How to Select Variables in SAS Datasets Using KEEP and DROP Statements

Authored by

stats writer

December 1, 2025

The Necessity of Variable Selection in SAS Programming

The **KEEP** and **DROP** statements are fundamental tools within the SAS system, designed specifically for efficient variable selection when manipulating datasets. In modern data analysis, researchers often encounter extremely large datasets containing hundreds or even thousands of variables. Not all of these variables are necessary for a specific analysis, and processing extraneous columns significantly increases computation time and memory consumption. Therefore, mastering these two statements is crucial for optimizing data processing workflows.

The primary purpose of both the **KEEP** statement and the **DROP** statement is to control which variables are included in a newly created or modified dataset during a Data Step. The **KEEP** statement operates by specifying a whitelist: it instructs SAS to retain only the variables explicitly listed, discarding all others. Conversely, the **DROP** statement utilizes a blacklist approach: it specifies which variables should be excluded from the resulting dataset, while keeping everything else. Utilizing these statements allows analysts to streamline their data structures, reducing complexity and focusing the analytical effort precisely on the relevant data points.

Understanding Variable Selection in the SAS Data Step

You utilize the **KEEP** and **DROP** statements exclusively within a SAS Data Step, which is the cornerstone environment for data manipulation, cleaning, and transformation. The Data Step begins with the `DATA` statement, defining the output dataset, and typically uses the `SET` statement to specify the input dataset. The **KEEP** or **DROP** statement is then inserted between the `SET` statement and the final `RUN` statement, controlling the structure of the variables passed into the new dataset.

It is essential to understand that these statements do not permanently modify the original source dataset. Instead, they operate during the execution of the Data Step to create a brand new dataset that inherits the structure and observations of the original but contains only the specified subset of variables. This non-destructive approach is a fundamental safety mechanism in SAS programming, ensuring that raw data integrity is always maintained while analysts work with targeted subsets.

Basic Syntax and Functionality of the KEEP and DROP Statements

The syntax for both statements is straightforward, requiring the keyword followed by a space-separated list of the variables intended for inclusion or exclusion. Proper placement within the Data Step ensures that SAS processes the variable selection logic correctly before finalizing the output dataset structure.

Method 1: Choosing Which Variables to KEEP

The KEEP statement is utilized when the analyst knows exactly which few variables are required

for the downstream analysis. It is highly efficient when the input dataset contains a vast number of columns, but only a small handful are needed.

```
data new_data;
set original_data;
keep var1 var3;
run;
```

Method 2: Choosing Which Variables to DROP

The DROP statement is preferred when the analyst only needs to exclude a few specific variables, retaining the majority of the columns from the source data. This method saves time by avoiding the need to list every single variable that should be preserved.

```
data new_data;
set original_data;
drop var5;
run;
```

Setting Up the Illustrative Example Dataset

To demonstrate the practical application of these statements, we will first create a small example dataset named `original_data`. This dataset simulates sports statistics, featuring three key variables: `team` (character variable), `points` (numeric variable), and `rebounds` (numeric variable). This simple structure provides a clear basis for illustrating how both the KEEP statement and the DROP statement manipulate the resulting data structure.

```
/*create original dataset using DATALINES*/
data original_data;
input team $ points rebounds;
datalines;
Warriors 25 8
Wizards 18 12
Rockets 22 6
Celtics 24 11
Thunder 27 14
Spurs 33 19
Nets 31 20
;
run;
```

```
/*view dataset using PROC PRINT*/
proc print data=original_data;
```

The execution of this code block generates the initial data structure. The `PROC PRINT` procedure is utilized here to display the contents of the newly created `original_data` dataset, confirming that all three variables (`team`, `points`, and `rebounds`) are present and correctly populated according to the input lines.

Obs	team	points	rebounds
1	Warriors	25	8
2	Wizards	18	12
3	Rockets	22	6
4	Celtics	24	11
5	Thunder	27	14
6	Spurs	33	19
7	Nets	31	20

Practical Application: Using the KEEP Statement

In the first example, our analytical goal is to examine only the relationship between the `team` name and the number of `rebounds` they accumulated, completely ignoring the `points` variable. This scenario is perfectly suited for the KEEP statement, as we are explicitly defining a small subset of variables to retain.

The following code demonstrates how to create a new dataset named `new_data_keep`. By employing the `keep team rebounds;` statement, we mandate that only these two variables are carried over from the `original_data` dataset during the execution of the Data Step. All other variables present in the source dataset are automatically excluded from the final output.

```
/*create new dataset using KEEP statement*/
data new_data_keep;
set original_data;
keep team rebounds;
run;

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

The output confirms that the resulting dataset, `new_data_keep`, contains only the **team** and **rebounds** variables. The **points** variable, which was not explicitly listed in the KEEP statement, has been effectively dropped. This exemplifies the precise control offered by this method when reducing the dimensionality of a dataset.

Obs	team	rebounds
1	Warriors	8
2	Wizards	12
3	Rockets	6
4	Celtics	11
5	Thunder	14
6	Spurs	19
7	Nets	20

Practical Application: Using the DROP Statement

For our second example, imagine the analyst decides that the `rebounds` statistic is irrelevant for their current analysis, but they wish to keep both `team` and `points`. Since we only need to remove one variable, using the DROP statement is the most concise approach.

The code below generates a new dataset named `new_data_drop`. The instruction `drop rebounds;` tells the SAS system to exclude the **rebounds** column from the output file, automatically retaining all other variables present in the `original_data`.

```
/*create new dataset using DROP statement*/
data new_data_drop;
set original_data;
drop rebounds;
run;

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

After running the code, we observe the resulting structure. The **rebounds** variable was successfully excluded, leaving the **team** and **points** variables intact. This illustrates the fundamental difference in mechanism: while **KEEP** requires positive declaration of variables to include, **DROP** requires positive declaration of variables to omit.

Obs	team	points
1	Warriors	25
2	Wizards	18
3	Rockets	22
4	Celtics	24
5	Thunder	27
6	Spurs	33
7	Nets	31

Advanced Techniques: Using Variable Lists with KEEP and DROP

When dealing with datasets containing many variables, manually listing every variable to keep or drop can be tedious and prone to error. Fortunately, SAS offers powerful shortcuts for specifying lists of variables. These list shortcuts drastically improve coding efficiency, particularly when variables are named sequentially or belong to a specific type.

Common methods for specifying variable lists include:

`VAR1--VARN`: This specifies all variables in the source dataset that are physically positioned between VAR1 and VARN (inclusive). The order is determined by how they appear in the source dataset definition.

`VAR1-NUMERIC-VARN`: This specifies all numeric variables positioned between VAR1 and VARN.

`_CHARACTER_`: This is a special keyword used to refer to all character variables in the dataset.

`_NUMERIC_`: This refers to all numeric variables in the dataset.

`_ALL_`: This refers to all variables in the dataset. (While useful for other statements, **KEEP** and **DROP** are typically used to select a subset, making explicit listing usually necessary unless combined with renaming or other conditional logic.)

For example, if you had variables named V1, V2, V3, and V4, and you wanted to keep V1, V2, and V3, you could use: `keep V1-V3;`. Alternatively, if you wanted to drop V4, you could use: `drop V4;`. Utilizing these variable list shortcuts is highly recommended for maintaining readable and scalable SAS code, especially in production environments where datasets often undergo structural changes.

Strategic Comparison: KEEP vs. DROP

While both the **KEEP** and **DROP** statements achieve the same outcome--creating a new dataset with a subset of variables--the choice between them is a matter of programming efficiency, clarity, and safety. There is no difference in execution speed; the choice is purely logistical based on the number of variables involved.

A simple rule of thumb guides the selection process:

Use the KEEP statement when you want to retain only a few variables and discard many. Listing the few variables to keep is far quicker and less error-prone than listing the dozens or hundreds of variables to drop.

Use the DROP statement when you want to discard only a few variables and retain many. Listing the small number of variables to drop is vastly more efficient than listing the majority of variables that should be kept.

The choice also influences code maintainability. If new variables are added to the source dataset later, the **DROP** statement is safer if you intend to keep all new variables automatically. Conversely, the **KEEP** statement is safer if you want the structure of the output dataset to remain immutable, preventing any new variables from automatically appearing in the output unless explicitly added to the **KEEP** list. This safety consideration is often the deciding factor in highly regulated or production-critical programming environments.

Best Practices for Efficient SAS Programming

When implementing variable selection in large-scale data processing workflows, adherence to certain best practices can prevent errors and improve execution performance. Properly placing the KEEP statement or the DROP statement is paramount. They should generally appear immediately after the `SET` statement to ensure that variables are selected before any subsequent data manipulations (such as calculations or conditional assignments) occur in the Data Step.

It is also important to recognize the difference between the **KEEP**/**DROP** statements used within the Data Step and the **KEEP=**/**DROP=** dataset options used directly on the `SET` or `DATA` statements.

Statement (e.g., `DROP var1;`): Applies the selection logic at the end of the Data Step execution, only affecting the output dataset. Variables dropped by the statement are available for use and calculation within the Data Step itself.

Dataset Option (e.g., `SET original_data(DROP=var1);`): Applies the selection logic when the data is read into the Data Step. Variables dropped using this option are never brought into the

processing environment and cannot be referenced or used for calculations within the Data Step.

For performance optimization, using the **KEEP=** or **DROP=** dataset options on the `SET` statement is often superior, especially with massive files, as it minimizes the amount of data read from disk and loaded into memory. This is particularly beneficial in high-performance computing environments where I/O efficiency is critical.

Conclusion and Further Reading

The **KEEP** and **DROP** statements are indispensable commands for any SAS programmer seeking to manage data efficiently. By providing clear mechanisms for variable selection, they enable the creation of lightweight, focused datasets that accelerate analysis and reduce computational overhead. Choosing between the two depends primarily on which method requires the shortest list of variables to specify--the list of those to keep or the list of those to drop.

Mastering these techniques ensures your SAS programs are not only functionally correct but also optimized for performance and clarity.

For further tutorials that explain how to perform other common tasks in SAS, consider the following resources: