Learning SAS: A Comprehensive Guide to PROC SORT with Examples

Authored by

Mohammed looti

May 10, 2026

The **PROC SORT** procedure in SAS is an indispensable tool for organizing your data, serving as a fundamental building block for analytical workflows. This routine allows users to efficiently rearrange the observations within a dataset by ordering them based on the values of one or more specified variables. Proper data ordering is critical for numerous data analysis tasks, including preparing inputs for sophisticated statistical modeling, ensuring successful dataset merges, or generating cleanly structured reports.

Mastering the effective utilization of **PROC SORT** is essential for significantly enhancing your data manipulation capabilities within the SAS environment. This comprehensive guide will walk you through a series of practical, step-by-step examples. We will demonstrate techniques for sorting data in both ascending and descending sequences, as well as applying complex sorting criteria across multiple columns simultaneously. By building upon a consistent sample dataset, these illustrations will clarify the precise syntax required and the resulting structural changes in the data's organization.

Understanding the Sample Dataset

Before we delve into the mechanics of sorting, it is necessary to establish the baseline data we will use throughout this tutorial. We introduce a sample dataset, appropriately named **original_data**, which contains simple performance metrics--specifically points and rebounds--for various teams. This straightforward structure is intentionally designed to clearly illustrate the immediate and precise effects of the various sorting procedures we will implement.

The initial **DATA step** code below is used to create and populate this preliminary dataset. Following its creation, we employ the **PROC PRINT** procedure to display the data in its raw, unsorted state. This serves as a vital point of comparison, allowing us to accurately assess the impact of the subsequent sorting operations demonstrated in the following examples.

```
/*create dataset*/
data original_data;
input team $ points rebounds;
datalines;
A 12 8
A 12 7
A 14 5
A 23 9
A 20 12
A 11 7
A 14 7
B 20 2
```

```
B 20 5
B 29 4
B 14 7
B 19 8
B 17 9
B 30 9
;
run;

/*view dataset*/
proc print data=original_data;
```

Obs	team	points	rebounds
1	A	12	8
2	A	12	7
3	A	14	5
4	A	23	9
5	A	20	12
6	A	11	7
7	A	14	7
8	B	20	2
9	B	20	5
10	B	29	4
11	B	14	7
12	B	19	8
13	B	17	9
14	B	30	9

As clearly depicted in the output, the **original_data** dataset currently lacks any predetermined sequence. The observations are merely listed in the chronological order of their entry, a structure that is rarely optimal for analytical endeavors or formal presentation. The subsequent examples will detail the exact commands required to transform this raw input into highly organized and structured formats using the power of **PROC SORT**.

Example 1: Sorting Observations in Ascending Order

Arranging data from the smallest value to the largest, known as **ascending order**, represents the most frequent application of the **PROC SORT** procedure. This is the default behavior initiated by

SAS whenever you specify a sorting variable using the **BY statement** without any modifiers.

In this introductory example, our goal is to sort the **original_data** dataset based exclusively on the values contained within the **points** column. We will utilize the optional **OUT=** parameter to save the resulting, organized data into a distinct new dataset named **data2**. This practice of generating a separate output dataset is highly recommended best practice, as it ensures the long-term integrity and preservation of your original source data.

```
/*sort by points ascending*/
proc sort data=original_data out=data2;
by points;
run;

/*view sorted dataset*/
proc print data=data2;
```

Obs	team	points	rebounds
1	A	11	7
2	A	12	8
3	A	12	7
4	A	14	5
5	A	14	7
6	B	14	7
7	B	17	9
8	B	19	8
9	A	20	12
10	B	20	2
11	B	20	5
12	A	23	9
13	B	29	4
14	B	30	9

The visual output of the resulting dataset, **data2**, confirms that the observations have been successfully rearranged. They are now listed sequentially in ascending order based on the magnitude of the values in the **points** column. This simple execution effectively demonstrates the core, default functionality of **PROC SORT**: arranging data from the minimum value up to the maximum value.

Example 2: Sorting Observations in Descending Order

While ascending order is often the statistical default, many analytical tasks, such as ranking performance or identifying outliers, specifically require data to be sorted from the largest value to the smallest, or in **descending order**. The **PROC SORT** procedure facilitates this requirement using a concise keyword modifier.

To reverse the natural sorting sequence, you simply need to precede the variable name within the **BY statement** with the **DESCENDING** keyword. This explicit instruction directs SAS to invert the default sorting direction for that particular variable. For this example, we will apply the **DESCENDING** instruction to the **points** column of our **original_data**, storing the resultant dataset in **data3**.

```
/*sort by points descending*/
proc sort data=original_data out=data3;
by descending points;
run;

/*view sorted dataset*/
proc print data=data3;
```

Obs	team	points	rebounds
1	B	30	9
2	B	29	4
3	A	23	9
4	A	20	12
5	B	20	2
6	B	20	5
7	B	19	8
8	B	17	9
9	A	14	5
10	A	14	7
11	B	14	7
12	A	12	8
13	A	12	7
14	A	11	7

A careful review of the **data3** dataset confirms that the observations are now rigorously sorted in descending order, organized by the highest score in the **points** column down to the lowest. This capability is paramount for generating reports where the immediate identification of top-ranking entities or maximum values is essential for quick decision-making.

Example 3: Sorting Observations by Multiple Columns

Relying on a single sorting criterion is frequently insufficient for achieving complex or highly specific data organization needs. **PROC SORT** provides powerful, granular control by enabling the sorting of observations based on multiple variables simultaneously, establishing a hierarchical structure. When two or more variables are specified sequentially in the **BY statement**, SAS performs a multi-level sort.

The sorting process begins with the primary sort key--the first variable listed. If the values of this initial variable result in ties (i.e., multiple observations have the same value), SAS then uses the second variable listed to break those ties and sort the tied subset. This recursive process continues through every subsequent variable specified. Crucially, each individual variable in the **BY** statement can be independently modified to sort in either ascending (default) or descending order.

In this advanced example, we will instruct SAS to sort the **original_data** hierarchically: first, by the **points** column in ascending order, and second, for all observations sharing identical **points** scores, they must be sub-sorted by the **rebounds** column, also in ascending order. The final, meticulously sorted output will be stored in a new dataset named **data4**.

```
/*sort by points ascending, then by rebounds ascending*/
proc sort data=original_data out=data4;
by points rebounds;
run;

/*view sorted dataset*/
proc print data=data4;
```

Obs	team	points	rebounds
1	A	11	7
2	A	12	7
3	A	12	8
4	A	14	5
5	A	14	7
6	B	14	7
7	B	17	9
8	B	19	8
9	B	20	2
10	B	20	5
11	A	20	12
12	A	23	9
13	B	29	4
14	B	30	9

The final dataset, **data4**, perfectly demonstrates the principles of hierarchical sorting. Observations are primarily ordered by the **points** variable. Wherever multiple observations share the same value for **points** (e.g., points = 12 or points = 14), they are then correctly sub-sorted based on their respective **rebounds** values. This multi-variable functionality is indispensable for detailed data preparation and complex analytical grouping.

Conclusion: Mastering Data Organization with PROC SORT

The **PROC SORT** procedure stands as a foundational cornerstone of robust data management within the **SAS statistical environment**. It offers unparalleled functionality for organizing datasets with high precision, tailored exactly to analytical requirements. Ranging from straightforward ascending or descending arrangements on a single variable to complex, hierarchical sequencing across multiple columns, the flexibility offered by this procedure is invaluable for every stage of data preparation, analysis, and reporting.

By achieving proficiency in **PROC SORT**, programmers gain essential control over the structural integrity of their data, a prerequisite for accurate and computationally efficient data processing. A key takeaway is the consistent recommendation to utilize the **OUT=** option to save the sorted result into a new dataset, thereby ensuring the permanent maintenance and integrity of your original source data against accidental modification.

Additional Resources for SAS Programming

To further expand your expertise in SAS programming and data manipulation techniques, we encourage you to explore these authoritative resources and related tutorials that explain how to perform other common data management tasks:

Official SAS Documentation for PROC SORT Syntax
Introduction to the SAS DATA Step Environment
Utilizing SAS PROC PRINT for Data Inspection and Viewing