Learning SAS: Sorting Data with PROC SORT and the KEEP Statement

Authored by

Mohammed looti

November 16, 2025

Optimizing Data Workflows: Integrating Sorting and Subsetting in SAS

In the specialized field of statistical computing, particularly within the SAS environment, the ability to efficiently manage, organize, and refine massive quantities of information is foundational to successful data analysis. The **PROC SORT** procedure is arguably the most critical command for data organization, primarily designed to rearrange the rows--or observations--of a dataset based on specific keys defined by one or more variables. While simple sorting is essential, high-performance data preparation often requires simultaneous dimensionality reduction; analysts frequently need to retain only a subset of the original fields immediately after the reordering process is complete.

This demand for combined functionality--sorting and column selection--is elegantly met by incorporating the **KEEP statement** directly into the **PROC SORT** step. The **KEEP statement** is a powerful dataset option that explicitly dictates which columns should be preserved in the resulting output dataset, effectively performing a critical subsetting operation alongside the physical reordering of observations. This integrated approach ensures that the output data is not only correctly sorted but also perfectly structured for the next stage of the workflow, offering significant advantages in both coding simplicity and execution speed.

By mastering this combined technique, data professionals can construct highly optimized data manipulation processes that conserve system resources and improve code maintainability. This comprehensive guide will detail the precise syntax required to utilize **PROC SORT** with the **KEEP statement**, provide a clear, practical demonstration using sample data, and thoroughly outline the strategic benefits this methodology offers for enterprise-level data processing and resource management.

Understanding the Integrated Syntax Structure

The key to successfully implementing simultaneous sorting and column selection lies in understanding the precise placement and function of the **KEEP statement** within the **PROC SORT** syntax. Crucially, in this context, the **KEEP statement** is not used as a standalone procedure statement; instead, it operates as a dataset option, which must be appended directly to the reference for the output dataset. This placement ensures that the column reduction occurs dynamically during the creation of the new dataset, maximizing efficiency.

The fundamental structure for executing this integrated command is concise yet comprehensive, clearly defining the source data, the desired output name, the fields to retain, and the specific criteria for sorting:

```
proc sort data=my_data out=sorted_data (keep=var1 var2);
by var2;
```

run;

A precise understanding of each element within this syntax is necessary for correct procedure execution:

The `proc sort` command initiates the data sorting routine.
`data=my_data` designates the existing, unsorted input data source.
`out=sorted_data` provides the name for the newly created, rearranged output dataset.
The essential component is the `(keep=var1 var2)` option, which must be enclosed in parentheses and placed immediately after the output dataset name. This dataset option instructs the procedure to carry forward only the listed variables (`var1` and `var2`), discarding all other columns present in the source data.
The `by var2` statement establishes the sorting key, meaning the rows of the dataset will be ordered based on the values in `var2`, typically defaulting to ascending order.
Finally, the `run` statement executes the entire procedure block, finalizing the sort and subsetting operation.

This methodology ensures that the sorting operation is performed efficiently, and the resulting dataset is immediately structured to contain only the necessary fields, providing a critical level of control during the data preparation phase.

Demonstration Setup: Creating the Initial Source Data

To provide a clear visual and functional illustration of combining sorting with column selection, we must first establish a robust source dataset. For this example, we will simulate a common scenario in data analysis involving sports metrics, where raw data often contains extraneous variables that must be excluded for specific reports. We will create a sample dataset in SAS containing metrics for fictional basketball teams.

The following SAS code block generates our initial input dataset, named `my_data`, which includes three distinct variables: `team` (character), `points` (numeric score), and `assists` (numeric count).

```
/*create dataset*/
data my_data;
input team $ points assists;
datalines;
Mavs 113 22
Pacers 95 19
Cavs 100 34
Lakers 114 20
```

```
Heat 123 39
Kings 100 22
Raptors 105 11
Hawks 95 25
Magic 103 26
Spurs 119 29
;
run;

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

Upon successful execution, the final `proc print` statement verifies the structure of the input data, showing all ten observations (teams) and three associated variables. This raw dataset acts as the essential starting point for our manipulation, allowing us to accurately observe and confirm the structural changes introduced by the subsequent sorting and subsetting operations.

Obs	team	points	assists
1	Mavs	113	22
2	Pacers	95	19
3	Cavs	100	34
4	Lakers	114	20
5	Heat	123	39
6	Kings	100	22
7	Raptors	105	11
8	Hawks	95	25
9	Magic	103	26
10	Spurs	119	29

Establishing a Baseline: Standard Sorting Behavior

Before proceeding to the advanced combination technique, it is beneficial to review the default behavior of the sorting procedure. When **PROC SORT** is executed without specific dataset options like `KEEP` or `DROP`, it performs the necessary reordering based on the specified key variables but faithfully carries over all original columns into the new output dataset. This standard behavior ensures the preservation of complete data integrity unless explicitly instructed otherwise.

We will sort our `my_data` based solely on the `points` column. This step demonstrates that while the positional order of the rows changes, the dimensionality (the number of columns) of the data remains constant, confirming the need for the **KEEP statement** when column reduction is required.

```
/*sort rows in dataset based on values in points column*/
proc sort data=my_data out=sorted_data;
by points;
run;

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

Reviewing the output confirms that the observations in `sorted_data` are now ordered by `points`, from the lowest score upward, successfully demonstrating the basic sorting function. Critically, as expected for the standard operation, all three variables--`team`, `points`, and `assists`--are still present in the output. This result underscores why utilizing the **KEEP statement** is necessary when the goal is to perform efficient subsetting and exclude unnecessary information from the final dataset.

Obs	team	points	assists
1	Pacers	95	19
2	Hawks	95	25
3	Cavs	100	34
4	Kings	100	22
5	Magic	103	26
6	Raptors	105	11
7	Mavs	113	22
8	Lakers	114	20
9	Spurs	119	29
10	Heat	123	39

Refining Data Structure: Implementing KEEP within PROC SORT

When preparing a dataset for specific reports, analytical modeling, or sharing, retaining extraneous columns can significantly impede performance and clarity, especially when handling large-scale data. This scenario highlights the core benefit of integrating the **KEEP statement** directly within

PROC SORT, enabling immediate data structure optimization.

Let us refine our previous example. Assume our immediate task requires the teams to be sorted by their score, but the `assists` column is irrelevant for the current reporting need. We can modify the **PROC SORT** step to include the **KEEP statement** as a dataset option, thereby limiting the output to only the `team` and `points` variables. This single action efficiently achieves both goals: sorting the data and reducing the dataset's physical footprint.

```
/*sort rows in dataset based on values in points column and only keep team and points*/
proc sort data=my_data out=sorted_data (keep=team points);
by points;
run;

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

The resulting `sorted_data` generated by this refined code block clearly demonstrates the dual functionality. The rows are still accurately arranged in ascending order based on `points`, but the column structure has been successfully simplified. Only the `team` and `points` columns are present; the `assists` column has been completely excluded from the output dataset, proving that precise control over the data structure can be achieved simultaneously with the sorting procedure.

Obs	team	points
1	Pacers	95
2	Hawks	95
3	Cavs	100
4	Kings	100
5	Magic	103
6	Raptors	105
7	Mavs	113
8	Lakers	114
9	Spurs	119
10	Heat	123

Strategic Benefits for High-Performance Data Processing

The integration of **PROC SORT** and the **KEEP statement** transcends mere coding efficiency; it

represents a fundamental strategic optimization in data manipulation within the demanding SAS environment. These procedural efficiencies become critically important when analysts manage large-scale, enterprise data where minimizing processing time, memory use, and I/O operations is paramount.

Resource Optimization: By immediately discarding unnecessary columns, the resulting output dataset is substantially smaller. This reduction in size directly minimizes input/output (I/O) burdens, lowers memory consumption during processing, and results in significantly faster overall data processing times. This advantage is amplified when the reduced output dataset is utilized repeatedly in subsequent analytical or reporting steps.

Enhanced Clarity and Maintainability: A dataset that contains only the essential information is inherently easier to audit, manage, and interpret. Analysts can focus their attention purely on the required metrics, which reduces the potential for coding errors and substantially simplifies the logic required for downstream statistical modeling or complex report generation.

Streamlined Workflow: Employing the **KEEP statement** as a dataset option within the sort procedure eliminates the need for a separate, dedicated data step for column selection. This crucial consolidation simplifies the code base, making the entire workflow more readable, cleaner, and easier to maintain, especially within complex programming projects.

Facilitating Data Governance and Security: In regulated industries, such as pharmaceutical research or financial services, data often contains sensitive identifiers or proprietary values that must be restricted. This combined method allows programmers to quickly generate a sorted, sanitized public version of the data, ensuring only permissible information is carried forward, thereby adhering to strict data subsetting protocols.

Leveraging this powerful combined technique ensures that data is not only correctly ordered but also perfectly shaped and sized for the next phase of data analysis or reporting, maximizing operational efficiency throughout the entire data lifecycle.

Next Steps in Advanced Data Processing

Achieving mastery in SAS programming fundamentally relies on understanding how to leverage various procedures and statements for highly effective data management. The strategic combination of **PROC SORT** with the **KEEP statement** is a crucial foundational technique for efficient data processing and the creation of focused, streamlined datasets.

To further enhance your proficiency in SAS and build upon these concepts, it is highly recommended to explore documentation and tutorials covering related advanced techniques. This includes utilizing the `DROP` statement (which functions as the precise inverse of `KEEP`) or

exploring the use of data views to prevent the creation of physical datasets altogether, saving considerable disk space and I/O resources. These advanced methods build directly upon the principles of efficiency and control demonstrated through the combined use of sorting and column selection.

The following tutorials explain how to perform other common tasks in SAS: