**Excel at Cutting Costs: Combining SAS and Microsoft Excel to Reduce Headcount**

Keith Fredlund, Independent Researcher, St. Louis, MO
Cody Cattell, Bronson Healthcare, Kalamazoo, MI
Anson Phillips, Charles River Laboratories, Mattawan, MI

ABSTRACT

Many organizations employ skilled statistical programmers to run analyses and create reports using SAS software. But the ability to program in SAS is an advanced and uncommon skill requiring special training to acquire. In comparison to the percentage of the workforce skilled in SAS programming, the percentage of people able to use Microsoft Excel is quite large. Accordingly, it can be difficult for organizations to find individuals with skills in SAS programming and companies pay a salary premium for employees with this capability. When conducting analyses or creating reports that have repeated elements across iterations, it can be cost-effective for a company to employ an individual without SAS programming skills to conduct analyses using a Microsoft Excel file as an interface to SAS. This paper describes a handful of techniques, useable by an individual with a basic level of programming skill, which would allow a company to reduce costs by substituting a worker having Excel skills into a role that would normally require someone with SAS programming capability. An additional benefit of this approach is that it takes advantage of the unique benefits of both SAS and Excel. In cases where complete automation is not possible, these techniques can be leveraged to reduce errors and cut costs.

INTRODUCTION

The ability to use Microsoft Excel is comparatively more common than the ability to write code in SAS. Accordingly, companies pay a premium for individuals with SAS skills. Companies that run analyses or reports that cannot be completely automated (e.g., where important information needs to be pulled from within paragraphs of text or when incoming data does not arrive with standardized nomenclature), but where changes are only required to defined parameters, can use employees with modest skill in Excel to run these analyses without requiring a skilled SAS programmer to be involved each time a program is run.

In some situations, it may be possible for a non-SAS user to learn enough coding skills to modify a SAS program each time a program is used (perhaps by changing the values of a series of macro variables positioned at the start of a program), but this approach has a few risks.

First of these is that an unskilled programmer may inadvertently change a part of a SAS program, perhaps by mistakenly removing a semicolon or by failing to include quotation marks where necessary, which causes errors or warnings to appear in the log when the program is run.

Secondly, an unskilled programmer may inadvertently enter values into macro variables within a SAS program that cannot be interpreted by the program. Using Excel it is possible to restrict entries in a given cell so that a user may enter only codes that are on an approved list. For instance, if a given input parameter expects only imperial units, but the user attempts to enter “kg” (for kilograms), Excel can prevent the user from entering the improper units while SAS will not. Another benefit of excel is that in cases where only numbers between 3 and 300 are acceptable, Excel can prevent a user from entering values outside of this range.

The approach described in this paper allows a beginner SAS programmer with little skill with Excel, to create an Excel file that can function as an interface with SAS. This approach will result in data being pulled from an Excel sheet into macro variables to be used in a SAS file, where they can be used to control the operation of that program.

## TECHNIQUE: BUILDING AN EXCEL FILE TO INPUT VALUES

In implementing this approach, the first step is to create an Excel file to be used as an input for a SAS program, allowing a user to enter responses to questions that that will be converted by the SAS file into macro variables.

In this example (Fig. 1 and Fig. 2), an excel file has been set up with two worksheets (Sheet1 and Sheet2), each with the same columns:

1. Question_N: Populated with sequential integers.
2. Question: Fields containing each of the questions that needs to be answered.
3. Response: This field can be set up to accept only a limited number of responses (ie. Male, Female, or Both) or a restricted type of response (i.e., only numeric values within certain limits for measurements of distance).
4. Notes and Guidelines: This field can be used to describe the types of values that can be entered in the Response column, provide advice on how to choose between values, or identify where information on how to answer the question could be found.
5. Allowable Response List (As Appropriate): In cases where the acceptable values for a Response are limited and definable, a list of the allowable responses can be included starting in the column named Allowable Response List (As Appropriate) and extended across columns to the right.

Fig. 1: Excel File Sheet1

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | **Question_N** | **Question** | **Response** | **Notes and Guidelines** | **Allowable Response List (As Appropriate)** | | | |
| 2 | 1 | Cell Type? | | Choose between RBC or WBC. | RBC | WBC | | |
| 3 | 2 | Type of animal? | | Choose between Dog, Mouse, Rat or Pig. | Dog | Mouse | Rat | Pig |
| 4 | 3 | What was the maximum weight of animal requested from the supplier (in grams)? | | Enter a number. | | | | |
| 5 | 4 | What sexes are present in the study? | | Choose between Male, Female or Both. | Male | Female | Both | |

Sheet1 Sheet2

Fig. 2: Excel File Sheet2

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | **Question_N** | **Question** | **Response** | **Notes and Guidelines** | **Allowable Response List (As Appropriate)** | | |
| 2 | 1 | Type of analysis | | Choose between RMANCOVA and Latin Square RMANCOVA. | RMANCOVA | Latin Square RMANCOVA | |
| 3 | 2 | Within-Subject Test: Use univariate approach if possible? | | Choose Yes or No. If Yes is chosen and the assumption of sphericity is met, the univariate test will be used. | Yes | No | |
| 4 | 3 | First covariance matrix to test? | | Choose between AR(1), CS or UN. Consult the Statistics section of the study protocol. | AR(1) | CS | UN |
| 5 | 4 | Second covariance matrix to test? | | Choose between AR(1), CS, UN or leave blank. Consult the Statistics section of the study protocol. | AR(1) | CS | UN |
| 6 | 5 | Third covariance matrix to test? | | Choose between AR(1), CS, UN or leave blank. Consult the Statistics section of the study protocol. | AR(1) | CS | UN |

Sheet1 Sheet2

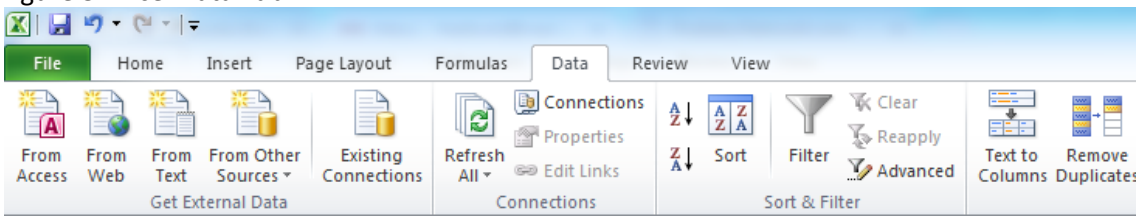
When the Excel file has been created, the user can fill in cells in the Response column with any of the values in the Allowable Response List, if the required value is limited to a list, or they may enter other values (e.g., numbers) if a measurement is appropriate as a response to the query in the Question field. Note that in the example above (Fig. 1), Question_N 3 is not limited to set responses (i.e., column E and those the right are blank); instead the Response cell for this row has been set to accept any numeric response.

USING DATA VALIDATION IN EXCEL TO REDUCE THE POSSIBILITY OF ERRORS

One of the significant benefits to using Excel as an interface is the opportunity to use Data Validation to restrict allowable entries in the response. In the 2010 version of Excel, these are the steps to implement data validation:

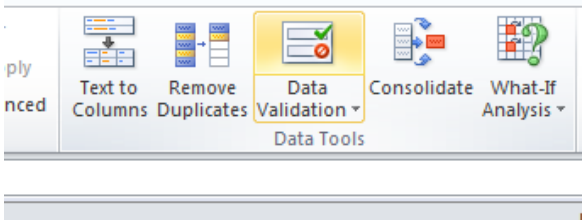
1. Select the cell in the Response column to be validated. This step must be completed once for each Question requiring a restriction on the type of entries that can be used.
2. Click the Data tab at the top of the page (Fig. 3).

Figure 3: Excel Data Tab



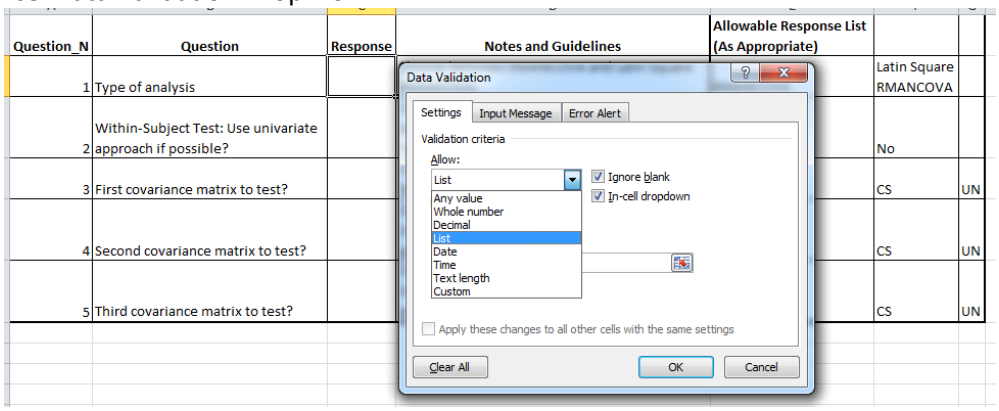

3. From the ribbon at the top of the page, choose the Data Validation button (Fig. 4).

Figure 4: Excel Data Validation Button



4. A pop-up window will appear allowing the user to choose the type of restrictions to put on data entered into the cell.
5. From the Allow drop-down, choose from one of the options: Any value, Whole number, Decimal, List, Date, Time, Text length or Custom.
    a. For responses with a set group of allowable responses, choose List and enter the range of appropriate cells from the Allowable Response List (As Appropriate) (Fig. 5).

Figure 5: Excel Data Validation Drop-Down




    b. For numeric responses that must be within certain range (e.g., Question_N 3 on Sheet1), the Allow field can be set to "Between" and the minimum and maximum values can be entered into the Minimum and Maximum fields that appear in the Data Validation pop-up window.
6. Click OK.
7. Repeat Steps 1 through Step 6 for each question on each worksheet in the excel file.
8. Save this file into a folder that will contain both it and the SAS program that references it (described below).

Rather than saving this file as a regular Excel file, it will have greatest usefulness if saved as a template. When a future user saves the interface after modifying it, Excel will provide a prompt to save it with a new file name, removing the risk that the original file will be overwritten.

After the template has been opened and assigned a new file name, the user can prepare the Excel interface for use with a SAS program by filling in the cells in the Response column. Figure 6 is an example of how the first sheet of an Excel file would appear when updated.

Figure 6: Excel File Sheet1 - Completed

| Question_N | Question | Response | Notes and Guidelines | Allowable Response List (As Appropriate) | | | |
|---|---|---|---|---|---|---|---|
| 1 | Cell Type? | WBC | Choose between RBC or WBC. | RBC | WBC | | |
| 2 | Type of animal? | Mouse | Choose between Dog, Mouse, Rat or Pig. | Dog | Mouse | Rat | Pig |
| 3 | What was the maximum weight of animal requested from the supplier (in grams)? | 30 | Enter a number. | | | | |
| 4 | What sexes are present in the study? | Male | Choose between Male, Female or Both. | Male | Female | Both | |

For the SAS program described in the following section to work without modification, the names of the Excel worksheets should be Sheet1, Sheet2 and so on.

IMPORTING INFORMATION (CODES) FROM EXCEL INTO SAS

The following simple SAS program imports selected data from the Excel interface, creating a series of macro variables to store the information from the Excel sheet.

```
OPTION VALIDVARNAME=ANY;
OPTION SYMBOLGEN;
```

These options allow SAS to read a wider range of Excel names than is allowed by default in SAS (option validvarname) and display the values of created variables (option symbolgen).

```
LIBNAME SOURCE "./<STANDARDIZED_NAME_OF_EXCEL_INTERFACE_FILE.XLS>";
```

In order for the Excel interface file and the SAS program to work in tandem without requiring changes to the SAS program, the Excel interface file must always have the same name.

```
%LET MV_NUMBER_OF_EXCEL_SHEETS = 2;
      *<-- SET THIS NUMBER BASED ON THE NUMBER OF WORKSHEETS YOU CREATE IN
      YOUR EXCEL FILE.;
%LET MV_NUMBER_OF_QUESTIONS_SHEET_1 =3;
      *<-- SET THIS NUMBER BASED ON THE NUMBER OF QUESTIONS IN WORKSHEET 1.;
%LET MV_NUMBER_OF_QUESTIONS_SHEET_2 =5;
      *<-- SET THIS NUMBER BASED ON THE NUMBER OF QUESTIONS IN WORKSHEET 2.;
```

If only one Excel worksheet is needed, delete the second macro variable in the section above. If more than two Excel worksheets are needed, additional lines of code stipulating the number of macro variables on these sheets can be added. The macro variable for the number of questions on each sheet needs to be set individuals for each Excel worksheet.

```
%MACRO M_READ_DATA_SETS;
      %DO I = 1 %TO &MV_NUMBER_OF_EXCEL_SHEETS;
            DATA DS_ORIGINAL_SHEET&I.; SET SOURCE."SHEET&I.$"N;
            RUN;
      %END;
%MEND;
%M_READ_DATA_SETS;
```

The M_Read_Data_Sets macro reads in as many excel sheets as were stipulated in the MV_Number_Of_Excel_Sheets macro variable.

```
%MACRO M_SET_MACRO_VARIABLES;
      %DO I = 1 %TO &MV_NUMBER_OF_EXCEL_SHEETS.;
      *<--- CREATES AN INDICATOR FOR THE NUMBER OF EXCEL WORKSHEETS;
            %DO J = 1 %TO &&MV_NUMBER_OF_QUESTIONS_SHEET_&I.;
            *<--- CREATES AN INDICATOR FOR THE NUMBER OF QUESTIONS ON EACH
            EXCEL WORKSHEET;
            %GLOBAL MV_SHEET_&&I._Q&J.;
                  DATA _NULL_;
                        SET DS_ORIGINAL_SHEET&I.;
```

```
                        *CYCLES THROUGH THE DATA SETS;
                              IF QUESTION_N = "&J." THEN CALL
                                    SYMPUT("MV_SHEET_&&I._Q&J.",RESPONSE);
                  RUN;
            %END;         *<--- ENDS THE J INDICATOR LOOP.;
      %END;               *<--- THE I INDICATOR LOOP.;

%MEND;
%M_SET_MACRO_VARIABLES;
```

The preceding section of code reads in all of the values in the Response column of the Excel interface as a series of macro variables named MV_Sheet_1_Q_1, MV_Sheet_1_Q_2, etc.

```
LIBNAME SOURCE CLEAR;
```

Without including the code Libname Source Clear, SAS will not close access to the Excel interface file, preventing the user from opening and editing it until SAS is closed.

A SAS programmer can add whatever code is necessary after the code described above to run the desired reports or analyses, using the created macro variables to control the program and set parameters.

RUNNING A SAS FILE WITHOUT OPENING SAS THROUGH BATCH PROCESSING

The SAS program in the previous section is written in a way that prompts SAS to look within the current folder for the Excel interface file. Saving both the SAS program file and the Excel interface file into the same folder allows for the method of execution with the fewest possible keystrokes and fewest possible errors: batch execution.

Every time a SAS program is opened, there is a possibility that keystrokes will be added (inadvertently or on purpose) that alter the way the program runs. Consequently, we recommend that the user of an Excel interface does not open the SAS file, but instead updates, saves, and closes the Excel interface file, then uses batch processing (typically by right-clicking on the SAS file and choosing the desired batch mode (e.g., Batch Submit with SAS 9.4)) to run the program.

When the program is submitted in batch mode, SAS executes the program in the background, displaying a pop-up window that persists until the program finishes running. This window disappears after processing has been completed, signaling to the user that the program has finished running.

Running a program in batch mode provides three advantages:[1]

1. The first benefit of this technique is that by running in batch mode, the opportunity for the introduction of new keystroke errors is eliminated.
2. Secondly, running the SAS program in batch mode allows programs to run faster because SAS does not display the output and the log to the screen while running.

[1] Technology Services, and VCU. “Using SAS® for Windows in Batch Mode.” Blackboard Upgrade - May 15, 2018 | VCU Technology Services, ts.vcu.edu/askit/research-math-science/sas-for-teaching-research/sas-tips-links--faqs/using-sas-for-windows-in-batch-mode/.

3. A third benefit to batch processing is that the user can use their workstation for other activities while the submitted program runs in the background. While being able to use the workstation for other activities is a valuable benefit, these other activities will compete with SAS for resources (e.g., memory, I/O, CPU), causing SAS to run more slowly.

As one of the main goals of this paper is to describe a simple and efficient approach, it is worth highlighting that the use of batch processing is very elegant, requiring only two mouse clicks for the user, no time is consumed in opening the SAS interface before processing begins.

A somewhat more cumbersome way to run the batch process is by using the SYSIN option in combination with the command prompt or Run dialog box. The syntax for this approach is:

"<path-to-SAS.exe>" -sysin <path-to-the-program-to-be-run> -config "<path-to-SASv9.cfg>".[2]

While the use of the SYSIN option may add somewhat to the complexity to this approach, it may be a viable option in those cases where right-clicking and selecting batch processing is not a viable option.

BENEFITS

The largest benefits to this approach will typically be realized by departments that are of moderate to large size, having enough of a consistent throughput of studies and reports that some degree of specialization of staff is appropriate and feasible.

The most easily quantifiable benefits to implementing the approach described in this paper will be found in cases where an entire SAS programming role can be eliminated through the transfer of analyses or reports manageable by the Excel interface to employees that can be hired at a lower level of compensation. According to the website Glassdoor.com, the national average salary for a SAS programmer in 2016 is seventy-five thousand dollars a year[3] while the average for a data entry position is thirty-one thousand dollars a year[4]. If a non-SAS-using employee can be employed at about a third of the cost of an employee skilled with SAS, then the cost savings that can be realized by a department are more than sufficient to justify the minimal time required to create the Excel interface.

As an alternative to hiring a SAS programmer, companies could elect to train a data entry employee to become a certified SAS programmer. While it is possible for employees to study and take free online classes to prepare for a SAS certification exam, many employees will require courses and textbooks. SAS indicates that using their books is the best option for successful testing. The base and advanced programming guides are each quite expensive while using other test preparation books can cost anywhere between $40 to $80. The exams for both the base and advanced programming certification, if taken in the United States, add an additional $180 to the cost. Rather than go through process of paying for a data entry employee to be certified then having to compensate them for their advancement, using excel as an interface can provide a cost-savings at a low level of risk.

---

[2] SAS® 9.4 Companion for Windows, Fifth Edition. (2016-, November 22). Retrieved from http://support.sas.com/documentation/cdl/en/hostwin/69955/HTML/default/viewer.htm#p16esisc4nrd5sn1ps5l6u8f79k6.htm

[3] Salary: SAS Programmer. (n.d.). Retrieved from https://www.glassdoor.com/Salaries/sas-programmer-salary-SRCH_KO0,14.htm

[4] Salary: Data Entry. (n.d.). Retrieved from https://www.glassdoor.com/Salaries/data-entry-salary-SRCH_KO0,10.htm

Even if it is not possible to eliminate an entire SAS programming role from a department, there are still benefits to implementing an Excel interface:

1. First, transferring the types of studies or reports that lend themselves to the use of an Excel interface away from the most skilled SAS programmers allows those programmers to focus their skills and time on those activities that require higher skill, and hopefully, result in a greater competitive advantage, and/or have a higher profit margin.
2. Secondly, the ability to create fields which only allow validated entries can reduce the possibility for entries being mistyped into a SAS program.

For some situations, the introduction of additional complexity into the Excel interface, for example through the inclusion of macros programmed in Visual Basic, would provide additional benefits, but the method described in this paper allows individuals with only modest facility in both SAS and Excel to achieve significantly improved outcomes without a large investment of resources and time.

## LIMITATIONS

This paper assumes that the user has a license for both SAS Access Interface to PC Files and Microsoft Excel. For those businesses that either do not have the needed SAS license and/or have moved away from Excel to a competitor (e.g., Google Docs or Open Office), the methods outlined will are not a good fit. It is possible to create a similar system using a comma delimited file but the benefits that come from Excel's ability to validate entries will be lost.

While it is possible that future versions of SAS may prevent the Excel Interface from running as intended, SAS has historically created new versions of SAS that are backwards compatible, allowing older programs to run without issue.

The approach outlined in this paper is only appropriate for those types of analyses and processes that are largely standardized, but require input from and the judgment of an employee to choose the appropriate parameters.

There may be cases in which the parameters which the Excel interface file is set up to accept are not sufficient to analyze a particular study and the SAS program fails to run without errors when the batch statement is submitted. To manage the risks associated with this situation, it is recommended that the user of the Excel interface be asked to review the logs and outputs created by the SAS program to look for errors and warnings. In cases where errors or warnings appear, the study/report could be passed on to another employee with SAS programming skill who can write appropriate new code to accommodate the unusual situation.

It is probably not viable for a company to operate without any SAS programmers at all, but this method allows for cost savings by reducing the number of SAS programmers that a company requires.

## CONCLUSIONS

Employee compensation is one of the largest costs for an analytics department. This paper has described one method through which some of these costs can be reduced through the use of employees who are skilled in the use of Excel, but not in the use of SAS. Creating an Excel interface like the one

described in this paper can be an easy first step in increasing the throughput of a department by leveraging commonly available skills.

ACKNOWLEDGEMENTS

The authors would like to thank Steven Denham and Jared Slain for their support and advice.