Paper 030-31

# The SORT Procedure: Beyond the Basics

Britta KelseyBassett, MGI PHARMA, Bloomington, MN

## ABSTRACT

The SORT procedure is a very useful procedure with many options that often get overlooked. There are many options that you can use in both a DATA step as well as the SORT procedure, consequently eliminating the need for a DATA step in many instances. In this paper, I will describe these options in greater detail and show examples of how to use them. All examples shown were done in the SAS® system for PCs, version 8.2. The intended audience for this paper is beginner level SAS programmers with the basic knowledge of the SORT procedure.

## INTRODUCTION

One of the first things that most SAS programmers learn is how to sort data using PROC SORT, but did you know you can also do things like create a new data set, subset your data, rename, drop, keep, format, or label your variables all in that same procedure? Throughout this paper I will go beyond the basics of PROC SORT and explore some of its options further.

## OUT= OPTION

When programming with SAS, do you ever create a data set by using a DATA step and then sort this new data set using PROC SORT? You can actually accomplish both of these tasks within just the SORT procedure by using the OUT= option. Without the OUT= option, PROC SORT overwrites the original data set. This may be fine when doing a basic sort but if you start to use more data set options, which I will expand on later in this paper, you may want to use the OUT= option. Here is an example:

```
data demo;
  input patient 1-2 sex $ 3-4 age 5-7 ps 8-9;
     datalines;
       1 F 45 0
       4 M 63 2
       3 M 57 1
       5 F 72 3
       2 F 39 0
       3 M 57 1
       4 M 63 0
       ;
run;

proc sort data=demo out=demo1;
  by patient;
run;
```

In this example, I first created a data set, called DEMO, which will be used throughout this paper. This data set is similar to one used for a clinical study and contains patient identification numbers, sex, age at diagnosis, and performance status. Within the PROC SORT, I created a new data set called DEMO1 that is sorted by patient number. Here is how the output data set DEMO1 looks:

| PATIENT | SEX | AGE | PS |
|---|---|---|---|
| 1 | F | 45 | 0 |
| 2 | F | 39 | 0 |
| 3 | M | 57 | 1 |
| 3 | M | 57 | 1 |

| | | | |
|---|---|---|---|
| 4 | M | 63 | 2 |
| 4 | M | 63 | 0 |
| 5 | F | 72 | 3 |

You can see this is just a basic sort leaving all the same variables and number of observations as in the DEMO data set but the data set DEMO1 is sorted by patient number.

### DESCENDING OPTION

Sometimes it may be useful to reverse the order of the BY variable. You can do this using the DESCENDING option. This option will sort the data set in descending order by the variable that immediately follows the word DESCENDING in the BY statement. Here is an example followed by how the output data set looks:

```
proc sort data=demo out=demo2;
  by descending patient;
run;
```

DEMO2 data set:

| PATIENT | SEX | AGE | PS |
|---|---|---|---|
| 5 | F | 72 | 3 |
| 4 | M | 63 | 2 |
| 4 | M | 63 | 0 |
| 3 | M | 57 | 1 |
| 3 | M | 57 | 1 |
| 2 | F | 39 | 0 |
| 1 | F | 45 | 0 |

You can see that the data set DEMO2 is now sorted with patient number 5 first and patient number 1 last.

### DROP=, KEEP=, AND RENAME= OPTIONS

You can use the DROP=, KEEP=, and RENAME= options within the SORT procedure just as you can within a DATA step. Here are some examples using these options along with how the output data sets look:

```
proc sort data=demo out=demo3(keep=patient age);
  by patient;
run;
```

DEMO3 data set:

| PATIENT | AGE |
|---|---|
| 1 | 45 |
| 2 | 39 |
| 3 | 57 |
| 3 | 57 |
| 4 | 63 |
| 4 | 63 |
| 5 | 72 |

```
proc sort data=demo out=demo4(rename=(patient=pt));
  by patient;
run;
```

DEMO4 data set:

| PT | SEX | AGE | PS |
|---|---|---|---|
| 1 | F | 45 | 0 |
| 2 | F | 39 | 0 |
| 3 | M | 57 | 1 |
| 3 | M | 57 | 1 |
| 4 | M | 63 | 2 |
| 4 | M | 63 | 0 |
| 5 | F | 72 | 3 |

There are a few of things to keep in mind when using these options. First, if you use the RENAME= option, SAS changes the name of the variable in that procedure. Second, if you use RENAME= with either the DROP= or the KEEP= options, the DROP= and the KEEP= options are applied before RENAME=. Thus, use the "old name" in the DROP= and KEEP= options. The order in which you place the RENAME=, DROP=, and KEEP= options does not matter and does not change process order. In regard to using parentheses, a list of *multiple* variables to rename must be enclosed in parentheses, renaming just one variable does not, and the KEEP= variables should not be enclosed in parentheses. Another thing to remember is that you cannot drop and rename the same variable in the same statement. Here is an example of using both the KEEP= and RENAME= options:

```
proc sort data=demo out=demo5(rename=(patient=pt age=dxage) keep=patient age);
  by patient;
run;
```

DEMO5 data set:

| PT | DXAGE |
|---|---|
| 1 | 45 |
| 2 | 39 |
| 3 | 57 |
| 3 | 57 |
| 4 | 63 |
| 4 | 63 |
| 5 | 72 |

Notice that in this example you have to use the variable names PATIENT and AGE in the KEEP= statement instead of PT and DXAGE. Also notice that you have to use PATIENT instead of PT in the BY statement. You can actually rearrange this example in a couple of different ways and end up with the same result. Below are two alternate ways:

```
proc sort data=demo out=demo5(keep=patient age rename=(patient=pt age=dxage));
  by patient;
run;

proc sort data=demo(keep=patient age rename=(patient=pt age=dxage)) out=demo5;
  by pt;
run;
```

Note that in the second rearrangement above that the variable name PT must be used instead of PATIENT in the BY statement because the KEEP= and RENAME= options are used before the OUT= option versus after it like in the original configuration.

## FORMAT AND LABEL STATEMENTS

Other statements that are the same in the SORT procedure as in a DATA step are the FORMAT and LABEL statements. You can apply a variable format or create variable labels within PROC SORT. First, I will show an example using a format:

```
proc format;
     value $SEX 'F'='Female'
                'M'='Male';
run;

proc sort data=demo out=demo6;
 format sex $SEX.;
  by patient;
run;
```

DEMO6 data set:

| PATIENT | SEX | AGE | PS |
|---|---|---|---|
| 1 | Female | 45 | 0 |
| 2 | Female | 39 | 0 |
| 3 | Male | 57 | 1 |
| 3 | Male | 57 | 1 |
| 4 | Male | 63 | 2 |

```
4         Male    63    0
5         Female  72    3
```

In this example, the format $SEX was created in the FORMAT procedure. This format was then applied to the variable SEX in a PROC SORT statement. Now, in the output data set, instead of 'F' and 'M', you see the formatted values 'Female' and 'Male'. Note that the FORMAT statement does not permanently alter the variables in the input data set. Next I will show an example using labels:

```
proc sort data=demo out=demo7;
 label ps='Performance Status'
       age='Age at Diagnosis';
  by patient;
run;

proc print data=demo7 label;
run;
```

The output looks like:

| Obs | patient | sex | Age at Diagnosis | Performance Status |
|---|---|---|---|---|
| 1 | 1 | F | 45 | 0 |
| 2 | 2 | F | 39 | 0 |
| 3 | 3 | M | 57 | 1 |
| 4 | 3 | M | 57 | 1 |
| 5 | 4 | M | 63 | 2 |
| 6 | 4 | M | 63 | 0 |
| 7 | 5 | F | 72 | 3 |

By using the PRINT procedure with the label option following the PROC SORT statement, you can see the labels created for the variables PS and AGE. Like the FORMAT statement, the LABEL statement does not permanently alter the variables in the input data set.

## WHERE= OPTION OR WHERE STATEMENT

There are times when it is necessary to subset your data and the SORT procedure allows you to do this by using the WHERE= option or WHERE STATEMENT. Both of these work similarly by selecting observations that meet the condition specified in the WHERE expression before SAS brings them into the PROC SORT for processing. This may improve the efficiency of your SAS programs because SAS is not required to read all observations from the input data set. Here is an example using the WHERE= option:

```
proc sort data=demo(where=(age>50)) out=demo8;
  by patient;
run;
```

Here is an example using the WHERE statement:

```
proc sort data=demo out=demo8;
 where age>50;
  by patient;
run;
```

Both of these produce the same output data set:

| PATIENT | SEX | AGE | PS |
|---|---|---|---|
| 3 | M | 57 | 1 |
| 3 | M | 57 | 1 |
| 4 | M | 63 | 2 |
| 4 | M | 63 | 0 |
| 5 | F | 72 | 3 |

A nice feature of the WHERE= option and WHERE statement is being able to use some exclusive WHERE expressions such as: BETWEEN-AND, ? or CONTAINS, IS NULL or IS MISSING, LIKE (matches patterns), =* (sounds like), and SAME-AND.

## FIRSTOBS= AND OBS= OPTIONS

There may be circumstances when you have a very large data set and you would like to split it up into smaller, more manageable data sets. This may be a time when it would be helpful to use the FIRSTOBS= and OBS= options within the SORT procedure. The FIRSTOBS= option causes SAS to begin reading at a specified observation or record. The OBS= option specifies at which observation SAS processing ends. The two options are often used together to define a range of observations to be processed. However, you do not have to use both options together. If you do not include the OBS= option with the FIRSTOBS= option, by default PROC SORT will stop at the last observation. If you do not include the FIRSTOBS= option with the OBS= option, by default PROC SORT will start at the first observation. Here is an example of using both the FIRSTOBS= and OBS= options:

```
proc sort data=demo(firstobs=3 obs=5) out=demo9;
  by patient;
run;
```

DEMO9 data set:

| PATIENT | SEX | AGE | PS |
|---|---|---|---|
| 2 | F | 39 | 0 |
| 3 | M | 57 | 1 |
| 5 | F | 72 | 3 |

In this example, the procedure first took observations 3, 4, and 5 from the original, *unsorted* data set DEMO and *then* sorted it by PATIENT.

## NODUPRECS AND NODUPKEY OPTIONS

The NODUPRECS (or NODUP) and NODUPKEY options work similarly in that they both can eliminate unwanted observations, but NODUP compares *all* the variables in your data set while NODUPKEY compares just the *BY* variables. More specifically, if you use the NODUP option, PROC SORT compares *all* variable values for each observation to those for the previous observation that was written to the output data set. If an exact match is found, the observation is not written to the output data set. If you specify the NODUPKEY option, PROC SORT compares all *BY* variable values for each observation to those for the previous observation written to the output data set. If an exact match using the BY variable values is found, the observation is not written to the output data set. First I will show an example of the NODUP option:

```
proc sort data=demo nodup out=demo10;
  by patient;
run;
```

DEMO10 data set:

| PATIENT | SEX | AGE | PS |
|---|---|---|---|
| 1 | F | 45 | 0 |
| 2 | F | 39 | 0 |
| 3 | M | 57 | 1 |
| 4 | M | 63 | 2 |
| 4 | M | 63 | 0 |
| 5 | F | 72 | 3 |

Notice in the output data set DEMO10, the second observation for patient 03 is eliminated because all the variable values are the same. However, the second observation for patient 04 is not eliminated because performance score is different for these two observations. Next I will show an example of the NODUPKEY option:

```
proc sort data=demo nodupkey out=demo11;
  by patient;
run;
```

DEMO11 data set:

| PATIENT | SEX | AGE | PS |
|---|---|---|---|
| 1 | F | 45 | 0 |
| 2 | F | 39 | 0 |
| 3 | M | 57 | 1 |
| 4 | M | 63 | 2 |
| 5 | F | 72 | 3 |

In this output data set you can see that the second observations for both patient 03 and 04 are eliminated because only the BY variable (in this case PATIENT) had to be the same. In this example, using NODUPKEY instead of NODUP may not be the better option to use because by using NODUPKEY, you eliminate the second observation for patient 04 even though there was a different performance score for this observation. This may be important information that needs to be kept in the data set.

## CONCLUSION

There are many different, often overlooked, options that are available to the SORT procedure. Now that you have learned more about these options, you can use them to enhance your programs and improve efficiency.

## REFERENCES

SAS Institute Inc. (1990), *SAS Procedures Guide, Version 6, Third Edition,* Cary, NC: SAS Institute Inc.

## ACKNOWLEDGEMENTS

I would like to thank Carrie Loebertmann for her review and helpful comments on this paper.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author at:

Britta KelseyBassett
MGI Pharma, Inc.
5775 West Old Shakopee Road, Suite 100
Bloomington, MN 55437-3174
E-mail: britta.kelseybassett@mgipharma.com