How to Easily Import CSV Files into SAS

Authored by

stats writer

December 1, 2025

The ability to efficiently integrate external data is foundational to effective statistical analysis. Among the most common external formats is the CSV (Comma Separated Values) file, a plain-text format widely used for storing tabular data. This guide provides a comprehensive, step-by-step approach to importing CSV files into SAS (Statistical Analysis System), ensuring your data is ready for immediate processing and manipulation.

While SAS offers several methods for data ingestion, the most robust and commonly employed technique for batch processing and repeatable code execution is the use of the proc import procedure. This powerful procedure streamlines the tedious tasks of data definition, field type identification, and path specification, making the transition from flat file to SAS data set seamless. Understanding the syntax and its various options is essential for handling the diverse complexities that external data often presents.

Understanding the PROC IMPORT Procedure

The cornerstone of importing external flat files, including CSVs, into the SAS environment is the proc import procedure. This procedure is designed specifically for reading standard database files and various delimited text formats, automating the critical step of generating the necessary Data Step code behind the scenes. By using proc import, analysts can bypass manual data definition, which significantly reduces the potential for human error and speeds up workflow execution.

The basic structure of the proc import statement requires the user to define three primary pieces of information: the location of the source file, the format of the source file, and the name and location of the resulting SAS data set. It is crucial that the file path specified in the code is fully accessible by the SAS session being executed, whether locally or on a server environment.

When preparing to run this code, ensure that the input CSV file adheres to a consistent structure. While the procedure is highly adaptive, unexpected characters, inconsistent field counts, or non-standard encoding can lead to truncation errors or incorrect variable typing upon import. Always examine the first few rows of your source file to confirm its integrity and structure before initiating the import process.

PROC IMPORT Standard Syntax

The core functionality of importing a CSV relies on a set of standardized keywords and parameters. Below is the canonical structure for importing a generic file named `my_data.csv`, followed by a detailed explanation of each required option. Mastering this fundamental structure is the first step toward advanced data management in SAS.

```
/*import data from CSV file called my_data.csv*/
proc import out=my_data
```

```
datafile="/home/u13181/my_data.csv"
dbms=csv
replace;
getnames=YES;
run;
```

This procedure provides high flexibility through various statements and options, allowing users to precisely control how the external data is mapped to SAS variables. By default, the procedure attempts to infer the data types (numeric, character, or date) based on the observed values, but these inferences can be fine-tuned using additional parameters not shown in the basic example.

Detailed Explanation of PROC IMPORT Options

Each keyword within the standard proc import statement serves a critical function in defining the source, destination, and behavior of the import process. Understanding these parameters is vital for successful data transfer and management, especially when dealing with complex or large-scale data files.

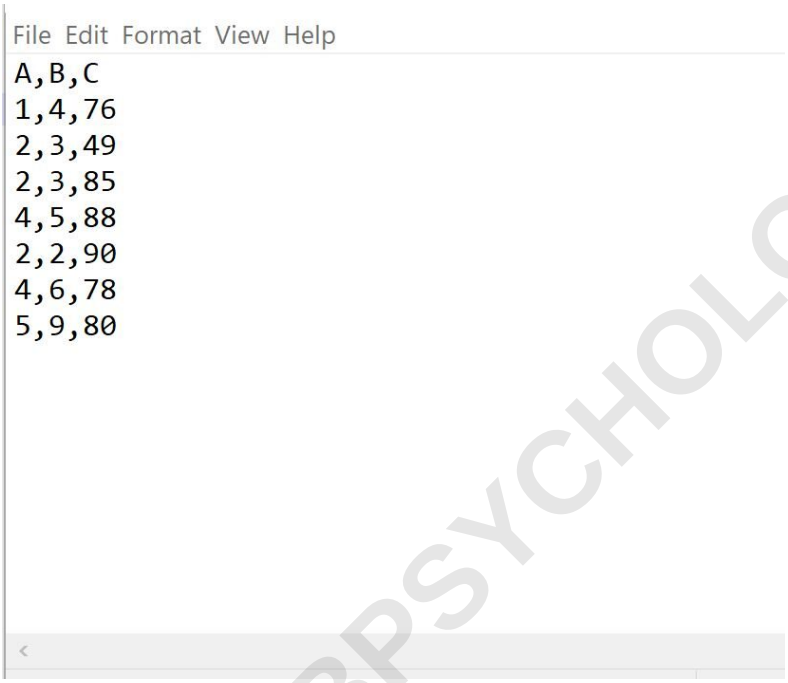
OUT: Specifies the name of the output SAS data set that will be created upon successful import. This parameter is mandatory. If a two-level name (e.g., `library.dataset_name`) is not specified, SAS defaults to saving the file in the temporary `WORK` library.
DATAFILE: Indicates the full path and filename of the external file you wish to import. This path must be enclosed in quotation marks and accessible by the SAS session.
DBMS: This option defines the format or type of the file being imported. For standard comma-separated files, setting this option to `CSV` is typically sufficient. This tells SAS to utilize its internal engine designed to handle CSV formatting rules, including handling quoted strings.
REPLACE: This optional statement instructs SAS to overwrite the output data set (specified by `OUT`) if a SAS data set with the same name already exists in the specified library. Using `REPLACE` is highly recommended to avoid run-time errors due to existing files during iterative development or testing.
GETNAMES: This is a crucial statement that determines whether the first row of the input file should be treated as variable names or as the first row of data. Setting `GETNAMES=YES` (or `GETNAMES=NO`) directly impacts how SAS structures the resulting data set.

Understanding when to use `GETNAMES=YES` versus `GETNAMES=NO` is paramount. If your external data source includes a header row containing descriptive labels for each column, use `YES`. If the file contains only raw observation data starting from the very first line, use `NO`, and SAS will automatically assign generic names like VAR1, VAR2, and so forth.

Example 1: Standard Import with Header Row

In the most common scenario, your external data file will be structured correctly, utilizing commas as field separators and including a header row that identifies the variables. Consider a standard CSV file named **my_data.csv** containing basic demographic or experimental results. Since this file includes clear column headers, we must instruct SAS to treat the first line differently from the subsequent data rows.

The input file `my_data.csv` is visualized below. Note the distinct variable names in the first row: `Name`, `Age`, and `Score`.



To successfully bring this data into SAS and name the resulting data set `new_data`, we execute the following procedure. We explicitly set the `GETNAMES=YES` option, confirming that SAS should utilize the descriptive labels provided by the source file for naming the variables within the new data set. We also include a `proc print` step immediately after the import to verify the successful creation and structure of the imported data.

```
/*import data from CSV file called my_data.csv*/
proc import out=new_data
datafile="/home/u13181/my_data.csv"
dbms=csv
replace;
getnames=YES;
```

```
run;

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

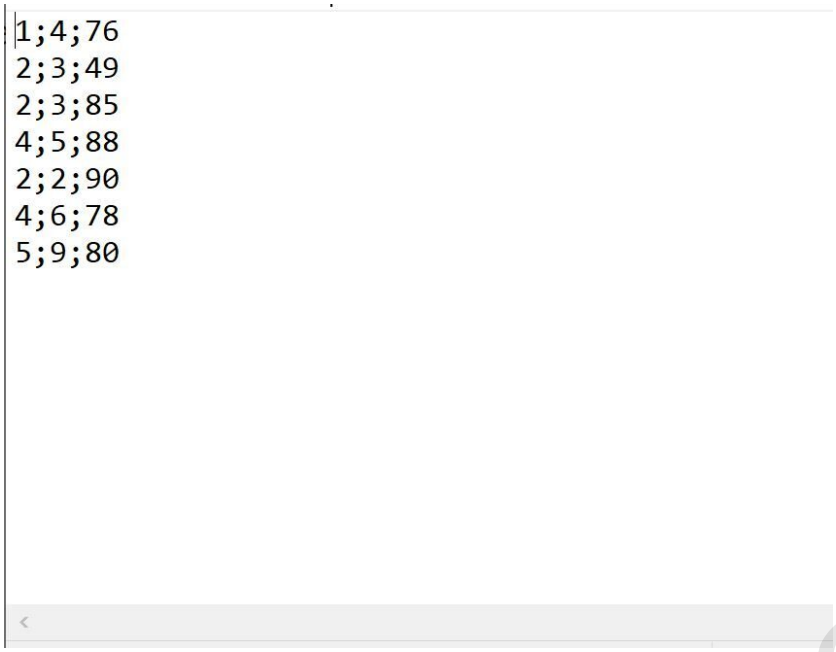
Upon execution, the SAS output confirms that the data has been loaded correctly. Notice how the variable names in the output--Name, Age, Score--exactly match the header row of the original CSV file, demonstrating the functionality of `GETNAMES=YES`. This validation step is crucial for ensuring data integrity before proceeding with complex analysis.

Obs	A	B	C
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90
6	4	6	78
7	5	9	80

Example 2: Importing Data with Custom Delimiter and No Header

Not all data files strictly follow the Comma Separated Values convention. Sometimes, particularly in international settings or specialized system exports, fields may be separated by other characters, such as tabs, pipes, or semicolons. Furthermore, the file might contain only raw data rows without a descriptive header. This example demonstrates how to adapt the proc import procedure to handle these non-standard characteristics efficiently.

We are working with a file named `data.csv`, which, despite its extension, utilizes the semicolon (`;`) as its primary delimiter, and it lacks an initial row of variable names. The structure of this challenging input file is shown below, highlighting the semicolon separators.



To successfully import this data, two specific adjustments are necessary within the proc import code. First, we must introduce the `DELIMITER` statement and specify the semicolon character (`DELIMITER=";"`). This tells SAS exactly how to parse the boundaries between data elements. Second, we must change the `GETNAMES` option to `GETNAMES=NO`, instructing SAS that all lines contain observation data, thus prompting the system to assign default variable names.

```
/*import data from CSV file called data.csv*/
proc import out=new_data
datafile="/home/u13181/data.csv"
dbms=csv
replace;
delimiter=";";
getnames=NO;
run;

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

The output confirms that the data has been imported successfully, recognizing the custom delimiter. Since `GETNAMES=NO` was specified, SAS assigns sequential default variable names (VAR1, VAR2, VAR3, etc.) to the columns. If required for analysis, these variables should be renamed immediately after the import using a subsequent Data Step or `PROC DATASETS` statement to ensure clarity in the analytical process.

Obs	VAR1	VAR2	VAR3
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90
6	4	6	78
7	5	9	80

Advanced Considerations and Best Practices

While proc import handles the vast majority of standard CSV files effectively, analysts often encounter edge cases that require greater control over the data ingestion process. One common challenge involves data type handling. By default, SAS scans a limited number of rows to determine if a column should be numeric or character. If it encounters a character value late in the file for a column it previously determined was numeric, it may misinterpret the data, leading to missing values or truncation.

For data integrity, it is sometimes preferable to use the `DATA STEP` with `INFILE` and `INPUT` statements instead of proc import, especially when dealing with extremely large files or highly inconsistent data. However, if you must use the procedure, ensure that your data is clean before import. A universal best practice is to always perform a frequency check or summary statistics check immediately after import to verify that the variable types and data counts match expectations.

Another critical consideration is handling international character sets. If your CSV file uses encoding formats like UTF-8, you must ensure that your SAS session is also running under the appropriate encoding configuration. Importing a UTF-8 file into a Latin1 session, for instance, will result in corrupted characters. While the core procedure syntax remains the same, configuring the correct session encoding is a prerequisite for seamless global data handling.

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