An Introduction to SAS-Lecture 3

Konstantinos Fokianos
University of Cyprus

Reading and Combining Data Sets

Suppose that you have the following data set

```
1 68 144 M 23
2 78 202 M 34
3 62 99 F 37
4 61 101 F 45
5 73 135 M 24
6 60 104 F 34
```

where the variables (by column) are `ID`, `HEIGHT`, `WEIGHT`, `GENDER` and `AGE`. We want to create a new data set where we keep all the observations with `SEX=F`.

Reading and Combining Data Sets

This is accomplished by the following program:

```
DATA TOY;
     INFILE "TOYEXAMPLEDATA.TXT";
INPUT  ID HEIGHT WEIGHT GENDER $ AGE;
DATA FEMALE;
     SET TOY;
 IF GENDER EQ 'F';
PROC PRINT DATA=FEMALE;
RUN;
```

The process of accessing the existing data set for subsetting a part of it is run through the statement `SET`. The `SET` statement does the same work as the `INPUT` statement; the only difference is that it reads data from an existening SAS data set.

Reading and Combining Data Sets

The `IF` statement is a fast way for executing the equivalent statement `IF NOT` *condition* `THEN DELETE`. The resulting data set (FEMALE) contains only those observations where `GENDER=F`.
An alternative method to carry out the same manipulation is the following:

```
DATA FEMALE2;
     SET TOY;
 WHERE GENDER EQ 'F';
PROC PRINT DATA=FEMALE2;
RUN;
```

Reading and Combining Data Sets

There are some differences between the IF and WHERE statement.

- The WHERE statement is more efficient than the IF statement.
- The WHERE statement can be included in SAS procedures. This saves a lot of time when we want to analyze a subset of data without creating a new data set.

Reading and Combining Data Sets

WHERE statement operators:

- BETWEEN-AND: Selects observations which fall (inclusively) within a specified range.

  ```
  WHERE AGE BETWEEN  20 AND 40
  ```

- CONTAINS or ?: Used for character variables to select records that include or contain the specified string

  ```
  WHERE NAME CONTAINS 'Mc';
  WHERE NAME ? 'Mc';
  ```

- Some other operators used with the WHERE statement are IS MISSING or IS NULL (selects observations whose values are missing), LIKE and =*.

KEEP and DROP statements

We will now several modifications of the above programs. First, suppose that we only want to keep the variables ID and HEIGHT in our data sets. This is accomplished by the KEEP statement as shown below:

```
DATA TOY;
     INFILE "TOYEXAMPLEDATA.TXT";
INPUT  ID HEIGHT WEIGHT GENDER $ AGE;


DATA FEMALE2;
     SET TOY;
 WHERE GENDER EQ 'F';
 KEEP ID HEIGHT;


PROC PRINT DATA=FEMALE2;
RUN;
```

KEEP and DROP statements

A KEEP statement placed anywhere in the DATA step causes the variables which are listed to be kept in the newly created SAS dataset. You can also use the DROP statement if you want to keep most of the variables in the data.

```
DROP WEIGHT GENDER  AGE;
```

yields the same results as in the previous program.
Give priority to the KEEP statement!

KEEP and DROP statements

An alternative to KEEP or DROP statement is a KEEP= or DROP= data set options. Below is an example:

```
DATA FEMALE3;
     SET TOY (KEEP=ID HEIGHT GENDER);
 WHERE GENDER EQ 'F';
 DROP GENDER;
PROC PRINT DATA=FEMALE3;
RUN;
```

KEEP and DROP statements

- The KEEP= data set option tells SAS to read the variables ID HEIGHT GENDER. Using this option yields to more efficient programming.
- The variable GENDER has to be included in the KEEP= statement. Otherwise, it is impossible to execute the WHERE statement since the variable GENDER will not exist.

SET statement

Suppose that we have another data set which is related to the previous one.

```
7   65 150 M 25
8   73 198 M 32
9   67 105 F 39
10  59 107 F 43
11  77 129 M 26
```

We want to add observations from the one dataset to the other. This is a situation which frequently occurs in practice. For example, you have data from different years and you want to combine them in a single data set.

SET statement

```
DATA TOY;
     INFILE "TOYEXAMPLEDATA.TXT";
INPUT  ID HEIGHT WEIGHT GENDER $ AGE;


DATA TOY2;
     iNFILE "TOYEXAMPLEDATA2.TXT";
INPUT  ID HEIGHT WEIGHT GENDER $ AGE;

DATA COMBINED;
     SET TOY TOY2;

PROC PRINT DATA=COMBINED;
RUN;
```

SET statement

- The effect of including multiple data sets in the `SET` statement is to combine the data sets in the order which they are listed.
- If the two data sets do not contain identical variables, then the results of the `SET` statement is still to create a new data set with missing values for the variables that exist in any of the combined data sets.
- In a dew words, the `SET` statement stacks one data set underneath the other.

MERGE statement

Suppose now that we want to combine two data sets by adding the variables of one to the variables of the other. Usually both collections will have the same set of observations, or partially the same. This procedure is like placing the one data set next to the other. Suppose that we have data as follows:

```
DATA SET LEFT                DATA SET RIGHT

ID WEIGHT   HEIGHT           GENDER  RACE
1  68       155                 M     B
2  62       102                 F     W
3  72       220                 M     W
```

MERGE statement

You can combine the variables from these two data sets into a single data set by using the `MERGE` statement.

```
DATA TOY;
     INFILE "MERGEDATA1.TXT";
INPUT  ID HEIGHT WEIGHT;
DATA TOY2;
     INFILE "MERGEDATA2.TXT";
INPUT  GENDER $ RACE $;
DATA MERGED;
     MERGE TOY TOY2;

PROC PRINT DATA=MERGED;
RUN;
```

The above code merges the variables from the LEFT with those data from the RIGHT, one observation at a time, in the order that the observations are listed in the files.

MERGE statement

It is much better to use a `BY` statement together with the `MERGE` statement for correct matching of data sets. Suppose that we have the following data:

```
DATA SET LEFT                DATA SET RIGHT

ID  GENDER   STATE           ID    DEPT       SALARY
1    M       NY              1    PARTS       21000
5    M       NY              2    SALES       45000
2    F       NJ              3    SALES       20000
3    F       NJ              5    SALES       35000
```

Notice that both datasets contain an ID variable which is going to be used for matching.

MERGE statement

```
DATA TOY3;
     INFILE "MERGEDATA3.TXT";
INPUT  ID GENDER $  STATE $;


DATA TOY4;
     INFILE "MERGEDATA4.TXT";
INPUT  ID  DEPT $ SALARY;

PROC SORT DATA=TOY3;
     BY ID;

PROC SORT DATA=TOY4;
      BY ID;

DATA NEWMERGE;
     MERGE TOY3 TOY4;
 BY ID;

PROC PRINT DATA=NEWMERGE;
     TITLE 'Match-Merged Data';
RUN;
```

MERGE statement

- You use the `SORT` procedure to sort both data sets by ID. (Second `SORT` procedure previously reported is useless).
- If you use the `SORT` procedure to an already sorted set, then nothing really happens (see SAS log).
- The `BY` statement tells SAS to match-merge records from both data sets on the matching variable (ID).
- If there exists an ID in one data set but not in the other, then still merging takes place but the resulting data set has a missing value.

MERGE statement

Suppose that you have instead the following LEFT data set:

```
1     M        NY
5     M        NY
2     F        NJ
3     F        NJ
4     M        NY
```

As it was mentioned before, the resulting merged data set will contain the employe with ID=4 but there will be missing values for PARTS and SALARY variables. Situations like this is rather the rule than the exception in real applications.

MERGE statement

```
DATA TOY3;
     INFILE "MERGEDATA5.TXT";
INPUT  ID GENDER $  STATE $;

DATA TOY4;
     INFILE "MERGEDATA4.TXT";
INPUT  ID  DEPT $ SALARY;

PROC SORT DATA=TOY3;
     BY ID;

PROC SORT DATA=TOY4;
      BY ID;

DATA NEWMERGE;
     MERGE TOY3 TOY4 (IN=EMP);
 BY ID;
 IF EMP=1;

PROC PRINT DATA=NEWMERGE;
     TITLE 'Match-Merged Data';
RUN;
```

MERGE statement

- When you perform a merge statement, SAS checks whether if an observation is being contributed from each data set listed in the `MERGE` statement. In this case, `ID=2` exists in both data sets but `ID=4` exists only in one of them.
- The statement `IN=EMP`, in the previous program, creates a logical variable with values 1 (TRUE) and 0 (FALSE). As each observation is built and if `TOY4` has data to contribute then `EMP=1`. These `IN` variables are temporary and specify, in this example, that `EMP` is 1 except `ID=4`. Then use the `IF` statement to keep the observations with `EMP=1`.
- If you want to select observations that are in **both** data sets, then you can use

```
DATA NEWMERGE;
     MERGE TOY3 (IN=DUMMY) TOY4 (IN=EMP);
 BY ID;
 IF DUMMY=1 AND  EMP=1;
```

Some Mathematical Functions

Suppose that we have some values of variable, say X. We want to compute the following transformations

$$\begin{aligned} Y_1 &= \log X, \\ Y_2 &= \sqrt{X}, \\ Y_3 &= \arcsin X. \end{aligned}$$

You can do this by using the following simple function:

```
DATA TRANSFRM;
        SET TEST;
        Y1 = LOG (X);
        Y2 = ARSIN (SQRT(X));
    RUN;
```

Some Mathematical Functions

Choosing every *m*th observation from a SAS data set:

```
DATA THIRD;
       SET OLD;
       IF MOD (_N_,3) = 1;
    RUN;
```

The function modulus is described next. A number n modulo m is the remainder when n is divided by m. For instance 5 mod 3 =2. And 19 mod 3 =1, and so on. The above program selects every third observation from a data set, using the function modulus. The variable_N_ is automatically created and it simply counts the number of observations in a data set. So the function `MOD (_N_,3) = 1` tells SAS to to choose observations number 1,4,7,10,....

Some Mathematical Functions

Rounding and truncating numbers
To round a variable you can use the function `ROUND`. So you can give a statement

```
Y=ROUND(X,20)
```

and this tells SAS to round X to the nearest 20 and create a new variable, called Y.
To truncate a variable (drop off the fractional part), you should use the function `INT`. It works like

```
Y=INT(X)
```

with obvious notation.

Some Mathematical Functions

Suppose that each subject in some study answers 50 questions in a psychological test. Each question is scored on a 1 to 5 scale. To ensure that an accurate assessment is made, suppose that the mean score per subject is computed only if 40 or more questions were answered. The following program computes the mean of 50 questions (`ITEM1--ITEM50`) but only *if 40 of them are non missing*.

```
DATA NEWTEST;
        SET OLDTEST;
        IF N (OF ITEM1-ITEM50) GE 40 THEN
           SCORE = MEAN (OF ITEM1-ITEM50);
     RUN;
```

Some Mathematical Functions

- The function `N` calculates the number of non missing items.
- When a SAS statement tries to do simple operations on missing values, then the results is always missing!
- Some other useful functions are
  - `NMISS`: returns the number of variables with missing values.
  - `SUM`: returns the sum of non missing values.