Tips and Tricks on Rearranging and Manipulating Data and Datasets For Efficient Data Processing and Output Generation

Donald D. M. Tong, The Upjohn Company
Thomas E. Schneider, The Upjohn Company
Pan-Yu Lai, The Upjohn Company

Efficient and effective data processing or output generation is crucial in processing large data sets with many variables and observations. This means minimal typing, less data manipulation and programming. The SAS® language provides many such tools, for example, PROC PRINT, PROC MEANS, etc., without the use of a VAR statement or with the use of implicit VAR statements X5 - X21 or SMALL_S - - LARGE_P; the BY statement; MERGE, SET, and UPDATE for multiple data sets. The goal of this paper (also session) is to provide additional tips, tricks and techniques to fully utilize the nice SAS® features.

The techniques discussed cover four areas:

1. Rearrange the default order of variables,
2. Preserve the order of observations after sorting,
3. Have multiple observations available for FSEDIT,
4. Produce cross product, expansion, reduction and subset of observations for multiple data sets beyond MERGE, SET and UPDATE statements.

1. Rearrange the Default Order of Variables.

Usage: In order to process variables in blocks with implicit variable name specification, i.e., X5 - X21 or SMALL_S - - LARGE_P, the variables need to be arranged in a desired order. Once the arrangement is established, data can be efficiently processed by using DROP, KEEP and VAR, e.g., (KEEP = SMALL_S - - LARGE_P) in DATA step, or VAR SMALL_S - - LARGE_P in PROC step. PROC PRINT and MEANS pick up the default order of variables and process them without any VAR statement. (Save typing and error for repeated processing).

Example: A set of CARSALE data is given below:

DATA CARSALE:

CITY	YEAR	SMALL_S	MEDIUM_S	LARGE_S	MINDEX	SMALL_P	...	INRATING	TOTAL_S	TOTAL_P
Phi	89	673	942	764	1			D	2379	65
Mia	87	0	51	24	2	.		O	81	5
.	.	.	.	.	.			.	.	.
.	.	.	.	.	.			.	.	.
.	.	.	.	.	.			.	.	.
Sea	89	613	429	604	4			K	1646	69

↑ with missing data — ↑ formatted — ↑ character

We want to group the sales and profit of the same size cars together.

Trick #1: Use RETAIN before SET will move variables to the left (beginning) of a data set.

```
DATA CARSALE1;
RETAIN CITY YEAR SMALL_S SMALL_P MEDIUM_S
MEDIUM_P ..;
SET CARSALE;
```

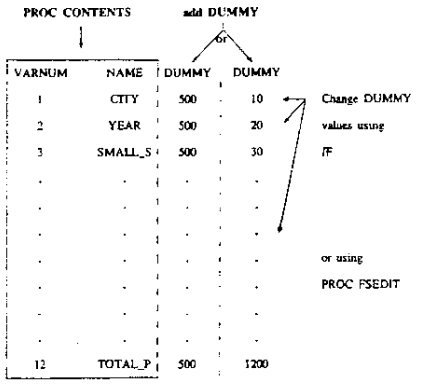
Advantage/Disadvantage: RETAIN is simple to use for a few variables. Unfortunately, variables preceding the ones to be arranged need to be typed. It is also typing intensive for inserting boundary variables, e.g., _1 ··· _2 ··· _3 ···, etc. for processing data in blocks. (RETAIN SMALL_S - - LARGE_P; does not work).

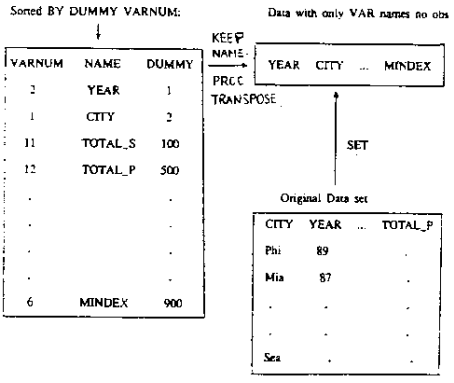
Trick #2: (Imperfect technique with two PROC TRANSPOSE discussed in the workshop only).

Trick #3: Use PROCs CONTENTS, SORT, and TRANSPOSE, and SET. They are given in these steps:

1. Use PROC CONTENTS to obtain NAME (variable names) and VARNUM (order of variables).
2. Create a DUMMY variable, e.g., DUMMY = 500; (for grouping), or DUMMY = VARNUM*10; (for inserting variables).
3. Change the values of DUMMY by using IF statement or FSEDIT, e.g.,

   IF 5.5<VARNUM<11.5 THEN DUMMY = 21;.
4. Sort data (variable names) BY DUMMY VARNUM; (to obtain the desired order).
5. PROC TRANSPOSE only the NAME into a new data set.
6. SET the latter empty data set continuing nothing but variable names with the original data set.





Example: Plot sales of each city in the order of descending sales volume of year 89.

```
PROC SORT;
  BY DESCENDING TOTAL_S;

PROC GPLOT;
  PLOT TOTAL_S*CITY;

PROC GCHART;
  VBAR CITY/SUMVAR=TOTAL_S TYPE=SUM;
```

The result is that none of the output graphics are plotted against cities with descending sales volume.

TRICK #4: Format a numerical dummy variable CITY_ with the values (i.e., cities) of variable CITY.

Steps:

1. Create a dummy variable CITY_=_N_;
2. Format CITY_ with cities of CITY using MACRO MAKEFMT.
3. PLOT TOTAL_S against CITY_

Example: SAS® Statement

```
PROC SORT DATA=CARSALE7;
  SET CARSALE7;
  CITY_=_N_;
  LABEL CITY_='CITY';
```

```
************************************************;
*              MACRO MAKEFMT PROGRAM
************************************************;

%MACRO MAKEFMT(FMTNAME, FROMDATA=,
VALUE=, LABEL=);
DATA_NULL_;
SET &FROMDATA END =LASTREC;
CI=LEFT(PUT(_N_,5.));
RIGHT='"'||&VALUE||'"='"'||&LABEL||'"';
CALL SYMPUT('FMT'||CI,RIGHT);
IF LASTREC THEN CALL
SYMPUT('ENTRIES',LEFT(PUT(_N_,5.)));
RUN;
PROC FORMAT;
VALUE &FMTNAME
%DO I=1 %TO &ENTRIES;
&&FMT&I
%END;
%MEND MAKEFMT;
RUN;
```

```
*************************************************;
*          CALLING MACRO MAKEFMT PROGRAM
*************************************************;

%MAKEFMT(CITY_, FROMDATA=CARSALE7,
VALUE=CITY_, LABEL=CITY);
RUN;

PROC GPLOT DATA=CARSALE7;
   PLOT TOTAL_S*CITY_;

PROC GCHART DATA=CARSALE7;
   VBAR CITY/SUMVAR=TOTAL_S TYPE=SUM;
```

Advantage/Disadvantage: This is the simplest way to get what we want. The use of a MACRO is unavoidable. One drawback is that the formatted label is not passed along. This will be shown in the next trick.

TRICK #5: Passing formatted labels to the dummy variable

Steps:

1. Create a dummy variable _CITY=_N_;
2. Create a variable LBL_CITY bolding labels of CITY (explain in Trick #6)
3. Format _CITY with LBL_CITY using MACRO MAKEFMT

TRICK #6: Create LBL_CITY holding labels of CITY

Method 1: LBL_CITY=PUT(CITY, $CITY.);

(Find the format $CITY. through PROC CONTENTS)

Method 2: (This is presented as another illustration of the use of dummy variable techniques)

- Without knowing the format name
- Get contents from PRINTTO file without counting position
  1. Use PROC PRINTTO to put label values of city to a file.
  2. Access the label values through a dummy variable created.

     IF DUMXZ_='ZXZZX' THEN DO;
     (To avoid counting lines and columns)

Example: SAS® Statements of Method 2:

```
DATA; SET CARSALE8T (KEEP=_CITY);
RETAIN DUMZX_;
LENGTH DUMZX_ $5.;
DUMXZ_='ZXZZX';

FILENAME LSTOUT 'C:\SUGI15\OUTPUT.LST'
TITLE1;
PROC PRINTTO PRINT=LSTOUT NEW;
PROC PRINT NOOBS;
PROC PRINTTO;

DATA;
FILE PRINT NOPRINT;
INFILE LSTOUT;
INPUT @1 NAME $@;
PUT _INFILE_;
IF NAME='ZXZZX' THEN DO;
  INPUT LBL_CITY $20. _CITY;
  OUTPUT;
  END;
```

Advantage/Disadvantage: At this moment, this is the simplest technique that we know of.

TRICK #7: Presenting Multiple Observations on One Screen using PROC TRANSPOSE and PROC FSEDIT.

It is often the case that it would be more convenient to present multiple observations on an FSEDIT screen for the purpose of editing data. FSEDIT is limited in that it will only present one observation of a SAS data set at a time.

In this trick, for demonstration and simplicity's sake, a data set with only one variable is used. The basic algorithm is simple:

1. Transpose the data set by a newly created variable (SCREEN) that identifies how many observations (N) to place on a screen. SCREEN would contain the value 1 for the first observations, 2 for the next N observations, and so on.
2. Do a PROC FSEDIT on this transposed data set and edit the values as normal.
3. Transpose the data set back to its original form. Because the data set has been transposed, it will usually be the case that there will be some newly created observations at the end of the data set with missing values. The algorithm presented here simply deletes all observations with missing values.

Example: SAS Statements

```
Data EX4B;
  Keep name screen newname;
  Length newname $8;
  Do p=1 to n;
    Set ex4a point=p nobs=n;
    perscrn=12;
    screen=int((p-1)/perscrn);
    m=mod((p+perscrn-1),perscrn)+1;
    newname='Name'||left(m);
    Output;
    end;
  stop;
run;

Proc Transpose Data=EX4B
            Out=EX4C(drop=_name_ screen);
  Id newname;
  By screen;
  Var name;

Proc Fsedit Data=EX4C;
  Run;

Data EX4C;
  Set EX4C;
  oldname='NAME';
  oldobs=_n_;

Proc Transpose Data=EX4C
            Out=EX4D(drop=_name_ oldobs);
  Id oldname;
  By oldobs;
  Var name1-name12;

Data EX4A;
  Set ex4d;
  if name=' ' then delete;
  Run;
```

NOTES:

The preceding example only works for a data set with one variable. The advantage of having only one variable is that the algorithm to transform tbe data set is simple and that the default presentation of the data on the FSEDIT screen is 'pretty'. It is obvious, however, that we don't always have the luxury of working with a data set with only one variable. Similar algorithms could be developed to extend the idea using data sets with more than one variable.

This algorithm also deletes all missing values in order to remove the missing observations that are created with the second transpose. This feature may not be desirable for your particular application.

TRICK #8: Using the XMERGE Macro to Subset, Expand, and Merge Data Sets.

The XMERGE Macro will take as input any number of data sets and perform a cross product with those data sets either across all observations or within a specified BY group. The statement to invoke the XMERGE macro is as follows:

```
%XMERGE(INDATA=dataset1 dataset2 ...,
   OUTDATA=dataout,
   BYVARS=byvar1 byvar2 ...,
   SORT=YES|NO,
   INSET=insetstatement);
```

The **INDATA** parameter specifies the data sets that are to be cross merged together. At least two data set must be specified. Due to the explosive nature of cross products, care must be taken

The **OUTDATA** parameter specifies the data set that will contain the cross merged data set. The **BYVARS** parameter specifies the variables by which the data sets are cross merged together. In order to cross merge using BY variables, the input data sets must be sorted. The SORT option can be used to tell the macros to sort the data sets internally before doing the cross merge. If the BYVARS parameter is not included in the macro call, the data sets will be cross merged across all observations.

The **SORT** parameter specifies whether or not to sort the input data sets on the BY variables before they are cross merged. If SORT=YES is specified then the input data sets will not be sorted before they are cross merged. If SORT=NO is specified, then the data sets will be sorted internally.

The **INSET** parameter is a logical and/or statement that specifies how to subset the data set based upon whether or not the values of variables are included across all data sets. An example of an INSET parameter would be

This macro was also presented at SUGI13 in Dallas, Texas. For a more in-depth presentation of this macro and its application refer to the SUGI13 Proceedings.

Conclusion: We find processing large data sets is a challenging and exciting area of work. Every tip or trick helps in speeding up data processing. Working with PC SAS®, we found it is a powerful and flexible tool to examine what SAS can and cannot do. It is very effective in testing tricks and techniques for solving problems. In addition, it also can process large data sets effectively. We gave you some of our tricks and experience and hope they will help your work. Finally, we would like to hear your tips and tricks.

SUGI SAS® Users Group International
Proceedings of the Fifteenth Annual Conference