SAS: Remove Commas from String

Authored by

Mohammed looti

November 16, 2025

Master Data Cleansing: Removing Commas from SAS Strings

In the realm of statistical analysis, ensuring **data integrity** is non-negotiable. Raw datasets frequently contain unwanted characters, such as extraneous commas, that can severely interfere with processing, computation, or visualization. Within the SAS environment, the most efficient and powerful method for cleansing a character string of these unwanted delimiters involves the strategic pairing of the **TRANSLATE** and **COMPRESS** functions. This combined technique is a fundamental skill for any SAS programmer, guaranteeing that data preparation is executed swiftly and accurately, thus making variables suitable for subsequent analytical procedures. Understanding the unique operational characteristics of these two functions, and how they complement each other, is the cornerstone of effective string manipulation in SAS.

The **TRANSLATE** function is primarily designed for character replacement: converting every instance of a 'source' character to a 'target' character within a string. A common mistake is attempting to use **TRANSLATE** alone to eliminate a comma by replacing it with an empty string (`""`). However, SAS requires a designated replacement character. When the replacement value is set to an empty string, SAS interprets this instruction as replacing the character with a single, crucial artifact: a **blank space**. This introduction of unwanted whitespace demands an immediate second step to finalize the cleansing process, as leaving these spaces would lead to inaccurate data storage and comparison errors.

Consequently, the standard industry best practice dictates that **TRANSLATE** must always be coupled with the **COMPRESS** function for complete character removal. While **TRANSLATE** isolates the target character (the comma) by turning it into a space, **COMPRESS** specializes in eliminating specified characters, most notably whitespace. By using **TRANSLATE** to convert the comma to a space, and then deploying **COMPRESS** (with its default setting) to eradicate all resulting spaces, we achieve a pristine output. This two-step process ensures that the original commas are entirely removed without leaving behind any lingering structural remnants.

Implementing the Robust TRANSLATE and COMPRESS Solution

To reliably strip all commas from a specific variable within a dataset, expert SAS programmers utilize a specific and elegant construct within the Data Step. This sequence reads the raw data, applies the cleaning logic to the variable in question, and then writes the transformed results to a new dataset. This methodology not only cleans the data but also preserves the integrity of the original source data for full auditability. Because of its reliable handling of variable lengths and character positions, this nested function approach is universally regarded as the standard solution for character cleansing in SAS.

The core syntax required for implementing this cleaning process is highly functional and

remarkably concise. It sequentially targets the variable, applies the character translation, and then compresses the intermediate string, assigning the final, clean value back to the variable. It is important to emphasize that this operation only functions on **character variables**. If the variable intended for modification is numeric, a necessary prerequisite step involves converting it to a character type, as SAS string manipulation functions are strictly designed for character data. Ensuring proper variable definition is critical before attempting these transformations.

The following template provides the standard syntax used within a SAS Data Step to execute this highly efficient cleaning operation across an entire dataset:

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

This implementation beautifully illustrates the nested nature of the functions. The TRANSLATE function executes first on the original string, generating an output that is immediately passed as input to the COMPRESS function. The resulting clean string is then stored in the variable `string_var` within the newly created dataset, `new_data`. This single line of code is highly optimized for performance, making it an ideal solution for processing large volumes of data where routine string cleaning is a recurrent requirement.

Dissecting the Syntax: A Step-by-Step Execution Flow

To fully appreciate the efficacy and elegance of this method, we must analyze the assignment statement--`string_var = compress(translate(string_var,"",','));`--by examining the flow of execution, which rigorously proceeds from the innermost function outward.

The process begins with the execution of the TRANSLATE function, which follows the syntax `TRANSLATE(source, to, from)`. In our specific context:

The `source` argument is `string_var`, which holds the raw character data containing the commas.
The `from` character is `','`, representing the specific delimiter we intend to eliminate.
The `to` character is `""`, the empty string, which, as discussed, SAS interprets as replacement with a single **blank space**.

The **TRANSLATE** function systematically replaces every comma located within `string_var` with a single blank space, thereby producing an intermediate string. This intermediate string is now free of commas but contains unwanted spaces where those commas once resided.

Next, the COMPRESS function receives this intermediate string as its sole input argument. The

primary purpose of **COMPRESS** is to remove specified characters from a string. Critically, when **COMPRESS** is invoked with only a single argument (the string itself), it defaults to removing all standard **whitespace characters**, including blanks and tabs. By utilizing `COMPRESS(translated_string)`, we efficiently and completely eliminate the blank spaces that were purposefully introduced by the preceding **TRANSLATE** step, yielding a final, clean string that is entirely free of both commas and extraneous spaces.

This nested procedure represents the fundamental and highly recommended approach in SAS for high-volume character cleansing. It is specifically engineered to overcome the limitations of simple replacement methods, ensuring that the final string value is structurally sound and prepared for accurate comparison, sorting, and concatenation operations within the analytical pipeline.

Practical Demonstration: Setting Up Corrupted Sample Data

To concretely illustrate the power of this cleansing technique, let us consider a typical scenario involving sports data where human input errors have inadvertently introduced commas into team names. Such corrupted data is unsuitable for database joins or accurate reporting. We will construct a sample dataset containing basketball team information, intentionally embedding commas into the `team` variable to accurately simulate real-world data quality issues.

The following SAS code generates the sample dataset named `my_data`. Observe how the values in the `team` variable are deliberately compromised with misplaced commas--including leading commas, embedded commas, and multiple adjacent commas. The `DATA` step establishes the variable structure, and the `DATALINES` statement provides the input records. Following data creation, the `PROC PRINT` statement is executed to display the contents of this initial dataset before any transformation takes place, providing a clear baseline for comparison.

/*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;
```

Upon execution, SAS generates the initial dataset, and the output clearly confirms the presence of these problematic commas within the **team** column. This visual confirmation is vital, as it defines the scope of the data quality issue that our subsequent cleaning code is tasked with resolving. The image below provides a representation of the `PROC PRINT` output, confirming the 'messy' state of the initial data.

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 visually confirmed, entries such as ",Mavs,", ",Ca,vs", and "Spu,,rs" contain commas in various undesirable positions. These non-alphanumeric characters must be systematically eliminated to guarantee accurate categorization, comparison, and eventual joining of this data within the wider analytical workflow. We now proceed to the execution phase, applying the specialized SAS functions to rectify these errors efficiently.

Executing the Transformation and Verifying Clean Results

With the problematic dataset established, the next logical step is to deploy the cleaning strategy using the nested **TRANSLATE** and **COMPRESS** functions. We will create a new dataset, `new_data`, derived from the original `my_data`, and focus the cleaning operation exclusively on the `team` variable. This best practice ensures that the raw source data remains immutable while we

generate a verified, clean version ready for analysis.

The following syntax applies the transformation logic discussed in detail earlier. It commands SAS to sequentially replace all commas in the `team` variable with blank spaces (using **TRANSLATE**) and then immediately remove those resulting blank spaces (using **COMPRESS**). This comprehensive, two-pronged approach guarantees that all instances of the comma character are fully eradicated, irrespective of their original position or multiplicity within the string.

```
/*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;
```

Upon successful execution of this Data Step, the `new_data` dataset is finalized. We use `PROC PRINT` once more to meticulously inspect the transformation results. The output, displayed in the image below, provides conclusive verification that the data cleaning process was entirely successful. Every team name that previously contained commas has been corrected, providing consistent and clean character data perfectly suited for any subsequent analytical procedure.

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

A direct, side-by-side comparison between the initial corrupted dataset and the final resulting dataset powerfully confirms the immense value of this combined function approach. Complex

entries like "Spu,,rs" are now correctly rendered as "Spurs", and the malformed ",Mavs," is now simply "Mavs". This transformation validates the efficiency and unparalleled reliability of using the nested **TRANSLATE** and **COMPRESS** functions for targeted character removal within SAS character variables.

Why TRANSLATE and COMPRESS is Superior to Other Methods

While the implemented code is deceptively simple, the underlying technical rationale warrants a deeper review, particularly regarding why this pairing is superior to other potential SAS string functions. Many users, especially those new to SAS, might initially consider alternatives like **SUBSTR**, **SCAN**, or even complex regular expressions (via **PRXCHANGE**). However, for the straightforward task of eliminating specific, known characters, the **TRANSLATE/COMPRESS** combination offers unmatched performance, readability, and consistency.

The core challenge lies in the fundamental behavior of the **TRANSLATE** function. It is built for character-to-character replacement. When a user specifies an empty replacement string (`""`) for the target character (e.g., the comma), SAS cannot simply delete the character without leaving a placeholder. To maintain structural integrity during the translation process, SAS interprets the empty replacement string as an instruction to substitute the character with the smallest possible non-null entity, which is invariably a single blank space. This is the crucial nuance: if **TRANSLATE(team, "", ',')** were used in isolation, the output would contain embedded spaces corresponding to the original commas (e.g., "Spu,,rs" would become "Spu rs").

This necessity for cleanup makes the **COMPRESS** function indispensable. By default, **COMPRESS** is designed to remove all standard blanks from a string. By applying **COMPRESS** to the output of **TRANSLATE**, we perfectly clean up the intermediate results. The successful sequence of operations is thus guaranteed:

The **TRANSLATE** function replaces every occurrence of the comma delimiter with a blank space, effectively standardizing the unwanted characters into manageable whitespace.
The **COMPRESS** function subsequently removes these newly introduced blank spaces, resulting in a continuous, uninterrupted, and clean final string.

It is important to note that official documentation for the TRANSLATE function provides exhaustive details on how character replacements are handled, especially concerning the interpretation of null or blank replacement strings. A thorough understanding of this technical subtlety is essential for developing highly robust and efficient SAS code.

Conclusion: Adaptability and Next Steps in String Handling

The requirement to cleanse character data by systematically removing unwanted delimiters, such

as commas, is a foundational and frequently encountered operation in effective data management. The nested utilization of the **TRANSLATE** and **COMPRESS** functions offers a reliable, highly performant, and easily maintainable solution within the SAS Data Step environment. For the simple eradication of specific, known characters, this methodology is strongly recommended over more resource-intensive pattern matching or complex parsing techniques.

While the focus of this guide was the removal of commas, the identical syntax is remarkably adaptable and can be easily modified to remove other common delimiters or problematic characters--including semicolons, dollar signs, parentheses, or asterisks. This is achieved simply by altering the third argument (the `from` character set) in the TRANSLATE function. For instance, removing both commas and dollar signs would only require a minor adjustment to `TRANSLATE(team, "", ',$')`.

For analysts facing more sophisticated string cleaning challenges--such as removing all non-numeric characters, managing variable length constraints, or complex pattern extraction--SAS offers a rich library of additional specialized functions. Nevertheless, mastering string manipulation starts with understanding the core utility and efficient interaction of **TRANSLATE** and COMPRESS, which serves as the essential first step toward becoming a proficient SAS data handler.

Additional Resources for Advanced SAS String Manipulation

To continue advancing your proficiency in SAS data manipulation and character handling, the following resources and tutorials delve into related functions and techniques:

Detailed exploration of the **PRXCHANGE** function for robust regular expression-based data cleaning.
Techniques for handling variable length and padding issues in SAS character variables.
Comprehensive guides on utilizing **SUBSTR** and **SCAN** for intricate string parsing and extraction tasks.