Controlling Output Contents in DATA Step

STAT2005

Chapter 11

Introduction

- Sometimes we may want to retrieve partial information from an existing large data set and output to a new data set.
- We may want to drop out some variables and/or some records according to some criteria.
- This process is known as subsetting data.
- In this chapter, we shall illustrate how we can achieve this task.

OUTPUT and DELETE Statements

- OUTPUT and DELETE statements control which rows (**observations**) we want to store in a data set. Simple forms of OUTPUT and DELETE statements are

```
OUTPUT;
DELETE;
```

- The OUTPUT statement asks SAS to send the values of the current record to the output data set specified in the DATA statement.
- By default, SAS DATA step has an implicit OUTPUT statement at the end of the DATA step to write the current observation to the SAS data set. It explains why we do not need to include an OUTPUT statement in the programs from the preceding chapters.

For example, the program in Example 5 of Chapter 10 is in fact identical to the following program.

```
DATA ;
INPUT C1 - C3;
A1 = LOG(C1) - 1; A2 = MEAN(OF C1 - C3);
C3 = MIN(A2, C3+2);
OUTPUT; /* explicit OUTPUT */
CARDS;
. 64 56
3 30 .
4 22 45
RUN;
```

- OUTPUT statement can be placed in anywhere in a DATA step, and we can have multiple OUTPUT statements in a DATA step. The remaining statements in the DATA step will still be executed.
- The explicit OUTPUT statement asks SAS to write the current observation at the exact moment. Then, the implicit writing of observations at the end of the DATA step will be suppressed.

```
DATA TEST;
X = 1;
OUTPUT; * Explicit OUTPUT;
Y = 1;
* No implicit OUTPUT;
RUN;
```

Example 1

Suppose that we have a text file 'D:\SAS/SALARY.TXT' that stores the employee records. We want to create a SAS data set, called MSalary to store the records of all male employees.

```
DATA MSalary;
INFILE 'D:\SAS\SALARY.TXT';
INPUT NAME $ AGE 2. SEX $ SALARY;
IF SEX = 'M' THEN OUTPUT;
RUN;
```

As there is an OUTPUT statement in the data step, SAS do not add an implicit OUTPUT statement at the end of the program. We can have the same effect if the above IF statement is replaced by

```
IF ^(SEX = 'M' ) THEN DELETE;
```

The DELETE statement asks SAS to return to the beginning of the DATA step without outputting the current record to the output data set, and without executing remaining statements in the DATA step.

In this latter case, an implicit OUTPUT statement will be added by SAS after the IF statement. Therefore, the program is in fact equivalent to

```
DATA MSalary;
INFILE 'D:\SAS\SALARY.TXT';
INPUT NAME $ AGE 2. SEX $ SALARY;
IF ^(SEX = 'M' ) THEN DELETE;
OUTPUT;
RUN;
```

Suppose we change the program to

```
DATA MFSalary;
INFILE 'D:\SAS\SALARY.TXT';
INPUT NAME $ AGE 2. SEX $ SALARY;
IF SEX = 'M' THEN OUTPUT;
IF SEX = 'F' AND SALARY > 11000 THEN
OUTPUT;
RUN;
```

This program has two `OUTPUT` statements. `MFSalary` now contains records for males and females with salaries higher than 11000.

In this case, it is no longer correct to replace the first IF statement by

```
IF ^(SEX = 'M' ) THEN DELETE;
```

because no OUTPUT statement will be added to the program by SAS. Records with SEX = 'M' will not be outputted. Furthermore, if SEX = 'F', SAS will return to the beginning of the DATA step after the first IF statement and therefore the second IF statement will not be executed. A correct revision should read as follows.

```
DATA MFSalary;
INFILE 'D:\SAS\SALARY.TXT';
INPUT NAME $ AGE 2. SEX $ SALARY;
IF SEX = 'F' AND SALARY > 11000 THEN OUTPUT;
IF ^(SEX = 'M' ) THEN DELETE; OUTPUT;
RUN;
```

Example 2

The following program extracts a random subset with approximately 30% of the records from a data set STD. Remember that RANUNI(0) generates Unif(0,1) random numbers.

```
DATA SAMPLE;
SET STD;
IF (RANUNI(0) < 0.3) THEN OUTPUT;
RUN;
```

Example 3

Create a data set from an existing data set EMPLOYEE storing only records for employees called David.

```
DATA DAVID;
SET EMPLOYEE;
IF (INDEXW(UPCASE(NAME),'DAVID') > 0) THEN OUTPUT;
RUN;
```

We can replace the IF statement by

```
IF INDEXW(UPCASE(NAME),'DAVID') THEN OUTPUT;
```

Since we do not know whether `'David'` is entered as `'David'` or `'DAVID'`, we convert all characters to uppercase and check for the character string `'DAVID'`.

Question: What will happen if `INDEXW` function is replaced by `INDEX`?

Answer: In this case, records for persons named `'Davidson'` would also be stored.

WHERE statement

When SAS data is retrieved using a SET statement in a DATA step, we can select the observations with a WHERE data set option or a WHERE statement. The WHERE statement has the form

```
    WHERE condition;
```

With the WHERE statement, only the observations satisfying condition are selected. For example, the IF statement in Example 3 could be replaced by the following WHERE statement.

```
DATA DAVID;
SET EMPLOYEE;
WHERE (INDEXW(UPCASE(NAME),'DAVID') > 0); RUN;
```

IF vs WHERE

- IF statement is similar to the WHERE statement from previous examples. Both IF and WHERE statements can select observations in a DATA step.
- However, the WHERE statement is more efficient than the IF statement in subsetting data sets. This is because the WHERE statement executes prior to an observation being read by the SET statement whereas the IF statement executes after an observation is read.
- On the other hand, the WHERE statement is restricted to select observations based on variables in the existing data sets. The use of subsetting IF statement is more flexible as shown in Example 2.

DROP and KEEP Statements

In order to control which columns (**variables**) are to be stored in the created data set, we can use DROP or KEEP statements.

These two statements can only be used in the DATA step. The syntax of the DROP and KEEP statements are:

```
    DROP varlist;
    KEEP varlist;
```

DROP statement asks SAS not to include the variable(s) given in `varlist` in the output data set (variables not in the list are kept).

KEEP statement instructs SAS to include the variable(s) given in the `varlist` in the output data set (variables not in the list will not be stored in the data set).

The variables that are not going to be stored in the output data set can still be used in the program.

Do not use both DROP and KEEP statements in the same DATA step.

Example 4

Suppose we have a SAS data set in `D:\SAS` with file name `CLASS.SAS7BDAT`. It stores the following information of a class of students:

`SID`: Student ID number

`Name`: Name of student

`Birth`: Year of birth

`Gender`: Gender

`Father`: Father's name, in case student's father is alive; missing otherwise.

`Mother`: Mother's name, in case student's mother is alive; missing otherwise.

The following program creates a SAS data set which stores the following information:

`Parent`: name of student's parent

`Relation`: relation to student (Father/Mother)

`Name`: name of student

```
LIBNAME SCHOOL 'D:\SAS';
DATA SCHOOL.PARS;
  SET SCHOOL.CLASS;
  KEEP PARENT RELATION NAME;
  IF FATHER ^= '' THEN DO;
     PARENT = FATHER;
     RELATION = 'FATHER';
     OUTPUT;
  END;
  IF MOTHER ^= '' THEN DO;
     PARENT = MOTHER;
     RELATION = 'MOTHER';
     OUTPUT;
  END;
RUN;
```

Notice that if both parents of a student are alive, two records will be generated in PARS. One is for his/her father and one for his/her mother.

Creating more than one data set

We can create more than one data set in a single DATA step using the following DATA statement:

```
DATA [dsname...];
```

where `dsname` is a list of data file names.

When we create several data sets, we usually want them to have different contents. To do so, we need the following extended `OUTPUT` statement:

```
OUTPUT [dsname...];
```

The `dsname` behind the keyword `OUTPUT` is the name of the data set(s) to which SAS sends the observation. If `dsname` is not specified after `OUTPUT`, the default is all the SAS data sets created in the step.

Then, the DATA step would look like

```
DATA [dsname...];
<We can put some SAS statements here.>
OUTPUT [dsname...];
<We can put some SAS statements here.>
RUN;
```

The DATA statement informs SAS that two or more SAS data sets are to be created. Multiple OUTPUT statements are permitted in a DATA step.

Example 5

Suppose that we want to split the data file CLASS into two. One is for male student and one is for female student.

```
DATA MALE FEMALE;
  SET SCHOOL.CLASS;
  IF GENDER = 'M' THEN OUTPUT MALE;
  ELSE IF GENDER = 'F' THEN OUTPUT FEMALE;
RUN;
```

Question: What happens to the records with missing `GENDER` value?

Answer: The record will not be outputted to any files.

Question: What happens if `OUTPUT MALE;` is replaced by `OUTPUT;`?

Answer: The data set `MALE` contains records with `GENDER = 'M'` and the data set `FEMALE` contains records with `GENDER = 'M'` or `'F'`.

Question: For the data files MALE and FEMALE, the variable GENDER is no longer informative since all records in the data set have the same value of GENDER. How can we delete the variable GENDER?

Answer: We can modify the program as follows:

```
DATA MALE FEMALE;
  SET SCHOOL.CLASS; DROP GENDER;
  IF GENDER = 'M' THEN OUTPUT MALE;
  ELSE IF GENDER = 'F' THEN OUTPUT FEMALE;
RUN;
```

Note that the DROP statement has effect on all output files. In other words, variable GENDER is dropped from both MALE and FEMALE.

In the previous example, we use a DROP statement to drop a variable from all output data sets.

Another way to control which columns (variables) are stored is to use DROP = or KEEP = option in a DATA statement. The syntax is

```
DATA [dsname({DROP=variable_list} |
    {KEEP=variable_list})] ...;
```

Example 6

Consider the CLASS example again.

Suppose that we do not want to store the ages of female students as they are considered to be sensitive information. The program can read as follows:

```
DATA MALE(DROP = GENDER) FEMALE(DROP = GENDER
BIRTH);
  SET SCHOOL.CLASS;
  IF GENDER = 'M' THEN OUTPUT MALE;
  ELSE IF GENDER = 'F' THEN OUTPUT FEMALE;
RUN;
```

Example 7

We want to create a SAS data file storing the average mark, but not individual subject marks, of each student.

```
DATA MARK(DROP = CHINESE -- PHY);
INFILE 'C:\STUDENT';
INPUT NAME $ SEX $ AGE
CHINESE ENGLISH MATH BIO CHEM PHY;
AVE = MEAN(OF CHINESE -NUMERIC– PHY);
RUN;
```

The DATA statement can be replaced by

```
DATA MARK(KEEP = NAME SEX AGE AVE);
```

Note that although we are going to drop all individual subject marks, their values are still available so that we can compute the average using an assignment statement.

The DROP and KEEP options are useful particularly when we have two or more data sets. It is because the DROP and KEEP statements apply to all output data sets, but the DROP and KEEP options apply only to the output data set it attaches.

Therefore, we can drop one variable from an output file but not from another output file.

STOP statement

STOP statement tells SAS to terminate DATA step execution. Execution of the program resumes at the next unit of work. The syntax is

```
    STOP;
```

STOP statement allows the data set(s) in the DATA statement to be replaced. The current observation will not be written to the output data set.

Example 8

For the following SAS program, the data file TEMP contains only four observations of X with values 1, 3, 5, and 7 respectively. The fifth record (the one with X = .) and the data thereafter are not stored.

```
DATA TEMP;
   INPUT X @@;
   IF X = . THEN STOP;
   CARDS;
   1 3 5 7 . 9 . 11
RUN;
```

Example 9

The following program instructs SAS to read data and compute the compound interest daily (annual interest rate is 5%). We stop reading data if the program encounters an error, say the starting date is in the future.

```
DATA ABC;
INFILE 'D:\SAS\RECORDS.TXT';
INPUT ACC_NO START_D DDMMYY10. +1 AMOUNT COMMA7.;
T=TODAY();
IF (T < START_D) THEN STOP;
INTEREST = AMOUNT *((1.05)**((T - START_D)/365) -
1);
RUN;
```

Data set options

- KEEP and DROP options are examples of data set options. Here we introduce more about other data set options.
- The syntax of data set options is

```
      dsname (options)
```

  where `dsname` is the name of the data set.
- The data set options can apply to any data set in the DATA step, such as the DATA statement, and the SET statement, while some of them can also be used in the PROC step.

Below is a list of some common data set options.

```
KEEP = variable_list
```

Select the variables in the list for use. It cannot be used with DROP option.

<u>Examples</u>

```
DATA dsname (KEEP = A B);
```

Only variables A and B will be stored in the data set `dsname`.

```
SET dsname (KEEP = A B);
```

Only variables A and B will be read from the data set `dsname`.

```
DROP = variable_list
```

Not to select the variables in the list for use. It cannot be used with KEEP option.

Examples

```
DATA dsname (DROP = A B);
```

Variables A and B will not be stored in the data set `dsname`.

```
SET dsname (DROP = A B);
```

Variables A and B will not be read from the data set `dsname`.

```
RENAME = ({OLD_NAME = NEW_NAME} ...)
```

Change the name of the variable from OLD_NAME to NEW_NAME.

If it is used with the KEEP option, the KEEP option should be based on the old variable names.

Example

```
SET ABC (RENAME = (OLD=NEW N=M));
```

All variables in ABC will be read. However, the variable OLD in ABC will be read using a new name NEW, and the variable N in ABC will be read using a new name M.

Note that it would not change the contents of ABC. In ABC, we still have the variables OLD and N.

```
WHERE = (condition)
```

Select observations that satisfy `condition`.

If it is used on an **input** SAS data set, only the observations that meet the condition are available to the program.

```
SET ABC (WHERE = (AGE > 20));
```

If it is used on an **output** SAS data set, only observations that meet the condition are stored in the output SAS data set

```
DATA ABC (WHERE = (AGE > 20));
```

The variable(s) involved in the condition must be available in the output data set. Therefore,

```
DATA ABC (DROP = AGE WHERE = (AGE > 20));
```

is **incorrect**.

```
FIRSTOBS = n
```

Tells SAS that the n-th observation is the first observation to be used. If it is used with the `WHERE` option, n corresponds to the n-th observation that satisfies the `WHERE` condition.

```
OBS = n
```

Tells SAS that n is the last observation number to be used. If it is used with the `WHERE` option, n corresponds to the n-th observation that satisfies the `WHERE` condition.

Example

```
SET ABC (FIRSTOBS = 10 OBS = 20);
```

Read observations 10 through 20 from ABC.

```
LABEL = 'label'
```

Give a data set label.

Example

```
DATA ABC (LABEL = 'National Park
Data');
```

Example 10

Instead of drawing random subset from a data set STD as in Example 2, we want to store the 10th to 17th records with MATH > 70 in a data set SAMPLE.

```
DATA SAMPLE;
SET STD(FIRSTOBS = 10 OBS = 17
WHERE=(MATH > 70));
RUN;
```

Question: Can we rewrite the program as follows?

```
DATA SAMPLE(FIRSTOBS = 10 OBS = 17 WHERE=(MATH
> 70));
SET STD;
RUN;
```

Answer: No. The `FIRSTOBS` and `OBS` options cannot be used in the output data set.

Question: Can we rewrite the `SET` statement as follows?

```
SET STD(OBS = 17 WHERE=(MATH > 70) FIRSTOBS =
10);
```

Answer: Yes. The ordering of the options is not important.

Example 11

In Example 4, we create a SAS data set, called PARS storing the information of the parents of a class of students. Now we want to split it into two; one for father and one for mother.

```
DATA CLASS_FATHER(RENAME=(PARENT=FATHER)
LABEL='Data set for fathers of students')
CLASS_MOTHER(RENAME=(PARENT=MOTHER)
LABEL='Data set for mothers of students');
SET SCHOOL.PARS;
DROP RELATION;
IF (RELATION = 'FATHER') THEN OUTPUT CLASS_FATHER;
ELSE OUTPUT CLASS_MOTHER;
RUN;
```

The above program creates two files CLASS_FATHER and CLASS_MOTHER.

We also rename the variable PARENT when it is stored in the two output data files. If we do not use the RENAME option, the program may be written as follows.

```
DATA CLASS_FATHER(DROP = MOTHER
LABEL='Data set for fathers of students')
CLASS_MOTHER(DROP = FATHER
LABEL='Data set for mothers of students');
SET SCHOOL.PARS;
DROP PARENT RELATION;
IF (RELATION = 'FATHER') THEN DO;
FATHER=PARENT; OUTPUT CLASS_FATHER; END;
ELSE DO; MOTHER=PARENT; OUTPUT CLASS_MOTHER; END;
RUN;
```

Question: Is it acceptable if we name the two output SAS data files as FATHER and MOTHER respectively?

Answer: Yes, but it is not recommended as it may cause confusion since the data sets will then have the same name as their variables.

Question: Can the IF statement be replaced by

`IF (RELATION = 'Father') THEN DO;` ?

Answer: No, it is because value 'FATHER' is assigned to RELATION, and 'Father' is not equal to 'FATHER', and thus CLASS_FATHER will be empty and CLASS_MOTHER will contain all records.

The following example illustrate how the data set option can be used in PROC step

We want to print the first 10 observations with REGION = 1 of a data set AA.

```
PROC PRINT DATA=AA(OBS=10 WHERE=(REGION=1));
RUN;
```