Collaborations in SAS® Programming; or: Playing Nicely with Others

Kristina B. Metzger, Metzger Consulting, Austin, Texas
Melissa R. Pfeiffer, Children's Hospital of Philadelphia, Philadelphia, Pennsylvania

ABSTRACT

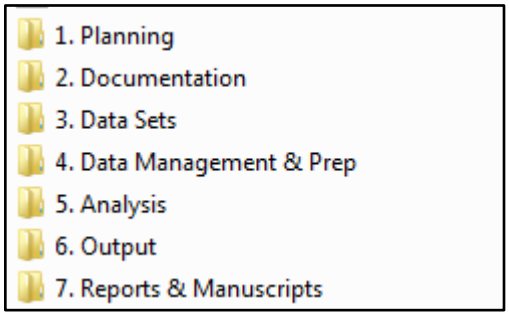
SAS® programmers rarely work in isolation, but rather as part of a larger team that includes other SAS programmers such as data managers and data analysts, as well as non-programmers like project coordinators. Some members of the team -- including the SAS programmers -- may work in different locations. Given these complex collaborations, it is increasingly important to adopt approaches to work effectively and easily in teams. In this presentation, we discuss strategies and methods for working with colleagues in varied roles. We first address file organization -- putting documents in places easily found by team members -- including the importance of numbering programs that are executed sequentially. While documentation is often a neglected activity, we next review the importance of documenting both within SAS and in other forms for the non-SAS users of your team. We also discuss strategies for sharing formats and writing friendly SAS coding for seamless work with other SAS programmers. Additionally, given that data sets are often in flux, we discuss approaches that add clarity to data sets and their production. Finally, we suggest tips for double-checking another programmer's code and/or output, including the importance of confirming the logic behind variable construction and the use of PROC COMPARE in the confirmation process. Ultimately, adopting strategies that ease working jointly helps when you have to review work you did in the past and makes for a better playground experience with your teammates.

INTRODUCTION

Projects involving data and requiring data analysis almost always involve a team of individuals working together. A project team often includes a principal investigator/project lead, a project coordinator, a data manager, and an analyst among others, some of whom may not work in the same physical location. Additionally, projects and project teams may morph over time, which requires that the background knowledge, database development processes, analytic decisions, and interim and final reports can be easily found and referenced. While this concept may seem obvious and intuitive, the process necessary to organize the project and the team is neither. We present several strategies that we have found to be helpful to facilitate an effective and efficient team.

ORGANIZATION; OR: PUT THINGS WHERE PEOPLE CAN FIND THEM (INCLUDING YOU, 6 MONTHS LATER)

A common network with appropriate levels of security is essential to facilitate document sharing across all team members. Team members should have the ability to access project-related documents in a folder (or set of folders) within this shared network. While each team member will have varying needs for accessing, modifying, and creating certain documents, other members will certainly need to be able to find these documents at some point. Developing a standard folder organization can go a long way to rapid document retrieval. For each project, the folder should include all the documentation and files used for that project, such as the research proposal, background documents, planning documents, data documentation, SAS programs, output files, reports, and abstract and manuscript drafts and submissions. We suggest creating a standard set of sub-folders to help organize all of these documents. These folders could include the following categories: Planning, Documentation, Data Sets, Data Management/Preparation, Analysis, Output, and Reports/Manuscripts (Display 1).



Display 1. Example of Project Sub-Folders

If there are collaborators who do not have access to the shared network drive, such as consultants working from a different location and without VPN (virtual private network) access to the common drive, we recommend using a document sharing site, such as DropBox or GoogleDrive or other secure joint location. For files that do not require high levels of security, the use of document sharing sites can reduce the size of emails and may minimize version control issues, as multiple copies of the same document do not need to be circulated.

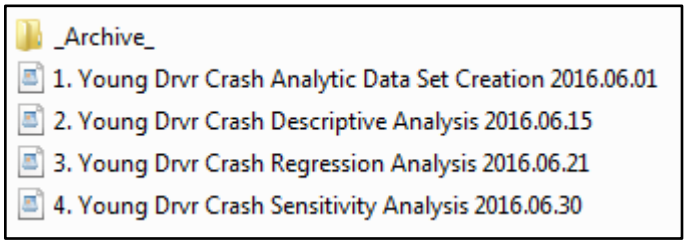
In addition to having a standard method of organizing and sharing documents, we suggest developing a standard manner of naming files. Files should be named as descriptively and concisely as possible. Because team members may be working on multiple projects at any one time, a file name that includes some indication of the project is beneficial. For instance, if the project involves the examination of crash outcome data for young drivers age 16 to 21, file names could include the prefix “Young Drvr Crash.” This naming convention can be used to easily differentiate documents from a project involving crash outcome data for drivers over age 65 named with the prefix “Older Drvr Crash.”

Documents and programs are never as final as you think they are. How many times have you found yourself naming a document “...final final” or “...final for real”? In order to clearly indicate changes over the course of a project, include appropriate information regarding versioning in the file name. We prefer using a date at the end of the file name in the form of “yyyy.mm.dd” – this naming convention keeps like files together and in order by date. In addition to a date to indicate the iteration of a document, you may want to institute the practice of including initials at the end of the document name to identify the last person who worked with the document. This can be helpful both when changes are being made by multiple people over a relatively short time-span as well as when a document is going to remain inactive for several months. Whichever convention is suitable for your group, select one naming structure and adhere to it.

Cluttered folders can be difficult to navigate. Within each sub-folder, include a sub-sub-folder to archive appropriately dated programs, data sets, output files, and other documents when they are determined to be out-of-date, rather than deleting them. With an underscore preceding the name (e.g., “_archive_”), this folder will always be at the top when sorted by name; alternatively, give the folder a name that will put it at the bottom (e.g., “ZZZ Archive”). This keeps the main folder well-organized and minimizes the chances of someone inadvertently referencing or modifying an out-of-date document. When the project is completed, these archived folders can be easily and quickly deleted if drive space is a concern.

Within data management and analysis folders, we suggest numbering SAS programs that were created and/or executed sequentially. Numbered programs allow you or another programmer to easily discern the order the programs must be run in order to achieve the desired output in the future. For example, the program used to create the analytic data could be named “1. Young Drvr Crash Analytic Data Set Creation.sas.” Then the program used to descriptively analyze the data could be called “2. Young Drvr Crash Descriptive Analysis.sas.” The program used to perform regression analysis could be called “3.

Young Drvr Crash Regression Analysis.sas" and the program used to conduct a sensitivity analysis could be called "4. Young Drvr Crash Sensitivity Analysis.sas" (Display 2). For a manuscript, report, or other publication, we also recommend saving a separate analytic program that includes only the numbers that are included in the text, tables, and figures, but nothing more.



Display 2. Example of Naming SAS Programs Descriptively

These guidelines regarding network folder and file organization can make your project more resilient to changes in team members over time. If everyone knows where to find things, anyone on the team can orient new team members to the file structure. Additionally, this will make your life easier because you will identify where to find things when you need to find them several months in the future.

DOCUMENTATION; OR WRITE DOWN WHAT YOU'VE DONE AND WHY

As we've mentioned, projects are fluid and original team members leave while new team members join. In order to make these changes easier, having written documents that describe the project are essential. Project documentation is critical at the data management and analysis level as well. The data manager should create written documents that describe the processes and decisions that impact data selection and variable creation. This should be done in a separate document than your SAS programs (e.g., a Word document) since some members of the team may not have access to SAS or be able to read SAS code. Though it takes time to create, this documentation will save you time later when you wonder about your rationale for decisions that were made. This document should be detailed enough to explain decisions and processes that are not obvious, but not so extensive as to become overwhelming and unreadable. Clear sections for different stages of the process are helpful. Consider including path information for important programs, data sets, and output - you may want to just indicate the folder for these files rather than the full file name so that you don't inadvertently reference out-of-date files.

Data dictionaries are essential for shared data sets. At a minimum, the data dictionary should include each variable name with a descriptive label and associated format, as well as the overall number of variables, the number of observations, and the date the data set was created. Data dictionaries may also include frequencies for key variables. Create data dictionaries for collaborators to access outside of SAS, such as in a Microsoft Excel file (Display 3).

Data dictionary for youngdrvrs_20160601

Data Set Name	EXPAND.youngdrvrs_20160601
Last Modified	Wednesday June 1, 2016 03:51:12 PM
Observations	148284
Variables	174
Sorted	YES
Sortedby	patient_id

#	Variable	Type	Len	Format	Informat	Label
1	patient_id	Num	8			Unique ID for patient (constructed)
2	dob_master	Num	8	MMDDYY10.		Date of birth
3	birthyear	Num	8			Patient year of birth
4	sex_master	Char	1			Sex (character)
5	race	Num	8	RACEFMT.	8	Patient race
6	ethnicity	Num	8	ETHFMT.	8	Patient ethnicity
7	race_ethnicity	Num	8	RACEETHFMT.		Race/ethnicity combined (unknown is missing)
8	age_lastenc	Num	8			Age at last encounter (any) date
9	ageint_lastenc	Num	8			Age at last encounter (any) date, integer

Display 3. Example of Data Dictionary, Created in SAS and Output to Microsoft Excel

Thorough annotation of each SAS program is essential for continuity and consistency with multiple SAS programmers on a project. At the beginning of each program, include a description that explains general information that may be important to the team. Use a standardized structure that includes information such as who wrote the program or has primary responsibility for it, when it was written and subsequently modified, a broad description of what the program does, which data sources it accesses, and any data sets and/or output that it creates (Display 4).

```
1  /** SAS PROGRAM NAME:    2. Young Drvr Crash Descriptive Analysis 2016.06.15.sas
2      WRITTEN BY:          Kristi Metzger
3      STARTED:             06/01/2016
4      LAST UPDATED:        06/15/2016
5      PURPOSE:             Descriptive Analysis of Young Driver and Crash data
6
7      Uses dataset:
8      K:\Projects\Young Drvr\3. Data Sets\YOUNGDRVRS_20160601
9         Driver-level data (one obs = one driver)
10        Data on drivers born 1987-1995 with licensing and crash information from 2004-2012
11        --> created in <1. Young Drvr Crash Analytic Data Set Creation 2016.06.01.sas>
12
13     Produces output file:
14      K:\Projects\Young Drvr\5. Analysis\Output\Young_Drvr_Descriptive_2016.06.15.xlsx **/
```

Display 4. Example of SAS Program Header

Within the program, comment copiously by thoroughly describing each part of the program. Comments should include brief descriptions of the purpose of each data step or procedure (e.g., Display 5a, lines 4-5). If code for checking or verification purposes is removed, retain a comment about what was done so that other programmers do not need to wonder if the check needs to be done (or to remind yourself what you checked!) (see Display 5b, lines 62-63). For example, if you've checked that dates for one variable are never before dates for another variable but do not want to keep that code in your final program, include a comment such as "Dates in variable Y are never before the dates in variable X (verification code removed)." Alternatively, retain the code in your program (along with the description of its purpose), but comment it out so that it is not rerun unless desired.

When writing your program, use white space – tabs and carriage returns – generously to make your code easy to read (Displays 5a and 5b). Code is much easier to read with blank lines between procedures and data steps. Liberal use of tabs within procedures or data steps can also make programs easier to read – particularly when using do-loops and arrays. Aligning similar parts of code over several lines, such as when using a series of IF-THEN/ELSE statements or within ATTRIB statements, can help others (and yourself!) read your code.

When multiple people are included on a project, be aware that the assigned path to your network drives and folders may differ from other programmers. Consider creating LIBNAME statements specific for each programmer to use, and commenting them out when not needed.

```
1  data youngdrvrs_20160601 ;
2      set alldrivers_indicator ;
3
4      *   driver seatbelt use - includes codes for "lap belt and harness"
5              and "airbag & seatbelts" but not "lap belt" only or "harness" only;
6      if occsafused in ('04','09') then belted=1;
7      else if occsafused in ('01', '02', '03', '05', '06', '07', '08', '10', '99')
8          then belted=0;
9      else if occsafused in (' ', '00') then belted=. ;
10
11     *   ages of all passengers ;
12     array pass{*} p_age: ;
13
14     *   set counter variables to 0;
15     _pass_0to13_ctr=0; _pass_14to20_ctr=0; _pass_21to34_ctr=0;
16     _pass_35plus_ctr=0; _pass_unk_ctr = 0;
17
18     *   if the first passenger age variable is null, all others are as well
19             (no passengers), therefore the number of passengers is 0;
20     if p_age1 = . then passenger_n = 0 ;
21     *   otherwise, count the number of passengers and enumerate each
22             passenger in a counter variable according to their age;
23     else do _i=1 to dim(pass);
24         if pass{_i} ne . then passenger_n = sum(passenger_n, 1) ;
25
26         if 0<=pass{_i}<14               then _pass_0to13_ctr = _pass_0to13_ctr+1;
27         else if 14 le pass{_i} le 20    then _pass_14to20_ctr = _pass_14to20_ctr+1;
28         else if 21 le pass{_i} le 34    then _pass_21to34_ctr = _pass_21to34_ctr+1;
29         else if 35 le pass{_i} < 999    then _pass_35plus_ctr = _pass_35plus_ctr+1;
30         else if pass{_i} = 9999         then _pass_unk_ctr = _pass_unk_ctr+1;
31     end;
32
```

Display 5a. Example of SAS Program

```
33      *   create our age-related passenger compliance variable:
34              compliant (1) if there is only 1 passenger or there is at least
35                  one passenger 35+.
36              non-compliant (0) if there is more than one passenger, no adults
37                  and no one with missing age.
38              unknown compliance (.) if there is more than one passenger, no adults,
39                  and someone with missing age.
40              There should be no codes of 888 in the end (used for data checking);
41      if passenger_n in (0, 1)
42          then passcompliance1 = 1;
43      else if passenger_n gt 1 and _pass_35plus_ctr ge 1
44          then passcompliance1 = 1;
45      else if passenger_n gt 1 and _pass_35plus_ctr = 0 and _pass_unk_ctr = 0
46          then passcompliance1 = 0;
47      else if passenger_n gt 1 and _pass_35plus_ctr = 0 and _pass_unk_ctr ge 1
48          then passcompliance1 = .;
49      else passcompliance1 = 888;
50
51      drop _: p_age: _pass_: p_restraint: ;
52
53      attrib  belted          format = ynufmt.
54                              label = 'Driver seat belt indicator 1/yes, 0/no, ./unknown'
55              passenger_n
56                              label = 'Number of passengers in vehicle'
57              passcompliance1 format = passcompfmt.
58                              label = 'Passenger compliance indicator'
59      ;
60
61  run;
62  *   checked construction of belted, passenger_n, passcompliance1.
63      code removed ;
```

Display 5b. Example of SAS Program

SHARE PROGRAMS; OR: DON'T RE-CREATE THE WHEEL

Whenever possible, SAS programmers working on the same project should share programs or elements of programs. The potential for error is magnified when rewriting or replicating code unnecessarily. A key to successfully sharing programs is to balance understandability and efficiency when coding. As we all know, in SAS there are often several different ways of coding a program to achieve the same outcome. While a more advanced SAS programmer may be able to write short, abbreviated code, a programmer with less experience may require more elaborate or lengthy code. When more efficient coding isn't required (e.g., the data sets are not overly large), the most elegant coding allows others with different skill levels and programming knowledge to understand your program. Additionally, by using concise but descriptive names for data sets, variables, arrays, and macros, your program can be more legible.

In our experience, variable formats have been one of the most critical parts of our data to share efficiently. We found, to our dismay, that traditional format catalogs could not be accessed by multiple programmers on our remote sharing network. Yet we wanted to assign permanent user-defined formats to variables to ensure that the values were interpreted correctly and an incorrect or out-of-date format was not mistakenly applied. We had to be creative and write standard SAS format code that could be invoked at each SAS session. This format code points to an Excel file with user-defined format details used by variables in the permanent data sets. This section of code ends with specific code that releases the Excel file to allow others access to it, either in SAS or Excel (Display 6).

```
1  libname formats excel "K:\Projects\Young Drvr\2. Documentation\Formats 2016.06.01.xlsx" ;
2
3  proc format cntlin = formats.'formats$'n ;
4  run;
5
6  libname formats clear;
```

Display 6. Example of Standard Format Code

Alternatively, we could have written a separate format program that includes this code, plus ad-hoc formats using PROC FORMAT statements. Then, we could use a %INCLUDE statement to point to that program in subsequent data set creation and analysis programs.

We recommend that programmers include the system option:

```
options nofmterr;
```

to the beginning of each of the primary programs to suppress format errors.

Ensure that the log generated by your program is free of errors, warnings, or notes that could be confusing to another programmer. If your program does create messages to the log that may be concerning to another programmer, be sure to include a comment about it in the program. Thus, another person won't spend valuable time troubleshooting a concerning message that you know is not an issue. For instance, creating a new variable by dividing with a variable in which zero is a valid value could lead to numerous notes, but won't stop a data step and may generate valid values. If there is a zero value in your denominator variable, your code will create a message indicating that a mathematical operation could not be performed. If you choose not to revise your code to prevent this message, include a note indicating that, such as "The log message regarding inappropriate division by zero is a result of valid zero values in the denominator. This has been checked and is ok."

We suggest keeping your program's output clean by removing temporary and/or intermediate data sets from your program by developing a standard convention that can be used to remove these data sets. By removing these data sets from your WORK library, you and others can more easily discern the correct data sets for use in subsequent data management steps and in data analysis. If the names of any intermediate or temporary data sets start with an underscore, you can drop them using a name prefix list, such as:

```
proc datasets library = work ;
       delete _: ;
quit ;
```

SHARE DATA; OR: DATA FOR EVERYBODY!

Similar to removing temporary and intermediate data set from your WORK library, we also recommend removing temporary and intermediate variables from your data sets by developing a standard convention that allows for their easy removal. Again, eliminating unnecessary variables from your data sets prevents their inappropriate or inadvertent use by you or another programmer. In order to achieve this routinely, you could adopt both a naming convention that capitalizes on name prefix lists (e.g., anything intermediate and/or temporary starts with an underscore) and a standard of writing code to remove extraneous variables, either within the data statement (see below) or using a DROP statement at the end of the DATA step (see Display 5b, line 51).

```
data new (drop = _: ) ;
       set intermediate ;
run ;
```

The programming team should also agree on naming conventions for similar types of variables to increase understandability and transparency in meanings (in addition to descriptive labels, of course). Naming conventions that we like include using the prefix “Date_” for date variables and using the suffix “_N” for variables that include total counts. Exact age variables can be differentiated from floored or integer age variables by naming them “Age” and “AgeInt”.

Standard conventions for variable attributes are essential to reduce confusion within and across projects and are necessary for data sets that will be combined horizontally or vertically. Standardization of variables both in terms of numeric vs. character variable type and values within variables is critical to consider. Take into account source data sets when choosing conventions for variable values – for example, if you routinely use data from one source (e.g., proprietary electronic health records, standardized surveys), you may consider adopting those values as your standard. Another consideration may be the manner in which the data are used. Any variables that will be analyzed should probably be numeric to make analysis easier. Epidemiologists often like dichotomous numeric variables to have values of “1” (one) and “0” (zero) to allow for easier interpretation in regression models. If categorical variables are to be included in regression models, consider using sequential numeric values starting with “1” (one). If many variables include values that indicate “unknown” or “no response,” use the same values for all of these variables, such as “9” for “unknown” and “8” for “no response” (or several 9’s or 8’s depending on the maximum length of the variable). Some programmers or analysts may prefer to have unknown values coded as null or special missing values (e.g., .A, .b) so that missing values are omitted from certain procedures with the default coding. Coding of categorical variable values should be logical whenever possible, such as using yes=1 and no=0; male=1 and female=0; and lowest quintile=1 to highest quintile=5.

When creating permanent data sets to share with colleagues, be sure to including a creation date in its name. Using a naming convention with date in the format yyyy.mm.dd makes certain that updated versions of a data set will be sorted sequentially in shared folders. When a data set is old or out-of-date, archive these data sets in a sub-folder rather than deleting them. By keeping these data sets, you or your colleagues will be able to access them if necessary for reanalysis or another unforeseen reason.

Finally, when needing to work with permanent data sets saved on the network for your project, write code in your program to immediately copy it to a temporary data set and use that to work from. By using a temporary data set, you won’t lock up the permanent data set in case another program also needs to access it. This strategy also guards against accidentally modifying or deleting a permanent data set in SAS.

DOUBLE CHECK OTHERS’ WORK; OR: FOUR EYES ARE BETTER THAN TWO

One advantage of working with other SAS programmers is the ability for colleagues to double-check each other’s work. This double-check should be done both for the creation of data sets and any analyses that are done, particularly prior to finalizing reports or publishing papers based on these data. One key recommendation is to recreate code used to make data sets and conduct analysis, rather than just re-running the original code. Among other checks, this allows for the confirmation of logic used in the creation of new variables. For example, is this metric for those under 35 years old or those 35 and younger? Is unknown grouped with no or excluded?

Also, when comparing original and re-created data sets, be sure to compare all observations in both data sets, not just Ns. For example, if values change in opposite directions for two observations, total Ns will still match. PROC COMPARE is a useful procedure for this comparison (Pfeiffer 2014).

Easy communication among team members is important in order to effectively validate programs and other project documents. We recommend scheduling regular weekly or biweekly meetings by phone or in person, depending on the physical work environment, in which the team discusses their individual and group progress on the project. These meetings are an important opportunity to talk about project issues that have come up or to ask for assistance. Email is also an important tool for communication for one-on-one discussions. Newer dedicated messaging forums such as Slack have expanded resources that allow continued one-on-one or group discussions that are easily searchable.

CONCLUSION

Working on teams affords many advantages, including learning new ways to code from other programmers, but can also present challenges. The best practices that we present in this paper can facilitate improved communication and collaboration among project team members. Since many projects may include multiple SAS programmers with different skill levels in addition to non-programmers, teams will benefit from standardized processes to organize files in common drives and organize information within files. We stress the importance of documentation within SAS and in other files to allow team members to more easily find, share, and confirm information about the project. We hope that by adopting these strategies, SAS programmers and non-SAS users on your project team will find that working together is easier and smoother than before.

REFERENCES

Pfeiffer, Melissa. PROC COMPARE: What Did I Inadvertently Change? Proceedings of South Central SAS Users Group Conference 2018. Austin, TX. http://www.scsug.org/proceedings/2018-papers/

ACKNOWLEDGEMENTS

The authors wish to thank Granger Huntress for his thoughtful review of this paper as well as Allison Curry for fostering an intellectual, collaborative, and enjoyable working environment. Any mistakes herein are the responsibility of the authors.

CONTACT INFORMATION

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

Kristina B. Metzger
Metzger Consulting
512-825-5543
krisbusmetz@gmail.com

Melissa R. Pfeiffer
PfeifferM@email.chop.edu