Learning SAS: Creating Datasets with the DATALINES Statement

Authored by

Mohammed looti

October 31, 2025

The process of data preparation often requires users of statistical software to quickly input small amounts of raw data for testing, demonstration, or immediate analysis. In the context of SAS programming, the **datalines statement** offers an elegant and efficient method for creating a new, self-contained dataset directly within the program code. This technique is indispensable for scenarios where data volume is minimal, or when demonstrating code functionality without relying on external file dependencies. Understanding how to properly structure a **datalines statement** block is fundamental to effective SAS programming, ensuring immediate data availability and reducing potential input errors associated with external file paths and formats. We will explore the precise syntax and practical applications of this powerful data input mechanism.

Establishing the Foundation: Core Syntax for Data Input

To construct a dataset using inline data, a standard sequence of steps must be followed. This process begins with the DATA step, which names the new dataset, followed by the INPUT statement, which defines the variables and their types. Finally, the datalines statement signals to the SAS compiler that the subsequent lines contain the raw data values that correspond to the variables defined earlier. It is crucial that the data block is terminated correctly, typically using a semicolon (;) immediately followed by the mandatory **run statement**, which executes the entire DATA step and commits the new dataset to the SAS environment.

The following structure represents the minimum required syntax for leveraging the **datalines statement** to initialize a basic dataset. This structure outlines the naming of the resulting dataset, the definition of its internal variables (including data types), and the provision of the raw data itself.

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

Each component within this block plays a specific and critical role in the successful creation of the SAS dataset. Defining these roles clearly is essential for debugging and modifying data input procedures. The variables declared in the **INPUT statement** must align perfectly with the sequence of values provided in the data lines, ensuring that data points are correctly assigned to their respective columns. If the sequence or data types mismatch, SAS may issue warnings or, worse, incorrectly interpret the input values, leading to unreliable analytical results.

data: This command initializes the DATA step and assigns the chosen name (e.g., `original_data`) to the new SAS dataset being created in memory.
input: The INPUT statement lists the names of the variables sequentially, exactly as they appear in the subsequent data lines, alongside any necessary format or type specifiers.
datalines: This keyword serves as the delimiter, indicating the immediate end of the program logic and the start of the raw data input block.

A fundamental concept when defining variables in SAS is distinguishing between data types. Specifically, a dollar sign (**$**) immediately following a variable name in the **INPUT statement** is the explicit method used to inform SAS that the variable should be treated as a character variable (textual data). Variables defined without the dollar sign are automatically assumed to be **numeric variables** (values that can be used in calculations). This distinction is vital, as attempting to perform arithmetic operations on a character variable will result in errors, and conversely, failing to designate a textual field as character will cause SAS to treat non-numeric entries as missing values.

Example 1: Generating a Dataset with Purely Numeric Variables

In many analytical scenarios, the data being input consists solely of numerical measurements, such as counts, scores, or physical dimensions. When all variables are numeric, the syntax for the INPUT statement is simplified, as no explicit type declaration (like the **$** sign) is necessary. SAS automatically recognizes the variables as numeric data types based on the absence of the character specifier. This streamlined approach is preferred when working with quantitative data, as it reduces code clutter and potential input errors.

Consider a scenario where we need to quickly input sports statistics--specifically, points, assists, and rebounds--for a small group of players. Since all these metrics are quantitative, they will be defined as **numeric variables**. The following comprehensive code block demonstrates how to structure the DATA step, define the variables, supply the data via **datalines**, and then immediately verify the resulting dataset structure using the PROC PRINT procedure. The inclusion of comments (starting with `/*`) provides clarity regarding the purpose of each section of the script.

```
/*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;
```

Upon execution, SAS processes the DATA step, reads the nine lines of raw data, and constructs a temporary or permanent dataset named `original_data`. The subsequent PROC PRINT command then displays the contents of this newly created structure, confirming that all data was input correctly. The resulting structure, as seen in the output image below, clearly shows three columns, each populated with numeric values, ready for statistical analysis or reporting. This example solidifies the understanding that for numeric data, simplicity in the **INPUT statement** is key to rapid data entry.

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 output confirms the successful ingestion of the raw data. The result is a dataset comprising three numeric variables: `points`, `assists`, and `rebounds`. This demonstration underscores the ease with which small datasets can be generated directly within the SAS environment using the **datalines statement**, providing immediate accessibility for subsequent procedural steps.

Example 2: Constructing Datasets with Mixed Data Types

Real-world data seldom consists purely of numeric values; most datasets involve a mix of quantitative data (like scores or counts) and qualitative identifiers (like names, teams, or categories). When utilizing the datalines statement to input mixed data, the definition of character variables becomes critically important. Any variable containing letters, special symbols, or intended textual identifiers must be explicitly designated as a character type using the **$** suffix in the INPUT statement. Failing to do so will result in SAS attempting to read text as numbers, causing data corruption or missing values.

To illustrate this necessity, let's expand the previous example to include identifying information: the team identifier (`team`) and the player's position (`position`). Both of these variables are categorical and contain alphabetical characters, meaning they must be declared as **character variables**. The subsequent variables, `points` and `assists`, remain numeric. The code below demonstrates the necessary configuration of the **INPUT statement** to handle this mixed-type structure successfully. Note the placement of the dollar sign immediately following the character variable names.

```
/*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;
```

The execution of this DATA step successfully reads the data. The data block contains the raw entries, where the first two fields (Team and Position) are read as text, and the last two fields (Points and Assists) are read as numbers. The resulting dataset, viewed using PROC PRINT, visually confirms that all four variables have been correctly imported and stored. The ability to seamlessly integrate different data types within a single **datalines block** highlights the versatility of this input method, provided the variable types are accurately specified upfront in the **INPUT**

statement.

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

Verification and Inspection using PROC CONTENTS

After creating any dataset, particularly one generated from raw, inline data, it is a crucial best practice to verify the structure and attributes of the resulting dataset. While PROC PRINT shows the data values, it does not explicitly confirm the internal data type assigned by SAS. To definitively check how SAS has categorized each variable--as either numeric or character--we utilize the **PROC CONTENTS** procedure. This procedure provides metadata about the dataset, including variable names, labels, formats, and, most importantly, their types.

We can use the **proc contents** function to check the type of each variable, ensuring that our definitions in the **INPUT statement** were correctly interpreted by the SAS compiler. This confirmation step is especially vital in complex programming environments where downstream analysis relies heavily on accurate data typing (e.g., using a character variable in a calculation would halt the program).

proc contents data=original_data;

The output generated by **PROC CONTENTS** provides a detailed summary of the dataset structure. This summary is the ultimate source of truth regarding the dataset's characteristics. By reviewing the output, a user can confirm that the variables designated with the **$** sign (`team` and `position`) have been correctly established as Type: Char, while those without the **$** sign (`points` and `assists`) are correctly listed as Type: Num. This verification step completes the data creation cycle, guaranteeing that the dataset is structurally sound and ready for advanced statistical procedures.

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

From the output provided by **PROC CONTENTS**, we can clearly see that **team** and **position** are properly recognized as character variables, indicated by the 'Char' type designation. Conversely, **points** and **assists** are confirmed as **numeric variables** ('Num' type). This successful verification confirms the proper implementation of the **INPUT statement** and the integrity of the data structure defined using the datalines statement.

Understanding Limitations and Alternatives

While the **datalines statement** is exceptionally useful for small, structured data inputs, it is important to recognize its limitations and when alternative input methods, like external file importing, are more appropriate. The primary constraints of **datalines** relate to data volume, complexity, and the handling of specialized characters or formats. Since the data is embedded directly in the program code, maintaining hundreds or thousands of lines becomes cumbersome and inefficient for version control and editing.

Furthermore, the default behavior of **datalines** assumes that data fields are separated by single spaces (list input). Handling data that contains embedded spaces (e.g., names like "New York"), or dealing with missing values that must be represented by a period (`.`), requires careful adherence to the input rules. For large or highly formatted data sets, utilizing external files (CSV, XLSX) combined with `INFILE` or `PROC IMPORT` statements is the standard and recommended practice within SAS, offering greater flexibility and robustness in data handling. However, for quick testing and small examples, the simplicity and self-contained nature of **datalines** make it an invaluable tool for any SAS programmer.

Additional Resources for SAS Programming

To further deepen your understanding of data manipulation and procedure usage within the SAS environment, the following tutorials explain how to perform other common tasks and procedures: