Learning to Create Multivariate Scatterplots in R for Data Visualization

Authored by

Mohammed Iooti

November 3, 2025

RECOMMENDED CITATION

Mohammed Iooti (2025). *Learning to Create Multivariate Scatterplots in R for Data Visualization*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=9249

Visualizing Complex Relationships: Multivariate Scatterplots in R

Creating effective data visualizations is the cornerstone of robust statistical analysis. While the classic scatterplot is indispensable for illustrating the correlation between two variables, advanced analytical tasks often demand the simultaneous visualization of relationships involving multiple variable pairs on a single canvas. This technique, known as multivariate plotting, is crucial for comparing divergent distributions, assessing differences in scale, or discerning contrasting trends across distinct datasets within a unified visual framework.

In the versatile R programming environment, this comparative visualization is efficiently accomplished using the fundamental capabilities of the base graphics system. The methodology involves layering several distinct sets of data points onto the same coordinate plane, resulting in a coherent and readily interpretable graphic. This guide provides an in-depth exploration of how to create, refine, and accurately interpret scatterplots that feature multiple variables, utilizing the foundational plot() function and associated tools available within base R.

The successful execution of this data layering process requires careful and sequential command execution. The initial plotting command must correctly establish the boundaries, axes, and primary structure of the visualization. Subsequent datasets are then carefully added, ensuring they integrate seamlessly without overriding the established graphical context. Mastering this fundamental layering technique is essential for any analyst seeking to produce high-quality, comparative statistical graphics in **R**.

The Foundational Base R Syntax for Layering Plots

To construct a robust multivariate scatterplot in R, we rely heavily on the precise interplay of three primary functions from the base graphics system. First, the plot() function is executed to initialize the plot area, define the axes limits based on the first dataset, and render the initial set of data points. Second, the points() function is subsequently employed to append additional datasets to the existing graph without resetting any established graphical parameters.

Finally, the legend() function is absolutely critical for the interpretability of the graph, as it maps the various colors, symbols, and line types back to their corresponding data series. This ensures that viewers can accurately distinguish between the variables being compared. The visual separation of datasets is achieved immediately through the careful use of the `col` (color) argument during both the initial plotting and the layering stages.

The following syntax illustrates the general structure required for plotting two distinct sets of (X, Y) coordinates, here labeled as (x1, y1) and (x2, y2). Note the precise placement of the functions and the necessary inclusion of parameters like color and plotting characters (`pch`) to maintain visual clarity. When positioning the legend using coordinates (e.g., 1, 25), analysts must ensure the

legend box does not obstruct critical data points or obscure important trends.

 # Initialize the plot area and draw Data 1 (x1 vs. y1)
plot(x1, y1, col='red')

# Layer the second dataset (x2 vs. y2) onto the existing canvas
points(x2, y2, col='blue')

# Add the legend, linking colors and symbols to their data series
legend(1, 25, legend=c('Data 1', 'Data 2'), pch=c(19, 19), col=c('red', 'blue'))

It is vital to grasp the functional difference between `plot()` and points(). If an analyst were to mistakenly use `plot()` for the second dataset (x2, y2), the entire graphical environment--including the first dataset and the original axis definitions established by the first command--would be instantly overwritten. Using points() guarantees that the new dataset is added cumulatively, preserving all previously drawn elements and the overall graphical context.

Example 1: Practical Implementation of a Dual-Variable Scatterplot

Our first practical demonstration illustrates the process of defining and visualizing two small, distinct data series on a shared graphical plane using the core base R functions previously introduced. We begin by defining `x1` and `y1` as our primary independent and dependent variables, and `x2` and `y2` as the comparative dataset. Both sets of data are instantiated using simple numeric vectors, which is typical for initial data exploration within the **R environment**.

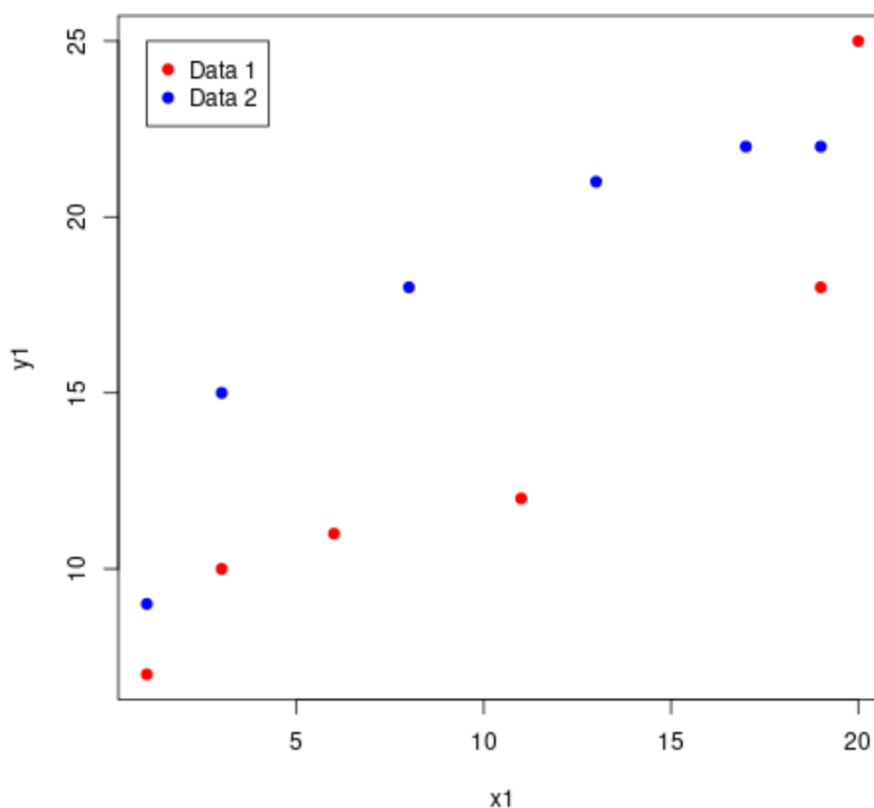
In this specific implementation, we utilize the important graphical parameter `pch=19`. The **`pch`** argument dictates the plotting character used for the points; the value 19 specifically generates solid, filled circles. These solid symbols are generally preferred in visualization because they offer superior visual contrast and clarity compared to open symbols, especially when datasets are dense or overlapping. Consistency is key: this parameter must be applied in both the `plot()` and points() calls.

Following the rendering of both datasets, the legend() function is invoked. It is imperative that the parameters passed to the legend call--specifically `pch` and `col`--are an exact match for the settings used to draw the data series. This ensures that the legend accurately maps the visual characteristics (color and shape) back to the corresponding data series labels (Data 1 and Data 2).

 # Define datasets for two variables
x1 = c(1, 3, 6, 11, 19, 20)
y1 = c(7, 10, 11, 12, 18, 25)

```
x2 = c(1, 3, 8, 13, 17, 19)
y2 = c(9, 15, 18, 21, 22, 22)

# Create the initial scatterplot (x1 vs. y1) with solid red circles
plot(x1, y1, col='red', pch=19)

# Add the second scatterplot (x2 vs. y2) with solid blue circles
points(x2, y2, col='blue', pch=19)

# Add legend to identify the data series
legend(1, 25, legend=c('Data 1', 'Data 2'), pch=c(19, 19), col=c('red', 'blue'))
```

Upon execution, the resulting visual output clearly delineates the two distinct data series. This clear differentiation enables straightforward comparison of their respective trends, distributions, and potential correlations across the shared X and Y axes, facilitating rapid analytical insight.



Example 2: Customizing Visual Parameters for Enhanced Clarity and Professionalism

While a basic plot successfully renders the data, effective statistical graphics must transcend

functionality to achieve clear communication. This requires meticulous labeling and aesthetic refinement. Base R offers an extensive array of graphical parameters designed to control every aspect of appearance, including titles, axis labels, and the size of the plotted symbols. This second example demonstrates how to seamlessly integrate these essential parameters directly into the initial plot() function call.

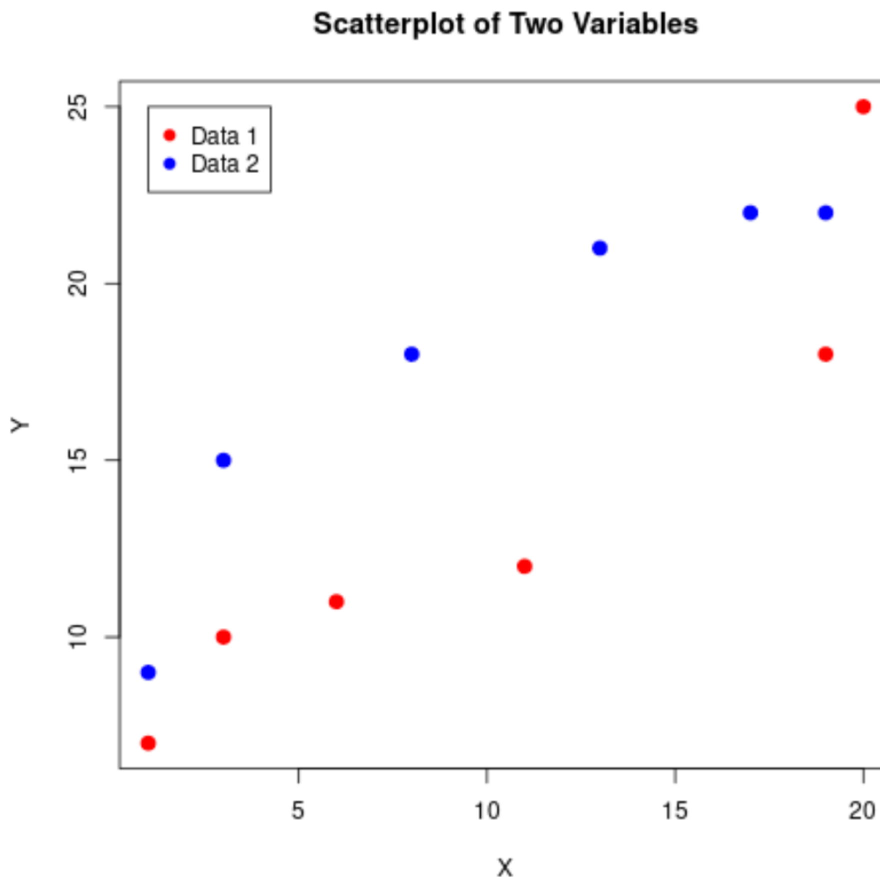
We introduce three critical customization arguments to enhance context: `xlab` and `ylab` are used to define precise, descriptive labels for the horizontal (X) and vertical (Y) axes, respectively. The `main` argument sets the primary title of the plot, immediately orienting the viewer to the visualization's purpose. Additionally, we leverage the **`cex`** argument, which controls the character expansion factor of the plotting symbols. By setting `cex=1.3`, we increase the size of the points by 30% relative to the default setting, making them visually more prominent and easier to track, especially where data points are sparse or highly concentrated.

A fundamental rule in base R plotting is that parameters defining the overall canvas--such as `xlab`, `ylab`, and `main`--must be included exclusively within the initial `plot()` call. Conversely, parameters that define the appearance of individual points, such as `col`, `pch`, and `cex`, must be consistently repeated in every subsequent points() function call. This ensures that the visual characteristics of all layered datasets are uniform and intentional, leading to a cohesive and professional graphic.

```
# Define datasets (Utilizing the same data as Example 1)
x1 = c(1, 3, 6, 11, 19, 20)
y1 = c(7, 10, 11, 12, 18, 25)

x2 = c(1, 3, 8, 13, 17, 19)
y2 = c(9, 15, 18, 21, 22, 22)

# Create initial scatterplot with full customization (title, axes, larger points)
plot(x1, y1, col='red', pch=19, cex=1.3,
xlab='X Variable Measurement', ylab='Y Variable Measurement', main='Comparative Scatterplot of Two Data Series')

# Overlay second scatterplot (Must match 'pch' and 'cex' for visual balance)
points(x2, y2, col='blue', pch=19, cex=1.3)

# Add legend, ensuring coordinates (1, 25) fit the expanded canvas
legend(1, 25, legend=c('Data Series 1', 'Data Series 2'), pch=c(19, 19), col=c('red', 'blue'))
```

The final plot resulting from this code is professional grade: clearly titled, axes are descriptively labeled, and the visual elements are appropriately sized. This comprehensive customization ensures the graphic immediately conveys the necessary context and facilitates effective

interpretation of the relationship between the measured variables.



Mastering Graphical Parameters: Differentiating Data with pch and cex

The overall analytical power of a multivariate scatterplot is often highly dependent on the deliberate and thoughtful selection of specific graphical parameters. Among the most critical parameters for visually separating layered data series are `pch` (plotting character) and `cex` (character expansion factor). These tools allow analysts to create contrast and prevent confusion when multiple points occupy the same visual space.

The **`pch`** argument determines the symbol used for drawing data points. R offers 25 standard symbols, each represented by a numerical value. For example, a value of 1 represents an open circle, 15 represents a solid square, and 19 specifies a solid, filled-in circle. When overlaying three or more distinct datasets, relying solely on color (`col`) is often insufficient. Best practice dictates using a combination of clearly distinct `pch` values--such as 19 (filled circle), 17 (filled triangle), and 15 (filled square)--in conjunction with varying colors to maximize visual differentiation and ensure accurate data identification via the legend.

The **`cex`** argument is crucial for scaling the size of the plotted symbols. While `cex=1` is the

established default size, adjusting this parameter is essential in two primary analytical situations: first, when data points are sparse and need increased visual prominence to prevent being overlooked; and second, conversely, when data density is high, requiring slightly smaller symbols (e.g., `cex=0.8`) to mitigate the issue of overplotting, where too many points overlap and obscure underlying patterns. It is imperative to maintain consistency: the `cex` value must be identical across all calls to plot() and points() if the various datasets are intended to hold equal visual importance.

Advanced Visualization Techniques and Further Resources

For data scientists and analysts seeking to move beyond the base R graphics system, the **ggplot2 package** is universally recommended. ggplot2 implements the influential grammar of graphics approach, which fundamentally simplifies the creation of intricate, layered visualizations, including highly complex multivariate scatterplots, especially when managing data organized within data frames. This package offers a declarative syntax for managing aesthetics, scaling, and layering that many users find significantly more intuitive and flexible than traditional base R commands.

However, to gain deeper, low-level control over base R plots, continuous review of the official documentation for the primary graphics functions--`plot`, `points`, and legend--remains indispensable. This documentation covers advanced topics necessary for professional output, such as manually controlling axis limits (using `xlim` and `ylim`), managing the visibility and location of tick marks (`xaxt` and `yaxt`), and implementing sophisticated, custom color palettes. A strong understanding of these underlying mechanisms provides a foundational skill set crucial for all forms of data visualization in R.