

# Sum Product

NEWSLETTER #113 - April 2022

[www.sumproduct.com](http://www.sumproduct.com) | [www.sumproduct.com/thought](http://www.sumproduct.com/thought)



*Big month for Excel* with no fewer than 14 new Excel functions to assist text manipulation and combining, shaping and resizing arrays. Given there are updates to Excel for the Web, we've related these too. We should just write about those and give up at this stage for this month's newsletter.

But we wouldn't do that to you.

In the current economic climate, we thought it might be good to consider how to model working capital and convert Profit & Loss projections into their cash flow counterparts. But no, that's not enough either.

We have another Beat the Boredom Challenge, plus our usual articles on Charts & Dashboards, Visual Basics, Power Pivot Principles, Power Query Pointers and Power BI updates. And because discussing 14 new functions is clearly insufficient, we also have our usual A to Z of Excel Functions under the imaginary cosh (**IMCOSH**) too. Finally, we see you **RIGHT** with our Keyboard Shortcuts.

Big, big month!

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

*Liam Bastick*, Managing Director, SumProduct



## 14 New Functions for Excel



Microsoft doesn't want to do things by halves, does it? Last month saw the announcement of no less than 14 (!) new Excel functions designed to help manipulate text and arrays in your worksheets. If only we could find some that would massage the financial as well...

Now, before I go any further, let me be completely clear about availability. At the time of writing, these functions are currently only available to users running Beta Channel, Version 2203 (Build 15104.20004) or later on Windows and Version 16.60 (Build 22030400) or later on Mac. Even then, the availability is "flighted". Here at SumProduct HQ, we have had to get clever with virtual machines to get our grubby little hands

on them, so please don't despair if you can't access them yet. They are definitely circulating!!

In summary, the functions are essentially grouped as follows:

- text manipulation
- combining arrays
- shaping arrays
- resizing arrays.

Let's take a look at each of these four subsets in turn.

## Text Manipulation Functions

We have talked about the common text manipulation functions such as **FIND**, **LEFT**, **LEN**, **MID**, **RIGHT**, **SEARCH** and **SUBSTITUTE** before, but these new functions allow you to dismember text strings without requiring a PhD in Astrophysics.

If it **TEXT** you too long to understand these older functions, then you needn't worry anymore. There are three new functions that may help here:

1. **TEXTBEFORE**: returns text that's before delimiting characters
2. **TEXTAFTER**: returns text that's after delimiting characters
3. **TEXTSPLIT**: splits text into rows or columns using delimiters.

### The **TEXTBEFORE** function



The **TEXTBEFORE** function returns the string of text that occurs before a given substring (*i.e.* a character or set of characters) in that string. It is the opposite of the **TEXTAFTER** function. **TEXTBEFORE** has the following syntax:

**TEXTBEFORE**(text, delimiter, [instance number], [ignore case])

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

- **text**: this is required and represents the text string you are searching within. Wildcard characters are not allowed
- **delimiter**: this is also required and represents the text in the **text** string that marks the point before which you wish to extract
- **instance number**: this is the first optional argument and denotes the **n**th instance of the **delimiter** before which you wish to extract. By default, this is equal to one [1]. If a negative number is used here, the function starts searching for the **delimiter** from the end rather than the beginning
- **ignore case**: this too is an optional argument and determines whether the search is case sensitive or not. The default is TRUE, which means the search for the **delimiter** is case insensitive; explicitly use FALSE to make the search case sensitive.

It should be further noted that:

- Excel should return an **#N/A** error if the **delimiter** is an empty string, but the current Beta version appears to return a blank
- Excel returns an **#VALUE!** error if the **instance number** is zero (the default is one)
- Excel returns an **#N/A** error if the **delimiter** does not occur within the **text**
- Excel returns an **#N/A** error if the **instance number** is greater than the number of occurrences of the **delimiter** within the **text**.

Please see the examples below:

	A	B	C
1	Data		
2	Many many occurrences of "many" in this sentence.		
3		empty	
4			
5			
6	Formula	Description	Result
7	=TEXTBEFORE(A2, "Many")	Identifies there is no text before "Many" is used as the <b>delimiter</b> (case sensitive is default).	
8	=TEXTBEFORE(A2, "many")	Identifies there is text before "many" is used as the <b>delimiter</b> (case sensitive is default).	Many
9	=TEXTBEFORE(A2, "many",, FALSE)	Identifies the text before "many" is used as the <b>delimiter</b> (case sensitive using FALSE).	Many
10	=TEXTBEFORE(A2, "many",, TRUE)	Identifies there is no text before "many" is used as the <b>delimiter</b> (case insensitive using TRUE).	
11	=TEXTBEFORE(A2, "many", 2, FALSE)	Identifies text before second occurrence of "many" (case sensitive).	Many many occurrences of "
12	=TEXTBEFORE(A2, "many", 2, TRUE)	Identifies text before second occurrence of "many" (case insensitive).	Many
13	=TEXTBEFORE(A2, "Many", 2, FALSE)	No second occurrence of "Many" (case sensitive).	#N/A
14	=TEXTBEFORE(A2, "Many", 2)	No second occurrence of "Many" (case sensitive).	#N/A
15	=TEXTBEFORE(A2, "many", -1, FALSE)	Identifies there is text before the final occurrence of "many" is used as the <b>delimiter</b> (case sensitive).	Many many occurrences of "
16	=TEXTBEFORE(A2, "many", -2, FALSE)	Identifies there is text before the penultimate occurrence of "many" is used as the <b>delimiter</b> (case sensitive).	Many
17	=TEXTBEFORE(A2, "Many", -2, FALSE)	Insufficient occurrences of "Many" (case sensitive).	#N/A
18	=TEXTBEFORE(A2,)	The <b>delimiter</b> is not specified.	
19	=TEXTBEFORE(A2, "")	The <b>delimiter</b> is not specified.	
20	=TEXTBEFORE(A3, "Many")	Insufficient occurrences of "Many" (case sensitive) (cell blank).	#VALUE!

The **TEXTAFTER** function



The **TEXTAFTER** function returns the string of text that occurs after a given substring (*i.e.* a character or set of characters) in that string. It is the opposite of the **TEXTBEFORE** function. **TEXTAFTER** has the following syntax:

**TEXTAFTER(text, delimiter, [instance number], [ignore case])**

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

- **text**: this is required and represents the text string you are searching within. Wildcard characters are not allowed
- **delimiter**: this is also required and represents the text in the **text** string that marks the point after which you wish to extract
- **instance number**: this is the first optional argument and denotes the nth instance of the delimiter after which you wish to extract. By default, this is equal to one [1]. If a negative number is used here, the function starts searching for the **delimiter** from the end rather than the beginning
- **ignore case**: this too is an optional argument and determines whether the search is case sensitive or not. The default is TRUE, which means the search for the delimiter is case insensitive; explicitly use FALSE to make the search case sensitive.

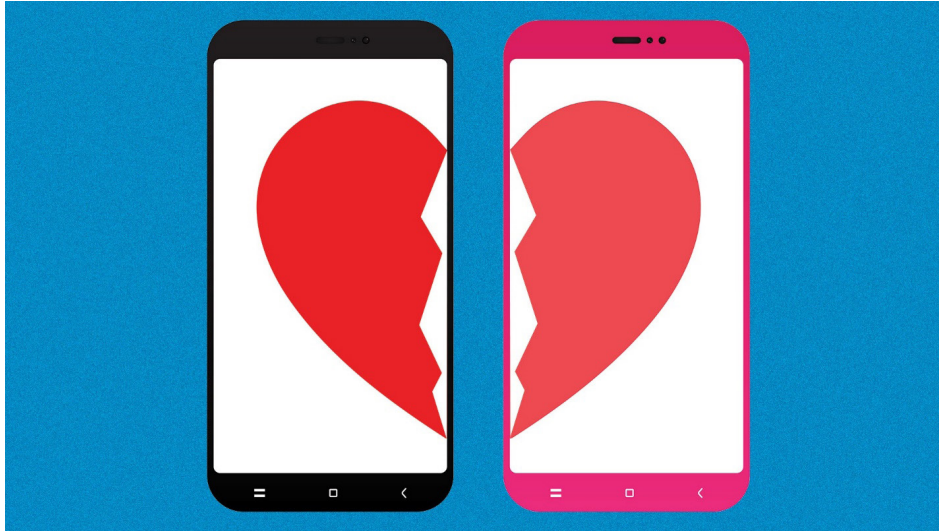
It should be further noted that:

- Excel should return an #N/A error if the **delimiter** is an empty string, but the current Beta version appears to return a blank
- Excel returns an #VALUE! error if the **instance number** is zero (the default is one)
- Excel returns an #N/A error if the **delimiter** does not occur within the **text**
- Excel returns an #N/A error if the **instance number** is greater than the number of occurrences of the delimiter within the **text**.

Please see relevant examples below:

	A	B	C
1	Data		
2	Many many occurrences of "many" in this sentence.		
3		empty	
4			
5			
6	Formula	Description	Result
7	=TEXTAFTER(A2, "Many")	Identifies there is text after "Many" is used as the <b>delimiter</b> (case sensitive is default).	many occurrences of "many" in this sentence.
8	=TEXTAFTER(A2, "many")	Identifies there is text after "many" is used as the <b>delimiter</b> (case sensitive is default).	occurrences of "many" in this sentence.
9	=TEXTAFTER(A2, "many",, FALSE)	Identifies the text after "many" is used as the <b>delimiter</b> (case sensitive using FALSE).	occurrences of "many" in this sentence.
10	=TEXTAFTER(A2, "many",, TRUE)	Identifies there is text after "many" is used as the <b>delimiter</b> (case insensitive using TRUE).	many occurrences of "many" in this sentence.
11	=TEXTAFTER(A2, "many", 2, FALSE)	Identifies text after second occurrence of "many" (case sensitive).	" in this sentence.
12	=TEXTAFTER(A2, "many", 2, TRUE)	Identifies text after second occurrence of "many" (case insensitive).	occurrences of "many" in this sentence.
13	=TEXTAFTER(A2, "Many", 2, FALSE)	No second occurrence of "Many" (case sensitive).	#N/A
14	=TEXTAFTER(A2, "Many", 2)	No second occurrence of "Many" (case sensitive).	#N/A
15	=TEXTAFTER(A2, "many", -1, FALSE)	Identifies there is text after the final occurrence of "many" is used as the <b>delimiter</b> (case sensitive).	" in this sentence.
16	=TEXTAFTER(A2, "many", -2, FALSE)	Identifies there is text after the penultimate occurrence of "many" is used as the <b>delimiter</b> (case sensitive).	occurrences of "many" in this sentence.
17	=TEXTAFTER(A2, "Many", -2, FALSE)	Insufficient occurrences of "Many" (case sensitive).	#N/A
18	=TEXTAFTER(A2,)	The <b>delimiter</b> is not specified.	Many many occurrences of "many" in this sentence.
19	=TEXTAFTER(A2, "")	The <b>delimiter</b> is not specified.	Many many occurrences of "many" in this sentence.
20	=TEXTAFTER(A3, "Many")	Insufficient occurrences of "Many" (case sensitive) (cell blank).	#VALUE!

## The TEXTAFTER function



The **TEXTSPLIT** function is intended to work like the Text to Columns button on the Data tab of the Ribbon, almost like the “inverse” of the **TEXTJOIN** function. It allows you to split a given text across rows or down columns. **TEXTSPLIT** has the following syntax:

**TEXTSPLIT(text, [column delimiter], [row delimiter], [ignore empty], [pad with])**

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

- **text**: this is required and represents the text string you wish to split
- **column delimiter**: this is optional and denotes one or more characters that specify where to split the text across columns
- **row delimiter**: this is optional and denotes one or more characters that specify where to spill the text down rows
- **ignore empty**: another optional argument, you should specify TRUE to create an empty cell when two delimiters are used. This argument defaults to FALSE, which means don't create an empty cell
- **pad with**: not to be confused with Pad Thai, this final optional argument “pads” the resulting text range where cells would otherwise be blank. The default is N/A.

If there is more than one delimiter (row or column), then an array constant must be used. For example, to split by both a comma (,) and a period (full stop, .), use **=TEXTSPLIT(text, {“,”, “.”})**.

Just for a change, some more examples:

A	B	C	D	E	F	G	H	I	J
Data									
Many sentences. There are three. Just three.									
padding									
	Many	sentences.		There	are	three.		Just	three.

**=TEXTSPLIT(A2, “ ”)**

A	B	C	D
Data			
Many sentences. There are three. Just three.			
padding			
	Many sentences	There are three	Just three

**=TEXTSPLIT(A2, “. ”)**

A	B	C
Data		
Many sentences. There are three. Just three.		
padding		
	Many sentences	<b>=TEXTSPLIT(A2,, “. ”)</b>
	There are three	
	Just three	

	A	B	C	D	E	F
1	Data					
2	Many sentences. There are three. Just three.					
3	padding					
4						
5						
6		Many	sentences	#N/A	#N/A	#N/A
7				There	are	three
8				Just	three	#N/A
9			#N/A	#N/A	#N/A	#N/A
10						
11						
12						

**=TEXTSPLIT(A2, " ", ".")**

	A	B	C	D
1	Data			
2	Many sentences. There are three. Just three.			
3	padding			
4				
5				
6		Many	sentences	#N/A
7		There	are	three
8		Just	three	#N/A
9				
10				
11				
12				

**=TEXTSPLIT(A2, " ", ". ", TRUE)**

	A	B	C	D	E	F
1	Data					
2	Many sentences. There are three. Just three.					
3	padding					
4						
5						
6		Many	sentences	#N/A	#N/A	#N/A
7				There	are	three
8				Just	three	#N/A
9			#N/A	#N/A	#N/A	#N/A
10						
11						
12						

**=TEXTSPLIT(A2, " ", ". ", FALSE)**

	A	B	C	D	E
1	Data				
2	Many sentences. There are three. Just three.				
3	padding				
4					
5					
6		Many	sentences	padding	
7		There	are	three	
8		Just	three	padding	
9					
10					
11					
12					

**=TEXTSPLIT(A2, " ", ". ", TRUE, A3)**

	A	B	C	D	E	F	G	H	I	J	K	L
1	Data											
2	Many sentences. There are three. Just three.											
3	padding											
4												
5												
6		Many	sentences	There	are	three	Just	three				
7												
8												
9												

**=TEXTSPLIT(A2, {" ", ". "})**

	A	B	C	D
1	Data			
2	Many sentences. There are three. Just three.			
3	padding			
4				
5				
6		Many	=TEXTSPLIT(A2, {" ", "."})	
7		sentences		
8				
9				
10		There		
11		are		
12		three		
13				
14				
15		Just		
16		three		
17				

### Combining Arrays

Since end users have been playing with arrays more and more, it has become noticeable that it can be quite challenging to combine data, especially when their sources are flexible in size. There are two new functions that may assist:

1. **HSTACK**: combine dynamic arrays, stacking horizontally
2. **VSTACK**: combine dynamic arrays, stacking vertically.

### The HSTACK function



The **HSTACK** function returns the array formed by appending each of the array arguments in a column-wise fashion (Microsoft's jargon, not ours). It has the following syntax:

**HSTACK(array1, [array2, ...])**

The **HSTACK** function has the following argument(s):

- **array**: the first argument is required (others are optional) and represents the **array(s)** to append.

It should be noted that:

- **HSTACK** returns the array formed by appending each of the array arguments in a column-wise fashion. The resulting **array** will be the following dimensions:
  - o columns: the maximum of the column count from each of the array arguments
  - o rows: the combined count of all the rows from each of the array arguments
- Excel returns an **#N/A** error if an array has fewer rows or columns than the maximum in any selected array. To remove the errors, you should use the **IFERROR** function.

Please see the following examples:

	A	B	C	D	E	F	G	H
1								
2		Mary	had	a		Her	father	shot
3		little	lamb			the	shepherd	
4								
5								
6		Mary	had	a	Her	father	shot	
7		little	lamb		0 the	shepherd		0
8								
9		<b>=HSTACK(B2:D3, F2:H3)</b>						

	A	B	C	D	E	F	G
1							
2		Mary	had	a		It	
3		little	lamb			was	delicious
4							
5							
6		Mary	had	a	It		0
7		little	lamb		0 was	delicious	
8							
9		<b>=HSTACK(B2:D3, F2:G3)</b>					

	A	B	C	D	E	F	G	H	I
1									
2		Mary	had	a		Little			
3		little	lamb			woolly	jumper		
4									
5									
6		Mary	had	a	little	lamb	Little	woolly	jumper
7									
8		<b>=HSTACK(B2:D2, B3:C3, F2, F3:G3)</b>							

**The VSTACK function**



The **VSTACK** function returns the array formed by appending each of the array arguments in a row-wise fashion (Microsoft's jargon, not ours). It has the following syntax:

**VSTACK(array1, [array2, ...])**

The **VSTACK** function has the following argument(s):

- **array**: the first argument is required (others are optional) and represents the **array(s)** to append.

It should be noted that:

- **VSTACK** returns the array formed by appending each of the array arguments in a row-wise fashion. The resulting array will be the following dimensions:
  - o rows: the maximum of the row count from each of the array arguments
  - o columns: the combined count of all the columns from each of the array arguments
- Excel returns an #N/A error if an array has fewer rows or columns than the maximum in any selected array. To remove the errors, you should use the **IFERROR** function.

Some illustrations:

	A	B	C	D	E	F	G	H
1								
2		Mary	had	a		Her	father	shot
3		little	lamb			the	shepherd	
4								
5								
6		Mary	had	a		<b>=VSTACK(B2:D3, F2:H3)</b>		
7		little	lamb		0			
8		Her	father	shot				
9		the	shepherd		0			
10								

	A	B	C	D	E	F	G	H
1								
2		Mary	had	a		It		
3		little	lamb			was	delicious	
4								
5								
6		Mary	had	a		<b>=VSTACK(B2:D3, F2:G3)</b>		
7		little	lamb		0			
8		It		0	#N/A			
9		was	delicious		#N/A			
10								

	A	B	C	D	E	F	G	H
1								
2		Roses	are	red	Violets	are	#REF!	
3		Some	poems	rhyme	but	this		
4		one	doesn't					
5								
6								
7		Roses	are	#N/A		<b>=VSTACK(B2:C4, D2:F3, G2)</b>		
8		Some	poems	#N/A				
9		one	doesn't	#N/A				
10		red	Violets	are				
11		rhyme	but	this				
12		#REF!	#N/A	#N/A				
13								



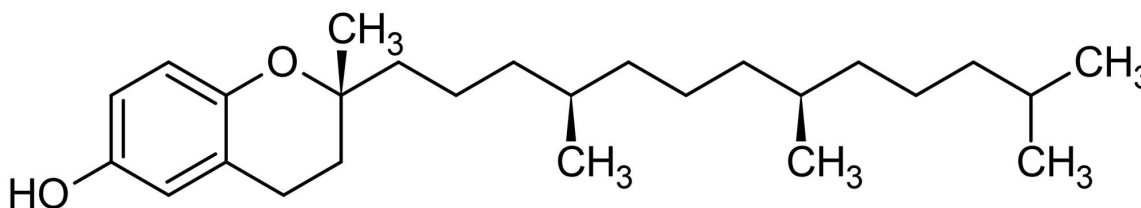
	A	B	C	D	E	F	G	H
1								
2		Mary	had	a		Little		
3		little	lamb			woolly	jumper	
4								
5								
6		Mary	had	a		<b>=VSTACK(B2:D2, B3:C3, F2, F3:G3)</b>		
7		little	lamb	#N/A				
8		Little	#N/A	#N/A				
9		woolly	jumper	#N/A				

## Shaping Arrays

Changing the “shape” of data in Excel, especially from arrays to lists and vice versa, is a popular request with our clients and is difficult to achieve formulaically (Power Query makes it nice and easy though!). This is where the next four functions come into play:

1. **TOCOL**: convert a two-dimensional array into a single column (list) of data
2. **TOROW**: convert a two-dimensional array into a single row (list) of data
3. **WRAPCOLS**: creates a two-dimensional array of a specified height by wrapping data from a column (list) of data once the prescribed height is achieved (this is essentially the opposite of the **TOCOL** or **TOROW** functions)
4. **WRAPROWS**: creates a two-dimensional array of a specified width by wrapping data from a row (list) of data once the prescribed width is achieved (this is essentially the opposite of the **TOCOL** or **TOROW** functions).

### The TOCOL function



The **TOCOL** function returns a column vector containing all of the items in the source array. It has the following syntax:

**TOCOL(array, [ignore], [scan by column])**

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

- **array**: this is required and denotes the array or reference to return as a column
- **ignore**: this is optional and identifies whether to ignore certain types of values; by default, no values are ignored. The omissions are governed as follows:

Icon	Meaning
0	Keep all values (default)
1	Ignore blanks
2	Ignore errors
3	Ignore blanks and errors

- **scan by column**: this is optional and sets the scan of the array by column. However, by default, the **array** is scanned by row.

It should be noted that:

- if **scan by column** is omitted or FALSE, the **array** is scanned by row; if TRUE, the **array** is scanned by column
- Excel returns an **#VALUE!** error when an **array** constant contains one or more numbers that are not a whole number
- Excel returns an **#NUM!** error when **array** becomes too large.

Just for a change, some examples:

	A	B	C	D	E
1					
2		One	Two	Three	Four
3		Five	Six	Seven	Eight
4		Nine	Ten	Eleven	Twelve
5					
6					
7		One		<b>=TOCOL(B2:E4)</b>	
8		Two			
9		Three			
10		Four			
11		Five			
12		Six			
13		Seven			
14		Eight			
15		Nine			
16		Ten			
17		Eleven			
18		Twelve			

	A	B	C	D	E
1					
2		One	Two	Three	Four
3		Five	Six	Seven	Eight
4		Nine	Ten		
5					
6					
7		One		<b>=TOCOL(B2:E4)</b>	
8		Two			
9		Three			
10		Four			
11		Five			
12		Six			
13		Seven			
14		Eight			
15		Nine			
16		Ten			
17			0		
18			0		

	A	B	C	D	E
1					
2		#DIV/0!	Two	Three	Four
3		Five	Six	#NULL!	Eight
4		Nine	Ten		
5					
6					
7		Two		<b>=TOCOL(B2:E4, 3)</b>	
8		Three			
9		Four			
10		Five			
11		Six			
12		Eight			
13		Nine			
14		Ten			

	A	B	C	D	E
1					
2		#DIV/0!	Two	Three	Four
3		Five	Six	#NULL!	Eight
4		Nine	Ten		
5					
6					
7		#DIV/0!		=TOCOL(B2:E4, 1, TRUE)	
8		Five			
9		Nine			
10		Two			
11		Six			
12		Ten			
13		Three			
14		#NULL!			
15		Four			
16		Eight			
17					

## The TOROW function



The **TOROW** function returns a row vector containing all of the items in the source array. It has the following syntax:

**TOROW(array, [ignore], [scan by column])**

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

- **array**: this is required and denotes the array or reference to return as a row
- **ignore**: this is optional and identifies whether to ignore certain types of values; by default, no values are ignored. The omissions are governed as follows:

Icon	Meaning
0	Keep all values (default)
1	Ignore blanks
2	Ignore errors
3	Ignore blanks and errors

- **scan by column**: this is optional and sets the scan of the array by column. However, by default, the **array** is scanned by row.

It should be noted that:

- if **scan by column** is omitted or FALSE, the **array** is scanned by row; if TRUE, the **array** is scanned by column
- Excel returns an #VALUE! error when an **array** constant contains one or more numbers that are not a whole number
- Excel returns an #NUM! error when **array** becomes too large.

Some illustrations:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2		One	Two	Three	Four								
3		Five	Six	Seven	Eight								
4		Nine	Ten	Eleven	Twelve								
5													
6													
7		One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Ten	Eleven	Twelve
8													
9													

**=TOROW(B2:E4)**

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2		One	Two	Three	Four								
3		Five	Six	Seven	Eight								
4		Nine	Ten										
5													
6													
7		One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Ten	0	0
8													
9													

**=TOROW(B2:E4)**

	A	B	C	D	E	F	G	H	I
1									
2		#DIV/0!	Two	Three	Four				
3		Five	Six	#NULL!	Eight				
4		Nine	Ten						
5									
6									
7		Two	Three	Four	Five	Six	Eight	Nine	Ten
8									
9									

**=TOROW(B2:E4, 3)**

	A	B	C	D	E	F	G	H	I	J	K
1											
2		#DIV/0!	Two	Three	Four						
3		Five	Six	#NULL!	Eight						
4		Nine	Ten								
5											
6											
7		#DIV/0!	Five	Nine	Two	Six	Ten	Three	#NULL!	Four	Eight
8											
9											

**=TOROW(B2:E4, 1, TRUE)**

The WRAPCOLS function



The **WRAPCOLS** function wraps the provided vector by columns after a specified number of elements. It has the following syntax:

**WRAPCOLS(vector, wrap count, [pad with])**

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

- **vector**: this is required and denotes the row or column vector / reference to wrap
- **wrap count**: this is also required and represents the maximum number of values (depth / height) for each column
- **pad with**: this is optional and defines the value with which to pad. The default is *N/A*.

It should be noted that:

- the elements of the vector are placed into a two-dimensional array by column
- each column has **wrap count** elements
- the column is padded with **pad width** if there are insufficient elements to fill it
- if **wrap count** is greater or equal to the number of elements in **vector**, then the **vector** is simply returned as the column vector result of the function
- Excel returns an *#VALUE!* error when **vector** is not a one-dimensional array
- Excel returns an *#VALUE!* error when **wrap count** is less than one [1] or is not an integer.

Please see the following examples:

	A	B	C	D	E	F	G	H
1								
2		One		One	Six			=WRAPCOLS(B2:B9, 5)
3		Two		Two	Seven			
4		Three		Three	Eight			
5		Four		Four	#N/A			
6		Five		Five	#N/A			
7		Six						
8		Seven						
9		Eight						

	A	B	C	D	E	F	G	H	I
1									
2		One	Two	Three	Four	Five	Six	Seven	Eight
3									
4									
5		One	Six						=WRAPCOLS(B2:I2, 5)
6		Two	Seven						
7		Three	Eight						
8		Four	#N/A						
9		Five	#N/A						

	A	B	C	D	E	F	G	H	I
1									
2		One	Two	Three	Four	Five	Six	Seven	Eight
3									
4		Padding							
5									
6		One	Six						=WRAPCOLS(B2:I2, 5, B4)
7		Two	Seven						
8		Three	Eight						
9		Four	Padding						
10		Five	Padding						

## The WRAPROWS function



The **WRAPROWS** function wraps the provided vector by rows after a specified number of elements. It has the following syntax:

**WRAPROWS(vector, wrap count, [pad with])**

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

- **vector**: this is required and denotes the row or column vector / reference to wrap
- **wrap count**: this is also required and represents the maximum number of values (width) for each row
- **pad with**: this is optional and defines the value with which to pad. The default is *N/A*.

It should be noted that:

- the elements of the vector are placed into a two-dimensional array by row
- each row has **wrap count** elements
- the row is padded with **pad width** if there are insufficient elements to fill it
- if **wrap count** is greater or equal to the number of elements in vector, then the **vector** is simply returned as the row **vector** result of the function
- Excel returns an *#VALUE!* error when **vector** is not a one-dimensional array
- Excel returns an *#VALUE!* error when **wrap count** is less than one [1] or is not an integer.

More examples:

	A	B	C	D	E	F	G	H
1								
2		One		One	Two	Three	Four	Five
3		Two		Six	Seven	Eight	#N/A	#N/A
4		Three						
5		Four						
6		Five						
7		Six						
8		Seven						
9		Eight						
10								

**=WRAPROWS(B2:B9, 5)**

	A	B	C	D	E	F	G	H	I
1									
2		One	Two	Three	Four	Five	Six	Seven	Eight
3									
4									
5		One	Two	Three	Four	Five			
6		Six	Seven	Eight	#N/A	#N/A			
7									
8									

**=WRAPROWS(B2:I2, 5)**

	A	B	C	D	E	F	G	H	I
1									
2		One	Two	Three	Four	Five	Six	Seven	Eight
3									
4		Padding							
5									
6		One	Two	Three	Four	Five			
7		Six	Seven	Eight	Padding	Padding			
8									
9									

**=WRAPROWS(B2:I2, 5, B4)**

### Resizing Arrays

The final array manipulation covered by this myriad of new functions concerns resizing. This is where the last five functions prove useful:

1. **CHOOSECOLS**: returns the specified rows from an array
2. **CHOSEROWS**: returns the specified columns from an array
3. **DROP**: drops rows or columns from an array start or end
4. **EXPAND**: expands an array to the specified dimensions
5. **TAKE**: returns rows or columns from the start or end of an array.

### The CHOOSECOLS function



The **CHOOSECOLS** function returns the specified columns from an array. It has the following syntax:

**CHOOSECOLS(array, column number 1, [column number 2, ...])**

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

- **array**: this is required and represents the selected array
- **column number 1**: this is also required and denotes the column number of the first column to be returned
- **column number 2**: this and subsequent arguments are optional. This / these represent(s) the second and subsequent column numbers to be returned.

It should be noted that Excel will return an **#VALUE!** error if the absolute value of any of the **column number** arguments is zero or exceeds the number of columns in the **array**.

Some examples:

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		Delta						<b>=CHOOSECOLS(B2:F6, 4)</b>
10		India						
11		November						
12		Sierra						
13		Vodka						

	A	B	C	D	E	F	G	H	I
1									
2		Alpha	Bravo	Charlie	Delta	Echo			
3		Foxtrot	Golf	Hotel	India	Juliet			
4		Kilo	Liam	Mike	November	Oscar			
5		Papa	Quebec	Romeo	Sierra	Tango			
6		Uniform	Victor	Whiskey	Vodka	Collapse			
7									
8									
9		Alpha	Charlie	Echo	Alpha				<b>=CHOOSECOLS(B2:F6, 1, 3, 5, 1)</b>
10		Foxtrot	Hotel	Juliet	Foxtrot				
11		Kilo	Mike	Oscar	Kilo				
12		Papa	Romeo	Tango	Papa				
13		Uniform	Whiskey	Collapse	Uniform				

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		#VALUE!						<b>=CHOOSECOLS(B2:F6, )</b>

	A	B	C	D	E	F	G	H	I
1									
2		Alpha	Bravo	Charlie	Delta	Echo			
3		Foxtrot	Golf	Hotel	India	Juliet			
4		Kilo	Liam	Mike	November	Oscar			
5		Papa	Quebec	Romeo	Sierra	Tango			
6		Uniform	Victor	Whiskey	Vodka	Collapse			
7									
8									
9		#VALUE!							<b>=CHOOSECOLS(B2:F6, 2, 4, 6)</b>



## The CHOOSEROWS function



The **CHOOSEROWS** function returns the specified rows from an array. It has the following syntax:

**CHOOSEROWS(array, row number 1, [row number 2, ...])**

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

- **array**: this is required and represents the selected array
- **row number 1**: this is also required and denotes the row number of the first row to be returned
- **row number 2**: this and subsequent arguments are optional. This / these represent(s) the second and subsequent row numbers to be returned.

It should be noted that Excel will return an **#VALUE!** error if the absolute value of any of the **row number** arguments is zero or exceeds the **number** of rows in the **array**.

Illustrations:

	A	B	C	D	E	F	G	H	I
1									
2		Alpha	Bravo	Charlie	Delta	Echo			
3		Foxtrot	Golf	Hotel	India	Juliet			
4		Kilo	Liam	Mike	November	Oscar			
5		Papa	Quebec	Romeo	Sierra	Tango			
6		Uniform	Victor	Whiskey	Vodka	Collapse			
7									
8									
9		Papa	Quebec	Romeo	Sierra	Tango	=CHOOSEROWS(B2:F6, 4)		

	A	B	C	D	E	F	G	H	I
1									
2		Alpha	Bravo	Charlie	Delta	Echo			
3		Foxtrot	Golf	Hotel	India	Juliet			
4		Kilo	Liam	Mike	November	Oscar			
5		Papa	Quebec	Romeo	Sierra	Tango			
6		Uniform	Victor	Whiskey	Vodka	Collapse			
7									
8									
9		Alpha	Bravo	Charlie	Delta	Echo	=CHOOSEROWS(B2:F6, 1, 3, 5, 1)		
10		Kilo	Liam	Mike	November	Oscar			
11		Uniform	Victor	Whiskey	Vodka	Collapse			
12		Alpha	Bravo	Charlie	Delta	Echo			

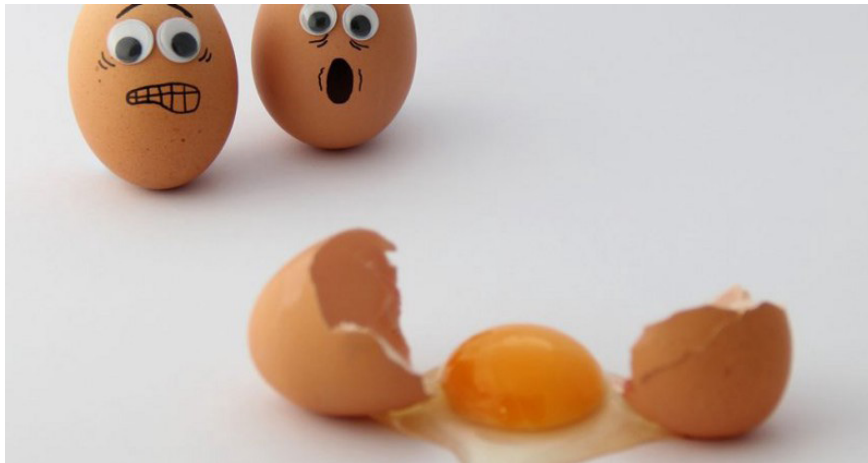
	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		#VALUE!						

=CHOOSEROWS(B2:F6, )

	A	B	C	D	E	F	G	H	I
1									
2		Alpha	Bravo	Charlie	Delta	Echo			
3		Foxtrot	Golf	Hotel	India	Juliet			
4		Kilo	Liam	Mike	November	Oscar			
5		Papa	Quebec	Romeo	Sierra	Tango			
6		Uniform	Victor	Whiskey	Vodka	Collapse			
7									
8									
9		#VALUE!							

=CHOOSEROWS(B2:F6, 2, 4, 6)

## The DROP function



The **DROP** function excludes a specified number of contiguous rows or columns from either the start or the end of an array. It has the following syntax:

**DROP(array, rows, [columns])**

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

- **array**: this is required and represents the selected array from which to drop the rows or columns
- **rows**: this is also required and denotes the number of rows to drop (exclude) from the top. If this number is negative, the values drop from the bottom of the **array**
- **columns**: this is optional and denotes the number of columns to drop (exclude). If this number is negative, the values drop from the end of the **array**.

It should be noted that:

- when **rows** or **columns** are not provided or missing, all rows and columns are returned
- if the absolute value of **rows** or **columns** is greater than the number of rows or columns in the array, then all rows or columns are supposed to be returned, but presently **#VALUE!** appears to be the favoured treatment
- Excel returns an **#CALC!** error to indicate an empty **array** when **rows** or **columns** is zero [0]
- Excel returns an **#NUM!** when **array** is too large.

Please see the examples below:

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		Foxtrot	Golf	Hotel	India	Juliet		<b>=DROP(B2:F6, 1)</b>
10		Kilo	Liam	Mike	November	Oscar		
11		Papa	Quebec	Romeo	Sierra	Tango		
12		Uniform	Victor	Whiskey	Vodka	Collapse		

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		Bravo	Charlie	Delta	Echo			<b>=DROP(B2:F6,, 1)</b>
10		Golf	Hotel	India	Juliet			
11		Liam	Mike	November	Oscar			
12		Quebec	Romeo	Sierra	Tango			
13		Victor	Whiskey	Vodka	Collapse			

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		Bravo	Charlie	Delta	Echo			<b>=DROP(B2:F6, -2, 1)</b>
10		Golf	Hotel	India	Juliet			
11		Liam	Mike	November	Oscar			

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		#VALUE!						<b>=DROP(B2:F6,, 6)</b>

## The EXPAND function



The **EXPAND** function expands (or pads) an array to specified row and column dimensions. It has the following syntax:

**EXPAND(array, rows, [columns], [pad with])**

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

- **array**: this is required and represents the selected array to be expanded
- **rows**: this is also required and denotes the number of rows in the expanded **array**. If this argument is missing (not bad for a required argument!), **rows** will not be expanded
- **columns**: this is optional and denotes the number of columns in the expanded **array**. Again, should columns not be specified, this dimension will not be expanded
- **pad with**: this is an optional value with which to pad. The default is *N/A*.

It should be noted that:

- if **rows** isn't provided or is empty, the default value is the number of rows in the **array** argument (as aforementioned)
- if **columns** isn't provided or is empty, the default value is the number of columns in the **array** argument
- if **pad with** is not provided and array has one value for that dimension, then that value is used. This operation is commonly referred to as array "broadcasting"; however, this does not appear to work presently
- Excel returns an **#VALUE!** error when the rows or columns argument is less than the **rows** or **columns** in the **array** argument
- Excel returns an **#N/A** error when **pad with** is greater than a single column or row
- Excel returns an **#NUM!** when **array** is too large.

Please see our penultimate examples below:

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie				
3		Delta	Echo	Golf				
4		Hotel	India	Juliet				
5								
6								
7		Alpha	Bravo	Charlie	#N/A			=EXPAND(B2:D4, 5, 4)
8		Delta	Echo	Golf	#N/A			
9		Hotel	India	Juliet	#N/A			
10		#N/A	#N/A	#N/A	#N/A			
11		#N/A	#N/A	#N/A	#N/A			

	A	B	C	D	E	F	G	H	I
1									
2		Alpha	Bravo	Charlie					
3		Delta	Echo	Golf					
4		Hotel	India	Juliet					
5									
6									
7		Alpha	Bravo	Charlie	Zulu				
8		Delta	Echo	Golf	Zulu				
9		Hotel	India	Juliet	Zulu				
10		Zulu	Zulu	Zulu	Zulu				
11		Zulu	Zulu	Zulu	Zulu				

**=EXPAND(B2:D4, 5, 4, "Zulu")**

	A	B	C	D	E	F	G	H
1								
2		Alpha						
3		Delta						
4		Hotel						
5								
6								
7		Alpha	#N/A	#N/A				
8		#N/A	#N/A	#N/A				
9		#N/A	#N/A	#N/A				
10								

**=EXPAND(B2, 3, 3)**

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie				
3		Delta	Echo	Golf				
4		Hotel	India	Juliet				
5								
6								
7		#VALUE!						

**=EXPAND(B2:D4,, 2)**

## The TAKE function



The TAKE function returns a specified number of contiguous rows or columns from either the start or the end of an array. It has the following syntax:

**TAKE(array, rows, [columns])**

The TAKE function has the following arguments:

- **array**: this is required and represents the selected array from which to take (extract) the rows or columns
- **rows**: this is also required and denotes the number of rows to take from the top. If this number is negative, the values are taken from the bottom of the **array**
- **columns**: this is optional and denotes the number of columns to take. If this number is negative, the values take from the end of the **array** also.

It should be noted that:

- when **rows** or **columns** are not provided or missing, all rows and columns are returned
- if the absolute value of **rows** or **columns** is greater than the number of rows or columns in the array, then all rows or columns are supposed to be returned, but presently **#VALUE!** appears to be the favoured treatment
- Excel returns an **#CALC!** error to indicate an empty **array** when **rows** or **columns** is zero [0]
- Excel returns an **#NUM!** when **array** is too large.

Please see the final examples below:

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		Alpha	Bravo	Charlie	Delta	Echo		=TAKE(B2:F6, 1)

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		Alpha						=TAKE(B2:F6, 1)
10		Foxtrot						
11		Kilo						
12		Papa						
13		Uniform						

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		Papa						=TAKE(B2:F6, -2, 1)
10		Uniform						

	A	B	C	D	E	F	G	H
1								
2		Alpha	Bravo	Charlie	Delta	Echo		
3		Foxtrot	Golf	Hotel	India	Juliet		
4		Kilo	Liam	Mike	November	Oscar		
5		Papa	Quebec	Romeo	Sierra	Tango		
6		Uniform	Victor	Whiskey	Vodka	Collapse		
7								
8								
9		#VALUE!						=DROP(B2:F6, 6)

### Word to the Wise

These functions are “hot off the press” and presently “in Beta”, so it is possible they may change and / or their behaviour may be modified. This should not deter you from trying these out. Compared to the recent onslaught of **LET** and **LAMBDA** related functions, the concepts in play are reasonably simple to understand and could prove highly useful for those looking to work with arrays more in Excel.

Bring on the next batch!

## More Updates for Excel for the Web

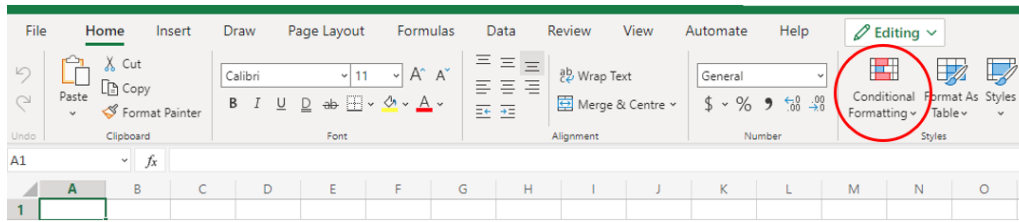
It really seems that the Microsoft Excel team has got its programming claws well and truly into the interwobble version. Excel for the Web has had further improvements this month:

- new Conditional Formatting experience
- Function library
- new Filter menu
- Insert Slicer
- Insert online pictures.

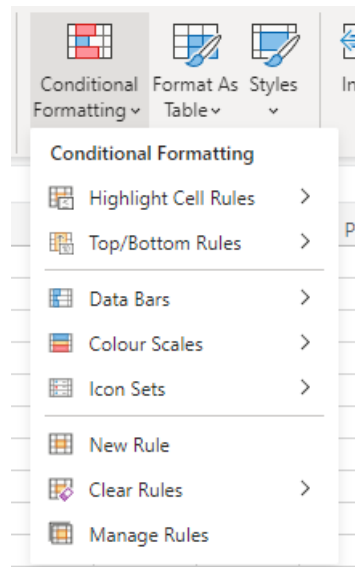
Let’s take a look.

## New Conditional Formatting Experience

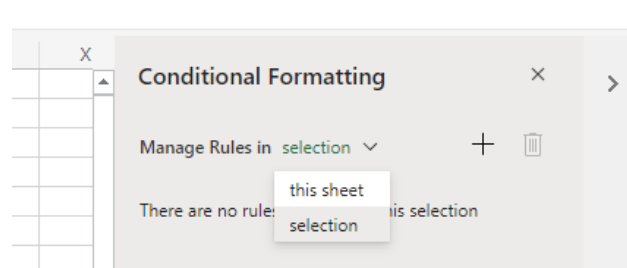
The new Conditional Formatting capabilities allow for simpler management, editing and creating formatting rules in Excel for the Web. Conditional formatting remains on the Home tab:



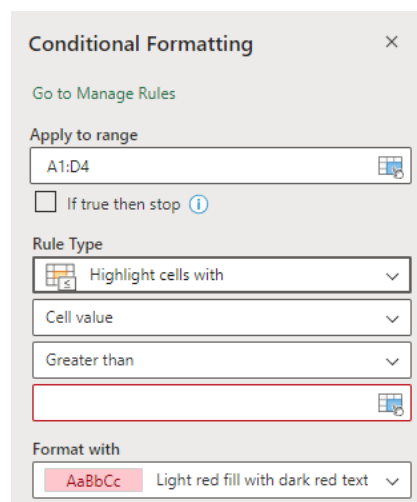
Clicking on this button activates the dropdown menu:



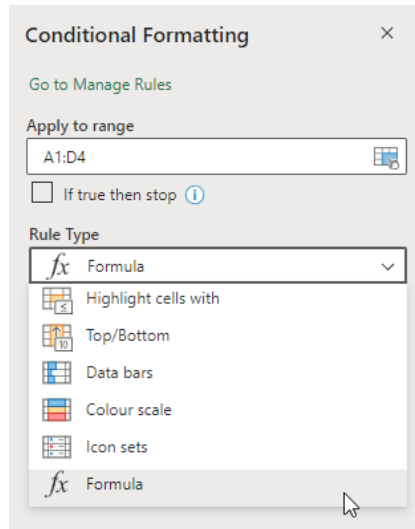
This acts in a similar way to its desktop counterpart, save for 'Manage Rule', which activates the 'Conditional Formatting' pane, viz.



Once you have decided on your selection, the '+' button allows you to create a rule where you may select your range



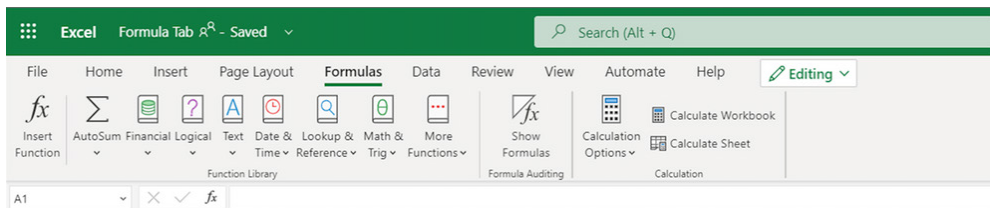
and your rule type



From then on, it works very similarly to Excel Desktop.

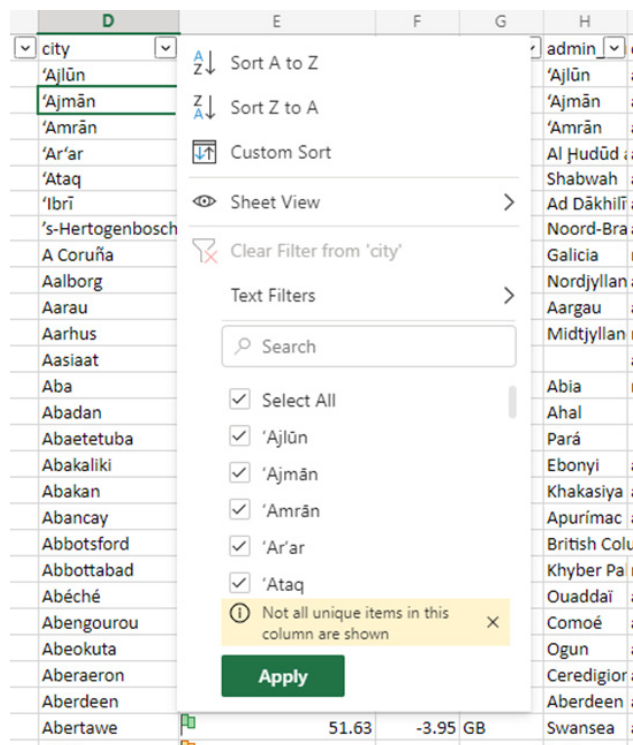
## Function Library

Many users rely on the function library to help find and identify the right function to use. This update brings Excel's familiar formula library to the web.



## New Filter Menu

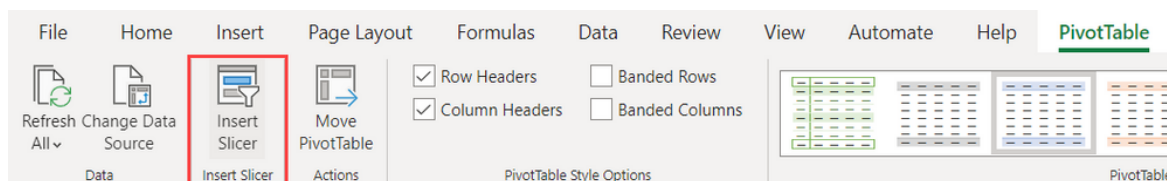
Filtering data is now easier and more convenient too:



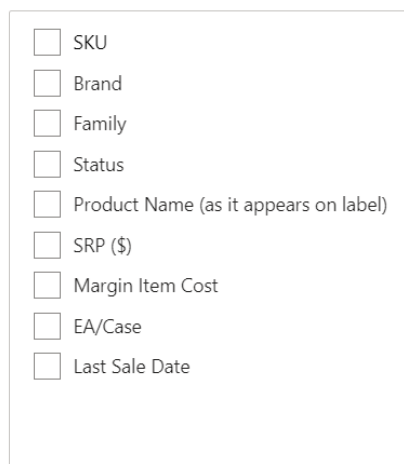


## Insert Slicer

We are pretty confident many of you will like this feature, as Slicers may now be added for PivotTables (Slicers for Tables do not yet appear to be supported). Simply click on your PivotTable and then go to the context specific tab:



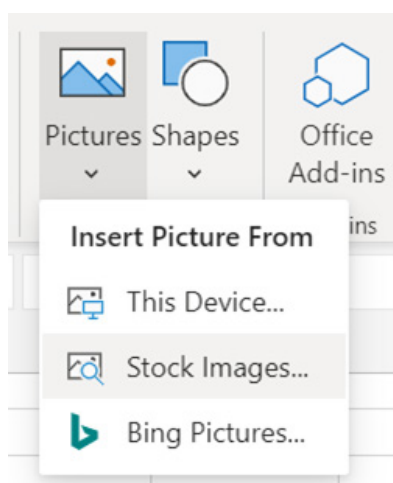
### Insert Slicers

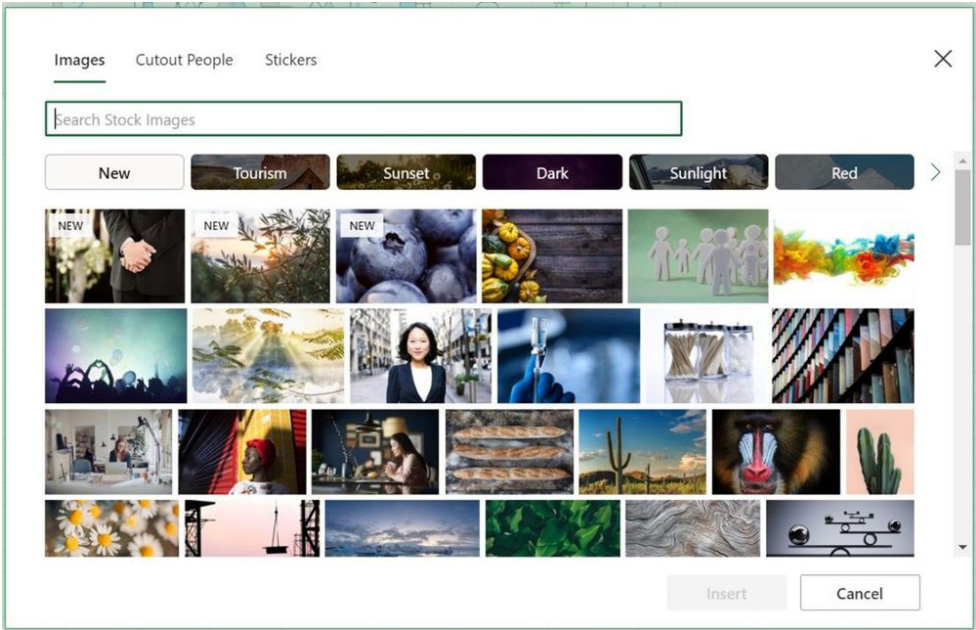


## Insert Online Pictures

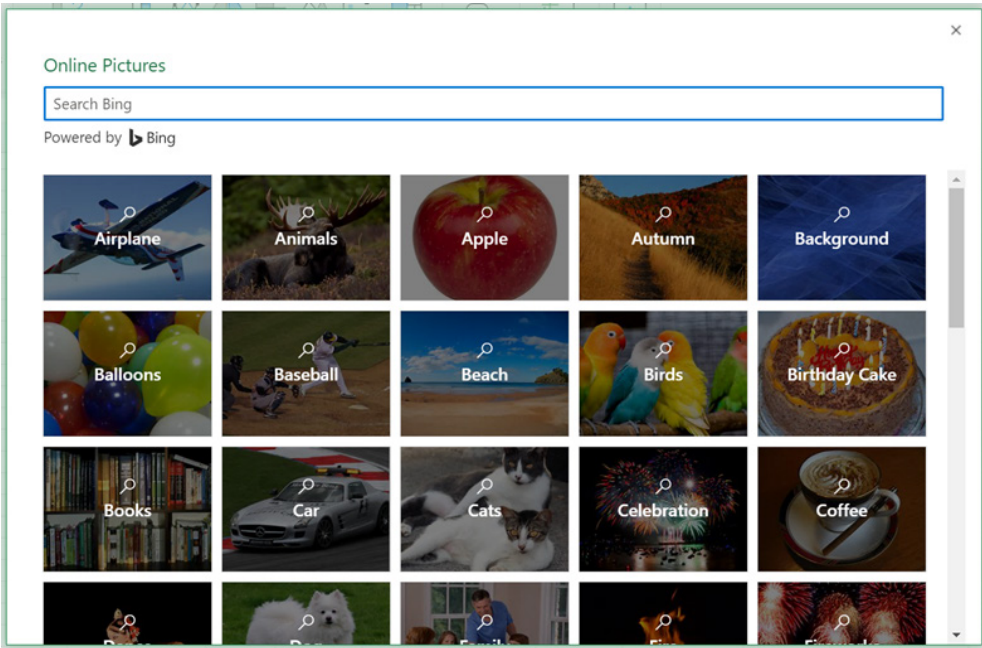
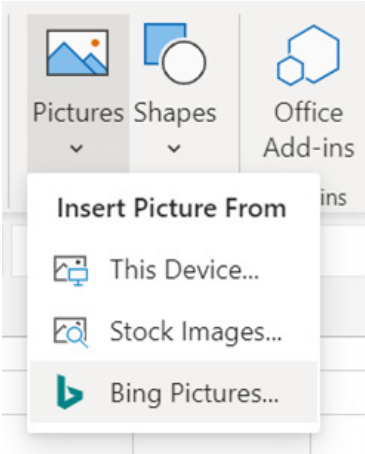
You may now find high-quality images from Microsoft Stock images and add them to your worksheet (use the categories on the 'Stock images' window to filter out the images as you need). Stock Images also provides you with Cutout People and Stickers on Excel for the Web. You can also

search Stock Images, Cutout people and Stickers using keywords in the Search bar. You can select multiple images across all categories and insert them onto the Excel worksheet.





Bing Pictures search is now available on Excel for the Web. Here, you may search for a picture on the web using keywords or use the prepopulated picture categories. You can select multiple pictures and insert them onto the Excel worksheet.



More improvements soon, we're sure!

## Modelling Working Capital

In the current uncertain economic climate, cash is often of paramount importance. Therefore, it is important in financial modelling to understand what working capital assumptions are and how they work.

If money is owed, the people who owe it are **Debtors** and, assuming no write-offs of sales made, the amounts owed are known as **Accounts**

**Receivable** or simply **Receivables**. This amount owed is viewed as an asset of the business. It isn't cash but it should convert to cash *soon*. Cynical businesses may quite rightly think if only it were that simple, but it is for the purposes of this case study.

Consider the following example:

### Control Account

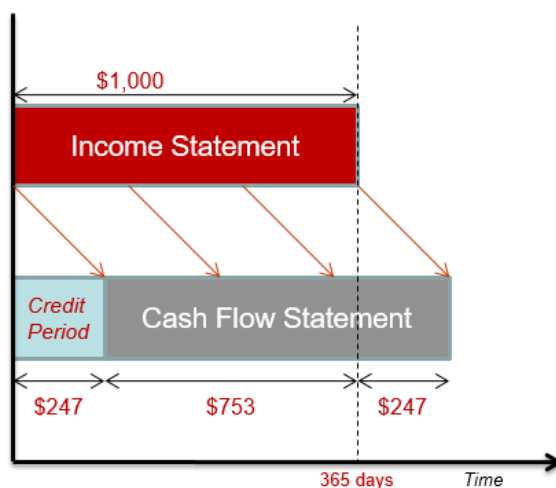
Opening Debtors	-	BS
Sales in Period	1,000	IS
Cash Receipts	(753)	CFS
Closing Debtors	<u>247</u>	BS

Imagine a company has just started off in business (*i.e.* has no amounts due) and generates sales of \$1,000 in the period. At the end of the period, assuming no bad debts, \$753 has been paid, leaving a closing debtor balance of \$247. This difference is what I refer to as the **working capital adjustment**. If we had modelled the sales of \$1,000 in the period, how might we generate the cash receipts forecast such that if the assumptions changed, the receipts would calculate appropriately?

Clearly, if I am given the closing debtor balances, the problem becomes trivial, so I will assume that this is not so. Therefore, I am going to

consider an alternative approach and some of the associated underlying issues that need to be considered when modelling. Let me first derive an alternative method.

I will assume that the sales accrue evenly over the period of time and for the sake of this example, that period is one year (365 days). Presuming (i) all sales are made on credit terms, (ii) all customers pay their invoices on the day the amounts fall due and (iii) no bad debts are incurred, this can be reflected graphically as follows:



Clearly, the credit period is the "gap" at the beginning of the time period, *i.e.*  $247/1000 \times 365 \text{ days} = 90 \text{ days}$ . This can be represented formulaically as:

$$\text{Days Receivable} = (\text{Closing Debtors} \times \text{Days in Period}) / \text{Sales in Period}$$

Rearranging, this becomes:

$$\text{Closing Debtors} = (\text{Sales in Period} \times \text{Days Receivable}) / \text{Days in Period},$$

*e.g.* in our example:  $247 = (1000 \times 90) / 365$ .

Therefore, in modelling, we often set the number of days receivable (and days payable) as key assumptions for cash flow forecasting. However, it is not always as simple as that. Let me explain. Consider we are planning to build a monthly model (assuming 30 days in a month) and sales for the month are again \$1,000. Debtor days remain at 90 days.

Based on these calculations, we would generate the following control account:

### Control Account

Opening Debtors	-	BS
Sales in Period	1,000	IS
Cash Receipts	2,000	CFS
Closing Debtors	<u>3,000</u>	BS

Erm, that's right: make sales of \$1,000 and have \$3,000 (= 90/30 x 365) owing to you by the end of the month. Also, the company pays \$2,000 to customers a reclaimable \$2 for each \$1 spent. That's nonsense – and yet, as an experienced model auditor I have seen this erroneous calculation crop up on a regular basis. The problem is, in this current economic climate most businesses want to prepare monthly – sometimes weekly and even daily – cash flow projections. Clearly, if the days receivable or days payable assumption exceeds the number of days in each forecast period this approach is inappropriate and will lead to calculation errors. This is an example of where we ought to employ error checks to ensure that our inputs do not breach this key assumption.

There are alternatives. If payments are made exactly one month or two months or three months later (and so on), the resolution is simple: the receipts can be calculated using a simple **OFFSET** (displacement) formula.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P				
6	Monthly Calculations Example																			
7	Assumptions																			
11	Days in an Average Month																			
13	Required for calculations										30		days							
17	Days Receivable																			
18	Days Receivable										75		<table border="1"> <thead> <tr> <th>Complete Mths</th> <th>Partial Mth</th> </tr> </thead> <tbody> <tr> <td>2</td> <td>50%</td> </tr> </tbody> </table>				Complete Mths	Partial Mth	2	50%
Complete Mths	Partial Mth																			
2	50%																			
21	Calculations																			
23	Month Number																			
24	Opening Debtors																			
25	Sales in Period																			
26	Cash Receipts																			
27	Closing Debtors																			
28																				
29																				

In this illustration (above), cells **J18** and **K18** break the number of days receivable (cell **G18**) into the number of whole months and residual proportion respectively, assuming that each month has 30 days (cell **H13**).

The key formula here is the calculation for Closing Debtors (Cash Receipts is simply the balancing figure). For example, the formula in cell **J28** (above) is:

$$=IF(\$J\$18,SUM(OFFSET(J26,,1,-MIN(\$J\$18,J\$23))),)+IF(J\$23-$J\$18<=0,,OFFSET(J26,-$J\$18)*$K$18)$$

It may seem a little complex upon first inspection, but it's not as bad as it seems. Essentially, there are two parts to this formula identified by the two added **IF** statements:

1. **IF(\$J\$18,SUM(OFFSET(J26,,1,-MIN(\$J\$18,J\$23))),)** considers the completed number of months where sales remain outstanding and adds up the sales for these periods.

In essence, this part of the formula checks that the number of completed months is not zero (in this case the amount is just zero), and assuming this is not the case, it sums the sales for the relevant number of completed months (*i.e.* starts with

Rather than consider that situation, let me complicate the scenario slightly. Imagine we are building a monthly forecast model, but that the days receivable figure is 75 days. For the purposes of keeping this article reasonably brief, I will simplify the problem by assuming an average number of days in a month (say, 30). Using this simplifying assumption, this will mean that payments are made on average 2.5 (2.5 = 75 / 30) months after the sale has been made.

That 2.5 months figure is important. The integer part (2) denotes how many complete months (including the current month) have sales payments outstanding. The residual (0.5 or 50%) shows the proportion of the month preceding these complete months that is also outstanding. With this borne in mind, the **OFFSET** function can now come to the rescue, *viz.*

the current month and then considers the sales in previous months, working from right to left in the spreadsheet). The **MIN** formula is required to ensure that the model does not try to include periods prior to the beginning of the forecast period).

2. **IF(J\$23-\$J\$18<=0,,OFFSET(J26,-\$J\$18)\*\$K\$18)** considers the residual (remaining) amount for the month before the earliest completed month. For example, if the credit period is 2.5 months and the current month is April, then March and April will be "whole months" where no payment has been received, with half of February's monies still outstanding too.

The reason for the **IF** statement here is to prevent calculations considering periods before the beginning of the forecast period.

To clarify, consider the Closing Debtor figure of \$1,050 in Period 5 (above, cell **N28** in the illustration). This is calculated as the sales for Periods 4 and 5 (400 + 500 respectively), plus half of the Period 3 sales (300 x 0.5 = 150), *i.e.* 400 + 500 + 150 = 1,050.

Working capital adjustments may become even more complicated. What if payments are not made evenly? Or that some sales are written off as payments are never made (*i.e.* bad debts)?

More Sophisticated Monthly Example												
Assumptions												
Month Number												
Forecast Sales Revenue												
Cash Receipt Profile												
Bad Debt												
Month Written Off												
Amount Written Off												
Calculations												
Simple Grid Method												
Month Number												
Sales: Period 1												
Sales: Period 2												
Sales: Period 3												
Sales: Period 4												
Sales: Period 5												
Sales: Period 6												
Sales: Period 7												
Sales: Period 8												
Sales: Period 9												
Sales: Period 10												
Sales: Period 11												
Sales: Period 12												
Cash Receipts												
Control Account												
Month Number												
Opening Debtors												
Sales in Period												
Bad Debts Written Off												
Cash Receipts												
Closing Debtors												

Some of you realise this looks like a depreciation grid. I shall discuss this concept more later. For the meantime, in this illustration it is worth noting that the Cash Receipt Profile percentages do not add up to 100%. This is deliberate – the missing 5% is the assumed bad debt here.

There is one other method. Large infrastructure projects may simply

have cash payments input rather than calculated, triggered by certain milestone payments – these just require specific **IF** formulae. This goes to show you that bigger does not always mean more complex.

Do remember to consider working capital implications in your financial models!

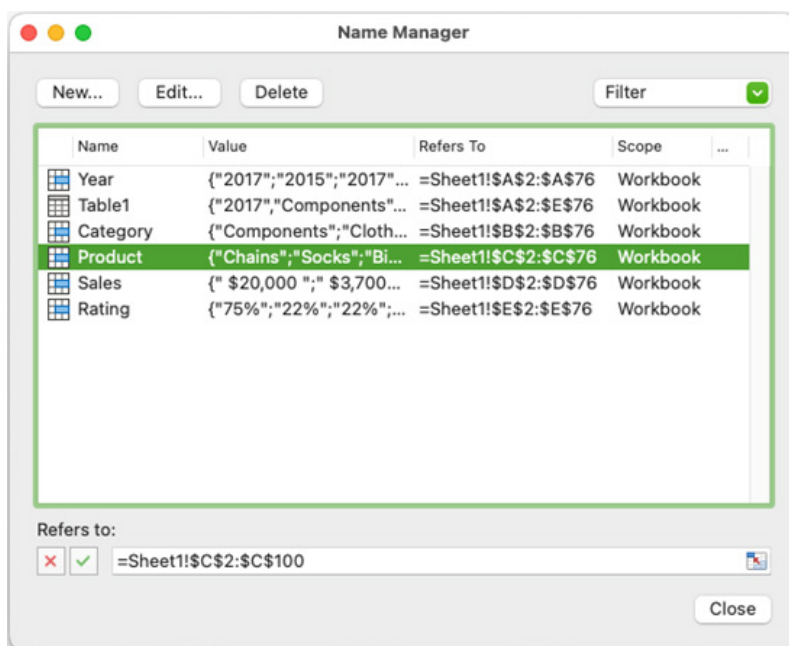
## Improved Name Manager in Excel for Mac

There is a new Name Manager in Excel for Mac, which now emulates all the capabilities of Excel for Windows (allegedly!). Let's just not mention how long this has taken to happen...

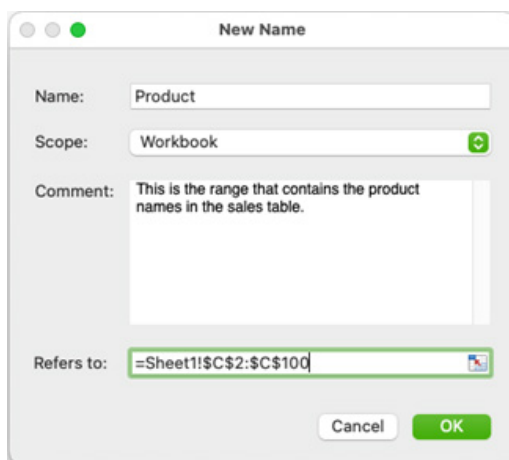
Previously, the Name Manager in Excel for Mac allowed you to create

- select 'Name Manager' to open a dialog box where you can view the complete list of defined names in your workbook, which includes all the important information about your names and can be filtered. This dialog box makes it easy to find, add, remove or edit the names

and edit defined names, but it didn't give you full control of those names. But no more. To try out the new experience, go to the Formulas (sic) tab and access the 'Defined Names' section of the Ribbon. In this section you may:



- select 'Define Name' to set the scope and add a comment to your defined names. This 'New Name' dialog provides you with better control over how the names are applied in your workbook and helps to document the purpose of each name.

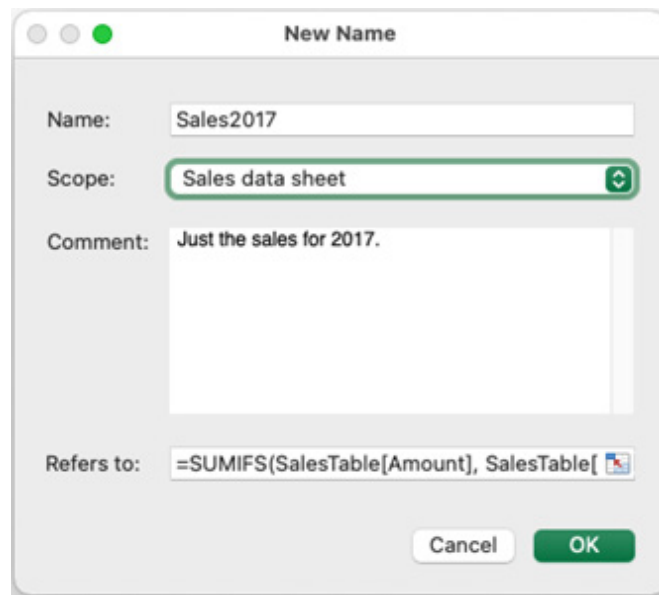


You should note that you may also resize the 'Name Manager' and 'Define Name' ('New Name') dialog boxes, which makes them easier to use and allows you to view more information (this will prove useful if you're using the new **LAMBDA** function).

Sometimes you may have names with **#REF!** errors, but these were

difficult to spot without the 'Name Manager' dialog. Now, you can quickly scan the list of names or use the Filter option to find names that have errors and fix or remove them. To do this, select the Formulas tab, click 'Name Manager', then click the Filter dropdown and select 'Names with Errors'.

You may also create a new name with a comment and set the scope to a specific sheet. To do this, click the Formulas tab and select **Name Manager** -> **New**. Set the Scope to 'workbook' or pick a specific sheet if you want the defined name to only work in formulae on that sheet.



Previously, you weren't able to see whether a name was scoped to a specific sheet or to the entire workbook. Now, you may look at the Scope field in the 'Name Manager' to quickly see the scope for all your defined names.

This feature is available now to Microsoft 365 subscribers with Version 16.58 or greater. To get the latest updates to Excel there are two options:

1. if you installed Excel from the Mac App Store, then go to the App Store and check for updates
2. if you installed Excel another way, go to the Help menu in Excel and choose 'Check for Updates'.

## 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...*

This month's challenge is straightforward. Below is a table:

	A	B	C	D	E	
3						
4		<b>Business Unit</b>			<b>Quarter</b>	<b>Sales</b>
5		A		1	10	
6		B		1	20	
7					30	
8		C		2	40	
9		B			50	
10				3	60	
11						

Notice the blanks? Yes, that is part of the problem this month. The challenge this month – can you set up a single formula to sum the sales amounts based on the values in the Business Unit and Quarter columns? Note that your formula has to be capable of reading the blank values as a criterion.

For instance, your formula should return with a total of 50 when we enter B in the Business Unit and a blank value in Quarter.

Business Unit	B
Quarter	
Total Sales	50

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

## Charts and Dashboards

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

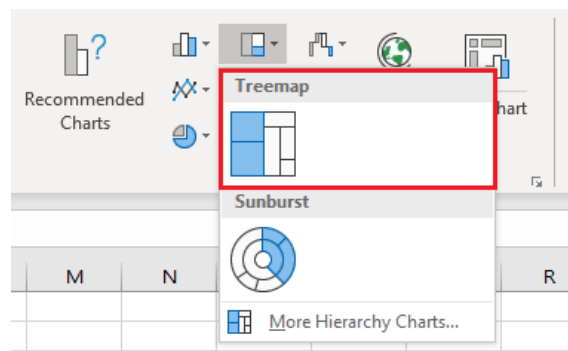
Like last month's article on Sunburst charts, a Treemap chart also provides a hierarchical view of data, but instead of displaying the data in a circular format, a Treemap chart represents the data using rectangles. The top-level category is displayed as a large rectangle with each element

within the category shown as a smaller rectangle within the larger one. The data table needs to be configured exactly the same way I would for the Sunburst chart:

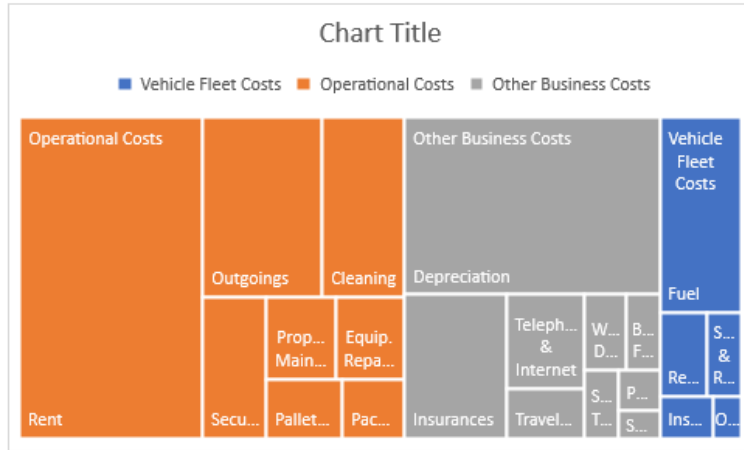
Business Expenses Excluding Payroll for 2018/19			
Expense Group	Expense Account	Amount	
Vehicle Fleet Costs	Fuel	\$ 18,036	
	Registration	\$ 4,512	
	Insurance	\$ 2,568	
	Servicing & Repairs	\$ 3,216	
	Other Vehicle Expenses	\$ 1,296	
Operational Costs	Rent	\$ 66,840	
	Outgoings	\$ 24,432	
	Property Maintenance	\$ 6,456	
	Equip. Repairs & Maint.	\$ 6,408	
	Security	\$ 10,284	
	Cleaning	\$ 16,716	
	Pallets & Cartons	\$ 5,136	
	Packag. & Consum.	\$ 4,176	
	Other Business Costs	Staff Train. & Dev.	\$ 2,568
		Staff Amenities	\$ 1,296
Telephone & Internet		\$ 8,364	
Insurances		\$ 16,716	
Postage & Couriers		\$ 1,932	
Travel Costs		\$ 4,476	
Waste Disposal		\$ 3,564	
Depreciation		\$ 51,420	
	Bank Fees & Interest	\$ 2,892	

Creating the Treemap chart is almost identical to creating the Sunburst chart: select the data table, go to the Insert tab on the Ribbon, select the Treemap chart from the small middle icon across the top row of the

Charts section, or use the 'Recommended Charts' icon or the small arrow in the bottom right corner. There is only one type of Treemap chart available in Excel:



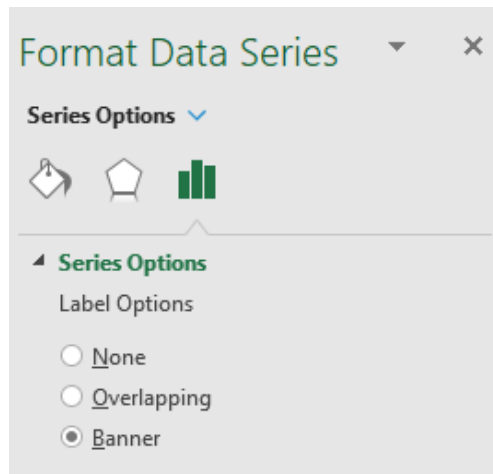
The chart initially appears like this:



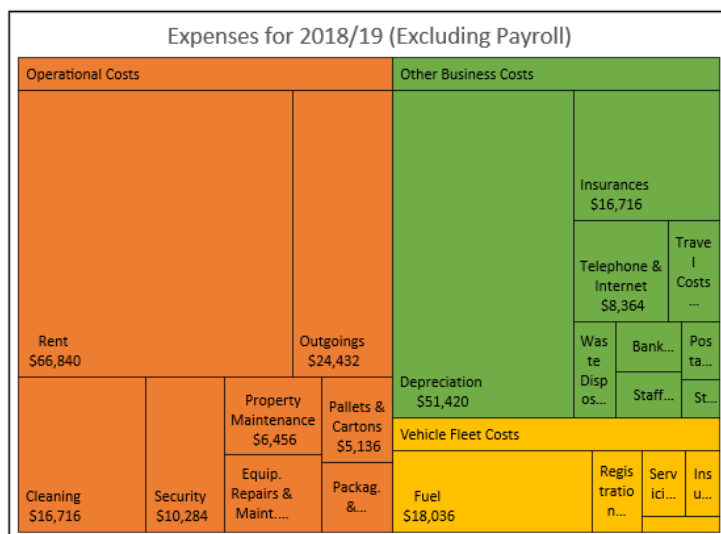
The formatting options available for the Treemap chart are also nearly the same as for the Sunburst chart, including the limitations on visibility of all data labels and values, and not being able to reorder the data on the chart as Excel automatically sorts and collates the data.

One unique formatting option available for Treemap charts is associated with how the top hierarchy labels are displayed. In the initial Treemap chart, as above, you can see the legend at the top listing the categories of the top-level hierarchy, but each of the categories is also a data label within the largest box of each series.

If I click on one of the rectangles thereby selecting a data series, right-click and choose 'Format Data Series', under 'Series Options', I can choose how I want the top-level categories displayed. The default is Overlapping, so the label sits on top of the largest rectangle in each series. An alternative is to set the 'Label Options' to None removing the categories from the chart altogether. The third option is to set the preference to Banner, which displays the categories in their own small rectangle above their data series. If I choose Overlapping or Banner, the legend, which is now unnecessary, may be removed.



I apply some general formatting and the chart is ready:



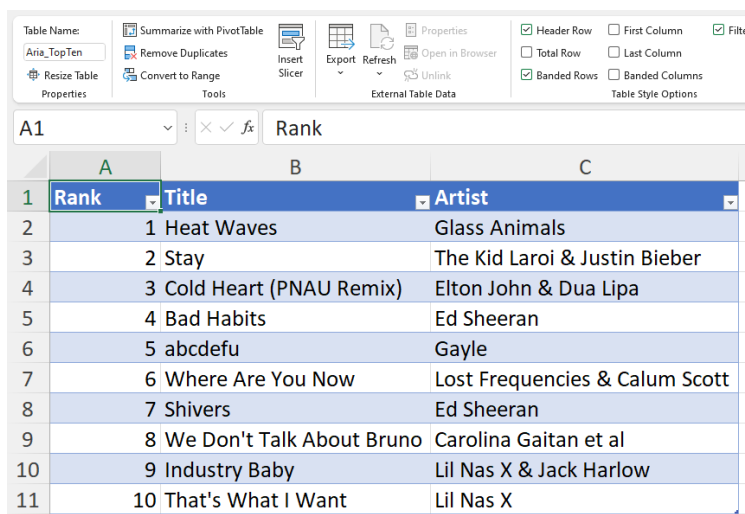
More next month...



## 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 continue looking at using ListObjects to manipulate Tables within an Excel workbook in VBA, this time we continue to consider ListColumns.

We thought we'd take a look at the Top 10 songs in Australia this month according to the local ARIA chart. I have created an **Aria\_TopTen** table:



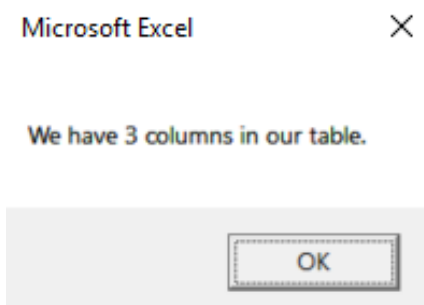
Rank	Title	Artist
1	Heat Waves	Glass Animals
2	Stay	The Kid Laroi & Justin Bieber
3	Cold Heart (PNAU Remix)	Elton John & Dua Lipa
4	Bad Habits	Ed Sheeran
5	abcdefu	Gayle
6	Where Are You Now	Lost Frequencies & Calum Scott
7	Shivers	Ed Sheeran
8	We Don't Talk About Bruno	Carolina Gaitan et al
9	Industry Baby	Lil Nas X & Jack Harlow
10	That's What I Want	Lil Nas X

Visually, it is clear that there are three [3] columns in the table. **ListColumns** may be used to determine how many **columns** are in a table. This is done by using the Count method.

```
Option Explicit
Sub Test()
    Dim MyTable As ListObject
    Set MyTable = Range("Aria_TopTen").ListObject

    MsgBox "We have " & MyTable.ListColumns.Count & " columns in our table."
End Sub
```

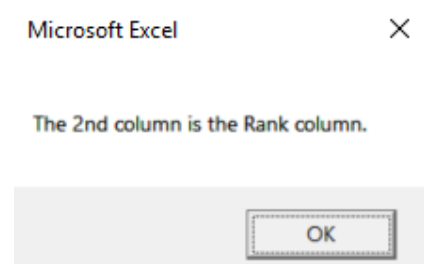
The result is:



However, what if I wanted to know what was the second column in the table? In this case, the **Item** method can help impart that information:

```
Sub Test2()
    Dim MyTable As ListObject
    Set MyTable = Range("Aria_TopTen").ListObject

    MsgBox "The 2nd column is the " & MyTable.ListColumns.Item(1) & " column."
End Sub
```



You should note that this is directly equivalent to using the **Cells** property of **HeaderRowRange** as examined in a previous article.

More next time.

## Power Pivot Principles

We continue our series on the Excel COM add-in, Power Pivot. This month, we review the DAX syntax naming requirements and conventions when it comes to tables, columns and measures.

A single Power Pivot window can contain numerous tables with several columns in each table. Couple this with the measures associated with each table and things may get a little confusing while working with such a large dataset. It would be only natural for us to give each table, column and measure a unique name... right?

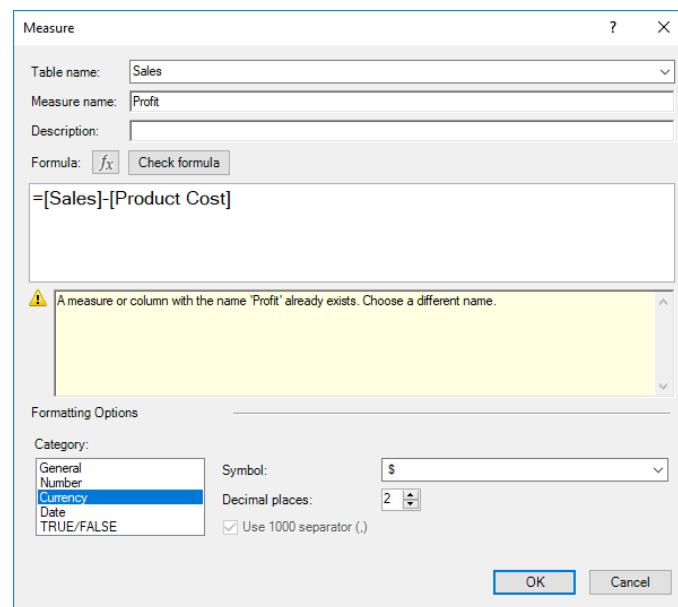
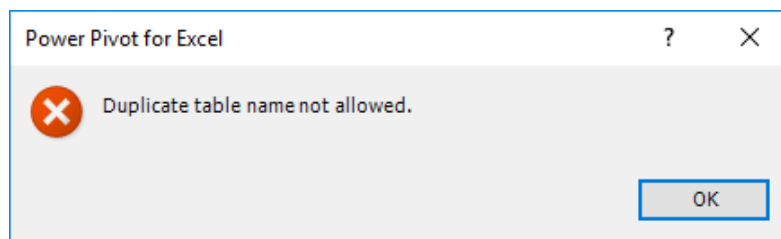
Here are some built-in rules and safety nets in Power Pivot that we should keep borne in mind while working with our datasets.

We have touched in this topic briefly in a different blog explaining the syntax differences between tables, fields and measures, which may be found here: <https://www.sumproduct.com/blog/article/power-pivot-principles/ppp-tables-fields-and-measures>.

Here are the rules when naming tables, columns and measures:

- within a single database all tables must have unique names
- the names of the columns within each table must be unique
  - columns in different tables may possess the same names
  - when the same column name is referenced from two or more tables, we must use a fully qualified name (see last month's newsletter)
- all objects (tables, fields, and measures) are **case insensitive**
  - for instance, the name 'DATE TABLE' and 'Date Table' would represent the same table.

It might be worth pointing out that Power Pivot does have a built-in duplicate name check, just in case we forget and give two tables or measures the same name:



If we give two columns within the same table duplicate names, Power Pivot will override the name and add a numerical counter at the end of the name, viz.

FiscalYear	FiscalYear2	FiscalYear3
2017		
2017		
2017		
2017		

Power Pivot is friendly to work with after all, just remember to give your tables, columns and measures unique and meaningful names!

More *Power Pivot Principles* next month.

# Power Query Pointers

Each month we'll reproduce one of our articles on Power Query (Excel 2010 and 2013) / Get & Transform (Office 365, Excel 2016 and 2019) from [www.sumproduct.com/blog](http://www.sumproduct.com/blog). If you wish to read more in the meantime, simply check out our Blog section each Wednesday. This month, we continue to look at how to use the **M** function **List.Accumulate**, on this occasion, when the seed is not zero.

As I described in last month's newsletter, the **List.Accumulate** function can transform data by performing several steps at once. As a reminder, I provide the description below once more:

"...Accumulates a result from the list. Starting from the initial value seed this function applies the **accumulator** function and returns the final result..."

**List.Accumulate(list as list, seed as any, accumulator as function) as any**

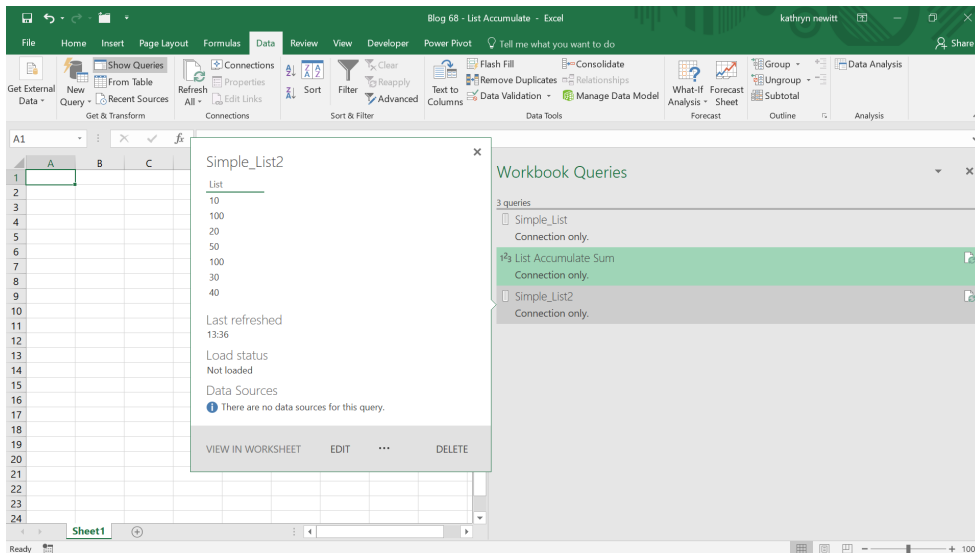
Where:

- list** = the list to check
- seed** = the initial value seed
- accumulator** = the value accumulator function.

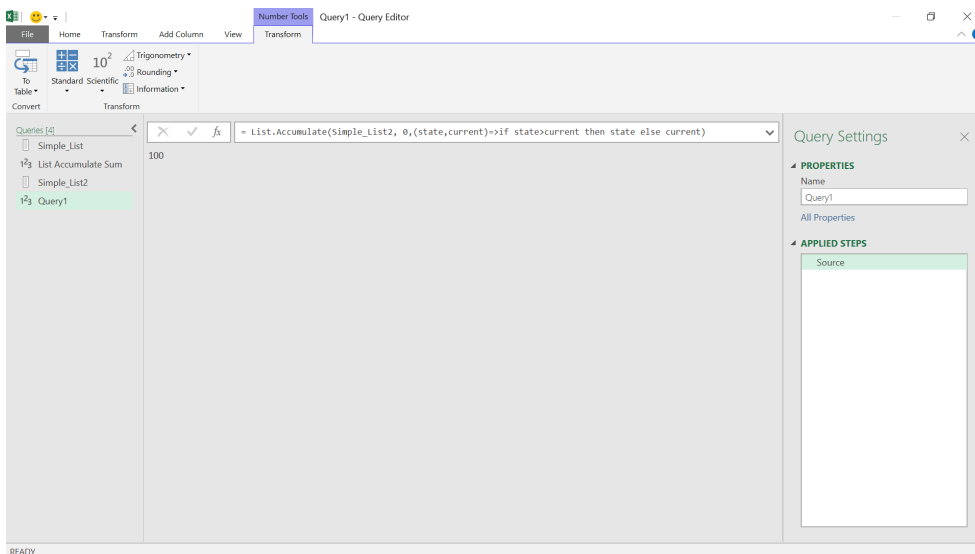
Last time, I looked at some examples where the 'seed' is usually set to zero [0] (summing and counting). This time, I shall consider examples where the 'seed' is not zero. I am going to provide some examples as a demonstration here, namely maximum, multiplication, concatenation and record conditions.

## Maximum

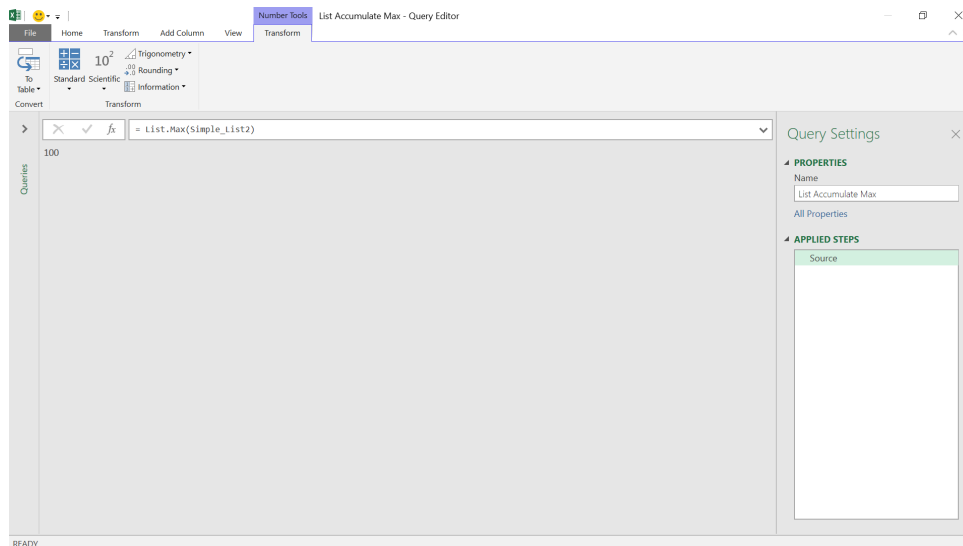
There is a **List.Max** function in the **M** language which could be used to do achieve this goal, so I will be comparing the results from this with my **List.Accumulate** calculation. I begin with another list, just for a change:



I create a new blank query which uses **List.Accumulate**.



Just to show that **List.Max** gives me the same value:



Going back to my **List.Accumulate** syntax, I have:

**= List.Accumulate(Simple\_List2, 0, (state,current) => if state > current then state else current)**

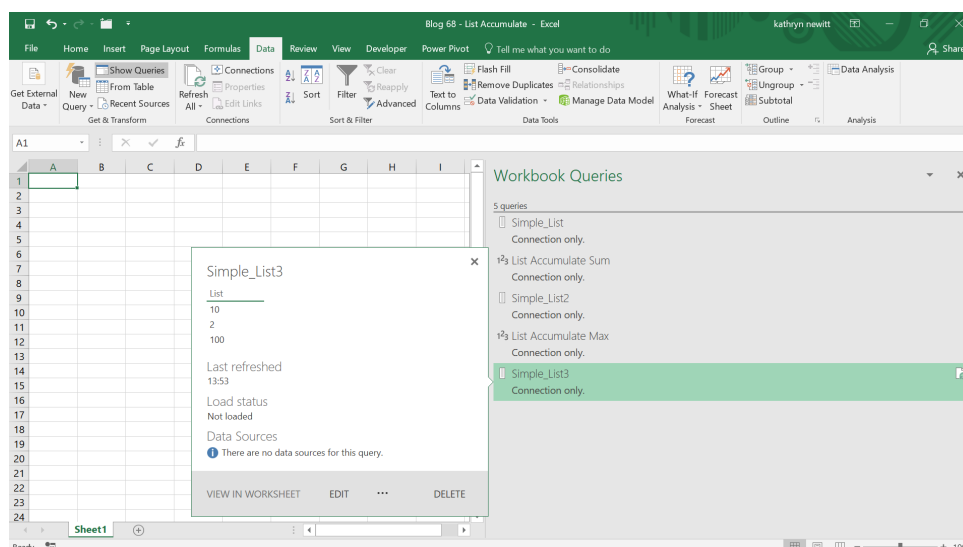
This time, **List.Accumulate** goes through each item in my list and updates 'state' if the 'current' value is greater than the greatest so far.

Current	State	New Value for State
10	0	10
100	10	100
20	100	100
50	100	100
100	100	100
30	100	100
40	100	100

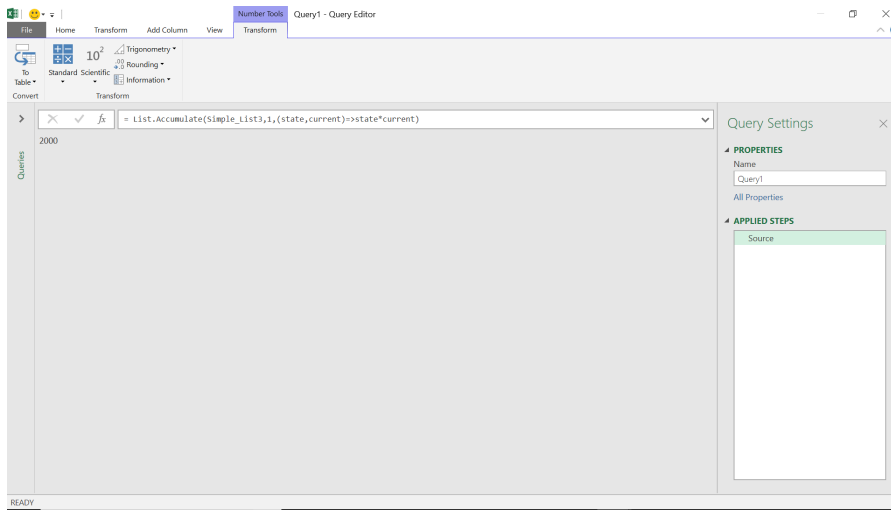
In this case, I have set the 'seed' to zero [0], but it just needs to be less than or equal to the maximum for the function to work. For minima, it would be necessary to set the 'seed' to a value higher or equal to the minimum (so this is not a seedless example!).

### Multiplication (or Division)

When using **List.Accumulate** to perform calculations involving multiplication or division, the 'seed' is important. It can't be zero because it will either render the product zero or invalidate the division. Allow me to illustrate with another list.



This time, my blank query will look at the product of the items on my list.



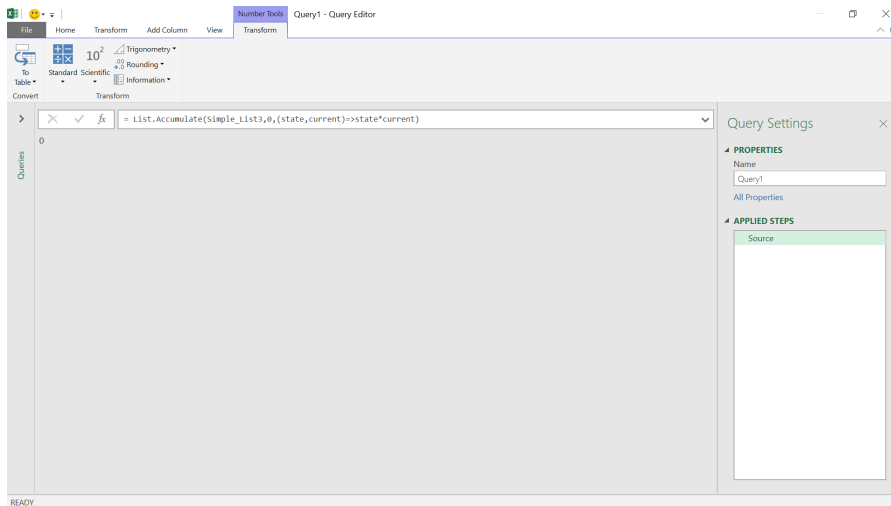
This time, my function is

**= List.Accumulate(Simple\_List3, 1, (state,current) => state \* current)**

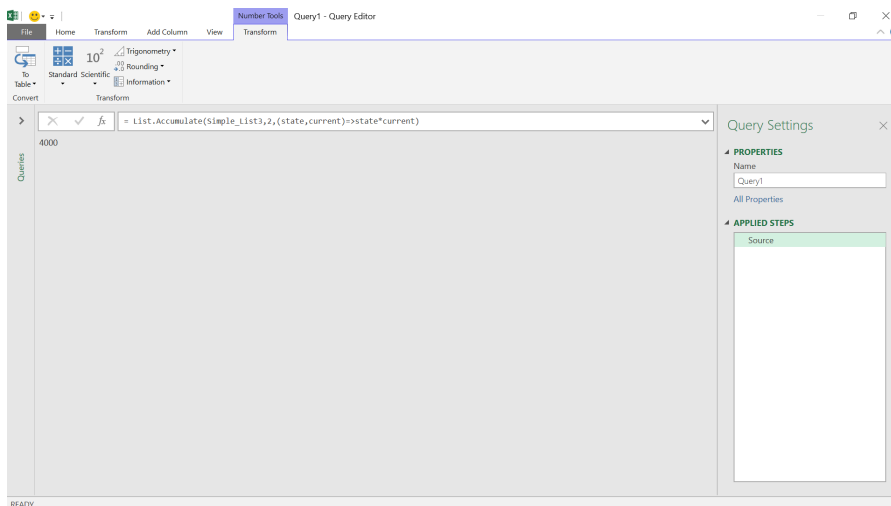
which acts on my list as follows:

Current	State	State*current
10	1	10
2	10	20
100	20	2000

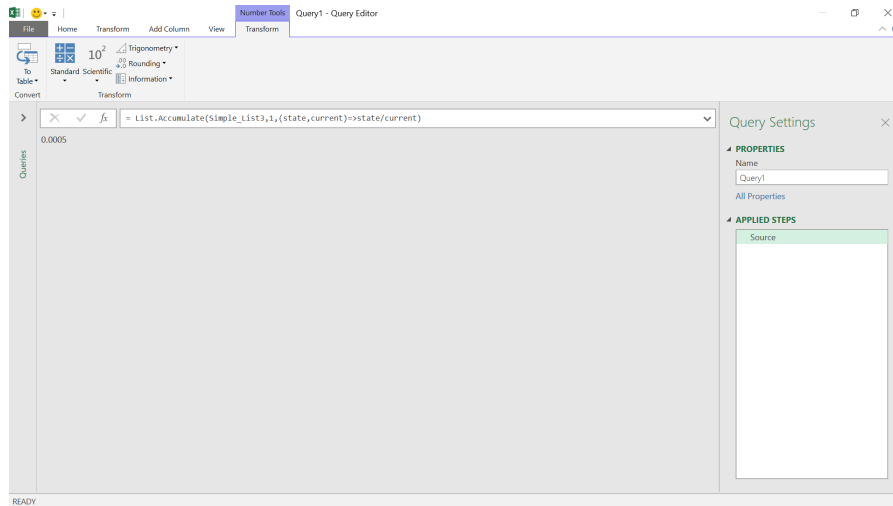
In this case, 'seed' is a factor in my calculation. If I change it to zero, then my result is zero, because the first value for the state will be zero.



If I change 'seed' to two [2], the first value of 'state' will be 2, and the whole calculation will be multiplied by 2.



The division is similar:



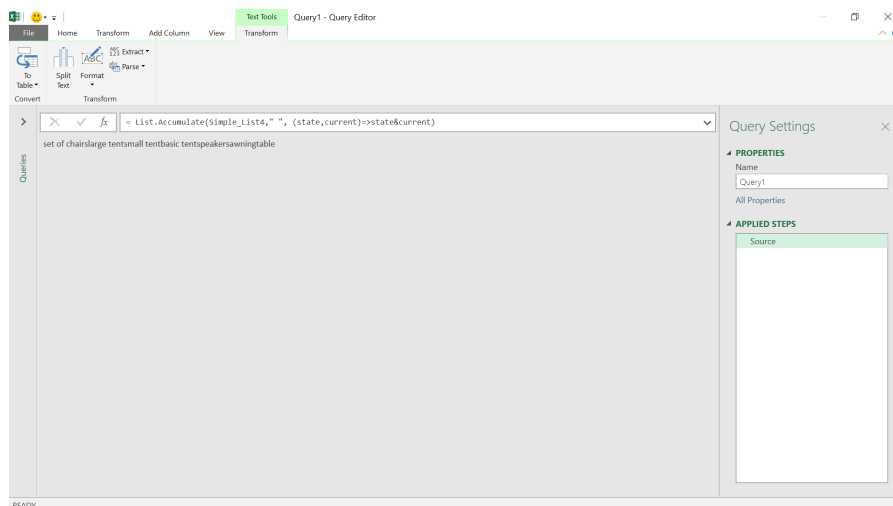
where **List.Accumulate** acts on my list accordingly:

Current	State	State/current
10	1	0.1
2	0.1	0.05
100	0.05	0.0005

Clearly, the seed can't be nought [0] as my first step would result in an error caused by dividing by zero.

### Concatenating

I will use my text list 'Simple\_List4' for this, which was a list of tent equipment. I can concatenate all my items using **List.Accumulate**:



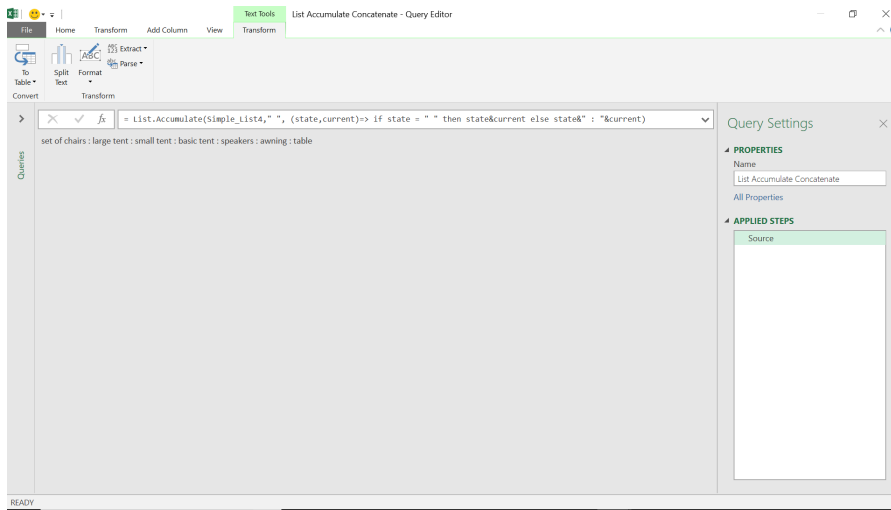
This has extracted all the items in my list and put them into a single string.

**= List.Accumulate(Simple\_List4, "", (state, current) => state&current)**

I could add a delimiter to make it clearer. The delimiter would only be needed after the first item, so I would need to check if it was the first item, and if not, add my delimiter.

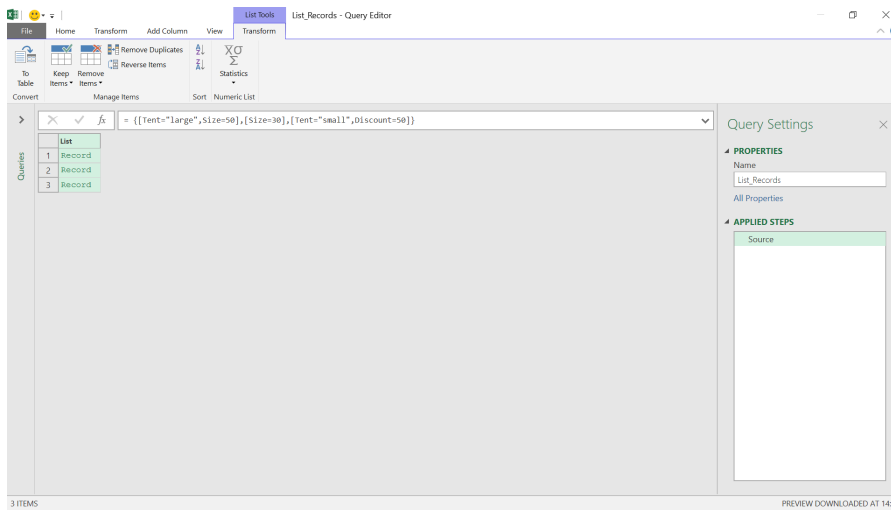
**= List.Accumulate(Simple\_List4," ", (state,current) => if state = " " then state&current else state&" " : "&current)**

The results are shown below:

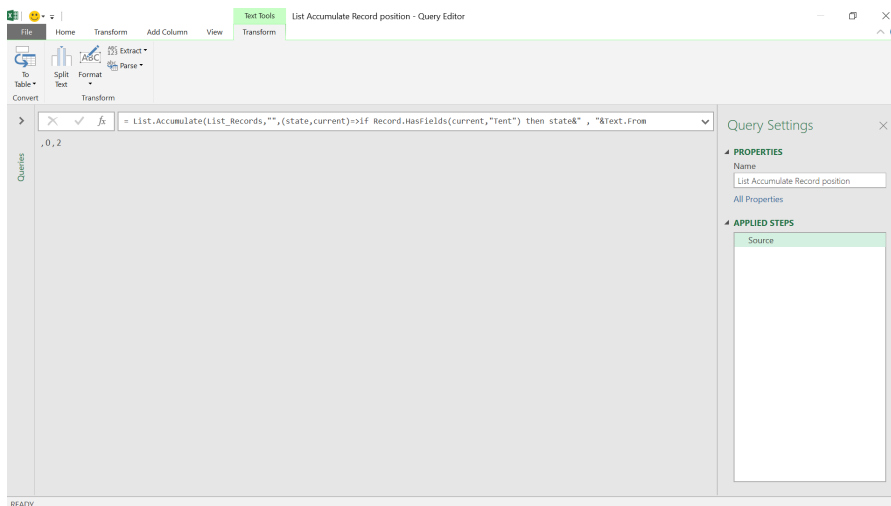


### Conditions on Records

The examples so far show how **List.Accumulate** works, but one area where it can be very useful, is when I have a list of records. I have an example below where my list is made up of records:



Each record has some item data, but I want to know the position of records that mention a 'Tent'. This would take several steps, as I can't see the contents of the records in the list, but I can use **List.Accumulate**.



I haven't tried to stop the comma delimiter appearing before the first record found as I want to keep the syntax as simple as it can be.

```
= List.Accumulate(List_Records, "", (state, current) => if Record.HasFields(current, "Tent") then state & ", "& Text.From(List.PositionOf(List_Records, current)) else state)
```

This looks at each item on the list, and checks for the field 'Tent'. If it is found, then the function returns the position number of the record (which starts at 0, not 1). As the function loops round each item, the delimiter is placed between each result.

Current	state	new value for state
Record (1)	""	,0
Record (2)	,0	,0
Record (3)	,0	,0,2

which tells me that records 1 and 3 have the field 'Tent' in them.

More next month!

## Power BI Updates

There is no letting up with these updates: 2022 is in full swing now! This month's updates see error bars for Line charts and dynamic format strings support for all chart elements. Also, there are updates to the Azure Maps visual and Sensitivity labels, datasets hub and Power BI Goals.

The full list is as follows:

### Reporting

- New Format Pane in Preview: General Availability coming in May
- Error bars in Preview
- Dynamic format strings now supported for all chart elements
- Updates to the Azure Maps visual in Preview
- Sensitivity labels update
- Multi-row card selection

### Data Connectivity and Preparation

- BitSight Security Ratings (New Connector)
- Bloomberg Enterprise Data and Analytics (Connector Update)
- Anaplan (Connector Update)
- FactSet Analytics (Connector Update)
- AssembleViews (Connector Update)

### Service

- Datasets hub improvements
- Power BI Goals enhancements:
  - Custom statuses in Scorecards
  - Power BI Goals Teams notification

### Embedded Analytics

- New improved method for deploying Power BI Embedded multi-tenancy solution in scale
- Embedding paginated reports: support for SSO datasources (delayed until later this month)

### Visualisations

- New visuals in AppSource
- Chartulator visual now certified
- Drill Down Donut PRO by ZoomCharts
- graphomate matrix 2021.4
- Strip Plot by Nova Silva
- accoPLANNING by Accobat

### Other

- Quickly create reports from SharePoint document libraries
- Changing the default Power BI Home layout
- Updated slicer defaults for accessibility improvements.

Let's now go through each in turn.

### New Format Pane in Preview: General Availability coming in May

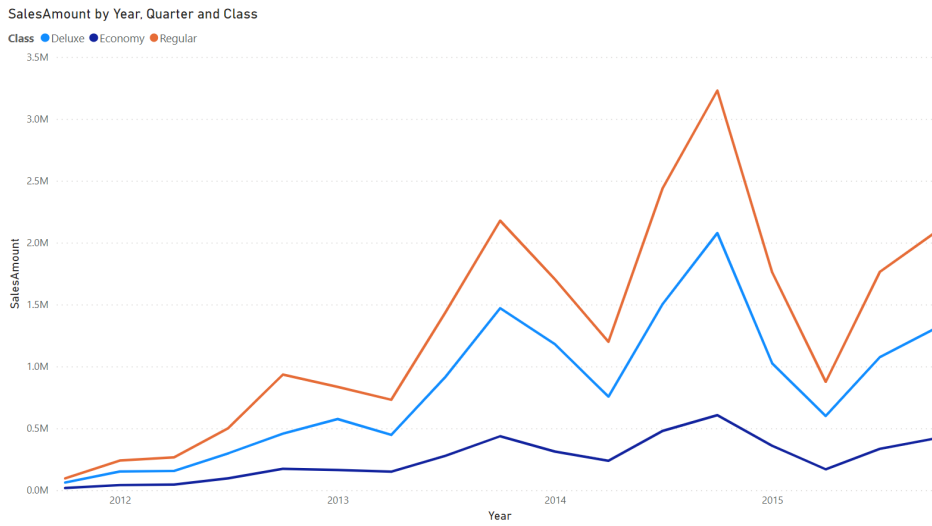
The new Format pane has been in Preview since November and was turned on by default for users in last month's update. Microsoft has announced during this update that they plan to make it Generally Available in May. This will mean users will no longer be able to switch back to the old Format pane.

### Error bars in Preview

You may now add error bars to your line charts to visualise the uncertainty in your data. For forecasting in particular, uncertainty is an important aspect to data, especially in scientific disciplines and other fields where the displayed value of a point needs to be contextualised by the range of its possible values. With this new feature, you can specify upper and lower bounds to the values on your line chart and customise the way the uncertainty is presented within your chart.

To begin, make sure that the error bars Preview feature is enabled by going to **File -> Options -> Preview features** and making sure the error bars checkbox is checked. Also, please make sure you enabled 'New Format pane' feature as this 'Error bars' functionality is available for the new Format pane only. Then, create a Line chart and add a measure for which you have upper and lower bound fields to display.





In the Analytics pane, you will see a new 'Error bars' card. At the top, notice a dropdown to select the measure to which you would like to add error bars. Beneath it, you'll find options to enable error bars for this measure, add upper and lower bound fields, and indicate how the upper and lower bounds relate to the measure. 'Absolute' means that the fields contain the exact value of the upper or lower bound, whilst 'Relative' means that the fields contain the difference between the upper or lower bound and the measure.

For example, if at one point your measure has a value of 60, an upper bound of 90, and a lower bound of 45, your upper bound field in absolute terms would be 90 and your lower bound 45, whilst the upper bound in relative terms would be 30 with the lower bound -15. Choose the option which suits the format of your upper and lower bound fields accordingly.

▼ Error bars

Apply settings to

Measure

SalesAmount ▼

▼ Options

Enabled  On

Upper Bound

Sum of Sal... ▼ ✕

Lower Bound

Sum of Sal... ▼ ✕

Relationship to measure

Absolute ▼

> Bar  On

> Line  Off

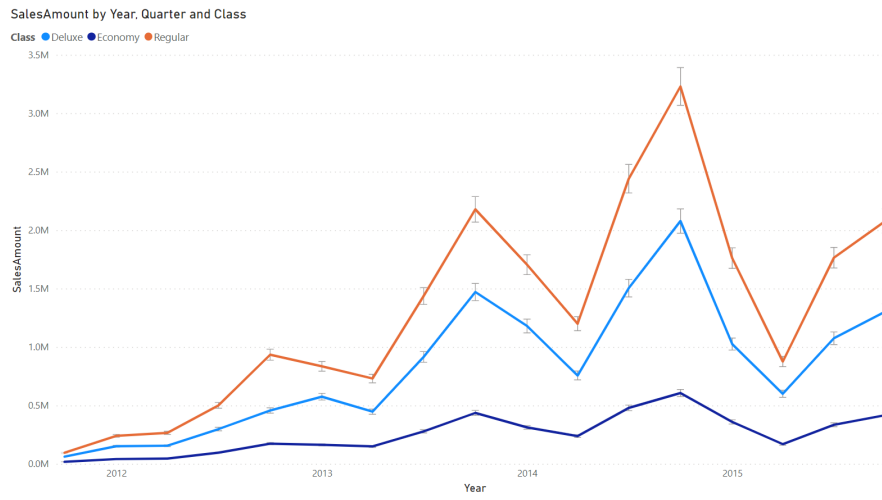
> Shade area  Off

> Markers  Off

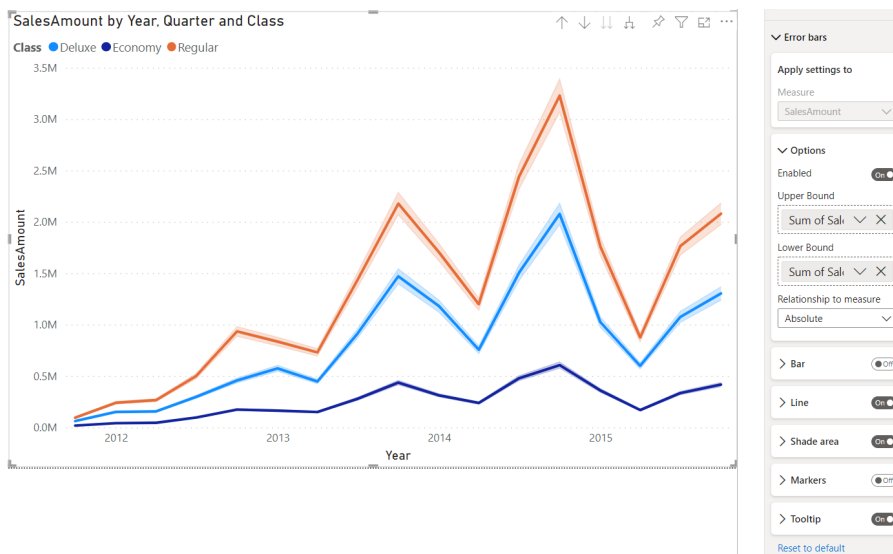
> Tooltip  On

[Reset to default](#)

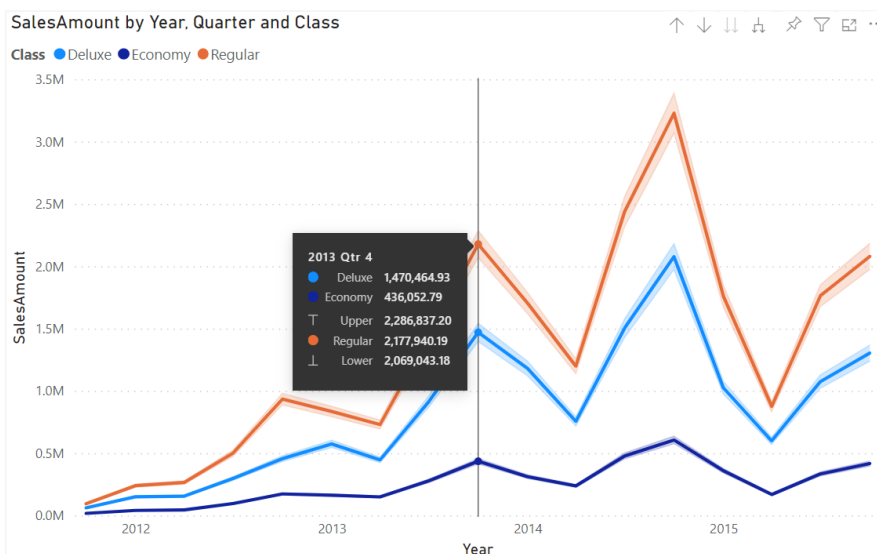
By default, these intervals will be represented by a vertical line with horizontal caps on either end.



You can also represent your uncertainty in other ways, like with lines, shade areas, and markers. Turn them on or off and format them in their associated sub-cards.



Hovering over a point will show the values of the nearby error bars in a Tooltip:

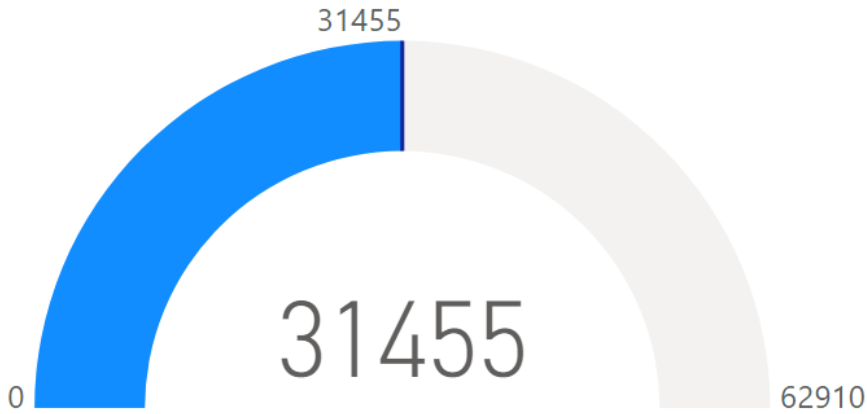


### ***Dynamic format strings now supported for all chart elements***

Reports with AS data sources can provide dynamic format strings, or cell-level formatting, to customise the formatting of their data. Reports with calculation groups will also convert regular format strings into dynamic format strings. Microsoft has admitted they have received feedback that these dynamic format strings were applying inconsistently across a number of different visuals, including the gauge visual and some cases of categorical bar charts.

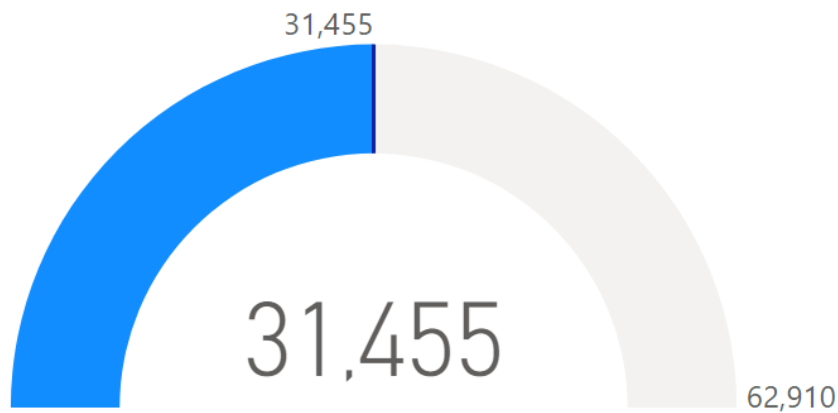
Noting that this inconsistency has affected decisions surrounding report design, this update now supports categorical visuals (hopefully) for all support dynamic format strings. As an example, here is how a gauge visual might display a value string with commas to separate thousands before the changes:

Order Count and Order Count



and this is how it should do it now:

Order Count and Order Count



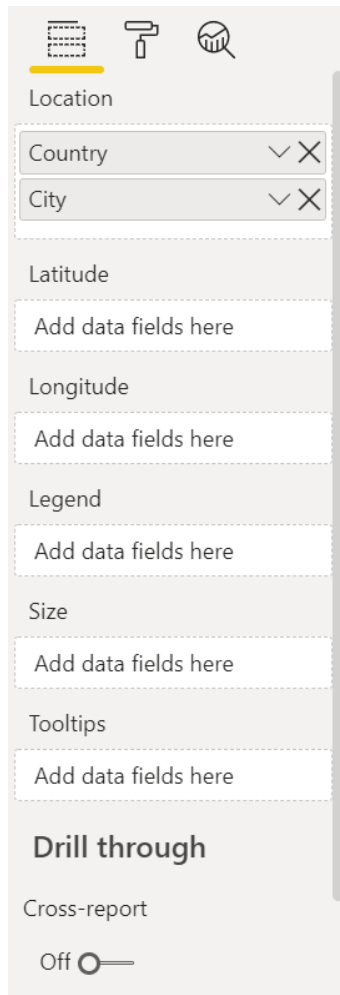
### ***Updates to the Azure Maps visual in Preview***

This month, Power BI has updated the Azure Maps visual with two powerful new capabilities: geocoding and pie chart layers. To try these features out, you'll first need to enable the Azure Map visual in **File -> Options and Settings -> Options -> Preview Features -> Azure map visual**.

#### **GEOCODING**

Most Power BI users work with data that contains geographic information not stored in latitude-longitude format. This data might be the addresses of for a business or even simply the names of the countries it operates in. The Azure maps visual now allows you to supply this more intuitive geographic data directly to Power BI. Just drag your fields into the

Location field well, and Azure Maps will perform the geocoding for you, translating your data into latitude / longitude coordinates. The visual supports geocoding for country or region, state or province, city, county, postal code and address data. As with the Map and Filled Map visuals, you may also drill down on multiple fields in the Location field well too.

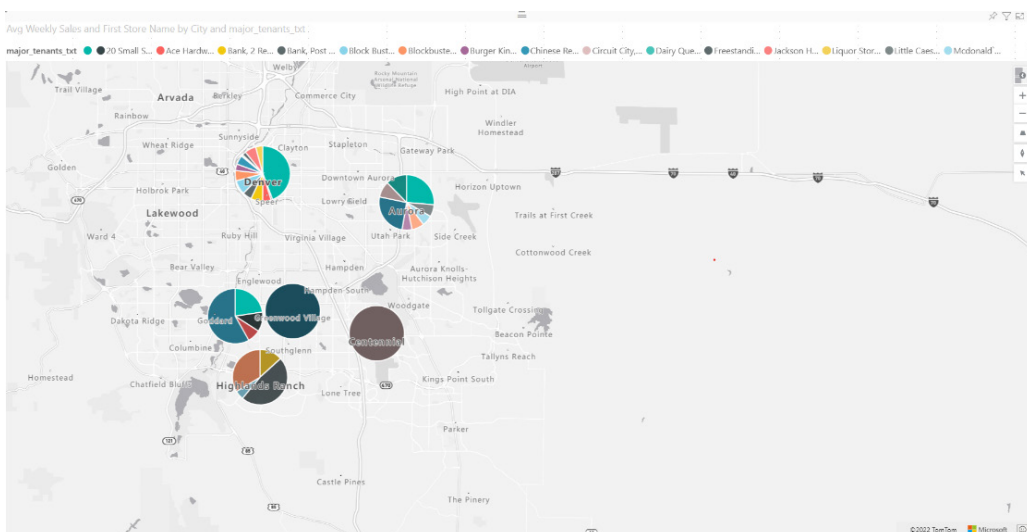


## PIE CHARTS IN AZURE MAPS

This month's update also allows you to turn your bubble layers into Pie charts in order to visualise the proportions by which your data at each location among different categories. This allows you to partition the data in your bubble layers across a new dimension. For example, you can use Pie charts to understand market share of each of your products in a region, with the bubble size representing total market size. You can also

leave the size constant and use Pie charts to demonstrate just regional distribution, such as showing the distribution of education levels across people in each state.

To render such a Pie chart, drag a categorical field into the Legend field well of your map.



## Sensitivity labels update

The 2 GB file-size limitation on saving labelled and protected PBIX files has been removed. PBIX files over 2 GB in size may now be saved with a sensitivity label that carries protection.

## Multi-row card selection

Last month it was announced that you could now select rows in your multi-row card to cross highlight and cross filter other visuals in your report. But, er, oops, it didn't happen. Well the Power BI gang has put it right with this update!

## BitSight Security Ratings (New Connector)

BitSight is a security ratings provider, surfacing rich data on the cybersecurity performance of companies. BitSight for Security Performance Management (SPM) enables Chief Information Security Officers (CISOs) the ability to measure, monitor, manage and report on their cybersecurity program performance over time, and to facilitate a universal understanding of cyber risk across their organisation.

This connector enables organisations to easily pull the BitSight data into

Microsoft Power BI for further analysis and dashboard creation. The dashboarding and analysis generated in Power BI may then be leveraged in executive reporting, tracking remediation progress, combining the BitSight data with other security data sources to gain a more complete view of their cybersecurity program performance over time and help bring universal understanding of cyber risk to stakeholders, as well as other use cases.

## Bloomberg Enterprise Data and Analytics (Connector Update)

The Bloomberg Enterprise Data and Analytics connector has been updated. This release improves the usability of the connector. After upgrading, firm administrators will only need to update their dataset credentials once a year, instead of daily.

## Anaplan (Connector Update)

This updated version of the Anaplan connector for Power BI includes various enhancements. The connector has been updated to support Anaplan data exports up to 5GB in size. This is an increase from 1GB in the previous version of the connector. Users may now also re-run Anaplan exports without a 10 minute delay. The connector has been updated to allow re-run of exports as determined by user.

## FactSet Analytics (Connector Update)

The v1.2.1 New Datastore API now supports 2-Support Security in Component Detail for PA & Vault 3-New Navigator layout. I just wish I knew what that meant...

## AssembleViews (Connector Update)

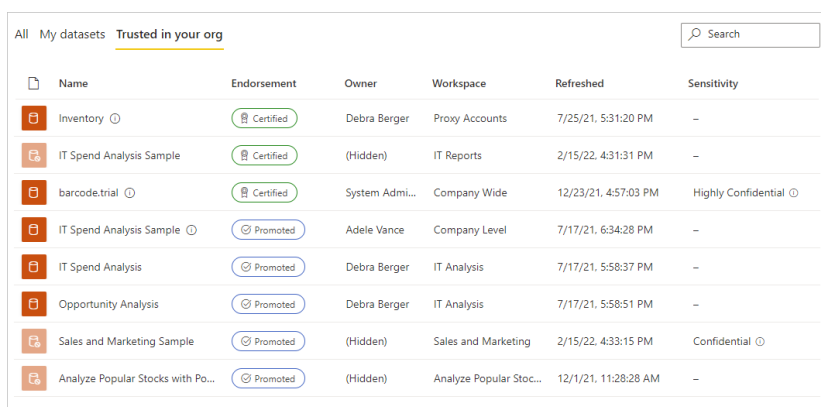
This update now sees the View Date parameter able to accept Date, DateTime, and DateTimeZone types. For Date and DateTime types, the local time zone of the executing machine will be assumed. You should include the query parameter when retrieving project views to reduce load time.

## Datasets hub improvements

The datasets hub (accessible from Power BI's Navigation pane) shows you all the datasets that you have access to, and also any endorsed datasets that have been set as discoverable so you can find them and request access.

The 'All' tab is sorted by the time you last visited the dataset, either by opening the dataset details page or by opening a report that is built on top of the dataset. The 'My' datasets tab includes all the datasets that you are the owner of.

As the starting place for data discovery, one of the main purposes of the datasets hub is to help users find high quality data and reuse it for their own needs. With this in mind, Microsoft has made some changes to the datasets hub tabs. Now there is a new Trusted in your org tab that focuses exclusively on your organisation's trusted datasets: these are the endorsed datasets. When you select the Trusted in your organisation tab, you'll see a list of all the endorsed datasets in your organisation. The certified datasets are listed first, followed by the promoted datasets:

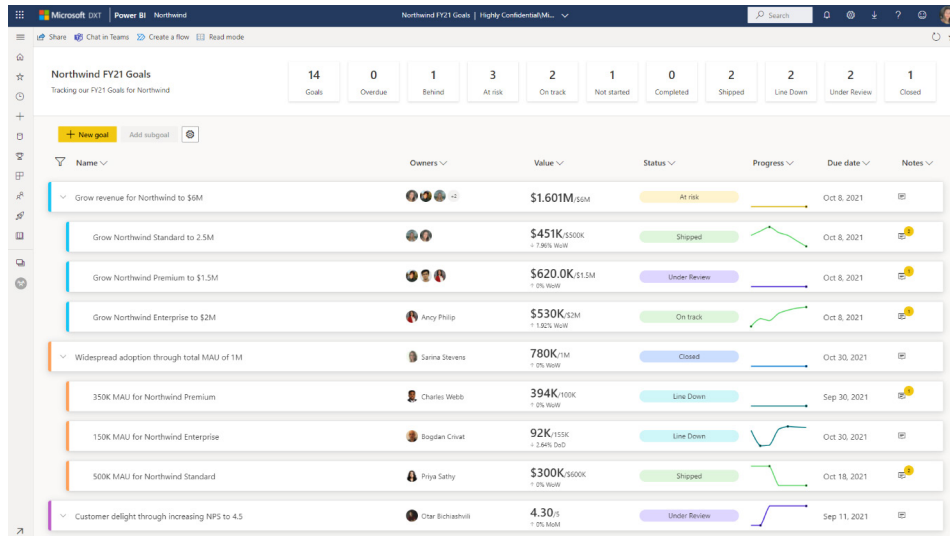


Name	Endorsement	Owner	Workspace	Refreshed	Sensitivity
Inventory	Certified	Debra Berger	Proxy Accounts	7/25/21, 5:31:20 PM	-
IT Spend Analysis Sample	Certified	(Hidden)	IT Reports	2/15/22, 4:31:31 PM	-
barcode.trial	Certified	System Admi...	Company Wide	12/23/21, 4:57:03 PM	Highly Confidential
IT Spend Analysis Sample	Promoted	Adele Vance	Company Level	7/17/21, 6:34:28 PM	-
IT Spend Analysis	Promoted	Debra Berger	IT Analysis	7/17/21, 5:58:37 PM	-
Opportunity Analysis	Promoted	Debra Berger	IT Analysis	7/17/21, 5:58:51 PM	-
Sales and Marketing Sample	Promoted	(Hidden)	Sales and Marketing	2/15/22, 4:33:15 PM	Confidential
Analyze Popular Stocks with Po...	Promoted	(Hidden)	Analyze Popular Stoc...	12/1/21, 11:28:28 AM	-

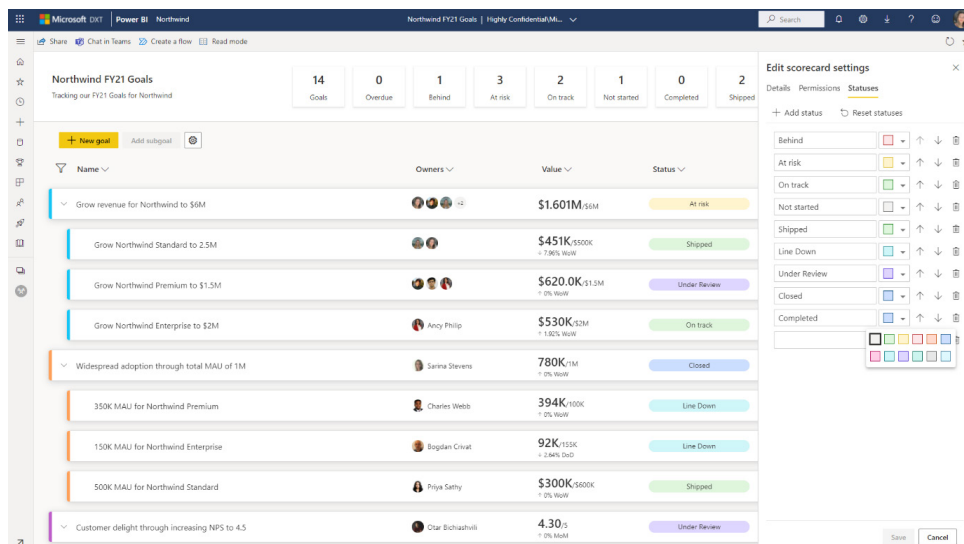
## Power BI Goals enhancements

### CUSTOM STATUSES IN SCORECARDS

Statuses are an integral part of goal tracking, and with this update, you can now customise your statuses so they're set appropriately for your organisation.



In Edit mode, simply click the settings gear, and enter the section called statuses to start modifying your scorecard statuses. Here, you may create new statuses, rename existing ones, and customise the colour(s).



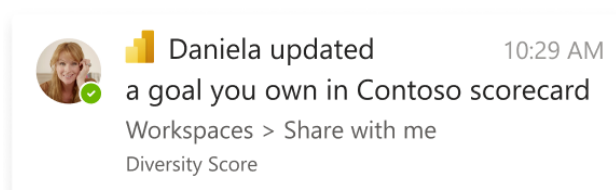
With this update, you may customise the statuses to the terminology you use every day in your organisation, ensuring everyone in your team easily understands. This makes it easier for Goals to work seamlessly throughout your business.

### POWER BI GOALS TEAMS NOTIFICATION

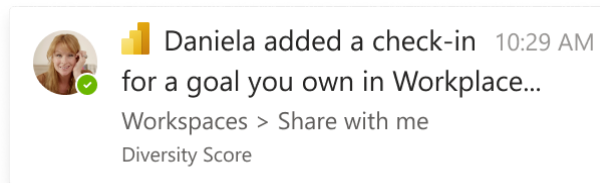
Last month, Microsoft released Teams notifications to help you stay up to date about the goals you're responsible for. This month, this latest Power BI update sees the release of a few more notifications for goal and check-in updates.

Goal update notifications are sent when a goal is updated. The goal owners receive a Teams notification from the user updating the goal

if they have the Power BI app for Teams installed. They will get a notification in the activity feed and as a banner (toast card). When they click the notification, the scorecard opens and the Details pane for the goal is shown. This notification will be sent on any of the manual updates to the goal such as changes to goal name, start / due date, adding or removing owners, adding or changing status rules.



The owners of the goal receive a goal check-in activity feed notification when a user adds a new check-in or edits an existing check-in. This notification will be sent when a value or status is either added or edited directly on the goal or through the check-in experience in the Details pane.



You should note that the recipient needs to have the Power BI app for Microsoft Teams installed and have access to the corresponding scorecards to get these notifications.

***New improved method for deploying Power BI Embedded multi-tenancy solution in scale***

Until now, ISVs and other Power BI Embedded application owners were able to deploy the embedded solution to production by using one of the following two [2] options:

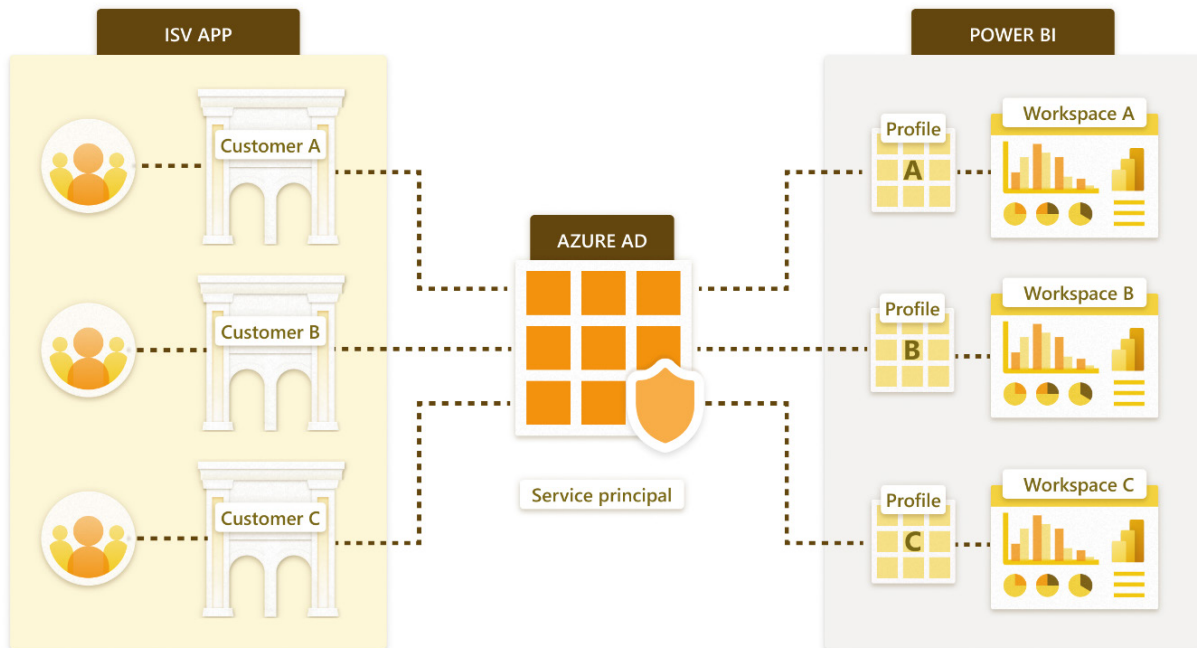
1. Create one service principal for all their customer tenants, where they have a Power BI limit of 1,000 workspaces per each service principal
2. Create one service principal per each customer tenant, where they have the burden of managing thousands of service principals.

Now, Power BI Embedded Analytics has introduced a new method that allows a much larger number of customer tenants per one service principal by introducing a service principal "child" named service principal

profile. Each service principal profile can administer a customer tenant, and the number of profiles per service principal can reach hundreds of thousands.

Service principal profiles allow the independent software vendor (ISV) to build a scalable application that enables better customer data isolation and establishes tighter security boundaries between customers.

Using service principal profiles enables the ISV application to host multiple customers on a single Power BI tenant. Each profile represents one customer in Power BI, i.e. each profile creates and manages datasets, reports, dashboards and other artifacts that are connected only to one specific customer's data.



***Embedding paginated reports: support for SSO datasources (delayed until later this month)***

With Power BI Embedded Analytics, ISVs and customers may also now create Power BI content that displays paginated reports in a fully integrated and interactive application. They can embed paginated reports using the solution that works best for them, embed for your customers or embed for your organisation.

Until now, data sources that require single sign-on (SSO) were not supported, but support has now been added for SSO data sources, including Power BI datasets having SSO data sources.

## New visuals in AppSource

The following are new visuals this update:

- Area chart with custom ToolTip
- Bar chart colour formatting based on legend
- Bar chart with colour formatting
- Bar chart with custom ToolTip
- Bubble chart with categorical x and y Axes
- Bullet chart with custom label placement
- Cluster bar chart side by side
- Column chart with dual y axis
- Column chart with two [2] x axes
- Custom ToolTip for Line and Clustered Column chart
- Dual Axis Line chart with legends
- Event viewer
- Filter by List
- Multiple Vertical Line chart
- Pie chart with full legends label
- Power Slicer
- Side By side Bar chart
- Stacked Column chart With custom legend placement
- Stacked Bar chart with Data label and Data percentage.

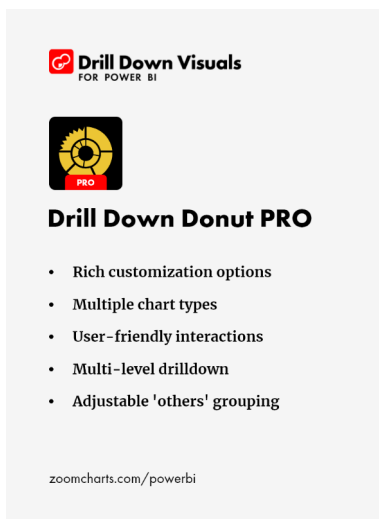
## Charticulator visual now certified

As of version 1.4.1, the Charticulator custom visual is now certified. With this update, there are a few changes and bugfixes:

- fixed an issue with tick formatting in axes with the Date data type
- fixed an issue with text mapping for columns with the Date data type
- added support for number of ticks in numerical axes
- added a drop zone for tick data for numerical axes
- fixed an issue with the 'Sort by another column' panel
- added auto-updating for values in gradient scales.

## Drill Down Donut PRO by ZoomCharts

Drill Down Donut PRO was designed to allow all interactions to take place on the chart itself without hidden controls, thus ensuring a quick and easy data exploration experience. Cross-chart filtering is aided by smooth animations, and every aspect of the chart may be customised.



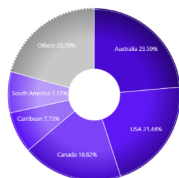
**Drill Down Visuals**  
FOR POWER BI

**Drill Down Donut PRO**

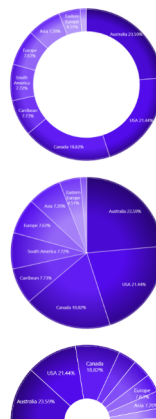
- Rich customization options
- Multiple chart types
- User-friendly interactions
- Multi-level drilldown
- Adjustable 'others' grouping

zoomcharts.com/powerbi

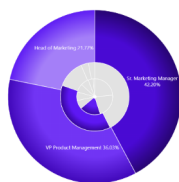
Adjustable 'others' grouping



Multiple chart types



Multi-level drilldown



Main features include:

- adjustable 'Others' slice: set the number of visible slices and group the rest as you see fit
- on-chart interactions
- custom ToolTips: you may select ZoomCharts custom ToolTip or a Power BI built-in ToolTip
- full customisation: modify every slice, label and legend
- Desktop and mobile device navigation: explore charts the same way on any device.

Drill Down Donut PRO may be downloaded from AppSource.

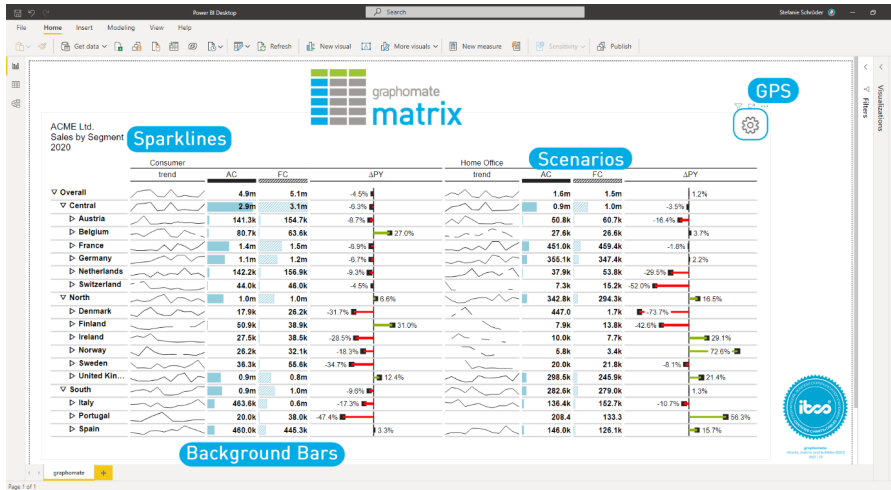


## graphomate matrix 2021.4

The graphomate matrix may be used to create tables within your reports, with the ability to design the layout. In addition, the graphomate matrix enables both hierarchical and tabular (crosstab) views. Hierarchies may also be used in the columns, and additional columns with their own calculations may be added too.

Examples:

- apply scenarios in column headers
- create different types of charts in the cells (Bar charts, Deviation charts, Sparklines)
- align your table design according to IBCS (Intel Binary Compatibility Standard)
- set the number formatting for each cell
- Quick Access to the graphomate property sheet (GPS) via the gear wheel.

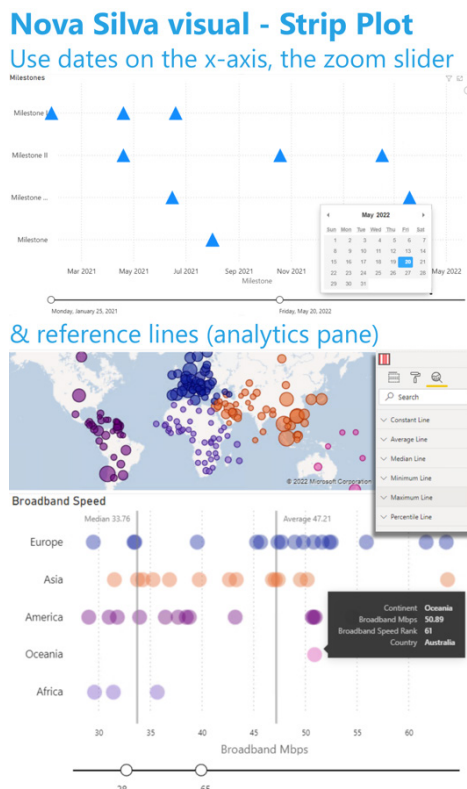


With the new release version sees efficiencies made and Sparklines introduced. Furthermore, people may process patterns more easily than pure text and numerical values. Therefore, Sparklines in cells are a good way to provide information about the temporal behaviour of a Key Performance Indicator (KPI).

The graphomate matrix may be downloaded from the AppSource.

## Strip Plot by Nova Silva

Most charts will force you to summarise or categorise data before it is displayed. This can hide important details and may be misleading. The Strip Plot shows all your data observations in one go without hiding important details. It shows each data point on a single continuous scale.



This update allows you to use dates on the x axis, which has also been enhanced with the optional Zoom-Slider.

Each visual in Power BI has three different panes: Fields, Format and Analytics. The Analytics pane allows you to add reference lines to your visual (e.g. fixed value, median, average, max). This update allows you to use this functionality with the Strip Plot.

All functionality of the Strip Plot is available through the standard Power BI interface: there is need to learn any new interface.

Again, this may be downloaded from the AppSource.

### accoPLANNING by Accobat

This allows you to writeback for any Power BI model and use Power BI for planning and forecast.

With this visual you may now writeback numbers, text and dates on any existing or new Power BI model. There are also new advanced features, such as splashing numbers from an aggregated level to all leaf levels and even splash the numbers relative according to another measure in your model.

Key new functionalities:

#### Transaction keys

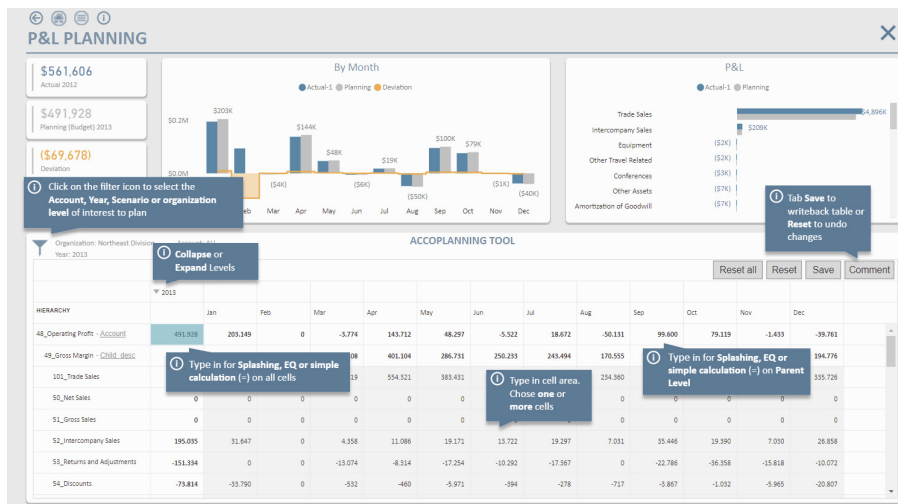
- using transaction keys are now a possibility to get a better coherent and valid data model
- this means you can have a business orientated description in the grid and slides but the writeback table are fill out with the corresponding key

#### Calculate further on existing numbers

- you may now add calculate on top of existing numbers or other measures
- this is useful when you wish to vary a potential increase on actual numbers as basis for you forecast

#### Read only cells

- you can now lock cell for typing and splashing
- this is useful when you want to lock part of a dataset or certain cells for changing the numbers.

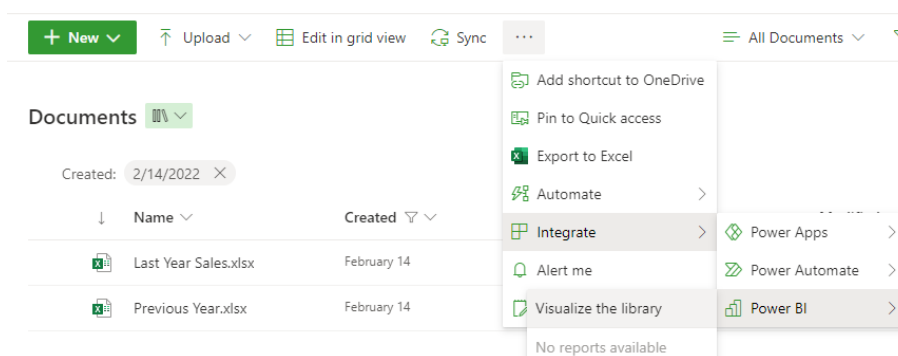


With the accoPLANNING visual, you may combine the planning and reporting processes in Power BI. The accoPLANNING visual may be downloaded from the AppSource.

### Quickly create reports from SharePoint document libraries

Previously, Microsoft launched an integration in SharePoint lists that, with just a few clicks, allows you to automatically create a Power BI report off list data. Now, that same capability is available on SharePoint document

libraries. This will let you easily explore your file metadata, such as how frequently files are modified, who makes the most modifications, etc.



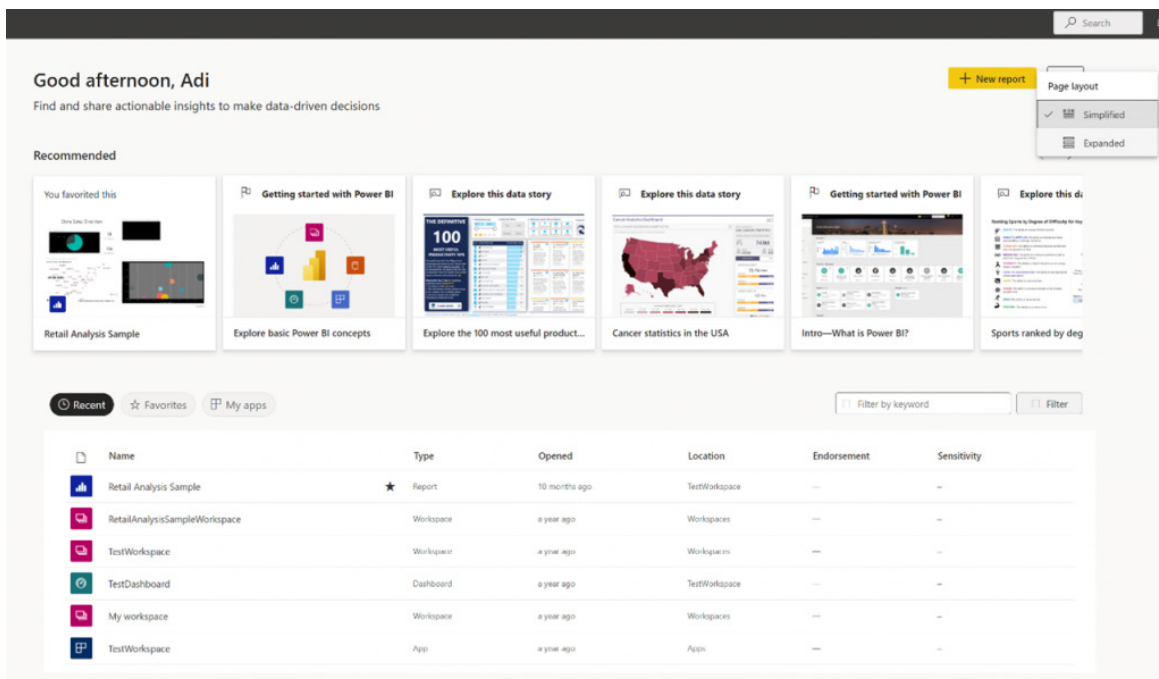
Currently, this option is only available at the parent, or root, folder of the document libraries and will not pull metadata for files within folders. However, this restriction will be removed in a future update (watch this space!).



### Changing the default Power BI Home layout

Recently Power BI introduced a simplified layout for Power BI Home to make discovery of relevant content easier. This new layout introduced a Recommended widget at the top of the page and a list with several pivots below it. Power BI will display personalised content recommendations in the widget as well as frequently consumed and favorited content. Only content that a user has permissions to access will be displayed.

In this update, Microsoft has switched all Power BI service users to have the new simplified layout by default. The previous layout (aka Expanded layout) will still be accessible via the layout switcher on Home. A user's selected layout will persist for subsequent visits to Home.



### Updated slicer defaults for accessibility improvements

The new base theme has been updated for slicers to enhance overall accessibility. This will affect all newly created reports for some slicers, including List and Date slicers. List slicers will have slightly larger font and spacing between the items to accommodate touch, and generally make selection easier. Default colours are also updated to have slightly higher contrast, for List and Horizontal slicer types. Date pickers will now

have a calendar icon, and input labels by default. The new defaults are designed to meet standard accessibility requirements, so to best create accessible content it is advised to keep the default settings. These items may also be modified in the new Formatting pane or controlled via a custom theme if you want your report defaults to be different.

Example JSON snippet of custom theme file:

```
"slicer": {
  "*": {
    "date": [
      {
        "hideDatePickerButton": false
      }
    ],
    "items": [
      {
        "fontColor": {
          "solid": {
            "color": "#252423"
          }
        },
        "padding": 4,
        "accessibilityContrastProperties": true
      }
    ]
  }
},
```

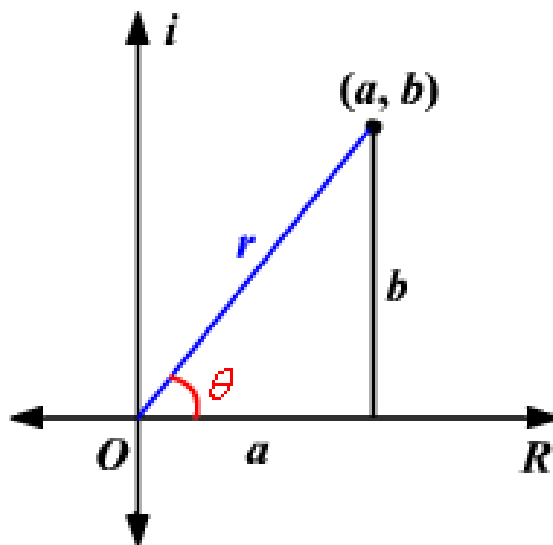
More of the same next month, we're sure!!

## The A to Z of Excel Functions: IMCOS

An imaginary number is a complex number that can be written as a real number multiplied by the imaginary unit  $i$  (sometimes denoted  $j$ ) which is defined by its property  $i^2 = -1$ . In general, the square of an imaginary number  $bi$  is  $-b^2$ . For example,  $9i$  is an imaginary number, and its square is  $-81$ . Zero is considered to be both real and imaginary.

An **imaginary** number  $bi$  can be added to a **real** number  $a$  to form a **complex number** of the form  $a + bi$ , where the real numbers  $a$  and  $b$  are called, respectively, the **real part** and the **imaginary part** of the **complex number**.

The **polar form** of a complex number is another way to represent the number. The form  $z = a + bi$  is called the **rectangular form** of a complex number.



The horizontal axis is the real axis and the vertical axis is the imaginary axis. You can find the real and imaginary components in terms of  $r$  and  $\theta$ , where  $r$  is the length of the vector and  $\theta$  is the angle made with the real axis.

From the Pythagorean Theorem,

$$r^2 = a^2 + b^2$$

By using the basic trigonometric ratios,

$$\cos \theta = a / r \text{ and } \sin \theta = b / r$$

Therefore, multiplying each side by r:

$$r \cos \theta = a \text{ and } r \sin \theta = b$$

Therefore,

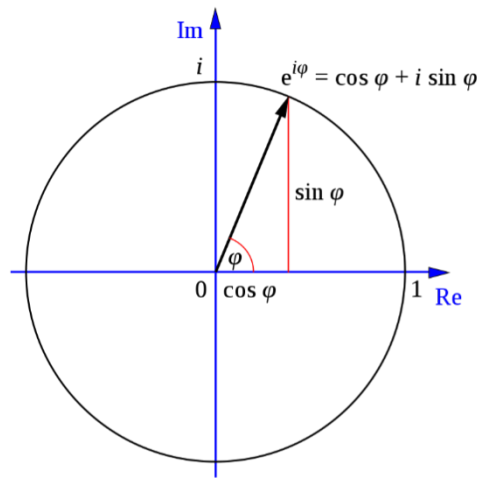
$$z = a + bi$$

$$\Leftrightarrow z = r \cos \theta + (r \sin \theta)i$$

$$\Leftrightarrow z = r(\cos \theta + i \sin \theta)$$

In the case of a complex number,  $r$  represents the **absolute value**, or **modulus** (where  $r = |z| = \sqrt{a^2 + b^2}$ ), and the angle  $\theta$  is called the **argument** of the complex number ( $\theta = \tan^{-1}(\frac{b}{a})$  for  $a > 0$  and  $\theta = \tan^{-1}(\frac{b}{a}) + \pi$  for  $a < 0$ ).

Using **Euler's Formula**,



$$e^{i\theta} = \cos \theta + i \sin \theta$$

Given  $z = \frac{e^{iz} + e^{-iz}}{2}$ , by doing more mathematics than you would probably ever wish to read,

$$\begin{aligned} \cos a \cosh b - i \sin a \sinh b &= \frac{e^{ia} + e^{-ia}}{2} \frac{e^b + e^{-b}}{2} - i \frac{e^{ia} - e^{-ia}}{2i} \frac{e^b - e^{-b}}{2} \\ &= \frac{e^{b+ia} + e^{-b+ia} + e^{b-ia} + e^{-b-ia} - e^{b+ia} + e^{-b+ia} + e^{b-ia} - e^{-b-ia}}{4} \\ &= \frac{e^{-b+ia} + e^{b-ia}}{2} \\ &= \frac{e^{i(a+bi)} + e^{-i(a+bi)}}{2} \\ &= \cos(a + bi) \end{aligned}$$

you eventually get:

$$\cos(x + yi) = \cos(x) \cosh(y) - \sin(x) \sinh(y)i$$

The **IMCOS** function returns the cosine of a complex number in **x + yi** or **x + yj** text format.

The **IMCOS** function employs the following syntax to operate:

**IMCOS(number)**

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

- **number**: this is required and represents the complex number for which you want to calculate the cosine.

It should be further noted that:

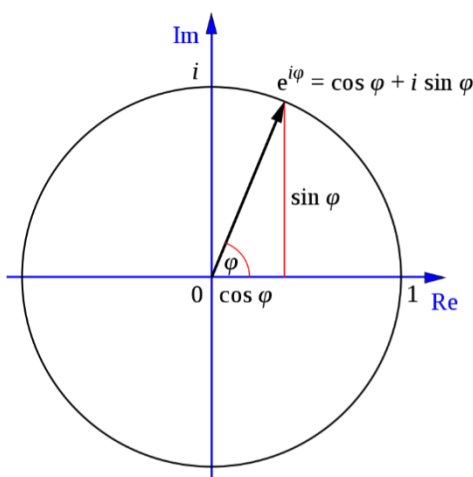
- you should use **COMPLEX** to convert real and imaginary coefficients into a complex number
- **IMCOS** recognises either the **i** or **j** notation
- if **number** is a value that is not in the **x + yi** or **x + yj** text format, **IMCOS** returns the **#NUM!** error value
- if **number** is a logical value, **IMCOS** returns the **#VALUE!** error value
- if the complex number ends in **+i** or **-i** (or **j**), *i.e.* there is no coefficient between the operator and the imaginary unit, there must be no space, otherwise **IMCOS** will return an **#NUM!** error.

Please see our example below:

	A	B	C
1	Formula	Description	Result
2	<b>=IMCOS("3+4i")</b>	Complex cosine of the complex number 3 + 4i	<b>-27.0349456030742-3.85115333481178i</b>

## The A to Z of Excel Functions: IMCOSH

Returning to Euler's Formula,



$$e^{i\theta} = \cos \theta + i \sin \theta$$

Given  $\cos z = \frac{e^{iz} + e^{-iz}}{2}$ , then  $\cosh z = \frac{e^z + e^{-z}}{2}$ . It then follows

$$\begin{aligned} \cosh a \cos b - i \sinh a \sin b &= \frac{e^a + e^{-a}}{2} \frac{e^{ib} + e^{-ib}}{2} + i \frac{e^a - e^{-a}}{2i} \frac{e^{ib} - e^{-ib}}{2} \\ &= \frac{e^{a+ib} + e^{-a+ib} + e^{a-ib} + e^{-a-ib} + e^{a+ib} - e^{-a+ib} - e^{a-ib} + e^{-a-ib}}{4} \\ &= \frac{e^{a+ib} + e^{-(a+ib)}}{2} \\ &= \cosh(a + bi) \end{aligned}$$

you eventually get:

$$\cosh(x + yi) = \cosh x \cos y - i \sinh x \sin y$$

The **IMCOSH** function returns the hyperbolic cosine of a complex number in  $x + yi$  or  $x + yj$  text format.

The **IMCOSH** function employs the following syntax to operate:

**IMCOSH(number)**

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

- **number**: this is required and represents the complex number for which you want to calculate the hyperbolic cosine.

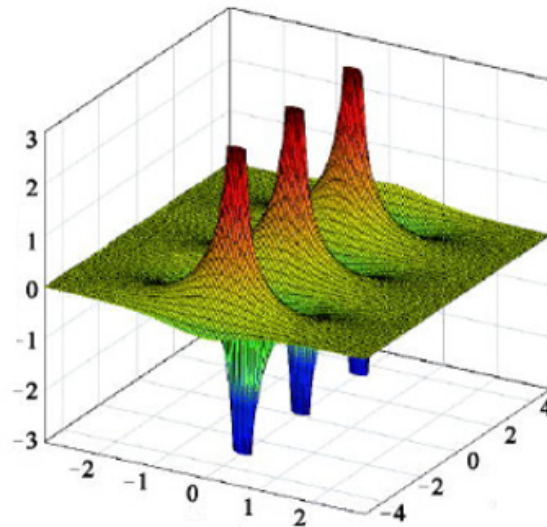
It should be further noted that:

- you should use **COMPLEX** to convert real and imaginary coefficients into a complex number
- **IMCOSH** recognises either the **i** or **j** notation
- if **number** is a value that is not in the  $x + yi$  or  $x + yj$  text format, **IMCOSH** returns the **#NUM!** error value
- if **number** is a logical value, **IMCOSH** returns the **#VALUE!** error value
- if the complex number ends in  $+i$  or  $-i$  (or  $j$ ), *i.e.* there is no coefficient between the operator and the imaginary unit, there must be no space, otherwise **IMCOSH** will return an **#NUM!** error.

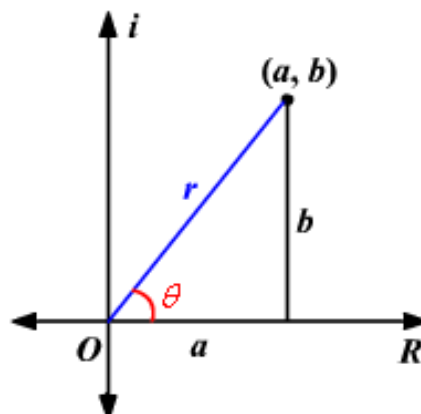
Please see another example below:

	A	B	C
1	Formula	Description	Result
2	<code>=IMCOSH("3+4i")</code>	Complex hyperbolic cosine of the complex number 3 + 4i	-6.58066304055116-7.58155274274654i

## The A to Z of Excel Functions: IMCOT



The **polar form** of a complex number is another way to represent the number. The form  $z = a + bi$  is called the **rectangular form** of a complex number.



The horizontal axis is the real axis and the vertical axis is the imaginary axis. You can find the real and imaginary components in terms of  $r$  and  $\theta$ , where  $r$  is the length of the vector and  $\theta$  is the angle made with the real axis.

From the Pythagorean Theorem,

$$r^2 = a^2 + b^2$$

By using the basic trigonometric ratios,

$$\cos \theta = a / r \text{ and } \sin \theta = b / r$$

Therefore, multiplying each side by  $r$ :

$$r \cos \theta = a \text{ and } r \sin \theta = b$$

Therefore,

$$z = a + bi$$

$$\Leftrightarrow z = r \cos \theta + (r \sin \theta)i$$

$$\Leftrightarrow z = r(\cos \theta + i \sin \theta)$$

In the case of a complex number,  $r$  represents the **absolute value**, or **modulus** (where  $r = |z| = \sqrt{a^2 + b^2}$ ), and the angle  $\theta$  is called the **argument** of the complex number ( $\theta = \tan^{-1}(\frac{b}{a})$  for  $a > 0$  and  $\theta = \tan^{-1}(\frac{b}{a}) + \pi$  for  $a < 0$ ).

Since the cotangent is the reciprocal of the tangent, the cotangent is equal to the adjacent side divided by the length of the opposite side for  $\theta$ . Ignoring all the required hefty mathematics,

$$\cot(a + bi) = \frac{\cos a \cosh b - i \sin a \sinh b}{\sin a \cosh b + i \cos a \sinh b}$$

The **IMCOT** function returns the cotangent of a complex number in  $x + yi$  or  $x + yj$  text format.

The **IMCOT** function employs the following syntax to operate:

$$\text{IMCOT}(\text{inumber})$$

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

- **inumber**: this is required and represents the complex number for which you want to calculate the cotangent.

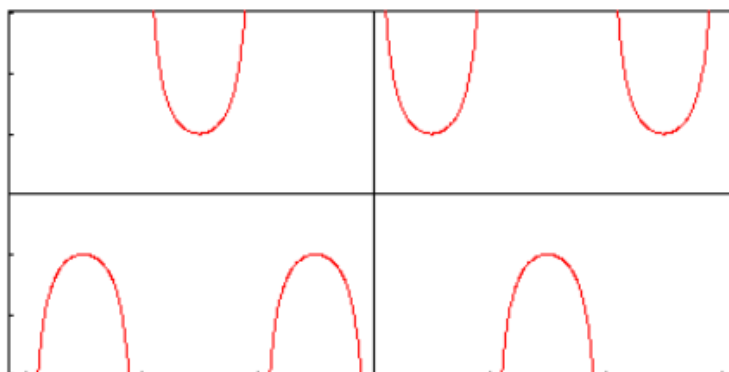
It should be further noted that:

- you should use **COMPLEX** to convert real and imaginary coefficients into a complex number
- **IMCOT** recognises either the  $i$  or  $j$  notation
- if **inumber** is a value that is not in the  $x + yi$  or  $x + yj$  text format, **IMCOT** returns the **#NUM!** error value
- if **inumber** is a logical value, **IMCOT** returns the **#VALUE!** error value
- if the complex number ends in  $+i$  or  $-i$  (or  $j$ ), *i.e.* there is no coefficient between the operator and the imaginary unit, there must be no space, otherwise **IMCOT** will return an **#NUM!** error.

Please see yet another example below:

	A	B	C
1	Formula	Description	Result
2	=IMCOT("3+4i")	Cotangent of the complex number 3 + 4i	-0.000187587737983659-1.00064439247156i

## The A to Z of Excel Functions: IMCSC





From above,

$$z = r(\cos \theta + i \sin \theta)$$

In the case of a complex number,  $r$  represents the **absolute value**, or **modulus** (where  $r = |z| = \sqrt{a^2 + b^2}$ ), and the angle  $\theta$  is called the **argument** of the complex number ( $\theta = \tan^{-1}\left(\frac{b}{a}\right)$  for  $a > 0$  and  $\theta = \tan^{-1}\left(\frac{b}{a}\right) + \pi$  for  $a < 0$ ).

The cosecant is simply the reciprocal of the sine function. The **IMCSC** function returns the cosecant of a complex number in  $x + yi$  or  $x + yj$  text format.

The **IMCSC** function employs the following syntax to operate:

$$\text{IMCSC}(\text{inumber})$$

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

- **inumber**: this is required and represents the complex number for which you want to calculate the cosecant.

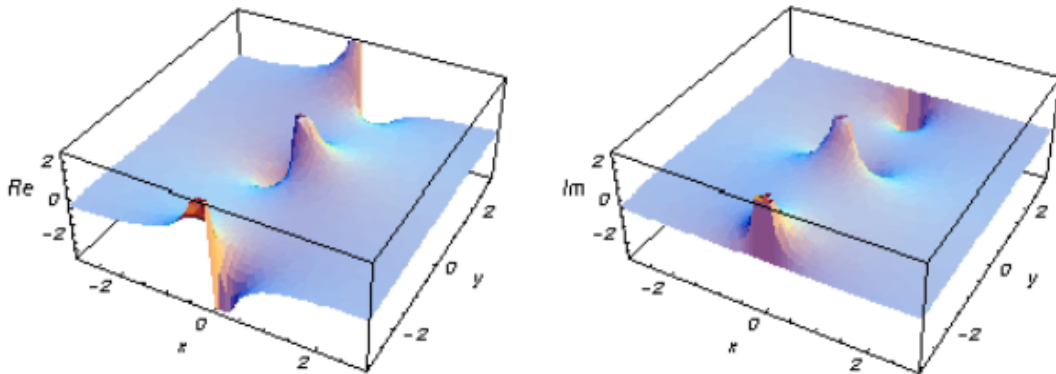
It should be further noted that:

- you should use **COMPLEX** to convert real and imaginary coefficients into a complex number
- **IMCSC** recognises either the  $i$  or  $j$  notation
- if **inumber** is a value that is not in the  $x + yi$  or  $x + yj$  text format, **IMCSC** returns the **#NUM!** error value
- if **inumber** is a logical value, **IMCSC** returns the **#VALUE!** error value
- if the complex number ends in  $+i$  or  $-i$  (or  $j$ ), *i.e.* there is no coefficient between the operator and the imaginary unit, there must be no space, otherwise **IMCSC** will return an **#NUM!** error.

Please see our penultimate example below:

	A	B	C
1	Formula	Description	Result
2	=IMCSC("3 +4i")	Cosecant of the complex number 3 +4i	0.0051744731840194+0.036275889628626i

## The A to Z of Excel Functions: IMCSCH



As a reminder (!),

$$z = r(\cos \theta + i \sin \theta)$$

In the case of a complex number,  $r$  represents the **absolute value**, or **modulus** (where  $r = |z| = \sqrt{a^2 + b^2}$ ), and the angle  $\theta$  is called the **argument** of the complex number ( $\theta = \tan^{-1}\left(\frac{b}{a}\right)$  for  $a > 0$  and  $\theta = \tan^{-1}\left(\frac{b}{a}\right) + \pi$  for  $a < 0$ ).

The hyperbolic cosecant (the reciprocal of the hyperbolic sine function) is defined by the following relationships:

$$\begin{aligned} i \operatorname{csch} z &= \frac{i}{\sinh z} \\ &= \frac{-1}{i \sinh z} \\ &= \frac{-1}{\sin(iz)} \\ &= -\operatorname{csc}(iz) \end{aligned}$$

The **IMCSCH** function returns the hyperbolic cosecant of a complex number in **x + yi** or **x + yj** text format.

The **IMCSCH** function employs the following syntax to operate:

**IMCSCH(inumber)**

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

- **inumber**: this is required and represents the complex number for which you want to calculate the hyperbolic cosecant.

It should be further noted that:

- you should use **COMPLEX** to convert real and imaginary coefficients into a complex number
- **IMCSCH** recognises either the **i** or **j** notation
- if **inumber** is a value that is not in the **x + yi** or **x + yj** text format, **IMCSCH** returns the **#NUM!** error value
- if **inumber** is a logical value, **IMCSCH** returns the **#VALUE!** error value
- if the complex number ends in **+i** or **-i** (or **j**), *i.e.* there is no coefficient between the operator and the imaginary unit, there must be no space, otherwise **IMCSCH** will return an **#NUM!** error.

Please see our final example for this month below:

	A	B	C
1	Formula	Description	Result
2	<b>=IMCSCH("3 + 4i")</b>	Hyperbolic cosecant of the complex number 3 + 4i	<b>-0.0648774713706355+0.0754898329158637i</b>

More Excel Functions next month.

## Beat the Boredom Suggested Solution

Let's recap our problem from earlier in the newsletter.

	A	B	C	D	E
3					
4		Business Unit	Quarter	Sales	
5		A	1	10	
6		B	1	20	
7				30	
8		C	2	40	
9		B		50	
10			3	60	
11					

This month's challenge was about creating a single formula to sum the sales amounts based on the values in the Business Unit and Quarter columns that is also able to use blank values as a criterion.

Let's try using the **SUMIFS** function:

**=SUMIFS(D5:D10,B5:B10,H5,C5:C10,H6)**

	A	B	C	D	E	F	G	H
3								
4		Business Unit	Quarter	Sales				
5		A	1	10			Business Unit	B
6		B	1	20			Quarter	
7				30			Total Sales	0
8		C	2	40				
9		B		50				
10			3	60				
11								

The **SUMIFS** function is unable to pick up blank values as a criterion. With the current input we would expect a total sales value of 50. So how do we get **SUMIF** to recognise blank values? We can use quotation marks (""") as the criterion:

**=SUMIFS(D5:D10,B5:B10,H5,C5:C10,"")**

	A	B	C	D	E	F	G	H
3								
4		Business Unit	Quarter	Sales				
5		A	1	10			Business Unit	B
6		B	1	20			Quarter	
7				30			Total Sales	50
8		C	2	40				
9		B		50				
10			3	60				
11								

Now we have to consider the different input combinations we can have. For instance, a blank value in Business Unit and '3' in Quarter. Hard coding for all the criteria will not work for all combinations. We have to incorporate the **IF** function into our formula to allow for all possible input combinations:

**=IF(AND(H5="",H6=""),SUMIFS(D5:D10,B5:B10,"",C5:C10,""),IF(H5="",SUMIFS(D5:D10,B5:B10,"",C5:C10,H6),IF(H6="",SUMIFS(D5:D10,B5:B10,H5,C5:C10,""),SUMIFS(D5:D10,B5:B10,H5,C5:C10,H6))))**

The screenshot shows a spreadsheet with a data table and a summary table. The data table has columns for Business Unit, Quarter, and Sales. The summary table has rows for Business Unit, Quarter, and Total Sales. The formula in cell H7 is a complex nested IF statement that handles various combinations of blank and non-blank values in the input cells.

This solution is very long and contains three [3] nested **IF** statements, which not only greatly increases the chance of an error but is confusing to read. Furthermore, what happens when we add another column? We would need to have seven [7] nested **IF** statements for the formula to work.

Clearly, this is not the right path to take.

### Suggested Solution

We can use the ampersand, '&', operator to concatenate the blank value and the input cell into the criteria:

**=SUMIFS(D5:D10,B5:B10,""&H5,C5:C10,""&H6)**

The screenshot shows the same data table as before, but the summary table now shows Business Unit B, Quarter 1, and Total Sales 50. The formula in cell H7 is simplified to use the ampersand operator to concatenate the blank value and the input cell into the criteria.

In this case, incorporating the '&' operator into the **SUMIFS** function allows us to use blank values as a criterion and allows for easy scaling if we wish to include even more columns:

**=SUMIFS(E5:E10,B5:B10,""&I5,C5:C10,""&I6,D5:D10,""&I7)**

The screenshot shows a new data table with columns for Business Unit, Quarter, Staff, and Sales. The summary table now shows Business Unit B, Quarter 1, Staff 5, and Total Sales 50. The formula in cell I8 uses the ampersand operator to concatenate the blank value and the input cell into the criteria for multiple columns.

There you have it: our solution to use blanks as a criterion with the option to use multiple criterion.

Until next time.

## Upcoming SumProduct Training Courses - COVID-19 update

Due to the COVID-19 pandemic that is currently spreading around the globe, we are suspending our in-person courses until further notice. However, to accommodate the new working-from-home dynamic, we are switching our public and in-house courses to an online delivery stream, presented via Microsoft Teams, with a live presenter running through the same course material, downloadable workbooks to complete the hands-on exercises during the training session, and a recording of the sessions for

your use within 1 month for you to refer back to in the event of technical difficulties. To assist with the pacing and flow of the course, we will also have a moderator who will help answer questions during the course.

If you're still not sure how this will work, please contact us at [training@sumproduct.com](mailto:training@sumproduct.com) and we'll be happy to walk you through the process.

Location	Course	Date	Date	Duration	Duration
Online (Australia)	Excel Tips and Tricks	11 Apr 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	12 - 13 Apr 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	2 Days

Location	Course	Date	Date	Duration	Duration
Online (Australia)	Power Pivot, Power Query and Power BI	10 - 12 May 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	17 May 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	18 - 19 May 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	19 - 21 Jul 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	26 Jul 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	27 - 28 Jul 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	2 Days
Online (Australia)	Excel Tips and Tricks	29 Aug 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	30 - 31 Aug 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	28 -30 Sep 2022	09:00-17:00 AEST	(-1 day) 23:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	5 Oct 2022	09:00-17:00 AEDT	(-1 day) 22:00-06:00 GMT	1 Day
Online (Australia)	Financial Modelling	6 - 7 Oct 2022	09:00-17:00 AEDT	(-1 day) 22:00-06:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	9 - 11 Nov 2022	09:00-17:00 AEDT	(-1 day) 22:00-06:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	16 Nov 2022	09:00-17:00 AEDT	(-1 day) 22:00-06:00 GMT	1 Day
Online (Australia)	Financial Modelling	17 - 18 Nov 2022	09:00-17:00 AEDT	(-1 day) 22:00-06:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	7 - 9 Dec 2022	09:00-17:00 AEDT	(-1 day) 22:00-06:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	14 Dec 2022	09:00-17:00 AEDT	(-1 day) 22:00-06:00 GMT	1 Day
Online (Australia)	Financial Modelling	15 - 16 Dec 2022	09:00-17:00 AEDT	(-1 day) 22:00-06:00 GMT	2 Days

## Key Strokes

Each newsletter, we'd like to introduce you to useful keystrokes you may or may not be aware of. This month we thought we'd do the **RIGHT** thing with this month's keyboard shortcuts:

Keystroke	What it does
RIGHT	Move one cell to the right
ALT + RIGHT	Forward (hyperlink navigation)
CTRL + RIGHT	Select the last cell in the area right
SHIFT + RIGHT	Extend selection one cell right
ALT + SHIFT + RIGHT	Group
CTRL + ALT + RIGHT	Intel Chipset: Turn screen -90 degrees; else: move active cell to next non-adjacent area within selection
CTRL + SHIFT + RIGHT	Extend selection down to last cell in area right

There are c.550 keyboard shortcuts in Excel. For a comprehensive list, please download our Excel file at [www.sumproduct.com/thought/keyboard-shortcuts](http://www.sumproduct.com/thought/keyboard-shortcuts). Also, check out our new daily **Excel Tip of the Day** feature on the [www.sumproduct.com](http://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](mailto: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](http://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](mailto: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](mailto:training@sumproduct.com) for a copy of the brochure or download it directly from [www.sumproduct.com/training](http://www.sumproduct.com/training).

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

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