PharmaSUG 2022 - Paper AP-072

SAS Standard Programming Practices – Handy Tips for the Savvy Programmer

Navneet Agnihotri, Syneos Health®, Gurgaon, India
Sumit Pradhan, Syneos Health, Gurgaon, India
Rachel Brown, Syneos Health, Texas, U.S.

ABSTRACT

As a programmer, did you ever paused for a moment and pondered that your program code could stand up to the review and re-use by your colleagues (beginners, intermediate or experts)? Ever wondered if the algorithm used in your program and that your programming style would pass the "white glove" assessment?

We've all been there! Every programmer at some point in their career certainly encounters a scenario wherein some critical delivery to a client is planned. It is late in the day and to meet the client deadline, you're unfortunately required to pick up another programmer's code. As you open the program and scroll through it, your heart sinks. You quickly realize that resolving the issue is not a fast fix but will take hours of deciphering. But what shall we do now; we grumble, sigh and either re-write or struggle to clean it up to make it better for the next programmer and future deliveries. Inconsistent and unstructured code affects us all, including the client. Had the previous programmer followed a few key rules, the fix might have been a quick and simple modification. Code is an intellectual tangible asset and it's essential that SAS® users possess the necessary skills to implement "best practice" programming techniques to ensure greater code readability, maintainability and longevity while ensuring code reusability.

This paper summarizes strategies that should be used in your daily programming to effectively adhere to good programming practices and ensure that code is structured, readable, understandable, portable, and maintainable.

INTRODUCTION

Due to increased needs of the pharmaceutical industry and the success of SAS®, the need for a SAS Standard Programming Practice (SPP) Guideline is more important now than ever before. At a time of reduced timelines, limited resources, increase in remote teams, the practice of sharing and re-using code is more prevalent and necessary than ever before. However, in order to make the sharing and reuse of code effective, it is important that the code is easy to read, understand and update. It can also help to drive standards and ensure consistent outputs are produced across studies. If any of these three simple tests fail, then sharing and reuse of code is less beneficial with little or no actual benefit than rewriting it from scratch. Best programming techniques and practices help to clarify the sequence of instructions in code, permit others to read code as well as understand it, assist in the maintainability of code, permit greater opportunity to reuse code, achieve measurable results, reduce costs in developing and supporting code, and assist in performance improvements (e.g., CPU, I/O, Elapsed time, DASD, Memory).

WHAT IS STANDARD PROGRAMMING PRACTICES CONCEPT?

Standard Programming Practices Guideline is a set of rules developed over time that have shown to produce robust programs that not only produce correct results, but are easy to read, understand and maintain over time.

Best practice programming techniques achieve status by being:

- Measurable with quantifiable results.
- Accomplishing stated goals and objectives with complete clarity in program.
- Reproducible or adaptable to specific needs.

ELEMENTS OF STANDARD PROGRAMMING PRACTICES

- Presentable code appearance and structure using modular design, logic scenarios, controlled loops, sub-routines, and embedded control flow.
- Code compatibility and portability across applications and operating platforms.
- Developing readable code and program documentation.
- Applying statements, system options and definitions to achieve the greatest advantage in the program environment.
- Implementing program generality into code to enable its continued operation with little or no modifications.

GENERAL RECOMMENDATIONS

The following general pointers are recommended to be followed while developing SAS programs:

- SAS programmers must use a Software Development Life Cycle (SDLC) Methodology, whether developing complex systems or individual programs, whether manual or automated, and whether new or a modification of some existing project. These steps may be compressed to include only those activities that would be appropriate for a particular project. The basic design of a general SDLC is shown below in Figure 1.

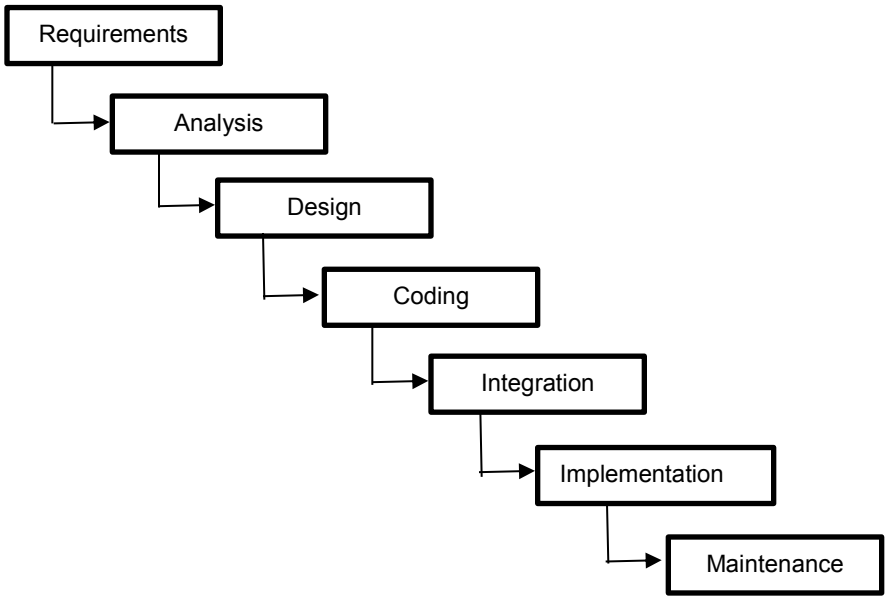


Figure 1 – Typical Software Development Life Cycle

- SAS programs shall be planned that is robust, based on a variety of data, yet supportable when the need arises. Evaluate and use macros as appropriate to maximize efficiency while maintaining quality.
- Each SAS program shall contain a complete, correct, and up-to-date header at the beginning of the program that documents authorship, project, purpose, and revision history in chronological order (including detailed reason for revision).

Here is a skeletal example of a program header. It would be placed at the very beginning of the program.

```
/*************************************************************************************************************
Program Name:     xxx.sas
Client:           XXX
Project Number:   XXX
Purpose:          This program will be used to generate report for missing data values

Author Name:      Navneet Agnihotri
Date of Program:  DDMMMYYYY

Additional Details:

*************************************************************************************************************
Change Log
*************************************************************************************************************
Date:             Modifier Name:      Update:
DDMMMYYYY         Xxx Xxx             Xxxxxxxxxxxx
*************************************************************************************************************/
```

- SAS options, such as MPRINT, MLOGIC, and SYMBOLGEN be used only during code development. Some options might be turned off after testing is completed to avoid lengthy SAS logs when macros are used.
- When working with large data sets it is recommended to use the COMPRESS=YES option. This can reduce processing time up to 50%.
- Place all system interface statements and presentation-controlling procedures that apply to the entire program immediately after the program header. These include, but are not limited to, LIBNAME, FILENAME, ODS, %INCLUDE and system options. If there are interface statements and presentation-controlling procedures that apply to only a certain block of code, then these can be specified just before that block of code.
- Place all compilation statements (ATTRIB, LABEL, [IN]FORMAT, LENGTH, ARRAY, KEEP, DROP, RENAME) at a consistent location within a data Step.

READABILITY & APPEARANCE

SAS programs must be written using a modular approach, grouping related programming tasks logically within the program. Indentation and spacing should be used to ensure that code can be understood and modified easily by a programmer unfamiliar with the project or program. Programs should look neat and orderly, and the visual appearance of a program listing should mirror its logical structure. The following below pointers is recommended to be followed –

- Each line of source code contains only one executable statement for readability (i.e., new line after each semi-colon).
- Place at least one blank line between each DATA step and PROC step.
- Consistently use spaces for indentation and ensure that all program code lines on the same ‘level’ are indented the same distance. Make generous use of spaces to increase readability.
- Indent all statements in a logical grouping by the same amount. There should be same number of spaces or tabs to be used across the program for and within statements. The sub-processing should be clearly visible within main processing.
- Indent all the statements within a DATA or PROC step.

- Align each END statement with its corresponding DO statement. Indent conditional blocks and DO groups and do it consistently. The logic will be easier to follow and verify.
- The program must be in a single case (preferably lower case) except for the scenarios where any value is being compared (using any case function) or assigning value to a variable explicitly. The program should not be in mixed/irregular cases.

  Figure 2 below shows few illustrations of above points.

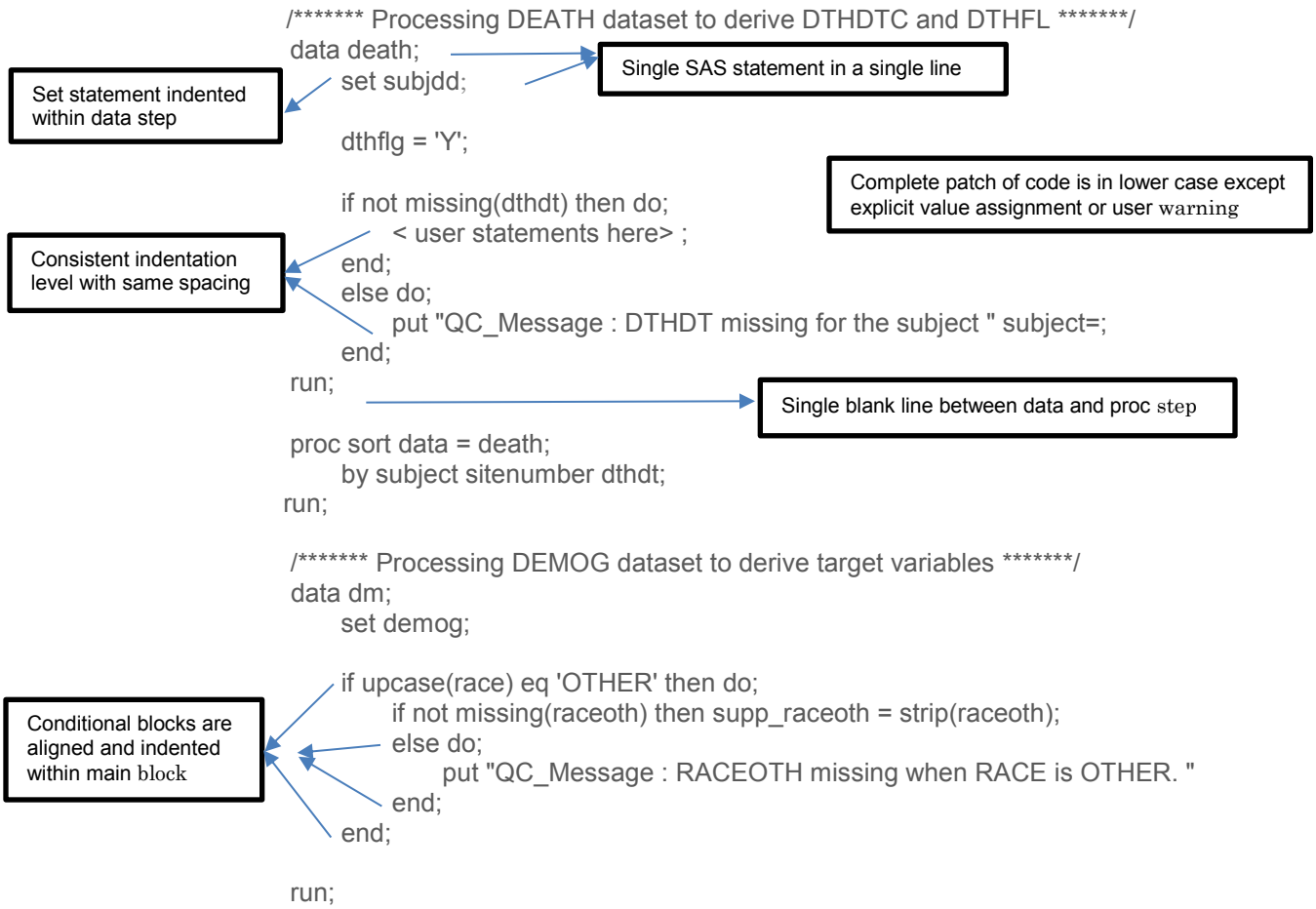


Figure 2 – Illustration for Enhanced Readability and Understandability

EFFICIENT AND EFFECTIVE DATA MANIPULATION

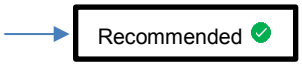
Programming efficiently is more than simply using specific SAS statements. It is understanding that you can create efficiencies through programmatic changes, thoughtful code placement, understanding the data analysis, re-using previously developed code, making use of macros, and recognizing how all this comes together. One method of improving the human efficiency of the programs that we write is to use a linearized, modular style of programming. Programs written in this manner generally require less programmer time for debugging, updating, or modifying. Following the below pointers can enhance your program efficiency and effectiveness significantly:

- It is recommended that the DATA= option be used whenever available, such as, on all procedures even when referencing the most recently created data set to avoid referencing incorrect SAS data

sets.

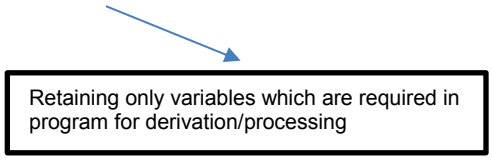
```
data ae_all ;
      set ae sae dlt ;
run ;

proc sort ;
      by subjid aedt ;
run ;
```


```
proc sort data = ae_all ;
      by subjid aedt ;
run ;
```



- Keep only the variables required in your program. Streamlining of the variables is a huge factor for helping quickly understand your program code. The KEEP = or DROP = option (whichever is more appropriate) is used with SET, MERGE, and DATA statements to retain only the variables needed. This helps minimize processing time, particularly when working with large data sets.

```
proc sort data = ic out = ic_sf (keep = subject sitenumber scrnfail) ;
      by subject sitenumber ;
run ;
```




```
data dm_tmp ;
      merge dm(in=a)   ic_sf ;
      by subject sitenumber ;
      if a ;
run ;
```

- Declare the length of all character variables created within the data Step using the LENGTH or ATTRIB statement. This occurs at the beginning of the SAS data Step. This avoids data truncation and irrelevant length related warnings in SAS log.
- The conversion of variables from numeric to character or vice-versa is always performed with the INPUT or PUT function (i.e., do not allow automatic conversions by SAS) to avoid conversion notes in SAS log.

  Note – EXDOSE is a numeric variable and EXDOSTXT is a character variable.

```
exdose = exdostxt ;
```


```
exdose = input(exdostxt, best.) ;
```

- Avoid use of single and double question mark (“?” or “??”) in the format parameter to avoid forced conversion of values and suppress error messages and the setting of the automatic variable _ERROR_. When appropriate, write a message to the log to help facilitate the data cleaning process.

```
exdose = input(exdostxt, ??best.) ;
```


```
exdose = input(exdostxt, best.) ;
```

- It is good coding practice to terminate DATA Steps and PROC statements with an appropriate RUN and/or QUIT statement.

```
proc sort data = demog ;
      by subjid ;
```

→ No RUN statement, hence, Not Recommended

```
proc sort data = demog ;
      by subjid ;
run ;
```

→ Recommended

- Minimize the number of times you read your data. Ensure that the maximum data processing and data manipulation is done in a single data step as much as possible. Do not increase processing time and length of the program by performing repetitive processing.

```
data one ;
      set alldata ;
      where <conditional processing for filtering one dataset> ;
run ;

data two ;
      set alldata ;
      where <conditional processing for filtering two dataset> ;
run ;

data three ;
      set alldata ;
      where <conditional processing for filtering three dataset> ;
run ;
```

→ Not Recommended as the same data set is being read again and again with different data filtering conditions

```
data one two three ;
      set alldata ;

      if <condition> then output one ;
      else if <condition> then output two ;
      else if <condition> then output three ;
run ;
```

→ Recommended as the source data set is read only once

- Subset your data as early as possible. One of the biggest time savers when reading in data comes from sub-setting the data early. Subset as soon as you have all necessary values to prevent unnecessary creation of variables and additional processing.
- When sub-setting a data set, it is recommended that the WHERE or sub-setting IF statement be positioned as close as possible to the SET or MERGE statement.

  Note - If you are sub-setting SAS data set in a DATA or PROC step, use WHERE instead of sub-setting IF. WHERE is a pre-processor. It subsets data before it is loaded into the Program Data Vector (PDV).
- The database can capture the normative expected data response, but a typo in data entry or different capitalization in the data can cause untold issues in our program codes if we do not account for the unexpected. When comparing values for a conditional processing, it is recommended to use UPCASE, LOWCASE or PROPCASE functions to handle mixed cases adequately. This will avoid

future program updates due to mixed case values in data.

```
if upcase(race) eq 'OTHER' then supp_raceoth = strip(raceoth) ;
```

This will ensure that all possible data values 'OTHER', 'Other', 'other' gets handled without any update in program in future.

- Work and permanent SAS data sets, macros, macro parameters, and variables are given meaningful and unique names preferably relating to the data type in question. Moreover, user supplied names (data sets, variables, macros) shall not conflict with SAS keywords, functions, or any other SAS-supplied names. Create these names in a professional manner as SAS programs may be a part of deliverable to the Client.
- Use SAS functions and statements wisely to decrease processing time. For example, instead of IF-THEN statements, use IF-THEN-ELSE statements where applicable. This will allow the control to move out of the conditional processing when one of the conditions is met and proceed with next executable statement of the program instead of checking all the conditions available in continuous IF-THEN statements. Also, try to order the conditions in decreasing order of probability of occurrence.

```
data one two three ;
      set alldata ;

      if <condition> then output one ;
      if <condition> then output two ;
      if <condition> then output three ;
run ;
```

Not Recommended as the control will check all the successive conditions even though the first condition is met true

```
data one two three ;
      set alldata ;

      if <condition> then output one ;
      else if <condition> then output two ;
      else if <condition> then output three ;
run ;
```

Recommended as the control will move to the RUN statement as soon as it executes the condition which is met true

- Design programs defensively, to anticipate possible data findings and avoid SAS warnings or unexpected results. Always program to handle missing and unexpected values.

  For example, if values in the data are going to lead to SAS WARNINGs caused by a zero or missing value for the denominator of a division operation, then it is best to avoid the SAS WARNINGs and possible invalid results by writing customized code using appropriate conditions to yield the expected results.

```
percnt  = (count1/count2)*100 ;
```

This may yield unexpected results and SAS warnings in case of missing or 0 value in COUNT2 variable

```
if nmiss(count1, count2) eq 0 and count2 ne 0 then percnt = (count1/count2)*100 ;
```

Additional precautionary conditions will ensure correct output and avoid unwanted SAS Warnings, notes, and errors.

- For efficient use of computer/network disk storage during program execution, it is recommended that the procedure PROC DATASETS be used to delete temporary SAS files that are no longer needed or if disk space is a constraint.
- It is recognized that using SQL could yield significant efficiencies when sub-setting large data sets or performing complex joins on multiple data sets. Use of SQL could be considered when it provides a significant efficiency over DATA Step processing. However, when using SQL always include comments describing its purpose to aid future code development.
- To assist others with the review of large or complex macros, it is valuable to write to the log the values of the macro parameters upon macro invocation.

INTERNAL PROGRAM DOCUMENTATION

Comments and documentation serve to assist in the understanding of what's in data sets and variables, provide descriptive information about the intricacies and flow of a program, save time during support or maintenance of a program and document program run instructions. They also provide important information about the purpose of the small patches in a program; what the input, processing and output is; and special operations that need additional understanding. Comments makes the code easier to follow and eases the programmer to review the logic of the code. Few suggestions for adding documentation and comments in your program are provided below:

- SAS programs must be properly and consistently commented. Adequate documentation will appear throughout the program to give the reader a good overview of the function and structure of all areas of the program. Since programs have to be maintained and revised over time often by different programmers, clear comments will help a new programmer quickly understand a program.
- It is good practice to comment on each main step, new idea, or algorithm within the program. More comments may be needed within a DATA or PROC step to further clarify complicated programming statements or logic.

```
/*** Processing Country_Info excel to populate COUNTRY and INVNAM ***/
proc import datafile = "c:\users\navneet\country_info"  out = country dbms = xlsx replace ;
run ;

/*** Processing Dummy Randomization file to populate Treatment values ***/
proc import datafile = "c:\users\navneet\rand_info"  out=ext_rand dbms=xlsx replace ;
run ;

/*** Processing IC dataset for Screen Failure flag ***/
proc sort data = raw.ic out = ic_sf (keep = subject sitenumber scrnfail) ;
      by subject sitenumber ;
run ;
```

Comments to address what each PROC step is intended to

- A high-level comment description may be added for a lengthy or complex derivation at the start of derivation and then subsequent short comments in between derivation logic and DATA or PROC steps (as mentioned above) to provide sequential flow of algorithm.

```
/*************************************************************************************/
/*** Deriving Death Date of Subject DTHDTC. Source datasets DEATH,       ***/
/*** AE and EOS will be processed separately and earliest of the death   ***/
/*** date of subject from the three datasets will be considered as DTHDTC  ***/
/*************************************************************************************/
```

- The following points are noteworthy with respect to the comments that could be added for user reference –
    - Show logical process flow
    - Explain derivations
    - Make a note of a specific client request and note reference emails (for filing)
    - Explain data checks on unclean data
    - Document temporary code until data is fixed with a note for removal
    - Modification notes to link back to program header

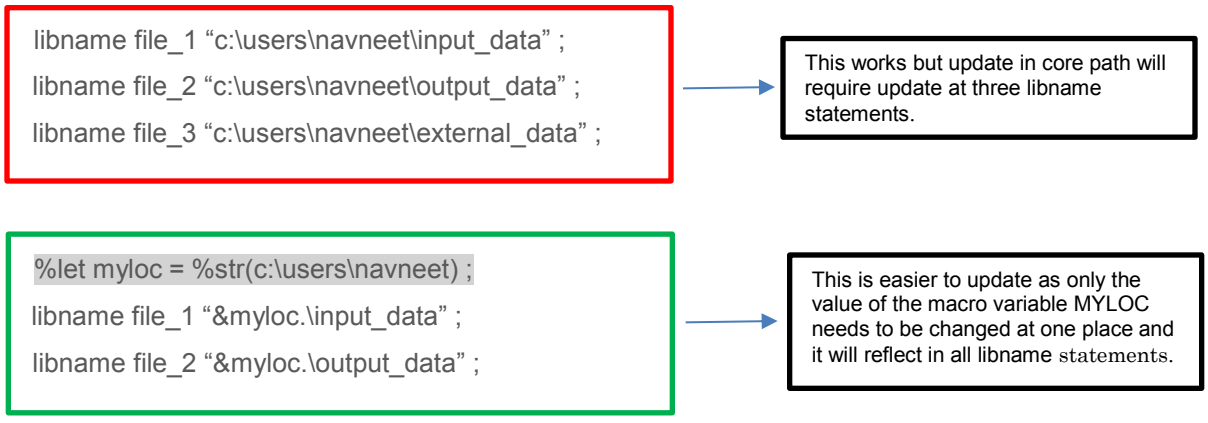
REUSABILITY AND EASY MAINTAINANCE

There are certain tasks and activities that is/or may repeatedly require in all programs to serve a common specific purpose. In such cases, instead of using the same code redundantly in your programs, it is better to develop a macro for the same and invoke it as a single line statement in your programs. Develop a code that can be re-used in different programs with just change in parameter values. You can follow below recommendations to make your program reusable and easily maintainable:

- It is preferred that macro parameters in macros be passed as keyword parameters, rather than as positional parameters. This provides the user a clear understanding of what variables or values is to be passed to each macro parameter for correct and expected output.

  Note – Only use macros when needed, do not overly complicate your code. No one needs a macro for 3 lines of stand-alone code.

- Use macro variables to simplify maintenance and to reduce the number of changes that must be applied manually when code needs to be updated.



QUALITY CHECKS

Quality check is one of the key component, may it be programming, data or output. Quality checks allows the programmer to ensure that the generated output is correct without any ambiguity. The following quality checks could be followed by the programmer to ensure reliability of his/her program and data integrity of output:

- Review the program log for the appearance of the following words: ERROR, WARNING, uninitialized, not resolved, lost cards, invalid, W.D., INFO, or repeats (due to MERGE statements that have more than one data set with repeats of BY values) and other messages that indicate incorrect or error-

prone steps. If any of these appear, review for relevance, or rectify if required.

- Remove all “temporary” code before finalizing a program. This includes code being tested, but not yet fully implemented, code created just for troubleshooting, etc. Programs deemed final and ready to be delivered to client or archived, should not contain obsolete code or comments.
- After transformation of non-SAS native data sets into SAS data sets (e.g., Excel spreadsheets), use any appropriate means necessary to verify that the import and transformation was correctly applied.
- When merging data from different sources it is possible to have unexpected results. Thus, after such a merge, verify that the observations from each data source match up and those unmatched observations are outputted in a separate data set for cross check.
- If observations are to be deleted from a data set, it is recommended that the deleted observations are sent to a separate data set and verified that the correct observations have been deleted, or another adequate means be used to verify that the observations that remain are the correct observations. Also include a comment describing why these data are being deleted.
- Avoid repeated use of data set names throughout the program to aid clarity. Also, do not overwrite the existing temporary data set to ease in debugging. Provide a new data set name in a data step or use an OUT = parameter in SAS procedures.

```
data dm ;
      set dm ;

     < user statements>  ;
run ;

data dm_1 ;
      set dm ;

      < user statements>  ;
run ;
```

Overwriting the data set, hence, not recommended

Creating a new data set, hence recommended

```
proc transpose data = dm_1 ;
      by subjid sitenumber ;
      var height weight ;
run ;

proc transpose data = dm_1 out = dm_2 ;
      by subjid sitenumber ;
      var height weight ;
run ;
```

Overwriting the data set, hence, not recommended

Creating a new data set, hence recommended

- Include user messages in the program corresponding to unexpected results so that user messages written to the SAS log could be used to identify the programming bug or a data issue easily.

```
if upcase(race) eq 'OTHER' then do ;
      if not missing(raceoth) then supp_raceoth = strip(raceoth) ;
      else do ;
           put "QC_Message : RACEOTH missing when RACE is OTHER. "
      end ;
end ;
```

Providing a QC message in case RACEOTH is not present when RACE = OTHER. This will provide the programmer heads-up for the data issue and affected output.

POINTS TO AVOID

- Excessive documentation so the program loses the flow and becomes unreadable.
- Use of many data steps when a lot of updates can be done on one read of the data.
- Invoking nested macros (macro invoked within a macro) unless explicitly required.
- Reference to external files dotted throughout the code.
- Sorting data sets when it is already sorted, or no sorting is required for processing.
- Deleting temporary data sets in between of the program or replacing existing data sets.

CONCLUSION

SAS is a very flexible language that allows the programmer a great deal of leeway when writing a program. However, an important aspect of good programming practice is to have a well-planned approach to writing a program. Its time-consuming and painful to comb through programs line by line, deciphering un-commented code with inexplicable variable names, and disjointed logic flow. The source code should flow naturally from the steps that access the data, then modifications, analysis, and finally printing results. While developing any program, bear in mind that the steps you take today in your program code, can help your entire team in the future. It will "future-proof" your work, ensuring it can be reused and understood in the future as well as improve their overall efficiency.

REFERENCES

Shafi Chowdhury, Mark Foxwell and Cindy Song – "Essential Guide to Good Programming Practice", PharmaSUG 2015

Thomas J. Winn Jr. – "Guidelines for Coding of SAS® Programs"

ACKNOWLEDGMENTS

The authors would like to acknowledge the "Material Review Board" group of Syneos Health for their thoughtful review of this manuscript.

CONTACT INFORMATION

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

Navneet Agnihotri
Syneos Health, Principal Statistical Programmer
navneet.agnihotri@syneoshealth.com
https://www.linkedin.com/in/navneet-agnihotri-78540553/

Sumit Pratap Pradhan
Syneos Health, Principal Statistical Programmer
sumit.pradhan@syneoshealth.com
https://www.linkedin.com/in/sumit-pradhan-71133345/

Rachel Brown
Syneos Health, Manager (Statistical Programmming)
rachel.brown@syneoshealth.com