Paper 18-27

Advanced Tips and Techniques with PROC MEANS

Andrew H. Karp
Sierra Information Services, Inc.
Sonoma, California USA

PROC MEANS (and is "sister," PROC SUMMARY) have been base SAS® Software procedures for a long time. Most SAS Software users have found their ability to rapidly analyze and summarize the values of numeric variables to be essential tools in their programs and applications. A number of new features have been added to PROC MEANS in The Nashville Release (Version 8) of the SAS System which will are discussed in this paper. Applying these new features will enable you to simplify the analysis of your data, make it easier for you to create data sets containing summaries/analyses of your data, and take advantage other tools in SAS Version 8, such as Multilabel Formats (MLFs), which are created by using PROC FORMAT.

Background

This paper assumes the reader is already familiar with core PROC MEANS/SUMMARY capabilities. As a reminder, with the release of Version 6 of the SAS System in 1991 PROC MEANS and PROC SUMMARY became identical procedures, with just some very minor differences that are documented in the base SAS Software documentation. For the purposes of this paper we can treat them as identical procedures. The core difference is that by default PROC MEANS sends the results of its "work" to our Output Window and that PROC SUMMARY, by default, creates a SAS data set. All of the examples in this paper show PROC MEANS syntax, which you can easily switch to PROC SUMMARY if you want.

The core function of these procedures is to analyze the values of numeric variables in SAS data sets (or SAS views to data stored in other RDBMS products). For both PROCs, only numeric variables can be placed in the VAR statement. Variables places in the VAR statement are considered *analysis variables.* If you put the names of character variables in the VAR statement, PROC MEANS/SUMMARY will not execute and an error will be shown in your SASLOG.

Variables placed in the CLASS or BY Statement are considered *classification variables*, and may be either character or numeric. Starting in Version 6, you could use the CLASS statement to request analyses at the different levels of an unsorted classification variable by using the CLASS, instead of the BY Statement. This feature remains in Version 8, and enhancements to it are discussed below.

New Statistics Available in Version 8

Prior to Version 8, the only base SAS procedure that could calculate the values of *quantile statistics* such as the median (50th percentile) was PROC UNIVARIATE. In Version 8, PROCs MEANS and SUMMARY (as well as PROC TABULATE) can also analyze and report the values of quantile statistics.

Consider the following PROC MEANS task, which analyzes a data set containing electric consumption data from a public utility.

```
proc MEANS NOPRINT
data=electric.elec_V8;
class rate_schedule;
var total_revenue;
output out=new2 sum=median_REV
mean=total_REV p50=mean_REV;
run;
```

The OUTPUT Statement directs the placement, in a temporary SAS data set, of new variables containing the results of the analylses requested elsewhere in the PROC MEANS Task. The *Statistics Keyword* **P50** requests the fiftieth percentile, or the median, of the analysis variable be placed in a variable called mean_REV in output data set new2.

This PROC MEANS task will run without errors. Mean_REV is a valid SAS variable name, even though what we are doing is storing the median in a variable called "mean_REV." If you look at the OUTPUT Statement again you will see that the sum will be stored in "Median_REV" and the mean will be stored in "Total_REV," all valid SAS variable names in Version 8. (Remember, in V8 we can use up to 32 characters for variables names,) You would probably not notice your mistake until sometime later in the project when you--or perhaps your boss--realize that the values associated with the variable names simply don't "make sense." A new Version 8 feature, discussed below, will keep you from ever making this mistake again. It's called the AUTONAME option.

The AUTONAME Option

This handy option goes in the OUTPUT statement. When you use it, PROC MEANS will automatically give names to the variables in the output SAS data sets. Here is an example, again using the electric consumption data set.

```
proc means noprint
data=x.electricity;
class division serial;
var kwh1 rev1;
output out=new4 mean(kwh1) =
             sum(rev1) =/autoname;
run;
```

Output data set NEW4 contains the classification variables DIVISION and SERIAL, the automatically generated variables _TYPE_ and _FREQ_ (about which more later), and the variables KWH1_mean and REV1_mean. The AUTONAME option automatically appended the statistics keyword to the name of the analysis variable, with an underscore symbol between the analysis variable and the statistics keyword. Using this new option will prevent you from giving variables in our output data sets the "wrong" names!

The CHARTYPE Option

This V8 addition to PROC MEANS dramatically simplifies creation of multiple output SAS data sets in a single use of PROC MEANS.

Users can code multiple OUTPUT statements in a single PROC MEANS task and then use a WHERE clause data set option to limit the output of observations to each data set. This capability reduces--and often eliminates--the need to run PROC MEANS several times to create different output data sets. You can usually do everything you need to do in one invocation of the procedure.

Here is an example. A catalog retail firm has a SAS data set containing order records and an analyst wants to create several output SAS data sets, using PROC MEANS, containing analyses at different combinations of the values of four classification variables. Consider the following PROC MEANS task:

```
proc means noprint
data=order.orderfile2;class mailcode
dept_nbr segment status;var itmprice
itm_qty;output out=new  sum=;run;
```

With four variables in the CLASS statement, the temporary SAS data set, NEW, created by the OUTPUT statement, will contain the sum of analysis variables ITMPRICE and ITM_QTY at all possible combinations of the classification variables. There will be sixteen values of the SAS-generated variable _TYPE_ in output data set NEW, representing what I like to call a "complete" analysis of the numeric analysis variables at all possible combinations of the classification variables.

Suppose the analyst wanted to create three separate output SAS data sets, containing the sum of ITMPRICE and ITM_QTY. The separate data sets would contain the desired analyses

By MAILCODE and SEGMENT
By MAILCODE, SEGMENT and STATUS
By DEPT_NBR and SEGMENT

One approach would be to run PROC MEANS three separate times, with different classification variables in the CLASS Statement. When working with very large data sets, this approach, while giving the desired results, wastes computing resources as the source data set will be read three separate times.

Another approach would be to create "one big data set," containing the "complete" analysis, as shown above, and then use a data step to read each observation and output some of them to various subset data sets. Using the previous example, the observations in temporary data set NEW would be tested in a data step and those meeting the condition of interest would be output to new data sets. While this approach still yields the desired outcome, what the analyst would have to do is create a big data set and then test each of its observations to find the ones she wants to put in the smaller data sets. Here is an example:

```
Data A B C:
  SET NEW;
   IF _TYPE_ = '1001'B then OUTPUT A;
ELSE IF _TYPE_ = '1011'B
      then OUTPUT B;
ELSE IF _TYPE_ = '0110'B then
OUTPUT 'C';
RUN;
```

This data step shows an underutilized feature of the SAS Programming Language, which is called the "bit-testing facility." By using it, we don't have to know the numeric value of _TYPE_, we just need to know the position of the classification variables in the CLASS statement. In this context, a number one means "I want this variable," and a number zero means "I don't want this variable." You can check for yourself that 1001 in base 2 is equal to nine (9) in base 10.

A third approach would be to determine the numeric values of the desired values of _TYPE_ and put them in the output statement as WHERE clause conditions.

The new **CHARTYPE** option makes all of this unnecessary. This option, placed in the PROC MEANS statement, converts the (default) numeric values of _TYPE_ to a character variable containing zeros and ones. The length of this variable is equal to the number of variables in the CLASS (or BY) statement. Keep in mind, that even though this variable contains zeros and ones, it is a character variable.

Using the **CHARTYPE** option makes it much easier to create multiple output data sets in a single application of PROC MEANS. Our marketing analyst could make the three data sets she wants by submitting the following PROC MEANS task:

```
proc means noprint data=order.orderfile2
CHARTYPE;class mailcode dept_nbr segment
status;var itmprice itm_qty;output
out=one(where=(_type_ = '1001'))
        sum=;output out=two(where=(_type_ =
'1011'))
        sum=;output out=three(where=(_type_ =
'0110'))
        sum = ;run;
```

By using the **CHARTYPE** option, the analyst easily creates the three desired data sets in a single use of PROC MEANS. As with the previously-described bit-testing facility in the SAS Programming Language, a one (1) means "I want this variable" and a zero (0) meams "I don't want this variable." Again this is a character, not a numeric variable, and in order to use it effectively you need to know the ordering of the variables in your CLASS statement.

The DESCENDTYPES Option

By default, observations in data sets created by PROC MEANS are ordered in ascending values of the variable _TYPE_. So, _TYPE_ = 0 will be first, followed by _TYPE_ = 1, and so forth, with the highest value of _TYPE_ at the bottom.

The new **DESCENDTYPES** option, which is placed in the PROC MEANS statement, instructs the procedure to order observations in the output data sets it creates in descending value of _TYPE_.

This option is very handy if you want the observation with _TYPE_ = 0 at the bottom of your data set, rather than at the top.

Combining the CHARTYPE and DESCENDTYPES Options

The previous example of how to use the CHARTYPE option and several OUTPUT statements to create multiple SAS data sets demonstrates a very useful way to avoid coding multiple PROC MEANS tasks to generate s series of output SAS data sets. When you create several output data sets in a single PROC MEANS task, the observations with the same value of _TYPE_ can be output to more than one data set.

Let's use the previous PROC MEANS task to provide an example. In that example three output temporary SAS data sets were created, each containing the SUMs of the analysis variables at different combinations of the CLASS statement variables. Suppose, however, that we needed to put the grand totals (i.e., the "grand sum') in each of these data sets. So, we would to include the observation where _TYPE_ = 0 to in each output data set. And, we want to put that observation at the bottom of the output data sets (because, perhaps, we are going to subsequently export them to Excel™ spreadsheets we'd like to avoid PROC SORT steps before exporting the data from SAS to Excel).

We can accomplish these tasks by: a) specifying the DESCENDTYPES option in the PROC MEANS statement; and, 2) using the IN operator in conjunction with the WHERE clauses used to select observations for the three output data sets created by PROC MEANS.

The following PROC MEANS task implements the desired results.

```
proc means noprint data=order.orderfile2
CHARTYPE DESCENDTYPES;class mailcode
dept_nbr segment status;var itmprice
itm_qty;output out=one(where=(_type_
IN('0000','1001'))) sum=;output
out=two(where=(_type_ IN('0000','1011')))
sum=;output out=three(where=(_type_
IN('0000','0110'))) sum = ;run;
```

The TYPES Statement

This Version 8 enhancement to PROC MEANS should not be confused with the _TYPE_ variable discussed previously.

By default, PROC MEANS will analyze the numeric analysis variables at all possible combinations of the values of the classification variables. With the **TYPES** statement, only the analyses specified in it are carried out by PROC MEANS. This new feature can save you both programming and processing time.

The next example shows how the **TYPES** statement is used to restrict the operation of PROC MEANS to analyzing the values of the analysis variables to just the combination(s) of classification variables in the CLASS or BY Statement. In PROC MEANS task that follows, the marketing analyst working with the previous-discussed order history file wants to create a single output data set containing analyses of ITMPRICE and ITM_QTY at the following combinations of the CLASS variables

Overall (_TYPE_ = 0)
SEGMENT and STATUS
MAILCODE and SEGMENT
MAILCODE and DEPT_NBR and SEGMENT

The **TYPES** Statement shown below limits the execution of the PROC MEANS task to just the combinations of the classification variables specified in this statement. Because only only output data set is requested in this task, temporary data set A will contain all of the analyses requested by **TYPES** statement.

```
proc means noprint
data=order.orderfile2 ;class mailcode
dept_nbr segment status;types ()
segment * status      mailcode *
segment      mailcode * dept_nbr *
segment;var itmprice itm_qty;output
out=a sum = ;run;
```

More complex implementations of the **TYPES** statement are documented in the PROC MEANS chapter in the SAS Procedures documentation.

Multiple CLASS Statements

Multiple CLASS Statements are now permitted in PROC MEANS. In previous releases of SAS System software the values of classification variables were either portrayed in the Output Window, or had their values stored in output data sets, in "sort order." New options in the CLASS statement permit user control over how the levels of the classification variables are portrayed.

When using multiple CLASS statements, how you order them in the PROC MEANS task is very important. If you have two CLASS statements, for example, the values of the classification variables in the *second* CLASS statement will be *nested within* the values of the variables in the *first* class statement.

The next PROC MEANS task shows how two CLASS statements were used to analyze some electrical utility data stored in a SAS data set. The first CLASS statement requests an analysis by the values of REGION. The DESCENDING option to the right of the slash in the first CLASS statement instructs PROC MEANS to analyze the data in DESCENDING order of the values of REGION.

The second CLASS statement requests that the data be analyzed by the values of the classification variable TRANSFORMER. Since no options are specified in the second CLASS statement, the values of TRANSFORMER will be portrayed in "sort order," within the descending values of REGION.

```
proc means
data=electric.elec_v8 noprint nway;
class region/descending;
class transformer;
var total_revenue ;
output out=c sum= mean= /autoname;
run;
```

Additional CLASS Statement Options

There are several other useful options now available in the CLASS Statement. A complete list is found on pages 636-638 of the Version 8 SAS Procedures Guide within Chapter 24 (The Means Procedure). Of these, we will discuss the DESCENDING. EXCLUSIVE, GROUPINTERNAL, MISSING, MLF, ORDER= and PRELOADFMT options.

- ***The DESCENDING Option***

This option, discussed above, orders the levels of the CLASS variables in descending order.

- ***The EXCLUSIVE Option***

This option, used in conjunction with the **PRELOADFMT** option excludes from analysis all the combinations of CLASS variables that are not in the preloaded range of user-defined formats (see the **PRELOADFMT** option, below

- ***The GROUPINTERNAL Option***

This option, which saves computing resources when your numeric classification variables contain discrete values, tells PROC MEANS to NOT apply formats to the class variables when it groups the values to create combinations of CLASS variables.

- ***The MISSING Option***

This option instructs PROC MEANS to consider as valued values missing values for variables in the CLASS statement. If you do not use the MISSING option, PROC MEANS will *exclude observations with a missing CLASS variable value from the analysis.* A related PROC MEANS option, **COMPLETETYPES,** is discussed later on in this paper.

- ***The MLF Option***

The new MLF option permits PROC MEANS to use the primary and secondary format labels to create subgroup combinations when a mulitlabel format is assigned to variable(s) in the CLASS statement. For more information on the new mutilabel formats now available in Version 8, please consult the PROC FORMAT documentation.

Some patience and testing is often required when using Multilabel Formats with PROC MEANS (as well as when using it with other PROCS that support its use, such as PROCs REPORT and TABULATE.) In Version 8, there are differences in how PROC MEANS orders the rows (observations) in output where a MLF has been applied depending on whether the analysis is reported in the Output Window or stored in an output SAS data set.

- ***The ORDER= Option***

This option specifies the order PROC MEANS will group the levels of the classification variables in the output it generates. (See above for a discussion of the **DESCENDING** option in the CLASS Statement.) The arguments to the **ORDER=** option are: **DATA, FORMATTED, FREQ** and **UNFORMATTED.** Notice that DESCENDING is not an argument to this option, but is a "standalone" option within the CLASS statement.

- ***The PRELOADFMT Option***

This option instructs the SAS System to pre-load formats in memory before executing the statements in PROC MEANS. In order to use it, you must also specify either the **COMPLETETYPES, EXCLUSIVE or ORDER=DATA** options. Using **PRELOADFMT** in conjunction with, for example, the **COMPLETETYPES** option creates and outputs all combinations of class variables even if the combination does not occur in the input data set. By specifying both **PRELOADFMT** in the CLASS statement and **COMPLETETYPES** option in the PROC MEANS statement, your output will include all combinations of the classification variables, including those with no observations.

The COMPLETETYPES and EXCLUSIVE Options

As discussed above, the **COMPLETETYPES** option instructs the procedure to create all possible combinations of the values of the classification variables, even if that combination does not exist in the data set. It is most often used with CLASS statement options such as **PRELOADFMT.** The **EXCLUSIVE** option is used in conjunction with the **CLASSDATA=** option (also new to Version 8, and discussed below) to include any observations in the input data set whose combination of classification variable values are not in the **CLASSDATA=** data set.

The CLASSDATA= Option

Starting in Version 8 you can specify the name of a data set (temporary or permanent) containing the desired combinations of classification variables that PROC MEANS is to use to filter or to supplement in the input data set. This option is particularly useful if you have many classification variables, and many values of the classification variables, and you only want analyses for some of these combinations. See the PROC MEANS documentation for additional details on how the **CLASSDATA=** option is utilized.

The WAYS Statement

This new PROC MEANS statement specifies the number of ways that the procedure is to create unique combinations of the CLASS statement variables.

We can see how the **WAYS** Statement with the following example. Returning to the customer order file, suppose the marketing analyst wants to create an output data set containing all two-day analyses of the classification variables. (That is, at each unique combination of the CLASS statement variables taken two at a time.) The following PROC MEANS task creates temporary SAS data set B, which contains the desired analysis.

```
proc means noprint
data=order.orderfile2 ;class mailcode
dept_nbr segment; types segment *
status      mailcode * segment
mailcode * dept_nbr;var itmprice
itm_qty;output out=b sum = ;run;
```

The same results would obtain if the analyst used the **WAYS** statement rather than the **TYPES** statement. The following PROC MEANS task, utilizing the **TYPES** statement, requests analyses of the numeric analysis variables at all two-way combinations of the CLASS statement variables.

```
proc means noprint
data=order.orderfile2 ;class mailcode
dept_nbr segment;
WAYS 2;var itmprice itm_qty;output
out=b sum = ;run;
```

Identifying Extreme Values of Analysis Variables using the IDGROUP Option

This new OUTPUT statement option combines and extends the features of the **ID** statement, the **IDMIN** option in the PROC MEANS statement and the **MAXID** and **MINID** options in the Output statement so that you can create an output data set containing variables identifying multiple extreme values of the analysis variables,

Here is an example utilizing the IDGROUP option on the electrical utility data set shown previously.

```
proc means
data=electric.elec_v8 noprint nway;
class transformer;
var total_revenue ;
output out=c
  idgroup (max(total_revenue)
out[2] (total_revenue)=maxrev)
  idgroup (min(total_revenue)
out[2] (total_revenue)=minrev)
  sum= mean= /autoname;
```

In this example, output temporary data set C will contain the SUM and MEAN of analysis variable TOTAL_REVENUE. Using the **AUTONAME** option, discussed above, the names of these variables in the output data set will be TOTAL_REVENUE_SUM and TOTAL_REVENUE_MEAN. The **IDGROUP** options instruct PROC MEANS to also determine the two largest (MAX) and smallest (MIN) values of analysis variable TOTAL_REVENUE, at each value of the single classification variable in the CLASS statement, and output those values in to variables called MINREV_1, MINREV_2, MAXREV_1 and MAXREV_2. The **OUT[2]** argument within each **IDGROUP** option specifies the number of extreme values to output. You can use select from 1 to 100 extreme values to output.

Before using this new feature, you should carefully read the PROC MEANS documentation regarding the **MAXID** and **MINID** syntax. If you are not careful, you can create output variables with the same name! If that occurs, only the first variable with the same name will appear in the output data set. You can avoid this potential outcome by using the previously discussed **AUTONAME** option.

Additional Changes and Enhancements to PROC MEANS

There are a number of other changes and enhancements to PROC MEANS. This paper has shown you what, in my experience, I feel the most important and useful new features have been added in Version 8. You can obtain a list of all enhancements to PROC MEANS from the SAS Institute web site. The complete URL for this topic is:

http://www.sas.com/service/library/onlinedoc/v8/whatsnew/tw5508/z1335349.htm

Acknowledgements

Thanks to Robert Ray of SAS Institute's BASE Information Technology group for his insights in to PROC MEANS and many of the enhancements added to it in Version 8. Also, thanks to the many people who have attended my "Summarizing and Reporting Data Using the SAS System" seminar who have made

comments or asked questions that challenged me to learn more about PROC MEANS.

Author contact

Andrew H. Karp
President
Sierra Information Services, Inc.
19229 Sonoma Highway PMB 264
Sonoma, California 94115 USA
707 996 7380
SierraInfo@AOL.COM
www.SierraInformation.com

Copyright