Learning to Select Variables in SAS: Using KEEP and DROP Statements

Authored by

Mohammed looti

October 31, 2025

The process of data preparation often requires the user to select specific subsets of variables from a larger existing dataset. In SAS, this crucial task of variable management is efficiently handled using the **KEEP** and **DROP** statements within the DATA step. These powerful statements allow analysts to streamline their data structures, improving computational performance, enhancing data security by removing sensitive fields, and simplifying subsequent analysis by focusing only on relevant attributes. Understanding the nuances of when and how to deploy **KEEP** versus **DROP** is fundamental to effective SAS programming.

While both statements achieve the goal of creating a new, reduced dataset based on an existing source, they operate on conceptually opposite principles. The **KEEP** statement functions as an inclusion mechanism, explicitly listing every variable that must be preserved in the output dataset, with all unlisted variables automatically discarded. Conversely, the **DROP** statement acts as an exclusion mechanism, explicitly listing only those variables that should be removed or ignored, thereby retaining all other variables present in the source dataset. Mastering this distinction allows the programmer to choose the most efficient and readable approach for any given subsetting requirement.

The Core Syntax of KEEP and DROP Statements

Both the **KEEP** and **DROP** statements are typically placed immediately after the `SET` statement within the DATA step. This placement ensures that the instructions for variable retention or exclusion are processed during the compilation phase, optimizing how SAS handles the data records as they are read from the input dataset and written to the new output dataset. Adhering to this standard syntax structure is vital for maintaining clean and predictable code flow in SAS environments.

Method 1: Choosing Which Variables to KEEP involves specifying a definitive list of the variables you wish to carry forward into the new dataset. This method is highly transparent, as the resulting dataset's column structure is immediately visible by reviewing the **KEEP** list. This is the preferred method when dealing with source datasets containing a large number of variables, but where only a small subset is required for the analysis. The syntax below illustrates this inclusion-based approach:

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

Method 2: Choosing Which Variables to DROP utilizes an exclusive mechanism, where you list only the variables that you intend to discard from the new dataset. All variables not explicitly listed

in the **DROP** statement are automatically included in the output dataset. This method is advantageous when the majority of variables in the source dataset are relevant, and only a few extraneous or redundant fields need to be removed. The syntax below demonstrates this exclusion-based approach:

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

To demonstrate these methods, we will utilize a sample dataset named `original_data`. This dataset contains three distinct variables: `team` (character), `points` (numeric), and `rebounds` (numeric). The creation of this sample data structure is achieved using the standard SAS input method, which is followed by a simple view using the PROC PRINT procedure to confirm the initial structure before subsetting operations begin.

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

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 many analytical scenarios, researchers are often interested in only a small fraction of the available data fields. When this occurs, the **KEEP** statement provides the clearest path to achieving the desired subset. For instance, if our analysis requires only the identification of the team and their respective rebound counts, excluding the 'points' variable, the **KEEP** statement allows us to define this concise structure explicitly within the DATA step. This approach minimizes the risk of errors associated with mistakenly omitting a needed variable, as the intent is clearly defined by the list of variables to be retained.

The following code block demonstrates how to create a new dataset named `new_data` by selectively retaining only the `team` and `rebounds` variables from the `original_data` source. The `KEEP` statement ensures that the variable `points` is never written to the output file, thereby optimizing storage and memory usage immediately.

```
/*create new dataset*/
data new_data;
set original_data;
keep team rebounds;
run;

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

Upon viewing the output generated by the PROC PRINT procedure, we can observe that the structure of `new_data` has been successfully reduced. Only the columns explicitly listed in the **KEEP** statement--`team` and `rebounds`--are present. The variable `points`, which existed in the original dataset, has been effectively dropped by omission.

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

Conversely, there are scenarios where the source dataset is expansive, perhaps containing dozens of variables, but only one or two fields are deemed unnecessary or irrelevant for the new analysis. In such cases, the **DROP** statement offers a much more efficient and less error-prone solution than listing every variable to be retained. By utilizing **DROP**, the programmer only needs to identify the specific variables to exclude, making the code significantly shorter and easier to maintain.

The following example illustrates the creation of the `new_data` dataset using the **DROP** statement to remove only the `rebounds` variable. This means that both `team` and `points` will automatically be included in the output dataset. This method is particularly useful in exploratory data analysis where variables are temporarily excluded without the need to reformulate a long **KEEP** list.

```
/*create new dataset*/
data new_data;
set original_data;
drop rebounds;
run;

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

The resulting output confirms that the `rebounds` variable has been successfully excluded from the new dataset. Importantly, all other variables present in the `original_data`--namely `team` and `points`--were preserved automatically because they were not explicitly named in the **DROP** list. This illustrates the fundamental difference in approach: **KEEP** requires naming what stays, while **DROP** requires naming what goes.

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

Strategic Decision Making: KEEP vs. DROP

While **KEEP** and **DROP** statements are functionally inverse operations, leading to identical final dataset structures when applied correctly, the decision of which to use is primarily governed by principles of code efficiency, maintenance, and clarity. The core principle revolves around minimizing the length of the list of variables the programmer must type and manage.

A general best practice in SAS programming dictates that if the list of variables to be retained is significantly shorter than the list of variables to be discarded, the **KEEP** statement should be used. Conversely, if the list of variables to be discarded is much shorter than the list of variables to be retained, the **DROP** statement is the more appropriate choice. This simple heuristic drastically reduces the chances of typographical errors, especially in environments where datasets may contain hundreds of columns.

Furthermore, considering future maintenance is critical. If new variables are added to the source dataset over time, using **KEEP** ensures that only the specifically required variables are included, preventing unintended side effects. However, if the intent is to maintain the dataset structure while excluding a few legacy or derived variables, **DROP** is safer, as new variables added to the source dataset will automatically pass through to the target dataset without needing code modification.

Advanced Considerations and Alternatives

Beyond their use as standalone statements within the DATA step, the variable subsetting functionality of **KEEP** and **DROP** can also be applied as dataset options. These options are enclosed in parentheses immediately following the dataset name in the `SET` statement or the `DATA` statement.

When used as dataset options on the `SET` statement, the filtering occurs immediately upon reading

the data from the source, meaning those variables are never brought into the program data vector (PDV). The syntax would appear as: `SET original_data (KEEP=team rebounds);`. This method is often preferred for performance reasons, particularly when working with very large source datasets, as it reduces the I/O burden.

It is also important to note that **KEEP** and **DROP** options can be used temporarily in many SAS procedures, such as PROC PRINT, `PROC MEANS`, or `PROC UNIVARIATE`, to limit the analysis scope without creating a permanent new dataset. For example, `PROC MEANS DATA=original_data (DROP=team);` would run the procedure on the data without the `team` variable, but the original dataset remains unchanged. This flexibility ensures that analysts can manage their data effectively regardless of whether they need a permanent structural change or just a temporary restriction for a specific statistical run.

Additional Resources

For users seeking to broaden their proficiency in SAS data manipulation, exploring other fundamental tutorials is highly recommended. These resources explain how to perform other common tasks in SAS:

How to handle missing data values in SAS.
Techniques for combining multiple datasets using MERGE and APPEND.
Methods for conditional data processing using the IF-THEN/ELSE structure.