How to Easily Convert Numeric Variables to Character Variables in SAS

Authored by
stats writer

December 1, 2025

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2025). *How to Easily Convert Numeric Variables to Character Variables in SAS*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=103242

In the field of data analysis and statistical programming, particularly within SAS, the ability to accurately transform data types is crucial. The PUT function serves as the primary mechanism for converting numeric variables into character strings. This function mandates the use of a specific character format, which defines the precise length and aesthetic representation of the resulting value. Proper application of the format ensures that the numeric variable is accurately and reliably converted into its required character form, facilitating tasks like reporting, concatenation, and merging data based on non-numeric keys.

The Mechanics of the PUT Function

The **PUT function** is the definitive tool within the SAS DATA step for performing the critical task of converting a numeric variable into a character variable. This conversion is essential when the raw numerical representation must be treated as textual data--for instance, when concatenating values or ensuring a specific display format in reports.

The fundamental power of the PUT function lies in its use of a specific SAS format to dictate how the numeric value is written out as a string. Without a specified format, the conversion is meaningless, as the system needs clear instructions on length, decimal precision, and alignment. The format acts as a template, guiding the conversion process with precision.

The structure is straightforward, requiring two primary arguments. First, the source **numeric variable**, which holds the raw data to be transformed. Second, a required format (e.g., **8.**, **ZDV3.**, **DATE9.**) that defines the length and appearance of the resulting character output. It is crucial to remember that the length of the newly created character variable will be determined by the width specified in the chosen format.

This function uses the following basic syntax:

character_var = put(numeric_var, 8.);

In this common usage, the target variable, **character_var**, is created based on the source **numeric_var**, utilizing the standard **8.** format. This instructs SAS to represent the numeric value as a character string up to eight characters long, right-justified, and padded with spaces if necessary. The following example shows how to use this function in practice.

Example: Setting Up the Source Data

To illustrate the practical application of the PUT function, we will establish a baseline SAS dataset

ARABPSYCHOLOGY.COM

named **original_data**. This dataset simulates a typical scenario in business analytics, recording the daily sales performance of a retail store over a ten-day period. It contains two inherently numeric variables: **day** (representing the sequence of days) and **sales** (representing the total units sold).

In many analytical contexts, variables like **day** are best maintained as numeric, especially if arithmetic operations or time-series indexing are required. However, for certain reporting tasks--such as creating categorical labels or concatenating the day number with descriptive text--it becomes necessary to treat these values as character strings. This example will focus on converting the **day** variable.

We use a standard SAS DATA step with inline datalines to rapidly create the necessary structure. This method ensures that the initial data type assignment is clearly numeric, setting the stage for the conversion process. Reviewing the initial output using **PROC PRINT** confirms the data's integrity and structure before any transformation occurs.

```
/*create dataset: Tracking daily sales for 10 days*/
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 the initial dataset structure*/
proc print data=original_data;
```

ARABPSYCHOLOGY.COM

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

As visualized above, the dataset is loaded correctly, showing clear numerical representations for both the day index and the sales figures. Our next logical step is to confirm the underlying data attributes to ensure we are working exclusively with numeric variables before transformation.

Validating Initial Data Types using PROC CONTENTS

Before initiating any critical data conversion, it is best practice to verify the structure and attributes of the source SAS dataset. The PROC CONTENTS procedure is the standard utility in SAS used specifically for this purpose. It generates a detailed description of the dataset metadata, including variable names, labels, formats, lengths, and, most importantly, their assigned data types.

Executing PROC CONTENTS confirms that both variables, **day** and **sales**, are currently classified as **numeric variables**. This verification step is vital because if the variable were already character-based, applying the PUT function would be redundant or could lead to unintended results if used incorrectly.

We execute the following code to retrieve the variable attributes:

/*Execute PROC CONTENTS to display metadata for each variable*/
proc contents data=original_data;

ARABPSYCHOLOGY.COM

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

The output clearly indicates that the Type for both **day** and **sales** is **Num** (Numeric). This confirms our starting point and validates the necessity of the upcoming conversion step.

Performing the Conversion using the PUT Function

We can use the following code to create a new dataset in which we convert the **day** variable from numeric to character:

To execute the conversion, we initiate a new SAS DATA step, creating a target dataset named **new_data**. Within this step, we use the **SET** statement to read all records from the **original_data** source. The core of the transformation happens when we define the new variable, **char_day**.

The statement `char_day = put(day, 8.);` is where the conversion occurs. The PUT function takes the numeric value stored in **day** and applies the **8.** format. The **8.** format signifies a standard numeric output with a width of 8 characters. Since the target variable, **char_day**, is assigned the result of a character function, SAS automatically defines **char_day** as a character variable with a length equal to the format width, which is 8 bytes in this case.

We also utilize the DROP statement within the DATA step. The statement `drop day;` is included here for clarity, ensuring that the final **new_data** dataset contains only the relevant variables: the converted character variable (**char_day**) and the unchanged numeric variable (**sales**).

/*create new dataset where 'day' is character*/
data new_data;
set original_data;
char_day = put(day, 8.);
drop day; /*Note: We used the DROP function to remove the original numeric day variable.*/
run;

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

Obs	sales	char_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

The resulting dataset, **new_data**, now displays the newly created **char_day** variable alongside **sales**, effectively replacing the original numeric day index.

Verifying the Converted Variable Type

The final and most crucial step in data transformation is verification. We must use PROC CONTENTS once more, this time targeting the newly created dataset, **new_data**, to confirm that **char_day** has been successfully assigned the character variable type.

Relying solely on the visual appearance of data in a PROC PRINT output can be misleading; only the metadata reported by PROC CONTENTS definitively confirms the internal data structure and type assigned by SAS.

Executing the following code confirms the type change and verifies the length of the new variable:

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

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

As confirmed by the output, the variable **char_day** now lists its Type as **Char** (Character) and its

ARABPSYCHOLOGY.COM

Length as 8, which corresponds precisely to the **8.** format used in the PUT function. We can see that the new variable we created, **char_day**, is a character variable.

Choosing the Appropriate SAS Numeric Format

The success and utility of the numeric-to-character conversion hinge entirely upon the specific SAS format provided as the second argument to the PUT function. The format dictates not only the resulting length of the character variable but also how padding, decimals, and special characters (like dollar signs or commas) are applied to the textual representation.

When selecting a format, consider the maximum possible width of the numeric input. If the format width (w) is too small, SAS will be unable to convert the number completely, resulting in asterisks (*****) being written to the character field, indicating an overflow error. Conversely, using an overly generous width results in unnecessary leading spaces, which may require subsequent trimming using functions like the **TRIM** or **LEFT** functions.

Several standard numeric formats are commonly employed for conversion purposes:

w.: This is the general numeric format, where 'w' specifies the width. It is ideal for integers, as seen in our example (**8.**). The resulting string will be right-justified, padded with spaces on the left.
w.d: This format is crucial when dealing with numbers that have decimal components. 'w' is the total width (including the decimal point and the sign), and 'd' specifies the number of digits to the right of the decimal point. For example, `PUT(sales, 6.2)` converts 12.5 to ' 12.50'.
Zw.: The Z format is used to left-pad the resulting character string with leading zeros instead of spaces. This is exceptionally useful when creating standard identifiers or codes that must maintain a fixed width, such as employee IDs or sequential record numbers.

DATA Step Behavior and Automatic Variable Creation

When working within the SAS DATA step, the context in which a variable is defined dictates its type and initial length. This is particularly important when using the PUT function to create a new variable, as SAS follows a specific set of rules for automatic variable definition.

Since the PUT function always returns a character value, the variable assigned the result (e.g., **char_day**) is automatically defined as a character variable. The length of this new variable is implicitly set by the width of the format specified in the function call. If the format used is **8.**, the resulting character variable will have a length of 8 bytes.

If you need the resulting character variable to have a different length than the format width, you must explicitly declare the length using the **LENGTH** statement **before** the assignment statement. For instance, if you use format **4.** but require the resulting character variable to be 10 characters long for future padding or concatenation, the **LENGTH** statement must precede the PUT function assignment.

```
/*Explicitly defining length before conversion*/
data padded_data;
set original_data;
length char_day $10;
char_day = put(day, 4.); /*The result (4 bytes) is padded to 10 bytes*/
run;
```

This careful management of variable definition ensures memory efficiency and prevents unexpected truncation or padding issues during complex data transformations within the SAS DATA step.

Common Pitfalls and Troubleshooting Numeric-to-Character Conversion

While the PUT function is robust, several common errors occur when converting numeric variables to character format in SAS. Awareness of these issues is crucial for successful data processing.

Format Truncation Error (Asterisks)

The most frequent error is specifying a format width that is too narrow for the numeric value. If the number of digits in the numeric value exceeds the width specified by the format (including space for a potential decimal point or sign), the output character string will contain only asterisks (*****). SAS does this to alert the user that the true value could not be represented in the space provided, preventing the loss of significant data. The solution is simply to increase the format width (e.g., using **5.** or **8.**).

Handling Missing Values

In SAS, a standard numeric missing value is represented by a single period (**.**), or by special missing values (A-Z or underscore). When the PUT function encounters a standard missing value, it converts it to a single space character in the resulting character variable. If you need the missing value to be represented by a specific character string (such as 'N/A' or 'MISSING'), you must utilize conditional logic (like an IF-THEN statement) before the PUT assignment to check for missingness.

Unwanted Leading Spaces

By default, most numeric formats right-justify the number and pad the left side with spaces. If you require a character string that is left-justified without leading spaces (often necessary for concatenation or database integration), you must wrap the PUT function with the **LEFT** or **TRIMN**

ARABPSYCHOLOGY.COM

functions. For instance: `char_day = left(put(day, 8.));`. This ensures a clean, space-free character representation of the original numeric data.

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

ARABPSYCHOLOGY.COM