To Subset or Not to Subset

Janet E. Stuelpner, ASG, Inc., Cary, North Carolina

ABSTRACT

Sometimes you need all the data in a data set. Sometimes you don't. When testing a program, it is better to use a subset of the data than the whole population. How can you write a program without the need to remove a bunch of code after you are finished testing? Macros are the answer. Macros can be great tools when trying to subset data. By defining three small macros at the beginning of your program (%iff, %wear, %and), very few changes need to be made to change from one subset to another. Or you can just run the entire population from the data set. This paper will show you how to do it.

INTRODUCTION

Sometimes it is beneficial to define macros in separate autocall libraries. This allows the user to invoke the same macro over and over again without the need to change anything in the macro. However, there are certain times when it is necessary to change code within a macro for each submission of a job. The data can be subset easily with several macros defined in the beginning of your source code. If all the data in the database is keyed to the same variables (e.g., patients, books, policies, accounts), these macros are the perfect answer to testing code before it is put into production. They are also useful when it is necessary to produce reports and/or listings on a small group of observations. Let's see how they work.

SUBSETTING MACROS

When testing a program, especially a lengthy one, it is handy to run only a few observations through the program. This not only cuts down on the runtime, but allows sufficient data to debug the program. There are many ways to reduce the number of observations in your output data sets. Of course, one can always use the OBS option. This can cause several problems. One problem with this is that you only get the first several observations (whatever number is defined on the OBS statement) and not necessarily complete information for an individual item, whether it be patients, books, transactions or widgets. This happens primarily when your data has a vertical structure where many observations define an item. If you chose 10 observations and in a specific data set it takes 15 or 20 observations to define that item, you will miss possibly half of the information. The other problem is that you have to review the code before it is run for the final time with all of the data to remove all of the OBS statements. This can be particularly tedious if you purposely place OBS statements in your code. When removing them, you have to pay very careful attention to be sure that you do not remove the wrong statements. Macros are used to insure that each data set captures the same items and code removal is not necessary, macros.

There are three situations where the macros are needed and therefore, three macros are defined. Remember, each file used must be keyed to the same variables. The first is in a DATA step where a subsetting IF can be used. The %iff macro contains one IF statement that can be used in any DATA step. Therefore, if the test data will include only one city or state, the IF statement will reflect that case. The second situation is a PROC step. The use of the IF statement is not allowed in SAS(®) procedures. Here, a WHERE clause is used. Only one WHERE can be used in any procedure, so, only one is placed inside the macro %wear. (Note: we can use a WHERE clause in a DATA step, however, we can only use one. We can create a compound situation using a WHERE ALSO, however, this would need to hold true for all DATA steps where this is used. Because we can use many IF statements, it is safer in a DATA step to use the %iff macro.) Lastly, a situation exists where it is necessary to use a compound statement. In this case, the WHERE clause begins with the first part of the subset condition and ends with %and. This macro contains the same subsetting information as in the other two macros, but the WHERE portion of the statement is in the beginning of the statement.

The nice feature of these macros is that they can be used empty without causing a syntax error. Therefore, while testing, the macro has code to specify that certain items be used. When testing is complete, the macro can have a different subset of subjects for a small report or an interim report. When the final listings and/or tables are required, the macro is void of code and the entire database population is utilized. (Note: %and must have a minimum of a semi-colon or an error condition will occur)

Below are a couple of short examples of situations where these macros are useful. In one example, only a few patient records are run through the program to test out the code. When the program is working to the programmer's satisfaction, the patient numbers can be removed and the whole population is used. The other example shows how subsets of a student population are used for testing and then the whole population can create the final report.

EXAMPLES

1. This example shows data typically found in clinical trials. The first data set that is defined is the demography data which can include sex, race and date of birth. The next data set defines the study medication taken by each patient. Lastly, we have the adverse event data set. This includes at least one observation for each patient. If a patient has an event, the field NONE will be blank. All of the data sets are merged together and then a listing is output. All of the data sets in this example are keyed to the variables INV (investigator) and PAT (patient).

```
**Define database;
LIBNAME DB 'C:\MYDIR\PATIENT\DATA';

**Define user written macros;
%macro iff;
   IF PAT IN(1,5,23,48,55);
%mend iff;

%macro wear;
   WHERE PAT IN(1,5,23,48,55);
%mend wear;

%macro and;
   AND PAT IN(1,5,23,48,55);
%mend and;

**Begin program;
DATA DM;
   SET DB.DEMOG;
   %iff;
   AGE=(VDATE-DOB)/365.25;
   KEEP INV PAT SEX RACE AGE;
RUN;

PROC SORT  DATA=DM;
   BY INV PAT;
RUN;

PROC SORT DATA=DB.STUDYMED
              OUT=SM
              (KEEP=INV PAT DRUG);
   %wear;
   BY INV PAT;
RUN;

PROC SORT DATA=DB.AE
              OUT=AE
              (KEEP=INV PAT EVENT);
   WHERE (NONE=' ') %and;
   BY INV PAT;
RUN;

DATA ALLPATS;
   MERGE  DM(IN=D)
          SM(IN=S)
          AE(IN=A);
   BY INV PAT;
   IF A;
RUN;

PROC PRINT DATA=ALLPATS;
   TITLE1 'LISTING OF AE PATIENTS';
RUN;
```

2. This example shows data from a school. The first step reads in data from a student file. This data can include student identification, the classes taken, whether a class was completed and the grade received. These data sets are keyed to STUID (student identification) and SSN (social security number)

```
**Define database;
LIBNAME DB 'C:\MYDIR\STUDENT\DATA';

**Define user written macros;
%macro iff;
   IF STUID IN(202, 356, 422);
%mend iff;

%macro wear;
   WHERE STUID IN(202,356,422);
%mend wear;

%macro and;
   AND STUID IN(202,356,422);
%mend and;

**Begin program;
DATA CLASS;
   SET DB.CLASS;
   %iff;
   IF CLASS='MATH' THEN CREDITS=4;
   ELSE CREDITS=3;
RUN;

PROC SORT DATA=CLASS;
   BY STUID SSN;
RUN;

PROC FREQ DATA=CLASS;
   %wear;
   TABLES CLASS;
RUN;

PROC PRINT DATA=CLASS;
   WHERE DROP='NO' %and;
   TITLE1 'COURSE COMPLETIONS';
RUN;
```

CONCLUSIONS

We can see that there are some very good reasons to subset data as we are testing our source code. It can cut runtime significantly. We have a great deal less output to review. The macros defined in this paper can help to achieve these benefits from subsetting the data. Because they are defined instream and placed at the beginning of the program, as recommended, they are easy to change. As long as all of your data sets are keyed to the same variable, this solution works well.

Janet Stuelpner
ASG, Inc.
326 Old Norwalk Road
New Canaan, CT 06840

(203) 966-7520 (voice)
(203) 966-8027 (fax)
jstuelpner@worldnet.att.net