SumProduct

NEWSLETTER #83 - October 2019

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

Big news this month – Excel has two new functions to add to its armoury. You can meet them below. They may only be in Preview mode at the time of writing, but they are going to have a massive impact upon the Excel community when fully released into the wild…

There are also the usual suspects: Keyboard Shortcuts, Visual Basics, Power Pivot Principles, Power Query Pointers and the standard 20,000 Power BI updates.

Until next month.

Liam Bastick, Managing Director, SumProduct

Meet the New X-Men: XLOOKUP and XMATCH

Late August 2019 and Microsoft has added two new functions, **XLOOKUP** and **XMATCH**. For reasons that will become clear, here I will mainly consider the former function – because once you understand **XLOOKUP**, **XMATCH** becomes obvious (nothing personal, **XMATCH**).

Therefore, let's take a look at the new addition to the **LOOKUP** family. I so wanted it to be called **FLOOKUP** but it was not to be…

Ask anyone and they will tell you two "truths":

1. They are a better than average driver and everyone else is an idiot on the roads
2. They are a better than average Excel user because they know how to use **VLOOKUP**.

It's well known I hate **VLOOKUP** with a passion and if anything can come along and hurry its demise, well, I shall welcome it with open arms. Ladies and gentlemen, may I present the future of looking up for the masses – **XLOOKUP**. Hopefully, it will make an "ex" of **VLOOKUP!**

Why I Loathe VLOOKUP

Just as a recap, let me just summarise the resident incumbent:

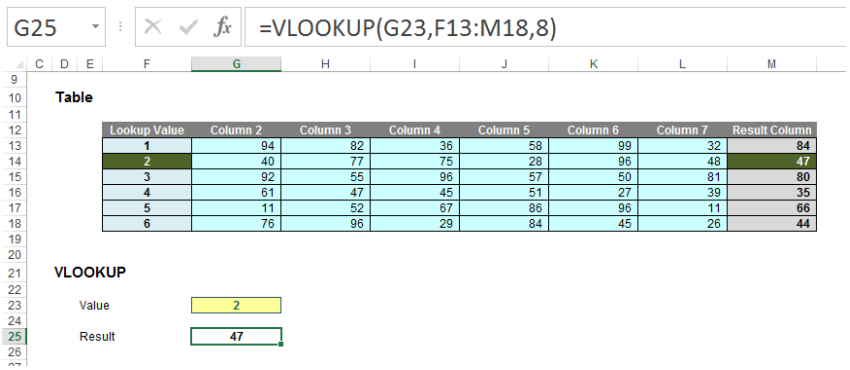
VLOOKUP(lookup_value, table_array, column_index_number, [range_lookup])

has the following syntax:

- **lookup_value:** what value do you want to look up?
- **table_array:** where is the lookup table?
- **column_index_number:** which column has the value you want returned?
- **[range_lookup]:** do you want an exact or an approximate match? This is optional and to begin with, I am going to ignore this argument exists.

HLOOKUP is similar, but works on a row, rather than a column, basis.

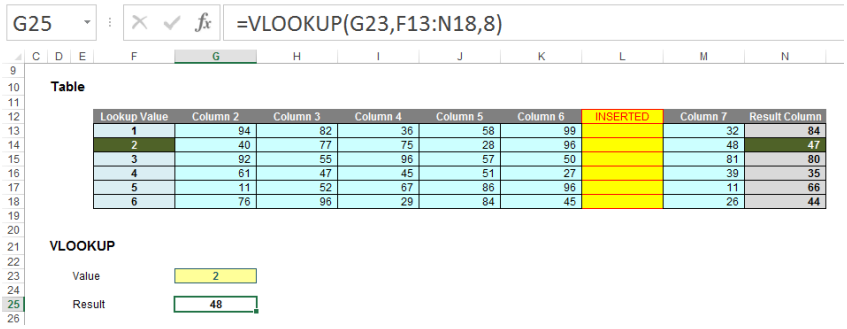
To show my disdain, I am going to use **VLOOKUP** throughout to keep things simple. **VLOOKUP** always looks for the **lookup_value** in the first column of a table (the **table_array**) and then returns a corresponding value so many columns to the right, determined by the **column_index_number**.



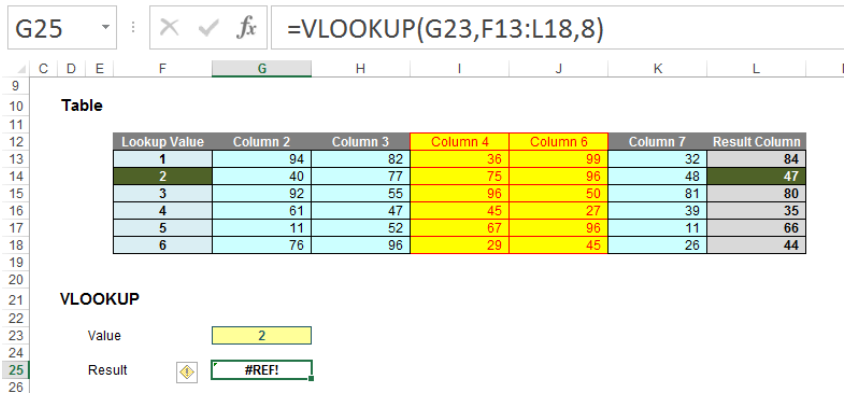
In this above example, the formula in cell **G25** seeks the value 2 in the first column of the table **F13:M18** and returns the corresponding value from the eighth column of the table (returning 47).

Pretty easy to understand; so far so good. So, what goes wrong? Well, what happens if you add or remove a column from the table range?

Adding (inserting) a column gives us the wrong value:

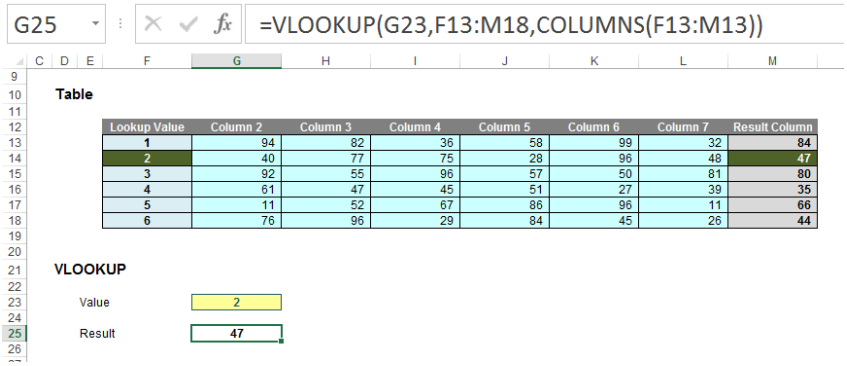


With a column inserted, the formula contains hard code (8) and therefore, the eighth column (**M**) is still referenced, giving rise to the wrong value. Deleting a column instead is even worse:



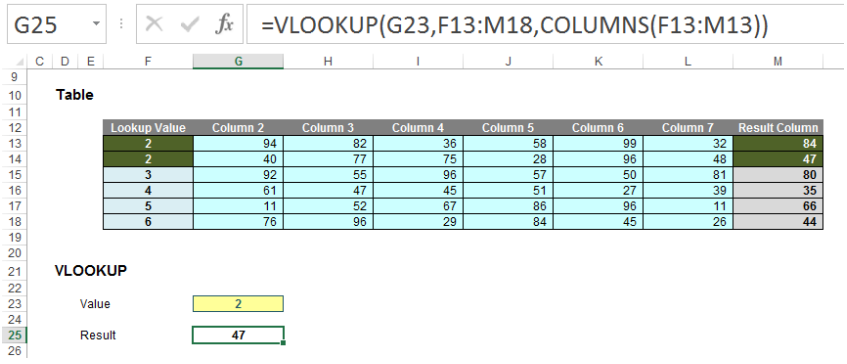
Now there are only seven columns so the formula returns *#REF!* Oops.

It is possible to make the column index number dynamic using the **COLUMNS** function:

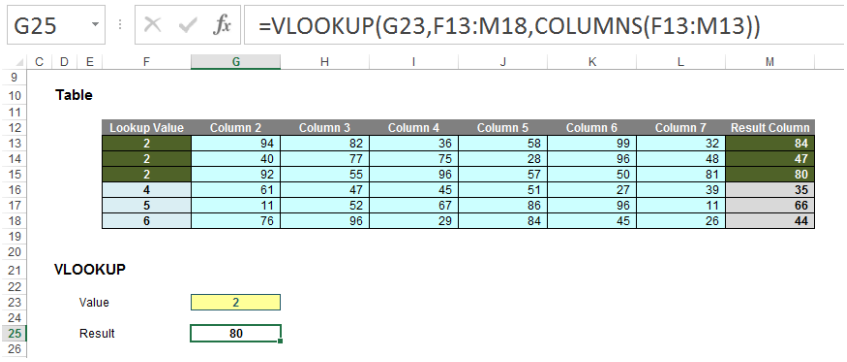


COLUMNS(reference) counts the number of columns in the **reference**. Using the range **F13:M13**, this formula will now keep track of how many columns there are between the lookup column (**F**) and the result column (**M**). This will prevent the problems illustrated above.

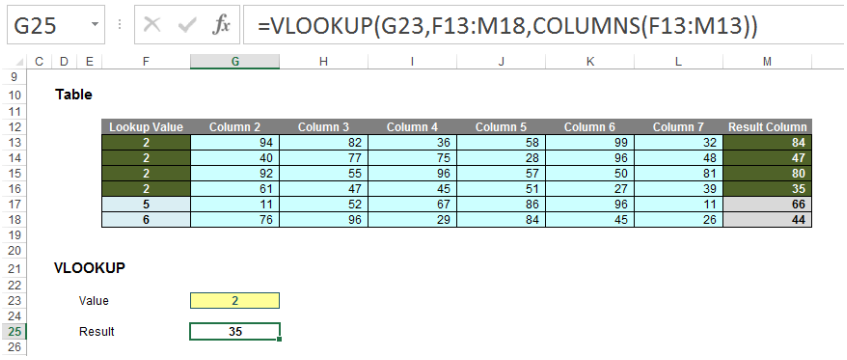
But there's more issues. Consider duplicate values in the lookup column. With one duplicate, the following happens:



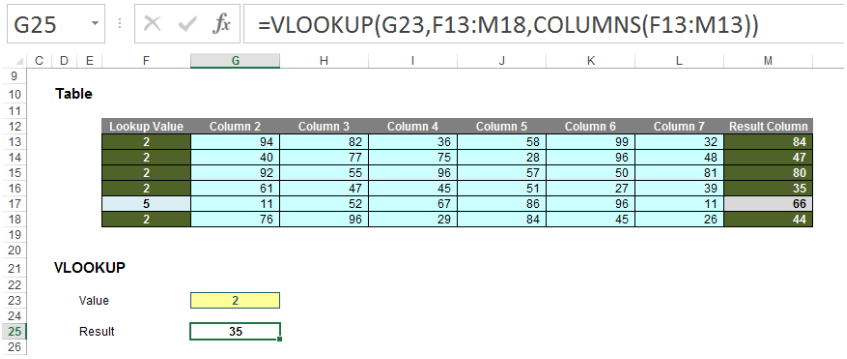
Here, the second value is returned, which might not be what is wanted. With two duplicates:



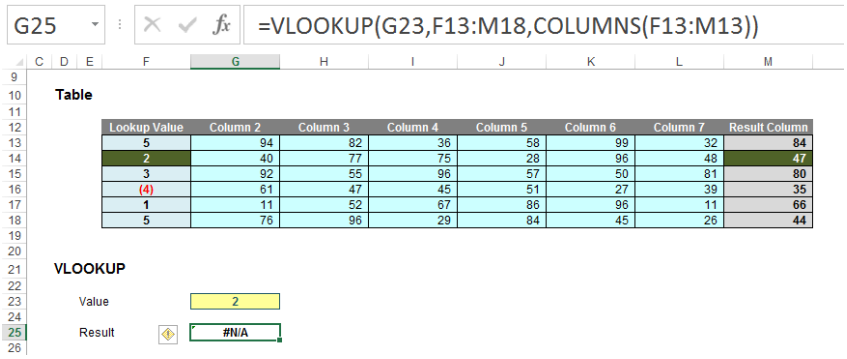
Ah, it looks like it might take the last occurrence. Testing this hypothesis with three duplicates:



Yes, there seems to be a pattern: **VLOOKUP** takes the last occurrence. Better make sure:



Rats. In this example, the value returned is the fourth of five. The problem is, there's no consistent logic and the formula and its result cannot be relied upon. It gets worse if we exclude duplicates but mix up the lookup column a little:



In this instance, **VLOOKUP** cannot even find the value 2!

So, what's going on? The problem – and common modelling mistake – is that the fourth argument has been ignored:

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

[range_lookup] appears in square brackets, which means it is optional. It has two values:

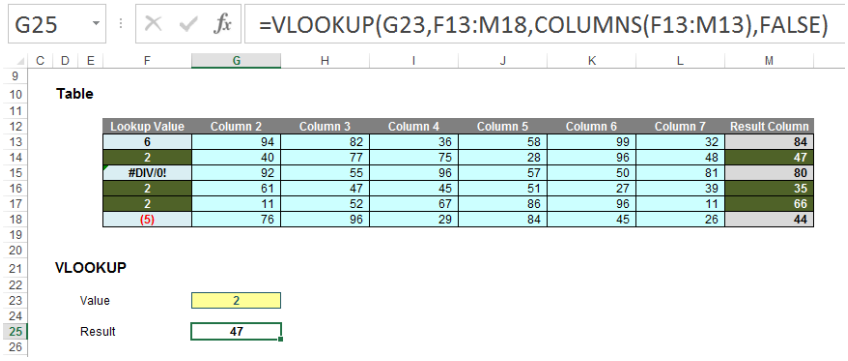
- **TRUE:** this is the default setting if the argument is not specified. Here, **VLOOKUP** will seek an approximate match, looking for the largest value less than or equal to the value sought.
  There is a price to be paid though: the values in the first column (or row for **HLOOKUP**) must be in strict ascending order – this means that each value must be larger than the value before, so no duplicates.

  This is useful when looking up postage rates for example where prices are given in categories of pounds and you have 2.7lb to post (say). It's worth noting though that this isn't the most common lookup when modelling.

- **FALSE:** this has to be specified. In this case, data can be any which way – including duplicates – and the result will be based upon the first occurrence of the value sought. If an exact match cannot be found, **VLOOKUP** will return the value *#N/A*.

And this is the problem highlighted by the above examples. The final argument was never specified so the lookup column data has to be in strict ascending order – and this premiss was continually breached.

The robust formula needs both **COLUMNS** and a fourth argument of **FALSE** to work as expected:

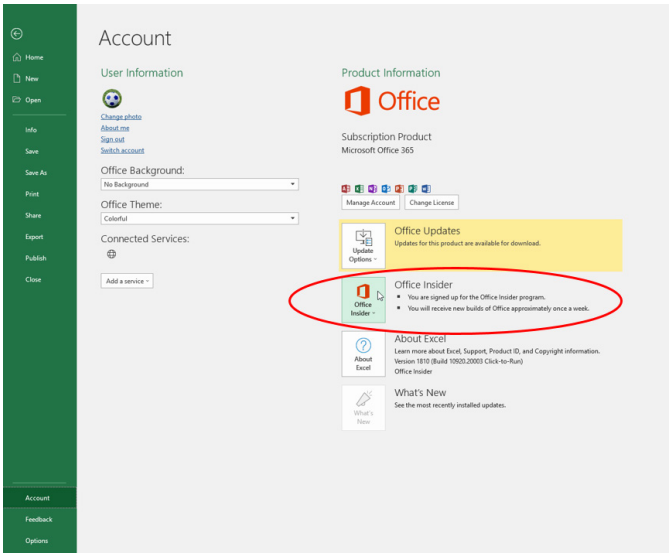


This is a very common mistake in modelling. Using a fourth argument of **FALSE**, **VLOOKUP** will return the corresponding result for the first occurrence of the **lookup_value**, regardless of number of duplicates, errors or series order. If an approximate match is required, the data must be in strict ascending order.

VLOOKUP (and consequently **HLOOKUP**) are not the simple, easy to use functions people think they are. In fact, they can never be used to return data for columns to the left (**VLOOKUP**) or rows above (**HLOOKUP**). So, what should modellers use instead..?

Introducing XLOOKUP

There's a new boss in town, but it's only in selected towns presently. This function has been released in what Microsoft refers to as "Preview" mode, *i.e.* it's not yet "Generally Available" but it is something you can try and hunt out. Presently, just like dynamic arrays, you need to be part of what is called the "Office Insider" programme which is an Office 365 fast track. You can register in **File -> Account -> Office Insider** in Excel's backstage area.



Even then, you're not guaranteed a ticket to the ball as only some will receive the new function as Microsoft slowly roll out these features and functions. Please don't let that put you off. This feature will be with all Office 365 subscribers soon.

XLOOKUP has the following syntax:

XLOOKUP(lookup_value, lookup_vector, results_array, [match_mode], [search_mode])

On first glance, it looks like it has too many arguments, but often you will only use the first three:

- **lookup_value:** this is required and defines what value you want to look up
- **lookup_vector:** this reference is required and is the row or column of data you are referencing to look up **lookup_value**

- **results_array:** this is where the corresponding item is you wish to return and is also required (even if it is the same as **lookup_vector**). This does not have to be a vector (*i.e.* one row or one column of cells): it may be an array (with at least two rows and at least two columns of cells). The only stipulation is that the number of rows / columns must equal the number of rows / columns in the column / row vector – but more on that later
- **match_mode:** this argument is optional. There are four choices:
  - **0:** exact match (default)
  - **-1:** exact match or else the largest value less than or equal to **lookup_value**
  - **1:** exact match or else smallest value greater than or equal to **lookup_value**
  - **2:** wildcard match. You should use the special character **?** to match any character and ***** to match any run of characters.

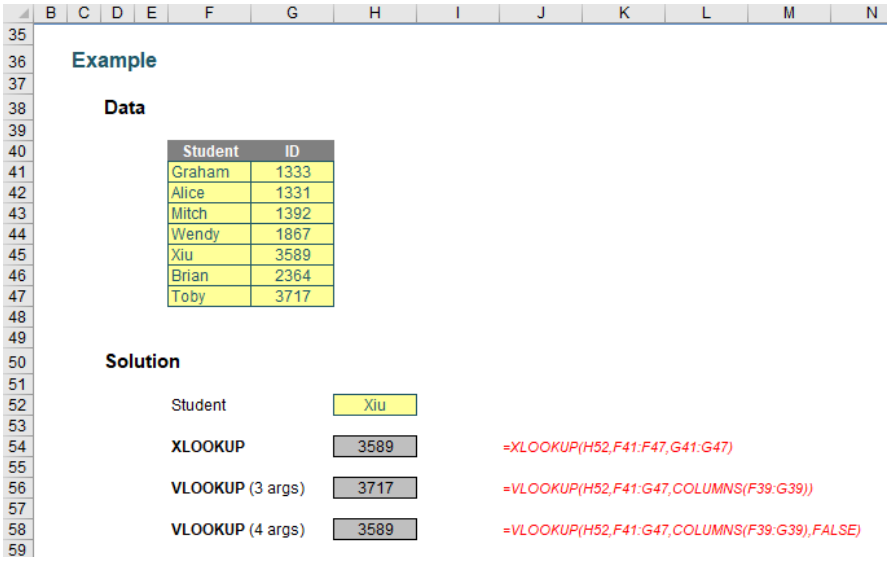
  What's impressive, though, is that for certain selections of the final argument (**search_mode**), you don't need to put your data in alphanumerical order! As far as I am aware, this is a first for Excel
- **search_mode:** this argument is also optional. There are again four choices:
  - **1:** search first to last (default)
  - **-1:** search last to first
  - **2:** what is known as a binary search, first to last (requires **lookup_vector** to be sorted). Just so you know, a binary search is a search algorithm that finds the position of a target value within a sorted array. A binary search compares the target value to the middle element of the array. If they are not equal, the half in which the target cannot lie is eliminated and the search continues on the remaining half, again taking the middle element to compare to the target value, and repeating this until the target value is found
  - **-2:** another binary search, this time last to first (and again, this requires **lookup_vector** to be sorted).

XLOOKUP compares favourably with VLOOKUP

While **VLOOKUP** is the third most used function in Excel (behind **SUM** and **AVERAGE**), it has several well-known limitations which **XLOOKUP** overcomes:

- **it defaults to an "approximate" match:** most often, users want an exact match, but this is not **VLOOKUP**'s default behaviour. To perform an exact match, you need to set the final argument to FALSE (as explained earlier). If you forget (which is easy to do), you'll probably get the wrong answer
- **it does not support column insertions / deletions: VLOOKUP**'s third argument is the column number you'd like returned. Since this is a hard-coded number, if you insert or delete a column you need to increment or decrement the column number inside the **VLOOKUP** – hence the need for the **COLUMNS** function (and the corresponding **ROWS** function for **HLOOKUP**)
- **it cannot look to the left: VLOOKUP** always searches the first column, then returns a column to the right. There is no way to return values from a column to the left, forcing users to rearrange their data
- **it cannot search from the bottom:** If you want to find the last occurrence, you need to reverse the order of your data
- **it cannot search for next larger item:** when performing an "approximate" match, only the item less than or equal to the searched item can be returned and only if correctly sorted
- **references more cells than necessary: VLOOKUP**'s second argument, **table_array**, needs to stretch from the lookup column to the results column. As a result, it typically references more cells than it truly depends on. This could result in unnecessary calculations, reducing the performance of your spreadsheets.

Let's have a look at **XLOOKUP** versus **VLOOKUP**:

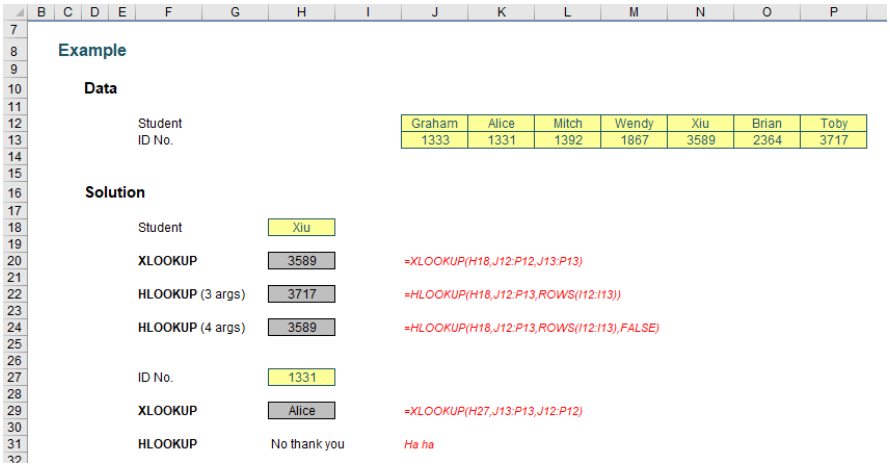


You can clearly see the **XLOOKUP** function is shorter:

=XLOOKUP(H52,F41:F47,G41:G47)

Only the first three arguments are needed, whereas **VLOOKUP** requires both a fourth argument, and, for full flexibility, the **COLUMNS** function as well. **XLOOKUP** will automatically update if rows / columns are inserted or deleted. It's just *simpler*.

HLOOKUP has similar issues:



Example

Data

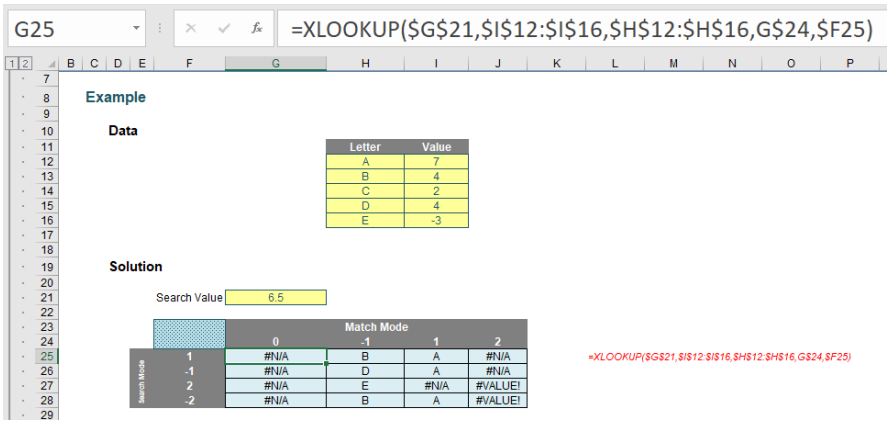
Student ID No.	Graham	Alice	Mitch	Wendy	Xiu	Brian	Toby
	1333	1331	1392	1867	3589	2364	3717

Solution

Student	Xiu	
XLOOKUP	3589	=XLOOKUP(H18,J12:P12,J13:P13)
HLOOKUP (3 args)	3717	=HLOOKUP(H18,J12:P13,ROWS(I12:I13))
HLOOKUP (4 args)	3589	=HLOOKUP(H18,J12:P13,ROWS(I12:I13),FALSE)
ID No.	1331	
XLOOKUP	Alice	=XLOOKUP(H27,J13:P13,J12:P12)
HLOOKUP	No thank you	Ha ha

Here, this highlights what happens if I try to deduce the student name from the Student ID. **HLOOKUP** cannot refer to earlier rows, just as **VLOOKUP** cannot consider columns to the left. Given any unused elements of the table are ignored also, it's just good news all round. Goodbye limitations, hello **XLOOKUP**.

Indeed, things get even more interesting when you start considering **XLOOKUP**'s final two arguments, namely **match_mode** and **search_mode**, *viz*.



Notice that I am searching the 'Value' column, which is neither sorted nor contains unique items. However, I can look for approximate matches – impossible with **VLOOKUP** and / or **HLOOKUP**.

Do you see how the results vary depending upon **match_mode** and **search_mode**?

Search Mode \ Match Mode	0	-1	1	2
1	#N/A	B	A	#N/A
-1	#N/A	D	A	#N/A
2	#N/A	E	#N/A	#VALUE!
-2	#N/A	B	A	#VALUE!

The **match_mode** zero (0) returns *#N/A* because there is no exact match.

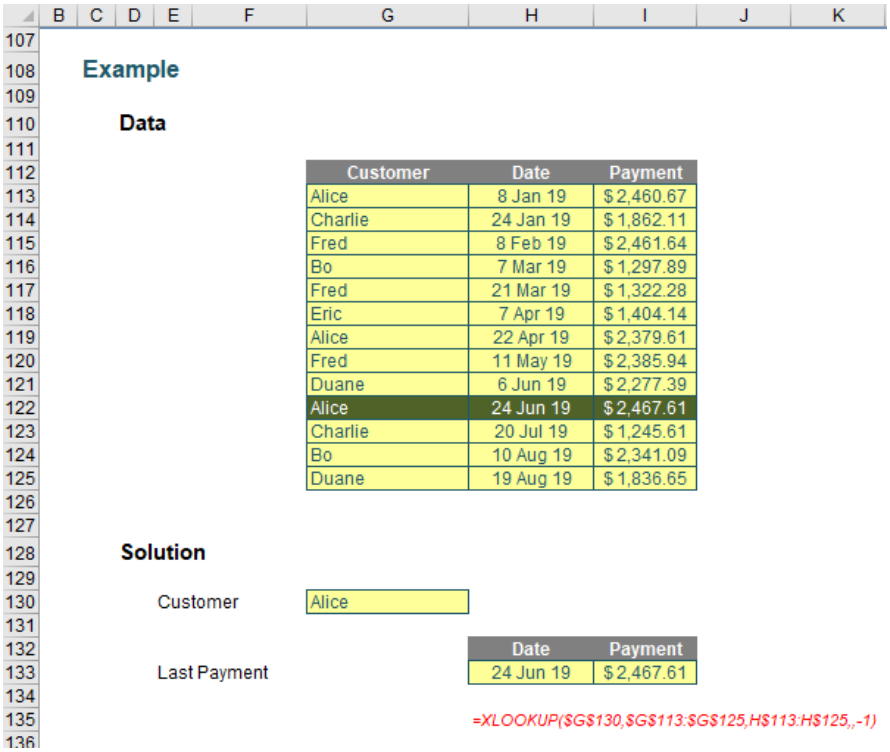
When **match_mode** is -1, **XLOOKUP** seeks an exact match or else the largest value less than or equal to **lookup_value** (6.5). That would be 4 – but this occurs more than once (B and D both have a value of 4). **XLOOKUP** chooses depending upon whether it is searching top down (**search_mode** 1, where B will be identified first) or bottom up (**search_mode** -1, where D will be identified first). Note that with binary searches (with a **search_mode** of 2 or -2), the data needs to be sorted. It isn't – hence we have garbage answers that cannot be relied upon.

With **match_mode** 1, the result is clearer cut. Only one value is the smallest value greater than or equal to 6.5. That is 7, and is related to A. Again, binary search results should be ignored.

The **match_mode** 2 results are spurious. This is seeking wildcard matches, but there are no matches, hence *N/A* for the only **search_modes** that may be seen as creditable (1 and -1).

Clearly binary searches are higher maintenance. In the past, it was worth investing in them as they did return results more quickly. However, according to Microsoft, this is no longer the case: apparently, there is "… no significant benefit to using (sic) the binary search options…". If this is indeed the case, then I would strongly recommend not using them going forward with **XLOOKUP**.

To show how simple it now is to search from the end, consider the following:



This used to be an awkward calculation – but not anymore! The formula is easy:

=XLOOKUP($G$130,$G$113:$G$125,H$113:H$125,,-1)

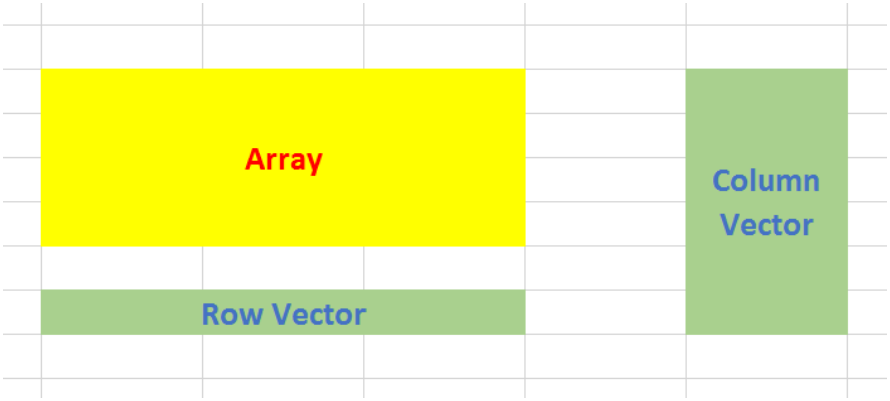
It's a "standard" **XLOOKUP** formula, with a "bottom up" search coerced by using the final value of -1 (forcing the **search_mode** to go into "reverse").

Comparisons with LOOKUP

Whilst **XLOOKUP** wins hands down against **HLOOKUP** and **VLOOKUP**, the same cannot necessarily be said for **LOOKUP**. You may recall **LOOKUP** has two forms: an array form and a vector form. As a reminder:

- an **array** is a collection of cells consisting of at least two rows and at least two columns
- a **vector** is a collection of cells across just one row (row vector) or down just one column (column vector).

The diagram should be self-explanatory:



The array form of **LOOKUP** looks in the first row or column of an array for the specified value and returns a value from the same position in the last row or column of the same array:

LOOKUP(lookup_value, array)

where:

- **lookup_value** is the value that **LOOKUP** searches for in an array. The **lookup_value** argument can be a number, text, a logical value, or a name or reference that refers to a value
- **array** is the range of cells that contains text, numbers, or logical values that you want to compare with **lookup_value**.

The array form of **LOOKUP** is very similar to the **HLOOKUP** and **VLOOKUP** functions. The difference is that **HLOOKUP** searches for the value of **lookup_value** in the first row, **VLOOKUP** searches in the first column, and **LOOKUP** searches according to the dimensions of array.

If **array** covers an area that is wider than it is tall (*i.e.* it has more columns than rows), **LOOKUP** searches for the value of **lookup_value** in the first row and returns the result from the last row. Otherwise, **LOOKUP** searches for the value of **lookup_value** in the first column and returns the result from the last column instead.

The alternative form is the vector form:

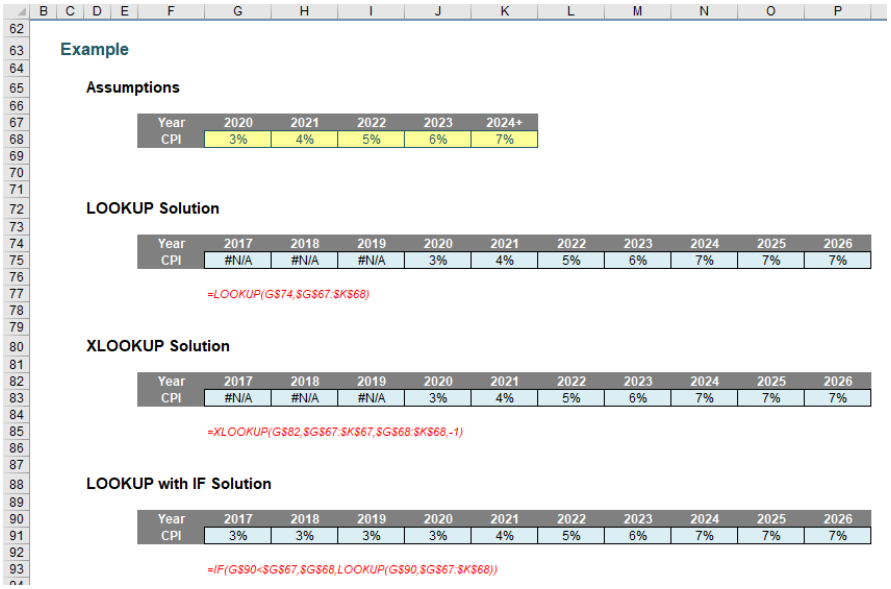
LOOKUP(lookup_value, lookup_vector, [result_vector])

The **LOOKUP** function vector form syntax has the following arguments:

- **lookup_value** is the value that **LOOKUP** searches for in the first vector
- **lookup_vector** is the range that contains only one row or one column
- **[result_vector]** is optional – if ignored, **lookup_vector** is used – this is the where the result will come from and must contain the same number of cells as the **lookup_vector**.

Like the default versions of **HLOOKUP** and **VLOOKUP, lookup_value** must be located in a range of ascending values.

Let me demonstrate with an example:



Example

Assumptions

Year	2020	2021	2022	2023	2024+
CPI	3%	4%	5%	6%	7%

LOOKUP Solution

Year	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026
CPI	#N/A	#N/A	#N/A	3%	4%	5%	6%	7%	7%	7%

=LOOKUP(G$74,$G$67:$K$68)

XLOOKUP Solution

Year	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026
CPI	#N/A	#N/A	#N/A	3%	4%	5%	6%	7%	7%	7%

=XLOOKUP(G$82,$G$67:$K$67,$G$68:$K$68,-1)

LOOKUP with IF Solution

Year	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026
CPI	3%	3%	3%	3%	4%	5%	6%	7%	7%	7%

=IF(G$90<$G$67,$G$68,LOOKUP(G$90,$G$67:$K$68))

LOOKUP is a great function to use with time series analysis / forecasting. Dates are in ascending order and the **LOOKUP** syntax is remarkably simple. As a modeller, I use it regularly when I am modelling many more forecast periods than I want assumption periods.

Here, you can see I carry assumptions only for 2020 until 2024 (the final value is 2024, just with a "+" in number formatting). The formula

=LOOKUP(G$74,$G$67:$K$68)

returns the corresponding value for the period that is either an exact match or else the largest value less than or equal to the lookup_value. **LOOKUP** uses the top row of the table for looking up its data and the final row for returning the corresponding value. Simple. As for **XLOOKUP**:

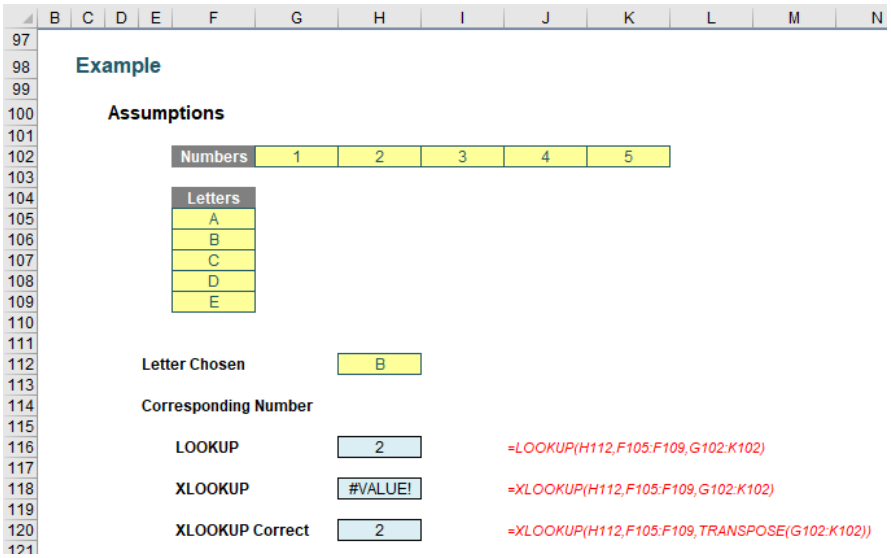
=XLOOKUP(G$82,$G$67:$K$67,$G$68:$K$68,-1)

This formula is longer and requires two additional arguments (**match_mode** -1 is required to mirror the behaviour of **LOOKUP**). Indeed, given that an **IF** statement is required to ensure no errors for earlier periods, *e.g.*

=IF(G$90<$G$67,$G$68,LOOKUP(G$90,$G$67:$K$68))

it may be argued that **LOOKUP** is a simpler function to use here than its counterpart.

This isn't the only time **LOOKUP** outperforms **XLOOKUP**:



Here, we do see a limitation of **XLOOKUP**. Whilst the third argument of **XLOOKUP**, **results_array**, does not need to be a vector, it cannot be the transposition of the **lookup_vector**. You would have to transpose it using the **TRANSPOSE** function, for example. This makes **LOOKUP** much easier to use – compare:

=LOOKUP(H112,F105:F109,G102:K102)

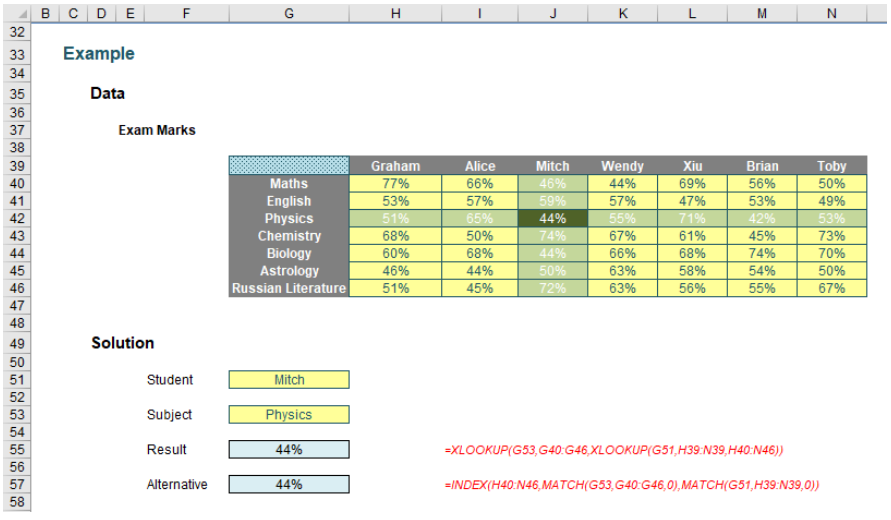
with

=XLOOKUP(H112,F105:F109,TRANSPOSE(G102:K102))

In this instance, **LOOKUP** wins.

Useful Features of XLOOKUP

XLOOKUP can be used to perform a two-way match, similar to **INDEX MATCH**:



	Graham	Alice	Mitch	Wendy	Xiu	Brian	Toby
Maths	77%	66%	46%	44%	69%	56%	50%
English	53%	57%	59%	57%	47%	53%	49%
Physics	51%	65%	44%	55%	71%	42%	53%
Chemistry	68%	50%	74%	67%	61%	45%	73%
Biology	60%	68%	44%	66%	68%	74%	70%
Astrology	46%	44%	50%	63%	58%	54%	50%
Russian Literature	51%	45%	72%	63%	56%	55%	67%

Many advanced users might use the formula

=INDEX(H40:N46,MATCH(G53,G40:G46,0),MATCH(G51,H39:N39,0))

where:

- **INDEX(array, row_number, [column_number])** returns a value or the reference to a value from within a table or range (list) citing the **row_number** and the **column_number**
- **MATCH(lookup_value, lookup_vector, [match_type])** returns the relative position of an item in an array that (approximately) matches a specified value. It's most commonly used with **match_type** zero (0), which requires an exact match.

Therefore, this formula finds the position in the row for the student and the position in the column of the subject. The intersection of these two provides the required result.

XLOOKUP does it differently:

=XLOOKUP(G53,G40:G46,XLOOKUP(G51,H39:N39,H40:N46))

Welcome to the wonderful world of the *nested* **XLOOKUP** function! Here, the internal formula

=XLOOKUP(G51,H39:N39,H40:N46)

demonstrates a key difference between this and your typical lookup function – the first argument is a cell, the second argument is a column vector and the third is an array – with, most importantly, the same number of rows as the **lookup_vector**. This means it returns a column vector of data, not a single value. This is great news in the brave new world of dynamic arrays.

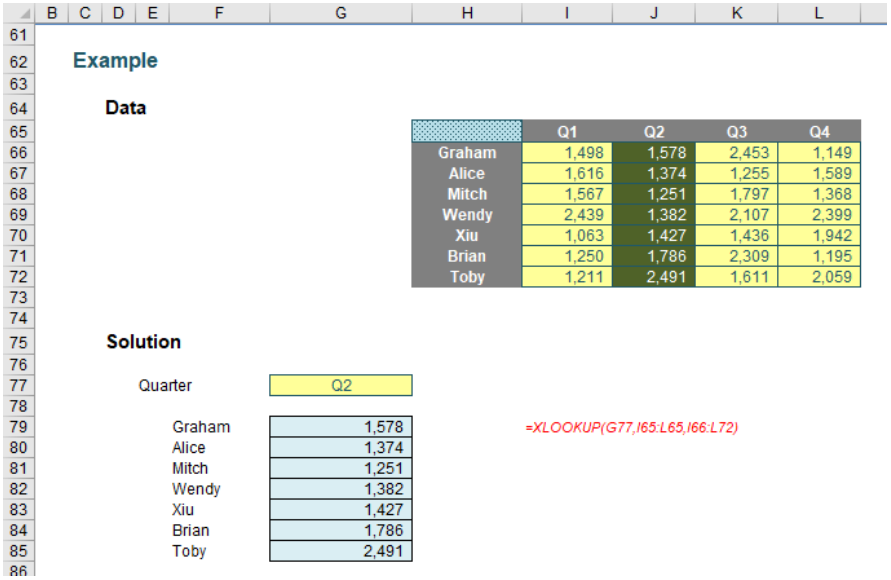
In essence, this means the formula resolves to

=XLOOKUP(G53,G40:G46,J40:J46)

as **J40:J46** is the resultant vector of **=XLOOKUP(G51,H39:N39,H40:N46)**. This is a really powerful – and virtually new – concept to get your head around, that admittedly **SUMPRODUCT** exploits too. Once you understand this, it's clear how this formula works and opens your eyes to the power of nested **XLOOKUP** functions.

I can't believe I am talking about the virtues of nested functions here! Let me change the subject quickly…

To show you how dynamic arrays can make the most of being able to create resultant vectors, consider the following example:



Example

Data

	Q1	Q2	Q3	Q4
Graham	1,498	1,578	2,453	1,149
Alice	1,616	1,374	1,255	1,589
Mitch	1,567	1,251	1,797	1,368
Wendy	2,439	1,382	2,107	2,399
Xiu	1,063	1,427	1,436	1,942
Brian	1,250	1,786	2,309	1,195
Toby	1,211	2,491	1,611	2,059

Solution

Quarter: Q2

Graham	1,578
Alice	1,374
Mitch	1,251
Wendy	1,382
Xiu	1,427
Brian	1,786
Toby	2,491

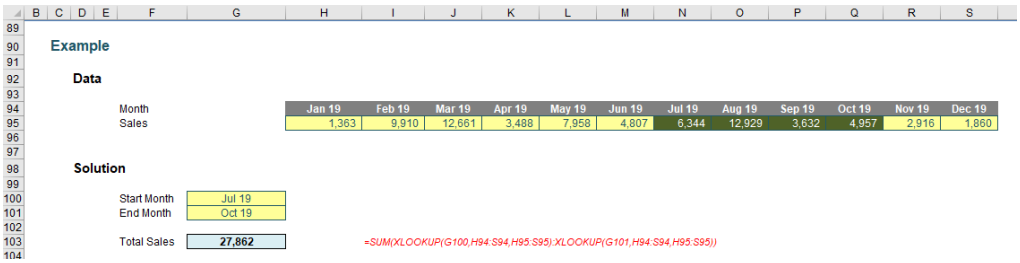
=XLOOKUP(G77,I65:L65,I66:L72)

The formula

=XLOOKUP(G77,I65:L65,I66:L72)

again resolves to a vector – but this time is allowed to spill as a dynamic array. Obviously, this will only work in Office 365, but it's a very useful tool that might just make you think it's time to drop that perpetual licence.

Once you start playing with the dynamic range side, you can start to get imaginative. For example:



Example

Data

Month	Jan 19	Feb 19	Mar 19	Apr 19	May 19	Jun 19	Jul 19	Aug 19	Sep 19	Oct 19	Nov 19	Dec 19
Sales	1,363	9,910	12,661	3,488	7,958	4,807	6,344	12,929	3,632	4,957	2,916	1,860

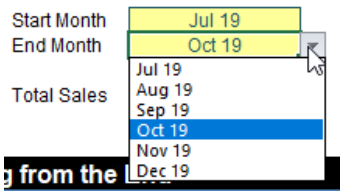
Solution

Start Month: Jul 19
End Month: Oct 19

Total Sales: 27,862

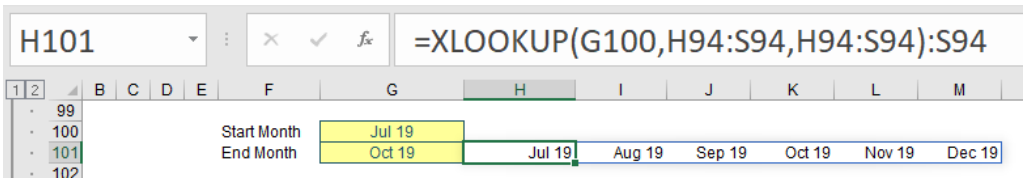
=SUM(XLOOKUP(G100,H94:S94,H95:S95):XLOOKUP(G101,H94:S94,H95:S95))

In this illustration, I want to calculate the sales between two periods:



This might seem like a simple drop-down list using data validation (**ALT + D + L**), but **XLOOKUP** has been used in determining the list to be used for the end months.

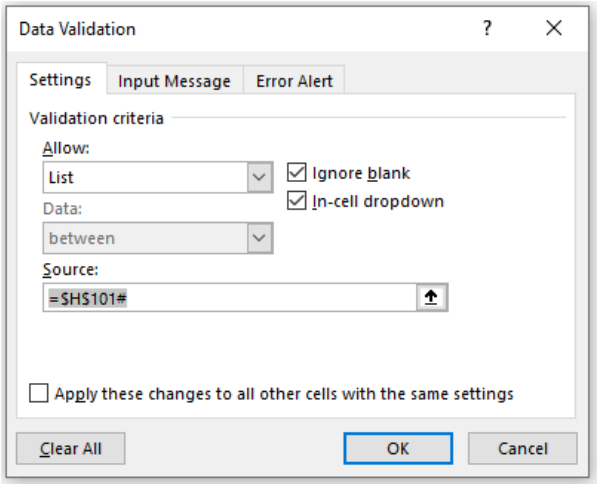
Let me explain. I have hidden the range of relevant dates in cell **H101** spilled across



XLOOKUP can return a reference, so the formula

=XLOOKUP(G100,H94:S94,H94:S94):S94

evaluates to the row vector **N94:S94** (since the start month is July). This spilled dynamic array formula is then referenced in the data validation:



(You may recall **$H$101#** means the spilled range starting in cell **H101**.) It should be noted that the formula **=XLOOKUP(G100,H94:S94,H94:S94):S94** may not be used directly in the 'Data Validation' dialog, but this is a neat trick to ensure you cannot select an end month before the start month (assuming you are a rational human being that selects the start before the end!).

The formula to sum the sales then is

=SUM(XLOOKUP(G100,H94:S94,H95:S95):XLOOKUP(G101,H94:S94,H95:S95))

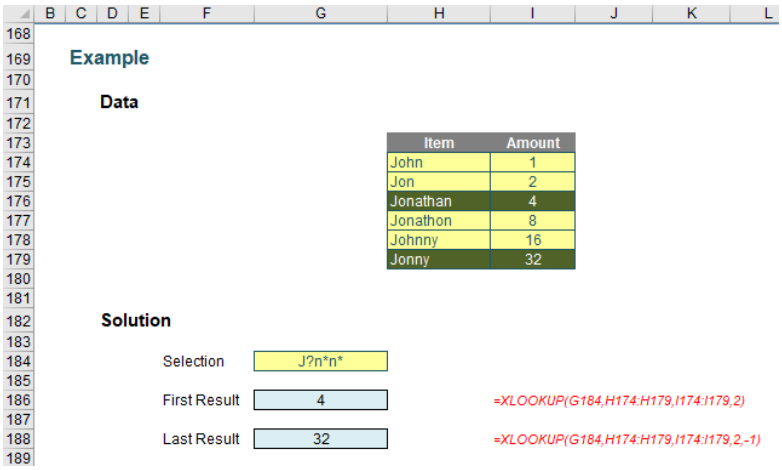
Again, this uses the fact **XLOOKUP** can return a reference, so this formula equates to

=SUM(N95:Q95)

Easy! Now I am combining two **XLOOKUP** formulae with a colon (:) to form a range. This joins other illustrious functions used this way such as **CHOOSE**, **IF**, **IFS**, **INDEX**, **INDIRECT**, **OFFSET**, **SINGLE (@)**, **SWITCH** and **TEXT**. First nesting, now joining – what's next?

Partial and Exact Matching

Seeking partial matches (sounds like an unfussy dating agency!) suddenly became a lot easier too. You can use wildcards if you want to – just set the **match_mode** to 2:



Here, I am searching for **J?n*n*** - which is fine as long as you know what the wildcard characters mean:

- **?** means "any character", but just one character. If you wanted to make space for two and only two characters you would use **??**
- ***** means "any number of characters' – including zero.

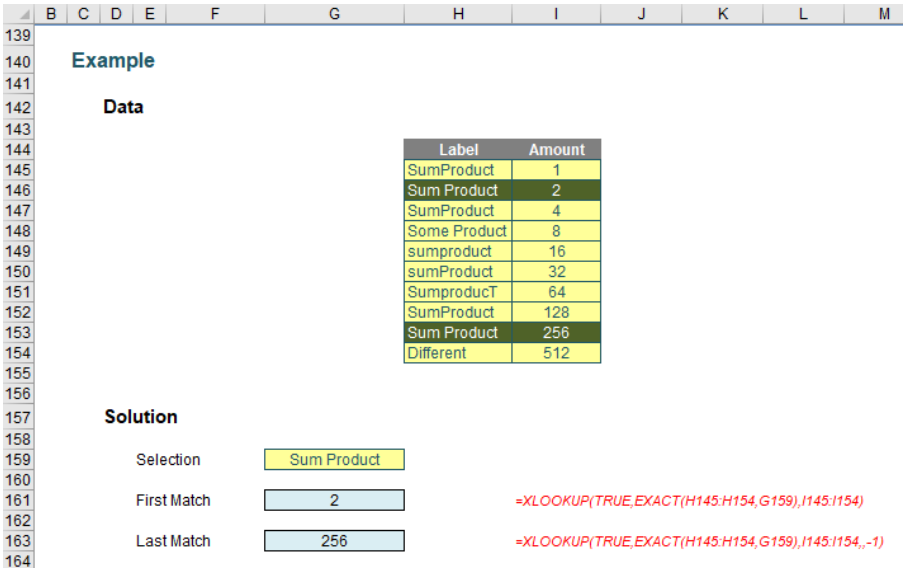
For example, **M?n*m*** would identify "Manmade", "minimum" and "Manikum" but would not accept "millennium". Here, our formulae

=XLOOKUP(G184,H174:H179,I174:I179,2)

=XLOOKUP(G184,H174:H179,I174:I179,2,-1)

would locate the first and last items that satisfied the condition **J?n*n*** (*i.e.* "Jonathan" and "Jonny" respectively).

But what if you wanted an exact match with case sensitivity? You just have to think a little but outside of the proverbial box:



Here, we use another feature of **XLOOKUP** – its ability to search a virtual vector, *i.e.* one that has been constructed in memory, rather than physically within the spreadsheet cells. Consider the formula

=XLOOKUP(TRUE,EXACT(H145:H154,G159),I145:I154)

Here, the interim calculation **=EXACT(H145:H154,G159)**, looks at the range **H145:H154** and deduces whether the cells are an exact match for the selection 'Sum Product' in cell **G159**. The **EXACT** function would evaluate as

{FALSE; TRUE; FALSE; FALSE; FALSE; FALSE; FALSE; FALSE; TRUE; FALSE}

Therefore, the formula coerces to

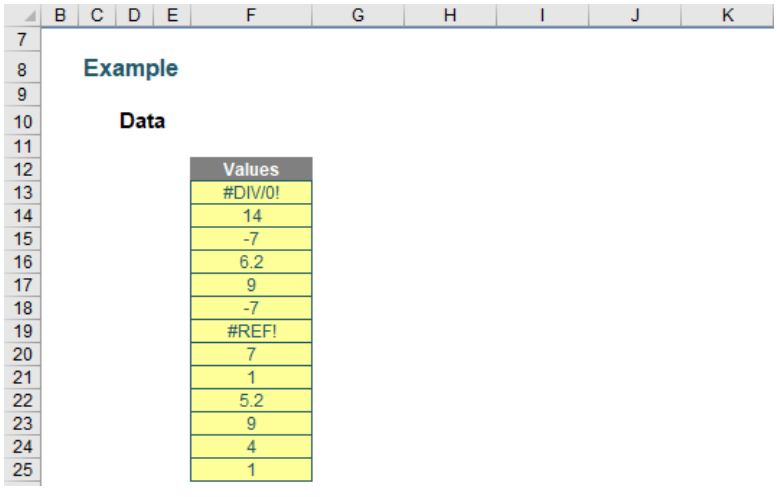
=XLOOKUP(TRUE,{FALSE; TRUE; FALSE; FALSE; FALSE; FALSE; FALSE; FALSE; TRUE; FALSE},I145:I154)

and then the formula becomes simple to understand.

No doubt there are many more great things you can do with **XLOOKUP**, but hey, it's just arrived and we are only getting started!

XMATCH

As I said at the beginning, **XLOOKUP** did not land in isolation. In addition to **XLOOKUP**, **XMATCH** has arrived with a similar signature to **XLOOKUP**, but instead it returns the index (position) of the matching item. **XMATCH** is both easier to use and more capable than its predecessor **MATCH**.



XMATCH has the following syntax:

XMATCH(lookup_value, lookup_vector, [match_mode], [search_mode])

where:

- **lookup_value:** this is required and defines what value you want to look up
- **lookup_vector:** this reference is required and is the row or column of data you are referencing to look up lookup_value
- **match_mode:** this argument is optional. There are four choices:
  - **0:** exact match (default)
  - **-1:** exact match or else the largest value less than or equal to **lookup_value**
  - **1:** exact match or else smallest value greater than or equal to **lookup_value**
  - **2:** wildcard match. You should use the special character **?** to match any character and ***** to match any run of characters.

  Again, for certain selections of the final argument (**search_mode**), you don't need to put your data in alphanumerical order
- **search_mode:** this argument is also optional. There are again four choices:
  - **1:** search first to last (default)
  - **-1:** search last to first
  - **2:** this is a binary search, first to last (requires **lookup_vector** to be sorted)
  - **-2:** another binary search, this time last to first (and again, this requires **lookup_vector** to be sorted).

As you can see, it's a fairly straightforward addition to the **MATCH** family. It acts similarly to **MATCH** – just with heaps more functionality.

Word to the Wise

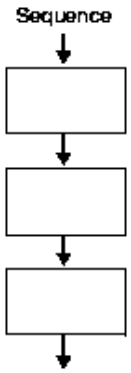
XLOOKUP and **XMATCH** open up new avenues for Excel to explore, but it must be remembered they are still in Preview and may only be accessed by a lucky few on the Insider track. Don't be too perturbed if your version of Excel does not recognise these functions yet.

Visual Basics

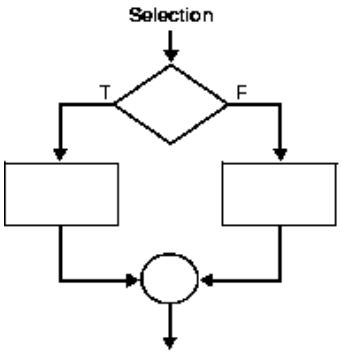
We thought we'd run an elementary series going through the rudiments of Visual Basic for Applications (VBA) as a springboard for newer users. This month, we continue our look at control structures, and iteration control structures in particular.

In a programming, a control structure determines the order in which statements are executed. Control structures can be grouped into three main categories:

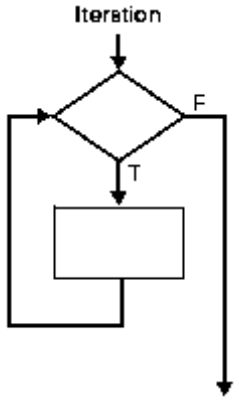
1. **Sequential:** Sequential execution is where each statement in the source code will be executed one by one in a sequential order. This is the default mode of execution



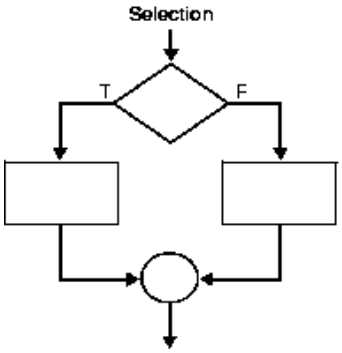
2. **Selection:** The selection control structure is used for making decisions and branching statements



3. **Iteration:** The iterative control structures are used for repetitively executing a block of code multiple times

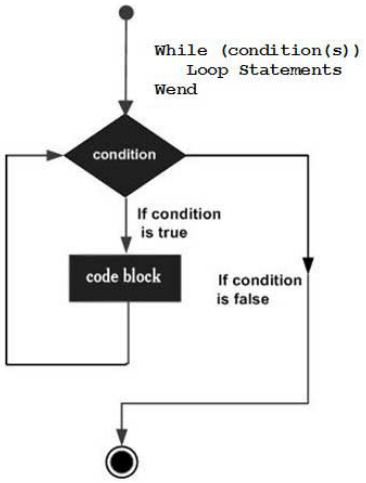


We've considered them all, as this month, we take a look at the final one. In programming, a control structure determines the order in which statements are executed. The iteration control structure is used for repetitively executing a block of code multiple times.



The iteration structure executes a sequence of statements repeatedly if a condition holds true. There are three main types of loops in VBA:

1. **WHILE…WEND**

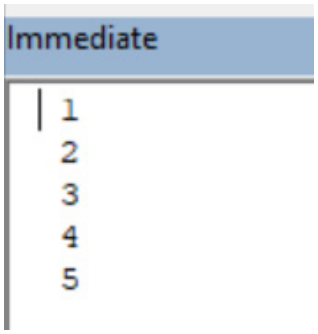


The **WHILE…WEND** loop executes a series of statements as long as a given condition is `True`. The syntax is very simple:

```
While condition
    [statements]
Wend
```

The condition must result in a Boolean value of **True** or **False**. **WHILE** tests the condition and if it is **True** then proceeds to execute the statements inside the loop.

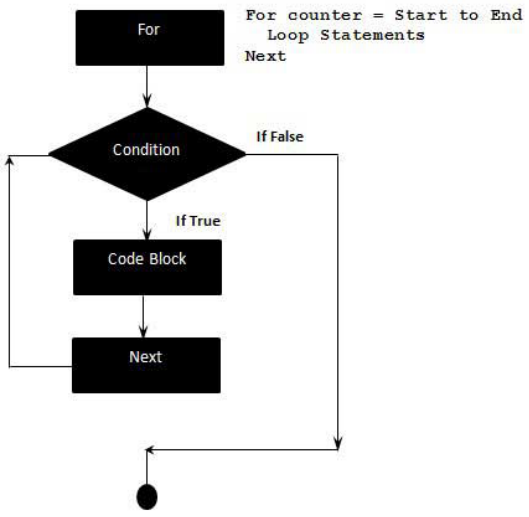
```
Sub WhileWend()

    Dim counter As Integer

    counter = 0

    While counter < 5

        counter = counter + 1

        Debug.Print counter

    Wend

End Sub
```



While loops are preferred when the number of iterations is unknown. For example, modelling how many days it takes to reach sales a target, or running through a worksheet column until it reaches an empty cell.

Notice how the condition is tested first – this means that the code will not run at all if the condition is not met. **WHILE…WEND** is a remnant from BASIC where VBA originated from. These are not as powerful as **DO…LOOP** (covered soon).

2. FOR

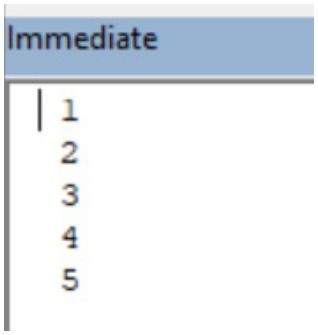


The **FOR…NEXT** loop uses a variable, which cycles through a series of values within a specified range and the statements inside the loop is then executed for each value.

```
For counter = start To end [ Step step ]

[ statements ]

Exit For

[ statements ]

Next [ counter ]
```

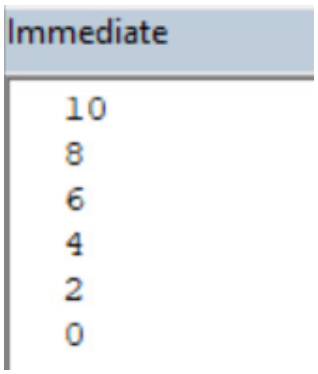
Here's a simple example:

```
Sub ForNext()
    Dim counter As Integer
    For counter = 1 To 5
        Debug.Print counter
    Next counter
End Sub
```



The **STEP** keyword allows the specification of how the counter changes. It defaults to an increment of 1, but it can be used for jumps and decrements.

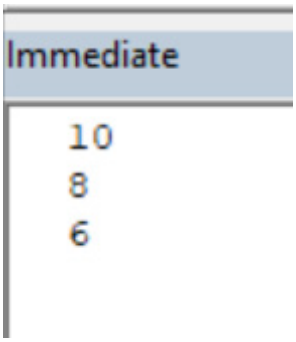
```
Sub ForNextStep()
    Dim counter As Integer
    For counter = 10 To 0 Step -2
        Debug.Print counter
    Next counter
End Sub
```



EXIT FOR

EXIT FOR statements may be placed anywhere in the loop as an alternate way to exit. This is often used after evaluating of some condition, for example **IF…THEN**, and then skips to the statements after the loop.

```
Sub ForNextExit()
    Dim counter As Integer
    For counter = 10 To 0 Step -2
        Debug.Print counter
        If counter = 6 Then
            Exit For
        End If
    Next counter
End Sub
```



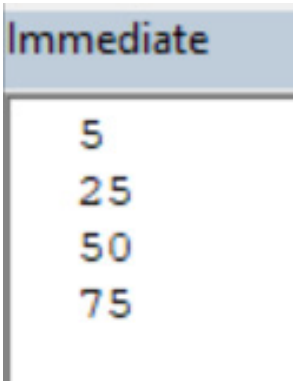
FOR EACH…NEXT

What if an action is needed to be performed to every object in a set?

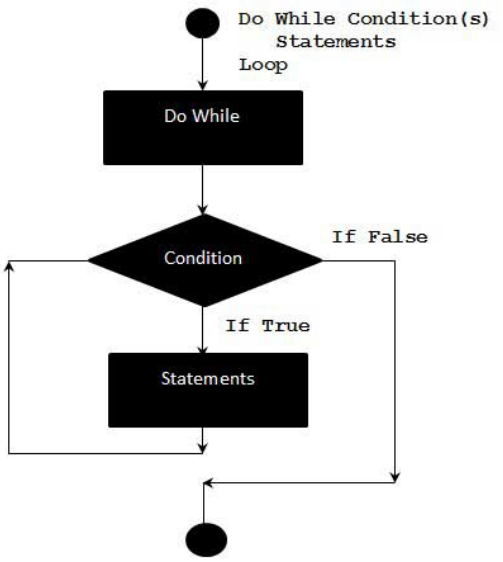
FOR EACH…NEXT loops are a great way to cycle through sets – like an array or a range. Sometimes the number of rows or columns is uncertain. It is relatively easy to count the number of objects and set the upper bound of the **FOR…NEXT** loop appropriately. However, using **FOR EACH…NEXT** more clearly illustrates that the instructions are happening to every object.

Example:

```
Sub ForEach()

    Dim myNumbers() As Variant

    myNumbers = Array(1, 5, 10, 15)

    Dim aNumber As Variant

    For Each aNumber In myNumbers

        Debug.Print aNumber * 5

    Next

End Sub
```



3. **DO**



DO…LOOP loops are considered the upgraded alternative to **WHILE WEND**. Let's have a look at how they work:

```
Do [{ While |Until } condition ]
      [ statements ]
      [ Exit Do ]
      [ statements]
Loop
```

How does the code change from **WHILE WEND** to **DO…LOOP**? Simply replace the **WHILE** with **DO WHILE** and **WEND** with **LOOP**. It's as easy as that!

```
Option Explicit

Sub JackAndJill()

Dim CurrentStepsTaken As Integer
CurrentStepsTaken = 0

Dim FallenDown As Boolean
FallenDown = False

While FallenDown = False

    CurrentStepsTaken = CurrentStepsTaken + 1

    If Rnd() <= 0.3 Then
        FallenDown = True
    End If

Wend

Debug.Print CurrentStepsTaken & " step(s) up the hill were taken."
Debug.Print "Jack fell down the hill!"
Debug.Print "Jill came tumbling after..."

End Sub
```

```
Option Explicit

Sub JackAndJillDo()

Dim CurrentStepsTaken As Integer
CurrentStepsTaken = 0

Dim FallenDown As Boolean
FallenDown = False

Do While FallenDown = False

    CurrentStepsTaken = CurrentStepsTaken + 1

    If Rnd() <= 0.3 Then
        FallenDown = True
    End If

Loop

Debug.Print CurrentStepsTaken & " step(s) up the hill were taken."
Debug.Print "Jack fell down the hill!"
Debug.Print "Jill came tumbling after..."

End Sub
```

DO…LOOP is superior to While Wend for several reasons:

- **WHILE WEND** has no ability to have an **EXIT**
- **WHILE WEND** loops check for the condition prior to running – but with **DO…LOOP** the condition can be checked at the end. This is useful if the code needs to be run at least once.

This is done by simply moving the "**WHILE** [condition]" part of the **DO** statement next to **LOOP**. The syntax changes to:

```
Do
[ statements ]
[ Exit Do ]
[ statements]
Loop [{ While |Until } condition ]
```

```
Sub JackAndJillDo()

Dim CurrentStepsTaken As Integer
CurrentStepsTaken = 0

Dim FallenDown As Boolean
FallenDown = False

Do While FallenDown = False

    CurrentStepsTaken = CurrentStepsTaken + 1

    If Rnd() <= 0.3 Then
        FallenDown = True
    End If

Loop

Debug.Print CurrentStepsTaken & " step(s) up the hill were taken."
Debug.Print "Jack fell down the hill!"
Debug.Print "Jill came tumbling after..."

End Sub
```

```
Sub JackAndJillDoItAtLeastOnce()

Dim CurrentStepsTaken As Integer
CurrentStepsTaken = 0

Dim FallenDown As Boolean
FallenDown = False

Do

    CurrentStepsTaken = CurrentStepsTaken + 1

    If Rnd() <= 0.3 Then
        FallenDown = True
    End If

Loop While FallenDown = False

Debug.Print CurrentStepsTaken & " step(s) up the hill were taken."
Debug.Print "Jack fell down the hill!"
Debug.Print "Jill came tumbling after..."

End Sub
```

- The ability to replace **WHILE** with **UNTIL:** what effect does this achieve? This essentially reverses the value of the condition to be tested.

  WHILE executes the block of code when the condition is `True` and keeps executing that till the condition becomes `False`. Once the condition becomes `False`, the loop is terminated. However, if the condition tested is initially `False`, the condition must be tested as:

  ```
  DO WHILE condition = FALSE
  ```

  UNTIL does the opposite. It executes the block of code when the condition is `False` and keep executing that till the condition becomes `True`. Once the condition becomes `True`, the **UNTIL** loop is terminated.

  It should be noted that the `[condition]` is a Boolean value, the loop can then be adjusted with the starting statement:

  ```
  DO UNTIL condition
  ```

```
Sub JackAndJillDo()

Dim CurrentStepsTaken As Integer
CurrentStepsTaken = 0

Dim FallenDown As Boolean
FallenDown = False

Do While FallenDown = False
    CurrentStepsTaken = CurrentStepsTaken + 1

    If Rnd() <= 0.3 Then
        FallenDown = True
    End If

Loop

Debug.Print CurrentStepsTaken & " step(s) up the hill were taken."
Debug.Print "Jack fell down the hill!"
Debug.Print "Jill came tumbling after..."

End Sub
```

```
Sub JackAndJillDoUntil()

Dim CurrentStepsTaken As Integer
CurrentStepsTaken = 0

Dim FallenDown As Boolean
FallenDown = False

Do Until FallenDown
    CurrentStepsTaken = CurrentStepsTaken + 1

    If Rnd() <= 0.3 Then
        FallenDown = True
    End If

Loop

Debug.Print CurrentStepsTaken & " step(s) up the hill were taken."
Debug.Print "Jack fell down the hill!"
Debug.Print "Jill came tumbling after..."

End Sub
```

More next month.

Power Pivot Principles

We continue our series on the Excel COM add-in, Power Pivot. This month, we look at how to create a calculated column.

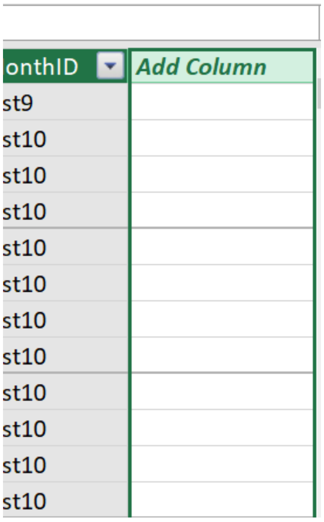
A **calculated column** is a simple formulaic calculation applied to an entire column that remains in the table you created it in. It can be later used by measures and other calculated columns to create more complex expressions.

Example

Let's go through an example to walk through how to add a calculated column into the Power Pivot model.

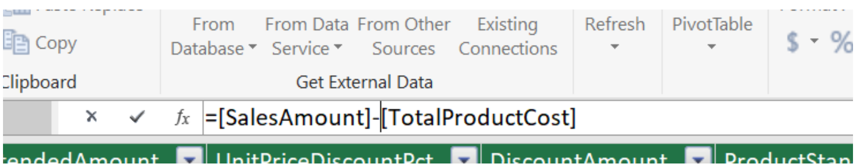
TotalProductCost	SalesAmount
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16
5.6	16

Let's say we want to add a column into the model which will calculate 'Gross Profit'. Next to the last column of data in your Power Pivot model will be a blank column with the title 'Add Column'.

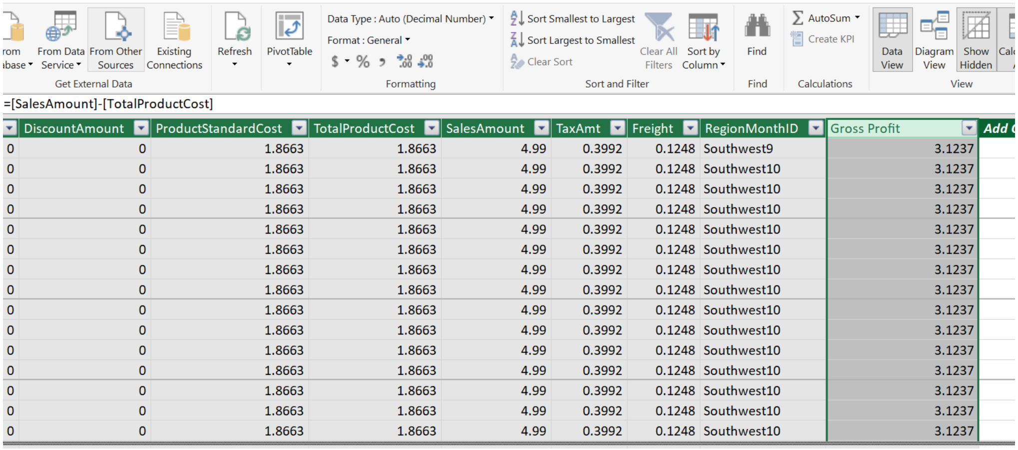


Double click on the heading and type ‘Gross Profit’, note, ensure all columns in your Power Pivot data model have a unique name. In the formula bar, you can type out the formula shown below or by selecting the columns of data you wish to use in the formula:

=[SalesAmount]-[TotalProductCost]



The new calculated column will appear in the data model as follows:



As this new calculated column contains a formula, it will be computed for each row in the data set. When the underlying data is refreshed the columns are recalculated each time.

This column may now be used in another calculated column, measure, chart, and even PivotTable like any other column in your data model.

Another useful calculated column to have might be a year column. To create this, add a new column, rename it ‘Year’ and use the **YEAR** function:

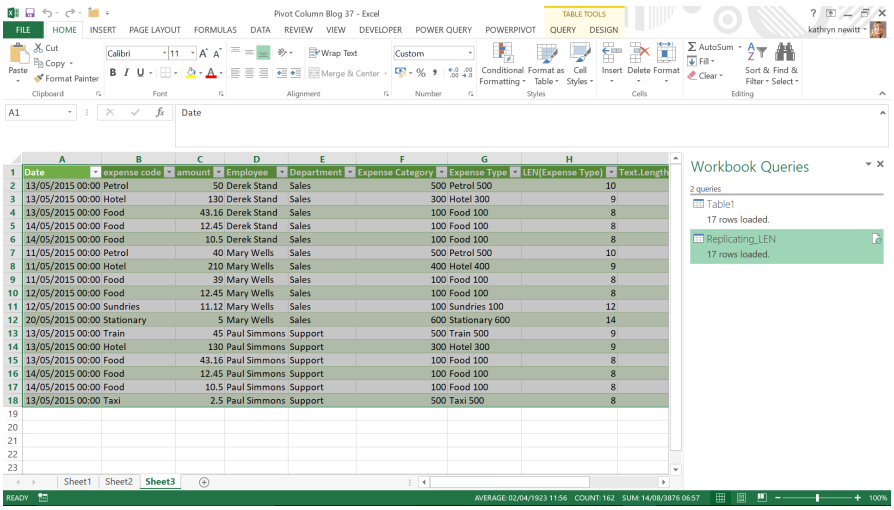
=YEAR(Sales[OrderDate])

More *Power Pivot Principles* next month.

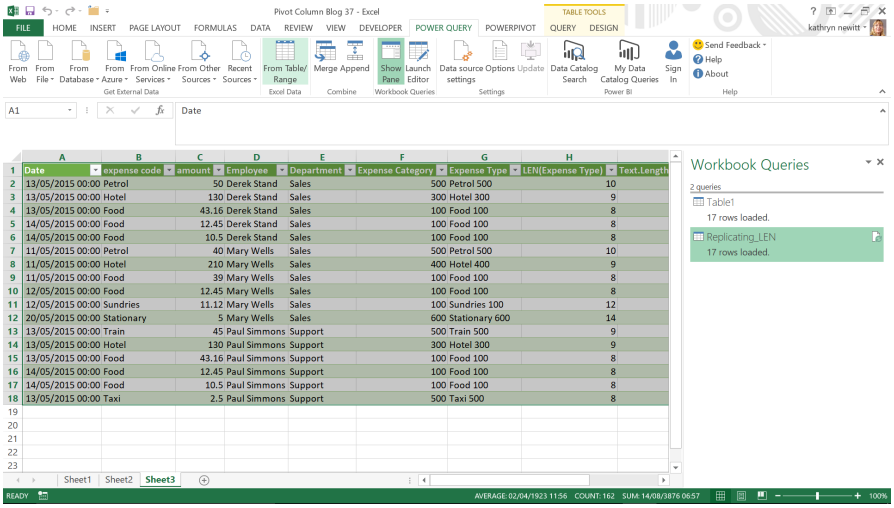
Power Query Pointers

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

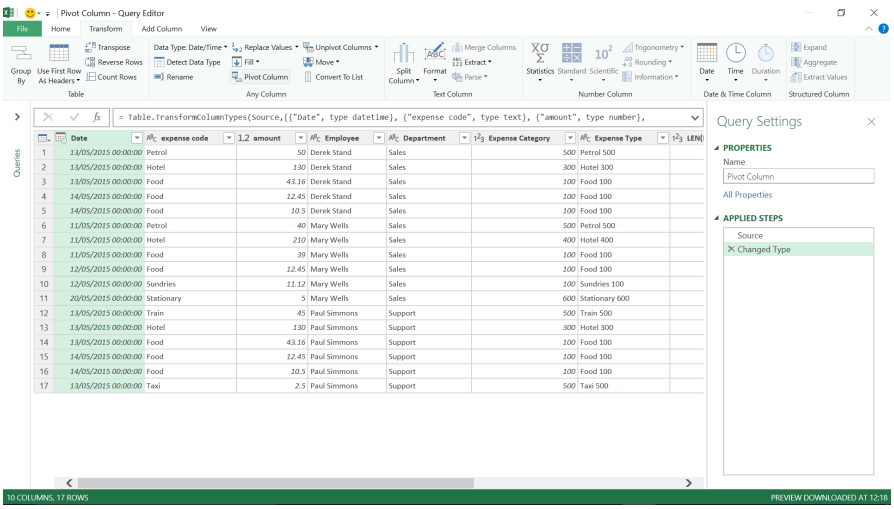
Power Query's ability to efficiently transform data is amply demonstrated by features which allow you to pivot (and unpivot) data at the touch of a button. Let's begin as usual with some data in an Excel worksheet.



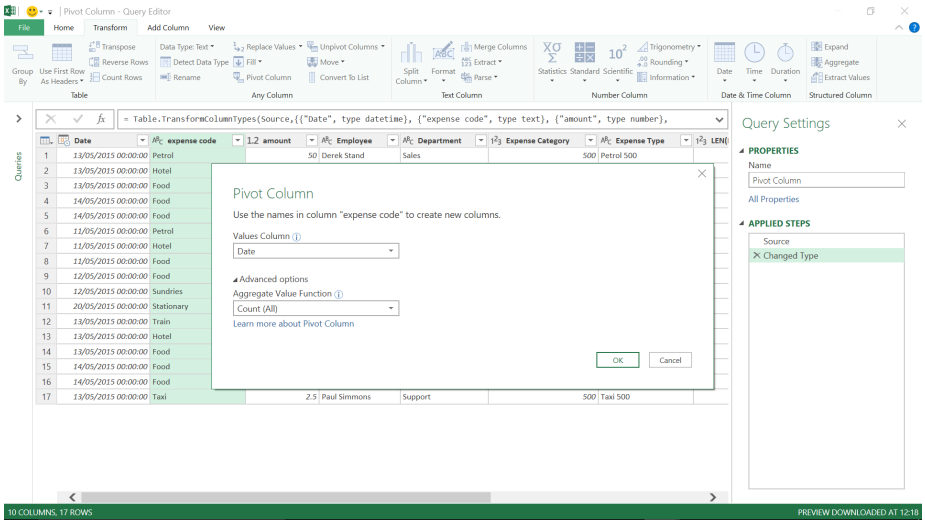
In the 'POWER QUERY' tab (assuming Excel 2013), in the 'Excel Data' section, we'll choose to extract the data 'From Table/Range':



Our data appears in a new query, which has been called 'Pivot Column'. In the 'Transform' tab, there is a section which groups together the transformations which can be applied to 'Any Column'. We are interested in the 'Pivot Column' option.

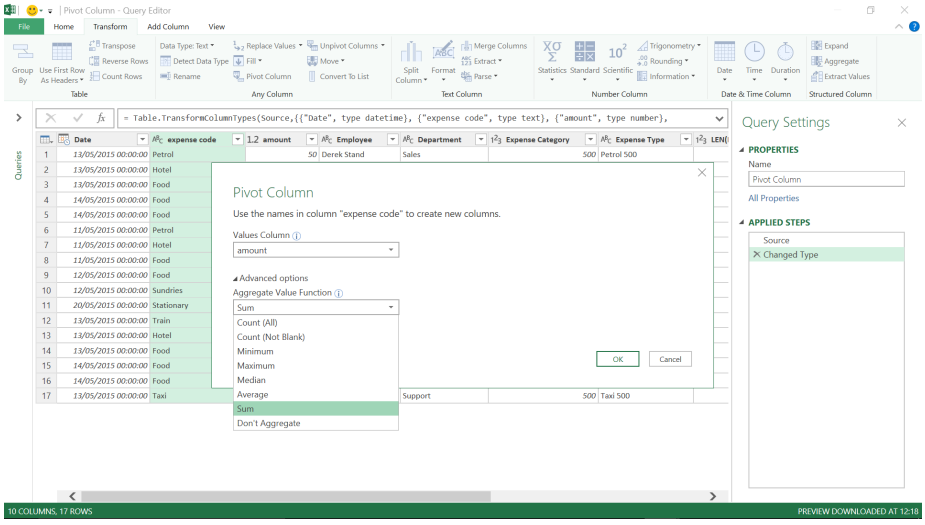


'Pivot Column' is available as long as you haven't selected multiple columns. You may either select a column before using 'Pivot Column' or it will assume you will want to use the first column in your query. We will select ***expense code***.

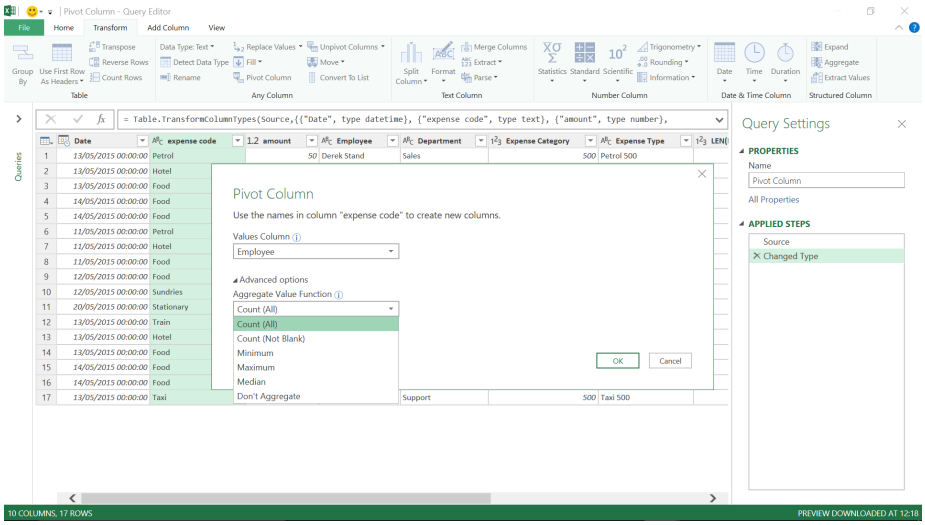


The 'Pivot Column' feature will remove the selected column and add columns to the table that contain an aggregate value for each unique value in that column. This is quite a concept to describe in one sentence, and it best demonstrated by an example!

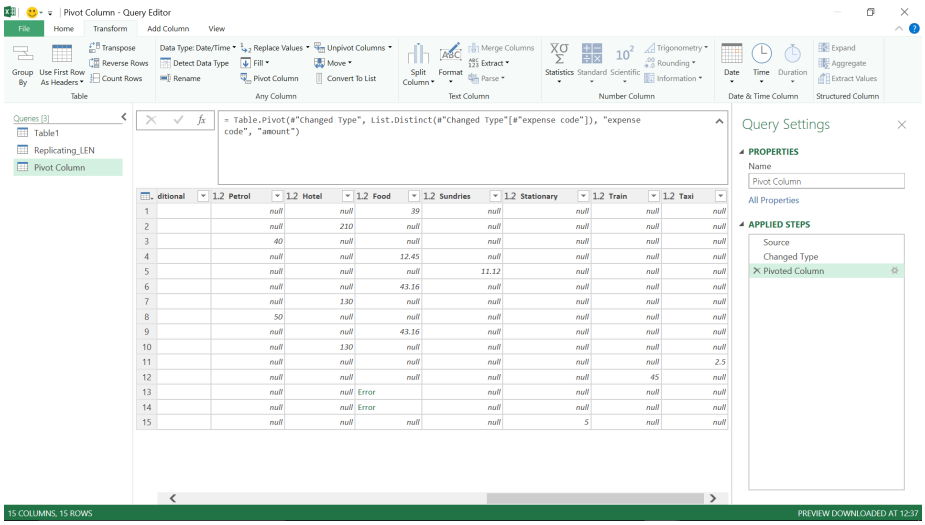
We have provided the names of the column (***expense code***), but we also need to specify what data will be aggregated, and the function associated with the aggregation. The data column will also be removed from our query when the new columns are added. We will be using the ***amount*** column as our data, which we have decided to sum *viz*.



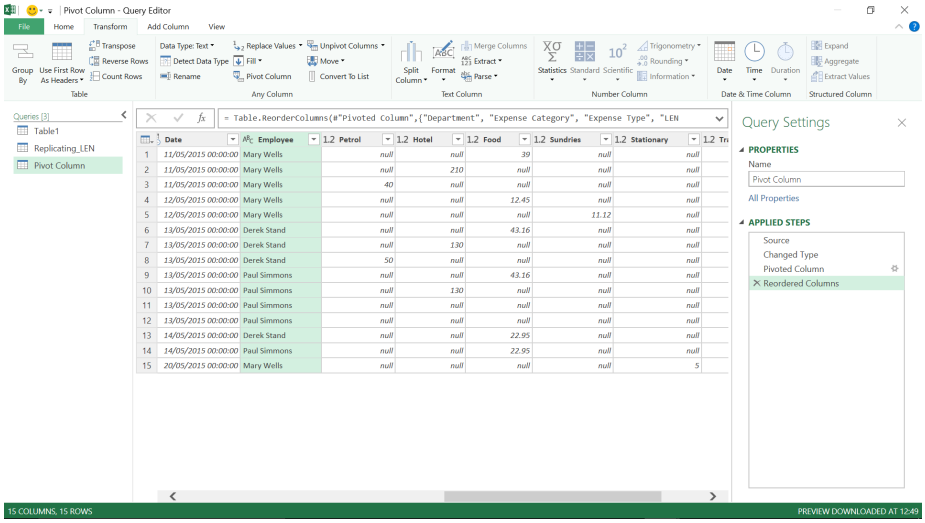
As soon as we pick ***amount***, Power Query recognises that we have chosen the datatype currency, and defaults to 'Sum'. The other options for the datatype are shown above. If we had chosen a text column, then the options would be different, as shown below:



Returning to our ***amount*** column, there is also an option 'Don't Aggregate'. This would return a grid which only has values when that particular expense code is populated, as shown below:

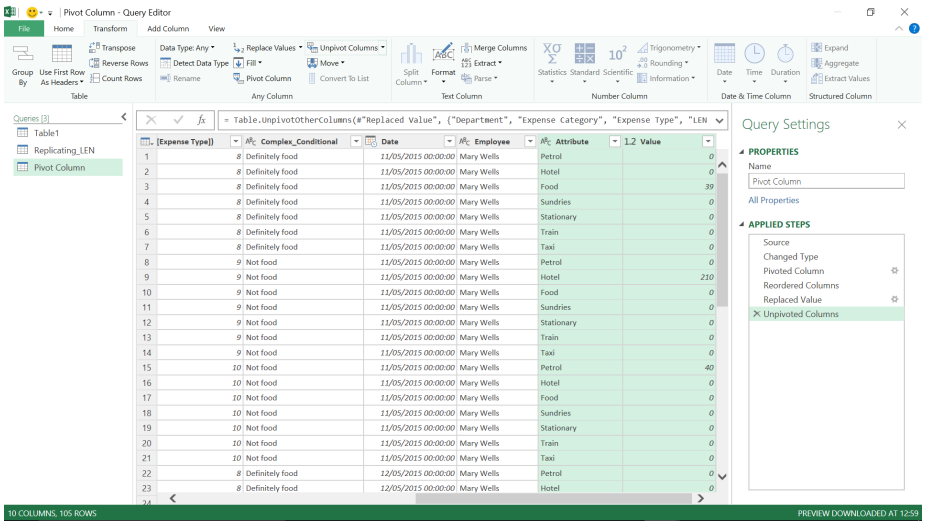


It didn't work too well when we had two consecutive food amounts, but the other amounts are shown. We are more interested to try aggregating by using the 'Sum' option, so let's delete this step and return to the 'Pivot Column' screen and try again…



This time our data is all populated as expected. We have already reordered the columns so that we can see who incurred what expense and on what date and we may tidy it up further by removing the nulls by using the 'Replace Values' facility.

Having demonstrated how easy it is to pivot columns, another nice feature to show is how easy it is to unpivot them again. If you select our new columns and choose to 'Unpivot' them using the option from within the 'Any Column' section in the 'Transform' tab, you will get this:



Our expense code (**Attribute**) and the amount (**Value**) have been reinstated!

More next month!

Power BI Desktop Update for September

Power BI Desktop received a makeover last month with its new, default theme! It's more than just redecorating though: Microsoft has doubled the number of built-in theme options for reports. Now, if you're a theme author, with the new updates to the theme JSON format makes it easier to create your own custom themes.

Another popular request was actioned in September too: Microsoft admits it was "…thanks to an intern project…", they are also releasing custom format strings.

Starting with this update, installation is now smoother and more streamlined with a single **.exe** installer that has all the languages wrapped into it. This means if you're automatically downloading it from the Download Center each month, you'll need to update your scripts.

Here's the complete list of September updates:

Reporting

- Color and text classes in themes
- New default themes
- 'Personalized Visualization' pane improvements (Preview)
  - Unpin default visuals
  - About info for visuals

Analytics

- Custom format strings
- Conditional formatting for more visual formatting options:
  - Alt text
  - Border colour
  - Gauge colours
- Drill through discoverability improvement
- New DAX expressions: **REMOVEFILTERS** and **CONVERT**

Visualizations

- PowerApps visual now Generally Available and certified

Data connectivity

- PostgreSQL connector enhancements:
  - Support for folding over Native Database queries
  - DirectQuery support Generally Available

Data preparation

- Copy to clipboard from data profiling

Template apps

- Google Analytics report

Other

- Performance improvements for multi-dimensional models
- Query performance improvements for DirectQuery models.

Let's take a look at each in turn.

Color and text classes in themes

When writing your own custom theme JSON file, it can be quite difficult to build a theme that changes the entire look of your report. For example, if you wanted to create a dark theme that set the page to a dark colour and use white text on top, you'd have to change many settings through the visual styles section of the JSON, which can be very time consuming and prone to errors.

To help with this, Power BI Desktop has expanded the color classes within the theme file to allow you to format many of the colours in your theme without needing to touch visual styles. If you're already writing theme files, you're likely already using a few existing color classes such as **foreground**, **background** and **tableAccent**, which update various settings within your report with one line. With this update, new color classes have been added to that list, which as a result, means you can colour all visual elements in a report just by setting six colours *(do you know how hard it is to check that "color" and "colour" are in the right place? – Ed.)*.

The color classes you now have available and what they format are as follows:

- **foreground**
  - Labels background colour (when outside data points)
  - Filter pane and Filter cards font and icon colours
  - Trend line colour
  - Textbox default colour
  - Table and matrix values and totals font colours
  - Data bars axis colour
  - Card data labels
  - Gauge callout value colour
  - KPI goal colour
  - KPI text colour
  - Slicer item colour (when in Focus mode)
  - Slicer dropdown item font colour
  - Slicer numeric input font colour
  - Slicer header font colour
  - Scatter chart ratio line colour
  - Line chart forecast line colour
  - Map leader line colour
- **foregroundNeutralSecondary**
  - Label colours
  - Legend label colour
  - Axis label colour
  - Table and matrix header font colour
  - Gauge target and target leader line colour
  - KPI trend axis colour
  - Slicer slider colour
  - Slicer item font colour
  - Slicer outline colour
  - Line chart hover colour
  - Multi-row card title colour
  - Ribbon chart stroke colour
  - Shape map border colour
  - Button text font colour
  - Button icon line colour
  - Button outline colour

- **foregroundNeutralTertiary**
  - Legend dimmed colour
  - Card category label colour
  - Multi-row card category labels colour
  - Multi-row card bar colour
  - Funnel chart conversion rate stroke colour
- **backgroundLight**
  - Filter card background colour for applied filters
  - Axis gridline colour
  - Table and matrix grid colour
  - Slicer header background colour (when in Focus mode)
  - Multi-row card outline colour
  - Shape fill colour
  - Gauge arc background colour
- **backgroundNeutral**
  - Table and matrix grid outline colour
  - Shape map default colour
  - Ribbon chart Ribbon fill colour (when match series option is turned off)
- **background**
  - Labels background colour (when inside data points)
  - Filter pane and available Filter card background colour
  - Slicer dropdown items background colour
  - Donut chart stroke colour
  - Treemap stroke colour
  - Combo chart background colour
  - Button fill colour
- **TableAccent**
  - Overrides table and matrix grid outline colour when present.

When writing your own custom theme JSON file, it can be quite difficult to build a theme that changes the entire look of your report. For example, if you wanted to create a dark theme that set the page to a dark colour and use white text on top, you'd have to change many settings through the visual styles section of the JSON, which can be very time consuming and prone to errors.

To help with this, Power BI Desktop has expanded the color classes within the theme file to allow you to format many of the colours in your theme without needing to touch visual styles. If you're already writing theme files, you're likely already using a few existing color classes such as **foreground**, **background** and **tableAccent**, which update various settings within your report with one line. With this update, new color classes have been added to that list, which as a result, means you can colour all visual elements in a report just by setting six colours *(do you know how hard it is to check that "color" and "colour" are in the right place? – Ed.)*.

In addition to color classes, text classes have also been added to make it easier and quicker to set text styles. There are four primary text classes that adjust a total of 14 total text classes. You can set the font family, font size and font color for each text classes. The four main text classes are: 'Title', 'Label, 'Callout' and 'Header'.

The Title and Label classes have several secondary classes that are automatically derived from the primary class settings, but they may be formatted individually. For example, if you set the Label class, which, for example, is used for the values in a table to 10pt, a 'Small Label', which is, for example, used in the search box text in slicers, would be automatically set to 9pt. The goal of the primary / secondary text classes is to make it quick and easy set the text classes, which still maintaining a visual hierarchy to the text.

The below table shows the primary classes with example settings and the secondary classes with the settings that make them unique compared to their associated primary class.

Primary class	Secondary classes	Class name in JSON	Settings	Associated visual objects
Callout	N/A	callout	DIN #252423 45pt	Card data labels KPI indicators
Header	N/A	header	Segoe UI Semibold #252423 12pt	Key influencers headers
Title		title	DIN #252423 12pt	Category axis title Value axis title Multi-row card title * Slicer header
	Large title	largeTitle	14pt	Visual title
Label		label	Segoe UI #252423 10pt	Table and matrix column headers Matrix row headers Table and matrix grid Table and matrix values
	Semibold	semiboldLabel	Segoe UI Semibold	Key influencers profile text
	Large	largeLabel	12pt	Multi-row card data labels
	Small	smallLabel	9pt	Reference line labels * Slicer date range labels Slicer numeric input text style Slicer search box Key influencers influencer text

Primary class	Secondary classes	Class name in JSON	Settings	Associated visual objects
	Light	lightLabel	#605E5C	Legend text Button text Category Axis labels Funnel chart data labels Funnel chart conversion rate labels Gauge target Scatter chart category label Slicer items
	Bold	boldLabel	Segoe UI Bold	Matrix subtotals Matrix grand totals Table totals
	Large and Light	largeLightLabel	#605E5C 12pt	Card category labels Gauge labels Multi-row card category labels
	Small and Light	smallLightLabel	#605E5C 9pt	Data labels Value axis labels

** The font color for these settings is based on the data colours, not the text class settings.*

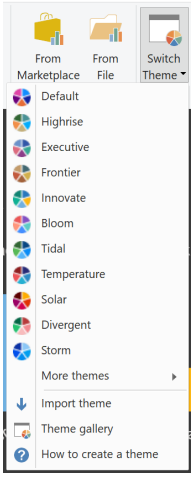
For example, here's a small theme that sets some of these properties:

```
{
    "name": "NewThemeOptions",
    "foreground": "#252423",
    "foregroundNeutralSecondary": "#605E5C",
    "foregroundNeutralTertiary": "#B3B0AD",
    "background": "#FFFFFF",
    "backgroundLight": "#F3F2F1",
    "backgroundNeutral": "#C8C6C4",
    "tableAccent": "#118DFF",
    "textClasses": {
        "callout": {
            "fontSize": 45,
            "fontFace": "DIN",
            "color": "#252423"
       },
        "title": {
            "fontSize": 12,
            "fontFace": "DIN",
            "color": "#252423"
        },
        "header": {
            "fontSize": 12,
            "fontFace": "Segoe UI Semibold",
            "color": "#252423"
        },
        "label": {
            "fontSize": 10,
            "fontFace": "Segoe UI",
            "color": "#252423"
        }
    }
}
```

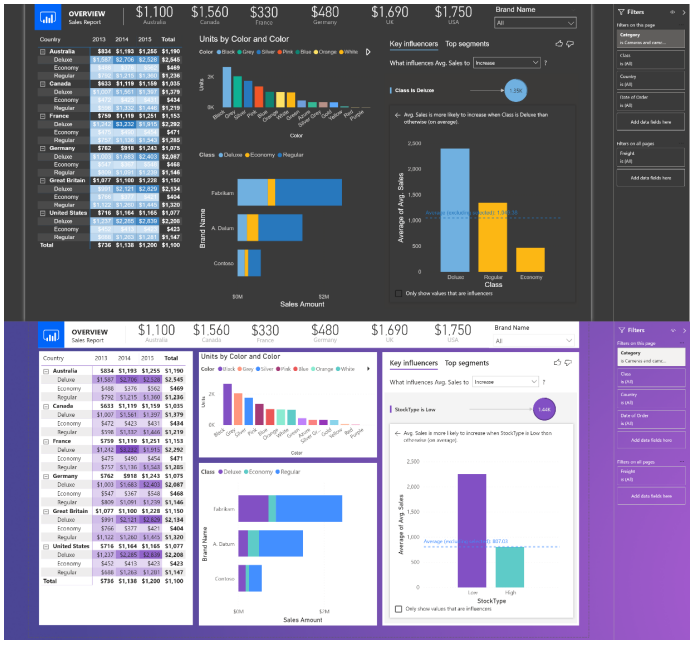
You can still use visual styles to set any specific formatting you want, and that will override any color or text classes that have been set. Microsoft expects with these new changes for most users to stick to the color and text classes the majority of the time, and only use the visual styles for non-color / text options (such as turning titles on / off by default).

New default themes

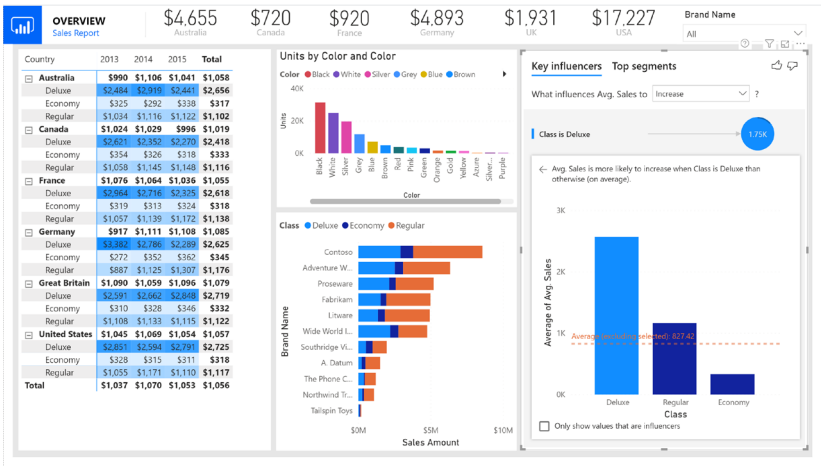
To go along with the new theme JSON options, Microsoft is updating the themes available for reports and changing the default theme for new reports.



Their Design and Research team spent time developing a new set of themes that have more variety (such as Innovate, a dark theme) and show off more theme-able feature (such as Bloom, with its background image):



The new default theme is meant to both align better with Microsoft's design language and follow best design practices for visuals.



Some visual updates with the new default theme are:

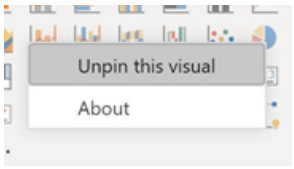
- Larger, darker, more readable text
- Smaller bubble sizes for scatter and map visuals
- Wider line strokes for line and combo charts
- Updated layout for pie and donut charts to improve readability
- Expand / collapse on by default for matrices
- Backgrounds on for visuals by default.

You can still find the previous themes under the 'More themes…' submenu any time you want to reach them. One thing to remember around custom themes is that they always build on top of the default theme. Therefore, if you created your own custom theme that just set the data colours and nothing else, if you imported that theme into a report with the new default applied, it would still have expand / collapse on by default, backgrounds on for visuals, *etc*. It may look different compared to when you imported it to a report with the old default theme. If you want your theme to look exactly the same way it did previously, you can first set the theme to 'Classic' in the theme dropdown, so the default theme has the old settings and then import your custom theme.

'Personalized Visualization' pane improvements (Preview)

UNPIN DEFAULT VISUALS

To go along with the new theme JSON options, Microsoft is updating the themes available for reports and changing the default theme for new reports.

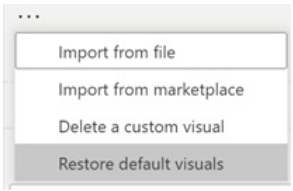


Once you unpin a built-in visual, it moves under the dotted line within the pane, and next time you open a report, it won't show unless you have that visual type used already within the report, in which case, it will show below the dotted line. This means that the only difference in experience between built-in and custom visuals is that the built-in ones initially show by default.

Some reasons you might wish to do this include:

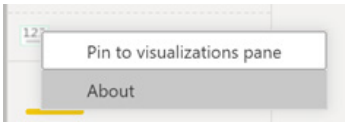
- You never use specific visuals and want to declutter the pane
- You don't require specific visual types and don't want to see it as an option
- You're using a custom visual version of a visual type and don't want to be mistakenly click on the built-in version you never use.

If you later want the default visuals back, you can use an option to restore the original visuals to the pane.

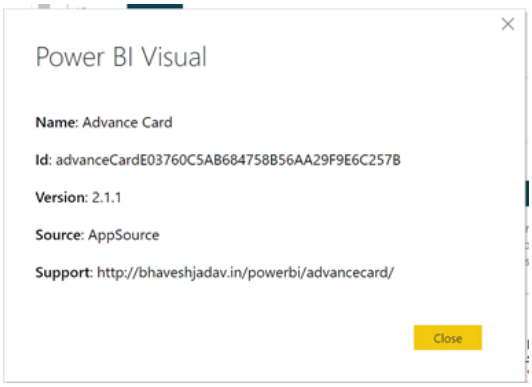


ABOUT INFO FOR VISUALS

Another update in this area is that you can now right-click on any visual in the Visualization pane to see an About dialog with more information on the visual.

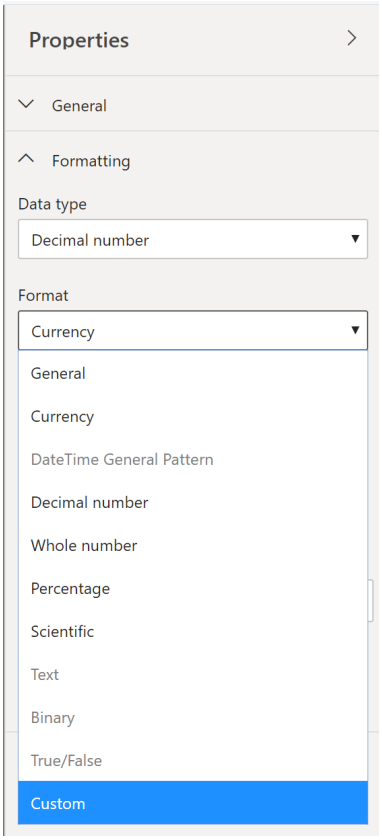


You'll be able to see things such as the visual's ID, version number, where you got the visual from, and the support information.

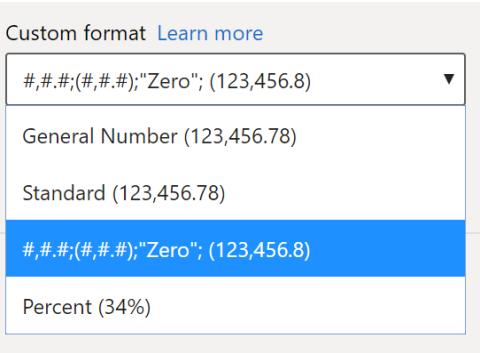


Custom format strings

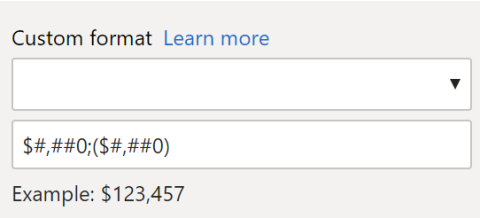
While in the Modeling view, you can now enter in a custom format string to customise how the field will show in visuals. To do this, first select the field in the Fields list and select the custom format option in the Formatting card of the Properties pane:



From there, you'll be able to select from a list of commonly used format strings.



You may use this input box to enter in your own format string:



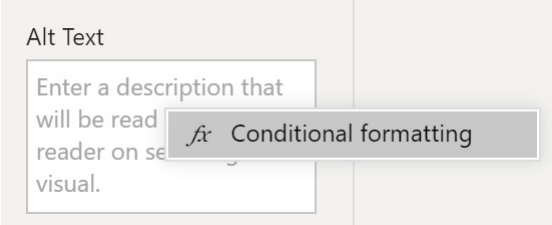
Once you enter in your custom format string, it will immediately be reflected in the Data view and Report view.

Conditional formatting for more visual formatting options

There are more formatting features in this update that may be conditionally formatted, moving Power BI Desktop ever closer to the goal of everything within the Formatting pane being conditionally formattable.

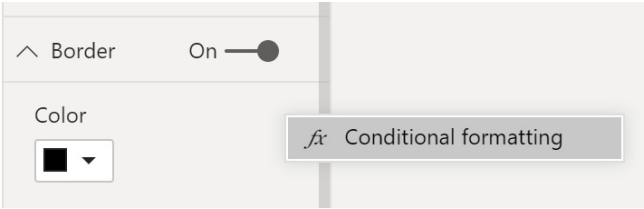
ALT TEXT

Alt text is the first feature that now has conditional formatting. This is a positive step forward in terms of accessibility. Alt text, which is read off by screen readers whenever the visuals are in focus, can now be dynamically changed based on the current filter state of the visual. For example, you may call out in the alternative text both the highest and the lowest categories and have confidence it will always reflect the current data in the visual. Now that's cool!



BORDER COLOUR

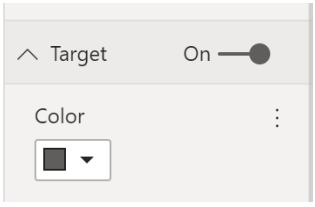
Border color is the next feature that has also added this capability. Just as you can with backgrounds, you can now dynamically adjust the border colour based on current filter state. This could be an option to use if you want to colour a visual based on a specific KPI but disagreed with the background colour that appeared by default.



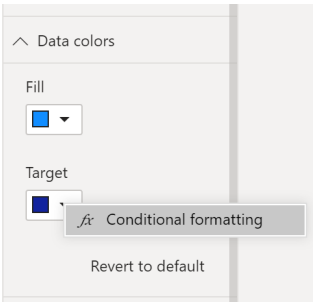
GAUGE COLOURS

Back in June, Microsoft added conditional formatting to 'fill color for gauge', and this update sees this expanding to four other gauge properties as well:

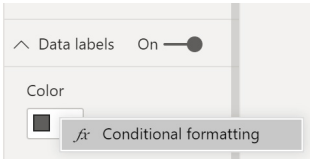
- Target text color



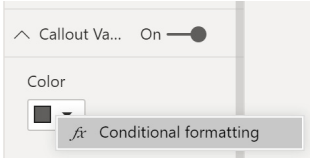
- Target fill color



- Data label color

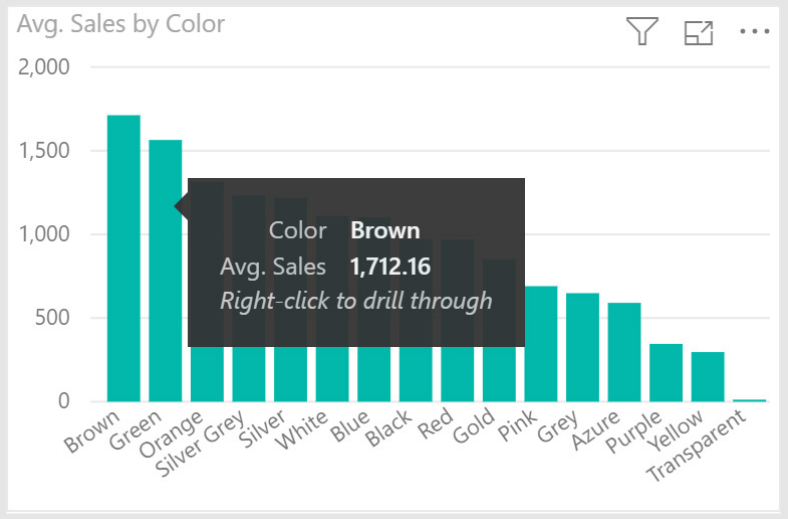


- Callout value color



Drill through discoverability improvement

Often, end users don't know when they may drill through on a visual to another report page. To help with this, Power BI Desktop has now added text to the visual's ToolTip to let them know it's enabled:



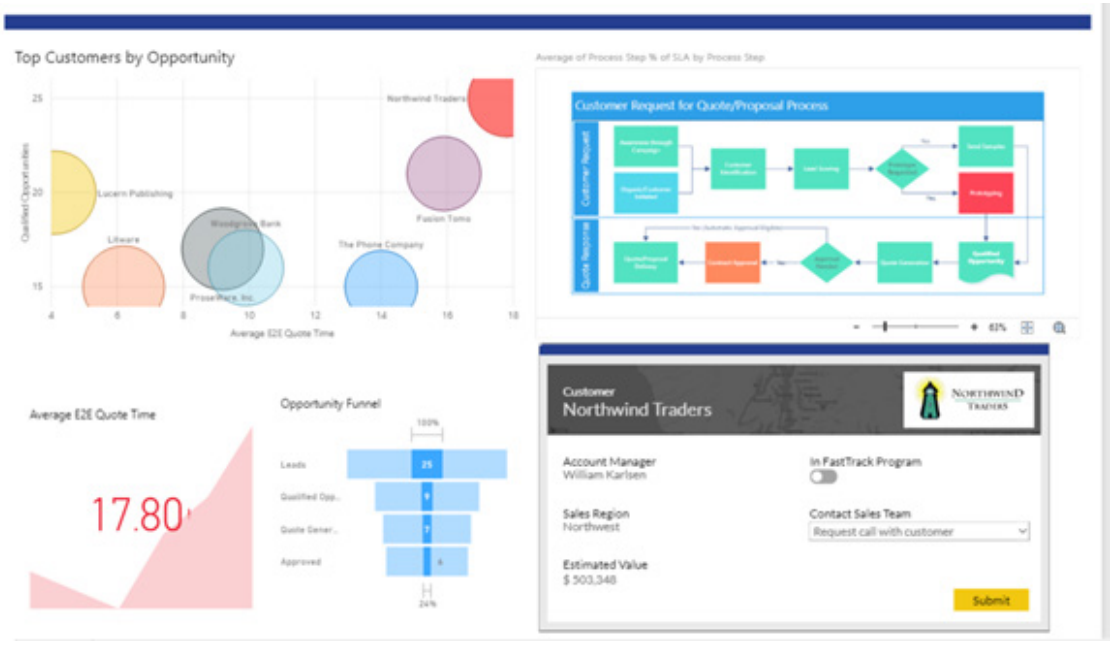
New DAX expressions: REMOVEFILTERS and CONVERT

There are two new DAX expressions this month:

- **REMOVEFILTERS** is the same as **ALL** when used inside **CALCULATE** to remove filters
- **CONVERT** changes the expression to another data type.

PowerApps visual now Generally Available and certified

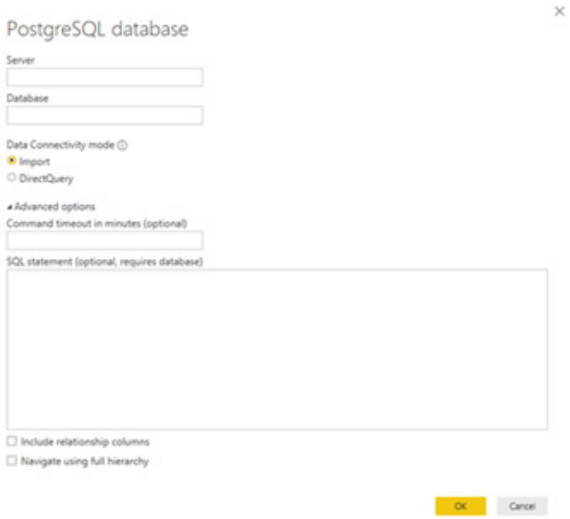
The PowerApps marketplace visual allows you to add the power of applications to your reports and dashboards. The PowerApps visual brings forms and data editing directly to Power BI by allowing you to embed your own PowerApps right in your report or dashboard. As of this update, the visual is now Generally Available and certified, which means you may use the visual in even more places.



PostgreSQL connector enhancements

SUPPORT FOR FOLDING OVER NATIVE DATABASE QUERIES

The PostgreSQL connector has been enhanced with query folding over a native query. You can now paste in a native query when connecting to a PostgreSQL database, with folding capable operations applied on top according to normal Import or Direct Query logic.

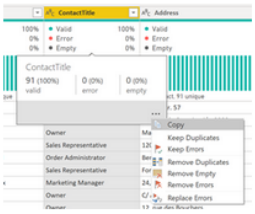
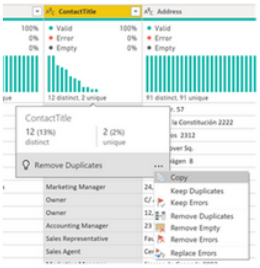
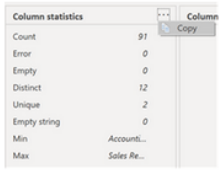
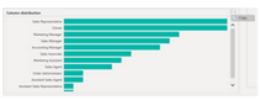


DIRECTQUERY SUPPORT GENERALLY AVAILABLE

The DirectQuery support within the PostgreSQL connect that was announced last month now becomes Generally Available.

Copy to clipboard from data profiling

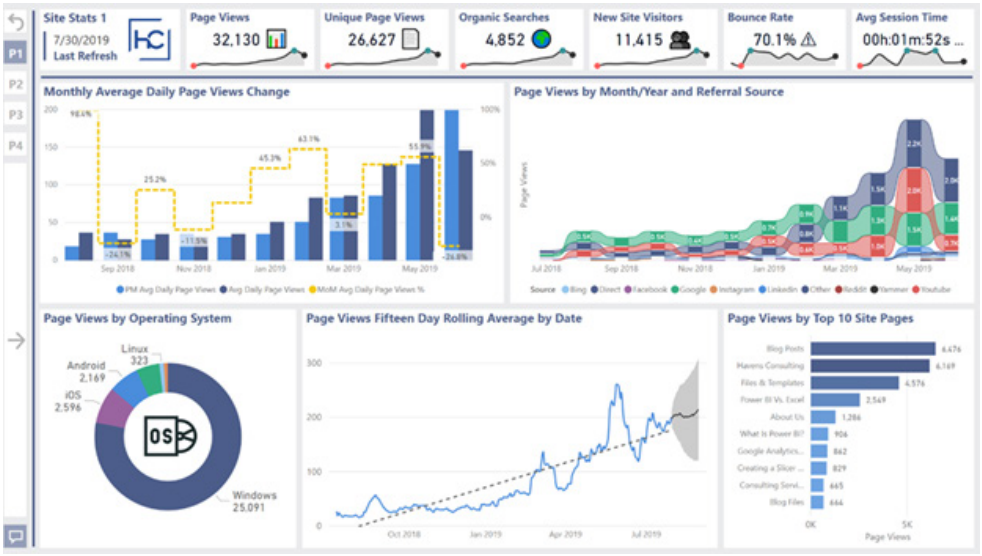
With this update, it is now possible to export data profiling information by copying it to the clipboard from the Power Query Editor. The new copy option is available from all Data Profiling surfaces as described below:

Data Profiling Surface	Screenshot	Data exported to clipboard
Column Quality Summary		Valid 91 100% Error 0 0% Empty 0 0%
Column Distribution charts		Sales Representative 17 Owner 17 Order Administrator 2 Marketing Manager 12 Accounting Manager 10 Sales Agent 5 Sales Associate 7 Sales Manager 11 Marketing Assistant 6 Assistant Sales Agent 2 Assistant Sales Representative 1 Owner/Marketing Assistant 1
Column Profile Details Pane – Column statistics		Count 91 Error 0 Empty 0 Distinct 12 Unique 2 Empty string 0 Min Accounting Manager Max Sales Representative
Column Profile Details Pane – Column distribution		Sales Representative 17 Owner 17 Order Administrator 2 Marketing Manager 12 Accounting Manager 10 Sales Agent 5 Sales Associate 7 Sales Manager 11 Marketing Assistant 6 Assistant Sales Agent 2 Assistant Sales Representative 1 Owner/Marketing Assistant 1

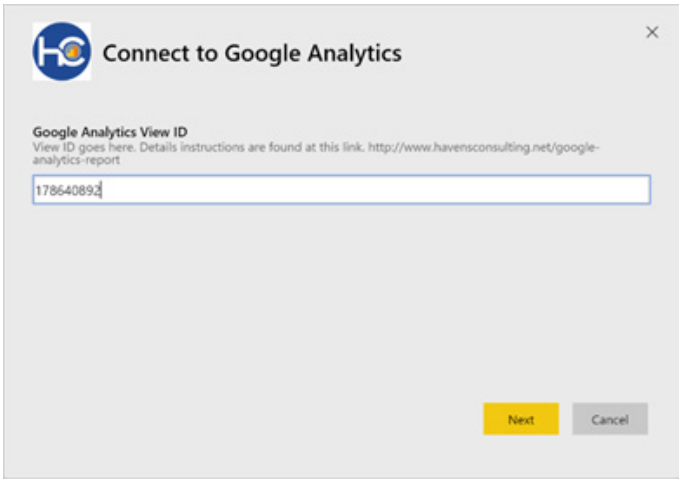
Google Analytics report

Template applications are a great way to provide Power BI users with installable, ready to use applications. Whilst they are not always exactly what you may want, these templates contain reports, dashboards and data models that can connect to your data, creating immediate value and action.

The Google Analytics report app is one of the more popular templates and is produced by Havens Consulting. The report model was built using the Google Analytics data connector. It can be connected to your own Google Analytics data using your own credentials.



The report itself enables users to perform a deep-dive analysis on daily page views with forecasting and adjustable rolling average, month-over-month traffic gain or loss, top pages visited, traffic source trends over time, and more. Furthermore, the report contains many useful examples of Power BI features and functionality including bookmarks, what-if parameters, advanced conditional formatting and custom visuals.



You can install the app right in the Power BI Service and set its parameters to your Google Analytics view ID. Once you install it, the report is now yours, so you can customise and share as much as you want.

Performance improvements for multi-dimensional models

This update sees general performance improvements for multi-dimensional users of Analysis Services 2019 (we'll pretend we know what this means). For those so affected, when using the new Release Candidate of SQL Analysis Services 2019, you'll see more performant queries and optimization of measure execution. This improvement, which some may know as "Super DAX", helps reduce the "chattiness" between Power BI and Analysis Services.

Some other features that come with this are multi-dimensional support for expand / collapse on row headers, cross-highlighting support across different tables, and high density sampling for visuals that support it. You will need to enable this setting in Analysis Services to take advantage of it in Power BI.

Query performance improvements for DirectQuery models

This update also introduces runtime constant folding to improve query performance for some DirectQuery models. The amount of performance improvements you'll see will vary depending on the DAX expressions you're using, the model structure, speed of the DirectQuery source, etc.

More next month we're sure!

Power BI Mobile and Service latest updates

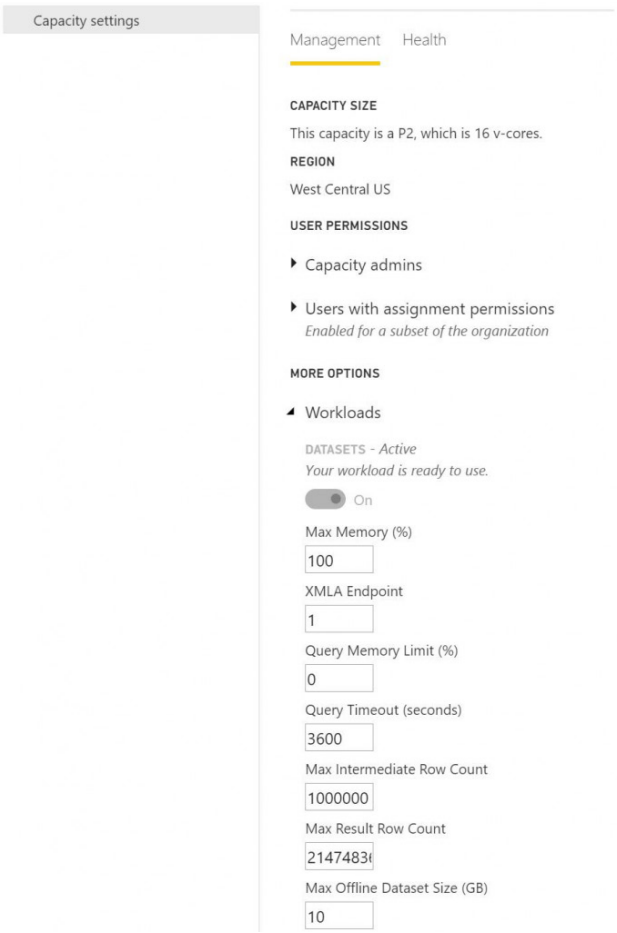
One month I am going to sit here and write, "nope, nothing happened this month". But it's not going to be this month. There are still more updates for Power BI Mobile and Service:

- New capacity settings for Power BI Premium
- Custom branding for your organisation
- Update for On-premises data gateway
- Summarised data export with build permission
- URL parameters for paginated reports
- Support for monthly e-mail subscriptions
- AAD app proxy integration with mobile.

Let's go through them.

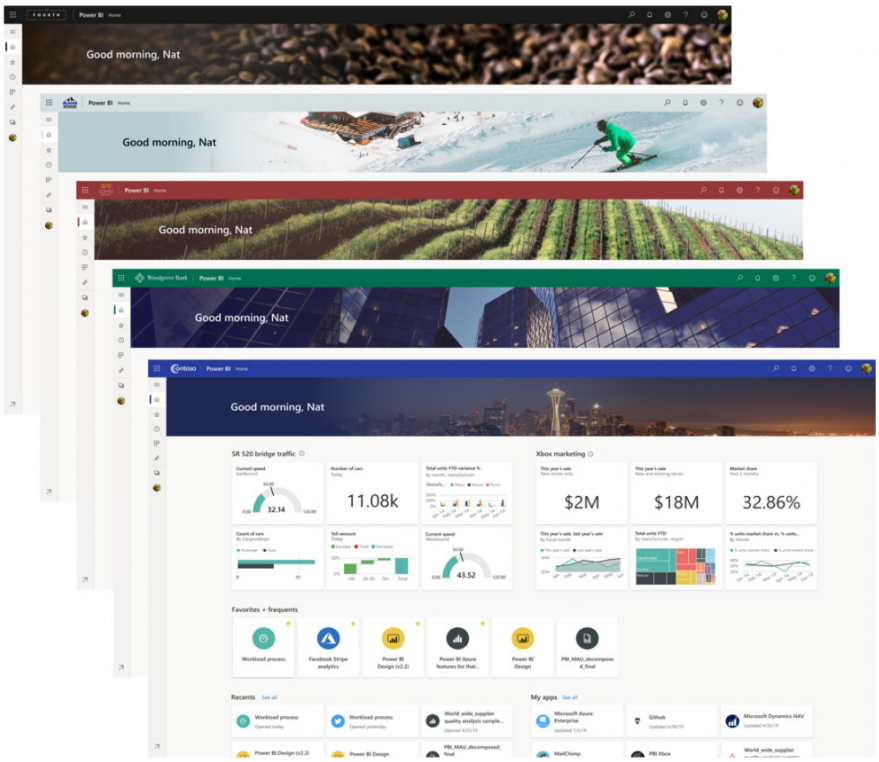
New capacity settings for Power BI Premium

There are five new settings for your Premium capacities in the admin portal. Now, you can set limits on "Max Offline Dataset Size" with the allowable range from 0.1 to 10 GB and even set limits on queries to prevent noisy reports from impacting others using the capacity. You can use these new settings to create a more predictable and performant capacity for your organisation.



Custom branding for your organisation

There is now custom branding for the Power BI Service. With this update, you change the theme colour that appears in the top navigation bar, add your company logo, and modify your default landing page by adding a cover image.



Update for On-premises data gateway

Now, there's a new, improved version of the on-premises data gateway together with an updated mashup engine to match the Power BI Desktop version.

Summarised data export with build permission

It has long been recognised that the limitations on export of summarised data from a visual are overly restrictive. Therefore, Microsoft has adjusted their approach to allow users with the Read permission to export summarised data from a visual. Please note that exporting the underlying data for a visual will continue to require Build permission. The underlying data is the detailed rows that were rolled up to show in the visual.

URL parameters for paginated reports

This update brings in the ability to use URL parameters for paginated reports. Now, report authors may send commands to paginated reports in Power BI by adding a parameter to a URL. For example, you can pass report parameters to a report by including them in a paginated report URL, and even construct this URL dynamically in a Power BI report and drillthrough to a paginated report by using a DAX measure.

Support for monthly e-mail subscriptions

Microsoft has continued to add more granular controls to e-mail subscriptions in the Power BI Service by adding an option for you to subscribe monthly to reports, dashboards and paginated reports.

AAD app proxy integration with mobile

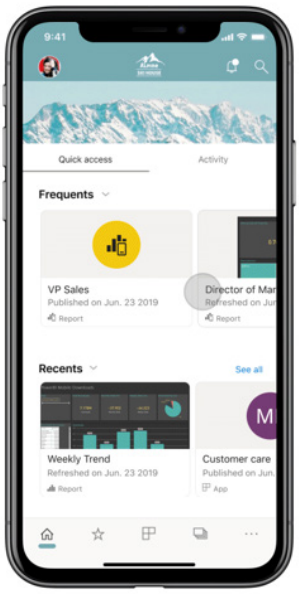
The Power BI team has partnered with the Azure Active Directory team to integrate Power BI Mobile applications with Azure Active Directory (AAD) Application Proxy. With this feature, you can now connect to Report Server hosted within your orgranisation from Power BI Mobile app, without the need to set up complex on-premise configurations.

That's it for this month – more next time, we're certain!

Power BI Mobile apps gets a new look (Preview)

Power BI Mobile provides the tools you need to stay connected to your data, wherever you may be. However, as organisations grow, it becomes increasingly difficult for Power BI users to discover and manage the content that matters to them on the go. Microsoft has indeed recognised that users must be able to find exactly what they need quickly and easily when they're using the Power BI Mobile app to view and interact with their content.

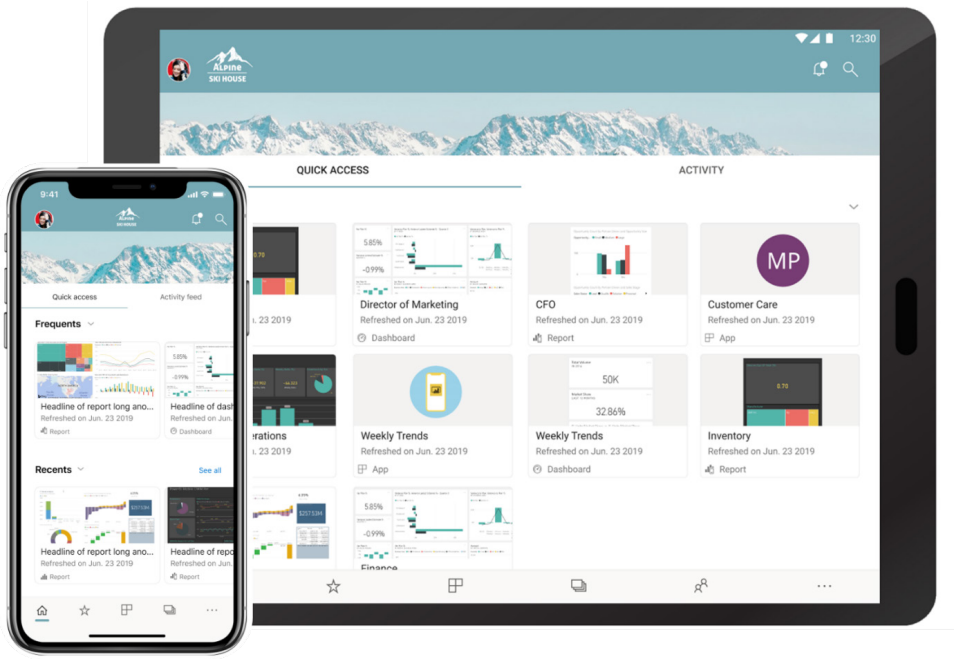
With this borne in mind, last month saw the public Preview of the 'new look' now available on Power BI Mobile. Now, there is a home page that provides quick access to your commonly used content and includes your organisation brand theme and new navigation bars that give you a simpler way of navigating through the app.



As this is a Preview, you'll need to opt in and turn it on in the mobile app. Once you do that, the app will display several new features:

- Home page
- Organisation branding
- Navigation bar.

Here's an example of how the new home page shows on the mobile app:



At the top, the app will show your organisation's branding, so it will have same look and feel as your Power BI Service to match your corporate theme. The organisation brand elements are defined by your Power BI administrator in the Power BI Service, and will show up in your Power BI Mobile app as well.

In addition, the home page provides two tabs:

- **Quick access:** here, you'll have a way to view and find your most commonly accessed items in Power BI. You'll be able to view the content cards that provide easy access to that content, in addition to metadata for the content itself, such as its last refresh time. You may look for your items in the Frequents section, where you'll see a list of items ordered by the number of times you viewed them; you can use the Recents list to view the items that you've recently accessed
- **Activity** *(coming soon):* here, you'll receive a feed with events happening in your Power BI account that are relevant to you. The feed ensures you can stay up to date with the latest comments and activities.

Microsoft has re-built the app navigation, implementing a more standard mobile navigation experience to make it easier to find your way in the Power BI mobile app. Now, instead of using the top left 'hamburger' menu, you have a bottom navigation bar, providing you with an easy way to swap between pages and use your preferred way to find your content.

From the top bar you can access the notification center and search for Power BI content. If you want to connect (or switch) to Report Server or access the app setting page, just tap on your account avatar. Simple.

It's noted this may not appeal to all. During the Preview period the new look experience will require you to turn it on explicitly, so you have time to become accustomed. There are several ways you can opt-in to the new look Preview. You may turn it on from the introduction banner or, if you have closed the banner, you can opt in from the side panel or from the app settings.

The Preview period allows us to get your input and feedback on our new design, learn from it and improve. During this period, Microsoft will continue to add new capabilities and *might* adjust the experience, based on your feedback. You have been warned!!

The A to Z of Excel Functions: DELTA

Probably not a function you are going to use every day of the week. Or any for that matter. This (sort of) replaces the equals operator ("=") in that it gives it a "Goodrem" for its money… Essentially, this function tests whether two values are equal, returning 1 if **number1** = **number2**, otherwise it returns 0 (zero). You may use this function to filter a set of values. For example, by summing several **DELTA** functions you calculate the count of equal pairs. This function is also known as the **Kronecker Delta function**.

The DELTA function employs the following syntax to operate:

DELTA(number1, [number2])

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

- **number1**: this is required and represents the first number
- **number2**: this argument is optional. This is the second number. If omitted, **number2** is assumed to be zero.

It should be further noted that:

- If **number1** is non-numeric, **DELTA** returns the *#VALUE!* error value
- If **number2** is non-numeric, **DELTA** returns the *#VALUE!* error value.

Please see our example below:

	A	B	C
1	Formula	Description	Result
2	=DELTA(7,-7)	Checks whether 7 equals -7.	0
3	=DELTA(0,0)	Checks whether 0 equals 0.	1
4	=DELTA(0.99999999,1)	Checks whether 0.99999999 equals 1.	0
5			

The A to Z of Excel Functions: DEVSQ

This function returns the sum of squares of deviations of data points from their sample mean. This is a key component of such statistical calculations as standard deviation.

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

DEVSQ(number1, [number2], ...)

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

- **number1, number2, ...: number1** is required, subsequent numbers are optional. You may have between one and 255 arguments for which you want to calculate the sum of squared deviations. You can also use a single array or a reference to an array instead of arguments separated by commas.

It should be further noted that:

- arguments can either be numbers or names, arrays, or references that contain numbers
- logical values and text representations of numbers that you type directly into the list of arguments are counted
- if an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included
- arguments that are error values or text that cannot be translated into numbers cause errors
- the equation for the sum of squared deviations is:

$$\text{DEVSQ} = \sum (x - \bar{x})^2$$

Please see the example below:

	A	B	C
1	Data	Deviation from Mean	Deviation Squared
2	4	2	4
3	5	1	1
4	8	2	4
5	7	1	1
6	11	5	25
7	4	2	4
8	3	3	9
9			
10		SUM	48
11			
12			

13	Formula	Description	Result
14	=AVERAGE(A2:A8)	Average	6
15	=DEVSQ(A2:A8)	Sum of squares of deviations from their sample mean.	48
16			

More Excel Functions next month…

Upcoming SumProduct Training Courses

Location	Course	Date	Duration
Sydney	Power Pivot, Power Query and Power BI	9 - 11 Oct 2019	3 Days
Sydney	Excel Tips and Tricks	4 Nov 2019	1 Day
Sydney	Financial Modelling	5 - 6 Nov 2019	2 Days
Sydney	Power Pivot, Power Query and Power BI	11 - 13 Nov 2019	3 Days
Melbourne	Excel Tips and Tricks	6 Nov 2019	1 Day
Melbourne	Financial Modelling	7 - 8 Nov 2019	2 Days
Melbourne	Power Pivot, Power Query and Power BI	9 - 11 Dec 2019	3 Days
Sydney	Power Pivot, Power Query and Power BI	9 - 11 Dec 2019	3 Days
Sydney	Excel Tips and Tricks	16 Dec 2019	1 Day

Sydney	Financial Modelling	17 - 18 Dec 2019	2 Days
Melbourne	Excel Tips and Tricks	13 Jan 2020	1 Day
Melbourne	Financial Modelling	14 - 15 Jan 2020	2 Days
Sydney	Power Pivot, Power Query and Power BI	15 - 17 Jan 2020	3 Days
Sydney	Excel Tips and Tricks	27 Jan 2020	1 Day
Sydney	Financial Modelling	28 - 29 Jan 2020	2 Days
Sydney	Power Pivot, Power Query and Power BI	17 - 19 Feb 2020	3 Days
Sydney	Excel Tips and Tricks	2 Mar 2020	1 Day
Sydney	Financial Modelling	3 - 4 Mar 2020	2 Days
Melbourne	Power Pivot, Power Query and Power BI	9 - 11 Mar 2020	3 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 would provide a reminder of the function keys, this time with **SHIFT**:

Keystroke	What it does
SHIFT + F1	What is... (Help)
SHIFT + F2	Insert / edit comment
SHIFT + F3	Function wizard
SHIFT + F4	Find next (from most recent search)
SHIFT + F5	Find dialog
SHIFT + F6	Previous Pane
SHIFT + F8	Add to Selection Mode
SHIFT + F9	Calculate sheet
SHIFT + F10	Activate context menu (right-click)
SHIFT + F11	Insert new worksheet
SHIFT + F12	Save

There are over 540 keyboard shortcuts in Excel. For a comprehensive list, please download our Excel file a www.sumproduct.com/thought/keyboard-shortcuts. Also, check out our new daily **Excel Tip of the Day** feature on the www.sumproduct.com homepage.

Our Services

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

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

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

Link to Others

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

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

Any Questions?

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

Training

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

Check out our more popular courses in our training brochure:

Drop us a line at training@sumproduct.com for a copy of the brochure or download it directly from 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, Level 9, 440 Collins Street, Melbourne, VIC 3000
Registered Address: SumProduct Pty Ltd, Level 6, 468 St Kilda Road, Melbourne, VIC 3004

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