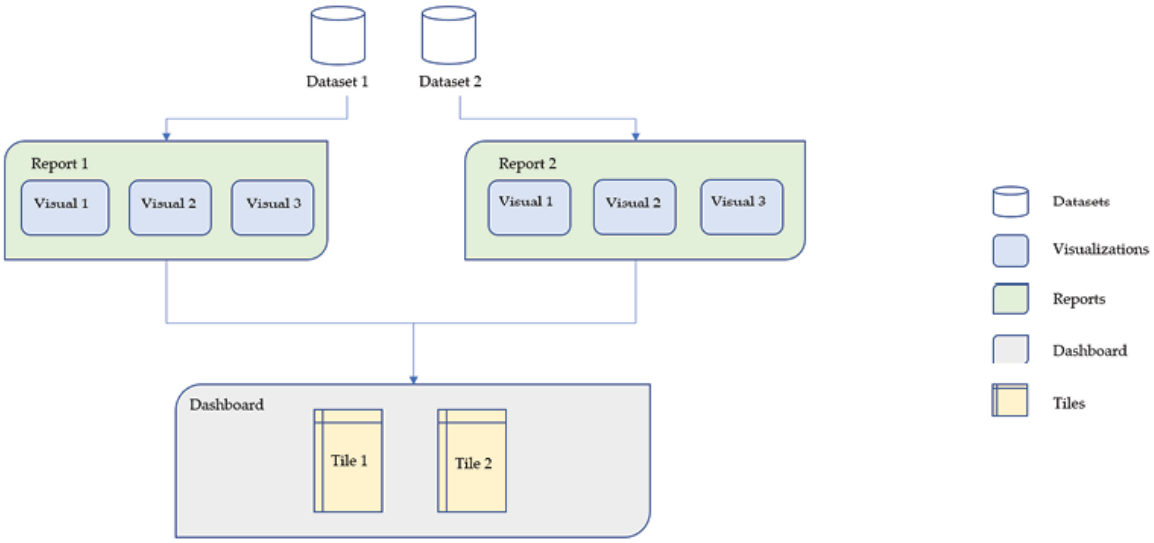



Figure Power BI Building Blocks

These elements can be used in different components of the tool, which we will discuss more in the subsequent chapters of the book.

## Installing Power BI

Power BI Desktop is a free desktop application which can be downloaded and installed from the Microsoft official webpage. Contents can be developed in both Power BI Service (online) and Power BI Desktop (offline), while the latter provides more control and flexibility in terms of features and usability. Power BI Service does not require any installation and users can login to it using their organizational account. We will dive deep into Power BI Service later in the book, this section will focus on setting up the Power BI Desktop which also should be the preferred development environment for majority of the Windows users.

Power BI Desktop gets an update each month, to install the latest version please type in the following URL or link (or keywords) on your web browser of choice:

https://www.microsoft.com/en-us/download/details.aspx?id=58494

Keywords: Download Power BI Desktop

The link should direct you to a webpage where you should be able to see the Download option as shown in Figure

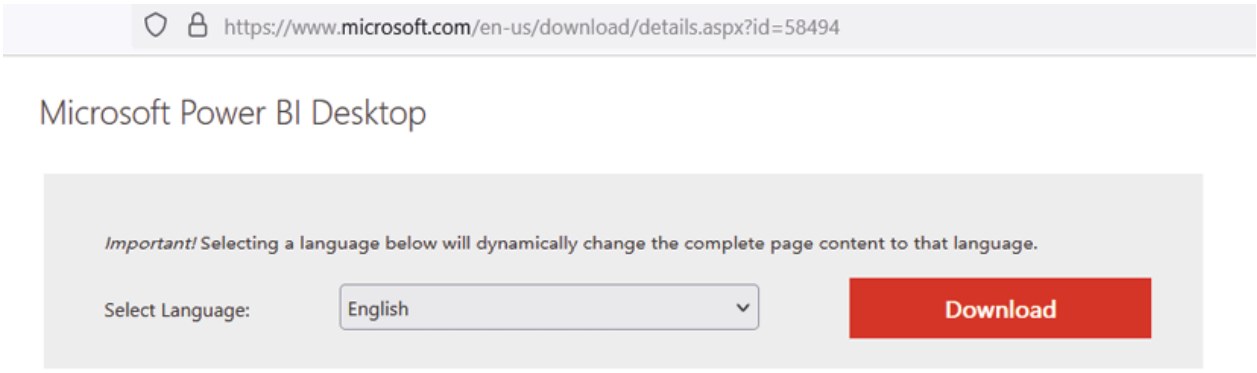



Figure Power BI Desktop download page

Clicking on the download button should redirect the user to the download options where the user needs to select the file depending on the bit version of Windows and proceed, refer to Figure

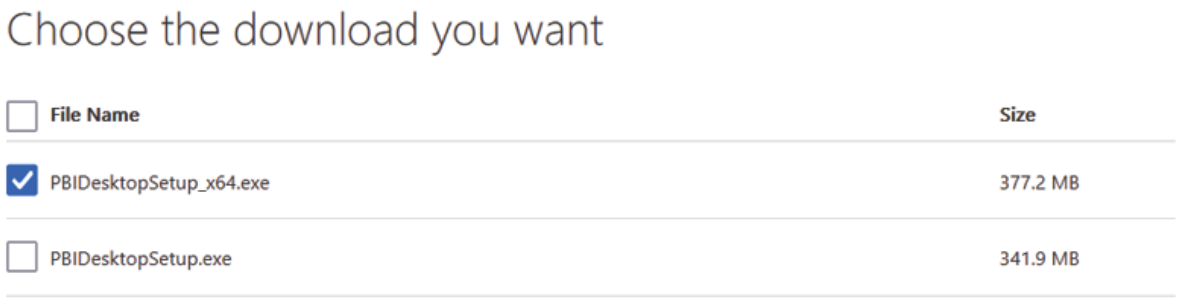



Figure Power BI Desktop download files

Tip: To check the Windows bit version, please follow the steps outlined in the given link:

https://support.microsoft.com/en-us/office/determine-whether-your-computer-is-running-a-32-bit-version-or-64-bit-version-of-the-windows-operating-system-aac162a1-0cb3-46f2-888f-2f22897396ce

Keywords: Check Windows bit version

This should download the executable installer on the user’s PC. The installer then should be executed by double clicking and the setup instructions need to be followed to complete the installation process.

Tip: During the installation process, please check on the ‘Create a Desktop Shortcut’ option to get the app launcher by default on the user’s desktop.

## Quick Tour of Power BI Components

Power BI has some layers or components that work in tandem to provide a complete BI solution experience. The primary components of Power BI can be categorized as:

Power Query or Query Editor

Power BI Desktop

Power BI Service

Now, let's do a quick tour of each of these components to understand what they actually do. The details of all these components will be discussed throughout the rest of the book in the subsequent chapters.

Power Query or Query Editor: Power Query is the component to perform the ETL of data from the data source systems, the data as per the need of the data model, the transformed data to the next layer) tasks that are required to shape the data for the requirement.

Power Query is integrated with Power BI Desktop and gets installed at the same time.

Once Power BI Desktop is installed with a desktop shortcut, the tool can be launched and logged into with a school or work account. The authentication

process is not mandatory at this stage and can be skipped to continue working with the tool, however logging in provides a more integrated experience in terms of working with Power BI Service. Power BI home screen can be seen in Figure

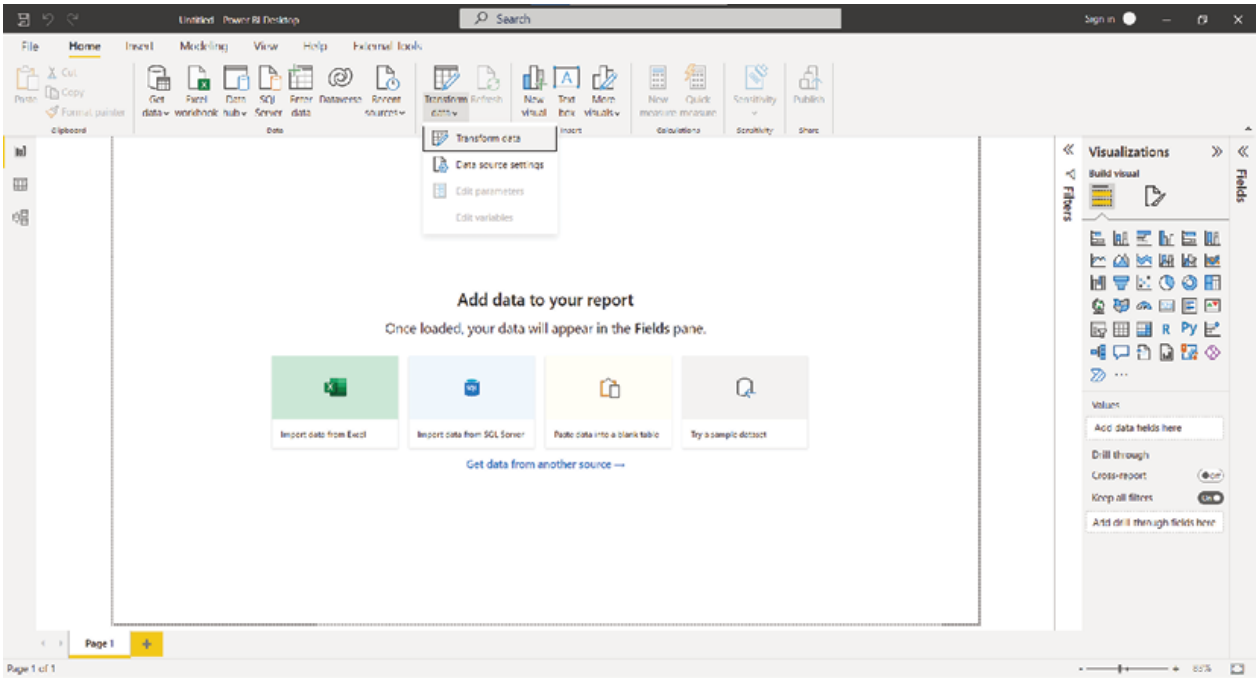



Figure Power BI Desktop home screen

On the home screen, from the ‘Transform Data’ option, the query editor can be launched. Figure 1.7 illustrates a few of the data transformations available in the Power Query editor once we connect to a data source and get some data:

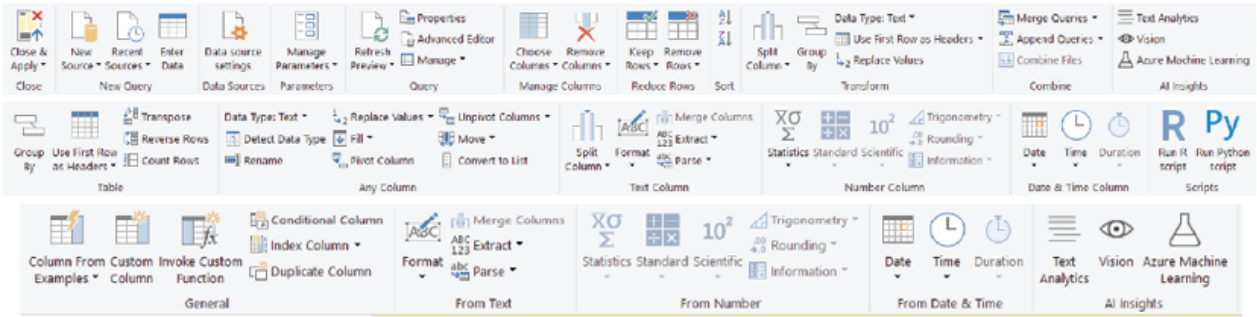


Figure Power Query transformation options

Power BI Desktop: Power BI Desktop is the development environment for most of the users where reports are being created and then published to the Power BI Service for collaboration. With Power BI Desktop, we can connect to a variety of data sources, shape and clean the data using Query Editor, model the data as required, and then create visuals on top of the model. Figure 1.8 illustrates the common functionalities of the desktop version:

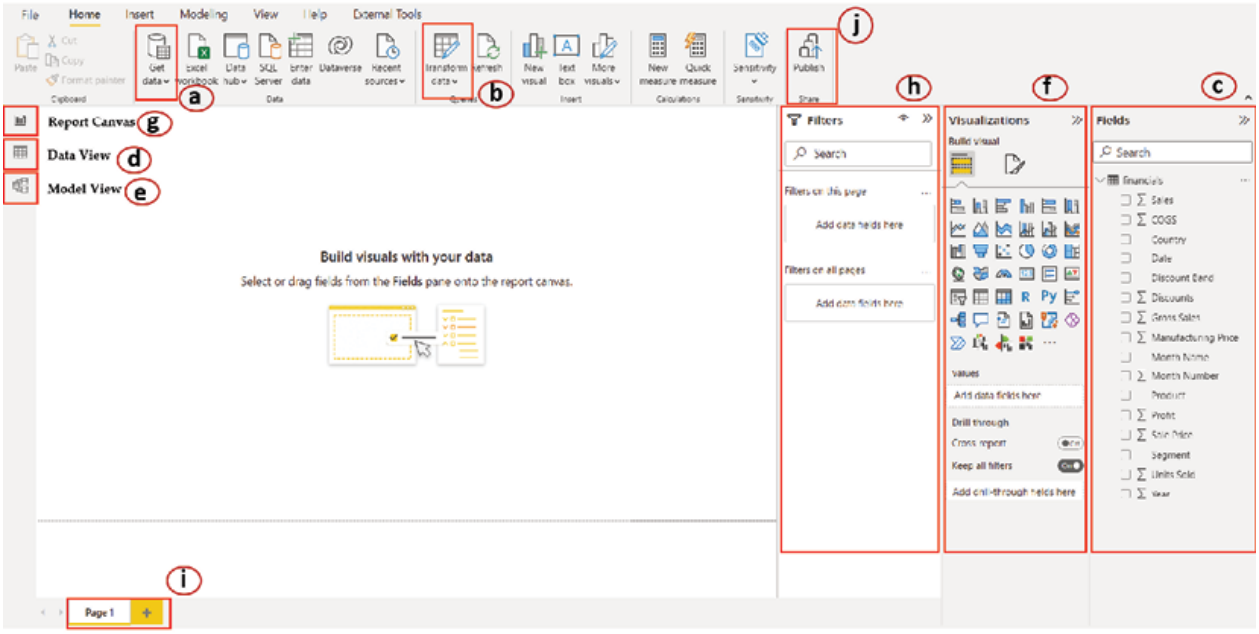



Figure Power BI Desktop common usage

The most common functionalities of Power BI Desktop are:
Get Data: The get data option allows users to connect to different data sources.
Transform Data: Using the transform data option, a query editor can be launched for data transformations.
Fields Pane: Under the fields pane, all the tables and columns would be visible to the users.
Data View: The data view can be used to view and validate the imported data.

Model View: The model view is used to manage relationships between tables and eventually create the data model.

Visualization Pane: From the visualization pane, visuals can be added to the canvas and then required fields can be added from the fields pane.
Report Canvas: This is where the magic happens! All the visuals would get created here to come up with a report page.
Filter Pane: Filters can be added here from the fields pane, which would get applied to the report or any specific visual.
New Page: This is used to create multi-page reports.
Publish: Using the publish button, reports can be published to the Power BI Service.

When a Power BI Desktop file is saved, it gets saved as a Power BI Desktop document (with an extension .pbix).

Power BI Service: Power BI Service is the cloud platform which hosts all the Power BI contents. It is a Software as a Service (SaaS) platform built on Azure cloud and managed by Microsoft.

The most basic elements of Power BI Service are Workspaces and Apps, as shown in Figure

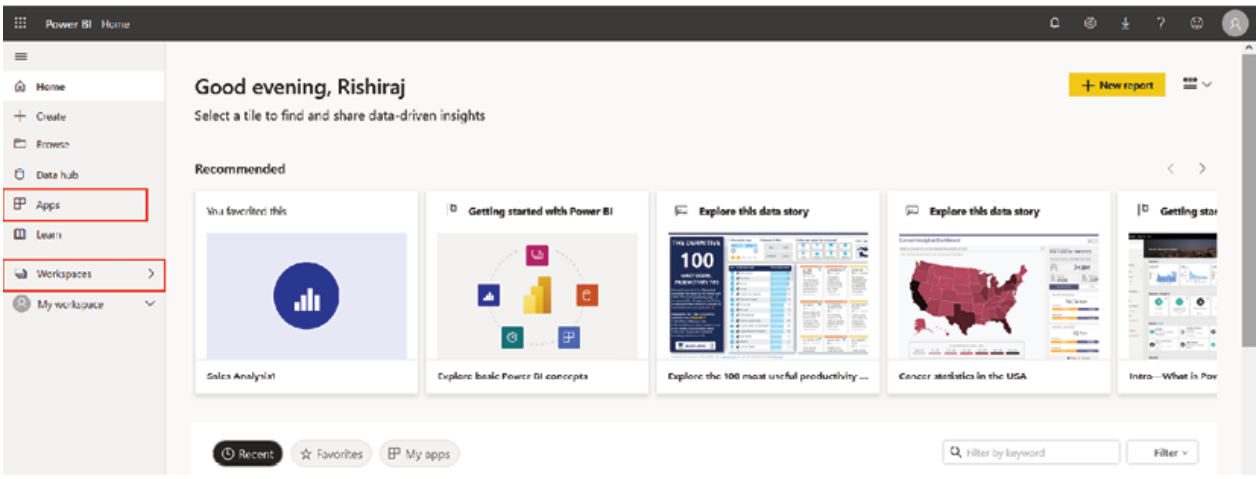


Figure Power BI Service home screen

Workspaces are containers for all the published Power BI reports. Users should create workspaces on the service with meaningful names, and while publishing reports from the desktop version, they need to select the corresponding workspace where they wish the report to get published. Basic elements that a workspace can contain are:

Datasets: When a .pbix file gets published, the underlying imported data gets uploaded to the service as a dataset in the workspace. We can schedule refreshes for the dataset which will update the report(s) that are attached to it.

Reports: This is the report page(s) created in the desktop version and attached to the underlying dataset. The reports can be further modified in the service, or even created from scratch on the service itself.

Dashboards: Collections of visuals from one or more reports on a single page which helps to provide an overview of the business.

DataFlows: This is a component having similar capabilities to Power Query; mostly useful in scenarios when we want to do centralized transformations on imported data.

Apart from the workspaces, another important (but optional) element of Power BI Service is the Apps. Apps are collections of dashboards and reports in one place, which is helpful in terms of content navigation as well as controlling access.

In this section, we went through the basic elements and components of the tool, mostly for becoming familiar with the terminologies and also for getting an idea of how things work. This basic understanding should help us when we move forward along with the book.

## Flow of work in Power BI

There can be multiple different ways of designing a Power BI project based on requirements and preferences, however in a typical reporting scenario, it all starts with Power BI Desktop. First, a connection is established with the data source, then required transformations are done in the query editor to shape the data as required. In the next step, the data is modeled in Power BI Desktop and a report is created. Finally, the report is published in a workspace in Power BI Service.

What remains then is to set up a refresh mechanism so that the reports can refresh as scheduled or on-demand basis. On Power BI Service, user credentials are set up using which the reports can refresh directly from the source. In the case of an on-premises data source, a data gateway is required to be used, which we will explore in detail in the upcoming chapters. The reports are then shared with the end users who can view and interact with them.

Figure 1.10 illustrates a common flow of work in Power BI:

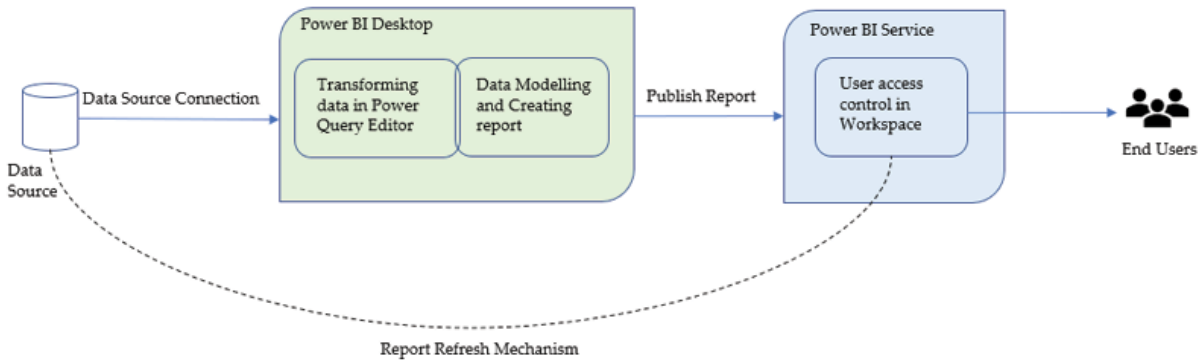


Figure Common flow of work in Power BI

Power BI licensing details

All users of Power BI Service need a license, while Power BI Desktop is a free-to-use application as of now. There are two types of licenses in Power BI Service; per-user-based and capacity based.

Per User Based: There are three user-based licenses:

Free License: Provides a personal sandbox environment to try out and learn Power BI. It is limited to content creation on personal workspace, users would not be able to share content or consume content created by others. Anyone logged in to Power BI Service using a work or school account would by default be assigned a free license on their tenant.

Pro License: Pro license allows you to create new workspaces, share content with other users and consume content shared by others. It costs a standard $9.99 per user per month and comes with most of the basic functionalities. Pro should be the license of choice for developers who do not really need premium features.

Limitations to consider: There is a 1-gigabyte limit for each model size & 10-gigabyte storage limit per user of the tenant. The scheduled refreshes of a model would be limited to 8 times a day.

Premium per user License: Comes with all the features of a Pro license, with additional premium capabilities like XMLA endpoint read/write connectivity (useful for big data processing), Application lifecycle management (using external tools like ALM Toolkit, Tabular editor), increased model size limit up to 100 gigabyte and total storage limit up to 100 Terabyte, maximum of 48 refreshes per day and so on. The standard cost is $20 per user per month.

Capacity Based: Commonly known as Power BI Premium, this is a dedicated capacity that can be assigned to individual workspaces across an organization. Premium workspaces provide all premium features (as in premium per-user license) and more! It also offers greater scale and performance for the contents. Additionally, it also enables users to embed content into external applications.

Apart from features, Power BI Premium also simplifies user access at an enterprise scale. Contents from a workspace backed by premium capacity can be consumed by free users as well, however, a Power BI Pro or Premium Per User would be required for publishing content. A common scenario where premium becomes cost-effective is where thousands of end users just need to consume reports or dashboards in Power BI. Here, it would probably make more sense to assign premium capacity to the workspaces so that free users can access the content while the developers can have Pro or PPU to publish content.

The cost of dedicated capacity starts from $4995 per month and can be scaled up based on requirements.

## Conclusion

In this chapter, we saw why visualizations are important and the common requirements that business intelligence projects have these days. We discussed why Power BI can be a tool of choice to meet the expectations. We walked you through the required steps for setting up the environment. The basic elements or building blocks of a Power BI report have been discussed along with the key components of the tool. We saw how a typical Power BI project is structured and went through the licensing details of the product. This should provide a holistic view of the framework which should help as we go along with the rest of the book.

In the next chapter, we will learn about performing transformations in the query editor and shaping data as per reporting requirements.

## Knowledge check

Power Query Editor helps in:

Transforming data

Creating visualization

Modeling data

Providing user access

You must sign in to your account while working with Power BI Desktop:

True

False

Which of the following is NOT an element of a workspace?

Dataset

Report

App

Dashboard

If a workspace is backed by premium capacity, then a free user would be able to publish a report to that workspace:

True

False

All ‘Knowledge Check’ answers are provided at the end of the book.

# CHAPTER 2

# Data Discovery Using Power Query

## Introduction

The ability to integrate with a wide variety of data sources is a distinctive feature of Power BI, which we are going to explore in detail in this chapter. After establishing a connection to the data sources, we will see which connectivity mode to use and how to shape the data in the Power Query editor. We are going to understand concepts like query folding to optimize our queries, explore data cleansing techniques, and see how to combine data from different data sources as well. Welcome to Power Query!

## Structure

In this chapter, we will discuss the following topics:

Connecting data sources

Data connectivity modes in Power BI

Query folding

Introduction to the Mashup

Common data transformations in the Query Editor

Data cleansing in the Query Editor

Appending and merging data

Power Query parameters

## Objectives

The objective of this chapter is to introduce the readers to the world of Power Query so that they can effectively perform data transformations required for their business reports without the help of their IT team! Business users understand the business requirements best and IT guys, obviously, have the technical skills! It is not always possible to cross-skill these people in a fast-paced environment, and hence, often roles are sought after who can bridge the gap and translate business requirements into technical specifications. Query Editor attempts to solve this problem and enables business users to integrate their reports with different data sources and performs effective transformations. However, to do all these efficiently, some understanding is required of the tool and related concepts. This chapter aims to provide that understanding, with minimal technical jargons.

## Connecting data sources

One of the main advantages of using Power BI as a reporting solution is its ability to integrate with a wide variety of data sources that are available today, and the number of supported data sources grows with new releases. Also, it allows to create custom connectors if required, enabling it to connect to virtually any data source. In this chapter, we will discuss the skills that are required to consume data in Power BI. Starting from the steps that need to be followed to connect to different data sources, we will discuss the connectivity modes, the transformation options available to shape the data and various other useful ETL techniques. Refer to Figure 2.35 for the sample dataset which has been used for many illustrations throughout the book.

Before visualizing data and creating reports, at first data needs to be loaded into the environment connecting one or more data sources. While connecting, we need to keep in mind an important design-related approach, which is, connectivity mode, which we will understand in the next section of this chapter. As of now, we will see how to effectively use the connectors to load data into Power BI from a few commonly used data sources to see how that works.

On Power BI Desktop, the ‘Get Data’ window can be launched using the get data option under the home tab, in the data group. The pop-up window lists all the default or out-of-the-box connectors Power BI supports. On the left hand, the connectors are categorized for ease of navigation. One can also search for a specific connector using the text search box provided.

The categories are:

File: for example, Excel, Text, JSON, SharePoint folder, and so on.

Database: SQL Server, Amazon Redshift, and so on.

Power BI datasets, Dataflows, and so on.

Azure: Azure SQL Database, Azure Synapse Analytics, and so on.

Online Services: SharePoint Online List, Microsoft Exchange Online, and so on.

Others: Generic connectors like OLE DB, ODBC, REST APIs, and so on.

Figure 2.1 shows the 'Get Data' option in Power BI Desktop:

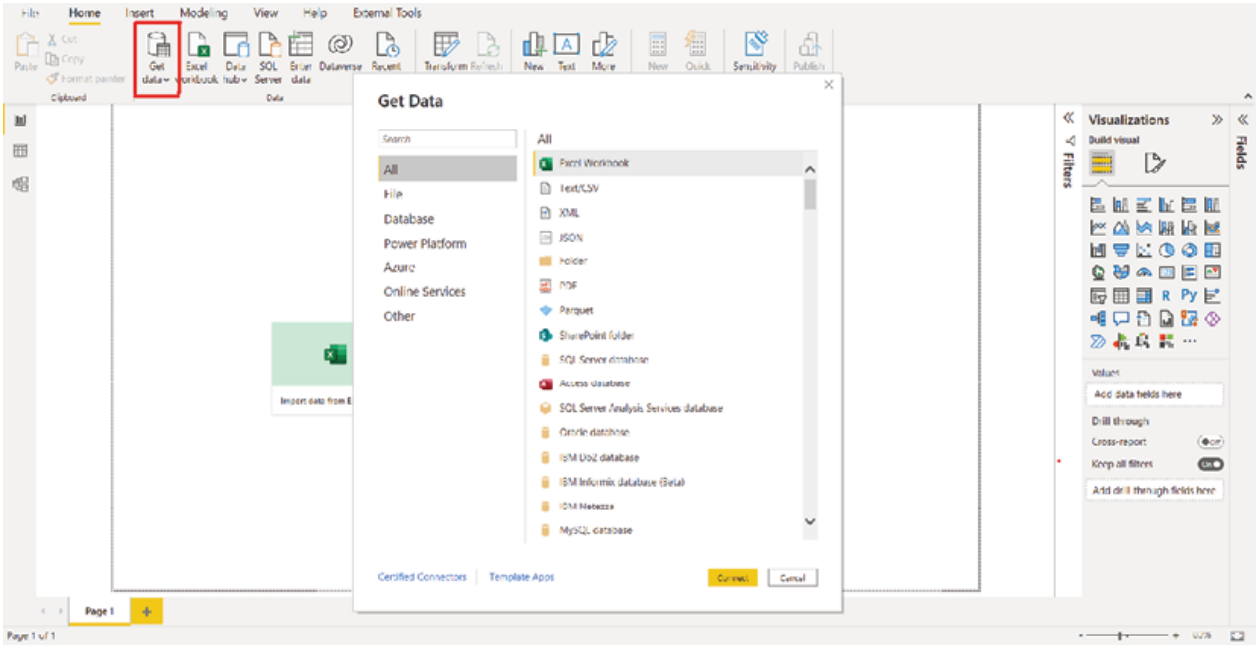



Figure 2.1: Power BI Get Data Window

Let us explore a few connectors from these categories to get familiarized with the concept:

Connecting a SharePoint folder: SharePoint has been one of the most popular repositories to store day-to-day project files, and hence, it often serves as a data source for Power BI reports. Let us assume that the data reside in multiple Excel workbooks and the files are saved in a SharePoint folder, from where we need to get the combined data and create a report.

From Power BI Desktop, Get Data | SharePoint Folder should launch the SharePoint folder connector. The site URL needs to be specified, which is the root URL of the SharePoint site, not including any subfolders. Figure 2.2 shows the SharePoint Folder connector:

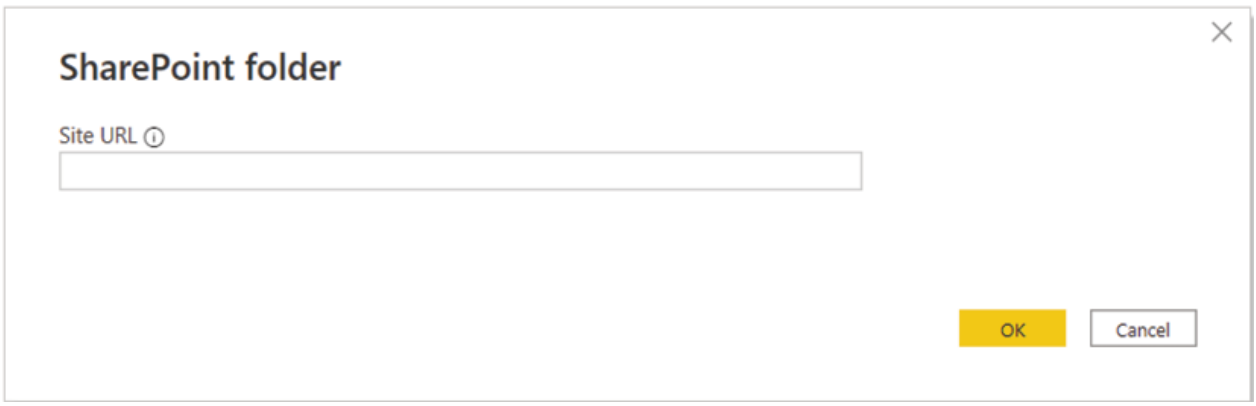



Figure 2.2: SharePoint Folder Connector

Tip: To find out the root URL of any SharePoint folder, open a page under that folder and click ‘Home’ on the left-hand navigation pane. The address of the home page should be the root URL of that site.

Figure 2.3 highlights SharePoint root URL*:

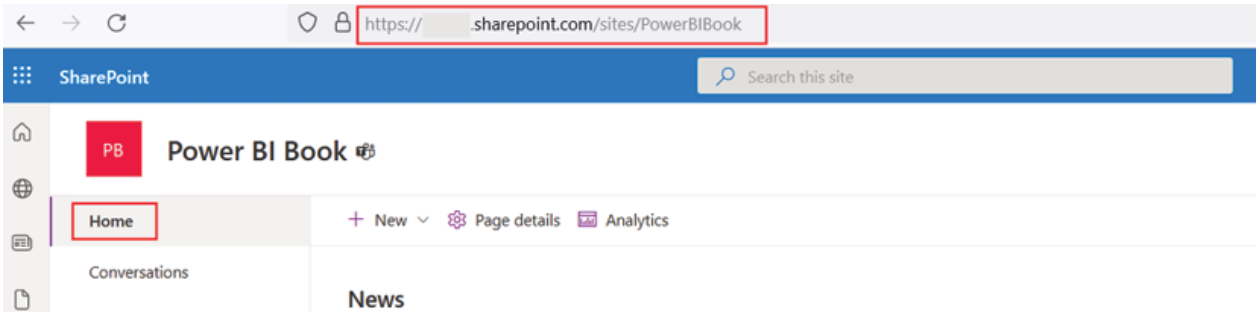



Figure 2.3: SharePoint Folder root URL

* Identifiable information are hidden on the images to maintain confidentiality

After specifying the root URL, clicking on 'OK' should take us to the credentials window where 3 options should be available to authenticate first-time site visitors. As an anonymous user, using Windows credentials or using a Microsoft account. After selecting the appropriate authentication method, the SharePoint folder can be connected by providing credentials and choosing the level these settings need to get applied to. Figure 2.4 shows the credential window:

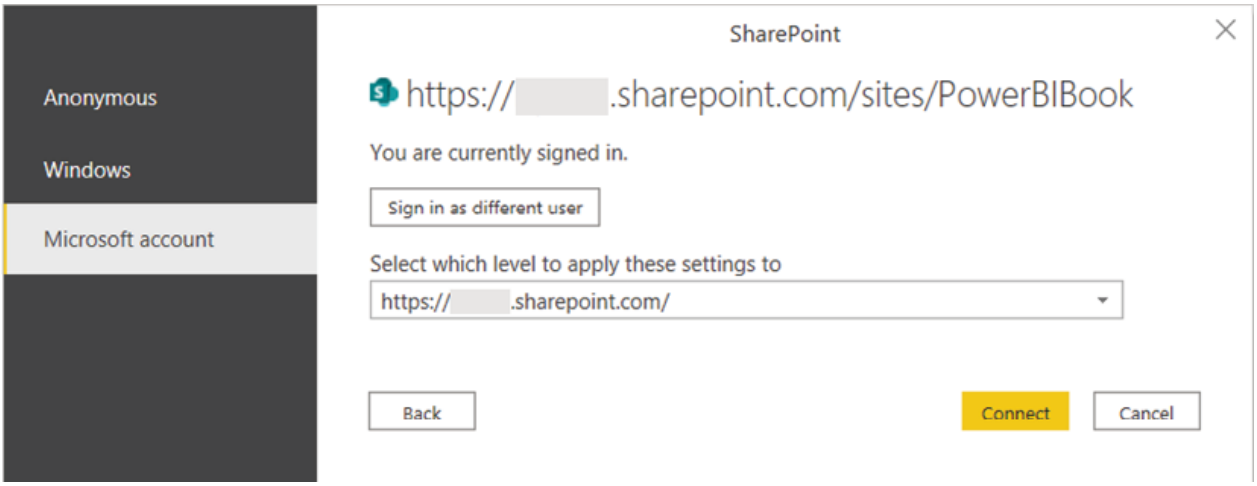


Figure 2.4: Credential Window of SharePoint Connector

Once connected, all files that are saved in that folder and associated subfolders should be visible in binary format under the ‘Content’ field, along with additional attributes like file name, file extension, creation date, folder path, and so on. At this point, one can choose either to combine files directly or load the file list to the Power Query editor using the ‘Transform Data’ option. File list on SharePoint folder as displayed in Figure

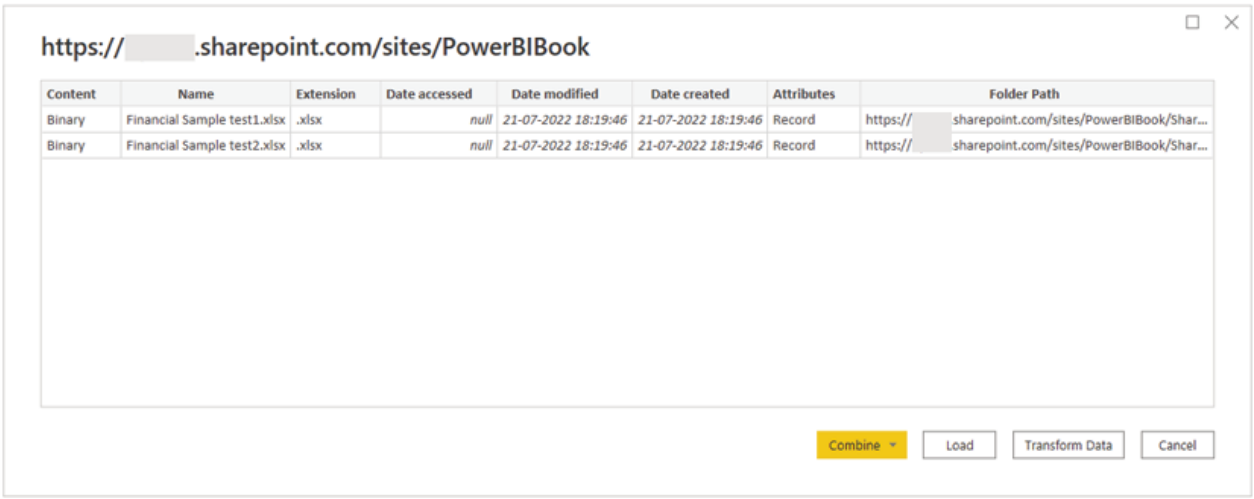



Figure 2.5: File list in SharePoint folder

In Query Editor, we can select the required files by applying appropriate filters and effectively combine those files in case the files have identical structures or schema. For example, a common scenario can be, files with the same metadata or schema get saved in the SharePoint folder on a daily basis through an automated process, and we need to report combining all those files. To apply the required filters effectively, the unique features of these files need to be identified. For instance, they can always be saved in a specific subfolder, or the file names can have specific keywords in them. For the first case, a filter can be applied on the ‘Folder Path’ field for a specific path. For the latter, a filter can be applied to the ‘Name’ field which should select only

the files containing that keyword as part of the file name. Whenever a new file would get added, the query would pick that up based on the applied filters, and the data would get included in the report during refresh accordingly.

Once all the required files are selected, click on the ‘Combine Files’ option on the home ribbon and Power BI will combine the data from selected binary contents and create a single table out of that. While combining files, one file needs to be chosen as a sample based on which the table structure gets defined. Figure 2.6 shows the options to combine files in the query editor:

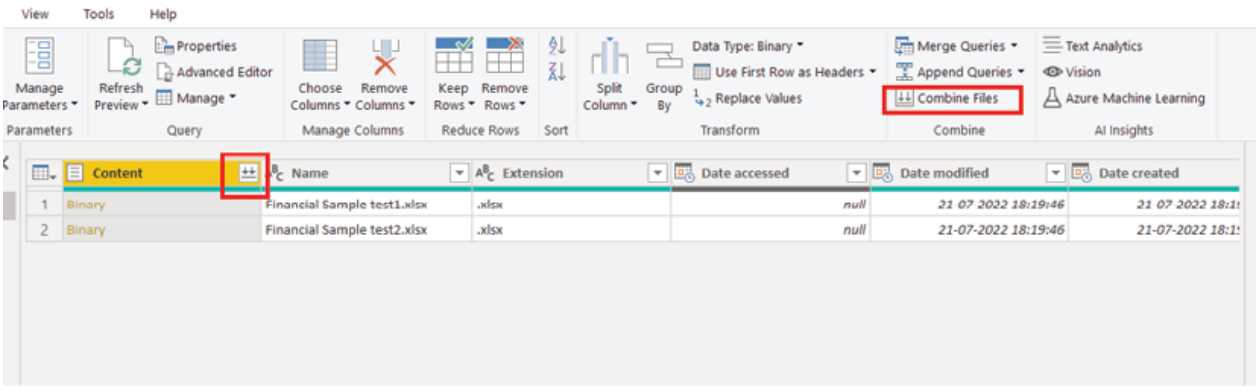



Figure Combining contents in Power Query editor

Once Power BI does its job of combining the files, the data can be previewed in a single table, as shown in Figure

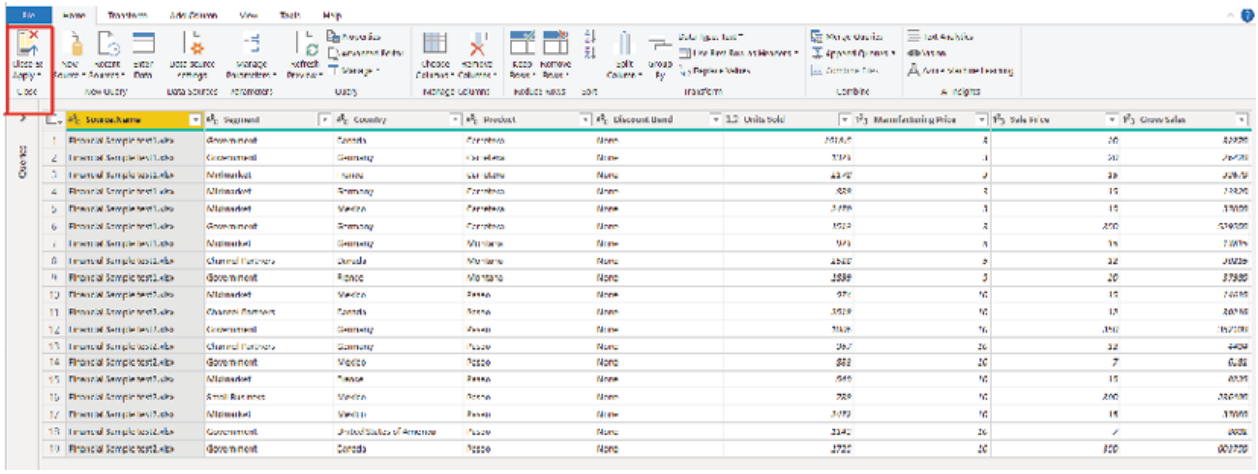


Figure 2.7: Combined data in Power Query table

The first column of the table ‘Source.Name’ can be referred to understand which record came from which file; however, if required, we can now safely get rid of that column at this point.

This table can now be loaded into the Power BI Desktop using the ‘Close & Apply’ option under the ‘Home’ tab for modeling and visualization, or we can continue performing further transformations in the query editor.

Connecting an Azure SQL Database: Microsoft Azure SQL database is a managed cloud database running on Power BI Desktop, Get Data > Azure SQL Database should launch the database connector.

The Azure SQL database connector is illustrated in Figure

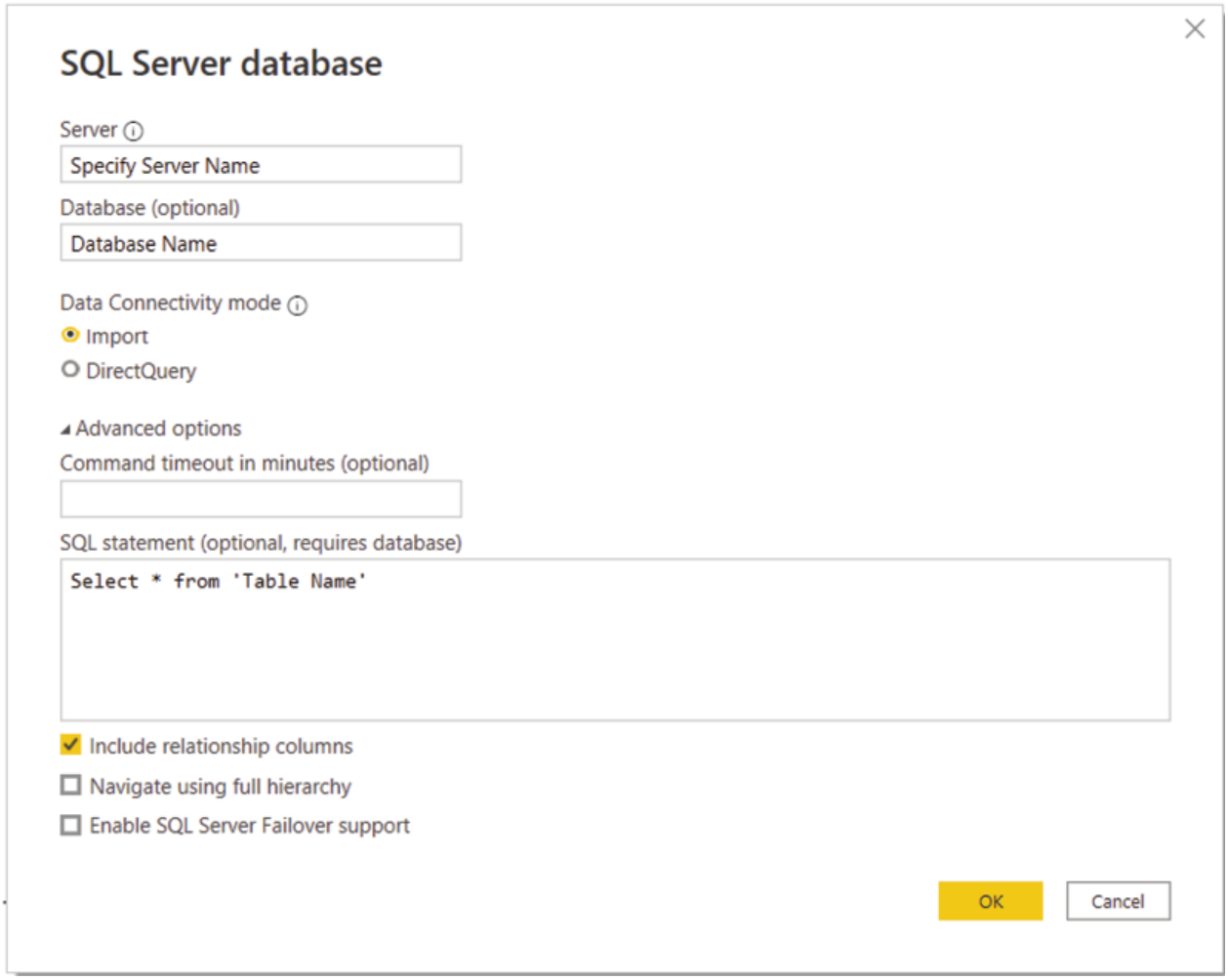



Figure 2.8: Azure SQL Database connector

Mandatorily, the server address and the connectivity mode need to be specified here, having ‘import’ as the default connectivity mode. Let us not go into connectivity mode in detail here as we are going to understand this in a separate section; and move ahead with the default selection which will load the data into Power BI Desktop. The name of the database is optional; without it, the connection would get established with the server and all the available databases would be visible based on user access, otherwise, the specified database would be connected to.

There are a few additional parameters available under advanced options:

The connector allows passing a SQL statement as optional, which would get executed on the server and load the query output to Power BI. In case we want to pass a SQL statement, the database name must be specified and cannot be left blank.

If a connection lasts longer than 10 minutes, it gets timed out by default. Using the optional ‘Command Timeout’ option, the connection can be kept open for longer if required, from the Power BI Desktop. This option should ideally be explored only in case the requirement demands.

Checking the ‘include relationship columns’ checkbox ensures to include columns having relationships with other tables.

The ‘navigate using full hierarchy’ option, if checked, the navigator displays the existing hierarchy of tables in the database.

The ‘SQL server failover support’ option, if checked, when a node in the Azure SQL failover group isn’t available, it can move to another node in case of a failover. This option again should ideally be explored only in case the requirement demands.

For most scenarios, going ahead with just the mandatory parameters (server name and connectivity mode) should be fine.

After providing all the required details, clicking ‘OK’ should take us to the credential window. Apart from the Windows credential and Microsoft account, one can authenticate using a database user as well. In case of connecting the server for the first time, after choosing the authentication method and providing credentials, clicking ‘Connect’ should establish a

connection to the database server. Figure 2.9 shows the credential window of the Azure SQL database:

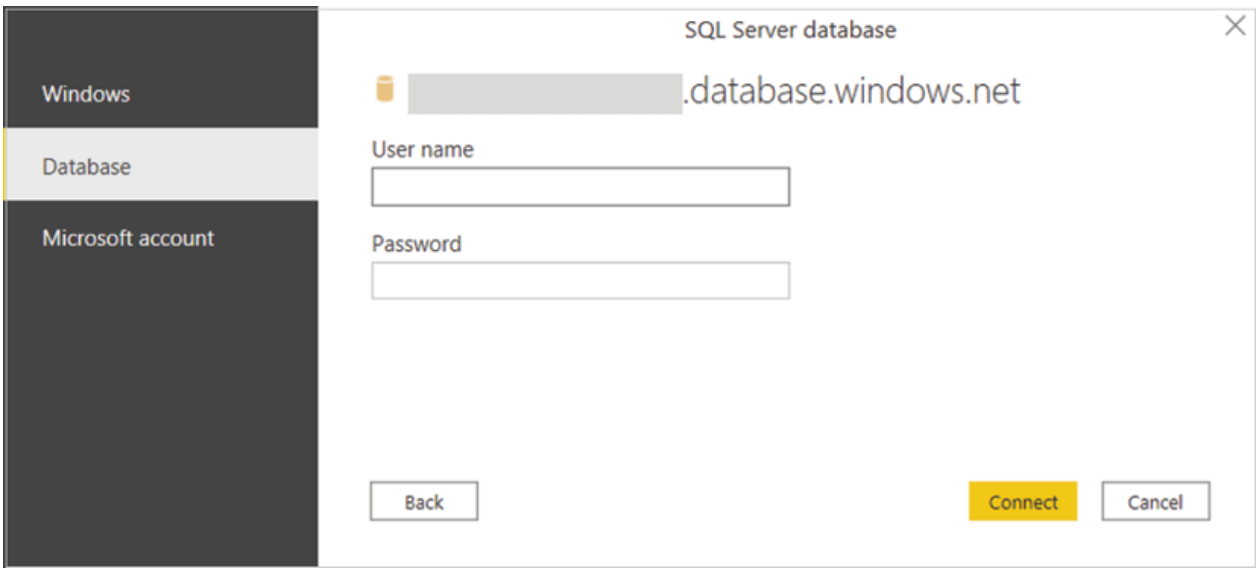



Figure 2.9: Credential window

Once connected to the database, all available database objects should be visible on the left-hand side panel. The preview of the database object is shown in Figure

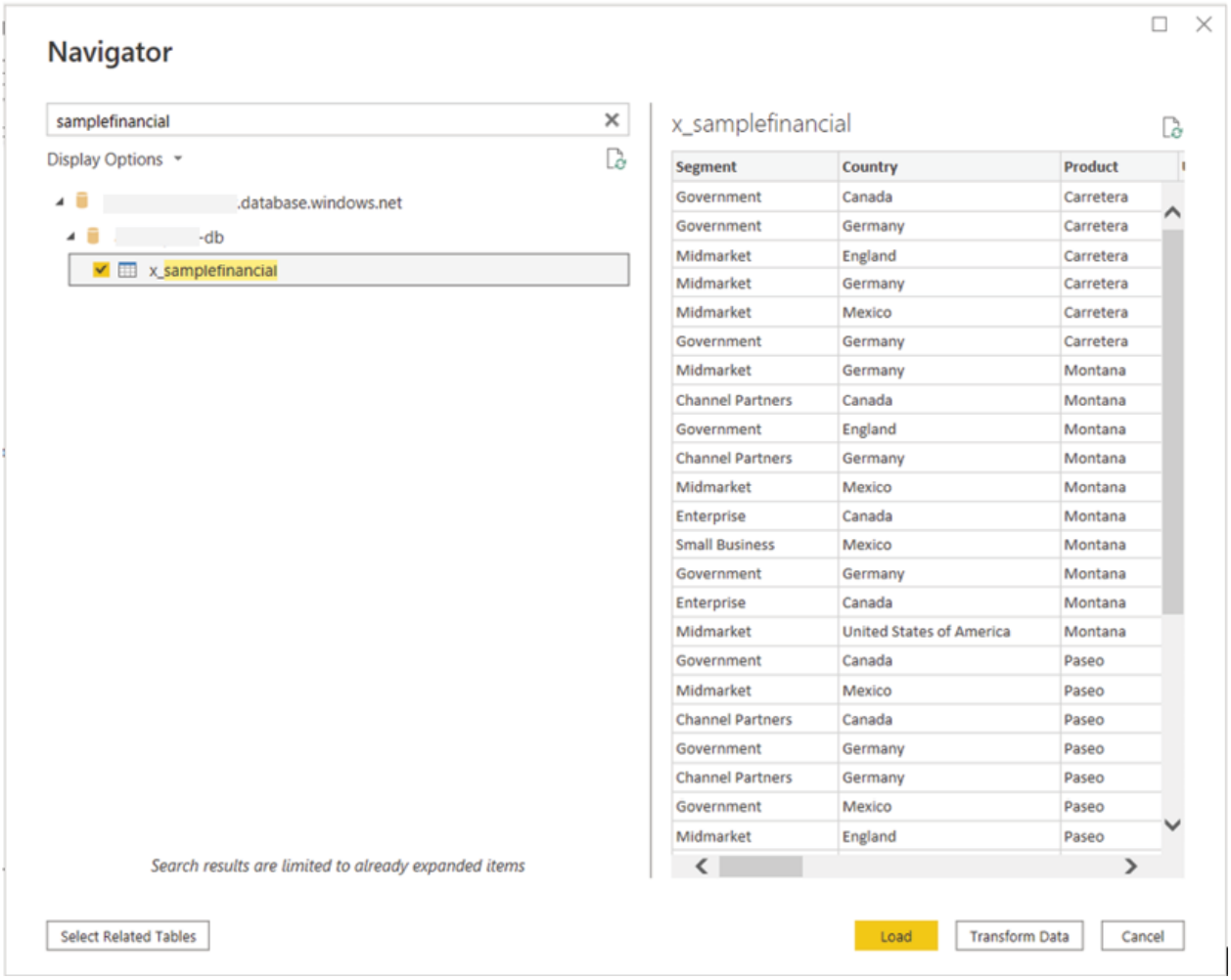



Figure 2.10: Database table preview

At this point, one or more tables can be selected to load the data or transform further in the Power Query editor.

The Azure SQL Database connector is one of the most comprehensive database connectors in terms of features, others may or may not support all the features. However, the working principles of most of the database connectors are almost similar.

Note: The client IP address of the machine Power BI Desktop is installed on may need to be added to the Azure SQL Database firewall settings.

Connecting an Amazon Redshift Database: Just like any other database, Amazon Redshift can be connected from Power BI launch the native Redshift connector, Get Data | Amazon Redshift. Figure 2.11 shows the Amazon Redshift connector:

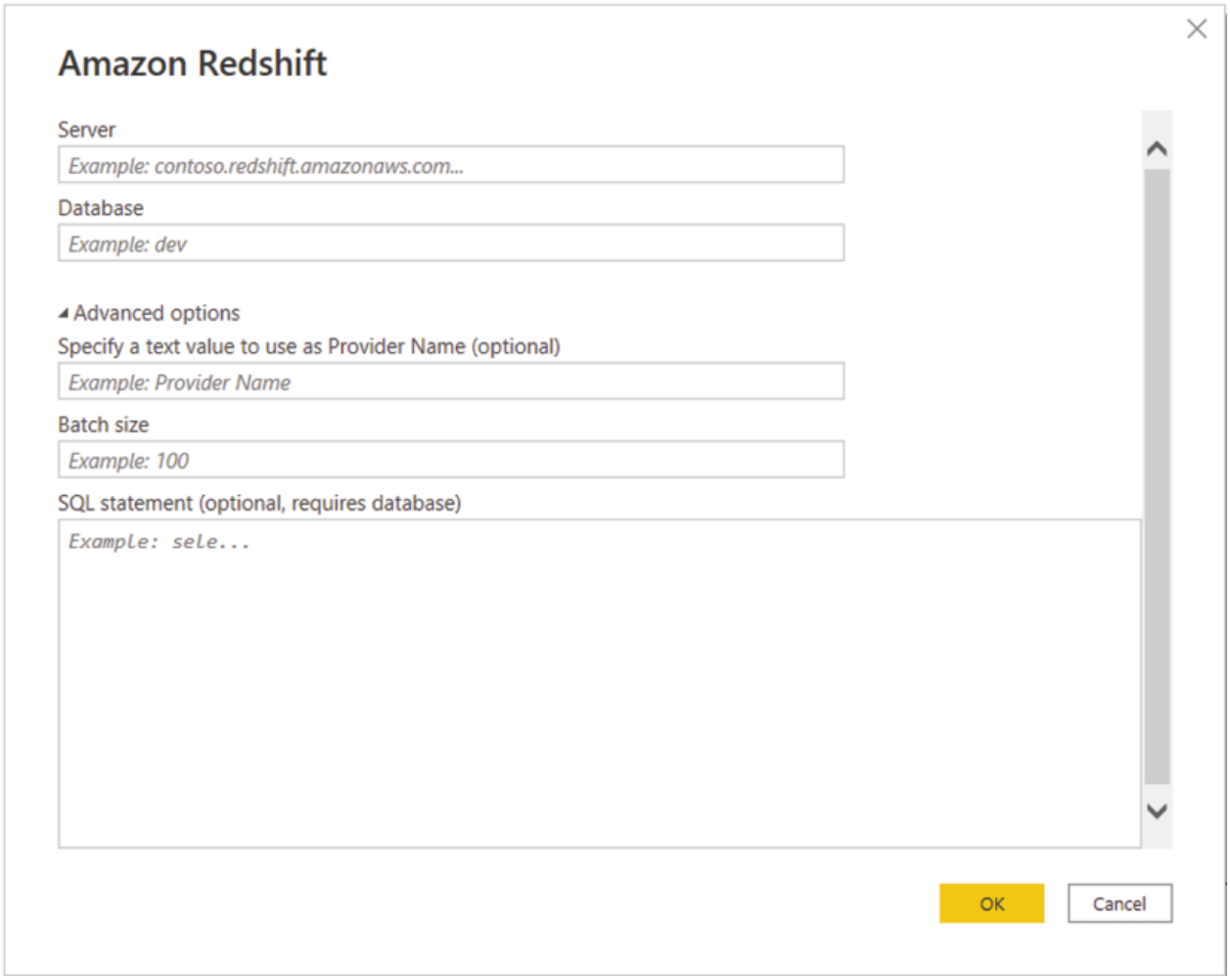



Figure 2.11: Amazon Redshift connector

The server name and database name need to be provided to be able to establish a connection. The advanced options are optional and can be used in case required. After successful authentication, the connection should be established and all the available database objects should be visible, from which we can select one or multiple tables as required and either directly load the data or start transformations in the Power Query editor.

Connecting to Power BI Dataflow: Power BI Desktop allows to connect to Power BI dataflows which is a workspace component in Power BI Service. Dataflows have capabilities similar to Power Query editor while being on the cloud enables it to create reusable transformation logics which can be shared across multiple datasets and reports inside Power BI. Get Data | Power BI dataflows should launch the Power BI dataflow connector. If connecting for the first time, the user needs to authenticate using the Power BI Service account and connect. The Power BI dataflow navigator is shown in Figure

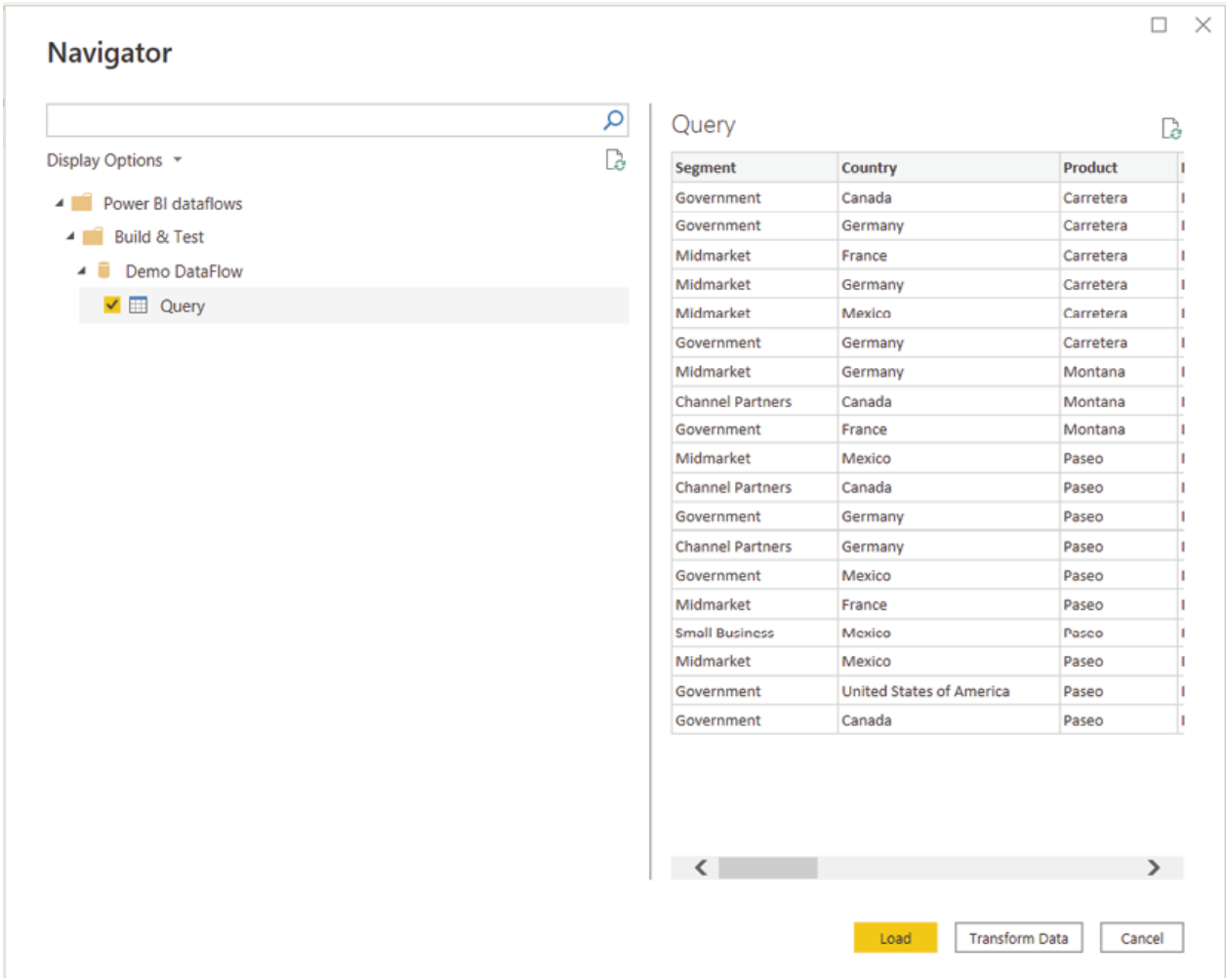



Figure 2.12: Power BI dataflow navigator

Once connected, all the workspaces user has access to and which have at least one dataflow in it should be visible on the left-hand side panel of the navigator. One can navigate from a workspace to the dataflows inside it, and then from dataflow to the respective entities (tables). In Figure ‘Build & Test’ is the workspace name, ‘Demo DataFlow’ is the dataflow inside it and ‘Query’ is the table name. One or more tables can be selected at once, and then the data can either be loaded or can be transformed in the Power Query editor.

Now, let us explore one of the most common requirements for project-related internal reporting:

Connecting to an Excel file: There are many ways to connect to a single Excel file from Power BI. However, in case the file is saved on the local machine of the user, we need to keep in mind that once the report is published on Power BI Service, it would require a data gateway to refresh. This is because the published report would sit on the Azure cloud, and the Microsoft datacenter would only be able to communicate to an on-premises data source (here the file on a local machine) via a data gateway.

Instead of going through the hassles of setting up a data gateway (which we will go through later in the book while we discuss Power BI Service in detail), for this scenario, one popular way is to save the file on a cloud service, like SharePoint online, which should not require a gateway for refreshing data. And the easiest way to connect to such files is to use the ‘Web’ connector of Power BI. Get Data | Web should launch the connector. Power BI Web connector, as shown in Figure

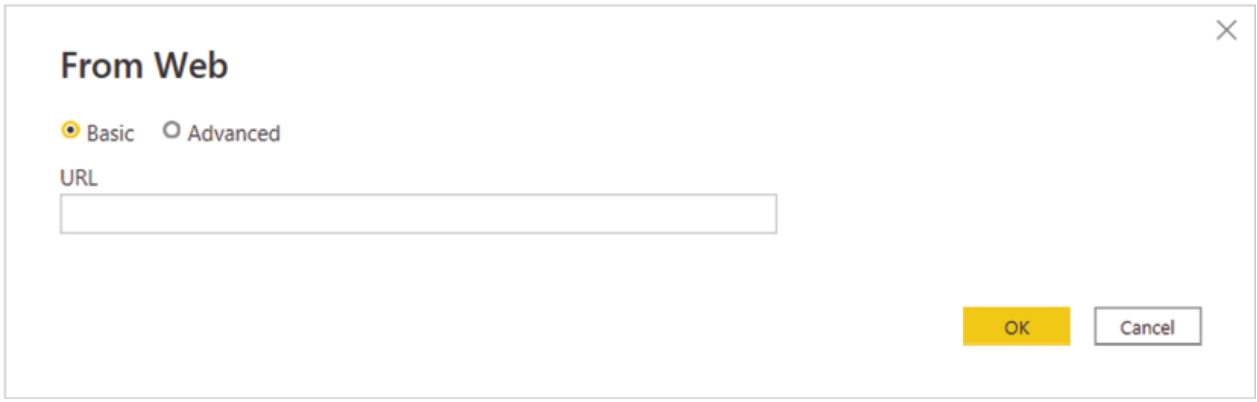



Figure 2.13: Power BI Web connector

We need to specify the direct file path here. For SharePoint, we need to go to the page having the file, select the file and the direct path can be copied from ‘More details’ as shown in Figure

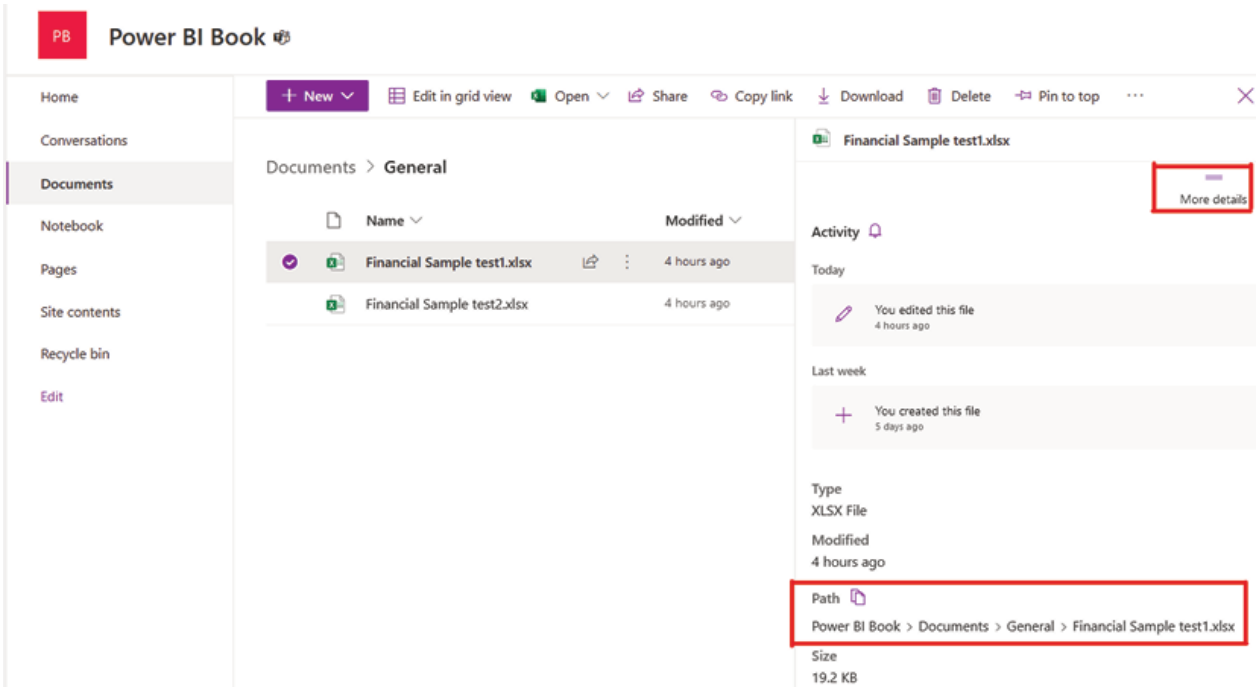



Figure 2.14: SharePoint direct file path

After providing the path on the Web connector, clicking on ‘OK’ should launch the authentication window for first time users. Successful

authentication should open the navigator where all worksheets and tables in the Excel file should be visible. Required tables or sheets can be selected and either loaded or transformed in the Power Query editor.

Apart from the default native connectors available in Get Data, Power BI allows you to connect to other data sources as well, using generic interfaces like ODBC, OLE DB etcetera.

The ODBC connector allows to connect to data sources using the available ODBC driver for the source. The third-party ODBC driver needs to be installed and configured first with parameters like Server address, Database name, user credentials etc., along with a Data Source Name Get Data | ODBC should launch the ODBC driver, and the DSN needs to be selected to be able to connect to the respective data source. Optionally, a SQL statement can also be specified to execute against the ODBC driver. Figure 2.15 displays the Power BI ODBC connector:

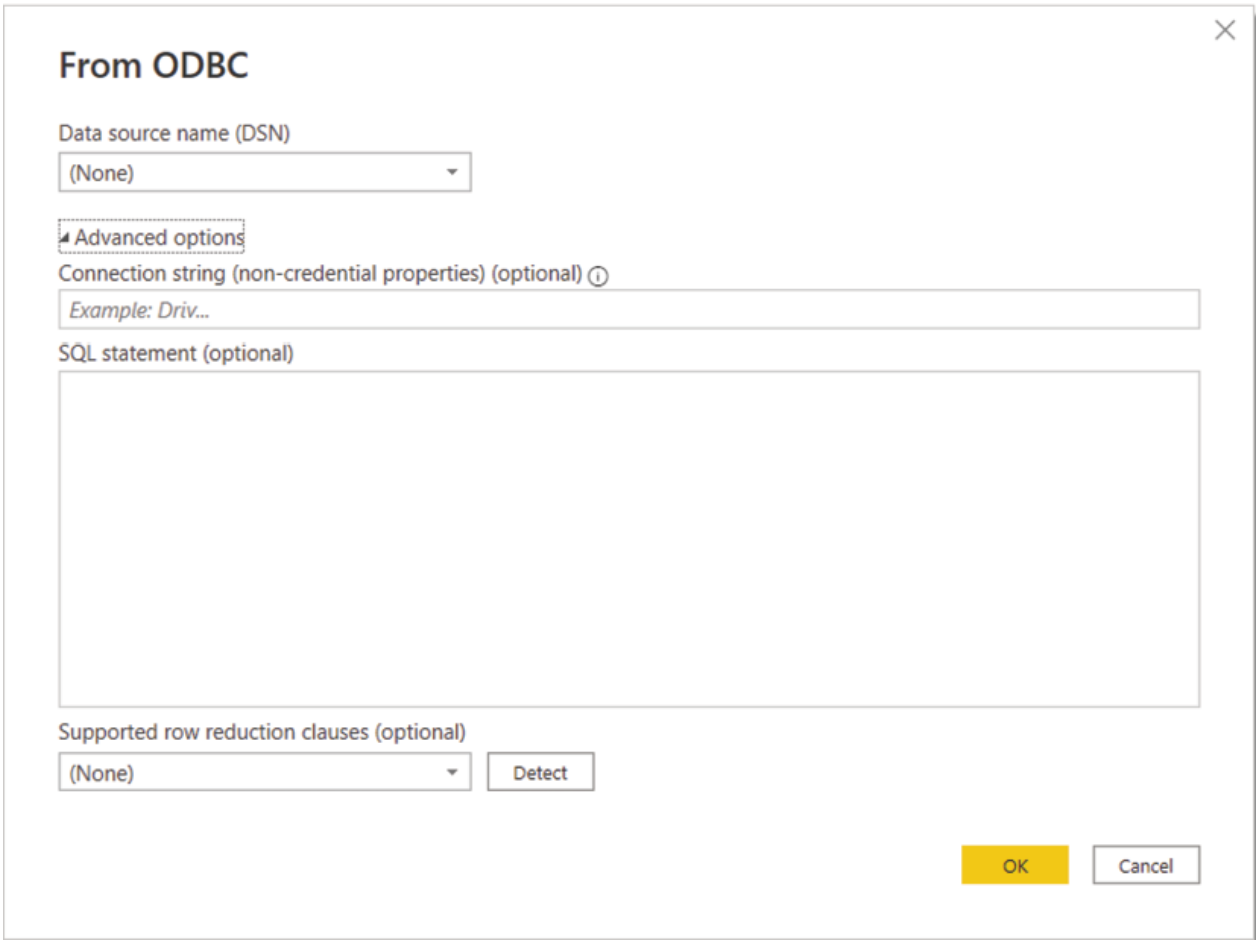



Figure Power BI ODBC connector

## Data connectivity modes in Power BI

Let us now understand the data connectivity modes, which we already have seen as required to specify for some data source connectors (refer to Figure Azure SQL Database

Connectivity mode plays an important role in terms of the report architecture. Here are the options that we have:

Import Mode: mode enables to load the data physically from data source to the Power BI file and store it in a compressed format, using an in-memory engine called VertiPaq (also known as xVelocity). Import should be the default choice unless required otherwise, for multiple reasons. The report responsiveness would be fastest in case of import as query would run against in-memory data. In terms of features, it provides maximum flexibility as the Power BI native languages like M & DAX can be fully utilized. One important limitation of import mode is that there is a 1 gigabyte limit imposed, after compression, per data model for Power BI Pro license.

DirectQuery Mode: In DirectQuery mode, only the metadata (or table structure like field names, data types etcetera.) gets loaded into Power BI. No physical transfer of actual data happens. DirectQuery is primarily intended for near real-time reporting as every time user interacts with the report, all queries directly hit the data source, get evaluated and retrieve data to Power BI. For DirectQuery, query folding or the ability to translate the query to the data source's native language is a must (which we will explore later in detail).

DirectQuery mode comes with several downsides, first of which is compromising with performance. As all the queries go back to the data source and get evaluated there, the response time drastically increases in comparison with import mode, and the performance largely depends on the optimization at the data source end. Also, not every feature is available with DirectQuery, nor every data source supports it. The data view in Power BI Desktop would not be available in DirectQuery.

Live Connection Mode: This to DirectQuery; however, only available for data sources like SQL Server Analysis Services (SSAS), Azure Analysis Services and so on. In the case of a live connection, the performance would be better than DirectQuery as this is based on the analytical engine of SSAS Tabular; the same engine which powers published Power BI datasets.

Composite Mode: Composite modes are a mix of DirectQuery and import mode. In a model, there can be multiple tables where few can be imported, and a few can have DirectQuery connectivity. The table storage mode can be configured as Import, DirectQuery or Dual. Before the introduction of composite models, when DirectQuery was used in a report, no other data connections were allowed for that report. Data views of a composite model are shown in Figure

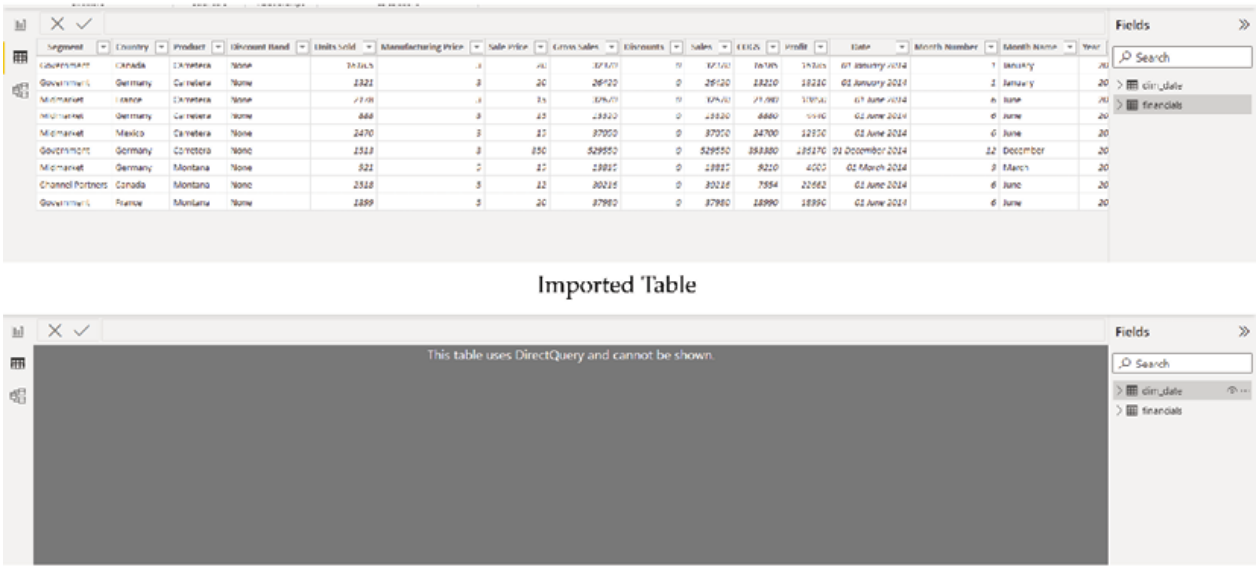



Figure 2.16: Data view visible for imported table, but not for DirectQuery, in a composite model

## Query folding

Apart from connectivity modes, another important aspect of integrating Power BI with different data sources is query folding, especially for scenarios where large data volumes are involved.

Query folding is the ability to translate any transformation step into the data source's native language. Any specific transformation step folds mean that will get executed at the data source's end.

Before going further into it, let us have a closer look at the Power Query editor itself, using a table imported from Azure SQL DB for illustration. Once the data gets loaded into the query editor, it can be previewed and transformed using different transformation options that are available.

The query editor can be divided into four categories as illustrated in Figure

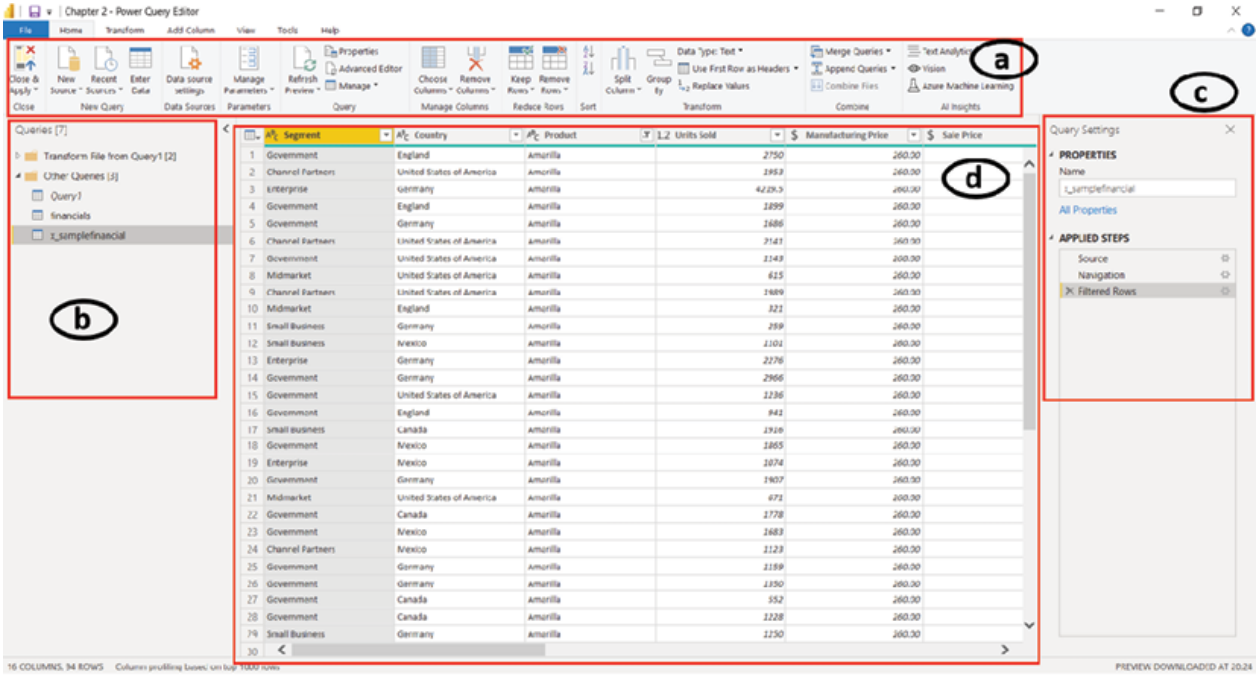



Figure 2.17: Power Query overview

Transformation ribbon: Different transformation options can be found here.
Queries Panel: every connection to the data source(s) would appear as a query here.
Query Settings: All transformations applied to a specific query would be listed here under Applied The query editor records every step of transformation whereas the last step provides the query output. These steps get automatically applied to the query sequentially during each subsequent data refresh.
Data Preview: A sample data can be previewed here.

Coming back to query folding, suppose we need to report only for a specific country which is and hence, we apply a filter on the Country field for England so that all other countries get filtered out. Now if the step folds, then the data would get filtered on the database itself and only a subset of data, which is relevant for would get imported to Power BI. Contrarily if that step does not fold, the entire dataset would get imported to Power BI, and on top

of that, the filter Country = England would get applied locally. This feature becomes invaluable, especially while working with large data volumes as it can drastically reduce the data that we are actually bringing into Power BI, resulting in improved performance and usability.

Now that we have seen why query folding is important, let us try to figure out how to confirm whether it is happening or not, which can sometimes be tricky!

After applying transformation steps to a specific query, we can right click on any step to check the View Native Query option. If the option is enabled, that means the query is folding up to that specific step. So, in case the option is enabled (not greyed out) on the last applied step of any query means the entire query is folding.

Let us perform a basic transformation on the data we have imported from Azure SQL DB and see how we can determine whether query folding is happening or not. Let us assume we do not really require the Segment column as we just want to report on a country level, hence we can remove that column from our table. Just selecting the column and clicking on the Remove Columns button on the transformation ribbon should do the trick as well as add a new step Removed Columns under Applied Steps list. Now, let us right click on Removed Columns to check query folding. The View Native Query option is shown in Figure 2.18 to help determine query folding:

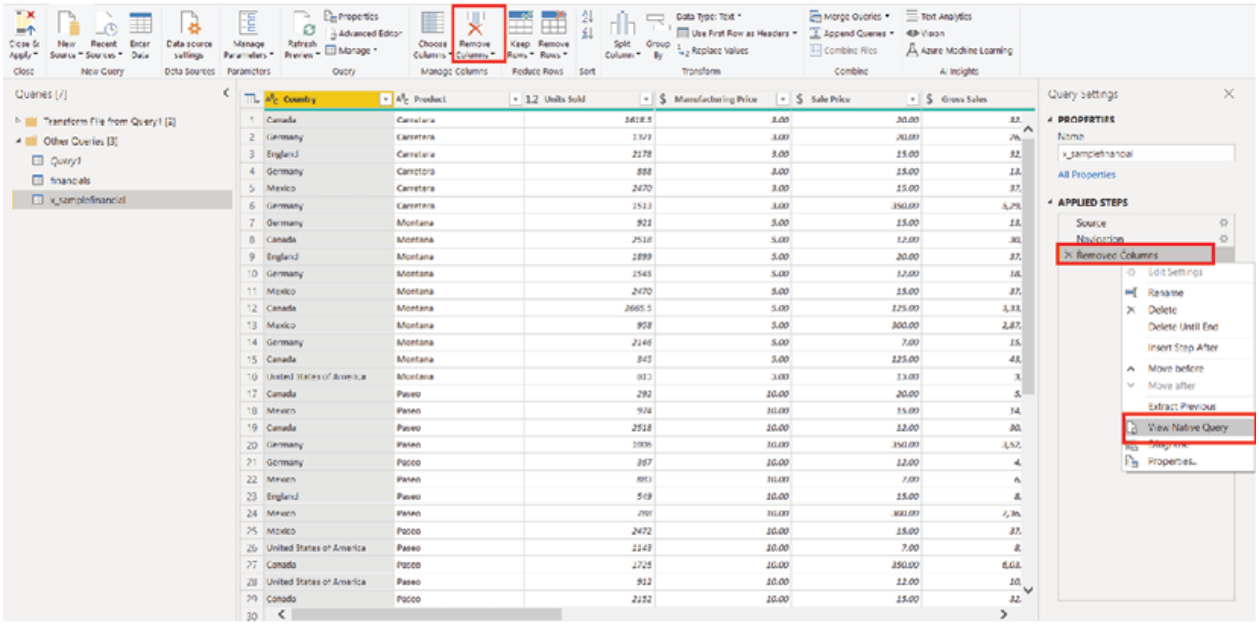



Figure 2.18: Determining query folding

As the option is enabled, it indicates the query is folding. To view the folded query, the View Native Query option can be clicked, and the actual query will be presented which is getting passed to the data source. Native Query is shown in Figure

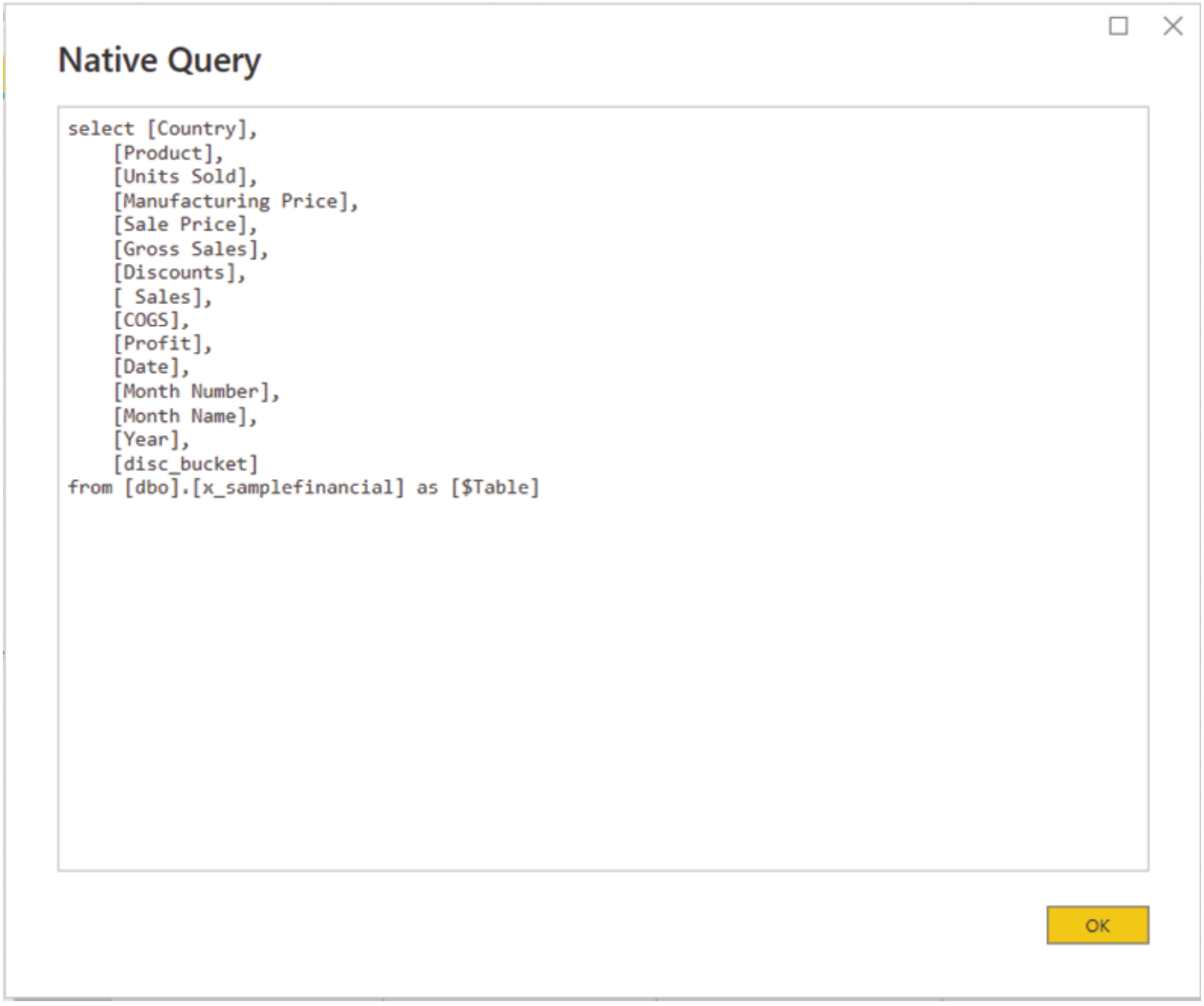



Figure 2.19: Native Query

As can be seen here, the Segment column that we have removed is not part of the SELECT statement that is getting passed to the data source, which is, Azure SQL DB.

However, if the View Native Query option is not enabled or greyed out, that does not necessarily mean that the query involved is not folding. Firstly, it could mean a subset of earlier steps are still folding in that query, secondly, the View Native Query option is not supported for all data sources.

The more accurate way to confirm query folding and conclusively track the query that is getting passed to any data source is to either use the Query Diagnostic tool, which is now integrated with Power Query editor or use external tools like SQL Profiler to analyse traces in real time. DataFlows or Power Query Online also have step folding indicators which can help to get an early indication regarding query folding.

Tips: In case all transformation steps in a query cannot be folded, it is important to identify the step that is preventing or breaking the query folding. The best practice is to keep the steps which are indeed folding on the top together in a logical order for ensuring to delegate as much processing as possible to the data source itself. This should enhance the overall performance of the model and will limit the resource utilization on the Power BI end.

## Introduction to the Mashup language

Now, let us talk about data transformation. Often, we experience that the data that we get from different sources need to be shaped or transformed in order to use it effectively for the specific requirement we are working on. This is because data is stored in data warehouses or databases not to meet a specific requirement, but rather to cater to different purposes from a common repository or single point of truth.

The transformation options on the Power Query editor can be found either on the transformation ribbon or by right clicking on the column headers on the data preview. Transformation options are categorised by tabs present just above the transformation ribbon. For example, under the Transform tab, the options to transform existing columns can be found while under the Add Column tab, different options to create new columns can be explored. Transformation categories highlighted in Figure

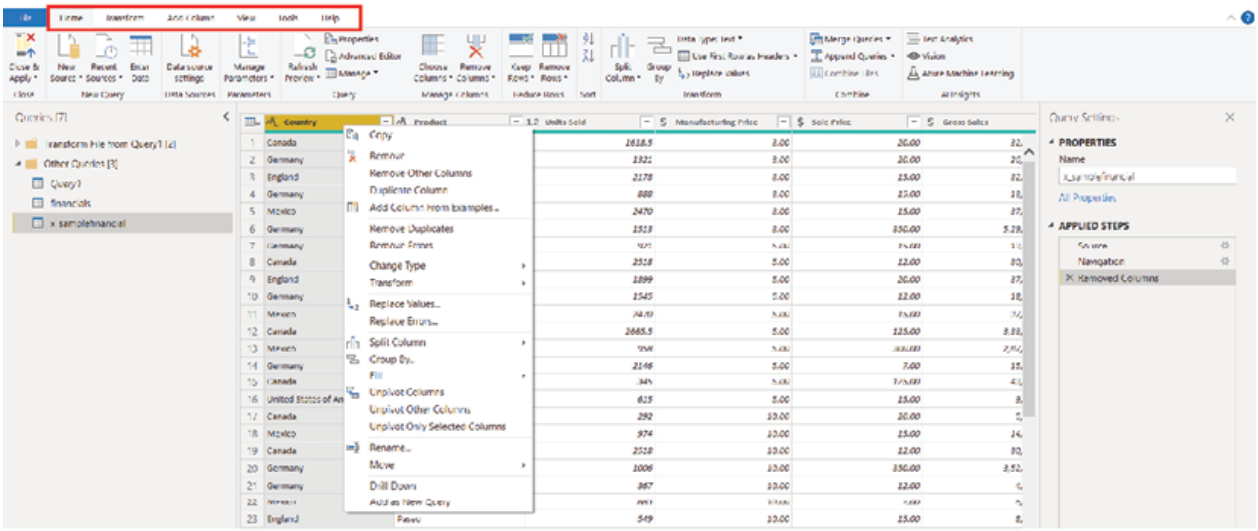



Figure 2.20: Transformation options in query editor

In case we create a transformation step, which later needs to be removed or reversed, that can be done by simply removing the step itself. For instance, assume that we actually need that the Segment column which we earlier removed considering as redundant. If we just remove that transformation step, we will get our column back. The transformation step is shown in Figure

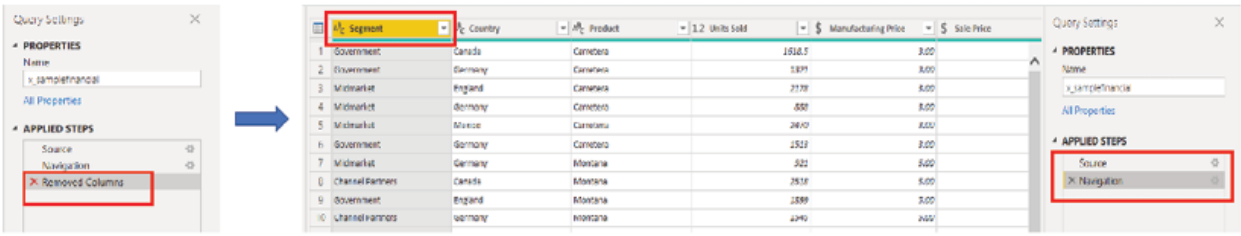



Figure 2.21: Removing a transformation step

Tip: Care should be taken while removing a step, as in case any subsequent step has a dependency on the step we are trying to remove, the query might break.

The query editor has a native language called M, which stands for the Mashup language, which is extremely useful while performing transformations. One necessarily need not master M for using query editor effectively; however, knowing how to use it can make the life of a developer a lot easier in comparison with only using wizards.

From the menu bar, under the View tab, the Formula Bar checkbox can be used to enable the formula bar, if not already enabled. The M code corresponding to any step of a query can be viewed on the formula bar by selecting the step. The M code for an entire query can be seen using the Advanced Editor under the Home tab.

Let us now understand the importance of M with a very simple example. Transformation options like renaming columns, removing columns etcetera are self-explanatory; however, we need to keep in mind that whenever we are performing any transformation, that step would get recorded under Applied Steps and get applied every time the data is refreshed. Hence, it becomes important to avoid repetitive steps whenever possible for improved performance and efficiency. Continuing with our imported data, let us assume the requirement is to filter the data only for segment Government and change the column name Country to Country of For filtering the segment column, just like in Excel, the value Government can be chosen from the column drop-down, or a text filter available under the drop-down menu can be applied.

The column name can be changed either by right clicking on the column header and using the Rename option, or directly by double clicking on the column header. The Power Query formula bar and the option to launch the advanced editor are highlighted in Figure

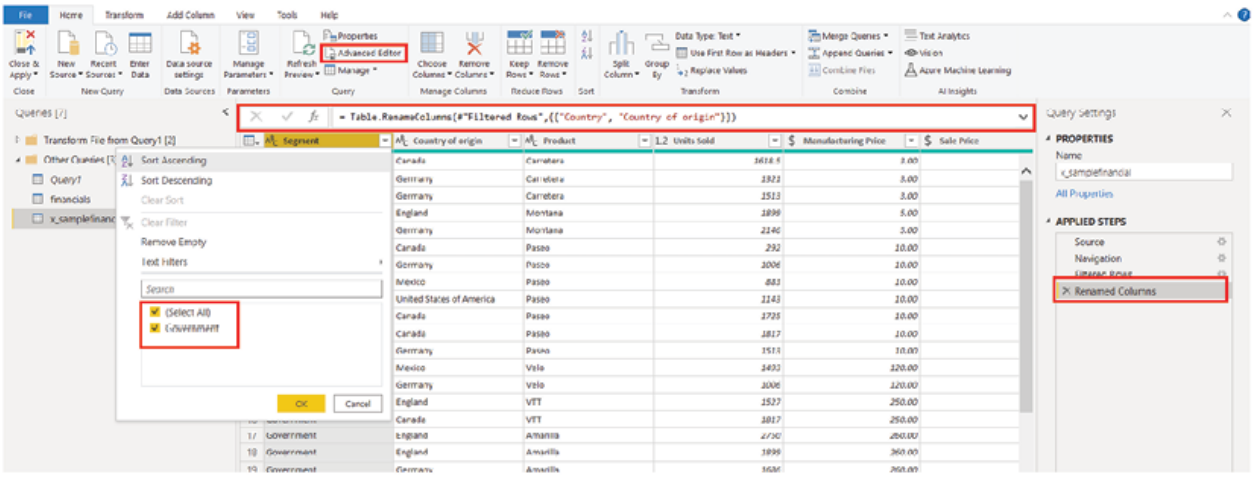



Figure 2.22: Advanced Editor, formula bar & available segments

Change is the only constant - proving the saying correct; let us say we now have a requirement to include data for segment Small Business as well! As we have already filtered out all other segments apart from Government in an earlier step, other segments would not be available on the field dropdown anymore. Here, the most convenient way to incorporate the change would be to tweak the M code, either on the formula bar by selecting the step where we actually filtered the data, or on the advanced editor at the query level. A transformation step with updated M code is depicted in Figure

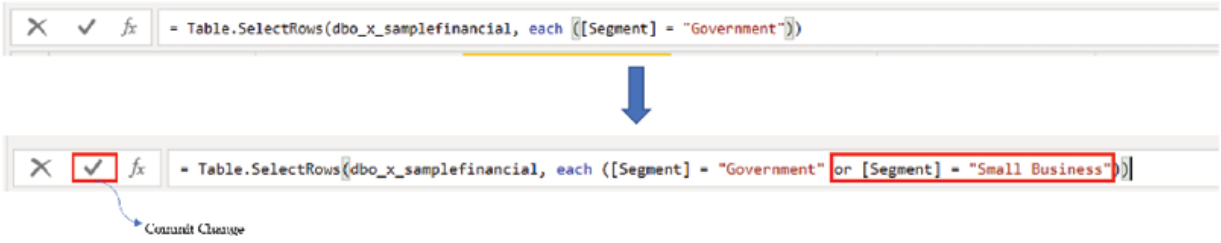



Figure 2.23: Updating M code and committing changes

After updating the M code with the required changes (here, including the segment Small Business in our data), we need to commit the change and we should now have the data filtered for both the segments as required.

Tip: M is a functional case sensitive language; hence, care should be taken while using uppercase or lowercase letters.

Now, if we take a look at the advanced editor, we should be able to view and interact with the M code for the entire query, as created against all the transformation steps under Applied Figure 2.24 shows all the transformation steps for a query in the advanced editor:

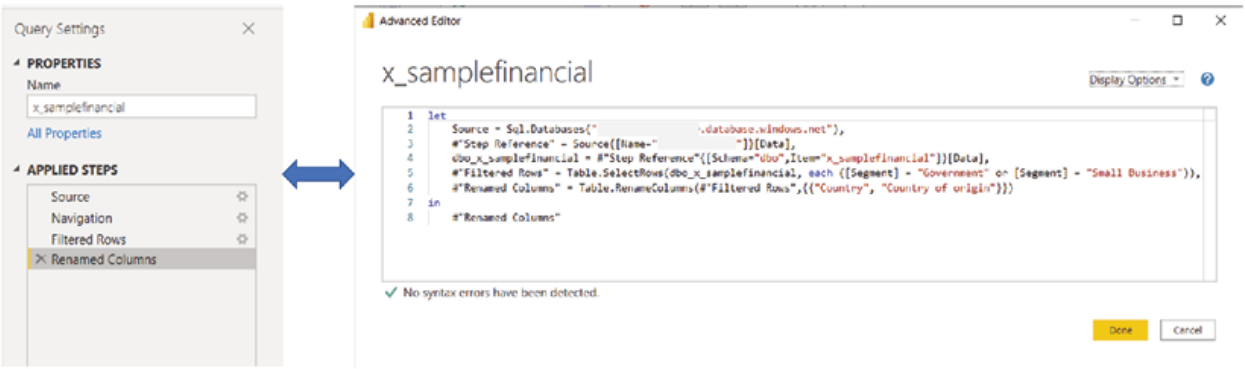



Figure 2.24: M code for the query

By now, we should have the basic understanding of how data transformation works in Power Query editor, as well as how M can help in performing those transformations. Time to explore some interesting transformations now.

## Common data transformations in the query editor

We already have seen how to remove columns, rename columns and filter data in the previous sections. Let us now have a look at a few of the other most common transformations in the query editor:

Changing Data Data types are defined at the field level and determine how the field values would get stored in the dataset. In the query editor, usually, there are multiple ways to perform a transformation. For instance, to change the data type of any field, one can right click on the field name and open the Change Type menu to see all available options. The same menu can be populated by clicking on the Data Type option, on the Home tab, under the Transform group. Clicking on the left icon on any field name populates the same menu as well. Different options for changing data types can be seen in Figure

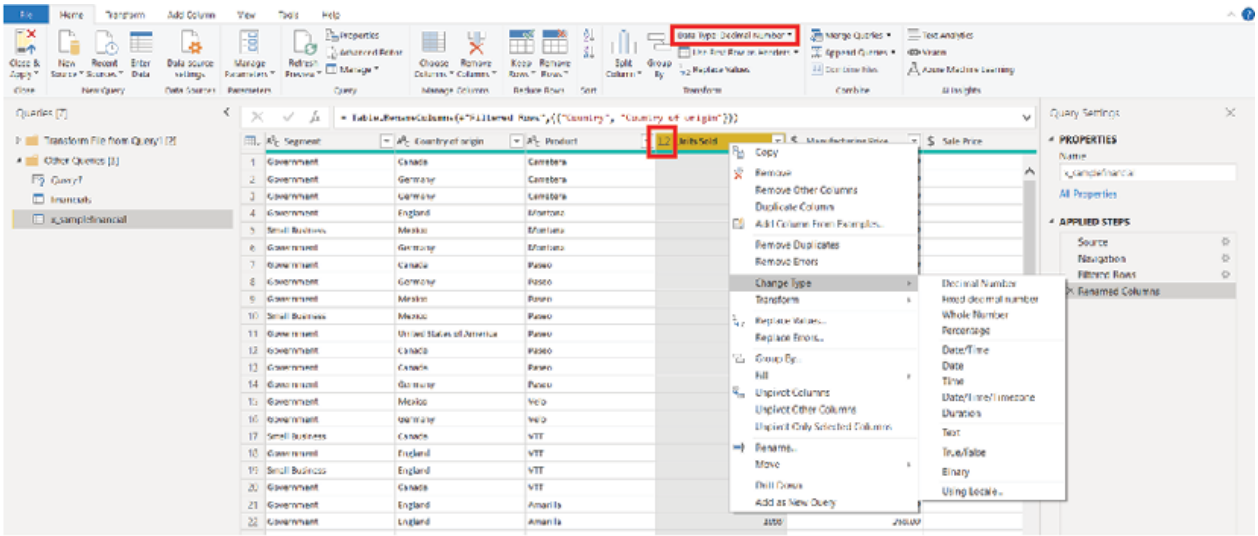


Figure 2.25: Changing data type options

Someone inquisitive enough can even find the same Data Type option on the Transform tab under Any Column group as well! All these options work in the same way and the user can use any of them as per convenience.

Let us say we want to see Units Sold as a whole number and want to disregard the decimal part. By clicking on the Whole Number option on the menu should store the field values as whole numbers. Figure 2.26 highlights the updated data type for the field:

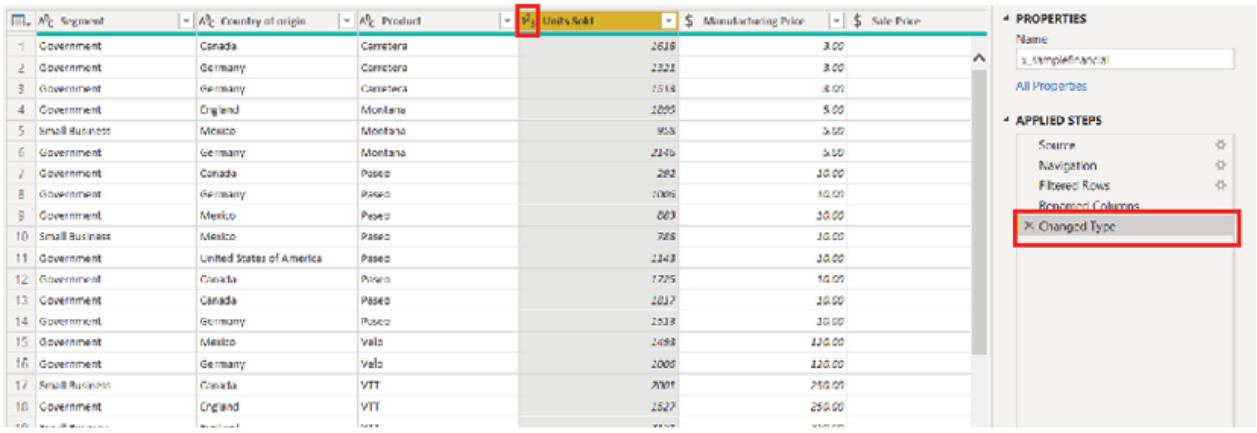



Figure 2.26: Data type changed from decimal to whole number

Note that the left-hand icon of the field has also changed to indicate the new data type, also a new transformation step has been added under Applied recording the change as expected.

Case Let us now see how to perform case transformations which is often a common requirement. To convert text values to either lowercase letters or uppercase letters, or to capitalize every first letter of a word, no complex string operations are required in the Power Query editor. The relevant options can be seen after right clicking on any column having text values and exploring the Transform option, as shown in Figure

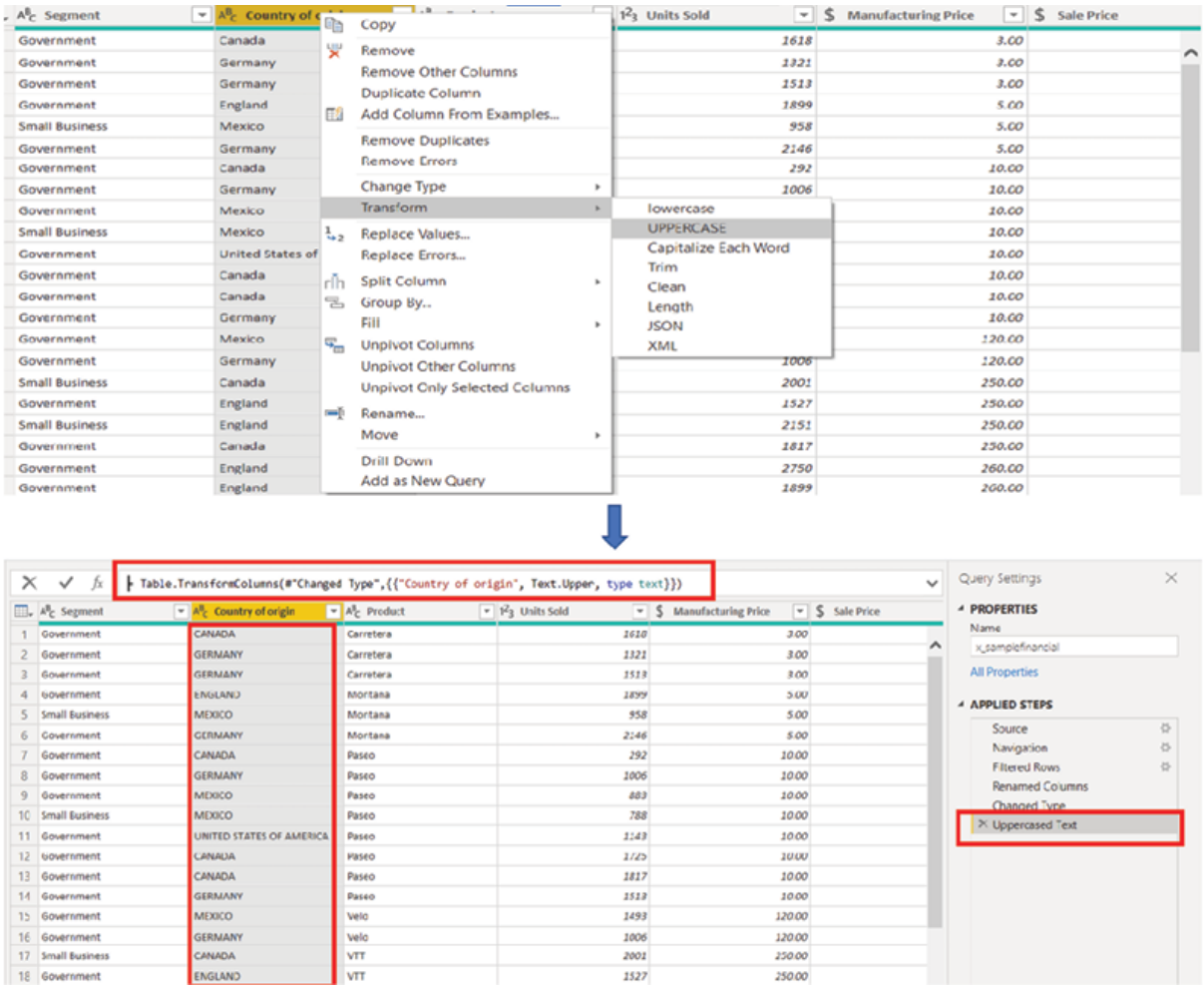



Figure 2.27: Transforming case of text values

Performing the transformation, as applied above for Country of should convert all the country names to uppercases. Also, this would create a new step under Applied Steps and the corresponding M code can be seen on the formula bar.

Unpivot This is an interesting transformation option in the query editor which otherwise, may not be as easy to implement. Sometimes, it makes sense to flatten the data in a matrix format to make it more suitable for BI reporting. Often data is maintained by pairing attributes with values which needs to be

unpacked to meet reporting requirements. The concept can be understood by referring to Figure

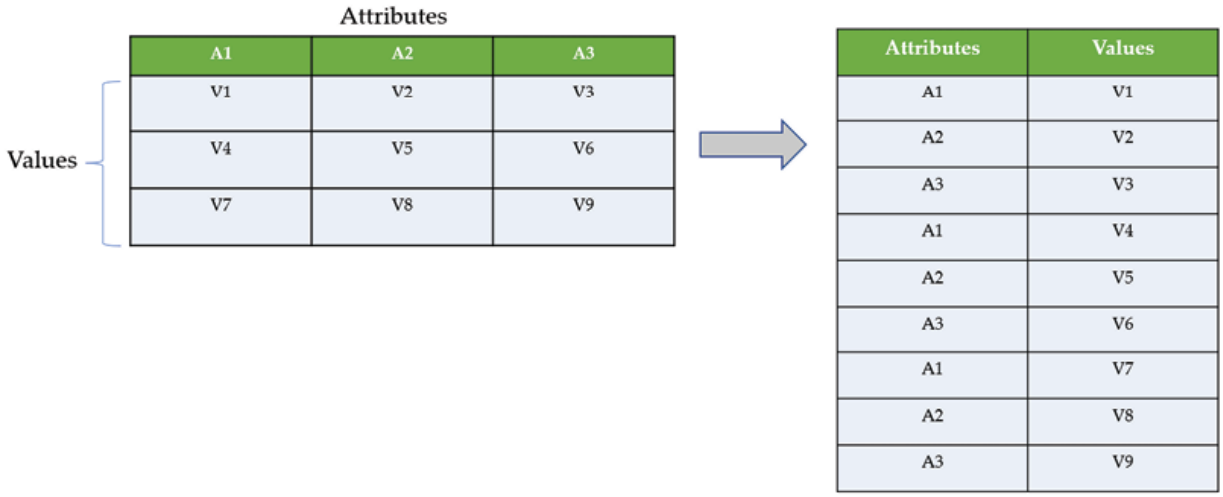



Figure 2.28: Unpairing attribute-value pairs

Let us now see a real-life example. Consider we have the below data in a table as shown in Figure

| Region/Product | Aspen | Bellen | Carlota | Yanaki |
|---|---|---|---|---|
| East | 10 | 77 | 39 | 42 |
| MidWest | 74 | 47 | 57 | 26 |
| South | 14 | 61 | 89 | 20 |
| West | 67 | 51 | 56 | 77 |

Figure 2.29: Partially structured data

The matrix captures the no. of products sold for different regions. This type of representation is not ideal for reporting where we slice and dice the data, hence, this needs to be flattened.

The first required step would be to select all the columns that we need to unpivot (here, all product columns), right-click and use the Unpivot Columns option. This should create two columns, one for the attributes like product

names and another for the values. The original product columns would be removed.

Figure 2.30 illustrates the unpivot operation, applied on the matrix as shown in Figure

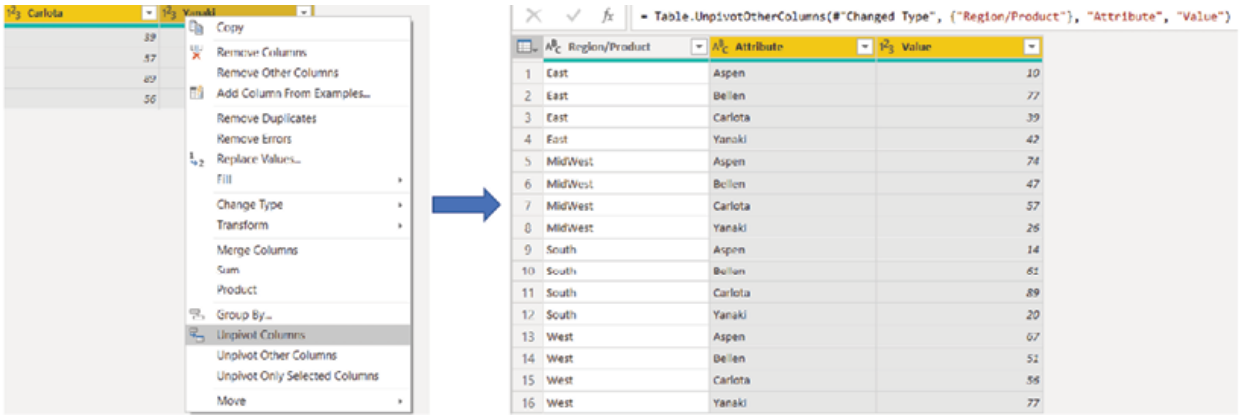



Figure Unpivoting Columns

The table now is ready for reporting and can be loaded to the Power BI Desktop model using the Close & Apply option.

Add Conditional One common requirement for data transformation is to be able to create additional columns based on the existing data that we have. There are multiple ways to do it in Power Query Editor, conditional columns are one of them. Conditional columns in the query editor provide a no-code way of implementing simple IF-ELSE conditions, in a matter of few clicks! Let us assume that we are working on the same table imported from Azure SQL DB (refer to Figure Just to quickly refer to the columns that this table has, we can click the Choose Column option under the Home tab, which should provide us with a view of all available fields in the table, to choose from. The field list for our imported table is shown in Figure

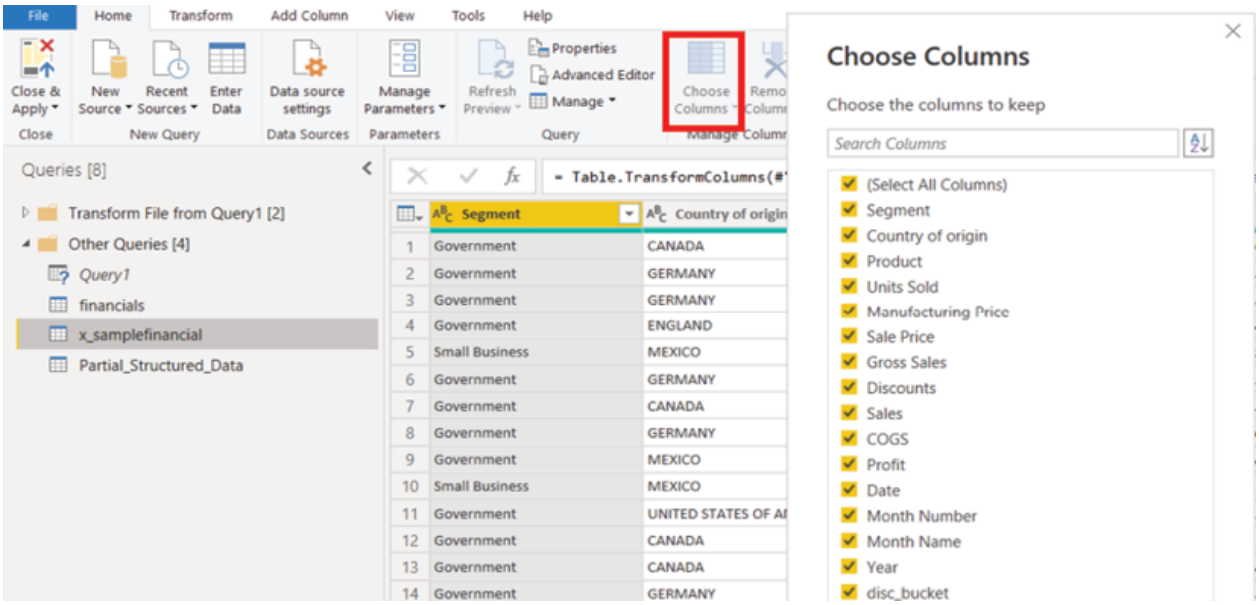



Figure 2.31: Columns available in the table

Assuming that we need to introduce a new field Market where for Country of origin - UNITED STATES OF AMERICA it should be and for the rest it should be The logic can be implemented by creating a conditional column, using the Add Conditional Column dialog box which can be launched from the Add Column tab, as shown in Figure

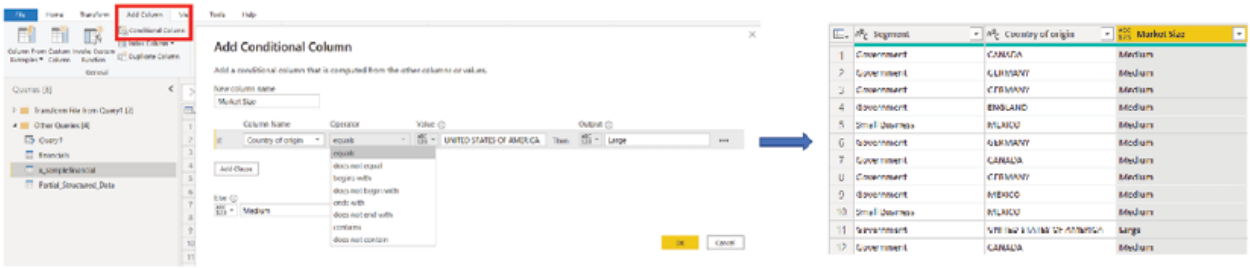



Figure 2.32: Adding a conditional column

The options on the dialog box are self-explanatory: a name for the new column needs to be provided, then we need to select the column on which the

logic would get applied to (here Country of After that, we need to select a comparison operator as shown in the preceding figure; finally, we need to provide the value to compare and the desired output. After the new column gets created, it can be dragged to re-order, like any other columns on the data preview in the query editor. The condition can be modified using the gear icon, next to the Added Conditional Column step, in the Applied Steps of the Query Settings pane, as shown in Figure

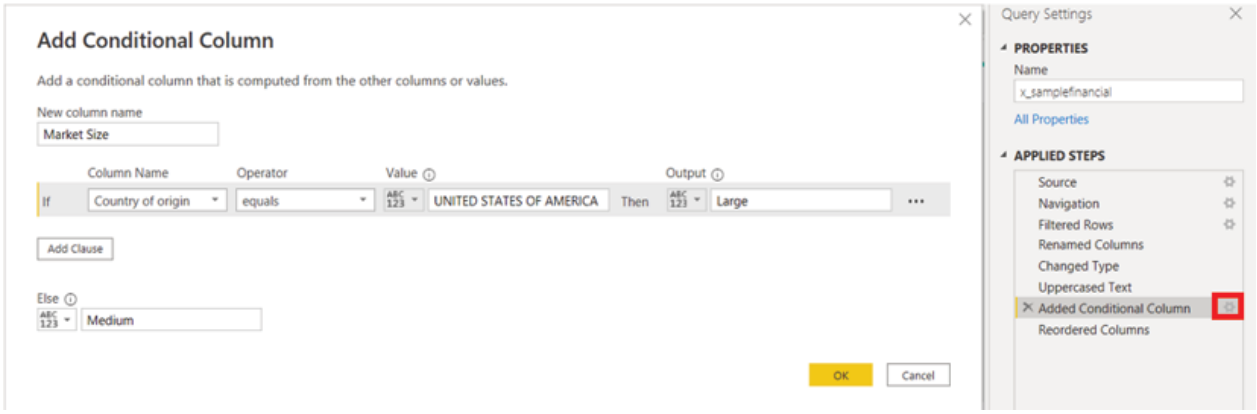



Figure 2.33: Modifying a conditional column

Conditional columns are great for simple logic implementation, however there is another option for introducing new columns with more control, which is, Custom which we will explore next.

Add Custom Custom columns provide more control while adding new columns as the logic can be implemented using M code. Continuing with the same table, let us see how to add a new column which is based on two columns, Segment and Market If Segment is Government and Market Size is then the Rating should be 1, otherwise 2.

The dialog box to create a custom column can be launched from Add Column | Custom and the required logic can be implemented using M, as shown in Figure

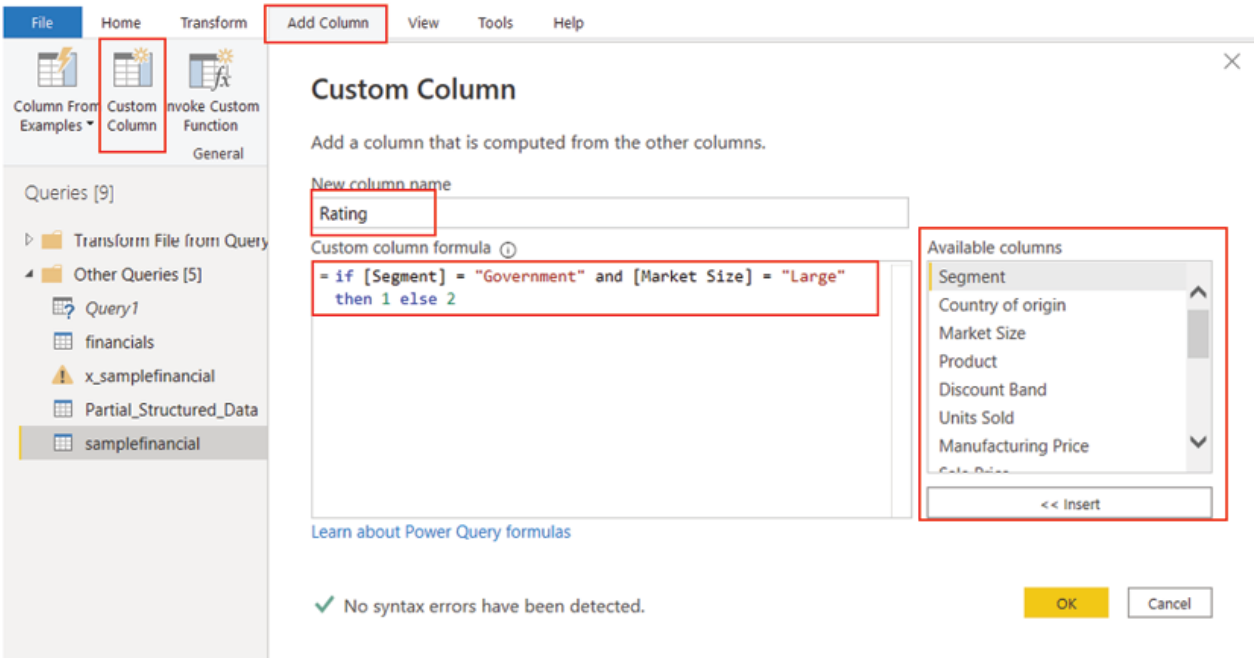



Figure 2.34: Creating a Custom Column

All the available fields for the table can be accessed on the right-hand side panel to include in the formula, and it requires a simple self-explanatory IF-ELSE statement to implement the logic, as shown in the figure above. Once created, the column should be available on the data preview and can be modified using the gear icon, next to the Added Custom step, in the Applied Steps of the Query Settings pane.

There are a whole lot of other transformations available in the query editor, which readers are encouraged to try out themselves and explore!

Tip: Microsoft provides a sample dataset to practice the concepts, which can be loaded to a blank Power BI file from the home screen by matter of few clicks, as illustrated in Figure

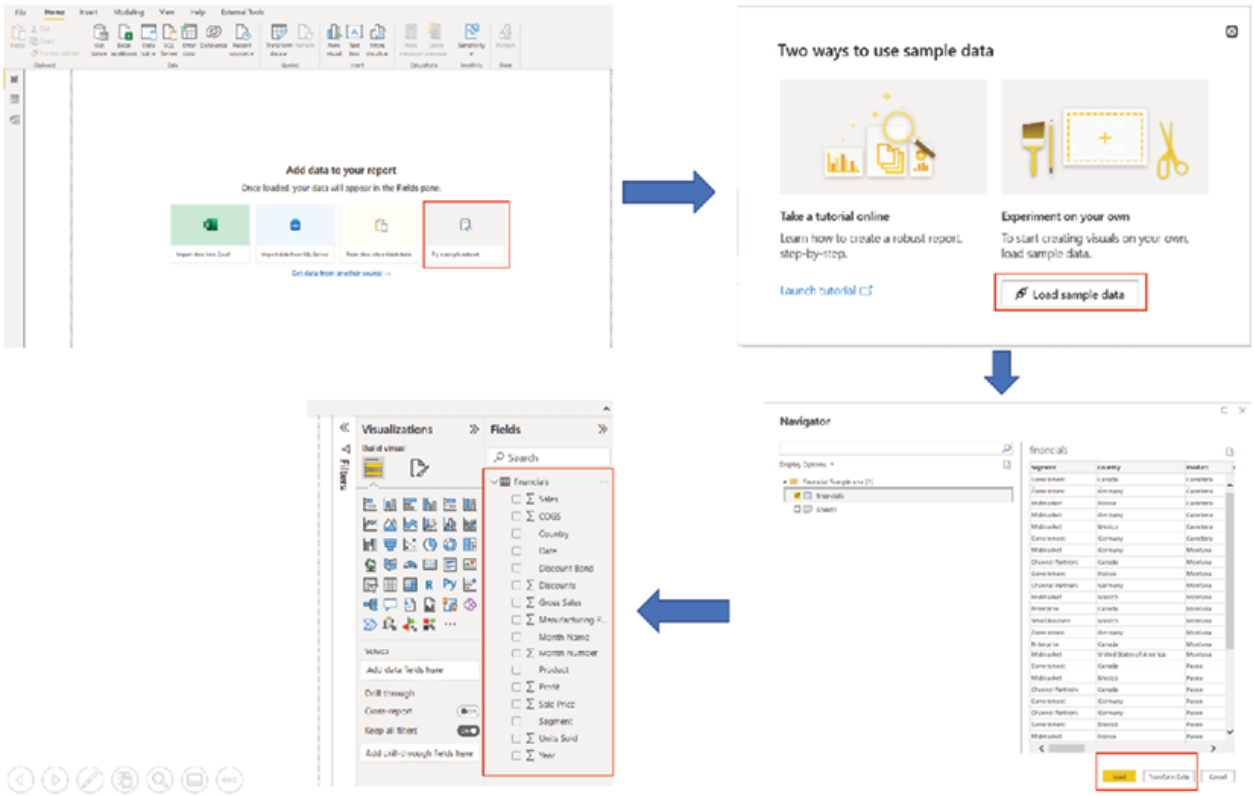



Figure 2.35: Getting ready-to-use data to practice

It is the same dataset that we have been using for many explanations and will continue to use throughout the book.

## Data cleansing in the query editor

There are some transformations specifically used for data cleansing; we will explore a few of those now. We may need to cleanse data on the reporting layer unless the data cleansing aspect is managed on the data source. Few commonly used data cleansing transformations are:

Managing Missing Values: Let us assume the table that we have imported has some missing values for the Sale Price Column which we want to replace with suitable alternatives, in this case, with the average sale price of the records. There are multiple ways of doing it, we can directly write a piece of code in M in a custom column; however, let us see how we can derive a scalar value first from a column using some available transformation options, and use it for this purpose. The table with the missing values is shown in Figure

| Segment | Country | Product | Units Sold | Sale Price | Manufacturing Price |
|---|---|---|---|---|---|
| Government | Canada | Carretera | 1618.5 | 20 | |
| Government | Germany | Carretera | 1321 | 20 | |
| Midmarket | France | Carretera | 2178 | 15 | |
| Midmarket | Germany | Carretera | 888 | null | |
| Midmarket | Mexico | Carretera | 2470 | 15 | |
| Government | Germany | Carretera | 1513 | 350 | |
| Midmarket | Germany | Montana | 921 | null | |
| Channel Partners | Canada | Montana | 2518 | 12 | |
| Government | France | Montana | 1899 | 20 | |

Figure Table with missing values

To calculate the average value, right-click on the Sale Price column and then select Add as New This should create a new query, only for this column, in the form of a list. The query icon should change as well, representing a list

now. At this point, few options would be available to transform the list using the List for our purpose, Statistics | Average should calculate the average value of the list. This should also transform the list into a scalar value (the icon would change again to represent a scalar value this time), which can be used to replace the missing values in the original query. The transformations are illustrated in Figure

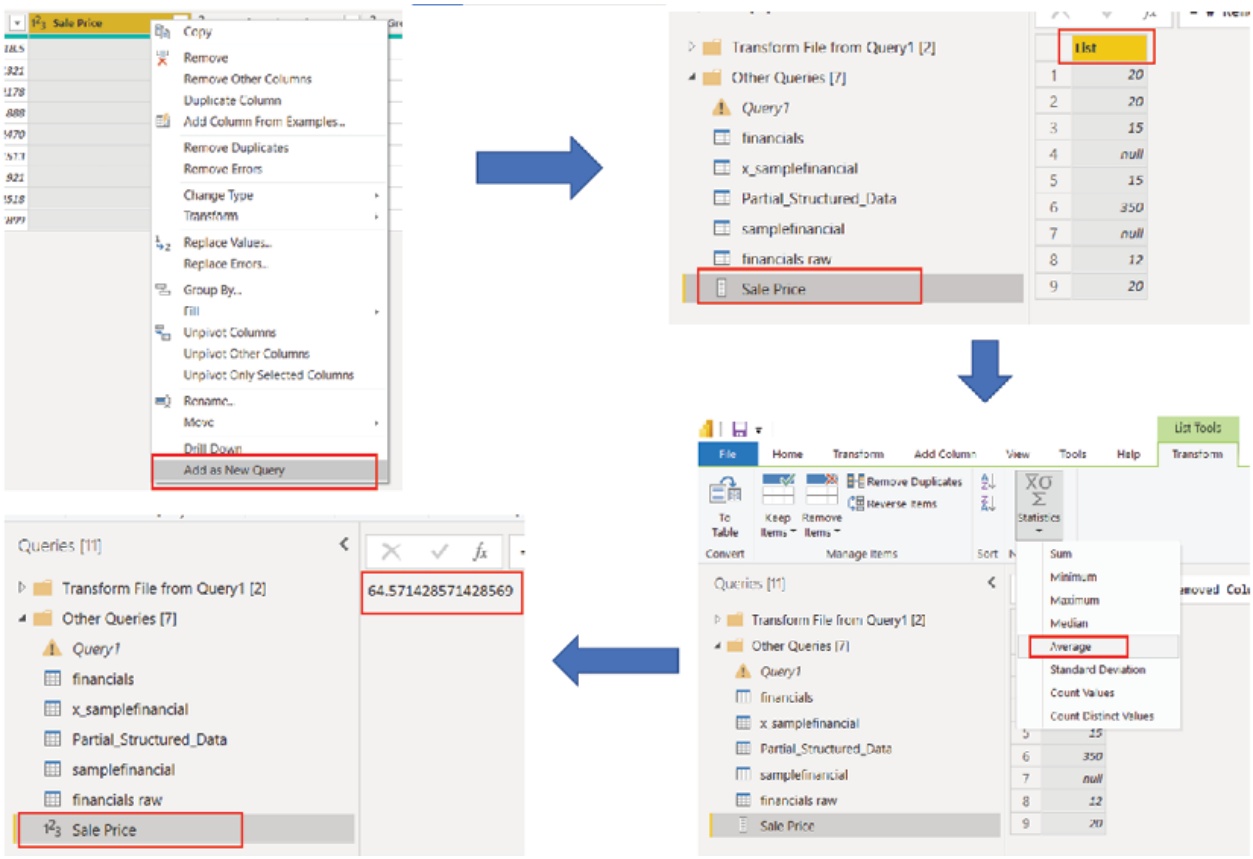



Figure 2.37: Calculating Average value for a column without M functions

Now, this pre-calculated average value can be used in a custom column, in the original query, to replace the null values as shown in Figure

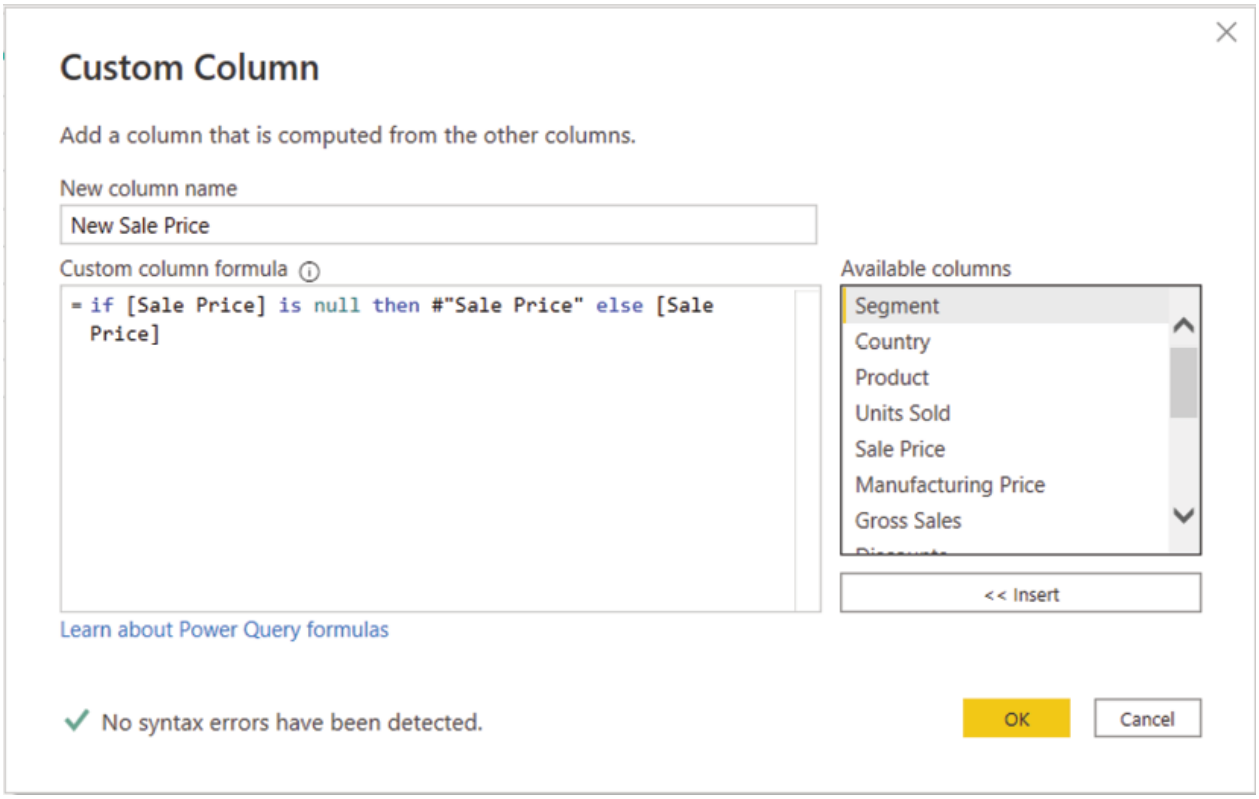



Figure 2.38: Custom formula to replace null using pre-calculated average value

Tip: In M, columns can be referred by enclosing the column names in square brackets, and queries, parameters etcetera can be referred by their names enclosing in double quotation marks and adding a hash (#) prefix; as shown in the preceding figure.

Managing Data Quality: The query editor has multiple features to maintain the quality of data. There is a Remove Duplicate option that can be applied on a column level to maintain unique values for a field, for example, in case the key values of a dimension table has duplicates due to some technical debt. There are ways to handle errors as well, errors can be filtered out, or replaced, depending on the business objective.

Removing extra spaces: In case a text field has leading or trailing whitespaces, the Trim transformation can be used to remove apply the transformation, right click on any text column header and select Transform | Alternatively, on the Transform tab, Format | Trim can be used to apply the same transformation on any text field.

Removing non-printable characters: In case a text field has non-printable characters, such as line feeds, the same can be removed using the in the query editor. The transformation can be applied either using Transform | Clean or on the Transform tab using Format |

Tip: A few of these transformations might prevent query folding. All the steps that support query folding should be placed before the one that might break it. Also, a few of these can be performance heavy, so it is recommended, especially while working with large volumes of data to push the transformations as much as possible to the data source end.

Appending and merging data

Now, we will see how to effectively combine data from same or different data sources. There are two basic ways to combine: Appending and Merging:

Append The append operation in the query editor creates a single table from the content of two or more tables, by adding data vertically, resulting in an increased number of rows. However, if the tables do not have the exact same column names, all column headers from all the tables would be present in the appended table and missing values of any of the tables would get displayed as which can increase the number of columns as well. Let us understand the concept by referring to Figure

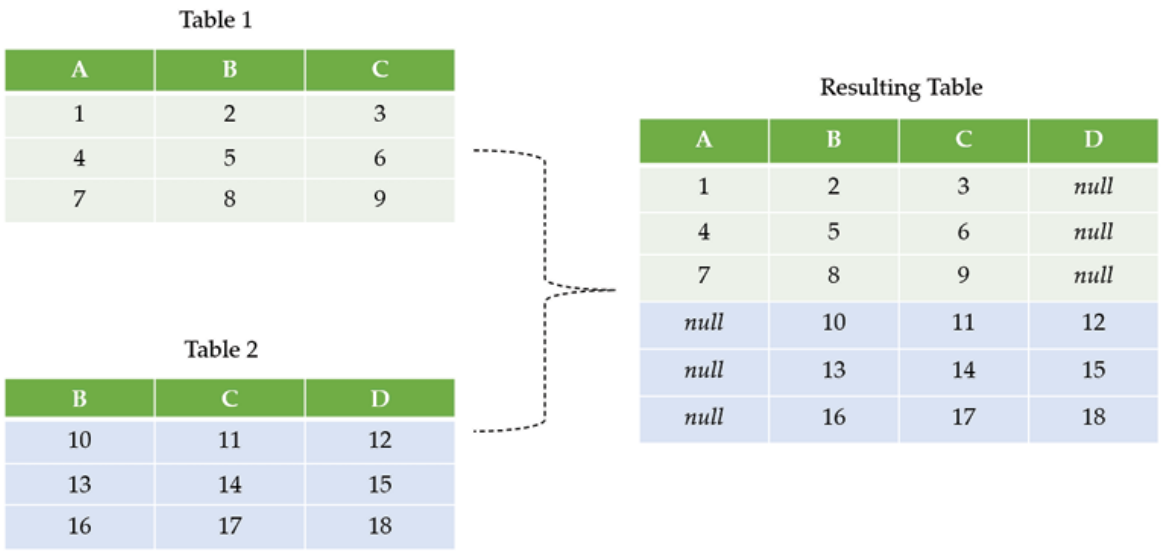



Figure 2.39: Append operation working principle

In Figure Table 1 does not have column D and Table 2 does not have Column A. The resulting table has all 4 columns, with null values populated for the

missing data in respective tables.

In the query editor, the option to append queries can be found under the Home tab and Combine group, as highlighted in Figure

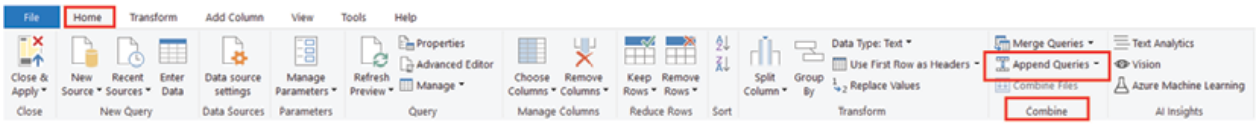



Figure 2.40: Append Queries on query editor

The append operation can be applied on two or more tables, either to add data to an existing table or to create a new resulting table.

Merge Merging queries is another way to combine data, which adds data horizontally based on columns that drive the join, potentially increasing the number of columns in the resulting table. Common columns (key field or identifier) need to be selected for performing a merge operation; however, it is not required for the column headers to match as in this case, as we will explicitly select the column.

In the query editor, the option to merge queries can be found under the Home tab and Combine group, as highlighted in Figure

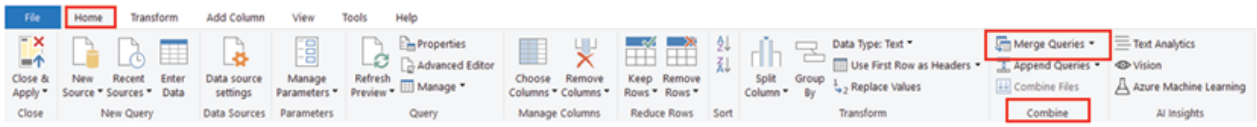



Figure 2.41: Merge Queries on query editor

The merge operation requires to select two tables, where the first table is considered as the Left table and the second table is considered as the Right After selecting the tables, the columns in each table need to be selected based on which the join will happen. Then, the Join Kind will require to be chosen, which determines how a merge operation will be performed. Figure 2.42 illustrates different join kinds available in Power Query

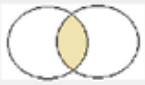
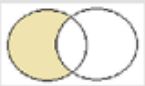
| Join Kind | Description | Venn Diagram |
|---|---|---|
| Left Outer | All rows from left table, matching rows from right table |  |
| Right Outer | All rows from right table, matching rows from left table |  |
| Full Outer | All rows from both tables |  |
| Inner | Only matching rows from both tables |  |
| Left Anti | Only rows from the left table |  |
| Right Anti | Only rows from the right table |  |

Figure 2.42: Available Join Kinds on query editor

For example, let us assume that we have two tables, dimension and where the dimension table has Country Id (as a key field) and country the fact table has date-wise information of the Country Code and as shown in Figure

Table: dimension

| Country ID | Name |
|---|---|
| 1 | USA |
| 2 | India |
| 3 | Canada |

Table: fact

| | Date | Country Code | Sales |
|---|---|---|---|
| 1 | 01-08-2022 | 2 | 10000 |
| 2 | 09-08-2022 | 3 | 5000 |
| 3 | 11-08-2022 | 1 | 8000 |
| 4 | 15-08-2022 | 2 | 2000 |

Figure Tables need to be merged

If we want to create a new merged table which should have the aggregated sales amount for each country, we can do a Left Outer join between the dimension table (left table) and the fact table (right table) based on the common Country Code columns, so that the aggregated sales amount gets populated for each country as in the dimension table. Figure 2.44 illustrates the merge operation that can be used:

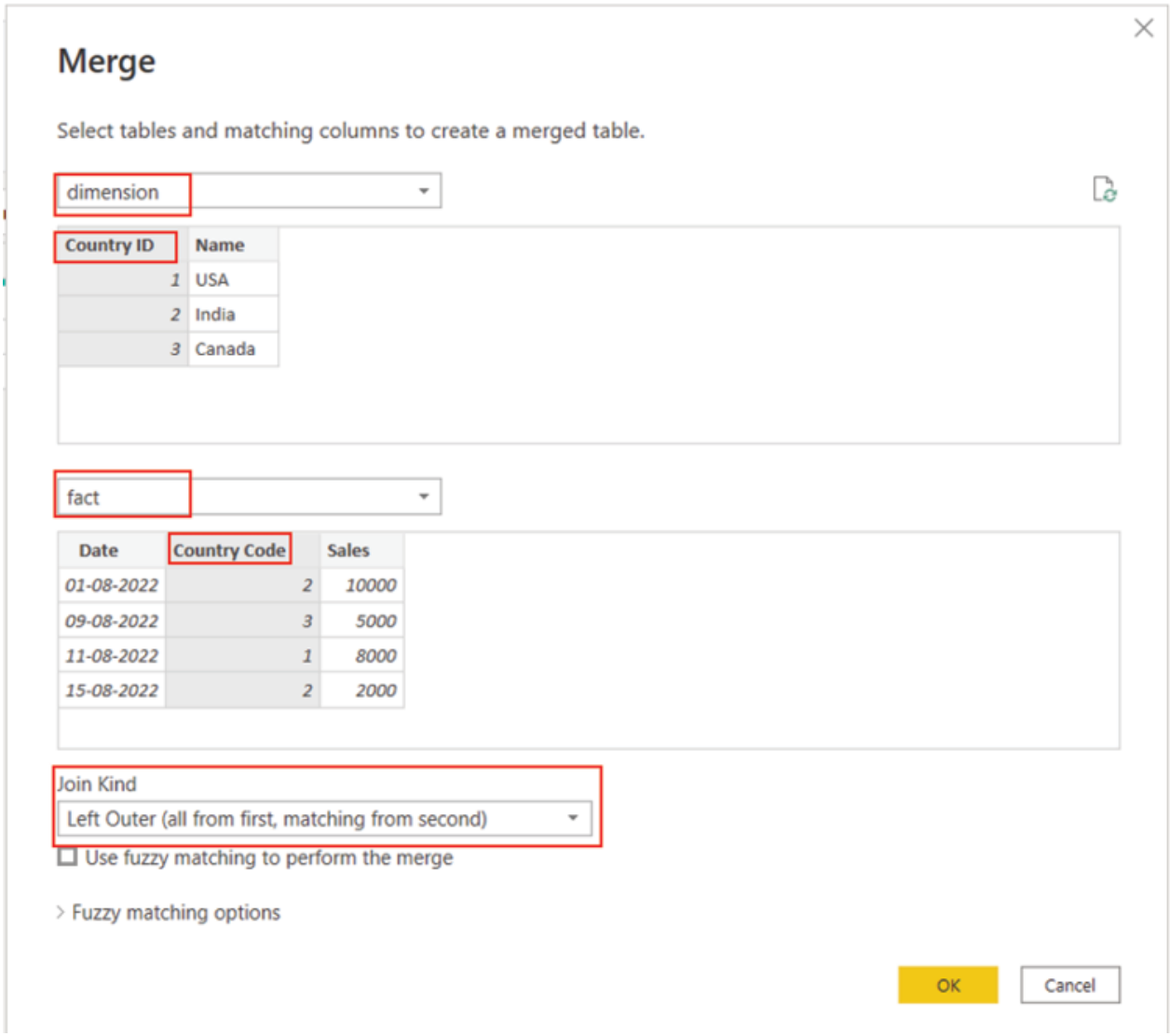



Figure 2.44: Merging in Query Editor

After selecting OK on the merge dialog box, the resulting table would have all columns from the left table (here and a new column would get added with the same name as the right table (here The column would hold the values of the fact table on a row-by-row basis. From that column, we can either Expand or Aggregate the fields as required. For this instance, we use Aggregate and choose Sum of Sales to be shown on the final table output, as illustrated in Figure


Figure 2.45: Aggregating the merged table column

Combining data can particularly be useful while working with multiple data sources; however, we should always consider the performance aspect as these transformations can increase the refresh time of the datasets, especially while working with large data volumes.

Tip: The Query Dependency view on query editor, which can be found under the View tab, can be referred to understand how different queries are linked together inside Power BI. For example, the query dependencies for the merge operation that we performed is illustrated in Figure

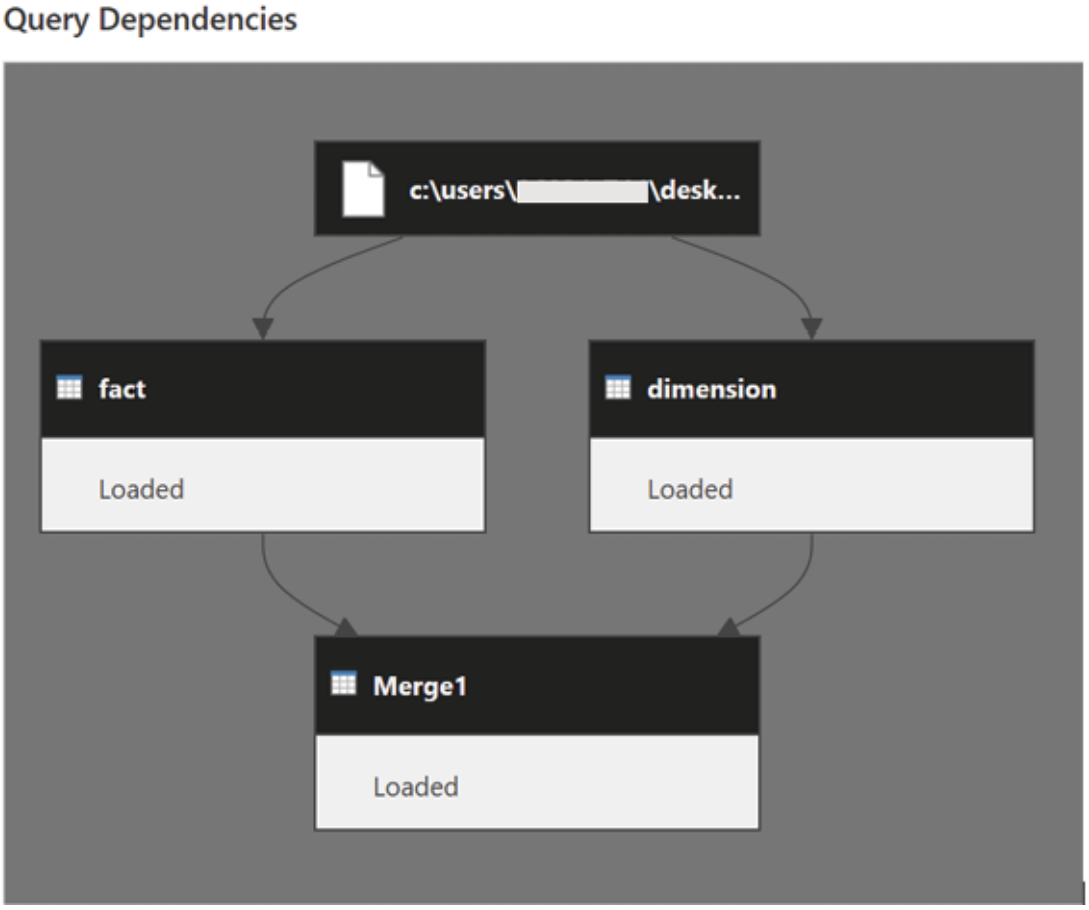



Figure 2.46: Query dependency view for the merge operation

Power Query parameters

One important feature of query editor is query which can store and manage a value which can be reused as required. Parameters allow to dynamically change the query output, depending on the value that it holds. Let us quickly understand the concept through an example.

Parameters can be created or managed using the Manage Parameters option under the Home tab. Consider we want to be able to import data specific for the country we want. For this, we can first create a parameter which would hold the name of the country we select. The value of the parameter can be a query output, any input by user or a list of possible values, as shown in Figure

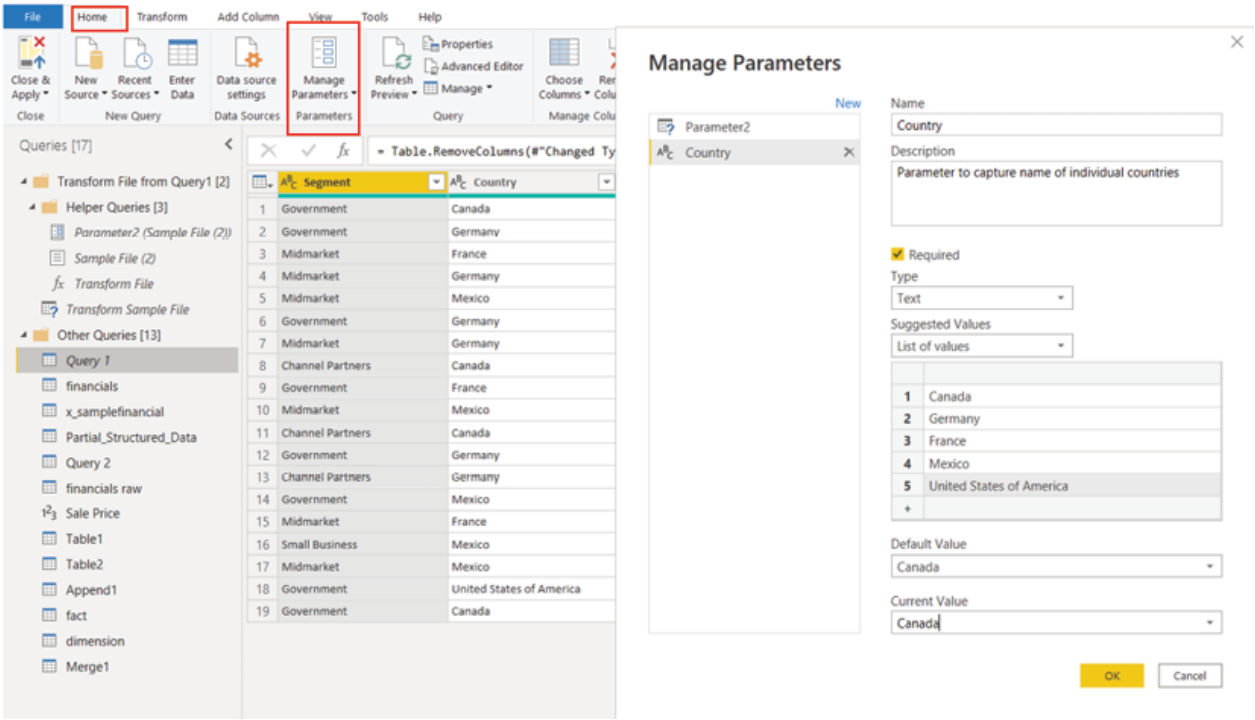



Figure 2.47: Creating a parameter for different countries

We have named the parameter as now to use the parameter we need to go to the column dropdown on which the filter needs to be applied, then clicking on Text Filter should populate the Filter Rows dialog box where the parameter can be applied, as shown in Figure

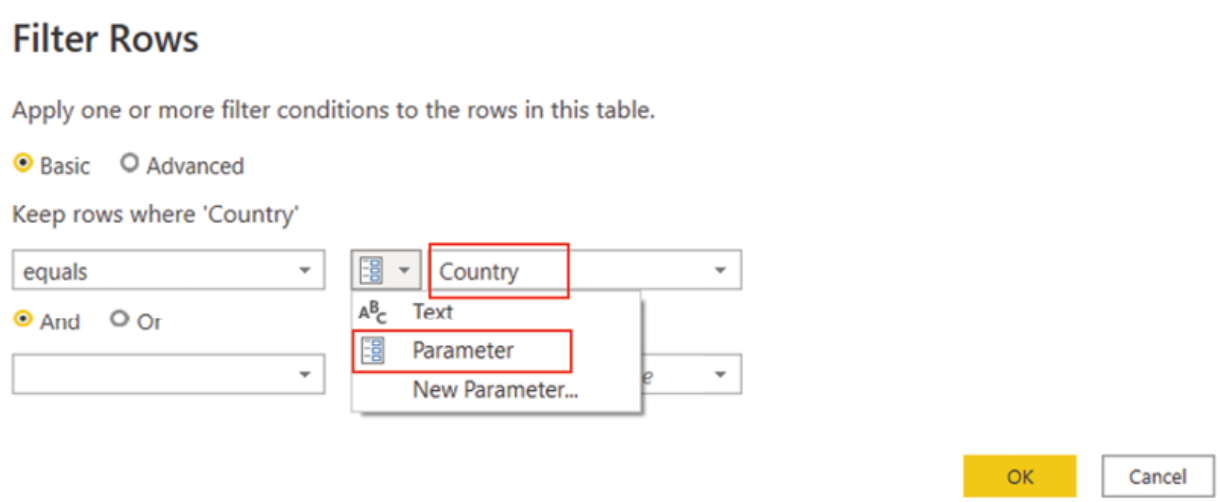



Figure 2.48: Filtering records using parameter

Once the column is filtered by the parameter, then each time the parameter value is changed, the dataset would import only the data relevant for the parameter, provided query folding is happening for the transformation step. In the subsequent chapters, we will see how to interact with the parameters from both Power BI Desktop as well as Power BI Service.

## Conclusion

In this chapter, we explored how to integrate with various external data sources from Power BI along with the architectural design considerations like connectivity modes. We have seen why concepts like query folding is important and how to perform transformations in suitable order to optimize performance. We discussed the Power Query native language M and how effective it can be to perform certain transformations. There is a plethora of transformation options available in the query editor, and we explored few commonly used ones, including those which are used for data cleansing. We also saw how data can be combined inside Power BI from multiple data sources. Finally, we discussed the usefulness of query parameters. With this knowledge, even business users should be able to integrate Power BI with external data sources and perform effective transformations on their own.

In the next chapter, we will discuss how to model the data that we are bringing into Power BI, so that ultimately, we can create those good-looking visuals on top of that data model.

## Knowledge check

The requirement is to create a Power BI report to show near real-time data. Which connectivity mode should ideally be used in this scenario?

Import mode

DirectQuery mode

Composite mode

In case a transformation step prevents query folding, the step should ideally be placed in which order?

Should be placed at the top

Order does not matter

After all the steps that fold

Query parameters are useful because:

They enable users to dynamically change the query output.

They ensure that query folding happens.

They can help to avoid writing M formulas.

Data can be combined in query editor only if they come from the same data source:

True

False

All Knowledge Check answers are provided at the end of the book.

**Join our book’s Discord space**

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 3

# Data Modeling in Power BI

## Introduction

So far, we have seen how to integrate Power BI with different data sources and then shape or transform the data as per reporting requirements and load the transformed data to Power BI Desktop. At this point, we may have multiple different tables which are not connected to each other and thus cannot really work in synergy, for which a data model is required. Power BI works best with a star schema instead of a single denormalized table; here in this chapter, we elaborate on the star schema design and its relevance to develop Power BI data models optimized for performance and usability. Also, different ways of implementing custom calculations using Data Analysis Expressions would be discussed to enhance the model.

## Structure

In this chapter, we will discuss the following topics:

Star schema overview

Relationships in the Power BI data model

Implementing a star schema in Power BI

Introduction to DAX

Calculated tables

Calculated columns

Measures

Optimizing models in Power BI

## Objectives

The objective of this chapter is to enable readers to create data models in Power BI without having any prior experience. Business users often do not have in-depth knowledge or expertise and are reluctant to perform data modeling exercises. A good data model is the key to create a sustainable report which also performs well. By the end of this chapter, readers should be aware of how to create an effective data model in Power BI, and how to enhance the model by creating calculated entities like calculated columns, calculated tables, and measures, to achieve specific visualization goals.

## Star schema overview

Firstly, it is not mandatory to have a star schema for creating a Power BI report. Not always it is possible to have a star schema depending on how the data is managed in the data source, and we may need to work with a single denormalized table which should be absolutely fine as well. However, Power BI loves a star schema, as it performs best with one.

In this chapter, we will explore how to implement a quick star schema data model from Power BI reporting perspective, without going into minute details about the star schema design which is a vast topic in itself. We would focus towards understanding what a star schema actually is and then see how to implement one in Power BI.

Star schema is a modeling approach which requires classifying the tables in scope as either fact or There is no table property available for the modeler to configure the table classification; it is actually determined by the relationships that exist between tables, as we will see later in this chapter.

In a star schema model, dimension tables describe business entities like products, people, time, and so on (for example, a date dimension table). A dimension table should contain one or more key columns, which act as unique identifiers. Apart from that, a dimension table usually has descriptive columns which represent additional information about the entity. Figure 3.1 illustrates how a typical date dimension table looks like:

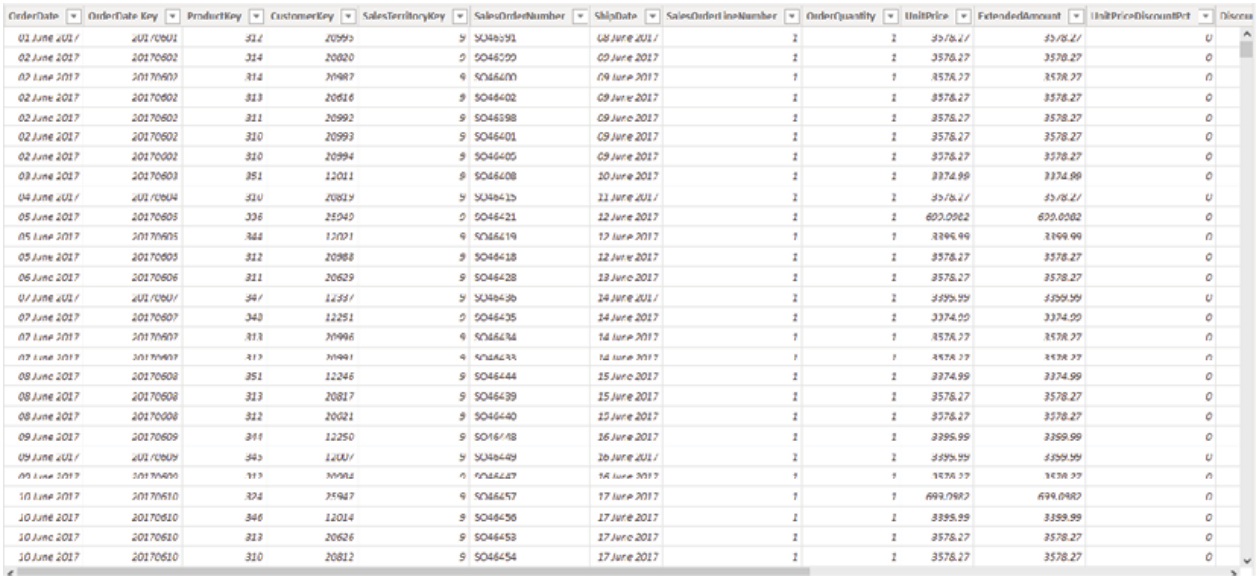
| ID | Date | DayNumberOfWeek | DayName | MonthName | MonthNumberOfYear | WeekNumberOfYear | CalendarQuarter | CalendarYear | Fiscal Year | DayNumberOfMonth | Period | Month ID |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 01 July 2016 | 6 | Friday | July | 7 | 27 | 3 | 2016 | 2017 | 1 | 201607 | 1 |
| 2 | 02 July 2016 | 7 | Saturday | July | 7 | 27 | 3 | 2016 | 2017 | 2 | 201607 | 1 |
| 3 | 03 July 2016 | 1 | Sunday | July | 7 | 27 | 3 | 2016 | 2017 | 3 | 201607 | 1 |
| 4 | 04 July 2016 | 2 | Monday | July | 7 | 28 | 3 | 2016 | 2017 | 4 | 201607 | 1 |
| 5 | 05 July 2016 | 3 | Tuesday | July | 7 | 28 | 3 | 2016 | 2017 | 5 | 201607 | 1 |
| 6 | 06 July 2016 | 4 | Wednesday | July | 7 | 28 | 3 | 2016 | 2017 | 6 | 201607 | 1 |
| 7 | 07 July 2016 | 5 | Thursday | July | 7 | 28 | 3 | 2016 | 2017 | 7 | 201607 | 1 |
| 8 | 08 July 2016 | 6 | Friday | July | 7 | 28 | 3 | 2016 | 2017 | 8 | 201607 | 1 |
| 9 | 09 July 2016 | 7 | Saturday | July | 7 | 28 | 3 | 2016 | 2017 | 9 | 201607 | 1 |
| 10 | 10 July 2016 | 1 | Sunday | July | 7 | 28 | 3 | 2016 | 2017 | 10 | 201607 | 1 |
| 11 | 11 July 2016 | 2 | Monday | July | 7 | 29 | 3 | 2016 | 2017 | 11 | 201607 | 1 |
| 12 | 12 July 2016 | 3 | Tuesday | July | 7 | 29 | 3 | 2016 | 2017 | 12 | 201607 | 1 |
| 13 | 13 July 2016 | 4 | Wednesday | July | 7 | 29 | 3 | 2016 | 2017 | 13 | 201607 | 1 |
| 14 | 14 July 2016 | 5 | Thursday | July | 7 | 29 | 3 | 2016 | 2017 | 14 | 201607 | 1 |
| 15 | 15 July 2016 | 6 | Friday | July | 7 | 29 | 3 | 2016 | 2017 | 15 | 201607 | 1 |
| 16 | 16 July 2016 | 7 | Saturday | July | 7 | 29 | 3 | 2016 | 2017 | 16 | 201607 | 1 |
| 17 | 17 July 2016 | 1 | Sunday | July | 7 | 29 | 3 | 2016 | 2017 | 17 | 201607 | 1 |
| 18 | 18 July 2016 | 2 | Monday | July | 7 | 30 | 3 | 2016 | 2017 | 18 | 201607 | 1 |
| 19 | 19 July 2016 | 3 | Tuesday | July | 7 | 30 | 3 | 2016 | 2017 | 19 | 201607 | 1 |
| 20 | 20 July 2016 | 4 | Wednesday | July | 7 | 30 | 3 | 2016 | 2017 | 20 | 201607 | 1 |
| 21 | 21 July 2016 | 5 | Thursday | July | 7 | 30 | 3 | 2016 | 2017 | 21 | 201607 | 1 |
| 22 | 22 July 2016 | 6 | Friday | July | 7 | 30 | 3 | 2016 | 2017 | 22 | 201607 | 1 |
| 23 | 23 July 2016 | 7 | Saturday | July | 7 | 30 | 3 | 2016 | 2017 | 23 | 201607 | 1 |
| 24 | 24 July 2016 | 1 | Sunday | July | 7 | 30 | 3 | 2016 | 2017 | 24 | 201607 | 1 |
| 25 | 25 July 2016 | 2 | Monday | July | 7 | 31 | 3 | 2016 | 2017 | 25 | 201607 | 1 |

Figure 3.1: Date dimension table

Here, the Date field is the unique identifier for the table while the other columns represent additional information like day name, month name, calendar year, and so on.

Fact tables store observations or transactions of events like sales, orders. A fact table should ideally contain the key columns from the dimension tables, as well as the numeric values. For example, let us consider a fact table which stores daily sales data having a dimension key column Order which represents the date on which the order has been placed. Now in the same model, we have the dimension table dim_date having a unique identifier Date which corresponds to Order Date in the fact table. Here, the values of the dimension key column would determine the granularity of the fact table, which would be at a daily level for this instance.

Usually, dimension tables have relatively fewer rows, while fact tables tend to have a large number of records as the transactional data continue to grow over time. Figure 3.2 shows a sample fact table containing dimension key columns like etc. as well as numeric fields like UnitPrice and



| OrderDate | OrderDate Key | ProductKey | CustomerKey | SalesTerritoryKey | SalesOrderNumber | ShipDate | SalesOrderLineNumber | OrderQuantity | UnitPrice | ExtendedAmount | UnitPriceDiscountPct | Discou |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 01 June 2017 | 20170601 | 312 | 20993 | 9 | SO46391 | 08 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 02 June 2017 | 20170602 | 314 | 20820 | 9 | SO46399 | 09 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 02 June 2017 | 20170602 | 314 | 20987 | 9 | SO46400 | 09 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 02 June 2017 | 20170602 | 313 | 20816 | 9 | SO46402 | 09 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 02 June 2017 | 20170602 | 311 | 20992 | 9 | SO46398 | 09 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 02 June 2017 | 20170602 | 310 | 20993 | 9 | SO46401 | 09 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 02 June 2017 | 20170602 | 310 | 20994 | 9 | SO46405 | 09 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 03 June 2017 | 20170603 | 351 | 12011 | 9 | SO46408 | 10 June 2017 | 1 | 1 | 3374.99 | 3374.99 | 0 | |
| 04 June 2017 | 20170604 | 310 | 20819 | 9 | SO46415 | 11 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 05 June 2017 | 20170605 | 336 | 25940 | 9 | SO46421 | 12 June 2017 | 1 | 1 | 699.0982 | 699.0982 | 0 | |
| 05 June 2017 | 20170605 | 344 | 12031 | 9 | SO46419 | 12 June 2017 | 1 | 1 | 3399.99 | 3399.99 | 0 | |
| 05 June 2017 | 20170605 | 312 | 20988 | 9 | SO46418 | 12 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 06 June 2017 | 20170606 | 311 | 20629 | 9 | SO46428 | 13 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 07 June 2017 | 20170607 | 347 | 12251 | 9 | SO46436 | 14 June 2017 | 1 | 1 | 3399.99 | 3399.99 | 0 | |
| 07 June 2017 | 20170607 | 348 | 12251 | 9 | SO46435 | 14 June 2017 | 1 | 1 | 3374.99 | 3374.99 | 0 | |
| 07 June 2017 | 20170607 | 313 | 20996 | 9 | SO46434 | 14 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 07 June 2017 | 20170607 | 313 | 20991 | 9 | SO46433 | 14 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 08 June 2017 | 20170608 | 351 | 12246 | 9 | SO46444 | 15 June 2017 | 1 | 1 | 3374.99 | 3374.99 | 0 | |
| 08 June 2017 | 20170608 | 313 | 20817 | 9 | SO46439 | 15 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 08 June 2017 | 20170608 | 312 | 20821 | 9 | SO46440 | 15 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 09 June 2017 | 20170609 | 344 | 12250 | 9 | SO46448 | 16 June 2017 | 1 | 1 | 3399.99 | 3399.99 | 0 | |
| 09 June 2017 | 20170609 | 343 | 12001 | 9 | SO46449 | 16 June 2017 | 1 | 1 | 3399.99 | 3399.99 | 0 | |
| 09 June 2017 | 20170609 | 313 | 20992 | 9 | SO46447 | 16 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 10 June 2017 | 20170610 | 324 | 25947 | 9 | SO46457 | 17 June 2017 | 1 | 1 | 699.0982 | 699.0982 | 0 | |
| 10 June 2017 | 20170610 | 346 | 12014 | 9 | SO46456 | 17 June 2017 | 1 | 1 | 3399.99 | 3399.99 | 0 | |
| 10 June 2017 | 20170610 | 313 | 20626 | 9 | SO46453 | 17 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |
| 10 June 2017 | 20170610 | 310 | 20812 | 9 | SO46454 | 17 June 2017 | 1 | 1 | 3578.27 | 3578.27 | 0 | |

Figure 3.2: Sample fact table

In a typical star schema model, there are usually one or more fact table(s) surrounded by multiple dimension tables connected via relationships between the key columns in the dimension table and the dimension key columns in the fact tables, resembling the shape of a star, as shown in Figure

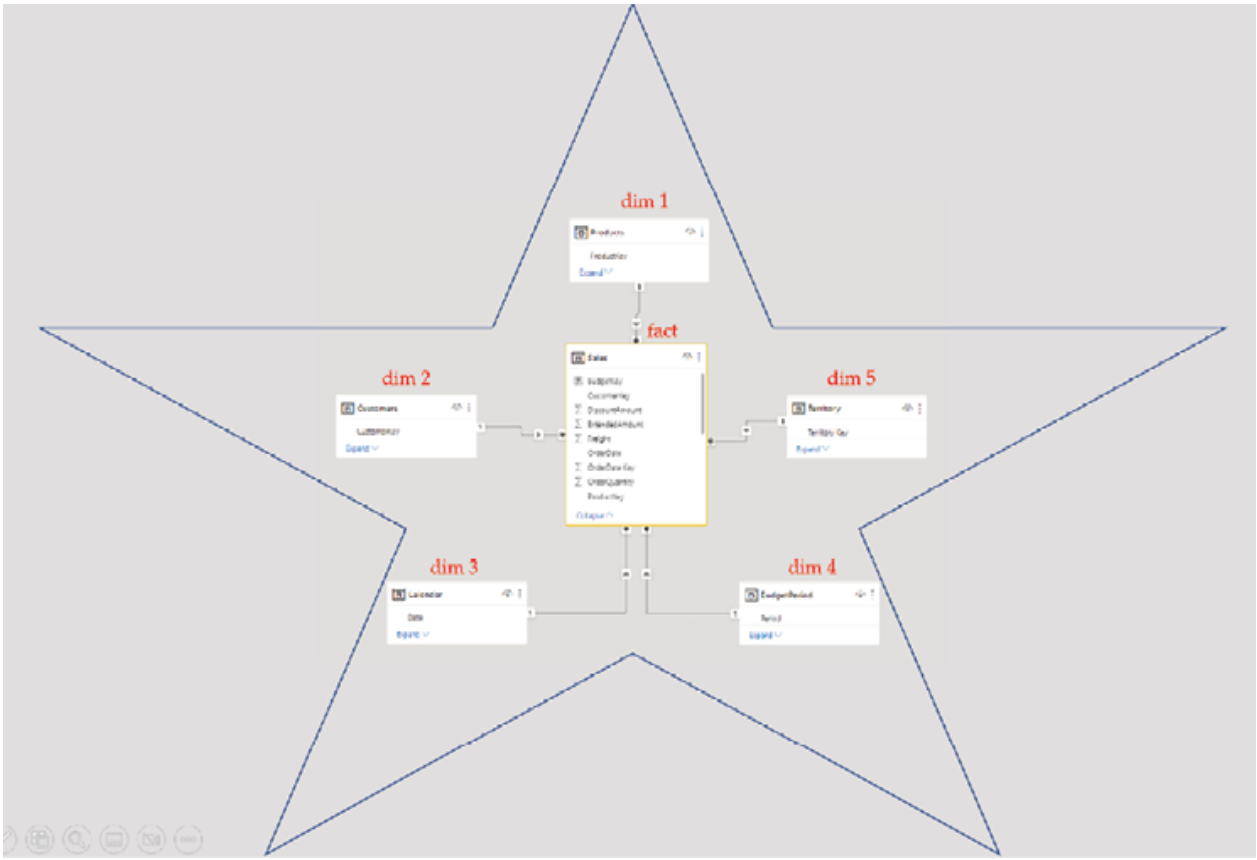



Figure 3.3: A typical star schema

Let us now try to understand the reasons for which a star schema is preferred by data modelers over a single table. In a star schema, the fact table(s) should be This term is used to describe data that is stored in a way to reduce repetitiveness or redundancy. Let us consider a dimension table product which has the ProductKey as the unique identifier, and additional columns which represent product characteristics like product name, colour, and so on. A fact table would be considered normalized when it would store only the dimension key column which is the ProductKey and not the additional characteristic columns from a dimension table. Figure 3.4 illustrates a normalized fact table

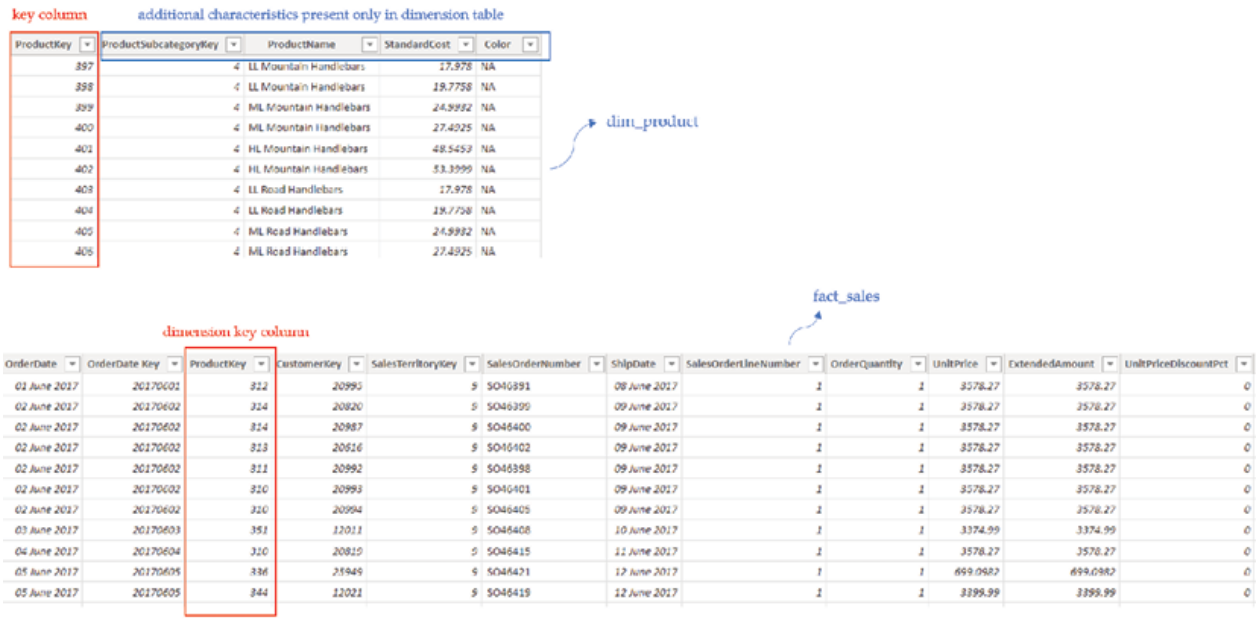



Figure 3.4: Normalized fact_sales

In this case, the fact table does not store additional characteristics related to the product, limiting the number of columns that it has to store. To create a star schema, fact_sales can be related to via a relationship involving fact_sales and dim_product This would allow these two tables to work in synergy and the product characteristics can be populated in run-time from alongside fact_sales values on a single visual, if required.

Contrarily, there could have been one single table fact_sales having all required product attributes in it. In that case for each record, the table would have to store all the product characteristics repeatedly. As fact tables have a large number of records which tend to grow over time; this approach would result into higher redundancy and inefficiency as each time a product is sold, all the additional product characteristics need to get stored against the same product key over and over again. However, in case of a star schema, there would be a normalized fact table and a dimension table with unique product keys, for this instance. The latter would serve the same purpose, only with improved performance and efficiency!

In Power BI, each visual generates a query that hits the underlying dataset for filtering and summarizing data. In a star schema design, generally dimension tables are used for filtering and grouping categorical data, while fact tables are used for summarization or aggregation of values. A typical star schema requires a one-to-many relationship cardinality between the dimension and the fact table; the concept we are going to understand in the upcoming section.

## Relationships in the Power BI data model

When a model uses multiple tables, it is likely that data from all the tables would be used in some way. Once data is loaded into Power BI, all the table schemas or metadata become visible under the Model view. It is the relationships between the tables which make it possible for the data model to work as a single unit and accurately calculate the aggregation results. In Power BI, a relationship can get created in two ways: either it can be auto detected, or it can be created manually:

Auto-detection during In case the data model has multiple tables and in Power BI Desktop the option to detect relationship is enabled, then Power BI tries to match the column names in different tables to determine any potential relationship during data load. If it finds a match with a high level of confidence, then it creates the relationship automatically, otherwise it does not. In case of the latter, any new relationship needs to be created manually inside the tool. In Power BI Desktop, File | Options and settings | Options | Data Load (CURRENT FILE) should populate the dialog box where relevant options can be found to control relationship detection, as shown in Figure

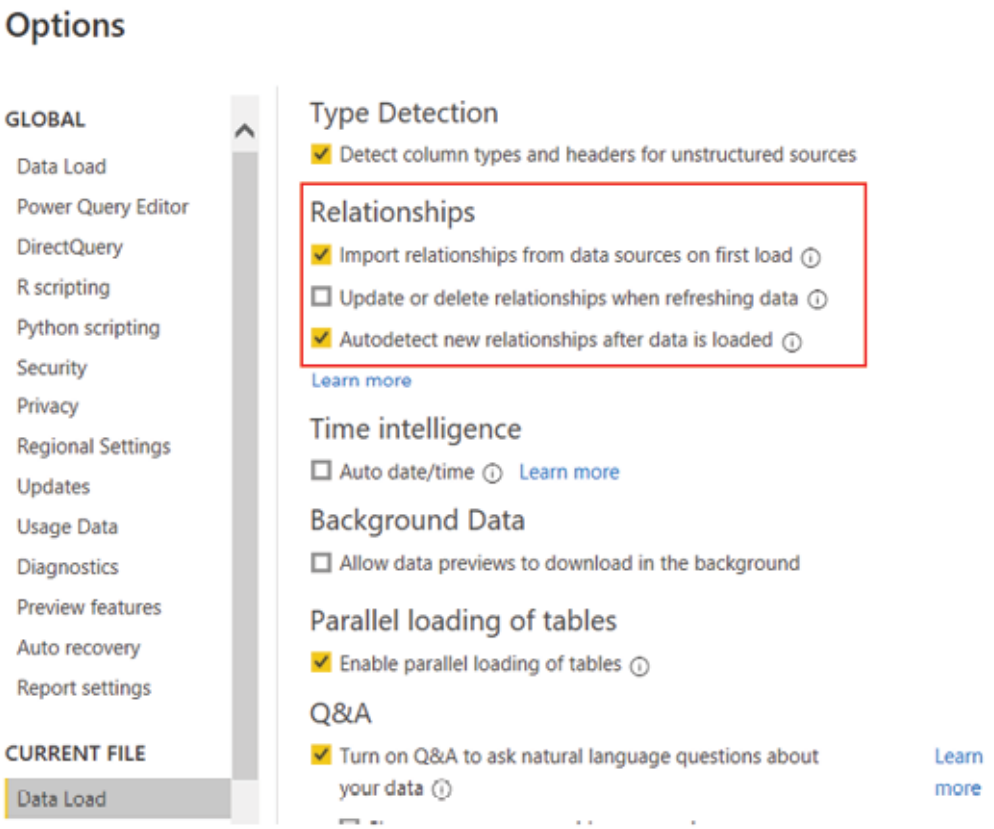



Figure 3.5: Autodetecting relationships

Enabling the relationship detection options can cause issues for various reasons like it might create relationships which do not make sense from a business perspective. And hence, probably it is safer not to blindly depend on autodetection, and manually verify or create all required relationships after data gets loaded to Power BI.

Manually creating Once the tables are loaded in Power BI, relationships can be created manually between them from the Model view of Power BI Desktop. In the Model view, selecting Manage relationships | New should bring up Create relationship dialog box where first both the tables need to be selected for which we want to create the relationship. After that on each table, we need to select the columns which will be used in the relationship. Once these are done, after clicking on BI will automatically configure other

relevant properties for a relationship like Cross filter direction and Active relationship – which we will understand in a while. Once a relationship is created, it can be seen as a line connecting the tables, on the canvas of the model view. Figure 3.6 illustrates the steps involved to create a new relationship manually:

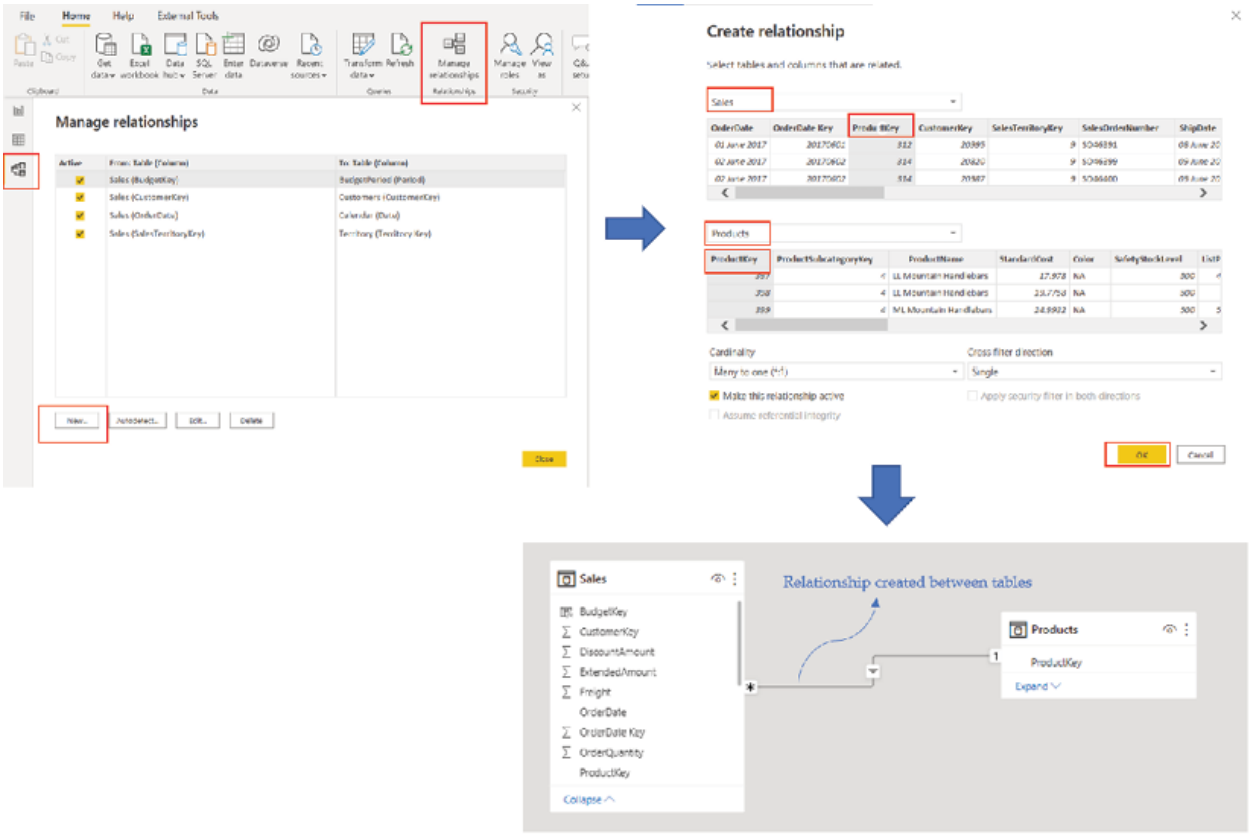



Figure 3.6: Creating a new relationship manually

When a relationship is created or edited, additional important options can be configured, which we will explore now:

Relationship cardinalities define how the columns involved in the relationship are connected to each other. The cardinality option can have one of the below configurations:

One to one In a one-to-one relationship, both the columns in the related tables have unique values (only one instance of a particular value).

One to many In a one-to-many relationship, the column in one table has only one instance of a particular value, while the related table column can have many instances of a value.

For example, let us assume we have a customer dimension table which holds details of all unique customers of a business. We also have a fact table for orders which has details of all the orders being placed. Now in the fact table, the customer ID is likely to get repeated as the same customer can place multiple orders. In case we relate these two tables based on customer ID, then the relationship between the dimension and the fact tables would be one-to-many, for this instance.

Many to one In a many-to-one relationship, a column in a given table can have multiple instances of the same value, while the related table column would have only one instance of a particular value.

A many-to-one relationship is similar to a one-to-many relationship, only viewed from a different perspective. Continuing with the previous customer and order table example, the relationship between the customer and order would be one-to-many while the relationship between the order and customer would be many-to-one. The latter can be interpreted as – multiple orders can be placed by the same customer.

Many to many Power BI supports creating a many-to-many relationship, which means that both the columns in the related tables can have multiple instances of a particular value.

As dimension tables should have unique values on the key field in a star schema, it is quite obvious that for a star schema model, the relationship cardinality would be many to one, from the fact to the dimension The use of many-to-many relationship cardinality should be avoided generally as that can introduce ambiguity. Alternatively, a bridge table can be created containing all the distinct key values, which can then be linked to both the original columns of the related tables.

Cardinalities are represented by 1 and an asterisk (*) to indicate one and many, respectively.

Cross filter The cross-filter direction option determines how any filter would propagate between the tables involved. The available configurations are:

The default and most common option, which means that when a table is filtered on the one (1) side of a relationship, the filter propagates to the table which is on the many (*) side of the relationship as well. However, filtering the many (*) side of the table does not automatically filter the one (1) side of the table.

For example, if a one-to-many relationship exists between the customer and the order table, with the cross-filter direction configured as the customer table would filter the order table, but not the other way round.

Also known as a bidirectional filter, ensures both the tables, on one (1) side as well as the many (*) side of a relationship, filters each other. This option should be considered carefully as it can have a negative performance impact and hence is not considered a best practice to follow for general data models.

Cross filter directions are represented by an arrow on the relationship line. The head of the arrow indicates the filter propagation direction.

Active/Inactive The relationship property Make this relationship when checked or enabled, the relationship serves as an active relationship, represented by a solid, continuous line. An active relationship enables Power BI to create visuals from both the related table objects. There can be only one active relationship between two tables at any point in time. However, there might exist inactive relationships, which programmatically can be activated and used during run-time. The Make this relationship active property needs to be checked off for any inactive relationship. An inactive relationship is represented by a dashed line between the tables.

One use case of having multiple relationships can be to avoid duplicating dimension tables in the model, for filtering the same fact table using different keys. For instance, there might be a requirement of filtering the same sales fact table by order date as well as by delivery Instead of having two date dimension tables in the model, two relationships can be created between the dim_date and fact_sales tables, where only one can be made active, let us say the relationship involving order When the fact table needs to be filtered by delivery the inactive relationship can be activated using the DAX USERELATIONSHIP function, which we will explore later in the book.

A relationship can be edited or modified by double clicking on the relationship line and can be deleted by right clicking on it and selecting

## Implementing a star schema in Power BI

Time to create an actual star schema data model in Power BI, which can be simple as a process in itself for one who has the required business understanding! Let us say we have a fact table fact_Sales and five dimension tables dim_BudgetPeriod and

Fact_Sales stores the daily transactional sales data and has dimension key columns like and Apart from these, the fact table stores values like unit sales amount, and so on. As required ideally, the dimension key columns are the unique identifiers in each of the respective dimension tables. That means ProductKey is a unique field in CustomerKey is a unique field in Territory Key is a unique field in Period is a unique field in dim_BudgetPeriod, and Date is a unique field in

Looking at the unique fields of the dimension tables, it seems that though few fields have slightly different names; however, as the values match with the corresponding dimension key columns of the fact table, refer to Table they may be related:

Table 3.1: Field mappings between fact and dimensions

At this point in time, we need to make a judgement whether relating or mapping these fields would make valid business sense or not, as once done, the model would start to behave as a single unit based on the relationship configurations chosen. For this instance, the mappings seem to make perfect

sense as we want to slice and dice our fact table's aggregated values by the dimension table filters based on the mappings as maintained in Table

To relate the tables, on the Model view, Manage Relationships | New should bring up the Create relationship dialog box where the mappings can be chosen and required relationship properties can be configured, as shown in Figure

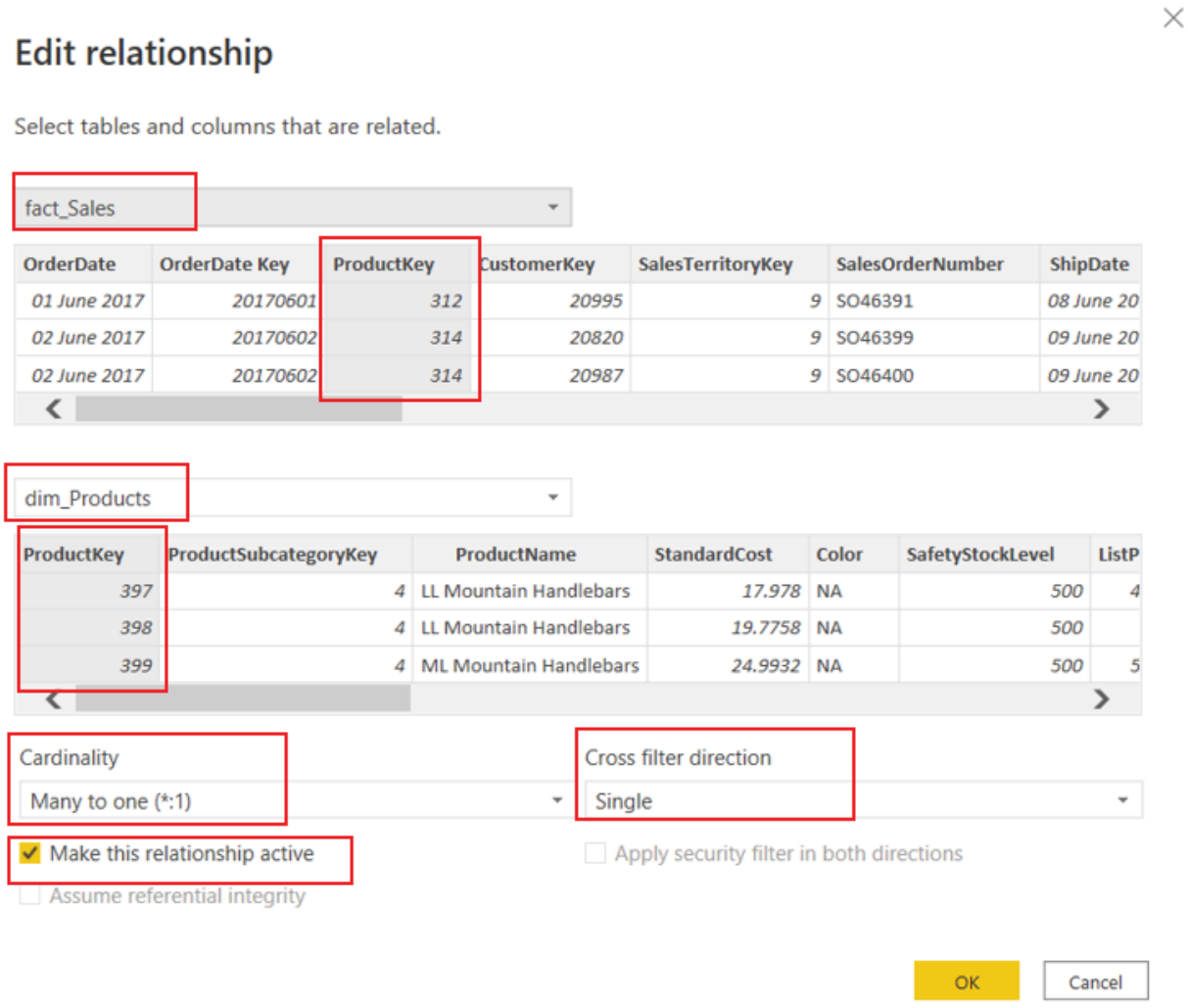



Figure 3.7: Relationship properties between fact_Sales and dim_Products

Here, we can see the relationship cardinality between fact_Sales and dim_Products is Many to one which indicates that there are many instances of

the same product in the fact table; however, the dimension table only has unique products stored, which is ideal for a star schema. The cross-filter direction is which signifies that dim_Products would filter fact_Sales based on exactly the way we want it. Lastly, the single relationship between the tables, by default would be an active relationship.

Once all required relationships are created between the tables, clicking on Manage relationships should populate the dialog box to manage existing relationships or to create any new relationship required. All relationship mappings for our model, as well as the model diagram, is illustrated in Figure

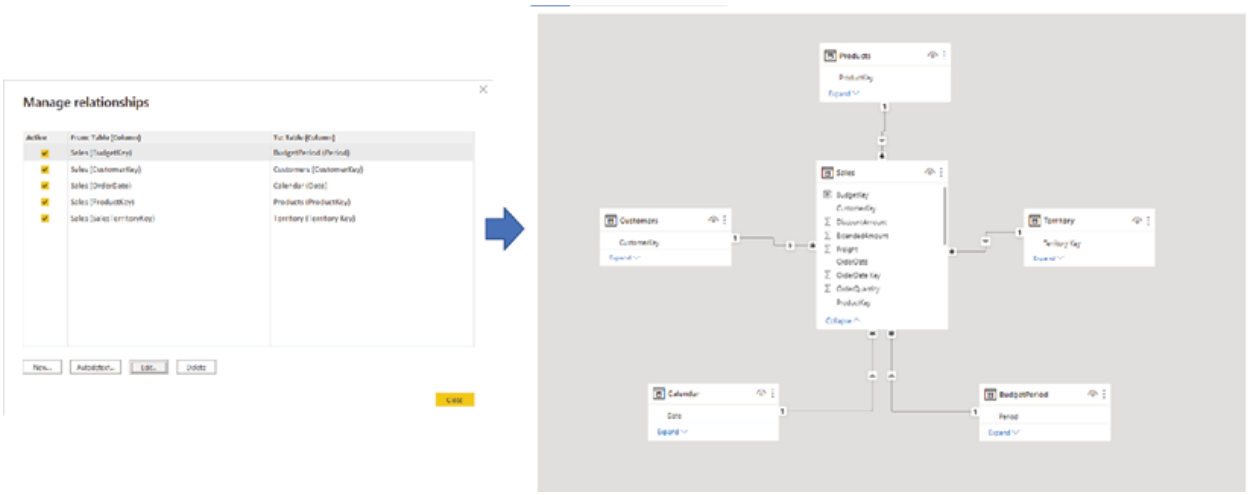



Figure 3.8: Star schema data model

At this point, all the tables in our data model is related to each other with valid relationships, and the model should be ready for reporting.

For complicated data models, there can be a large number of inter-related tables in the relationship view, hence it might increasingly become difficult to understand and maintain the network of relationships in a single diagram. To address this issue, the Model view allows you to create additional layouts while the default All tables layout includes all the tables used in the model.

Tables can be added to a layout by dragging from the right-hand side Fields pane, as shown in Figure


Figure 3.9: Additional model layout

The same Model view of Power BI Desktop can also be used to configure properties for any imported table by selecting that table, as shown in Figure

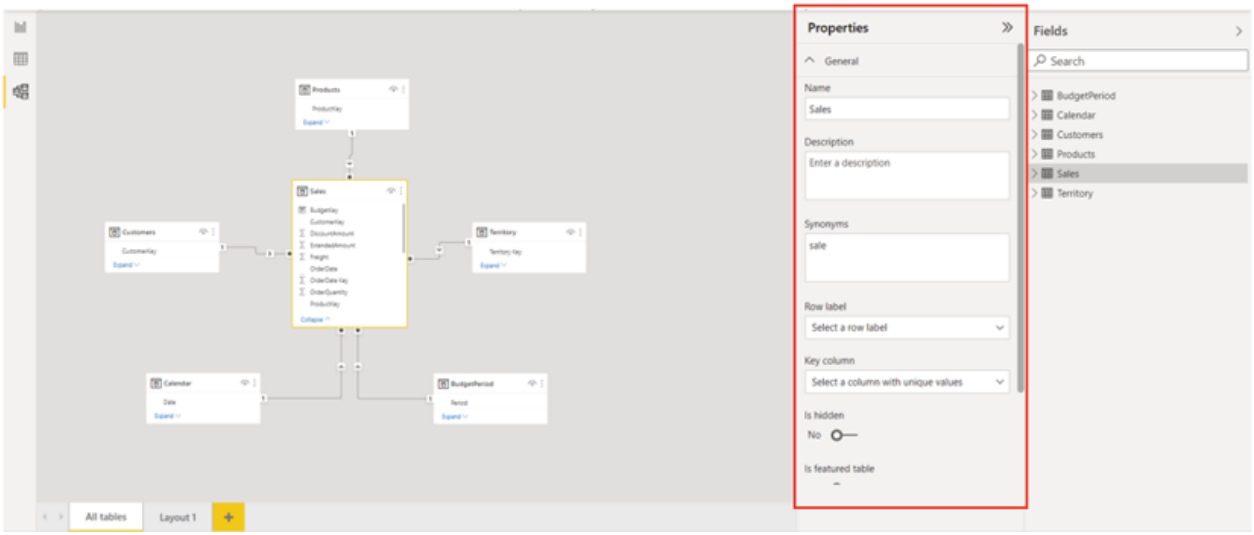



Figure 3.10: Configuring table properties

Now as we have seen how to create a quick and effective data model in Power BI, time to explore techniques that can help us to further enhance the model we have created so that it can meet the reporting requirements.

## Introduction to DAX

Having a well-built data model makes reporting a lot easier besides ensuring that all the calculations that are done on the reporting layer are accurate. Power BI allows you to implement custom business logic on the reporting layer by use of an expression language called DAX, which stands for Data Analysis Expressions.

Using DAX, a data model can be enhanced in various ways by introducing Calculated Calculated Columns and Apart from that DAX is also useful in terms of defining Row Level Security in Power BI. We will discuss each of these concepts in detail in subsequent sections and chapters of the book.

DAX is a functional language that to some extent resembles Excel formulas, and there are many common functions that appear in both. Unlike the Power Query M language, DAX is not case-sensitive in general and has the following important characteristics:

Unlike Excel, there is no concept of cells in DAX. To get a specific value from a column, appropriate filters would require to be applied to that column down to the value.

DAX does not allow to mix values of different data types in the same column.

DAX formulas or expressions can range from being very simple to extremely complicated. To build a DAX formula, the DAX formula editor can be used on Power BI Desktop which gets activated while creating a Calculated

Column or Calculated Figure 3.11 shows the DAX editor or formula bar on Power BI Desktop while creating a new measure:

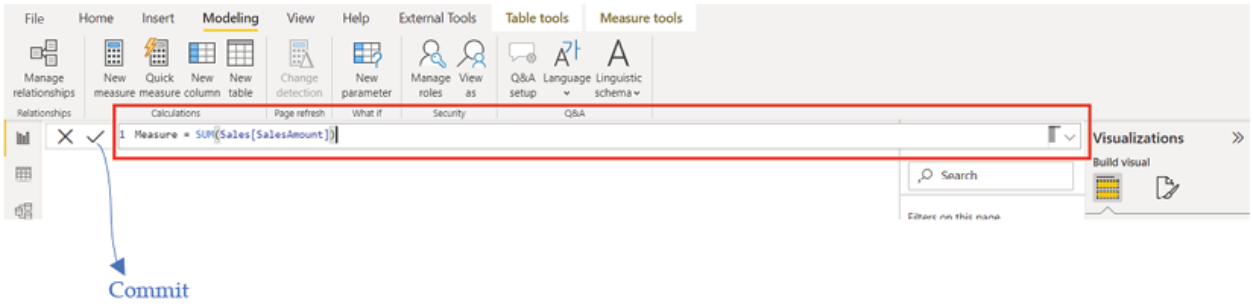



Figure DAX formula bar on Power BI Desktop

The following steps can be followed while creating any DAX formula:

Each formula must begin with an equal (=) sign, after specifying the name of the measure or calculated column or calculated table.

While typing the first few letters of any function name, AutoComplete displays a list of possible functions to choose from, which can be selected by using the Up/Down arrows and eventually added using the TAB key.

The AutoComplete feature also suggests a list of tables and columns, which can be passed as function arguments or used in the expression.

After completing the formula, pressing ENTER key or clicking the commit icon will save the formula if no syntax error has been found. AutoComplete does not rectify syntax errors like missing parenthesis, and so on.

DAX has a rich set of functions. A function always returns a value; however, many DAX functions return a table instead of a single value, which can be

used as input for other functions. Popular DAX functions can be categorized as follows depending on their usage:

Aggregation functions

Date-Time functions

Filter functions

Information functions

Logical functions

Relationship Management functions

Text functions

Table Specific functions

Time Intelligence functions

Other functions

We would dive deep into each of these categories and see examples of commonly used functions in a separate chapter of the book. Apart from that, whenever we are going to use a new DAX function from now on for the rest of the book, we will also discuss that function in detail.

When data is imported into a Power BI data model, the data is converted to a tabular model data type. When the data is used in a calculation, the data is then converted to a DAX data type. A few common data types supported by DAX are:

Decimal This can store both integers and fractions.

Whole This stores integers.

This stores dates and time.

This stores Text strings

This stores either a True or a False value.

DAX uses operators to create expressions that compare values or perform calculations. Operators in DAX can be categorized as follows:

Arithmetic Operators: Perform basic mathematical operations, example: Addition (+), Subtraction (-), Multiplication (*), Division (/) and so on.

Comparison Operators: Used for comparing values and returning a logical TRUE or FALSE as a result. The operators are Equal to (=), Greater than (>), Less than (<), Greater than or equal to (>=), Less than or equal to (<=), Not equal to (<>), and so on.

Concatenation Operator: The ampersand (&) or concatenation operator is used to join two or more text strings together and produce a single text.

Logical Operators: Logical operators like AND/OR are used to combine expressions.

Now that we have a brief idea about what DAX is, time to see it in action and understand more about it along the way!

## Calculated tables

Calculated tables are computed dynamically based on a DAX expression and can be derived from other tables in the data model, which can be very useful in terms of enhancing the data model which is already in place. Instead of loading values from the data source, in calculated tables, data is loaded from other imported tables. Once loaded, it behaves exactly the same way as any other table in the model and supports relationships with other tables. Calculated tables are re-calculated if any of the underlying tables, it pulls data from are refreshed or updated.

As mentioned earlier, some of the DAX functions return tables and can be used to create calculated tables to implement different business logic.

To create a calculated table, the DAX formula editor can be enabled or activated from both the Report view and Data view of the Power BI Desktop. From the Report view, Modeling | New table should activate the formula editor, while from the Data view Home | New table should do the trick. One advantage of creating a calculated table from the Data view is that the user would be able to immediately view the newly created table. Figure 3.12 illustrates how to activate the DAX formula editor for creating a calculated table from the Data view:

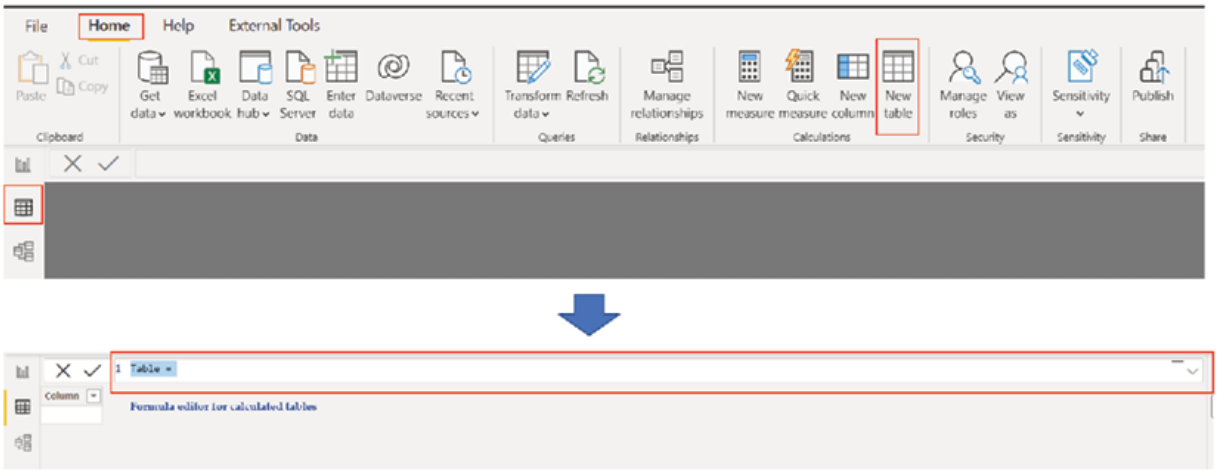



Figure Enabling DAX formula bar to create a calculated table

We will see now few scenarios which require to create a calculated table and how we can create those using DAX:

Creating a calculated duplicate In case we need to duplicate a dimension table (example: for the data model that we have, instead of importing it again from the source, we can simply create a calculated table which will get populated with values from the original dim_Calendar table and will get updated automatically whenever dim_Calendar gets refreshed.

To create a duplicate or referenced table, on the formula editor for calculated tables, we need to provide the new table name and then start typing the name of the table to which we want to refer. The AutoComplete should prompt the matching table list from where the required table can be selected. After pressing ENTER or hitting the Commit button, the table can be created, as shown in Figure

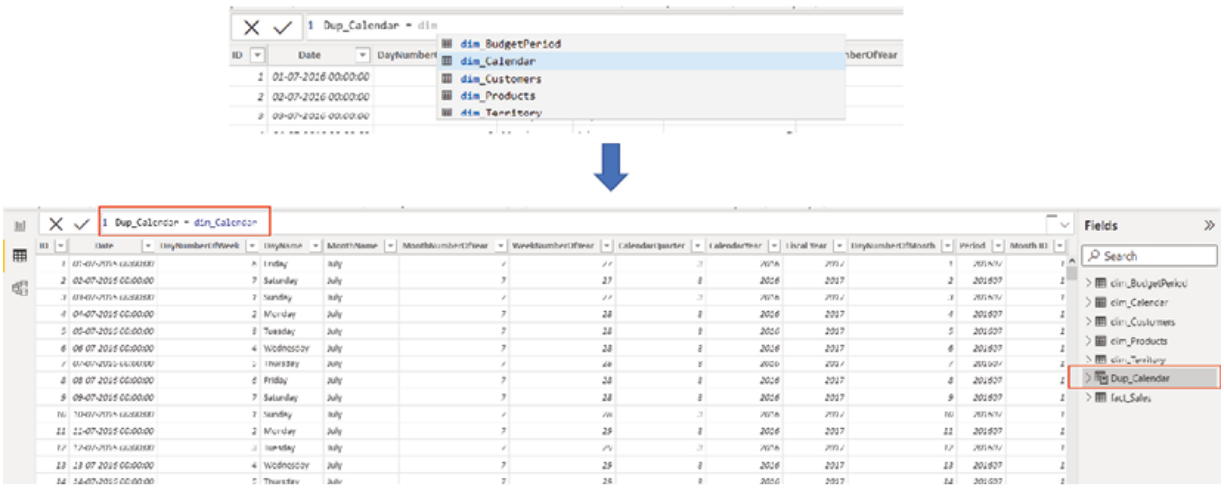



Figure Creating a calculated duplicate table

It can be noticed on the right-hand side Fields pane that the icon for the calculated table is different than that of a regular imported table.

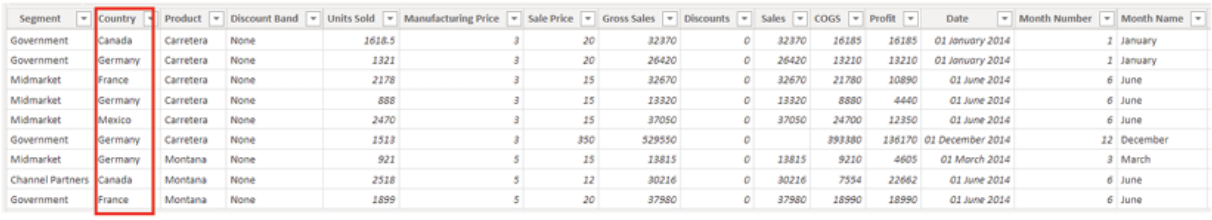
Creating a calculated lookup We may have a requirement of having a lookup table in the model with all the unique values for a specific field. Let us say we have a fact table which records all financial transactions, and we need to have a single column table listing all the unique countries where the transactions took place. We can simply create a calculated table from the original fact table using the DISTINCT function of DAX.

DISTINCT: This returns a one-column table containing distinct values from the column which is passed as a parameter to this function.

Syntax: DISTINCT()

Parameter: The column from which unique values are to be returned.

Figure 3.14 shows a preview of the actual fact table



| Segment | Country | Product | Discount Band | Units Sold | Manufacturing Price | Sale Price | Gross Sales | Discounts | Sales | COGS | Profit | Date | Month Number | Month Name |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Government | Canada | Carretera | None | 1618.5 | 3 | 20 | 32370 | 0 | 32370 | 16185 | 16185 | 01 January 2014 | 1 | January |
| Government | Germany | Carretera | None | 1321 | 3 | 20 | 26420 | 0 | 26420 | 13210 | 13210 | 01 January 2014 | 1 | January |
| Midmarket | France | Carretera | None | 2178 | 3 | 15 | 32670 | 0 | 32670 | 21780 | 10890 | 01 June 2014 | 6 | June |
| Midmarket | Germany | Carretera | None | 888 | 3 | 15 | 13320 | 0 | 13320 | 8880 | 4440 | 01 June 2014 | 6 | June |
| Midmarket | Mexico | Carretera | None | 2470 | 3 | 15 | 37050 | 0 | 37050 | 24700 | 12350 | 01 June 2014 | 6 | June |
| Government | Germany | Carretera | None | 1513 | 3 | 350 | 529550 | 0 |  | 393380 | 136170 | 01 December 2014 | 12 | December |
| Midmarket | Germany | Montana | None | 921 | 5 | 15 | 13815 | 0 | 13815 | 9210 | 4605 | 01 March 2014 | 3 | March |
| Channel Partners | Canada | Montana | None | 2518 | 5 | 12 | 30216 | 0 | 30216 | 7554 | 22662 | 01 June 2014 | 6 | June |
| Government | France | Montana | None | 1899 | 5 | 20 | 37980 | 0 | 37980 | 18990 | 18990 | 01 June 2014 | 6 | June |

Figure 3.14: Preview of underlying fact table

In the formula editor, after providing the calculated table name (ex: once we select the DISTINCT function, AutoComplete should suggest all the columns that are available to be used in the function. We need to choose the Country column from the fact table, as that is the column from which we want to get our unique values, and hence need to be passed as a parameter to the function.

After pressing ENTER, we should be able to see our calculated table having all the unique country names as shown in Figure

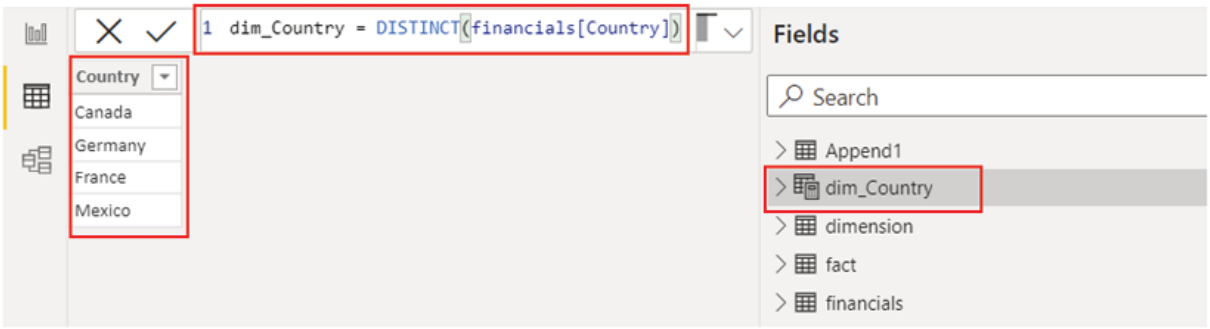



Figure 3.15: Calculated table dim_Country

Creating a calculated combined Let us now see another use case of calculated tables. We already have seen how to append table in the Power Query editor in the previous chapter. It is possible to combine multiple table on the reporting layer as well if required, using the DAX UNION function.

UNION: Create a combined table from a pair of tables having the same number of columns.

Syntax: UNION(, )

Parameter: Any DAX expression that returns a table.

Let us assume that we have two dimension tables in our model, Product1 and which store different product names and the corresponding prices. We can combine these two files and create a single dim_Product table in the reporting layer by creating a calculated table. After selecting the UNION function in the formula editor, we can select the two individual tables as function parameters to create the calculated combined table, as shown in Figure

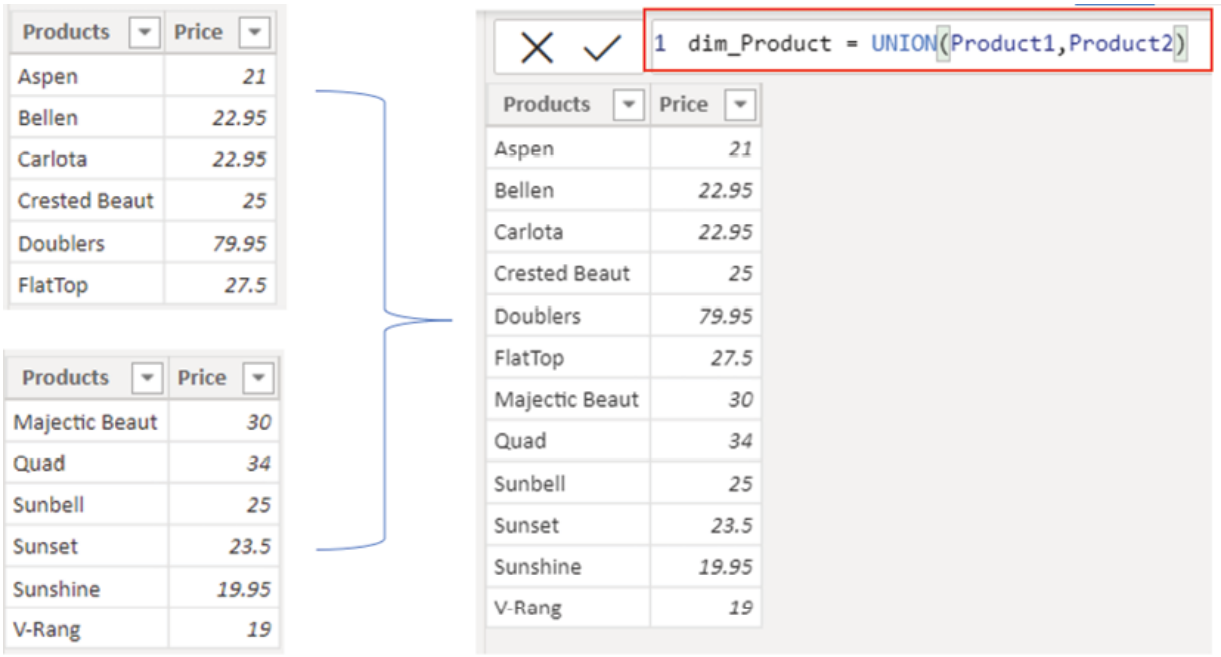



Figure 3.16: Calculated combined table using UNION

The column names in the return table will match the column names in table_expression1 and the columns would be combined by position in their respective tables. UNION will retain any duplicate rows that are present in the tables.

Creating a calculated subset One of the most common requirements while working towards implementing business logic come in the form of the ability to be able to reduce the number of records in a table and use only a subset of the data for calculations. This can be done by creating a subset table using the FILTER function of DAX.

FILTER: This returns a table that represents a subset of another table or expression.

Syntax: FILTER(, )
Parameters:

The table (or expression returning a table) to be filtered.

An expression that is to be evaluated for each row of the table.

Let us say we want to create a subset of our financial fact table (refer to Figure where we want to keep records only for a specific country, ex: For this, after selecting the FILTER function in the formula editor, we need to pass two parameters required for the function, which are, the table that we want to filter and the filter expression: financials[Country] = After executing the formula, the calculated subset table can be seen as shown in Figure

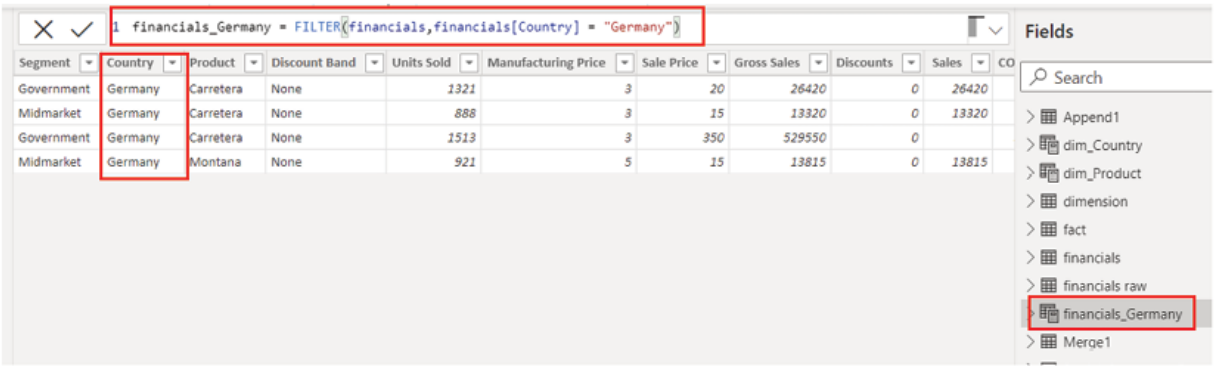



Figure 3.17: Creating calculated subset table using FILTER

The FILTER function is mostly used in combination with other functions which require a table as an argument, we will further explore it in detail later in time while discussing

In this section, we have seen how to make use of calculated tables using DAX. Time to move to the next element in the list, which is, Calculated

## Calculated columns

A calculated column is an additional column that can be defined in a table using DAX, based on the already loaded data in the model. Once created, the calculated columns would appear on the right-hand side Fields pane of Power BI Desktop just like any other column of a table; however, it would be represented by a separate icon than what represents a regular column, indicating that the values of that column are the result of a DAX formula.

Calculated columns work in a row which means the formula is evaluated for each row of a table to which the column is added. The column values would be recalculated whenever the underlying data or the model is refreshed.

To open the DAX formula editor for creating a calculated column, both the Report view and Data view can be used. From the Report view, Modeling | New column should launch the editor, while from the Data view, Home | New column can be followed. Figure 3.18 illustrates how to activate the DAX formula editor for creating a calculated column from the Data view:

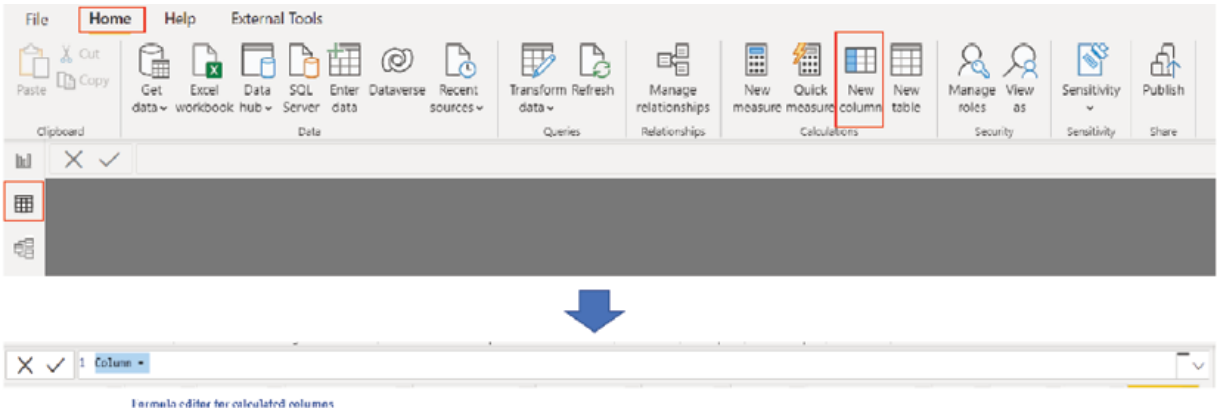


Figure Enabling DAX formula bar to create a calculated column

Let us now go through some examples to understand the concept further.

Concatenating Fields: In our financials fact table, we have the Month Name and Year as two separate fields. The requirement is to create a filter on the report based on a ‘Month-Year’ field, for example, ‘January-2022’. The table does not have any such field at the moment. A calculated column can be created here using the DAX ampersand operator to solve the problem.

After selecting the fact table and enabling the formula editor to create a calculated column, the desired column name needs to be provided followed by an equals (=) operator, and then the actual logic needs to be written. Here we just need to provide the name of the two columns we need to concatenate, also need to introduce a hyphen (-) as a separator. Once we start typing the column names, the AutoComplete or IntelliSense should prompt the available columns which we can then select. Figure 3.19 illustrates the Month-Year calculated column created using DAX:

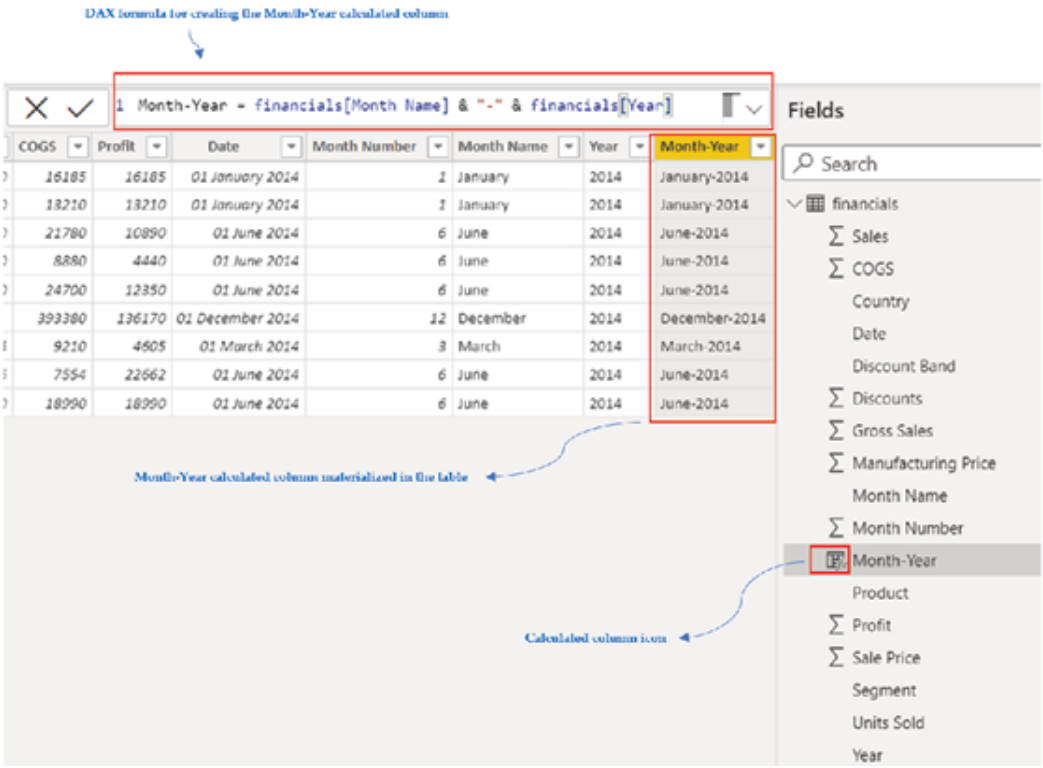



Figure 3.19: Creating a calculated column by concatenating two fields

Tip: In Power BI, columns of a table should be referred using a fully qualified syntax, which is the table name followed by the column name enclosed in square brackets.

Creating a referenced calculated column from a related table: Let us say we have our fact_Sales table, related with dim_Customers using the CustomerKey field (refer Figure 3.8). The cardinality is Many-to-one, from fact_Sales to dim_Customers as expected in a star schema.

We can refer to a column from a table which is in a one-to-many relationship with the current table, using the RELATED function of DAX.

RELATED: This returns a related value from another table.

Syntax: RELATED()

Parameter: The column that needs to be referred.

For instance, the customer names, which is a field in can be referred from the fact_Sales table as shown in Figure

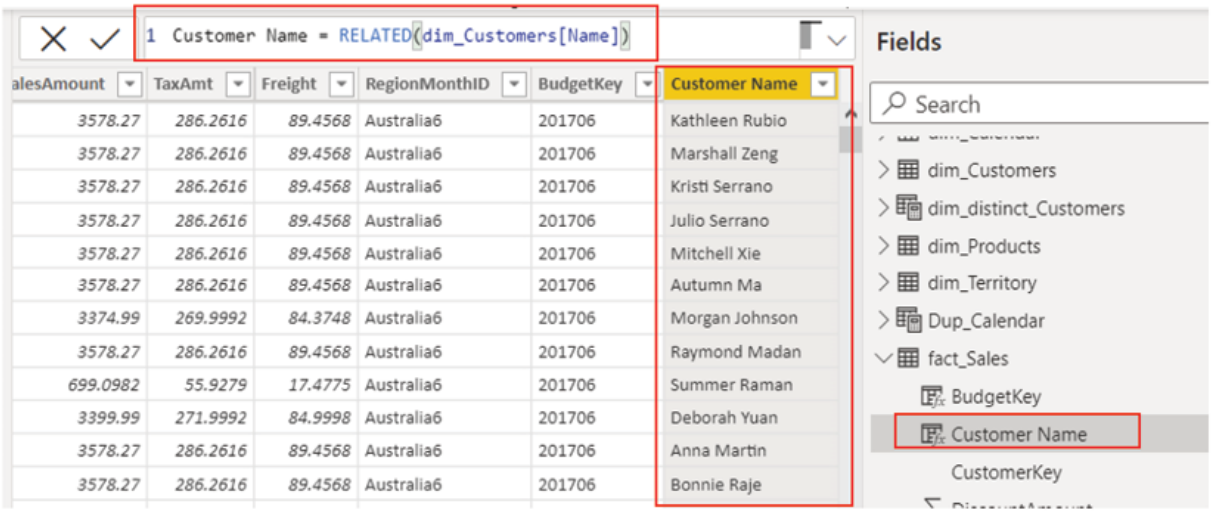



Figure 3.20: Using RELATED to refer a field from another table

The RELATED function is generally used in combination with other functions. One important downside of using calculated columns in a model is that they can increase the model size as each column would be materialized in the data model itself, resulting increased file size. Hence, it is recommended to avoid creating calculated columns in fact tables (as they tend to grow over time), especially for large data models.

Creating a calculated column with conditions: Now, let us see how to implement some business logic in a calculated column using DAX. Going back to our fact table financials, let us assume that we have two columns there, Gross Sales and Discounts. The requirement is that we need to able to filter the visuals of our reports by discount categories like

High/Medium/Low. The problem here is that we do not actually have any such column in our model, hence again calculated columns come to the rescue!

In order to create the discount categories or let us call it discount bucket, first we need to have an understanding of what the different categories mean. Let us say Low means transactions for which up to 5% discounts have been offered, Medium means more than 5% and less than or equal to 10% discount while anything above 10% should be categorized as High. Time to create the column now! First, we will see how it can be done step by step and then, we will look at a more efficient way to do the same.

Let us figure out first the percentage of discount that has been offered for each row of the table, using the DIVIDE function of DAX.

DIVIDE: Performs division and returns the alternate result or BLANK on division by 0.

Syntax: DIVIDE(, [,result>])

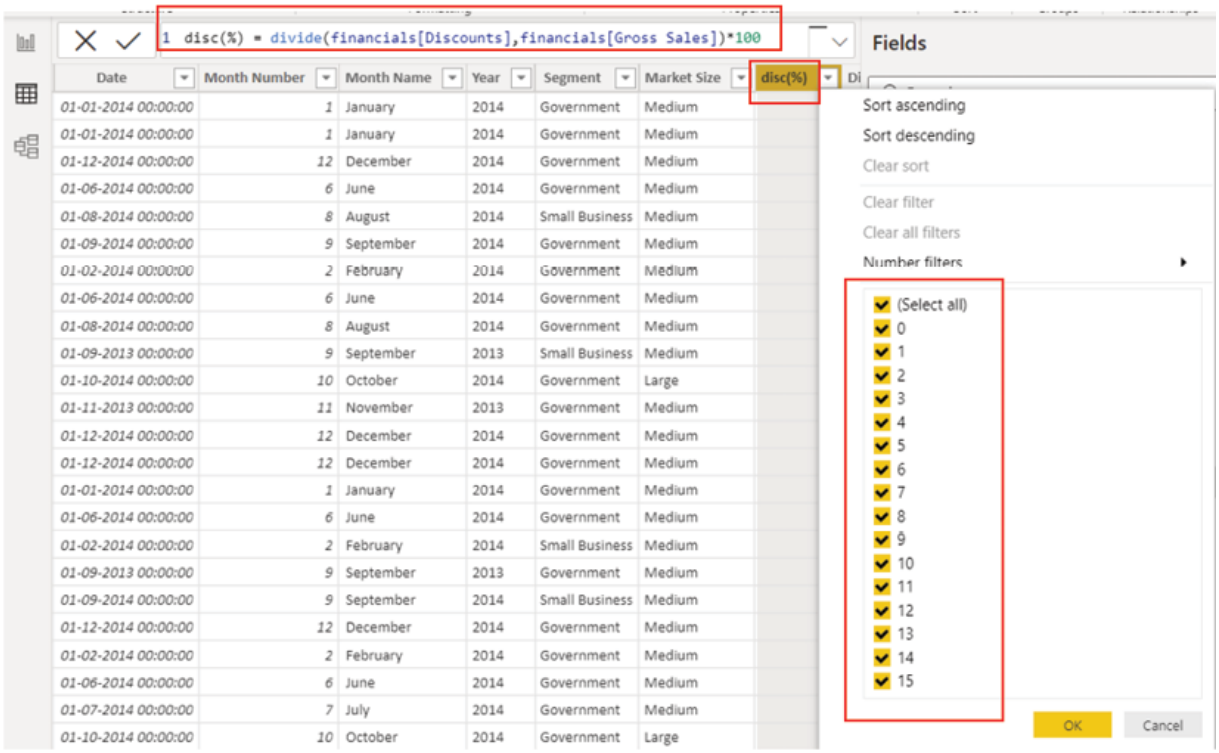
Parameters:

Numerator is the dividend or number to divide.

Denominator is the divisor or number to divide by.

The 3rd parameter is optional. When provided, the value is returned instead of an error while dividing by 0, else a BLANK is returned.

In the formula editor for calculated columns, the following formula as shown in Figure 3.21 should create the disc% calculated column in the table, which should hold the discount percentages for each row of the table ranging between 0% to 15%:




| Date | Month Number | Month Name | Year | Segment | Market Size | disc(%) |
|---|---|---|---|---|---|---|
| 01-01-2014 00:00:00 | 1 | January | 2014 | Government | Medium | |
| 01-01-2014 00:00:00 | 1 | January | 2014 | Government | Medium | |
| 01-12-2014 00:00:00 | 12 | December | 2014 | Government | Medium | |
| 01-06-2014 00:00:00 | 6 | June | 2014 | Government | Medium | |
| 01-08-2014 00:00:00 | 8 | August | 2014 | Small Business | Medium | |
| 01-09-2014 00:00:00 | 9 | September | 2014 | Government | Medium | |
| 01-02-2014 00:00:00 | 2 | February | 2014 | Government | Medium | |
| 01-06-2014 00:00:00 | 6 | June | 2014 | Government | Medium | |
| 01-08-2014 00:00:00 | 8 | August | 2014 | Government | Medium | |
| 01-09-2013 00:00:00 | 9 | September | 2013 | Small Business | Medium | |
| 01-10-2014 00:00:00 | 10 | October | 2014 | Government | Large | |
| 01-11-2013 00:00:00 | 11 | November | 2013 | Government | Medium | |
| 01-12-2014 00:00:00 | 12 | December | 2014 | Government | Medium | |
| 01-12-2014 00:00:00 | 12 | December | 2014 | Government | Medium | |
| 01-01-2014 00:00:00 | 1 | January | 2014 | Government | Medium | |
| 01-06-2014 00:00:00 | 6 | June | 2014 | Government | Medium | |
| 01-02-2014 00:00:00 | 2 | February | 2014 | Small Business | Medium | |
| 01-09-2013 00:00:00 | 9 | September | 2013 | Government | Medium | |
| 01-09-2014 00:00:00 | 9 | September | 2014 | Small Business | Medium | |
| 01-12-2014 00:00:00 | 12 | December | 2014 | Government | Medium | |
| 01-02-2014 00:00:00 | 2 | February | 2014 | Government | Medium | |
| 01-06-2014 00:00:00 | 6 | June | 2014 | Government | Medium | |
| 01-07-2014 00:00:00 | 7 | July | 2014 | Government | Medium | |
| 01-10-2014 00:00:00 | 10 | October | 2014 | Government | Large | |

Figure 3.21: Creating a calculated column to compute the discount percentage for each row

Once created, this calculated column can be further used as an input for another calculated column which we can create to compute the desired discount bands. To do that, we will use the IF function of DAX.

IF: This validates a condition and returns a value when TRUE, otherwise returns an alternate value.

Syntax: IF(, [, ])

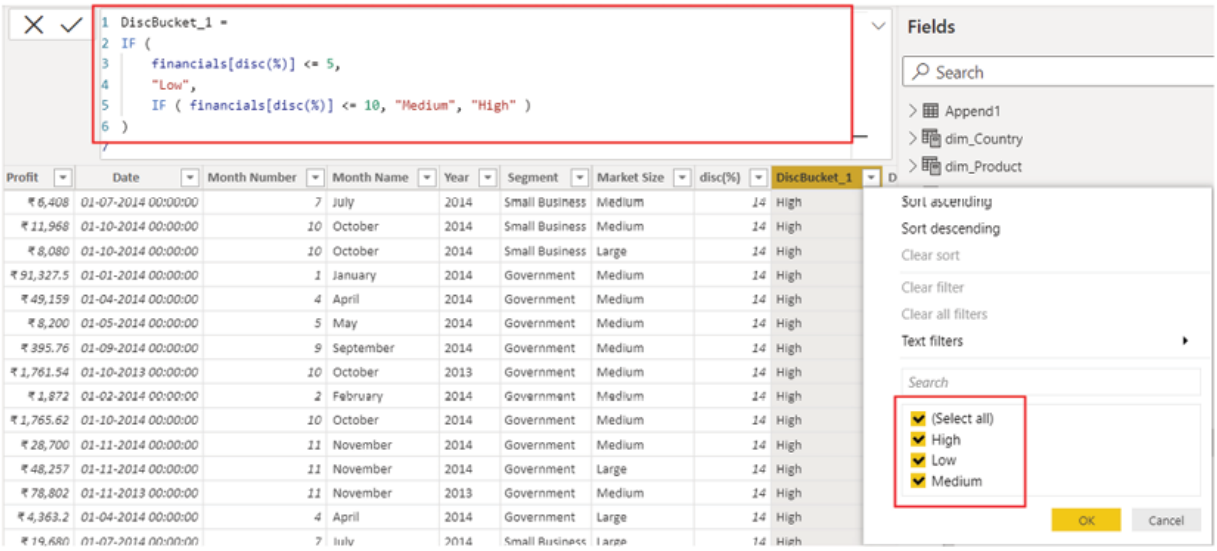
Parameters:

Any expression that returns either TRUE or FALSE.

The value to be returned if a logical test is TRUE.

The value to be returned if a logical test is FALSE. This is optional and if omitted, BLANK is returned.

Before looking at the formula, let us first try to understand how to construct it. Firstly, as a logical test, we will check for each row whether the discount percentage is less than or equal to 5% or not. If that is TRUE, then we will return the value If for any given row, the first logical test becomes FALSE (which means the percentage is more than 5%), then we are going to perform a second logical test using a nested IF function (which is an IF function inside another IF function), which will check whether the discount percentage is less than or equal to 10% or not. In case the second test becomes TRUE, we would return the value Medium for that row. For everything else, which will include rows where the discount percentage is more than 10%, the value returned would be Figure 3.22 illustrates the formula as well as shows the calculated column with different discount bands:




| Profit | Date | Month Number | Month Name | Year | Segment | Market Size | disc(%) | DiscBucket_1 |
|---|---|---|---|---|---|---|---|---|
| ₹6,408 | 01-07-2014 00:00:00 | 7 | July | 2014 | Small Business | Medium | 14 | High |
| ₹11,968 | 01-10-2014 00:00:00 | 10 | October | 2014 | Small Business | Medium | 14 | High |
| ₹8,080 | 01-10-2014 00:00:00 | 10 | October | 2014 | Small Business | Large | 14 | High |
| ₹91,327.5 | 01-01-2014 00:00:00 | 1 | January | 2014 | Government | Medium | 14 | High |
| ₹49,159 | 01-04-2014 00:00:00 | 4 | April | 2014 | Government | Medium | 14 | High |
| ₹8,200 | 01-05-2014 00:00:00 | 5 | May | 2014 | Government | Medium | 14 | High |
| ₹395.76 | 01-09-2014 00:00:00 | 9 | September | 2014 | Government | Medium | 14 | High |
| ₹1,761.54 | 01-10-2013 00:00:00 | 10 | October | 2013 | Government | Medium | 14 | High |
| ₹1,872 | 01-02-2014 00:00:00 | 2 | February | 2014 | Government | Medium | 14 | High |
| ₹1,765.62 | 01-10-2014 00:00:00 | 10 | October | 2014 | Government | Medium | 14 | High |
| ₹28,700 | 01-11-2014 00:00:00 | 11 | November | 2014 | Government | Medium | 14 | High |
| ₹48,257 | 01-11-2014 00:00:00 | 11 | November | 2014 | Government | Large | 14 | High |
| ₹78,802 | 01-11-2013 00:00:00 | 11 | November | 2013 | Government | Medium | 14 | High |
| ₹4,363.2 | 01-04-2014 00:00:00 | 4 | April | 2014 | Government | Large | 14 | High |
| ₹19,680 | 01-07-2014 00:00:00 | 7 | July | 2014 | Small Business | Large | 14 | High |

Figure Creating a calculated column to compute the discount bucket

Once this column is created, it can be used in a report slicer to filter other visuals on a report. However, this formula can be improved and can be made to work more efficiently using a DAX which can be used in a DAX formula using the VAR keyword.

VAR: This stores the result of an expression in a named variable, which can be passed as an argument to other expressions, using the RETURN keyword.

Syntax: VAR =

Parameters:

The name of the variable.

The expression, value of which would be stored in the variable.

Instead of storing the discount percentage values in a separate column, we can store them in a variable on a row-by-row basis, which would hold the percentage values on runtime and the calculated column can use that variable value to compute the discount bucket for each row. Figure 3.23 shows the formula to create the same discount bucket calculated column, but this time, in one step using a variable:

```
1 DiscBucket2 =
2 VAR DiscPercent =
3     DIVIDE ( financials[Discounts], financials[Gross Sales] ) * 100
4 RETURN
5     IF ( DiscPercent <= 5, "low", IF ( DiscPercent <= 10, "Medium", "High" ) )
```

Figure 3.23: Creating a calculated column to compute the discount bucket using a variable

As seen in the preceding figure, here we are holding the value of discount percentage in a variable named DiscPercent and later passing the value to the IF function using the RETURN keyword, to compute the discount bucket, all in a single formula.

By now, the use of calculated columns should be clear enough to the readers. We would anyway go through a lot more examples and create some fairly complicated calculated columns in due course of time.

Tip: DAX code snippets can be formatted to make them more readable and cleaner, using a free website called DAX Formatter

## Measures

As we have seen, calculated columns are to be used only when we want to evaluate the expression on a row-by-row basis, however, that is not what we always need.

Measures aggregate values of columns and tables, and return a single scalar value, working in a filter Filter context means all the filters that would be considered to evaluate the value of an expression. Let us refer to a Matrix visual of Power BI, which can be used to create a pivot table, to understand the concept further. We would not go into detail about how to create visuals at this stage, as that will be discussed in the upcoming chapter of the book where we will specifically cover reporting and visualization. Refer to Figure 3.24 which shows the pivot table:

| Product | January | February | March | April | May | June | July | August | September | October | November | December |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Amarilla | $20,506.2 | $6,23,275 | $1,39,179 | | $16,312 | $1,26,849.03 | $10,117.89 | $40,020 | $6,09,199.5 | $6,36,411.19 | $57,172.8 | $3,36,540 |
| Carretera | $58,392.4 | | $3,81,527 | $42,820.8 | $39,864.6 | $1,43,616 | $60,198 | | $19,043.2 | $3,21,529.14 | $9,410.02 | $6,70,604.92 |
| Montana | $2,74,874 | $7,220.82 | | $3,37,205.5 | $3,33,616 | $35,706 | $35,584.9 | $1,62,685.9 | $3,64,726.62 | $2,52,901.71 | | $64,993.07 |
| Paseo | $3,91,609.335 | $56,101.6 | $57,949.8 | $1,26,688.945 | $4,03,172.83 | $5,82,095 | $3,70,581.75 | $26,655.28 | $1,67,870.22 | $2,32,426.47 | $10,29,891.5 | $8,43,557.47 |
| Velo | $2,986 | | | $70,425.1 | $698.66 | $3,67,463.59 | $3,33,606 | $53,745.2 | $52,993.4 | $10,60,602.7 | $85,740.28 | $1,51,293.62 |
| VTT | | $3,85,285 | $9,699.56 | $1,96,294.5 | | $5,733.56 | $9,725.1 | $4,31,872 | $4,54,815 | $7,31,497.55 | $86,825.8 | $5,11,055.64 |
| **Total** | **$7,48,367.935** | **$10,71,882.42** | **$5,89,355.36** | **$7,81,434.845** | **$7,93,664.09** | **$12,61,463.18** | **$8,19,813.64** | **$7,14,978.38** | **$16,68,647.94** | **$32,35,648.76** | **$12,69,040.4** | **$25,78,044.72** |

Figure Pivot table

The preceding pivot table displays the profit earned by month, for different products. Here, for instance, the context of the selected cell value ($7,220.82) would be Product = Montana and Month = as those are the filters based on which Power BI calculated that specific value and hence can be thought of as the context for that value. In this case, each cell of the

table would have a different filter context and hence contains different values. Apart from the visual itself, filters can be applied on various layers like report level filters, page level filters, and so on. and all of those would be considered as the filter context for a visual.

Now that we have understood what a filter context is, let us try to understand by another example why we might need a filter context and hence a measure, instead of a calculated column in the first place. Here we have a table having Products, Sales value, Discounts offered and Regions where the products are sold, as shown in Figure

| Region | Product | Sales Value | Discount |
|---|---|---|---|
| North | A | 1900 | 190 |
| North | B | 2100 | 168 |
| North | C | 700 | 35 |
| North | D | 1200 | 144 |

Figure 3.25: Sample data for North region

Let us consider the requirement is to display the average discount percentage value on the report, for each region. In this case, if we calculate it on a row context (row by row basis) using a calculated column, we need to create a new column Discount% first, as shown in Figure 3.26 and then use that column in the visual to show the aggregated value:

| Region | Product | Sales Value | Discount | Discount% |
|---|---|---|---|---|
| North | A | 1900 | 190 | 10 |
| North | B | 2100 | 168 | 8 |
| North | C | 700 | 35 | 5 |
| North | D | 1200 | 144 | 12 |

Figure Discount% calculated for each row

When we use this calculated column in a visual, we need to choose the aggregation method which should be Average in this case. So, the average Discount% would result in (10+8+5+12)/4 = for the North region. Now, this would not be accurate as what we needed is to calculate the average percentage for the North region, which should be (total discount offered/total sales)*100 for that region, across all products it has. Hence, the correct value should be (190+168+35+144)/(1900+2100+700+1200)*100 = So, we need to filter the data for each region first, and then for each subset of data need to calculate the percentage, then only it will produce correct results if we filter the value with the region in the report and hence, we need a measure here.

While in many cases either a measure or a calculated column can be used, there are important differences that we need to keep in mind while using one. Below are the most important ones:

Calculated columns are recalculated when the data model is refreshed as they are materialized in a data model, on the other hand, measures are recalculated at query refresh time which means, every time we interact with a visual.

Calculated columns can increase the data model refresh time while measures can increase the report or visual response time, hence we need to be mindful about when to use one and not the other.

Calculated columns are to be used when we want to calculate the formula on a row-by-row basis, but in case we want to calculate based on filter context, we should go for a measure.

Creating a measure is similar to how we create a calculated table or a calculated column. To enable the DAX formula editor for creating a measure, both the Report view and Data view can be used. From the Report view, Modeling | New measure should launch the editor, while from the Data view, Home | New measure can be followed.

As always, let us see an example of the use of DAX measures.

Creating a measure computing total sales for a country: Let us continue with the same financial data that we have referred to multiple times by now. The requirement is to display the Sales value for Germany on the report. While there are many ways to do it, as we are discussing measures, just to illustrate the idea, let us create a measure Sales_Germany for doing it using a DAX function called CALCULATE.

CALCULATE: This evaluates an expression in a context modified by filters.

Syntax: CALCULATE( [, 1> [, 2> [, …]]])

Parameters:

The expression to be evaluated (should return a single scalar value).

Expression that defines a filter (optional and repeatable).

For this instance, CALCULATE should evaluate the expression SUM(Sales) in the context of Country =

SUM: This adds all numbers in a column.

Syntax: SUM()

Parameter: Column containing values to sum up.

Figure 3.27 illustrates the DAX formula to create the measure

```
1 Sales_Germany =
2 CALCULATE ( SUM ( financials[ Sales] ), financials[Country] = "Germany" )
3
```

Figure 3.27: DAX formula for Sales_Germany

The measure outcome would be a single scalar value which unlike calculated columns, cannot be seen on the Data view. The value of the measure can only be seen on the report itself. However once created, the object would be visible on the right-hand side Fields pane, under the table selecting which the measure is created, with a distinct icon, as shown in Figure

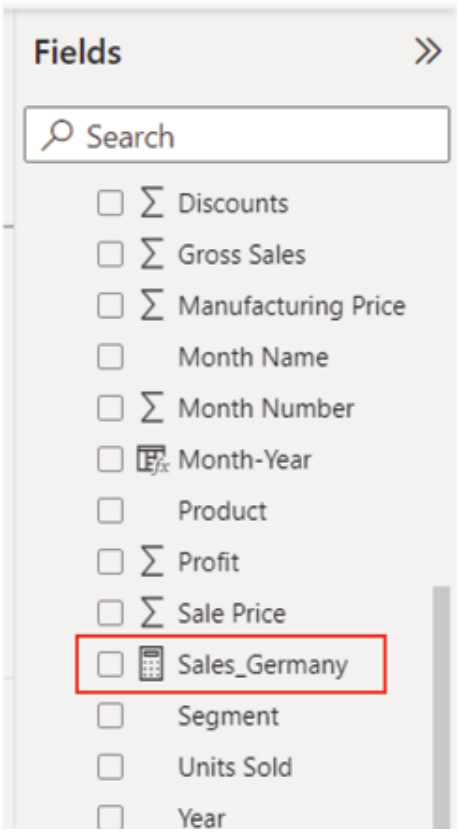



Figure 3.28: Measure icon

DAX measures are reusable and once created, can be referred from another measure, calculated columns and so on. For example, now that we have created the Sales_Germany measure, in case we need the Sales figure for Germany to use in another measure, we would not require to re-calculate it in that measure and instead, we can simply refer to Sales_Germany in the formula editor. To see all the measures that are available for the report, in the formula editor for measures, we just need to type opening square bracket ( [ ) and the AutoComplete should prompt the list of available measures from which the required one can be selected, as shown in Figure

Figure 3.29: Referring Sales_Germany from another measure

As already mentioned, measures by default reside in the table selecting which they are created. However, they can be moved from one table to another as they are not materialized in any table. Instead of saving measures in any of the fact or dimension tables in the model, a separate Measure Table can be created which can store all the measures in a report.

To create a measure table, first, a local table can be created manually in Power BI Desktop using the Home | Enter data option, either from the Data view or the Report view, as shown in Figure

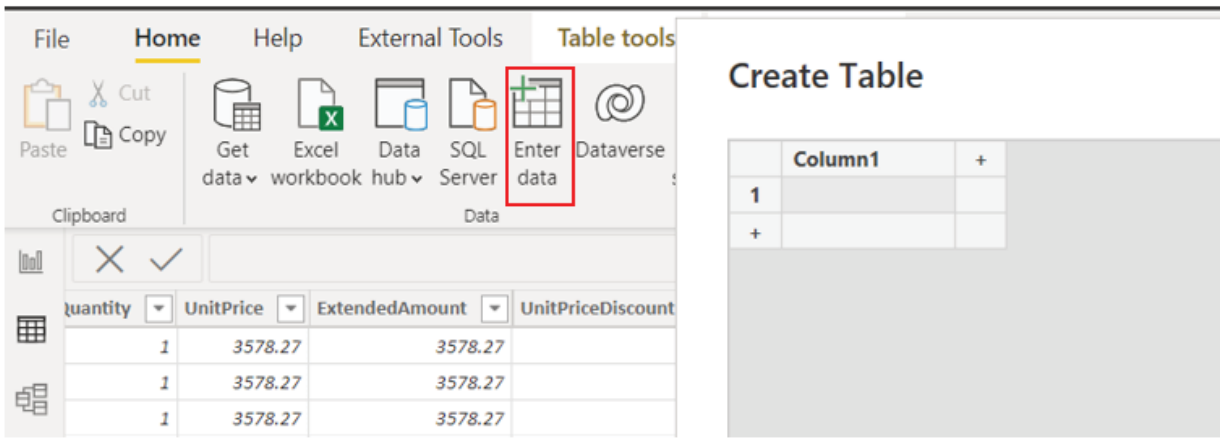



Figure 3.30: Creating a local table in Power BI Desktop manually

All we need is the table object, so the columns can be left as-is, only name of the table can be updated as desired. Let us name it as All_Measures for this instance. By default, the table is created with a single column named Column After that, the measures we want to move to this table need to be selected on the Fields pane, which should activate the Measure tools option on the top ribbon. The table under which any measure is saved is called the Home table for that measure. On Measure the Home table can be switched by selecting the required table from the dropdown.

For our purpose, all the measures should be moved to the All_Measures table following the process mentioned above. Once done, the default Column 1 needs to be removed from the All_Measures table by selecting the field on Fields pane and using the Delete from model option from the right hand side ellipsis of the selected field.

Now, the All_Measures table should contain only the measures from the report, which should qualify it for a Measure also the table icon should get updated reflecting the change. Figure 3.31 should illustrate the entire process:

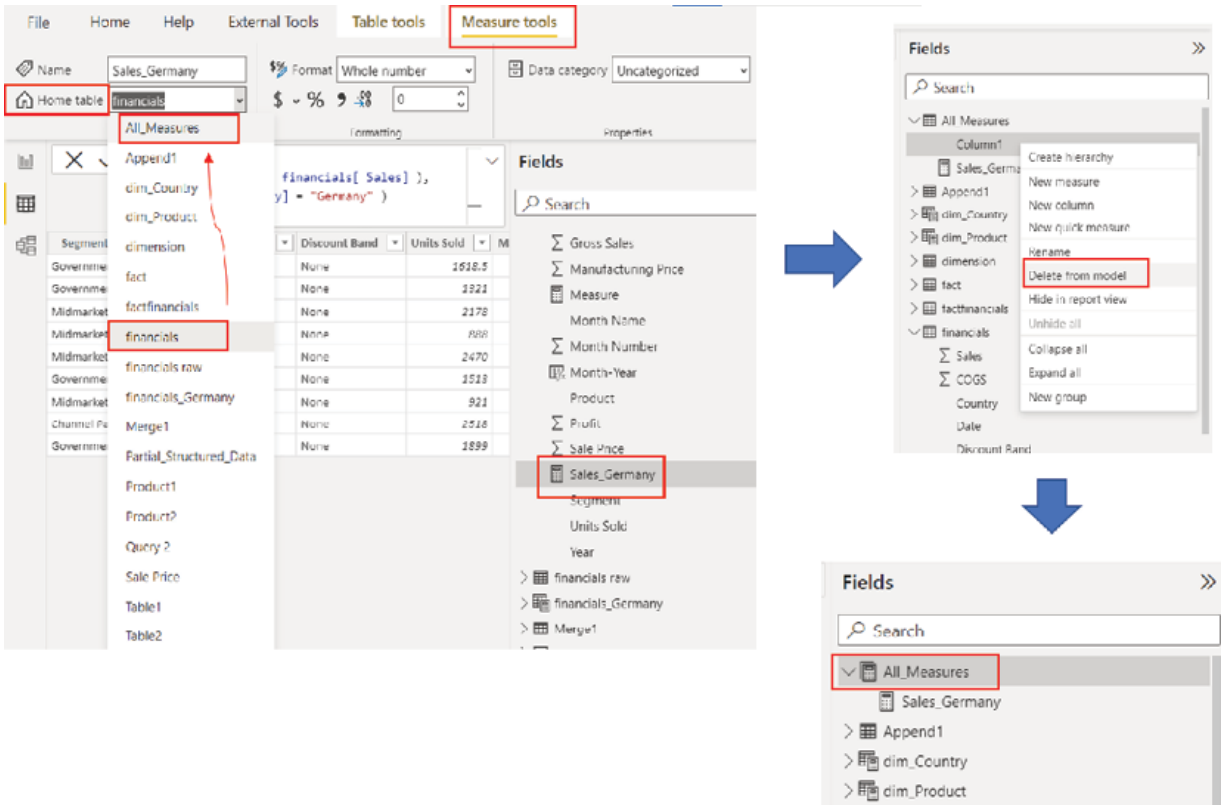



Figure 3.31: Creating a Measure Table in Power BI Desktop

Measure tables are very useful for organizing the measures if a report is going to have a lot of them. However, the number of measures used in a

report can also be reduced using a special table called Calculation which can be created using an external tool known as Tabular Once Tabular Editor is installed, it can be integrated into Power BI Desktop and can be used to view and edit objects in the model.

Calculation Groups can apply specific calculations on top of existing DAX measures and are specifically useful to create different variations of a base measure using a placeholder for that measure. DAX time intelligence functions are good examples to implement Calculation We will explore some Time Intelligence functions separately in the next chapter as well as in the DAX Reference Guide.

As we have seen, the value of measures can only be viewed on the report itself, we are going to explore more examples while we discuss the reports, however the concept of what measures are and how to use them should be clear by now for the readers.

## Optimizing models in Power BI

As we already have seen how to create a data model and enhance it further with the help of DAX, let us see how a data model can be optimized for best performance following some common best practices for imported tables:

Remove unnecessary Data models are recommended to be designed with only the columns required for reporting or to support model relationships, security columns can significantly increase the refresh time of the model causing performance issues.

Limit number of Along with columns, care should be taken to avoid loading unnecessary rows as well to the This can be done by applying specific filters or parameterizing queries to bring only a subset of data relevant for reporting.

Optimize Data The data types of columns can play an important role in the performance of a data model. The key columns of the tables which are used in relationships should ideally have numeric data type (which provides the highest optimization) instead of text. This can result in significant performance improvement, especially for cases where the columns contain unique values and have high cardinality.

Disable auto For date fields, Power BI by default includes an Auto date/time option which generates a hidden auto date/timetable to support grouping, drill-down through date hierarchies etcetera. Disabling this

option can significantly reduce the model size, and hence should be considered in case the feature is not required. The option to control this can be found on Power BI Desktop following File | Options and settings | Options | Data as shown in Figure

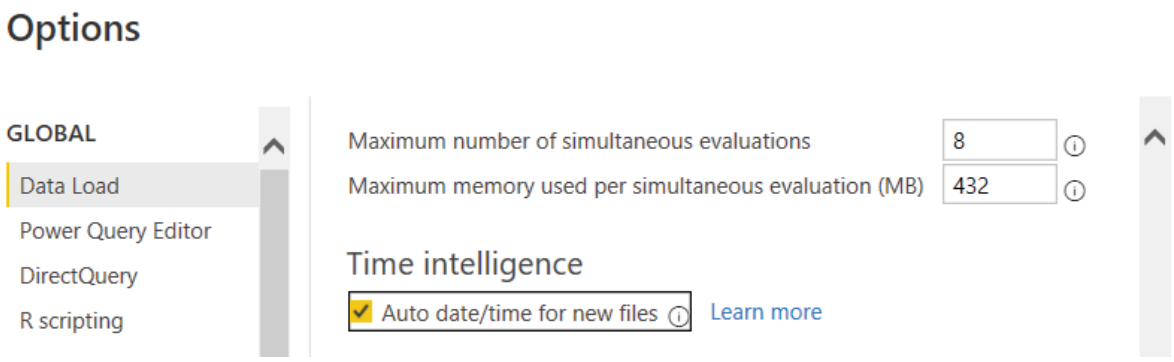



Figure Controlling Auto date/time option

Avoid bi-directional As we already have touched upon this one while discussing relationships, use of filtering should be minimized in time of designing a data model as they can negatively impact model queries. Also, as both tables involved get filtered by one another, it can potentially create ambiguity in end-user experience.

Apart from the above common best practices, there are various ways to monitor and optimize a data model further using external tools like Tabular Editor and DAX We will discuss more about these tools in upcoming chapters, however, let us have a quick look at them now just to be aware of the ecosystem.

Both Tabular Editor 2 and DAX Studio are free desktop applications and do not have any licensing requirements for installation. Once installed, these tools could be launched from the External Tools ribbon on Power BI Desktop, as shown in Figure

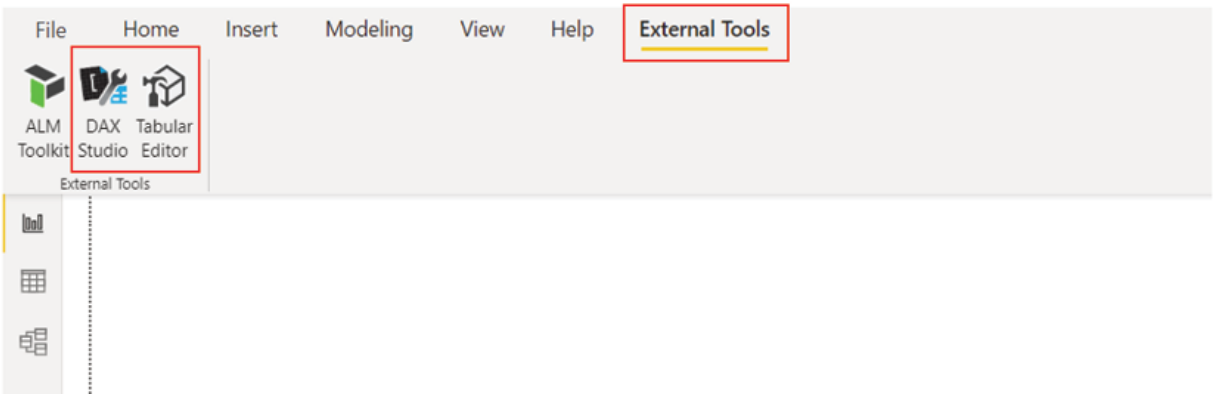



Figure 3.33: External tools in Power BI Desktop

Launching these tools from Power BI Desktop enables them to interact and edit the underlying data model objects, providing users a seamless integrated experience.

Monitoring data model using DAX Once DAX Studio is launched, all the objects of the underlying data model would be visible on the left-hand panel. The VertiPaq Analyzer can be accessed from the Advanced option on the top ribbon. VertiPaq Analyzer is useful to analyze the storage structure of a data model in Power BI. A number of entities and attributes of a data model like Table Column Hierarchy Size, and so on. can be analyzed using the View Metrics option, to understand the contributing factors for the underperformance of a specific data model and take corrective actions. Figure 3.34 below shows the VertiPaq Analyzer Metrics available for a Power BI data model:

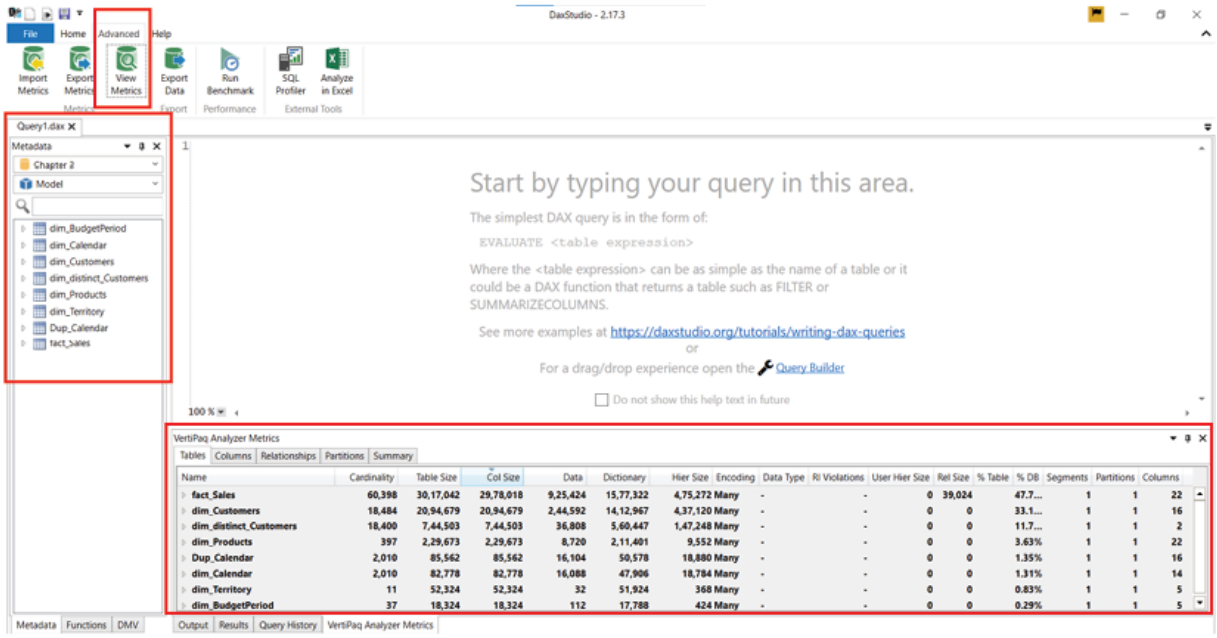



Figure VertiPaq Analyzer Metrics in DAX Studio

Optimization or performance tuning of a data model is a broad topic and many more can be explored and done regarding this, however the topics we have discussed should provide a reasonably good starting point for the intended users.

## Conclusion

In this chapter, we have seen how we can build a data model in Power BI to tie up all those individual tables together so that they can work as a single unit. We have emphasized a star schema data model and understood the relevance of it in the context of Power BI. We have seen how relationships can be used in a data model and explored important properties of a relationship like cardinality, cross-filter direction etcetera . The readers have been introduced to DAX and different ways of enhancing a data model by means of Calculated Tables, Calculated Columns and Measures are explained. Finally, we have seen few common best practices to optimize a data model for performance. Data modeling is a vast topic in itself and probably we have touched just the tip of the iceberg! However, this should enable readers to try out things hands-on and learn further along the way.

In the next chapter, we will learn about visualization and see how to create a nice report on top of the data model that we have created.

## Knowledge check

If in a Power BI data model, the cross-filter direction between two tables (having a one-to-many relationship) is that means:

The table on the one side of the relationship should filter the table on the many side of the relationship.

The table on the many side of the relationship should filter the table on the one side of the relationship.

Both the tables would filter each other.

Bi-directional filtering is only for many-to-many relationships.

Which of the following statement is not true for a calculated column?

Calculated columns work in row context.

Calculated columns are materialized in the data model.

Calculated columns are refreshed while users interact with the report.

Calculated columns can be referred from a measure.

To calculate an expression in the context of applied filters, what should be used?

Measure

Calculated Column

Calculated Table

Filter Parameter

Which external tool can be used to view the VertPaq Analyzer Metrics of a data model?

SQL Server Management Studio (SSMS)

Tabular Editor

ALM Toolkit

DAX Studio

All Knowledge Check answers are provided at the end of the book.

**Join our book's Discord space**

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

# CHAPTER 4

# Visualizing Data in Power BI

## Introduction

We have advanced a lot in our journey of learning how to create an end-to-end Power BI reporting solution. We have seen how to integrate Power BI with different data sources and then shape or transform the data as required. We have explored how to create a data model in Power BI which should support the report we want to build. Now, it is time to leverage all our work so far and create a visually appealing report that can provide quick and actionable insights to the report consumers. This chapter will focus on how to create visualizations in Power BI. Commonly used visuals will be discussed in detail along with different formatting options, to impart the knowledge of how to create a positive user experience. We will discuss the default visuals as well as the custom visuals from the marketplace, also known as AppSource.

## Structure

In this chapter, we will discuss the following topics:

Creating a report template

Creating the first visual in Power BI

Bar charts and column charts

Slicers

Trend analysis

Visualizing geographical data using Maps

Use of cards

Tables

Matrix

Introduction to custom visuals

Bookmarks pane and Selection pane

Report tooltips

Visual interactions in Power BI

Analyzing report performance

## Objectives

The objective of this chapter is to impart relevant knowledge to the readers regarding visualization in Power BI. Look and feel is an important aspect of any report for providing a positive experience to the report consumers. Report authors should be aware of which visual to use for providing specific insights, as well as how to design a report so that it can tell a story about the underlying data it is built on. Also, the chapter should help the readers for making optimum use of the space or real estate available for any report. Along the way, we would create a functional report using all the visuals discussed in the chapter.

## Creating a report template

We already know what the home screen looks like when someone opens the Power BI Desktop. We can readily start creating reports on the home page; however, a template helps to create a nice look and feel as well as minimizes formatting efforts.

Let us consider we have created a simple image using PowerPoint with the desired background color and a logo which we want to use as a canvas background on the Power BI report. The option to import the image can be found under the Visualizations pane following the Format page | Canvas background. The image needs to be browsed and opened, and then a few properties like Image fit and Transparency may need to be configured as shown in Figure 4.1 to apply it to the page:

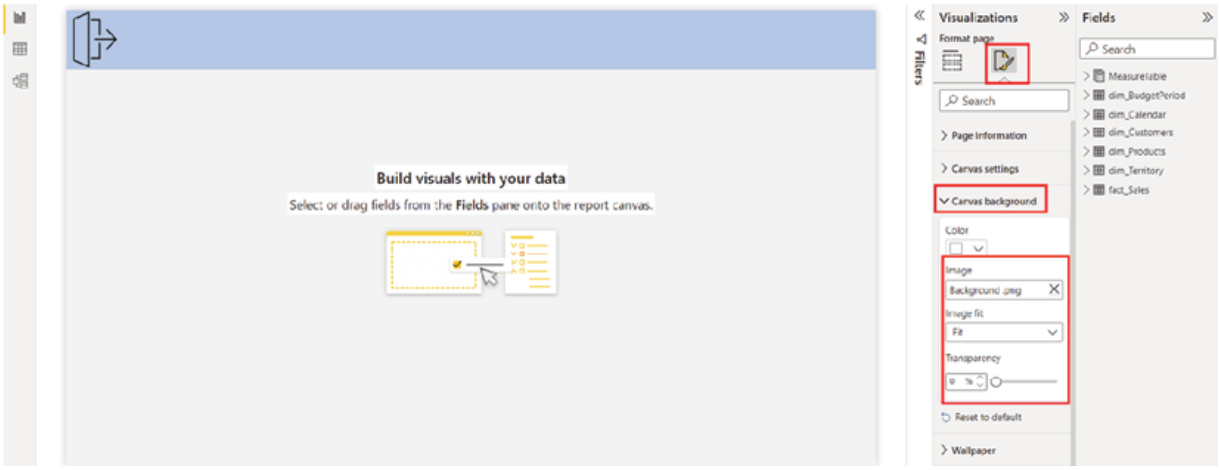



Figure 4.1: Applying an image as a canvas background

The theme or color scheme of a Power BI report can be configured using View | Themes option. The option allows you to view the theme that has been presently applied to the report, along with all the available default themes. The Customize current theme option allows you to modify each color component used in a default theme, apart from that a custom theme can also be generated in the JSON format and imported using the Browse for themes option. Figure 4.2 illustrates all the useful options regarding themes in the Power BI Desktop:

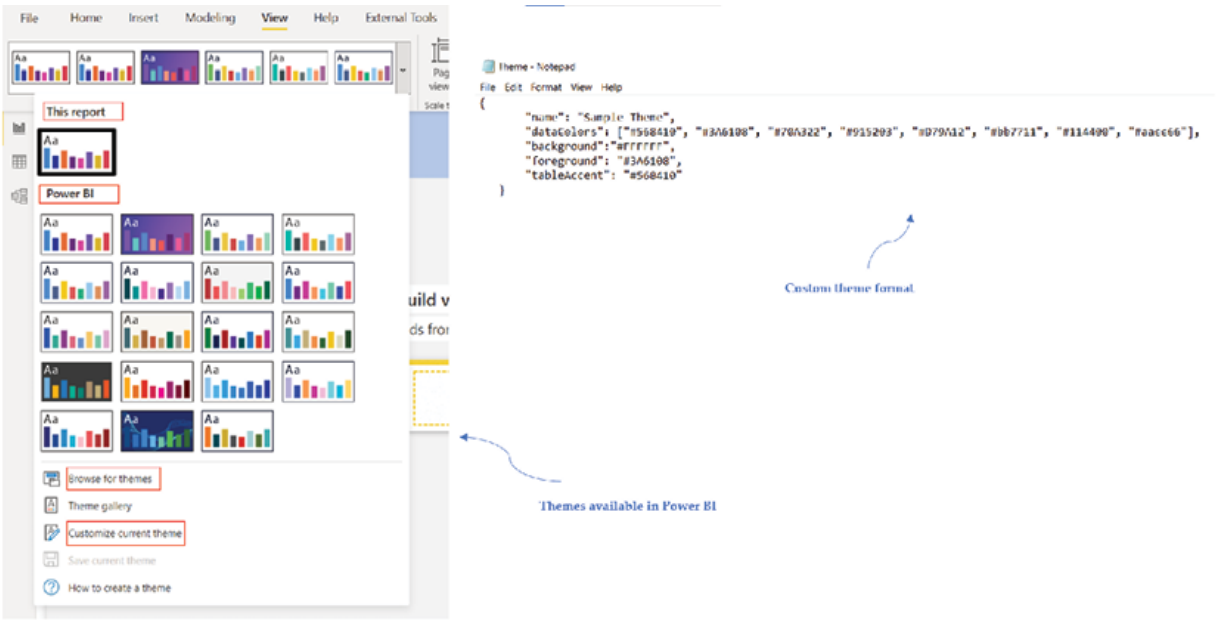



Figure 4.2: Themes in Power BI Desktop

In the case of importing a custom theme, for dataColors, the hexadecimal color codes need to be used while for foreground, and color classes can be used. The list for dataColors can consist of as many color codes as required.

Once a theme is applied, the colors would stick to the visuals used in a report and we would not require to manually update the color every time. Without a theme, though it takes a little longer to format visuals individually, it might provide more control and scope for creativity.

## Creating the first visual on Power BI

To create a visual on Power BI, the required visual icon under the Visualizations pane needs to be selected which will create the visual container on the report canvas. This should also enable the field wells corresponding to the visual where data can be added from the Fields pane to populate the visual, as shown in Figure

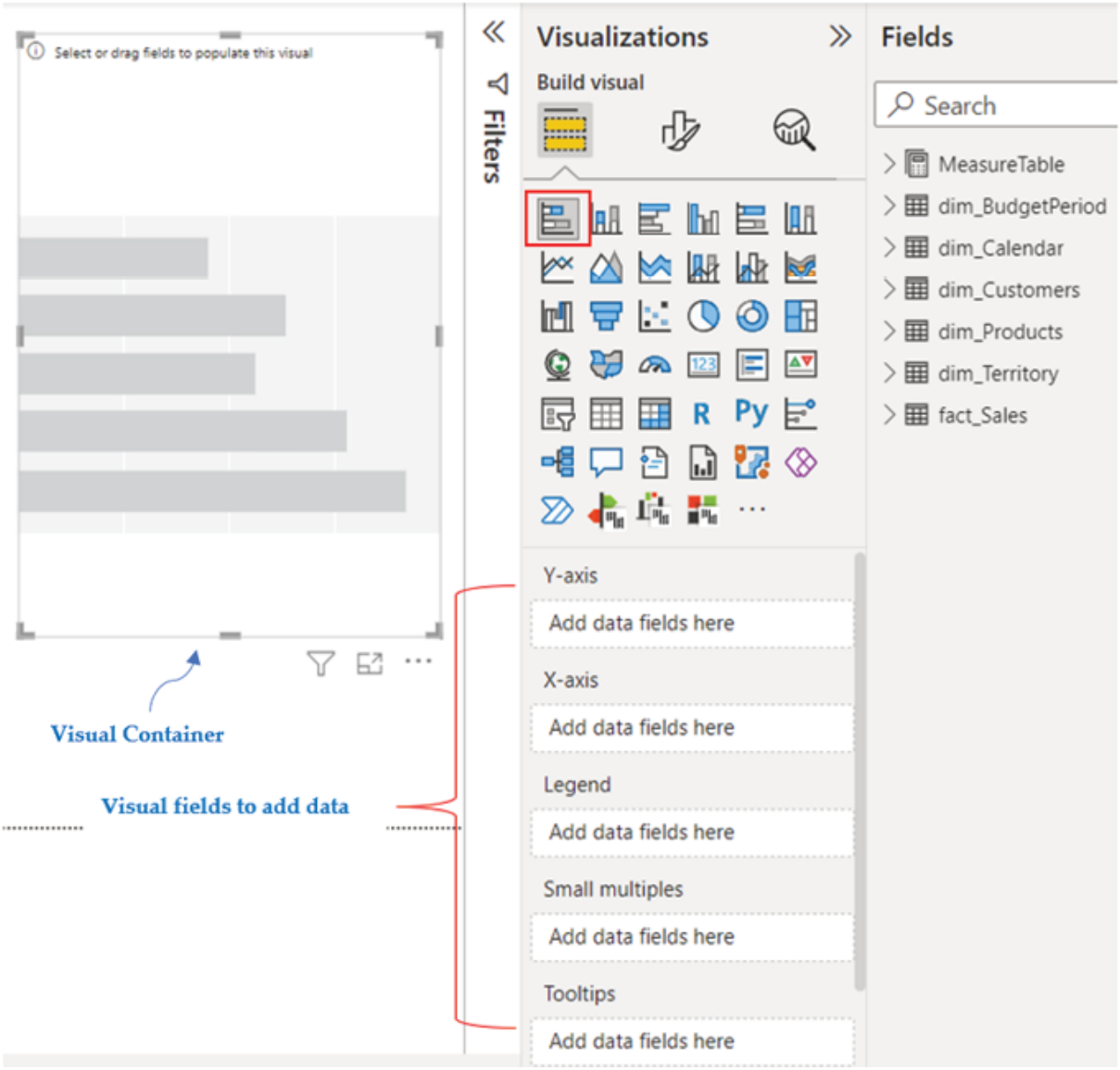


Figure 4.3: Creating a visual container on Power BI

After data is added to the field, the visual should populate in the visual container. There is a wide range of formatting options available for each visual, which can be explored on the Format your visual option under the same Visualizations pane. The formatting options are categorized as Visual and The visual-specific settings have options for the visual type which is currently selected while the general settings have options like visual size, title, effects, and so on, which are consistent across all visual types. Figure 4.4 illustrates how to access different formatting options for a visual, using a Stacked bar chart as an example:

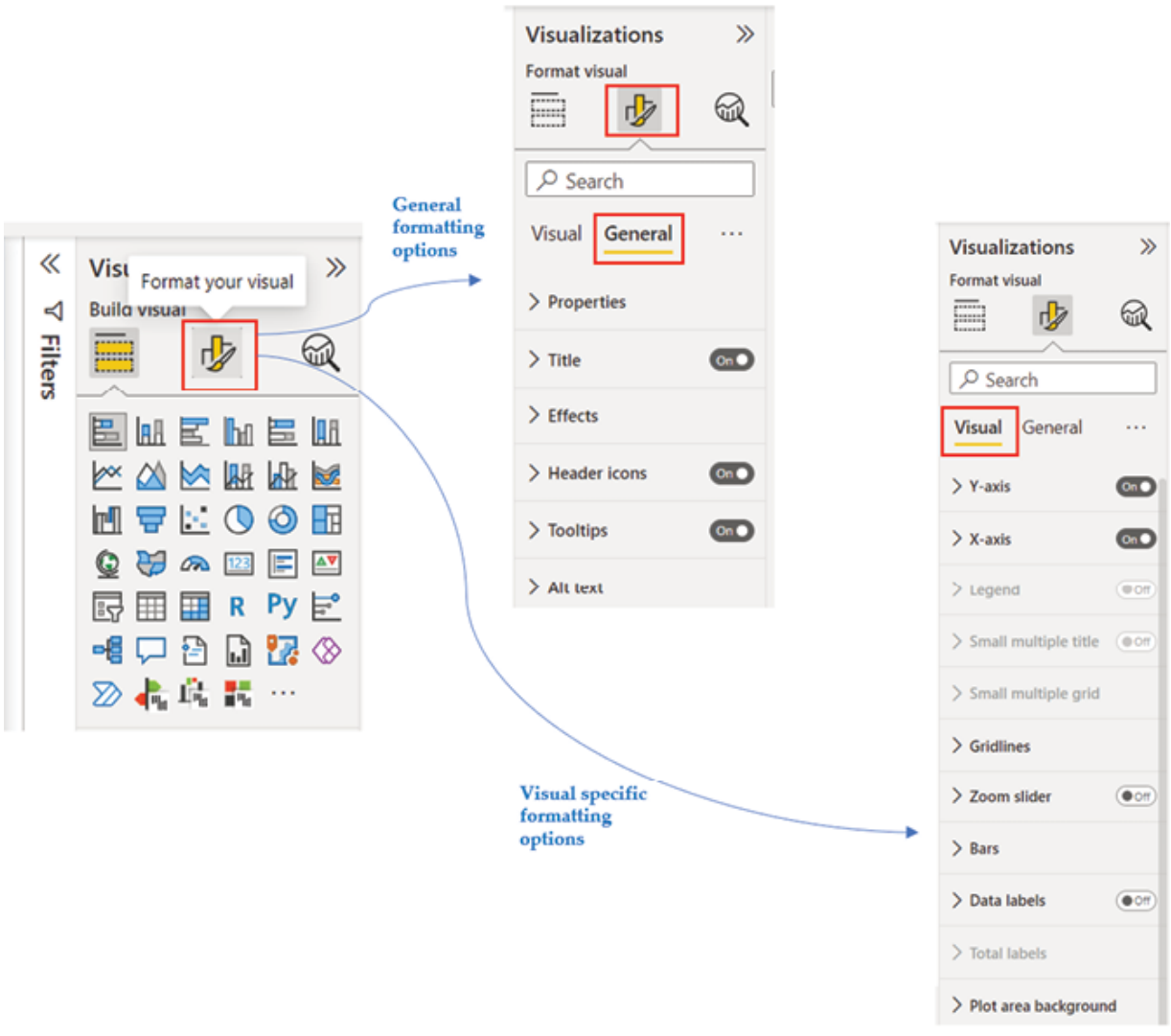


Figure Formatting options for a Power BI visual

Although the visual-specific options are different based on visual types, however, most of them should have a few common options to control color, labels, and so on. Let us now explore a few interesting visuals which are available by default in Power BI.

## Bar charts and column charts

Bar and column charts are two of the most popular options when comparing values by attributes like categories, descriptions, and so on. Power BI provides multiple variations of bar and column charts, as part of the default visuals that come out of the box, as shown in Figure

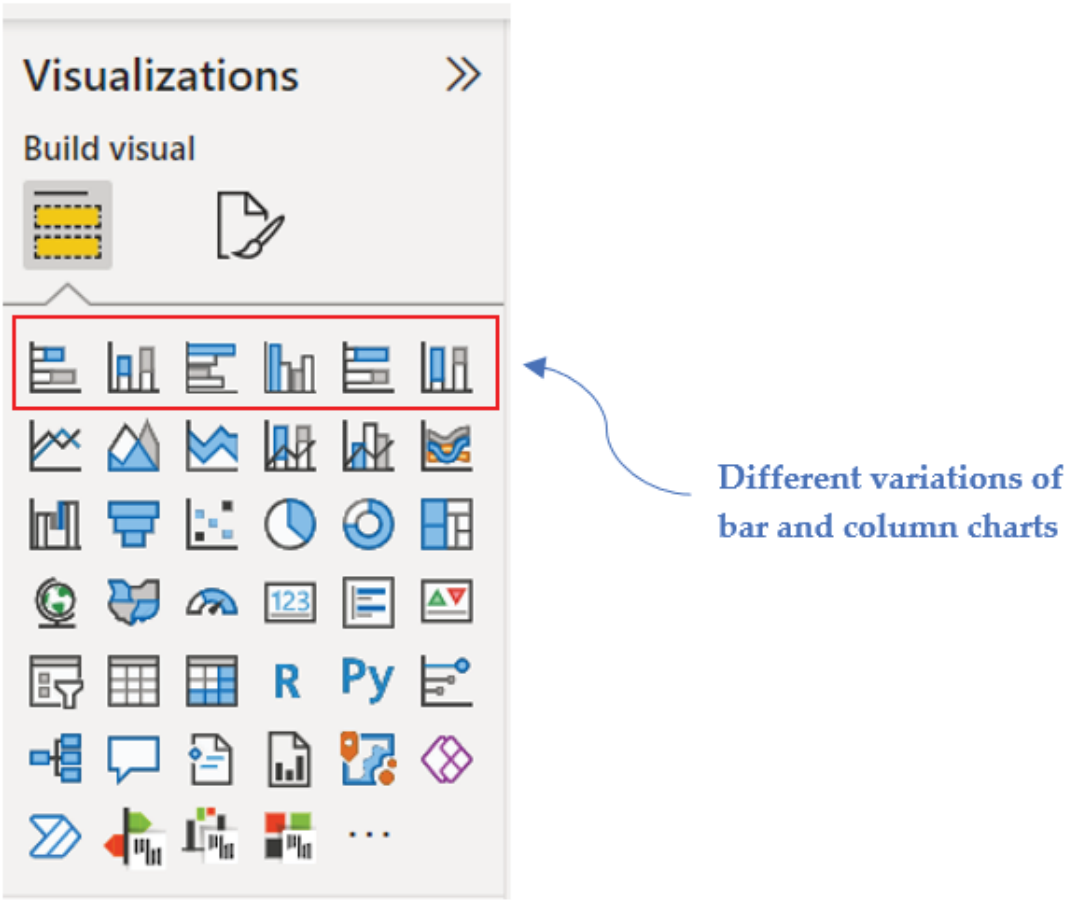



Figure Bar and Column chart default options in Power BI

The variations are as follows:

Stacked bar chart

Stacked column chart

Clustered bar chart

Clustered column chart

100% Stacked bar chart

100% Stacked column chart

Figure 4.6 shows how each of these variations looks like while plotting a value Amount) against different categories, across multiple groups or regions which are represented as visual legends:


Figure 4.6: Variations of different bar and column charts

As seen in the preceding figure, the values are stacked over one another in a stacked chart while in a clustered chart, data values are displayed side by side for each legend. However, in both cases, the bar or column lengths are proportional to the data values. In a 100% stacked chart, the bar or column lengths are of relative percentages while the total length always corresponds to 100%.

Let us now start creating a report using the template created as per Figure based on a star schema data model that we already have seen in the previous chapter (refer to Figure Let us do a quick recap; the model has a fact table which stores the sales transaction data, along with five dimension tables namely dim_BudgetPeriod and All dimension tables are related to the fact table with one-to-many relationships, as illustrated in Figure

To create a Sales Report, we have multiple numeric columns in the fact table like OrderQuantity, and so on, which can be analyzed by the attributes present in the dimension tables. Let us first try to see the sales performance across different product categories using a bar chart. To do this, we can select the Stacked bar chart from the Visualizations pane, add SalesAmount from fact_Sales to the X axis and Category from dim_Products to the Y axis of the visual container, to populate the bar chart as shown in Figure

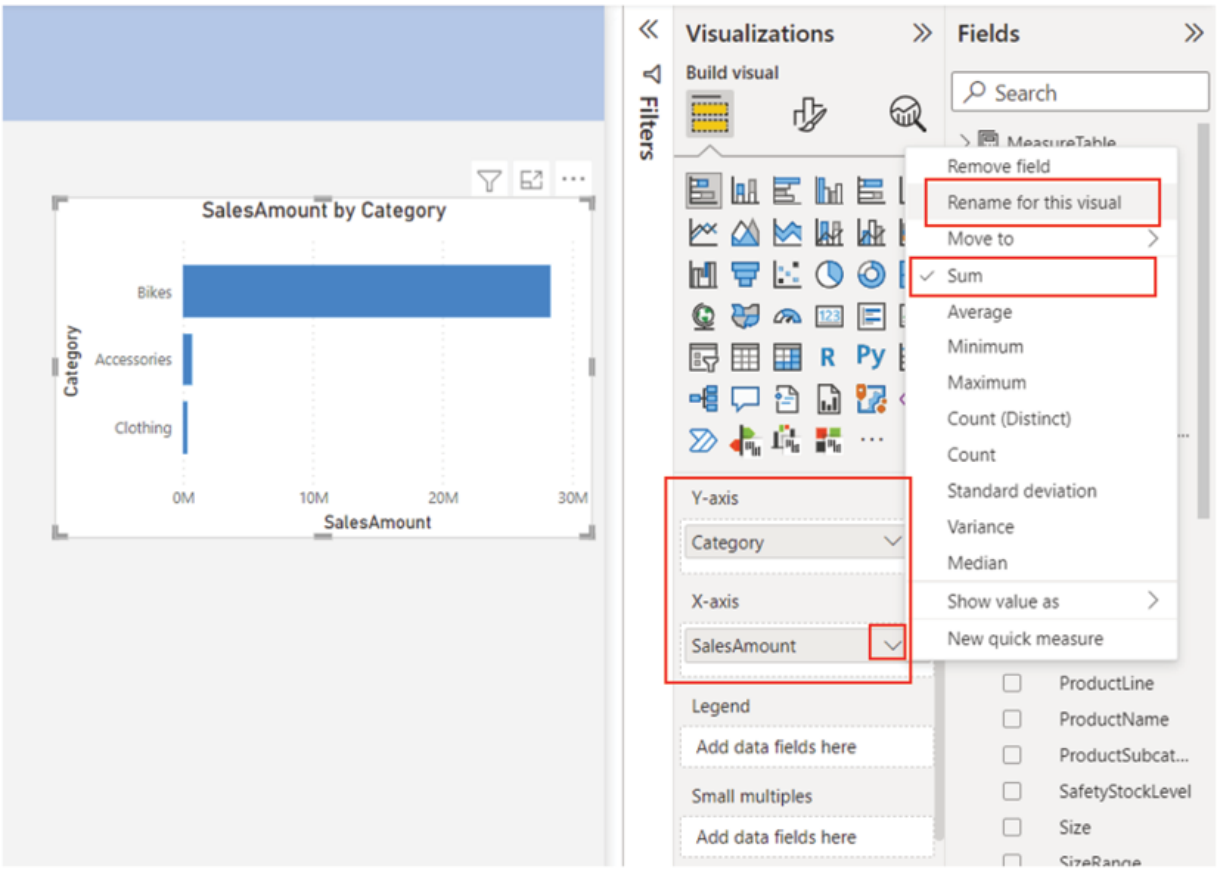



Figure 4.7: Bar chart showing SalesAmount by different product categories

Clicking on the drop-down button on the SalesAmount field should populate the options menu for that field; for this instance, the aggregation logic is Sum which gets applied by default and can be changed to any other available options by simply selecting the option as required. Also, the menu can be used to rename a field, remove a field etcetera.

As we have a nice star schema as a data model behind, it allows us to filter the sum of sales amount from the fact table by the dimension table attributes (like product category as in this bar chart), and all the tables work as a single unit as expected. Whenever a visual is created, it triggers a query to the underlying data to render the values.

As sales performance can be analyzed for different product categories using the bar chart we have just created, let us create another visual to view the top ten performing products based on the number of orders placed. To do this, we can select the Stacked column chart from the Visualizations pane, add OrderQuantity from fact_Sales to the X axis and ProductName from dim_Products to the Y axis of the visual container, to render the visual. To improve it further, we can add Category from dim_Products as Legend so that the visual can be displayed across product categories. Figure 4.8 shows the column chart displaying the number of orders placed for different products and categories:

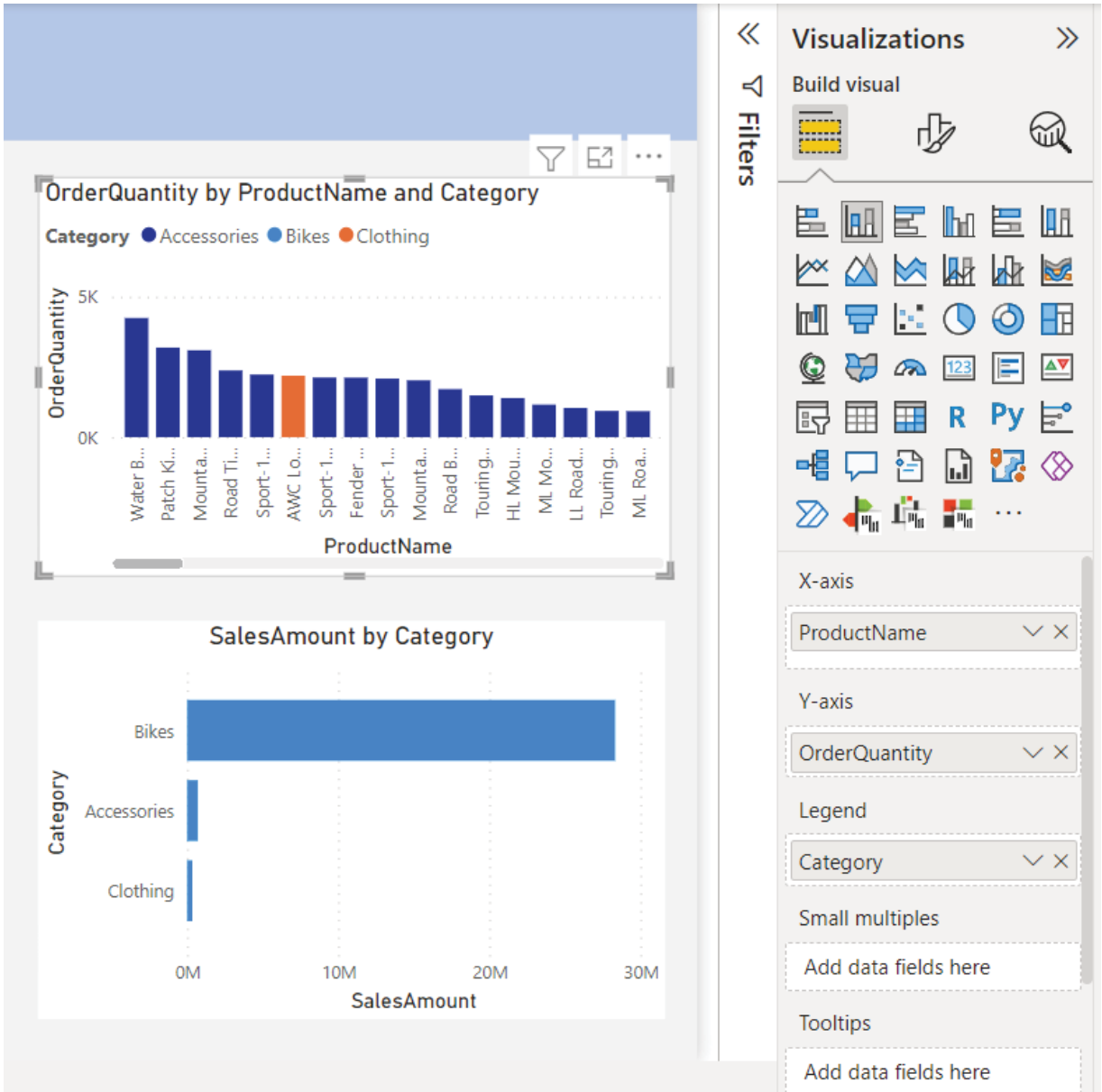


Figure 4.8: Column chart showing no. of orders by product names and categories

At this point, the column chart shows all the products that are there in the dimension table. Now all that remains is to figure out the top ten products which have been ordered the most. To do this, we can make use of the Filters pane, located between the report canvas and the Visualizations pane.

The Filters pane enables report authors to apply filters either on the selected visual, on the active report page, or all pages of the report. The corresponding options are Filters on this Filters on this page and Filters on all These options are also commonly known as Visual level Page level filters and Report level filters respectively. Whenever a visual is created, all the fields used in the visual are automatically added to the visual level filter, with a default selection as

For our purpose, we need to apply a visual level filter on the column chart which should only select the top 10 products based on the number of orders being placed. To do this, the ProductName field on the visual level filter needs to be expanded, the Filter type should be selected as Top the Show items field should be Top 10 and in the By value field, OrderQuantity needs to be added or dragged from the fact_Sales table. After applying the filter, the visual should show only the top 10 products as expected. All these configurations are one-time activities; each time the data is refreshed, the top 10 products would get dynamically populated in the visual based on the order quantity values we have in the fact table. Figure 4.9 illustrates the use of the Filters pane as discussed:

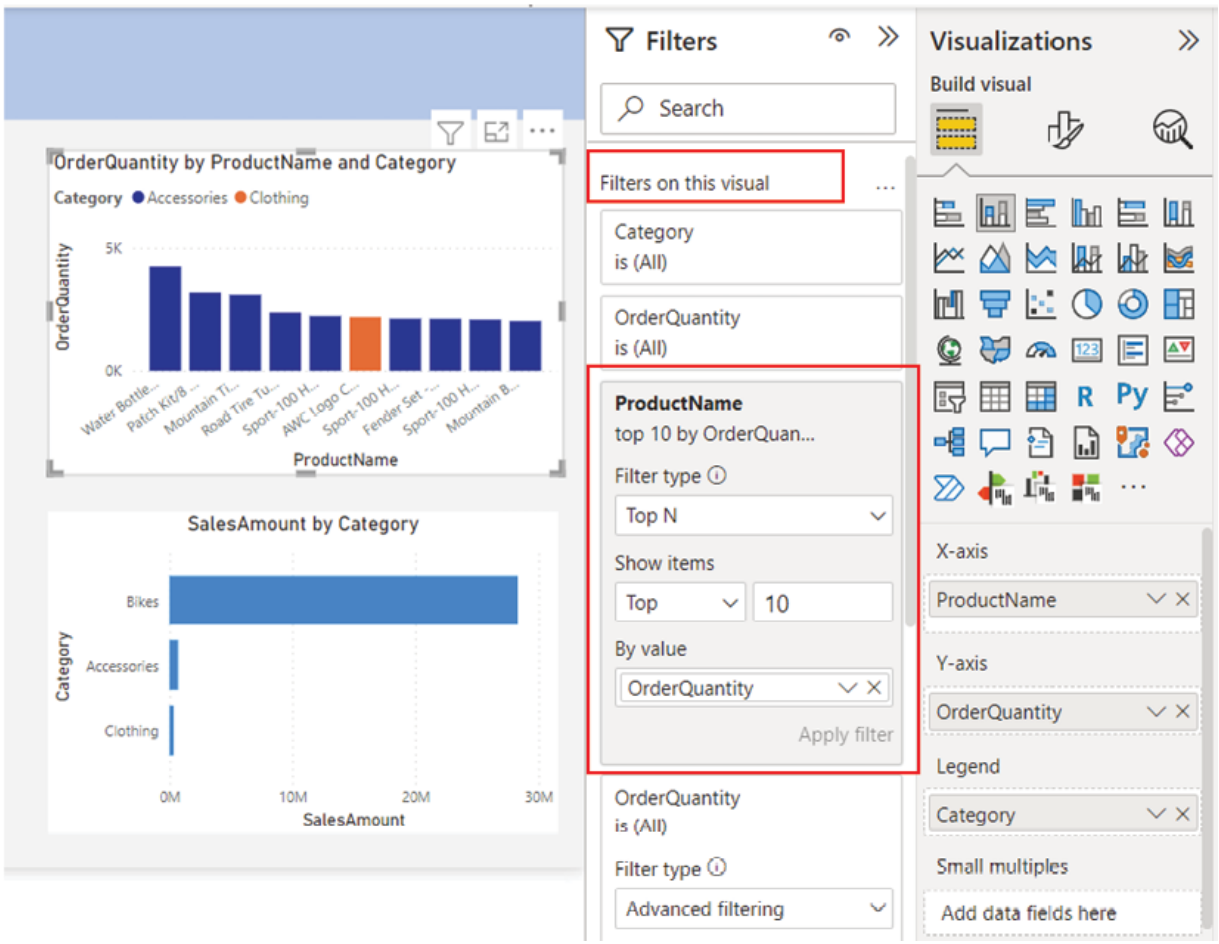



Figure Top 10 products by order quantity

The visual title can be updated from Format visual | General | Title while the color of each column can be updated from Format visual | Visual | Columns |

On hovering any visual, the ellipsis icon on the top right-hand corner opens More options which enable report authors to take a few additional actions like exporting data from the visual, sorting axis by ascending or descending order, and so on. On the left side adjacent to More there is the Focus which as the name suggests enlarges the visual and fits it to the entire page so that viewers can have a closer look. Again, on the left adjacent to the Focus there is the filter icon which can be referred to view the list of all the filters that are applied to the visual. Figure 4.10 helps to find all these options for the column chart we have just created:


Figure 4.10: Visual options

Once a visual is created, it can be dragged anywhere in the report canvas as required. Let us place these bar and column charts towards the left of the canvas, after leaving some space from the top ribbon, where the report slicers can be placed, which, we will discuss next.

## Slicers

Slicers are a way of filtering the visuals on a Power BI report, which are easier to access as they can be placed on the report canvas itself. Slicers have different settings and orientations which can provide a rich experience to the end users, besides enabling them to slice and dice the report in a much more convenient way instead of using the default filter pane.

As in our report, now we already have two visuals for analysing sales performance by different categories and top-performing products, let us enhance the report by introducing a couple of slicers for the report consumers. It would be nice to be able to slice the values over a time component, hence slicers for Year and Month should make perfect sense. To create a slicer for a year, we can select the Slicer visual from the Visualizations pane and add CalendarYear from dim_Calendar to the container field, to render the visual. The data type of CalendarYear is Whole and Power BI creates a slider by default for numbers and dates in a slicer, for conveniently selecting a range, which can of course be disabled from visual formatting options.

The slicer settings for CalendarYear would have different options like dropdown etcetera, while list is the default option. Each of these options represents different styles which are illustrated in Figure

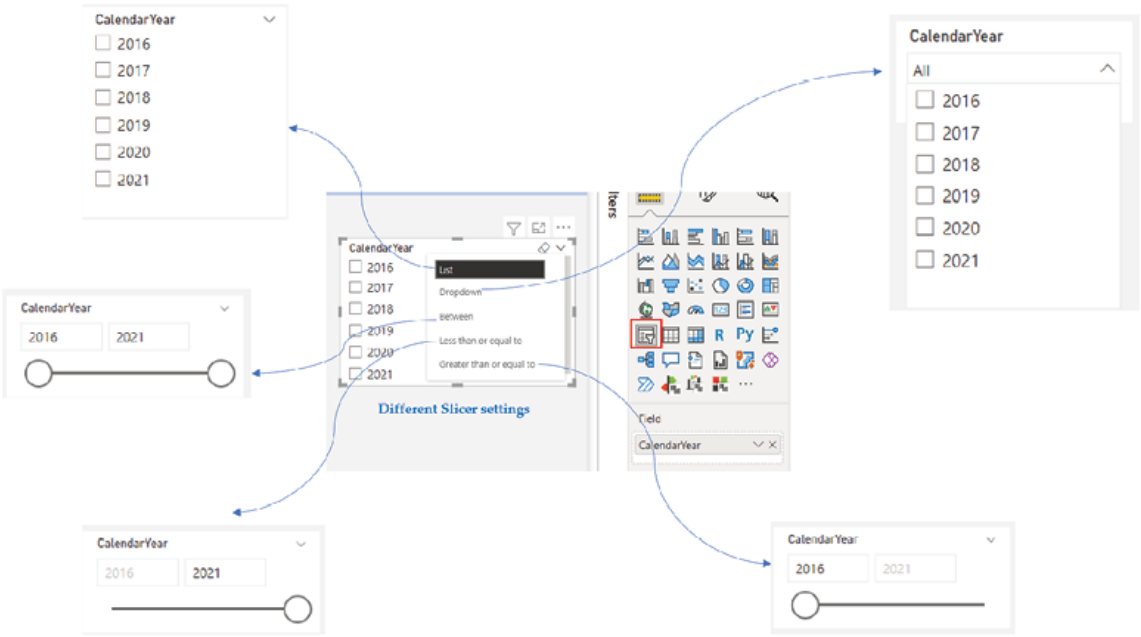



Figure 4.11: Different slicer styles

Apart from the styles, the slicers can also be oriented either vertically or horizontally as required using the Orientation option. For controlling the behavior while selecting items in a slicer, the Single select or Multi-select options can be used. All these options can be seen in Figure

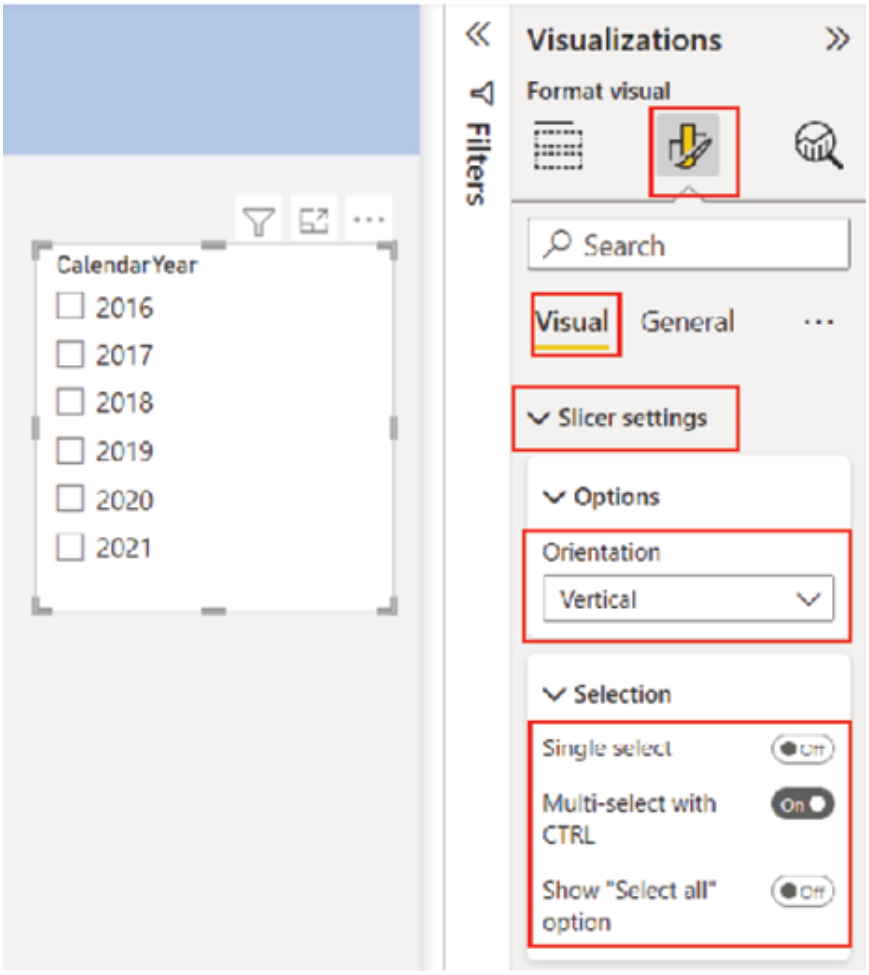



Figure 4.12: Key slicer formatting options

For our report, let us use the drop-down option for the CalendarYear slicer and place it on the top left, above the bar chart. Similarly, a second slicer for Month can be created, again using the dim_Calendar table and placed adjacent to the first slicer. At this point, the report should look like as shown in Figure


Figure 4.13: Sales report draft 1

If we remember the relationships in our star schema model, the Date column of dim_Calendar is related to the OrderDate column of by a one-to-many relationship. Hence whenever a filter is applied on a slicer, the fact_Sales would get filtered by the corresponding values of and the aggregation of SalesAmount and OrderQuantity would be calculated based on the subset of data in

As the concept of slicers should be clear by now, let us explore a few other visuals which can add value to our report.

Tip: To create the report name as Sales Report, a textbox can be used. To insert a textbox, go to Insert | Text box.

## Trend analysis

For analyzing trends, or to see how a value changes over a period, there are few combo visuals in Power BI which works well. The present default options are Line and stacked column chart and Line and clustered column These charts are appropriate choices specifically for plotting two fields which are very different in terms of the range of values they have, such as dollar amounts and quantity, making them perfect choices for trend analysis.

For the report we have been building, it would be interesting to see how sales amount and order quantity change over time and find out the trend. To do this, we can select the Line and clustered column chart from the Visualizations pane, add MonthName from dim_Calendar to the X axis and SalesAmount from fact_Sales to the Y axis of the visual container, to render the visual. This should create the column chart representing the sales spreading over months. To add the trend line to it, OrderQuantity can be added to Line which should populate the trend line on the column chart as shown in Figure

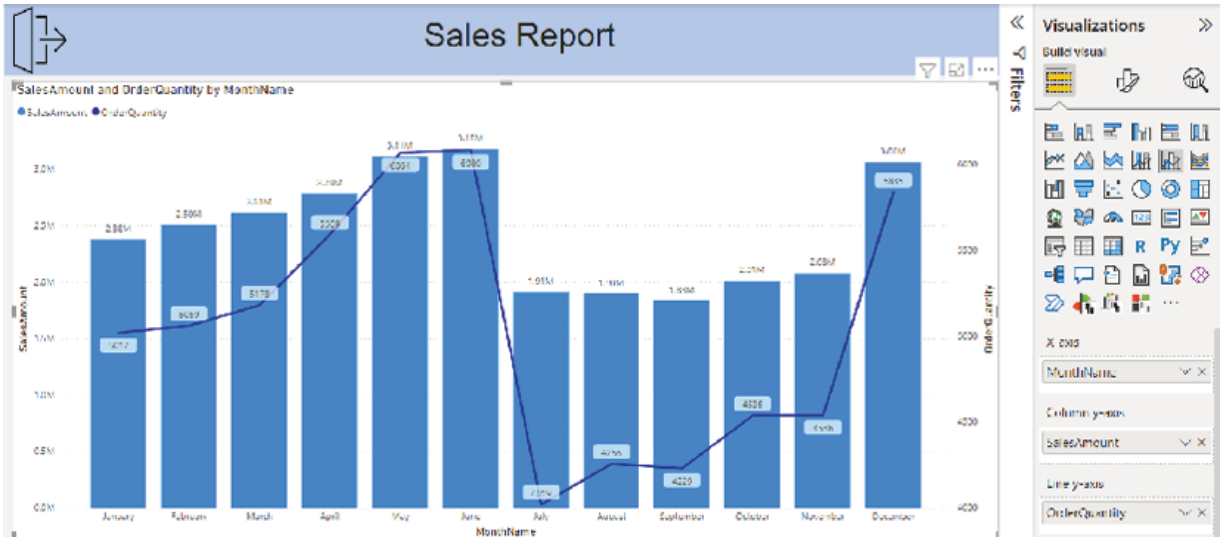



Figure 4.14: Trend Chart

The data labels can be turned on or off from Format visual | Data as required. Apart from that the month names on the X axis may need to be sorted to maintain a logical order. By default, the visuals are sorted based on the values which can be changed from More options of the visual, as shown in Figure

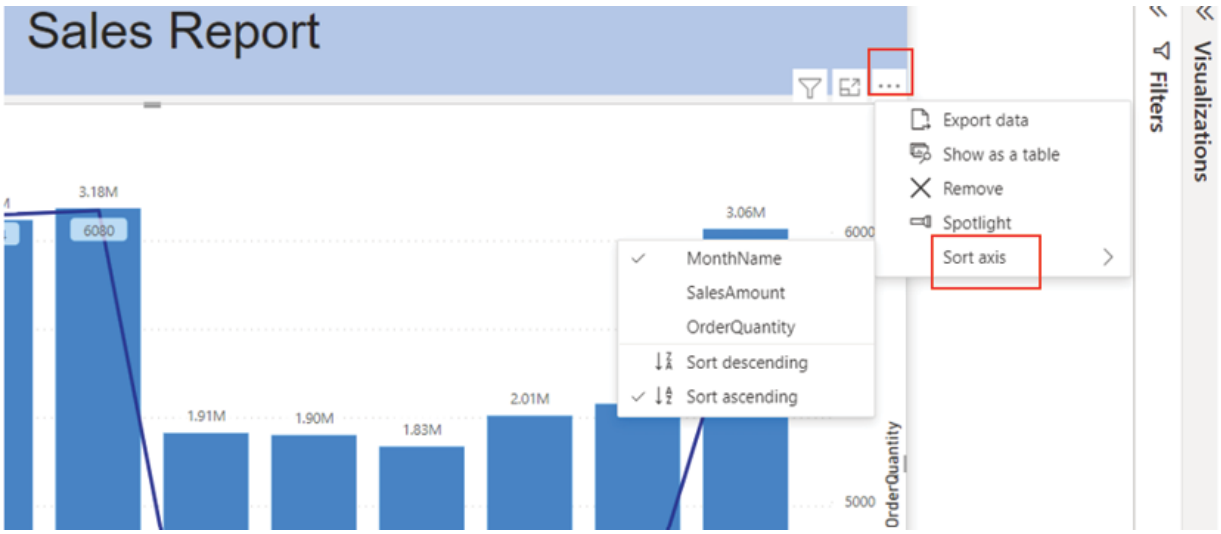



Figure 4.15: Changing the sort axis for a visual

Selecting MonthName as the Sort axis should sort the X axis by month names; however, by default, it would sort alphabetically which again would not make sense here as what we typically need is January should come at first and December at last. To do that, the MonthName field can be selected on the dim_Calendar table, then it can be sorted by a field MonthNumberOf Year following Column tools | Sort by as shown in Figure

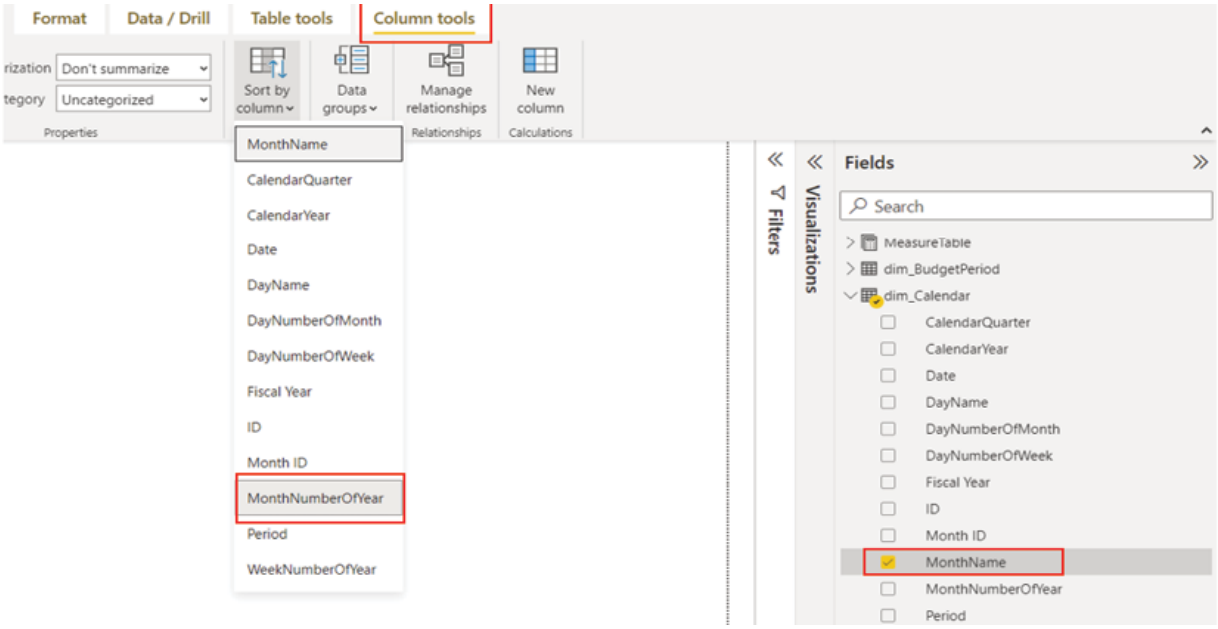



Figure 4.16: Using Sort by column

The MonthNumberOfYear field that we have in dim_Calendar represents the month number of a date, starting from 1 as January and going to 12 as December. Once done, the MonthName field should get sorted by month numbers, wherever used in the report.

Finally, the trend chart can be resized and placed as required; for our report, let us place it below the column chart that we created earlier and move to the next visual we are going to explore.

## Visualizing geographical data using maps

Power BI has a set of maps to represent geographical data; among them, the default bubble map is arguably the most popular one. To use it effectively, all that is needed is a location field in the model using which Power BI can map the coordinates. A location field can include city, postal code, state, country and so on.

Power BI uses the location field for geo-coding and creates the map accordingly. However, a proper categorization of the fields involved can increase the likelihood of correct geo-coding. After selecting a field, Column tools | Data category can be followed to categorize it with one of the available options, as shown in Figure

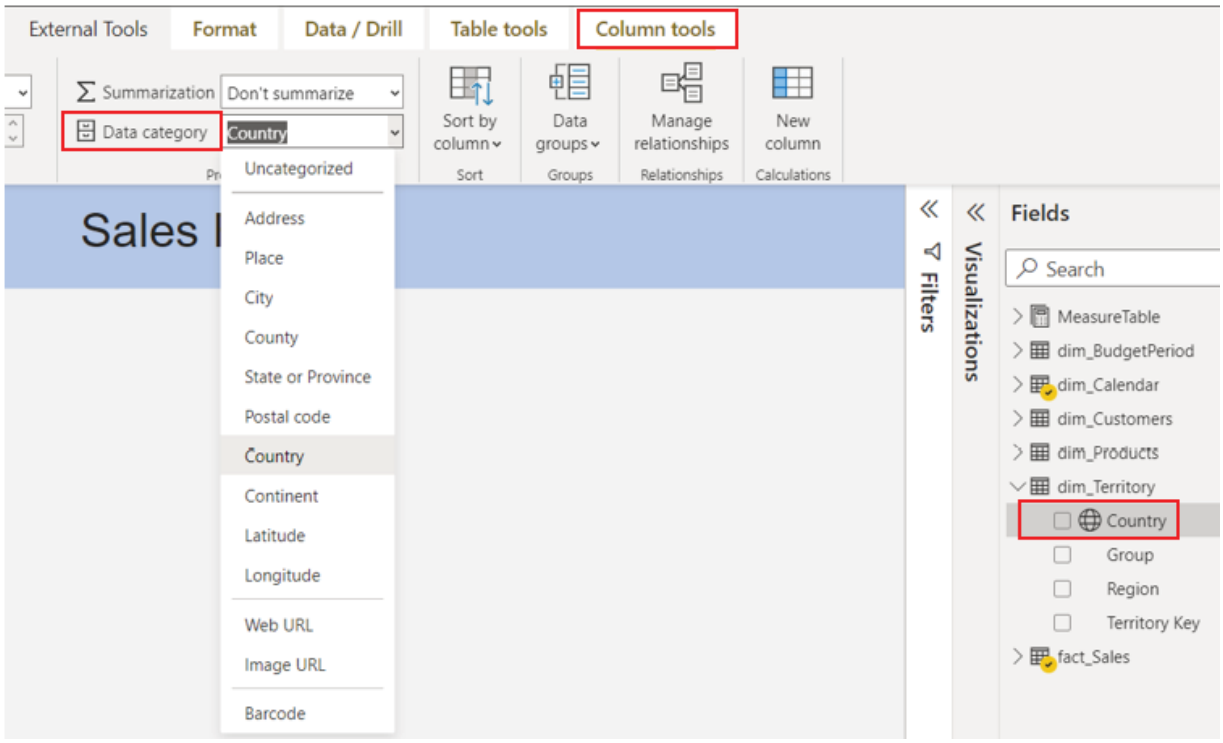


Figure 4.17: Categorizing a field in Power BI

An icon on the field would indicate a successful categorization. The size of the bubble can be determined by the value that is been placed in the Bubble size bucket. For our report, a map can be plotted to view the sales amount for different countries. For doing so, we can select the Map visual from the Visualizations pane, add Country from dim_Territory to the Location bucket and SalesAmount from fact_Sales to the Bubble size bucket of the visual container, to render the visual as shown in Figure

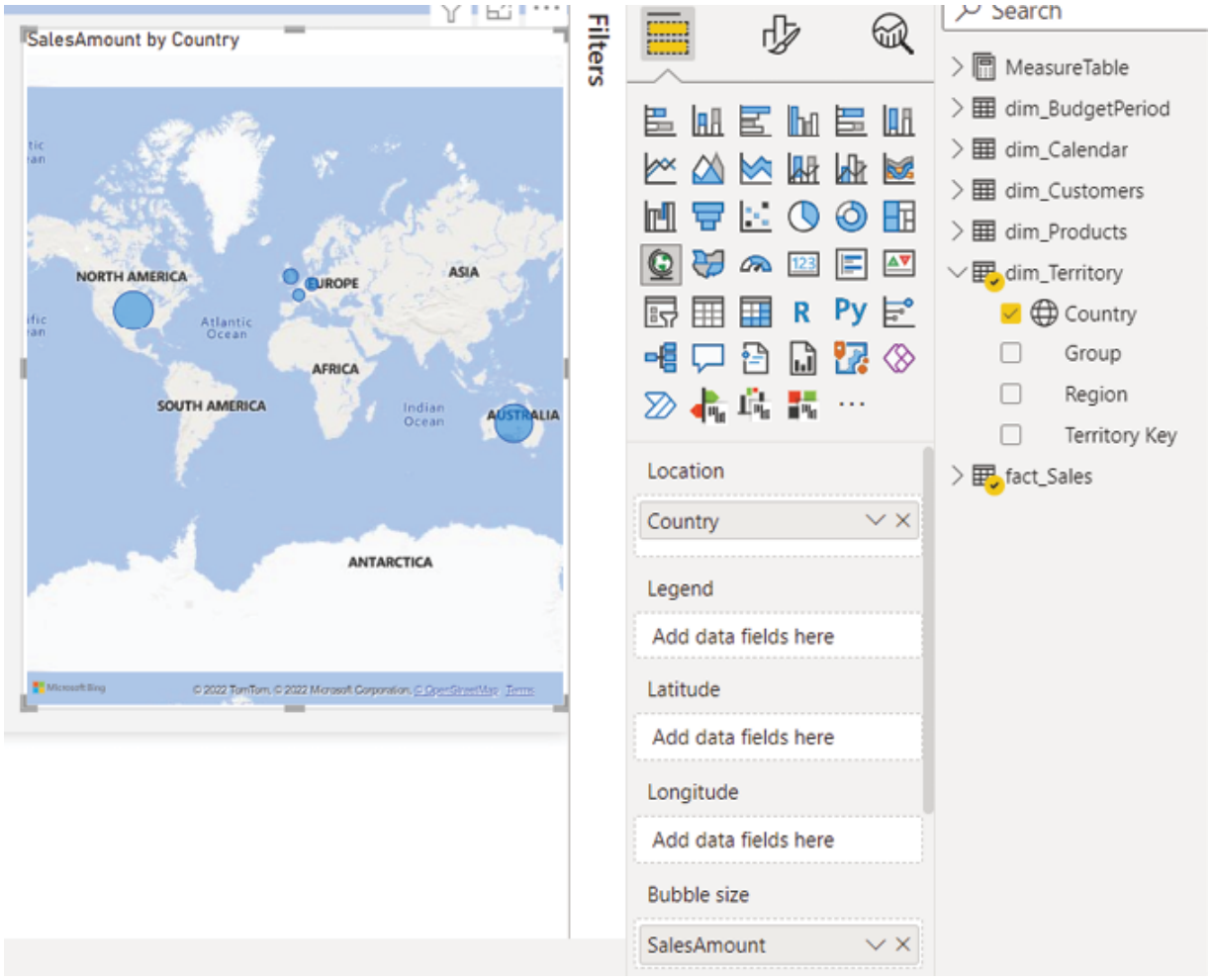



Figure 4.18: Creating a bubble map in Power BI

Only adding one location field might not always be enough, for example, if we add a field for the city as location, it can cause ambiguity as multiple countries can have cities with the same name. To increase the precision and help Power BI to accurately geo-code the location, multiple columns can be used as the location, or additional buckets like Latitude and Longitude can be used if available in the dataset.

Power BI integrates with Bing Maps to geo-code locations. If Latitude and Longitude are provided, then no data is sent to Bing, otherwise, any data in the Location bucket gets sent to Bing.

As always, the visual can be resized and placed wherever required. For our sales report, let us place it just below the bar chart, and move on.

## Use of cards

So far, we have seen how to analyze values across categories; however, many times what is required is to derive a particular value from the underlying data and view the report. This is specifically where a card would be useful as it can be used to display a single scalar value.

For sales reports, time-based calculations like Year to Same period last Year over year growth percentage etcetera are usually the important key performance indicators (KPIs) to track. In Power BI, all these can be calculated conveniently using the time intelligence functions of DAX. Let us first see how to calculate the required measures and then, we would focus on visualizing those on the report. The following four measures can be used in our report to provide us with a comprehensive descriptive analysis of what is been going on with our sales data:

Total The Total Sales measure should calculate the sum of sales amount in the context of the report, using the SUM function of DAX, which we already have explored in the previous chapter. Figure 4.19 shows the DAX formula involved:


Figure 4.19: Total Sales measure

As we are already aware, once created, this measure can be reused by referring to it enclosed within a square bracket.

YTD The year-to-date sales, or YTD Sales measure calculates the running total for sales, use the TOTALYTD function of DAX.

TOTALYTD: This evaluates the year-to-date value of an expression in the current context.

SYNTAX: TOTALYTD(, [,] [,])

PARAMETERS:

Expression returning a scalar value.

A date column.

Expression specifying a filter condition (optional).

A string defining the year-end date (optional, default is December

For our report, let us assume we want to compute the YTD sales based on OrderDate from the fact_Sales table. As the Date field of dim_Calendar is related to OrderDate from it perfectly qualifies to be used as a parameter for our measure. Figure 4.20 illustrates the DAX formula to create the measure YTD

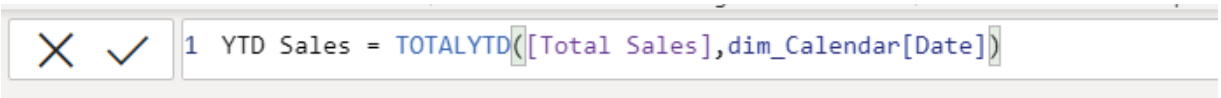



Figure 4.20: YTD Sales measure

Once used in the report, this measure should calculate the year-to-date sales in the context of the report, which we will soon see with an example.

Same Period Last Year Sales: The Same Period Last Year Sales measure calculates the sales amount for the last year, in the current context of the report, using the SAMEPERIODLASTYEAR function of DAX.

SAMEPERIODLASTYEAR: This returns a date column shifted one year back in time from the dates that have been specified, in the current context.

SYNTAX: SAMEPERIODLASTYEAR()

PARAMETER: A date column

For our report, the Same Period Last Year Sales measure can be created as shown in Figure

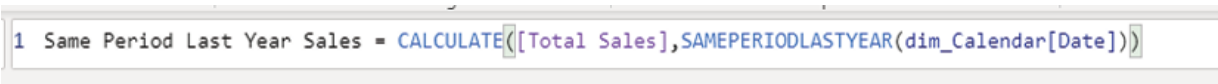



Figure 4.21: Same Period Last Year Sales measure

Again, we will see this measure in work in the upcoming example.

YoY The YoY Growth% measure calculates the percentage of growth of the sales amount compared to the last year, using the DAX code as shown in Figure

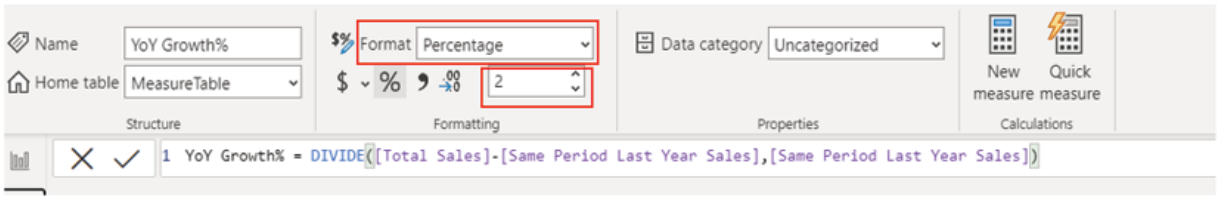



Figure 4.22: YoY Growth% measure

As this is a percentage of growth, the measure can be formatted accordingly as highlighted in the preceding figure.

Let us now refer to an example as shown in Figure 4.23 to understand further how these measures would work in the context of a report, and also validate that they are working as expected:

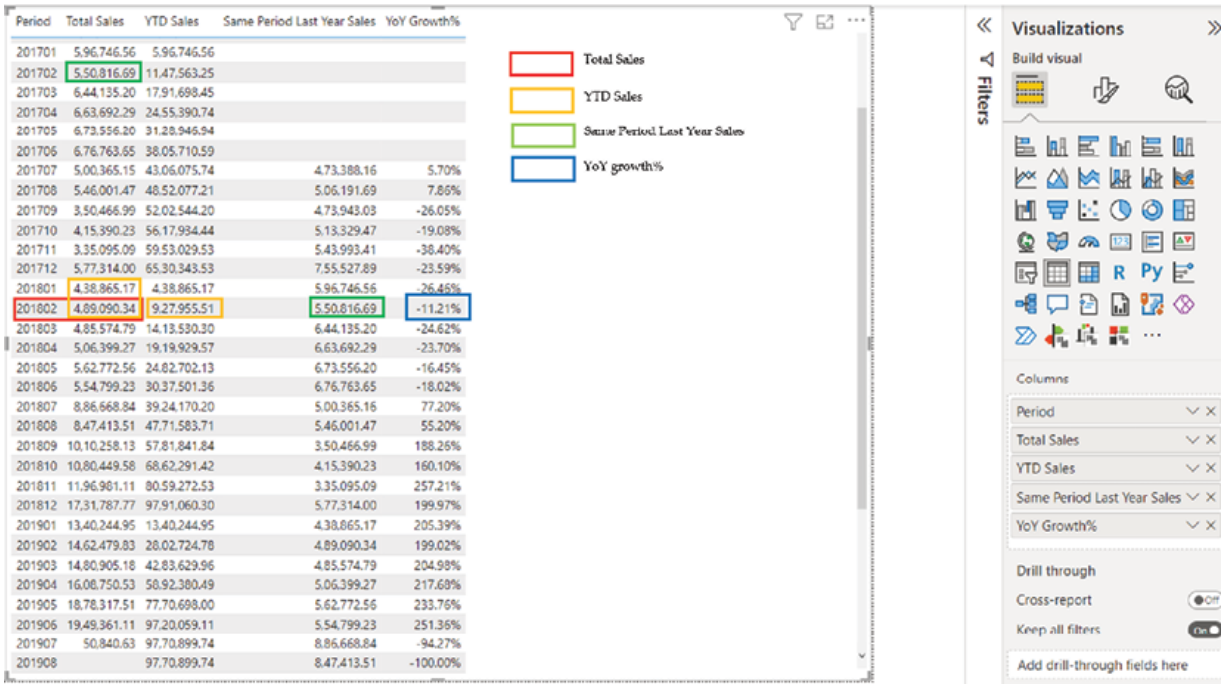



Figure 4.23: Example explaining the time-based measures

In the preceding example, we have used the Table visual and added Period from dim_Calendar as the first column to create a context for the measures.

Then, we added all four measures as Columns of the table.

Let us consider the context of period 201802 as highlighted. The total sales amount for the period is 4,89,090.34 which is exactly what we are having from the Total Sales measure. The running total or the YTD value for sales should consist of all the periods from the beginning of the year, which is 201801 and 201802. The aggregated sales would be (4,38,865.17 + 4,89,090.34) = 9,27,955.51 which is exactly the value we are getting from the measure YTD To get the sales value for the same period of the last year, we should go one year back and refer to the period 201702. Period 201702 has a sales amount of 5,50,816.69 which is the same value we are getting for the measure Same Period Last Year Sales for period 201802. Finally, the year-over-year growth for the period 201802 should be (Sales for 201802 – Sales for 201702), and the percentage would be (Sales for 201802 – Sales for 201702)/ Sales for 201702. If we calculate it, we should get the result as (4,89,090.34 - 5,50,816.69)/ 5,50,816.69 or -11.21%, again matching exactly with the outcome of the YoY Growth% measure!

Now that we have created all the measures required, let us visualize those in our report. To do this, we can select the Card from the Visualizations pane, and add the measure Total Sales to the Fields bucket, to render the visual as shown in Figure

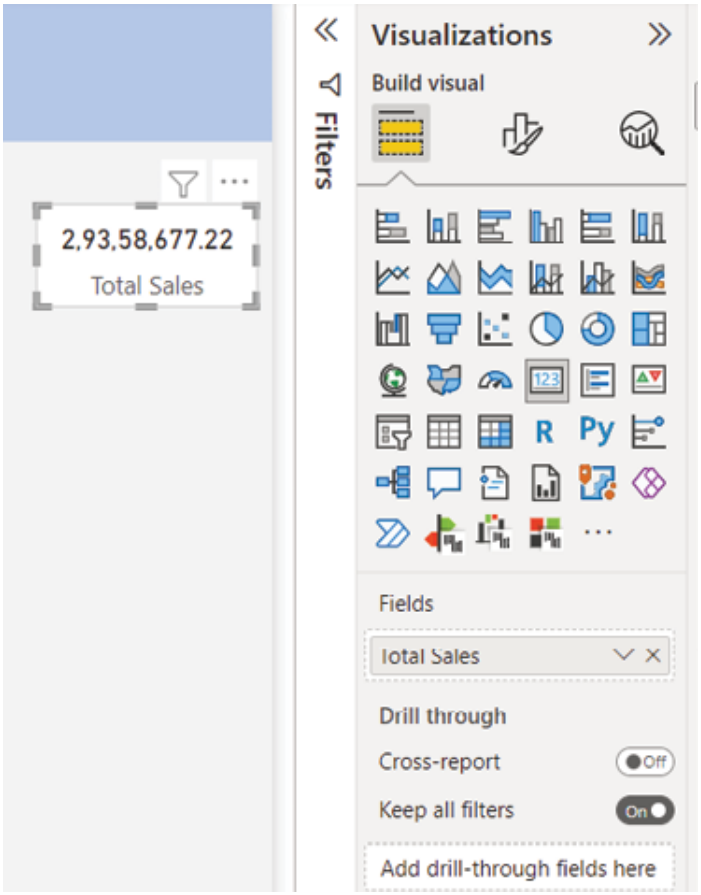



Figure 4.24: Creating a card

The card would display the value of the measure in the context of the report. As always, the visual can be formatted as required. For example, the display unit can be configured by following Format visual | Visual | Callout value | Display

Similarly, we can create cards for the other three measures as well. After resizing and placing the cards, the sales report should now look as shown in Figure

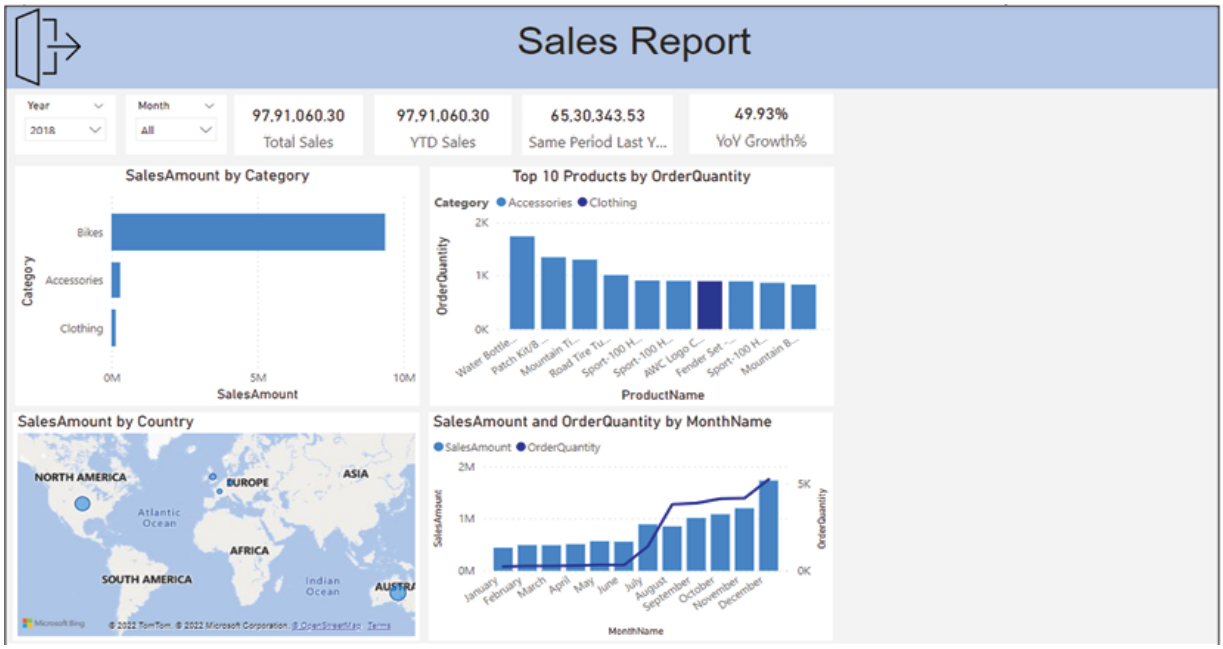



Figure 4.25: Sales report draft 2

Now with every slicer selection, all the values displayed on the card should change and show values as per the context of the report!

Let us explore further and see how to best utilize the space that we are left with on the report page.

Tables

Alongside visualizing data with graphs and charts, we often require viewing the underlying raw data itself to check the granular level details, which a table can help us with. A table visual allows us to take a look at the underlying data in a flat format, without necessarily aggregating.

Let us create a duplicate page of our sales report and rename it by right clicking as Details while the original report page can be named as shown in Figure

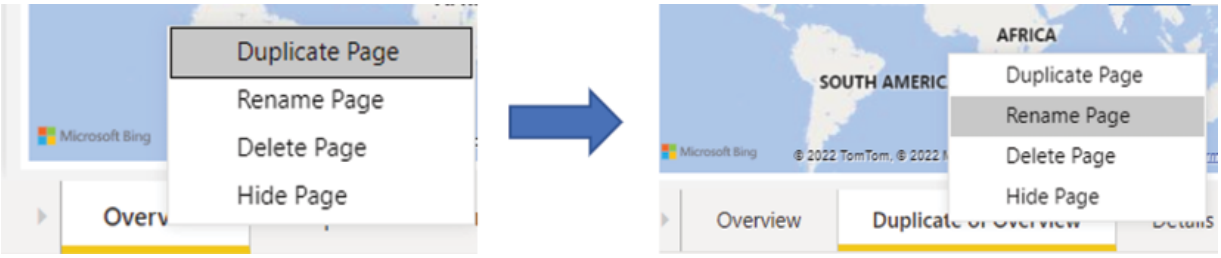



Figure 4.26: Adding a page in the Sales report

The advantage of creating a duplicate page is that we do not need to redesign the page and create the theme all over again. We can remove all the visuals on the duplicated page that are not required and continue with the rest of it.

For our report, let us remove all the other visuals apart from the Year and Month slicers. As we intend to create a table on this page to look at the granular details of the underlying data, it would make sense to introduce a few more slicers to slice and dice the table with different attributes. Hence

let us add a few more slicers namely Country from ProductName and ModelName from Occupation, and YearlyIncome from

To create the table, we can select the Table visual from the Visualizations pane, and add all required categories and values to the Columns bucket of the visual container to render the visual.

The Table visual has a lot of visual-specific formatting options which can be accessed following Format visual | Let us select the style Alternating rows from the Style presets option to apply the style to the table. Now the report page should look like this as shown in Figure


| Group | Country | ProductName | Category | Color | Name | Occupation | YearlyIncome | SalesAmount | DiscountAmount | Freight | OrderQuantity | ProductStandardCost | TaxAmt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Europe | France | All-Purpose Bike Stand | Accessories | NA | April Shan | Clerical | 40000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Bonnie Lal | Clerical | 10000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Brad Chande | Clerical | 40000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Curtis Liu | Manual | 10000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Damien Sun | Management | 100000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Desiree Jimenez | Clerical | 30000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Elijah Jai | Clerical | 30000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Frank Gomez | Clerical | 30000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Josue Blanco | Manual | 10000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Kevin Bryant | Clerical | 30000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Lisa Cai | Professional | 100000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Neil Rubio | Manual | 20000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Omar Zhou | Manual | 30000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Rebecca Hernandez | Manual | 20000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Roger Zeng | Manual | 10000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Taylor Russell | Manual | 20000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Tiffany Zhou | Manual | 20000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Tyrone Sanz | Manual | 10000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |
| Europe | France | All-Purpose Bike Stand | Accessories | NA | Willie Chander | Professional | 110000 | 159.00 | 0 | 3.98 | 1 | 59.47 | 12.72 |

Figure 4.27: Details page in the Sales report

Let us now go back to the Overview page to explore another form of a table, which is the Matrix visual.

## Matrix

While the Table visual is just a two-dimensional grid containing related data in rows and columns, the Matrix visual allows you to display data in multiple dimensions by allowing users to drill down. This can be an ideal option to replace all those pivot tables created in Excel.

The visual content can be created by selecting Matrix on the Visualizations pane. The attributes can be placed on either Rows or Columns bucket, while numeric fields and measures are added as Once multiple fields are added to Rows or the Matrix creates a hierarchy enabling users to drill down (and drill up) through the hierarchy levels and analyze data at each level.

For the report we have been creating, let us add Category and SubCategory from dim_Products to and SalesAmount and OrderQuantity from fact_Sales to Values bucket. This should enable us to analyze the values by product categories. We should also be able to go to a more granular level, which is the product sub category and analyze data, as shown in Figure

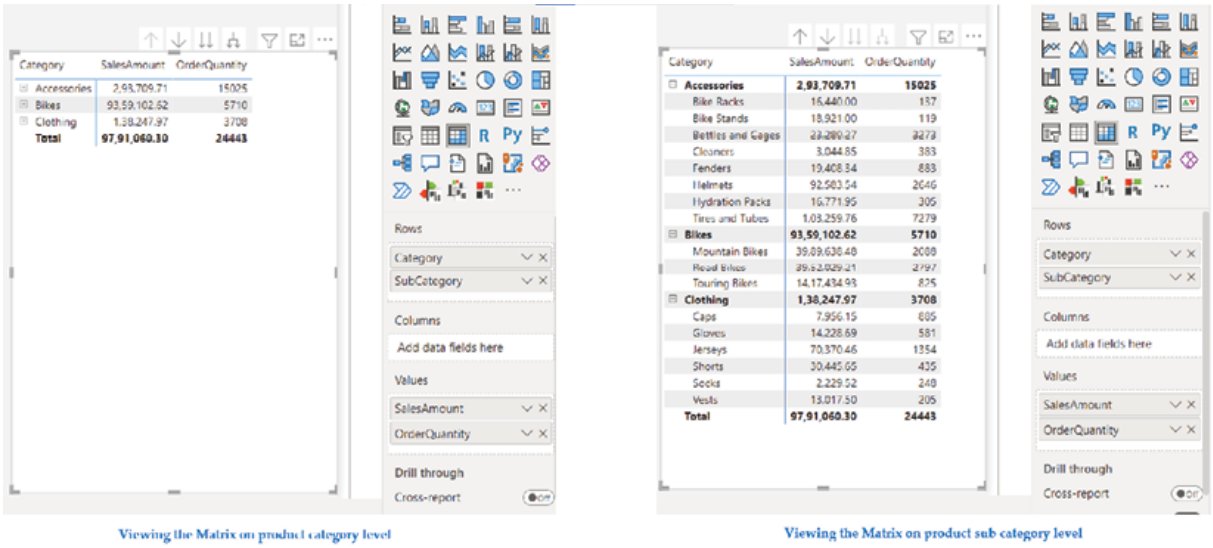



Figure 4.28: Viewing the same Matrix visual on different granularity

As seen in the preceding figure, the aggregated values for each product category are exactly the same on both the views; however, on a higher granularity level, we also get to see the details for individual sub categories, under different product categories. Similarly, hierarchies can be created on Columns as well if required.

To navigate through different hierarchies or granular levels, there are a few ways. By default, a Matrix displays the least granular view unless drilled down further. Taking the Matrix that we have just created as an example, expanding the plus (+) icon beside each product category should take us to the next level of granularity only for that specific category while the other categories will still remain in the existing granularity level, as shown in Figure

| Category | SalesAmount | OrderQuantity |
|---|---|---|
| **Accessories** | **2,93,709.71** | **15025** |
| Bike Racks | 16,440.00 | 137 |
| Bike Stands | 18,921.00 | 119 |
| Bottles and Cages | 23,280.27 | 3273 |
| Cleaners | 3,044.85 | 383 |
| Fenders | 19,408.34 | 883 |
| Helmets | 92,583.54 | 2646 |
| Hydration Packs | 16,771.95 | 305 |
| Tires and Tubes | 1,03,259.76 | 7279 |
| **Bikes** | **93,59,102.62** | **5710** |
| **Clothing** | **1,38,247.97** | **3708** |
| **Total** | **97,91,060.30** | **24443** |

Figure 4.29: Moving to the next granular level for a specific category

As expected, collapsing with the minus (-) icon should restore the granularity of category

Apart from using the icons, right clicking on any category should open the menu having Drill down as an option, which also can be used to navigate to the next level of granularity only for that category, while the other categories would not be visible anymore. The behavior is illustrated in Figure

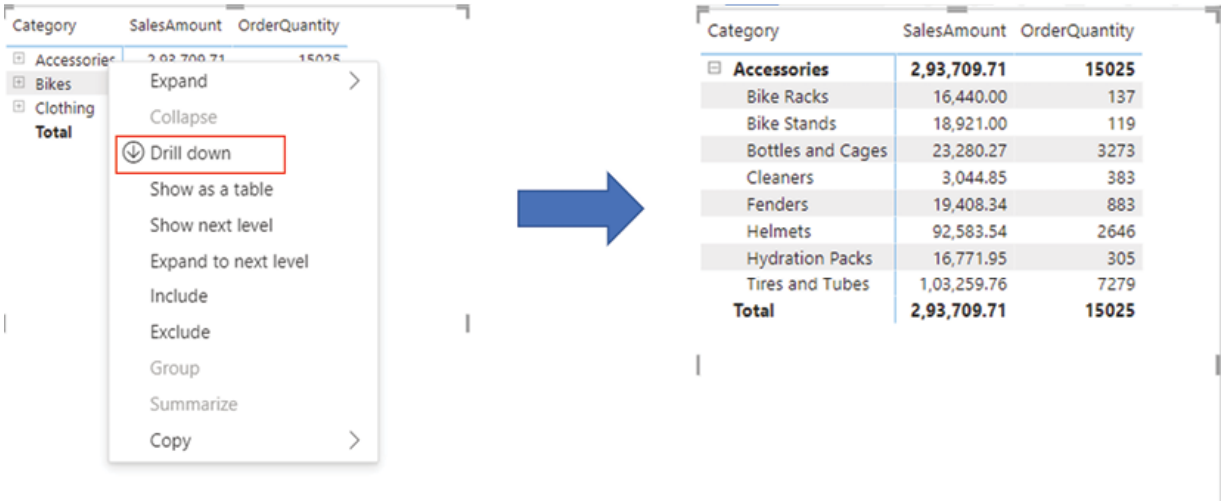



Figure 4.30: Drilling down to the next granular level for a specific category

The Drill up option can of course be used to move back to the aggregated level, by right clicking on any sub category.

Hovering over or selecting any Matrix visual should also display the navigation arrows on the top right corner of the visual. The pitchfork button on the extreme right would expand all categories down one level in the hierarchy while the downward double arrow should take us to the next level of the hierarchy, displaying all the subcategories. Figure 4.31 should help to illustrate the different behaviors:

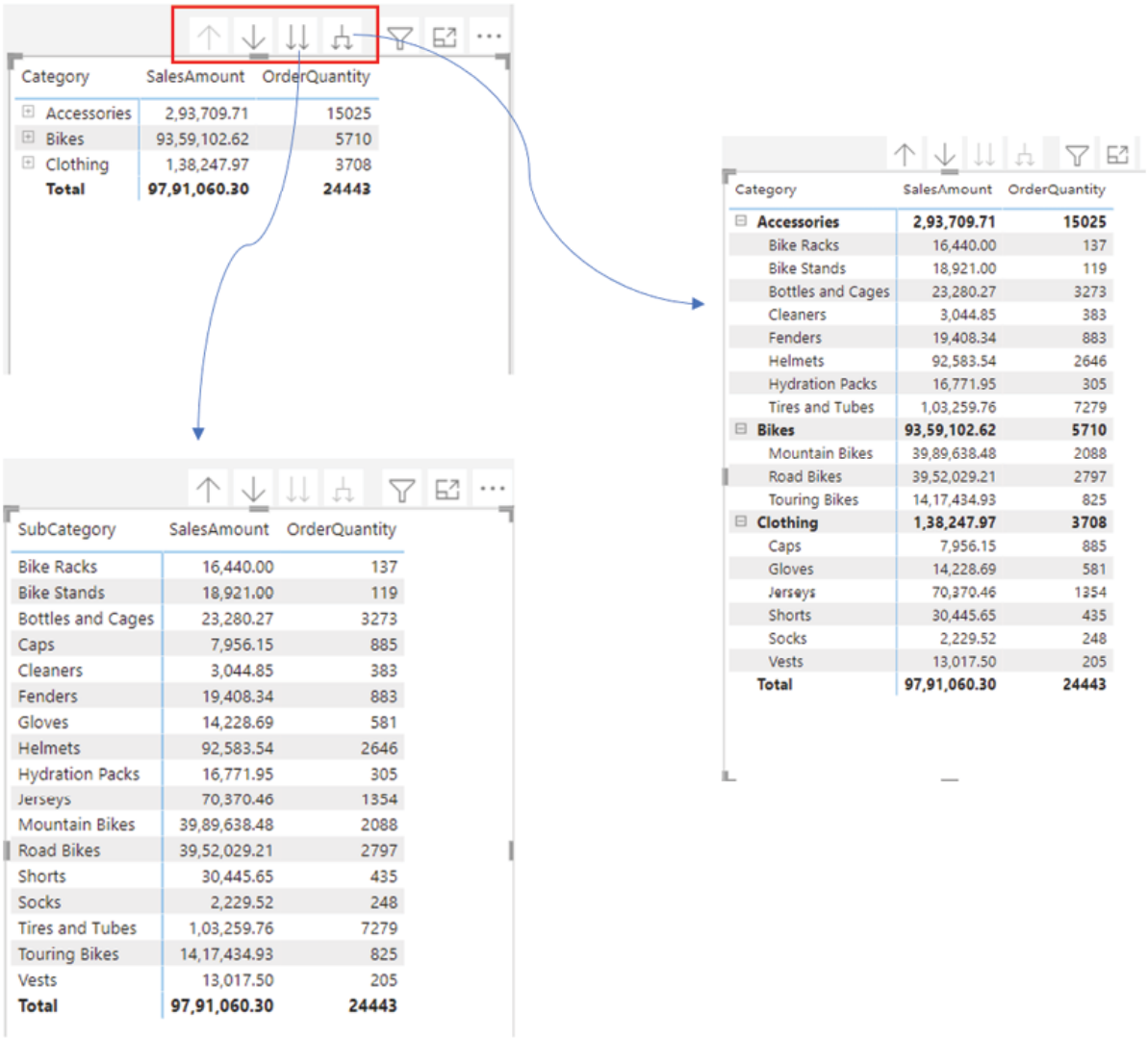



Figure 4.31: Drilling down on the visual

The single down arrow can be used to enable the drill mode, which allows drilling down by simply clicking a data point. Lastly, the up arrow can be used to drill up through the hierarchy levels.

By default, the Matrix visual indents subcategories in a hierarchy beneath the parent; this property is known as Stepped layout and can be controlled from the visual formatting by following Format visual | Row headers | Options | Stepped Figure 4.32 shows how the visual looks like when the Stepped layout is enabled versus when it is disabled:

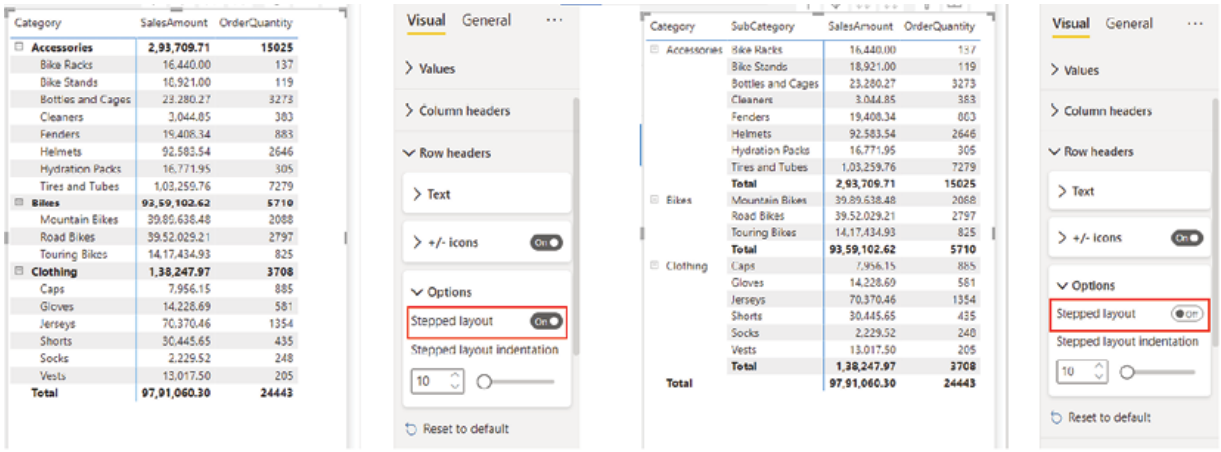



Figure 4.32: Stepped layout On vs. Off

The Matrix visual has a wide range of formatting options that can be explored. For our report, let us apply a few conditional formatting for the SalesAmount field. On the Visualizations pane, clicking on the drop-down button on the SalesAmount field opens the menu where under Conditional all available options can be explored. Using the Data bars option, each cell of the SalesAmount can be formatted with divergent bar colors based on the values, as shown in Figure

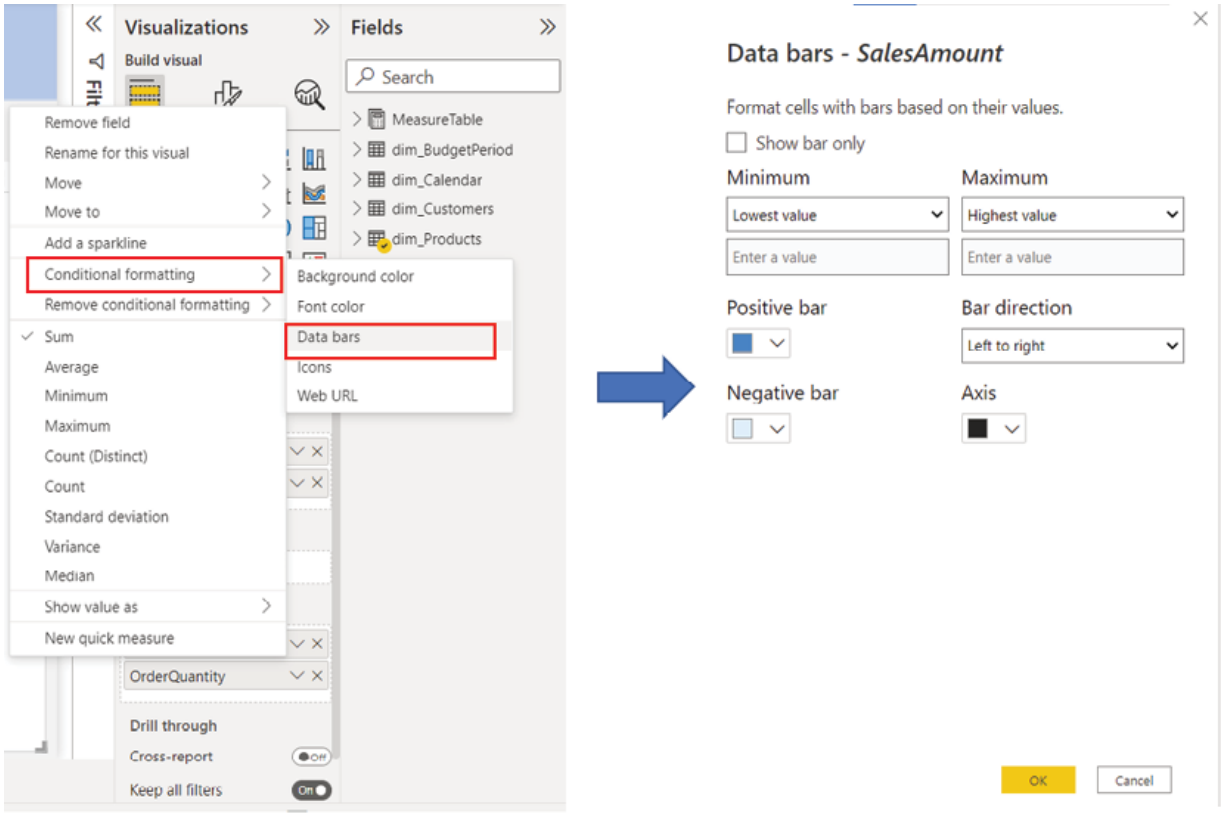



Figure 4.33: Applying Data bars to SalesAmount

Also, the Icons option just after Data bars can be used to conditionally add indicative icons for each value representing how the specific item is performing. Let us add the following configuration for Icons as shown in Figure

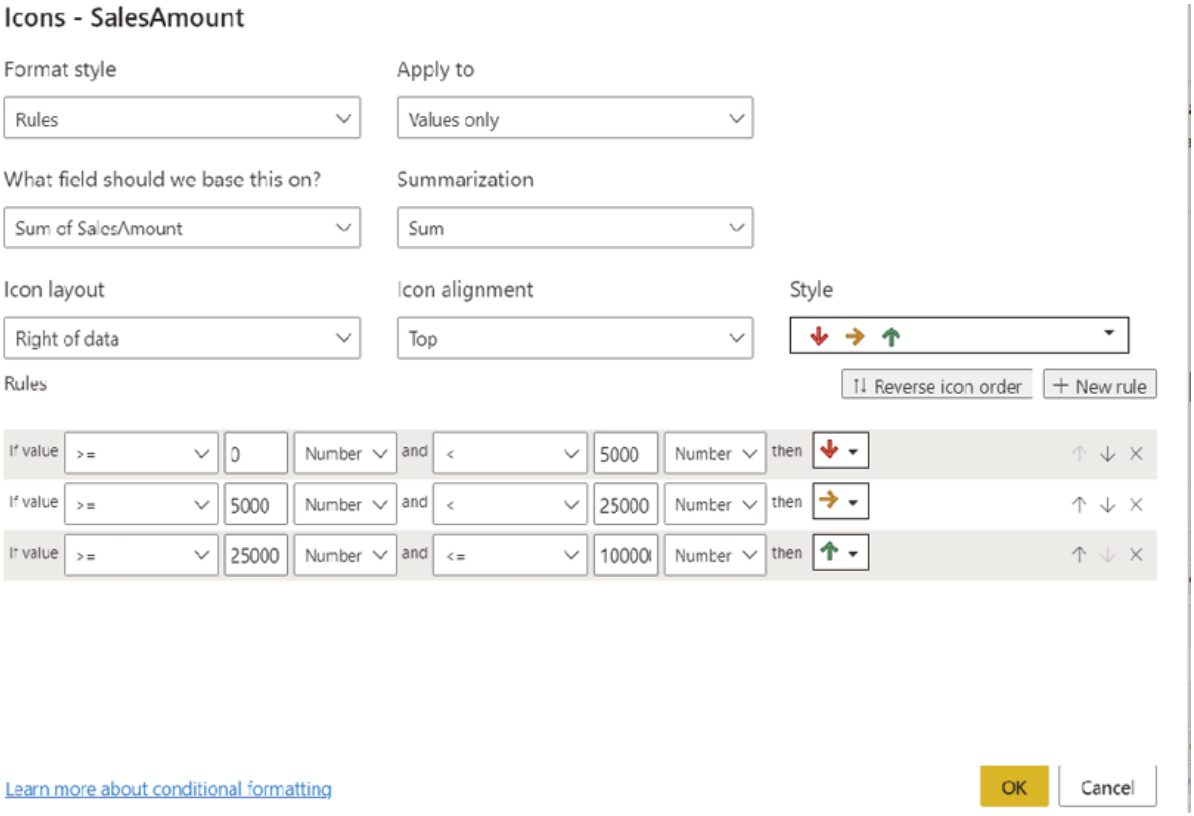



Figure 4.34: Applying Icons to SalesAmount

After applying these conditional formatting configurations, the Matrix should look like as shown in Figure

| Category | SalesAmount | OrderQuantity |
|---|---|---|
| **Accessories** | **2,93,709.71** | **15025** |
| Bike Racks | 16,440.00 | 137 |
| Bike Stands | 18,921.00 | 119 |
| Bottles and Cages | 23,280.27 | 3273 |
| Cleaners | 3,044.85 | 383 |
| Fenders | 19,408.34 | 883 |
| Helmets | 92,583.54 | 2646 |
| Hydration Packs | 16,771.95 | 305 |
| Tires and Tubes | 1,03,259.76 | 7279 |
| **Bikes** | **93,59,102.62** | **5710** |
| Mountain Bikes | 39,89,638.48 | 2088 |
| Road Bikes | 39,52,029.21 | 2797 |
| Touring Bikes | 14,17,434.93 | 825 |
| **Clothing** | **1,38,247.97** | **3708** |
| Caps | 7,956.15 | 885 |
| Gloves | 14,228.69 | 581 |
| Jerseys | 70,370.46 | 1354 |
| Shorts | 30,445.65 | 435 |
| Socks | 2,229.52 | 248 |
| Vests | 13,017.50 | 205 |
| **Total** | **97,91,060.30** | **24443** |

Figure 4.35: Matrix with conditional formatting

All these different configuration options make the Matrix as one of the most powerful and popular visuals of Power BI. Let us resize the Matrix as required and place it on the empty space that is available on the Overview page, after leaving some space from the top ribbon.

Tip: For tables and matrices, the Total field is not always the addition of the rows displayed in the visual, rather it is evaluated on the underlying data and displayed in the context of the report. If not required, the option of displaying Total can be disabled from the visual formatting options.

Introduction to custom visuals

At this point of authoring the report, the Overview page should look like Figure

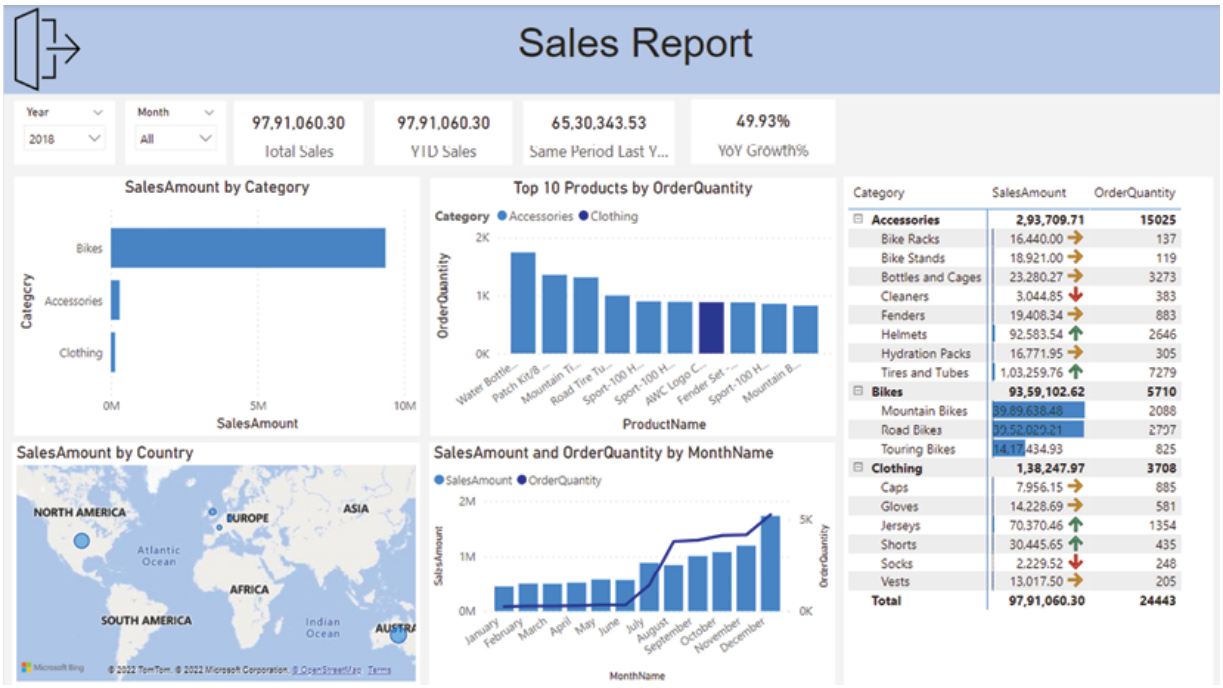



Figure 4.36: Sales report draft 3

Though the report already seems able to provide a lot of information and insight about the data it is built on; however, in terms of comparing with the last year's data, all we have is scalar values represented by cards. It would have been really nice to see the month by month comparison in a single chart. To do this, the default Line chart under the Visualizations pane can be utilized, however, there is an even more powerful option to explore.

Power BI has a rich set of visuals available at the fingertip, however, only a few selected commonly used visuals are pre-packaged and come out of the box. Apart from the default ones, many more certified custom visuals are available in the Microsoft which can also be utilized. AppSource can be considered as a marketplace where Microsoft and partners publish custom visuals, certified to be used in Power BI.

To go to the the Get more visuals option can be accessed from the ellipsis icon of the Visualizations pane, as shown in Figure

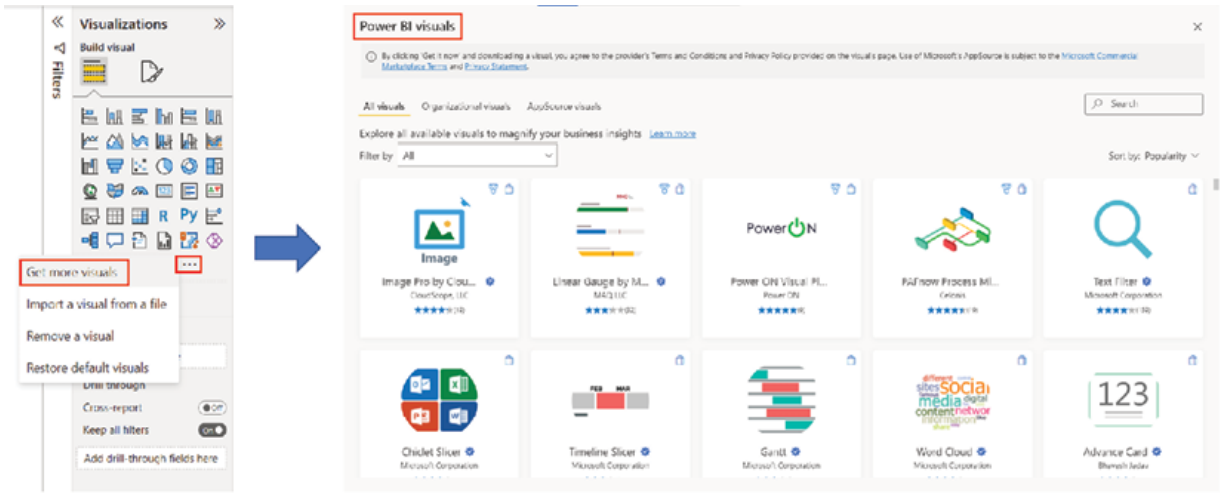



Figure 4.37: Microsoft AppSource for Power BI visuals

On the clicking on any visual redirects to that visual page, where the visual overview and sample reports using those visuals are available. The user can navigate through the AppSource visuals to find out the suitable one, or any specific visual can also be searched for using the search box on the top right. Many AppSource visuals are free to use in Power BI, while some additional licenses are required. The pricing information for a visual can also be found on the visual page itself.

We are interested in using a free visual named Power after locating the visual on AppSource and entering the visual page, the visual can be added

using the Add button to the Visualizations pane of Power BI, as shown in Figure

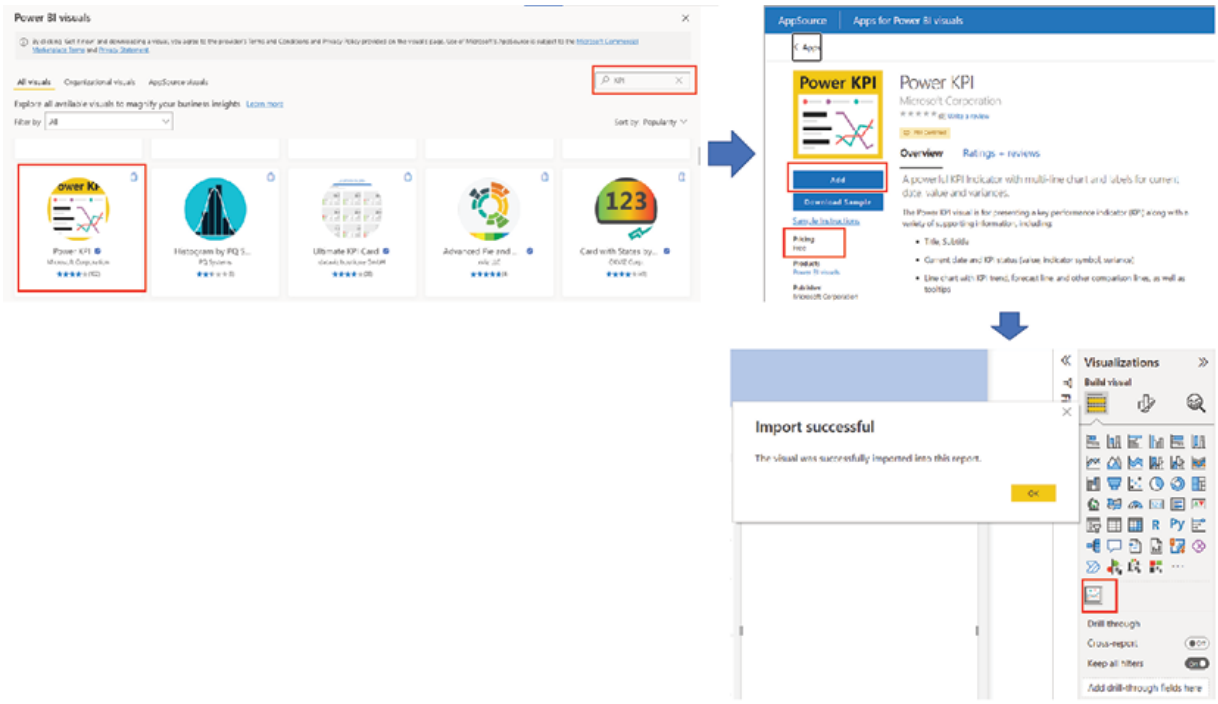



Figure 4.38: Adding a custom visual to Power BI

Once added, the visual appears as a new icon in the Visualizations pane for the current report. To make the visual available for all reports, the Pin to visualizations pane option can be used after right clicking the visual.

Now that we have added the custom visual as we want, let us proceed with rendering the visual by selecting it on the Visualizations pane and then adding MonthName from dim_Calendar as Axis and the measures Total Same Period Last Year Sales as as we want to compare the sales values between the current and previous year, in the report context.

Once created it can be seen that not only it populates the comparison lines between the present and last year as expected, but it also displays the value for the last month in the filter context of the report, as well as the variance

of it with respect to the last year, as labels on the top! Figure 4.39 shows the Power KPI visual we have just created:

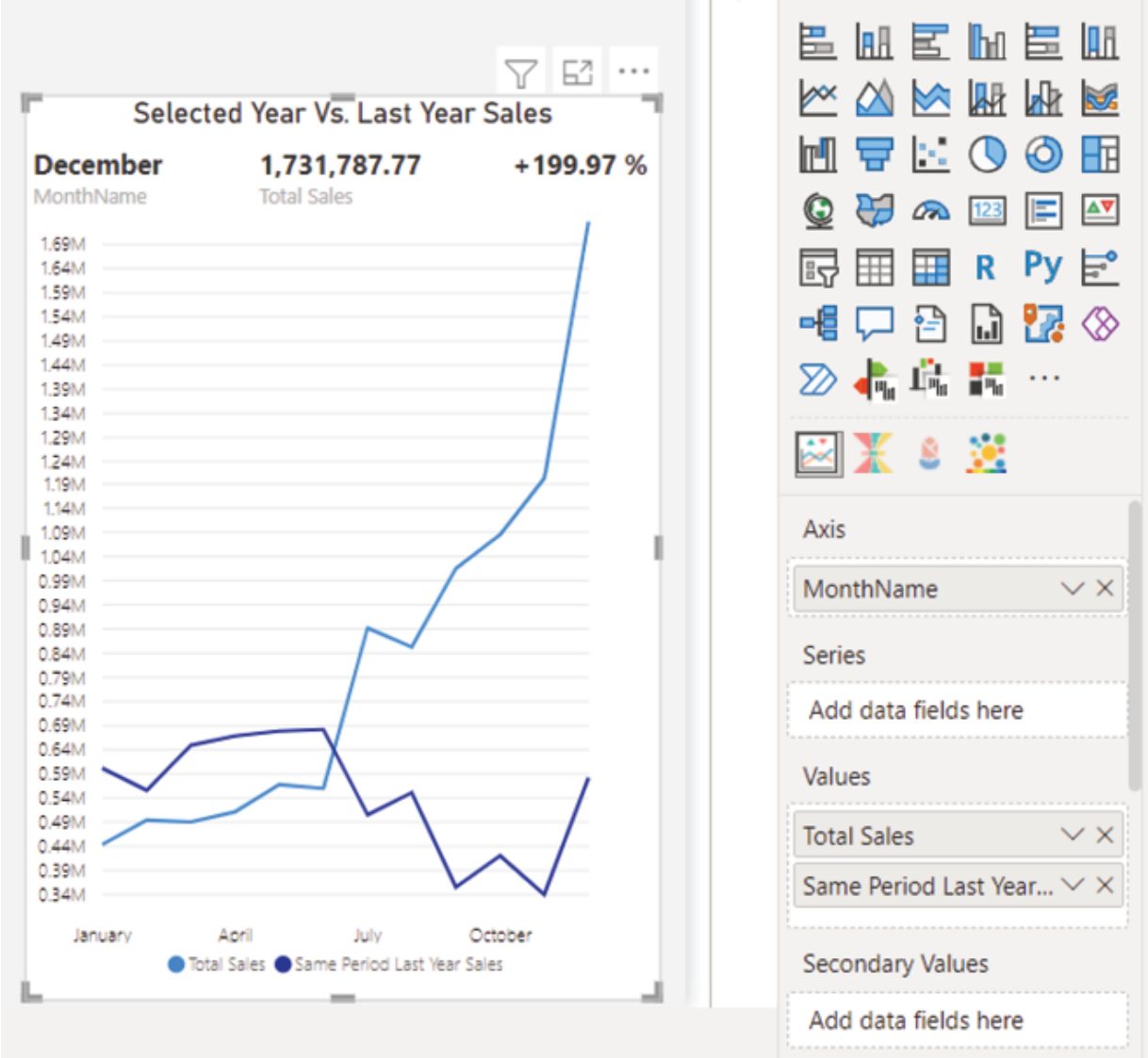



Figure 4.39: Custom visual Power KPI

As always, properties like visual header, data color, and so on can be controlled from the visual formatting options.

Though we have created the visual now, there is no space available on the Overview page to place it anymore. For similar scenarios, we may aim to

toggle between two or more visuals with the help of the Selection pane and Bookmarks which we are going to discuss next.

Bookmarks pane and Selection pane

The Power KPI visual that we created, let us resize it with the exact same dimensions as the Matrix visual and then place it on the top of the Matrix so that the two visuals overlap with each other.

Tip: To adjust the dimension of any visual, select the visual and then go to Format visual | General | Properties | Size.

The idea of toggling between visuals is, when one visual is displayed, the other will be hidden and vice versa. This can be achieved by assigning different actions to two buttons, as shown in Table

Table 4.1: The concept of toggling using bookmarks

As seen in the preceding figure, only one visual would be visible at any given point in time. The properties of hiding or displaying a visual can be controlled using the Selection which can be activated following View | The Selection pane would have all the objects that have been used in the active report page, with the icon to show/hide every available object, as shown in Figure

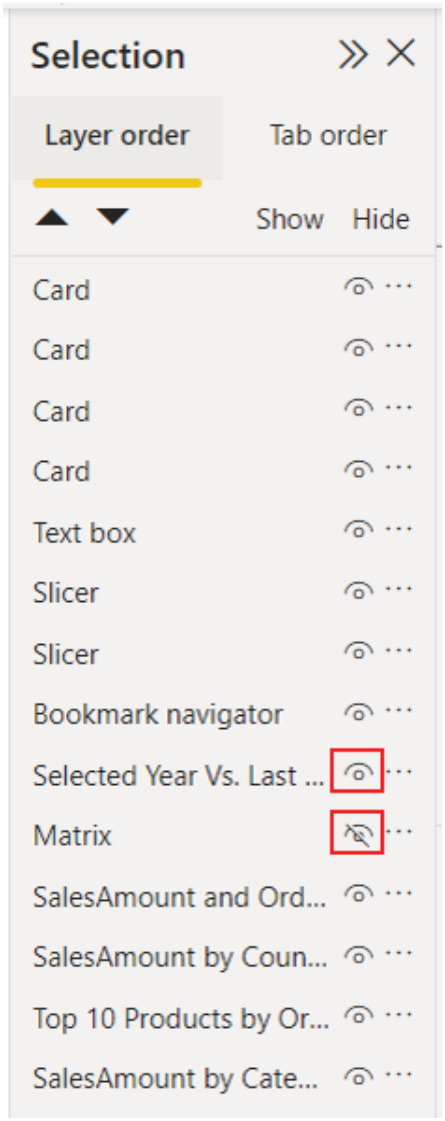



Figure 4.40: Selection pane in Power BI

As shown in Figure the Matrix is hidden while the Power KPI visual is shown; in this state of the report, only the Power KPI visual would be displayed. Here comes the Bookmarks pane using which we can capture a specific state of the report.

To open the Bookmarks pane, click on View | For our purpose, we need to create two bookmarks capturing two different states of the report. For one state, the Power KPI would be visible, and the Matrix would be hidden,

while it should be the other way round for another state. To create a bookmark, we just need to update the report in the state which we want to capture, and then bookmark that state using the Add button under the Bookmarks pane by simply providing a name for the bookmark. Figure 4.41 shows the two bookmarks that we need, along with the report states that they are assigned to:

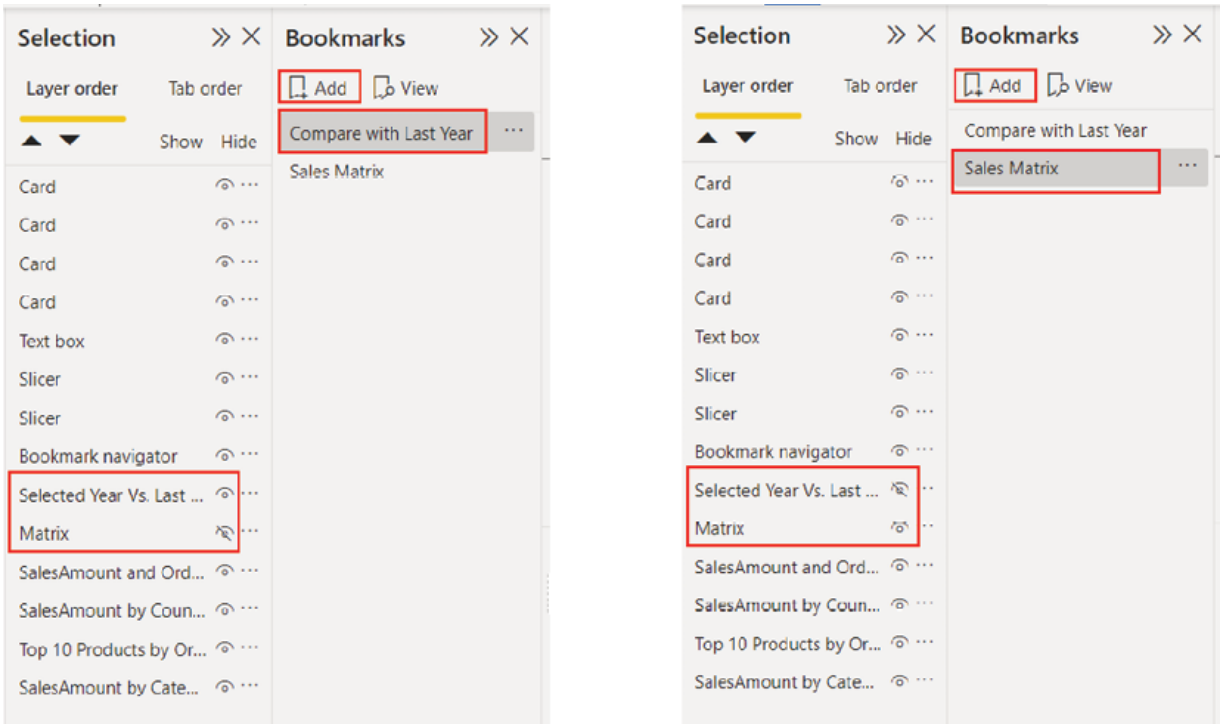



Figure 4.41: Bookmarks capturing different states of the report

Now, whenever the bookmark named Compare with Last Year would be selected, the report will display the Power KPI visual while the Matrix would be hidden. On the other hand, Selecting the Sales Matrix bookmark will cause the report to display the Matrix visual and hide the Power

What is left now is to create a mechanism for the report users to access those bookmarks from the report itself. There is more than one way to do that, for instance, a blank button can be inserted into the report and a bookmark can be selected as an Action for that button, however, the easiest

way for us would be to use the Bookmark which can be found by following Insert | Buttons | Navigator | Bookmark Once added, it will create a group of navigation buttons for all the bookmarks available in the report. The buttons can be selected and formatted just like any other visual using the Format navigator options. Let us choose the button shape as Chevron arrow by following Format navigator | Visual | as illustrated in Figure

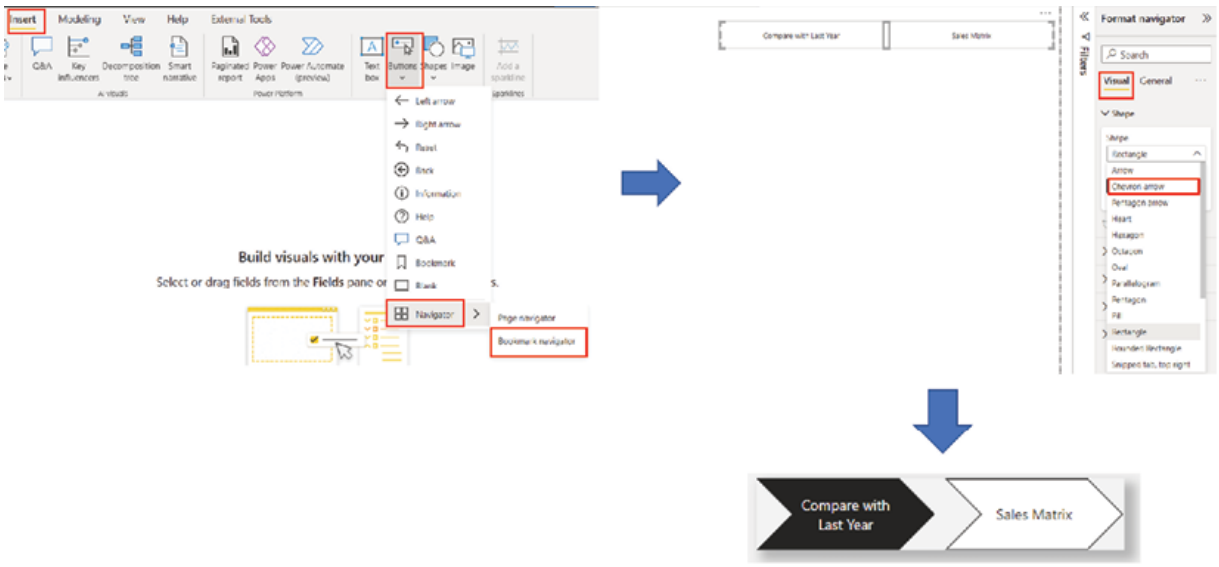



Figure 4.42: Creating bookmark navigator

Let us resize the navigator and place it to the obvious location; just above the Matrix or Power KPI visual. Now, we can toggle between the two visuals using the navigator buttons, only one visual being displayed at any given time as illustrated in Figure

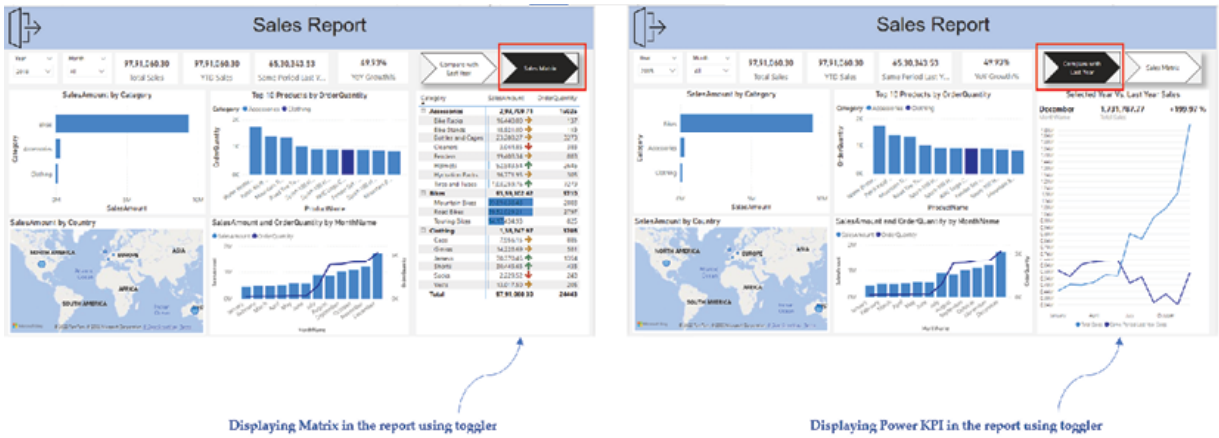



Figure 4.43: Toggling between visuals using bookmark navigator

If needed, the bookmarks can be updated to capture a new state, using the Update option accessible from the ellipsis icon available for every bookmark in the Bookmarks as shown in Figure

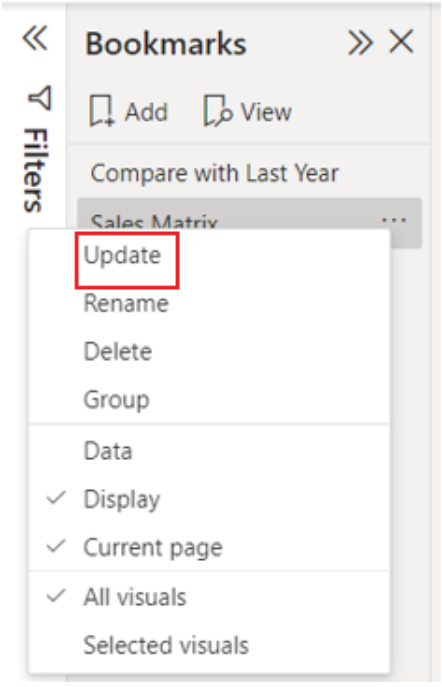



Figure 4.44: Updating a bookmark

Tip: The Data option should generally remain not selected, unless we want to capture the state of present data as well in the bookmark, along with the present report structure.

## Report tooltips

Now that we are almost there with what we wanted to accomplish with our sales report, let us explore a bit about how the report can be further enhanced. One usual challenge is how to effectively provide additional information in the report, where Report tooltips can help.

When we hover over a visual, the default tooltip comes into play and provides us with information in the context of the visual. Whenever a visual is created, the Tooltips property is enabled by default, which can be controlled from the visual formatting options. Figure 4.45 shows the default tooltip on the Map visual of our Overview page, as well as the option to control the tooltip properties:

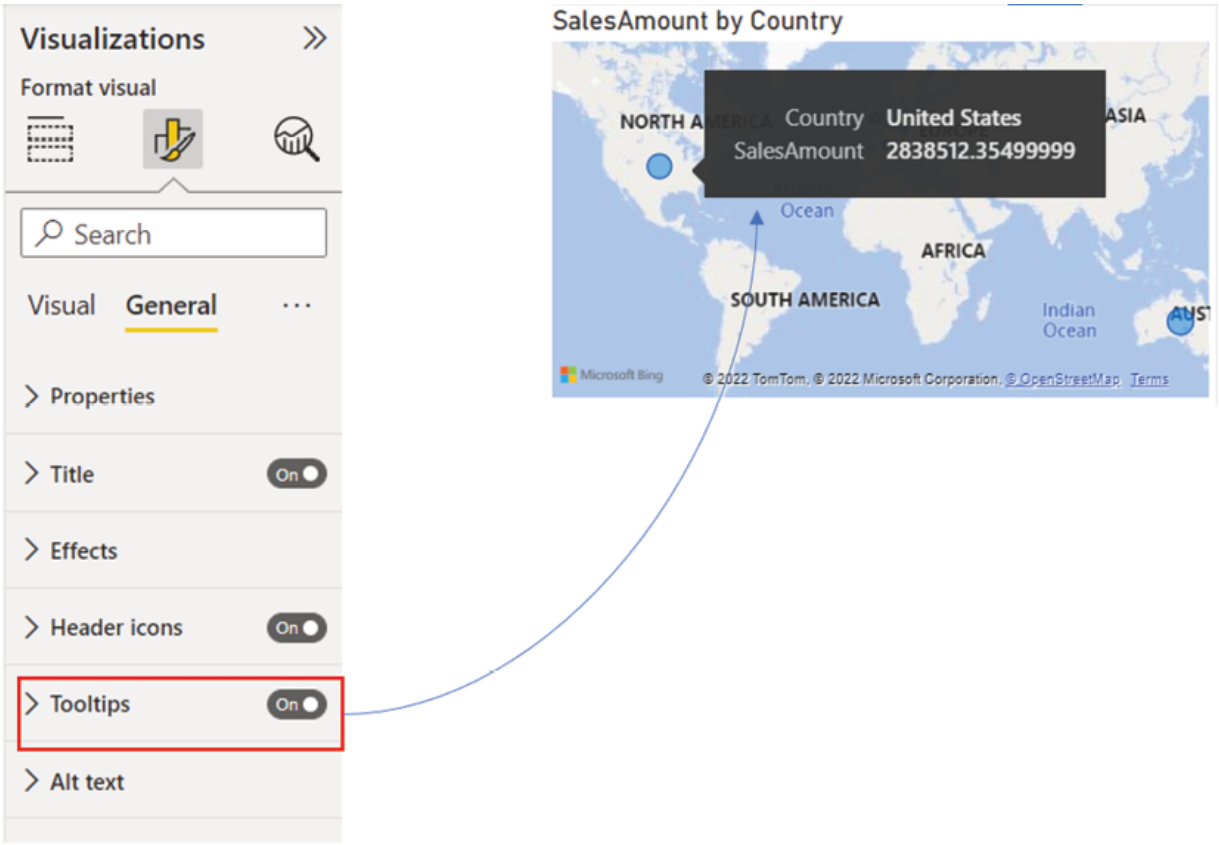


Figure 4.45: Default tooltip

Instead of the default tooltips, Report tooltips can be used to provide additional information about the visual. Few configurations are required to use a custom tooltip on a visual. First of which is of course, we need to create the tooltip.

A separate page can be added and renamed as to create the tooltip we want to use for the Map visual of the Overview page. Let us use the Area chart from the Visualizations pane and add Category from dim_Products to the X axis and SalesAmount from fact_Sales to the Y axis of the visual. The Area chart should then render and display the total sales amount across different product categories. Then, we need to go to the Format page options and turn on the Always use as tooltip option under Page This should allow the Tooltip page to be used for displaying tooltips. Then, under the Canvas settings option, the Type needs to be selected as which should resize the Area chart in accordance with a tooltip. Finally, the Tooltip page can be hidden so that once published, it does not get displayed for the report consumers. Figure 4.46 illustrates the configurations required on the tooltip page along with the Area chart that has been created:

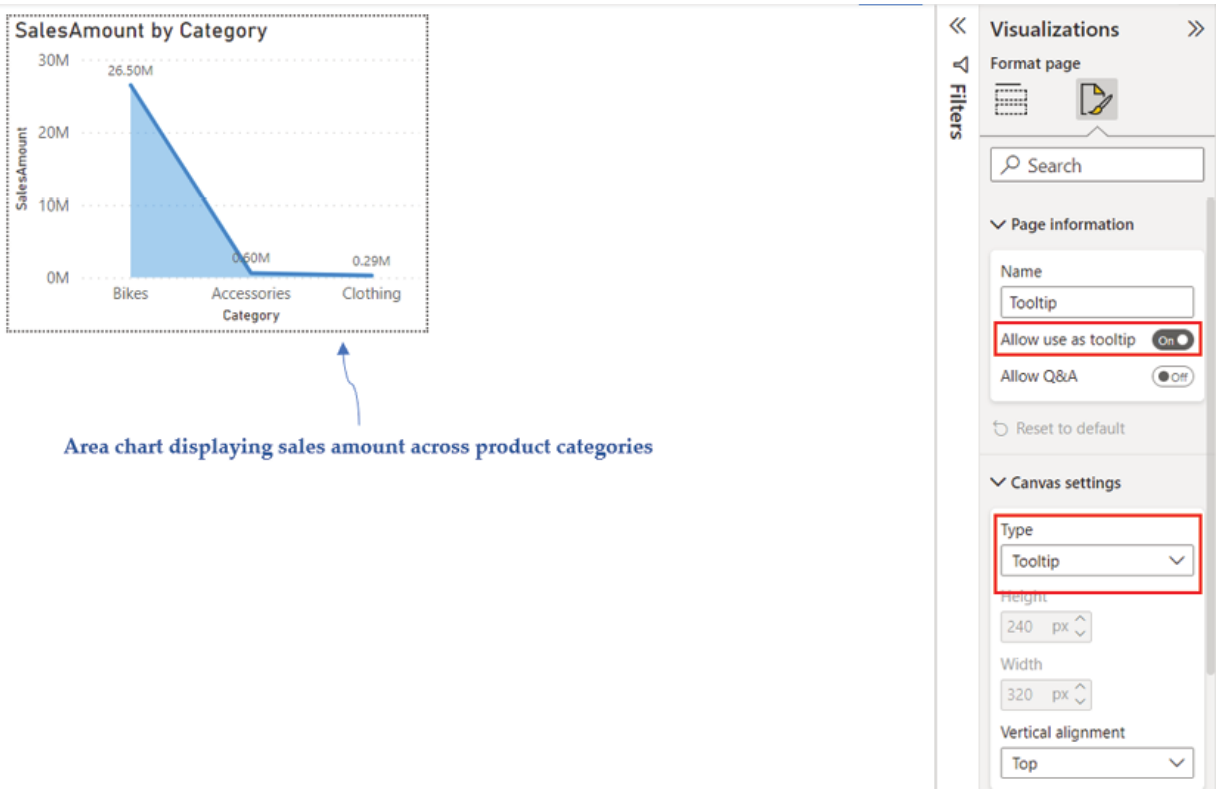



Figure 4.46: Creating a tooltip page

As we want to use this tooltip on the Map visual, we need to go back to the Overview page and select the Then, following Format visual | General | Tooltips | the Type needs to be selected as Report page and the page name needs to be selected as well, which in this case, would be Once done, then hovering over the map bubbles should display the new tooltip that we have just created instead of the default one, as shown in Figure

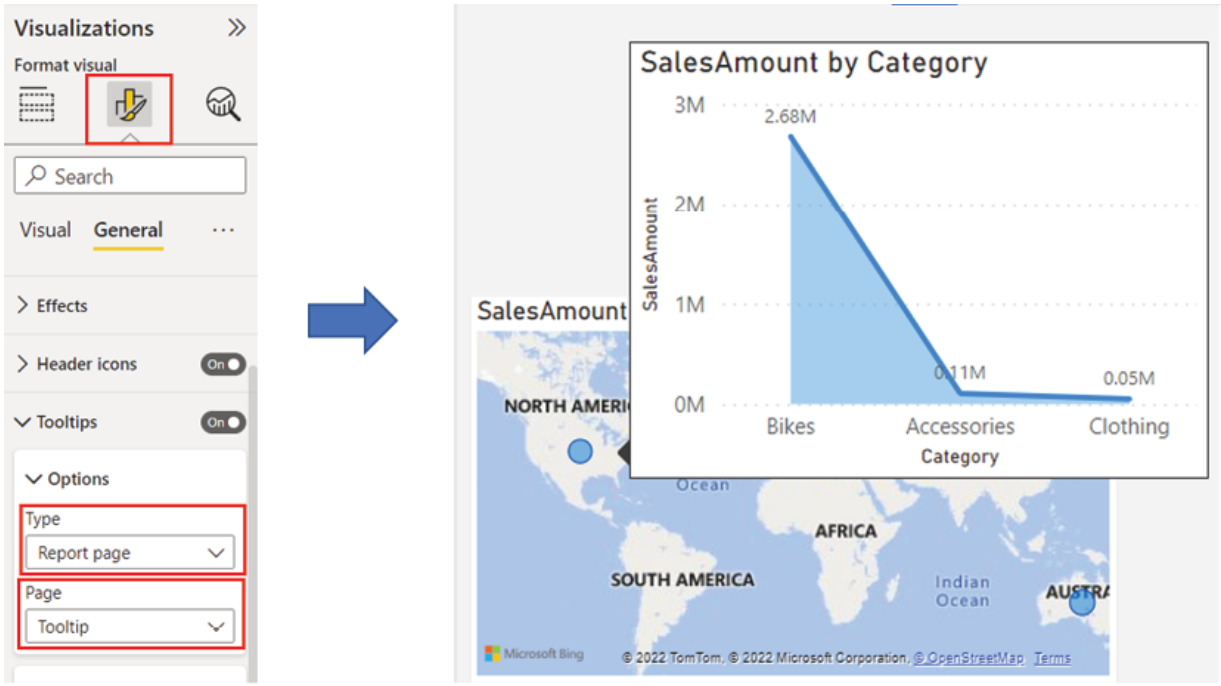



Figure 4.47: Displaying report tooltips

If we compare the values with the default tooltip, it can be seen that the values exactly match for the United combining all categories on the custom tooltip.

Let us now explore how to effectively use the report as Power BI reports are highly interactive. We shall also look at an option for controlling the behaviors during interactions.

Visual interactions in Power BI

When we say Power BI reports are interactive, what that essentially means is, visuals on a report page cross-filter or cross-highlight other visuals on the same page by default. As an example, Figure 4.48 illustrates the behavior for the year 2018 (as selected on the Year slicer), when we click on the bubble for the United States on the Map visual:

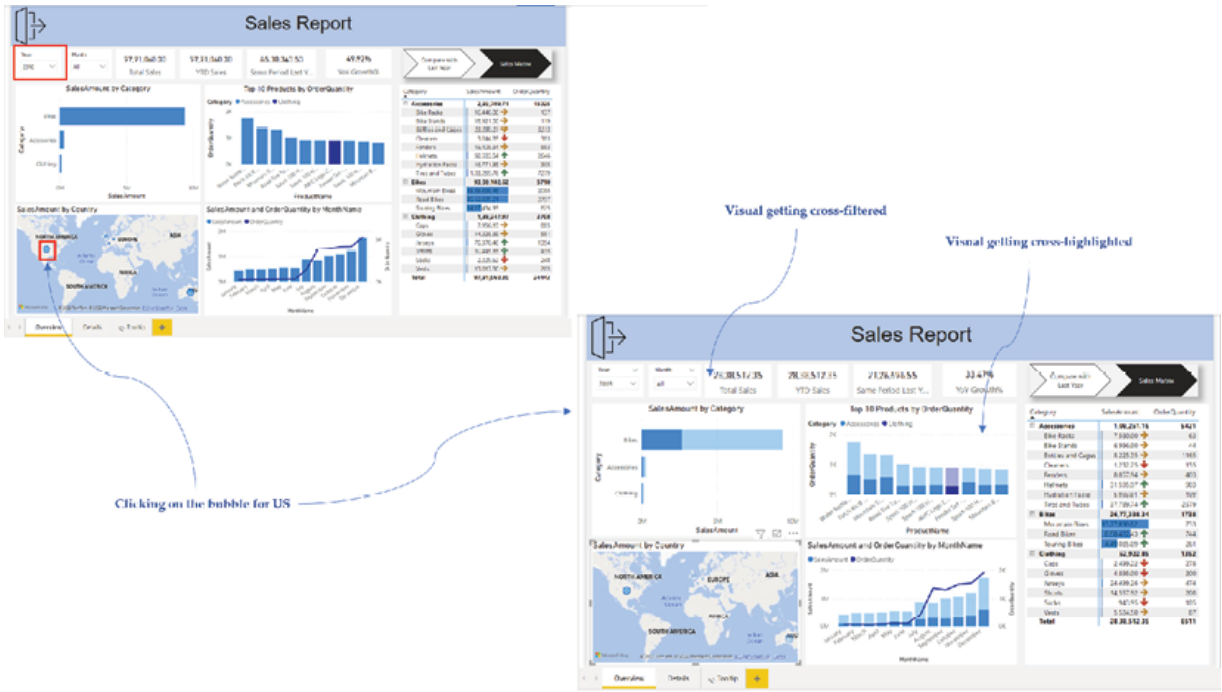



Figure 4.48: Cross-highlighting and cross-filtering visuals

This behavior can be controlled for any visual, using the Edit interactions feature which can be accessed after selecting the visual and following Format | Edit as shown in Figure

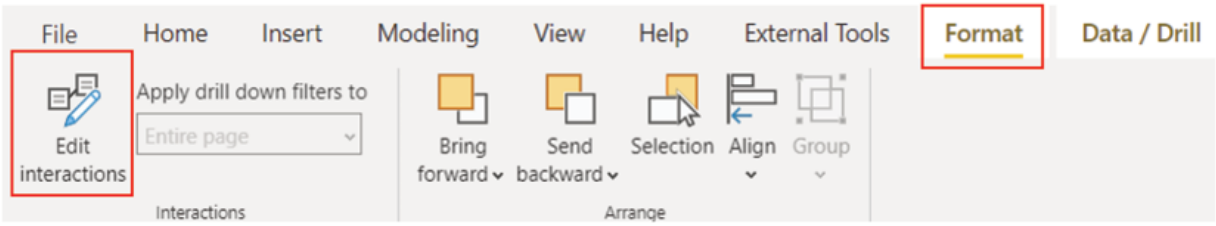



Figure 4.49: Edit interactions option

For example, in case we want to change the interactions for the Map with other visuals in the report, after selecting clicking on Edit interactions should allow us to see the existing interactions as well as update them. The existing interactions of all the visuals with the Map is shown in Figure

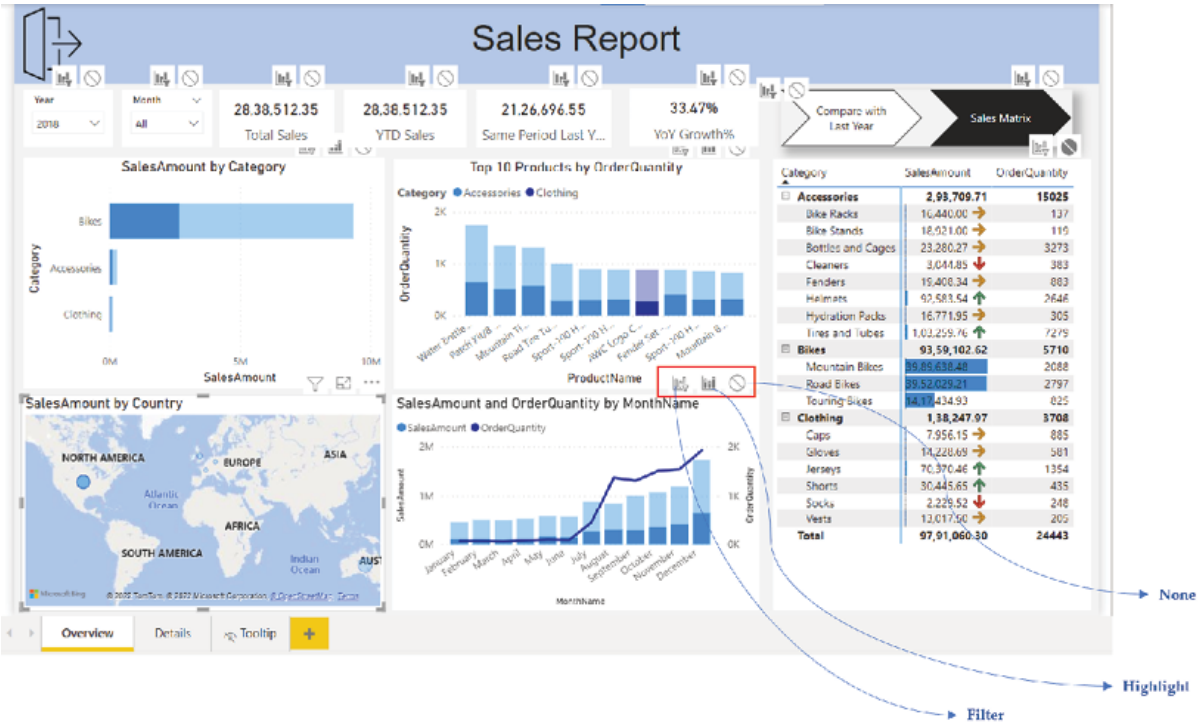



Figure 4.50: Existing interactions of all visuals to the Map

As seen in the preceding figure, there are three available interaction options for the trend chart, while for a few other specific visual types, there are only two. The options are:

This can be used to disable the interaction. Once selected, for our example, the trend chart would not interact with the Map anymore.

This can be used to cross-highlight the visual. As this is selected by default; the trend chart is getting cross-highlighted by the

This can be used to cross-filter the visual. In our example, the Matrix and the getting cross-filtered by the Map by default.

If we update the interaction on the trend chart as Filter with respect to the then if we select the bubble for the United States on the Map again, it will filter the trend chart, instead of cross-highlighting it, as shown in Figure


Figure 4.51: Highlight vs. Filter

To turn off editing interactions, it needs to be clicked again. Edit interactions can be extremely useful to control how a visual would respond

to other visuals, using only the user interface instead of writing complicated DAX expressions.

After all the work we have done so far, the report is now complete and ready to be published!

To refresh the report with updated data, the Refresh button under the Home tab can be clicked. Before publishing, let us check how the report actually performs using a tool integrated with Power BI Desktop itself.

## Analyzing report performance

Using the Performance analyzer tool in Power BI Desktop, the performance of the report and the individual elements can be measured. To display the Performance analyzer pane, View | Performance analyzer can be followed as shown in Figure

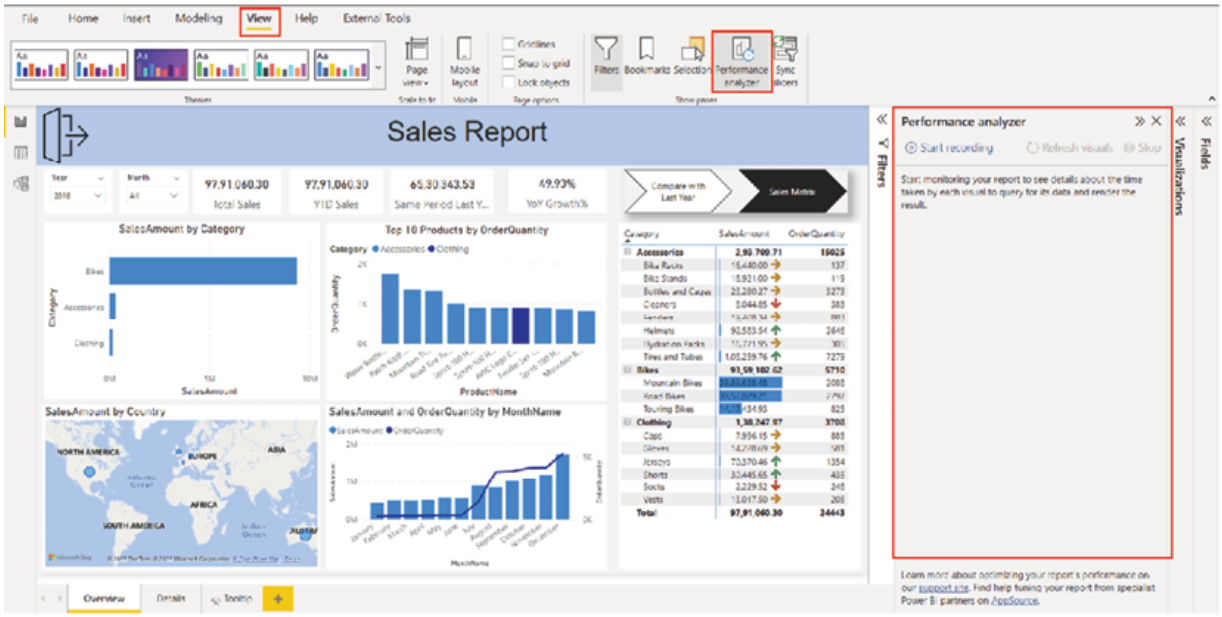



Figure 4.52: Performance analyzer in Power BI Desktop

A Performance analyzer can be used to measure the processing time for the entire report page, as well as for individual visuals. Apart from that it can record user interactions, such as changing a Slicer selection and measuring the processing time for the report page in the new context.

To begin recording, the Start recording button needs to be clicked on the Performance analyzer pane and then any interaction with the report gets

recorded, with the duration time for each operation to complete. Once the recording is started, the Refresh visuals button can be clicked to refresh all visuals on the report page and measure corresponding durations, while the Stop button can be used to stop the recording.

Figure 4.53 displays the performance for our sales report Overview page using the Refresh visuals option:

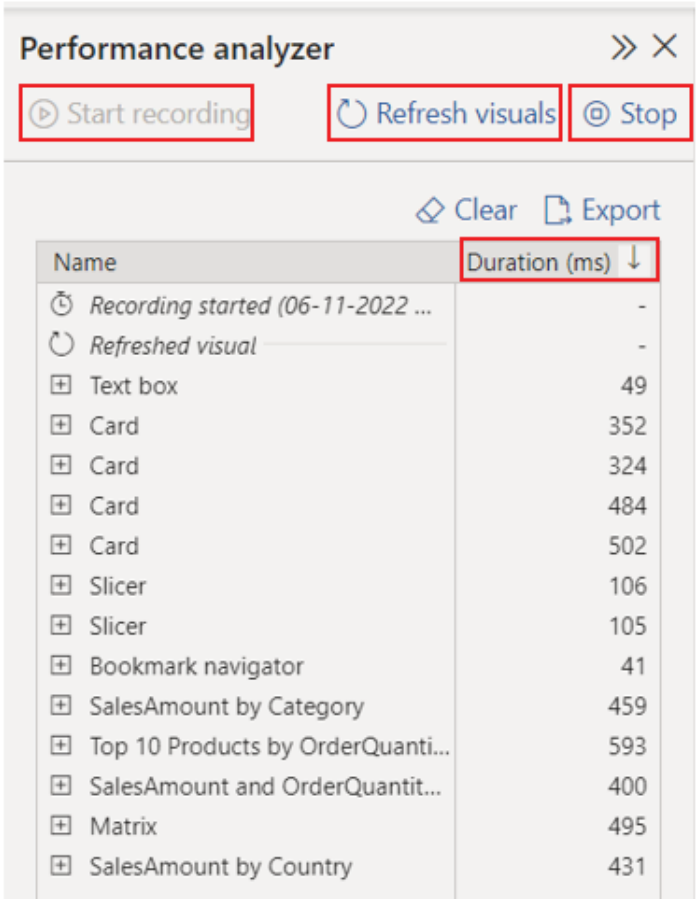



Figure 4.53: Performance analyzer log

While the Performance analyzer is running, hovering over any visual displays the option Analyze this which can be used to measure performance for that specific visual or element as shown in Figure

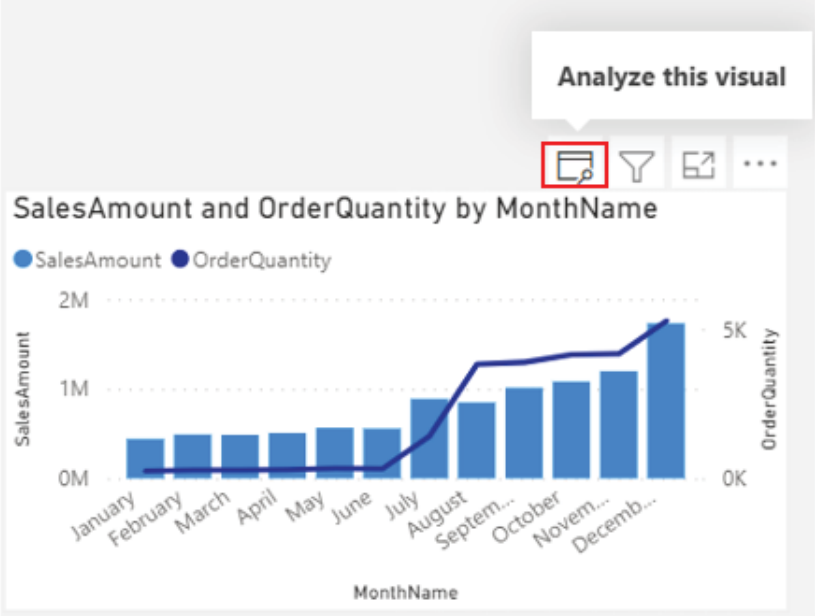



Figure 4.54: Analyzing individual visual

If we expand the elements under the Performance analyzer pane, we can see different categories for each element like DAX Visual display etcetera. There is an option to copy the DAX query which is getting triggered to the underlying dataset to render the visual as well. The DAX query can be analyzed further using external tools like DAX Figure 4.55 shows the options available for each element in the Performance analyzer pane:

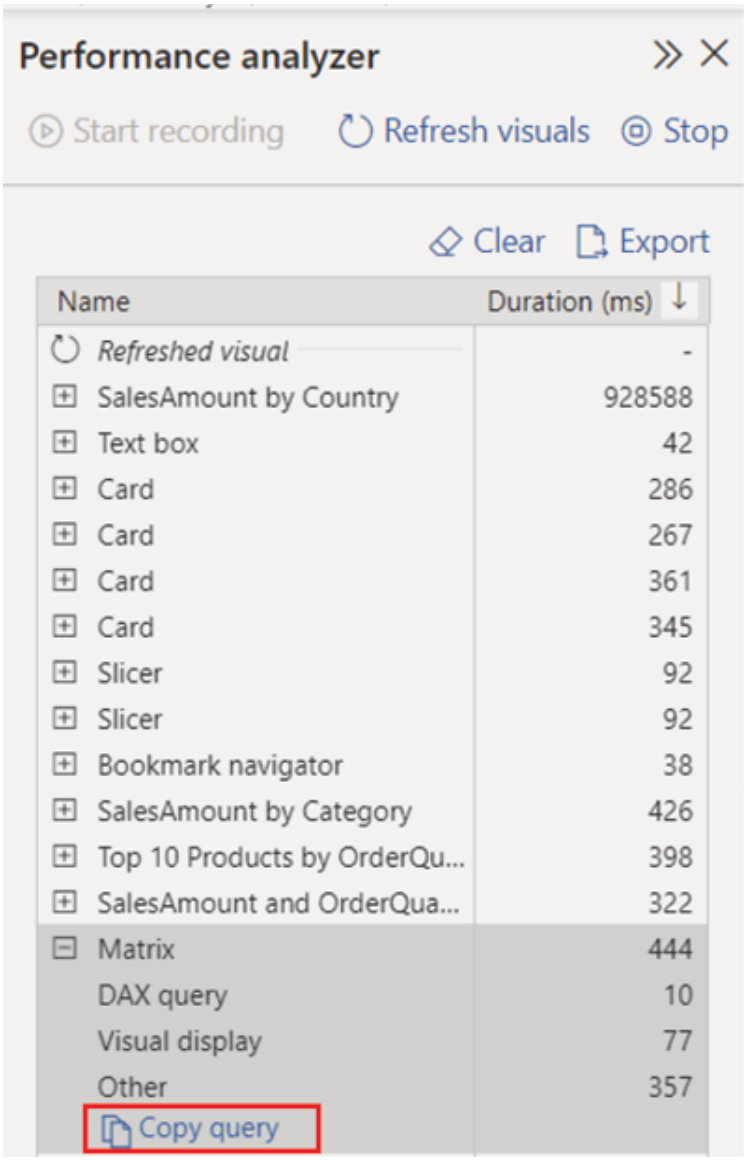



Figure 4.55: Available categories for each element in Performance analyzer along with the Copy query option

Performance analyzer can be an extremely useful tool for identifying the visuals that take a long time to load, or time-consuming operations, which can then be optimized for performance.

## Conclusion

In this chapter, we saw how to create visualizations in Power BI besides thoroughly exploring a number of useful visuals that are available. We explored the AppSource and saw how to import a custom visual to Power BI Desktop if required. Concepts like bookmarking were discussed in detail which will help to make optimum use of the report real-estate. We also saw how to enhance the reports using tooltips, how to control interactions between visuals, and even how to measure the report performance. Along the way, we created a fully functional report as a project, ready to be published! This should give readers enough confidence to start building their own reports, and further explore the diverse visualization options that they have with Power BI.

In the next chapter, we will explore the final component of the framework, which is Power BI Service, and learn how to manage the reports and collaborate with others, all in the Azure cloud.

## Knowledge check

What is the preferred out-of-the-box visual in Power BI for trend analysis?

Area chart

Donut chart

Slicer

Line and clustered column chart

What is the preferred out-of-the-box visual in Power BI for displaying a single scalar value?

Matrix

Slicer

Card

Python visual

The Bookmarks pane can help to capture the state of a report:

True

False

By default, Power BI visuals are interactive:

True

False

All Knowledge Check answers are provided at the end of the book.

# CHAPTER 5

# Managing Reports in Power BI Service

## Introduction

After discussing integration, data transformation, data modeling and visualization, now we are about to discuss the final component of Power BI which is the Power BI Service. So far, we have done all the work in the Power BI Desktop, which is a local application installed on our respective workstations. Now is the time to publish the report and share it with others. One of the key features of Power BI is easy collaboration, and there can hardly be a better way to be able to collaborate online using a cloud-based platform, which is centrally hosted, and subscription-based. This chapter would focus on managing reports on the Power BI Service, including important topics like access control and security. We would learn how to keep our reports up to date and also would explore what data gateways are and why we need them. We will discuss concepts like when using Dataflows makes sense or how Apps can provide a better user experience!

## Structure

In this chapter, we will discuss the following topics:

Workspaces in Power BI Service

Publishing reports

Working with reports on Power BI Service

Exporting data from a report

Admin portal

Workspace user roles

Managing security

Dashboards and alerts

Refreshing data and data gateways

Apps

Dataflows

## Objectives

The objective of this chapter is to familiarize readers with Power BI Service, which is a Software as a Service platform hosted on the Azure cloud. By the end of this chapter, readers should be able to take key decisions like how the reports should be distributed, which access role needs to be assigned to their report consumers, and so on. Content can be shared on Power BI Service in more than one way, and the readers will be introduced to multiple options that are available today. They should be able to schedule refreshes for the reports on their own, and if required would be able to work with data gateways as well. Apart from that, concepts like Dataflows should equip the readers with more options while taking design-related decisions for their projects. The chapter should help the readers to attain proficiency with Power BI Service and make them ready to be able to deploy an end-to-end business intelligence solution.

## Workspaces in Power BI Service

While we touched upon the Power BI Service in the introduction of the book, let us dive deeper now to understand how things work here.

The root URL of Power BI Service is where users can log in by authenticating using their organizational account. Once logged in, on the welcome page, the navigation pane can be found on the left. On the navigation all the elements of Power BI Service are listed, including workspaces, as shown in Figure

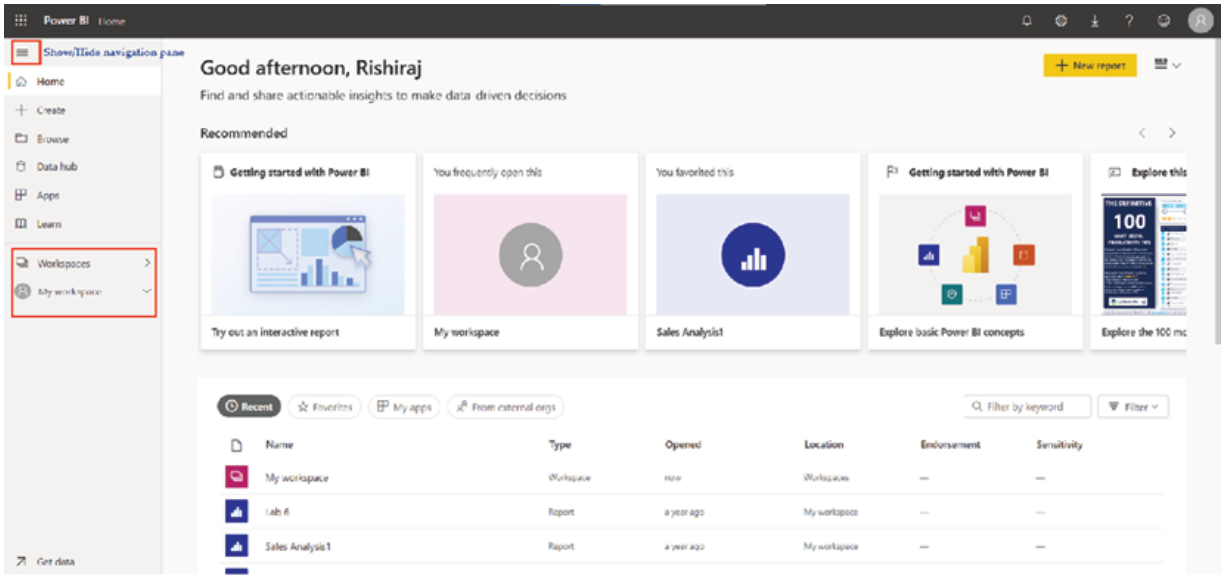



Figure 5.1: Power BI Service home page

Workspaces are containers for artefacts like reports, datasets, dashboards, dataflows, etc. By default, workspaces are created on shared capacity. There are two types of workspaces, My workspace and workspace (previously known as App workspace).

My This is a personal playground or sandbox for any Power BI user, be it a free user or a pro user. Every Power BI user account would have a My workspace by default.

Workspaces are for collaboration and sharing content with others. To create a Workspace, either a Power BI Pro or a Premium Per User license is required. To create a go to Workspaces | Create a workspace as shown in Figure

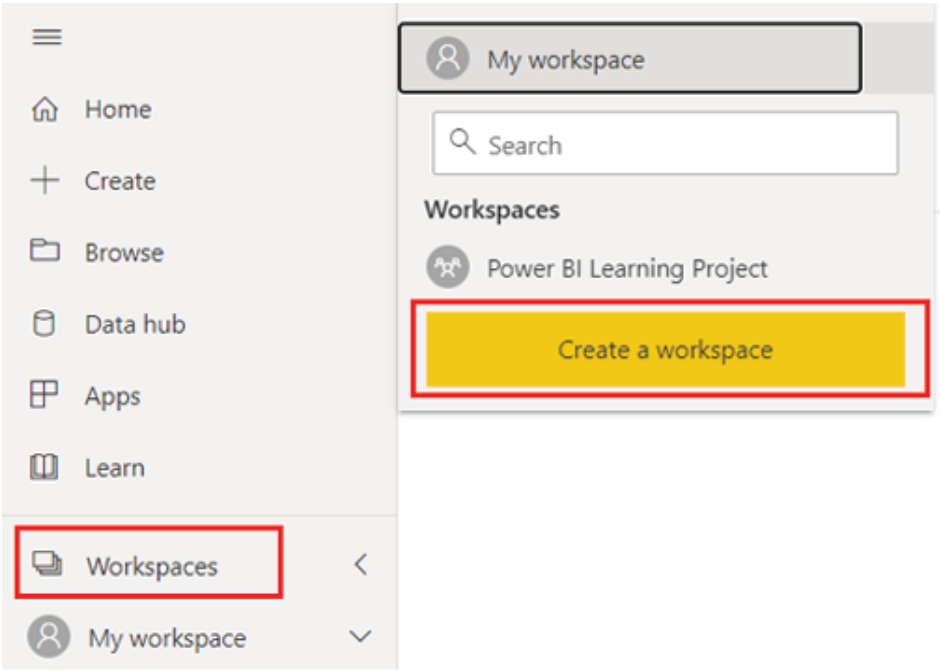



Figure Creating a workspace

A free user should be prompted with the Upgrade to Pro message while trying to create a workspace. Alternatively, a free user can opt for a free 60-day individual trial period, to explore and experience the paid features.

For a Pro or PPU user once clicked on the Create a workspace button, a window will open requesting inputs like Workspace name and Description (optional). Once provided, the workspace will be created, as shown in Figure

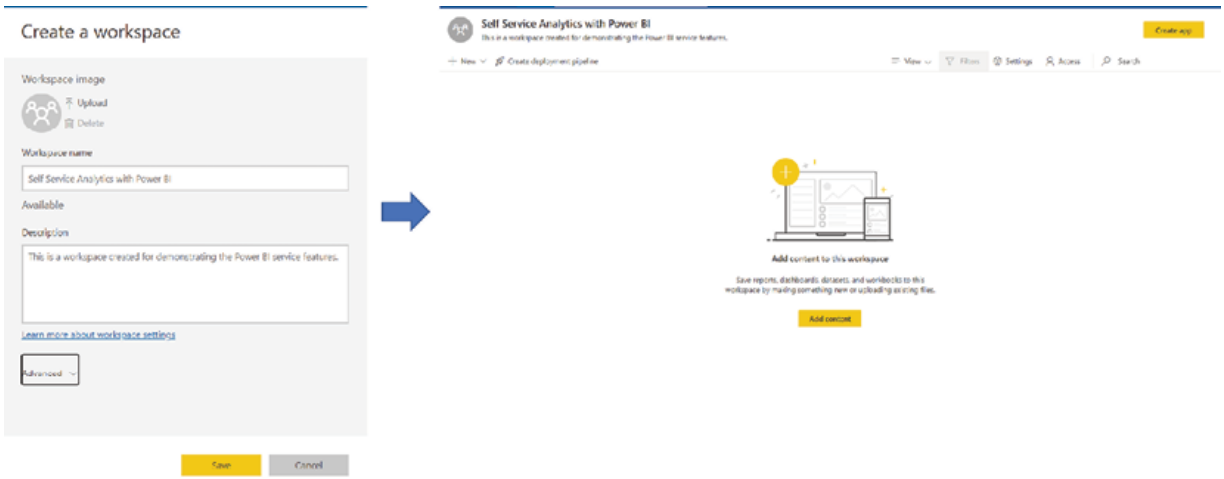



Figure Power BI workspace

Once created, the workspace would become ready to host all the published Power BI content.

## Publishing reports

Let us now discuss how to publish reports to the workspace we just created. Going back to the Sales Report in the Power BI Desktop, if already not signed in, the report author should sign in using the same organizational account used for Power BI Service. Figure 5.4 shows the Sign in option in the Power BI Desktop:

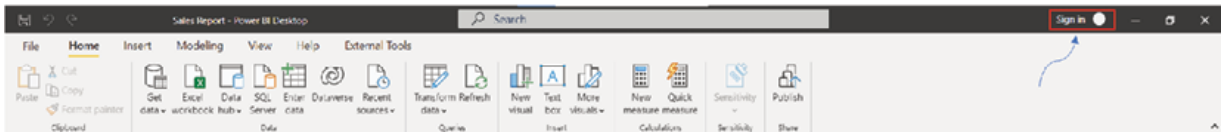



Figure Signing-in to Power BI from the desktop version

Once signed in, clicking on the Publish button under the Home tab would open the Publish to Power BI window. All the Power BI workspaces that the user has access to would be displayed. We can search for workspaces that we wish to host our report and click on which should publish the report with a success message. Figure 5.5 illustrates the process of publishing the Sales Report to the workspace Self Service Analytics with Power

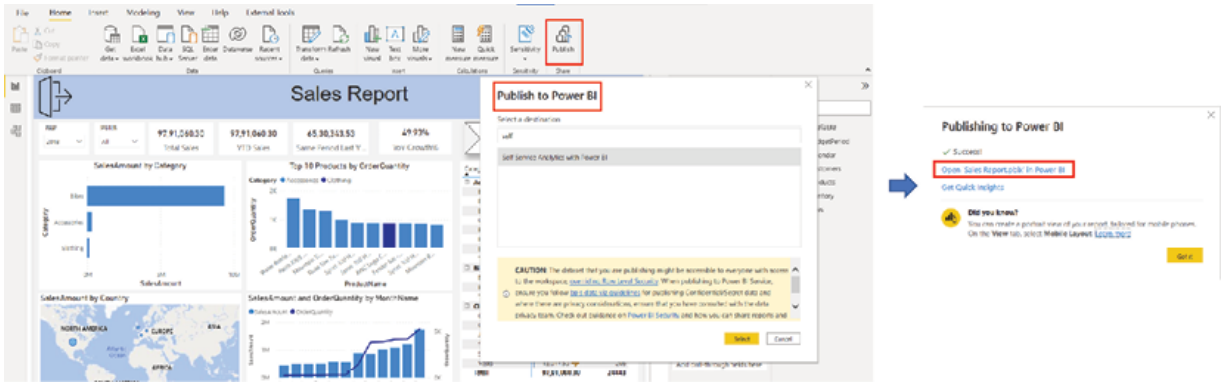


Figure Publishing report from Power BI Desktop to Power BI Service

The success message contains a direct link to the report which can be utilized, or one can navigate to the workspace itself from Power BI Service to access the published content, as shown in Figure

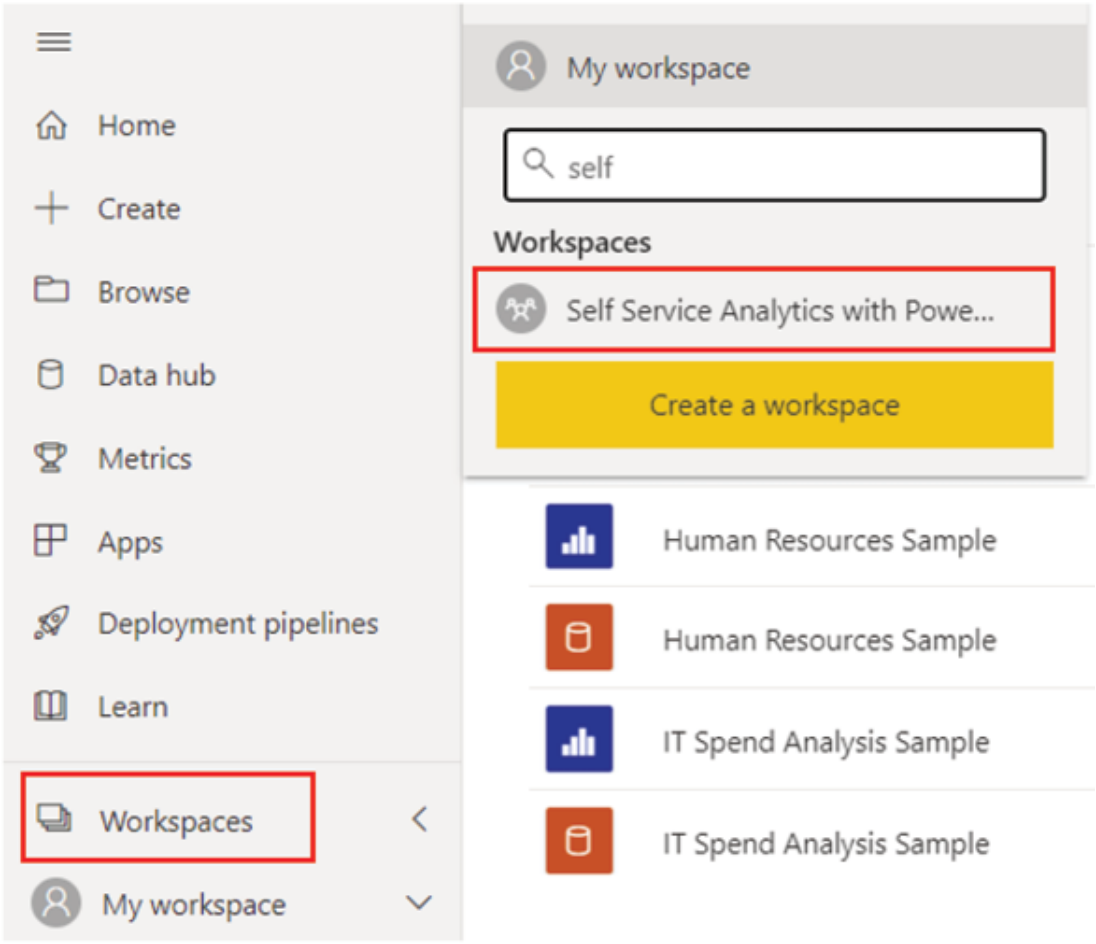



Figure 5.6: Navigating to a workspace in Power BI Service

Note: Only a Pro or PPU user would be able to publish content to or consume content from, a workspace other than My workspace. For a free user to consume content, the workspace hosting the content must be backed by premium capacity. All the concepts discussed involving Power BI Service from now on would be valid only for a paid license.

The Power BI Desktop or .pbix file contains both the definition of the Power BI model, as well as the report connected to it. However, once published to the service, the file is split up and the Power BI model/dataset and report get stored as separate entities in the workspace.

Once entered in the workspace, all the published elements are visible under the All tab. The Content tab would have a list of all the reports and dashboards, while all the Power BI models and dataflows would be visible under the Datasets + dataflows tab, as shown in Figure

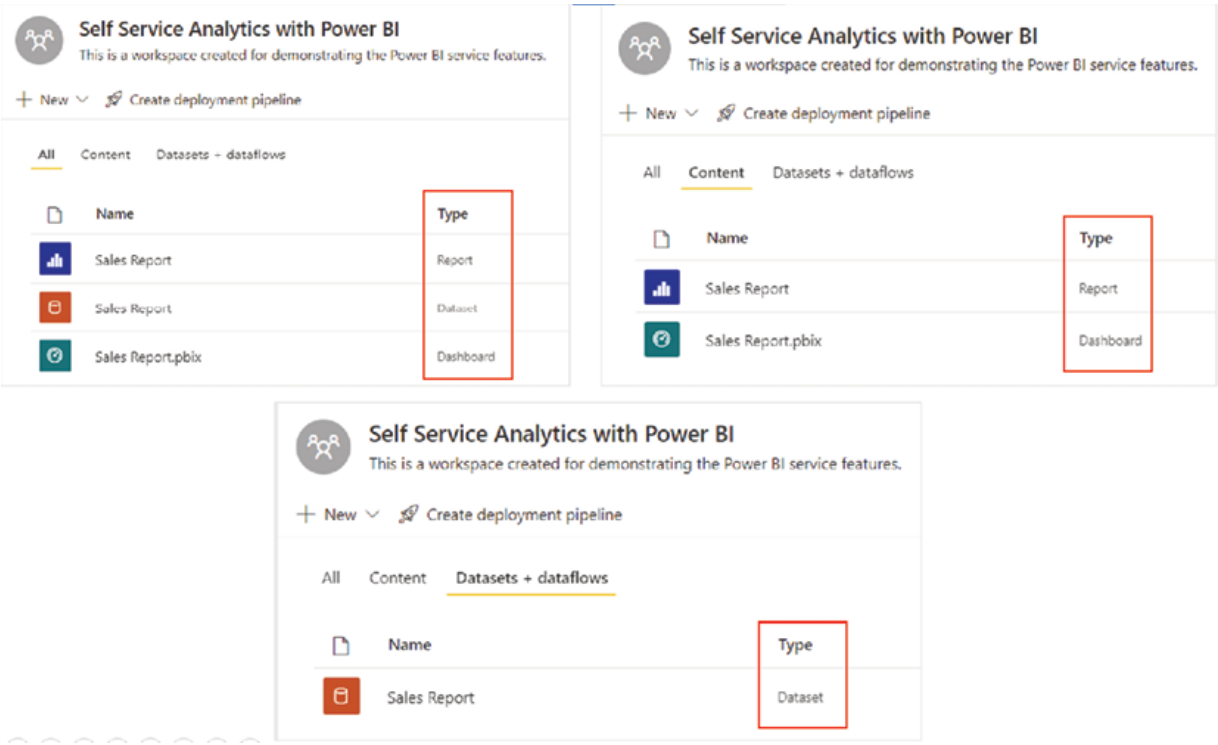



Figure Different tabs in a workspace

It can be observed that whenever a new report is published in the service, a blank dashboard gets created by default with the same name as the report, where visuals from individual reports can be pinned; we will discuss more about the dashboard later in the chapter.

Each of these elements or artefacts have additional options or properties that can be configured by hovering over and clicking on the ellipsis or More options towards the right side. We will explore a few of those options in detail as we move forward.

## Working with reports on Power BI Service

From the workspace, we can open the report by simply clicking on it. Different report pages are displayed by default on the left-hand side Pages pane. The report will open in the same state it has been saved in the Power BI Desktop before publishing and will be fully functional as configured. On the right-hand side of the report, the Filters pane would also be visible, if that is not explicitly hidden on the Power BI Desktop before publishing. Figure 5.8 shows how the report looks on the service:

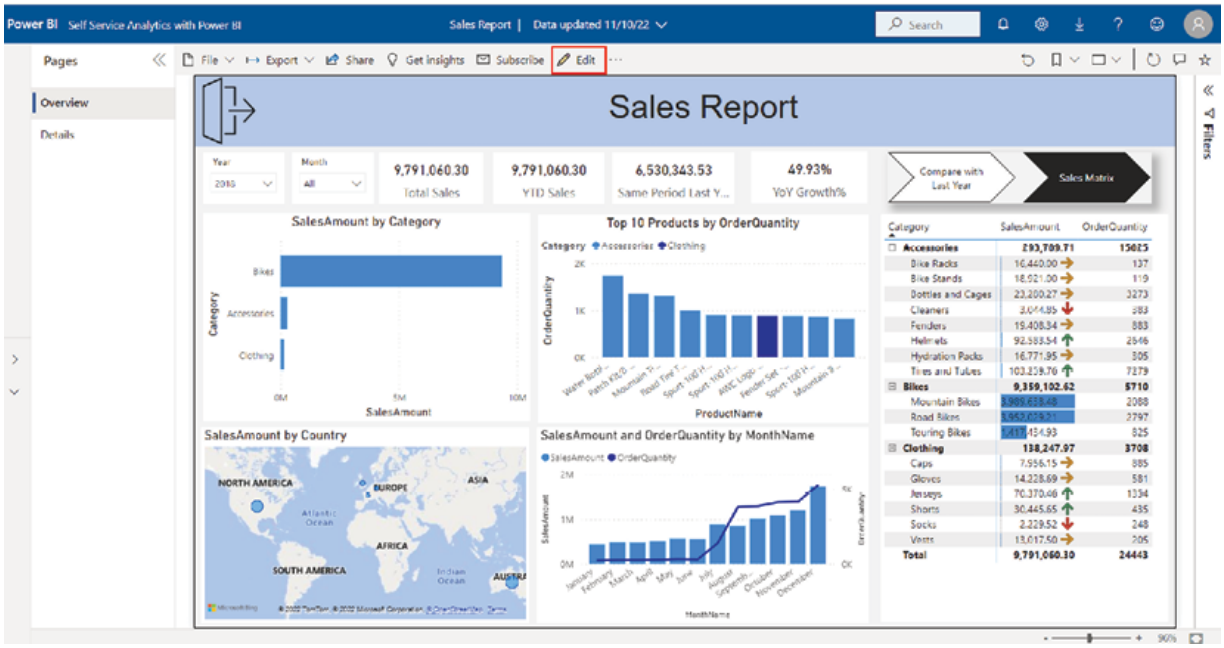



Figure Sales report on Power BI Service

Besides accessing the report, workspace users having edit permission are also allowed to edit the report online using the Edit button on the top of the report, as highlighted in Figure Clicking on the Edit button opens the Fields pane and the Visualizations pane for the users, and all the underlying

tables and columns get exposed. However, users can only interact with the fields and use them for visualization; it is not allowed to edit or update the data model in any way from the report, as of now. Figure 5.9 illustrates the editing view of the report:

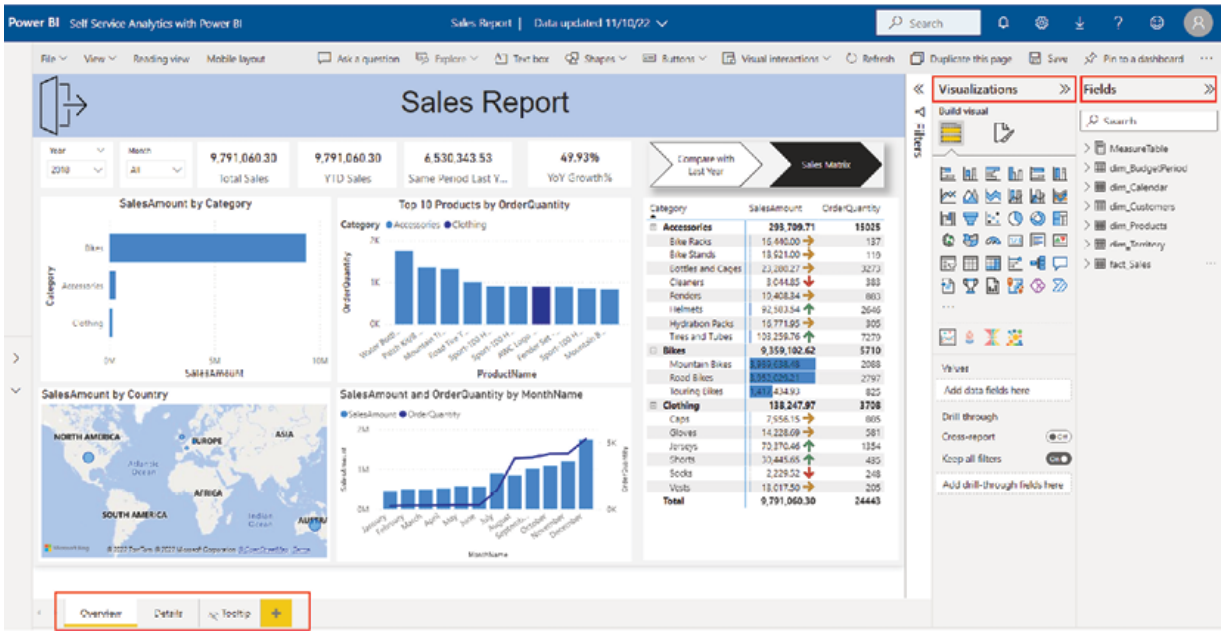



Figure Editing view of Sales report on Power BI Service

The key functionalities that can be performed from the editing view include adding or deleting visualizations, formatting visualizations, adding or deleting report pages, updating filters using the Filters pane, etc. The list of capabilities that can be performed online is getting extended with almost every new update of Power BI Service, however, the Power BI Desktop is still usually preferred as the primary development environment as it provides better flexibility and overall experience.

After performing any change on the editing view, the report can be saved using the Save button and we can exit the editing view by selecting the Reading as shown in Figure

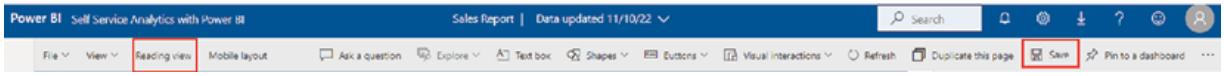



Figure Saving changes on editing view and going back to the Reading view

The Power BI Desktop file can be downloaded from the reading view of a report by following File | Download this file option, by any user who has edit permission of the report. However, there are a few limitations associated with downloading the .pbix file. For example, reports that are originally created on Power BI Service cannot be downloaded, and datasets configured with incremental refresh policy cannot be downloaded as well. Apart from that the ability to download, the .pbix file can also be disabled by the tenant administrator. The option for downloading the .pbix file is shown in Figure

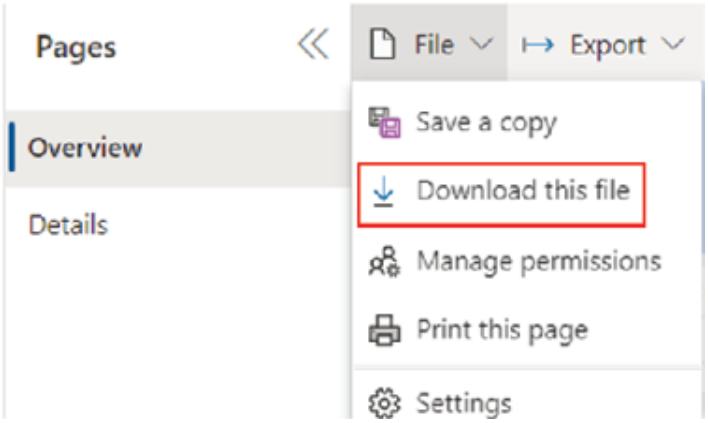



Figure Downloading the .pbix file from the published report

Once downloaded, the local copy of the report, that is, the .pbix file can be further updated for performing any additional changes required. The changes can be done on any of the layers, including the Power Query editor, the underlying data model or the report itself, and the updated file can be re-published to Power BI Service, as many times as required.

In time of republishing a file, the dataset on Power BI Service is replaced or overwritten with the updated dataset of the Power BI Desktop file only if the .pbix file has the same name as the published dataset. Otherwise, the .pbix file gets published as a new report and dataset. Figure 5.12 displays the message which gets prompted while overwriting a dataset on Power BI Service:

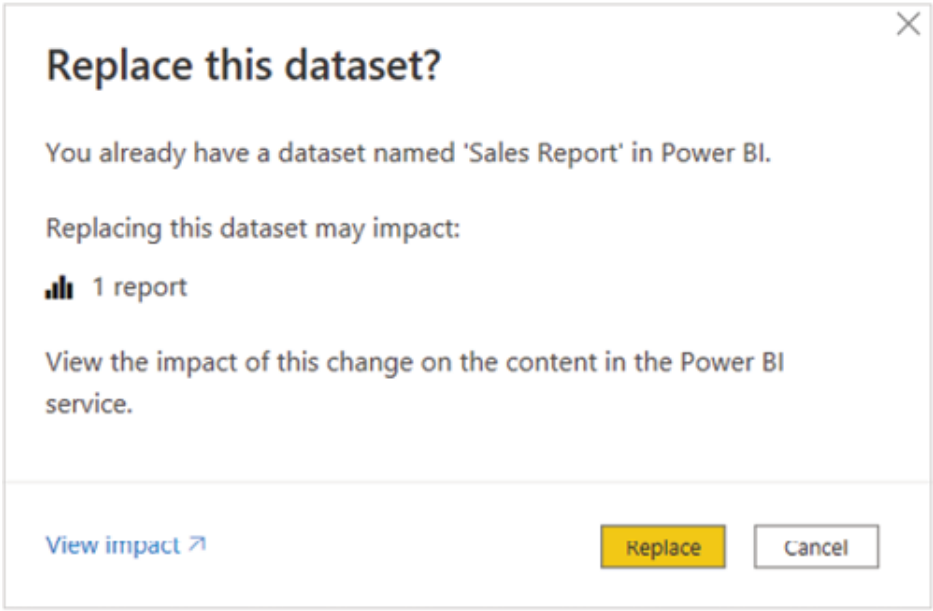



Figure Warning message while overwriting a dataset

There are many ways to share a Power BI report, and one is there right on the top of the reading view of a report, as highlighted in Figure

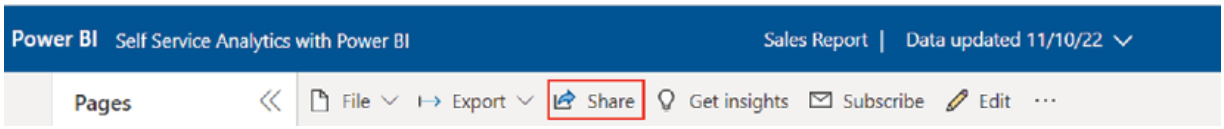



Figure Sharing a report, option 1

Once the Share button is clicked, it opens the Send link dialog which allows you to create report links and share with a selected group of people

for whom the links would work, as illustrated in Figure

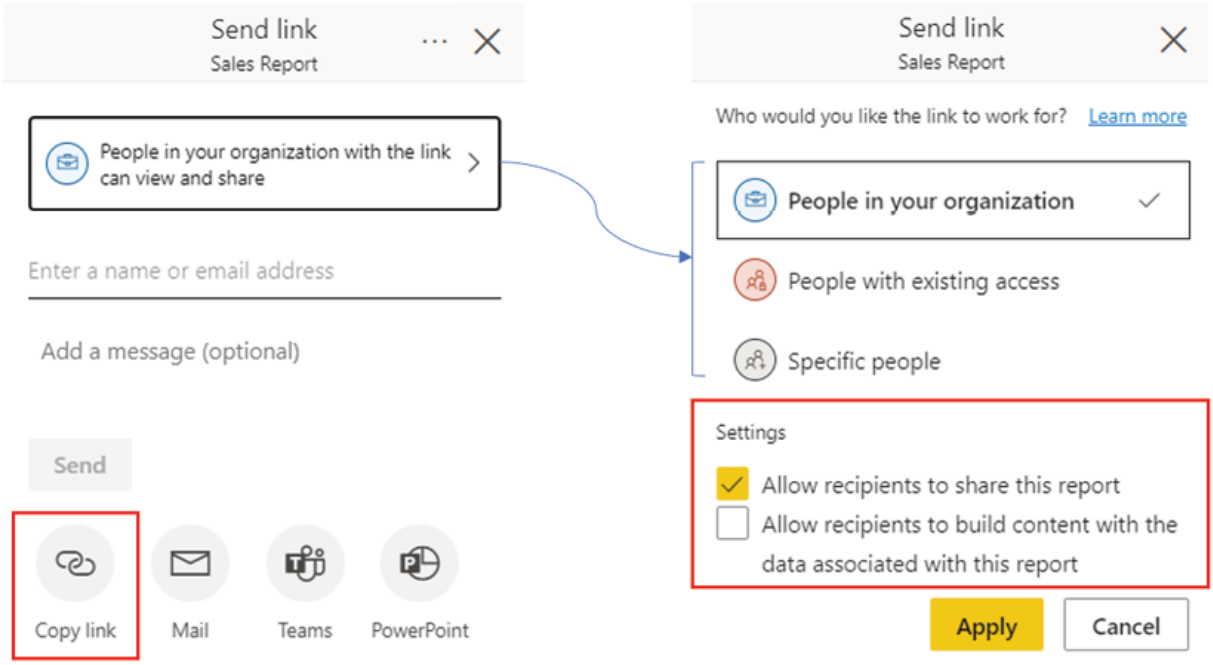



Figure Options while sharing a report

The available options for creating the links are:

People in your organization: be used to generate a shareable link for people inside the organization; the link would not work for external users or guest users. The link can be copied using the Copy link option and shared manually, or the email addresses of the intended recipients can be typed in and the Send button can be clicked. Once sent, recipients receive a mail notification with the direct report link.

People with existing access: a URL to the report, which can be accessed by the workspace users or someone who already has access.

Specific people: This allows specific individuals or groups to access the link, including guest users of the organization's Azure Active Directory Again, the link can be copied or sent using the Send button which triggers a mail notification with the link attached.

Apart from controlling who can access the link, the Send link dialog can also be used to control what further actions can be taken with the link using The recipients can be allowed to re-share the report, as well as build content with the associated data, as highlighted in Figure

To view additional information about the report, the report name can be clicked on the top ribbon, as shown in Figure

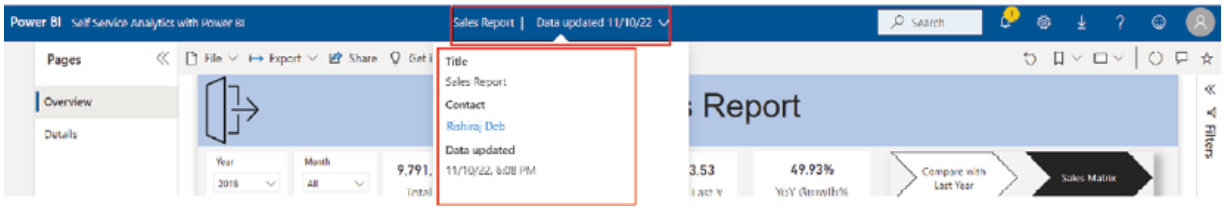



Figure Details information about the report

The detailed information includes the Data updated timestamp, which is useful in terms of understanding when the underlying dataset last refreshed.

The Contact header by default should show the person who creates or publishes the report. If someone gets a report link that they do not have access to, they can request access to the report which routes to the report contact for approval. The default report contact can be altered from the Report Settings as shown in Figure

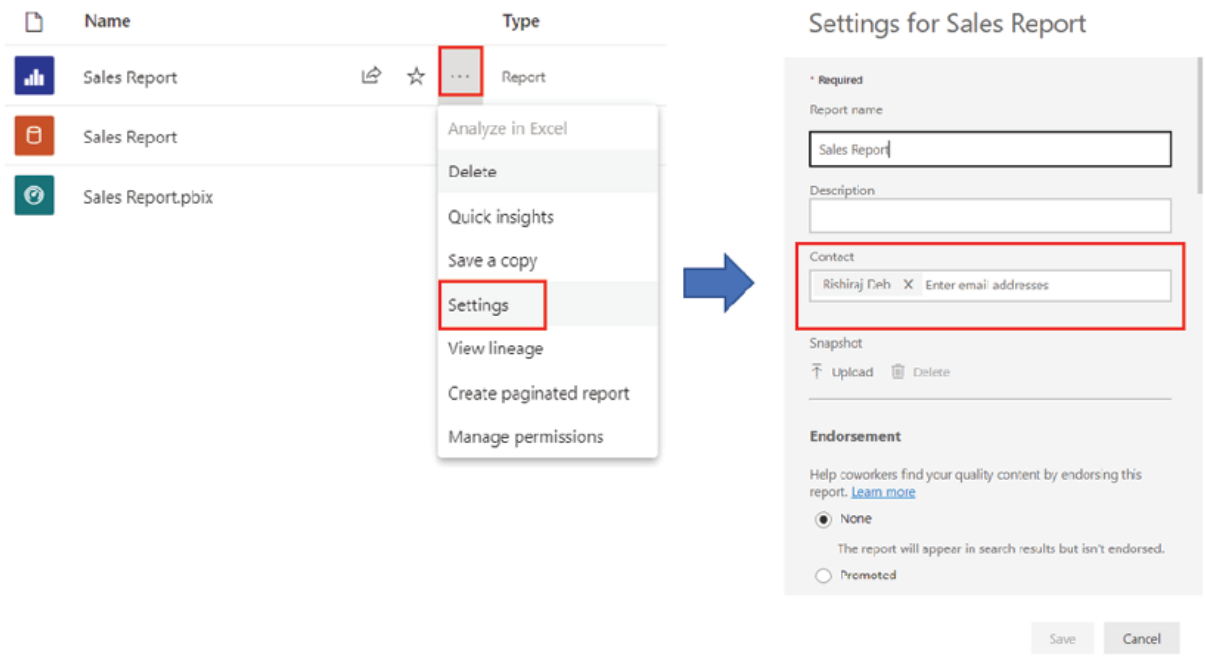



Figure Updating report contact

In case we remove all users or groups from the report contact list, then the workspace contact list is shown and in the absence of the workspace contact list, the workspace admins are shown as the report contact. We will discuss the workspace user roles in detail in the upcoming sections of this chapter. The workspace contact list can be accessed by following workspace Settings | as shown in Figure

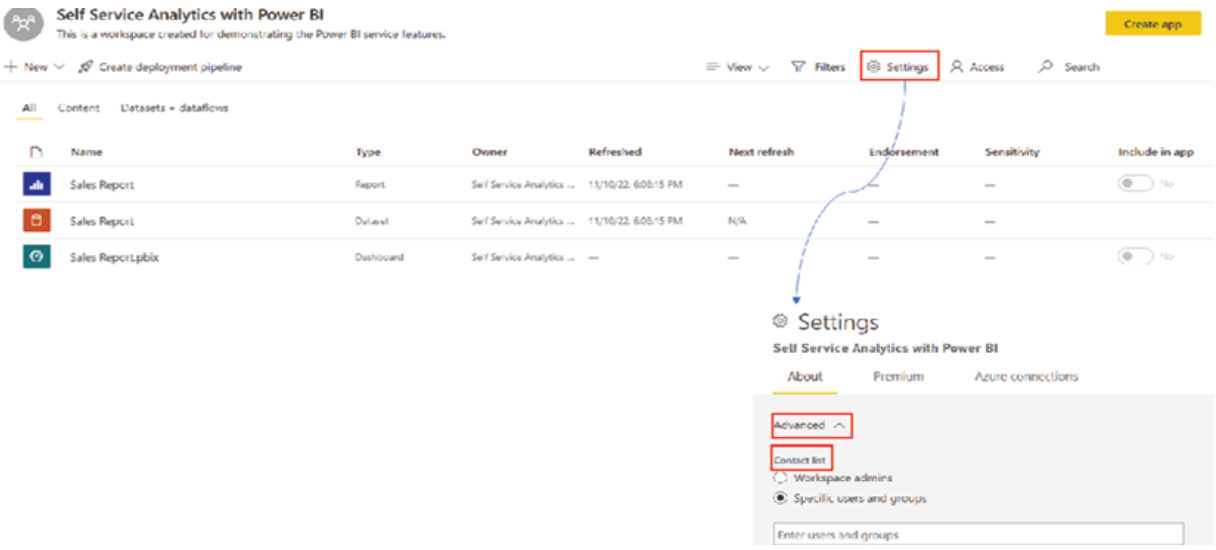


Figure Updating workspace contact

While being on the reading view of the report, to navigate back to the workspace home page, we can simply click on the workspace name, either on the top ribbon or on the left side navigation pane. The report can also be shared from the workspace's All or Content tab as the Share button becomes visible when we hover over the report, as shown in Figure

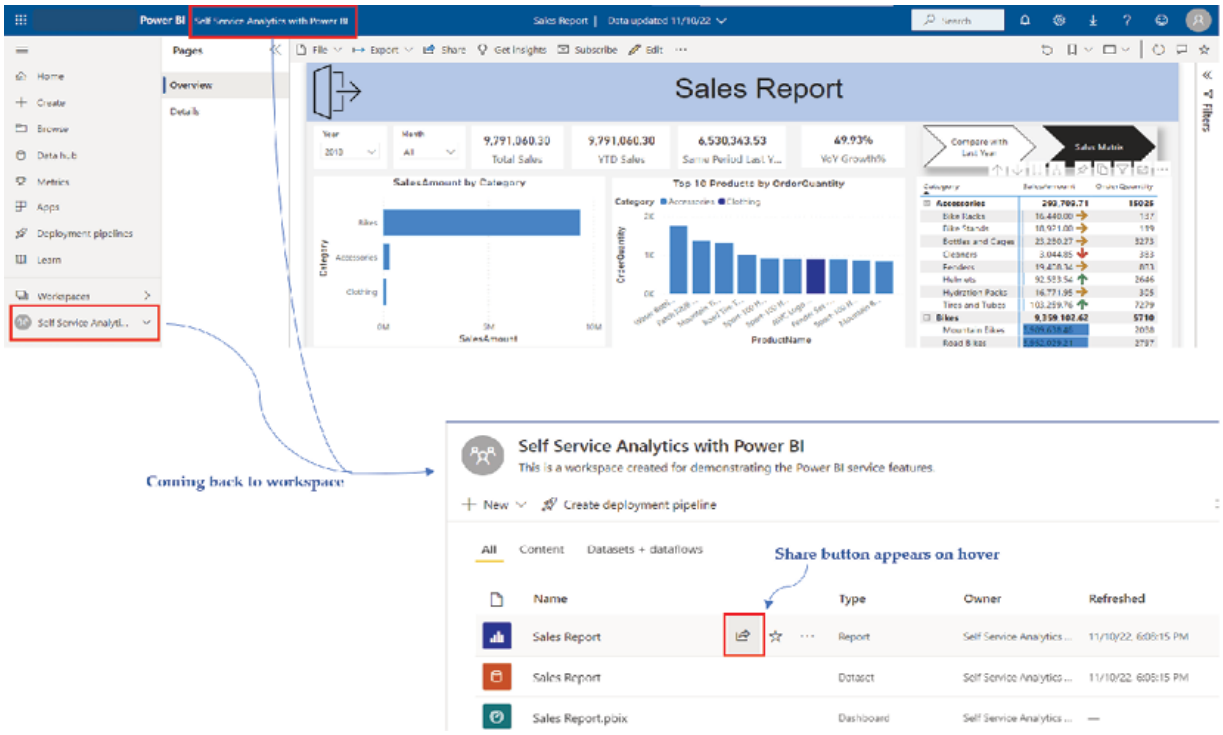



Figure Sharing a report, option 2

By now, we should be familiar enough regarding working with reports in Power BI Service, let us explore a few different ways to export data or information out of the report.

Exporting data from a report

There is more than one way to get information out of a Power BI report. While the availability of all these options can depend on the tenant settings as well as the permissions granted to the user, the most common options can be categorized as follows:

Exporting images: It is possible to export images of the entire report as well as specific visuals. To export the report itself, from the reading view of the report, go to Export | PowerPoint or Export | as shown in Figure

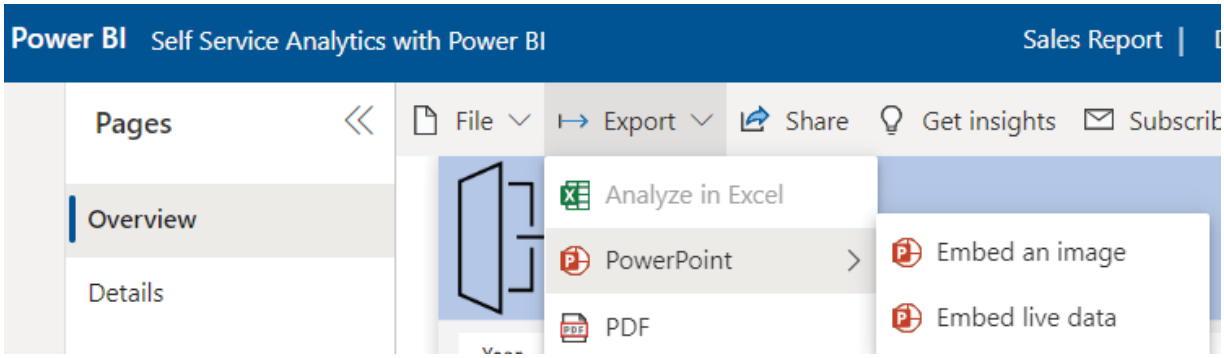



Figure Exporting report images

In case we are exporting the report to PowerPoint, we can either choose to save the report as an image snapshot or embed the live report in a PowerPoint presentation. For embedding live data, users who have access to the Power BI report would be able to interact with the report from inside PowerPoint. Different report page gets exported as different PowerPoint presentation pages. The different behaviours and options while exporting a report to PowerPoint are illustrated in Figure

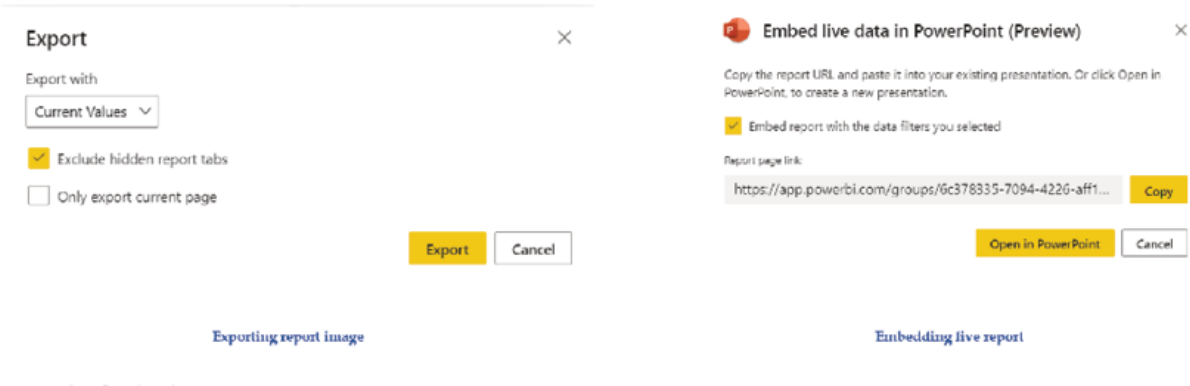



Figure Exporting report to PowerPoint

In the case of exporting the report to PDF, again the report snapshot gets exported in PDF format, with different report pages.

Apart from exporting the entire report, individual visuals can also be copied to a clipboard from Power BI Service using the Copy as image with caption option, and pasted to an external application, as shown in Figure

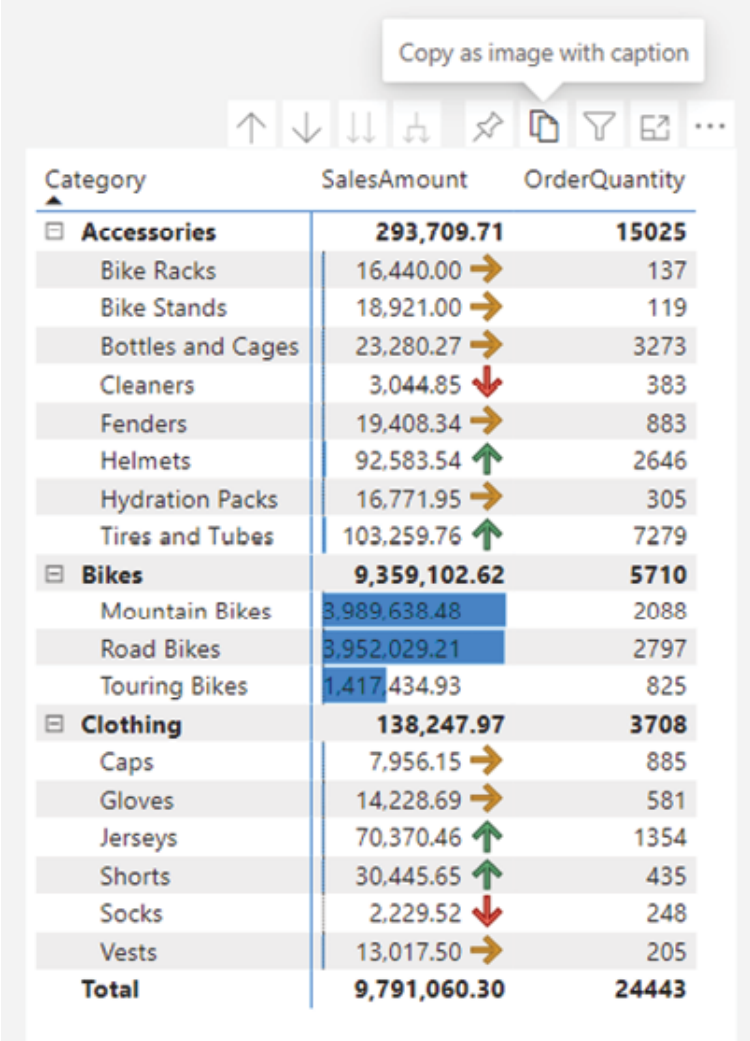



Figure Copying an individual visual

Note: While exporting image snapshots, visuals with scrollbars are exported in their default state and will display all possible rows starting from the first. Any attempt to scroll down the rows or axis would not work and the visual should retain the original state.

Exporting data from visuals: Power BI Service also allows users to export the summarized or underlying data from visuals. The available options for the end users can be controlled from the Power BI Desktop file, by following File | Options and settings | Options | Report settings (CURRENT The Export data section has options to control what would be

available to the users when they want to export the data. For our sales report, we allowed the users to export both the summarized and underlying data, as shown in Figure

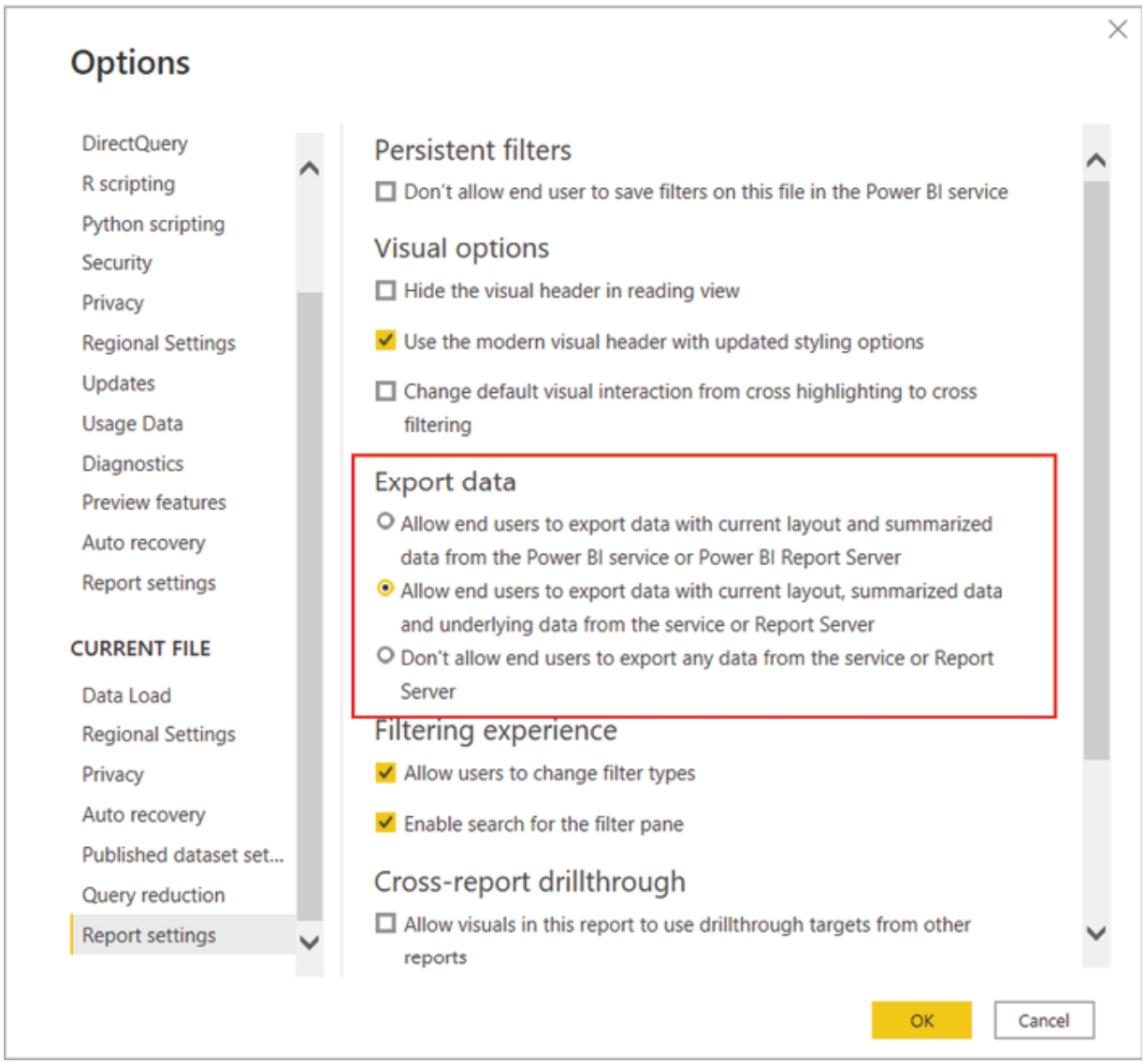



Figure Export data configuration options in Power BI Desktop

Back to Power BI Service, data can be exported from a visual by clicking on More options or the ellipsis icon. We can export Data with the current which is applicable only for table and matrix visuals, and retains the layout as created in Power BI after exporting it to an Excel file having the .xlsx format. Summarized data can be exported to get the aggregated data in the

context of the visual, in either an Excel or a CSV file. The Underlying data option allows you to export the raw data that is used to calculate the data of the visual to an Excel file.

Different options for exporting data are illustrated in Figure

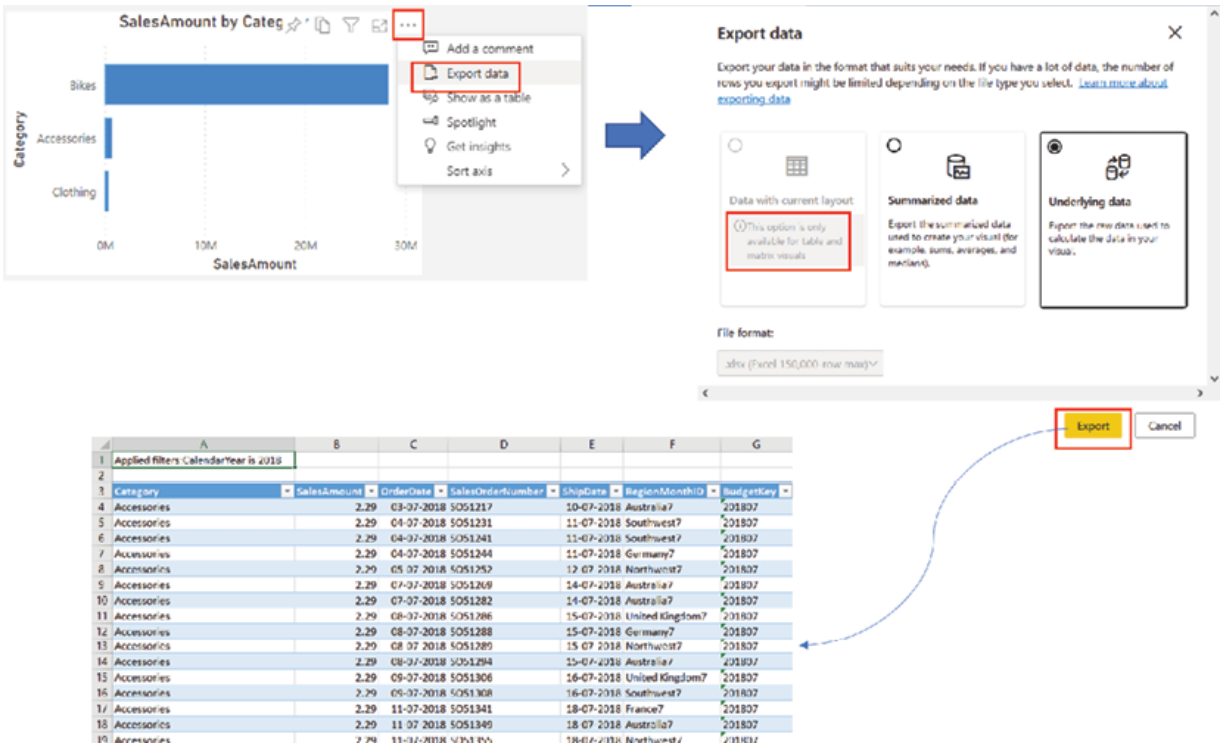



Figure 5.23: Different export data options in Power BI Service

In case of Excel, the maximum number of records allowed to export is 1,50,000 while for CSV the limit is 30,000 as of now. For a report created using DirectQuery, the maximum amount of data that can be exported is 16 MB of uncompressed data.

Connecting live to the underlying dataset: Other than exporting data, Power BI Service also allows external applications like Excel to connect live to the underlying dataset, or directly create a live connected Excel file

from the report using the Analyze in Excel feature. We will explore this in detail in one of our use cases, later in the book.

## Admin Portal

The Admin portal allows to govern Power BI for an entire organization or tenant. To administer Power BI from an organizational perspective, either Power BI admin or Power Platform admin or Microsoft 365 global admin role is required. Microsoft 365 user management administrators can assign the Power BI admin or Power Platform admin roles to the users, using Microsoft 365 admin or by using PowerShell scripts.

The Admin portal can be accessed by following Settings | Admin from the Power BI Service home page as shown in Figure

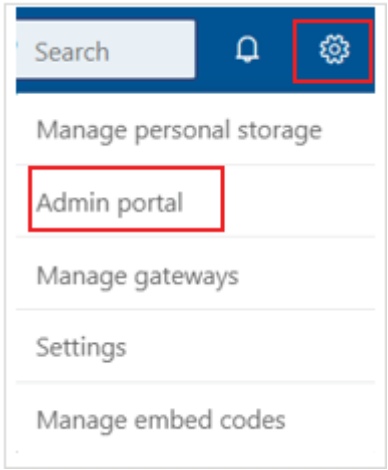



Figure Navigating to the Admin portal on Power BI Service

A few important controls available in the Admin portal are as follows:

Tenant This can be used to control the features made available for the Power BI users across the organization.

Usage This can be used to view the report usage across the tenant.

Audit This can be used to track user activities in Power BI.

Capacity This can be used to manage Power BI premium capacities.

Organizational This can be used to control the types of visuals users can access.

This can be used for viewing and managing workspaces across the organization.

Custom This can be used to customize the look and feel of Power BI Service.

While users having the Power BI admin or Power Platform admin roles can use the Admin portal for governance, these roles do not have specific capabilities like modifying users and licenses. After procuring Power BI pro licenses, users can be assigned to either Microsoft 365 admin center or the Azure

In Microsoft 365 admin only Global admin or License admin or User admin roles can assign licenses. In the case of the Azure to assign licenses, the user must be an owner of the Azure subscription that Power BI uses for Azure Active Directory lookups.

Workspace user roles

User roles determine who can do what in a workspace. Roles can be assigned to individual users as well as user groups such as security groups, Microsoft 365 groups or distribution lists. Once entered in the workspace, selecting the Access button opens the Access window, where available roles can be selected and assigned to individual users or user groups, as shown in Figure

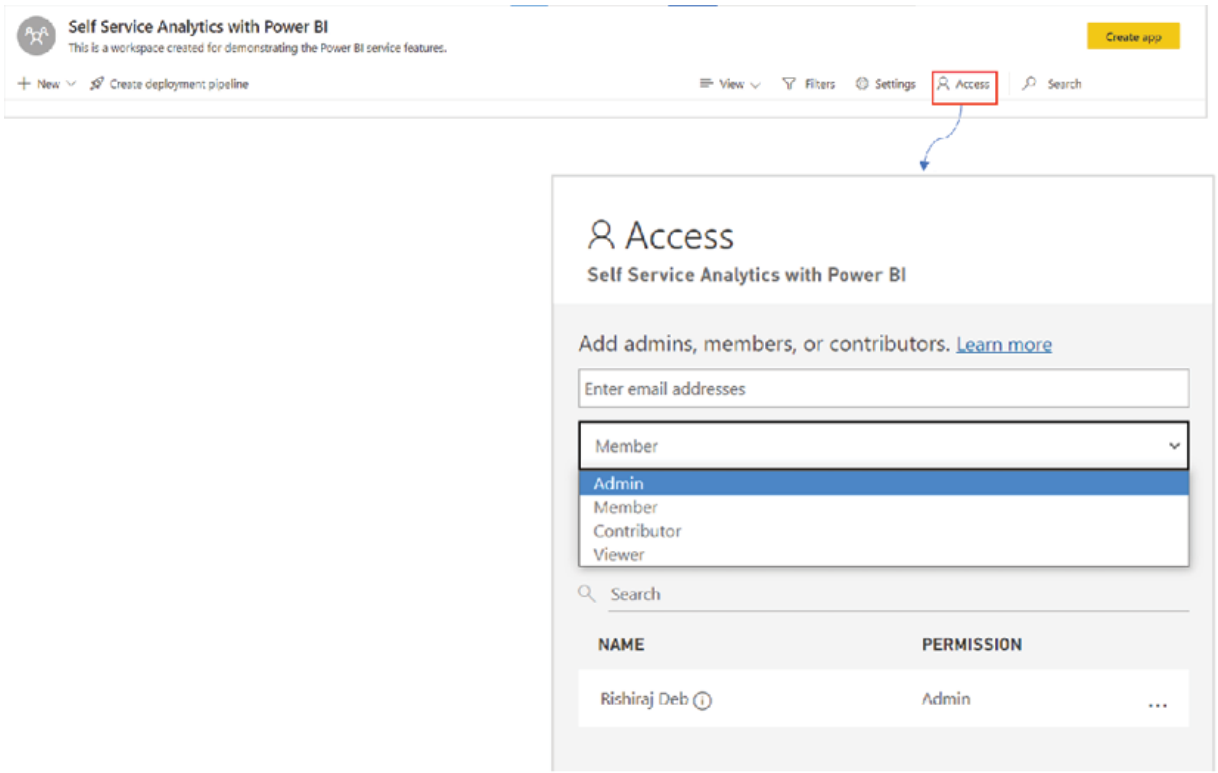



Figure Selecting user roles from the Access menu

The available roles in a Power BI workspace are as follows:

Viewer: The most basic role; provides read-only access to different workspace items. Viewers can view as well as interact with the visuals.

Contributor: Contributors have more privileges than viewers and can create, edit, copy or delete content in a workspace, in addition to publishing reports, schedule refreshes etcetera.

Member: Apart from having all the privileges that contributors have, additional capabilities like managing dataset permissions, publishing and updating Apps, and adding members/contributors/viewers to workspaces come with this role.

Admin: Workspace administrators have full access to the available functionalities.

In a workspace, if we explore the More options for any report, the Manage permissions option should take us to a page where permissions granted for each role can be viewed for different workspace items.

Let us add one user group to each of the four different roles that we have in our Self-Service Analytics with Power BI workspace, as shown in Figure

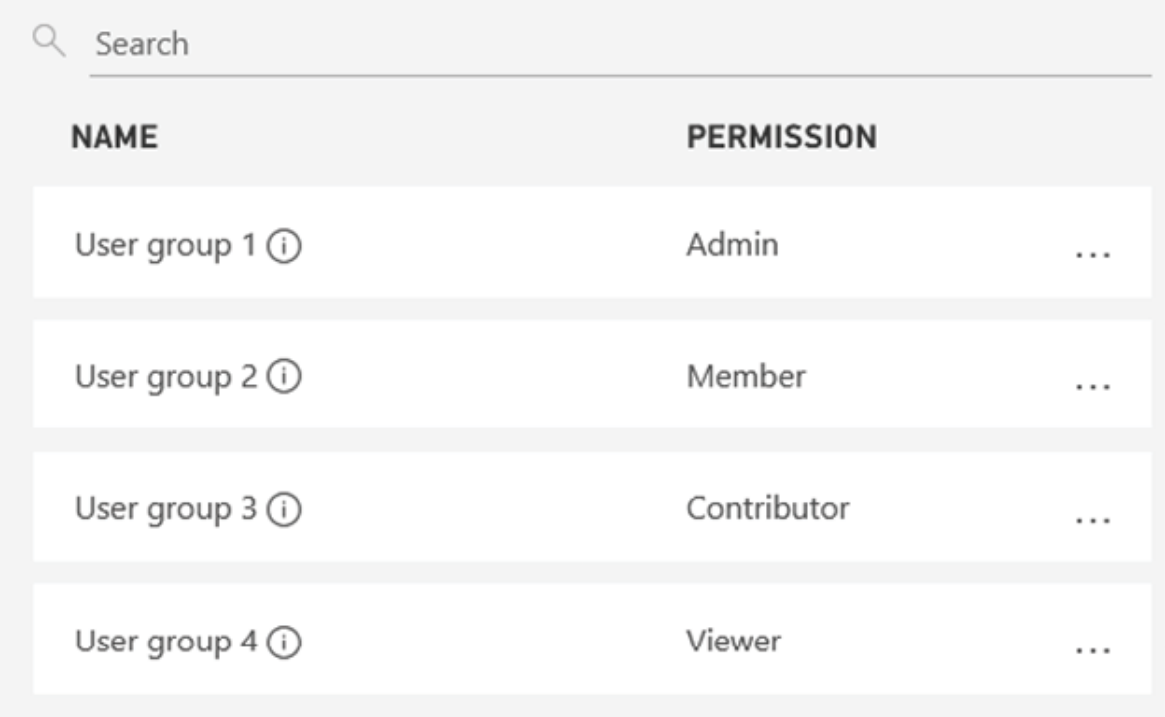



Figure Assigned roles to different user groups

Now for our sales report, upon selecting Manage permissions and navigating to the Direct access tab, permission details for all the user groups will be visible, as illustrated in Figure

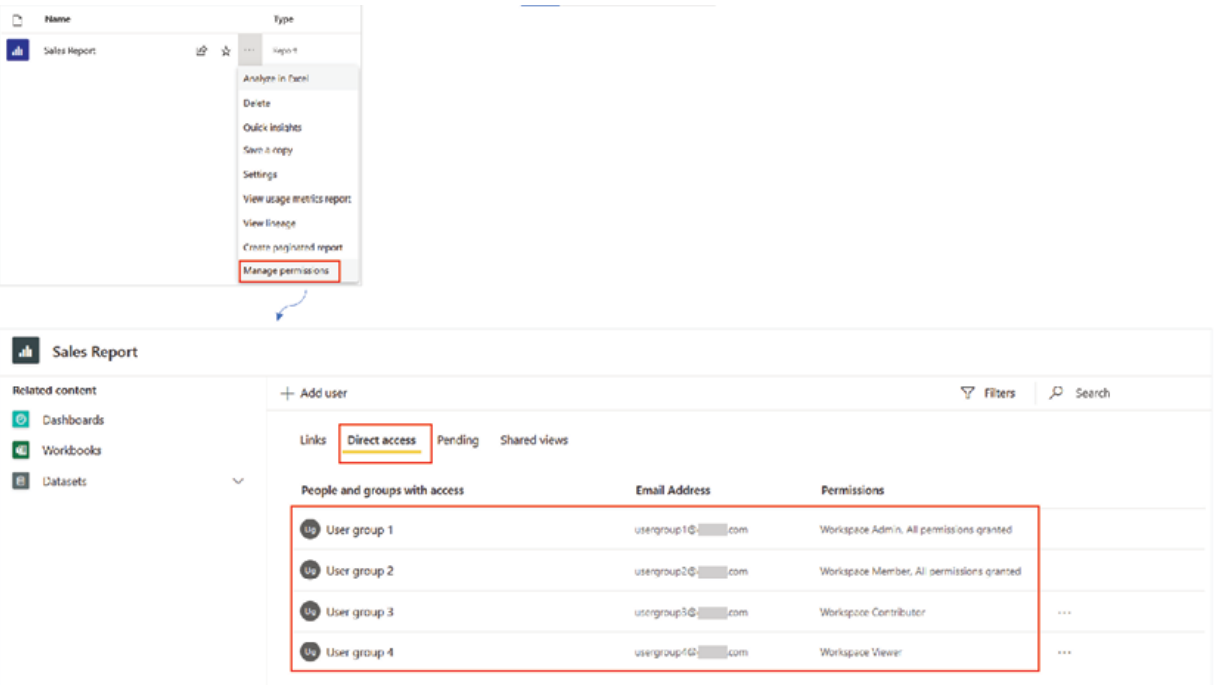


Figure 5.27: Detailed view of granted permissions in a workspace

Roles should be assigned on a need-to-have basis for the users in a workspace depending on the requirement.

## Managing security

Data security is an extremely important aspect today, and there are multiple approaches available in Power BI in this regard to explore and implement. We have already discussed the authentication requirement for Power BI Service, the ability to restrict features across a tenant using the admin portal, providing workspace roles to intended users and so on, all of which play parts in terms of creating a secure environment.

However, a common business requirement is to implement security at the data level, so that different people viewing the same report can see different subsets of data. For example, in our sales report, we have multiple countries like the USA, Canada, Germany, etc. The requirement can be to restrict users from one country from viewing data that belongs to other countries. In Power BI, this can be achieved using a feature known as Row Level Security or which enables to apply security, down to each record or row in the dataset.

RLS needs to be configured in both Power BI Desktop and Power BI Service; in Power BI Desktop roles need to be defined and then in Power BI Service, users need to be added to the roles. Let us go back to our Sales Report.pbix file to create a security role.

On Power BI Desktop, the Manage roles wizard can be opened by following Modeling | Manage as shown in Figure

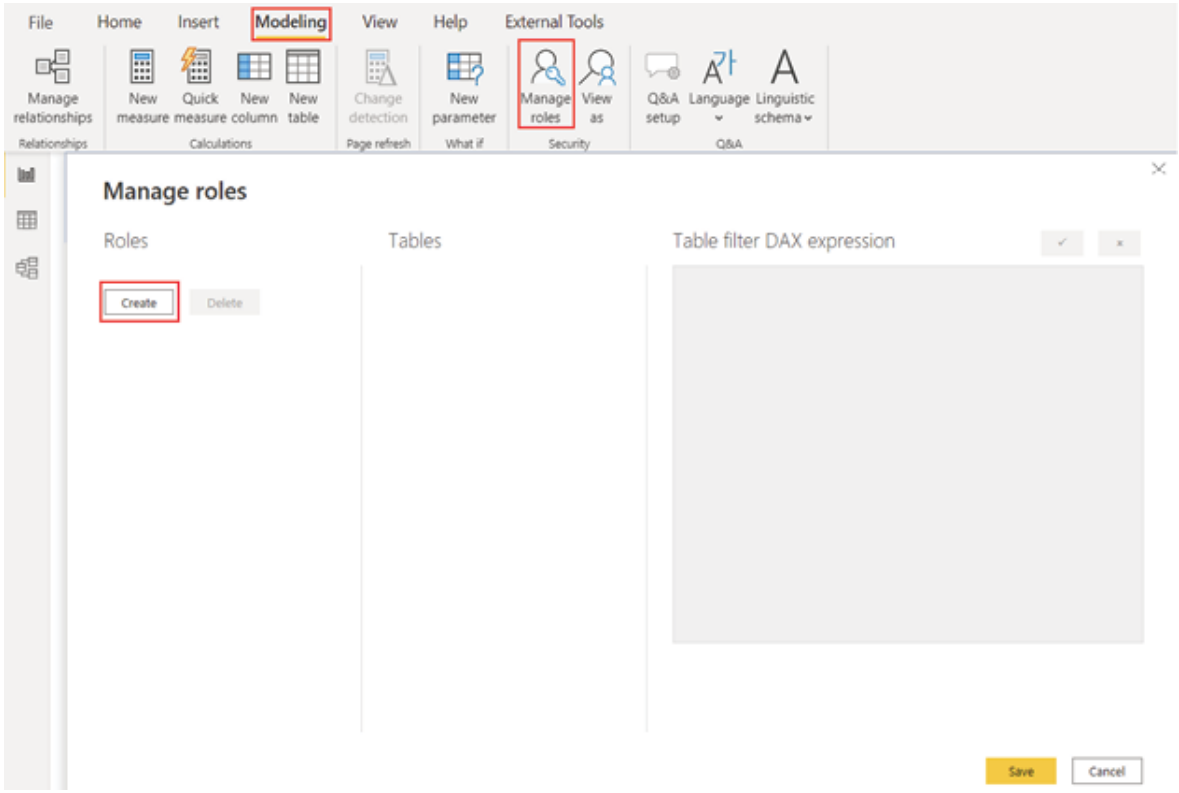



Figure 5.28: Launching the wizard for managing roles

The Create button can be used to create a new role. A meaningful name should be provided for each role as later in the service, we need to add relevant users to the roles. Once the role is created, all available tables in the model become visible under the Tables section. Within each role, filters can be applied using a DAX expression to the required table, which will eventually restrict data access for intended users.

Let us now create a role for US Users which filters the dim_Territory table with the United as illustrated in Figure

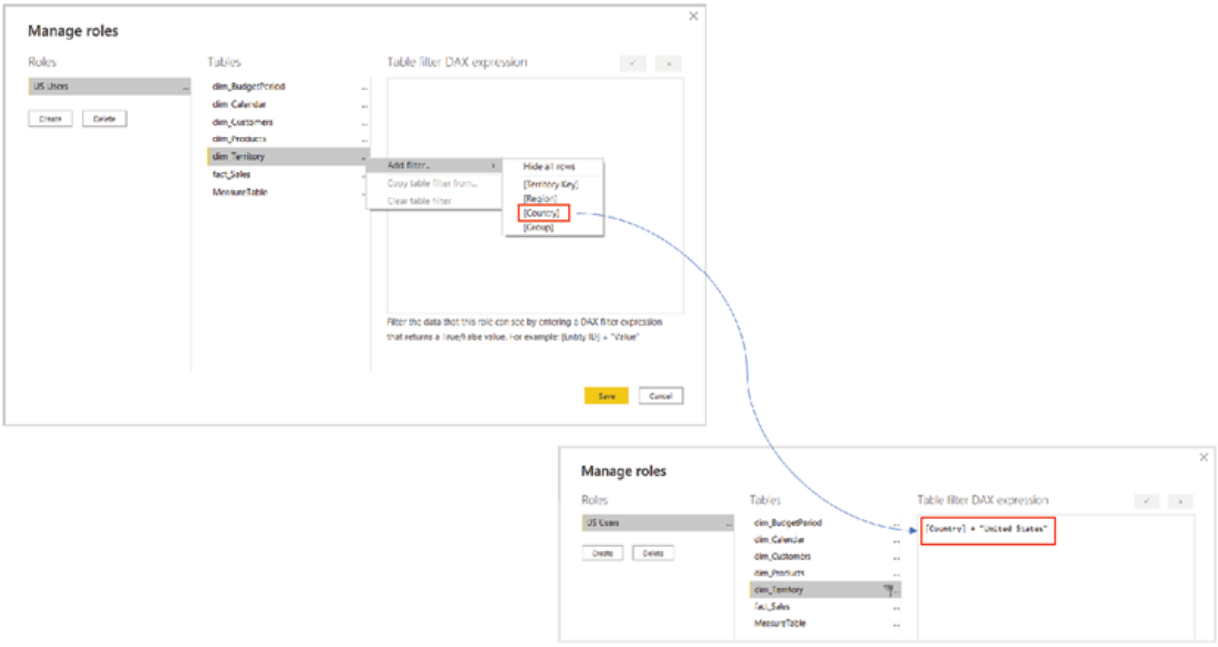



Figure Creating a role

When applied, the expression should be evaluated against each row of dim_Territory and filters the table displaying all the records for which the expression comes as As dim_Territory filters this security filter should propagate and filter fact_Sales as well, with United States data.

Once the role is created, it can be validated in Power BI Desktop using the View as an option under the Modeling tab. Using this option, any of the available roles can be selected and the report can be viewed with the role, as shown in Figure

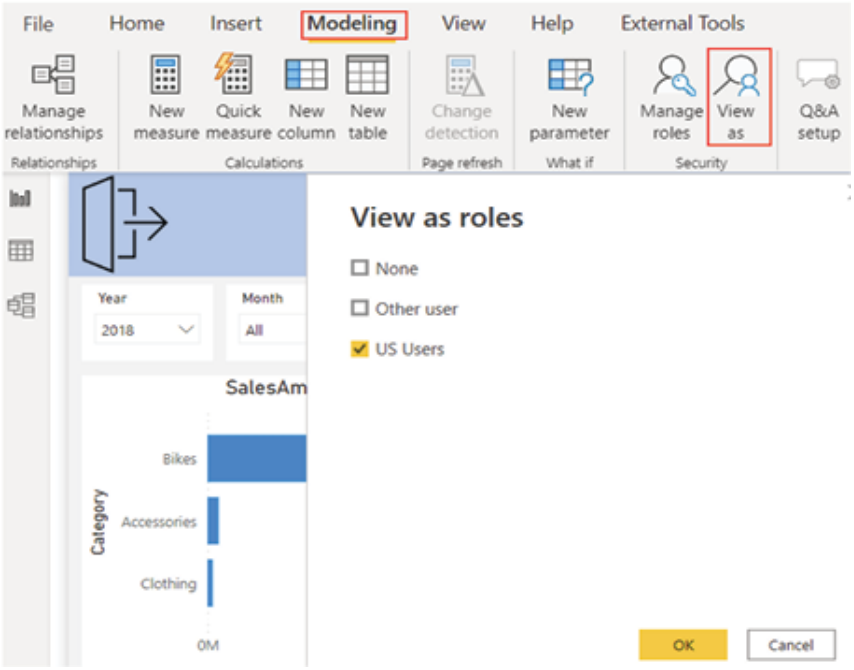



Figure Validating a role

Viewing the report with the role of US shows only the data relevant for United as illustrated in Figure

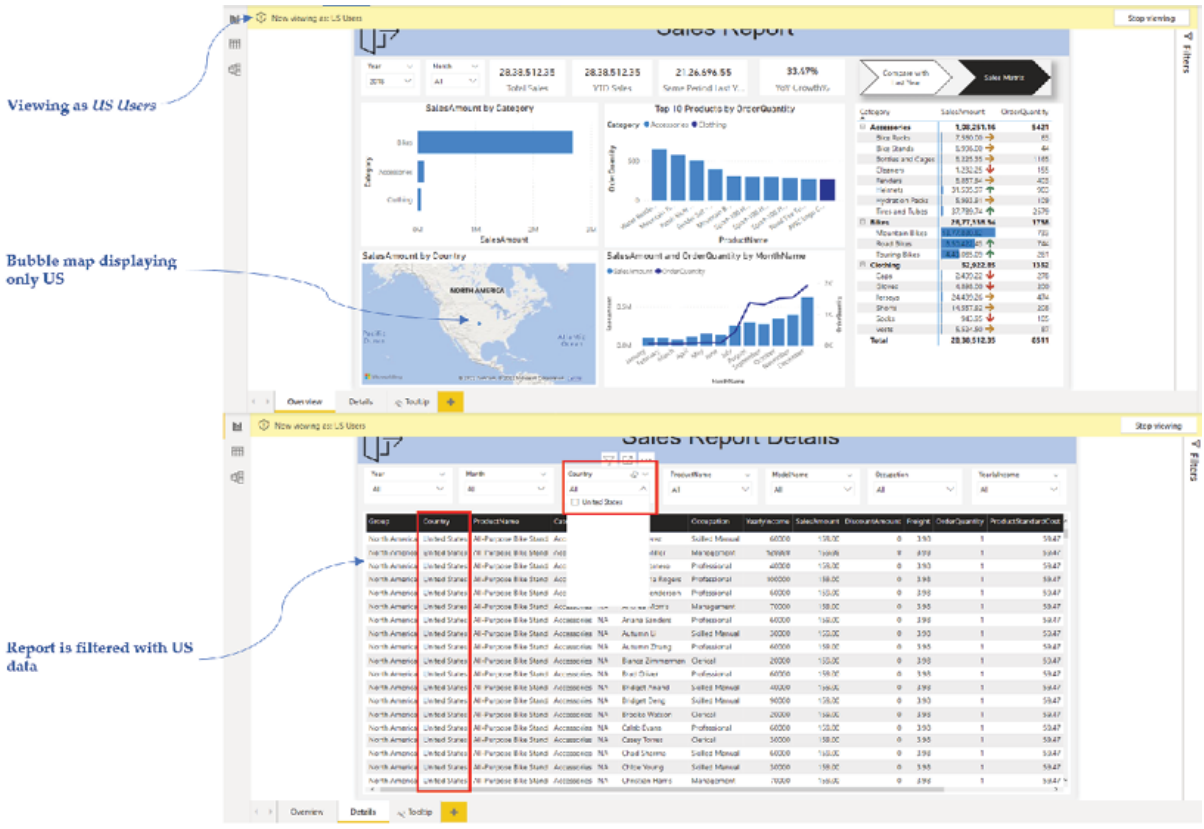


Figure Viewing report with US Users role

Now as the role is validated and working as expected, the file can be published to Power BI Service. In our case, we need to overwrite the existing dataset as we already published the report. While publishing, the role definitions also flow to Power BI Service, where we can now add users to the available security roles.

In Power BI Service, from the More options menu of the dataset, selecting Security should take us to the Row-Level Security page, where the roles created in Power BI Desktop would be visible and users can be added as the security role members, as illustrated in Figure

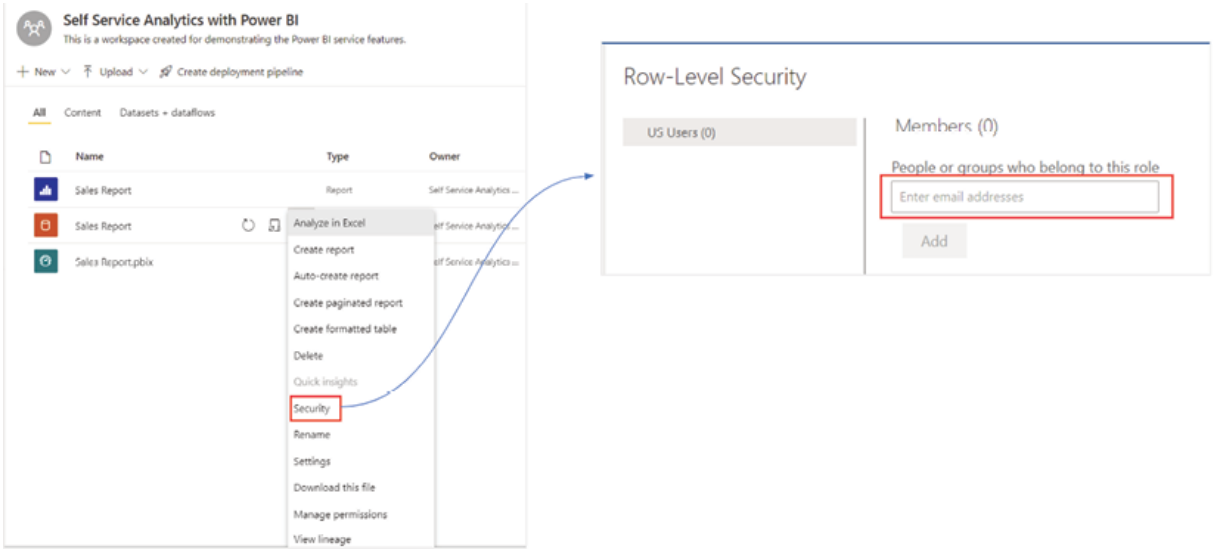



Figure Adding members to the US Users role

Only workspace contributors or higher access level users can add members to a security role. Both individual users, as well as user groups (distribution lists or security groups), can be added as members of a security role. Like the Power BI Desktop, a role can be validated in Power BI Service as well

using the Test as role feature, which is available on the More options menu of the security role in the Row-Level Security page, as shown in Figure

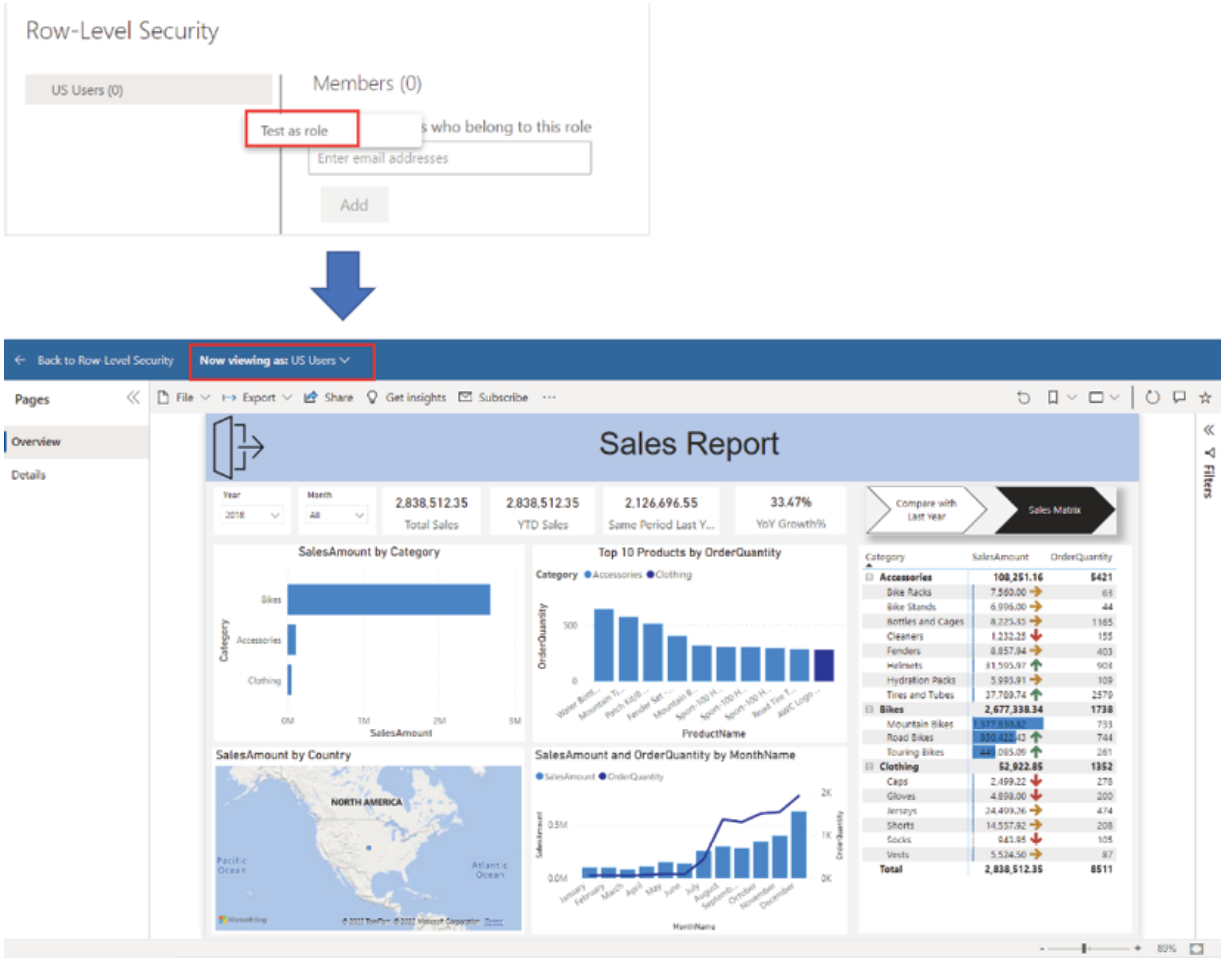



Figure Validating roles in Power BI Service

The Row Level Security only works for workspace Viewers or users who have read-only access to the report. All the other workspace user roles like Members and Admins have edit permission to the dataset and hence RLS is not applicable to them.

The security option that we just discussed is also known as Static due to its not-so-flexible nature. We need to create new roles to the Power BI Desktop file whenever required, and then again add members to that role.

Also, in this way, we can end up with a large number of security roles, having different filters or rules defined as per business needs.

An alternate approach, known as Dynamic RLS, is also available in case the data model has user email addresses in it, along with information about the relevant data the users should have access to. Here, we need to create only one role using the DAX USERPRINCIPALNAME() function, which essentially returns the email address of the logged-in user in the Power BI Service. Whenever a user tries to access the report on the service, the email address gets passed to the underlying dataset and filters the user email address field of the data model, hence rendering the report with relevant data for the logged-in user.

Although RLS is an excellent option to implement data level security in Power BI, implementing it to maintain a good performance, especially for large data volumes can sometimes be tricky. Another simple option can be workspace-level security, where we can split the dataset (and hence the report) in the Power Query editor for different countries, and publish country-specific reports to different workspaces. Then, we can add, for example, United States users to the workspace where the US-specific report has been published and so on, hence restricting users from viewing reports for other countries. Figure 5.34 illustrates a workspace-level security implementation:

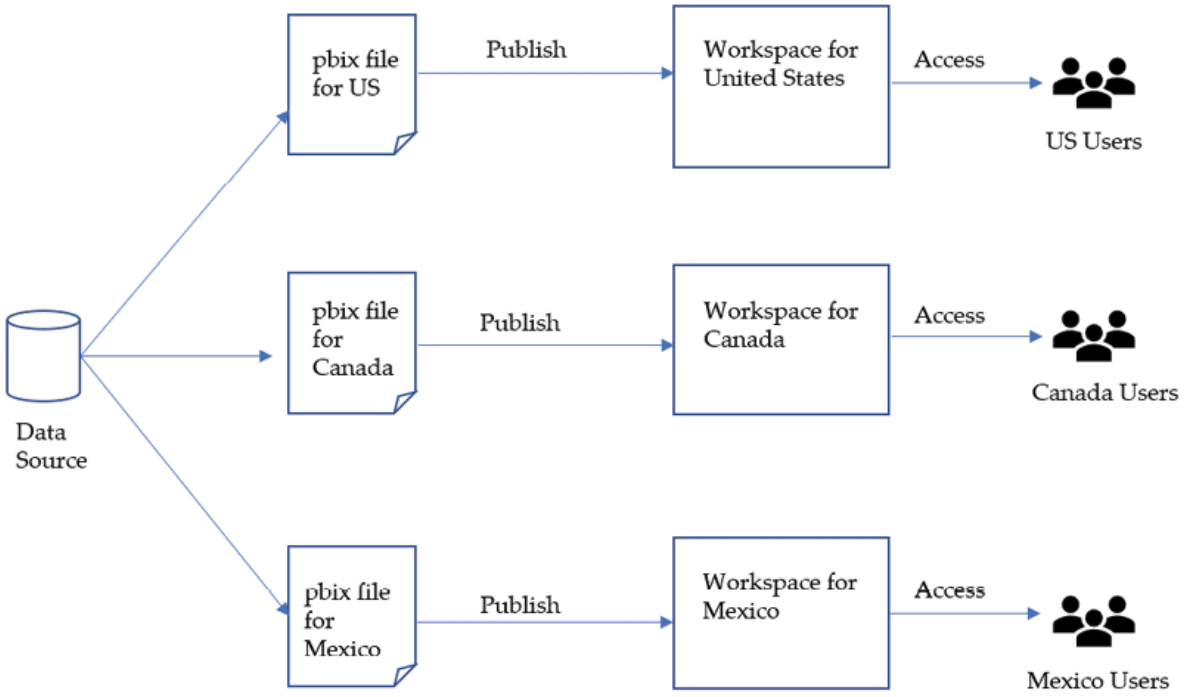



Figure Workspace level security option

Note: RLS can be implemented for imported data as well as for selected DirectQuery connections like SQL Server, however, is not available to be implemented in Power BI for live connections like while working with Analysis Services.

## Dashboards and alerts

So far, we discussed reports in Power BI. Reports are usually created for specific functions, for example, Sales Inventory Report, and so on. On the other hand, dashboards can provide a holistic view of what is happening in an organization across a spectrum of reports, all in a single page. Dashboards can bring together visuals or tiles from multiple reports of a workspace in a single, scrollable page, to tell a story.

Whoever has edit permission of a report can pin a visual to a dashboard, using the Pin visual option which appears on hovering over a visual in the report. The visual can be pinned to an existing dashboard, or to a new dashboard which gets created along the way. The process is illustrated in Figure

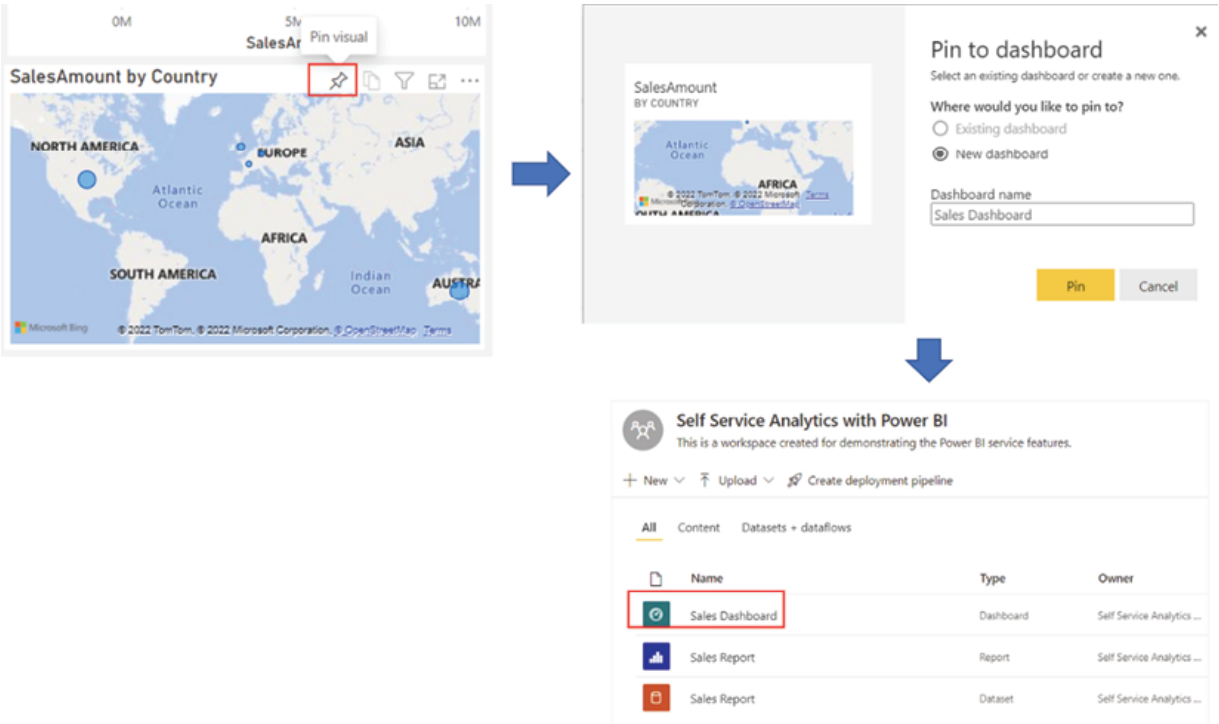


Figure Pinning a visual to a dashboard

The dashboard can be found under both the All and Content tab of the workspace. We can open the dashboard by simply selecting it and all the visuals that have been pinned on it from different reports would be visible on a single page. Pinned visuals in a dashboard are set with the filter context at the time of pin. If the filter context of the underlying visual changes, the dashboard tile needs to be updated as well to reflect the change. At present, slicers and filters cannot be pinned to a dashboard individually.

To pin more than one visual at a time, an entire report page can be pinned to a dashboard using the Pin to a dashboard option, under the More options menu, as shown in Figure

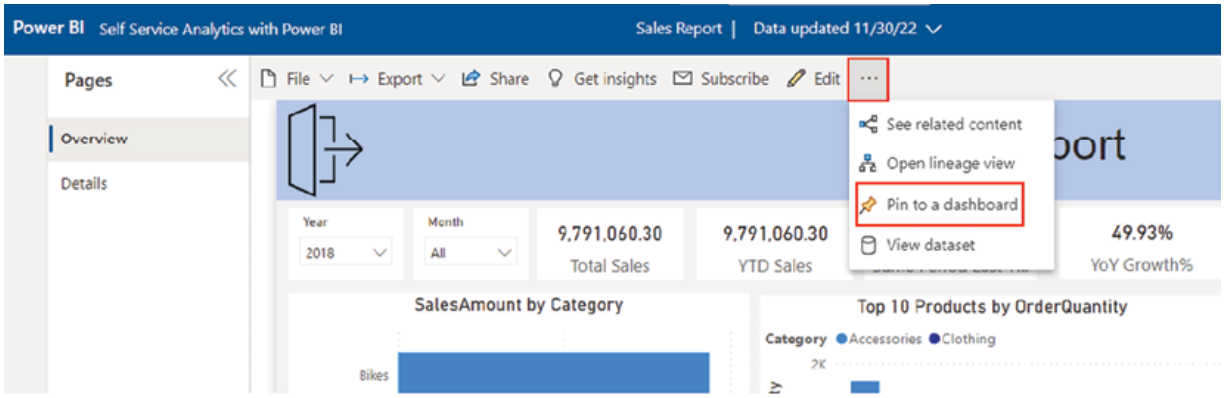



Figure Pinning a report page to a dashboard

In the case of pinning an entire page, the tiles or the visuals of the page are live in the dashboard, which means they can be interacted with in the dashboard itself. Selecting any tile on the dashboard should re-direct the user to the report from which the tile has been pinned.

Another interesting feature of a dashboard is the ability to create data-driven notification alerts. As of now, alerts can be created only on KPI and Card visuals. To create an alert, Manage alerts can be selected from the More options menu of the tile, as shown in Figure

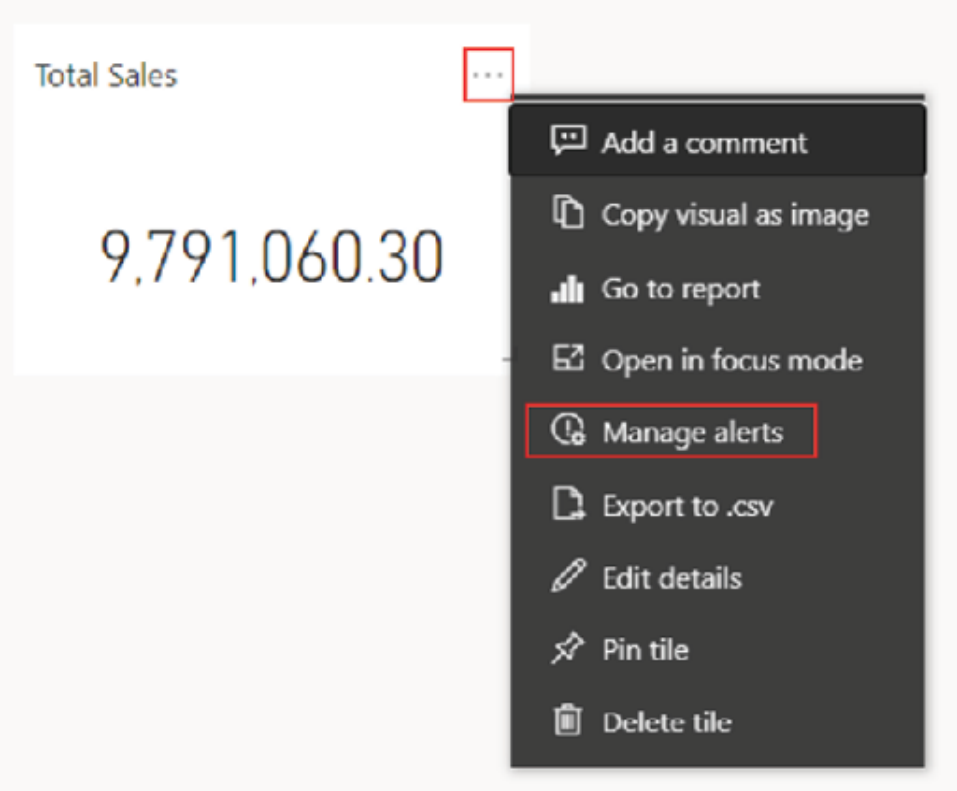



Figure Managing alerts on a dashboard

On the Manage alerts window, an alert rule needs to be configured with a threshold value. Whenever the value of the tile changes and goes beyond the threshold limit, a notification gets triggered depending on the frequency that has been chosen. Figure 5.38 shows a sample alert created for the Total Sales card that has been pinned to the dashboard from our sales report:

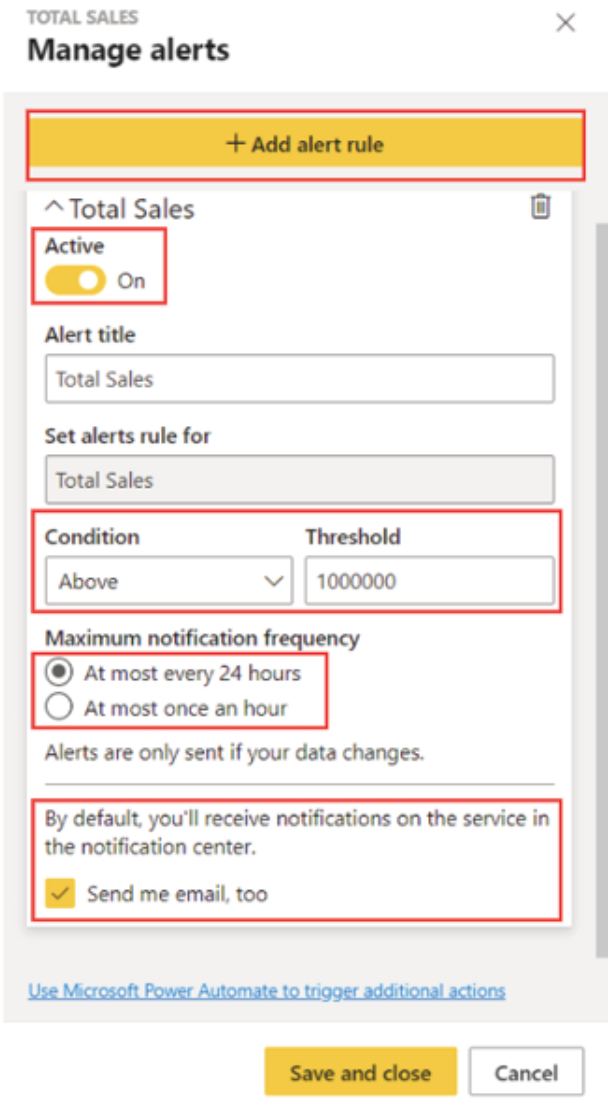



Figure Sample alert configuration

By default, Power BI sends notifications to the notification centre of the service, and optionally to the mailbox in case opted for an email. A sample notification is shown in Figure

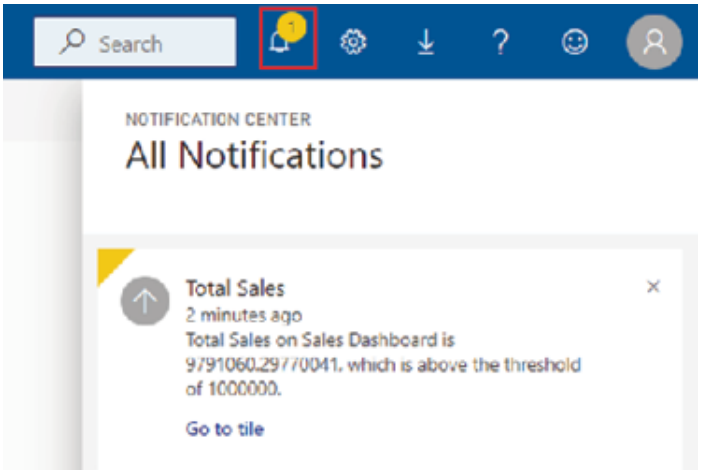



Figure Sample notification in the notification centre

Alerts only work on refreshed data. During the data refresh if the threshold limit for an alert is crossed, only then notifications are sent.

## Refreshing data and data gateways

Now that we have created our report, published it on Power BI Service, and configured the environment as required, the only remaining part is setting up a refresh mechanism for the report to keep it up to date with the latest data.

In the case data is imported in a Power BI dataset, refreshing data means querying the underlying data source and loading the data into the dataset overwriting any existing data. Once data is updated in the dataset, all the visuals in the reports that are created using the dataset also get updated. However, in the case of a DirectQuery mode, no data is imported to the Power BI model, instead, every report interaction sends a query back to the data source and returns the result. Hence for a DirectQuery mode, the dataset or data model does not require to be refreshed separately.

Once a report is published to Power BI Service, then the cloud service needs to establish a connection to the underlying data source, for the refresh to happen. If a dataset connects to a data source that Power BI cannot access over a direct network connection, such as on-premises data sources, a data gateway needs to be configured to be able to refresh the dataset from Power BI Service.

A data gateway is software that acts as a bridge between the cloud service and the on-premise data source. Two types of data gateways are available to be used in Power BI, as follows:

Enterprise data gateway: This can be used by multiple users to refresh data from an on-premise data source. All required data source definitions like server addresses, authentication modes, credentials, etc. need to be added to the enterprise gateway. Only gateway admins have right to add data sources to an enterprise gateway. Gateway admins can also add a list of users as gateway users, who would have permission to use the gateway and assign the gateway to a Power BI dataset.

Personal data gateway: A personal data gateway can be used by a single person only. For a personal gateway, it is not required to add the data source definitions to the gateway; the data source configurations are managed by the Data source credentials section in the dataset settings, which can be accessed using the More options menu of a dataset, as shown in Figure

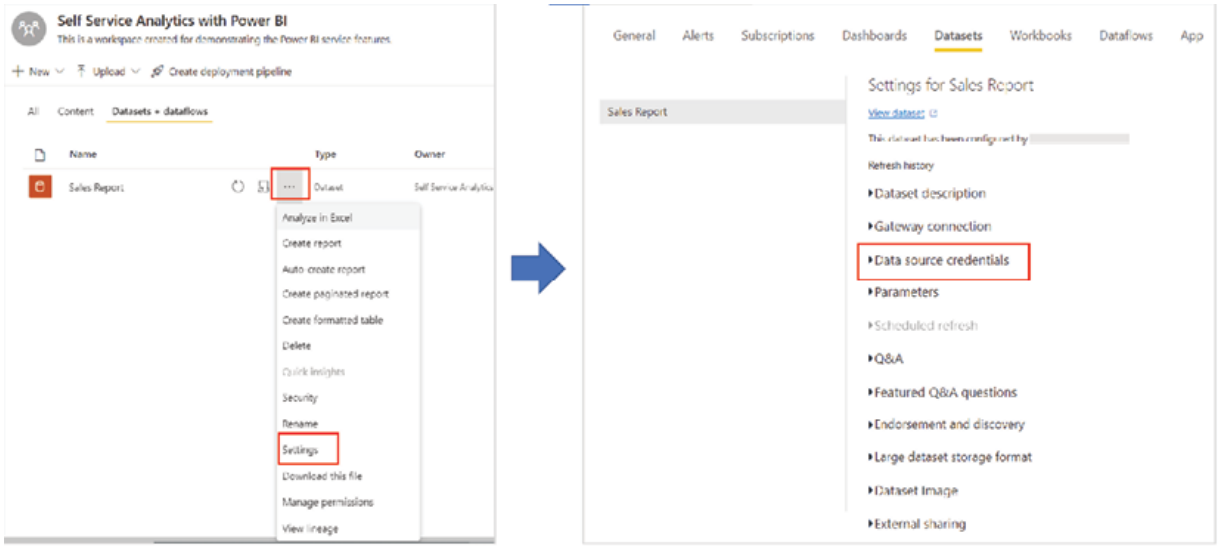



Figure Data source credentials section of a dataset

On Power BI Service, by following Download | Data Gateway redirects to the Microsoft's webpage for Power BI gateways, which has options to

download either the Standard mode (enterprise mode) or the Personal mode data gateway, as shown in Figure

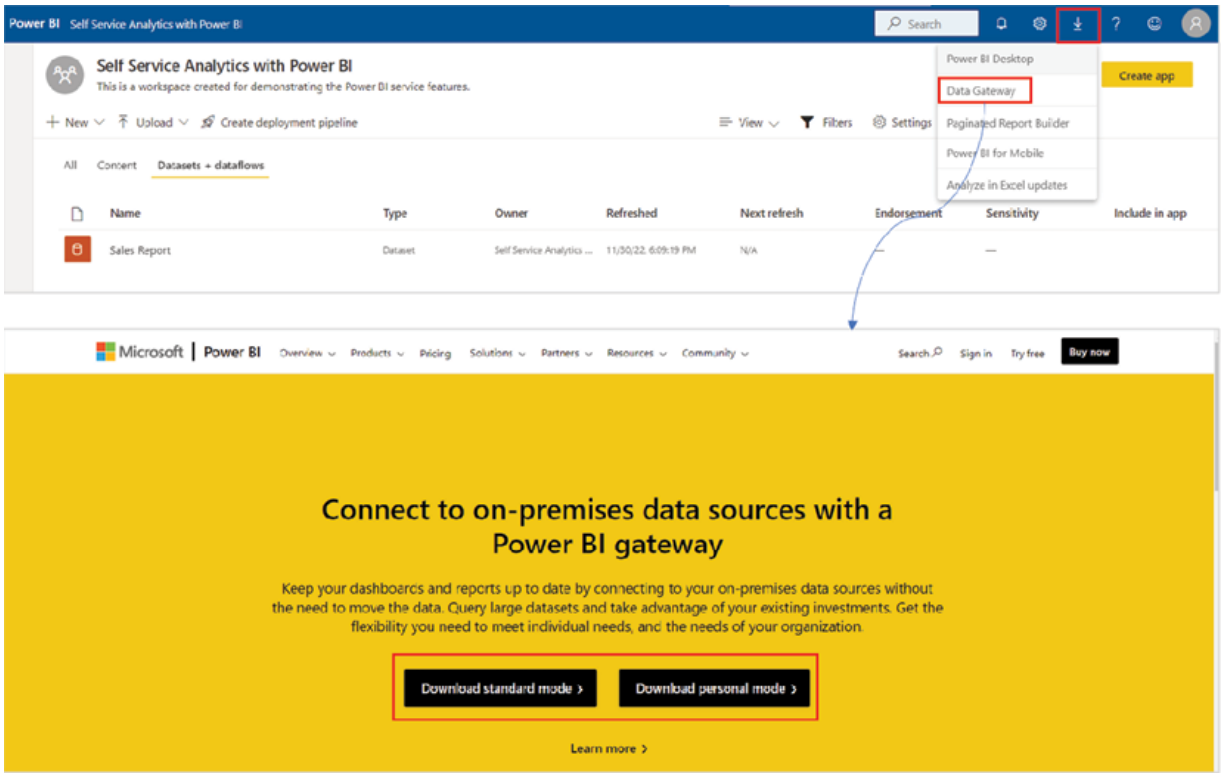



Figure Downloading data gateway

For our sales report, we have imported the data from an Excel file saved on a local PC, hence, to be able to refresh it, we will need a personal gateway. Let us download the personal mode data gateway and install it. During installation, we need to register it with the Power BI Service account, as shown in Figure

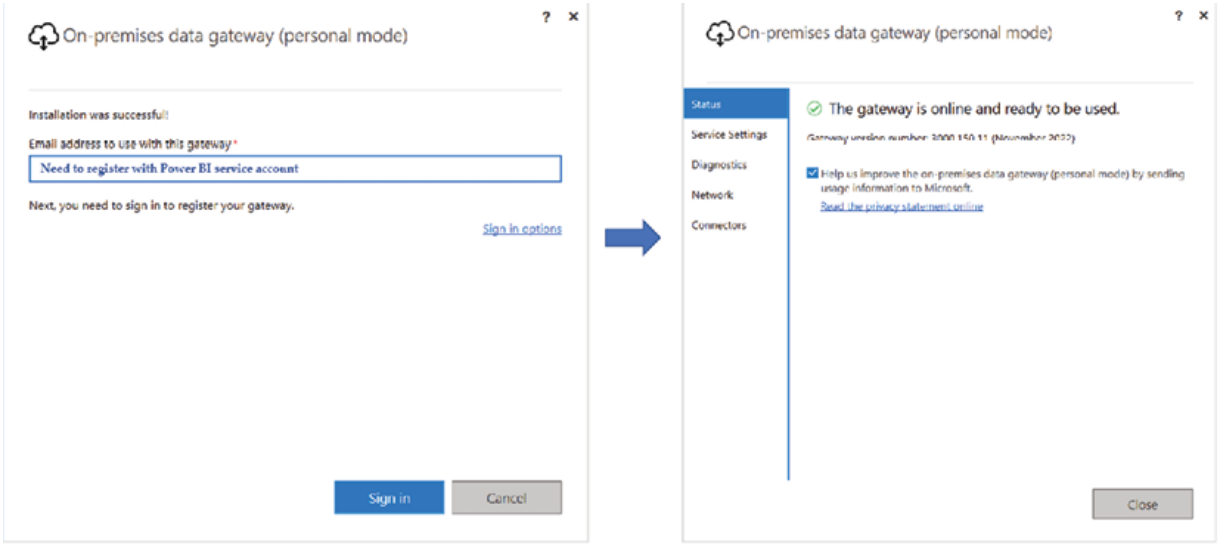



Figure Registering a personal mode data gateway

Once registered against the tenant, the gateway should be discoverable in Power BI Service on the On-premises data gateways tab by following Settings | Manage connections and as illustrated in Figure

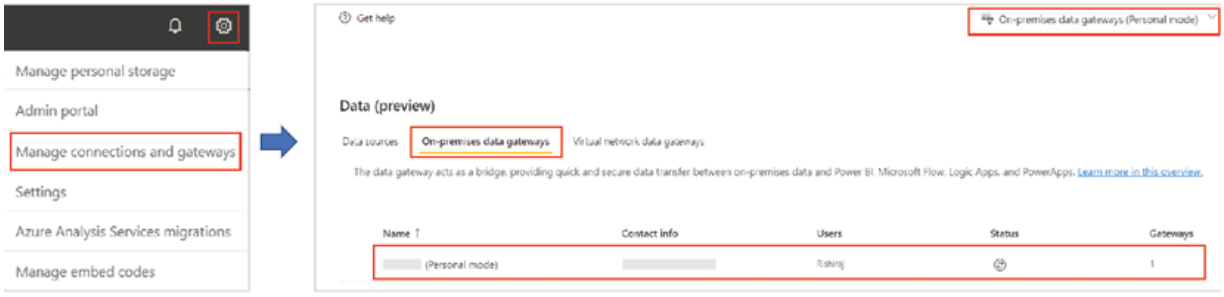



Figure Personal gateway available in Power BI Service

For any published dataset, the gateway can be assigned using the Gateway connection section, from the More options menu of the dataset, as shown in Figure

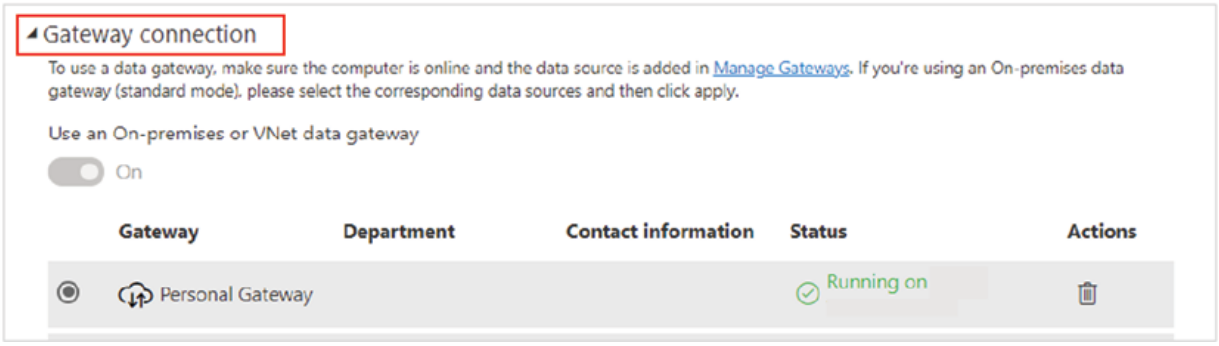



Figure Gateway assigned to the dataset

Datasets created out of cloud services do not usually require any gateway, provided Power BI can establish a direct network connection to the data source. However, any restriction for a direct network access like a firewall rule, can force Power BI to refresh data via a data gateway only.

Once a refresh mechanism is established, the Power BI dataset can be refreshed based on a schedule or on-demand. To configure a scheduled refresh, we need to go to the dataset settings first using either the Schedule refresh option which appears on hovering the dataset, or from the More options menu of the dataset. Then, the Scheduled refresh section of the dataset settings can be used to schedule a refresh, as shown in Figure

Figure Scheduling a dataset refresh on Power BI Service

A pro workspace which is on shared capacity supports a maximum of eight scheduled refreshes per day, while for a dedicated capacity or premium workspace, we can schedule as many as forty-eight refreshes daily. Apart from setting up the refresh frequency, the scheduler can also be used to send refresh failure notifications to dataset owners and/or other key stakeholders.

A dataset can be refreshed on-demand as many times as required without any limitation, using the Refresh now option which appears upon hovering over the dataset, as shown in Figure

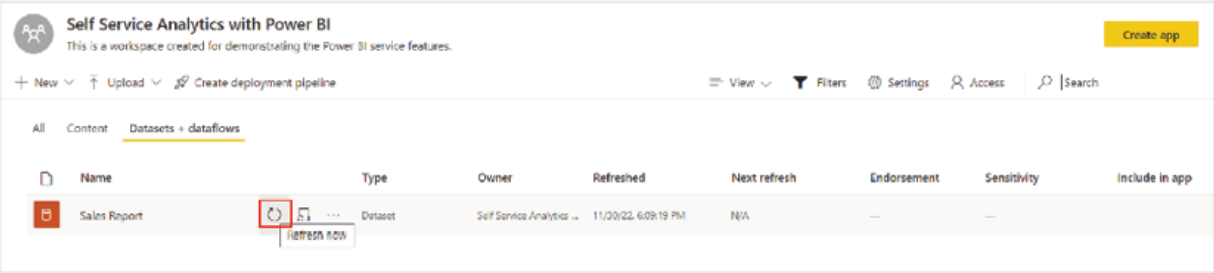



Figure Refreshing on-demand using Refresh now option

Apart from scheduled and on-demand refreshes, a Power BI dataset refresh can be automated programmatically to make it event-based, using the Power BI REST as well.

The refresh the history of a dataset can be accessed from dataset settings, and contains information like refresh type, refresh start and end time, refresh status, failure message, etc. as shown in Figure

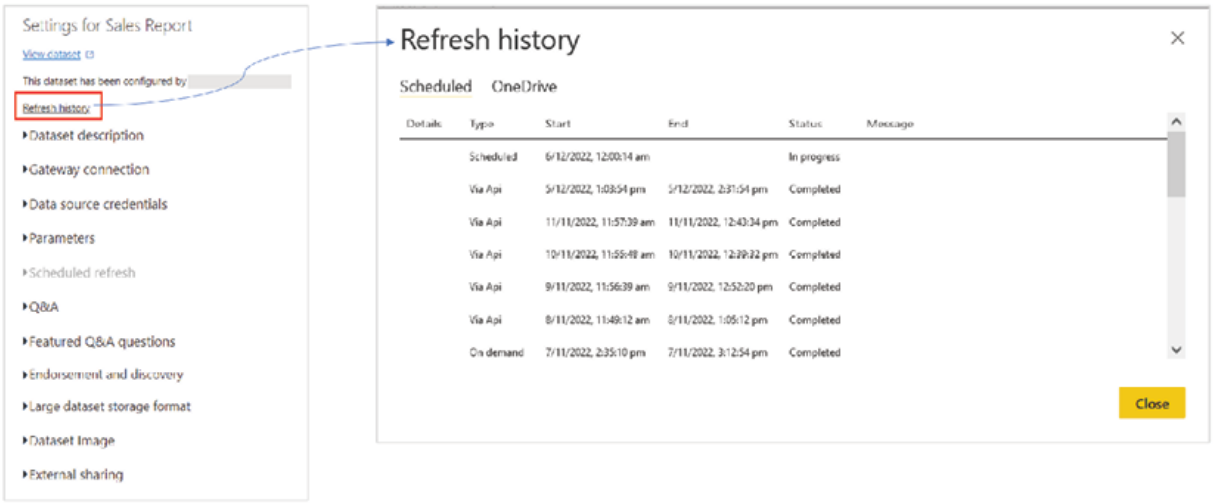



Figure Refresh history of a Power BI dataset

For a dataset with import mode, an ongoing refresh can be cancelled in case the dataset resides in a premium or dedicated capacity, using the Cancel refresh option that appears upon hovering over the dataset, as shown in Figure


Figure Cancelling an ongoing dataset refresh

For a pro dataset, the maximum refresh duration is two hours while for a premium dataset, the duration is five hours before timeout.

Note: Every dataset has one owner who can only perform certain activities like updating credentials, schedule refreshes etcetera, for the dataset.

Anyone who is not the owner needs to take over the dataset first, using the Take over button on the dataset settings page, to have full control over the dataset.

## Apps

Apps are an easy and convenient way to package multiple Power BI content together from a workspace and distribute it to a large audience. Apps can be created from a workspace which hosts the Power BI contents, using the Create app button, as shown in Figure

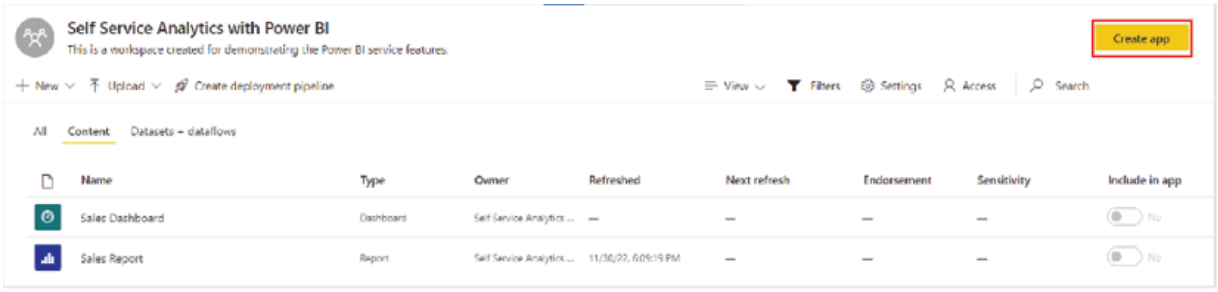



Figure Creating an App from Workspace

Once you click on the Create app button, the app creator opens which has three tabs, namely Content, and

The Setup tab allows you to configure the App in terms of setting up themes, pushing the app automatically to the Power BI account of business users eliminating the need to install the App separately. The Content tab allows the App creators to include content from the workspace and bundle in the App using the Add content option. Figure 5.50 illustrates both the Setup as well as Content tab usage while creating an App:

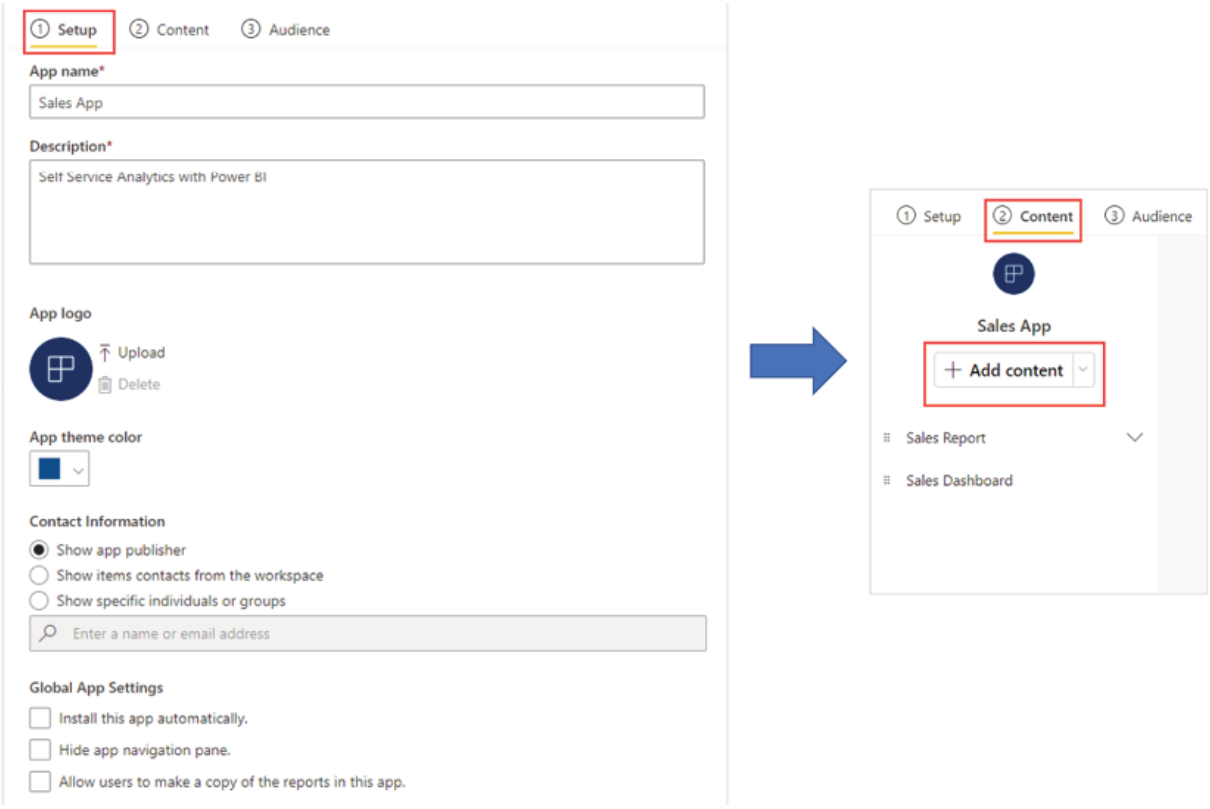



Figure Setting up an App and adding content

Power BI supports publishing only one App per workspace, hence we should include all contents that we intend to distribute to the end users in the Content tab. However, it is possible that we do not want everyone to access everything that we have included in the App, and this is where the Audience section comes into play.

Using the Audience tab, different items of the App can be distributed to different groups of people or audiences, based on persona or roles. Hovering over any item on the Audience tab displays the hide button using which specific items can be excluded from the present view of the App, and then the view can be distributed to a specific user group. The New Audience option is essentially a different view of the same App with all the original items by default, where again specific items can be excluded, and the view can be distributed to a separate user group than before. The Audience tab is illustrated in Figure

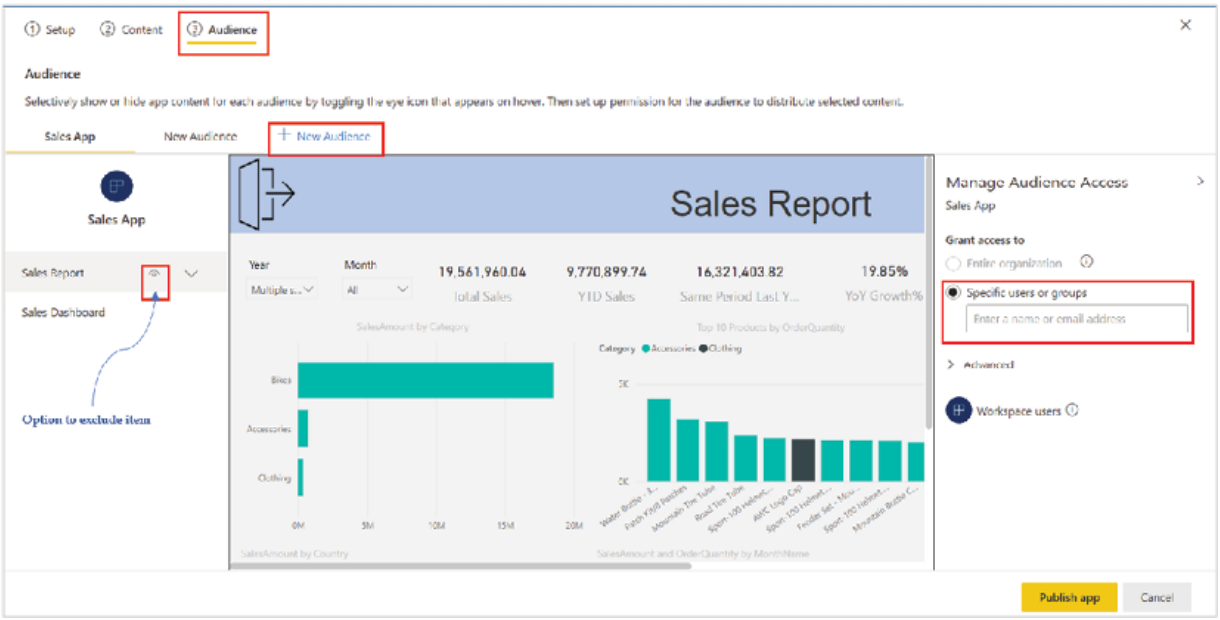



Figure Audience option of an App

Although it is the same app that gets published from the workspace, the Audience option thus provides a way to provide different users access to different contents based on the user group or distribution list they belong to.

Finally, the App can be deployed using the Publish app button, which also generates a direct link to the App, as shown in Figure

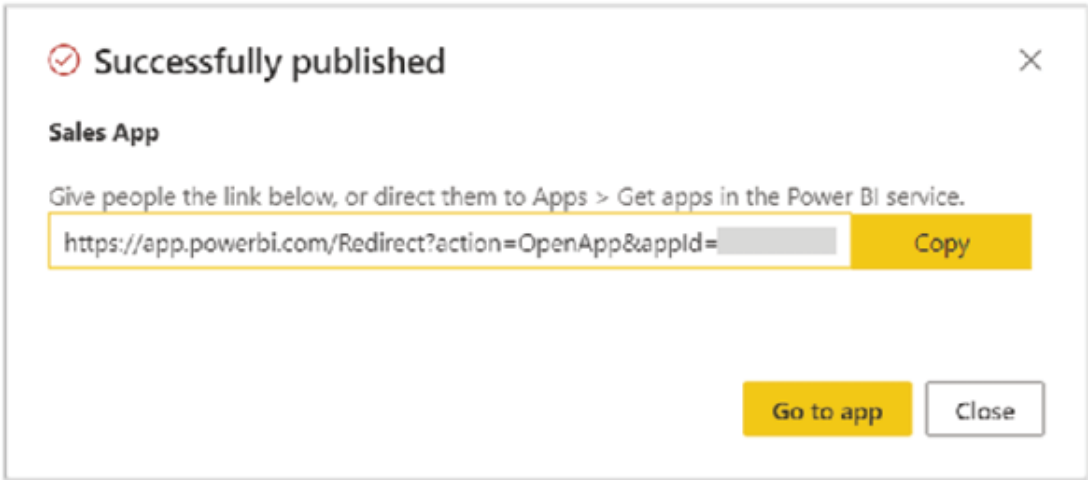


Figure App direct link

Alternatively, the App can be installed and accessed using the Apps section of Power BI Service. The Get apps option in the Apps section launches the App finder or Power BI apps where all Apps that the user is allowed to access would be listed. The Sales App can be searched and then installed using the Get it now option, as illustrated in Figure

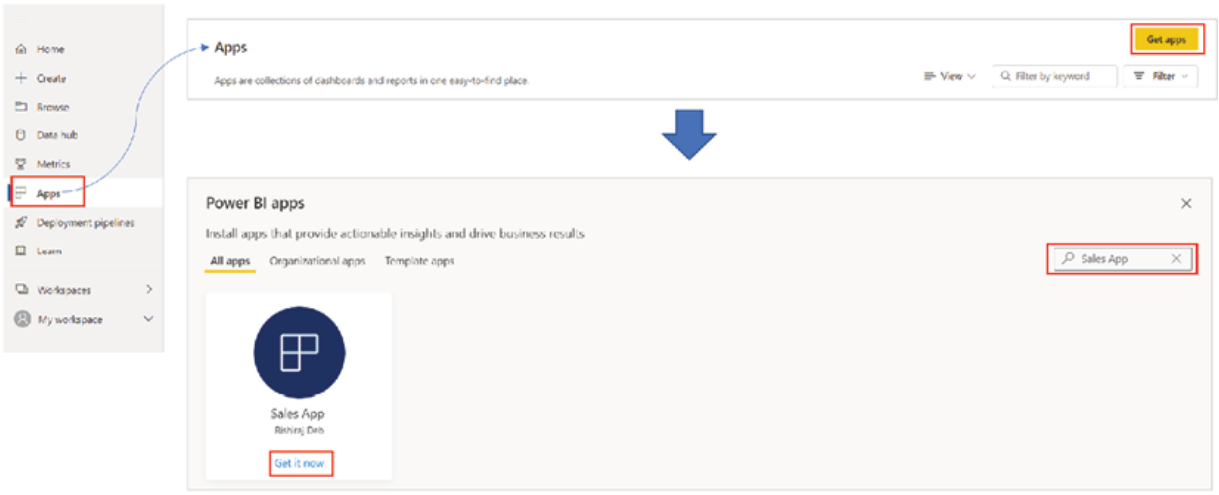



Figure Installing an App

Once installed, the App would be available under the same Apps section, ready to be used. Depending on the access, different items of the App can be viewed and navigated conveniently using the left hand navigation panel. Figure 5.54 displays how the Sales App looks like once published:

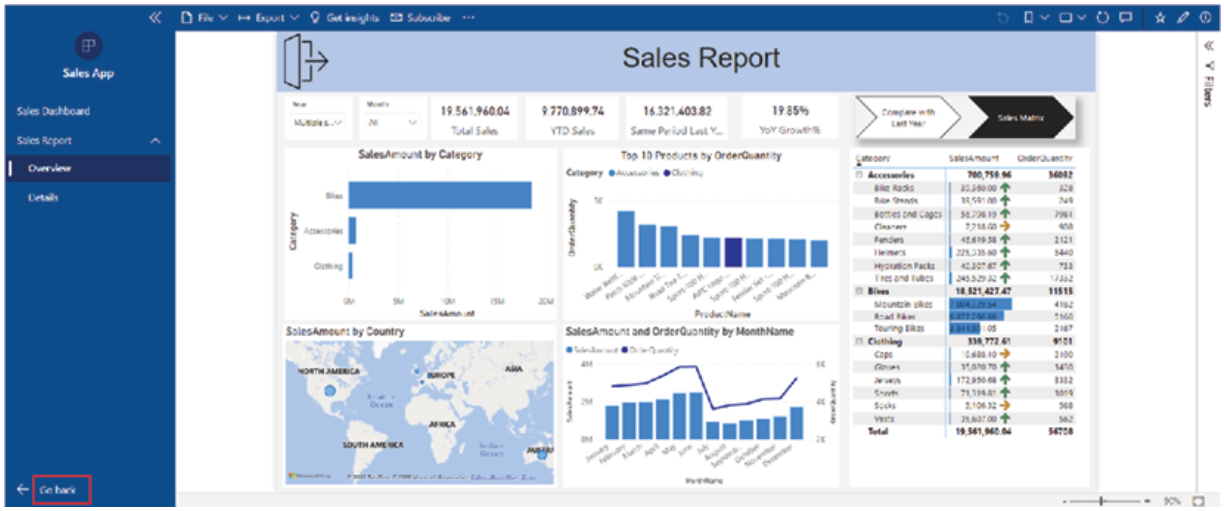



Figure Sales App

To exit from the App experience, the Go back button on the bottom left can be used.

Publishing an App is not mandatory for distributing contents to end users, as access can be given on the workspace itself serving the same purpose. However, in case of working with a large number of report consumers, it is recommended to use an App instead. With an App, we can provide users access to specific items, without exposing other workspace artifacts like datasets, dataflows, and so on. Apps also can provide better user experience and ease of navigation between items.

Once an App is published, it can be further modified using the Update app option which would be available on the workspace itself, as shown in Figure

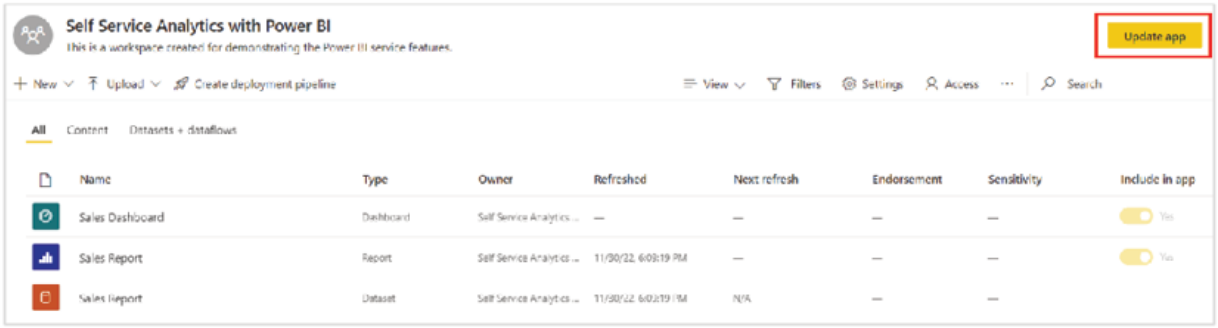



Figure Updating an App

Apps need to be updated only in case of any structural change of the reports or App items. From a data perspective, all App items would be automatically updated once the underlying dataset gets updated with latest data. In case of any structural change, Apps also provide an opportunity to validate the changes on the workspace first, before making it available for the end report consumers.

## Dataflows

Dataflows are collections of tables or entities that are created and managed inside Power BI Service. From a report authoring perspective, dataflows can be considered as an online version of Power Query where after connecting data from various data sources, ETL transformations can be performed on individual entities and then saved, to be further consumed in Power BI datasets.

In a workspace, to create a dataflow, the Dataflow option needs to be selected from the New button menu. The following page would have different options to work with Dataflows; let us add a new table from the Define new tables section. Selecting Add new tables should open the Power Query editor, from where the required data source can be searched for, as shown in Figure

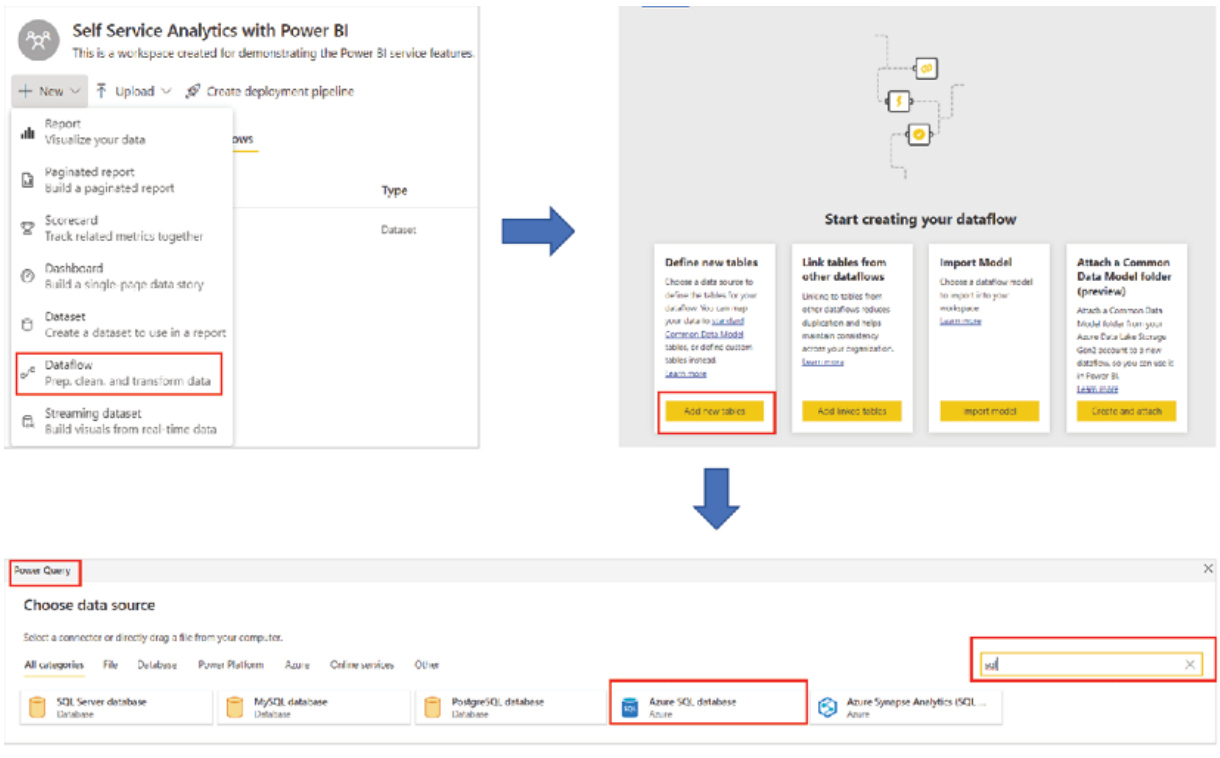


Figure Creating a Dataflow

Let us select an Azure SQL database to connect and provide the required connection parameters like server address and credentials. If the connection string matches with a pre-configured data gateway, a gateway user would be able to assign the data gateway on the connection configuration window, if required. Once the connection is established, all the database schemas and tables that the user has access should be visible and ready to be selected and loaded to the Power Query editor, using the Transform data option, as shown in Figure

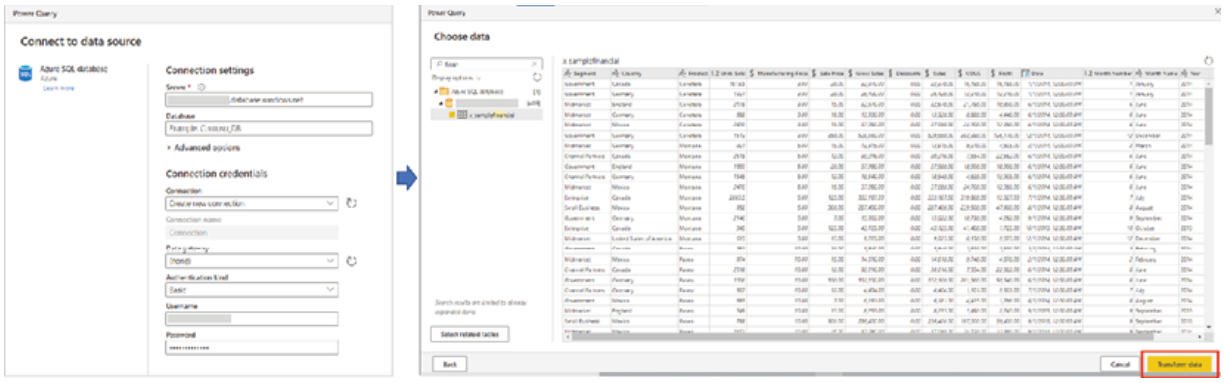



Figure Connecting to an Azure SQL database from Dataflow

Once any table is loaded in the Power Query editor, all transformation options would be available to perform, and gets recorded under Applied steps in Query Selecting Save & close validates the query and saves the Dataflow in the workspace, as shown in Figure

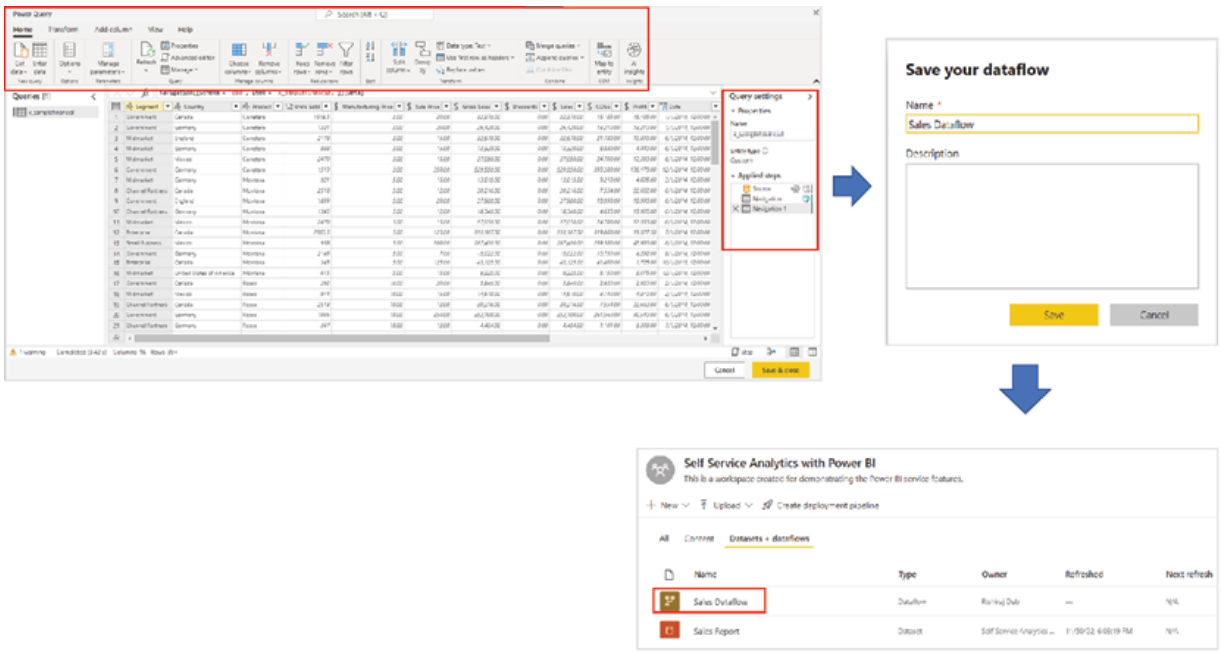



Figure Saving a Dataflow

After saving the Dataflow, it needs to be refreshed to load data into the entities it contains. All available entities in a Dataflow can be viewed upon entering the Dataflow by selecting it in the workspace. Once we are inside the Dataflow, the Edit tables option can be used to modify an existing query, the Add tables option can help to add a new query and the Close option helps to exit the Dataflow back to the workspace items, as illustrated in Figure


Figure Dataflow entities

Hovering over any Dataflow displays the option to and the More options menu can be used to view the Refresh history as well as navigate to the Dataflow Settings from where the refreshes can be scheduled, similar to a dataset.

Once a Dataflow is ready in the service, the entities of it can be imported from a Power BI Desktop file just like any other data source as shown in Figure eliminating the need for Power BI Desktop to directly connect to the underlying data source:

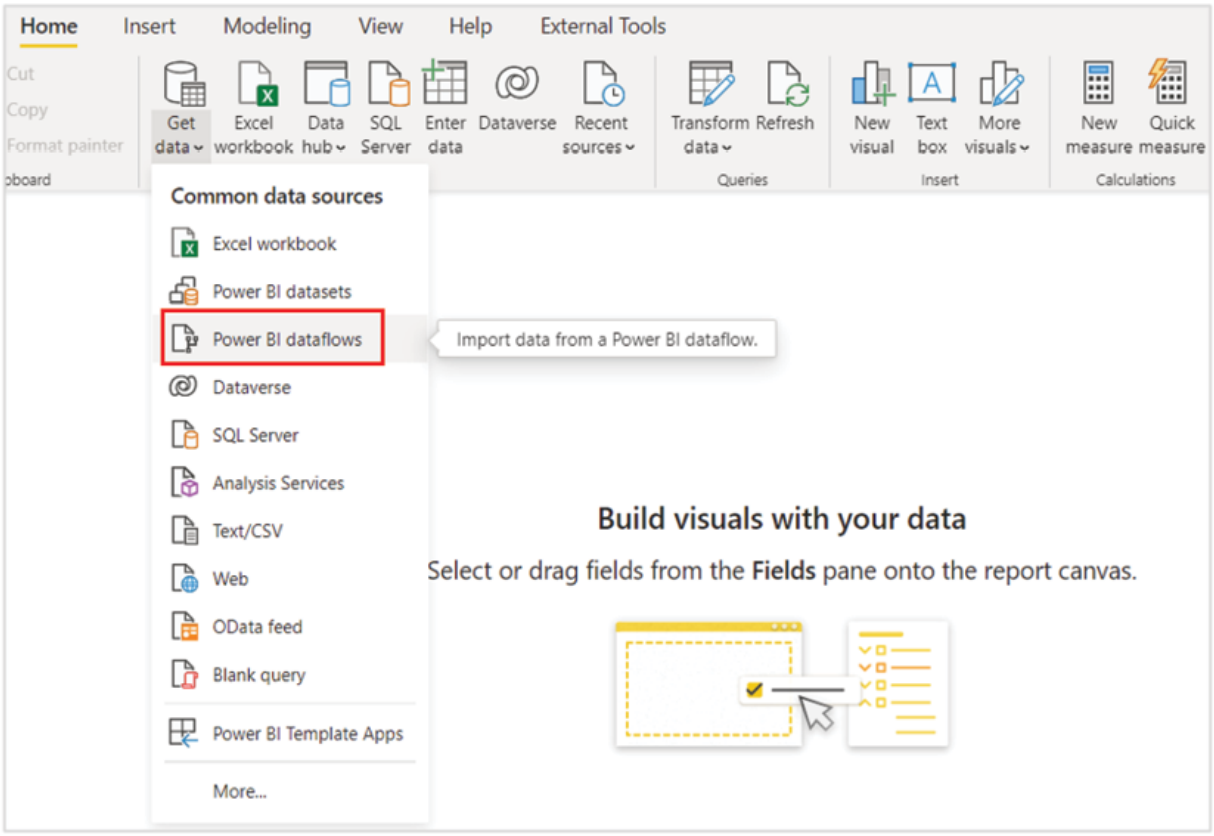



Figure Consuming data from a Dataflow in Power BI Desktop

Using a Dataflow, we can separate the Power Query or the ETL layer from the Power BI dataset or data model. This is particularly useful to perform common reusable transformations for multiple datasets. Instead of performing the transformations repeatedly on individual dataset tables, we can do it once on a Dataflow entity and then consume it from different datasets maintaining a single point of truth.

Another great use case of Dataflows is to enable authoring reports without having to provide reports authors access to the underlying data source. A Dataflow can be configured with all the required entities from a data source and then can be exposed to the report authors by providing them with appropriate workspace accesses. However, whether to use a Dataflow or not is a design decision that needs to be made considering the business use case that we have in hand.

## Conclusion

In this chapter, we explored the concepts involving Power BI Service in terms of managing Power BI contents in cloud. We understood how reports are stored as well as how to effectively work with them from a business user perspective. We touched upon the administrative capabilities of the service besides going through multiple security approaches like Row-level security and workspace-level security. We need data gateways to refresh data, mostly for on-premises data sources, and we have seen how to install and configure a personal data gateway. Finally, we saw how features like Apps and Dataflows enable report authors to choose from multiple design options and implement the one that fits best for the project!

In the next chapter, we will see how to work with large data volumes in Power BI, which is otherwise difficult to manage with the skills we have acquired.

## Knowledge check

Which of the following activities is allowed to perform with a free license?

Create a workspace

Publish a report to My workspace

Assign a data gateway to a dataset

Adding a workspace viewer

Which of the following option is not available while exporting data from a visual?

Export summarized data

Export selected data

Export data with current layout

Export underlying data

Which of the following DAX function is used to implement a dynamic RLS?

USERPRINCIPALNAME()

CALCULATE()

SUMMARIZE()

HASONEFILTER()

An App can be distributed to different audience groups:

True

False

All Knowledge Check answers are provided at the end of the book.

# CHAPTER 6

# Working with Large Data Volumes

## Introduction

We have already discussed the core concepts of Power BI in the previous chapters. In this chapter, we will focus on a very specialized topic, which is how to work with large data volumes in Power BI. Big data reporting is a pressing need across many industries these days, although whether a dataset can be termed as Big Data or not can be a topic of debate! In this chapter, we will learn about tools and techniques which can be used to process large amounts of data which otherwise is difficult using the concepts we have learnt so far. The concepts that will be discussed in this chapter are going to be crucial to be able to handle huge data volumes which is often the crux of reporting requirements nowadays. We will avoid technical jargons as much as possible and instead focus on applying the concepts hands-on, as we did throughout the book.

## Structure

In this chapter, we will discuss the following topics:

Power BI Premium features

Table partitioning

Configuring incremental refresh

Refresh management using SSMS

Managing datasets using the Tabular Editor

Metadata deployment using the ALM Toolkit

## Objectives

The objective of this chapter is to enable users to work with large data volumes themselves. This chapter illustrates the concept of delta load or incremental load which enables to refresh only a subset of data that is updating over time, instead of a full load or refreshing the entire historical static data that is present in the data source, during each and every refresh. This approach not only allows us to work with data warehouses where millions and billions of records are saved historically, but it also helps to bring down the data refresh time even for small to medium datasets by refreshing only what is needed as per the refresh schedule.

While we already discussed DirectQuery which eliminates the need of importing data in the first place, however, that only works well for non-interactive reports where we do not need to slice and dice the report too often. Otherwise, applying DirectQuery on a large data volume can have a negative impact on the report responsiveness and hence on the user experience. Using incremental an alternate solution can be devised for similar scenarios and by end of this chapter, readers should be able to configure an efficient incremental refresh policy for the Power BI datasets on their own!

Power BI premium features

Although incremental refresh for a Power BI dataset can work on a pro workspace as well; however, for large data refreshes, it is recommended to have premium capacity which brings in a lot of flexibility in terms of maintaining the data models on Power BI Service.

Once a workspace is assigned to a premium capacity, a diamond icon appears just beside the workspace name, as shown in Figure

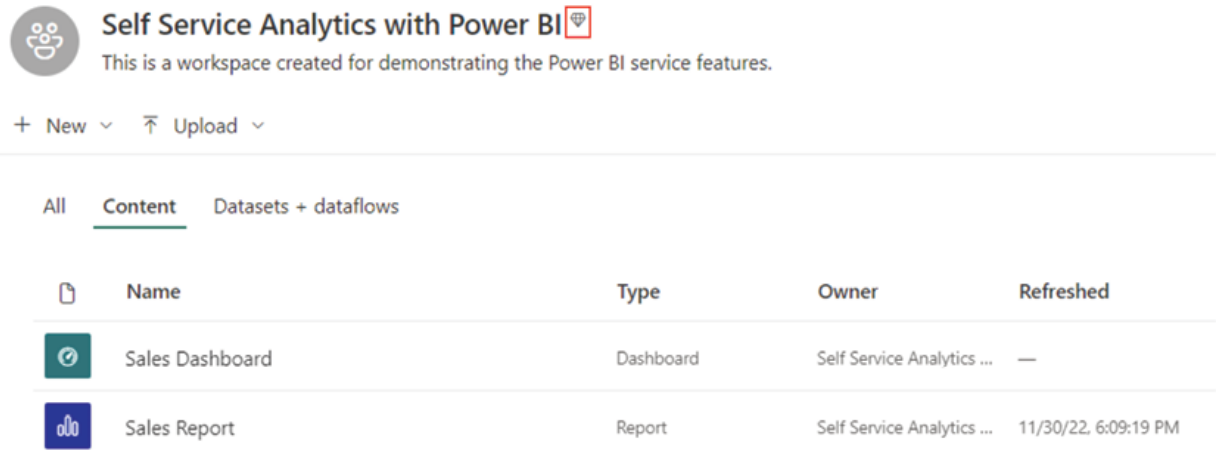



Figure Power BI Premium workspace

In a premium workspace, enabling a Large dataset storage format ensures that the dataset size would be limited only by the premium capacity size or by the maximum size allowed by the administrator. The large dataset storage format can be enabled for all datasets in a premium workspace by following Settings | Premium | Default storage as illustrated in Figure

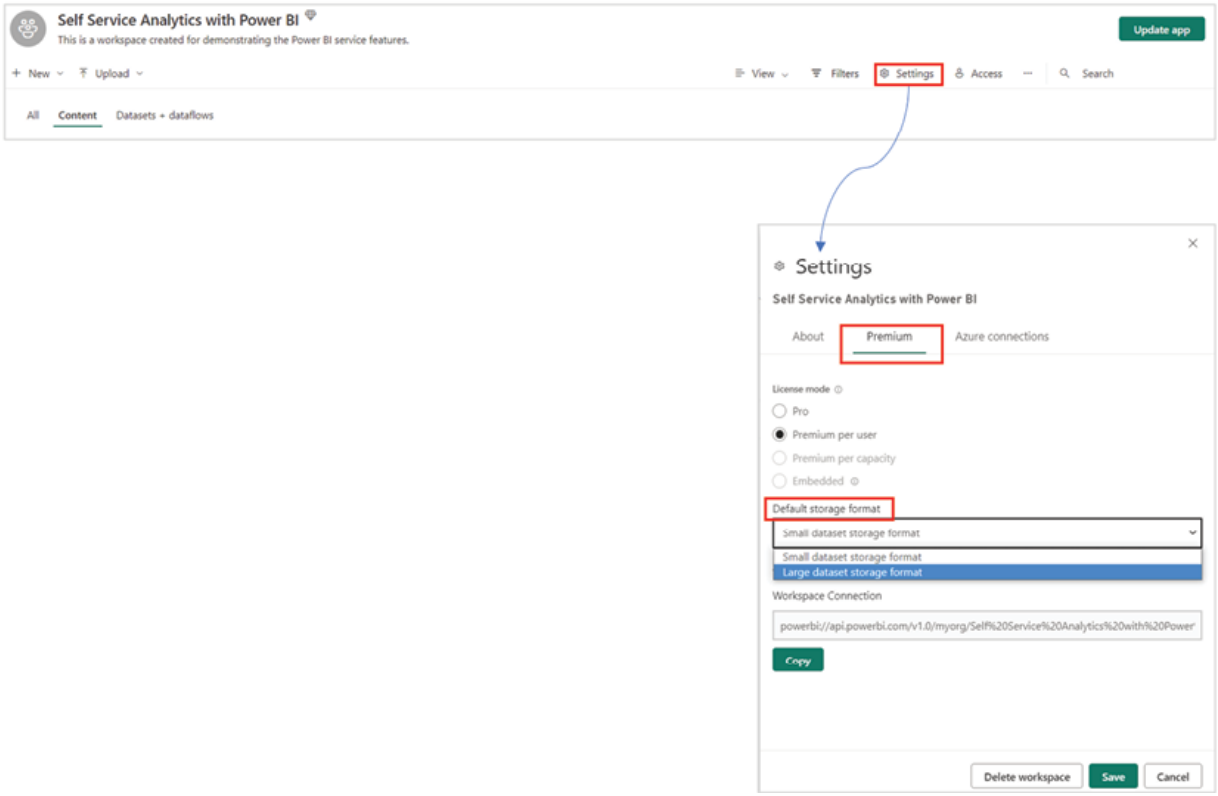



Figure Storage format for workspace

Alternatively, the storage format can be configured for individual datasets as well, from dataset settings by following More options | Settings | Large dataset storage where the option can be turned off or turned on, as shown in Figure

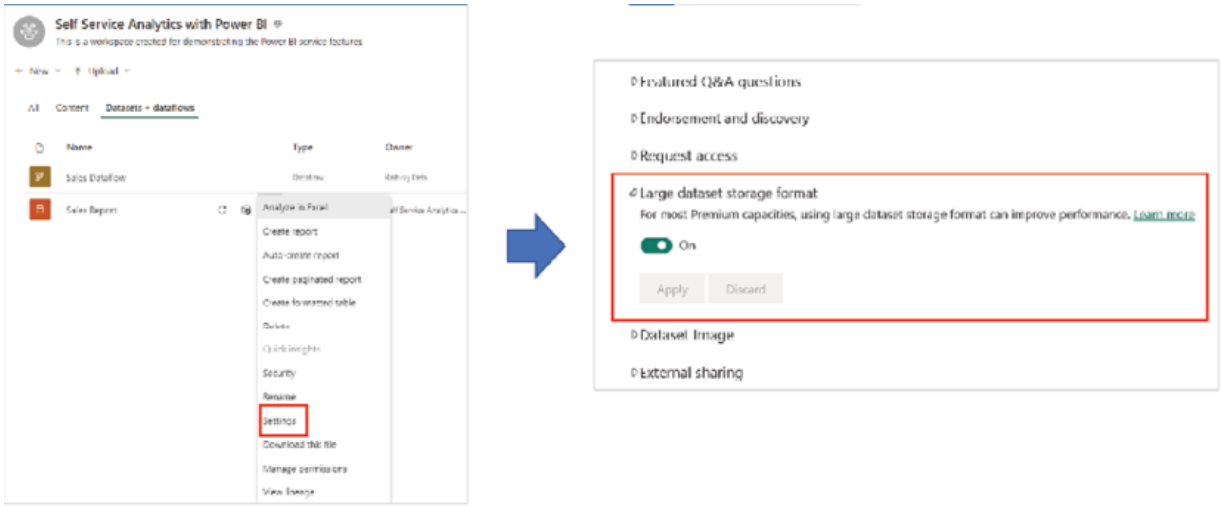


Figure Storage format for the individual dataset

Probably the most important feature that comes with Power BI Premium in terms of working with large amounts of data is the accessibility of XML for Analysis An XMLA endpoint allows client applications to connect to the underlying Microsoft Analysis Services engine of the platform that manages Power BI datasets and workspaces.

By default, XMLA endpoints are enabled for read-only operations which means external applications can only query a dataset. For external applications to perform write operations, the XMLA endpoint property must be set to Read Write using the Admin portal of Power BI Service.

Few most common external applications that can access Power BI Premium datasets through XMLA endpoints are SQL Server Management Studio SQL Server Tabular DAX ALM Toolkit etcetera.

Just like the Storage format settings, XMLA endpoint connection details for a premium workspace or a Premium Per User license can be copied by following the workspace Settings | Premium | Workspace as shown in Figure

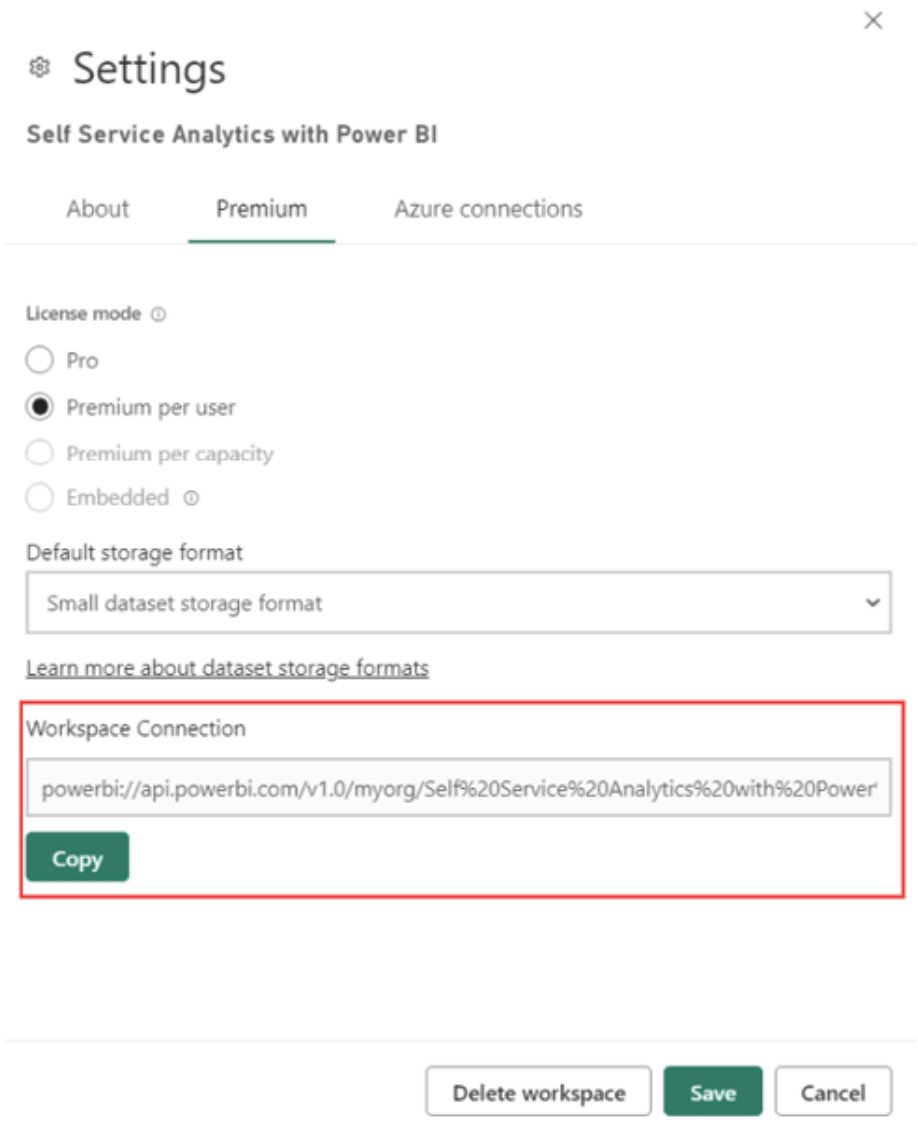



Figure XMLA endpoint for a workspace

Once we have the connection URL, the underlying analysis services engine can be connected using any application which supports connecting Power BI workspaces and datasets, after successful authentication. When connected, the Power BI workspace can be considered as the server and the individual datasets as databases. Using XMLA many critical actions can be performed, including refresh management, source code maintenance and so on, which we will explore throughout the rest of the chapter.

## Table partitioning

When a Power BI dataset gets published to the service, each table in the model has only one partition by default, which holds all the records for that table. If the table grows over time, like a fact table of a star schema, the table can eventually have a very large number of records stored.

During data refresh, this entire data gets truncated and re-loaded every time for the table, which can be extremely resource heavy and causes inefficiency. Using an incremental refresh policy, a table can be partitioned, and the daily refresh can import data only for the recent partitions as configured in the policy, while the rest of the old partitions' data can be archived. This ensures a faster data refresh while including the latest changes at the data source in the query results.

Power BI Service dynamically creates partitions in a table based on the incremental refresh policy that has been configured for it, during the first refresh of the model. However, these partitions are not visible on Power BI Service and can only be managed through the XMLA endpoint using external tools. After publishing a Power BI dataset having an incremental refresh policy to Power BI Service, when the dataset is refreshed for the first time, based on the Date/Time parameters RangeStart and the partitions are created automatically. We will explore the use of these parameters in detail in the next section of the chapter.

Partitions are created and named based on period granularities like Months, and So, first daily partitions are created; once a full month is completed, then monthly partitions are created, and so on. For example, if we create an incremental refresh policy for our fact table to refresh the last 2 days' data on a daily basis and store or archive the last 2 years' data then 4 daily partitions will be created along with 2 yearly partitions in case we refresh the model today January 2023), as shown in Figure

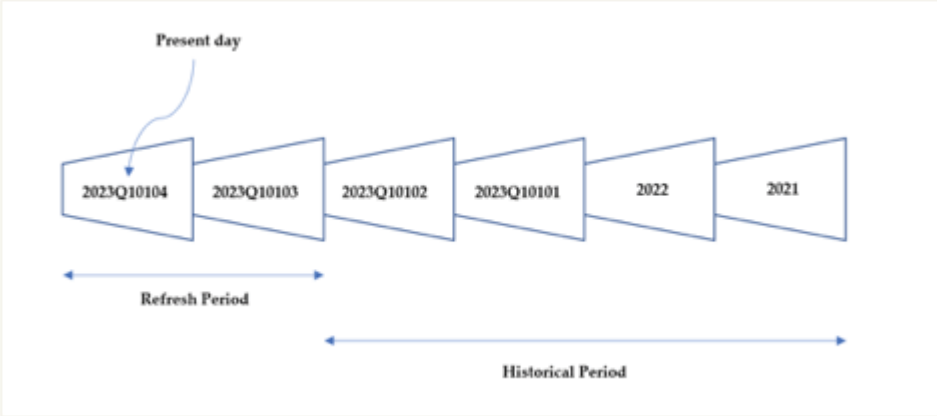



Figure Partitions created based on incremental refresh policy

The latest partition named 2023Q10104 represents January 2023, where 2023 is the year, Q1 is the first quarter, 01 is the month and 04 is the day. No quarterly or monthly partition would get created as per the period granularity, as no full quarter or month has been completed yet in 2023.

As whole periods are complete, partitions are merged. For example, on the first day of a new month, all daily partitions of the previous

month are merged into a monthly partition. On the first day of a new quarter, all three previous monthly partitions are merged into a quarterly partition. Likewise, on the first day of a new year, all four previous quarterly partitions are merged into a yearly partition.

Once partitions are created, with each subsequent refresh, only the partitions belonging to the Refresh period would be refreshed with updated data. Both the Refresh period and the Historical period follow a rolling window pattern. With every new daily partition created, the partitions which no longer are part of the Refresh period become part of the Historical period and are archived. When a partition is no longer part of the Historical period as defined by the policy, that partition gets removed from the Power BI dataset. Over time, historical partitions become less granular as they get merged. At any given point in time, a dataset always retains the total number of partitions in accordance with the refresh policy defined for an incremental refresh.

Now that we have understood what table partitions are and how they are created based on the incremental refresh policy defined for the table, let us explore the steps involved to configure an effective incremental refresh policy.

## Configuring an incremental refresh

For the incremental refresh to take effect, at first, we should be aware of a few key considerations:

An incremental refresh only works for data sources which support query folding.

The incremental refresh policy needs to be based on a Date/Time field which should ideally be an operational field the value of which does not change over time. Otherwise, it can cause issues like duplicating the same record in different partitions, as the partitions for an incremental refresh are created based on the Date/Time field which has been used in the policy. We would understand this in detail once we actually configure a policy and go through an example.

Once a dataset with an incremental refresh policy is published to the service, the .pbix file cannot be downloaded back. This makes sense because the published dataset can grow so big that it might not be practical to download it in the first place.

If we republish the original Power BI Desktop file, it will remove the existing partitions along with the data stored in them.

Although a dataset with incremental refresh can be published to a pro workspace, it is advisable to use premium capacity when implementing incremental refresh, especially for the above two reasons. Using XMLA

endpoints available with a premium, the published datasets can be maintained on the server itself using Tabular Model Scripting Language scripts, as well as any schema-related change can be deployed directly to the server using external tools like ALM Toolkit and Tabular All these features we would explore briefly in the upcoming sections of the chapter.

Let us now implement an incremental refresh policy on the same fact_Sales table that we have been using. Only this time, the table is imported separately in a blank Power BI Desktop file named fact Sales from a SQL-based source which supports query folding.

For incremental refresh to work, the data needs to be filtered dynamically to include only the records that are required. To do this, first, we need to create two parameters with the reserved keywords RangeStart and as parameter names. The parameters need to have the data type and initially, we would load just some sample data to Power BI Desktop for configuring the policy.

The parameters can be created on the query editor by following Manage Parameters | New as shown in Figure

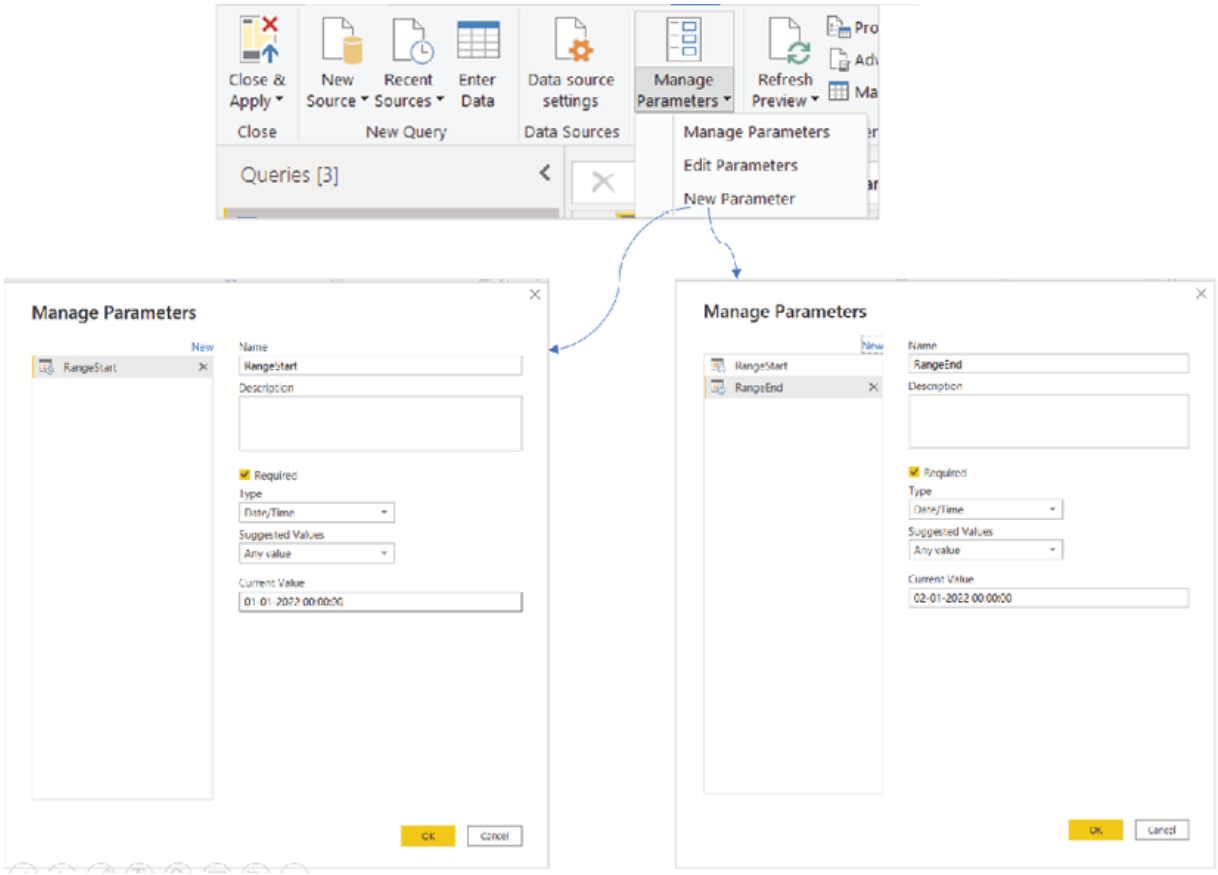



Figure Creating parameters for an incremental refresh

While the parameter Name and Type would always be the same, we can choose any Date/Time value for RangeStart and RangeEnd in the Current Value field. However, it is recommended to choose a small date range to initially load sample data to Power BI Desktop as eventually, these parameters would be used to filter data in the fact table.

After creating the parameters, the next step is to apply custom date filters on a Date/Time field of the fact table. This field is the one based on which the table would ultimately get partitioned in Power BI Service. At this point, the applied filter would select a subset of data which will be loaded to the Power BI Desktop.

We have the OrderDate column in our fact_Sales table, which is representing the date on which an order has been placed. This date is unlikely to change through the lifecycle of the record, even if the record gets updated for some other attribute, and hence qualifies for an incremental refresh. As this was originally a Date field, the data type has been updated to Date/Time to have the time component in it.

After selecting the field, following Date/Time Filters | Custom the Filter Rows wizard can be launched, as shown in Figure

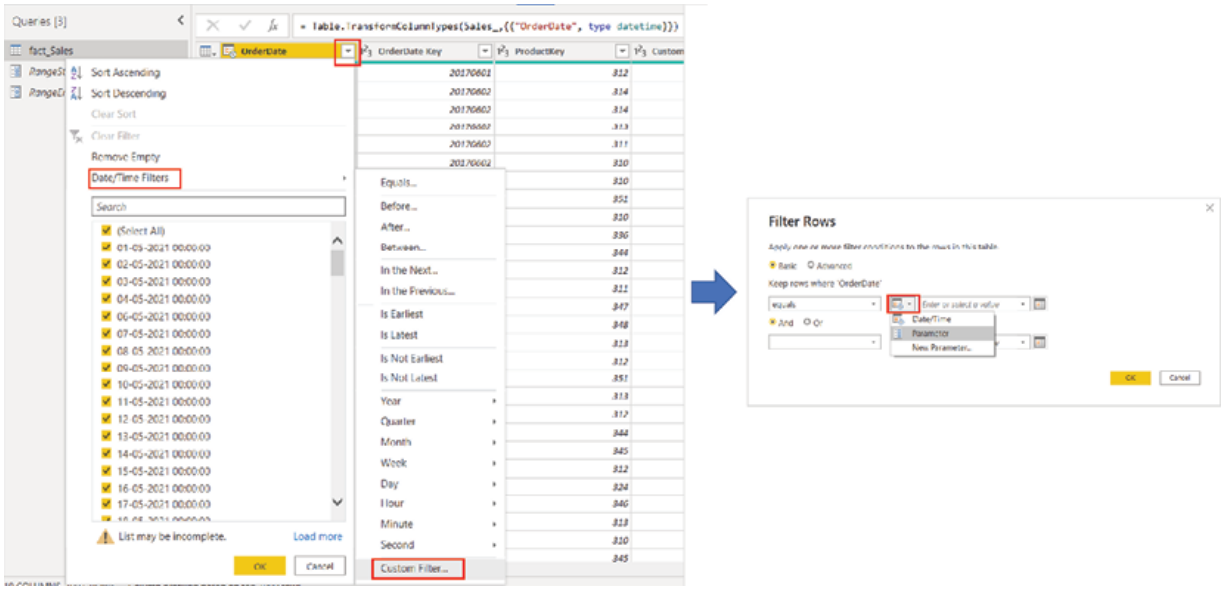



Figure Applying a custom filter

To specify the first condition in Filter we have to select is after or is after or equal to in the first drop-down, Parameter in the second drop-down and finally need to select the RangeStart parameter.

For specifying the second condition, if is after is selected in the first condition, then we need to select is before or equal In case we have selected is after or equal to in the first condition, then we need to select is for the first dropdown. After that we need to select the Parameter in the

second dropdown and finally the RangeEnd parameter in the third dropdown.

Figure 6.8 shows the filter conditions for the OrderDate column of the fact_Sales table:

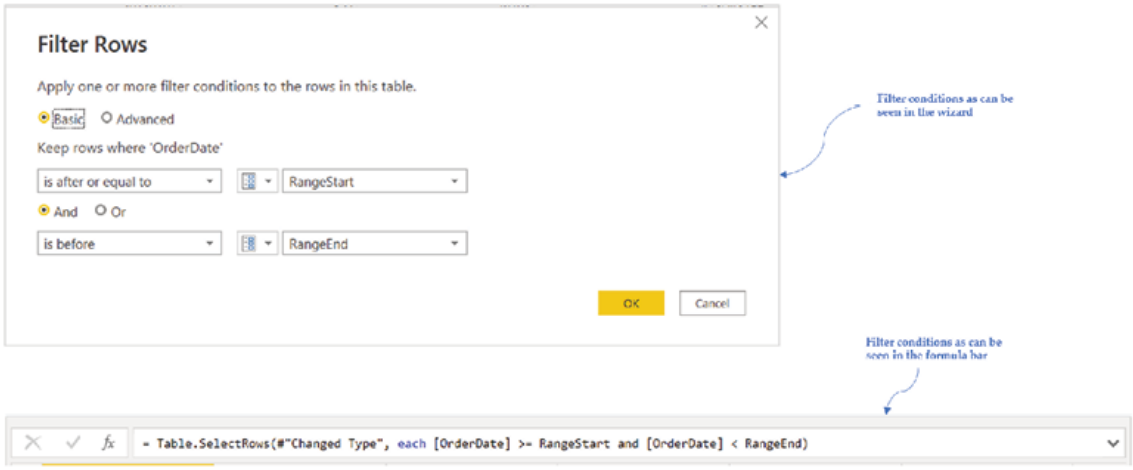



Figure Custom filter conditions

After applying the filters, the query loads data based on the date range as specified in the RangeStart and RangeEnd parameters. As we have configured the required steps in the query editor, let us now load the sample data to Power BI Desktop and proceed with configuring the refresh policy.

In Power BI Desktop, from More options of a table, selecting Incremental refresh launches the context menu for an incremental refresh, as shown in Figure

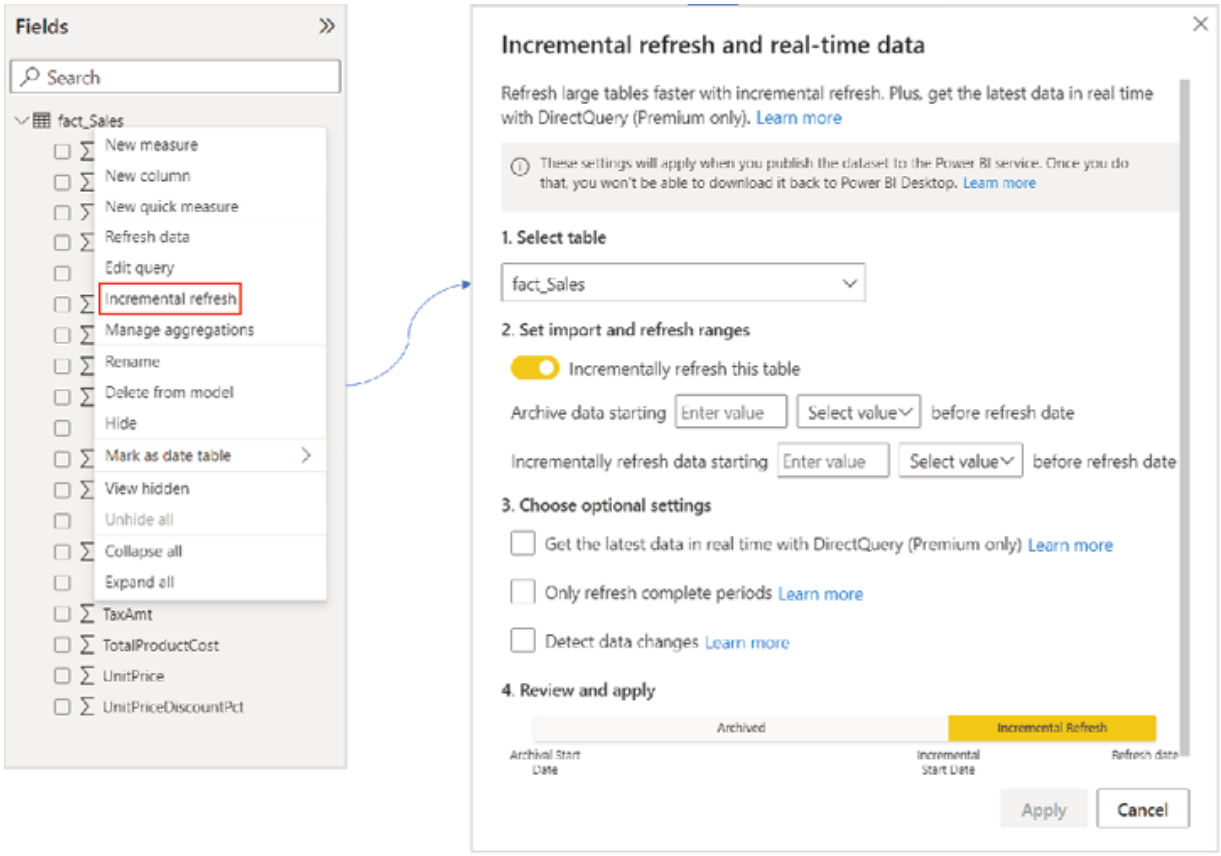



Figure Launching an incremental refresh menu in the Power BI Desktop

From the Select table dropdown, the table for which we want to configure an incremental refresh needs to be selected first, in this case, which is Then, from Set import and refresh the toggle for Incrementally refresh this table needs to be turned on. For archiving, the historical period needs to be selected in the Archive data starting section. Finally, the refresh period needs to be specified in the Incrementally refresh data starting section. This should conclude the mandatory configuration requirements for the policy, as shown in Figure 6.10 (the policy has been configured on January 2023):

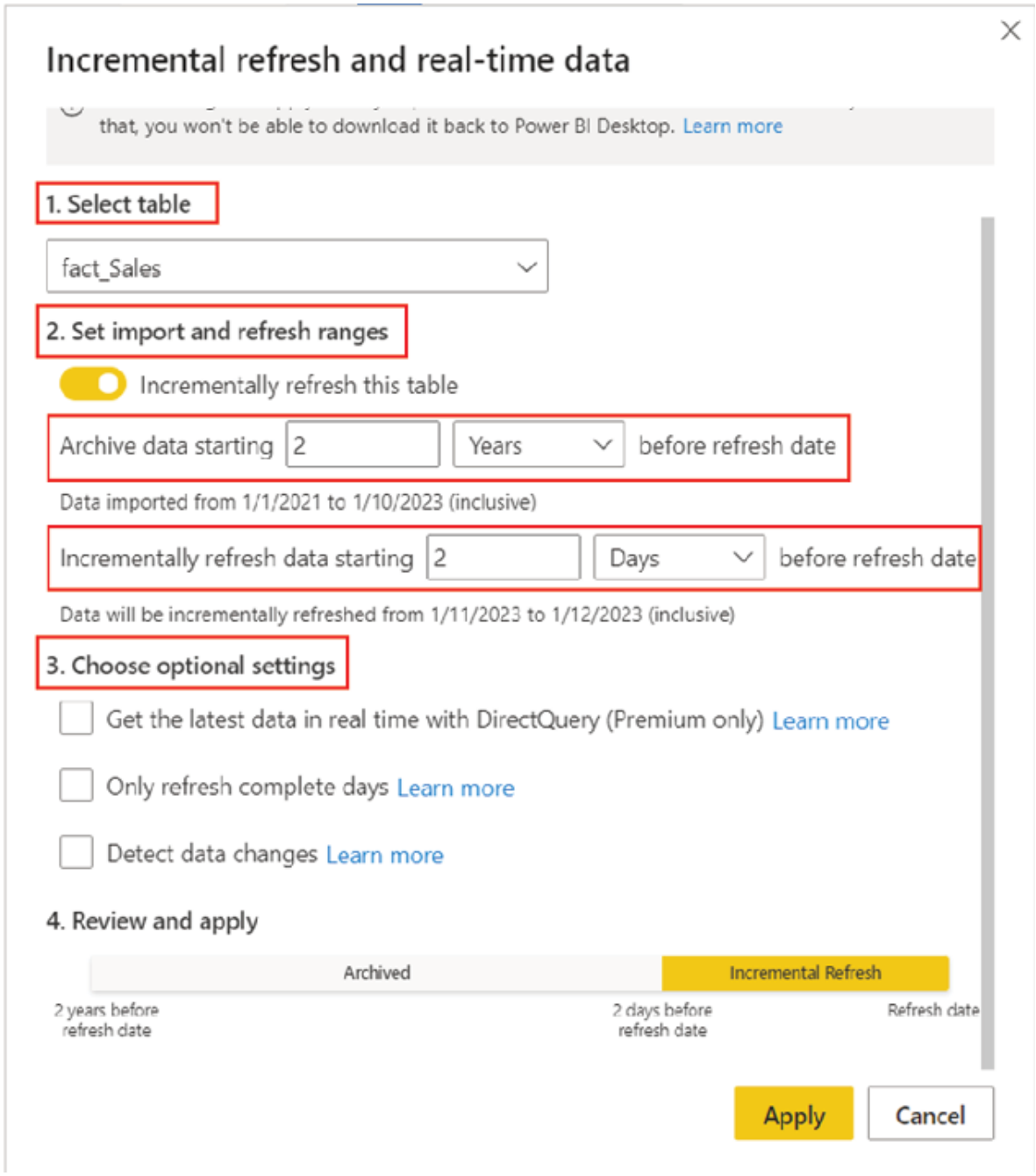



Figure Incremental refresh policy in Power BI Desktop

This policy would be applied overwriting the current Date/Time values that have been specified in the RangeStart and RangeEnd parameters, only after this model gets published to Power BI Service and an initial refresh operation is performed.

All records that fall under the historical period (here 2 years, as well as the refresh period (here 2 days), will be loaded into the model during the initial full refresh, while only the records belonging to the refresh period will the re-loaded and updated for each subsequent refresh, on an incremental basis.

There are a few optional configurations available for incremental refresh, as seen under the Choose optional settings section in Figure

The Get the latest data in real time with DirectQuery (Premium only) option can be used to configure an incremental refresh policy for a DirectQuery partition.

The Only refresh complete days option can ensure we refresh for full days only. If a whole day is not completed at the time of refresh, the data belonging to the day would not get refreshed.

The Detect data changes feature can make the refreshes faster by identifying the partitions for which the data has actually been updated since the last refresh and refreshing only those partitions. To implement this feature, we need another Date/Time column (not the one that has been used to configure incremental refresh) like an audit column the value of which updates whenever there is any change for the record. During each refresh, the maximum value for this column is evaluated for each partition. In case the value has not changed since the last refresh, that signifies no record for that partition is changed since the last refresh and hence the partition is skipped, or Power BI does not refresh that partition. This feature can potentially reduce the dataset or model refresh time significantly for suitable use cases.

Now that we have created the incremental refresh policy for the fact_Sales table, let us now save the .pbix file and publish the model to a premium

workspace in Power BI Service. After establishing the refresh mechanism, let us perform an initial on-demand refresh on the dataset. With this initial refresh performed on January 2023, we can expect Power BI Service to create all the required partitions in the table as per our incremental refresh policy and load data for both the historical period as well as the refresh For each subsequent refresh, only the refresh period should get processed. In the next section, we will validate whether it is working as expected or not, after connecting this dataset in Power BI Service, through the XMLA endpoint of the premium workspace.

Note: The Power BI Desktop file should be saved safely, as we would not be able to download the file anymore from Power BI Service for having an incremental refresh policy defined in it.

## Refresh management using SSMS

The SQL Server Management popularly known as SSMS, is an Integrated Development Environment to manage SQL infrastructures as well as Analysis services. SSMS is free to download from the official Microsoft webpage on the Internet and does not have any licensing requirements for installation and normal usage. To connect a Power BI Premium workspace from SSMS, we need SSMS version 18.9 or higher. The following Microsoft Learn page can be referred to for additional information about downloading and installing SSMS.

https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

Once SSMS is installed and launched, we need to select Analysis Services as the Server type and provide the XMLA endpoint of the premium workspace as the Server We already discussed how to get the XMLA endpoint for a premium workspace earlier in this chapter (refer to Figure The Authentication has to be Azure Active Directory - Universal with MFA while the User name should be the user with which we sign in to Power BI Service. The user needs to have read-write access to the workspace. Figure 6.11 shows the properties while connecting to the Analysis service:

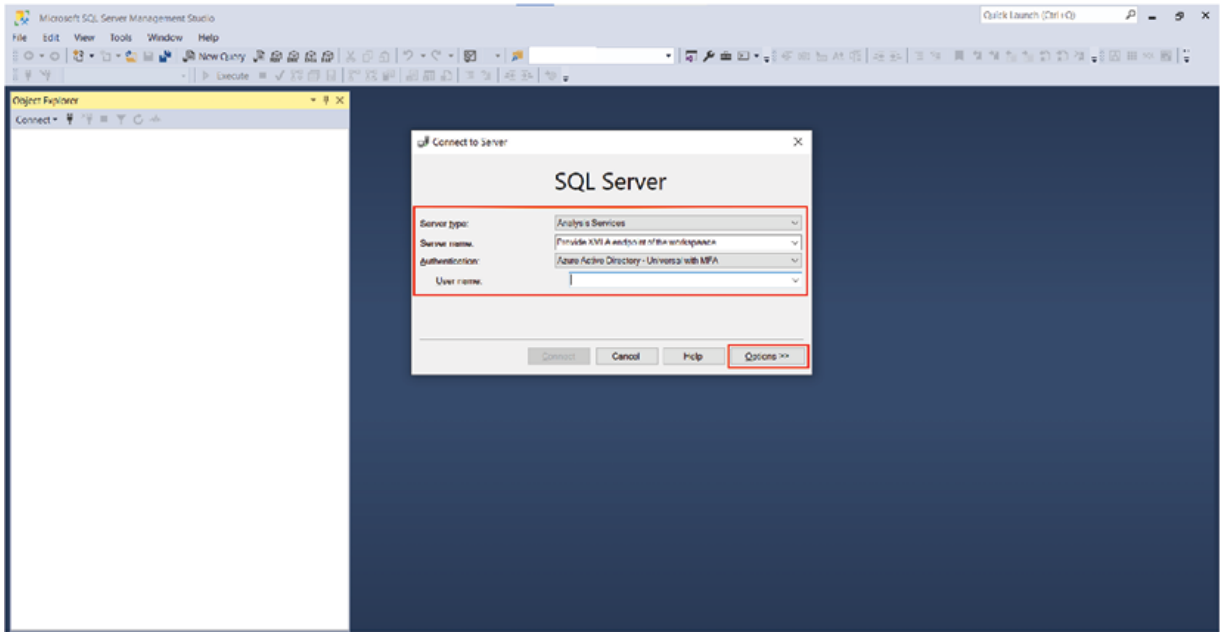



Figure Connecting Power BI Premium workspace using SSMS

Tip: In case of connectivity issues, try using the Options button as highlighted in Figure to search for a specific dataset in the workspace and connect to it, as shown in Figure

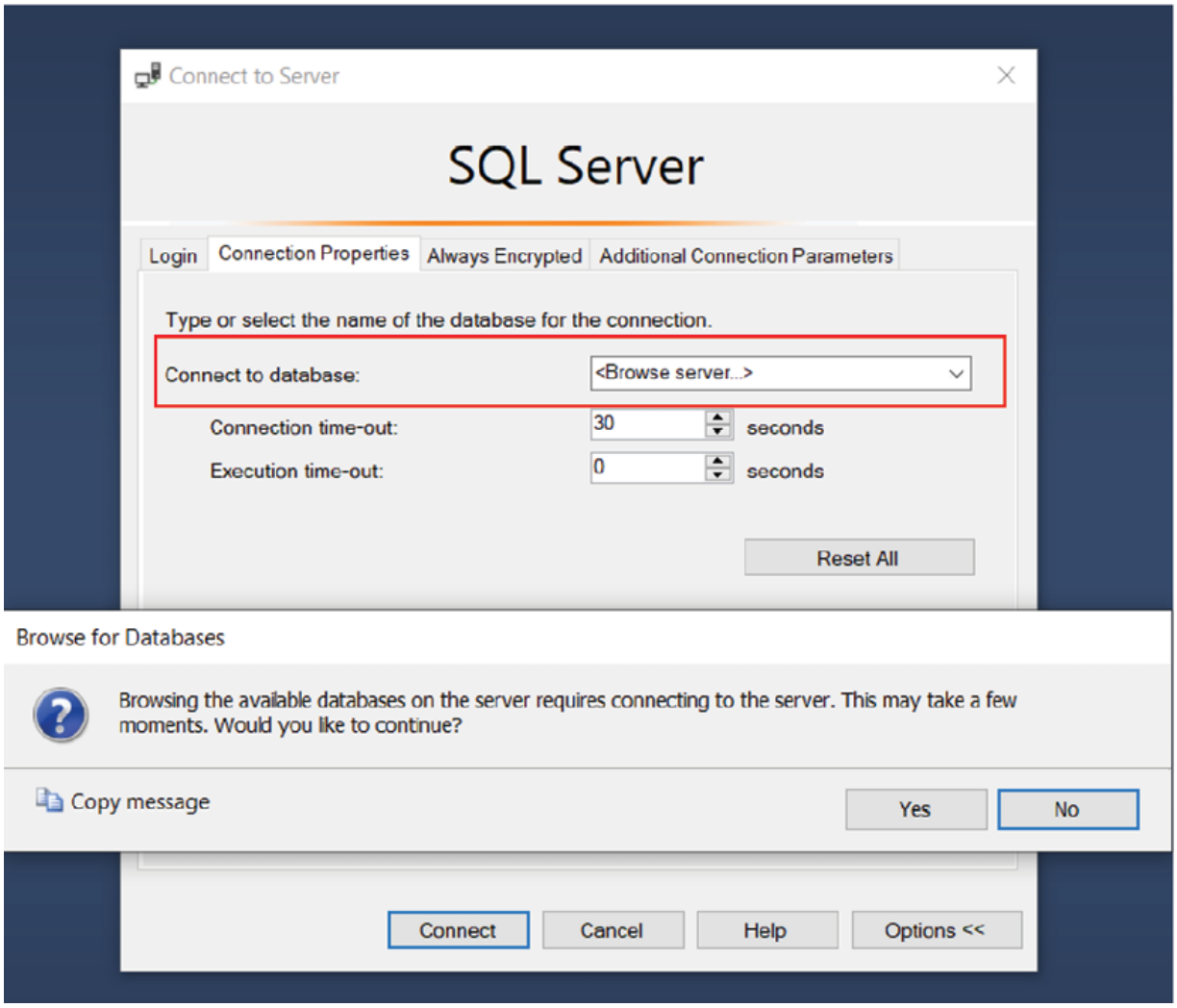



Figure Browsing for a dataset in the workspace using SSMS

Once connected, the workspace would get displayed as the server and the datasets belonging to the workspace would be listed as on the Object Explorer of SSMS. Expanding each dataset allows you to view all the tables that are present in the model, after expanding The Object Explorer for the workspace we have connected to is shown in Figure

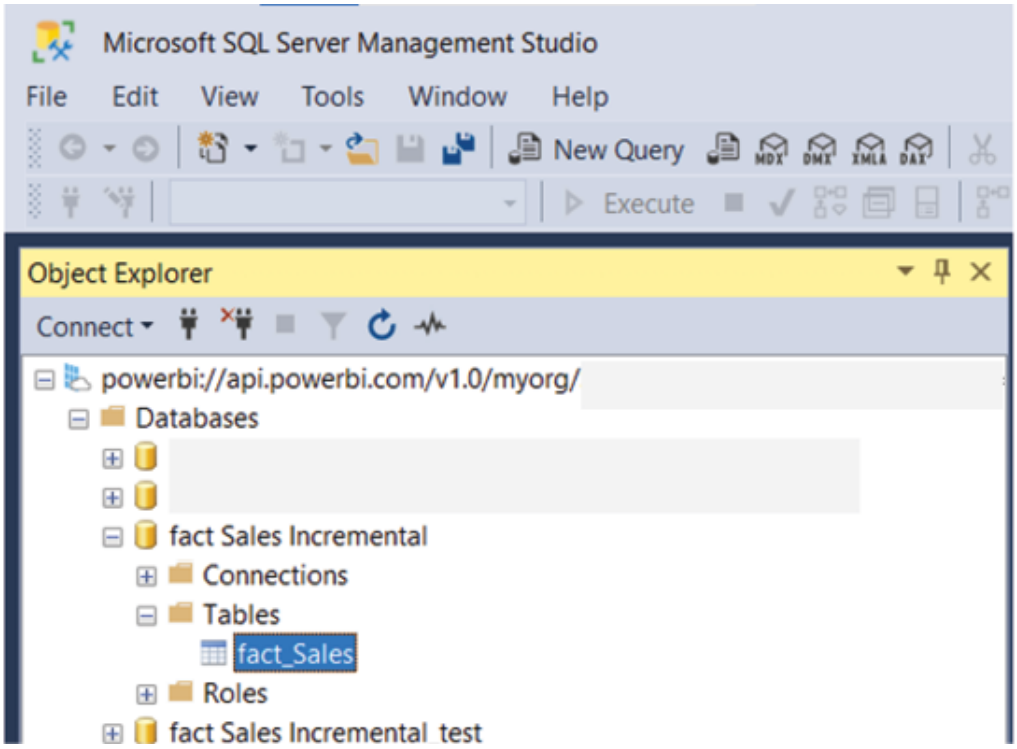



Figure Object Explorer in SSMS

Coming back to the incremental refresh, to validate the data loaded during the initial refresh of fact Sales we need to select the fact_Sales table in the Object right click and select which should display details of all the partitions available for the table as shown in Figure

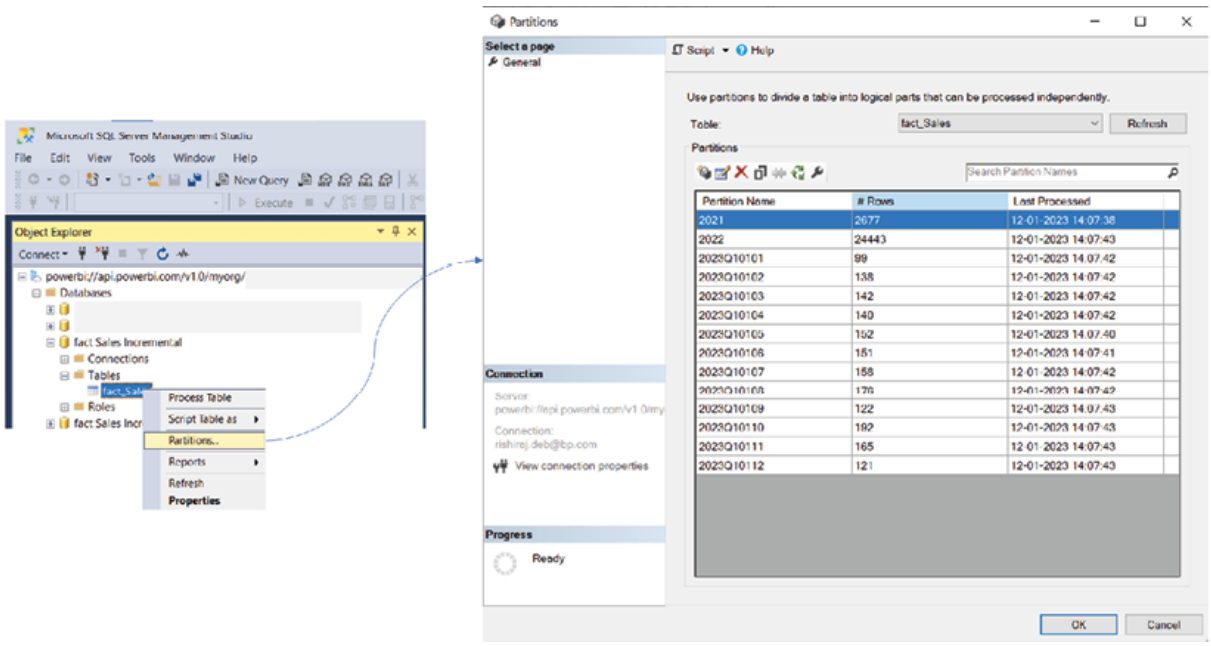


Figure Partitions created for fact_Sales during the initial refresh

As the model first refreshed on January 2023, we can see 12 daily partitions for January 2023, along with 2 yearly partitions for 2021 and 2022 created, per the incremental refresh policy defined for the table. Also, data has been loaded for all the partitions. In the Partitions window, the first column represents the name of the partition, the column has the number of records loaded for each partition, and the column indicates the last processing timestamp.

Let us now refresh the model again to check which partitions update for subsequent refreshes. If we have a closer look at the last processing timestamp of the existing partitions in Figure it can be seen that all the partitions were refreshed around the hour or 2 PM. After a successful refresh operation is performed at the hour, Figure 6.15 shows the Partitions window with the updated details:

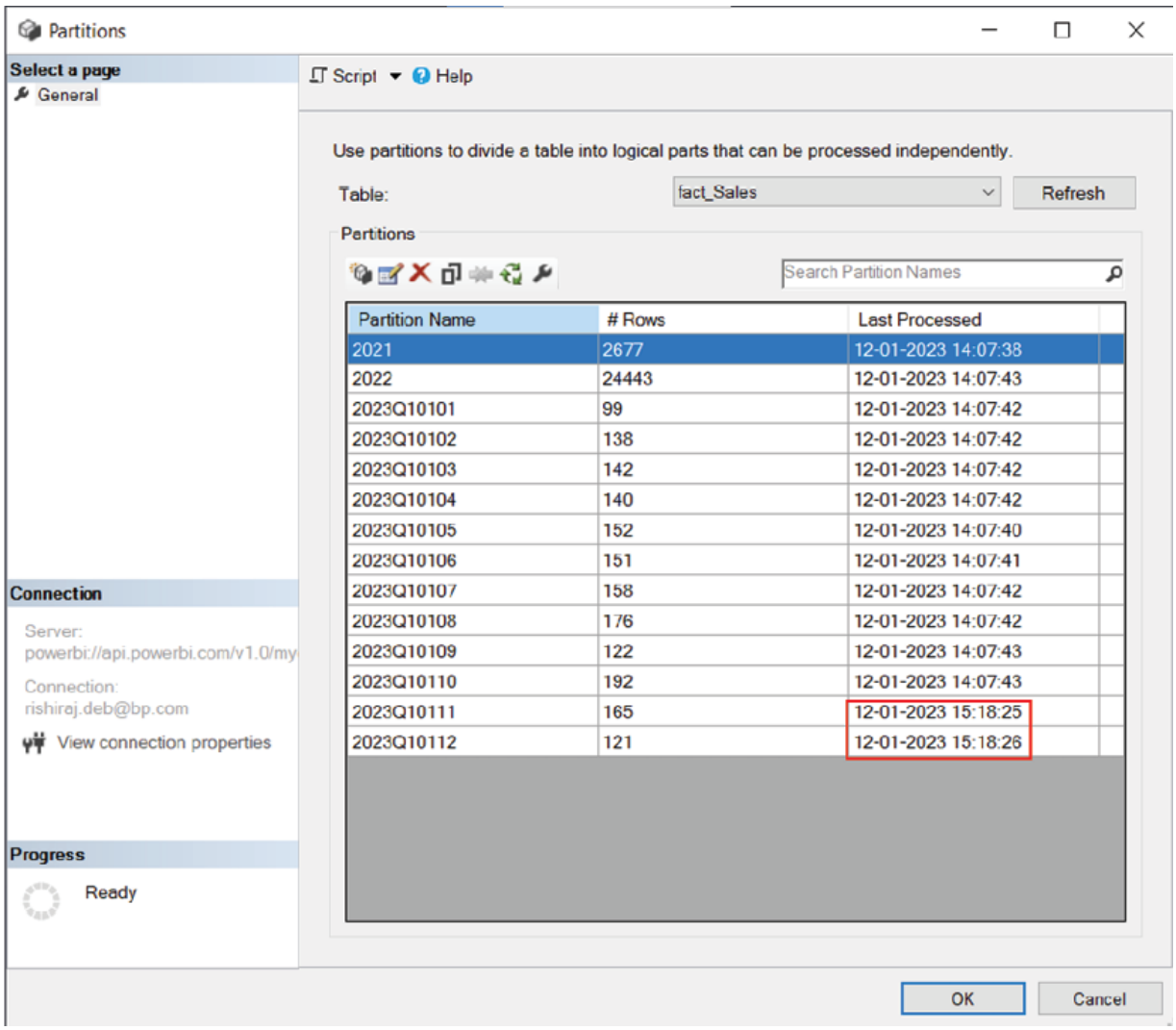



Figure Partition details after the second refresh

As we can see by the refresh timestamp, only the last two partitions 2023Q10112 and 2023Q10111 have been updated as these two fall under the Refresh as per our incremental refresh policy. The rest of the partitions have not been refreshed and hence, became archived; just as we expected.

We have already mentioned earlier that the field on which incremental refresh has been configured, which in our case is should not ideally change throughout the lifecycle of the record. Let us now understand the reason behind this in a bit more detail. Suppose there is an existing record having

an OrderDate of January 2023. That record would come under the 2023Q10111 partition, and any change of that record would get updated if we refresh the report today January 2023), as that partition falls under the Refresh period as of now. However, in case we refresh the report in future, that record would have already moved to a historical partition as per our refresh policy and become archived, hence, would not update anymore. So down the line, hypothetically, if OrderDate for that record changes, for example, to January 2023, the partition 2023Q10123 would have the same record again, just with an updated OrderDate value. This can cause duplication of the key fields for the same record in different partitions and eventually contradict the underlying model.

Apart from viewing the partition details, we can manually process individual partitions using the same Partitions window. Selecting any partition, and clicking on the Process button opens the Process Partition(s) window where we can select the processing The Process Data mode loads data into selected partitions. After selecting the processing clicking on OK launches the Data Processing window where the processing status can be monitored. Figure 6.16 illustrates the steps of processing individual partitions:

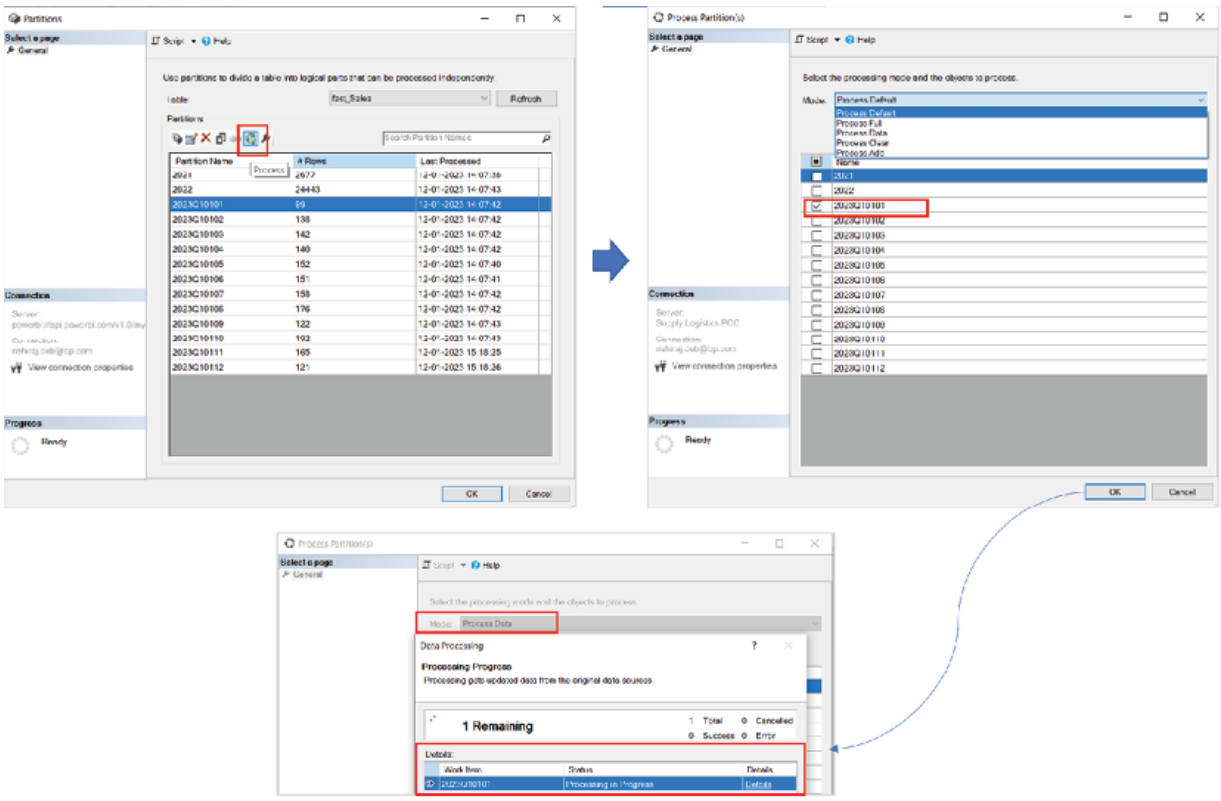


Figure Processing individual partitions

In case we process data for an individual partition of a dataset, we might need to recalculate the dataset to update the hierarchies, relationships and calculated columns, for the report to function properly. To recalculate a dataset, we need to right click on it and select Process then select Process Recalc as the Mode and click on as shown in Figure

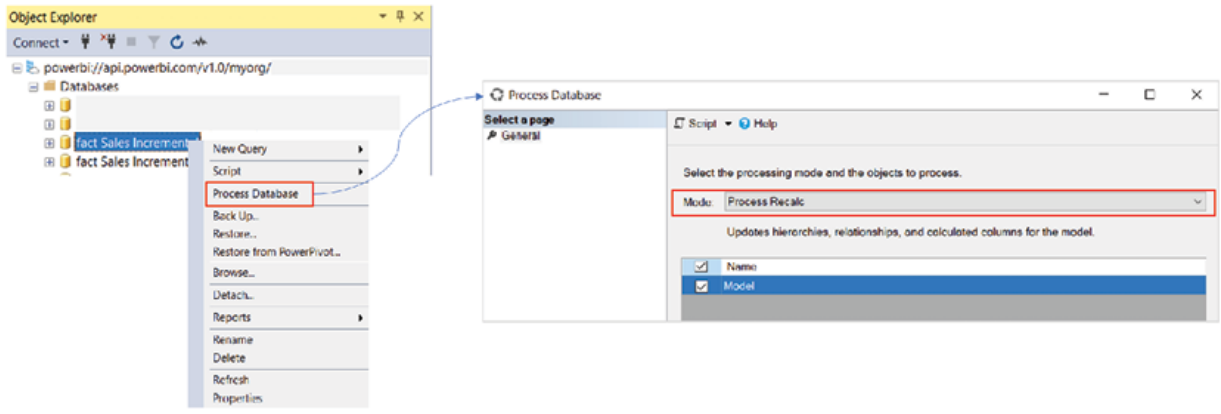


Figure Recalculating a dataset

The ability to load data manually into individual partitions is a huge help for processing large amounts of data. In the case of dealing with really large datasets, we may not be able to load all the data during the initial refresh after publishing the model, as premium datasets get timed out after five hours.

For similar scenarios, we can apply an additional filter to the Power Query editor in the Power BI Desktop, which will initially filter out all the records from the fact table. We can then publish the model and perform the initial full refresh just to create the partitions, without loading any data into them. After that, using Tabular Model Scripting Language scripts, we can remove the filter from the dataset and process individual partitions separately. Once we load all data in the partitions, from subsequent refreshes the model anyway would only process the data belonging to the refresh This technique, also known as allows us to work with big datasets in Power BI using SSMS.

To access the TMSL script for any dataset, after right clicking on the dataset, follow Script | Script Database as | CREATE OR REPLACE To | New Query Editor as shown in Figure

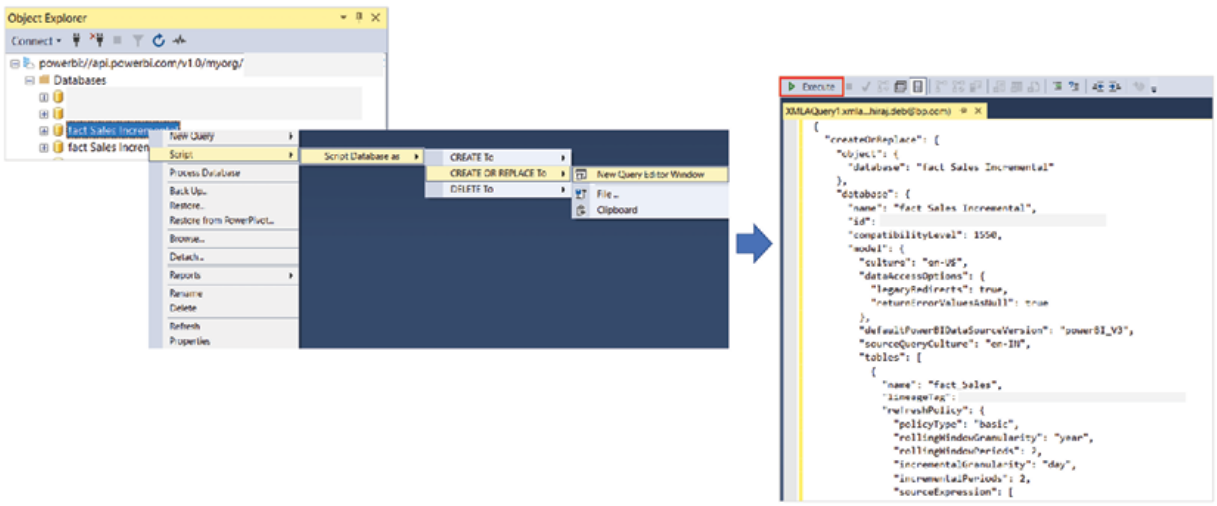


Figure Accessing the TMSL script for a Power BI model

Once we make any changes to the script and execute it, the changes would directly get applied to the model published on Power BI Service. This way we can maintain a published model on the service itself, without having to use the Power BI Desktop file at all. We will explore a few alternate options as well for maintaining Power BI datasets in the upcoming sections.

## Managing datasets using the Tabular Editor

The Tabular Editor is an open-source tool for managing tabular models which include Power BI datasets. Similar to SSMS, we can connect to underlying Power BI datasets using XMLA from the Tabular To perform any changes to the model or dataset published to the service, we would require read-write access to the hosting premium workspace. The tabular editor can be downloaded from the Internet for free, and once installed can be accessed from the External Tools section of the Power BI Desktop, as shown in Figure

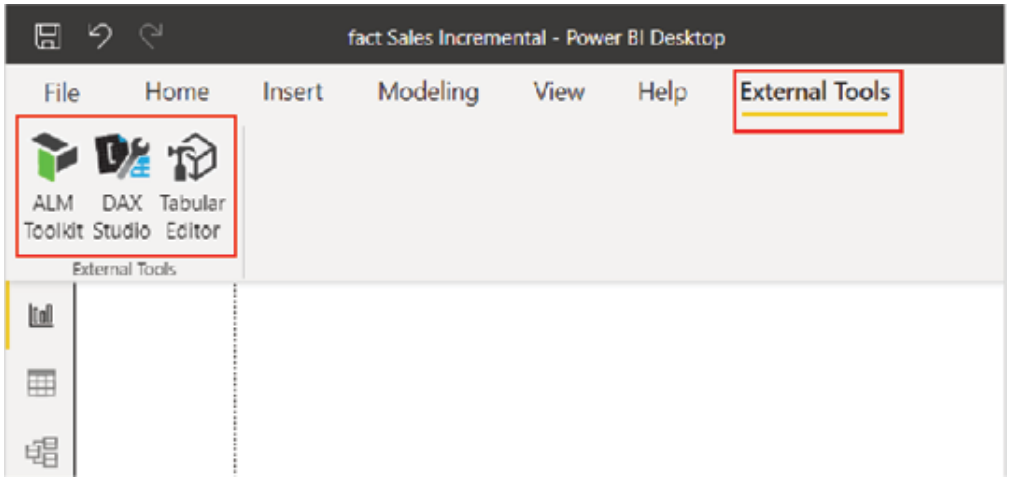



Figure 6.19: External Tools in Power BI Desktop

Just like the Tabular there are other tools that can be launched from the External Tools section of the Power BI Desktop after installation like ALM Toolkit and DAX as highlighted in Figure If we launch any of these tools from the Power BI Desktop, by default, they will be connected to the local Power BI model from which they are being launched. Alternatively, we can

launch the standalone tools separately and connect to a local Power BI model or a Power BI Service workspace using an XMLA

To connect to the server, after launching the Tabular we can use the Open a Tabular Model from an existing database button, provide the XMLA endpoint as the Server and authenticate using the Windows Integrated or Azure AD login option, as shown in Figure

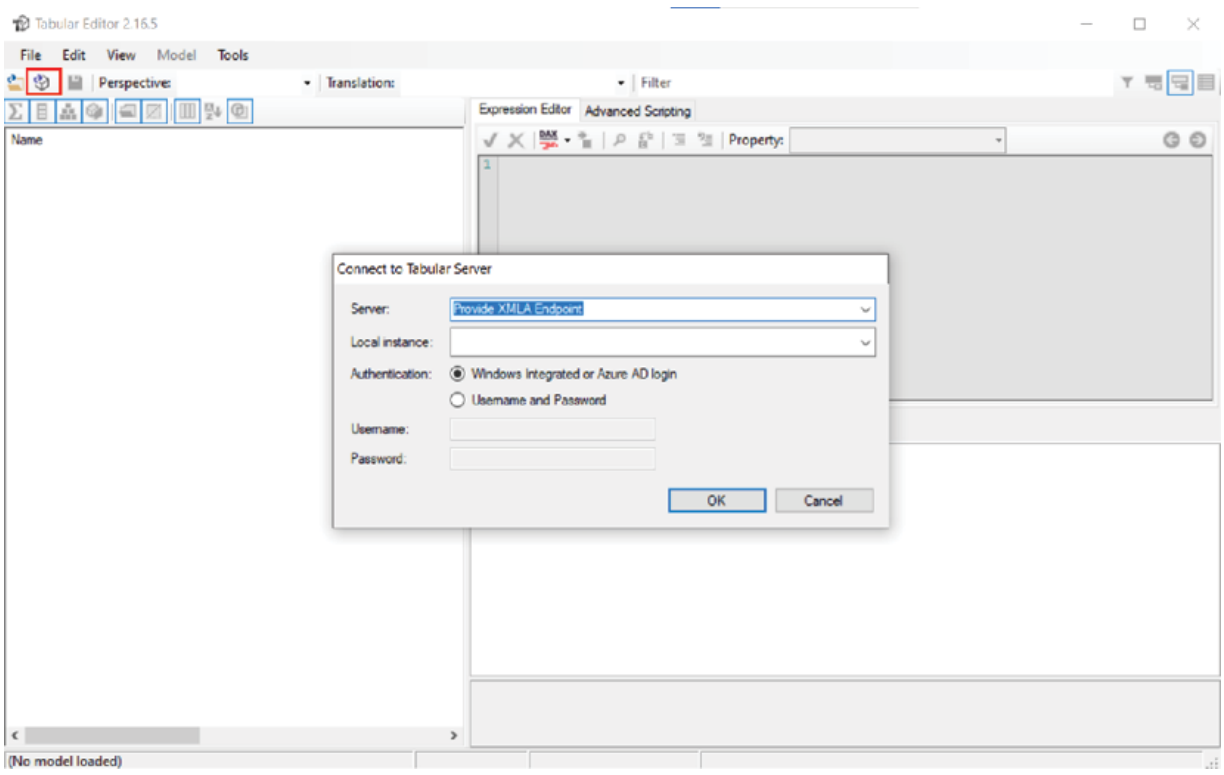



Figure Connecting Power BI workspace from the Tabular Editor

After successful authentication, we can choose a dataset from a list of available datasets in the workspace, using the Choose Database window. Once connected to the dataset, the metadata of the model would be visible on the left-hand pane, as shown in Figure

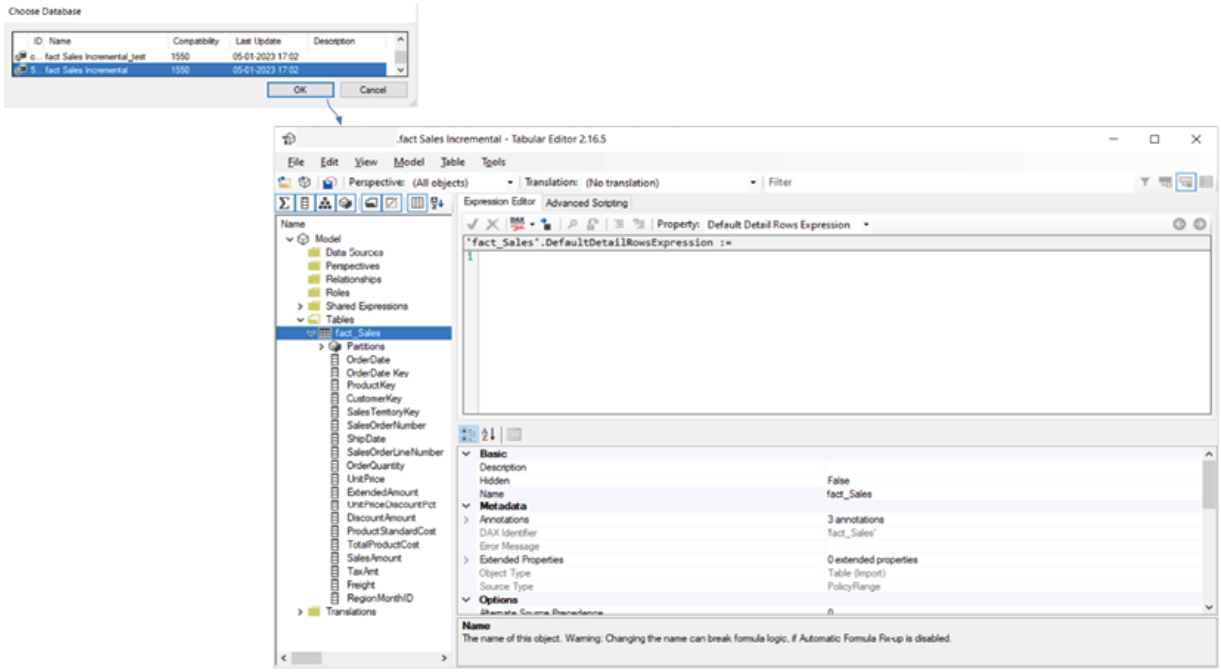



Figure Metadata of Power BI model in Tabular Editor

The Tabular Editor is an excellent tool in terms of maintaining Power BI datasets. For example, let us just hide the key fields of our model on the service, so that those do not remain visible anymore for the end users on the report. To do this, we can select a field after expanding the fact_Sales table, then from the properties of the field, the property Hidden can be set to True as shown in Figure

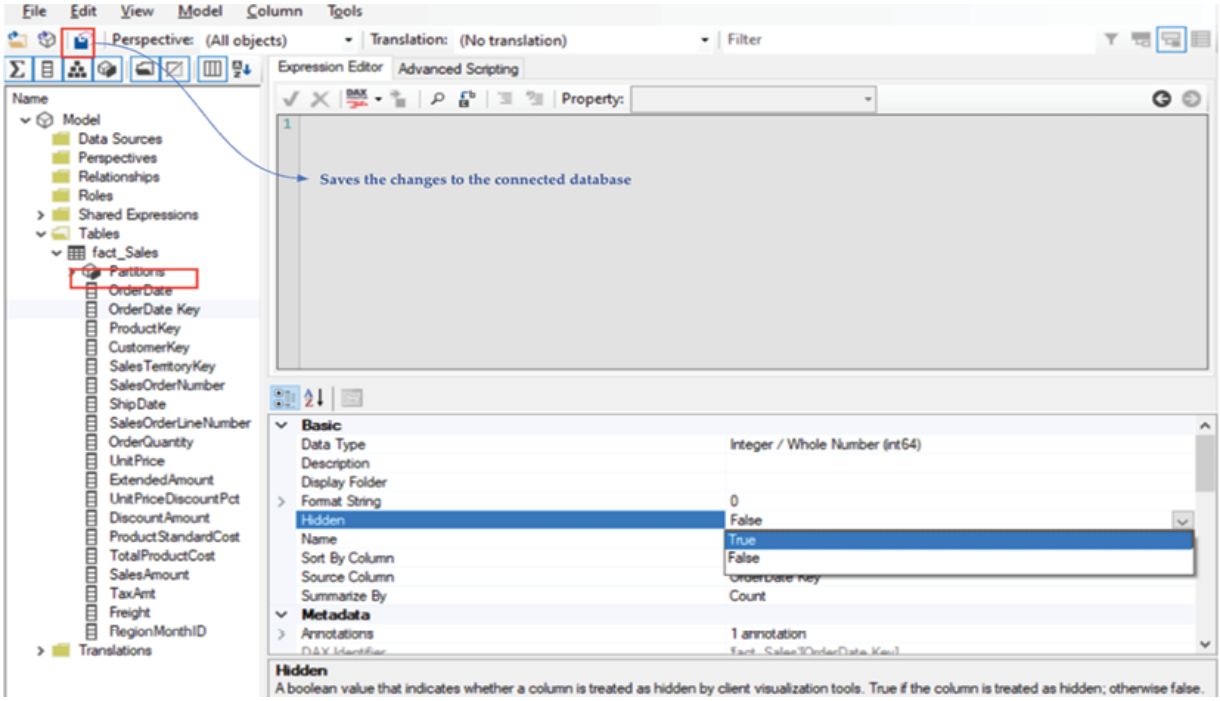



Figure Updating a field property in Tabular Editor

After performing the required changes for all the fields, selecting Saves the changes to the connected database will deploy the changes directly to the dataset in Power BI Service.

If we now edit the report on Power BI Service, the key fields would not be visible anymore under the Fields as shown in Figure

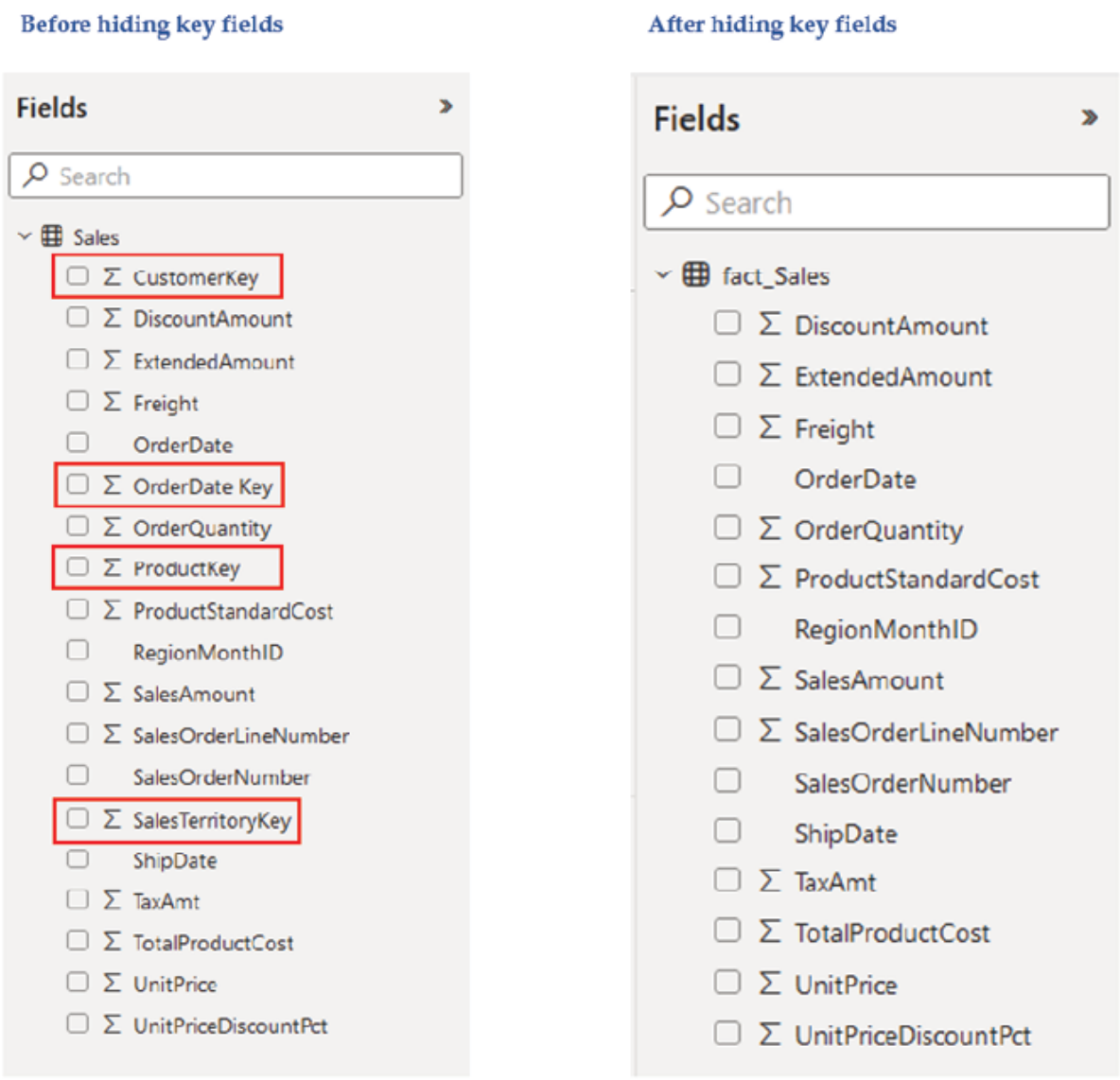



Figure Changes deployed using Tabular Editor reflecting in Power BI Service

The Tabular Editor is an extremely powerful tool to update the properties of objects and build, maintain and conveniently manage tabular models.

## Metadata deployment using the ALM Toolkit

Another free external tool, which can make deployment easier by enabling schema-only The ALM Toolkit is a schema comparison tool, using which we can deploy any change performed on the local Power BI Desktop file to the dataset published on the service, by comparing the metadata or schema of the two models.

Let us understand how it works with an example. As we have the fact Sales Incremental report published on the service, in case we want to create a new measure like we can create the measure on the local copy of the Power BI Desktop file that we initially published and then deploy the change directly to the service without having to republish the file, using ALM Toolkit.

If we launch the ALM Toolkit from the External Tools section of a Power BI Desktop file, by default the .pbix file from which it has been launched is selected as We need to select the which should be the dataset to which we want to deploy the changes. As always, the XMLA endpoint of the workspace needs to be provided and the dataset needs to be selected to connect to it, as shown in Figure

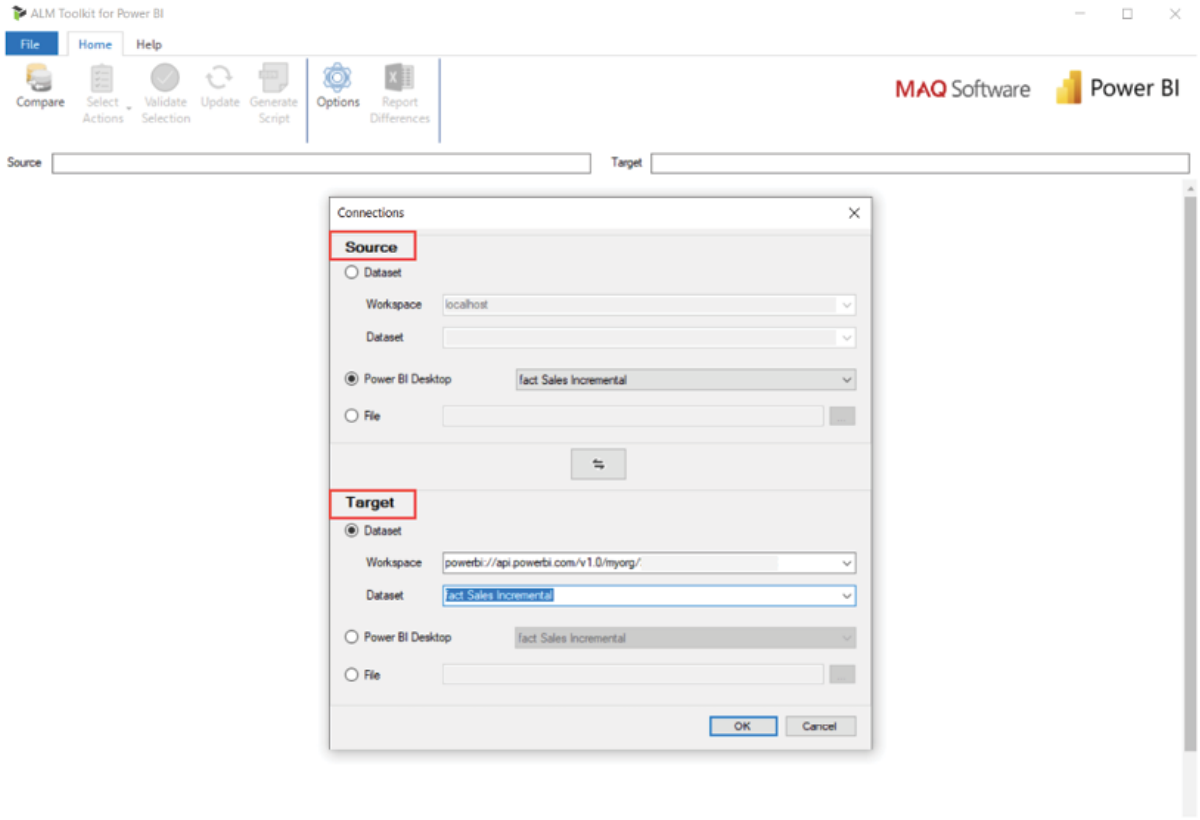



Figure Connecting Source and Target from the ALM Toolkit

Once we click on the ALM Toolkit performs a schema comparison and displays the schema definitions of both the Source and the Target model, on the left-hand and right-hand side, respectively, as shown in Figure

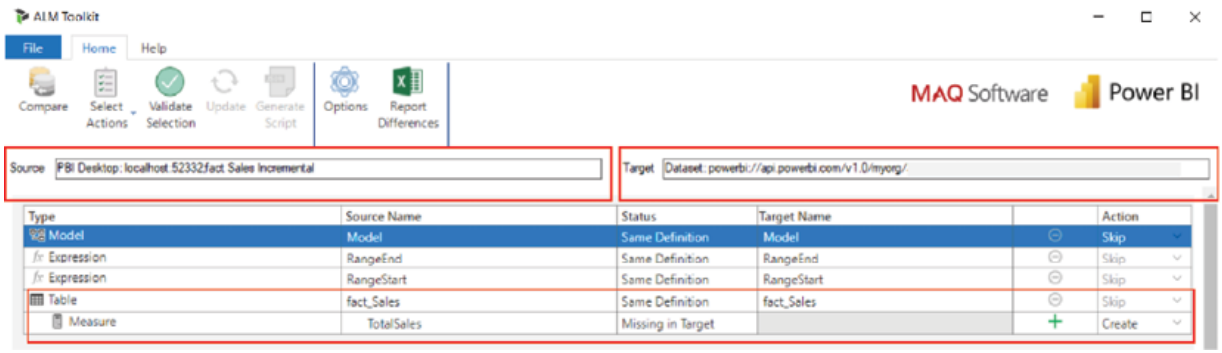



Figure Comparison results in ALM Toolkit

We can click on Select Actions and select Hide Skip Objects with Same Definition to view only the differences. By doing that, in our case, we can see the only difference between the Source and the Target is the TotalSales measure in the fact_Sales table which we created in the local .pbix file and is missing in the If we do not want to deploy a specific change in definition, the record can be skipped on the Action drop-down on the right-hand side. To deploy any change, the selected line item(s) need to be validated first using the Validate Selection option and then can be deployed using the Update option. The process is illustrated in Figure

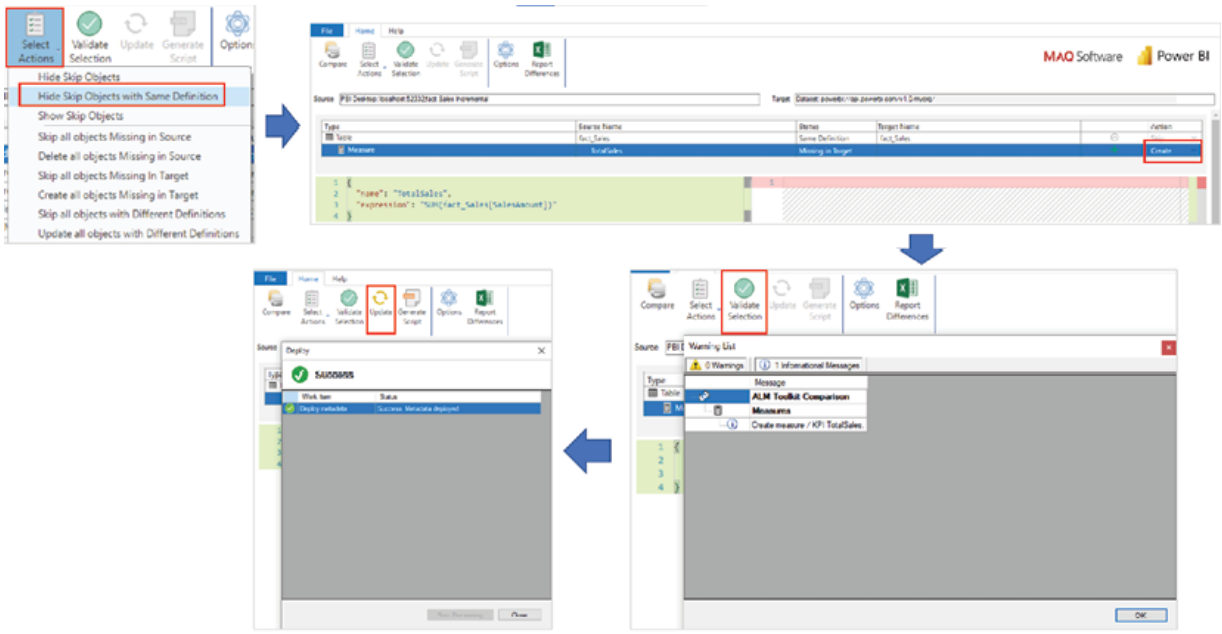



Figure Deploying metadata using ALM Toolkit

If we edit the report on Power BI Service, we would be able to see the measure now under fact_Sales as shown in Figure

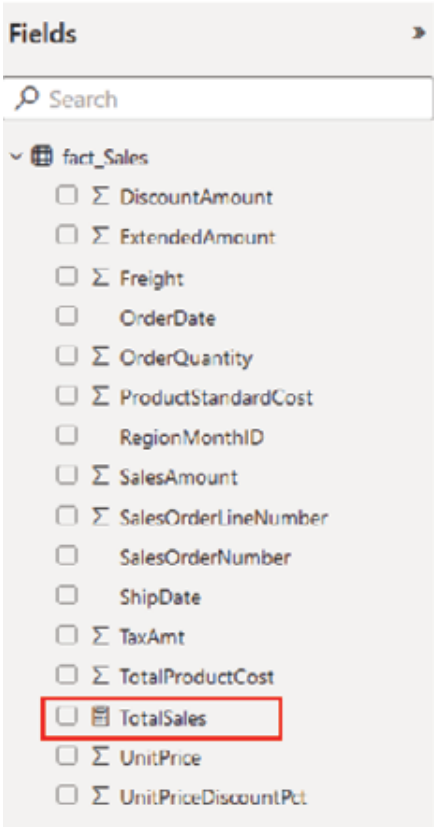



Figure Measure TotalSales successfully deployed using ALM Toolkit

Note: If we want to just deploy metadata and do not want to process any data while deploying, we need to make sure to select Do Not Process as the Processing which is available under as shown in Figure

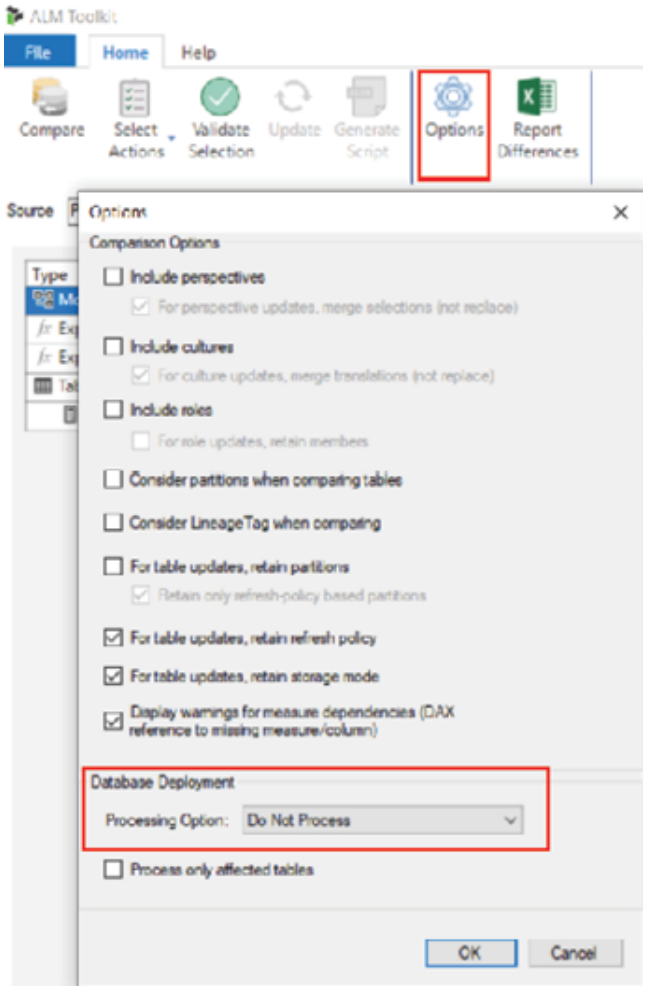



Figure Different options in ALM Toolkit

There are multiple comparison and deployment options available under Options in the ALM Toolkit, which can be used for different deployment requirements.

## Conclusion

In this chapter, we learned how to work with really large datasets in Power BI, using an incremental refresh. We explored in detail how concepts like table partitioning work and the key considerations that we need to keep in mind in the time of configuring an incremental refresh policy. The table partitions are not visible on Power BI Service and can only be viewed and managed using the XMLA endpoint of a premium workspace from tools like SSMS. We have gone through the refresh management process in SSMS as well as discussed how to access the TMSL script of a Power BI dataset. Finally, we saw how external tools like the Tabular Editor and ALM Toolkit can help to manage reports directly on Power BI Service. This knowledge should help readers to deploy an end-to-end enterprise solution using Power BI.

## Knowledge Check

Which of the following tools allows us to compare schemas between two Power BI models?

Tabular Editor

ALM Toolkit

SQL Server Management Studio

Partitions in a table get created during the initial refresh of the Power BI model:

True

False

Using XMLA endpoint, we can connect a Power BI workspace from external tools:

True

False

All Knowledge Check answers are provided at the end of the book.

CHAPTER 7

# DAX Reference Guide

## Introduction

In this chapter, we are going to explore some of the most commonly used Data Analysis Expressions that are available in Power BI. As we already have seen, DAX is extremely useful in enhancing the data model by introducing calculated tables, columns and measures, and this chapter should provide enough information about the common DAX functions including the syntax, parameters and return values with suitable examples. Apart from that, the functions would also be categorized based on their usage for easy navigation. The functions that we already have discussed in earlier chapters, would also be listed here with the syntax just for reference, however, we would not go into further details about them, as the applicability of those functions has already been demonstrated.

## Structure

In this chapter, we will discuss the following topics:

Aggregation functions

Date-Time functions

Filter functions

Information functions

Logical functions

Relationship management functions

Text functions

Table-specific functions

Time intelligence functions

Other functions

## Objectives

DAX has hundreds of functions, and the objective of this chapter is not to discuss each one of them, but rather to introduce readers to some selective functions which they can apply for building logic in common reporting scenarios. The Epilogue section of the book would have a link to the official documentation from Microsoft for DAX functions, which should be referred to for a complete list with detailed information. The expectation from the readers is that after going through the examples of this chapter, they would implement similar logic with their own set of data to get more familiarized and confident about DAX, and then explore other functions and variations themselves. Hopefully, this reference guide would help readers to derive much more insights about their data using DAX for the Power BI projects they are working on.

## Aggregation functions

Aggregation functions aggregate values for rows in a column or table and return a single scalar value. A few commonly used aggregation functions in DAX are:

SUM: Adds all numbers in a column

Syntax: SUM()

Parameter: Column containing values to sum up.

Refer to Chapter Data Modeling in Power BI for more details.

AVERAGE: Returns the average value or mean value of all numbers in a column

Syntax: AVERAGE()

Parameter: Column containing values to compute the average.

For example, for the fact_Sales table that we have used in the Sales the following DAX code computes the average value of SalesAmount for the measure

AvgSales =

COUNT: Counts the number of non-blank rows in the specified column. Supported data types: Number, Date, String.

Syntax: COUNT()

Parameter: Column containing values to be counted.

For example, the following DAX code calculates the total number of sales orders present in the fact_Sales table, for measure

SalesOrderCount1 =

COUNTA: Counts the number of non-blank rows in the specified column. Supported data types: Number, Date, String and Boolean.

Syntax: COUNTA()

Parameter: Column containing values to be counted.

COUNT and COUNTA should produce the same result for columns with the supported data types, except for Boolean or True/False columns that only COUNTA supports.

Similar to the measure SalesOrderCount2 also calculates the total number of sales orders in

SalesOrderCount2 =

COUNTBLANK: Counts the number of blank records in a column

Syntax: COUNTBLANK()

Parameter: Column containing the blank values to be counted.

To find if there are any blank records in the SalesOrderNumber column, the following expression can be used to measure

BlankOrders =

If no record meets the criteria, the function returns blank.

COUNTROWS: Counts the number of records in a table

Syntax: COUNTROWS(

)

Parameter: The table containing the records or rows to be counted.

The following expression counts the total number of records in the fact_Sales table for the measure

RowCount = COUNTROWS(fact_Sales)

DISTINCTCOUNT: Counts the number of distinct or unique values in a column

Syntax: DISTINCTCOUNT()

Parameter: Column containing the values to be counted.

The following expression counts the number of distinct orders in the fact_Sales table, for measure

DistinctOrders = distinctcount(fact_Sales[SalesOrderNumber])

MAX: Returns the largest value in a column, or between two scalar expressions

Syntax: MAX() or MAX(,)

Parameter: Either of the below:

Column containing the values to find the maximum.

returning single scalar values.

To find the largest order values in the following expression can be used in

MaxOrder = MAX(fact_Sales[SalesAmount])

MIN: Returns the smallest value in a column, or between two scalar expressions

Syntax: MIN() or MIN(,)

Parameter: Either of the below:

Column containing the values to find the minimum.

Expressions returning single scalar values.

To find the smallest order values in the following expression can be used to measure

MinOrder = MIN(fact_Sales[SalesAmount])

SUMX: Returns the sum of an expression, evaluated in a row context

Syntax: SUMX(

,)

Parameters:

Table containing rows for which the expression would be evaluated.

Expression to be evaluated for each row.

In the fact_Sales table, we have two fields OrderQuantity and If we want to calculate the total price for all orders, we need to multiply OrderQuantity and UnitPrice for each row or in a row context, and then aggregate. The following expression computes the total price for the TotalPrice measure:

```
TotalPrice =
SUMX(fact_Sales,fact_Sales[OrderQuantity]*fact_Sales[UnitPrice])
```

## Date-Time functions

Date-Time functions help to create calculations based on date and time. A few commonly used Date-Time functions are:

DATE: Returns a date in datetime format

Syntax: DATE(,,)

Parameters:

A number representing the year (a valid year should be more than or equal to 1900).

A number representing the month (valid values are 1 to 12, 1 represents January while 12 represents December).

A number representing the day (valid values are 1 to 31).

For example, the following DAX formula represents March 2023, as used in measure

MsDate = DATE(2023,3,7)

DATEDIFF: Returns the number of intervals between two dates

Syntax: DATEDIFF(,, )

Parameters:

The first date value.

The second date value.

The interval used when comparing dates. Valid values are:

SECOND

MINUTE

HOUR

DAY

WEEK

MONTH

QUARTER

YEAR

To find the difference of date (for example, March 2023) with the last year, the following expression can be used to measure

```
MSDateDiff =

VAR date1 =

  DATE ( 2023, 3, 8 )

VAR date2 =

  DATE ( 2022, 3, 8 )



RETURN

  DATEDIFF ( date1, date2, YEAR )
```

The function returns a positive value if Date2 is larger than otherwise, it returns a negative value. Here, the output of MSDateDiff should be -1.

EDATE: Returns a date that is the indicated number of months before or after the start date

Syntax: EDATE(,)

Parameters:

A date representing the Start_Date.

An integer representing the number of months we want to move forward or backward.

The following expression as used in measure 3MonthsAfter returns June 2023, which is a 3-month future date from March 2023:

3MonthsAfter = EDATE(date(2023,3,8),3)

CALENDAR: Returns a table with a single column named ‘Date’ containing a contiguous set of dates ranging between the start date and end date as specified, inclusive of the two dates.

Syntax: CALENDAR(,)

Parameters:

DAX expression representing the start date.

DAX expression representing the end date.

The following expression returns a single columned table having all dates of 2023:

DateTable = CALENDAR(date(2023,1,1),date(2023,12,31))

TODAY: Returns the current date, in a datetime format

Syntax: TODAY()

Parameter: No argument required.

The following expression returns the current date for the measure

CurrDate = Today()

UTCNOW: Returns the current date and time in UTC

Syntax: UTCNOW()

Parameter: No argument required.

The following expression returns the current date and time in UTC, for the measure

UTCDateTime = UTCNOW()

## Filter functions

The Filter functions help to implement dynamic calculations by manipulating the filter context. A few commonly used filter functions are:

CALCULATE: Evaluates an expression in a context modified by filters.

Syntax: CALCULATE( [, 1> [, 2> [, …]]])

Parameters:

The expression to be evaluated (should return a single scalar value).

Expression that defines a filter (optional and repeatable).

Refer to Chapter Data Modeling in Power BI for more details.

FILTER: Returns a table that represents a subset of another table or expression.

Syntax: )

Parameters:

The table (or expression returning a table) to be filtered.

An expression that is to be evaluated for each row of the table.

Refer to Chapter Data Modeling in Power BI for more details.

ALL: Returns all rows in a table, or all values from a column, ignoring any filter that might have been applied to the table or column respectively. ALL is mostly used as an intermediate function while performing other calculations.

Syntax: ALL(

or )
Parameters (optional):

Table for which the applied filters would be ignored.

Column for which the applied filters be ignored.

Let us understand it further in the context of a report. The following measure TotalSalesAcrossCat computes the total sales amount for ignoring any filter that might have been applied for the Category column of dim_Products filters fact_Sales on the ProductKey field):

```
TotalSalesAcrossCat =
CALCULATE([TotalSales],ALL(dim_Products[Category]))
```

The TotalSales measure is just summing up the sales amount in

```
TotalSales = SUM(fact_Sales[SalesAmount])
```

In case we display both the TotalSales and the TotalSalesAcrossCat measures in a report using cards, with a slicer for Category created from the dim_Products table, the two measures should show the same value when we do not select anything on the slicer. However, if we select a specific category on the slicer, the value of TotalSales should be filtered as per the slicer selection, while the TotalSalesAcrossCat measure should ignore the applied category filter and show the same value as before, as shown in Figure


Figure Ignoring context filter using ALL

When no argument is passed to it ignores all filters that may have been applied across the model, for the calculation involved.

ALLEXCEPT: Removes all context filters in the table except for filters that have been applied to the specified columns.

Syntax: ALLEXCEPT(

,)

Parameters:

The table for which all context filters are removed.

The column(s) for which the context filters must be preserved.

ALLEXCEPT is used as an intermediate function while performing other calculations. For example, the following expression retains all filters that have been applied on the Country column of table while ignoring all other context filters for the rest of the columns in the table:

TotalSalesFilterCountry =

CALCULATE ([TotalSales], ALLEXCEPT (dim_Territory, dim_Territory[Country]))

SELECTEDVALUE: Returns the value when the context for the specified column has been filtered down to only one distinct value, otherwise returns the alternate result as specified.

Syntax: SELECTEDVALUE(,)

Parameters:

The name of an existing column cannot be an expression.

(Optional) The value to be returned when the context of the specified column has been filtered down to zero or more than one distinct value. If not provided, BLANK() is returned by default.

One common use of SELECTEDVALUE is to capture the value that has been selected in a report slicer. For example, as we have created the slicer for Category from the dim_Products table (refer to Figure the following

expression as used in measure returns the value that has been selected in the Category slicer:

SlicerSelection = SELECTEDVALUE(dim_Products[Category])

Figure 7.2 shows the value of SlicerSelection using a card, in the context of the report:


Figure Capturing the selection of a slicer using SELECTEDVALUE

CALCULATETABLE: Evaluates an expression in a context modified by filters

Syntax: CALCULATETABLE(

,)

Parameters:

The table expression to be evaluated.

(Optional and repeatable) expressions that define filters.

The following expression returns a table containing only the customers having Management as the filtering the dim_Customers table:

```
CustomersMgmt =

CALCULATETABLE ( dim_Customers, dim_Customers[Occupation] =
"Management" )
```

CALCULATETABLE is similar to except for the fact that it returns a table instead of a scalar value.

## Information functions

The information functions look at the argument reference and provide information about whether the value matches the expected type. A few commonly used information functions are:

ISBLANK: Checks whether a value is blank, and returns either True or False

Syntax: ISBLANK()

Parameter: The value or expression to be tested.

For example, the following expression used in the calculated column returns a True whenever it encounters a blank value for and returns a False for all non-blank values:

BlankOrNot = ISBLANK('DAX TEST'[Column1])

Figure 7.3 shows the outcome of applying ISBLANK in

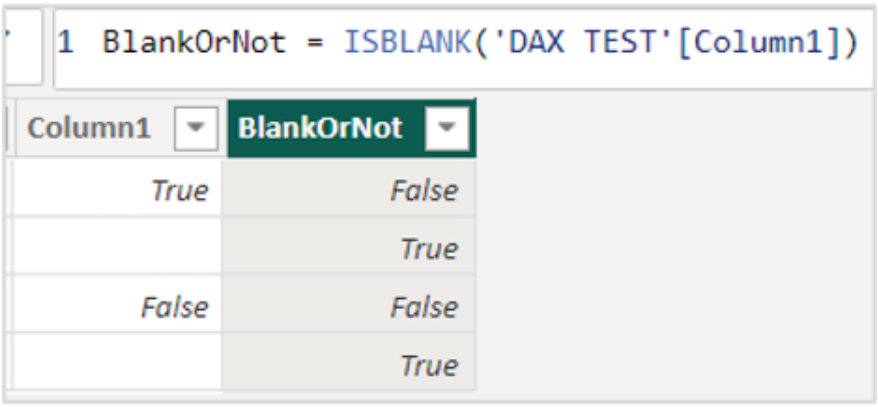



Figure ISBLANK returning True or False

ISERROR: Checks whether a value is an error, and returns either True or False

Syntax: ISERROR()

Parameter: The value to be tested.

The ISERROR function is used to check for errors, such as division by zero. For example, the following expression computes the cost-to-sales ratio for the measure Cost:Sales in fact_Sales and returns blank for any error. Otherwise, the ratio is returned:

Cost:Sales =

VAR ratio =

SUM ( fact_Sales[TotalProductCost] ) / SUM ( fact_Sales[SalesAmount] )

RETURN

IF ( ISERROR ( ratio ), BLANK (), ratio )

ISFILTERED: Returns True if the specified table or column is being filtered directly

Syntax: ISFILTERED(

or )

Parameter: The name of an existing table or column.

For example, the expression used in the measure FilteredOrNot returns a True in case we select a value in the Category slicer. If nothing is selected, which means no filters are applied to the category, then the measure returns as shown in Figure

FilteredOrNot = ISFILTERED(dim_Products[Category])

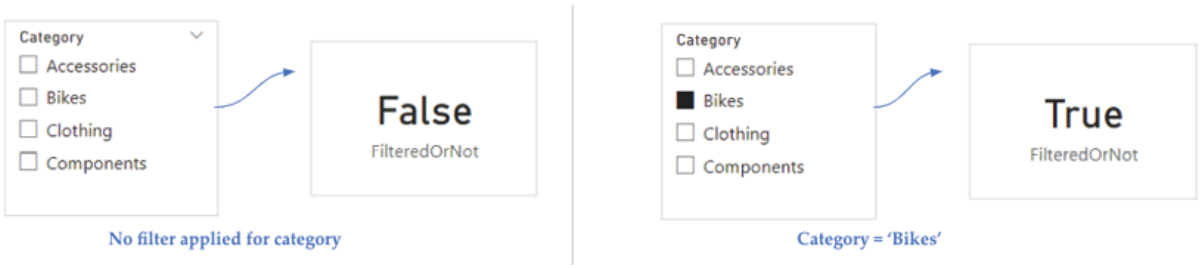



Figure ISFILTERED returning True when the specified column is filtered

ISNUMBER: Checks whether the specified value is a number or not, and returns True or False

Syntax: ISNUMBER()

Parameter: The value to be tested.

For example, the following expression, as used in measure returns True for 123 being a number:

NumberOrNot = ISNUMBER(123)

ISTEXT: Checks whether the specified value is text or not, and returns True or False

Syntax: ISTEXT()

Parameter: The value to be tested.

For example, the following expression, as used in measure returns True for ABC is a text:

TextOrNot = ISTEXT("ABC")

USERNAME: Returns the domain name and username from the credentials provided to the system at the connection time

Syntax: USERNAME()

Parameter: No argument required.

In Power BI Desktop, USERNAME returns the user information in the format of

USERPRINCIPALNAME: Returns the user principal name

Syntax: USERPRINCIPALNAME()

Parameter: No argument required.

In Power BI Desktop, USERPRINCIPALNAME returns the user information in the format of User@domain.com

USERNAME and USERPRINCIPALNAME functions are mostly used while configuring row-level security or RLS in Power BI.

## Logical functions

Logical functions act upon an expression and return information about the values or sets in the expression. A few commonly used logical functions are:

AND: Returns True when both arguments are True, otherwise returns False

Syntax: AND(,)

Parameters: The logical values to be tested.

The following expression as used in measure returns True as both the arguments > 1 and 3 > are True in nature:

UseOfAND = AND(2>1,3>2)

The AND function in DAX accepts only two arguments. If needed for multiple arguments, the AND function can be nested, or the AND operator can be used.

OR: Returns True when any one of the arguments is True, returns False when both arguments are False

Syntax: OR(,)

Parameters: The logical values to be tested.

The following expression as used in measure returns True as the first argument > 1) is True in nature, while the second (3 < 2) is

UseOfOR = OR(2>1,3<2)

The OR function in DAX accepts only two arguments. If needed for multiple arguments, the OR function can be nested, or the OR operator ( || ) can be used.

NOT: Changes False to True, or True to False

Syntax: NOT()

Parameter: A value or expression that can be evaluated as True or False.

In the following expression, as used in measure the logical test 2 > 3 is False in nature. Hence the test NOT(2 > 3) becomes True, and the result Expression is True is returned:

UseOfNOT = IF(NOT(2>3), "Expression is True","Expression is False")

IF: Validates a condition and returns a value when TRUE, otherwise returns an alternate value.

Syntax: IF(, [, ])

Parameters:

Any expression that returns either TRUE or FALSE.

The value to be returned if a logical test is TRUE.

The value to be returned if a logical test is FALSE. This is optional and if omitted, BLANK is returned.

Refer to Chapter Data Modeling in Power BI for more details.

SWITCH: Evaluates an expression against a list of values and returns result expressions as specified

Syntax: SWITCH(, , [,, ]…[,])

Parameters:

DAX expression returning a single scalar value.

Constant value to be matched with the results of the expression.

The result expression in case of a match.

Optional, the alternate result.

In case, we have a column representing month numbers in a table, the following expression used is calculated column MonthName should return the corresponding month names (starting from 1 for January to 12 for December):

```
MonthName =

SWITCH (

  'Month'[MonthNumber],

  1, "January",



  2, "February",

  3, "March",

  4, "April",

  5, "May",

  6, "June",
```

7, "July",

8, "August",

9, "September",

10, "October",

11, "November",

12, "December",

"Not Valid")

Also, in case of any invalid month number, the alternate result Not Valid would be returned.

TRUE: Returns the logical value True

Syntax: TRUE()

Parameter: No argument required.

The function always returns the word True is also interpreted as logical value True.

FALSE: Returns the logical value False

Syntax: FALSE()

Parameter: No argument required.

The function always returns the word False is also interpreted as logical value False.

## Relationship management functions

Relationship functions help to manage and utilize relationships between tables. A few commonly used relationship functions are:

RELATED: Returns a related value from another table.

Syntax: RELATED()

Parameter: Column that needs to be referred.

Refer to Chapter Data Modeling in Power BI for more details.

USERELATIONSHIP: Specifies the relationship to be used in a calculation. Though Power BI supports only one active relationship, there can exist multiple inactive relationships between two tables. USERELATIONSHIP can enable the indicated inactive relationship for the duration of the calculation. The function itself does not return any value.

Syntax: USERELATIONSHIP(, )

Parameters:

The name of an existing column, that usually represents the many (*) side of the relationship to be used.

The name of an existing column, that usually represents the one (1) side of the relationship to be

In our Sales Report data model, there is an active relationship between fact_Sales and If we want to slice a measure (for example, sum of using ShipDate instead of the default an inactive relationship can be created between fact_Sales and Then, the following expression as used in measuring SalesByShipDate can be used to override the default active relationship with the newly created inactive one:

```
SalesByShipDate =

CALCULATE (

  SUM ( fact_Sales[SalesAmount] ),

  USERELATIONSHIP ( fact_Sales[ShipDate], dim_Calendar[Date]
)

)
```

## Text functions

The Text functions help to perform string operations in DAX. A few commonly used Text functions are:

CONCATENATE: The CONCATENATE function joins two text strings into one text string

Syntax: CONCATENATE(, )

Parameters:

The first text string (the string can include both texts and numbers).

The second text string (the string can include both texts and numbers).

In the dim_Customers table of the Sales we have AddressLine1 and The following expression as used in the calculated column combines the two parts of the address into a single complete address, separated by a space:

FullAddress =

CONCATENATE ( dim_Customers[AddressLine1] & " ", dim_Customers[AddressLine2] )

FIXED: Rounds a number to the specified number of decimals and returns the result as text.

Syntax: FIXED(, , )

Parameters:

The number is to be rounded and converted to text.

Optional, the number of decimal places to be shown (2 decimals by default).

Optional, if 1, commas are not displayed in the returned text while for 0, commas are displayed.

The following expression returns 0.7 as a text, as used in measure

UseOfFIXED = FIXED(2/3,1)

This is because the result of 2 divided by 3 is 0.67; as we are showing a single decimal place in the measure, the value is rounded up to 0.7.

FORMAT: Converts a value to text according to the specified format

Syntax: FORMAT(, )

Parameters:

The value to be converted to text.

A string with the formatting template.

The following expression as used in measure UseOfFORMAT converts the date March 2023 to the text

UseOfFORMAT = FORMAT(DATE(2023,3,22),"MMM/YYYY")

LEN: Returns the number of characters in a text string

Syntax: LEN()

Parameter: The text for which the length is to be determined. Spaces count as

For example, the following expression as used in measure returns 8 for the text Power

TextLength = LEN("Power BI")

LEFT: Returns the specified number of characters from the start of a string

Syntax: LEFT(, )

Parameters:

The text string containing the characters to be extracted.

Optional, the number of characters to be extracted; the default value is 1.

For example, the following expression as used in measure ExtractLeft returns Self (4 characters from the left):

ExtractLeft = left("Self Service Analytics",4)

RIGHT: Returns the specified number of characters from the end of a string

Syntax: RIGHT(, )

Parameters:

The text string containing the characters to be extracted.

Optional, the number of characters to be extracted; the default value is 1.

For example, the following expression as used in measure ExtractRight returns Analytics (9 characters from the end):

ExtractRight = right("Self Service Analytics",9)

MID: Returns a string of characters from the middle of a text string, for the specified starting position and length

Syntax: MID(, , )

Parameters:

The text string containing the characters to be extracted.

The position of the first character to be extracted; the position starts from 1.

The number of characters to be extracted.

For example, the following expression as used in measure ExtractMid returns Service (extracting 7 characters starting from position

ExtractMid = Mid("Self Service Analytics",6,7)

SUBSTITUTE: Replaces existing text with new text in a text string

Syntax: SUBSTITUTE(, , , )

Parameters:

The text string containing the characters to be substituted.

The existing text is to be replaced.

The new text will replace the existing text.

Optional, the occurrence of existing text to be replaced; if omitted, by default every instance of the existing text is replaced.

For example, the following expression as used in the measure returns Self Service Analytics with Power

SubstituteText =

SUBSTITUTE ("Self Service Analytics", "Analytics", "Analytics with Power BI")

## Table specific functions

Table-specific functions help to manipulate existing tables or return new tables. A few commonly used table-specific functions are:

ADDCOLUMNS: Adds calculated columns to the specified table or table expression

Syntax: ADDCOLUMNS(

, , [„ ]..)
Parameters:

Table where the column is to be added.

Name of the new column, enclosed in double quotes.

DAX expression evaluated for each row of the table.

For example, the following expression returns a calculated table based on the original dim_Customers table with an added column Income following the logic as defined in the expression:

CustomerTable =

```
ADDCOLUMNS (
  dim_Customers,
    "Income Group",
      IF (
          dim_Customers[YearlyIncome] >= 100000,
          "High",
          IF (dim_Customers[YearlyIncome] >= 50000, "Medium",
"Low")
      )
)
```

DISTINCT: Returns a one-column table containing distinct values from the column which is passed as a parameter to this function.

Syntax: DISTINCT()

Parameter: The column from which unique values are to be returned.

Refer to Chapter Data Modeling in Power BI for more details.

SUMMARIZE: Returns a summary table for the specified expression over a set of groups

Syntax: SUMMARIZE(

, [, , ]…)
Parameters:

Table to summarize data.

Optional, existing column(s) to create summary groups.

The name of the summarized column; enclosed in double quotes

DAX expression evaluated for each row.

In the fact_Sales table, we have the sales transactions where the SalesAmount represents the sales quantity for an individual transaction. The dim_Territory table stores geographical information like Country, and so on. The dim_Territory table filters fact_Sales on the SalesTerritoryKey field. The following expression groups the SalesAmount in fact_Sales by different countries of and creates the new calculated table GroupByCountry as shown in Figure

GroupByCountry =

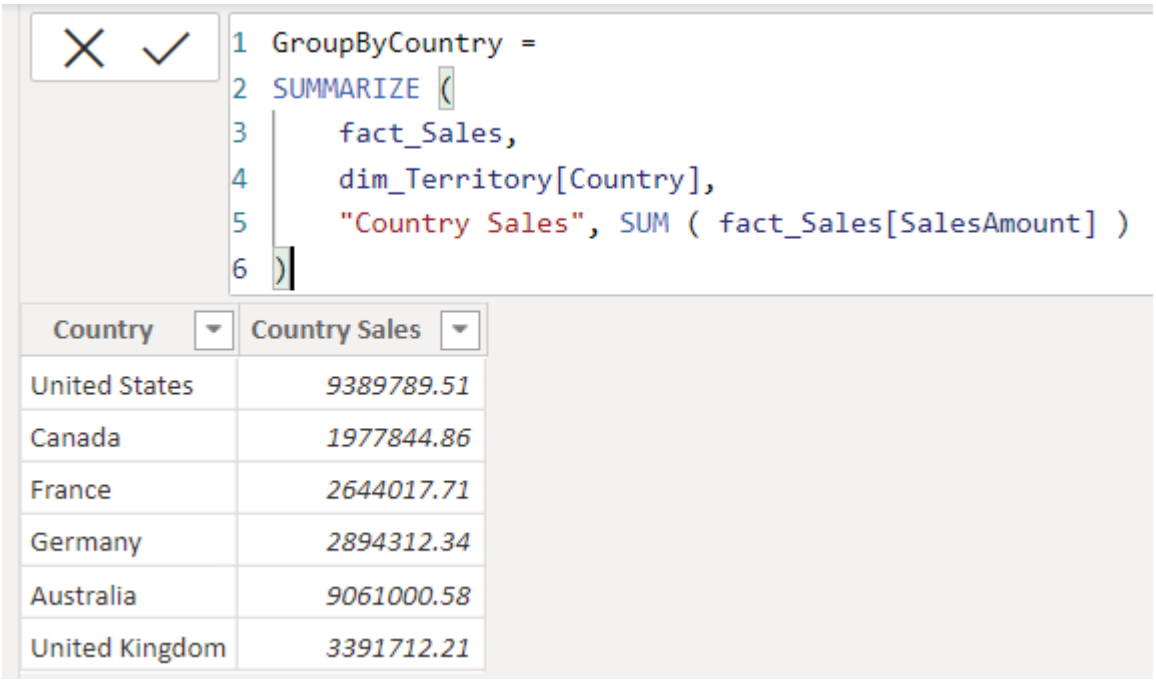
SUMMARIZE (

fact_Sales,

dim_Territory[Country],

"Country Sales", SUM ( fact_Sales[SalesAmount] )

)




| Country | Country Sales |
|---|---|
| United States | 9389789.51 |
| Canada | 1977844.86 |
| France | 2644017.71 |
| Germany | 2894312.34 |
| Australia | 9061000.58 |
| United Kingdom | 3391712.21 |

Figure Using SUMMARIZE to group by

UNION: Create a combined table from a pair of tables having the same number of columns.

Syntax: UNION(, )

Parameter: Any DAX expression that returns a table.

Refer to Chapter Data Modeling in Power BI for more details.

## Time intelligence functions

Time intelligence functions help to manipulate data using time periods including days, months, quarters and years. A few commonly used time intelligence functions are:

DATEADD: Returns a column of dates, shifted either forward or backward in time by the specified number of intervals from the dates in the current context

Syntax: DATEADD(, , )

Parameters:

Column containing dates.

An integer specifying the number of intervals.

The interval by which to shift the dates. Valid values are:

year

quarter

month

day

For example, the following expression as used in the calculated column LastYearOrderDate returns dates that are one year before the OrderDate in the current context:

LastYearOrderDate = DATEADD(fact_Sales[OrderDate],-1,YEAR)

FIRSTDATE: Returns the first date or the earliest date in the current context for the specified column of dates

Syntax: FIRSTDATE()

Parameter: Column containing dates

The expression as used in measure FirstNonBlankOrderDate returns the earliest order date in the context of the report:

FirstNonBlankOrderDate = FIRSTDATE(fact_Sales[OrderDate])

LASTDATE: Returns the last date or the latest date in the current context for the specified column of dates

Syntax: LASTDATE()

Parameter: Column containing dates.

The expression as used in measure LastNonBlankOrderDate returns the latest order date in the context of the report:

LastNonBlankOrderDate = LASTDATE(fact_Sales[OrderDate])

SAMEPERIODLASTYEAR: Returns a date column shifted one year back in time from the dates that have been specified, in the current context.

Syntax: SAMEPERIODLASTYEAR()

Parameter: A date column.

Refer to Chapter Visualizing Data in Power BI for more details.

TOTALYTD: Evaluates the year-to-date value of an expression in the current context.

Syntax: TOTALYTD(, [,] [,])

Parameters:

Expression returning a scalar value.

A date column.

Expression specifying a filter condition (optional).

A string defining the year-end date (optional, default is December 31).

Refer to Chapter Visualizing Data in Power BI for more details.

## Other functions

Miscellaneous functions to perform specific actions. A few commonly used functions in this category are:

BLANK: Returns a blank. Blanks and nulls or empty strings (“”) are not always equivalent, however, some operations may treat them as such

Syntax: BLANK()

Parameter: No argument required.

For example, while discussing the ISERROR function, we have created the Cost:Sales measure as below:

```
1 Cost:Sales =
2 VAR ratio =
3     SUM ( fact_Sales[TotalProductCost] ) / SUM ( fact_Sales[SalesAmount] )
4 RETURN
5     IF ( ISERROR ( ratio ), BLANK (), ratio )
```

Figure Cost:Sales measure

In the measure, if the variable ratio returns an error (for example, if the denominator is zero), then the measure would return a blank, otherwise the ratio itself is returned.

ERROR: Raises a user-specified error. When triggered, this function stops the execution raising an error with the description as provided in the error

text.

Syntax: ERROR()

Parameter: The error message as a text string.

For example, the following expression raises an error instead of a blank when the denominator is zero for the Cost:Sales_WithError measure, as shown in Figure

```
Cost:Sales_WithError =

VAR ratio =

  SUM ( fact_Sales[TotalProductCost] ) / SUM ( fact_Sales[SalesAmount] )

RETURN

  IF (

  SUM ( fact_Sales[SalesAmount] ) = 0,

  ERROR ( "No sales for this period" ),

  ratio

  )
```

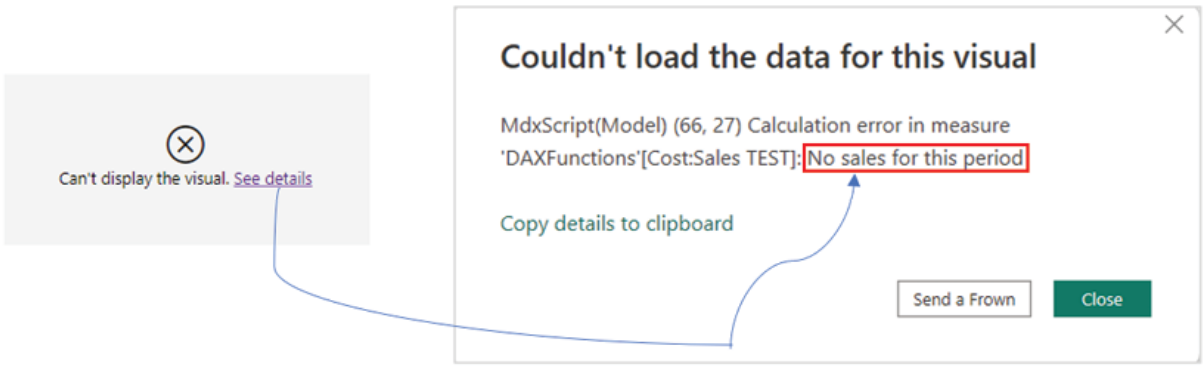



Figure Cost:Sales_WithError measure

## Conclusion

In this chapter, we went through different categories of DAX functions and explored a few commonly used ones from each category. As already mentioned, this is not an exclusive list of all DAX functions, but instead a collection of selected functions which have high usability besides being easy to understand, especially for business users. By the end of the chapter, readers should be able to build their custom logic using DAX as per the reporting requirements they have and eventually gain expertise over the time!

## Knowledge Check

Which of the following functions evaluate an expression in a row context?

SUMX

CALCULATE

FILTER

Which of the following functions can be used to round a decimal number and return it as text?

ROUND

FIXED

UP

To count the number of non-blank rows in a Boolean column, which function can be used?

COUNTNONBLANK

COUNTA

COUNT

All Knowledge Check answers are provided at the end of the book.

**Join our book’s Discord space**

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

# CHAPTER 8

# Use Case - Creating a Risk Report in Power BI

## Introduction

You have now reached the final chapter of the book! Now that we have a solid understanding of the subject, time to face some real-world business problems and see how Power BI can provide an efficient solution. In this chapter, we are going to create a solution for a problem that is extremely common across different industries. In any project, from a management perspective, it is required to identify, analyze and mitigate potential risks throughout the lifecycle of the project. Besides that, risks need to be categorized as well to identify which one needs the earliest attention for the project to remain on track. Usually, this problem is handled manually using Excel files which can be a tedious job to carry out regularly. Let us see how Power BI can come to the rescue to create an automated solution, in the form of a self-service risk report, which immediately can enable capabilities like live project risk monitoring without any or minimum manual interventions.

## Structure

In this chapter, we will discuss the following topics:

The reporting requirements

Creating the design

Performing transformations

Modeling data

Creating the report

Publishing reports and scheduling refreshes

Creating a cross-workspace design

Report personalization

Analyze in Excel

## Objectives

The objective of this chapter is focused towards enabling readers to apply the concepts learnt throughout the book with a business use case. Developing a business intelligence report demands creativity besides technical knowledge, and this chapter shall aim to demonstrate how an outside-the-box solution can be developed, even for an extremely common problem. Apart from that, readers would also be introduced to new concepts like cross-workspace design, personalizing reports for the end users to self-serve and so on. We would also explore how to integrate Power BI datasets with Excel. By the end of this final chapter, readers would, not only have more tools at their disposal but should also become more exploratory while designing a solution using Power BI.

## The reporting requirements

The project records information involving potential risks like failed deadlines, delayed delivery, out-of-the-scope requirements, and so on. Every risk is assigned to a unique identifier called Risk A risk record captures information like a description of the risk, record creation date, the status of the risk, the severity of the risk, the likelihood of the risk, any mitigation plan, and so on.

Severity and Likelihood are the parameters based on which risk is assessed. Severity implies the impact that the risk can pose to the project or business, while the Likelihood of risk means the probability of the risk to happen. In the project, Severity is categorized as Medium, or High while the categories for Likelihood are Very Medium or For example, in the case of a mission-critical database for business, the Severity of failure would be High while the Likelihood of failure should be Low or Very as usually mission-critical systems are configured to be fault tolerant.

Figure 8.1 shows a snapshot of the risk data that is being maintained for the project:

| Risk Id | Title | Date Modifie | Date Created | Description | Status | Category | Severity | Likelihood | Action | Comments |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Sample Risk:4 | 22-May-22 | 12-Nov-20 | Risk Description:4 | Closed | Scope | High | Medium | Action:4 | Comment:4 |
| 2 | Sample Risk:52 | 27-Oct-21 | 15-Nov-20 | Risk Description:52 | Closed | Scope | High | High | Action:52 | Comment:52 |
| 3 | Sample Risk:80 | 27-Oct-21 | 23-Nov-20 | Risk Description:80 | Closed | Scope | Medium | Low | Action:80 | Comment:80 |
| 4 | Sample Risk:67 | 22-May-22 | 23-Nov-20 | Risk Description:67 | Closed | Scope | Medium | Low | Action:67 | Comment:67 |
| 5 | Sample Risk:1 | 22-May-22 | 23-Nov-20 | Risk Description:1 | Closed | Scope | Low | Low | Action:1 | Comment:1 |
| 6 | Sample Risk:57 | 03-Feb-21 | 23-Nov-20 | Risk Description:57 | Closed | Scope | High | High | Action:57 | Comment:57 |
| 7 | Sample Risk:35 | 22-May-22 | 29-Nov-20 | Risk Description:35 | Closed | Technical | Medium | Medium | Action:35 | Comment:35 |
| 8 | Sample Risk:68 | 22-May-22 | 01-Dec-20 | Risk Description:68 | Closed | Scope | Medium | High | Action:68 | Comment:68 |
| 9 | Sample Risk:2 | 18-Jan-21 | 01-Dec-20 | Risk Description:2 | Closed | Scope | High | Medium | Action:2 | Comment:2 |
| 10 | Sample Risk:27 | 17-Jan-22 | 02-Dec-20 | Risk Description:27 | Closed | Scope | Medium | Low | Action:27 | Comment:27 |
| 11 | Sample Risk:37 | 22-May-22 | 06-Jan-21 | Risk Description:37 | Closed | Scope | Low | Very Low | Action:37 | Comment:37 |
| 12 | Sample Risk:63 | 03-Aug-22 | 20-Jan-21 | Risk Description:63 | Closed | Scope | High | Medium | Action:63 | Comment:63 |
| 13 | Sample Risk:19 | 03-Aug-22 | 20-Jan-21 | Risk Description:19 | Closed | Quality | High | Very Low | Action:19 | Comment:19 |
| 14 | Sample Risk:14 | 22-May-22 | 20-Jan-21 | Risk Description:14 | Closed | Scope | Low | Medium | Action:14 | Comment:14 |
| 15 | Sample Risk:97 | 27-Oct-21 | 09-Feb-21 | Risk Description:97 | Closed | Technical | High | High | Action:97 | Comment:97 |
| 16 | Sample Risk:30 | 03-Aug-22 | 14-Feb-21 | Risk Description:30 | Closed | Resource | High | Medium | Action:30 | Comment:30 |
| 17 | Sample Risk:36 | 22-May-22 | 14-Feb-21 | Risk Description:36 | Closed | Technical | High | Medium | Action:36 | Comment:36 |
| 18 | Sample Risk:29 | 22-May-22 | 14-Feb-21 | Risk Description:29 | Closed | Scope | High | Low | Action:29 | Comment:29 |
| 19 | Sample Risk:49 | 22-May-22 | 14-Feb-21 | Risk Description:49 | Closed | Technical | High | High | Action:49 | Comment:49 |
| 20 | Sample Risk:69 | 22-May-22 | 14-Feb-21 | Risk Description:69 | Closed | Scope | High | High | Action:69 | Comment:69 |
| 21 | Sample Risk:46 | 22-May-22 | 18-Feb-21 | Risk Description:46 | Closed | Technical | Low | Medium | Action:46 | Comment:46 |
| 22 | Sample Risk:60 | 27-Oct-21 | 28-Feb-21 | Risk Description:60 | Closed | Technical | High | Medium | Action:60 | Comment:60 |
| 23 | Sample Risk:77 | 22-May-22 | 05-Oct-21 | Risk Description:77 | Closed | Scope | High | Medium | Action:77 | Comment:77 |
| 24 | Sample Risk:55 | 01-Aug-22 | 05-Oct-21 | Risk Description:55 | Closed | Scope | Medium | Low | Action:55 | Comment:55 |

Figure Mock-up risk data

The data is used to assess the status of every risk record as per the mapping shown in Figure

| Severity | Likelihood | Risk Status |
|---|---|---|
| Low | Very Low | Low Risk |
| Low | Low | Low Risk |
| Low | Medium | Medium Risk |
| Low | High | Medium Risk |
| Medium | Very Low | Low Risk |
| Medium | Low | Medium Risk |
| Medium | Medium | Medium Risk |
| Medium | High | High Risk |
| High | Very Low | Medium Risk |
| High | Low | Medium Risk |
| High | Medium | High Risk |
| High | High | High Risk |

Figure Mapping for assessing the status of risks in a project

Depending on the risk status, the priority of the action or mitigation plan is determined. Hence the primary requirement is to have an automated way of computing and visualizing the risk status for every risk at a glance. The report users should also be able to view additional information associated with a specific risk. Apart from that, the report should be interactive enough and also users would ideally be able to self-serve in terms of creating their own views using the underlying data, to cater to individual visualization requirements maintaining a single point of truth.

Let us now explore how a solution can be created using Power BI, delivering all the requirement specifications that have been presented.

## Creating the design

The source file of the risk data is a CSV file, that gets generated from a custom application daily and saved in a SharePoint online folder. Power BI will consume the data from SharePoint. After performing the required transformations, the data will be modeled, and the report will be created in Power BI Desktop. Once the report is published to Power BI Service, having SharePoint online as the data source, the dataset should directly refresh using the data source credentials without the need for any data gateway.

As we need to create a self-service model, we will publish a copy of the live connected report to a separate workspace. The workspace having the dataset would be restricted to only the report authors or developers who will be maintaining the dataset. The workspace containing the live connected report can be accessed by the Super who would be able to update or alter the report if required. Finally, the live connected report will be distributed to the end user group via an app, who would be able to personalize the report as per their own need as well as create custom views.

Figure 8.3 illustrates the architectural design that we will implement for this requirement:

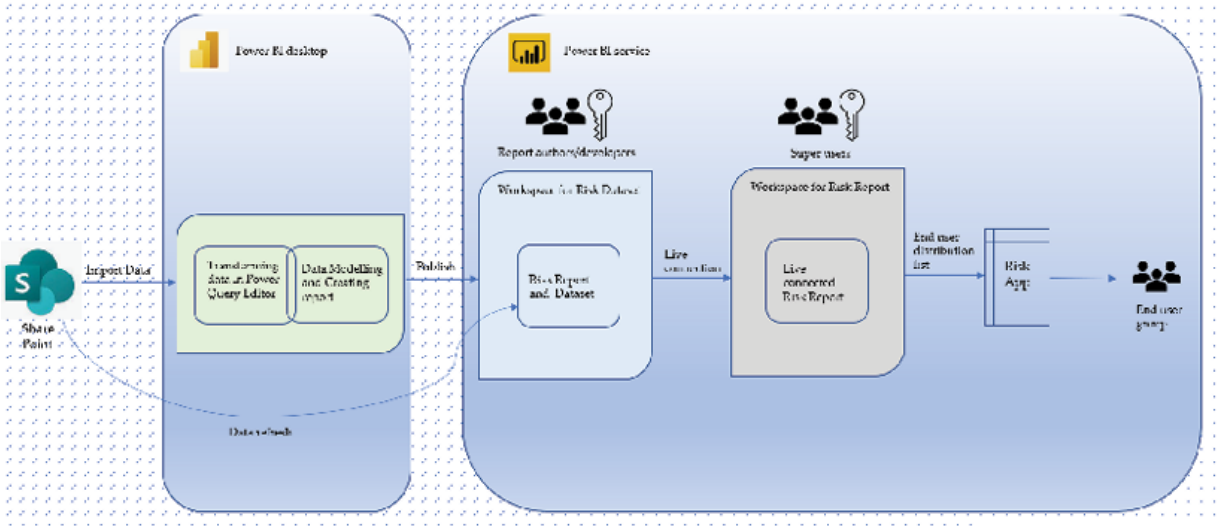



Figure Architectural design diagram

The design should support all the requirement specifications that have been provided. However, we need to also think about the best possible way to design the report itself, so that risk status across the spectrum can be visualized intuitively. Looking at the mappings as shown in Figure every record in the dataset would fall into either of the three possible risk statuses, depending on its severity and likelihood categories. There are 3 severity categories and 4 likelihood categories. Hence, if we create a 3 by 4 matrix, that should accommodate every risk that we have in the data. If we plot it, the chart should look somewhat like Figure

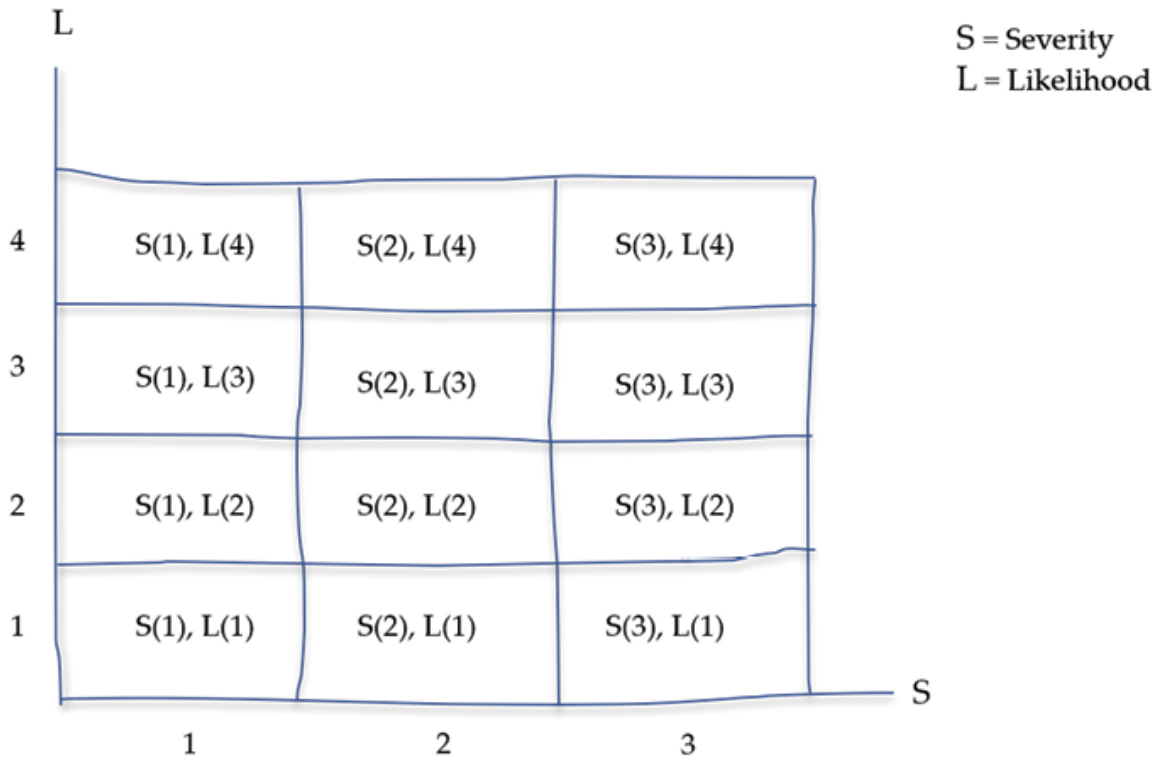



Figure Visualizing a risk matrix

Now that we have a design in place alongside a plan regarding how to proceed with developing the solution, let us try to implement it hands-on and see the outcome!

## Performing transformations

In Power BI Desktop, once the risk data is loaded into Query Editor connecting SharePoint, let us first try to apply all the transformations that we foresee would be required to shape the data. If any further transformation is required at any later stage, we can always come back to perform those in the Query Editor.

Just looking at the Severity and Likelihood columns, it seems that there is a leading space for all the values which might cause issues later while computing the risk matrix. After trimming those two text fields, the cleaned data can be seen in Figure

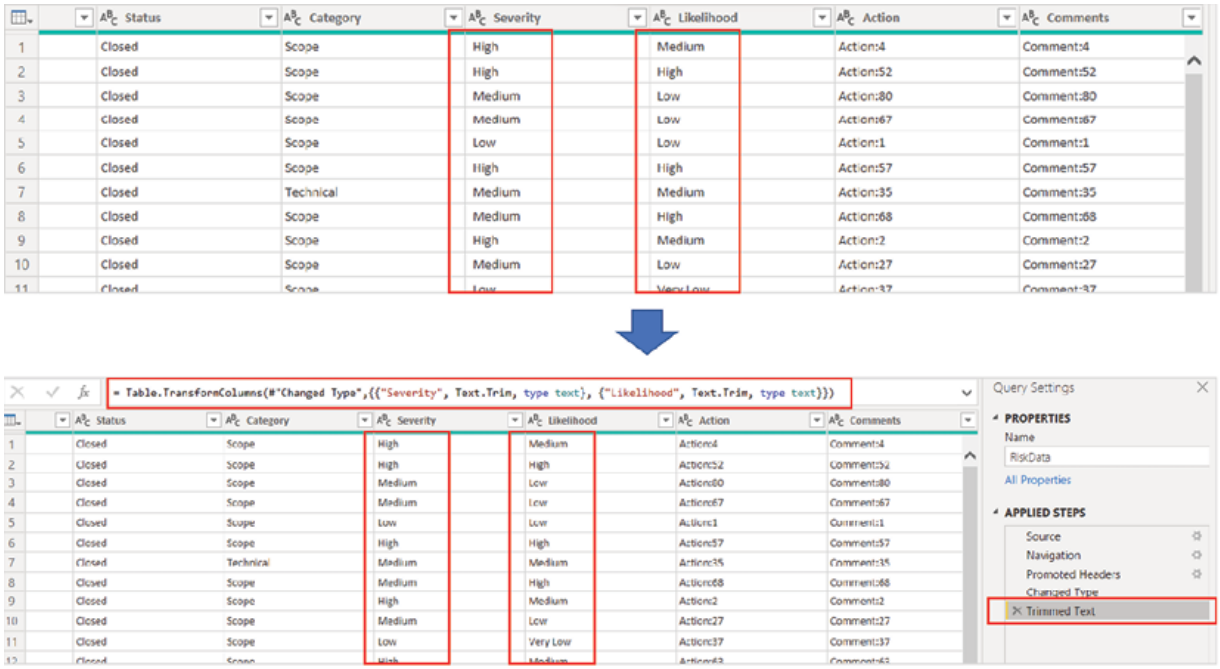



Figure Cleaning data in Query Editor

Apart from this, it would be good if we include a data refresh timestamp on the report itself, which should indicate when the Power BI dataset is refreshed from the source. To do this, in the Query Editor, we can make use of a Blank Query and search for the function which returns the present date and time in Coordinated Universal Time Then we can convert the query into a table, and finally rename the table as well as the only field as TimeStamp and update the data type of the field as The process is illustrated in Figure

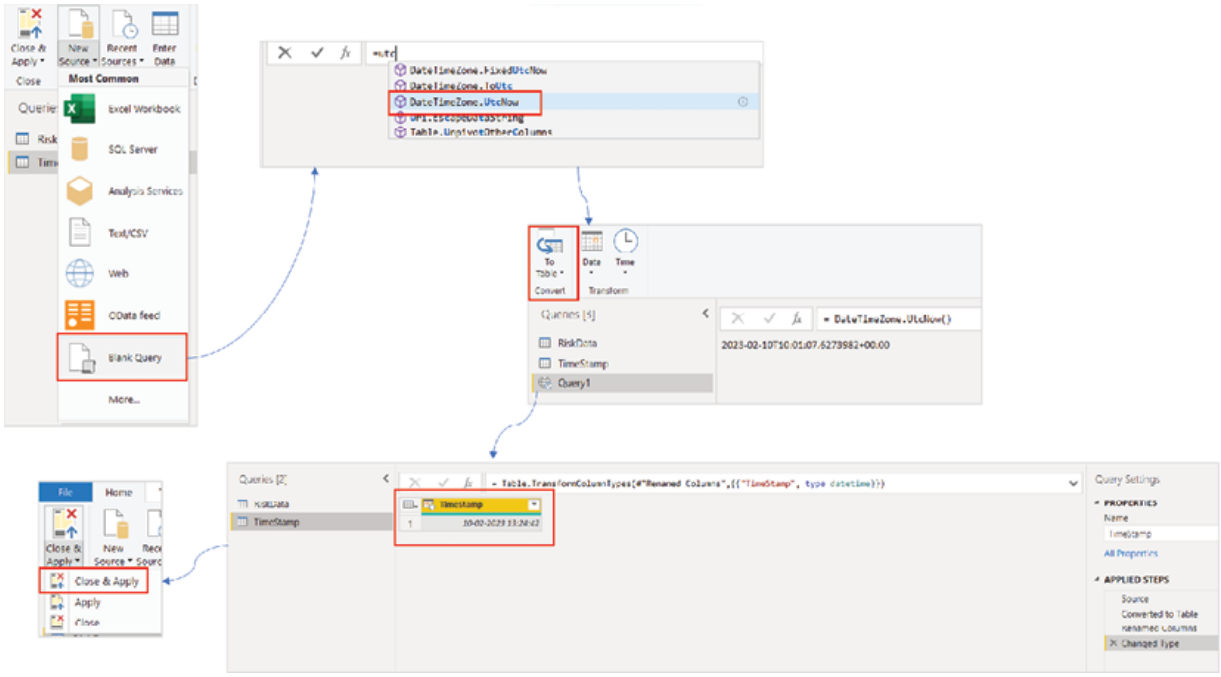



Figure 8.6: Creating a data refresh timestamp table

As we have now performed all the required transformations in Query Editor, let us Close & Apply the changes to Power BI Desktop, and proceed with modeling the data.

Modeling data

In the data model, we have two unrelated tables now. The RiskData table has all the risk-related records imported from SharePoint, while the

TimeStamp table holds a single value which is the dataset refresh timestamp in UTC and gets updated during each data refresh. There is no need to create any relationship between them and hence no additional work is required from a star schema creation perspective.

However, the existing data model would not support us to create the risk matrix (refer to Figure that we intend to create and hence, we need to enhance the data model. As the risk co-ordinates need to be calculated for each record, that means the computation has to work on a per row basis and for that, we need a calculated column. Let us create a calculated column following the mappings as mentioned in Figure Here is the DAX code snippet that we are going to use:

```
RiskMatrix =

SWITCH (

    TRUE (),

    AND ( RiskData[Severity] = "Low", RiskData[Likelihood] = "Very Low" ), "1,1",

    AND ( RiskData[Severity] = "Medium", RiskData[Likelihood] = "Very Low" ), "2,1",

    AND ( RiskData[Severity] = "High", RiskData[Likelihood] = "Very Low" ), "3,1",

    AND ( RiskData[Severity] = "Low", RiskData[Likelihood] = "Low" ), "1,2",
```

```
    AND ( RiskData[Severity] = "Medium", RiskData[Likelihood] = "Low"
), "2,2",

    AND ( RiskData[Severity] = "High", RiskData[Likelihood] = "Low" ),
"3,2",

    AND ( RiskData[Severity] = "Low", RiskData[Likelihood] = "Medium"
), "1,3",



    AND ( RiskData[Severity] = "Medium", RiskData[Likelihood] =
"Medium" ), "2,3",

    AND ( RiskData[Severity] = "High", RiskData[Likelihood] = "Medium"
), "3,3",

    AND ( RiskData[Severity] = "Low", RiskData[Likelihood] = "High" ),
"1,4",

    AND ( RiskData[Severity] = "Medium", RiskData[Likelihood] = "High"
), "2,4",

    AND ( RiskData[Severity] = "High", RiskData[Likelihood] = "High" ),
"3,4"

)
```

Once committed, this should create the RiskMatrix calculated column with the required risk co-ordinates, as shown in Figure

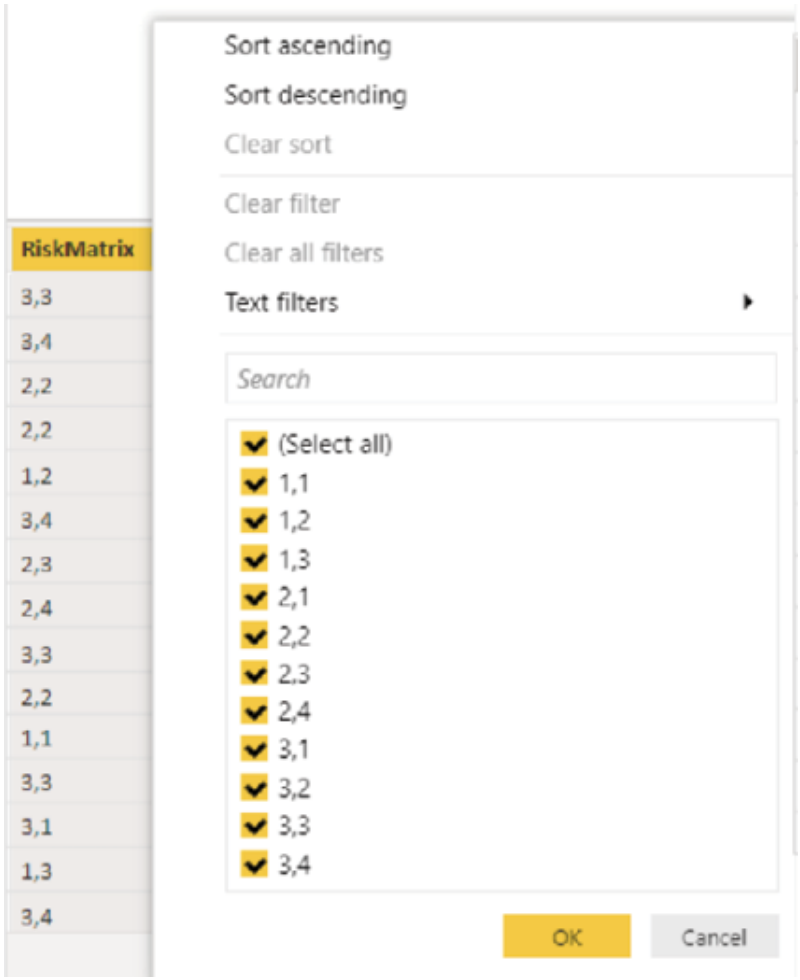



Figure Calculated column with risk co-ordinates

Now, this column can be used to filter the Risk For instance, filtering by should display all the risks for which the Severity is Low and Likelihood is Very The only thing left is to create another column for the status of the risks (again, refer to Figure which we can use for slicing the report. To create the calculated column the following DAX code snippet can be used:

```
RiskStatus =
SWITCH (
    TRUE (),
    RiskData[RiskMatrix] = "1,1"
```

```
    || RiskData[RiskMatrix] = "2,1"
    || RiskData[RiskMatrix] = "1,2", "Low Risk",
  RiskData[RiskMatrix] = "2,4"
    || RiskData[RiskMatrix] = "3,3"
    || RiskData[RiskMatrix] = "3,4", "High Risk",
  "Medium Risk"
)
```

Once created, the column should contain all the relevant risk statuses, as shown in Figure

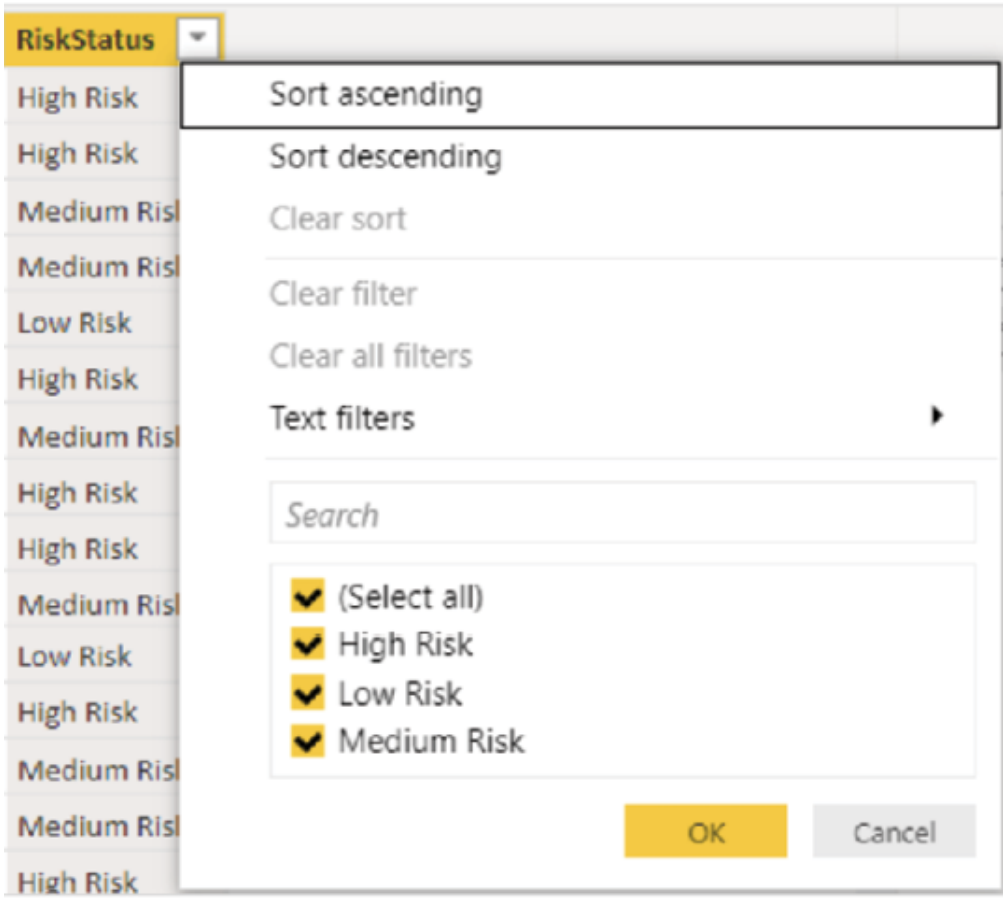



Figure Calculated column with risk status

The model should now be ready for creating the risk report, which we will explore in the next section.

## Creating the report

This report is not going to be heavy in terms of the number of visuals, and the risk matrix would be the key one here. However, there is no default visual that we can use straight away for the purpose. Instead, what we can do is create an image of the risk matrix (refer to Figure which should have placeholders for each possible Severity-Likelihood combination. As we have 3 Severity categories and 4 Likelihood categories, the matrix should have 12 cells to accommodate all possible risk statuses Medium Risk, or High

Once we do that, the idea is to filter each cell of the matrix by the corresponding risk co-ordinates, starting from (1,1) to Also, the image should have color codes for each risk status, Reddish for High Yellowish for Medium Risk and Greenish for Low Figure 8.9 shows the PowerPoint image of the risk matrix that we will use in the report:

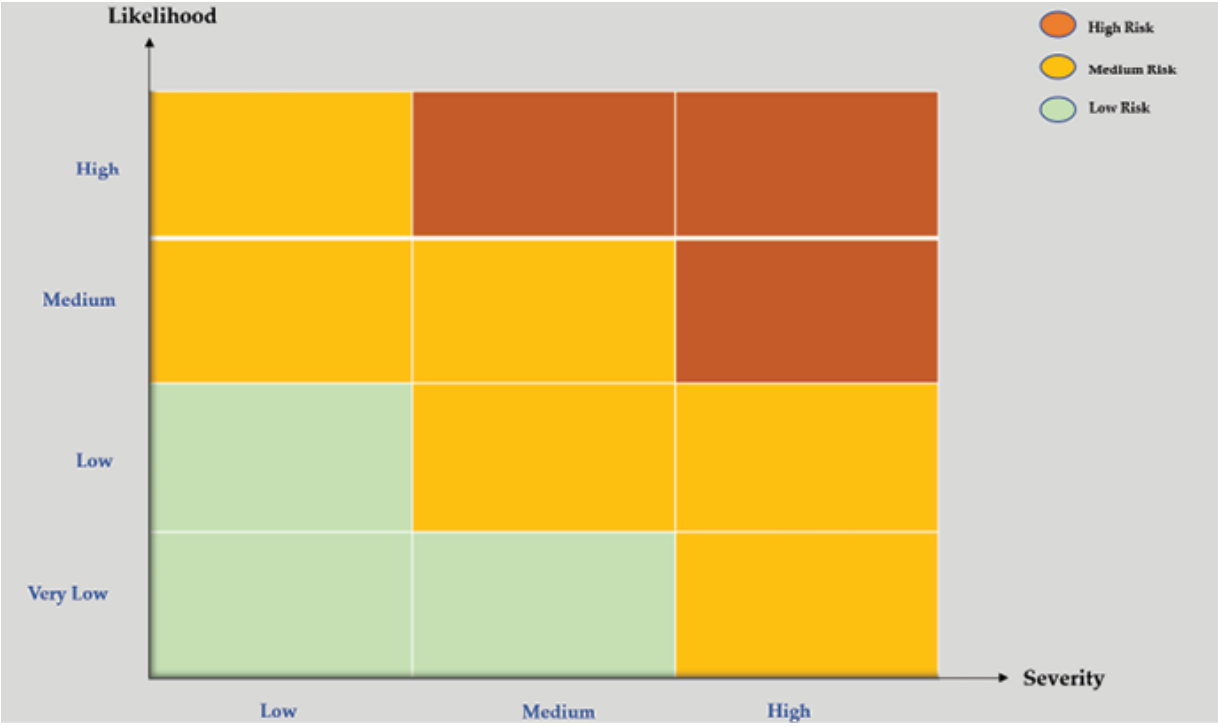



Figure Risk matrix image created in PowerPoint

As seen in the preceding figure, the X axis is representing Severity while the Y axis is representing Also, the cells are color coded according to the risk status they are representing based on the risk co-ordinates.

In the canvas of Power BI Desktop, let us first apply the same theme that we used in our Sales Report earlier, and name the report as Risk After that, the PowerPoint image of the risk matrix can be inserted in the canvas following Insert | and then resized and placed towards the left-hand side of the page as shown in Figure

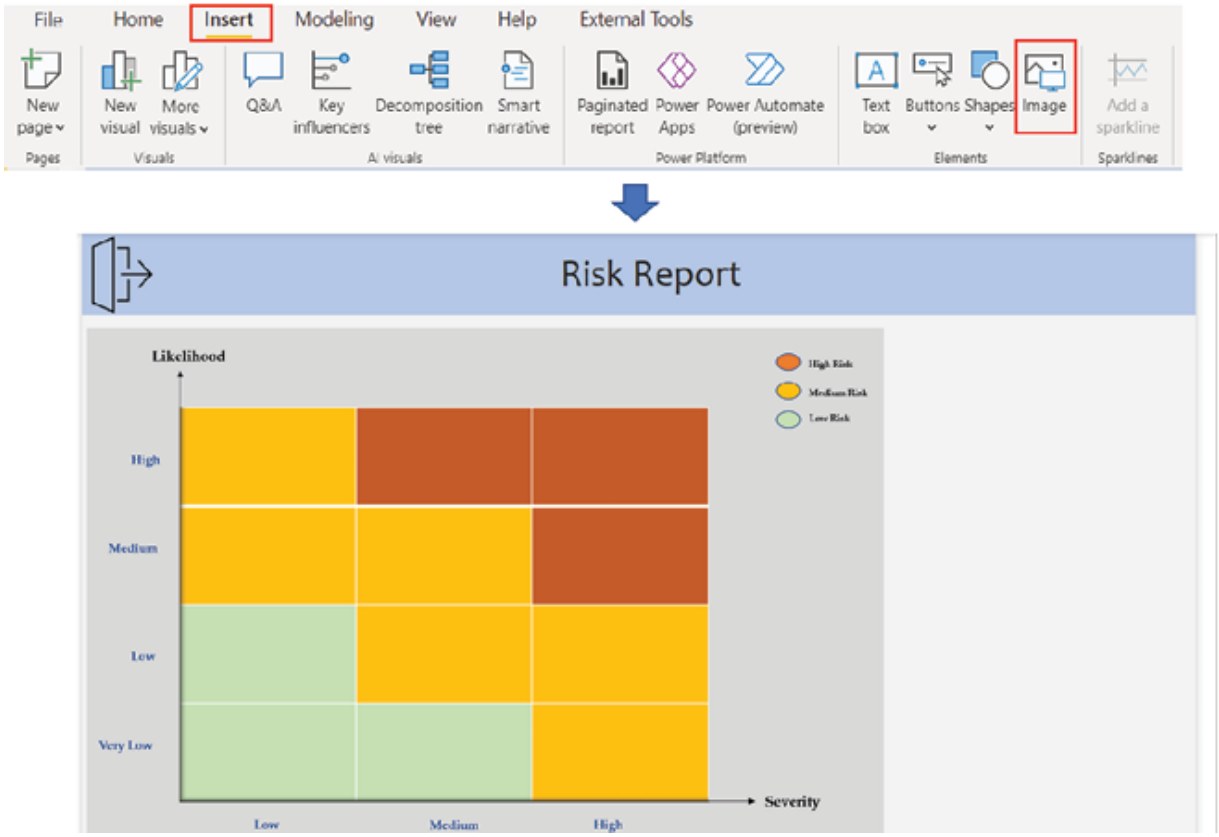



Figure Inserting an image into the risk report

Now, we need to show the Risk which is unique for any risk, in the respective cell that it belongs to. We will use a slicer for displaying the Risk to make it interactive. However, instead of using the standard slicer, we will use a custom visual from named Chiclet After adding the Chiclet Slicer to the report and pinning it to the Visualizations pane, it should be ready to be used as shown in Figure

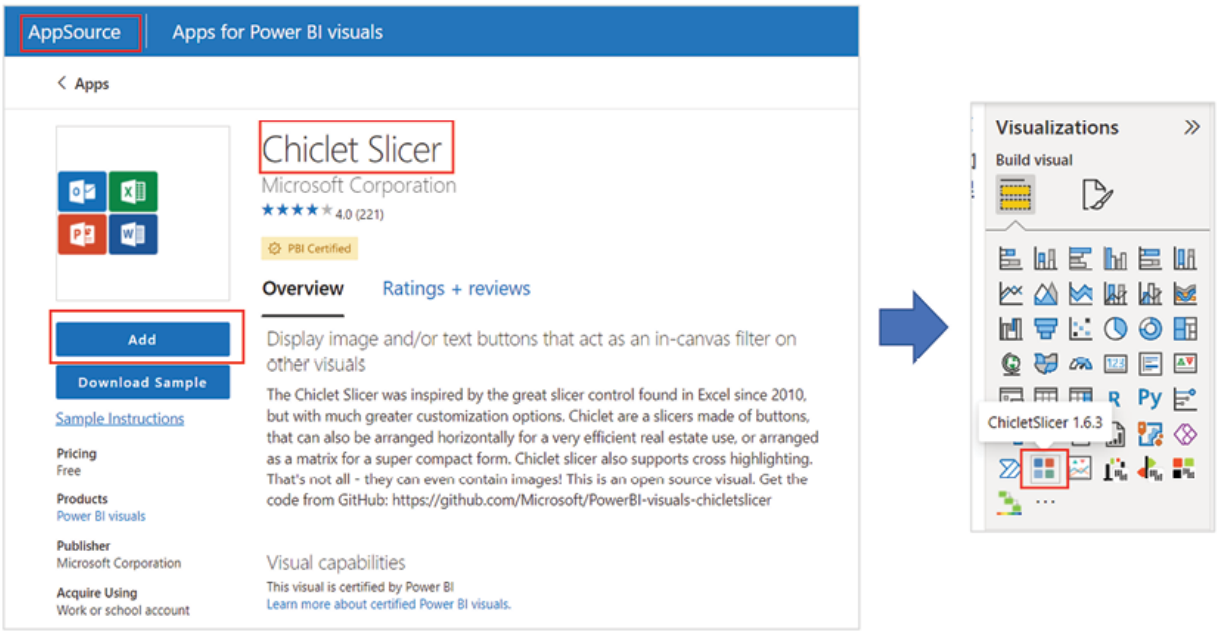



Figure Adding Chiclet slicer to the report

Once added, let us select the Chiclet Slicer and add Risk to the Category field of the visual container. At this point, all the available Risk would be visible in the slicer. The only formatting that would be required to perform is to disable the background by following Format visual | General | Effects | Background as well as the header following Format visual | Visual | Now, the slicer would have a transparent background and can be resized and placed on a cell of the image of the risk matrix, as shown in Figure

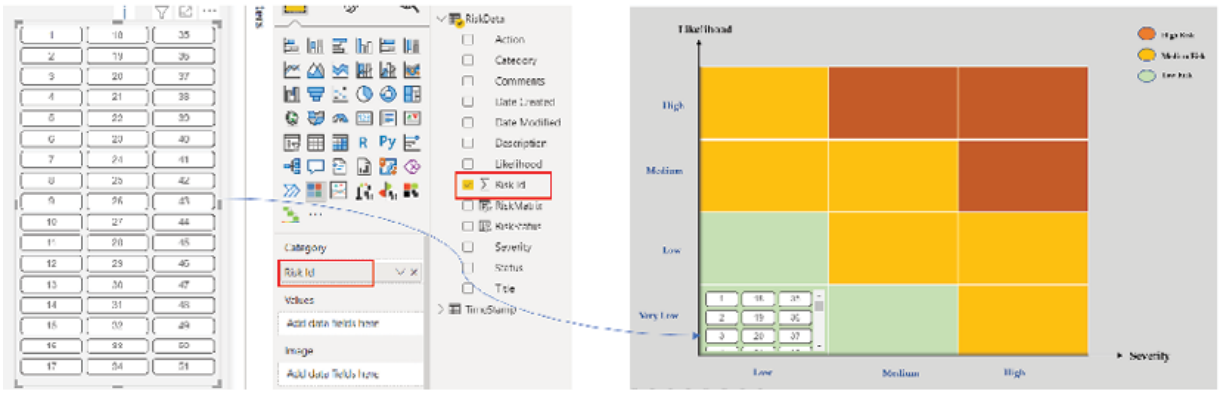



Figure Formatting Chiclet slicer and placing to the matrix cell

As seen in Figure we have placed the slicer on the cell having co-ordinate which represents Low for Severity and Very Low for Likelihood (refer Figure and hence would have the status of Low Risk (greenish Now what is left is to apply the risk matrix co-ordinate as a filter on the slicer using the Filters pane so that it only displays the Risk which have a Low Severity and Very Low Figure 8.13 illustrates the process:

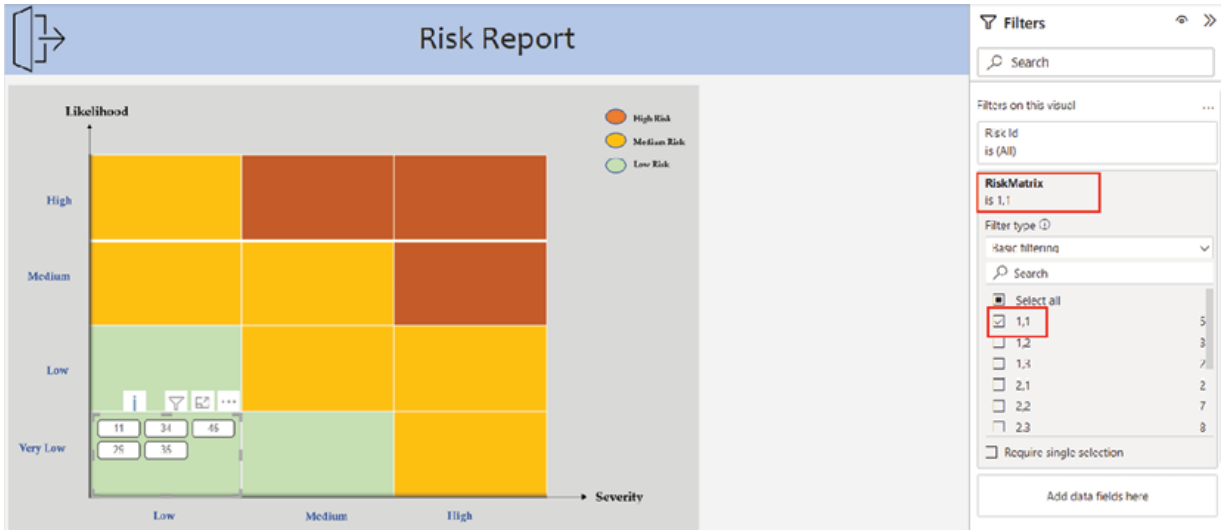



Figure Filtering Chiclet slicer with corresponding risk co-ordinate

As can be seen, only five Risk 29, 34, 35, qualify to be in that cell. If we validate the result on the Data view of Power BI Desktop by filtering the RiskMatrix column by we should get the same five records having the Risk as mentioned above. Next is the tedious part of repeating the same process for all the rest of the cells. The easy way of doing it should be to just copy the slicer of co-ordinate paste it to another cell and finally change the risk matrix co-ordinate on the Filters pane. Once completed, the risk matrix should look like Figure


Figure The final risk matrix

As per the requirement specifications, this visual does provide an automated way of visualizing different risk statuses at a glance.

To provide additional information about the risks, let us now create a Table and place it just to the right side of the report page, having information like risk title, description, and comment. By default, the table should display information about all the risks that we have in our dataset, however, if we select any specific Risk Id on the risk matrix, the table would be filtered and display only relevant data. To format the table, Alternating rows is selected as style, following Format visual | Visual | Style The background color for the column headers can be selected following Format visual | Visual | Column headers | Background

After that, to provide the report users with some more interactivity, let us create a regular slicer with the calculated column RiskStatus so that users can filter the report for Low or Medium or High risks. Let us keep the orientation of the slicer following Format visual | Visual | Slicer settings |

Finally, the data refresh timestamp can be displayed on a Card from the TimeStamp table, and placed on the right side of the top ribbon. The background of the visual should be disabled so that it becomes transparent, and the field is renamed as Last The value of the timestamp is updated whenever the dataset is refreshed and hence represents the last refreshed time for the report. Figure 8.15 shows how the final risk report looks like:


Figure The completed risk report

The report is completely interactive, and can be cross-filtered by both the Risk of the risk matrix and the risk as shown in Figure

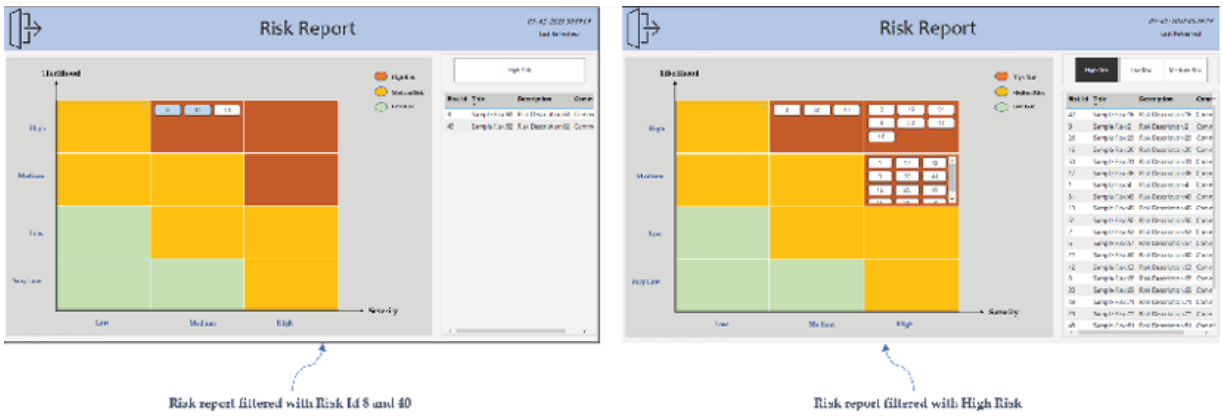



Figure Interacting with and cross-filtering the risk report

As we have created our risk report now, let us further explore how to create a self-service model for the report once it is published to Power BI Service.

Publishing reports and scheduling refreshes

The risk report is created from a SharePoint online folder, using the Web connector which we have discussed in detail in Chapter Data Discovery Using Power Once published to the workspace Self Service Analytics with Power the report would not require any gateway for data refresh. However, before initiating a refresh, we need to configure the credentials for data source authentication.

From the dataset Settings of the report, the option for Edit credentials can be found under the Data source credentials section. Selecting Edit credentials prompts the configuration window where properties like Authentication method and Privacy level settings can be maintained. Figure 8.17 illustrates the configurations:

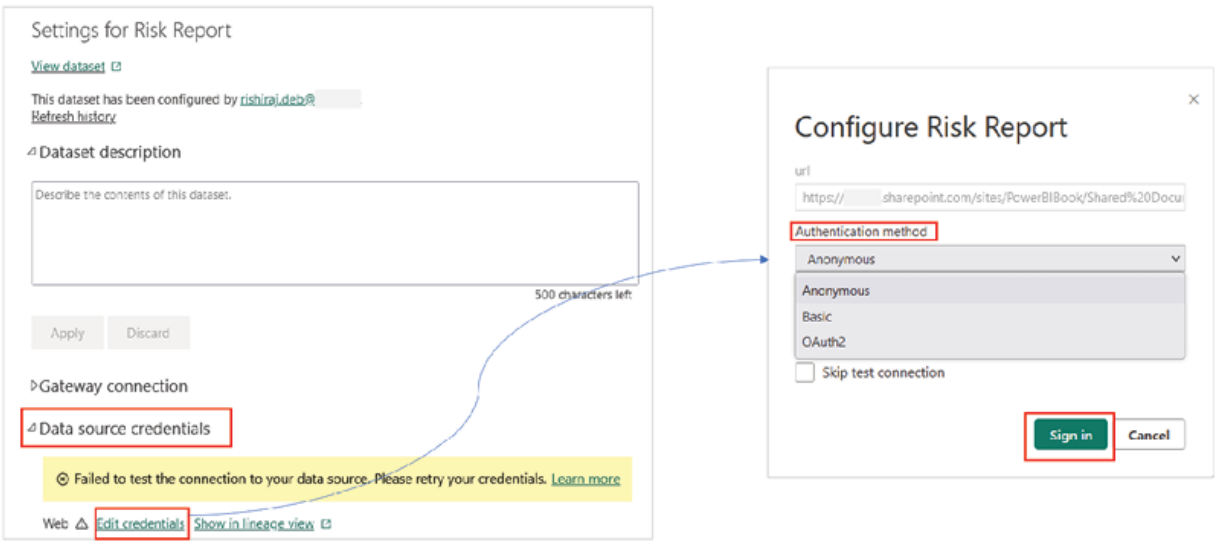



Figure 8.17: Setting up credentials in Power BI Service

This is a one-time activity and once successfully signed in, the report should subsequently refresh using the credentials maintained.

Once the refresh mechanism is established, the next required action is to assign a refresh schedule for the report to keep it up-to-date with the source, eliminating the need to manually refresh the report every day. Again, from dataset the Scheduled refresh toggle can be enabled, and then after selecting the Refresh frequency and Time the refresh times can be added. In case multiple refreshes are required daily, additional time slots can be added using the Add another time option. The refresh schedule for the risk report is shown in Figure

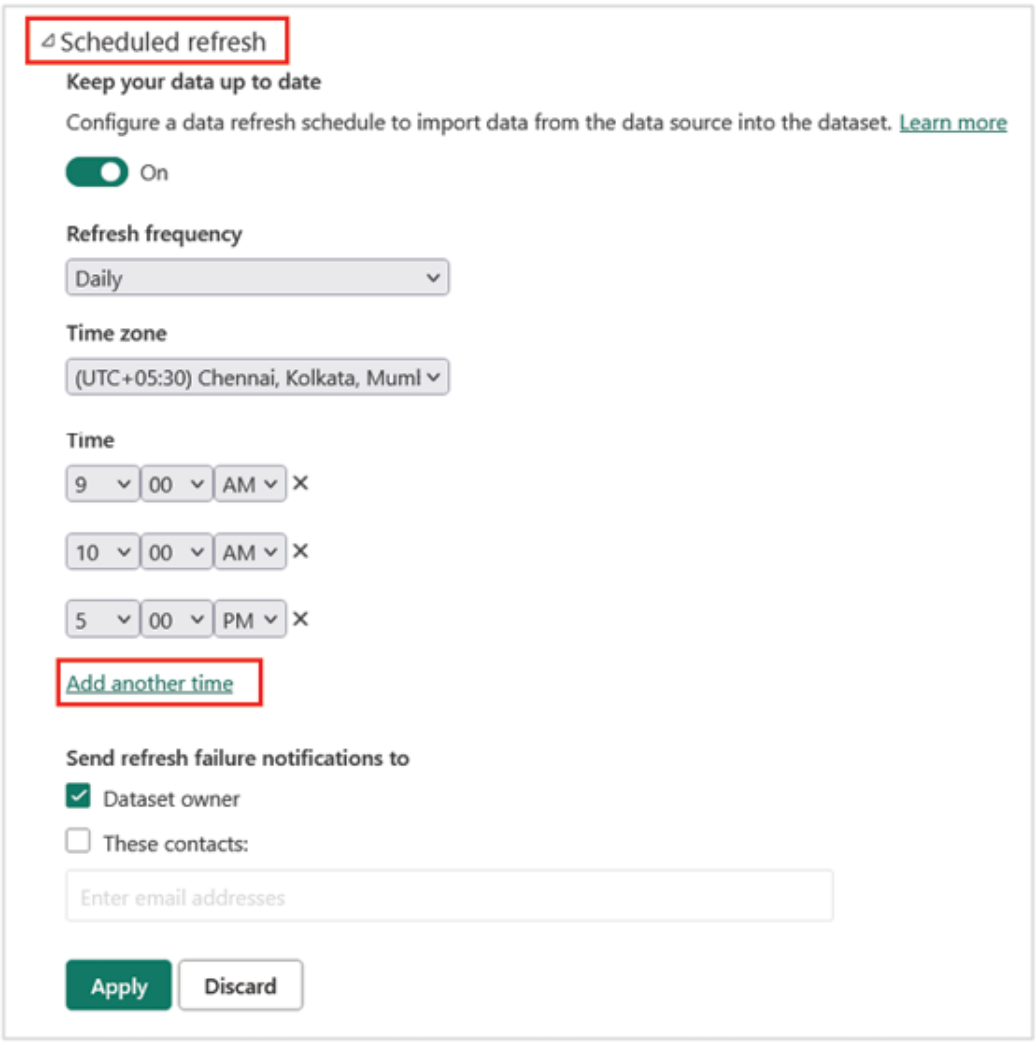


Figure Scheduled refresh configurations

In the next section, let us explore how to create a cross-workspace design for our report.

## Creating a cross-workspace design

Now that we have published the risk report and scheduled refreshes for it, we can separate out the report from the model. Though once the report is published to the service, the report and the dataset get stored in the workspace separately by default, however, both of the entities reside in the same workspace. Here separating out means publishing the live connected report to a separate workspace, which is an optional approach that can be considered based on the design we are trying to implement.

If we have another look at the architectural design diagram (refer to Figure the Super Users should have access to the report to perform any report-level alterations if required, however, the dataset or model access should be restricted only to the report authors or developers. To implement that, isolating the report workspace and the dataset workspace should be a good idea.

The easiest way to do it would be to download the live connected report (a feature introduced of late) and publish it back to a new workspace. To download the live connected report, after opening the published report on Power BI Service, the Download this file option can be selected from That should prompt a dialog box asking whether we want to download the report with data or a live-connected version of it. Choosing A copy of your report with a live connection to data online (.pbix) and selecting Download should download the live connected report on the local machine.

Once downloaded, when we open the report in Power BI Desktop, the bottom right of the report page should have a message indicating that it is

indeed a live connected report with the dataset we published in the workspace Self Service Analytics with Power

Figure 8.19 illustrates the process of downloading the live connected report from Power BI Service:

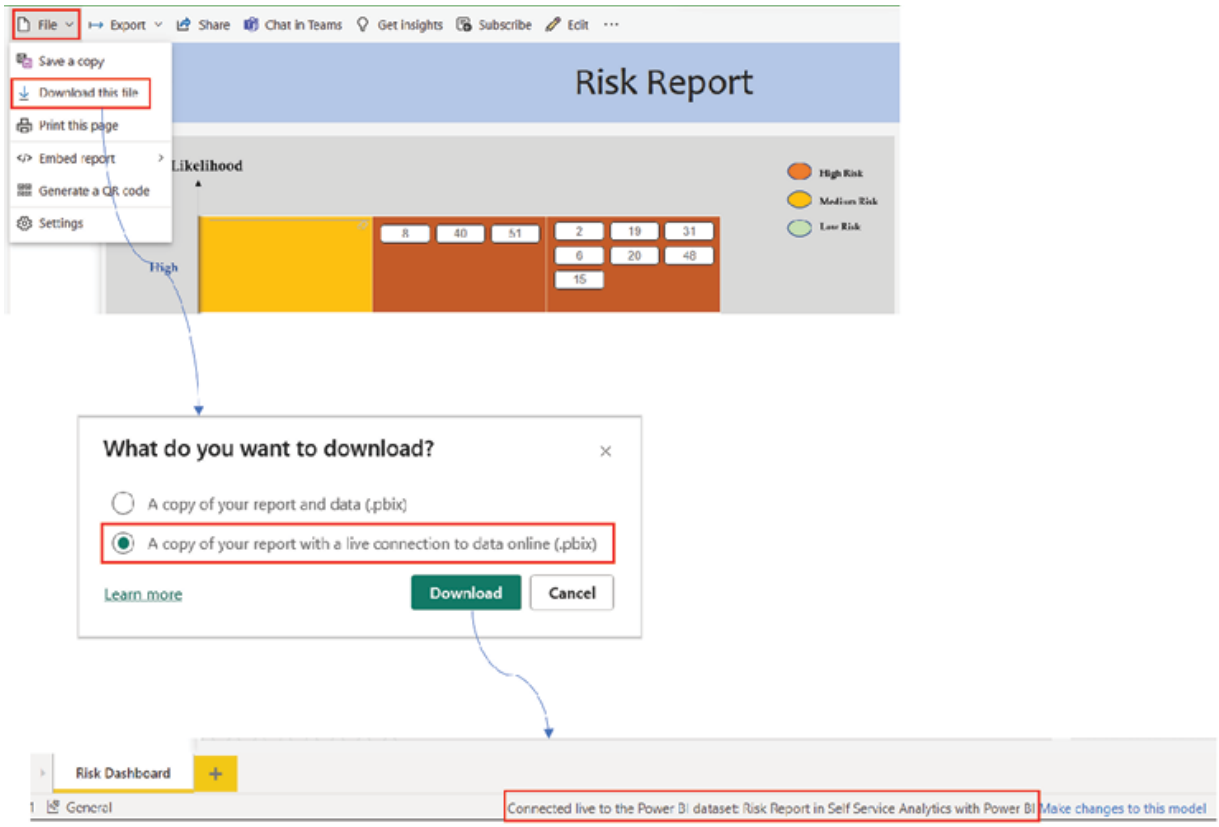



Figure Downloading a live connected report

Let us now rename this report as Risk Report Live and publish it to a newly created workspace named Reports: Self Service Analytics with Power Once published, the live connected report would be the only entity for this new workspace. We can check the lineage of the report from More options of the Risk Report Live and selecting View as shown in Figure

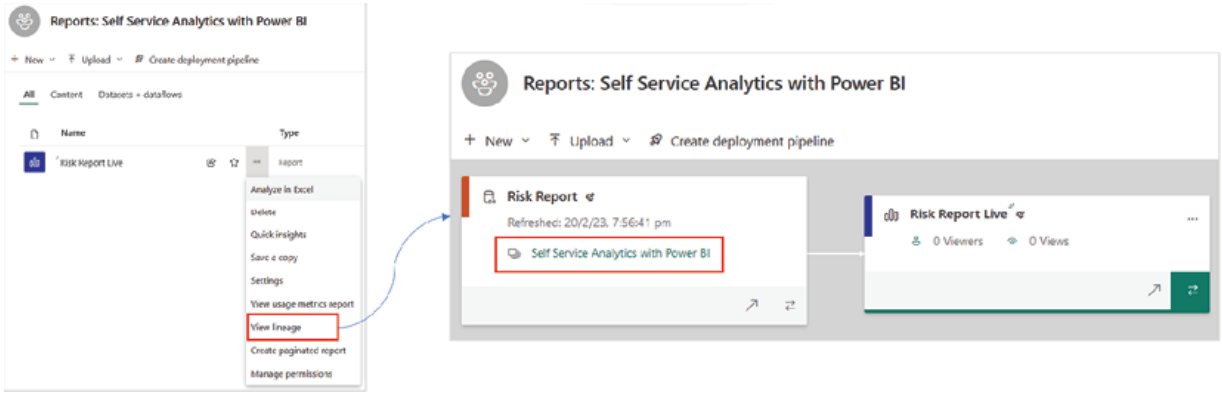



Figure Viewing lineage of a report

As can be seen on the lineage diagram, the Risk Report Live is connected to the dataset published in the workspace Self Service Analytics with Power maintaining a single point of truth.

Similarly, the lineage for a dataset can also be checked from More options of the dataset itself.

Now, we can add the Developer distribution list or group to the Self Service Analytics with Power BI workspace, the Super User distribution list to the Reports: Self Service Analytics with Power BI workspace (with a higher role than and then publish the App from the report workspace, adding both the Super User and End User distribution lists as the App This way every user group would have clear ownership and responsibilities, which should in turn reduce ambiguity and compliance concerns while working in a large, cross-functional team environment.

Note: In a cross-workspace design, the App Audience need to have access to the underlying dataset as the dataset resides in a separate workspace from the one from which the App is published.

The permission for a dataset can be managed from the dataset More Users can be given access by selecting Manage permissions and then the Add user button, as shown in Figure

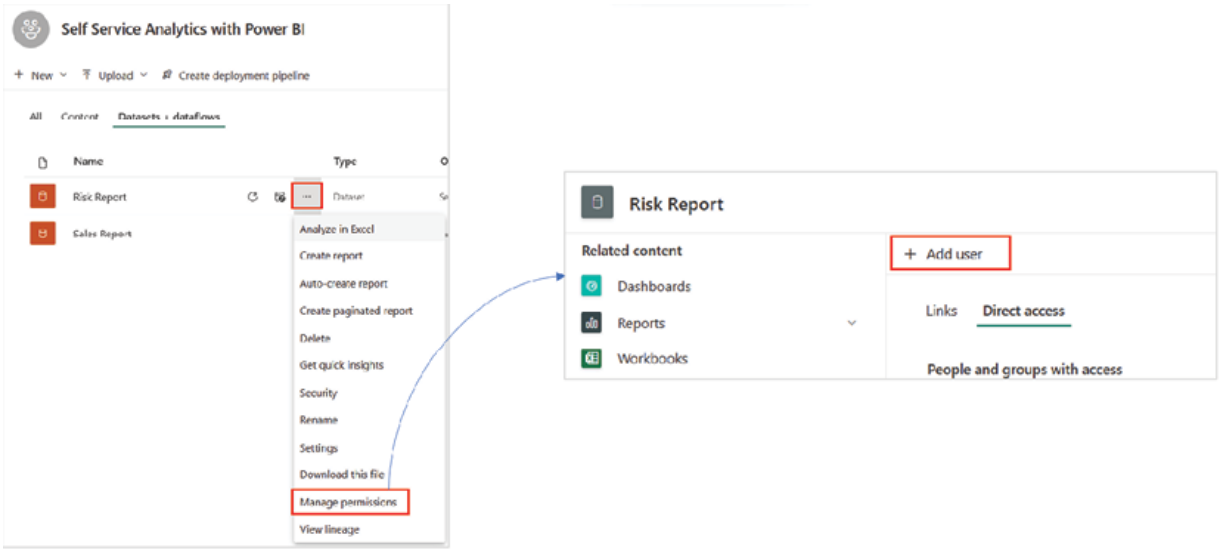



Figure Managing dataset permissions

All users of any workspace having roles above Viewer (which means Members and would have build permission to all the datasets in that workspace by default.

Time to see now how the end users can personalize the risk report that is shared with them via an App and create content on their own using the underlying data.

## Report personalization

The primary aspect of a self-service model is to enable end users to perform basic maintenance activities themselves like adding additional columns, introducing new visuals etcetera, to fulfil individual aspirations, without actually altering the original report that has been provided by the developers. In our risk report, the end users would access the report via the Risk and would not be able to make changes to the original report itself (only Super Users who have access to the report workspace would be able to edit and save the original report and update the However, for the end users to be able to create their own customized views in the a few additional configurations need to be done.

We talked about providing the end users or the App Audience access to the underlying dataset in the last section. While adding users to the risk dataset, we would also provide them with build permission which enables creating content using the data associated with the dataset. From the Manage permissions option of a dataset, selecting Add user opens the window for granting access and providing build as highlighted in Figure

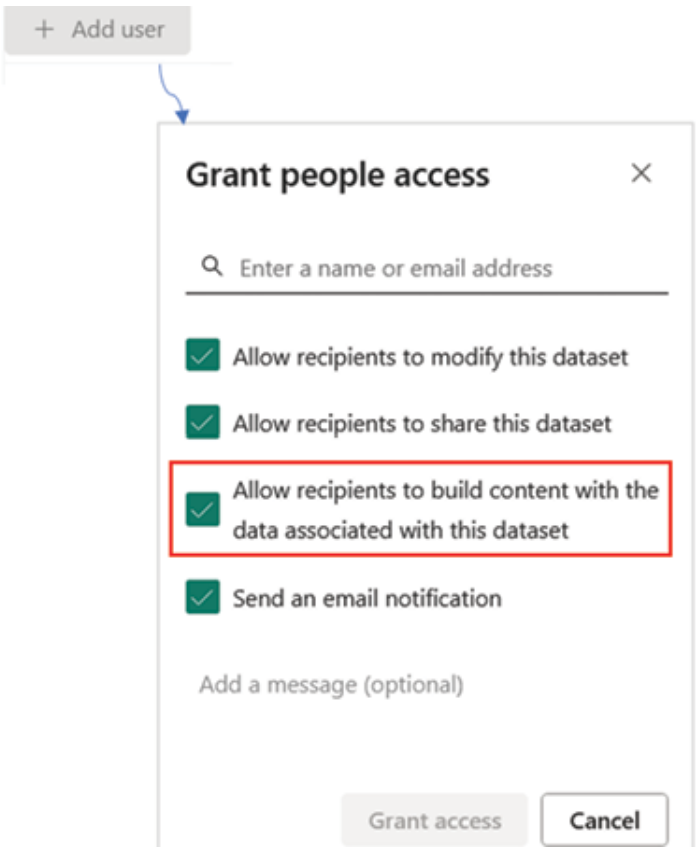



Figure Providing build permission to the End User distribution list

Apart from that, we would enable the option for personalizing visuals from the report settings. Let us go back to the Reports: Self Service Analytics with Power BI workspace and from More options of the Risk Report select The Settings window has multiple options for report level configurations, of which, let us turn on the Personalize visuals toggle and save, as shown in Figure

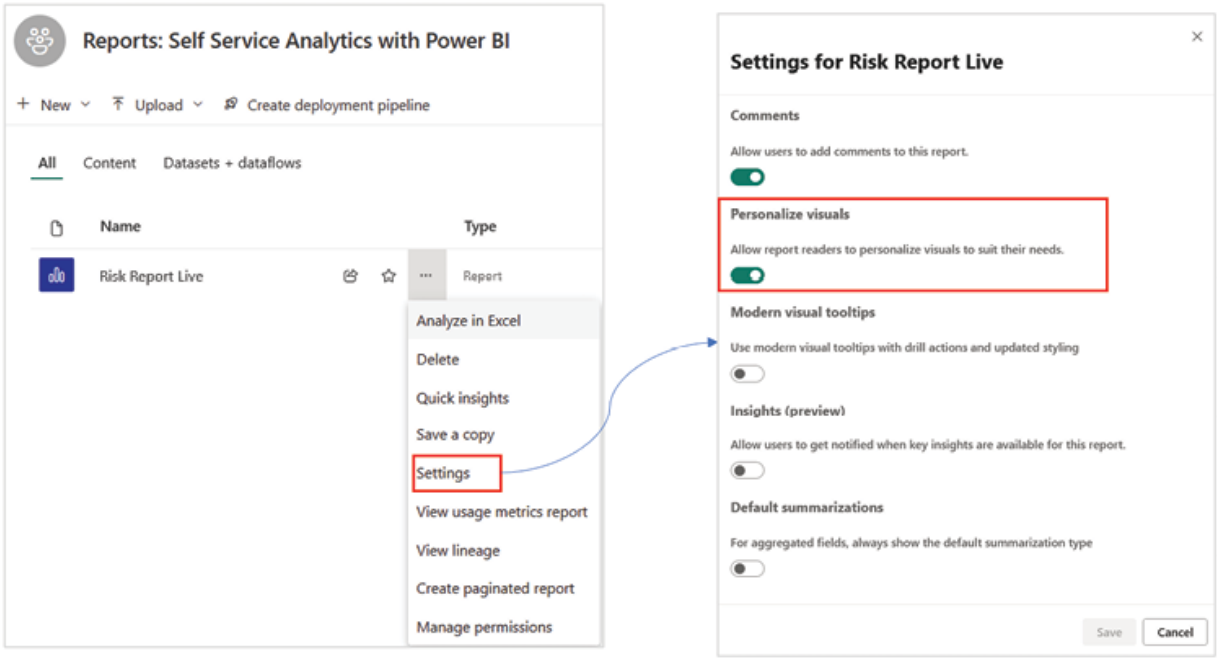



Figure Enabling personalization for visuals

To reflect the changes for the Risk the App needs to be updated from the report workspace. Once done, this should allow the App users new exploration capabilities along with capturing and sharing their personalized views.

If we go to the Risk App now, hovering over the default visuals like Table and Slicer should show the Personalize this visual option. Once selected, the Personalize window opens which allows customizations like changing the Visualization adding new columns using the plus icon etcetera.

Figure 8.24 illustrates how to add a new column Action in the Table that we have in our risk report, for the App

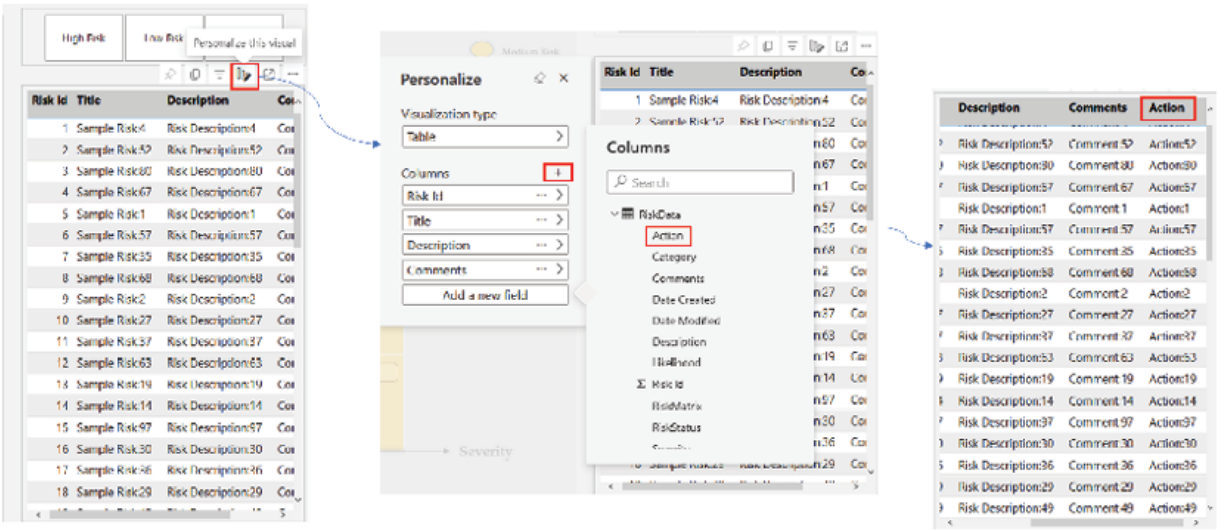



Figure Adding a new column to the table visual in Risk App

However, this change would not persist automatically unless saved as a bookmark. An App bookmark captures the current state of the report. To save a bookmark in the the Bookmarks option on the top right ribbon can be selected followed by selecting Add a personal as shown in Figure

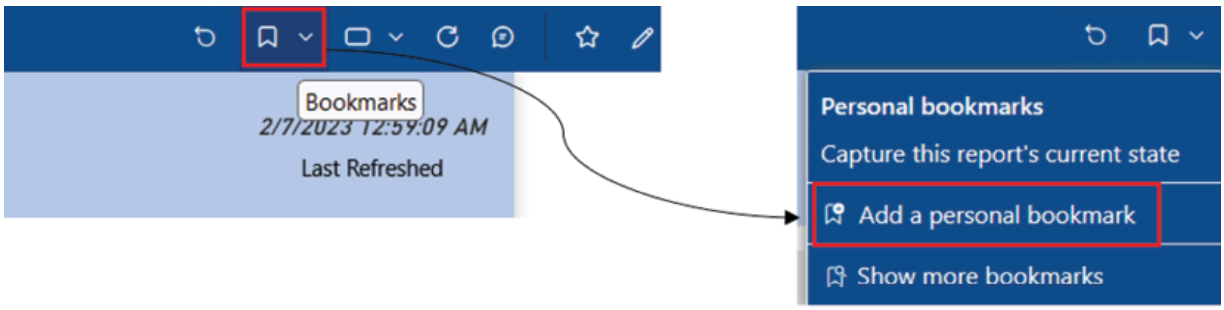



Figure Adding a personal bookmark

Once added, the bookmark can be used to access the personalized view (here the table with an added column) while the default view would not display any personalization unless the bookmark is chosen as the default view.

Apart from personalizing reports in the App users should also be able to make a copy of the report as they already have build permission to the underlying dataset. The other pre-requisite for this capability is to select the option Allow users to make a copy of the reports in this app under the Setup section while publishing the as shown in Figure

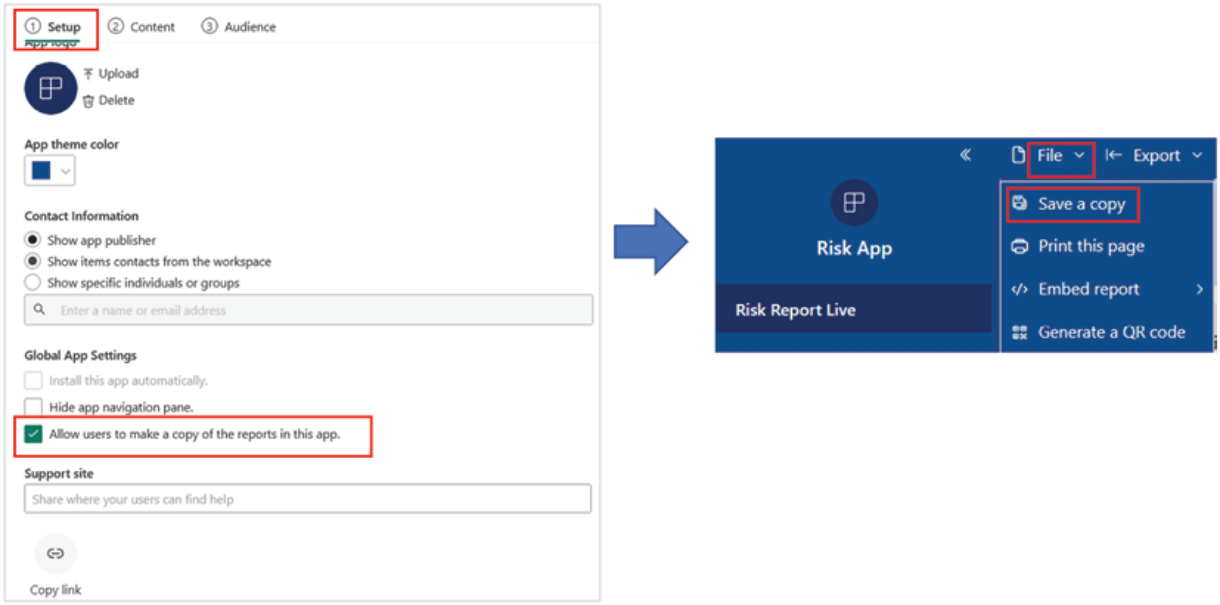



Figure Allowing App users to create copies of reports

Inside the following File | Save a a copy of the report can be created and saved in any workspace the user has access to. The report copy can then also be edited and personalized by the App All of these options help to create a sustainable self-service model in Power BI for a large user base.

## Analyze in Excel

In the previous sections of the chapter, we have seen how the Risk App that we created fulfils all the requirement specifications that we initially discussed. We have provided end users permission to build content on top of the underlying data along the way, which apart from helping to create a self-service model, also helps in terms of live connecting Excel workbooks to the underlying dataset. However, this feature can be enabled or disabled from the tenant settings.

In the Risk following Export | Analyze in a live connected Excel workbook can be generated. This workbook would have the entire data of the dataset and can be used for quick analysis using pivot tables and other Excel features. Figure 8.27 illustrates the process:

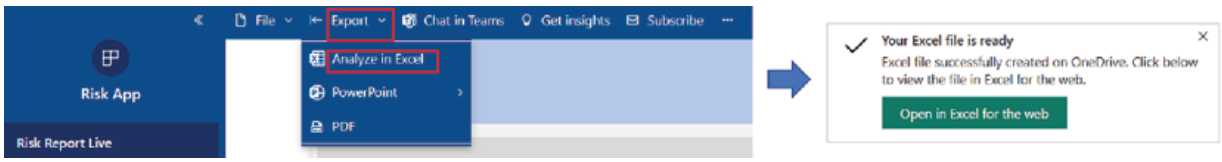



Figure Analyze in Excel option in Power BI Service

The live connected file would have the same name as the report and by default gets saved in the OneDrive account. In the absence of a OneDrive account, it gets saved on the local computer. The file can also be accessed directly on the web. Once the file is opened, all the tables, columns and measures of the Power BI dataset would be available under the PivotTable ready to be dragged and dropped for building pivot tables in Excel, as shown in Figure

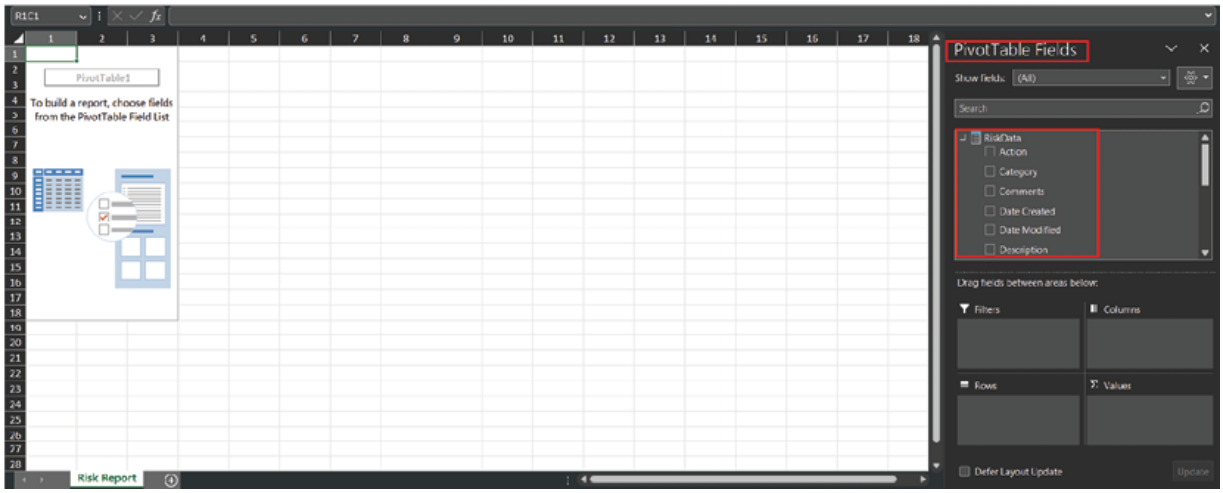



Figure Live connected Excel on a local computer

Alternatively, a Power BI dataset can be connected live directly from an Excel desktop application, following Data | Get Data | From Power Platform | From Power BI in Excel, provided that the user is signed in to Excel with the same organizational account as used for Power BI Service. Once connected, all the datasets that the user has permission to use, should be visible and can be accessed via a live connection, directly from Excel.

The Excel workbooks are refreshable and can be refreshed directly from inside the workbook following Data | Refresh ensuring that users are always working with up-to-date data. While accessed in Excel, properties like dataset endorsement labels and sensitivity labels are inherited, making sure that the data always remains secure.

## Conclusion

In this chapter, we have developed an end-to-end solution for a business problem from scratch and deployed it enabling a self-service model for the end business users. Along the way we have learnt new concepts like cross workspace design, analyzing data in Excel and report personalization. Besides that, many concepts that we have learnt throughout the book have been applied in the solution which should also help readers to understand how to make best use of the learnings in real-life business use cases. Hopefully, this should help the readers of the book to become more confident in their Power BI journey and encourage them to embrace the technology at their workplaces to create an enhanced data culture.

## Knowledge Check

If a report and the dataset are published in separate workspaces, then the audience of the App which is created from the report workspace must have which permission (following the principle of least privilege):

Access permission to the underlying dataset

The report workspace Member access

The report workspace Contributor access

To refresh data in an Excel file connected live with a Power BI dataset, the file needs to be re-generated from Power BI Service:

True

False

Report personalization option only allows to add new columns in a Table or

True

False

All Knowledge Check answers are provided at the end of the book.

**Join our book’s Discord space**

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

# Epilogue

Congratulations readers for finishing all the chapters of the book! Hope it helped you in the journey of becoming a Power BI expert! Learning is a continuous process and although in the book, we have discussed all aspects of Power BI in detail, this part of the book will have additional information about official documentations which can be referred further to remain up to date in the ever evolving landscape of Power BI. Apart from that, Power BI has an active online community for seeking help, the details of which has also been provided. Best of luck!

Official DAX function reference:

https://learn.microsoft.com/en-us/dax/dax-function-reference

Power BI

https://powerbi.microsoft.com/en-au/blog/

Power BI

https://community.fabric.microsoft.com/t5/Microsoft-Power-BI-Community/ct-p/powerbi

Knowledge Check Answers

## Chapter 1

1. a 2. b 3. c 4. b

## Chapter 2

1. b 2. c 3. a 4. b

## Chapter 3

1. c 2. c 3. a 4. d

## Chapter 4

1. d 2. c 3. a 4. a

## Chapter 5

1. b & c 2. b 3. a 4. a

## Chapter 6

1. b 2. a 3. a

## Chapter 7

1. a 2. b 3. b

## Chapter 8

1. a 2. b 3. b

Index

**A**

**B**

**C**

**E**

**F**

**G**



**I**

**L**



**M**

**P**

**Q**



**R**

**S**

**T**

## V



## W

## X