4

Power BI Cheat Sheets to Expand Your BI Skills

datacamp Microsoft

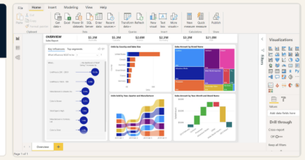
Power BI for Business Intelligence

Power BI Cheat Sheet

Learn Power BI online at www.DataCamp.com

What is Power BI?

Power BI is a business intelligence tool that allows you to effectively report insights through easy-to-use customizable visualizations and dashboards.



> Why use Power BI?

- Easy to use—no coding involved
- Integrates seamlessly with any data source
- Fast and can handle large datasets

> Power BI Components

There are three components to Power BI—each of them serving different purposes

POWER BI DESKTOP
Free desktop application that provides data analysis and creation tools.

POWER BI SERVICE
Cloud-based version of Power BI with report editing and publishing features.

POWER BI MOBILE
A mobile app of Power BI, which allows you to author, view, and share reports on the go.

> Getting started with Power BI

There are three main views in Power BI

REPORT VIEW
This view is the default view, where you can visualize data and create reports

DATA VIEW
This view lets you examine datasets associated with your reports

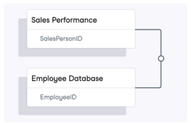
MODEL VIEW
This view helps you establish different relationships between datasets

> Visualizing your first dataset

Upload datasets into Power BI

- Underneath the *Home* tab, click on *Get Data*
- Choose any of your datasets and double click
- Click on *Load* if not prior data needs processing
- If you need to transform the data, click *Transform* which will launch Power Query. Keep reading this cheat sheet for how to apply transformations in Power Query.
- Inspect your data by clicking on the *Data View*

Create relationships in Power BI



- If you have different datasets you want to connect. First, upload them into Power BI
- Click on the *Model View* from the left-hand pane
- Connect key columns from different datasets by dragging one to another (e.g., EmployeeID to e.g., SalespersonID)

Create your first visualization

- Click on the *Report View* and go to the *Visualizations* pane on the right-hand side
- Select the type of visualization you would like to plot your data on. Keep reading this cheat to learn different visualizations available in Power BI.
- Under the *Field* pane on the right-hand side, drag the variables of your choice into *Values* or *Axis*.

Values let you visualize aggregate measures (e.g. Total Revenue)
Axis let you visualize categories (e.g. Sales Person)

Aggregating data

Power BI sums numerical fields when visualizing them under *Values*. However, you can choose different aggregations

- Select the visualization you just created
- Go to the *Visualizations* section on the right-hand side
- Go to *Values*—the visualized column should be there
- On the selected column—click on the dropdown arrow and change the aggregation (i.e., `AVERAGE`, `MAX`, `COUNT`, etc..)

> Data Visualizations in Power BI

Power BI provides a wide range of data visualizations. Here is a list of the most useful visualizations you have in Power BI

- Bar Charts: Horizontal bars used for comparing specific values across categories (e.g. sales by region)
- Column Charts: Vertical columns for comparing specific values across categories
- Line Charts: Used for looking at a numeric value over time (e.g. revenue over time)
- Area Chart: Based on the line chart with the difference that the area between the axis and line is filled in (e.g. sales by month)
- Scatter: Displays one set of numerical data along the horizontal axis and another set along the vertical axis (e.g. relation age and loan)
- Combo Chart: Combines a column chart and a line chart (e.g. actual sales performance vs target)
- Treemaps: Used to visualize categories with colored rectangles, sized with respect to their value (e.g. product category based on sales)
- Pie Chart: Circle divided into slices representing a category's proportion of the whole (e.g. market share)
- Donut Chart: Similar to pie charts; used to show the proportion of sectors to a whole (e.g. market share)
- Maps: Used to map categorical and quantitative information to spatial locations (e.g. sales per state)
- Cards: Used for displaying a single fact or single data point (e.g. total sales)
- Table: Grid used to display data in a logical series of rows and columns (e.g. all products with sold items)

> Power Query Editor in Power BI

Power Query is Microsoft's data transformation and data preparation engine. It is part of Power BI Desktop, and lets you connect to one or many data sources, shape and transform data to meet your needs, and load it into Power BI.

Open the Power Query Editor

While loading data
- Underneath the *Home* tab, click on *Get Data*
- Choose any of your datasets and double click
- Click on *Transform* Data

When data is already loaded
- Go to the *Data View*
- Under *Queries* in the *Home* tab of the ribbon, click on *Transform Data* drop-down, then on the *Transform Data* button

Using the Power Query Editor

Removing rows
You can remove rows dependent on their location, and properties
- Click on the *Home* tab in the *Query* ribbon
- Click on *Remove Rows* in the *Reduce Rows* group
- Choose which option to remove, whether *Remove Top Rows*, *Remove Bottom Rows*, etc..
- Choose the number of rows to remove
- You can undo your action by removing it from the *Applied Steps* list on the right-hand side

Adding a new column
You can create new columns based on existing or new data
- Click on the *Add Column* tab in the *Query* ribbon
- Click on *Custom Column* in the *General* group
- Name your new column by using the *New Column Name* option
- Define the new column formula under the custom column formula using the available data

Replace values
You can replace one value with another value wherever that value is found in a column
- In the Power Query Editor, select the cell or column you want to replace
- Click on the column or value, and click on *Replace Values* under the *Home* tab under the *Transform* group
- Fill the *Value to Find* and *Replace With fields* to complete your operation

Appending datasets
You can append one dataset to another

- Click on *Append Queries* under the *Home* tab under the *Combine* group
- Select to append either *Two tables* or *Three or more tables*
- Add tables to append under the provided section in the same window

Merge Queries
You can use merge tables based on a related column

- Click on *Merge Queries* under the *Home tab* under the *Combine group*
- Select the first table and the second table you would like to merge
- Select the columns you would like to join the tables on by clicking on the column from the first dataset, and from the second dataset
- Select the *Join Kind* that suits your operation:

- Click on *Ok*—new columns will be added to your current table

Data profiling
Data Profiling is a feature in Power Query that provides intuitive information about your data

- Click on the *View* tab in the *Query* ribbon
- In the *Data Preview* tab—tick the options you want to visualize
- Tick *Column Quality* to see the amount of missing data
- Tick *Column Distribution* to see the statistical distribution under every column
- Tick *Column Profile* to see summary statistics and more detailed frequency information of columns

> DAX Expressions

Data Analysis Expressions (DAX) is a calculation language used in Power BI that lets you create calculations and perform data analysis. It is used to create calculated columns, measures, and custom tables. DAX functions are predefined formulas that perform calculations on specific values called arguments.

Sample data
Throughout this section, we'll use the columns listed in this sample table of `sales_data`

deal_size	sales_person	date	customer_name
1,000	Maria Shuttleworth	30-03-2022	Acme Inc.
3,000	Nuno Rocha	29-03-2022	Spotflix
2,300	Terence Mickey	13-04-2022	DataChamp

Simple aggregations
- `SUM(<column>)` adds all the numbers in a column
- `AVERAGE(<column>)` returns the average (arithmetic mean) of all numbers in a column
- `MEDIAN(<column>)` returns the median of numbers in a column
- `MIN/MAX(<column>)` returns the smallest/biggest value in a column
- `COUNT(<column>)` counts the number of cells in a column that contain non-blank values
- `DISTINCTCOUNT(<column>)` counts the number of distinct values in a column.

EXAMPLES
- Sum of all deals — `SUM('sales_data'[deal_size])`
- Average deal size — `AVERAGE('sales_data'[deal_size])`
- Distinct number of customers — `DISTINCTCOUNT('sales_data'[customer_name])`

Logical functions
- `IF(<logical_test>, <value_if_true>[, <value_if_false>])` check the result of an expression and create conditional results

EXAMPLES
- Create a column called `large_deal` that returns "Yes" if `deal_size` is bigger than 2,000 and "No" otherwise
  `large_deal = IF( 'sales_data'[deal_size] > 2000, "Yes", "No")`

Text Functions
- `LEFT(<text>, <num_chars>)` returns the specified number of characters from the start of a text
- `LOWER(<text>)` converts a text string to all lowercase letters
- `UPPER (<text>)` converts a text string to all uppercase letters
- `REPLACE(<old_text>, <start_num>, <num_chars>, <new_text>)` replaces part of a text string with a different text string.

EXAMPLES
- Change column `customer_name` be only lower case
  `customer_name = LOWER('sales_data'[customer_name])`

Date and time functions
- `CALENDAR(<start date>, <end date>)` generates a column of continuous sets of dates
- `DATE(<year>, <month>, <day>)` returns the specified date in the datetime format
- `WEEKDAY(<date>, <return_type>)` returns 1-7 corresponding to the day of the week of a date (`return_type` indicates week start and end (1: Sunday-Saturday, 2: Monday-Sunday)

EXAMPLES
- Return the day of week of each deal
  `week_day = WEEKDAY('sales_data'[date], 2)`

datacamp Microsoft

Power BI for Business Intelligence

DAX Cheat Sheet

> Math & statistical functions

- **SUM(<column>)** Adds all the numbers in a column.
- **SUMX(<table>, <expression>)** Returns the sum of an expression evaluated for each row in a table.
- **AVERAGE(<column>)** Returns the average (arithmetic mean) of all the numbers in a column.
- **AVERAGEX(<table>, <expression>)** Calculates the average (arithmetic mean) of a set of expressions evaluated over a table.
- **MEDIAN(<column>)** Returns the median of a column.
- **MEDIANX(<table>, <expression>)** Calculates the median of a set of expressions evaluated over a table.
- **GEOMEAN(<column>)** Calculates the geometric mean of a column.
- **GEOMEANX(<table>, <expression>)** Calculates the geometric mean of a set of expressions evaluated over a table.
- **COUNT(<column>)** Returns the number of cells in a column that contain non-blank values.
- **COUNTX(<table>, <expression>)** Counts the number of rows from an expression that evaluates to a non-blank value.
- **DIVIDE(<numerator>, <denominator> [,<alternateresult>])** Performs division and returns alternate result or BLANK() on division by 0.
- **MIN(<column>)** Returns a minimum value of a column.
- **MAX(<column>)** Returns a maximum value of a column.
- **COUNTROWS([<table>])** Counts the number of rows in a table.
- **DISTINCTCOUNT(<column>)** Counts the number of distinct values in a column.
- **RANKX(<table>, <expression>[, <value>[, <order>[, <ties>]]])** Returns the ranking of a number in a list of numbers for each row in the table argument.

> Filter functions

- **FILTER(<table>, <filter>)** Returns a table that is a subset of another table or expression.
- **CALCULATE(<expression>[, <filter1> [, <filter2> [, …]]])** Evaluates an expression in a filter context.
- **HASONEVALUE(<columnName>)** Returns TRUE when the context for columnName has been filtered down to one distinct value only. Otherwise it is FALSE.
- **ALLNOBLANKROW(<table> | <column>[, <column>[, <column>[,…]]])** Returns a table that is a subset of another table or expression.
- **ALL([<table> | <column>[, <column>[, <column>[,…]]]])** Returns all the rows in a table, or all the values in a column, ignoring any filters that might have been applied.
- **ALLEXCEPT(<table>, <column>[, <column>[,..]])** Returns all the rows in a table except for those rows that are affected by the specified column filters.
- **REMOVEFILTERS([<table> | <column>][, <column>[, <column>[,…]]])** Clear all filters from designated tables or columns.

> Logical functions

- **IF(<logical_test>, <value_if_true>[, <value_if_false>])** Checks a condition, and returns a certain value depending on whether it is true or false.
- **AND(<logical 1>, <logical 2>)** Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE. Otherwise, it returns FALSE.
- **OR(<logical 1>, <logical 2>)** Checks whether one of the arguments is TRUE to return TRUE. The function returns FALSE if both arguments are FALSE.
- **NOT(<logical>)** Changes TRUE to FALSE and vice versa.
- **SWITCH(<expression>, <value>, <result>[, <value>, <result>]…[, <else>])** Evaluates an expression against a list of values and returns one of possible results
- **IFERROR(<value>, <value_if_error>)** Returns value_if_error if the first expression is an error and the value of the expression itself otherwise.

> Date & time functions

- **CALENDAR(<start_date>, <end_date>)** Returns a table with a single column named "Date" that contains a contiguous set of dates.
- **DATE(<year>, <month>, <day>)** Returns the specified date in datetime format.
- **DATEDIFF(<date_1>, <date_2>, <interval>)** Returns the number of units between two dates as defined in **<interval>**.
- **DATEVALUE(<date_text>)** Converts a date in text to a date in datetime format.
- **DAY(<date>)** Returns a number from 1 to 31 representing the day of the month.
- **WEEKNUM(<date>)** Returns weeknumber in the year.
- **MONTH(<date>)** Returns a number from 1 to 12 representing a month.
- **QUARTER(<date>)** Returns a number from 1 to 4 representing a quarter.

> Time intelligence functions

- **DATEADD(<dates>, <number_of_intervals>, <interval>)** Moves a date by a specific interval.
- **DATESBETWEEN(<dates>, <date_1>, <date_2>)** Returns the dates between specified dates.
- **TOTALYTD(<expression>, <dates>[, <filter>][, <year_end_date>])** Evaluates the year-to-date value of the expression in the current context.
- **SAMEPERIODLASTYEAR(<dates>)** Returns a table that contains a column of dates shifted one year back in time.
- **STARTOFMONTH(<dates>) // ENDOFMONTH(<dates>)** Returns the start // end of the month.
- **STARTOFQUARTER(<dates>) // ENDOFQUARTER(<dates>)** Returns the start // end of the quarter.
- **STARTOFYEAR(<dates>) // ENDOFYEAR(<dates>)** Returns the start // end of the quarter.

> Relationship functions

- **CROSSFILTER(<left_column>, <right_column>, <crossfiltertype>)** Specifies the cross-filtering direction to be used in a calculation.
- **RELATED(<column>)** Returns a related value from another table.

> Table manipulation functions

- **SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]…[, <name>, <expression>]…)** Returns a summary table for the requested totals over a set of groups.
- **DISTINCT(<table>)** Returns a table by removing duplicate rows from another table or expression.
- **ADDCOLUMNS(<table>, <name>, <expression>[, <name>, <expression>]…)** Adds calculated columns to the given table or table expression.
- **SELECTCOLUMNS(<table>, <name>, <expression>[, <name>, <expression>]…)** Selects calculated columns from the given table or table expression.
- **GROUPBY(<table> [, <groupBy_columnName>[, [<column_name>] [<expression>]]…)** Create a summary of the input table grouped by specific columns.
- **INTERSECT(<left_table>, <right_table>)** Returns the rows of the left-side table that appear in the right-side table.
- **NATURALINNERJOIN(<left_table>, <right_table>)** Joins two tables using an inner join.
- **NATURALLEFTOUTERJOIN(<left_table>, <right_table>)** Joins two tables using a left outer join.
- **UNION(<table>, <table>[, <table> [,…]])** Returns the union of tables with matching columns.

> Text functions

- **EXACT(<text_1>, <text_2>)** Checks if two strings are identical (**EXACT()** is case sensitive).
- **FIND(<text_tofind>, <in_text>)** Returns the starting position a text within another text (**FIND()** is case sensitive).
- **FORMAT(<value>, <format>)** Converts a value to a text in the specified number format.
- **LEFT(<text>, <num_chars>)** Returns the number of characters from the start of a string.
- **RIGHT(<text>, <num_chars>)** Returns the number of characters from the end of a string.
- **LEN(<text>)** Returns the number of characters in a string of text.
- **LOWER(<text>)** Converts all letters in a string to lowercase.
- **UPPER(<text>)** Converts all letters in a string to uppercase.
- **TRIM(<text>)** Remove all spaces from a text string.
- **CONCATENATE(<text_1>, <text_2>)** Joins two strings together into one string.
- **SUBSTITUTE(<text>, <old_text>, <new_text>, <instance_num>)** Replaces existing text with new text in a string.
- **REPLACE(<old_text>, <start_posotion>, <num_chars>, <new_text>)** Replaces part of a string with a new string.

> Information functions

- **COLUMNSTATISTICS()** Returns statistics regarding every column in every table. This function has no arguments.
- **NAMEOF(<value>)** Returns the column or measure name of a value.
- **ISBLANK(<value>) // ISERROR(<value>)** Returns whether the value is blank // an error.
- **ISLOGICAL(<value>)** Checks whether a value is logical or not.
- **ISNUMBER(<value>)** Checks whether a value is a number or not.
- **ISFILTERED(<table> | <column>)** Returns true when there are direct filters on a column.
- **ISCROSSFILTERED(<table> | <column>)** Returns true when there are crossfilters on a column.
- **USERPRINCIPALNAME()** Returns the user principal name or email address. This function has no arguments.

> DAX statements

- **VAR(<name> = <expression>)** Stores the result of an expression as a named variable. To return the variable, use RETURN after the variable is defined.
- **COLUMN(<table>[<column>] = <expression>)** Stores the result of an expression as a column in a table.
- **ORDER BY(<table>[<column>])** Defines the sort order of a column. Every column can be sorted in ascending (**ASC**) or descending (**DESC**) way.

> DAX Operators

Comparison operators	Meaning
=	Equal to
= =	Strict equal to
>	Greater than
<	Smaller than
> =	Greater than or equal to
= <	Smaller than or equal to
< >	Not equal to

Text operator	Meaning	Example
&	Concatenates text values	**Concatenates text values** | [City]&", "&[State]

Logical operator	Meaning	Example
&&	**AND** condition	([City] = "Bru") && ([Return] = "Yes"))
||	**OR** condition	([City] = "Bru") || ([Return] = "Yes"))
IN {}	**OR** condition for each row	Product[Color] IN {"Red", "Blue", "Gold"}

datacamp

Working With Tables in Power Query M in Power BI

Learn Power BI online at www.DataCamp.com

> Records

Creation

```
// Define a record with [name = value]
//You can include different data types including lists
[
  Name = "William Playfair",
  BirthYear = 1759,
  IsDataScientist = true,
  ChartsInvented = {"line", "bar", "area", "pie"}
]

// Create a record from lists of values and fields with FromList()
Record.FromList(
  {"William Playfair", 1759, true, {"line", "bar", "area", "pie"}},
  {"Name", "BirthYear", "IsDataScientist", "ChartsInvented"}
)
// Create a record from a table of records with FromTable()
Record.FromTable(
  Table.FromRecords({
    [Name = "Name", Value = "William Playfair"],
    [Name = "BirthYear", Value = 1759],
    [Name = "IsDataScientist", Value = true],
    [Name = "ChartsInvented", Value = {"line", "bar", "area", "pie"}]
  })
)

// Records can be nested
[
  Name = [First = "William", Last = Playfair"],
  BirthYear = 1759,
  IsDataScientist = true,
  ChartsInvented = {"line", "bar", "area", "pie"}
]
```

Example records

```
// Define a record
let
  TaylorSwift = [
    FirstName = "Taylor",
    LastName = "Swift",
    BirthDate = #date(1989, 12, 13)
  ]
in
  TaylorSwift
```

Counting

```
// Get the number of fields with FieldCount()
Record.FieldCount(TaylorSwift) // Returns 3

// Determine if a record has a field name with HasFields()
Record.HasFields(TaylorSwift, "LastName") // Returns true
```

Manipulation/Transformation

```
// Add a new field with AddField()
Record.AddField(TaylorSwift, "MiddleName", "Alison")

// Combine fields from records with Combine()
Record.Combine(TaylorSwift)

// Remove fields with RemoveFields()
Record.RemoveFields(TaylorSwift)

// Change the order of fields with ReorderFields()
Record.ReorderFields(TaylorSwift)

// Change values in a field with TransformFields()
Record.TransformFields(TaylorSwift, {"BirthDate", Date.ToText})
```

Metadata

```
// Add a metadata record to a value with meta
"To a collector of curios, the dust is metadata." meta [
  ContentType = "quote",
  Author = "David Weinberger",
  Source = "Everything Is Miscellaneous: The Power of the New Digital Disorder"
]

// Remove all metadata with RemoveMetadata()
Value.RemoveMetadata(curios)

// Remove specific metadata with RemoveMetadata(, metaValue)
Value.RemoveMetadata(curios, "Author")
```

> Tables

Creation

```
// Create a table with #table()
#table(
  {"Name", "BirthYear", "IsDataScientist", "ChartsInvented"},
  {
    {"William Playfair", 1759, true, {"line", "bar", "area", "pie"}},
    {"Karl Pearson", 1857, true, {"histogram"}}
  }
)

// Create a table from a list of records with FromRecords()
Table.FromRecords({
  [
    Name = "William Playfair",
    BirthYear = 1759,
    IsDataScientist = true,
    ChartsInvented = {"line", "bar", "area", "pie"}
  ],
  [
    Name = "Karl Pearson",
    BirthYear = 1857,
    IsDataScientist = true,
    ChartsInvented = {"histogram"}
  ]}
)

// Enforce column data types with type table[]
Table.FromRecords({
  [
    Name = "William Playfair",
    BirthYear = 1759,
    IsDataScientist = true,
    ChartsInvented = {"line", "bar", "area", "pie"}
  ],
  [
    Name = "Karl Pearson",
    BirthYear = 1857,
    IsDataScientist = true,
    ChartsInvented = {"histogram"}
  ]},
  type table[Name = text, BirthYear = number, IsDataScientist = logical, ChartsInvented = list]
)

// Create a table from a list of lists with FromColumns()
Table.FromColumns({
 {"William Playfair", "Karl Pearson"},
 {1759, 1857},
 {true, true},
 {{"line", "bar", "area", "pie"}, {"histogram"}}
  },
  {"Name", "BirthYear", "IsDataScientist ", "ChartsInvented"}
)

// Create a table from a list of lists with FromRows()
Table.FromRows({
 {"William Playfair", 1759, true, {"line", "bar", "area", "pie"}},
 {"Karl Pearson", 1857, true, {"histogram"}}
  },
  {"Name", "BirthYear", "IsDataScientist ", "ChartsInvented"}
)
```

Example tables

```
// Define tables
let
  Musicians = #table(
    {"ID", "FirstName", "LastName", "BirthDate"},
    {
      {1, "Taylor", "Swift", #date(1989, 12, 13)},
      {2, "Ed", "Sheeran", #date(1991, 2, 17)}
    }
  )
in
  Musicians

let
  Albums = #table(
    {"ID", "ArtistID", "Title"},
    {
      {1, 1, "1989"},
      {2, 2, "-"},
      {3, 2, "5"}
    }
)
in
  Albums
```

Counting

```
// Get the number of rows with RowCount()
Table.RowCount(Musicians) // Returns 2

// Get the number of columns with ColumnCount()
Table.ColumnCount(Musicians) // Returns 4

// Get the column names with ColumnNames()
Table.ColumnNames(Musicians) // Returns {"ID", "FirstName", "LastName", "BirthDate"}

// Get details of the columns with Schema()
Table.Schema(Musicians) // Returns a table of column details

// Get a summary of number columns with Profile()
Table.Profile(Musicians) // Returns a table of min, max, mean, etc. by column
```

Selection

```
// Get a record by position with {}
Musicians{0} // Returns Taylor Swift record

// Get a column with []
Musicians[FirstName]

// Get a column dynamically with Column()
Table.Column(Musicians, "FirstName")

// Get the first few rows with FirstN()
Table.FirstN(Musicians, 1) // Returns first record

// Get the last few element with LastN()
Table.LastN(Musicians, 1) // Returns last record

// Select unique records with Distinct()
Table.Distinct(Table.Combine(Musicians, Musicians)) // Returns Musicians

// Get elements that match a criteria with SelectRows()
Table.SelectRows(Musicians, each Text.Contains([FirstName], "Tay")) // Returns the Taylor Swift record

// Return true if all elements match a criteria with MatchesAllRows()
Table.MatchesAllRows(Musicians, each [IsDataScientist]) // Returns true

// Return true if any elements match a criteria with MatchesAnyRows()
Table.MatchesAnyRows(Musicians, each Text.Contains([FirstName], "Drake")) // Returns false
```

Row manipulation

```
// Insert records into a table with InsertRows()
Table.InsertRows(
  Musicians,
  1,
  {[FirstName = "Bad", LastName = "Bunny", BirthDate = #date(1994, 3, 10)]}
) // Returns a table with new record after previous 1st record

// Vertically concatenate tables with Combine()
Table.Combine(
  Musicians,
  Table.FromRecords({
    [FirstName = "Bad", LastName = "Bunny", BirthDate = #date(1994, 3, 10)]
  })
) // Returns a table with 3 records

// Remove records with RemoveRows()
Table.RemoveRows(Musicians, 0) // Returns table without 0th record

// Change the order of records with Sort()
Table.Sort(Musicians, {"FirstName"}) // Returns by alphabetical order of FirstName

// Change values in a field with TransformRows()
Table.TransformRows(Musicians, {"BirthDate", Date.ToText})

// Calculate grouped aggregations with Group()
Table.Group(Musicians, "FirstName", each List.Min([BirthDate])
```

Column manipulation

```
// Add a column to a table with AddColumn()
Table.AddColumn(Musicians, "FullName", each [FirstName] & [LastName]) // Returns table with extra column

// Select columns of a table with SelectColumns()
Table.SelectColumns(Musicians, {"FirstName", "LastName"}) // Returns 2 columns

// Drop columns of a table with RemoveColumns()
Table.RemoveColumns(Musicians, {"BirthDate"}) // Returns remaining 3 columns

// Change the order of columns with ReorderColumns()
Table.ReorderColumns(Musicians, {"LastName", "FirstName", "BirthDate", "ID"}) // Returns table with new column order

// Change values in a field with TransformColumns()
Table.TransformColumns(
  Musicians,
  {{"FirstName", Text.Upper}, {"LastName", Text.Lower}}
)
```

Table Relations

```
// Set as column as the primary key with AddKey(, , true)
Table.AddKey(Musicians, "ID", true)

// Set as column as the secondary key with AddKey(, , false)
Table.AddKey(Albums, "ArtistID", false)

// Join two tables with Join()
Table.Join(Musicians, "ID", Albums, "ArtistID", JoinKind.LeftOuter)
```

Pivoting

```
// Convert from wide to long with Unpivot()
Table.Unpivot(Musicians, {"FirstName", "LastName"}, "NameType", "NameValue") // Returns table with FirstName and LastName on their own rows

// Convert from long to wide with Pivot()
Table.Unpivot(MusiciansLong, {"FirstName", "LastName"}, "NameType", "NameValue") // Reverses the unpivot step
```

datacamp

Data Transformation with Power Query M in Power BI

Learn Power BI at www.DataCamp.com

> Definitions

Power Query is a tool for extract-transform-load (ETL). That is, it lets you import and prepare your data for use in Microsoft data platforms including Power BI, Excel, Azure Data Lake Storage and Dataverse.

Power Query Editor is the graphical user interface to Power Query.

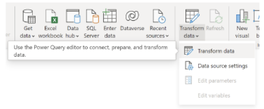
Power Query M ("M" for short) is the functional programming language used in Power Query.

DAX is the other programming language available for Power BI. DAX is used for data analysis rather than ETL. Learn more about it in the DataCamp DAX cheat sheet.

An **expression** is a single formula that returns a value.
A **query** is a sequence of expressions used to define more complex data transformations. Queries are defined using let-in code blocks.

> Accessing M in Power BI

M code can be seen in Power Query Editor. In the ribbon, click on 'Transform data' to open the Power Query Editor.



M code is shown in the Formula Bar.

M code can also be seen in the Advanced Editor window. Click 'Advanced Editor' to open the Advanced Editor window.

> Creating values

```
999 // Define a number

true // Define a logical value

"DataCamp" // Define a text value

null // Define a null (missing value)

#date(2023, 12, 31) // Define a date with #date()

#datetime(2022, 9, 8, 15, 10, 0) // Define a datetime with #datetime()
```

> Variables

```
// Variables are assigned by writing a query with let-in
let
  // Intermediate calculations
  TempF = 50
  TempC = 5 / 9 * (TempF - 32)
in
  // Resulting variable
  TempC

// By convention, variable names are UpperCamelCase
HeightM

// Quote variable names and prefix with # for non-standard names
#"Height in Meters"
```

> Operators

Arithmetic operators

```
102 + 37 // Add two numbers with +
102 - 37 // Subtract a number with -
4 * 6    // Multiply two numbers with *
22 / 7   // Divide a number by another with /
```

Numeric comparison operators

```
3 = 3  // Test for equality with =
3 <> 3 // Test for inequality with <>
3 > 1  // Test greater than with >
3 >= 3 // Test greater than or equal to with >=
3 < 4  // Test less than with <
3 <= 4 // Test less than or equal to with <=
```

Logical Operators

```
not (2 = 2)          // Logical NOT with not
(1 <> 1) and (1 < 1) // Logical AND with and
(1 >= 1) or (1 < 1)  // Logical OR with or
```

Text Operators

```
"fish" & " & " & "chips" // Combine text with &
```

> Numbers

Arithmetic

```
Number.Power(3, 4) // Raise to the power with Power()
Number.IntegerDivide(22, 7) // Integer divide a number with IntegerDivide()
Number.Mod(22, 7) // Get the remainder after division with Mod()
Value.Equals(1.999999, 2, Precision.Double) // Check number close to equal with Equals()
```

Math functions

```
Number.Ln(10) // Calculate natural logarithm with Ln()
Number.Exp(3) // Calculate exponential with Exp()
Number.Round(12.3456, 2) // Round to n decimal places with Round()
Number.Abs(-3) // Calculate absolute values with Abs()
Number.Sqrt(49) // Calculate the square root with Sqrt()
Number.IsNaN(Number.NaN) // Returns true if not a number
```

> Text Values

Creating text

```
// Text values are double-quoted, and can span multiple lines
"M is a programming
language for ETL"

// Embed quotes in strings by doubling them
"""M is magnificent"", mentioned Mike."

// Include control characters with #()
"Split text with a tab character, #(tab), or start a new line with carriage-return line feed, #(cr,lf)"

// Embed # in strings with #(#)
"Hex codes for colors start with #(#)"
```

Creating text

```
// Text values are double-quoted, and can span multiple lines
"M is a programming
language for ETL"

// Embed quotes in strings by doubling them
"""M is magnificent"", mentioned Mike."

// Include control characters with #()
"Split text with a tab character, #(tab), or start a new line with carriage-return line feed, #(cr,lf)"

// Embed # in strings with #(#)
"Hex codes for colors start with #(#)"
```

Indexing

```
// Get the number of characters in text with Length()
Text.Length("How long will dinner be? About 25cm.")

// Get a substring with Middle()
Text.Middle("Zip code: 10018", 10, 5)
```

Splitting and combining text

```
// Combine text, optionally separated with Combine()
Text.Combine({"fish", "chips"}, " & ")

// Split text on a delimiter with Split()
Text.Split("fish & chips", " & ")
```

Mutating text

```
// Convert text to upper case with Upper()
Text.Upper("iN cAsE oF eMeRgEnCy") // Returns "IN CASE OF EMERGENCY"

// Convert text to LOWer case with Lower()
Text.Lower("iN cAsE oF eMeRgEnCy") // Returns "in case of emergency"

// Convert text to title case with Proper()
Text.Proper("iN cAsE oF eMeRgEnCy") // Returns "In Case Of Emergency"

// Replace characters in text with Replace()
Text.Replace("Have a nice trip", " n", "n ") // Returns "Have an ice trip"
```

Type Conversion

```
// Convert value to number with Number.From()
Number.From(true) // Returns 1

// Dates and datetimes given as time in days since 1899-12-30
Number.From(#datetime(1969, 7, 21, 2, 56, 0)) // Returns 25405.12

// Convert text to number with Number.FromText()
Number.FromText("4.5E3") // Returns 4500

// Convert number to text with Number.ToText() — Formats: "D": integer digits, "E": exponential, "F": fixed, G": general, "N": number, "P": percent
Number.ToText(4500, "E") // Returns "4.5E3"

// Convert value to logical with Logical.From()
Logical.From(2)
```

Functions

```
// Define a function with (args) => calculations
let
  Hypotenuse = (x, y) => Number.Sqrt(Number.Power(x, 2) + Number.Power(y, 2))
in
  Hypotenuse
// each is syntactic sugar for a function with 1 arg named _
// Use it to iterate over lists and tables
each Number.Power(_, 2) // Same as (_) => Number.Power(_, 2)
```

> Lists

Creation

```
// Define a list with {}
//You can include different data types including null
{999, true, "DataCamp", null}

// Define a sequence of numbers with m..n
{-1..3, 100} // Equivalent to {-1, 0, 1, 2, 3, 100}

// Lists can be nested
{"outer", {"inner"}}

// Concatenate lists with &
{1, 4} & {4, 9} // Returns {1, 4, 4, 9}
```

Example lists

```
let
  Fruits = {"apple", null, "cherry"}
in
  Fruits

let
  Menage = {1, -1, 0, 1, 2, 13, 80, 579}
in
  Menage
```

Counting

```
// Access list elements with {}, zero-indexed
Fruits{0} // 1st element; returns "apple"

// Accessing elements outside the range throws an error
Fruits{3} // Throws an Expression.Error

// Append ? to return null if the index is out of range
Fruits{3}? // Returns null

// Get the first few elements with FirstN()
List.FirstN(Fruits, 2) // Returns {"apple", null}

// Get the last few elements with LastN()
List.LastN(Fruits, 2) // Returns {null, "cherry"}

// Get unique elements with Distinct()
List.Distinct(Menage) // Returns {1, -1, 0, 2, 13, 80, 579}
```

Selection

```
// Access list elements with {}, zero-indexed
Fruits{0} // 1st element; returns "apple"

// Accessing elements outside the range throws an error
Fruits{3} // Throws an Expression.Error

// Append ? to return null if the index is out of range
Fruits{3}? // Returns null

// Get the first few elements with FirstN()
List.FirstN(Fruits, 2) // Returns {"apple", null}

// Get the last few elements with LastN()
List.LastN(Fruits, 2) // Returns {null, "cherry"}

// Get unique elements with Distinct()
List.Distinct(Menage) // Returns {1, -1, 0, 2, 13, 80, 579}

// Get elements that match a criteria with Select()
List.Select(Menage, each _ > 1) // Returns {2, 13, 80, 579}

// Return true if all elements match a criteria with MatchesAll()
List.MatchesAll(Menage, each _ > 1) // Returns false

// Return true if any elements match a criteria with MatchesAny()
List.MatchesAny(Menage, each _ > 1) // Returns true

// Get value from list of length 1, or return default, with SingleOrDefault()
List.SingleOrDefault(Menage, -999) // Returns -999
```

Manipulation

```
// Sort items in ascending order with Sort()
List.Sort(Menage) // Returns {-1, 0, 1, 1, 2, 13, 80, 579}

// Sort items in descending order
List.Sort(Menage, Order.Descending) // Returns {579, 80, 13, 2, 1, 1, 0, -1}

// Reverse the order of items in a list with Reverse()
List.Reverse(Menage) // Returns {579, 80, 13, 2, 1, 0, -1, 1}

// Get non-null values with RemoveNulls()
List.RemoveNulls(Fruits) // Returns {"apple", "cherry"}

// Remove items by position with RemoveRange()
List.RemoveRange(Menage, 2, 3) // Returns {1, -1, 13, 80, 579}

// Repeat elements with Repeat()
List.Repeat({"one", "two"}, 2) // Returns {"one", "two", "one", "two"}

// Split list into lists of specified size with Split()
List.Split(Menage, 2) // Returns {{1, -1}, {0, 1}, {2, 13}, {80, 579}}

// Flatten lists by removing 1 level of nesting with Combine()
List.Combine({{"alpha"}, {"bravo", {"charlie", "delta"}}}) // Returns {"alpha", "bravo", {"charlie", "delta"}}
```

Equality & membership

```
// Lists are equal if they contain the same elements in the same order
{1, 2} = {1, 2} // true     {1, 2} = {2, 1} // false    {1, 2} = {1, 2, 3} // false
```

Calculations

```
// Get the minimum value in a list with Min()
List.Min({0, 7, -3, 2, 1})

// Get the minimum value in a list with Max()
List.Max({0, 7, -3, 2, 1})

// Get quantile values from a list with Percentile()
List.Percentile(
  {0, 7, -3, 2, 1}, {0.25, 0.5, 0.75},
  [PercentileMode=PercentileMode.SqlDisc]
)

// Get the sum of values in a list with Sum()
List.Sum({0, 7, -3, 2, 1})

// Get the product of values in a list with Product()
List.Product({0, 7, -3, 2, 1})

// Get the mean of values in a list with Average()
List.Average({0, 7, -3, 2, 1})

// Get the standard deviation of values in a list with StandardDeviation()
List.StandardDeviation({0, 7, -3, 2, 1})
```

Generation

```
// Generate random numbers between 0 and 1 with Random()
List.Random(3)

// Generate a sequence of dates with Dates()
List.Dates(#date(2023, 1, 1), 3, #duration(7, 0, 0, 0))

// Generate a sequence of numbers with Numbers()
List.Numbers(1, 5, 2)

// Mimic a for loop with Generate()
List.Generate(
  () => 2,
  each _ < 20,
  each Number.Power(_, 2))
```

Set operations

```
// Get values in all inputs with Intersect()
List.Intersect({{1, 3, 6, 10, 15}, {21, 15, 9, 3}, {0, 3, 6}}) // Returns {3}

// Get values in any inputs with Union()
List.Union({{1, 3, 6, 10, 15}, {21, 15, 9, 3}, {0, 3, 6}}) // Returns {0, 1, 3, 6, 9, 10, 15, 21}

// Get value in one set but not the other with Difference()
List.Difference({1, 3, 6, 10, 15}, {21, 15, 9, 3}) // Returns {1, 6, 10}
```

Learn Power BI Online at
www.DataCamp.com

datacamp

Our entire cheat sheet library is freely available to anyone, anywhere.

www.datacamp.com/cheat-sheet

Discover More