Paper TT12

Advance Tips for Manipulating Data in commonly used SAS Procedures

Raj Suligavi, HTC Global Services Inc., Bloomington, IL
Jyotheeswar N Yellanki, HTC Global Services Inc., Bloomington, IL

ABSTRACT

As a SAS programmer one is overwhelmed by the sheer size of the SAS options provided. It's not only easy to finish your task but to complete we must also think is this the only or the best efficient way to do. So as an expert programmer one always thinks of efficient methods to code and apply successfully. With these scenarios in the background I'm discussing few tips as an advanced SAS programmer that one stumbles in the job scenarios ;

- While working with large number of variables, is it really efficient to use DROP and KEEP variables? If so, which one is preferred?
- How a DATA step statement works? How set option facilitates the data step? How to use ABORT and STOP statement in data step.
- What's the best way to subset data - IF and/or Where statement? Demoed by use of some PROC statements.
- Is it possible to make PROC steps more efficient?

This paper should help beginners and intermediate programmers to make more informed decisions. This paper discussion steps through some examples of code and accompanying info to support these tips.

INTRODUCTION

SAS is a great language to learn and easy to play hands-on. As one codes and applies data and proc steps during various projects one often refers and learns from SAS Publications and online help. What may seem like a fine program may in fact be inefficient or lacks intuition, worse produce inaccurate results. One should have ample time and patience to code and become a good programmer. It certainly takes consistent efforts to search and debug various programs referring SAS documentation and SAS online help. This presentation is oriented towards discussing various examples to facilitate how SAS data manipulation works and describe few useful techniques in commonly used procedures.

There have been many papers and work done to describe efficient programming in SAS. In order to present this paper the techniques were first tested in SAS Windows in version 9 and use of SAS samples provided with SAS9 software. The results reported in this paper and some pitfalls explained would facilitate the beginners and as well intermediate SAS programmers to be aware of these traps.

DATA AND PROC STEP - BEGINNER'S TIPS

Data step is the heart of SAS program without it its' impossible to manipulate data to meet the needs of analysis. As known SAS can be used in either interactively in the windows environment or in batch mode. Any information written in log window explains how a SAS process was executed. If no errors reported doesn't mean that the program was ok and results are good. One should always be cautious and observe the data all the time. If one is executing multiple data /proc steps then one should not be satisfied considering only the last data/proc step but be watchful about problems/ inconsistencies seen in the intermediate steps too. Having understood this point there are different methods to look at data - firstly PROC PRINT - which prints every variable, every value, and every observation in the dataset. So when one needs to use this procedure use an option (OBS=abc) and VAR to define only selected variables.

```
PROC PRINT DATA=PHARMA (OBS=50);
       VAR HEIGHT WEIGHT AGE TOTAL; RUN;
```

Note in windowing environment we can also use VIEWTABLE or PROC FSVIEW to see the data in SAS data sets. Not only methods are easy to use but also provide table-like display of the data and can allow various interactive mechanisms to subset the data and/or select variables to be displayed.

It's nice to know what data set statement and options are and how they can facilitate SAS processing intelligently. Remember data set options can also be used in PROC steps. If no data manipulation required using the procedure would be more efficient then creating an intermediate data set. A statement exists in either data or proc step and may affect all the data sets involved in that step. A data set option is attached to specific data set and is active only for that particular data set. Following example explains how data set options can be used on either input and/or output data set.

```
DATA PHARMA;
       SET SURVEY (OBS=10);
       TOTAL = Weight1 + Weight2;
       If TOTAL GT 350 THEN OUTPUT;
RUN;
```

```
PROC PRINT DATA=PHARMA (OBS=5);
       VAR SURVEYID TOTAL;
RUN;
```

So when you specify the FIRSTOBS= value, SAS begins to read at that observation and continues to read until SAS reaches the observation number specified in the OBS= value. The OBS= option limits the number of observations or records that a data step statement or proc step reads from an input file. For example, with the option OBS=1, it stops after reading only 1 observation or record. The value can be a whole number or the word MAX. The word MAX represents the idea of using all available observations or records, but it actually indicates a large positive integer, such as 2147483647.

Use of KEEP/ DROP statements and data set options to select only those relevant variables needed for analysis. Its certain that by select only valid variables for processing you would save CPU time and enable quick data analysis. Emphasized here is data step keep statement affects the output data set where as data set keep option affects the input data set. Same usage is possible with respect drop statement and drop data set options. Following example illustrates it.

```
DATA PHARMA;
       SET SURVEY (OBS=10);
       TOTAL = Weight1 + Weight2;
       KEEP SURVEYID TOTAL WEIGHT1 WEIGHT2 AGE; /* Keep Statement */
RUN;
DATA PHARMA;
SET SURVEY(KEEP= SURVEYID WEIGHT1 WEIGHT2 AGE); /* Keep data set option*/
       TOTAL = Weight1 + Weight2;
RUN;
```

To facilitate sub setting data SAS has IF and WHERE statements. Both can be used in DATA step, while IF statement cannot be used in PROC step. But WHERE can be used as both as a statement and also a data set option. Table below shows the details of these two keywords.

Usage	IF	WHERE
DATA SET STATEMENT	Yes	Yes
DATA SET OPTION	No	Yes
PROC STATEMENT	No	Yes
WITH OBS= OPTION	Yes	No
WITH FIRST. Or LAST.	Yes	No

Note its proven that Where statement and data set options are more efficient than If statements. SAS provides a variety of operators and functions that can be used in both IF and WHERE statements, but certain ones like BETWEEN-AND operators are very useful and valid with where only. Its imperative that if a program contains a statement / option that accesses the data by observation numbers like OBS= or FIRST.var, a WHERE statement or Option cannot be used.

```
DATA PHARMA;
SET SURVEY (OBS=100);
       WHERE PHTYPE='SENIOR';
       KEEP SURVEYID PHTYPE AGE;
RUN;
LOG Message:
ERROR: A where clause may not be used with the FIRSTOBS or the OBS data set
options.
```

There are SAS language shortcuts to identify variables as lists. Commonly used operators are Dashes and Double Dashes to list variables. Following examples illustrate the details.

```
PROC PRINT DATA=PHARMA;
       VAR UNITS1-UNITS4; /* Only the values of 4 variables are printed */
RUN;
DATA PHARMA;
       SET SURVEY;
       TOTAL=SUM (of UNITS1-UNITS4);
       UNITS5=Sum (UNITS2, UNITS4);
RUN;
PROC PRINT DATA=PHARMA (OBS=5);
       VAR UNITS1-UNITS5; /* Displays all the values of UNITS variables, plus
       the value for each variable located between UNITS1 and UNITS5 */
RUN;
```

On similar note usage of IN and OR operators is quite useful while selecting the observations from the list of variables. But when these operators are used explicit assignment operation is must otherwise SAS errors out. Following example describes it.

```
            DATA PHARMA;
                  SET SURVEY;
                  IF PHTYPE='SENIOR' OR PHTYPE='ONMED' THEN OUTPUT;
Also can use      --- IF PHTYPE in ('SENIOR','ONMED') THEN OUTPUT;
            RUN;
(If used as PHTYPE='SENIOR' OR 'ONMED' - SAS errors out. Same situation holds good for
Numeric values to).
```

If you do not instruct it to do otherwise, SAS writes all variables and all observations from input data sets to output data sets. You can, however, control which variables and observations you want to read and write by using SAS statements, data set options, and functions. The statements and data set options that you can use are listed in the following table.

Table: Statements and Options That Control Reading and Writing

Task	Statements	Data set options	System options
Control variables	DROP	DROP=	
	KEEP	KEEP=	
	RENAME	RENAME=	
Control observations	WHERE	WHERE=	FIRSTOBS=
	sub setting IF	FIRSTOBS=	OBS=
	DELETE	OBS=	
	REMOVE		
	OUTPUT		

Use statements or data set options (such as KEEP= and DROP=) to control the variables and observations you want to write to the output data set. The WHERE statement is an exception: it controls which observations are read into the program data vector based on the value of a variable. You can use data set options (including WHERE=) on input or output data sets, depending on their function and what you want to control. You can also use SAS system options to control your data.

USE OF ABORT AND STOP:

A DATA step that reads observations from a SAS data set with a SET statement that uses the POINT= option has no way to detect the end of the input SAS data set. (This method is called direct or random access.) Such a DATA step usually requires a STOP statement. A DATA step also stops when it executes a STOP or an ABORT statement. Some system options and data set options, such as OBS=, can cause a DATA step to stop earlier than it would otherwise.

The STOP statement causes SAS to stop processing the current DATA step immediately and resume processing statements after the end of the current DATA step. The STOP statement can be used alone or in an IF-THEN statement or SELECT group.

Use STOP with any features that read SAS data sets using random access methods, such as the POINT= option in the SET statement. Because SAS does not detect an end-of-file with this access method, you must include program statements to prevent continuous processing of the DATA step.

- When you use a windowing environment or other interactive methods of operation, the ABORT statement and the STOP statement both stop processing. The ABORT statement sets the value of the automatic variable _ERROR_ to 1, but the STOP statement does not.
- In batch or noninteractive mode, the two statements also have different effects. Use the STOP statement in batch or noninteractive mode to continue processing with the next DATA or PROC step.

This example shows how to use STOP to avoid an infinite loop within a DATA step when you are using random access methods:

```
Data sample;
   do sampleobs=1 to totalobs by 10;
      set master.research point=sampleobs
                          nobs=totalobs;
      output;
   end;
   stop;
run;
```

If you specify no argument, the ABORT statement produces these results under the following methods of operation:

Batch mode and noninteractive mode

- stops processing the current DATA step and writes an error message to the SAS log. Data sets can contain an incomplete number of observations or no observations, depending on when SAS encountered the ABORT statement.
- sets the OBS= system option to 0.
- continues limited processing of the remainder of the SAS job, including executing macro statements, executing system options statements, and syntax checking of program statements.
- creates output data sets for subsequent DATA and PROC steps with no observations.

Windowing environment

- stops processing the current DATA step
- creates a data set that contains the observations that are processed before the ABORT statement is encountered
- prints a message to the log that an ABORT statement terminated the DATA step
- continues processing any DATA or PROC steps that follow the ABORT statement.

Interactive line mode

stops processing the current DATA step. Any further DATA steps or procedures execute normally.

ADVANCE DATA MANIPULATION TIPS:

USE OF PUT STATEMENT:

Use the PUT statement to write lines to the SAS log, to the SAS output window, or to an external location. If you do not execute a FILE statement before the PUT statement in the current iteration of a DATA step, SAS writes the lines to the SAS log. If you specify the PRINT option in the FILE statement, SAS writes the lines to the SAS output window. The PUT statement can write lines that contain variable values, character strings, and hexadecimal character constants. With specifications in the PUT statement, you specify what to write, where to write it, and how to format it.

Holding and Releasing Output Lines: This DATA step demonstrates how to hold and release an output line with a PUT statement:

```
data _null_;
   input idno name $ startwght 3.;
   put name @;
   if startwght ne . then
     put @15 startwght;
   else put;
   datalines;
032 David 180
049 Amelia 145
126 Monica
219 Alan 210
;
```

In this example, the trailing @ in the first PUT statement holds the current output line after the value of NAME is written if the condition is met in the IF-THEN statement, the second PUT statement writes the value of STARTWGHT and releases the current output line. If the condition is not met, the second PUT never executes. Instead, the ELSE PUT statement executes. This releases the output line and positions the output pointer at column 1 in the output buffer. The program writes the following lines to the SAS log:*

```
----+----1----+----2
David          180
Amelia         145
Monica
Alan           210
```

Example : Writing the Current Input Record to the Log

When a value for ID is less than 1000, PUT _INFILE_ executes and writes the current input record to the SAS log. The DELETE statement prevents the DATA step from writing the observation to the TEAM data set.

```
Data team;
   input id team $ score1 score2;
   if id le 1000 then
     do;
       put _infile_;
```

```
        delete;
      end;
   Datalines;
032 red 180 165
049 yellow 145 124
219 red 210 192
;

The program writes the following line to the SAS log:*
----+----1----+----2
219 red 210 192
```

INPUT STATEMENT WITH SPECIAL INFORMAT: The $VARYING*w*. informat is a special SAS informat that enables you to read a character value, which has a length that differs from record to record. General form, INPUT statement with the $VARYING*w*. informat:

INPUT *variable* **$VARYING*w*.** *length-variable*; where

- variable is a character variable
- $VARYINGw. specifies the length of the variable

length-variable specifies a numeric variable that contains the actual width of the character field in the current record.

Raw Data File Phonedat

```
>----+----10---+----20
 1802JOHNSON2123
 1803BARKER2142
 1804EDMUNDSON2325
 1805RIVERS2543
 1806MASON2646
 1807JACKSON2049
 1808LEVY2856
 1809THOMAS2222
```

It's important to understand how the $VARYING*w*. informat is different from other character informats. This informat is composed of two parts, the informat and the length variable.

INPUT *variable* **$VARYING*w*.** *length-variable*;

The second INPUT statement uses the $VARYING*w*. informat to read the values for Name from column 5 until the length is specified by namelen. Then the values for PhoneExt are read.

```
 data perm.phones;
infile phondat length=reclen;
input ID 4. @;
namelen=reclen-8;
input Name $varying10. namelen
      PhoneExt;
```

MODIFIED LIST INPUT:

A more flexible version of list input, called modified list input, includes format modifiers. The following format modifiers enable you to use list input to read nonstandard data by using SAS informats:

- The & (ampersand) format modifier enables you to read character values that contain embedded blanks with list input and to specify a character informat. SAS reads until it encounters multiple blanks.
- The : (colon) format modifier enables you to use list input but also to specify an informat after a variable name, whether character or numeric. SAS reads until it encounters a blank column.
- The ~ (tilde) format modifier enables you to read and retain single quotation marks, double quotation marks, and delimiters within character values.

The following is an example of the : and ~ format modifiers:

```
data scores;
   infile datalines dsd;
   input Name : $9. Score1-Score3 Team ~ $25. Div $;
   datalines;
Smith,12,22,46,"Green Hornets, Atlanta",AAA
Mitchel,23,19,25,"High Volts, Portland",AAA
Jones,09,17,54,"Vulcans, Las Vegas",AA
;
proc print data=scores noobs;
run;
```

Output from Example with Format Modifiers

```
   Name       Score1    Score2    Score3         Team               Div
Smith         12        22        46      "Green Hornets, Atlanta"    AAA
Mitchel       23        19        25      "High Volts, Portland"      AAA
Jones          9        17        54      "Vulcans, Las Vegas"        AA
```

USE OF WHERE AND IF EXPRESSION:

A “where” expression restricts processing on a data file to a subset of the observations. Using an index and a WHERE expression together is called "optimizing the WHERE expression". See the conditions below for WHERE conditions that can be optimized:

Table: Tasks Requiring Either WHERE Expression or Subsetting IF Statement

Task	**Method**
Make the selection in a procedure without using a preceding DATA step	WHERE expression
Take advantage of the efficiency available with an indexed data set	WHERE expression
Use one of a group of special operators, such as BETWEEN-AND, CONTAINS, IS MISSING or IS NULL, LIKE, SAME-AND, and Sounds-Like	WHERE expression
Base the selection on anything other than a variable value that already exists in a SAS data set. For example, you can select a value that is read from raw data, or a value that is calculated or assigned during the course of the DATA step	subsetting IF
Make the selection at some point during a DATA step rather than at the beginning	subsetting IF
Execute the selection conditionally	subsetting IF

When processing the WHERE expression, the SAS System decides whether to use the index or to read the data file sequentially. First, the SAS System identifies the available indexes for use with the WHERE expression, if that expression can be optimized. A composite index may be used for WHERE expression optimization only when the first key variable is a variable in the WHERE expression.

Table: WHERE Conditions That Can Be Optimized

Condition	Examples
comparison operators, which include the EQ operator; directional comparisons like less than or greater than; and the IN operator	where empnum eq 3374; where empnum < 2000; where state in ('NC','TX')';
comparison operators with NOT	where empnum ^= 3374; where x not in (5, 10);
comparison operators with the colon modifier	where lastname gt: 'Sm';
CONTAINS operator	where lastname contains 'Sm';
fully-bounded range conditions specifying both an upper and loser limit, which includes the BETWEEN-AND operator	where 1<x<10; where empnum between 500 and 1000;

pattern-matching operators LIKE and NOT LIKE	where firstname like '%Rob_%';
IS NULL or IS MISSING operator	Where name is null; where idnum is missing;
TRIM function	where trim(state) = 'Texas';
WHERE SUBSTR(*variable*, *position*, *length*)='*string*'; when the following conditions are met: *position* is equal to 1, *length* is less than or equal to the length of *variable,* and *length* is equal to the length of *string*	where substr(name,1,3)='Mac' and (city='Charleston' or city='Atlanta');

USING THE SCAN AND SUBSTR FUNCTION:

Suppose you want to produce an alphabetical list by last name, but your NAME variable contains FIRST, possibly a middle initial, and LAST name. The SCAN function makes quick work of this. Note that the LAST_NAME variable in PROC REPORT has the attribute of ORDER and NOPRINT, so that the list is in alphabetical order of last name but all that shows up is the original NAME variable in First, Middle, and Last name order.

```
DATA FIRST_LAST;
   INPUT @1  NAME  $20.
         @21 PHONE $13.;
   ***Extract the last name from NAME;
   LAST_NAME = SCAN(NAME,-1,' '); /* Scans from the right */

DATALINES;
Jeff W. Snoker      (908)782-4382
Raymond Albert      (732)235-4444
Steven J. Foster    (201)567-9876
Jose Romerez        (516)593-2377
;
  PROC REPORT DATA=FIRST_LAST NOWD;
  TITLE "Names and Phone Numbers in Alphabetical Order (by Last Name)";
  COLUMNS NAME PHONE LAST_NAME;
  DEFINE LAST_NAME / ORDER NOPRINT WIDTH=20;
  DEFINE NAME      / DISPLAY 'Name' LEFT WIDTH=20;
  DEFINE PHONE     / DISPLAY 'Phone Number' WIDTH=13 FORMAT=$13.;
  RUN;

Another example;
DATA PHARMA;
Firstvar= 'WELCOME to PHARMASUG 2005';
Finalword = SCAN (firstvar, -1);
Put Finalword=;

Results: Firstvar=WELCOME to PHARMASUG 2005 Finalword=2005
```

The following DATA step produces a SAS data set that contains only observations from data set CUSTOMER in which the value of NAME begins with Mac and the value of variable CITY is Charleston or Atlanta:

```
Data testmacs;
   set customer;
   where substr (name,1,3) = 'Mac' and
   (city='Charleston' or city='Atlanta');
Run;
```

USING THE PROPCASE FUNCTION:

The "old" way to capitalize the first letter of words was to use LOWCASE, UPCASE, and the SUBSTR function, like this:

```
DATA CAPITALIZE;
   INFORMAT FIRST LAST $30.;
   INPUT FIRST LAST;
   FIRST = LOWCASE(FIRST);
   LAST = LOWCASE(LAST);
   SUBSTR(FIRST,1,1) = UPCASE(SUBSTR(FIRST,1,1));
   SUBSTR(LAST,1,1) = UPCASE(SUBSTR(LAST,1,1));
DATALINES;
ronald cODy
THomaS eDISON
albert einstein
;
PROC PRINT DATA=CAPITALIZE NOOBS;
   TITLE "Listing of Data Set CAPITALIZE";
RUN;
```

With the PROPCASE function in SAS 9.1, it's much easier.

```
DATA PROPER;
   INPUT NAME $60.;
   NAME = PROPCASE(NAME);
DATALINES;
ronald cODy
THomaS eDISON
albert einstein
;
PROC PRINT DATA=PROPER NOOBS;
   TITLE "Listing of Data Set PROPER";
RUN;
```

COMPLEX CHARACTER COMPARISONS USING IN OPERATOR AND THE COLON (:) OPERATOR:

These two techniques are very versatile and handy during complex character comparisons. The use of colon (:) operator modifier to truncate the length of the longer value. This eliminates the need for the substr function and /or creating additional variables. Another technique the use of IN operator to accept character constants of different lengths. Following example explains some details;

```
'010'=ZIP_code compares as '010bb'='01034'
       and
'010'=: ZIP_code compares as '010'='010';
Also, data pharma; set survey;
       If surveyid in: ('010', '020', '0130', '0155');
```

Using a wildcard in variable lists can save a lot of typing and make your code much more generic (in some instances). In a variable list, you can specify all variables that begin with the same letters by specifying the letters followed by a colon. Most SAS statements that accept a variable list allow this type of variable abbreviation, also known as wildcard notation. In this example, only the variables that begin with BA are printed. Note that The colon must be at the end of the name, not embedded within. And, it cannot be used in a SUM function in place of a list of variables.

```
data ZOO ;
   input BARACUDA BEAR CAT DOG BAT ;
proc print data=zoo ;
   var BA: ;
run ;
```

DO LOOP WITH LEAVE COMMAND:

The in operator allows a list of values to be tested. In the following example the value of the test procedure should not be in that list of missing values. If indeed a missing value is encountered, the leave command causes the do loop to end.

```
DATA VERIFY;
Set Pharma;
Length test $5;
ARRAY testid(6) $ testid1-testid6;
       Do I =1 to 6;
       If testid(i) not in(' ','000', '0000', '00000') then do;
       Test= testid(i);
       Output;
       End;
```

```
       Else leave;
              End;
              Drop I;
       Run;
```

USE OF MOD OPERATOR:

Example to select a third observation from the data set use this code.

```
Data Pharma;
       Set Pharma;
              If  MOD (_N_,3)=1;
       run;
```

MANIPULATING DATA WITH PROC SQL:

This tip provides character alignment, concatenation and pattern matching while manipulating data. In explaining the examples only SELECT statement was used thought there are other number of ways to accomplish the objectives. Here are the types of operators and functions in PROC SQL: comparison operators, logical operators, arithmetic operators, character string operators, and summary functions

Character Alignment and Concatenation

Usually, character string operators and functions are used with character data. These operators enable you to control character alignment and string concatenation. The default alignment for character data is to the left. However, character columns or expressions can also be aligned to the right. Two functions in PROC SQL are available for character alignment: LEFT and RIGHT. The next example combines the concatenation operator **||** and the **TRIM** function with the **LEFT** function to left align a character expression, and inserts blank spaces and a dash (-) between two character columns to create the subset "PG-rated" from the table MOVIES.

```
       Code:   PROC SQL;
       SELECT TRIM(LEFT(title) || " - " || category) AS Concatenation_Alignment
                     FROM movies
                  WHERE rating = "PG";
                  QUIT;
       Results: Concatenation_Alignment
```

```
         Casablanca - Drama
         Jaws - Action Adventure
         Poltergeist - Horror
         Rocky - Action Adventure
         Star Wars - Action Sci-Fi
       The Hunt for Red October - Action Adventure
```

Phonetic Matching (Sounds-Like Operator, =*):

A technique for finding names that sound alike or names that have spelling variations is available in PROC SQL. Although not technically a function, the sounds-like operator **=*** searches and selects character data based on two expressions: the search value and the matched value. Anyone that has looked for a last name in a telephone directory is quickly reminded of the possible phonetic variations. To show how the sounds-like operator works, we will search on the column TITLE in the MOVIES table, using the string "Rucky", for all phonetic variations that are related to the movie title "Rocky".

```
          PROC SQL;
            SELECT title, rating, category
              FROM movies
                WHERE title =* "Rucky";
          QUIT;
The results:
   Title                                                                Rating     Category
_____________________________________________________________
   Rocky                           PG      Action Adventure
```

Finding Patterns in a String (Pattern Matching, % and _):

Constructing specific search patterns in string expressions is a simple process with the LIKE predicate. The percent sign (%) acts as a wildcard character that represents multiple characters, including any combination of uppercase or lowercase characters. Combining the LIKE predicate with the percent sign (%) permits case-sensitive searches. It's a popular technique used by savvy SQL programmers to find patterns in their data. The next example finds patterns in the CATEGORY column in the MOVIE table that contains the uppercase character **D** in the first position, followed by any number of characters.

```
       PROC SQL;
            SELECT title, rating, category
```

```
            FROM movies
              WHERE category LIKE 'D%';
      QUIT;
```

The results:

Title	Rating	Category
Casablanca	PG	Drama
Forrest Gump	PG-13	Drama
Michael	PG-13	Drama
Dressed to Kill	R	Drama Mysteries
Ghost	PG-13	Drama Romance
Titanic	PG-13	Drama Romance
Silence of the Lambs	R	Drama Suspense

SOME TIPS ON PROC SORT:

It's observed by SAS users that the SORT procedure is one of the easiest procedures to use and one of the most abused. One important strategy frequently used when processing, and in particular sorting, large data sets is to first spend time understanding what your objective is with the sort, and then spend a little more time planning how to best achieve that objective. The trick to performing a successful sort is to first understand what your objective is, and then find out how to achieve that objective. Here are a few guidelines to follow whenever you find yourself sorting a large data set.

- Determine if the data in a large data set is already in sorted order. If it is, then a sort may not be necessary.
- Determine whether the data is in a particular sort order by specifying the BY statement in a DATA or PROC step (other than the SORT procedure). If the data isn't in the desired sort order specified in the BY statement, a runtime error is produced.
- Specify the NOTSORTED option when the observations with the same BY value are grouped, but the observations within the group are in unsorted order.
- Determine whether any installation or resource constraints (e.g., shortage of space, etc.) exist that could prevent a data set from being sorted.
- Understand the requirements of the DATA or PROC step that you plan to use as he input data set. Many procedures, including all of the summary procedures, don't need data in sorted order because they produce results in sorted order after summaries are produced anyway.
- Consider using a tag sort if a sort is needed. When a data set is too large to fit in memory, then tag sorting may be more efficient because it requires less temporary storage space and memory. Specify the TAGSORT option in the SORT procedure statement.
- Consider only sorting the observations and variables that are needed to satisfy the conditions of the analysis.

SOME TIPS ON PROC REPORT:

It’s very difficult to force PROC REPORT to print all of the variables on the same page if there is not enough room to do so. It may help to reduce the spacing between the columns, reduce the width of the columns and/or increase the line size. However, the reduced space still has to be large enough to print out all of the variables on one page or PROC REPORT will continue printing the remaining variables on the next physical page.

- To reduce the space between variables, set the SPACING= option to either 0 or 1 on the PROC REPORT statement. This reduces the space between the variables from the default value of 2.
- To control spacing for a particular column, use the SPACING= option on the DEFINE statement within PROC REPORT.
- You can change the report column width using the COLWIDTH= option on the PROC REPORT statement. Also, the WIDTH= option on the DEFINE statement controls the width of a particular variable.
- Finally, the line size can be changed on the PROC REPORT statement (via the LS= option) or on an OPTIONS statement before the PROC REPORT statement.

PROC REPORT in Base SAS software is an extremely flexible tool that combines formatting, summarization and analysis features with report generation. Because it is powerful and easy -to use, it has become a favorite with many SAS software users.

CONCLUSION:

Evaluating your raw data and choosing the most appropriate data style to manipulate and facilitate your analysis is a very important task. You have probably should not only understand how and what data characteristics are but also actively think what and where data need to be manipulated to achieve the required results. Hence as stated data manipulation is an evolving skill expertise as one observes, understands and interprets data knowledge.

REFERENCES:

1. Simplifying complex character comparisons by using the IN operator and Colon (:) operator modifier by Paul Grant, SAS Institute Inc.
2. Tips for manipulating data by Marge Scerbo - SUGI 27 Paper
3. Quicktip: Avoiding Problems Related to Sorting by Kirk Paul Lafler, Software Intelligence Corporation.
4. Power Indexing: A Guide to Using Indexes Effectively in Nashville Releases
   By Diane Olson, SAS Institute Inc., Cary, NC.
5. "SAS Today! A Year of Terrific Tips" by Helen Carey and Ginger Carey
6. "In the Know… SAS Tips & Techniques From Around the World" by Phil Mason.
7. Array tutorial (2) - beginning intermediate - by Marge Scerbo, Sugi29 Paper.
8. SAS Institute Inc., SAS OnlineDoc9 ® http://v9doc.sas.com/sasdoc/
9. SAS Online Proceedings ® http://support.sas.com/usergroups/sugi/proceedings/index.html
10. Cody's Data Cleaning Techniques using SAS software by Ron Cody.
11. The Little SAS Book - A Primer by L D Delwiche and S J Slaughter - Third Edition

CONTACT INFORMATION

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

Raj Suligavi
Phone: 309-663-0858
Email: bsuliga@yahoo.com
&
Jyotheeswara Naidu Yellanki
Phone: 309-662-5163
Email: yjnaidu@hotmail.com