Paper 80-26

Efficient Statistical Programming? - Let SAS® Do the Work

Keiko I. Powers, Ph.D., J. D. Power and Associates, Agoura Hills, CA

ABSTRACT

SAS® software is a statistical package that is flexible in both data management and statistical analyses. Even with the advanced SAS capabilities, however, applied statisticians often have to go through time-consuming programming steps to perform various analyses. In many years of using SAS software, I have learned that there often are much easier programming methods to handle the same computation tasks, in such cases as calculating different group means or proportions, performing mean substitution for missing values or obtaining predicted values using a regression model. These more efficient computation setups take advantage of SAS procedures, such as PROC STANDARD, or PROC SCORE from SAS/STAT® software. These experiences taught me that applied statisticians should always keep learning various features of SAS software to see if there are more efficient programming approaches to obtain the same results.

INTRODUCTION

For applied statisticians, efficient programming of statistical packages is a key concern for conducting research projects successfully. Because of sophisticated computer programs for statistics, such as SAS/STAT, and today's powerful computers, various statistical analyses can be performed relatively effortlessly. However, analysis routines often involve very time-consuming data management steps beforehand. These steps require tailored programs for data processing because many databases have unique features that have to be handled differently each time. Therefore, it is often the case that statisticians spend more time setting up SAS programs for data management steps than the actual statistical analysis. For this reason, good statisticians are at the same time good programmers for data management. The process of writing SAS codes to handle complicated data management steps is no easy task. It is similar to trying to solve a difficult math problem, which requires a clear understanding of the process and goal and a logical thinking for an appropriate setup. In fact, when we statisticians successfully write a program to perform a complicated data management routine, it is a very rewarding experience.

It should be noted, however, that our main goal as statisticians is to perform statistical analysis as accurately and efficiently as possible. Because good applied statisticians are often good SAS programmers, and because we get a sense of accomplishment from writing a complicated program for *data management*, we tend to apply the same programming mindset to *statistical analyses* and spend more time than necessary to perform the given task. I myself still have this tendency and have to keep reminding myself to focus on producing outcomes as efficiently as possible rather than enjoying the programming challenge. For applied statisticians, producing analysis results efficiently should be more important than writing your own programs, unless there are no other options to perform the same analyses. In other words, the simpler the codes for statistical analysis, the better.

If there are several options to perform the same task, we should try to obtain results with the simple program that requires less time. For example, in an extreme case, we would not want to write a FORTRAN type program to compute a t-test statistic, when a SAS routine, PROC TTEST, does the same thing for us. I have not witnessed a case this extreme, but I have seen programs in which several complicated steps were included to perform data analysis or statistical analysis when only a few steps are needed if programmed differently. Obviously, a complicated program would unnecessarily increase the programming time in two different ways – with the initial setup time and with the debugging or revising time.

To make my point more clear, I will list several examples that contrast two programs – one being considered as a longer and more elaborate program and the other a shorter program that uses an appropriate SAS procedure. It should be noted that these examples focus on tasks that involve some forms of data analyses, such as scoring based on a regression model. In this sense, the focus is somewhat different from the paper by Lafler (1999) in which he described how to handle data management to improve the computer processing time. The 'elaborate' programs I describe here are not created for this paper. In fact, they were written for statistical analyses of actual projects in the past. These examples range from (1) calculate means by various group (or class) variables, (2) compute proportions, (3) perform mean substitution of missing values, to (4) compute predicted scores using a regression model.

EXAMPLE 1: PROC MEANS FOR MEANS WITH DIFFERENT GROUP (CLASS) VARIABLES

In this case, the assignment is to compute means for different combinations of 'class' or group variables. For example, we need quarterly averages of vehicle price for Toyota Camry and Honda Accord but monthly averages for Ford Explorer and Jeep Grand Cherokee.

One approach to handle this problem is to have two PROC MEANS steps –

```
PROC MEANS DATA=VPRICE(WHERE = (MODEL IN
('CAMREY' 'ACCORD')));
CLASS MODEL QUARTER; VAR PRICE;
RUN;

PROC MEANS DATA=VPRICE(WHERE = (MODEL IN
('EXPLORER'  'GRAND CHEROKEE')));
CLASS MODEL MONTH; VAR PRICE;
RUN;
```

Here, as Lafler (1999) recommended, it makes more sense to use the 'CLASS' option to compute group means rather than the 'BY' option. The latter requires PROC SORT step, which could take quite some time with a personal computer. These steps definitely produce results we need, but there is another feature with PROC MEANS that is very useful in making the programming easier. We use the output option and create SAS library.

```
PROC MEANS DATA=VPRICE NOPRINT;
CLASS QUARTER MONTH MODEL; VAR PRICE;
OUTPUT OUT=PROUT MEAN=;
```

```
RUN;
```

The output file 'PROUT' gives the grand mean as well as all the group means for vehicle price, and these means are included in PROUT in the following structure (see SAS Procedures Guide under PROC MEANS for the complete descriptions):

Quarter	Month	Model	_TYPE_	_FREQ_	price
			0	a	aa
		Accord	1	b	bb
		:	:		
	1		2		
	:		:		
	1	Accord	3	c	cc
	1	Camrey	3	d	dd
	:	:	:		
1		Accord	5		
:		:	:		
1	1	Accord	7		
:	:	:	:		
4	12	G. Cherokee	7		

TYPE and _FREQ_ are SAS generated variables; _TYPE_ is an indicator variable reflecting particular class-variable combinations, and _FREQ_ shows the number of observations for a given combination. For example, if we need means by model (across different time periods), we look at the PRICE values for each model when _TYPE_ = 1. The cell 'b' in the above table gives the number of observations for model = 'Accord', and 'bb' gives the mean price for Accord (across all the months and quarters). If we need model means for each month (i.e., price by model and by month), then we look at PRICE when _TYPE_ = 3 (see the cells 'cc' and 'dd') and so on. The cell 'aa' for _TYPE_ = 0 gives the grand mean of vehicle price across all models/months/quarters (with the cell 'a' = the total number of observations). This means by setting up one PROC MEANS step with the appropriate class variables, we can obtain all the sub-group means we need by producing one output file. If we have many different combinations of class variables, the second approach is much more efficient than the first approach.

It should be noted that this 'combination' setup can get affected by missing values of the class variables for mean computation. If any of the class variables has missing values, then the sample size can be different from that of the first setup where various group means are computed with several PROC MEANS steps. Therefore, if there are many missing 'class' values, it is suggested that you create a separate category for missing cases or delete these cases in order to get the mean values consistent with your research hypotheses.

EXAMPLE 2: PROC FREQ FOR OBTAINING FREQUENCIES OR PROPORTIONS

We all know that PROC FREQ produces 1-way frequency or multi-way cross-tabulation. We also know that it computes proportions for us. However, it seems that the procedure is often under-utilized when the data processing procedure requires the frequencies or proportions to be used for further analyses.

For example, suppose we have to compute monthly market share (or proportion) of 15 minivans. The original data contain monthly sales volume of each minivan for the last 10 years. A typical 'elaborate' programming for this purpose would be:

```
DATA VAN2; SET VAN;  BY YEAR MONTH MODEL;
IF FIRST.MONTH THEN TOTAL=SALE;
ELSE                TOTAL+SALE;
IF LAST.MONTH;
KEEP YEAR MONTH TOTAL;
RUN;

DATA VAN3; MERGE VAN VAN2; BY YEAR MONTH;
MSHARE = SALE/TOTAL;
RUN;
```

MSHARE in VAN3 gives the monthly market shares. I have seen this setup many times, and it seems that we have this pre-conceptualized idea that when we are computing proportions, we have to go through the summing up procedure to get the total count first. On the other hand, if we use PROC FREQ with the output option:

```
PROC FREQ DATA=VAN NOPRINT; BY YEAR MONTH;
TABLE MODEL / OUT=MSHARE (KEEP = YEAR MONTH
MODEL PERCENT);
RUN;
```

The output file MSHARE contains the monthly market share for each minivan. The variable 'PERCENT' is a SAS generated variable that contains the market share, or proportion, values. The advantage of the second approach is not just the length of the program, but elimination of the MERGE step, the step we always have to take extra caution with.

EXAMPLE 3: PROC STANDARD FOR MEAN SUBSTITUTION

When we cannot afford to lose any observations due to missing values, a common way to handle the problem is to substitute the missing values with the corresponding means. (There are many sophisticated imputation methods – see e.g., Rubin (1987), but discussion on the various methods is beyond the scope of this paper). One programming pattern I often witnessed is the combination of PROC MEANS for mean computation, data merging, and IF statements to replace missing values with the means computed with PROC MEANS.

```
PROC MEANS DATA=MISS1 NOPRINT;
CLASS VMODEL;
VAR PRICE PROFIT;
OUTPUT OUT=MMODEL MEAN=MPRICE MPROFIT;
RUN;

DATA NOMISS; MERGE MISS1(IN=DD1) MMODEL;
BY VMODEL;
IF DD1;
IF PRICE = . THEN PRICE = MPRICE;
IF PROFIT=. THEN PROFIT = MPROFIT;
RUN;
```

Both PRICE and PROFIT in the data NOMISS now have no missing observations since the two IF statements replaced them with the mean price and mean profit. There is another approach to handle the same routine; PROC STANDARD, which is very useful in standardizing data values, is also the procedure tailored for mean replacement of missing values.

```
PROC STANDARD DATA=MISS1 OUT=NOMISS REPLACE;
BY VMODEL;
VAR PRICE PROFIT;
RUN;
```

By including the option 'REPLACE', PROC STANDARD replaces all the missing values of PRICE and PROFIT with group means (by VMODEL) for PRICE and PROFIT, respectively. Again, the second programming setup eliminates the necessity of data merging, and it also needs one less SAS work datasets – i.e., two data sets (MISS1 and NOMISS), whereas the first setup needs MISS1, MMODEL, and NOMISS.

EXAMPLE 4: PROC SCORE TO COMPUTE PREDICTED VALUES

This is the situation where we just finished developing a regression model, and now we are ready to compute predicted values of the dependent variable using the regression model. The computational procedure for this purpose is rather straightforward. First, we retain the parameter estimates from the regression model. Next we bring in the data values of the variables used for the model. The data might be the original data used for modeling or they might be hold-up data for model validation. So, for example, suppose we developed the following regression model:

SALES = C + B1*PRICE + B2*REBATE + B3*APR + E,

where C is a constant (intercept), B1, B2, and B3 are parameters (or weights) for PRICE (vehicle price), REBATE (consumer rebate), and APR (annual percentage rate for financing), respectively, and E is an error term. If we are computing predicted values of the original data used for model development, the procedure is fairly simple. The following setup will produce predicted values for SALES:

```
PROC REG DATA=SALES;
  MODEL SALES = PRICE REBATE APR;
  OUTPUT OUT=PSALES
       P=PREDSALE;
RUN;
```

The variable PREDSALE in the output data set PSALES gives the predicted values for SALES. On the other hand, if we are computing predicted values of hold-up data, we have to rely on a different procedure. As many SAS users know, PROC REG creates a SAS data set that contains all the parameter estimates (use OUTEST option). Because many statisticians try to use the fact that predicted scores can be computed by taking the linear combination of these variables, the programming setup I often see is:

```
PROC REG DATA=SALES OUTEST=ESTOUT;
  MODEL SALES = PRICE REBATE APR;
RUN;

DATA ESTOUT; SET ESTOUT;
TOMERGE=1;                  /* add dummy variable for merging */
KEEP INTERCEP PRICE REBATE APR TOMERGE;
RENAME PRICE = PPRICE REBATE = PREBATE APR =
PAPR;
RUN;

DATA  HOLDDATA; SET HOLDDATA;
TOMERGE=1;                  /* add dummy variable for merging */
RUN;

DATA SALES2; MERGE ESTOUT HOLDDATA;
BY TOMERGE;
PREDSALE = INTERCEP + PPRICE*PRICE +
PREBATE*REBATE + PAPR*APR;
DROP TOMERGE;
RUN;
```

The variable PREDSALE in SALE2 data set contains predicted values for the hold-up data HOLDDATA. I included only three explanatory variables, i.e., PRICE, REBATE, APR as an example, so the setup is still reasonable here. However, as we start including more variables in the regression model, the procedure could get tedious very quickly. For example, creating different variable names for parameter estimates of all the explanatory variables would get more and more time-consuming. For the same routine, however, we can use PROC SCORE instead, and it is obviously much more efficient:

```
PROC REG DATA=SALES OUTEST=ESTOUT;
  PREDSALE:MODEL SALES = PRICE REBATE APR;
RUN;

PROC SCORE DATA=HOLDDATA SCORE=ESTOUT
OUT=SALES2 TYPE=PARMS PREDICT;
VAR SALES PRICE REBATE APR;
RUN;
```

PROC SCORE computes predicted values for the HOLDDATA data using the parameter estimates kept in ESTOUT and writes out the predicted values under the variable name PREDSALE (reflecting the name assigned to the MODEL in PROC REG step). The output data, SALES2, contain all the variables in the original hold-up data in addition to the predicted scores. With PROC SCORE, there is no variable renaming nor data merging. PROC SCORE can also be used to compute residual values by changing the option from PREDICT to RESIDUAL (see SAS/STAT User's Guide for more details).

CONCLUSION

As these examples illustrate, a quick glance at the first-type programs may give the impression that they are longer and more sophisticated programs; however, if you compare the second-type program to the first type, you will soon realize that the second type is performing the same task more efficiently with much less programming. The second-type approach requires less time for programming and at the same time helps reduce chances of making errors by, for example, eliminating the data merging step.

It should be noted that these illustrations did not compare the computer computation time between the two setups. It is possible that these shorter programs I consider more advantageous from a statistician's perspective may not be necessarily superior with respect to the computing time. However, as computers get more powerful and faster, I believe cutting down the programming and debugging time becomes more important than reducing the computer time.

A lesson from these illustrations is clear. As applied statisticians, our goal is to handle statistical analyses as quickly and accurately as possible. It is important to keep reminding ourselves not to start writing a "FORTRAN-type" program with SAS software before studying available options first. In this sense, we should keep exposing ourselves to various features SAS software can offer. After all, we do not want to spend hours programming something that SAS already worked on and set up for us.

REFERENCES

Lafler, Kirk Paul (1999) “Efficient SAS Programming Techniques”, Proceedings of the Twenty-fifth Annual SAS Users Group International Conference, Paper 146-25.

Rubin, Donald B. (1987) “Multiple Imputation for Nonresponse in Surveys”, New York: John Wiley & Sons.

ACKNOWLEDGMENTS

I would like to express my appreciation to Tie Gao for his helpful comments, and the Marketing Science Group, Power Information Network of J. D. Power and Associates for their support in preparing this paper.

CONTACT INFORMATION

Keiko Powers, Ph.D. is a Senior Manager, Statistical Analysis, at the Power Information Network Division of J. D. Power and Associates, Agoura Hills, CA. She has been a SAS user for the last 20 years. She uses various SAS routines for large database management and for advanced statistical analyses.

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

Keiko I. Powers, Ph.D.
J. D. Power and Associates
30401 Agoura Road, Suite 200
Agoura Hills, CA 91301

Phone: 805-480-3114
Fax: 805-480-3241
E-mail: Keiko.Powers@jdpa.com
Web: www.jdpa.com