Learning to Import CSV Data Files into SAS: A Step-by-Step Guide

Authored by

Mohammed looti

November 1, 2025

In the realm of statistical analysis and enterprise data management, the ability to seamlessly integrate external data sources into your analytical environment is fundamental. For users of the powerful statistical software SAS (Statistical Analysis System), one of the most frequent requirements is importing data stored in the standardized Comma Separated Values (CSV) format. Fortunately, SAS provides a highly efficient and standardized mechanism for this purpose: the **proc import** statement.

This critical procedure is specifically engineered to handle common delimited file types, quickly transforming them into a native SAS dataset. Generating a SAS dataset is an essential prerequisite for all subsequent data preparation, cleaning, reporting, and advanced analysis steps. Consequently, achieving proficiency in the fundamental syntax and operational parameters of **proc import** is a cornerstone skill for any data professional utilizing the SAS platform.

Mastering the PROC IMPORT Procedure for Data Ingestion

The PROC IMPORT procedure represents the official and recommended standard for reading external delimited files, such as CSVs or general text files, and converting their structure into a temporary or permanent SAS dataset. A key advantage of using **proc import** over manually coding a complex DATA Step is its automated intelligence. The procedure automatically attempts to infer the correct data type (such as character or numeric) and the appropriate length for each variable by examining the content of the source file.

Despite its automation, successful execution of **proc import** relies on the correct specification of a few core options. These options are necessary to precisely locate the source file, specify its format, and define the naming conventions for the resulting SAS dataset. A thorough understanding of these parameters is crucial for ensuring a smooth and accurate data transfer, minimizing common errors related to incorrect file paths or misassigned variable definitions.

The flexibility inherent in the **proc import** statement allows it to manage a wide array of data scenarios. This includes handling input files that either contain or lack header rows, as well as those utilizing non-standard delimiters--such as tabs, semicolons, or pipe characters--instead of the default comma separator expected in typical CSV files. This adaptability makes it the go-to tool for initial data loading.

Deconstructing the Essential PROC IMPORT Syntax

The foundational structure required to use **proc import** for loading data from an external CSV file into a SAS session is highly systematic. It mandates the specification of several key arguments that guide the software through the import process. The following syntax block illustrates the fundamental command structure necessary to execute a successful import operation:

```
/*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;
```

Each component within this procedure plays a vital role in directing SAS on how to process the input file and manage the newly created dataset. Understanding the function of each option is key to manipulating the import behavior, especially when dealing with varied data sources. Below is a detailed explanation of the essential options required for most CSV imports:

out: This argument designates the name that will be assigned to the resultant SAS dataset upon completion of the import procedure. If the user does not specify a library reference (e.g., `LIBNAME.dataset`), the dataset is automatically stored as a temporary file within the default `WORK` library.
datafile: Functionally, this is the most critical parameter, as it must contain the precise location, or absolute path, of the external CSV file intended for importation. Specifying the correct file path is mandatory for the procedure to initiate and execute successfully.
dbms: Standing for Database Management System, this option specifies the format of the external file being imported. By setting **dbms=csv**, we explicitly instruct SAS that the input source is a standard comma-delimited text file, allowing it to apply the appropriate default parsing rules.
replace: The **replace** option acts as a necessary safeguard during repeated execution. If a SAS dataset with the exact name specified in the `OUT=` option already exists in the designated output location, this directive ensures that the existing file is safely overwritten by the newly imported data, preventing errors related to duplicate names.
getnames: This parameter controls how the first row of the input file is interpreted. Setting **getnames=YES** is the standard approach when the CSV includes descriptive column headers, ensuring these headers become the variable names in the SAS dataset. Conversely, if set to **NO**, SAS will automatically assign generic names like VAR1, VAR2, and so forth.

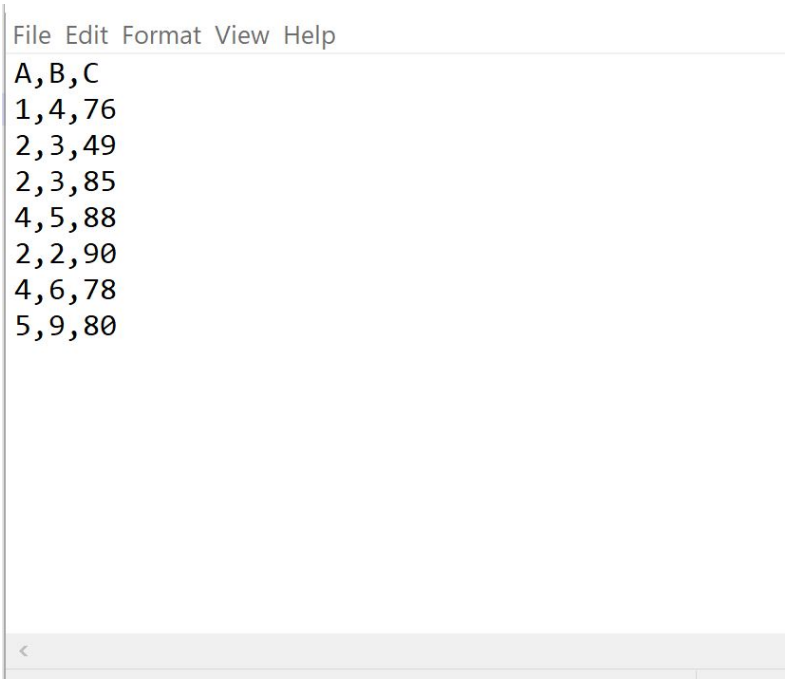
By accurately defining these parameters, data analysts can guarantee that external data is transformed efficiently and accurately into a usable SAS format, immediately preparing it for subsequent analytical operations.

Case Study 1: Importing a Standard CSV File with Headers

To practically demonstrate the primary functionality of PROC IMPORT, let us consider a common real-world scenario involving a dataset that adheres to standard conventions, meaning it includes

header rows and uses commas as field delimiters. Assume we are working with an external file named **my_data.csv** containing typical observational data that needs to be loaded into our active SAS session for processing.

The internal structure of **my_data.csv** is visualized below. Note the explicit labels for the columns--`ID`, `Score`, and `Group`--in the first row. This configuration necessitates the explicit inclusion of the `GETNAMES=YES` option within the import procedure to correctly map these headers to variable names.



We can utilize the following SAS code block to import this specific file, instructing SAS to name the resulting dataset **new_data**. The code is carefully constructed to ensure that the header row is correctly interpreted as variable names, which remains the most typical requirement for statistical data analysis workflows.

```
/*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;
```

Following the execution of the import procedure, the subsequent **proc print** statement allows the user to immediately verify the structure of the newly generated SAS dataset, **new_data**. This verification step confirms that the data has been imported successfully and that the essential data integrity, including correct variable naming, has been fully preserved.

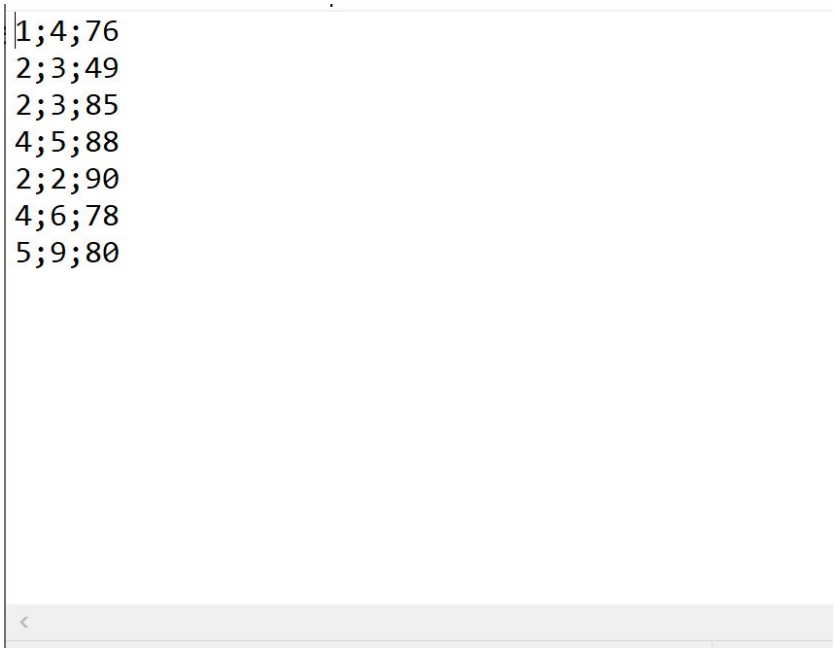
Obs	A	B	C
1	1	4	76
2	2	3	49
3	2	3	85
4	4	5	88
5	2	2	90
6	4	6	78
7	5	9	80

As clearly demonstrated by the output table, the variable names (ID, Score, Group) were accurately extracted from the first row of the source CSV file, thereby validating the effective implementation of the **getnames=YES** option for standard file imports.

Case Study 2: Managing Non-Standard Delimiters and Missing Headers

While the default functionality of PROC IMPORT excels at handling standard comma-delimited files, data encountered in the real world often presents structural variations that require procedural adjustments. Two frequent challenges involve files that lack a descriptive header row and files that employ a non-comma character--such as a semicolon (common in European data exports) or a tab--as the primary field separator.

Let us examine a second file example, named **data.csv**, which features the structure displayed below. It is important to observe two deviations from the standard: first, the data values are separated by semi-colons, and second, there is no explicit descriptive header row to identify the variables contained within the columns.



To successfully ingest this non-standard file format, we must introduce modifications to the standard **proc import** command. Specifically, we must leverage the **delimiter** option and simultaneously set the **getnames=NO** parameter. The powerful **delimiter** option enables the user to explicitly specify the exact field separator character used in the source file, which is absolutely necessary when the data does not conform to the expected comma separation.

The following code snippet details the essential adjustments required to accurately handle this specific, challenging data configuration. This ensures that the data is correctly parsed despite its non-standard structure:

```
/*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;
```

By setting **delimiter=";"**, SAS is correctly instructed to identify the distinct data fields based on the semi-colon separators. Furthermore, because **getnames=NO**, the first row of data is correctly

treated as observational input rather than being erroneously used as variable labels. The resulting SAS output confirms the successful and accurate import of the non-standard file:

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

Since no explicit variable names were provided via a header row, SAS automatically defaults to assigning sequential, generic identifiers, resulting in variables labeled VAR1, VAR2, and VAR3. Data professionals typically follow this type of generic import with a dedicated DATA Step to rename these variables to more meaningful and descriptive identifiers for analysis.

Advanced Considerations: Data Type Inference and Alternative Methods

While the PROC IMPORT procedure is exceptionally efficient for most standard delimited files, data preparation workflows involving highly complex structures or stringent data typing requirements may demand more advanced attention. A crucial aspect to consider is the automatic data type inference capability of SAS.

SAS attempts to determine whether a column should be designated as character or numeric based on the sampled data. A common issue arises when a column primarily contains numbers but includes even a single non-numeric value (such as a text code or a missing value represented by 'N/A'). In such instances, SAS may conservatively default the entire column to a character type, which subsequently complicates or prevents essential mathematical operations and statistical analysis.

For highly customized import scenarios--particularly those demanding rigorous control over data formats, variable lengths, or complex conditional reading logic--the automatic nature of **proc import** may prove insufficient. In these specialized cases, the traditional DATA Step, utilizing the `INFILE` statement combined with explicit `INPUT` specifications, provides the necessary granular control. This robust alternative empowers the user to manually define data types, enforce specific formats, and effectively manage complex formatting issues that automatic procedures might otherwise misinterpret.

Finally, regardless of the method chosen, always ensure the **datafile** path is absolutely correct and accessible by the SAS session running the procedure. In modern server-based environments, this path must reference the server's file system, not the user's local machine or desktop. Utilizing absolute paths (paths starting from the root directory) is highly recommended over relative paths to prevent ambiguity and ensure reliable execution across different environments.

Summary of Best Practices for SAS Data Import

Leveraging the **proc import** procedure remains the fastest, most effective, and standardized method for introducing external delimited data, such as CSV files, into the SAS environment. By diligently managing a small set of procedural options, users can rapidly transform raw data into impeccably structured SAS datasets that are immediately ready for rigorous analysis.

A successful data importation strategy hinges on adhering to these key procedural takeaways:

Always specify the **datafile** path accurately using the absolute file location.
Use **dbms=csv** when importing standard comma-delimited files.
Control the naming and location of the resulting dataset via the **out** option.
Utilize **getnames=YES** exclusively when the source file contains descriptive headers; conversely, use **getnames=NO** when headers are absent.
For non-comma field separators, explicitly define the character used via the crucial **delimiter** option.

While **proc import** capably handles the vast majority of standard import tasks, it is prudent to remember that the DATA Step paired with `INFILE` and `INPUT` statements remains the ultimate and most versatile tool for complex or highly customized data reading scenarios, offering complete control over the input process.

Additional Resources for SAS Programming

The following resources provide further tutorials explaining how to perform other common data manipulation and analysis tasks in SAS: