



Happy new year! Resolve to remain inFORMed about the latest Excel developments

It's simple – just subscribe to our newsletter and you can stay up to date with what's new in Excel as well as Power BI and the winning lottery numbers. Well, maybe not lottery numbers – do you think I'd be here writing this each month if I'd figured that out!?

Well of course I would... 😊

This month we inFORM you about Microsoft Forms now integrated into Excel and focus on Focus Cells now Generally Available too (regular readers take note: there are some updates since our original article).

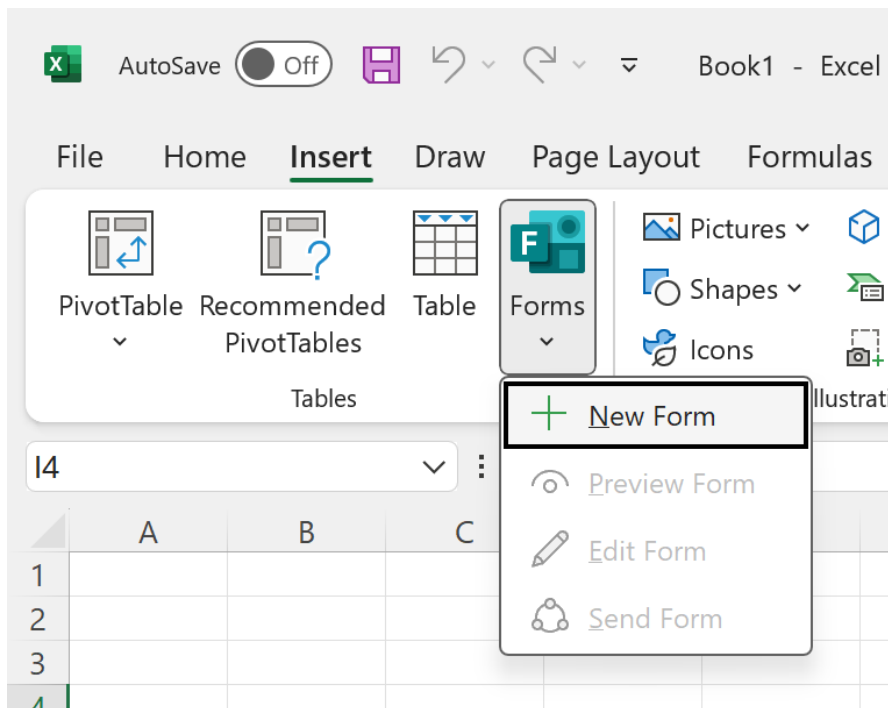
The New Year printing deadlines prevent us detailing any Power BI Updates this month, but don't worry, that will just mean we will have a 2,000-page newsletter next month. In the meantime, the remainder of regulars stand up to be counted: there is the usual Beat the Boredom Challenge, Charts & Dashboards Tips, Over to AI, Excel for Mac, Visual Basics, Power Pivot Principles, Power Query Pointers, the latest Excel Updates, plus one **OR** two A to Z of Excel functions and finally, we follow Keyboard Shortcuts to the letter.

Bring it on 2025! The bad puns remain. Happy reading and remember: stay safe, stay happy, stay healthy.

Liam Bastick, Managing Director, SumProduct



Microsoft Forms Now Integrated on Excel Desktop

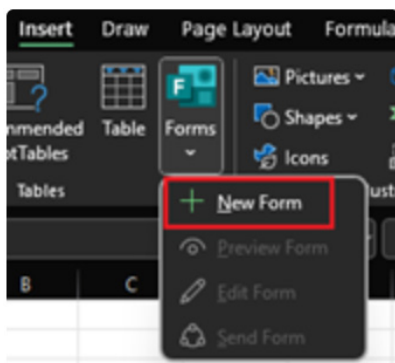


With the latest updates, you can now create a new Microsoft Form directly within an Excel workbook, linking the form to the workbook seamlessly. This feature, previously available in Excel for the Web only, is now available in the Windows app, with Mac support coming soon.

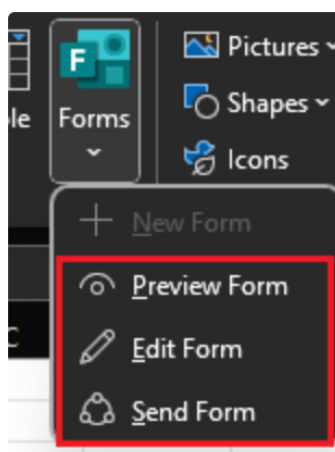
This new integration is particularly beneficial for users who frequently collect data from others. By enabling the creation of a form directly from an Excel workbook, you can streamline the data collection process, making it more efficient and less prone to errors.

It works as follows:

- On the Insert tab, select **Forms -> New Form**:



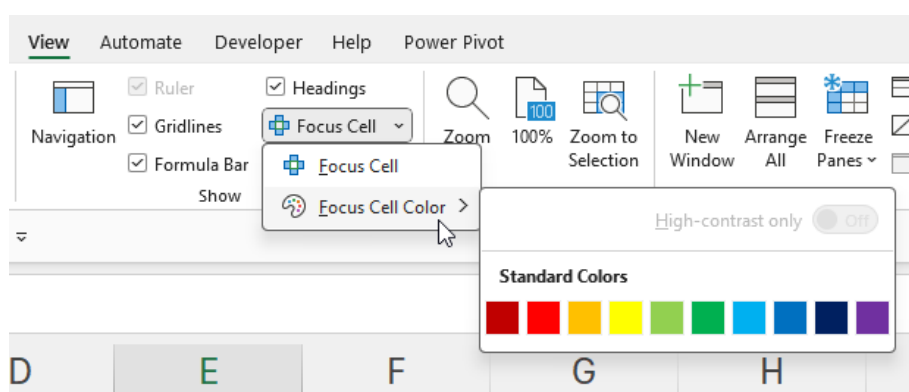
- A new browser tab will open, presenting you with a blank form to customise according to your needs. Simultaneously, a linked Table will appear in your workbook, ready to synchronise with the form. As you edit the form or receive new submissions, the linked Table in your workbook will automatically update, ensuring that your data is always current and accurate
- After creating your form, you can access several other capabilities from the Forms menu:



- **Preview Form:** this option will open the form in preview mode in a new browser tab, allowing you to see how the form will look to respondents
- **Edit Form:** this option will open the form in edit mode in a new browser tab, so you can make changes to the form as needed
- **Send Form:** this option will open the form in a new browser tab and show the dialog letting you send the form out to respondents and begin collecting responses.

The new Microsoft Forms integration is available to all Current Channel users running Version 2410 (Build 16.0.18227.20000) or later. Mac users can also look forward to this integration in an upcoming update.

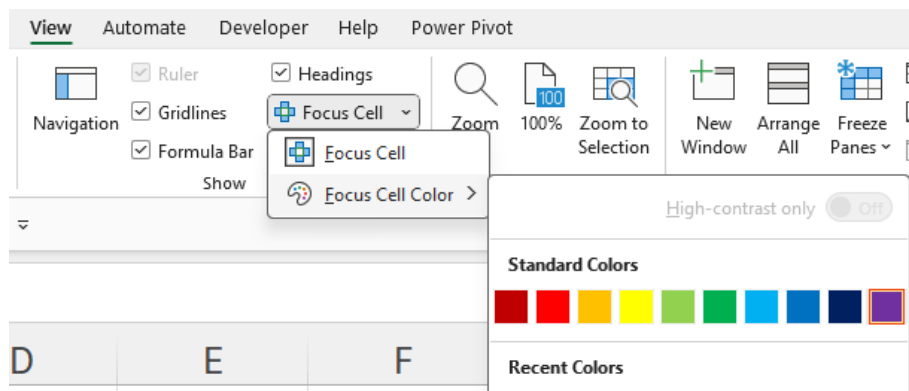
Focus Cell Now Generally Available



A few months back, we mentioned this feature was in Preview. Now, Microsoft has announced that Focus Cell is now Generally Available on desktop, both Windows and Mac.

Found on the View tab in the Ribbon, 'Focus Cell' provides a small drop-down menu:

- **Focus Cell:** this toggles the feature on or off
- **Focus Cell Color:** this allows you to select from a wide array of, er, 10 colours to use for highlighting. Actually, that's not true as 'Recent Colors' (*sic*) will show other colours that you may employ:



At this stage, we are not sure what the 'High-contrast only' toggle switch does as it still appears to be permanently disabled.

So what does it do? It appears to enhance accessibility for the visually impaired as it highlights the active row and column of the cell selected, viz.

	A	B	C	D
1				
2				
3				
4				
5				

Here, cell **B2** imitates Whoopi Goldberg's 1985 hit 'The Color Purple', although clearly the colour displayed is not quite the same as the colours depicted in the 'Focus Cell Color' dialog.

Clearly, this feature helps users to zoom in on the cell selected. Other colours may be used, e.g.

	A	B	C	D
1				
2				
3				
4				
5				

Here, cell **A1** sees green. Envious, methinks.

If multiple cells are selected, the active cell remains highlighted:

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					

or

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					

It does seem to act a little unexpectedly though if entire rows or column are chosen:

	A	B	C	D
1				
2				
3				
4				
5				

	A	B	C	D
1				
2				
3				
4				
5				

Presently, this feature now appears to work with 'Freeze Pane' or 'Split' employed, but still has some issues with files created in earlier versions of Excel.

To avail yourself of this colourful feature, you must be using Version 2410 (Build 18118.20000) or later for Windows and Version 16.91 (Build 24109300) or later for Mac.

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...

Suppose you have a Table named **Data** in Excel, containing a list of names under the column **Name** and corresponding numerical data under the column **Grade**, viz.

The screenshot shows an Excel spreadsheet with the following content:

- Row 1: **SUM IF between**
- Row 2: **SP SUM IF between 2 Names.xlsm**
- Row 3: **Navigator**
- Row 4: Error Checks:
- Row 6: **1. SUM IF between**
- Row 8: **Data**
- Table with 2 columns: **Name** and **Grade**
- Rows 11-24: Data entries for names and grades.
- Row 28: Between
- Row 29: and

Name	Grade
Charlie Taylor	6
Ellie Nguyen	8
Eloise Graham	7
Emily White	10
Ezra Gray	5
Freya Ward	5
Harrison Adams	10
Isabelle Davies	3
Marcus Graham	7
Matilda Scott	7
Olive Gray	9
Piper James	8
Samuel Gray	7
Scarlett Wilson	7

Your task is to write a single Excel formula that sums the **Grade** data between two names exclusively, which we referred to as **Name_1** (Cell **G28**) and **Name_2** (Cell **G29**). These names are inputs that could be changed, and the sum should dynamically update to reflect the range of data between these two names in the **Data** table.

As always, there are some requirements:

- the formula needs to be within just one column (no "helper" cells)
- the solution must work even if the order of **Name_1** and **Name_2** are swapped in the list
- the formula should be dynamic so that they should update when a new entry was added.

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



Here at SumProduct we write many articles for many magazines and our authoring contracts have been revised recently to ensure Artificial Intelligence hasn't taken over our duties. We can assure everyone they have not.

But it did get us thinking: *what if we did let AI loose and write a regular article for the newsletter?*

This month, it looks like we feed its infatuation with **VLOOKUP**...

This Month's Prompt: Which function in Excel is most frequently misunderstood? Why?

Excel is a powerful tool used by millions of people around the world for various purposes, from simple calculations to complex data analysis. Among its myriad of functions, there is one that stands out both for its utility and for the confusion it often causes: the **VLOOKUP** function. Despite its potential to simplify data retrieval from large datasets, **VLOOKUP** is frequently misunderstood and misused. Here, we will explore why **VLOOKUP** is misunderstood, how it works, common pitfalls and best practices for its effective use (*don't! – Ed.*).

WHAT IS VLOOKUP?

VLOOKUP, short for "Vertical Lookup," is a function that allows users to search for a value in the first column of a table array and return a value in the same row from a specified column. It is particularly useful in large datasets where manually searching for data would be time-consuming and prone to errors.

SYNTAX OF VLOOKUP

To understand why **VLOOKUP** is often misunderstood, it is essential to start with its syntax. The **VLOOKUP** function has the following format:

VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])

where it has the following arguments:

- **lookup_value**: this is the value you want to search for in the first column of the table array
- **table_array**: the range of cells that contains the data you want to search through
- **col_index_num**: the column number in the table array from which to retrieve the value. The first column of the table array is one [1]
- **[range_lookup]**: an optional argument that specifies whether you want an exact match (FALSE) or an approximate match (TRUE). The default is TRUE.

COMMON MISUNDERSTANDINGS AND PITFALLS

Despite its straightforward appearance, **VLOOKUP** is frequently misused. Here are some common reasons why.

Misconception 1: Incorrect Range for Table Array

One of the most common mistakes is specifying an incorrect range for the table array. Users often forget that the **lookup_value** must be in the first column of the table array. If the **lookup_value** is not in the first column, **VLOOKUP** will return an error or incorrect data.

Misconception 2: Wrong Column Index Number

Another frequent error is using the wrong column index number. Users sometimes count columns incorrectly or use a static number when the table array has been adjusted, leading to incorrect results.

Misconception 3: Ignoring the Range Lookup Argument

The **range_lookup** argument is often misunderstood or ignored. By default, **VLOOKUP** uses an approximate match, which can lead to unexpected results if an exact match is needed. Specifying **FALSE** for the **range_lookup** argument ensures that only exact matches are returned.

Misconception 4: Performance Issues with Large Datasets

VLOOKUP can be slow when dealing with large datasets because it scans each row sequentially until it finds a match. Users expecting faster performance may be disappointed unless they optimise their data or consider alternative functions like **INDEX** and **MATCH**.

Misconception 5: Limitations of VLOOKUP

VLOOKUP has limitations that users might not be aware of. It can only search for values in the first column of the table array, and it cannot look up data to the left of the lookup column. These limitations can be frustrating and may require users to rearrange their data or use alternative functions.

BEST PRACTICES FOR USING VLOOKUP

Understanding how to use **VLOOKUP** correctly can significantly enhance its effectiveness. Here are some best practices to follow:

- 1. Ensure the Lookup Value is in the First Column.**
Always double-check that your lookup value is in the first column of your table array. If necessary, rearrange your data to meet this requirement
- 2. Use Absolute References.**
When specifying your table array, use absolute references (e.g. **\$A\$1:\$D\$10**) to prevent errors when copying the formula to other cells
- 3. Specify the Range Lookup Argument.**
Always define the **range_lookup** argument explicitly. Use **FALSE** for an exact match to avoid unexpected results
- 4. Optimise Your Data.**
If working with large datasets, consider sorting your data by the lookup column and using approximate **MATCH** (**TRUE**) to improve performance. Alternatively, consider using the **INDEX** and **MATCH** functions for more flexibility and efficiency
- 5. Understand When to Use Alternative Functions.**
In cases where **VLOOKUP**'s limitations are problematic, consider using other functions like **INDEX** and **MATCH**, which offer greater flexibility and can overcome some of **VLOOKUP**'s constraints.

CONCLUSION

The **VLOOKUP** function in Excel is a powerful tool that can simplify data retrieval and enhance productivity. However, its frequent misuse stems from misunderstandings of its syntax and limitations. By gaining a thorough understanding of how **VLOOKUP** works and following best practices, users can harness its full potential and avoid common pitfalls. Whether for simple lookups or complex data analysis, mastering **VLOOKUP** is an invaluable skill for any Excel user.

What do you think? Would you use **VLOOKUP** (you know our views if you are a regular reader!)? Send us your comments and ideas for questions to contact@sumproduct.com.

Excel for Mac

With Steve Kraynak joining the team, we thought we would exploit his knowledge and recant all about Microsoft Excel for Mac. Each month, we'll cover a different topic to help you understand how Excel for Mac is different than Excel for Windows. This month, we show you an important

tip about 'Flash Fill'. On Mac, it's one of those things that you need to know. If you know, you know. If you don't know, then 'Flash Fill' won't help you.

If you've ever experienced an automatic 'Flash Fill' in Excel for Windows, you know it feels magical when it happens. You might be typing some values in a new column next to your data, and Excel automatically fills in the rest of the values based on the first few examples that you've typed.

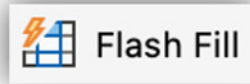
Flash Fill Works on Mac, But it's Not Automatic

Many features in Excel are powerful, but you can't take advantage of them unless you know about them. What makes 'Flash Fill' a bit different, at least on Windows, is that you don't need to know about it. You can just enter some data, and it shows itself when it can help. It does come at a slight cost, though. 'Flash Fill' is always watching what you're typing

and trying to detect if there's a pattern that it can follow. In most cases, there's no noticeable impact on performance, so you won't notice any slowdown in Excel. However, concern about performance is probably the reason why it's not automatic on Mac.

Even so, it is very simple to use:

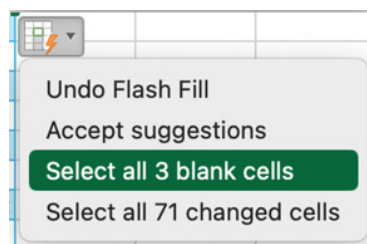
- Type a value in a cell next to existing data in your sheet. In the example shown below, we typed "blue" in cell **B2**
- Then press the 'Flash Fill' button on the Data tab of the Ribbon or press **CTRL + E** on your keyboard



- 'Flash Fill' will then fill in all the other rows to follow along with the example you typed. You can then proceed with making corrections or accepting the suggestions
- You'll see a pop-up button near the top of the list of cells where 'Flash Fill' did its work. You can press the button or you may select all the cells that were changed by 'Flash Fill':

	A	B	C	D	E	F
1	Product name	Color	ID	Mfg		
2	Litware 80mm LED Dual PCI Slot Fan E1501 Blue	blue	E1501	Litware		
3	Litware 80mm LED Dual PCI Slot Fan E1501 Pink	pink	E1501	Litware		
4	Litware 20" Weather-Shield Performance Box Fan E1601 Orange	orange		Litware		
5	Litware 20" Weather-Shield Performance Box Fan E1601 Yellow	yellow		Litware		
6	Litware 17" Oscillating Pedestal Fan M125 Pink	pink				
7	Litware 17" Oscillating Pedestal Fan M125 Yellow	yellow				
8	Litware 17" Oscillating Pedestal Fan M125 Blue	blue				
9	Litware 18" Oscillating Pedestal Fan M135 Pink	pink				
10	Litware 18" Oscillating Pedestal Fan M135 Yellow	yellow				
11	Litware 18" Oscillating Pedestal Fan M135 Blue	blue				
12	Litware 18" Oscillating Pedestal Fan M145 Pink	pink	M145	Litware		
13	Litware 18" Oscillating Pedestal Fan M145 Yellow	yellow	M145	Litware		
14	Litware 18" Oscillating Pedestal Fan M145 Blue	blue	M145	Litware		
15	NT Bluetooth Stereo Headphones E52 Blue	blue	E52	NT		
16	NT Bluetooth Stereo Headphones E52 Yellow	yellow	E52	NT		
17	NT Bluetooth Stereo Headphones E52 Pink	pink	E52	NT		
18	NT Wireless Bluetooth Stereo Headphones E102 Blue	blue	E102	NT		

If some cells were left blank, you can select those from the menu, making it easy to find them. Then you can type a value that you want for those cells that Flash Fill wasn't able to fill for you.

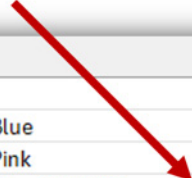


You should note that another difference between Windows and Mac is that on Mac there is no keyboard shortcut to open the 'Flash Fill' options menu that appears. You can only open it on Mac by clicking it with your mouse. On Windows, you can simply press the **CTRL** key on your keyboard.

The End Result is the Same

Despite the differences mentioned above, the values that result should be the same as you would get in Excel for Windows. The differences are just that it's not automatic, and it doesn't show a preview of the data (as shown below), which is where the suggested values appear in a light grey. It just fills in the values.

Mac will not show the preview.



	A	B
1	Product name	Color
2	Litware 80mm LED Dual PCI Slot Fan E1501 Blue	blue
3	Litware 80mm LED Dual PCI Slot Fan E1501 Pink	pink
4	Litware 20" Weather-Shield Performance Box Fan E1601 Orange	orange
5	Litware 20" Weather-Shield Performance Box Fan E1601 Yellow	yellow
6	Litware 17" Oscillating Pedestal Fan M125 Pink	pink
7	Litware 17" Oscillating Pedestal Fan M125 Yellow	yellow
8	Litware 17" Oscillating Pedestal Fan M125 Blue	blue
9	Litware 18" Oscillating Pedestal Fan M135 Pink	pink
10	Litware 18" Oscillating Pedestal Fan M135 Yellow	yellow
11	Litware 18" Oscillating Pedestal Fan M135 Blue	blue
12	Litware 18" Oscillating Pedestal Fan M145 Pink	pink
13	Litware 18" Oscillating Pedestal Fan M145 Yellow	yellow
14	Litware 18" Oscillating Pedestal Fan M145 Blue	blue
15	NT Bluetooth Stereo Headphones E52 Blue	blue
16	NT Bluetooth Stereo Headphones E52 Yellow	yellow
17	NT Bluetooth Stereo Headphones E52 Pink	pink
18	NT Wireless Bluetooth Stereo Headphones E102 Blue	blue

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 stay on the case of Case.

A client has asked us about the overall efficiency of the **Select Case** statement, and whether it was a problem that we might have dozens of different **Case** statements. In particular, the concern was that if the macro found the answer in the very first statement, that it would then waste time checking against every other **Case** in the list. Given that this was being run once per row, for thousands of rows, they felt justifiably concerned!

Well, there's an easy way to test this. We can take the numeric example we showed last month and amend it slightly so that two cases overlap:

```
Select Case margin
    Case Is > 0.7
        MsgBox ("Great result!")
    Case Is > 0.3
        MsgBox ("Not a bad result")
    Case Is < 0.3
        MsgBox ("Need to do better")
End Select
```

In this instance, if we provide a margin value of 0.8, it should trigger the first **Case** statement. Then, if we don't get a second message box, then we can confirm it stops after hitting the first **Case** statement. Otherwise, if we get the second message box, then we will know that it jumps to **End Select** once it's completed a **Case** successfully.

As it turns out, it does jump to the end. Therefore, even if you have dozens of **Case** statements, if the first one is true, it will ignore the rest and go straight to the end, saving a bit of processing time from checking the other **Cases**.

More next time.

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 will talk about how to create a dynamic legend.

We recently wrote a small series about building a conditional Do(ugh)nut chart where the colour of chart series representing the group's rating changes depending on their rating. Here, we complete the chart with the data labels.

Chart Data

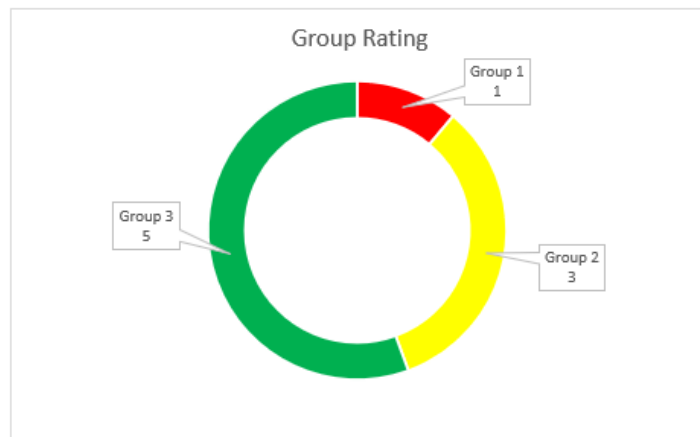
Group Rating

Group	Rating
Group 1	1
Group 2	3
Group 3	5

Chart Data

Donut Chart

Donut Chart



The data labels do not look nice at all times. For example, when we change the group rating, they look as follows:

Chart Data

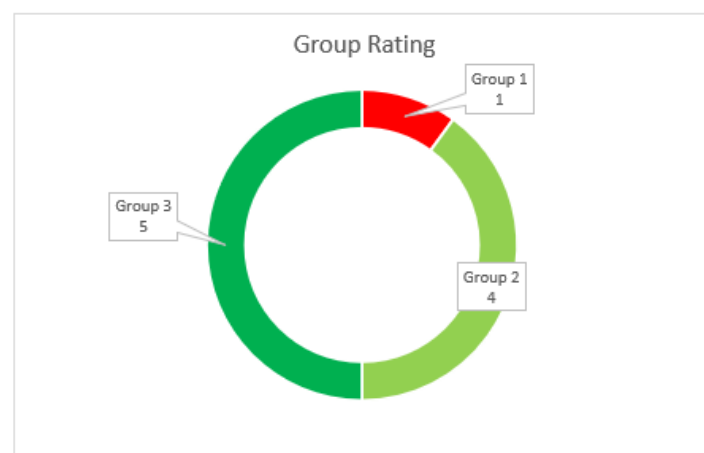
Group Rating

Group	Rating
Group 1	1
Group 2	4
Group 3	5

Chart Data

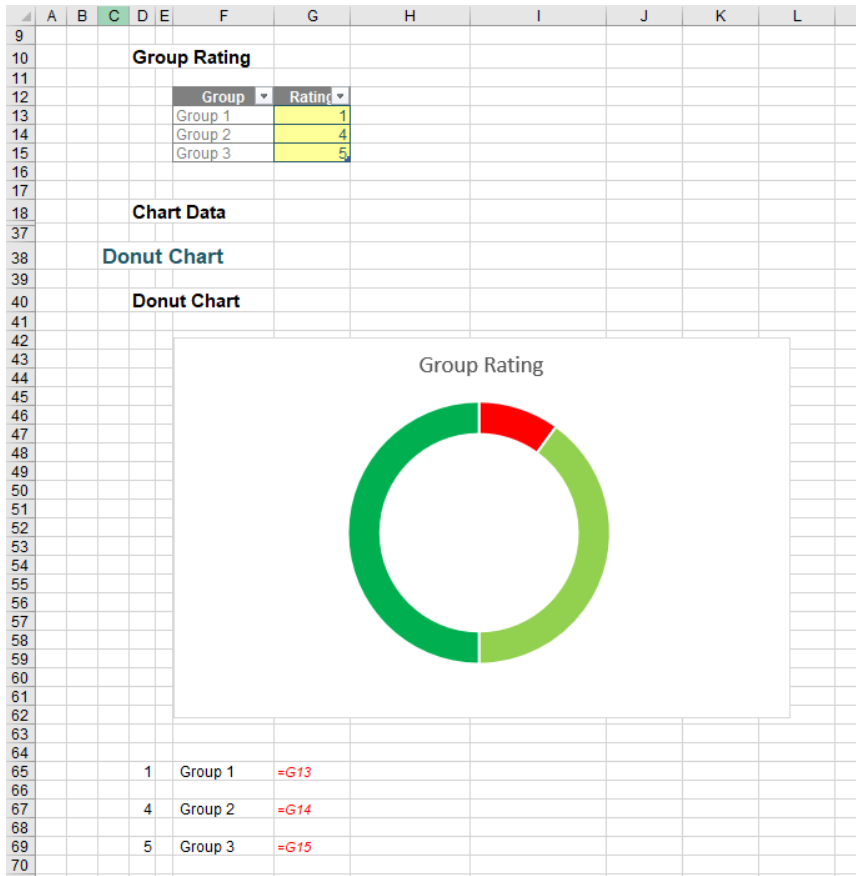
Donut Chart

Donut Chart

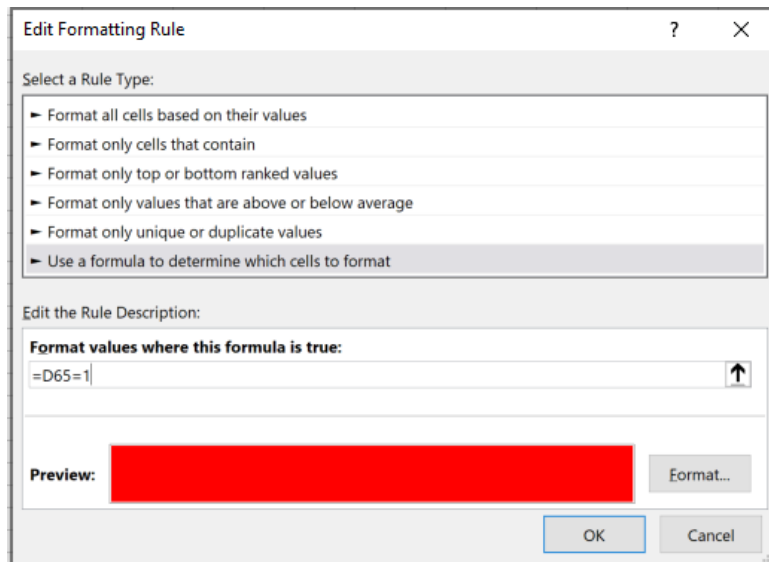


Here, the colour has changed, without explanation. Therefore, we want to create a legend to fix this issue. In this case, the legend should be dynamic so that the colour on the legend should change following the changes in the data.

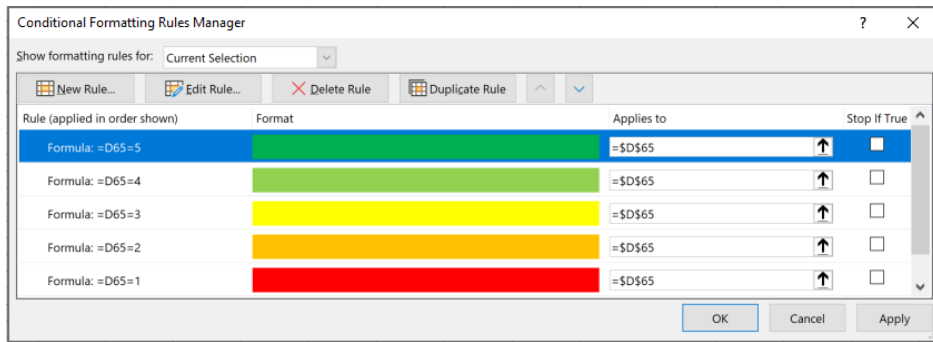
First, we will remove the data labels from the chart and create a range of cells for the legend. In our example, this will be in cells **D65:F69** (below):



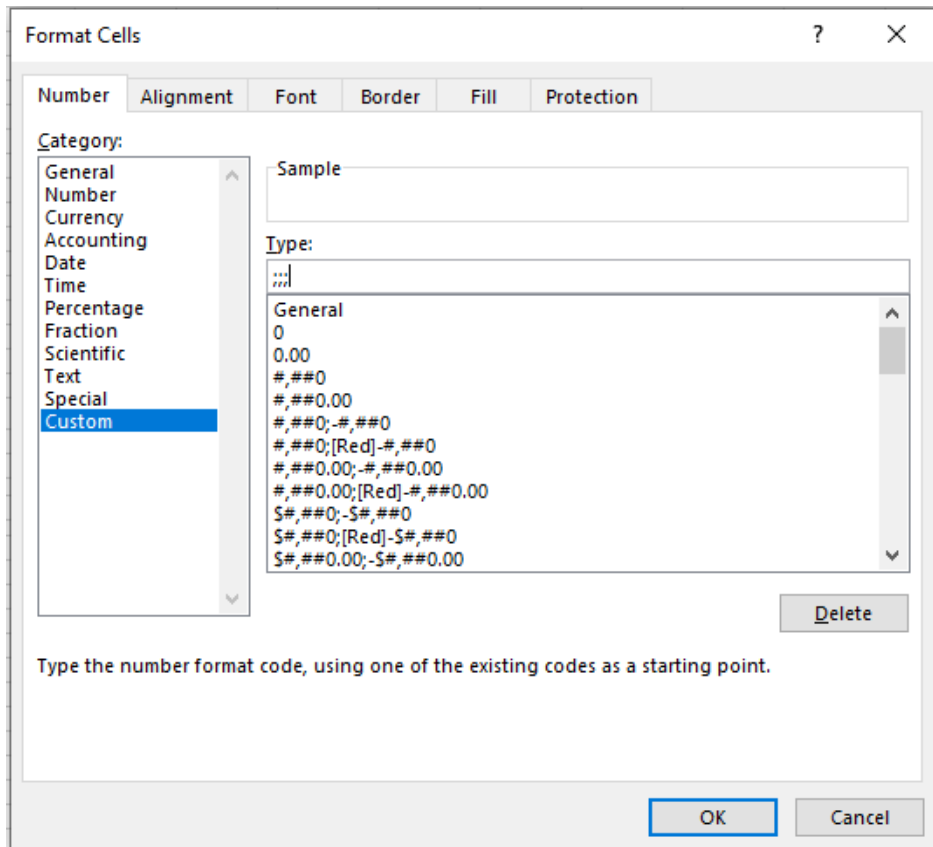
Next, we will apply the colour conditional formatting to cell **D65** so that it will display the colour based on the rating by navigating to the Home tab on the Ribbon and choosing **Conditional Formatting -> New Rules...** as shown below:



We will need to repeat this process five times, with the value being increased to two [2], then three [3], then four [4], and finally five [5], viz.



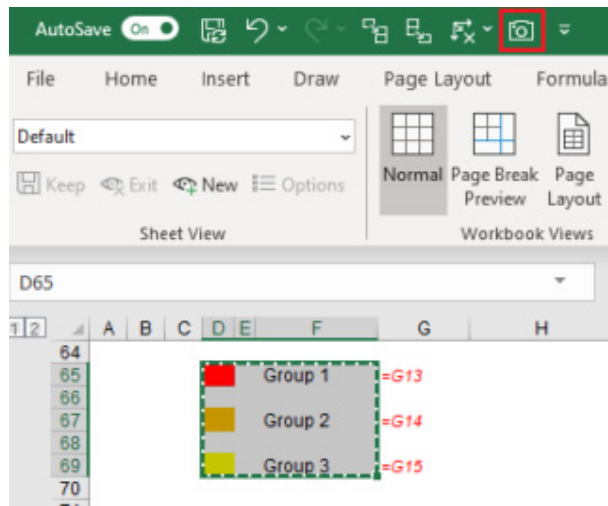
Then, we want the number in cell **D65** to be invisible so that only the colour is displayed. To do this, right-click on the cell **D65** and select 'Format Cell...' or use the **CTRL + 1** keys to open the 'Format Cells' dialog. Under the **Number -> Custom**, enter ';;;' in the Type box to make the number invisible.



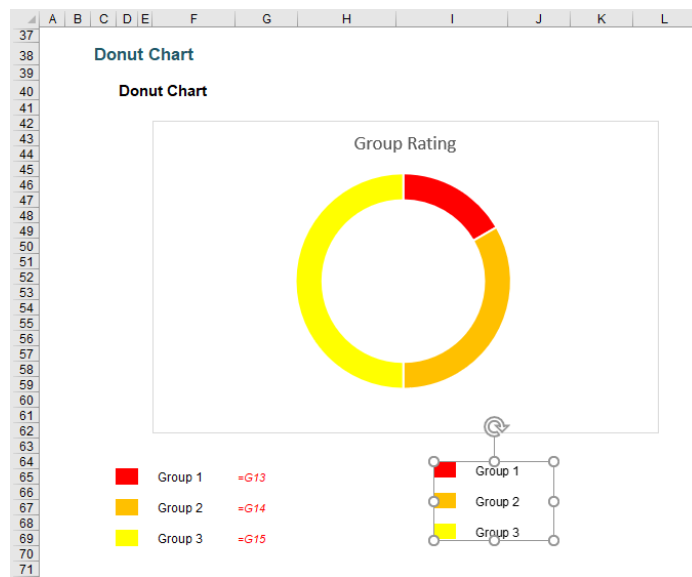
We just need to copy the formatting of cell **D65** to cell **D67** and cell **D69**. We are now having a range of cells which looks like a legend which colour depends on the group rating.

	A	B	C	D	E	F	G
64							
65						Group 1	=G13
66							
67						Group 2	=G14
68							
69						Group 3	=G15
70							
71							

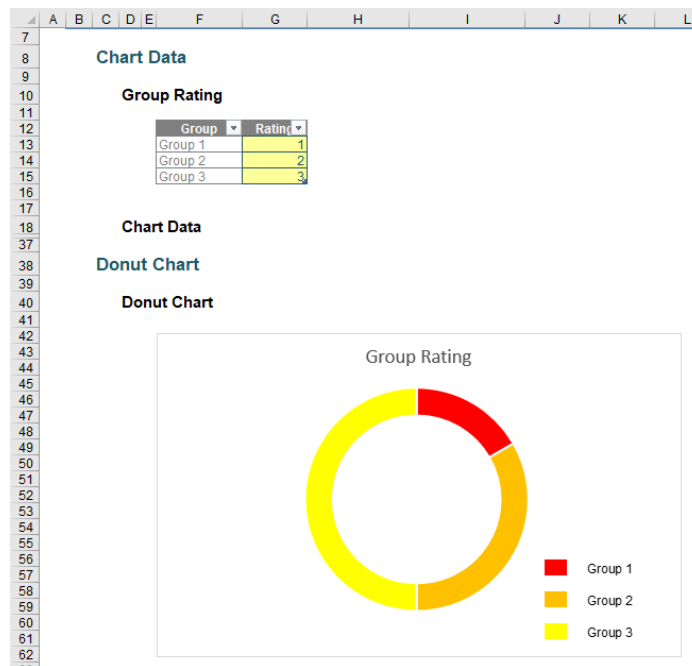
To get the legend 'image' to the chart, we will use the Camera tool (from last month's newsletter on how to get the tool and how to use it when creating charts and dashboards). We will take a snapshot of the legend by selecting the range of cells and click on the Camera icon on the Quick Access Toolbar.



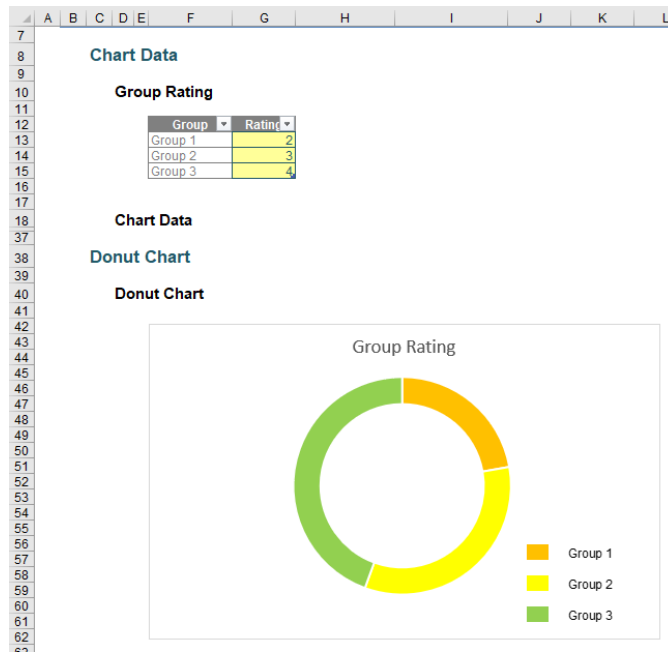
Then, click somewhere on the sheet to paste the image.



To remove the image border, right-click on the image and choose 'Format Picture...'. In the 'Format Picture' panel, let Fill be 'No fill' and Line be 'No line'. Then, we just need to drag the legend image onto the Donut chart.



Now that the group ratings are changed, the colours of the chart series and the legend are also updated accordingly.



More next time.

Power Pivot Principles

We continue our series on the Excel COM add-in, Power Pivot. This month, here's something we should have done **EARLIER**...

We will need to explain several concepts before covering the **EARLIER** function. The first concept we are going to cover is the idea of **Row Context**.

In **DAX**, the **Row Context** is a term that means: "a row by row evaluation". Essentially, it means that the formula is executed one row at a time. This may return with different results on each row of a table.

Let's take a look at an example. We are going to use the following data set (the picture below is not exhaustive):

Product ID	Product Category	Price	Amount Sold
10022	Hats	184.00	28
10023	Shoes	307.00	79
10024	Hats	96.00	75
10025	Shoes	215.00	37
10026	Hats	20.00	41
10027	Shirts	462.00	40
10028	Pants	391.00	21
10029	Shoes	360.00	38
10030	Pants	220.00	47
10031	Pants	496.00	56
10032	Shoes	206.00	92
10033	Pants	164.00	47
10034	Hats	211.00	26
10035	Shirts	111.00	93
10036	Shirts	285.00	23
10037	Shirts	336.00	49
10038	Hats	116.00	68
10039	Shoes	94.00	17
10040	Pants	188.00	90
10041	Pants	170.00	6
10042	Shoes	280.00	86
10043	Shirts	221.00	3
10044	Shoes	63.00	49
10045	Shoes	273.00	87
10046	Pants	117.00	33
10047	Pants	306.00	46
10048	Hats	462.00	88
10049	Pants	494.00	96
10050	Shoes	29.00	40
10051	Shoes	289.00	85

After loading this data into our data model, we can create the following calculated column:

=SaleTbl[Price]*SaleTbl[Amount Sold]

Product ID	Product Category	Price	Amount Sold	Sale Amount
1	10022 Hats	\$184.00	28	\$5,152.00
2	10023 Shoes	\$307.00	79	\$24,253.00
3	10024 Hats	\$96.00	75	\$7,200.00
4	10025 Shoes	\$215.00	37	\$7,955.00
5	10026 Hats	\$20.00	41	\$820.00
6	10027 Shirts	\$462.00	40	\$18,480.00
7	10028 Pants	\$391.00	21	\$8,211.00
8	10029 Shoes	\$360.00	38	\$13,680.00
9	10030 Pants	\$220.00	47	\$10,340.00
10	10031 Pants	\$496.00	56	\$27,776.00
11	10032 Shoes	\$206.00	92	\$18,952.00
12	10033 Pants	\$164.00	47	\$7,708.00
13	10034 Hats	\$211.00	26	\$5,486.00
14	10035 Shirts	\$111.00	93	\$10,323.00
15	10036 Shirts	\$285.00	23	\$6,555.00
16	10037 Shirts	\$336.00	49	\$16,464.00
17	10038 Hats	\$116.00	68	\$7,888.00
18	10039 Shoes	\$94.00	17	\$1,598.00
19	10040 Pants	\$188.00	90	\$16,920.00
20	10041 Pants	\$170.00	6	\$1,020.00
21	10042 Shoes	\$280.00	86	\$24,080.00

This would give us the total **Sale Amount**, since we are multiplying the price per product with the amount sold. This single **DAX** formula is calculating a different sale amounts for each row. This is because the **Sale Amount** is being multiplied individually for each product one row at a time.

Simple? Now, we can look at **Nested Row Contexts**. This means that there are multiple Row Contexts, or multiple row by row calculations in each row. Perhaps this is better illustrated in the following example. Using the same dataset as before we can create the calculated column:

=RANKX(SaleTbl,[Sale Amount],,ASC)

Product ID	Product Category	Price	Amount Sold	Sale Amount	Rank of Sales
1	10022 Hats	\$184.00	28	\$5,152.00	14
2	10023 Shoes	\$307.00	79	\$24,253.00	44
3	10024 Hats	\$96.00	75	\$7,200.00	21
4	10025 Shoes	\$215.00	37	\$7,955.00	24
5	10026 Hats	\$20.00	41	\$820.00	5
6	10027 Shirts	\$462.00	40	\$18,480.00	39
7	10028 Pants	\$391.00	21	\$8,211.00	25
8	10029 Shoes	\$360.00	38	\$13,680.00	30
9	10030 Pants	\$220.00	47	\$10,340.00	27
10	10031 Pants	\$496.00	56	\$27,776.00	46
11	10032 Shoes	\$206.00	92	\$18,952.00	40
12	10033 Pants	\$164.00	47	\$7,708.00	22
13	10034 Hats	\$211.00	26	\$5,486.00	16
14	10035 Shirts	\$111.00	93	\$10,323.00	26
15	10036 Shirts	\$285.00	23	\$6,555.00	18
16	10037 Shirts	\$336.00	49	\$16,464.00	37
17	10038 Hats	\$116.00	68	\$7,888.00	23
18	10039 Shoes	\$94.00	17	\$1,598.00	10
19	10040 Pants	\$188.00	90	\$16,920.00	38
20	10041 Pants	\$170.00	6	\$1,020.00	6
21	10042 Shoes	\$280.00	86	\$24,080.00	43

The **RANKX** formula has two [2] nested row contexts. It has two [2] steps:

1. It works out the total **Sale Amount** for each **Product ID**. This is the **outer row context**
2. It ranks the **Product ID** on the current row verses the entire table based on the total **Sale Amount**. This is the **inner row context**.

Now that we are all on the same page regarding inner and outer row contexts, we can move on to the **EARLIER** function.

The **EARLIER** function uses the following syntax to operate:

EARLIER (column, [number])

where:

- the **column** parameter **must** refer to a column that has numeric values or dates
- the **number** parameter is optional, it delineates the number of row contexts to step out of before evaluation. If omitted, it will default to one [1].

Let's recreate the **Rank Sales** column, this time with the **EARLIER** function.

```
=  
COUNTROWS(  
FILTER(SaleTbl,  
[Sale Amount] > EARLIER([Sale Amount])  
)  
) + 1
```

Product ID	Product Category	Price	Amount Sold	Sale Amount	Rank of Sales	Calculated Column 2
1	10060	Pants	\$0.00	43	\$0.00	50
2	10058	Hats	\$290.00	1	\$290.00	49
3	10067	Shirts	\$112.00	5	\$560.00	48
4	10043	Shirts	\$221.00	3	\$663.00	47
5	10026	Hats	\$20.00	41	\$820.00	46
6	10041	Pants	\$170.00	6	\$1,020.00	45
7	10061	Shirts	\$65.00	17	\$1,105.00	44
8	10053	Pants	\$185.00	6	\$1,110.00	43

This formula has two [2] nested row contexts:

1. the first one (inner row context) is in the **FILTER** function where each row of the table is evaluated based on the condition
2. the second (outer row context) is the **Sale Amount** calculation (**Price * Amount Sold**).

Another way to think about it is that this formula has two loops: an inner and outer loop. The inner loop is where the **FILTER** function has to evaluate all of the **Sale Amounts** in the table. The outer loop is when the **EARLIER** function instructs the program disregard the inner loop and evaluate the outer loop, defined as **[Sale Amount]**, which is the current row's **Price * Amount Sold**.

To further step it out, the formula evaluates as follows:

- the **COUNTROWS** function requires a table input
- the **FILTER** returns with a table, based upon a condition
 - the **FILTER** function begins by evaluating the first row of the **SaleTbl**, where it evaluates all the **Sale Amount** values for each row, then returns with a table of all the **Sale Amount** values that are greater than **EARLIER([Sale Amount])**
- the **EARLIER** function instructs the formula to disregard the current row context in the **FILTER** function jump one level up to the 'outer loop' where the **EARLIER** function will evaluate to '0.00' on the first row
 - the outer loop of this evaluation is the **Sale Amount** calculation, which is **Price * Amount Sold**.

The **FILTER** function can now return with a table where all of the rows have a value greater than \$0.00. The **COUNTROWS** function then counts all of the rows in that table, which is 49. This is why we had to add a "+1" at the end of the formula to return with 50. This loop is repeated for each of the remaining rows and the ranking column is calculated.

We only had two nested row contexts in this formula, therefore we omitted the second parameter (**[number]**) in this formula. If we had more levels of nested row contexts (say three) and wanted the **EARLIER** function to evaluate at the first level, we would enter '2' as the **[number]**.

That brings us to an end to this, er, simple article.

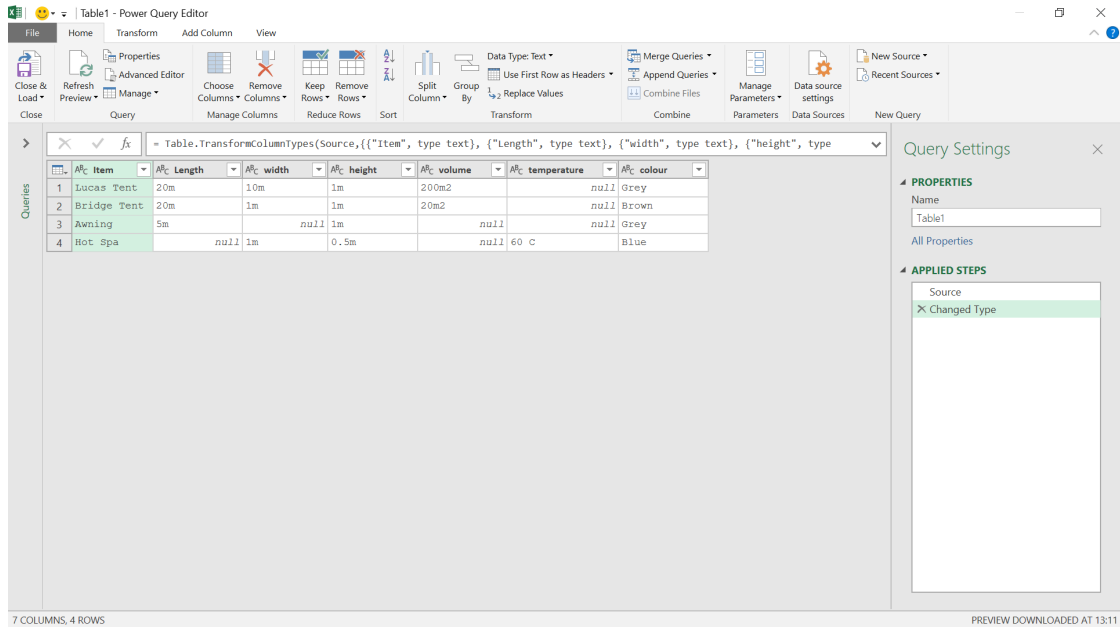
That's it for this month; more next time.

Power Query Pointers

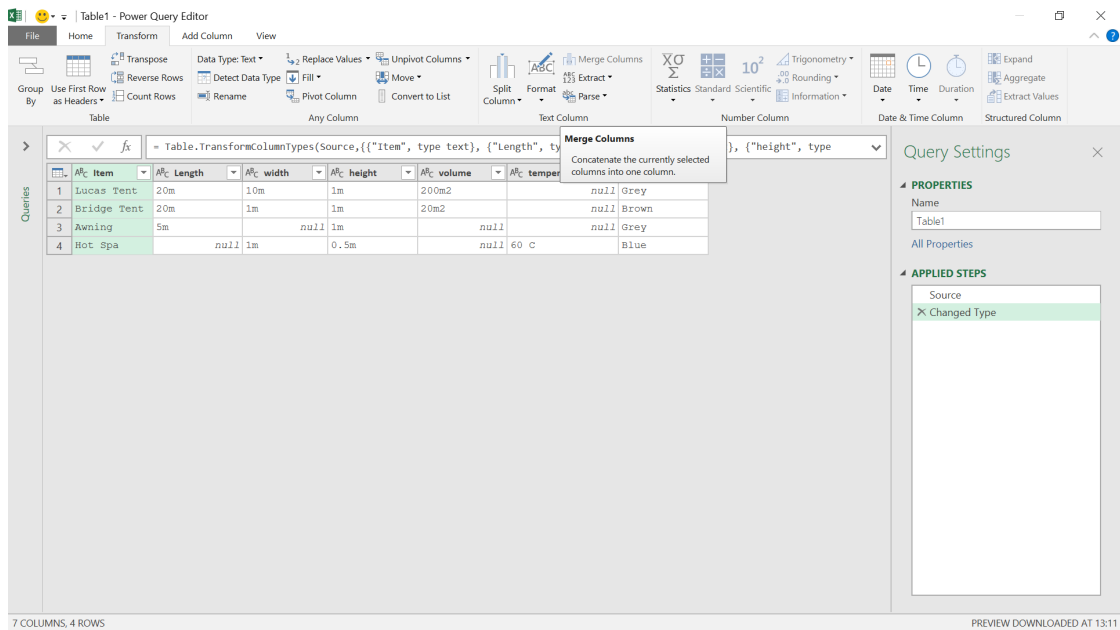
Each month we'll reproduce one of our articles on Power Query (Excel 2010 and 2013) / Get & Transform (Office 365, Excel 2016 and 2019) from www.sumproduct.com/blog. If you wish to read more in the meantime, simply check out our Blog section each Wednesday. This month, we see why not all 'Merge Columns' options are born equal.

There are currently some problems that can occur when merging columns. To demonstrate, let's use the following subset of the regularly-used tent data:

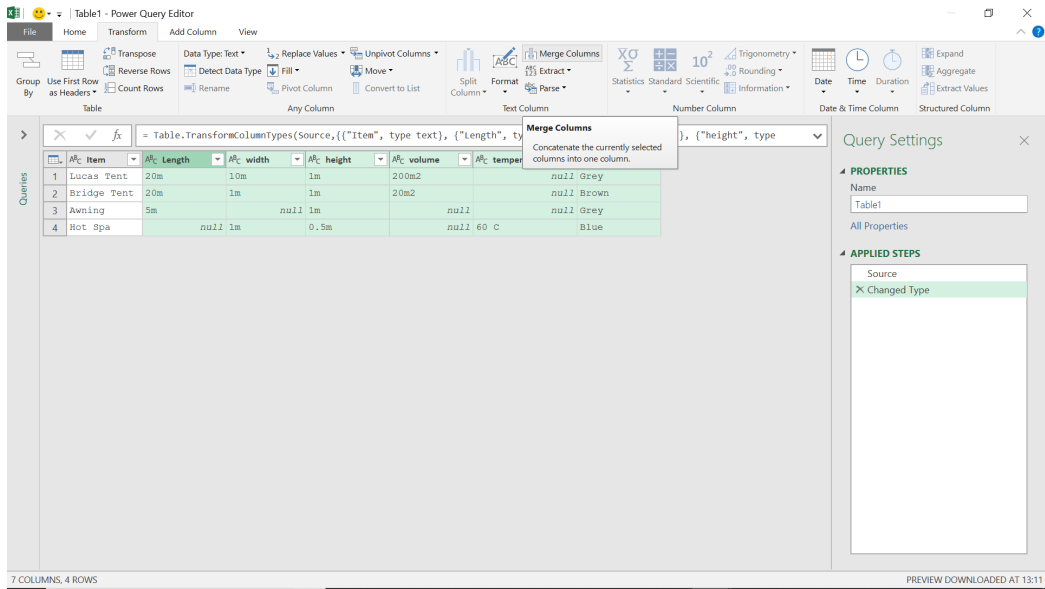
Let's take a look at an example. We are going to use the following data set (the picture below is not exhaustive):



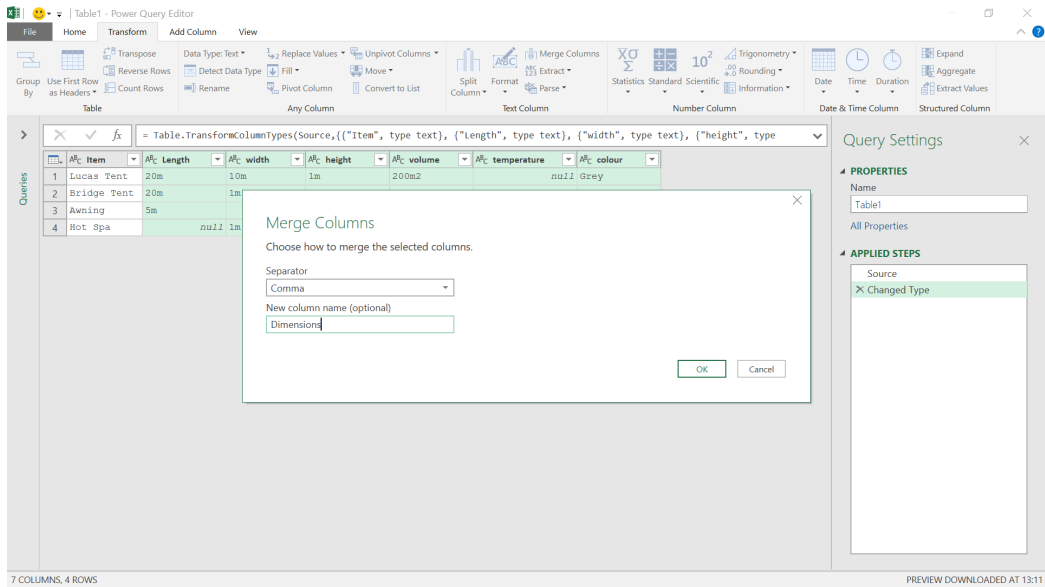
Let's create a new column which will contain all of the dimension data.



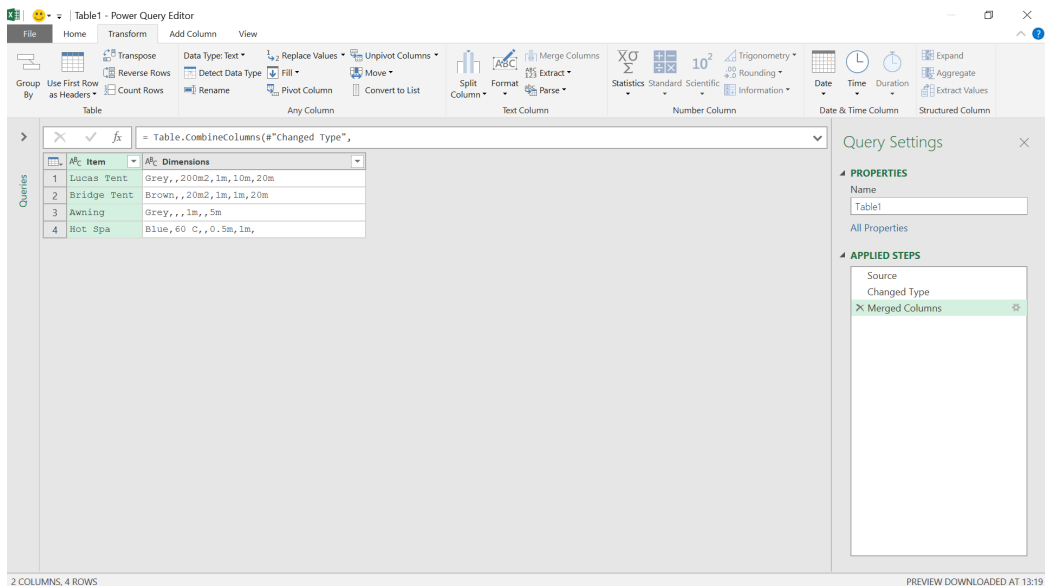
In the 'Transform' data tab, we can choose to merge columns. First, we must select them, which we can do by holding down the **CTRL** key as we click on them.



We have selected all the columns containing dimensional data, then we will choose to merge:



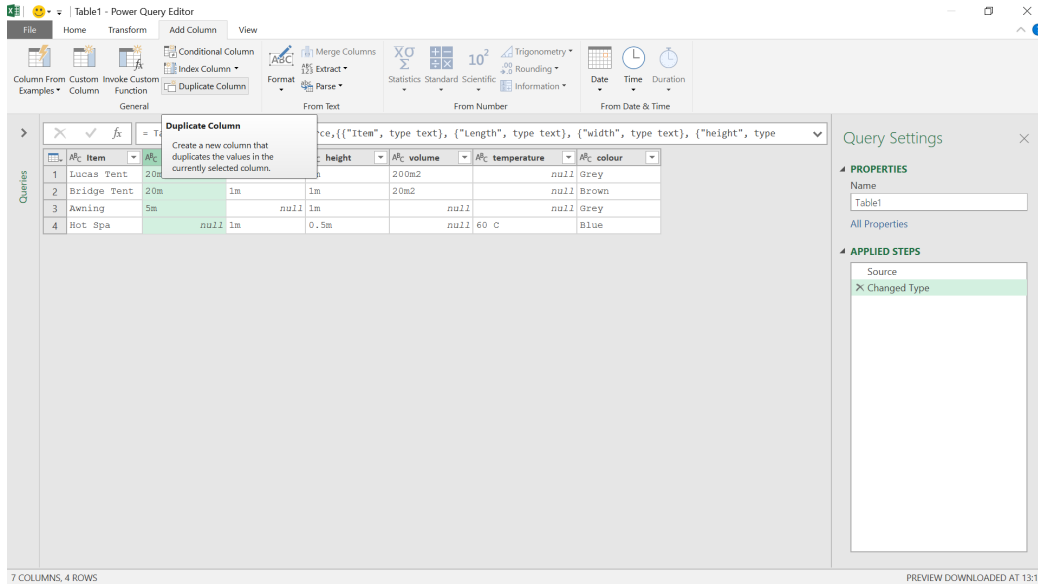
Let's choose to separate the data by commas, and we will call the new column **Dimensions**.



There are several problems here:

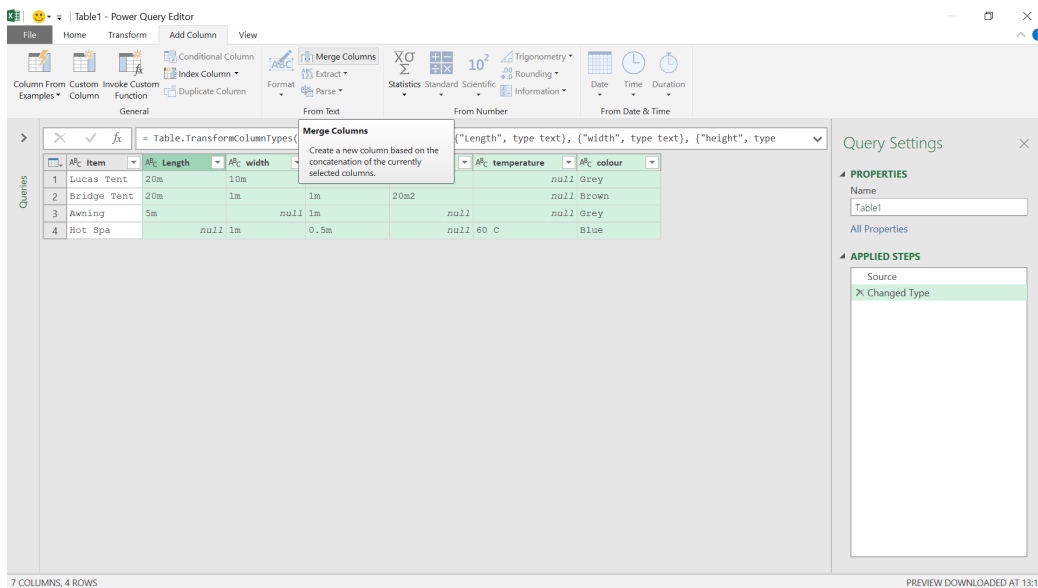
1. The original columns have disappeared
2. The data is not in the order we would have expected
3. There are extra commas.

The first problem could be resolved by copying my columns before we merge them:

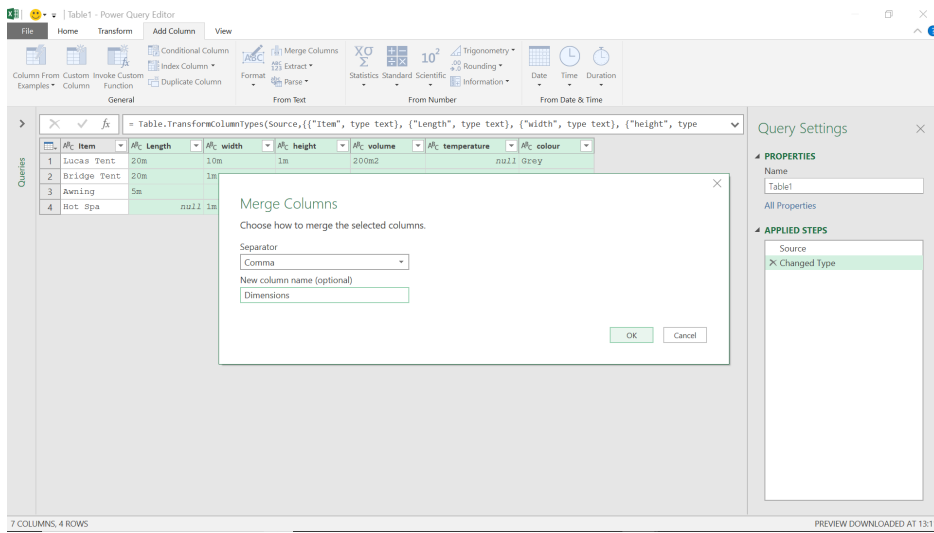


In the 'Add Column' tab, we may create a duplicate of the selected column. We have to do this one column at a time, but having created the duplicates we can select them instead of the original columns.

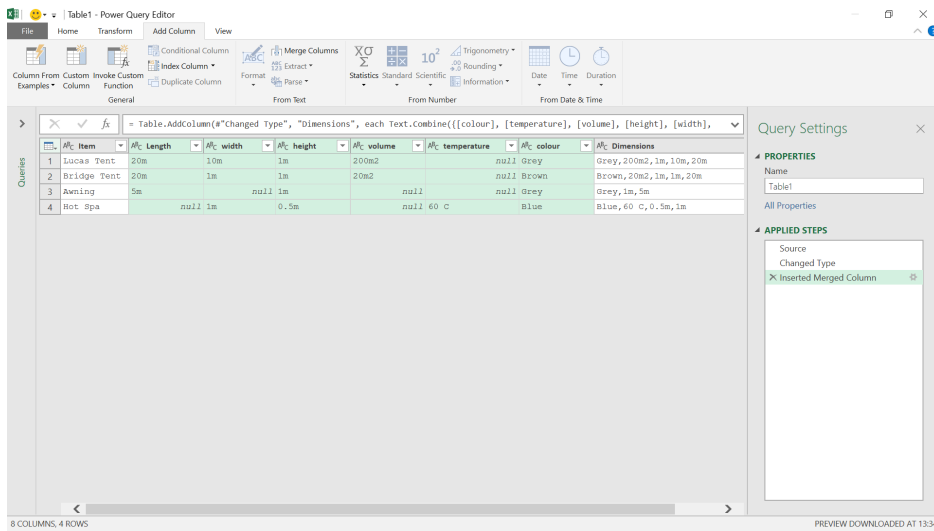
Surprisingly, there is an easier way: we may merge columns from the 'Add Column' tab instead:



We can merge columns here too, but the process behaves slightly differently.

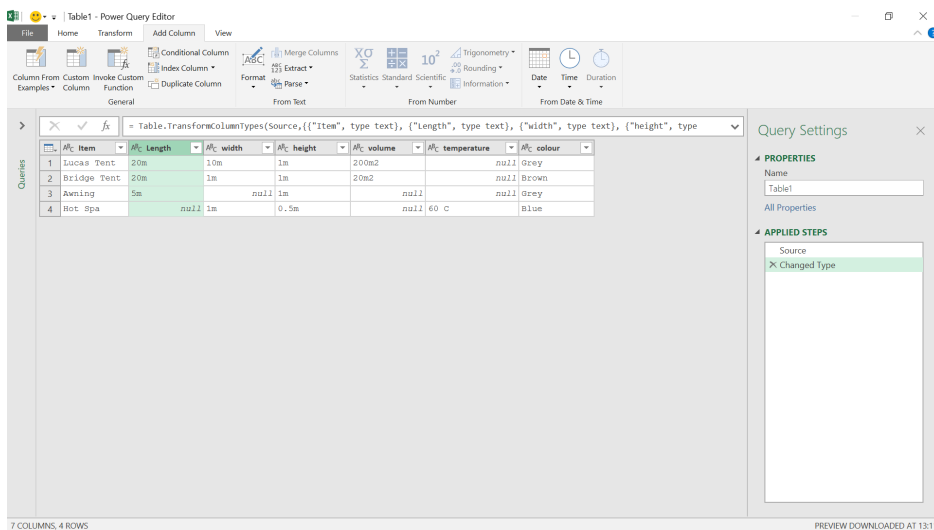


This part looks the same; we can pick the separator and the name of the new column.

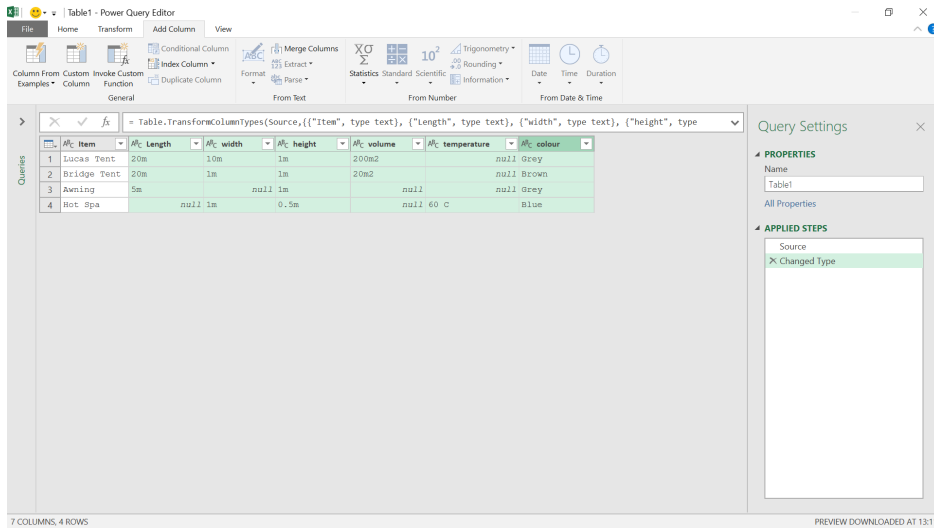


However, this time we get to keep our original columns. Confusing, but useful when we know that 'Merge Columns' on the 'Transform' tab deletes the original columns, whereas 'Merge Columns' from the 'Add Column' tab keeps the originals.

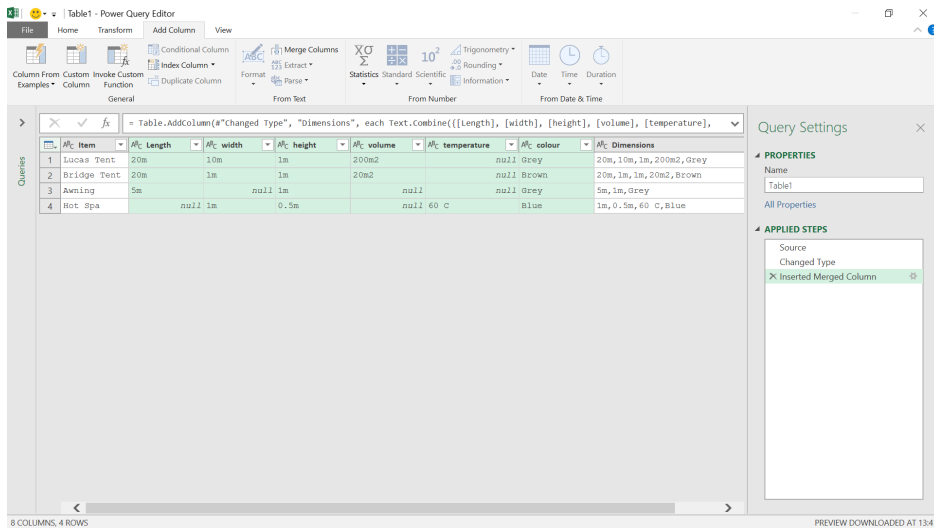
They are still in the wrong order though, and now we can see the original columns, we may work out the order. The merge is constructed in the order in which we select our columns. We selected **colour** first and worked backwards, so our merge does too. We may rectify this by selecting our columns in the order we want them to merge.



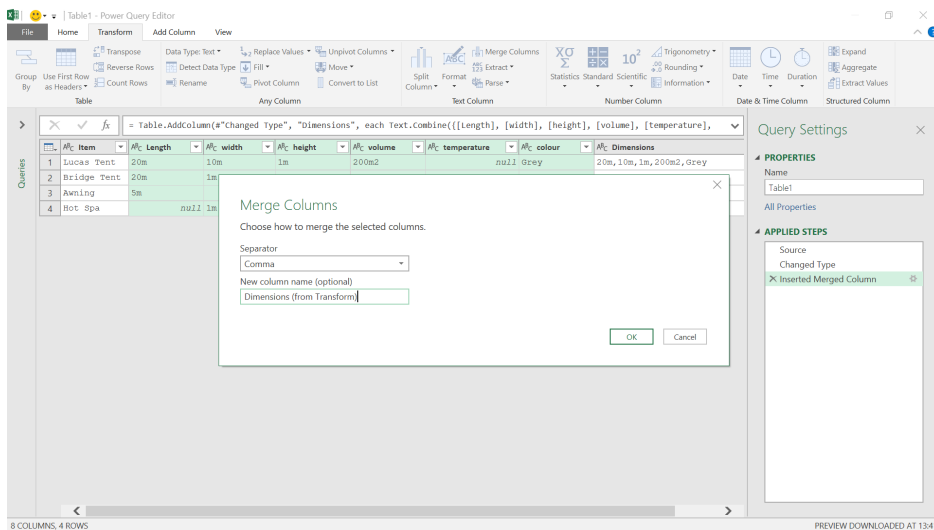
Having selected **Length**, we can select our remaining columns in the right order by holding down the **SHIFT** key and then selecting **colour**. This is quicker and will ensure the order is right.



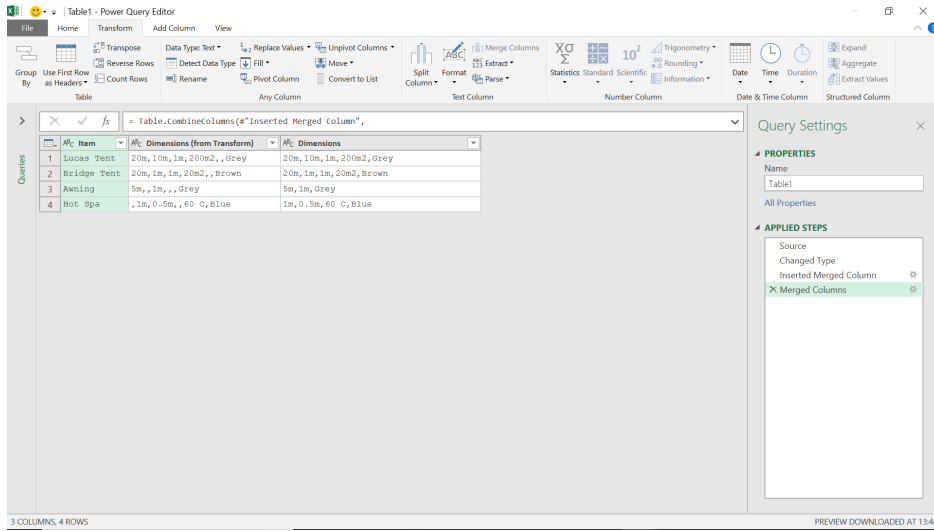
Having created our merge (from the 'Add Column' tab), we check the order again.



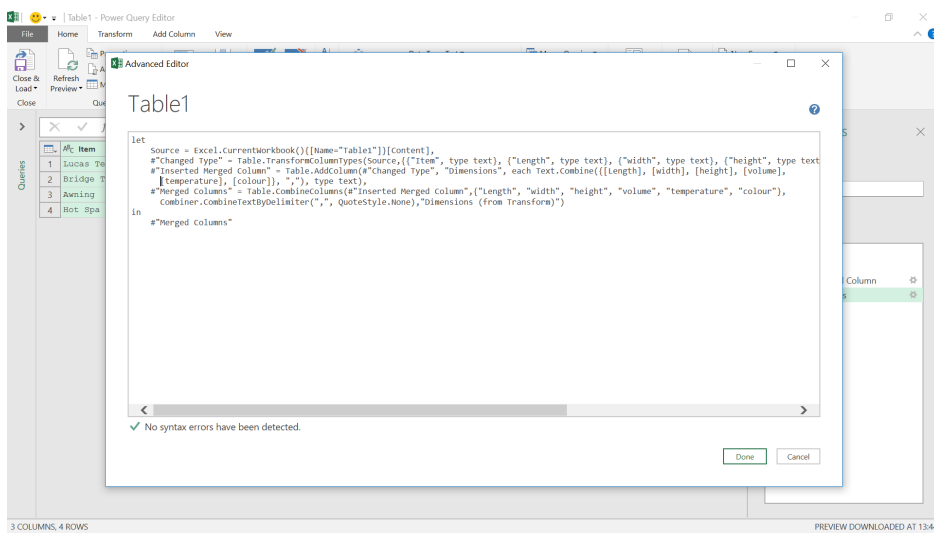
The order is correct, and looking more closely, our last problem (extra commas) has also been resolved! This is because there are more differences to the way the 'Merge Columns' option on the 'Add Column' tab works compared to the 'Transform' tab. To show the difference in the results, we will merge the columns from the 'Transform' tab:



We give the column a different name so we may see what is happening.



The 'Merge Column' from the 'Transform' tab does not cope with the *null* values, but the 'Merge Column' from the 'Add Column' tab does!



The answer to this discrepancy is in the Advanced Editor, where we can see what each 'Merge Columns' has done in M. First of all, let's consider the 'Transform' tab version:

#\"Merged Columns\" = Table.CombineColumns(#\"Inserted Merged Column\",{\"Length\", \"width\", \"height\", \"volume\", \"temperature\", \"colour\"}, Combiner.CombineTextByDelimiter(\" \", QuoteStyle.None),\"Dimensions (from Transform)\")

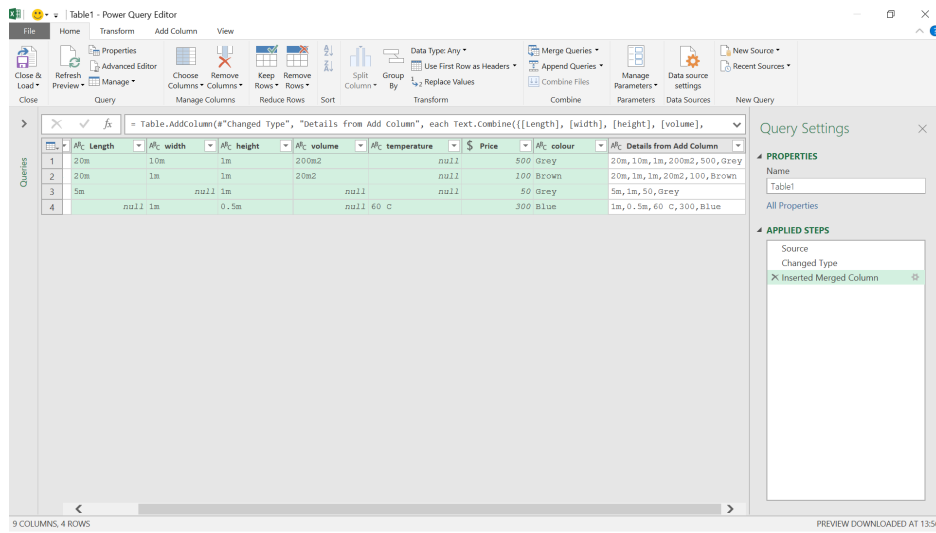
This version of the merge has created a new column using the M function **Combiner.CombineTextByDelimiter**. Clearly, this function does not cope with *null* values.

The 'Add Column' tab version has used a different M function:

#\"Inserted Merged Column\" = Table.AddColumn(#\"Changed Type\", \"Dimensions\", each Text.Combine({[Length], [width], [height], [volume],[temperature], [colour]}, \" \"), type text)

This time, the M function used is **Text.Combine**, which does cope with the *null* values.

This does leave the question of whether the 'Add Column' merge column feature will cope equally well with numerical columns...



The answer is yes, the price is included too, and looking at the **M** code, we can see why:

```
= Table.AddColumn("#Changed Type", "Details from Add Column", each Text.Combine({Length, [width], [height], [volume], [temperature], Text.From([Price], "en-GB"), [colour]}, ", "), type text)
```

The price is converted to text so that it can be combined with the other columns.

In summary, if merging columns, it's currently best to do so from the 'Add Column' tab. It keeps the original columns and copes with *null* values. This is correct at the time of writing, but we wouldn't be surprised to find that the 'Transform' tab is soon updated to use the same functionality for the 'Merge Columns' option. We still have to remember to add the columns in the order we want to merge them, but allowing that functionality means that we don't need to reorder my columns if we do need them to be merged in a different order.

Until next month.

Power BI Updates



Microsoft has announced that their Power BI team is taking a break (no stamina, we're telling you!). They state that their planned updates will roll out next month including the next Power BI Desktop release.

Take a well-earned rest, gang, and we will report their latest changes as usual in next month's newsletter.

New Features for Excel

This month introduces new features in Copilot in Excel that can transform data analysis, assist with conditional formatting, plus a new Excel chat helper. RegEx functions are now rolling out to Windows and Mac users too.

The full list is as follows:

Excel for Windows, Mac and the web

- Copilot in Excel: transforming data analysis
- Conditional formatting with Copilot in Excel
- Copilot in Excel: Excel Chat Helper

Excel for Windows and Mac

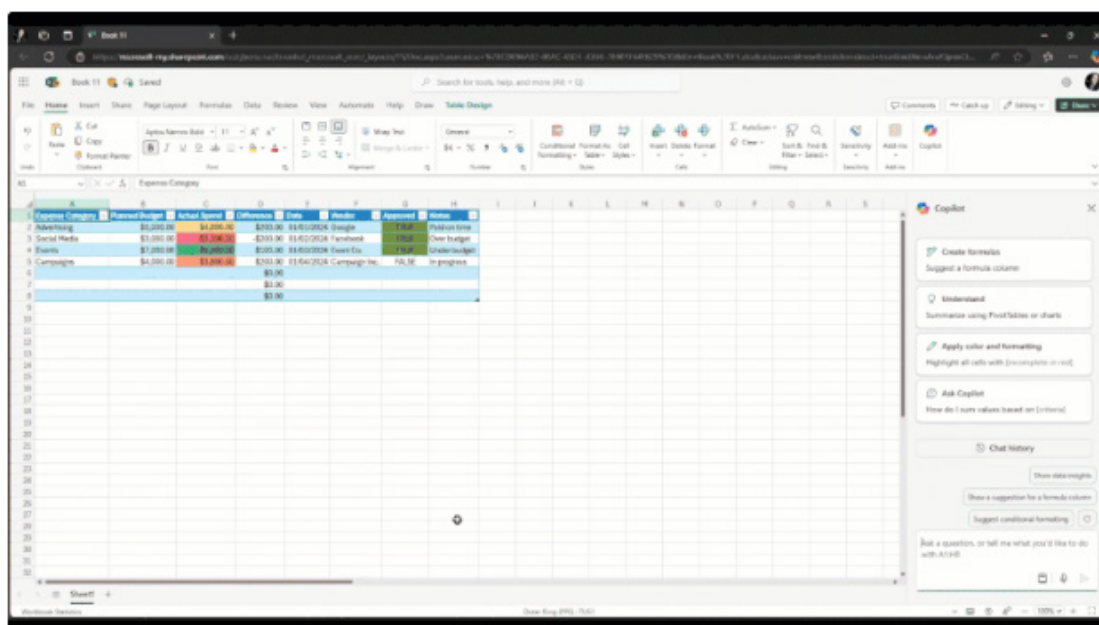
- New Regular Expression (RegEx) functions

Excel for Mac

- Recent widgets for Word, Excel and PowerPoint (Insiders).

Let's get started.

Copilot in Excel: transforming data analysis



Copilot in Excel has been Generally Available for a while now. However, additional features keep rolling out. Here are this month's for Excel for Windows, Mac and the web:

- Copilot in Excel with Python is now Generally Available in US (EN-US) for Windows
- you can create a table (Table?) specific to your needs with Copilot
- Copilot helps you pull in data from your organisation and search the web
- Copilot's enhanced text analysis capabilities provide new ways to reason over your data and derive insights.

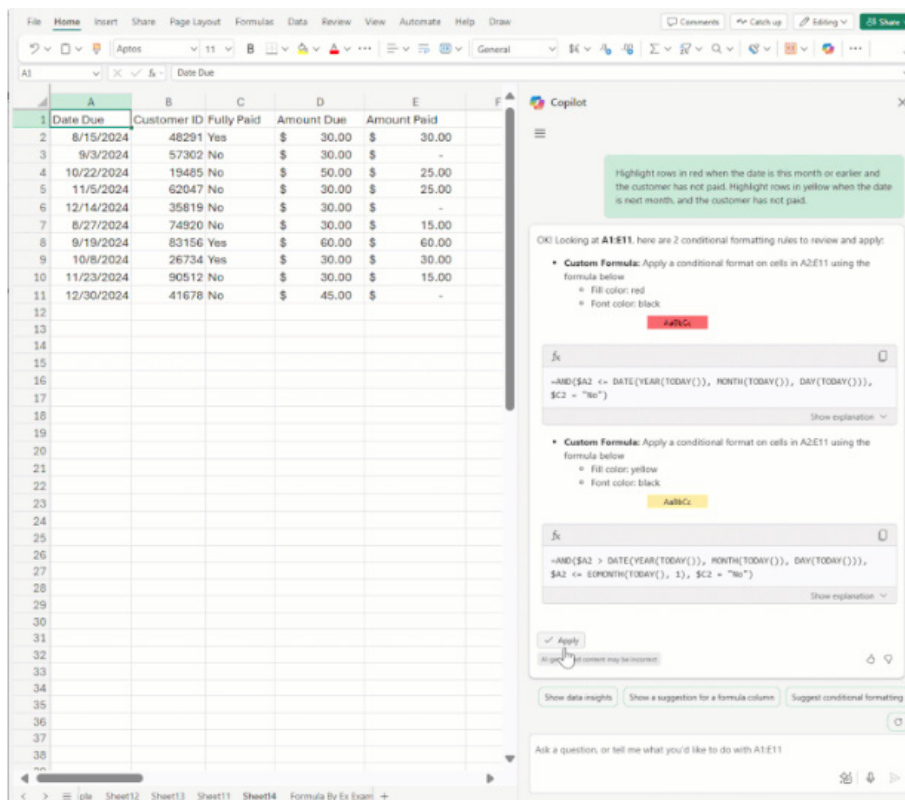
Conditional formatting with Copilot in Excel

Conditional formatting in Excel is a well-known, frequently used tool that allows you to apply specific formatting to cells that meet certain criteria. This can include changing the cell's background colour, font colour or adding icons to highlight differences in data. By using conditional formatting, you can help data to stand out and emphasise what's important, applying your rules automatically as your data changes and grows.

Using a formula to determine which cells to format is one of the most versatile conditional formatting tools. However, it can be challenging to set it up to do exactly what you want. You must get the formula and

syntax exactly right without having the tools and tips of the Formula bar or seeing interim results in cells. With Copilot, you can simply use language to describe exactly what you want to happen.

Microsoft provides the following example. Imagine a user knows precisely how they wish to colour their data. They want to flag certain info in different ways based on a payment status and a date. If they use Copilot, they could ask, "Highlight rows in red when the date is this month or earlier and the customer has not paid. Highlight rows in yellow when the date is next month and the customer has not paid".

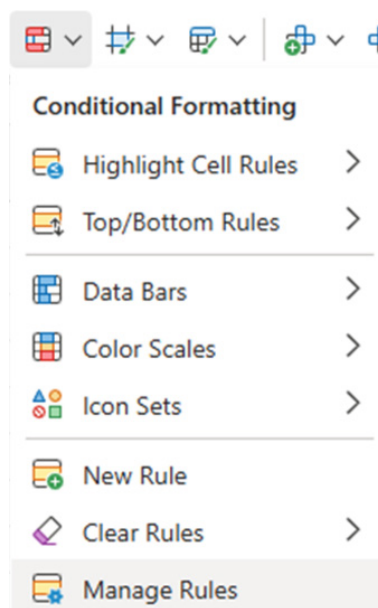


In this example above, Copilot has understood the table of data, interpreted the prompt and responded with two conditional formatting for review and application. You do not need to know the logical order, need formulae or exact syntax to get this result.

You can also ask Copilot to do other types of highlighting, e.g.

- make values in 'Column Name' greater than 'number' have white text on a black background
- highlight cells in light blue for 'Column Name' that contain 'specific text'
- highlight the top 10% of values in 'Column Name' using bold font
- Apply a Red and Green colour scale to the values in 'Column Name'.

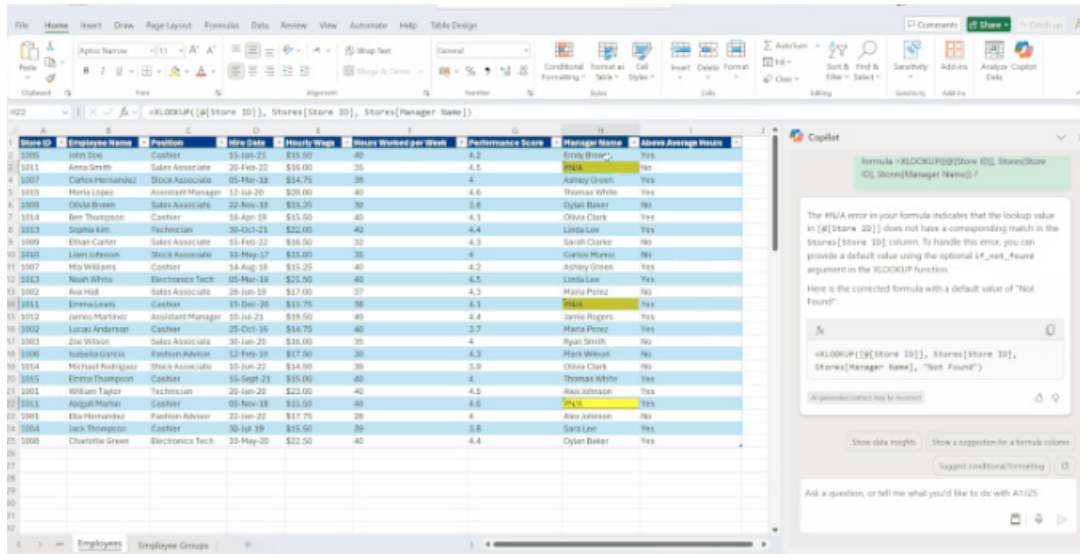
When you have applied any conditional formatting rules using Copilot, you can manage and edit existing rules by choosing **Conditional Formatting** -> **Manage Rules** from the toolbar or Ribbon.



Conditional formatting in Excel is a versatile tool that can help you analyse and present your data more effectively. By exploring the various capabilities of conditional formatting in combination with Copilot you can unlock the full potential of this feature and make your spreadsheets more informative and visually engaging.

We may also use Copilot to assist with formulaic issues we don't know how to fix or edit an existing formula to do something different, e.g.

- Can you suggest a fix for the #N/A error in this formula =XLOOKUP([@[Store ID]], Stores[Store ID], Stores[Manager Name])?
- I would prefer it if the formula would show an empty cell instead of "Not Found". Can you help me modify it?



Copilot may be a simple resource for mastering formulae when you are starting out, simplifying complex tasks, and enhancing your productivity.

New Regular Expression (Regex) functions

Regular expressions, or 'RegEx' / 'regex' are sequences of characters that define search patterns, commonly used for string searching and text parsing. They are incredibly versatile and are often used to check if a string contains a certain pattern, extract substrings that match the pattern or replace substrings that match the pattern.

The new regex functions being rolled out to Excel for Windows and Mac are:

- **REGEXTEST:** checks whether any part of supplied text matches a regex pattern
- **REGEXEXTRACT:** extracts one or more parts of supplied text that match a regex pattern
- **REGEXREPLACE:** searches for a regex pattern within supplied text and replaces it with different text.

We have explained these functions in previous newsletters, but for those who missed it and / or for completion, we reproduce the article below.

Microsoft has stated that the version of Regex coming to Excel uses a "flavor" (*sic*) called **PCRE2 (PHP>=7.3)** for those that need to know the underlying technical stuff.

Clearly, we need to learn a little about "regular expressions" before continuing. Alternatively referred to "rational expressions" upon occasion, a regular expression is a sequence of characters that specifies what is known as a "match pattern" in text. You have most likely used this functionality in Excel already, with features such as "Find and Replace" or by using the **FIND** or **SEARCH** functions in Excel. The purpose of these three [3] new functions (presumably, this is just a start!) is to help you match, locate and manage text (strings) in Excel.

The text is obvious but understanding patterns requires you to learn the syntax for regular expressions. Here is a crash course table, which summarises some – but not all – of the main elements, usually referred to as "tokens".

Token	Meaning
\	This converts special characters (metacharacters) to literal characters, and also allows the literal matching of the regex delimiter in use, e.g. '/'
.	Matches any character other than newline
^	Matches the start of string without consuming any characters. If multiline mode is used, this will also match immediately after a newline character
\$	Matches the end of string without consuming any characters. If multiline mode is used, this will also match immediately before a newline character
a?	Matches zero [0] or one [1] of a. This matches an 'a' character or nothing
a*	Matches zero [0] or more of a. This matches zero or consecutive 'a' characters
a+	Matches one [1] or more of a. This matches consecutive 'a' characters
a{4}	Matches exactly four [4] instances of 'a'
a{4,}	Matches four [4] or more instances of 'a'

Token	Meaning
a{4,6}	Matches between four [4] and six [6] instances of 'a'
\A	Matches the start of a string only. Unlike ^, this is not affected by multiline mode
\Z	Matches the end of a string only. Unlike \$, this is not affected by multiline mode
\z	Matches the absolute end of a string only. Unlike \$, this is not affected by multiline mode and in contrast to \Z, this will not match before a trailing newline at the end of a string
\b	Matches a word boundary. It matches without consuming any characters, immediately between a character matched by \w and a character not matched by \w. It cannot be used to separate non-words from words
\B	Matches a non-word boundary. It matches without consuming any characters, at the position between two characters matched by \w or \W
i	A case insensitive match is performed
x	Ignore whitespace / verbose. This flag instructs the engine to ignore all whitespace and allow for comments in the regex, also known as verbose. Comments are indicated by starting with the # character and then escaping with \
xx	Ignore all whitespace / verbose. Similar to x, but whitespace is also ignored inside of character classes
s	Known as single line, this enables the dot (.) metacharacter to also match newlines, thus treating the whole string as a single line input
\n	Matches a newline character
\N	Matches anything other than a newline character
\r	Matches a carriage return, Unicode character U+2185
\R	Careful! Matches any Unicode newline sequence
\t	Matches a tab character (typically, tab stops happen every eight [8] characters)
\0 [zero]	Matches a <i>null</i> character, Unicode character U+2400
\d	Matches any decimal / digit. Equivalent to [0-9]
\D	Matches anything other than a decimal / digit
\s	Matches any whitespace character (space, tab or newline)
\S	Matches any non-whitespace character (anything other space, tab or newline)
\w	Matches any word character (any letter, digit or underscore). Equivalent to [a-zA-Z0-9_]
\W	Matches any non-word character (anything other than a letter, digit or underscore). Equivalent to [^a-zA-Z0-9_]
[abc]	Matches an 'a', 'b' or 'c' character
[^abc]	Matches any character except 'a', 'b' or 'c'
a b	Alternate match: matches what is before or after , in this case 'a' or 'b'
[a-z]	Matches any characters between a and z inclusive
[^a-z]	Matches any characters, except those in the range a to z inclusive
[a-zA-Z]	Matches any characters between a to z or A to Z inclusive
[:alnum:]	Double square brackets are required here. Matches letters and digits. This is equivalent to [A-Za-z0-9]
[:alpha:]	Matches letters. Equivalent to [a-zA-Z]
[:ascii:]	Matches any character in the valid ASCII range (any basic Latin character). ASCII codes 0 to 127 inclusive
[:blank:]	Matches spaces and tabs (but not newlines). Equivalent to [\t]
[:cntrl:]	Matches characters that are often used to control text presentation, including newlines, <i>null</i> characters, tabs and the escape character

Token	Meaning
<code>[:digit:]</code>	Matches decimal / digits. Equivalent to <code>[0-9]</code> or <code>\d</code>
<code>[:graph:]</code>	Matches visible characters (not space: printable, non-whitespace, non-control characters only)
<code>[:lower:]</code>	Matches lowercase letters. Equivalent to <code>[a-z]</code>
<code>[:print:]</code>	Matches printable characters, part of the basic Latin set, such as letters and spaces, but not including control characters
<code>[:punct:]</code>	Matches visible punctuation characters that are not whitespace, letters or numbers
<code>[:space:]</code>	Matches whitespace characters. Equivalent to <code>\s</code>
<code>[:upper:]</code>	Matches uppercase letters. Equivalent to <code>[A-Z]</code>
<code>[:word:]</code>	Matches word characters (letters, numbers and underscores). Equivalent to <code>\w</code> or <code>[a-zA-Z0-9_]</code>
<code>[:<:]</code>	Matches the start of word
<code>[:>:]</code>	Matches the end of word
<code>(?..)</code>	Match everything enclosed. For example, repeating 1-3 digits and a period 3 times can be identified as follows: <code>/(?:\d{1,3}\.){3}\d{1,3}/</code>
<code>(...)</code>	Capture everything enclosed

Now we are all experts in regex, let's go through the three new functions being rolled out.

REGEXEXTRACT

This function is used extract one or more strings that match a specified pattern from the text being analysed. You may extract the first match, all matches or capturing groups from the first match. Its syntax is as follows:

REGEXEXTRACT(text, pattern, [return_mode], [ignore_case])

It has the following three arguments:

- **text:** this is required, and represents the **text** you are searching within
- **pattern:** this is also required. This is the regular expression to be applied
- **return_mode:** the first of two optional arguments, this specifies which matches to return. It has three [3] alternatives:

Return Mode	Description
0	First match (default)
1	All matches
2	Capture groups of first match

Capturing groups are part of a regular expression ("regex") pattern surrounded by parentheses "(...)". They allow you to return separate parts of a single match individually

- **ignore_case:** the final (optional) argument. This determines whether the match should be case sensitive. It has the following two [2] options:

Ignore Case	Description
0	Case sensitive match (default)
1	Case insensitive match

This function always returns text values. You may convert these results back to numerical values using the **VALUE** function.

Consider the following examples:

	A	B	C
1	Name		
2	Liam Bastick		
3			
4			
5	Formula	Description	Result
6	<code>=REGEXEXTRACT(A2,"[a-zA-Z]+")</code>	Extract first name based upon letters with pattern "[a-zA-Z]+"	Liam
7	<code>=REGEXEXTRACT(A2,"[a-zA-Z]+",1)</code>	Extract all names based upon letters with pattern "[a-zA-Z]+"	Liam
8	<i>Spilled from above</i>		Bastick
9			

	A	B	C
1	Data		
2	Harry Potter (378) 555-4195		
3	Snooker Potter (878) 555-8622		
4	Beatrix Potter (437) 555-8987		
5	Beer Tricks Potter (619) 555-4212		
6	Trixie Potter (579) 555-5658		
7	Bumble Dore (346) 555-0925		
8	Dumble Bore (405) 555-0887		
9	Bumble Bee (666) 555-1872		
10			
11			
12	Formula	Description	Result
13	<code>=REGEXEXTRACT(A2:A9,"[0-9()]+ [0-9-]+",1)</code>		(378) 555-4195
14	<i>Spilled from above</i>		(878) 555-8622
15	<i>Spilled from above</i>		(437) 555-8987
16	<i>Spilled from above</i>	Extracts phone numbers based upon their pattern	(619) 555-4212
17	<i>Spilled from above</i>		(579) 555-5658
18	<i>Spilled from above</i>		(346) 555-0925
19	<i>Spilled from above</i>		(405) 555-0887
20	<i>Spilled from above</i>		(666) 555-1872
21			

REGEXREPLACE

The **REGEXREPLACE** function replaces strings within the provided text that matches the pattern with replacement. The syntax of the **REGEXREPLACE** function is:

REGEXREPLACE(text, pattern, replacement, [occurrence], [case_sensitivity])

where:

- **text**: this is required, and represents the **text** or the reference to a cell containing the **text** you wish to replace strings within
- **pattern**: this is also required. This is the regular expression ("regex") that describes the pattern you wish to replace
- **replacement**: another required argument, this is the text you wish to replace instances of **pattern**
- **occurrence**: the first of two optional arguments, this specifies which instance of the pattern you wish to replace. By default, **occurrence** is zero [0], which will replace all instances. It should be noted that a negative number replaces that instance, searching from the end instead
- **case_sensitivity**: the final (optional) argument. This determines whether the match should be case sensitive. It has the following two [2] options:

Case Sensitivity	Description
0	Case sensitive match (default)
1	Case insensitive match

This function always returns text values. You may convert these results back to numerical values using the **VALUE** function.

Consider the following examples:

	A	B	C
1	Name		
2	LiamBastick		
3			
4			
5	Formula	Description	Result
6	=REGEXREPLACE(A2,"([A-Z][a-z]+)([A-Z][a-z]+)","\$2, \$1")	Capturing groups to separate and reorder first and last names	Bastick, Liam
7			

	A	B	C
1	Data		
2	Harry Potter (378) 555-4195		
3	Snooker Potter (878) 555-8622		
4	Beatrix Potter (437) 555-8987		
5	Beer Tricks Potter (619) 555-4212		
6	Trixie Potter (579) 555-5658		
7	Bumble Dore (346) 555-0925		
8	Dumble Bore (405) 555-0887		
9	Bumble Bee (666) 555-1872		
10			
11			
12	Formula	Description	Result
13	=REGEXREPLACE(A2:A9,"[0-9]+","***-")		Harry Potter (378) ***-4195
14	Spilled from above		Snooker Potter (878) ***-8622
15	Spilled from above		Beatrix Potter (437) ***-8987
16	Spilled from above	Used to anonymise phone numbers by replacing the first three digits of main number with ***	Beer Tricks Potter (619) ***-4212
17	Spilled from above		Trixie Potter (579) ***-5658
18	Spilled from above		Bumble Dore (346) ***-0925
19	Spilled from above		Dumble Bore (405) ***-0887
20	Spilled from above		Bumble Bee (666) ***-1872
21			

REGEXTEST

The **REGEXTEST** function allows you to check whether any part of supplied text matches a regular expression ("regex"). It will return TRUE if there is a match and FALSE otherwise. The syntax of the **REGEXTEST** function is:

REGEXTEST(text, pattern, [case_sensitivity])

where:

- **text:** this is required, and represents the **text** or the reference to a cell containing the **text** you wish to match against
- **pattern:** this is also required. This is the regular expression ("regex") that you wish to match
- **case_sensitivity:** the final (optional) argument. This determines whether the match should be case sensitive. It has the following two [2] options:

Case Sensitivity	Description
0	Case sensitive match (default)
1	Case insensitive match

This function always returns text values. You may convert these results back to numerical values using the **VALUE** function.

Consider the following examples:

	A	B	C
1	Data		
2	Liam.Bastick@sumproduct.com		
3			
4			
5	Formula	Description	Result
6	<code>=REGEXTEST(A2,"@[a-zA-Z0-9_+]+\.[a-zA-Z0-9_+]"</code>)	Is this an email address?	TRUE
7			

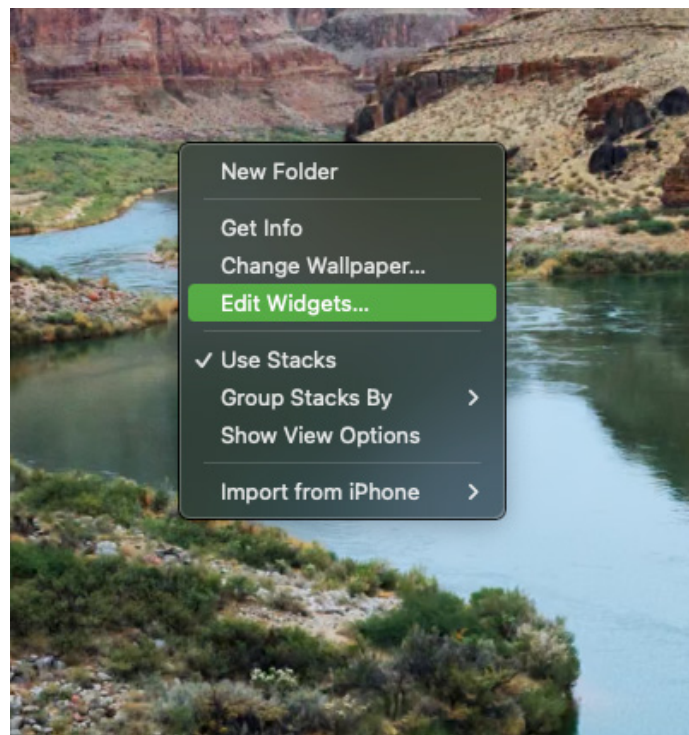
	A	B	C
1	Data		
2	alfalfa		
3			
4			
5	Formula	Description	Result
6	<code>=REGEXTEST(A2,"a")</code>	Does the string contain the letter 'a'?	TRUE
7	<code>=REGEXTEST(A2,"[a-z]"</code>)	Are there any lower case letters?	TRUE
8	<code>=REGEXTEST(A2,"[A-Z]"</code>)	Are there any upper case letters?	FALSE
9	<code>=REGEXTEST(A2,"[aeiou]"</code>)	Are there any vowels?	TRUE
10	<code>=REGEXTEST(A2,"[0-9]"</code>)	Does the string contain digits?	FALSE
11			

Recent widgets for Word, Excel and PowerPoint (Insiders)

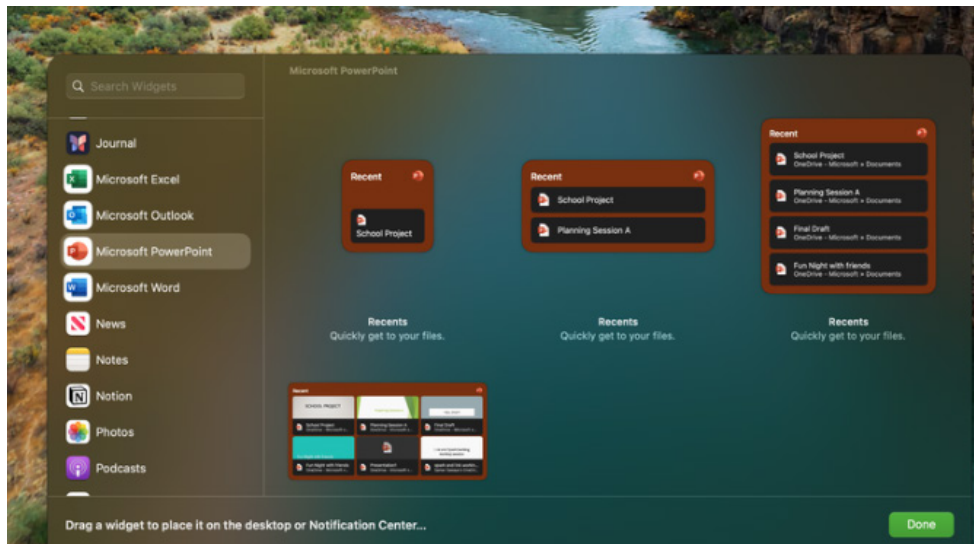
The Recent widgets added to iOS have made their way to Excel for Mac.

You may now add Recent widgets for Word, Excel and PowerPoint directly to your Mac desktop. The widgets allow you to both view and open your most recently accessed files from the desktop.

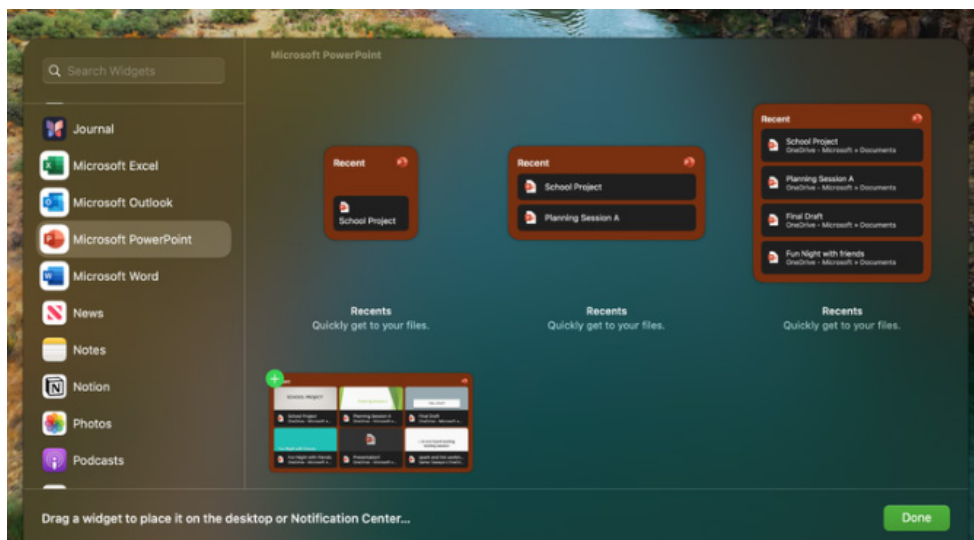
On your Mac, right-click your desktop and select the 'Edit Widgets' command:



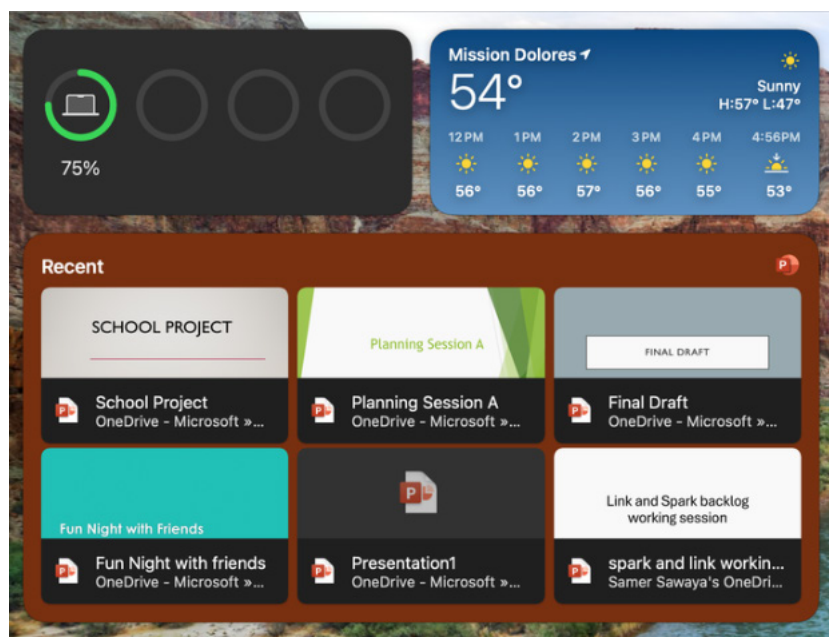
Then, select the app widget you want from the resulting list:



Select the size of widget you want and then hover over it and select the green + button in the top left corner.



To access one of your recent files from your desktop, simply click the File card:



The Recent widget offers four [4] sizes to choose from, ranging from small to extra-large. If you would like to open the application's home page instead of a recent file, click anywhere on the widget outside of the File cards.

It should be noted that this feature is rolling out to Current Channel (Preview) users of Excel for Mac running Version 16.91 (Build 24110320) or later. Until next month.

The A to Z of Excel Functions: OFFSET

The older I get the more invaluable **OFFSET** becomes. The syntax for **OFFSET** is as follows:

OFFSET(reference, rows, columns, [height], [width]).

The arguments in square brackets (**height** and **width**) may be omitted from the function. The default values are the same dimensions as the original **reference**.

In its most basic form, **OFFSET(ref, x, y)** will select a reference **x** rows down (**-x** would be **x** rows up) and **y** columns to the right (**-y** would be **y** columns to the left) of the reference **ref**. For example, consider the following grid:

	A	B	C	D	E	F
1	1	2	3	4	5	6
2	7	8	9	10	11	12
3	13	14	15	16	17	18
4	19	20	21	22	23	24
5	25	26	27	28	29	30
6	31	32	33	34	35	36

OFFSET(A1,2,3) would take us two rows down and three columns across to cell **D3**. Therefore, **OFFSET(A1,2,3) = 16**, viz.

	A	B	C	D	E	F
1	1	2	3	4	5	6
2	7	8	9	10	11	12
3	13	14	15	16	17	18
4	19	20	21	22	23	24
5	25	26	27	28	29	30
6	31	32	33	34	35	36

OFFSET(D4,-1,-2) would take us one row up and two rows to the left to cell **B3**. Therefore, **OFFSET(D4,-1,-2) = 14**, viz.

	A	B	C	D	E	F
1	1	2	3	4	5	6
2	7	8	9	10	11	12
3	13	14	15	16	17	18
4	19	20	21	22	23	24
5	25	26	27	28	29	30
6	31	32	33	34	35	36

We can use these mechanics to construct a very simple scenario table:

		1	2	3	4	5
Key Data For Model						
Scenario Selected:		1				
Assumption Description	Amts Used					
Base Year Unit Price	\$3.70	\$3.70	\$4.15	\$3.00	\$2.85	\$2.99
Unit Price Growth Rate	2.0%	2.0%	3.0%	4.0%	5.0%	6.0%
Base Year Volume	80,000.0	80,000.0	85,000.0	82,500.0	77,900.0	83,000.0
Volume Increase p.a.	200.0	200.0	250.0	300.0	200.0	150.0
Unit Cost Price	\$1.00	\$1.00	\$1.30	\$1.10	\$1.25	\$1.29
Unit Cost Growth Rate	3.0%	3.0%	3.0%	3.0%	3.0%	3.0%
Tax Rate	28.0%	28.0%	30.0%	30.0%	28.0%	33.0%

Essentially, the assumptions used in this illustration are linked from cells **J14:J20** (in yellow). These values are drawn from the scenario table to the right of the highlighted yellow range (e.g. cells **L14:L20** constitute Scenario 1, cells **M14:M20** constitute Scenario 2).

The Scenario Selector is located in cell **J11**. Using **OFFSET** scenarios may be selected at will. For example, the formula in cell **J14** is simply **OFFSET(K14,,,\$J\$11)**, that is, start at cell **K14** and displace zero rows down and the value in **J11** columns across. In the image above, the formula locates the cell one column to the right, which is Scenario 1.

The advantage of **OFFSET** over other functions such as **INDEX**, **CHOOSE** and **LOOKUP** functions is that the range of data can be added to. Whilst the other functions require a specified range whereas we can keep

adding scenarios without changing the formula / making the model inefficient.

Furthermore, **OFFSET** can be used for other practical uses in Excel, taking advantage of the **height** and **width** arguments. Consider the **OFFSET** example from earlier. If we extend the formula to **OFFSET(D4,-1,-2,-2,3)**, it would again take us to cell **B3** but then we would select a range based on the **height** and **width** parameters. The **height** would be two rows going up the sheet, with row 14 as the base (i.e. rows 13 and 14), and the **width** would be three columns going from left to right, with column **B** as the base (i.e. columns **B**, **C** and **D**).

Hence **OFFSET(D4,-1,-2,-2,3)** would select the range **B2:D3**, viz.

	A	B	C	D	E	F
1	1	2	3	4	5	6
2	7	8	9	10	11	12
3	13	14	15	16	17	18
4	19	20	21	22	23	24
5	25	26	27	28	29	30
6	31	32	33	34	35	36

Note that **OFFSET(D4,-1,-2,-2,3)** equals **#VALUE!** in legacy Excel, since this version of Excel cannot display a matrix in one cell (Excel 365 can as it supports dynamic arrays), but it does recognise it. However, if after typing in **OFFSET(D4,-1,-2,-2,3)** we press **CTRL + SHIFT + ENTER** in older versions, we turn the formula into an array formula:

{OFFSET(D4,-1,-2,-2,3)}

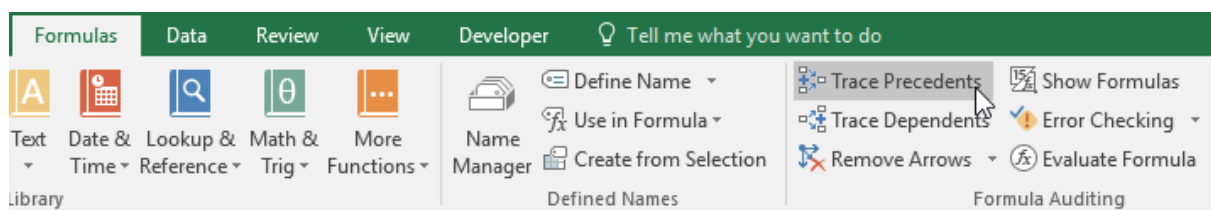
(do not type the braces in, they will appear automatically as part of the Excel syntax). This gives a value of eight [8], which is the value in the top left-hand corner of the matrix, *but Excel is storing more than that*. This can be seen as follows:

- **SUM(OFFSET(D4,-1,-2,-2,3)) = 72** (i.e. **SUM(B2:D3)**)
- **AVERAGE(OFFSET(D4,-1,-2,-2,3)) = 12** (i.e. **AVERAGE(B2:D3)**).

Indeed, you may construct a simple depreciation calculation or transpose references using **OFFSET**'s **height** and **width** functionalities. But more on that later...

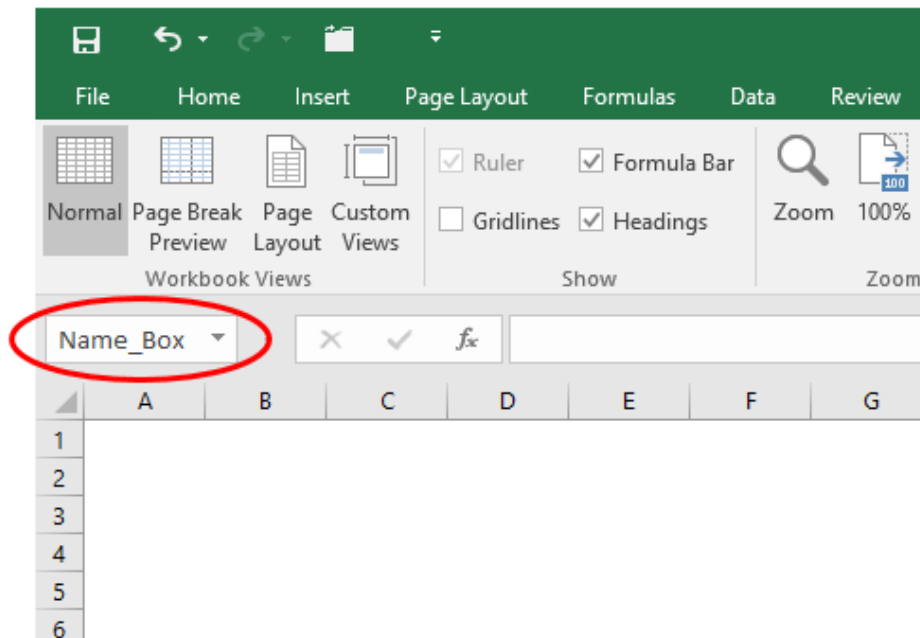
There are a couple of problems with **OFFSET**:

- values returned by an **OFFSET** function confuse Excel. Only the original **reference** is recognised as a precedent reference to the formula by Excel's auditing tools.



The result returned is most likely to come from another cell which will not be highlighted by this technique. If you think about it, this actually makes sense as potentially all of the cells on a worksheet are potential precedents.

To take account of this, I suggest you give the **reference** a **range name**. Range names are discussed in detail later in this book, but for now to name a cell, click on the cell and then type the desired name in the 'Name box' in Excel:



This range name should start with **BC_**. This prefix stands for “Base Cell” and makes it easier to sort / locate range names later. When users or model auditors alike inspect a formula with a **Reference** starting with **BC_** for **Base Cell** (e.g. **BC_Example_Reference**), this can alert them to the fact that the model may be using cells in the region of this **Reference** that do not appear to have any dependents.

- The other issue is that **OFFSET** is what is known as a **volatile** function. A volatile function is one that causes recalculation of the formula in the cell where it resides every time Excel recalculates. This can really slow down your model if there are too many **OFFSET** functions, for example.

Aside: Volatile Functions

As stated above, a **volatile function** is one that causes recalculation of the formula in the cell where it resides every time Excel recalculates. This occurs regardless of whether precedent cells / calculations have changed, or whether the formula also contains non-volatile functions. One test to check whether your workbook is volatile is close a file after saving and see if Excel prompts you to save it a second time (this is an indicative test only). This can really slow down your model if there are too many **OFFSET** functions, for example.

Just because a function is volatile in one version of Excel does not mean it is volatile in all versions. Perhaps the best example of this is **INDEX**, which was volatile prior to Excel 97. Microsoft still states this function is volatile, but this does not appear to be the case except when used as the second part of a range reference, for example **=\$A\$1:INDEX(\$A\$2:A\$10,4)**, will also cause the reference to be flagged as “dirty” (i.e. needs to be recalculated) when the workbook is opened only.

Another common ‘semi-volatile’ function is **SUMIF**, which has been so since Excel 2002. This function becomes volatile whenever the size of the first range argument is not the same as the second (**sum_range**) argument, e.g. **SUMIF(A1:A4,1,B1)** is volatile whereas **SUMIF(A1:A4,1,B1:B4)** is not.

IF and **CHOOSE** do not calculate all arguments, aabut iaaaaf any of the arguments are volatile – regardless of whether they are used – the formula is deemed to be volatile. Therefore, **IF(1>0,1,RAND())** is always volatile, even though the **value_if_false** argument will never be calculated. It is not quite as simple as this though. If the formula in cell **A1** is **=NOW()** then this cell will be volatile, but **IF(1>0,1,A1)** will not be.

In essence, direct references or dependents of volatile functions will always be recalculated, whereas indirect ones will only recalculate when activated or in certain other functions that always calculate all arguments such as **AND** and our next version **OR**...

The A to Z of Excel Functions: OR

The **OR** function is similar to **AND** but only requires one condition to be TRUE. Similar to **AND**, the **OR** function may be used to expand the usefulness of other functions that perform logical tests. For example, the **IF** function performs a logical test and then returns one value if the

test evaluates to TRUE and another value if the test evaluates to FALSE. By using the **OR** function as the **logical_test** argument of the **IF** function, you can test many different conditions instead of just one.

For example, imagine you are in London on a Tuesday. Consider the expression

=OR(condition1, condition2, condition3)

where:

- **condition1** is the condition, “today is Tuesday”
- **condition2** is the condition, “you are awake” and
- **condition3** is the condition, “the Earth is flat”.

This would clearly be TRUE as you should be awake if you are reading this (that is, **condition2** holds).

The syntax for **OR** is as follows:

OR(logical1, [logical2], ...)

where:

- **logical1**: the first condition that you want to test that can evaluate to either TRUE or FALSE
- **logical2**: additional conditions that you want to test that can evaluate to either TRUE or FALSE, up to a maximum of 255 conditions. **logical2** is optional and is not needed in the syntax.

It should be noted that:

- the arguments must evaluate to logical values, such as TRUE or FALSE, or the arguments must be arrays or references that contain logical values
- if an array or reference argument contains text or empty cells, those values are ignored
- if the specified range contains no logical values, the **OR** function returns the #VALUE! error value.

In summary, **OR** works as follows:

	A	B	C
1	Condition 1	TRUE	
2	Condition 2	FALSE	
3	Condition 3	FALSE	
4			
5			
6	Description	Results	Formula
7	At least one argument is true	TRUE	=OR(B1:B3)
8	All arguments are false	FALSE	=NOT(OR(B1:B3))
9	At least one argument is false	TRUE	=NOT(AND(B1:B3))
10			

More Excel Functions next month.

Beat the Boredom Suggested Solution

Earlier in this newsletter, we asked you to imagine you had a Table named **Data** in Excel, containing a list of names under the column **Name** and corresponding numerical data under the column **Grade** as follows:

The screenshot shows an Excel spreadsheet with the following data:

Name	Grade
Charlie Taylor	6
Ellie Nguyen	8
Eloise Graham	7
Emily White	10
Ezra Gray	5
Freya Ward	5
Harrison Adams	10
Isabelle Davies	3
Marcus Graham	7
Matilda Scott	7
Olive Gray	9
Piper James	8
Samuel Gray	7
Scarlett Wilson	7

Below the table, the 'Between' field contains 'Emily White' and the 'and' field contains 'Olive Gray'. The formula bar shows: `=XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade])`

Your task was to write a single Excel formula that summed the **Grade** data between two [2] names exclusively, which we referred to as **Name_1** (Cell **G28**) and **Name_2** (Cell **G29**). These names were inputs that could be changed, and the sum was to dynamically update to reflect the range of data between these two [2] names in the **Data** table. You could download the original question file [here](#).

As always, there were some requirements:

- the formula needs to be within just one [1] column (no “helper” cells)
- the solution must work even if the order of **Name_1** and **Name_2** are swapped in the list
- the formula should be dynamic so that they should update when a new entry was added.

Suggested Solution for Modern Excel Users (Excel 365 and Later Versions)

The steps are detailed below.

Crafting a Data Range with XLOOKUP and Colon

We initiate our formula with the **XLOOKUP** function, deploying it twice to accurately retrieve the grades corresponding to **Name_1** and **Name_2**.

=XLOOKUP(Name_1,Data[Name],Data[Grade])

=XLOOKUP(Name_2,Data[Name],Data[Grade])

This will return us the respective grades for **Name_1** and **Name_2**:

Data

Name	Grade
Charlie Taylor	6
Ellie Nguyen	8
Eloise Graham	7
Emily White	10
Ezra Gray	5
Freya Ward	5
Harrison Adams	10
Isabelle Davies	3
Marcus Graham	7
Matilda Scott	7
Olive Gray	9
Piper James	8
Samuel Gray	7
Scarlett Wilson	7

Between	Emily White
and	Olive Gray

10 =XLOOKUP(Name_1,Data[Name],Data[Grade])
 9 =XLOOKUP(Name_2,Data[Name],Data[Grade])

The real transformation occurs when we introduce a colon ‘:’ between these two [2] **XLOOKUP** functions. This action forms a dynamic array that spans from the grade of **Name_1** to that of **Name_2**.

=XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade])

The screenshot shows an Excel spreadsheet with the following content:

- Row 1:** **SUM IF between**
- Row 2:** SP SUM IF between 2 Names.xlsm
- Row 3:** Navigator
- Row 4:** Error Checks:
- Row 6:** **1. SUM IF between**
- Row 8:** **Data**
- Row 10-24:** The 'Data' table from the previous block.
- Row 27-29:** Input cells for 'Between' (Emily White) and 'and' (Olive Gray).
- Row 14-24:** A column of results from the formula: 10, 5, 5, 10, 3, 7, 7, 9.
- Row 14:** The formula **=XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade])** is visible in the background.

An intriguing aspect of this approach is its adaptability. Even if the names selected from the dropdown are not in sequential order, the formula dynamically adjusts, ensuring the correct range is captured between the two [2] individuals:

	A	B	C	D	E	F	G	H	I	J	K	L																														
1	SUM IF between																																									
2	SP SUM IF between 2 Names.xlsm																																									
3	Navigator																																									
4	Error Checks: <input checked="" type="checkbox"/>																																									
5																																										
6	1. SUM IF between																																									
7																																										
8	Data																																									
9																																										
10	<table border="1"> <thead> <tr> <th>Name</th> <th>Grade</th> </tr> </thead> <tbody> <tr><td>Charlie Taylor</td><td>6</td></tr> <tr><td>Ellie Nguyen</td><td>8</td></tr> <tr><td>Eloise Graham</td><td>7</td></tr> <tr><td>Emily White</td><td>10</td></tr> <tr><td>Ezra Gray</td><td>5</td></tr> <tr><td>Freya Ward</td><td>5</td></tr> <tr><td>Harrison Adams</td><td>10</td></tr> <tr><td>Isabelle Davies</td><td>3</td></tr> <tr><td>Marcus Graham</td><td>7</td></tr> <tr><td>Matilda Scott</td><td>7</td></tr> <tr><td>Olive Gray</td><td>9</td></tr> <tr><td>Piper James</td><td>8</td></tr> <tr><td>Samuel Gray</td><td>7</td></tr> <tr><td>Scarlett Wilson</td><td>7</td></tr> </tbody> </table>												Name	Grade	Charlie Taylor	6	Ellie Nguyen	8	Eloise Graham	7	Emily White	10	Ezra Gray	5	Freya Ward	5	Harrison Adams	10	Isabelle Davies	3	Marcus Graham	7	Matilda Scott	7	Olive Gray	9	Piper James	8	Samuel Gray	7	Scarlett Wilson	7
Name	Grade																																									
Charlie Taylor	6																																									
Ellie Nguyen	8																																									
Eloise Graham	7																																									
Emily White	10																																									
Ezra Gray	5																																									
Freya Ward	5																																									
Harrison Adams	10																																									
Isabelle Davies	3																																									
Marcus Graham	7																																									
Matilda Scott	7																																									
Olive Gray	9																																									
Piper James	8																																									
Samuel Gray	7																																									
Scarlett Wilson	7																																									
11																																										
12																																										
13																																										
14																																										
15																																										
16																																										
17																																										
18																																										
19																																										
20																																										
21																																										
22																																										
23																																										
24																																										
25																																										
26																																										
27																																										
28																																										
29																																										
30																																										

6 =XLOOKUP(Name_1,Data[Name],Data[Grade]);XLOOKUP(Name_2,Data[Name],Data[Grade])
8
7
10
5
7
10
3
7
7

Between Matilda Scott
and Charlie Taylor

Refining the Range with DROP

At this stage, while we could directly sum the array and then subtract the two [2] values of the lookup, we opt for a more refined approach by incorporating the **DROP** function. This function is instrumental in sculpting our dynamic range further before performing the summation.

We can use:

=DROP(Range,1)

and

=DROP(Range,-1)

to meticulously remove the first and last entries in our range, the grades directly associated with **Name_1** and **Name_2**.

Let's combine all of that in this formula:

=DROP(DROP(Range,1),-1)

This formula elegantly encapsulates the operation of removing first and last data. When we substitute **Range** with our earlier **XLOOKUP** array, we attain a perfectly tailored range for our summation:

=SUM(DROP(DROP(XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade]),1),-1))

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O																														
1	SUM IF between																																												
2	SP SUM IF between 2 Names.xlsm																																												
3	Navigator																																												
4	Error Checks: <input checked="" type="checkbox"/>																																												
5																																													
6	Data																																												
7																																													
8	<table border="1"> <thead> <tr> <th>Name</th> <th>Grade</th> </tr> </thead> <tbody> <tr><td>Charlie Taylor</td><td>6</td></tr> <tr><td>Ellie Nguyen</td><td>8</td></tr> <tr><td>Eloise Graham</td><td>7</td></tr> <tr><td>Emily White</td><td>10</td></tr> <tr><td>Ezra Gray</td><td>5</td></tr> <tr><td>Freya Ward</td><td>5</td></tr> <tr><td>Harrison Adams</td><td>10</td></tr> <tr><td>Isabelle Davies</td><td>3</td></tr> <tr><td>Marcus Graham</td><td>7</td></tr> <tr><td>Matilda Scott</td><td>7</td></tr> <tr><td>Olive Gray</td><td>9</td></tr> <tr><td>Piper James</td><td>8</td></tr> <tr><td>Samuel Gray</td><td>7</td></tr> <tr><td>Scarlett Wilson</td><td>7</td></tr> </tbody> </table>															Name	Grade	Charlie Taylor	6	Ellie Nguyen	8	Eloise Graham	7	Emily White	10	Ezra Gray	5	Freya Ward	5	Harrison Adams	10	Isabelle Davies	3	Marcus Graham	7	Matilda Scott	7	Olive Gray	9	Piper James	8	Samuel Gray	7	Scarlett Wilson	7
Name	Grade																																												
Charlie Taylor	6																																												
Ellie Nguyen	8																																												
Eloise Graham	7																																												
Emily White	10																																												
Ezra Gray	5																																												
Freya Ward	5																																												
Harrison Adams	10																																												
Isabelle Davies	3																																												
Marcus Graham	7																																												
Matilda Scott	7																																												
Olive Gray	9																																												
Piper James	8																																												
Samuel Gray	7																																												
Scarlett Wilson	7																																												
9																																													
10																																													
11																																													
12																																													
13																																													
14																																													
15																																													
16																																													
17																																													
18																																													
19																																													
20																																													
21																																													
22																																													
23																																													
24																																													
25																																													
26																																													
27																																													
28																																													
29																																													
30																																													

5 =DROP(DROP(XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade]),1),-1)
5
10
3
7

Between Emily White
and Olive Gray

SUM and Error Trapping

The penultimate step here is to **SUM**, hence we quickly add the **SUM** function here:

=SUM(DROP(DROP(XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade]),1),-1))

SUM IF between
 SP SUM IF between 2 Names.xlsxm
 Navigator
 Error Checks:

Data

Name	Grade
Charlie Taylor	6
Elle Nguyen	8
Eloise Graham	7
Emily White	10
Ezra Gray	5
Freya Ward	5
Harrison Adams	10
Isabelle Davies	3
Marcus Graham	7
Matilda Scott	7
Olive Gray	9
Piper James	8
Samuel Gray	7
Scarlett Wilson	7

Between and:

37 =SUM(DROP(DROP(XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade]),1),-1))

However, we must consider edge cases, such as the non-selection of names or the selection of adjacent names, which could lead to errors in our formula:

SUM IF between
 SP SUM IF between 2 Names.xlsxm
 Navigator
 Error Checks:

Data

Name	Grade
Charlie Taylor	6
Elle Nguyen	8
Eloise Graham	7
Emily White	10
Ezra Gray	5
Freya Ward	5
Harrison Adams	10
Isabelle Davies	3
Marcus Graham	7
Matilda Scott	7
Olive Gray	9
Piper James	8
Samuel Gray	7
Scarlett Wilson	7

Between and:

#CALC! =SUM(DROP(DROP(XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade]),1),-1))

To address this, we wrap our summation formula within an **IFERROR** statement.

=IFERROR(SUM(DROP(DROP(XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade]),1),-1)),0)

SUM IF between
 SP SUM IF between 2 Names.xlsxm
 Navigator
 Error Checks:

Data

Name	Grade
Charlie Taylor	6
Elle Nguyen	8
Eloise Graham	7
Emily White	10
Ezra Gray	5
Freya Ward	5
Harrison Adams	10
Isabelle Davies	3
Marcus Graham	7
Matilda Scott	7
Olive Gray	9
Piper James	8
Samuel Gray	7
Scarlett Wilson	7

Between and:

Solution

New version:
 =IFERROR(SUM(DROP(DROP(XLOOKUP(Name_1,Data[Name],Data[Grade]):XLOOKUP(Name_2,Data[Name],Data[Grade]),1),-1)),0)

This formula not only performs the summation but also ensures that in the face of any discrepancies, our formula robustly returns a zero [0] instead of an error.

Through this meticulous process, we ensure that our solution remains resilient, dynamic and accurate, adeptly managing the data between the specified names and gracefully handling any potential anomalies.

Suggested Solution for Legacy Excel Users (Older Versions of Excel)

The steps are detailed below.

Using INDEX MATCH and Colon to Create a Data Range

In a manner akin to the approach for modern Excel users, we will use the **INDEX MATCH** partnership combined with a colon to define the data range. This method uses

=INDEX(Data[Grade],MATCH(Name_1,Data[Name],0))

to locate the grade for **Name_1**. Similarly,

=INDEX(Data[Grade],MATCH(Name_2,Data[Name],0))

The above formula is employed to find the grade for **Name_2**.

Placing a colon between these two [2] formulae creates a range spanning from the grade of **Name_1** to **Name_2**, viz.

=INDEX(Data[Grade],MATCH(Name_1,Data[Name],0):INDEX(Data[Grade],MATCH(Name_2,Data[Name],0))

Data

Name	Grade
Charlie Taylor	6
Elie Nguyen	8
Eloise Graham	7
Emily White	10
Ezra Gray	5
Freya Ward	5
Harrison Adams	10
Isabelle Davies	3
Marcus Graham	7
Matilda Scott	7
Olive Gray	9
Piper James	8
Samuel Gray	7
Scarlett Wilson	7

7 =INDEX(Data[Grade],MATCH(Name_1,Data[Name],0)):INDEX(Data[Grade],MATCH(Name_2,Data[Name],0))
 10
 5
 5
 10
 3
 7
 7
 9

Between
and

Eloise Graham
Olive Gray

Making it Exclusive

To exclusively select the data between **Name_1** and **Name_2**, we introduce the **MAX** and **MIN** functions. These functions are employed to determine the endpoints of our data range:

=MAX(MATCH(Name_1,Data[Name],0),MATCH(Name_2,Data[Name],0))

This formula is used to find the last position in the data range.

=MIN(MATCH(Name_1,Data[Name],0),MATCH(Name_2,Data[Name],0))

Similarly, this above formula identifies the first position.

Subsequently, we modify our original **INDEX:INDEX** formula to replicate the effect of the **DROP** function used in the modern Excel solution:

=INDEX(Data[Grade],MIN(MATCH(Name_1,Data[Name],0),MATCH(Name_2,Data[Name],0))+1):INDEX(Data[Grade],MAX(MATCH(Name_1,Data[Name],0),MATCH(Name_2,Data[Name],0))-1)

1 SUM IF between
 2 SP SUM IF between 2 Names.xlsm
 3 Navigator
 4 Error Checks: [X]

8 Data

Name	Grade
Charlie Taylor	6
Elie Nguyen	8
Eloise Graham	7
Emily White	10
Ezra Gray	5
Freya Ward	5
Harrison Adams	10
Isabelle Davies	3
Marcus Graham	7
Matilda Scott	7
Olive Gray	9
Piper James	8
Samuel Gray	7
Scarlett Wilson	7

5 =INDEX(Data[Grade],MIN(MATCH(Name_1,Data[Name],0),MATCH(Name_2,Data[Name],0))+1):INDEX(Data[Grade],MAX(MATCH(Name_1,Data[Name],0),MATCH(Name_2,Data[Name],0))-1)
 5
 10
 3
 7
 7

27 Between
 28 and
 29 Emily White
 30 Olive Gray

Addressing Specific Selection Scenarios

While effective, our method may not be perfect in all cases. For instance, if **Name_1** and **Name_2** are adjacent, the same name is selected twice or the input cells are left empty, the formula might produce an incorrect range or result in an error.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
1	SUM IF between																					
2	SP SUM IF between 2 Names.xlsx																					
3	Navigator																					
4	Error Checks: <input type="checkbox"/>																					
5																						
6																						
7																						
8																						
9																						
10																						
11																						
12																						
13																						
14																						
15																						
16																						
17																						
18																						
19																						
20																						
21																						
22																						
23																						
24																						
25																						
26																						
27																						
28																						
29																						
30																						

To address this, we implement the following formula:

$$=IFERROR(ABS(MATCH(Name_1,Data[Name],0)-MATCH(Name_2,Data[Name],0)),0)<=1$$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	SUM IF between													
2	SP SUM IF between 2 Names - Complete.xlsx													
3	Navigator													
4	Error Checks: <input type="checkbox"/>													
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														
25														
26														
27														
28														
29														
30														

This formula utilises **ABS** and **MATCH** functions to detect if **Name_1** and **Name_2** are adjacent or identical. Additionally, the **IFERROR** function will intercept any #N/A errors from the **MATCH** function not able to match any name and then converting them to zero [0]. The subsequent comparison (**<=1**) checks if the range is suitable for summation. We encapsulate this logic within a comprehensive **IF** statement:

$$=IF(IFERROR(ABS(MATCH(Name_1,Data[Name],0)-MATCH(Name_2,Data[Name],0)),0)<=1, 0, SUM(INDEX(Data[Grade], MIN(MATCH(Name_1,Data[Name],0),MATCH(Name_2,Data[Name],0))+1) : INDEX(Data[Grade], MAX(MATCH(Name_1,Data[Name],0),MATCH(Name_2,Data[Name],0))-1)))$$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	SUM IF between														
2	SP SUM IF between 2 Names - Complete.xlsx														
3	Navigator														
4	Error Checks: <input type="checkbox"/>														
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															
27															
28															
29															
30															
31															
32															
33															
34															
35															
36															
37															
38															

By using this refined approach, we cater to the nuances of legacy Excel, ensuring an accurate and exclusive summation between the selected names.

Word to the Wise

We appreciate there are many, many ways this could have been achieved. If you have come up with an alternative, radically different approach, congratulations – that’s half the fun of Excel!

Upcoming SumProduct Training Courses

Location	Course	Course Date	Local Time	UTC	Duration
London UK	Financial Modelling	13 January 2025 - 14 January 2025	09:00 - 17:00 GMT	13 January 2025 09:00 UTC - 14 January 2025 16:00 UTC	2 Days
Melbourne Australia	Power Pivot, Power Query and Power BI	14 January 2025 - 15 January 2025	09:00 - 17:00 AEDT	13 January 2025 22:00 UTC - 15 January 2025 06:00 UTC	2 Days
Melbourne Australia	ChatGPT	15 January 2025 - 16 January 2025	09:00 - 17:00 AEDT	14 January 2025 22:00 UTC - 16 January 2025 06:00 UTC	2 Days
Melbourne Australia	Excel Tips and Tricks	20 January 2025	09:00 - 17:00 AEDT	19 January 2025 22:00 UTC - 20 January 2025 06:00 UTC	1 Day
Sydney Australia	Excel Tips and Tricks	22 January 2025	09:00 - 17:00 AEDT	21 January 2025 22:00 UTC - 22 January 2025 06:00 UTC	1 Day
London UK	ChatGPT	22 January 2025 - 23 January 2025	09:00 - 17:00 GMT	22 January 2025 09:00 UTC - 23 January 2025 17:00 UTC	2 Days
London UK	Financial Modelling	17 February 2025 - 18 February 2025	09:00 - 17:00 BST	17 February 2025 09:00 UTC - 18 February 2025 16:00 UTC	2 Days
Melbourne Australia	Power Pivot, Power Query and Power BI	18 February 2025 - 19 February 2025	09:00 - 17:00 AEDT	17 February 2025 22:00 UTC - 19 February 2025 06:00 UTC	2 Days
Melbourne Australia	ChatGPT	19 February 2025 - 20 February 2025	09:00 - 17:00 AEDT	18 February 2025 22:00 UTC - 20 February 2025 06:00 UTC	2 Days

Key Strokes

Each newsletter, we'd like to introduce you to useful keystrokes you may or may not be aware of. This time, we thought we would regain **CTRL** of the alphabet:

Keystroke	What it does
CTRL + A	Select current region, select all
CTRL + B	Bold (toggle)
CTRL + C	Copy
CTRL + D	Fill down
CTRL + E	Flash fill
CTRL + F	Find dialog
CTRL + G	Go To
CTRL + H	Replace
CTRL + I	Italic (toggle)
CTRL + J	Enter a line break (for Find / Replace or Text to Columns)
CTRL + K	Insert Hyperlink
CTRL + L	Excel 2007: Create Table; Excel 2003: Create List

Keystroke	What it does
CTRL + N	New Workbook
CTRL + O	Open Workbook
CTRL + P	Print
CTRL + Q	Quick Analysis
CTRL + R	Fill right
CTRL + S	Save
CTRL + T	Excel 2007: Insert Table
CTRL + U	Underline (toggle)
CTRL + V	Paste
CTRL + W	Close Window
CTRL + X	Cut
CTRL + Y	Redo
CTRL + Z	Undo

There are c.550 keyboard shortcuts in Excel. For a comprehensive list, please download our Excel file at <http://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.