Learning SAS: How to Sort Data and Remove Duplicates with PROC SORT and NODUPKEY

Authored by

Mohammed looti

May 9, 2026

Mastering Data Ordering and Uniqueness with PROC SORT and NODUPKEY in SAS

In modern statistical software environments, efficiency and data integrity are paramount. SAS remains a foundational tool for advanced data analysis and complex manipulation tasks. Central to nearly all SAS workflows is the ability to structure and clean incoming information. The **PROC SORT** statement is the primary mechanism used for ordering data, a necessary step for sequential processing or visual reporting. However, its true power is unlocked when combined with the **NODUPKEY** option. This procedural pairing goes beyond simple organization, providing a streamlined method to enforce uniqueness within a dataset based on specific sorting variables. Mastering this technique is essential for any professional tasked with high-stakes data cleaning, ensuring that redundant observations do not skew subsequent statistical models.

Data quality is fundamentally compromised by duplicate records, which can inflate counts, distort distributions, and lead to faulty conclusions during data interpretation. The challenge often lies in identifying and systematically removing these redundancies based not on the entire record, but on a subset of key fields. The **NODUPKEY** option specifically targets this need, instructing SAS to compare only the values of the variables listed in the **BY statement**. If the key values match across multiple observations, only the first occurrence is retained in the output file, effectively de-duplicating the data set based on the defined primary keys. This technique is far more robust and efficient than manual checks or complex conditional logic, establishing it as a cornerstone of reliable data preparation in the SAS environment.

This guide provides a focused, step-by-step examination of how to leverage the **PROC SORT** procedure in conjunction with the powerful **NODUPKEY** option. We will walk through the practical syntax, demonstrating how this combination transforms raw data into a structured and unique analytical asset. By the end of this tutorial, you will possess a clear understanding of how to implement this critical procedure to ensure your data is perfectly sorted and uniquely represented according to your analytical needs, whether you are preparing data for advanced modeling or simply generating clean reports.

Preparing and Inspecting Our Sample Dataset

To fully appreciate the functionality of **PROC SORT** and **NODUPKEY**, we must first establish a sample environment containing the exact data issues we aim to resolve. We begin by creating a sample dataset, named `original_data`, using the SAS code **DATA step**. This dataset models sports statistics, specifically tracking team performance across two crucial metrics: `points` and `rebounds`. Crucially, this input data has been intentionally structured to include multiple identical records, simulating the common real-world problem of data entry errors or flawed merging processes.

The subsequent code snippet utilizes the **DATALINES** statement to inject raw data directly into the program, followed by **PROC PRINT** to display the initial, unsorted, and duplicated contents of the `original_data` dataset. Observing this raw state is essential, as it provides the baseline against which we will measure the effectiveness of our sorting and de-duplication efforts. The visual output immediately highlights the redundancy, such as several records for Team A achieving 12 points and 8 rebounds, and similar repeating entries for Team B, confirming the need for robust data preprocessing.

```
/*create dataset*/
data original_data;
input team $ points rebounds;
datalines;
A 12 8
A 12 8
A 12 8
A 23 9
A 20 12
A 14 7
A 14 7
B 20 2
B 20 5
B 29 4
B 14 7
B 20 2
B 20 2
B 20 5
;
run;

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

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

As clearly demonstrated by the initial output, our `original_data` contains numerous instances where entire records are identical across all dimensions (`team`, `points`, and `rebounds`). While these are instances of full record redundancy, the **NODUPKEY** option allows for more flexible de-duplication, where uniqueness can be defined by a single field or a combination of fields. The prevalence of these redundant observations necessitates an immediate and effective strategy for data cleaning before any meaningful data analysis can be conducted.

Initial Sorting: Understanding Standard PROC SORT Behavior

Before introducing the de-duplication logic, it is crucial to first isolate and understand the fundamental mechanism of the **PROC SORT** procedure itself. The primary function of **PROC SORT** is to rearrange the sequence of observations within a SAS dataset based on the values of one or more specified variables. By default, sorting occurs in ascending order, meaning values are arranged from smallest to largest or alphabetically. This basic ordering capability is achieved by defining the sorting criteria using the essential BY statement.

For our first demonstration, we will sort the `original_data` based exclusively on the `points` variable. The syntax below directs **PROC SORT** to take the initial dataset as input and generate a new output dataset named `data2`. We explicitly use the `BY points` statement, which ensures the resulting dataset is sequenced according to the scoring metric. Importantly, without any additional options, this procedure only reorders the data; it does not remove any existing duplicate entries,

regardless of how many identical records exist.

```
/*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	12	8
2	A	12	8
3	A	12	8
4	A	14	7
5	A	14	7
6	B	14	7
7	A	20	12
8	B	20	2
9	B	20	5
10	B	20	2
11	B	20	2
12	B	20	5
13	A	23	9
14	B	29	4

The output for `data2` confirms that the observations are correctly arranged in ascending order based on the `points` field, ranging from 12 up to 29. While this fulfills the sorting requirement, a visual check reveals that the duplicates persist. For example, the record corresponding to 12 points appears three times, and the record for 14 points appears twice. This output clearly illustrates the limitation of a standard **PROC SORT** when the objective is to create a clean, unique data structure suitable for accurate statistical modeling or summarized reporting.

Implementing Uniqueness: The Power of NODUPKEY

The inherent redundancy observed in the previous sorted dataset is precisely the problem solved

by the **NODUPKEY** option. This powerful directive transforms **PROC SORT** from a simple ordering utility into a critical data cleaning tool. When **NODUPKEY** is included in the statement, SAS performs an internal check: as it sorts the data, it compares the values of the key variables defined in the **BY statement**. If sequential observations share identical values across all specified keys, SAS discards the subsequent matching records, retaining only the first instance encountered.

The practical application of **NODUPKEY** is straightforward: it is included directly within the main **PROC SORT** statement, immediately preceding the semicolon or the **BY statement**. This ensures that the de-duplication logic is applied directly alongside the ordering logic. By applying this option, we instruct SAS to produce a new dataset, `data3`, where every value in the `points` column is guaranteed to be unique. This distinction is vital: if two records have the same points value but different rebound values, only one record will survive because the de-duplication is keyed only to `points`.

Executing the following code demonstrates this dual functionality. We are sorting `original_data` by `points` in the default ascending order, while simultaneously purging all subsequent duplicate entries based on that key. This procedure is instrumental in transforming raw, messy input into high-quality, normalized data suitable for critical analytical processing.

```
/*sort by points ascending and remove duplicates*/
proc sort data=original_data out=data3 nodupkey;
by points;
run;

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

Obs	team	points	rebounds
1	A	12	8
2	A	14	7
3	A	20	12
4	A	23	9
5	B	29	4

The resulting output for `data3` is significantly condensed compared to the previous datasets. We can verify that the data remains sorted by `points` in the designated order, but now, the value 12 appears only once, as does 14, 20, and all other key values. The successful removal of

observations that shared identical key values confirms the efficiency and efficacy of the **NODUPKEY** option. This clean dataset is now ready for use in scenarios demanding uniqueness, such as creating lookup tables or performing aggregation functions where each key value must be represented only once.

Combining NODUPKEY with Descending Order Sorting

The utility of **PROC SORT** is not restricted to the default ascending order. Analysts often require data to be prioritized by the largest values first, necessitating a descending order sort. Crucially, the **NODUPKEY** functionality integrates seamlessly with this requirement, allowing for both reverse ordering and de-duplication in a single step. To achieve a descending sort, one simply prefixes the key variable within the **BY statement** with the **DESCENDING** keyword.

When **DESCENDING** is used in conjunction with **NODUPKEY**, the logic remains consistent: the data is first sorted from the highest key value to the lowest, and then, during this process, any subsequent duplicates based on that key are discarded. This feature is particularly valuable when performing operations that require identifying the highest unique values quickly, such as ranking analysis or finding top performers in a dataset. For our sports data example, sorting by `points` in descending order allows us to prioritize the teams with the highest scores, while ensuring that each score is represented only once.

The following code snippet demonstrates the creation of `data4`, which is sorted by the `points` variable from largest to smallest, with all duplicates based on the point value removed. This showcases the comprehensive control **PROC SORT** offers over both the organization and the quality of the output dataset. Note that the record retained for each unique point value will be the one that appeared first in the original unsorted dataset, adjusted to the new sort order.

```
/*sort by points descending and remove duplicates*/
proc sort data=original_data out=data4 nodupkey;
by descending points;
run;

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

Obs	team	points	rebounds
1	B	29	4
2	A	23	9
3	A	20	12
4	A	14	7
5	A	12	8

The final output confirms that `data4` is ordered with the highest point totals appearing first, ensuring the required descending order. Just as importantly, the presence of **NODUPKEY** guarantees that only one record is retained for each distinct value in the `points` column. This successful combination of sorting direction control and data uniqueness enforcement provides a complete and powerful solution for preparing analytical data in SAS, yielding data that is organized, reliable, and free of redundant key entries.

Summary and Strategic Data Management Takeaways

The **PROC SORT** procedure is foundational to data management within the SAS statistical environment. By integrating the **NODUPKEY** option, SAS programmers gain access to an indispensable tool for efficient and automated data cleaning. This procedure allows for the precise ordering of observations while simultaneously guaranteeing that the output dataset contains only unique entries based on the specific criteria defined in the **BY statement**. This capability is absolutely crucial in ensuring the robustness of any subsequent data analysis, as duplicate records can severely distort aggregation, counts, and statistical measures.

When implementing **PROC SORT** with **NODUPKEY**, it is vital to remember the core principle: de-duplication is applied only to the variables listed in the **BY statement**. If you need uniqueness based on all columns, all columns must be included in the **BY statement**. If you only require uniqueness based on a primary identifier (like an ID or, as in our case, `points`), then only those variables should be listed. Furthermore, because SAS retains the *first* matching observation it encounters, the sort order (ascending or descending) can influence which specific record is kept when other variables in the row differ. This strategic decision-making process highlights the sophisticated control this procedure offers over dataset integrity and structure.

Ultimately, whether your goal is to prepare a dataset for advanced statistical models, such as complex statistical models, or simply to generate a clean, summarized report, integrating **PROC SORT** and **NODUPKEY** into your standard SAS programming toolkit ensures high-quality data output. By mastering the nuances of this command, you significantly enhance your capacity to manage large, complex datasets, leading directly to more reliable results and insightful conclusions

in all your data preparation tasks.

Additional Resources for SAS Proficiency

To further enhance your skills in SAS programming and data manipulation, we recommend exploring related tutorials and documentation. Building a comprehensive understanding of SAS procedures, especially those related to data quality and structure, is key to advancing your analytical capabilities.