How to Create Custom Formats in SAS with PROC FORMAT

Authored by

stats writer

November 21, 2025

The **PROC FORMAT** procedure is a fundamental tool within the **SAS** statistical software suite. It is specifically designed to create custom user-defined formats, which allow analysts to map raw data values to meaningful descriptive labels. This capability significantly enhances data readability and interpretation, making statistical output much clearer for end-users.

By using **PROC FORMAT**, you can assign labels and values to **variables**, transforming numerical codes or ranges into understandable text categories. Furthermore, it is essential for standardizing the display of specific data types, such as applying specific formats for currency, dates, or time values across different analyses, ensuring consistency throughout the data processing pipeline.

You can use **PROC FORMAT** in **SAS** to define a precise mapping of data values into corresponding descriptive labels. This mechanism is crucial for condensing continuous data or categorizing discrete observations effectively.

Understanding the Basic PROC FORMAT Syntax

The fundamental usage of the **PROC FORMAT** procedure requires defining a format name using the `VALUE` statement and then specifying the input ranges or values and their corresponding output labels. The procedure must be terminated by a `RUN` statement.

The following example illustrates the basic syntax used to define a custom format named `points_range`, which categorizes numerical values into qualitative labels:

```
proc format;
value points_range
25-high='High'
15-<25='Medium'
other ='Low';
run;
```

This structure effectively establishes clear cut-off points for the data, translating quantitative scores into meaningful categories. Specifically, this definition creates the following categorical mapping logic:

Values equal to 25 or greater (up to the highest value) will be displayed as the label '**High**'.
Values ranging from 15 up to, but not including, 25 will be shown as '**Medium**'.
All other numerical values that fall outside the defined ranges are captured by the `OTHER` keyword and will be shown as '**Low**'.

Setting Up the Sample Dataset

To demonstrate the practical application of **PROC FORMAT**, we will utilize a small sample **dataset** containing player statistics. This data simulates scores (`points`) associated with different teams and positions.

The following **SAS** code uses the `DATA` step and `DATALINES` statement to create the temporary **dataset** named `my_data`, followed by the `PROC PRINT` step to display its contents:

```
/*create dataset*/
data my_data;
input team $ position $ points;
datalines;
A Guard 25
A Guard 20
A Guard 30
A Forward 25
A Forward 10
B Guard 10
B Guard 22
B Forward 30
B Forward 10
B Forward 10
B Forward 25
;
run;

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

The resulting table, displayed below, shows the raw, unformatted point values for each observation:

Obs	team	position	points
1	A	Guard	25
2	A	Guard	20
3	A	Guard	30
4	A	Forward	25
5	A	Forward	10
6	B	Guard	10
7	B	Guard	22
8	B	Forward	30
9	B	Forward	10
10	B	Forward	10
11	B	Forward	25

Example 1: Applying Formats to Generate Descriptive Frequency Tables

One of the most common applications of **PROC FORMAT** is enhancing statistical reports, particularly when summarizing quantitative data. Before applying the format, let's generate a simple **frequency table** for the `points` **variable** using `PROC FREQ`:

```
/*calculate frequency of values in points column*/
proc freq data = my_data;
table points;
run;
```

As expected, the initial output displays the raw count and percentage of observations for each unique numerical value in the `points` column. While accurate, this raw output lacks descriptive context, especially if we are interested in performance bands rather than individual scores.

The FREQ Procedure

points	Frequency	Percent	Cumulative Frequency	Cumulative Percent
10	4	36.36	4	36.36
20	1	9.09	5	45.45
22	1	9.09	6	54.55
25	3	27.27	9	81.82
30	2	18.18	11	100.00

To make this output more insightful, we want to group these discrete points into performance categories. We will implement the previously defined categorization scheme:

Values equal to 25 or greater will be displayed as '**High**' performance.
Values between 15 (inclusive) and 25 (exclusive) will be shown as '**Medium**' performance.
All other remaining values will be categorized as '**Low**' performance.

We combine the format definition step with the `PROC FREQ` step, using the `FORMAT` statement within `PROC FREQ` to apply the user-defined format `points_range.` to the `points` **variable**:

```
/*define formatting for points variable*/
proc format;
value points_range
25-high='High'
15-<25='Medium'
other ='Low';
run;

/*create frequency table for points variable, using formatting defined above*/
proc freq data = my_data;
table points;
format points points_range.;
run;
```

The resulting **frequency table** successfully utilizes the defined format labels, grouping the raw values of the `points` variable into aggregated categories. This transformation significantly enhances the clarity of the statistical summary.

The FREQ Procedure

points	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Low	4	36.36	4	36.36
Medium	2	18.18	6	54.55
High	5	45.45	11	100.00

Example 2: Creating a New Categorical Variable using the PUT Function

While the previous example demonstrated how to apply a format temporarily for reporting purposes (like in `PROC FREQ`), sometimes it is necessary to permanently store the formatted labels as a new categorical **variable** within the **dataset**.

This permanent transformation is achieved using the `PUT` function within a **SAS** `DATA` step. The `PUT` function reads the numerical value of the source variable (`points`) and applies the specified format (`points_range.`) to assign the corresponding character label to the new variable (`point_range`).

The following syntax defines the format and then executes the `DATA` step to create the new **dataset**, `new_data`:

```
/*define formatting for points variable*/
proc format;
value points_range
25-high='High'
15-<25='Medium'
other ='Low';
run;

/*create new dataset with points_range variable*/
data new_data;
set my_data;
point_range = put(points, points_range.);
run;

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

The resulting output clearly shows the newly generated variable, `point_range`, appended to the

original data. This new **variable** contains the textual labels ('Low', 'Medium', or 'High') corresponding directly to the numerical performance score in the `points` column.

Obs	team	position	points	point_range
1	A	Guard	25	High
2	A	Guard	20	Medium
3	A	Guard	30	High
4	A	Forward	25	High
5	A	Forward	10	Low
6	B	Guard	10	Low
7	B	Guard	22	Medium
8	B	Forward	30	High
9	B	Forward	10	Low
10	B	Forward	10	Low
11	B	Forward	25	High

Key Takeaways on Using PROC FORMAT

The examples provided illustrate the power and flexibility of **PROC FORMAT** in **SAS** programming. Whether you need to temporarily enhance the readability of reporting procedures like `PROC FREQ` or permanently categorize data using the `PUT` function to create a new **dataset** variable, user-defined formats are essential for effective data management and communication.

The ability to map complex numerical ranges to simple, descriptive text labels is vital for producing clear, actionable statistical output, especially when handling large volumes of raw data or generating reports intended for non-technical audiences.

Further Learning and Resources

For advanced options, such as defining picture formats for dates and times, or handling specific character formats, it is always recommended to consult the official documentation. You can find the complete and comprehensive documentation for **PROC FORMAT** at the official SAS website.

To deepen your expertise in data manipulation and statistical analysis within the SAS environment, consider exploring tutorials on related procedures such as `PROC FREQ`, `PROC PRINT`, or the `DATA` step syntax.

The following list provides references to tutorials that explain how to perform other common tasks

in SAS:

Tutorial 1: Using SAS to merge datasets.
Tutorial 2: Advanced filtering techniques with WHERE statements.
Tutorial 3: Understanding macro variables in SAS.