How to Easily Remove Commas from Strings in SAS

Authored by

stats writer

November 20, 2025

Data cleaning is a critical step in any analytical workflow, and frequently, this process involves removing unwanted characters, such as commas, from text variables. In the SAS environment, there are several powerful functions available for efficient string manipulation. While functions like TRANWRD (Translate Word) can be utilized, the most straightforward and versatile method for single-character removal involves combining the power of the **TRANSLATE** and **COMPRESS** functions.

The primary challenge when dealing with numeric data stored as text or text fields containing stray punctuation is ensuring data integrity before analysis. Commas often appear in imported data due to manual input errors or formatting issues designed for human readability (e.g., thousands separators). If these commas are not removed, they can prevent SAS from correctly interpreting the variable as a numeric type, leading to computational errors or incorrect results during data processing steps.

Understanding String Cleaning Needs in SAS

When approaching character cleanup in SAS, it is essential to distinguish between removing single, specific characters and replacing whole words or phrases. For simple character removal, especially delimiters like commas, the **TRANSLATE** function offers superior performance and clarity compared to functions designed for complex pattern matching or word replacement. This high-efficiency approach is vital when working with large datasets where optimization is necessary.

While the TRANWRD function is a common utility for replacing substrings, it requires specific syntax where the old string is replaced by a new string. For example, to remove a comma, you would replace ',' with ''. However, the standard practice in SAS programming for removing characters efficiently relies on functions specifically designed for character-level transformations, which we will detail in the subsequent sections. We aim to present the most recommended method that ensures clean, concise, and professional code.

The Primary Method: Combining TRANSLATE and COMPRESS

The standard and most efficient technique for removing a specific single character, such as a comma, from a string in SAS involves nesting the **TRANSLATE** function within the **COMPRESS** function. This combination is powerful because **TRANSLATE** handles the replacement, and **COMPRESS** ensures any resulting extraneous spaces are eliminated, leading to a perfectly cleaned variable.

The core mechanism of this method relies on using **TRANSLATE** to replace the comma (or any target character) with a space. While simply replacing the character with a null string (`''`) might seem intuitive, **TRANSLATE** requires the replacement character to be present in the target string. By substituting the comma with a space, we prepare the string for the subsequent cleaning step

handled by **COMPRESS**.

The general syntax for applying this transformation within a SAS DATA step is straightforward. We are creating a new variable or overwriting an existing one, applying the nested functions to every observation in the source dataset. This approach is highly scalable and ensures consistency across all rows, mitigating the risks associated with manual data cleaning.

Observe the fundamental SAS code required to execute this operation on a variable named `string_var`:

```
data new_data;
set original_data;
string_var = compress(translate(string_var,"",','));
run;
```

This snippet demonstrates how to create a new dataset, `new_data`, based on `original_data`, where every instance of a comma within the `string_var` column is systematically removed. It is a robust method favored by professional SAS programmers for its efficiency and clear intent. The next sections will dissect how each function contributes to the final clean output.

Detailed Breakdown of the TRANSLATE Function

The **TRANSLATE** function is specifically designed for character-level manipulation, allowing a user to change every occurrence of one set of characters to another set of characters within a source string. Its syntax typically requires three arguments: the source string, the target replacement character(s), and the character(s) to be replaced. For our purpose of comma removal, we specify the comma (`','`) as the character to be replaced.

In the expression `translate(string_var, "", ',')`, we are instructing SAS to find all instances of the third argument (`','`) within the first argument (`string_var`) and replace them with the corresponding character(s) in the second argument (`""`). When the second argument is a null string (`""`) in the context of **TRANSLATE**, it defaults to replacing the target character with a blank space. This is a crucial detail: **TRANSLATE**, unlike **COMPRESS**, does not truly delete characters; it substitutes them.

If the comma was located immediately adjacent to another character, its replacement with a space can introduce undesirable leading, trailing, or embedded blank spaces into the string. For instance, if the original value was `"King,s"`, applying **TRANSLATE** results in `"King s"`. If the original was `",Mavs,"`, the result is `" Mavs "`. While the comma is gone, the presence of these new spaces means the variable is still not optimally clean for analysis or conversion. This necessity for space

removal immediately highlights why **COMPRESS** is essential in this compound cleaning strategy.

Leveraging the COMPRESS Function for Final Cleaning

The **COMPRESS** function is arguably the most powerful general-purpose cleaning utility in SAS. It is designed to remove specified characters, including blank spaces, from a string. When used without any additional arguments, **COMPRESS** automatically removes all leading, trailing, and multiple embedded spaces, which is precisely what is needed after the **TRANSLATE** function has done its work of substitution.

By nesting the **TRANSLATE** operation inside the **COMPRESS** function--as seen in the structure `compress(translate(...))`--we ensure that the intermediate spaces created by **TRANSLATE** are immediately and completely eradicated. This results in a dense string containing only the desired alphanumeric characters. If the resulting value needs to be converted to a numeric type, this dual function approach is highly reliable because it guarantees a clean output free of any non-numeric punctuation or spaces.

If you were only interested in removing spaces, the **COMPRESS** function alone would suffice. However, since the goal is to remove commas, we must first substitute them using **TRANSLATE** before leveraging **COMPRESS** to finalize the cleaning process. This methodology provides a comprehensive and elegant solution for character removal in SAS data handling.

Defining the Sample Dataset in SAS

To illustrate this technique, let us work with a sample dataset containing common data quality issues, specifically misplaced commas within team names. This scenario is typical when data has been manually entered or sourced from systems that do not enforce strict naming conventions.

Suppose we have the following sample dataset in SAS that tracks basketball team scores. Notice the inconsistencies and embedded commas within the `team` variable:

```
/*create dataset*/
data my_data;
input team $ points;
datalines;
,Mavs, 113
Pacers 95
,Ca,vs 120
Lakers 114
Heat 123
King,s 119
```

```
Raptors 105
,Hawks 95
Ma,gic 103
Spu,,rs 119
;
run;

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

Executing this code and viewing the output using `proc print` reveals the raw state of the data, clearly showing the misplaced commas:

Obs	team	points
1	,Mavs,	113
2	Pacers	95
3	,Ca,vs	120
4	Lakers	114
5	Heat	123
6	King,s	119
7	Raptors	105
8	,Hawks	95
9	Ma,gic	103
10	Spu,,rs	119

As evident in the output, several observations in the **team** column contain leading, trailing, or embedded commas. Processing or merging this variable would be impossible without first performing rigorous cleanup. Our goal is to transform `",Mavs,"` into `"Mavs"` and `"Spu,,rs"` into `"Spurs"`, among others.

Applying TRANSLATE and COMPRESS to the Data

We will now apply the combined **TRANSLATE** and **COMPRESS** logic within a new DATA step to correct the contents of the `team` variable. We are creating a new dataset, `new_data`, to hold the cleaned results while preserving the original `my_data` dataset for verification purposes.

The syntax below mirrors the general principle discussed earlier, specifically targeting the `team` variable for comma removal. The variable assignment statement `team =`

`compress(translate(team,"",','));` is the core of the transformation:

```
/*create new dataset where commas are removed from each string in team column*/
data new_data;
set my_data;
team = compress(translate(team,"",','));
run;

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

After executing the DATA step and running the final `proc print`, the output confirms that the cleaning process was successful. All commas, regardless of their position (beginning, middle, or end of the string), have been completely removed, and the resulting team names are correctly formatted:

Obs	team	points
1	Mavs	113
2	Pacers	95
3	Cavs	120
4	Lakers	114
5	Heat	123
6	Kings	119
7	Raptors	105
8	Hawks	95
9	Magic	103
10	Spurs	119

This result confirms the efficacy of nesting these two functions. The process ensured that `"Spu,,rs"` became `"Spurs"`, demonstrating that both functions handled the replacement of multiple adjacent commas and the subsequent removal of the resulting spaces flawlessly.

Dissecting the Function Execution Order

Understanding the order of operations within the nested functions is crucial for mastering this SAS technique. Since the functions are nested, the inner function executes first, passing its result to the outer function for subsequent processing. This ensures a staged cleanup process:

The **TRANSLATE** function (inner operation) scans the original string (e.g., `",Ca,vs"`) and replaces every comma (`','`) with a blank space, yielding an intermediate string (e.g., `" Ca vs"`).
The resulting intermediate string is then passed to the **COMPRESS** function (outer operation), which interprets the blank space as a character to be removed by default, eliminating all spaces and collapsing the string to its clean final form (e.g., `"Cavs"`).

This two-step approach is superior to attempting complex logic with a single function. If we tried to use only **TRANSLATE** and replaced the comma with a true null character, the behavior might vary depending on the SAS version and environment, whereas the space-replacement followed by **COMPRESS** is uniformly reliable across all standard SAS implementations.

Alternative Approaches: Using the TRANWRD Function

While the **TRANWRD** function is a valid alternative for removing substrings, it is typically less efficient for single-character removal than the **TRANSLATE** approach presented above. The TRANWRD function is designed to replace every occurrence of a specified substring (the "word" or phrase) with another specified substring.

To use TRANWRD for removing commas, the syntax would look like this: `team = tranwrd(team, ',', '');`. Here, the comma (`','`) is replaced by a zero-length string (`''`), effectively deleting it. This method seems simpler because it requires only one function call, unlike the combined **TRANSLATE**/**COMPRESS** method.

However, a crucial limitation of TRANWRD is its handling of multiple delimiters. If the original string contains multiple adjacent commas (e.g., `"Spu,,rs"`), TRANWRD will successfully remove all of them, yielding `"Spurs"`, just like the combined function method. The primary reason developers often prefer the **TRANSLATE**/**COMPRESS** combination is its historical reputation for raw speed when dealing with very large datasets and its flexible applicability to removing multiple different characters simultaneously by listing them in the arguments.

Summary and Best Practices for Data Quality

In summary, the most robust and standard method for removing commas (or any single character) from a string variable in a SAS DATA step is the nested application of **COMPRESS** and **TRANSLATE**. This technique guarantees the removal of the targeted character while simultaneously cleaning up any resulting blank spaces, ensuring the final output is optimized for subsequent analysis or type conversion.

When implementing data cleaning procedures, always remember to test the code on a small subset of the data before applying it broadly. Ensure that the resulting variable length is sufficient, especially if you are creating a new variable. While character removal generally shrinks the string,

maintaining consistency in data type and length definition is a key best practice in SAS programming.

For more detailed documentation regarding the utility functions discussed here, please refer to the official SAS documentation.

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