How to Easily Overlay Plots in R

Authored by

stats writer

December 4, 2025

Overlaying plots in R is a fundamental technique in data visualization, allowing analysts and researchers to present complex comparative data effectively on a single graphic canvas. This method involves combining multiple datasets or analytical results into one cohesive view, which is essential for identifying trends, making direct comparisons between variables, or contrasting model outputs across different groups.

Unlike modern visualization packages that rely on a layered grammar (such as ggplot2), the base R graphics system uses a sequential drawing approach. You first establish the graphical environment using the initial **plot()** function, which sets the axes and primary display area. Subsequent layers--whether they are additional lines, points, or text annotations--are then added using specialized functions that draw directly onto the existing plot window. This sequential method provides immense flexibility and fine-grained control over every element of the final visualization.

The primary functions facilitating this layering are **plot()**, which initializes the graph, and **lines()** or **points()**, which append secondary data series. By meticulously setting graphical parameters such as color (**col**), line type (**lty**), and point character (**pch**), you can differentiate between overlaid plots clearly. Furthermore, incorporating elements like descriptive titles, axis labels, and a comprehensive legend is critical to ensure the audience can correctly interpret the combined visual information. Mastering these techniques transforms raw data into compelling, easy-to-understand narratives.

The Core Functions for Graphical Overlay in R

The base **R programming language** offers a powerful, yet straightforward, set of tools for generating high-quality graphics. To successfully overlay multiple plots, it is essential to understand the distinct roles of the three primary functions involved. The **plot()** function is mandatory as the initial step; it creates the base plot, defines the coordinate system, and automatically scales the axes based on the first dataset provided. Crucially, the subsequent data added must fall within the range established by this initial call, although axis limits can be manually adjusted using arguments like **xlim** and **ylim** for better coverage.

Once the base plot is established, the **lines()** function is employed to draw continuous line segments connecting data points. This function is typically used when visualizing time series data, trends, or mathematical functions where continuity is important. It requires vectors for both the x and y coordinates and draws directly over the existing graph, respecting the current graphical parameters unless overridden by function arguments. For instance, you can specify a unique color

or line thickness specifically for the overlaid line without affecting the initial plot's settings.

Conversely, the **points()** function is utilized when the goal is to add discrete markers to the existing plot. This is the standard practice when overlaying multiple scatterplots or highlighting specific observations on a line graph. Similar to **lines()**, **points()** takes x and y vectors and applies the markers specified by the **pch** (plotting character) argument. Utilizing both **lines()** and **points()** allows for versatile graphic creation, such as combining a model's fitted line (using **lines()**) with the original observed data points (using **points()**) to assess model fit visually.

Fundamental Workflow: Starting and Adding Layers

The sequential nature of base R graphics dictates a specific and strict workflow. You must always start by calling **plot()** with your first dataset. This action initializes the graphical device and sets the stage for all subsequent elements. Any plotting functions that follow, such as **lines()**, **points()**, **text()**, or **legend()**, will then be drawn sequentially on the same graphical device until the device is explicitly closed or overwritten. Understanding this layering dependency is key to preventing common errors where secondary data might fall outside the initial plot boundaries or fail to display correctly.

The fundamental syntax below illustrates this core concept clearly. We establish the first visualization, perhaps a foundational scatterplot, and then build upon it by adding further data series using functions designed specifically for augmentation. The ability to integrate different types of visualizations--a line plot and a scatterplot--onto the same coordinate plane highlights the operational flexibility inherent in the base R graphics system.

You can use the **lines()** and **points()** functions to overlay multiple plots in R:

```
#create scatterplot of x1 vs. y1
plot(x1, y1)

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

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

It is imperative to manage the visual aesthetics of these overlaid elements carefully. If all lines or points share the default color (black) and symbol type, the resulting visualization will be ambiguous

and difficult to decode, negating the benefit of the overlay. Therefore, the strategic use of graphical parameters, which we explore in the subsequent examples, is not merely for aesthetics but is crucial for maintaining the functional clarity and interpretability of the composite plot.

The following examples show how to use each of these functions in practice.

Example 1: Mastering Overlayed Line Plots

Line plots are exceptionally useful for tracking changes over a continuous domain, typically time or distance. When overlaying multiple line plots, we are often comparing the trajectory of different groups or variables across the same scale. In this first detailed example, we define three separate datasets (x1/y1, x2/y2, and x3/y3) that represent distinct trends we wish to compare directly. The objective is to display all three trends simultaneously, ensuring they are easily distinguishable through color coding.

The visualization begins with the **plot()** function using the first dataset (x1, y1). Note the vital argument **type='l'**, which explicitly instructs R to draw the initial plot as a continuous line instead of the default points. We also assign a color, **col='red'**, to this baseline plot for clear identification. Subsequently, the **lines()** function is used twice, once for x2/y2 and once for x3/y3. For each call to **lines()**, we specify a unique color (blue and purple, respectively) to ensure maximum visual differentiation between the three underlying data series, thereby facilitating direct visual comparison of their slopes and magnitudes.

This example highlights the efficiency of the **lines()** function for augmentation. Since the axes were already defined by the initial **plot()** call, **lines()** only requires the coordinate vectors and the desired aesthetics to draw the new data. Following the plotting commands, we add the **legend()** function. This crucial element maps the aesthetic choices (colors) back to the data series they represent, rendering the complex graph immediately understandable to the viewer.

The following code shows how to overlay three line plots in a single plot in R:

```
#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)
```

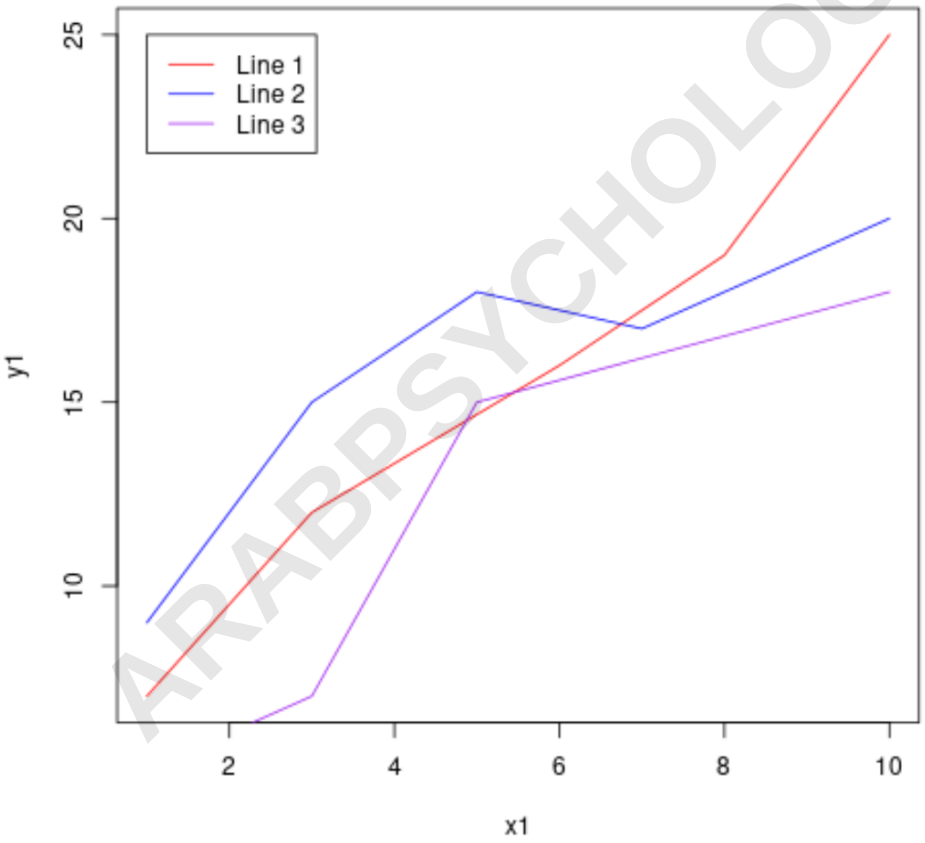
```
y3 = c(5, 6, 7, 15, 18)

#create line plot of x1 vs. y1
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 Crucial Role of Legends and Aesthetics

When working with overlaid graphics, the use of visual differentiation--through color, line type, or point shape--is paramount. However, these aesthetic choices are meaningless without a key to

unlock their meaning. This is where the **legend()** function becomes indispensable. A well-placed legend acts as a map, explicitly linking the visual elements on the plot (like the red, blue, and purple lines) back to their corresponding data labels (Line 1, Line 2, Line 3).

In the line plot example, the **legend()** function call requires several key arguments. First, we specify the coordinates (1, 25) which define the desired location of the legend's top-left corner on the plot. Next, the **legend** argument takes a character vector containing the text labels for each data series. Most importantly, we must ensure that the subsequent aesthetic arguments, such as **col** and **lty** (line type), align perfectly with the arguments used in the preceding **plot()** and **lines()** calls. Mismatches here are a common and frustrating source of errors in R plotting.

Beyond color, graphical parameters such as **lty** (line type: solid, dashed, dotted, etc.) and **lwd** (line width) offer alternative ways to differentiate overlapping lines, which is especially useful when printing graphics in black and white. While using distinct colors is generally preferred for digital displays, relying solely on color can make the visualization inaccessible to color-blind individuals. Therefore, integrating multiple aesthetic cues--like combining unique colors with varied line types or thicknesses--represents a crucial best practice in robust statistical graphics design, ensuring accessibility and clarity.

Example 2: Visualizing Multivariate Data with Overlayed Scatterplots

Scatterplots are the primary visualization tool for exploring the relationship between two continuous variables. When we overlay multiple scatterplots, we are typically comparing how different subgroups within our data relate along the same axes. This technique is highly effective for identifying distinct clustering patterns, assessing separation between categories, or determining how a third categorical variable influences the bivariate relationship between X and Y.

In this example, we compare two distinct datasets, (x1, y1) and (x2, y2). The initial plot is generated using **plot(x1, y1)**, where we specify the color as **col='red'** and, crucially, the point shape using **pch=19**, which designates a solid circle. Because the default setting for **plot()** is to draw points (type='p'), we do not explicitly need the **type** argument. The second dataset is then added using the **points()** function. We again specify a unique color (blue) and maintain the same point shape (pch=19) for consistency, ensuring that the only differentiating factor is the color.

The **points()** function is the dedicated tool for adding discrete markers to an existing plot without connecting them. Unlike **lines()**, which emphasizes continuity, **points()** emphasizes individual

observations. Once again, the graph is completed with a call to **legend()**. For scatterplots, the **legend()** function requires the **pch** argument to be specified within the legend call, ensuring that the markers displayed in the key accurately match those used in the plot itself. This provides a clear, accurate visual representation of the data stratification.

The following code shows how to overlay two scatterplots in a single plot in R:

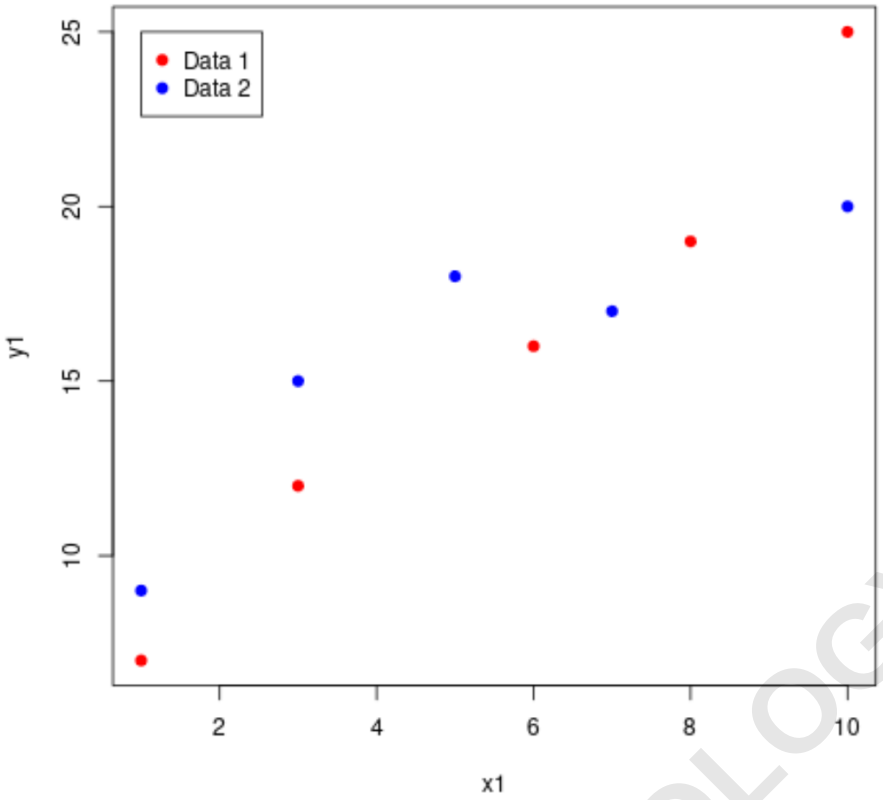
```
#define datasets
x1 = c(1, 3, 6, 8, 10)
y1 = c(7, 12, 16, 19, 25)

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'))
```



Advanced Customization: Using `pch` and Other Graphical Parameters

The argument **pch** (plotting character) is central to customizing scatterplots in R. It dictates the symbol used to represent each data point. The argument takes integer values ranging from 0 to 25, each corresponding to a unique shape. As noted in the example above, a **pch** value of 19 specifies a solid, filled-in circle. Utilizing the full range of **pch** values allows for high-level discrimination between many data groups simultaneously, particularly important when overlaying more than two datasets.

The range of available point characters includes values 0 through 14, which represent hollow or open shapes (squares, triangles, crosses, etc.), and values 15 through 20, which represent solid, filled shapes. A special set of values, 21 through 25, offers specialized symbols that can be customized using two distinct colors: one for the border (controlled by **col**) and one for the internal filling (controlled by **bg**, or background color). This dual-color capability offers sophisticated options for visualizing complex multivariate data structures where two different aesthetic mappings need to be applied to a single point.

Beyond **pch**, several other graphical parameters are essential for professional plotting in R. The

cex argument controls the character expansion factor, allowing you to increase or decrease the size of points or text. Similarly, **lwd** controls line width, enhancing the visual prominence of specific overlaid lines. When preparing graphics for publication or presentation, the careful calibration of these parameters--along with thoughtful color selection--ensures that the visualization is not only mathematically accurate but also visually impactful and accessible to the intended audience.

Note that the **pch** argument specifies the shape of the points in the plot. A pch value of 19 specifies a filled-in circle.

You can find a complete list of pch values and their corresponding shapes.

Conclusion and Next Steps in R Plotting

Overlaying plots in R, utilizing the foundational functions **plot()**, **lines()**, and **points()**, is a core skill for any data scientist or analyst working within the R environment. This technique allows for the efficient creation of rich, comparative visualizations crucial for exploratory data analysis and formal reporting. We have demonstrated how to overlay both line plots and scatterplots, emphasizing the importance of aesthetic controls like **col**, **lty**, and **pch** for maintaining graphical clarity.

The successful implementation of overlaid plots relies heavily on meticulous attention to graphical parameters and the proper use of the **legend()** function. Always prioritize clear visual distinction between data series and ensure that all custom aesthetic choices are accurately documented in the legend. While base R graphics offer profound control, advanced users often transition to systems like ggplot2 for handling very complex multivariate data due to its simplified grammar of graphics approach, though mastering the base system remains foundational to understanding how R handles graphical output.

To further enhance your R plotting expertise, consider exploring tutorials on manipulating axis labels, adding mathematical annotations using **mtext()** or **text()**, or using the **par()** function to create multi-panel plots (faceting). These techniques build directly upon the overlay skills learned here, enabling the creation of advanced, publication-quality statistical visualizations suitable for any analytical context.

The following tutorials explain how to perform other common plotting functions in R: