How to Easily Remove Duplicate Records in SAS

Authored by

stats writer

December 1, 2025

Data quality is paramount in successful statistical analysis and business intelligence. One of the most common issues encountered when preparing raw data is the presence of redundant or duplicate records. In the SAS programming environment, the task of identifying and eliminating these duplicate observations is streamlined and highly efficient, relying primarily on the powerful **PROC SORT** procedure. This procedure not only organizes your data based on specified variables but also provides specialized options designed specifically for cleansing tasks. Understanding how to leverage these options is essential for any SAS user seeking to maintain data integrity and accuracy.

The core mechanism for de-duplication in SAS requires two fundamental steps. First, the input dataset must be sorted using the PROC SORT statement. Sorting groups identical records together, making them easy targets for removal. Second, specialized options--namely NODUPKEY and NODUPRECS--are applied within the procedure to flag and suppress the output of redundant observations into the new output dataset. This methodical approach ensures that your resulting data structure is clean and ready for immediate statistical processing or reporting, maximizing computational efficiency and the trustworthiness of subsequent analyses.

While simple duplication removal covers most cases, advanced scenarios sometimes require keeping a specific instance of a duplicate record, perhaps the first entry, the last entry, or the record associated with the highest value in a key metric. For these complex situations, the process can be enhanced using tools like the **DATA step** in conjunction with the **FIRST.** and **LAST.** automatic variables, or by integrating the RETAIN statement. This tutorial will provide a comprehensive guide, starting with the fundamental PROC SORT syntax and progressing through practical examples to handle diverse duplication challenges.

The primary tool for this operation is the **proc sort** statement in SAS, which allows for the swift generation of a new dataset purged of redundant rows.

This critical procedure utilizes the following generalized syntax structure for performing de-duplication:

```
proc sort data=original_data out=no_dups_data nodupkey;
by _all_;
run;
```

It is essential to understand the roles of the included arguments. The **by** argument is fundamental, as it explicitly instructs PROC SORT which variables (columns) to analyze when determining if a record constitutes a duplicate. If the values in the specified **by** variables match across multiple rows, those rows are considered duplicates for the purpose of the de-duplication process.

Understanding the Importance of Data Cleansing

Before diving into the code, it is vital to appreciate why data cleansing, particularly the removal of duplicates, holds such significant weight in data analysis. Duplicate records can arise from numerous sources, including errors in data entry, merging datasets from disparate systems, or faulty data pipelines. If these duplicates are left unchecked, they can lead to severely skewed results. For example, counting duplicate customer IDs could inflate sales figures, or duplicated patient records could distort epidemiological statistics, leading to incorrect strategic decisions or misinformed conclusions.

Effective data cleansing ensures the principle of "one record, one observation," which is critical for statistical integrity. Furthermore, working with a clean, de-duplicated dataset significantly reduces the memory footprint and processing time required for complex procedures. When dealing with big data environments, minimizing unnecessary computational load translates directly into faster turnaround times and lower operational costs. Therefore, mastering the efficient removal of duplicates using SAS procedures is a foundational skill for data management professionals.

The methods we employ in SAS are designed to provide control over the definition of a duplicate. We can define a record as a duplicate if **every single variable** matches across two rows (using `_ALL_`), or we can be highly selective, defining duplication based only on a critical subset of fields, such as identifiers like `customer_id` and `transaction_date`. This flexibility is what makes PROC SORT with its de-duplication options an invaluable tool in the data scientist's toolkit.

Key PROC SORT Options: NODUPKEY vs. NODUPRECS

When executing the PROC SORT procedure for de-duplication, you have a choice between two primary options: NODUPKEY and NODUPRECS. While both serve the ultimate goal of removing redundancy, they operate on slightly different definitions of what constitutes a duplicate observation, making the selection process dependent on your data cleaning objective.

The NODUPKEY option is the most commonly utilized method. It instructs PROC SORT to examine only the variables specified in the **BY** statement. If two or more records have identical values for all variables listed in the **BY** statement, NODUPKEY ensures that only the first occurrence of that unique combination of key variables is written to the output dataset. Any subsequent records sharing the same key values are discarded. This is ideal when you want to enforce uniqueness based on primary identifiers, such as ensuring each customer ID appears only once.

In contrast, the NODUPRECS option is much stricter. It requires that two records be **perfectly identical across all variables** in the dataset--including those not listed in the **BY** statement--for one to be considered a duplicate. If you specify `BY _ALL_` along with NODUPKEY, the result is the

same as using NODUPRECS, as the key variables encompass the entire record. However, if you omit the **BY** statement entirely and use NODUPRECS, PROC SORT will check all columns implicitly. For clarity and control, using NODUPKEY with explicit **BY** variables (or `_ALL_`) is generally the preferred best practice.

The Essential Syntax for Duplicate Elimination

To successfully implement duplicate removal in SAS, a precise understanding of the necessary components within the PROC SORT statement is required. Every de-duplication routine must specify the source data, the destination for the cleaned data, and the criteria for sorting and eliminating duplicates.

The standard syntax includes the `DATA=` option to specify the input dataset containing duplicates, and the `OUT=` option, which names the new dataset that will contain only the unique records. Crucially, one of the de-duplication options, typically NODUPKEY, must be included directly after the `OUT=` option within the PROC SORT statement.

The subsequent **BY** statement dictates the sorting order and, more importantly when using NODUPKEY, which variables constitute the key definition of uniqueness. If you list `BY Variable_A Variable_B;`, the procedure will only remove records where both `Variable_A` and `Variable_B` match across observations. If you use `BY _ALL_;`, the sort key includes every variable in the input dataset, ensuring that only rows that are absolutely identical are removed.

Preparing the Sample Dataset for Demonstration

To illustrate these concepts effectively, we will work with a simple, common scenario involving performance data for two hypothetical basketball teams. This sample data contains several intentional duplicate entries that we will systematically remove using the methods discussed. The following code demonstrates the creation and initial view of the `original_data` dataset.

This dataset includes columns for the `team`, the player's `position`, and their accrued `points`. Notice that several rows, such as Team A, Guard, 20, and Team B, Guard, 12, are repeated multiple times. Our goal is to apply different de-duplication strategies to see how the output changes based on whether we check all columns or specific columns.

The data is defined using the **DATA step** and **DATALINES**, a common pattern for defining static test data within SAS programming. We then use **PROC PRINT** to display the contents of the unsorted dataset.

```
/*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;
```

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 Duplicates Across All Columns

The simplest form of duplicate removal involves ensuring that no two rows are identical across every single variable. This is often the first step in cleaning raw data to remove accidental, full-row redundancies. To achieve this, we instruct PROC SORT to use all existing columns as the key for identifying duplicates.

We accomplish this by specifying the `BY _ALL_` statement. When combined with the NODUPKEY option, PROC SORT evaluates every variable (`team`, `position`, and `points`) simultaneously. If it encounters a subsequent observation where all three values match the first occurrence, the subsequent observation is discarded. The resulting `no_dups_data` dataset will only contain unique rows.

The following SAS code executes this full-record de-duplication process. Notice the clear separation between the sorting and de-duplication logic and the subsequent verification step using **PROC PRINT** to display the outcome.

```
/*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;

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

Upon reviewing the output dataset, we observe that the process has successfully identified and removed five redundant rows from the original data structure. Specifically, duplicate observations like 'A Guard 20', 'A Forward 15', 'B Guard 12', and 'B Forward 10' (which appeared three times) have been reduced to single, unique instances, providing a clean foundation for subsequent analysis of player performance.

Example 2: Targeting Duplicates Based on Specific Columns

Often, the requirement is not to eliminate strictly identical rows, but rather to ensure uniqueness based on a subset of key identifiers, while allowing other variables (like timestamp or score) to vary. This is achieved by specifically defining the key variables in the **BY** statement, ignoring all other variables in the dataset during the de-duplication check.

For instance, we might want to ensure that each unique combination of `team` and `position` appears only once in our dataset, regardless of the `points` scored in that observation. This scenario requires using the NODUPKEY option and specifying `team` and `position` in the **BY** statement. When a match is found on these two columns, the record is considered a duplicate, and only the first instance encountered during the sort process is retained.

The key takeaway here is that since we are sorting by `team` and `position`, the record that is

retained will be the one with the lowest value in the `points` column (assuming default ascending sort order), because PROC SORT keeps the first record in the group. If retaining the record with the highest score is necessary, an additional step involving sorting by the metric variable (e.g., `points`) in descending order (`DESCENDING points`) must be performed prior to the final de-duplication step.

```
/*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;
```

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

Advanced Considerations: Keeping Specific Duplicate Instances

While NODUPKEY effectively removes subsequent duplicates, it always retains the first record encountered during the sort process. In real-world data management, you frequently need to apply conditional logic to decide which record to keep--perhaps the most recent transaction, the highest sale price, or the record with the most complete information. This is where the **DATA step** provides superior flexibility compared to the simpler PROC SORT approach.

The core strategy involves using PROC SORT only to group the data, followed by a **DATA step** that utilizes the automatic variables **FIRST.** and **LAST.**. When data is sorted by grouping variables (e.g., `BY customer_ID`), SAS automatically creates boolean variables `FIRST.customer_ID` and `LAST.customer_ID`. These are set to 1 for the first and last observation within that group, respectively. To retain the oldest record (the first record encountered), you use the condition `IF FIRST.customer_ID;`.

For complex scenarios, such as keeping the record with the maximum value for a specific variable among all duplicates, you must first sort the data in descending order based on that metric variable

(e.g., `PROC SORT DATA=... BY customer_ID DESCENDING metric_score;`). Then, applying the `IF FIRST.customer_ID;` condition will ensure that the retained observation is the one with the highest `metric_score`, as it appears first in the sorted group. This advanced control is essential for maintaining accuracy when handling updated records or conflicting data entries. The RETAIN statement is generally reserved for carrying values forward across observations within a group, but the combination of sorting and `FIRST.`/`LAST.` variables is the standard for conditional de-duplication.

Summary of Best Practices and Key Takeaways

Mastering duplicate removal in SAS hinges on selecting the appropriate tool and correctly defining the key variables. Here is a summary of best practices to ensure your data cleaning processes are robust and reliable:

Always Sort First: De-duplication using NODUPKEY requires the input data to be sorted by the key variables. This is not just a prerequisite but is the mechanism that groups the duplicates together for easy identification.

Define Your Key Precisely: The **BY** statement dictates the definition of a duplicate. Using `_ALL_` is appropriate only when rows must be entirely identical. For uniqueness based on identifiers, explicitly list only the identifying variables (e.g., `BY ID DATE;`).

Choose the Right Option: Use NODUPKEY for defining uniqueness based on the **BY** variables. Reserve NODUPRECS for situations where you must guarantee that the entire record is unique, irrespective of the **BY** statement (if present).

Use the DATA Step for Conditional Retention: When you need to keep the newest, oldest, or highest-scoring record among duplicates, combine PROC SORT (possibly with a `DESCENDING` option) and the **DATA step** using `IF FIRST.group_variable;`.

By systematically applying these techniques, data professionals can maintain high standards of data quality, leading to more accurate analyses and trustworthy reporting outcomes within the SAS environment.

Further Resources for SAS Data Manipulation

To continue honing your data preparation skills in SAS, consider exploring tutorials on related operations that complement duplicate removal, such as merging and subsetting data. These procedures are frequently required in conjunction with data cleansing to prepare comprehensive, high-quality analytical datasets.

The following resources offer additional guidance on common SAS data manipulation tasks:

How to efficiently merge multiple datasets in SAS using **PROC SQL**.
Techniques for subsetting data based on conditional criteria using the **WHERE** statement in various procedures.
Strategies for handling missing values and data imputation in the SAS environment.