

How to Rank Data in SAS with PROC RANK: A Step-by-Step Guide

Authored by
stats writer

December 1, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Rank Data in SAS with PROC RANK: A Step-by-Step Guide*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=103024>

The PROC RANK procedure in SAS is one of the most fundamental tools available for data preparation and statistical analysis. Its primary function is to assign a rank to each observation within a data set based on the values of one or more specified numeric variables. This procedure is invaluable when analysts need to convert continuous data into ordinal measurements, facilitating comparisons, non-parametric statistical testing, or preparation for grouping algorithms.

When using PROC RANK, users can specify how ties are handled--a crucial feature when multiple observations share the exact same value. By default, SAS assigns the mean rank to tied values, but options exist for low rank (minimum) or high rank (maximum) assignments. Furthermore, the procedure can efficiently generate various rank-based statistics, including normal scores and related distributional values, making it highly versatile for different stages of a data workflow.

Understanding the proper syntax and control statements associated with PROC RANK is essential for leveraging its full power. This guide details the four most common applications of this procedure, providing clear examples and explanations for each use case within the SAS programming environment.

Understanding the Core Syntax of PROC RANK

The syntax for the PROC RANK procedure is straightforward yet powerful, revolving around four key statements: the PROC statement itself, and the required VAR and RANKS statements, often paired with the DATA and OUT options.

The main statement, **PROC RANK**, initiates the procedure and typically includes the **DATA=** option to specify the input data set and the **OUT=** option to name the new data set that will contain the original variables plus the newly calculated rank variables. This output data set is where the results are stored for subsequent analysis or reporting.

The **VAR** statement is mandatory; it lists the numeric variables for which ranks are to be calculated. If multiple variables are listed, SAS calculates ranks independently for each variable. Following the **VAR** statement is the **RANKS** statement, which specifies the names of the new variables that will store the calculated ranks. It is critical that the number of variables listed in the **RANKS** statement exactly matches the number of variables listed in the **VAR** statement, maintaining the order of association.

Optional but highly useful statements include the **BY** statement for performing calculations within defined groups, and the **GROUPS** statement for dividing the data into discrete rank groups, such as quartiles or percentiles. These options allow for sophisticated data manipulation tailored to specific analytical needs, such as non-parametric tests or descriptive group summaries.

The Four Primary Applications of PROC RANK

The utility of PROC RANK can be categorized into four primary scenarios, each addressing a common data preparation requirement:

You can use **PROC RANK** in SAS to calculate the rank for one or more numeric variables. Below are the four principal methods used in analytical practice:

Here are the four most common ways to use this procedure:

Method 1: Rank One Variable

This is the simplest application, where a single numeric variable is ranked across all observations in the data set. The resulting rank variable is added to the output data set.

```
proc rank data=original_data out=ranked_data;  
var var1;  
ranks var1_rank;  
run;
```

Method 2: Rank One Variable by Group

When using the **BY** statement, ranking is performed separately within each distinct group defined by the BY variable (which must be sorted prior to execution). This is critical for comparing observations relative to their peers within specific categories.

```
proc rank data=original_data out=ranked_data;  
var var1;  
by var2;  
ranks var1_rank;  
run;
```

Method 3: Rank One Variable into Percentiles

The **GROUPS=n** option transforms the continuous rank into discrete ordinal categories, such as percentiles (GROUPS=100) or quartiles (GROUPS=4). This method is widely used for creating risk bands or performance tiers.

```
proc rank data=original_data groups=4 out=ranked_data;  
var var1;  
ranks var1_rank;
```

run;

Method 4: Rank Multiple Variables

This allows for the concurrent ranking of several independent variables within a single procedure call. SAS processes each variable separately, adding a new rank column for each variable specified.

```
proc rank data=original_data out=ranked_data;  
var var1 var2;  
ranks var1_rank var2_rank;  
run;
```

To illustrate these methods clearly, the following examples utilize a sample sports performance data set containing team identifiers, points scored, and rebounds collected:

```
/*create dataset*/  
data original_data;  
input team $ points rebounds;  
datalines;  
A 25 10  
A 18 4  
A 18 7  
A 24 8  
B 27 9  
B 33 13  
B 31 11  
B 30 16  
;  
run;
```



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

Obs	team	points	rebounds
1	A	25	10
2	A	18	4
3	A	18	7
4	A	24	8
5	B	27	9
6	B	33	13
7	B	31	11
8	B	30	16

Example 1: Generating Simple Ranks

This example demonstrates the fundamental use case of `PROC RANK` by calculating the rank of the **points** variable across the entire data set. We create a new variable named **points_rank**, where the lowest score receives the lowest rank (1, 2, 3, etc.).

When running a simple rank procedure, it is important to observe how tied values are treated. By default, `PROC RANK` uses the **MEAN** option for tie resolution. This means if two observations are tied for the first and second lowest positions, they are both assigned the average of those ranks, which is $(1 + 2) / 2 = 1.5$. This ensures that the sum of the ranks remains consistent even with ties.

The code below executes this basic ranking and then uses **PROC PRINT** to display the resulting data set, allowing us to inspect the new **points_rank** variable alongside the original data.

```
/*rank points scored by team*/  
proc rank data=original_data out=ranked_data;  
var points;  
ranks points_rank;  
run;  
  
/*view ranks*/  
proc print data=ranked_data;
```

Obs	team	points	rebounds	points_rank
1	A	25	10	4.0
2	A	18	4	1.5
3	A	18	7	1.5
4	A	24	8	3.0
5	B	27	9	5.0
6	B	33	13	8.0
7	B	31	11	7.0
8	B	30	16	6.0

As illustrated in the output, the two rows that scored 18 points were tied for the lowest values, receiving a rank of 1.5. The next lowest value received rank 3. Note that the sum of ranks (1+2+3+4+...) is preserved despite the mean assignment. This method ensures statistical validity when transforming data for non-parametric tests like the Wilcoxon Rank Sum test.

It is often necessary to rank values in descending order, meaning the highest score receives the rank of 1. This is achieved by simply adding the **DESCENDING** option to the **PROC RANK** statement. The following code demonstrates this modification, which is particularly useful in competitive scenarios where higher values are superior:

```
/*rank points scored by team in descending order*/  
proc rank data=original_data descending out=ranked_data;  
var points;  
ranks points_rank;  
run;
```

```
/*view ranks*/  
proc print data=ranked_data;
```

Obs	team	points	rebounds	points_rank
1	A	25	10	5.0
2	A	18	4	7.5
3	A	18	7	7.5
4	A	24	8	6.0
5	B	27	9	4.0
6	B	33	13	1.0
7	B	31	11	2.0
8	B	30	16	3.0

Observe in this output that the team with the highest score (33 points) now holds rank 1, and the tied scores of 18 points are assigned the average of the two lowest ranks (7 and 8), resulting in a rank of 7.5. The **DESCENDING** option simply reverses the order in which the ranks are assigned.

Example 2: Ranking by Categorical Groups

In many analytical tasks, rankings must be performed relative to a specific category or group rather than across the entire data set. For instance, we may wish to know how a player ranks in scoring only among members of their own team (Team A or Team B).

To achieve this subgroup ranking, we utilize the **BY** statement within PROC RANK. Crucially, before using the **BY** statement, the input data set must be sorted by the grouping variable (in this case, **team**). SAS will then reset the ranking count (starting again at 1) for each new category it encounters.

The code below first ensures the data is sorted by the **team** variable (though the sample data was already ordered, this step is mandatory in general practice) and then applies the rank procedure, calculating **points_rank** separately for Team A and Team B.

```
/*rank points scored, grouped by team*/
proc rank data=original_data out=ranked_data;
var points;
by team;
ranks points_rank;
run;

/*view ranks*/
proc print data=ranked_data;
```

Obs	team	points	rebounds	points_rank
1	A	25	10	4.0
2	A	18	4	1.5
3	A	18	7	1.5
4	A	24	8	3.0
5	B	27	9	1.0
6	B	33	13	4.0
7	B	31	11	3.0
8	B	30	16	2.0

The resulting output clearly shows the rank resets. For Team A, the lowest score of 18 (tied) receives a rank of 1.5, and the highest score of 25 receives rank 4. When the procedure moves to Team B, the ranking process begins anew, with the lowest score of 27 receiving rank 1 and the highest score of 33 receiving rank 4. This grouped ranking is essential for intra-group comparisons and normalization.

Example 3: Calculating Quartiles and Percentile Groups

Instead of generating a continuous rank (1, 2, 3, ...), analysts often prefer to categorize data into discrete groups or bins, such as quartiles, deciles, or percentiles. This transformation is accomplished using the **GROUPS=n** option in PROC RANK.

When **GROUPS=n** is specified, the procedure divides the data into 'n' groups of approximately equal size based on the distribution of the **VAR** variable. The resulting rank variable is an integer ranging from 0 to n-1. For example, if **GROUPS=4** is used (creating quartiles), the output ranks will be 0, 1, 2, and 3.

The following code demonstrates ranking the **points** variable into four groups (quartiles). This is useful for segmenting performance data, where players in Group 3 represent the top 25% of scorers and players in Group 0 represent the bottom 25%.

```
/*rank points into quartiles*/
```

```
proc rank data=original_data groups=4 out=ranked_data;
```

```
var points;
```

```
ranks points_rank;
```

```
run;
```

```
/*view ranks*/
```

```
proc print data=ranked_data;
```

Obs	team	points	rebounds	points_rank
1	A	25	10	1
2	A	18	4	0
3	A	18	7	0
4	A	24	8	1
5	B	27	9	2
6	B	33	13	3
7	B	31	11	3
8	B	30	16	2

In this quartile output, the rows with the lowest point values fall into group **0** (the first quartile), while those with the highest scores are assigned to group **3** (the fourth quartile). If you wished to assign values into deciles (10 equal groups), you would simply specify **groups=10**. This feature is powerful for converting continuous variables into discrete, ordinal factors for use in predictive models or simple classification reports.

Example 4: Handling Multiple Variables

When analyzing large data sets, it is efficient to calculate ranks for several independent variables simultaneously using a single PROC RANK step. This reduces code complexity and processing time compared to running the procedure multiple times.

To rank multiple variables, you simply list all input variables in the **VAR** statement and then list the corresponding output rank variable names in the **RANKS** statement, ensuring the order is maintained. SAS processes the ranking for each variable independently of the others.

In this final example, we calculate the ranks for both **points** and **rebounds**, creating two new variables: **points_rank** and **rebounds_rank**.

```
proc rank data=original_data out=ranked_data;  
var points rebounds;  
ranks points_rank rebounds_rank;  
run;
```

Obs	team	points	rebounds	points_rank	rebounds_rank
1	A	25	10	4.0	5
2	A	18	4	1.5	1
3	A	18	7	1.5	2
4	A	24	8	3.0	3
5	B	27	9	5.0	4
6	B	33	13	8.0	7
7	B	31	11	7.0	6
8	B	30	16	6.0	8

The resulting output data set now includes four core numeric variables: the original **points** and **rebounds**, and their respective rank variables. Notice that the rank values are independent; the player with the highest points score (rank 1.0 in points) might not have the highest rebound total (which is rank 8.0 in rebounds). This simultaneous ranking capability ensures that analysts can quickly transform broad numerical metrics into standardized ordinal measures across the entire data set.

Mastering PROC RANK is crucial for any user of SAS who deals with non-parametric statistics or requires data normalization based on relative standing rather than absolute values. Its flexibility in handling ties, grouping variables, and creating discrete categories makes it a cornerstone of data analysis within the environment.

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