How to Easily Create a SAS Dataset Using the DATALINES Statement

Authored by

stats writer

December 1, 2025

The DATALINES statement in the SAS programming environment stands as a fundamental and highly versatile component for initiating data creation. This statement empowers users to embed raw data directly within the program code itself, treating the code editor much like an in-line spreadsheet. This method is particularly efficient for rapid prototyping, developing proof-of-concept analyses, or generating small datasets essential for testing complex SAS procedures and syntax modules. It bridges the gap between external data files and the necessary internal structure required by the SAS system.

Historically, this statement was often referred to using the alias **CARDS statement**, although DATALINES is now the preferred modern term and functionality is identical. Regardless of the keyword used, the primary function remains the same: it signals the start of the data section within a DATA step. The raw data values immediately follow this keyword, typically delimited by single spaces or commas, depending on the complexity of the data fields and the input style chosen. Crucially, the data input process must be terminated formally; in modern practice, this termination is signaled by a single semicolon on the line immediately following the last data record.

Mastery of the DATALINES statement is invaluable for any SAS professional, especially when dealing with scenarios where external file access is restricted or when the data volume is minimal enough not to warrant creating a separate text file. It ensures the entire data creation and processing logic remains contained within a single program file, enhancing script portability and reducing reliance on external dependencies. This self-contained approach is highly beneficial for tutorials, quick demonstrations, and debugging small code segments.

The Role of DATALINES in the SAS DATA Step

The foundation of data handling in SAS rests upon the DATA step, which is the mechanism responsible for reading, transforming, and structuring data into a usable dataset. The DATALINES statement is strategically placed within this step after the DATA and INPUT statements, instructing the system that the raw data source is embedded directly below, rather than pointing to an external file path using the `INFILE` statement.

This sequential integration is mandatory because the DATA step must first establish the structure and metadata of the future dataset before attempting to read any actual values. The DATA statement names the new object, and the INPUT statement defines the column headers (variable names) and their corresponding types (numeric or character). Only once this foundational structure is successfully established does the DATALINES statement command the SAS Data Engine to process the subsequent lines of text as data records, mapping each value to the defined variables.

The use of DATALINES simplifies the logistics of data input for quick tasks by eliminating the need to manage external files and file paths, which can often be cumbersome in different operating environments. It ensures that the process of data definition and population is entirely encapsulated,

resulting in cleaner, more portable code scripts that are easier to share and reproduce.

Essential Syntax for Using DATALINES

You can use the **datalines** statement in SAS to quickly create a new dataset from scratch. This involves a four-part structure: the DATA statement, the INPUT statement, the DATALINES block, and the `RUN` statement.

You can use the following basic syntax to successfully define and populate a simple two-variable dataset:

```
data original_data;
input var1 $ var2;
datalines;
A 12
B 19
C 23
D 40
;
run;
```

Deconstructing the Essential SAS Statements

Each keyword within the DATA step serves a crucial function in instructing the SAS compiler how to interpret the code and the data. Understanding the precise role of each statement is fundamental to writing error-free SAS programs, particularly when defining variable types.

data: This statement initiates the DATA step and assigns a name (e.g., `original_data`) to the new dataset being constructed. If no library name is specified, SAS saves the dataset temporarily in the `WORK` library.
input: The INPUT statement defines the variables, their sequential order, and their data type within the records. The presence of the dollar sign (`$`) immediately following a variable name is the explicit instruction for SAS to treat that variable as a **character variable**.
datalines: This critical statement serves as a demarcation point, halting the processing of program logic and beginning the reading of raw data records. All subsequent lines, up until the terminating semicolon, are treated as records to be processed against the variable definitions provided in the INPUT statement.
run: The `RUN` statement signals the completion of the DATA step block. It triggers the execution of all preceding statements, processes the data lines defined under DATALINES, and finalizes the creation and storage of the new dataset.

Handling Character and Numeric Variables

Accurate variable typing is paramount in SAS. Data must be classified as either **numeric** (containing only digits, decimal points, and signs, suitable for calculation) or **character** (containing letters, symbols, or alphanumeric combinations).

As established, the dollar sign "**$**" following a variable name in the INPUT statement is the essential identifier for a **character variable**. If this symbol is omitted, SAS defaults to interpreting the variable as a **numeric variable**. This strict distinction dictates how the data is stored and which procedures can be applied to it. For instance, statistical procedures like means calculation can only be applied to numeric variables; applying them to character variables will result in errors or warnings.

When inputting data using DATALINES with list input, SAS expects values to be separated by one or more spaces. This works well for simple tokens, but if character values inherently contain spaces (e.g., proper names or descriptions), the input style must be adjusted to use modified list input or column input to prevent the internal space from being read as a delimiter separating two distinct values. For the purpose of these examples, we rely on atomic data tokens without internal spaces.

Example 1: Creating a Dataset with All Numeric Variables

The first example demonstrates the straightforward creation of a dataset consisting exclusively of numeric variables, which is typical for analyzing quantitative research data or performance metrics. This method avoids the need for the dollar sign operator in the INPUT statement, leading to concise code.

The following code creates a dataset named `original_data` to track basketball statistics, including **points**, **assists**, and **rebound**s. Since the data consists only of integers, the default numeric type is sufficient. We follow the DATA step with a **PROC PRINT** step to immediately display the contents and confirm successful creation.

```
/*create dataset*/
data original_data;
input points assists rebounds;
datalines;
22 8 4
29 5 4
31 12 8
30 9 14
22 7 1
```

```
24 9 2
18 6 4
20 5 5
25 1 4
;
run;

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

Obs	points	assists	rebounds
1	22	8	4
2	29	5	4
3	31	12	8
4	30	9	14
5	22	7	1
6	24	9	2
7	18	6	4
8	20	5	5
9	25	1	4

The result is a neatly formatted table showing the nine observations and three variables. The output confirms that the DATALINES statement successfully transformed the raw, space-delimited text into a valid SAS dataset with three numeric variables.

Example 2: Handling Mixed Character & Numeric Variables

When data involves descriptive information alongside metrics, we must explicitly declare which variables are text-based. This second example expands upon the first by integrating two **character variables**: `team` and `position`.

The following code shows how to create a dataset that correctly distinguishes between the two variable types. Notice the placement of the dollar sign ($) immediately after `team` and `position` in the INPUT statement, ensuring that these fields are read as categorical text data, while `points` and `assists` are processed as numeric values suitable for computation.

/*create dataset*/

```
data original_data;
input team $ position $ points assists;
datalines;
A Guard 8 4
A Guard 5 4
A Forward 12 8
A Forward 9 14
A Forward 7 1
B Guard 9 2
B Guard 14 9
B Forward 15 8
B Forward 11 4
;
run;

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

Obs	team	position	points	assists
1	A	Guard	8	4
2	A	Guard	5	4
3	A	Forward	12	8
4	A	Forward	9	14
5	A	Forward	7	1
6	B	Guard	9	2
7	B	Guard	14	9
8	B	Forward	15	8
9	B	Forward	11	4

The resulting table confirms that the character values (Team and Position) are correctly stored as text. The key to successful mixed input is matching the order of variables in the INPUT statement with the order of the raw values supplied in the DATALINES statement, ensuring type indicators ($) are placed only where needed.

Verifying Dataset Structure using PROC CONTENTS

To ensure data integrity, especially after complex data manipulation or input from raw sources like DATALINES, it is critical to use the **PROC CONTENTS** procedure. This procedure provides a comprehensive summary of the metadata associated with the dataset, including vital information on variable types and lengths.

We use **proc contents** on the `original_data` created in Example 2 to formally verify that the variables were assigned their intended types during the DATA step execution.

```
proc contents data=original_data;
run;
```

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
4	assists	Num	8
3	points	Num	8
2	position	Char	8
1	team	Char	8

The resulting table confirms the definitions: **team** and **position** are listed as Type **Char** (Character variables), while **points** and **assists** are listed as Type **Num** (Numeric variables). This validation provides absolute confidence that the data is structured correctly and ready for appropriate statistical analysis or reporting procedures.

Conclusion

The DATALINES statement is a cornerstone of efficiency in SAS programming, offering a direct, self-contained method for defining and populating small to medium-sized datasets. By mastering the coordination between the DATA statement, the variable definition in the INPUT statement (especially the use of the dollar sign for character variables), and the termination of the raw data block, users gain a powerful tool for rapid development and testing.

While external file handling (using `INFILE`) is necessary for large-scale production environments, the simplicity and portability afforded by the DATALINES statement make it an essential skill for all SAS users. The ability to quickly create a test environment without relying on file system access significantly enhances productivity in educational, developmental, and exploratory data analysis contexts.

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