Remove Duplicates in SAS (With Examples)

Authored by

Mohammed looti

November 1, 2025

In the realm of data management and statistical analysis, data cleaning stands as a foundational requirement. Dealing with duplicate records is perhaps the most frequent challenge faced by analysts, particularly when integrating data from diverse sources or handling large imports. Within the environment of SAS (Statistical Analysis System), the ability to identify and efficiently remove these redundant entries is paramount for maintaining the integrity and accuracy of any subsequent statistical procedure or modeling effort.

The presence of duplicate observations can severely compromise analytical outcomes. They can artificially inflate sample sizes, distort measures of variability, and ultimately lead to skewed statistical results and erroneous conclusions. Fortunately, SAS offers robust, highly optimized procedures designed specifically for data quality management. This comprehensive guide details the most efficient and widely accepted technique for deduplication: harnessing the power of the **proc sort** procedure.

By mastering the specific SAS techniques detailed in the following sections, you will be equipped to handle not only simple scenarios involving identical row duplication but also more complex cases where uniqueness is defined only by a subset of key columns. This expertise is critical for preparing your dataset to meet the stringent requirements of rigorous statistical modeling and reporting.

The Core Tool: Leveraging PROC SORT for Efficient Deduplication

The primary utility function for eliminating duplicate observations in SAS is the **proc sort** procedure. While its conventional role is ordering data based on specified variables, it includes a crucial option tailored for deduplication. This feature makes it significantly more efficient and faster for removing duplicates compared to alternative, often cumbersome methods, such as iterative DATA steps or complex subqueries within PROC SQL.

The cornerstone of successful deduplication using this procedure is the combination of **proc sort** with the **NODUPKEY** option. When this option is invoked, **proc sort** performs an internal check: it discards any observation whose sorting key values are identical to the preceding observation. This mechanism guarantees that only the initial instance of a unique combination of key values is retained in the output dataset.

The fundamental structure for implementing this technique is remarkably simple and highly readable. You must explicitly define the input data using the **DATA** option, specify the new, deduplicated output dataset name using the **OUT** option, and, critically, include the **NODUPKEY** option to trigger the filtering process. The general syntax for removing duplicates across all columns is shown below:

proc sort data=original_data out=no_dups_data nodupkey;

```
by _all_;
run;
```

It is essential to grasp the function of the **BY statement** in this context. The variables listed within the **BY statement** collectively define the unique "key" that **proc sort** uses for comparison. When the **NODUPKEY** option is present, the procedure only examines these specified variables to determine if the record constitutes a duplicate. If the combination of key values matches a preceding record, the current record is immediately discarded.

Setting Up the Demonstration Dataset

To effectively demonstrate the robust deduplication capabilities provided by **proc sort** and the **NODUPKEY** option, we will utilize a small, focused sample dataset. This data contains hypothetical athlete performance metrics and has been deliberately structured to include various types of redundant records, allowing us to clearly see how the procedure handles different deduplication requirements.

The demonstration dataset is named `original_data` and consists of three variables: **team** (a character variable), **position** (a character variable), and **points** (a numeric variable). As you inspect the raw data, you will notice multiple instances where entire rows are repeated, as well as instances where only the key identifying columns are duplicated, while the metric (points) differs.

The following code block is used to construct and display the initial, unprocessed dataset, which serves as our starting point for the data cleaning exercise:

```
/*create dataset*/
data original_data;
input team $ position $ points;
datalines;
A Guard 12
A Guard 20
A Guard 20
A Guard 24
A Forward 15
A Forward 15
A Forward 19
A Forward 28
B Guard 10
B Guard 12
B Guard 12
```

```
B Guard 26
B Forward 10
B Forward 10
B Forward 10
B Forward 19
;
run;

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

The visualization below confirms the presence of multiple identical rows, which must be systematically removed to ensure that our statistical analysis is based on accurate, unique observations:

Obs	team	position	points
1	A	Guard	12
2	A	Guard	20
3	A	Guard	20
4	A	Guard	24
5	A	Forward	15
6	A	Forward	15
7	A	Forward	19
8	A	Forward	28
9	B	Guard	10
10	B	Guard	12
11	B	Guard	12
12	B	Guard	26
13	B	Forward	10
14	B	Forward	10
15	B	Forward	10
16	B	Forward	19

Example 1: Eliminating Identical Rows Across All Variables

The simplest and most common form of data cleaning involves removing observations that are exact replicas across every column. This procedure ensures that every single row in the resulting

dataset is unique in its entirety. To achieve this comprehensive row-level deduplication, we leverage the special, reserved variable list _**all**_ within the **BY statement**.

By specifying `BY _ALL_`, the **proc sort** procedure is instructed to treat the combination of values from every single column as the unique key. Consequently, the **NODUPKEY** option will only retain records where the entire set of values is unique. This methodology provides the most robust means of complete row deduplication, ensuring no two rows are exactly alike.

The code below executes this process, creating a new dataset named `no_dups_data`, which contains only the truly unique records from the input data:

```
/*create dataset with no duplicate rows*/
proc sort data=original_data out=no_dups_data nodupkey;
by _all_;
run;

/*view dataset with no duplicate rows*/
proc print data=no_dups_data;
```

Reviewing the resulting output confirms the successful removal of all exact duplicate rows. Specifically, observations such as `A Guard 20`, `A Forward 15`, `B Guard 12`, and two instances of `B Forward 10` were identified as redundant based on the full row comparison and subsequently dropped. This leaves behind a clean, normalized dataset ready for accurate analysis.

Obs	team	position	points
1	A	Forward	15
2	A	Forward	19
3	A	Forward	28
4	A	Guard	12
5	A	Guard	20
6	A	Guard	24
7	B	Forward	10
8	B	Forward	19
9	B	Guard	10
10	B	Guard	12
11	B	Guard	26

Example 2: Identifying and Removing Duplicates Based on Key Variables

Frequently, the definition of uniqueness in real-world analytical tasks depends on only a subset of identifying variables, rather than the entire row. For example, if we are analyzing a time-series of athlete records, we might consider a record a logical duplicate if the combination of **team** and **position** has already been observed, even if the non-key metrics, such as **points** scored, are different. In this scenario, we are seeking to keep only the first entry for each unique Team/Position pairing.

To execute this targeted, conditional deduplication, we must specifically list only the identifying columns within the **BY statement**. This action clearly defines the key to the **proc sort** procedure, instructing it to only group and check these specified variables for redundancy, ignoring all others during the deduplication phase.

In this specific demonstration, our goal is to eliminate rows that share identical values for both **team** and **position**. The procedure operates by retaining the very first record it encounters for each unique Team/Position combination and systematically dropping all subsequent records that share the same key, regardless of any variation in the **points** value. This is a crucial distinction from the previous example, which required an exact match across the entire row.

The following code block illustrates how to target duplicates solely based on a user-defined key composed of the **team** and **position** columns:

```
/*create dataset with no duplicate rows in team and position columns*/
proc sort data=original_data out=no_dups_data nodupkey;
by team position;
run;

/*view dataset with no duplicate rows in team and position columns*/
proc print data=no_dups_data;
```

The resulting output clearly shows only four unique records, corresponding to the four possible combinations of Team and Position (A Guard, A Forward, B Guard, B Forward). Every other record in the original data, irrespective of its 'points' value, was treated as a duplicate of the first instance found for that specific key combination and consequently removed, achieving our goal of key-based uniqueness:

Obs	team	position	points
1	A	Forward	15
2	A	Guard	12
3	B	Forward	10
4	B	Guard	10

Advanced Considerations and Best Practices

While the combination of **proc sort** with **NODUPKEY** is exceptionally powerful, users must possess a thorough understanding of SAS's inherent observation retention logic, especially when working with keys defined by multiple columns. This knowledge is crucial for ensuring the correct record is preserved when dealing with identical key values but differing non-key data.

By design, the **NODUPKEY** option always retains the **first observation** encountered in the data stream for any unique key combination, dropping all subsequent observations that share that key. If your original dataset's order is arbitrary, or if your requirement is to retain a specific record (e.g., the entry associated with the highest score or the latest date), you must ensure the data is sorted appropriately *prior* to the deduplication step.

Consider implementing the following best practices when constructing your deduplication strategy:

Retaining Specific Records: If the goal is to keep the observation associated with the maximum or minimum value among all records sharing the same key, you must sort by the key variables first, and then include the retention variable with the appropriate sorting direction. For instance, using `BY team position descending points;` ensures that the record with the highest points for that unique team/position combination is placed first in the temporary sort group, guaranteeing its retention when **NODUPKEY** is applied.

Handling Missing Values: Analysts must be aware of how SAS handles missing data during the sorting process. By default, both character and numeric missing values are treated as the lowest possible values. If missingness is a component of your duplication key, confirm that this default behavior aligns with your specific data cleaning objectives, as it directly influences which record is considered the "first" and therefore retained.

Memory Management: For extremely large datasets that exceed available physical memory, the **proc sort** operation can become resource-intensive. While it is generally the most efficient method for straightforward deduplication, if severe memory constraints are encountered, the **PROC SQL DISTINCT** clause offers a viable, often less memory-demanding, alternative, although it lacks the granular control over retention offered by pre-sorting with `NODUPKEY`.

Conclusion and Further Reading

The **proc sort** procedure, when correctly paired with the powerful **NODUPKEY** option, establishes itself as the most efficient and flexible mechanism for achieving data deduplication within SAS. This method simplifies the complex data preparation phase, whether the requirement is to remove observations that are exact replicas or to eliminate redundancy based only on a specifically defined set of identifying variables.

By precisely defining the sorting key through the **BY statement**, data analysts retain complete control over the criteria for uniqueness. This rigorous approach ensures that the resulting dataset is reliably clean, highly accurate, and optimally prepared for advanced statistical applications and modeling efforts.

Additional Resources

To further enhance your mastery of data manipulation and preparation techniques in SAS, consider exploring the following related tutorials:

Tutorial on Merging Data in SAS using the DATA Step.
Guide to Conditional Logic with IF-THEN/ELSE Statements.
Using PROC MEANS for Summary Statistics.