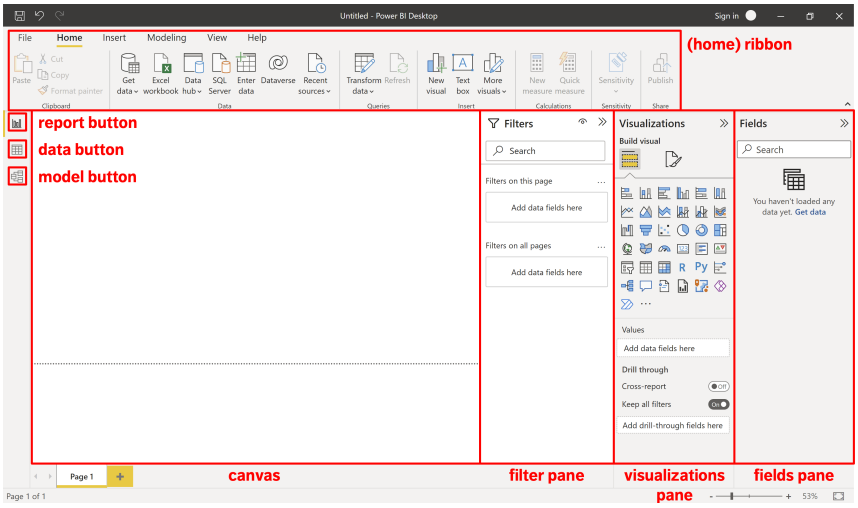
13 Power BI for Beginners

Dashboarding tools are, for the most part, **story-driven** tools: Microsoft's Power BI is one such tool. Among its main attributes, we note that its learning curve (while reasonably steep, as is the case with all new tools) is attenuated somewhat by its distinctly Microsoft-ish **point-and-click functionality**, and that it allows dashboards to be easily **published** and **distributed** on the web (either internally and/or externally).*

On the negative side of the ledger, Power BI does not play nicely with MacOS and Linux (although there are workarounds).

Once Power BI has been downloaded and installed on a PC, we tap it open. The layout will look as in Figure 13.1.

We will explain what each of these components does as we go through the explanations: for now, we simply want to ensure that their locations on the screen is easy to find.[1]



One of the challenges with writing detailed how-to instructions for software that updates regularly is that the material quickly becomes obsolete – you may need to visit online sources for updated instructions.

* The point-and-click approach can quickly become byzantine; there is a programmatic approach using M and DAX. We will discuss those in the forthcoming sections.

1: With older versions of Power BI, the layout might instead look like:

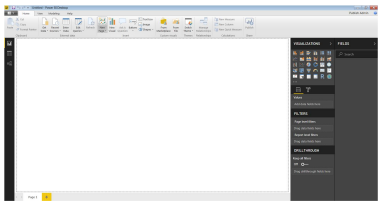


Figure 13.1: The components of the default Power BI view.

Case Study **Borealis Terraformers LLC** is a terraforming company. It does business, it transacts with consultants, it has employees, it pays them, it delivers products, it records its transactions. It also has departments/projects, and budget categories and codes (Major Capital [MC], Minor Capital [MIC], Operations and Management [O&M], and Salary [SA]).[2]

2: Those of you with a background in accounting and project management will realize that this is not exactly a realistic description... but just roll with it.

We will demonstrate the various Power BI capabilities using datasets associated to this "company":

- `Data-Set-Accounting.xlsx`
- `Data-Set-Projects.xlsx`

Download them to your local machine, in an easily-accessible location (you may need/want to rename the files).

The accounting dataset consists of data relating to accounting transactions.

1. The first sheet (`Accounting Transactions`) contains the transactions themselves:

Accounting Control Number	Journal Voucher Type Code	Accounting Effective Date	Journal Voucher Item Amount	Project Identifier
5000085	MC	01-Mar-18	$173,516.11	PR007
5000086	SA	02-Mar-18	$54,298.15	PR009
5000087	O&M	03-Mar-18	$49,584.50	PR010
5000088	MIC	04-Mar-18	$89,293.40	PR011
5000089	SA	05-Mar-18	$93,866.67	PR012
5000090	O&M	06-Mar-18	$47,942.67	PR013
5000091	MC	02-Apr-18	$177,734.54	PR007
5000093	SA	03-Apr-18	$84,391.34	PR009
5000094	O&M	04-Apr-18	$21,520.07	PR010

Table 13.1: First observations in the accounting transaction tab.

2. The second sheet (`Journal Voucher Type Code`) contains the voucher codes:

Code	Description
O&M	Operations and Maintenance
MC	Major Capital
MIC	Minor Capital
SA	Salary

Table 13.2: Journal voucher codes in the accounting dataset.

The projects dataset consists of data relating to projects and departments.

3. The first sheet (`Project Tombstones`) contains information about projects, directors, and budgets:

Project Identifier	Director	Project Name	O&M Budget	Salary Budget	Major Cap Budget	Minor Cap Budget	FTE Budget
PR001	A. Thakur	Parks	$2,500,000	$2,000,000	$5,000,000	$1,000,000	9.00
PR002	G. Bertrand	Buildings	$5,000,000	$4,000,000	$5,000,000	$5,000,000	6.00
PR003	C. Power	Emergency Response	$3,000,000	$7,000,000	$800,000	$3,000,000	6.00
PR004	H. Schlivofszky	Office	$4,000,000	$4,000,000	$8,000,000	$200,000	12.00
PR005	G. Bertrand	Roads	$5,000,000	$2,000,000	$2,500,000	$1,000,000	7.00
PR006	B. Bouraoui	Science	$5,000,000	$10,000,000	$8,000,000	$2,000,000	7.00
PR007	A. Thakur	Heritage	$1,500,000	$5,000,000	$3,000,000	$1,700,000	9.00
PR008	A. Thakur	Celebration	$2,000,000	$4,000,000	$0	$0	7.00
PR009	B. Bouraoui	Research	$5,000,000	$1,200,000	$10,000,000	$500,000	6.50
PR010	G. Bertrand	Upgrades	$4,000,000	$2,000,000	$10,000,000	$500,000	10.00

Table 13.3: Project information in the project dataset.

4. The second sheet (`Project FTE Code`) records the full time equivalent (FTE) changes in the roster of the various projects:

Project Code	Date	FTE (-out +)	Group-Level
PR001	01-Apr-18	2	AS-04
PR002	01-May-18	2	AS-05
PR003	01-Jun-18	2	ENG-01
PR004	01-Jul-18	4	PR-01
PR005	01-Aug-18	3	PA-03
PR006	01-Sep-18	2	AS-02
PR007	01-Oct-18	1	CR-03
PR008	01-Nov-18	3	FI-02
PR009	01-Dec-18	4	FO-03
PR010	01-Jan-19	2	CR-01
PR011	01-Feb-19	3	AS-03

Table 13.4: Changes in projects FTEs (extract).

The conceptual interaction diagram of these datasets is shown in Figure 13.2.

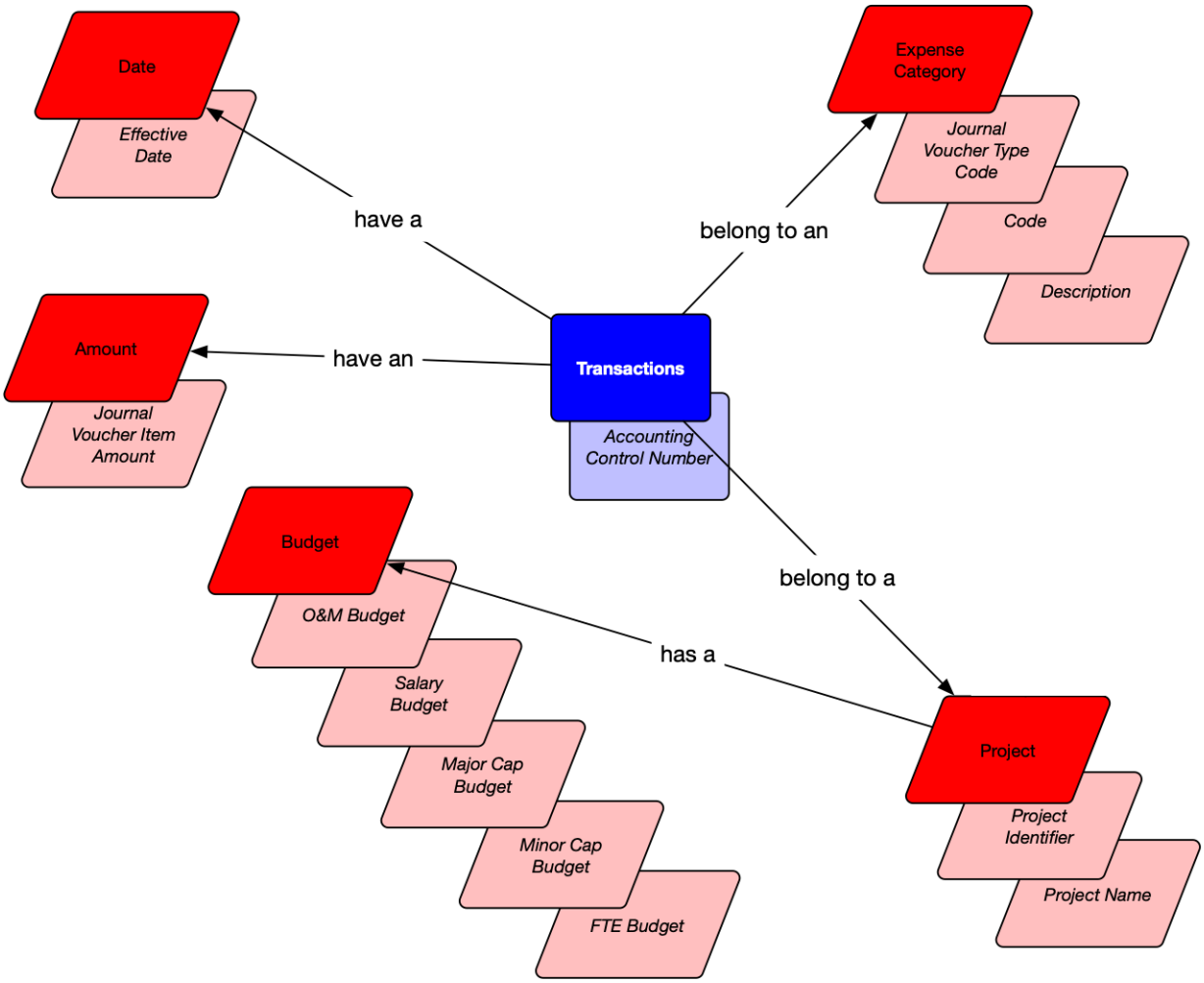


Figure 13.2: Conceptual interactions in the Borealis Terraformers LLC data.

13.1 Importing and Modeling Data

It is important to realize that Power BI is really the combination of two products, *Power Query* and *Power BI*.[3]

3: Power Query can also be found in Microsoft Excel and other software.

Data is **imported** and **manipulated** at a low-level through Power Query (this is done in the background using the Microsoft language M). Once this is completed and we close Power Query, the data is **pushed** into Power BI, where we manipluate it further using an Excel-like language called DAX.

M is Power BI's **data transformation** engine; M Query is a mashup query language used to query sets accross multiple data sources. M contains commands to transform data and to return query and transformation results to the Power BI **data model**.[4] Normally, we would use M Query to query data sources, and to clean and load data into Power BI.

4: See Information Gathering for more information on the topic [1].

M can be used for data preparation and data transformation tasks **before** the data is loaded into the model. For instance, instead of bringing 3 tables into Power BI, we could use M to remove unneeded columns and merge the tables together into a single table, which would then be loaded into the data model. This can help improve the performance of Power BI once the data model is fully loaded.

DAX (**d**ata **a**nalysis e**X**pression language) is Power BI's **analytical engine**. It is a common language used by SQL Server Analysis Services Tabular, Power BI, and Excel's Power Pivot. Once the data is loaded in Power BI, it is used to create custom columns, tables, and/or measures (among others).

Unlike M, DAX has similarities to Excel functions, but it provides a much wider range of functionality (and more power) than the latter.[5]

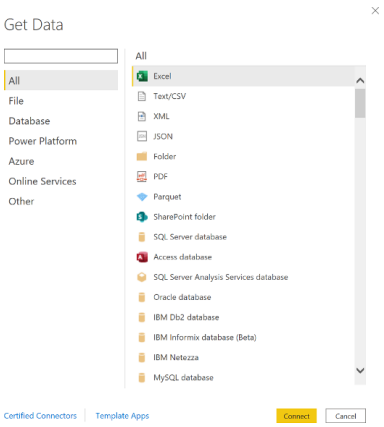
5: We will provide a deeper discussion of DAX at a later stage.

Importing Datasets

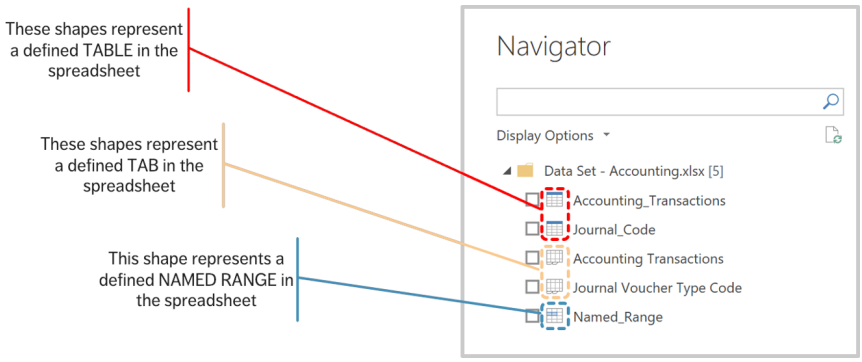
We might want to "tweak" the data prior to loading it into Power BI. This can be accomplished using the **Power Query Interface**.

Let us illustrate the steps with one of the Borealis Terraformers datasets.

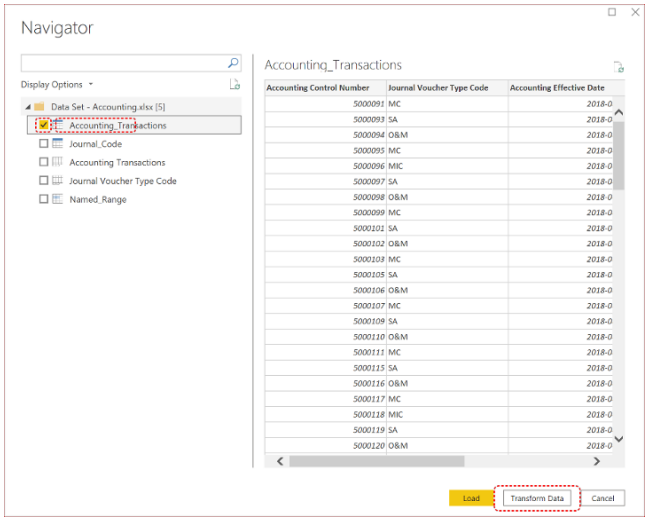
6: As below:



1. ensure datasets are saved in a known, navigable location;
2. open Power BI;
3. close the green "Hello" screen (or yellow, if older version);
4. in the "Home" ribbon at the top of the Power BI screen, there is a clickable region named "Get Data" – activate it to bring up several data format options;
5. among all options, select Excel;[6]
6. click on `Data-Set-Accounting.xlsx` and select "Open";
7. click on the first TABLE named "Accounting_Transactions" in the "Navigator" window (**but don't select the checkmark yet**) – a display of the data appears (see image at the top on the next page);
8. click on the other options to see their displays;



9. go back to "Accounting_Transactions" and select the checkbox (**don't click on "Load" yet**);
10. we want to modify the data before loading it, so we edit the transformation by clicking on "Transform Data";



11. the Power Query Editor appears as in Figure 13.3.

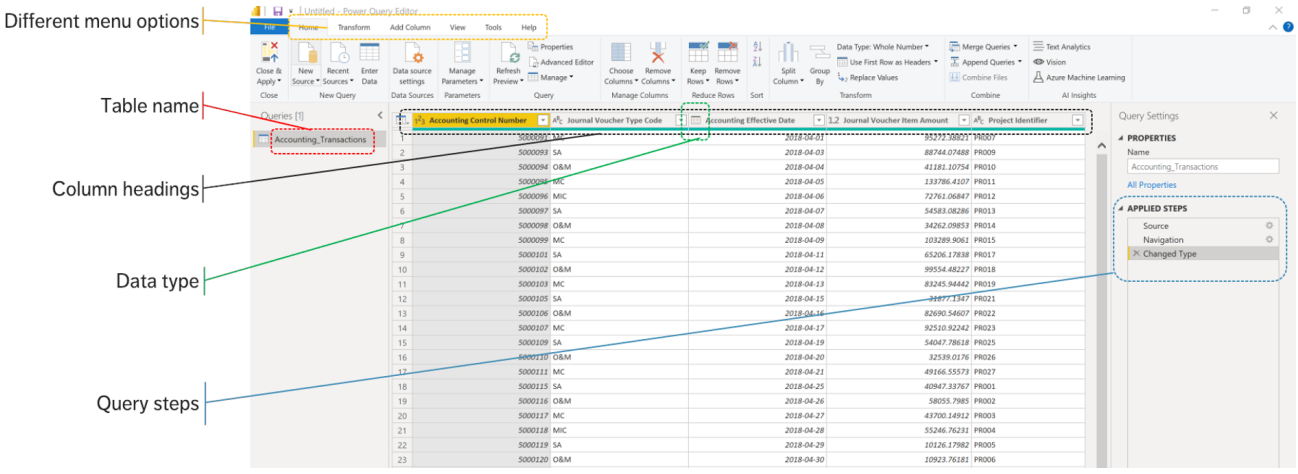


Figure 13.3: Power BI's Power Query Editor for the `Accounting_Transaction` table.

We proceed with the following steps to **transform the data**:

1. change the name of the table – double click (or right click) on the table name and edit to remove the underscore, yielding the "Accounting Transaction" table;
2. change the name of the "Accounting Effective Date" column to "Effective Date" and "Journal Voucher Item Amount" to "Item Amount" (as in the previous step);
3. remove the dates from March 2018 in the "Effective Date" column – click on the drop-down arrow by the column header, select "date filters", then "after", then select March 31, 2018, then "OK";
4. close and apply the transformation – click on "Close and Apply", and
5. save the `.pbix` file.

Prior to closing and applying the transformation, we see that there are new steps in the query step box.

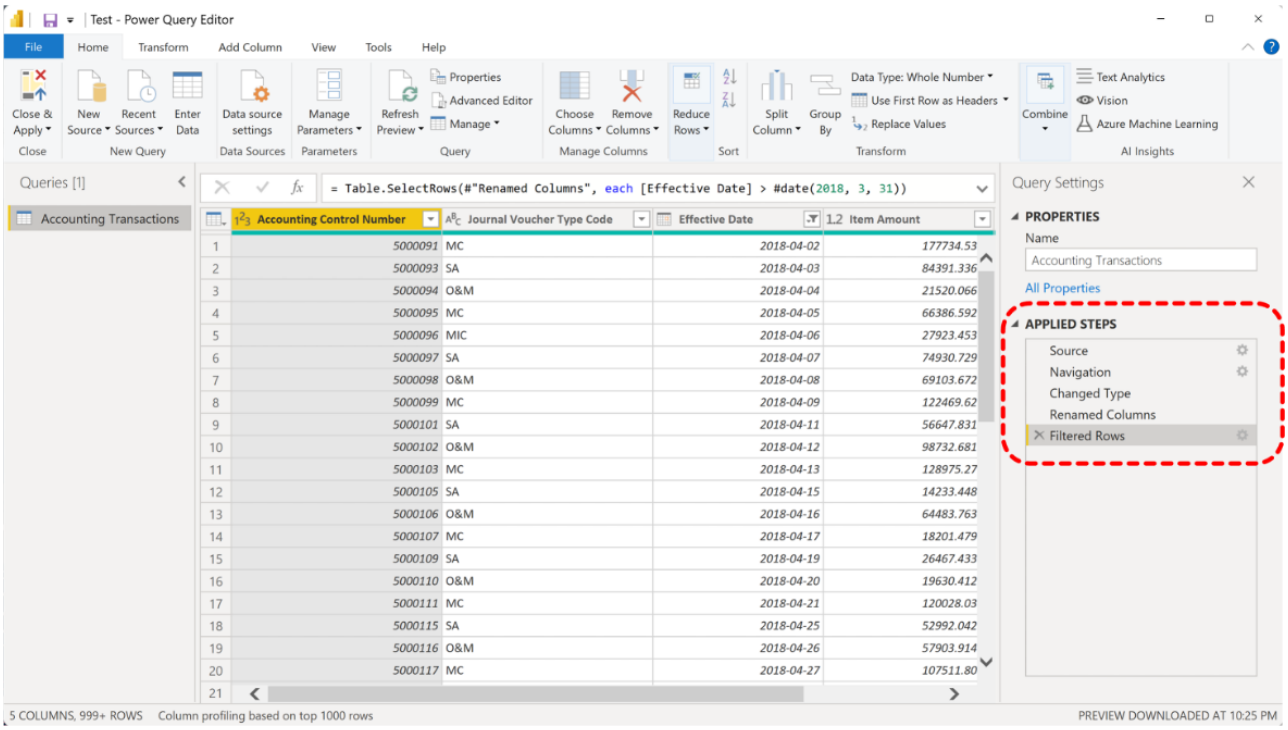


Figure 13.4: The Power Query steps can be saved and re-traced.

Exercise: use the same steps as above to load the "Journal_Code" table from the same Excel file (`Data-Set-Accounting.xlsx`) into Power BI and rename it "XREF Journal Code".[7] Other than renaming, there is nothing more you need to do to this table inside Power Query.

7: More on why we use "XREF" later.

Linking Datasets

We now have two tables. We will use Power BI to **link** them (assuming that there is a natural way to do so, i.e., that they have one column in common):

1. click on the "Model" button in Power BI;
2. two tables appear, but they are not yet linked (see Figure 13.5);

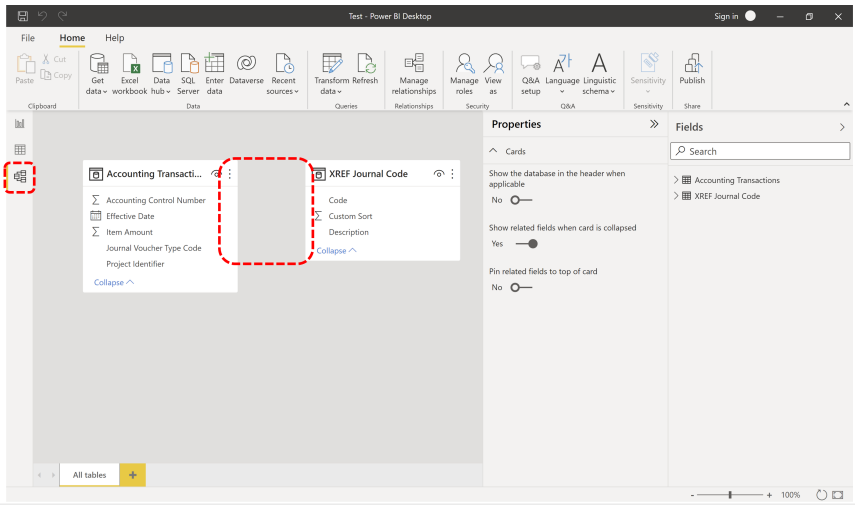


Figure 13.5: Two unlinked tables in the data model space.

3. left-click on ‘Code‘ in the "XREF Journal Code" table, drag across to ‘Journal Voucher Type Code‘ in the "Accounting Transactions" box, and release to see the new connection in the data model (see Figure 13.6).

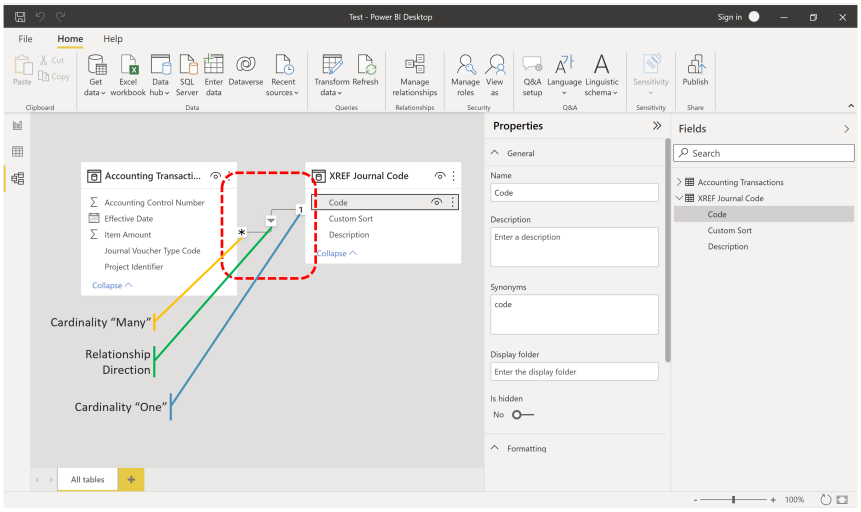


Figure 13.6: Two linked tables in the data model space.

Double clicking the connection/line between the tables brings up the relationship editor (see Figure 13.7). The greyed-out columns show which variables are **connected** in the data model, while the **cardinality** describes how the two tables are related:

- one-to-many / many-to-one,
- one-to-one, or
- many-to-many (which is generally not recommended, for a variety of reasons).[8]

8: See `Getting Insight From Data` for (slightly) more information on the topic [1].

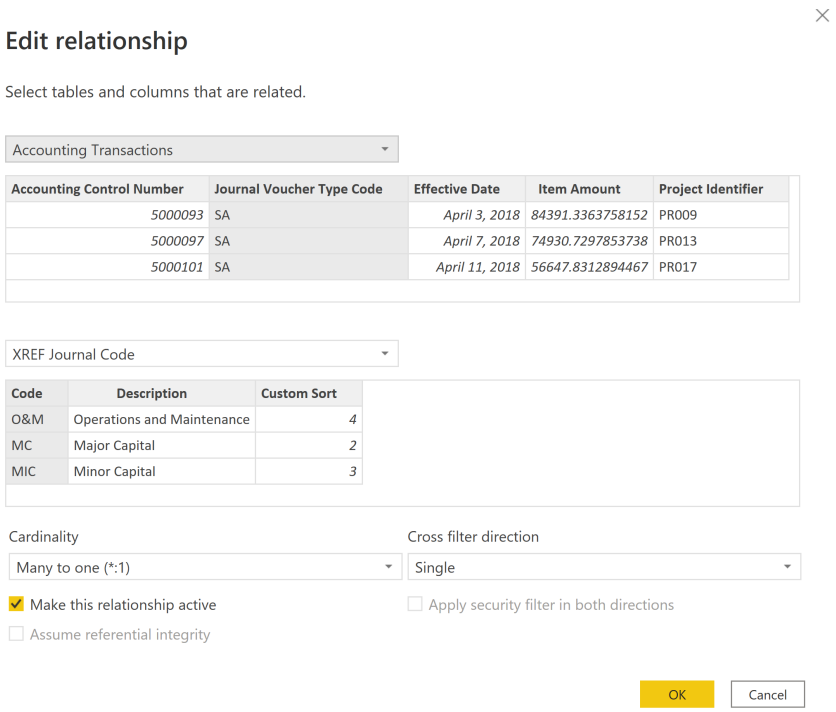


Figure 13.7: The edit menu for linking tables.

The **cross-filter** direction is the direction in which Power BI applies a filter, from one table to another. We leave as "Single" for now, but note that this could be modified if required.

We are now in a position to create calculations and (most excitingly) charts.

13.2 Calculations and Charts

In Excel, formulas are entered in cells; in Power BI, the corresponding quantities are independent calculations made using the **DAX** language.

DAX is a collection of functions, operators, and constants, which are used to create new data entities (measures, columns, tables); in effect, DAX helps to create new information from data already found in the model (it is a form of data reduction) [129].

Calculated Columns, Measures, & Tables

Consider the following typical DAX formula:

```
Total Amount = sum('Accounting Transactions'[Item Amount])
```

In this example:

- the measure **name** is `Total Amount`;
- the operator = indicates the **beginning** of the formula – when calculated, the formula returns a single **result**;
- the **function** (in this case, `sum`) contains a lone argument, which is contained inside the parentheses `( ... )`;
- the **table** from which the data is taken is found just to the left of the brackets `[ ... ]` (in this case, `Accounting Transactions`), and
- the **referenced column** from the table is found inside the brackets `[ ... ]` (in this case, `Item Amount`).

The single quotation marks around the table name are needed because it contains a space character; they would not be needed around the column name if it contained a space character, however. As an example, the following would be acceptable:[9]

9: Assuming these were the actual table and column names.

```
Total Amount = sum(Accounting_Transactions[Item Amount])
```

In Excel, calculations are added to a cell (to be sure, we can also add a calculation to every cells in a range, but it's the same calculation, possibly using different inputs). In Power BI a calculation can "live" in one any of three places (not including in Power Query, but this is a topic for a later section).

A calculation can create:

- a new data **column** (a calculated column);
- a new data **value** calculated solely to be used in a visualization (a measure), or
- a new data **table** (a calculated table).

These are accessed *via* the Modeling ribbon (see below).

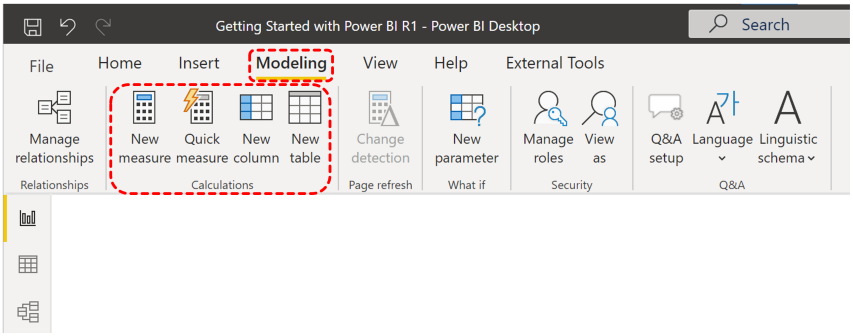


Figure 13.8: Calculation buttons found in the Modeling ribbon.

Let us take a quick look at each of the types.

Calculated columns are added to an existing data table; they are evaluated for each row in the table, immediately after 'Enter' is hit to complete the formula. These columns are saved into the model, so they take up space in the model (a calculated column on a table with 1M rows adds 1M new data points to the table). Calculated columns are often used to filter on their result, rather than simply as a calculated result (in slicers, for instance).

Calculated measures, on the other hand, are only evaluated when used in a visualization (or when the visualization is rendered); they are therefore not saved anywhere (aside from the cache). This helps explain why measures are generally preferred to calculated columns, but there are trade-offs related to performance (report runtime vs. pre-processing), storage space, and what type of expressions can be used in the formula.

Finally, **calculated tables** are new tables of data, calculated from existing tables in the data model. As is the case with calculated columns, they are evaluated for each row in the source table, immediately after 'Enter' is hit to complete the formula, and they are saved into the model, which means that they take up space. They can be linked to any other table in the data model.

When do we use calculated columns over measures? Apart from space considerations,[10] it is often the case that both are reasonable options. In most situations, the **computation needs** make the determination.

10: Recall that measures are more computationally efficient than calculated columns and that they do not add additional data into the model.

For instance, we would use calculated **columns** to accomplish some tasks:

- place the calculated results in a slicer, or in the axis of a chart, or use the result as a filter condition in a DAX query;
- define an expression that is strictly bound to the current row (for example, `Cost * Volume` does not work on an average or on a sum of two or more columns);
- categorize text or numbers (for example, a range of values for a measure, a range of customer ages, such as `0-18`, `18-25`, etc.).

One the other hand, we would use **calculated measures** whenever a resulting calculation needs to be displayed in the values area in the plot area of a chart, such as in calculating the cost percentage on a certain subset of the data. As measures can incorporate data from different tables, they do not therefore "belong" to a specific table in the same manner as a calculated column does. Consequently, it is a good practice to create a separate "home" (table) for calculated measures.

We only discuss **calculated tables** briefly for the moment. They could prove useful as intermediate steps in data analytic endeavours, or simply when:

- the desired calculated result is a matrix of some form;
- data tables need to be summarized;
- data needs to be isolated from other tables;
- new relationships need to be created in the data model.

Examples of Calculated Columns and Measures

As a first example of a DAX expression, we divide the "Item Amount" column by 1000 to obtain a new calculated **column** that represents the item amount in units of 1000$:

1. click on the data button to access the table view (note the available tables and columns in the Fields panel, at the right of the screen);

2. select the “Accounting Transactions” table in the Fields panel (by clicking on it) to produce a new ribbon item called “Table tools”;

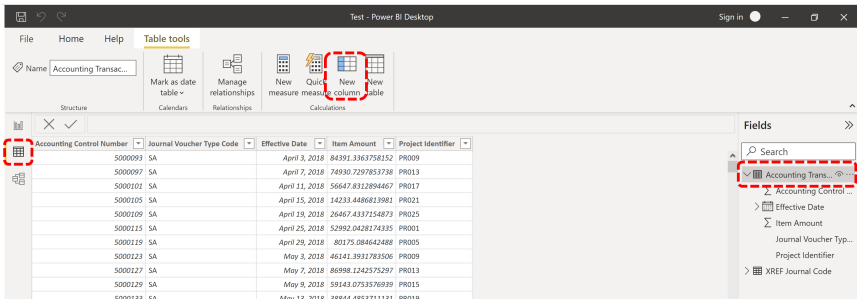


Figure 13.9: Accessing the table tools.

3. click on “New Column” (you can also access this functionality from the “Home” ribbon);

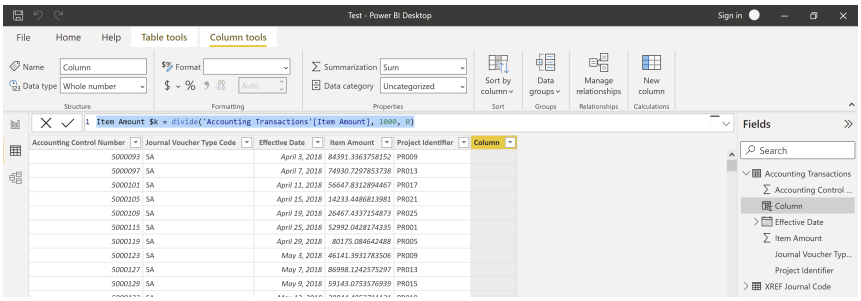


Figure 13.10: Accessing the column tools.

4. Type in the formula below in the space reserved to that effect, then hit “Enter” (the auto-complete functionality will not appear when cut-and-pasting):

```
Item Amount $k = divide('Accounting Transactions'[Item Amount], 1000, 0)
```

We can see a new column in the Fields panel:

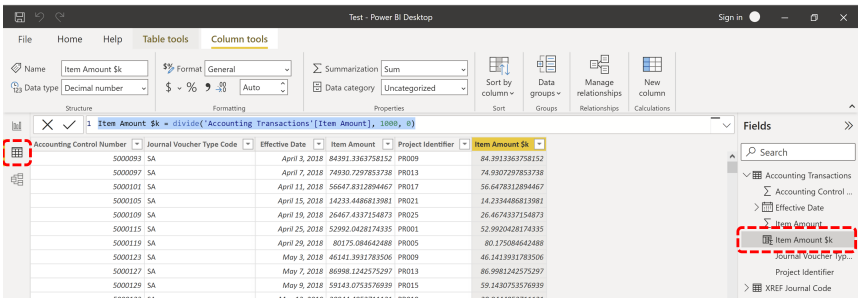


Figure 13.11: Creating a new column with DAX.

We can do a bit of housekeeping to **clean things up** before we create a chart:

1. click on the column header for `Item Amount $k` (alternatively, select that variable from the Fields panel, on the right), bring up the “Column tools” in the top ribbon;
2. the “Format” dropdown menu is found in the “Formatting section” of the “Column tools” ribbon; click on the menu and change the format to “Currency”;

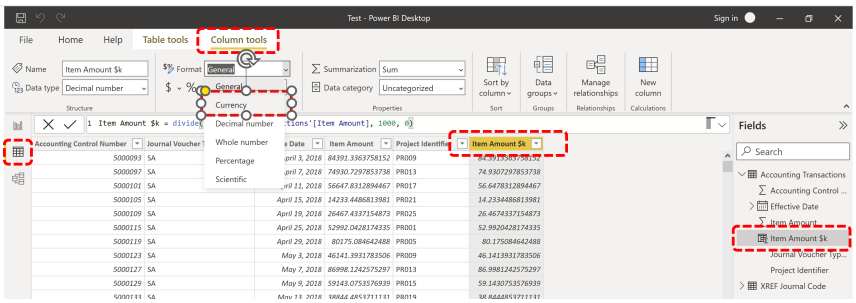


Figure 13.12: Cleaning up the new column.

3. to display the currency with the usual 2 decimal places, locate the dropdown box that says "Auto", and change the value to "2".[11]

11: This can also be done for the original `Item Amount` column.

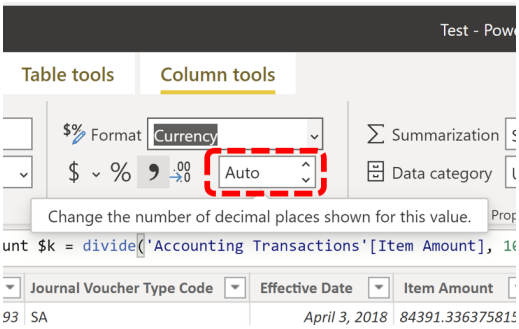


Figure 13.13: Formatting the number of decimal places.

We can create a simple Power BI display using the new column:

1. access the "Report" screen;
2. in the Visualizations panel, select the "Stacked Bar Chart" icon;[12]
3. from the "XREF Journal Code" table (in the Fields panel), drag the `Description` column on the "Y-Axis" field;
4. from the "Accounting Transactions" table, drag the `Item Amount $k` column onto the "X-Axis" field.

12: It is the first icon in the list.

The resulting chart is shown below.

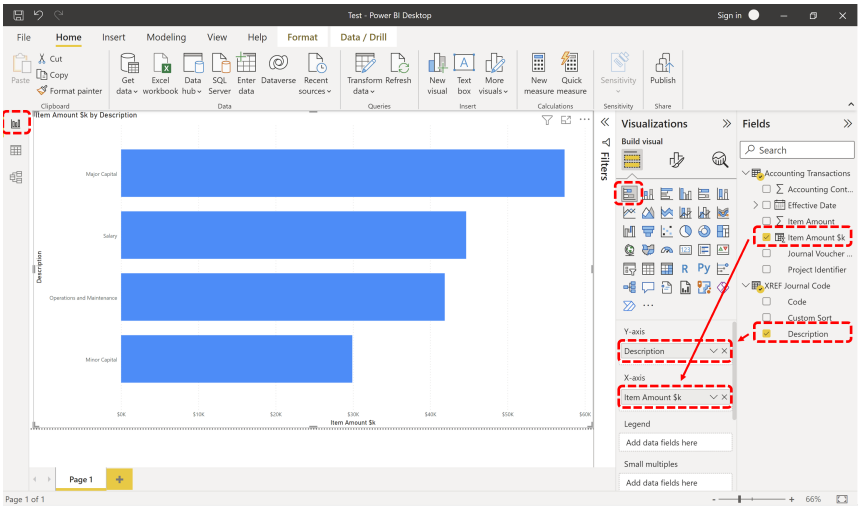


Figure 13.14: Our first Power BI chart!

The quantity that is displayed on the $x-$axis is the sum of `Item Amount $k` for all items with the same `Description` in the "Accounting Transactions" table; that is the default statistic.

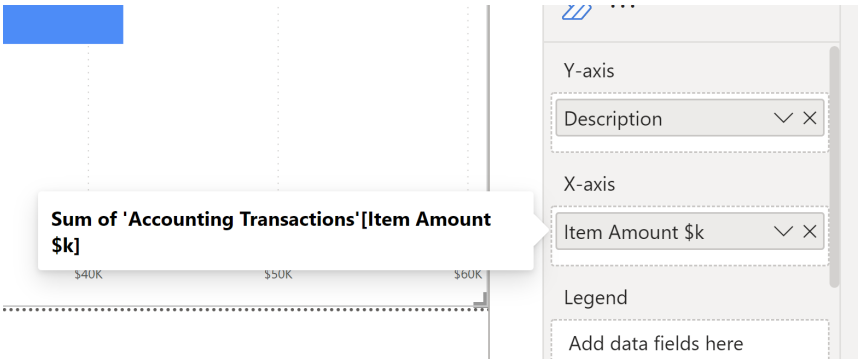


Figure 13.15: The sum of a numerical value is the default statistic for a number of Power BI charts.

But it does not have to be the sum; right-clicking on the "X-Axis" menu brings up the other statistics.[13]

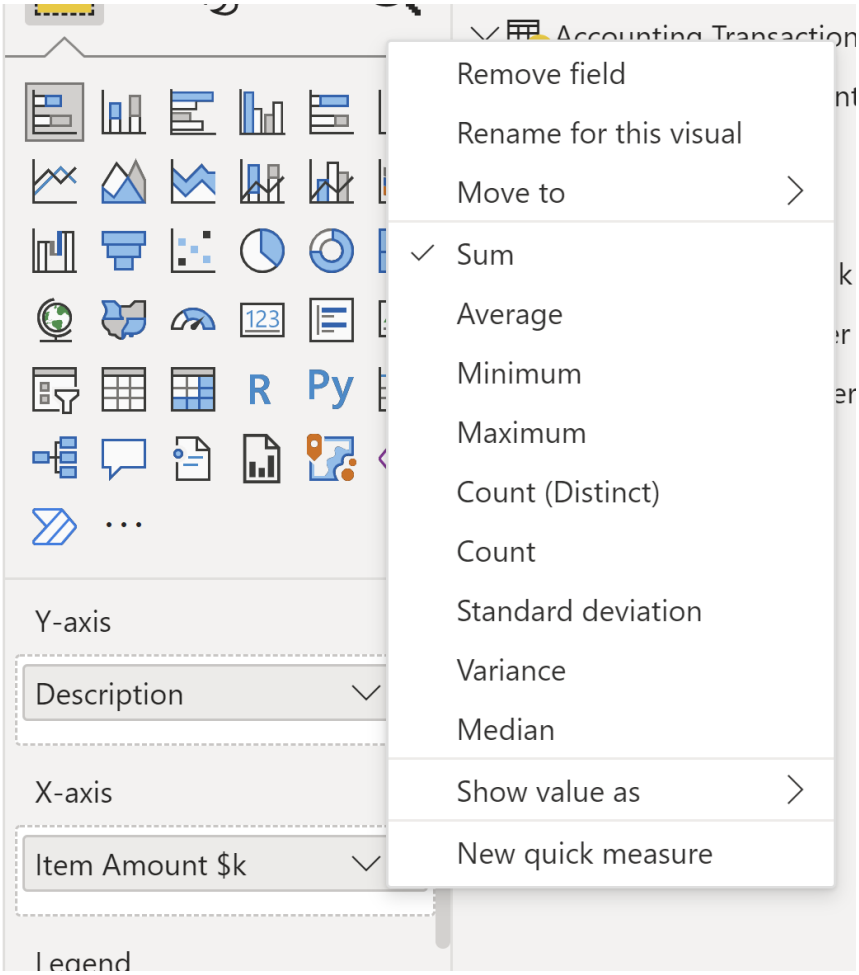


Figure 13.16: A list of the built-in statistics for numerical variables.

For instance, if we select the average `Item Amount $k` per row in "Accounting Transactions", we get the following chart of Figure 13.17.

13: For text data, the only options are "Count" and "Count (Distinct)".

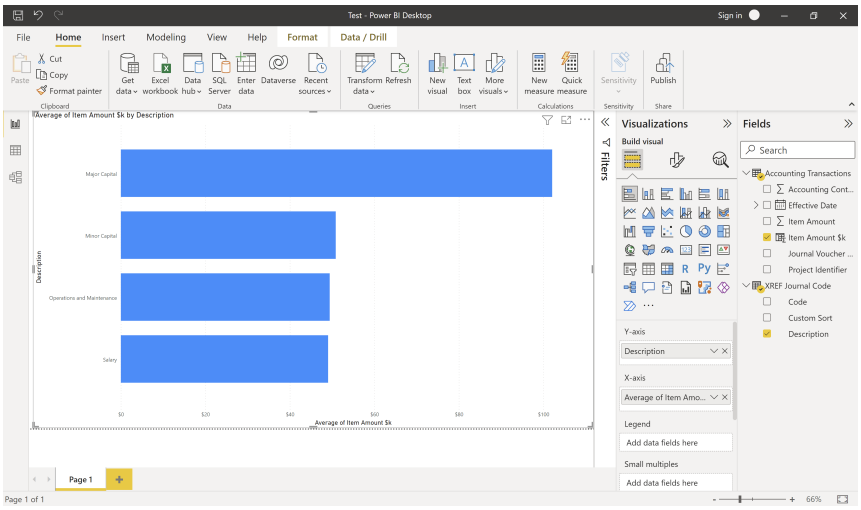


Figure 13.17: A new Power BI chart, using the same underlying data, but a different statistic: 'average' instead of 'sum'.

We can also format the aesthetics of the chart. This can be accomplished by first selecting the chart (simply by clicking on it), and then pressing on the "Format your visual" icon in the Visualization panel (the icon with a paintbrush and a chart). This will bring up a number of items, the exact listing depending on the chart type.[14] For instance, we can modify the colour of each bar in the chart through the "Bars" option.

14: It is important for budding Power BI users to explore the ways in which these do differ from chart to chart.

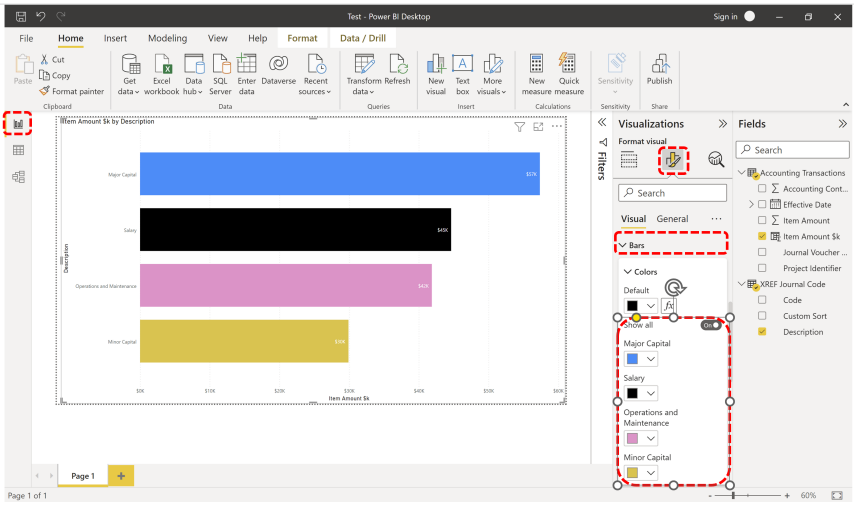


Figure 13.18: Changing the aesthetics of the Power BI chart.

We can easily modify the chart, by selecting a different chart type in the list: for instance, a **donut chart**, as in Figure 13.19.

Let us now build a **measure**. The best practice in such a case is to create a "Measure Table" in which the measure can reside:

1. in the "Home" ribbon, click on the "Enter Data" icon;[15]

15: In the "Data" section.

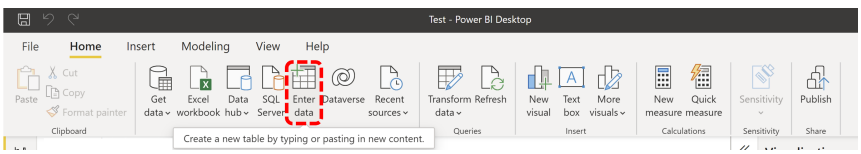


Figure 13.20: On the way to a calculated measure.

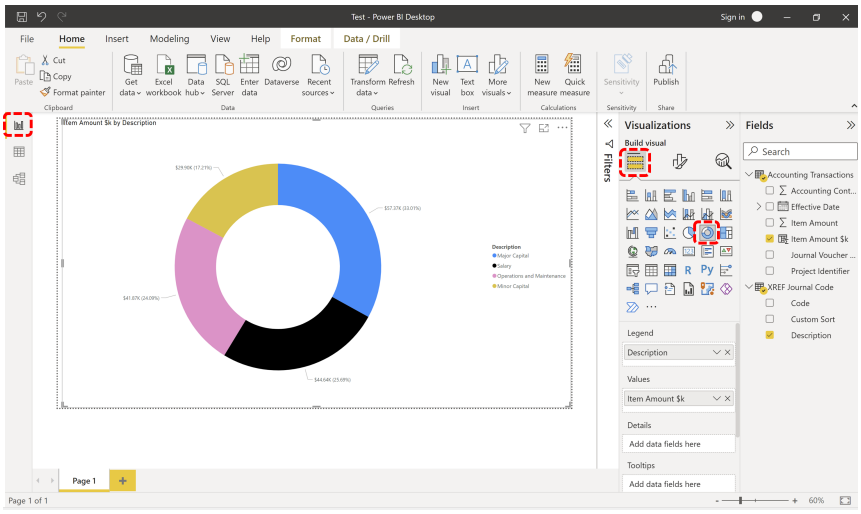


Figure 13.19: Another Power BI chart!

2. ignore everything except the "Name" field at the bottom (left);
3. override the name "Table" with "_Measures"[16]

16: Making sure to include the underscore "_".

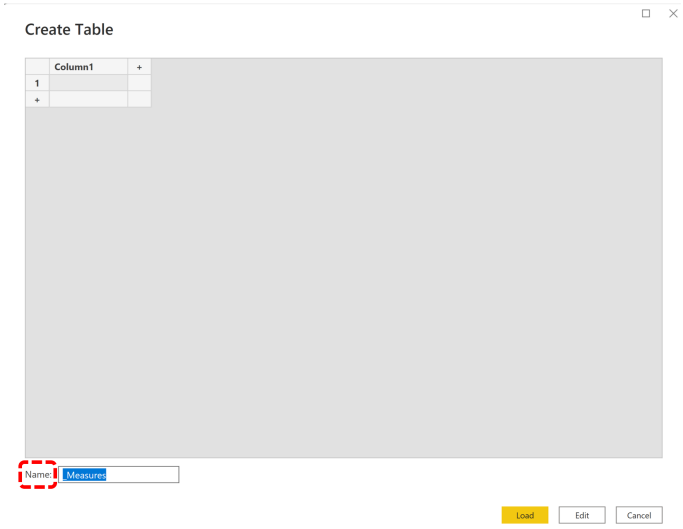


Figure 13.21: On the way to a calculated measure.

4. click on "Load" – the measure table appears at the top of the list in the Fields panel (because of the underscore).

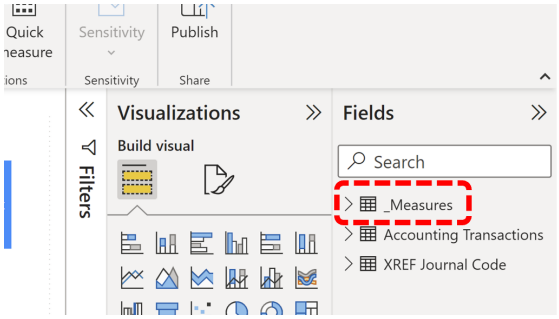


Figure 13.22: On the way to a calculated measure.

Next, we populate that table with a measure that once again uses the DIVIDE function:

1. click on the new "_Measures" table in the Fields panel;[17] .
2. click on "New measure";

17: The top ribbon will change to "Table tools"

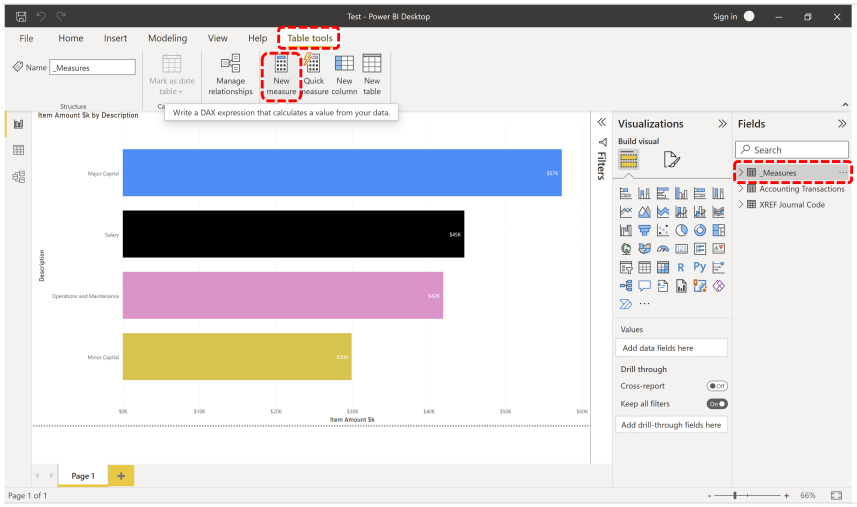


Figure 13.23: Creating a new measure.

3. start to TYPE the following (do not cut and paste):

```
Item Amount $k Measure = DIVIDE(item ...
```

Power BI is not completing the string in the same way as it did when we created a calculated column – why should that be so?

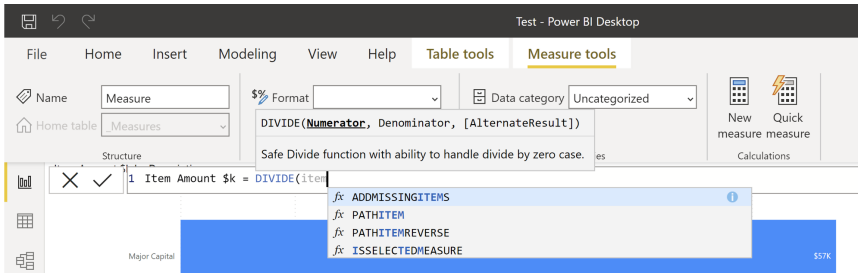


Figure 13.24: Trouble in paradise?

The following would provide auto-complete options, however, as seen in Figure 13.25:

```
Item Amount $k Measure = DIVIDE(SUM('Accounting Transactions'[Item Amount $k]), 1000, 0)
```

By default, measures perform an operation on an **entire column**, not row-by-row (there are exceptions, which will be covered at a later stage). In this case, we could also DIVIDE the average, max, min, standard deviation, etc. by 1000;[18]

18: Even though the measure was created from a column that was formatted as currency, we still need to separately format the measure as currency.

4. complete the measure formula and create a chart from it as we did for the calculated column, as in Figure 13.26 (note the icon next to the measure in the Fields panel, on the right).

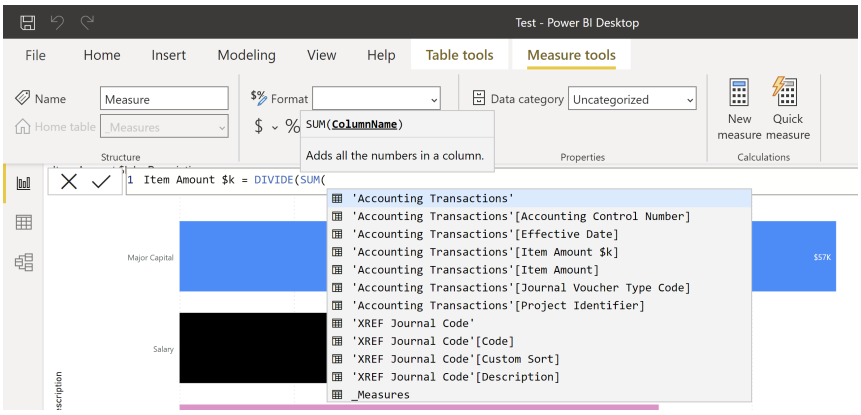


Figure 13.25: No trouble in paradise.

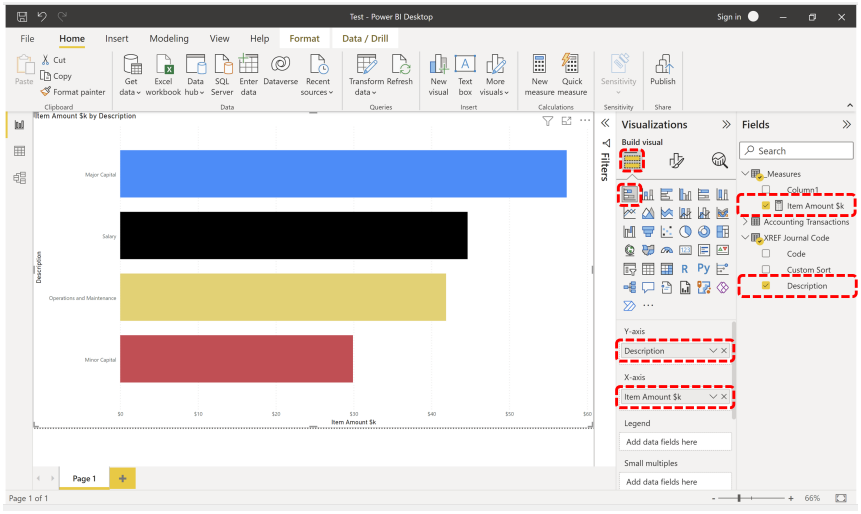


Figure 13.26: A chart created from a measure.

Mathematical and Logical Operations

In the preceding text, we showed a standard use of SUM and DIVIDE; the expected standard **arithmetic expressions** (SUM, DIVIDE, PRODUCT, MAX, MIN, AVERAGE, etc.) are all available in DAX.

Most arithmetic expressions are straightforward, but some are very specific, such as the DIVIDE operation, say. It is easy to manually divide using a `/` – we all do it and most of the time no problems will arise – but there is a **failure option** built into DIVIDE that provides alternate results, such as `0` for a "divide by 0" error.

```
DIVIDE(<numerator>, <denominator> [,<alternateresult>])
```

It is worth spending a bit of time playing around with these expressions to determine their full functionality.

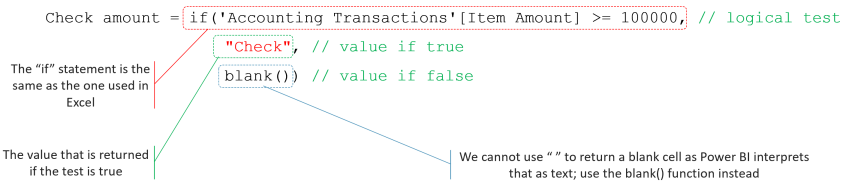
We finish this section on DAX by introducing **logical operations**, such as the AND or the OR operation. There are multiple ways of achieving this; we use a simple IF statement to select a value when a logical condition is met.

As an example, assume we want to flag a transaction when the Item Amount value is above $100,000. We do this by adding a new calculated column with the following logic:

- if the Item Amount value is equal to or greater then $100,000, then we add the word "Check";
- if the value is below $100,000, then we leave the value in the new column blank.

This can be accomplished as follows:

1. click on the "Accounting Transactions" table (in the Fields panel);
2. click on "Column Tools", then "New Column" (in the top ribbon);
3. type in the following formula:



4. press "Enter" to complete the calculation.

We can now create charts using this new column:

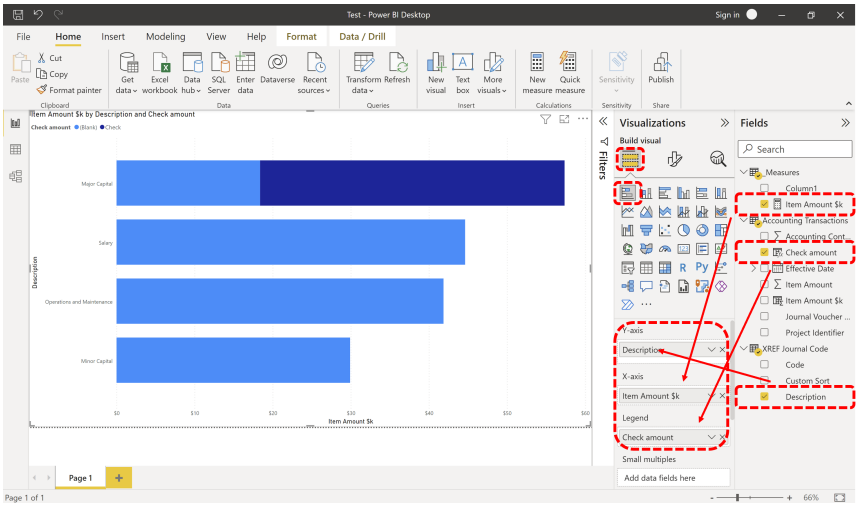


Figure 13.27: A chart using the result of a logical computation.

Filtering and Slicing

As we can see in the previous examples, we calculate measures and columns by performing an operation on an entire data column. What if we only want to accomplish this on a subset of the data? Here are two ways to do so:

1. build the **filter** into the calculation itself;
2. filter the result using a **slicer** or using the built-in filter function.

We are going to provide an example of each.

First, we build a filter directly into our calculations, using probably THE most useful DAX operation ("CALCULATE").

Suppose we want to add the dollar amounts, but only those that arise from salaries (in the "Accounting Transactions" table):

1. click on the "_Measures" table (in the Fields panel);
2. create a new measure, and type the following at the prompt:

```
Total Salary = CALCULATE(Sum('Accounting Transactions'[Item Amount]),
                         'Accounting Transactions'[Journal Voucher Type
                             Code]="SA")
```

This is the operation that is being calculated

This is the filter that is being applied

As a quick aside, we can include a carriage return in a formula by holding down "Shift" and hitting "Enter"; it is good practice to use carriage returns to ensure that measures and columns are easy to read. Another best practice is to **comment code**, which is done in Power BI by typing 2 forward slashes "//", as below:

```
Total Salary = CALCULATE(SUM('Accounting Transactions'[Item Amount]), //this is a comment
                         'Accounting Transactions'[Journal Voucher Type Code]="SA") //another comment
```

We can easily create three new measures Total Major Capital, Total Minor Capital, and Total O&M, corresponding to Journal Voucher Type Codes MC, MIC, and O&M, respectively (remember to format the measures again), and build a clustered bar chart from them.

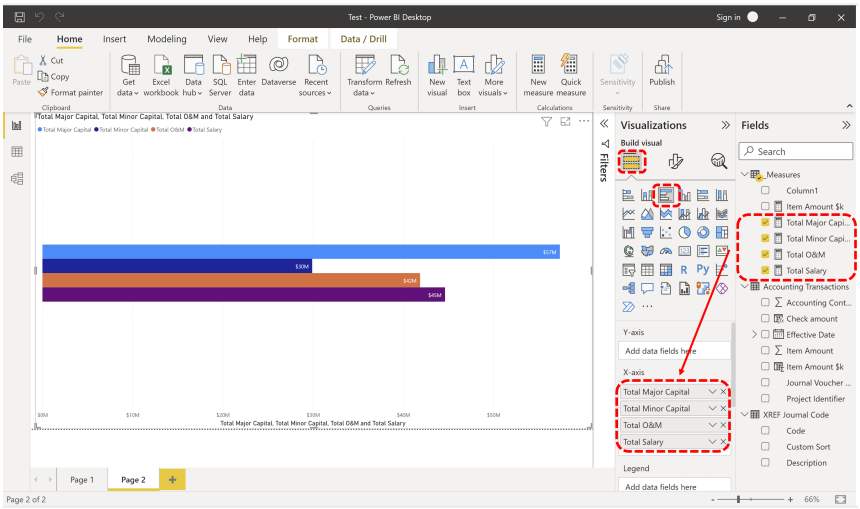


Figure 13.28: A bar chart built entirely out of measures.

This is not the only way to do this, of course (we have created the chart in a previous section); the nice thing about using measures is that they are easy to re-order (by changing the order in the "X-Axis" field.

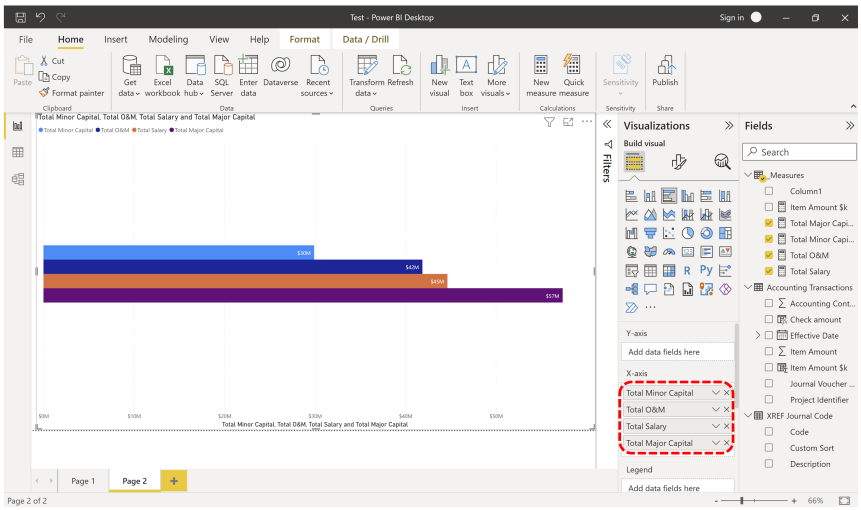


Figure 13.29: Re-ordering a bar chart built entirely out of measures.

We can also filter visualizations by using slicers. In fact, the default setting for Power BI is to allow a user to click on any chart which will in turn filter other charts.

We set-up a simple slicer using "Project Identifier" by clicking on the slicer icon in the Visualization panel, and then dragging the "Project Identifier" column into the "Field" box.

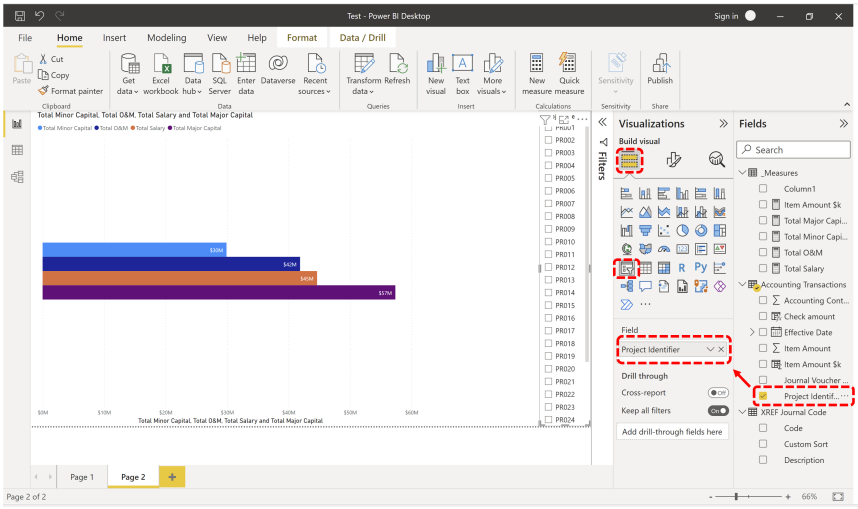


Figure 13.30: Using a slicer on a chart.

The chart can be updated for various combinations of project identifiers. We can also filter visualizations by using the Filter panel: click on the desired chart and expand the filter pane, then drag the column to use as a filter in the corresponding slot (see Figure 13.31).

13.3 Organizing Data

Organizing the data is a critical part of building reports and dashboards; Power BI has some powerful built-in capabilities to accomplish this task.

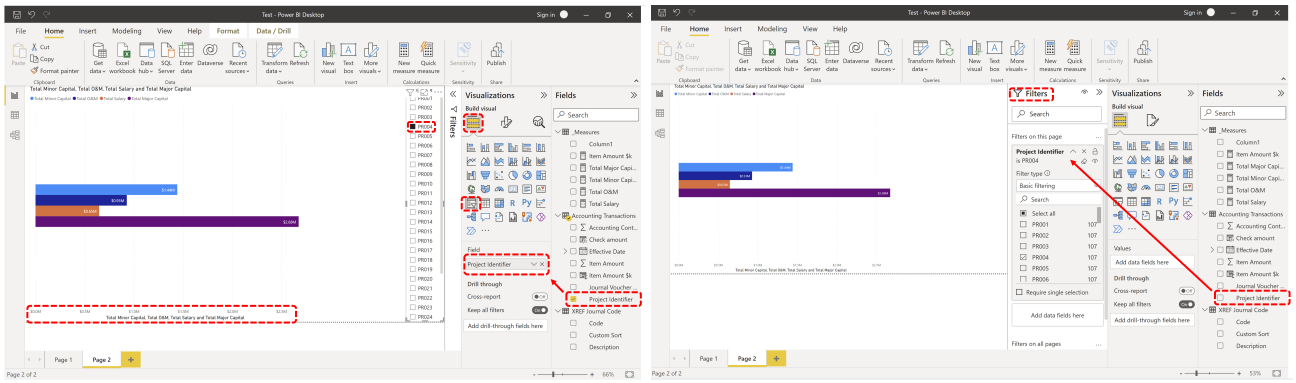


Figure 13.31: Chart for one specific project (PR004).

Hierarchies

Hierarchies provide a way to order data levels. They allow us to **summarize** data in different ways. When a date is imported in Power BI, a **date hierarchy** is automatically created: **Year** > **Quarter** > **Month** > **Week** > **Day**.

For instance, the hierarchies created in Exercise 10 can be navigated up and down, as shown in Figure 13.32.

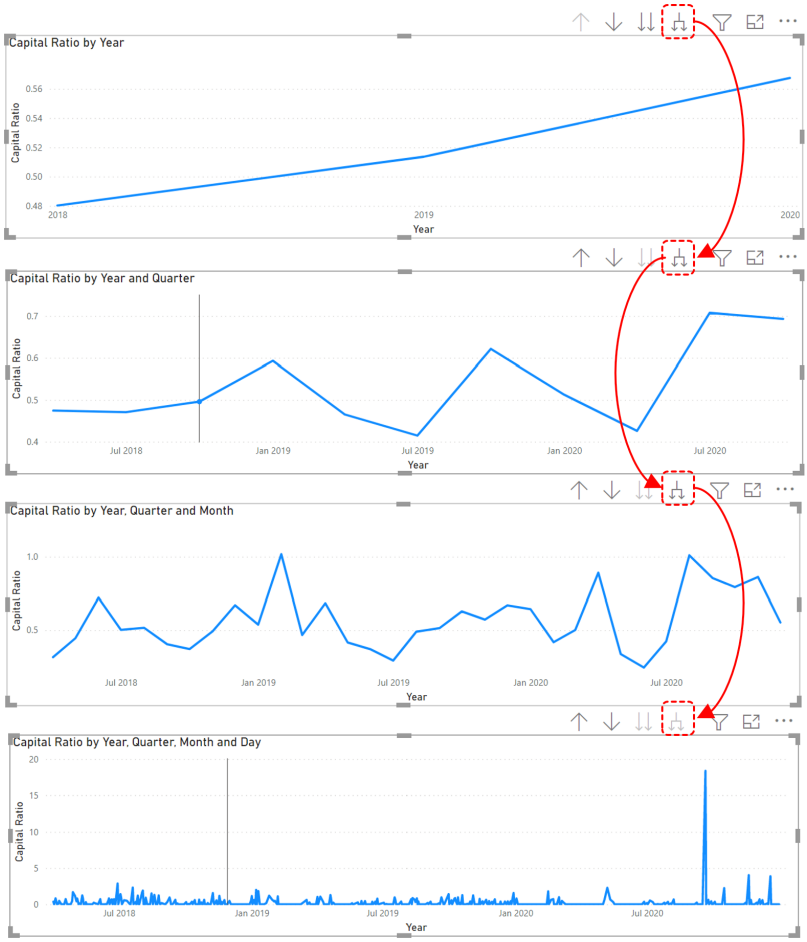


Figure 13.32: Illustration of the hierarchical structure of dates in Power BI.

It is a very simple process to create a **custom hierarchy** in Power BI: simply drag a column on top of another. As an example:

1. enter the report view;
2. left-click on "Project Identifier" (or click on the 3 dots . . .) and select "Create Hierarchy";
3. left-click on "Journal Voucher Type Code" (or click on the 3 dots . . .) and select "Add to Hierarchy > Project Identifier Hierarchy";

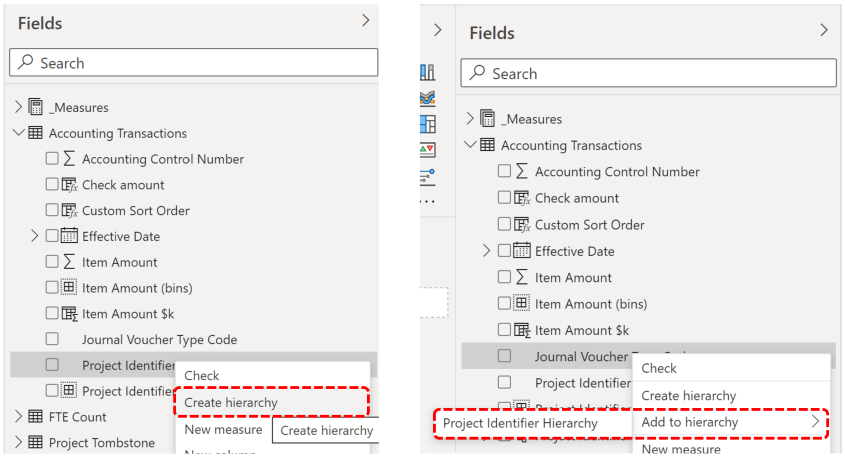


Figure 13.33: Building a custom hierarchy.

4. create a column chart and add the hierarchy to the X-Axis and "Item Amount" to the values.

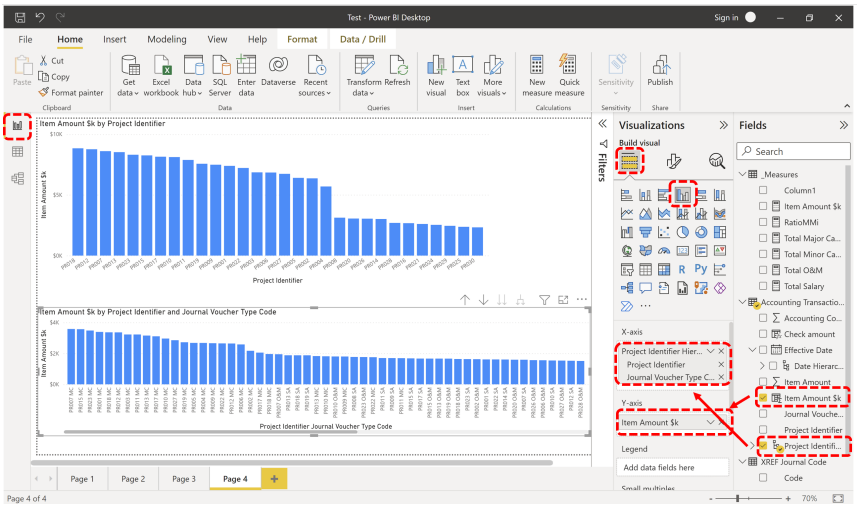


Figure 13.34: Bar chart with a custom hierarchical structure (two levels: projects and codes).

We can also use hierarchies with matrices and arrays, as in the example below:

1. insert a matrix on the canvas (use the icon in the Visualization panel, as in Figure 13.35);
2. add the hierarchy to the "Rows" field;
3. add "Item Amount" to the "Values" field;

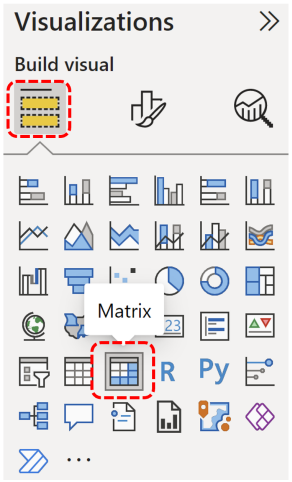


Figure 13.35: Matrix icon in the visualization panel.

4. open the "Effective Date" Hierarchy and drag the "Year" to the "Columns" field, in essence creating a **pivot table**.

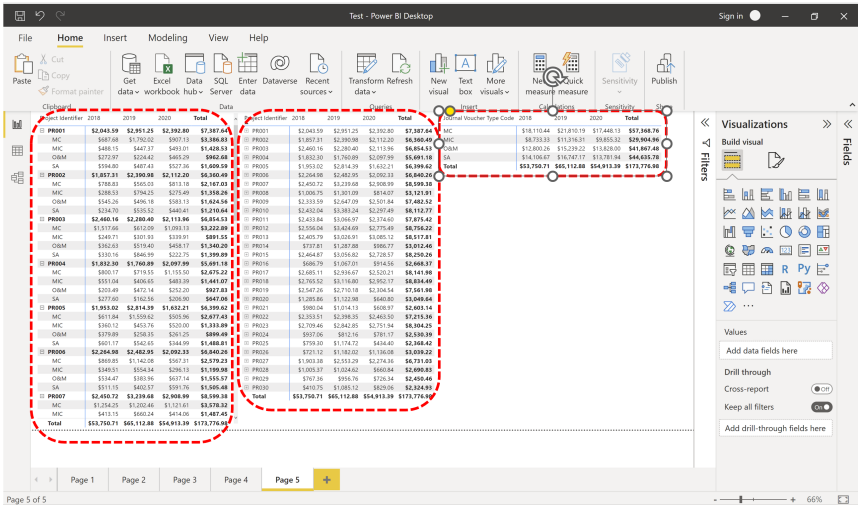


Figure 13.36: Pivot tables in Power BI.

Groups

We use **groups** when we want to categorize or classify similar rows of data in a table. The built-in Power BI function can do so for two types of data:

- grouping **text** or **categories** (such as grouping all EX equivalent job categories together, say), and
- grouping or "bucketing" **numerical intervals** (such as creating an axis where we can count the number of items that fall in a particular set of ranges: 0-100, 101-200, 201-300, > 300).

Neither approach is perfect, but 90% of new users' requirements will be covered; we will discuss more complex scenarios at a later stage.

Let us start with an example where we group **categorical data** *via* the following steps:

1. in either the Data or Report views, click on the "Accounting Transactions" table in the Fields panel;
2. click on the "Project Identifier" column and either click on "Data groups" in the "Column tools" menu or the 3 dots ... next to the "Project Identifier" name in the Fields panel, and select "New group";
3. in the list called "Ungrouped Values", click on PR001, hold down SHIFT and click on PR0010 to select that block;[19]

19: 'The 'CTRL" key also works for individual items.

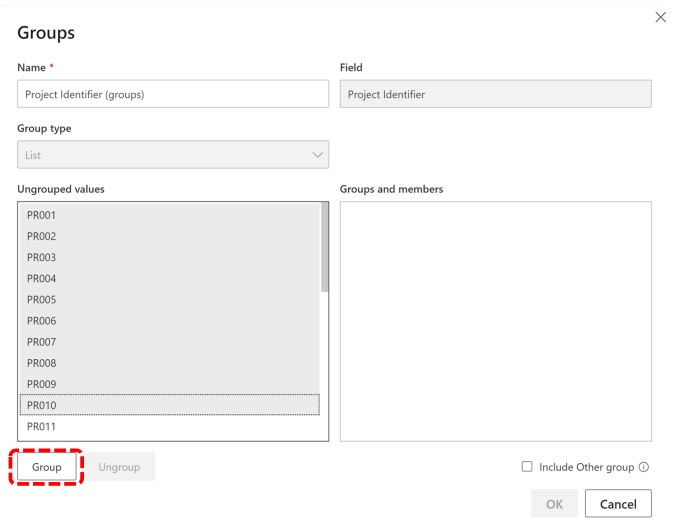


Figure 13.37: Groups dialog box (categorical).

4. click on "Group" and rename the Group in the "Groups and members" field to "Group 1";[20]
5. repeat the process for PR0011 - PR0020 (Group 2) and PR0021 - PR0030 (Group 3);

20: Double-click on the concatenation of all names to edit the group name.

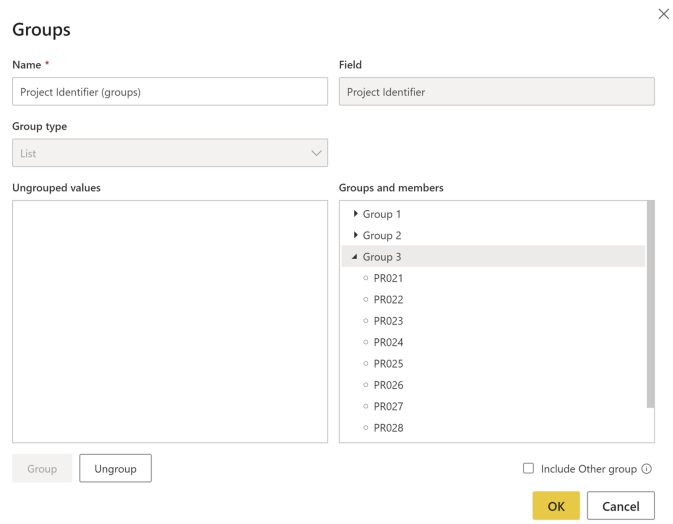


Figure 13.38: New groups: Group 1, Group 2, Group 3.

6. Hit "OK".

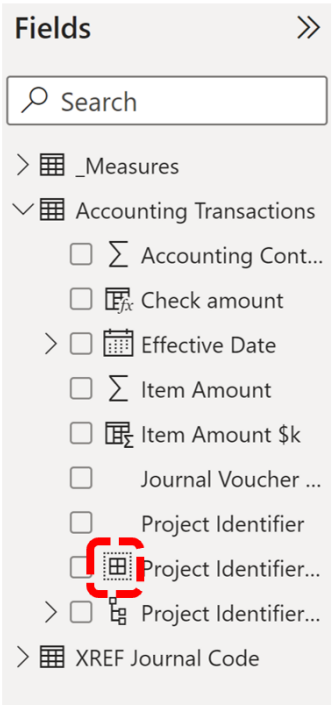


Figure 13.39: Project Identifier grouped variable.

There is now a new column in the "Accounting Transactions" table, displaying the Project Identifier group associated with each transaction (row). This column can be used in a chart axis, slicer, filter, or as data (for count operations only, however).[21]

21: To obtain an "Other bucket", check the "Include other group" checkbox in the Groups dialog box.

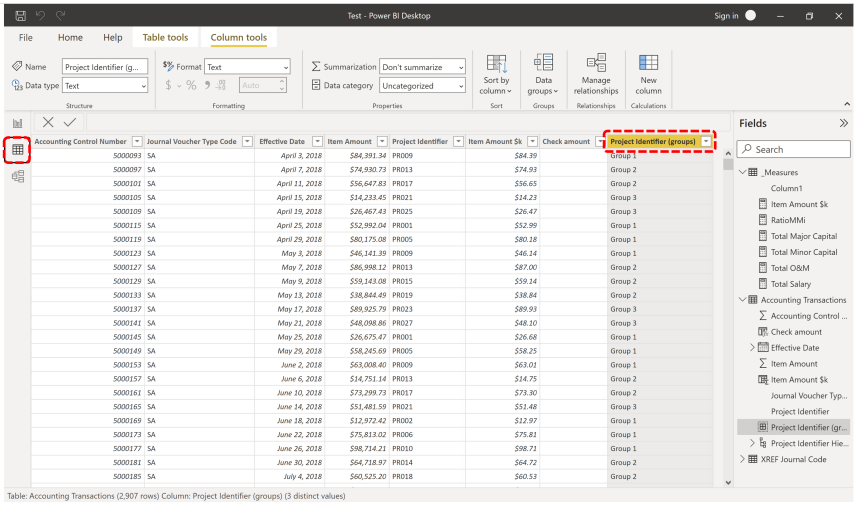


Figure 13.40: New variable in the Accounting Transactions table.

Next, we provide an example that illustrates how to group **numerical data** into intervals:

1. in either the Data or Report views, click on the "Accounting Transactions" table;

2. click on the "Item Amount $k" column and as per the previous example either click on "Data groups" in the "Column tools" menu or the 3 dots . . . next to the "Item Amount $k" name in the Fields menu, and select "New group";

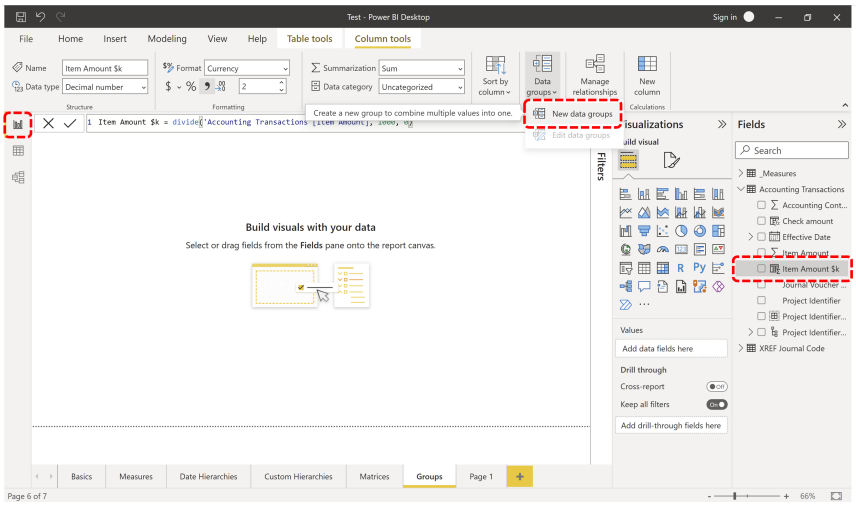


Figure 13.41: Reaching the groups dialog box (numerical).

3. There are 2 grouping options for numerical data (see "Group Type" dropdown menu);
   a. **lists** work in the same way as the previous example where we were grouping categorical project identifiers;
   b. **bins**, which we will use for this example, are used to create "buckets" (intervals) of column values, which can either be determined using the "Size of Bins" or the "Number of Bins" parameters.

"Size of bins" is typically more useful: in this example, using "Bin count" = 50 yields a "Bin size" of $3828.27, which does not particularly roll off the tongue. Setting the "Bin size" to something sensible (say, 10) will provide reasonable (and easily interpretable) intervals.[22]

22: Remember, the field we are working with is already diving the "Item Amount" by 1000.

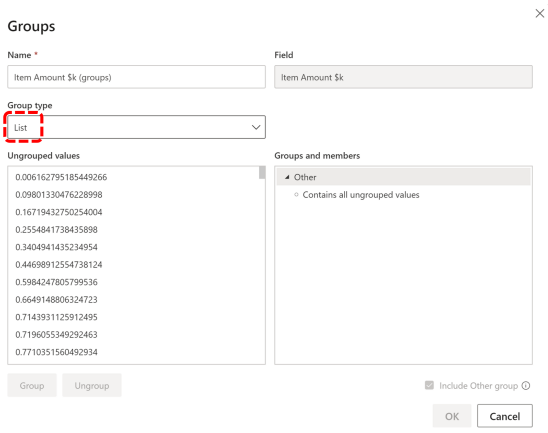


Figure 13.42: Groups dialog box (numerical: list and bins).

When we do so, we see a new column appear in the "Accounting Transactions" table, "Item Amount $k (bins)" (the name can be changed in the Group dialog, if needed).

We create a histogram with this new column:

1. In the report view, add a Stacked Column Chart;
2. drag the "Item Amount $k (bins)" column to the X-axis field;
3. drag the "Item Amount $k (bins)" column to the Y-axis field (in the dropdown menu, make sure that "count" is selected rather than some other statistic);
4. edit the bin size (as in Figure 13.43,on the right) to obtain coarser or finer histograms.

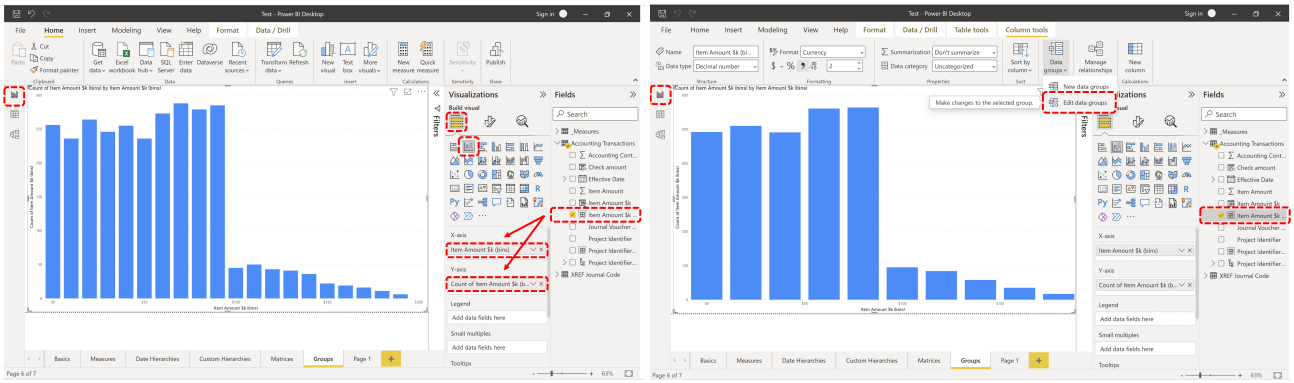


Figure 13.43: Histograms using a grouped numerical variable; bin size = 20,000 (left); 5000, (right).

Custom Sorting

We can click on a Power BI chart anywhere in the canvas, you can sort by either axes, or by the legend (ascending or descending, either numerical or alphabetical).

Some examples are shown below:

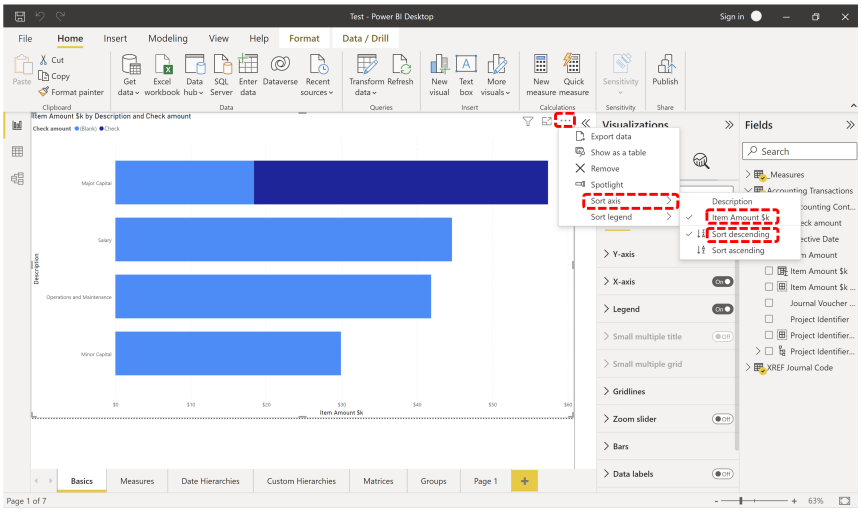


Figure 13.44: Various ways of sorting a chart.

We can also sort according to values provided in another column – in the "XREF Journal Code" table, there is a column named "Custom Sort" (see Figure 13.45).

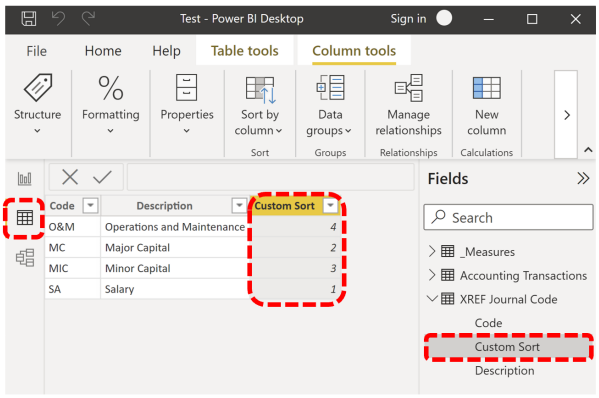


Figure 13.45: Custom sort order for journal codes (I).

We can sort the chart according to the order shown in this column as follows:

1. ensure you have in front of you a chart that is using the "Description" field (otherwise you will not see the change that is about to happen);
2. click on the "Description" column name in the Fields panel;
3. in the Column Tools menu click on "Sort By Column" and select "Custom Sort", as in Figure 13.46.

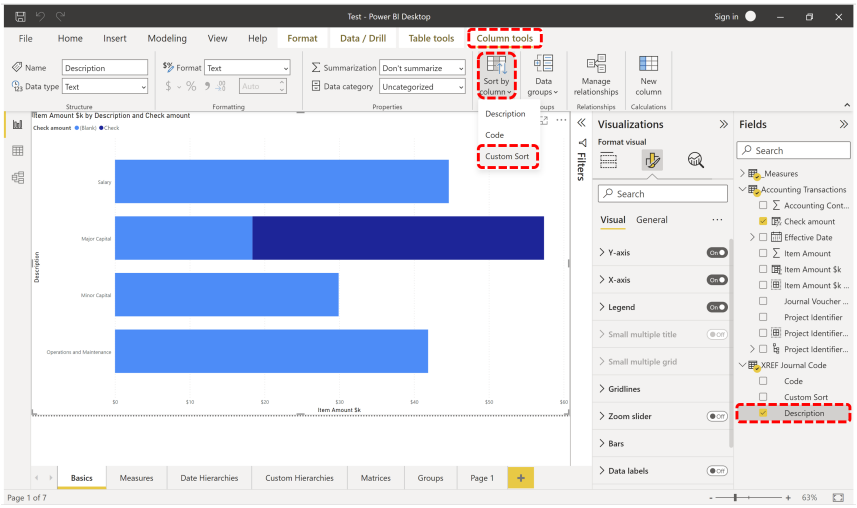
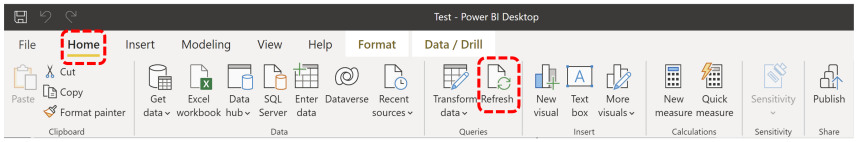


Figure 13.46: Custom sort order for journal codes (II).

We change the custom sort order by swapping the values around in the Excel spreadsheet and hitting "Refresh", the chart will update with the new sort order. This works because the "Custom Sort" field is in the same table as the "Description" field.



There is another way to sort charts by a custom sort field, of course. As an illustration, suppose that we create a chart with some columns from the

"Accounting Transactions" table and "Journal Voucher Type Code" on the axis, but keeping the custom sort order. This could come into play for a variety of reasons (e.g., such as needing to pull in a value from another table to do row-by-row calculations), so let us look at how this could be achieved using programmatic means.)

This can be achieved using the DAX function RELATED:

1. click on the "Accounting Transactions" table and click on "New Column" in the "Table tools";
2. type in the following DAX code at the prompt:

```
Custom Sort Order = RELATED('XREF Journal Code'[Custom Sort])
```

3. the "RELATED" operation looks at all connected tables;
4. we specify the column in the connected table we want to reference;
5. once selected, the code copies the data to the new column.

In essence, this produces results much as "VLOOKUP" would in Excel.

13.4 Data Wrangling

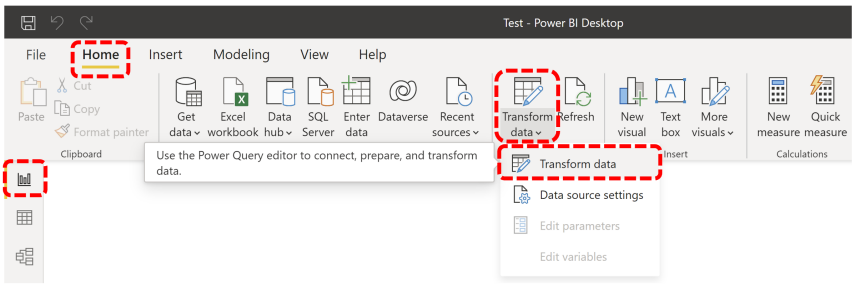
> "Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics. [...] Data wrangling typically follows a set of general steps which begin with extracting the data in a raw form from the data source, "munging" the raw data (e.g., sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a, data sink for storage and future use." [130]

In this section, we look at a few **high-level** tips to import and clean data before visualizing it, all of which are done within Power Query.[23]

23: It must be noted that some of these activities could also be completed in Power BI itself.

The main reason why we suggest doing the majority of data wrangling in Power Query is that only the post-processed data is loaded into Power BI, making the dataset more streamlined. If we wrangle the data in Power BI using DAX, then the entire dataset has to be uploaded first, which is usually not as efficient an approach.

In all of the following examples, then, we need to access the data through Power Query; this is accomplished by pressing the "Transform Data" button in the "Home" menu, as below.



Removing Rows

We can remove rows in Power Query by using the column filter.[24] As an example:

1. in the "Accounting Transactions" query, click on the "Effective Date" column;
2. to remove all transactions taking place after 31-Mar-2018, click on the small down arrow next to the column name, then on `Date Filters > Before...`, as in the Figure below.

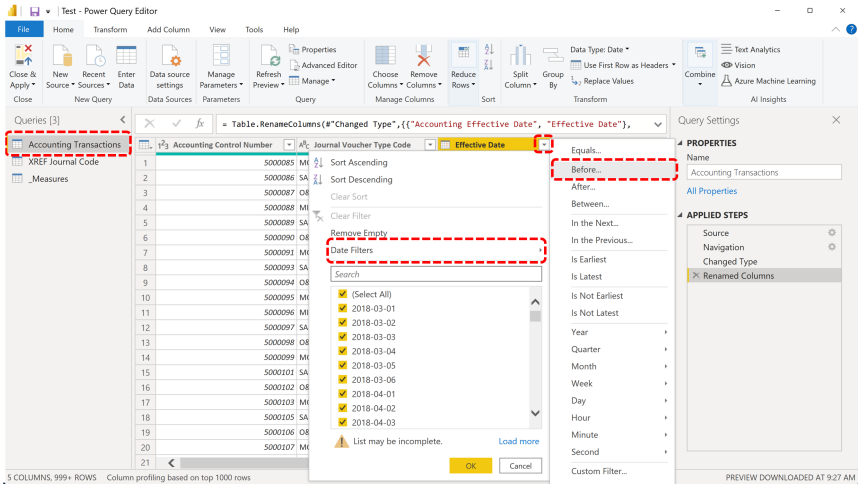


Figure 13.47: Removing rows using filters.

3. enter the appropriate filter in the dialog box and press "OK".

Those rows are now removed completely from the dataset and will NOT be loaded into Power BI.

The second way to remove rows is to use the "Remove Rows" button in the "Home" ribbon:

- "Remove Top Rows" removes the top X rows from a table: this can prove useful when a data source (such as an Excel spreadsheet) contains fronting non-data information, say;
- "Remove Bottom Rows" is identical to the above, except that it removes rows found at the bottom of a data source;
- "Remove Alternate Rows" removes alternate rows in a pattern: this can proved useful when a data source contains rows and subtotals, say;

24: This is quite similar to the "Filter" function in Excel.

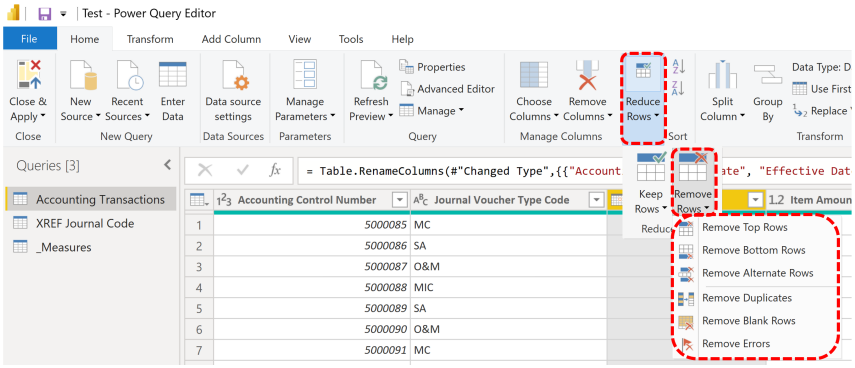


Figure 13.48: Removing rows using the menu.

- "Remove Duplicates" removes all duplicate rows;
- "Remove Blank Rows" removes all blank rows;
- "Remove Errors" removes rows with errors.[25]

25: This should only be use sparingly... fixing errors is preferable to their removal.

Replacing Values

Another useful data wrangling trick is to replace values. As an example, if a transaction's date is unknown for some reason, we may find a "N/A" in an observation's date column.

The presence of such an "N/A" makes it impossible to format that column as a date, which could "break" measures and charts that use this column.

Faced with this situation, we might decide to replace the "N/A" with a null (or empty) value:[26]

1. in Power Query, select the column in which you want to replace values;
2. click on the "Replace Values" button in the "Home" ribbon;
3. fill the "Value To Find" and "Replace With" fields;
4. the options to match the entire cell contents and/or to replace using special characters is also available in the "Advanced options" dialog (not available in every Power BI version).

26: Technically, we should really try to figure out what the actual transaction date was, but this is not always possible.

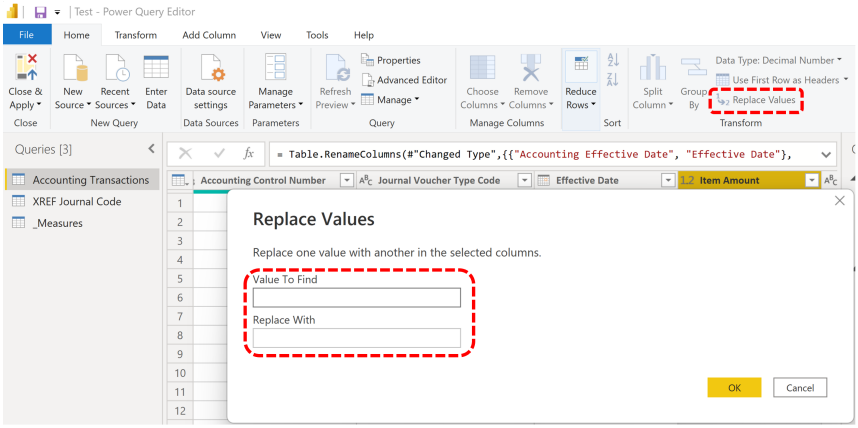


Figure 13.49: Replacing values dialog box.

Splitting Columns

At times, a text data column may contain information which could be more useful if it was split and found in one (or more) separate column.

For example, if we have a column named "Group - Level" where the entries all consist of a group name and a level, it could be advantageous, from a visual hierarchy point of view, to split the entries into two separate columns: "Group" and "Level" (we suggest that you keep the original column in the dataset, however).

In this example, we remove the project number from the project identifier:

1. select the "Project Identifier" column from "Accounting Transactions";
2. right-click on the column heading and select "Duplicate Column";
3. select the duplicate and click on the "Split Column" button and select "By Number of Characters";

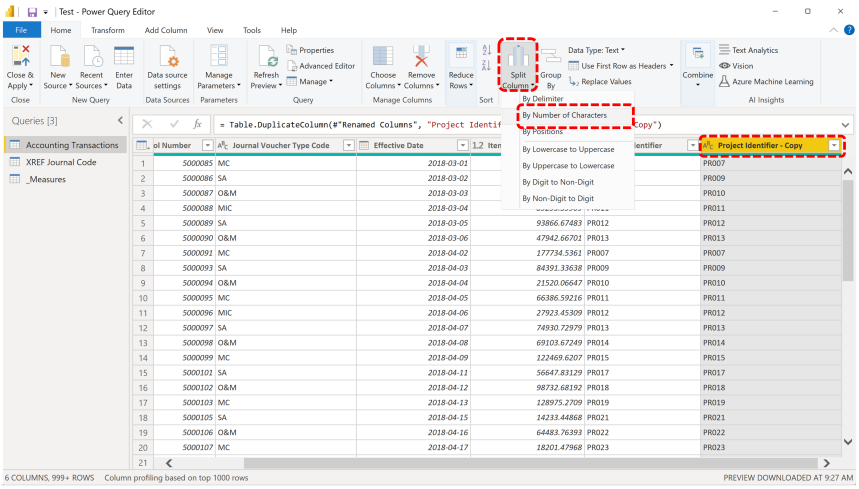
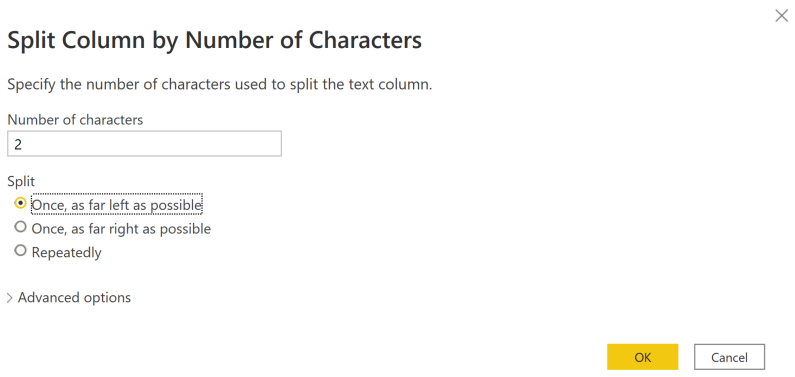


Figure 13.50: Splitting a column.

4. Type in "2" in the "Number of characters" field (given the structure of the project identifiers) and then select "Once as far left as possible":



5. Press "OK";
6. Rename the columns "Project Code" and "Project Number".

There are a few different options when it comes to splitting columns:

- "By Delimiter" – split at every instance of a comma, say;
- "By Number of Characters" – as in example on the preceding page;
- "By Positions" – at fixed points in the string;
- "By Lowercase to Uppercase" – e.g., `abcDEF` would split into `abc` and then `DEF`;
- "By Uppercase to Lowercase" – as above but in reverse;
- "By Digit to Non-Digit" – e.g., `123ABC` would split into `123` and `ABC`;
- "By Non-Digit to Digit" – as above but in reverse.

Trimming and Cleaning

Often, we need to import data that comes with leading and/or trailing white spaces on some records. We can trim such records by selecting any text column, pressing the "Format" button in the "Transform" menu, and finally selecting "Trim".

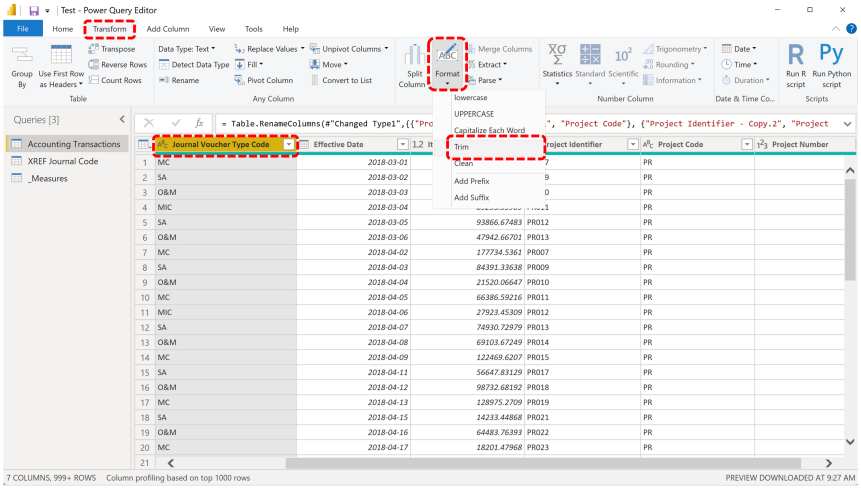


Figure 13.51: Trimming a text column.

NOTE: the "Trim" function sometimes fails to achieve the desired result as there are two types of blank spaces in text files:

- a **regular space** (ASCII code 32), and
- a **non-breaking space** (ASCII code 160).

The latter will not be removed using this procedure. Instead, we must use a special character search and replace the tricky strings.

At times, data contains unprintable (or non-printable) characters: carriage returns, tabs, line breaks etc.[27] To remove these, simply perform the steps as per the previous slide but select "Clean" from the options.

If the data contains a lot of text (e.g., comments or descriptions), it is a best practice to perform a "Clean" step in the Power Query before importing the data into Power BI.

Other useful text formatting options include:

27: As an aside, the first 32 ASCII codes (0 – 31) are reserved as control codes for printers and other peripherals.

- "lowercase" – changes all characters in a text column to lowercase;
- "UPPERCASE" – changes all characters in a text column to UPPERCASE;
- "Capitalize Each Word" – performs as described;
- "Trim" – see preceding example;
- "Clean" – see preceding example;
- "Add Prefix" – adds text to the front of each string in a text column;
- "Add Suffix" – adds text to the end of each string in a text column.

Appending Tables

In the rest of this section, we are going to work through an exercise that includes appending three tables together. The exercise will also use several items we have reviewed up to now.

We want to import new tables of data named "Project Identifier". We will create a cross-reference table using "Project Identifier" to link all the tables together in the data model.

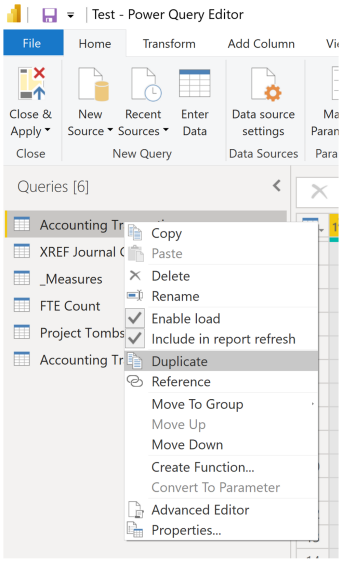
This could be done manually,[28] but this can become annoying when new projects are added to either table. Instead, we will dynamically create an automatic cross-reference table from all of the "Project Identifier" columns:

28: And in fact, most users might chose this approach.

1. import BOTH tables from the `DataSetProject.xlsx` file (`Project_Tombstone` and `FTE_Count`);
2. change the name of both the tables to remove the underscore;
3. review the formatting of all columns in both tables;

Next, we create a single XREF table containing a set of unique "Project Identifiers" that can be used to link all of the tables together:

4. right-click the "Accounting Transactions" query and select the "Duplicate" option;



5. select the duplicate table and do the following:

i. rename it “XREF Project Identifier”;
ii. select the “Project Identifier” column, right click and select “Remove Other Columns”;

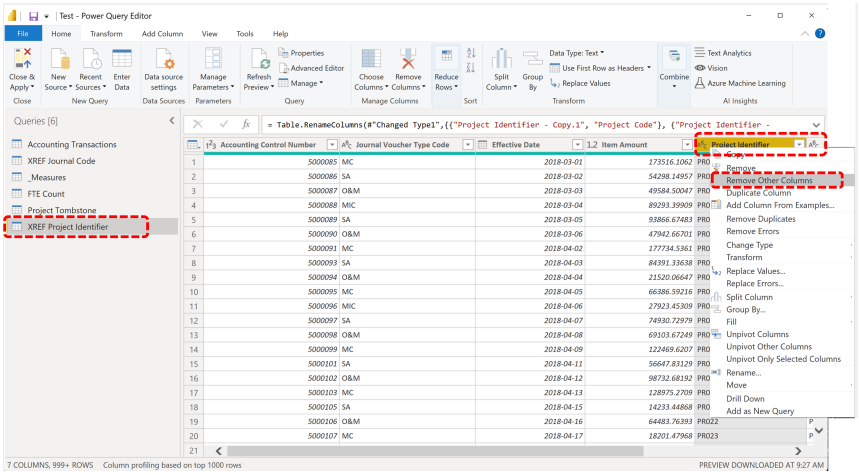


Figure 13.52

iii. do the same for “Project Tombstone” query but rename it “DNL Project Tombstone”;
iv. do the same for “FTE Count” query but rename it “DNL FTE Count” (DNL stands for DO NOT LOAD – we will come back to that later).

We now have 3 tables each containing a single column which is “Project Identifier”. **NOTE:** the column names in each table need to be **identical**!

Next, we are going to “Append” these tables, in other words put them all together into a single table. We do so by first clicking on the “target” table (the one which will eventually be uploaded into Power BI). We could create a new table, but for illustration purposes, we are simply going to use the “XREF Project Identifier” table.

Click on “XREF Project Identifier”, then in the “Home” menu click on “Append Queries”. You will see 2 options, select “Append Queries” (not as new).

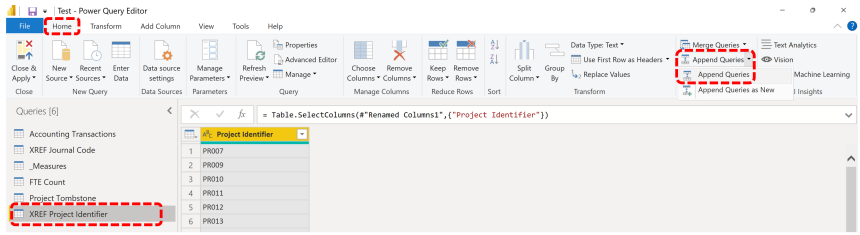


Figure 13.53: Location of the Append Queries button.

The Append dialog box asks whether we want to append only 2 or 3+ tables. Select the “Three or more” option; click on “DNL FTE Count” and then “ADD”, then repeat for “DNL Project Tombstone”.

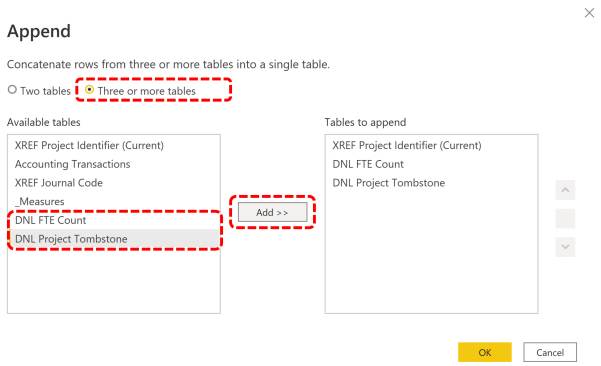


Figure 13.54: The Append dialog box.

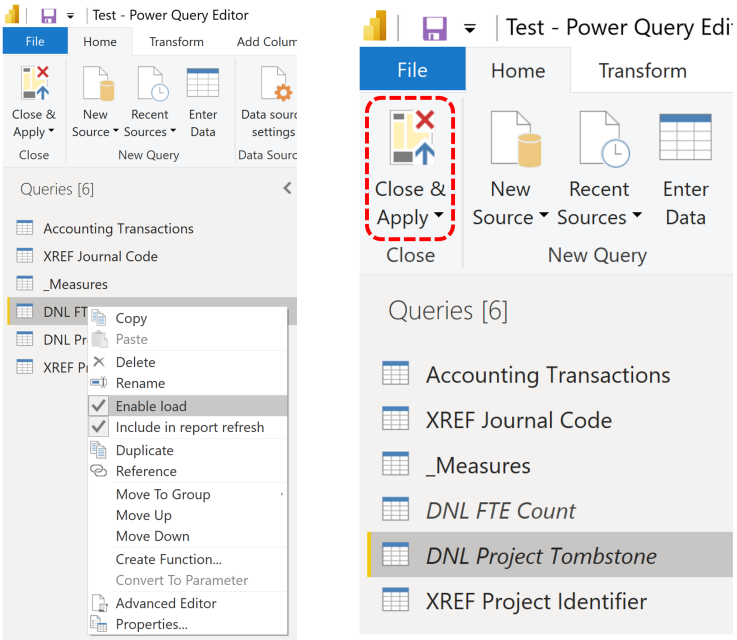
The resulting "XREF Project Identifier" query might not look different, but the instructions now exist for Power Query to append all the tables together. We can now apply the following data transformations:

1. trim the "Project Identifier" column.
2. clean to the "Project Identifier" column.
3. select "Remove Rows > Remove Duplicates".
4. select "Remove Rows > Remove Blank Rows".

The column now contains a unique list of Project Identifiers based on all three tables. We are not quite finished, however.

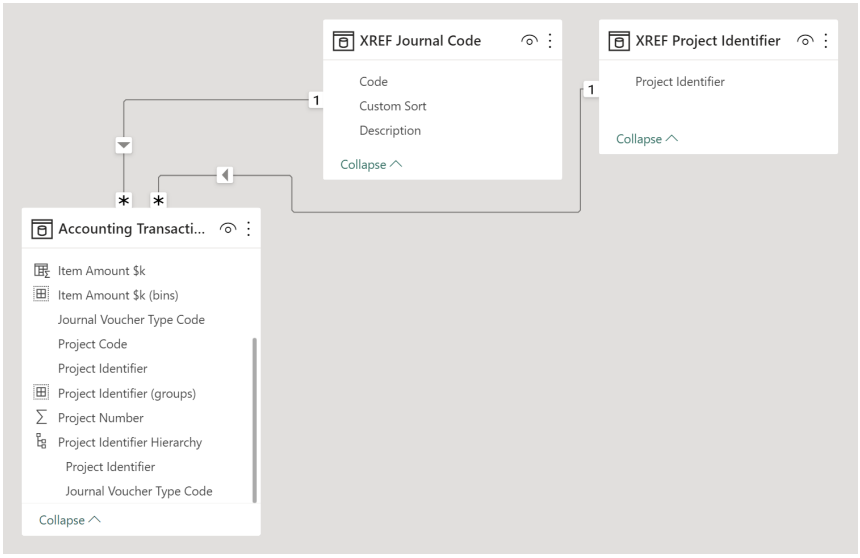
Remember those tables we renamed with the DNL prefix? If we hit "Close and Apply" now, they will be loaded into the dataset and create confusion. To ensure that they do not load, right-click the "DNL FTE Count" query, say, and uncheck "Enable Load".[29] Repeat these steps for the "DNL Project Tombstone" query. Finally hit "Close and Apply", then "Save" in the Power BI canvas.

29: This step is known to be successful when the query name changes to *italics*.

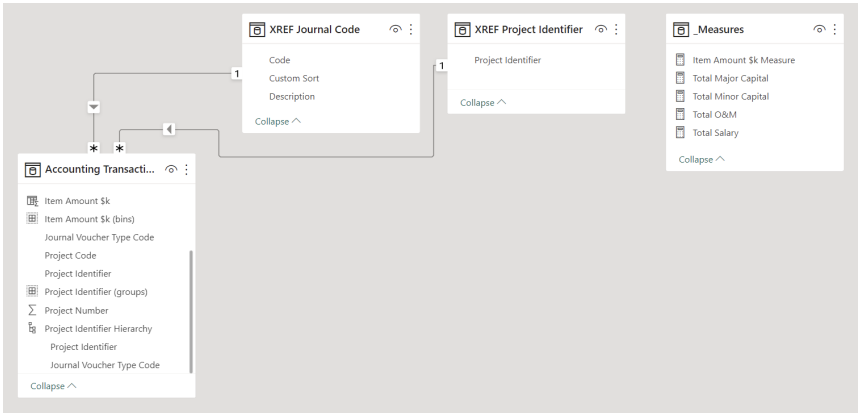


When we hit "Close and Apply", Power Query tries to link the tables together: this should show up in the modeling tab.

The initial view is very messy so move the tables closer together.



As we can see Power BI did not get all the connections correctly, so we **delete the wrong table connections** and manually create the correct ones.



Finally, note that it is good practice to place the XREF tables across the top, and the data tables down to the left: this tends to make the data model **easier to read**. We can now create charts, measures, columns, slicers, etc. using "Project Identifier" from the new XREF table.

A lot more could be said about Power BI – practice (and failure!) will help analysts get the most out of the data. Do not hesitate to consult online forums to get more advanced help, and remember that no matter how sophisticated

a tool is, it is most useful when its use is planned according to the various principles we have discussed in this book.

In our experience, any combination of competencies across the following 20 additional advanced Power BI topics would prove beneficial to analysts and users in the coming years:

1. **Exploring Power BI Architecture** – understanding the components of Power BI including Power BI Desktop, Power BI Service, and Power BI Mobile;
2. **Advanced Data Modeling** – learning how to create complex data models and relationships;
3. **Deep Dive into DAX** – advanced usage of DAX (Data Analysis Expressions) for creating complex calculations;
4. **Row Level Security (RLS)** – understanding and implementing row-level security in Power BI reports;
5. **Advanced Visualizations** – creating complex and interactive visualizations, including custom visuals;
6. **Using AI in Power BI** – understanding the AI capabilities within Power BI, such as Key Influencers and Q&A;
7. **Integration with Azure Machine Learning, `R`, and `Python`** – how to integrate Power BI with sophisticated analytics tools;
8. **Power BI and Big Data** – working with large datasets and understanding Power BI's data compression abilities;
9. **Data Gateway** – understanding the on-premises data gateway for refreshing data from on-premise data sources;
10. **Power BI Embedded** – understanding how to embed Power BI reports and dashboards into other applications;
11. **Custom Visuals Development** – building and incorporating custom visuals using JavaScript or Typescript;
12. **Power Query for Data Transformation** – advanced techniques for data cleaning and transformation using Power Query;
13. **Advanced Report Design** – best practices for designing effective and attractive reports.
14. **Performance Optimization** – techniques for optimizing the performance of Power BI reports and dashboards;
15. **Power BI APIs** – using Power BI APIs for embedding, administration, and user resources;
16. **Integration with Power Platform** – understanding the interaction between Power BI and other Power Platform components like Power Apps and Power Automate;
17. **Understanding Power BI Premium and its Benefits** – an in-depth look at the premium features of Power BI, including large data volume handling, increased refresh rates, and more;
18. **Advanced Drill-Through Features** – how to create and use advanced drill-through features in reports;
19. **Bookmarks and Storytelling** – creating interactive reports using bookmarks and storytelling features;
20. **Preparing for Power BI Certification** – guidance and resources for preparing for the *Microsoft Certified Data Analyst Associate* exam.