Learning SAS: A Practical Guide to Ranking Data with PROC RANK

Authored by

Mohammed looti

October 31, 2025

The **PROC RANK** procedure in **SAS** is a foundational utility utilized by analysts and programmers to compute the **rank** of observations based on the values of one or more specified **numeric variables** within a dataset. This powerful transformation is essential for a variety of **data analysis** activities, including assessing relative performance, normalizing data distributions, or preparing data for non-parametric statistical tests where the relative order of values is more important than their raw magnitude.

Understanding the effective utilization of **PROC RANK** is crucial for transforming raw data into insightful **ordinal measures**. This comprehensive guide will delve into the procedural mechanics of this SAS tool, exploring the four most common and practical applications. We will provide detailed explanations and clear, executable examples to help you master its functionality and integrate ranking techniques seamlessly into your data processing workflow.

Understanding PROC RANK in SAS

PROC RANK operates by assigning a positional **rank** to every observation based on the magnitude of the values in the specified **numeric variables**. By default, the procedure executes an ascending sort, assigning rank 1 to the observation with the smallest value, rank 2 to the next smallest, and so forth, up to the maximum number of observations. This transformation is highly valuable in scenarios where the relative standing of an observation is the primary metric of interest, rather than its absolute measurement.

The core utility of **PROC RANK** extends well beyond simple sequential ordering. It is frequently employed to normalize non-normally distributed data, to identify potential outliers based on extreme relative positions, or to segment datasets into predefined groups such as **percentiles** or **quantiles**. Once created, these ranked variables can serve as direct inputs for advanced statistical procedures, including non-parametric tests like the Wilcoxon Rank-Sum test, or they can be used directly for informative reporting.

The fundamental structure for executing **PROC RANK** requires specifying the input dataset, defining an output dataset to store the results, identifying the variables targeted for ranking, and assigning names for the newly created rank variables. Furthermore, SAS provides optional statements that enable more complex ranking methodologies, such as computing ranks separately within distinct categorical groups or producing categorical groupings like **quantiles** instead of continuous ranks.

Setting Up Your Example Dataset

To effectively illustrate the diverse functionalities and syntax options available within **PROC RANK**, we will first construct a foundational sample dataset. This dataset simulates performance data for multiple teams, including metrics such as points scored and rebounds achieved. Using this sports-

related context allows us to apply and observe the ranking methods in a clear and practical analytical setting.

The following **DATA step** code demonstrates the process for creating and populating this dataset within the **SAS** environment. We utilize the **INPUT statement** to define the structure and format of our variables (team, points, and rebounds), followed by the **DATALINES statement**, which embeds the raw data directly into the program for immediate execution.

Once the dataset, named `original_data`, is successfully generated, we employ the **PROC PRINT** procedure. This step is crucial for verifying that the data has been loaded correctly and is structured as expected, confirming that the input is ready for subsequent ranking operations and statistical analysis.

```
/*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

Method 1: Ranking a Single Variable

The most elementary, yet frequently used, application of **PROC RANK** involves computing the **rank** for a single **numeric variable** across all observations in the dataset. This process assigns an ordinal position to each data point, typically starting with rank 1 for the lowest value and increasing sequentially for higher values, thereby providing a clear picture of the variable's distribution relative to the overall population.

In the ensuing example, we aim to generate a new output variable named `points_rank`, which will contain the **rank** of the `points` scored by each team entry within our example dataset. To achieve this, the **VAR statement** is utilized to explicitly identify the input variable targeted for ranking (`points`), while the mandatory **RANKS statement** specifies the name (`points_rank`) of the new variable that will store the computed rank values in the output dataset.

```
/*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

A critical consideration when performing ranking is the handling of tied values, which occur when multiple observations share the same value for the ranked variable. By default, **PROC RANK** employs a specific tie-breaking technique: it assigns the **mean rank** to all tied values. For instance, if two values are tied for the theoretical ranks of 5 and 6, both will receive a calculated rank of 5.5, ensuring the statistical integrity of the rank sum remains preserved. Should your analytical needs require the largest values to be assigned the lowest ranks (a common necessity in competitive scoring, where a higher score is good but a lower rank is desired), you can easily reverse the ranking direction by including the **DESCENDING statement** within the procedure call.

```
/*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

Method 2: Ranking Within Groups

In advanced **data analysis**, it is frequently necessary to calculate ranks not globally across the entire dataset, but locally within distinct, predefined subgroups. For example, if you have performance data across several different teams, you might need to determine the internal **rank** of each player only relative to their own team members. **PROC RANK** seamlessly handles this requirement through the inclusion of the **BY statement**.

When the **BY statement** is incorporated into the procedure, **PROC RANK** instructs SAS to calculate an entirely separate set of ranks for every unique value present in the **BY variable**. Crucially, the rank counter automatically resets to 1 at the beginning of each new group, ensuring that the resulting ranks are localized measures of relative standing. This capability is invaluable for comparative analysis, allowing analysts to assess performance within homogeneous categories.

The following code snippet demonstrates how to generate a new rank variable, `points_rank`, that specifically assigns ranks to points scored independently within each `team`. By observing the output, you will note how the **BY statement** dictates that the ranking process is performed separately for Team A and Team B, providing an accurate, localized ranking for internal performance assessment.

```
/*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

Method 3: Creating Percentiles and Quantiles

Beyond calculating simple ordinal ranks, **PROC RANK** offers robust functionality for categorizing **numeric variables** into discrete groups, often referred to as **percentiles** or **quantiles**. This segmentation is achieved by utilizing the powerful **GROUPS option**, which instructs the procedure to divide the data into a specified number of approximately equal-sized parts based on the distribution of values.

For illustrative purposes, consider setting **GROUPS=4**. This command divides the dataset into four distinct **quartiles**, assigning a discrete group identifier (typically 0, 1, 2, or 3) to each observation. Observations with values falling into the lowest **quartile** are assigned group 0, the next **quartile** receives group 1, and so on, up to the highest **quartile**. This process of data segmentation is immensely useful for classifying data points, such as identifying low, medium, and high performers for targeted intervention or reporting.

We will now apply this technique to our example dataset by ranking the `points` variable into **quartiles**. Note that the output will produce a new `points_rank` variable containing categorical integers (0, 1, 2, 3) instead of the continuous rank numbers seen in Method 1. This categorical assignment provides a clear, actionable method for analyzing the distribution and relative standing of values within the variable.

/*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

It is important to recognize the flexibility of the **GROUPS option**. While we demonstrated the creation of **quartiles** (four groups), you can easily adjust the parameter to meet specific analytical needs. For instance, to divide the data into **deciles** (ten groups), you would simply specify **GROUPS=10**. This adaptability makes the procedure a highly versatile tool for various data segmentation requirements, particularly in large-scale data studies.

Method 4: Ranking Multiple Variables Simultaneously

In complex analytical situations, analysts frequently need to compute the **rank** for several distinct **numeric variables** concurrently within a single execution of the SAS procedure. **PROC RANK** is designed for efficiency, handling this requirement by permitting the specification of multiple variables in both the **VAR statement** and the corresponding **RANKS statement**.

When multiple variables are listed, **PROC RANK** processes each one independently and sequentially. It is essential to understand that the rank calculation for one variable is isolated and remains uninfluenced by the values or ranks of the other variables listed in the procedure call. This capability is exceptionally useful when the objective is to assess the relative standing of observations across different metrics simultaneously, such as comparing a team's ranking in points versus its ranking in rebounds.

The following example demonstrates how to efficiently compute the **rank** for both `points` and `rebounds` in one procedure call. The sequence of variables in the **VAR statement** must align precisely with the sequence of new rank variables specified in the **RANKS statement**. The resulting output dataset will include both `points_rank` and `rebounds_rank`, offering a holistic view

of each team's performance across these two key statistics.

```
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

Conclusion and Best Practices

The **PROC RANK** procedure remains an exceptionally versatile and essential tool within the **SAS** environment for comprehensive data manipulation and analysis. Its core functionality--the ability to accurately assign **ranks**, manage tied values using the mean rank method, segment data into groups, and create precise **quantiles**--makes it indispensable for a broad spectrum of statistical applications, ranging from basic data sorting to the necessary preparation steps for advanced non-parametric tests.

By thoroughly understanding and strategically applying the methodologies explored in this guide--including global ranking, localized ranking by group, the derivation of **percentiles**, and the simultaneous ranking of multiple **numeric variables**--you can extract significantly deeper insights from your datasets. It is crucial to always consider the analytical implications of the default tie-breaking rules and to appropriately utilize the **DESCENDING statement** when the analytical objective requires reversing the rank order (i.e., assigning rank 1 to the maximum value).

Mastering the nuances of **PROC RANK** substantially enhances your data preprocessing capabilities, allowing you to transform raw, continuous measurements into meaningful, actionable ordinal data. We highly recommend integrating these ranking techniques into your standard **SAS** workflow to ensure more robust, accurate, and statistically sound subsequent analyses.

Additional Resources

To further enhance your **SAS** programming skills and explore other common data manipulation tasks essential for effective data analysis, consider reviewing the following tutorials: