Learning to Display Multiple ggplot2 Plots in R: A Step-by-Step Guide

Authored by

Mohammed looti

November 12, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Learning to Display Multiple ggplot2 Plots in R: A Step-by-Step Guide*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=23852

The Challenge of Displaying Multiple R Visualizations

The ability to create compelling charts and graphs is fundamental to effective data analysis. Within the R programming language,
one of the most powerful and widely adopted libraries for this purpose is **ggplot2**.
Built upon the grammar of graphics, **ggplot2** allows analysts to construct highly customizable and aesthetically pleasing visualizations,
ranging from simple bar charts to complex statistical plots. However, a common necessity in advanced reporting or comparative analysis is the requirement to
display several distinct plots simultaneously within a single viewing window or page.

While **ggplot2** excels at generating individual plots, it does not inherently provide a straightforward function for combining multiple plot objects into a composite layout.
Analysts frequently need to compare different facets of a dataset, such as various relationships between variables or the same relationship across different data subsets.
Presenting these comparisons side-by-side or stacked vertically significantly enhances the clarity and impact of the overall data visualization narrative,
making efficient plot arrangement a critical skill for any R user.

This need for flexible plot arrangement led to the development of several utility functions and packages designed to manage layouts within R's graphical environment.
One of the most accessible and historically significant methods for achieving this goal involves using the **multiplot()** function.
This function simplifies the complex process of defining graphical parameters and viewports required to organize multiple **ggplot2** objects effectively on a single canvas,
allowing users to focus more on the analytical insights and less on the underlying rendering mechanics.

Introducing the multiplot() Function and Syntax

The most direct route to arranging multiple plots created with **ggplot2** is by utilizing the **multiplot()** function.
This specialized function is typically sourced from the **scater** package,
which provides a convenient wrapper around R's base graphics engine mechanisms, specifically leveraging the **grid** package for layout management.
Before executing this function, it is essential to ensure that both **ggplot2** (to create the plot objects) and **scater** (to provide the **multiplot()** function) are loaded into the current R session using the `library()` command.

The syntax for **multiplot()** is designed to be intuitive, accepting the plot objects as primary

arguments followed by an optional parameter to control the column structure.
The basic structure is highly adaptable, allowing for the inclusion of any number of plot objects that have been previously defined and stored as variables in the R environment.
The core syntax is demonstrated below, illustrating how multiple plot objects (p1, p2, p3, etc.) are passed to the function:

multiplot(p1, p2, p3, col=N, ...)

The key arguments used within this function call are defined as follows:

p1, **p2**, **p3**, etc.: These represent the names of the individual plot objects, which must be created using the **ggplot2** library prior to the function call. The order in which they are listed determines their placement in the resulting layout grid.

col: This crucial argument specifies the desired number of columns to be used when arranging the plots on the display page. The function automatically calculates the necessary number of rows based on the total number of plots provided and the specified column count, aiming for the most efficient use of space.

It is important to note the default behavior of the **multiplot()** function. If the optional **col** argument is omitted entirely from the function call, the mechanism defaults to using a single column (i.e., `col=1`). This results in all supplied charts being displayed in a vertical stack, one above the other, which is suitable for sequential viewing or when the plots are significantly taller than they are wide. Understanding this default behavior is key to controlling the final visual presentation of the combined graphics.

Practical Demonstration: Using multiplot() for Side-by-Side Views

To illustrate the practical application of **multiplot()**, we will use the well-known **mtcars** dataset, which is conveniently built into R and contains data on various characteristics of 32 automobiles. Our objective is to create two distinct **scatterplots** from this dataset and display them adjacently, allowing for an immediate visual comparison of two different relationships involving engine displacement (`disp`).

We begin by defining the two plot objects, `p1` and `p2`, using standard **ggplot2** syntax. Plot `p1` will visualize the relationship between engine displacement and quarter-mile time (`qsec`), while plot `p2` will map engine displacement against miles per gallon (`mpg`). Once these graphical objects are defined, we invoke **multiplot()**, explicitly setting the number of columns to two (`cols=2`) to ensure a horizontal arrangement, which is ideal for viewing side-by-side trends.

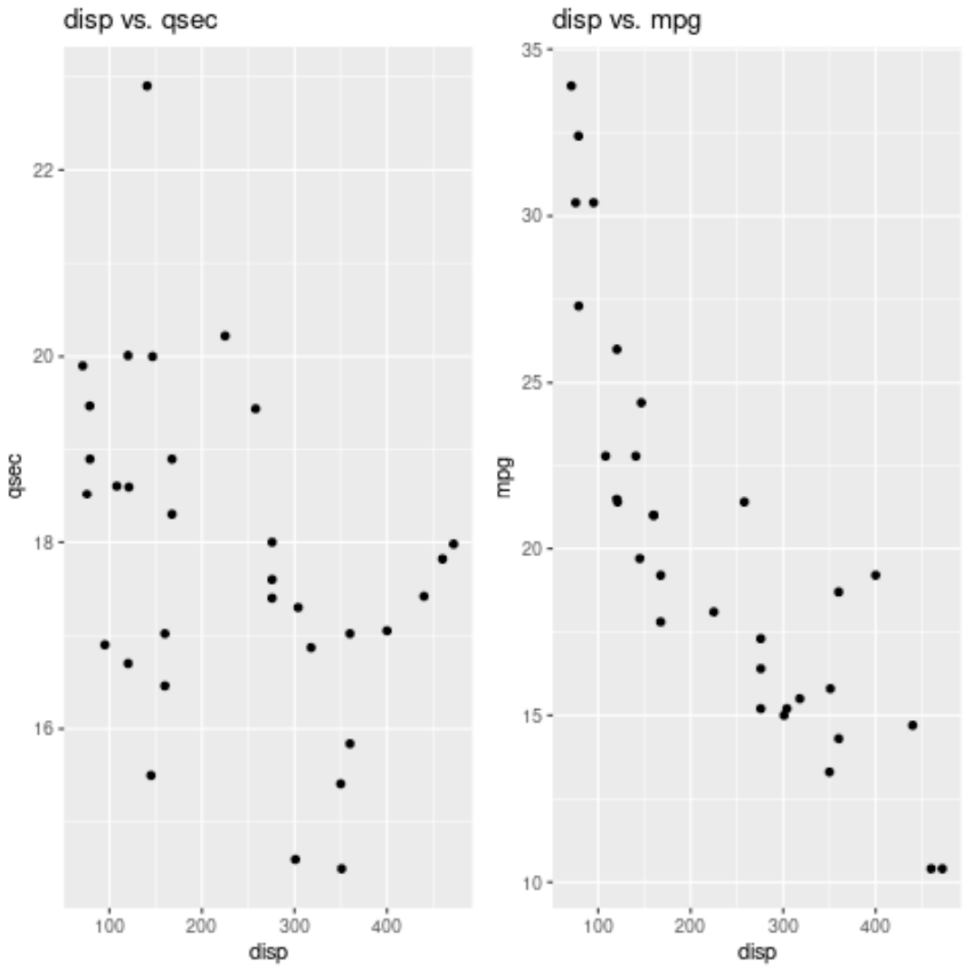
The following R code block demonstrates the necessary steps, including loading the required

packages, defining the plot objects, and ultimately executing the layout command to produce the composite graphic. We use the **scater** package to access the **multiplot()** function, enabling efficient visualization management:

```
#load necessary packages
library(ggplot2)
library(scater)

#create first plot
p1 <- ggplot(mtcars, aes(x=disp, y=qsec)) +
geom_point() +
ggtitle("disp vs. qsec")

#create second plot
p2 <- ggplot(mtcars, aes(x=disp, y=mpg)) +
geom_point() +
ggtitle("disp vs. mpg")

#display both plots on same page using 2 columns
multiplot(p1, p2, cols=2)
```

Execution of the code above yields a single output graphic where the two scatterplots are arranged horizontally. Because we specified **col=2**, the **multiplot()** function successfully organized the two plots into a 1x2 grid structure, presenting them next to each other for immediate visual analysis. This configuration is particularly useful when the plots share a common theme or when direct comparison is the primary goal of the visualization.

This produces the following result:



Configuring Vertical Layouts with multiplot()

While displaying plots side-by-side (horizontally) is often preferred for comparison, there are many scenarios where a vertical arrangement is more appropriate. For instance, if the plots contain numerous labels on the x-axis or if the comparison is sequential rather than simultaneous, stacking the plots can improve readability and ensure that each plot retains adequate space without compression. The **multiplot()** function facilitates this vertical arrangement simply by omitting or explicitly setting the **col** argument to 1.

When the **col** argument is left out, the function automatically defaults to `col=1`. This instructs R to arrange all provided plot objects into a single vertical column. The plots are then placed one on top of the other, following the sequence in which they were listed in the function call (i.e., `p1` appears first, followed by `p2`, and so on). This approach is highly effective for generating long, detailed reports or for outputs destined for print media where page breaks are controlled.

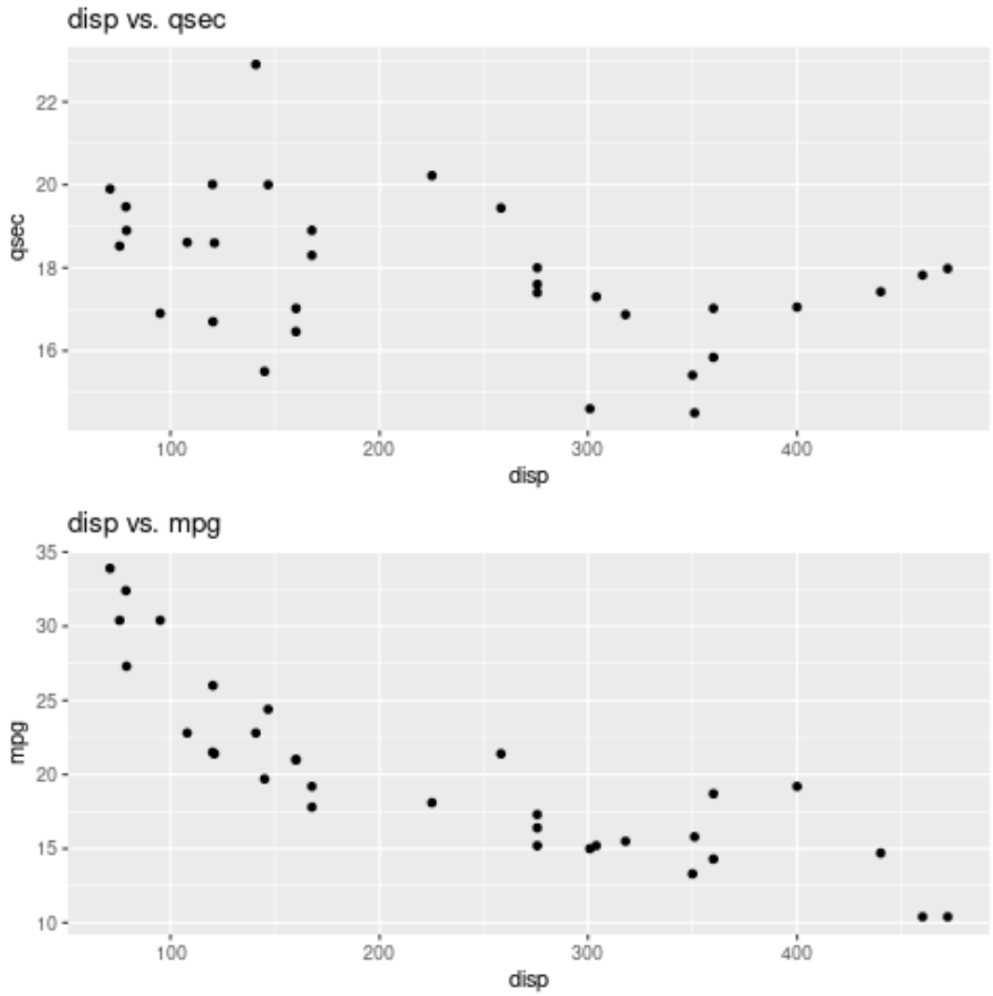
The underlying plot objects (`p1` and `p2`) remain unchanged from the previous example; only the

final function call is modified to demonstrate the vertical configuration. Note that in the example below, we have removed the `cols=2` argument entirely.

```
#load necessary packages
library(ggplot2)
library(scater)

#create first plot
p1 <- ggplot(mtcars, aes(x=disp, y=qsec)) +
geom_point() +
ggtitle("disp vs. qsec")

#create second plot
p2 <- ggplot(mtcars, aes(x=disp, y=mpg)) +
geom_point() +
ggtitle("disp vs. mpg")

#display both plots on same page using 1 column
multiplot(p1, p)
```

Execution of this revised syntax produces the following output, where the plots are now stacked vertically, occupying a single column. This ensures maximum vertical space for each plot, preventing potential crowding issues that might arise if the plots were forced into a horizontal layout with limited screen width.

This produces the following result:



The Robust Alternative: Manually Defining the multiplot() Function

While relying on the **scater** package for the **multiplot()** function is convenient, R environments can sometimes present challenges related to package dependencies, version conflicts, or restricted access to installation repositories. In such cases, or when working in environments where minimizing external package dependencies is critical, analysts often resort to including the definition of the **multiplot()** function directly within their script. This approach ensures that the plot arrangement capability is always available, provided the necessary base R packages, specifically **grid**, are installed.

The manual definition provided below recreates the functionality of the **multiplot()** function. This custom function handles the fundamental steps required for complex plot layouts: gathering the plots, determining the layout matrix based on the specified number of columns (`cols`), initializing a new graphics page (`grid.newpage()`), and iterating through the plots to place them in their designated positions using viewports and the **grid.layout()** function. This method is a reliable fallback and demonstrates a deeper understanding of R's graphical architecture.

If you encounter issues loading **scater** or prefer a self-contained solution, you can prepend the following function definition to your script. Once defined, you can skip the `library(scater)` call and proceed directly to arranging your **ggplot2** objects. This definition ensures maximum compatibility across different R sessions:

```
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
library(grid)

plots <- c(list(...), plotlist)

numPlots = length(plots)

if (is.null(layout)) {
# Make the panel
# ncol: Number of columns of plots
# nrow: Number of rows needed, calculated from # of cols
layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
ncol = cols, nrow = ceiling(numPlots/cols))
}

if (numPlots==1) {
print(plots])

} else {
grid.newpage()
pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

for (i in 1:numPlots) {
# Get the i,j matrix positions of the regions that contain this subplot
matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

print(plots], vp = viewport(layout.pos.row = matchidx$row,
layout.pos.col = matchidx$col))
}
}
}
```

By including this code snippet, you gain independence from external package dependencies for plot arrangement while retaining the full functionality and flexibility of the standard **multiplot()** call. This demonstrates a robust coding practice, especially for production environments or when sharing code with collaborators who may have diverse R configurations.

Summary and Best Practices for Visualization Layouts

The **multiplot()** function, whether sourced from the **scater** package or manually defined, provides a simple yet effective solution for combining multiple **ggplot2** visualizations onto a single page. This functionality is essential for comparative analysis, dashboard creation, and generating comprehensive reports where visual context is paramount. The primary strength of this function lies in its straightforward input requirements: passing the plot objects as arguments and specifying the desired number of columns via the `cols` parameter.

It is important to remember the flexibility of this approach. While our examples focused on combining two plots, the **multiplot()** function is designed to handle an arbitrary number of plot objects. You can easily specify three, four, or more **ggplot2** plots within the function call, and the layout engine will intelligently adapt to the specified number of columns, wrapping the plots onto new rows as needed. This scalability makes **multiplot()** suitable for complex visual summaries.

For users engaged in advanced layout tasks, such as combining plots of different sizes, adjusting relative widths, or nesting complex visual elements, more modern alternatives like the `patchwork` or `cowplot` packages offer additional syntactic simplicity and control. However, for most standard reporting needs requiring simple grids of equally sized plots, **multiplot()** remains a reliable, transparent, and easy-to-implement tool for effective graphical organization in R programming language environments. Mastering this function is a foundational step toward producing professional-quality data reports.