Learning SAS: How to Convert Character Variables to Numeric

Authored by

Mohammed looti

November 1, 2025

Understanding the Necessity of Data Type Conversion in SAS Programming

In the complex environment of data management and statistical analysis, raw data rarely arrives in a perfectly structured format. One of the most frequent and critical tasks undertaken by data analysts using SAS is the manipulation of data types. Specifically, converting a character variable, which stores textual information, into a numeric variable is essential for enabling any meaningful mathematical computation or advanced statistical modeling. Without this crucial conversion, quantitative variables mistakenly classified as text strings are rendered useless for aggregation, averaging, or regression analysis.

This requirement often arises when external data sources, such as delimited files or database extracts, misclassify numerical identifiers, monetary values, or measurement data as character strings. Common issues leading to this misclassification include the presence of leading zeros, embedded special characters (like commas or currency symbols), or inconsistent formatting during the data import process. If a variable intended to hold sales volume is stored as a character type, attempting a simple procedure like calculating the mean will result in immediate program failure or incorrect output; thus, mastering the mechanism for accurate type transformation is fundamental to robust SAS programming.

Fortunately, the SAS system is equipped with powerful built-in tools designed precisely for this purpose. The primary instrument for transforming text strings into computable numbers is the **INPUT function**. This function provides a reliable, controlled method for reading the value stored in a character expression and interpreting it numerically based on a specified set of reading instructions, known as an informat. Understanding the syntax and application of this function is the bedrock for cleaning and preparing data for analytical readiness.

The Technical Backbone: Leveraging the INPUT Function and Informats

The **INPUT function** serves as the definitive standard for converting character values to numeric values within the SAS data step. This function operates by demanding two indispensable parameters: the source character variable containing the raw text data, and a valid SAS informat. The informat acts as a crucial instruction set, dictating to SAS exactly how to parse and interpret the character string, ensuring that the resulting numeric value is accurate and correctly formatted.

The execution of the conversion typically occurs during the creation of a new variable within the data step. By assigning the output of the INPUT function to a newly named variable, SAS automatically recognizes this resulting variable as a numeric variable, preparing it for subsequent calculations. This method guarantees that the original, potentially problematic character column remains available for reference while the new, clean numeric column is used for analysis.

The underlying syntax for implementing this critical function is concise and highly effective,

providing clarity within the programming environment. It is typically structured as follows, executed within the body of a data step:

numeric_var = input(character_var, comma9.);

In this illustrative example, `numeric_var` represents the newly defined column destined to hold the numerical data, `character_var` is the original source column, and **`comma9.`** is the specified informat. The `comma9.` informat is highly practical for real-world data, as it is designed to strip out common non-numeric separators such as commas, dollar signs, and certain types of parenthetical expressions often used in financial reporting, thereby facilitating a clean and reliable transformation of complex numeric strings.

Practical Demonstration: Setting Up the Scenario Dataset

To effectively demonstrate the character-to-numeric conversion process, we must first simulate a typical data entry or import error by intentionally creating a dataset where a numerical identifier is stored incorrectly as a character variable. Our scenario involves a simple tracking dataset recording daily sales figures over ten consecutive days. The critical point here is that we define the variable tracking the day number as a character type, preventing its use in sequential analysis.

The following SAS code establishes the initial dataset, named `original_data`. Observe that the `input` statement explicitly includes the dollar sign (`$`) modifier for the `day` variable, instructing SAS to treat its values as text strings, even though they appear to be simple integers. This deliberate misclassification sets the stage for our necessary data cleaning exercise.

```
/*create dataset: 'day' is defined as character ($)*/
data original_data;
input day $ sales;
datalines;
1 7
2 12
3 15
4 14
5 13
6 11
7 10
8 16
9 18
10 24
;
```

```
run;

/*view dataset content*/
proc print data=original_data;
```

Upon execution of this code, the `original_data` table is generated. While a visual inspection of the printed output confirms that the values in the `day` column are indeed single-digit or double-digit numbers, their underlying structural classification within SAS is text. This character designation means that if we tried to, for example, calculate the average day number or sort the dataset by the day variable numerically (which differs from character sorting), the results would be erroneous or the procedure would fail entirely.

Obs	day	sales
1	1	7
2	2	12
3	3	15
4	4	14
5	5	13
6	6	11
7	7	10
8	8	16
9	9	18
10	10	24

Pre-Conversion Diagnosis using PROC CONTENTS

Before any data manipulation or conversion is attempted, professional practice dictates that we formally verify the current data structure. The **PROC CONTENTS** procedure is the ideal tool for this diagnosis, as it generates a comprehensive metadata report for the specified dataset. This report details essential characteristics, including variable names, assigned labels, storage lengths, and, most crucially for our task, the data type--either Character or Numeric.

Running **PROC CONTENTS** on our newly created `original_data` provides the necessary confirmation of the data type assignment, verifying the premise of our conversion challenge. This step is not merely academic; it is vital for debugging complex data pipelines where type misclassification might occur subtly across hundreds of variables.

```
/*display data type for each variable*/
```

proc contents data=original_data;

The resulting output definitively confirms the variable types currently held within the dataset structure.

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	day	Char	8
2	sales	Num	8

As anticipated and confirmed by the metadata, the **day** variable is explicitly categorized as 'Char' (Character), while **sales** is correctly listed as 'Num' (Numeric). This diagnostic step underscores the immediate need for conversion if the `day` variable is to be utilized as a quantitative measure in any subsequent analysis. With the problem confirmed, we can now proceed to implement the solution using the INPUT function.

Executing the Conversion with the INPUT Function

The solution requires the creation of a new data step, which will read the data from the problematic `original_data` and generate a clean version, `new_data`. Within this step, we apply the foundational conversion logic: invoking the **INPUT function** to transform the character string into a true numeric value. We define a new variable, `numeric_day`, specifically to house the output of this successful conversion, drawing its input from the original `day` variable.

For the informat, we again utilize `comma9.`. While the current input data consists only of simple digits (1 through 10), using a robust informat like this is a best practice, ensuring that the code is scalable and capable of handling complex character strings that might include separators or delimiters in future datasets. The width of 9 is more than adequate for the small integers in this example but provides flexibility.

Furthermore, to maintain efficiency and eliminate redundancy, we include the **drop statement** within the data step. This statement explicitly instructs SAS to exclude the original, structurally incorrect `day` variable from the resulting `new_data` dataset, leaving us with only the necessary and correctly typed variables for downstream processing.

```
/*create new dataset where 'day' is numeric*/
data new_data;
set original_data;
```

```
numeric_day = input(day, comma9.);
drop day;
run;

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

The subsequent output from the `proc print` command confirms the successful transformation. The new column, `numeric_day`, now contains the correct, quantitative values, seamlessly replacing the need for the original character column. The use of the `drop` statement, a powerful component of the data step, emphasizes the importance of data hygiene by ensuring that the final dataset only retains variables suitable for analysis.

Obs	sales	numeric_day
1	7	1
2	12	2
3	15	3
4	14	4
5	13	5
6	11	6
7	10	7
8	16	8
9	18	9
10	24	10

Final Validation and Data Integrity

The final and non-negotiable step in any data conversion task is the post-conversion verification of the data types. This validation step confirms that the **INPUT function** executed the transformation exactly as intended, ensuring that the new variable, `numeric_day`, is genuinely ready for all forms of quantitative analysis. Failure to perform this check could lead to incorrect assumptions about the data structure, potentially undermining subsequent statistical conclusions.

We run **PROC CONTENTS** one last time, directing it toward the newly created and cleaned dataset, `new_data`, to inspect its metadata.

```
/*display data type for each variable in new dataset*/
proc contents data=new_data;
```

The resulting output confirms the complete success of the conversion strategy, providing the final stamp of approval on our data cleaning efforts.

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	numeric_day	Num	8
1	sales	Num	8

As clearly demonstrated in the metadata report, the new variable, **numeric_day**, is now correctly designated as a numeric variable ('Num'). This successful classification means the variable can now be accurately used for sequential sorting, calculation of time series metrics, or incorporation as a continuous predictor in complex statistical models. This entire workflow, from diagnosis via PROC CONTENTS to transformation using the INPUT function, represents a robust methodology for maintaining data integrity in SAS.

Conclusion and Next Steps for Data Mastery

The conversion of character variables to numeric variables is more than just a technical step; it is a prerequisite for meaningful data analysis in SAS. By utilizing the **INPUT function** in combination with the appropriate informat, programmers can efficiently resolve common data type issues arising from data import and cleaning processes. This core skill ensures that quantitative information is treated correctly, allowing for reliable reporting and statistical inference.

To solidify expertise in data preparation, it is highly recommended to explore related conversion tasks. Data transformation flows frequently require reverse operations or more nuanced handling of complex string data. Continuous practice with different informats and format statements will enhance proficiency in preparing diverse datasets for analytical rigor.

Consider exploring the following advanced topics to further enhance your data manipulation toolkit:

Converting Numeric Variables Back to Character Variables using the **put() function**, which is the complementary process to the INPUT function.

Techniques for gracefully handling **Missing Values** (system missing or user-defined missing) that may arise during complex data type conversions, especially when character strings contain non-numeric characters that cannot be interpreted by the informat.

In-depth study of **Advanced Informats**, necessary for parsing complex string structures such as dates, times, or custom formatted codes.