Paper 51-27

Tips for Manipulating Data

Marge Scerbo, CHPDM/UMBC

Abstract

As a beginning SAS® programmer, you could be easily overwhelmed with the sheer size of the language. It is not easy to learn the best way to complete a task, if a best way actually exists. Does one method actually work better than another does?

For example:

- When working with a large number of variables, does it really make a difference to DROP or KEEP variables? If so, is one better than the other?
- What is a DATA step statement? And what is a data set option? Are they the same?
- What is the best way to subset the data, IF or WHERE?
- When should a data set be created? Should it be temporary or permanent?
- Are PROC steps always more efficient?

This tutorial should help the beginning programmer make more informed decisions. The discussion steps through some examples of simple SAS code with accompanying information on why, when and where to use.

Introduction

SAS is a hands-on language. The more a person writes code, the better a person gets. Maybe… What may seem like a fine program may in fact be inefficient or, worse, produce inaccurate results. It takes time and a bit of compulsiveness to become a better programmer.

And – it takes constant searching of documentation for the answers. Use the SAS manuals or Online Help often. And don't ignore the vast amounts of information produced by users in conference proceedings.

This presentation steps through some ideas that may help a beginning programmer save time and headache. It can only provide a minimum number of tips, but it is a beginning.

Testing Process for the Paper

There have been many articles written about efficient programming in SAS. These papers discuss techniques to save both machine time and space and human power. Some of these papers are aimed at beginners and others at advanced programmers. Many of these papers were written for earlier releases of SAS, but SAS Version 8 was rewritten with major changes affecting efficiency.

In order to prepare for this paper, the techniques were first tested with Version 8 SAS on a SUN Unix machine. All the tests were run against an existing Version 6.12 SAS data set that was neither indexed nor compressed. This was a very large file, containing almost 3 million observations and 55 variables. The variables in the data set were a mix of numbers, formatted date variables, and character fields, none of which was longer than 11 characters. For each test, a new permanent data set was created.

The results were quite amazing to someone who has been programming in earlier versions of SAS. To paraphrase Paul Kent, Director of Base SAS Research, "In Version 8 if you can decrease the size of your data set by excluding variables or observations or both, your program will be more efficient." It's that simple.

The paper includes methods to accomplish this basic concept. The results are reported in the following sections. In addition, some pitfalls that can catch a beginning programmer are discussed and possible solutions provided.

Test as you go

Oftentimes, it is difficult to know where to start. A beginning (or experienced) programmer may be unsure how a particular piece of code works. Of course, a program can be tested against the large data set used in a project, but this may be time consuming and inefficient.

There is a better way. Using SAS either interactively in the windowing environment or in a batch program, it is easy to create a small data set and execute tests against it. For example, in order to test the difference between a simple assignment statement (adding 3 numbers together) and a SUM function, the code below could be used:

```
data test;
        input a b c;
        x = a + b + c ;
        put x = ;
         y = sum(a,b,c) ;
        put y = ;
        datalines;
        1 2 3
        2 3 4
        2 . 4
        ;
run;
```

The above code creates a new data set (**TEST**) by reading in the values of a, b, and c (**input a b c;**) from the lines located below the **datalines;** statement. Note the LOG below:

```
The SAS LOG would contain:
        x=6
        y=6
        x=9
        y=9
        x=.
        y=6
NOTE: Missing values were generated as a result
of performing an operation on missing values.
NOTE: The data set WORK.TEST has 3 observations
and 5 variables.
```

This LOG shows the difference in how an assignment statement (**x = a + b + c;**) and the SUM function (**y = sum(a,b,c);**) resolve, particularly where missing values occur in the data. It is important to understand this difference and use each appropriately in future analysis. If calculations are to include variables with possible missing values, the SUM function should be used to produce all non-missing results.

Try the technique of creating small data sets to test out new ideas and methods to accomplish a task.

Looking at the Data

In the above example, PUT statements (**put x = ;** and **put y = ;**) were used to display the variable values of x and y in the LOG. This is one way to look at data.

It is important to look at data – all the time! Simply because there are no errors in the SAS LOG does not mean that the result is correct. It is rarely a waste of time to check the output data set with each step of a process. If there are multiple DATA or PROC steps and only the final data set is studied, there may be problems and/or inconsistencies created in the intermediate steps that are not apparent.

Having said that, there are different methods to look at data. The first is the old standby PROC PRINT. It is an easy procedure to use. But, remember, without any control this procedure prints every variable, every value, and every observation in the data set. So use PROC PRINT with care:

```
proc print data =  physician (obs = 50);
          var idnum total cost1 cost2;
run;
```

In this example, only the first 50 observations (**obs = 50;**) and only 4 variables (**var idnum total cost1 cost2;**) are printed. The variables chosen allow for verification that the new variable (TOTAL) was created correctly by summing the 2 variables (COST1, COST2).

In a windowing environment, it is also possible to use either PROC FSVIEW or VIEWTABLE (VT) to look at data in SAS data sets. These methods provide a table-like display of the data. In addition, they allow various interactive mechanisms to subset the data and/or select variables to be displayed. Therefore, they are more flexible than PROC PRINT is. PROC FSVIEW or VT are easy to use, and On Line help is readily available to answer questions.

DATA Statements and Options

The DATA step is the heart of the SAS language. Without it, it would be impossible to manipulate data to meet the needs of analysis. But for a beginning SAS programmer, there may be certain concepts that are difficult to grasp.

What is a DATA statement? What is a data set option? What is the difference? Does it matter when one is used versus another?

A DATA statement can be easily shown in the code below:

```
data  physician;
          set allhealth;
          total = cost1 + cost2;
run;
```

Each of these 4 DATA step lines is a statement. A statement may include keywords(s), and each statement ends with a semicolon. DATA, SET, and RUN are SAS keywords, and each begins the statement. Although it is possible to use a keyword as a variable or data set name, there are possible unknown outcomes in their use.

A data set option is attached or placed on a specific data set and appears directly after the data set is named. These options are enclosed in parentheses. To show a simple data set option, in the following code the **OBS=** data set option is used to read the first 5 observations from the data set ALLHEALTH:

```
data  physician;
          set allhealth (obs = 5);
          total = cost1 + cost2;
run;
```

Data set options can be used on either the input and/or the output data set. In the above example, only the first 5 observations are read from the input data set and thus the output data set contains only 5 observations. If only the first 5 observations are needed for analysis or reporting, this is an easy, effective, and efficient method to use.

If selection criteria are to be imposed on the input data set and only 5 observations are needed in the PHYSICIAN output data set, then the code shown below is not useful. To place the **OBS=** option on the input data set can result in an output data set containing 0 to 5 observations, depending on which of the first 5 observations meet the subset criteria.

```
*Output data set contains ? observations;
data  physician;
          set allhealth (obs = 5);
          total = cost1 + cost2;
          if total gt 100 then output;
run;
```

To correctly output 5 observations meeting the selection criteria, the following code could be used. Observations will be read from the input data set until there are 5 records output.

```
*Output data set contains 5 observations;
data  physician;
          set allhealth;
```

```
            total = cost1 + cost2;
            if total gt 100 then do;
                        cnt + 1;
                        if cnt lt 6 then output;
                        else stop;
            end;
run;
```

Data set options can also be used in PROC steps. If no data manipulation were required, using the procedure would be much more efficient then creating an intermediate data set.

```
proc print data =  physician (obs = 5);
            vars total cost1 cost2;
run;
```

A statement exists in either a DATA or PROC step and may affect all the data sets involved in that step. A data set option is attached to a specific data set and affects only that particular data set.

Selecting Variables

Whether working with large or small data sets, it makes sense to select only those variables that are to be used in the particular analysis. During the testing process for the paper, the number of variables in the data set did make a difference in the amount of CPU time to complete the process.

KEEP and DROP are available as both statements and data set options as shown in the examples below:

```
*data step KEEP statement – affects the output data set;
data  physician;
            set allhealth;
            total = cost1 + cost2;
            keep total idnum cost1 cost2;
run;

*data set KEEP option – affects the input data set;
data  physician;
            set allhealth (keep = idnum cost1 cost2);
            total = cost1 + cost2;
run;
```

Note there is a difference between the two examples. The first DATA step ‘KEEPs’ 4 variables that are to be stored in the output data set. The second example ‘KEEPs’ only the 3 input data set variables that are needed for the process. Both resulting data sets contain the same 4 variables.

Is there a difference in these two cases? In the testing process for the paper, the first step was to create a ‘copy’ of the same large data set using a SET statement. This process took a little over 3 minutes. Adding a DATA step statement to KEEP only the variables output decreased the time to 42 seconds. The KEEP data set option took only 39 seconds. While there is minimal difference between the two methods of ‘KEEPing’ specific variables, there is a vast difference in not selecting any variables. In addition, the original data set is over 10 times larger than the new data set that contains only 5 variables.

The DROP statement/option is also available. There is no discernable difference in the processing of DROP and KEEP statement/options. Since it is more difficult to identify the remaining variables if the DROP statement is used, programmer time may be affected. The KEEP statement/option readily identifies the variables to anyone reviewing the program.

Does it matter where these statements appear (what their position in the DATA step is)? The DROP or KEEP DATA step statements can occur anywhere in the DATA step. For example:

```
*Example #1;
data  physician hospital;
            set allhealth;
            keep idnum total cost1 cost2;
            total = cost1 + cost2;
run;

*Example #2;
data  physician hospital;
            set allhealth;
            total = cost1 + cost2;
            keep idnum total cost1 cost2;
run;
```

In both examples, the output data sets, PHYSICIAN and HOSPITAL, contain the four variables named in the KEEP statement.

In order to create different output data sets, use the data set options as shown below:

```
* multiple data sets;
data  physician (keep = idnum total)
        hospital    (keep = idnum total cost1 cost2);
            set allhealth;
            total = cost1 + cost2;
run;
```

In this case, the data set PHYSICIAN contains 2 variables while the data set HOSPITAL contains 4.

Does the use of DROP/KEEP statements also affect procedures? In testing a basic procedure as shown below, there was minimal difference in the machine time between to two sets of code.

```
*Example #1 – no KEEP statement;
proc univariate data = hospital;
            var total;
            by hospital;
run;

*Example #2 –  KEEP statement;
proc univariate data = hospital (keep =  total hospital);
            var total;
            by hospital;
run;
```

Since there are many different procedures available in SAS, it is difficult to test each for efficiency. Therefore, it is good practice to select only the variables needed for the analysis as shown in example #2.

So remember, whenever possible, select the variables needed by using a KEEP or DROP statement or data set option.

Selecting Observations

Of course, there are studies where all observations in a data set are included, but often there should be some selection criteria imposed on the data. In SAS, it is possible to subset a data set in either a DATA or PROC step and with either a statement or a data set option. Two of the SAS keywords used for subset are IF and WHERE.

What are the differences between IF and WHERE? Are there easy ways to know when and how to use them?

First, it is possible to use both IF or WHERE statements in a DATA step, while the IF statement cannot be used in a PROC step. WHERE can be used as both a statement and a data set option; IF is specifically a DATA step statement.

The table below displays some of the similarities and differences in the two SAS keywords. Both provide methodology for the selection of observations meeting specific criteria, but how and when to use the IF and WHERE keywords differ:

USAGE	IF	WHERE
DATA step statement	Yes	Yes
Data set option	No	Yes
PROC Statement	No	Yes
With all Operators	No	Yes
Variables	All	Input data set only
With OBS= option	Yes	No
With FIRST. or LAST.	Yes	No

So, both IF and WHERE can be used to create subset a data set within a DATA step as shown below:

```
*IF Statement;
data surgery;
            set physician;
            if providertype = 'SURG';
            keep provider providertype cost1;
run;

*WHERE statement;
data surgery;
            set physician;
            where providertype = 'SURG';
            keep provider providertype cost1;
run;
```

In Version 6 SAS, there was a vast efficiency difference in the processing of IF and WHERE statements. WHERE statements and data set options were much more efficient than IF statements. In Version 8, this is not the case. In the tests run for this paper, the IF statement was actually faster than the WHERE statement.

The IF statement cannot be used as a data step option while WHERE can:

```
*WHERE data set option;
data surgery;
            set physician (where =
                        ( providertype = 'SURG'));
            keep provider providertype cost1;
run;
```

The IF statement cannot be used with procedures. If the required analysis can be accomplished by using a SAS procedure with the subset of a data set, the use of a WHERE statement or data set option is most likely the best choice. Unless there are complicated algorithms and/or data manipulation required, this process can be accomplished in one step:

```
*WHERE statement;
proc freq data =  physician;
            where = cost1 gt 9999;
            tables provider;
            keep provider providertype cost1;
run;

*WHERE data set option;
proc freq data =  physician (where = (cost1 gt 9999));
            tables provider;
            keep provider providertype cost1;
run;
```

Both of these statements produce the same output. In testing the two methods, there was minimal efficiency (CPU time) difference between the two. The data set option requires correct placement of the parentheses and equal sign, and more complicated selection criteria may be difficult to code. In that case, use the WHERE statement; accurate results are imperative.

SAS provides a variety of operators and functions. Many of the operators and functions can be used in both IF and WHERE statements, but there are certain ones that can be used only with WHERE. For example, the BETWEEN-AND operators are very useful but are valid only with WHERE:

```
*WHERE statement;
data surgery;
            set physician;
            where cost1 between 1000 and 9999;
            keep provider providertype cost1;
run;
```

Check the SAS Manual or OnLine help for more information.

While it appears WHERE is more useful than IF, there are situations when the WHERE statement or option cannot be used. For example, WHERE can be used only with variables that exist in the input data set. Note the code below:

```
*WHERE statement will cause an error;
data surgery;
        set physician;
        total = sum(cost1,cost2);
        where total gt 5000;
        keep provider providertype cost1 cost2 total;
run;
```

This code will cause the following error:

```
ERROR: Variable total is not on file WORK.PHYSICIAN.
```

If such selection criterion is needed, use the following code:

```
*IF statement will work;
data surgery;
        set physician;
        total = sum(cost1,cost2);
        if total gt 5000;
        keep provider providertype cost1 cost2 total;
run;
```

If only a portion of the data is to be analyzed, for example the first 1000 observations, the WHERE statement or data set option can not be used:

```
* WHERE statement will cause an error;
data surgery;
        set physician (obs = 1000);
        where providertype = ‘SURG’;
        keep provider providertype cost1;
run;
```

If a program contains a statement or option that accesses the data by observation numbers, for example OBS= or FIRST.var, a WHERE statement or option cannot be used. The LOG for the code above contains the following error:

```
ERROR: A where clause may not be used with the
FIRSTOBS or the OBS data set options.
```

Instead, use the IF statement:

```
*IF Statement selects surgeons from the first 1000
observations in the PHYSICIAN data set;
data surgery;
        set physician (obs = 1000);
        if providertype = ‘SURG’;
        keep provider providertype cost1;
run;
```

Since the efficiency differences between IF and WHERE are of little concern in SAS V8, use the one most appropriate to the situation.

Creating Data Sets

Many sites have specific rules as to the storage of data. This may be a result of an old system, large amounts of data, multiple access, etc. But assuming that there is availability of disk space, should permanent data sets always be created? One step further, should any data set be created?

There are some simple rules of thumb that may help in these decisions:

- If there is a single procedural analysis to be run on an existing data set that requires no preprocessing, in many cases it is best to use a direct reference to the data set:

```
*right way;
proc freq data =  cy00.hospital;
        tables diagnosis;
run;

*wrong way;
data hospital;
        set cy00.hospital;
        keep diagnosis;
run;
proc freq data =  hospital;
        tables diagnosis;
run;
```

- If there are many reports to be run against a data set and each run requires the same subset of observations and/or variables, it may prove efficient to create a new data set with the subset of observations and variables for the analysis. This allows all studies to be completed on a smaller data set:

```
data hospital;
        set cy00.hospital;
        keep diagnosis provider hospid cost1;
        where county = ‘AA’;
run;

proc freq data =  hospital;
        tables diagnosis;
run;
proc univariate data = hospital;
        var cost1;
run;
```

The tests for the paper showed a decrease in machine time by creating a smaller data set and running studies against it. Remember, as a rule of thumb, the smaller (narrower or shorter) the data set, the less processing time.

DATA vs PROC Steps

These two steps were created for different purposes. The SAS DATA step reads in data, creates new data sets and variables, performs calculations, and manipulates the data. SAS procedures provide a wonderful array of analytic, graphic, and reporting tools. Some procedures create new data sets and/or variables. So are there times to use one or the other? Of course, this is situation specific.

It is important to understand the differences between these two steps. In addition, remember that although there is overlap with some statements between the two steps, most statements are unique to the type of step. Moreover, each PROC is unique and must be coded to meet its specific criteria. For example, note the difference between these two basic procedures:

```
proc freq data =  cy00.hospital;
          tables diagnosis;
run;

proc print data = cy00.hospital;
          var diagnosis;
run;
```

It is possible to append one SAS data set to another using a SAS DATA step. This DATA step reads two data sets (FY00.HOSPITAL and NEWQUARTER) and concatenates the second to the first:

```
*using a SAS data set for append;
data fy00.hospital;
          set fy00.hospital newquarter;
run;
```

The same process can also be accomplished more efficiently using PROC APPEND, assuming the variable structure is the same in both data sets. This procedure reads only the data set to be appended, not the base data set:

```
*Use of PROC APPEND;
proc append base = fy00.hospital
          data = newquarter;
run;
```

In both cases, the data set (FY00.HOSPITAL) is updated by concatenating the data set NEWQUARTER to the existing data set. If is the original data set is not to be overwritten and a new data set is to be created, then the difference between the two options is not as apparent:

```
*Using SET to create a new data set from two;
data hospitalcy00;
          set fy00.hospital
              fy01.hospital;
          where servicedate between mdy(1,1,2000)
                    and mdy(12,31,2000);
run;
```

This process reads both fiscal year data sets in order to create a new calendar year data set. It is also possible to use PROC APPEND to create the same data set. In using this procedure, it is important to recognize that the BASE data set, the final data set, must be appended in place. Therefore since a new data set (HOSPITALCY00) containing both the original and the new data sets is necessary, the first step would be to create a temporary data set:

```
data hospitalcy00;
          set fy00.hospital ;
          where servicedate between mdy(1,1,2000)
                    and mdy(12,31,00);
run;

proc append base = hospitalcy00
          data = fy01.hospital;
          where servicedate between mdy(1,1,2000)
                    and mdy(12,31,00);
run;
```

This case, whether to use a procedure (PROC APPEND) or a DATA step SET statement, is dependent on the situation. If the original data set can be overwritten, then the procedure is more efficient.

Creating New Variables

The creation of a new variable or variables is a common task. It is also a trap for new programmers. The code below was written to report the frequency of the field GENDER in which the value '0' identifies male and '1' identifies female:

```
*code1;
data study1;
          set enrollment;
  *line1;
          if gender = '0' then gendername = 'Male';
  *line2;
          if gender = '1' then gendername = 'Female';
   *line3;
          else gendername = 'Other';
run;

proc freq data = study1;
          tables gendername;
run;
```

The output from this code shows:

```
GENDERNAME                              Frequency
Fema                                      193
Othe                                      146
```

Every beginning programmer has a similar experience. What happened in this example?

- By setting GENDERNAME to 'Male' in *line1*, the length of the variable GENDERNAME is 4.
- In *line2*, although the value is set to 'Female', the length of the field is unaffected so the value is actually 'Fema';
- The ELSE statement of *line3* resets all values of GENDERNAME where gender not equal to 1 to 'Othe'.

To edit the code above correctly:

```
*code2;
data study1;
```

```
   set enrollment;
  *line1;
   length gendername $6;
   if gender = '0' then gendername = 'Male';
  *line2;
      else if gender = '1' then gendername = 'Female';
      else gendername = 'Other';
run;

proc freq data = study1;
        tables gendername;
run;
```

Resulting in the following output:

```
GENDERNAME                          Frequency
Female                                  193
Male                                    123
Other                                    23
```

What makes code2 work?

- The **LENGTH** statement sets the length of a variable. It is good practice to always set the length to the longest value.
- The **IF THEN ELSE** statements create the new field by checking each value and proceeding to the next **IF THEN ELSE**.

Code2 is correct, but there is actually a more efficient method:

```
*code3;
proc format;
        value $gender
          '0'     = 'Male'
          '1'     = 'Female'
           other  = 'Other' ;
run;
proc freq data = enrollment;
        tables gender;
        format gender $gender.;
run;
```

This process does not create a new data set for the purpose of the report and is a much more efficient program. Code3 uses a PROC FORMAT to create the values desired for the report. Check the SAS Language Guide or OnLine help for more information on PROC FORMAT.

More on Creating New Variables

Most experienced programmers strive to have a 'clean' LOG at the end of a run. This not only means void of errors but also void of warnings and messages. The conversion from numeric to character variables or vice-versa can cause problems if done incorrectly. For example, the following code sets a character variable from a numeric field:

```
*sex is a numeric variable;
        sex = 1;
*gender is a character variable;
        length gender $1;
        gender = sex;
```

When this program is executed, the LOG contains no ERRORs, but the following message:

```
Note: Numeric values have been converted to character values.
```

The reverse, setting a numeric variable to a character value, as shown in:

```
*sex is a numeric field;
        sex = 1;
*now it is set to a character value;
        sex = 'M';
```

When the program is run, the LOG contains the following information:

```
NOTE: Invalid numeric data, 'M', at line 3 column 13.
        sex = . _ERROR_ = 1 _N_ = 1
```

This code not only causes an ERROR, but the value of sex is set to a missing value, which could affect your analysis.

There are many functions in SAS Base language. Two of these, PUT and INPUT, can be used in the conversion of variables and thus eliminate warnings or errors.

In the code below, a new character variable GENDER is created from an existing numeric variable SEX. A numeric FORMAT statement is used to assign the variable GENDER a width of 3 bytes. The INPUT function then converts the value of SEX to a number to be stored in the field GENDER .

```
*Create numeric variable GENDER from character
variable SEX;
data study1;
        set enrollment;
        format gender 3. ;
        gender = input(sex,3.) ;
run;
```

The conversion from a character to numeric is a bit more complicated:

```
*Create character variable GENDER from numeric field
SEX;
proc format;
        value sexfmt
                1 = 'M'
                2 = 'F' ;
run;

data study2;
        set enrollment;
        length gender $1;
        gender = put(sex,sexfmt.);
run;
```

This PROC FORMAT creates character values for numbers 1 to M and 2 to F. The DATA step includes a LENGTH statement defining GENDER as a character field with a length of 1. The PUT function is used to 'format' the values of SEX and create the values of GENDER.

Do not hesitate to use the testing method discussed on Page 1 to better understand these mechanisms.

Dash vs Double-Dash

There are shortcuts in the SAS language to identify variables as lists. But of course, it might not be easy for a beginner to find these shortcuts, much less use them properly. Dashes and double-dashes are easy ways to list variables, but the difference between the two is not readily apparent.

For example, a data set has two groups of similarly named variables:

- diagnosis1-diagnosis4
- units1-units4

The first four variables (DIAGNOSIS) are character and the next four (UNITS) are numeric. To print one of the groups, use a single dash as shown:

```
proc print data = physician;
        var units1-units4;
run;
```

Only the values of these four variables are printed, no matter the position of each variable. If the following code were submitted:

```
data physician;
        set physician;
        total = sum(of unit1-unit4);
        unit5 = sum(unit2, unit4);
run;

proc print data = physician;
        var units1--units5;
run;
```

The output would display the values of each of the UNITs variables, plus the value for each variable located between UNIT1 and UNIT5:

Obs	Unit1	Unit2	Unit3	Unit4	Total	Unit5
1	3	4	.	8	15	12

So sometimes shortcuts lead to inaccurate or unexpected results. Understanding the difference between dashes and double-dashes is important.

IN or OR

The OR operator allows multiple criteria in a selection statement. The IN Operator is quite useful in selecting observations based on a list of values. Their usage can be shown as:

```
* use of OR operator for character values;
data study1;
        set enrollment;
        if category = 'AB' or category = 'AC'
          or category = 'DE' or category = 'PE'
                then output;
run;

*comparable example of IN;
data study1;
        set enrollment;
        if category in('AB','AC','DE','PE')  then output;
run;

*use of OR for numeric values ;
data study2;
        set enrollment;
        if units = 2 or units = 4 or units = 6 or units = 8
                then output;
run;

*comparable example of IN operator;
data study2;
        set enrollment;
        if units in(2,4,6,8)  then output;
run;
```

The following code will not produce the same output data set:

```
*Example #1 - incorrect use of OR  for character values;
data study1;
        set enrollment;
        if category = 'AB' or 'AC' or  'DE' or 'PE'
                then output;
run;

*Example #2 - incorrect use of OR  for numeric values;
data study2;
        set enrollment;
        if units = 2 or 4 or 6 or 8
                then output;
run;
```

In the first example, where the OR operator is incorrectly used for character values, the resulting data set STUDY1 contains all observations where CATEGORY = 'AB'. The LOG contains lines such as:

```
NOTE: invalid numeric data, 'AC', at line 4, column 24.
NOTE: invalid numeric data, 'DE', at line 4, column 32.
NOTE: invalid numeric data, 'PE', at line 4, column 40.

ERROR: Limit set by ERRORS= options reached. Further
errors of this type will not be printed.
```

In Example #2 showing the OR operator incorrectly used with numeric values, the output data set STUDY2 contains all observations in the input data set ENROLLMENT. The value 4 is evaluated as true (not equal to 0) and therefore all observations are selected.

Although in Version 6 there were efficiency differences between IN and OR operations, those concerns no longer apply. Use the IN operator often. The examples above show the IN operator as part of an IF statement. It is also possible to use IN and WHERE together. The example below shows a WHERE statement that contains both an IN operator and a substring function:

```
*example of IN operator for numeric values;
proc freq data = cy00.hospital;
          tables provider;
          where substr(category,1,1) in('A','D','E');
run;
```

This code is an example of the power and flexibility of SAS. The result of the program is a frequency of providers whose records have a category of service beginning with A, D, or E. It is very efficient programming as it selects observations and produces output within one step.

Sorting Data Sets

It is often necessary to sort a SAS data set to correctly merge files or create a BY variable process. PROC SORT is an intense utilizer of system resources.

In testing for this paper, the following three programs were run and showed little or no difference in CPU use:

```
*Example 1 – WHERE data set option;
proc sort data = cy00.hospital
             (where = (substr(diagnosis,1,2) = '65'))
              out = hospital;
          by hospital;
          keep hospital diagnosis units cost1;
run;

*Example 2 – WHERE statement;
proc sort data = cy00.hospital  out = hospital;
          where substr(diagnosis,1,2) = '65';
          by hospital;
          keep hospital diagnosis units cost1;
run;

*Example 3– IF statement;
data hospital;
          set cy00.hospital;
          keep hospital diagnosis units cost1;
          if substr(diagnosis,1,2) = '65';
run;

proc sort data = hospital;
          by hospital;
run;
```

All three examples took similar CPU time and produce an output data set HOSPITAL with 4 variables (**keep hospital diagnosis units cost1;**) and observations meeting the subset criteria (**if substr(diagnosis,1,2) = '65';**). Clearly, if additional manipulation of the data set were required, then the code in Example #3 is the best.

If a data set has already been sorted and later needs to be unduplicated by the BY variable, there are two different methods that can be used:

```
*Example #1 – SORT with NODUPKEY option;
proc sort data = cy00.hospital out = hospital
                    nodupkey;
          by hospital;
run;

*Example #2 – IF and FIRST.variable – data set must
be presorted;
data hospital;
          set cy00.hospital;
          by hospital;
          keep hospital diagnosis units cost1;
          if first.hospital;
run;
```

Tests of these two programs in SAS Version 8 showed that the second example, using the IF and the FIRST.variable, were much more efficient than the PROC SORT with the UNDUPKEY option.

A Few Last Tidbits

Here are a few bullets with ideas that may make programming or upkeep of code easier:

- Always name your data sets. In other words, do not use the default, or last, data set. When adding new steps to a program, it is easy to suddenly find the wrong data set being analyzed.
- Always indent code, with DATA, PROC, and RUN statements in the leftmost columns and proceeding statements indented accordingly.
- Use a RUN statement to signify the end of each PROC or DATA step. When using Display Manager, if a RUN statement is missing from the final step, that step is not executed.
- Always use the manual or OnLine Help.
- Never think a question is stupid. Ask for help.

Writing good SAS code is not difficult. Remember that it takes time to learn how to do it well and the only way to get better at it is to code.

References

Scerbo, M., Dickstein, C., and Wilson, A. (2001). Health Care Data and the SAS System, Cary, NC: SAS Institute, Inc.

Contact Information

For more information contact:
Marge Scerbo
CHPDM/UMBC
1000 Hilltop Circle
Social Science Room 309
Baltimore, MD 21250
Email: scerbo@chpdm.umbc.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.