Paper SA-07-2015

Getting and Staying Organized: Tips for Improving the SAS® Data Analysis/Analyst Experience

Harlan Sayles, University of Nebraska Medical Center, Omaha, NE

ABSTRACT

This paper discusses a variety of simple tips that users can use to be better organized and more efficient SAS users. Main points of emphasis include use of the autoexec.sas file, file structure organization, organization of the user's SAS program files, including use of the %INCLUDE statement, options, the Format procedure, and code organization, and the writing of good, meaningful comments. The tips and recommendations are not restricted to any specific software or version of SAS. The targeted audience ranges from newer SAS users to those who have been using SAS for a little while and are looking for information about how to improve their skill and organization.

INTRODUCTION

There are a number of relatively simple things that most SAS users can do to help make them better, more efficient and more organized SAS programmers. From the moment that they first open the SAS program and SAS automatically runs the autoexec.sas file, to starting a new project and coming up with names and structures for files, to going back and digging through old code and data sets to try to modify some code or just plain figure out what they did months or years earlier, there are things that can be done to make all of these processes easier.

THE AUTOEXEC.SAS FILE

There is likely, although not necessarily, a file somewhere on your computer called autoexec.sas. The purpose of this file is to provide SAS with a list of statements that you want SAS to execute every time you use SAS. Using an autoexec.sas file is one of several ways that you can customize your SAS experience. Other ways include modifying your SAS configuration file and template modifications. The SAS configuration file is a necessary file that SAS runs prior to running your autoexec.sas file and which can be used to tell SAS which autoexec.sas file you want it to run if you happen to have more than one. Template modifications affect how SAS presents your output by adjusting styles, fonts, colors, content, etc. and can be included in your autoexec.sas file. If you modify the configuration file, you can tell SAS to run any file as your autoexec.sas file, although in the interest of simplification and avoiding confusion, I recommend keeping the autoexec.sas name.

When you open SAS without a modified configuration file telling SAS specifically which autoexec.sas file to run, the program will look for a file with the default name (autoexec.sas) in one of a few locations and run it if it exists. SAS will search for a file in the following locations and run the first file called autoexec.sas that it finds: the current folder, paths defined by the Windows PATH environment variable (for PC based SAS applications), the root folder for the current drive, and the folder that contains your sas.exe program file. Most of these locations can be a little difficult to find, but since all of my programs are stored on my C drive and therefore my C drive is always my current drive, I find it works well to store my autoexec.sas file in C:\. If you have multiple drives on your computer, you may be better off finding your sas.exe file and storing the autoexec.sas file in that folder. When SAS finds an autoexec.sas file and it does run, you should see a message in the SAS log that looks something like this:

```
NOTE: AUTOEXEC processing beginning; file is C:\autoexec.sas.
```

The autoexec.sas file is a great place for global options that you want set a certain way, modifications to standard templates, %INCLUDE statements that make references to other .sas files that you want SAS to run, LIBNAME statements to define libraries of files that you use frequently, and possibly some common format definitions defined using the Format Procedure. Any time you discover a new system option that seems helpful to you or create a new format that you plan on using repeatedly, rather than writing it into each program that you write, you should put it in your autoexec.sas file.

For those who are not familiar, %INCLUDE statements are essentially links to other SAS files that you want SAS to run. You could get the same effect by copying and pasting the contents of the other SAS program files directly into your current file, but it's cleaner and easier to use an %INCLUDE statement. This also makes it easier to update your code later as illustrated by the following example. I actually use SAS on several computers that are all part of the same network, so the autoexec.exe file saved in C:\ on each computer I use contains only the following single

%INCLUDE statement telling SAS to run my primary autoexec.sas file that I have saved on a network drive:

```
%include 'u:\projects\autoexec.sas';
```

By doing this, I only need to update the one file on the network drive (U:\projects\autoexec.sas) to update the commands run during the autoexec.sas portion of the startup on all of the computers I use.

FILE ORGANIZATION – NAMING AND STRUCTURE

After you've been using SAS for even a moderate amount of time, you will likely find that you have dozens, hundreds or even thousands of SAS program files, log files and various output files. Without good organization and file naming, it can quickly become a Herculean feat to try to find anything. To avoid never being able to find any of your files again, I recommend using a good file structure and giving meaningful names to your files.

Depending on the size of your project, you may want to use a single folder or a whole set of folders and subfolders. Sometimes it isn't clear at the beginning which of these is a better choice, but you can modify things later and this will be easier to do if you are organized from the beginning. For simple projects, with one or two raw data sets and a relatively small number of outputs (other data sets, analysis results and graphs) a single folder is fine. For large projects with many data sets or many outputs, you may want to use a whole set of files such as one for requests, analysis plans and other project protocols, one for raw data sets (which we NEVER modify), one for analysis data sets that you created from the raw data sets, one for your SAS programs, one for results from SAS, and one for drafts and final products (e.g. presentations and manuscripts) that come from the project.

Within your file structure, it is much easier to find things later if you give your files meaningful names! Incidentally, this applies to all files, not just SAS files. Instead of giving your project folder a name like "Smith_Project" call it "John_Smith_New_Cancer_Drug_Trial_June2015". Use a similar level of detail for your SAS files and any data sets that you create so that you don't end up with dozens of files on your computer just called "Data", "Results", or "Analysis".

ORGANIZING YOUR SAS PROGRAMS

Once you have your autoexec.sas file set up the way you want and you have your folder structure arranged with good names for all of your files, it is time to start writing your SAS code and there are several organizational steps that you can take to improve the ease of reading and referencing your code.

IN THE BEGINNING

The beginning of a SAS data analysis file is a good place to put all of the things that you want to run every time you open the file. This shouldn't be confused with things that you want to run every time you run SAS, those things should go in your autoexec.sas file. This space is for things that are project or subproject specific. If you have certain system options or output options that you want to use for your project, put them here. You can still change your output type (e.g. list, rtf, html, etc.) as necessary throughout your analysis, but anything that might be considered a "global" option should go at the beginning.

All LIBNAME statements should go here along with all formats used in the file. As analysis files become long and begin to contain many parts, users may find themselves needing to define new formats for new variables that they create along the way. Sometimes the data step where a variable is created and the procedure step where a variable is used end up being far apart and users may forget that they defined a format for a variable or which values were assigned which labels. Obviously, this isn't as big of a problem in simple files, but for complex analyses that involve many different sessions, this can become an issue. It you're not careful, you'll end up defining a new format with the same name as an old format and overwriting your old values! Keeping all of the formats at the top of the program in a single PROC FORMAT step, reduces or eliminates this possibility and allows you to quickly redefine all of your formats when you come back to a project and reference your format definitions while working on a project..

Another option for projects with dozens or even hundreds of format definitions is the use of an %INCLUDE statement to point SAS to a separate .sas file that includes your PROC FORMAT statement:

```
%INCLUDE 'u:\projects\this_project\my_formats.sas';
```

Contents of "my_formats.sas":

```
PROC FORMAT;
    VALUE firstformat 1="Value for 1" 2="Value for 2";
```

```
    VALUE secondformat 1-100="First Range" 101-200="Second Range";
      .
      . (Possibly hundreds or thousands of other format definitions go here)
      .
    VALUE $lastformat "Y"="Yes" "N"="No";
RUN;
```

Now for a quick side note about formats and how they work. When you assign a format to a variable in a DATA step using a FORMAT statement, SAS will remember that the assigned format goes with that variable. If you save your data set, SAS saves the format assignments, but what SAS doesn't save are the format definitions. Say you assign a format with the following statements:

```
DATA mylib.js_cancer_analysis; SET mydata;
    FORMAT variable1 firstformat.;
RUN;
```

SAS will always remember that the format "firstformat" goes with "variable1" even after you close and reopen SAS. But SAS won't remember the definition of "firstformat" that is given in the PROC FORMAT statement above and that's why you have to rerun your format definitions (i.e. your PROC FORMAT statements) each time you open SAS and want to use those definitions.

The %INCLUDE statement is also very useful for defining any macros that you may use in your program. Rather than cluttering up your analysis file with lengthy macro definitions, you can put these macro definitions in a separate .sas file and ask SAS to run that file via an %INCLUDE statement as needed. Alternatively, you could put this %INCLUDE statement in your autoexec.sas file and then your macros would be automatically loaded and available for use as soon as you open SAS.

Sometimes groups of SAS users who work with the same data files or who work at the same company and want to share programming knowledge with one another will maintain a common set of format definitions and/or macro definitions on a network drive. Each person in the group can then put an %INCLUDE statement in their autoexec.sas file to run this common file and the whole group will then be using the same format and macro definitions.

Last but not least, the beginning of each SAS program file is a good place to put a few notes about the file such as what project this file is for, where the raw data are from and what the general purpose of the program is intended to be.

DATA MANIPULATION AND ANALYSIS

Nearly every project will require the user to perform some manipulation of a COPY of the raw data prior to analysis. I emphasize "copy" again, because you should never alter your raw data file for any reason whatsoever. Any changes should be made in SAS with SAS code so that the changes can be documented and tracked via the program and easily undone if you find that you made an error in your "corrections". For more complicated projects, there may be many input data sets and a large amount of manipulation may be required to get to a final analysis set. In such cases, good planning and organization is imperative to making sure that everything gets merged together, manipulated, and arranged correctly.

To the extent possible, data manipulation should be done in a straightforward, logical, step-by-step manner that is clear and easy to follow using a series or several series of temporary data sets. Temporary data sets, with numbering to track their order, are recommended to prevent filling your files up with a bunch of interim data sets that can be confusing later and have no practical use. Here is an example of several DATA steps in a row which create sequential temporary data sets with a final DATA step to save the data file that will be used for analysis. You'll notice that I like to indent lines that are part of the same step and I also like to put related statements on the same line. I find that this makes the code easier to read and keeps the files from getting any longer than necessary.

```
DATA mydat1; SET mylib.rawdata;
    IF id = "B4005" THEN DO; group = 1; segment = 2; END;
    IF group IN(1 2);
RUN;
DATA mydat2; SET mydat1;
    IF group = 1 AND segment = 1 THEN newgroup = 1;
    IF group = 1 AND segment = 2 THEN newgroup = 2;
    IF group = 2 AND segment = 1 THEN newgroup = 3;
    IF group = 2 AND segment = 2 THEN newgroup = 4;
```

```
RUN;
DATA mylib.js_cancer_analysis; SET mydat2;
    LABEL newgroup = "New Group Variable";
    FORMAT var1 var2 firstformat.; newgroup ngfmt.;
RUN;
```

For ease of reading and understanding, DATA steps should be ordered in the following manner: variable creation/manipulation, labeling, formatting. Do all of the work of generating, manipulating, recoding, and reordering your variables first and then conclude your DATA steps with a single LABEL statement to label all variables that need to be labeled and a single FORMAT statement to assign formats to all of the variables that need to be formatted. The previous example, which was for illustrative purposes only to show the creation of sequential data sets, could actually be done with a single DATA step using the guidelines above:

```
DATA mylib.js_cancer_analysis; SET mylib.rawdata;
    IF id = "B4005" THEN DO; group = 1; segment = 2; END;
    IF group = 1 AND segment = 1 THEN newgroup = 1;
    IF group = 1 AND segment = 2 THEN newgroup = 2;
    IF group = 2 AND segment = 1 THEN newgroup = 3;
    IF group = 2 AND segment = 2 THEN newgroup = 4;
    IF group IN(1 2);
    LABEL newgroup = "New Group Variable";
    FORMAT var1 var2 firstformat.; newgroup ngfmt.;
RUN;
```

In some instances it is best to keep all of the data manipulation separate from all of the data analysis. You will sometimes need to recreate your analysis data set when you receive an updated raw data file or when you discover the need to create new variables or manipulate other variables needed for your analyses. It is easy to recreate your analysis data set by simply rerunning all of the data steps when they are organized into a single, simple block of code.

Sometimes you do need to create a subset of your data or make other changes that you don't want to make to your entire analysis data set (at least not permanently). In this instance, it does make sense to create a temporary data set from your analysis data set right next to the PROC steps that will use this data set.

```
/* We need to get a frequency distribution of the number of patients who appear
    each unique number of times in the data set (e.g. X patients appear once, Y
    patients appear twice, Z patients appear thrice, etc.). We can do this by using
    the Frequency Procedure to generate an output data set of patient ID's with
    their number of appearances. We can then run PROC FREQ on the variable "Count"
    in this new data set to get our answer. */
PROC FREQ DATA = mylib.js_cancer_analysis; TABLE id / NOPRINT OUT=id_freqs;
RUN;
PROC FREQ DATA = id_freqs; TABLE count;
RUN;
```

In either case, it is good to keep notes about what you did and why, which brings me to my final topic.

/* COMMENTS!!! */

Following exhaustive research (in my own files and those of others) it has been concluded that as of this writing, no SAS code has ever been written that included too many comments. On the other hand, much code has been written that includes far too few comments. Good comments should include all information relevant to what you are trying to do such as when you are doing it (dates), who asked you to do it, what they asked you to do (copy in emails with their exact words if that helps), and what you expected the code to do (what was the purpose of this analysis). It is vital, especially for more complicated analyses, to write out an explanation in plain language of exactly what you expected the code to do, why you were running it, and what you are going to do with the results. This doesn't have to be overly complicated, but the more detailed you are, the more your future self will thank you when you come back later and try to figure out what happened. Here is an example of code that contains a sufficient amount of comments:

```
/* This analysis uses data from the July 2015 version of the Rheumatoid Arthritis
```

```
    Outcomes data set. */
/* Email from John Smith received Sept 10, 2015:
    Hi Harlan,
    Can you please use the attached data for biomarker LMK14 to see if it
    correlates with any of our disease activity measures (DAS-28, ESR, CRP,
    joint counts, Sharp score)?
    Thanks, John */
/* John plans to submit this work as an abstract to the Arthritis conference in
    Miami. His abstract deadline is October 22. */
/* I will look for possible associations using both Pearson and Spearman
    Correlation coefficients. */
PROC CORR DATA = mylib.js_lmk14_data PEARSON SPEARMAN;
    VAR lmk14 das28 esr crp tjc sjc sharp;
RUN;
```

While it may seem like this is overly wordy and not necessary since most of this information is either obvious or recorded elsewhere, it will make it very easy to come back to this code when John is preparing his conference paper and figure out exactly what was done and why. The following is an example of the same code without sufficient comments:

```
/* For John's abstract. */
PROC CORR DATA = mylib.js_lmk14_data PEARSON SPEARMAN;
    VAR lmk14 das28 esr crp tjc sjc sharp;
RUN;
```

This may seem like enough information until you realize that over the course of a single year, John actually submitted 14 abstracts to three professional meetings and five journals using five different sets of rheumatoid arthritis data!

CONCLUSION

Making good use of your autoexec.sas file, having a good, well-organized file structure with meaningful names, writing clean, concise, neatly arranged SAS code, and providing yourself and others who may see your code with plenty of comments are just a few of the ways that you can become a better and more efficient SAS programmer.

It should be noted that just like most things in SAS (and in life, really) there is almost always more than one way to do something, and in some cases it can be difficult to say which way is the "right" way or even the easiest way. These recommendations are simply the methods that I have found useful for keeping myself organized over the years and hopefully they are at least somewhat helpful for you too.

ACKNOWLEDGMENTS

Thank you to David Kadonsky and Elizabeth Lyden who provided valuable feedback on early drafts of this paper and thank you to everyone who has helped me to become a better SAS programmer. Your patience and guidance have been very much appreciated!

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at: hsayles@unmc.edu