ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

- Power BI-ს გადმოწერა და პარამეტრების დაყენება
- Power BI-ს ინტერფეისი და სამუშაო სივრცე
- მონაცემთა შეერთების ტიპები
- Query Editor

Power BI Desktop ვერსიის ინსტალაცია

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

Power BI Desktop

Microsoft Power BI Desktop is built for the analyst. It combines state-of-the-art interactive visualizations, with industry-leading data query and modeling built-in. Create and publish your reports to Power BI. Power BI Desktop helps you empower others with timely critical insights, anytime, anywhere.

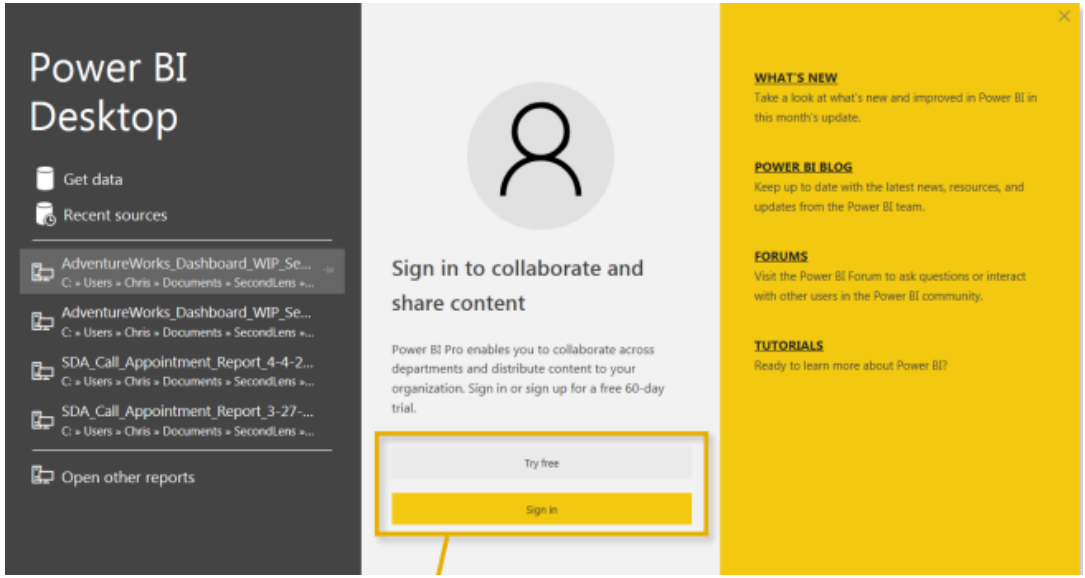
Important! Selecting a language below will dynamically change the complete page content to that language.

Select language English Download

Choose the download you want

File Name	Size
PBIDesktopSetup.exe	405.4 MB
PBIDesktopSetup_x64.exe	446.2 MB

Download Total size: 446.2 MB



მნიშვნელოვანია:

- Power BI Desktop-ზე წვდომისთვის არ გჭირდებათ შესვლა ან რეგისტრაცია Power BI Pro ანგარიშზე.
- შესვლა საჭიროა მხოლოდ Power BI სერვისის საშუალებით რეპორტის გაზიარებისთვის (app.powerbi.com)

რა არის Power BI?

- Power BI არის Microsoft-ის Business Intelligence-ის პროდუქტი, რომელიც მოიცავს როგორც დესკტოპის, ისე ვებ-ზე დაფუძნებულ სერვისის აპლიკაციას, მონაცემთა ჩატვირთვის, მოდელირებისა და ვიზუალიზაციისთვის.

რატომ Power BI?

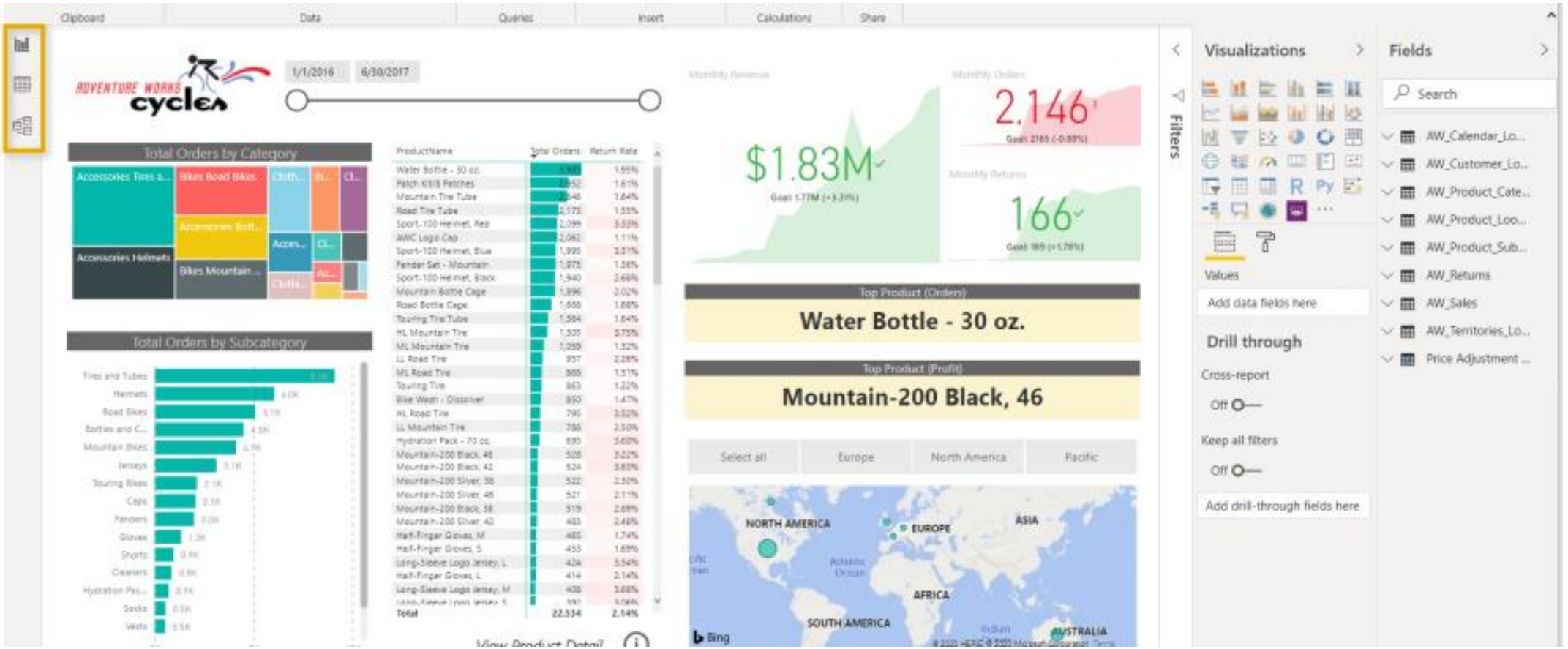
Power BI-ის საშუალებით შესაძლებელია :

- მონაცემთა მილიონობით მწკრივის დაკავშირება, გარდაქმნა და გააანალიზება
- რელაციური მოდელების შექმნა რამდენიმე წყაროდან მონაცემების დაკავშირებისთვის
- რთული გამოთვლები DAX-ის გამოყენებით
- მონაცემთა ვიზუალიზაცია ინტერაქტიული რეპორტებით და Dashboard-ებით

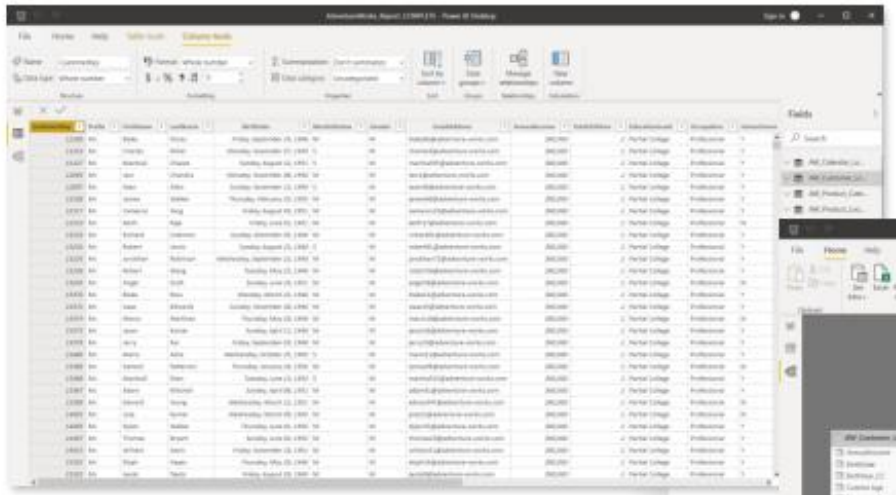
THE POWER BI ინტერფეისი

Three Core Views:

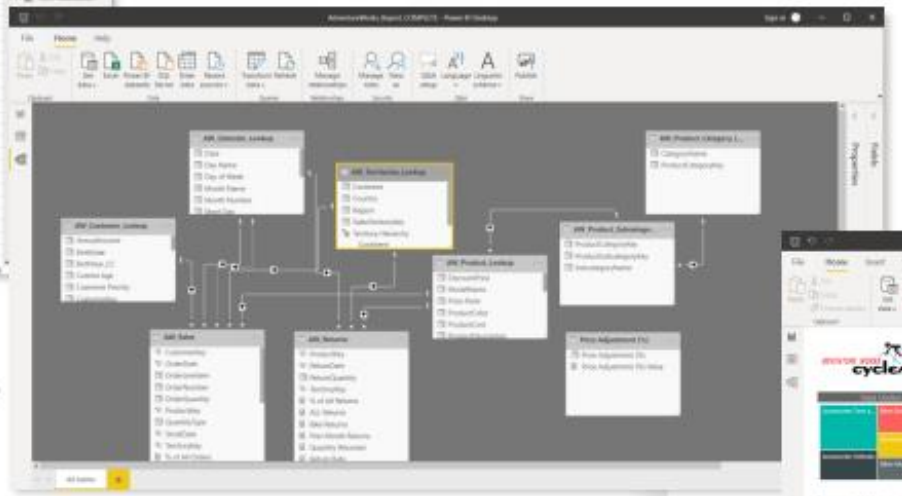
- ***Report***
- ***Data***
- ***Relationships***
  *(aka **Model**)*



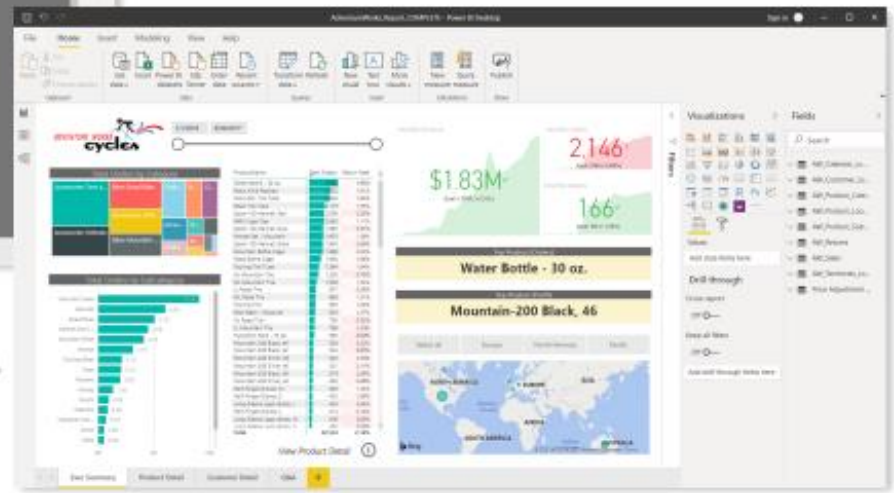
POWER BI-ის WORKFLOW



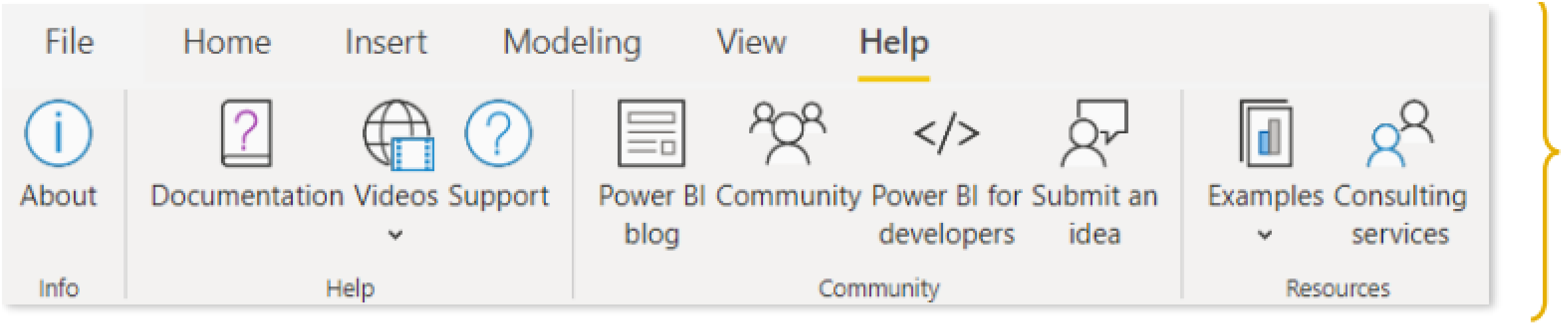
*Connect, shape and transform raw **data***



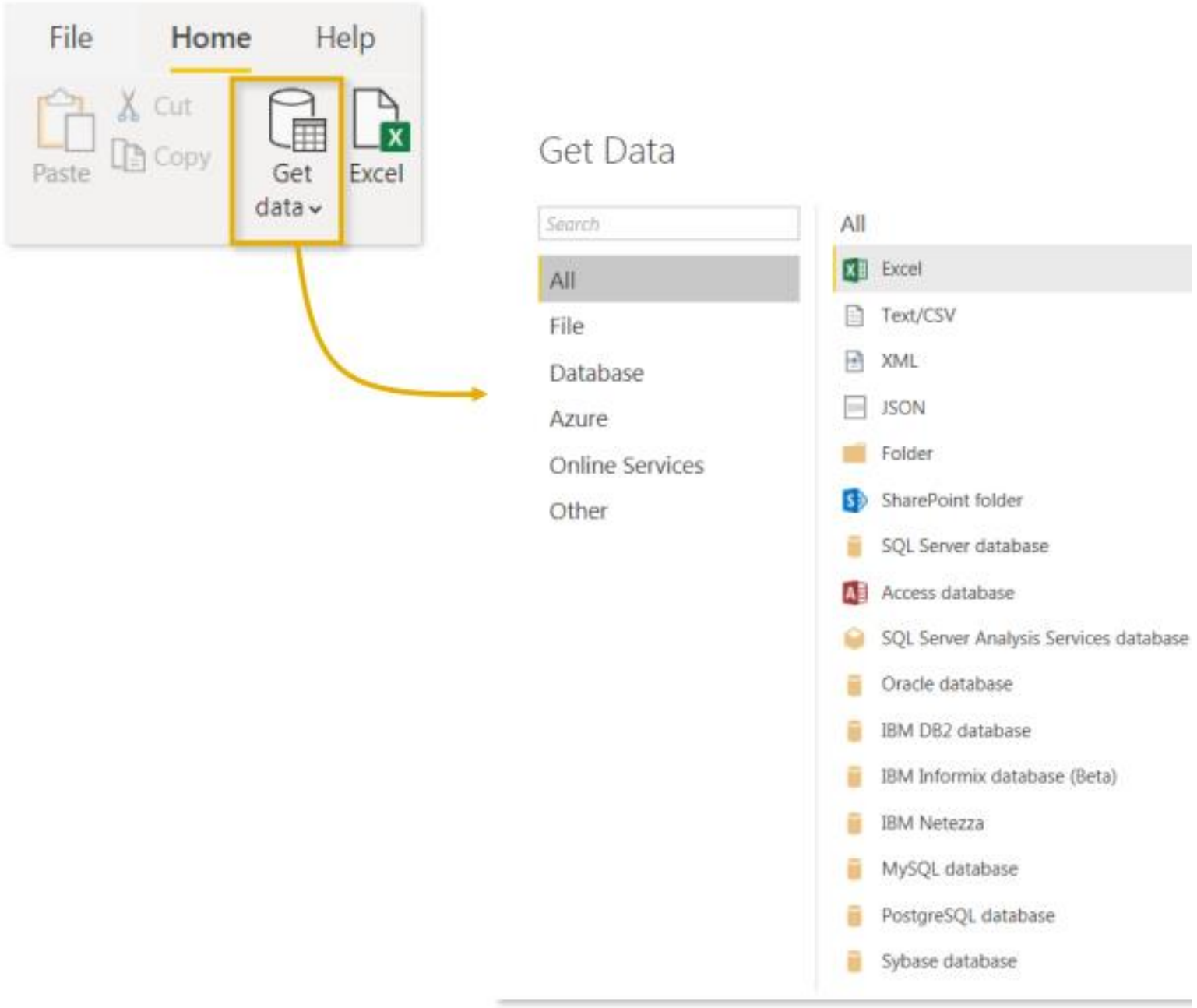
*Build table relationships to create a data **model***



*Design interactive **reports** to explore and visualize data*

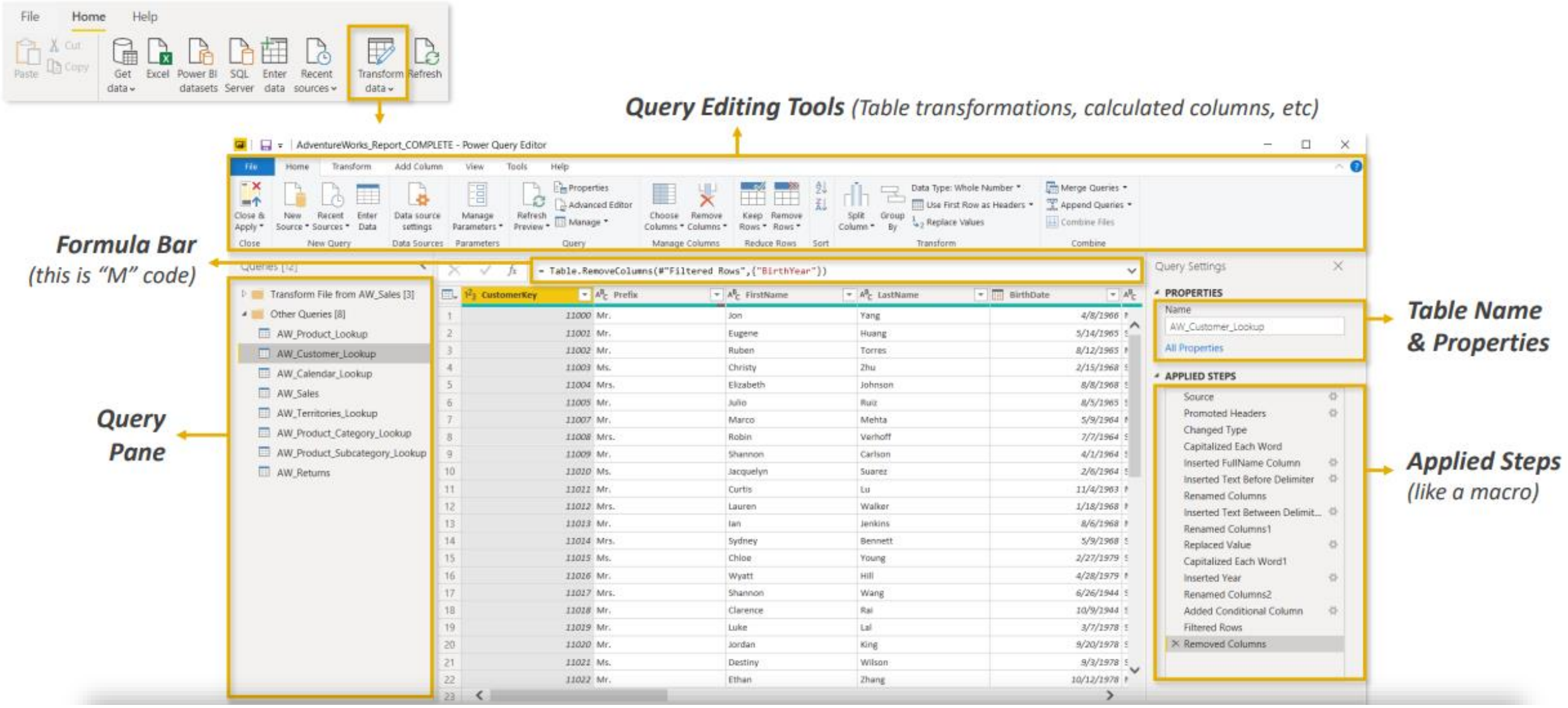


“Help”ტაბი მოიცავს დოკუმენტაციას, ტრენინგების ვიდეოებს, ნიმუშის ფაილებს და ლინკებს რომელიც დაგეხმარებათ გაიგოს ყველაფერი Power BI Desktop ვერსიის შესახებ



მონაცემთა შეერთების ტიპები

- Power BI-ს შეუძლია დაუკავშირდეს თითქმის ნებისმიერი ტიპის მონაცემთა წყაროს, მათ შორის :
  - მონაცემთა ბაზებს (SQL, Access, Oracle, IBM, Azure და ა.შ.)
  - ონლაინ სერვისებს (SharePoint, GitHub, Dynamics 365,Google Analytics, Salesforce და ა.შ.)
  - ფაილებს და ფოლდერებს (csv, ტექსტი, xls და ა.შ.)
  - სხვა (ვებ არხები, R სკრიპტები, Spark, Hadoop და ა.შ.)



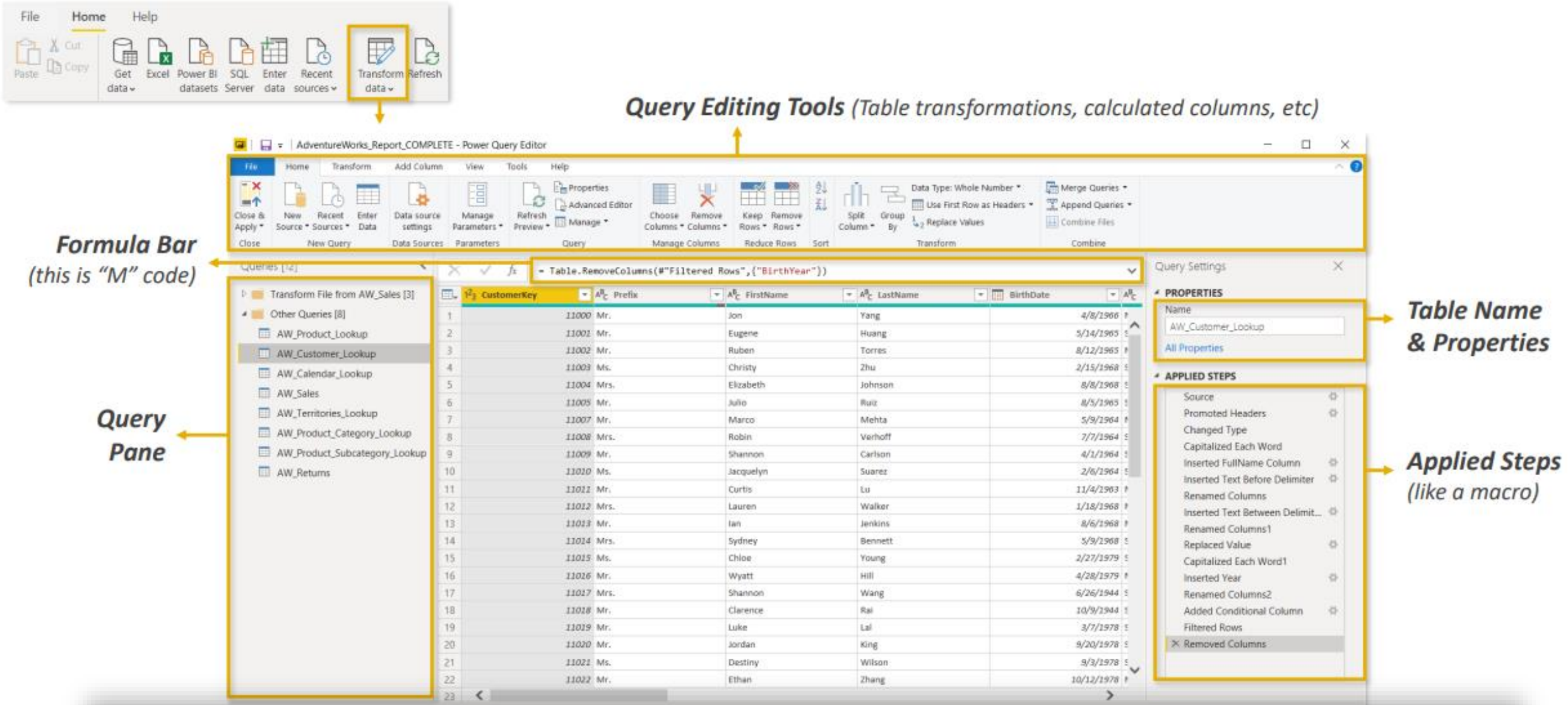
Query Editor

მადლობა ყურადღებისთვის

ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

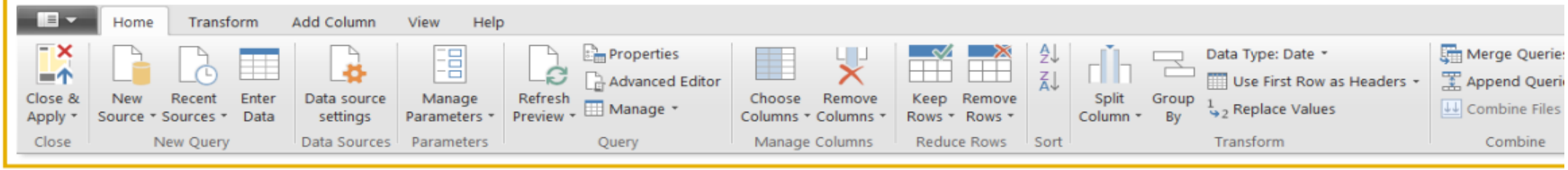
- მონაცემთა დაკავშირება და ფორმირება
- ცხრილის ძირითადი გარდაქმნები
- ტექსტურ და რიცხვით მონაცემებთან მუშაობა, თარიღის მონაცემებთან მუშაობა
- მოძრავი კალენდრის შექმნა, ინდექსისა და პირობითი სვეტის შექმნა,მონაცემთა დაჯგუფება და ჯამური მონაცემების კალკულაციები
- Pivot და Unpivot ბრძანება, Merge Queries ბრძანება



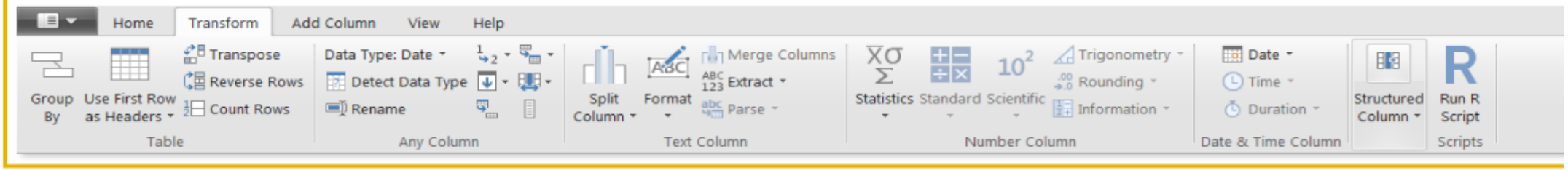
Query Editor

Query Editing Tools

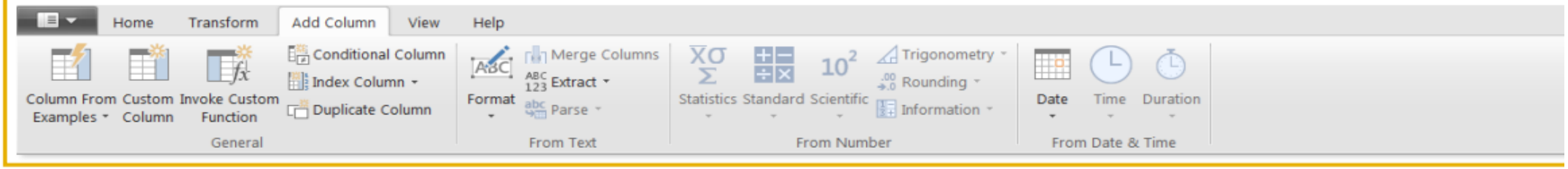
*The **HOME** tab includes **general settings** and **common table transformation tools***

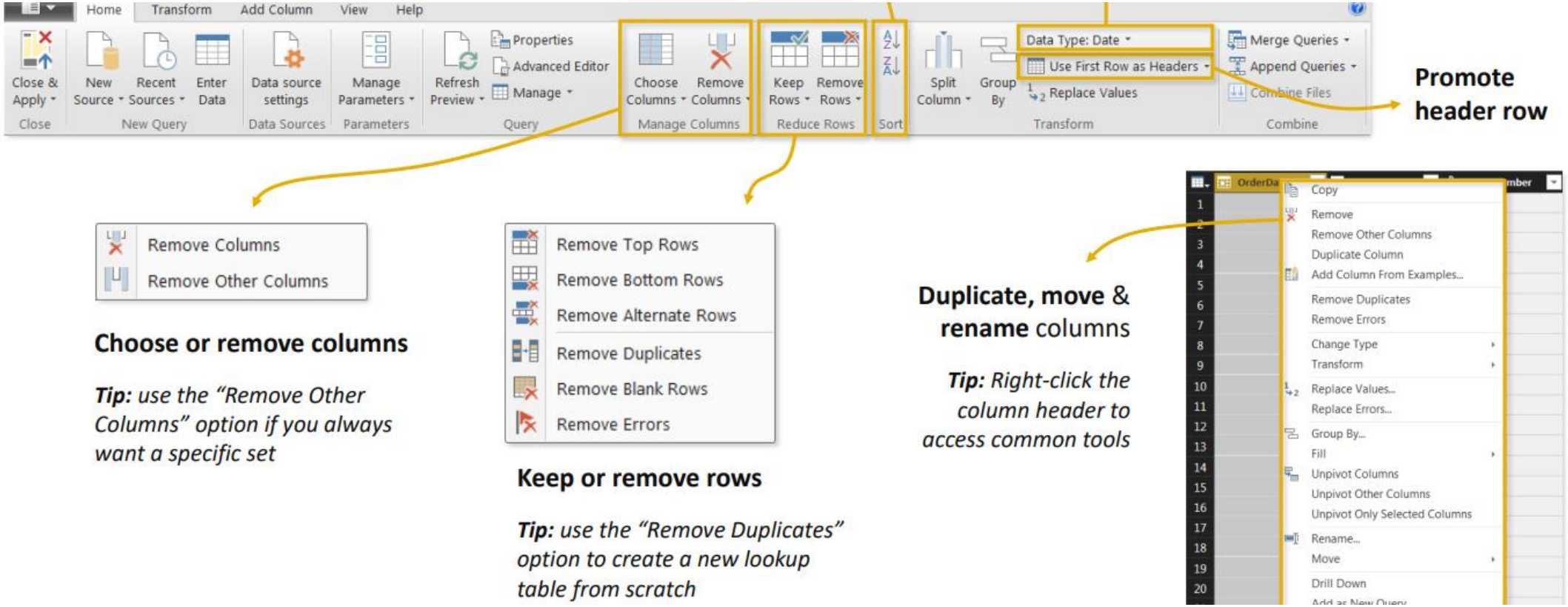


*The **TRANSFORM** tab includes tools to **modify existing columns** (splitting/grouping, transposing, extracting text, etc)*



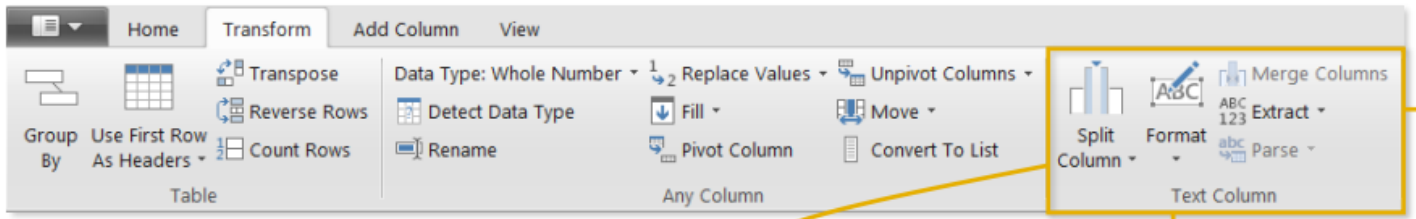
*The **ADD COLUMN** tools **create new columns** (based on conditional rules, text operations, calculations, dates, etc)*

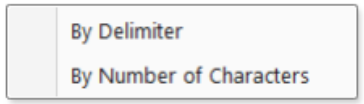




- **ძირითადი ტრანსფორმაციები**

TEXT-SPECIFIC TOOLS

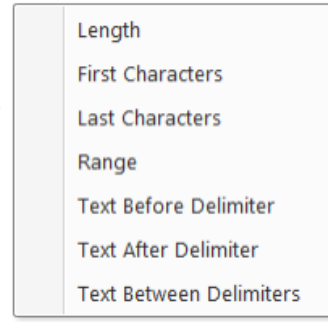




Split a text column based on either a specific delimiter or a number of characters

Format a text column to upper, lower or proper case, or add a prefix or suffix

Tip: *Use "Trim" to eliminate leading & trailing spaces, or "Clean" to remove non-printable characters*



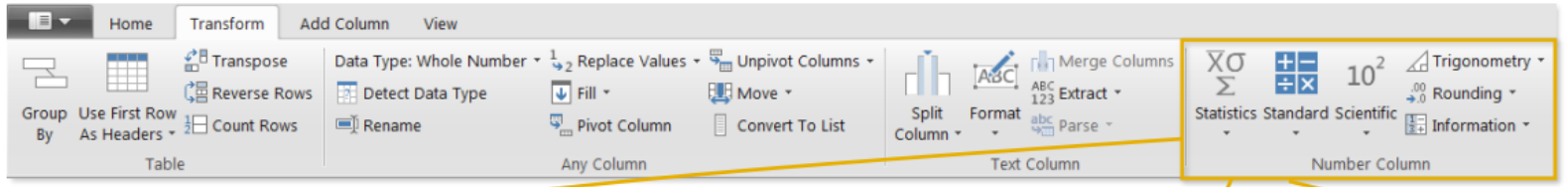
Extract characters fror column based on fixed first/last, ranges or del

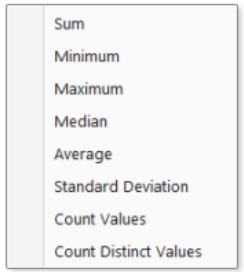
Tip*: Select two or more co* ***merge*** *(or* ***concatenate****) fi*

HEY THIS IS IMPORTANT!

You can access many of these tools in both the "**Transform**" and "**Add Column**" menus -- the difference is whether you want to ***add a new column*** or ***modify an existing one***

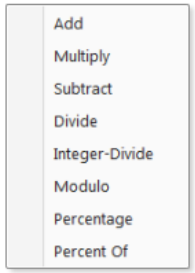
NUMBER-SPECIFIC TOOLS



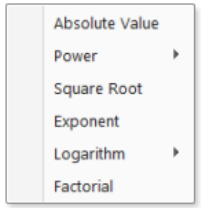


Statistics functions allow you to evaluate basic stats for the selected column (sum, min/max, average, count, countdistinct, etc)

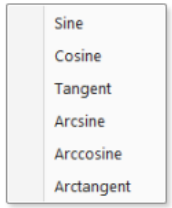
Note: *These tools return a SINGLE value,*



Standard

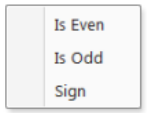


Scientific



Trigonometry

Standard, Scientific and **Trigonometry** tools allow you to apply standard operations (addition, multiplication, division, etc.) or more advanced calculations (power, logarithm, sine, tangent, etc) to each value in a column



Information tools allow you to define binary flags (*TRUE/FALSE or 1/0*) to mark each row in a column as even, odd, positive or negative

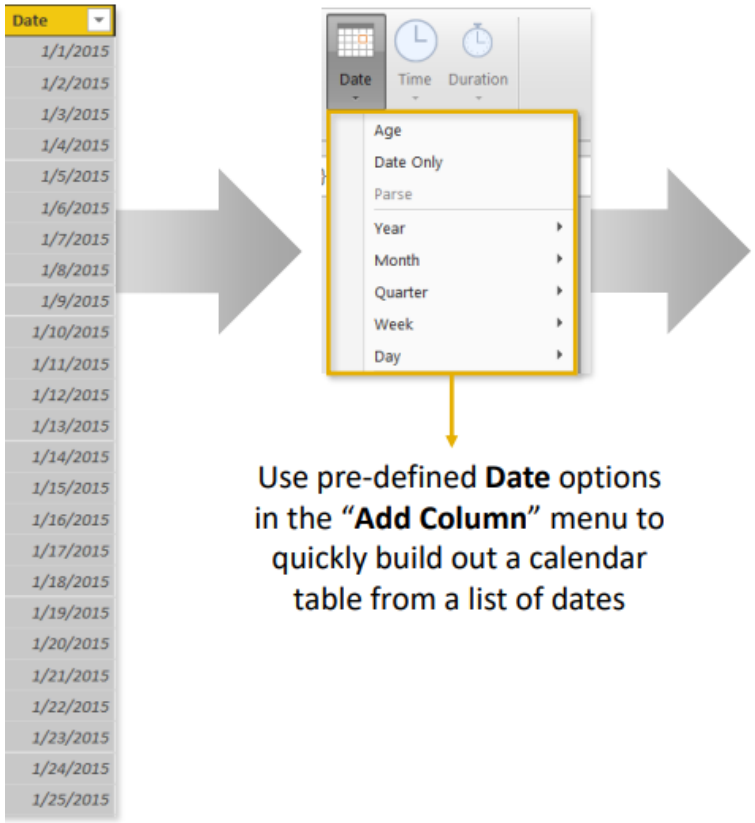
DATE-SPECIFIC TOOLS

Date & Time tools are relatively straight-forward, and include the following options:

- **Age**: Difference between the current time and the date in each row
- **Date Only:** Removes the time component of a date/time field
- **Year/Month/Quarter/Week/Day**: Extracts individual components from a date field (Time-specific options include Hour, Minute, Second, etc.)
- **Earliest/Latest:** Evaluates the earliest or latest date from a column as a single value (can only be accessed from the "Transform" menu)

Note: *You will almost always want to perform these operations from the "Add Column" menu to build out new fields, rather than transforming an individual date/time column*

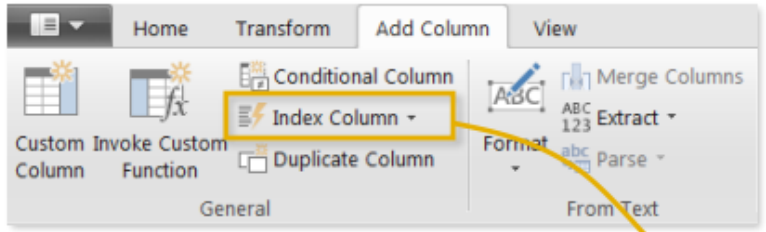
კალენდრის შექმნა



Use pre-defined **Date** options in the "**Add Column**" menu to quickly build out a calendar table from a list of dates

	Date	Day	Day of Week	Day Name	Start of Week	Month
1	1/1/2015	1	4	Thursday	12/28/2014	1
2	1/2/2015	2	5	Friday	12/28/2014	1
3	1/3/2015	3	6	Saturday	12/28/2014	1
4	1/4/2015	4	0	Sunday	1/4/2015	1
5	1/5/2015	5	1	Monday	1/4/2015	1
6	1/6/2015	6	2	Tuesday	1/4/2015	1
7	1/7/2015	7	3	Wednesday	1/4/2015	1
8	1/8/2015	8	4	Thursday	1/4/2015	1
9	1/9/2015	9	5	Friday	1/4/2015	1
10	1/10/2015	10	6	Saturday	1/4/2015	1
11	1/11/2015	11	0	Sunday	1/11/2015	1
12	1/12/2015	12	1	Monday	1/11/2015	1
13	1/13/2015	13	2	Tuesday	1/11/2015	1
14	1/14/2015	14	3	Wednesday	1/11/2015	1
15	1/15/2015	15	4	Thursday	1/11/2015	1
16	1/16/2015	16	5	Friday	1/11/2015	1
17	1/17/2015	17	6	Saturday	1/11/2015	1
18	1/18/2015	18	0	Sunday	1/18/2015	1
19	1/19/2015	19	1	Monday	1/18/2015	1
20	1/20/2015	20	2	Tuesday	1/18/2015	1
21	1/21/2015	21	3	Wednesday	1/18/2015	1
22	1/22/2015	22	4	Thursday	1/18/2015	1
23	1/23/2015	23	5	Friday	1/18/2015	1
24	1/24/2015	24	6	Saturday	1/18/2015	1
25	1/25/2015	25	0	Sunday	1/25/2015	1

ADDING INDEX COLUMNS

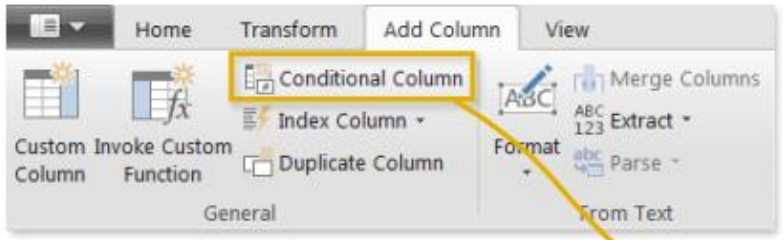


Index Columns contain a list of sequential values that can be used to identify each unique row in a table (*typically starting from 0 or 1*)

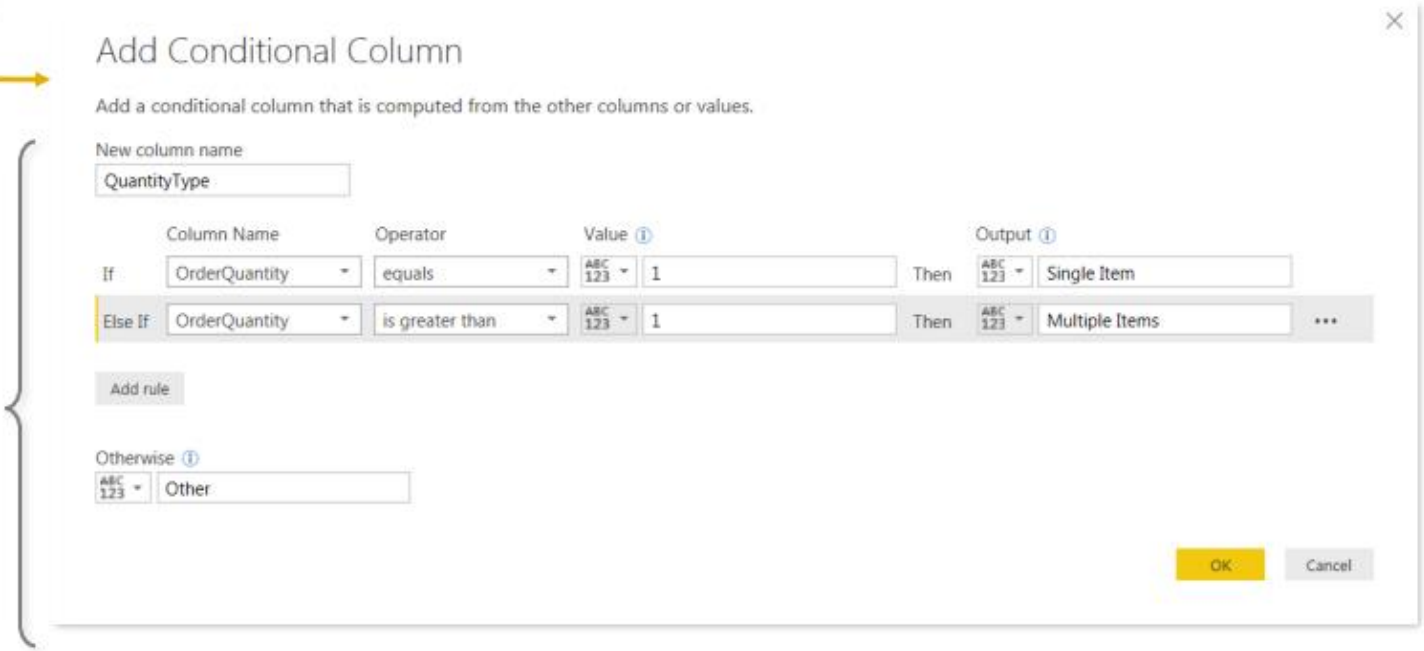
These columns are often used to create **unique IDs** that can be used to form relationships between tables (*more on that later!*)

	Index	OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey
1	1	1/1/2015	9/21/2001	SO45080	332	14657
2	2	1/1/2015	12/5/2001	SO45079	312	29255
3	3	1/1/2015	10/29/2001	SO45082	350	11455
4	4	1/1/2015	11/16/2001	SO45081	338	26782
5	5	1/2/2015	12/15/2001	SO45083	312	14947
6	6	1/2/2015	10/12/2001	SO45084	310	29143
7	7	1/2/2015	12/18/2001	SO45086	314	18747
8	8	1/2/2015	10/9/2001	SO45085	312	18746
9	9	1/3/2015	10/3/2001	SO45093	312	18906
10	10	1/3/2015	9/29/2001	SO45090	310	29170
11	11	1/3/2015	12/11/2001	SO45088	345	11398
12	12	1/3/2015	10/24/2001	SO45092	313	18899
13	13	1/3/2015	12/16/2001	SO45089	351	25977
14	14	1/3/2015	10/26/2001	SO45091	314	18909
15	15	1/3/2015	9/11/2001	SO45087	350	11388
16	16	1/3/2015	9/11/2001	SO45094	310	22785
17	17	1/4/2015	10/30/2001	SO45096	312	12483
18	18	1/4/2015	10/30/2001	SO45097	313	29151

ADDING CONDITIONAL COLUMNS



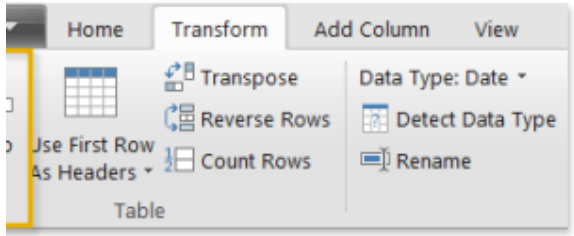
Conditional Columns allow you to define new fields based on logical rules and conditions (*IF/THEN statements*)



*In this case we're creating a new conditional column called "**QuantityType**", which depends on the values in the "**OrderQuantity**" column, as follows:*

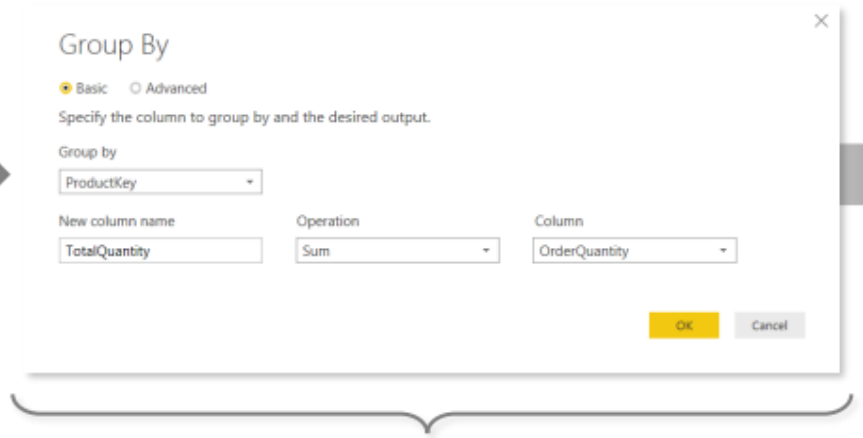
- *If OrderQuantity **=1**, QuantityType = "**Single Item**"*
- *If OrderQuantity **>1**, QuantityType = "**Multiple Items**"*
- *Otherwise QuantityType = "**Other**"*

GROUPING & AGGREGATING DATA



Group By allows you to aggregate your data at a different level
(*i.e. transform daily data into monthly, roll up transaction-level data by store, etc*)

OrderDate	ProductKey	CustomerKey	OrderQuantity
6/25/2017	214	14719	1
7/16/2016	214	11243	1
12/31/2016	214	21452	1
6/29/2017	214	22748	1
10/6/2016	214	25025	1
10/7/2016	214	16504	1
10/13/2016	214	13043	1
1/19/2017	214	23101	1
9/7/2016	214	24900	1
1/19/2017	214	24196	1
6/29/2017	214	12963	1
11/6/2016	214	14570	1
11/13/2016	214	16999	1
7/31/2016	214	12281	1
10/9/2016	214	15685	1
8/1/2016	214	16982	1
12/4/2016	214	12835	1

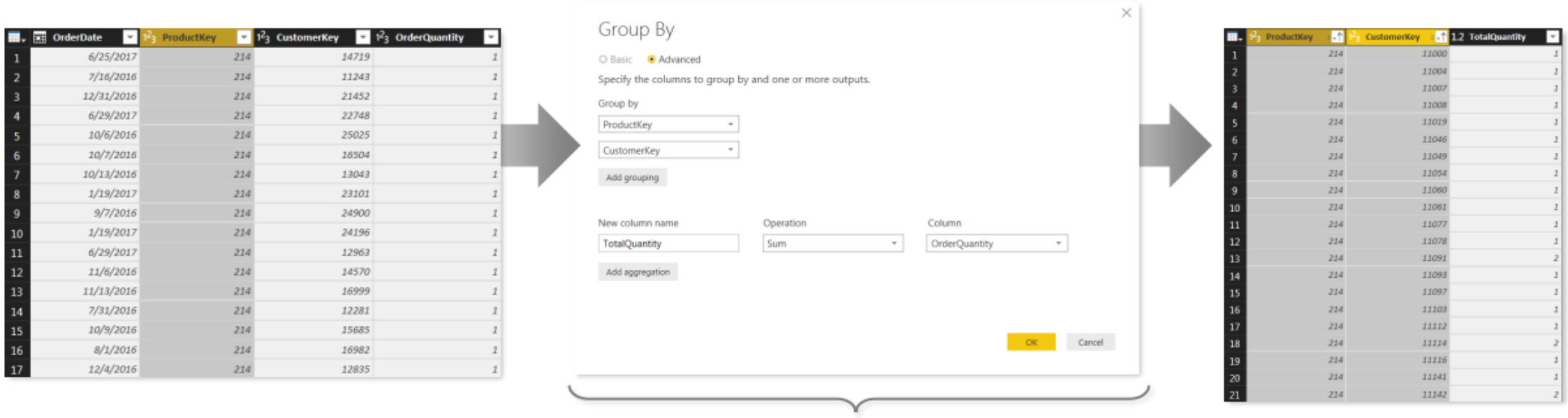


	ProductKey	TotalQuantity
1	214	2099
2	217	1940
3	222	1995
4	225	4151
5	228	392
6	231	408
7	234	424
8	237	381
9	310	169
10	311	139
11	312	179
12	313	168
13	314	157
14	320	10
15	321	55
16	322	5
17	323	34

*In this case we're transforming a daily, transaction-level table into a summary of "**TotalQuantity**" rolled up by "**ProductKey**"*

***NOTE:** Any fields not specified in the Group By settings are lost*

GROUPING & AGGREGATING DATA (ADVANCED)



	OrderDate	ProductKey	CustomerKey	OrderQuantity
1	6/25/2017	214	14719	1
2	7/16/2016	214	11243	1
3	12/31/2016	214	21452	1
4	6/29/2017	214	22748	1
5	10/6/2016	214	25025	1
6	10/7/2016	214	16504	1
7	10/13/2016	214	13043	1
8	1/19/2017	214	23101	1
9	9/7/2016	214	24900	1
10	1/19/2017	214	24196	1
11	6/29/2017	214	12963	1
12	11/6/2016	214	14570	1
13	11/13/2016	214	16999	1
14	7/31/2016	214	12281	1
15	10/9/2016	214	15685	1
16	8/1/2016	214	16982	1
17	12/4/2016	214	12835	1

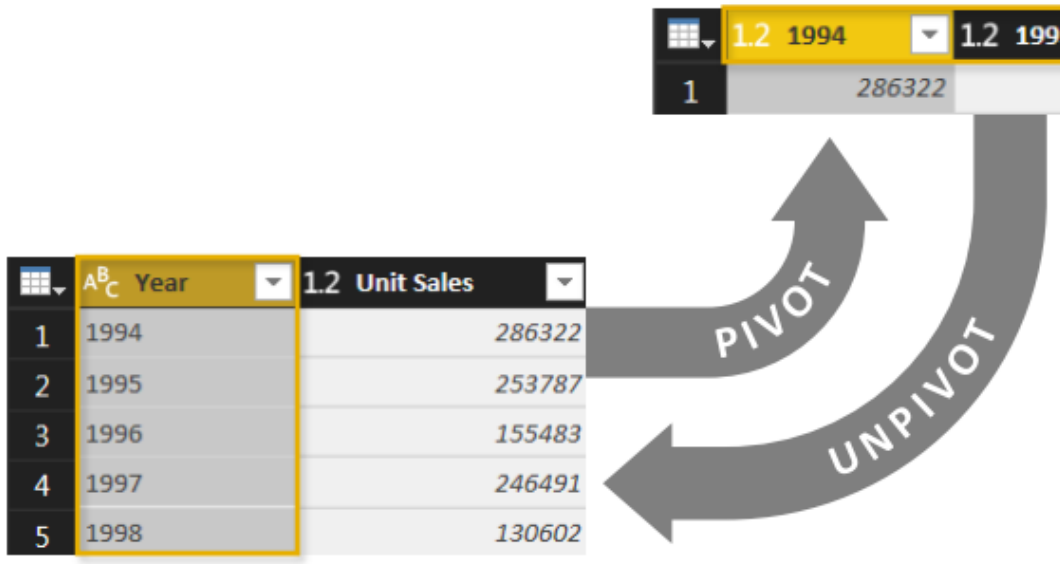
	ProductKey	CustomerKey	TotalQuantity
1	214	11000	1
2	214	11004	1
3	214	11007	1
4	214	11008	1
5	214	11019	1
6	214	11046	1
7	214	11049	1
8	214	11054	1
9	214	11060	1
10	214	11061	1
11	214	11077	1
12	214	11078	1
13	214	11091	2
14	214	11093	1
15	214	11097	1
16	214	11103	1
17	214	11112	1
18	214	11114	2
19	214	11116	1
20	214	11141	1
21	214	11142	2

*This time we're transforming the daily, transaction-level table into a summary of "**TotalQuantity**" aggregated by both "**ProductKey**" and "**CustomerKey**" (using the advanced option in the dialog box)*

***NOTE:** This is similar to creating a PivotTable in Excel and pulling in "**Sum of OrderQuantity**" with **ProductKey** and **CustomerKey** as row labels*

PIVOTING & UNPIVOTING

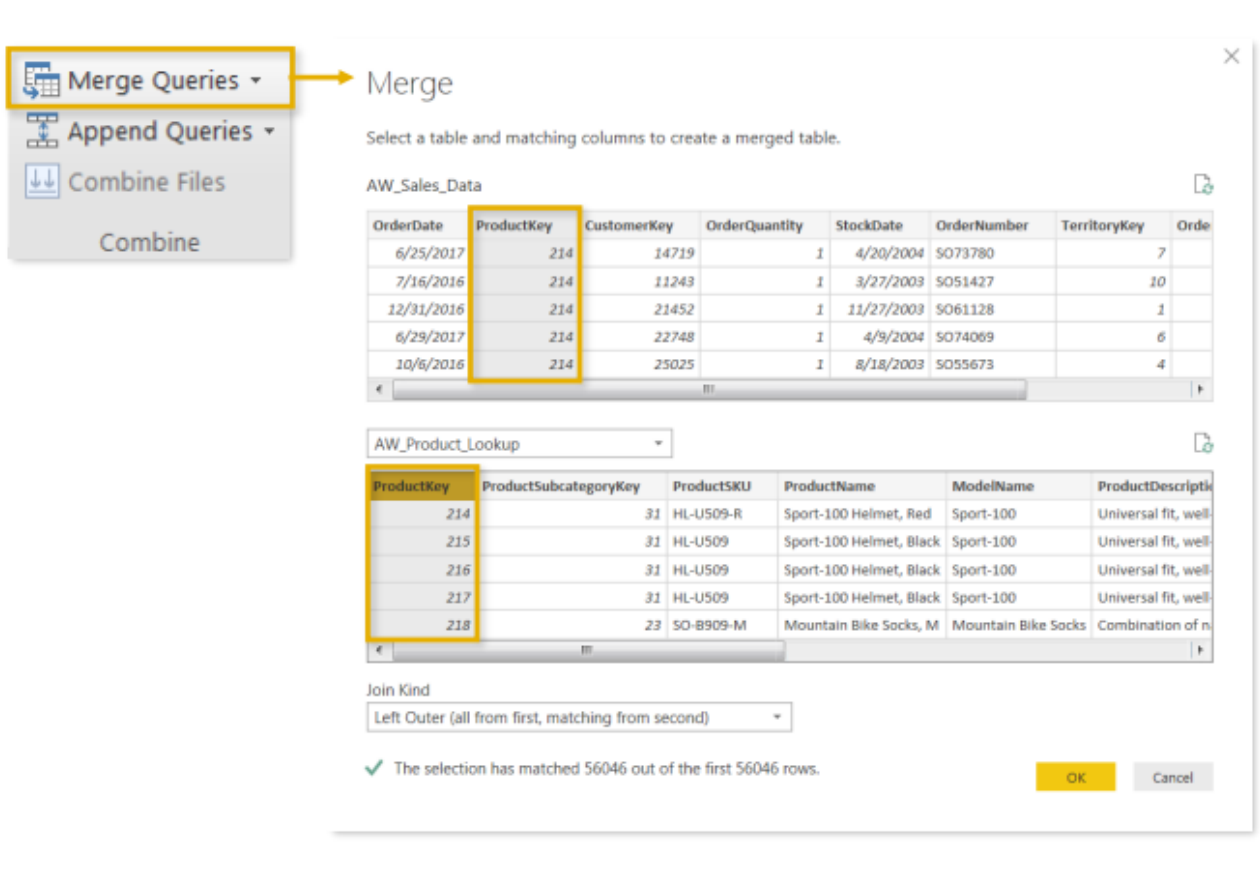
"Pivoting" is a fancy way to describe the process of turning **distinct row values** into **columns** (*"pivoting"*) or turning **columns** into **rows** (*"unpivoting"*)



*Imagine that the table is on a hinge; pivoting is like rotating it from a **vertical** to a **horizontal** layout, and unpivoting is like rotating it from **horizontal** to **vertical***

***NOTE: Transpose** works very similarly, but doesn't recognize unique values; instead, the entire table is transformed so that each row becomes a column and vice versa*

MERGING QUERIES



Merging queries allows you to **join tables** based on a common column (like VLOOKUP)

In this case we're merging the **AW_Sales_Data** table with the **AW_Product_Lookup** table, which share a common "*ProductKey*" column

NOTE: Merging ***adds columns*** to an existing table

HEY THIS IS IMPORTANT!

Just because you ***can*** merge tables, doesn't mean you ***should***.

In general, it's better to keep tables separate and define **relationships** between them (*more on that later!*)

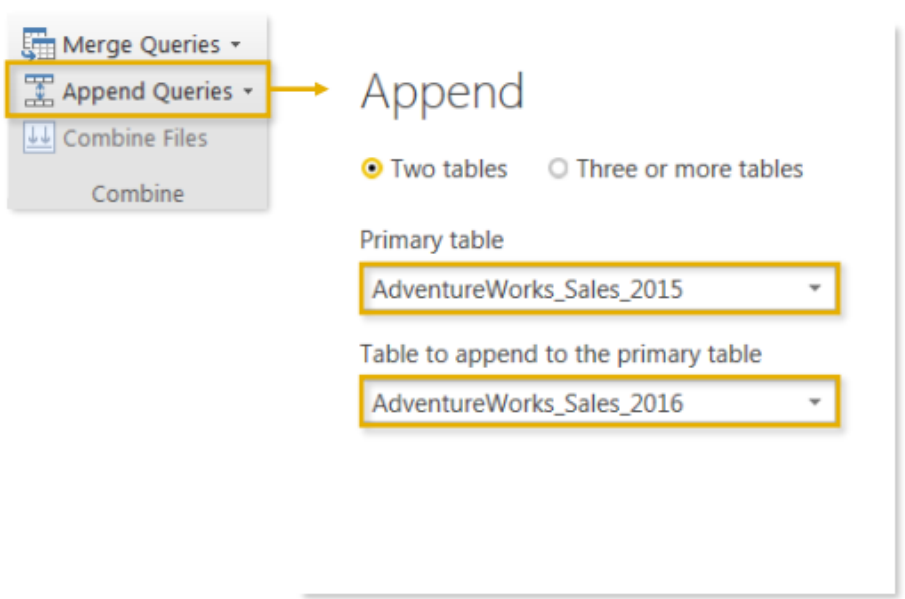
მადლობა ყურადღებისთვის

ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

- Append Queries ბრძანება, Append ბრძანების ფაილზე მიბმა;
- Query-ს განახლების პარამეტრები; დამატებითი მონაცემთა ტიპები და კატეგორიები;
- იერარქიის განსაზღვრა; ექსელის ფაილის იმპორტირება
- მონაცემთა კავშირის საუკეთესო პრაქტიკა

APPENDING QUERIES



Appending queries allows you to **combine** (or ***stack***) tables that share the exact same column structure and data types

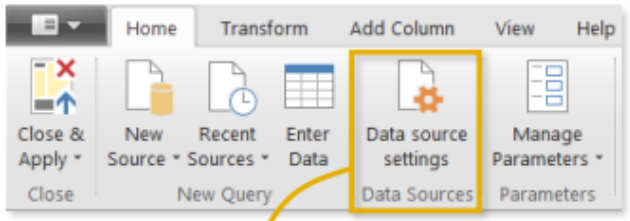
In this case we're appending the **AdventureWorks_Sales_2015** table to the **AdventureWorks_Sales_2016** table, which is valid since they share identical table structures

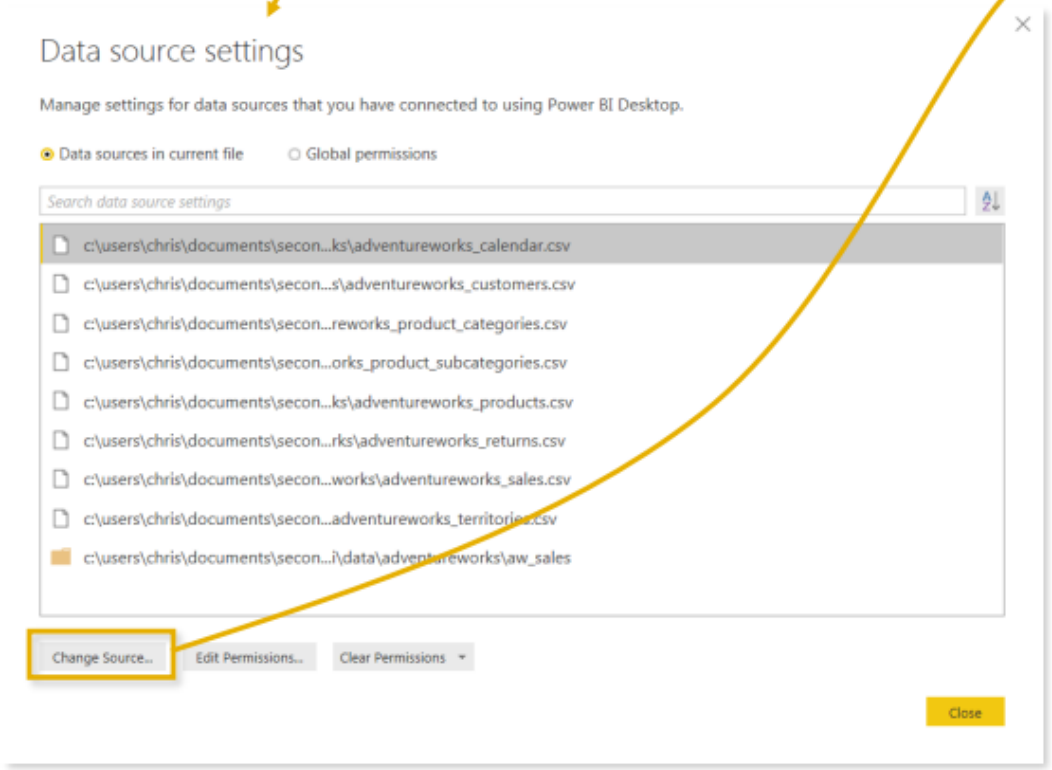
NOTE: Appending ***adds rows*** to an existing table

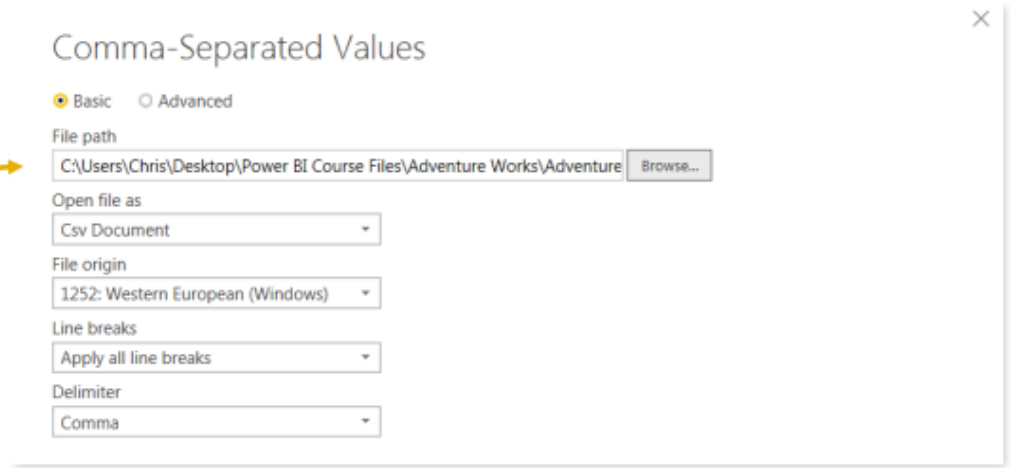
PRO TIP:

*Use the "**Folder**" option (Get Data > More > Folder) to append all files within a folder (assuming they share the same structure); as you add new files, simply refresh the query and they will automatically append!*

DATA SOURCE SETTINGS







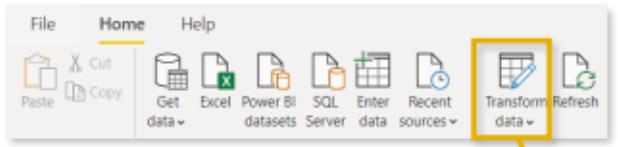
The **Data Source Settings** in the Query Editor allow you to manage data connections and permissions

HEY THIS IS IMPORTANT!

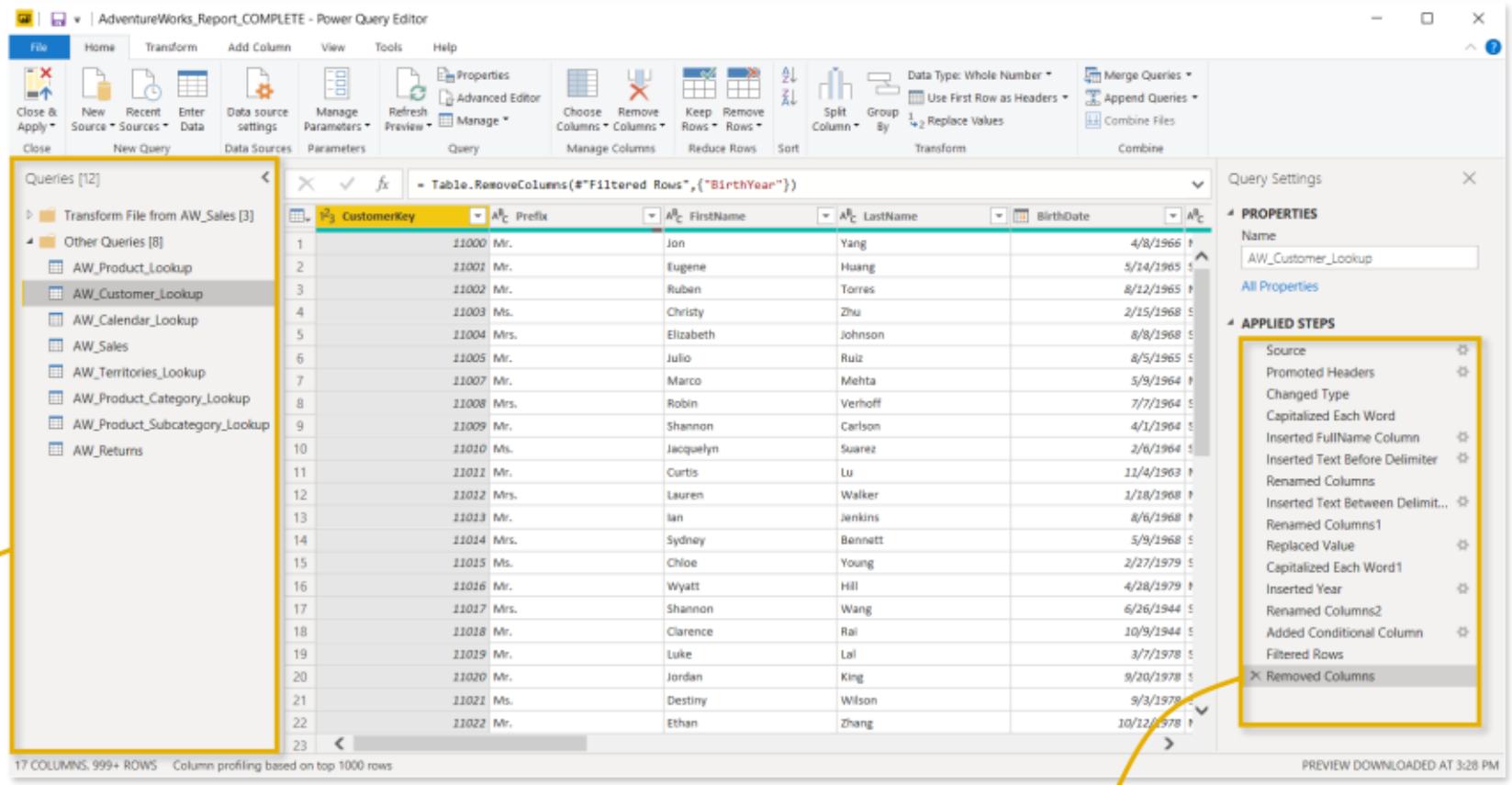
Connections to local files reference the *exact* path

If the file name or location changes, **you will need to change the source and browse to the current version**

MODIFYING QUERIES



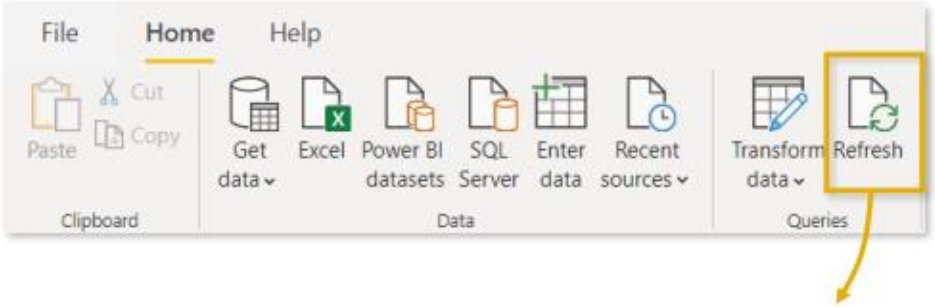
Select **Transform Data*** from the **Home** tab to launch the Query Editor



Within the editor, view or modify existing queries in the "**Queries**" pane

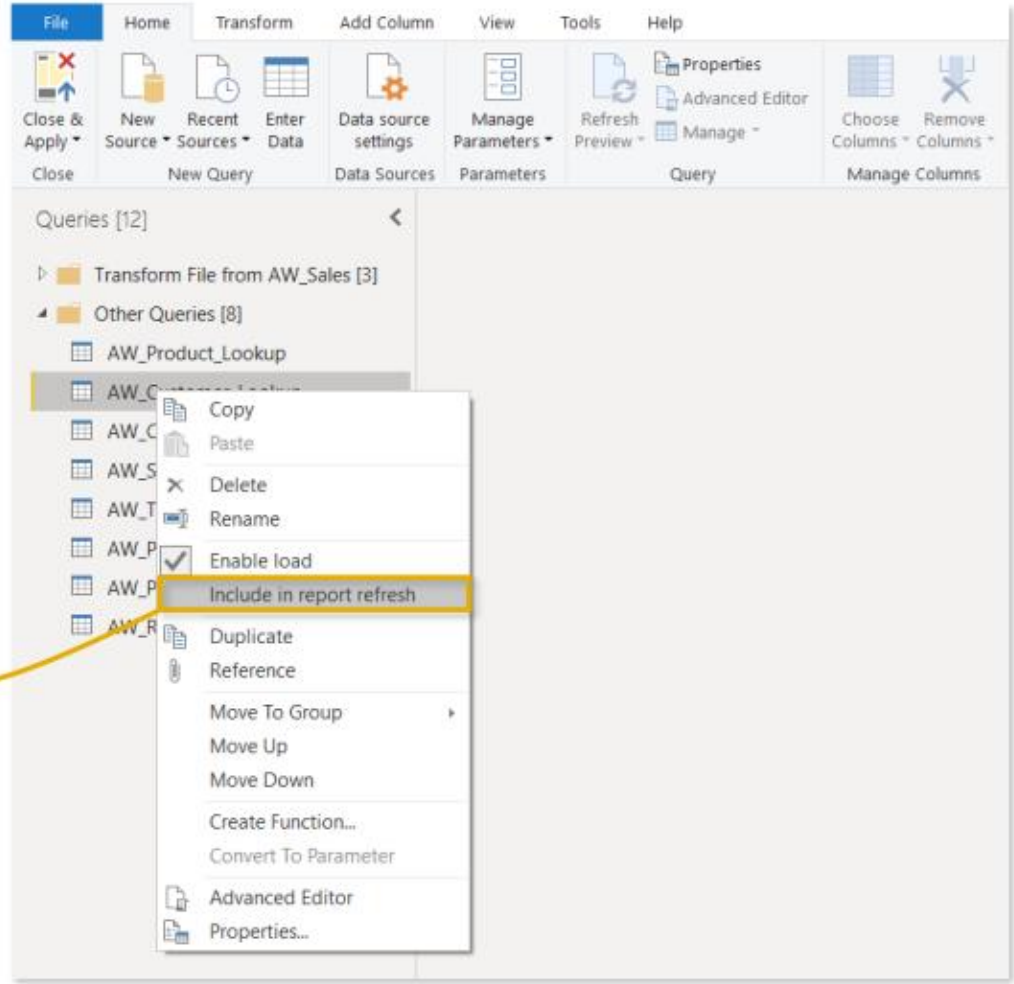
Within each query, you can click each item within the "**Applied Steps**" pane to view each stage of the transformation, add new steps or delete existing ones, or modify individual steps by clicking the gear icons

REFRESHING QUERIES



By default, **ALL** queries in the model will refresh when you use the "*Refresh*" command from the **Home** tab

From the Query Editor, uncheck "***Include in report refresh***" to exclude individual queries from the refresh



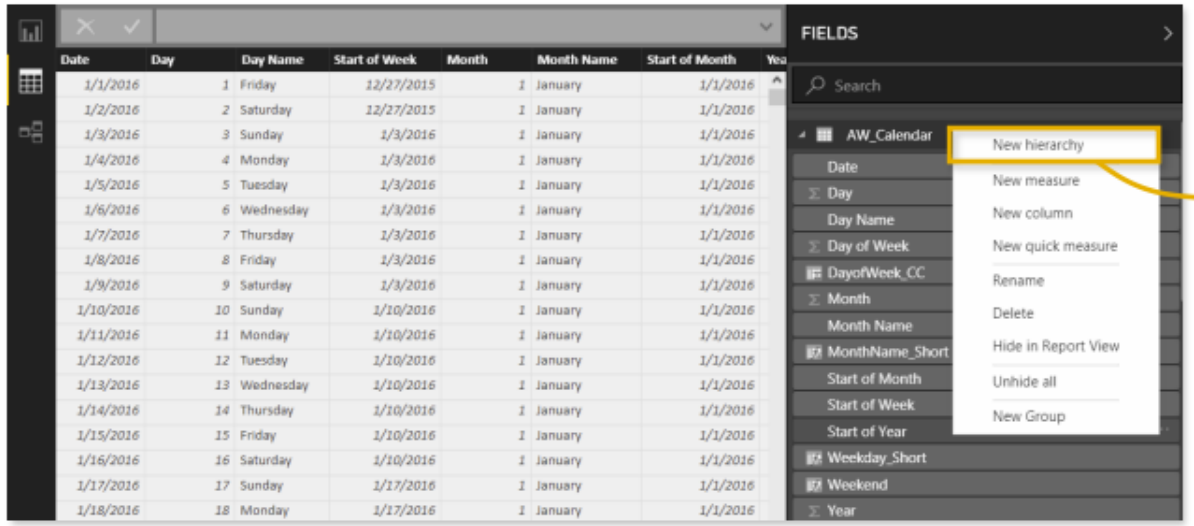
PRO TIP:

Exclude queries that don't change often, like lookups or static data tables

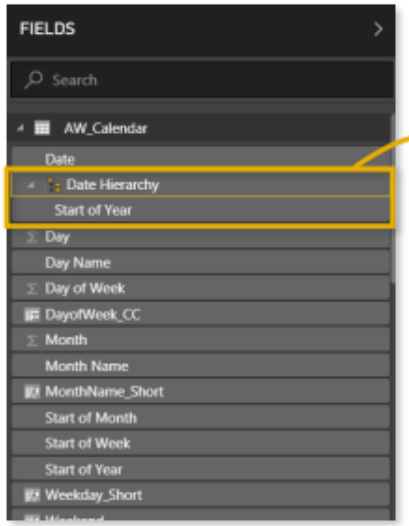
DEFINING HIERARCHIES

Hierarchies are groups of nested columns that reflect multiple levels of granularity

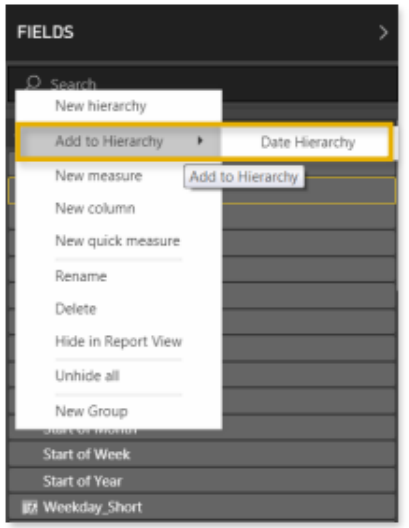
- For example, a "**Geography**" hierarchy might include **Country**, **State**, and **City** columns
- Each hierarchy can be treated as a **single item** in tables and reports, allowing users to "drill up" and "drill down" through different levels of the hierarchy in a meaningful way



1) From within the **Data** view, right-click a field (or click the ellipsis) and select "**New hierarchy**" (here we've selected "*Start of Year*")



2) This creates a hierarchy field containing "*Start of Year*", which we've renamed "**Date Hierarchy**"



3) Right-click other fields (like "*Start of Month*") and select "**Add to Hierarchy**"

BEST PRACTICES: CONNECTING & SHAPING DATA

Get yourself organized, *before* loading the data into Power BI

- *Define clear and intuitive table names (no spaces!) from the start; updating them later can be a headache, especially if you've referenced them in multiple places*
- *Establish a file/folder structure that makes sense from the start, to avoid having to modify data source settings if file names or locations change*

Disabling report refresh for any static sources

- *There's no need to constantly refresh sources that don't update frequently (or at all), like lookups or static data tables; only enable refresh for tables that will be changing*

When working with large tables, only load the data you need

- *Don't include hourly data when you only need daily, or product-level transactions when you only care about store-level performance; extra data will only slow you down*

მადლობა ყურადღებისთვის

ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

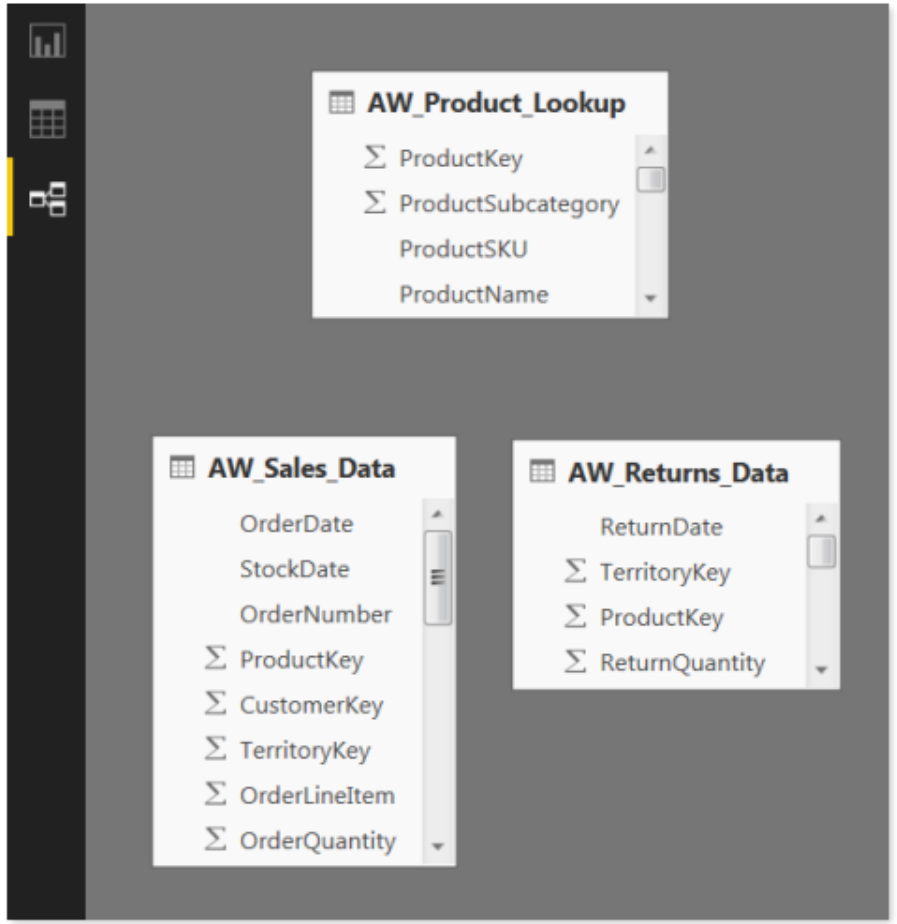
რა არის მონაცემთა მოდელი;
მონაცემთა ბაზის ნორმალიზაციის პრინციპები;
განსხვავება მონაცემთა ცხრილებს და საძიებო ცხრილებს შორის;

განსხვავება Table Relationship და Merged Queries შორის; Table Relationship-ს შექმნა;

"Snowflake" სქემები; Relationship-ის მართვა და რედაქტირება;
აქტიური და არააქტიური Relationship-ის მართვა;

Relationship-ის კარდინალურობა;
მრავალი მონაცემთა ცხრილის დაკავშირება; ფილტრის ნაკადი; ორმხრივი ფილტრები;
ველების დამალვა Report View-დან

WHAT'S A "**DATA MODEL**"?

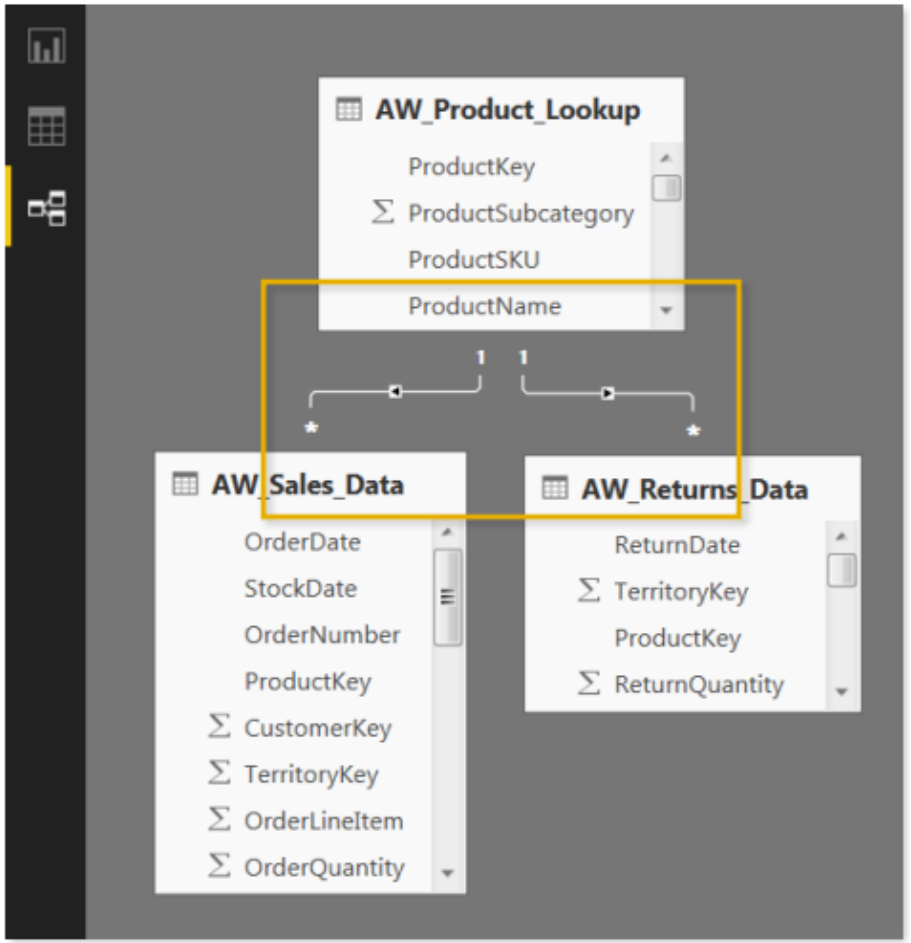


This **IS NOT** a data model

- This is a collection of independent tables, which share no connections or relationships
- If you tried to visualize **Orders** and **Returns** by **Product**, this is what you'd get

ProductName	OrderQuantity	ReturnQuantity
All-Purpose Bike Stand	84,174	1,828
AWC Logo Cap	84,174	1,828
Bike Wash - Dissolver	84,174	1,828
Cable Lock	84,174	1,828
Chain	84,174	1,828
Classic Vest, L	84,174	1,828
Classic Vest, M	84,174	1,828
Classic Vest, S	84,174	1,828
Fender Set - Mountain	84,174	1,828
Total	**84,174**	**1,828**

WHAT'S A "**DATA MODEL**"?



This **IS** a data model!

- The tables are connected via relationships, based on the common *ProductKey* field
- Now the **Sales** and **Returns** tables know how to filter using fields from the **Product** table!

ProductName	OrderQuantity	ReturnQuantity
All-Purpose Bike Stand	234	8
AWC Logo Cap	4,151	46
Bike Wash - Dissolver	1,706	25
Classic Vest, L	182	4
Classic Vest, M	182	7
Classic Vest, S	157	8
Fender Set - Mountain	3,960	54
Half-Finger Gloves, L	840	18
Half-Finger Gloves, M	918	16
Total	**84,174**	**1,828**

DATABASE NORMALIZATION

Normalization is the process of organizing the tables and columns in a relational database to reduce redundancy and preserve data integrity. It's commonly used to:

- **Eliminate redundant data** to decrease table sizes and improve processing speed & efficiency
- **Minimize errors and anomalies** from data modifications (inserting, updating or deleting records)
- **Simplify queries** and structure the database for meaningful analysis

TIP: In a normalized database, each table should serve a ***distinct*** and ***specific*** purpose (*i.e. product information, dates, transaction records, customer attributes, etc.*)

date	product_id	quantity	product_brand	product_name	product_sku	product_weight
1/1/1997	869	5	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/7/1997	869	2	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/3/1997	1	4	Washington	Washington Berry Juice	90748583674	8.39
1/1/1997	1472	3	Fort West	Fort West Fudge Cookies	37276054024	8.28
1/6/1997	1472	2	Fort West	Fort West Fudge Cookies	37276054024	8.28
1/5/1997	2	4	Washington	Washington Mango Drink	96516502499	7.42
1/1/1997	76	4	Red Spade	Red Spade Sliced Chicken	62054644227	18.1
1/1/1997	76	2	Red Spade	Red Spade Sliced Chicken	62054644227	18.1
1/5/1997	3	2	Washington	Washington Strawberry Drink	58427771925	13.1
1/7/1997	3	2	Washington	Washington Strawberry Drink	58427771925	13.1
1/1/1997	320	3	Excellent	Excellent Cranberry Juice	36570182442	16.4

*When you **don't** normalize, you end up with tables like this; all of the rows with duplicate product info could be eliminated with a lookup table based on **product_id***

This may not seem critical now, but minor inefficiencies can become major problems as databases scale in size!

DATA TABLES VS. LOOKUP TABLES

Models generally contain two types of tables: **data** (or "*fact*") tables, and **lookup** (or "*dimension*") tables

- **Data tables** contain *numbers* or *values*, typically at a granular level, with ID or "*key*" columns that can be used to create table relationships
- **Lookup tables** provide descriptive, often text-based *attributes* about each dimension in a table

date	product_id	quantity
1/1/1997	869	5
1/1/1997	1472	3
1/1/1997	76	4
1/1/1997	320	3
1/1/1997	4	4
1/1/1997	952	4
1/1/1997	1222	4
1/1/1997	517	4
1/1/1997	1359	4
1/1/1997	357	4
1/1/1997	1426	5
1/1/1997	190	4
1/1/1997	367	4
1/1/1997	250	5
1/1/1997	600	4
1/1/1997	702	5

*This **Data Table** contains "**quantity**" values, and connects to lookup tables via the "**date**" and "**product_id**" columns*

date	day_of_month	month	year	weekday	week_of_year	week_ending	month_name	quarter
1/1/1997	1	1	1997	Wednesday	1	1/5/1997	January	Q1
1/2/1997	2	1	1997	Thursday	1	1/5/1997	January	Q1
1/3/1997	3	1	1997	Friday	1	1/5/1997	January	Q1
1/4/1997	4	1	1997	Saturday	1	1/5/1997	January	Q1
1/5/1997	5	1	1997	Sunday	2	1/5/1997	January	Q1
1/6/1997	6	1	1997	Monday	2	1/12/1997	January	Q1

*This **Calendar Lookup** table provides additional attributes about each **date** (month, year, weekday, quarter, etc.)*

product_id	product_brand	product_name	product_sku	product_retail_price	product_cost	product_weight
1	Washington	Washington Berry Juice	90748583674	2.85	0.94	8.39
2	Washington	Washington Mango Drink	96516502499	0.74	0.26	7.42
3	Washington	Washington Strawberry Drink	58427771925	0.83	0.4	13.1
4	Washington	Washington Cream Soda	64412155747	3.64	1.64	10.6
5	Washington	Washington Diet Soda	85561191439	2.19	0.77	6.66
6	Washington	Washington Cola	29804642796	1.15	0.37	15.8
7	Washington	Washington Diet Cola	20191444754	2.61	0.91	18
8	Washington	Washington Orange Juice	89770532250	2.59	0.8	8.97

*This **Product Lookup** table provides additional attributes about each **product** (brand, product name, sku, price, etc.)*

PRIMARY VS. FOREIGN KEYS

date	product_id	quantity
1/1/1997	869	5
1/1/1997	1472	3
1/1/1997	76	4
1/1/1997	320	3
1/1/1997	4	4
1/1/1997	952	4
1/1/1997	1222	4
1/1/1997	517	4
1/1/1997	1359	4
1/1/1997	357	4
1/1/1997	1426	5
1/1/1997	190	4
1/1/1997	367	4
1/1/1997	250	5
1/1/1997	600	4
1/1/1997	702	5

These columns are **foreign keys**; they contain *multiple* instances of each value, and are used to match the **primary keys** in related lookup tables

date	day_of_month	month	year	weekday	week_of_year	week_ending	month_name	quarter
1/1/1997	1	1	1997	Wednesday	1	1/5/1997	January	Q1
1/2/1997	2	1	1997	Thursday	1	1/5/1997	January	Q1
1/3/1997	3	1	1997	Friday	1	1/5/1997	January	Q1
1/4/1997	4	1	1997	Saturday	1	1/5/1997	January	Q1
1/5/1997	5	1	1997	Sunday	2	1/5/1997	January	Q1
1/6/1997	6	1	1997	Monday	2	1/12/1997	January	Q1

product_id	product_brand	product_name	product_sku	product_retail_price	product_cost	product_weight
1	Washington	Washington Berry Juice	90748583674	2.85	0.94	8.39
2	Washington	Washington Mango Drink	96516502499	0.74	0.26	7.42
3	Washington	Washington Strawberry Drink	58427771925	0.83	0.4	13.1
4	Washington	Washington Cream Soda	64412155747	3.64	1.64	10.6
5	Washington	Washington Diet Soda	85561191439	2.19	0.77	6.66
6	Washington	Washington Cola	29804642796	1.15	0.37	15.8
7	Washington	Washington Diet Cola	20191444754	2.61	0.91	18
8	Washington	Washington Orange Juice	89770532250	2.59	0.8	8.97

These columns are **primary keys**; they *uniquely* identify each row of a table, and match the **foreign keys** in related data tables

RELATIONSHIPS VS. MERGED TABLES

*Can't I just **merge queries** or use **LOOKUP** or **RELATED** functions to pull those attributes into the fact table itself, so that I have everything in one place??*

-Anonymous confused man

*Original **Fact Table** fields* | *Attributes from **Calendar Lookup** table* | *Attributes from **Product Lookup** table*

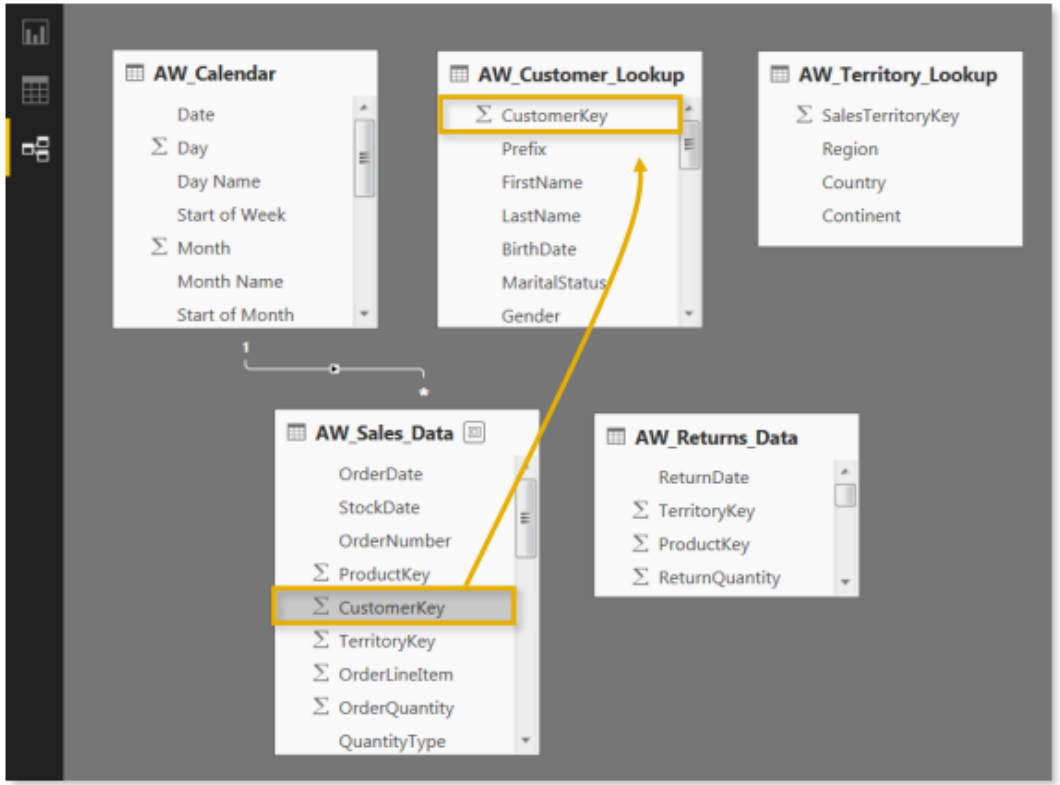
date	product_id	quantity	day_of_month	month	year	weekday	month_name	quarter	product_brand	product_name	product_sku	product_weight
1/1/1997	869	5	1	1	1997	Wednesday	January	Q1	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/7/1997	869	2	7	1	1997	Tuesday	January	Q1	Nationeel	Nationeel Grape Fruit Roll	52382137179	17
1/3/1997	1	4	3	1	1997	Friday	January	Q1	Washington	Washington Berry Juice	90748583674	8.39
1/1/1997	1472	3	1	1	1997	Wednesday	January	Q1	Fort West	Fort West Fudge Cookies	37276054024	8.28
1/6/1997	1472	2	6	1	1997	Monday	January	Q1	Fort West	Fort West Fudge Cookies	37276054024	8.28
1/5/1997	2	4	5	1	1997	Sunday	January	Q1	Washington	Washington Mango Drink	96516502499	7.42
1/1/1997	76	4	1	1	1997	Wednesday	January	Q1	Red Spade	Red Spade Sliced Chicken	62054644227	18.1
1/1/1997	76	2	1	1	1997	Wednesday	January	Q1	Red Spade	Red Spade Sliced Chicken	62054644227	18.1
1/5/1997	3	2	5	1	1997	Sunday	January	Q1	Washington	Washington Strawberry Drink	58427771925	13.1
1/7/1997	3	2	7	1	1997	Tuesday	January	Q1	Washington	Washington Strawberry Drink	58427771925	13.1
1/1/1997	320	3	1	1	1997	Wednesday	January	Q1	Excellent	Excellent Cranberry Juice	36570182442	16.4

Sure you can, ***but it's inefficient!***

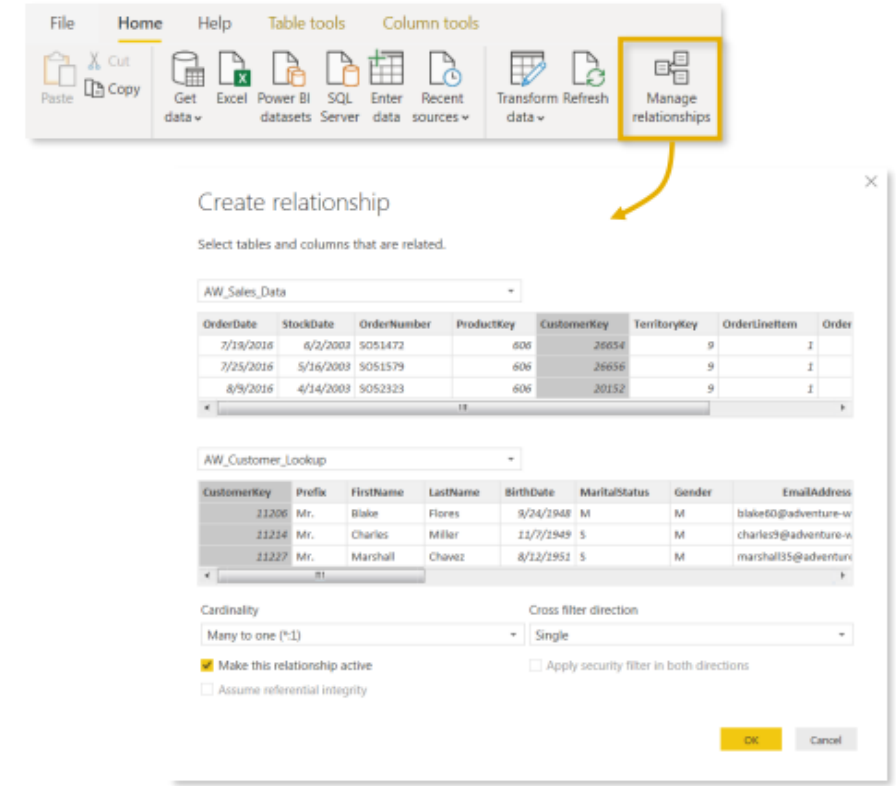
- Merging data in this way creates **redundant data** and utilizes **significantly more memory and processing power** than creating relationships between multiple small tables

CREATING TABLE RELATIONSHIPS

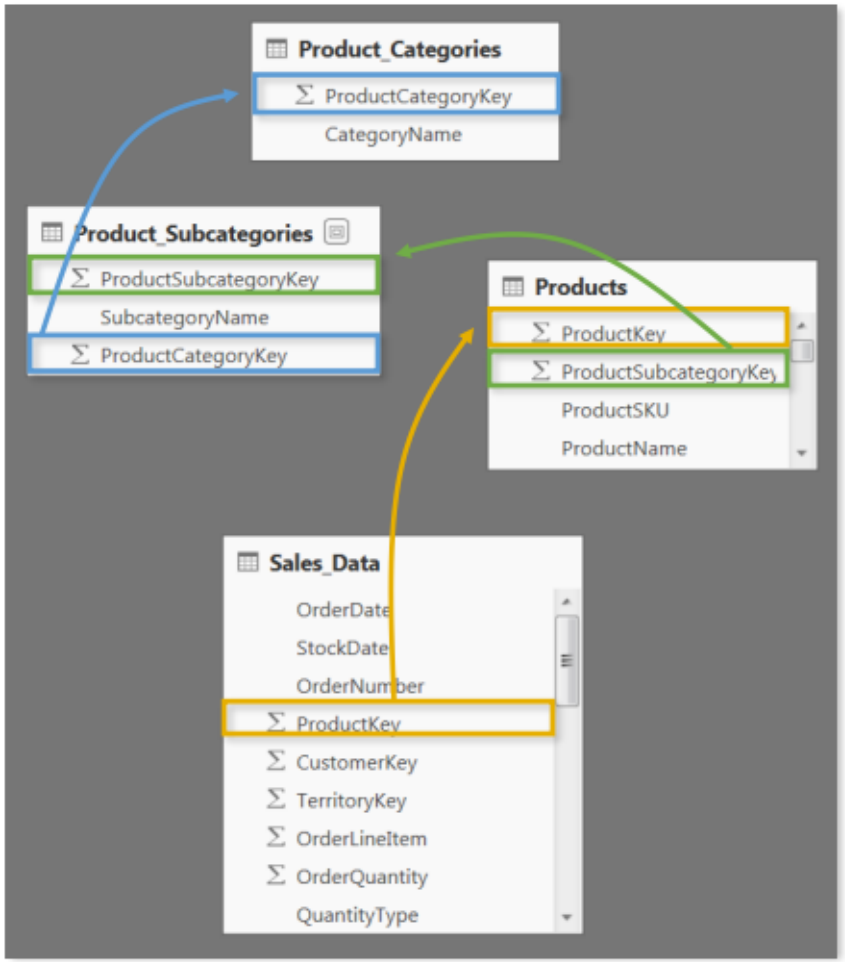
Option 1: Click and drag to connect primary and foreign keys within the **Relationships** pane



Option 2: Add or detect relationships using the "**Manage Relationships**" dialog box



CREATING “SNOWFLAKE” SCHEMAS



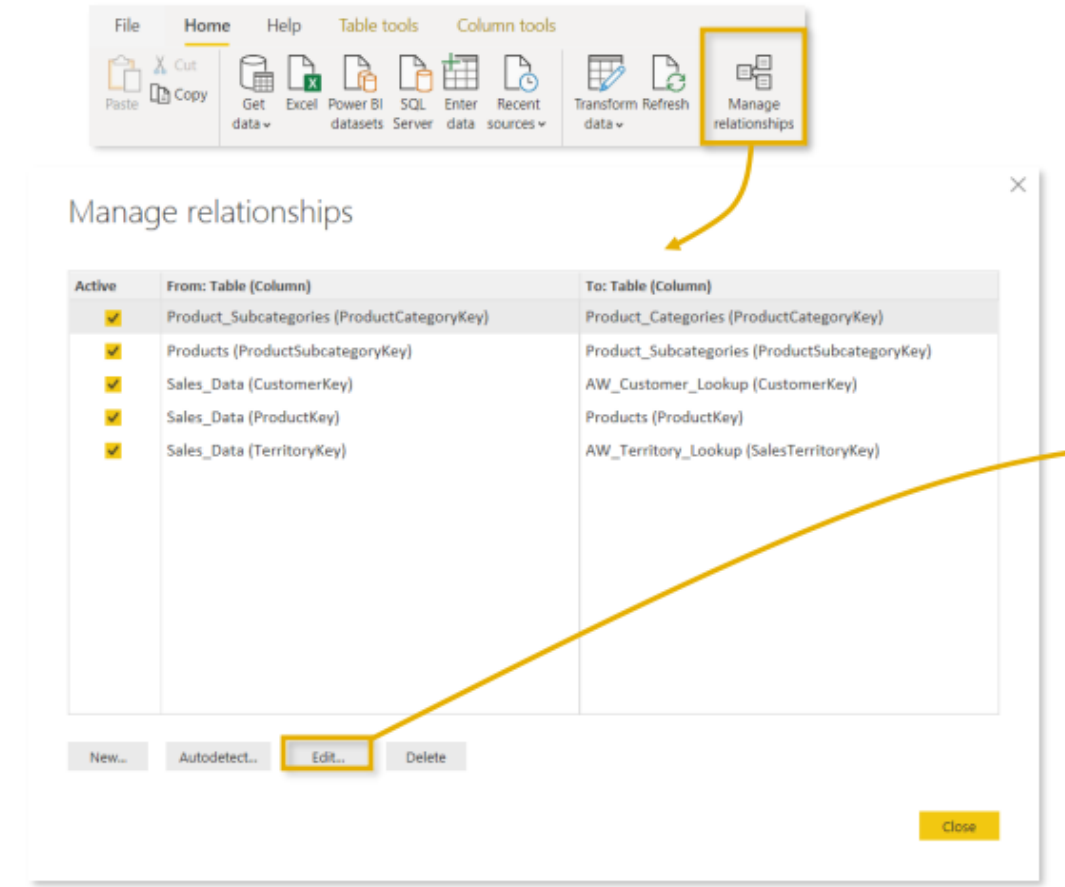
The **Sales_Data** table can connect to **Products** using the ***ProductKey*** field, but cannot connect directly to the **Subcategories** or **Categories** tables

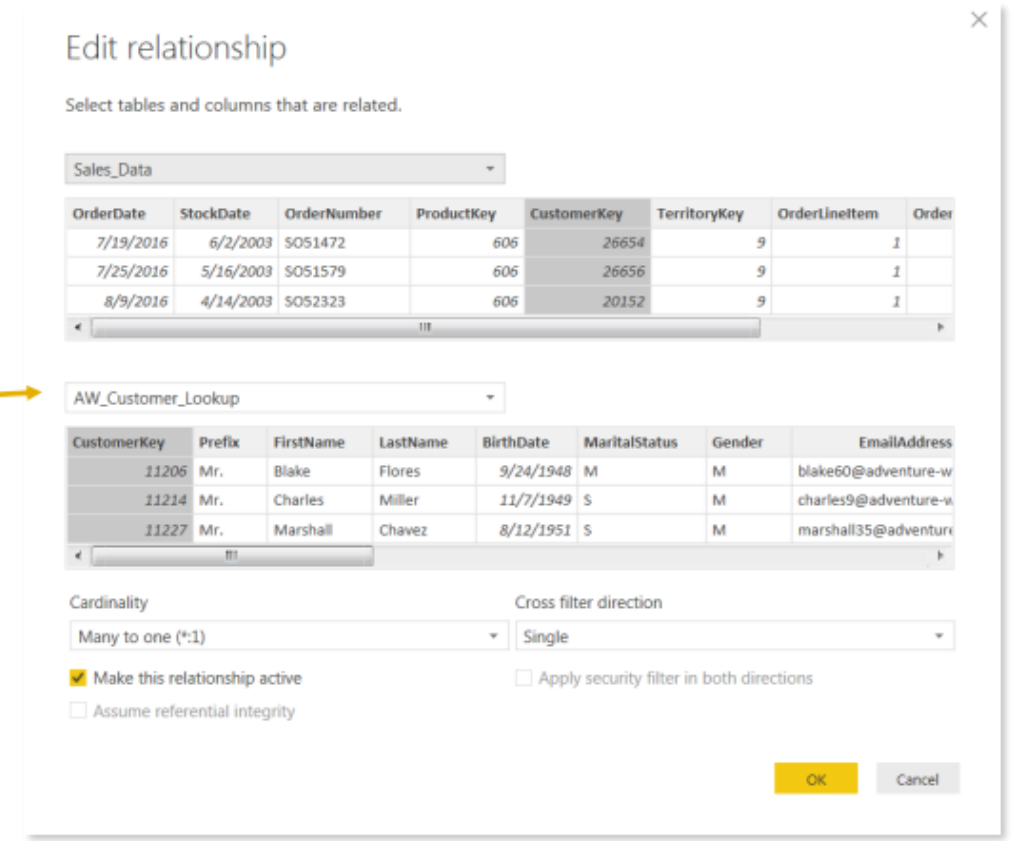
By creating relationships from **Products** to **Subcategories** (using ***ProductSubcategoryKey***) and **Subcategories** to **Categories** (using ***ProductCategoryKey***), we have essentially connected **Sales_Data** to each lookup table; filter context will now flow all the way down the chain

PRO TIP:

*Models with chains of dimension tables are often called “**snowflake**” schemas (whereas “**star**” schemas usually have individual lookup tables surrounding a central data table)*

MANAGING & EDITING RELATIONSHIPS

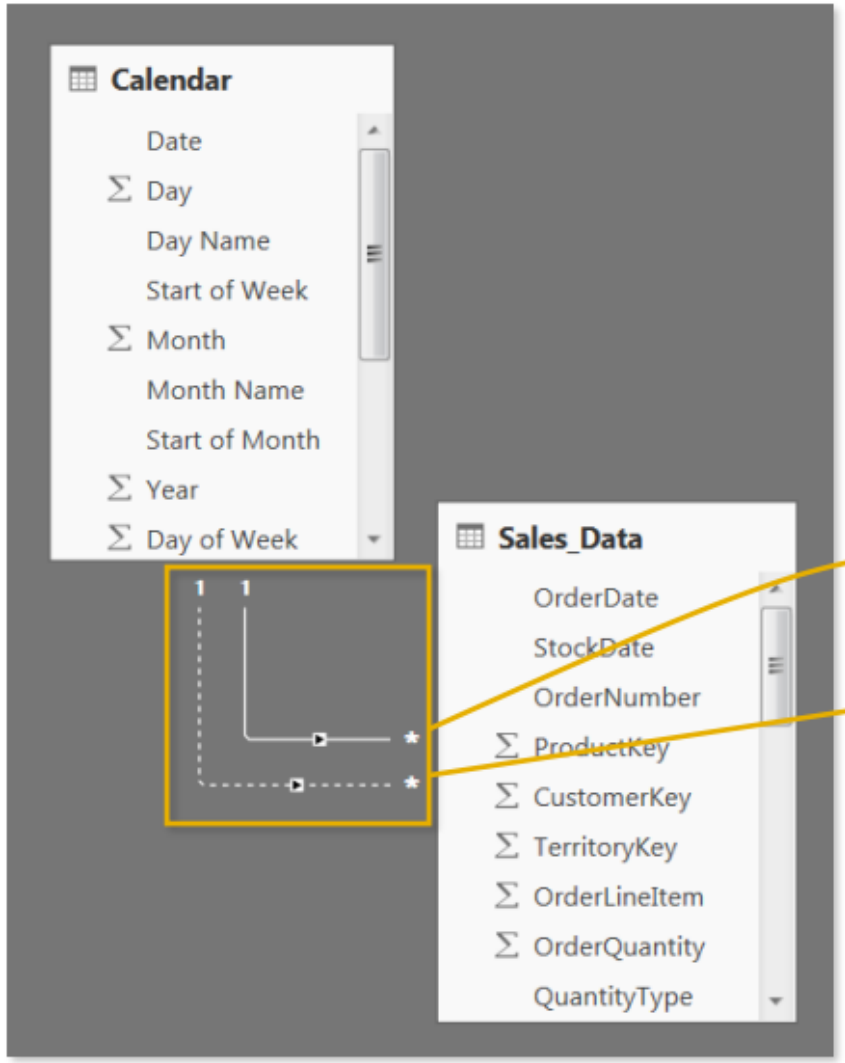


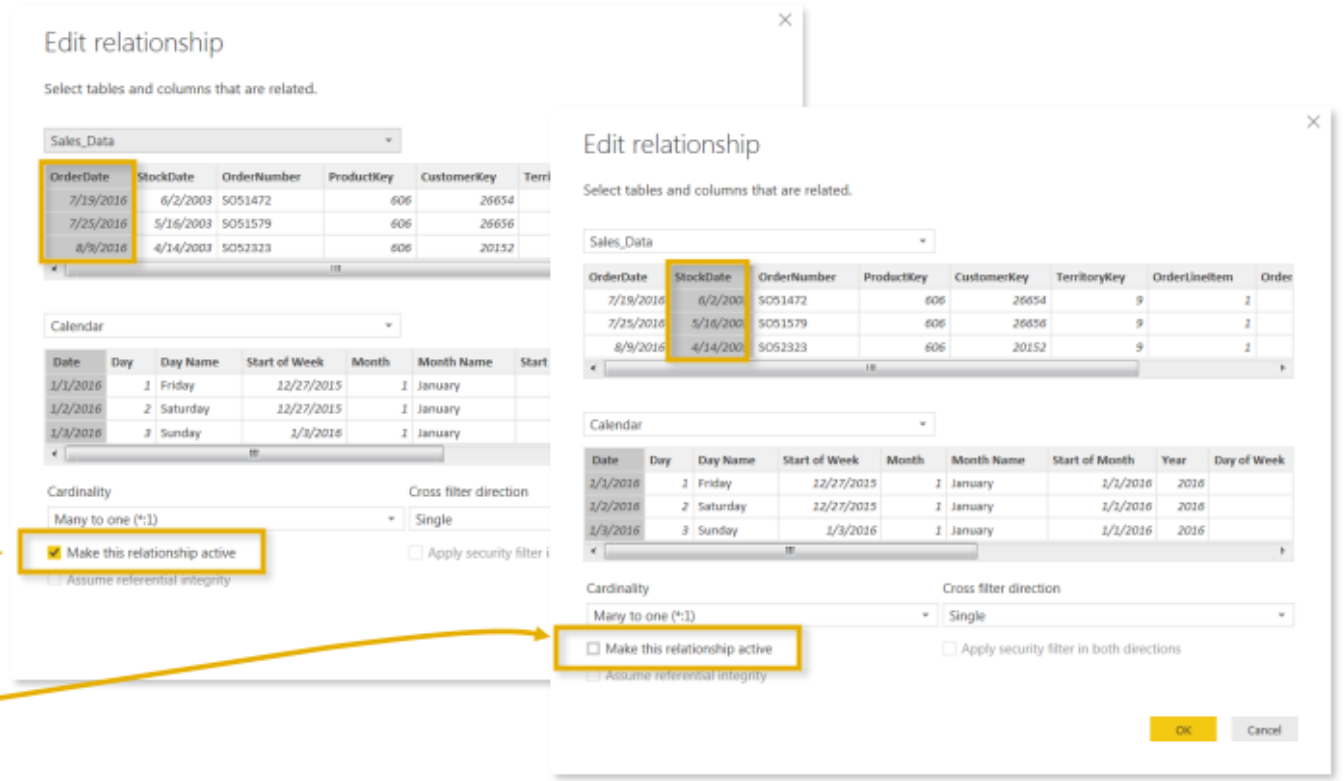


*The "**Manage Relationships**" dialog box allows you to **add**, **edit**, or **delete** table relationships*

*Editing tools allow you to **activate/deactivate** relationships, view **cardinality**, and modify the **cross filter direction** (stay tuned!)*

ACTIVE VS. INACTIVE RELATIONSHIPS

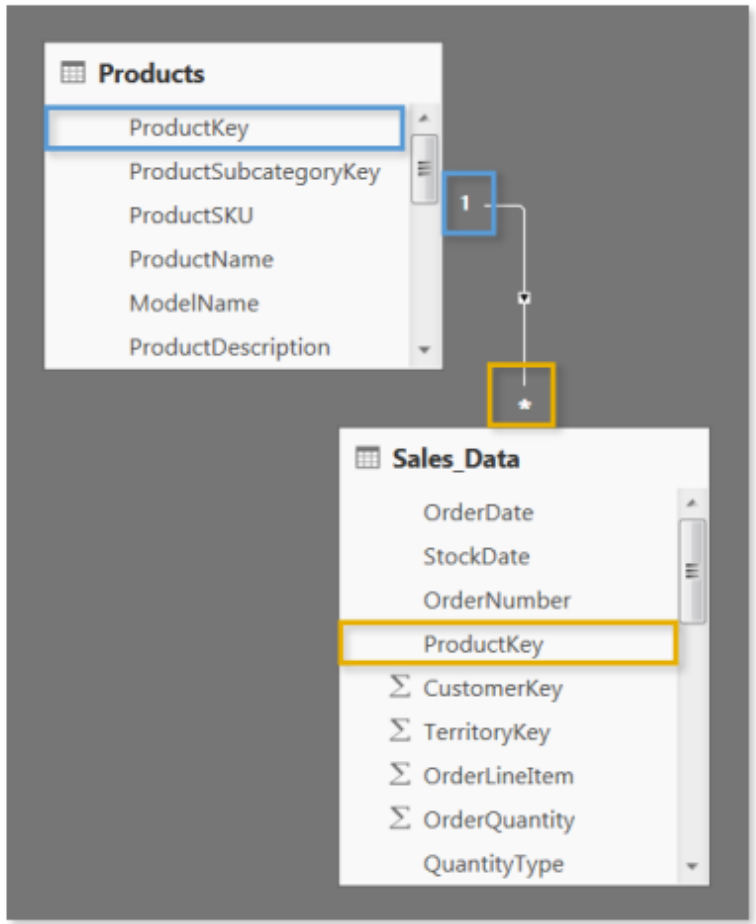




The **Sales_Data** table contains two date fields (***OrderDate*** & ***StockDate***), but there can only be one *active* relationship to the Date field in the Calendar table

Double-click the relationship line, and check the "***Make this relationship active***" box to toggle (note that you have to deactivate one in order to activate another)

RELATIONSHIP CARDINALITY



Cardinality refers to the *uniqueness of values* in a column

- For our purposes, all relationships in the data model should follow a "**one-to-many**" cardinality; **one** instance of each *primary key*, but potentially **many** instances of each *foreign key*

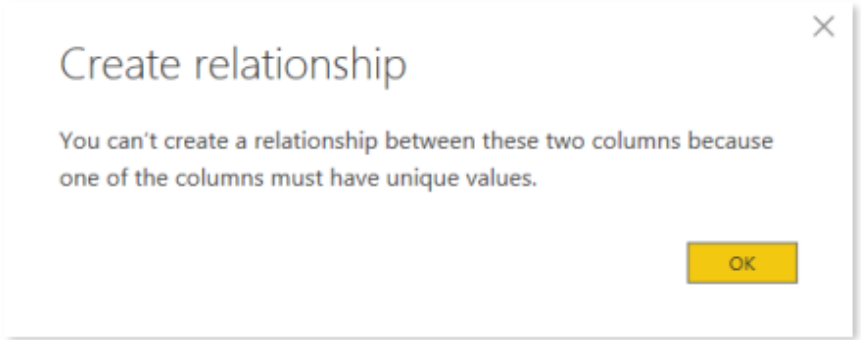
*In this case, there is only **ONE instance of each ProductKey** in the Products table (noted by the "**1**"), since each row contains **attributes of a single product** (Name, SKU, Description, Retail Price, etc)*

*There are **MANY instances of each ProductKey** in the Sales_Data table (noted by the asterisk *), since there are **multiple sales associated with each product***

CARDINALITY CASE STUDY: MANY-TO-MANY

product_id	product_name	product_sku
4	Washington Cream Soda	64412155747
4	Washington Diet Cream Soda	81727382373
5	Washington Diet Soda	85561191439
7	Washington Diet Cola	20191444754
8	Washington Orange Juice	89770532250

date	product_id	transactions
1/1/2017	4	12
1/2/2017	4	9
1/3/2017	4	11
1/1/2017	5	16
1/2/2017	5	19
1/1/2017	7	11



- If we try to connect these tables using **product_id**, we'll get a "***many-to-many relationship***" error since there are multiple instances of each ID in both tables
- Even if we *could* create this relationship, how would you know which product was actually sold on each date – ***Cream Soda*** or ***Diet Cream Soda***?

CARDINALITY CASE STUDY: ONE-TO-ONE

product_id	product_name	product_sku
4	Washington Cream Soda	64412155747
5	Washington Diet Soda	85561191439
7	Washington Diet Cola	20191444754
8	Washington Orange Juice	89770532250

product_id	product_price
4	$3.64
5	$2.19
7	$2.61
8	$2.59

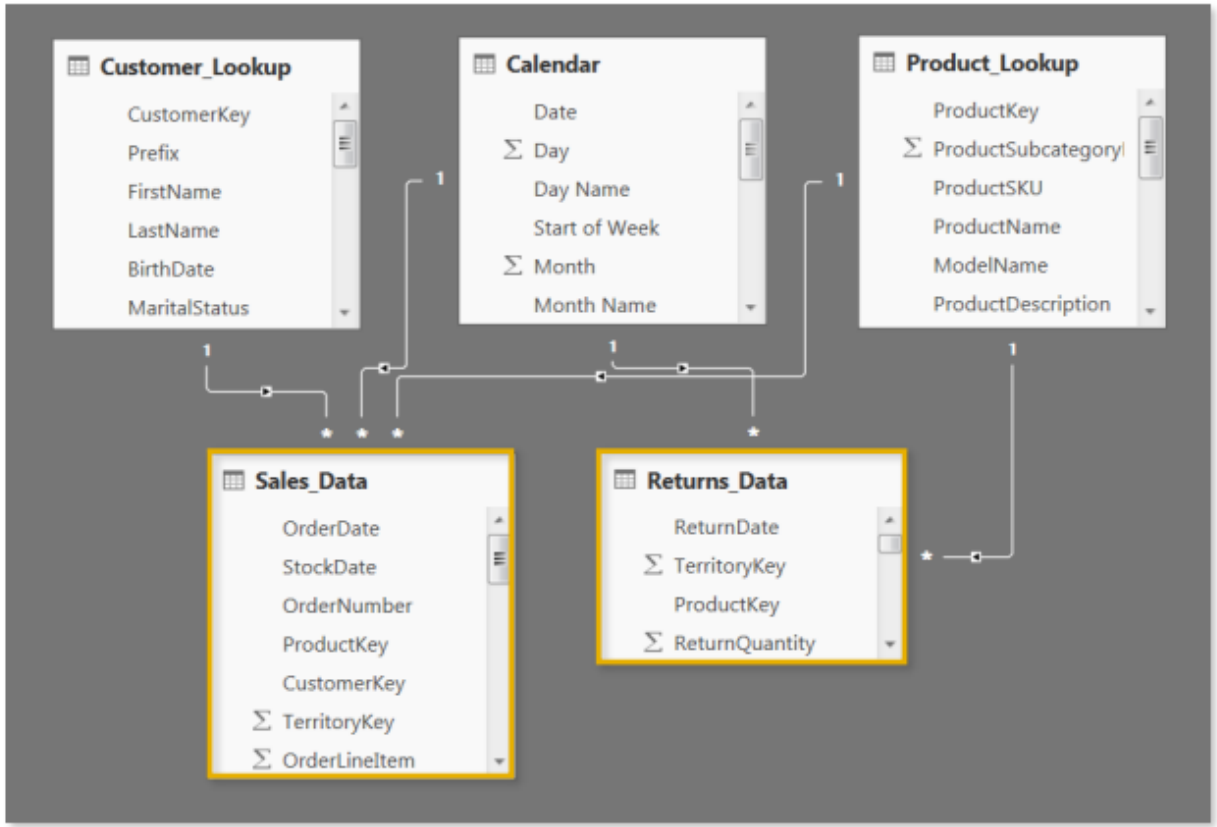
- Connecting the two tables above using the **product_id** field creates a ***one-to-one relationship***, since each ID only appears once in each table
- Unlike many-to-many, there is nothing *illegal* about this relationship; it's just **inefficient**

To eliminate the inefficiency, you could simply ***merge the two tables*** *into a single, valid lookup*

NOTE: *this still respects the laws of normalization, since all rows are unique and capture attributes related to the primary key*

product_id	product_name	product_sku	product_price
4	Washington Cream Soda	64412155747	$3.64
5	Washington Diet Soda	85561191439	$2.19
7	Washington Diet Cola	20191444754	$2.61
8	Washington Orange Juice	89770532250	$2.59

CONNECTING MULTIPLE DATA TABLES



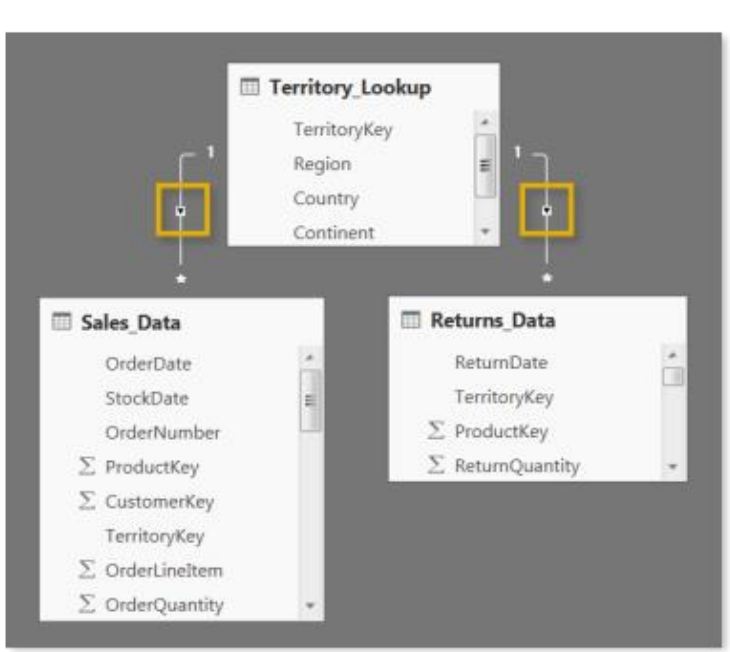
This model contains two data tables: **Sales_Data** and **Returns_Data**

- Note that the **Returns** table connects to **Calendar** and **Product_Lookup** just like the **Sales** table, but without a *CustomerKey* field it cannot be joined to **Customer_Lookup**
- This allows us to analyze sales and returns within the same view, ***but only if we filter or segment the data using shared lookups***
  - In other words, we know which ***product*** was returned and on which ***date***, but nothing about which ***customer*** made the return

HEY THIS IS IMPORTANT!

*In general, never create **direct relationships** between data tables; instead, **connect them through shared lookups***

FILTER FLOW



Here we have two data tables (**Sales_Data** and **Returns_Data**), connected to **Territory_Lookup**

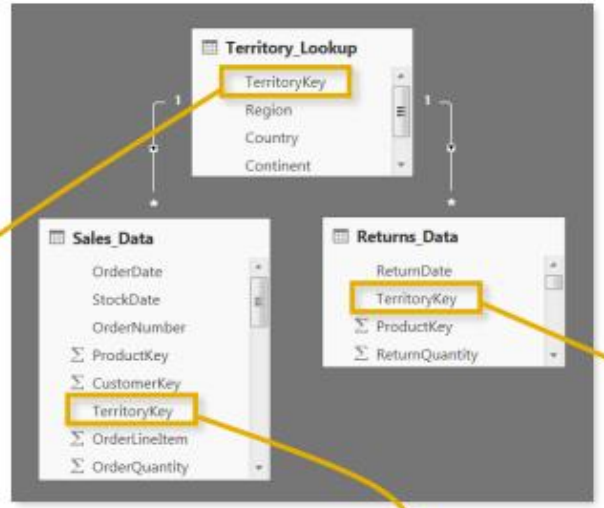
Note the filter directions (shown as arrows) in each relationship; by default, **these will point from the "one" side of the relationship** *(lookups)* **to the "many" side** *(data)*

- When you filter a table, that filter context is passed along to all related "*downstream*" tables (following the direction of the arrow)
- Filters **cannot** flow "*upstream*" (against the direction of the arrow)

PRO TIP:

*Arrange your lookup tables **above** your data tables in your model as a visual reminder that filters flow "**downstream**"*

FILTER FLOW (CONT.)



In this case, the only valid way filter both **Sales** and **Returns** data by Territory is to use the *TerritoryKey* field from the **Territory_Lookup** table, which is upstream and related to *both* data tables

- Filtering using *TerritoryKey* from the **Sales** table yields incorrect **Returns** values, since the filter context ***cannot flow upstream*** to either one of the other tables
- Similarly, filtering using *TerritoryKey* from the **Returns** table yields incorrect **Sales** data; in addition, ***only territories that registered returns are visible in the table*** (even though they registered sales)

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

1) *Filtering using TerritoryKey from the **Territory_Lookup** table*

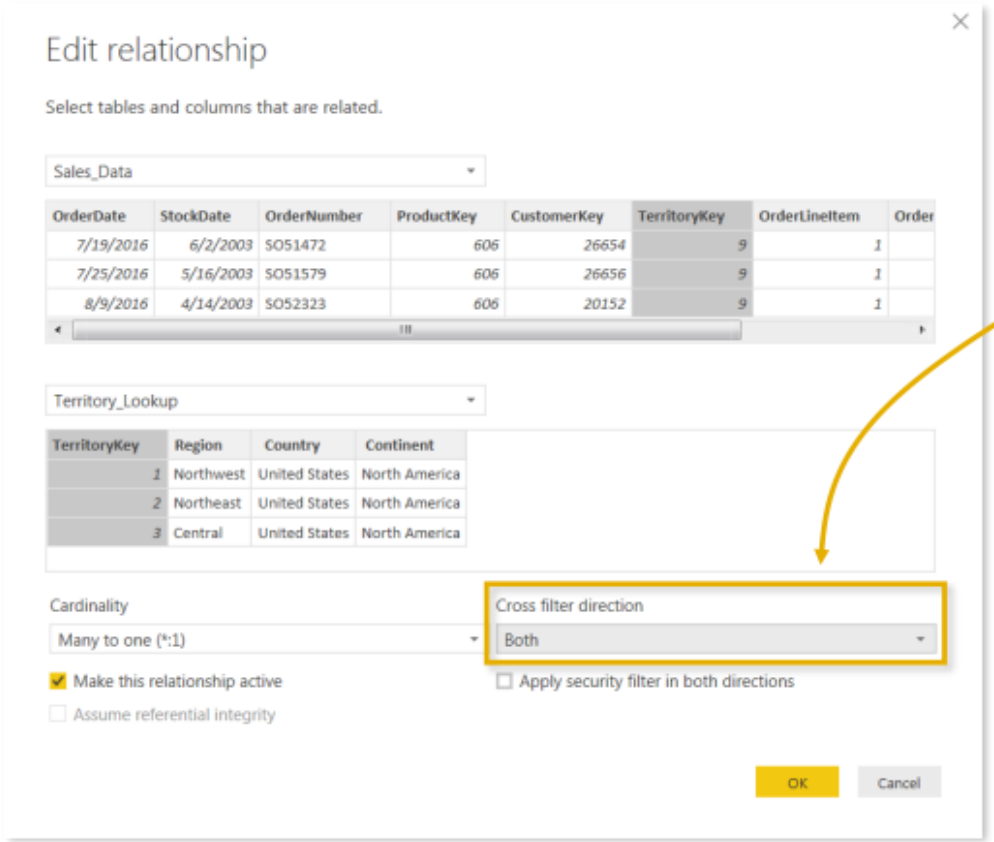
TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	1,828
2	40	1,828
3	30	1,828
4	17,191	1,828
5	49	1,828
6	894	1,828
7	7,862	1,828
8	7,950	1,828
9	17,951	1,828
10	9,694	1,828
Total	**84,174**	**1,828**

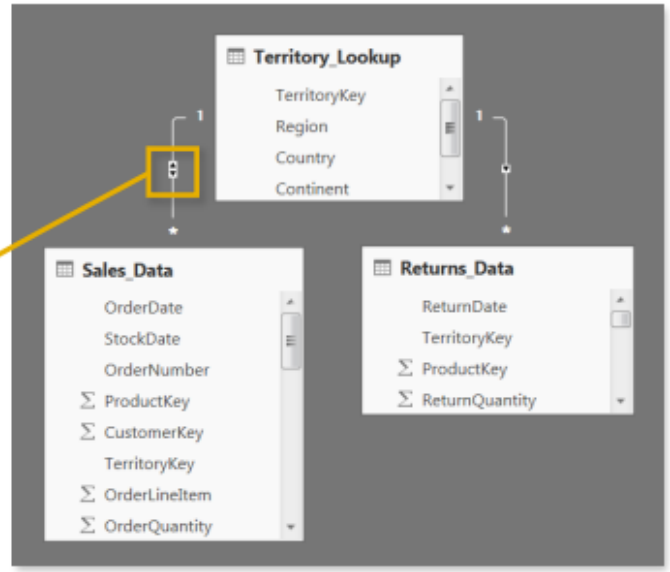
2) *Filtering using TerritoryKey from the **Sales_Data** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	84,174	270
4	84,174	362
5	84,174	1
6	84,174	238
7	84,174	186
8	84,174	163
9	84,174	404
10	84,174	204
Total	**84,174**	**1,828**

3) *Filtering using TerritoryKey from the **Returns_Data** table*

TWO-WAY FILTERS



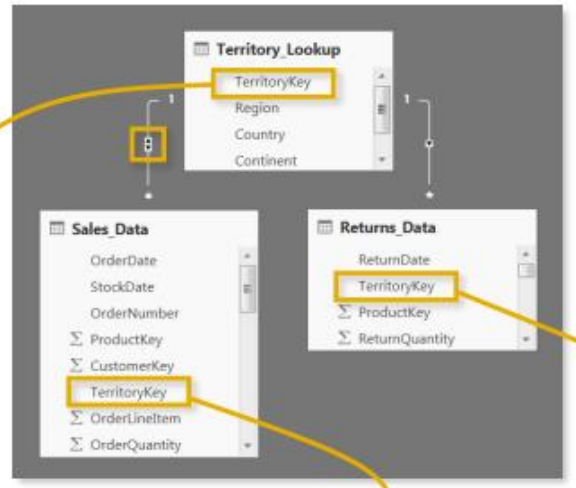


Updating the filter direction between **Sales** and **Territory** from "*Single*" to "*Both*" allows filter context to flow both ways

- This means that filters applied to the **Sales_Data** table will pass to the lookup, and then down to the **Returns_Data** table

NOTE: *The "Apply security filter in both directions" option relates to row-level security (RLS) settings, which are not covered in this course*

TWO-WAY FILTERS (CONT.)



With two-way cross-filtering enabled between the **Sales** and **Territory** tables, we now see correct values using *TerritoryKey* from either table

- The filter context for **Sales_Data[TerritoryKey]** now passes *up* to the **Territory_Lookup**, and then *down* to the **Returns_Data** table
- Note that we still see incorrect values when filtering using *TerritoryKey* from the **Returns** table, since the filter context is isolated to that single table

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

*1) Filtering using TerritoryKey from the **Territory_Lookup** table*

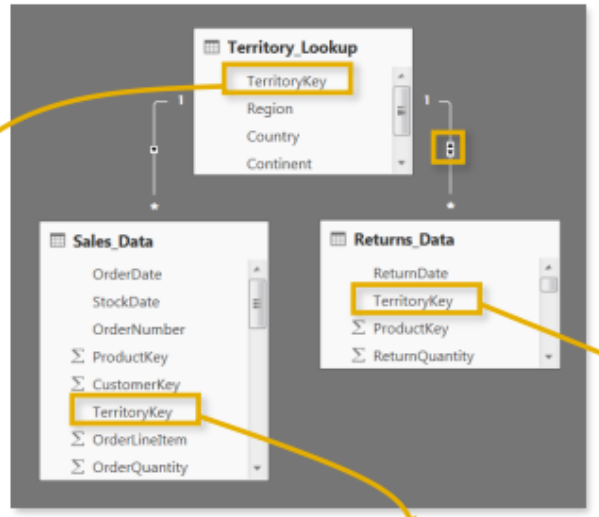
TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

*2) Filtering using TerritoryKey from the **Sales_Data** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	84,174	270
4	84,174	362
5	84,174	1
6	84,174	238
7	84,174	186
8	84,174	163
9	84,174	404
10	84,174	204
Total	**84,174**	**1,828**

*3) Filtering using TerritoryKey from the **Returns_Data** table*

TWO-WAY FILTERS (CONT.)



In this case, we've enabled two-way cross-filtering between the **Returns** and **Territory** tables

- As expected, we now see incorrect values when filtering using *TerritoryKey* from the **Sales** table, since the filter context is isolated to that single table
- While the values *appear* to be correct when filtering using *TerritoryKey* from the **Returns** table, we're ***missing sales data*** from any territories that didn't register returns (*specifically Territories **2** & **3***)

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
2	40	
3	30	
4	17,191	362
5	49	1
6	10,894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

*1) Filtering using TerritoryKey from the **Territory_Lookup** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	1,828
2	40	1,828
3	30	1,828
4	17,191	1,828
5	49	1,828
6	10,894	1,828
7	7,862	1,828
8	7,950	1,828
9	17,951	1,828
10	9,694	1,828
Total	**84,174**	**1,828**

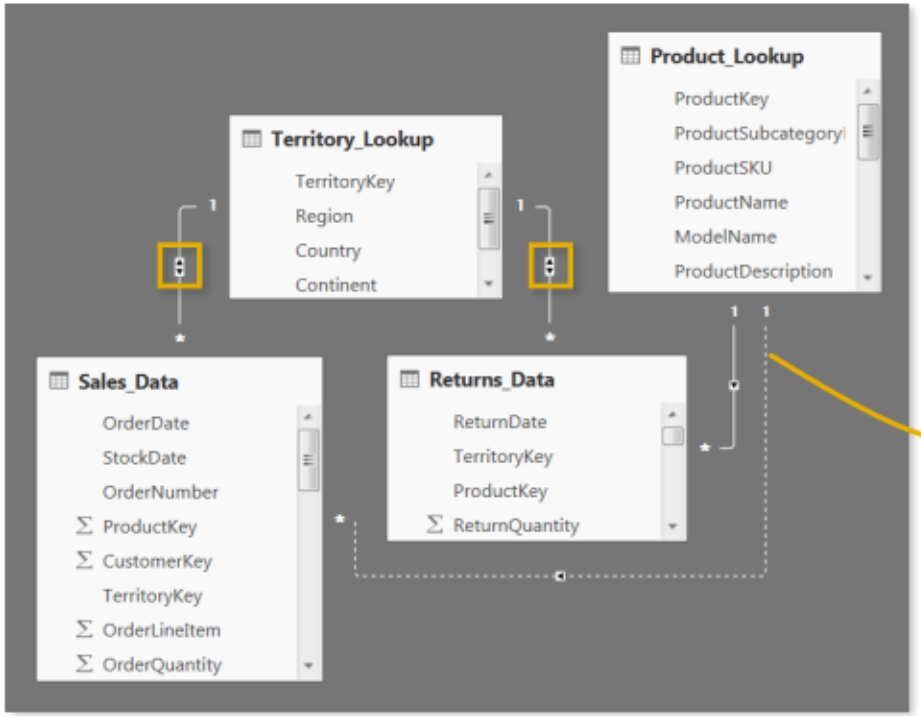
*2) Filtering using TerritoryKey from the **Sales_Data** table*

TerritoryKey	OrderQuantity	ReturnQuantity
1	12,513	270
4	17,191	362
5	49	1
6	10,894	238
7	7,862	186
8	7,950	163
9	17,951	404
10	9,694	204
Total	**84,174**	**1,828**

*3) Filtering using TerritoryKey from the **Returns_Data** table*

*Since no information about Territory 2 or 3 is passed from the **Returns_Data** table to **Territory_Lookup**, they get filtered out of the lookup, and subsequently filtered out of the **Sales_Data***

TWO-WAY FILTERS: **A WORD OF WARNING**



Use two-way filters carefully, and **only when necessary***

- If you try to use multiple two-way filters in a more complex model, you run the risk of creating "***ambiguous relationships***" by introducing multiple filter paths between tables:

> You can't create a direct active relationship between Sales_Data and Product_Lookup because that would introduce ambiguity between tables Product_Lookup and Territory_Lookup. To make this relationship active, deactivate or delete one of the relationships between Product_Lookup and Territory_Lookup first.

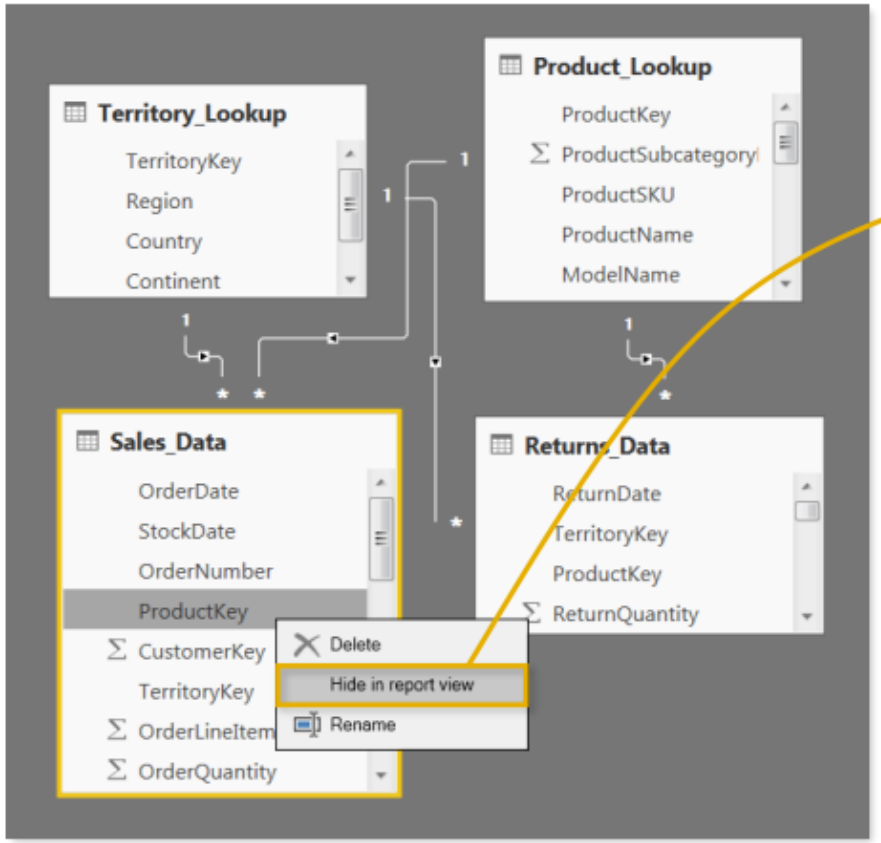
*In this model, filter context from the **Product_Lookup** table can pass down to **Returns_Data** and up to **Territory_Lookup**, which would filter accordingly based on the TerritoryKey values passed from the Returns table*

*If we were able to activate the relationship between **Product_Lookup** and **Sales_Data** as well, filters could pass from the **Product_Lookup** table through EITHER the Sales or Returns table to reach the **Territory_Lookup**, which could yield conflicting filter context*

PRO TIP:

*Design your models with **one-way filters** and **1-to-Many cardinality**, unless more complex relationships are necessary*

HIDING FIELDS FROM REPORT VIEW



Hiding fields from Report View makes them inaccessible from the Report tab (*although they can still be accessed within the **Data** or **Relationships** views*)

This is commonly used to prevent users from filtering using invalid fields, or to hide irrelevant metrics from view

PRO TIP:

*Hide the **foreign key columns** in your data tables to force users to filter using the **primary keys** in the lookup tables*

მადლობა ყურადღებისთვის

ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

მონაცემთა ანალიზი DAXის გამოყენებით;
ამოთვლილი სვეტები VS DAX კალკულაციები

Implicit vs. Explicit DAX კალკულაციები;

DAX სინტაქსი და ოპერატორები;
DAX ფუნქციების საერთო კატეგორიები;
ძირითადი თარიღი და დროის ფუნქციები;

პირობითი და ლოგიკური ფუნქციები (IF / AND / OR); ტექსტის საერთო ფუნქციები

მონაცემთა შეერთება (RELATED); ძირითადი მათემატიკისა და სტატისტიკის ფუნქციები

MEET **DAX**

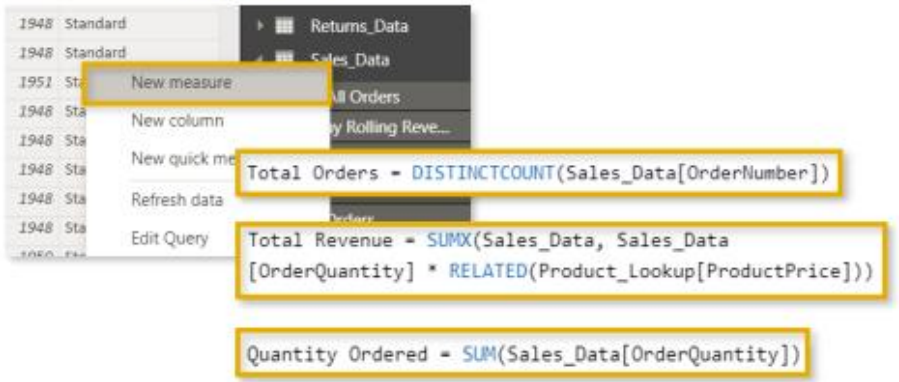
Data Analysis Expressions, commonly known as **DAX**, is the formula language that drives Power BI. With DAX, you can:

- Add ***calculated columns*** and ***measures*** to your model, using intuitive syntax
- Go beyond the capabilities of traditional "grid-style" formulas, with powerful and flexible functions built specifically to work with relational data models

Two ways to use DAX

1) *Calculated Columns*

2) *Measures*



CALCULATED COLUMNS

Calculated columns allow you to add new, formula-based columns to tables

- No "A1-style" references; calculated columns refer to **entire tables** or **columns**
- Calculated columns generate values for each row, which are **visible within tables in the Data view**
- Calculated columns understand **row context**; they're great for defining properties based on information in each row, but generally useless for aggregation (*SUM, COUNT, etc*)

HEY THIS IS IMPORTANT!

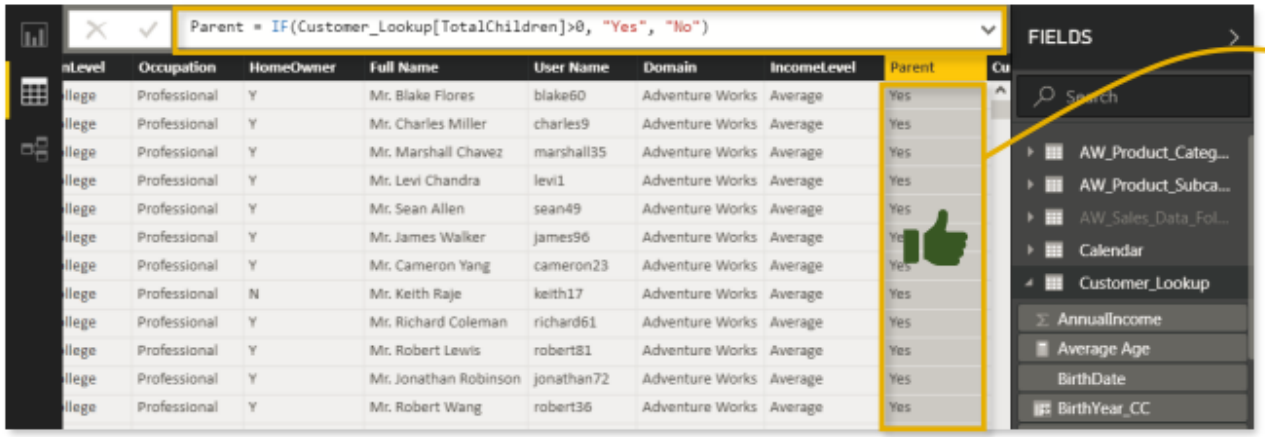
As a rule of thumb, use calculated columns when you want to "stamp" static, fixed values to each row in a table (*or use the Query Editor!*)

DO NOT use calculated columns for aggregation formulas, or to calculate fields for the "Values" area of a visualization (use **measures** instead)

PRO TIP:

*Calculated columns are typically used for **filtering** data, rather than creating numerical values*

CALCULATED COLUMNS (EXAMPLES)



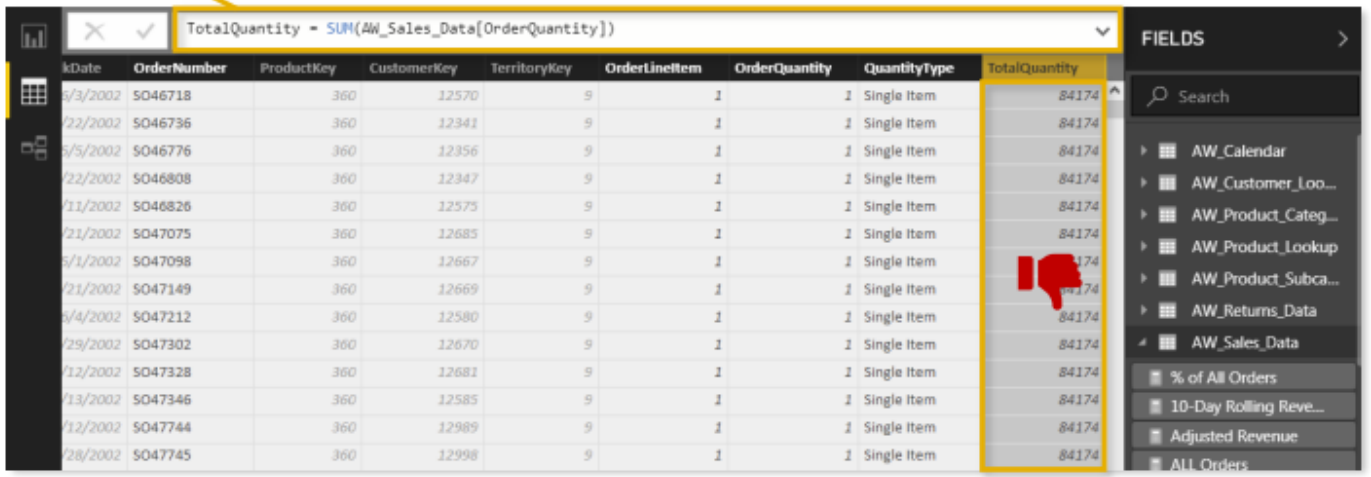
nLevel	Occupation	HomeOwner	Full Name	User Name	Domain	IncomeLevel	Parent
llege	Professional	Y	Mr. Blake Flores	blake60	Adventure Works	Average	Yes
llege	Professional	Y	Mr. Charles Miller	charles9	Adventure Works	Average	Yes
llege	Professional	Y	Mr. Marshall Chavez	marshall35	Adventure Works	Average	Yes
llege	Professional	Y	Mr. Levi Chandra	levi1	Adventure Works	Average	Yes
llege	Professional	Y	Mr. Sean Allen	sean49	Adventure Works	Average	Yes
llege	Professional	Y	Mr. James Walker	james96	Adventure Works	Average	Ye
llege	Professional	Y	Mr. Cameron Yang	cameron23	Adventure Works	Average	Yes
llege	Professional	N	Mr. Keith Raje	keith17	Adventure Works	Average	Yes
llege	Professional	Y	Mr. Richard Coleman	richard61	Adventure Works	Average	Yes
llege	Professional	Y	Mr. Robert Lewis	robert81	Adventure Works	Average	Yes
llege	Professional	Y	Mr. Jonathan Robinson	jonathan72	Adventure Works	Average	Yes
llege	Professional	Y	Mr. Robert Wang	robert36	Adventure Works	Average	Yes

In this case we've added a **calculated column** named **"Parent"**, which equals "***Yes***" if the [TotalChildren] field is greater than 0, and "***No***" otherwise (*just like Excel!*)

- Since calculated columns understand **row context**, a new value is calculated in each row based on the value in the [TotalChildren] column
- This is a **valid use** of calculated columns; it creates a new row "property" that we can now use to filter or segment any related data within the model

Here we're using an aggregation function (SUM) to calculate a new column named **TotalQuantity**

- Since calculated columns do not understand **filter context**, the same grand total is returned in *every single row* of the table
- This is **not a valid use** of calculated columns; these values are statically "stamped" onto the table and can't be filtered, sliced, subdivided, etc.



kDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity	QuantityType	TotalQuantity
/3/2002	SO46718	360	12570	9	1	1	Single Item	84174
/22/2002	SO46736	360	12341	9	1	1	Single Item	84174
/5/2002	SO46776	360	12356	9	1	1	Single Item	84174
/22/2002	SO46808	360	12347	9	1	1	Single Item	84174
/11/2002	SO46826	360	12575	9	1	1	Single Item	84174
/21/2002	SO47075	360	12685	9	1	1	Single Item	84174
/1/2002	SO47098	360	12667	9	1	1	Single Item	84174
/21/2002	SO47149	360	12669	9	1	1	Single Item	84174
/4/2002	SO47212	360	12580	9	1	1	Single Item	84174
/29/2002	SO47302	360	12670	9	1	1	Single Item	84174
/12/2002	SO47328	360	12681	9	1	1	Single Item	84174
/13/2002	SO47346	360	12585	9	1	1	Single Item	84174
/12/2002	SO47744	360	12989	9	1	1	Single Item	84174
/28/2002	SO47745	360	12998	9	1	1	Single Item	84174

MEASURES

Measures are DAX formulas used to generate new calculated values

- Like calculated columns, measures reference **entire tables** or **columns** (*no A1-style or "grid" references*)
- *Unlike* calculated columns, **measure** values aren't visible within tables; they can only be "*seen*" within a visualization like a chart or matrix (*similar to a calculated field in an Excel pivot*)
- Measures are evaluated based on **filter context**, which means they recalculate when the fields or filters around them change (*like when new row or column labels are pulled into a matrix or when new filters are applied to a report*)

HEY THIS IS IMPORTANT!

As a rule of thumb, use measures (*vs. calculated columns*) when a single row can't give you the answer (*in other words, when you need to **aggregate***)

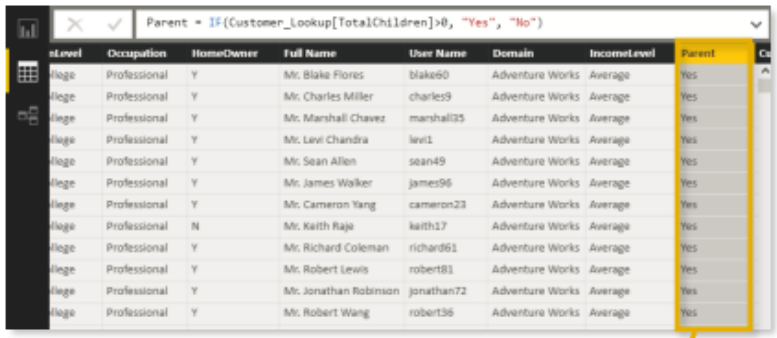
PRO TIP:

*Use measures to create **numerical, calculated values** that can be analyzed in the "**values**" field of a report visual*

RECAP: CALCULATED COLUMNS VS. MEASURES

CALCULATED COLUMNS

- Values are calculated based on information from each row of a table (has **row context**)
- Appends static values to each row in a table and stores them in the model (*which increases file size*)
- Recalculate on data source refresh or when changes are made to component columns
- Primarily used as **rows**, **columns**, **slicers** or **filters**



*Calculated columns "live" in **tables***

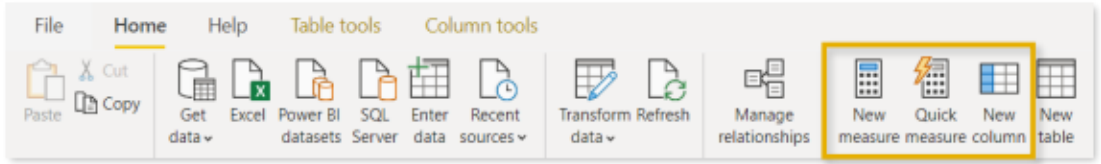
MEASURES

- Values are calculated based on information from any filters in the report (has **filter context**)
- Does not create new data in the tables themselves (*doesn't increase file size*)
- Recalculate in response to any change to filters within the report
- Almost *always* used within the **values** field of a visual

*Measures "live" in **visuals***

ADDING COLUMNS & MEASURES

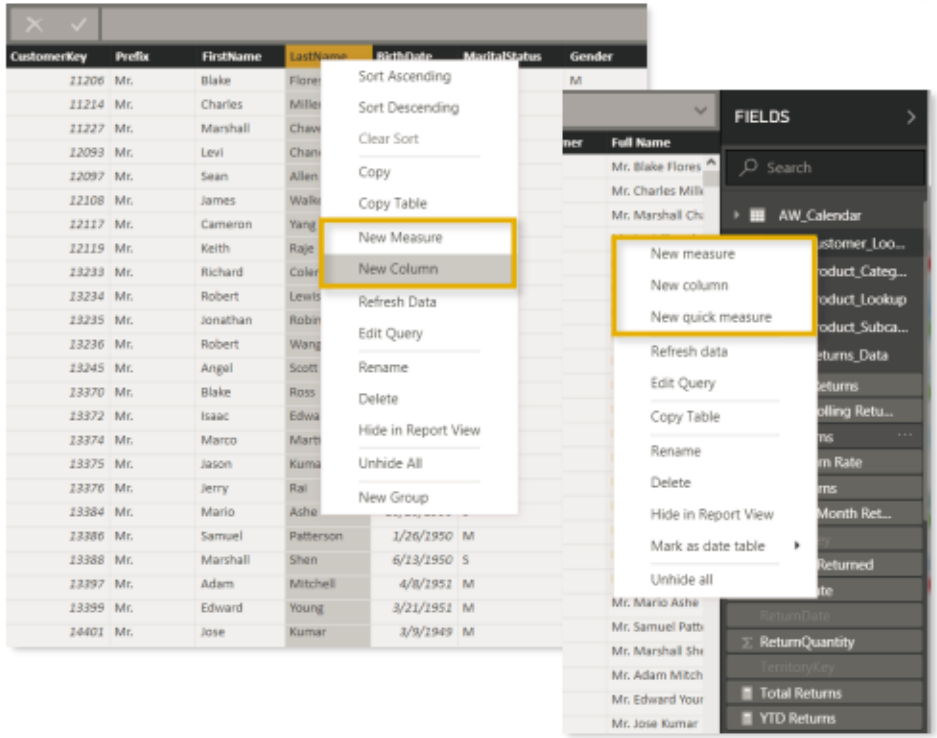
Option 1: Select "**New Measure**" or "**New Column**" from the **Home** tab*



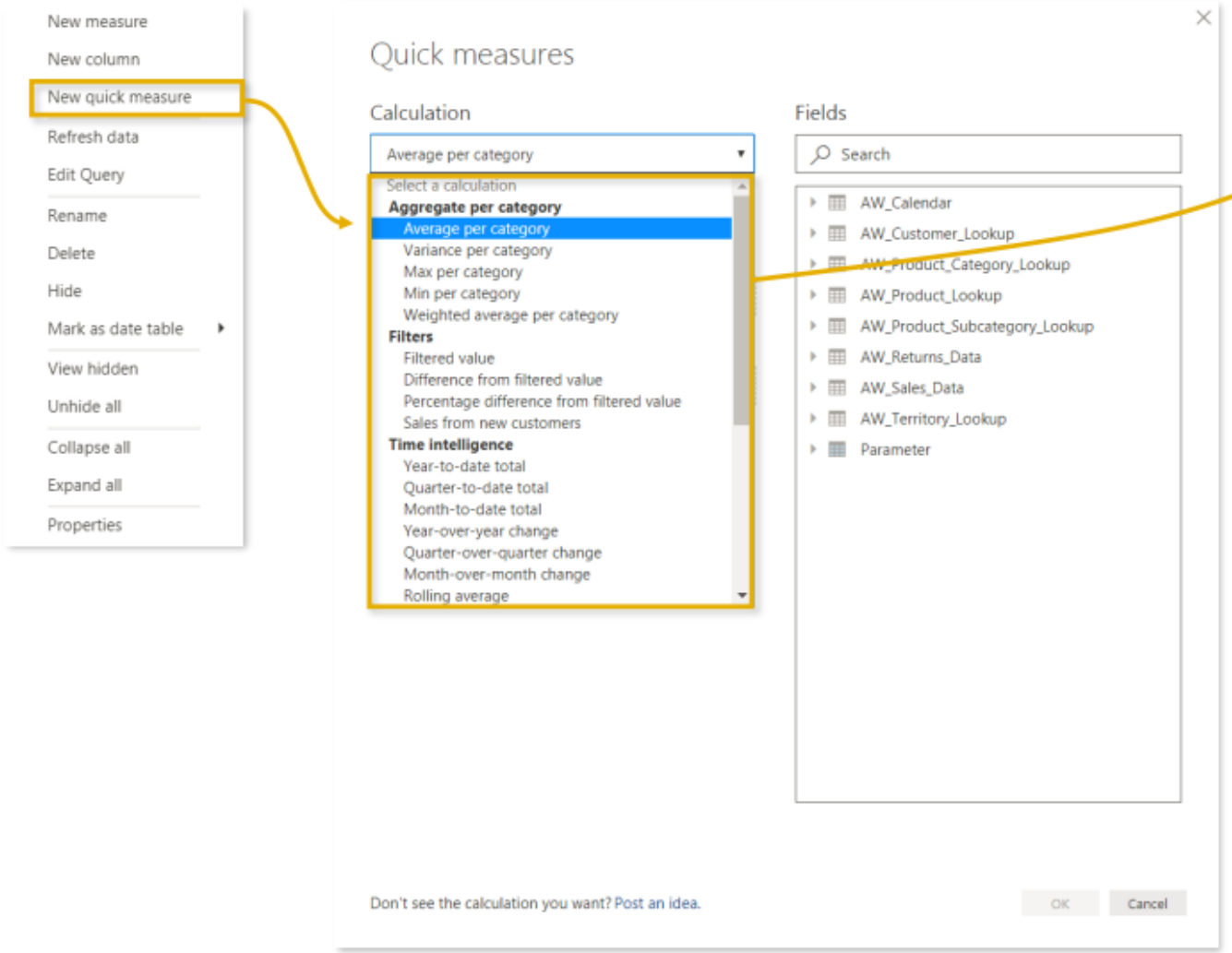
When you insert Columns or Measures using the **Home** tab (Option 1), they are assigned to whichever table is *currently selected*, or the *first table in the field list* by default

- Measures can be reassigned to new "Home" tables (under the "*Structure*" options in the contextual **Measure Tools** tab), but Option 2 allows you to be more deliberate about placing them
- **NOTE:** Assigning measures to specific tables doesn't have ANY impact on functionality – it's just a way to keep them organized

Option 2: Right-click within the **table** (in the **Data** view) or the **Field List** (in either the **Data** or **Report** view)



QUICK MEASURES



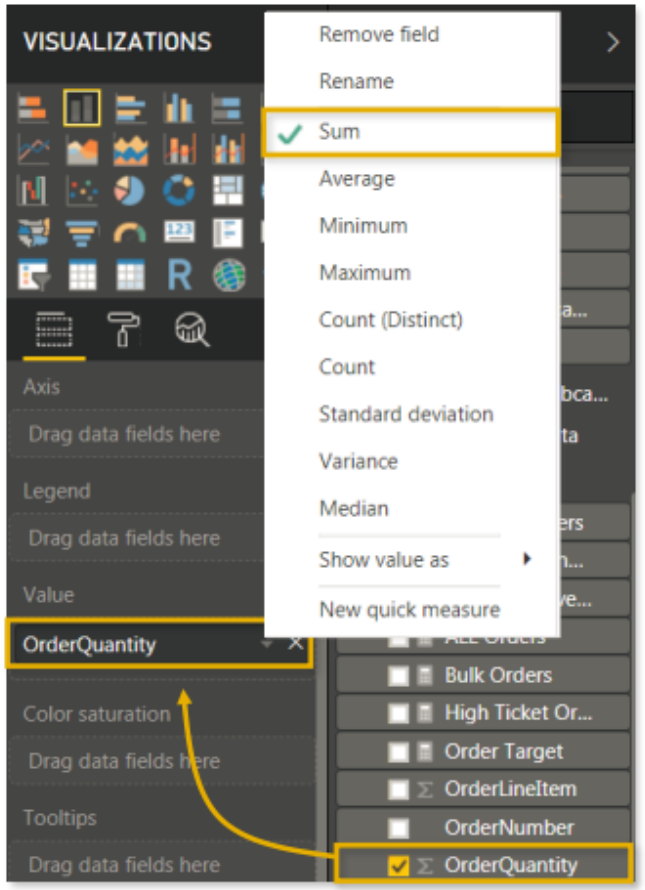
Quick Measures are pre-built formula templates that allow you to drag and drop fields, rather than write DAX from scratch

While these tools can be helpful for defining more complex measures (*like weighted averages or time intelligence formulas*), they encourage laziness and don't help you understand the fundamentals of DAX

PRO TIP:

*Just say "**NO**" to quick measures*
(you're better than that)

IMPLICIT VS. EXPLICIT MEASURES



*Example of an **implicit measure***

Implicit measures are created when you drag raw numerical fields (like "*OrderQuantity*") into the values pane of a visual and manually select the aggregation mode (*Sum, Average, Min/Max, etc*)

Explicit measures are created by actually entering DAX functions (or adding "*quick measures*") to define calculated columns or measures

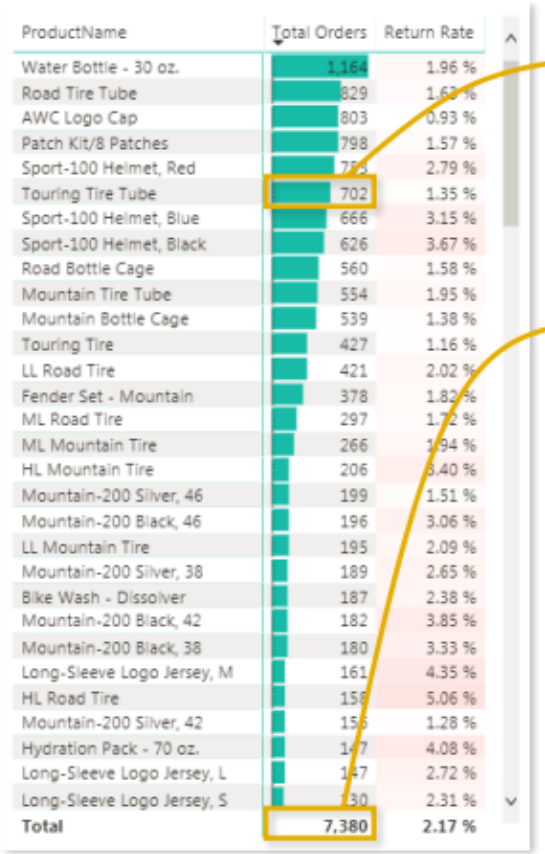
HEY THIS IS IMPORTANT!

Implicit measures are *only accessible* within the specific visualization in which it was created, and cannot be referenced elsewhere

Explicit measures can be used anywhere in the report, and referenced within other DAX calculations to create "measure trees"

UNDERSTANDING FILTER CONTEXT

Remember that measures are evaluated based on **filter context**, which means that they recalculate whenever the fields or filters around them change



ProductName	Total Orders	Return Rate
Water Bottle - 30 oz.	1,164	1.96 %
Road Tire Tube	829	1.63 %
AWC Logo Cap	803	0.93 %
Patch Kit/8 Patches	798	1.57 %
Sport-100 Helmet, Red	7[illegible]	2.79 %
Touring Tire Tube	702	1.35 %
Sport-100 Helmet, Blue	666	3.15 %
Sport-100 Helmet, Black	626	3.67 %
Road Bottle Cage	560	1.58 %
Mountain Tire Tube	554	1.95 %
Mountain Bottle Cage	539	1.38 %
Touring Tire	427	1.16 %
LL Road Tire	421	2.02 %
Fender Set - Mountain	378	1.82 %
ML Road Tire	297	1.72 %
ML Mountain Tire	266	1.94 %
HL Mountain Tire	206	3.40 %
Mountain-200 Silver, 46	199	1.51 %
Mountain-200 Black, 46	196	3.06 %
LL Mountain Tire	195	2.09 %
Mountain-200 Silver, 38	189	2.65 %
Bike Wash - Dissolver	187	2.38 %
Mountain-200 Black, 42	182	3.85 %
Mountain-200 Black, 38	180	3.33 %
Long-Sleeve Logo Jersey, M	161	4.35 %
HL Road Tire	158	5.06 %
Mountain-200 Silver, 42	156	1.28 %
Hydration Pack - 70 oz.	147	4.08 %
Long-Sleeve Logo Jersey, L	147	2.72 %
Long-Sleeve Logo Jersey, S	130	2.31 %
Total	**7,380**	**2.17 %**

For this particular value in the matrix, the **Total Orders** measure is calculated based on the following filter context: *Products[**ProductName**] = "**Touring Tire Tube**"*

- This allows the measure to return the total order quantity for each product specifically (or whatever the row and column labels dictate – *years, countries, product categories, customer names,* etc)

This Total is ***not*** calculated by summing the values above; it evaluates as its own measure, with **no filter context** (*since we aren't calculating orders for a specific product*)

HEY THIS IS IMPORTANT!

Each measure value in a report is ***like an island***, and calculates according to it's own filter context (*even Totals and Grand Totals*)

FILTER CONTEXT (EXAMPLES)

MEASURE: Total Revenue

FILTER CONTEXT:

- *Calendar[**Year**]* = 2016 *or* 2017
- *Customers[**Full Name**]* = Mr. Larry Munoz

MEASURE: Total Orders

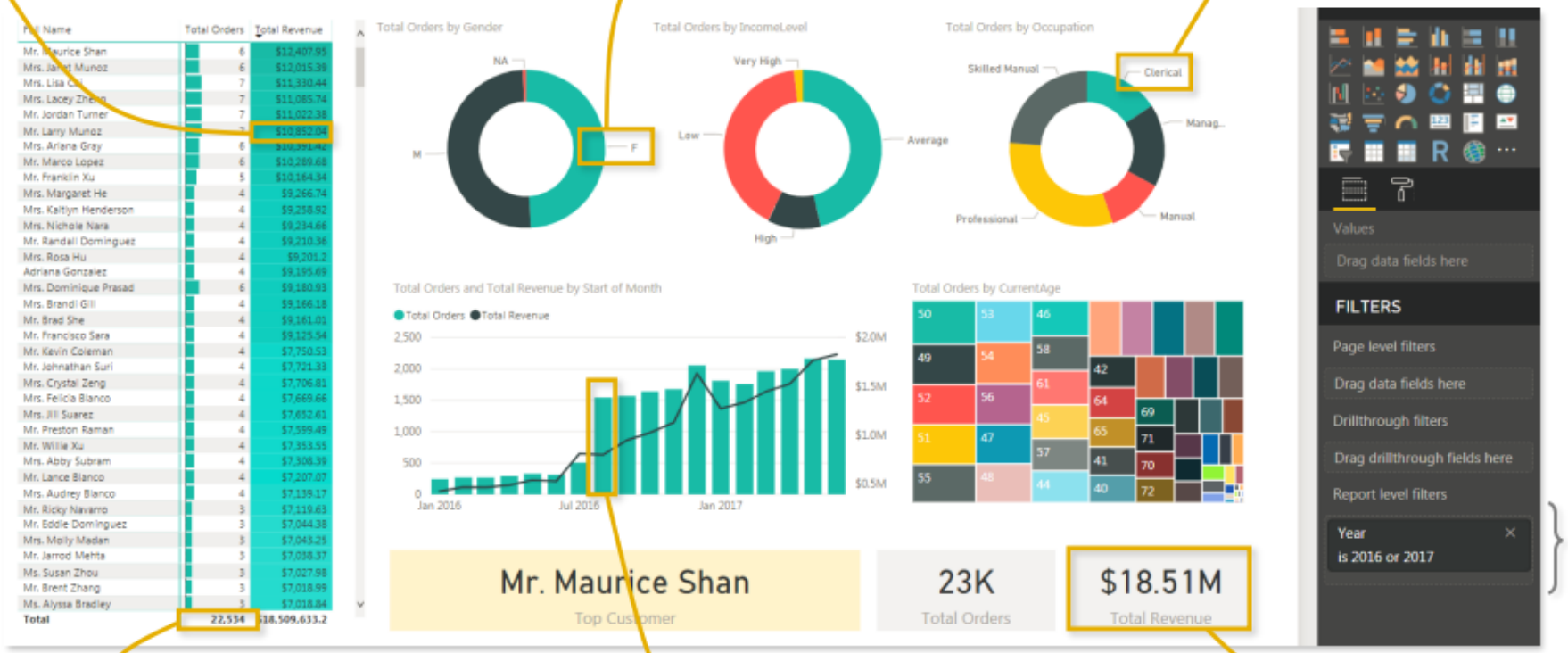
FILTER CONTEXT:

- *Calendar[**Year**]* = 2016 *or* 2017
- *Customers[**Gender**]* = F (Female)

MEASURE: Total Orders

FILTER CONTEXT:

- *Calendar[**Year**]* = 2016 *or* 2017
- *Customers[**Occupation**]* = Clerical



*This is **a page-level filter**, which impact **ALL** visuals on the report page*

MEASURE: Total Orders

FILTER CONTEXT:

- *Calendar[**Year**]* = 2016 *or* 2017

MEASURE: Total Orders

FILTER CONTEXT:

- *Calendar[**Year**]* = 2016 *or* 2017
- *Calendar[**Month**]* = August 2016

MEASURE: Total Revenue

FILTER CONTEXT:

- *Calendar[**Year**]* = 2016 *or* 2017

STEP-BY-STEP MEASURE CALCULATION

CategoryName	Total Returns
Accessories	1,115
Bikes	342
Clothing	267

How *exactly* is this measure calculated?

- **REMEMBER:** This all happens *instantly* behind the scenes, every time the filter context changes

STEP 1

Filter context is detected & applied

CategoryName	Total Returns
Accessories	1,115
Bikes	342
Clothing	267

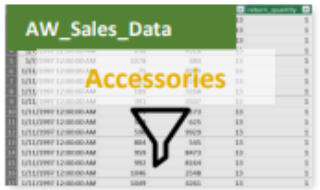
Product[**CategoryName**] = "Accessories"

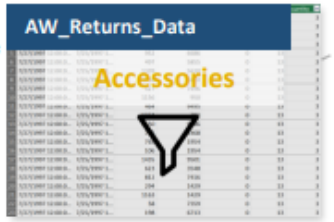
STEP 2

Filters flow "downstream" to all related tables

1 1

* *





STEP 3

Measure formula evaluates against the filtered table

```
Total Returns = COUNTROWS(AW_Returns_Data)
```

*Count of rows in the **AW_Returns_Data** table, filtered down to only rows where the product category is "Accessories"* = **1,115**

DAX SYNTAX

MEASURE NAME

- ***Note:*** *Measures are always surrounded in brackets (i.e.* ***[Total Quantity]****) when referenced in formulas, so spaces are OK*

Referenced **TABLE NAME**

Referenced **COLUMN NAME**

Total Quantity: =SUM(Transactions[quantity])

FUNCTION NAME

- Calculated columns don't always use functions, but measures do:
  - In a **Calculated Column**, =Transactions[quantity] returns the value from the quantity column in each row (*since it evaluates one row at a time*)
  - In a **Measure**, =Transactions[quantity] will return an ***error*** since Power BI doesn't know how to translate that as a single value (*you need some sort of aggregation*)

Note: *This is a* ***"fully qualified"*** *column, since it's preceeded by the table name -- table names with spaces must be surrounded by* ***single quotes****:*

- *Without a space:* ***Transactions****[quantity]*
- *With a space:* ***'Transactions Table'****[quantity]*

PRO TIP:

For ***column*** *references, use the fully qualified name (i.e.* ***Table[Column]****)*
For ***measure*** *references, just use the measure name (i.e.* ***[Measure]****)*

DAX OPERATORS

Arithmetic Operator	Meaning	Example
+	Addition	2 + 7
-	Subtraction	5 – 3
*	Multiplication	2 * 6
/	Division	4 / 2
^	Exponent	2 ^ 5

Comparison Operator	Meaning	Example
=	Equal to	[City]="Boston"
>	Greater than	[Quantity]>10
<	Less than	[Quantity]<10
>=	Greater than or equal to	[Unit_Price]>=2.5
<=	Less than or equal to	[Unit_Price]<=2.5
<>	Not equal to	[Country]<>"Mexico"

Pay attention to these!

Text/Logical Operator	Meaning	Example
&	Concatenates two values to produce one text string	[City] & " " & [State]
&&	Create an AND condition between two logical expressions	([State]="MA") && ([Quantity]>10)
|| (double pipe)	Create an OR condition between two logical expressions	([State]="MA") || ([State]="CT")
IN	Creates a logical OR condition based on a given list (using curly brackets)	'Store Lookup'[State] IN { "MA", "CT", "NY" }

COMMON FUNCTION CATEGORIES

MATH & STATS Functions

*Basic **aggregation** functions as well as "**iterators**" evaluated at the row-level*

Common Examples:

- SUM
- AVERAGE
- MAX/MIN
- DIVIDE
- COUNT/COUNTA
- COUNTROWS
- DISTINCTCOUNT

Iterator Functions:

- SUMX
- AVERAGEX
- MAXX/MINX
- RANKX
- COUNTX

LOGICAL Functions

*Functions for returning information about values in a given **conditional expression***

Common Examples:

- IF
- IFERROR
- AND
- OR
- NOT
- SWITCH
- TRUE
- FALSE

TEXT Functions

*Functions to manipulate **text strings** or **control formats** for dates, times or numbers*

Common Examples:

- CONCATENATE
- FORMAT
- LEFT/MID/RIGHT
- UPPER/LOWER
- PROPER
- LEN
- SEARCH/FIND
- REPLACE
- REPT
- SUBSTITUTE
- TRIM
- UNICHAR

FILTER Functions

***Lookup** functions based on related tables and **filtering** functions for dynamic calculations*

Common Examples:

- CALCULATE
- FILTER
- ALL
- ALLEXCEPT
- RELATED
- RELATEDTABLE
- DISTINCT
- VALUES
- EARLIER/EARLIEST
- HASONEVALUE
- HASONEFILTER
- ISFILTERED
- USERELATIONSHIP

DATE & TIME Functions

*Basic **date and time** functions as well as advanced **time intelligence** operations*

Common Examples:

- DATEDIFF
- YEARFRAC
- YEAR/MONTH/DAY
- HOUR/MINUTE/SECOND
- TODAY/NOW
- WEEKDAY/WEEKNUM

Time Intelligence Functions:

- DATESYTD
- DATESQTD
- DATESMTD
- DATEADD
- DATESINPERIOD

BASIC DATE & TIME FUNCTIONS

DAY/MONTH/ YEAR()	*Returns the day of the month (1-31), month of the year (1-12), or year of a given date*	=**DAY/MONTH/YEAR**(Date)
HOUR/MINUTE/ SECOND()	*Returns the hour (0-23), minute (0-59), or second (0-59) of a given datetime value*	=**HOUR/MINUTE/SECOND**(Datetime)
TODAY/NOW()	*Returns the current date or exact time*	=**TODAY/NOW**()
WEEKDAY/ WEEKNUM()	*Returns a weekday number from 1 (Sunday) to 7 (Saturday), or the week # of the year*	=**WEEKDAY/WEEKNUM**(Date, *[ReturnType]*)
EOMONTH()	*Returns the date of the last day of the month, +/- a specified number of months*	=**EOMONTH**(StartDate, Months)
DATEDIFF()	*Returns the difference between two dates, based on a selected interval*	=**DATEDIFF**(Date1, Date2, Interval)

BASIC LOGICAL FUNCTIONS (IF/AND/OR)

IF()	*Checks if a given condition is met, and returns one value if the condition is TRUE, and another if the condition is FALSE*	=IF(LogicalTest, ResultIfTrue, *[ResultIfFalse]*)
IFERROR()	*Evaluates an expression and returns a specified value if the expression returns an error, otherwise returns the expression itself*	=IFERROR(Value, ValueIfError)
AND()	*Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE, otherwise returns FALSE*	=AND(Logical1, Logical2)
OR()	*Checks whether one of the arguments is TRUE to return TRUE, and returns FALSE if both arguments are FALSE*	=OR(Logical1, Logical2)

Note: *Use the **&&** and **||** operators if you want to include more than two conditions!*

BASIC MATH & STATS FUNCTIONS

SUM()	*Evaluates the sum of a column*	=SUM(ColumnName)
AVERAGE()	*Returns the average (arithmetic mean) of all the numbers in a column*	=AVERAGE(ColumnName)
MAX()	*Returns the largest value in a column or between two scalar expressions*	=MAX(ColumnName) *or* =MAX(Scalar1, *[Scalar2]*)
MIN()	*Returns the smallest value in a column or between two scalar expressions*	=MIN(ColumnName) *or* =MIN(Scalar1, *[Scalar2]*)
DIVIDE()	*Performs division and returns the alternate result (or blank) if div/0*	=DIVIDE(Numerator, Denominator, *[AlternateResult]*)

TEXT FUNCTIONS

LEN()	*Returns the number of characters in a string*	=LEN(Text)
CONCATENATE()	*Joins two text strings into one*	=CONCATENATE(Text1, Text2)
LEFT/MID/ RIGHT()	*Returns a number of characters from the start/middle/end of a text string*	=LEFT/RIGHT(Text, *[NumChars]*) =MID(Text, StartPosition, NumChars)
UPPER/LOWER/ PROPER()	*Converts letters in a string to upper/lower/proper case*	=UPPER/LOWER/PROPER(Text)
SUBSTITUTE()	*Replaces an instance of existing text with new text in a string*	=SUBSTITUTE(Text, OldText, NewText, *[InstanceNumber]*)
SEARCH()	*Returns the position where a specified string or character is found, reading left to right*	=SEARCH(FindText, WithinText, *[StartPosition], [NotFoundValue]*)

Note: *Use the* ***&*** *operator as a shortcut, or to combine more than two strings!*

RELATED

RELATED() *Returns related values in each row of a table based on relationships with other tables*

=RELATED(ColumnName)

The column that contains the values you want to retrieve

Examples:

- *Product_Lookup[ProductName]*
- *Territory_Lookup[Country]*

HEY THIS IS IMPORTANT!

RELATED works almost *exactly* like a **VLOOKUP** function – it uses the relationship between tables (*defined by primary and foreign keys*) to pull values from one table into a new column of another

Since this function requires row context, it can only be used as a **calculated column** or as part of an **iterator function** that cycles through all rows in a table (*FILTER, SUMX, MAXX, etc*)

PRO TIP:

Avoid using RELATED to create redundant calculated columns unless you absolutely need them, since those extra columns increase file size; instead, use RELATED within a measure like FILTER or SUMX

მადლობა ყურადღებისთვის

ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

- COUNT ფუნქცია (COUNTA, DISTINCTCOUNT, COUNTROWS)
- CALCULATE ბრძანება; CALCULATE და ALL შეკვეთა
- CALCULATE და FILTER ბრძანება;
- Iterator ფუნქციები (SUMX, RANKX); Time intelligence ფორმულები; DAX-ის საუკეთესო პრაქტიკა)

BASIC MATH & STATS FUNCTIONS

SUM()	*Evaluates the sum of a column*	=SUM(ColumnName)
AVERAGE()	*Returns the average (arithmetic mean) of all the numbers in a column*	=AVERAGE(ColumnName)
MAX()	*Returns the largest value in a column or between two scalar expressions*	=MAX(ColumnName) *or* =MAX(Scalar1, *[Scalar2]*)
MIN()	*Returns the smallest value in a column or between two scalar expressions*	=MIN(ColumnName) *or* =MIN(Scalar1, *[Scalar2]*)
DIVIDE()	*Performs division and returns the alternate result (or blank) if div/0*	=DIVIDE(Numerator, Denominator, *[AlternateResult]*)

*Copyright 2018, Excel Maven & Maven Analytics, LLC

COUNT, COUNTA, DISTINCTCOUNT & COUNTROWS

COUNT()	*Counts the number of cells in a column that contain numbers*	=COUNT(ColumnName)
COUNTA()	*Counts the number of non-empty cells in a column (numerical and non-numerical)*	=COUNTA(ColumnName)
DISTINCTCOUNT()	*Counts the number of distinct or unique values in a column*	=DISTINCTCOUNT(ColumnName)
COUNTROWS()	*Counts the number of rows in the specified table, or a table defined by an expression*	=COUNTROWS(Table)

*Copyright 2018, Excel Maven & Maven Analytics, LLC

CALCULATE

CALCULATE() — *Evaluates a given expression or formula under a set of defined filters*

=CALCULATE(Expression, *[Filter1], [Filter2],...*)

Name of an existing measure, or a DAX formula for a valid measure

Examples:

- *[Total Orders]*
- *SUM(Returns_Data[ReturnQuantity])*

*List of simple Boolean (True/False) filter expressions (**note:** these require simple, fixed values; you cannot create filters based on measures)*

Examples:

- *Territory_Lookup[Country] = "USA"*
- *Calendar[Year] > 1998*

PRO TIP:

*CALCULATE works just like **SUMIF** or **COUNTIF** in Excel, except it can evaluate measures based on ANY sort of calculation (not just a sum, count, etc); it may help to think of it like **"CALCULATEIF"***

*Copyright 2018, Excel Maven & Maven Analytics, LLC

CALCULATE (EXAMPLE)

```
Bike Returns = CALCULATE([Total Returns], Products[CategoryName] = "Bikes")
```

CategoryName	Total Returns	Bike Returns
Accessories	1,116	342
Bikes	342	342
Clothing	267	342
Components		342
Total	1,724	342

Here we've defined a new measure named "**Bike Returns**", which evaluates the "**Total Returns**" measure when the *CategoryName* in the **Products** table equals "**Bikes**"

Wait, why do we see the same repeating values when we view a matrix with different categories on rows?

Shouldn't these cells have different filter contexts for Accessories, Clothing, Components, etc?

HEY THIS IS IMPORTANT!

CALCULATE ***modifies*** and ***overrules*** any competing filter context!

In this example, the "Clothing" row has filter context of CategoryName = "**Clothing**" (*defined by the row label*) ***and*** CategoryName= "**Bikes**" (*defined by the CALCULATE function*)

Both cannot be true at the same time, so the "**Clothing**" filter is overwritten and the "**Bikes**" filter (from CALCULATE) takes priority

*Copyright 2018, Excel Maven & Maven Analytics, LLC

CALCULATE **CHANGES** THE FILTER CONTEXT

CALCULATE

Filters modified by CALCULATE

[CategoryName] = "Bikes"

If the measure being evaluated contains a **CALCULATE** function, filter context is *overwritten* between **Step 1** & **Step 2**

STEP 1

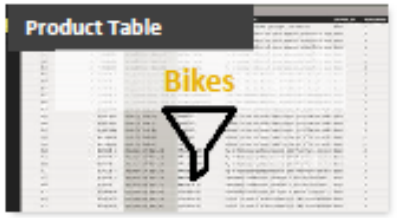
Filter context is detected & applied

CategoryName	Total Returns	Bike Returns
Accessories	1,115	342
Bikes	342	342
Clothing	267	142
Components		142
Total	1,724	342

Products[CategoryName] = "Accessories"

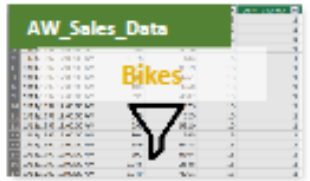
STEP 2

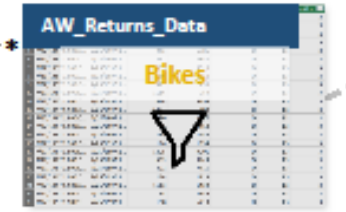
Filters flow "downstream" to all related tables



1 1

* *





STEP 3

Measure formula evaluates against the filtered table

*Count of rows in the **AW_Returns_Data** table, filtered down to only rows where the product category is "Bikes"* = **342**

*Copyright 2018, Excel Maven & Maven Analytics, LLC

ALL

ALL() *Returns all rows in a table, or all values in a column, ignoring any filters that have been applied*

=ALL(Table *or* ColumnName, *[ColumnName1], [ColumnName2],…*)

The table or column that you want to clear filters on

Examples:

- *Transactions*
- *Products[ProductCategory]*

List of columns that you want to clear filters on (optional)

Notes:

- *If your first parameter is a table, you can't specify additional columns*
- *All columns must include the table name, and come from the same table*

Examples:

- *Customer_Lookup[CustomerCity], Customer_Lookup[CustomerCountry]*
- *Products[ProductName]*

PRO TIP:

*Instead of adding filter context, ALL **removes it**. This is often used when you need unfiltered values that won't react to changes in filter context (i.e. **% of Total**, where the denominator needs to remain fixed)*

*Copyright 2018, Excel Maven & Maven Analytics, LLC

FILTER

FILTER() *Returns a table that represents a subset of another table or expression*

=FILTER(Table, FilterExpression)

Table to be filtered

Examples:

- *Territory_Lookup*
- *Customer_Lookup*

A Boolean (True/False) filter expression to be evaluated for each row of the table

Examples:

- *Territory_Lookup[Country] = "USA"*
- *Calendar[Year] = 1998*
- *Products[Price] > [Overall Avg Price]*

HEY THIS IS IMPORTANT!

FILTER is used to add new filter context, and can handle ***more complex filter expressions*** than CALCULATE (***by referencing measures, for example***)

Since FILTER returns an entire table, it's almost always used as an *input* to other functions, like CALCULATE or SUMX

PRO TIP:

Since FILTER iterates through each row in a table, it can be slow and processor-intensive; don't use FILTER if a CALCULATE function will accomplish the same thing

*Copyright 2018, Excel Maven & Maven Analytics, LLC

ITERATOR ("X") FUNCTIONS

Iterator (or "**X**") **functions** allow you to loop through the same calculation or expression on *each row of a table*, and then apply some sort of aggregation to the results (*SUM*, *MAX*, etc)

=**SUMX**(Table, Expression)

*Aggregation to apply to calculated rows**

Examples:

- *SUMX*
- *COUNTX*
- *AVERAGEX*
- *RANKX*
- *MAXX/MINX*

Table in which the expression will be evaluated

Examples:

- *Sales*
- *FILTER(Sales, RELATED(Products[Category])="Clothing")*

Expression to be evaluated for each row of the given table

Examples:

- *[Total Orders]*
- *Sales[RetailPrice] * Sales[Quantity]*

PRO TIP:

*Imagine the function **adding a temporary new column** to the table, calculating the value in each row (based on the expression) and then applying the aggregation to that new column (like SUMPRODUCT)*

**In this example we're looking at SUMX, but other "X" functions follow a similar syntax*

*Copyright 2018, Excel Maven & Maven Analytics, LLC

TIME INTELLIGENCE FORMULAS

Time Intelligence functions allow you to easily calculate common time comparisons:

Performance To-Date

```
=CALCULATE(Measure, DATESYTD(Calendar[Date]))
```

Use **DATESQTD** *for Quarters or* **DATESMTD** *for Months*

Previous Period

```
=CALCULATE(Measure, DATEADD(Calendar[Date], -1, MONTH))
```

*Select an interval (***DAY, MONTH, QUARTER,** *or* **YEAR***) and the # of intervals to compare (i.e. previous month, rolling 10-day)*

Running Total

```
=CALCULATE(Measure,
    DATESINPERIOD(Calendar[Date], MAX(Calendar[Date]), -10, DAY))
```

PRO TIP:

To calculate a ***moving average****, use the running total calculation above and divide by the number of intervals*

*Copyright 2018, Excel Maven & Maven Analytics, LLC

BEST PRACTICES: CALCULATED COLUMNS & MEASURES

Don't use a calculated column when a measure will do the trick

- *Only use calculated columns to "stamp" static, fixed values to each row in a table*
- *Use measures when aggregation is necessary, or to create dynamic values in a report*

Write measures for even the simplest calculations *(i.e. Sum of Sales)*

- *Once you create a measure it can be used anywhere in the report and as an input to other, more complex calculations (no implicit measures!)*

Break measures down into simple, component parts

- *DAX is a difficult language to master; focus on practicing and understanding simple components at first, then assemble them into more advanced formulas*

Reference columns with the table name, and measures alone

- *Using "fully qualified" column references (preceeded by the table name) helps make formulas more readable and intuitive, and differentiates them from measure references*

*Copyright 2018, Excel Maven & Maven Analytics, LLC

BEST PRACTICES: SPEED & PERFORMANCE

Eliminate redundant columns; keep data tables narrow

- *Data tables should ideally only contain only quantitative values and foreign keys; any extra descriptive columns can usually live in a related lookup table*

Imported columns are better than calculated columns

- *When possible, create calculated columns at the source (i.e. in your raw database) or within the Query Editor; this is more efficient than processing those calculations in the Data Model*

Minimize iterator functions (FILTER, SUMX, etc.)

- *Functions that cycle through each row in a table are "expensive", meaning that they take time and consume processing power*

*Copyright 2018, Excel Maven & Maven Analytics, LLC

მადლობა ყურადღებისთვის

ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

მონაცემთა ვიზუალიზაცია; მოკლე მიმოხილვა; "Report" ჩანართი;

მარტივი ობიექტების დამატება (Report Canvas); ძირითადი გრაფიკების და დიაგრამების დამატება;

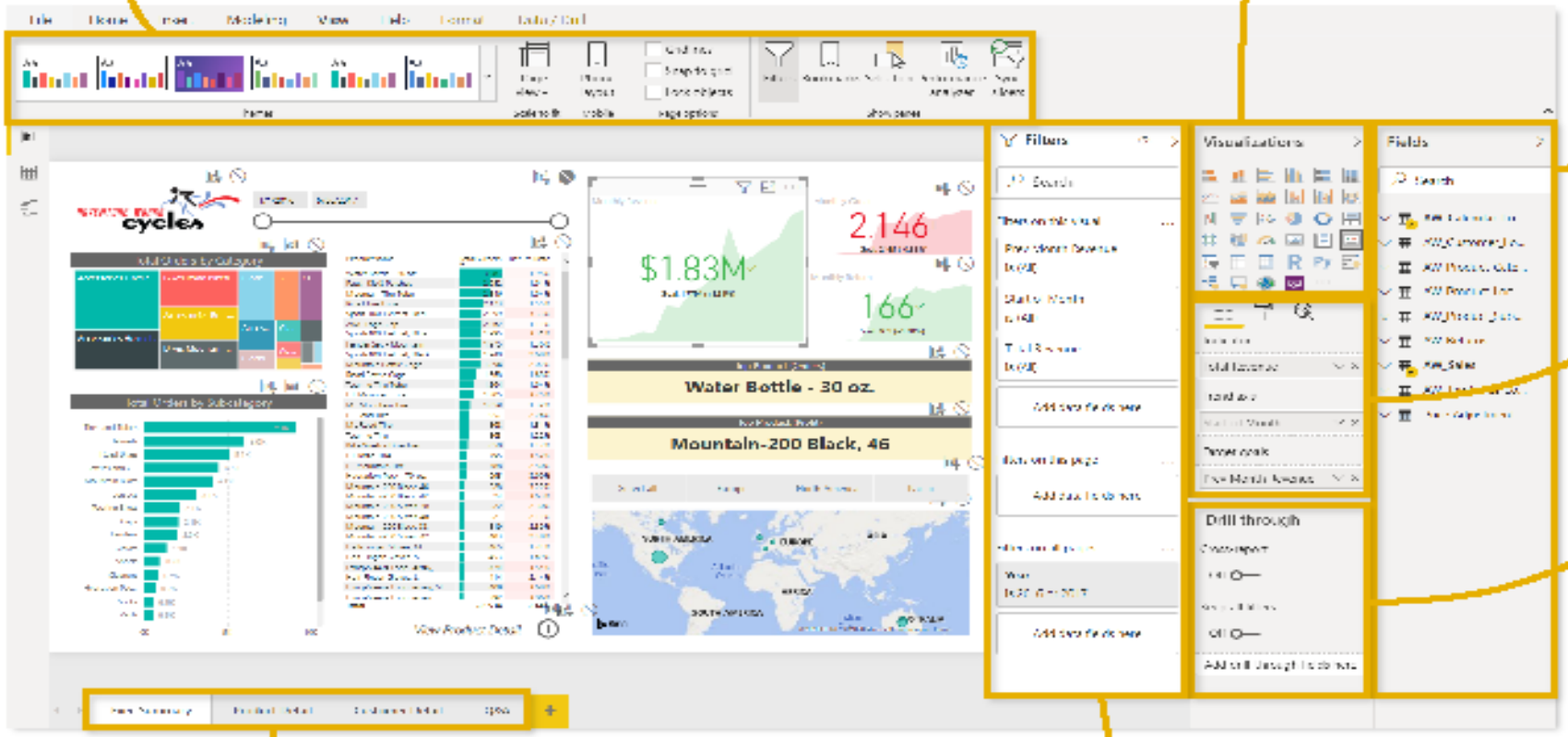
პირობითი ფორმატირება; რეპორტის ფორმატირების პარამეტრები;

რეპორტის ფილტრაციის პარამეტრები;

THE POWER BI REPORT VIEW

View Options (*Themes, Layouts, Gridlines, Filter/Bookmarks/Selection Panes, etc*)

Visualization Options (*Charts, Slicers, Maps, Matrices, etc*)



Field List (*Tables, Columns, Measures*)

Fields/Format/Analytics Pane (*Visual-specific configuration & formatting tools*)

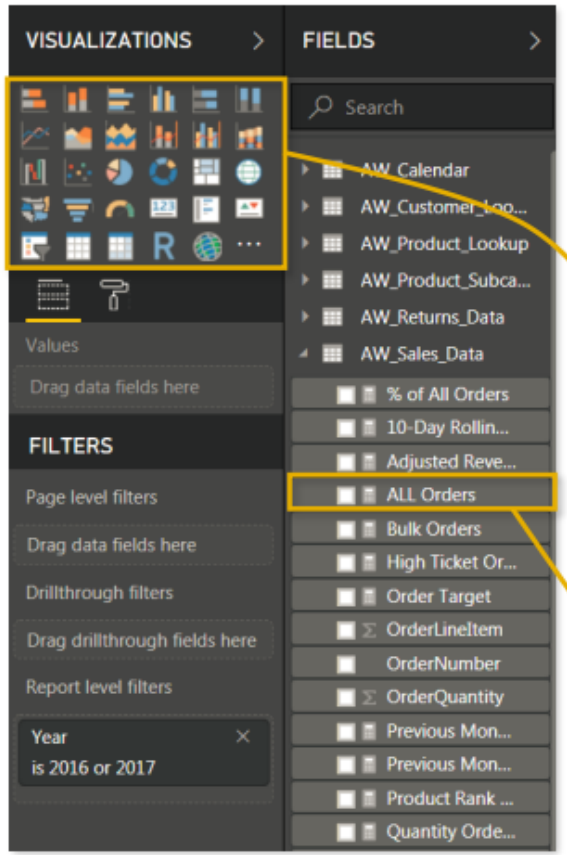
Drill through Filters (*Options for page-level drill through filters*)

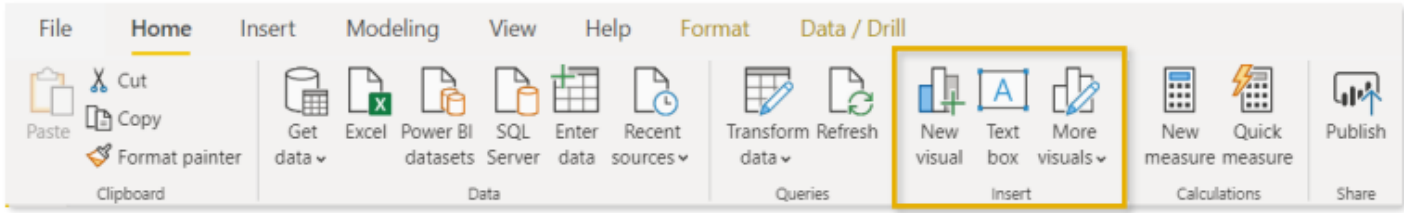
Report Pages (*Similar to Excel tabs; each is a blank reporting canvas*)

Filters Pane (*Visual-Level, Page-Level, and Report-Level Filters*)

*Copyright 2018, Excel Maven & Maven Analytics, LLC

INSERTING OBJECTS & BASIC CHARTS





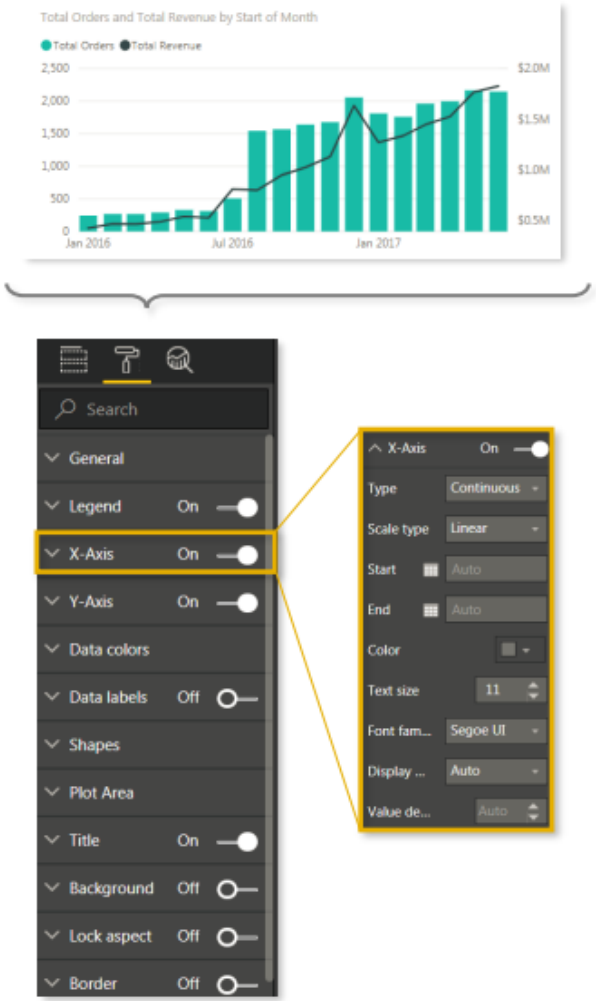
Click on a visualization type or use the "*New Visual*" option in the **Home** tab to insert a blank chart template (*usually a column chart by default*)

Note: *You can also add new visuals, along with Pages, Buttons, Images, and more from the* ***Insert*** *tab*

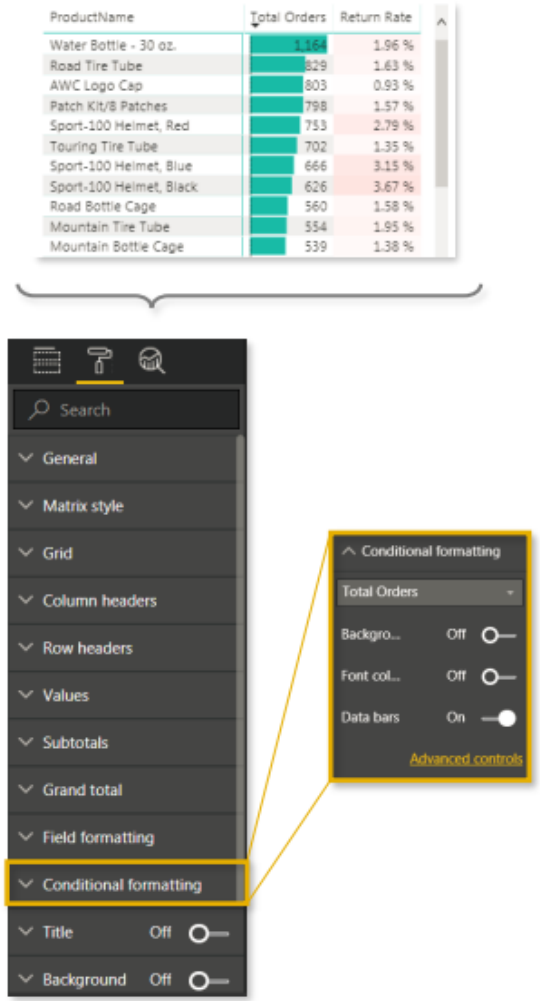
Drag fields or measures directly into the report canvas to automatically generate a new visual

FORMATTING OPTIONS

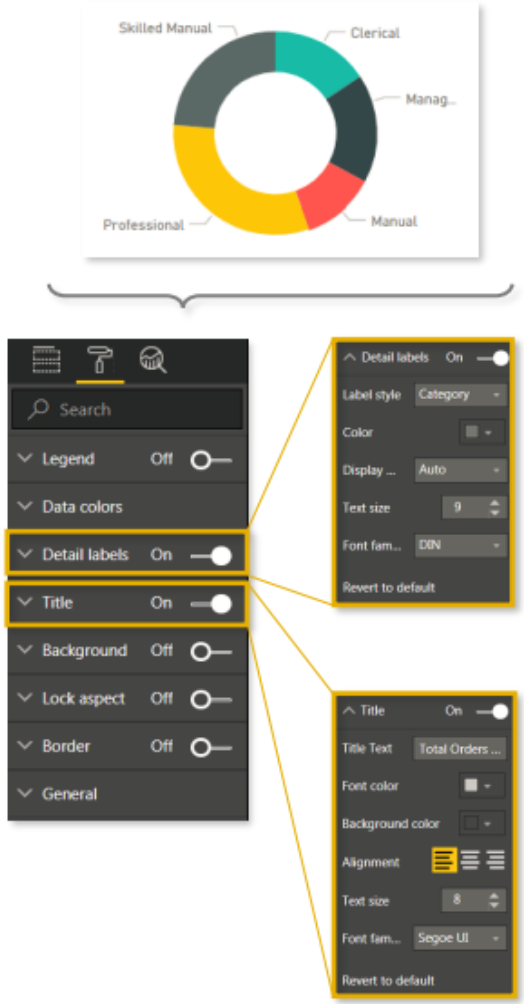
Example: ***Line & Column Chart***



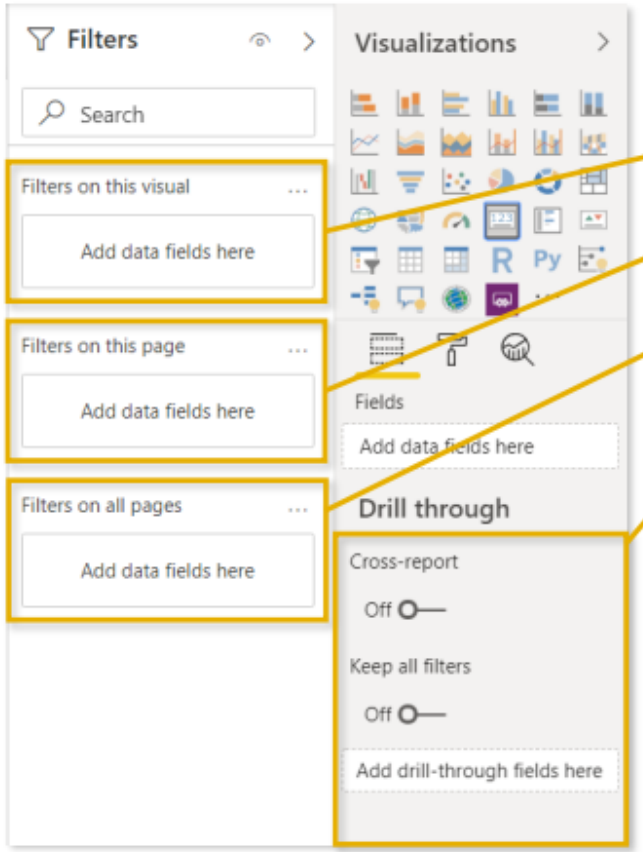
Example: ***Matrix***



Example: ***Donut Chart***



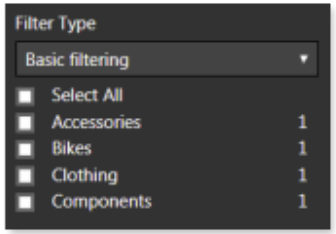
FILTERING OPTIONS



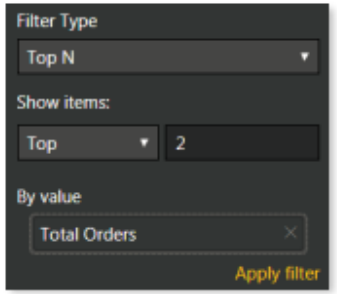
There are **four (x4)** primary filter types in Power BI reports:

1. **Visual Level:** Applies only to the ***specific visual*** in which it is defined
2. **Page Level:** Applies to ***all visuals on the specific page*** in which it is defined
3. **Report Level:** Applies to ***all visuals*** across ***all pages*** of the report
4. **Drill through:** Applies to ***specific pages***, and ***updates*** based on the item clicked

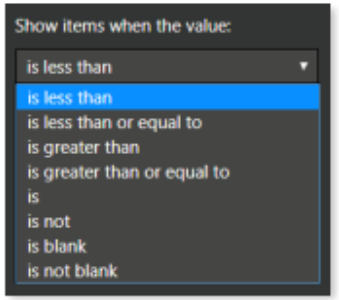
Filter settings include Basic, Advanced, and Top N options



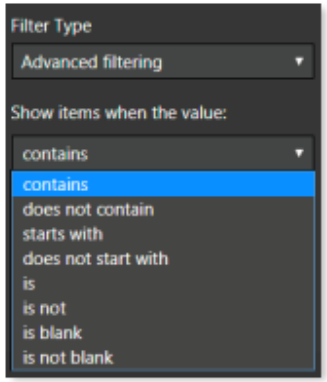
Basic Options



Top N Options



***Advanced** (Values)*



***Advanced** (Text)*

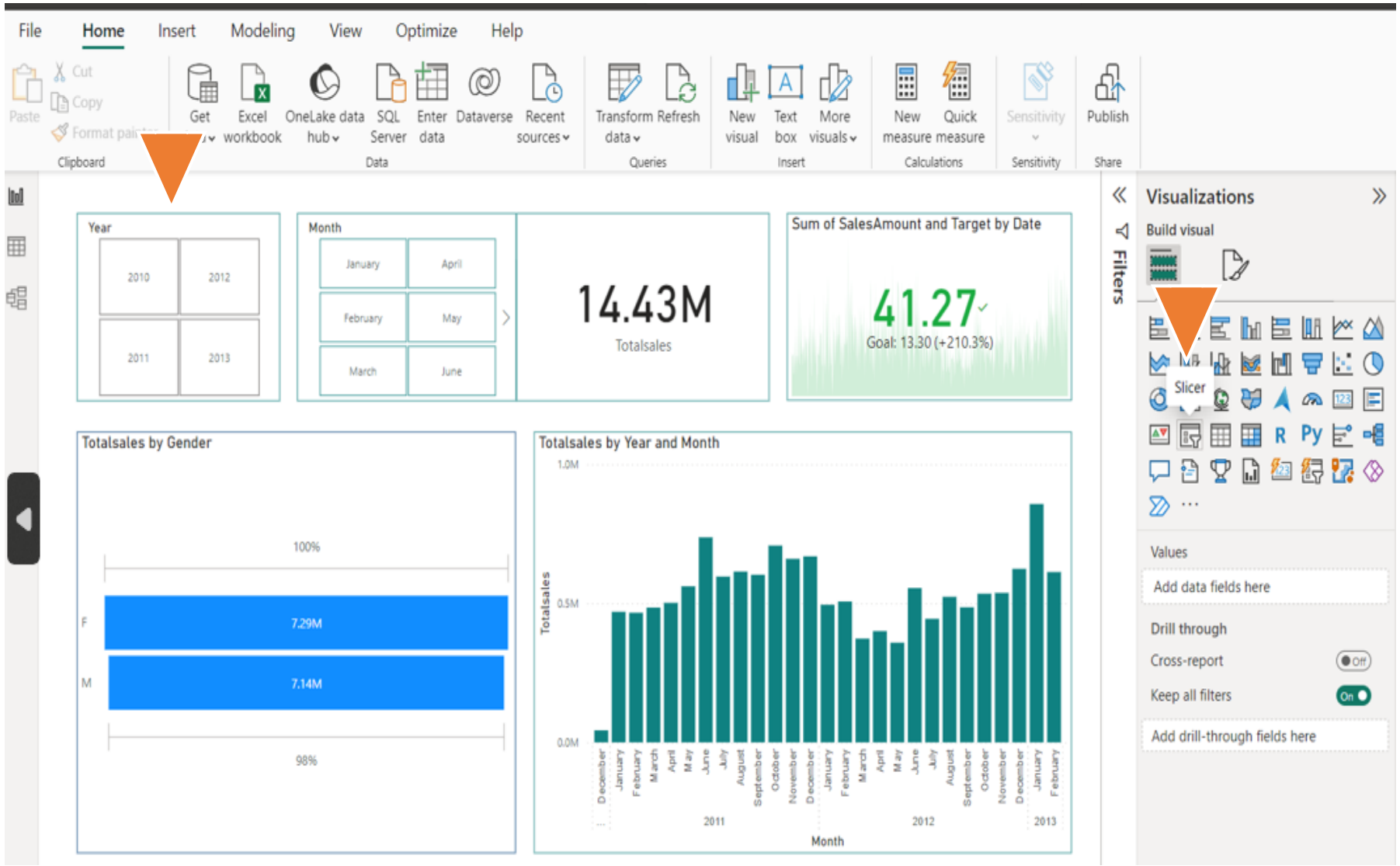
მადლობა ყურადღებისთვის

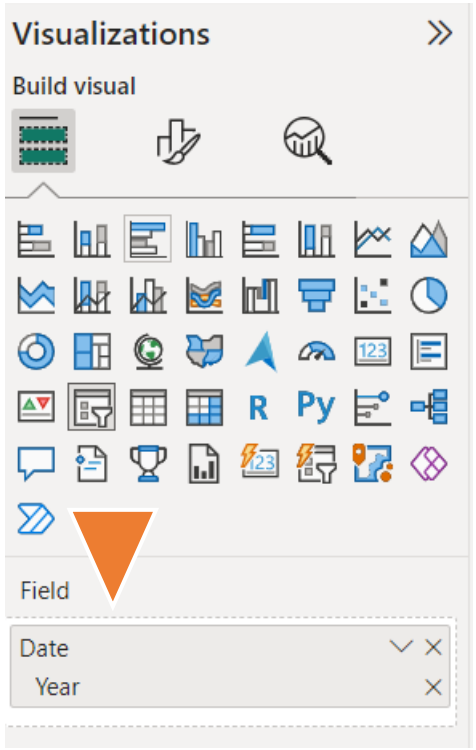
ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

- მონაცემთა ვიზუალიზაცია; მონაცემების გაფილტვრა Slicer-ის გამოყენებით
- ძირითადი მეთოდების ჩვენება Cards და KPI ვიზუალებით;
- Geospatial მონაცემების ვიზუალიზაცია რუკების გამოყენებით;
- მონაცემების ვიზუალიზაცია Treemaps-ის გამოყენებით;
- ტენდენციების ჩვენება ხაზოვანი და რეგიონალური დიაგრამებით

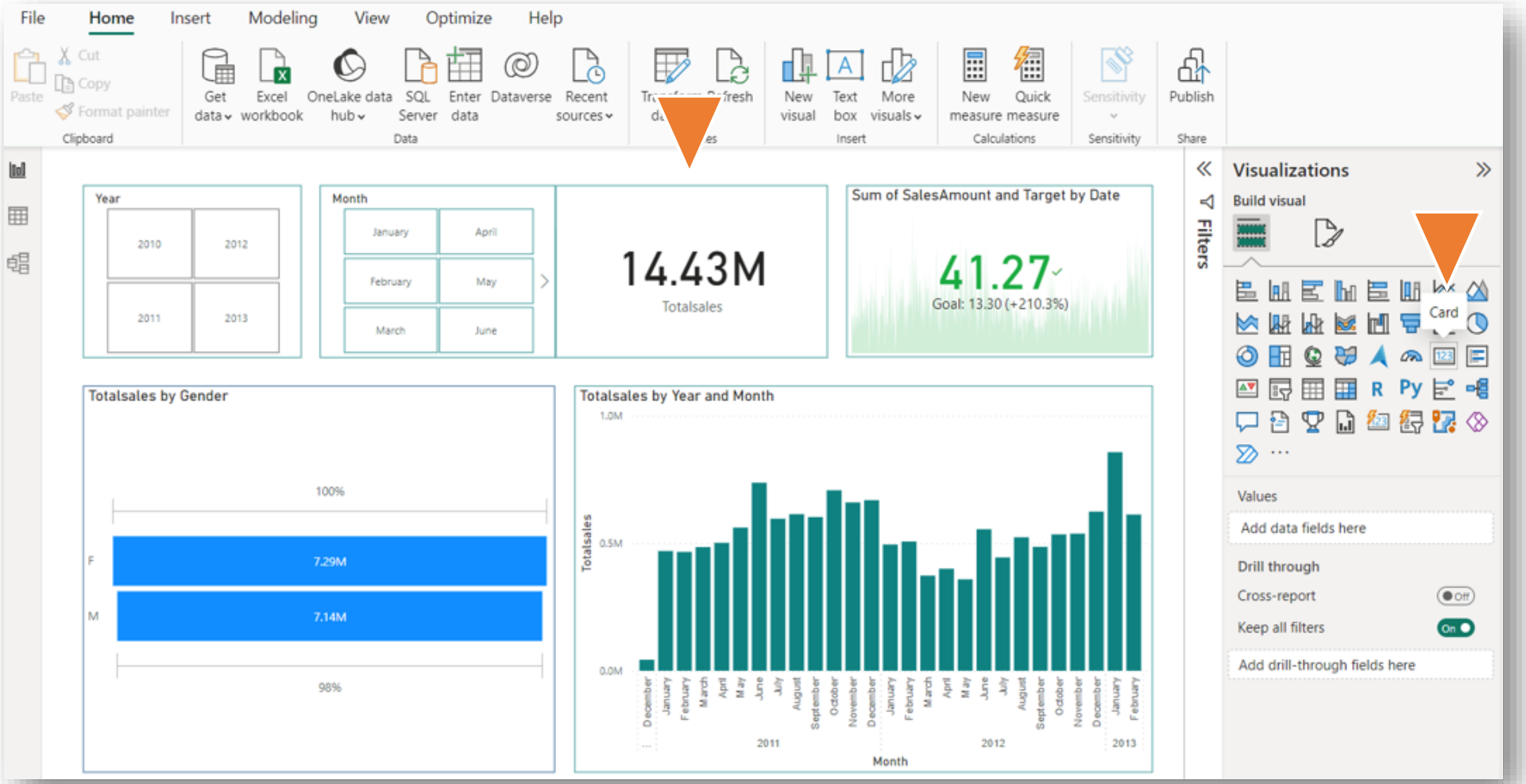
მონაცემების გაფილტვრა Slicer-ის გამოყენებით

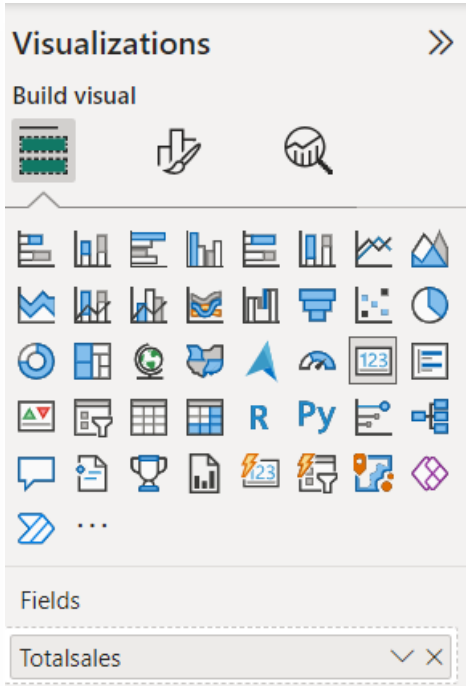




Fields-ს ჩანართში ვაგდებთ იმ სვეტს, რომელიც გვინდა Slicer-ზე გავააქტიუროთ

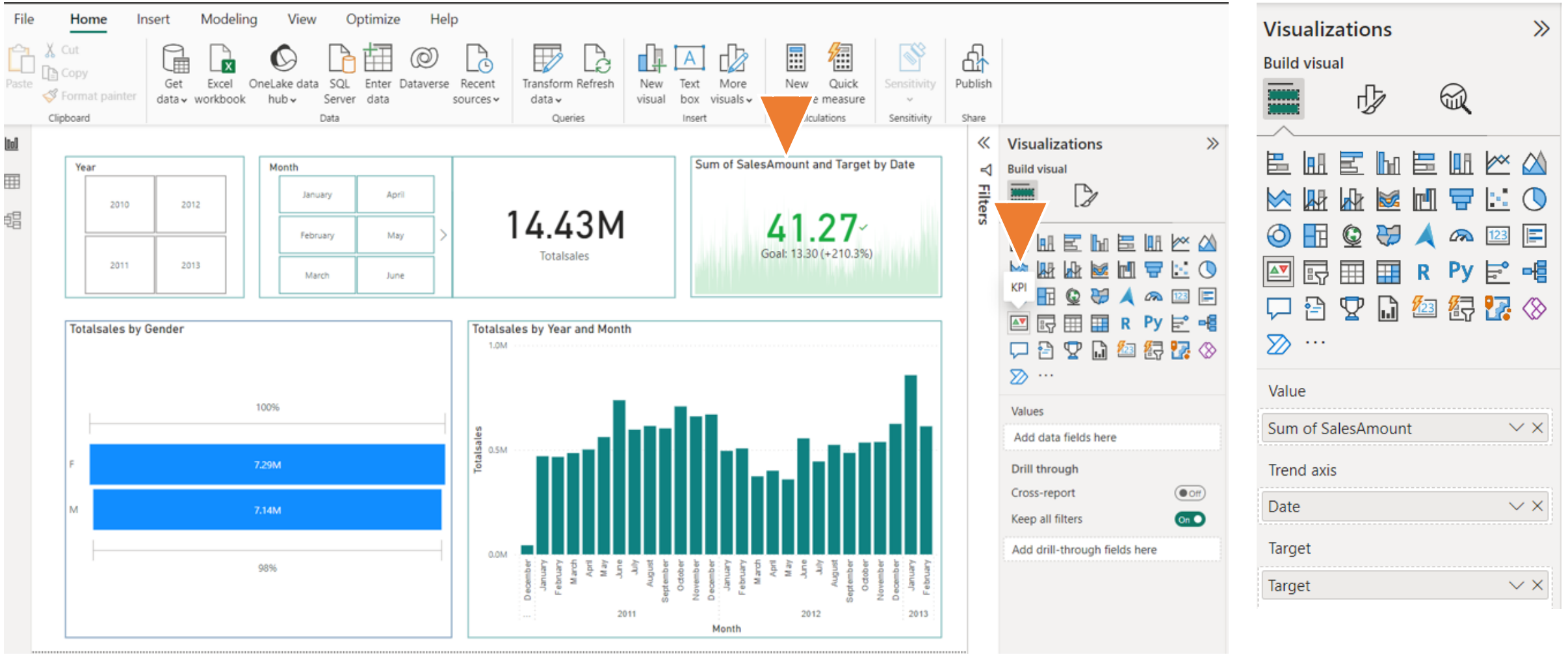
ძირითადი მეთოდების ჩვენება Cards ვიზუალებით





Fields-ს ჩანართში ვაგდებთ იმ Measure-ს, რომელიც გვინდა Cards-ში გამოვიტანოთ.

ძირითადი მეთოდების ჩვენება KPI ვიზუალებით;

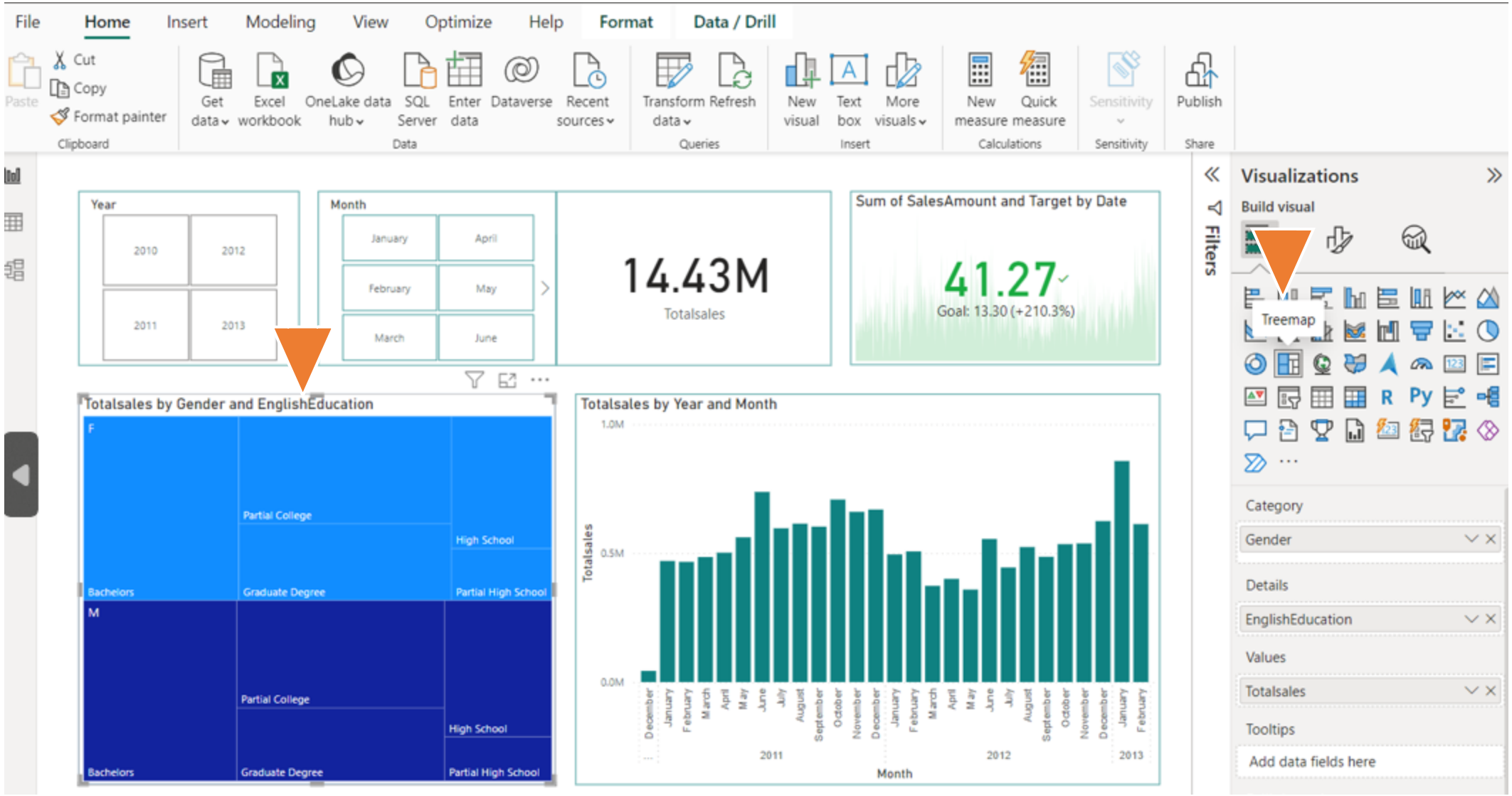


Trend axis -აჩვენებს ტენდენციის ღერძს, როგორც KPI ვიზუალის ფონს, ჩვენს შემთხვევაში გამოყენებულია თარიღი.

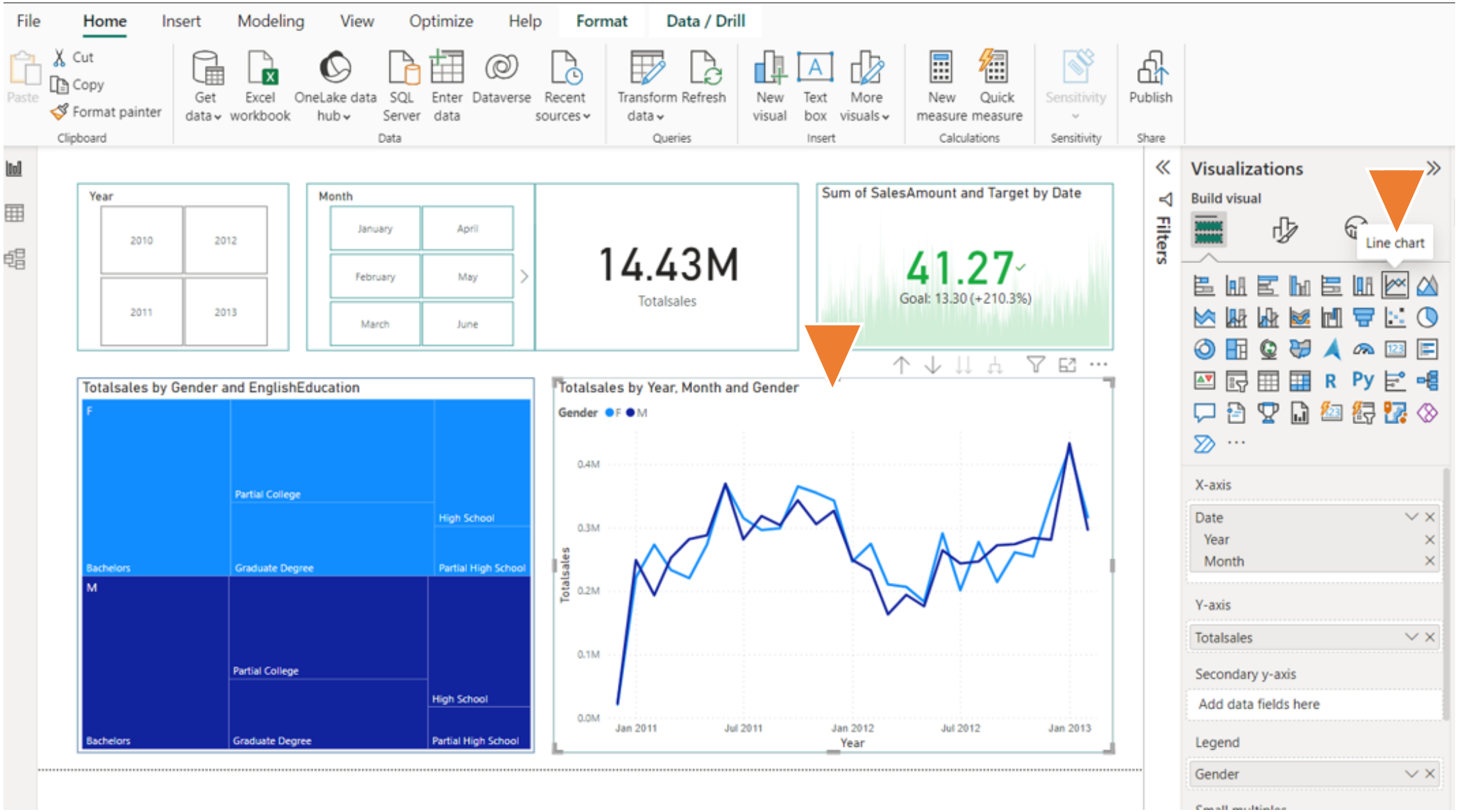
Value - ფაქტიურად დამდგარი შედეგი

Target - სამიზნე მაჩვენებელი

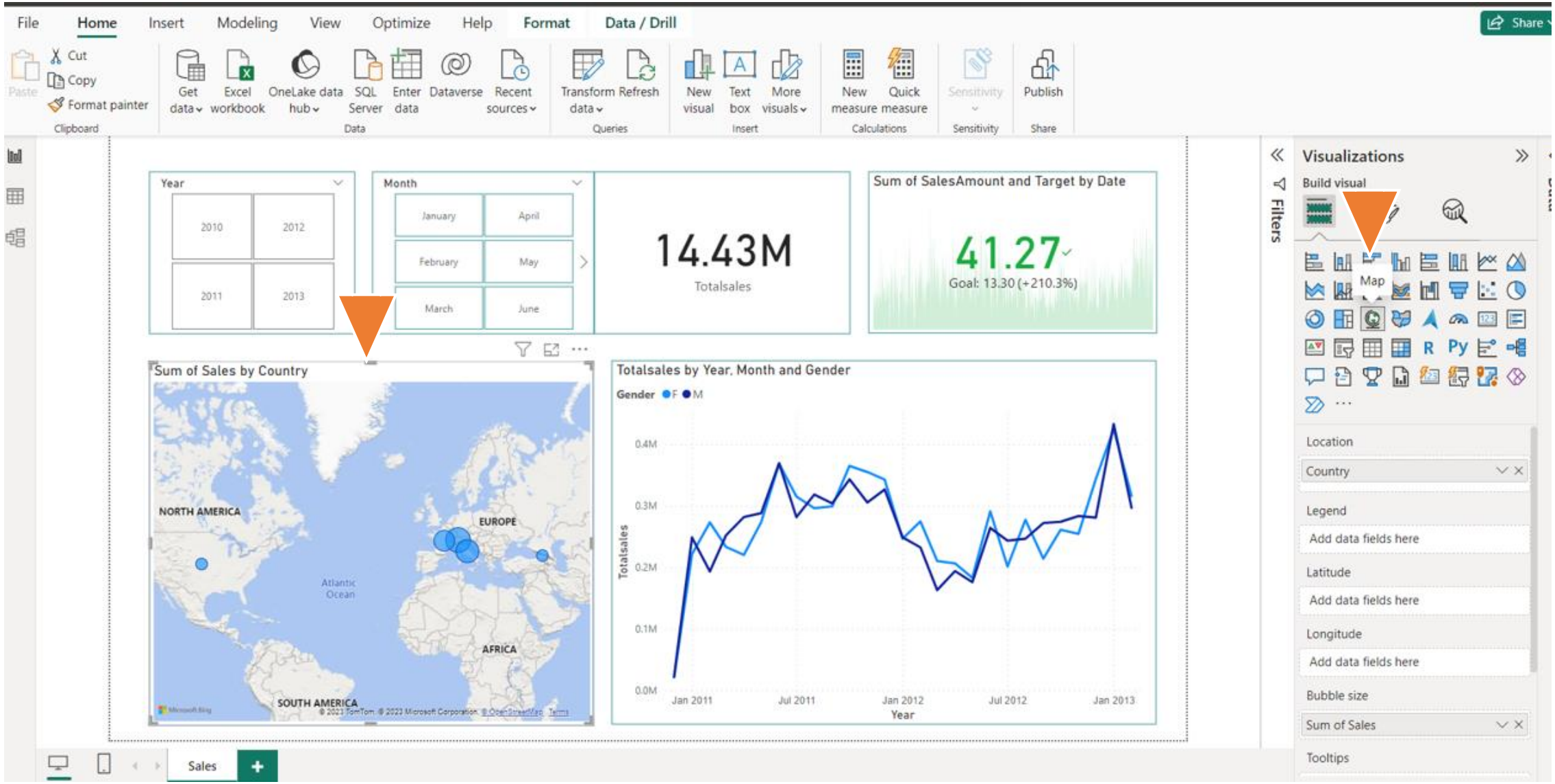
მონაცემების ვიზუალიზაცია Treemap-ის გამოყენებით;



ტენდენციების ჩვენება ხაზოვანი და რეგიონალური დიაგრამებით



Geospatial მონაცემების ვიზუალიზაცია რუკების გამოყენებით;



მადლობა ყურადღებისთვის

ანალიტიკური ინსტრუმენტი – Microsoft Power BI Desktop

გასავლელი საკითხები

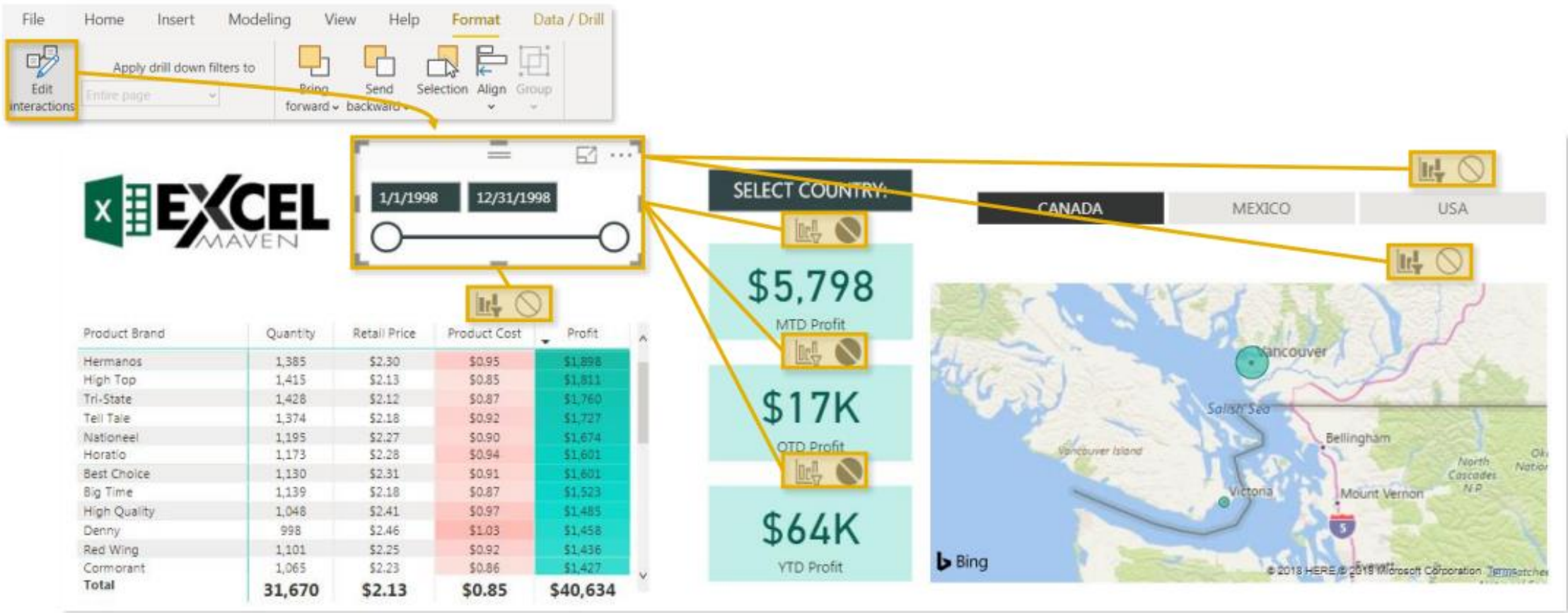
მონაცემთა ვიზუალიზაცია; ტენდენციური ხაზების და პროგნოზების დამატება; Gauge Chart;

ურთიერთქმედების რედაქტირება; Drillthrough ფილტრების დამატება; სანიშნეების გამოყენება;

სცენარების ტესტირება "What-If" პარამეტრებით; როლების მართვა და დათვალიერება;

პროექტირება ტელეფონისა და დესკტოპის მნახველებისთვის; გამოქვეყნება Power BI Server-ში; Power BI მონაცემთა ვიზუალიზაციის საუკეთესო პრაქტიკა;

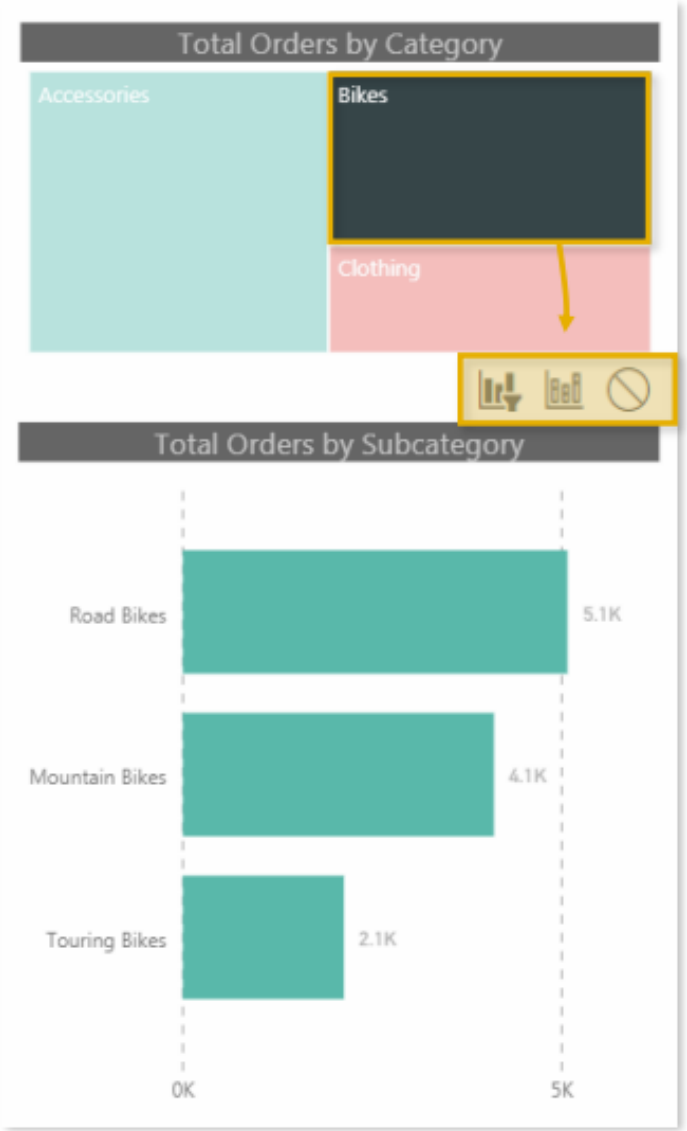
EDITING REPORT INTERACTIONS

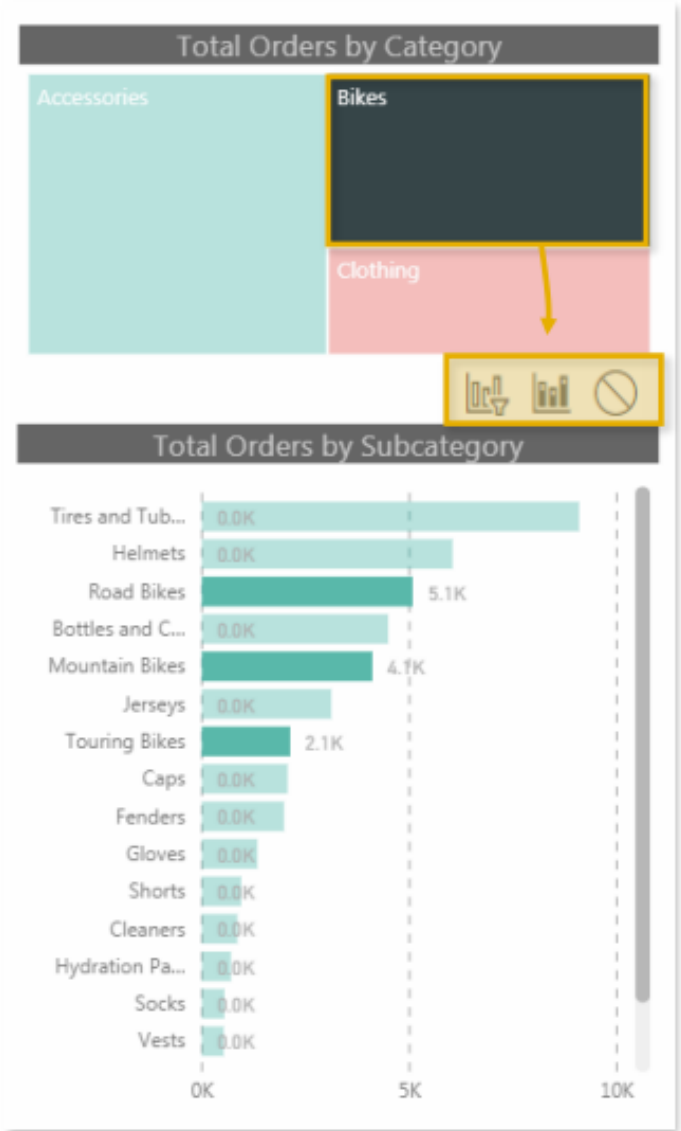


Report interactions allow you to determine how filters applied to *one* visual impact the *others*

- For example, by selecting the Timeline visual and enabling "*Edit interactions*" from the **Format** tab, we can manually determine which visuals should "*react*" when the date range changes
- In this case the **Product matrix**, **Country slicer** and **Map** will filter in response to timeline changes ( ), but the **MTD**, **QTD**, and **YTD Profit** cards *will not* ( )

EDITING REPORT INTERACTIONS (CONT.)

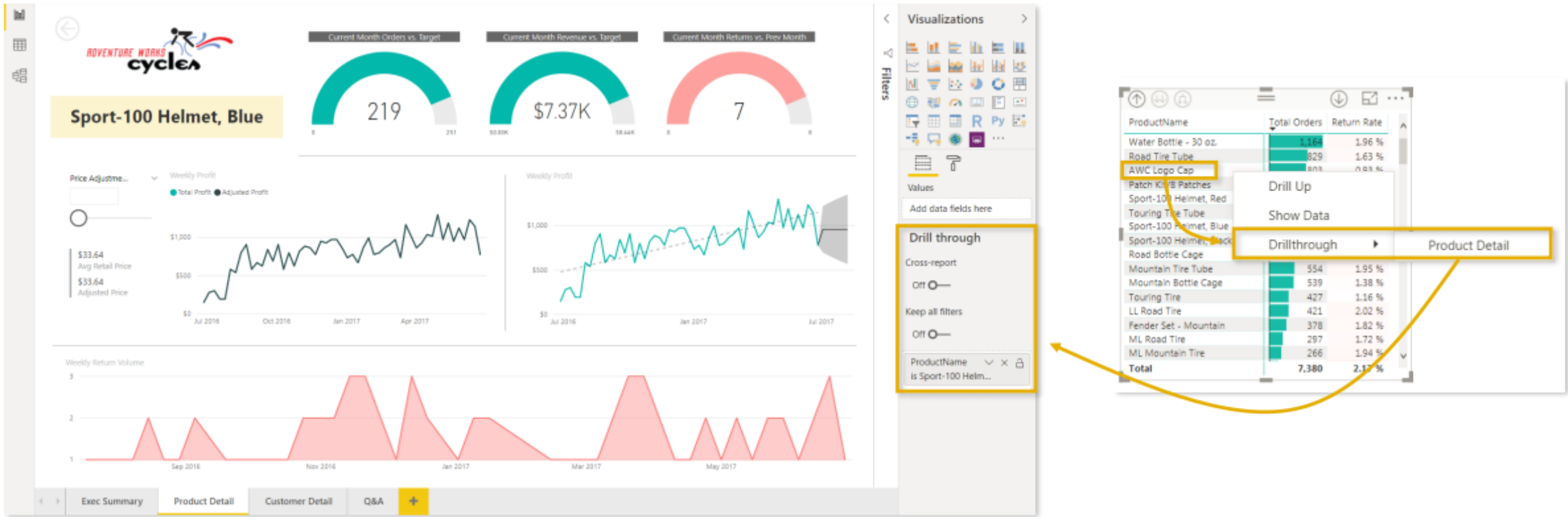




For certain types of visuals, a third option allows you to "**highlight**" sub-segments of the data, rather than simply filtering vs. not filtering

- When the interaction mode is set to "**filter**", selecting the "***Bikes***" category in the treemap produces a filtered list of subcategories in the chart below
- When the interaction mode is set to "**highlight**", selecting the "***Bikes***" category in the treemap highlights the relevant subsegment of data in the chart below

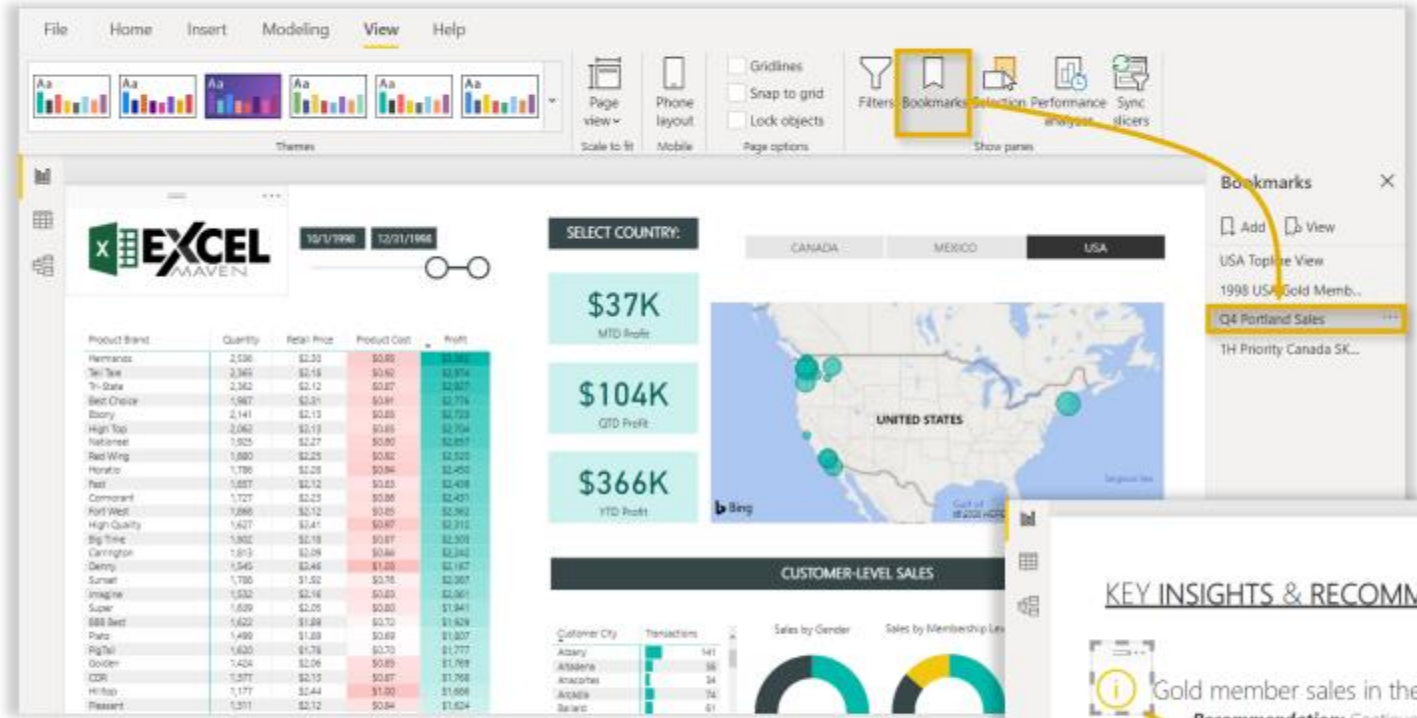
DRILLTHROUGH FILTERS



Drill through filters allow users to jump to different report pages (*like bookmarks*), while simultaneously filtering based on the *specific item selected*

- Here we've built a report page ("**Product Detail**") featuring *product-level* performance, and added a Drillthrough filter for **ProductName**; users can now right-click any report visual containing product names, and jump straight to a pre-filtered version of this page (*"**Sport-100 Helmet, Blue**" shown in the example above*)

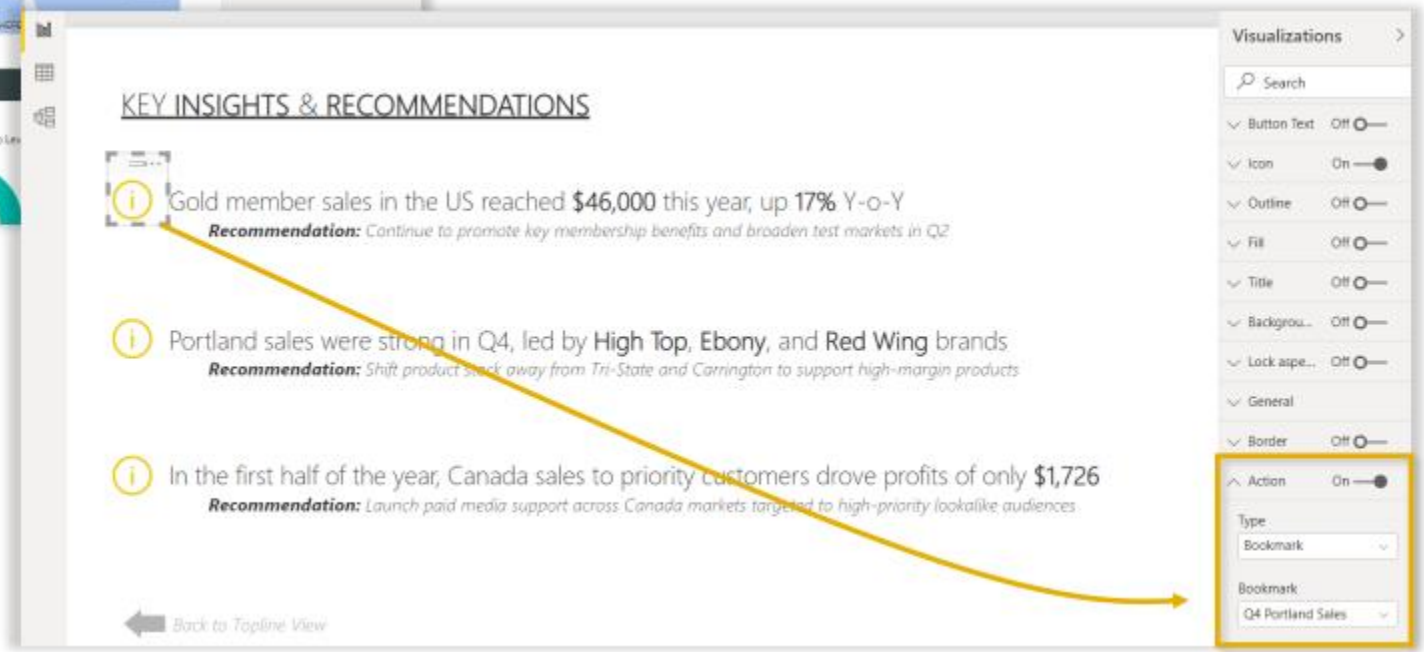
ADDING & LINKING BOOKMARKS



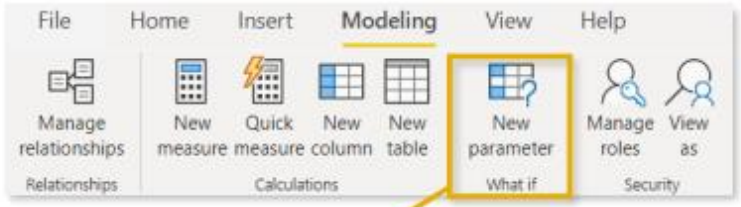
In this example, we notice that Q4 sales were particularly strong in the Portland market, so we add a new **bookmark** (*View > Bookmarks Pane > Add*) and name it "**Q4 Portland Sales**"

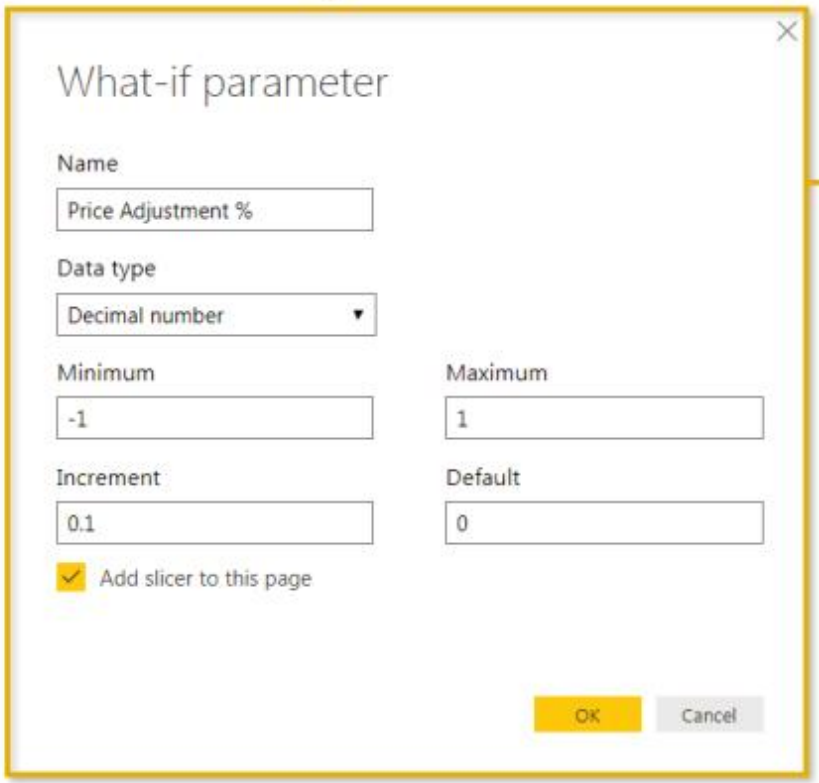
On a new page, we present our key insights, insert buttons, and link them to bookmarks using the object "***Action***" properties

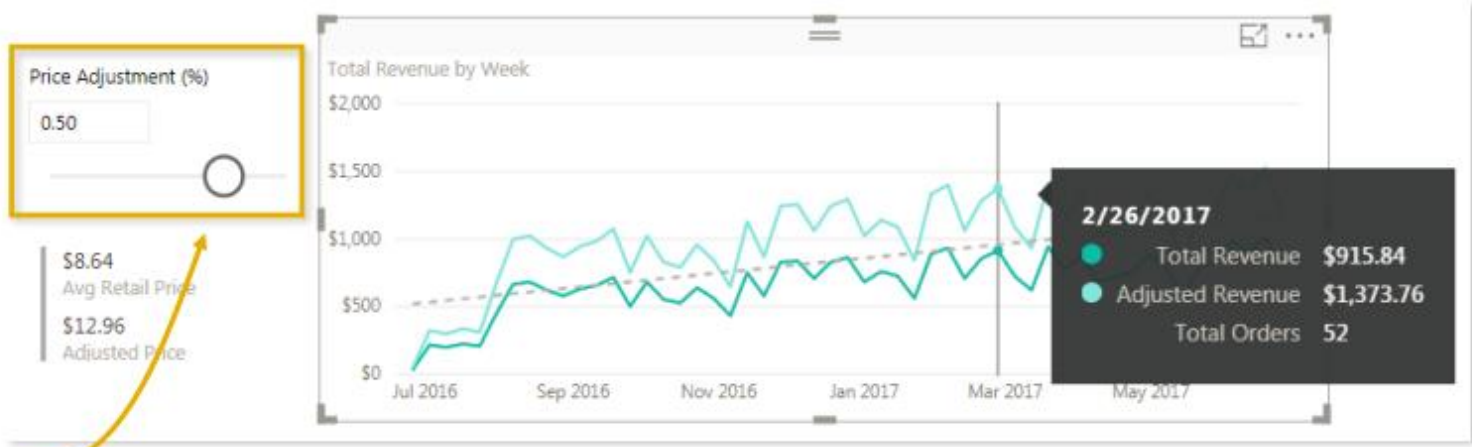
Now we're able to create a *narrative* from the data, and really bring our insights to life!



“WHAT-IF” PARAMETERS







“What If” Parameters are essentially pre-set measures that produce values within a given range, based on user-inputs (*data type, min/max, increment, and default*)

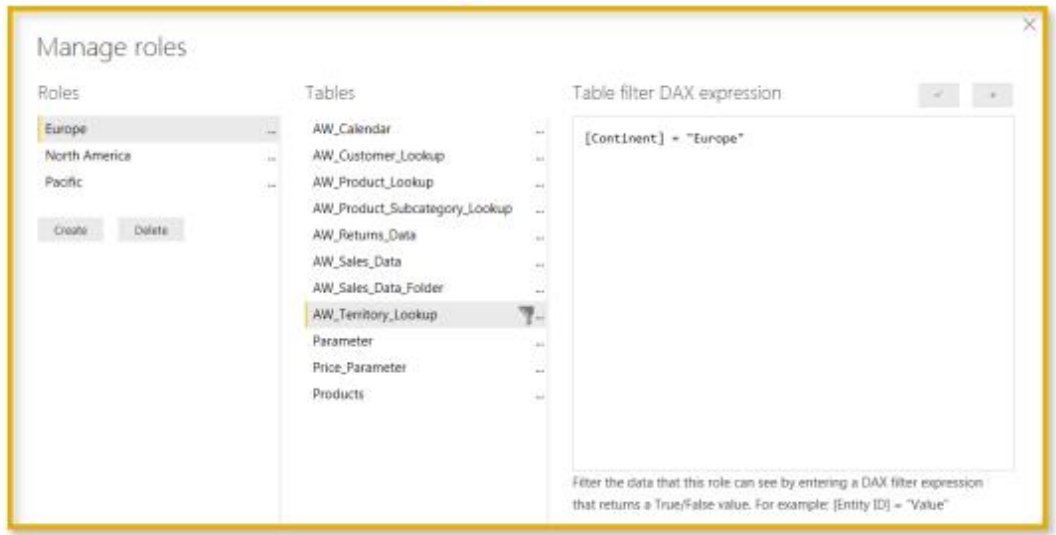
These can be great tools for forecasting or scenario testing; here we’ve created a “***Price Adjustment %***” parameter in order to compare **Total Revenue** (based on the *actual* price) against **Adjusted Revenue** (based on the *parameter-adjusted* price)

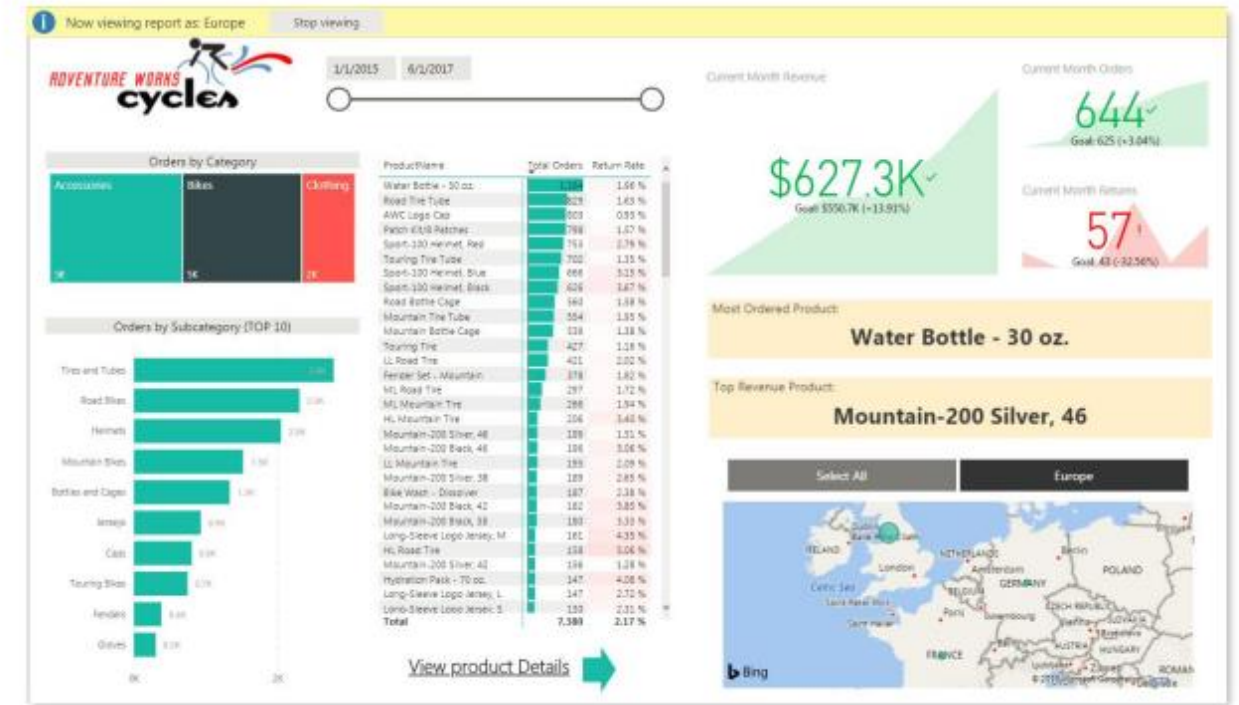
NOTE: *When you create a parameter, a new table is automatically added with DAX calculations for “**Parameter**” and “**Parameter Value**”, which look something like this:*

```
Parameter = GENERATESERIES(-1, 1, 0.1)
```

```
Parameter Value = SELECTEDVALUE(Parameter[Parameter], 0)
```

MANAGING & VIEWING AS ROLES

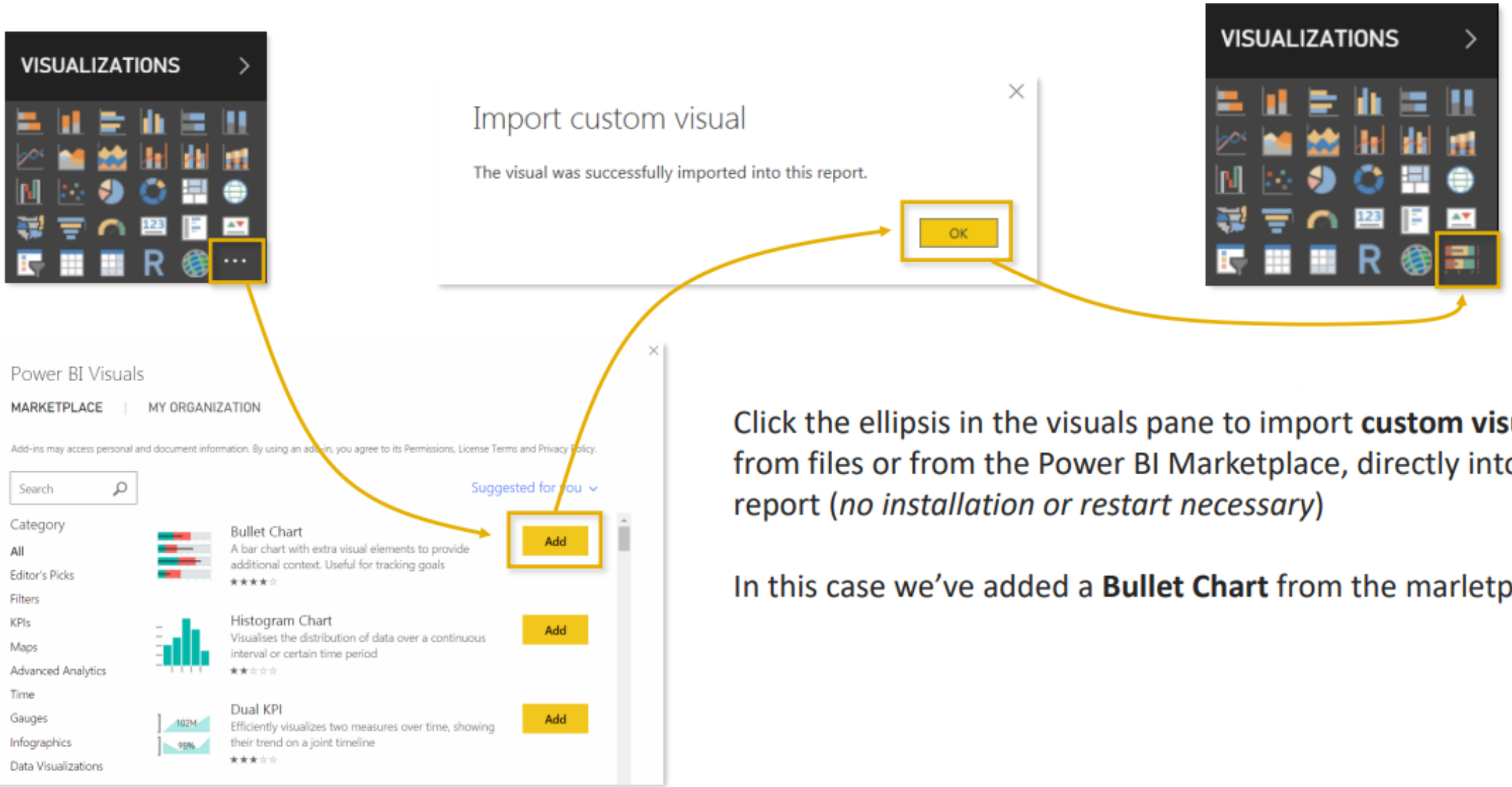




Roles allow you to define filtered views that can be tailored to specific audiences

- In this example we've created views for each territory manager (***Europe***, ***North America***, *and* ***Pacific***) based on simple DAX filter statements

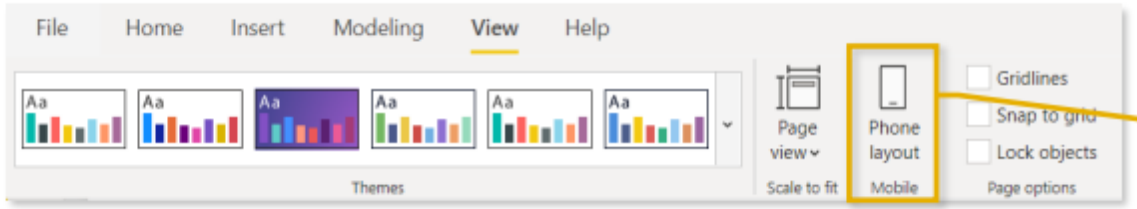
IMPORTING CUSTOM VISUALS

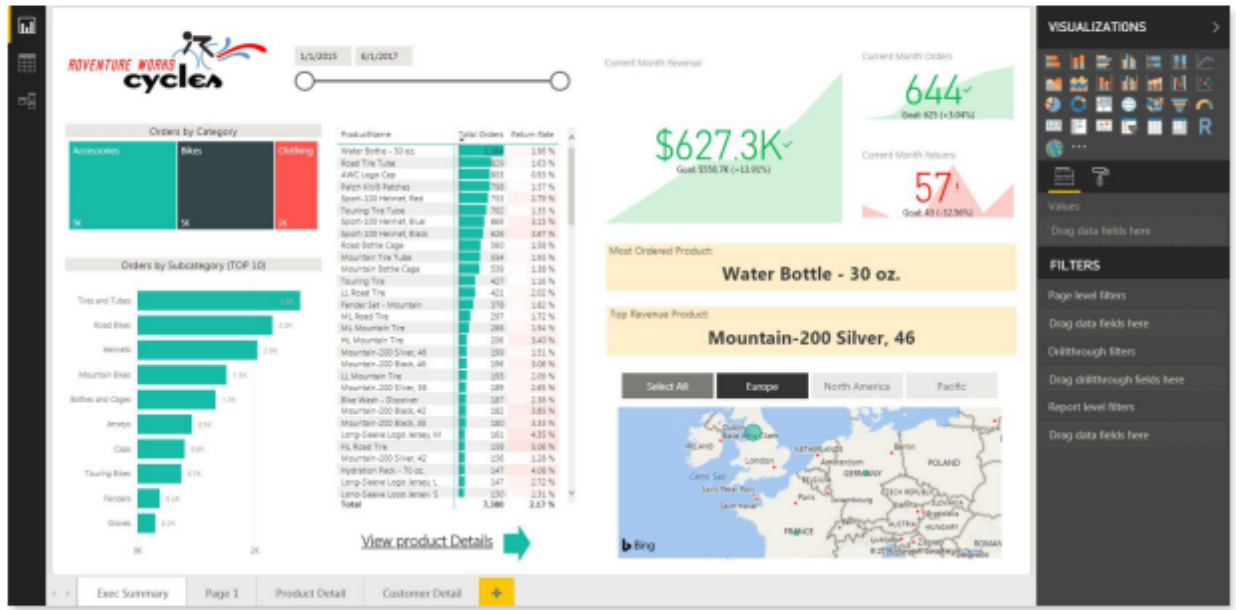


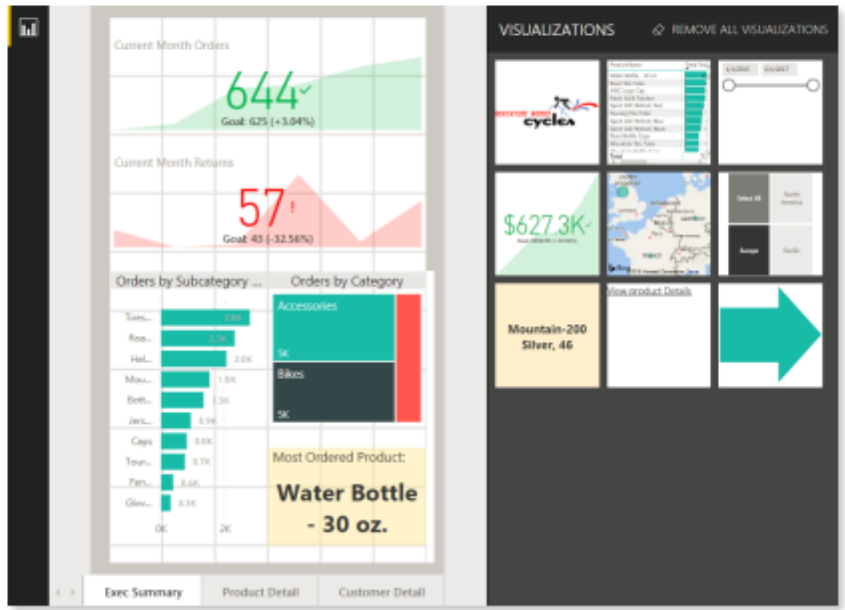
Click the ellipsis in the visuals pane to import **custom visuals** from files or from the Power BI Marketplace, directly into the report (*no installation or restart necessary*)

In this case we've added a **Bullet Chart** from the marletplace

DESKTOP VS. PHONE LAYOUT







Phone Layout view allows you to design on a canvas size optimized for mobile viewing (*vs. desktop*)

- ***NOTE:*** *You can't actually build content within the Phone Layout view; recommend building in* ***Desktop Layout****, and assembling select visuals for mobile if you plan to share content via the Power BI app*

DATA VISUALIZATION **BEST PRACTICES**

Strive for clarity & simplicity, above all else

- *Aim to maximize impact and minimize noise; it's all about balancing design and function*

Don't just build charts and graphs; create a *narrative*

- *Without context, data is meaningless; use filters, bookmarks, and effective visualizations to translate raw data into powerful insights and implications*

Always ask yourself the three key questions:

1. *What **type of data** are you visualizing? (Integer, categorical, time-series, geo-spatial, etc)*
2. *What are you trying to **communicate**? (Relationships, compositions, trending, etc)*
3. *Who is the **end user** consuming this information? (Analyst, CEO, client, intern, etc)*

მადლობა ყურადღებისთვის