Learn How to Arrange ggplot2 Plots with ggarrange() in R

Authored by

Mohammed looti

November 13, 2025

In the realm of advanced data visualization using the R programming language, analysts frequently need to combine multiple graphical outputs onto a single, cohesive canvas. Achieving a professional, publication-ready layout--especially when comparing related variables or models--demands a solution that goes beyond merely generating individual plots. Effectively arranging these visualizations is critical for maintaining visual consistency and facilitating immediate comparison for the audience. While numerous packages within **R** offer tools for paneling plots, the ultimate goal is always to locate a method that balances high flexibility with straightforward implementation.

Manually positioning and resizing plots often leads to frustrating inconsistencies in margins, alignment, and scale, undermining the integrity of the visual analysis. This challenge highlights the necessity of specialized tools for robust, multi-panel layouts. Common alternatives include packages like **ggpubr** and the highly popular **patchwork** package. However, the **egg** package provides a dedicated, exceptionally efficient function, **ggarrange()**, specifically engineered for combining **ggplot2** objects. This function grants the user precise control over the resulting grid dimensions and the relative sizes of each component plot, making it an indispensable tool for data storytelling.

Mastering the ggarrange() Function Syntax

The **ggarrange()** function, residing within the **egg** package, represents one of the most streamlined approaches for consolidating multiple plot objects generated using the **ggplot2** grammar of graphics. Its core utility lies in its simplicity: users can pass any number of plot objects and then define the desired grid structure using intuitive row and column parameters. A significant advantage of utilizing the **egg** package is its efficient management of the underlying grid system, which automatically ensures plot elements, such as axis labels and titles, are adjusted to fit the combined layout seamlessly without manual intervention.

The fundamental syntax of **ggarrange()** is designed for immediate application while simultaneously offering powerful customization capabilities through various optional arguments. Grasping these parameters is essential for moving beyond simple vertical stacking toward creating complex, asymmetric, and highly tailored panel layouts. By controlling the grid structure and relative plot sizes, analysts can dictate the visual hierarchy of the combined graphic. The function utilizes the following structure, where the parameters listed below are key to controlling the arrangement:

ggarrange(..., nrow=NULL, ncol=NULL, widths=NULL, heights=NULL, ...)

We detail the primary arguments that govern the final visualization structure:

...: This required argument accepts a comma-separated sequence of individual **ggplot2** objects that the user intends to combine into a single figure.

nrow: Defines the exact total number of rows desired for the resulting layout grid. If both **nrow** and **ncol** are omitted during the function call, the default behavior is to arrange all input plots vertically in a single column.
ncol: Defines the total number of columns necessary for the resulting layout grid. This parameter works collaboratively with **nrow** to establish the precise dimensions of the visualization panel.
widths: An optional numerical vector used to specify the relative widths assigned to each column. For instance, `widths = c(2, 1)` would ensure the first column occupies twice the horizontal space of the second column.
heights: An optional numerical vector used to specify the relative heights allocated to each row, operating functionally in the same way as the **widths** argument but for the vertical dimension.

It is important to recognize that the **egg** package functions as an advanced extension to the core **ggplot2** ecosystem. To begin using these functionalities, loading the **egg** package using `library(egg)` is sufficient. While the package typically manages necessary dependencies, explicitly loading **ggplot2** first is a widely accepted best practice in scripting to ensure clarity and avoid potential namespace conflicts when working within **R**.

Practical Example: Preparing the Data

To clearly demonstrate the versatility and power of the **ggarrange()** function, we will employ a foundational, built-in dataset readily available within the **R** environment: the **mtcars** dataset. This dataset originates from the 1974 Motor Trend magazine and contains comprehensive metrics for 32 different automobiles, detailing 11 attributes such as mileage (mpg), horsepower (hp), and vehicle weight (wt). Its manageable size and diversity of continuous variables make it an ideal choice for creating comparative visualizations.

Before proceeding with the generation of plots, a quick structural examination of the data confirms its readiness for graphical analysis. Utilizing the standard **head()** function allows us to inspect the initial observations, providing a clear understanding of variable organization. This crucial preliminary step ensures that the data has been loaded correctly and confirms the exact variable names required for accurate subsequent calls to the **ggplot2** functions.

We inspect the structure of the **mtcars** dataset using the **head()** function:

 # View the initial observations of the mtcars dataset
head(mtcars)

mpg cyl disp hp drat wt qsec vs am gear carb
Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1

Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

Our analytical objective is to generate three distinct scatterplots, each designed to investigate the relationship between Miles Per Gallon (mpg) and key vehicle performance metrics. This systematic comparison will enable us to swiftly assess how fuel efficiency correlates with different mechanical attributes. The three visualizations we plan to generate are:

Plot 1: Fuel Efficiency (mpg) visualized against Vehicle Weight (wt).
Plot 2: Fuel Efficiency (mpg) visualized against Engine Displacement (disp).
Plot 3: Fuel Efficiency (mpg) visualized against Gross Horsepower (hp).

Default Behavior: Vertical Stacking

The most basic and immediate application of **ggarrange()** involves simply passing the desired plot objects without specifying explicit row or column constraints. In this default operational mode, the function prioritizes clarity and immediate display by arranging all input plots vertically in a single column. This stacked layout proves particularly effective when plots are intended to be viewed sequentially, or when horizontal screen real estate is limited. We begin by loading the necessary package and defining our three individual plot objects using standard **ggplot2** syntax, ensuring each plot is ready for combination.

Once the individual plots (`plot1`, `plot2`, and `plot3`) are meticulously defined, they are passed directly into the **ggarrange()** function. The function recognizes that three distinct objects have been supplied and automatically calculates the most space-efficient arrangement, which, by convention, results in a configuration of `nrow=3` and `ncol=1`. This behavior guarantees that, even without explicit parameterization, the visualizations are immediately consolidated and displayed in a unified view.

We use the following syntax with the **ggarrange()** function from the **egg** package to display these visualizations together:

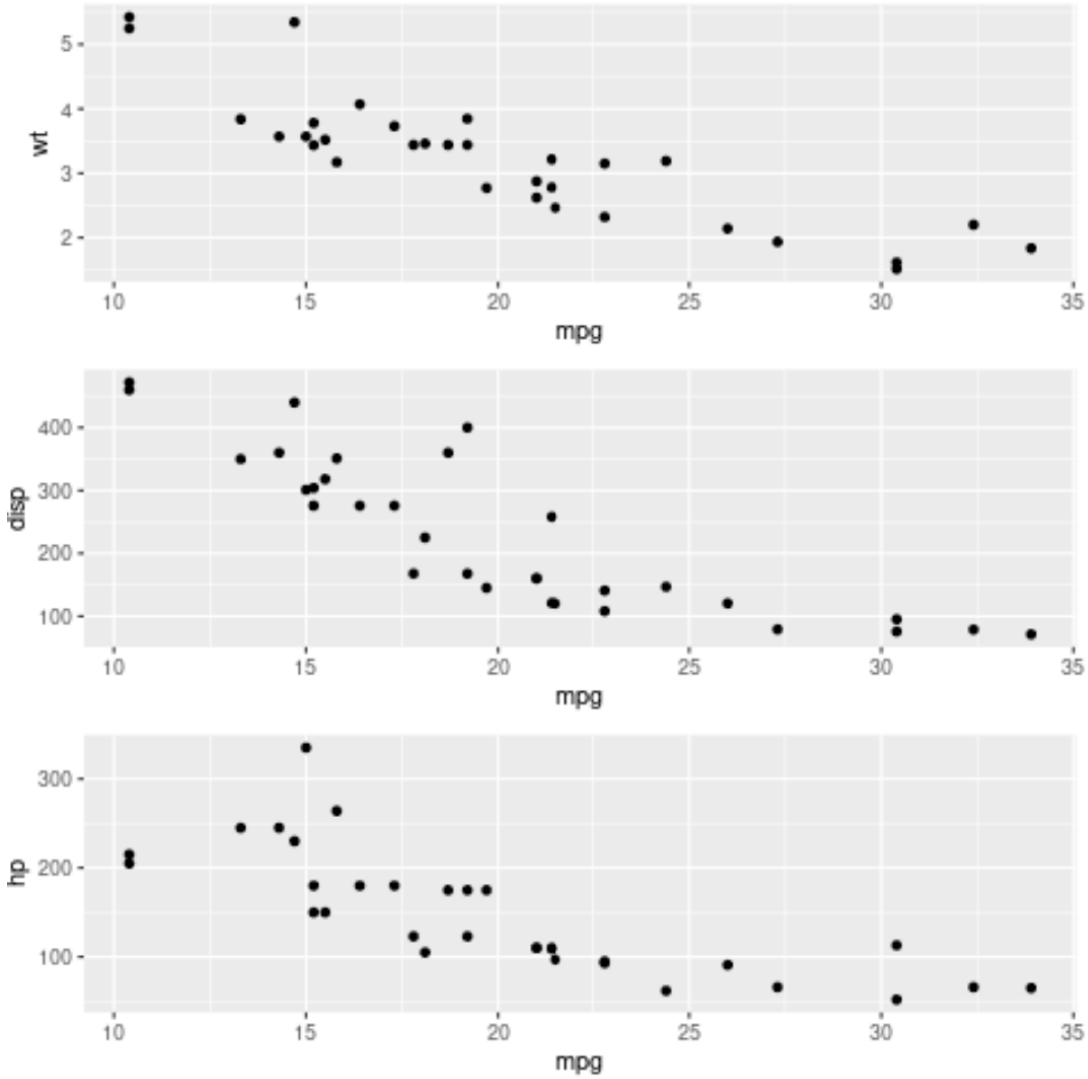
library(egg)

# Generate three scatterplots comparing MPG against three variables
plot1 <- ggplot(mtcars, aes(mpg, wt)) +
geom_point()

plot2 <- ggplot(mtcars, aes(mpg, disp)) +
geom_point()

```
plot3 <- ggplot(mtcars, aes(mpg, hp)) +
geom_point()

# Display all three scatterplots using the default (stacked) layout
ggarrange(plot1, plot2, plot3)
```



As clearly illustrated in the resulting image, all three scatterplots are rendered together on a single page, arranged in a vertical sequence. This visual confirmation underscores the default behavior of **ggarrange()**: when no grid dimensions are manually specified, it maximizes vertical use by stacking the plots into a single column. Understanding this default setting is crucial, as it establishes the baseline from which all more complex layout customizations are derived.

Defining Grids with ncol and nrow for Comparison

While the default single-column layout is perfectly suitable for visualizations intended for sequential viewing, more sophisticated analyses frequently demand that plots be displayed side-by-side to enable immediate and effective visual comparison. The true power of the **ggarrange()** function is unleashed when utilizing the **nrow** and **ncol** arguments, which empower the user to define the precise, intended grid structure for the combined graphic. These arguments provide instantaneous control over the aspect ratio, density, and overall presentation of the resulting visualization.

For instance, if we possess three distinct plots and wish to lay them out horizontally across the page for easy comparison, we explicitly set the number of columns (**ncol**) to three. The function then automatically deduces the necessary number of rows (one, in this scenario). Conversely, if an analyst had five plots and desired a nearly square arrangement, they might choose `nrow=2` and `ncol=3`, allowing the function to fill the grid sequentially, prioritizing row completion. This manual specification ensures that the visual hierarchy of the combined figure aligns perfectly with the underlying analytical intent of the presentation.

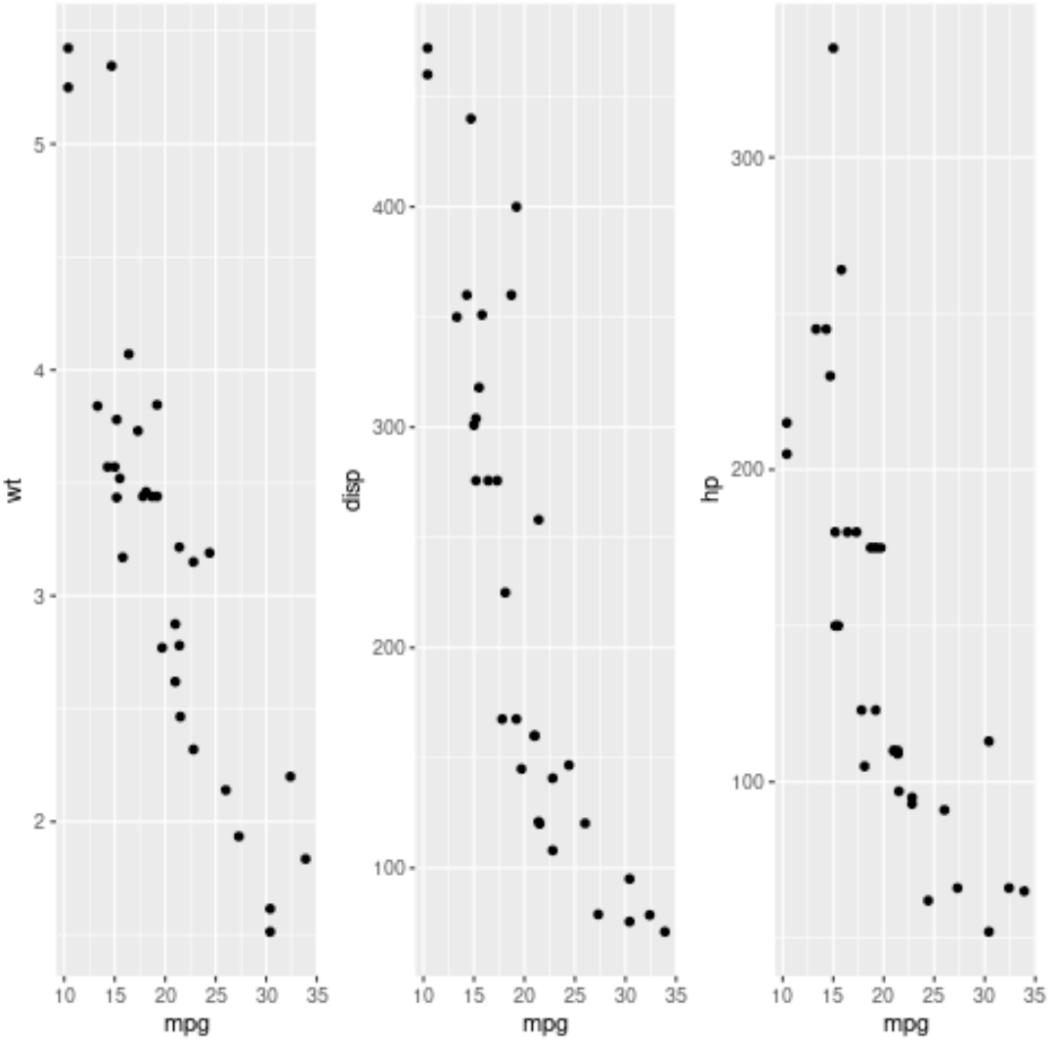
To demonstrate a compelling horizontal arrangement, we will redefine the display settings for our three scatterplots, explicitly requesting a three-column layout. This adjustment forces the plots to be placed adjacent to one another, making the correlations between MPG and the different variables--Weight, Displacement, and Horsepower--significantly easier to compare simultaneously across all three metrics:

library(egg)

Plots 1, 2, and 3 remain defined as before

# Display all three scatterplots using three total columns
ggarrange(plot1, plot2, plot3, ncol=3)

This strategic configuration yields the following horizontal display:



The resulting visual confirms that all three **ggplot2** objects are now perfectly displayed in a single row across three columns, exactly as specified in the function call. Users are granted complete freedom to choose any combination of **nrow** and **ncol** that optimally serves their data visualization needs, whether it necessitates a 2x2 grid, a 4x1 stack, or any other layout required to articulate the analytical narrative effectively.

Advanced Control: Asymmetric Sizing with widths and heights

Moving beyond basic grid arrangements defined by equally sized rows and columns, the **ggarrange()** function provides highly granular control over the relative dimensions of each plot panel through the powerful **widths** and **heights** arguments. These parameters accept precise numerical vectors that assign a specific weight or proportion to each respective dimension, enabling certain plots to occupy proportionally more space than their companions within the overall combined visualization. This capability is absolutely essential when one specific plot contains a critical summary, a detailed view, or a complex distribution that requires significantly more visual

emphasis than the surrounding, contextual plots.

For example, if plots are arranged in a two-column layout (`ncol=2`), setting `widths = c(3, 1)` immediately dictates that the first column will be rendered three times wider than the second. This deliberate creation of an asymmetric horizontal division ensures that the plot placed in the first column is visually dominant. Similarly, when arranging plots into two rows (`nrow=2`), setting `heights = c(0.5, 1.5)` guarantees that the second plot takes up three times the vertical space of the first. This level of precise fine-tuning is what truly elevates **ggarrange()** from a simple layout utility to a sophisticated compositional instrument for professional statistical reporting and analysis.

While **ggarrange()** primarily operates within a strict rectangular grid structure--meaning it cannot easily handle complex, non-rectangular arrangements as effortlessly as the **patchwork** package--its ability to achieve relative sizing using the **widths** and **heights** weights within that grid is invaluable. For most general-purpose layouts, specifying these relative sizes is key to ensuring that plots containing denser information, more complex details, or longer axis labels receive the visual priority required for maximum effectiveness. Mastering the strategic use of **widths** and **heights** represents the final step in achieving complete control over the visual presentation when compiling multi-panel figures using the **egg** package.

Conclusion and Best Practices

The **ggarrange()** function provides an exceptionally robust, intuitive, and user-friendly interface for seamlessly combining disparate **ggplot2** objects into coherent, multi-panel visualizations within the **R** environment. By significantly simplifying the often-tedious process of defining and managing grid structures, it allows researchers and analysts to dedicate their focus to the analytical narrative and data interpretation rather than cumbersome graphical adjustments. Whether the specific requirement is a simple vertical stack, a comparative horizontal display, or a complex, size-weighted layout, the parameters **nrow**, **ncol**, **widths**, and **heights** collectively offer all the necessary tools for generating professional-grade statistical output.

When implementing **ggarrange()**, the standard best practice is to ensure that all individual plots are fully finalized--including their titles, axis labels, and chosen themes--before attempting the arrangement process. While **ggarrange()** skillfully handles alignment automatically, inconsistencies in elements like font sizes or color palettes across the individual plots will be magnified in the final combined view, potentially distracting the audience. Furthermore, for highly intricate visualization requirements involving shared legends, complex non-rectangular structures, or embedding tables within plots, analysts might benefit from exploring alternatives such as the powerful **patchwork** package. However, for standard, grid-based layouts, **ggarrange()** frequently remains the most straightforward and efficient choice due to its clear, explicit parameter definitions.

Ultimately, the capacity to quickly and accurately produce compelling, multi-faceted visualizations

is a core cornerstone of effective data communication. The **egg** package and its central function, **ggarrange()**, represent an indispensable component within the modern **R** data science toolkit, expertly streamlining the critical transition from isolated plots to comprehensive, publication-quality figures. We strongly encourage users to experiment freely with the **nrow**, **ncol**, and sizing arguments to discover the optimal visual presentation that best supports their specific dataset and analytical objective.

Additional Resources

The following tutorials explain how to perform other common tasks in **ggplot2**: