Learning ggplot2: Connecting Points with Lines Using geom_line()

Authored by

Mohammed looti

October 26, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Learning ggplot2: Connecting Points with Lines Using geom_line()*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=3844

Understanding Line Plots in Data Visualization

Line plots, often referred to as line charts, are one of the most fundamental and powerful tools in data visualization, particularly when illustrating trends over time or sequential data. They are instrumental in revealing patterns, continuity, and the rate of change between data points. When working within the R environment, the ggplot2 package stands out as the premier tool for generating high-quality statistical graphics, adhering closely to the principles of the Grammar of Graphics.

The process of connecting individual data points with lines transforms a collection of discrete observations into a compelling time series visualization, allowing the viewer to easily track the progression from one measurement to the next. This combination--using both points to denote specific observations and lines to show the relationship between them--provides maximum clarity and avoids misinterpretation of the underlying data structure. The ggplot2 library simplifies this complex visual task into a straightforward application of additive layers, making sophisticated plotting accessible to analysts and statisticians alike.

The Essential Syntax for Connecting Points in ggplot2

To successfully connect points with lines using the ggplot2 package, we rely on combining two primary geometric objects, or "geoms": `geom_line()` and `geom_point()`. The basic structure involves initializing the plot, defining the aesthetic mappings (which variables map to the X and Y axes), and then sequentially layering these two geoms onto the base plot. This sequential layering ensures that the points are plotted directly on top of the connecting lines, creating a unified visual representation where both the continuity and the exact measured values are visible.

The following syntax represents the core foundation necessary for achieving this common visualization goal. Note the inclusion of the initial library call, which is mandatory before accessing any ggplot2 functions within the R session.

library(ggplot2)

ggplot(df, aes(x=x_var, y=y_var)) +
geom_line() +
geom_point()

In this structure, `ggplot(df, aes(x=x_var, y=y_var))` initializes the plot, specifying the data frame (`df`) and establishing the primary aesthetic mappings (`aes`). Subsequently, `geom_line()` draws the lines connecting the specified coordinates based on their order in the data, and `geom_point()` overlays the distinct markers onto those lines. Understanding this fundamental

layered syntax is the first step toward generating robust and meaningful plots in R.

A Practical Demonstration: Visualizing Time Series Data

To illustrate the practical application of this syntax, we will utilize a sample data frame simulating daily sales figures over a period of ten consecutive days. This type of sequential data is ideally suited for visualization using connected points, as it clearly demonstrates the evolution of sales performance over time. Establishing a clear, tidy data structure is always the precursor to effective visualization in R, ensuring variables are correctly mapped to their aesthetic roles.

Suppose we define the following data frame containing two variables: `day` (the independent, sequential variable, suitable for the x-axis) and `sales` (the dependent variable, suitable for the y-axis). The data reflects minor fluctuations, which a line plot is perfectly designed to highlight, showing the trajectory of sales over the ten-day period:

#create data frame
df <- data.frame(day=1:10,
sales=c(3, 5, 5, 8, 12, 10, 8, 8, 5, 9))

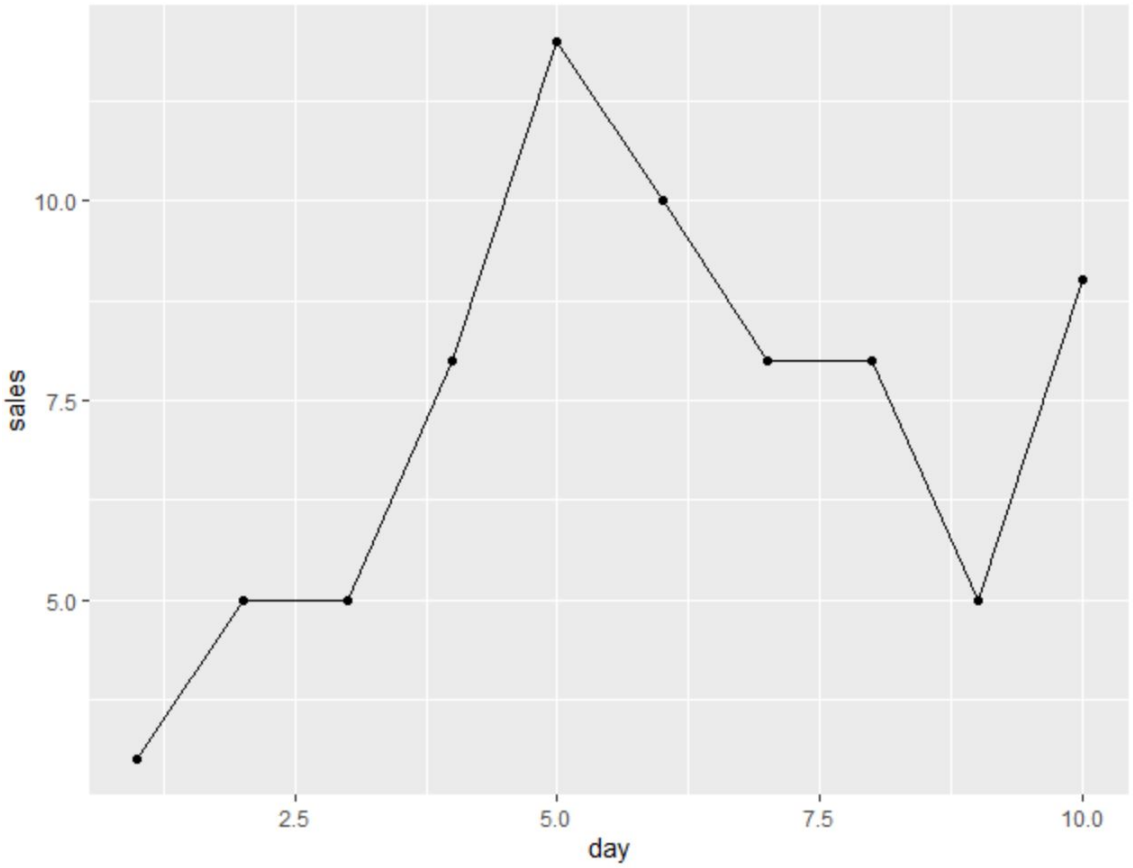
#view data frame
df

day sales
1 1 3
2 2 5
3 3 5
4 4 8
5 5 12
6 6 10
7 7 8
8 8 8
9 9 5
10 10 9

With our data prepared, we can now apply the previously introduced layered syntax to generate the visualization. We map the `day` variable to the x-axis and the `sales` variable to the y-axis, instructing ggplot2 to draw both the connecting lines (`geom_line()`) and the individual data markers (`geom_point()`). This results in a clear time series visualization that immediately conveys the daily sales trend.

library(ggplot2)

```
#create plot with connected points
ggplot(df, aes(x=day, y=sales)) +
geom_line() +
geom_point()
```



As evident in the resulting graphic, the x-axis successfully displays the sequential progression of the days (1 through 10), while the y-axis accurately represents the corresponding sales figures. This default visualization is clean and highly informative, clearly illustrating the sales peak on day 5 and subsequent fluctuations, thereby fulfilling the primary objective of visualizing sequential data.

Decoding the ggplot2 Layers: geom_line() and geom_point()

The true power of ggplot2 lies in its adherence to the Grammar of Graphics, which promotes plot construction through independent, customizable layers. When constructing a plot with connected points, it is critical to understand the distinct and non-overlapping roles played by `geom_line()` and `geom_point()`. Although they are used together to create a single coherent image, they are functionally independent layers, each responsible for drawing a specific component of the graph.

The function `geom_line()` is specifically designed to render path geometries. It connects

observations sequentially based on the order defined by the mapped x-axis variable. It is essential for showing movement or continuity across a dimension, emphasizing the trend rather than the individual observation. If this geom were used alone, the resulting graphic would show only the path, without any explicit markers for the individual data observations, which can sometimes lead to ambiguity regarding where the original measurements were taken, especially if the underlying data is sparse or highly discrete.

Conversely, `geom_point()` is responsible for creating scatter plots. It plots each individual observation as a distinct geometric shape based on its mapped x and y coordinates. When used in conjunction with `geom_line()`, it ensures that the viewer can precisely identify the location of the measured value for each day or sequential step. The combination of these two geoms is the standard best practice for visualizing time series or sequential data where both the overall trend and the precise observation location are important for analysis and reporting.

Advanced Customization: Modifying Aesthetics for Visual Impact

While the default plot provides a clear representation, one of the greatest benefits of using the ggplot2 framework is the ability to exert fine-grained control over every visual aesthetic. Both the line and the points can be independently customized using specific arguments passed directly within their respective geom functions. These customizations allow users to adjust the visual hierarchy, drawing attention to specific features of the data. Key aesthetic properties that can be modified statically include **color**, **size**, **linetype**, **shape**, and **fill** characteristics.

The customization arguments are applied as follows:

The **color** argument: This controls the outline or stroke color of the points and the hue of the line. It is a fundamental tool for differentiating series or simply enhancing visibility against a white background.

The **size** argument: Determines the thickness of the line (measured in millimeters) or the diameter/scale of the points. Increasing the size can help emphasize critical features.

The **linetype** argument: Only applicable to lines, allowing selection of patterns such as `solid`, `dashed`, `dotted`, or other coded patterns, which is useful when overlaying multiple series.

The **shape** argument: Only applicable to points, defining the geometric form of the marker. Shapes are numerically coded (e.g., 19 for solid circles, 22 for a filled square).

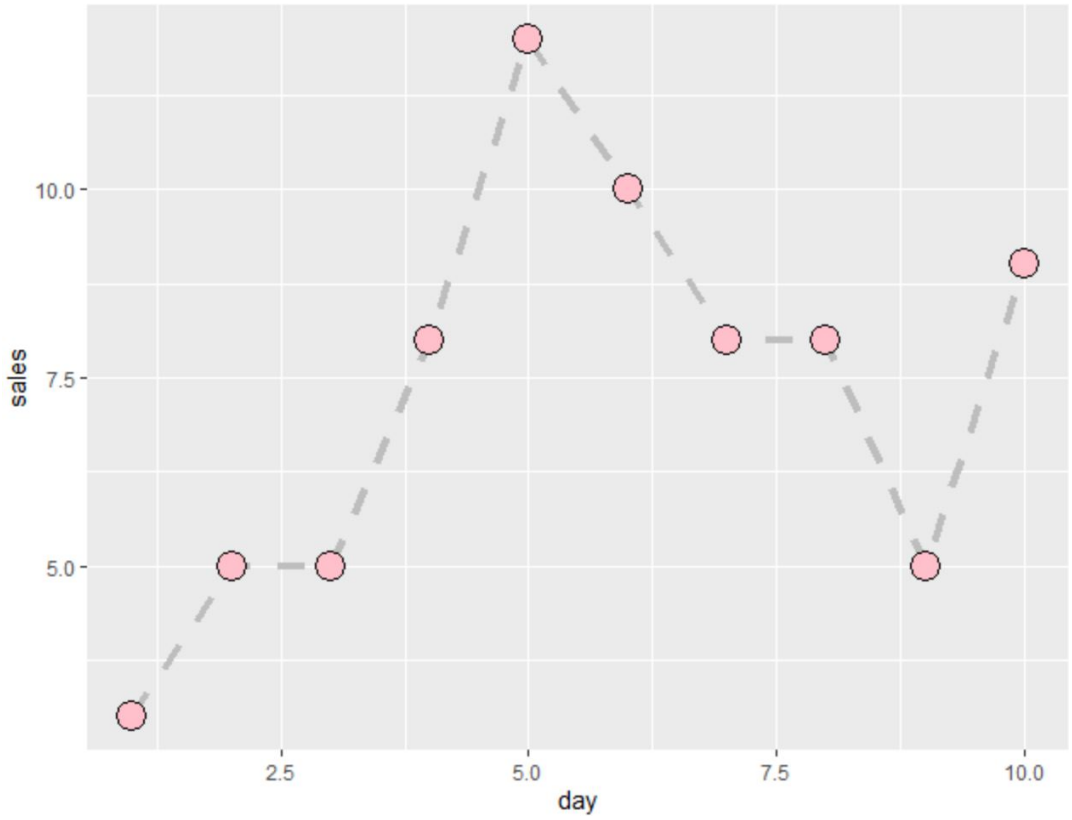
The **fill** argument: Controls the interior color of certain point shapes (specifically shapes 21 through 25), allowing for points that have both an outline (controlled by **color**) and an internal color (controlled by **fill**).

By defining these parameters directly within the geom functions, we apply static aesthetic modifications (i.e., modifications that do not depend on a variable in the data). For instance, we can make the line thicker and dashed, and simultaneously make the points larger, changing their shape and filling them with a contrasting color for maximum visibility, as demonstrated in the extended code example below:

library(ggplot2)

#create plot with connected points
ggplot(df, aes(x=day, y=sales)) +
geom_line(color='grey', size=1.5, linetype='dashed') +
geom_point(shape=21, color='black', fill='pink', size=6)

This code produces a visually distinct plot where the trend line is subdued (grey and dashed), drawing primary attention to the large, pink-filled data points defined by **shape=21**. The careful selection of these aesthetic parameters is crucial for emphasizing key aspects of the data story and ensuring the graphic meets specific presentation standards, such as those required for academic publishing or corporate reporting.



We highly recommend thorough experimentation with the values for any of these arguments.

Adjusting parameters such as **shape** or the **size** of the elements can dramatically alter the visual hierarchy and overall effectiveness of the plot. Customization ensures the final output is not only accurately representing the data but is also visually compelling and tailored precisely to the communication needs of the project.

Conclusion and Further Resources

Creating professional visualizations that connect points with lines in R is efficiently accomplished using the powerful layered approach of ggplot2. By mastering the synergy between `geom_line()` and `geom_point()` and applying detailed aesthetic customizations, analysts can produce clear, compelling time series graphics that effectively communicate temporal trends and observed values. This technique forms the backbone of many analytical reports where tracking performance or change over a sequential dimension is paramount to making informed decisions.

For those looking to deepen their expertise in advanced plotting techniques within the R ecosystem, the following resources provide guidance on other common visualization tasks and functions available within the ggplot2 framework:

The following tutorials explain how to perform other common tasks in ggplot2: