# Learning How to Add Row Numbers to SAS Datasets: A Comprehensive Guide with Examples

Authored by

**Mohammed looti**

October 31, 2025

## Strategies for Generating Row Numbers in SAS

Assigning a unique sequential identifier to each observation is a fundamental requirement in data management and analysis. In SAS (Statistical Analysis System), there are several effective and efficient ways to add row numbers to a dataset, depending on whether you need simple sequential indexing or numbering that restarts based on specific categorical groups. Understanding these techniques is crucial for tasks like creating unique keys, performing conditional processing, or preparing data for merging operations.

We will explore two primary methods that utilize the power of the DATA step: using the built-in automatic variable _N_ for overall sequential numbering, and leveraging BY group processing for group-specific counts.

## Method 1: Implementing Simple Sequential Row Numbering

The most straightforward approach to assigning a row number across an entire dataset is by using the automatic variable, _**N**_. In SAS, _**N**_ is automatically generated and maintained by the DATA step. It represents the number of times the DATA step has iterated. Since the DATA step typically iterates once for every observation read, _**N**_ serves as a perfect running counter for observations.

To implement this, you simply assign the value of _**N**_ to a new variable (here, `row_number`) within the DATA step. It is best practice to place the assignment statement before the SET statement to ensure the counter increments before the observation is read into the Program Data Vector (PDV), though placing it after the SET statement usually yields the same result unless complex logic is involved.

The following syntax demonstrates how to create a new dataset, `my_data2`, containing the sequential row numbers based on the original dataset, `my_data1`:

```
data my_data2;
row_number = _N_;
set my_data1;
run;
```

## Method 2: Implementing Row Numbering by Group

When your analytical needs require sequential numbering that restarts for each category within a grouping variable (e.g., numbering employees within each department, or orders within each customer), you must employ BY group processing. This advanced technique relies on two crucial steps: first, sorting the data by the grouping variable, and second, utilizing the automatic

`first.variable` indicator within the DATA step.

The initial requirement for BY group processing is that the input dataset must be sorted by the variable specified in the `BY` statement. This is accomplished using the PROC SORT procedure. Once sorted, the subsequent DATA step uses the `first.var1` variable--which evaluates to true (1) only for the first observation of a new group--to reset the row counter.

In the code below, we define a retained variable, `row_number`, and conditionally reset it to zero whenever a new group (defined by `var1`) begins. By then applying an iterative sum (`row_number + 1`, which is a retained sum statement), the counter increments sequentially within that group until `first.var1` signals the start of the next group. This provides a robust method for hierarchical numbering within SAS datasets.

```
/*sort original dataset by var1*/
proc sort data=my_data1;
by var1;
run;

/*create new dataset that shows row number by var1*/
data my_data2;
set my_data1;
by var1;
if first.var1 then row_number=0;
row_number+1;
run;
```

## Setting Up the Sample Data

To illustrate both sequential and grouped row numbering techniques, we will use a small, representative dataset called `my_data1`. This dataset contains observations related to different basketball teams and their scores (points), allowing us to demonstrate both overall counting and counting that resets per team.

The creation of this sample data is handled via the DATA step and `DATALINES` statement, which is a common method for inputting small amounts of raw data directly into a SAS session. The structure includes two variables: `team` (character, indicated by `$`) and `points` (numeric).

The following code block generates the dataset and then uses `PROC PRINT` to display the contents of `my_data1`, allowing us to visualize the starting point before any row numbers are added.

```
/*create dataset*/
```

```
data my_data1;
input team $ points;
datalines;
Mavs 22
Mavs 40
Rockets 41
Rockets 29
Rockets 30
Spurs 18
Spurs 22
Spurs 27
Warriors 13
Warriors 19
;
run;

/*view dataset*/
proc print data=my_data1;
```

| Obs | team | points |
|---|---|---|
| 1 | Mavs | 22 |
| 2 | Mavs | 40 |
| 3 | Rockets | 41 |
| 4 | Rockets | 29 |
| 5 | Rockets | 30 |
| 6 | Spurs | 18 |
| 7 | Spurs | 22 |
| 8 | Spurs | 27 |
| 9 | Warriors | 13 |
| 10 | Warriors | 19 |

## Example 1: Generating a Simple Sequential Row Index

This example demonstrates the application of Method 1, where we use the automatic variable _N_ to assign a unique, continuous row number to every observation in the dataset, regardless of the team. This technique is invaluable when the relative position of an observation within the full dataset is required for referencing or indexing.

We create a new dataset, `my_data2`, by reading `my_data1` using the SET statement. Before the data is processed, the statement `row_number = _N_;` executes, assigning the current iteration count of the DATA step to our new variable. As illustrated in the output, this results in a sequential count from 1 to 10.

The following code shows how to add a new column called **row_number** that contains the sequential row number for each observation:

```
/*create new dataset with column for row numbers*/
data my_data2;
row_number = _N_;
set my_data1;
run;
```

Upon examining the resulting dataset, `my_data2`, we observe that a new column named **row_number** has been successfully appended. This column serves as a consistent, non-grouped index for every observation derived from the original dataset.

| Obs | row_number | team | points |
|---|---|---|---|
| 1 | 1 | Mavs | 22 |
| 2 | 2 | Mavs | 40 |
| 3 | 3 | Rockets | 41 |
| 4 | 4 | Rockets | 29 |
| 5 | 5 | Rockets | 30 |
| 6 | 6 | Spurs | 18 |
| 7 | 7 | Spurs | 22 |
| 8 | 8 | Spurs | 27 |
| 9 | 9 | Warriors | 13 |
| 10 | 10 | Warriors | 19 |

Notice that a new column called **row_number** has been added that contains the row number for each observation in the dataset, counting from 1 through 10.

## Example 2: Generating Row Numbers Reset by Group

In contrast to the overall sequential numbering, this example applies Method 2 to create a row index that restarts its count for every unique value in the `team` variable. This is particularly useful for ranking or identifying the position of records within subsets of data.

The process begins by using PROC SORT to order `my_data1` by the grouping variable, `team`. This step is non-negotiable for correct BY group processing. Following the sort, the DATA step is executed with the `BY var1` statement (where `var1` is implicitly `team` in this context, although the original code used `var1`).

The core logic involves the conditional reset: `if first.var1 then row_number=0;`. When SAS encounters the first record for a new team (e.g., the first 'Mavs' or the first 'Rockets'), the `row_number` is set to 0. The subsequent retained sum statement, `row_number + 1;`, then increments the counter for every observation within that group until the next `first.var1` flag is raised, effectively restarting the count.

The following code shows how to add a row number that resets its count based on the grouping variable `team`:

```
/*sort original dataset by team*/
proc sort data=my_data1;
by var1;
run;

/*create new dataset that shows row number by team*/
data my_data2;
set my_data1;
by var1;
if first.var1 then row_number=0;
row_number+1;
run;
```

The resulting dataset, `my_data2`, clearly demonstrates the effect of this grouped numbering scheme. The **row_number** variable counts observations starting from 1 for 'Mavs', resets to 1 for 'Rockets', and continues this pattern for 'Spurs' and 'Warriors', providing highly specific positional information within each group.

| Obs | team | points | row_number |
|---|---|---|---|
| 1 | Mavs | 22 | 1 |
| 2 | Mavs | 40 | 2 |
| 3 | Rockets | 41 | 1 |
| 4 | Rockets | 29 | 2 |
| 5 | Rockets | 30 | 3 |
| 6 | Spurs | 18 | 1 |
| 7 | Spurs | 22 | 2 |
| 8 | Spurs | 27 | 3 |
| 9 | Warriors | 13 | 1 |
| 10 | Warriors | 19 | 2 |

Notice that the row numbers start over (resetting to 1) for each new team, successfully implementing the logic of BY group processing.

## Additional Resources for SAS Programming

Mastering the DATA step, the SET statement, and automatic variables like _N_ is fundamental to advanced data manipulation in SAS. The methods outlined above provide robust solutions for indexing observations, whether globally or within specific subgroups.

For those seeking to expand their proficiency in data handling and analysis using Statistical Analysis System, the following resources and tutorials explain how to perform other common tasks and procedures.