EXCEL OFF THE GRID

POWER QUERY QUICK STEPS

Ridiculously easy transformations for tricky situations

Making Power Query eaiser for longer

Contents

	Page
Introduction	3
Useful Reference	4
Parameter Functions	
fxGetNameParameter	10
fxGetTableParameter	11
Column Name Functions	
fxAutoCleanColumnNames	13
fxFlattenHeaderRows	14
fxRenameColumns	15
fxRenameColumnsByPosition	16
Data Transformation Functions	
fxBlankToNull	18
fxCartesianJoin	19
fxColumnReAlign	20
fxDeleteNullMoveLeft	21
fxExpandColumnDynamic	22
fxFillRightOrLeft	23
fxGroupColumnUnpivot	24
fxGroupRowUnpivot	25
fxFilterByList	26
fxPivotAllRows	27
fxRemoveNullColumns	28
fxRemoveToBottomNull	29
fxRemoveTopRowsUntilValue	30
fxRepeatValueToNull	31
fxUnstackOnInterval	32
fxUnstackOnValue	33
Text Functions	
fxMultiFindReplace	35
fxTextRemoveSpaces	36
Calculation Functions	
fxGroupRunningTotal	38
fxRunningTotal	39
fxPreviousRow	40

Contents

Lookup Functions
fxLookupApproximateMatch 42
fxLookupExactMatch 43

Data Type Functions
fxAutoDetectDataType 45

Ranking & Order Functions
fxRowNumberByGroup 47

Date and Time Functions
fxCalendarTable 49
fxCalendarTableNonStandard 50
fxFinancialPeriod 51
fxTimeGroups 52
fxTimeTable 53

Files & Folder Functions
fxCombineWorksheetsByName 55
fxCombineWorksheetsByPosition 57
fxExcelDataFromList 59
fxFilesInSharePointFolder 60

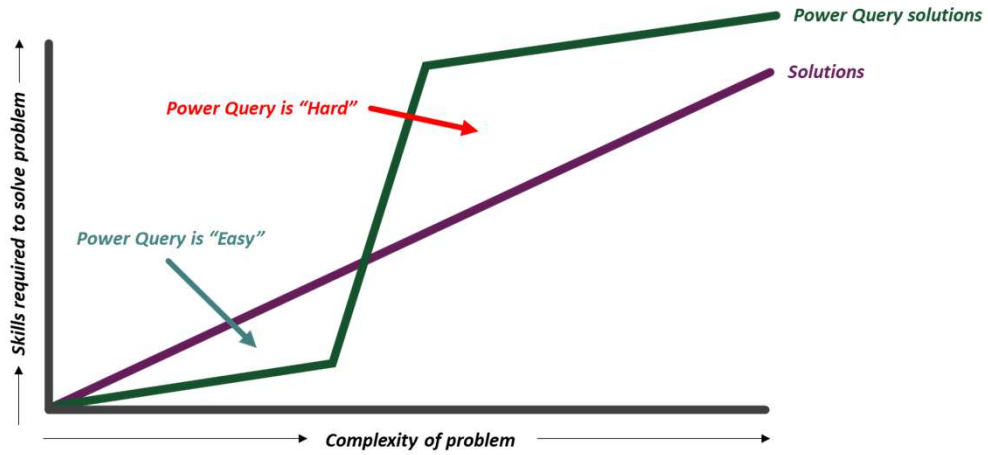
Nested Table Functions
fxTransformNestedTable 62

Introduction

The "problem" with Power Query

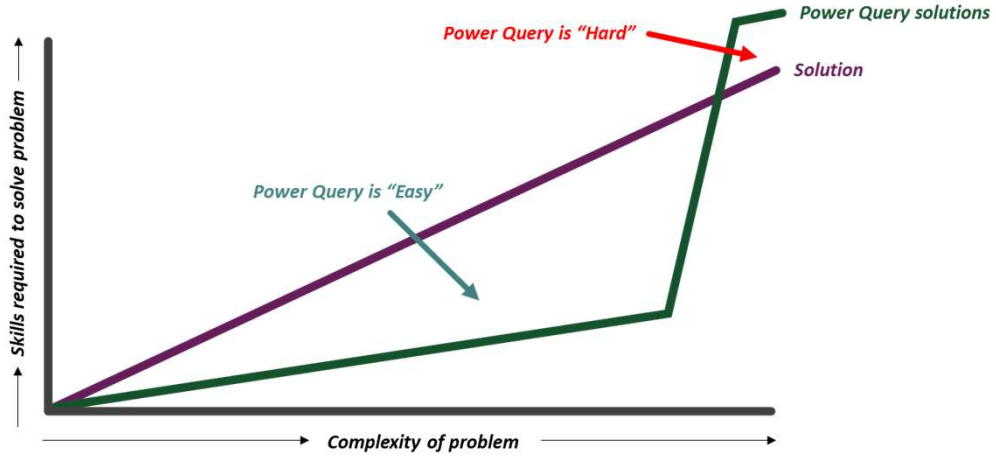
Power Query is easy…until it isn't, and then it's really hard.

Initially, Power Query appears easy; we can solve lots of problems with very little knowledge. However, it reaches a point where the learning curve accelerates massively. Suddenly, problems which are slightly more complex require significantly more complex solutions (and advanced M code skills).



Making Power Query easier for longer

Having seen many struggle with the learning curve, I wanted to do something to help. The majority of users are never going to be M code experts (and don't want to be either). Therefore, the question is, **how can we make Power Query easier for longer?** This is the goal of Power Query Quick Steps.



Power Query Quick Steps is a collection of custom functions. They provide users with ridiculously easy transformations for tricky situations.

For any users who become M code experts, I applaud you, it's a tough and frustrating road. For any users who just want to get the job done as easily as possible, this is for you.

Useful reference

Functions and arguments

The Quick Steps custom functions are constructed as a function name, followed by any arguments enclosed in brackets/parentheses.

Syntax*: functionName (argument1, argument2, [argument3])*

In this documentation and supporting videos, optional arguments are shown in square brackets (see argument3 above) this maintains consistency with Excel formulas which readers may already be familiar with. The square brackets do not need to be entered into Power Query and they will not appear in Power Query's IntelliSense.

Even if no arguments are needs, the opening and close bracket are still required.

Power Query is case sensitive, so take care to ensure arguments are entered correctly.

Arguments may require different data types: tables, text, numbers, logical, dates or lists. These are all detailed below.

Tables

A table is a result which contains a 2-dimensional grid. It may be the result for a query, or a step from within an existing query.

Where a table name contains no spaces they are referenced by their name (e.g. *MyQuery*). However, where a table name contains spaces or some special characters they are enclosed in double quotes and proceeded by a # symbol (e.g. *#"My Query")*.

Text

Text is any value enclosed in double quotes (e.g. *"Alpha"*). This excludes Table names, which are also preceded by a # symbol as noted above.

Numbers

Numbers may be positive, negative and include decimal places.

Note: Any numbers enclosed in double quotes are treated as text.

Logical

Logical values (also known as boolean values) can have the value true or false. In Power Query, logical values are entered as lower case.

Useful reference

Dates

Any argument requiring a date must receive that value a date data type.

Different countries use different date formats, therefore the presentation of a date may appear different in your region.

The following are examples of valid dates:

- Specific date: ***#date(2023,3,31)***
- Specific date from text (local): ***Date.From("31/01/2023")*** – example shows dd/mm/yyyy local format
- Specific date from text (ISO8601): ***Date.From("2023-01-31")***
- Today's date (local): ***Date.From(DateTimeZone.LocalNow())***
- Today's date (UTC): ***Date.From(DateTimeZone.UtcNow())***
- Earliest date in date column: ***List.Min(QueryName[DateColumn])***
- Latest date in date column: ***List.Max(QueryName[DateColumn])***
- Parameter dates: Dates using the **fxGetNameParameter** or **fxGetTableParameter** functions

Lists

Many custom functions include list arguments. Lists are special objects in Power Query and can be applied in multiple ways.

- **Manual lists**
  Shown in curly brackets with a comma separator between each element in the list.
  Example: ***{"Alpha","Bravo","Charlie","Delta"}***
- **Table column lists**
  An individual table column is a list.
  Example: ***#"Table Name"[Column Name]***
- **List function lists**
  Some Power Query functions return a list
  Example: ***Text.Split("Alpha,Bravo,Charlie,Delta",",")***

Note: Lists can contain any data type.

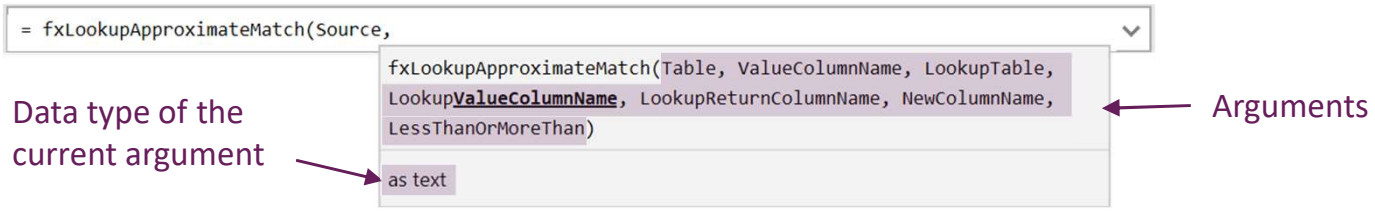
Other data types

Power Query's other data types are not included as arguments with the Quick Steps custom functions.

Useful reference

IntelliSense

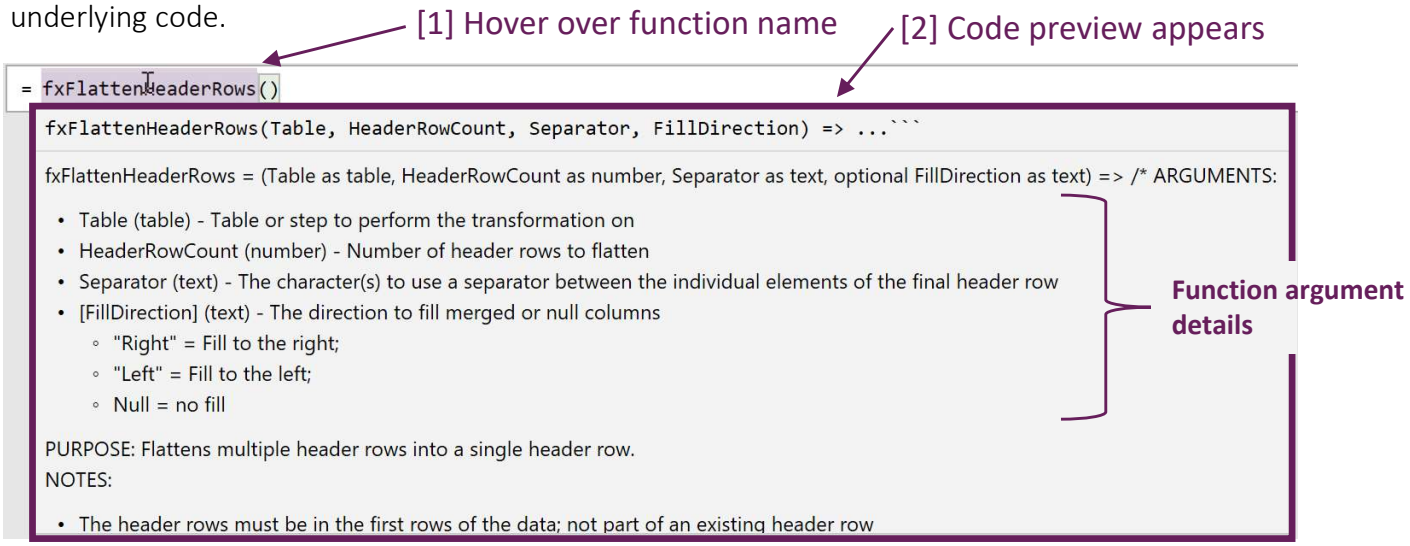
When entering a custom function, the IntelliSense for function arguments will become visible to help enter arguments in the correct order.



Due to a bug in Power Query, the bold and underlined text may not be the next argument to enter into the formula bar. For example, in the screenshot above the second argument is **ValueColumnName**, but Power Query has highlighted part of the **LookupValueColumnName** argument.

Code Preview

While entering a custom function, hovering over the name of the function displays a preview of the underlying code.



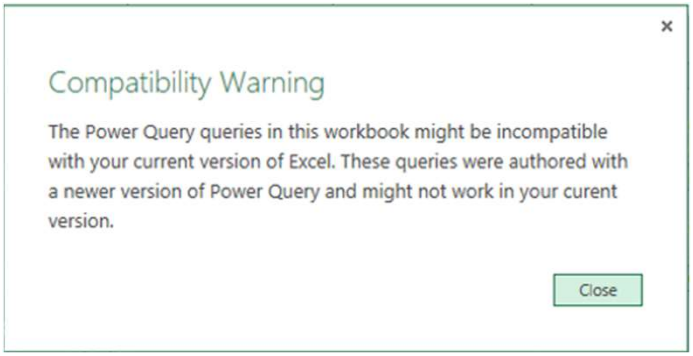
Within Excel, the code preview cannot scroll to see more information.

Where space allows, Quick Step displays detail for each function argument.

Useful reference

Compatibility warning

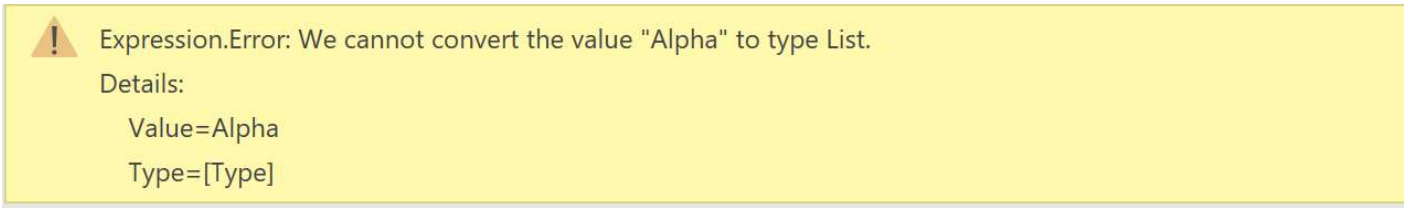
When using custom functions or example files the Compatibility Warning message may appear.



Unless explicitly stated in the documentation, all functions and examples should be compatible with all versions of Power Query for Excel and Power BI.

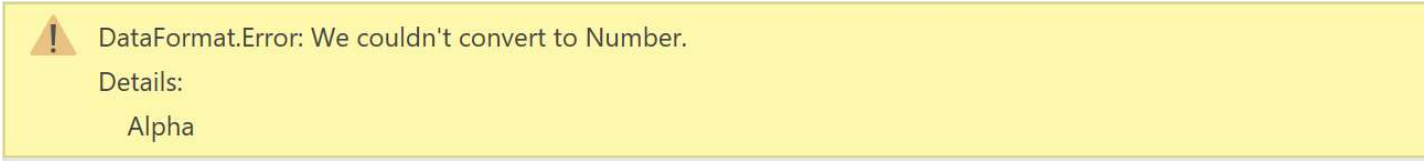
Error messages

Power Query error messages can be difficult to understand. When using the Quick Steps custom functions,



In the screenshot above, a value of **"Alpha"** has been provided for an argument which requires a list. Therefore, the argument value must be changed to **{"Alpha"}**.

If a custom function calculates a value which is the wrong data type, it may also cause an error.



In the screenshot above, a value of "Alpha" exists in the data set, however, the custom function requires the data type to be a number for it to calculate correctly.

Useful reference

Query folding

The Quick Step custom functions are designed primarily for dealing with files and folders, rather than database connections. Therefore, functions may not be optimized for query folding.

Bugs & feature requests

Every effort has been made to make this product and all associated material as complete and accurate as possible, but no warranty or fitness is implied. The information and downloads are provided on an “as is” basis. The author and publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from any information or downloads contained in this product.

If you find any bugs or errors, please report them here: https://excelofftthegrid.com/report-bug.

To request new features, or if you discover faster methods to perform the custom functions, visit this page: https://excelofftthegrid.com/feature-request.

Parameter Functions

fxGetNameParameter

PURPOSE:
Returns a value from a named range in the current workbook to use as a parameter inside a query.

SYNTAX:
fxGetNameParameter (NamedRange, [DataTypeName])

- **NamedRange** (text) - Excel named range containing the parameter.
- **[DataTypeName]** (text) - Data type of the parameter value.
  - Valid values include "Text", "Number", "Date/Time", "Date", "Time", "Duration", "True/False".
  - If excluded, the default type is Any.

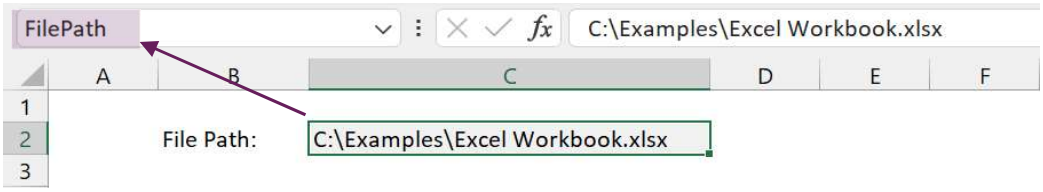
NOTES:
- (None)

EXAMPLE #1

The Source step includes reference to a hard coded file path

```
= Excel.Workbook(File.Contents("C:\Examples\Excel Workbook.xlsx"), null, true)
```

The Excel workbook contains a named range called **FilePath**



Replace the hardcoded path with the named range using **fxGetNameParameter**.

```
= Excel.Workbook(File.Contents(fxGetNameParameter("FilePath","Text")), null, true)
```

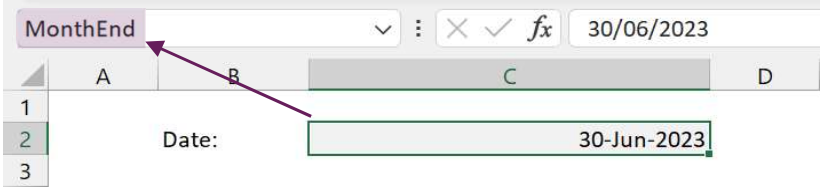
- **"FilePath"**: The named range
- **"Text"**: FilePath is a text data type

EXAMPLE #2

A step filters to include all dates before 30 June 2023

```
= Table.SelectRows(#"Changed Type", each [Date] <= #date(2023, 6, 30))
```

The Excel workbook contains a named range called Month End



Replace the hardcoded date with the named range using **fxGetNameParameter**.

```
= Table.SelectRows(#"Changed Type", each [Date] <= fxGetNameParameter("MonthEnd","Date"))
```

- **"MonthEnd"**: The named range
- **"Date"**: Month End is a date data type

fxGetTableParameter

PURPOSE:
Returns a value from a table in the current workbook to use as a parameter inside a query.

SYNTAX:
fxGetTableParameter (TableName, ParameterName, ParameterNameColumn, ParameterValueColumn, [DataTypeName])

- **TableName** (text) - Table containing the parameter.
- **ParameterName** (text) - Name of the parameter in the table.
- **ParameterNameColumn** (text) - Column name containing the parameter names.
- **ParameterValueColumn** (text) - Column name containing the value to return.
- **[DataTypeName]** (text) - Data type of the parameter value.
    - Valid values include "Text", "Number", "Date/Time", "Date", "Time", "Duration", "True/False".
    - If excluded, the default type is Any.

NOTES:
- (None)

TABLE FOR EXAMPLES

The workbook includes a table called **Parameters**

Name	Value
File Path	C:\Examples\Power Query\Excel Workbook.xlsx
Report Date	31-Mar-2023

← Parameters

EXAMPLE #1

The Source step includes reference to a hard coded file path

```
= Excel.Workbook(File.Contents("C:\Examples\Excel Workbook.xlsx"), null, true)
```

Replace the hardcoded path with the value from the Parameters table using **fxGetTableParameter**.

```
= Excel.Workbook(File.Contents(fxGetTableParameter("Parameters", "File Path", "Name", "Value", "Text")), null, true)
```

- **"Parameters"**: The table name
- **"File Path"**: The lookup value
- **"Name"**: The lookup column
- **"Value"**: The return column
- **"Text"**: The File Path is a text data type

EXAMPLE #2

A step filters to include all dates before 30 June 2023

```
= Table.SelectRows(#"Changed Type", each [Date] <= #date(2023, 6, 30))
```

Replace the hardcoded date with the value from the Parameters table using **fxGetTableParameter.**

```
= Table.SelectRows(#"Changed Type", each [Date] <= fxGetTableParameter("Parameters", "Date", "Name", "Value", "Date")
```

- **"Parameters"**: The table name
- **"Date"**: The lookup value
- **"Name"**: The lookup column
- **"Value"**: The return column
- **"Date"**: Date is a date data type

Column Name Functions

fxAutoCleanColumnNames

PURPOSE:
Changes columns names to new names based on rules. Inserts space and applies capitalization when a string:

- Changes from numbers to text.
- Changes from text to numbers.
- Changes from lower case to upper case.
- Includes an underscore.

SYNTAX:
fxAutoCleanColumnNames (Table, [ColumnNamesList], [ListTypeIsInclude])

- **Table** (table) - Table or step to perform the transformation on.
- **[ColumnNamesList]** (list of text) - List of column names to clean (e.g. {"Column1","Column2"}).
  - If excluded apply to all columns.
- **[ListTypeIsInclude]** (logical) - Switch to determine if ColumnNamesList includes or excludes the list of names to clean.
  - true: apply to items in the list
  - false: apply to items not in the list

NOTES:

- Only use **ColumnNamesList** and **ListTypeIsIncude** where the action is to include or exclude specific column headers.
- If used, both optional arguments are required.

EXAMPLE

The source Excel file includes column headers which are difficult to read and inconsistent.

	A	B	C	D	E
1	itemName	Phase6Status	90DayOutstanding	Lead_Manager	
2	Alpha	Live	20	Tim	
3	Bravo	Live	30	Sally	
4	Charlie	Draft	15	Dave	
5	Delta	Live	23	Jessica	
6					

Difficult to read & inconsistent column names

Load the data into Power Query and ensure headers are promoted. Use the **fxAutoCleanColumnNames** function to automatically clean the existing column names.

```
= fxAutoCleanColumnNames(#"Promoted Headers")
```

- **#"Promoted Headers"**: Name of previous step

Result:

	Item Name	Phase 6 Status	90 Day Outstanding	Lead Manager
1	Alpha	Live	20	Tim
2	Bravo	Live	30	Sally
3	Charlie	Draft	15	Dave
4	Delta	Live	23	Jessica

Clean column names

fxFlattenHeaderRows

PURPOSE:
Flattens multiple header rows into a single header row.

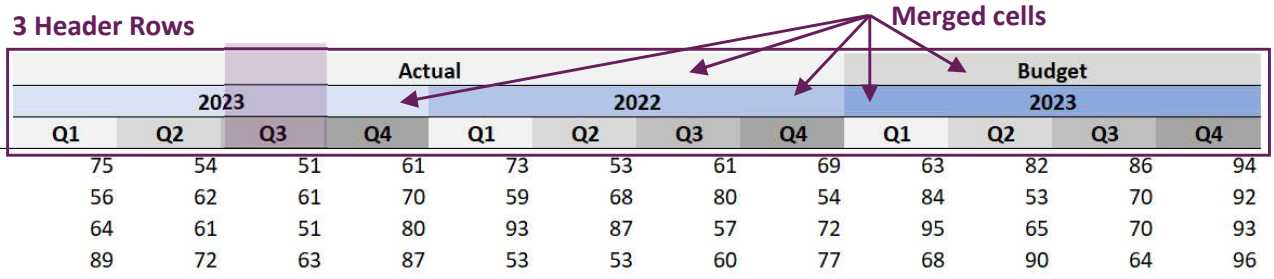
SYNTAX:
fxFlattenHeaderRows (Table, HeaderRowCount, Separator, [FillDirection])

- **Table** (table) - Table or step to perform the transformation on.
- **HeaderRowCount** (number) - Number of header rows to flatten.
- **Separator** (text) - The character(s) to use as a separator between the individual elements of the final header row.
- **[FillDirection]** (text) - The direction to fill the missing column data.
  - "Right": Fill to the right.
  - "Left": Fill to the left.
  - Null: no fill.

NOTES:
- The header rows must be in the first rows of the data; not part of an existing header row.

EXAMPLE

The source Excel file includes data with multiple header rows (including merged cells).



	Actual								Budget			
	2023				2022				2023			
Item	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Alpha	75	54	51	61	73	53	61	69	63	82	86	94
Bravo	56	62	61	70	59	68	80	54	84	53	70	92
Charlie	64	61	51	80	93	87	57	72	95	65	70	93
Delta	89	72	63	87	53	53	60	77	68	90	64	96

Load the data into Power Query, ensure headers are not promoted. Note: cells are no longer merged (see Actual 2023 Q1).

	Column1	Column2	Column3	Colulumn4	Colum5	Column6	Column7
1	null	Actual	null	null	null	null	null
2	null	2023	null	null	null	2022	null
3	Item	Q1	Q2	Q3	Q4	Q1	Q2
4	Alpha	75	54	51	61	73	53
5	Bravo	56	62	61	70	59	68
6	Charlie	64	61	51	80	93	87
7	Delta	89	72	63	87	53	53

Use the **fxFlattenHeaderRows** function to flatten the 3 header rows to a single row. Missing headers are filled to the right.

```
= fxFlattenHeaderRows(Source, 3, "|", "Right")
```

- **Source**: Name of previous step
- **3**: Number of header rows
- **"|"**: The character used as the separator when flattening rows
- **"Right"**: Headers are filled to the right, replacing any null values.

Result: Header rows flattened into a single row

```
= fxFlattenHeaderRows(Source, 3, "|", "Right")
```

	Item	Actual|2023|Q1	Actual|2023|Q2	Actual|2023|Q3	Actual|2023|Q4	Actual|2022|Q1	Actual|2022|Q2
1	Alpha	75	54	51	61	73	53
2	Bravo	56	62	61	70	59	68
3	Charlie	64	61	51	80	93	87
4	Delta	89	72	63	87	53	53

fxRenameColumns

PURPOSE:
Renames columns, if the names exist.

SYNTAX:
fxRenameColumns (Table, ColumnNamesList, NewColumnNamesList)

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnNamesList** (list of text) - List of columns to rename (e.g. {"Column1","Column2","Column3"}).
- **NewColumnNamesList** (list of text) - List of new column names (e.g. {"NewColumn1", "NewColumn2", "NewColumn3"}).

NOTES:
- The old and new column names must be in the corresponding location in each list; therefore, the lists will contain the same number of items.

EXAMPLE

The source file includes a table with the following columns:

	Item	Region	Size	Value
1	Alpha	North	Small	56
2	Bravo	South	Large	23
3	Charlie	East	Small	45

Column names to be replaced are contained in a Table:

	Find	Replace
1	Item	Product
2	region	Division
3	Region	Division

Query Name: FindReplace

Column names does not exist in Table, but will not cause error

Use **fxRenameColumns** to rename columns, if the column name exists

```
= fxRenameColumns(#"Changed Type",FindReplace[Find],FindReplace[Replace])
```

- **#"Changed Type"**: Name of previous step
- **FindReplace[Find]**: The list of column names to find
- **FindReplace[Replace]**: The list of column names to replace the old names

Result:

	Product	Division	Size	Value
1	Alpha	North	Small	56
2	Bravo	South	Large	23
3	Charlie	East	Small	45
4	Delta	West	Large	89

fxRenameColumnsByPosition

PURPOSE:
Renames columns based on their position (zero based).

SYNTAX:
fxRenameColumnsByPosition (Table, ColumnPositionsList, NewColumnNamesList)

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnPositionsList** (list of numbers) - List of column numbers (e.g. {0,2,4}).
- **NewColumnNamesList** (list of text) - List of new column names (e.g. {"NewColumn1", "NewColumn2", "NewColumn3"}).

NOTES:
- The column positions and new column names must be in the corresponding location in each list; therefore, the lists will contain the same number of items.

EXAMPLE

The source file includes a table with the following columns:

	Item	Region	Size	Value
1	Alpha	North	Small	56
2	Bravo	South	Large	23
3	Charlie	East	Small	45

Use **fxRenameColumnsByPosition** to rename the first and third columns.

```
= fxRenameColumnsByPosition(#"Changed Type",{0,2},{"Product","Type"})
```

- **#"Changed Type"**: Name of previous step
- **{0,2}**: Replace the first and third columns (zero based)
- **{"Product", "Type"}**: The list of column names to replace

Result:

	Product	Region	Type	Value
1	Alpha	North	Small	56
2	Bravo	South	Large	23
3	Charlie	East	Small	45
4	Delta	West	Large	89

Data Transformation Functions

fxBlankToNull

PURPOSE:
Converts all blank values in a table or column to null.

SYNTAX:
fxBlankToNull (Table, [ColumnNamesList])

- **Table** (table) - Table or step to perform the transformation on.
- **[ColumnNamesList]** (list of text) - List of column names to convert blanks to null (e.g. {"Column1","Column2"}).
    - If excluded apply to all columns.

NOTES:
- (none)

EXAMPLE

The source data includes blank values which we wish to be null.

	Item	Region	Value
1	Alpha	North	56
2	Bravo	North	null
3	Charlie	North	97
4	Delta	North	null
5	Alpha		120
6			62
7	Charlie		null
8			81

Use the **fxBlankToNull** function to convert all blank values in a table to null.

```
= fxBlankToNull(#"Changed Type")
```

- **#"Changed Type"**: Name of previous step

Result:

	Item	Region	Value
1	Alpha	North	56
2	Bravo	North	null
3	Charlie	North	97
4	Delta	North	null
5	Alpha	null	120
6	null	null	62
7	Charlie	null	null
8	null	null	81

Add a list of column names to change only the specified columns:

```
= fxBlankToNull(#"Changed Type", {"Item", "Region"})
```

- **#"Changed Type"**: Name of previous step
- **{"Item", "Region"}**: List of column names

fxCartesianJoin

PURPOSE:
Performs a cartesian join on all tables included in a list. (i.e. creates a table of all combinations).

SYNTAX:
fxCartesianJoin (TableList)

- **TableList** (list of tables) - List of tables to join (e.g. {Table1,Table2,Table3}).

NOTES:
- TableList may also include names of previous steps.

EXAMPLE

There are 3 queries in Power Query (**Item**, **Region**, **Size**)

Query: Item

	Item	Value
1	Alpha	1
2	Bravo	2
3	Charlie	3
4	Delta	4
5	Echo	5
6	Foxtrot	6
7	Golf	7
8	Hotel	8

Query: Region

	Region	Value
1	North	11
2	South	12
3	East	13
4	West	14
5	Central	15

Query: Size

	Size	Value
1	Small	21
2	Medium	22
3	Large	23

In a blank query, enter the following formula:

```
= fxCartesianJoin({Item,Region,Size})
```

Result (complete list of everything of everything):

	Item	Value	Region	Value_1	Size	Value_2
1	Alpha	1	North	11	Small	21
2	Alpha	1	North	11	Medium	22
3	Alpha	1	North	11	Large	23
4	Alpha	1	South	12	Small	21
5	Alpha	1	South	12	Medium	22
6	Alpha	1	South	12	Large	23
7	Alpha	1	East	13	Small	21
8	Alpha	1	East	13	Medium	22
9	Alpha	1	East	13	Large	23
10	Alpha	1	West	14	Small	21
11	Alpha	1	West	14	Medium	22
12	Alpha	1	West	14	Large	23
13	Alpha	1	Central	15	Small	21
14	Alpha	1	Central	15	Medium	22

fxColumnReAlign

PURPOSE:
Re-align columns where data does not stack correctly in a single column.

SYNTAX:
fxColumnReAlign (Table)

- **Table** (table) - Table or step to perform the transformation on.

NOTES:
- Re-alignment works where empty values are null values, and there are no genuine null values in the data set.

EXAMPLE

The source Excel file includes data where each row has the same number of values, but they are not aligned in a column.

	A	E	G	H	J	O
1	**Ref**	**Name**	**Value**		**Outstanding**	**Category**
2	ABL102	Abletone	145645		119402	Premium
3	SEM056	Semaphore	74589		53420	Premium
4	WHI004	Whittlesford Oak	65748		11930	Standard
5						
6	GRE213	Green and Blue		96457	76794	Premium
7	GRI104	Grinsborough		54163	41594	Standard
8						

Load the data into Power Query. Use the **fxColumnReAlign** function to place the data into the correct columns.

```
= fxColumnReAlign(Data_Sheet)
```

- **Data_Sheet**: Name of previous step

Result:

	Data.1	Data.2	Data.3	Data.4	Data.5
1	Ref	Name	Value	Outstanding	Category
2	ABL102	Abletone	145645	119402	Premium
3	SEM056	Semaphore	74589	53420	Premium
4	WHI004	Whittlesford Oak	65748	11930	Standard
5	*null*	*null*	*null*	*null*	*null*
6	GRE213	Green and Blue	96457	76794	Premium
7	GRI104	Grinsborough	54163	41594	Standard

fxDeleteNullMoveLeft

PURPOSE:
Deletes null values in a column and shifts the cells left to fill the gap.

SYNTAX:
fxDeleteNullMoveLeft (Table, ColumnName)

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Column name containing the null values to delete.

NOTES:
- Works where empty values are represented by null and, and there are no genuine null values in the column.

EXAMPLE

The data contains columns which are not aligned.

	Item	Size	Credit	Debit	Column1
1	Alpha	Small	10	null	null
2	Bravo	Medium	null	-60	null
3	Charlie	Small	50	null	null
4	Delta	~~null~~	~~Small~~	40	null
5	Echo	~~null~~	~~Medium~~	null	-10
6	Foxtrot	~~null~~	~~Large~~	60	null
7	Golf	Large	null	-90	null
8	Hotel	Large	20	null	null

fxColumnReAlign is not suitable as some columns contain valid null values. Use **fxDeleteNullMoveLeft** to remove nulls in the Size column and shift the values left.

```
= fxDeleteNullMoveLeft(#"Changed Type", "Size")
```

- **#"Changed Type"**: Name of previous step
- **"Size"**: Name of the column to remove null values from

Result:

	Item	Size	Credit	Debit	Column1
1	Alpha	Small	10	null	null
2	Bravo	Medium	null	-60	null
3	Charlie	Small	50	null	null
4	Delta	Small	40	null	null
5	Echo	Medium	null	-10	null
6	Foxtrot	Large	60	null	null
7	Golf	Large	null	-90	null
8	Hotel	Large	20	null	null

fxExpandColumnDynamic

PURPOSE:
Expands a column dynamically including adding any new data.

SYNTAX:
fxExpandColumnDynamic (Table, ColumnName, [PromoteHeaders])

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Name of column containing the tables to expand.
- **[PromoteHeaders]** (logical) - Should headers be promoted prior to expanding.
  - true: Promote headers.
  - false / null: Do not promote headers.

NOTES:
- Use **fxAutoDetectDataType** to automatically apply data type to columns

EXAMPLE

WARNING: This hardcodes column names.

A workbook has two sheets **Data #1** and **Data #2**. Later an additional sheet is added (**Data #3)** with a new column (**Size**).

	Item	Region	Value
1	Alpha	North	112
2	Bravo	South	85
3	Charlie	East	59
4	Delta	West	77
5	Alpha	Central	143
6	Bravo	North	114
7	Charlie	South	139
8	Delta	East	128
9	Alpha	West	148
10	Bravo	Central	133
11	Charlie	North	122
12	Delta	South	69

Data #1

	Item	Region	Value
1	Alpha	South	142
2	Bravo	East	79
3	Charlie	West	97
4	Delta	Central	135
5	Alpha	North	59
6	Bravo	South	58
7	Charlie	East	149
8	Delta	West	138
9	Alpha	Central	114
10	Bravo	North	58
11	Charlie	South	73
12	Delta	East	75

Data #2

	Item	Region	Value	Size
1	Alpha	West	69	Small
2	Bravo	Central	99	Small
3	Charlie	North	109	Medium
4	Delta	South	100	Large
5	Alpha	East	113	Small
6	Bravo	West	92	Large
7	Charlie	Central	116	Medium
8	Delta	North	114	Medium
9	Alpha	South	98	Small
10	Bravo	East	89	Large
11	Charlie	West	83	Small
12	Delta	Central	121	Medium

New column

Data #3 (Added Later)

To ensure new columns are included with the data is expanded, use **fxExpandColumnDynamic.**

```
= fxExpandColumnDynamic(#"Removed Other Columns","Data",true)
```

- **#"Removed Other Columns"**: Name of previous step
- **"Data"**: Name of the column to expand
- **true**: Promote column headers before expanding data

New column added

Result:

	Name	Item	Region	Value	Size
1	Data #1	Alpha	North	112	null
2	Data #1	Bravo	South	85	null
23	Data #2	Charlie	South	73	null
24	Data #2	Delta	East	75	null
25	Data #3	Alpha	West	69	Small
26	Data #3	Bravo	Central	99	Small
27	Data #3	Charlie	North	109	Medium
28	Data #3	Delta	South	100	Large
29	Data #3	Alpha	East	113	Small
30	Data #3	Bravo	West	92	Large
31	Data #3	Charlie	Central	116	Medium
32	Data #3	Delta	North	114	Medium
33	Data #3	Alpha	South	98	Small

fxFillRightOrLeft

PURPOSE:
Fills null values with the values from the left or right.

SYNTAX:
fxFillRightOrLeft (Table, [FillRight])

- **Table** (table) - Table or step to perform the transformation on.
- **[FillRight]** (logical) - Determine the direction to fill from.
    - true/null: fill values to the right.
    - false: fill values to the left.

NOTES:
- (None)

EXAMPLE

The data includes an **Item** column and a **New Name** column. Where there is no value in the **New Name** column we want to fill the value from the **Item** column.

	Item	New Name	Value
1	Alpha	Zulu	73
2	Bravo	null	56
3	Charlie	Yankee	84
4	Delta	X-Ray	87
5	Echo	null	81
6	Foxtrot	null	60

Use **fxFillRightOrLeft** to replace the null values.

```
= fxFillRightOrLeft(#"Changed Type",true)
```

- **#"ChangedType"**: Name of previous step
- **true**: Fill the values right (i.e. replaced by the Item column).

Result:

	Item	New Name	Value
1	Alpha	Zulu	73
2	Bravo	Bravo	56
3	Charlie	Yankee	84
4	Delta	X-Ray	87
5	Echo	Echo	81
6	Foxtrot	Foxtrot	60

fxGroupColumnUnpivot

PURPOSE:
Unpivots multiple columns presented in a repeating column interval.

SYNTAX:
fxGroupColumnUnpivot (Table, GroupLength, [KeepColumnNamesList], [NewColumnNamesList])

- **Table** (table) - Table or Step name to perform the transformation on.
- **Group Length** (number) - Number of columns contained in each group of columns.
- **[KeepColumnNamesList]** (list) - List of column names that are unchanged.
- **[NewColumnNamesList]** (list) - List of names to apply to unpivoted columns. Default column names applied if null.

NOTES:
- NewColumnNamesList must contain the same number of items as the repeating column pattern.

EXAMPLE

The example data contains a column followed by 3 pairs of Size and Values columns.

Item	Size	Value	Size	Value	Size	Value
Alpha	Small	10	Medium	24	Large	24
Bravo	Small	20	Medium	19	Large	19
Charlie	Small	36	Medium	36	Large	36
Delta	Small	28	Medium	52	Large	52

To get the Size and Value columns into two columns use the **fxGroupColumnUnpivot** function:

```
= fxGroupColumnUnpivot(#"Promoted Headers",2,{"Item"},{"Size","Value"})
```

- **#"Promoted Headers"**: Name of the previous step
- **2**: The number of columns in each group
- **{"Item"}:** The column name which is retained
- **{"Size","Value"}**: The list of column names to give to the columns

Result:

	Item	Size	Value
1	Alpha	Small	10
2	Alpha	Medium	24
3	Alpha	Large	24
4	Bravo	Small	20
5	Bravo	Medium	19
6	Bravo	Large	19
7	Charlie	Small	36
8	Charlie	Medium	36
9	Charlie	Large	36
10	Delta	Small	28
11	Delta	Medium	52
12	Delta	Large	52

fxGroupRowUnpivot

PURPOSE:
Unpivots multiple rows presented in a repeating row interval.

SYNTAX:
fxGroupRowUnpivot (Table, KeepColumnNamesList, [NewColumnNamesList], [MissingDataFillDirection])

- **Table** (table) - Table or step to perform the transformation on.
- **KeepColumnNamesList** (list) – List of column names that are unchanged.
- **[NewColumnNamesList]** (list) - List of names to apply to unpivoted columns. Default names applied if null.
- **[MissingDataFillDirection]** (text) - Direction to fill the KeepColumnsNamesList to avoid null values.
  - "Up" - Fills up
  - "Down" / null / any other value - Fills down

NOTES:
- If used, NewColumnNamesList must contain the same number of items as the repeating row pattern.

EXAMPLE

A workbook contains a Table where the data is within paired rows.

	Item	Monday	Tuesday	Wednesday	Thursday	Friday
1	Alpha	01/01/2024 00:00:00	02/01/2024 00:00:00	03/01/2024 00:00:00	05/01/2024 00:00:00	06/01/2024 00:00:00
2	null	10	15	12	13	18
3	Bravo	01/01/2024 00:00:00	02/01/2024 00:00:00	03/01/2024 00:00:00	05/01/2024 00:00:00	06/01/2024 00:00:00
4	null	22	13	18	16	19
5	Charlie	01/01/2024 00:00:00	02/01/2024 00:00:00	03/01/2024 00:00:00	05/01/2024 00:00:00	06/01/2024 00:00:00
6	null	28	24	26	32	41
7	Delta	01/01/2024 00:00:00	02/01/2024 00:00:00	03/01/2024 00:00:00	05/01/2024 00:00:00	06/01/2024 00:00:00
8	null	8	6	4	5	7

Use **fxGroupRowUnpivot** to convert the data into columns.

```
= fxGroupRowUnpivot(Source,{"Item"}, {"Date", "Value"}, "Down")
```

- **Source**: Name of previous step
- **{"Item"}**: List of column names that are not unpivoted.
- **{"Date", "Value"}**: The new column names for the paired rows
- **"Down"**: The fill direction to apply to the Item column to replace null values.

Result:

	Item	Date	Value
1	Alpha	01/01/2024 00:00:00	10
2	Alpha	02/01/2024 00:00:00	15
3	Alpha	03/01/2024 00:00:00	12
4	Alpha	05/01/2024 00:00:00	13
5	Alpha	06/01/2024 00:00:00	18
6	Bravo	01/01/2024 00:00:00	22
7	Bravo	02/01/2024 00:00:00	13
8	Bravo	03/01/2024 00:00:00	18
9	Bravo	05/01/2024 00:00:00	16
10	Bravo	06/01/2024 00:00:00	19
11	Charlie	01/01/2024 00:00:00	28
12	Charlie	02/01/2024 00:00:00	24
13	Charlie	03/01/2024 00:00:00	26
14	Charlie	05/01/2024 00:00:00	32

fxFilterByList

PURPOSE:
Filters a table column based on a list.

SYNTAX:
fxFilterByList (Table, ColumnName, FilterList, [ListIsInclude])

- **Table** (table) - Table or Step name to perform the transformation on.
- **ColumnName** (text) - Name of column containing values.
- **FilterList** (list) - List to filter by (e.g. {"Alpha","Bravo","Charlie"} or {1,2,3}).
- **[ListIsInclude]** (logical) - Does the list exclude or include the items in the list.
    - true / null: Include the items in the list.
    - false: Exclude the items in the list.

NOTES:
- (None)

EXAMPLE

A workbook has two tables **Data** and **Filter**. Both The goal is the filter the **Item** column by those items in the **List** column, Both Tables have are loaded into Power query

Item	Value
Alpha	56
Bravo	89
Charlie	74
Delta	51
Alpha	59
Bravo	63
Charlie	78
Delta	57

Table Name: Data

List
Alpha
Charlie

Table Name: Filter

To filter by a the list, use the **fxFilterByList** function:

```
= fxFilterByList(#"Changed Type", "Item", Filter[List])
```

- **#"Changed Type"**: Name of previous step
- **"Item"**: Name of the column to filter
- **Filter[List]**: Table name and column to filter by

Result:

	Item	Value
1	Alpha	56
2	Charlie	74
3	Alpha	59
4	Charlie	78

Set the optional ListIsInclude parameter to false to invert the filter

```
= fxFilterByList(#"Changed Type", "Item", Filter[List], false)
```

- **#"Changed Type"**: Name of previous step
- **"Item"**: Name of the column to filter
- **Filter[List]**: Table name and column to filter by
- **false**: invert the list to exclude the items in the list

Exclude items in list

	Item	Value
1	Bravo	89
2	Delta	51
3	Bravo	63
4	Delta	57

fxPivotAllRows

PURPOSE:
Pivots all rows without causing an error.

SYNTAX:
fxPivotAllRows (Table, PivotByColumnName, PivotValuesColumnName)

- **Table** (table) - Table or step to perform the transformation on.
- **PivotByColumnName** (text) - Name of column to pivot on.
- **PivotValuesColumnName** (text) - Name of values column to include in the pivot.

NOTES:
- (None)

EXAMPLE

When Pivoting data in Power Query, it can cause errors where the column combinations are not unique.

	Item	Region	Size
1	Alpha	North	Small
2	Alpha	North	Large
3	Alpha	South	Small
4	Alpha	East	Large
5	Bravo	North	Small
6	Bravo	South	Large
7	Bravo	East	Small
8	Bravo	West	Large

Pivoting on **Region** with the **Aggregate Value Function** for **Size** set as **Don't Aggregate** causes an error.

	Item	North	South	East	West
1	Alpha	Error	Small	Large	*null*
2	Bravo	Small	Large	Small	Large

Use **fxPivotAllRows** to pivot on **Region** using **Size** as values.

```
= fxPivotAllRows(#"Changed Type", "Region", "Size")
```

- **#"Changed Type"**: Name of previous step
- **"Region"**: Pivot on the **Region** column
- **"Size"**: Place unaggregated text from the Size column into the values section

Result:

	Item	North	South	East	West
1	Alpha	Small	Small	Large	*null*
2	Alpha	Large	*null*	*null*	*null*
3	Bravo	Small	Large	Small	Large

fxRemoveNullColumns

PURPOSE:
Removes any columns with 100% null values.

SYNTAX:
fxRemoveNullColumns (Table)

- **Table** (table) - Table or step to perform the transformation on.

NOTES:
- (None)

EXAMPLE

The source Excel file includes columns that contain no values.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Ref**			**Name**		**Value**		**Outstanding**				**Category**	
2	ABL102			Abletone		145645		119402				Premium	
3	SEM056			Semaphore		74589		53420				Premium	
4	WHI004			Whittlesford Oak		65748		11930				Standard	
5	GRE213			Green and Blue		96457		76794				Premium	
6	GRI104			Grinsborough		54163		41594				Standard	
7													

Null value columns

Load the data into Power Query. Use the **fxRemoveNullColumns** function to retain only columns with data.

```
= fxRemoveNullColumns(Data_Sheet)
```

- **Data_Sheet**: Name of previous step

Result:

	Column1	Column4	Column6	Column8	Column12
1	Ref	Name	Value	Outstanding	Category
2	ABL102	Abletone	*145645*	*119402*	Premium
3	SEM056	Semaphore	*74589*	*53420*	Premium
4	WHI004	Whittlesford Oak	*65748*	*11930*	Standard
5	GRE213	Green and Blue	*96457*	*76794*	Premium
6	GRI104	Grinsborough	*54163*	*41594*	Standard

fxRemoveTopBottomNull

PURPOSE:
Removes null values from top and/or bottom of a table.

SYNTAX:
fxRemoveTopBottomNull (Table, ColumnName, [ReturnOptions])

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Name of column to base transformation on.
- **[ReturnOptions]** (text) - Which null values should be removed.
  - "Top": Remove null values from the top.
  - "Bottom": Remove null values from the bottom.
  - [any other value] / null: Remove null values from Top and Bottom.

NOTES:
- (None)

EXAMPLE

The source Excel file includes an unknown number of rows before the data header row.

	Column1	Column2	Column3
1	Item Report	null	null
2	Date: 30/04/2023	null	null
3	Run By: Mark	null	null
4	null	null	null
5	Item Summary	null	null
6	Alpha	217	null
7	Bravo	302	null
8	Charlie	293	null
9	Delta	274	null
10	null	null	null
11	Item	Date	Value
12	Alpha	31/01/2023	113
13	Bravo	31/01/2023	64
14	Charlie	31/01/2023	104

Unknown number of rows before header

Data header row

Load the data into Power Query. Use the **fxRemoveTopBottomNull** function to remove all null rows at the top of the data.

```
= fxRemoveTopBottomNull(Data_Sheet,"Column3","Top")
```

- **Data_Sheet**: Name of previous step
- **"Column3"**: Column name in Power Query use as the basis for removing null values
- **"Top"**: Remove null values form the top

Result: Nulls removed ready for head to be promoted

	Column1	Column2	Column3
1	Item	Date	Value
2	Alpha	31/01/2023	113
3	Bravo	31/01/2023	64
4	Charlie	31/01/2023	104

fxRemoveTopRowsUntilValue

PURPOSE:
Removes an unknown number of rows from the top until search term found.

SYNTAX:
fxRemoveTopRowsUntilValue (Table, SearchColumnName, SearchValue, [PartialMatch], [InstanceNumber])

- **Table** (table) - Table or step to perform the transformation on.
- **SearchColumnName** (text) - Name of column to search.
- **SearchValue** (any) - Value to search for.
- **[PartialMatch]** (logical) - Determines if only partial match required (includes ignoring case).
  - true: Partial match permitted.
  - false / null: Exact match only.
- **[InstanceNumber]** (number): Where multiple matches exist, which instance to use as the basis for removing rows.

NOTES:
- (None)

EXAMPLE

The source Excel file includes an unknown number of rows before the data header row. We want to remove all rows above the third instance of the word **Item**.

	Column1	Column2	Column3
1	Item Report	null	null
2	Date: 30/04/2023	null	null
3	Run By: Mark	null	null
4	null	null	null
2	Item Summary	null	null
6	Alpha	217	null
7	Bravo	302	null
8	Charlie	293	null
9	Delta	274	null
10	null	null	null
3	Item	Date	Value
12	Alpha	31/01/2023	113
13	Bravo	31/01/2023	64
14	Charlie	31/01/2023	104

Remove these rows

3rd Instance of word Item

Load the data into Power Query. Use the **fxRemoveTopRowsUntilValue** function to remove the rows.

```
= fxRemoveTopRowsUntilValue(Data_Sheet,"Column1","Item",true,3)
```

- **Data_Sheet**: Name of previous step
- **"Column1"**: Name of column to search
- **"Item"**: Text to find
- **true**: Partial match permitted
- **3**: Find the 3rd instance of the search text

Result

	Column1	Column2	Column3
1	Item	Date	Value
2	Alpha	31/01/2023	113
3	Bravo	31/01/2023	64
4	Charlie	31/01/2023	104
5	Delta	31/01/2023	96

fxRepeatValueToNull

PURPOSE:
Changes repeat values in a column to null.

SYNTAX:
fxRepeatValueToNull (Table, ColumnNamesList)

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnNamesList** (list of text) - List of column names containing the repeat values to change to null.

NOTES:
- (None)

EXAMPLE

The Table below shows repeat values in the **Year** and **Quarter** columns.

	Year	Quarter	Month	Value
1	2023	Q3	Jul	92
2	2023	Q3	Aug	98
3	2023	Q3	Sep	61
4	2023	Q4	Oct	66
5	2023	Q4	Nov	62
6	2023	Q4	Dec	62
7	2024	Q1	Jan	74
8	2024	Q1	Feb	55
9	2024	Q1	Mar	86
10	2024	Q2	Apr	69
11	2024	Q2	May	96
12	2024	Q2	Jun	69

Use the **fxRepeatValueToNull** to convert repeated values to null.

```
= fxRepeatValueToNull(#"Changed Type", {"Year","Quarter"})
```

- **#"Changed Type"**: Name of previous step
- **{"Year","Quarter"}**: Columns to convert repeat values to null

Result

	Year	Quarter	Month	Value
1	2023	Q3	Jul	92
2	null	null	Aug	98
3	null	null	Sep	61
4	null	Q4	Oct	66
5	null	null	Nov	62
6	null	null	Dec	62
7	2024	Q1	Jan	74
8	null	null	Feb	55
9	null	null	Mar	86
10	null	Q2	Apr	69
11	null	null	May	96
12	null	null	Jun	69

fxUnstackOnInterval

PURPOSE:
Unstacks a column of data into separate columns based on a repeating numerical interval.

SYNTAX:
fxUnstackOnInterval (Table, ColumnName, Interval)

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Name of column containing the stacked data.
- **Interval** (number) - Length of repeating interval.

NOTES:
- (None)

EXAMPLE

The source Excel file includes address data in a single column. The addresses start every 4 rows.

	Column1
1	David Wilson
2	37 Jackson Drive
3	Newbury
4	NE5 TRL
5	Sandy Smith
6	12 Holsten Road
7	Heathdale
8	HE53 2RL
9	Douglas Wicks
10	43 Browning Way
11	Whittlesborough
12	WH7 3TL

Record #1
Record #2
Record #3

Use the **fxUnstackOnInterval** function to convert the data into 4 columns.

```
= fxUnstackOnInterval(#"Removed Top Rows","Column1",4)
```

- **#"Removed Top Rows"**: Name of previous step
- **"Column1"**: The Column containing the stacked data
- **4**: The rows in the repeating interval

Result:

	Column1	Column2	Column3	Column4
1	David Wilson	37 Jackson Drive	Newbury	NE5 TRL
2	Sandy Smith	12 Holsten Road	Heathdale	HE53 2RL
3	Douglas Wicks	43 Browning Way	Whittlesborough	WH7 3TL

fxUnstackOnValue

PURPOSE:
Unstacks a column of data into separate columns based on the occurrence of a text string.

SYNTAX:
fxUnstackOnValue (Table, ColumnName, SearchValue, [IgnoreCase])

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Name of column containing the stacked data.
- **SearchValue** (text) - Text value to split on.
- **[IgnoreCase]** (logical) - Should case be ignored when matching the search value.
    - true: ignore case for the search.
    - false / null: apply case for search.

NOTES:
- Split occurs on Text values. To split on null or errors, replace null or errors with other values prior to unstacking.

EXAMPLE

The source Excel file includes company information (Year, Company Number, Company Name, Turnover, Number Of Employees). The Number of Employees has been excluded from 2019. The number rows for reach record is not consistent. Each record starts with the word **Year:**.

	Column1
1	Year: 2019
2	0126745
3	Tillings Bridge Limited
4	647,000
5	Year: 2020
6	0126745
7	Tillings Bridge Limited
8	5,647,000
9	15
10	Year: 2021
11	0126745
12	Tillings Bridge Limited
13	8,148,000
14	15

Year: - indicates start of a new record

Use the **fxUnstackOnValue** function to convert the data into columns.

```
= fxUnstackOnValue(#"Removed Top Rows","Column1","Year:",true)
```

- **#"Removed Top Rows"**: Name of previous step
- **"Column1"**: The Column containing the stacked data
- **"Year:"**: Start new row at each instance of **Year:**
- **true**: Ignore case applied. **Year:** and **year:** would be treated the same (not relevant in this example).

Result:

	Column1	Column2	Column3	Column4	Column5
1	Year: 2019	0126745	Tillings Bridge Limited	647,000	*null*
2	Year: 2020	0126745	Tillings Bridge Limited	5,647,000	15
3	Year: 2021	0126745	Tillings Bridge Limited	8,148,000	15

Text Functions

fxMultiFindReplace

PURPOSE:
Find and replace values based on a list.

SYNTAX:
fxMultiFindReplace (Table, ColumnNamesList, FindList, ReplaceList, PartialMatch)

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnNamesList** (list of text) - List of column names to perform the find and replace action on.
- **FindList** (list of text) - List of text to find (e.g. {"A","B","C"}).
- **ReplaceList** (list of text) - List of text to replace (e.g. {"D","E","F"}).
- **PartialMatch** (logical) - Can the find and replace be a partial match.
    - true: Find and replace is performed on a partial text string.
    - false: Find and replace is performed on entire cell values only.

NOTES:
- The find and replace values must be in corresponding locations in each list.
- The transformation is case sensitive.

EXAMPLE

In the **Item** and **Region** columns, replace entire cell values of the **Alpha** with **Echo**, **South** with **Central** and **a** with **z.**

	Item	Region	Value
1	Alpha	North	34
2	Bravo	South	56
3	Charlie	East	78
4	Delta	West	23
5	Alpha	North	74
6	Bravo	South	23
7	Charlie	East	63
8	Delta	West	51

Use **fxMultiFindReplace** to find and replace the values.

```
= fxMultiFindReplace(#"Changed Type",{"Item","Region"},{"Alpha","South","a"},{"Echo","Central","z"},false)
```

- **#"Changed Type"**: Name of previous step
- **{"Item","Region"}**: Name of columns to apply the find and replace to
- **{"Alpha","South","a"}**:List of text to find
- **{"Echo","Central","z"}**: List of text to replace
- **false:** Partial match is not allowed, match full words only

Result:

	Item	Region	Value
1	Echo	North	34
2	Bravo	Central	56
3	Charlie	East	78
4	Delta	West	23
5	Echo	North	74
6	Bravo	Central	23
7	Charlie	East	63
8	Delta	West	51

To replace every instance of **a** with **z** change the **PartialMatch** parameter to **true**.

fxTextRemoveSpaces

PURPOSE:
Removes excess spaces from text values.

SYNTAX:
fxTextRemoveSpaces (Table, [ColumnNamesList])

- **Table** (table) - Table or step to perform the transformation on.
- **[ColumnNamesList]** (list of text) – List of column names on which to remove spaces.

NOTES:
- (None)

EXAMPLE

The source Table includes a text with leading, mid, trailing and non-breaking spaces.

	A	B	C
1	Item	Text	Value
2	Alpha	Text with leading spaces	1
3	Bravo	Text with mid spaces	2
4	Charlie	Text with trailing spaces	3
5	Delta	Text with non-breaking space character	4

Power Query's Trim transformation only removes leading and trailing spaces; this will not work in this scenario.

Load the data into Power Query and promote the headers. Use **fxTextRemoveSpaces** to clean the excess spaces.

```
= fxTextRemoveSpaces(#"Changed Type",{"Text"})
```

- **#"Changed Type"**: Name of previous step
- **{"Text"}**: List of column names containing the text to remove spaces. This example has a single column.

Result:

	Item	Text	Value
1	Alpha	Text with leading spaces	1
2	Bravo	Text with mid spaces	2
3	Charlie	Text with trailing spaces	3
4	Delta	Text with non-breaking space character	4

To apply the multiple columns, include the column names in the list (e.g. **{"Text", "Another Text Column", "More Text Columns"}**)

WARNING: Power Query may not display the excess spaces in the preview window. But they are there and will display in Excel.

	Item	Text	Value
1	Alpha	Text with leading spaces	1
2	Bravo	Text with mid spaces	2
3	Charlie	Text with trailing spaces	3
4	Delta	Text with non-breaking space character	

Power Query preview is not displaying the leading, mid or trailing spaces. But they are definitely there!

	A	B	C	D
1	Item	Text	Value	
2	Alpha	Text with leading spaces	1	
3	Bravo	Text with mid spaces	2	
4	Charlie	Text with trailing spaces	3	
5	Delta	Text with non-breaking space character	4	
6				

Calculation Functions

fxGroupRunningTotal

PURPOSE:
Adds a running total column where the total resets at the start of each group.

SYNTAX:
fxGroupRunningTotal (Table, ColumnName, NewColumnName , GroupByColumnNamesList , [RetainOrder])

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Column name to perform the calculation on.
- **NewColumnName** (text) - Name of the new running total column.
- **GroupByColumnNamesList** (list of text) - List of the column names to group by.
- **[RetainOrder]** (logical) - Should data be presented in the original order.
    - true: values are returned to their original order.
    - false / null: values ordered within each group.

NOTES:
- (None)

EXAMPLE

The source file includes a Table of data.

	Item	Region	Value
1	Alpha	North	62
2	Bravo	South	109
3	Charlie	East	76
4	Delta	West	51
5	Alpha	Central	95
6	Bravo	North	83
7	Charlie	South	107

Use the **fxGroupRunningTotal** function to add a running total calculation based on the Value column grouped by Region.

```
= fxGroupRunningTotal(#"Changed Type","Value","Running Total",{"Region"},false)
```

- **#"Changed Type"**: Name of previous step
- **"Value"**: Name of the column to create the running total for
- **"Running Total"**: The name of the new running total column.
- **"Region"**: Name of the column to group by
- **false**: Do not return records to their original order

Result:

	Item	Region	Value	Running Total
1	Alpha	North	62	62
2	Bravo	North	83	145
3	Charlie	North	92	237
4	Alpha	North	89	326
5	Bravo	North	87	413
6	Charlie	North	65	478
7	Delta	North	91	569
8	Bravo	South	109	109
9	Charlie	South	107	216
10	Delta	South	116	332
11	Bravo	South	56	388
12	Charlie	South	96	484
13	Delta	South	71	555

North Running Total

South Running Total

fxRunningTotal

PURPOSE:
Adds a running total column.

SYNTAX:
fxRunningTotal (Table, ColumnName , NewColumnName)

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Column name to perform the calculation on.
- **NewColumnName** (text) - Name of the new running total column.

NOTES:
- (None)

EXAMPLE

The source file includes a Table of data.

	Item	Region	Value
1	Alpha	North	62
2	Bravo	South	109
3	Charlie	East	76
4	Delta	West	51
5	Alpha	Central	95
6	Bravo	North	83
7	Charlie	South	107
8	Delta	East	92
9	Alpha	West	74
10	Bravo	Central	58
11	Charlie	North	92
12	Delta	South	116

Use the **fxRunningTotal** function to add a running total calculation based on the Value column.

```
= fxRunningTotal(#"Changed Type","Value","Running Total")
```

- **#"Changed Type"**: Name of previous step
- **"Value"**: Name of the column to create the running total for
- **"Running Total"**: The name of the new running total column.

Result:

	Item	Region	Value	Running Total
1	Alpha	North	62	62
2	Bravo	South	109	171
3	Charlie	East	76	247
4	Delta	West	51	298
5	Alpha	Central	95	393
6	Bravo	North	83	476
7	Charlie	South	107	583
8	Delta	East	92	675
9	Alpha	West	74	749
10	Bravo	Central	58	807
11	Charlie	North	92	899
12	Delta	South	116	1015

fxPreviousRow

PURPOSE:
Returns or calculates a value based on a previous or subsequent row in the table.

SYNTAX:
fxPreviousRow (Table, ColumnName, NewColumnName, RowOffset, [ReturnOptions])

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Name of column containing values.
- **NewColumnName** (text) - The name for the new column.
- **RowOffset** (number) - A number representing the rows to offset by.
- **[ReturnOptions]** (text) - Determine the type of value returned.
    - "Value" / null: Returns the previous value.
    - "Variance": Returns the numeric variance.
    - "Percentage": Returns the percentage variance.

NOTES:
- RowOffset can include positive or negative numbers to get rows after or before.
- All missing values are shown as null.

EXAMPLE

The source data includes a Value column.

	Date	Value
1	31/01/2023	56
2	28/02/2023	45
3	31/03/2023	89
4	30/04/2023	124
5	31/05/2023	52
6	30/06/2023	63
7	31/07/2023	78

Use the **fxPreviousRow** function to add a column with % variance to the prior row

```
= fxPreviousRow(#"Changed Type","Value","Variance",1,"Percentage")
```

- **#"Changed Type"**: Name of previous step
- **"Value"**: Name of the column to retrieve the previous row from
- **"3 Months Prior"**: The new column name
- **1:** Get the value from 1 row above
- **"Percentage"**: Returns the percentage variance

Result:

	Date	Value	% Variance
1	31/01/2023	56	null
2	28/02/2023	45	-19.64%
3	31/03/2023	89	97.78%
4	30/04/2023	124	39.33%
5	31/05/2023	52	-58.06%
6	30/06/2023	63	21.15%
7	31/07/2023	78	23.81%

Lookup Functions

fxLookupApproximateMatch

PURPOSE:
Returns an approximate match lookup from another table.

SYNTAX:
fxLookupApproximateMatch (Table, ValueColumnName, LookupTable , LookupValueColumnName, LookupReturnColumnName, NewColumnName, [LessThanOrMoreThan])

- **Table** (table) - Table or step to perform the transformation on.
- **ValueColumnName** (text) - Name of the column containing the lookup value.
- **LookupTable** (table) - Table to lookup the value from.
- **LookupValueColumnName** (text) - Name of the column to lookup.
- **LookupReturnColumnName** (text) - Name of the column to return value from.
- **NewColumnName** (text) - The name of the column to be added.
- **[LessThanOrMoreThan]** (text) - Should the return value *be less than or equal* to, or *more than or equal to* the lookup value.
  - "LessThan" / [any other value]: Return values less than or equal to the value.
  - "MoreThan": Return values more than or equal to the value.

NOTES:
- (None)

EXAMPLE

The source file includes a two tables; **Data** and **Discount**.

	Name	Value
1	Alpha	8512
2	Bravo	123487
3	Charlie	15676
4	Delta	75432

Data

	Threshold	Discount Rate
1	10000	10.00%
2	50000	25.00%
3	100000	50.00%

Discount

Use the **fxLookupApproximateMatch** function to add the **Discount Rate** column into the **Data** table. **Values** over the **Threshold** receive the discount rate.

```
= fxLookupApproximateMatch(#"Changed Type","Value",Discount,"Threshold","Discount Rate","Discount","LessThan")
```

- **#"Changed Type"**: Name of previous step
- **"Value"**: Name of the column to use for the lookup value
- **Discount**: The name of table to lookup from
- **"Threshold"**: The column to lookup in the lookup table
- **"Discount Rate"**: The column to return the value from in the lookup table
- **"Discount"**: The new column name
- **"LessThan"**: Return the value less than or equal to the lookup value.

Result:

	Name	Value	Discount
1	Alpha	8512	null
2	Bravo	123487	50.00%
3	Charlie	15676	10.00%
4	Delta	75432	25.00%

fxLookupExactMatch

PURPOSE:
Returns an exact match lookup from another table.

SYNTAX:
fxLookupExactMatch (Table, ValueColumnNameList, LookupTable , LookupValueColumnNameList, LookupReturnColumnName, NewColumnName, [ReturnOptions])

- **Table** (table) - Table or step to perform the transformation on.
- **ValueColumnNameList** (list of text) - Name of the columns containing the lookup value as a list (e.g. {"Col1","Col2","Col3"} for multicolumn, or {"Col1"} for a single column).
- **LookupTable** (table) - Table to lookup the value from.
- **LookupValueColumnNameList** (list of text) - Name of columns to lookup (e.g. {"Col1","Col2","Col3"} for multicolumn, or {"Col1"} for a single column).
- **LookupReturnColumnName** (text) - Name of the column to return value from.
- **NewColumnName** (text) - Name of the column to be added.
- **[ReturnOptions]** (text) - Which item(s) should the lookup return.
    - "First": Return the first item.
    - "Last": Return the last item.
    - [any other value] / null: Return all items.

NOTES:
- **ValueColumnNameList** and **LookupValueColumnNameList** must contain the same number of items in the same order.

EXAMPLE

The source file includes two tables; **Data** and **Category**

	Name	Size	Value
1	Alpha	Small	8512
2	Bravo	Small	123487
3	Charlie	Small	15676
4	Delta	Small	75432
5	Alpha	Large	76231
6	Bravo	Large	19776
7	Charlie	Large	6534
8	Delta	Large	165984

Data

	Name	Size	Category
1	Alpha	Small	Standard
2	Bravo	Small	Premium
3	Charlie	Small	Standard
4	Delta	Small	Premium
5	Alpha	Large	Premium
6	Bravo	Large	Premium
7	Charlie	Large	Standard
8	Delta	Large	Premium

Category

Use **fxLookupExactMatch** to lookup values from another table. Example, using the values in the Name and Size columns of the Data table, lookup the Name and Size columns in the Category table and return the value found in Category column.

```
= fxLookupExactMatch(#"Changed Type",{"Name","Size"},Category,{"Name","Size"},"Category","Category","First")
```

- **#"Changed Type"**: Name of previous step
- **{"Name","Size"}**: List of column names to use for the lookup value
- **Category**: The name of table to lookup from.
- **{"Name","Size"}**: List of column names to lookup
- **"Category"**: The column to return the value from
- **"Category"**: The new column name
- **"First"**: Return the first matching item

	Name	Size	Value	Category
1	Alpha	Small	8512	Standard
2	Bravo	Small	123487	Premium
3	Charlie	Small	15676	Standard
4	Delta	Small	75432	Premium
5	Alpha	Large	76231	Premium
6	Bravo	Large	19776	Premium

Data Type Functions

fxAutoDetectDataType

PURPOSE:
Auto applies data types based on the data in a column.

SYNTAX:
fxAutoDetectDataType (Table, [Threshold], [SampleSize])

- **Table** (table) - Table or step to perform the transformation on.
- **[Threshold]** (number) - Declare the % of valid items before applying the data type (e.g. 0.75 - 75% of the data needs to be of a specific data type otherwise treated as 'any').
  - If omitted, default value is 1 (e.g., 100%).
- **[SampleSize]** (number) - Declare how many records to include in the sample.
  - If omitted, default value is 10.

NOTES:
- On large data sets, this transformation can be slow, so recommended to always run on a sample.
- Where SampleSize exceeds the number of rows, the full data set is used.
- Using a Threshold < 0 or > 1 results in all columns being 'any' data type.
- DateTime data types are converted to Date where the time value is 00:00:00.
- Detects Whole Numbers, Decimal Numbers, Dates, DateTimes, Logical and Text. Others treated as 'any' or as 'text'.

EXAMPLE

The source data includes columns with various data types.

	Text	Whole Number	Decimal Number	Date	Date Time	Logical
1	Alpha	81	103.8704749	10/09/2023 00:00:00	02/10/2023 20:53:29	TRUE
2	Bravo	94	126.8124311	23/09/2023 00:00:00	25/10/2023 19:29:54	FALSE
3	Charlie	81	91.35351897	10/09/2023 00:00:00	20/09/2023 08:29:04	TRUE
4	Delta	88	56.35144579	17/09/2023 00:00:00	16/08/2023 08:26:05	FALSE
5	Echo	105	56.65256481	04/10/2023 00:00:00	16/08/2023 15:39:42	TRUE
6	Foxtrot	83	99.40630174	12/09/2023 00:00:00	28/09/2023 09:45:04	FALSE
7	Golf	70	53.25675526	30/08/2023 00:00:00	13/08/2023 06:09:44	TRUE
8	Hotel	74	118.4011121	03/09/2023 00:00:00	17/10/2023 09:37:36	FALSE
9	India	50	55.64688647	10/08/2023 00:00:00	15/08/2023 15:31:31	TRUE

Detecting data types hardcodes the column names into the M code. If column names could change, using **fxAutoDetectDataType** avoids using the names explicitly

```
= fxAutoDetectDataType(#"Promoted Headers",0.95,10)
```

- **#"Promoted Headers"**: Name of previous step
- **0.95**: Data Type is applied if 95% of the sample is of a specific type
- **10**: Data Type is determined based on the first 10 rows

Result:

	Text	Whole Number	Decimal Number	Date	Date Time	Logical
1	Alpha	81	103.8704749	10/09/2023	02/10/2023 20:53:29	TRUE
2	Bravo	94	126.8124311	23/09/2023	25/10/2023 19:29:54	FALSE
3	Charlie	81	91.35351897	10/09/2023	20/09/2023 08:29:04	TRUE
4	Delta	88	56.35144579	17/09/2023	16/08/2023 08:26:05	FALSE
5	Echo	105	56.65256481	04/10/2023	16/08/2023 15:39:42	TRUE
6	Foxtrot	83	99.40630174	12/09/2023	28/09/2023 09:45:04	FALSE
7	Golf	70	53.25675526	30/08/2023	13/08/2023 06:09:44	TRUE
8	Hotel	74	118.4011121	03/09/2023	17/10/2023 09:37:36	FALSE
9	India	50	55.64688647	10/08/2023	15/08/2023 15:31:31	TRUE

Ranking & Order Functions

fxRowNumberByGroup

PURPOSE:
Adds a row number for each item in a group.

SYNTAX:
fxRowNumberByGroup (Table, GroupByColumnNamesList, NewColumnName, [RetainOrder])

- **Table** (table) - Table or step to perform the transformation on.
- **GroupByColumnNamesList** (list of text) - List of the column names to group by.
- **NewColumnName** (text) - Name of the column to add.
- **[RetainOrder]** (logical) - Should data be presented in the original order.
    - true: values are returned to their original order.
    - false / null: values ordered within each group.

NOTES:
- (none)

EXAMPLE

The source file includes the following Table.

	Item	Size	Value
1	Alpha	Small	93
2	Bravo	Medium	67
3	Charlie	Small	59
4	Alpha	Medium	96
5	Bravo	Small	95
6	Charlie	Medium	68
7	Alpha	Small	78
8	Bravo	Medium	90

Use **fxRowNumberByGroup** to add a row number for each item in a group.

```
= fxRowNumberByGroup(#"Changed Type",{"Item"},"Row Number",false)
```

- **#"Changed Type"**: Name of previous step
- **{"Item"}**: The list of columns to group by. Include more column names to increase granularity of groups e.g. **{"Item", "Size"}**
- **"Row Number"**: Name of the column to add.
- **false**: Data presented in grouped order. Use **true** to present in original data order.

Result:

	Item	Size	Value	Row Number
1	Alpha	Small	93	1
2	Alpha	Medium	96	2
3	Alpha	Small	78	3
4	Alpha	Medium	100	4
5	Bravo	Medium	67	1
6	Bravo	Small	95	2
7	Bravo	Medium	90	3
8	Bravo	Small	55	4
9	Charlie	Small	59	1
10	Charlie	Medium	68	2
11	Charlie	Small	58	3
12	Charlie	Medium	58	4

Alpha Row Numbers
Bravo Row Numbers
Charlie Row Numbers

Date & Time Functions

fxCalendarTable

PURPOSE:
Creates a calendar table based on start date & end date, or start date & duration.

SYNTAX:
fxCalendarTable (StartDate, [EndDate], [DaysDuration])

- **StartDate** (date) - The first date in the calendar.
- **[EndDate]** (date) - The last date in the calendar.
- **[DaysDuration]** (number) - The number of days from the start date.

NOTES:
- If EndDate is provided, a calendar is created using StartDate & EndDate.
- If EndDate is null, a calendar is created using StartDate & DaysDuration.

EXAMPLE

Start with a Blank Query:
- In Excel, click **Data > Get Data > From Other Sources > Blank Query**
- In Power Query, click **Home > New Sources > Other Sources > Blank Query**

Use the **fxCalendarTable** function to generate a calendar table.

Scenario #1: Calendar table from 1 March 2022 to 31 March 2023

```
= fxCalendarTable(#date(2022,3,1),#date(2023,3,31))
```

- **#date(2022,3,1)**: The date for 1 March 2022
- **#date(2022,3,1)**: The date for 31 March 2023

Scenario #2: Calendar table from 1 March 2022 for 400 days

```
= fxCalendarTable(#date(2022,3,1),null,400)
```

- **#date(2022,3,1)**: The date for 1 March 2022
- **null**: End date not provided. Base the calendar table on duration
- **400**: number of days to include in the calendar table

Result:

	Date
1	01/03/2022
2	02/03/2022
3	03/03/2022
4	04/03/2022
5	05/03/2022
6	06/03/2022
7	07/03/2022
8	08/03/2022
9	09/03/2022

Alternative ways to generate dates:

- **Specific date:**
  #date(2023,3,31)
- **Specific date from text (local)**
  Date.From("31/01/2023")
  Example is dd/mm/yyyy, use your local format
- **Specific date from text (ISO8601):**
  Date.From("2023-01-31")
- **Today's date (local):**
  Date.From(DateTimeZone.LocalNow())
- **Today's date (UTC):**
  Date.From(DateTimeZone.UtcNow())
- **Earliest date in date column:**
  List.Min(QueryName[DateColumn])
- **Latest date in date column:**
  List.Max(QueryName[DateColumn])
- **Parameter Dates**:
  Get dates from a worksheet using *fxGetNameParameter* or *fxGetTableParameter*

fxCalendarTableNonStandard

PURPOSE:
Creates a calendar table from a table of non-calendar periods.

SYNTAX:
fxCalendarTableNonStandard (Table, PeriodEndDateColumnName, NewDateColumnName)

- **Table** (table) - Table containing data about the period ends.
- **PeriodEndDateColumnName** (text) - Name of column containing the period end date.
- **NewDateColumnName** (text) - Name to give to the new date column.

NOTES:
- (None)

EXAMPLE

Start with a manual table of period end dates. In this example: 4-4-5 calendar with closest Sunday to 31 March as the year end.

	Period End Date	Period	Year
1	02/04/2023	12	2023
2	30/04/2023	1	2024
3	28/05/2023	2	2024
4	02/07/2023	3	2024
5	30/07/2023	4	2024
6	27/08/2023	5	2024

Use the **fxCalendarTableNonStandard** function to generate a calendar table.

```
= fxCalendarTableNonStandard (#"Changed Type","Period End Date", "Date")
```

- **#"Changed Type"**: Name of previous step
- **"Period End Date"**: Column containing the period end dates
- **"Date"**: Name for the new date column

Result:

	Date	Period End Date	Period	Year
1	02/04/2023	02/04/2023	12	2023
2	03/04/2023	30/04/2023	1	2024
3	04/04/2023	30/04/2023	1	2024
4	05/04/2023	30/04/2023	1	2024
5	06/04/2023	30/04/2023	1	2024
6	07/04/2023	30/04/2023	1	2024
7	08/04/2023	30/04/2023	1	2024
8	09/04/2023	30/04/2023	1	2024
9	10/04/2023	30/04/2023	1	2024
10	11/04/2023	30/04/2023	1	2024
11	12/04/2023	30/04/2023	1	2024
12	13/04/2023	30/04/2023	1	2024

Date: Full calendar table

Period End Date, Period and Year expanded for each date

fxFinancialPeriod

PURPOSE:
Adds a Financial Month, Year, or Quarter column.

SYNTAX:
fxFinancialPeriod (Table, DateColumnName, FinancialYearEndMonth, NewColumnName, TimePeriod)

- **Table** (table) - Table or step to perform the transformation on.
- **DateColumnName** (text) - Name of the column containing dates.
- **FinancialYearEndMonth** (number) - Month number of the financial year end.
- **NewColumnName** (text) - Name of the new column.
- **TimePeriod** (text) - Set return value as Financial Year or Financial Month.
    - "Month": Returns the Financial Month.
    - "Quarter": Return the Financial Quarter.
    - "Year": Return the Financial Year.

NOTES:
- Assumes financial periods end on the last day of each month.

EXAMPLE

The source data includes a date column.

	Date
1	31/01/2023
2	28/02/2023
3	31/03/2023
4	30/04/2023
5	31/05/2023
6	30/06/2023
7	31/07/2023

The Financial Year ends in March. Use **fxFinancialPeriod** to add the financial year quarter, or month. Example below adds a month.

```
= fxFinancialPeriod(#"Changed Type","Date",3,"Fin Month","Month")
```

- **#"Changed Type"**: Name of previous step
- **"Date"**: Name of the date column
- **3**: The Financial Year ends in March; March is the 3rd month of the calendar year.
- **"Fin Month":** The new column name
- **"Month"**: Adds a financial **Month** column.

Result:

	Date	Fin Month
1	31/01/2023	10
2	28/02/2023	11
3	31/03/2023	12
4	30/04/2023	1
5	31/05/2023	2
6	30/06/2023	3

fxTimeGroups

PURPOSE:
Groups time into Hour, Minute and Second time segments.

SYNTAX:
fxTimeGroups (Table, ColumnName, Units, UnitGroupSize, RoundType, NewColumnName)

- **Table** (table) - Table or step to perform the transformation on.
- **ColumnName** (text) - Name of the column containing the time or datetime.
- **Units** (text) - Text value of time units to group by.
  - "Hours": Hour time segments.
  - "Minutes": Minute time segments.
  - "Seconds" / [any other value] / null: Second time segments.
- **UnitGroupSize** (number) - The length of each unit group (e.g. When Units = "Minutes", 5 is 5 Minute segments).
- **RoundType** (text) - Text value of the how to round each value.
  - "Up": Round up.
  - "Nearest": Round to nearest.
  - "Down" / [any other value] / null: Round down.
- **NewColumnName** (text) - The name of the column to be added.

NOTES:
- (None)

EXAMPLE

Start with a column of times (or can be datetimes)

	Time
1	05:06:37
2	11:27:48
3	03:04:56
4	07:08:31
5	15:06:28
6	18:56:23
7	22:38:14
8	23:58:01

Use **fxTimeGroups** to group into 15-minute segments

```
= fxTimeGroups(#"Changed Type","Time","Minutes",15,"Down","Rounded Time")
```

- **"Changed Type":** Name of previous step
- **"Time"**: Name of column containing the times
- **"Minutes"**: Unit for creating the time groups
- **15**: Group into 15 minute segments
- **"Down"**: Rounds time down to
- **"Rounded Time"**: Name of the new column

Result:

	Time	Rounded Time
1	05:06:37	05:00:00
2	11:27:48	11:15:00
3	03:04:56	03:00:00
4	07:08:31	07:00:00
5	15:06:28	15:00:00
6	18:56:23	18:45:00
7	22:38:14	22:30:00
8	23:58:01	23:45:00

15-minute segments

fxTimeTable

PURPOSE:
Creates a time table based in hour, minute, or second intervals.

SYNTAX:
fxTimeTable ([Units])

- **[Units]** (text) - Text value of time units.
  - "Hours": Hour time segments.
  - "Minutes": Minute time segments.
  - "Seconds" / [any other value] / null: Second time segments.

NOTES:
- (None)

EXAMPLE

Start with a Blank Query:
- In Excel, click **Data > Get Data > From Other Sources > Blank Query**
- In Power Query, click **Home > New Sources > Other Sources > Blank Query**

Use **fxTimeTable** function to generate a time table.

Scenario #1: Time table in hours

```
= fxTimeTable("Hours")
```

- **"Hours"**: Unit for creating a time table in hours

Scenario #1 Result (Hours):

	Time
1	00:00:00
2	01:00:00
3	02:00:00
4	03:00:00
5	04:00:00
6	05:00:00
7	06:00:00
8	07:00:00
9	08:00:00
10	09:00:00
11	10:00:00
12	11:00:00
13	12:00:00
14	13:00:00
15	14:00:00
16	15:00:00
17	16:00:00
18	17:00:00
19	18:00:00
20	19:00:00
21	20:00:00
22	21:00:00
23	22:00:00
24	23:00:00

Scenario #2: Time table in seconds

```
= fxTimeTable("Minutes")
```

- **"Minutes"**: Unit for creating a time table in minutes

Scenario #2 Result (Minutes):

	Time
1	00:00:00
2	00:01:00
3	00:02:00
4	00:03:00
5	00:04:00
6	00:05:00
7	00:06:00
8	00:07:00
9	00:08:00
10	00:09:00
11	00:10:00
12	00:11:00
13	00:12:00
14	00:13:00
15	00:14:00
16	00:15:00
17	00:16:00
18	00:17:00
19	00:18:00
20	00:19:00
21	00:20:00
22	00:21:00
23	00:22:00
24	00:23:00

Scenario #3: Time table in seconds

```
= fxTimeTable()
```

- Defaults to seconds

Scenario #3 Result (Seconds):

	Time
1	00:00:00
2	00:00:01
3	00:00:02
4	00:00:03
5	00:00:04
6	00:00:05
7	00:00:06
8	00:00:07
9	00:00:08
10	00:00:09
11	00:00:10
12	00:00:11
13	00:00:12
14	00:00:13
15	00:00:14
16	00:00:15

Files & Folder Functions

fxCombineWorksheetsByName

PURPOSE:
Combine worksheets from workbooks in a folder using sheet or table names.

SYNTAX:
fxCombineWorksheetsByName (Table, ContentColumnName, ObjectName, PartialMatch, SheetOrTable, [PromoteHeaders], [AutoExpand])

- **Table** (table) - Table or step containing the files in the folder.
- **ContentColumnName** (text) - Name of column containing the workbook binaries.
- **ObjectName** (text) - The table or sheet name to combine.
- **PartialMatch** (logical) - Determines if only partial match required (includes ignoring case).
    - true: Partial match permitted.
    - false / null: Exact match only.
- **SheetOrTable** (text) - Determine if objects to combine are sheets or tables.
- **[PromoteHeaders]** (logical) - Should headers be promoted.
    - true: Promote headers.
    - false / null: Do not promote headers.
- **[AutoExpand]** (logical) - Should data should be expanded automatically (dynamically).
    - true: Expand data.
    - false / null: Do not expand data.

NOTES:
- (None)

EXAMPLE

Folder contains 3 workbooks. The workbooks contain 3 worksheets
- Example Excel Workbook #1 Data.xlsx [Data #1-1, Data #1-2, Data #1-3]
- Example Excel Workbook #2 Data.xlsx [Data #2-1, Data #2-2, Data #2-3]
- Example Excel Workbook #2 Data.xlsx [Data #3-1, Data #3-2, Data #3-3]

In Power Query the folder connection shows the workbooks

	Content	Name
1	Binary	Example Excel Workbook #1 Data.xlsx
2	Binary	Example Excel Workbook #2 Data.xlsx
3	Binary	Example Excel Workbook #3 Data.xlsx

Use **fxCombineWorksheetsByName** function to combine all worksheets with **-2** in the name

```
= fxCombineWorksheetsByName(#"Removed Other Columns","Content","-2",true,"Sheet",true,true)
```

- **#"Removed Other Columns"**: Name of previous step
- **"Content"**: Column with the workbook binaries
- **"-2":** Object name string to find
- **true**: Partial string match permitted
- **"Sheet"**: Combine worksheets
- **true**: Promote headers in each worksheet
- **true**: Auto expand the data

fxCombineWorksheetsByName (page 2)

Result:

	Content	Name	Excel Object Name	Item	Region	Value
1	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Alpha	North	112
2	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Bravo	South	85
3	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Charlie	East	59
4	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Delta	West	77
5	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Alpha	Central	143
6	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Bravo	North	114
7	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Charlie	South	139
8	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Delta	East	128
9	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Alpha	West	148
10	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Bravo	Central	133
11	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Charlie	North	122
12	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Delta	South	69
13	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Alpha	South	142
14	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Bravo	East	79
15	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Charlie	West	97
16	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Delta	Central	135
17	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Alpha	North	59
17	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Bravo	South	58
18	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Charlie	East	149
19	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Delta	West	138
20	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Alpha	Central	114
21	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Bravo	North	58
22	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Charlie	South	73
23	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Delta	East	75
24	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Alpha	West	69
25	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Bravo	Central	99
26	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Charlie	North	109
27	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Delta	South	100
28	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Alpha	East	113
29	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Bravo	West	92
30	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Charlie	Central	116
31	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Delta	North	114
32	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Alpha	South	98
33	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Bravo	East	89
34	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Charlie	West	83

Sheet: Data #1-2

Sheet: Data #2-2

Sheet: Data #3-2

fxCombineWorksheetsByPosition

PURPOSE:
Combine worksheets from workbooks in a folder using sheet or table position.

SYNTAX:
fxCombineWorksheetsByPosition (Table, ContentColumnName, Position, SheetOrTable, [PromoteHeaders], [AutoExpand])

- **Table** (table) - Table or step containing the files in the folder.
- **ContentColumnName** (text) - Name of column containing the workbook binaries.
- **Position** (number) - Number of the sheet or table to combine (zero-based).
- **SheetOrTable** (text) - Determine if objects to combine are sheets or tables.
- **[PromoteHeaders]** - (logical) - Should headers be promoted.
  - true: Promote headers.
  - false / null: Do not promote headers.
- **[AutoExpand]** (logical) - Should data should be expanded automatically (dynamically).
  - true: Expand data.
  - false / null: Do not expand data.

NOTES:
- (None)

EXAMPLE

Folder contains 3 workbooks. The workbooks contain 3 worksheets
- Example Excel Workbook #1 Data.xlsx [Data #1-1, Data #1-2, Data #1-3]
- Example Excel Workbook #2 Data.xlsx [Data #2-1, Data #2-2, Data #2-3]
- Example Excel Workbook #2 Data.xlsx [Data #3-1, Data #3-2, Data #3-3]

In Power Query the folder connection shows the workbooks

	Content	Name
1	Binary	Example Excel Workbook #1 Data.xlsx
2	Binary	Example Excel Workbook #2 Data.xlsx
3	Binary	Example Excel Workbook #3 Data.xlsx

Use **fxCombineWorksheetsByPosition** function to combine the 2nd sheet in each workbook

```
= fxCombineWorksheetsByPosition(#"Removed Other Columns","Content",1,"Sheet",true,true)
```

- **#"Removed Other Columns"**: Name of previous step
- **"Content"**: Column with the workbook binaries
- **1:** Combine 2nd sheet from each workbook
- **"Sheet"**: Combine worksheets
- **true**: Promote headers in each worksheet
- **true**: Auto expand the data

fxCombineWorksheetsByPosition (page 2)

Result:

	Content	Name	Excel Object Name	Item	Region	Value
1	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Alpha	North	112
2	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Bravo	South	85
3	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Charlie	East	59
4	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Delta	West	77
5	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Alpha	Central	143
6	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Bravo	North	114
7	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Charlie	South	139
8	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Delta	East	128
9	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Alpha	West	148
10	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Bravo	Central	133
11	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Charlie	North	122
12	Binary	Example Excel Workbook #1 Data.xlsx	Data #1-2	Delta	South	69
13	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Alpha	South	142
14	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Bravo	East	79
15	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Charlie	West	97
16	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Delta	Central	135
17	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Alpha	North	59
17	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Bravo	South	58
18	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Charlie	East	149
19	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Delta	West	138
20	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Alpha	Central	114
21	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Bravo	North	58
22	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Charlie	South	73
23	Binary	Example Excel Workbook #2 Data.xlsx	Data #2-2	Delta	East	75
24	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Alpha	West	69
25	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Bravo	Central	99
26	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Charlie	North	109
27	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Delta	South	100
28	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Alpha	East	113
29	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Bravo	West	92
30	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Charlie	Central	116
31	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Delta	North	114
32	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Alpha	South	98
33	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Bravo	East	89
34	Binary	Example Excel Workbook #3 Data.xlsx	Data #3-2	Charlie	West	83

Sheet: Data #1-2

Sheet: Data #2-2

Sheet: Data #3-2

fxExcelDataFromList

PURPOSE:
Uses a table with file path and data columns to get data from a list of Excel Workbooks.

SYNTAX:
fxExcelDataFromList (Table, FilePathColumnName, DataColumnName, [PromoteHeaders], [AutoExpand])

- **Table** (table) - Table or step to perform the transformation on.
- **FilePathColumnName** (text) - The column name which contain the file paths.
- **DataColumnName** (text) - The name of the data object in the workbook.
- **[PromoteHeaders]** (logical) - Should headers be promoted prior to expanding.
    - true: Promote headers.
    - false / null: Do not promote headers.
- **[AutoExpand]** (logical) - Should data should be expanded automatically (dynamically).
    - true: Expand data.
    - false / null: Do not expand data.

NOTES:
- (None)

EXAMPLE

A workbook has a table which includes the File Path and Sheet Names to combine.

File Path	Sheet Name
C:\Examples\Power Query\Example Excel Workbook #1 Data.xlsx	Data #1
C:\Examples\Power Query\Example Excel Workbook #2 Data.xlsx	Data #2
C:\Examples\Power Query\Example Excel Workbook #3 Data.xlsx	Data #3

Use **fxExcelDataFromList** to combine the data from different worksheets in different files.
Note: Best used where files not stored in a single folder, or where multiple sheets with different names exist in a workbook.

```
= fxExcelDataFromList(#"Changed Type","File Path","Sheet Name",true,false)
```

- **#"Changed Type"**: Name of previous step
- **"File Path"**: Column name containing the file path
- **"Sheet Name"**: Column name containing the sheet names to combine
- **true**: headers are promoted in each sheet
- **false**: columns are not expanded dynamically

Result: Files combined - data ready for expansion:

	File Path	Sheet Name	Excel Data Column
1	C:\Examples\Power Query\Example Excel Workbook #1 Data.xlsx	Data #1	Table
2	C:\Examples\Power Query\Example Excel Workbook #2 Data.xlsx	Data #2	Table
3	C:\Examples\Power Query\Example Excel Workbook #3 Data.xlsx	Data #3	Table

Alternative Result: Change **AutoExpand** to **true**:

```
= fxExcelDataFromList(#"Changed Type","File Path","Sheet Name",true,true)
```

	File Path	Sheet Name	Item	Region	Value
1	C:\Examples\Power Query\Example Excel Workbook #1 Data.xlsx	Data #1	Alpha	North	112
2	C:\Examples\Power Query\Example Excel Workbook #1 Data.xlsx	Data #1	Bravo	South	85
3	C:\Examples\Power Query\Example Excel Workbook #1 Data.xlsx	Data #1	Charlie	East	59
4	C:\Examples\Power Query\Example Excel Workbook #1 Data.xlsx	Data #1	Delta	West	77
5	C:\Examples\Power Query\Example Excel Workbook #1 Data.xlsx	Data #1	Alpha	Central	143
6	C:\Examples\Power Query\Example Excel Workbook #1 Data.xlsx	Data #1	Bravo	North	114

fxFilesInSharePointFolder

PURPOSE:
Gets all the files in a named SharePoint folder based on a folder URL.

SYNTAX:
fxFilesInSharePointFolder (FullURL, [IncludeSubFolders])

- **FullURL** (text) - The full URL of the SharePoint folder.
- **[IncludeSubFolders]** (logical) **-** Should the files in subfolders be included in the files returned.
    - true: Include subfolders.
    - false / null: Exclude subfolders.

NOTES:
- (None)

EXAMPLE

To connect to a SharePoint folder without navigating through the hierarchy use the **fxFilesInSharePointFolder** function.

```
= fxFilesInSharePointFolder("https://[redacted].sharepoint.com/sites/SharePointExample/Shared%20Documents/Example/",true)
```

- **"https://xxxxxxxxxx.sharepoint.com/sites/SharePointExample/Shared%20Documents/Example/"**: SharePoint URL for the folder.
- **true**: Sub folders are included.

NOTE:
There is no example file for this function. Apply the function to your SharePoint environment.

Nested Table Functions

fxTransformNestedTable

PURPOSE:
Performs table transformations on nested tables.

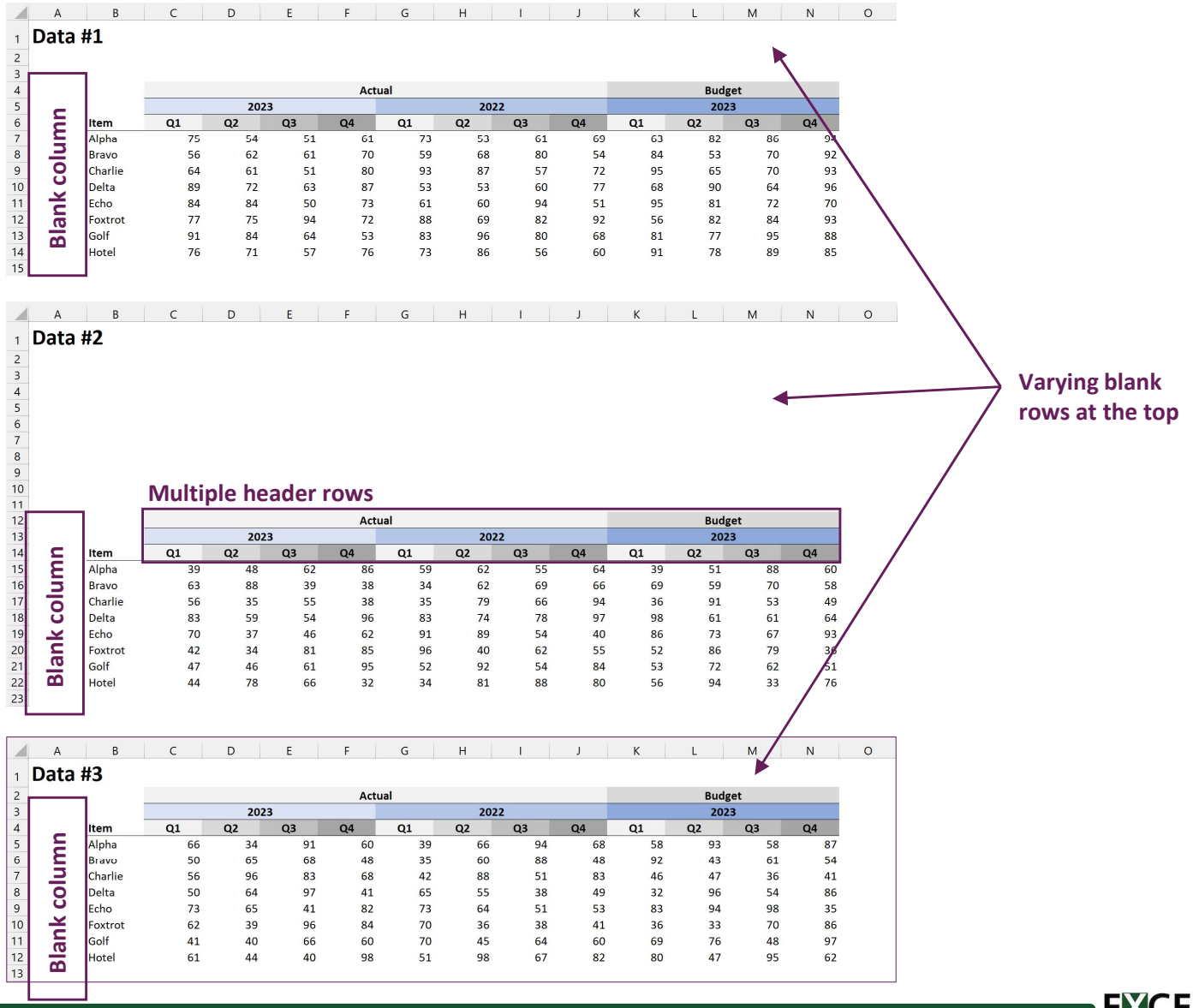
SYNTAX:
fxTransformNestedTable (Table, ColumnName, eachFunctionList)

- **Table** (table) - Table or step containing the nested tables.
- **ColumnName** (text) - Name of column containing the nested tables.
- **eachFunctionList** (List of nested functions) - List of functions to perform on the nested tables.
    - Every function must be preceded by the word **each.**
    - Refer to nested tables with an underscore (_).

NOTES:
- (None)

EXAMPLE

A workbook contains 3 sheets (Data #1, Data #2, Data #3). Each worksheet has different number of blank rows at the top before the data, multiple header rows and a blank column at the start.



fxTransformNestedTable (page 2)

In Power Query connect to the workbook.

	Name	Data	Item	Kind	Hidden
1	Data #1	Table	Data #1	Sheet	FALSE
2	Data #2	Table	Data #2	Sheet	FALSE
3	Data #3	Table	Data #3	Sheet	FALSE

Use the **fxTransformNestedTable** function to apply the following functions on each nested table:

- **fxRemoveTopBottomNull**
- **fxFlattenHeaderRows**
- **fxRemoveNullColumns**

```
= fxTransformNestedTable(Source,"Data",{
each fxRemoveTopBottomNull(_,"Column3","Top"),
each fxFlattenHeaderRows(_,3,"|","Right"),
each fxRemoveNullColumns(_)
})
```

NOTE
Each function is a separate item in a list; therefore, each function must be separated by a comma.

- **Source**: Name of previous step
- **"Data"**: Name of column containing the nested tables
- **{**
  each fxRemoveTopBottomNull(_,"Column3","Top"),
  each fxFlattenHeaderRows(_,3,"|","Right"),
  each fxRemoveNullColumns(_)
  }: Functions list applied to the nested tables.

Result:
The nested have been transformed into a consistent data structure. After expanding the **Data** column, the data looks like the following:

	Name	Item.1	Actual|2023|Q1	Actual|2023|Q2	Actual|2023|Q3	Actual|2023|Q4	Actual|2022|Q1	Actual|2022|Q2
1	Data #1	Alpha	75	54	51	61	73	51
2	Data #1	Bravo	56	62	61	70	59	62
3	Data #1	Charlie	64	61	51	80	93	87
4	Data #1	Delta	89	72	63	87	53	53
5	Data #1	Echo	84	84	50	73	61	60
6	Data #1	Foxtrot	77	75	94	72	88	69
7	Data #1	Golf	91	84	64	53	83	94
8	Data #1	Hotel	76	71	57	76	73	80
9	Data #2	Alpha	39	48	62	86	59	62
10	Data #2	Bravo	63	88	39	38	34	62
11	Data #2	Charlie	56	35	55	38	35	79
12	Data #2	Delta	83	59	54	96	83	74
13	Data #2	Echo	70	37	46	62	91	87
14	Data #2	Foxtrot	42	34	81	85	96	41
15	Data #2	Golf	47	46	61	95	52	94
16	Data #2	Hotel	44	78	66	32	34	96

NOTE: Using native Power Query table functions:

The **fxTransformNestedTable** function is also compatible with native Power Query functions. The example below applies the **Table.PromoteHeaders** and **Table.RemoveFirstN** functions.

```
= fxTransformNestedTable( Source,"Data", {
each Table.PromoteHeaders(_),
each Table.RemoveFirstN(_,1)
})
```