PharmaSUG 2011 - Paper SAS-DM01

Choosing the Road Less Traveled: Performing Similar Tasks with either SAS® DATA Step Processing or with Base SAS® Procedures

Kathryn McLawhorn, SAS Institute Inc., Cary, NC

ABSTRACT

When you are seeking a solution that requires SAS code, do you choose the path that is marked with DATA step programming? Or, do you prefer the path that applies Base SAS procedures? Programmers often tend to use just one path, based on their programming skills and personal preferences. In many situations, you can use either DATA step logic or procedure code to arrive at the same solution.

To help broaden your horizons, this paper provides examples of common situations in which using either method leads to the same result. While this paper compares coding methods, it does not promote one particular method nor examine differences in efficiency. The discussion is aimed at intermediate to high-level users. Whether you are a loyal DATA step coder or a devoted proponent of procedure syntax, this paper offers some alternative techniques to help diversify your programming knowledge and skills.

INTRODUCTION

Everyday tasks can be accomplished in a variety of ways. Often the method that you choose is based on a combination of personal preference, past experience, and habit. The same is true for programming jobs in SAS. Rarely is there just one way that you can write your code to give you the expected result. Personal preference and programming skills determine how you choose to approach tasks in SAS.

This paper shows some common tasks that you can solve by using either DATA step logic or Base SAS procedures. Neither of these methods is more favorable nor more efficient than the other. You reach your final destination, which is your expected output, regardless of the path that you take. People typically favor one direction in programming because of their knowledge and skills. This paper offers alternative methods, illustrated by examples, for accomplishing common tasks. The goal is to broaden your understanding and encourage you to try other methods in your programming.

CALCULATING THE NUMBER OF OBSERVATIONS IN A BY GROUP

Suppose that you have a data set consisting of students' names and ages, and you want to distribute the students equally, based on age, among classes. To do that, you need to know the number of students in each category. Programmatically, you do that by calculating the number of observations in a BY group. The following sections illustrate two ways to accomplish this task: using a DATA step and using the FREQ procedure.

DATA STEP METHOD

In DATA step processing, you accomplish this task by using FIRST.*variable* (sometimes known as FIRST.*BYvar*) logic. This logic requires sorting (via the SORT procedure) the data set in advance, as shown in the DATA step example below.

This method creates a counter variable named COUNT that is initialized to **0**. The counter variable increases incrementally through all of the values in the BY group until the BY-variable changes. Finally, the last observation for the BY group is output.

DATA Step Example

```
proc sort data=sashelp.class out=class(keep=age);
   by age;
run;

data class1;
   set class;
   by age;
```

```
 if first.age then Count=0;
    count+1;
    if last.age then output;
 run;

proc print noobs;
run;
```

For this DATA step, the output that is generated by the PRINT procedure is as follows:

Age	Count
11	2
12	5
13	3
14	4
15	4
16	1

Table 1: Output for Calculating the Number of Observations in a BY Group

PROC FREQ METHOD

You can also output the number of observations for a grouping variable using the FREQ procedure. By default, this procedure returns the frequency count of any variables in the TABLES statement. In this example, PROC FREQ returns the count for AGE.

PROC FREQ Example

```
proc freq data=sashelp.class;
   tables age / out=new(drop=percent);
run;

proc print data=new noobs;
run;
```

This PROC FREQ method generates output that is identical to the output (Table 1) that is generated by the previous DATA step method.

COMPUTING A TOTAL FOR A BY GROUP

In this example, your data set contains scores for multiple players (ID) of two games (GAME 1 and GAME2). Each player can have multiple observations. You want to know the total score for each player for both of the games. To do that, you need to compute a total for each BY group.

DATA STEP METHOD

With the DATA step method, you must first sort the data set. This example uses PROC SORT to do that. In the second DATA step, the variables are renamed (GAME1=TEMP1; GAME2=TEMP2) so that they can be summed. The counter variables (GAME1 and GAME2) are then initialized to **0**. These variables increase incrementally until the BY value changes. The new variables are then added to the input numeric variables (GAME1 and GAME2) to generate cumulative values. Finally, the last observation in each BY group is output.

DATA Step Example

```
data scores;
   input ID $ Game1 Game2 ;
   cards;
A 2 3
A 5 6
B 1 2
C 1 2
C 4 5
C 7 8
;
run;
proc sort data=scores;
   by ID;
run;

data grandtot(drop=temp1 temp2);
   set scores(rename= (Game1=temp1 Game2=temp2));
   by id;
   if first.id then
      do;
         Game1=0;
         Game2=0;
      end;
   Game1 + temp1;
   Game2 + temp2;
   if last.ID then output;
run;

proc print data=grandtot noobs;
run;
```

PROC PRINT generates the following output:

ID	Game1	Game2
A	7	9
B	1	2
C	12	15

Table 2: Output for Computing a BY-Group Total

PROC MEANS METHOD

You can accomplish this same task using several Base SAS procedures:

- the MEANS procedure
- the REPORT procedure
- the SUMMARY procedure
- the TABULATE procedure

The following method illustrates the same task using PROC MEANS. In this procedure, the CLASS statement defines the grouping variable (ID). Using this statement in the procedure also eliminates the need to sort the data set in advance. The VAR statement defines the analysis variables (GAME 1 and GAME2). PROC MEANS calculates statistics for these variables, which must be numeric.

The NWAY option in the PROC MEANS statement affects the output data set and returns the highest level of interaction among the CLASS variables. Otherwise, the output data set includes all combinations of the CLASS variables. The OUTPUT statement requests the SUM statistic. In addition, it creates the output data set that contains the overall sum for the analysis variables within the unique values of the CLASS variable.

PROC MEANS Example

```
proc means data=scores noprint nway;
   class ID;
   var Game1 Game2;
   output out=new(drop=_type_ _freq_) sum=;
run;

proc print data=new noobs;
run;
```

Using this method, the output is, again, identical to the output (Table 2) that is generated with the DATA step method.

COMPUTING AN AVERAGE FOR A BY GROUP

The sample data set below contains two BY groups: I and J. You want to know the average of X for the unique values of J within the unique values of I. This example is useful, in particular, when you have multiple groups or categories for which you want to calculate a statistic such as the average.

DATA STEP METHOD

As is done in the previous examples, this next example also takes advantage of the power in using FIRST.*variable* logic. Similar to the process for calculating a total, the data set is sorted first and the counter variables (JSUB and FREQ) are initialized to **0**.

Next, JSUB (the cumulative value of X) is calculated within the BY value of J, and the FREQ variable is increased incrementally. The FREQ variable evaluates to the number of observations in each BY group. That is, the last value of the FREQ variable in each BY group (J) is the total number of observations in that group. This value is used to calculate the average.

DATA Step Example

```
data test;
   input I J X;
   cards;
1 1 123
1 1 3245
1 2 23
1 2 543
1 2 87
1 3 90
2 1 88
2 1 86
;
run;

proc sort data=test;
   by I J;
run;

data jsubtot (keep=I j freq average);
   set test;
   by I J;
   retain jsub Freq;
   if first.J then
      do;
         jsub=0;
         freq=0;
   end;
   jsub + X;
   freq + 1;
   if last.J then do;
```

```
      Average=jsub/freq;
      output;
   end;
run;

proc print data=jsubtot noobs;
run;
```

For this example, PROC PRINT generates output that shows the variables, the frequency, and the average.

I	J	Freq	Average
1	1	2	1684.00
1	2	3	217.67
1	3	1	90.00
2	1	2	87.00

Table 3: Output for Computing a BY-Group Average

PROC MEANS METHOD

As an alternative to the DATA step method, you can calculate the averages using the MEAN statistic (MEAN=AVERAGE) in a PROC MEANS OUTPUT statement, as shown in the example that follows. As explained in the previous example, the CLASS statement defines the grouping variables (I and J), and the VAR statement defines the analysis variable (X). The MEAN statistic is available also in PROC SUMMARY, PROC TABULATE, and PROC REPORT.

PROC MEANS Example

```
proc means data=test noprint nway;
   class I J;
   var x;
   output out=I_J(drop=_TYPE_ rename=(_freq_=Freq)) mean=Average;
run;

proc print data=I_J noobs;
run;
```

The output generated by this procedure is, again, identical to the output (Table 3) generated by the DATA step.

COMPUTING A PERCENTAGE FOR A BY GROUP

In this example, the data set includes sales figures, by region and by employee. You want to know what percentage of sales each employee contributed to the total sales of a region. To determine these figures, you need to compute a percentage for each BY group. In addition to the DATA step method, this section also describes how to accomplish this task with both the TABULATE and the REPORT procedures.

DATA STEP METHOD

When you use DATA step logic to calculate a percentage for a BY group, multiple steps are required, including an additional procedure step. In addition to sorting the data set, you must use PROC MEANS to compute the total of the numeric variable within each BY group to use as the denominator. This total is added back to the original data set with a MERGE statement in a subsequent DATA Step. Finally, a new variable calculates the percentage for each observation using the original numeric variable and the denominator variable.

DATA Step Example

```
data sales;
   input @1 Region $char8.  @10 Repid 4.  @15 Amount 10. ;
   format Amount dollar12.;
   cards;
NORTH    1001 1000000
NORTH    1002 1100000
NORTH    1003 1550000
NORTH    1008 1250000
NORTH    1005  900000
SOUTH    1007 2105000
SOUTH    1010  875000
SOUTH    1012 1655000
EAST     1051 2508000
EAST     1055 1805000
;
run;

proc sort data=sales;
   by Region;
run;

proc means data=sales noprint nway;
   by Region;
   var Amount;
   output out=Regtot(keep=Regtotal Region) sum=Regtotal;
run;

data percent1;
   merge sales Regtot;
   by Region;
   Regpct=(Amount / Regtotal ) * 100;
   format regpct 6.2 Amount Regtotal dollar10.;
run;

proc print data=percent1 noobs;
run;
```

In this example, PROC PRINT generates the following table:

Region	RepID	Amount	Regtotal	Regpct
EAST	1051	$2,508,000	$4,313,000	58.15
EAST	1055	$1,805,000	$4,313,000	41.85
NORTH	1001	$1,000,000	$5,800,000	17.24
NORTH	1002	$1,100,000	$5,800,000	18.97
NORTH	1003	$1,550,000	$5,800,000	26.72
NORTH	1008	$1,250,000	$5,800,000	21.55
NORTH	1005	$900,000	$5,800,000	15.52
SOUTH	1007	$2,105,000	$4,635,000	45.42
SOUTH	1010	$875,000	$4,635,000	18.88
SOUTH	1012	$1,655,000	$4,635,000	35.71

Table 4: Computing a Percentage for a BY Group with DATA Step Processing

PROC TABULATE METHOD

PROC TABULATE contains the PCTSUM statistic that calculates a percentage by default. The denominator definition, enclosed in angle brackets (PCTSUM<REPID>), defines the grouping variable that is the basis for the denominator. The functions of the statements in PROC TABULATE are similar to those mentioned in the previous example for PROC MEANS:

- The CLASS statement defines the grouping variables.
- The VAR statement defines the analysis variable.
- The TABLE statement defines the structure of the output table.

Note: If you want to generate a data set from the PROC TABULATE results, add the OUT= option in the PROC TABULATE statement.

PROC TABULATE Example

```
proc tabulate data=sales out=out_tab;
   class Region Repid;
   var Amount;
   table Region*Repid, Amount*(Sum*f=dollar12. Pctsum<Repid>);
run;
```

The output generated by this procedure is as follows:

		Amount	
		Sum	PctSum
Region	RepID		
EAST	1051	$2,508,000	58.15
	1055	$1,805,000	41.85
NORTH	1001	$1,000,000	17.24
	1002	$1,100,000	18.97
	1003	$1,550,000	26.72
	1005	$900,000	15.52
	1008	$1,250,000	21.55
SOUTH	1007	$2,105,000	45.42
	1010	$875,000	18.88
	1012	$1,655,000	35.71

Table 5: Computing a Percentage for a BY Group Using PROC TABULATE

PROC REPORT METHOD

PROC REPORT also enables you to compute a percentage. In this procedure, the COLUMN statement lists the variables included in the table, while the DEFINE statements indicate how the variables are used. Variables in PROC REPORT are either grouping, analysis, or computed variables. Additional options in the DEFINE statements control appearance (for example, labels, format, and column width).

This example also uses two compute blocks:

- The first compute block contains programming statements that PROC REPORT executes as it builds the report. The block calculates the total of the analysis variable within a grouping variable. This total becomes the denominator value.
- The second compute block calculates the percentage by dividing the analysis variable by the denominator.

Note: If you want to generate a data set from the PROC REPORT results, add the OUT= option in the PROC REPORT statement.

PROC REPORT Example

```
proc report data=sales nowd out=out_rep;
   column Region Repid Amount Regpct;
   define Region / order;
   define Repid / display;
   define Amount / sum format=dollar12.;
   define Regpct / computed format=percent8.2;

   compute before Region;
      hold=Amount.sum;
   endcomp;

   compute Regpct;
      regpct=Amount.sum/hold;
   endcomp;
run;
```

The format of the PROC REPORT output is slightly different and is not shown, but the results for this procedure are the same as that for PROC TABULATE (Table 5).

DISPLAYING VALUES THAT DO NOT EXIST IN THE DATA

When you display your data, you might want to present a complete picture of all possible values that can occur for a particular variable. For this example, suppose your data contains students' grades, and the grade values that you want to show are **A**, **B**, **C**, **D**, and **F**. However, what do you do if the values for the students in the data set do not contain any of **C** or **F** values?

DATA STEP METHOD

With a DATA step, you can add values that do not exist by reading in the entire data set and adding new observations (for the **C** or **F** values) after you reach the end of the file. In the following example, the SET statement includes the END=EOF option, which detects when the program reaches the end of the data set. The program reads the entire data set. When the end of the data set is detected, new observations are added to create a data set that contains all possible values for the GRADES variable.

DATA Step Example

```
data student;
   input Grades $;
   cards;
A
B
D
;
run;

data student;
   set student end=eof;
   output;
   if eof then do;
      Grades='C'; output;
      Grades='F'; output;
   end;
run;

proc print data=student noobs;
run;
```

PROC PRINT generates output showing the data set with the all observations.

Grades
A
B
D
C
F

Table 6: Displaying GRADE Values That Do Not Exist in the STUDENT Data Set

PROC FORMAT AND PROC MEANS METHOD

Some Base SAS procedures (including PROC MEANS, PROC REPORT, PROC TABULATE, and PROC SUMMARY) support the PRELOADFMT option, which provides the ability to include values in the output that do not exist in the data. In this example, both the FORMAT and the MEANS procedures are used. PROC FORMAT builds all of the possible values. Then, the PRELOADFMT option is used, along with other procedure options, in PROC MEANS. **Note:** In PROC MEANS, you must use the COMPLETETYPES option in the PROC MEANS statement when you use the PRELOADFMT option.

PROC FORMAT and PROC MEANS Example

```
proc format;
   value $Grade 'A'='A' 'B'='B' 'C'='C'
                'D'='D' 'F'='F';
run;
proc means data=student nway noprint completetypes;
   class Grades / preloadfmt;
   format Grades $Grade.;
   output out=new(drop=_type_ _freq_);
run;

proc print data=new noobs;
run;
```

For this procedure example, the output is identical to that generated for the DATA step (Table 6), except for the order. You can adjust the order of the values by sorting the STUDENT data set.

CREATING A CUSTOM REPORT

After you manipulate your data set, you might want to create presentation-quality output of that data set. For example, you might want to display your data in a table as part of a report that you are preparing. Creating such output requires adding headings and aligning the data in columns, which you can achieve by using either DATA step programming or the REPORT procedure.

DATA STEP METHOD

In this example, the DATA step uses the PUT statement to create a custom report that is column-aligned and that includes a header. Column-style output specifies the starting and ending column numbers after the variable name. The PUT statement also uses a pointer control (*@n)* to align data in columns. In the FILE statement, the HEADER= option defines the header for the output.

Notice that two RETURN statements are required: one before and one after the header definition. This group of statements executes each time SAS begins a new page.

DATA Step Example

```
proc sort data=sashelp.class out=sorted;
   by sex;
run;
```

```
data _null_;
   set sorted;
   by sex;
   file print header=h notitles;
   if first.sex then put _page_;
   put name 1-8 age 13-15 @20 sex +5 weight 5.1 +5 height;

   return;
      H:
      n+1;
      put @35 "Page " n " of 2" //;
      put @1 "Name" @13 "Age" @20 "Sex" @26 "Weight" @36 "Height";
      put @1 "----" @13 "---" @20 "---" @26 "------" @36 "------";
   return;

run;
```

Aside from a few formatting differences, the appearance of the DATA step output is similar to that of the PROC REPORT output (Table 7).

PROC REPORT METHOD

PROC REPORT works in a way similar to that of a DATA step that contains PUT statements. The following procedure creates custom, print-quality reports in which the variables are automatically placed in columns. As mentioned previously in "Computing a Percentage for a BY Group," the COLUMN statement lists the variables that are included in the output. The DEFINE statements also indicate the use of the variables. The BREAK statement defines an action when the value of the grouping variable changes. In this case, the defined action is to create a page break.

PROC REPORT Example

```
proc report data=sorted nowd;
   column sex name age sex=sex1 weight height;
   define sex / order noprint;
   define name / order 'Name/--';
   define age / order 'Age/--' width=4;
   define sex1 / order 'Sex/--' width=3;
   define weight / display 'Weight/--' width=7;
   define height / display 'Height/--' width=7;
   break after sex / page;
run;
```

Table 7 is partial output that is generated by this REPORT procedure.

Name	Age	Sex	Weight	Height
Alice	13	F	84	56.5
Barbara	13	F	98	65.3
Carol	14	F	102.5	62.8
Jane	12	F	84.5	59.8
Janet	15	F	112.5	62.5
Joyce	11	F	50.5	51.3
Judy	14	F	90	64.3
Louise	12	F	77	56.3
Mary	15	F	112	66.5

Table 7: A Custom Report Created by the REPORT Procedure

COLLAPSE OBSERVATIONS IN A BY GROUP INTO A SINGLE OBSERVATION

Quite often, your data set might not be in the format that you want. For example, suppose you have a BY group that has multiple observations. You might want to collapse those observations in a BY group into a single observation, as shown in the following DATA step and TRANSPOSE procedure examples.

DATA STEP METHOD

The DATA Step method to reshape your data incorporates ARRAY logic. The RETAIN statement prevents the variables from being set to missing at the top of each iteration of the DATA step. Next, the ARRAY statement names the new variables, and the DO loop clears the array from one BY group to the next. Then, values are assigned to the array elements. Finally, the last value in the BY group is output.

DATA STEP Example

```
data students;
   input Name:$ Score;
   cards;
Deborah      89
Deborah      90
Deborah      95
Martin       90
Stefan       89
Stefan       76
;
run;

data scores(keep=Name Score1-Score3);
   retain name Score1-Score3;
   array scores(*) Score1-Score3;
   set students;
   by name;
   if first.name then do;
      i=1;
      do j=1 to 3;
         scores(j)=.;
      end;
   end;
   scores(i)=score;
   if last.name then output;
   i+1;
run;

proc print noobs;
run;
```

This code generates the following table:

Name	Score1	Score2	Score3
Deborah	89	90	95
Martin	90	.	.
Stefan	89	76	.

Table 8: Collapsing Multiple Observations into a Single Observation

PROC TRANSPOSE METHOD

The TRANSPOSE procedure reshapes data by changing observations into variables and changing variables into observations. In this example, the BY statement defines the variable for which you want to create one observation. The VAR statement lists the variables that are to be transposed.

Note: PROC TRANSPOSE does not create printed output. It uses the OUT= option to create an output data set. You can view this data set using PROC PRINT. The PREFIX= option names the variables in the output data set.

PROC TRANSPOSE Example

```
proc transpose data=students out=new(drop=_name_) prefix=score;
   by name;
   var score;
run;

proc print data=new noobs;
run;
```

The output that you can view with PROC PRINT is identical to the output in Table 8.

CONCLUSION

Both DATA step logic and Base SAS procedure code are resourceful tools for accomplishing tasks in SAS. As this paper shows, both methods lead to identical results in some cases; in other cases, the results are fairly similar. These examples provide you with more alternatives, but the route that you take is your choice. However, with new-found diversity in your programming knowledge, you might find yourself taking a different path the next time you tackle a SAS issue! Learning new techniques and skills when you choose the road less traveled makes the journey worth the effort.

CONTACT INFORMATION

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

Kathryn McLawhorn
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
E-mail: support@sas.com
Web: support.sas.com