Advanced Power BI

Dave Melillo

Table of Contents

Session 1: Introduction

Welcome! Over the next few days we will explore Microsoft Power BI, a powerful business intelligence and data visualization tool that empowers organizations to transform raw data into actionable insights. Power BI is designed to help businesses make informed decisions, discover trends, and gain a competitive edge in today's data-driven world.

What is Power BI?

Power BI is a suite of business analytics tools that allows you to connect to various data sources, transform data, create interactive visual reports and dashboards, and share them with your team or stakeholders. It provides end-to-end data analytics capabilities, from data preparation to visualization, all within a user-friendly interface.

Core Features of Power BI:

Data Connectivity: Power BI offers a wide range of connectors to connect to various data sources, including databases, cloud services, Excel files, and more. This flexibility enables you to access and analyze data from multiple platforms.

Data Transformation: With Power Query, you can clean, shape, and transform your data to make it suitable for analysis. This is crucial for ensuring the quality and accuracy of your insights.

Data Modeling: Power BI allows you to create data models with relationships between tables. This capability is essential for combining data from multiple sources and enabling complex analysis.

Data Visualization: One of Power BI's strengths is its rich set of visualization options. You can create stunning and interactive charts, graphs, maps, and tables to represent your data effectively.

DAX (Data Analysis Expressions): DAX is a formula language that enables you to create custom calculations, measures, and expressions. It's a powerful tool for advanced data analysis and manipulation.

Power BI Desktop: This desktop application is where you design and build your reports and dashboards. It provides a drag-and-drop interface for creating visuals and defining data models.

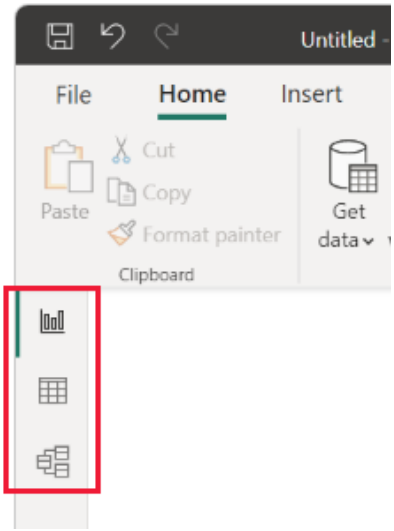
Power BI Service: The cloud-based service allows you to publish and share your reports securely with colleagues or clients. You can access your reports from anywhere using a web browser.

Collaboration: Power BI fosters collaboration within teams by allowing users to comment on and discuss reports, share insights, and collaborate on data-driven decisions.

Natural Language Query: Users can ask questions in plain English, and Power BI will generate visualizations and insights based on the query, making data exploration more accessible.

Security and Governance: Power BI provides robust security features, including role-based access control, encryption, and data loss prevention, ensuring data remains protected.

Power BI is a versatile and user-friendly tool that enables organizations to harness the power of their data. By connecting, transforming, and visualizing data, it empowers users to make data-driven decisions, uncover hidden insights, and drive business growth. As we delve deeper into this course, you will become proficient in leveraging Power BI's advanced features to unlock the full potential of your data analytics journey.

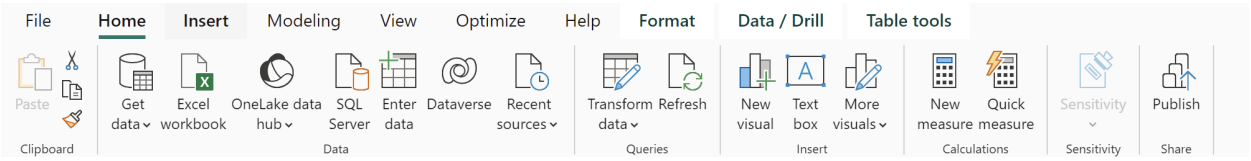


There are three views available in Power BI Desktop, which you select on the left side of the canvas. The views, shown in the order they appear, are as follows:

Report: You create reports and visuals, where most of your creation time is spent.

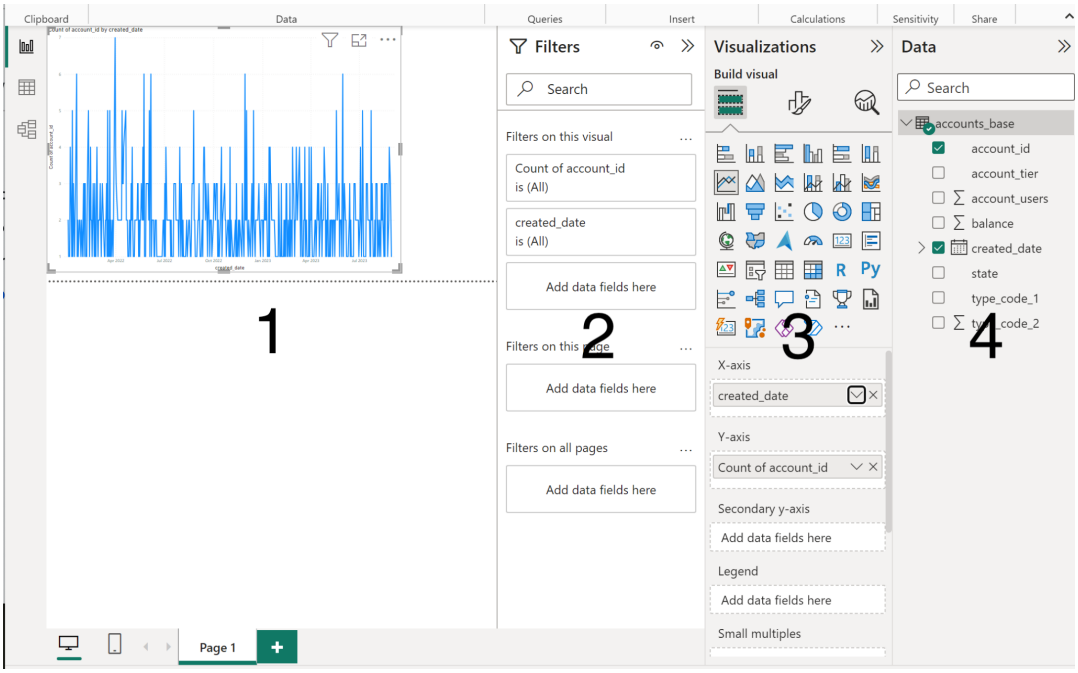
Data: You see the tables, measures, and other data used in the data model associated with your report, and transform the data for best use in the report's model.

Model: You see and manage the relationships among tables in your data mode



The Home ribbon in Power BI is a central hub for essential actions and functions related to managing your reports and data. It provides quick access to commands for tasks like opening, saving, and exporting reports, as well as options for formatting visuals and managing data connections.

Report View



1. Canvas:

The Canvas is the primary workspace where you design and arrange visuals (charts, tables, maps, etc.) to create your reports and dashboards. You can drag and drop visuals onto the canvas and adjust their size and position to craft compelling data stories. Interactivity, such as clicking on visuals for filtering, can also be configured on the canvas.

2. Filter Shelf:

The Filter Shelf is a crucial area for controlling the interactivity and data filtering in your report. Here, you can add filters and slicers, which allow users to interact with the data by selecting specific criteria. Filters can be based on various fields from your dataset, helping you focus on relevant data subsets and enabling dynamic exploration.

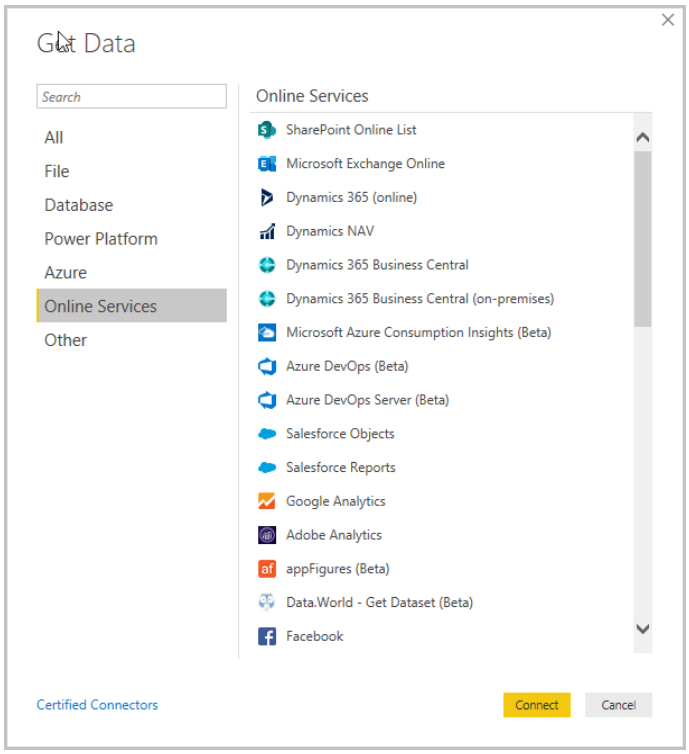
3. Visualization Shelf:

The Visualization Shelf is where you choose and configure the type of visuals you want to display on the canvas. Power BI offers a wide range of visualizations, including bar charts, pie charts, scatter plots, and more. You can select a visualization type, drag and drop fields onto it to define data categories, and customize visual properties like colors, labels, and titles to effectively convey insights.

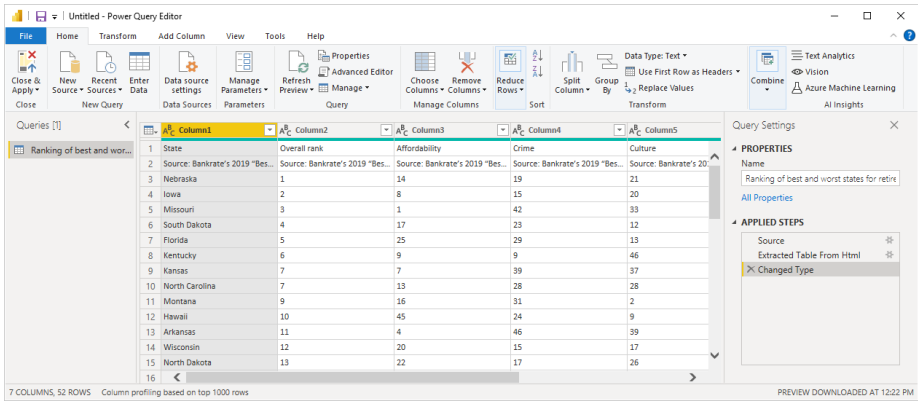
4. Data Shelf:

The Data Shelf provides an overview of the data fields available in your dataset. You can drag fields from the Data Shelf to the various areas of your visuals to define data roles (e.g., X-axis, Y-axis, values, categories) for your visualizations. This area also allows you to create calculated columns and measures using DAX (Data Analysis Expressions) to perform advanced calculations on your data.

To get started with Power BI Desktop, the first step is to connect to data. There are many different data sources you can connect to from Power BI Desktop.From the Home ribbon, select Get Data > More. The Get Data window appears, showing the many categories to which Power BI Desktop can connect.



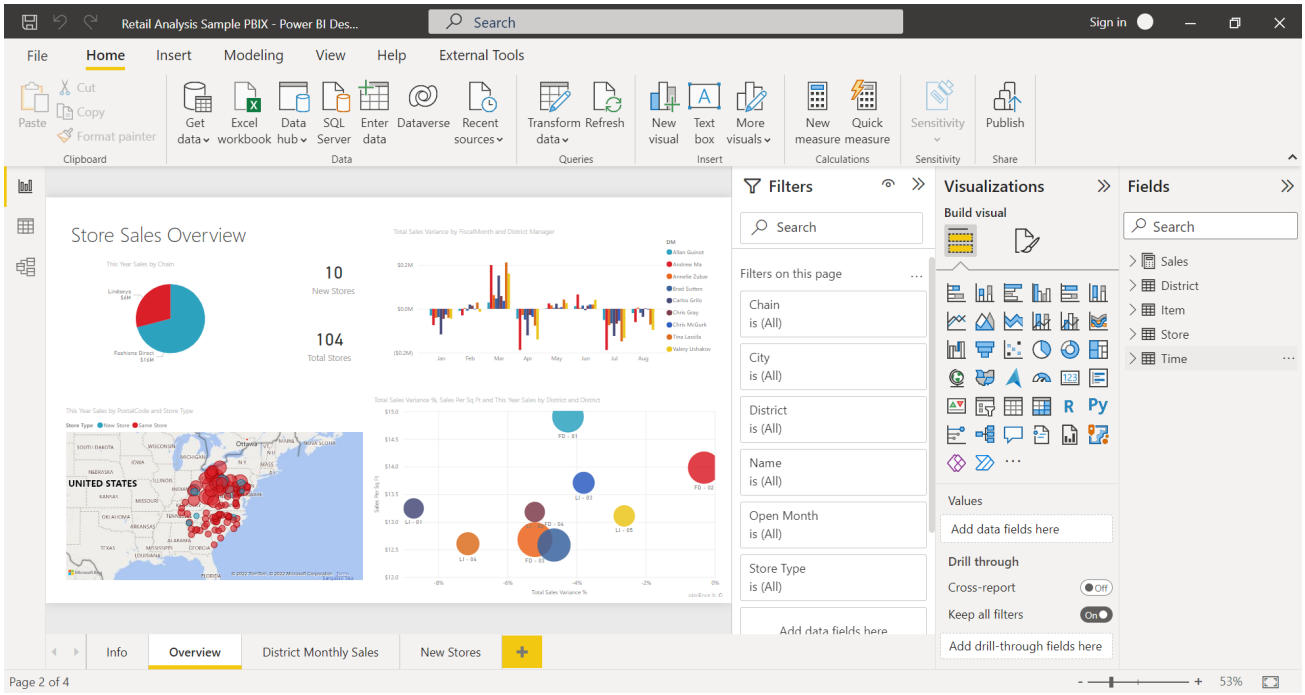
In Power BI Desktop, you can clean and transform data using the built-in Power Query Editor. With Power Query Editor, you make changes to your data, such as changing a data type, removing columns, or combining data from multiple sources. It's like sculpting: you start with a large block of clay (or data), then shave off pieces or add others as needed, until the shape of the data is how you want it.



More often, you'll want to create a collection of visuals that show various aspects of the data you've used to create your model in Power BI Desktop. A collection of visuals, in one Power BI Desktop file, is called a *report*. A report can have one or more pages, just like an Excel file can have one or more worksheets.

With Power BI Desktop you can create complex and visually rich reports, using data from multiple sources, all in one report that you can share with others in your organization.

After a report is ready to share with others, you can *publish* the report to the Power BI service, and make it available to anyone in your organization who has a Power BI license.



Scoping & Designing Effective BI Projects

Design starts with understanding your data and the basic tools you have in Power BI

Please refer to the **fin_serv_accounts** files found in the accompanying GitHub repository for the duration of this exercise.

Problem Statement: FinServ Inc, a fictitious company that offers insurance, financial and investment products to the public, needs you to do some analysis on a list of customer accounts which includes accounts balances, account users and other details.

The three files we were given are:

- **accounts_base.csv**
    - A summary of customer accounts with descriptive details such as account balance, creation date, customer tier, and more.
- **accounts_states.csv**
    - Descriptive information about each state in the USA, such as what region they belong to. This will be helpful to summarize accounts by different dimensions.
- **accounts_types.csv**
    - Descriptive information about each account product offered by FinServ Inc.
- **recent_transactions.csv**
    - Recent transactions with descriptive attributes such as date, time and type. Many to one relationship with accounts_base.

Identify Audience

Identifying the audience is one of the most important steps in the report design process. It enables the report author to create a final result that can be efficiently used and will meet the needs of the report consumer.

The three broad report consumer audiences are:

- Executive
- Analyst
- Information worker

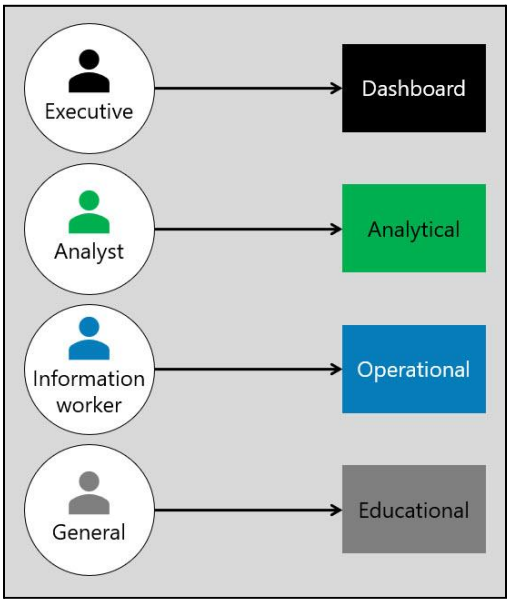
Determine Report Types

Generally, report design can be classified by report type. Often, a direct mapping between the report audience and the report type occurs. Audience needs can be met by one, or possibly a combination, of four report types:

- Dashboard
- Analytical
- Operational

- Educational

Commonly, executives work with dashboards, analysts work with analytical reports, and information workers work with operational reports.



Define User Interface Requirements

- Form Factor
- Input Method
- Style and Theme

Define User Experience Requirements

- Support for interactions, such as:
  - Drill up, drill down, or drill through to details.
  - Navigation within the report or to other reports.
  - Filters or slicers that can be applied to report visuals, specific pages, or all pages.
  - Data export as specific data formats, such as Microsoft Excel or a comma-separated value (CSV) file.
- Support for ad hoc questions to retrieve a response in the form of a data visualization.
- Configuring of data alerts to notify people when specific data values change or exceed predefined thresholds.
- Links to open webpages.
- Actions to open applications, write back data entry values, or trigger workflows.
- What-if analysis that allows the report consumer to modify "what-if" values to understand the consequences of different scenarios. For example, what-if analysis could allow consumers to predict sales revenue based on different consumer demand estimates.
- Page layouts that can extend over multiple pages and are suitable for printing as multi-page documents.
- Printing the report to a physical printer or as a PDF document.
- Subscribing to the report so that it can be automatically delivered as a document on a scheduled basis.

Instructor Do: Demonstrate Power BI linear workflow; Ingest, Transform, Visualize

csv: accounts_base.csv
pbix: d1_s1_instructor.pbix

This is a basic exercise meant to demonstrate, quickly, the Power BI ingest, transform and visualize workflow. In the following exercise, we will focus on

1. Click ‘Get Data’ on the Home Ribbon.

2. Choose Text/CSV and press the green ‘Connect’ button

3. When the next data preview window appears choose the ‘Transform Data’ option.

4. In the PowerQuery editor, select the ‘state’ column and adjust the filter to exclude N/A values. ‘Close & Apply’ the changes in the PowerQuery editor.

5. After the query loads, select a line chart visualization from the Visualizations shelf and drag it onto the Report View canvas. Drag ‘created_date’ from the account_base query available in the Data shelf to the X-axis of the visualization and drag account_id to the Y axis. Make any necessary adjustments to confirm data can be visualized.

Session 2: Core ETL with Power BI

Now that we've demonstrated how to ingest, transform and visualize data quickly. We will focus on applying the common transformations below to all of the datasets in the fin_serv_accounts directory. Full walk through available in the 'Student Do' activity.

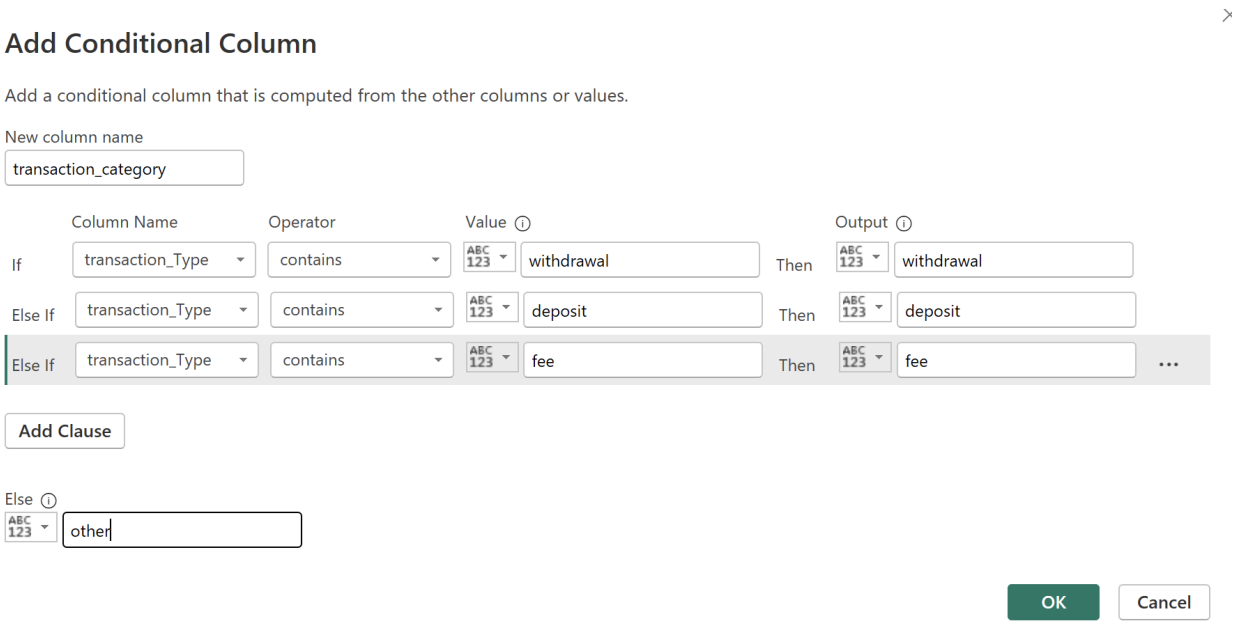
- Identify column headers and names
- Promote headers
- Rename columns
- Remove columns
- Rename a query
- Replace values
- Replace values
- Remove duplicates
- Change the column data type
- Parse Text
- Concatenate Text
- Add Conditional Columns

Student Do: Core ETL with Power BI

csv: all files in fin_serv_accounts directory
pbix: d1_s2_student.pbix

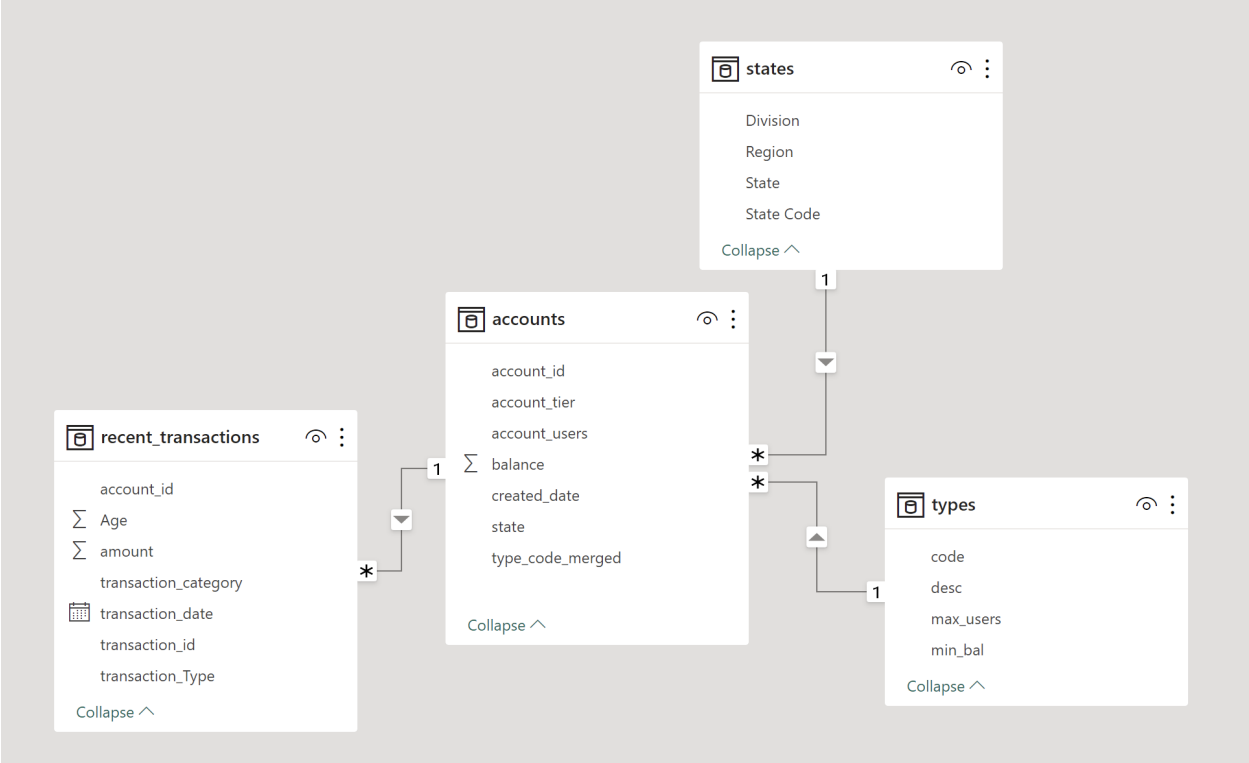
1. Click 'Get Data' in the Home Ribbon and select Text/CSV as the source option.
2. Choose the 'accounts_base' csv file to start with.
    a. Change the name of query to accounts
    b. Filter out N/A values in the state field
    c. Change the Data Type of balance to Decimal or Whole Number
    d. Extract Text Before Delimiter (-) in the account_tier field
    e. Change the Data Type of type_code_2 to Text
    f. Merge type_code_1 and type_code_2 into a new field called type_code_merged
    g. Close & Apply
    h. Challenge: Create a simple line chart with created_date on the X axis and Count of account_id on the Y axis.
3. Click 'Get Data' in the Home Ribbon and select Text/CSV as the source option.
4. Choose the 'accounts_states' csv file.
    a. Change the name of the query to states
    b. In the Home tab, click the Use First Rows as Headers transformation
    c. Close & Apply
    d. Challenge: Create a simple Filled Map with state as Location and Division as Legend

5. Click ‘Get Data’ in the Home Ribbon and select Text/CSV as the source option.
6. Choose the ‘accounts_types’ csv file.
    a. Change the name of the query to types
    b. Close & Apply
    c. Challenge: Create a bar chart with desc on the Y-Axis and Average of Max Users on the X-Axis
7. Click ‘Get Data’ in the Home Ribbon and select Text/CSV as the source option.
8. Choose the ‘recent_transactions’ csv file.
    a. Rename the type field to transaction_type
    b. Rename the date field to transaction_date
    c. Create an Age field based on the transaction_date
    d. Create a conditional column called transaction_category as configured below



    e. Close and Apply
    f. Challenge: Create a line chart with transaction_date on the X-Axis and Count of account_id on the Y axis
9. We now have all of the data sets loaded into Power BI with basic transformations applied. It is time to create a relation amongst all of them so we can quickly and easily create new measures and visuals.
10. Navigate to the Model View
11. Create the following associations between queries by dragging your mouse from one object to another after highlighting the associated keys
    a. accounts.state > states.State Code
    b. accounts.type_code_merged > types.code

c. Accounts should already be related to recent_transactions via account_id, but if not, make the association now
d. If done correctly, your Model View should resemble the image below
e. Challenge: Go back to the Report View and create a new page. On the new page create a Matrix visualization with states.Region and types.desc on the Rows and Sum of accounts.balance and Sum of transactions.amount on the Values. Expand all of the subsections of the Matrix and confirm that there are values for each desc.



Session 3: Advanced ETL with Power BI

Appending Queries:
Appending queries is a method in Power BI where you combine rows from two or more queries into a single table by stacking them on top of each other. This approach is useful when you have data from similar sources with identical structures, and you want to create a single, larger dataset without altering the original data. Appending is like stacking one table on top of another, creating a union of the rows.

Merging Queries:
Merging queries, on the other hand, involves combining data from different tables based on a common column, typically using some form of key or identifier. This method is beneficial when you have related but separate datasets, and you need to consolidate information by linking records together based on shared values in a specific column. Merging creates a new table that combines data from multiple sources based on the defined relationships.

Group By:

The "Group By" transformation in Power BI is a powerful data manipulation operation that allows you to group data based on one or more columns in your dataset. It's often used to aggregate and summarize data, especially when dealing with large datasets.

Key points about the "Group By" transformation:

Aggregation: You can apply various aggregation functions (e.g., sum, count, average, minimum, maximum) to columns within each group to perform calculations on grouped data.

Grouping Columns: Select one or more columns by which you want to group the data. The selected columns determine how the data is partitioned into distinct groups.

New Columns: The transformation creates new columns in your dataset, one for each aggregation function applied, resulting in summarized data for each group.

Summarization: The transformation helps in summarizing data based on the criteria you specify, making it easier to analyze and visualize trends or patterns.

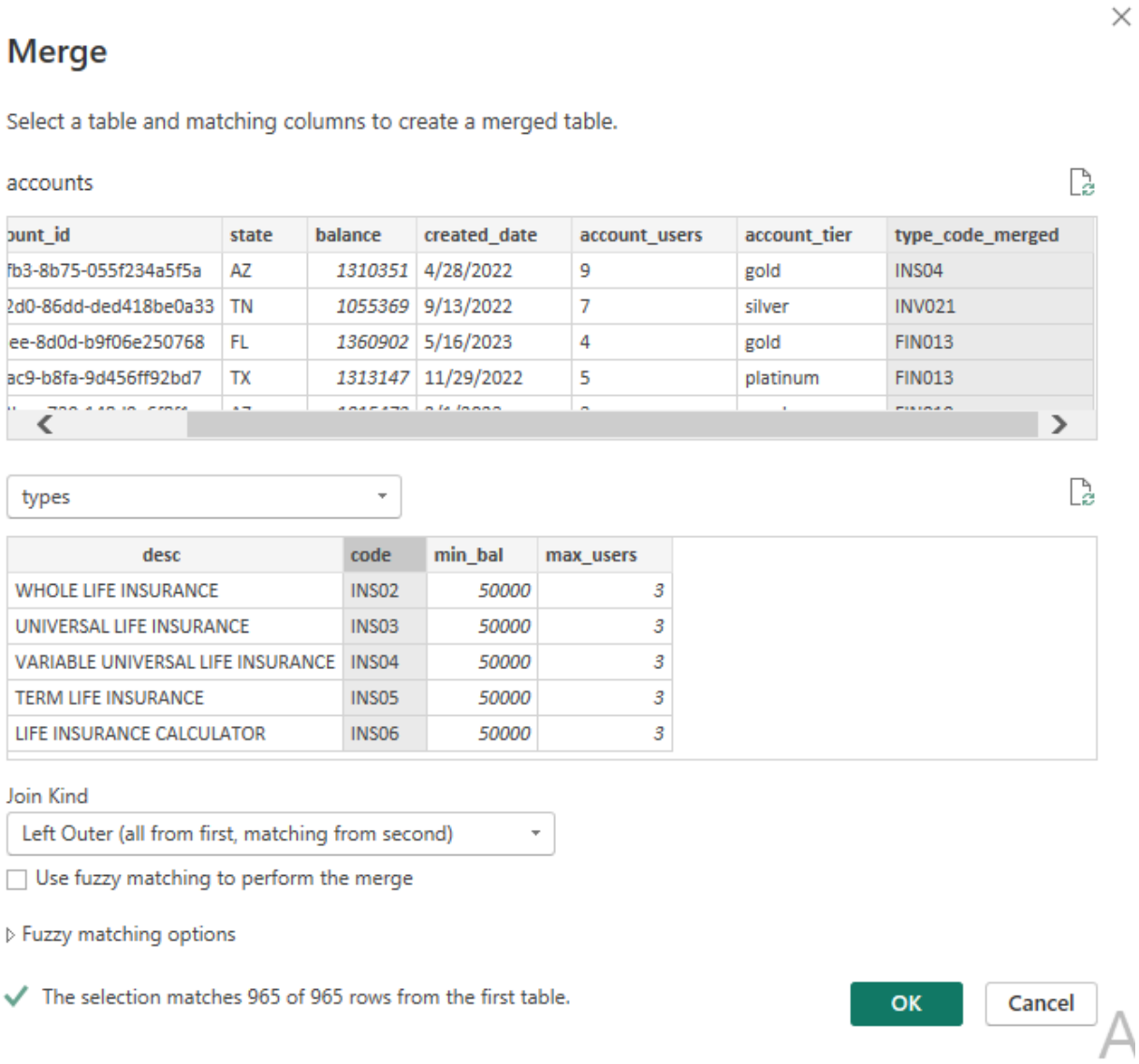
Use Cases: Common use cases for "Group By" include creating summary tables, generating aggregated reports, calculating statistics, and more.

By using the "Group By" transformation, you can efficiently condense and analyze data, making it a valuable tool for data modeling and analysis in Power BI.

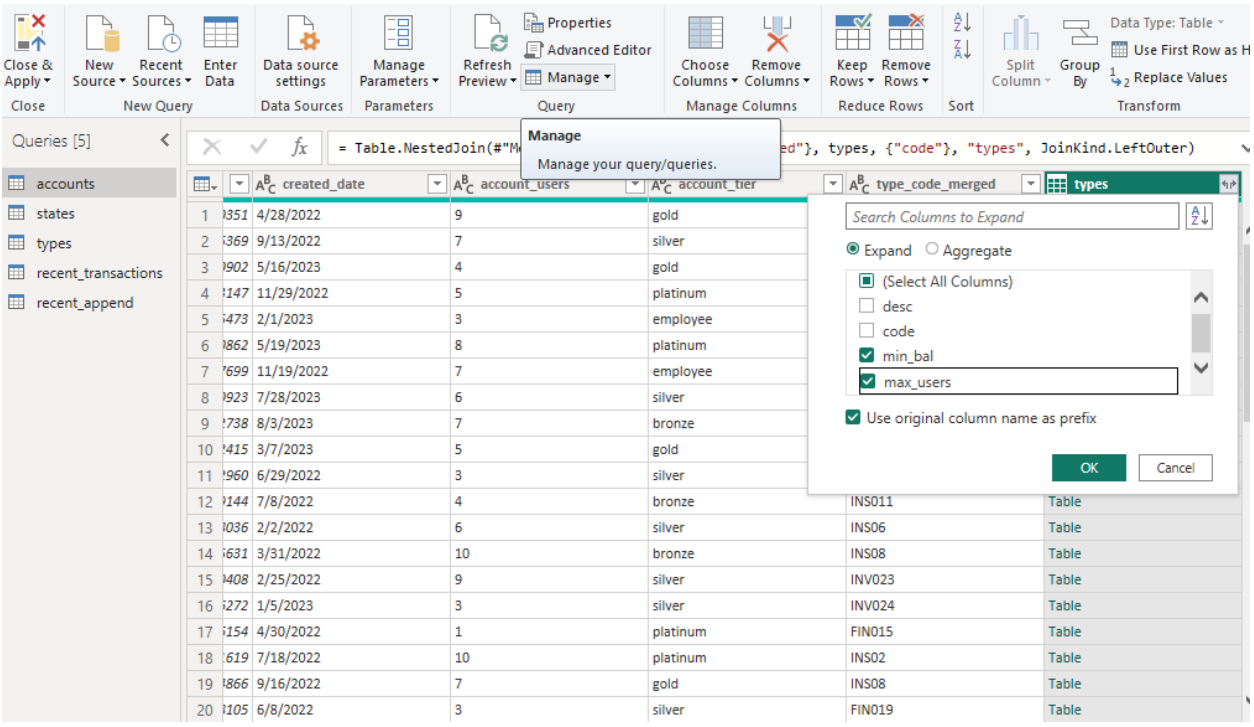
Student Do: Advanced ETL with Power BI

csv: all files in fin_serv_accounts directory
pbix: d1_s3_student.pbix

1. Click ‘Get Data’ in the Home Ribbon and select Text/CSV as the source option.
2. Choose the ‘recent_append’ csv file and the “Transform Data” option to navigate into the Power Query Editor.
3. Now in the Power Query Editor, navigate to the recent_transactions query.
    a. Remove all applied steps up to “Source”
    b. Apply the “Use First Row as Headers” transformation
    c. Change the data type of the date field to date
    d. In the “Home” ribbon choose the “Combine” option and the “Append Queries” sub option
    e. In the Append Queries dialog box choose “Two Tables” and recent_append as the value for “Table to append”
    f. Add in previous transformations on the new appended data set
        i. Rename date to transaction_date
        ii. Add an Age field based of transaction_date
        iii. Add a conditional column called transaction_category based on the type field
    g. Close & Apply Changes
    h. Extra Credit: Create a line chart with date on the X axis and amount on the Y axis from recent_transactions to confirm append
4. Navigate back to the Power Query Editor to the “accounts” query
    a. In the “Home” ribbon choose the “Combine” option and the “Merge Queries” sub option
    b. In the “Merge” dialog box, choose “types” as the query to be merged and create an association between type_code_merged in the “accounts” query and “code” in the “types” query
    c. Leave the rest of the settings as-is (reference the image below) and press OK to navigate back to the Power Query Editor

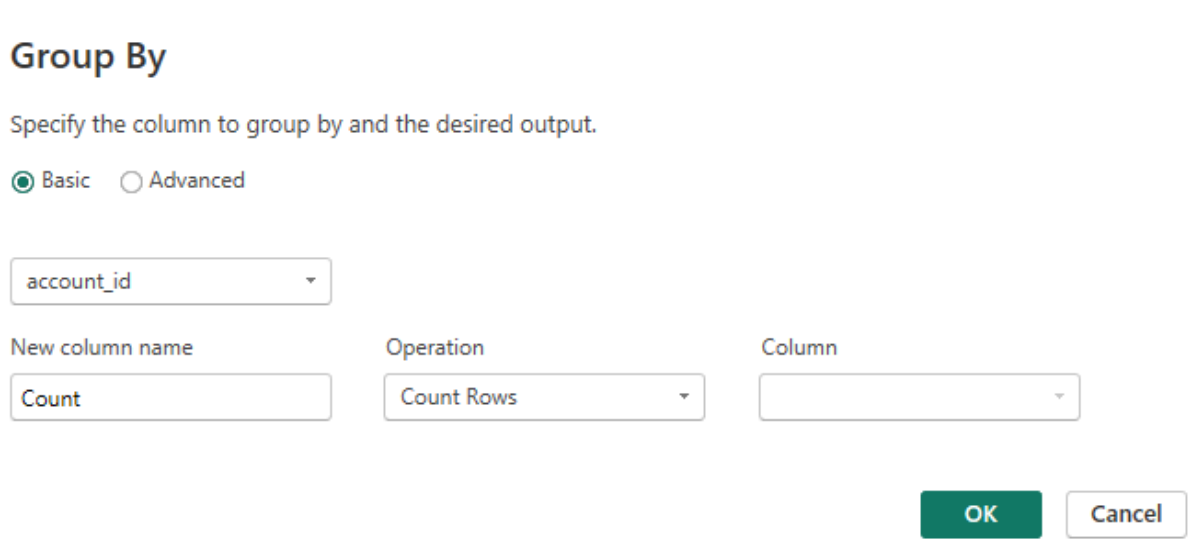


d. Now back in the “accounts” query, press the icon next to the new types field and select the min_bal and max_users options in the sub menu. Refer to image below for assistance.

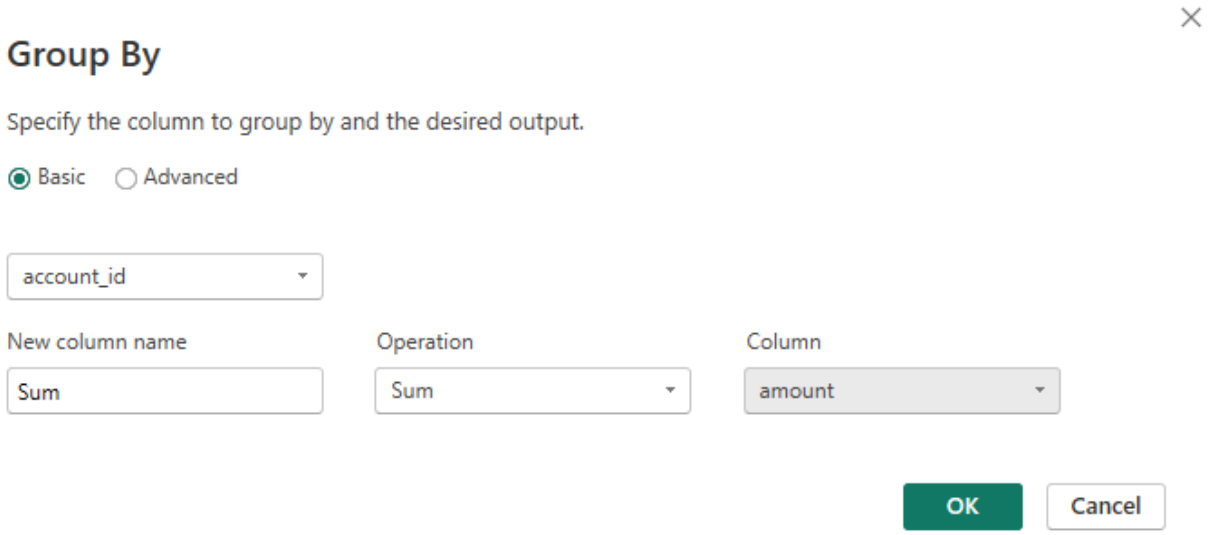


    e. Optionally, rename these newly Merged columns
    f. Close & Apply Changes
5. Navigate back to the Power Query Editor
6. Duplicate the recent_transactions query
7. Rename this query account_recent_transactions_summary
8. In the “Transform” ribbon select the Group By option
    a. In the Group By dialog box, choose Basic, account_id as the Group By column, Count as the New Column Name and Count Rows as the

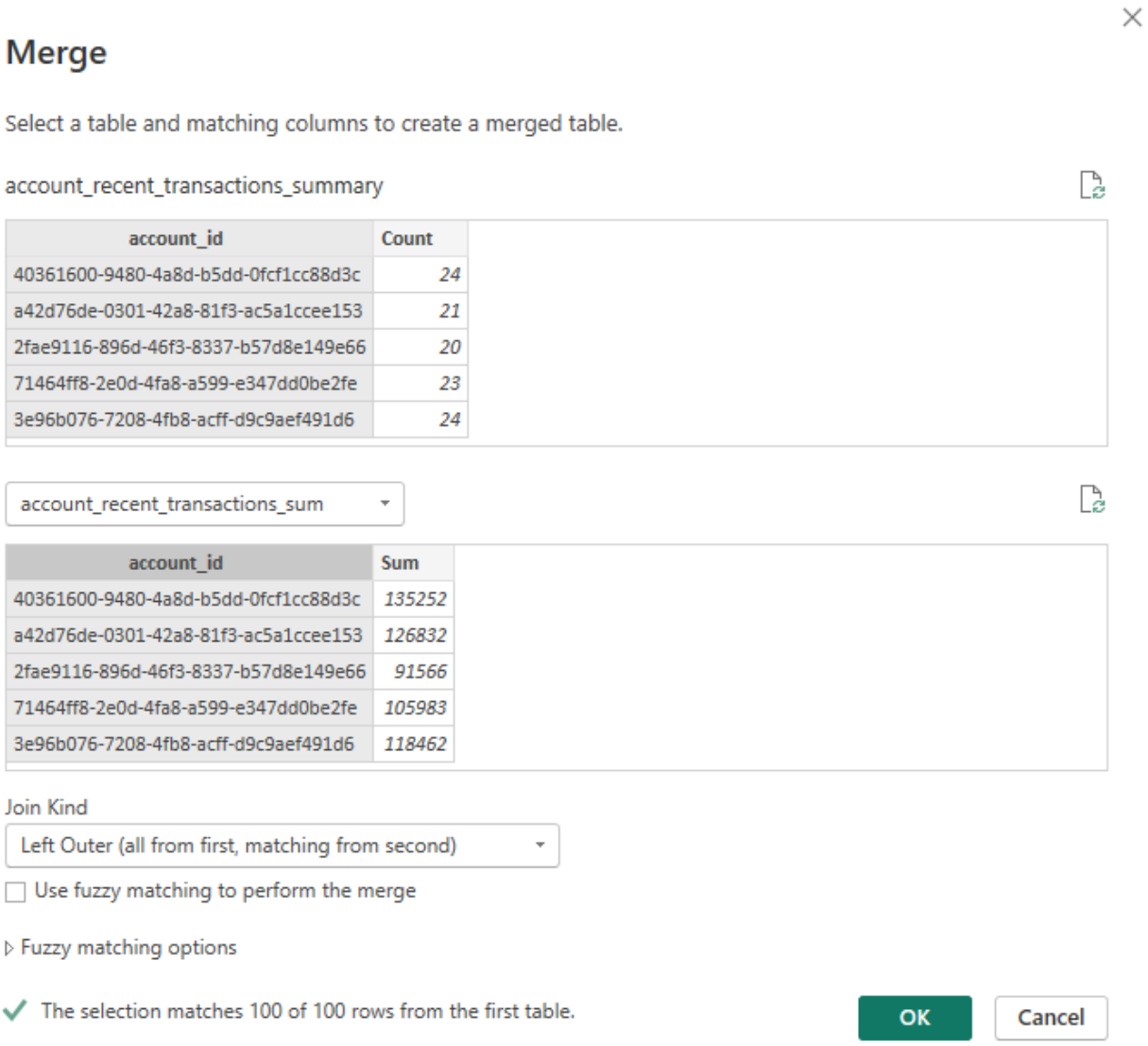
operation



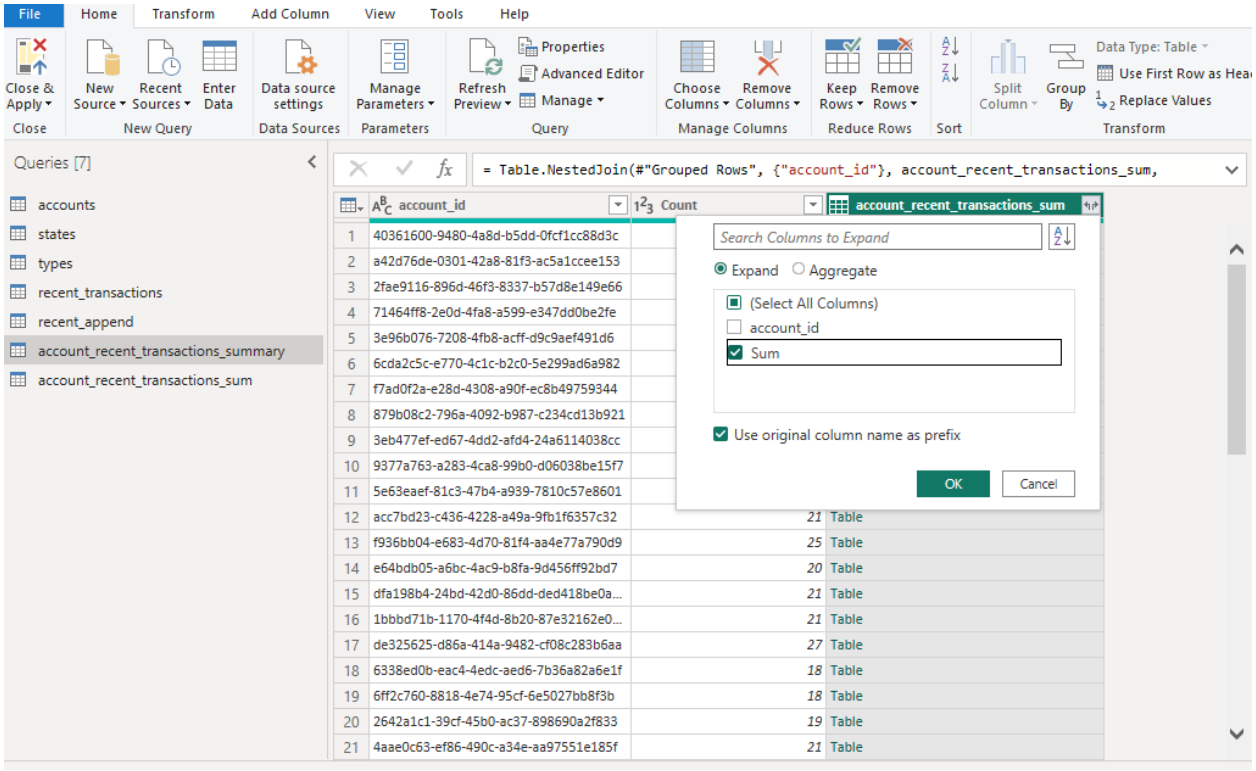
9. Duplicate the recent_transactions query
10. Rename this query account_recent_transactions_sum
11. In the “Transform” ribbon select the Group By option
    a. In the Group By dialog box, choose Basic, account_id as the Group By column, Sum as the New Column Name, Sum as the operation and amount as the Column.



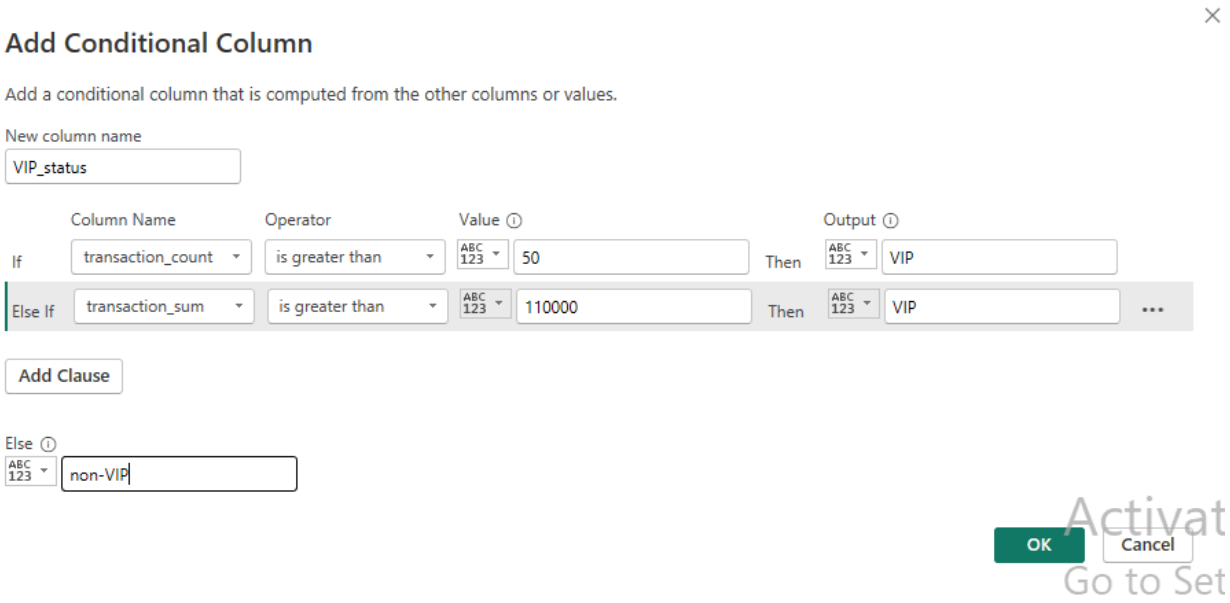
12. Navigate back to account_recent_transactions_summary query, choose “Combine” from the Home Ribbon and the Merge Queries sub option
13. In the Merge dialog, choose account_recent_transactions_sum as the join table and make an association between the account_id in both data sets. Leave the rest of the default configurations and press OK.



14. In the new column that appears, click the icon next to the column name and choose only the Sum column to merge into account_recent_transactions_summary



15. You should now have three columns in account_recent_transactions_summary. Change the name of the Count column to transaction_count and the Sum column to transaction_sum.
16. Add a new conditional column called VIP_status with the following configuration



17. Close & Apply Changes

18. Challenge: Create a pie chart using the new VIP_status column in account_recent_transactions_summary with VIP_status as the legend and Count of account_id as the Values.

Session 4: Core Visualizations

Line Chart: In Power BI, line charts are frequently employed to visually depict data trends over time. They are particularly effective for showing how a specific metric or value evolves and changes continuously. For instance, you can use a line chart to showcase the revenue growth of a product over months or years, enabling you to easily identify patterns and trends in your data.

Bar Chart: Bar charts are widely used in Power BI to compare and contrast discrete data categories. They are ideal for illustrating the magnitude of data points concerning each other. In practice, bar charts are commonly utilized to compare data across different categories, such as displaying the sales performance of various products, evaluating revenue across regions, or assessing the performance of employees.

Tabular/Matrix: Tabular or matrix visuals in Power BI are designed to present data in a structured, grid-like format, facilitating the comparison of information across rows and columns. These visuals are essential for showcasing detailed, structured data, often used in tables or pivot tables for presenting raw data or creating cross-tabulated summaries.

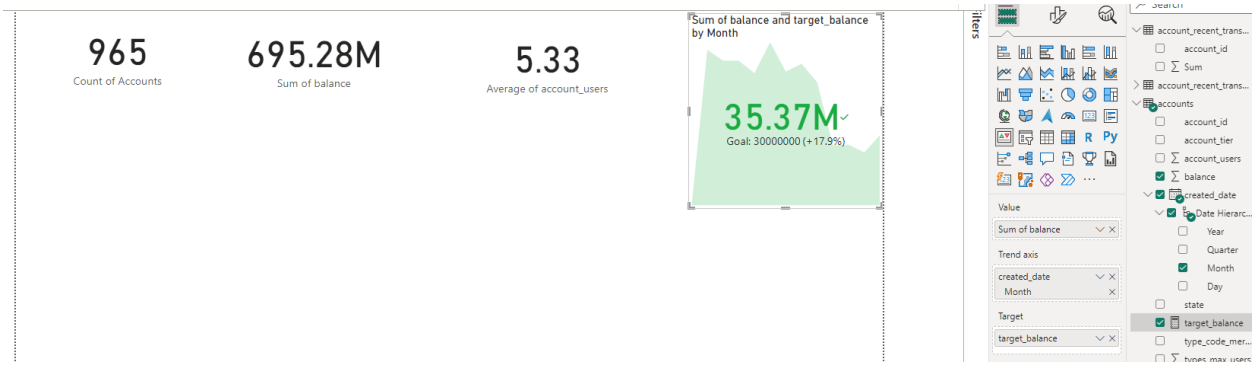
Pie Chart: Pie charts in Power BI are effective for showing the relative contribution of each category to a whole. They are utilized to highlight the proportion of different components within a single entity, such as market share among products in a specific industry.

KPI/Card: KPI (Key Performance Indicator) cards are specifically designed to present single, high-impact data points, often representing critical metrics or values in Power BI. KPI cards provide a quick snapshot of crucial data points, making it easy to assess performance at a glance. They are typically used to showcase key performance indicators like total sales, current revenue, or customer satisfaction scores, providing immediate insights into essential aspects of your data.

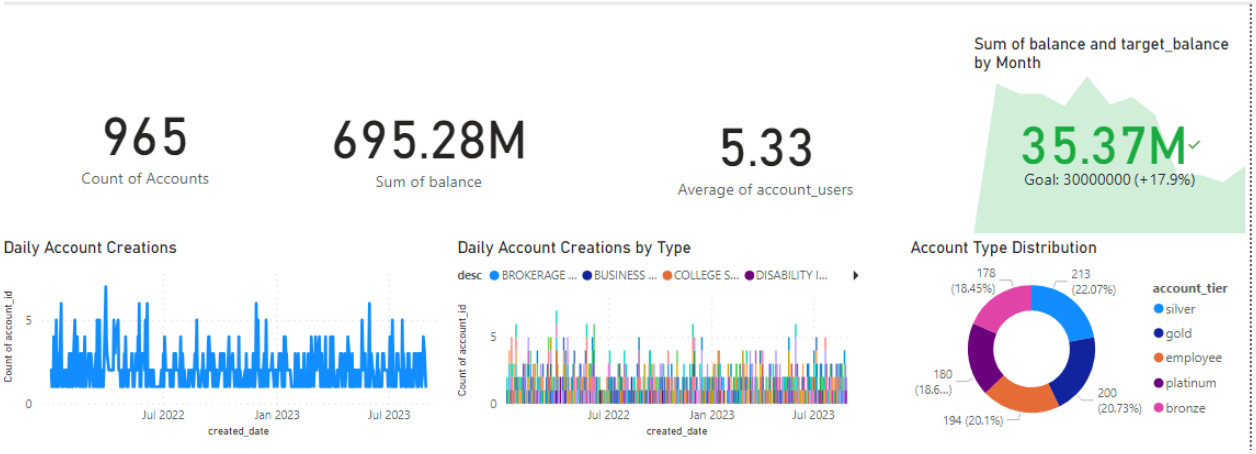
Student Do: Core Visualizations

csv: all files in fin_serv_accounts directory
pbix: d1_s4_student.pbix

1. Ideally, students are starting with all of the ETL modeling steps from Session 2 and 3.
2. We are going to start by creating some Card and KPI charts on a blank canvas
    a. Create a Card chart with a count distinct of account_id from accounts
    b. Rename the field Count of Accounts
    c. Create a Card chart with a sum of balance from accounts
    d. Create a Card chart with an average of users from accounts. You may need to change the data type of the users field before this is possible.
    e. Create a KPI chart with sum of balance from accounts as the value and created date month as the Trend Axis. You may need to convert the created_date data type to accomplish this.
        i. Create a new measure called target_balance and make it = 30000000
        ii. Use this measure as the target in the KPI card
    f. See the image below for KPI card settings and final Card layout



3. Next we are going to create some line and bar charts
    a. Create a line chart with created_date on the X Axis and count of account_id from accounts on the Y axis. Turn off the date hierarchy option for created_date so it shows daily account creations. Change the title of the chart to Daily Account Creations in Format Visuals > General > Title.
    b. Create a stacked column chart with created_date on the X Axis , count of account_id from accounts on the Y axis and desc from types on the Legend shelf. Change the title of the chart to Daily Account Creations by Type in Format Visuals > General > Title.
4. Time to create a pie/donut chart
    a. Create donut chart withaccount_tier from accounts on the Legend shelf and count distinct account_id from accounts on the Values shelf.
    b. Change the title of the chart to Account Type Distribution in Format Visuals > General > Title.
    c. Your report should look like the below image.



5. Finally lets add a table with some details to finish off our report
    a. Create a Table visual and add created_date, state, balance, account_users and account_tier from accounts. Also add desc from types.
    b. Adjust the column widths manually until the entire width of the Table frame is filled.
    c. Your report should look like the below image.

965
Count of Accounts

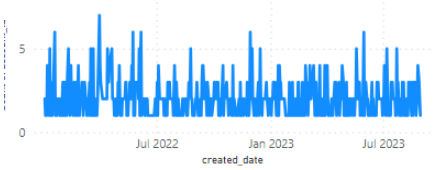
695.28M
Sum of balance

5.33
Average of account_users

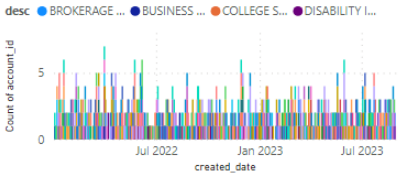
Sum of balance and target_balance by Month

35.37M
Goal: 30000000 (+17.9%)

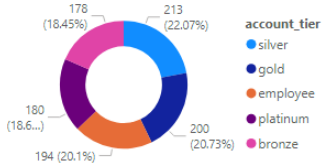
Daily Account Creations



Daily Account Creations by Type



Account Type Distribution



created_date	state	Sum of balance	Sum of account_users	account_tier	desc
Wednesday, August 30, 2023	OR	501,306.00	6	silver	DISABILITY INSURANCE CALCULATOR
Tuesday, August 29, 2023	IA	367,819.00	3	employee	DISABILITY INSURANCEFOR DOCTORS AND D
Tuesday, August 29, 2023	MO	464,075.00	3	employee	LIFE INSURANCE CALCULATOR
Monday, August 28, 2023	MI	1,338,341.00	4	platinum	MARKET COMMENTARY
Monday, August 28, 2023	MO	612,116.00	6	silver	UNIVERSAL LIFE INSURANCE
Monday, August 28, 2023	OH	371,636.00	4	bronze	BUSINESS SERVICES
Saturday, August 26, 2023	CA	311,566.00	1	silver	PRIVATE WEALTH MANAGEMENT
Saturday, August 26, 2023	FL	532,770.00	5	gold	VARIABLE UNIVERSAL LIFE INSURANCE
Saturday, August 26, 2023	LA	1,396,518.00	1	bronze	MARKET COMMENTARY
Total		**695,280,616.00**	**5140**		

Session 5: Advanced Visualizations

Bubble Charts/Scatter Plots: Bubble charts and scatter plots are used to show multiple dimensions of data by comparing entities in terms of their relative values, positions, and sizes. They are valuable for visualizing relationships and identifying patterns in data points.

Funnel Charts: Funnel charts display data in a funnel-like structure, illustrating a process or sequence of steps, often used for sales and marketing to visualize conversion rates at each stage.

Waterfall Charts: Waterfall charts represent incremental changes in data, typically used to show how an initial value is affected by a series of positive and negative values. They are useful for understanding the cumulative impact of various factors on a total.

Matrices and Hierarchies: Matrices and hierarchies are used to display data in a structured grid or tree-like format, making it easier to compare and analyze data in a tabular or hierarchical manner. They are suitable for multi-dimensional data.

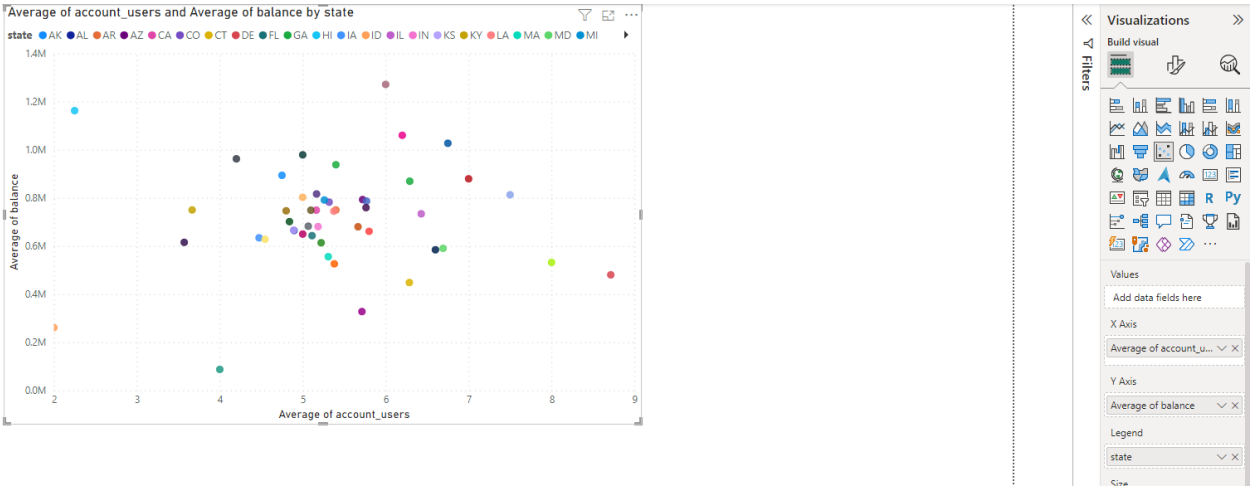
Maps: Maps are best for showing a geographical representation of data. They can display data by location, helping to identify regional trends or disparities. Common uses include plotting sales data, population statistics, or any location-based information.

Custom Visuals with R & Python: Custom visuals allow you to incorporate additional data visualization libraries or create custom charts using R or Python code. This flexibility is handy when standard Power BI visuals don't meet your specific requirements.

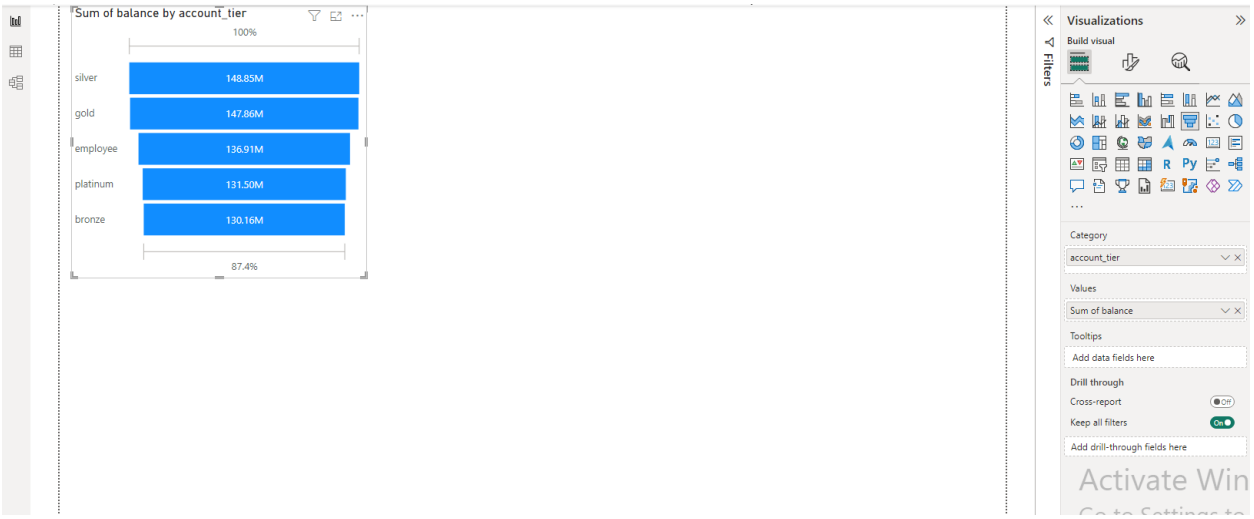
Student Do: Advanced Visualizations

csv: all files in fin_serv_accounts directory
pbix: d1_s5_student.pbix

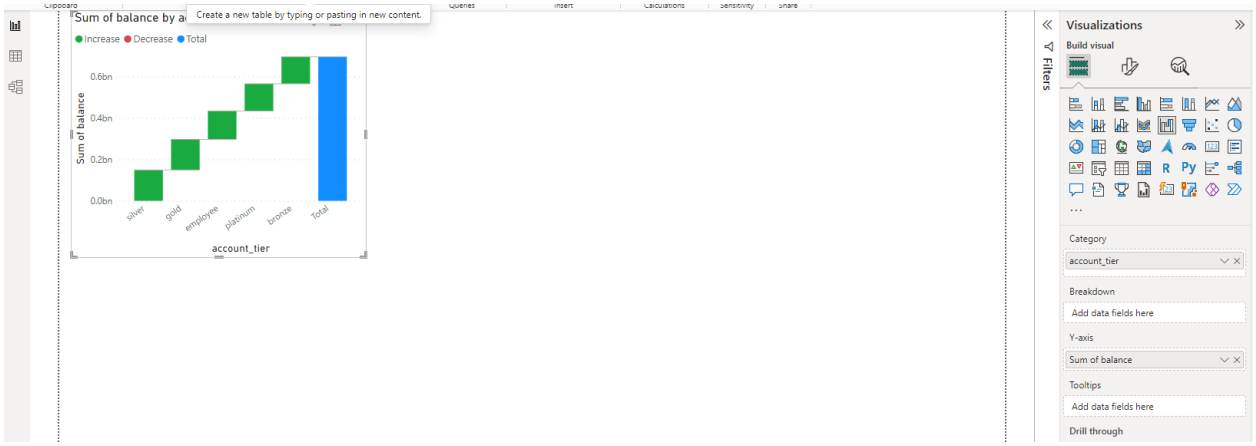
1. For the advanced visuals we will be creating a new page for each visual, unlike the core visuals which we put together on one page.
2. Lets start by creating a scatter chart
    a. Using the accounts query, put average of account_users on the X axis and average of balance on the Y axis. Put state on the legend shelf.
    b. Your chart should look like something below.



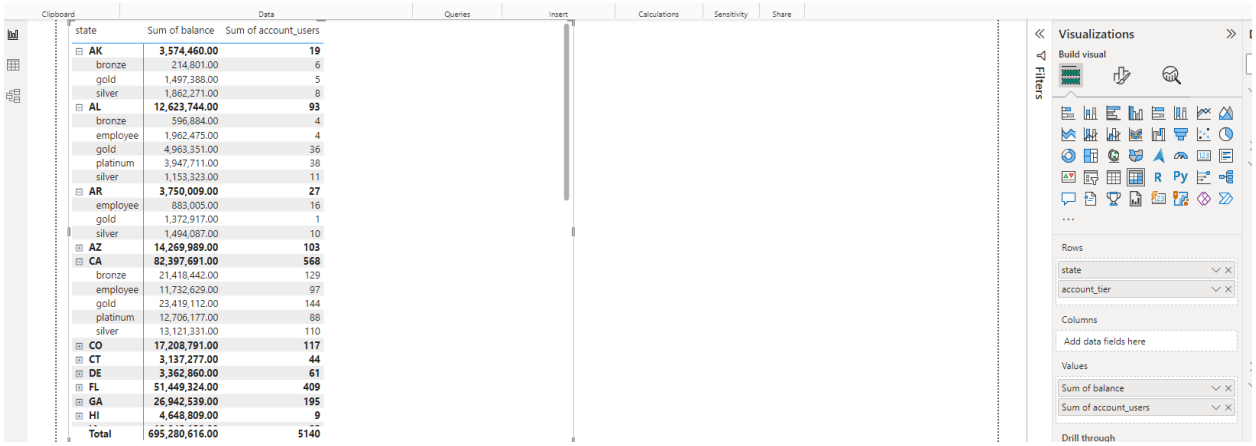
3. Next lets create a funnel chart
    a. Using the accounts query put account_tier on the Category shelf and put the Sum of balance on the Values shelf.
    b. Challenge: Cycle through other categorical attributes on the Category shelf to see how the visual changes. Created date is especially interesting!
    c. Your chart should look like something below.



4. Next is a waterfall chart
    a. Using the accounts query put account_tier on the Category shelf and put Sum of Balance on the Y axis.
    b. Your chart should look like something below.



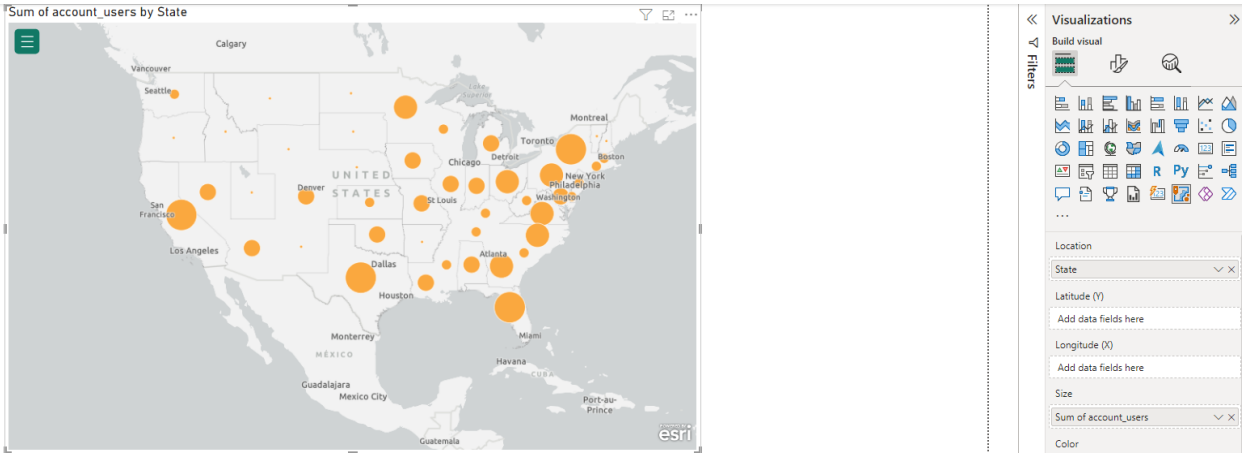
5. Matrix Charts and Hierarchies
    a. Create a Matrix visualization.
    b. Drag the state and account_tier fields from the accounts query to the Rows shelf. This creates a hierarchy which nests account_tier under state.
    c. Drag Sum of Balance and Sum of account_users to the Values shelf.
    d. Your chart should look like something below. Explore the power of hierarchies by expanding each state value to expose the tier metrics per state in a Pivot style chart.



state	Sum of balance	Sum of account_users
AK	3,574,460.00	19
bronze	214,801.00	6
gold	1,497,388.00	5
silver	1,862,271.00	8
AL	12,623,744.00	93
bronze	596,884.00	4
employee	1,962,475.00	4
gold	4,963,351.00	36
platinum	3,947,711.00	38
silver	1,153,323.00	11
AR	3,750,009.00	27
employee	883,005.00	16
gold	1,372,917.00	1
silver	1,494,087.00	10
AZ	14,269,989.00	103
CA	82,397,691.00	568
bronze	21,418,442.00	129
employee	11,732,629.00	97
gold	23,419,112.00	144
platinum	12,706,177.00	88
silver	13,121,331.00	110
CO	17,208,791.00	117
CT	3,137,277.00	44
DE	3,362,860.00	61
FL	51,449,324.00	409
GA	26,942,539.00	195
HI	4,648,809.00	9
Total	695,280,616.00	5140

6. Maps
    a. There are few different maps you can create in Power BI: Map, Filled Map and ArcGIS Maps for Power BI. This demo will use ArcGIS Maps for Power BI.
    b. Drag State from the states query to Location and Sum of account_users to the Size shelf.

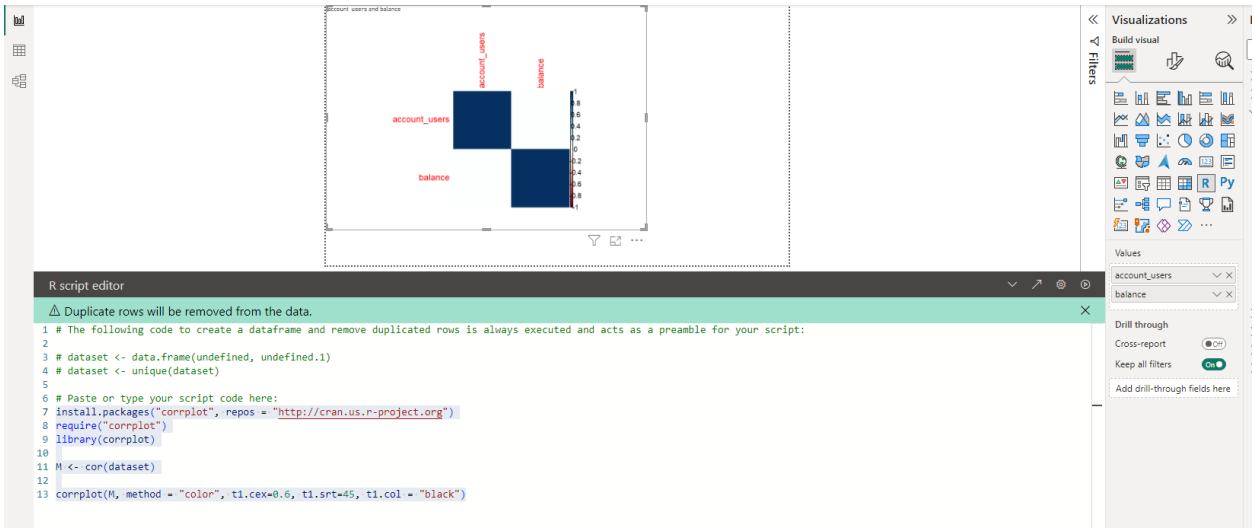
c. Your chart should look like something below



7. Custom visuals with R & Python
    a. R
        i. By default, Power BI Desktop doesn't include, deploy, or install the R engine. To run R scripts in Power BI Desktop, you must separately install R on your local computer. You can download and install R for free from many locations, including from the CRAN Repository. The current release of R scripting in Power BI Desktop supports Unicode characters and spaces (empty characters) in the installation path.
        ii. From the Power BI Desktop menu, select File > Options and settings > Options.
        iii. On the left side of the Options page, under Global, select R scripting.
        iv. Under R script options, verify that your local R installation is specified in Detected R home directories and that it properly reflects the local R installation you want Power BI Desktop to use. In the following image, the path to the local installation of R is C:\Program Files\R Open\R-3.4.4\.
        v. Once setup is complete, navigate to the Report View and create an R Script Visual
        vi. Drag account_users and balance from the accounts queries to the Values shelf.
        vii. You will be prompted with the R script editor, where you can enter the script below:

```r
install.packages("corrplot", repos =
"http://cran.us.r-project.org")
require("corrplot")
library(corrplot)

M <- cor(dataset)

corrplot(M, method = "color", t1.cex=0.6, t1.srt=45,
t1.col = "black")
```

        viii. Upon successful execution, your Report View should resemble the below image.



b. Python

i. Install Python on your local machine.

ii. Enable Python scripting in Power BI Desktop.

iii. Install the pandas and Matplotlib Python libraries.

iv. You will be prompted with the Python script editor, where you can enter the script below:

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

correlation_matrix = data.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True,
cmap='coolwarm', fmt='.2f', linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

Session 6: Day 1 Capstone

csv: all files in pokemon directory
pbix: d1_capstone.pbix

In this capstone project, you will work with Pokémon trading card sales data to build a Power BI report.

The project is designed to reinforce the concepts and skills learned on the first day of the course, covering ETL and visualizations.

Project Objectives

Data Import and Preliminary Inspection:

- Import the Pokémon trading card sales dataset into Power BI.
- Check the data for any missing values and outliers.
- Identify the data types of each column.
- Remove any duplicate values based on ID/transaction ID.

Data Cleaning:

- Handle missing values, if any, using appropriate methods.
- Address any outliers as needed.
- Rename columns and format values to make the data more user-friendly.

Data Transformation and ETL:

- Append Q1, Q2 and Q3 sales into one total pokemon_sales query.
- Create a relationship between the sales data and the Pokemon data.
- Create a conditional column in the Pokemon query to identify Pokemon with a Total > 500 as Powerful Pokemon.
- Create an aggregate table which calculates total sales per Pokemon and then Merge that total sales number into the Pokemon query. There should now be a “total_sales” figure next to each Pokemon.

Basic Visualizations:

- Create Card charts and KPIs for all basic sales metrics; count and amount sold.
- Create a line chart to visualize the trend of total sales over time.
- Build a bar chart to display the top-selling Pokémon by Type 1.
- Generate a pie chart to show the distribution of sales by Generation.

Advanced Visualizations:

- Design a correlation matrix heatmap to explore relationships between Pokemon attributes and their total_sales.
- Create a matrix table visualization to display detailed information about the top-selling cards.
- Pick total_sales and an additional attribute from the Pokemon query and use them in a Scatter Plot to understand the relationship.

Report Layout and Design:

- Organize your report with appropriate chart titles and design.
- Format visuals, colors, and fonts to make your report visually appealing.
- Challenge: Include a filter and build a new measure

Session 7: Recap & Introduction to Data Modelling

Introduction to Data Modeling in Power BI

Data modeling in Power BI is a crucial aspect of creating effective and insightful reports and dashboards. It involves structuring data to support interactive and dynamic data visualization. Central to data modeling in Power BI are the concepts of Entity-Relationship Diagrams (ERDs), facts, dimensions, and various types of joins.

Entity-Relationship Diagram (ERD) in Power BI

While ERDs are more commonly associated with database design, Power BI utilizes similar concepts in its data modeling process. In Power BI, an ERD is a visual representation of tables, columns, and relationships between those tables. These relationships define how different data tables are related, allowing you to combine and analyze data from multiple sources.

Facts and Dimensions in Power BI

In Power BI, facts and dimensions play a pivotal role in data modeling, similar to data warehousing. Here's how these concepts apply:

- Facts: In Power BI, facts represent the numerical measures or values of interest in your reports. They include data such as "Sales Revenue," "Profit," "Quantity Sold," and more. Facts serve as the core of your data model, forming the foundation for analysis and visualization.
- Dimensions: Dimensions provide the context for your facts. They describe the attributes by which you analyze your facts. Dimensions include attributes like "Time" (e.g., date), "Location" (e.g., city or store), "Customer," and "Product." By associating facts with dimensions, you can explore your data from various angles, creating meaningful insights.

Types of Joins in Power BI

Data modeling in Power BI often requires combining data from different tables. To accomplish this, Power BI offers several types of joins:

- Inner Join: An inner join combines rows from two tables based on a matching condition. Only the rows that meet the condition in both tables are included in the result.
- Left Join (or Left Outer Join): A left join combines all the rows from the left table and the matching rows from the right table. If there's no match in the right table, null values are added.
- Right Join (or Right Outer Join): A right join is similar to a left join but includes all the rows from the right table and matching rows from the left table.
- Full Outer Join: A full outer join combines all rows from both tables, including rows with no matches in either table. Null values are added where there's no match.
- Cross Join (or Cartesian Join): A cross join combines each row from the left table with every row from the right table, creating a Cartesian product.

- Anti Join (or Anti Semi Join): An anti join returns rows from the left table where there's no match in the right table.
- Self-Join: In Power BI, you can also join a table with itself, known as a self-join. This is useful for hierarchical or recursive relationships, such as organizational structures.

Instructor Do: Build a star schema data model in Power BI.

csv: all files in the stocks directory
pbix: d2_s7_instructor.pbix

This exercise will serve as a review, as well as an extension of some of the ETL and visualization concepts we learned yesterday, with an emphasis on facts, dimensions and data relationships.

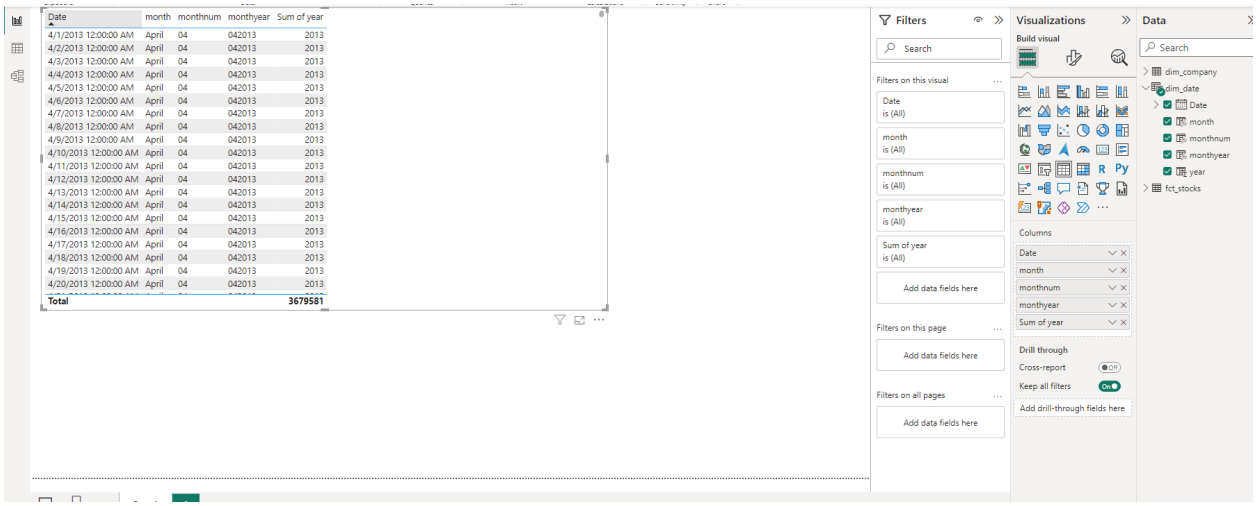
1. Click “Get Data” in the Home Ribbon and select the all_stocks_5yr.csv file.
2. Choose the “Transform Data” option to be sure all the columns are formatted correctly, there are no duplicates and that the data quality is generally good.
3. This will be our main fact table, so lets rename the query fct_stocks
4. Close & Apply
5. Click “Get Data” in the Home Ribbon and select the constituents.csv file.
6. Choose the “Transform Data” option to be sure all the columns are formatted correctly, there are no duplicates and that the data quality is generally good.
7. This will be one of our dimension tables, so lets rename the query dim_company
8. Now we will start to introduce DAX by creating a dim_date table using DAX expressions.
9. Got to the “Modeling” tab on the Home Ribbon and choose “New Table”
10. Use the expression below in order to create a new table called dim_date that is set to the min and max range of the date in our fact table.

```
dim_date = CALENDAR(min(fct_stocks[date]),max(fct_stocks[date]))
```

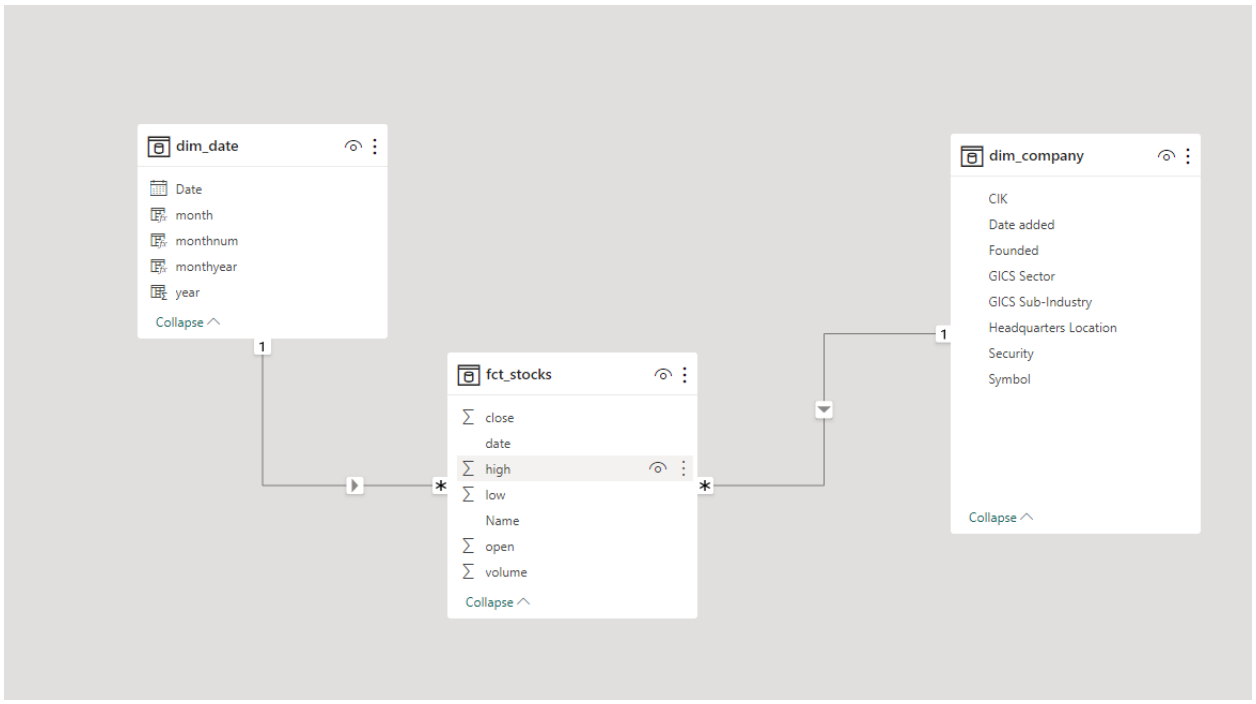
11. With the dim_date table selected create new columns using the expressions below.

```
year = YEAR(dim_date[Date])

month = FORMAT(dim_date[Date],"mmmm")

monthnum = FORMAT(MONTH(dim_date[Date]),"00")

monthyear = CONCATENATE(dim_date[monthnum],dim_date[year])
```

12. If executed correctly you should be able to build a Table visual with the new table and fields you created. See the image below.



13. Now, navigate to the “Model View”
14. We want to connect the dim_ tables to our fact_ table to create a star schema. Connect dim_date to fct_stocks using the date field, and connect dim_company to fct_stocks using symbol and name. Your view should look like the image below



Session 8: Core DAX

Each model calculation type, calculated table, calculated column, or measure is defined by its name, followed by the equals symbol (=), which is then followed by a DAX formula. Use the following template to create a model calculation:

```
<Calculation name> = <DAX formula>
```

For example, the definition of the dim_date calculated table that duplicates the Date table data is:

```
dim_date = CALENDAR(min(fct_stocks[date]),max(fct_stocks[date]))
```

A DAX formula is essentially a set of instructions that yields a specific outcome. This outcome can take the form of either a complete table or a single, discrete value. It's important to note that, in calculated tables, the formula must result in an entire table, while calculated columns and measures should produce a solitary scalar value.

These formulas are constructed by employing a variety of components, including:

- DAX functions
- DAX operators
- References to model objects
- Whitespace for formatting and clarity

DAX functions, much like in Microsoft Excel, play a fundamental role in DAX, which is essentially a functional language. This means that DAX formulas are built upon the foundation of functions to achieve specific objectives. Usually, DAX functions come with parameters, enabling you to supply variables as needed. As a result, DAX formulas can involve multiple function calls, often with functions nested within one another.

Here is the full listing of DAX functions: https://learn.microsoft.com/en-us/dax/dax-function-reference/

In a DAX formula, it's crucial to recognize that function names are always accompanied by parentheses, inside of which you provide the necessary variables. This structured approach enables you to perform a wide range of operations and calculations in your models.

DAX operators are another key component of formulas. They serve as tools for conducting various tasks, including arithmetic computations, value comparisons, manipulation of text, and the assessment of conditions.

Formulas can only refer to three types of model objects: tables, columns, or measures. A formula can't refer to a hierarchy or a hierarchy level. (Recall that a hierarchy level is based on a column, so your formula can refer to a hierarchy level's column.)

When you reference a table in a formula, officially, the table name is enclosed within single quotation marks.

Single quotation marks can be omitted when both of the following conditions are true:

1. The table name does not include embedded spaces.
2. The table name isn't a reserved word that's used by DAX. All DAX function names and operators are reserved words. Date is a DAX function name, which explains why, when you are referencing a table named Date, that you must enclose it within single quotation marks.

```
Stocks = SP 500
```

When you reference a column in a formula, the column name must be enclosed within square brackets. Optionally, it can be preceded by its table name. For example, the following measure definition refers to the Open column.

```
Close_c = SUM(fct_stocks[close])
```

```
Open_c = SUM(fct_stocks[open])
```

When you reference a measure in a formula, like column name references, the measure name must be enclosed within square brackets. For example, the following measure definition refers to the Revenue and Cost measures.

```
Daily Change = [Open_c] - [Close_c]
```

Whitespace encompasses characters that facilitate the formatting of your formulas in a manner that's both swift and easy to comprehend. These whitespace characters consist of:

- Spaces
- Tabs
- Carriage returns

It's important to note that whitespace remains an optional feature and has no impact on the logic or performance of your formula. We strongly advise you to adopt a consistent formatting style and consider the following suggestions:

- Insert spaces between operators for clarity.
- Utilize tabs to neatly indent nested function calls.

- Employ carriage returns to segregate function arguments, especially when they extend beyond a single line. This formatting approach simplifies troubleshooting, particularly in situations where a parenthesis might be missing.

In general, it's advisable to be somewhat generous with the use of whitespace, erring on the side of more rather than less, as it enhances the readability and maintainability of your formulas.

Data Types

Data model columns are associated with specific data types to ensure the consistency of values within those columns. The data types for columns are primarily defined in Power Query, except in the case of calculated columns where they are inferred from the formula. Measure data types, like calculated column data types, are also inferred from the formula.

It's essential to distinguish between model data types and DAX data types, even though they share a direct relationship. The table below outlines the various model data types and their corresponding DAX data types, along with a description of each type and the supported value ranges.

Whole number : Corresponds to a 64-bit integer in DAX. Values range from -263 to 263-1.

Decimal number : Translates to a 64-bit real in DAX. This type accommodates negative values ranging from -1.79 x 10^308 to -2.23 x 10^-308, zero (0), and positive values from 2.23 x 10^-308 to 1.79 x 10^308, limited to 17 decimal digits.

Boolean : Maps to the Boolean data type in DAX, representing either TRUE or FALSE.

Text : Corresponds to the String data type in DAX, representing a Unicode character string.

Date: Translates to the Date/time data type in DAX, with valid dates starting from March 1, 1900.

Currency: Corresponds to the Currency data type in DAX. This type covers a range from -9.22 x 10^14 to 9.22 x 10^14 and is limited to four decimal digits of fixed precision.

N/A : Represents the equivalent of a database (SQL) NULL and is referred to as BLANK in DAX.

BLANK Data Type:

The BLANK data type is a unique concept in DAX. It's used to denote both database NULL and empty cells in Excel, but it should not be confused with zero. Instead, think of it as the absence of a value.

Two DAX functions are closely related to the BLANK data type: the BLANK DAX function, which returns BLANK, and the ISBLANK DAX function, which assesses whether an expression evaluates to BLANK.

DAX Operators

Your DAX formulas utilize operators to create expressions that perform various tasks, such as arithmetic calculations, value comparisons, string manipulations, and condition testing.

Arithmetic Operators:

You can use arithmetic operators like addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (^) to perform mathematical operations.

Comparison Operators:

Comparison operators allow you to compare values and return either TRUE or FALSE. They include operators like equal to (=), strict equal to (==), greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=), and not equal to (<>).

Text Concatenation Operator:

To combine two text values and create a single continuous text value, use the ampersand (&) character for text concatenation.

Logical Operators:

Logical operators are used to combine expressions that produce a single result. You can employ logical operators like AND (&&), OR (||), IN, and NOT for various logical conditions.

Operator Precedence:

In DAX, operator precedence determines the order in which operations are evaluated. This order is defined as follows:

- Exponentiation (^)
- Sign (-, as in -1)
- Multiplication and division (* and /)
- NOT
- Addition and subtraction (+ and -)
- Concatenation of text (&)
- Comparison operators (=, ==, <, >, <=, >=, <>)

If operators share the same precedence, they are evaluated from left to right. It's worth noting that the operator precedence in DAX is largely consistent with that in Excel. If you need to override the evaluation order, you can use parentheses to group operations.

DAX Variables

You can declare DAX variables in your formula expressions. When you declare at least one variable, a RETURN clause is used to define the expression, which then refers to the variables.

We recommend that you use variables because they offer several benefits:

- Improving the readability and maintenance of your formulas.
- Improving performance because variables are evaluated once and only when or if they're needed.
- Allowing (at design time) straightforward testing of a complex formula by returning the variable of interest.

```
Close YoY % =

VAR CloseYearPrior =

    CALCULATE(

        [Close_c],

        SAMEPERIODLASTYEAR(dim_date[Date])

    )

RETURN

    DIVIDE(

        [Close_c] - CloseYearPrior,

        CloseYearPrior

    )
```

Student Do: Core DAX

csv: all files in stocks directory
pbix: d2_s8.pbix

In this exercise we will use what we have learned so far to create Columns and Measures using DAX.

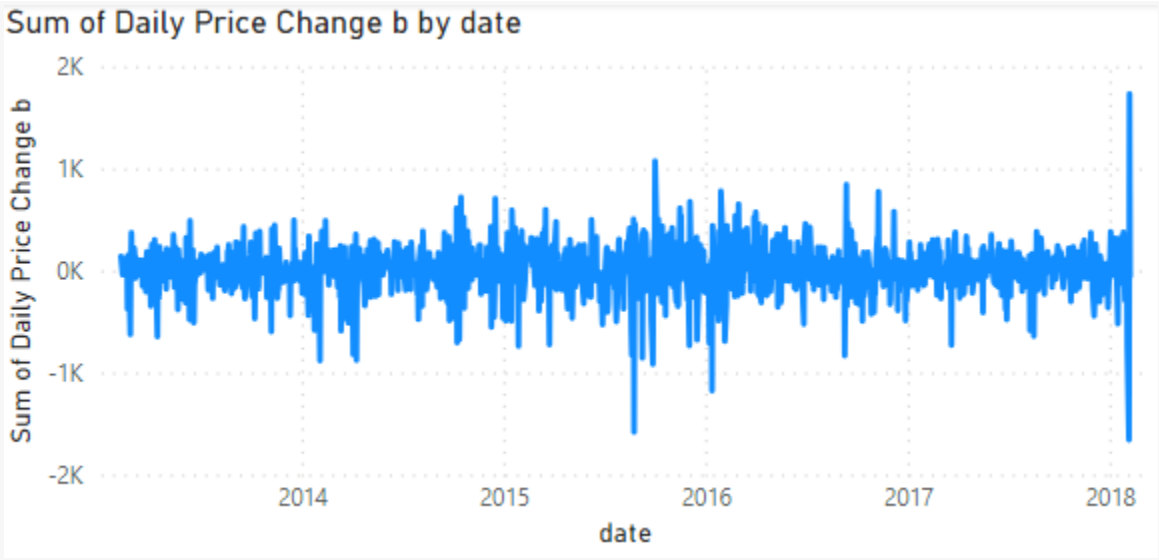
DAX Columns

You can write a DAX formula to add a calculated column to any table in your model. A calculated column formula must return a scalar or single value.

Calculated columns in import models have a cost: They increase the model storage size and they can prolong the data refresh time. The reason is because calculated columns recalculate when they have formula dependencies to refreshed tables.

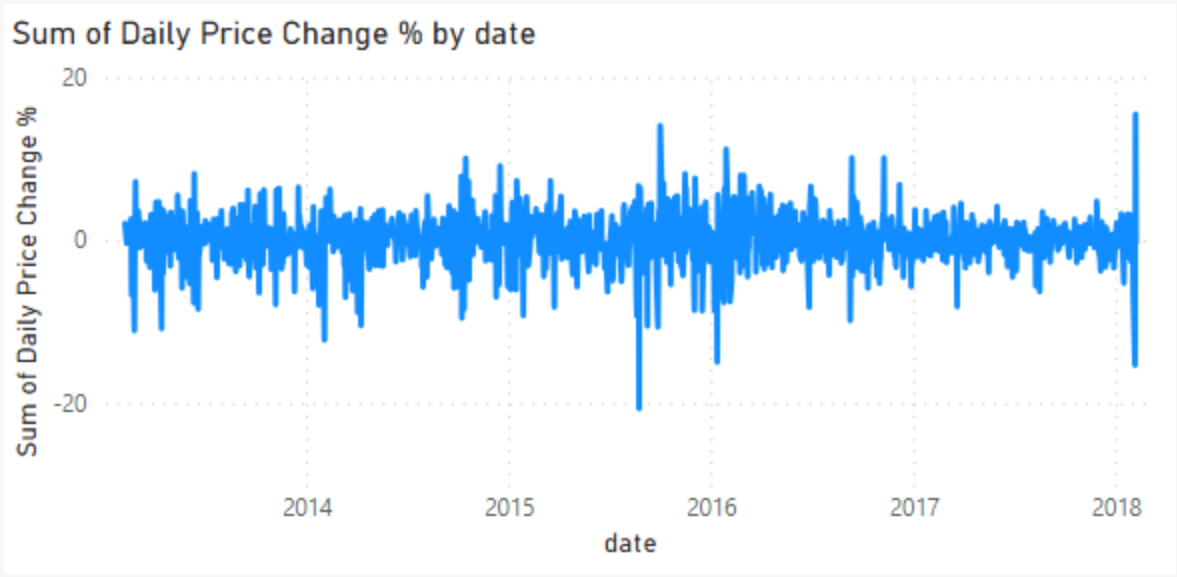
1. Daily Price Change:

- Code: `Daily Price Change = 'fct_stocks'[Close] - 'fct_stocks'[Open]`
- Explanation: This column calculates the difference between the closing price ('Close') and the opening price ('Open') for each day, representing the daily price change.



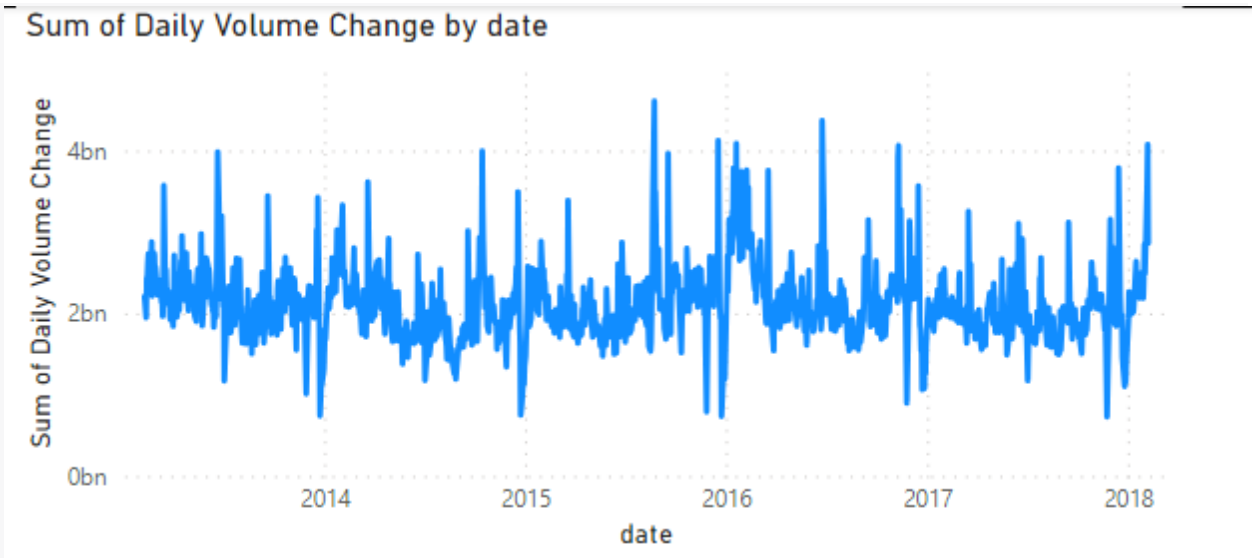
2. Daily Price Change Percentage:

- Code: `Daily Price Change % = DIVIDE('fct_stocks'[Close] - 'fct_stocks'[Open], 'fct_stocks'[Open])`
- Explanation: This column computes the daily price change as a percentage by taking the difference between the closing and opening prices and dividing it by the opening price.



3. Daily Volume Change:

- Code: `'fct_stocks'[Volume] - CALCULATE(MAX('fct_stocks'[Volume]), PREVIOUSMONTH('fct_stocks'[Date]))`
- Explanation: This column calculates the change in daily trading volume by subtracting the previous month's volume from the current day's volume. If there's no previous day's month's, it defaults to 0.



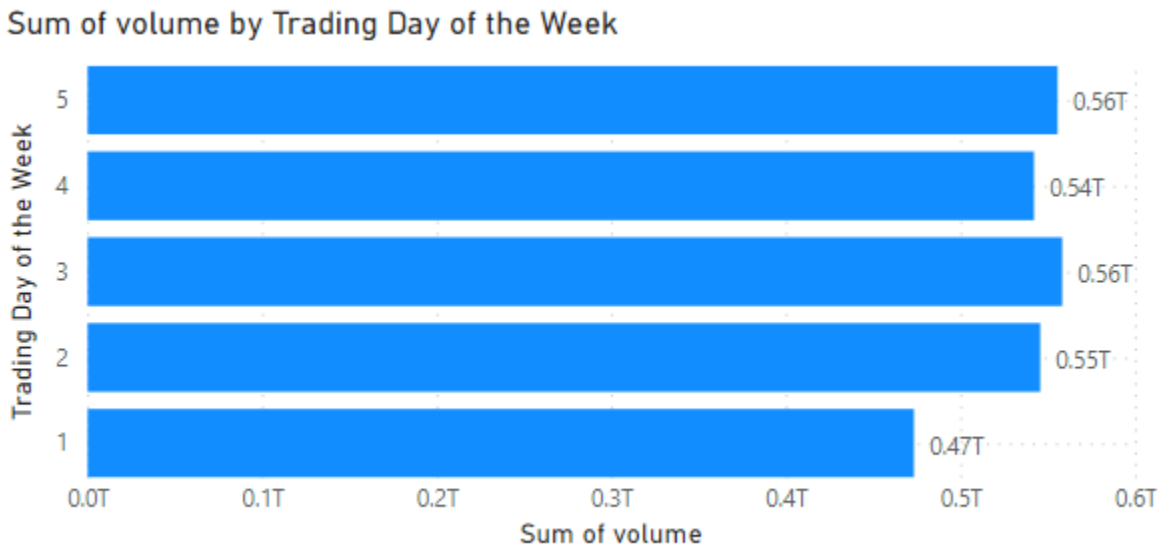
4. Gain/Loss Indicator:

- Code: `Gain/Loss Indicator = IF('fct_stocks'[Close] >= 'fct_stocks'[Open], "Gain", "Loss")`

- Explanation: This column categorizes daily price movement as either a "Gain" or a "Loss" based on whether the closing price is greater than or equal to the opening price.

5. Trading Day of the Week:

- Code: `Trading Day of the Week = WEEKDAY('fct_stocks'[Date], 2)`
- Explanation: This column extracts the day of the week (1 to 7, Monday to Sunday) from the date, allowing you to analyze stock performance based on the day of the week.



DAX Measures

1. Average Daily Volume:

- Code: `Average Daily Volume = AVERAGE('fct_stocks'[Volume])`
- Explanation: This measure calculates the average daily trading volume.

2. Total Gain Days:

- Code: `Total Gain Days = COUNTROWS(FILTER('fct_stocks', 'fct_stocks'[Close] >= 'fct_stocks'[Open]))`
- Explanation: This measure counts the total number of days with a closing price higher than the opening price.

3. Total Volume Traded on Gain Days:

- Code: `Total Volume on Gain Days = SUMX(FILTER('fct_stocks', 'fct_stocks'[Close] >= 'fct_stocks'[Open]), 'fct_stocks'[Volume])`

- Explanation: This measure calculates the total volume traded on days with gains.

4. Largest Single-Day Gain:

- Code: `Largest Single-Day Gain = MAX(fct_stocks[Daily Price Change b])`
- Explanation: This measure finds the largest single-day gain.

5. 30-Day Rolling Average Volume:

- Code: `30-Day Rolling Avg Volume = AVERAGEX( TOPN(30, 'fct_stocks', 'fct_stocks'[Date], DESC), 'fct_stocks'[Volume] )`
- Explanation: This measure computes the 30-day rolling average trading volume.

6. Largest Intraday Price Swing:

- Code: `Largest Intraday Price Swing = MAXX( 'fct_stocks', 'fct_stocks'[High] - 'fct_stocks'[Low] )`
- Explanation: This measure calculates the largest intraday price swing.

Session 8: Advanced DAX

Iterator Functions

Data Analysis Expressions (DAX) provide a set of functions called iterator functions. Iterator functions are designed to go through each row of a specified table and evaluate a given expression for each row. They offer you the ability to precisely control how your model calculations summarize data.

You're probably already familiar with single-column summarization functions like SUM, COUNT, MIN, and MAX, which work on individual columns. Each of these functions has a counterpart iterator function that has an "X" suffix, such as SUMX, COUNTX, MINX, and MAXX. These iterator functions allow you to perform more advanced calculations.

In the case of iterator functions, you need to provide a table and an expression. The table can be a reference to a model table or an expression that generates a table. The expression should produce a single value or scalar. This combination of table and expression allows you to work with detailed data while summarizing it in a way that meets your specific needs.

Below are a list of the most common DAX iterator functions:

SUMX: Calculates the sum of a specified expression for each row in a table or a table expression.

COUNTX: Counts the number of rows in a table or table expression where a specified condition is met.

AVERAGEX: Calculates the average of a specified expression for each row in a table or table expression.

MINX: Determines the minimum value of a specified expression for each row in a table or table expression.

MAXX: Finds the maximum value of a specified expression for each row in a table or table expression.

MEDIANX: Calculates the median value of a specified expression for each row in a table or table expression.

STDEVX.**P**: Computes the population standard deviation of a specified expression for each row in a table or table expression.

STDEVX.S: Calculates the sample standard deviation of a specified expression for each row in a table or table expression.

VARX.P: Calculates the population variance of a specified expression for each row in a table or table expression.

VARX.S: Computes the sample variance of a specified expression for each row in a table or table expression.

RANKX

The RANKX DAX iterator function is used to rank values in a column based on the values of another column. It assigns a rank to each row in a table or table expression, indicating its position in the sorted order of values in the specified column. Here's how RANKX works:

Syntax

```
RANKX(<table>, <expression>[, <value>[, <order>[, <ties>]]])
```

<table>: This is the table or table expression containing the rows you want to rank.

<expression>: This is the DAX expression that defines the values you want to rank. The expression is evaluated for each row in the table.

<value> (optional): You can specify a value that you want to find the rank for within the sorted values. If omitted, RANKX assigns ranks to all rows in the table based on the specified column.

<order> (optional): This argument defines the sort order. You can use 1 for ascending (default) or -1 for descending.

<ties> (optional): If there are ties (rows with the same values), you can specify how RANKX should handle them. You can use one of the following options: 0 (default) for average ranking, 1 for minimum ranking, and -1 for maximum ranking.

Explanation:

RANKX starts by evaluating the <expression> for each row in the specified <table>. It creates a list of these values and sorts them in either ascending or descending order based on the <order> parameter. Then, it assigns a rank to each row, indicating its position in the sorted list of values.

The optional <value> parameter allows you to find the rank of a specific value within the sorted list.

The optional <ties> parameter determines how RANKX handles ties, which occur when multiple rows have the same value. You can choose to assign the average rank, minimum rank, or maximum rank to tied values.

The CALCULATE Function and Filter Context

You can use the CALCULATE DAX function to modify filter context in your formulas. The syntax for the CALCULATE function is as follows:

Syntax

```
CALCULATE(<expression>, [[<filter1>], <filter2>]...)
```

The function requires passing in an expression that returns a scalar value and as many filters as you need. The expression can be a measure (which is a named expression) or any expression that can be evaluated in filter context.

Filters can be Boolean expressions or table expressions. It's also possible to pass in filter modification functions that provide additional control when you're modifying filter context.

When you have multiple filters, they're evaluated by using the AND logical operator, which means that all conditions must be TRUE at the same time.

Remove filters

Use the REMOVEFILTERS DAX function as a CALCULATE filter expression to remove filters from filter context. It can remove filters from one or more columns or from all columns of a single table.

Preserve filters

You can use the KEEPFILTERS DAX function as a filter expression in the CALCULATE function to preserve filters.

Three other functions that you can use to test filter state are:

ISFILTERED - Returns TRUE when a passed-in column reference is directly filtered.

ISCROSSFILTERED - Returns TRUE when a passed-in column reference is indirectly filtered. A column is cross-filtered when a filter that is applied to another column in the same table, or in a related table, affects the reference column by filtering it.

ISINSCOPE - Returns TRUE when a passed-in column reference is the level in a hierarchy of levels.

Student Do: Advanced DAX

csv: all files in stocks directory
pbix: d2_s9_student.pbix

Let's start by working with aggregation iterator DAX functions. Try to recreate each MEASURE using the S&P 500 stock data.

1. **SUMX - Calculating the total daily trade volume for each stock ticker:**

```
Total Volume = SUMX(fct_stocks, fct_stocks[Volume])
```

This calculated column sums the daily trading volumes for each stock ticker in the 'fct_stocks' table.

2. **COUNTX - Counting the number of days when the closing price is above a certain threshold (e.g., $100):**

```
Days Above $100 = COUNTX(fct_stocks, IF(fct_stocks[Close] > 100, 1, 0))
```

This calculated column counts the number of days when the closing price of a stock is above $100.

3. **AVERAGEX - Calculating the average daily change for each stock ticker:**

```
Avg Daily Change = AVERAGEX(fct_stocks, fct_stocks[Close] - fct_stocks[open])
```

This column calculates the average daily return for each stock ticker based on the closing prices.

4. **MINX - Finding the minimum daily low price for each stock ticker:**

```
Min Low = MINX(fct_stocks, fct_stocks[Low])
```

This column determines the minimum low price for each stock ticker.

5. **MAXX - Identifying the maximum daily high price for each stock ticker:**

```
Max High = MAXX(fct_stocks, fct_stocks[High])
```

This calculated column finds the maximum high price for each stock ticker.

6. **MEDIANX - Calculating the median daily volume for each stock ticker:**

```
Median Volume = MEDIANX(fct_stocks, fct_stocks[Volume])
```

This column computes the median daily trading volume for each stock ticker.

7. **STDEVX.P - Calculating the population standard deviation of daily closing prices for each stock ticker:**

```
Stdev Population Close = STDEVX.P(fct_stocks, fct_stocks[Close])
```

This column computes the population standard deviation of daily closing prices for each stock ticker.

8. **STDEVX.S - Calculating the sample standard deviation of daily high prices for each stock ticker:**

```
Stdev Sample High = STDEVX.S(fct_stocks, fct_stocks[High])
```

This calculated column calculates the sample standard deviation of daily high prices for each stock ticker.

9. **VARX.P - Determining the population variance of daily low prices for each stock ticker:**

```
Variance Population Low = VARX.P(fct_stocks, fct_stocks[Low])
```

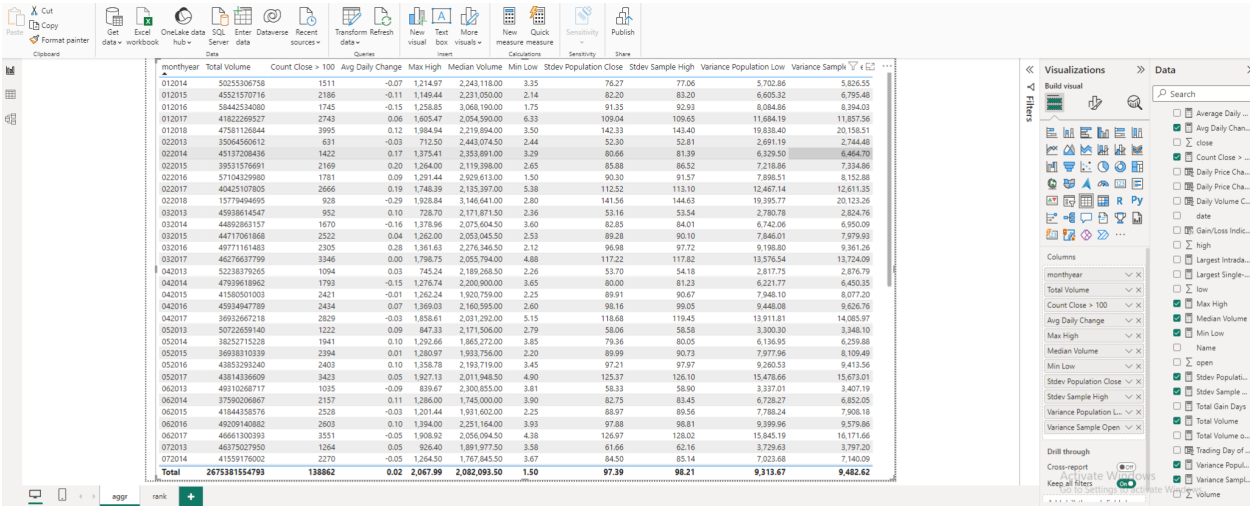
This column computes the population variance of daily low prices for each stock ticker.

10. **VARX.S - Computing the sample variance of daily open prices for each stock ticker:**

```
Variance Sample Open = VARX.S(fct_stocks, fct_stocks[Open])
```

This calculated column calculates the sample variance of daily open prices for each stock ticker.

After adding all of these DAX iterator measures you should combine them all into a table using the 'monthyear' of the dim_date query as the main aggregate category. See image below.



monthyear	Total Volume	Count Close > 100	Avg Daily Change	Max High	Median Volume	Min Low	Stdev Population Close	Stdev Sample High	Variance Population Low	Variance Sample
012014	50255306758	1511	-0.07	1,214.97	2,243,118.00	3.35	76.27	77.06	5,702.86	5,826.55
012015	45521570716	2186	-0.11	1,149.44	2,231,050.00	2.14	82.20	83.20	6,605.32	6,795.48
012016	58442534080	1745	-0.15	1,258.85	3,068,190.00	1.75	91.35	92.93	8,084.86	8,394.03
012017	41822269527	2743	0.06	1,605.47	2,054,590.00	6.33	109.04	109.65	11,684.19	11,857.56
012018	47581126844	3995	0.12	1,984.94	2,219,894.00	3.50	142.33	143.40	19,838.40	20,158.51
022013	35064560612	631	-0.03	712.50	2,443,074.50	2.44	52.30	52.81	2,691.19	2,744.48
022014	45137208436	1422	0.17	1,375.41	2,353,891.00	3.29	80.66	81.39	6,329.50	6,464.70
022015	39531576691	2169	0.20	1,264.00	2,119,398.00	2.65	85.88	86.52	7,218.86	7,334.86
022016	57104329980	1781	0.09	1,291.44	2,929,613.00	1.50	90.30	91.57	7,898.51	8,152.88
022017	40425107805	2666	0.19	1,748.39	2,135,397.00	5.38	112.52	113.10	12,467.14	12,611.35
022018	15779494695	928	-0.29	1,928.84	3,146,641.00	2.80	141.56	144.63	19,395.77	20,123.26
032013	45938614547	952	0.10	728.70	2,171,871.50	2.36	53.16	53.54	2,780.78	2,824.76
032014	44892863157	1670	-0.16	1,378.96	2,075,604.50	3.60	82.85	84.01	6,742.06	6,950.09
032015	44717061868	2522	0.04	1,262.00	2,053,045.50	2.53	89.28	90.10	7,846.01	7,979.93
032016	49771161483	2305	0.28	1,361.63	2,276,346.50	2.12	96.98	97.72	9,198.80	9,361.26
032017	46276637799	3346	0.00	1,798.75	2,055,794.00	4.88	117.22	117.82	13,576.54	13,724.09
042013	52238379265	1094	0.03	745.24	2,189,268.50	2.26	53.70	54.18	2,817.75	2,876.79
042014	47939618962	1793	-0.15	1,276.74	2,200,900.00	3.65	80.00	81.23	6,221.77	6,450.35
042015	41580501003	2421	-0.01	1,262.24	1,920,759.00	2.25	89.91	90.67	7,948.10	8,077.20
042016	45934947789	2434	0.07	1,369.03	2,160,595.00	2.60	98.16	99.05	9,448.08	9,626.76
042017	36932667218	2829	-0.03	1,858.61	2,031,292.00	5.15	118.68	119.45	13,911.81	14,085.97
052013	50722659140	1222	0.09	847.33	2,171,506.00	2.79	58.06	58.58	3,300.30	3,348.10
052014	38252715228	1941	0.10	1,292.66	1,865,272.00	3.85	79.36	80.05	6,136.95	6,259.88
052015	36938310339	2394	0.01	1,280.97	1,933,756.00	2.20	89.99	90.73	7,977.96	8,109.49
052016	43853292240	2403	0.10	1,358.78	2,193,719.00	3.45	97.21	97.97	9,260.53	9,413.56
052017	43814336609	3423	0.05	1,927.13	2,011,948.50	4.90	125.37	126.10	15,478.66	15,673.01
062013	49310268717	1035	-0.09	839.67	2,300,855.00	3.81	58.33	58.90	3,337.01	3,407.19
062014	37590206867	2157	0.11	1,286.00	1,745,000.00	3.90	82.75	83.45	6,728.27	6,852.05
062015	41844358576	2528	-0.03	1,201.44	1,931,602.00	2.25	88.97	89.56	7,788.24	7,908.18
062016	49209140882	2603	0.10	1,394.00	2,251,164.00	3.93	97.88	98.81	9,399.96	9,579.86
062017	46661300393	3551	-0.05	1,908.92	2,056,094.50	4.38	126.97	128.02	15,845.19	16,171.66
072013	46375027950	1264	0.05	926.40	1,891,977.50	3.58	61.66	62.16	3,729.63	3,797.20
072014	41559176002	2270	-0.05	1,264.50	1,767,845.50	3.67	84.50	85.14	7,023.68	7,140.09
Total	**2675381554793**	**138862**	**0.02**	**2,067.99**	**2,082,093.50**	**1.50**	**97.39**	**98.21**	**9,313.67**	**9,482.62**

Let's continue by creating some RANK measures, leveraging several of the aggregate iterator measures we made in the previous step.

Rank by Volume:

```
Stock Volume Rank =

RANKX(

    ALL('fct_stocks'[name]),
```

```
    fct_stocks[Total Volume]
)
```

Rank by Close:

```
Stock Close Rank =
RANKX(
    ALL('fct_stocks'[name]),
    fct_stocks[Total Close]
)
```

Rank by Avg Close:

```
Stock Avg Close Rank =
RANKX(
    ALL('fct_stocks'[name]),
    fct_stocks[Total Avg Close]
)
```

You should be able to combine all of these ranks into one table, aggregate by fct_stocks name or ticker. See below to confirm.

Name	Stock Volume Rank	Stock Close Rank	Stock Avg Close Rank
A	246	331	341
AAL	43	393	402
AAP	407	57	58
AAPL	2	89	90
ABBV	55	259	268
ABC	280	163	167
ABT	68	367	378
ACN	221	109	110
ADBE	202	135	136
ADI	231	257	266
ADM	157	369	379
ADP	276	146	148
ADS	474	13	13
ADSK	228	246	252
AEE	320	354	365
AEP	212	272	282
AES	91	494	502
AET	216	100	101
AFL	264	238	245
AGN	207	15	15
AIG	49	293	302
AIV	384	397	406
AIZ	481	184	188
AJG	432	334	345
AKAM	259	284	294
ALB	388	185	189
ALGN	437	156	159
ALK	337	255	263
ALL	238	231	237
ALLE	473	298	253
ALXN	305	46	46
AMAT	26	455	468
AMD	7	503	505
Total	**1**	**1**	**164**

FInally, let's see some examples of using the CALCULATE functions with advanced FILTERS. This may require Merging some of the attributes from dim_company into fct_stocks.

```
Total Volume FAANG =

CALCULATE(

   SUM('fct_stocks'[Volume]),

   FILTER('fct_stocks', 'fct_stocks'[Name] IN {"FB", "AAPL", "AMZN", "NFLX", "GOOGL"})

)

Avg Close Price InfoTech =

CALCULATE(

   AVERAGE('fct_stocks'[Close]),

   FILTER('fct_stocks', 'fct_stocks'[Sector] = "Information Technology")

)

Max Daily Return Energy =

CALCULATE(

    MAXX(

        FILTER('fct_stocks', 'fct_stocks'[Sector] = "Energy"),

        'fct_stocks'[Close] / 'fct_stocks'[Open] - 1

    )

)

Total Volume Recent IPO =

CALCULATE(

   SUM('fct_stocks'[Volume]),

   FILTER('fct_stocks', YEAR('fct_stocks'[date_added]) >= YEAR(TODAY()) - 5)

)
```

Session 10: Advanced Filtering

In this session, we'll begin by revisiting the fundamentals of filtering and slicing data in Power BI.

Filters and Slicers Overview:
Filters and slicers are the cornerstones of interactivity in Power BI. These tools are essential for refining and controlling the data presented in your visuals. Filters allow you to narrow down data based on specific criteria, while slicers provide user-friendly controls for end-users to interact with the data. Understanding the distinction between page-level filters, visual-level filters, and slicers is crucial.

Applying Basic Filters:
At its core, Power BI provides the ability to apply filters at different levels. **Page-level filters** affect all visuals on a given page. **Slicers**, which are user-friendly, customizable controls, allow end-users to dynamically filter data by making selections. We will explore how to create slicers for different data fields and enhance the user experience.

Interactive Filtering:
Filters and slicers in Power BI are interactive and flexible. They enable dynamic exploration of data. Participants will learn how to make selections, drill down into hierarchical data structures, and reset to default views, providing them with the tools to gain deeper insights and tailor their reports to specific requirements.

II. Advanced Filtering/Slicer Techniques

Visual-Level Filters:
Visual-level filters are the next step in enhancing your reporting. These filters allow you to exert granular control over individual visuals on a report page. By applying filters at the visual level, you can focus on specific aspects of your data within a single visual while keeping other visuals untouched. We'll cover scenarios where visual-level filters are particularly valuable.

Drillthrough Filters:
Drillthrough is a powerful technique for detailed data exploration. We'll set up drillthrough pages and configure filters to enable this feature. This function empowers users to drill down into specific data points, revealing a deeper layer of information. This can be invaluable for analyzing hierarchies, examining outliers, or performing root-cause analysis.

III. Syncing Slicers

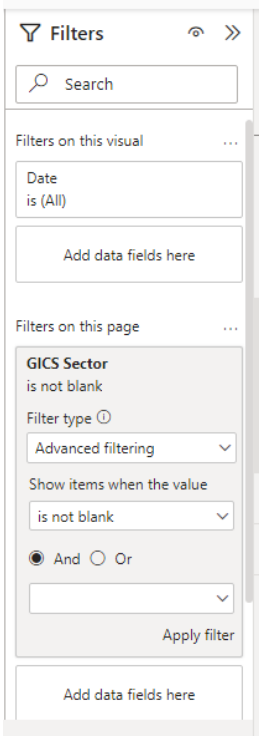
You can also sync two or more separate slicers. Syncing slicers is useful when working with composite models, as you might want to make the same selection across sources without relying on cross-source group relationships. To sync two or more separate slicers, you mark them as being part of a group.

Student Do: Advanced Filtering

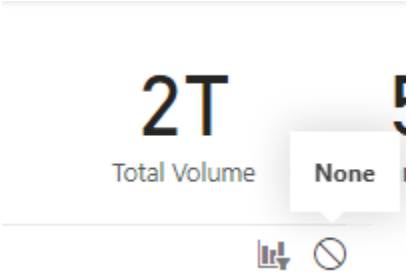
csv: all files in stocks directory
pbix: d2_s10_student.pbix & d2_s10_student_starter.pbix

In this exercise we will be implementing a number of filtering options in Power BI. It's important that you have some charts and graphs to use in order to confirm the filters are having the intended effect. You can opt to create your own visuals, or you can use the workbook provided as a starting point (d2_s10_student_starter.pbix).

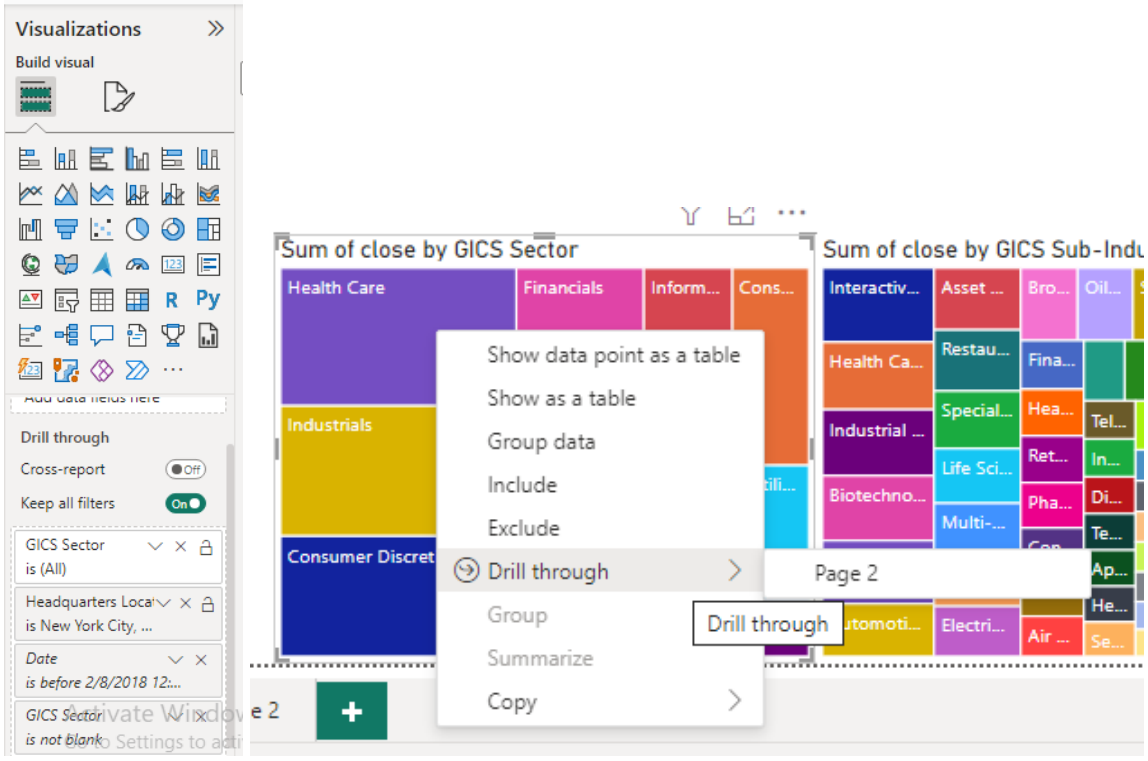
1. Page Filters
    a. Start by opening up the Filters Panel in the Report View.
    b. In the Filter Panel, drag GICS Sector from dim_company to the “Filters on this page” section of the Filter Panel.
    c. Under GICS Sector in the Filter Panel, select Advanced filtering as the value in the Filter type menu.
    d. Select “is not blank” as the value in the “Show items when the value” menu
    e. This will filter OUT all of the records where GICS Sector is blank from this entire page.
    f. Follow the steps above and add 2-3 more page filters.



2. Slicers
    a. Dates & Numbers
        i. Select the Slicer option from the Visualizations Panel and drag Date from dim_date to the Field shelf
        ii. Navigate to Format Visual sub-menu, Slicer Settings > Options and choose the Between value for Style.
        iii. Repeat this process and create several new Slicers with different Style Options.
    b. Categorical Data
        i. Select the Slicer option from the Visualizations Panel and drag Name from fct_stocks to the Field shelf
        ii. Navigate to Format Visual sub-menu, Slicer Settings > Options and choose the Dropdown option..
        iii. Use this new Slicer to select multiple stock tickers by holding down control and selecting multiple values.
3. Visual Level Filters
    a. Use the Treemaps in the provided workbook to filter your Report. Select different values in the Treemap to understand how your selections effect the entire report.
    b. Go to the Format tab on the Home Ribbon and select Edit Interactions.
    c. On on of the provided KPI charts, Choose the new “None” icon.
    d. As you adjust sliders notice how this chart remains unaffected. This is how you can create static visuals regardless of what Slicer values are selected.



4. Drill Through
    a. Create a new page in Power BI.
    b. On this new page, create a simple table with stock tickers and any number of columns/measures.
    c. In the Visualizations Pane, scroll down until you see a Drill Through section and then drag GICS Sector and Headquarters from dim_company to this section of the Visualizations Pane.
    d. Now navigate back to Page 1, select a value on the GICS or HQ Treemap and then right click the Visual. There should now be an option in the sub menu to Drill Through to Page 2.
    e. Repeat the process with multiple pages to create a user experience that allows users to drill from high level summary visuals to more detailed information on a separate Page.



5. Syncing Slicers
    a. On Page 2, create a new Slicer with Symbol from dim_company as the Field value.
    b. In the Home Ribbon, select the View tab and then press the Sync Sliders option. This should make another Panel in between Filters and Visualizations visible.
    c. Click on the Symbol slicer, and in the new Sync Slicers Pane enter the group name “ticker” for the Slicer.
    d. Navigate to Page 1 and repeat the steps above for the existing Name Slicer.
    e. Once complete, notice how the selection on one Page carries over to the next on the now connected Slicers.
    f. Repeat this with other Fields and Slicers to create a unified experience for your users in Power BI reports.