How to Easily Create Line Plots in SAS Using PROC SGPLOT

Authored by

stats writer

December 1, 2025

The creation of effective line plots is fundamental for analyzing trends and changes in data, particularly over time or a sequential series. In the realm of the SAS statistical software, generating these visualizations is efficiently handled by the powerful PROC SGPLOT procedure. This modern approach to statistical graphics in SAS allows users to create highly customized and publication-ready plots with streamlined syntax.

To produce a line plot, the core requirement is utilizing the `SERIES` statement within `PROC SGPLOT`. This statement necessitates specifying the variables for the x-axis and the y-axis, which define the relationship being visualized. Furthermore, advanced features such as customizing line attributes--including style, color, thickness, and adding markers--are readily available to enhance the plot's clarity and visual appeal. This article serves as an expert guide, illustrating the practical application of `PROC SGPLOT` through detailed examples, starting from basic single-line plots and progressing to complex multi-line comparisons.

Understanding the PROC SGPLOT Procedure

The `PROC SGPLOT` procedure stands as the cornerstone for modern statistical graphics generation within the SAS System. It is an essential tool for high-quality data visualization, offering declarative syntax that is often easier to read and maintain compared to older SAS/GRAPH methods. Understanding the structure of this procedure is the first step toward mastering line plot creation.

`PROC SGPLOT` operates by taking a dataset as input and then utilizing one or more plot statements (like `SERIES`, `SCATTER`, `HBAR`, etc.) to define the visual representation. For line plots, the `SERIES` statement is specifically employed, telling SAS to connect the data points defined by the x and y variables sequentially. This capability makes it ideal for visualizing continuous data or ordered categories.

Effective use of `PROC SGPLOT` requires attention to various options outside the primary plot statement. Global options, such as `TITLE` for adding descriptive headers or `STYLEATTRS` for controlling global color palettes, allow for comprehensive chart customization. By integrating these elements, analysts can ensure their generated plots are not only accurate representations of the data but also aesthetically pleasing and easily interpretable for stakeholders.

Essential Syntax for Creating Line Plots

Creating a basic line plot in SAS using `PROC SGPLOT` is straightforward and relies on minimal syntax. The procedure begins with calling `PROC SGPLOT` and specifying the input data. The crucial component is the `SERIES` statement, where the relationship between the two variables is defined.

The basic syntax structure requires only the definition of the X and Y variables. The X variable typically represents the sequence or time component, while the Y variable represents the

measurement or value corresponding to that sequence. The following template demonstrates the fundamental syntax used to initiate the generation of a line plot:

You can use **proc sgplot** to create line plots in SAS.

This procedure uses the following basic syntax:

```
/*create dataset*/
proc sgplot data=my_data;
series x=x_variable y=y_variable;
run;
```

This simple structure forms the foundation for all subsequent line plots. By utilizing the `DATA=` option, we instruct SAS which dataset to reference. The `SERIES X=Y=` statement then executes the core visualization task. The following examples demonstrate how to use this procedure to create increasingly complex line plots in SAS, showing both univariate trends and multivariate comparisons.

Example 1: Visualizing Univariate Time Series Data

Our first example focuses on visualizing a single measure over a sequence, often referred to as time series data, even if the sequence variable is not strictly time but an ordered category (like Day 1, Day 2, etc.). We will analyze the total sales recorded by a single retail store across ten consecutive days. This scenario is perfect for illustrating the most basic application of the `PROC SGPLOT` procedure for line generation.

To begin, we must first establish the dataset. The following SAS code block demonstrates the creation of the `my_data` dataset using the `DATALINES` statement, followed by the verification of the data using `PROC PRINT`. Notice that the 'day' variable, while sequential, is defined as a character type (denoted by the '$') in the `INPUT` statement, although it will be interpreted sequentially on the axis.

Suppose we have the following dataset in SAS that shows the total sales made by a store during 10 consecutive days:

```
/*create dataset*/
data my_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;

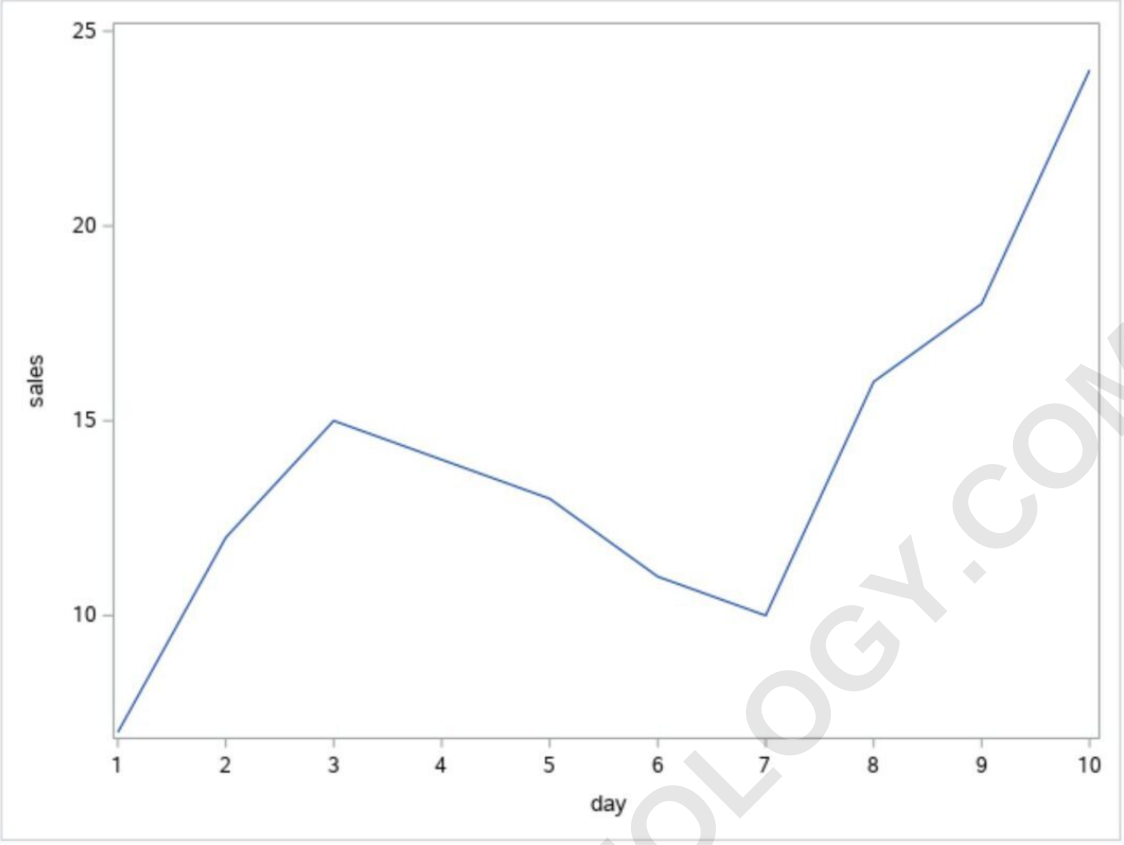
/*view dataset*/
proc print data=my_data;

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

Once the data is prepared and verified, we proceed to the plotting phase. The `PROC SGPLOT` statement is called, referencing the `my_data` dataset. We then use the `SERIES` statement, specifying `x=day` and `y=sales`. This instructs SAS to plot the sales figures chronologically by day, automatically generating a visual representation of the sales trend over the ten-day period.

We can use **proc sgplot** to create a line plot that displays the day on the x-axis and sales on the y-axis:

/*create line plot that displays sales by day*/
proc sgplot data=my_data;
series x=day y=sales;
run;



Customizing Single Line Plot Appearance

While the basic line plot provides a clear representation of the trend, effective data visualization often requires customizing the appearance to match institutional branding, emphasize specific data features, or simply improve overall chart readability. `PROC SGPLOT` offers extensive options for visual customization through global statements and options within the `SERIES` statement itself.

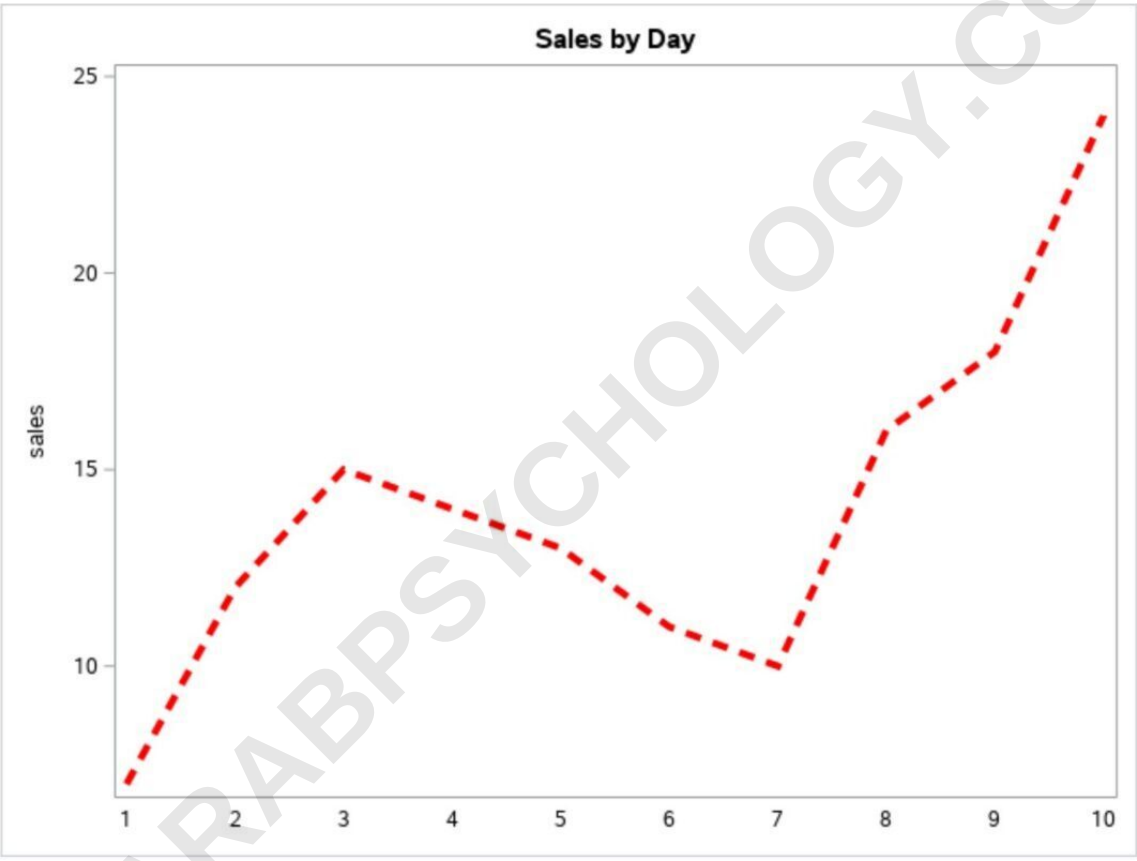
Customizations can include setting a detailed title, adjusting axis properties, and modifying the line aesthetics. The `LINEATTRS` option within the `SERIES` statement is particularly powerful, allowing control over the line's **color**, **pattern** (e.g., solid, dashed, dotted), and **thickness**. Furthermore, the `XAXIS` and `YAXIS` statements allow for fine-tuning of the coordinate system, such as suppressing labels, lines, or tick marks to achieve a cleaner look.

The example below demonstrates how to apply several critical customizations. We use the `TITLE` statement to assign a clear, descriptive heading. Within the `SERIES` statement, we define the line to be red, dashed (`pattern=dash`), and thicker (`thickness=4`). Finally, the `XAXIS` statement is used to remove redundant elements like labels and tick marks, focusing the viewer's attention purely on the plotted data trajectory.

We can use the following code to modify the appearance of the chart, including the title, labels,

colors, line pattern, and line thickness:

```
/*create custom line plot*/
title "Sales by Day";
proc sgplot data=my_data;
series x=day y=sales / lineattrs=(color=red pattern=dash thickness=4);
xaxis display=(nolabel noline noticks);
run;
title;
```



Example 2: Comparing Multiple Groups

Often, data analysis requires comparing trends across multiple categories simultaneously. In SAS, this is achieved by creating a multi-line plot, where each category is represented by its own distinct line. This is particularly useful for comparative analysis of different groups, such as comparing the performance of multiple products, regions, or in this case, different stores.

For this example, we expand our dataset to include sales data from three separate stores (A, B, and C) over five consecutive days. This introduces a categorical variable, `store`, which is essential

for differentiating the lines in the plot. The structure of the data requires that we input not only the day and sales but also the identifier for the store associated with those sales figures.

The following SAS code block constructs this comparative dataset. Note the inclusion of the `store` variable in the `INPUT` statement, ensuring that each sales observation is correctly associated with its respective store. We then verify the structure of this expanded dataset using `PROC PRINT` before proceeding to the visualization stage.

Suppose we have the following dataset in SAS that shows the total sales made by three different stores during five consecutive days:

```
/*create dataset*/
data my_data;
input store $ day $ sales;
datalines;
A 1 13
A 2 18
A 3 20
A 4 25
A 5 26
B 1 3
B 2 7
B 3 12
B 4 12
B 5 11
C 1 6
C 2 12
C 3 19
C 4 20
C 5 21
;
run;

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

Sales by Day

Obs	store	day	sales
1	A	1	13
2	A	2	18
3	A	3	20
4	A	4	25
5	A	5	26
6	B	1	3
7	B	2	7
8	B	3	12
9	B	4	12
10	B	5	11
11	C	1	6
12	C	2	12
13	C	3	19
14	C	4	20
15	C	5	21

Enhancing Multi-Line Plots with Group Options

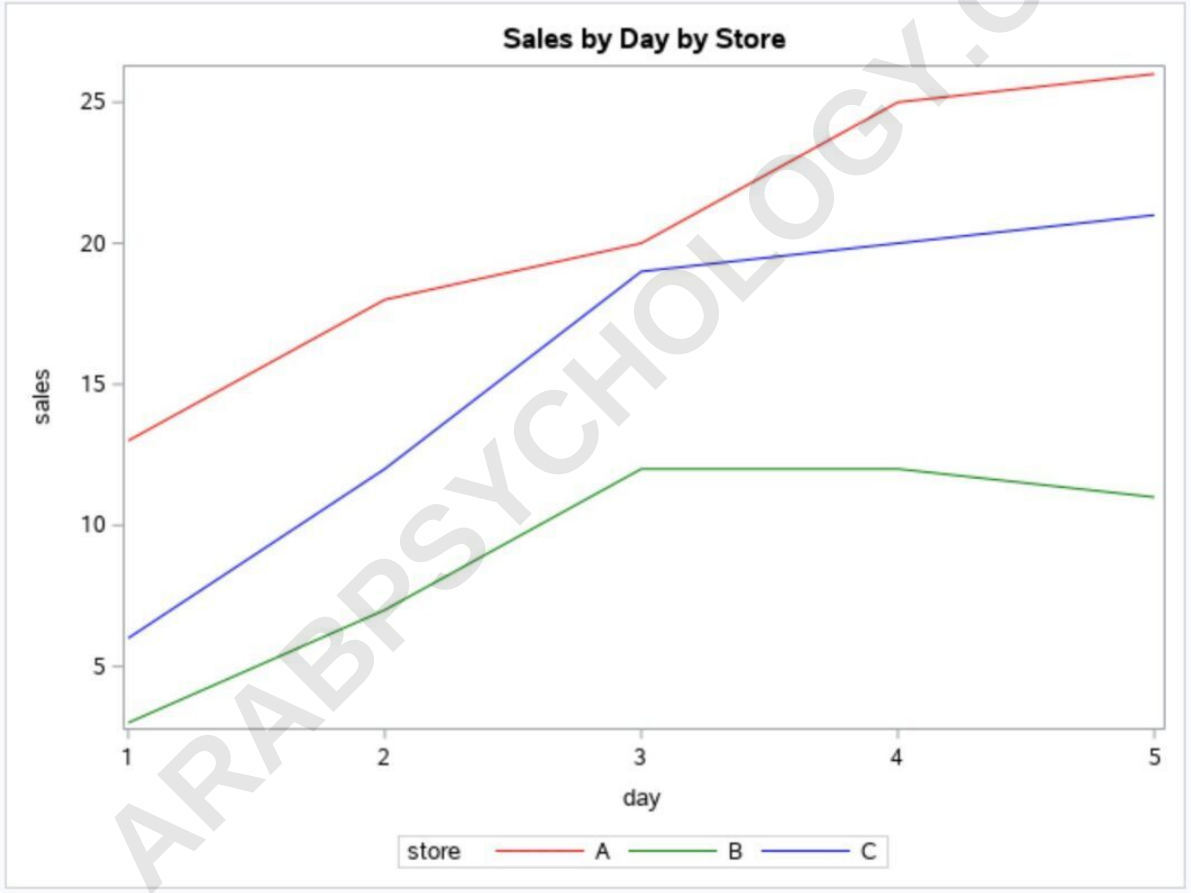
To differentiate the multiple lines derived from the categorical variable (store), the `PROC SGPLOT` procedure utilizes the crucial `GROUP=` argument within the `SERIES` statement. The `GROUP=store` option instructs SAS to partition the data based on the unique values of the `store` variable and generate a separate line for each group. SAS automatically assigns different default colors to these lines to ensure visual distinction, a necessary feature for effective comparative line plots.

While default colors are functional, highly customized visualizations often require specific color choices. This is where the `STYLEATTRS` statement becomes invaluable. By using `STYLEATTRS DATACONTRASTCOLORS=()`, we can override the default color scheme and assign a specific list of colors (e.g., red, green, blue) that SAS will cycle through for each group defined by the `GROUP` option. This provides precise control over the visual output, ensuring the colors are distinct and align with any presentation requirements.

The code below demonstrates the implementation of both the `GROUP` option and the `STYLEATTRS` statement to create a clear, multi-line comparison of sales trends across the three stores. The plot clearly shows the daily sales performance of each store, allowing for immediate visual comparison of their trajectories over the five-day period.

We can use **proc sgplot** with the **group** argument to create a line plot that displays the sales made by each of the three stores:

```
/*create line plot that displays sales by day for each store*/
title "Sales by Day by Store";
proc sgplot data=my_data;
styleattrs datacontrastcolors=(red green blue);
series x=day y=sales / group=store;
run;
title;
```



In the resultant visualization, the x-axis properly displays the sequence (day), and the y-axis represents the quantitative measure (sales). Importantly, the inclusion of the legend, automatically generated by `PROC SGPLOT` when the `GROUP` option is used, clearly maps each line's color to its corresponding store identifier. The three individual lines effectively show the sales made by each of the three stores during each day, facilitating a clear and comprehensive comparative analysis.

Best Practices for Effective Line Plot Visualization

Mastering `PROC SGPLOT` extends beyond simple syntax; it involves adopting best practices to ensure that the generated line plots are clear, informative, and free from misleading visual elements. A poorly configured plot can obscure critical trends or misrepresent data relationships, undermining the integrity of the analysis.

One primary best practice is ensuring appropriate axis scaling. While SAS defaults often work well, analysts must verify that the y-axis scale starts at zero, especially when absolute quantities are being compared, to avoid exaggerating small fluctuations. If the data represents percentage changes or relative measures, a non-zero baseline might be acceptable, but this decision must be made consciously and clearly noted.

Furthermore, effective labeling is paramount. Every plot must have a clear title (using the `TITLE` statement), and both axes should be explicitly labeled (although `PROC SGPLOT` often attempts to derive labels from variable names, manual overrides are often superior). When utilizing multi-line plots, ensure the legend is prominently displayed and easy to interpret, using high-contrast colors defined via `STYLEATTRS` to maximize differentiation between lines.

Finally, consider the use of markers. While the line itself shows the continuous trend, adding markers (e.g., small circles or squares) at each data point can help viewers identify the exact coordinates used in the plot, especially useful when dealing with discrete data points that are connected to imply continuity. The `MARKERATTRS` option can be used alongside the `SERIES` statement to incorporate this valuable visual element, further improving the plot's overall readability and precision.