Learning to Create Grouped Scatter Plots in R: A Step-by-Step Guide

Authored by

Mohammed looti

October 30, 2025

Introduction to Visualizing Relationships by Group in R

Scatter plots are perhaps the most fundamental visualization technique in data analysis, primarily used to quickly illustrate the relationship, correlation, or lack thereof between two numerical variables.
However, standard plots often fail to capture complexities hidden within heterogeneous datasets. When analyzing real-world data, it is absolutely essential to determine if the established relationship changes across different categories or segments within your sample.
This comprehensive tutorial focuses on the crucial skill of creating effective scatter plots that visually differentiate data points based on a grouping variable using R, the leading statistical programming environment.

The ability to stratify data visualization by group reveals vital analytical insights that an aggregated view might obscure. For instance, if you are comparing medical treatment effects, demographic trends, or different experimental conditions, visualizing these segments separately is non-negotiable.
By mapping distinct colors, shapes, or sizes to specific groups on your scatter plots, you significantly enhance the interpretability and clarity of your findings.
We will explore two powerful, yet distinct, approaches within R: utilizing the native **Base R** graphics system for simplicity and speed, and leveraging the highly flexible **ggplot2** package for advanced, publication-ready aesthetics.

Contrasting Visualization Methods: Base R vs. ggplot2

R provides analysts with two primary and equally robust methods for generating customized scatter plots that effectively handle grouping variables. Choosing between them often depends on the required level of customization, complexity, and the ultimate destination of the visualization (e.g., quick exploratory analysis versus formal publication).

Method 1: Utilizing Base R Graphics

The **Base R** graphics system is the built-in default environment for plotting in R. It offers a direct, procedural, and highly efficient way to create plots without relying on external packages.
This method is perfectly suited for quick exploratory data visualization (EDV) and for users who prefer a minimalist, console-based approach.
The core function for generating scatter plots is `plot()`, which can be easily adapted by assigning a color scheme based on the categorical grouping variable, as demonstrated below:

plot(dfx, dfy, col=as.factor(df$group))

Method 2: Employing the ggplot2 Package

For visualizations that require aesthetic sophistication, high customization, and adherence to the principles of the "grammar of graphics," the ggplot2 package is the industry standard within the R ecosystem.
ggplot2 constructs plots layer by layer, starting with data and aesthetic mappings, then adding geometric objects (like points or lines).
This structured approach provides unparalleled control over every visual element, making it the ideal choice for generating professional, publication-quality graphics.

library(ggplot2)

ggplot(df, aes(x, y)) +
geom_point(aes(color=group))

Essential Data Preparation: Defining the Sample Data Frame

To effectively demonstrate the plotting examples, we must first define the structure and content of the data we will be using. In R, the primary structure for tabular data is the data frame--a collection of vectors of equal length, where each column can hold different data types.
For our purposes, we will construct a simple data frame named `df`, which contains the bare minimum variables necessary for grouped plotting: two continuous numerical variables (`x` and `y`) and one categorical grouping variable (`group`).

The following code snippet creates and displays the sample data frame. Notice how the `group` column contains distinct character values ('A' and 'B'). These categorical levels will drive the visual differentiation in our subsequent plots, allowing us to compare the relationship between X and Y for each specific group.

#create data frame
df <- data.frame(x=c(1, 2, 2, 3, 5, 6, 7),
y=c(4, 8, 7, 9, 15, 14, 20),
group=c('A', 'A', 'A', 'B', 'B', 'B', 'B'))

#view data frame
df

x y group
1 1 4 A
2 2 8 A
3 2 7 A
4 3 9 B
5 5 15 B

6 6 14 B
7 7 20 B

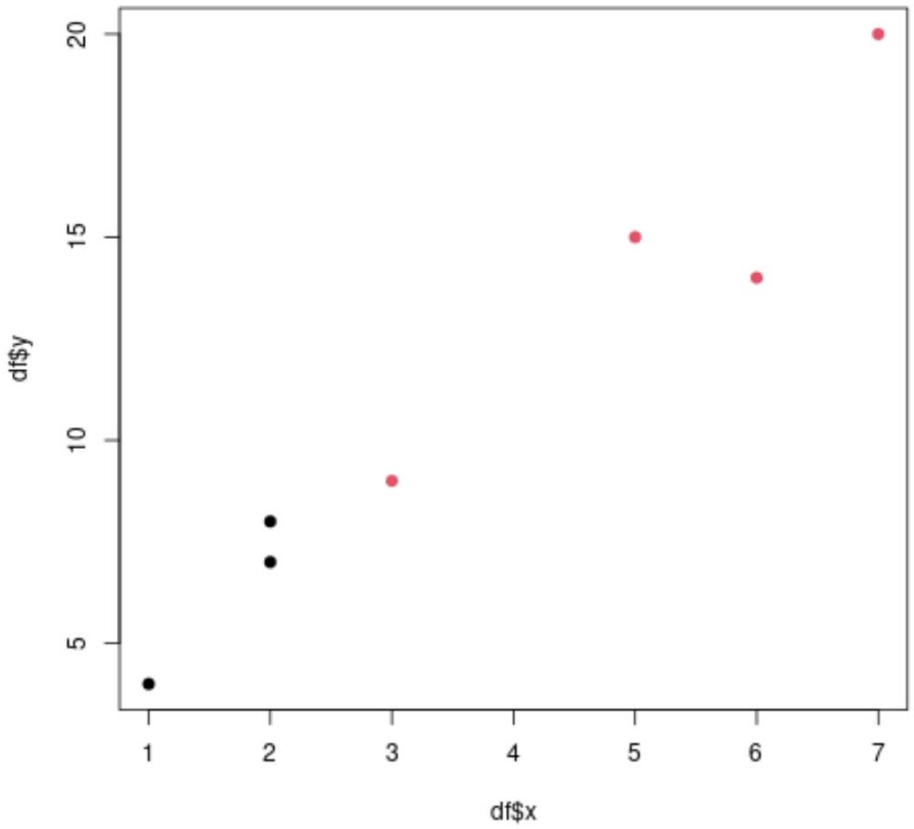
This structure is ideal for demonstrating grouped visualization. The 'x' and 'y' variables define the coordinates of the points, while the 'group' variable provides the mechanism for coloring or shaping those points differently, ensuring a clear visual separation between the subsets of observations.

Example 1: Grouped Scatter Plots Using Base R Graphics

The **Base R** `plot()` function is powerful and highly versatile, enabling the creation of grouped visualizations with minimal code. To distinguish points by their assigned group, we must ensure that the grouping variable is treated as a factor when mapped to the color aesthetic.

In the code below, we use the `col` argument within the `plot()` function. By passing `as.factor(df$group)` to this argument, we instruct R to interpret the categorical variable and automatically assign a distinct color (or numerical index) to each unique level ('A' and 'B'). Additionally, we use `pch=19` to specify that the points should be solid circles, maximizing visibility.

#create scatterplot with points colored by group
plot(dfx, dfy, col=as.factor(df$group), pch=19)



The resulting plot immediately highlights the difference between the groups. Points belonging to 'group A' are automatically rendered in black (color index 1 in Base R's default palette), while points for 'group B' are displayed in red (color index 2). This immediate visual separation is invaluable for initial exploratory analysis, allowing the analyst to quickly identify potential differences in trends or slopes between the subsets of the data frame.

The use of the `pch` argument is critical for defining the graphical marker. While `pch=19` creates solid circles, R supports a wide array of plotting characters, enabling further visual differentiation if needed. You can use different `pch` values to assign unique shapes to each group, offering an additional layer of clarity, especially if the plot is printed in grayscale. For a full list of available symbols and their corresponding values, consulting the R Plotting Symbols documentation is highly recommended.

Example 2: Achieving High Aesthetics with the ggplot2 Package

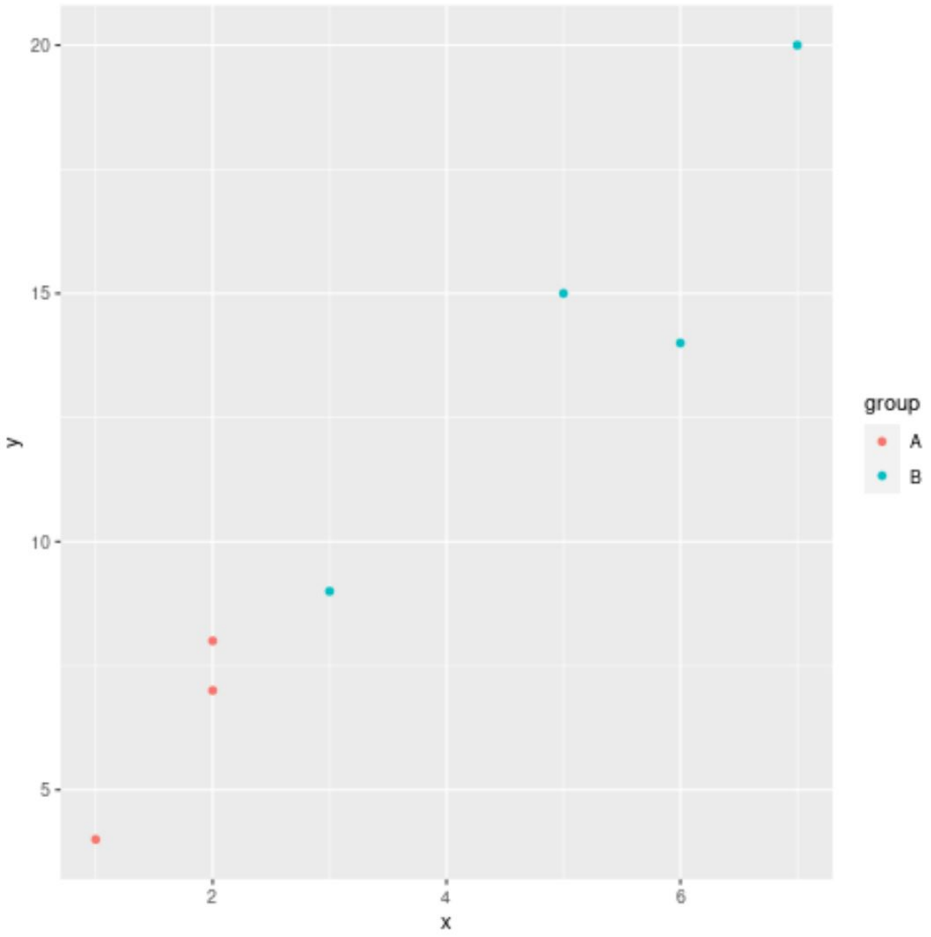
When moving beyond quick exploration towards sophisticated data presentation, the ggplot2 package offers unmatched control and visual quality. ggplot2 operates on the principle of mapping data variables to visual aesthetics, making the process of grouping intuitive and systematic.

To begin, we must load the necessary library. The plot construction starts with the `ggplot()`

function, which defines the dataset (`df`) and the core aesthetic mappings (`x` and `y`). We then add the geometric layer, `geom_point()`, which is responsible for drawing the scatter plot points. The grouping instruction is passed directly within the aesthetics of this geometric layer.

library(ggplot2)

#create scatterplot with points colored by group
ggplot(df, aes(x, y)) +
geom_point(aes(color=group))



The key to grouping here lies in `aes(color=group)`. By mapping the categorical variable `group` to the `color` aesthetic, ggplot2 automatically handles two critical tasks: first, it assigns a visually distinct color palette to each level of the variable; and second, it automatically generates a clear, labeled legend, eliminating the manual legend creation required in Base R. This automation simplifies the workflow and ensures the resulting visualization is immediately interpretable.

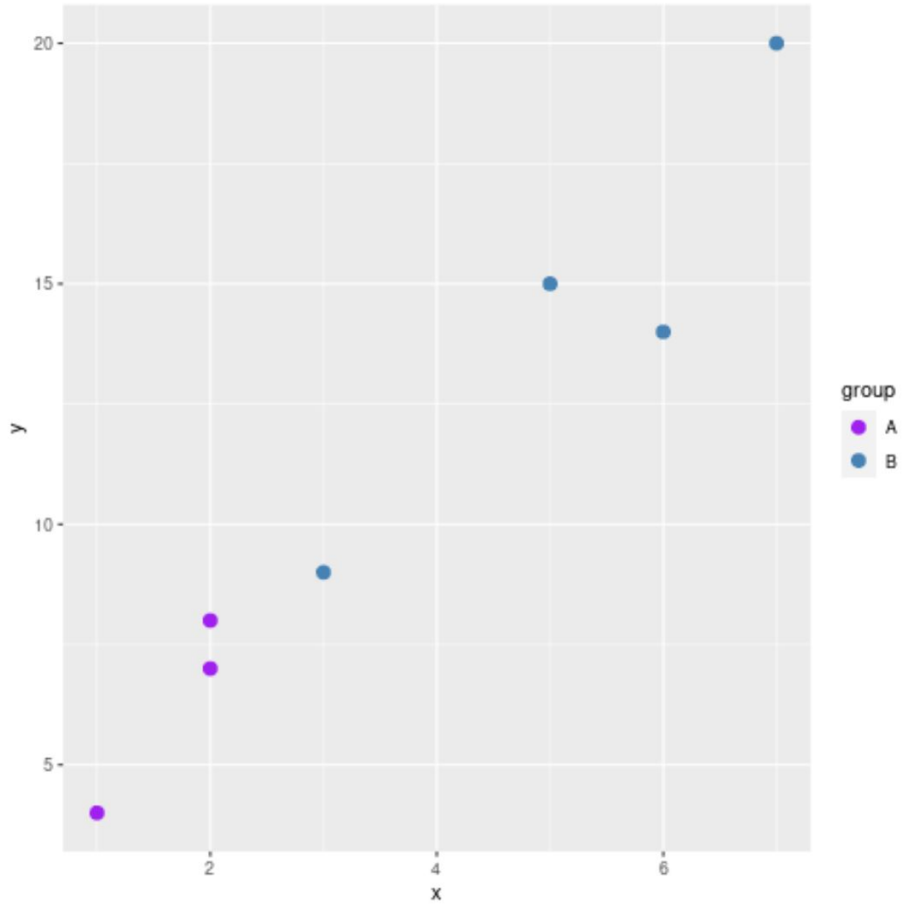
Advanced Customization: Tailoring ggplot2 Visualizations

One of the foremost reasons analysts favor ggplot2 is its capacity for detailed customization, allowing users to move beyond default colors and sizes to create graphics that meet specific branding or publication requirements. Customizing the visual characteristics is achieved by adding subsequent layers to the plot object.

We can enhance the visual impact of the grouped scatter plot by controlling the size of the points and manually overriding the default color scheme. The following code demonstrates these two essential customization techniques, resulting in a more polished and tailored visualization:

```
library(ggplot2)

#create scatterplot with points colored by group
ggplot(df, aes(x, y)) +
geom_point(aes(color=group), size=3) +
scale_color_manual(values=c('purple', 'steelblue'))
```



The first modification, `size=3`, is placed outside the main aesthetic mapping within `geom_point()`, making all points uniformly larger, which increases their prominence. The second, and more powerful, modification is the addition of the `scale_color_manual()` layer. This function allows the user to define a vector of specific colors (in this case, 'purple' and 'steelblue') that will be mapped directly to the levels of the grouping variable ('A' and 'B', in order). This precise control is critical for adhering to institutional color palettes or for maximizing contrast in presentations.

Summary and Next Steps in R Visualization

Creating grouped visualizations is a fundamental technique for performing insightful data analysis. By applying grouping variables to the color or shape aesthetics of a plot, you can effectively compare and contrast relationships across different segments of your dataset.
This tutorial has provided clear pathways for achieving this goal using two of the most popular methods in R: the straightforward efficiency of **Base R** and the advanced aesthetic control offered by ggplot2.
Mastering these techniques ensures that your data representations are not only accurate but also visually compelling and easy for any audience to understand.

To continue developing your visualization expertise in R, we recommend exploring more advanced topics that build upon these foundational skills:

Adding trend lines or regression lines to grouped scatter plots to quantify relationships.
Creating interactive scatter plots using packages like Plotly, allowing users to hover over points for detailed information.
Customizing plot legends, titles, and axis labels for strict adherence to academic or professional publication standards.