PharmaSUG 2018 - Paper BB-04

Seeing the Forest for the Trees: Part Deux of Defensive Coding by Example

Nancy Brucken and Donna E Levy, Syneos Health™

ABSTRACT

As statisticians and programmers, SAS® is part of our daily life. Through assessing patterns, data quality, programming datasets, analysis displays or developing simulations, we need to determine the best ways to conduct our daily work, allowing us to see the forest for the trees. This paper provides guidance on quality defensive programming, efficient coding as well as good programming concepts. Programming no no's will also be discussed. The concepts discussed will allow us to navigate through the trees --- that is, seeing the trees for the forest. We may have been programming in SAS for weeks, months, years or decades. Regardless, we should continue to expand our skills and continue learning and updating our techniques. With this paper, we will provide reminders for paths lost in the past, as well as new tips to help us clear the brush from the trail. This paper is part deux of Defensive Coding by Example (Brucken and Levy, 2015), quenching our thirst for adventure in the great SAS hinterland.

INTRODUCTION

Just as SAS has evolved and changed over time, we as statisticians and programmers need to continue honing and sharpening our SAS tools. In addition, sometimes we need a bit of a refresher on concepts or programming techniques that we once knew; that perhaps we have forgotten or have dulled over time. As in our last paper (Brucken and Levy, 2015), we will discuss and provide programming situations including examples that include techniques to improve our programming plan of attack, programming methods and techniques. The areas of focus are related to efficient coding (tips on improving the quality and productivity of your code), good programming concepts (increasing the readability of your code as well as continuing to improve your SAS skills) and of course our favorite/non-favorite programming no no's (no need for a description). Our solutions may not be the only solutions to the problems described and though we hope what we present is a better option; our goal is to show the importance of continuing to look for other coding options to address a programming problem.

EFFICIENT CODING

As the definition of what constitutes a large dataset continues to increase over time, and even with increases in machine processing power, it is still important to write programs that make efficient use of both system and programmer resources. Adopting techniques such as minimizing the number of passes a program makes through an entire dataset, or modularizing and documenting your code, can go a long way towards making your programs run quickly, and be easier to maintain for both you and other programmers.

USE OF MODULAR CODE

In a recent webinar, the presenter indicated that as a programmer, they try to code in a modular format. This way they can have a greater understanding as to what is occurring in the smaller piece of code, as well as be better able to debug the code. Furthermore, modular coding allows for the code to be more easily moved from one program to another, and allows for increased adaptability through the use of macro variables. This type of coding and packing allows for the same code to be used multiple times leading to reduced debugging and reduced redundant code. If the code is specific to a task or two, modular code allows the code to be reused and more flexible in the future.

Before you start coding, organize your thoughts (and your program) in a logical order, and think about what pieces of code you are writing might be easily used in other places in your work. It's also far easier to test your code in smaller pieces as you go along, rather than waiting until the end.

Here is an example of some modular code used in Wicklin (2013b).

**** Example code 1.A;

```
%MACRO RandExp(sigma);
  ((&sigma) * RAND("Exponential"))
%MEND;
```

As the code can be used for time to event simulations for the time and censoring variable, making this piece of code a macro, thus modular makes sense since the simple piece of code will be used multiple times per simulation record.

MINIMIZE PASSES THROUGH DATA

Simple things such as minimizing the number of times that a program processes an entire large dataset can make a big difference in processing times. Only sort your data when necessary, and use WHERE and KEEP statements on your input datasets when possible to reduce the number of variables or records and dataset size before processing. Remember that a WHERE statement operates on the data as it is being read into the Program Data Vector (PDV), so records that you don't want never make it into your input dataset. An IF statement operates on the records already stored in the PDV, so everything from the input dataset is read in before any subsetting is performed. Placement of WHERE and KEEP statements is also critical- be aware of whether they are operating on the input or output dataset, as that can make a big difference in execution time. Summarize the data as early as possible- that also reduces the size of the dataset you are utilizing.

Here are two examples, each using the same test dataset consisting of 1,000,000 records and 50 variables. In the first example, the IF statement forces SAS to read everything from the input dataset before subsetting, and the KEEP statement in the DATA step operates on the output dataset.

Example code 2.A:

```
69         data subset1;
70           set new;
71           if x1 < 100 or x2 < 100 or x3 < 100 or x4 < 100;
72           keep x1 x2 x3 x4;
73         run;

NOTE: There were 1000000 observations read from the data set WORK.NEW.
NOTE: The data set WORK.SUBSET1 has 49 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           1.40 seconds
      cpu time            0.49 seconds
```

In the second example, the WHERE statement subsets records as they are read into the PDV, while the KEEP option on the input dataset causes only those variables listed to be created in the PDV.

Example code 2.B:

```
75         data subset2;
76           set new (where=(x1 < 100 or x2 < 100 or x3 < 100 or x4 < 100)
77                    keep=x1-x4);
78         run;

NOTE: There were 49 observations read from the data set WORK.NEW.
      WHERE (x1<100) or (x2<100) or (x3<100) or (x4<100);
NOTE: The data set WORK.SUBSET2 has 49 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.51 seconds
      cpu time            0.48 seconds
```

Related to this topic, there appears to be a philosophy among many companies that summary tables should be programmed cell by cell. I have seen many programs where each treatment group is processed separately using a macro variable, and then the columns merged together at the end. Not only does this approach require multiple passes through the dataset, but it can be extremely difficult to debug, and nearly impossible to modify in case of a subsequent request to add or remove information from the table. SAS provides BY-group processing in nearly all procedures, and many tables can be generated by passing all of the variables needed for the table through a single PROC UNIVARIATE (or PROC MEANS or PROC SUMMARY) call, by treatment arm and any subgroup variables, and then transposing the results as needed. The summarized dataset from the procedure call is small, and multiple transposes run quickly at this point. It is also easy to add or remove variables from the procedure call.

MINIMIZE I/O OPERATIONS

One thing that slows SAS down is reading from the source data, regardless of whether it is stored in SAS datasets on disk, or in an external database. SAS runs much faster when reading from memory. We recommend that you structure your programs to read in your source data once, instead of repeatedly going back to retrieve records.

EFFICIENT PROCEDURES

With increased computer power have come increased dataset sizes. Clinical trials have been growing larger, as have other sources of information, such as grocery store and credit card data. Regardless of increased computation power, programmers and statisticians alike should be looking at improving computation skills as well as assessing new procedures that improve efficiency within the program. These updated skills will save valuable run time. For example, consider using the HPGENSELECT procedure instead of PROC GENMOD. While GENMOD is useful for moderate to large data, HPGENSELECT is useful for analysis of large to massive data (Rodriguez, Gibbs and Tobias, 2017).

PROC GLMSELECT, as compared to PROC REG, also provides some increased efficiencies related to addressing computational demands of large datasets as well related to screening and selection methods reducing the number of regressors and dataset size. Rodriguez et al. (2017) have provided some excellent details on updated features and newer procedures available in SAS that improve coding efficiencies.

With datasets getting larger and larger, there may be a more efficient procedure that may not bring your code to a screeching halt (or at least a painfully slow walk). This will keep your boots shiny and new as well as provide new tools and techniques for future analyses. Rodriquez, Gibbs & Tobias (2017) did a nice write up on updated and new procedures that we should all read. I am sure there are other similar papers. Start searching.

PROGRAMMING EFFICIENTLY

We statisticians and programmers tend to be an independent bunch. We thrive for the challenge. Enjoy the gauntlet being dropped. Ehhhmmm. Sorry about that --- got a little carried away. Sure we enjoy the challenges and the creativity of our job. That is one of the things that keep it interesting, right? However, sometimes we need to call mercy. Sometimes we need to call in the paratroopers, but not for the reasons that you might think. Sometimes we need to call in the cavalry not to program, but to talk through our problems. Discuss our programming problems, that is. This will allow you to talk through your programming problem step by step with another individual (for us, this is often all that we need). If the individual you are talking to has worked on a similar problem, then they can provide some guidance. If they have not worked on a similar problem, then you can both work through the problem and brainstorm possible solutions. This can be a win for both of you, as this will solve your current problem, and whoever you are talking to will be able to address the problem in the future. Win. Win.

Another scenario- you have just been assigned to a new project. How exciting. The previous programmer has started many of the programs. Awesome. How exciting. You pull up the previous programmer's code and open it up. Danger. Danger. Danger. Steep cliff ahead. No comments. No

white space. No indenting. No problem. That can be fixed. You diligently go through the code and clean it up to be more readable. However, that is not the end to your woes. The code does not seem to be doing what is expected. Time for some good weeding. Line by line, you go through the code. Sadly, no success. There is also some code that you do not completely understand, but you press on. However, to no avail. When do you call 'when'? What suggestions can be provided so that you get back onto the right path?

This sadly needs to be a "go with your gut" decision. At some point, you have to decide to keep on persevering with the current code or start over. Yes, this is a difficult call, but you definitely need to consider whether the issue you are finding in the code will resolve itself, or will only lead to another issue. Perhaps having someone else take a look at the code might provide more insight. Sometimes walking away from the computer can lead to a clearer solution. This leads us back to our original comment that as we are writing code, we all need to think we are writing code for someone else. We are ALWAYS writing code for someone else. That 'someone else' may be a colleague, a future colleague, a reviewer, or a person in regulatory. That 'someone else' may even be ourselves, in a few days, weeks or months. We are ALWAYS writing code for someone else. Providing someone code should be a gift and not a detriment to the new programmer and the project. A gift, with a big red bow. If you remember this every day you sit down at your computer, then hopefully the programmer taking over your project will appreciate your gift, instead of cursing you. Try to pleasantly surprise your colleagues. White space, indentation and comments go a long way.

And finally, walk away. This is probably the most important piece of information we can tell you. Walk. Away. Sometimes we get so bound and determined while working on a project, that we are a detriment to ourselves. We start down a rabbit hole and keep on going, even though our path seems to be heading in the wrong direction. So our tip to you is: Walk away. Not permanently of course. However, sometimes a walk around the office, block or park might help clear your mind. This little break might help you clear the path in the right direction. Walking away even for a minute or two might lead you back to the path faster than if you stay at your desk, literally and figuratively banging your head. ***Walk. Away.***

GOOD PROGRAMMING CONCEPTS

A scout's motto is "Be prepared". Before we head out for a hike or camping trip, we think about the weather and site conditions, what we plan to eat, how we plan to cook our feast and appropriately pack the needed gear. We also keep our equipment up-to-date and in good condition, for a more enjoyable experience.

PLAN YOUR HIKE

Like camping and hiking, we should plan our code before we start writing our code. Planning your code helps you process where you are and where you are going, as well as the correct sequence of needed coding variables in between. As programmers, we need to plan our path before we start programming. For example, we need to determine the variables required for the planned analysis before the coding is initiated. Pseudocode may help in this process, which includes some details about what you plan to pack on your multi-step trip. What you want to pack is based on your identified needs. Plan your hike. If you want to see the waterfalls, you need to review the map so that you take the right paths to get there. It is important to know where you are going, even before you start each DATA and PROC step.

KEEP YOUR EQUIPMENT CURRENT

We have all heard the old adage "Use it or lose it"; however, as statisticians and SAS programmers, though we may use SAS frequently, we also should expose ourselves to methods and procedures that we may not use every day. If we continue to expose and make ourselves aware of these new methods, the equipment available for our packs continues to expand. Furthermore, though the new methods may not be needed immediately, perhaps in the future you may be reviewing some material and a previous training may be recalled that can be utilized. You may not remember all the details pertaining to the methods, but hopefully you can go back in that brain, look up the notes or start a successful internet search. Keep those tools sharp, and keep expanding your tools. The widget of information remains available so that when you need it, you are not starting from square one. "Be prepared" is our motto.

Periodically looking at the new features in the current SAS release can also be beneficial as there are likely aspects we may have forgotten that may be useful now. In addition, periodically looking back at the same new release information should also be done as there may be some procedures that were not applicable then, but may be useful now.

As SAS programming is a significant part of our daily tasks, we have a tendency to use the same tools over and over again. Using and reusing code can be very efficient, but these tools may become dull, overused, inappropriately used and abused. Think of the first pair of hiking boots you ever purchased. They may have been leather, shiny and new when you purchased them, but after many miles on the trails, they have become weathered, and lost their treads, as well as their ability to keep your feet dry and warm. Thus the boots may protect you from some elements, but they are no longer ideal from protecting you from the rest of the elements. At some point you will likely have to try something new. This situation also applies to your SAS code. New and more efficient procedures, functions or options are being introduced with each SAS version that we should all be aware of --- whether we need them now or not. For example, if you are looking for a random number generator, RANUNI may have been your go to function. However, RAND and RANDGEN have improved statistical properties including a longer period and improved true randomness when compared to RANUNI (Wicklin, 2013b). The random functions have become more random --- exactly what we should want when using a random number generator. At a minimum, learning about new methods may show that in your current analyses, that the old methods are more appropriate with justification.

Along with updated functions, you should also learn about updates to procedures, as well as new procedures that have been released in the latest SAS version. SAS and SAS Support provides highlights prior to, as well as after, the updates are released. As SAS users, we should be aware of these updates, enhancements and new procedures, so we can continue to improve our SAS code and make our SAS code more efficient. SAS Support provides an overview of the changes that we should all familiarize ourselves with on a regular basis. Assessing the differences and the advantages of the newer procedure should be evaluated prior to the analysis to ensure the best and most appropriate analysis methods are being utilized. For example SAS recently introduced the GEE procedure, which has some similar analysis functions when compared to PROC GENMOD. If you have a concern whether your data's missingness pattern is missing at random, PROC GEE may be more applicable. In addition, over the years, model goodness-of-fit tests have been added to PROC GENMOD. As SAS evolves, we as SAS users need to continue to evolve as well.

Optimization and new features in SAS have also been introduced in procedures such as in GLMSELECT (as compared to PROC GLM) which has some additional options for selection methods (Rodriguez, Gibbs & Tobias, 2017). For example, using the stepwise selection method with the DROP option of COMPETITIVE, each stepwise selection is based on the AICC criterion where adjustments are made to minimize the AICC value (SAS, n.d.). We are sure there are plenty of other examples just like this one. As such, when a new version of SAS comes up, or a procedure is updated, be sure to read about or attend a training session about the updates. With the rapid changes in statistics and SAS, it is getting harder keeping up, however a quick search can get you on your way. Keep in mind, newer does not always mean better or applicable. I am sure we can all think of products that have been introduced that have led to a partial or full recall or a lot of frustration for users regardless of the hype. However, being aware of these updates or new procedures may lead to increased code efficiency or more appropriate analysis.

THANKS FOR THE BETA - BEWARE OF THE DEFAULTS

Remember our motto is "Be prepared". While it is nice that SAS had defaults set up for procedures, as statisticians and programmers, we should be aware of the values. We also should not rely on the fact that the defaults will remain the same (remember: Be prepared). As the defaults can change in SAS over time, to be safe we should all actually specify the options, whether they are defaults or not. Not specifying the options in your code now may lead to unknown grief and validation issues down the road. This approach also includes specifying input dataset names in procedures. Someone else modifying the code later on may not notice that the input dataset is not specified in a procedure called in a step after their addition, and the wrong dataset may get pulled in. Remember we are always coding for someone else--- including ourselves.

Here is some SAS code that produces the same output within the same program. However, Example 3.B is more robust in comparison to Example 3.A, as the dataset that will be used is specified (DATA=InData), the significance levels for the confidence intervals produced for the predictive values is specified (ALPHA=0.05), and the sort order lf the levels of the classification variables is specified (ORDER=FORMATTED). While all of these options are specified at their default level, Example 3.B is more informative for the current user, as well as whoever may inherit or review your code in the future.

Example 3.A (no defaults specified):

```
PROC GLM;
  CLASS classvar1;
  MODEL outvar = invar1 invar2 / CLM P;
RUN;
```

Example 3.B (with defaults specified):

```
PROC GLM DATA=InData ALPHA=0.05 ORDER=FORMATTED;
  CLASS classvar1;
  MODEL outvar = invar1 invar2 / CLM P;
RUN;
```

HAVE WE MENTIONED THAT COMMENTS ARE FREE?

We are sure you have heard it before, and you will hear it again and again. Comment. Comment. Comment. Even if you have rules that you use for presentation, including indentation, white space and comments, these habits still may not be enough. Have you ever opened a program where you thought there were too many comments? We are willing to guess no.

For example, on a recent special project, the programming was a bit complicated with multiple simulations, estimates and summary statistics within and across simulations, and I was not able to work on the code regularly. Sometimes there would be a week or so where I could dive in; at other times, the code was set aside for weeks or months at a time. Each time I had to go back to the code, I had to figure out and relearn what I was doing and why. You would think that would have been a lesson for me, but sadly, I had to learn and relearn from my mistakes. For the individual that was reviewing my code, I made their job even harder than it already was. Though I had intentions of cleaning up the code, unfortunately that critical step just never happened. The reviewer politely requested that I add more comments for greater understanding. If only I had done that when I first wrote the code, both of us would have saved a lot of time on our journey. Not only did I make it hard for myself, I also made it hard for my reviewer. There were no big, red bows. You may think that you do not have time to comment, but your future self and your code inheritor or reviewer will thank you. And thank you with time saved in the long run as well.

PROGRAMMING NO-NO'S

Finally, there are some things that SAS will allow you to do, but will probably cause trouble for you later on. This is not meant to be a comprehensive list, but simply a few things we have encountered that may prove to serve as a helpful warning for you, as well.

OVERWRITING DATASETS

How many times have you gone to debug someone else's program and run it all the way through, only to discover that they wrote to the same work dataset over and over again, and you are now going to have to run it one step at a time in order to get to the place you need to start from? Or worse- the dataset you need to look at is created at the beginning, but then overwritten by another step near the end, and you did not notice? Or the programmer simply numbered the work datasets sequentially, but now you need to insert several steps between datasets WORK6 and WORK7, and as you know, SAS does not allow decimal points in dataset names? Give some thought to how you name your work datasets, as those names can help to serve as documentation for the program. For example, if your dataset creates analysis visits, calling it something like AVISIT or AVISIT01 helps to explain its purpose and origination.

This is where planning your code using pseudocode can be helpful. Think about what you want to do before you do it (and perhaps include dataset names). This way, you can feel comfortable, when you go back to update your code, that the version of AVISIT visible in your workspace at the end of the program has not changed since it was created. Plan your path. Follow the map. Be consistent and find a dataset naming convention that works for you (and all the other individuals that you are programming for).

APPENDING CHANGES

Many moons ago, we learned of a programming technique that we would not recommend. On that project, the assigned statistician was taught that when you are working on a program that is fully functioning, and an update is needed, you make the update at the end of the code. This way, you are not messing with code that is working. While we understand the logic behind this suggestion, there are multiple issues with this programming technique.

For example, suppose you need to add another category to a variable. Using this same strategy, the code would be added at the end of your program (even though the original code and categorization would be in another location in your code). That means you are creating a variable and then updating the variable in two locations (ugggh – debug nightmare). That also means you are overwriting your current variable with a variable of the same name, assuming you do not want to do further updates on outputs using the original variable ---a serious programming and debugging no-no. That approach leads to the situation shown in Example 4.A.

Example 4.A (update at the end):

```
DATA bad;
  SET x;
  IF age NE . AND age LE 18 THEN agecat="LE 18 years";
    ELSE IF age GT 18 THEN agecat="GT 18 years";
RUN;

*** other code ***;

DATA final;
  SET almost;
  IF age NE . AND age LE 10 THEN agecat="LE 10 years";
RUN;
```

Once again, we understand the concern with not wanting to mess with the code that was working. However, with careful planning (and saving the original program just in case), the variable can be updated or added in its appropriate location, thus leading to a well-planned out path in the program and well-organized packing of your code, as shown in Example 4.B.

Example 4.B (update where the variable is created):

```
DATA MuchBetter;
  SET x;
  IF age NE . AND age LE 10 THEN agecat="LE 10 years";  *** additional
category added;
    ELSE IF age NE . AND age LE 18 THEN agecat="LE 18 years";
    ELSE IF age GT 18 THEN agecat="GT 18 years";
RUN;
```

If you are running the code over and over again, and thus debugging over and over again over months or even years, in the end, time, energy and frustration would probably be saved by putting the updated code in the correct and logical place- where the original age variable was developed. In the end, keeping related code together will probably save time debugging (assuming you are going to use the code again). This is not a shortcut, but the safest route to quality programming code. While the code provide is a

simple example, the more complex the code, the greater the importance of grouping related variables for the short and long term.

BETA FROM A STATISTICIAN

As a statistician, while I consider myself a competent SAS programmer, I know I do not have mad skills. I hold base SAS certification, and while I have lofty goals of getting other certifications, I expect a lot of studying will be needed. Also as a statistician, I do not consider it my job or my skill set to make outputs look pretty. I tend to code for quality or confirmation, not presentation. This section discusses some lesson learned I would like to share with other statisticians as well as programmers.

CALL IN THE PARK RANGERS

Recently I was working on a simulation where we wanted to make the summary output look pretty. However, no matter how hard we tried, we could not get the bar plot or forest plots to come out the way we wanted. Google did not even save us from ourselves. I am willing to give it the old college try, but at some point you need to call in the park rangers. SAS programmer to the rescue!! After a few short hours, they had our output dolled up and ready to go. As SAS users, we periodically need to remember that at some point we need to stop circling the trail. Sometimes just talking through the problem with someone else will lead us down the correct trail head. Sometimes others have previously experienced what you are dealing with, and can provide some excellent beta to get over the crux of the problem. By all means, give the problem a valiant effort; however, as we did, sometimes you have to put up the white flag, and call in the troops.

BEWARE OF MERGING BLINDLY

While new options and new procedures should be assessed, we also should not give up on “Old Faithful” code. Some coding habits have simply been tried and true, and are considered classic. For example, when you have a dataset and you are merging many records to many records, you must be cautious. Sometimes you might get what you want. Sometimes you might not. Now as statisticians, we may know some basic features of SAS when working with datasets. However, there is one technique that one must know, and that is the use of PROC SQL when merging many records to many records. As a statistician, I am not SQL savvy (though I wish I were), but this is code that I keep handy and accessible. See the coding example below (examples 5.A and 5.B). Of course, the coding does not stop here. Check the records and record counts in the output dataset against the raw data to ENSURE the merge went as planned. Also make sure your key and common variables have the same format, as once again you might not get what you expect or even the correct data at all. Example code 5.A and 5.B is below:

```
data work.patient_treatment;
  input patientid treatment $ date_treatment :mmddyy10.;
  format date_treatment mmddyy10.;
  datalines;
    1 A 1/5/2018
    1 A 2/5/2018
  ;
run;

data work.treatment_drug;
  input treatment $ drug $;
  datalines;
  A DRUG1
  A DRUG2
  ;
run;
```

**** Example code 5.A;

```
data work.merged;
  merge work.patient_treatment work.treatment_drug;
```

```
  by treatment;
run;
title "With merge";
proc print data=work.merged;
run;
```

Error! Reference source not found. shows the output from example code 5.A

With merge

Obs	patientid	treatment	date_ treatment	drug
1	1	A	01/05/2018	DRUG1
2	1	A	02/05/2018	DRUG2

Output 1. Output from a MERGE statement

```
**** Example code 5.B;
title "With Proc SQL";
proc sql;
  select a.patientid, a.treatment, a.date_treatment, b.drug
  from work.patient_treatment a,
       work.treatment_drug b
  where a.treatment=b.treatment;
quit;
```

Output 2 shows the output from example code 5.B

With Proc SQL

patientid	treatment	date_ treatment	drug
1	A	01/05/2018	DRUG1
1	A	01/05/2018	DRUG2
1	A	02/05/2018	DRUG1
1	A	02/05/2018	DRUG2

Output 2. Output from a PROC SQL

CONCLUSION

We hope you never get lost in the woods. Following the tips described in this paper for writing efficient programs, keeping your skills sharp and updated, continually looking to improve your SAS knowledge, and avoiding some common programming no-nos should help you keep your pack manageable, and your feet on the trail.

REFERENCES

Brucken, Nancy. Levy, Donna E. 2015. “Defensive Coding by Example: Kick the Tires, Pump the Breaks, Check Your Blind Spots, and Merge Ahead!.” *Proceedings of the SAS Global Forum 2015 Conference*. Available at http://support.sas.com/resources/papers/proceedings15/3305-2015.pdf.

Hughes, Ed. 2015. “SAS/OR 14.1: Improvements and New Features.” Accessed February 17, 2018. https://blogs.sas.com/content/operations/2015/09/04/sasor-14-1-improvements-and-new-features/.

Rodriguez, Robert N. Gibbs, Phil. Tobias, Randy. 2017. “Step Up Your Statistical Practice with Today’s SAS/STAT® Software.” *Proceedings of the SAS Global Forum 2017 Conference*. Available at http://support.sas.com/resources/papers/proceedings17/SAS0521-2017.pdf.

SAS (n.d.) SAS/STAT(R) 9.22 User’s Guide – Stepwise Selection (STEPWISE). Available at https://support.sas.com/documentation/cdl/en/statug/63347/HTML/default/viewer.htm#statug_glmselect_a0000000241.htm

Wicklin, R. 2013a. *Simulating data with SAS*. SAS Institute, Cary, NC.

Wicklin, Rick.2013b. “Six reasons you should stop using the RANUNI function to generate random numbers.” Accessed February 17, 2018. https://blogs.sas.com/content/iml/2013/07/10/stop-using-ranuni.html.

ACKNOWLEDGMENTS

Thanks to Daniel Quinn for his review and provided examples. Thanks also to our colleagues, including Dan Strieter for a thorough review and comments on the paper, presentation and slide set.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Nancy Brucken
Syneos Health
Nancy.Brucken@syneoshealth.com

Donna Levy
Syneos Health
Donna.Levy@syneoshealth.com