How to Use the SAS DATA Step to Manipulate Your Data: A Beginner’s Guide

Authored by

stats writer

November 21, 2025

Introduction to the SAS DATA Step: The Core of Data Management

The DATA step is the fundamental building block within the SAS programming environment, designed specifically for reading, transforming, and outputting data. It serves as the engine for data preprocessing, ensuring that raw information is organized, cleaned, and shaped into a functional SAS dataset--the primary structure for subsequent analysis. Mastering the DATA step is essential for any SAS professional seeking to manage complex data workflows efficiently.

A typical DATA step consists of a sequence of statements that instruct SAS on how to process input data, manipulate variables, handle missing values, and structure the final output dataset. Unlike procedural steps (like those starting with `PROC`), the DATA step operates record-by-record, iteratively processing observations until all input data has been read. This iterative process allows for precise control over data transformations and the creation of highly customized datasets tailored to specific analytical needs.

The versatility of the DATA step is unmatched when performing critical data management tasks. Common applications include filtering unwanted observations, sorting records based on key variables, summarizing information across groups, creating new calculated variables, and ensuring data quality by identifying and eliminating duplicate records. Essentially, if a data modification is required before analytical modeling begins, the DATA step is the designated tool for the job.

Understanding the Phases of the DATA Step

To effectively utilize the DATA step, it is crucial to understand the two distinct operational phases that SAS employs: the Compilation phase and the Execution phase. These phases work sequentially to process the code and generate the resulting dataset.

During the **Compilation phase**, SAS reads all the statements within the DATA step, verifying syntax and identifying the structure of the intended output dataset. It determines the number, names, and types of variables that will exist in the Program Data Vector (PDV)--an internal buffer used to hold the current observation being processed. No data is read during compilation; this phase is purely structural and logical. If any fundamental syntax errors are found, the DATA step will usually fail before execution begins.

The **Execution phase** commences immediately after successful compilation. In this phase, SAS reads the input raw data (or data from existing SAS files) one observation at a time. For each observation, the PDV is populated with the values, and all statements (like assignment statements, conditional logic, or transformation functions) are executed in order. This iterative loop continues until all input records have been processed, at which point the final dataset is written to the specified location.

Primary Applications of the DATA Step

The DATA step is broadly categorized based on its source of input. While complex operations often combine multiple sources and statements, most usages fall into one of two fundamental categories related to dataset creation:

Creating a Dataset from Scratch: This involves reading external raw data, often entered directly into the program using `DATALINES`, or reading from an external file using an `INFILE` statement. This method is crucial when importing data that has not yet been processed or standardized into the SAS format.

Creating a Dataset from an Existing Dataset: This method utilizes the `SET` statement to copy or modify variables and observations from one or more existing SAS datasets. This is the standard procedure for subsetting, merging, or updating existing analytical files.

The following examples demonstrate these two common approaches, illustrating the syntax required to initiate and finalize these data construction processes.

Example 1: Generating a Dataset Using Inline Data

Creating a dataset directly within the SAS code using inline data is often the quickest way to demonstrate concepts or create small lookup tables. This process requires explicitly defining the variable names and types using the `INPUT` statement, followed by the raw data entered after the `DATALINES` statement.

The following syntax creates a dataset named **my_data** that records performance statistics for different team members, utilizing three variables: `team`, `position`, and `points`.

```
/*create dataset*/
data my_data;
input team $ position $ points;
datalines;
A Guard 25
A Guard 20
A Guard 30
A Forward 25
A Forward 10
B Guard 10
B Guard 22
B Forward 30
B Forward 10
```

```
B Forward 10
B Forward 25
;
run;

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

Obs	team	position	points
1	A	Guard	25
2	A	Guard	20
3	A	Guard	30
4	A	Forward	25
5	A	Forward	10
6	B	Guard	10
7	B	Guard	22
8	B	Forward	30
9	B	Forward	10
10	B	Forward	10
11	B	Forward	25

Once the DATA step concludes execution, a subsequent PROC PRINT statement is used to display the contents of the newly created dataset, confirming that the data was read and structured correctly by SAS.

Dissecting the Code: Key Statements for Data Input

A detailed examination of the syntax in Example 1 reveals the foundational statements necessary for inline data entry within the DATA step environment:

First, the **`DATA` statement** (`data my_data;`) initializes the DATA step and assigns the name **my_data** to the resulting output dataset. Every DATA step must begin with this statement, clearly defining the output file path or name.

Second, the **`INPUT` statement** (`input team $ position $ points;`) defines the order, names, and types of the variables being read from the raw data lines. The dollar sign (`$`) following a variable name, such as **team** and **position**, explicitly specifies that these are character variables (text strings), while variables without the dollar sign (like **points**) are treated as numeric variables.

Finally, the **`DATALINES` statement** (`datalines;`) signals to SAS that the subsequent lines contain the actual raw data to be processed and stored in the **my_data** dataset. Data input concludes when SAS encounters a semicolon (`;`) on a line by itself, followed by the terminating **`RUN` statement**, which signals the end of the DATA step and triggers the writing of the final dataset.

Method 2: Deriving New Datasets from Existing SAS Files

A more frequent use case for the DATA step involves creating derivative datasets based on data that already exists in a SAS file format. This technique is crucial for tasks like creating subsets of variables, filtering observations, or integrating data from multiple sources.

To achieve this, the primary mechanism is the **`SET` statement**. The **SET** statement instructs the DATA step to read observations sequentially from the specified existing SAS dataset(s), populating the PDV with the source data before any transformations or conditional logic are applied. This is significantly more efficient than re-reading raw data, as the variables are already defined and formatted correctly.

By combining the **`DATA` statement** (to name the new file) and the **`SET` statement** (to designate the source file), programmers can easily initiate the creation of a derivative file. Further refinement statements, such as `DROP`, `KEEP`, or `WHERE`, can then be inserted to specify exactly which variables or observations should be included in the final output.

Example 2: Subset Creation Using Existing Data

This example demonstrates how to leverage an existing SAS dataset, such as the **my_data** file created earlier, to generate a new dataset named **new_data**. The goal here is to create a copy of the original data while simultaneously excluding (dropping) a specific variable, assumed to be 'returns' for demonstration purposes. This illustrates the power of subsetting options within the DATA step.

We use the **`DROP` statement** to exclude the variable 'returns' from the output dataset, ensuring that **new_data** retains all information from **my_data** except for that one column.

```
/*create new dataset that drops returns from my_data*/
data new_data;
set my_data;
drop returns;
run;

/*view dataset*/
```

proc print data=new_data;

Obs	store	sales
1	A	10
2	A	7
3	A	7
4	A	8
5	A	6
6	B	10
7	B	14
8	B	13
9	B	9
10	B	5
11	C	12
12	C	10
13	C	10
14	C	12
15	C	9

The result is a structurally sound new dataset, **new_data**, which contains all the variables and observations from the source dataset **my_data**, minus the specified variable that was excluded using the `DROP` statement. The PROC PRINT output confirms this structural modification.

Advanced Manipulation within the DATA Step

Beyond simple creation and copying, the DATA step provides powerful tools for intricate data manipulation. Statements like **`DROP`** and **`KEEP`** offer fine-grained control over the variables included in the output, while conditional processing statements (e.g., `IF-THEN/ELSE`) allow for highly flexible record modification and filtering.

The **`DROP` statement**, as shown in Example 2, specifies variables that should be read into the PDV but excluded from the final output dataset. Conversely, the **`KEEP` statement** specifies only those variables that should be retained in the output, implicitly dropping all others. Using **KEEP** often results in cleaner code when only a small subset of variables is needed. These statements are fundamental to optimizing dataset size and simplifying subsequent analysis.

Moreover, the DATA step supports complex logic using conditional statements. For instance, an **IF**

statement allows the programmer to execute operations only when certain criteria are met, such as calculating a new variable based on existing data values (`IF points > 25 THEN elite = 1;`). This capability is crucial for transforming raw measurements into meaningful analytical features or for selectively removing erroneous or irrelevant observations from the dataset based on predefined conditions.

Summary and Best Practices for DATA Step Utilization

The DATA step is the cornerstone of data handling in SAS, providing the infrastructure necessary for everything from reading raw data to performing sophisticated transformations. Understanding its two-phase structure--Compilation and Execution--is key to debugging and writing efficient code.

For robust SAS programming, adhering to best practices during the DATA step is essential. Always clearly document the purpose of the DATA step using comments (`/* comments */`), especially when performing complex calculations or transformations. Use explicit variable types (numeric vs. character variable, denoted by `$`) during input to avoid potential data interpretation errors. Furthermore, prefer using **KEEP** over **DROP** when the desired output variables are fewer than the input variables, improving code readability and reducing the chance of accidental variable inclusion.

By mastering the syntax demonstrated in the examples--whether using **INPUT** and **DATALINES** for initial dataset creation or employing **SET** and **DROP**/**KEEP** for derived dataset management--SAS programmers can ensure that their data is always clean, structured, and ready for advanced statistical analysis.

The following tutorials explain how to perform other common tasks in SAS: