PharmaSUG 2025 Paper AP-059

Validate the Code, not just the Data : A System for SAS® program validation

Jay Iyengar, Data Systems Consultants LLC

ABSTRACT

Regardless of the industry they work in, SAS® programmers are focused on validating data, and devote a considerable amount of attention to the quality of data, whether its raw source data, submitted SAS data sets, or SAS output, including figures and listings. No less important is the validity of code and the SAS programs which extract, manipulate, and analyze data. Although code validity can be assessed through the SAS log, there other ways to produce metrics on code validity. This paper introduces a system for SAS program validation which produces useful information on lines of code, number of data steps, total run and CPU time and other metrics for project-related SAS programs.

INTRODUCTION

Programmers are tasked with manipulating and analyzing SAS data sets which contain real-world data. The SAS data sets might contain data records on hospital patients, patient visits to a clinic for patients enrolled in a clinical study, company employees, or company customers, for example. The purpose of storing this data is to summarize and report it to generate insights which assist clinical or organizational decision-making. For this task, the programmer has to give much attention to the quality and validity of the data. They develop SAS code to perform various tasks and use SAS procedures to assess the validity of SAS data sets, such as PROC CONTENTS, PROC PRINT, PROC FREQ, PROC MEANS and PROC COMPARE. However, not as much attention is given to evaluating the quality and validity of the code developed to perform these tasks. What tools are available to determine the quality and validity of code? To this issue we now turn our attention.

CODE ANALYSIS

How does one go about analyzing SAS code? When developers review their code, they can evaluate it from many standpoints, such as efficiency, format, readability, or documentation adequacy. They can also test-run SAS code, to discover its overall run-time. Then, by browsing the SAS log, they can collect statistics from it on CPU time, Real time and other performance metrics. Then this compiled information can be used to evaluate performance.

However, it would be helpful if there was a more systematic way to analyze and assess code. This is possible through converting your SAS programs to SAS data sets, so code becomes data. Then you can manipulate and analyze the lines of a SAS program, just as you would a record in a SAS data set.

MEASURES OF PROGRAM PERFORMANCE

The objective of this project is to produce metrics for SAS code which will be used to validate and evaluate it. So, what metrics would be valuable to evaluate code?

What measures would be useful to compute which tell us how robust the program is, how efficient it is, how easy it is to maintain and update, and how well documented the code is.

NUMBER OF LINES OF CODE

Discovering the total number of lines in the program could potentially be useful. The greater the number of lines of code, the longer the run time of the code. That being said, there are many lines in your program which don't contain actual code.

Programs usually contain blank lines to separate steps in the code, and to separate SAS statements within a step for the purposes of enhancing readability. It's of greater value to know the number of lines of code in the program. So, number of lines of code is definitely one of the metrics to compute.

The number of lines of SAS code will tell you how long the SAS program is. Lengthy SAS programs are harder to maintain, because there's more code that needs to be updated. In general, lengthy SAS programs are also more time consuming to read and understand by third partes who which aren't familiar with it.

In this project, if we create a SAS data set based off our code, the SAS data set will contain a single record for each line in the program. Thus, the number of program lines in our source program is equivalent to the number of observations of the SAS data set.

NUMBER OF SAS STATEMENTS

SAS statements refer to code elements which are step-specific or global statements, many of which start with a keyword. The number of SAS statements is not equivalent to the number of lines in your SAS program. SAS statements don't include comment statements, and of course blank lines which are used to add whitespace and separate steps in your code. Statements always end with a semi-colon.

For SAS statements, another issue to consider is that each statement cannot always be counted equal to one line of code. A single SAS statement may wrap around multiple lines in your SAS program. Conversely, multiple SAS statements may appear on a single line in the program. It all depends on the preferences and coding style of the developer.

NUMBER OF STEPS

The number of steps in a SAS program consists of the number of DATA STEPS and PROC steps or SAS procedures. This might be a better measure of a program's efficiency than lines of code, since some lines of code are global statements which don't process data.

Most DATA STEPS and PROC steps process data. A DATA STEP with a non-executing SET statement is an example of a DATA STEP that doesn't process data. A DATA _NULL_ step processes half the amount of data, since it doesn't create an output SAS data set.

Examples of SAS procedures which don't process data include utility procedures such as PROC CONTENTS and PROC DATASETS. Both procedures output the descriptor portion of a SAS data set, which contains variable attributes and overall data set information.

NUMBER OF DATA STEPS

The number of DATA STEPSs in your SAS program can be a useful measure to evaluate the code's performance and processing efficiency. The DATA STEP can be a resource-intensive construct, and consume a lot of computing resources, such as CPU, Input/Output and memory.

In general, DATA STEPs read and write substantial amounts of data, depending on the size of the SAS data set. DATA STEPs usually produce temporary or permanent SAS data sets, which consume available storage space, another computing resource.

All else being equal, the larger the number of DATA STEPs, the longer the program takes to execute and the less efficient the code is. As a programmer, if you can minimize the number of DATA STEPs, then you'll minimize the number of passes through the data.

NUMBER OF SPECIFIC CONSTRUCTS AND PROCEDURES

There are specific BASE SAS constructs which consume more computing resources than others. We've mentioned the DATA STEP as being one.

PROC SORT, for instance, is a resource-intensive construct, which can use a lot of CPU, Input/Output, Memory, and Storage Space. The greater the number of sorts, the longer real-time the program takes to run. To evaluate a program on efficiency, tracking the number of PROC SORTs can be useful.

The DATA STEP Merge is another resource-intensive construct. In some situations, the DATA STEP Merge could be replaced with an alternate table-lookups technique, such as the PROC SQL join. Monitoring the number of DATA STEP Merges has its advantages.

READING IN A SAS PROGRAM AS AN EXTERNAL FILE

In order to convert your SAS code to SAS data sets, you need to save your code as an external file, such as a text file or CSV file. From this point, you can read the text file into SAS, using the DATA STEP and the traditional INFILE and INPUT statements.

An alternative method is to use PROC IMPORT to read the external file. If you save the code as a space-delimited file, or a tab-delimited file then you can specify the respective value in the DBMS= option.

Yet another way to go about this is to save your code as a text file, import your code into EXCEL and save it as an XLSX file. Then you can use PROC IMPORT to import the XLSX file as it would any EXCEL spreadsheet. This method is illustrated by the code sample inserted here.

```
Filename REFFILE '/Home/lyenj/SAS Papers/Code_Validation/Code_Sample1.xlsx';

Proc Import Datafile=REFFILE
    Dbms=XLSX
    Out=Work.Code_Sample;
    Datarow=1;
    Getnames=No;
Run;

Proc Contents Data=Work.Code_Sample;
Run;

Proc Print Data=Work.Code_Sample (Obs=25);
Run;
```

The above code uses a FILENAME statement to create a FILEREF pointing to the EXCEL workbook and then uses PROC IMPORT to read the EXCEL file into SAS and convert it to a SAS data set.

Notice that PROC IMPORT uses the DATAROW=1 option to start reading the file at row 1. By Default, PROC IMPORT starts reading external files at row 2, because it designates the first row for variable names and headers. In Figure 1 is a view of the SAS data set created by the import process.

Obs	Line
1	options symbolgen;
2	
3	Libname SRC 'C:\Users\scoot\Documents\SAS Files\Admin_Healthcare_Data_Medicare\Course Datafiles\Datasets';
4	Libname FINAL 'C:\Users\scoot\Documents\SAS Files\Admin_Healthcare_Data_Medicare\Course Datafiles\Datasets';
5	
6	ODS HTML;
7	
8	/* STEP 5.1: SORT CARRIER LINE FILE IN PREPARATION FOR TRANSFORMATION */
9	proc sort data=src.carrier2010line out=carrier2010line;
10	by bene_id clm_id clm_ln;
11	run;
12	
13	proc print data=carrier2010line(obs=3);
14	where bene_id='00016F745862898F';
15	var bene_id clm_id clm_ln line_icd_dgns_cd;
16	run;
17	
18	/* STEP 5.2: TRANSFORM CARRIER LINE FILE */
19	data carrier2010line_wide(drop=i expnsdt1 expnsdt2 hcpcs_cd line_icd_dgns_cd clm_ln linepmt prfnpi tax_num);
20	format expnsdt1_1-expnsdt1_13 mmddyy10.
21	expnsdt2_1-expnsdt2_13 mmddyy10.
22	line_icd_dgns_cd1- line_icd_dgns_cd13 $5.
23	hcpcs_cd1-hcpcs_cd13 $7.
24	linepmt1-linepmt13 10.2
25	prfnpi1-prfnpi13 $12.

Figure 1. SAS data set of SAS program code.

For the SAS data set in Figure 1, only the first 25 records are displayed. This is because the OBS= option was used when printing to the listing destination using PROC PRINT. You'll notice from the SAS data set that it contains one record per line of the SAS program, regardless of whether that line contains actual code. Blank lines which separate steps in the code have their own record.

To extract the program name and the directory path where it's stored, SASHELP views were used.

SASHELP views contain the same metadata as dictionary tables, except they can be referenced in the DATA STEP instead of PROC SQL. The view, SASHELP.VEXTFL includes the XPath variable which contains the path and filename.

The DATA _NULL_ step to extract the path and filename is provided in Figure 2.

```
                    /* Define Macrovariable for File Directory */

Data _Null_;
    Set Sashelp.VEXTFL (Where=(FILEREF='REFFILE'));

                    /* Extract Filename from Xpath variable */
    File_Name = Scan(Xpath, -1, '/');
    Put File_Name= ;

                    /* Extract Length Of Filename and Path */
    File_Name_L = Length(File_Name);
    Path_L = Length(XPath)-File_Name_L;
    Put File_Name_L= Path_L= ;

                    /* Extract Directory Path */
    DirPath = Substr(XPath, 1, Path_L);
    Put DirPath= ;

                    /* Create Macro Variables For FileName and Directory Path */
    Call Sympust ('DirPath', DirPath);
    Call Sympust ('FileN', File_Name);

Run;

%Put &DirPath= &FileN= ;
```

Figure 2. Extracting Path and Filename from SASHELP View.

The path and filename are split into separate variables and macro variables are created to store those values, which are referenced later on in the code.

COMPUTING CODE PERFORMANCE METRICS

The next step is to query the records in our SAS data set for specific SAS constructs and code elements and flag those records for the different performance metrics defined earlier in the paper. For this task, DATA STEP programming proved very valuable.

In Figure 3 is an excerpt of SAS code using the DATA STEP to create a series of flag variables for the various performance measures discussed earlier. The DATA STEP searches the SAS program for specific keywords and elements in the code.

```
Data Code_Sample_v2;
        Set Code_Sample (Rename=(A=Line)) End=FINAL;

                        /* Compute Length of Code Line*/
        Line_Length = Length(Line);
        If _N_=1 Then Put Line_Length= ;

                        /* Count all Lines in Program, whether or not they contain code */
        LineCount+1;

                        /* Create Flag Indicator Variables for specific SAS constructs */
        If Line^=' ' Then Do;

                If Index(Line, ';')>0 Then SAS_Stmnt=1;
                Else SAS_Stmnt=0; *SAS Statements;

                If Index(Propcase(Line), 'Data ')>0 Then Data_Step=1;
                Else Data_Step=0; *Data Steps;

                If Index(Propcase(Line), 'Merge ')>0 Then DS_Merge=1;
                Else DS_Merge=0; *Data Step Merge;

                If Index(Propcase(Line), 'Proc ')>0 Then Proc_Step=1;
                Else Proc_Step=0;     *SAS Procedures;

                If (Substr(Line, 1, 2)='/*' or Substr(Line, Line_Length-1, 2)='*/') and
                  Index(Propcase(Line), ';')=0 Then Comment=1;
                Else Comment=0;      *Comments;

                                /* Records containing actual code */
                Code_Line=1;
        End;

        Else Do;
                SAS_Stmnt=0;
                Data_Step=0;
                DS_Merge=0;
                Proc_Step=0;
                Comment=0;
                Code_Line=0;
        End;

        If Final=1 Then Call SymputX('TotCount', LineCount);
Run;
```

Figure 3. DATA STEP creating Code Construct Flag variables.

The code computes the length of each line in the program, and counts the total number of lines, without respect to whether they contain actual code.

For records where the line of the program was not blank, the code uses conditional logic and several SAS functions to locate and flag specific terms.

Specifically, the code uses the INDEX function, and the SUBSTR function to search for character strings inside variables on the SAS data set. Some of the search terms are SAS keywords (Data, Proc, Merge), or special characters (/*, ;) which indicate specific constructs or code elements.

The code creates flag variables as binary variables with 0 / 1 values, indicating the absence or presence of the code element respectively.

Listed in Figure 4 below is a PROC PRINT output of the SAS data set containing the flag variables for SAS construct performance metrics.

Obs	Line	LineCount	Code_Line	SAS_Stmnt	Data_Step	DS_Merge	Proc_Step	Comment
1	options symbolgen;	1	1	1	0	0	0	0
2		2	0	0	0	0	0	0
3	Libname SRC 'C:\Users\scoot\Documents\SAS Files\Admin_Healthcare_Data_Medicare\Course Datafiles\Datasets';	3	1	1	0	0	0	0
4	Libname FINAL 'C:\Users\scoot\Documents\SAS Files\Admin_Healthcare_Data_Medicare\Course Datafiles\Datasets';	4	1	1	0	0	0	0
5		5	0	0	0	0	0	0
6	ODS HTML;	6	1	1	0	0	0	0
7		7	0	0	0	0	0	0
8	/* STEP 5.1: SORT CARRIER LINE FILE IN PREPARATION FOR TRANSFORMATION */	8	1	0	0	0	0	1
9	proc sort data=src.carrier2010line out=carrier2010line;	9	1	1	0	0	1	0
10	by bene_id clm_id clm_ln;	10	1	1	0	0	0	0
11	run;	11	1	1	0	0	0	0
12		12	0	0	0	0	0	0
13	proc print data=carrier2010line(obs=3);	13	1	1	0	0	1	0
14	where bene_id='00016F745862898F';	14	1	1	0	0	0	0
15	var bene_id clm_id clm_ln line_icd_dgns_cd;	15	1	1	0	0	0	0
16	run;	16	1	1	0	0	0	0
17		17	0	0	0	0	0	0
18	/* STEP 5.2: TRANSFORM CARRIER LINE FILE */	18	1	0	0	0	0	1
19	data carrier2010line_wide(drop=i expnsdt1 expnsdt2 hcpcs_cd line_icd_dgns_cd clm_ln linepmt prfnpi tax_num);	19	1	1	1	0	0	0
20	format expnsdt1_1-expnsdt1_13 mmddyy10.	20	1	0	0	0	0	0
21	expnsdt2_1-expnsdt2_13 mmddyy10.	21	1	0	0	0	0	0
22	line_icd_dgns_cd1- line_icd_dgns_cd13 $5.	22	1	0	0	0	0	0
23	hcpcs_cd1-hcpcs_cd13 $7.	23	1	0	0	0	0	0

Figure 4. Intermediate SAS data set with Flag Variables

In the SAS data set, you will notice that the flag variables refer to specific SAS constructs, such as DATA STEP, DATA STEP MERGE, PROC step, etc. which are defined here as program performance measures and correspond to the lines in the program.

SUMMARIZING THE PERFORMANCE MEASURES

The last step in this process of analyzing code is to summarize the flag indicator variables for performance metrics, and generate a report with overall descriptive statistics.

Although many other constructs could've been used to perform this step, a PROC SQL summary query using aggregate functions was executed to generate the output table.

The PROC SQL query is presented in Figure 5 below.

```
Proc Sql;
        Create Table Code_Summary as
        Select "&DirPath" as Directory_Path,
                "&FileN" as Program_Name,
                "&TotCount" as Num_Lines,
                Sum(Code_Line) as Num_Lines_Code,
                Sum(SAS_Stmnt) as Num_Statements,
                Sum(Data_Step) as Num_DataSteps,
                Sum(DS_Merge) as Num_Merge,
                Sum(Proc_Step) as Num_ProcSteps,
                Sum(Comment) as Num_Comments

        From Code_Sample_v2;
Run;
```

Figure 5. PROC SQL Query to produce Summary Metrics

As the code in Figure 5 illustrates, the PROC SQL query uses the SUM function to compute the totals for each performance metric. The macro variables containing the path; DirPath, and filename; FileN are referenced here to store that information in variables. The number of lines in the program isn't summed but is also pulled from the TotCount macro variable, defined in a previous step.

The code in Figure 5 produces a summary SAS data set with totals for each of the performance metrics. The data set contains only one observation, since we're only generating these numbers for one specific SAS program. The table in Figure 6 contains the PROC PRINT output of the summary SAS data set created in the code in Figure 5.

Obs	Directory_Path	Program_Name	Num_Lines	Num_Lines_Code	Num_Statements	Num_DataSteps	Num_Merge	Num_ProcSteps	Num_Comments
1	/home/iyenj/SAS Papers/Code_Validation/	Code_Sample1.xlsx	413	331	285	14	16	34	25

Figure 6. Output Table with Program Performance Metrics

The interpretation of the summary numbers for the performance metrics is as follows.

The program contains 413 total lines

The program contains 331 lines of actual code.

The program contains 285 SAS statements.

The program contains 25 comments lines.

There are 14 DATA STEPs in the program and 34 PROC Steps.

There are 16 DATA STEP Merges in the program.

The total number of steps in the program is equal to the number of DATA steps + number of PROC steps= 34+14 = 48.

HARNESSING PERFORMANCE INFORMATION IN THE SAS LOG

Similar to a SAS program, information in the SAS log can be extracted and manipulated and then analyzed to produce useful insights about a program's performance. The SAS log contains resource use data in response to the execution of SAS code.

For each step in the program, SAS provides statistics for real time, cpu time, and other statistics. When you specify the FULLSTIMER system option, SAS generates additional performance statistics in the log, including user cpu time, system cpu time, memory and OS memory. Below is a listing and definition of each metric with a brief discussion.

REAL TIME

Real time is processing time. It's the amount of elapsed time which a particular step took to process. Real time can also be thought of as the amount of time a SAS job took to process.

CPU TIME

CPU time is the full amount of time your code took to execute, including the amount of time SAS used to perform system overhead tasks. CPU time is the sum of user cpu time, and system cpu time. The amount of cpu time may be less than, equal to or greater than real time, depending on the step which executed, and the actions SAS performed in that step.

USER CPU TIME

User CPU is the portion of CPU time which your code took to execute. The DATA STEP and procedures which perform a lot of manipulations on SAS data sets will have higher user cpu time and higher cpu time in general. Procedures such as PROC PRINT, and utility procedures such as PROC CONTENTS usually have low amounts of user CPU time. Utility procedures only process metadata, the descriptor portion of the SAS data set.

SYSTEM CPU TIME

System CPU time is the amount of time your CPU spends doing system overhead tasks which support the execution of your SAS program. An example of a system overhead task is file management such as assigning a file to a folder or subdirectory or assigning a specific name to a file in that subdirectory.

MEMORY

The amount of memory that SAS uses to execute a step of SAS code is also provided in the SAS log. Although not by default, memory is displayed when the FULLSTIMER system option is used. The memory displayed in the log is RAM or random-access memory. Memory is measured in bytes, such as kilobytes (KB), megabytes (MB) or gigabytes (GB). Though an exception, the amount of memory could also potentially be in Terabytes (TB) or Petabytes (PB). In the log, only a single letter appears next to the amount of memory to indicate the units its in; k for KB, m for MB, etc.

OS MEMORY

OS Memory is the amount of memory which has been requested from the system. Rather than the amount of memory, which is actually being used to execute the code, it's the amount of memory which is available if needed. It's also measured in bytes (KB, MB, GB) similar to the memory statistic.

CONVERTING A SAS LOG TO A SAS DATA SET

The first step in the process of extracting performance metrics from the SAS log is to save the log as a text file and read it into SAS using one of several methods. You can read the external file using PROC IMPORT as we did for the SAS program. Or you can read it using the DATA STEP and the INFILE and INPUT statements. For this task, we chose to use the DATA STEP and the INFILE and INPUT statements. The code to perform this task is provided here.

```
Filename SASLOG '/home/iyenj/SAS Papers/Code_Validation/Code_Sample1_Log.txt';

                    /* Create SAS data set of SAS log */
Data CVFILES.CS1LOG;
      Length LINEHDR $ 5 MESSAGE $ 250 CODE $ 245;

                          /* Read in SAS Log as an external file */
      Infile SASLOG DLM=':' LRECL=275 TRUNCOVER;
      Input @1 LINEHDR $5. @;
            If (LINEHDR='NOTE' or LINEHDR=' ') Then
                  Input @7 MESSAGE $250.;
                 Else
                  Input @12 CODE $245.;
Run;
```

It was necessary to initially read in the first variable, Linehdr, and use the Single-trailing @ line-hold specifier as a record place holder. Based on the value in Linehdr, SAS will extract and process one of two items; either the message printed in the log, or the SAS code also printed in the log the message corresponds to.

Once this process is complete, the messages from the log are contained in the Message variable in a new SAS data set. The Table in Figure 7 below displays the first 25 records of the SAS data set containing printed notes from the log.

Obs	LINEHDR	MESSAGE
1		
2	1	
3	72	
4	73	
5	74	
6	75	
7	NOTE	Libref SRC was successfully assigned as follows:
8		Engine: V9
9		Physical Name: /home/iyenj/Training_Course_Files/Admin_hc_data/Data
10	76	
11	NOTE	Libref FINAL was successfully assigned as follows:
12		Engine: V9
13		Physical Name: /home/iyenj/Training_Course_Files/Admin_hc_data/Data/Output
14	77	
15	78	
16	79	
17	NOTE	Data file SRC.CARRIER2010LINE.DATA is in a format that is native to another host, or the file encoding does not match the
18		session encoding. Cross Environment Data Access will be used, which might require additional CPU resources and might reduce
19		performance.
20	80	
21	81	
22		
23	NOTE	There were 2326156 observations read from the data set SRC.CARRIER2010LINE.
24	NOTE	The data set WORK.CARRIER2010LINE has 2326156 observations and 15 variables.
25	NOTE	PROCEDURE SORT used (Total process time):

Figure 7. SAS Data set with Log Messages as data values.

Notice in the data set, the variable Message has values which are specific printed log messages for each step or global statement in the SAS program. Besides notes, If the SAS log contained any errors or warnings they would appear in values of the Message variable, where Linehdr equal to ’ERROR’ or ’WARNING’.

EXTRACTING SPECIFIC MEASURE PERFORMANCE STATISTICS

Now that the text file is a SAS data set, we can proceed with filtering the records in the data set for specific metrics and extracting the pertinent information from the log for a subset of those records. Specific keywords and character strings were used to locate records of the data set whose values include performance metrics.

If the records meet the specified conditions, then the information in the log message was extracted and stored in a set of new variables. Although it sounds straight-forward, this task involved using moderately complex DATA STEP programming, specifically conditional logic, and complex string manipulation. The code to perform this task is presented in Figure 8.

```
                        /* Search and extract Performance Metrics from SAS log */
Data CS1LOG;
     Set CVFILES.CS1LOG;

     Length Metric $20 Amount $15 Units $12;

                        /* Extract Real Time */
     If Substr(Message, 1, 9) = 'real time' Then Do;
                  Metric = 'Real Time';
                  Amount = Scan(Message, 3, ' ');
                  Units = Propcase(Scan(Message, 4, ' '));
     End;

                        /* Extract User CPU Time */
     Else If Substr(Message, 1, 13)= 'user cpu time' Then Do;
                  Metric = 'User CPU Time';
                  Amount = Scan(Message, 4, ' ');
                  Units = Propcase(Scan(Message, 5, ' '));
     End;

                        /* Extract User CPU Time */
     Else If Substr(Message, 1, 15)='system cpu time' Then Do;
                  Metric = 'System CPU Time';
                  Amount = Scan(Message, 4, ' ');
                  Units = Propcase(Scan(Message, 5, ' '));
     End;

                        /* Extract Memory */
     Else If Substr(Message, 1, 6)='memory' Then Do;
                  Metric = 'Memory';
                  Amount = Scan(Message, 2, ' ');
     End;

                         /* Extract OS Memory */
      Else If Substr(Message, 1, 9)='OS Memory' Then Do;
                  Metric = 'OS Memory';
                  Amount = Scan(Message, 3, ' ');
      End;

                         /* Define Units for Memory */
      If Metric In('Memory', 'OS Memory') Then Do;
                  If Substr(Amount, Length(Amount), 1)='k' Then
                        Units = 'KB';
                  Else If Substr(Amount, Length(Amount), 1)='m' Then
                        Units = 'MB';
                  Else If Substr(Amount, Length(Amount), 1)='g' Then
                        Units = 'GB';
      End;

      If LINEHDR In ('NOTE', ' ');
Run;
```

Figure 8. DATA STEP to extract performance metrics from SAS Log.

To manipulate the character strings, the code uses several character string functions; SUBSTR, SCAN, LENGTH and PROPCASE. Three new variables were created to contain the extracted processing data; Metric, Amount and Units.

Metric contains the name of the metric; OS Memory, Real Time, etc. Amount contains the numeric value for that metric. The variable Units displays the metric unit values, 'Seconds' for timing metrics, and 'KB', 'MB', or 'GB' for memory values.

In Figure 9 is a sample of observations from the intermediate SAS data set with the new variables.

Obs	LINEHDR	MESSAGE	Metric	Amount	Units
1					
2	NOTE	Libref SRC was successfully assigned as follows:			
3		Engine: V9			
4		Physical Name: /home/iyenj/Training_Course_Files/Admin_hc_data/Data			
5	NOTE	Libref FINAL was successfully assigned as follows:			
6		Engine: V9			
7		Physical Name: /home/iyenj/Training_Course_Files/Admin_hc_data/Data/Output			
8	NOTE	Data file SRC.CARRIER2010LINE.DATA is in a format that is native to another host, or the file encoding does not match the			
9		session encoding. Cross Environment Data Access will be used, which might require additional CPU resources and might reduce			
10		performance.			
11					
12	NOTE	There were 2326156 observations read from the data set SRC.CARRIER2010LINE.			
13	NOTE	The data set WORK.CARRIER2010LINE has 2326156 observations and 15 variables.			
14	NOTE	PROCEDURE SORT used (Total process time):			
15		real time 3.82 seconds	Real Time	3.82	Seconds
16		user cpu time 3.90 seconds	User CPU Time	3.90	Seconds
17		system cpu time 0.53 seconds	System CPU Time	0.53	Seconds
18		memory 480254.06k	Memory	480254.06k	KB
19		OS Memory 506948.00k	OS Memory	506948.00k	KB
20		Timestamp 03/04/2025 04:50:15 PM			
21		Step Count 24 Switch Count 8			
22		Page Faults 0			
23		Page Reclaims 109654			
24		Page Swaps 0			
25		Voluntary Context Switches 1959			

Figure 9. SAS data set with variables containing performance data variables

From Figure 9, notice that it is only a small subset of records in the SAS data set which contains the valuable information we're after. This is because there are only a small number of lines in the SAS log which contain statistics on performance metrics.

It's only the actual steps in the program which generate performance statistics, that is DATA STEPs and PROC steps, respectively. Other code in the program including global statements, such as LIBNAME, OPTIONS or TITLE statements don't generate performance statistics. No real data processing occurs while those statements are executing, and the actual amount of time to run those statements is minute.

As appearing in the data set, additional statistics are generated in the log which are not being utilized for this project. With the FULLSTIMER option invoked, SAS generates the following metrics; Step Count, Switch Count, Page Faults, Page Reclaims, Page Swaps, and Voluntary Context Switches. A deep dive into these metrics is outside the scope of this paper.

In order to make use out of this information further manipulation and transformation of the data set was required.

By paying close attention to Figure 9, you'll notice that the variable Amount contains non-numeric characters. These characters need to be removed before the numeric data can be properly analyzed. To perform this task, routine DATA STEP programming was conducted. The code which executed this further manipulation is presented in Figure 10.

```
Data CS1LOG_V2;
    Set CS1LOG(Rename=(Amount=AmtOrig));

    Length AmountTmp1 $10;

                /* Extract all characters but the last from Amount */
                /* Convert Amount to Numeric */
    If Substr(AmtOrig, Length(AmtOrig), 1) In ('k', 'm', 'g')
            Then AmountTmp1 = Substr(AmtOrig, 1, Length(AmtOrig)-1);

    Else AmountTmp1 = AmtOrig;

    Amount=Input(Compress(AmountTmp1, , 's'), 10.2);

    Keep Metric Units Amount;

    If Metric^=' ';
Run;
```

Figure 10. SAS code used to extract and convert specific characters

Similar to other parts of the code, this DATA STEP uses conditional logic with IF-THEN-ELSE processing as well as character functions, such as SUBSTR and LENGTH to complete the task. Also, it uses the COMPRESS and INPUT functions to exclude blanks and convert the string to a numeric variable.

Besides character string manipulation, we needed to exclude the records for lines in the log where there isn't any useful performance information. A subsetting IF statement appears at the bottom of the step to exclude records with missing values for Metric.

Obs	Metric	Units	Amount
1	Real Time	Seconds	3.82
2	User CPU Time	Seconds	3.90
3	System CPU Time	Seconds	0.53
4	Memory	KB	480254.06
5	OS Memory	KB	506948.00
6	Real Time	Seconds	0.02
7	User CPU Time	Seconds	0.02
8	System CPU Time	Seconds	0.01
9	Memory	KB	4209.68
10	OS Memory	KB	30380.00
11	Real Time	Seconds	2.79
12	User CPU Time	Seconds	2.33
13	System CPU Time	Seconds	0.43
14	Memory	KB	3804.75
15	OS Memory	KB	32684.00
16	Real Time	Seconds	0.01
17	User CPU Time	Seconds	0.01
18	System CPU Time	Seconds	0.01
19	Memory	KB	2779.71
20	OS Memory	KB	30892.00

Figure 11. Performance Metrics For each Program Step

The code in Figure 10 produces the temporary SAS data set included in Figure 11 above.

From the data set you'll notice that extraneous records have been deleted, and only the three primary variables are present. At this point, the data from the SAS log is tightly organized in a format which is easier to analyze.

SUMMARIZING THE PERFORMANCE MEASURES

There are a number of ways which performance metrics from the SAS log can be analyzed that are useful and provide value. For the purposes of this project, it was of primary importance to calculate program-specific totals for each of the metrics; CPU Time, Memory etc.

After summarizing the data, it was necessary to re-shape the aggregate data to output it in a format which is presentable and professional looking. PROC SQL was used to summarize the data, and PROC TRANSPOSE to restructure the data, respectively. The code to perform this task is displayed in Figure 12.

```
Proc Sql;
 Create Table Log_Stat_Sum as
 Select Metric, Put(TotAmt, Comma10.1)||' '||New_Units as Total_Amount
 From
        (Select Metric,
                Units,
                Case
                  When Units='KB' Then 'MB'
                  Else Units
                End as New_Units,
                Case
                  When Units='KB' Then Sum(Amount)/1000
                  Else Sum(Amount)
                End as TotAmt Format=Comma10.1
         From CS1LOG_V2
         Group By Metric, Units);
Quit;

Proc Transpose Data=Log_Stat_Sum Out=Log_Metric_Final;
       Var Total_Amount;
       ID Metric;
Run;
```

Figure 12. Summarizing and reshaping data with PROC SQL and PROC TRANSPOSE

The PROC SQL step utilizes subqueries in the FROM statement. In the inner query, the data is summarized by Metric and Units, and Units and Amount are re-coded to express memory in Megabytes (MB) for both Memory and OS Memory. The CASE expression logic is used to do the re-coding. The results are then passed to the outer query.

In the outer query, amount is simply re-formatted and concatenated with units for each metric. An output SAS data set is produced by the CREATE TABLE statement.

To restructure the data set, PROC TRANSPOSE was used. Either PROC TRANSPOSE or the DATA STEP can be used to restructure data sets and convert observations into variables or vice versa. For our purposes, a simple transposition was all that was needed to produce a final data set with the summary information on a single record.

In Figure 13 is a PROC PRINT listing of the summarized log performance data.

Obs	Memory	OS Memory	Real Time	System CPU Time	User CPU Time
1	3,506.4 MB	5,157.0 MB	26.7 Seconds	6.2 Seconds	18.8 Seconds

Figure 13. PROC PRINT Output of SAS Log Performance Metrics

Notice the discrepancy between real time and the sum of system CPU time and user CPU time. Real time should be equal to the sum of user CPU time and system CPU time. In the output in Figure 13, the figures don't match exactly. Nevertheless, a processing time of 26.7 seconds would be considered efficient for a SAS program containing 400 lines of code.

The amount of memory is expressed in Megabytes in the variables Memory and OS Memory. These figures indicate that the program consumed substantial amounts of memory. 3506.4 MB of memory and 5157 MB of Operating System memory translates to 3.5 gigabytes (GB) of memory and 5.1 GB of operating system memory.

In SAS, memory is used to store data while it is being processed in the DATA STEP through the Program Data Vector (PDV). It's also used to build temporary subsets of data which are produced from procedures, such as PROC SORT, and later combined to form a sorted SAS data set.

The large amounts of memory utilized in executing the code reflects the amount of data processed by it. The data used by the underlying code are large health claims files holding in excess of 1 million records. The code also has many passes through the data, as it contains 14 DATA STEPs, which was discovered in the first section of the paper.

CONSOLIDATING THE OUTPUT INTO A REPORT

Our aggregate program performance data has been produced in SAS data sets after extracting information from the SAS code and its corresponding log, respectively. It would be preferable to compile this information in a file format which a project team could reference.

With the Output Delivery System (ODS), you can export SAS data sets to output destinations for Microsoft Office files, and other formats used by common desktop software applications. The ODS EXCEL destination sends output objects to XLSX files, and allows extensive formatting of excel spreadsheets through its multitude of options.

For our project, it makes sense to output our results to an excel workbook which contains separate worksheets for the code report and the log report. The ODS EXCEL code used to perform this step is contained in Figure 14. In this excerpt of code, the reports are produced using PROC PRINT wrapped inside of an ODS shell.

```
ODS EXCEL FILE="/home/iyenj/SAS Papers/Code_Validation/Output_Files/CodeSample1_Metric_Report.xlsx"
OPTIONS(Sheet_Name='Code Metrics');

Proc Print Data=Code_Summary(Obs=25) Label;
     Var Directory_Path Program_Name Num_Lines Num_Lines_Code Num_Statements
          Num_DataSteps Num_Merge Num_ProcSteps Num_Comments;

     Label Program_Name = 'Name of SAS Program'
             Directory_Path = 'Directory of SAS Program'
             Num_Lines = 'Total Lines'
             Num_Lines_Code = 'Total Lines'
             Num_Statements = 'Number of SAS Statements'
             Num_DataSteps = 'Number of Data Steps'
             Num_Merge= 'Number of Data Step Merges'
             Num_ProcSteps= 'Number of Proc Steps'
             Num_Comments = 'Number of Comments';
Run;

ODS EXCEL OPTIONS(Sheet_Name='Log Performance Metrics');

Proc Print Data = Log_Metric_Final Label;
     Var Log_FileName Directory_Path OS_Memory Memory Real_Time System_Cpu_Time User_Cpu_Time;

     Label Log_FileName = 'Name of Log File'
             Directory_Path = 'Directory of Path';

     Title 'Log Performance Metrics';
Run;

ODS EXCEL CLOSE;
```

Figure 14. PROC PRINT Code wrapped in ODS EXCEL statements.

As shown in Figure 14, the ODS EXCEL code uses the SHEET_NAME= option to send reports to different tabs in an excel workbook.

Screenshots of the excel workbook and internal worksheet reports produced from the code in Figure 14 were created. The screenshots are displayed in Figure 15 below.

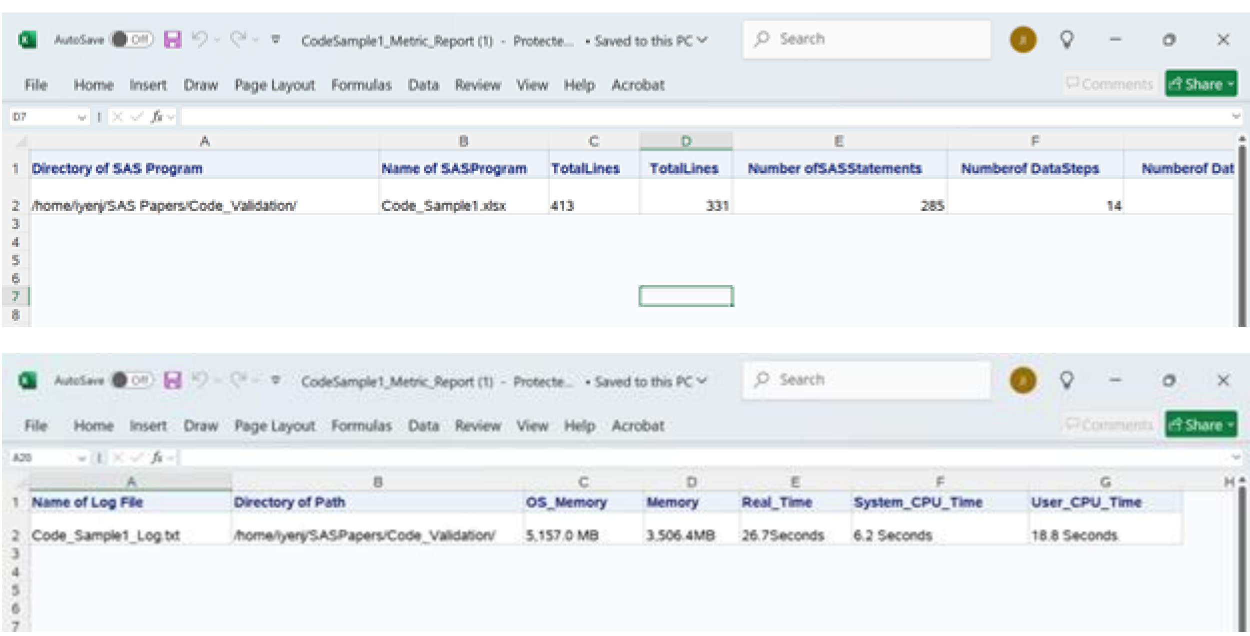

Figure 15. Excel worksheets of metric reports.

The information presented in the metric reports can be utilized to develop standards and benchmarks for code development on future projects. The information will prove valuable in evaluating whether existing code can be improved from an efficiency standpoint, and as a guide for projects whose objective is to streamline the existing base of code.

CONVERTING THE CODE INTO A MACRO

The examples we've illustrated have been applied to one SAS program file. Typically, on a project there's a series of SAS programs, which perform a variety of tasks, from data extraction, to manipulation, to reporting and graphical visualizations.

To produce performance measures for a series of SAS programs, the programs which have been developed for this project would be expanded and converted into a macro using SAS macro code. With macro coding, there's a series of steps to perform, from building the macro, choosing which elements need to be parameterized, and then testing and debugging the macro before its put into production.

Although not performed for this project, it's worthwhile to mention this additional project step as IT shops often have a large volume of existing code which needs to be analyzed for performance considerations. The case of a single SAS program has only been used here to illustrate how this process can be performed.

CONCLUSION

Harnessing and mining SAS programs and SAS logs for program performance measures is a useful exercise for IT and data analytic shops to perform. Undertaking this effort can lead to developing standards and benchmarks as well as revising best practices for efficient programming techniques. This paper presented a code analysis process which produces reports on SAS programs through extracting data from SAS code and logs. This process can be further expanded to cover additional metrics, and a series of SAS programs using macro code.

REFERENCES

Gillingham, Matthew. 'SAS Programming with Medicare Administrative Data' SAS Institute Press, 2014. https://support.sas.com/content/dam/SAS/support/en/books/sas-programming-with-medicare-administrative-data/64580_excerpt.pdf

Shah, Sapan and Agarwal, Sachin. 'Automatic Conversion of SAS Programs to Text Files for submissions to the FDA'. PharmaSUG 2020-Paper 209. https://www.lexjansen.com/pharmasug/2020 /QT/PharmaSUG-2020-QT-209.pdf

SAS 9.4 and SAS Viya Programming documentation, The SAS Institute. SAS 9.4 Companion for the Windows Environment, fifth edition. Features of the SAS language under Windows -> SAS System Options under Windows. https://documentation.sas.com /doc/en/pgmsascdc/9.4_3.5/hostwin.

ACKNOWLEDGMENTS

The author would like to thank Ajay Gupta, Academic Chair, Gary Moore, Operations Chair, Louise Hadden and Chary Akmyradov, Advanced Programming Section Co-Chairs, and the PharmaSUG Executive Committee and Conference Team for accepting my abstract and paper and for organizing this conference.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:
Jay Iyengar
Data Systems Consultants LLC
datasyscon@gmail.com
https://www.linkedin.com/in/datasysconsult/

Jay Iyengar is Director of Data Systems Consultants LLC. He's a SAS consultant, trainer, and SAS Certified Advanced Programmer. He's been an invited speaker at several SAS user group conferences (WIILSU, WCSUG, SESUG) and has presented papers and training seminars at SAS Global Forum, Pharmaceutical SAS Users Group (PharmaSUG), and other regional and local SAS User Group conferences (MWSUG, NESUG, WUSS, MISUG). He was co-leader and organizer of the Chicago SAS Users Group (WCSUG) from 2015-19. He received his bachelor's degree from Syracuse University in Public Policy and Economics, and his master's degree from the American University.

TRADEMARK CITATION

APPENDIX – SOURCE CODE

```
Options symbolgen;

Libname SRC '/home/iyenj/Training_Course_Files/Admin_hc_data/Data';
Libname FINAL '/home/iyenj/Training_Course_Files/Admin_hc_data/Data/Output';

            /* SORT CARRIER LINE FILE IN PREPARATION FOR TRANSFORMATION */
proc sort data=src.carrier2010line out=carrier2010line;
     by bene_id clm_id clm_ln;
run;

proc print data=carrier2010line(obs=3);
     where bene_id='00016F745862898F';
     var bene_id clm_id clm_ln line_icd_dgns_cd;
run;

            /* TRANSFORM CARRIER LINE FILE */
data carrier2010line_wide (drop=i expnsdt1 expnsdt2 hcpcs_cd line_icd_dgns_cd clm_ln linepmt prfnpi
                                      tax_num);
            format  expnsdt1_1-expnsdt1_13 mmddyy10. expnsdt2_1-expnsdt2_13 mmddyy10.
                        line_icd_dgns_cd1- line_icd_dgns_cd13 $5. hcpcs_cd1-hcpcs_cd13 $7.
                        linepmt1-linepmt13 10.2  prfnpi1-prfnpi13 $12.
                        tax_num1-tax_num13 $10.;

     set carrier2010line;

     by bene_id clm_id clm_ln;

     retain expnsdt1_1-expnsdt1_13 expnsdt2_1-expnsdt2_13 line_icd_dgns_cd1- line_icd_dgns_cd13
            hcpcs_cd1-hcpcs_cd13  linepmt1-linepmt13
            prfnpi1-prfnpi13  tax_num1-tax_num13;

     array   xline_icd_dgns_cd(13) line_icd_dgns_cd1-line_icd_dgns_cd13;
     array   xexpnsdt1_(13) expnsdt1_1-expnsdt1_13;
     array   xexpnsdt2_(13) expnsdt2_1-expnsdt2_13;
     array   xhcpcs_cd(13) hcpcs_cd1-hcpcs_cd13;
     array   xlinepmt(13) linepmt1-linepmt13;
     array   xprfnpi(13) prfnpi1-prfnpi13;
     array   xtax_num(13) tax_num1-tax_num13;

     if first.clm_id then do;
                        do i=1 to 13;
                                    xline_icd_dgns_cd(clm_ln)='';
                                    xexpnsdt1_(clm_ln)=.;
                                    xexpnsdt2_(clm_ln)=.;
                                    xhcpcs_cd(clm_ln)='';
                                    xlinepmt(clm_ln)=.;
                                    xprfnpi(clm_ln)='';
                                    xtax_num(clm_ln)='';
                        end;
     end;
```

```
                xline_icd_dgns_cd(clm_ln)=line_icd_dgns_cd;
                xexpnsdt1_(clm_ln)=expnsdt1;
                xexpnsdt2_(clm_ln)=expnsdt2;
                xhcpcs_cd(clm_ln)=hcpcs_cd;
                xlinepmt(clm_ln)=linepmt;
                xprfnpi(clm_ln)=prfnpi;
                xtax_num(clm_ln)=tax_num;

     if last.clm_id then output;
run;

proc print data=carrier2010line_wide(obs=1);
    var bene_id clm_id hcpcs_cd1 hcpcs_cd2 hcpcs_cd3 line_icd_dgns_cd1 line_icd_dgns_cd2
        line_icd_dgns_cd3;
    where bene_id='00016F745862898F';
    title "TRANSFORMED CARRIER LINE FILE";
run;

                /* SORT BASE CLAIM AND TRANSFORMED LINE FILES IN PREPARATION FOR MERGE */
proc sort data=src.carrier2010claim out=carrier2010claim;
    by bene_id clm_id;
run;

proc sort data=carrier2010line_wide;
    by bene_id clm_id;
run;

proc print data=carrier2010claim(obs=1);
    var bene_id clm_id from_dt thru_dt;
    where bene_id='00016F745862898F';
    title "SORTED CARRIER BASE CLAIM FILE";
run;

                /* MERGE BASE CLAIM AND TRANSFORMED LINE FILES */
data carr_2010 carr_nomatch;
    merge carrier2010claim(in=a) carrier2010line_wide(in=b);
      by bene_id clm_id;

    if a and b then output carr_2010;
    else output carr_nomatch;
run;

proc print data=carr_2010(obs=1);
    where bene_id='00016F745862898F';
    var bene_id clm_id hcpcs_cd1 hcpcs_cd2 hcpcs_cd3 line_icd_dgns_cd1 line_icd_dgns_cd2
        line_icd_dgns_cd3 from_dt thru_dt;
    title "MERGED TRANSFORMED CARRIER LINE AND BASE CLAIM FILES";
run;

proc datasets Library=Work;
    delete carrier2010line carrier2010line_wide carrier2010claim carr_nomatch;
quit;
```

```
            /* SORT INPATIENT REVENUE CENTER FILE IN PREPARATION FOR TRANSFORMATION */
proc sort data=src.ip2010line out=ip2010line;
    by bene_id clm_id clm_ln;
run;

proc print data=ip2010line(obs=10);
    var bene_id clm_id clm_ln hcpcs_cd;
    title "SORTED INPATIENT REVENUE CENTER FILE";
run;

                    /* TRANSFORM INPATIENT REVENUE CENTER FILE */
data ip2010line_wide(drop=i clm_ln hcpcs_cd);
    format  hcpcs_cd1-hcpcs_cd45 $5.;

    set ip2010line;

    by bene_id clm_id clm_ln;

    retain hcpcs_cd1-hcpcs_cd45;

    array xhcpcs_cd(45) hcpcs_cd1-hcpcs_cd45;

    if first.clm_id then do;
        do i=1 to 45;
          xhcpcs_cd(clm_ln)='';
        end;
    end;

    xhcpcs_cd(clm_ln)=hcpcs_cd;

    if last.clm_id then output;
run;

proc print data=ip2010line_wide(obs=2);
    var bene_id clm_id hcpcs_cd1 hcpcs_cd2 hcpcs_cd3;
    title "TRANSFORMED INPATIENT REVENUE CENTER FILE";
run;

             /* SORT BASE CLAIM AND TRANSFORMED REVENUE CENTER FILES */
proc sort data=src.ip2010claim out=ip2010claim;
    by bene_id clm_id;
run;

proc print data=ip2010claim(obs=10);
    var bene_id clm_id from_dt thru_dt;
    title "SORTED INPATIENT BASE CLAIM FILE";
run;

proc sort data=ip2010line_wide;
    by bene_id clm_id;
run;
```

```
          /* MERGE INPATIENT BASE CLAIM AND TRANSFORMED REVENUE CENTER FILES */
data ip_2010 ip_nomatch;
     merge ip2010claim(in=a) ip2010line_wide(in=b);
          by bene_id clm_id;
     if a and b then output ip_2010;
     else output ip_nomatch;
run;

proc print data=ip_2010(obs=2);
     var bene_id clm_id hcpcs_cd1 hcpcs_cd2 hcpcs_cd3 from_dt thru_dt;
     title "MERGED INPATIENT REVENUE CENTER AND BASE CLAIM FILES";
run;

proc datasets Library=Work;
     delete ip2010line ip2010line_wide ip2010claim ip_nomatch;
Quit;

           /* DME: COMBINE CLAIM HEADER AND CLAIM LINE FILES INTO A SINGLE FILE */
proc contents data=src.dm2010line;
run;

proc sort data=src.dm2010line out=dm2010line;
     by bene_id clm_id clm_ln;
run;

data dm2010line_wide(drop=i expnsdt1 expnsdt2 hcpcs_cd line_icd_dgns_cd clm_ln linepmt tax_num);
     format expnsdt1_1-expnsdt1_13 mmddyy10. expnsdt2_1-expnsdt2_13 mmddyy10.
            line_icd_dgns_cd1- line_icd_dgns_cd13 $5. hcpcs_cd1-hcpcs_cd13 $7.
            linepmt1-linepmt13 10.2  tax_num1-tax_num13 $10.;

     set dm2010line;

     by bene_id clm_id clm_ln;

     retain expnsdt1_1-expnsdt1_13 expnsdt2_1-expnsdt2_13
            line_icd_dgns_cd1- line_icd_dgns_cd13 hcpcs_cd1-hcpcs_cd13
            linepmt1-linepmt13 tax_num1-tax_num13;

     array  xline_icd_dgns_cd(13) line_icd_dgns_cd1-line_icd_dgns_cd13;
     array  xexpnsdt1_(13) expnsdt1_1-expnsdt1_13;
     array  xexpnsdt2_(13) expnsdt2_1-expnsdt2_13;
     array  xhcpcs_cd(13) hcpcs_cd1-hcpcs_cd13;
     array  xlinepmt(13) linepmt1-linepmt13;
     array  xtax_num(13) tax_num1-tax_num13;

     if first.clm_id then do;
          do i=1 to 13;
                    xline_icd_dgns_cd(clm_ln)='';
                    xexpnsdt1_(clm_ln)=.;
                    xexpnsdt2_(clm_ln)=.;
                    xhcpcs_cd(clm_ln)='';
                    xlinepmt(clm_ln)=.;
                    xtax_num(clm_ln)='';
```

```
            end;
      end;

      xline_icd_dgns_cd(clm_ln)=line_icd_dgns_cd;
      xexpnsdt1_(clm_ln)=expnsdt1;
      xexpnsdt2_(clm_ln)=expnsdt2;
      xhcpcs_cd(clm_ln)=hcpcs_cd;
      xlinepmt(clm_ln)=linepmt;
      xtax_num(clm_ln)=tax_num;

      if last.clm_id then output;
run;

        /* SORT HEADER AND LINE FILES AND MERGE TO CREATE ONE FULL RECORD PER CLAIM */
proc sort data=src.dm2010claim out=dm2010claim;
      by bene_id clm_id;
run;

proc sort data=dm2010line_wide;
     by bene_id clm_id;
run;

data dm_2010 dm_nomatch;
     merge dm2010claim(in=a) dm2010line_wide(in=b);
            by bene_id clm_id;
     if a and b then output dm_2010;
     else output dm_nomatch;
run;

proc datasets Library=Work;
     delete dm2010line dm2010line_wide dm2010claim dm_nomatch;
quit;

              /* OUTPATIENT: COMBINE CLAIM HEADER AND CLAIM LINE FILES INTO A SINGLE FILE */
proc contents data=src.op2010line;
run;

proc sort data=src.op2010line out=op2010line;
      by bene_id clm_id clm_ln;
run;

data op2010line_wide(drop=i clm_ln hcpcs_cd);
            format hcpcs_cd1-hcpcs_cd45 $5.;
     set op2010line;
        by bene_id clm_id clm_ln;

     retain hcpcs_cd1-hcpcs_cd45;

     array xhcpcs_cd(45) hcpcs_cd1-hcpcs_cd45;
```

```
    if first.clm_id then do;
        do i=1 to 45;
            xhcpcs_cd(clm_ln)='';
        end;
    end;

    xhcpcs_cd(clm_ln)=hcpcs_cd;

    if last.clm_id then output;
run;

    /* SORT HEADER AND LINE FILES AND MERGE TO CREATE ONE FULL RECORD PER CLAIM */
proc sort data=src.op2010claim out=op2010claim;
    by bene_id clm_id;
run;

proc sort data=op2010line_wide;
    by bene_id clm_id;
run;

data op_2010 op_nomatch;
    merge op2010claim(in=a) op2010line_wide(in=b);
        by bene_id clm_id;
    if a and b then output op_2010;
    else output op_nomatch;
run;

proc datasets library=work;
    delete op2010line op2010line_wide op2010claim op_nomatch;
quit;

            /* SNF: COMBINE CLAIM HEADER AND CLAIM LINE FILES INTO A SINGLE FILE */
proc contents data=src.sn2010line;
run;

proc sort data=src.sn2010line out=sn2010line;
    by bene_id clm_id clm_ln;
run;

data sn2010line_wide(drop=i clm_ln hcpcs_cd);
        format hcpcs_cd1-hcpcs_cd45 $5.;
    set sn2010line;

    by bene_id clm_id clm_ln;

    retain hcpcs_cd1-hcpcs_cd45;

    array xhcpcs_cd(45) hcpcs_cd1-hcpcs_cd45;

    if first.clm_id then do;
        do i=1 to 45;
            xhcpcs_cd(clm_ln)='';
        end;
    end;
```

```
        xhcpcs_cd(clm_ln)=hcpcs_cd;

        if last.clm_id then output;
run;


        /* SORT HEADER AND LINE FILES AND MERGE TO CREATE ONE FULL RECORD PER CLAIM */
proc sort data=src.sn2010claim out=sn2010claim;
       by bene_id clm_id;
run;

proc sort data=sn2010line_wide;
       by bene_id clm_id;
run;

data sn_2010 sn_nomatch;
       merge sn2010claim(in=a) sn2010line_wide(in=b);
              by bene_id clm_id;
       if a and b then output sn_2010;
       else output sn_nomatch;
run;

          /* HOME HEALTH: COMBINE CLAIM HEADER AND CLAIM LINE FILES INTO A SINGLE FILE */
proc contents data=src.hh2010line;
run;

proc sort data=src.hh2010line out=hh2010line;
       by bene_id clm_id clm_ln;
run;

data hh2010line_wide(drop=i clm_ln hcpcs_cd);
         format hcpcs_cd1-hcpcs_cd45 $5.;

       set hh2010line;

       by bene_id clm_id clm_ln;

       retain hcpcs_cd1-hcpcs_cd45;

       array xhcpcs_cd(45) hcpcs_cd1-hcpcs_cd45;

       if first.clm_id then do;
              do i=1 to 45;
                          xhcpcs_cd(clm_ln)='';
              end;
       end;

       xhcpcs_cd(clm_ln)=hcpcs_cd;

       if last.clm_id then output;
run;
```

```
     /* SORT HEADER AND LINE FILES AND MERGE TO CREATE ONE FULL RECORD PER CLAIM */
proc sort data=src.hh2010claim out=hh2010claim;
   by bene_id clm_id;
run;

proc sort data=hh2010line_wide;
   by bene_id clm_id;
run;

data hh_2010 hh_nomatch;
   merge hh2010claim(in=a) hh2010line_wide(in=b);
      by bene_id clm_id;
   If a and b then output hh_2010;
   else output hh_nomatch;
run;

          /* HOSPICE: COMBINE CLAIM HEADER AND CLAIM LINE FILES INTO A SINGLE FILE */
proc contents data=src.hs2010line;
run;

proc sort data=src.hs2010line out=hs2010line;
   by bene_id clm_id clm_ln;
run;

data hs2010line_wide(drop=i clm_ln hcpcs_cd);
     format hcpcs_cd1-hcpcs_cd45 $5.;
   set hs2010line;

   by bene_id clm_id clm_ln;

   retain hcpcs_cd1-hcpcs_cd45;

   array xhcpcs_cd(45) hcpcs_cd1-hcpcs_cd45;

   if first.clm_id then do;
      do i=1 to 45;
            xhcpcs_cd(clm_ln)='';
      end;
   end;

    xhcpcs_cd(clm_ln)=hcpcs_cd;

    if last.clm_id then output;
run;

      /* SORT HEADER AND LINE FILES AND MERGE TO CREATE ONE FULL RECORD PER CLAIM */
proc sort data=src.hs2010claim out=hs2010claim;
   by bene_id clm_id;
run;

proc sort data=hs2010line_wide;
   by bene_id clm_id;
run;
```

```
data hs_2010 hs_nomatch;
      merge hs2010claim(in=a) hs2010line_wide(in=b);
              by bene_id clm_id;
      if a and b then output hs_2010;
     else output hs_nomatch;
run;
```

APPENDIX II – Code Analysis Program 1 Log

```
1         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
72
73         /* Generated Code (IMPORT) */
74         /* Source File: Code_Sample1.xlsx */
75         /* Source Path: /home/iyenj/SAS Papers/Code_Validation */
76         /* Code generated on: 2/14/25, 5:04 PM */
77
78         Filename REFFILE '/home/iyenj/SAS Papers/Code_Validation/Code_Sample1.xlsx';
79
80         Proc Import Datafile=REFFILE
81           Dbms=XLSX
82           Out=Work.Code_Sample;
83           Datarow=1;
84           Getnames=No;
85         Run;

NOTE: One or more variables were converted because the data type is not supported by the V9 engine.
      For more details, run with options MSGLEVEL=I.

NOTE: The import data set has 413 observations and 1 variables.
NOTE: WORK.CODE_SAMPLE data set was successfully created.

NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.01 seconds
      user cpu time       0.01 seconds
      system cpu time     0.00 seconds
      memory              3418.50k
      OS Memory           28156.00k
86
87         Proc Contents Data=Work.Code_Sample;
88         Run;

NOTE: PROCEDURE CONTENTS used (Total process time):
      real time 0.04 seconds
      user cpu time 0.04 seconds
      system cpu time 0.00 seconds
      memory 3643.93k
      OS Memory 28332.00k
89
90         /* Define Macrovariable for File Directory */
91         Data _Null_;
92               Set Sashelp.VEXTFL (Where=(FILEREF='REFFILE'));
93
94                          /* Extract Filename from Xpath variable */
95               File_Name = Scan(Xpath, -1, '/');
96              Put File_Name= ;
97
98                          /* Extract Length Of Filename and Path */
99               File_Name_L = Length(File_Name);
100             Path_L = Length(XPath)-File_Name_L;
101             Put File_Name_L= Path_L= ;
102
```

```
                                /* Extract Directory Path */
          DirPath = Substr(XPath, 1, Path_L);
          Put DirPath= ;
                                /* Create Macro Variables For FileName and Directory Path */
         Call Symput ('DirPath', DirPath);
         Call Symput ('FileN', File_Name);
      Run;

File_Name=Code_Sample1.xlsx
File_Name_L=17 Path_L=39
DirPath=/home/iyenj/SAS Papers/Code_Validation/

NOTE: There were 1 observations read from the data set SASHELP.VEXTFL WHERE FILEREF='REFFILE';

NOTE: DATA statement used (Total process time):
      real time 0.00 seconds
      user cpu time 0.00 seconds
      system cpu time     0.00 seconds
      memory  5521.71k
      OS Memory         32936.00k

      %Put &DirPath= &FileN= ;
/home/iyenj/SAS Papers/Code_Validation/
Code_Sample1.xlsx
      /* Code_Validation-Example 2 */
      Data Code_Sample_v2;
          Set Code_Sample (Rename=(A=Line)) End=FINAL;
                   /* Compute Length of Code Line*/
      Line_Length = Length(Line);
      If _N_=1 Then Put Line_Length= ;
                   /* Count all Lines in Program, whether or not they contain code */
      LineCount+1;
                   /* Create Flag Indicator Variables for specific SAS constructs */
      If Line^=' ' Then Do;
      If Index(Line, ';')>0 Then SAS_Stmnt=1;
      Else SAS_Stmnt=0; *SAS Statements;
      If Index(Propcase(Line), 'Data ')>0 Then Data_Step=1;
      Else Data_Step=0; *Data Steps;
      If Index(Propcase(Line), 'Merge ')>0 Then DS_Merge=1;
      Else DS_Merge=0; *Data Step Merge;
      If Index(Propcase(Line), 'Proc ')>0 Then Proc_Step=1;
      Else Proc_Step=0;*SAS Procedures;
```

```
If (Substr(Line, 1, 2)='/*' or Substr(Line, Line_Length-1, 2)='*/') and
        Index(Propcase(Line), ';')=0 Then Comment=1;
Else Comment=0; *Comments;

            /* Records containing actual code */
Code_Line=1;
End;

Else Do;
 SAS_Stmnt=0;
 Data_Step=0;
 DS_Merge=0;
 Proc_Step=0;
 Comment=0;
 Code_Line=0;
End;

If Final=1 Then Call SymputX('TotCount', LineCount);

Run;
```

```
Line_Length=18

NOTE: There were 413 observations read from the data set WORK.CODE_SAMPLE.
NOTE: The data set WORK.CODE_SAMPLE_V2 has 413 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      user cpu time       0.01 seconds
      system cpu time     0.00 seconds
      memory              988.78k
      OS Memory           28844.00k
```

```

Proc Print Data=Code_Sample_V2 (Obs=25);
    Var Line LineCount Code_Line SAS_Stmnt Data_Step DS_Merge Proc_Step Comment;
Run;
```

```
NOTE: There were 25 observations read from the data set WORK.CODE_SAMPLE_V2.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.05 seconds
      user cpu time       0.05 seconds
      system cpu time     0.00 seconds
      memory              1468.21k
      OS Memory           29352.00k
```

```

Proc Sql;
Create Table Code_Summary as
Select "&DirPath" as Directory_Path,
```

```
NOTE: The quoted string currently being processed has become more than 262 bytes long.  You might have
unbalanced quotation marks.
           "&FileN" as Program_Name,
 NOTE: The quoted string currently being processed has become more than 262 bytes long.  You might have
unbalanced quotation marks.
           "&TotCount" as Num_Lines,
           Sum(Code_Line) as Num_Lines_Code,
           Sum(SAS_Stmnt) as Num_Statements,
           Sum(Data_Step) as Num_DataSteps,
           Sum(DS_Merge) as Num_Merge,
           Sum(Proc_Step) as Num_ProcSteps,
           Sum(Comment) as Num_Comments

           From Code_Sample_v2;
 NOTE: Table WORK.CODE_SUMMARY created, with 1 rows and 9 columns.
         Quit;
 NOTE: PROC SQL statements are executed immediately;

 NOTE: PROCEDURE SQL used (Total process time):
       real time           0.00 seconds
       user cpu time       0.01 seconds
       system cpu time     0.00 seconds
       memory              5674.50k
       OS Memory           34476.00k

         Proc Print Data=Code_Summary(Obs=25);
             Var Directory_Path Program_Name Num_Lines Num_Lines_Code Num_Statements
                 Num_DataSteps Num_Merge Num_ProcSteps Num_Comments;

           Label Directory_Path = 'Directory of SAS Program'
                 Program_Name = 'Name of SAS Program'
                 Num_Lines = 'Total Lines'
                 Num_Lines_Code = 'Total Lines'
                 Num_Statements = 'Number of SAS Statements'
                 Num_DataSteps = 'Number of Data Steps'
                 Num_Merge= 'Number of Data Step Merges'
                 Num_ProcSteps= 'Number of Proc Steps'
                 Num_Comments = 'Number of Comments';
         Run;

 NOTE: There were 1 observations read from the data set WORK.CODE_SUMMARY.
 NOTE: PROCEDURE PRINT used (Total process time):
       real time           0.01 seconds
       user cpu time       0.00 seconds
       system cpu time     0.00 seconds
       memory              723.18k
       OS Memory           29352.00k


         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;

```

APPENDIX III - Code Analysis Program 2 Log

```
OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
NOTE: ODS statements in the SAS Studio environment may disable some output features.

Options STIMER NOFULLSTIMER;
Libname CVFILES '/home/iyenj/SAS Papers/Code_Validation';
NOTE: Libref CVFILES was successfully assigned as follows:
      Engine:        V9
      Physical Name: /home/iyenj/SAS Papers/Code_Validation

Filename SASLOG '/home/iyenj/SAS Papers/Code_Validation/Code_Sample1_Log.txt';
/* Create SAS data set of SAS log */
Data CVFILES.CS1LOG;
Length LINEHDR $ 5 MESSAGE $ 250 CODE $ 245;
/* Read in SAS Log as an external file */
Infile SASLOG DLM=':' LRECL=275 TRUNCOVER;
Input @1 LINEHDR $5. @;
If (LINEHDR='NOTE' or LINEHDR=' ') Then
Input @7 MESSAGE $250.;
Else
   Input @12 CODE $245.;
Run;

NOTE: The infile SASLOG is:
      Filename=/home/iyenj/SAS Papers/Code_Validation/Code_Sample1_Log.txt,
      Owner Name=iyenj,Group Name=oda,
      Access Permission=-rw-r--r--,
      Last Modified=04Mar2025:13:04:21,
      File Size (bytes)=65636

NOTE: 1551 records were read from the infile SASLOG.
      The minimum record length was 1.
      The maximum record length was 133.

NOTE: The data set CVFILES.CS1LOG has 1551 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.01 seconds
            /* Define Macrovariable for File Directory */
Data _Null_;
   Set Sashelp.VEXTFL (Where=(FILEREF='SASLOG'));
           /* Extract Filename from Xpath variable */
   File_Name = Scan(Xpath, -1, '/');
   Put File_Name= ;
```

```
                /* Extract Length Of Filename and Path */
       File_Name_L = Length(File_Name);
       Path_L = Length(XPath)-File_Name_L;
       Put File_Name_L= Path_L= ;

                /* Extract Directory Path */
       DirPath = Substr(XPath, 1, Path_L);
       Put DirPath= ;

                /* Create Macro Variables For FileName and Directory Path */
       Call Sympt ('DirPath', DirPath);
       Call Symput ('FileN', File_Name);

       Run;

File_Name=Code_Sample1_Log.txt
File_Name_L=20 Path_L=39
DirPath=/home/iyenj/SAS Papers/Code_Validation/

NOTE: There were 1 observations read from the data set SASHELP.VEXTFL.
      WHERE FILEREF='SASLOG';

NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


       Proc Print Data = CVFILES.CS1LOG (OBS=25);
          Var LINEHDR MESSAGE;
          Title1 'Code - Line by Line';
          Title2 "Code Sample 1";
       Run;

NOTE: There were 25 observations read from the data set CVFILES.CS1LOG.

NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds


       /* Search and extract Performance Metrics from SAS log */
     Data CS1LOG;
        Set CVFILES.CS1LOG;

     Length Metric $20 Amount $15 Units $12;

                        /* Extract Real Time */
     If Substr(Message, 1, 9) = 'real time' Then Do;
           Metric = 'Real_Time';
           Amount = Scan(Message, 3, ' ');
           Units = Propcase(Scan(Message, 4, ' '));
     End;
```

```
                               /* Extract User CPU Time */
Else If Substr(Message, 1, 13)= 'user cpu time' Then Do;
          Metric = 'User_CPU_Time';
          Amount = Scan(Message, 4, ' ');
          Units = Propcase(Scan(Message, 5, ' '));
End;
                             /* Extract User CPU Time */
 Else If Substr(Message, 1, 15)='system cpu time' Then Do;
          Metric = 'System_CPU_Time';
          Amount = Scan(Message, 4, ' ');
          Units = Propcase(Scan(Message, 5, ' '));
 End;
                             /* Extract Memory */
 Else If Substr(Message, 1, 6)='memory' Then Do;
          Metric = 'Memory';
          Amount = Scan(Message, 2, ' ');
 End;
                             /* Extract OS Memory */
 Else If Substr(Message, 1, 9)='OS Memory' Then Do;
          Metric = 'OS_Memory';
          Amount = Scan(Message, 3, ' ');
 End;

                             /* Define Units for Memory */
 If Metric In('Memory', 'OS_Memory') Then Do;
          If Substr(Amount, Length(Amount), 1)='k' Then  Units = 'KB';
          Else If Substr(Amount, Length(Amount), 1)='m' Then  Units = 'MB';
          Else If Substr(Amount, Length(Amount), 1)='g' Then  Units = 'GB';
 End;

 If LINEHDR In ('NOTE', ' ');
 Run;

NOTE: There were 1551 observations read from the data set CVFILES.CS1LOG.
NOTE: The data set WORK.CS1LOG has 1112 observations and 6 variables.

NOTE: DATA statement used (Total process time):
      real time 0.00 seconds
      cpu time 0.00 seconds


   Proc Print Data = CS1LOG;
        Var LineHdr Message Metric Amount Units;
        Title 'Performance Metrics';
   Run;

NOTE: There were 1112 observations read from the data set WORK.CS1LOG.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          1.16 seconds
      cpu time           1.17 seconds
```

```
Data CS1LOG_V2;
      Set CS1LOG(Rename=(Amount=AmtOrig));

      Length AmountTmp1 $10;

                  /* Extract all characters but the last from Amount */
                  /* Convert Amount to Numeric */
      If Substr(AmtOrig, Length(AmtOrig), 1) In ('k', 'm', 'g')
      Then AmountTmp1 = Substr(AmtOrig, 1, Length(AmtOrig)-1);
      Else AmountTmp1 = AmtOrig;

      Amount=Input(Compress(AmountTmp1, , 's'), 10.2);

     Keep Metric Units Amount;

     If Metric^=' ';
Run;

NOTE: There were 1112 observations read from the data set WORK.CS1LOG.
NOTE: The data set WORK.CS1LOG_V2 has 260 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time 0.00 seconds
      cpu time 0.00 seconds

Proc Print Data = CS1LOG_V2 (Obs=20);
      Var Metric Units Amount;
Run;

NOTE: There were 20 observations read from the data set WORK.CS1LOG_V2.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.02 seconds
      cpu time           0.02 seconds

Proc Sql;
  Create Table Log_Stat_Sum as
  Select "&DirPath" as Directory_Path,
NOTE: The quoted string currently being processed has become more than 262 bytes long.  You might have
unbalanced quotation marks.
          "&FileN" as Log_FileName,
NOTE: The quoted string currently being processed has become more than 262 bytes long.  You might have
unbalanced quotation marks.
          Metric,
          Put(TotAmt, Comma10.1)||' '||New_Units as Total_Amount
  From
     (Select Metric,
             Units,
             Case
                When Units='KB' Then 'MB'
                Else Units
              End as New_Units,
```

```
209               Case
210                   When Units='KB' Then Sum(Amount)/1000
211                   Else Sum(Amount)
212                End as TotAmt Format=Comma10.1
213           From CS1LOG_V2
214           Group By Metric, Units)
215           Order By Directory_Path, Log_FileName;
NOTE: Table WORK.LOG_STAT_SUM created, with 5 rows and 4 columns.
215
216        Quit;
NOTE: PROCEDURE SQL used (Total process time):
        real time          0.00 seconds
        cpu time           0.01 seconds

217
218        Proc Transpose Data=Log_Stat_Sum Out=Log_Metric_Final(Drop=_NAME_);
219             Var Total_Amount;
220             ID Metric;
221             By Directory_Path Log_FileName;
222        Run;

NOTE: There were 5 observations read from the data set WORK.LOG_STAT_SUM.
NOTE: The data set WORK.LOG_METRIC_FINAL has 1 observations and 7 variables.
NOTE: PROCEDURE TRANSPOSE used (Total process time):
        real time          0.00 seconds
        cpu time           0.00 seconds

223
224        ODS EXCEL FILE="/home/iyenj/SAS Papers/Code_Validation/Output_Files /CodeSample1
                                 _Metric_Report.xlsx"
225        OPTIONS(Sheet_Name='Code Metrics');
226
227        Proc Print Data=Code_Summary(Obs=25) Label;
228             Var Directory_Path Program_Name Num_Lines Num_Lines_Code Num_Statements
229                 Num_DataSteps Num_Merge Num_ProcSteps Num_Comments;
230
231             Label Program_Name = 'Name of SAS Program'
232                   Directory_Path = 'Directory of SAS Program'
233                   Num_Lines = 'Total Lines'
234                   Num_Lines_Code = 'Total Lines'
235                   Num_Statements = 'Number of SAS Statements'
236                   Num_DataSteps = 'Number of Data Steps'
237                   Num_Merge= 'Number of Data Step Merges'
238                   Num_ProcSteps= 'Number of Proc Steps'
239                   Num_Comments = 'Number of Comments';
240        Run;

NOTE: There were 1 observations read from the data set WORK.CODE_SUMMARY.
NOTE: PROCEDURE PRINT used (Total process time):
        real time          0.02 seconds
        cpu time           0.02 seconds
```

```
ODS EXCEL OPTIONS(Sheet_Name='Log Performance Metrics');

Proc Print Data = Log_Metric_Final Label;
  Var Log_File Name Directory_Path OS_Memory Memory Real_Time System_Cpu_Time
      User_Cpu_Time;

  Label Log_FileName = 'Name of Log File'
        Directory_Path = 'Directory of Path';

  Title 'Log Performance Metrics';
Run;

NOTE: There were 1 observations read from the data set WORK.LOG_METRIC_FINAL.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds


ODS EXCEL CLOSE;
NOTE: Writing EXCEL file: /home/iyenj/SAS Papers/Code_Validation/Output_Files/CodeSample1_Metric
                          _Report.xlsx



OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```