Creating Overlay Plots in R: A Step-by-Step Guide

Authored by

Mohammed looti

November 3, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Creating Overlay Plots in R: A Step-by-Step Guide*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=9268

Effective data analysis frequently necessitates comparing multiple datasets or visualizing distinct trends within a unified graphical space. In the R programming environment, this powerful technique is termed **overlay plotting**. While sophisticated packages like ggplot2 offer declarative syntax for complex visualizations, mastering R's fundamental base graphics system provides essential control and flexibility for layering data quickly and precisely.

The foundational approach to generating **overlay plots** relies on a sequential drawing process. The workflow always begins by establishing the coordinate system using the primary `plot()` function. Once the graphical canvas is initialized, subsequent data layers are introduced using specialized auxiliary functions. The two core functions utilized for appending new data series are `lines()` and `points()`, which facilitate the addition of continuous trend lines and discrete markers, respectively, onto the existing plot area.

Understanding the distinction between these functions is vital for constructing complex visualizations. The initial call to `plot()` initializes the graphical device, determines the boundaries of the axes, and draws the first data series. Conversely, `lines()` and `points()` are specifically designed to add graphical elements without reinitializing the plot window or altering the existing axes limits. This sequential layering mechanism is what allows for the creation of rich, comparative visualizations.

Establishing the Foundation: The Base R Overlay Mechanism

To successfully combine plots, the crucial first step involves defining the entire coordinate space using the initial `plot()` function call. This step sets the stage, defining the X and Y boundaries that all subsequent data series must adhere to. Once the canvas is established, you can sequentially introduce additional datasets using the auxiliary functions `lines()` or `points()`. The general syntax below illustrates how these core functions interact to layer various plot types:

#create scatterplot of x1 vs. y1 (Initial plot sets the axes)
plot(x1, y1)

#overlay line plot of x2 vs. y2
lines() function(x2, y2)

#overlay scatterplot of x3 vs. y3
points() function(x2, y2)

The principle of sequential drawing is fundamental to R's base graphics. Every graphical command, unless specifically instructed otherwise, draws directly on top of the previous output. This characteristic simplifies the incremental construction of complex figures. However, this flexibility requires careful management of plotting parameters, particularly axis limits, to ensure all

overlaid data series are visible and correctly scaled within the initial framework established by `plot()`.

If the range of the first dataset (x1, y1) is significantly smaller than the subsequent datasets (x2, y2, etc.), the overlayed data might fall outside the visible area of the plot window. Therefore, controlling the axes limits (using the `xlim` and `ylim` arguments) during the initial `plot()` call is frequently necessary to ensure all data points are displayed correctly.

Visual Differentiation: Essential Parameters for Plot Clarity

When implementing overlay plots, effective visualization relies heavily on the ability to visually differentiate between the multiple datasets sharing the same space. Without distinct visual cues, the plot quickly becomes ambiguous and uninterpretable. Therefore, it is imperative to use unique visual attributes for each series, specifically controlling color, line type (`lty`), line width (`lwd`), and point shape (`pch`).

The two most critical parameters for distinguishing overlaid data series are color (`col`) and the point character (`pch`). The `col` argument accepts standard color names (e.g., 'red', 'blue') or precise hexadecimal color codes. Utilizing distinct colors is the quickest and most effective way to separate lines or groups of points. For professional graphics, selecting a palette that offers high contrast and is colorblind-friendly is highly recommended.

For scatterplots, the `pch` argument is essential for defining the shape of the markers. This parameter takes an integer value corresponding to a specific plotting character. Common values include 1 (open circle), 2 (open triangle), 3 (plus sign), 15 (filled square), and 19 (solid, filled circle). By varying both the color and the point shape, analysts can clearly mark and categorize observations belonging to different groups, even when they occupy overlapping regions of the plot.

It is crucial that these customization parameters are consistently applied across the `plot()`, `lines()`, and `points()` calls, and accurately documented within the accompanying legend() call, thereby maintaining graphical integrity and ensuring straightforward interpretation by the audience.

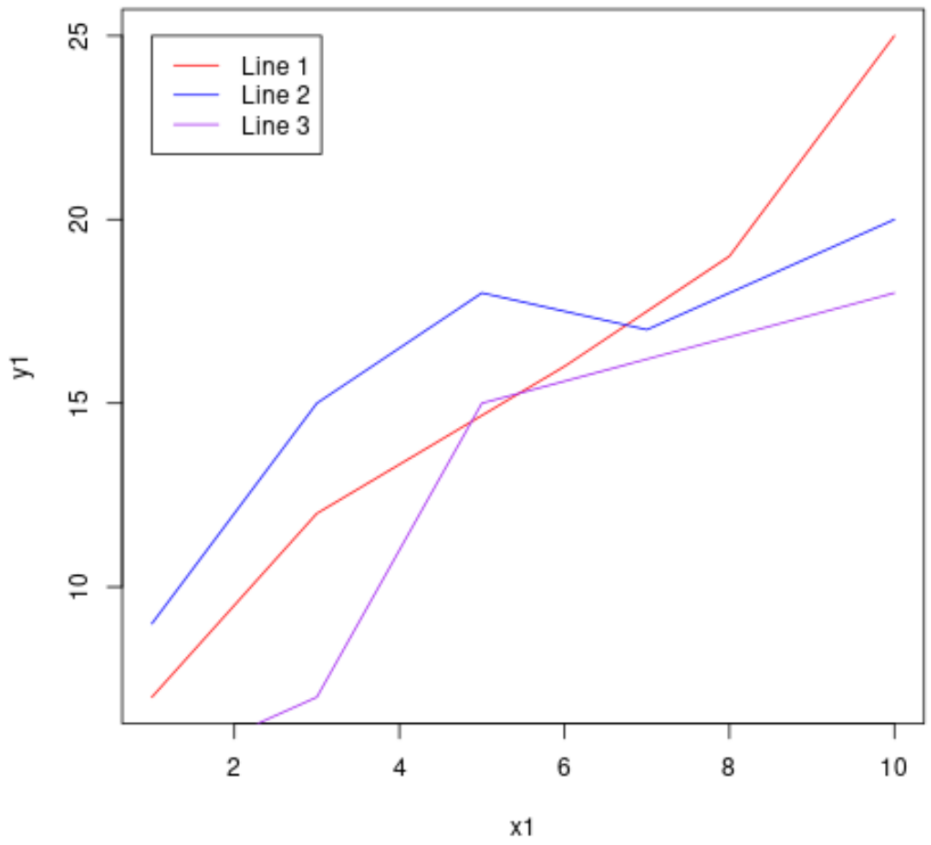
Example 1: Overlaying Multiple Line Plots for Trend Comparison

Line plots are the standard choice for visualizing trends across a continuous independent variable, such as time. When the objective is to compare the trajectory of two or more distinct processes, overlaying them onto a single graph significantly simplifies comparative analysis and highlights relative performance differences. This first practical example demonstrates how to overlay three separate data series using the dedicated `lines()` function, ensuring each trend is visually distinguishable.

In the following R code block, we first define three pairs of corresponding X and Y variables (x1/y1, x2/y2, x3/y3). The initial `plot()` command is critical; it uses the argument `type='l'` to explicitly specify that the primary visualization should be a line plot and establishes the maximum extent of the axes. Subsequent datasets are then added sequentially using the `lines()` function, with the `col` argument controlling the color of each line to ensure separation.

```
#define datasets
x1 = c(1, 3, 6, 8, 10)
y1 = c(7, 12, 16, 19, 25)

x2 = c(1, 3, 5, 7, 10)
y2 = c(9, 15, 18, 17, 20)

x3 = c(1, 2, 3, 5, 10)
y3 = c(5, 6, 7, 15, 18)

#create line plot of x1 vs. y1 (Sets the initial canvas)
plot(x1, y1, type='l', col='red')

#overlay line plot of x2 vs. y2
lines(x2, y2, col='blue')

#overlay line plot of x3 vs. y3
lines(x3, y3, col='purple')

#add legend
legend(1, 25, legend=c('Line 1', 'Line 2', 'Line 3'),
col=c('red', 'blue', 'purple'), lty=1)
```

The final, essential step in constructing a clear, comparative visualization is the inclusion of a comprehensive legend. The `legend()` function associates the specific colors and line styles with their corresponding datasets, ensuring that the reader can correctly interpret the visual differences. The coordinates (1, 25) define the anchor position of the legend box within the plot area, which must be chosen strategically to avoid obscuring any critical data points or overlapping lines.



Example 2: Overlaying Scatterplots for Distribution Comparison

Overlaying scatterplots is an exceptionally effective method for comparing the distribution, density, or correlation patterns of multiple distinct groups simultaneously. Since scatterplots focus on discrete data points rather than continuous trends, the meticulous selection of marker shape (`pch`) and color (`col`) becomes paramount for maintaining visual clarity. This example demonstrates how to combine two scatterplots using the dedicated `points()` function.

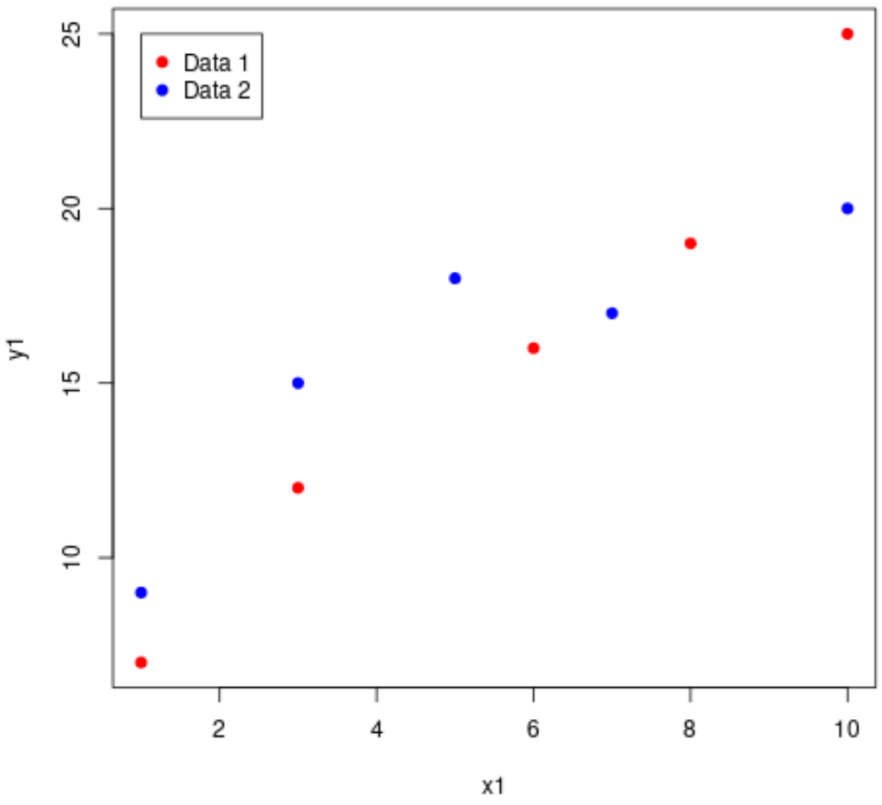
We begin by defining two separate datasets (x1/y1 and x2/y2). The initial `plot()` call creates the first set of points and establishes the plot boundaries. To ensure that the initial data is displayed with specific characteristics, we explicitly set the color (`col='red'`) and utilize `pch=19`, which specifies a solid, filled circle marker, providing a clean baseline for the comparison.

#define datasets
x1 = c(1, 3, 6, 8, 10)
y1 = c(7, 12, 16, 19, 25)

x2 = c(1, 3, 5, 7, 10)
y2 = c(9, 15, 18, 17, 20)

```
#create scatterplot of x1 vs. y1
plot(x1, y1, col='red', pch=19)

#overlay scatterplot of x2 vs. y2
points(x2, y2, col='blue', pch=19)

#add legend
legend(1, 25, legend=c('Data 1', 'Data 2'), pch=c(19, 19), col=c('red', 'blue'))
```

The second dataset is added using the `points()` function, which draws markers without connecting lines. In this case, both datasets utilize the same point shape (`pch=19`), but they are differentiated solely by color (red and blue). This methodology is highly effective when the primary goal is to illustrate how two different populations occupy the same shared space on the Cartesian plane, allowing for immediate visual assessment of overlap and displacement.



Advanced Axis Management and Scaling Issues

While the basic examples effectively demonstrated plotting when all data series fall within a similar range, more sophisticated visualization projects frequently require explicit axis management. A critical challenge arises when a second dataset possesses a significantly larger range than the

initial dataset used in the `plot()` call. In such instances, the second series will be severely clipped, or entirely invisible, unless the initial function preemptively defines sufficiently large axes limits.

To calculate the optimal plot range, the analyst must first determine the absolute minimum and maximum values across all X variables and all Y variables intended for plotting. The built-in `min()` and `max()` functions in R are indispensable tools for this aggregation. These calculated global limits should then be passed to the `xlim` (X-axis limit) and `ylim` (Y-axis limit) arguments within the primary `plot()` function. This guarantees that all subsequent overlaid data series, regardless of their individual extent, will be fully contained and visible within the graphical window.

A second, more complex issue in advanced overlays is the need for dual Y-axes--for example, when plotting temperature (small range) and rainfall (large range) against the same time variable. Although the base R graphics system does not natively support an automated second Y-axis on the right side of the plot, this effect can be manually simulated using the `par(new=TRUE)` command. By setting this parameter to `TRUE`, R is instructed to start drawing a new plot on the existing graphical device without clearing the previous content or resetting the coordinate system. This technique is often challenging, as it requires manual alignment of tick marks, labels, and titles, but it provides the necessary control when comparing series with vastly disparate scales.

Combining Plot Types and Further Resources

One of the great strengths of the base graphics system is the ability to freely mix and match plot types using the overlay functions. For instance, if the goal is to visualize a fitted statistical model against the raw observational data that generated it, you would use `points()` for the scatter data and `lines()` for the continuous model prediction. By applying unique combinations of `col`, `lty` (for the line), and `pch` (for the points), you establish a clear visual hierarchy that distinguishes the theoretical fit from the empirical observations.

Mastering base R plotting functions is fundamental for creating publication-quality data visualizations. The techniques demonstrated throughout this guide--leveraging the sequential drawing commands `plot()`, `lines()`, and `points()`--are the necessary building blocks for all more sophisticated statistical graphics. For users seeking to expand their visualization capabilities beyond simple overlays, exploring parameters related to annotation, custom titles, axis labels, and gridlines is highly recommended.

The following tutorials explain how to perform other common plotting functions in R, providing essential knowledge for creating comprehensive and informative comparative graphs: