Spreadsheet skills: seeking order out of chaos

February 2012

This month, we present a possible solution to our Christmas challenge – and announce the winners. By Liam Bastick FCMA, CGMA, director with SumProduct.

Recap
You may recall December’s challenge was based on a real life common problem that modellers find themselves up against on a regular basis: how to reorder input data automatically.

I provided the following example dataset:
Readers were asked to reorder the list, according to the following sorting rules:

- surname first, alphabetically from A to Z
- forename second, again from A to Z
- date of birth third, from oldest to youngest
- occupation fourth, alphabetically from A to Z
- ranking fifth, from highest to lowest.

Ties were allowed, but the reordered list needed to have the same number of entries as the original list.

I illustrated how the solution to the above dataset might look:

**Figure two: reordered data**
This list had to update automatically whenever data was revised / inserted into the input table. There were restrictions too: in constructing the solution:

- no macros, array (CTRL + SHIFT + ENTER) or user defined functions were to be used
- the output must update automatically as data was changed (so utilising Excel’s sorting features was not acceptable, for example).

Many readers picked up the gauntlet and our three lucky winners will be contacted by email.

**SUMPRODUCT solution**
You’re probably thinking this is some shameless attempt at plugging the company name, but as July 2009’s article demonstrated, SUMPRODUCT() is a highly versatile Excel function.

However, before using it to solve our puzzle, we have another issue to resolve: the names have to be split so that the surname may be considered first when sorting. To do this and calculate the appropriate order, we need some ‘helper columns’, which was the approach most entrants took.

My logic was as follows (click on the image for an expanded version):

**Figure three: helper columns**
As you can see from the illustration above, I have added six additional columns to assist with the ordering in columns adjacent to the original dataset (in my illustration in columns L to Q).

I will now go through the rationale and formula for each.

**Trimmed name**

The first column ensures my logic will work. I assume that all names in the unordered list are of the format ‘forename surname’. Therefore, I check that there are no additional spaces in the names typed in column F, the input section for names (above).

The TRIM() function removes all spaces from text except for single spaces between words. I will use it to ensure no irregular spacing between names and / or to remove any accidental spacing after the surname (touch typists sometimes make this mistake for example).

In my example, I therefore use the formula

=TRIM(F12)

to tidy up the names in column F.

Further (convoluted?) checks could be included to ignore middle names, non printing characters from text, and so on. My intention here is to keep the example simple and I note no-one went to this length in their competition entries, so I’m not going to either!

This clean up of data now allows me to proceed to the next step.

**Space posn**

To be able to break the name in half so to speak, I need to find the position of my delimiter (the element in the text string that determines the boundary between separable data).

Here, I have two data elements (forename and surname) separated by a consistent delimiter, a space (‘ ’). Therefore, I need to find the position of this character in the text string in column L (Trimmed Name).
If all names were of the format ‘forename surname’ the formula would be easy, viz.

=FIND(" ",L12)

The FIND(find_text,within_text,[start_num]) function syntax has the following arguments:

- Find_text the text you want to find
- Within_text the text containing the text you want to find
- Start_num an optional argument which specifies the character at which to start the search. The first character in within_text is character number 1. If you omit start_num as I did in my example above, it is assumed to be 1.

However, in my example dataset, row 26 contains the name 'Adele'; a name I have, er, made up for a musician. Thus, I need a slightly more sophisticated formula:

=IF(ISERROR(FIND(" ",L12)),LEN(L12)+1,FIND(" ",L12))

This is a variant on the above FIND formula, using the ISERROR() function. ISERROR(value) returns a value of TRUE if value (which could be a cell reference or a formula for example) refers to any error value (#N/A, #VALUE!, #REF!, #DIV/0!, #NUM!, #NAME? or #NULL!).

Professional modellers will often use ISERROR as follows:

=IF(ISERROR(calculation),error_trap,calculation)

ie calculation will be performed unless it returns an error, in which case, an alternative calculation or value will be used, which is the error_trap. Readers will see this is what I have done above with calculation being FIND(" ",L12) (as explained earlier) and error_trap being LEN(L12)+1 in this instance.

Since LEN(text string) returns the length of a text string, my error_trap here will return the value of (length of the text string in column L) plus one. In summary, this calculation returns the position of the (first) space character in a text string (if it occurs) otherwise it returns the value of the length of the text string plus one. This is needed to split our text string.

Readers using Excel 2007 or later may be aware of the IFERROR() function, created by Microsoft in recognition of the common usage of IF(ISERROR) syntax. I have not discussed it here as this new function is incompatible with earlier versions of Excel.

**Forename**

Column N in my example creates the forename from the text string in column L thus:

=LEFT(L12,M12-1)

The function LEFT(text,num_chars) returns the first num_chars characters (i.e. the first num_chars ‘left-most’) in the text string text. For example, =LEFT(‘Liam’,3) returns ‘Lia’, whereas =LEFT(‘Liam’,7) returns only ‘Liam’.
Here, this formula returns all the characters to the left of the space character, or the entire text string if there is no space character.

Surname
Column O in my example presents the remainder of the name (ie the surname as follows):

=RIGHT(L12,MAX(LEN(L12)-M12,0))

The function RIGHT(text,num_chars) returns the last num_chars characters (i.e. the first num_chars ‘right-most’) in the text string text. For example, =RIGHT(‘Liam’,3) returns ‘iam’, whereas =RIGHT(‘Liam’,7) returns only ‘Liam’.

Here, this formula returns all the characters to the right of the space character (or nothing if there is no space character). MAX() is required in the expression MAX(LEN(L12)-M12,0) since this will return a negative value if there is no space character (negative numbers are not allowed for use with the RIGHT() function).

Raw ranking
Readers of a nervous disposition may wish to look away now. Column P has one of those formulae that gives model users – and auditors alike – nightmares:

=SUMPRODUCT(--(O12>$O$12:$O$31))
+SUMPRODUCT((N12>$NS$12:$NS$31)*(O12=$OS$12:$OS$31))
+SUMPRODUCT((G12>$GS$12:$GS$31)*(N12=$NS$12:$NS$31)*(O12=$OS$12:$OS$31))
+(ROWS($PS$12:$PS12)/ROWS($PS$11:$PS31))

It looks absolutely horrible, but it’s not quite as bad as it may appear upon first glance. Let me explain. I have deliberately broken it down to make it easier to understand. SUMPRODUCT() is discussed at length in July 2009’s article and is very useful here.

Let’s consider the first line of the formula:

=SUMPRODUCT(--(O12>$O$12:$O$31))

One of the competition’s constraints was that no array formulae (ie those entered using CTRL + SHIF + ENTER and appear in braces ‘{‘ and ‘}’ were allowed. However, the array needs to be considered for ordering purposes, so I use what is often referred to as a ‘pseudo-array function’, namely SUMPRODUCT().
The expression \((O12>$O$12:$O$31)\) provides an array of TRUE’s and FALSE’s depending on whether \(O12>O12\), \(O12>P12\), \(O12>Q12\), etc. Adding ‘--’ works as follows. The first minus sign converts TRUE’s to -1 and FALSE’s to zeroes. The second minus ensures all values are non negative (ie -1 is converted to +1 and 0 remains 0). Adding \(SUMPRODUCT()\) simply totals this array, ie it provides the number of items in column O strictly less than the value in cell O12.

Let me work through an example. The value in cell O12 is ‘Broke’. When I talk about ‘strictly less than’ with text I mean all text strings which would be ordered ahead of ‘Broke’ if ordered alphanumerically in ascending order. There are four values in column O ahead of ‘Broke’: the empty string for Adele in cell O26, ‘Adder’ in cell O13, ‘Again’ in O17 and ‘Bond’ in cell O25. This expression would therefore return a value of 4, with ‘Nasser Broke’ ranked fifth according to our ordering rules.

Our second ordering rule requires us to sort alphabetically on forename if there is a ‘tie’ on surname. Consider the second line of our formula from hell:

\[+SUMPRODUCT((N12>$N$12:$N$31)*(O12=$O$12:$O$31))\]

By similar logic to that described above, this counts the number of values in column N (the forename column) that are ahead of N12 alphanumerically which share a common surname. In other words, it provides the secondary sorting factor.

Similarly, the third line,

\[+SUMPRODUCT((G12>$G$12:$G$31)*(N12=$N$12:$N$31)*(O12=$O$12:$O$31))\]

sorts on date of birth when entries share the same forename and surname. Notice that ‘>’ sorts alphanumerically in an ascending order, whereas ‘<’ sorts alphanumerically in descending order, which is what was required for the fifth factor, Ranking, which caught a lot of entrants out:


The penultimate row,


counts on how many occasions the five sorting factors appear in the dataset. For Nasser Broke (row 12), these precise details only occur once, so 1 is added to the 4 calculated earlier, giving a value of 5 – which would be Nasser’s position in the order.

The reason for this penultimate row is to allow for duplicates. If I had a list A, B, B and C (say) and sorted alphabetically using the logic so far described these items would be ranked first, joint third, joint third and fourth respectively.

This isn’t quite good enough, however, which is why we include the final line:
This row adds a ‘tiebreaker’ value greater than zero but less than one simply depending on where an item was positioned in the original list. This ensures all raw rankings are unique, which is necessary for the final ranking column (column Q).

**Final ranking**
The formula gets simpler now! This assigns a ranking on column P as follows:

= RANK($P12,$P$12:$P$31,1)

A fuller explanation of the RANK() function can be found in the suggested solution to our world cup challenge from 2010.

I am now in a position to create the reordered list, which will update automatically.

**Figure four: reordered list**

![Reordered list table]

This table is generated using just one formula, namely:

=INDEX(F$12:F$31,MATCH(ROWS($F$38:$F38),$Q$12:$Q$31,0))

which uses an old favourite combination of this column. A full discussion of how INDEX(MATCH) works can be found in my March 2009 article.

There are plenty of other ways to tackle this problem many of us are faced with on an almost daily basis. I am not exaggerating when I was inundated with many differing solutions, most of which used my helper columns approach. That reminds me…
Competition winners
Congratulations to the winners of our challenge. Your Amazon vouchers will be winding their way to you shortly!

The normal articles resume next month. Please keep sending in the inspiration for these articles in the meantime.

If you have a query for the spreadsheet skills section, please feel free to drop Liam a line at liam.bastick@sumproduct.com or visit the SumProduct website.

Links
Excel extras: PowerPivot web applications in action
Professional development
CIMA on demand