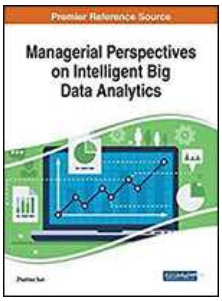

Managerial Perspectives on Intelligent Big Data Analytics

by Zhaohao Sun

Chapter 7: Using Excel and Excel VBA for Preliminary Analysis in Big Data Research

Paul John Blayney,
University of Sydney,
Australia

Zhaohao Sun,
Papua New Guinea University of Technology,
Papua New Guinea
https://orcid.org/0000-0003-0780-3271

ABSTRACT

Can big data research be effectively conducted using spreadsheet software (i.e., Microsoft Excel)? While a definitive response might be closer to "no" rather than "yes," this question cannot be unequivocally answered. As spreadsheet scholars, the authors' inclination is to answer in the positive. To this regard, the chapter looks at how Excel can be used in conjunction with other software and analytical techniques in big data research. This chapter also argues where and how to use spreadsheet software to conduct big data research. A focal argument of this chapter is that the key behind big data driven research is data cleansing and big data driven small data analysis. The proposed approach in this chapter might facilitate the research and development of intelligent big data analytics, big data analytics, and business intelligence.

1 INTRODUCTION

Trained programmers are competent with programming methods and design. They are not generally proficient in Big Data analysis (Raffensperger, 2001, p. 62). On the other hand, Big Data researchers do not necessarily possess programming skills. Spreadsheet software (e.g. Excel) provides a means for the non-programmer to conduct analysis that could previously only performed by a trained programmer or analyst.

Most Big Data analysts will be skeptical with the use of Excel for Big Data analysis. They will highlight that the volume, velocity and variety of Big Data far exceeds the capacity of spreadsheet software (Sun, Sun, & Strang, 2018). They're right from several perspectives. However, this chapter attempts to address the following research questions?

1. Have spreadsheets a place in Big Data research?
2. How can programming with Excel VBA contribute?
3. How can the Excel Power Pivot add-in contribute?

This paper does not suggest that spreadsheet software can replace the advanced analytical techniques used for Big Data analytics. However, it does propose that spreadsheets have a place in Big Data in the same way that the apps on your phone have a place in your everyday life. Modern day apps are useful because they're easy to use and readily available. They provide you with useful information on a real-time basis (i.e. when you don't have time to investigate properly or talk to an expert).

The research demonstrates that Excel (especially when supplemented with its Power Pivot add-in) can fill the same role in Big Data analytics. Spreadsheet analysis can provide valuable insights as to what advanced analytics are appropriate.

For example, preliminary testing can be conducted on a subset of the 50 terabytes (TB) of web server logs that an Internet Service Provider CEO wants looked at for the latest trends in customer demands for the company's products (Department of Communication and the Arts, 2018).

Skilled use of Excel will allow better use of the Big Data analyst scarce resource; that valuable analyst time is not wasted exploring "futile" data (i.e. data without significant relationships).

A word of caution is warranted prior to using spreadsheet software for preliminary Big Data (e.g. small data) analysis. While the benefits (in business and other) provided by spreadsheet use are substantial and immeasurable; the costs of spreadsheet errors are also huge and well publicised. For example, see Butler (2018).

To this regard, frightening or entertaining reading (depending on your point of view) is provided by the European Spreadsheet Risks Interest Group (EuSpRIG, n.d.-b). This non-profit organisation of academics and business professionals proclaim their website (www.eusprig.org/) as "*the World's premier site for information, action, conferences and dialogue on Spreadsheet*

Risk Management". One of the links provided on the EuSpRIG homepage is "Horror Stories" (EuSpRIG, n.d.-a).

Many of the spreadsheet errors cited in these stories can be argued to be human error. However, it can also be ascertained that spreadsheet software has been largely responsible for enabling the human error to take place. As elucidated by Ray Panko (1998) most spreadsheets contain errors - "the issue is how many errors there are, not whether an error exists". Therefore, the task for the Big Data analyst is to apply standard organisational programming development principles to their use of spreadsheets.

Much of what Plauger (1993) writes about programming is equally applicable to Big Data analysis. The protocols and regimes described by Plauger are not unique to computer programming. These conventions are general principles that can and should be followed for the successful execution of any complex task; be it organising your daughter's 21st birthday party or conducting Big Data analysis. The Big Data analyst is well advised to follow Plauger's protocols at all times, including the performance of small data analysis with spreadsheet software.

The primary goal of this chapter is a Big Data analyst's version of Plauger's (1993), p. 7) objective with computer programs. It hopes to add the Excel spreadsheet and Excel VBA to the Big Data analysts' repertoire.

> "My goal in writing this premiere essay is to convince you that no one tool is best … to introduce you to the many tools … show you where you can use it … show you where it is not at its best or where you should not use it at all."

Plauger (1993, p. 63) reinforces the above assertion later in his book with further support for a multi-tooled analytical approach.

> … can't see beyond their favourite tools. What is needed, clearly, is a healthy assortment of methods, plus some guidelines for choosing the right one at the right time.

Excel's low set-up cost for the performance of complex analysis is a double-edged sword. While untrained analysts can produce accurate results with extraordinary efficiency they may also generate output with significant errors with similar efficacy. These analysts will extoll the virtues of spreadsheet use given the ease with which they can adapt their model to deal with new data types or changing user requirements / requests. However, the spreadsheet's adaptability has a downside as a model will tend to grow in complexity and suffer a degradation of structure and integrity.

The Big Data analyst must be aware of not falling into the common trap with spreadsheet use. That is, to have their "scratch pad" of spreadsheet calculations grow into a complex model used to generate important analytics. And because it was a scratch pad proper development protocols have not been followed

This paper provides a description of procedures and methods to achieve the best of both worlds. That is, a world where the analyst does not require months or years of training with analytical procedures and design methodology. The "Excel trained Big Data analyst" will possess skills that emulate the competencies of a trained specialist and eliminate many of the errors that non-programmers tend to make.

The remainder of this chapter is organised as follows. The second section provides an overview of how small data analysis can contribute to Big Data analytics. Section 3 proffers an outline of good spreadsheet methods and the use of Excel's built-in analytical and statistical functions. This section includes Power Pivot. The fourth section delivers an introduction to Excel VBA (Visual Basic for Applications) – the programming language secreted behind the Excel worksheet.

> However, data analytics is a new form of data analysis empowered by the current information communication technology (ICT) and Web technology. (Sun, Sun, & Strang, 2018) (Sun & Wang, 2017)

Many recognise Excel as one of the software applications traditionally used for financial analysis (Sun, Sun, & Strang, 2018).

> …data analysis has been in the field of business and management for decades. For example, it has been offered as a course with Microsoft Excel or IBM SPSS in undergraduate programs of business for decades worldwide …

2 BIG DATA DERIVED SMALL DATA ANALYSIS

Big data derived small data analysis is important both for a big data approach and for big data analytics as a discipline. There are manifold reasons. First of all, big data has basically been controlled by global data giants such as Facebook, Google, Tencent, Baidu and Alibaba rather than by individual scholars. It is expensive for a scholar to collect and analyse Big Data. It can also be very expensive for a company like Cambridge Analytica to collect data working together with Facebook, e.g. Cambridge Analytica paid a big price through its bankruptcy (Baker, 2018).

Secondly, sampling is the process of collecting some data when collecting it all or analyzing it all is unreasonable (National Research Council, 2013, p. 120). Sampling is a component of any statistical modelling process. The implication from this is that

the majority of statistical inferences are based on reasoning from incomplete knowledge or data. Therefore, any statistical modelling or inference is a kind of big data derived small data analysis and reasoning (National Research Council, 2013, p. 120).

From a data processing viewpoint, the largest data analyses will be performed in large data centres of a few global data monopolies running specialized software such as Hadoop over HDFS to harness thousands of cores to process data distributed throughout the cluster (National Research Council, 2013, p. 55). This means that individuals must use big data derived small data analysis to analyse big data.

3 SECTION I: THE ART OF USING EXCEL

This section provides a general description of procedures and methods that are applicable for any serious use of spreadsheet software. Various Excel features that are especially applicable for Big Data analysis are described. A discussion of Excel's Power Pivot add-in is included in this section while Excel VBA (Visual Basic for Applications) is the topic of Section 4.

3.1 Basic Spreadsheet Design and Procedures

Spreadsheet model development is like traditional programming in more than one respect (Walkenbach, 2010, p. 111). Formula creation in Excel is a type of programming.

> Virtually every successful spreadsheet application uses formulas. In fact, constructing formulas can certainly be construed as a type of "programming". (Walkenbach, 2002, p. 37)

The typical first step in the software development process is the recognition that all programs can be reduced to IPO [software engineering] - the programming acronym for:

- Inputs
- Processing (steps required to generate the desired outputs)
- Outputs

While spreadsheet models are sometimes criticised by traditional programmers for combining the processing and output components (and thus confusing matters), the IPO model is still a useful starting point in structured systems analysis (Pressman & Maxim, 2014).

Good data is critical for all spreadsheet models. However, expert analysts concentrate their efforts on the structure of their model, not with the search for and analysis of data. Novice analysts will frequently spend a lot of time searching for and analysing data. Better or more accurate data doesn't necessarily mean that the model's outputs will be better. It is essential to remember that modelling is about forecasting the future. Data is past tense (i.e. history) which may be useful for predicting future events but maybe not (Sun, Sun, & Strang, 2018). The analyst should spend most of their time developing an appropriate model structure (e.g. accurate and logical formulas). And less time pursuing that elusive "perfect" data.

3.2 Cell Referenced Formulas (No Hard Coding)

The hard coding of input data or constants into spreadsheet formulas is widely recognised as poor model design (van der Aalst, ter Hofstede, & Weske, 2003, p. 15). However, the importance of avoiding such design defect errors appears to be underestimated as there is not an immediate error and there may never be one. The fact that many models fail to follow this basic software design principle is almost certainly a major contributor to the incidence of spreadsheet errors and the lack of reusability of many models.

Powell and Baker (2004, p. 97) provide an excellent explanation of why it is important to isolate input data or parameters.

> A common source of errors in spreadsheets is the tendency to bury parameters in cell formulas and to replicate the same parameter in multiple cells. This makes identifying parameters difficult, because they are not immediately visible. It's also difficult to know whether all numerical values of a parameter have been changed each time an update is required. By contrast, the habit of using a single and separate location considerably streamlines the building and debugging of a spreadsheet.

The problem of hard coding of values into formulae has an equivalence in traditional programming and VBA where a well-constructed program would rarely (never?) enter a value (a number) into a line of executable code. The recommended programming technique is to assign the desired value to a named variable (e.g. RedFont = 3) and use this variable in the

program code (Pressman & Maxim, 2014). This can be contrasted to the inferior design that would simply type the value 3 into a line of code. The prevalence of such inferior design practices is caused by the practice being quicker in the short run as the developer avoids to create a variable and assign a value to it. The resulting 'magic number' in the code makes programs more difficult to read, understand, and maintain".

3.3 Logical Functions: IF's, Nested IF's, AND's AND OR's

The Formulas tab provides access to Excel's Function Library – an extensive collection of mathematical, financial, statistical and other ready-made functions. This chapter introduces the extremely powerful and versatile group of Logical functions.

3.4 Modularisation

Powell and Baker (2004) provide a standard set of steps for engineering a spreadsheet model in the typical recommended layout. The basic steps recommended by these authors is use of the concept of modularisation to separate data (inputs), decision variables, detailed calculations (processing) and outcome measures (output). They emphasise the importance of isolating input data or input parameters in a specific location in the model (e.g. a designated range of cells for small models or a separate data worksheet in a large model).

Figure 1: Formula tab function library

Modularisation is an appropriate solution with complex modeling problems because the individual components (modules) are easier to understand and to work with than the whole. The process of modularisation also requires the analyst to plan the overall project and then to focus effort on one area at a time. The experienced analyst will also find that some modules can be used in other models with only minor changes. This reusability of modules can result in huge savings in development time as the experienced analyst doesn't have to start from scratch even with a totally novel modeling job.

The concept of breaking up a complex model into smaller, self-contained bits can also be applied to individual spreadsheet formula (Walkenbach, 2010, p. 54). Poor spreadsheet technique attempts to all components of a complicated formula in a single cell. It is much easier and better to do preliminary calculations in separate cells (Powell & Baker, 2004, p.97). It is good spreadsheet practice to break up your complicated formulas into many simple ones. It is much easier to "debug" such calculations. Debugging is also greatly assisted by a model that has separate areas (e.g. worksheets) for various components of the model.

Reusability of code is certainly one of the basic concepts of traditional programming that is partially achieved by the separation of data inputs from processing and outputs (Pressman & Maxim, 2014).

The idea of breaking up a complicated task into smaller chunks might be seen simply as common sense (Pressman & Maxim, 2014). However, this concept is also strongly supported by considerable research in education in cognitive load theory.

Cognitive load theory identifies the key component of a task's complexity as element interactivity; being the degree to which the task depends on another factor(s) that must be considered simultaneously (Sweller, 2010). Such interactivity is intrinsic to the task and cannot effectively be altered. However, the separation of the overall task into a few less complex components will lessen the difficulty of understanding the overall task.

The applicability of cognitive load theory in this context of this paper is illustrated by Van Merriënboer and Krammer's (1987, p. 256) description of the outcomes provided using a structured programming language.

> Especially well-structured languages should make a top-down approach feasible because they facilitate breaking down large programming tasks into smaller subtasks.

3.5 The IF Function

The IF function allows you to ask a question about a value or label on your spreadsheet (e.g. is cell C5 greater than 50) and return a certain result if True (e.g. Pass) or another result if False (e.g. Fail). You must of course get the syntax correct. You can use the IF function dialogue box provided by **Formulas -> Logical -> IF** as shown in Figure 2 or you can simply enter your

IF function by typing directly to the desired cell as you would for any other function.

Start with ***=IF(*** and then the function arguments. Proper syntax for the example described above is ***=IF(C5>50,"Pass","Fail")*** Note the commas separating the "question" or "logical test" C5>50, the True value "Pass" and the False value "Fail". Also note the opening & closing parentheses and the double quotations around the Pass and Fail results.

3.6 Demonstration of Logical Function With the Balance Sheet

A simple use of the IF function is a check that the balance sheet is in balance. Figure 3 has used the IF function dialogue box to enter such a check. My preference is to type my IF functions directly to cells. However, the dialogue box has a couple of features that you may like.

As you provide entries for each of the three IF function arguments (i.e. Logical test, True value and False value) you are provided with an indicator of the result of your entry. E.g. the Logical test returns a True result as the Total Net Assets value of $128,973 in cell J33 is equal to the Total Owners' Equity value in cell J38.

The True value is set to return the contents of cell H42 which contains the text string "*Great. Balance sheet is in balance"*. The False value has been entered as a text entry to the dialogue box. An indication of the result of the IF function is provided. E.g. *Great. Balance sheet is in balance*. This is what will be displayed in cell I32, the cell where the formula shown here has been entered (via the dialogue box).

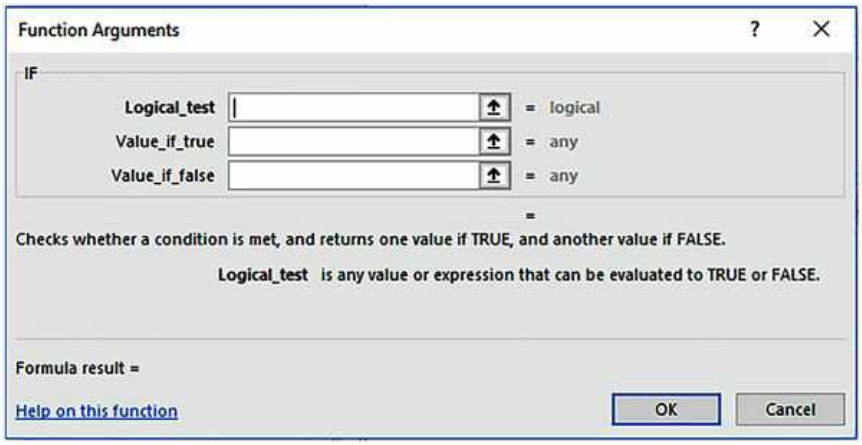

Figure 2: IF function dialogue box

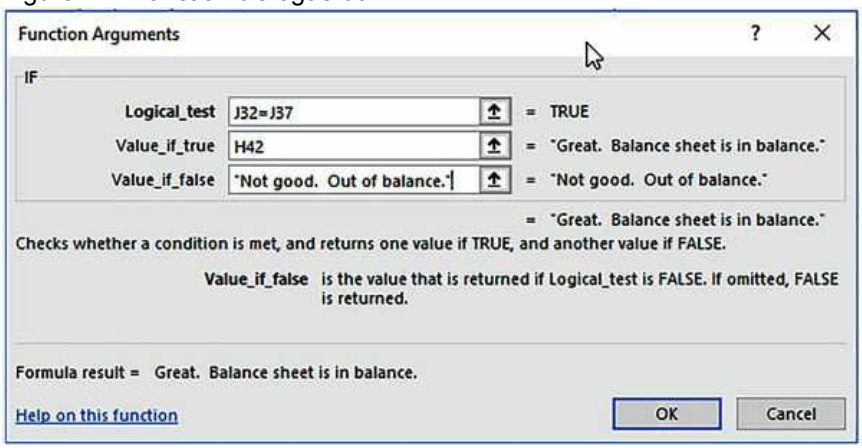

Figure 3: IF function to check that balance sheet is in balance

3.7 Excel Statistical Functions

The Excel spreadsheet provides a wide range of statistical functions (more than 100). Big Data analysts are provided with a complete assortment of descriptive statistics (e.g. Average, Maximum, Minimum, Standard Deviation) together with a variety of Inferential statistics (e.g. Least Squares Regression, Chi Square, F Test, T Test).

Excel's built in statistical functions are available to the Big Data analyst through the Formulas menu tab or to the experienced user by typing the name of the statistical function []. For example, typing *=TREND()* accesses an inferential statistical function that "returns numbers in a linear trend matching known data points, using the least squares method".

An advantage of using the Formulas menu tab to access functions is the informational dialogue box provided to assist entry of the various data required for the requested statistic. Figure 4 provides an illustration of the dialogue box provided for the TREND function.

3.8 Using Excel Data Filters

Excel provides some very useful features for filtering information through the Data tab. The "Sort & Filter" options are provided in the middle area of the Data tab (see Figure 5). Each of these methods require the user to highlight the data to be sorted or filtered before selecting Sort or Filter.

The highlighted area in the above figure will work nicely if the user wants to sort the income statement expense categories. With such sorting it is crucial that all relevant columns are included in the sort area, e.g. not good to sort just one column of values.

Use of the Filter option from the Data tab requires inclusion of a "header row" as this option designates the top row of your selection as a header row of titles that will not be filtered. Excel will allow you to specify a blank row for your header. E.g. the row above … *Selling & marketing expenses 240,100 207,340*.

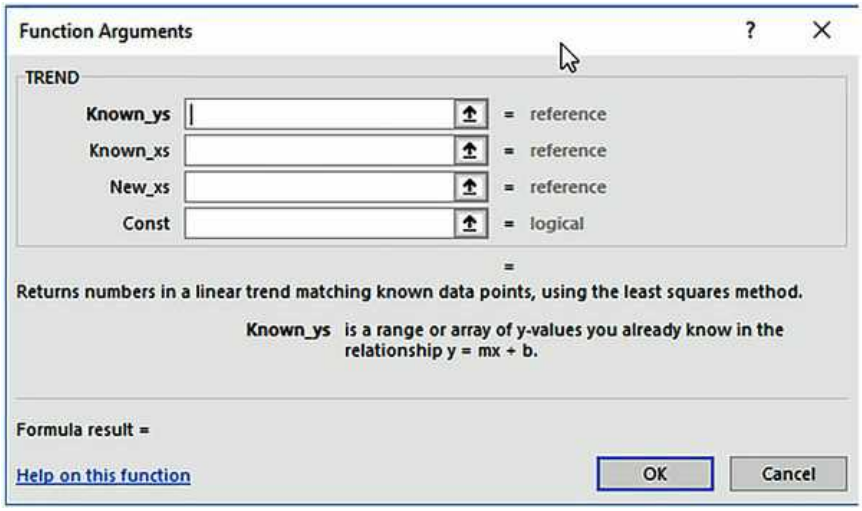

Figure 4: Dialogue box provided for TREND function

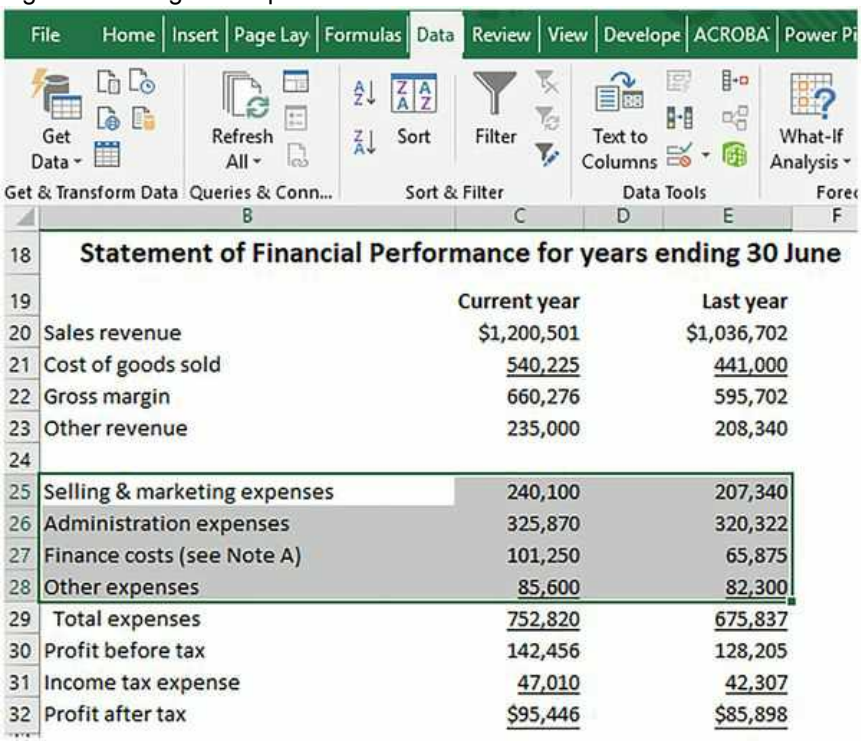

Figure 5: Sort and filter options within the "data" tab with highlighting of range to be sorted

The Data Filter output shown in Figure 6 shows the filter drop-downs that are added to each of the header row cells. You may also note that the label "Account names" has been added to cell B9. This is strictly a cosmetic addition to make it more obvious what data are being filtered in this column.

A more important change to the spreadsheet is the insertion of a blank row (row 14) after "Other expenses". This is necessary due to the Data Filter feature that includes data until a blank cell is found in the first column being filtered. Our blank row 14 serves to end the data filter area at row 13. Selection of the drop-down arrow in cell B9 provides the dialogue box shown in Figure 7.

The above figure shows the variety of options provided by the filter method. In addition to sorting alphabetically, reverse alphabetically and by Colour the method provides several options within the Text Filters choice. These options are in addition to the main filter method shown in the lower portion of the Figure 7 dialogue box. In this instance the user has selected two of the available choices (i.e. Administrative expenses and Other expenses). Selecting OK at this point yields the output shown in Figure 8.

Note from the Figure 8 that the non-selected rows have been hidden (i.e. filtered) and that the drop-down arrow in the "Account names" filter header cell has been changed to a "filter" indicator.

3.9 Power Pivot

Good Big Data analysts will use a wide variety of software applications and analytical methods for both preliminary analysis of "small Big Data" and for their full-blown analytics. The Excel spreadsheet will be an excellent and appropriate software choice for performing many small data analytics, to the knowledge of the authors. However, not infrequently the Big Data analyst will be better served by supplementing Excel with Power Pivot. Arguably (see below), this add-in provides the means for the Big Data analyst to extend their use of Excel beyond small data analysis.

	A	B	C	E	F
8		Other revenue	235,000	208,340	
9		Account names			
10		Selling & marketing expenses	240,100	207,340	
11		Administration expenses	325,870	320,322	
12		Finance costs (see Note A)	101,250	65,875	
13		Other expenses	85,600	82,300	
14					
15		Total expenses	752,820	675,837	
16		Profit before tax	142,456	128,205	
17		Income tax expense	47,010	42,307	
18		Profit after tax	$95,446	$85,898	
19					
20		Note A: These costs relate solely to interest on the long term loan.			
21		Other available data:			
22		Credit sales	840,351	704,957	
23		Cash flow from operations	91,500	92,156	
24					

Figure 6: Output of data filter method: filter drop-downs added to top row of selection

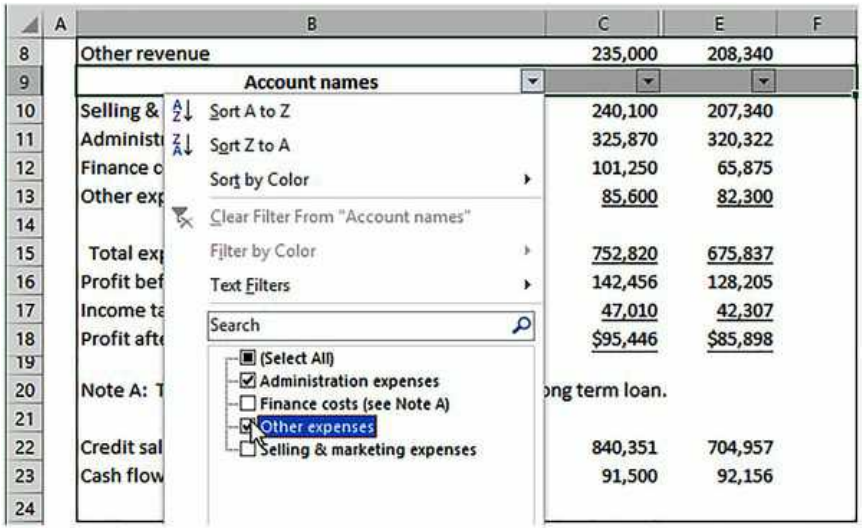

Figure 7: Data Filter dialogue box with filter drop-downs added to top row of selection

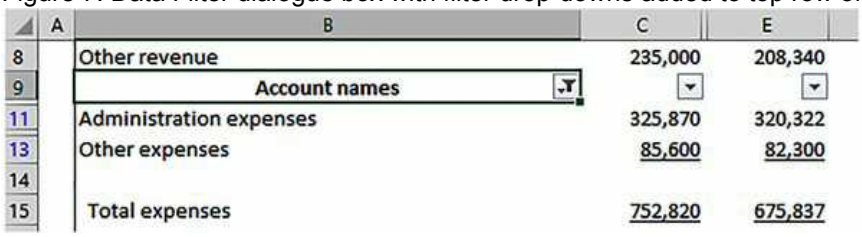

Figure 8: Final output of data filter method with hidden rows

Following is a summary of Microsoft's on-line description of Power Pivot (Microsoft, n.d.-b).

Power Pivot: Powerful Data Analysis and Data Modeling in Excel

Power Pivot is an Excel add-in you can use to perform powerful data analysis and create sophisticated data models.

In both Excel and in Power Pivot, you can create a Data Model, a collection of tables with relationships.

Top Features of Power Pivot for Excel

- Import millions of rows of data from multiple data sources
  - With Power Pivot for Excel, you can import millions of rows of data from multiple data sources into a single Excel workbook, create relationships between heterogeneous data, create calculated columns and measures using formulas, build PivotTables and PivotCharts, and then further analyze the data so that you can make timely business decisions—all without requiring IT assistance.
- Enjoy fast calculations and analysis
  - Process millions of rows in about the same time as thousands, and make the most of multi-core processors and gigabytes of memory for fastest processing of calculations.
- Virtually Unlimited Support of Data Sources
  - Provides the foundation to import and combine source data from any location for massive data analysis on the desktop, including relational databases, multidimensional sources, cloud services, data feeds, Excel files, text files, and data from the Web.
- Data Analysis Expressions (DAX)
  - DAX is a formula language that extends the data manipulation capabilities of Excel to enable more sophisticated and complex grouping, calculation, and analysis.

4 SECTION II: THE ART OF PROGRAMMING WITH VBA_

4.1 Introduction

The nature of Big Data provides insurmountable challenges for spreadsheet use by itself. The volume of transactions (e.g. millions of website transactions), the immergence of non-traditional forms of data (e.g. image based, unstructured textual) and the need to link or relate different datasets exceed the capabilities of even the most expert spreadsheeter. That is, for the spreadsheeter who doesn't know how to use Excel VBA. Unknown to most spreadsheet users is the existence of a powerful and user-friendly programming language (VBA or Visual Basic for Applications) sitting behind the Excel spreadsheet.

The competent Excel user can perform a significant amount of Big Data analysis. This amount can be increased exponentially through the mastery of some basic VBA techniques. As Excel VBA advocates the authors would propose that Big Data analysis of virtually any form could be programmed with Excel VBA. In some cases, this may not be efficient. E.g. why write VBA code to relate datasets when SQL does it better.

The answer lies with the expertise of the Big Data analyst. Do they understand SQL? If they do, the solution is obvious. Use SQL. However, other users will need to weigh the costs of learning SQL against using their preferred software (e.g. Excel and Excel/VBA).

This section of the paper describes the procedures required to access the Visual Basic Editor (VBE). Hidden behind the Excel worksheet, the VBE provides access to the VBA programming language. It then describes the Excel VBA provision of the two basic programming structures that will allow the analyst to perform Big Data analysis of virtually any size or form. Common to all programming languages are these structures: (1) Repeating of instructions (e.g. looping) and (2) Making decisions (e.g. If structures).

4.2 Getting Started With Excel VBA

Excel's macro security settings must be dealt with whenever your spreadsheet contains macros (i.e. Visual Basic code). You will receive a security warning advising you to this regard on opening of the file. Indicate your trust of the macro developer by choosing the option to Enable Content to allow you to run the macros. You should not enable macros from unknown sources.

You may have some setup work to do on your PC before you can use Excel VBA. If the DEVELOPER tab is displayed on the ribbon that's great. Developer provides access to VBA. If you don't see the Developer tab you need to select the FILE tab and then chose OPTIONS. Select the CUSTOMISE RIBBON option and in the Main Tabs section (right side of the screen) "tick" the box beside DEVELOPER. Select OK and your ribbon should now display the Developer tab." The next time you start Excel the Developer tab will be available from your spreadsheet's ribbon.

Macro security settings can be changed through the DEVELOPER tab. Select the MACRO SECURITY option (yellow triangle with exclamation mark) from the left side of the ribbon. Select the option to "disable all macros with notification".

In many instances the Big Data analyst will be creating their data analysis spreadsheet from scratch. Excel will allow you to create macros (e.g. write VBA code) in the default workbook file type. However, your programming code cannot be executed from the.xlsx file type. Your Excel file must be saved as a "Macro-enabled workbook".

From the DEVELOPER tab select the VISUAL BASIC option from the far left of the ribbon. The editor can also be opened with the ALT+F11 key combination and once opened can be accessed through its icon on your PC's taskbar (bottom of your monitor's screen). The VBE for a new workbook should look very similar to Figure 9.

The VBE Project Explorer is analogous to the File Explorer in Windows. It allows you to create various VBA programming objects (e.g. modules) and to navigate between them. Clicking the X in its top right corner closes the Project Explorer. You can redisplay with the View menu command (Project Explorer subcommand) or with the *CTRL+r* key combination shortcut.

Before you can write your program, you need somewhere to put your VBA code. Modules are where VBA code is stored and are easily created with the VBE Insert menu as shown in Figure 10.

The result of successfully inserting a new module is shown in Figure 11. The VBE default name of Module1 can be changed if you want through the Properties window (access with View –> Properties Window). As the VBE option to require variable declaration has been selected; the opening line of code (***Option Explicit***) in the Module1 window is automatically inserted by the VBE for all modules. Option Explicit forces you to follow the programming standard of declaring all variables that are used in your program.

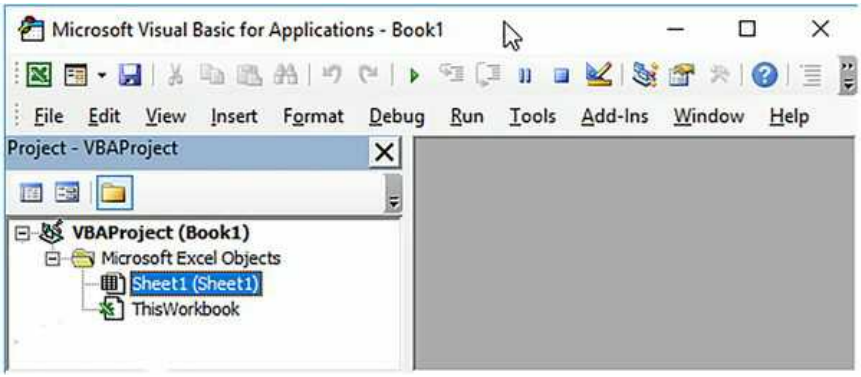

Figure 9: The Visual Basic Editor for a new workbook

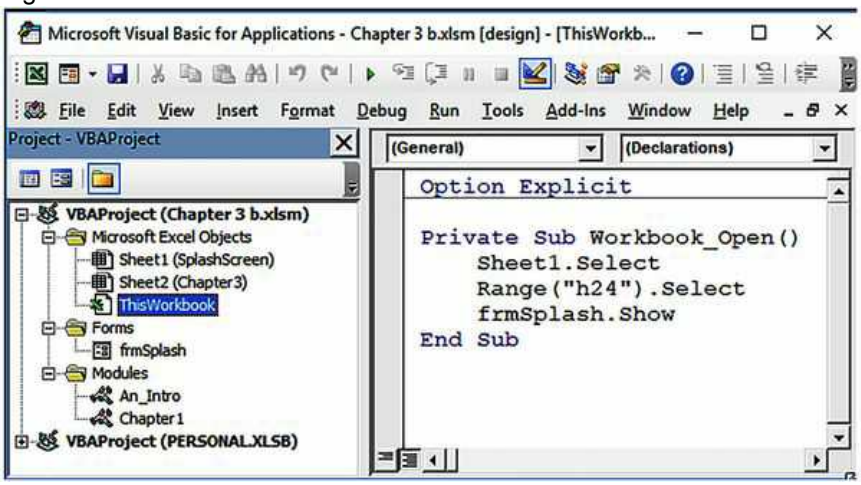

Figure 10: The Visual Basic Editor

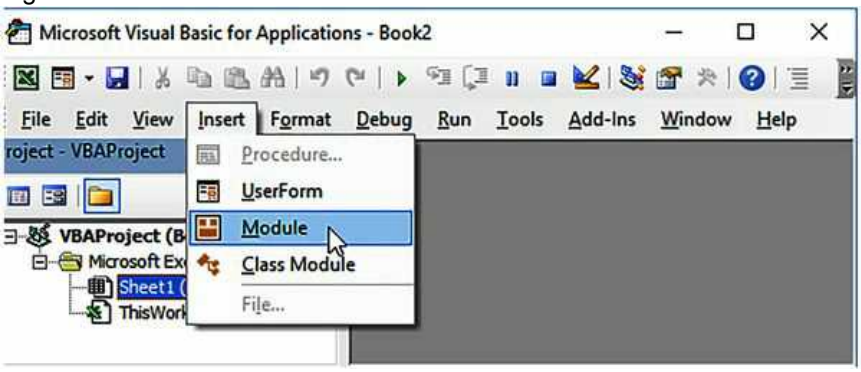

Figure 11: Inserting a module

All VBA subroutines begin with the word "sub" followed by the subroutine name and "()". The conclude with the words "end sub" and contain statements or instructions to be executed in order of appearance. Figure 13 shows the creation of a new subroutine. Note that that neither "()" nor "end sub" have been entered. Pressing Enter at the point displayed in Figure 13 results in the addition of "()" and "end sub" to the new subroutine. The VBE has many features like this to aid the programmer in their entering of VBA code.

The declaration of variables is good programming practice which must be followed because of the use of *Option Explicit*. Variables are usually declared at the start of a sub by typing the word Dim as the first line of code followed by one or more variable names. Variable names can consist of both letters and numbers but must begin with a letter. Spaces and special characters are not allowed. The underscore character _ is allowed. The use of mixed case, descriptive names (e.g. VariableCost1) is highly recommended as mixed case variable names are much easier to read. Multiple declarations on the same line of code must be separated by a comma.

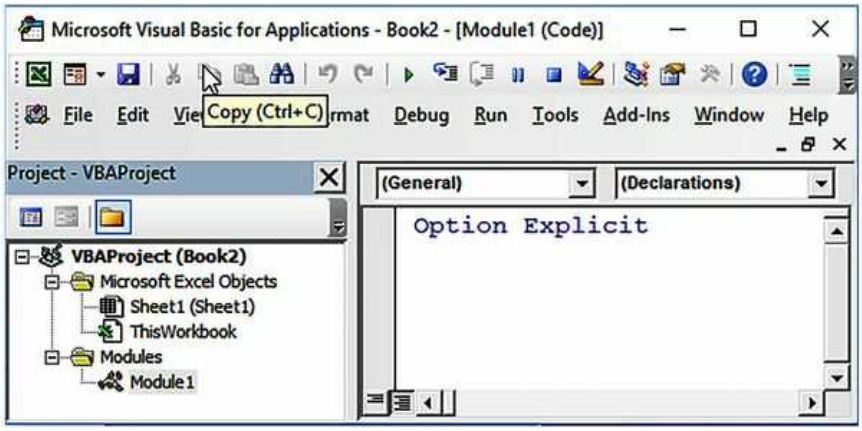

Figure 12: Module 1 successfully inserted

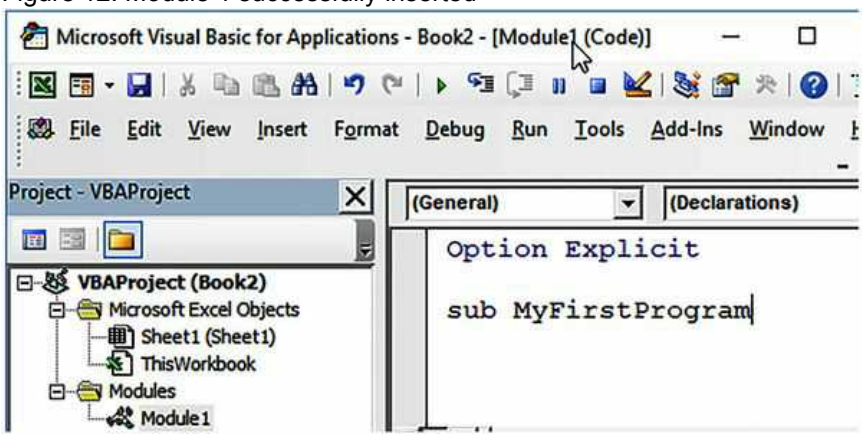

Figure 13: Start of coding for a new subroutine

4.3 Repetition Structures: Do Loops

Repetition structures provide the analyst or programmer with a vital tool for Big Data analysis. The volume of Big Data is going to be significant or it wouldn't be Big Data. The programmer or the spreadsheeter will often want to perform identical (or nearly identical) tasks repeatedly. VBA provides several concise and efficient ways in which this can be done, one of these methods being the Do/Loop structure:

```
Do
Statement(s)
Loop
```

The loop starts at the Do. It then executes each statement until it gets to the Loop statement, which transfers execution back to the Do, and the statements are executed again. The Do/Loop can be used in the simple RevenueLoop subroutine as follows:

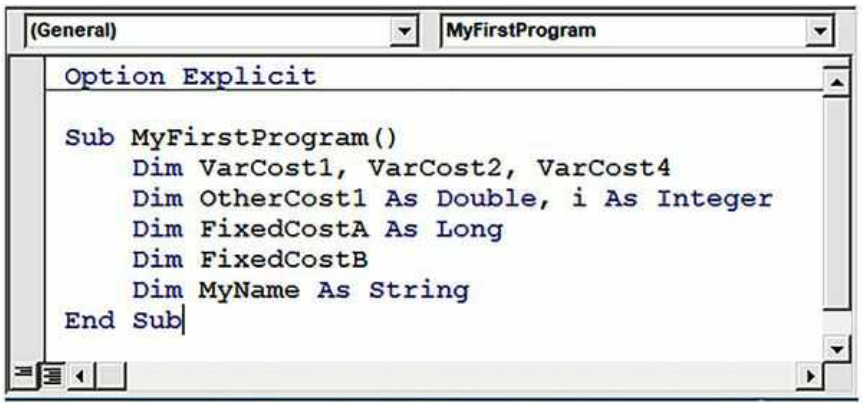

Figure 14: Declaration of variables

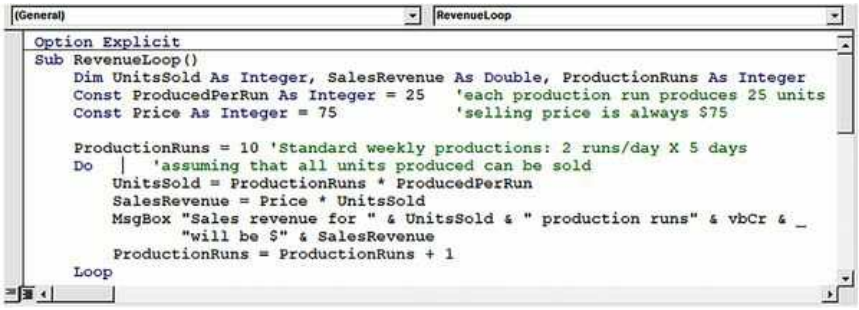

Figure 15: Infinite loop subroutine

The RevenueLoop sub begins by declaring three variables and two constants. Constants may be used in place of variables in situations where the value of an object is not expected to change. The format used is identical to that for declaring a variable with the addition of the equals sign and the constant value. E.g. *Const Price as Integer = 75* as shown in line 4 of Figure 15. The base level of *10 ProductionRuns* is set prior to beginning the Do loop.

The loop begins by multiplying the number of production runs (which is the base level of 10 for the first iteration of the loop) times the units produced per run (i.e. the constant value of 25). The result of 250 (10 * 25) is assigned to the UnitsSold variable. The product of UnitsSold and Price is then assigned to the SalesRevenue variable.

A message box then displays the current values of the UnitsSold and SalesRevenue variables with some explanatory labels. Before getting to the Loop statement and starting all over again, the ProductionRun variable is incremented by one. Figure 16 shows the message displayed after three iterations of the Do Loop.

A somewhat obvious fault of the RevenueLoop subroutine is that it never ends! This is commonly referred to as an infinite loop. To overcome this problem, provisions must be made so that the loop is exited after a finite number of repetitions. Two different approaches are employed to terminate loops.

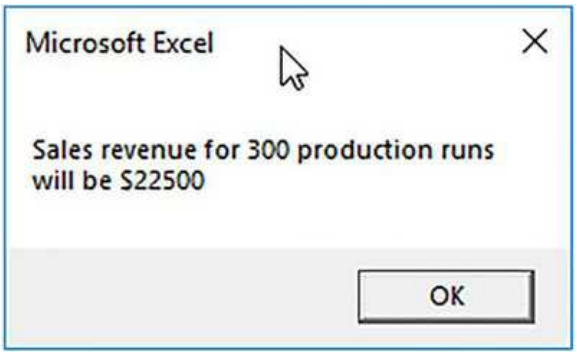

Figure 16: Output of RevenueLoop subroutine after 3 iterations of loop

Decision loops can be terminated based on the state of a logical expression (in other words, an expression that is either true or false). In this sense, decision loops are related to decision constructs. Because such loops involve a decision, they may repeat a different number of times on every execution. In contrast, count-controlled loops are pre-set to repeat a fixed number of times.

4.4 Decision Loops (Do/If Exit) Loops

As the name Do/If Exit implies, this decision loops terminates if a condition is true. The general representation is …

```
Do
Statement(s)
If condition Then Exit Do
Statement(s)
Loop
```

Where condition is a logical condition that tests True or False. Thus, a single-line If statement is used to exit the loop if the condition tests true. Note that, as shown, the exit can be placed in the middle of the loop (that is, with statements before and after it). Such a structure is called a mid-test loop.

If the problem required it, the exit could be placed at the very beginning of the code to create a pre-test loop. An example is:

```
Do
If x < 0 Then Exit Do x = x - 5
```

```
Loop
```

Notice how 5 is subtracted from x on each iteration. This subtraction represents a mechanism that allows the loop to terminate eventually. Every decision loop must have such a mechanism. Otherwise, it would repeat ad infinitum.

Alternatively, the If Exit could be placed at the very end and create a post-test loop:

```
Do x = x - 5
If x < 0 Then Exit Do
Loop
```

Each of the three loop structures are really the same. The only difference between them is the positioning of the exit; at the beginning, in the middle or at the end. Your choice will depend on the structure of your loop exit statement.

4.5 Count Controlled Loops

Although the Do/If Exit loop is certainly a feasible option for performing a specified number of iterations, such looping operations are so common that a special set of statements is available in VBA for accomplishing the same objective in a more efficient manner. Called the For/Next loop, this set of statements has the general format …

```
For counter = start To finish Step increment
Statement(s)
Next
```

Where ***counter*** is a numeric variable used as a loop counter, ***start*** is the initial value of counter, ***finish*** is the final value of counter, while ***Step*** increment is the optional argument for to alter the default increment of 1 for each loop.

The For/Next loop operates as follows: the variable counter is set at an initial value, ***start***. The program then compares ***counter*** with the desired final value, ***finish***. If counter is less than or equal to ***finish***, the program executes the body of the loop. When the ***Next*** statement that marks the end of the loop is reached, counter is increased by ***increment***, and the program loops back to the ***For*** statement. The process continues until ***counter*** becomes greater than ***finish***. At this point, the loop terminates, and the program continues to the line immediately following the loop's ***Next*** statement.

Following in Figure 17 is the code for the RevenueLoop subroutine using a For Next loop. The comments provided within the subroutine (e.g. lines beginning with an apostrophe) explain the differences between this subroutine (using the For Next structure) and the previous sub (using the Do Loop structure). The key difference is the specification of a stopping point; in this example 15 runs as specified by the LastProductionRun variable.

Figures 18 and 19 show the message box output for the first and final iterations of the For Next loop in the RevenueLoopUsingForNext subroutine. Operation is identical to the original sub except there is now an ending point. The infinite loop problem is resolved.

4.6 Decision Structures: IF Statements

Excel VBA provides several alternate structures for the programmer to use for making decisions. Discussed here are the different forms of the IF method. Each of the IF method forms are essentially equivalent. In some cases, one of the methods is arguably preferable but essentially it comes down to programmer predilection.

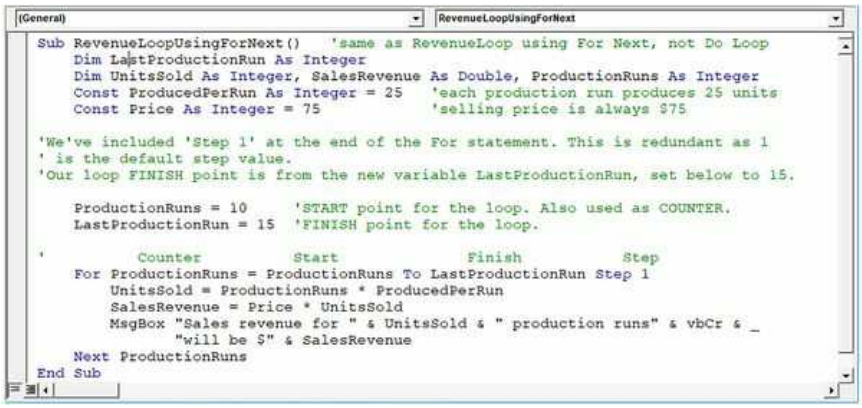

Figure 17: Code for the RevenueLoop sub using a For Next loop

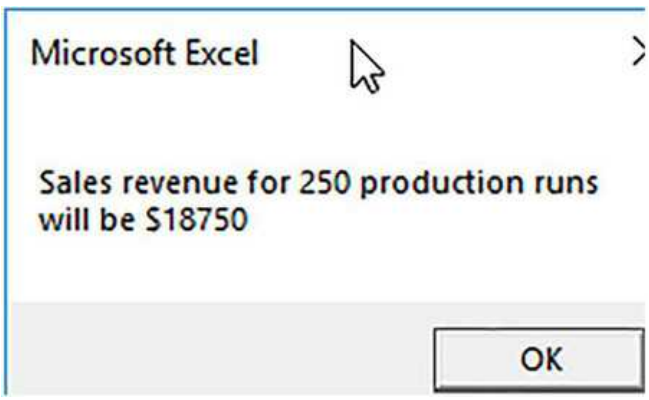

Figure 18: Output after first iteration of loop

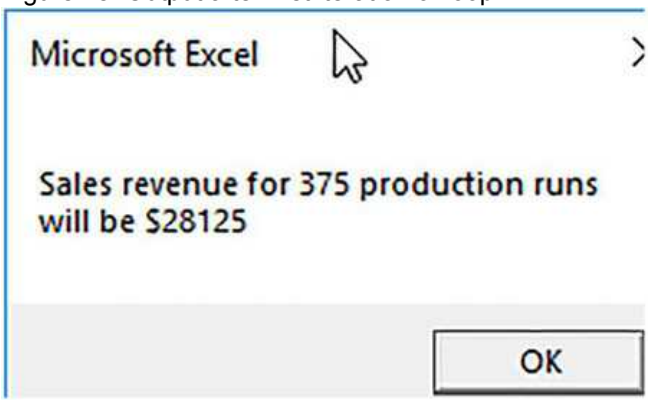

Figure 19: Output after final iteration

The If/Then/Else structure has the syntax

```
If condition Then
TrueStatement(s)
Else
FalseStatement(s)
End If
```

Where condition is a logical expression that evaluates to True or False, *TrueStatement(s)* are one or more statements that are executed if the condition is True, and *FalseStatement(s*) are one or more statements to be executed if the condition is False.

The simplest form of a condition is a single relational expression to compare two values, as in

```
value1 relation value2
```

Where values can be constants, variables, or expressions, and relation is one of the relational operators such as equals (=), greater than (>) or less than (<). Simple examples of conditions are

```
Mark = 100                    Mark >= 75                    Mark <= 50
```

An example of code that uses the If / Then / Else structure is

```
If Mark >= 50 Then
Grade = "Satisfactory"
Else
Grade = "Fail"
End If
```

You will notice the indentation of the lines of code. While not essential; it is good, standard programming practice. Indenting makes code easier to read and understand.

4.7 If /Then Single Decision Structure

In cases where there is no false alternative, the If/Then/Else structure can be simplified by dropping the Else clause, as in

```
If condition Then
TrueStatement(s)
```

```
End If
```

Following is an example using this structure. Suppose you develop a program that requires the user to enter a value for the number of units produced of an object into a worksheet cell, say, B5. The VBA code to fetch this value into your program and assign it to a variable Units can be written as:

```
Range("b5").Select
Units = ActiveCell.Value
```

or

```
Units = Range("b5").Value          Assigns the value without moving the
cursor.
```

or

```
Units = Range("b5")                  Relies on Value being the VBA de-
fault for Range.
```

Clearly, the number of units produced cannot be less than zero. Consequently, it would be nice to prevent the user from entering a value that was less than zero. The If/Then structure provides a neat way to detect whether a user's input contains such errors. The following code is illustrative:

```
'Error trap to prevent a negative entry.
If Units < 0 Then
MsgBox "Units cannot be less than zero. Try again."
Range("b5").Select                 'moves cursor to cell B5
End                                   'terminates execution of sub
End If
```

If a positive value is entered in cell b5, the program will immediately skip to the End If statement and proceed with the remainder of the program. If an incorrect value (i.e. a negative number or zero) is entered, a message box appears indicating that "Units cannot be less …"

The preceding code does three good things from a programming point of view:

1. It provides the user with constructive feedback in the form of an error message.
2. It places the active cell at the location that needs to be fixed (b5).
3. It terminates execution of the Sub so that the user can correct the mistake and run the program again. Use of the END statement is good programming practice as it prevents further code from running after an error has occurred.

The repetition and decision structure examples have included several assignments of values to VBA variables and spreadsheet cells. Labels or strings of text can also be assigned to variables or cells with the same process. As illustrated where the text string was assigned for ice hockey legend Wayne Gretzky, it's a straightforward process. Your text string needs to be within quotations marks on the right-hand side of your equation. Your VBA variable name or spreadsheet cell is on the left-hand side and assignment is performed with the equals sign, e.g. Range("a99") = "Wayne Gretzky" will input text into cell A99 of the active worksheet of the active workbook.

4.8 Documentation

Our discussion of spreadsheet techniques in Section I is incomplete as it doesn't include a dialogue of the importance of documentation. Appropriate documentation is a critical component of both good spreadsheeting and good programming as it enables the updating of the program / spreadsheet to deal with a changing environment (e.g. new AI technology used by customers) and / or with new statistical outputs required by the analyst or the analyst's boss (e.g. "could you perform a multi-factor regression that simultaneously considers the effect of previous purchases, time of day, phase of the moon and recent success of the customer's favourite sports team as determinants of the likelihood of a current purchase"). In his essay on 'What Tool is Best?' Plauger (1993, p. 67) advises the programmer that they "must know what constitutes adequate documentation both for internal maintenance and for use by the customer".

Documentation is critical but oftentimes it is not given adequate attention. Perhaps you're one of the lucky ones who's forced to provide documentation as your organisation demands it. Or perhaps you've learned the hard way as the authors have and attempted to use a model (or even a complicated formula) a long time after creating it. And spent ages working out how it works.

The authors can speak from experience and promise that a little bit of documentation can go a long way towards increasing the future usability of your model. By someone else or by the modeler themselves. Don't fall into the other trap and convince yourself that your model is so good that it is "self-documenting". While a well-structured and labelled model will be infinitely more usable than a "spaghetti code" abomination it will almost certainly still benefit from a few words of precise documentation.

And when's the best time to document your work? Immediately as the model is completed is alright but it is still going to be more difficult to do than if you add bits of documentation as you are creating the model. Hopefully it's extremely clear to you when you're writing the formula or line of code. It will be much less obvious in an hour/day/week. Do yourself a favour and write your documentation as you're building the model. You'll certainly thank yourself when you need to make changes in a year's time and perhaps even when you're doing the final debugging.

5 USING EXCEL, EXCEL VBA AND POWER PIVOT FOR PRELIMINARY ANALYSIS IN BIG DATA RESEARCH: A CASE STUDY OF BANKRUPTCY PREDICTION

This section uses corporate bankruptcy prediction to illustrate the use of Excel, Excel VBA and Power Pivot for performing data cleansing and small data analysis. Bankruptcy prediction is an ideal case study for our illustration in several respects.

Traditional bankruptcy research has relied on the calculation and analysis of various financial variables such as current ratio, acid test and debt to equity. Determination of values for these variables for a longitudinal analysis of 10 years, 20 years or more can be efficiently conducted by any competent spreadsheet user. Previously (e.g. before Big Data) the entire analysis component of a bankruptcy research study could arguably have been effectively performed using spreadsheet software. Not so in the twenty-first century with the arrival of Big Data.

The arrival of Big Data and advancement in the scope of bankruptcy predictors (beyond financial variables) has caused virtually all serious bankruptcy researchers to abandon their spreadsheets for the use of statistical packages such as SPSS, SAS and R. The data provided from advanced analytical techniques such as support vector machines, neural networks and AdaBoost will initially be viewed as being inappropriate (too voluminous or varied) for spreadsheet analysis. However, this case study will illustrate that spreadsheet software still has a valuable role to play with bankruptcy prediction and in other areas.

Even without advanced data (e.g. AdaBoost) the volume and variety of information available (e.g. from stock exchange databases) for bankruptcy research is immense. Figures 20 and 21 provide a snapshot of 12 rows from an ASX database comprised of over 800,000 rows of data.

The spreadsheet trained Big Data analyst might decide that analysis for such a database might begin with use of the Data Filter method described in section 3.8. Small data analysis could then include the use of the custom auto-filter from the numbers filter option. Figure 22 illustrates the analyst's extraction of extreme data for the "Price or Bid/Ask Average" variable; i.e. prices above $120 or below $8.

Figure 23 shows an extract of the filtered data from the Figure 22 number filter. As expected the Price values in column N are either less than $8 or greater than $120. The analyst may extract this data for further analysis in another statistical package or choose to conduct further investigation within the spreadsheet model.

Preliminary spreadsheet analysis such as that just described allows the analyst (i.e. the bankruptcy researcher in this case) to quickly obtain indications of where further "serious analysis" should be focused. Just as important, the spreadsheet analysis may indicate areas of analysis that should be avoided. Appropriate preliminary analysis in Big Data research will go a long way towards at least partially solving the problem of TMI (i.e. too much information).

Small data analysis could also include the use of Power Pivot as described in Section 3.8. Most spreadsheet users and business analysts are aware of pivot tables as an efficient means of summarizing data and highlighting desired information. The Excel spreadsheet by itself provides for the creation of pivot tables. However, the limitations of Excel (e.g. 1,048,576 rows) may come into play for the Big Data analyst. As described in this earlier section, Power Pivot expands the use of pivot tables to accommodate data tables that cannot be handled by Excel alone. Most spreadsheet users know all to well they are limited to much less than the maximum number of rows. Frequent crashes and calculation updates that take several minutes or longer become the norm when the analyst pushes their spreadsheet boundaries too much.

	A	B	C	D	E	F	G	H	L	M	N
1	PERMNO	Names Date	Exchange Code	Ticker Symbol	Company Name	Primary Exchange	Trading Status	CUSIP Header	Bid or Low Price	Ask or High Price	Price or Bid/Ask Average
803348	10026	04/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	102.81	117.13	113.99
803349	10026	07/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	113.195	116.83	116.59
803350	10026	08/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	115.8	117.9	115.99
803351	10026	09/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	112.9	117.6	116.85
803352	10026	10/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	105.06	116.77	116.21
803353	10026	11/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	114.69	119.16	119.01
803354	10026	14/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	118.34	120.18	119.32
803355	10026	15/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	118.535	120.115	119.32
803356	10026	16/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	118.1	122	120.8
803357	10026	17/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	120.34	122.78	122.19
803358	10026	18/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	122.47	124.5	124.17
803359	10026	21/11/2016	3	ABCD	ABCD Ltd	Q	A	46603210	122.09	124.65	123.87

Figure 20: ASX (Australia Stock Exchange) data – bottom 12 rows – columns A-N

	O	P	Q	R	S	T	U	V	W	X	Y
1	Volume	Returns	Bid	Ask	Shares Outstd'g	Price Alternate	Returns without Dividends	Value-Weighted Return-incl. dividends	Value-Weighted Return-excl. dividends	Equal-Weighted Return-incl. dividends	Equal-Weighted Return-excl. dividends
803348	127953	-0.02155	113.89	113.9	18668	114.89	-0.021545	-0.001239	-0.00128	0.000762	0.000692
803349	88091	0.022809	116.48	116.57	18668	114.44	0.022809	0.020921	0.020898	0.016048	0.015996
803350	69622	-0.00515	115.98	116.07	18683	115.87	-0.005146	0.003886	0.00353	0.002824	0.002656
803351	78725	0.007414	116.9	116.99	18683	114.75	0.007414	0.012049	0.011949	0.01576	0.015604
803352	88979	-0.00548	116.2	116.21	18683	115.8	-0.005477	0.001947	0.001836	0.006012	0.005822
803353	113433	0.024094	118.93	119.01	18683	116.54	0.024094	0.000248	0.000248	0.008794	0.008794
803354	81004	0.002605	119.23	119.32	18683	120.01	0.002605	0.002364	0.002254	0.005859	0.005792
803355	79049	0	119.21	119.32	18683	119.57	0	0.007917	0.00774	0.007791	0.007712
803356	58147	0.012404	120.8	120.9	18683	119.31	0.012404	-0.001509	-0.001714	0.001924	0.001801
803357	50253	0.011507	122.15	122.19	18683	121.26	0.011507	0.004593	0.004543	0.002985	0.002921
803358	59927	0.016204	124.19	124.33	18683	122.47	0.016204	-0.001348	-0.001473	0.001528	0.001467
803359	46620	-0.00242	123.86	123.87	18683	123.98	-0.002416	0.007775	0.007747	0.006578	0.006461

Figure 21: ASX (Australia Stock Exchange) data – bottom 12 rows – columns O-Y

H	L	M	N	O	P	Q	R	S
CUSIP Head	Bid or Low Pri	Ask or High Pri	Price or Bid/Ask Average	Volum	Return	Bid	Ask	Shares Outstd
36720410	7.44	7.59	7.52	15816	0.009396	7.52	7.58	10505
36720410	7.31	7.58	7.42	22037	-0.0133	7.36	7.4	10505
36720410	7.2501	7.55	7.53	11532	0.014825	7.48	7.55	10505

Custom AutoFilter

Show rows where:

Price or Bid/Ask Average

is greater than 120

And Or

is less than 8

Use ? to represent any single character

Use * to represent any series of characters

Figure 22: Custom number filter dialogue box

	A	B	C	D	E	F	G	H	L	M	N
1	PERMNO	Names Da	Exchange Code	Ticker Symb	Company Name	Primary Exchan	Trading Statu	CUSIP Header	Bid or Low Pri	Ask or High Pri	Price or Bid/Ask Averag
584065	10001	15/01/2016	2	EGAS	GAS NATU	A	A	36720410	7.8201	8.12	7.99
584066	10001	15/01/2016	2	EGAS	GAS NATU	A	A	36720410	7.8201	8.12	7.99
584067	10001	15/01/2016	2	EGAS	GAS NATU	A	A	36720410	7.8201	8.12	7.99
584068	10026	01/11/2016	3	EGAS	J & J SNAC	Q	A	46603210	119.76	122.485	120.05
584069	10026	01/11/2016	3	EGAS	J & J SNAC	Q	A	46603210	119.76	122.485	120.05
584070	10026	01/11/2016	3	EGAS	J & J SNAC	Q	A	46603210	119.76	122.485	120.05
584071	10026	01/11/2016	3	EGAS	J & J SNAC	Q	A	46603210	119.76	122.485	120.05

Figure 23: Extract of data filtered with custom number filter (col N) <8 or >120

As described in Section 3.8, Power Pivot provides for the analysis of millions of rows of data (from multiple data sources if desired) in a single workbook. The analyst can create relationships between heterogeneous data, calculated columns and measures using formulas. Figure 23 provides an illustration of a summary information table created using Power Pivot with the previous bankruptcy data set.

Figure 24 provides a rudimentary illustration of the power that Power Pivot adds to the Excel spreadsheet. In truth, the Big Data analyst's capabilities are dramatically extended through the use of Power Pivot. So much so that some analysts may extend their use of Excel beyond small data analysis.

6 CONCLUSION

This chapter looks at how Excel can be used in conjunction with other software and analytical techniques in big data research. This paper also argues where and how to use spreadsheet software to conduct big data research. A focal argument of this chapter is that the key behind big data driven research is data cleansing and big data driven small data analysis. The proposed approach in this chapter might facilitate the research and development of intelligent big data analytics, big data analytics, and business intelligence.

	A	B	C	D	E	F	G
1		Power Pivot analysis of Price or Bid/Ask Average Data - Filter of < $8 or > $120					
2							
3		Row Labels	Count of Pric	Sum of Price	Average	Max of Pri	Min of P
4		ABCD Ltd	4	$491.03	$122.76	$124.17	$120.80
5		GAS NATURAL INC	167,504	$1,231,571.51	$7.35	$7.99	$6.88
6		J & J SNACK FOODS CORP	62,810	$7,637,426.89	$121.60	$124.17	$120.05
7		Grand Total	230,318	$8,869,489.43	$38.51	$124.17	$6.88

Figure 24: Power Pivot summary of data filtered with custom number filter (col N) <8 or >120

It is the authors' firm opinion that the humble spreadsheet has a significant role to play in making Big Data analysis available to the masses. The global data giants (e.g. Google, Amazon, etc) will still be the key players. However, small data analysis with Excel will provide an invitation to the Big Data party for academics and business (i.e. non-global data giant companies). The use of Excel VBA and Power Pivot are the keys for turning Excel into a Big Data analyst's first choice for analysis.

REFERENCES

Baker, S. (2018, May 9). *Cambridge Analytica Won't Be Revived Under New Company Name*. Retrieved from Bloomberg: *https://www.bloomberg.com/news/articles/2018-05-08/cambridge- analytica-won-t-be-revived-under-new-company-name*

Butler, S. (2018, March 22). *Bargain Booze owner Conviviality must raise £125m to halt bankruptcy*. *The Guardian*. Retrieved from *https://www.theguardian.com/business/2018/mar/21/bargain- booze-owner-conviviality-must-raise-125m-to-halt-bankruptcy*

Department of Communication and the Arts. (2018, February 27). *Future trends in bandwidth demand*. Retrieved from *https://www.communications.gov.au/departmental-news/ future-trends-bandwidth-demand*

European Spreadsheet Risks Interest Group (EuSpRIG). (n.d.a). *Horror stories*. Retrieved from *http://www.eusprig.org/horror-stories.htm*

European Spreadsheet Risks Interest Group (EuSpRIG). (n.d.b). *Welcome*. Retrieved from *http://www.eusprig.org/index.htm*

Jones, S. (2017). *Corporate bankruptcy prediction: a high dimensional analysis*. *Review of Accounting Studies*, 22(3), 1366.

Jones, S., Johnstone, D., & Wilson, R. (2016). *Predicting Corporate Bankruptcy: An Evaluation of Alternative Statistical Frameworks*. *Journal of Business Finance & Accounting*, 44, 1–2. doi:10.1111/jbfa. 12218

Microsoft. (n.d.b). *Power Pivot: Powerful data analysis and data modeling in Excel*. Retrieved from *https://support.office.com/en-us/article/Power-Pivot- Powerful-data-analysis-and-data-modeling-in-Excel-A9C2C6E2- CC49-4976-A7D7-40896795D045*

National Research Council. (2013). *Frontiers in Massive Data Analysis*. Washington, DC: The National Research Press.

Panko, R. (1998). *What we know about spreadsheet errors*. *Journal of Organizational and End User Computing*, 10(2), 15–21. doi:10.4018/joeuc. 1998040102

Plauger, P. J. (1993). *Programming on purpose: Essays on software design*. Englewood Cliffs, NJ: Prentice Hall.

Powell, S. G., & Baker, K. R. (2004). *Management science: The art of modeling with spreadsheets*. Hoboken, NJ: Wiley.

Pressman, R., & Maxim, B. (2014). *Software engineering: A practitioner's approach*. Boston: McGraw-Hill.

Raffensperger, J. F. (2001). *New guidelines for spreadsheets*. *Proceedings of the European Spreadsheet Risks Information Group (EuSpRIG)*, 61-76.

Sun, Z., Sun, L., & Strang, K. (2018). *Big data analytics services for enhancing business intelligence*. *Journal of Computer Information Systems*, 58(2), 162–169. doi:10.1080/08874417.2016.1220239

Sun, Z., & Wang, P. (2017). *Big Data, Analytics and Intelligence: An Editorial Perspective*. *Journal of New Mathematics and Natural Computation*, 13(2), 75–81. doi:10.1142/S179300571702001X

Sweller, J. (2010). *Element interactivity and intrinsic, extraneous, and germane cognitive load*. *Educational Psychology Review*, 22(2), 123–138. doi:10.100710648-010-9128-5

van der Aalst, W. M. P., ter Hofstede, A. H. M., & Weske, M. (2003). Business process management: A survey. In W.M.P van der Aalst, & M. Weske (Eds.), *Business Process Management. Lecture Notes in Computer Science* (vol. 2678). Berlin: Springer. doi:10.1007/3-540-44895-0_1

Van Merriënboer, J. J. G., & Krammer, H. P. (1987). *Instructional strategies and tactics for the design of introductory computer programming courses in high school*. *Instructional Science*, 16(3), 251–285. doi:10.1007/BF00120253

Walkenbach, J. (2002). *Microsoft Excel 2000 power programming with VBA*. Foster City, CA: IDC Books.

Walkenbach, J. (2010). *Microsoft Excel 2010 power programming with VBA*. Hoboken, NJ: Wiley. doi:10.1002/9781118257616