How to Easily Remove Duplicate Records in SAS Datasets

Authored by

stats writer

November 22, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Remove Duplicate Records in SAS Datasets*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=99663

ARABPSYCHOLOGY.COM

Data integrity is paramount in statistical analysis, and few issues are as common or challenging as handling duplicate records within a Dataset. In SAS, users are provided with powerful, flexible tools to identify and eliminate these redundant observations, ensuring that analyses are based on accurate and unique data points. The most frequently employed methods revolve around using the **PROC SORT** statement combined with specific options like **NODUPKEY** or **NODUPRECS**, or utilizing the flexibility of PROC SQL with the DISTINCT keyword.

Understanding which method to apply depends heavily on whether you need to remove observations that are identical across all variables, or just observations that share identical values based on a subset of key variables. Both PROC SORT and PROC SQL offer robust, high-performance solutions for achieving a clean, unique dataset.

The Importance of Removing Duplicate Observations

Duplicate records can severely skew statistical results, leading to inflated sample sizes, biased parameter estimates, and incorrect inferences. This is particularly problematic in areas like market research, financial reporting, and clinical trials where each observation should represent a unique entity or event. The process of deduplication in SAS ensures that downstream analytical procedures operate on a foundation of clean data, thereby validating the integrity of the entire project.

When we discuss "duplicates," it is vital to define what constitutes a duplicate within a specific context. A duplicate can be an entire row that matches another row perfectly (a full record duplicate), or it can be a row that matches another row only on certain identifying variables (a key duplicate). SAS provides tools to handle both scenarios efficiently, predominantly through the use of the sorting procedure.

Method 1: Leveraging PROC SORT for Duplicate Removal

The **PROC SORT** procedure is perhaps the most common and powerful way to manage duplicate records in SAS. While its primary function is to order the observations in a dataset based on one or more variables specified in the **BY** statement, it has the added capability of detecting and eliminating duplicates when used in conjunction with specific options. This procedure not only organizes your data but also cleans it in a single, efficient step.

To use **PROC SORT** for deduplication, you must specify the variables that define uniqueness using the **BY** statement. Once the key variables are defined, you append the appropriate duplicate-removal option. The choice between **NODUPKEY** and **NODUPRECS** is critical and determines the scope of the deduplication process, as detailed in the next section.

ARABPSYCHOLOGY.COM

Understanding NODUPKEY vs. NODUPRECS

When employing PROC SORT, it is essential to distinguish between the two available options for removing duplicates:

NODUPKEY: This option instructs SAS to remove any observation whose values for the variables listed in the **BY** statement are duplicates of the preceding observation. If two or more records share the same values for the sort keys, only the first occurrence is kept. Note that the remaining variables (those not in the **BY** statement) do not have to match for the record to be flagged as a duplicate and removed. This is useful when you want to enforce uniqueness based on identifying fields (e.g., ID or Date).
NODUPRECS: This stricter option instructs SAS to remove only those observations that are exact duplicates of a preceding observation across **all** variables in the dataset, including the **BY** variables and any other non-key variables. If you do not include a **BY** statement, NODUPRECS operates on the entire record. If a **BY** statement is present, it removes only full-record duplicates within each BY group.

The subsequent examples will focus on the highly flexible **NODUPKEY** option, demonstrating how it enforces uniqueness based on the specified criteria.

Step-by-Step Example: Preparing the Sample Data

To illustrate the duplicate removal process using PROC SORT, we will begin by creating a sample dataset that intentionally contains several duplicate observations. This dataset tracks team performance metrics, specifically `team`, `points`, and `rebound` values. Note that observations 1, 2, and 3 are exact duplicates, as are observations 6 and 7, and several within Team B.

The following example shows the setup of our initial dataset in SAS:

```
/*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
```

ARABPSYCHOLOGY.COM

```
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

ARABPSYCHOLOGY.COM

As displayed in the output, the initial dataset contains 14 observations, many of which are redundant when considering the combination of all three variables.

Intermediate Step: Sorting Without Deduplication

Before demonstrating duplicate removal, it is helpful to see how PROC SORT operates purely as a sorting tool. Suppose we sort the observations in ascending order based only on the value in the **points** column. We create a temporary dataset called `data2` to hold the sorted output.

```
/*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

ARABPSYCHOLOGY.COM

The resulting dataset `data2` still contains all 14 original observations. While the records are correctly ordered based on the **points** column, the redundant rows remain present. This demonstrates the necessity of adding a deduplication option to the procedure call.

Implementing PROC SORT with NODUPKEY (Ascending Sort)

To efficiently sort the observations based on the values in the **points** column and simultaneously remove all duplicate values identified by this key, we incorporate the **NODUPKEY** option into the PROC SORT statement. Since our **BY** statement specifies only `points`, SAS will keep only the first observation for every unique value found in the `points` column, regardless of the values in the `team` or `rebound` columns.

```
/*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 dataset, `data3`, now contains only 8 observations. This is because all duplicate values for the `points` variable have been successfully collapsed, leaving a dataset where every row represents a unique score value. The observations are also correctly sorted in ascending order.

Applying NODUPKEY with Descending Order

The NODUPKEY option is flexible and works seamlessly with sorting modifiers. We can easily modify the code to sort the data in descending order (largest to smallest) while still enforcing uniqueness based on the `points` column. To achieve this, we simply add the **DESCENDING** keyword before the variable name in the **BY** statement.

Crucially, because NODUPKEY retains the first observation encountered for a key, changing the sort order (ascending vs. descending) may change which specific record is retained if there are other variable differences among the duplicate key groups. For instance, if records A and B both have 12 points, the ascending sort keeps A, but the descending sort might keep B if B appeared first in the source data after the descending sort criteria was applied.

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

ARABPSYCHOLOGY.COM

```
/*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 output `data4` confirms that the data is sorted starting with the highest point values, and the total count remains 8 unique observations based on the `points` key.

Method 2: Eliminating Duplicates Using PROC SQL DISTINCT

An equally powerful and often more intuitive approach for users familiar with database query language is utilizing PROC SQL. The DISTINCT keyword, when placed immediately after the `SELECT` statement, ensures that only unique rows are returned in the result set. Unlike PROC SORT, DISTINCT always evaluates the uniqueness across all variables selected in the statement, effectively acting like a global **NODUPRECS**.

If you select all columns (using `SELECT DISTINCT *`), PROC SQL will eliminate any row that is an exact duplicate of another row. This is the simplest way to enforce complete record uniqueness without needing a separate sorting step.

Example of using PROC SQL to remove exact duplicates:

```
/*Remove full record duplicates using SQL*/
proc sql;
create table data_sql as
select distinct *
from original_data;
quit;

/*view unique dataset*/
proc print data=data_sql;
```

ARABPSYCHOLOGY.COM

Choosing the Right Approach for Your Data Integrity Needs

The choice between PROC SORT and PROC SQL depends on the specific requirements of the deduplication task and your familiarity with the respective procedure syntax. Both methods are highly efficient for typical dataset sizes in SAS:

If you need to define uniqueness based on a subset of key variables (e.g., keeping only one record per ID, regardless of other variable discrepancies), **PROC SORT with NODUPKEY** is the precise tool. It also gives you explicit control over which record is kept (first or last) via the sort order.
If you only need to ensure that no two rows are completely identical across all variables, **PROC SQL with DISTINCT** offers the cleanest, most concise syntax. Alternatively, **PROC SORT with NODUPRECS** achieves the same result, especially if you also need the data sorted afterward.

In complex scenarios, especially when dealing with very large datasets or requiring complex conditional deduplication (e.g., keeping the record with the latest date among duplicates), PROC SORT often proves more versatile due to its interaction with the **BY** statement and the implied ordering logic.

Mastering these techniques ensures that your data preparation stage is robust, leading to reliable and trustworthy analytical results.

ARABPSYCHOLOGY.COM