SumProduct

NEWSLETTER #105 - August 2021

www.sumproduct.com | www.sumproduct.com/thought

BIG news this month for both Excel and on a personal level for two of the SumProduct gang, as Microsoft introduce no less than SEVEN brand spanking new functions (including the 500th!), whilst two of the team (including yours truly) are renewed as Most Valuable Professionals (MVPs).

Then there are the regulars: we have another Beat the Boredom Challenge, Charts & Dashboards, Visual Basics, Power Pivot Principles, Power Query Pointers, Power BI updates, Keyboard Shortcuts and we also "G" up the A to Z of Excel Functions this month. Even the old Excel 4.0 (XLM) macros get a look-in this month – as I said, it's another **BIG** month!

As always, happy reading and remember: stay safe, stay happy, stay healthy.

Liam Bastick, Managing Director, SumProduct

MVP Renewal

SumProduct is pleased to announce that two of our Directors, **Liam Bastick** and **Tim Heng**, have been re-awarded Microsoft's Most Valuable Professional (MVP) award for Excel for 2021-22. It's a landmark this time for Liam as well, as he celebrates his 10th "official" award.

This award recognises exceptional technical community leaders from around the world who voluntarily share their high quality, real world expertise with others. Microsoft MVPs are a highly select group of experts representing technology's best and brightest who share a deep commitment to community and a willingness to help others.

Worldwide, there are over 100 million participants in technical communities; of these participants, there are c.4,000 active Microsoft MVPs. In Excel, we believe there are approximately 80 that have received this award.

Microsoft's MVP Award evaluates technical expertise and voluntary community contributions for the past year, considering the quality, quantity and level of impact of contributions. It's a difficult award to attain and just as difficult to retain.

At SumProduct, you can rely on our experience and willingness to help - simply drop us a line at contact@sumproduct.com should you need assistance.

Excel Hits 500

As mentioned elsewhere in this newsletter, this month sees the introduction of no less than seven new Excel functions. With these newcomers brought in, ladies and gentlemen, may I welcome to the fore, **Excel's 500th function** – as agreed by fellow Excel Most Valuable Professional (MVP) Bill Jelen and ourselves.

MAP is Excel's 500th function and you can find out more about it below.

Welcome aboard!

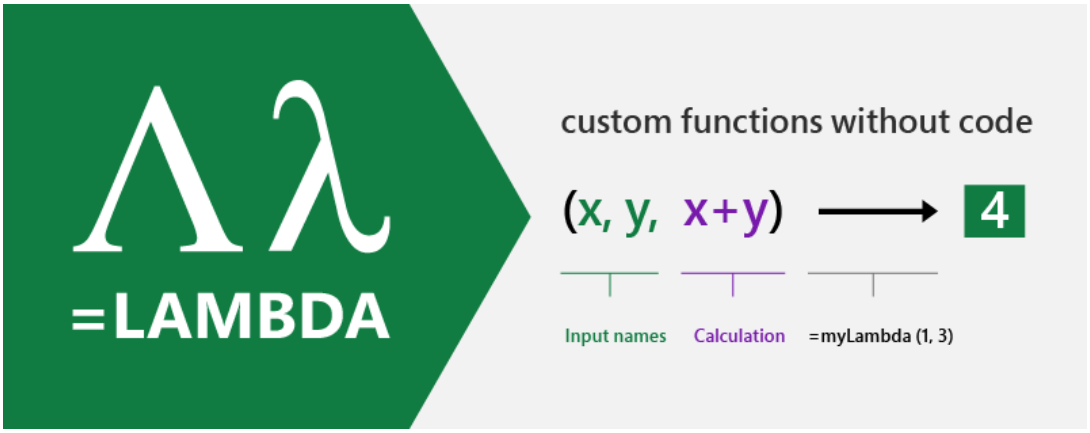
Seven New Excel Functions

It's busy times in the land of Excel 365 – if you want to live on the cutting edge. Microsoft has just announced the release of no less than seven new **LAMBDA**-associated functions (including a landmark 500th one – someone please bake Excel a cake) for the Beta variant, whilst also moving the recently released **LAMBDA** function a step closer to becoming Generally Available.

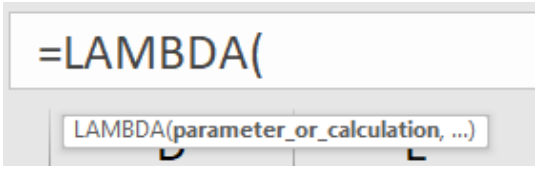
It's a lot to take in. Where do we start? Perhaps let's begin with revisiting **LAMBDA** itself.

LAMBDA Moves to Excel 365 Current Channel Preview

For those who have been hiding under a rock, let me start by saying **LAMBDA** rocks *(if this continues, this will be a very short article – Ed.).* Simply put, **LAMBDA** allows you to define your own custom functions using Excel's formula language. It's User Defined Functions without a PhD in VBA or JavaScript. Now moved to the Office 365 Current Channel Preview from Beta, **LAMBDA** allows you to define a custom function in Excel's very own formula language. Moreover, one function can call another (including itself), so there is no limit to the power you can deploy with a single function call.



As a reminder, the syntax of **LAMBDA** perhaps remains not the most informative:



That's, er, great. Perhaps a run-through might be best.

There are three key pieces of **=LAMBDA** to understand:

1. **LAMBDA** function components
2. Naming a lambda
3. Calling a lambda function.

1. LAMBDA function components

Let's take a simple example. Consider the following formula:

=LAMBDA(x, x + 1)

This is a very exciting formula, where we have **x** as the argument, which you may pass in when calling the **LAMBDA**, and **x+1** is the logic / operation to be performed.

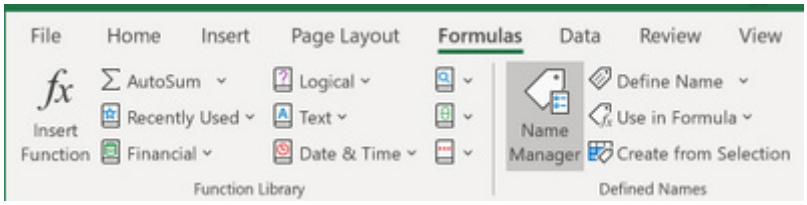
For example, if you were to call this lambda function and define x as equal to five (5), then Excel would calculate

5 + 1 = 6

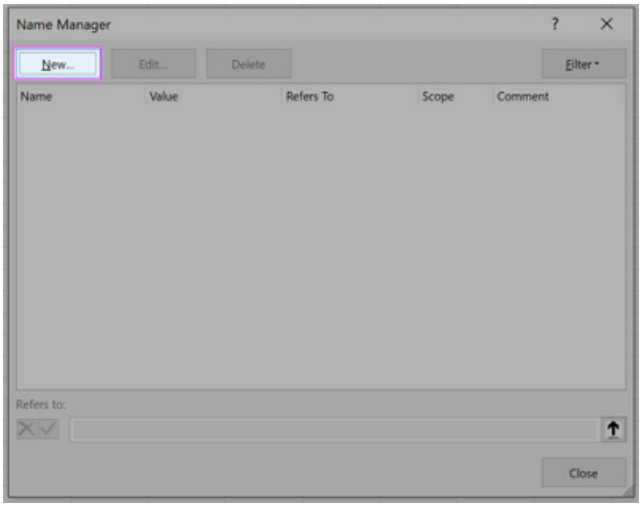
Except it wouldn't. If you tried this you would get *#CALC!* Oops. That's because it's not *quite* as simple as that. You need to name your little **LAMBDA**.

2. Naming a LAMBDA

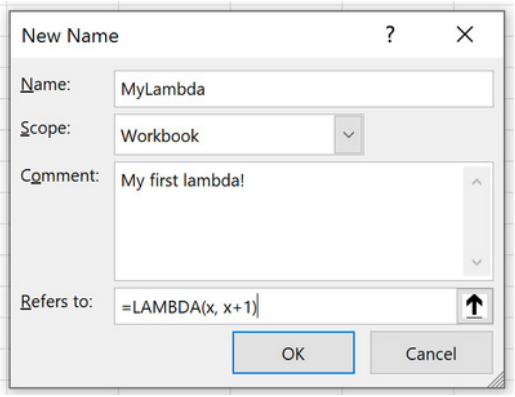
To give your **LAMBDA** a name so it can be re-used, you have to use the Name Manager (**CTRL + F3** / go to the Ribbon and then go to **Formulas -> Name Manager)**:



Once you open the 'Name Manager' you will see the following dialog:



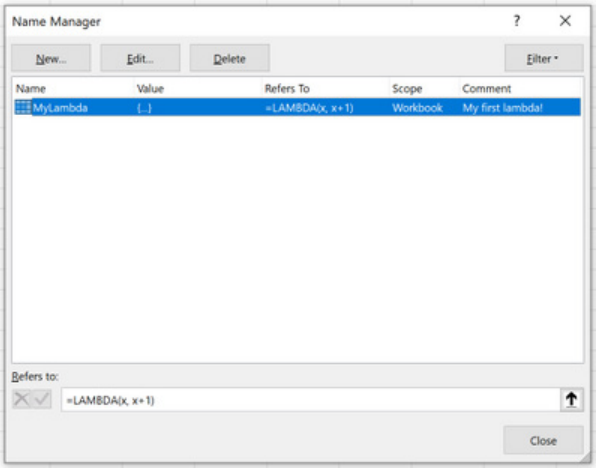
You then click on 'New' and fill out the related fields, *viz*.



To be clear:

- **Name:** the name of your function (this is where you name it!)
- **Comment:** a description and associated ToolTip, which will be shown when calling your function
- **Refers to:** your lambda function definition (this is where you put your formula – ***NOT*** in the Excel worksheet!).

Once completed, you may press 'OK' to store your lambda and you should see the definition returned in the resultant window.



3. Calling LAMBDA

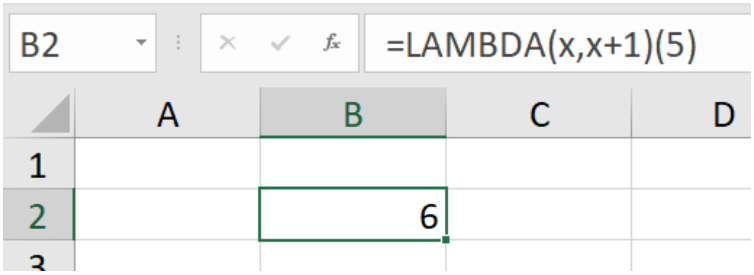
Now that you have done this, your first new lambda function may be called in just the same way as every other Excel function is cited, *e.g.*

=MYLAMBDA(5)

which would equal six (6) and not *#CALC!* as before.

You DON'T have to do it this way though if you don't want to. You may call a lambda without naming it. If we hadn't named this marvellous calculation, and simply authored it in the grid as we had first attempted, we could call it by simply typing:

=LAMBDA(x, x + 1)(5)

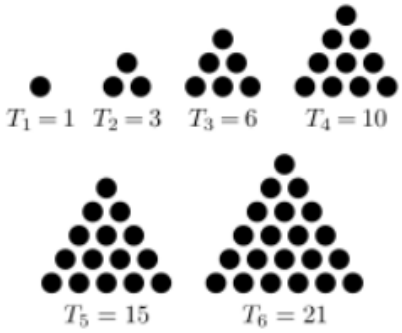


The sky's the limit. You are not restricted to just numbers and text. You can also use:

- **Dynamic arrays**: rather than passing a single value into a function, you can pass an array of values, and functions can also return arrays of values
- **Data Types:** the value stored in a cell is no longer just a string or a number. A single cell can contain a rich Data Type, with a large set of properties, as discussed previously.

Functions can take data types and arrays as arguments, and they can also return results as data types and arrays. The same is now true with the lambdas you build.

Indeed, formulaic recursion is possible too, such as creating a Triangle number



using the lambda formula

=LAMBDA(x, IF(x<2, 1, x + Triangle(x - 1)))

Now, yes, I know you can use the calculation

= x * (x + 1) / 2

but nobody likes a smart alec! You get the point.

To celebrate **LAMBDA** moving to Current Channel Preview, Microsoft has also added support for optional parameters. To make use of these new take-them-or-leave-them arguments, all you need to do is wrap the optional name in square brackets, **"[]"**, *e.g.*

=LAMBDA(arg1, [arg2], IF(ISOMITTED(arg2), arg1, arg2))

(In case you have no idea what **ISOMITTED** does (I am sure you can guess!), don't worry, it's one of the new functions divulged below…)

Simply put, this lambda will return the value of **arg1** if **arg2** is omitted; otherwise, it will return the value of **arg2**. I'd like to have called this lambda function Jason as it sort of checks if his args are naught, but sadly it's not Friday 13th…

The lambda function is available to members of the Current Channel Preview program running Windows and Mac builds of Excel, but is only available to a random 50% of users as at the time of writing. Microsoft has stated that they will increase this "flight", pending no bugs or other issues.

Introducing the New LAMBDA Helper Functions

LAMBDA now enhances Excel's formula language, with its ability to be treated as an accepted value type with the introduction of these new functions. This is an important concept, which has existed across many programming languages, and is tantamount to the concept of lambda functions in general, never mind just in Excel.

Recently, Excel has introduced new types of values. This has included Data Types (Wolfram, Geography, Stocks, Power BI and Power Query can create Data Types) and dynamic arrays. Lambdas continue these enhancements by allowing Excel to understand functions as a value. This was enabled by the introduction of **LAMBDA**, but it requires support, which is what these seven new functions bring.

This means that previous Excel tasks which previously seemed practically impossible, may now be achieved by writing a **LAMBDA** and passing it as a *value* into another function.

Bring on the newbies…

BYCOL

BYCOL is not a region of the Philippines, but rather a function that takes an array or range and calls a lambda, with all the data grouped by each row or column and then returns an array of single values.

Its syntax is as follows:

BYCOL(array, [lambda])

It has the following arguments:

- **array:** this is required, and represents an array to be separated by column
- **lambda:** an optional argument, this is a **LAMBDA** that takes a column as a single parameter and calculates just one result.

As an example, my co-author Chris and I have decided to make small talk and discuss the *weather* – in particular, *whether* we use Celsius or Fahrenheit as our temperature scale. Chris uses Fahrenheit because he is based in the United States (more on this fact later) and I use Celsius because I am *right*.

I have made up some average monthly temperatures for Melbourne (Australia), not that it matters given we are all in Lockdown:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		***Average Temperatures, by Month and Year, Melbourne (Australia), Degrees Celsius***												
2														
3		**Month**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**		
4		January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3		
5		February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7		
6		March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1		
7		April	13.1	13.0	19.8	13.9	17.2	18.5	12.2	11.4	16.9	11.4		
8		May	11.6	11.5	13.6	15.6	10.0	10.5	14.8	15.3	14.0	12.4		
9		June	7.6	10.4	6.9	10.0	9.3	13.3	13.5	11.1	7.8	10.7		
10		July	9.0	11.2	6.8	12.5	12.0	12.5	12.1	9.4	11.1	6.7		
11		August	8.2	7.5	11.5	11.8	10.7	9.3	10.7	14.4	6.7	13.2		
12		September	14.2	10.3	14.7	17.3	9.4	14.9	8.1	10.2	15.8	14.4		
13		October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8		
14		November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0		
15		December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8		
16														

I have called this Excel Table **Temps**, but it is a permanent name…

For the years 2011 to 2020 inclusive, I want to provide a detailed monthly breakdown of temperatures where the average temperature for the year was above 15 degrees Celsius. At this point, you would normally calculate the average temperature for each column using a formula such as

=AVERAGE(Temps[2011])

for column **C** of this example spreadsheet.

This would be a "helper row", *e.g.*

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		***Average Temperatures, by Month and Year, Melbourne (Australia), Degrees Celsius***													
2															
3		**Month**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**			
4		January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3			
5		February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7			
6		March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1			
7		April	13.1	13.0	19.8	13.9	17.2	18.5	12.2	11.4	16.9	11.4			
8		May	11.6	11.5	13.6	15.6	10.0	10.5	14.8	15.3	14.0	12.4			
9		June	7.6	10.4	6.9	10.0	9.3	13.3	13.5	11.1	7.8	10.7			
10		July	9.0	11.2	6.8	12.5	12.0	12.5	12.1	9.4	11.1	6.7			
11		August	8.2	7.5	11.5	11.8	10.7	9.3	10.7	14.4	6.7	13.2			
12		September	14.2	10.3	14.7	17.3	9.4	14.9	8.1	10.2	15.8	14.4			
13		October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8			
14		November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0			
15		December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8			
16															
17		**Average**	14.63	13.52	15.39	16.38	14.00	15.42	15.46	14.35	14.13	15.29		*=AVERAGE(Temps[2011])*	

But I am not going to do it that way. Instead, let's use **BYCOL**. First of all, let's see how it works. Consider the calculation

=BYCOL(Temps, LAMBDA(column, SUM(column)))

Note I have used **Temps** as my array, which is cells **B4:L15**, *i.e.* it omits the header row of the table (cells **B3:L3**). I have to ignore the headers because the years would be included in the column totals being numbers – a classic *gotcha!*

This would produce the following result:

Month	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3
February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7
March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1
April	13.1	13.0	19.8	13.9	17.2	18.5	12.2	11.4	16.9	11.4
May	11.6	11.5	13.6	15.6	10.0	10.5	14.8	15.3	14.0	12.4
June	7.6	10.4	6.9	10.0	9.3	13.3	13.5	11.1	7.8	10.7
July	9.0	11.2	6.8	12.5	12.0	12.5	12.1	9.4	11.1	6.7
August	8.2	7.5	11.5	11.8	10.7	9.3	10.7	14.4	6.7	13.2
September	14.2	10.3	14.7	17.3	9.4	14.9	8.1	10.2	15.8	14.4
October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8
November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0
December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8
-	175.5	162.2	184.7	196.5	168.0	185.0	185.5	172.2	169.5	183.5

BYCOL produces a row vector, summing up each column of the table **Temps**, excluding the header row. The formula spills, using dynamic array logic and matches the width of the underlying array (*i.e.* the Table **Temps**). It only produces one row of data, as we have created a summation (just one value to report for each column).

Now, let's consider the following formula:

=FILTER(Temps, BYCOL(Temps, LAMBDA(year, AVERAGE(year) > 15)))

Here, **BYCOL** produces a row of TRUE or FALSE values, depending upon whether the average for each column exceeds 15 degrees Celsius. The dynamic array function **FILTER**, one of the recently introduced dynamic array functions then filters each column in **Temps** based upon whether the corresponding **LAMBDA** equates to TRUE or FALSE, *viz*.

18.0	21.0	14.6	24.6	25.3
23.3	20.7	24.2	24.5	14.7
14.4	19.7	21.9	14.4	22.1
19.8	13.9	18.5	12.2	11.4
13.6	15.6	10.5	14.8	12.4
6.9	10.0	13.3	13.5	10.7
6.8	12.5	12.5	12.1	6.7
11.5	11.8	9.3	10.7	13.2
14.7	17.3	14.9	8.1	14.4
14.0	13.9	13.3	18.2	18.8
21.6	21.4	11.5	18.6	15.0
20.1	18.7	20.5	13.8	18.8

This returns the columnar data for the years 2013, 2014, 2016, 2017 and 2020 respectively – not that you would know from the above numerical dataset. I really wanted the headings, but having numerical values in the Table header did not help my cause (it would have caused my averages to calculate incorrectly). It is usually not a good idea to have numerical values in Table headers, and perhaps now you can understand why.

If I modify the Table's headers as follows, I can now use the *entire* Table:

Month	Yr 2011	Yr 2012	Yr 2013	Yr 2014	Yr 2015	Yr 2016	Yr 2017	Yr 2018	Yr 2019	Yr 2020
January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3
February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7
March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1
April	13.1	13.0	19.8	13.9	17.2	18.5	12.2	11.4	16.9	11.4
May	11.6	11.5	13.6	15.6	10.0	10.5	14.8	15.3	14.0	12.4
June	7.6	10.4	6.9	10.0	9.3	13.3	13.5	11.1	7.8	10.7
July	9.0	11.2	6.8	12.5	12.0	12.5	12.1	9.4	11.1	6.7
August	8.2	7.5	11.5	11.8	10.7	9.3	10.7	14.4	6.7	13.2
September	14.2	10.3	14.7	17.3	9.4	14.9	8.1	10.2	15.8	14.4
October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8
November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0
December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8

The formula may be modified to

=FILTER(Temps[#All], BYCOL(Temps[#All], LAMBDA(year, AVERAGE(year) > 15)))

which will produce a more informative spilled array:

Yr 2013	Yr 2014	Yr 2016	Yr 2017	Yr 2020
18.0	21.0	14.6	24.6	25.3
23.3	20.7	24.2	24.5	14.7
14.4	19.7	21.9	14.4	22.1
19.8	13.9	18.5	12.2	11.4
13.6	15.6	10.5	14.8	12.4
6.9	10.0	13.3	13.5	10.7
6.8	12.5	12.5	12.1	6.7
11.5	11.8	9.3	10.7	13.2
14.7	17.3	14.9	8.1	14.4
14.0	13.9	13.3	18.2	18.8
21.6	21.4	11.5	18.6	15.0
20.1	18.7	20.5	13.8	18.8

BYROW

BYROW works very similarly to **BYCOL** (it is analogous to the relationship between **HLOOOKUP** and **VLOOKUP**). This function applies a **LAMBDA** to each row and returns an array of the results. Its syntax is as follows:

BYROW(array, [lambda])

It has the following arguments:

- **array:** this is required, and represents an array to be separated by row
- **lambda:** an optional argument, this is a **LAMBDA** that takes a column as a single parameter and calculates just one result.

Let's return to my above example:

Month	Yr 2011	Yr 2012	Yr 2013	Yr 2014	Yr 2015	Yr 2016	Yr 2017	Yr 2018	Yr 2019	Yr 2020
January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3
February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7
March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1
April	13.1	13.0	19.8	13.9	17.2	18.5	12.2	11.4	16.9	11.4
May	11.6	11.5	13.6	15.6	10.0	10.5	14.8	15.3	14.0	12.4
June	7.6	10.4	6.9	10.0	9.3	13.3	13.5	11.1	7.8	10.7
July	9.0	11.2	6.8	12.5	12.0	12.5	12.1	9.4	11.1	6.7
August	8.2	7.5	11.5	11.8	10.7	9.3	10.7	14.4	6.7	13.2
September	14.2	10.3	14.7	17.3	9.4	14.9	8.1	10.2	15.8	14.4
October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8
November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0
December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8

BYROW effectively produces a column vector, summing up each row of the table **Temps**.

If I want the year-on-year comparisons for each month where the average temperature is above 15 degrees Celsius, I can again avoid using a "helper column" and instead use the formula

=FILTER(Temps, BYROW(Temps, LAMBDA(year, AVERAGE(year) > 15)))

This time, I can ignore the header row. This will return the array

January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3
February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7
March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1
October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8
November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0
December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8

Once you get the hang of these **LAMBDA** helper functions, you will begin to wonder how you ever managed without them.

ISOMITTED

This new function checks whether the value is missing, and returns either TRUE (value is missing) or FALSE (value is not missing) accordingly. The syntax is simple:

ISOMITTED(argument)

where:

- **argument** is a required parameter, and is the value you want to test, which may be based upon a **LAMBDA**.

The example

=LAMBDA(arg1, [arg2], IF(ISOMITTED(arg2), arg1, arg2))

was provided earlier, where this lambda will return the value of **arg1** if **arg2** is omitted; otherwise, it will return the value of **arg2**.

MAKEARRAY

MAKEARRAY returns a calculated array of a specified row and column size, by applying a **LAMBDA** function. This function is useful for situations where you wish to combine or transform arrays, as well as being useful for generating data. The syntax is as follows:

MAKEARRAY(rows, columns, lambda)

It has the following arguments:

- **rows:** this argument is required and represents the number of rows in the array (which must be greater than zero)
- **columns**: this argument is also required and represents the number of columns in the array (which again must be greater than zero)
- **lambda:** also necessary, this is the **LAMBDA** that is called to create the array. In particular, this lambda function must take two parameters, namely:
  - **row_index:** the index of the row (row number)
  - **column_index:** the index of the column (column number).

As an example, consider the following:

	A	B	C	D	E	F	G
1							
2		Number of rows		5			
3		Number of columns		3			
4							
5		Colours of Rainbow					
6		Red			Red	Orange	Violent
7		Orange			Orange	Red	Blue
8		Yellow			Red	Violent	Indigo
9		Green			Blue	Indigo	Violent
10		Blue			Green	Blue	Red
11		Indigo					
12		Violent					

Imagine, for reasons best known to myself, I wanted to generate an array of colours of the rainbow (albeit with the final colour, ahem, slightly amended). In the image above, I have specified the number of rows (cell **D2**) and the number of columns (cell **D3**) in my array, and listed the colours in cells **B6:B12** inclusive.

The formula in cell **E6** is given by

=MAKEARRAY(D2, D3, LAMBDA(row, column, INDEX(B6:B12, RANDBETWEEN(1, 7))))

The first two arguments in this formula are **D2** and **D3**, which refer to the number of rows and columns for the array to be generated respectively. The final argument of **MAKEARRAY** is the **LAMBDA**, which must take two parameters, corresponding to the value generated by **LAMBDA**, namely:

- **row:** the index of the row
- **column:** the index of the column.

The calculation thus uses the non-dynamic array function **RANDBETWEEN** to generate an integer between one [1] and seven [7] to select from the list of colours of the rainbow, stipulated in cells B6:B12. For example, if Excel generates the number 5, the value "Blue" will be chosen, *etc.*

Now it is true that existing functions could be used to achieve the same result, *e.g.*

=INDEX(B6:B12, RANDARRAY(D2, D3, 1, 7, TRUE))

This formula seems shorter and simpler, and indeed, may be the better option for this above illustration. But that is exactly what this is – a simple example. As more complex arrays need to be created, existing function counterparts may prove difficult, convoluted or impossible to construct – and this is precisely where **MAKEARRAY** and **LAMBDA** come in.

MAP

You should be aware that Excel's 500th function *(see earlier)*, the **MAP** function, does not actually return a map!

Instead, it returns an array formed by mapping each value in the array(s) to a new value and applying a **LAMBDA** to create a new value accordingly. It has the following syntax:

MAP(array1, lambda or array2, [lambda or array2, …])

where:

- **array1:** this is a required argument and represents the (first) array to be mapped
- **array2 and subsequent arrays:** these are optional arguments and represent additional arrays to be mapped
- **lambda:** this is a required argument which represents a **LAMBDA** which must be the final argument and must have a parameter for each array passed or another array to be mapped.

In short, **MAP** transforms values. Let's return to my Melbourne temperatures data:

Month	Yr 2011	Yr 2012	Yr 2013	Yr 2014	Yr 2015	Yr 2016	Yr 2017	Yr 2018	Yr 2019	Yr 2020
January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3
February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7
March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1
April	13.1	13.0	19.8	13.9	17.2	18.5	12.2	11.4	16.9	11.4
May	11.6	11.5	13.6	15.6	10.0	10.5	14.8	15.3	14.0	12.4
June	7.6	10.4	6.9	10.0	9.3	13.3	13.5	11.1	7.8	10.7
July	9.0	11.2	6.8	12.5	12.0	12.5	12.1	9.4	11.1	6.7
August	8.2	7.5	11.5	11.8	10.7	9.3	10.7	14.4	6.7	13.2
September	14.2	10.3	14.7	17.3	9.4	14.9	8.1	10.2	15.8	14.4
October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8
November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0
December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8

The formula

=FILTER(Temps, BYROW(Temps, LAMBDA(year, AVERAGE(year) > 15)))

which returned the array

January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3
February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7
March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1
October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8
November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0
December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8

The problem is, these temperatures have all been provided in Celsius, which my co-author Chris doesn't really understand. If he sees a temperature of 25 degrees, he will be breaking out the gloves, bobble hat and duffle coat, whereas us Aussies will be heading for the beach.

We need to convert – ***transform*** – this data to Fahrenheit, so our US colleagues may better understand. All I need to do is wrap the above formula in a **MAP** function:

=MAP(FILTER(Temps, BYROW(Temps, LAMBDA(year, AVERAGE(year) > 15))),
LAMBDA(temperature, IF(ISNUMBER(temperature),
CONVERT(temperature, "C", "F"), temperature)))

January	77.0	63.5	64.4	69.8	65.1	58.3	76.3	69.1	61.2	77.5
February	58.5	65.1	73.9	69.3	63.1	75.6	76.1	74.8	62.6	58.5
March	70.2	59.7	57.9	67.5	60.6	71.4	57.9	64.8	57.6	71.8
October	60.3	59.7	57.2	57.0	62.4	55.9	64.8	52.2	64.0	65.8
November	71.6	52.9	70.9	70.5	62.4	52.7	65.5	52.2	53.2	59.0
December	55.8	68.0	68.2	65.7	57.2	68.9	56.8	59.7	68.4	65.8

CONVERT(temperature, "C", "F") simply converts the variable **temperature** from degrees Celsius to degrees Fahrenheit. This is wrapped in an **IF(ISNUMBER())** check to ensure that we don't try to convert text values (as this would cause an error): the **IF** statement leaves the value of **temperature** "as is" in this instance, and **LAMBDA** just wraps around all of this in order to declare the variable **temperature** "work", so that **MAP** may do its work.

It's true you could generate this result in stages, but the whole idea of these **LAMBDA** helper functions is to be able to create dynamic arrays in one fell swoop.

REDUCE

This penultimate function reduces an array to an accumulated value by applying a **LAMBDA** function to each value and returning the total value in what is known as the **accumulator**. Its syntax is as follows:

REDUCE([initial_value], array, lambda)

where:

- **initial_value:** this is an optional argument and represents the starting value for the **accumulator**, *i.e.* the "running total" prompted by the **lambda** expression
- **array:** this is a required value and represents the array to be reduced
- **lambda:** this is also a required value and represents a **LAMBDA** function called to reduce the array, that consists of two parameters:
  - accumulator: the returned (aggregated) value from **LAMBDA**
  - **value:** a value from **array**.

Returning to our temperature example given by the Excel Table **Temp**:

Month	Yr 2011	Yr 2012	Yr 2013	Yr 2014	Yr 2015	Yr 2016	Yr 2017	Yr 2018	Yr 2019	Yr 2020
January	25.0	17.5	18.0	21.0	18.4	14.6	24.6	20.6	16.2	25.3
February	14.7	18.4	23.3	20.7	17.3	24.2	24.5	23.8	17.0	14.7
March	21.2	15.4	14.4	19.7	15.9	21.9	14.4	18.2	14.2	22.1
April	13.1	13.0	19.8	13.9	17.2	18.5	12.2	11.4	16.9	11.4
May	11.6	11.5	13.6	15.6	10.0	10.5	14.8	15.3	14.0	12.4
June	7.6	10.4	6.9	10.0	9.3	13.3	13.5	11.1	7.8	10.7
July	9.0	11.2	6.8	12.5	12.0	12.5	12.1	9.4	11.1	6.7
August	8.2	7.5	11.5	11.8	10.7	9.3	10.7	14.4	6.7	13.2
September	14.2	10.3	14.7	17.3	9.4	14.9	8.1	10.2	15.8	14.4
October	15.7	15.4	14.0	13.9	16.9	13.3	18.2	11.2	17.8	18.8
November	22.0	11.6	21.6	21.4	16.9	11.5	18.6	11.2	11.8	15.0
December	13.2	20.0	20.1	18.7	14.0	20.5	13.8	15.4	20.2	18.8

we could count how many months in the 10-year period had an average temperature between 15 and 20 degrees Celsius as follows:

=REDUCE(0, Temps, LAMBDA(accumulator, value,
IF(AND(value >= 15, value <= 20), 1 + accumulator, accumulator)))

For each element of the **Temps** Table, defined by the **LAMBDA** function as **value**, the **IF** statement tests whether the temperature is between 15 and 20 degrees Celsius:

AND(value >= 15, value <= 20)

If this is true, one gets added to the running total (**accumulator**), so that a count is maintained. The first argument of **REDUCE** – zero [0], the optional argument – simply specifies the starting value (**initial_value**) for the **accumulator**, which must be zero in order for the count to make sense.

Again, we could create a second **array** which performs a corresponding check for each cell and count the TRUE values, but this formula *reduces* the workload (*i.e.* it reduces the **array** of values to just one value by making use of the specified **LAMBDA**), allowing the computation to be performed once again without any helper stages.

SCAN

This final function scans an array by applying a **LAMBDA** to each value and returns an array that has each intermediate value. The syntax is as follows:

SCAN([initial_value], array, lambda)

where:

- **initial_value:** this is an optional argument and represents the starting value for the accumulator, *i.e.* the "running total" prompted by the **lambda** expression
- **array:** this is a required value and represents the array to be scanned
- **lambda:** this is also a required value and represents a **LAMBDA** function called to scan the array, that consists of two parameters:
  - **accumulator:** the returned (aggregated) value from **LAMBDA**
  - **value:** a value from **array**.

As a simple example, let's consider a common problem when working with structured references, *i.e.* Excel Tables. Imagine I have the following sales for the first six months of the year:

	A	B
1	Month	Sales
2	January	971
3	February	559
4	March	784
5	April	635
6	May	858
7	June	917

I might wish to create a running total of these sales. One way I have seen people do this is as follows:

	A	B	C	D	E
1	Month	Sales	Cumulative		
2	January	971	971		*=N(C1)+[@Sales]*
3	February	559	1,530		
4	March	784	2,314		
5	April	635	2,949		
6	May	858	3,807		
7	June	917	4,724		

This is a horrible "hotch potch" of a formula:

=N(C1) + [@Sales]

It mixes Excel cell referencing (cell **C1**, because you cannot refer to a value for a different record simply in an Excel Table), structured referencing (**[@Sales]**) and the **N** function, in order to treat the numerical value of text as zero [0] and therefore avoid *#VALUE!* errors when adding amounts together.

It seems to work if values are added:

	A	B	C	D	E
1	Month	Sales	Cumulative		
2	January	971	971		*=N(C1)+[@Sales]*
3	February	559	1,530		
4	March	784	2,314		
5	April	635	2,949		
6	May	858	3,807		
7	June	917	4,724		
8	July	1,000	5,724		
9	August	2,000	7,724		

However, it all goes pear shaped when values are inserted:

	A	B	C	D	E
1	Month	Sales	Cumulative		
2	January	971	971		=N(C1)+[@Sales]
3	February	559	1,530		
4	March	784	2,314		
5	April	635	2,949		
6	Additional	1,000,000	1,002,949		
7	May	858	3,807		
8	June	917	4,724		
9	July	1,000	5,724		
10	August	2,000	7,724		

This is where **SCAN** comes to the rescue. Assuming the Table is also called **Sales** (not just the field in column **B**), we can create the formula

=SCAN(0, Sales[Sales], LAMBDA(accumulator, value, accumulator + value))

SCAN "scans" the array (*i.e.* the Excel Table **Sales**) by applying a **LAMBDA** to each value. It then returns an array of results corresponding to the accumulator value returned by the **LAMBDA**. As stated above, **SCAN** takes two parameters:

- **accumulator:** the initial value returned by **SCAN** and each **LAMBDA** call
- **value:** a **value** from the supplied **array**.

As above, the **initial_value** is zero [0] so that the running total calculates correctly.

	A	B	C	D
1	Month	Sales		Cumulative
2	January	971		971
3	February	559		1,530
4	March	784		2,314
5	April	635		2,949
6	Additional	1,000,000		1,002,949
7	May	858		1,003,807
8	June	917		1,004,724
9	July	1,000		1,005,724
10	August	2,000		1,007,724

Word to the Wise

These functions will not be for everyone, in more ways than one. There are two key points:

1. The concepts discussed here revolve around the notion of lambda functions, which may not be relevant to all users, especially for those that use Excel for "simple" tasks. But don't let it scare or deter you from experimenting with these functions
2. These new functions are only available presently to c. 50% of Office Insiders users running Beta Channel Version 2108 (Build 14312.20008) or later on Windows, or Version 16.52 (Build 21072100) or later on Mac. The **LAMBDA** function is now available to Office Insiders running Current Channel Preview Version 2107 (Build 14228.20154) or later on Windows, or Version 16.51 (Build 21071101) or later on Mac.

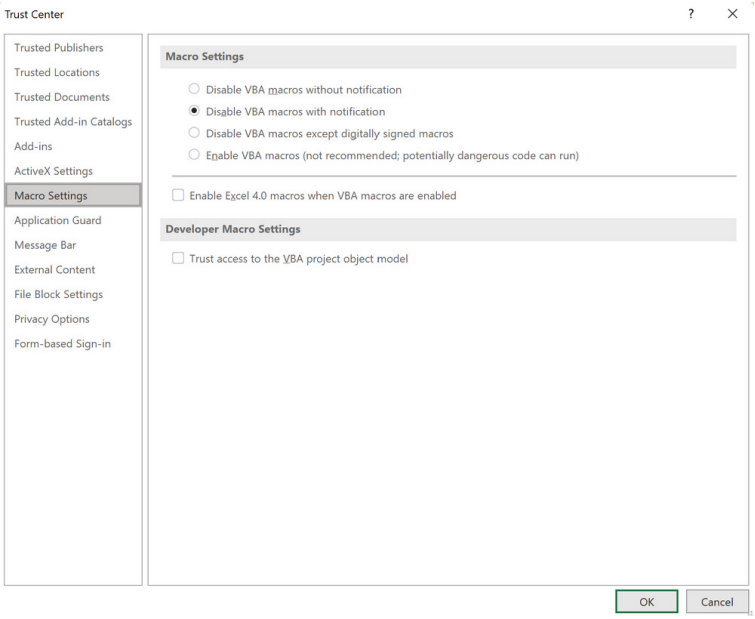
Restrict Usage of Excel 4.0 (XLM) Macros with New Macro Settings Control

A new Excel Trust Center *(sic)* settings option to further restrict the usage of Excel 4.0 (XLM) macros has now been made Generally Available. To better protect Microsoft 365 users against malicious macro-based threats, Microsoft recently expanded the integration of what is known as the Antimalware Scan Interface (AMSI) with Office 365 to include the runtime scanning of Excel 4.0 (XLM) macros.

Building on the recent release of AMSI integration for XLM macros, this new Trust Center setting enables Microsoft 365 customers to further protect themselves against the latest threats. Located in the 'Trust Center Macro Settings', this new checkbox setting, 'Enable Excel 4.0 macros when VBA macros are enabled', allows users to individually configure the behaviour of XLM macros ***without*** impacting their VBA macro counterparts.

The Excel Trust Center settings may be accessed through **File > Options > Trust Center > Trust Center Settings > Macro Settings**.

When the checkbox is selected, the above settings configured for VBA macros will also apply to XLM macros. To disable XLM macros without a notification, deselect the checkbox setting (this is the option recommended by Microsoft), as this configuration opts for a more secure behaviour. There is no impact to any default or previous macro settings configurations with this release; however, users should be aware that a change in default XLM macro behaviour is coming soon *(see below)*.



Customers can now independently disable XLM macros in the Trust Center Macro Settings by unchecking the setting "Enable Excel 4.0 macros when VBA macros are enabled."

Availability

This setting is currently available in Excel (build 2104).

Administrators can also use the existing Microsoft 365 applications policy control to configure this setting. The Group Policy setting 'Macro Notification Settings may be found at **User configuration > Administrative templates > Microsoft Excel 2016 > Excel Options > Security > Trust Center**.

Administrators also have the option to completely block all XLM macro usage (including in new user-created files) by enabling the Group Policy, 'Prevent Excel from running XLM macros', which is configurable via Group Policy Editor or registry key:

- Group Policy Path: **User configuration > Administrative templates > Microsoft Excel 2016 > Excel Options > Security > Trust Center**
- Registry Key Path: **Computer\HKEY_CURRENT_USER\SOFTWARE\Policies\Microsoft\Office\16.0\excel\security**

Do be aware that while the initial release of this setting does not impact any existing or default macro settings configurations, XLM macros will soon be disabled by default. Users should expect this upcoming change in default behaviour to occur in the following M365 updates:

- 2021 October Current Channel
- 2021 December Monthly Enterprise Channel
- 2022 January Semi-Annual Enterprise Channel (Preview)
- 2022 July Semi-Annual Enterprise Channel.

Beat the Boredom Challenge

*With many of us currently "working from home" / quarantined, there are only so Zoom / Teams calls and virtual parties you can make before you reach your (data) limit. Perhaps they should measure data allowance in blood pressure millimetres of mercury (mmHg). To try and keep our readers engaged, we will continue to reproduce some of our popular **Final Friday Fix** challenges from yesteryear in this and upcoming newsletters. One suggested solution may be found later in this newsletter. Here's this month's…*

Most models are not constructed on a daily basis. Data and calculations are consolidated on a weekly, monthly, quarterly or annual basis. Sometimes this causes problems – such as in the following example, a continuation of last month's challenge.

One such issue is when data straddles two or more consolidation periods, and this is something we have looked at before (*e.g.* Revising Forecasts). Indeed, we are going to complicate last month's challenge by adding *another* dimension: fees.

With management impressed by you solving last month's problem, imagine that you continue to work in an education establishment, seeking to model forecast fees for this calendar year. There are several terms relevant to your modelling period, similar to our last challenge, but now with fees data collated too:

Assumed Term Dates

Term	Start	End	Fees ($)	Days in Year	Check
1	30 Nov 20	6 Mar 21	10,000	65	☑
2	27 May 21	17 Jun 21	8,000	22	☑
3	1 Sep 21	4 Oct 21	12,000	34	☑
4	17 Oct 21	7 Aug 22	9,000	76	☑
			39,000	**197**	☑

All we need to do is allocate the fees (including weekends and public holidays) to each month of 2021, *i.e.*

		Jan 21	Feb 21	Mar 21	Apr 21	May 21	Jun 21	Jul 21	Aug 21	Sep 21	Oct 21	Nov 21	Dec 21
Term Fees in Each Month													
Term Fees in Each Month	29,020	3,196	2,887	619	-	1,818	6,182	-	-	10,588	1,869	915	946

The challenge is "simply" this: can you construct a calculation such that the correct fees will be allocated to each month? You may rely upon last month's solution, and assume the terms will be in chronological order, they will not overlap, there will never be more than two terms associated with any given month, and there will never be two start dates or two end dates in the same month.

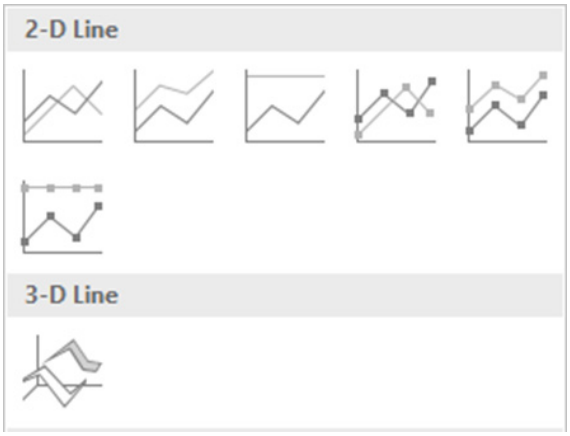
Sound easy? Try it. One solution *just might* be found later in this newsletter – but no reading ahead!

Charts and Dashboards

It's time to chart our progress with an introductory series into the world of creating charts and dashboards in Excel. This month, we look at Line charts.

The Line chart is one of the most frequently used charts. This is often necessary where many data points need to be visualised and it is arguably the simplest visualisation tool given that situation. The purpose of this type of graph is to demonstrate trends, typically over multiple time periods. Quite simply, the numbers within each data series are linked together by lines to show movements from one point to the next.

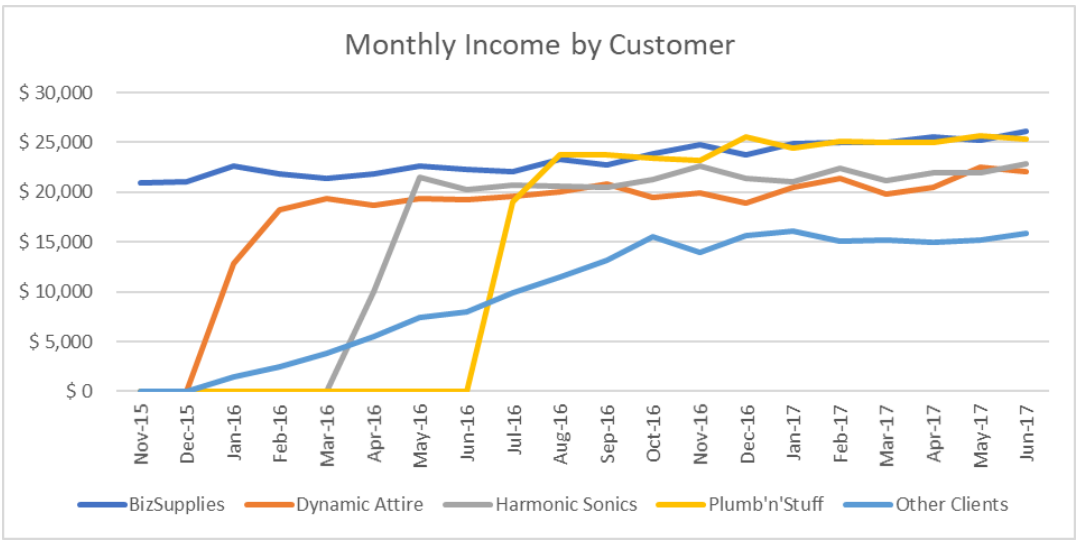
Excel provides a number of variations for Line charts, including the use of markers, stacked and 3-D.



For example, I have sample sales data in two financial years by customer group:

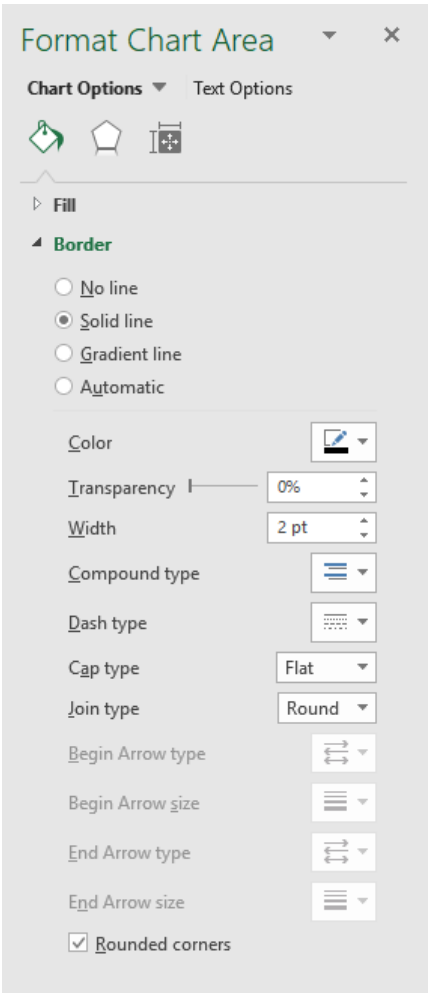
Monthly Income by Customer For 2015/16 and 2016/17											
Customer	Nov-15	Dec-15	Jan-16	Feb-16	Mar-16	Apr-16	. . . .	Mar-17	Apr-17	May-17	Jun-17
BizSupplies	$ 20,901	$ 21,034	$ 22,579	$ 21,872	$ 21,389	$ 21,787	. . . .	$ 24,988	$ 25,548	$ 25,229	$ 26,127
Dynamic Attire	$ 0	$ 0	$ 12,823	$ 18,255	$ 19,379	$ 18,665	. . . .	$ 19,785	$ 20,479	$ 22,549	$ 22,072
Harmonic Sonics	$ 0	$ 0	$ 0	$ 0	$ 0	$ 10,048	. . . .	$ 21,166	$ 21,989	$ 21,887	$ 22,850
Plumb'n'Stuff	$ 0	$ 0	$ 0	$ 0	$ 0	$ 0	. . . .	$ 24,952	$ 25,018	$ 25,692	$ 25,330
Other Clients	$ 0	$ 0	$ 1,430	$ 2,513	$ 3,835	$ 5,522	. . . .	$ 15,201	$ 15,002	$ 15,208	$ 15,857

A simple line chart using the data above would look like this:



Now, let's apply a few changes to improve this chart. Firstly, let's add some character to the chart frame:

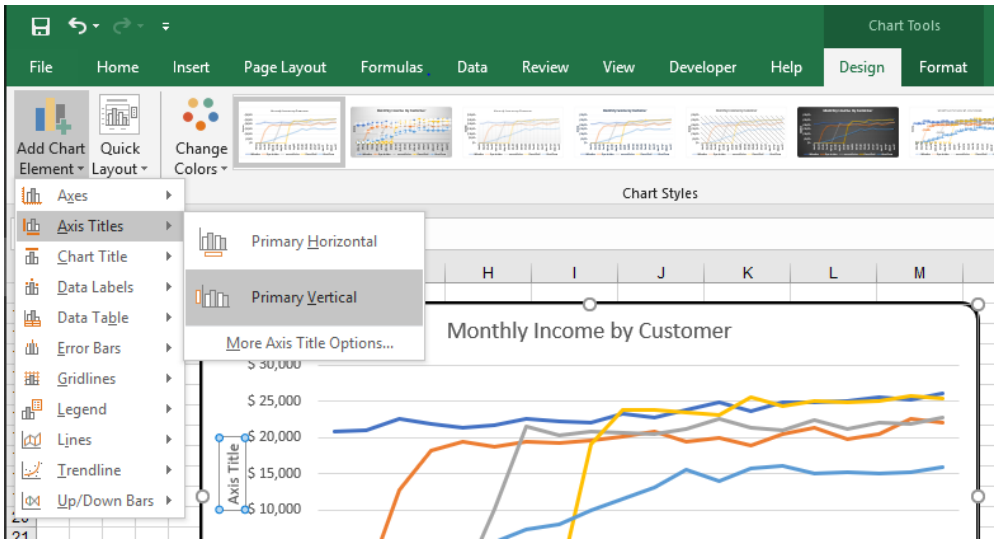
- click on the 'Chart Area', then right-click and choose 'Format Chart Area'
- under Border, choose 'Solid Line', change the colour to black, set the width to two (2) point and at the bottom tick for 'Rounded Corners'.



Next, my chart axes do not have titles (though admittedly they are pretty self-explanatory):

- click on any element of the chart and there will be two new menus on the Ribbon under the heading of 'Chart Tools', being Design and Format
- go to the Design tab and then click on the first button 'Add Chart Element' and pick 'Axis Titles', then 'Primary Vertical'. The words 'Axis Title' will appear along the vertical axis

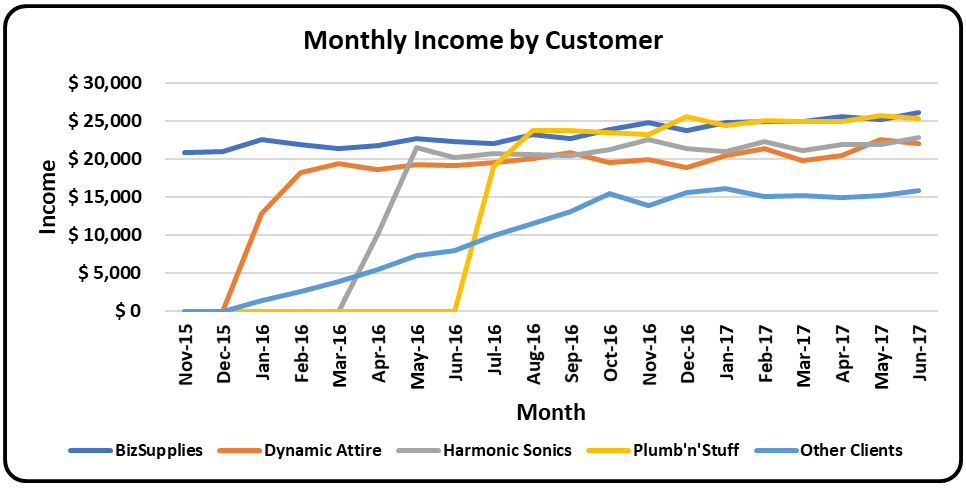
- click on the words and change it to read 'Income'. Increase the font from point 10 to 12 and make the title bold
- also embolden the 'Chart Title' and change the font colour to black
- repeat this process to add an 'Axis Title' on the 'Primary Horizontal' axis, label this 'Month' and also format it to point 12, black and bold.



Some final touches and the chart will be complete:

- click on the 'Chart Title' and change it to black and bold
- click on the vertical axis and change it to point 10, black and bold, then do the same for the horizontal axis
- click on the Legend and change it to black and bold.

Finally, now I have a line chart with more impact!



It's arguable whether we should embolden all (most psychologists would argue you should not), but remember, this is all for illustration purposes only!

We'll continue next month…

Visual Basics

We thought we'd run an elementary series going through the rudiments of Visual Basic for Applications (VBA) as a springboard for newer users. This month, we look at manipulating Tables within an Excel workbook, using VBA.

Warning! This was written whilst listening to a Nat King Cole playlist…

"Look What You've Done To Me", storing all the data in a table in the Excel workbook. How does one access the table from VBA and "Make Her Mine"? It's a lot simpler than one would think, "Pick Yourself Up" and consider the **ListObjects** method in VBA.

First, let's create a table from a Range in VBA. "I'm Never Satisfied" with storing data as raw, it's nicer to make it into a table. This involves using the Add method of **ListObjects**.

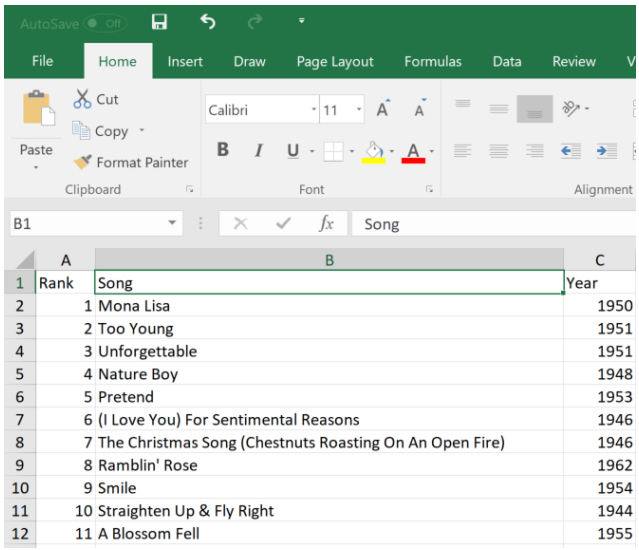
```
(General)                                                    CreateTable
Sub CreateTable()

    ActiveSheet.ListObjects.Add(xlSrcRange, Range("$A$1:$C$188"), , xlYes).Name = "NKCSongList"

End Sub
```

The parameters are as follows:

- **SourceType:** the source type. *xlSrcRange* indicates that is a Range item not a "Make Believe Island"
- **Source:** this is only applicable when *xlSrcRange* is used, so that VBA doesn't have to go searching "To The Ends of the Earth"
- **LinkSource:** this is when external data sets are used, and not applicable when using the *Range* item. "Don't Blame Me" for skipping it in this case as it isn't necessary
- **xlListObjectHasHeaders:** if the first row of the *Range* is actually a header row.

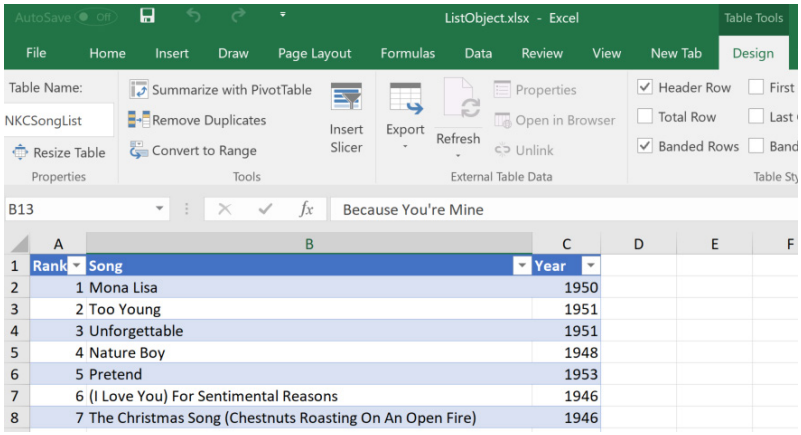
The *Name* method is added to the end of the **ListObjects** call to rename the table as required and stamp it "My Personal Possession".

Before:



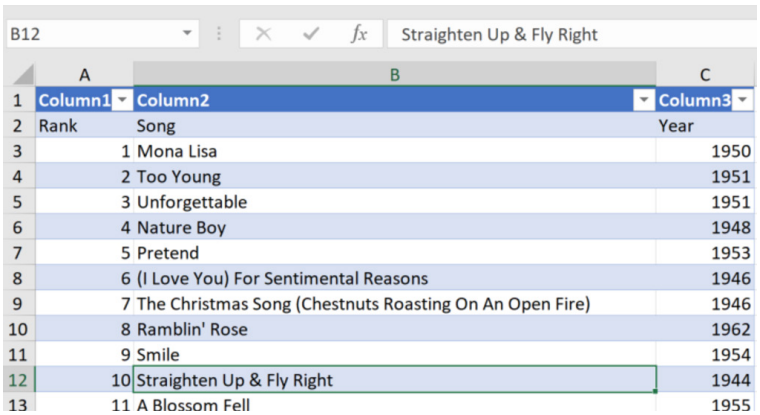
	A	B	C
1	Rank	Song	Year
2	1	Mona Lisa	1950
3	2	Too Young	1951
4	3	Unforgettable	1951
5	4	Nature Boy	1948
6	5	Pretend	1953
7	6	(I Love You) For Sentimental Reasons	1946
8	7	The Christmas Song (Chestnuts Roasting On An Open Fire)	1946
9	8	Ramblin' Rose	1962
10	9	Smile	1954
11	10	Straighten Up & Fly Right	1944
12	11	A Blossom Fell	1955

After:



	A	B	C
1	Rank	Song	Year
2	1	Mona Lisa	1950
3	2	Too Young	1951
4	3	Unforgettable	1951
5	4	Nature Boy	1948
6	5	Pretend	1953
7	6	(I Love You) For Sentimental Reasons	1946
8	7	The Christmas Song (Chestnuts Roasting On An Open Fire)	1946

"You Call It Madness", but if *xlListObjectHasHeaders* is set to xlNo for headers, the following result is achieved:



	A	B	C
1	Column1	Column2	Column3
2	Rank	Song	Year
3	1	Mona Lisa	1950
4	2	Too Young	1951
5	3	Unforgettable	1951
6	4	Nature Boy	1948
7	5	Pretend	1953
8	6	(I Love You) For Sentimental Reasons	1946
9	7	The Christmas Song (Chestnuts Roasting On An Open Fire)	1946
10	8	Ramblin' Rose	1962
11	9	Smile	1954
12	10	Straighten Up & Fly Right	1944
13	11	A Blossom Fell	1955

The range of the data is actually shifted down by one and generic headers have been created! By default, VBA will assume *xlGuess*, where it makes a judgment call on the data. It is best practice to specify to VBA directly the header intention, otherwise you may be "Looking Back" on unexpected results thinking "That Ain't Right" at all.

More next time.

Power Pivot Principles

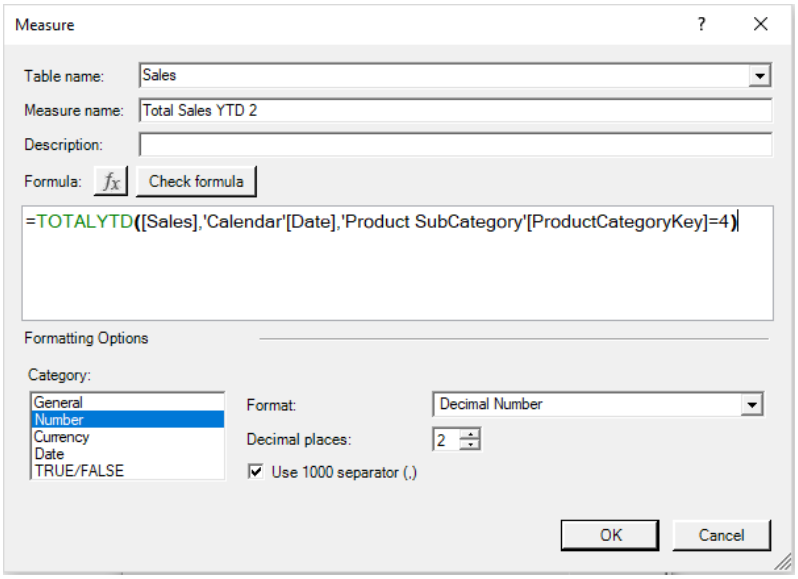
We continue our series on the Excel COM add-in, Power Pivot. We continue with our time intelligence functions this month, by discussing TOTALYTD.

TOTALYTD is a time intelligence function similar to the **DATESYTD** we discussed in last month's newsletter. The **TOTALYTD** function, like all time series functions, requires strict ascending and contiguous dates (*i.e.* dates must include all dates in order between the first and last dates with neither any gaps nor any duplication) in order to work properly.

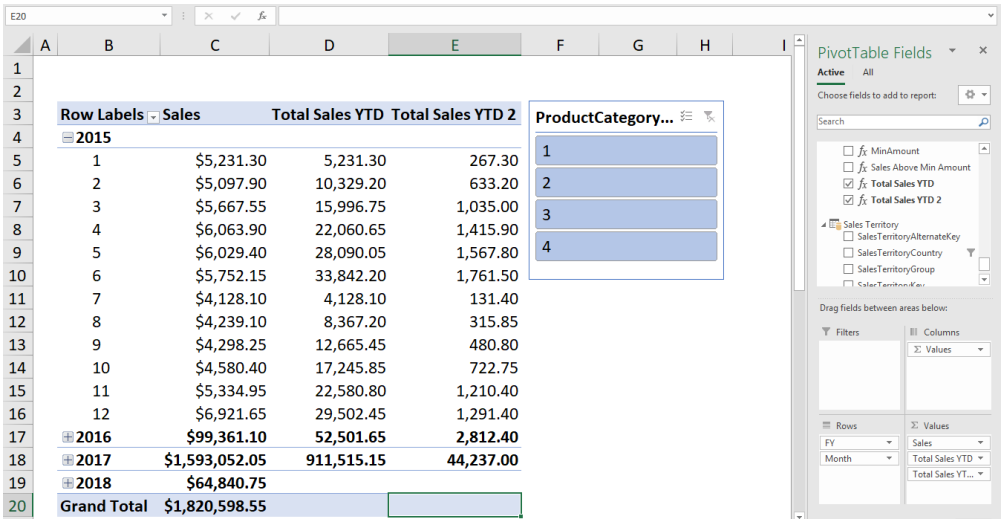
The **TOTALYTD** performs similarly to the **DATESYTD** function, but allows us to apply filters and specify a year end date. The syntax of the function is:

TOTALYTD(expression, dates[, filter][, year_end_date])

Unlike the **DATEYTD** function we do not need to use this function with **CALCULATE**. To demonstrate, let's create a measure to return the year to date (YTD) sales and filter on products assigned as category 4:



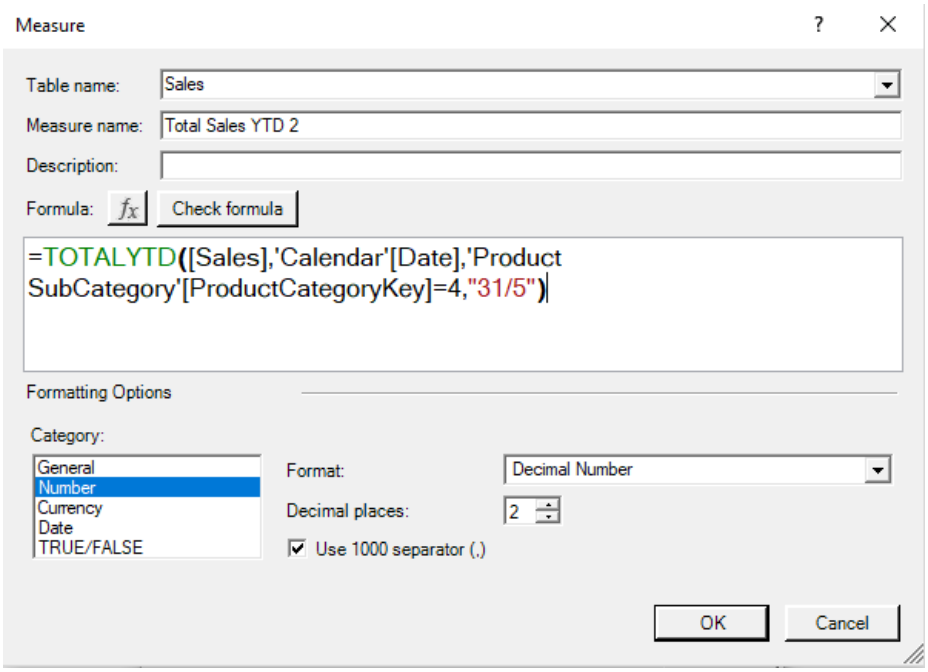
Now let's look at our PivotTable report:



Row Labels	Sales	Total Sales YTD	Total Sales YTD 2
2015			
1	$5,231.30	5,231.30	267.30
2	$5,097.90	10,329.20	633.20
3	$5,667.55	15,996.75	1,035.00
4	$6,063.90	22,060.65	1,415.90
5	$6,029.40	28,090.05	1,567.80
6	$5,752.15	33,842.20	1,761.50
7	$4,128.10	4,128.10	131.40
8	$4,239.10	8,367.20	315.85
9	$4,298.25	12,665.45	480.80
10	$4,580.40	17,245.85	722.75
11	$5,334.95	22,580.80	1,210.40
12	$6,921.65	29,502.45	1,291.40
2016	**$99,361.10**	**52,501.65**	**2,812.40**
2017	**$1,593,052.05**	**911,515.15**	**44,237.00**
2018	**$64,840.75**		
Grand Total	**$1,820,598.55**		

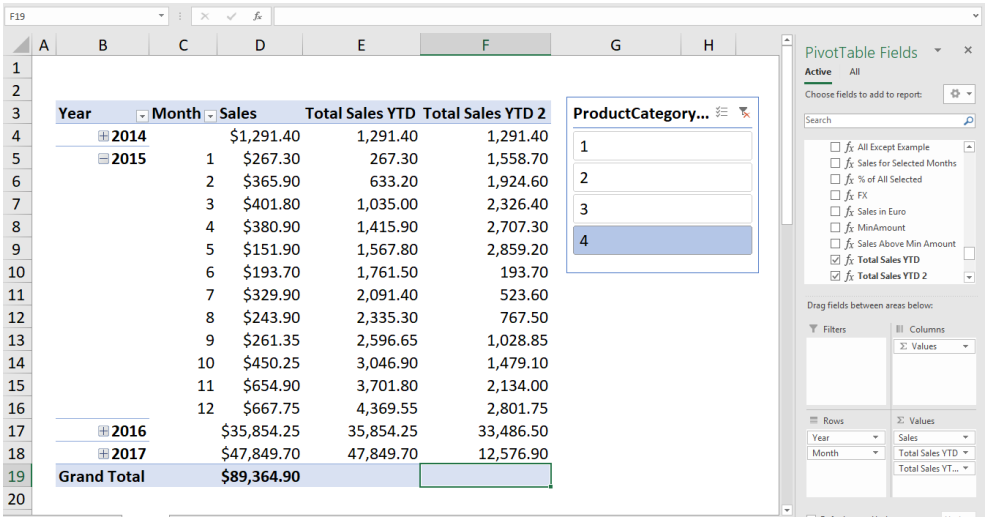
We can see that the **Total Sales YTD 2** measure is returning with the YTD sales but only for category 4 products, ignoring any slicers.

Lastly, we can specify the year end date, let's change the year end date to the end of May:



Note that the **year end date** has to be expressed as a string, therefore we use "31/5" to specify that the year end date is the last day of May.

Moving on to our PivotTable:



Year	Month	Sales	Total Sales YTD	Total Sales YTD 2
2014		$1,291.40	1,291.40	1,291.40
2015	1	$267.30	267.30	1,558.70
	2	$365.90	633.20	1,924.60
	3	$401.80	1,035.00	2,326.40
	4	$380.90	1,415.90	2,707.30
	5	$151.90	1,567.80	2,859.20
	6	$193.70	1,761.50	193.70
	7	$329.90	2,091.40	523.60
	8	$243.90	2,335.30	767.50
	9	$261.35	2,596.65	1,028.85
	10	$450.25	3,046.90	1,479.10
	11	$654.90	3,701.80	2,134.00
	12	$667.75	4,369.55	2,801.75
2016		$35,854.25	35,854.25	33,486.50
2017		$47,849.70	47,849.70	12,576.90
Grand Total		$89,364.90		

We can see that the **Total Sales YTD 2** measure now treats May as the end of the year.

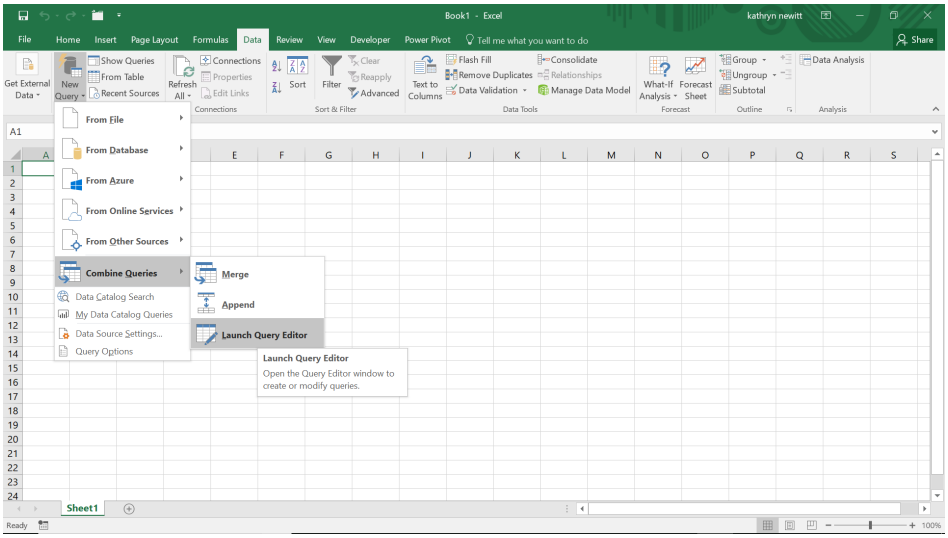
More *Power Pivot Principles* next month.

Power Query Pointers

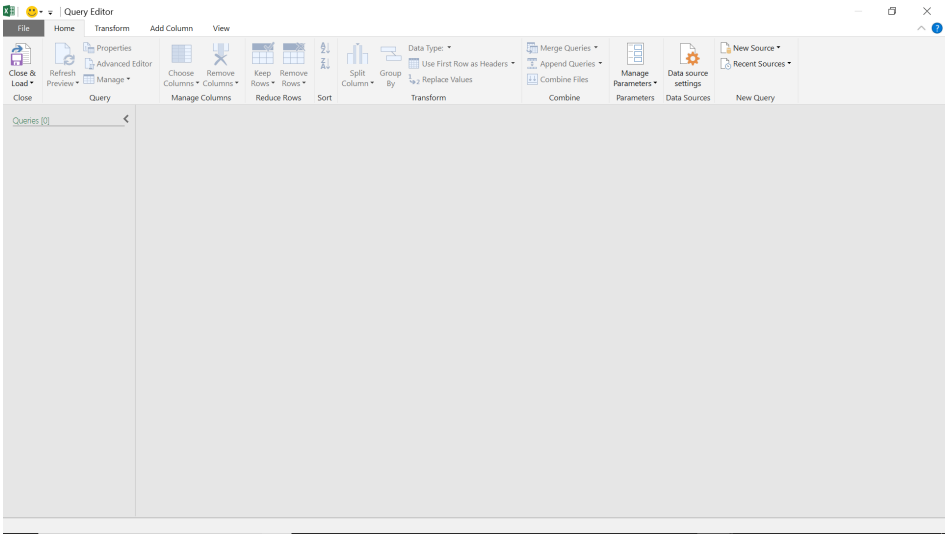
Each month we'll reproduce one of our articles on Power Query (Excel 2010 and 2013) / Get & Transform (Office 365, Excel 2016 and 2019) from www.sumproduct.com/blog. If you wish to read more in the meantime, simply check out our Blog section each Wednesday. This month, we look at a way to create and append new queries by starting with the Power Query Editor.

Many moons ago, one of our earliest articles looked as how to combine CSV files. Whilst this method is fine for files that appear at different times, sometimes I have a group of files that I need to append quickly. For this example, I am assuming that the files are not all in the same folder.

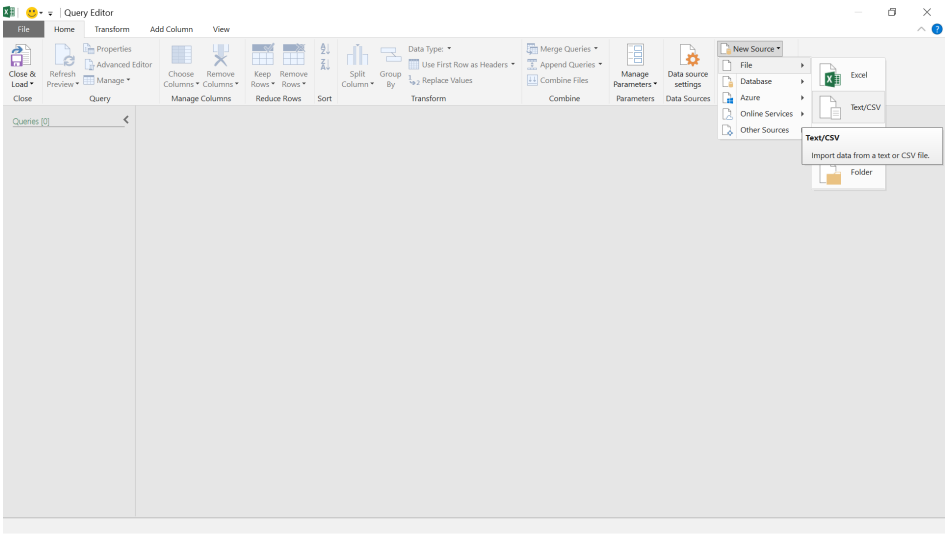
Let's start with a new workbook, and go to the Data tab.



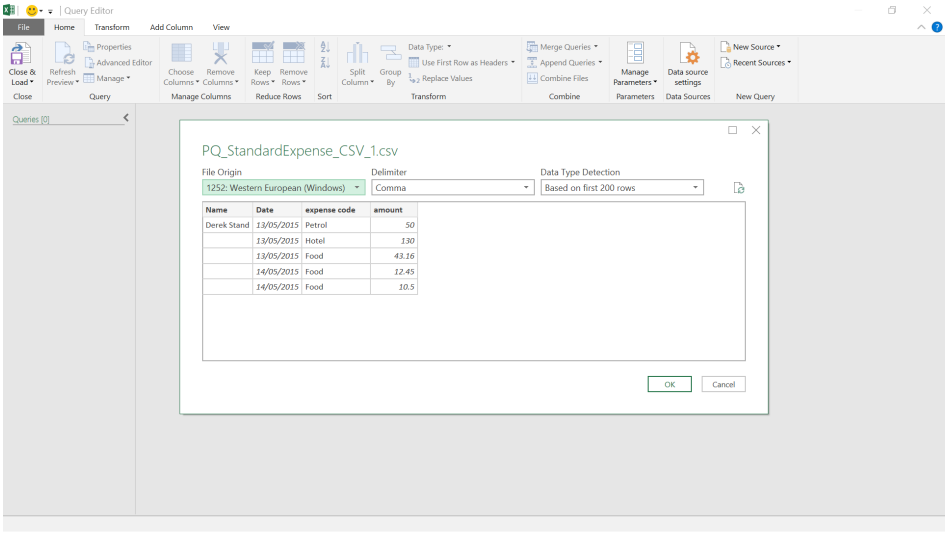
We have an option to call the Power Query Editor, which we may access from the 'New Query' dropdown in the 'Get and Transform' section. The 'Launch Query Editor' option is in the 'Combine' Queries' section (the 'Launch Query Editor' option is also available in Excel 2013 as an icon on the 'POWERQUERY' tab).



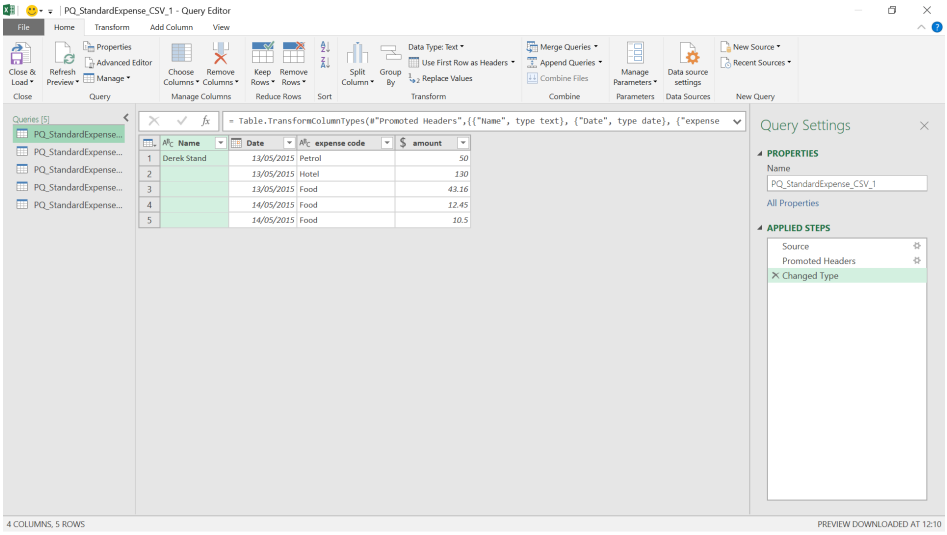
This view is clearly different from the 'Blank Query' option. There is no Formula bar or 'Query Settings' pane. The purpose of this format of the editor is to allow you to create and combine queries, so let's begin by using 'New Source' to locate the relevant file:



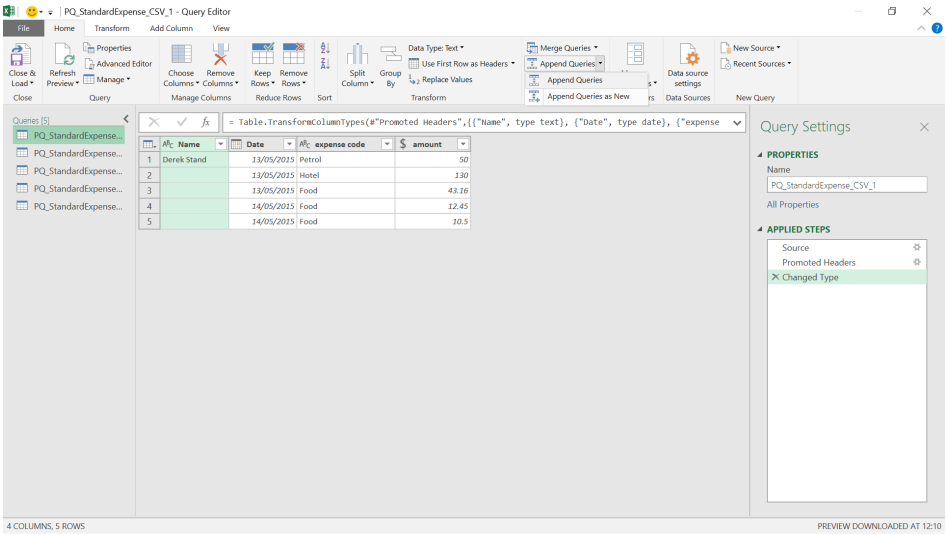
The options are similar to the query creation from the Data tab in the Excel workbook. We'll choose 'Text/CSV' and locate the file to begin with.



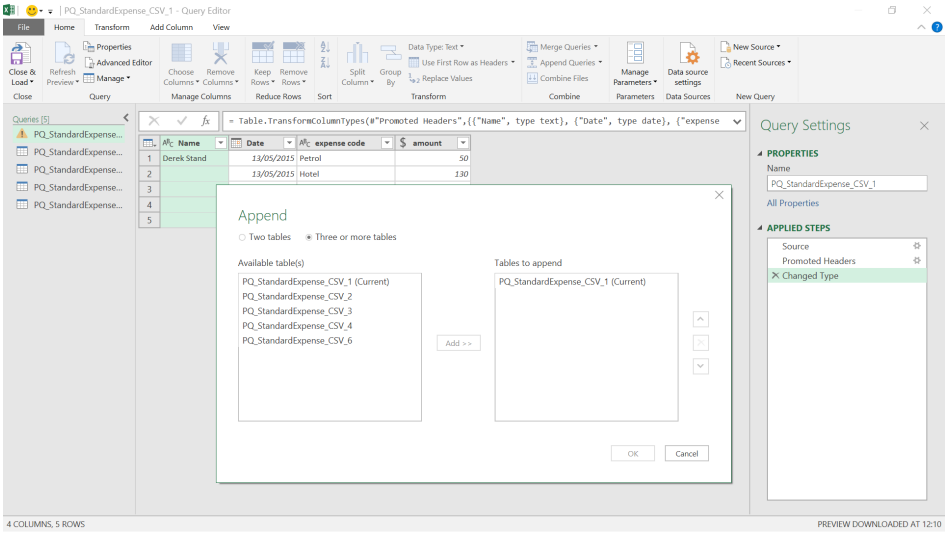
A preview of the file appears and we may click 'OK' to create a query.



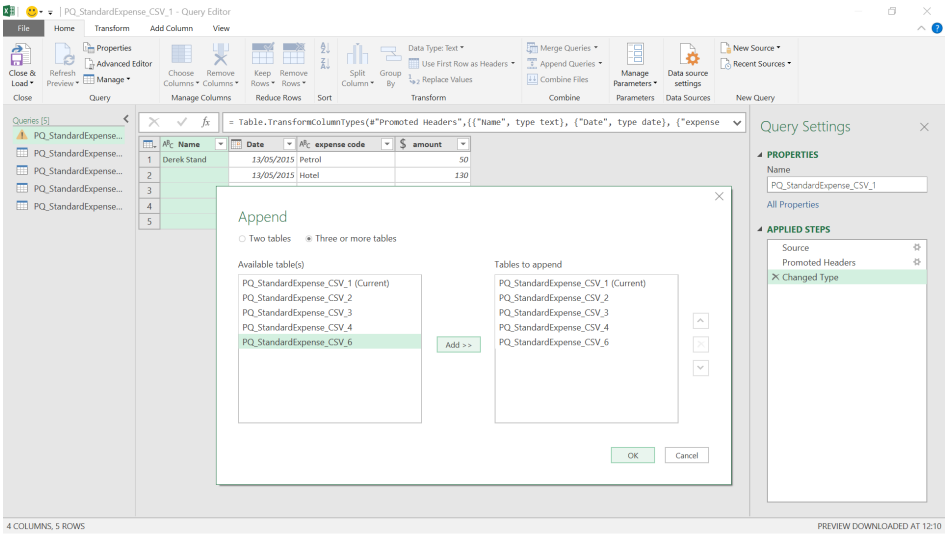
Our file is now extracted. The Formula bar and 'Query Settings' pane now appear (if these do not appear then on the 'File' tab there is a 'Query Options' screen where you may choose your). Let's repeat this process to get queries of five CSV files that we want to combine. Then, choose the 'Append Queries' option.



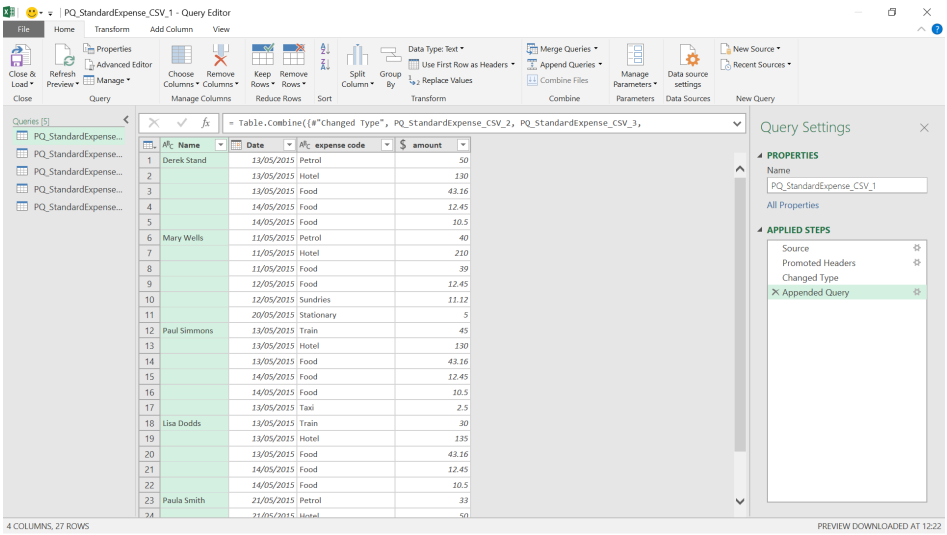
We're happy to add the other data to this query, so let's choose the 'Append Queries' option:



We will opt to append three or more tables, and we may choose from any of our queries – the current query is already included, so let's choose the other four.



Select 'OK' to continue.



All our queries are appended in one step and are ready for us to transform. This is a useful way of quickly appending new data that is extracted from different folders.

More next month!

Power BI Updates

This month's updates see small multiples become Generally Available, the new Model View / sensitivity labels in Desktop are propagated, plus there is a new Preview for streaming dataflows. The full list reads as follows:

Reporting

- Small multiples now Generally Available
- Conditional formatting for more properties
- Power BI's built-in visuals now include the Power Automate visual Preview
- Sensitivity labels in Power BI Desktop Generally Available
- Republish PBIX with option to not override label in destination
- Inherit sensitivity label set to Excel files when importing data into Power BI

Modelling

- New Model View Generally Available
- DirectQuery for Azure Analysis Services & Power BI datasets Updates Preview

Data Connectivity

- Amazon Athena (new connector)
- Databricks (updated connector)
- Dremio (updated connector)
- MariaDB (updated connector)

Service

- Streaming dataflows Preview
- Mandatory label policy Preview
- Custom help link for sensitivity labels
- Datasets hub improvements
- Goals support in lineage view
- Scanner API (Admin REST APIs) enhancements to include dataset tables, columns, measures, DAX expressions and mashup queries

Visualisations

- New visuals
- Charticulator (custom visual version 1.0.5)
- Multiple sparklines
- Acterys Reporting
- PureViz Infographic – from PowerPoint to Power BI
- Dynamic radial bar chart by JTA
- Drill Down Waterfall PRO
- Control Chart XmR by Nova Silva

Template apps

- Analyse your email marketing performance using Mailchimp and ActiveCampaign

Other

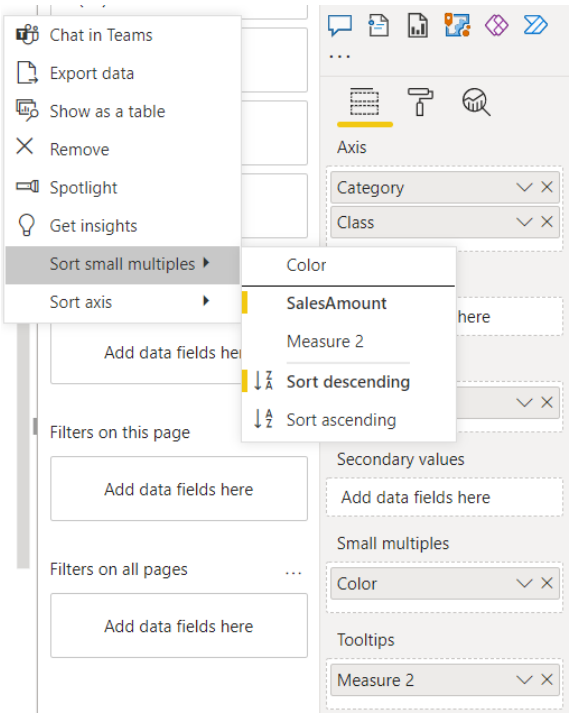
- Power BI Desktop Installer Changes and WebView2.

Let's go through each in turn.

Small multiples now Generally Available

Small multiples are now Generally Available. To celebrate, Microsoft has included two improvements to the feature this month:

1. **Keyboard navigation and screen reader support has been improved for small multiples.** Moving focus around the small multiples grid should now be more consistent and intuitive, and screen reader readouts should be descriptive
2. **Sorting functionality has been introduced for small multiples**, allowing you to sort the order in which they appear by the measures in your field wells. This is useful for cases such as seeing the highest cumulative value small multiple first and will help you make more useful comparisons. This feature is also coming with a slightly new User Interface (UI) in the context menu that will help us scale better to more sortable elements in the future.



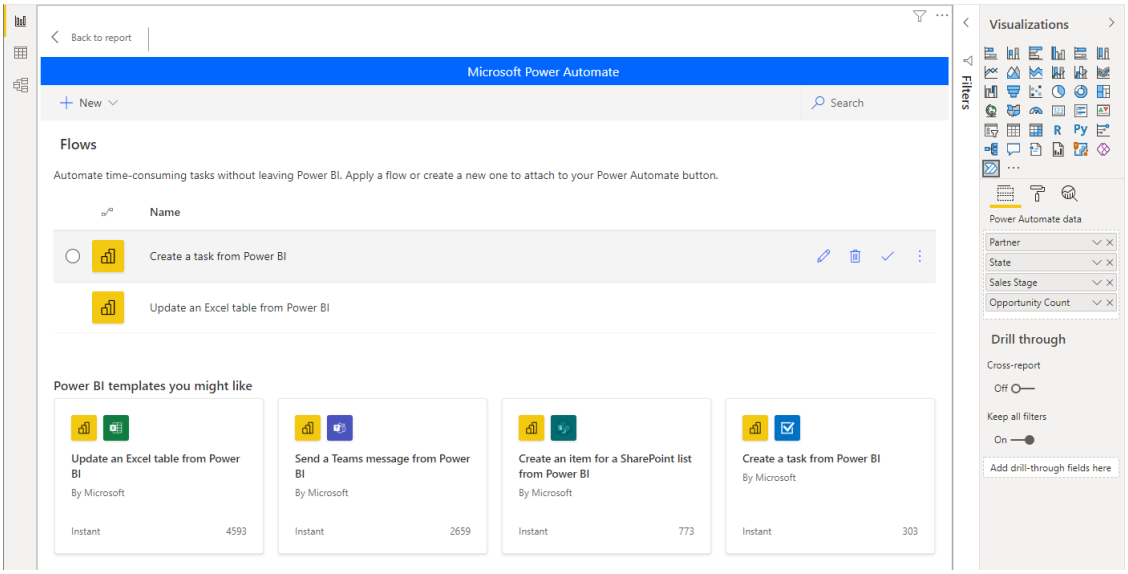
Conditional formatting for more properties

Conditional formatting options have been added to various properties across Power BI's visuals. Now, in addition to all of the properties which already supported conditional format, you will find the **fx** button next to:

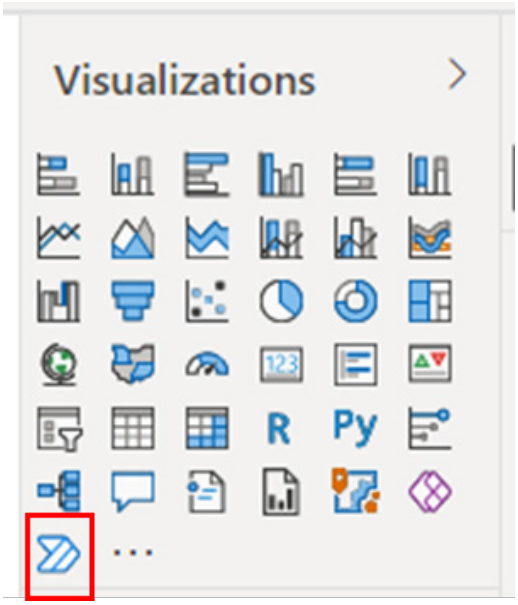
- Data label colours
- Total label colours
- Legend text colours
- Axis start and end
- Axis title, gridline, and label colours
- Funnel visual percent bar label colours
- Funnel visual category axis colour
- Multi-row card title text, data label colours and category label colours
- Gauge visual axis colours, including start, minimum and maximum
- Slicer slider and header font colours.

Power BI's built-in visuals now include the Power Automate visual Preview

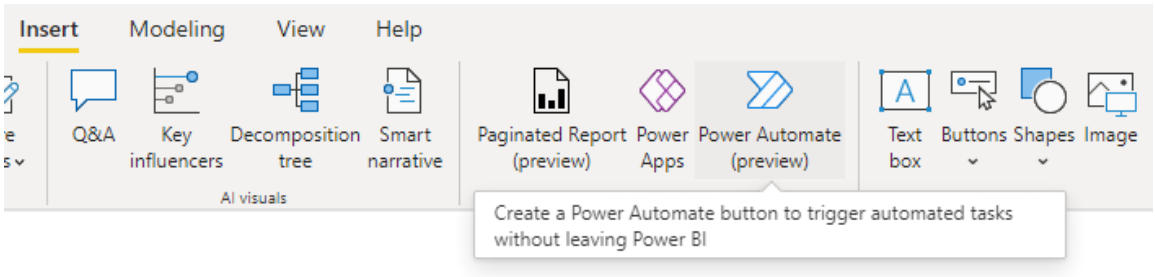
Back in April, Microsoft released a Preview of the Power Automate visual to AppSource. As a refresher, the Power Automate visual allows users go from insights to action without leaving the context of the report. In this month's release, the visual is now available in the Visualization pane by default. Now, with a single click, you can add the Power Automate visual to your reports, without having to download or import the visual.



Within Power BI Desktop or Power BI Service, you can add the visual from the Visualizations pane:

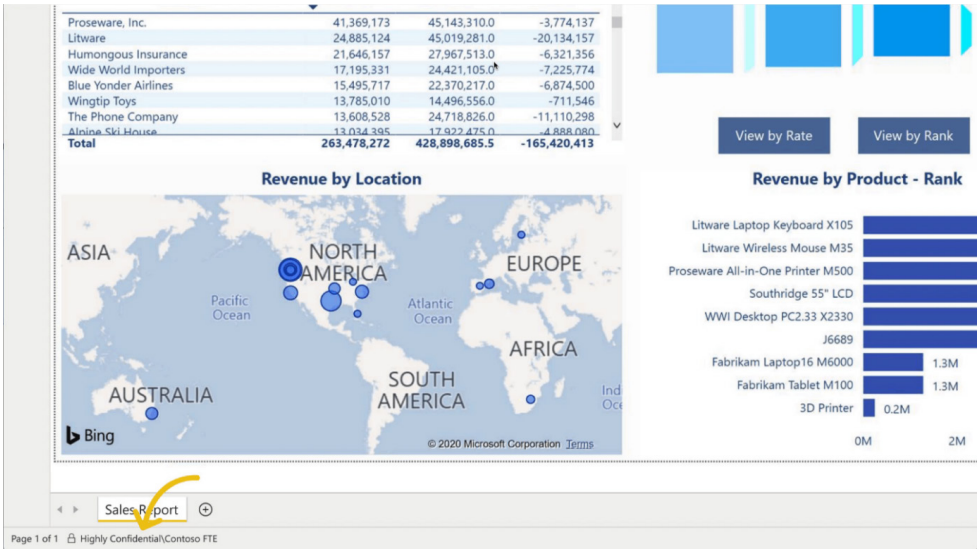


If you're using Power BI Desktop, you can also access the visual from the Ribbon, by navigating to the Insert tab and selecting 'Power Automate (preview)' within the Power Platform section:



Sensitivity labels in Power BI Desktop Generally Available

Microsoft Information Protection (MIP) sensitivity labels are now Generally Available in Power BI Desktop. With MIP labels, creators can label and protect sensitive content in Power BI Desktop.

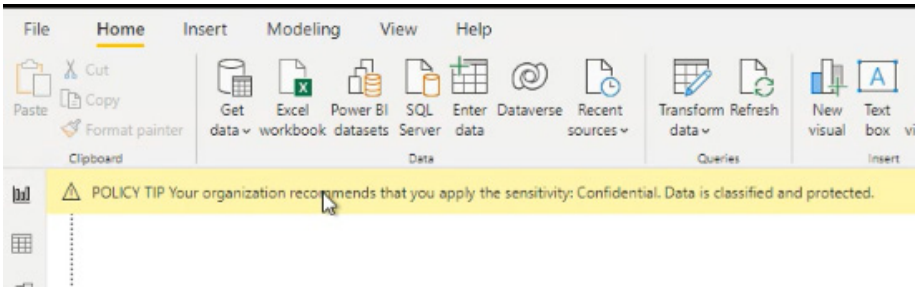


Republish PBIX with option to not override label in destination

With this new update, creators will be able to republish PBIX files without overriding sensitivity labels on destination dataset and / or report. In some cases, the sensitivity label applied on PBIX file doesn't match the sensitivity of the data of target dataset or report in Power BI Service, whether it's because the data sources are different or because the report in the service is applied with different sensitivity label than the one set on the dataset.

Inherit sensitivity label set to Excel files when importing data into Power BI

Many organisations use MIP labels to label and protect Excel files. Starting with this release, when you get data from protected Excel files in Power BI Desktop and in the Power BI Service, Power BI will inherit the MIP sensitivity label from the Excel file and apply it on downstream Power BI datasets and reports.



New Model View Generally Available

The new Model View UI is now Generally Available. Keep in mind that if you are using a report that you haven't already upgraded to the new Model View, it will automatically be upgraded for you now.

DirectQuery for Azure Analysis Services & Power BI datasets Updates Preview

This month's update sees the continuation of added support for more properties set on a remote model to flow to your local model. There are two of note:

1. **The first improvement is related to date tables and auto-generated date tables.** If the remote model you've connected to has date tables or auto-generated date tables, those will now show up in your local model
2. **The second improvement is related to lineage tags.** If your remote model has lineage tags defined, any renames on the remote model will show up in your local model without the renamed objects being deleted and recreated. This also means you won't lose any local customisations on objects that have been renamed.

Amazon Athena (new connector)

Amazon Athena is an interactive query service that analyses data stored in Amazon S3 and other relational, non-relational and custom data sources through federated queries. Using the Amazon Athena connector for Microsoft Power BI, you can now query, analyse, visualise and share insights from your data lake with work colleagues.

Databricks (updated connector)

This update includes improved stability and performance of large imports and fixes an issue with the handling of invalid credentials.

Dremio (updated connector)

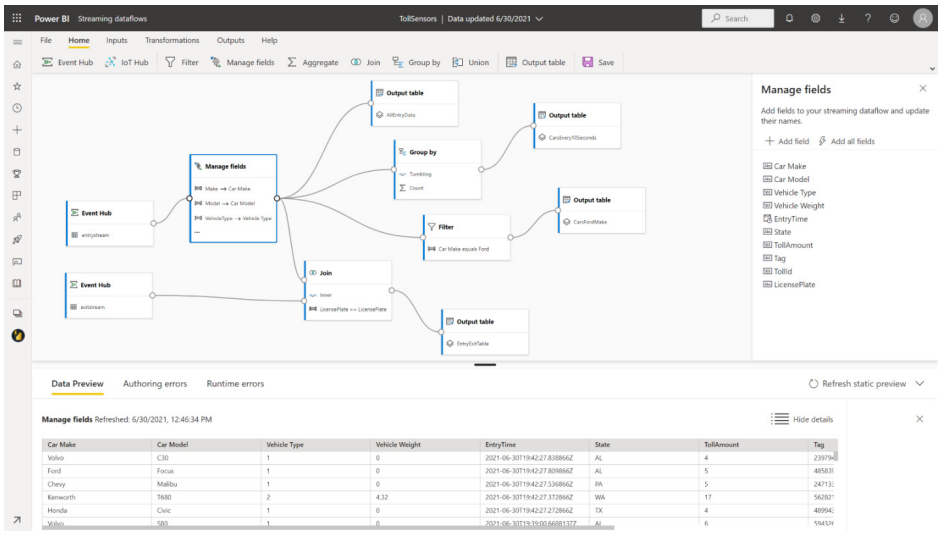
This update fixes an issue where any error would cause Power BI to repeatedly display an 'Encryption Support' prompt.

MariaDB (updated connector)

This update includes a fix for misreported **COLUMN_SIZE** for the **VARCHAR** data type.

Streaming dataflows Preview

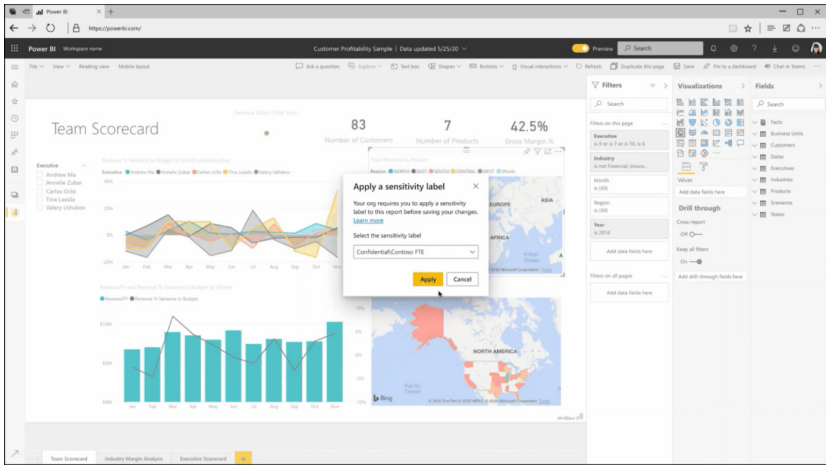
Microsoft is seeking to make the distinction between batch, real-time and streaming data disappear. The intention is for customers to be able to work with all data as soon as it is available. Therefore, streaming dataflows allows every business analyst to work with streaming data using a drag and drop, no-code experiences. This will mean working with streaming data is no longer limited just to data engineers. Users may connect to, prepare and view real-time data to create end-to-end streaming analytics solutions directly in Power BI.



As of today, streaming dataflows allows you to connect to streaming data in Azure (IoT Hub and Event Hub), perform streaming-specific data preparation operations like joins and filters, as well as time windowing aggregations (such as tumbling, hopping and session windows) for group by operations. All of Power BI's data visualisation capabilities will work with streaming data just as they with batch data today. Streaming dataflows is included as part of Power BI Premium, including Premium Per User.

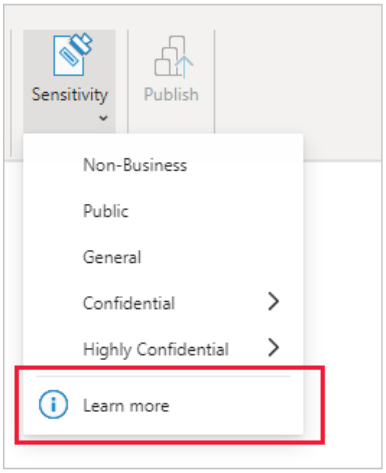
Mandatory label policy Preview

Many organisations need to classify their sensitive information to meet regulatory, compliance or internal audit requirements. Now, with a mandatory label policy, you can require creators to set a sensitivity label when they create new content or edit existing content in Power BI. Creators will be required to apply the sensitivity label when they upload PBIX file to Power BI or when they create new content or edit content without a label in Power BI.



Custom help link for sensitivity labels

To help you and your colleagues understand what your sensitivity labels mean or how they should be used, you can provide a 'Learn more' link pointing to your organisation's custom web page that users will see when they're applying or being prompted to apply sensitivity labels.



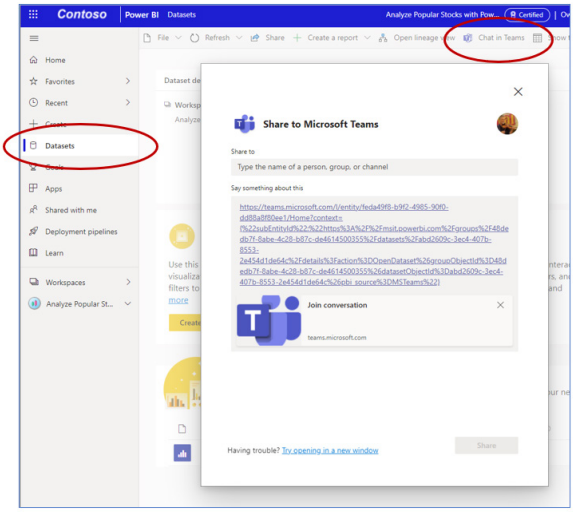
Datasets hub improvements

NEW SHARING CAPABILITIES

Microsoft has added new sharing capabilities to the Datasets hub to make it easier for you to share your content.

Chat in Teams

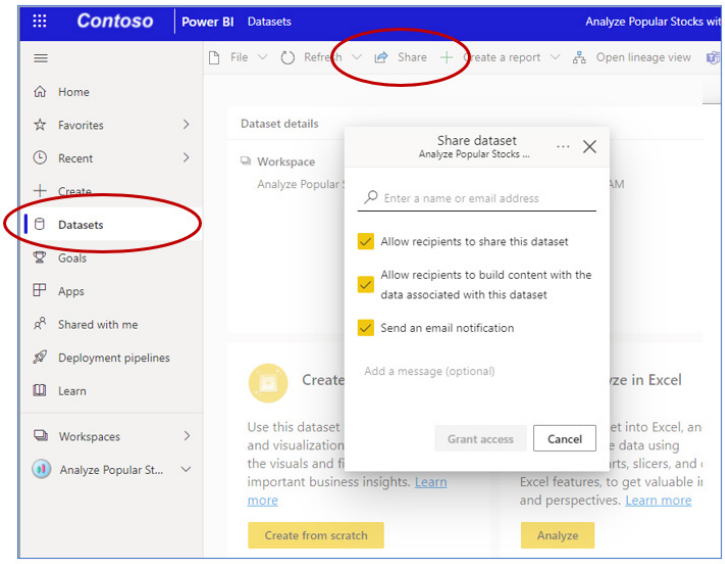
You may use the Chat in Teams feature to quickly start conversations when you're viewing your dataset and want to share it with others.



Only users with a Power BI license and relevant dataset permissions will be able to click the 'Chat in Teams' link. If you want to share content with users who don't have access to the dataset, you'll need to use the new Share capability instead.

Share

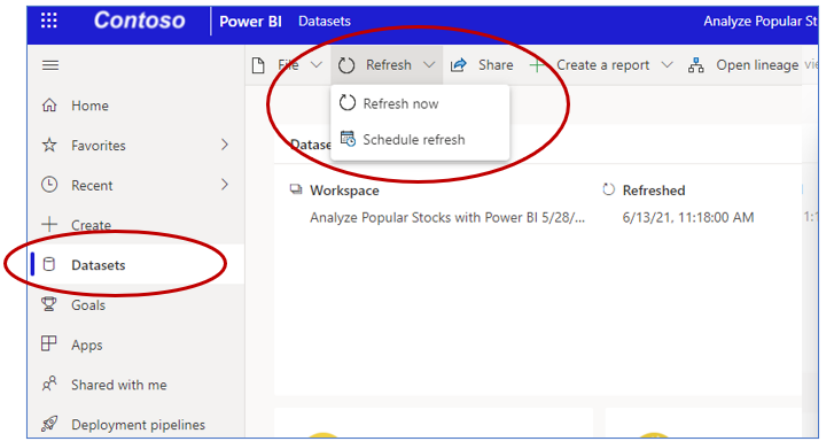
The new Share feature allows you to share your datasets with others and to provide the relevant access permissions. You can provide different types of access for different users, as well as send them a notification email.



You may also use the 'Manage permission' page (available via Dataset settings) to edit and / or remove dataset access from particular users.

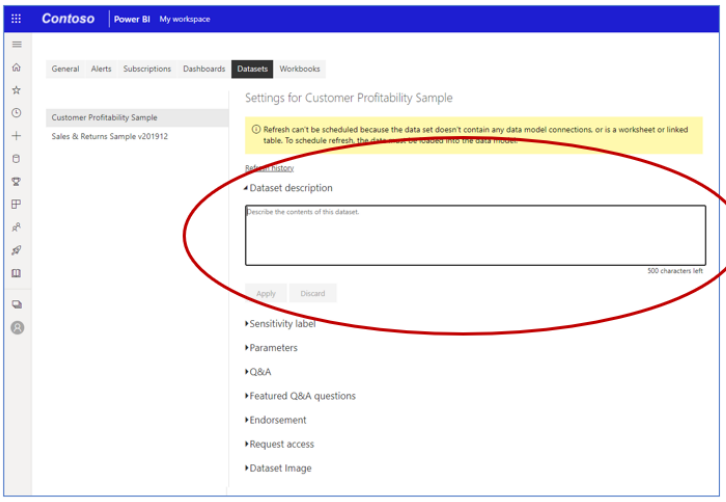
REFRESH AND SCHEDULE REFRESH

The new Share feature allows you to share your datasets with others and to provide the relevant access permissions. You can provide different types of access for different users, as well as send them a notification email.



DATASET DESCRIPTION

This update sees the dataset description field moved to a new Dataset description section in the Dataset settings, to make it easier for Dataset owners to find and fill in. The Dataset description is very important, and helps other analysts in the organisation see whether the dataset answers their business needs so that they can start using it.

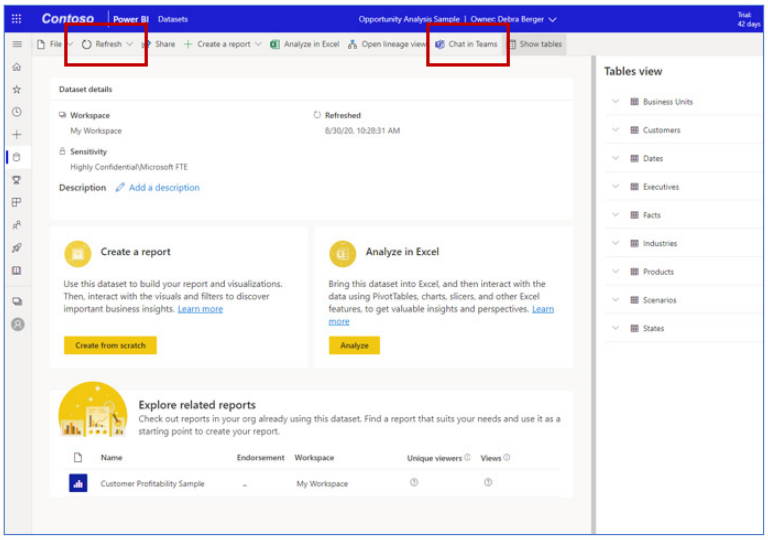


GOALS

With this release, Goals has been added to the list of dataset-related reports. When a new scorecard is created, it also generates a dataset that holds the scorecard data. You can find this dataset in the Datasets hub, and will see the scorecard among the other related reports in the dataset's related reports list.

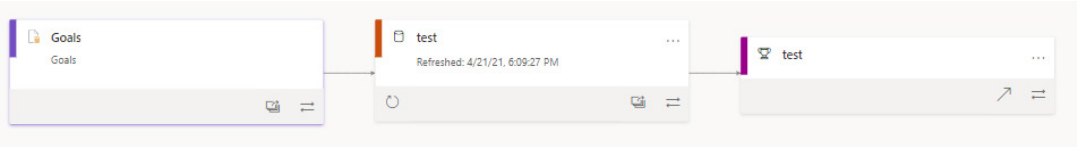
ALL ACTIONS ARE AVAILABLE IN THE DATASET HUB LANDING PAGE AND IN THE DATASET PAGE

When you click on a specific dataset, in the Datasets hub or in the workspace content list, you get to that dataset's Dataset page. On the Dataset page you see information about the dataset, actions you can perform and a list of reports that are built on top of the dataset (only reports you have access to are listed).



Goals support in lineage view

When you open lineage view, you can now see the workspace's scorecards and their connections.

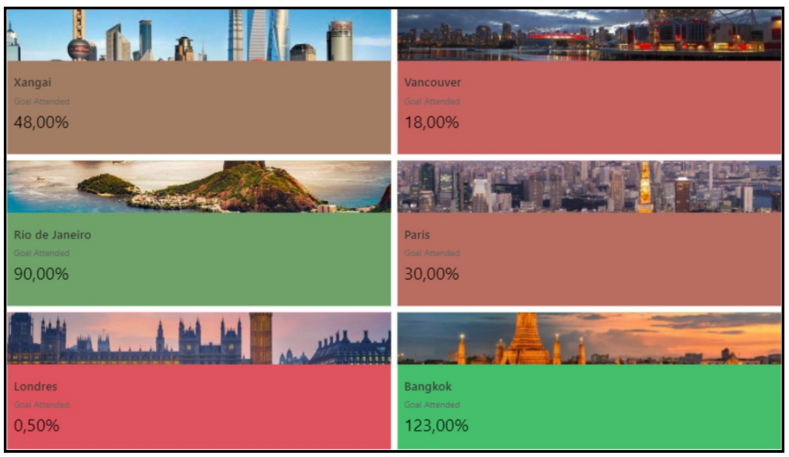


Scanner API (Admin REST APIs) enhancements to include dataset tables, columns, measures, DAX expressions and mashup queries

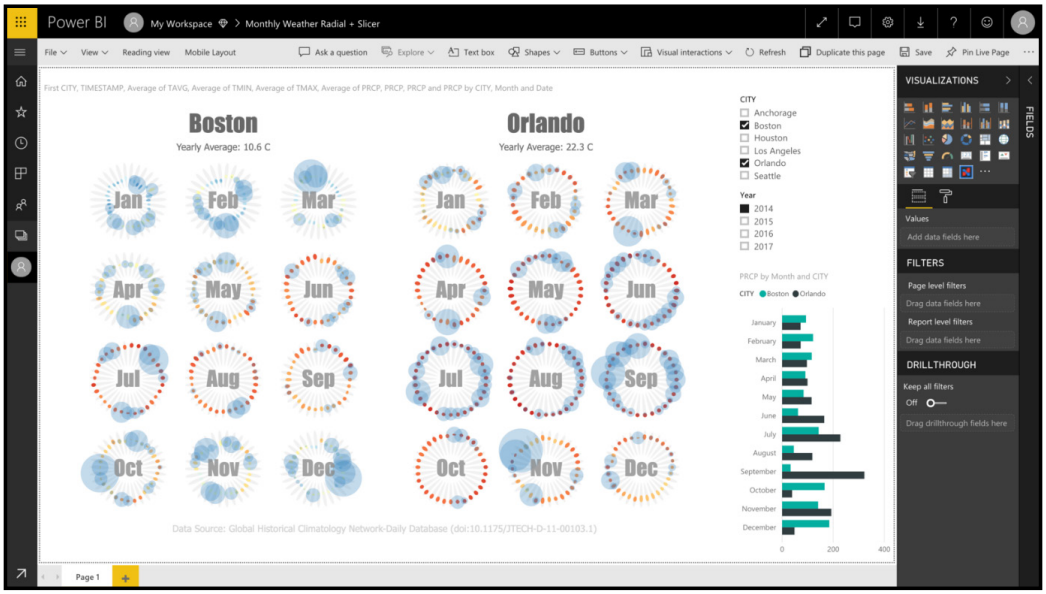
Towards the end of last year, Microsoft announced new Scanner APIs for extracting tenant-level metadata using Admin REST APIs. Many users are now using the new Scanner APIs to query Power BI, in order to build their own reporting and homegrown catalogues. The APIs have been constructed in such a way that they support scaling for big tenants and at the same time allow Power BI to add more metadata to the API response as time goes along. Now, as part of the API response, you can get a dataset's tables, columns, measures, DAX expressions and mashup queries.

New visuals

This month sees the advent of Multi Info Cards. This visual will generate multiple cards for a categorical column, and you can show up to eight measures and an optional image. It also supports filtering and highlighting, ToolTips (both default and page ToolTip), conditional formatting and many other customisation features.



Charticulator (custom visual version 1.0.5)



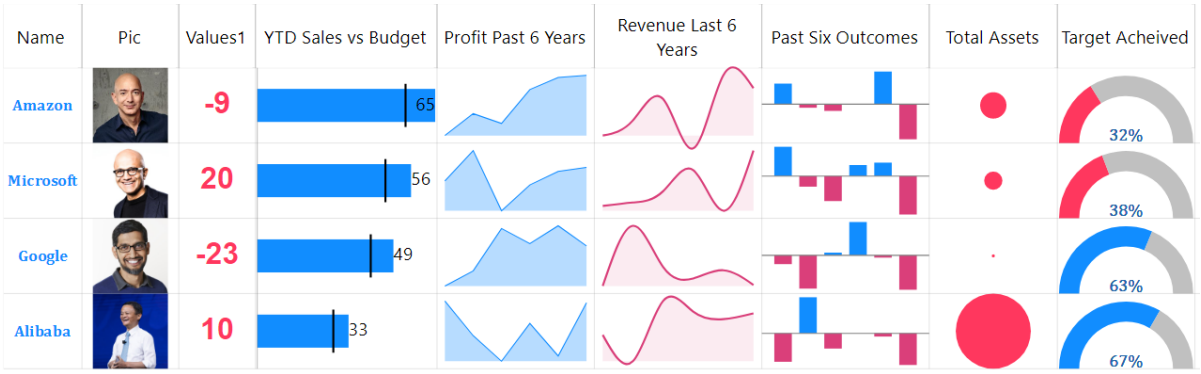
With Charticulator version 1.0.5, a number of updates and bugfixes have been made in order to expand the visual's functionality and address issues previously identified:

- set the default values for font (font family: Segoe UI; font size: 12) for text and text box
- set the default chart properties (size: 600×400; margins: 20, 20, 20, 20) and add them to the visuals' Format panel
- set the default colour (Power BI's default blue) for shapes and symbols
- enable the update of colours, based upon the report theme; users may turn this on / off from the Color *(sic)* settings property in the visuals' Format panel
- enable the setting of default panels position in the visuals' Format panel
- save the order for the categorical axis in the template
- save chart changes history on closing editor
- bind links to 'glyph center' if no marks with anchor
- scroll to the attribute field in the Attributes panel on opening a scale from the Scales panel
- for the Date data type, display date string (*e.g.* 06/08/2021) instead of a number in the scale editor
- set the default aggregation function to 'first' for categorical kinds (even if they are of the Number data type)
- fixed a number of bugs and issues, including:
    - save wasn't working in certain cases
    - labelling of the automatic domain inference, and moved the UI location in the Scale Editor
    - attribute mapping on drag guide in the 'glyph editor'
    - pop-up view opens at the top left corner
    - converting empty values from Power BI
    - updating data kind and data type.

Multiple sparklines

It is difficult to see trends or patterns when you are presented with a table full of rows and columns of numbers. This Power BI Custom Visual transforms your Table numbers into easier to read charts to give more insights into your data. The visual supports the following chart types:

- Line Chart / Area chart
- Column chart
- Bubble chart
- Donut chart
- Bullet/ Bar chart
- Normal values (Text, Numbers, Image URLs, Web URLs, Unicodes, *etc.*)



You can sort by any column by clicking on header row, rearrange charts / columns, drag rows and columns to resize, add data labels to charts, apply different formats and styles *etc*.

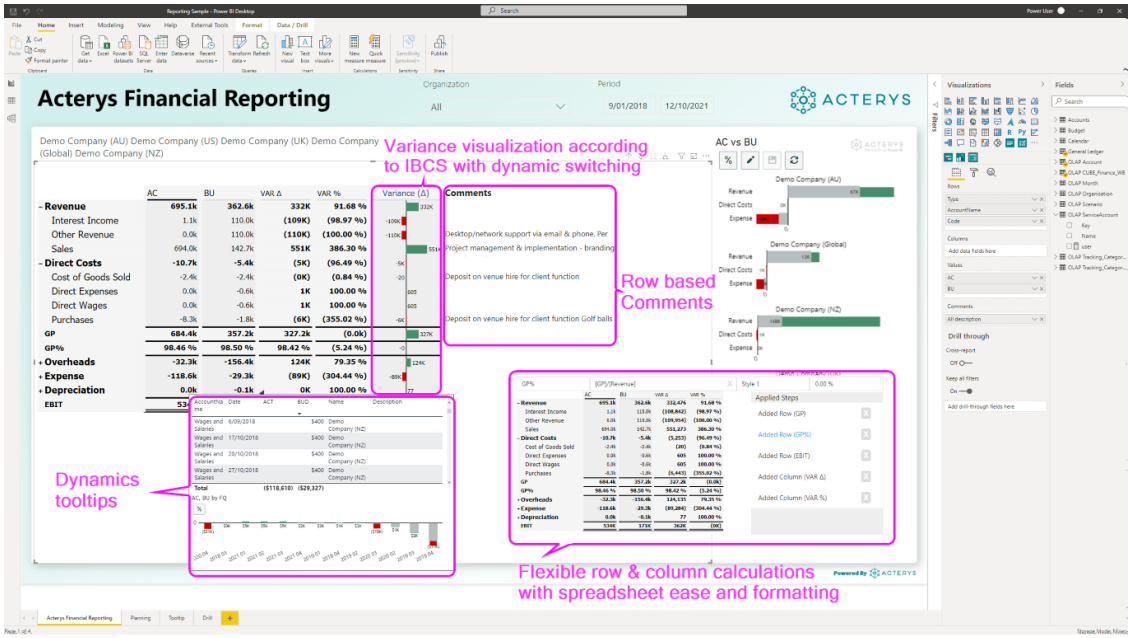
This chart may be downloaded from AppSource.

Acterys Reporting

Acterys Reporting is the latest addition to the Acterys integrated planning and performance management framework for Power BI. Here, users may create financial reports with spreadsheet flexibility without the need for DAX:

- adding row and column calculations
- applying typical financial reporting formatting requirements including visualisations according to IBCS principles
- adding row-based comments.

In addition to the reporting functionality, the visual supports all planning, writeback and data modelling functionality included in the seven other Acterys visuals listed on AppSource, the Acterys Modeller and the Power BI Sync external tool too.



PureViz Infographic – from PowerPoint to Power BI

PureViz Infographic is a new Power BI visual that lets business users to create their own visuals using PowerPoint and bring their designs to life in Power BI very easily. Users can format shapes, set text fields and animate any layer in their design using existing measures.

You may export your design as .SVG image and then select this file in Power BI. Then, you can use the following features of PureViz Infographic to finalise your visual:

- **shape formatting:** fill colour, gradient fill, size / position, visibility, rotation, scale, *etc.*
- **text formatting:** text content, font size, text position, visibility, rotation, *etc.*
- **conditional formatting:** binding data fields to properties, using if / else
- **maths and number formatting:** visual level calculations without new measures
- **shape transform animations:** rotation, position, colour / opacity, scale
- **text transform animations:** typing animation, count down / up
- **click and hover actions:** hyperlinks, ToolTips
- **advanced colour fill:** directional shape fill, dynamic fill, gauge fill
- **advanced formulae:** using nested brackets and arithmetic operators.

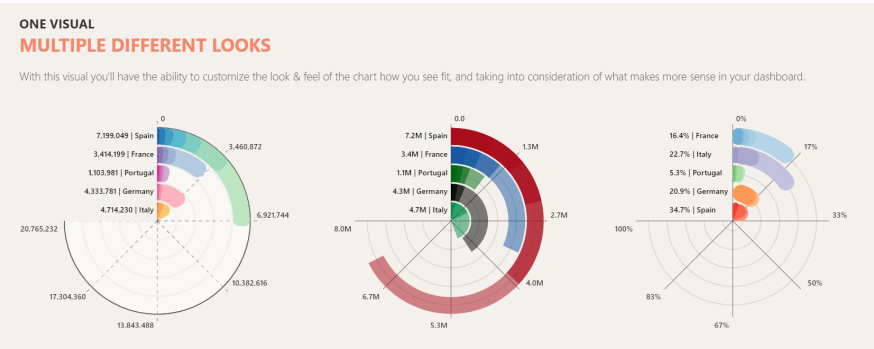
Dynamic radial bar chart by JTA

This chart effectively combines two of the most common charts in the data visualisation field, the bar chart and the radial chart. With multiple configurations available, the visual will allow you to navigate and quickly understand your data.

With a range of functionalities and customisable options, you will be able to drill-down your information to see it in detail, define global or individual targets by category to compare it with your data, choose between multiple fields when drilling-down and more.

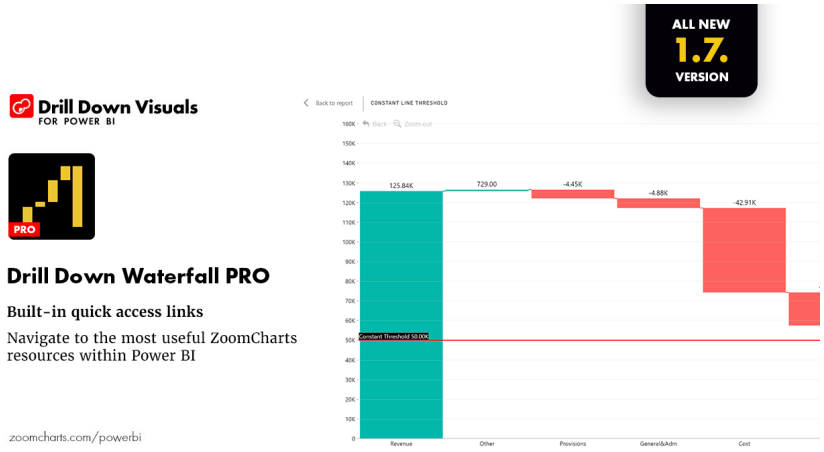
With the latest version, these are some of the key features you will find:

- updated logic to define target values or load them automatically as a field
- added options to choose whether you want your target to be displayed on a label or not
- improved ToolTip customisation, using data fields of your choice
- improved the total value calculation and relative totals, with the option to add only some categories to the grand total
- updated formatting options and default values for settings such as radial bars curviness, background and shadow colours, multiple choices of gradual colour schemes, *etc*.
- standard Power BI fonts added
- updated visual API version, to comply with the latest Power BI recommendations.



This may be downloaded from AppSource.

Drill Down Waterfall PRO



With this visualisation you may control the column sequence, add subtotals and drill down into each category. Other features include"

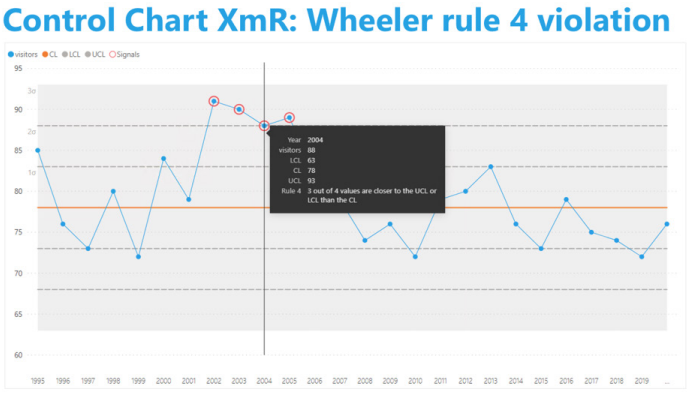
- **touch-driven slicer:** filter the report page by using the visual itself (no need for external slicers)
- **on-chart interactions:** zoom, click and drag or drill down to explore and filter data
- **display of total(s):** turn on or off the Total column
- **sub-totals:** set display values in your dataset or let the visual calculate them automatically
- **various customisation options:** customise increasing, decreasing and totals series separately (colours, outlines, column widths, connectors, value labels and more)
- **static and dynamic thresholds:** set up to four [4] thresholds to demonstrate targets or benchmarks
- **mobile friendly:** this may be used on touch and multi-touch devices.

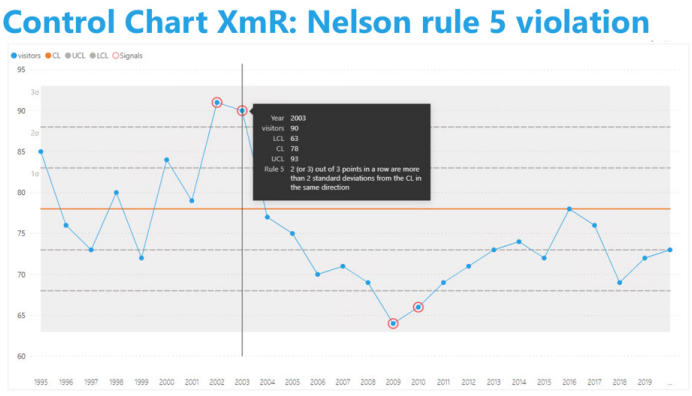
You may download the Drill Down Combo Bar PRO in AppSource.

Control Chart XmR by Nova Silva

Control charts allow you to split your temporal data in two: random noise and real signals.

Stacey Barr originally applied Control charts based upon the so-called Wheeler rules (as specified by Dr. Donald J. Wheeler), now these also may be based upon the Nelson rules too.





Again, this may be downloaded from AppSource.

Analyse your email marketing performance using Mailchimp and ActiveCampaign

You can connect to your Mailchimp or ActiveCampaign email marketing services to Power BI and analyse the performance of your campaigns and audiences.

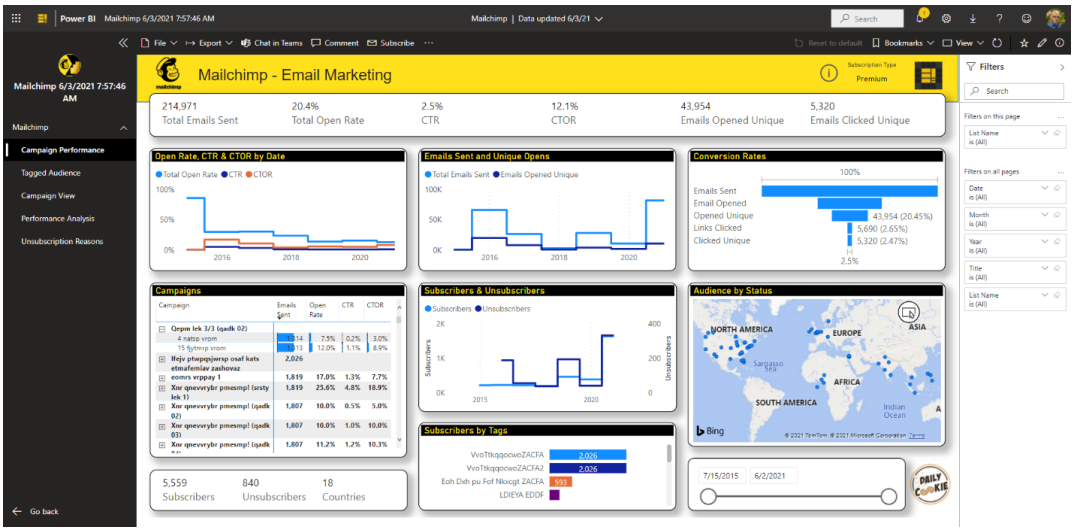
DataChant has teamed up with DailyCookie to release two Power BI apps on AppSource to get you to the "next level in understanding your email campaigns" on Mailchimp and ActiveCampaign.

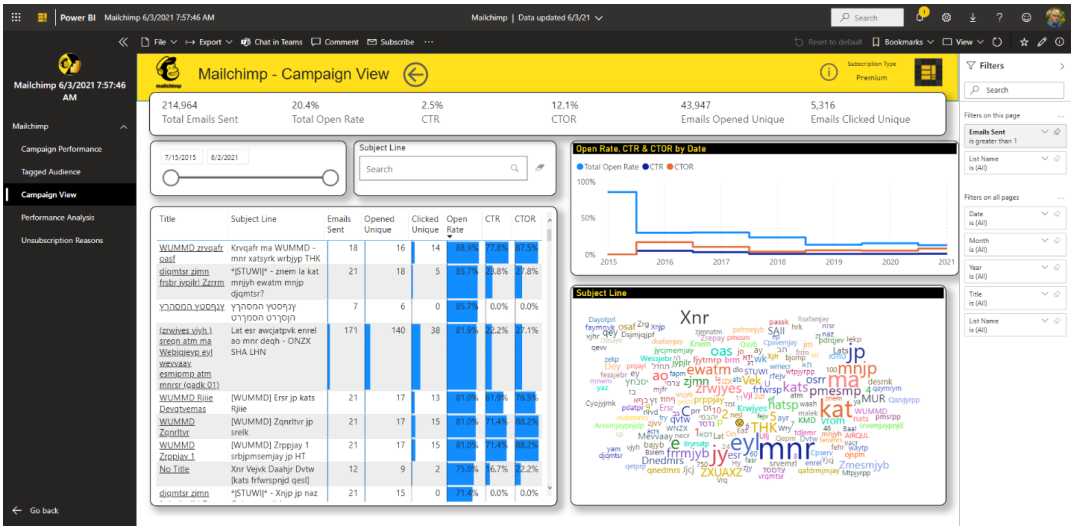
MAILCHIMP EMAIL MARKETING

Mailchimp Email Marketing is a Power BI app that connects to your Mailchimp account and analyses the performance of your campaigns. The app requires an active Mailchimp account and API Access Key to refresh automatically. You can explore the various pages on the left sidebar to analyse different aspects of your campaign, including a campaign view, tagged audiences, performance analysis and reasons for unsubscribing. The app enables interactive view to email marketing KPIs such as open rates, CTR (Clickthrough Rate) and CTOR (Click-to-Open Rate).

The basic version of this app is free. It allows you to import your last six months of recently created campaigns, up to 500 members per list, and up to five unsubscribers per campaign. To lift these limitations, you can subscribe for the Premium version, which allows you to import all of your historical data.

With both free and premium versions, the report layout may be edited, and the data can be imported or connected live via Excel and Power BI to build your own reporting tools from your Mailchimp data.

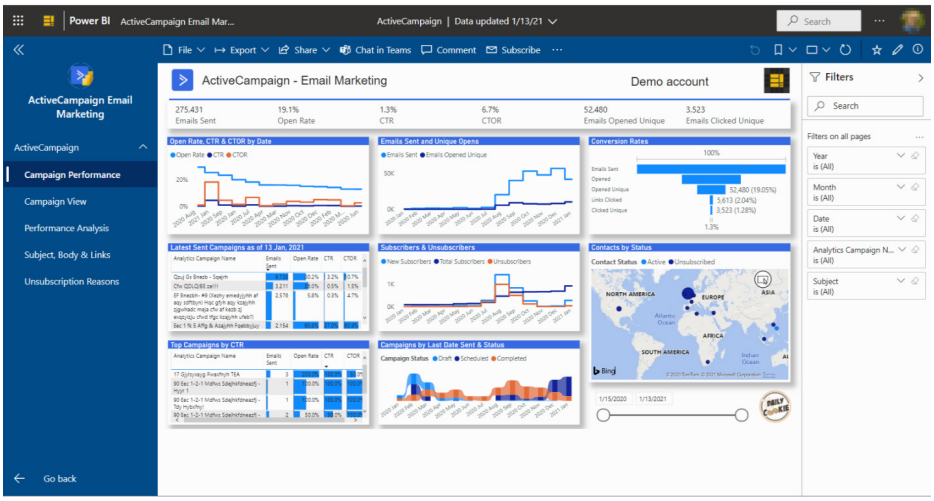


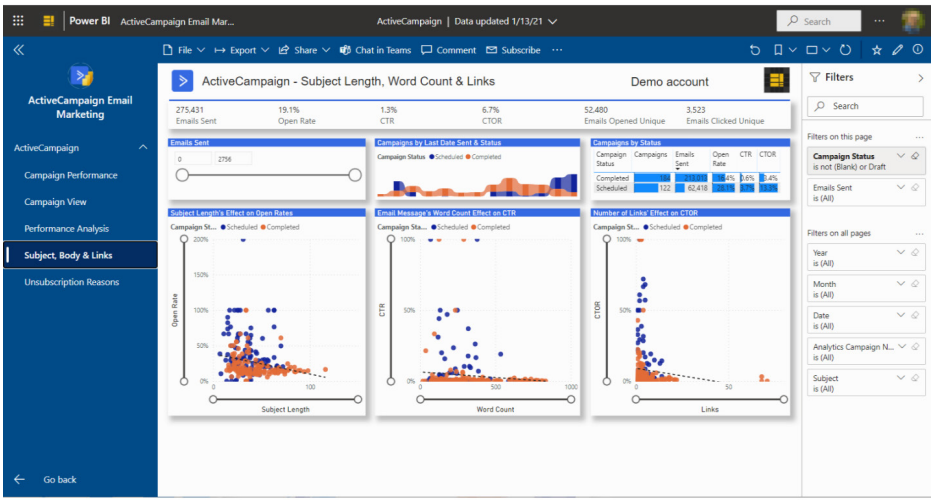


ACTIVECAMPAIGN EMAIL MARKETING

ActiveCampaign Email Marketing is a Power BI app that connects to your ActiveCampaign account and analyses the performance of your campaigns. The app requires an active ActiveCampaign account and API Access Key to refresh automatically. Like the MailChimp app, you can explore the various pages on the left sidebar to analyse different aspects of your campaign, including a campaign view, tagged audiences, performance analysis and unsubscription reasons. The app again enables interactive view to crucial email marketing KPIs such as open rates, CTR (Clickthrough Rate) and CTOR (Click-to-Open Rate).

This app was recognised by ActiveCampaign as an official app, and is featured on their 'Apps & Integration' portal.





Power BI Desktop Installer Changes and WebView2

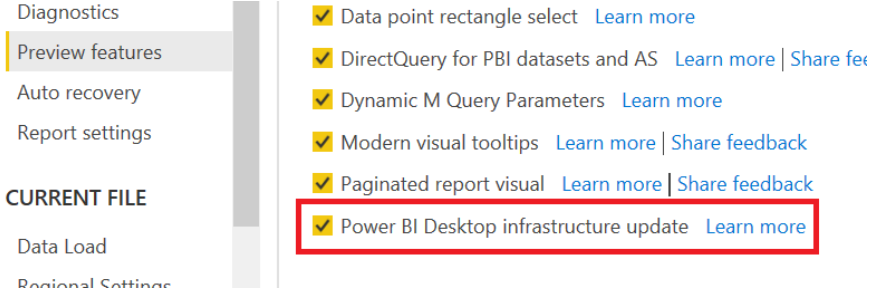
Starting this month, the Power BI Desktop .exe installer will now attempt to download Microsoft Edge WebView2 as part of the installation process. But don't worry. WebView2 is not a requirement at the moment, so if you're installing offline, you should not notice any changes. If WebView2 fails to download or install for any reason, everything will continue to work, and you will notice no difference during the install process or while using Power BI Desktop.

This is Microsoft's first step in transitioning part of their infrastructure from using CefSharp to WebView2. They are making this switch to assist the development and release process (which means they should be able to spend more time developing new features). It also means that you'll automatically get the latest security patches as the WebView2 team ships them instead of waiting for Microsoft to update Power BI Desktop.

As stated above, WebView2 is not currently required, but it's important to know that at some point in the future it **will** be. When that happens, you will be required to fully install and run Power BI Desktop. At that time, if you don't have WebView2 installed, you'll need to be connected to the internet when you're first installing through the .exe, so it can be downloaded automatically. For organisations installing Power BI Desktop for your employees, you'll also need to install WebView2, if it's not already installed.

With all this borne in mind, Microsoft has stated that they do not have a date set for this transition yet, and that they will give plenty of advance warning before it does become necessary. We shall see!

If you'd like to start using Power BI Desktop with WebView2 instead of CefSharp right away, you can enable it through the Power BI Desktop infrastructure update Preview feature. The option will only be visible if you have WebView2 installed. Once you've turned it on and restarted Power BI Desktop, you'll automatically start using WebView2. You shouldn't notice any changes between the two experiences, apparently, but no doubt time will tell.



More next month, we're sure!

The A to Z of Excel Functions: GAMMA

The Gamma distribution is widely used in engineering, science and business, to model continuous variables that are always positive and have skewed distributions. The Gamma distribution can be useful for any variable which is always positive, such as cohesion or shear strength for example.

It is a distribution that arises naturally in processes for which the waiting times between events are relevant, and is often thought of as a waiting time between Poisson distributed events (the Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval).

To understand it, first think of factorials, *e.g.* 5! = 5 x 4 x 3 x 2 x 1 = 120. So far, so good, but how do you calculate if the factorial number you want to evaluate isn't an integer? The Gamma function is used to calculate this:

Γ(N+1) = N * Γ(N)

That's great (if a little recursive), so can be expressed better (!) mathematically as follows:

$$\Gamma(N) = \int_0^{\infty} t^{N-1} e^{-t} dt$$

Clear as mud? Well, it gets better. The Gamma distribution just referred to has the following probability density function:

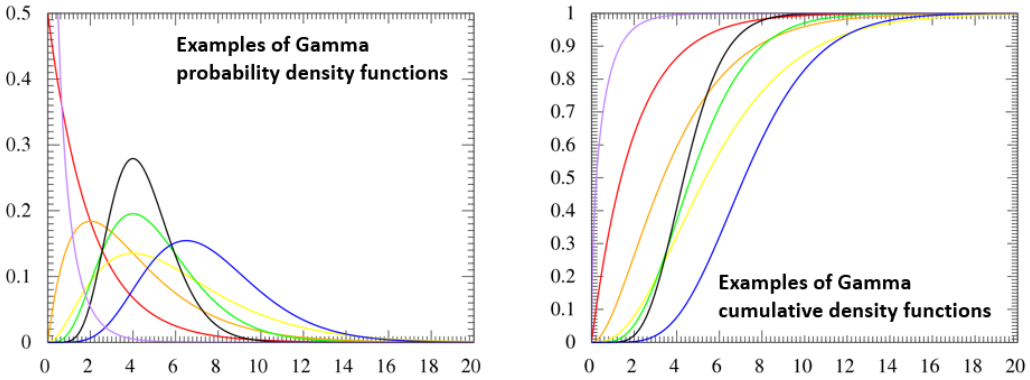
$$f(x;\alpha,\beta) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}$$

where **Γ(α)** is the Gamma function, and the parameters **α** and **β** are both positive, *i.e.* **α** > 0 and **β** > 0:

- **α** is known as the shape parameter, while **β** is referred to as the scale parameter
- **β** has the effect of stretching or compressing the range of the Gamma distribution. A Gamma distribution with **β** = 1 is known as the standard Gamma distribution.

The Gamma distribution represents a family of shapes. As suggested by its name, α controls the shape of the family of distributions. The fundamental shapes are characterized by the following values of **α**:

- Case I (**α** < 1): when **α** < 1, the Gamma distribution is exponentially shaped and asymptotic to both the vertical and horizontal axes
- Case II (**α** = 1): the Gamma distribution with shape parameter **α** = 1 and scale parameter **β** is the same as an exponential distribution of scale parameter (or mean) **β**
- Case III (**α** > 1): when **α** is greater than one, the Gamma distribution assumes a mounded (unimodal), but skewed shape. The skewness reduces as the value of **α** increases.



The **GAMMA** function here calculates the above function, **Γ(N)**, when **β** equals one (1):

GAMMA(number)

The **GAMMA** function has the following argument:

- **number:** this is required and represents the **number** to be evaluated.

It should be noted that:

- if **number** is either negative integer or zero (0), **GAMMA** returns the *#NUM!* error value (arguably, if it is zero, the value should actually be zero!)
- if **number** contains characters that are not valid, **GAMMA** returns the *#VALUE!* error value.

Please see our example below:

	A	B	C
1	Formula	Description	Result
2	=GAMMA(2.5)	Returns the gamma function value of 2.5.	1.3293
3	=GAMMA(-3.75)	Returns the gamma function value of -3.75.	0.2679
4	=GAMMA(0)	Returns the *#NUM!* error, because zero (0) is not a valid argument.	#NUM!
5	=GAMMA(-3)	Returns the *#NUM!* error, because a negative integer is not a valid argument.	#NUM!

The A to Z of Excel Functions: GAMMA.DIST

Following on from above, the **GAMMA.DIST** function has the following syntax

GAMMA.DIST(x, alpha, beta, cumulative)

The **GAMMA.DIST** function has the following arguments:

- **x:** this is required and this represents the value at which you want to evaluate the distribution
- **alpha:** this is also required. This is a parameter (the shape parameter) to the distribution
- **beta:** this too is required and is another parameter (the scale parameter) to the distribution. If **beta** = 1, **GAMMA.DIST** returns the standard gamma distribution
- **cumulative:** this final argument is required and is a logical value that determines the form of the function. If **cumulative** is TRUE, **GAMMA.DIST** returns the cumulative distribution function; if FALSE, it returns the probability density function.

It should be noted that:

- if **x, alpha** or **beta** is nonnumeric, **GAMMA.DIST** returns the *#VALUE!* error value
- if **x** < 0, if **alpha** ≤ 0 or if **beta** ≤ 0, **GAMMA.DIST** returns the *#NUM!* error value
- the equation for the gamma probability density function is:

$$f(x;\alpha,\beta) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}$$

- the standard gamma probability density function is:

$$f(x;\alpha) = \frac{x^{\alpha-1} e^{-x}}{\Gamma(\alpha)}$$

- when **alpha** = 1, **GAMMA.DIST** returns the exponential distribution with:

$$\lambda = \frac{1}{\beta}$$

- for a positive integer **n**, when **alpha = n**/2, **beta** = 2, and **cumulative** = TRUE, **GAMMA.DIST** returns (1 - **CHISQ.DIST.RT(x)**) with **n** degrees of freedom
- when **alpha** is a positive integer, **GAMMA.DIST** is also known as the **Erlang distribution**.

Please see our example below:

	A	B	C	D
1	Description	Scenario 1	Scenario 2	
2	Value at which you want to evaluate the distribution	9.975	9.975	
3	Alpha parameter to the distribution	9	9	
4	Beta parameter to the distribution	2	2	
5	Cumulative value	FALSE	TRUE	
6				
7				
8	Formula	Description	Result	
9	=GAMMA.DIST(B2,B3,B4,B5)	Probability density	0.032394	
10	=GAMMA.DIST(C2,C3,C4,C5)	Cumulative distribution	0.067281	
11				

The A to Z of Excel Functions: GAMMA.INV

The **GAMMA.INV** function is the inverse of the Gamma cumulative distribution, *i.e.* if **p = GAMMA.DIST(x, α, β)** then **x = GAMMA.INV(p, α, β)**. This can help to identify key points in a variable whose distribution may be skewed. It has the following syntax:

GAMMA.INV(probability, alpha, beta)

The **GAMMA.INV** function has the following arguments:

- **probability:** this is required and this represents the probability associated with the distribution
- **alpha:** this is also required. This is a parameter (the shape parameter) to the distribution
- **beta:** this too is required and is another parameter (the scale parameter) to the distribution. If **beta** = 1, **GAMMA.DIST** returns the standard gamma distribution.

It should be noted that:

- if any argument is text, **GAMMA.INV** returns the *#VALUE!* error value
- if **probability** < 0, if **probability** > 1, if **alpha** ≤ 0 or if **beta** ≤ 0, **GAMMA.INV** returns the *#NUM!* error value.

Given a value for **probability**, **GAMMA.INV** seeks that value **x** such that **GAMMA.DIST(x, alpha, beta, TRUE) = probability**. Thus, the precision of **GAMMA.INV** depends upon the precision of **GAMMA.DIST**. Therefore, **GAMMA.INV** uses an iterative search technique. If the search has not converged after 64 iterations, the function returns the *#N/A* error value.

Please see our example below:

	A	B	C	D
1	Description	Values		
2	Probability associated with the distribution	0.067281		
3	Alpha parameter to the distribution	9		
4	Beta parameter to the distribution	2		
5				
6				
7	Formula	Description		Result
8	=GAMMA.INV(B2,B3,B4)	Inverse of the Gamma cumulative distribution for the given probability		9.975

The A to Z of Excel Functions: GAMMADIST

The **GAMMADIST** function has the following syntax

GAMMADIST(x, alpha, beta, cumulative)

It's important to note that this function has been replaced with the above new function (**GAMMA.DIST**) that may provide improved accuracy and whose name better reflects its usage and consistency with other programming languages. Although this function is still available for backward compatibility, you should consider using the new functions from now on, because this function may not be available in future versions of Excel.

The **GAMMADIST** function has the following arguments:

- **x:** this is required and this represents the value at which you want to evaluate the distribution
- **alpha:** this is also required. This is a parameter (the shape parameter) to the distribution
- **beta:** this too is required and is another parameter (the scale parameter) to the distribution. If beta = 1, **GAMMADIST** returns the standard gamma distribution
- **cumulative:** this final argument is required and is a logical value that determines the form of the function. If **cumulative** is TRUE, **GAMMADIST** returns the cumulative distribution function; if FALSE, it returns the probability density function.

It should be noted that:

- if **x**, **alpha** or **beta** is nonnumeric, **GAMMADIST** returns the *#VALUE!* error value
- if **x** < 0, if **alpha** ≤ 0 or if **beta** ≤ 0, **GAMMADIST** returns the *#NUM!* error value
- the equation for the gamma probability density function is:

$$f(x;\alpha,\beta) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}$$

- the standard gamma probability density function is:

$$f(x;\alpha) = \frac{x^{\alpha-1} e^{-x}}{\Gamma(\alpha)}$$

- when **alpha** = 1, **GAMMADIST** returns the exponential distribution with:

$$\lambda = \frac{1}{\beta}$$

- for a positive integer **n**, when **alpha = n**/2, **beta** = 2, and **cumulative** = TRUE, **GAMMADIST** returns (1 - **CHIDIST(x)**) with **n** degrees of freedom
- when **alpha** is a positive integer, **GAMMADIST** is also known as the **Erlang distribution**.

Please see yet another example below:

	A	B	C	D
1	Description	Scenario 1	Scenario 2	
2	Value at which you want to evaluate the distribution	9.975	9.975	
3	Alpha parameter to the distribution	9	9	
4	Beta parameter to the distribution	2	2	
5	Cumulative value	FALSE	TRUE	
6				
7				
8	Formula	Description	Result	
9	=GAMMA.DIST(B2,B3,B4,B5)	Probability density	0.032394	
10	=GAMMA.DIST(C2,C3,C4,C5)	Cumulative distribution	0.067281	
11				

The A to Z of Excel Functions: GAMMAINV

The **GAMMAINV** function is the inverse of the Gamma cumulative distribution, *i.e.* if **p = GAMMADIST(x, α, β)** then **x = GAMMAINV(p, α, β)**. This can help to identify key points in a variable whose distribution may be skewed. It has the following syntax:

GAMMAINV(probability, alpha, beta)

Again, it's important to note that this function has been replaced with a new function (**GAMMA.INV**) that may provide improved accuracy and whose name better reflects its usage and consistency with other programming languages. Although this function is still available for backward compatibility, you should consider using the new functions from now on, because this function may not be available in future versions of Excel.

The **GAMMAINV** function has the following arguments:

- **probability:** this is required and this represents the probability associated with the distribution
- **alpha:** this is also required. This is a parameter (the shape parameter) to the distribution
- **beta:** this too is required and is another parameter (the scale parameter) to the distribution. If **beta** = 1, **GAMMADIST** returns the standard gamma distribution.

It should be noted that:

- if any argument is text, **GAMMAINV** returns the *#VALUE!* error value
- if **probability** < 0, if **probability** > 1, if **alpha** ≤ 0 or if **beta** ≤ 0, **GAMMAINV** returns the *#NUM!* error value.

Given a value for **probability, GAMMAINV** seeks that value **x** such that **GAMMADIST(x, alpha, beta, TRUE) = probability**. Thus, the precision of **GAMMAINV** depends upon the precision of **GAMMADIST**. Therefore, GAMMAINV uses an iterative search technique. If the search has not converged after 64 iterations, the function returns the *#N/A* error value.

Please see our example below:

	A	B	C	D
1	Description	Values		
2	Probability associated with the distribution	0.067281		
3	Alpha parameter to the distribution	9		
4	Beta parameter to the distribution	2		
5				
6				
7	Formula	Description	Result	
8	=GAMMAINV(B2,B3,B4)	Inverse of the Gamma cumulative distribution for the given probability	9.975	
9				

The A to Z of Excel Functions: GAMMALN.PRECISE

The **GAMMALN.PRECISE** function is the natural logarithm of the Gamma function, **Γ(N)**. It has the following syntax:

GAMMALN.PRECISE(x)

The **GAMMALN.PRECISE** function has the following argument:

- **x:** this is required and this represents the value for which you wish to calculate the natural logarithm of **Γ(x)**.

It should be noted that:

- if **x** is nonnumeric, **GAMMALN.PRECISE** returns the *#VALUE!* error value
- if **x** ≤ 0, **GAMMALN.PRECISE** returns the *#NUM!* error value
- the number **e** raised to the **GAMMALN.PRECISE(i)** power, where **i** is an integer, returns the same result as **(i - 1)!**
- **GAMMALN.PRECISE** is calculated as follows:
  GAMMALN.PRECISE = LN(Γ(x)).

Please see our penultimate example below:

	A	B	C
1	Formula	Description	Result
2	=GAMMALN.PRECISE(4)	Natural logarithm of the Gamma function at 4.	1.792

The A to Z of Excel Functions: GAMMALN

The **GAMMALN** function is the natural logarithm of the Gamma function, **Γ(N)**. It has the following syntax:

GAMMALN(x)

It's important to note that this function has been replaced with a new function (**GAMMALN.PRECISE**) that may provide improved accuracy and whose name better reflects its usage and consistency with other programming languages. Although this function is still available for backward compatibility, you should consider using the new functions from now on, because this function may not be available in future versions of Excel.

The **GAMMALN** function has the following argument:

- **x:** this is required and this represents the value for which you wish to calculate the natural logarithm of **Γ(x)**.

It should be noted that:

- if **x** is nonnumeric, **GAMMALN** returns the *#VALUE!* error value
- if **x** ≤ 0, **GAMMALN** returns the *#NUM!* error value
- the number **e** raised to the **GAMMALN(i)** power, where i is an integer, returns the same result as **(i - 1)!**
- **GAMMALN** is calculated as follows:
  GAMMALN = LN(Γ(x)).

Please see our final example for this month below:

	A	B	C
1	Formula	Description	Result
2	=GAMMALN(4)	Natural logarithm of the Gamma function at 4.	1.792

More Excel Functions next month.

Beat the Boredom Suggested Solution

This month, we consider allocating fees across monthly periods, where some dates are outside the forecast period and others straddle the (monthly) reporting periods.

The Challenge

As a reminder from earlier in this newsletter, this month sees you continue last month's work for an education establishment, seeking to model forecast fees for this calendar year. There are four terms relevant to your modelling period:

Assumed Term Dates

Term	Start	End	Fees ($)	Days in Year	Check
1	30 Nov 20	6 Mar 21	10,000	65	☑
2	27 May 21	17 Jun 21	8,000	22	☑
3	1 Sep 21	4 Oct 21	12,000	34	☑
4	17 Oct 21	7 Aug 22	9,000	76	☑
			39,000	**197**	☑

All we need to do is allocate the number of term days (including weekends and public holidays) to each month of 2021, *i.e.*

		Jan 21	Feb 21	Mar 21	Apr 21	May 21	Jun 21	Jul 21	Aug 21	Sep 21	Oct 21	Nov 21	Dec 21
Term Fees in Each Month													
Term Fees in Each Month	29,020	3,196	2,887	619	-	1,818	6,182	-	-	10,588	1,869	915	946

The challenge was "simply" this: were you able to construct a calculation such that the correct fees would be allocated to each month? You were advised that you may assume the terms would be in chronological order, that they would not overlap, there will never be more than two terms associated with any given month, and there will never be two start dates or two end dates in the same month.

Suggested Solution

There are no plans to recreate the wheel here. Last month's solution will be exploited fully, as essentially all we have to do is add in a few more calculations. And that begins immediately with the inputs: did you notice that the total of the fees for the 12 months in our screenshot *(above)* ($29,020) did not equal the total fees for the four terms ($39,000) in the previous image?

This is because some of these fees accrue for dates either before the start date of the calendar year (here, 1 January 2021) or after the end of the same calendar year (*i.e.* 31 December 2021). Therefore, we should add these calculations into our Assumptions section, so we can ensure fees reconcile accordingly, *viz*.

Assumed Term Dates

Term	Start	End	Fees ($)	Days in Year	Daily Fee ($)	Check
1	30 Nov 20	6 Mar 21	10,000	65	103	☑
2	27 May 21	17 Jun 21	8,000	22	364	☑
3	1 Sep 21	4 Oct 21	12,000	34	353	☑
4	17 Oct 21	7 Aug 22	9,000	76	31	☑
			39,000	**197**	**198**	☑
	In Year		**29,020**			

```
=IF(G18-F18+1,H18/(G18-F18+1),)
=IF(G19-F19+1,H19/(G19-F19+1),)
=IF(G20-F20+1,H20/(G20-F20+1),)
=IF(G21-F21+1,H21/(G21-F21+1),)

=SUMPRODUCT(I18:I21*J18:J21)
```

Cells **J18:J21** calculate the daily fee, by dividing the fees (column **H**) by the total number of days in the term, not just the number of days that relate to the period (*i.e.* the year 2021). For example, the formula in cell **J18** is given by

=IF(G18-F18+1,H18/(G18-F18+1),)

where **H18** contains the fees and **G18-F18+1** represents the number of the days in the period (one [1] has to be added, otherwise the first day of the period is excluded). The **IF** check is simply added to avoid an *#DIV/0!* error.

The formula in cell **H23**,

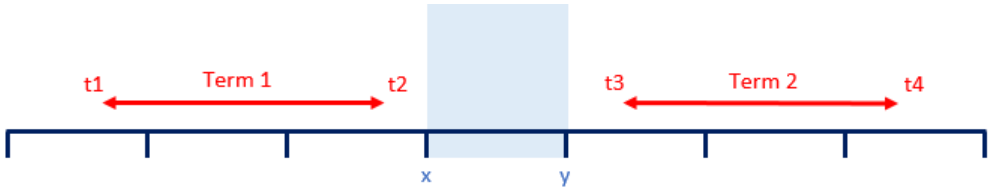
=SUMPRODUCT(I18:I21*J18:J21)

simply multiplies each term's daily fee by the number of days for that term that are in 2021 and adds them up. This is how we can confirm the cost of $29,020.

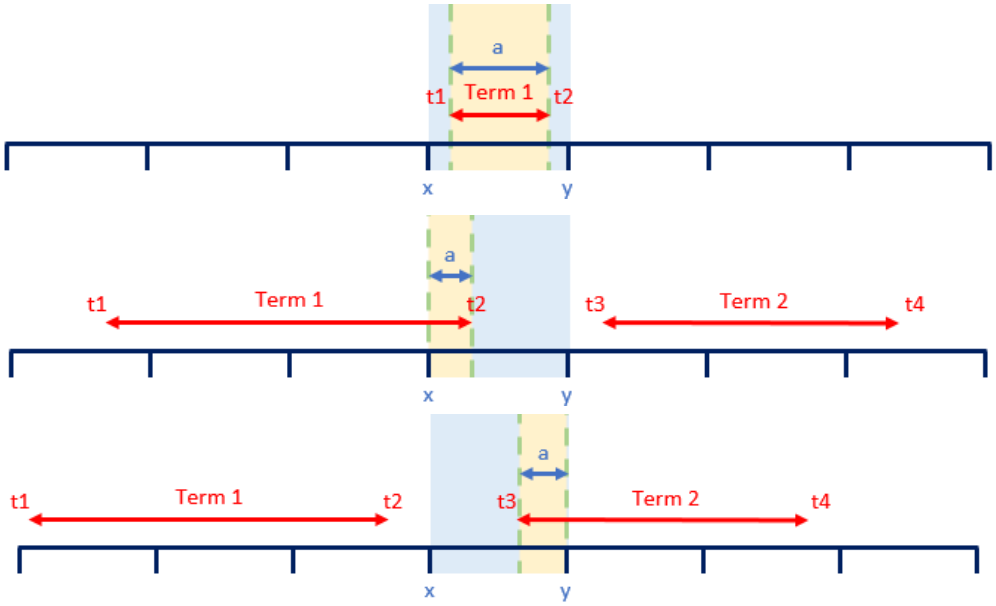
From last month's solution (which we have modified here), the idea is simple. We just need to calculate the number of days of each term are in each month, and then multiply the number of days identified by the daily rate *(already calculated, above)*. There is just one complication: which term?

This is where we need to go back to last month's reasoning. There are three scenarios for each term:

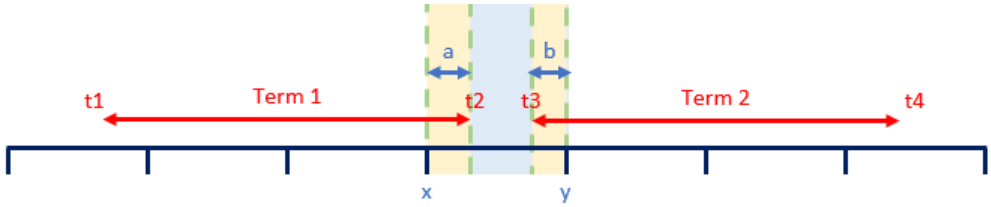
1. No term days are in the reporting period



2. At least one of the start and / or end dates for a given term is in the reporting period (start date must be less than or equal to the end date)



3. One term ends and the next term starts within the reporting period (start date occurs after the end date for that reporting period).



Calculating the relevant end dates are not the problem. If the end date of a term falls outside the reporting period, we just use the end of the period instead (deemed end). However, start dates have two issues:

1. The start date may have occurred prior to the forecast period. If the end date is also earlier, then this constitutes no issue, but if the end date is within the period, we will have to assume the deemed start is the first day of the reporting period
2. We need to check if the start date starts before / coincides or ends after the end date for a given reporting period. This will determine the method of calculation.

With all this borne in mind, we are now in a position to model the problem.

You may recall our final calculations from last time (do note that the rows have moved one row since last month, as a row has been added to the assumptions earlier):

		Jan 21	Feb 21	Mar 21	Apr 21
Term Dates in Each Month					
Relevant Start Date					
Relevant End Date				6 Mar 21	
Calculation Type		2	2	2	1
Number of Days	197	31	28	6	-

=IFERROR(IF(LOOKUP(J$5,$F$33:$F$36)=J$5,LOOKUP(J$5,$F$33:$F$36,$F$18:$F$21),IF(I$45>N(I$46),I$45,"")),"")
=IFERROR(IF(LOOKUP(J$5,$G$33:$G$36)=J$5,LOOKUP(J$5,$G$33:$G$36,$G$18:$G$21),""),"")
=(COUNT(J$45:J$46)<>0)*1+1+(N(J$45)>N(J$46))*(N(J$46)<>0)
=CHOOSE(J$47,,MIN(J$5,J$46)-MAX(J$6,J$45)+1,J$46+J$5-J$45-J$6+2)

Apologies for the still small, yet truncated, screenshot!

The key rows are rows 47 and 48:

- **Row 47 (Calculation Type):** This identifies whether Scenario 1, 2 or 3 applies *(defined above)*
- **Row 48 (Number of Days):** This calculates the total number of days in the month. Only in Scenario 3 will the number of term days come from two different terms and that is what is key here.

There aren't many more calculations to go, but the screenshots do get smaller!

		Jan 21	Feb 21	Mar 21	Apr 21
Term Fees in Each Month					
Daily Fee for First Term in Month		103	103	103	-
Daily Fee for Second Term in Month		-	-	-	-
Days in First Term for Month		31	28	6	-
Days in Second Term for Month		-	-	-	-
Number of Days	197	31	28	6	-
Days Check	☑	☑	☑	☑	☑
Term Fees in Each Month	29,020	3,196	2,887	619	-
Fees Check	☑				

=IF(J$47=3,SUMPRODUCT(($F$18:$F$21<=J46)*($G$18:$G$21>=J46)*$J$18:$J$21),SUMPRODUCT(($F$18:$F$21<=J7)*($G$18:$G$21>=J6)*$J$18:$J$21))
=IF(J$47=3,SUMPRODUCT(($F$18:$F$21<=J45)*($G$18:$G$21>=J45)*$J$18:$J$21),)
=CHOOSE(J$47,,MIN(J$7,J$46)-MAX(J$6,J$45)+1,J$46-J$6+1)
=IF(J$47=3,J$7-J$45+1,)
=CHOOSE(J$47,,MIN(J$5,J$46)-MAX(J$6,J$45)+1,J$46+J$5-J$45-J$6+2)
=(ROUND(J48-J62,Rounding_Accuracy)<>0)*1
=SUMPRODUCT(J57:J58*J60:J61)

For those with normal eyesight, never mind those of us visually impaired, let me explain, er, a little more clearly:

- in Scenario 1, there are no term dates in the month
- in Scenario 2, there are term dates in the month, but from only one term
- in Scenario 3, there are again term dates in the month, but some come from the end of one term and the remaining days come from the beginning of the next term.

Given the number of days has already been calculated, technically, we only need to identify Scenario 3 months and then calculate the day allocation between the two months. However, so that we may reconcile / check the total number of term days in each month, we will calculate the number of days again.

Row 57 calculates the daily fee for the first term in the month:

	Jan 21	Feb 21	Mar 21	Apr 21	May 21	Jun 21	Jul 21	Aug 21	Sep 21	Oct 21	Nov 21	Dec 21
Term Fees in Each Month												
Daily Fee for First Term in Month	103	103	103	-	364	364	-	-	353	353	31	31
Daily Fee for Second Term in Month	-	-	-	-	-	-	-	-	-	31	-	-

The formula in cell **J57** is given by

=IF(J$47=3,SUMPRODUCT(($F$18:$F$21<=J46)*($G$18:$G$21>=J46)*$J$18:$J$21),
SUMPRODUCT(($F$18:$F$21<=J7)*($G$18:$G$21>=J6)*$J$18:$J$21))

Whilst I have "skirted over" **SUMPRODUCT** earlier – our namesake! – perhaps now is the time to elaborate.

SUMPRODUCT Refresher

I must admit this is obviously one of our favourite functions in Excel – so much so our company was named after it (time for a shameless plug)!

At first glance, the basic form

SUMPRODUCT(range1, range2, ...)

appears quite humble. Before showing an example, though, let's look at the syntax carefully:

- we'll discuss them in more detail, but a **range** for Excel purposes is a collection of cells either one column wide or one row deep *usually* with **SUMPRODUCT**. The ranges must be contiguous
- this basic functionality uses the comma delimiter (**,**) to separate the arguments (ranges). Unlike most Excel functions, it is possible to use other delimiters, but this will be revisited shortly.

To show how **SUMPRODUCT** works, imagine you worked in retail and the electronic register had developed a fault. Consequently, all sales had to be kept in a tally chart, that is, the pricing points were listed in the first column of a spreadsheet and the sales were then noted in the second column. At the end of the day, you would have to calculate the total revenue and reconcile it with the payments received:

Example Sales Report

Dataset

Unit Price	# Sales	Total Sales
$1	10	$10
$2	18	$36
$5	25	$125
$10	6	$60
$50	9	$450
$100	4	$400

Summation	**$1,081**

=SUM(H12:H17)

The sales in column **H** are simply the product of columns **F** and **G**, for example, the formula in cell **H12** is simply **=F12*G12**. Then, to calculate the entire amount cell **H19** sums column **H**. This could all be performed much quicker using the following formula:

=SUMPRODUCT(F12:F17,G12:G17)

That is, **SUMPRODUCT** does exactly what it says on the tin: it sums the individual products.

Example Sales Report

Dataset

Unit Price	# Sales	Total Sales
$1	10	$10
$2	18	$36
$5	25	$125
$10	6	$60
$50	9	$450
$100	4	$400

Summation	**$1,081**

=SUM(H12:H17)

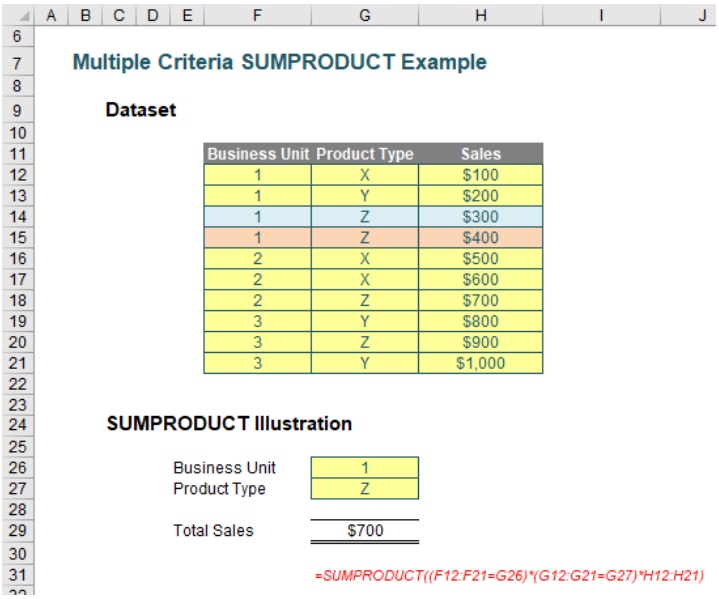
SUMPRODUCT Approach	**$1,081**

=SUMPRODUCT(F12:F17,G12:G17)

I mentioned the comma delimiter earlier. You can multiply the vectors together instead:

=SUMPRODUCT(F12:F21*G12:G21)

This will produce the same result. However, there is an important difference.



Multiple Criteria SUMPRODUCT Example

Dataset

Business Unit	Product Type	Sales
1	X	$100
1	Y	$200
1	Z	$300
1	Z	$400
2	X	$500
2	X	$600
2	Z	$700
3	Y	$800
3	Z	$900
3	Y	$1,000

SUMPRODUCT Illustration

Business Unit	1
Product Type	Z
Total Sales	$700

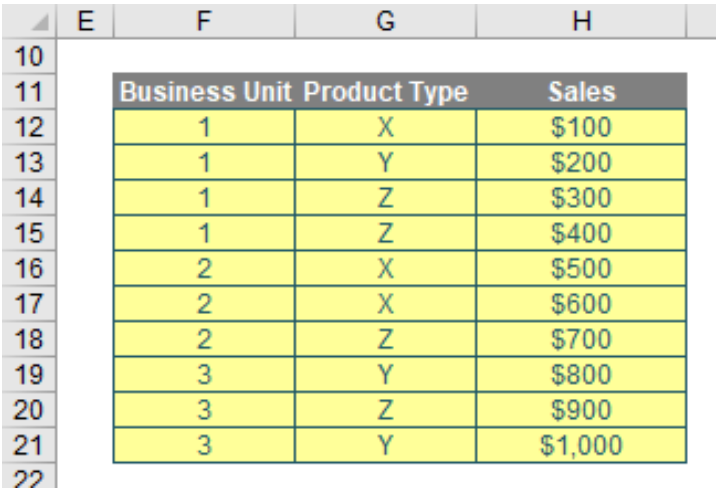
=SUMPRODUCT((F12:F21=G26)*(G12:G21=G27)*H12:H21)

SUMPRODUCT will work with numbers that aren't really numbers and different sized ranges. However, if you look at the formula in the example, you can be forgiven for not understanding the formula. Let me explain.

Where **SUMPRODUCT** comes into its own is when dealing with multiple criteria. This is done by considering the properties of TRUE and FALSE in Excel, namely:

- TRUE*number = number (for example, TRUE*7 = 7)
- FALSE*number = 0 (for example, FALSE*7=0).

Consider the following example:



Business Unit	Product Type	Sales
1	X	$100
1	Y	$200
1	Z	$300
1	Z	$400
2	X	$500
2	X	$600
2	Z	$700
3	Y	$800
3	Z	$900
3	Y	$1,000

we can test columns **F** and **G** to check whether they equal our required values. **SUMPRODUCT** could be used as follows to sum only sales made by Business Unit 1 for Product Z:

=SUMPRODUCT((F12:F21=1)*(G12:G21="Z")*H12:H21).

For the purposes of this calculation, (**F12:F21=1**) replaces the contents of cells **F12:F21** with either TRUE or FALSE depending on whether the value contained in each cell equals 1 or not. The brackets are required to force Excel to compute this first before cross-multiplying.

Similarly, **(G12:G21="Z")** replaces the contents of cells **G12:G21** with either TRUE or FALSE depending on whether the value "Z" is contained in each cell.

Therefore, the only time cells **H12:H21** will be summed is when the corresponding cell in the arrays **F12:F21** and **G12:G21** are both TRUE, then you will get TRUE*TRUE*number, which equals the said number.

Note also that this uses the * delimiter rather than the comma, analogous to TRUE*number. If you were to use the comma delimiter instead, the syntax would have to be modified thus:

=SUMPRODUCT(--(F12:F21=1),--(G12:G21="Z"),H12:H21)

Minus minus? The first negation in front of the brackets converts the array of TRUEs and FALSEs to numbers, albeit substituting -1 for TRUE and 0 for FALSE. The second minus sign negates these numbers so that TRUE is effectively 1, rather than -1, whilst FALSE remains equals to zero. This variant often confuses end users which is why I recommend the first version described above.

Returning to Our Suggested Solution

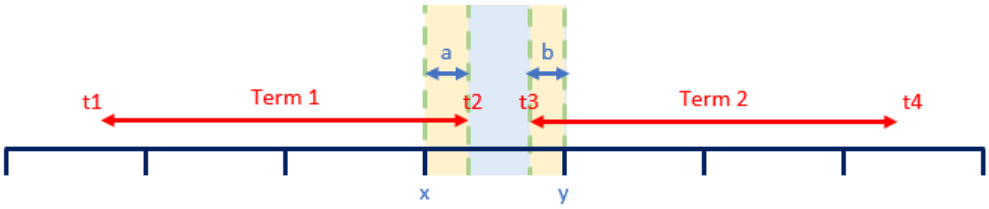
As mentioned earlier, row 57 calculates the daily fee for the first term in the month:

	Jan 21	Feb 21	Mar 21	Apr 21	May 21	Jun 21	Jul 21	Aug 21	Sep 21	Oct 21	Nov 21	Dec 21
Term Fees in Each Month												
Daily Fee for First Term in Month	103	103	103	-	364	364	-	-	353	353	31	31
Daily Fee for Second Term in Month	-	-	-	-	-	-	-	-	-	31	-	-

As a reminder, the formula in cell **J57** is given by

=IF(J$47=3,SUMPRODUCT(($F$18:$F$21<=J46)*($G$18:$G$21>=J46)*$J$18:$J$21),
SUMPRODUCT(($F$18:$F$21<=J7)*($G$18:$G$21>=J6)*$J$18:$J$21))

Perhaps now it is a little easier to follow. Here, the **IF** statement checks if Scenario 3 applies **(J$47=3)**:



If so, then we consider the end date of the first term (**t2,** not a James Cameron movie). This is given by cell **J46** (the Relevant End Date). We compare this to the relevant Start and End dates for each term:

Assumed Term Dates

Term	Start	End	Fees ($)	Days in Year	Daily Fee ($)	Check
1	30 Nov 20	6 Mar 21	10,000	65	103	☑
2	27 May 21	17 Jun 21	8,000	22	364	☑
3	1 Sep 21	4 Oct 21	12,000	34	353	☑
4	17 Oct 21	7 Aug 22	9,000	76	31	☑
			39,000	**197**	**198**	☑
	In Year		**29,020**			

The formula

SUMPRODUCT(($F$18:$F$21<=J46)*($G$18:$G$21>=J46)*$J$18:$J$21)

determines which term contains the end date and then returns the Daily Fee from column **J**.

If Scenario 3 does not apply, then this formula is modified slightly:

SUMPRODUCT(($F$18:$F$21<=J7)*($G$18:$G$21>=J6)*$J$18:$J$21)

The formula replaces **J46**, the end date of the relevant term with the end date (**J7**) and start date (**J6**) respectively of the selected month instead:

	Jan 21	Feb 21	Mar 21	Apr 21
Start Date	1 Jan 21	1 Feb 21	1 Mar 21	1 Apr 21
End Date	31 Jan 21	28 Feb 21	31 Mar 21	30 Apr 21
Number of Days	31	28	31	30
Counter	1	2	3	4

It does not matter whether Scenario 1 (no dates) or Scenario 2 (some, or all, term dates are captured) as the formula will either return nothing or the correct Daily Fee accordingly.

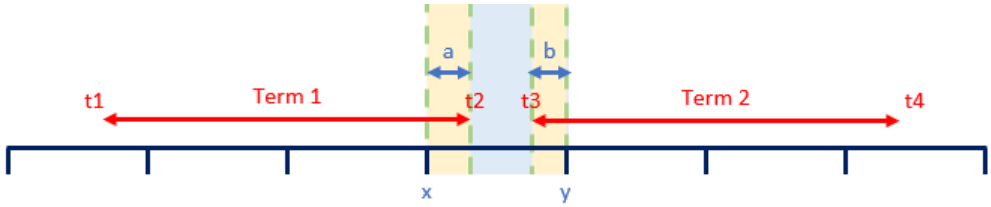
Row 58, Daily Fee for Second Term in Month, only applies to Scenario 3, by its very definition:

	Jan 21	Feb 21	Mar 21	Apr 21	May 21	Jun 21	Jul 21	Aug 21	Sep 21	Oct 21	Nov 21	Dec 21
Term Fees in Each Month												
Daily Fee for First Term in Month	103	103	103	-	364	364	-	-	353	353	31	31
Daily Fee for Second Term in Month	-	-	-	-	-	-	-	-	-	31	-	-

Here, cell **J58** contains the following formula:

=IF(J$47=3,SUMPRODUCT(($F$18:$F$21<=J45)*($G$18:$G$21>=J45)*$J$18:$J$21),)

After checking Scenario 3 applies (**J$47=3**), the formula again considers the mechanics of Scenario 3:



This time, we consider the *start* date of the second term (**t3**, not a good movie, whatever you think **t3** is). This is given by cell **J45** (the Relevant Start Date). We compare this to the relevant Start and End dates for each term once more:

SUMPRODUCT(($F$18:$F$21<=J45)*($G$18:$G$21>=J45)*$J$18:$J$21)

Similar to before, this formula determines which term contains the start date and then returns the Daily Fee from column **J**.

Now that we have the correct Daily Fee(s) allocated to each month, we need to deduce how many term days relate to this fee / these fees.

Row 60 calculates the days in the first term for the month:

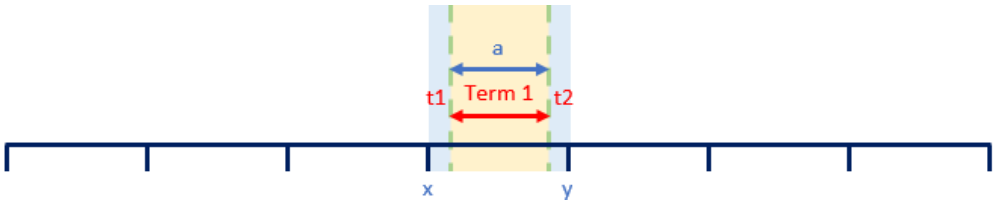
			Jan 21	Feb 21	Mar 21	Apr 21	May 21	Jun 21	Jul 21	Aug 21	Sep 21	Oct 21	Nov 21	Dec 21
59														
60	Days in First Term for Month		31	28	6	-	5	17	-	-	30	4	30	31
61	Days in Second Term for Month		-	-	-	-	-	-	-	-	-	15	-	-
62	Number of Days	197	31	28	6	-	5	17	-	-	30	19	30	31
63	Days Check	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑

For example, the formula in cell **J60** is given by

=CHOOSE(J$47,,MIN(J$7,J$46)-MAX(J$6,J$45)+1,J$46-J$6+1)

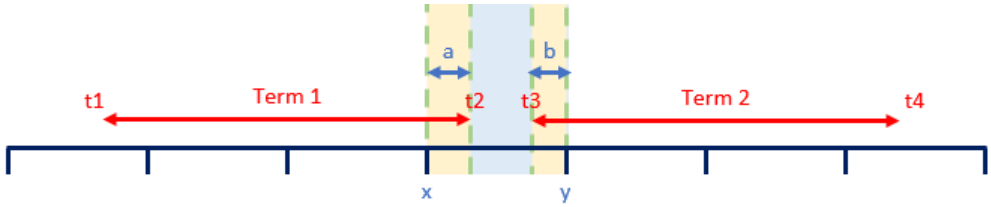
The **CHOOSE** function was covered last month: in essence, the first argument selects which of the subsequent arguments applies. Essentially, there are three scenarios:

1. If no term dates occur (**J$47** is equal to 1), then the value is zero (no days)
2. If one term's dates are (partially) included in the month (**J$47** is equal to 2), then the number of days is given by **MIN(J$7,J$46)-MAX(J$6,J$45)+1**



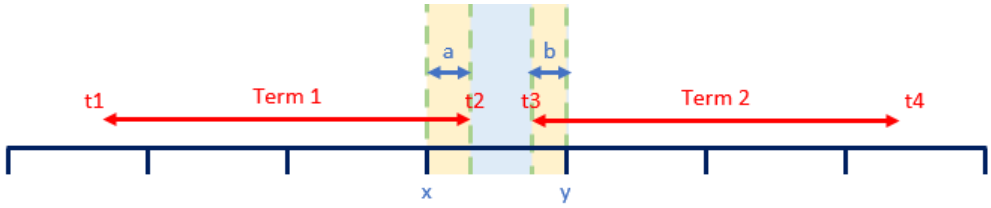
i.e. the number of days is the lower of the month end date (**y**) and the term end date (**t2**) less the higher of the month start date (**x**) and the term start date (**t1**) plus one (so the first day of the term dates is added back)

3. If two terms' days are included (**J$47** is equal to 3), then the number of days is given by **J$46-J$6+1**



i.e. the number of days is given by the first term end date (**t2**) less the month start date (x) plus one (to again add the first day of the term days back).

The Days in the Second Term for the Month (row 61) is not so complex. This only ever non-zero in Scenario 3 (where **J$47** is equal to 3) and the number of days is given by **J$7-J$45+1**



i.e. the number of days is given by the month end date (**y**) less the second term start date (**t3**) plus one (to again add the first day of the term days back). Therefore, the whole formula for cell **J61** is given by

=IF(J$47=3,J$7-J$45+1,)

Row 62, Number of Days, can either be the same formula calculated from last time *(pictured)*

5			Jan 21	Feb 21	Mar 21	Apr 21	
59							
60	Days in First Term for Month		31	28	6	-	=CHOOSE(J$47,,MIN(J$7,J$46)-MAX(J$6,J$45)+1,J$46-J$6+1)
61	Days in Second Term for Month		-	-	-	-	=IF(J$47=3,J$7-J$45+1,)
62	Number of Days	197	31	28	6	-	=CHOOSE(J$47,,MIN(J$5,J$46)-MAX(J$6,J$45)+1,J$46+J$5-J$45-J$6+2)
63	Days Check	☑	☑	☑	☑	☑	=(ROUND(J48-J62,Rounding_Accuracy)<>0)*1
64							
65	Term Fees in Each Month	29,020	3,196	2,887	619	-	=SUMPRODUCT(J57:J58*J60:J61)
66	Fees Check	☑					
67							

or else merely sum rows 60 and 61 (with row 63 providing a check that the number of days in rows 48 and 62 are identical, which only makes sense if you use a different formula to calculate the two lines!).

Row 65 then simply uses the **SUMPRODUCT** formula once more, *e.g.* in cell **J65**, the formula is given by

=SUMPRODUCT(J57:J58*J60:J61)

which multiplies the daily fee(s) by the number of days in the first term and second term (where applicable), and then adds them together. Cell **I66** ensures this total equates to the total calculated in cell **H23** earlier.

And that's it: you have the Term Fees calculated on a monthly basis.

Word to the Wise

I am conscious there are other ways to calculate a solution to this month's challenge, but I wanted to build on last month's suggested solution. It's important to re-use calculations where possible, and not reinvent the wheel pointlessly.

Until next time.

Location	Course	Date	Date	Duration	Duration
Online (Australia)	Financial Modelling	7 - 8 Oct 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	3 - 5 Nov 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	10 Nov 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	11 - 12 Nov 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	8 - 10 Dec 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	15 Dec 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	16 - 17 Dec 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days

Key Strokes

Each newsletter, we'd like to introduce you to useful keystrokes you may or may not be aware of. This year, we thought we'd revisit each function key in depth (there are 12 – one for each month of the year!). Given it's now August, let's look at the **F8** tips.

Keystroke	What it does
F8	Extend selection mode
ALT + F8	Run macro
CTRL + F8	Size window
SHIFT + F8	Add to selection mode

Hey, it's not our fault so many do the same thing this month!

There are *c.*550 keyboard shortcuts in Excel. For a comprehensive list, please download our Excel file at www.sumproduct.com/thought/keyboard-shortcuts. Also, check out our new daily **Excel Tip of the Day** feature on the www.sumproduct.com homepage.

Our Services

We have undertaken a vast array of assignments over the years, including:

- **Business planning**
- **Building three-way integrated financial statement projections**
- **Independent expert reviews**
- **Key driver analysis**
- **Model reviews / audits for internal and external purposes**
- **M&A work**
- **Model scoping**
- **Power BI, Power Query & Power Pivot**
- **Project finance**
- **Real options analysis**
- **Refinancing / restructuring**
- **Strategic modelling**
- **Valuations**
- **Working capital management**

If you require modelling assistance of any kind, please do not hesitate to contact us at contact@sumproduct.com.

Link to Others

These newsletters are not intended to be closely guarded secrets. Please feel free to forward this newsletter to anyone you think might be interested in converting to "the SumProduct way".

If you have received a forwarded newsletter and would like to receive future editions automatically, please subscribe by completing our newsletter registration process found at the foot of any www.sumproduct.com web page.

Any Questions?

If you have any tips, comments or queries for future newsletters, we'd be delighted to hear from you. Please drop us a line at newsletter@sumproduct.com.

Training

SumProduct offers a wide range of training courses, aimed at finance professionals and budding Excel experts. Courses include Excel Tricks & Tips, Financial Modelling 101, Introduction to Forecasting and M&A Modelling.

Check out our more popular courses in our training brochure:

Drop us a line at training@sumproduct.com for a copy of the brochure or download it directly from www.sumproduct.com/training.

Sydney Address: SumProduct Pty Ltd, Suite 803, Level 8, 276 Pitt Street, Sydney NSW 2000
New York Address: SumProduct Pty Ltd, 48 Wall Street, New York, NY, USA 10005
London Address: SumProduct Pty Ltd, Office 7, 3537 Ludgate Hill, London, EC4M 7JN, UK
Melbourne Address: SumProduct Pty Ltd, Ground Floor, 470 St Kilda Road, Melbourne, VIC 3004
Registered Address: SumProduct Pty Ltd, Level 14, 440 Collins Street, Melbourne, VIC 3000

contact@sumproduct.com
www.sumproduct.com
+61 3 9020 2071