Rename Variables in SAS (With Examples)

Authored by

Mohammed looti

November 1, 2025

Effective data management is paramount in analytical environments, and one of the most critical aspects of preparing data for modeling or reporting is ensuring clarity through descriptive variable naming. In the SAS (Statistical Analysis System) environment, variables frequently arrive with short, cryptic, or inconsistent names, especially when datasets are imported from external systems or merged from disparate sources. The process of renaming these elements is not just about aesthetics; it is a vital step toward improving code readability, reducing analytical errors, and streamlining collaboration among data professionals. Mastering the tools SAS provides for this operation--particularly the powerful **RENAME=** option--is essential for any SAS programmer.

The fundamental mechanism for modifying variable names in SAS revolves around the **RENAME=** dataset option. This option is primarily utilized within the Data Step, attached directly to the input dataset specified in the **SET** statement. It operates by mapping the existing variable name (the "old name") to the desired new name before the data is written to the new output dataset. It is crucial to understand that when applied in the **SET** statement, the **RENAME=** option does not modify the source dataset; rather, it dictates how variables are read into the Program Data Vector (PDV) and subsequently written out to the newly created dataset. This non-destructive approach preserves the integrity of the original data source.

The standard and most versatile syntax for implementing variable renaming in SAS is remarkably straightforward, requiring only the specification of the old and new names enclosed in parentheses following the RENAME= option. This structure provides a concise and clear method for documenting the transformation history of the variables within the Data Step:

```
data new_data;
set original_data (rename=(old_name=new_name));
run;
```

To provide concrete, reproducible examples of these techniques, we will utilize a simple initial dataset defined temporarily in the **WORK** library. This base dataset, named **ORIGINAL_DATA**, serves as the foundation for all subsequent renaming operations, allowing us to clearly observe the impact of the **RENAME=** option in various configurations. The initial setup code is shown below, followed by an image of the resulting dataset structure, which contains three basic variables: **X**, **Y**, and **Z**.

```
/*create dataset*/
data original_data;
input x y z;
datalines;
1 4 76
2 3 49
```

```
2 3 85
4 5 88
2 2 90
;
run;

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

As illustrated, the **ORIGINAL_DATA** file is simple, making it easy to track changes. We will proceed to use the **RENAME=** option to transform these generic variable labels into more informative identifiers.

Obs	x	y	z
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90

Example 1: Renaming a Single Variable

The most frequent use case for renaming is targeting a single variable whose existing name is ambiguous or conflicts with naming conventions required for downstream analysis. The process is executed entirely within the **SET** statement of a Data Step, leveraging the RENAME= option. This approach ensures that while the source dataset remains untouched, the new dataset reflects the desired changes with precision, allowing for better documentation of the variable's meaning.

To rename a single variable, the syntax requires mapping the current name to the desired name inside the parentheses, using an equals sign as the separator (e.g., `old_name=new_name`). This method is highly effective for localized changes and maintaining clarity in the dataset structure. We must ensure that the new name adheres to standard SAS naming rules, which typically means starting with a letter or underscore and containing no special characters other than underscores. This precise control over a single variable is invaluable during initial data cleaning phases.

Consider the task of renaming the generic variable **X** to the more informative name **NEW_X**, while retaining the structure and names of **Y** and **Z**. The following code demonstrates this focused

transformation, creating a new dataset called **NEW_DATA**. Note how the operation is contained strictly within the parentheses associated with the input data source:

```
/*rename one variable*/
data new_data;
set original_data (rename=(x=new_x));
run;

/*view new dataset*/
proc print data=new_data;
```

A careful review of the output dataset confirms the successful execution of the rename operation. The variable **X** is now unambiguously labeled as **NEW_X**, demonstrating that the **RENAME=** option was applied only to the specified variable pair. This method is the cornerstone of preparing raw data inputs for analysis, providing granular control over individual variable identification without resorting to more complex procedural steps. The efficiency and simplicity of this syntax make it the default choice for minor adjustments.

Obs	new_x	y	z
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90

Example 2: Renaming Multiple Variables Simultaneously

When the requirement extends beyond a single variable--perhaps when aligning dozens of imported columns to match an internal data dictionary--the **RENAME=** option easily scales to accommodate multiple changes within the same Data Step. The efficiency gained by handling all transformations in one statement significantly reduces code length and improves maintainability compared to sequential single-variable renames. This consolidated approach is critical for maintaining clean code, especially in environments where data pipelines involve numerous transformation steps.

To rename multiple variables, the key distinction from the single-variable approach is the inclusion of several `old_name=new_name` pairs within the same set of parentheses, separated merely by a space. Unlike many other programming contexts, SAS does not utilize commas or semicolons

within the **RENAME=** list itself; the space acts as the necessary delimiter, making the syntax clean and compact. This simultaneous mapping ensures atomic transformation, where all specified renames occur before the data is written to the output file, guaranteeing consistency across the newly created dataset.

This example demonstrates how to rename both the **X** variable to **NEW_X** and the **Y** variable to **NEW_Y**, leaving **Z** untouched. The variables are read from **ORIGINAL_DATA**, renamed during the reading process, and written to **NEW_DATA**. Notice the simplicity of listing the pairs consecutively within the option:

```
/*rename multiple variables*/
data new_data;
set original_data (rename=(x=new_x y=new_y));
run;

/*view new dataset*/
proc print data=new_data;
```

The resulting dataset provides immediate confirmation that both specified variables have been successfully updated according to the instructions provided in the **RENAME=** option. This technique is invaluable for large-scale data harmonization projects, particularly when merging datasets that share common variables but use different naming conventions. By consolidating all renaming tasks into a single, understandable statement, data preparation processes become more transparent and verifiable, confirming that **Z** remains unchanged.

Obs	new_x	new_y	z
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90

Example 3: Adding a Prefix to All Variables Using SAS Procedures

While the Data Step and the **RENAME=** option are ideal for targeted, explicit renaming, certain data preparation tasks require systematic, bulk modifications--such as applying a standardized prefix or suffix to every variable in a dataset. This is often necessary before merging data sources (to prevent name conflicts) or for archiving historical datasets (to denote their version or origin).

Performing this task manually or even semi-automatically in a Data Step becomes cumbersome and error-prone for datasets containing hundreds of variables.

For such comprehensive, programmatic transformations, the most robust methodology involves generating the necessary renaming instructions dynamically using PROC SQL and executing them using PROC DATASETS. PROC SQL is employed here not for data manipulation, but for metadata retrieval. It queries the **DICTIONARY.COLUMNS** table, which holds descriptive information about all datasets currently accessible to the SAS session, allowing us to capture the existing variable names and their attributes.

Within PROC SQL, we use the **CATS** function (Concatenate and Strip) to construct the exact renaming string required by the **RENAME** statement: `original_name=_NEWoriginal_name`. This generated list is then stored in a macro variable (named `&list`). Crucially, PROC DATASETS is then invoked with the **MODIFY** statement. Unlike the Data Step, PROC DATASETS modifies the dataset in place, making the change permanent within the specified library, thereby avoiding the creation of an entirely new dataset file. This is highly efficient for large data files.

The code below demonstrates how to add the prefix _**NEW** to variables **X**, **Y**, and **Z** in the **ORIGINAL_DATA** file using this dynamic method:

```
/*define prefix to append to each variable*/
proc sql noprint;
select cats(name, '=', '_NEW', name)
into :list
separated by ' '
from dictionary.columns
where libname = 'WORK' and memname = 'ORIGINAL_DATA';
quit;

/*add prefix to each variable in dataset*/
proc datasets library = work;
modify original_data;
rename &list;
quit;

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

The successful execution of this procedure results in a revised dataset where every variable name is systematically prefaced with _**NEW**. This programmatic approach ensures consistency across all columns and avoids the manual maintenance of extensive renaming lists, making it the preferred

technique for standardizing large dataset structures. The final output confirms the transformation of all three variables.

Obs	NEW_x	NEW_y	NEW_z
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90

Example 4: Adding a Suffix to All Variables

The technique for appending a suffix to all variables follows the identical powerful and dynamic framework established in the previous example. The ability to add suffixes is particularly valuable when merging time-series data or different versions of the same data structure, where the suffix can denote the observation year, measurement unit, or data source. This scalable methodology ensures that even if variables are added or dropped from the source dataset in the future, the renaming logic remains robust and operational without requiring code updates.

The primary difference lies within the PROC SQL **SELECT** statement. Instead of placing the new string before the `name` variable in the **CATS** function, we position it after. We instruct SAS to dynamically generate the renaming pair in the format `name=name_SUFFIX`. This list is efficiently stored in the macro variable `&list`, ready for implementation, and adheres precisely to the syntax required by **PROC DATASETS**.

Once the macro variable is populated, PROC DATASETS is utilized once more to apply these changes directly to the target dataset, minimizing I/O overhead. This two-step process--metadata extraction followed by procedural modification--is the standard for mass data structure alterations in SAS programming, offering superior performance compared to looping Data Steps.

Here is the implementation to add the suffix _**NEW** to all variables in the dataset:

```
/*define suffix to append to each variable*/
proc sql noprint;
select cats(name, '=', name, '_NEW')
into :list
separated by ' '
from dictionary.columns
```

```
where libname = 'WORK' and memname = 'ORIGINAL_DATA';
quit;

/*add suffix to each variable in dataset*/
proc datasets library = work;
modify original_data;
rename &list;
quit;

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

The resulting output clearly shows the variables **X**, **Y**, and **Z** have been renamed to **X_NEW**, **Y_NEW**, and **Z_NEW**. This procedure confirms the flexibility and power of combining metadata extraction through PROC SQL with the modification capabilities of PROC DATASETS for efficient, global variable renaming operations in a SAS environment.

Obs	x_NEW	y_NEW	z_NEW
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90

Summary and Key Takeaways

Renaming variables is far more than a simple clerical task; it is a critical step in ensuring the clarity, accuracy, and longevity of data analysis projects conducted using SAS. This comprehensive guide has detailed two primary strategies for variable renaming, catering to both granular, specific changes and large-scale, automated transformations. The choice of method depends entirely on the scope of the required modification and whether the source dataset needs to be preserved.

For explicit, variable-by-variable changes, the **RENAME=** dataset option, applied within the **SET** statement of the Data Step, offers unparalleled control and readability. It is the preferred tool when the number of variables to be renamed is small or when the goal is to create a new, modified version of an existing dataset without altering the original. This method is highly transparent and minimizes the risk of unintended consequences.

Conversely, when standardizing naming conventions across an entire data structure--such as applying prefixes or suffixes--the combination of PROC SQL for generating dynamic renaming lists and PROC DATASETS for in-place modification proves to be the most efficient and scalable solution. By mastering these techniques, SAS programmers can significantly enhance the quality of their data preparation phase, leading to more robust statistical outputs and easier collaboration. Understanding when to use the targeted Data Step approach versus the bulk procedural approach is a hallmark of efficient SAS coding.

For those looking to expand their knowledge of data handling and transformation, we recommend exploring the following related tutorials:

How to Merge Datasets in SAS
Understanding the Difference Between PROC SQL and Data Steps
Using the KEEP= and DROP= options for variable selection