How to plot a table in R (With Example)

Authored by

stats writer

November 21, 2025

Plotting a table directly alongside a visual representation in the R programming language is a powerful technique for generating comprehensive data summaries. This process involves more than just the basic plotting functions; it requires careful management of graphic objects and layout. To successfully combine tabular data with charts, one must first ensure the data is structured correctly--typically as a data frame--and then leverage specialized packages designed for graphical arrangement. While the simple `plot()` command is useful for basic visualizations, creating a complex layout that integrates a raw data table alongside, for instance, a scatterplot, demands more advanced tools. For example, simply running `plot(df, type="p")` produces a visualization, but displaying the underlying data table requires converting that data into a graphic object that can be merged with the plot object.

In sophisticated data analysis and reporting workflows, the need to present the underlying numerical data alongside its graphical representation is constant. This combination ensures maximum transparency and allows readers to quickly verify the visualization against the raw figures.

Fortunately, achieving this composite output in R is highly manageable through the utilization of functions provided by the **gridExtra** package, which specializes in combining multiple graphic elements based on the grid graphics system.

The subsequent sections provide a comprehensive, step-by-step example demonstrating exactly how to employ the functions within this package to effectively plot a data table directly beneath or next to a chart, maximizing both the clarity and completeness of your reports.

Understanding the Need for Composite Data Presentation

In analytical reports, the visual summary provided by a chart, such as a bar chart or a scatterplot, captures trends and patterns quickly. However, relying solely on visualization often sacrifices the precision of the raw data points. For high-stakes or detailed documentation, stakeholders frequently require access to the exact numerical values that contributed to the graph. Combining a clear visualization with an adjacent data table satisfies both requirements simultaneously: providing immediate insight via the chart and full detail via the table. This holistic approach significantly enhances the credibility and completeness of the data presentation, moving beyond simple visualization towards comprehensive data reporting.

The standard plotting functions in R, while powerful for generating individual graphics, are not inherently designed for complex multi-panel layouts where one panel is a graphic visualization and another is formatted textual output (like a table). This limitation necessitates the use of specialized extension packages. The base R graphics system, or even popular packages like **lattice**, require significant manual effort to coordinate object placement. This is where the **grid** system, and

packages built upon it--like **gridExtra** and **ggplot2**--become indispensable tools for achieving pixel-perfect arrangement of diverse graphic elements, including formalized data tables.

Prerequisites: Essential R Packages for Advanced Layouts

To successfully execute the techniques demonstrated in this guide, two primary packages must be installed and loaded into the R environment: **ggplot2** and **gridExtra**. The **ggplot2** package is the industry standard for declarative data visualization in R, renowned for its implementation of the Grammar of Graphics, which simplifies the creation of sophisticated statistical plots. It provides the foundation for generating the primary visualization component (in our case, a scatterplot).

The **gridExtra** package, on the other hand, is built on top of R's low-level graphics engine, known as the **grid** system. The core purpose of **gridExtra** is to facilitate the arrangement of multiple **grid** grobs (graphic objects) onto a single output page. Crucially, it contains the functions necessary not only to combine existing plots but also to convert raw data objects, like data frames, into grobs that can be manipulated and positioned alongside visualizations. Without **gridExtra**, the task of marrying a data table with a chart becomes exceedingly cumbersome and often requires manual coordinate mapping.

Step 1: Defining the Sample Dataset (The Data Frame)

Before any visualization or arrangement can occur, we must first establish the dataset we intend to analyze. In R, tabular data is most commonly stored and manipulated using a data frame. For our example, we will construct a simple data frame named `df` containing two numeric variables, `x` and `y`, representing seven observations. These values will serve as the input for both our scatterplot visualization and the raw data table display.

The following code block illustrates the creation of this sample data frame, followed by a command to display its contents in the console. This initial step ensures data integrity and confirms the structure before proceeding to the graphic generation phase. Defining the data structure clearly is the foundational step in any statistical analysis workflow in R.

Suppose we have the following data frame in R:

```
#create data frame
df <- data.frame(x=c(1, 2, 3, 4, 5, 6, 7),
y=c(3, 4, 4, 8, 6, 10, 14))

#view data frame
df
```

x y
1 1 3
2 2 4
3 3 4
4 4 8
5 5 6
6 6 10
7 7 14

This data frame provides the necessary structure, with paired X and Y coordinates that are ideal for demonstrating the functionality of a scatterplot. Note the structure involves column names (x and y) and seven rows of observations, which will be the basis of both the visual and tabular output we intend to create.

Step 2: Generating the Plot and Table Graphic Objects

The ultimate goal is to create a scatterplot to visually assess the relationship between the `x` and `y` variables, and simultaneously plot a table containing the exact raw values. To merge these two distinct elements--a traditional chart and a data table--into a single output, we must first convert both into specialized graphic objects (grobs) recognizable by the **grid** system. The **ggplot2** package handles the chart creation, and the **gridExtra** package provides the tool for the table conversion.

We begin by defining the scatterplot using the `ggplot()` function. This visualization object, saved as `my_plot`, utilizes the Aesthetics (`aes`) mapping to define `x` and `y`, and then employs `geom_point()` to render the data points. Subsequently, we employ the powerful `tableGrob()` function from the **gridExtra** package. This function takes the R data frame `df` and transforms it into a grid graphic object (`my_table`), ready for layout manipulation. This conversion step is critical, as the resulting table object is treated as any other plot or graphical element by the arrangement functions.

Step 3: Arranging Elements in a Vertical Layout using `grid.arrange()`

Once both the plot grob (`my_plot`) and the table grob (`my_table`) have been successfully generated, the final step involves arranging them onto a single viewing panel. This task is managed by the `grid.arrange()` function, which is the cornerstone of the **gridExtra** package for combining plots. By default, `grid.arrange()` attempts to organize the provided graphical elements in a logical, column-based structure, stacking them vertically if no specific layout parameters are provided.

The syntax below shows how to load the required libraries, define the two graphic objects, and then utilize `grid.arrange()` to combine the scatterplot (`my_plot`) and the tabular data (`my_table`), resulting in the table being positioned directly beneath the visualization. This vertical stacking is often the preferred method when the table serves as a reference appendix to the chart above it, ensuring the visual interpretation remains the primary focus while the raw data is immediately accessible below.

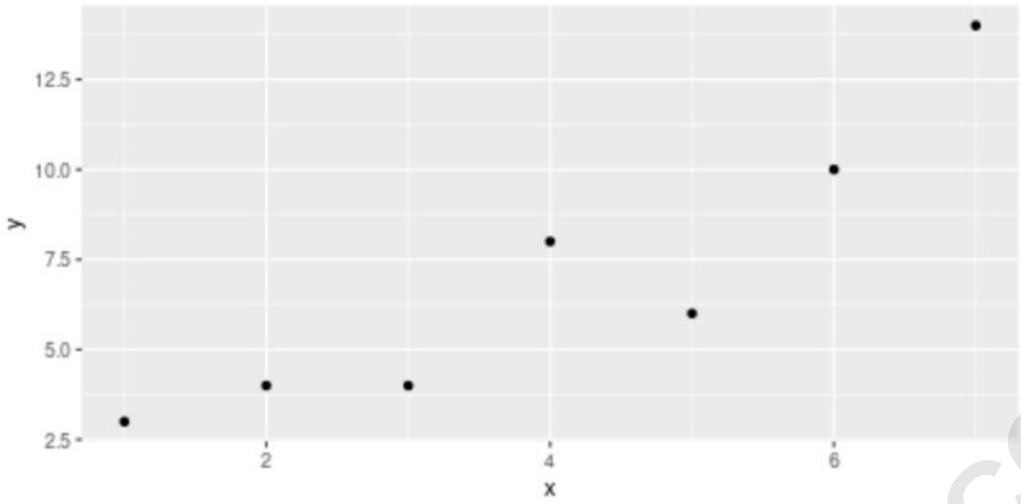
We can use the following syntax to define and arrange the elements:

```
library(gridExtra)
library(ggplot2)

#define scatterplot
my_plot <- ggplot(df, aes(x=x, y=y)) +
geom_point()

#define table
my_table <- tableGrob(df)

#create scatterplot and add table underneath it
grid.arrange(my_plot, my_table)
```



	x	y
1	1	3
2	2	4
3	3	4
4	4	8
5	5	6
6	6	10
7	7	14

Deconstructing the Vertical Arrangement Workflow

To ensure a thorough understanding of the preceding code, it is beneficial to analyze the specific role each function played in generating the composite output. This workflow utilizes key features of both the ggplot2 and gridExtra ecosystems, achieving a clean integration of different graphic types.

Here is how this code successfully produced the combined graphic:

We used **ggplot()** and **geom_point()** from the ggplot2 package to generate the primary visualization, the scatterplot, and saved this graphical object as `my_plot`.
We utilized the **tableGrob()** function, a critical component of **gridExtra**, to convert the raw R data frame (`df`) into a formal table graphic object (grob), which was assigned to `my_table`.
Finally, the **grid.arrange()** function took both `my_plot` and `my_table` as inputs and managed the complex task of plotting and aligning them together on the output device, stacking them vertically by default.

It is important to recognize that, by default, the `grid.arrange()` function is optimized for

sequential flow, meaning it attempts to arrange the list of graphic objects in a single column, consuming the entire width of the plotting pane for each object. While this vertical arrangement is often suitable for reports where the plot precedes the data, advanced reporting may require a more compact, side-by-side display, which introduces the need for explicit layout control parameters.

Advanced Layout Control: Displaying Tables and Plots Side-by-Side

For scenarios where screen real estate is limited or when a more compact visual summary is required, arranging the scatterplot and the data table side-by-side (horizontally) is highly desirable. Although `grid.arrange()` defaults to a single column, we can explicitly specify the number of columns (`ncol`) to achieve a multi-column layout. When arranging exactly two objects, setting `ncol=2` forces the arrangement function to place the elements adjacent to each other.

To implement this change, we use the `ncol` argument within the `grid.arrange()` function, or more precisely, utilize `arrangeGrob()` nested within `grid.arrange()` to apply the layout specifications before final rendering. The `arrangeGrob()` function is similar to `grid.arrange()` but returns a grob object itself, which allows for greater compositional control. By defining `ncol=2`, we instruct the arrangement engine to utilize two columns for the elements provided (`my_plot` and `my_table`), resulting in the horizontal display.

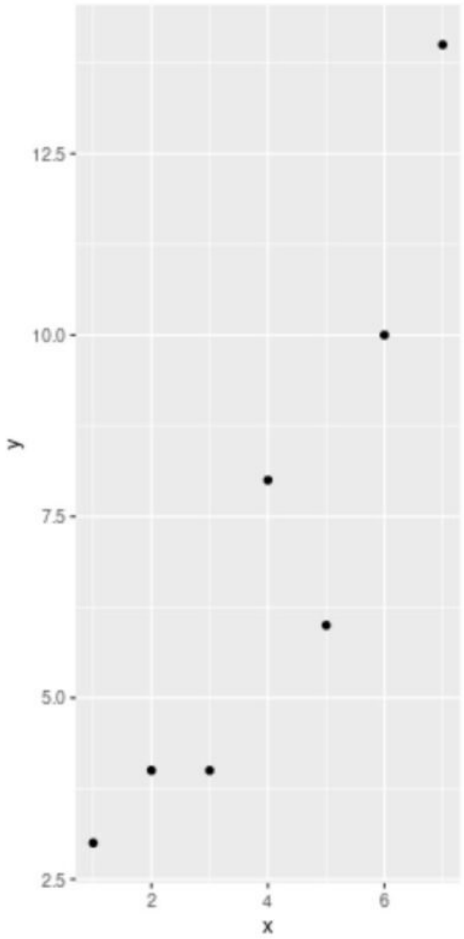
The revised code structure below demonstrates the implementation of the horizontal layout, ensuring that both the visualization and the raw data are visible simultaneously within a reduced vertical space. This technique is especially effective for dashboard creation or presentations where space efficiency is paramount:

```
library(gridExtra)
library(ggplot2)

#define scatterplot
my_plot <- ggplot(df, aes(x=x, y=y)) +
geom_point()

#define table
my_table <- tableGrob(df)

#create scatterplot and add table next to it
grid.arrange(arrangeGrob(my_plot, my_table, ncol=2))
```



	x	y
1	1	3
2	2	4
3	3	4
4	4	8
5	5	6
6	6	10
7	7	14

Following the execution of this revised code block, the graphic output confirms that the data table is now positioned immediately to the side of the scatterplot, achieving a more compact and horizontally aligned presentation compared to the default vertical stacking. This level of fine control over layout is what makes the **gridExtra** package an essential tool for high-quality, professional data reporting in R. The ability to precisely manage the arrangement of visualizations and accompanying tabular data greatly enhances the final output's aesthetic and informative value.

Conclusion: Enhancing Data Reporting Efficiency in R

The integration of raw data tables alongside their corresponding visualizations is a critical component of effective data communication. While generating plots in R is simple, combining disparate elements requires moving beyond base functions and embracing specialized packages. The workflow detailed here, leveraging **ggplot2** for visualization and **gridExtra** for arrangement and table generation, provides a robust, reproducible, and highly flexible methodology for creating composite graphics.

By understanding how to convert standard R objects like data frames into graphical objects using

`tableGrob()`, and subsequently controlling their layout using `arrangeGrob()` and `grid.arrange()`, R users gain significant control over their final output. Whether choosing a vertical layout for detailed reference or a horizontal layout for compact comparison, these tools ensure that data is presented accurately, clearly, and compellingly, cementing R's position as a premier environment for statistical reporting and analysis.