**R Visualization Exercises**

Table of Contents

[Example 1: Basic Barplot in R 2](#_Toc121503834)

[Example 2: Barplot with Color 3](#_Toc121503835)

[Example 3: Horizontal Barplot 3](#_Toc121503836)

[Example 4: Barplot with Labels 4](#_Toc121503837)

[Example 5: Stacked Barplot with Legend 6](#_Toc121503838)

[Example 6: Grouped Barplot with Legend 7](#_Toc121503839)

[Example 7: Barplot in ggplot2 Package 9](#_Toc121503840)

[Example 8: Barplot in plotly Package 10](#_Toc121503841)

[Example 1: Basic Box-and-Whisker Plot in R 14](#_Toc121503842)

[Example 2: Multiple Boxplots in Same Plot 15](#_Toc121503843)

[Example 3: Boxplot with User-Defined Title & Labels 16](#_Toc121503844)

[Example 4: Horizontal Boxplot 16](#_Toc121503845)

[Example 5: Add Notch to Box of Boxplot 17](#_Toc121503846)

[Example 6: Change Color of Boxplot 18](#_Toc121503847)

[Example 7: Specify Different Color for Each Boxplot 18](#_Toc121503848)

[Example 8: Add Space Between Boxplots of Different Groups 19](#_Toc121503849)

[Example 9: Boxplot in ggplot2 Package 20](#_Toc121503850)

[Example 1: Basic Kernel Density Plot in Base R 23](#_Toc121503851)

[Example 2: Modify Main Title & Axis Labels of Density Plot 24](#_Toc121503852)

[Example 3: Create Polygon Below Density Plot 24](#_Toc121503853)

[Example 4: Add Vertical Line for Mean to Density Plot 25](#_Toc121503854)

[Example 5: Histogram & Density in Same Plot 26](#_Toc121503855)

[Example 6: Multiple Densities in Same Plot 26](#_Toc121503856)

[Example 7: Add Legend to Density Plot 27](#_Toc121503857)

[Construction of Example Data 30](#_Toc121503858)

[Example 1: Create Heatmap with heatmap Function [Base R] 31](#_Toc121503859)

[Example 2: Create Heatmap with geom\_tile Function [ggplot2 Package] 33](#_Toc121503860)

[Example 3: Create Heatmap with plot\_ly Function [plotly Package] 35](#_Toc121503861)

[Example Data 42](#_Toc121503862)

[Example 1: Default Histogram in Base R 43](#_Toc121503863)

[Example 2: Histogram with Manual Main Title 43](#_Toc121503864)

[Example 3: Histogram with Colors 44](#_Toc121503865)

[Example 5: Histogram with Non-Uniform Width 45](#_Toc121503866)

[Example 6: Histogram with Manual Axis Limits 46](#_Toc121503867)

[Example 7: Histogram with Overlaid Density Line 47](#_Toc121503868)

[Example 8: Histogram with Values on Top of Bars 49](#_Toc121503869)

[Creating Example Data 53](#_Toc121503870)

[Example 1: Basic Creation of Line Graph in R 53](#_Toc121503871)

[Example 3: Change Color of Line 54](#_Toc121503872)

[Example 4: Modify Thickness of Line 55](#_Toc121503873)

[Example 5: Add Points to Line Graph 56](#_Toc121503874)

[Example 6: Plot Multiple Lines to One Graph 56](#_Toc121503875)

[Example 7: Different Point Symbol for Each Line 57](#_Toc121503876)

[Example 8: Line Graph in ggplot2 (geom\_line Function) 58](#_Toc121503877)

[Example 1: Basic Application of pairs() in R 62](#_Toc121503878)

[Example 2: Selecting Variables of pairs Plot 63](#_Toc121503879)

[Example 3: Modify Color, Shape of Points, Labels & Title 64](#_Toc121503880)

[Example 4: Modify pairs R Plot by Group 65](#_Toc121503881)

[Example 5: ggpairs R Function [ggplot2 & GGally] 66](#_Toc121503882)

[Example 1: Draw a Square Polygon in an R Plot 70](#_Toc121503883)

[Example 2: Colored Border of Polygon 71](#_Toc121503884)

[Example 3: Frequency Polygon in R 72](#_Toc121503885)

[Example 4: Draw Polygon Below Density 73](#_Toc121503886)

[Example 5: Density Polygon for Specific Range 74](#_Toc121503887)

[Example 6: Draw Polygon Between Two Densities 75](#_Toc121503888)

1. **Bar Charts**

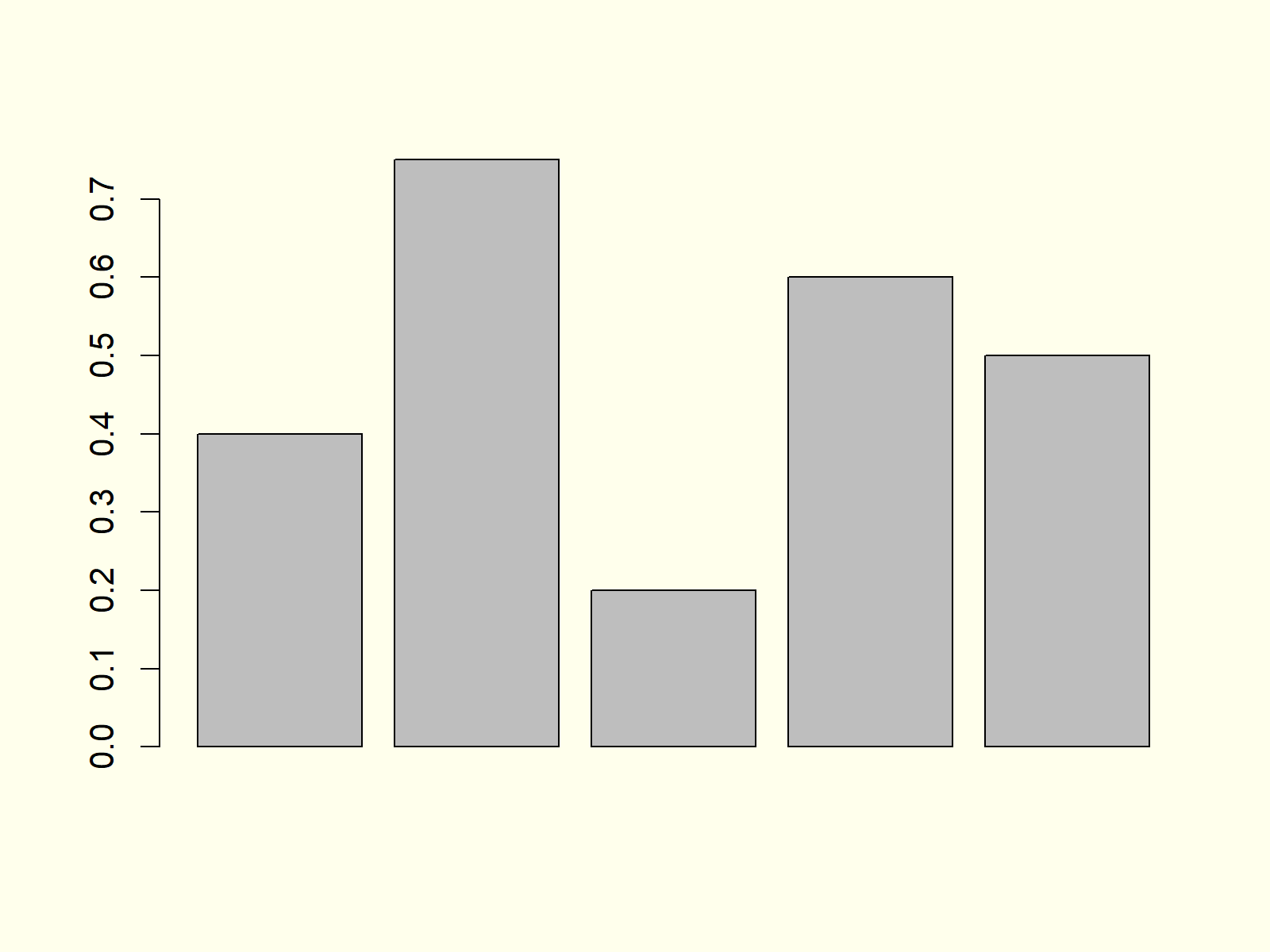
## Example 1: Basic Barplot in R

In Example 1, I’ll show you how to create a basic [barplot](https://stattrek.com/statistics/charts/histogram.aspx" \t "_blank) with the base installation of the R programming language. First, we need to create a vector containing the values of our bars:

|  |
| --- |
| values <- c(0.4, 0.75, 0.2, 0.6, 0.5) # Create values for barchart |

Now, we can use the barplot() function in R as follows:

|  |
| --- |
| barplot(values) # Basic barchart in R |



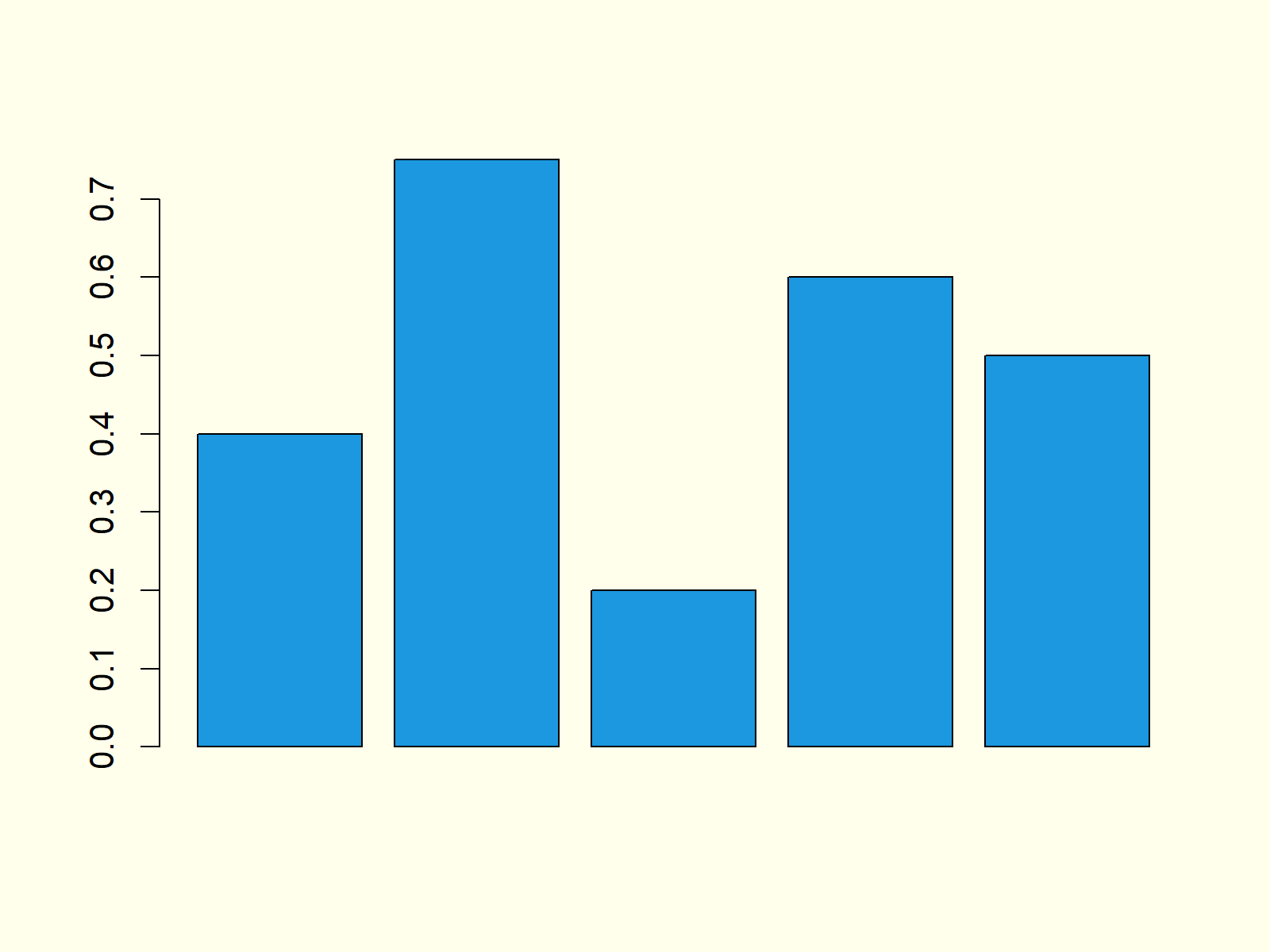
***Figure 1: Basic Barchart in R Programming Language.***

Figure 1 shows the output of the previous R code: A barchart with five bars. However, you can also see that our basic barchart is very plain and simple. In the next examples, I’ll show you how to modify this bargraph according to your specific needs. So keep on reading!

## Example 2: Barplot with Color

Example 2 shows how to add some color to the bars of our barplot with the col argument:

|  |
| --- |
| barplot(values, # Barchart with colored bars  col = "#1b98e0") |



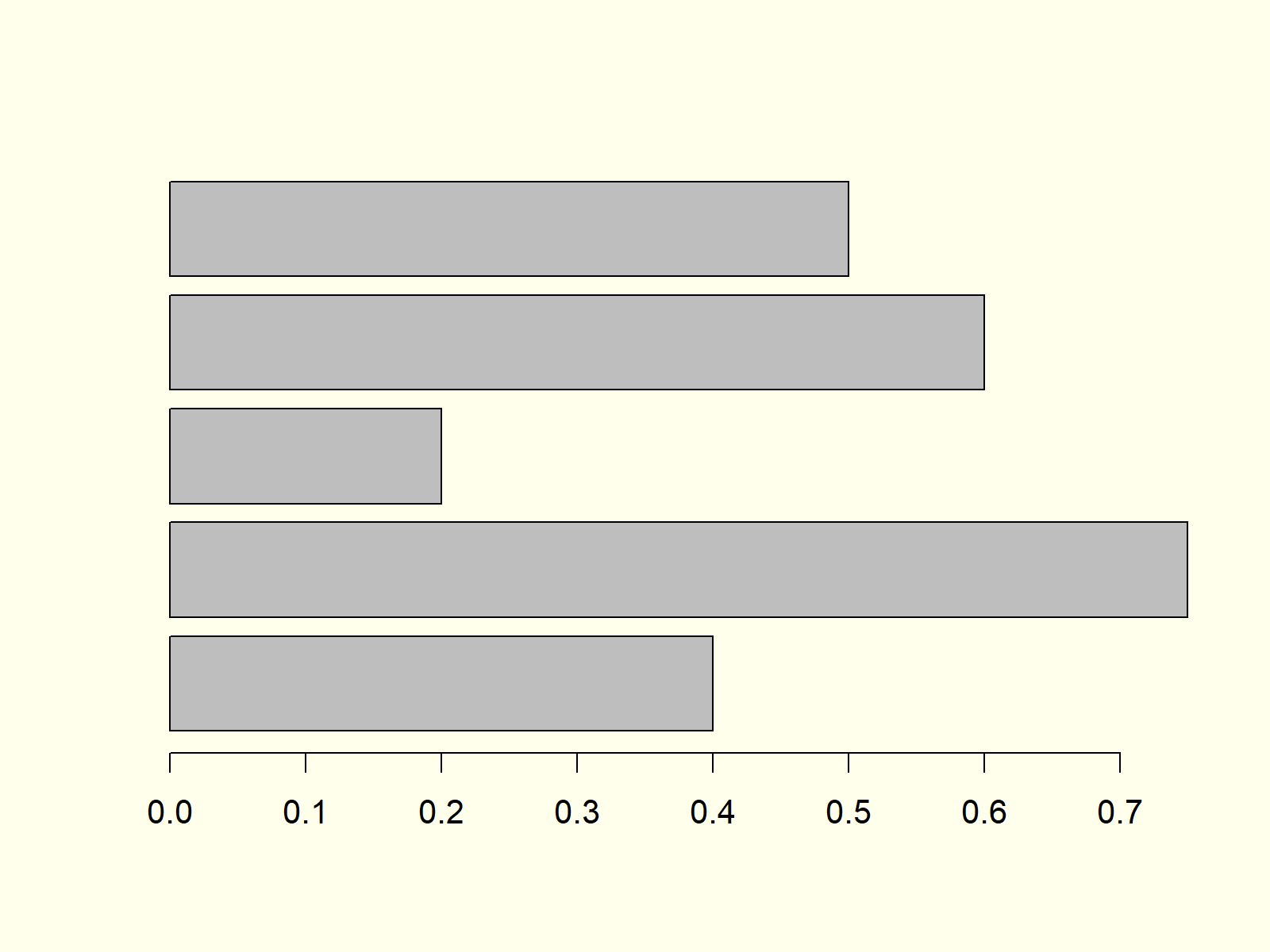
***Figure 2: Barchart with Colored Bars.***

Note that you could change the color of your bars to whatever color you want. Either you can use [HEX-Codes](https://htmlcolorcodes.com/) or you could use predefined color names. You can find an overview of [colors here](http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf).

## Example 3: Horizontal Barplot

We can align the bars of our bargrah horizontally by specifying the horiz option to be equal to TRUE:

|  |
| --- |
| barplot(values, # Horizontal barchart  horiz = **TRUE**) |



***Figure 3: Barchart with Horizontal Alignment.***

Figure 3 shows exactly the same bars and values as the previous examples, but this time with horizontal bars instead of vertical bars.

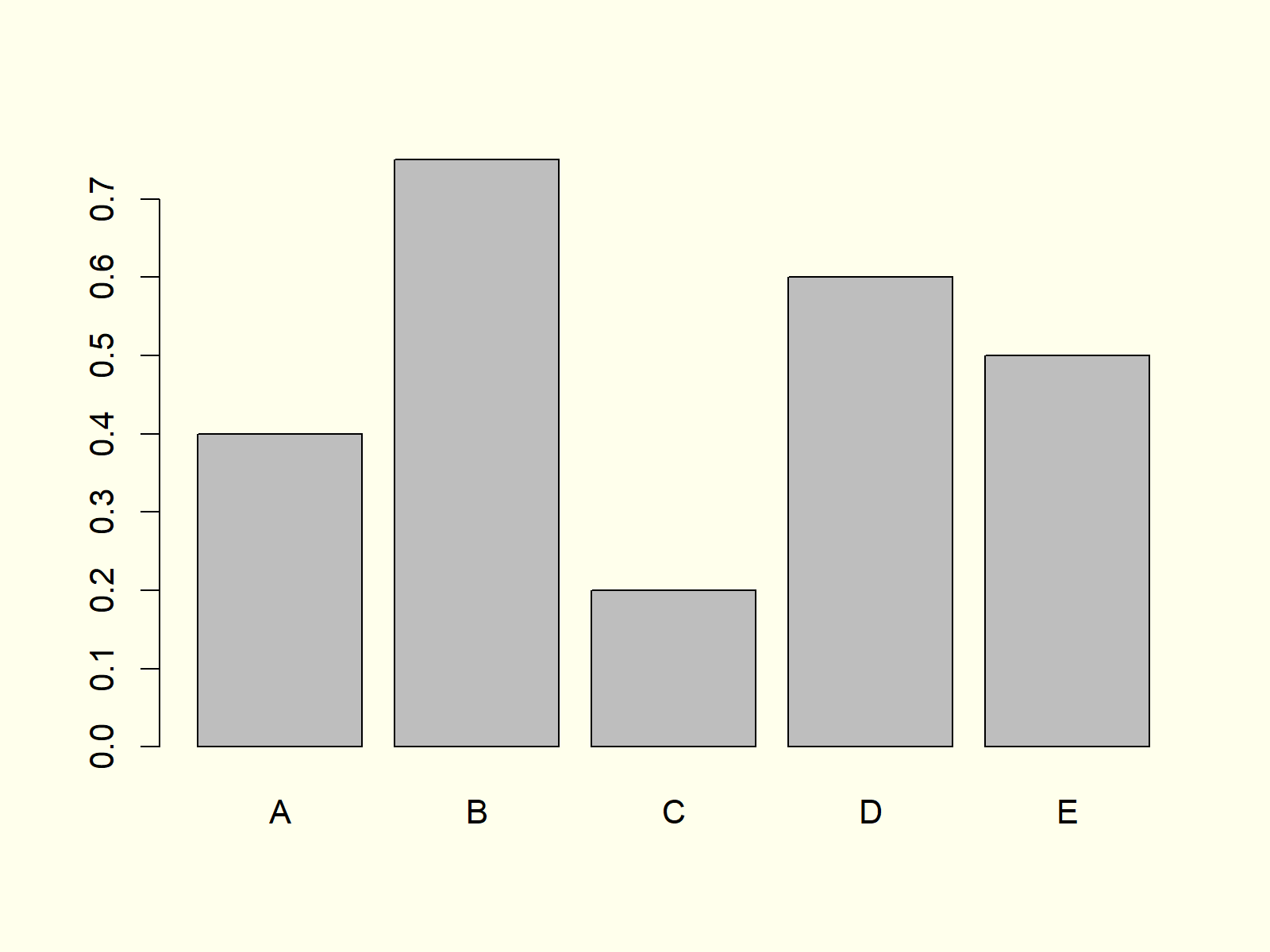
## Example 4: Barplot with Labels

It makes a lot of sense to add labels to our barchart in order to show the reader the meaning of each bar. First, we need to specify a vector consisting of the labels of our bars:

|  |
| --- |
| **group** <- LETTERS[1:5] # Create grouping variable |

Now, we can add these labels to our barplot with the names.arg option:

|  |
| --- |
| barplot(values, # Add labels to barplot  names.arg = **group**) |



***Figure 4: Barchart with Labels of Bars.***

Note that the vector containing our labels needs to have the same length and ordering as the vector containing our values.

## Example 5: Stacked Barplot with Legend

When we have data with several subgroups (e.g. male and female), it is often useful to plot a stacked barplot in R. For this task, we need to create some new example data:

|  |
| --- |
| data <- **as**.matrix(data.frame(A = c(0.2, 0.4), # Create matrix for stacked barchart  B = c(0.3, 0.1),  C = c(0.7, 0.1),  D = c(0.1, 0.2),  E = c(0.3, 0.3)))  rownames(data) <- c("Group 1", "Group 2")  data # Print matrix to console  # A B C D E  # Group 1 0.2 0.3 0.7 0.1 0.3  # Group 2 0.4 0.1 0.1 0.2 0.3 |

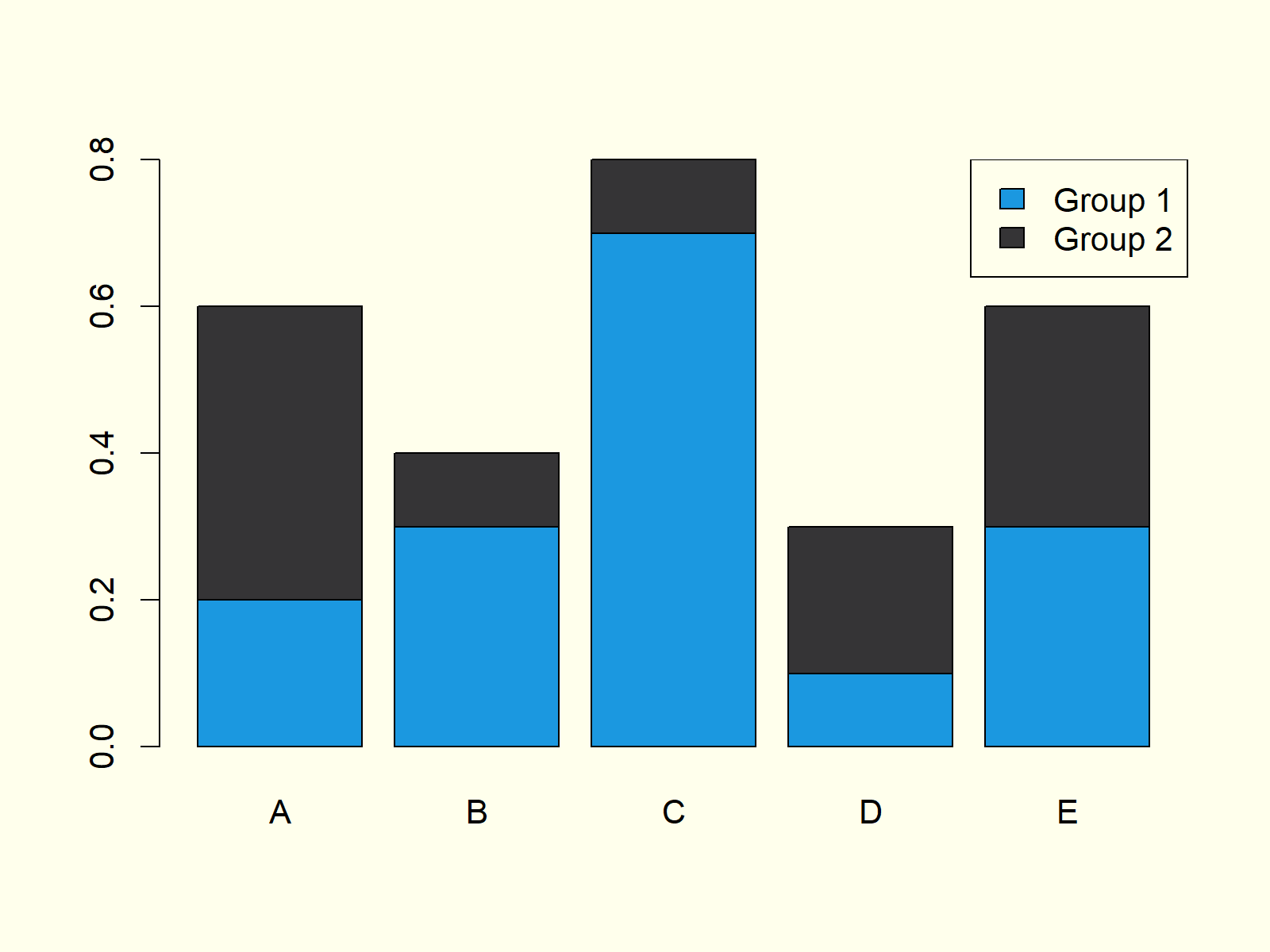
Based on the previous output of the RStudio console, you can see how our example data should look like: It’s a matrix consisting of a column for each bar and a row for each group.

Now, we can draw a stacked barchart by specifying our previously created matrix as input data for the barplot function:

|  |
| --- |
| barplot(data,  # Create stacked barchart  col = c("#1b98e0", "#353436")) |

Furthermore, we should [add a legend](https://statisticsglobe.com/add-legend-to-plot-in-base-r) to our stacked bargraph to illustrate the meaning of each color:

|  |
| --- |
| legend("topright", # Add legend to barplot  legend = c("Group 1", "Group 2"),  fill = c("#1b98e0", "#353436")) |



***Figure 5: Stacked Barchart with Legend in R.***

Figure 5 shows our stacked bargraph. It contains five bars, whereby each of the bars consists of two subgroups. The legend on the topright illustrates the meaning of the two colors of each bar.

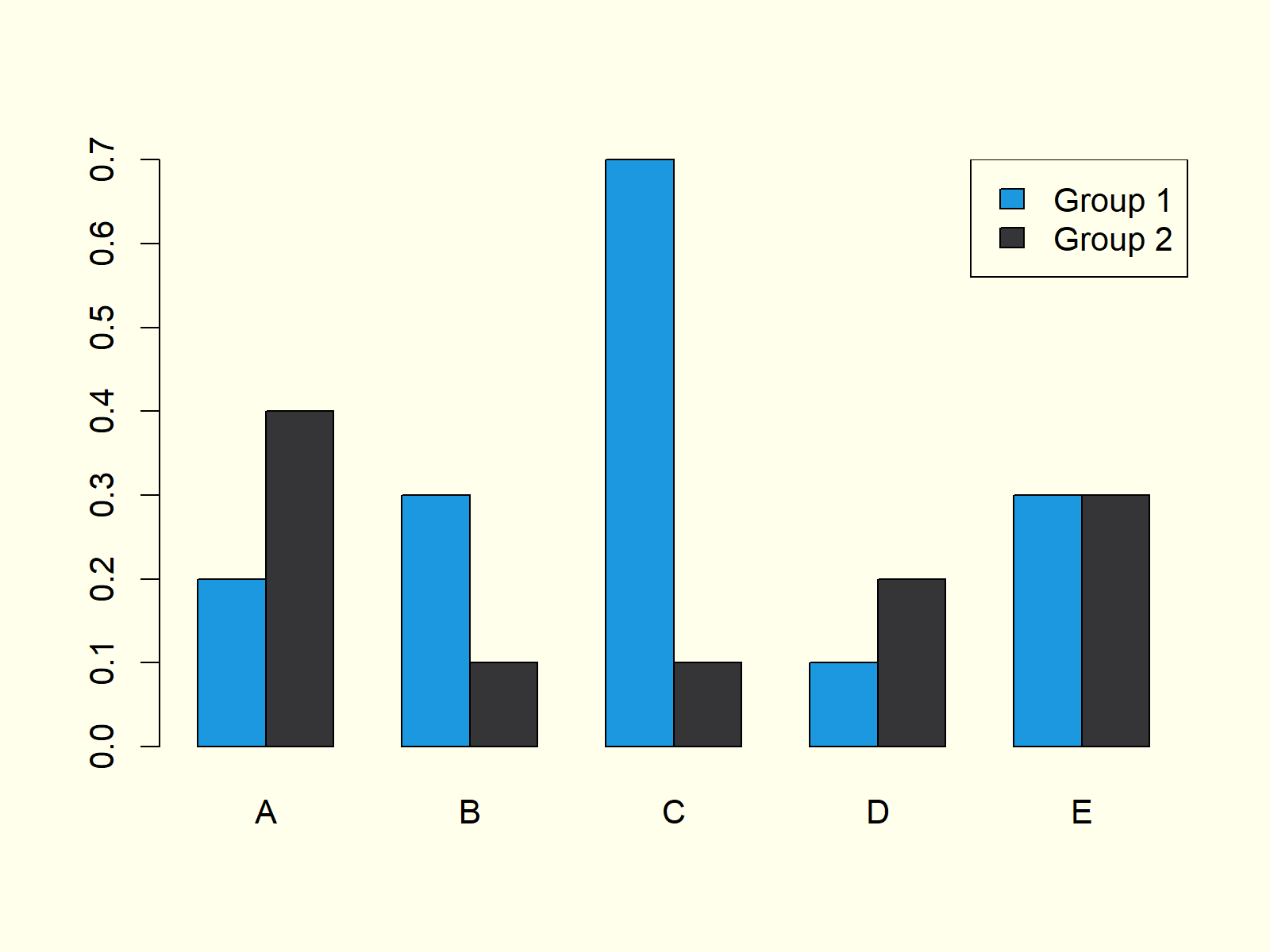
## Example 6: Grouped Barplot with Legend

Depending on our specific data situation it may be better to print a grouped barplot instead of a stacked barplot (as shown in Example 5). We can do that by specifying beside = TRUE within the barplot command:

|  |
| --- |
| barplot(data, # Create grouped barchart  col = c("#1b98e0", "#353436"),  beside = **TRUE**) |

As in Example 5, we should also print a legend to our barchart:

|  |
| --- |
| legend("topright", # Add legend to barplot  legend = c("Group 1", "Group 2"),  fill = c("#1b98e0", "#353436")) |



***Figure 6: Grouped Barchart with Legend in R.***

Compare Figure 5 and Figure 6. Both [graphics](https://statisticsglobe.com/graphics-in-r) contain the same values, once in a stacked barchart and once in a grouped barchart.

## Example 7: Barplot in ggplot2 Package

So far, we have created all barplots with the base installation of the R programming language. However, there are multiple packages available that also provide functions for the drawing of barcharts.

In this example you’ll learn how to make a basic Barplot with the [ggplot2 package](https://cran.r-project.org/web/packages/ggplot2/index.html). First, we need to install and load the package:

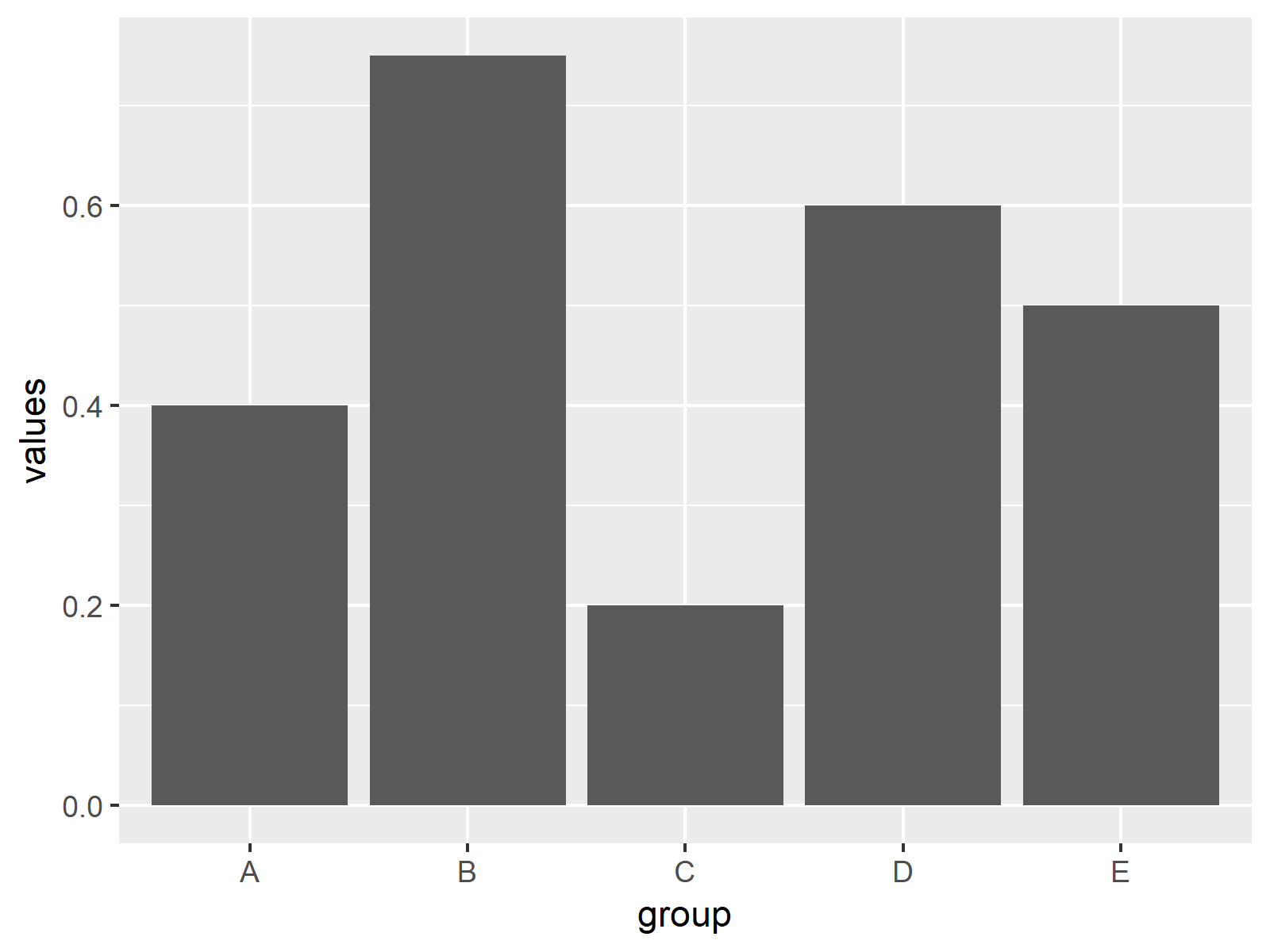
|  |
| --- |
| install.packages("ggplot2") # Install ggplot2 package  library("ggplot2") # Load ggplot2 package |

Then, we also need to store our group and value variables in a data frame – That is required by the functions of the ggplot2 package:

|  |
| --- |
| data\_ggp <- data.frame(**group**, values) # Create data frame for ggplot2 |

Now, we can apply the ggplot and the geom\_bar functions of the ggplot2 package to create a barplot:

|  |
| --- |
| ggplot(data\_ggp, aes(x = **group**, y = values)) + # Create barchart with ggplot2  geom\_bar(stat = "identity") |



***Figure 7: Barchart Created with ggplot2 Package.***

Figure 7 shows bars with the same values as in Examples 1-4. However, this time the bargraph is shown in the typical ggplot2 design.

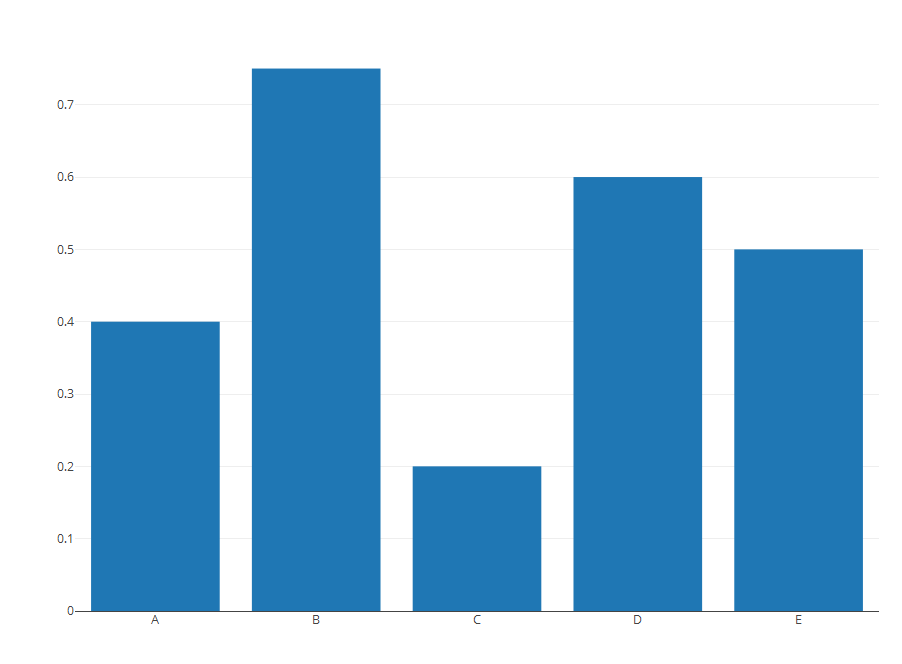
## Example 8: Barplot in plotly Package

Another powerful R add-on package for the printing of barcharts is the [plotly package](https://plot.ly/r/" \t "_blank). Let’s install and load the package to R:

|  |
| --- |
| install.packages("plotly") # Install plotly package  library("plotly") # Load plotly package |

The plotly package contains the plot\_ly function. We can use this function to make a barchart as follows:

|  |
| --- |
| plot\_ly(x = **group**, # Create barchart with plotly  y = values,  type = "bar") |



***Figure 8: Barchart Created with plotly Package.***

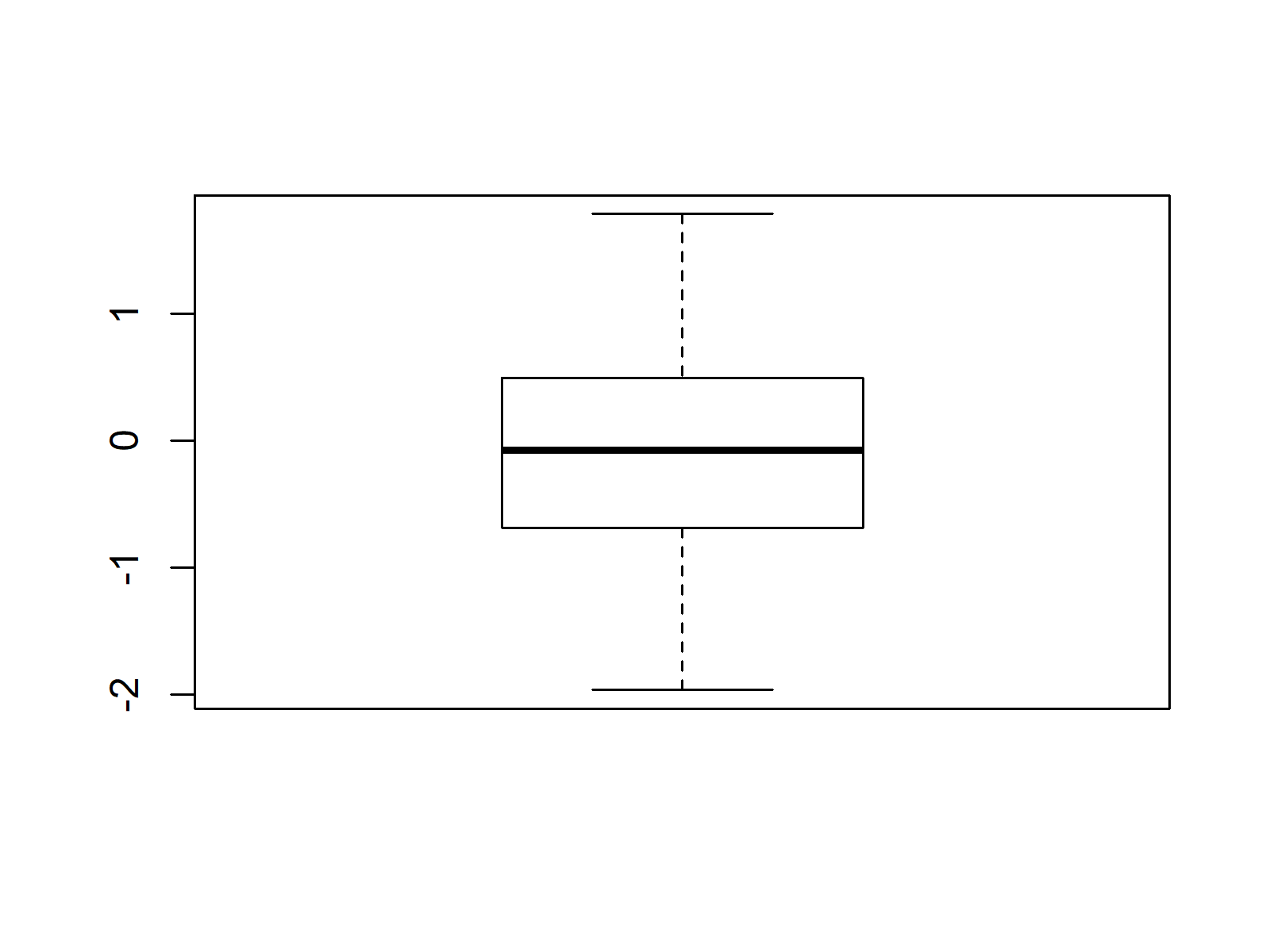
Figure 8 also visualizes the same values as Examples 1-5 and 7, but this time in plotly style.

1. **Box Plot**

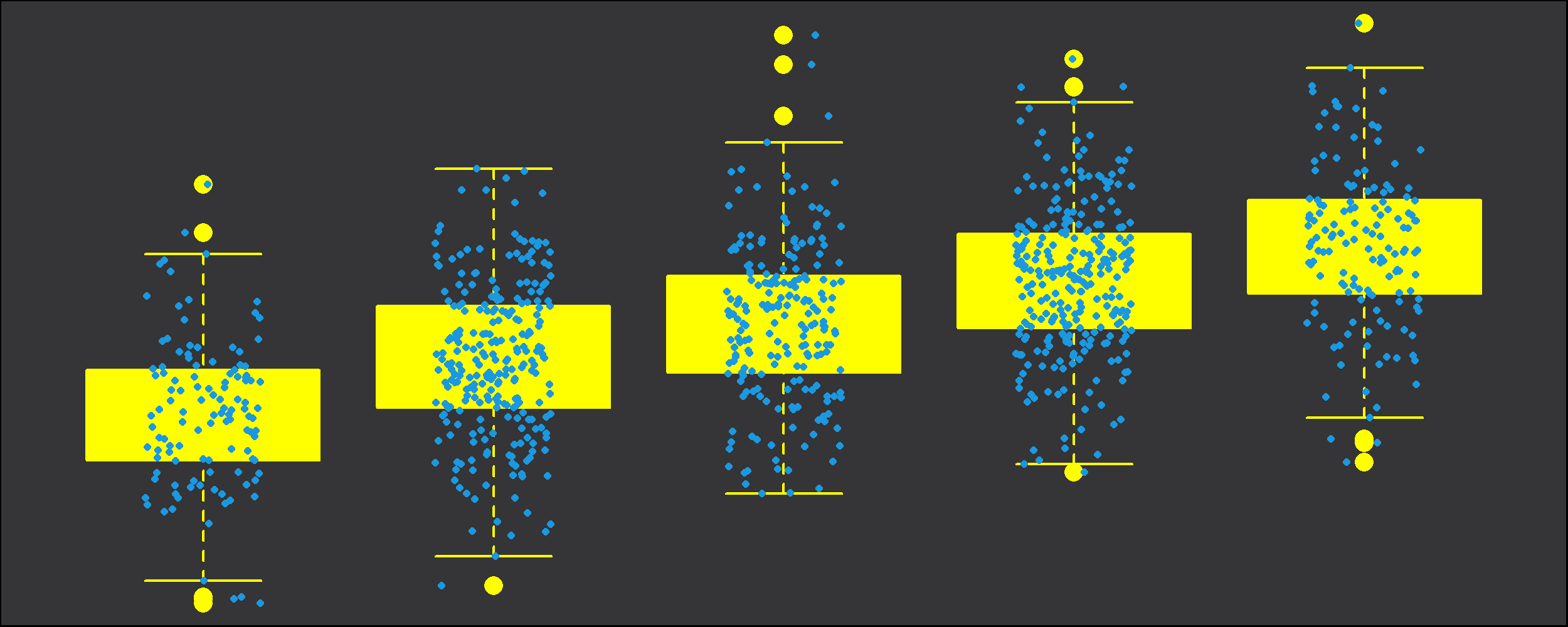
**Boxplot Definition:** A boxplot (or box-and-whisker plot) displays the distribution of a numerical variable based on five summary statistics: minimum non-outlier; first quartile; median; third quartile; and maximum non-outlier. Furthermore, boxplots show the positioning of outliers and whether the data is skewed.

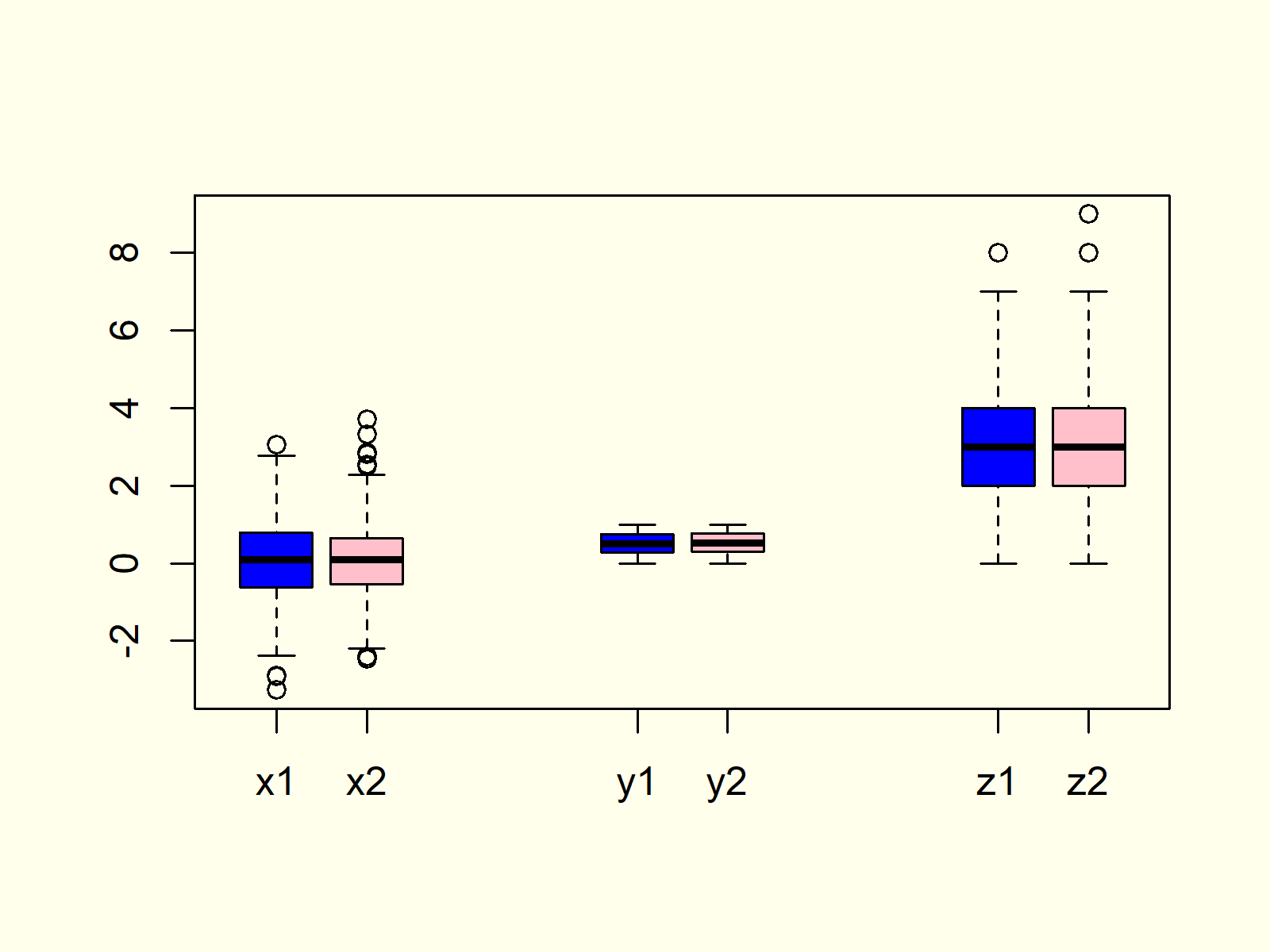
The following R syntax shows how to draw a basic boxplot in R:

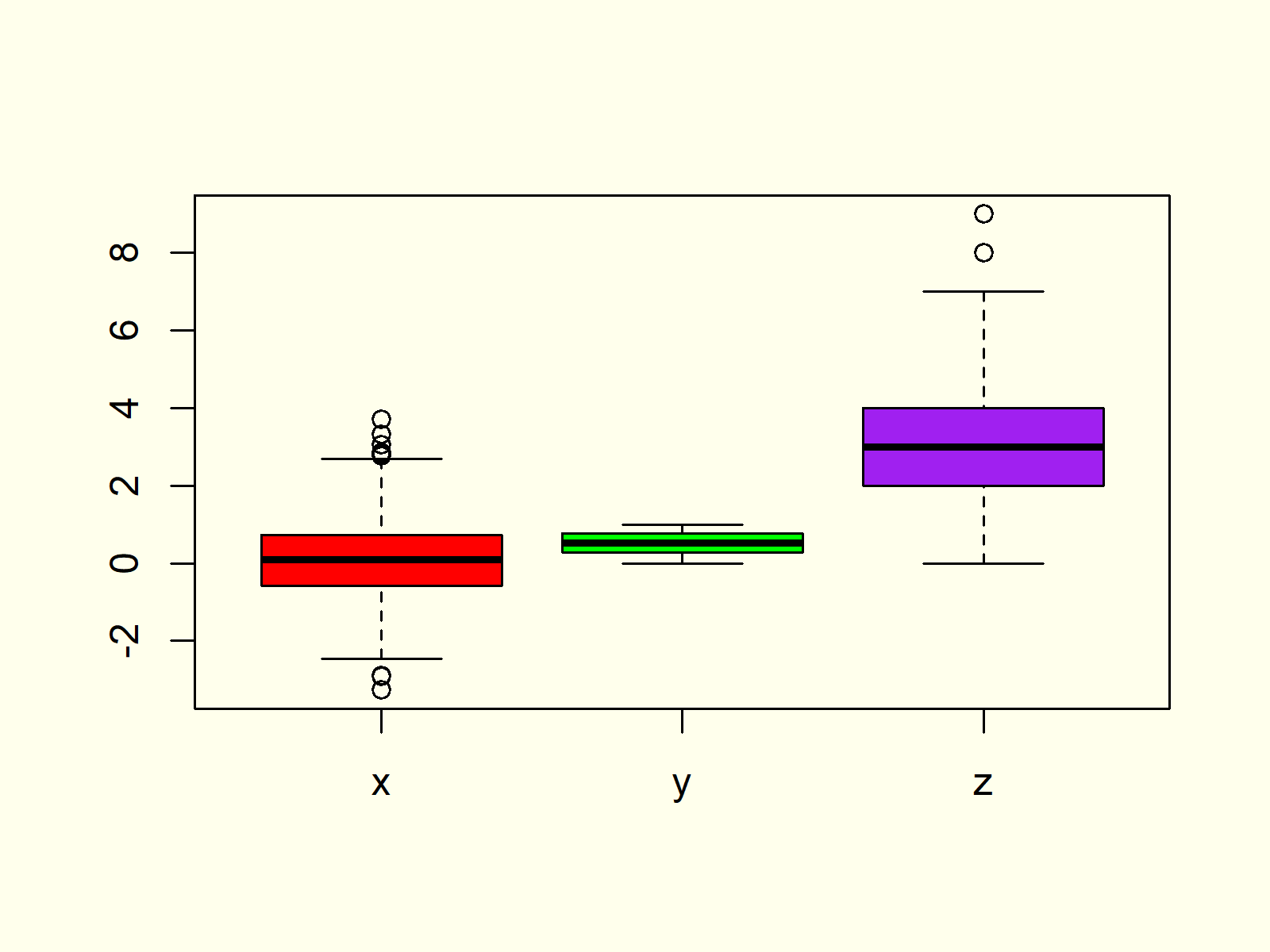
boxplot(x)

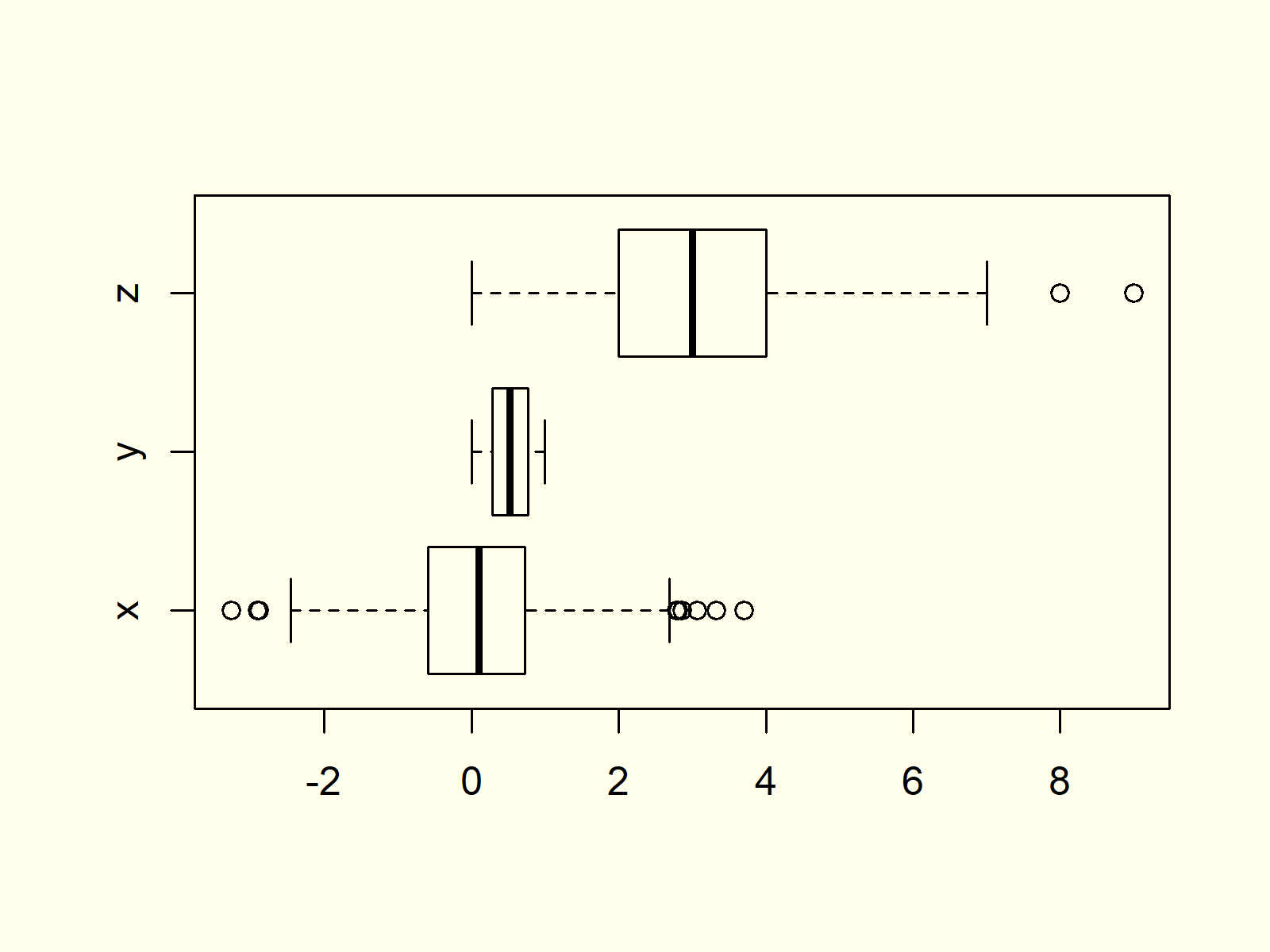


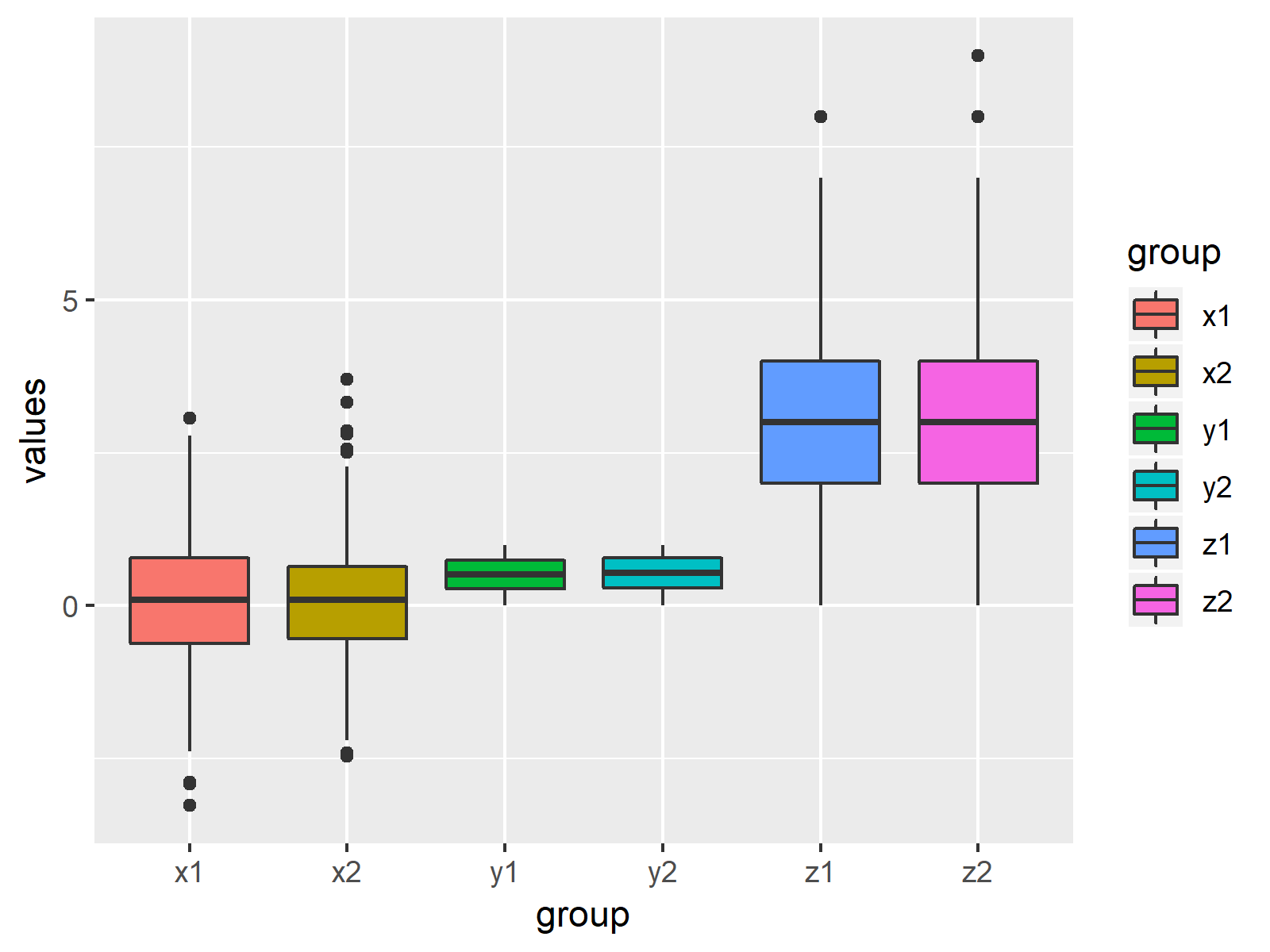
**Advanced Boxplots:** Find some advanced boxplots below. Click on the images to get more information and example R codes for each of the boxplots.











## Example 1: Basic Box-and-Whisker Plot in R

[Boxplots](https://en.wikipedia.org/wiki/Box_plot) are a popular [type of graphic](https://statisticsglobe.com/graphics-in-r) that visualize the minimum non-outlier, the first quartile, the median, the third quartile, and the maximum non-outlier of [numeric data](https://www.dummies.com/education/math/statistics/types-of-statistical-data-numerical-categorical-and-ordinal/) in a single plot.

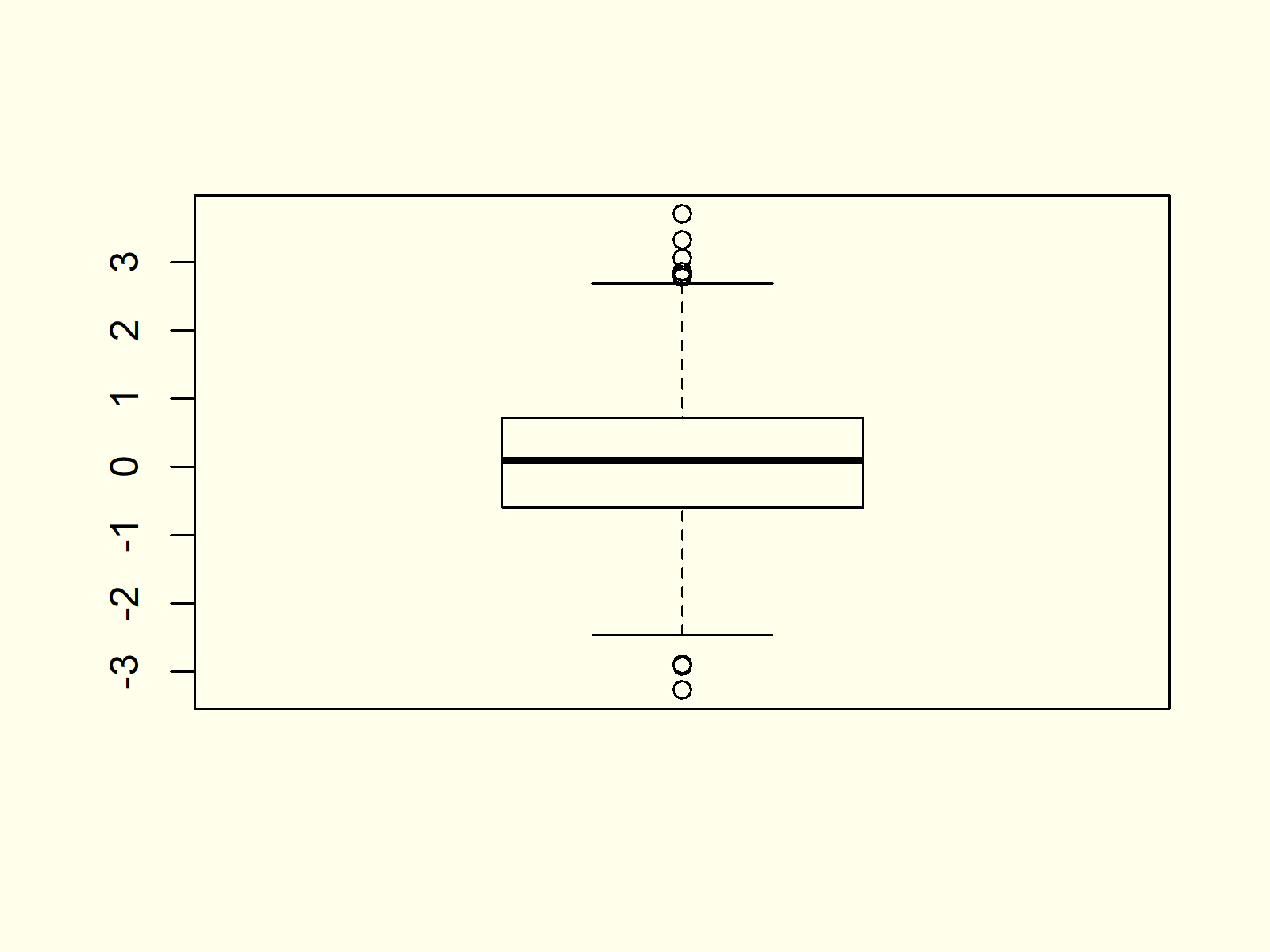
Let’s create some numeric example data in R and see how this looks in practice:

|  |
| --- |
| **set**.seed(8642) # Create random data  x <- rnorm(1000) |

Our example data is a random numeric vector [following the normal distribution](https://statisticsglobe.com/normal-distribution-in-r-dnorm-pnorm-qnorm-rnorm). The data is stored in the data object x.

We can now plot these data with the boxplot() function of the base installation of R:

|  |
| --- |
| boxplot(x) # Basic boxplot in R |



***Figure 1: Basic Boxplot in R.***

Figure 1 visualizes the output of the boxplot command: A box-and-whisker plot. As you can see, this boxplot is relatively simple. In the following examples I’ll show you how to modify the different parameters of such boxplots in the R programming language.

## Example 2: Multiple Boxplots in Same Plot

In Example 2 you’ll learn [how to draw a graph containing multiple boxplots side by side](https://statisticsglobe.com/draw-multiple-boxplots-in-one-graph-in-r) in R. First, we need to create some more data that we can plot in our graphic.

The following R code creates a [uniformly distributed](https://statisticsglobe.com/uniform-distribution-in-r-dunif-punif-qunif-runif) variable y and a [poisson distributed](https://statisticsglobe.com/poisson-distribution-in-r-dpois-ppois-qpois-rpois) variable z:

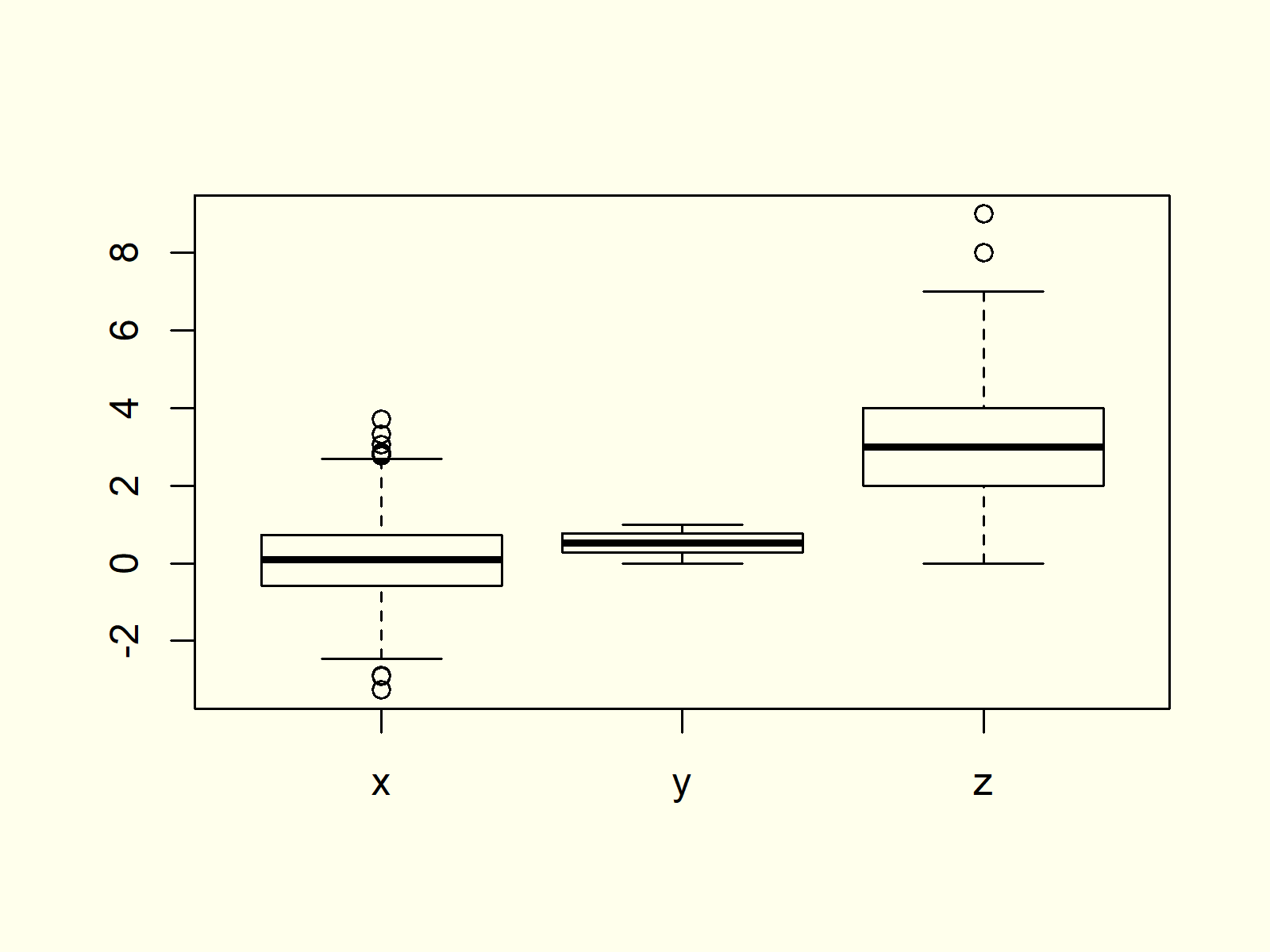
|  |
| --- |
| y <- runif(1000) # Create more variables  z <- rpois(1000, 3) |

Now, we can store our three variables x, y, and z in a data frame:

|  |
| --- |
| data <- data.frame(values = c(x, y, z), # Combine variables in data frame  **group** = c(rep("x", 1000),  rep("y", 1000),  rep("z", 1000)))  head(data) # First six rows of data  # values group  # -0.8035458 x  # 0.6384819 x  # -0.1417869 x  # 2.1542073 x  # -0.1220888 x  # -0.7332229 x |

If we want to [create a graphic](https://statisticsglobe.com/graphics-in-r) with multiple boxplots, we have to specify a column containing our numeric values, the grouping column, and the data frame containing our data:

|  |
| --- |
| boxplot(values ~ **group**, data) # Multiple boxplots in same graph |



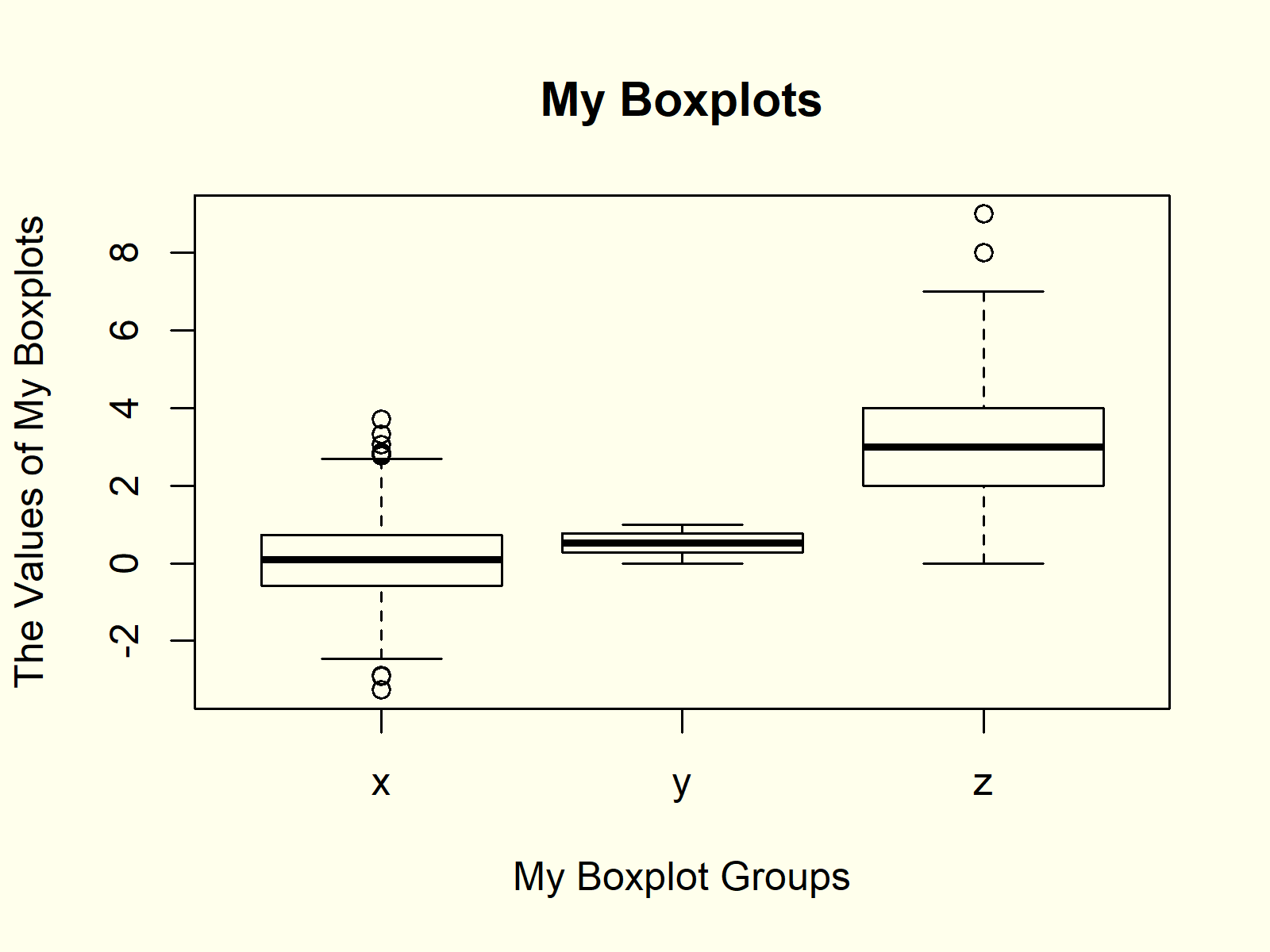
***Figure 2: Multiple Boxplots in Same Graphic.***

As you can see based on Figure 2, the previous R code created a graph with multiple boxplots.

## Example 3: Boxplot with User-Defined Title & Labels

The boxplot function also allows user-defined main titles and axis labels. If we want to add such text to our boxplot, we need to use the main, xlab, and ylab arguments:

|  |
| --- |
| boxplot(values ~ **group**, data, # Change main title and axis labels  main = "My Boxplots",  xlab = "My Boxplot Groups",  ylab = "The Values of My Boxplots") |

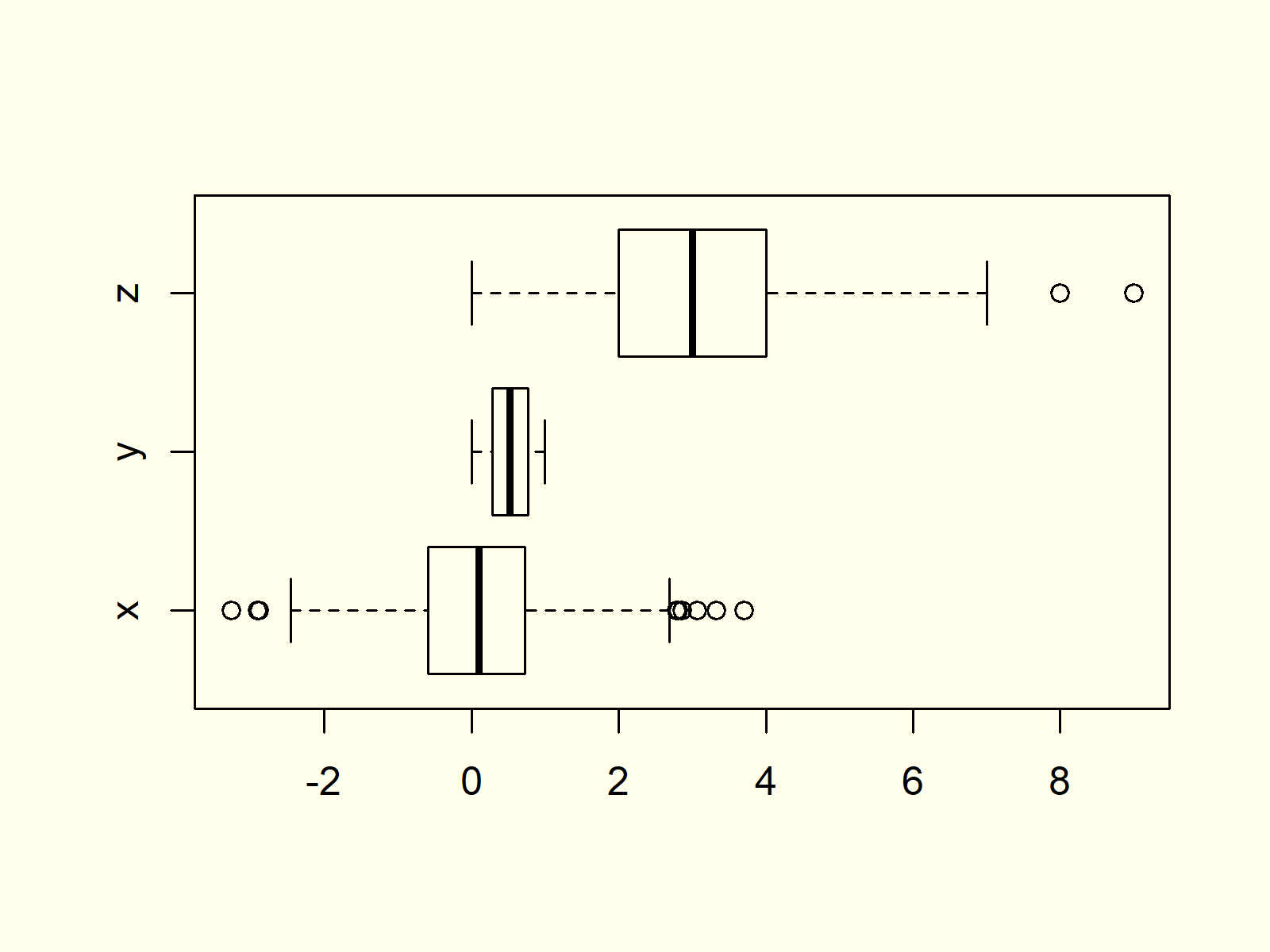


***Figure 3: Changed Main Title & Axis Labels.***

## Example 4: Horizontal Boxplot

We can align our boxplots horizontally with the argument horizontal = TRUE:

|  |
| --- |
| boxplot(values ~ **group**, data,  # Horizontal boxplots  horizontal = **TRUE**) |



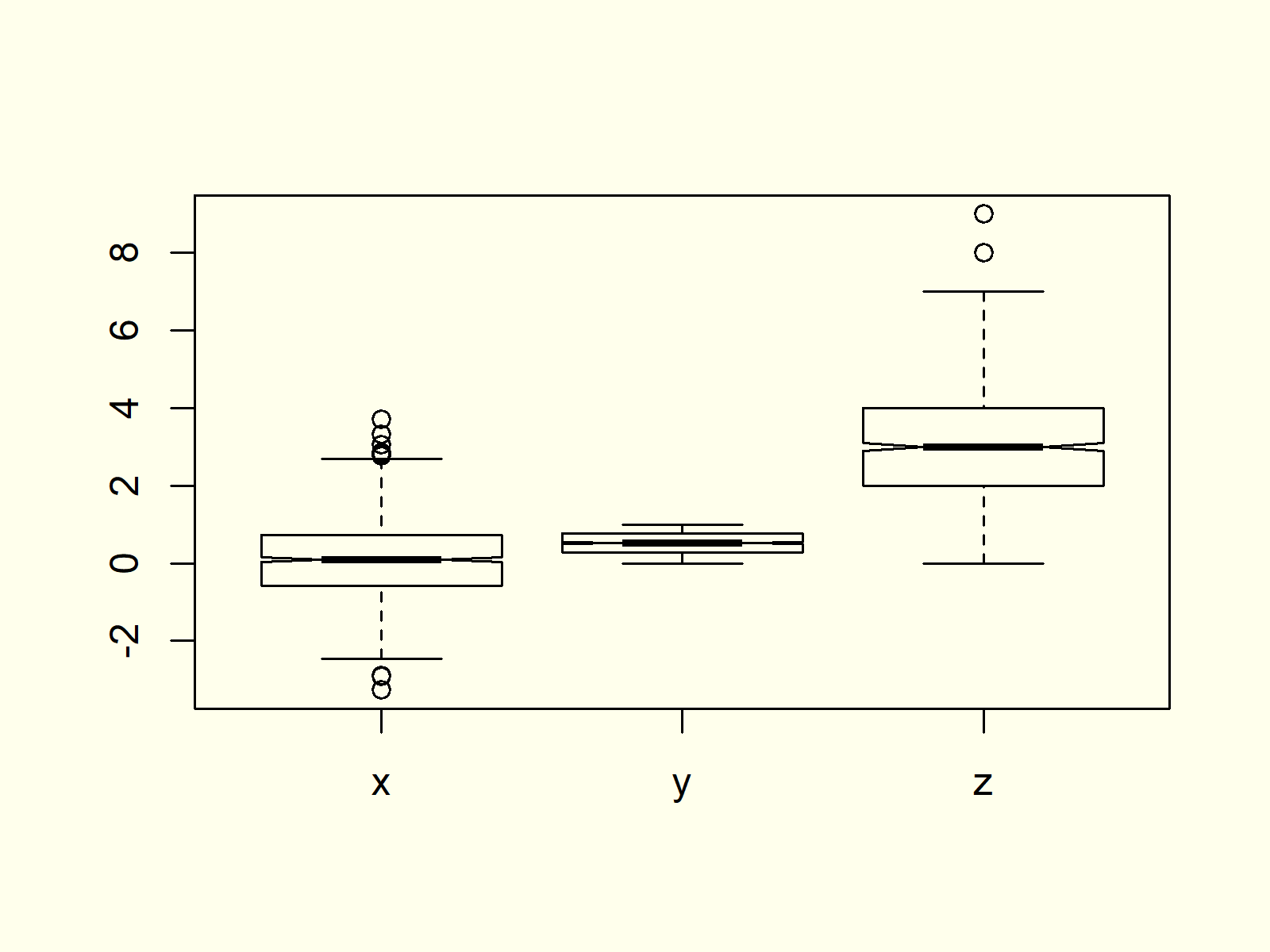
***Figure 4: Horizontally Aligned Boxplots.***

As you can see based on Figure 4, the previous R syntax changed the X- and Y-Axes of our plot.

## Example 5: Add Notch to Box of Boxplot

If we want to make the middle of our boxplots thinner, we can use the notch argument:

|  |
| --- |
| boxplot(values ~ **group**, data, # Thin boxplots  notch = **TRUE**) |

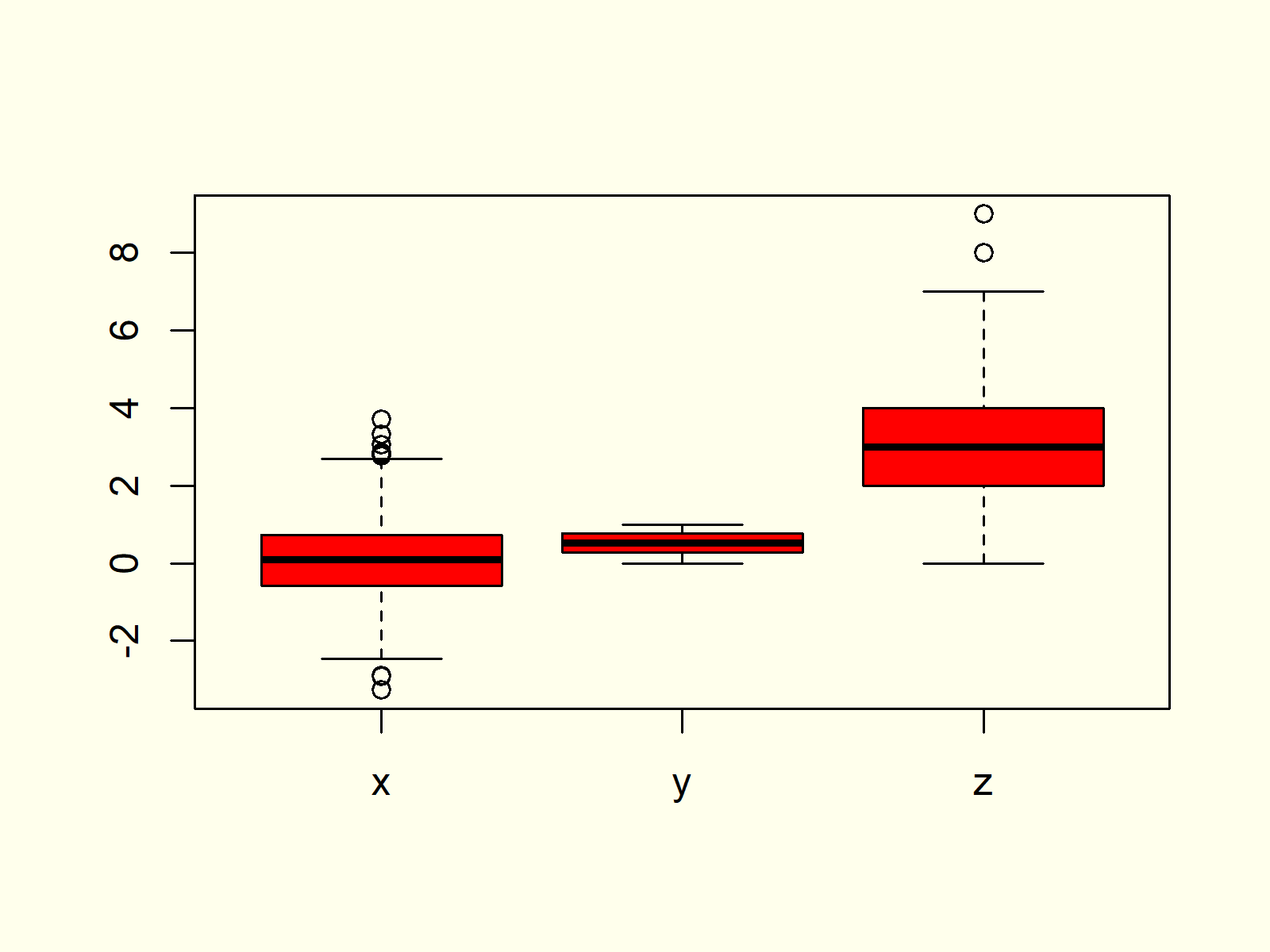


***Figure 5: Thinner Boxplots.***

## Example 6: Change Color of Boxplot

Another popular modification of boxplots is the filling color. If we want to change all our boxplots to the same color, we can specify the col argument to be equal to a single color:

|  |
| --- |
| boxplot(values ~ **group**, data, # Color of boxplots  col = "red") |

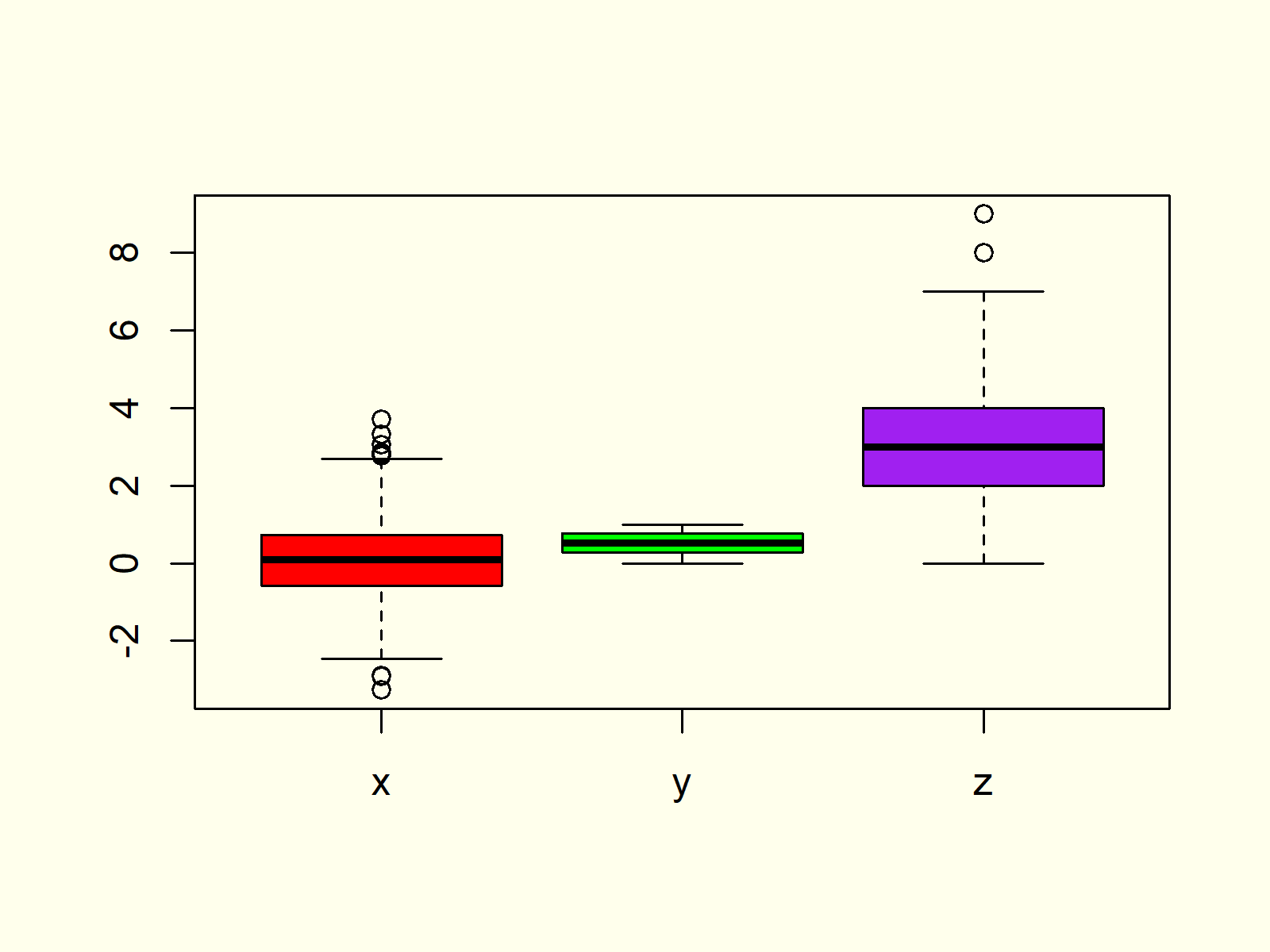


***Figure 6: Modify Color of All Boxplots.***

## Example 7: Specify Different Color for Each Boxplot

If we want to print each of our boxplots in a different color, we have to specify a vector of colors containing a color for each of our boxplots:

|  |
| --- |
| boxplot(values ~ **group**, data, # Different color for each boxplot  col = c("red", "green", "purple")) |



***Figure 7: Specify Separate Color for Each Boxplot.***

## Example 8: Add Space Between Boxplots of Different Groups

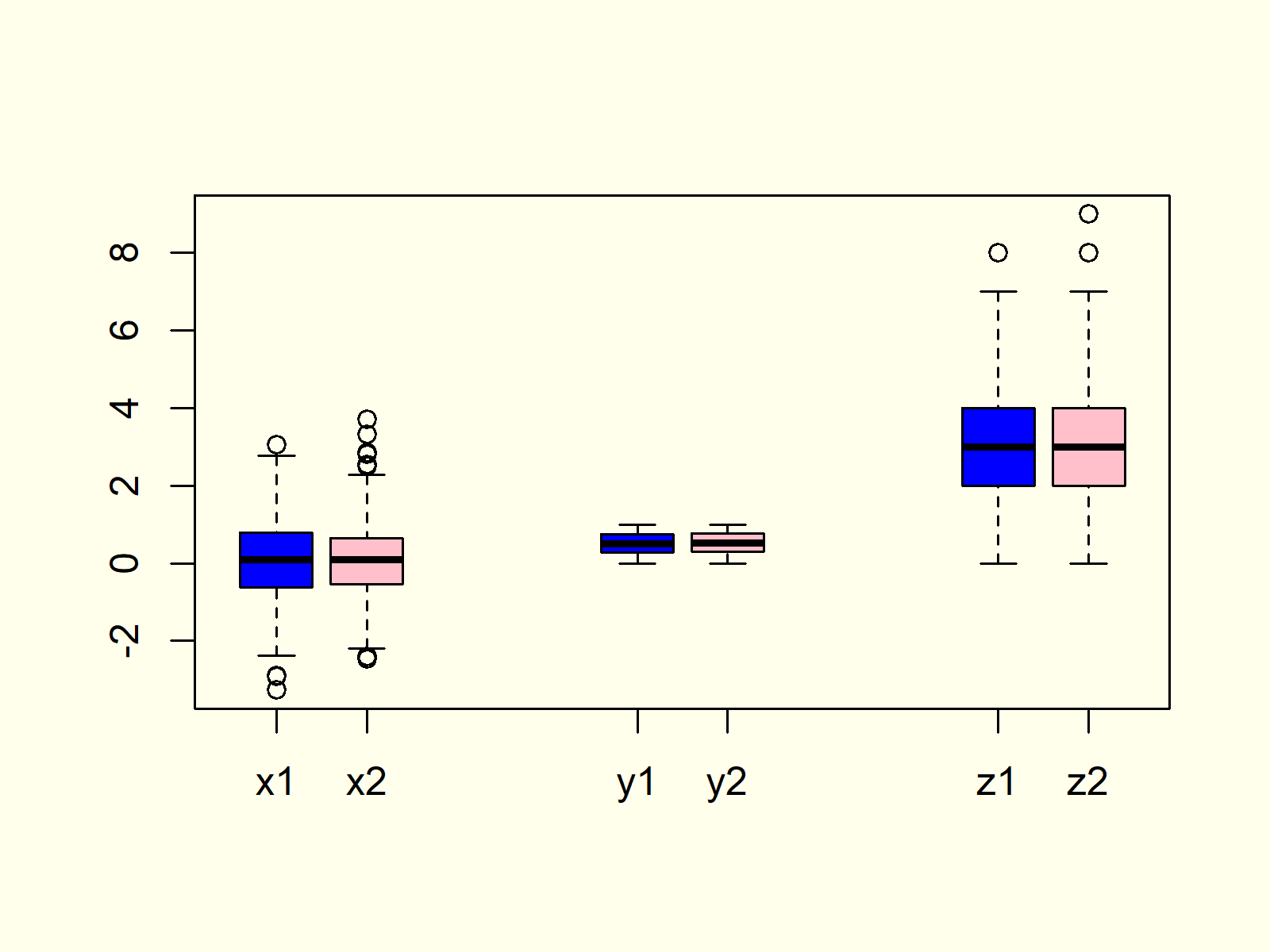
Often, we want to cluster our boxplots into different groups (e.g. male and female). In such a case it makes sense to add some additional spacing to our boxplot.

Let’s first modify our data so that each boxplot is divided into subgroups:

|  |
| --- |
| data2 <- data # Replicate data  data2$group <- c(rep("x1", 500), rep("x2", 500), # Modify group variable  rep("y1", 500), rep("y2", 500),  rep("z1", 500), rep("z2", 500)) |

Now, we can use the at option of the boxplot function to specify the exact positioning of each boxplot. Note that we are leaving out the positions 3, 4, 7, and 8:

|  |
| --- |
| boxplot(values ~ **group**, data2, # Boxplot with manual positions  col = c("blue", "pink"),  at = c(1, 2, 5, 6, 9, 10)) |



***Figure 8: Change Spacing/Positioning of Boxplots.***

## Example 9: Boxplot in ggplot2 Package

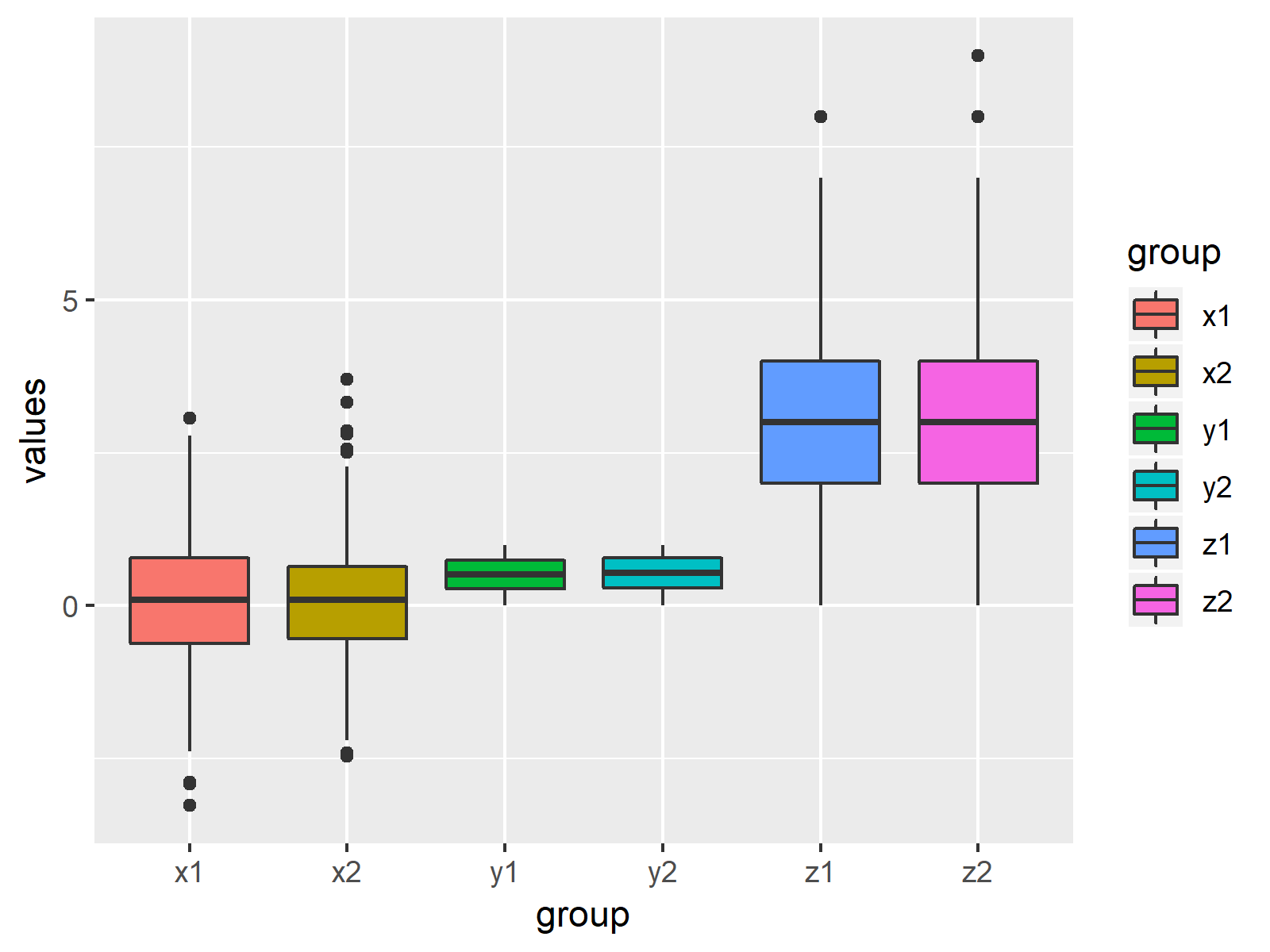
So far, we have created all the graphs and images with the boxplot function of Base R. However, there are also many packages that provide pretty designs and additional modification possibilities for boxplots.

In the example, I’ll show you how to create a boxplot with the [ggplot2 package](https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf). Let’s install and load the package to RStudio:

|  |
| --- |
| install.packages("ggplot2") # Install and load ggplot2  library("ggplot2") |

Now, we can use the ggplot and geom\_boxplot functions of the ggplot2 package to create a boxplot:

|  |
| --- |
| ggplot(data2, aes(x = **group**, y = values, fill = **group**)) + # Create boxplot chart in ggplot2  geom\_boxplot() |



***Figure 9: Boxplots Created by ggplot2 Package.***

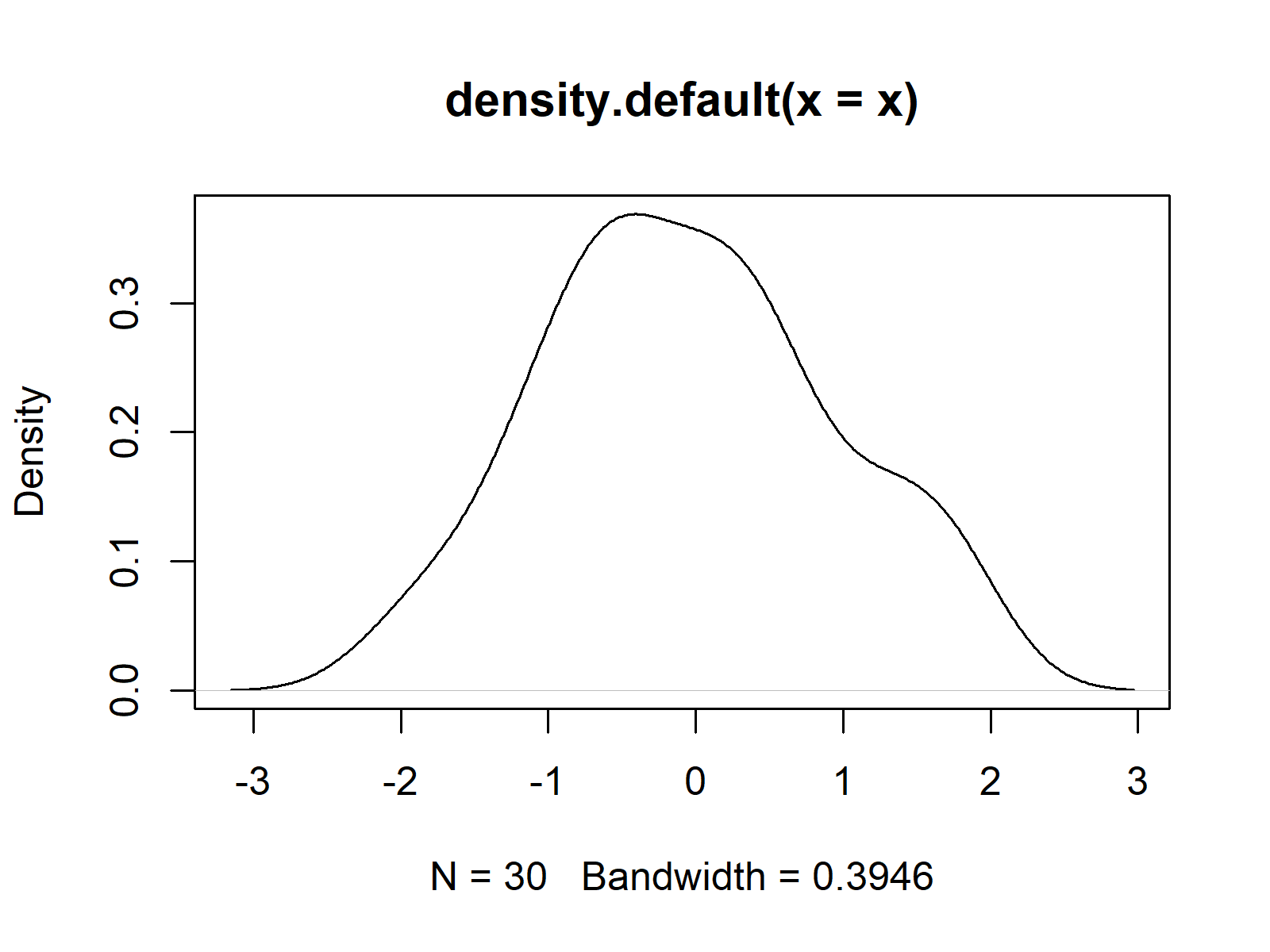
There are many other packages providing different designs and styles. However, the ggplot2 package is the most popular package among them.

1. **Density Plot**

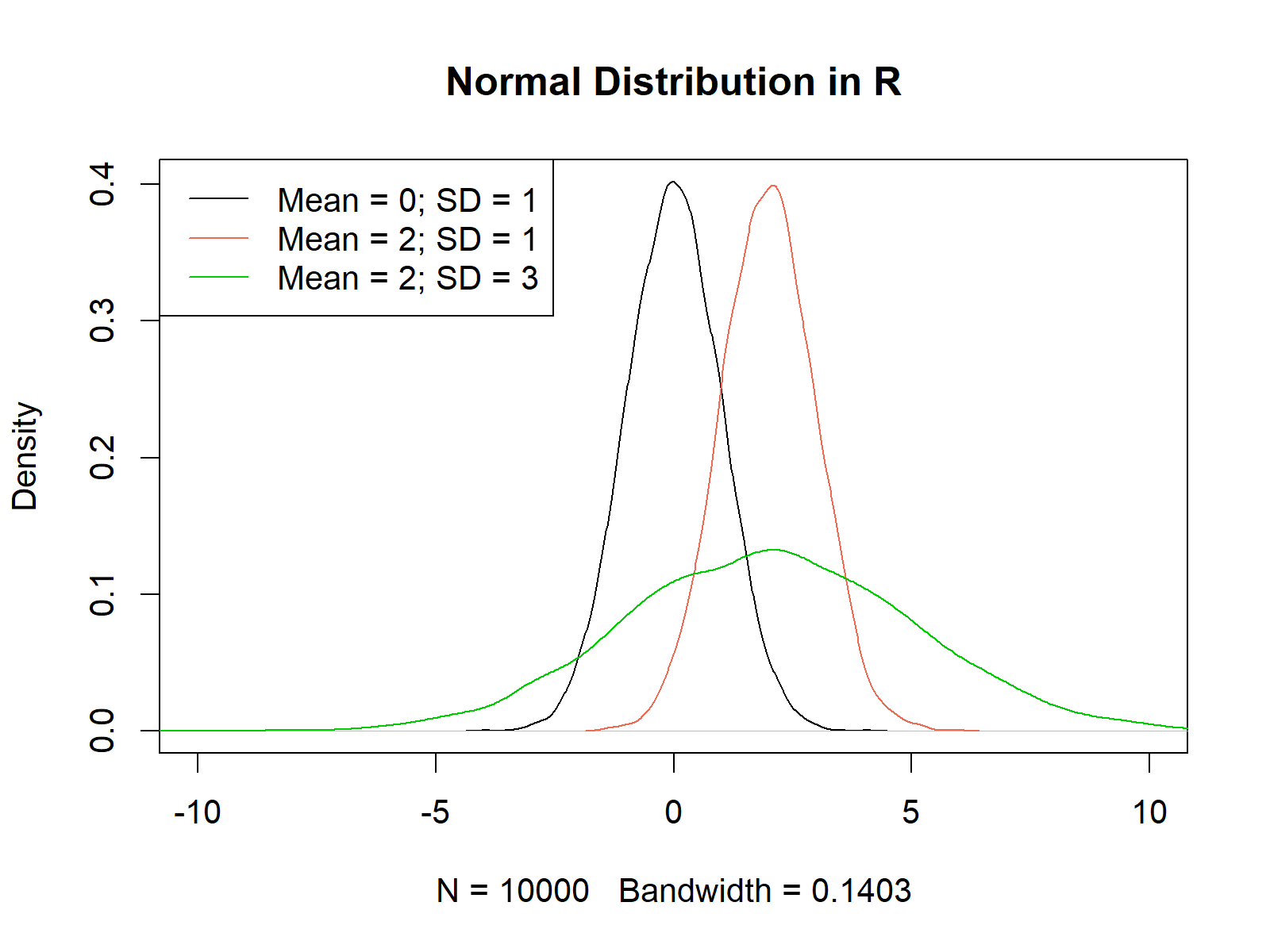
**Density Plot Definition:** A [density plot](https://datavizcatalogue.com/methods/density_plot.html) (or kernel density plot; density trace graph) shows the distribution of a numerical variable over a continuous interval. Peaks of a density plot visualize where the values of numerical variables are concentrated.

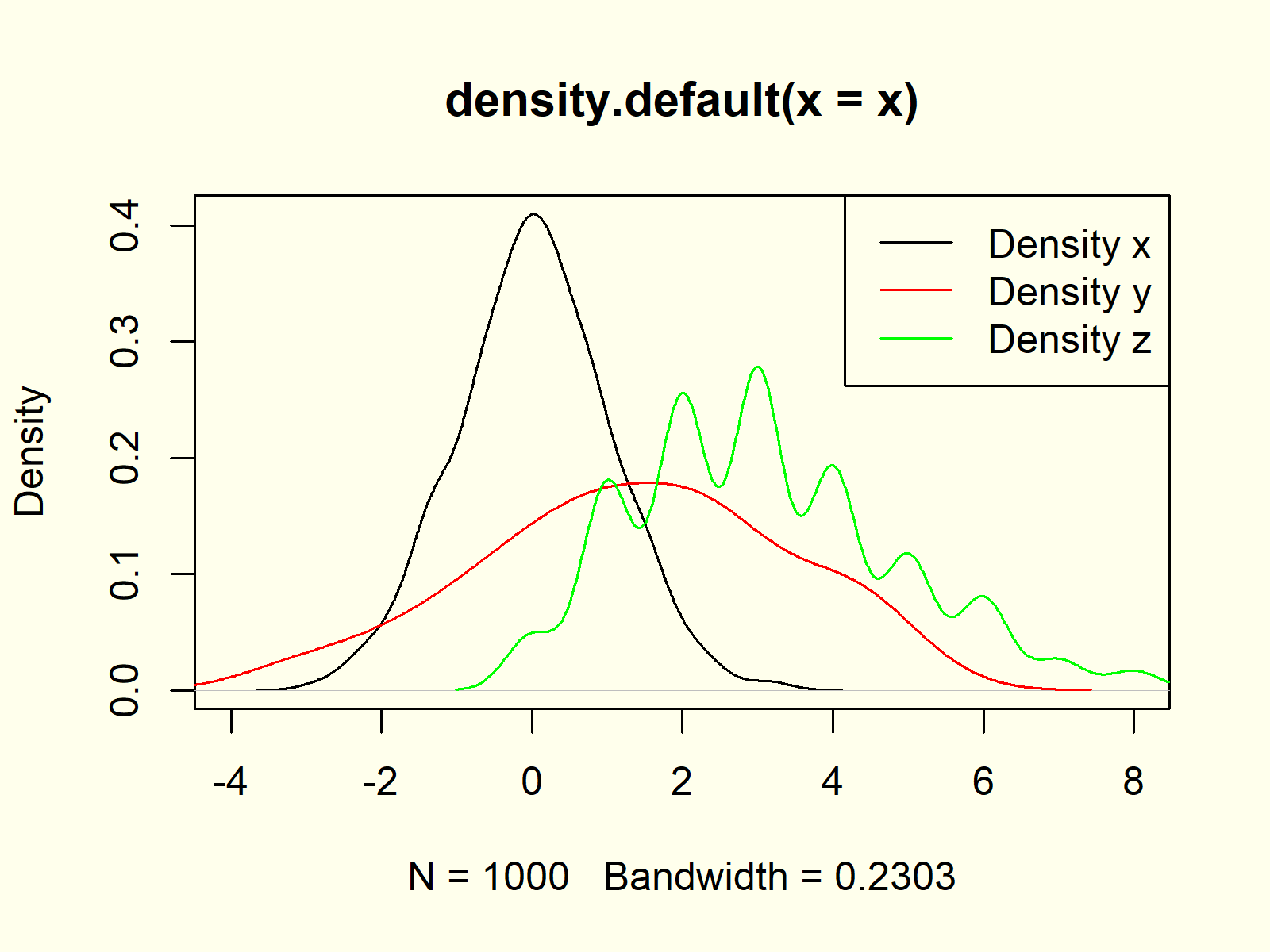
The following R syntax shows how to draw a basic density plot in R:

|  |
| --- |
| plot(density(x))  # Draw density plot in R |



**Advanced Density Plots:** Find some advanced density plots below. Click on the images to get more information and example R codes for each of the density plots.

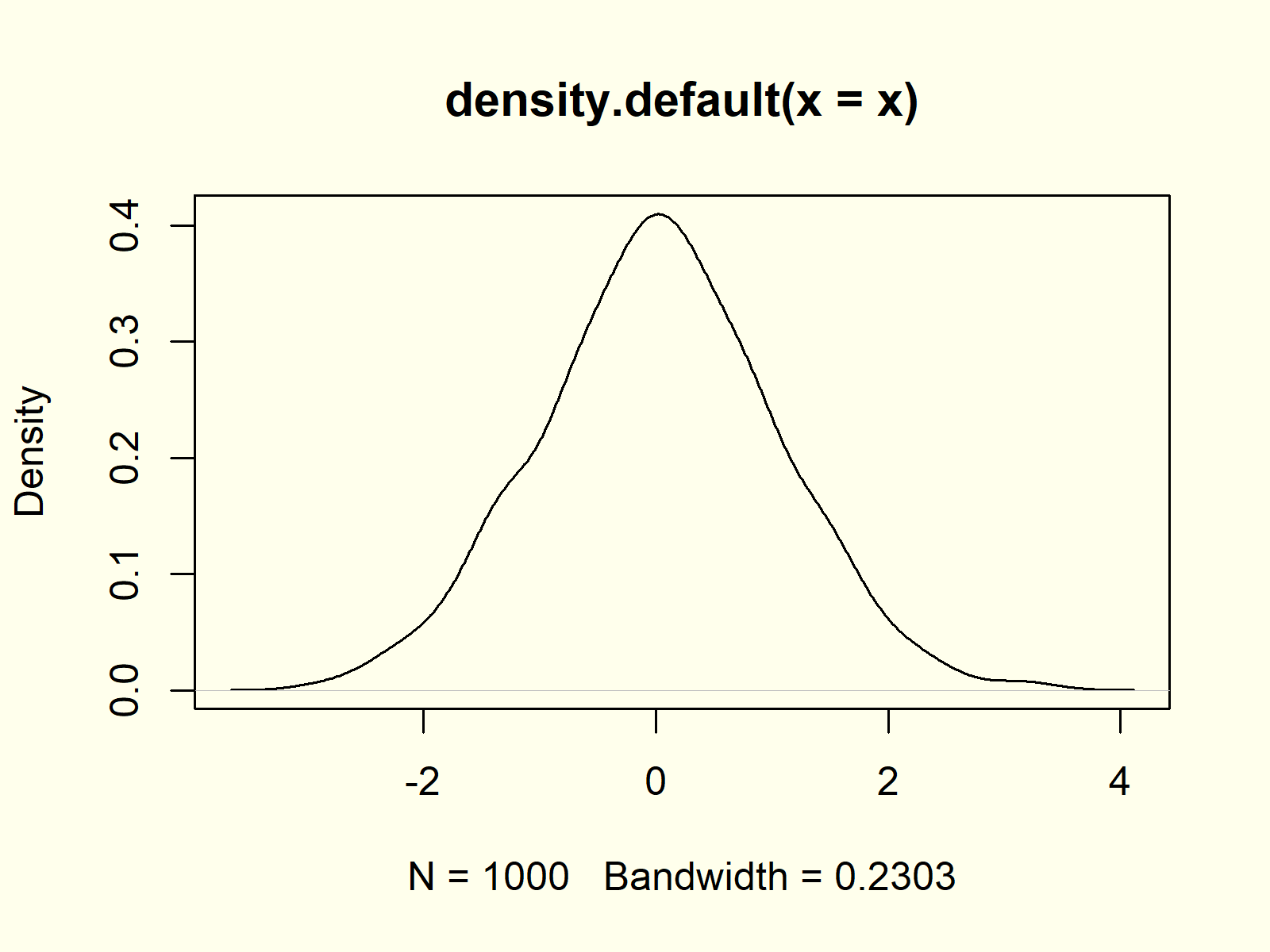




## Example 1: Basic Kernel Density Plot in Base R

If we want to create a [kernel density plot](http://core.ecu.edu/psyc/wuenschk/docs30/Kernal%20Density%20Plots.pdf) (or probability density plot) of our data in [Base R](https://stat.ethz.ch/R-manual/R-devel/library/base/html/00Index.html), we have to use a combination of the [plot() function](https://statisticsglobe.com/plot-in-r-example) and the density() function:

|  |
| --- |
| plot(density(x)) # Create basic density plot |

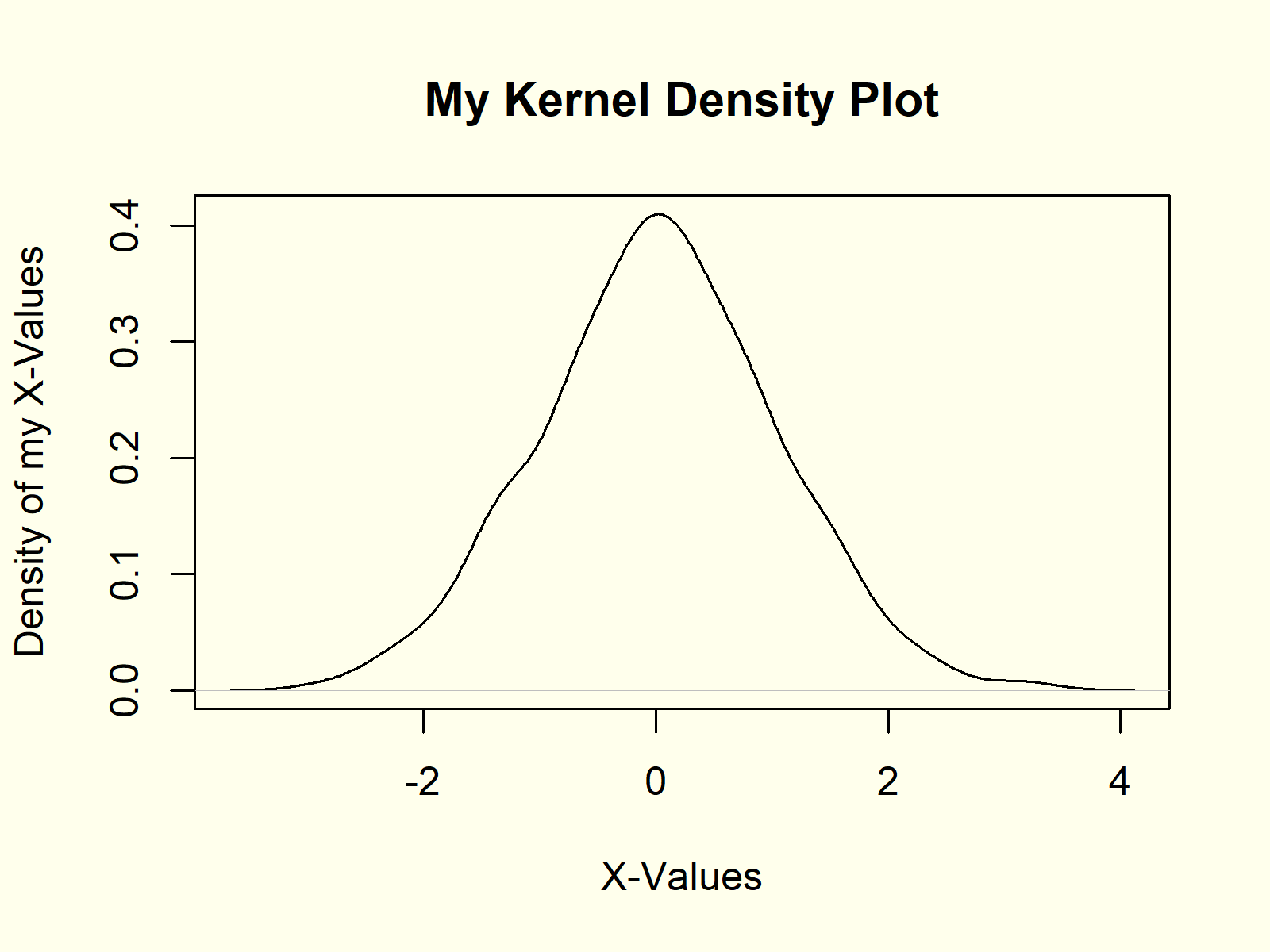


***Figure 1: Basic Kernel Density Plot in R.***

## Example 2: Modify Main Title & Axis Labels of Density Plot

The plot and density functions provide many options for the modification of density plots. With the main, xlab, and ylab arguments we can change the main title and axis labels of a density chart:

|  |
| --- |
| plot(density(x), # Modify main title & labels  main = "My Kernel Density Plot",  xlab = "X-Values",  ylab = "Density of my X-Values") |

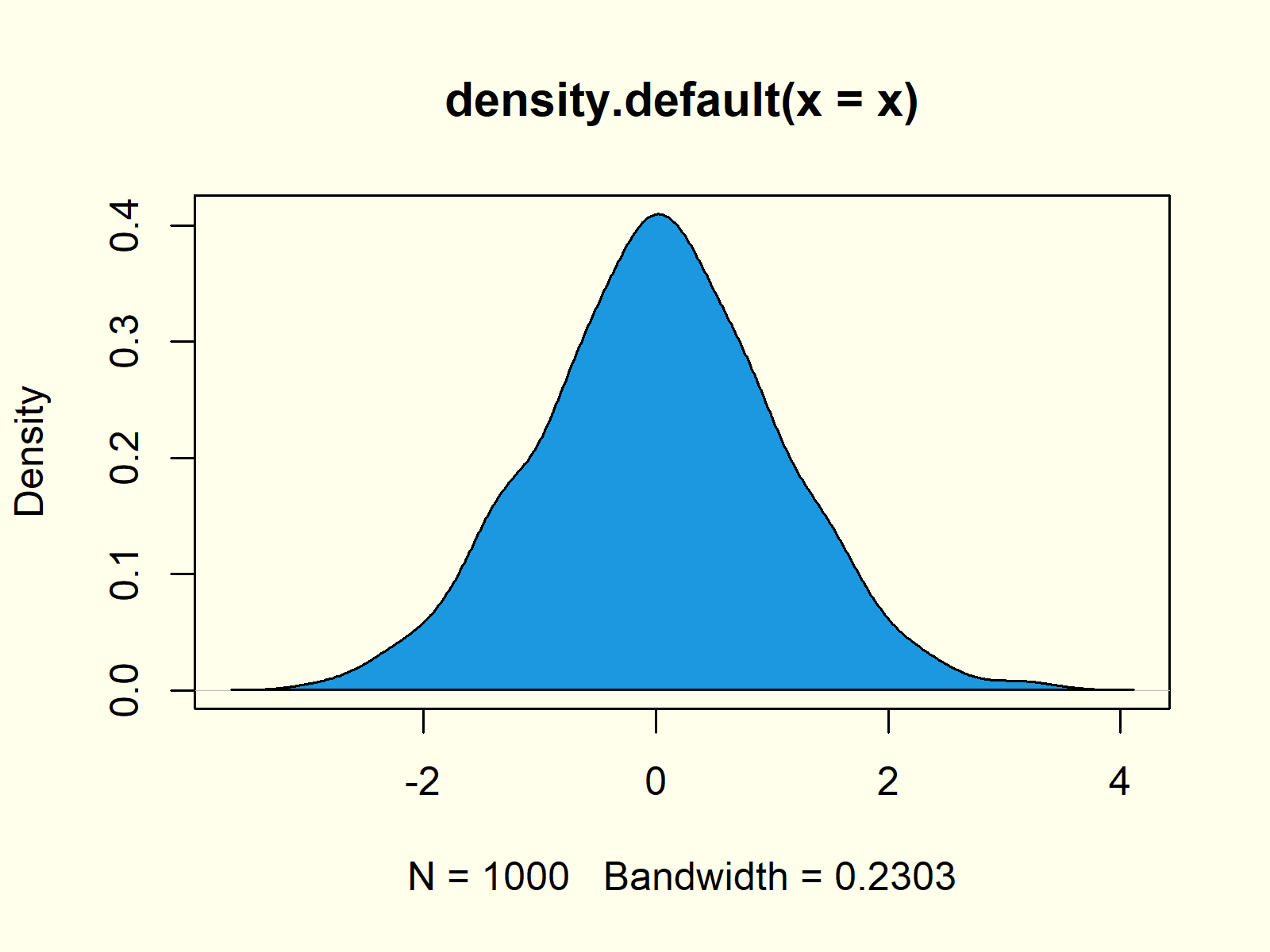


***Figure 2: Density Plot with Manual Text.***

## Example 3: Create Polygon Below Density Plot

We can also fill the area below the density with some color by using the [polygon function](https://statisticsglobe.com/r-polygon-function-plot/) in combination with the density function:

|  |
| --- |
| plot(density(x)) # Create polygon density  polygon(density(x), col = "#1b98e0") |

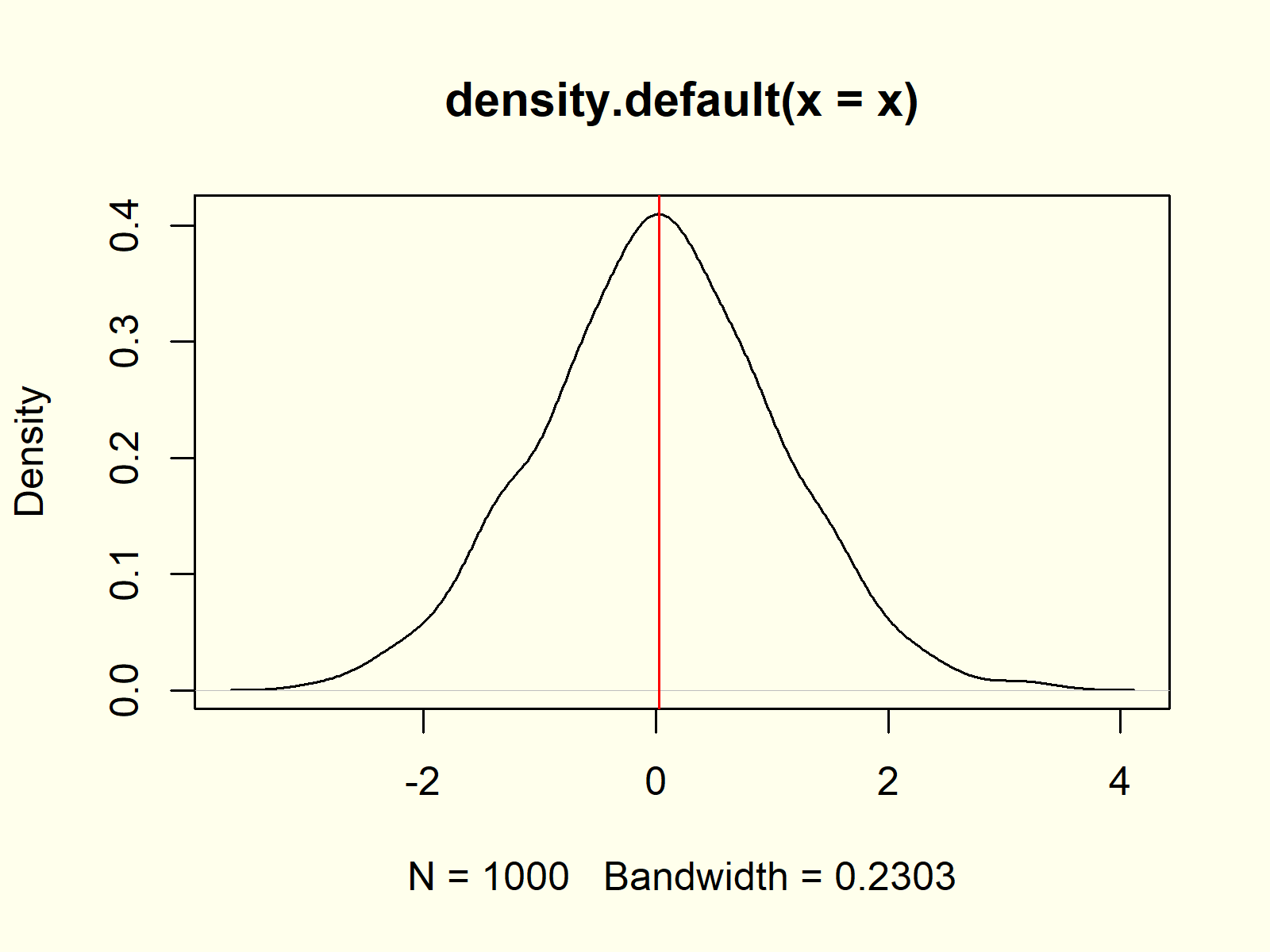


***Figure 3: Blue Polygon Below Density.***

## Example 4: Add Vertical Line for Mean to Density Plot

We may draw additional lines or segments to our density plot [with the abline function](https://statisticsglobe.com/abline-function-in-r/). In this example, I’m showing you how to add a red vertical line at the position of the [mean of our data](https://statisticsglobe.com/mean-function-in-r).

|  |
| --- |
| plot(density(x)) # Add mean line to density  abline(v = mean(x), col = "red") |

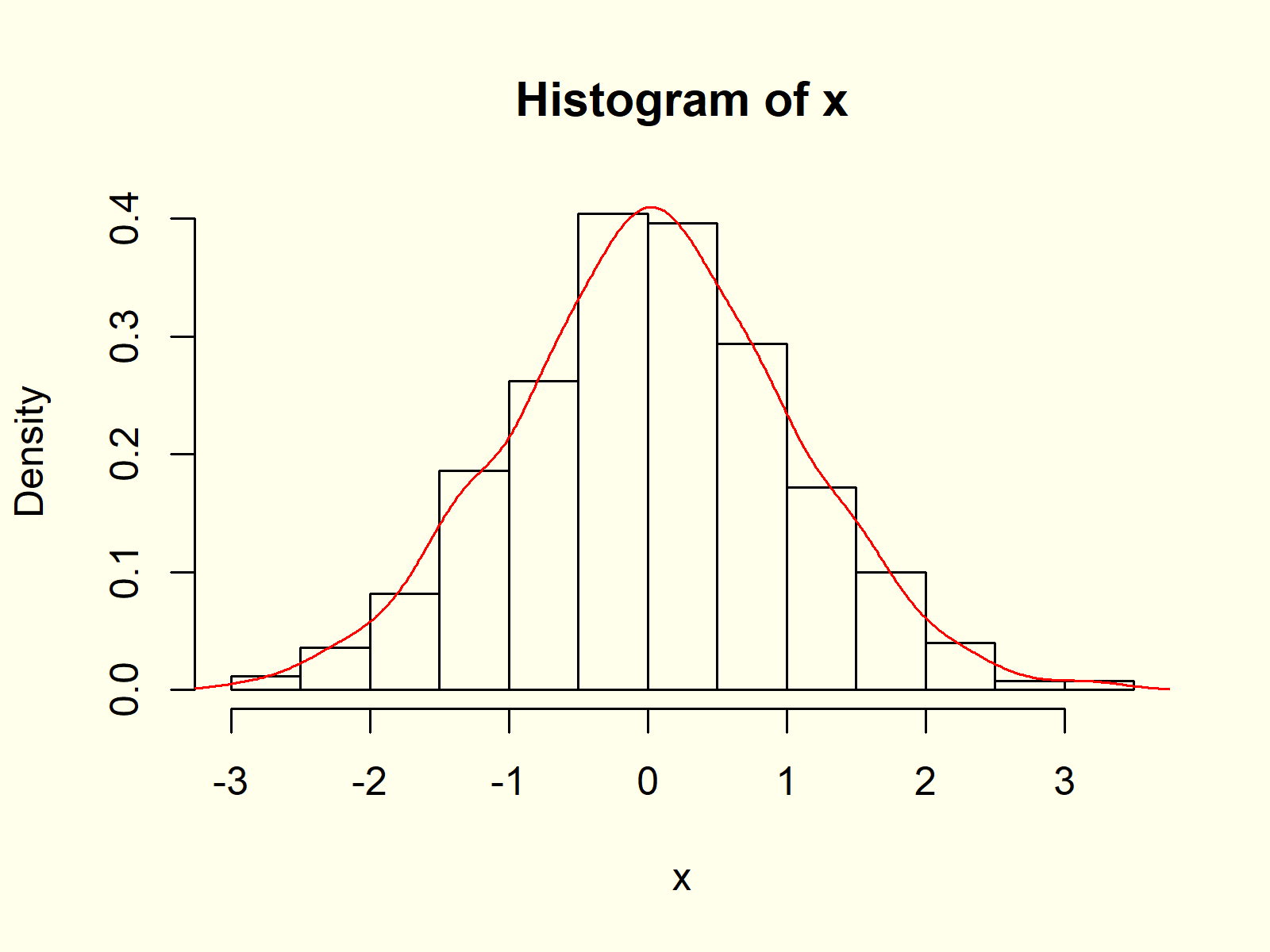


***Figure 4: Red Vertical Mean Line.***

## Example 5: Histogram & Density in Same Plot

It is possible to overlay existing graphics or diagrams with a density plot in R. This example shows [how to draw a histogram](https://statisticsglobe.com/histogram-in-base-r-hist-function) and a density in the same plot:

|  |
| --- |
| hist(x, prob = **TRUE**) # Histogram and density  lines(density(x), col = "red") |



***Figure 5: Histogram and Density in One Graph.***

Note that we have to use the lines) function instead of the plot function, in case we want to overlay an already existing graph with a density plot.

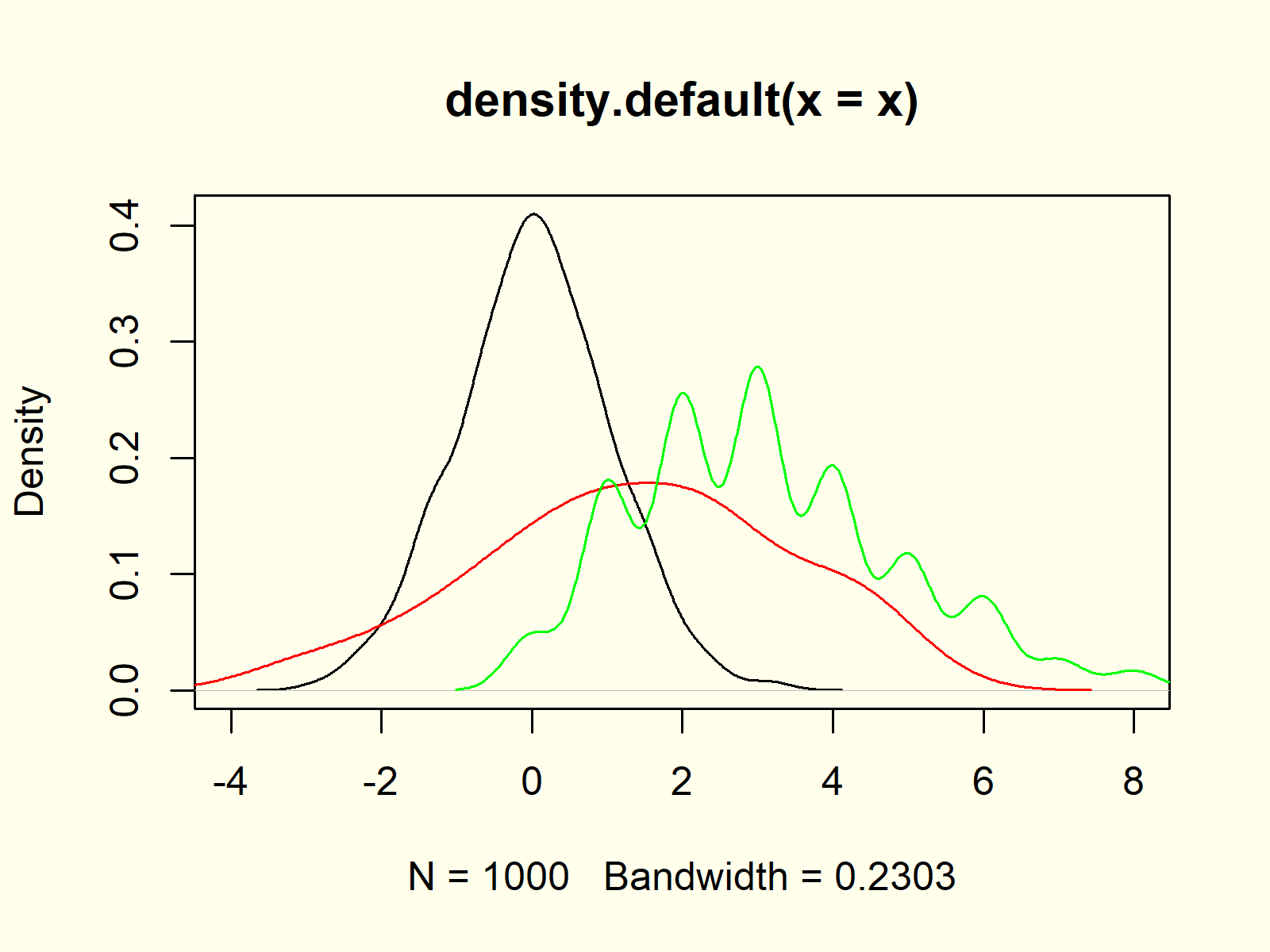
## Example 6: Multiple Densities in Same Plot

Similar to Example 5, we can create a graphic with multiple density plots in the same image. First, we need to create more [random variables](https://statisticsglobe.com/random-numbers-in-r) for the additional densities:

|  |
| --- |
| y <- rnorm(100, 1, 2) # Create two more variables  z <- rpois(1000, 3) |

Now, we can overlay our original density with these new densities:

|  |
| --- |
| plot(density(x), xlim = c(- 4, 8)) # Plot density of x  lines(density(y), col = "red") # Overlay density of y  lines(density(z), col = "green") # Overlay density of z |



***Figure 6: Several Densities in Same Graphic.***

Note that we had to replace the plot function with the lines function to keep all probability densities in the same graphic (as already explained in Example 5).

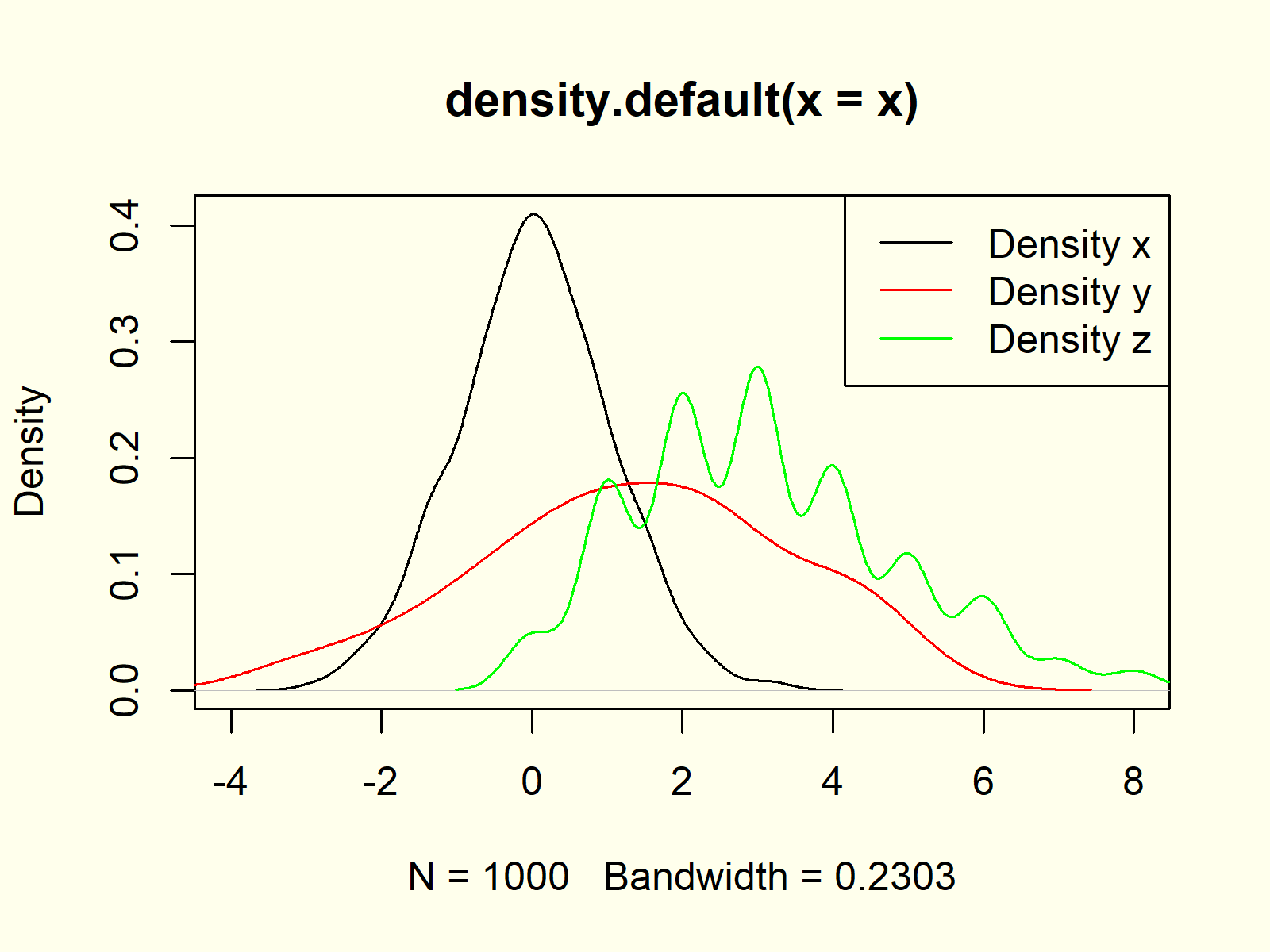
## Example 7: Add Legend to Density Plot

Whenever we visualize several variables or columns in the same picture, it makes sense to create a legend. First, let’s plot our data as already shown in Example 6:

|  |
| --- |
| plot(density(x), xlim = c(- 4, 8)) # Plot density of x  lines(density(y), col = "red") # Overlay density of y  lines(density(z), col = "green") # Overlay density of z |

Now, we can use the legend function to add a legend to our plot:

|  |
| --- |
| legend("topright", # Add legend to density  legend = c("Density x", "Density y", "Density z"),  col = c("black", "red", "green"),  lty = 1) |



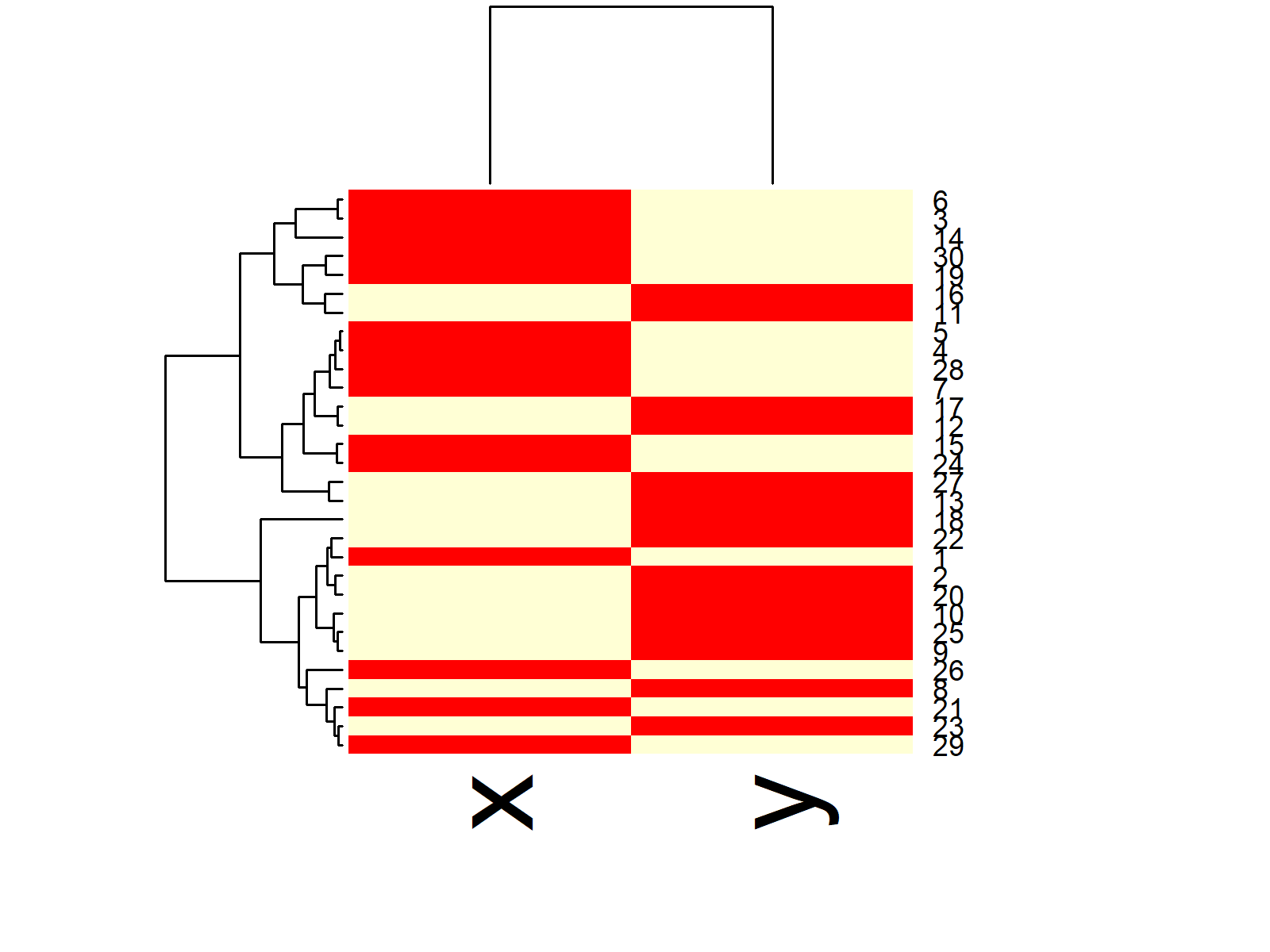
***Figure 7: Add Legend to Density Plot.***

1. **Heatmap**

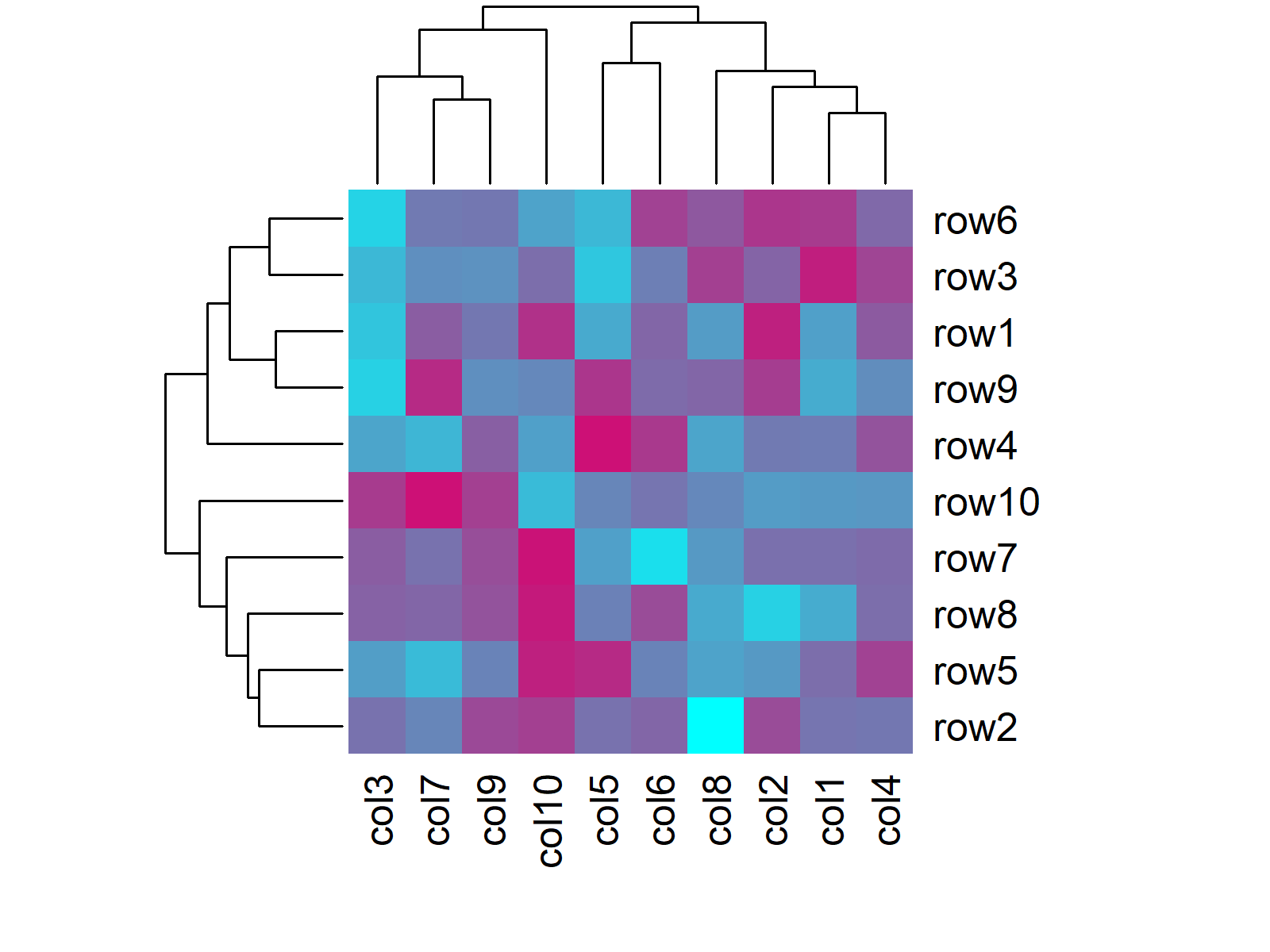
**Heatmap Definition:** A [heatmap](https://en.wikipedia.org/wiki/Heat_map) (or shading matrix) visualizes individual values of a matrix with colors. More common values are typically indicated by brighter reddish colors and less common values are typically indicated by darker colors.

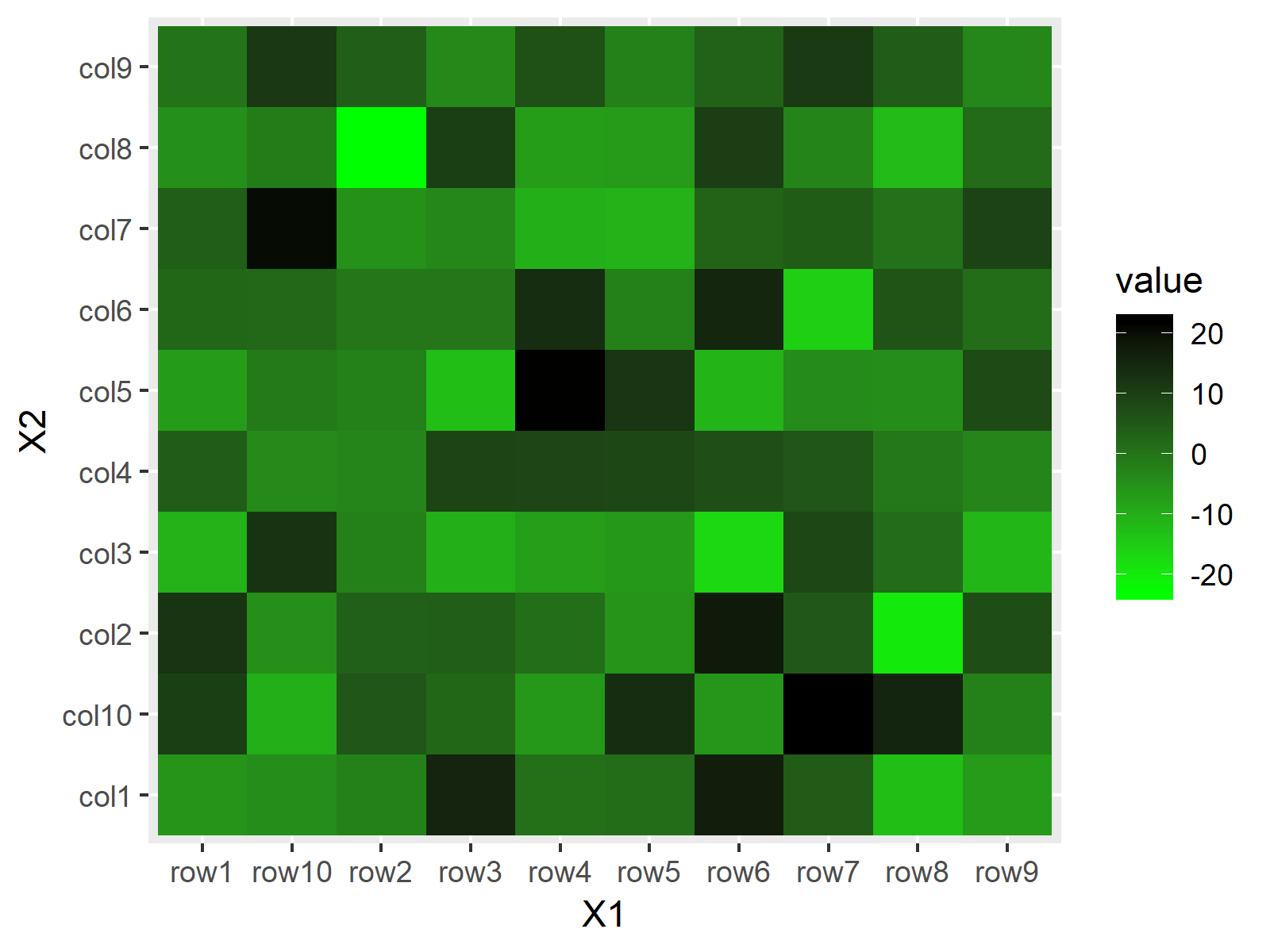
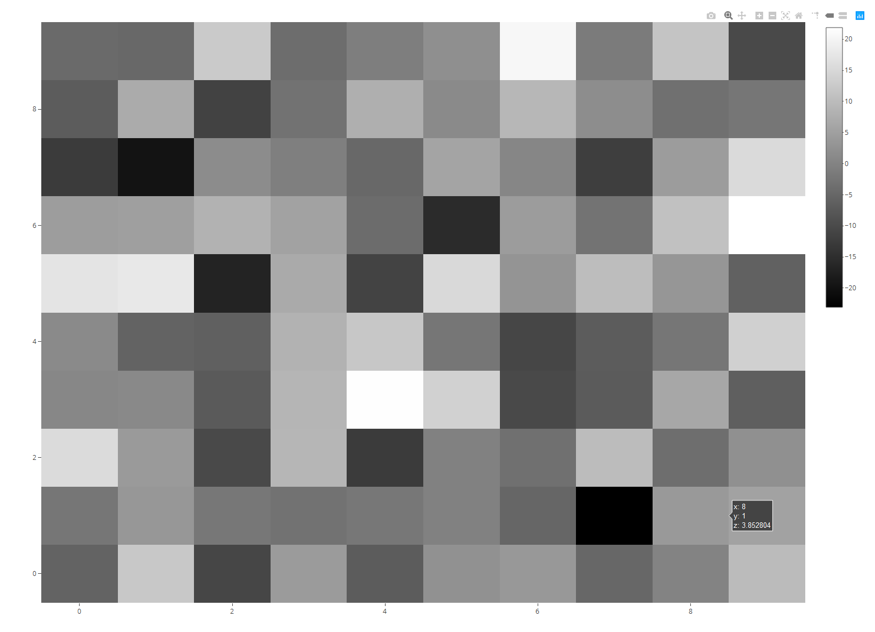
The following R syntax shows how to draw a basic heatmap in R:

|  |
| --- |
| heatmap(cbind(x, y)) # Draw heatmap in R |



**Advanced Heatmaps:** Find some advanced heatmaps below. Click on the images to get more information and example R codes for each of the heatmaps .





Construction of Example Data

We’ll use the following matrix for the examples of this R tutorial:

|  |
| --- |
| **set**.seed(123) # Set seed for reproducibility  data <- matrix(rnorm(100, 0, 10), nrow = 10, ncol = 10) # Create example data  colnames(data) <- paste0("col", 1:10) # Column names  rownames(data) <- paste0("row", 1:10) # Row names |

Our data contains ten columns and ten rows with [normally distributed](https://statisticsglobe.com/normal-distribution-in-r-dnorm-pnorm-qnorm-rnorm) random values.

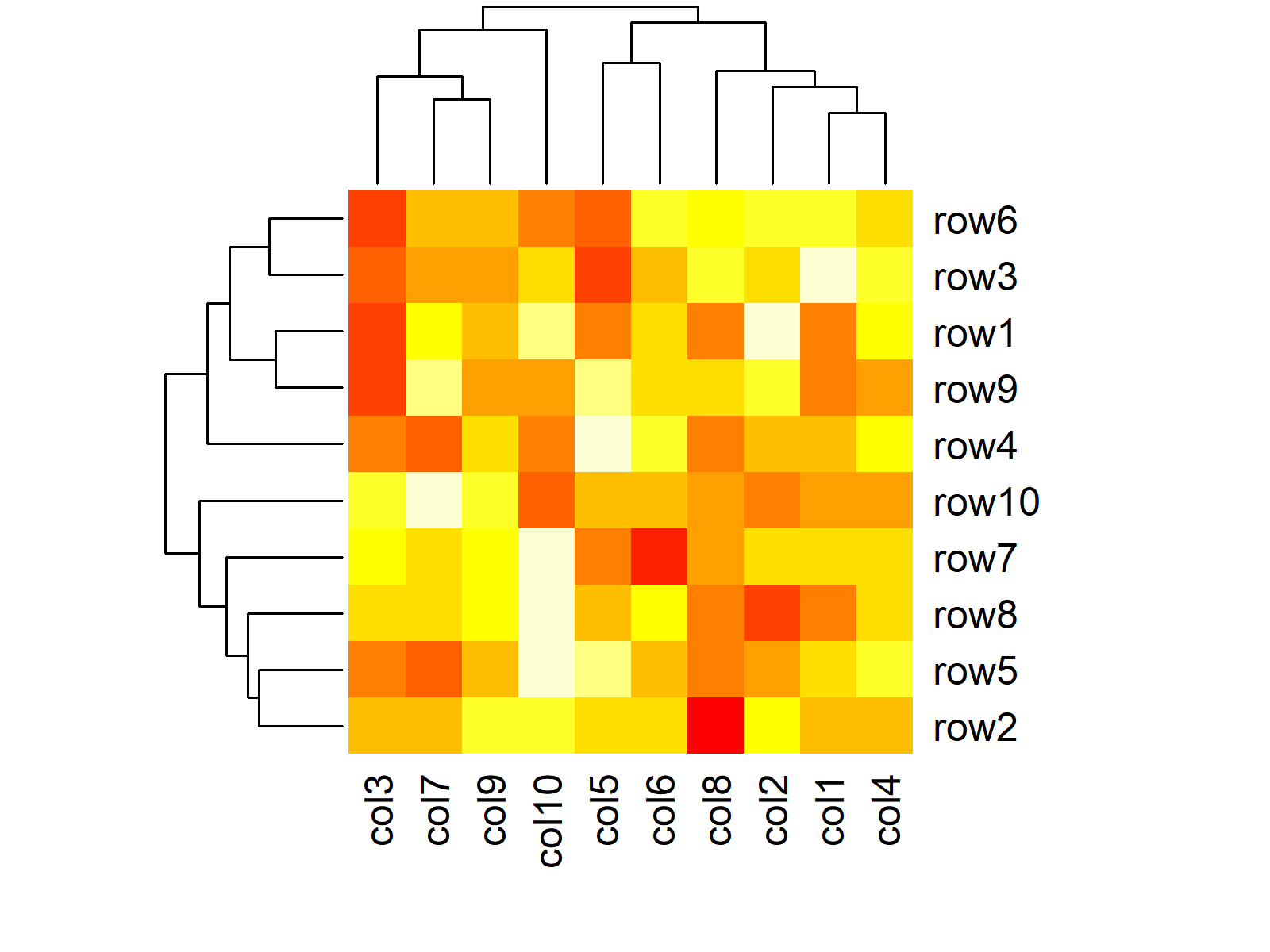
In the following examples, I’ll show how to create heatmaps in R based on different functions and packages. It’s useful to know different ways to create heatmaps, since every package provides a different heatmap design. So keep on reading until the end of the article!

Example 1: Create Heatmap with heatmap Function [Base R]

The most common function for creating [heatmaps](https://imotions.com/blog/analyze-heat-maps/) in R is the heatmap() function, which is already provided by the [base installation of R](https://www.datacamp.com/community/tutorials/installing-R-windows-mac-ubuntu).

The heatmap function is applied as shown below:

|  |
| --- |
| heatmap(data) # Apply heatmap function |

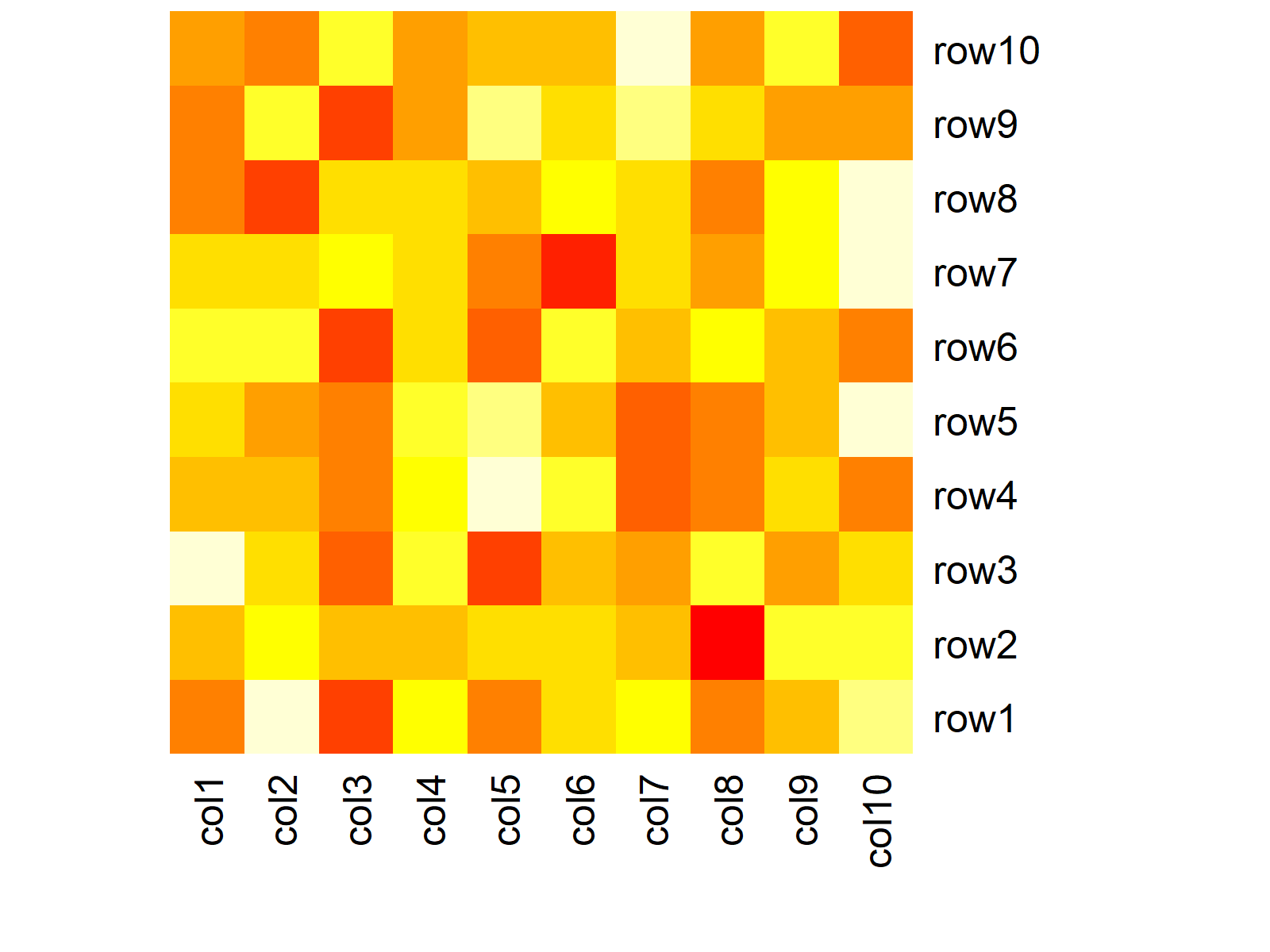


***Figure 1: Default Heatmap in Base R.***

Figure 1 illustrates the output of the previous R code. By default, the heatmap function returns a heatmap with column and row names as well as a [dendrogram](https://www.displayr.com/what-is-dendrogram/).

If we want, we can disable the automatically created dendrogram:

|  |
| --- |
| heatmap(data, Rowv = NA, Colv = NA) # Remove dendrogram |



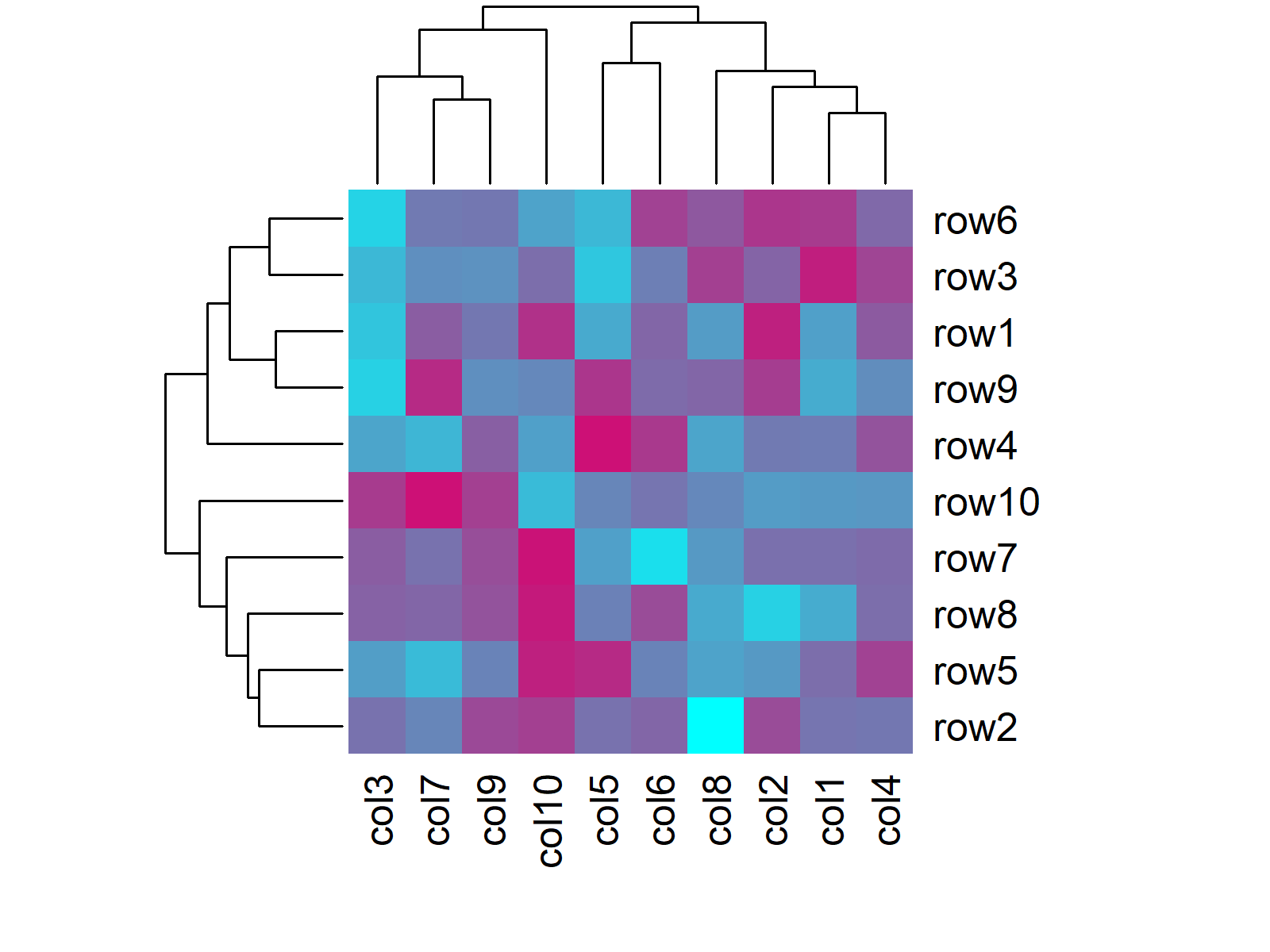
***Figure 2: Heatmap without Dendrogram in Base R.***

Furthermore, we can modify the colors of the heatmap by specifying our own color range with the colorRampPalette function. The following R code produces a function, which creates color ranges between the [colors cyan and deeppink3](http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf):

|  |
| --- |
| my\_colors <- colorRampPalette(c("cyan", "deeppink3")) # Manual color range |

Now, we can create a heatmap with this color range as follows:

|  |
| --- |
| heatmap(data, col = my\_colors(100)) # Heatmap with manual colors |



***Figure 3: Heatmap with Manual Color Range in Base R.***

Example 2: Create Heatmap with geom\_tile Function [ggplot2 Package]

As already mentioned in the beginning of this page, many [R packages](https://statisticsglobe.com/r-packages-list) are providing functions for the creation of heatmaps in R.

A popular package for graphics is the ggplot2 package of the [tidyverse](https://www.tidyverse.org/" \t "_blank) and in this example I’ll show you how to create a heatmap with ggplot2.

The ggplot2 package requires a long data format. We can create this data format with the reshape package…

|  |
| --- |
| install.packages("reshape") # Install reshape package  library("reshape") # Load reshape package |

…and the melt function:

|  |
| --- |
| data\_melt <- melt(data) # Reorder data  head(data\_melt) # First six rows of data  # X1 X2 value  # row1 col1 -14.15254153  # row2 col1 -6.88199228  # row3 col1 2.44972363  # row4 col1 -0.04076637  # row5 col1 3.54613210  # row6 col1 -2.57462492 |

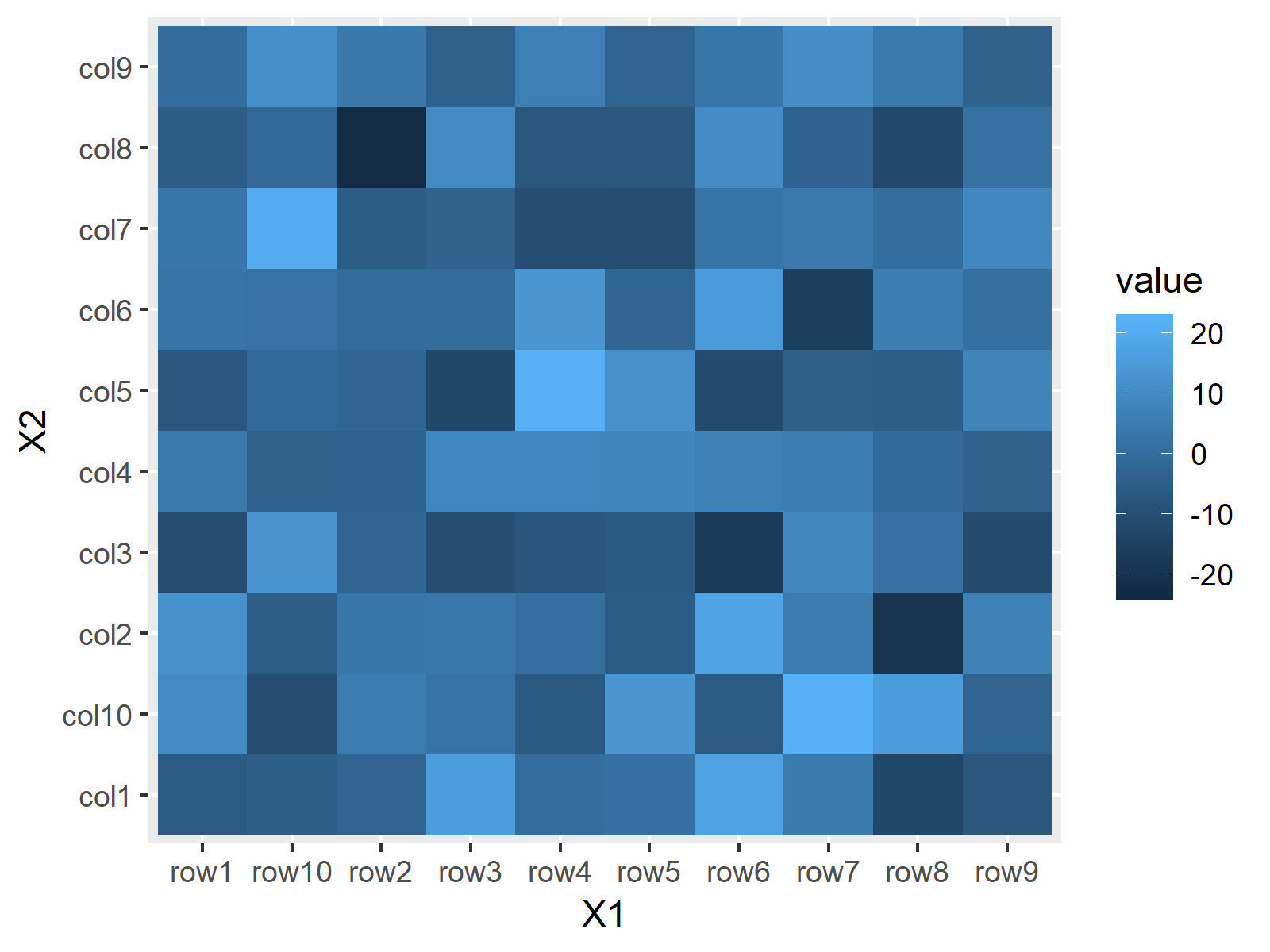
The RStudio console output shows the first 6 rows of our reshaped data. As you can see, the melt function created the two columns X1 and X2, which are containing every possible row and column combination, and a third column with the name value, which is containing the corresponding values.

In order to draw a heatmap with the ggplot2 package, we also need to install and load ggplot2:

|  |
| --- |
| install.packages("ggplot2") # Install ggplot2 package  library("ggplot2") # Load ggplot2 package |

Now, we can use the geom\_tile function of the ggplot2 package to make a basic heatmap:

|  |
| --- |
| ggp <- ggplot(data\_melt, aes(X1, X2)) + # Create heatmap with ggplot2  geom\_tile(aes(fill = **value**))  ggp # Print heatmap |

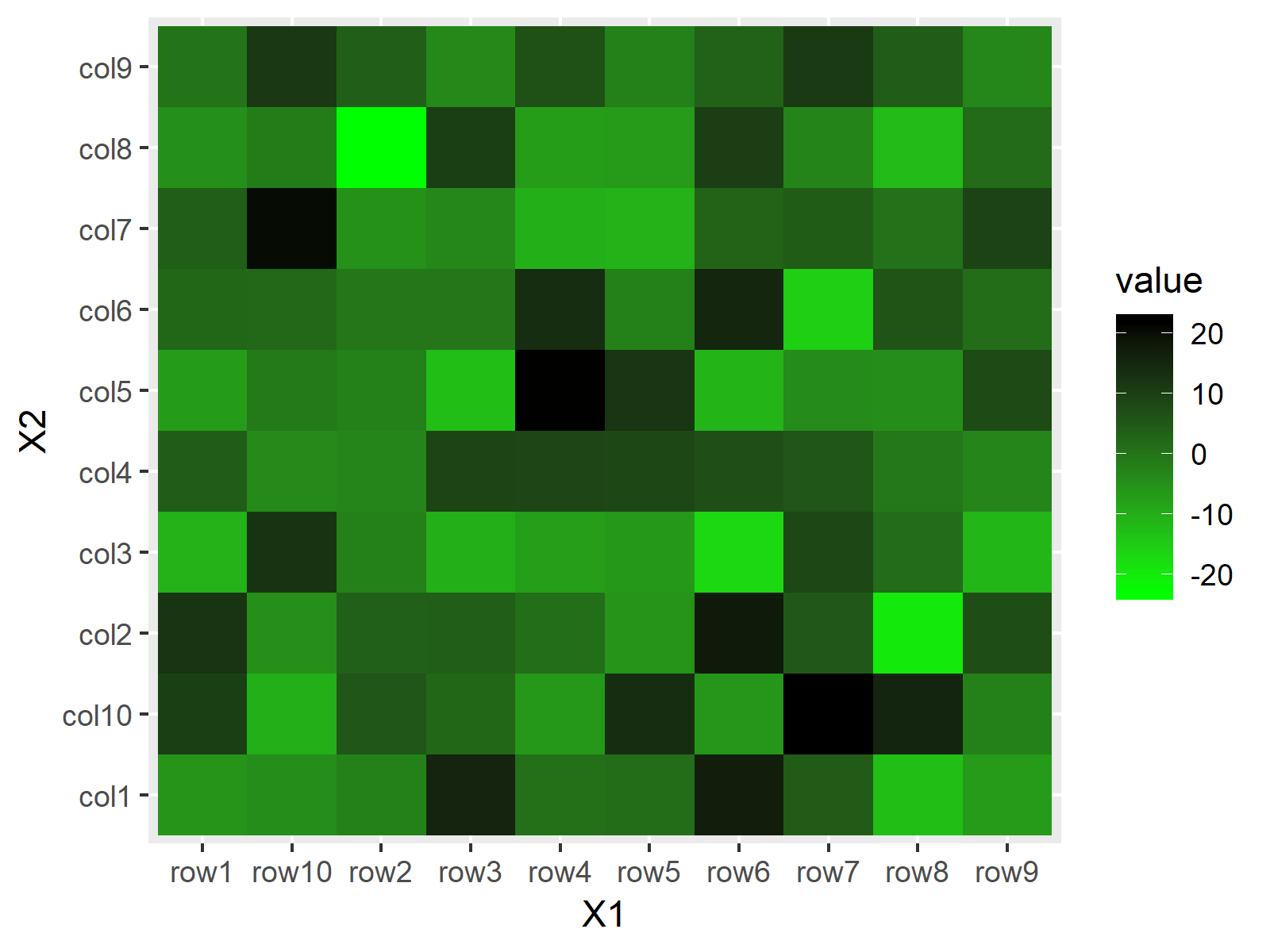


***Figure 4: Default Heatmap in ggplot2 Package.***

As you can see based on Figure 4, the patter of the heatmap cells is the same as in Base R. However, the general layout is in the typical ggplot2 style.

Of cause, ggplot2 also provides options for the modification of our heatmap. For instance, we can use the scale\_fill\_gradient function to draw a heatmap with a manual color range:

|  |
| --- |
| ggp + scale\_fill\_gradient(low = "green", high = "black") # Manual colors of heatmap |



***Figure 5: Heatmap with Manual Color Range in ggplot2 Package.***

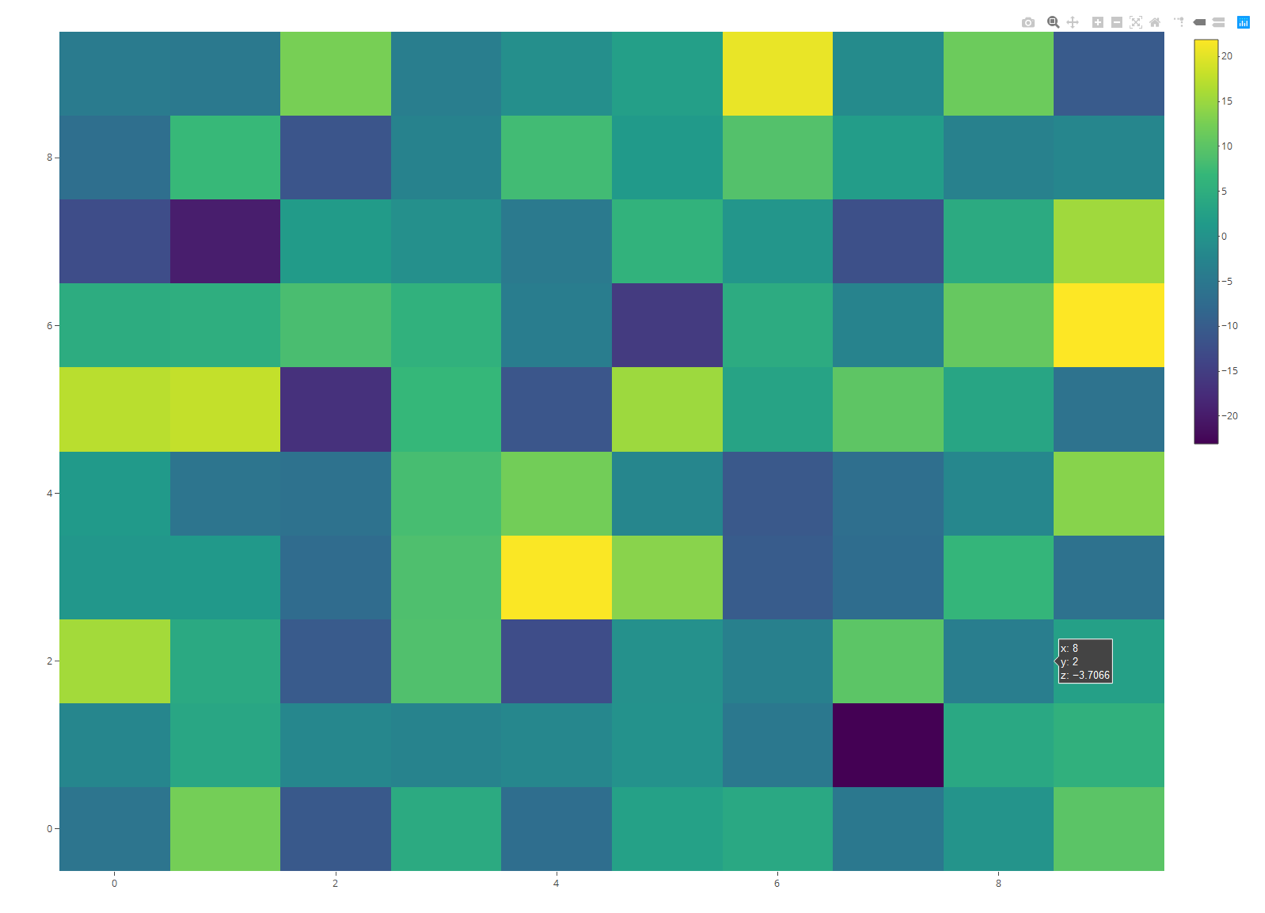
Example 3: Create Heatmap with plot\_ly Function [plotly Package]

Another popular package for heatmaps is the [plotly package](https://statisticsglobe.com/plotly-r-package):

|  |
| --- |
| install.packages("plotly") # Install plotly package  library("plotly") # Load plotly package |

The plotly package contains the plot\_ly function, which can be used to draw a heatmap by specifying type = “heatmap”:

|  |
| --- |
| plot\_ly(z = data, type = "heatmap") # Apply plot\_ly function |

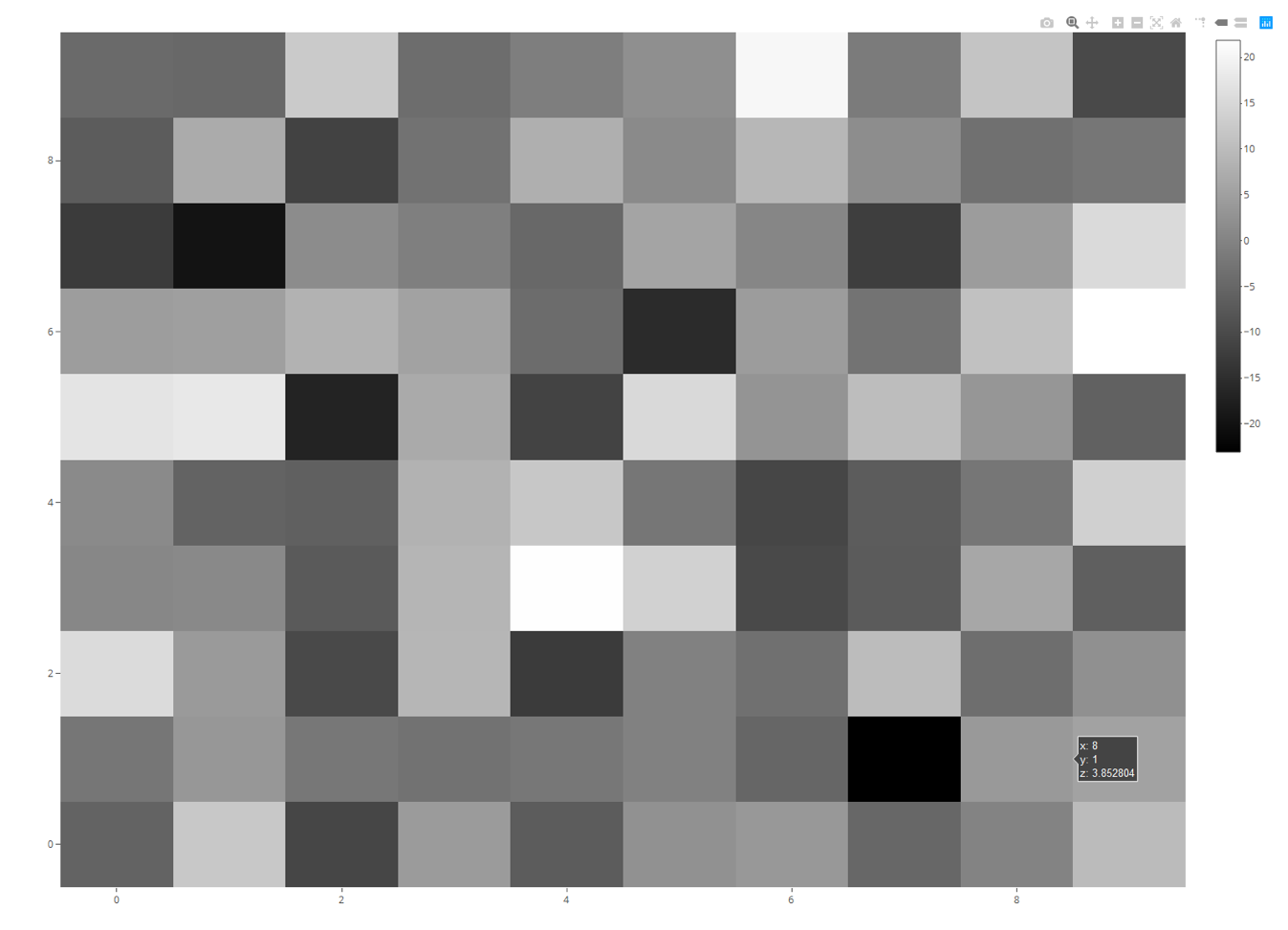


***Figure 6: Default Heatmap in plotly Package.***

Again, the patter is the same, but the general [plot style](https://statisticsglobe.com/graphics-in-r) is different.

The plotly package also provides additional options for the modification of the heatmap. If we want to change the color, we can either specify a color range manually, or we can use some predefined options such as colorscale = “Greys”:

|  |
| --- |
| plot\_ly(z = data, colorscale = "Greys", type = "heatmap") # Manual colors |



***Figure 7: Heatmap with Manual Color Range in plotly Package.***

As you can see based on Figure 7, the Greys specification created a heatmap in greyscale.

Note that the plotly package show its graphics in the RStudio viewer instead of the RStudio plot window. For that reason you need to [export these plots differently](https://stackoverflow.com/questions/40160125/how-to-save-a-plot-as-image-on-disk-from-viewer-in-rstudio).

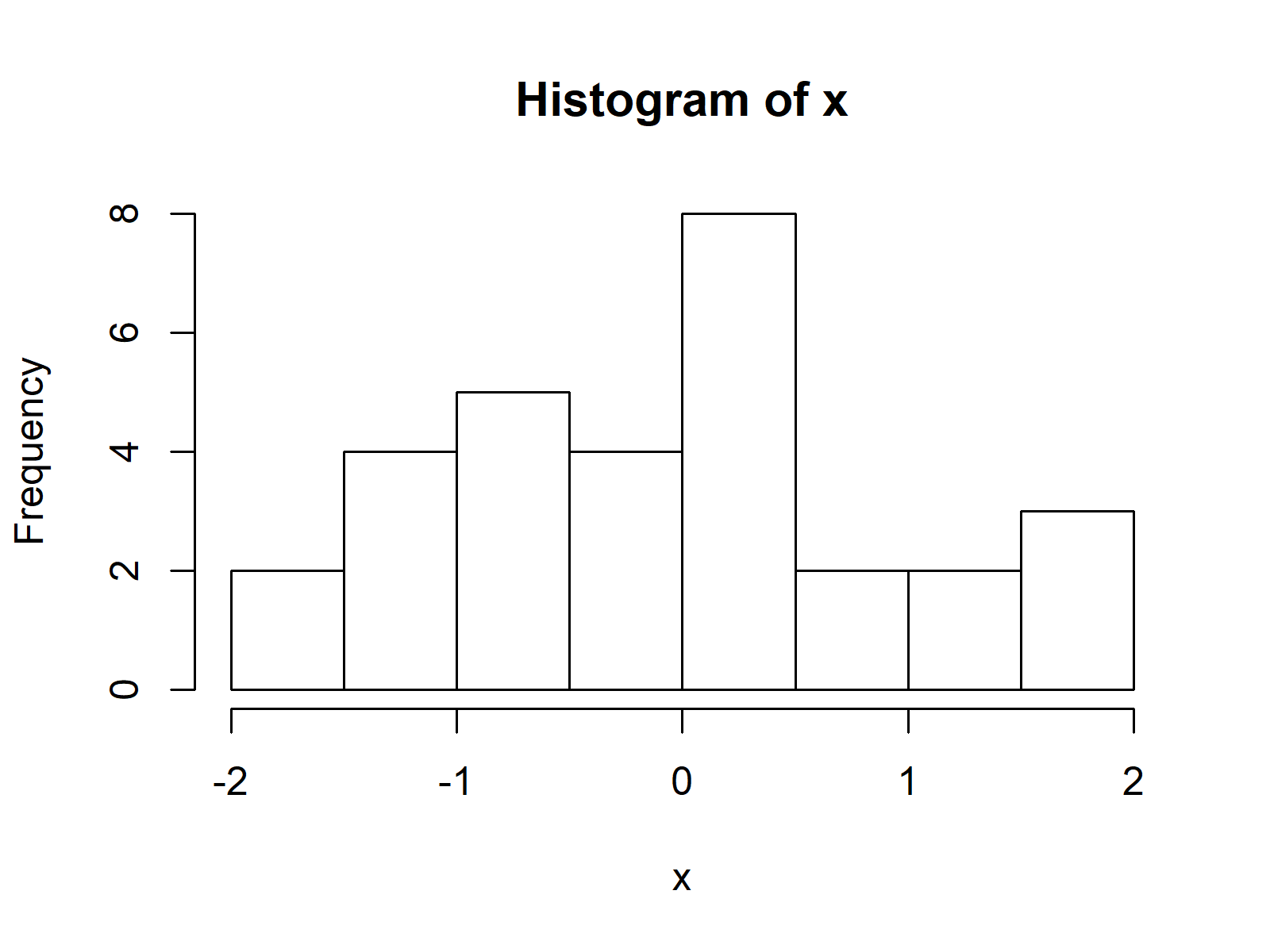
Also note that there are many other packages for the creation of heatmaps in R available. In my opinion, however, Base R, ggplot2, and plotly provide the best solutions.

1. **Histogram Plot**

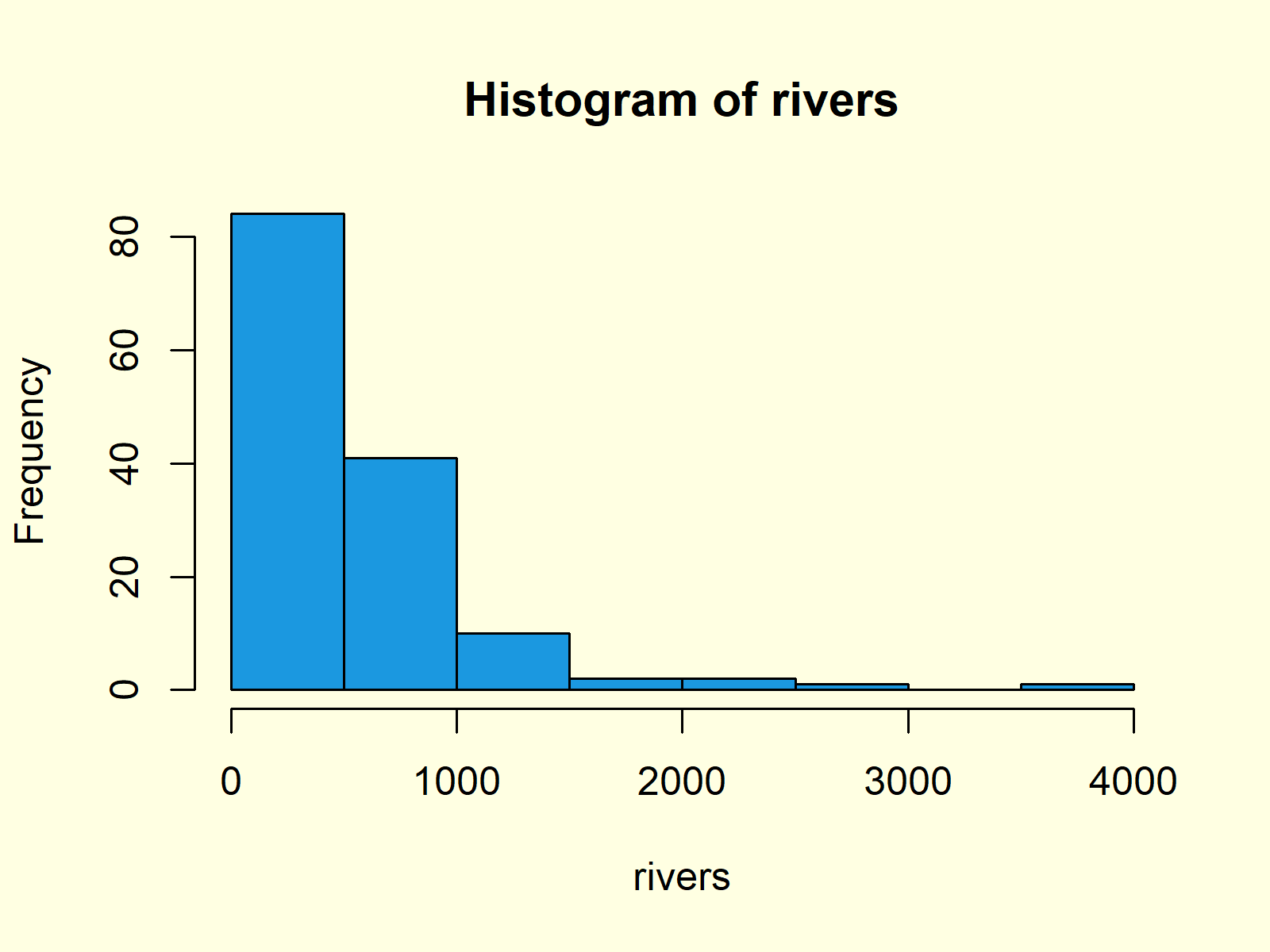
**Histogram Definition:** A [histogram](https://statistics.laerd.com/statistical-guides/understanding-histograms.php) groups continuous data into ranges and plots this data as bars. The height of each bar shows the amount of observations within each range.

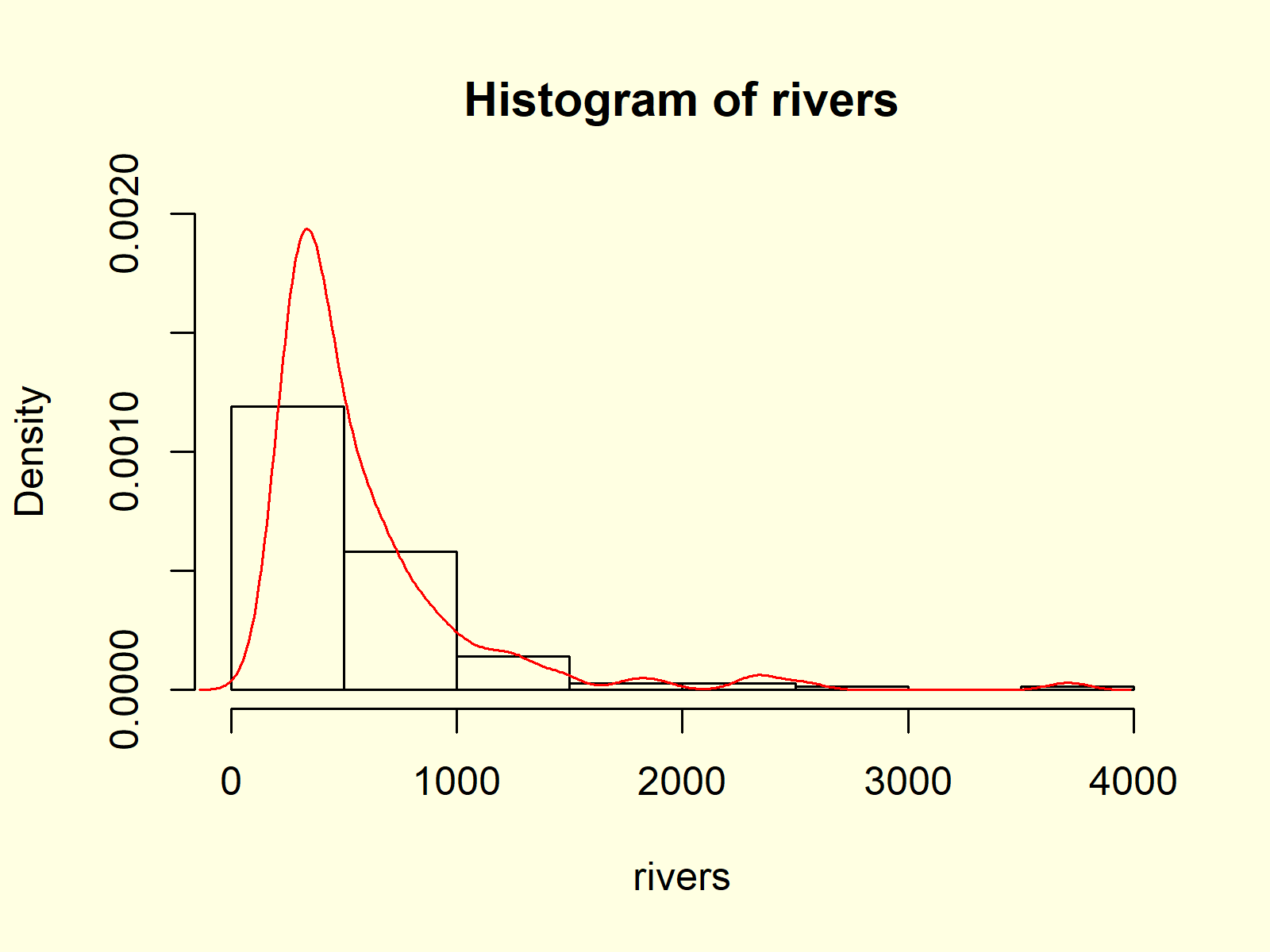
The following R syntax shows how to draw a basic histogram in R:

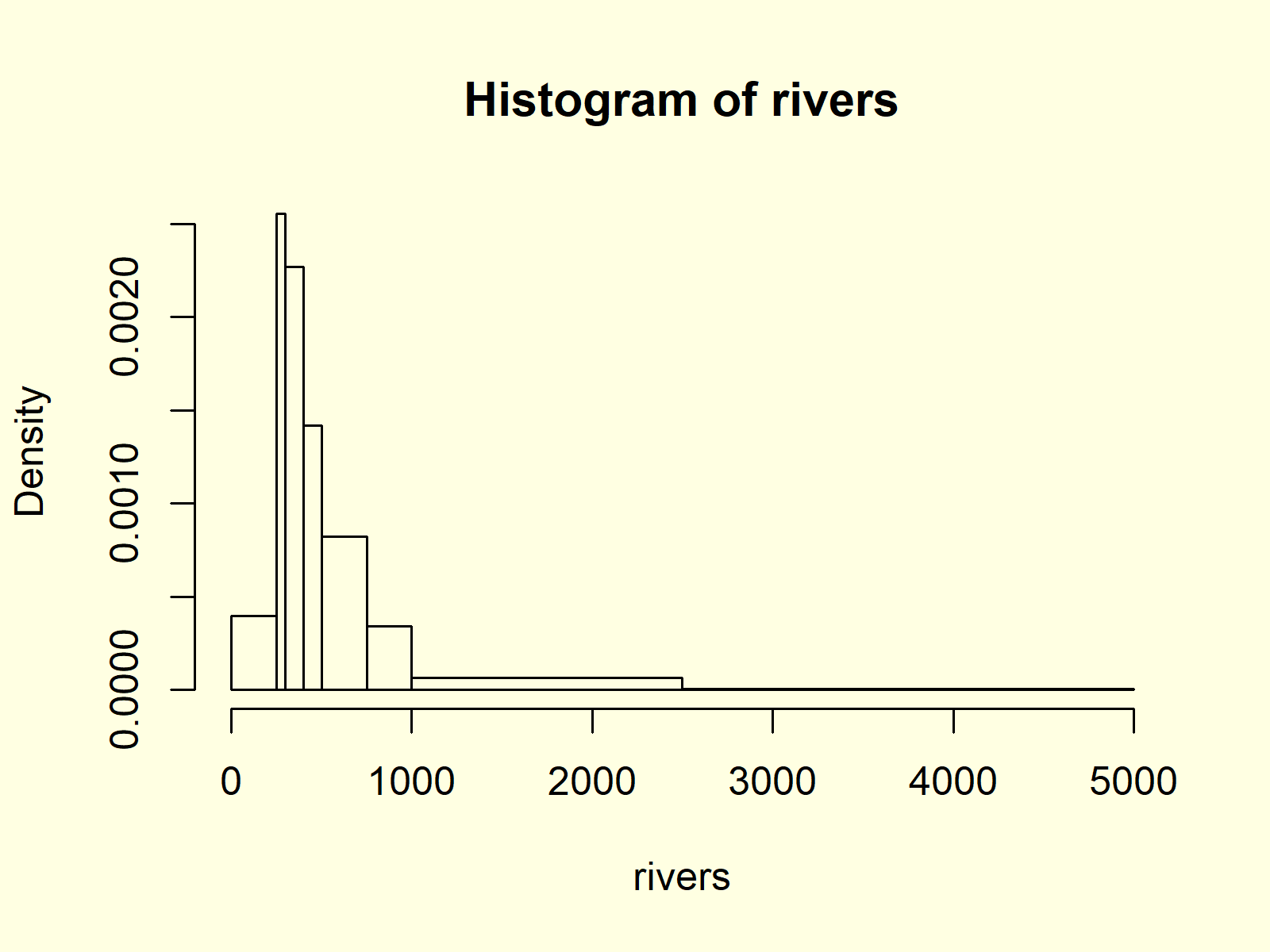
|  |
| --- |
| hist(x) # Draw histogram in R |

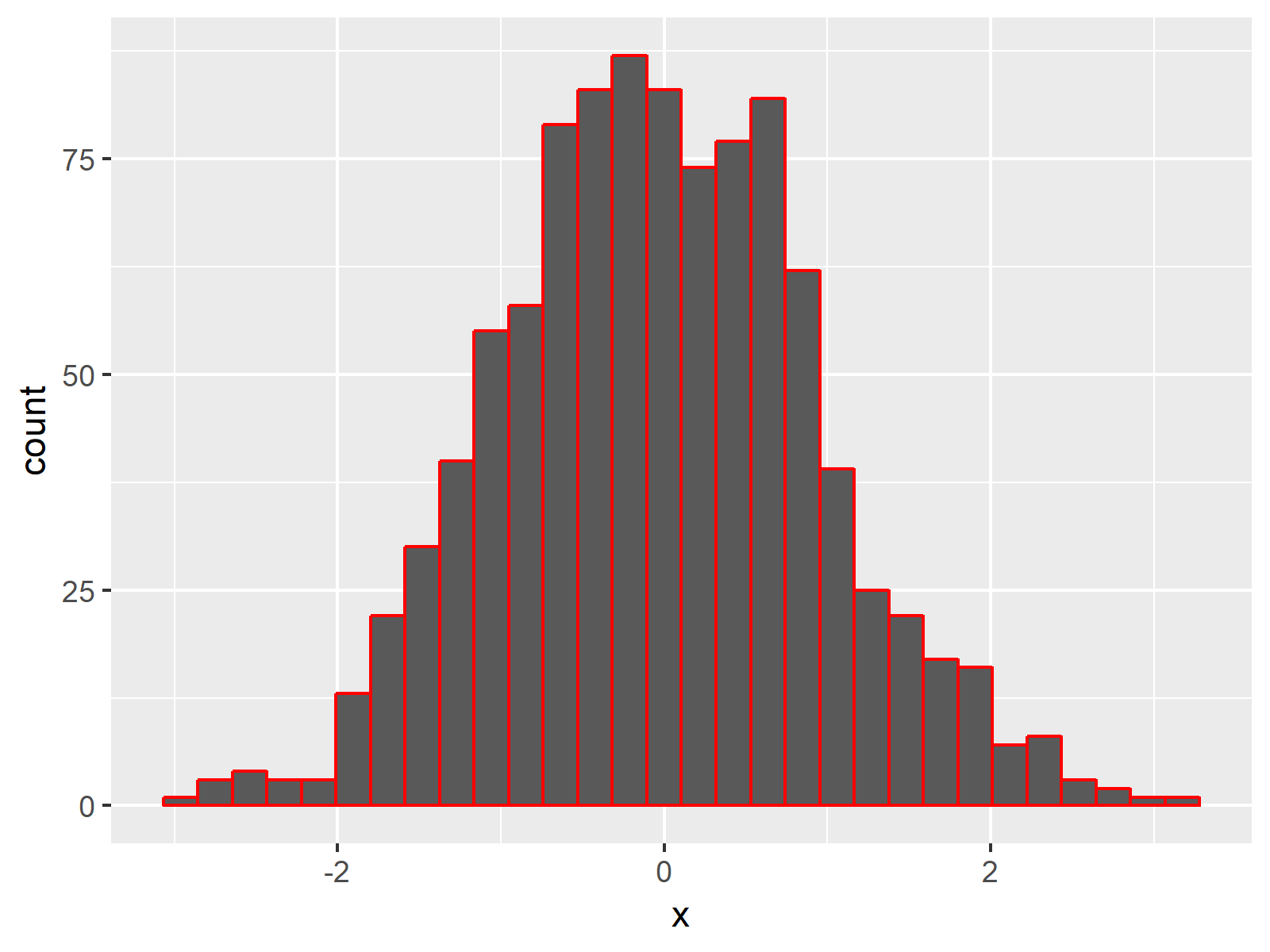


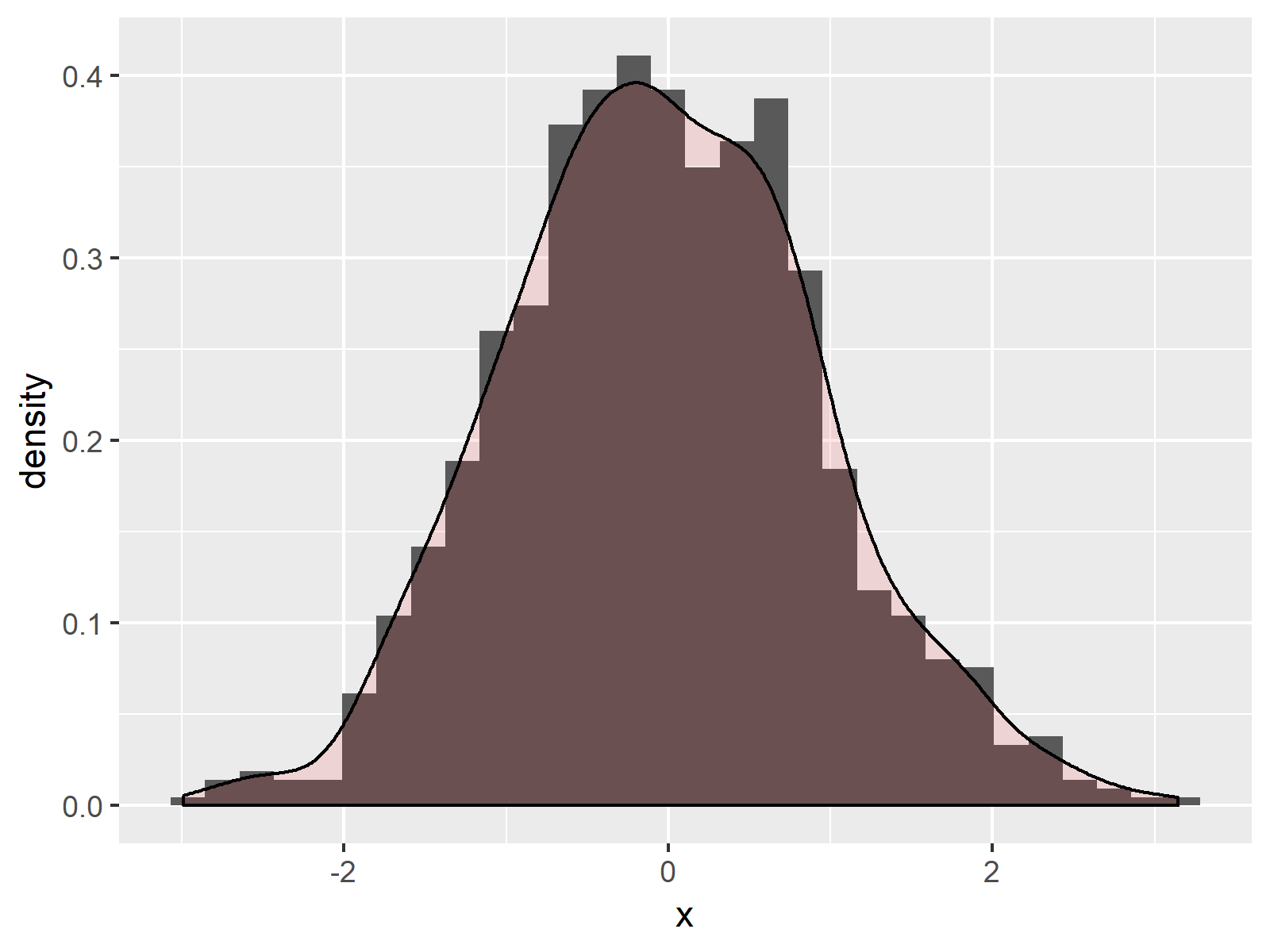
**Advanced Histograms:** Find some advanced histograms below. Click on the images to get more information and example R codes for each of the histograms.

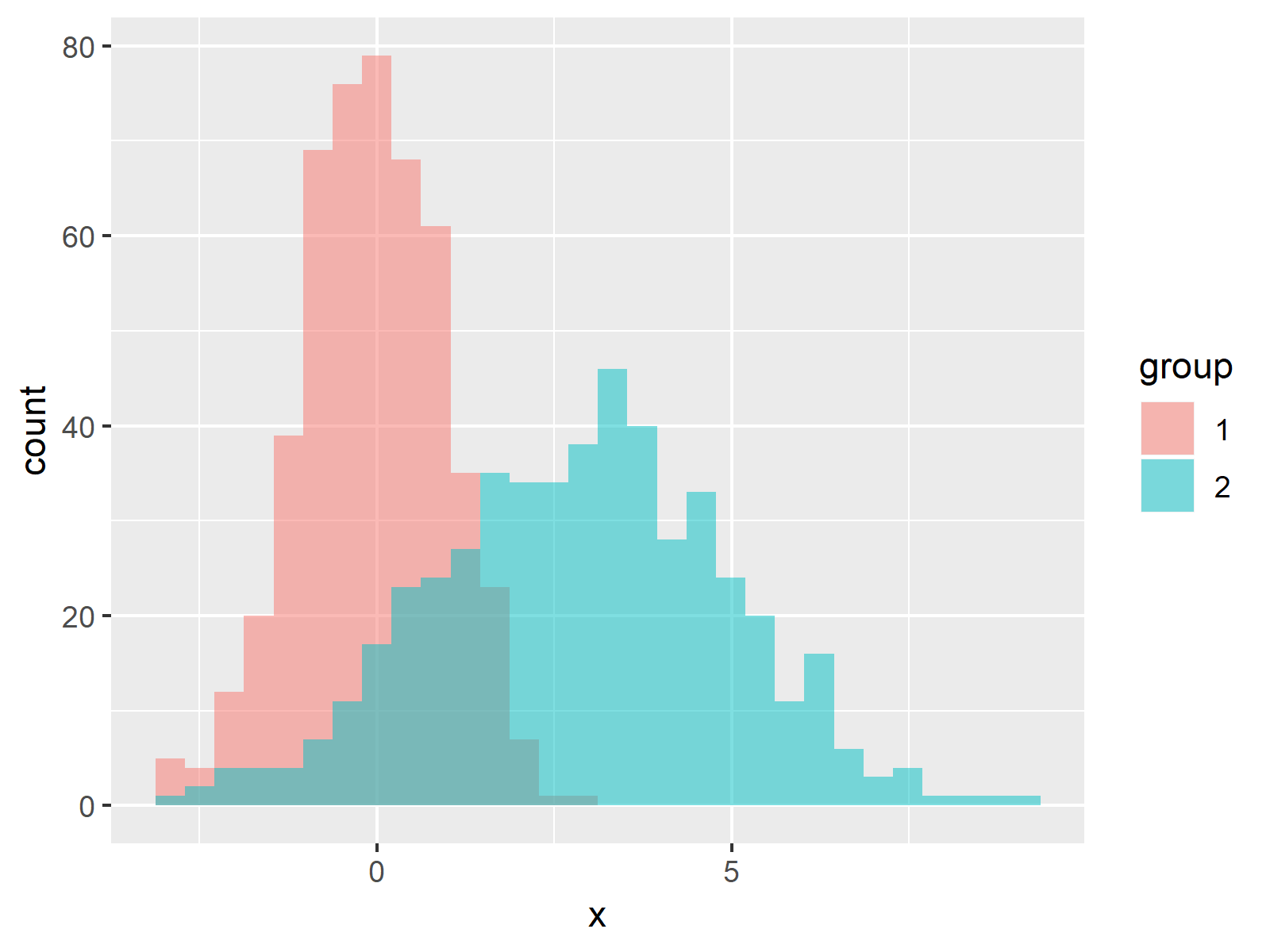
[](https://statisticsglobe.com/histogram-in-base-r-hist-function#third)

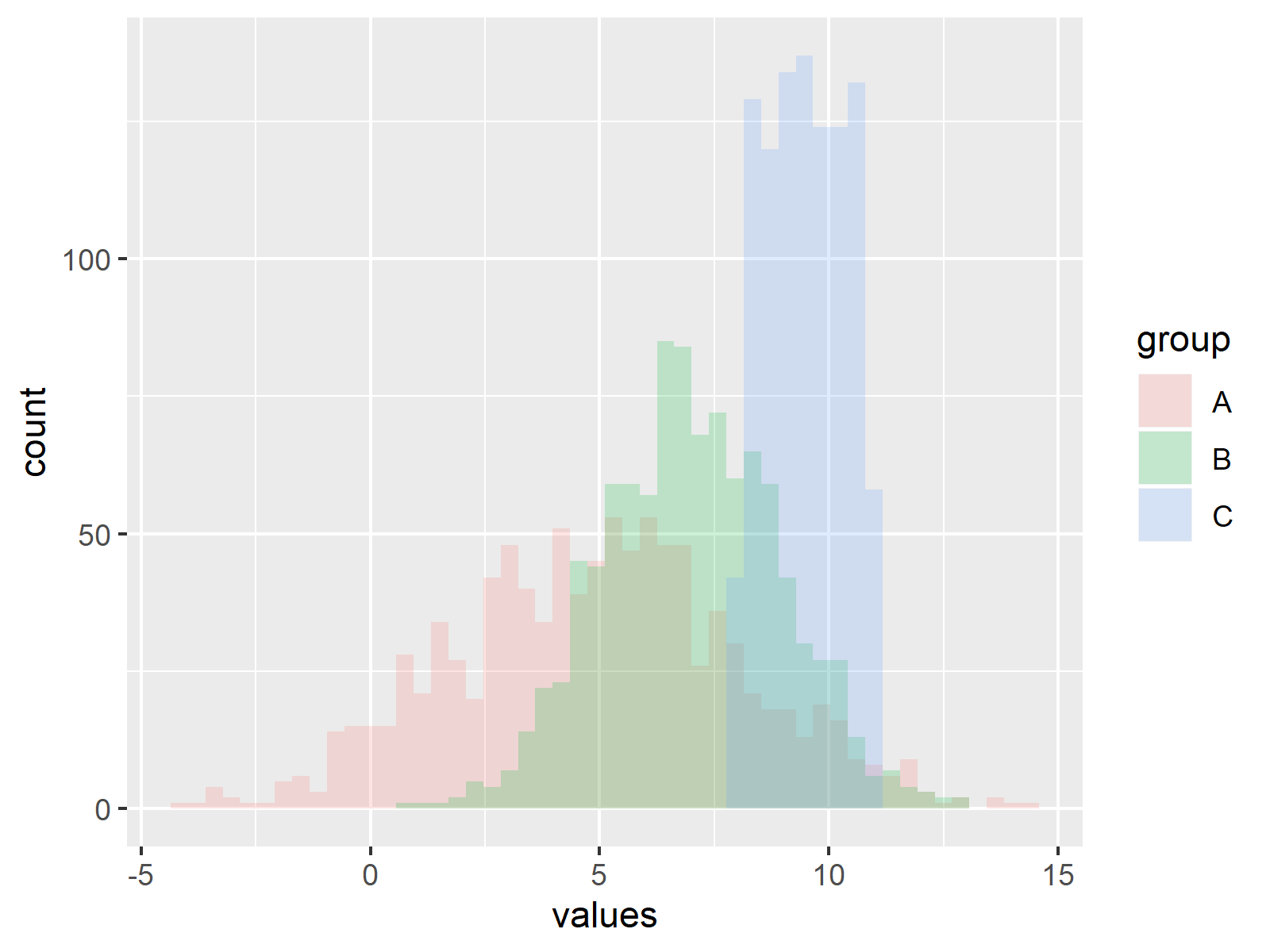
[](https://statisticsglobe.com/histogram-in-base-r-hist-function#seventh)

[](https://statisticsglobe.com/histogram-in-base-r-hist-function#fifth)

[](https://statisticsglobe.com/ggplot2-histogram-in-r-geom_histogram-function#third)

[](https://statisticsglobe.com/ggplot2-histogram-in-r-geom_histogram-function#sixth)

[](https://statisticsglobe.com/ggplot2-histogram-in-r-geom_histogram-function#seventh)

[](https://statisticsglobe.com/draw-overlaying-histograms-with-ggplot2-in-r)

**Histogram Resources:** Find some further resources on the creation of histograms below.

## Example Data

In the examples of this R tutorial, we’ll use the [rivers data set](https://stat.ethz.ch/R-manual/R-patched/library/datasets/html/rivers.html). The rivers data set contains the length in miles of 141 major rivers in North America.

The data set is already available in Base R and is stored in the numeric vector “rivers”:

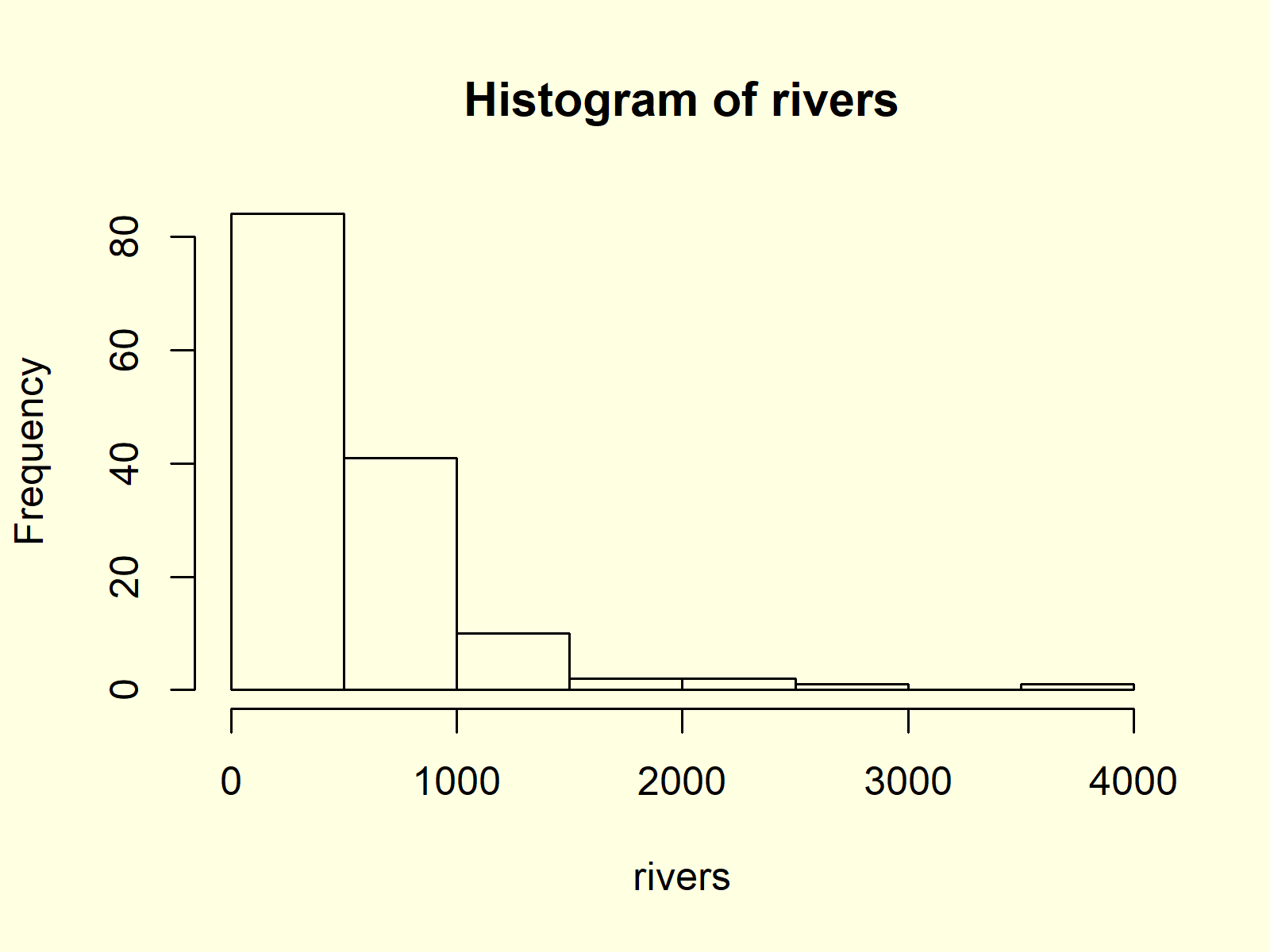
|  |
| --- |
| rivers # Inspect example data  # 735 320 325 392 524 450 1459 135 465... |

In the following examples, we’ll create different types of histograms of the rivers data. Let’s move on to the examples!

## Example 1: Default Histogram in Base R

The [Base installation of R](https://cran.r-project.org/doc/manuals/r-release/R-admin.html) provides the hist function. We can make a histogram with default specifications of the hist function as follows:

|  |
| --- |
| hist(rivers) # Default histogram |



***Figure 1: Histogram with Default Specifications.***

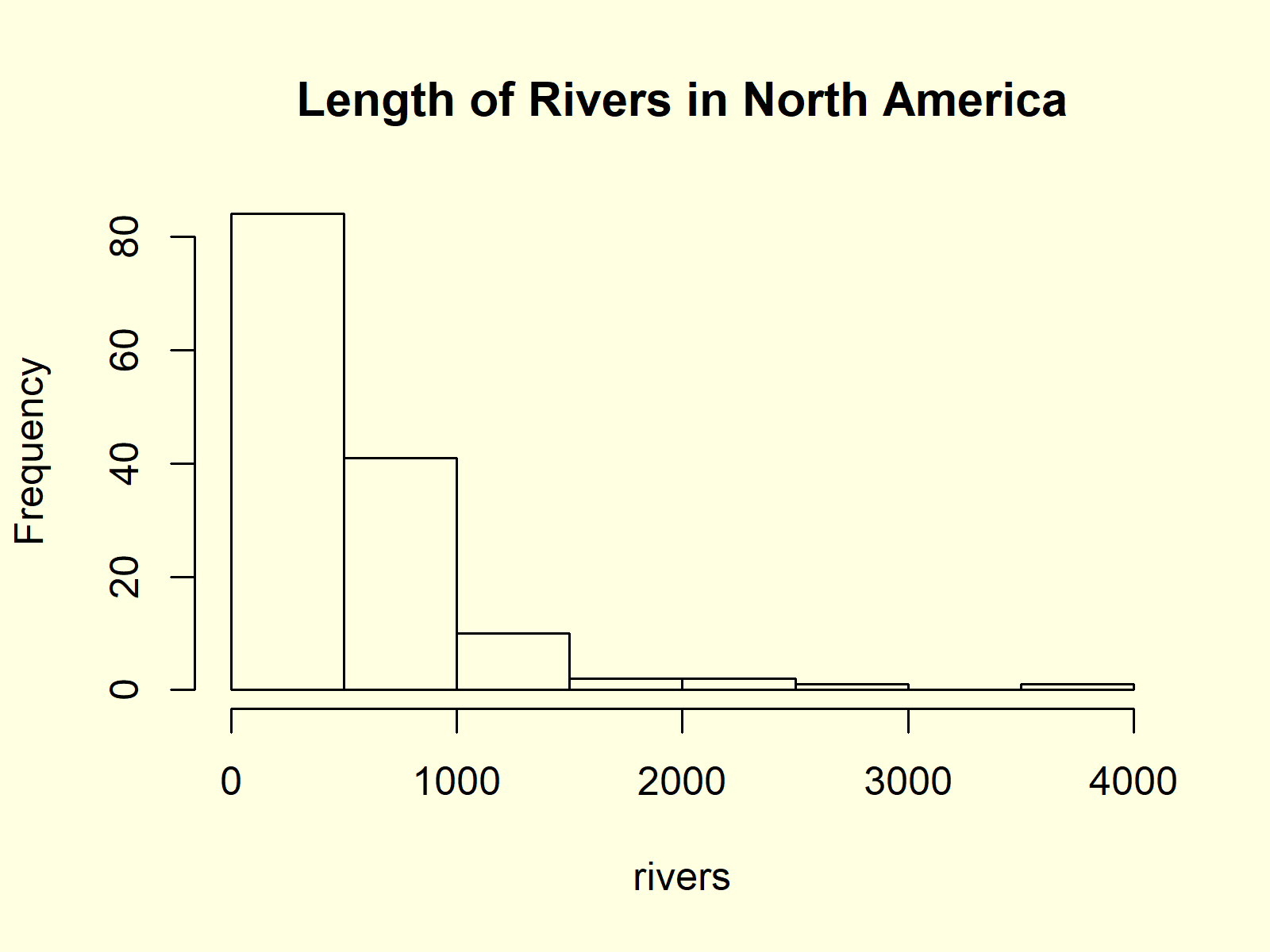
Figure 1 shows the output of the hist function: A histogram with relatively wide bars, without colors, and with automatic main titles and axis labels.

In the following you’ll learn how to modify the different components of this histogram.

## Example 2: Histogram with Manual Main Title

We can change the main title of our histogram by specifying the main argument of the hist function:

|  |
| --- |
| hist(rivers, # Change main title of histogram  main = "Length of Rivers in North America") |

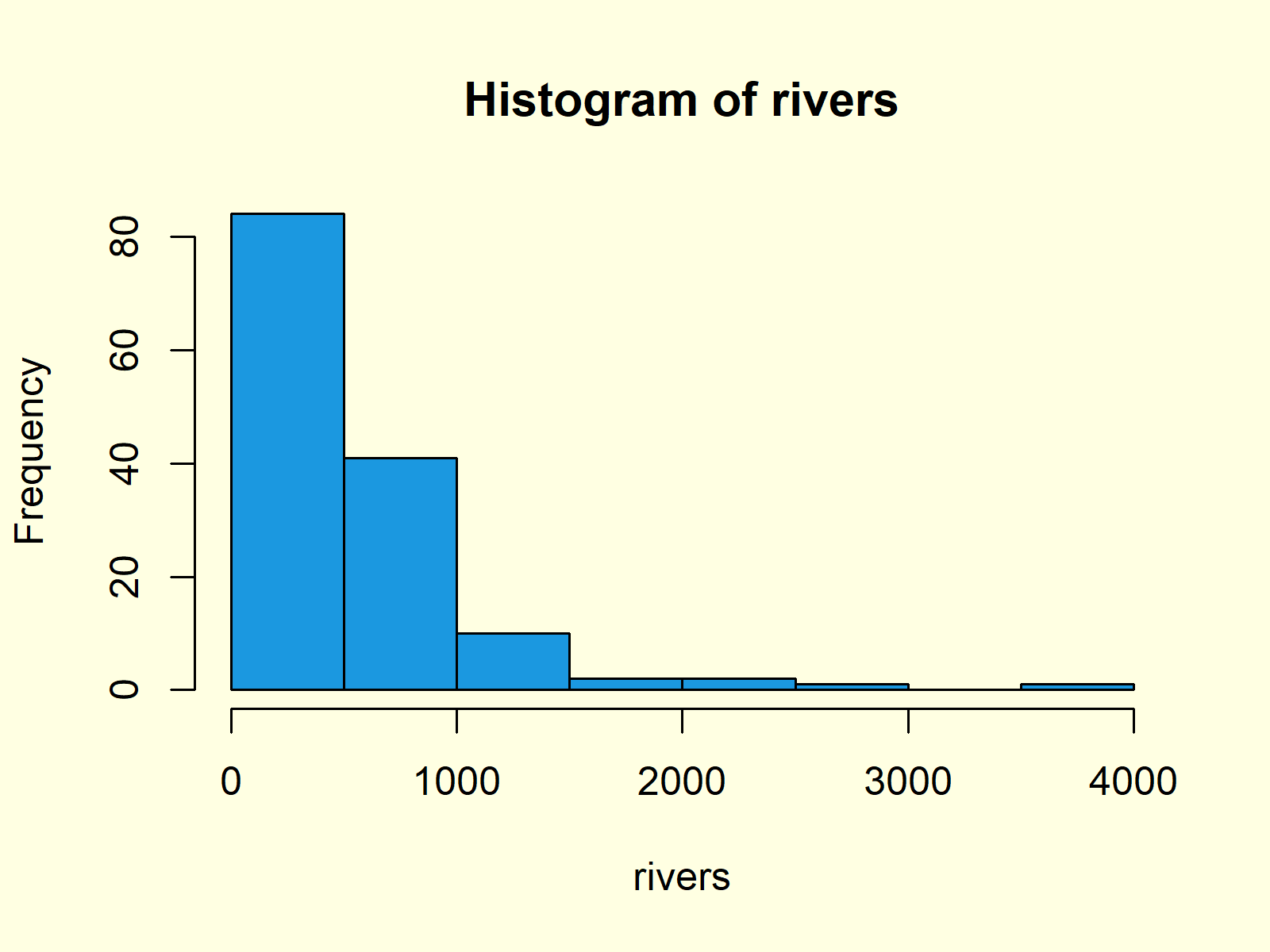


***Figure 2: Histogram with User-Defined Main Title.***

## Example 3: Histogram with Colors

If we want to color the bars of our histogram, we can use the col argument:

|  |
| --- |
| hist(rivers, # Change color of histogram  col = "#1b98e0") |

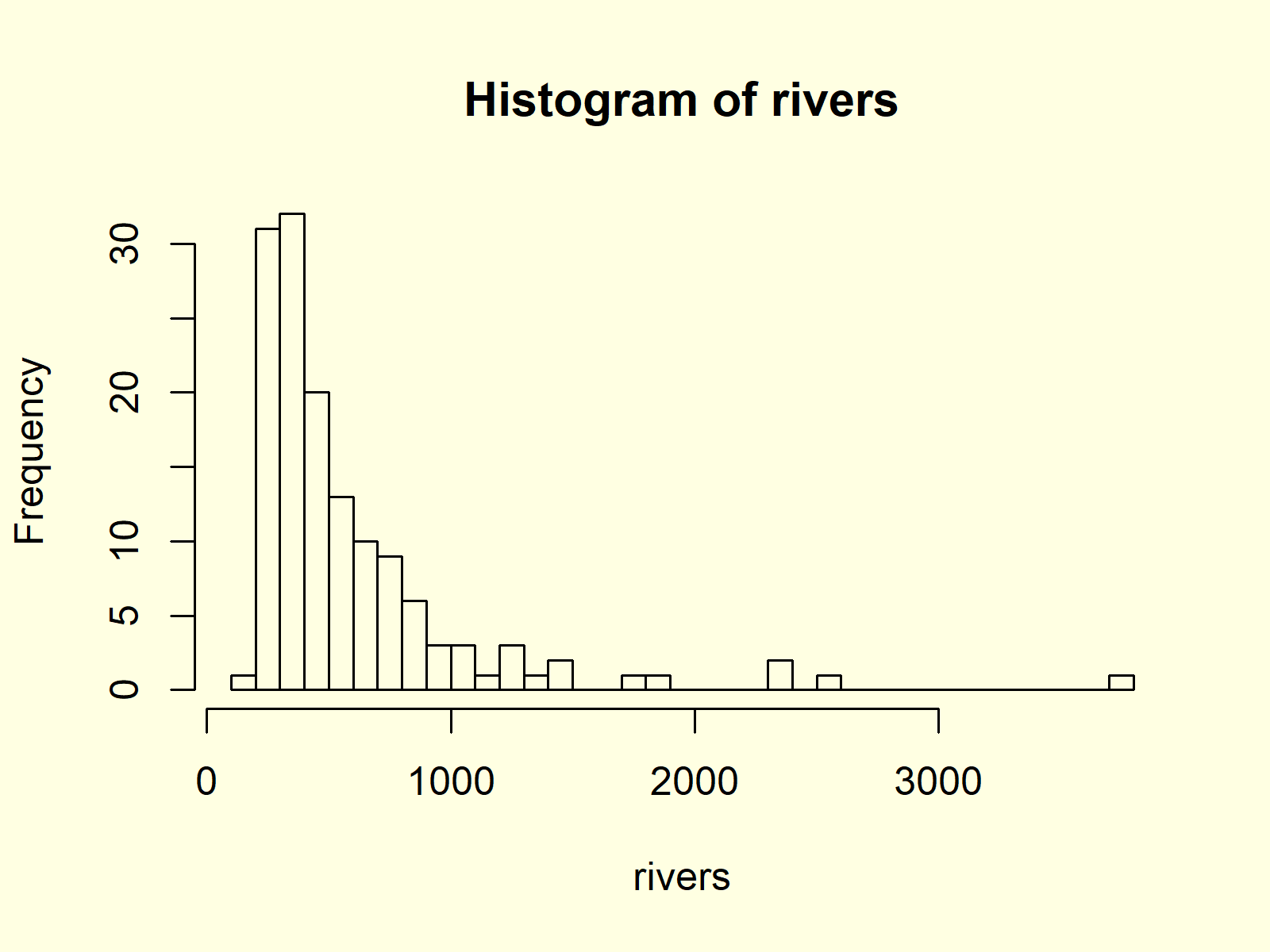


***Figure 3: Histogram with User-Defined Color.***

**Example 4: Histogram with Manual Number of Breaks**

You might have noticed that the bars of our histogram are relatively wide. We can change the width of our histogram bars with the break argument:

|  |
| --- |
| hist(rivers, # Change number of histogram breaks  breaks = 50) |



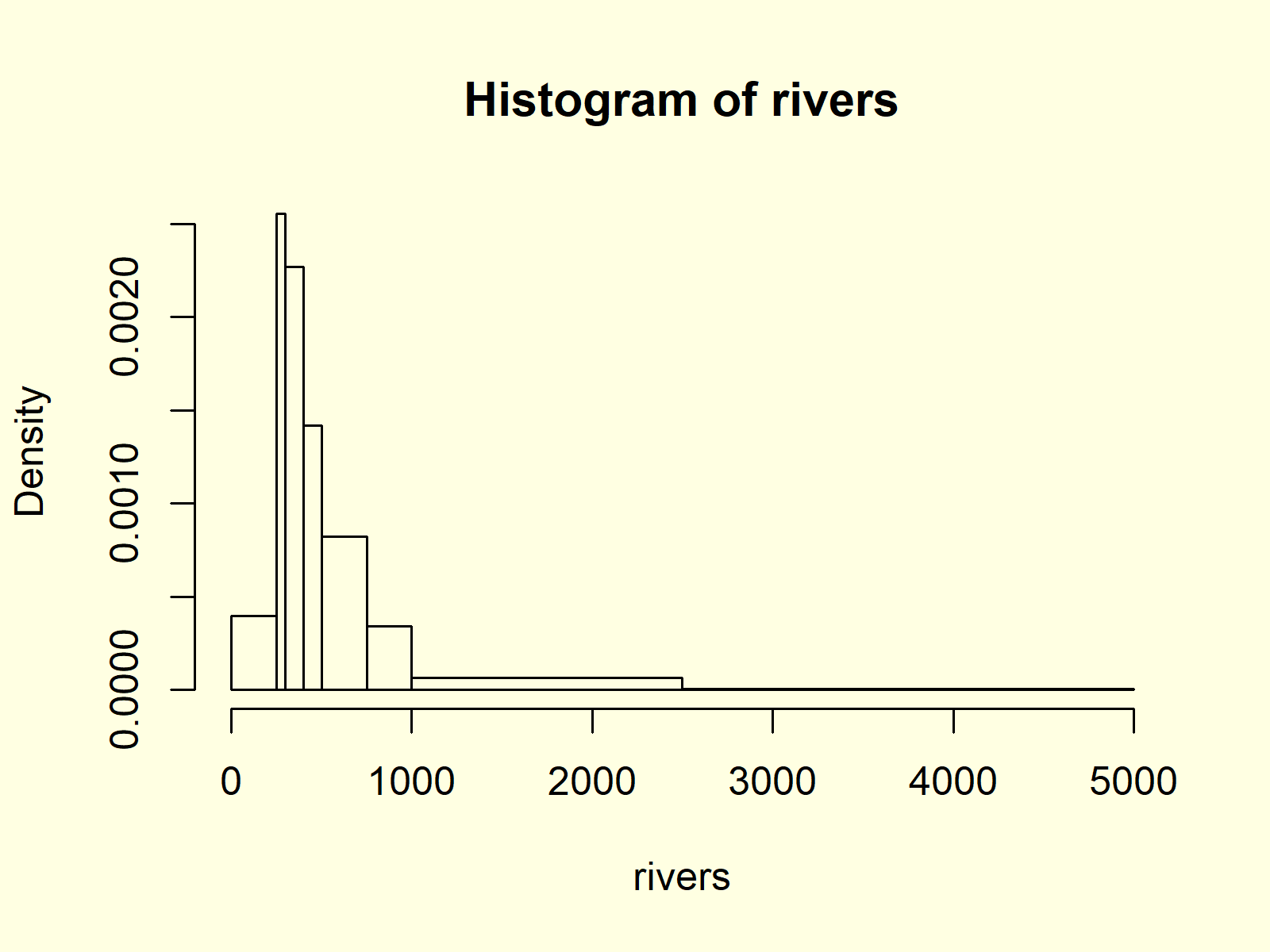
***Figure 4: Histogram with More Breaks.***

The higher the number of breaks, the smaller are the bars.

## Example 5: Histogram with Non-Uniform Width

In Example 4, you learned how to change the number of bars within a histogram by specifying the break argument. However, we can also use the break argument to draw a histogram showing bars with a different width. Consider the following R code:

|  |
| --- |
| hist(rivers,  # Specify fixed breaks with different width  breaks = c(0, 250, 300, 400, 500, 750, 1000, 2500, 5000)) |



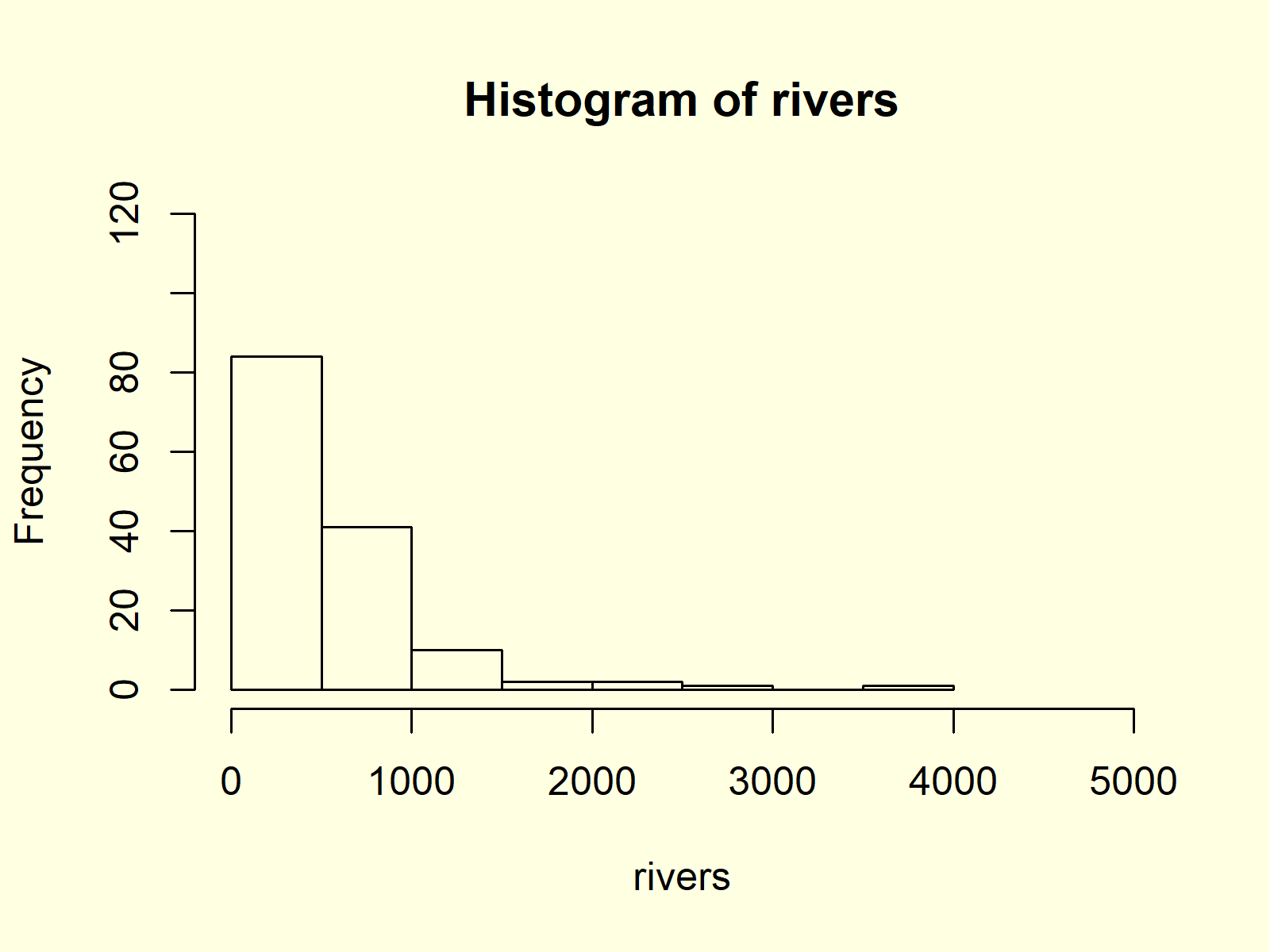
***Figure 5: Histogram with Non-Unified Breaks.***

As you can see based on Figure 5, each bar of our histogram has a different width.

## Example 6: Histogram with Manual Axis Limits

It is also possible to modify the width and height of the Y- and X-axes of our histogram by specifying the xlim and ylim options.

|  |
| --- |
| hist(rivers, # Change axis limits of histogram  xlim = c(0, 5000),  ylim = c(0, 120)) |



***Figure 6: Histogram with User-Defined Axis Limits of Y- & X-Axes.***

In the previous R syntax, we specified the x-axis limits to be 0 and 5000 and the y-axis limits to be 0 and 120.

## Example 7: Histogram with Overlaid Density Line

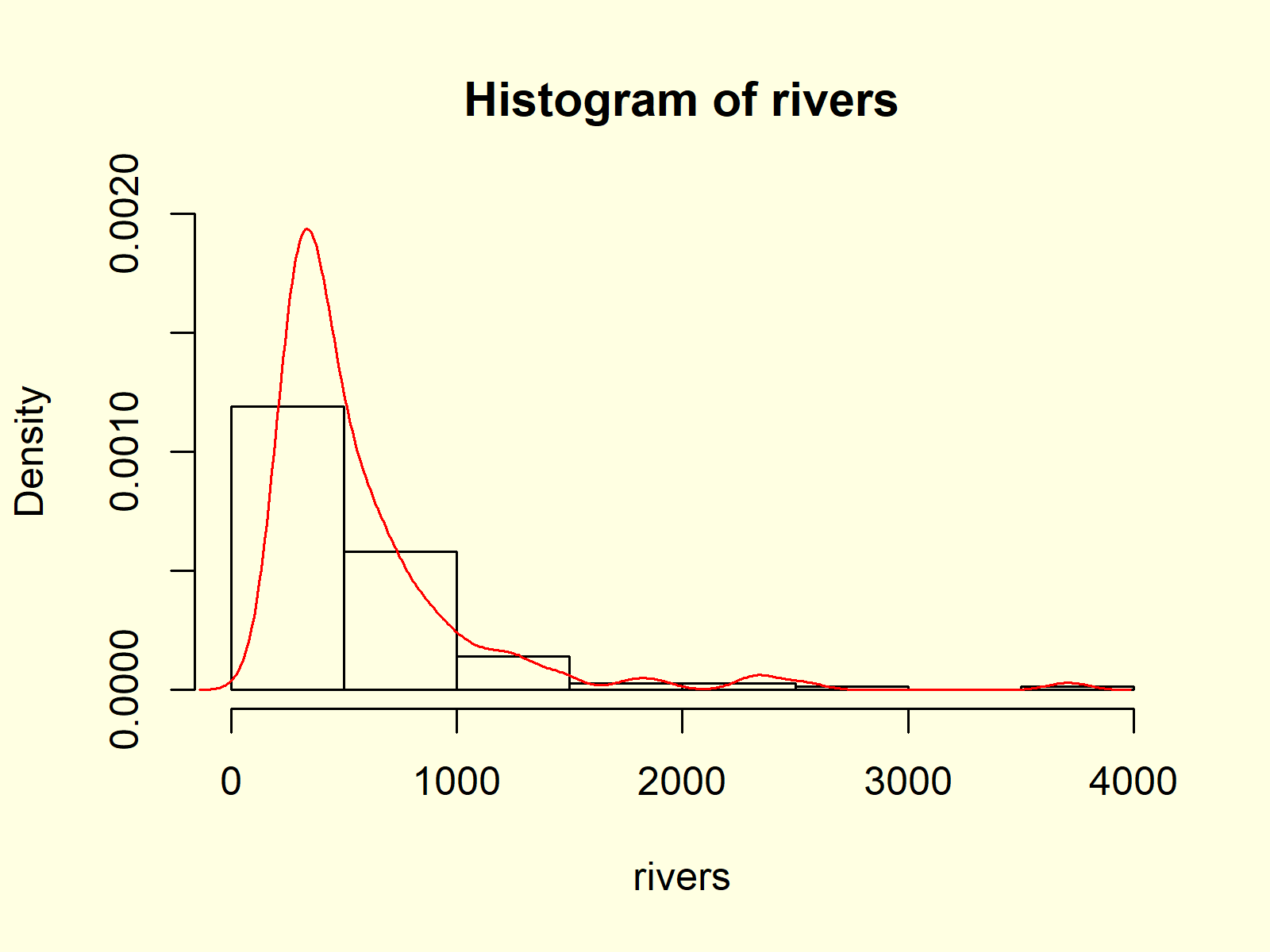
Density Plots are a smoother representation of numeric data than histograms. Sometimes it makes sense to plot the density and the histogram of numeric data [in the same plot window](https://statisticsglobe.com/draw-two-graphs-in-same-plot-in-r).

First, we have to create a histogram by specifying the prob argument to be equal to TRUE. Furthermore, it often makes sense to increase the upper y-axis limit, since the density plot may be cut off otherwise.

|  |
| --- |
| hist(rivers, # Draw histogram with probability  ylim = c(0, 0.002),  prob = **TRUE**) |

After drawing this histogram, we can apply a combination of the lines() and density() functions to overlay our histogram with a density line:

|  |
| --- |
| lines(density(rivers), col = "red") # Overlay density on histogram |



***Figure 7: Histogram & Density in One Plot.***

Figure 7 shows the output after running the whole R code of Example 7.

## Example 8: Histogram with Values on Top of Bars

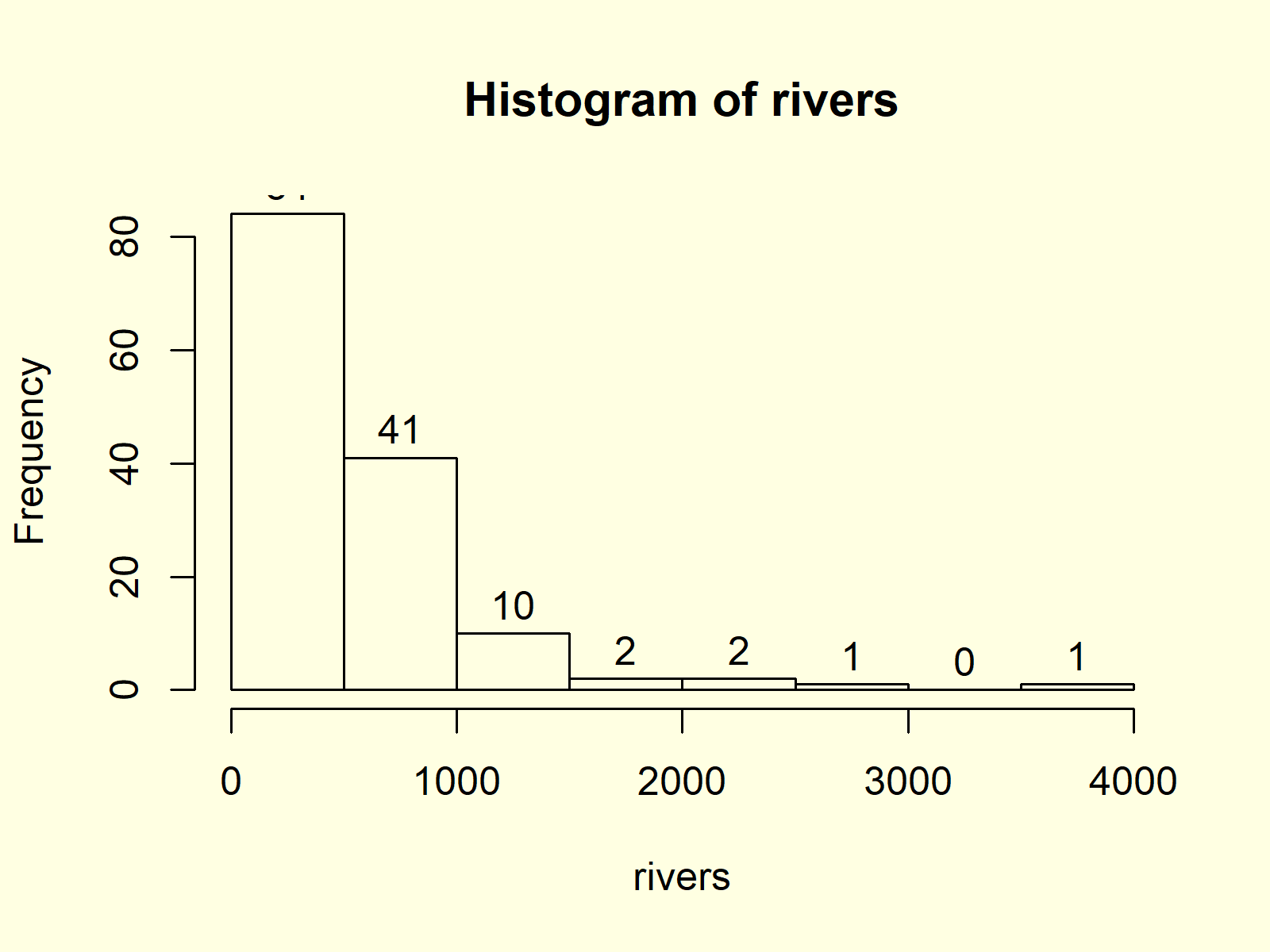
The hist command can also be used to extract the values of our histogram. Have a look at the following R syntax:

|  |
| --- |
| hist\_values <- hist(rivers) # Store values of histogram  hist\_values # Print values to RStudio console  # $breaks  # [1] 0 500 1000 1500 2000 2500 3000 3500 4000  #  # $counts  # [1] 84 41 10 2 2 1 0 1  #  # $density  # [1] 1.191489e-03 5.815603e-04 1.418440e-04 2.836879e-05 2.836879e-05 1.418440e-05 0.000000e+00 1.418440e-05  #  # $mids  # [1] 250 750 1250 1750 2250 2750 3250 3750  #  # $xname  # [1] "rivers"  #  # $equidist  # [1] TRUE  #  # attr(,"class")  # [1] "histogram" |

As you can see based on the RStudio console output, the hist function returns a lot of information on our histogram, i.e. breaks, counts, density, mids, xname, equidist, and attr. You may have a look at the [help documentation of the hist function](https://www.rdocumentation.org/packages/graphics/versions/3.6.1/topics/hist) to learn more about these information.

However, we’ll use only the mids and the counts of our histogram in this example:

|  |
| --- |
| text(hist\_values$mids, # Add values of histogram on top of bars  hist\_values$counts,  labels = hist\_values$counts,  adj = c(0.5, - 0.5)) |



***Figure 8: Histogram with Values of Bars on Top.***

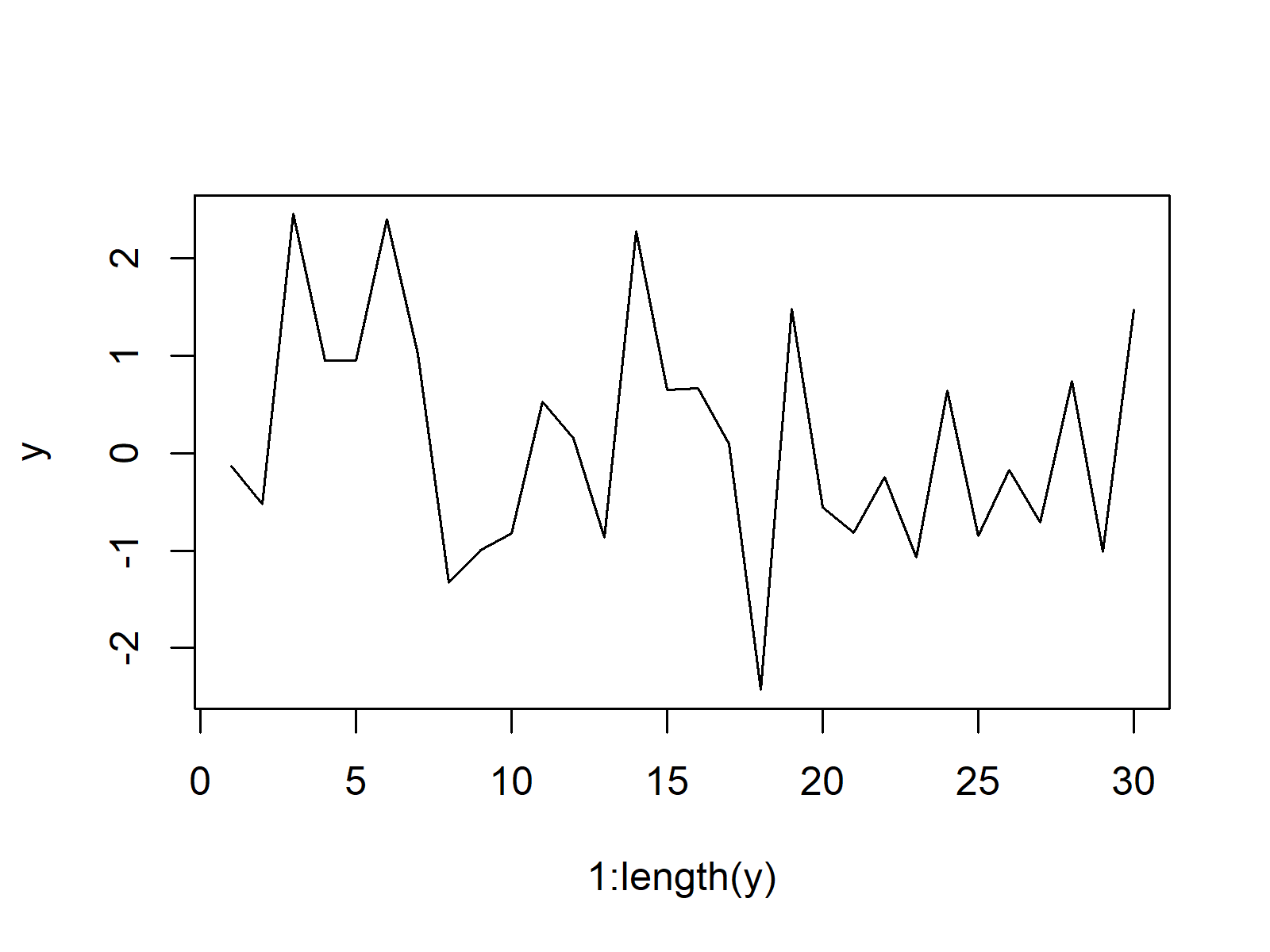
Figure 8 illustrates the resulting histogram. As you can see, we added the counts at the top of each bar.

1. **Line Plot**

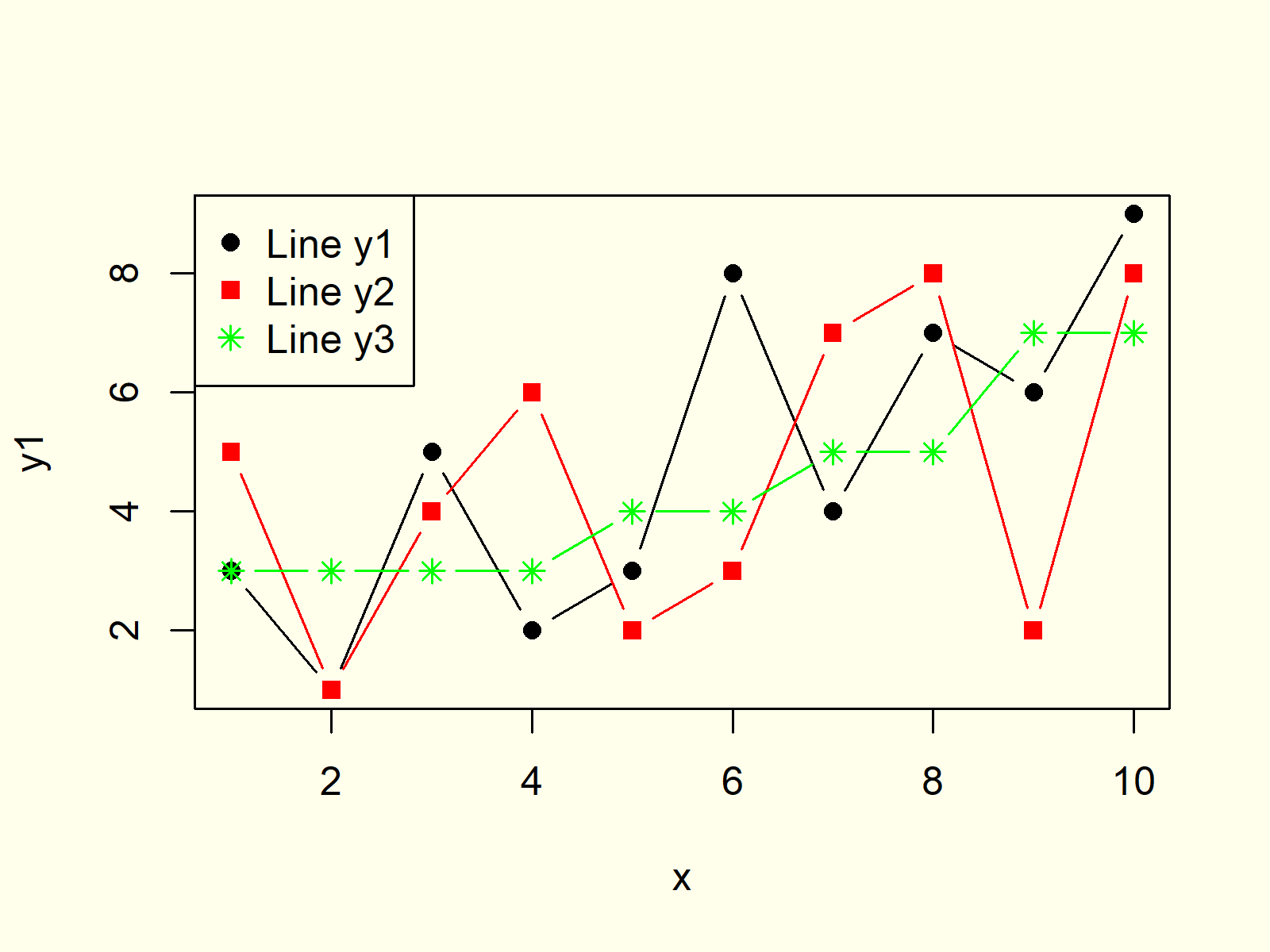
**Line Plot Definition:** A [line plot](https://www.smartdraw.com/line-graph/) (or line graph; line chart) visualizes values along a sequence (e.g. over time). Line plots consist of an x-axis and a y-axis. The x-axis usually displays the sequence and the y-axis the values corresponding to each point of the sequence.

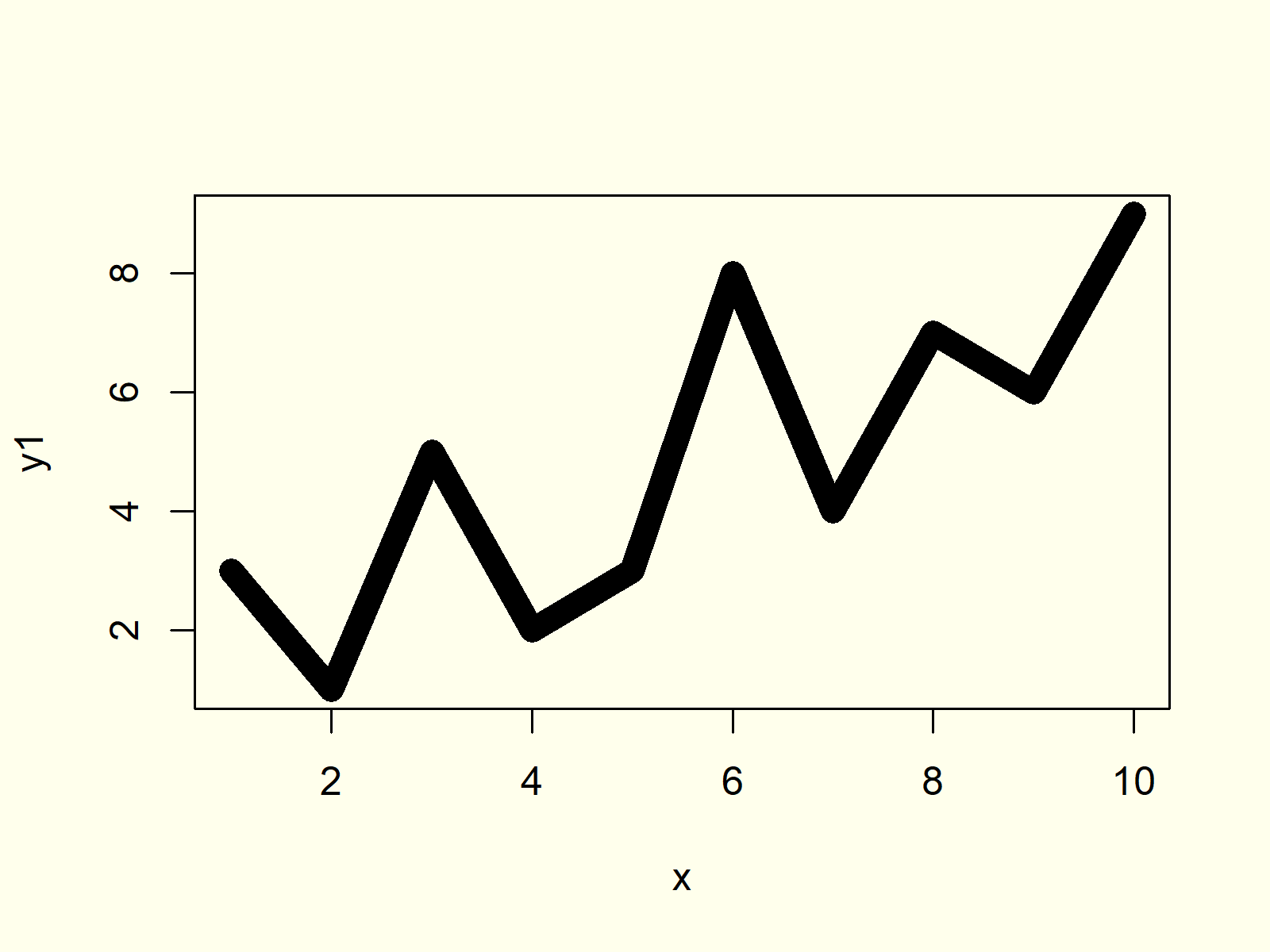
The following R syntax shows how to draw a basic line plot in R:

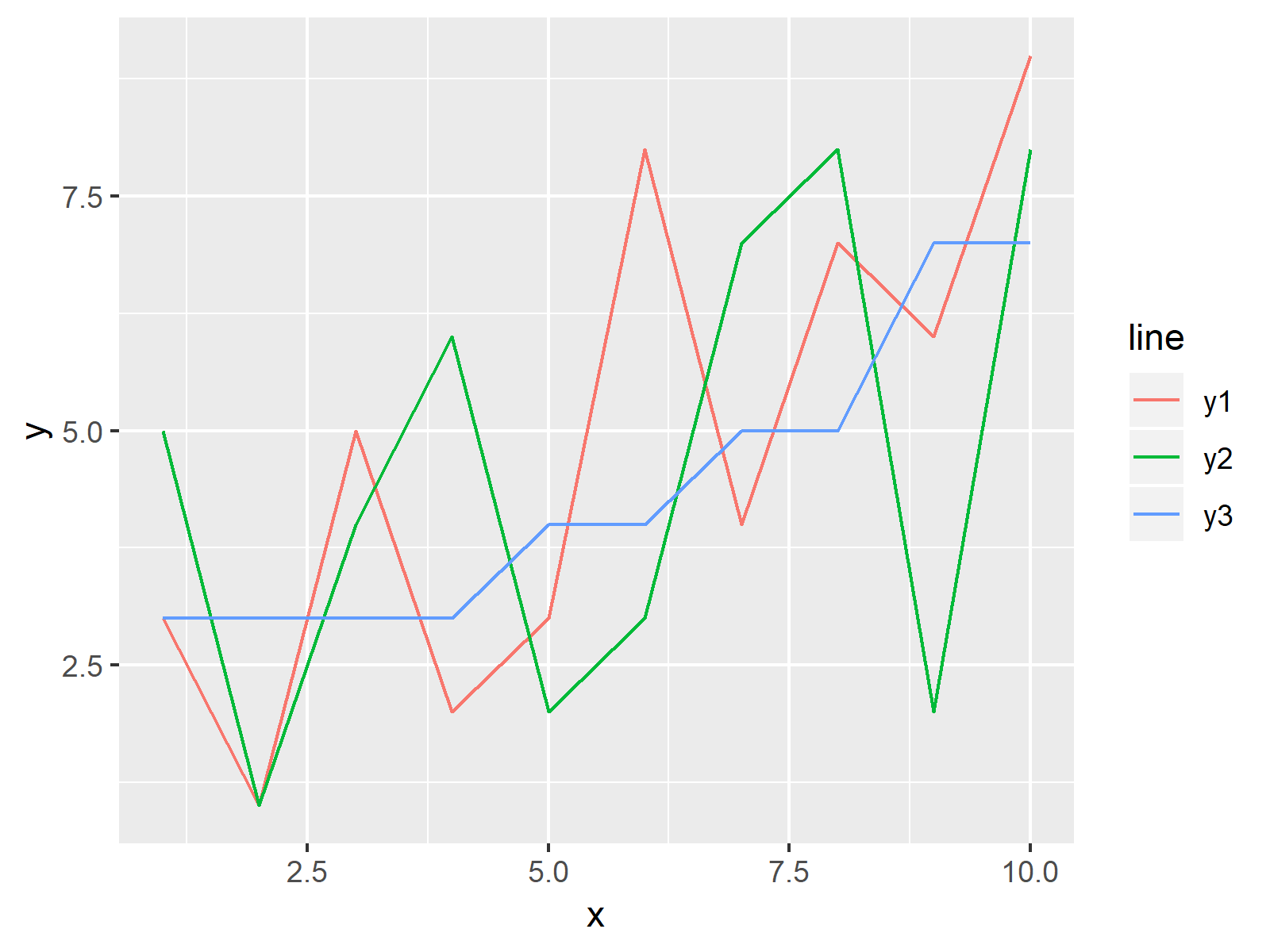
|  |
| --- |
| plot(1:length(y), y, type = "l")  # Draw line plot in R |

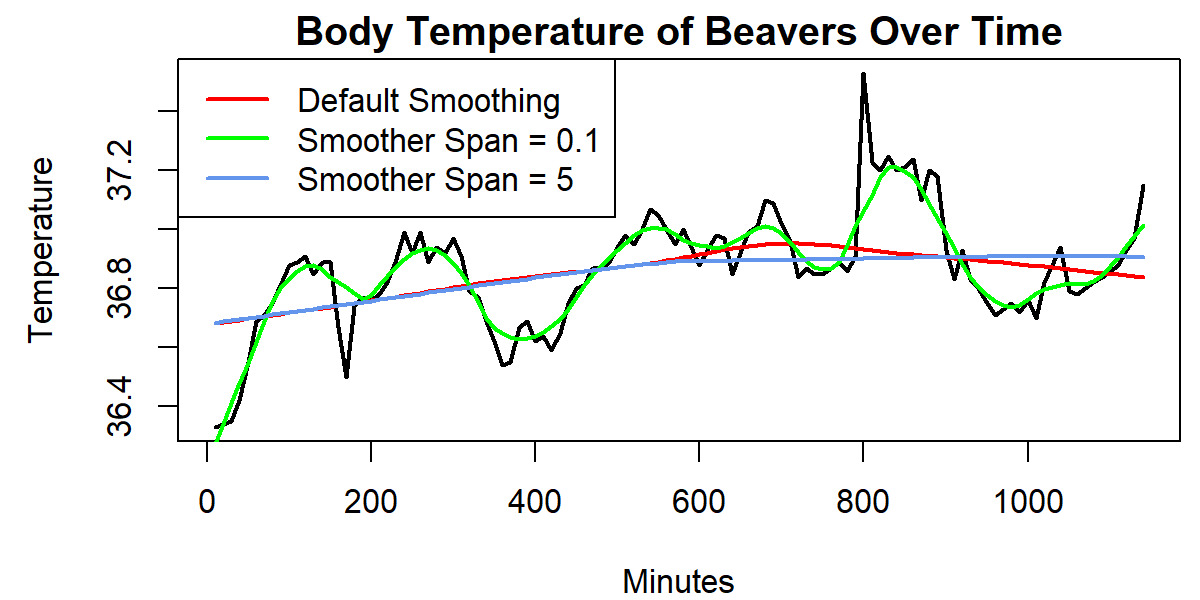


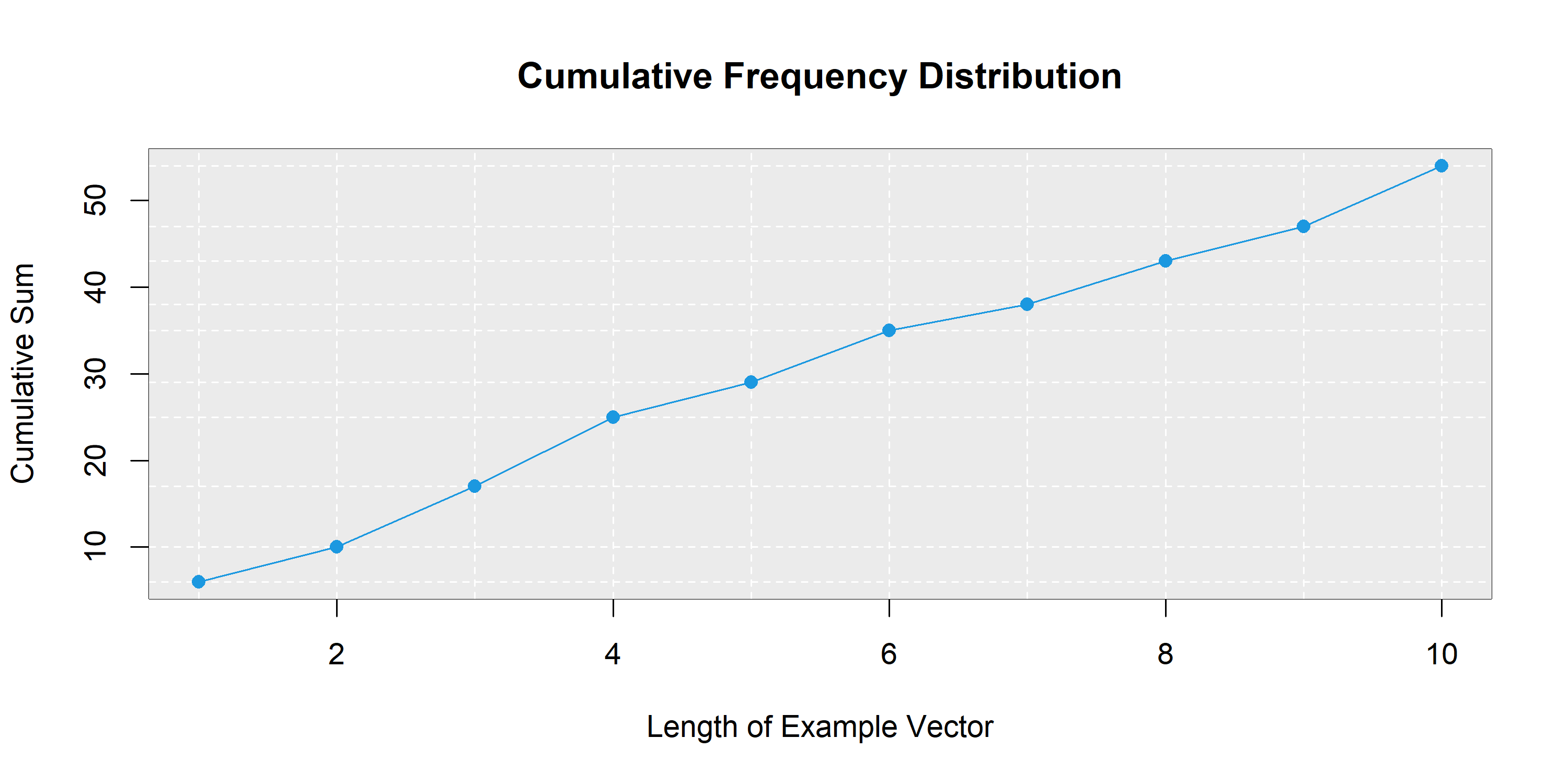
**Advanced Line Plots:** Find some advanced line plots below. Click on the images to get more information and example R codes for each of the line plots.











**Line Plot Resources:** Find some further resources on the creation of line plots below.

Creating Example Data

In the examples of this R tutorial, we’ll use the following example data:

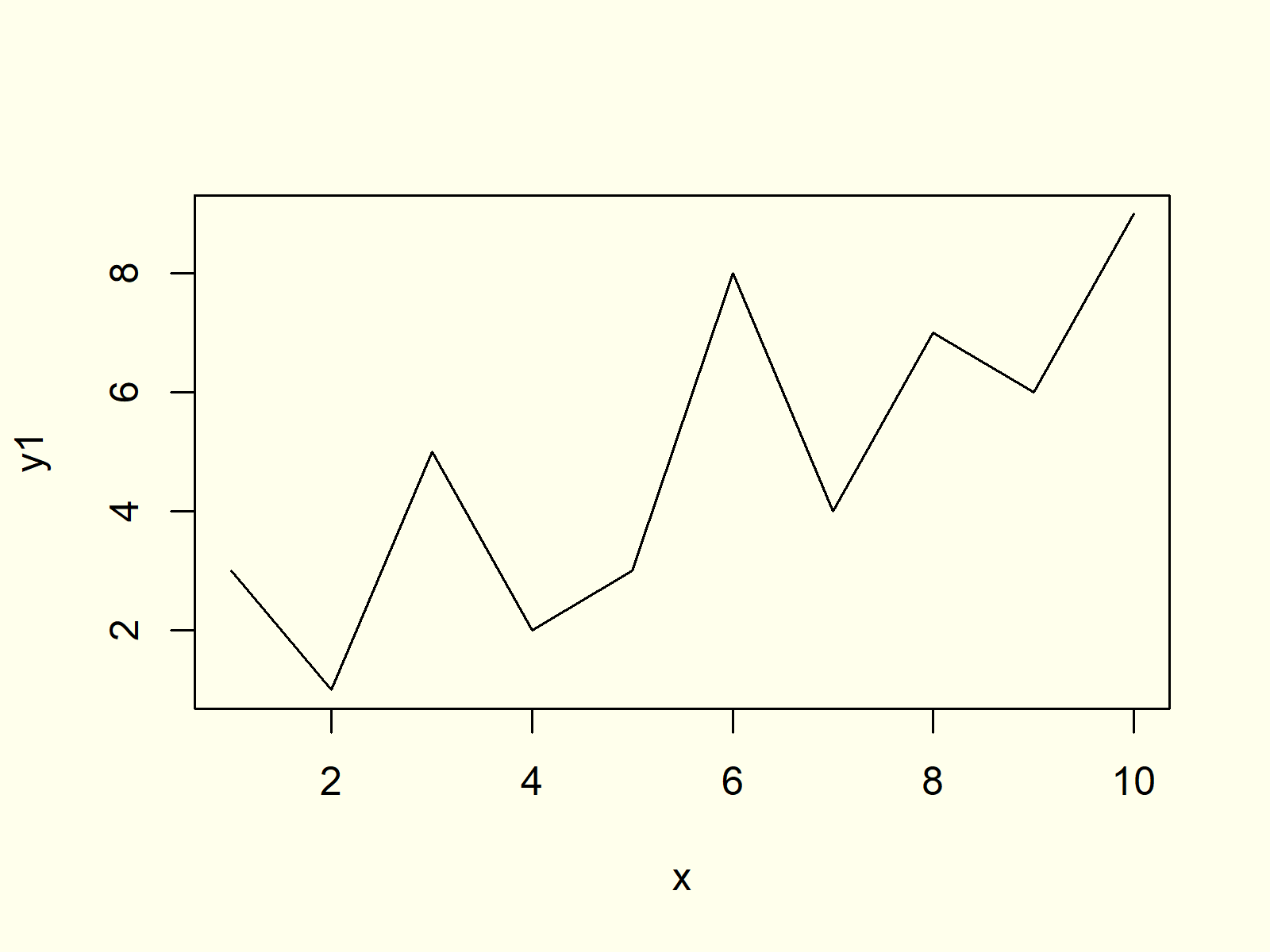
|  |
| --- |
| x <- 1:10  y1 <- c(3, 1, 5, 2, 3, 8, 4, 7, 6, 9) |

Our data consists of two [numeric vectors](https://www.math.ucla.edu/~anderson/rw1001/library/base/html/numeric.html) x and y1. The vector x contains a sequence from 1 to 10, y1 contains some random numeric values.

Example 1: Basic Creation of Line Graph in R

If we want to draw a basic line plot in R, we can use the [plot function](https://statisticsglobe.com/plot-in-r-example) with the specification type = “l”. Have a look at the following R code:

|  |
| --- |
| plot(x, y1, type = "l") |

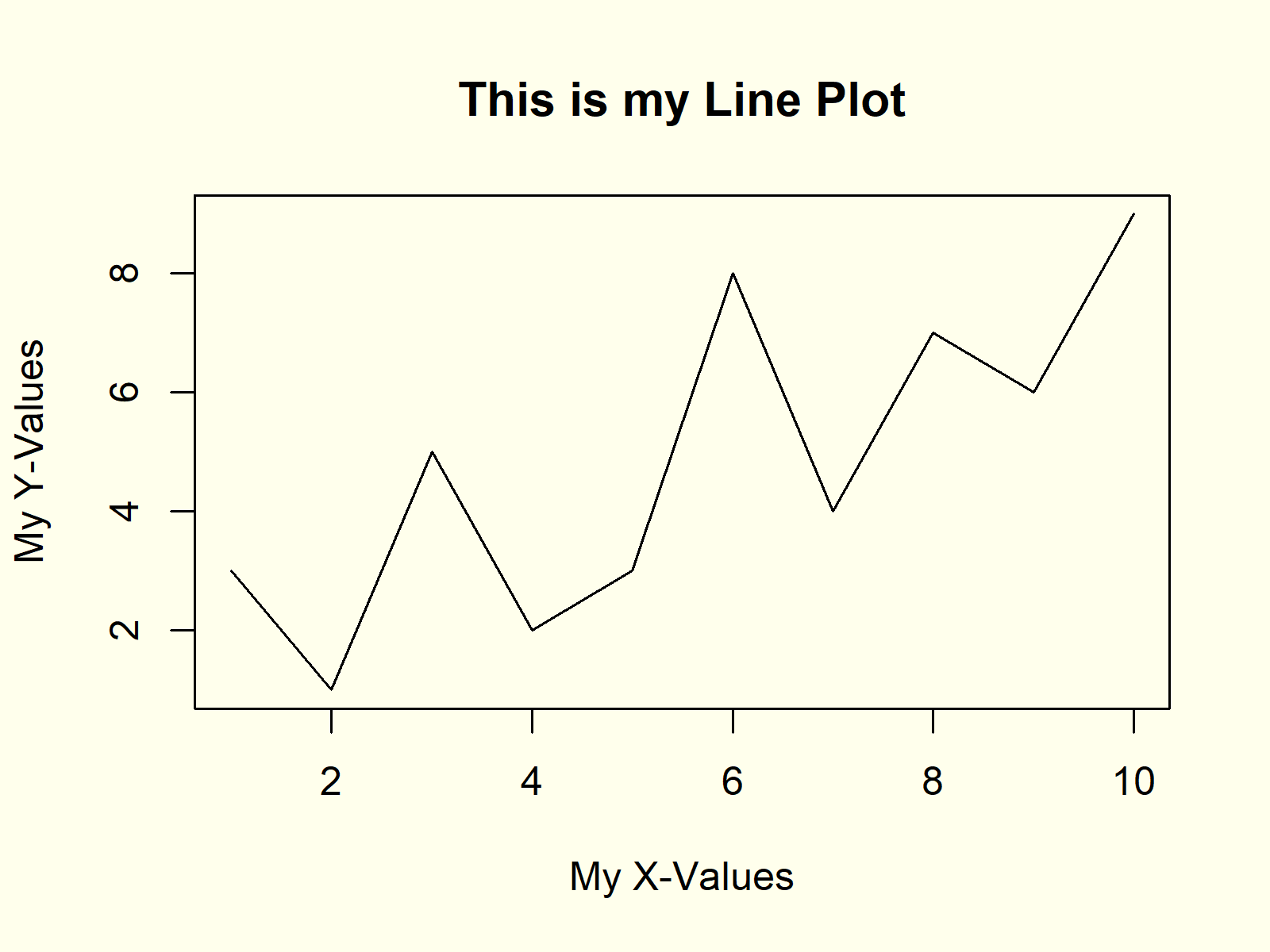


***Figure 1: Basic Line Plot in R.***

 Example 2: Add Main Title & Change Axis Labels

In Example 2, you’ll learn how to change the main title and the axis labels of our plot with the main, xlab, and ylab arguments of the plot function:

|  |
| --- |
| plot(x, y1, type = "l", # Change main title & axis labels  main = "This is my Line Plot",  xlab = "My X-Values",  ylab = "My Y-Values") |



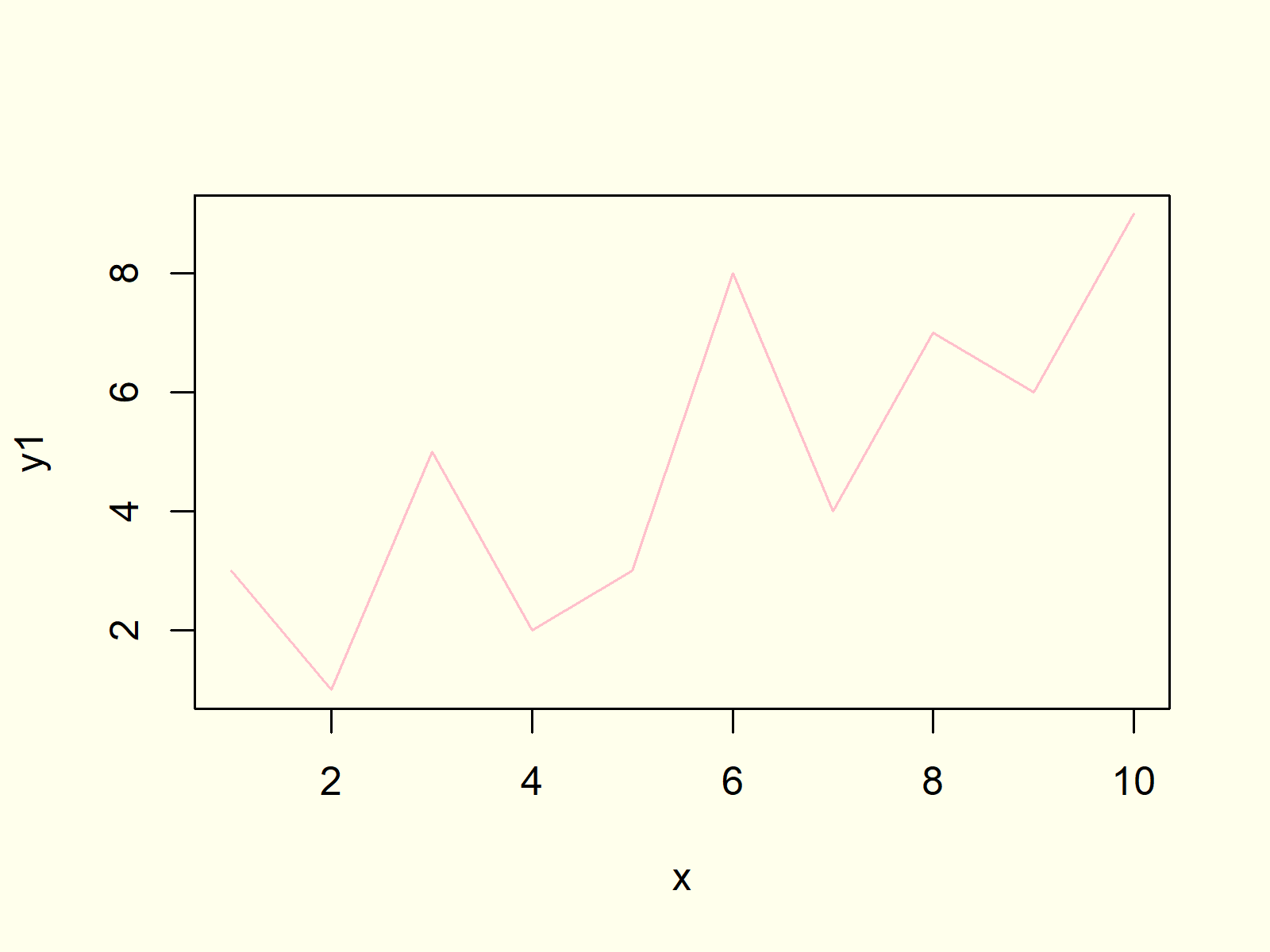
***Figure 2: Manual Main Title & Axis Labels.***

Have a look at Figure 2: Our new plot has the main title “This is my Line Plot”, the x-axis label “My X-Values”, and the y-axis label “My Y-Values”.

Example 3: Change Color of Line

We can also adjust the color of our line by using the col argument of the plot command:

|  |
| --- |
| plot(x, y1, type = "l", # Change color of line  col = "pink") |

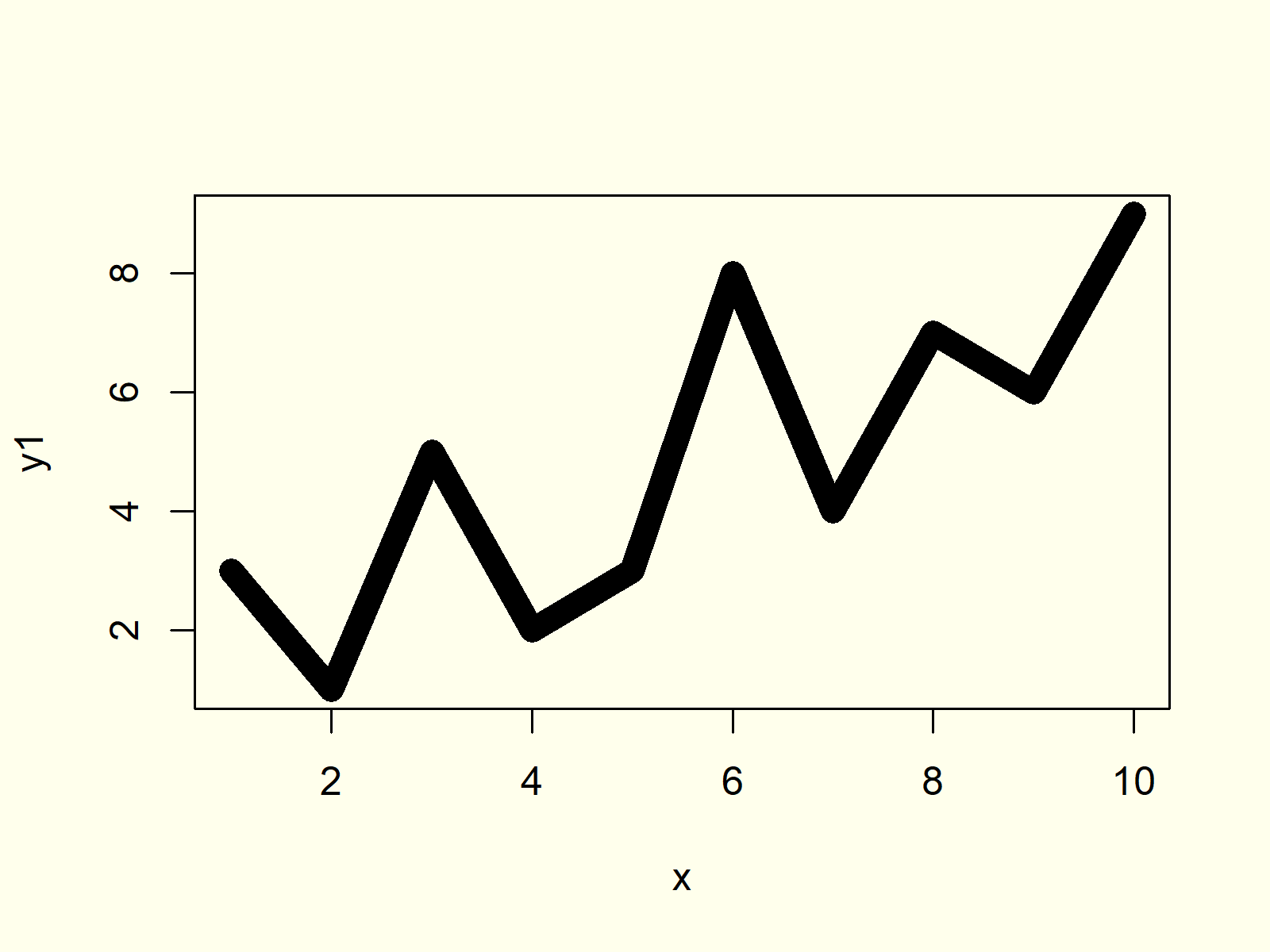


***Figure 3: Changed Color of Line Graph.***

Example 4: Modify Thickness of Line

We can increase or decrease the thickness of the lines of a line graphic with the lwd option as follows:

|  |
| --- |
| plot(x, y1, type = "l", # Change thickness of line  lwd = 10) |



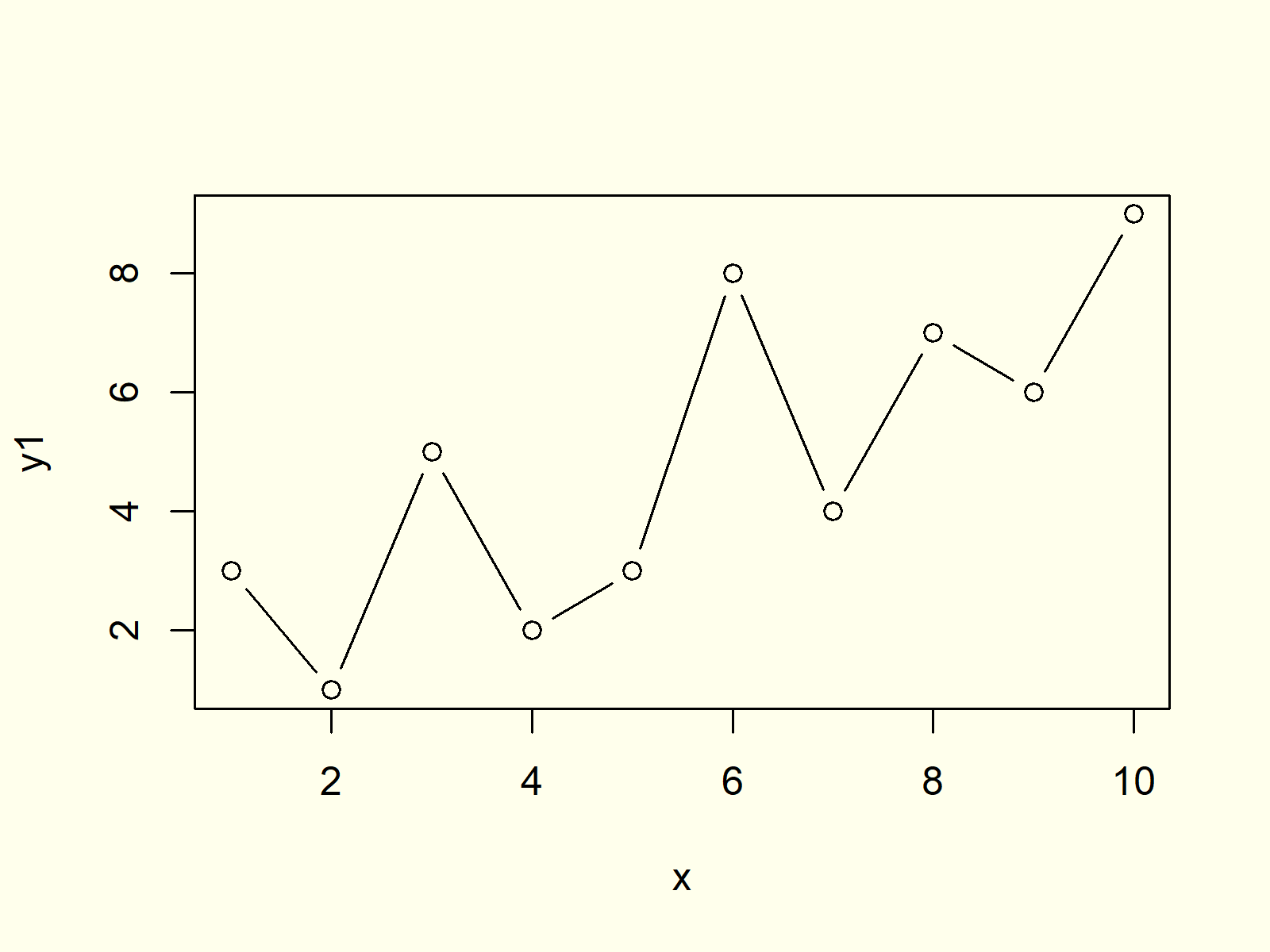
***Figure 4: User-Defined Thickness of Lines.***

In this example, we used an lwd of 10. By increasing this number, the thickness is getting larger, and by decreasing this number the line is becoming thinner. Note that the line thickness may also be changed, when exporting your image to your computer.

Example 5: Add Points to Line Graph

It is possible to add points to visualize the underlying data of our line plot even better. We simply need to replace the type of our graph from “l” to “b”:

|  |
| --- |
| plot(x, y1, type = "b") # Add symbols to points |



***Figure 5: Different Types of Line Plot.***

Example 6: Plot Multiple Lines to One Graph

In this example I want to show you how to plot multiple lines to a graph in R. First, we need to create further variables for our plot:

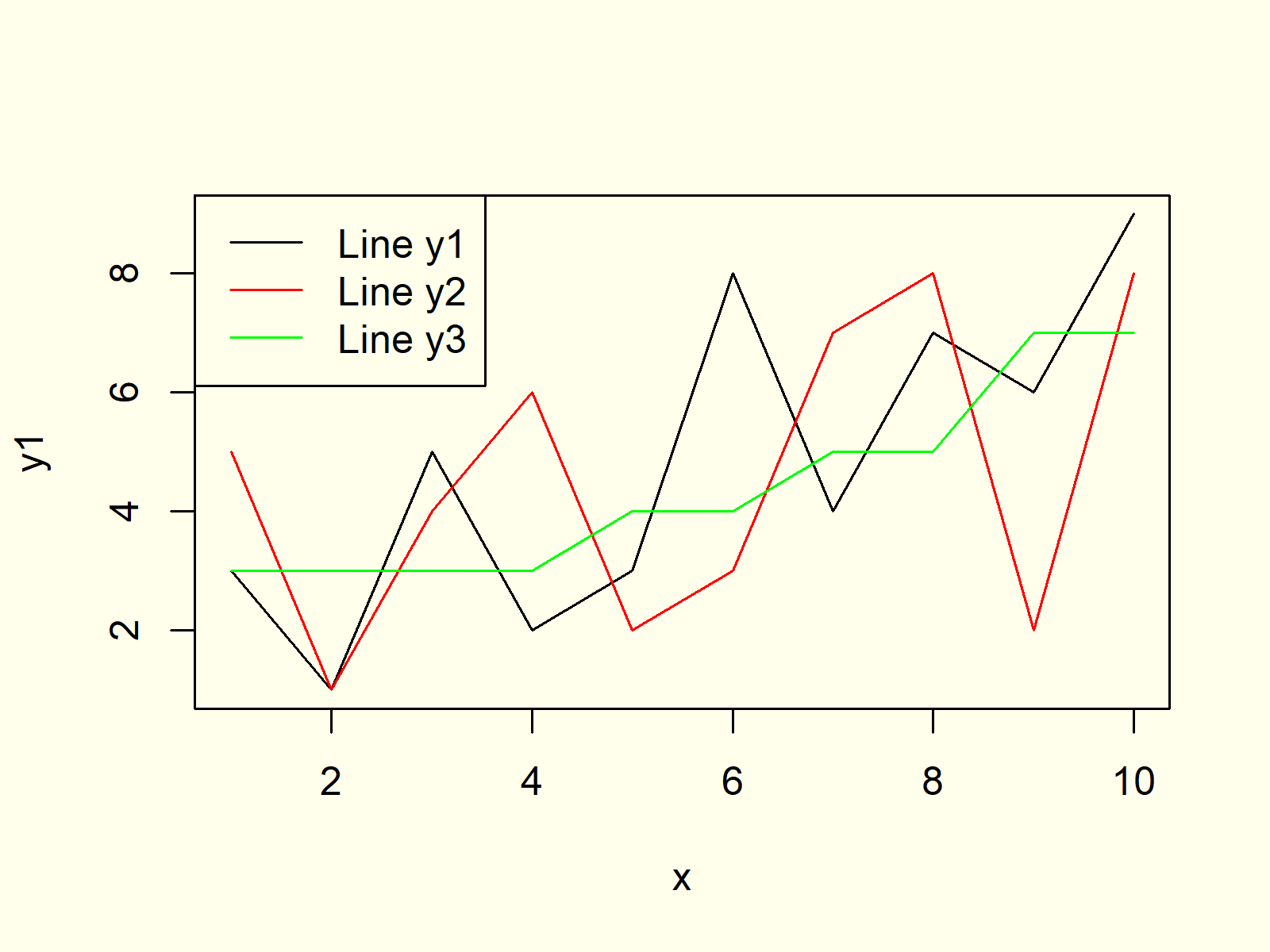
|  |
| --- |
| y2 <- c(5, 1, 4, 6, 2, 3, 7, 8, 2, 8) # Create more example data  y3 <- c(3, 3, 3, 3, 4, 4, 5, 5, 7, 7) |

Now, we can use the lines function to add these new data to our previously created line chart:

|  |
| --- |
| plot(x, y1, type = "l") # Draw first line  lines(x, y2, type = "l", col = "red") # Add second line  lines(x, y3, type = "l", col = "green") # Add third line |

Furthermore, we may [add a legend](https://statisticsglobe.com/add-legend-to-plot-in-base-r) to our picture to visualize which color refers to which of the different variables.

|  |
| --- |
| legend("topleft", # Add legend to plot  legend = c("Line y1", "Line y2", "Line y3"),  col = c("black", "red", "green"),  lty = 1) |



***Figure 6: Draw Several Lines in Same Graphic.***

Figure 6 shows the output of the R code of Example 6. We created a graph with multiple lines, different colors for each line, and a legend representing the different lines.

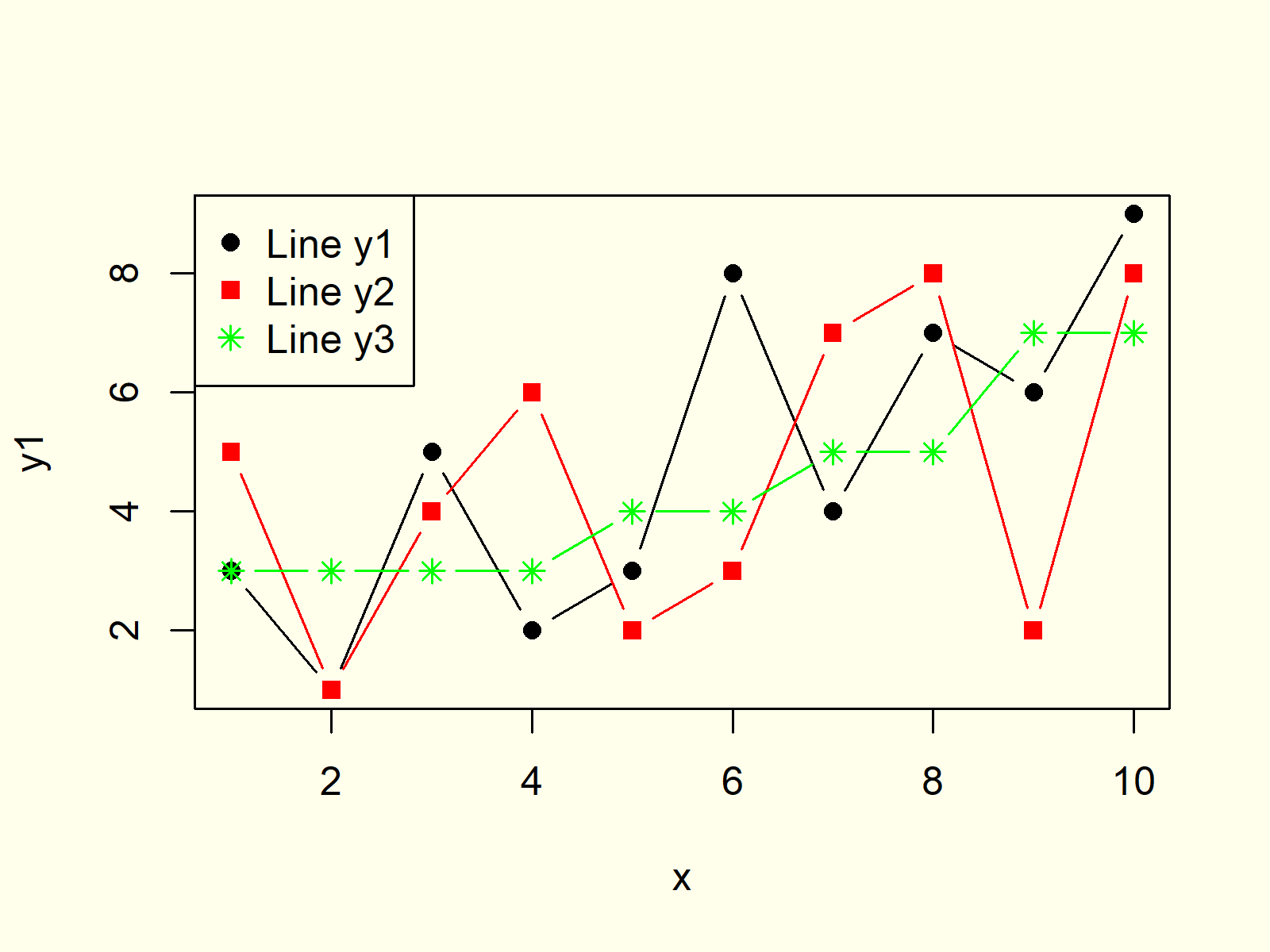
Example 7: Different Point Symbol for Each Line

Similar to Example 6, we can assign different point symbols to each of our lines by specifying type = “b”. With the pch argument we can specify a different point symbol for each line.

|  |
| --- |
| plot(x, y1, type = "b", pch = 16) # Change type of symbol  lines(x, y2, type = "b", col = "red", pch = 15)  lines(x, y3, type = "b", col = "green", pch = 8) |

We also need to consider these different point symbols in the legend of our plot:

|  |
| --- |
| legend("topleft", # Add legend to plot  legend = c("Line y1", "Line y2", "Line y3"),  col = c("black", "red", "green"),  pch = c(16, 15, 8)) |



***Figure 7: Change pch Symbols of Line Graph.***

Example 8: Line Graph in ggplot2 (geom\_line Function)

So far, we have only used functions of the base installation of the R programming language. However, there are many packages available that provide functions for the drawing of line charts.

One of the most powerful packages for the creation of graphics is the [ggplot2 package](https://ggplot2.tidyverse.org/reference/). We can install and load the ggplot2 package with the following two lines of R code:

|  |
| --- |
| install.packages("ggplot2") # Install and load ggplot2  library("ggplot2") |

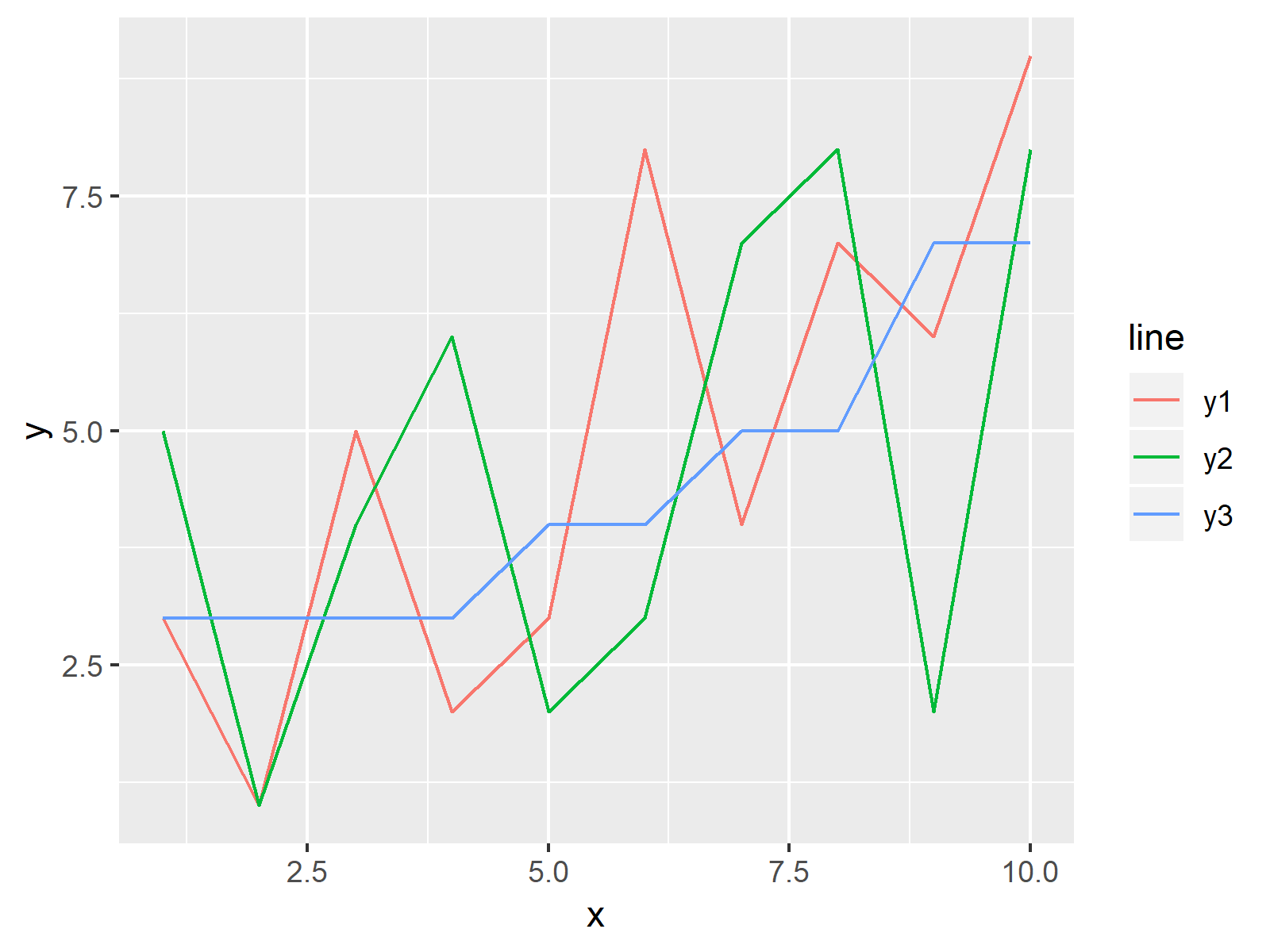
Furthermore, we need to store our data in a data frame, since the ggplot2 package is usually based on data frames:

|  |
| --- |
| data <- data.frame(x = rep(1:10, 3), # Create data frame  y = c(y1, y2, y3),  line = c(rep("y1", 10),  rep("y2", 10),  rep("y3", 10)))  head(data) # Print first 6 rows  # x y line  # 1 3 y1  # 2 1 y1  # 3 5 y1  # 4 2 y1  # 5 3 y1  # 6 8 y1 |

The RStudio console is showing how our new data is structured. Our data frame contains three columns and 30 rows. The first column contains of our x values (i.e. 1 to 10), the second column consists of the values of our three variables, and the third column is specifying to which variable the values of a row belong.

Now, we can apply the ggplot function in combination with the geom\_line function to draw a line graph with the ggplot2 package:

|  |
| --- |
| ggplot(data, aes(x = x, y = y, col = line)) + # Draw line plot with ggplot2  geom\_line() |



***Figure 8: Create Line Chart with ggplot2 Package.***

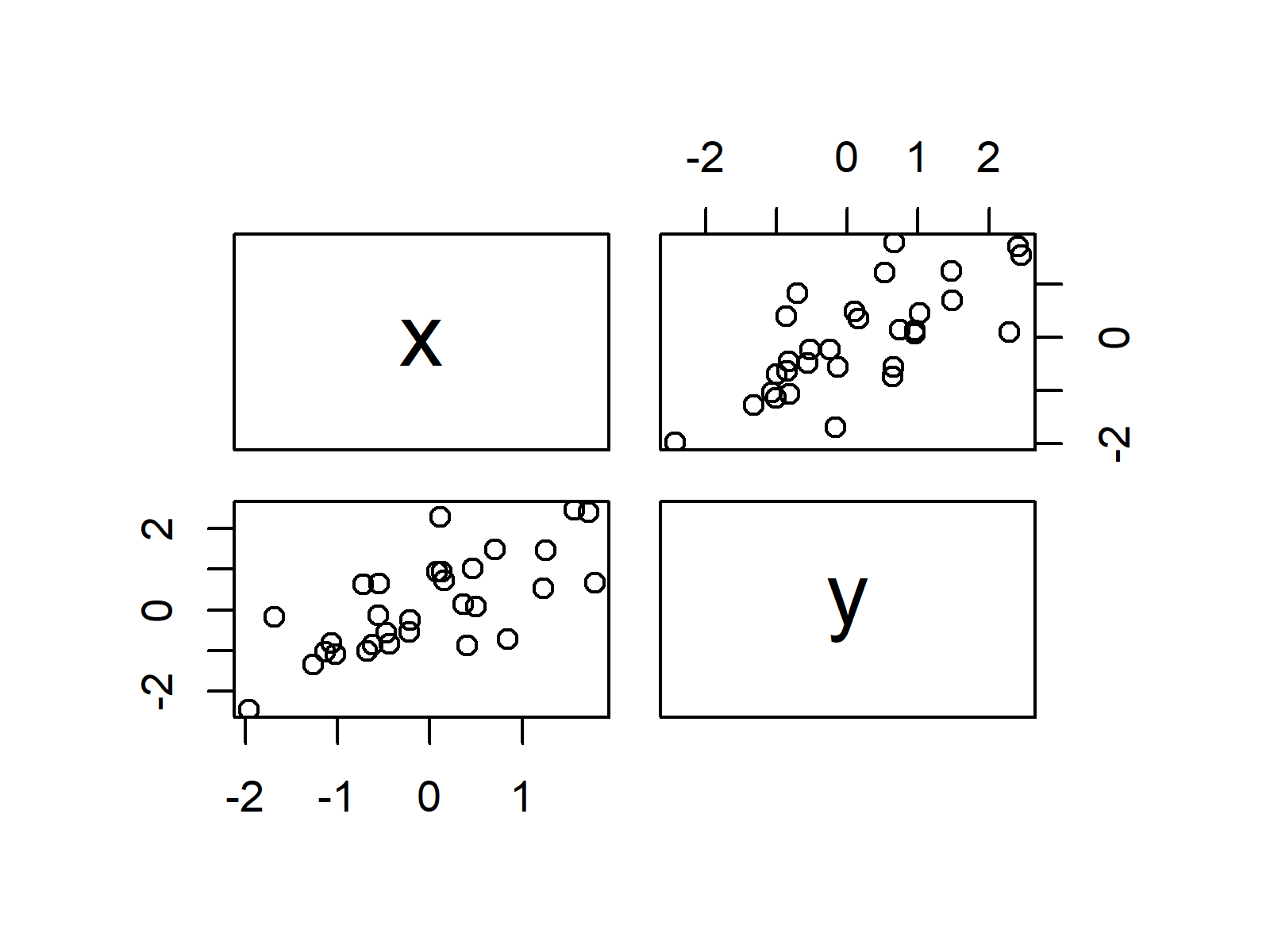
Figure 8 is showing how a ggplot2 line graph looks like. Of cause, the ggplot2 package is also providing many options for the modification of line graphics in R.

1. **Pair Plot**

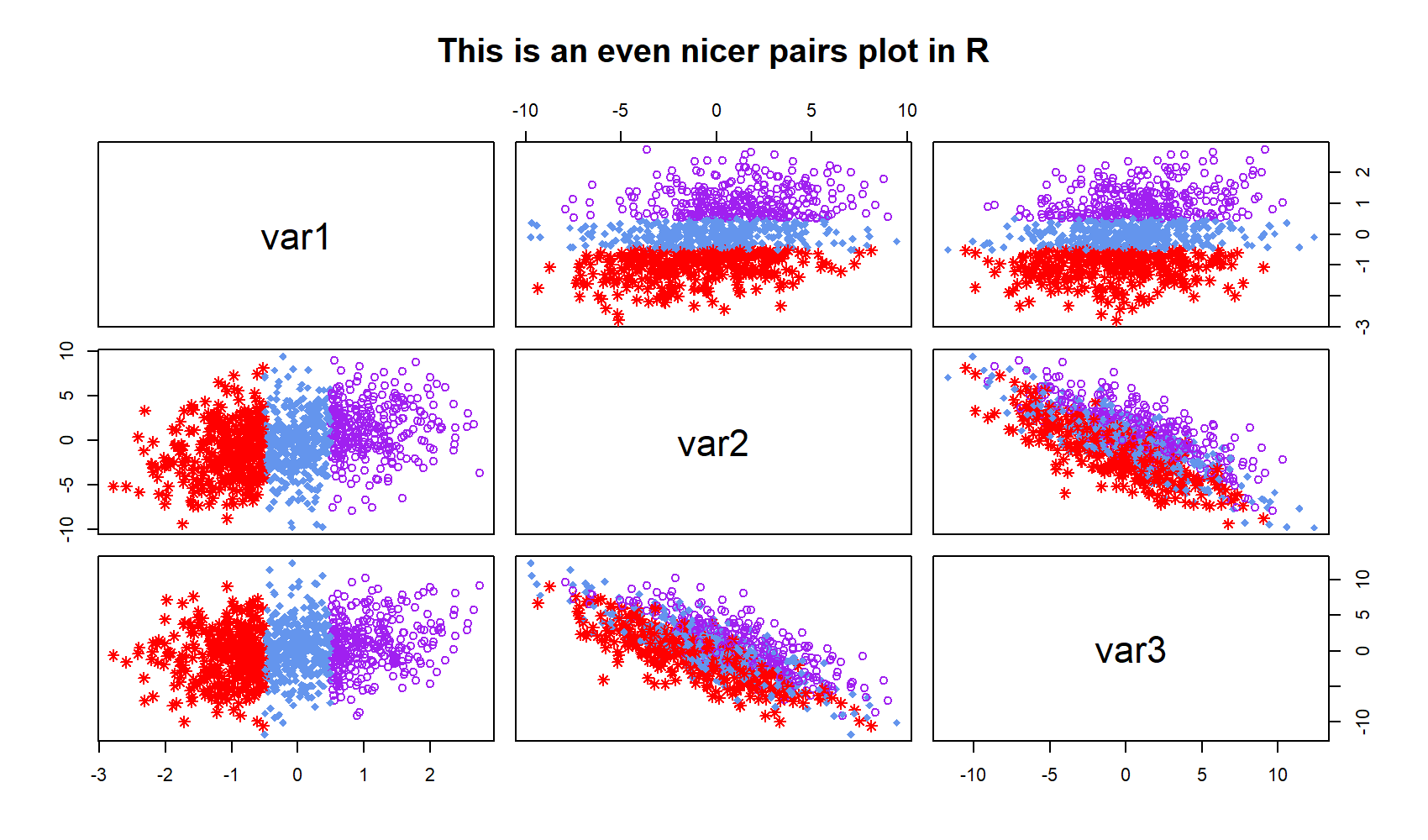
**Pairs Plot Definition:** A [pairs plot](https://statisticsglobe.com/r-pairs-plot-example/) is a plot matrix, consisting of scatterplots for each variable-combination of a data frame.

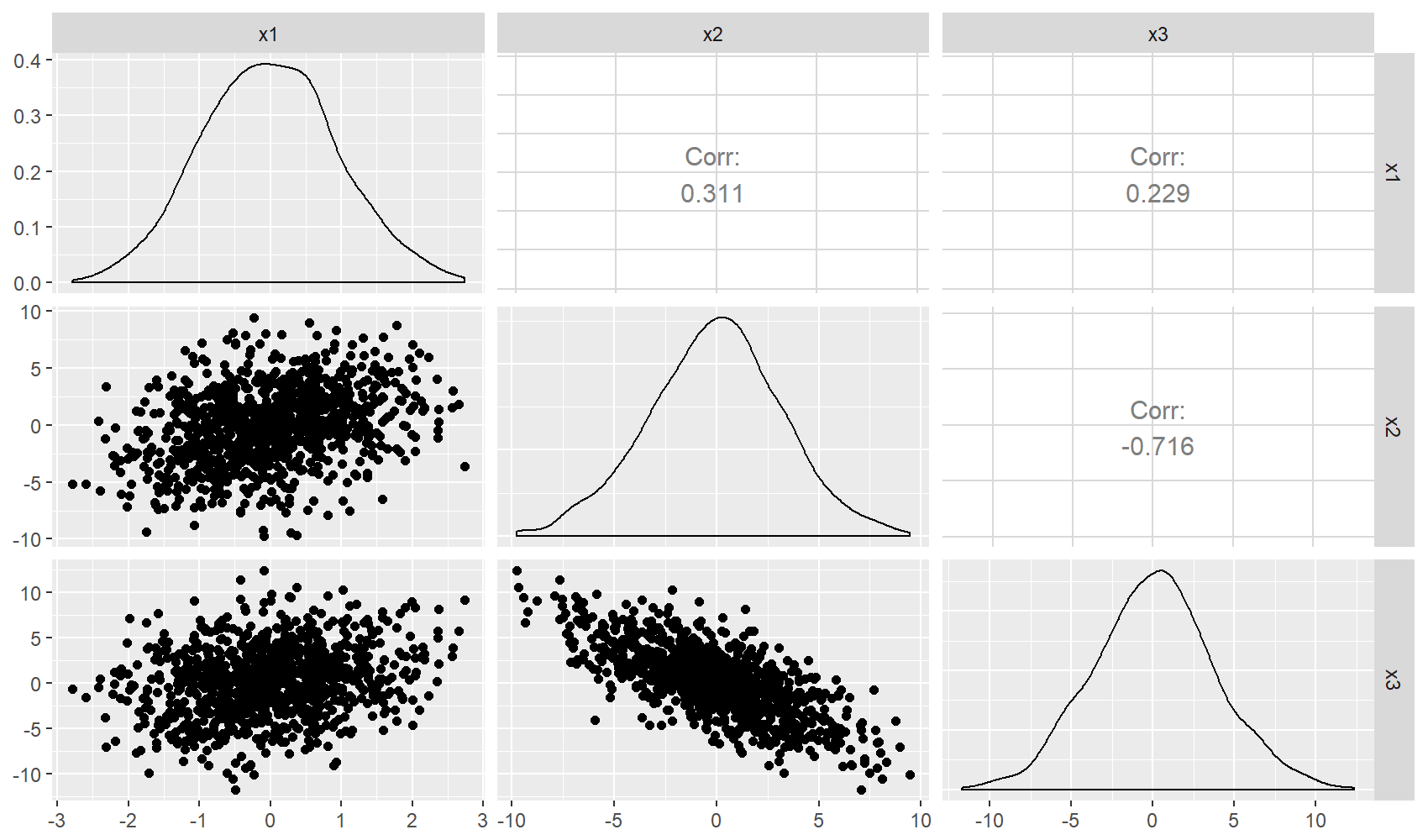
The following R syntax shows how to draw a basic pairs plot in R:

|  |
| --- |
| pairs(data.frame(x, y)) # Draw pairs plot in R |



**Advanced Pairs Plots:** Find some advanced pairs plots below. Click on the images to get more information and example R codes for each of the pairs plots.





**Pairs Plot Resources:** Find some further resources on the creation of pairs plots below.

## Example 1: Basic Application of pairs() in R

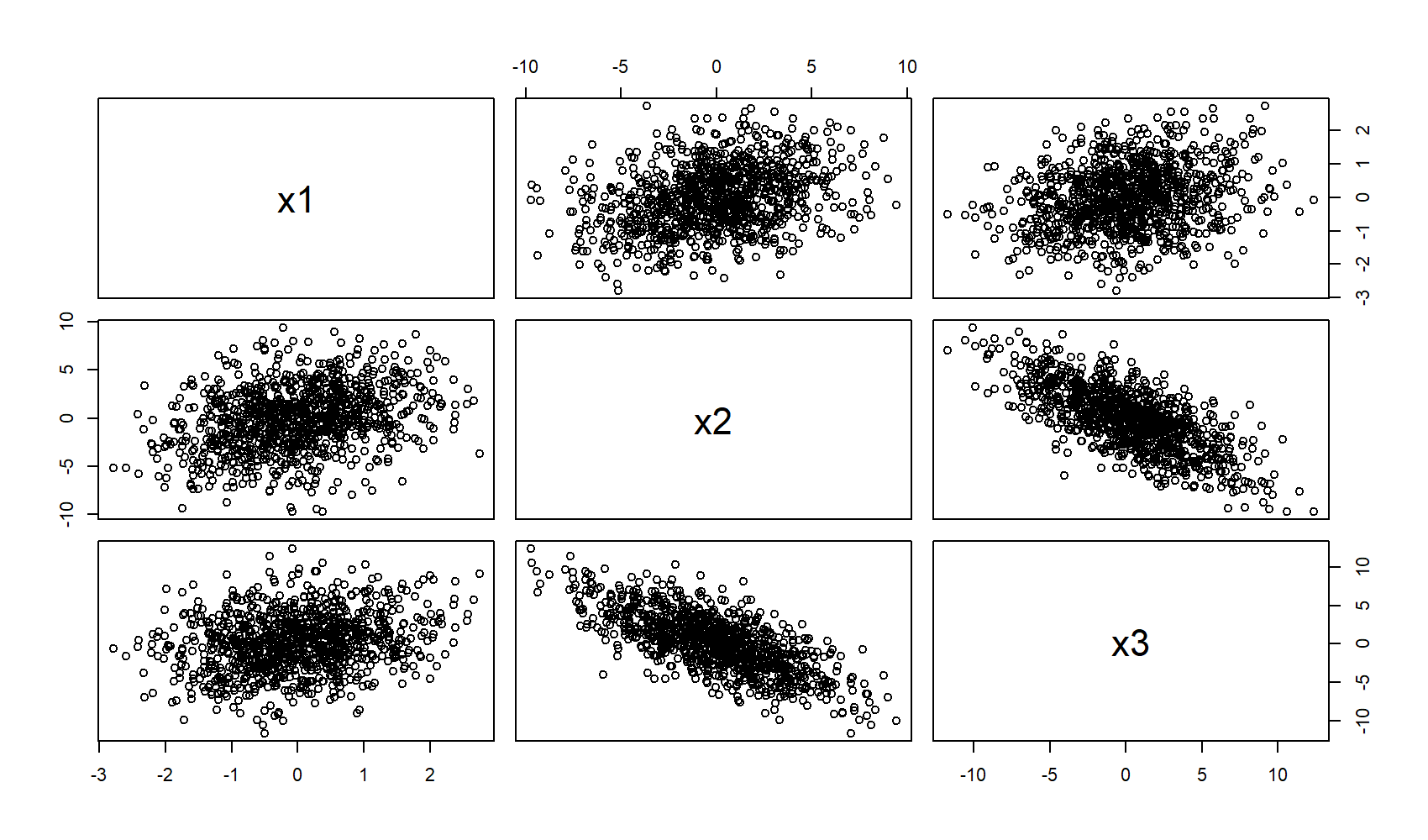
I’m going to start with a very basic application of the pairs R function. Let’s first create some random data for this example:

|  |
| --- |
| **set**.seed(525354) # Set seed for reproducibility  N <- 1000 # Sample size of 1000    x1 <- rnorm(N) # Create variable  x2 <- x1 + rnorm(N, 0, 3) # Create correlated variable  x3 <- 2 \* x1 - x2 + rnorm(N, 0, 2) # Create another correlated variable    data <- data.frame(x1, x2, x3) # Combine all variables to data.frame |

Our example data contains three [numeric variables](http://www.r-tutor.com/r-introduction/basic-data-types/numeric) and 1,000 rows.

Now, let’s apply the pairs function in R:

|  |
| --- |
| pairs(data) # Apply pairs function |



***Figure 1: Basic pairs()***[***R Plot***](https://statisticsglobe.com/graphics-in-r)***.***

As you can see, we are able to produce a relatively complex matrix of [scatterplots](https://chartio.com/learn/charts/what-is-a-scatter-plot/) with only one line of code. So, what does this pairs plot actually contain?

* The diagonal shows the names of the three numeric variables of our example data.
* The other cells of the plot matrix show a scatterplot (i.e. correlation plot) of each variable combination of our data frame. The middle graphic in the first row illustrates the correlation between x1 & x2; The right graph in the first row illustrates the correlation between x1 & x3; The left figure in the second row illustrates the correlation between x1 & x2 once more and so on…

In this first example, I have shown you the most basic usage of pairs in R. Let’s modify the options of the function a little bit…

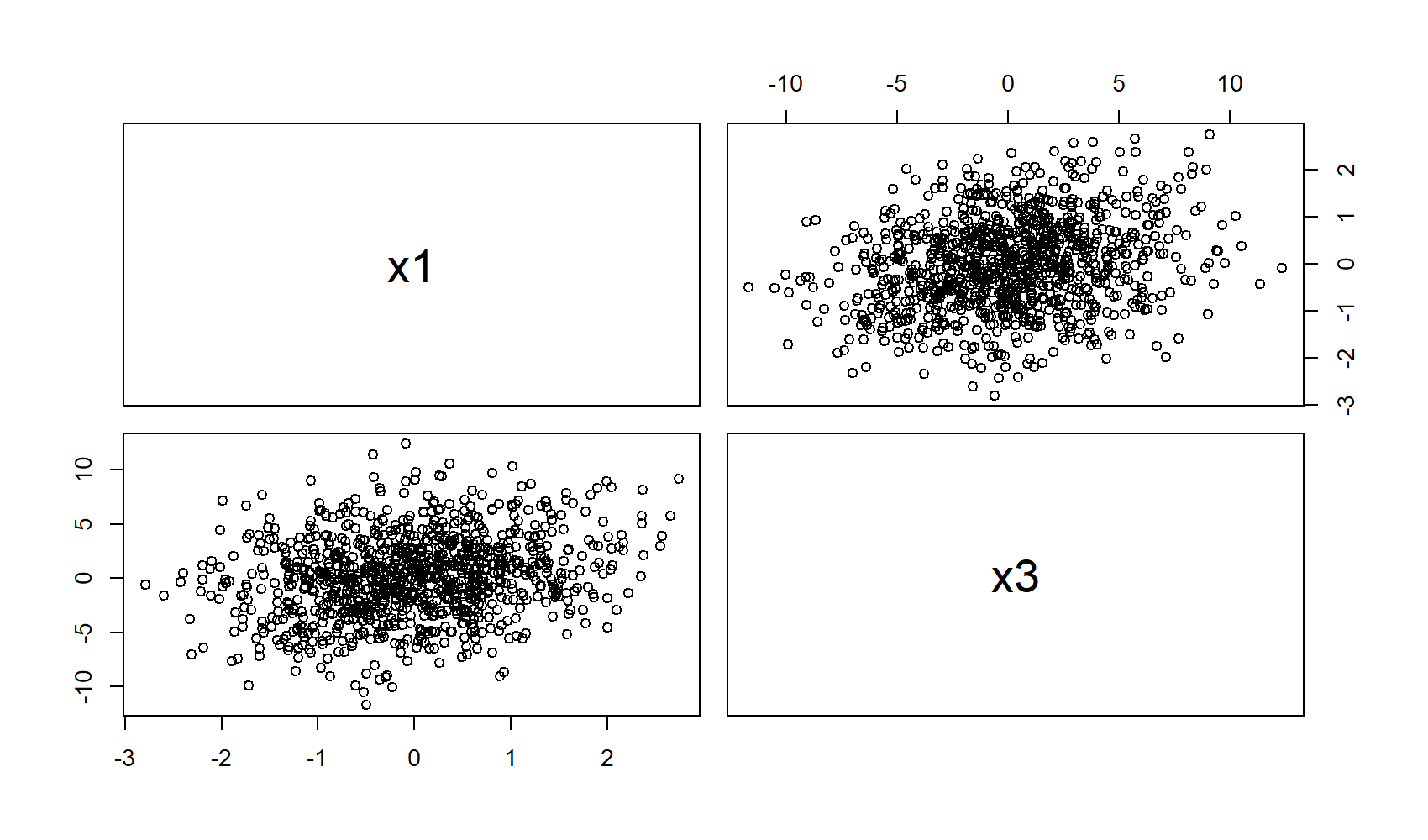
## Example 2: Selecting Variables of pairs Plot

Often, you will only be interested in the correlations of a few of your variables. Fortunately, this can be done easily by specifying a formula within the pairs command:

|  |
| --- |
| pairs(~ x1 + x2 + x3, data = data) # Produces same plot as in Example 1 |

With the code above, we can create exactly the same plot as in Example 1. However, we can simply remove the variables from the formula, for which we don’t want to produce a scatterplot:

|  |
| --- |
| pairs(~ x1 + x3, data = data) # Leave out one variable |



***Figure 2: Pairs Plot with Selection of Variables.***

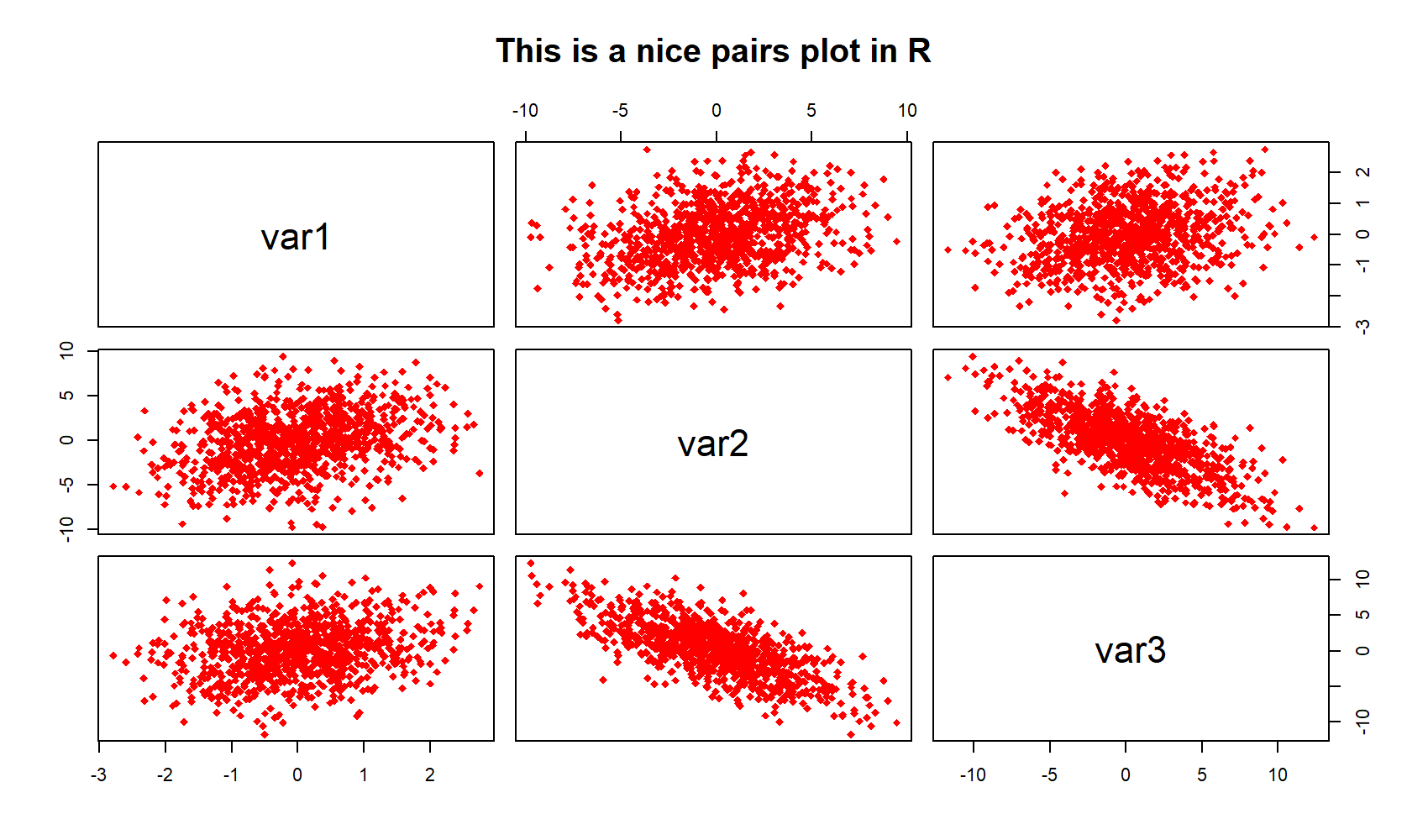
In this example, I deleted x2 from the formula, leading to a plot matrix that contains only the scatterplots of x1 and x3.

More modifications? Let’s do it!

## Example 3: Modify Color, Shape of Points, Labels & Title

In this example, I’m going to modify many different things:

|  |
| --- |
| pairs(data[ , 1:3],  col = "red", # Change color  pch = 18, # Change shape of points  labels = c("var1", "var2", "var3"), # Change labels of diagonal  main = "This is a nice pairs plot in R") # Add a main title |



***Figure 3: R Pairs Plot with Manual Color, Shape of Points, Labels, and Main Title.***

The modified pairs plot has a different color, diamonds instead of points, user-defined labels, and our own main title. For even more options, have a look at the help documentation of pairs by typing ?pairs to the RStudio console.

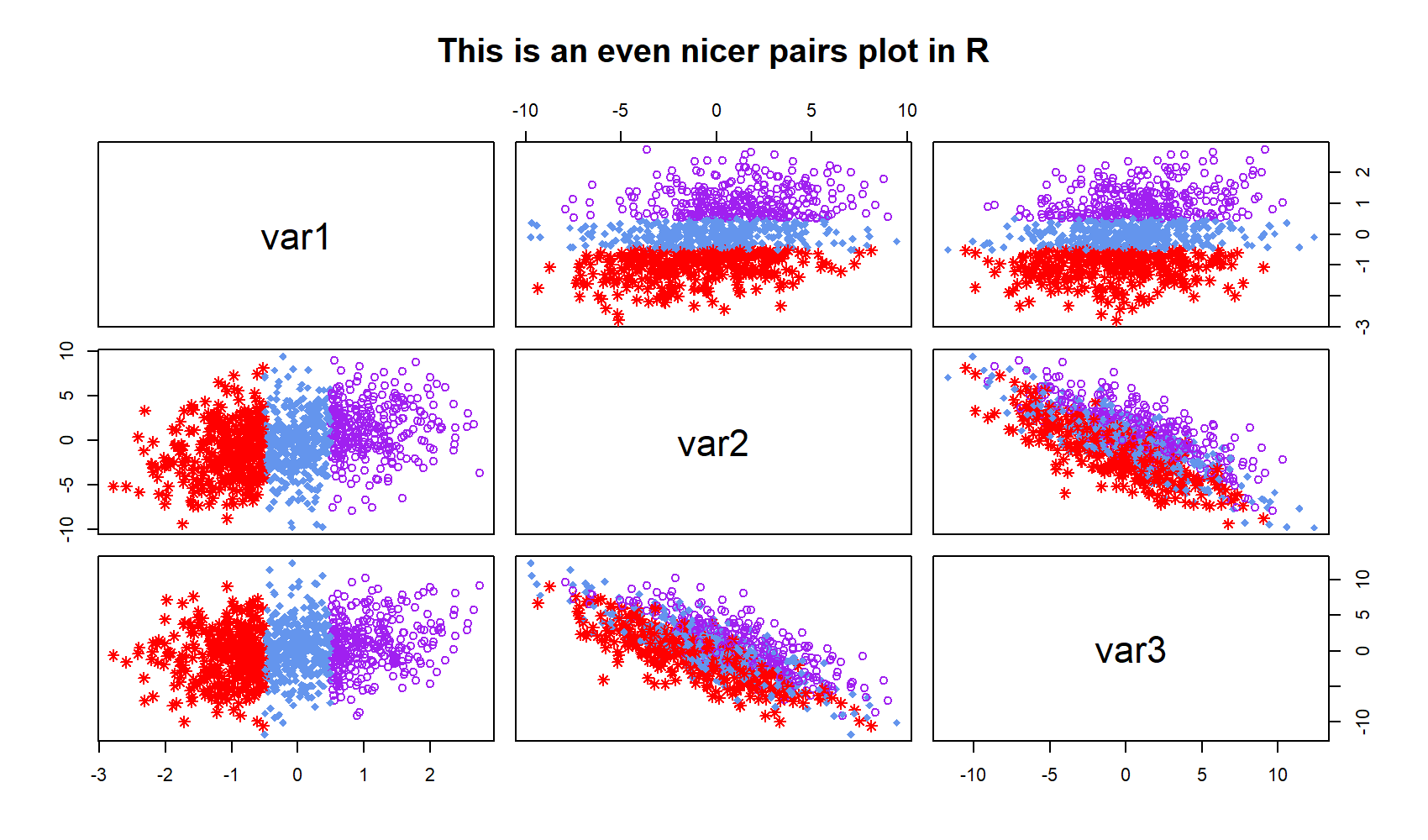
## Example 4: Modify pairs R Plot by Group

Quite often you will have different subsets or subgroups in your data. Let’s add a group indicator (three groups 1, 2 & 3) to our example data to simulate such a situation:

|  |
| --- |
| **group** <- NA  **group**[data$x1 < - 0.5] <- 1  **group**[data$x1 >= - 0.5 & data$x1 <= 0.5] <- 2  **group**[data$x1 > 0.5] <- 3 |

Now, let’s apply the pairs function again, but this time dependent on the group variable:

|  |
| --- |
| pairs(data[ , 1:3],  col = c("red", "cornflowerblue", "purple")[**group**], # Change color by group  pch = c(8, 18, 1)[**group**], # Change points by group  labels = c("var1", "var2", "var3"),  main = "This is an even nicer pairs plot in R") |



***Figure 4: pairs() Plot with Color & Points by Group.***

As you can see in Figure 4, we colored the plots and changed the shape of our data points according to our groups.

You need even more options? No problem, let’s move on…

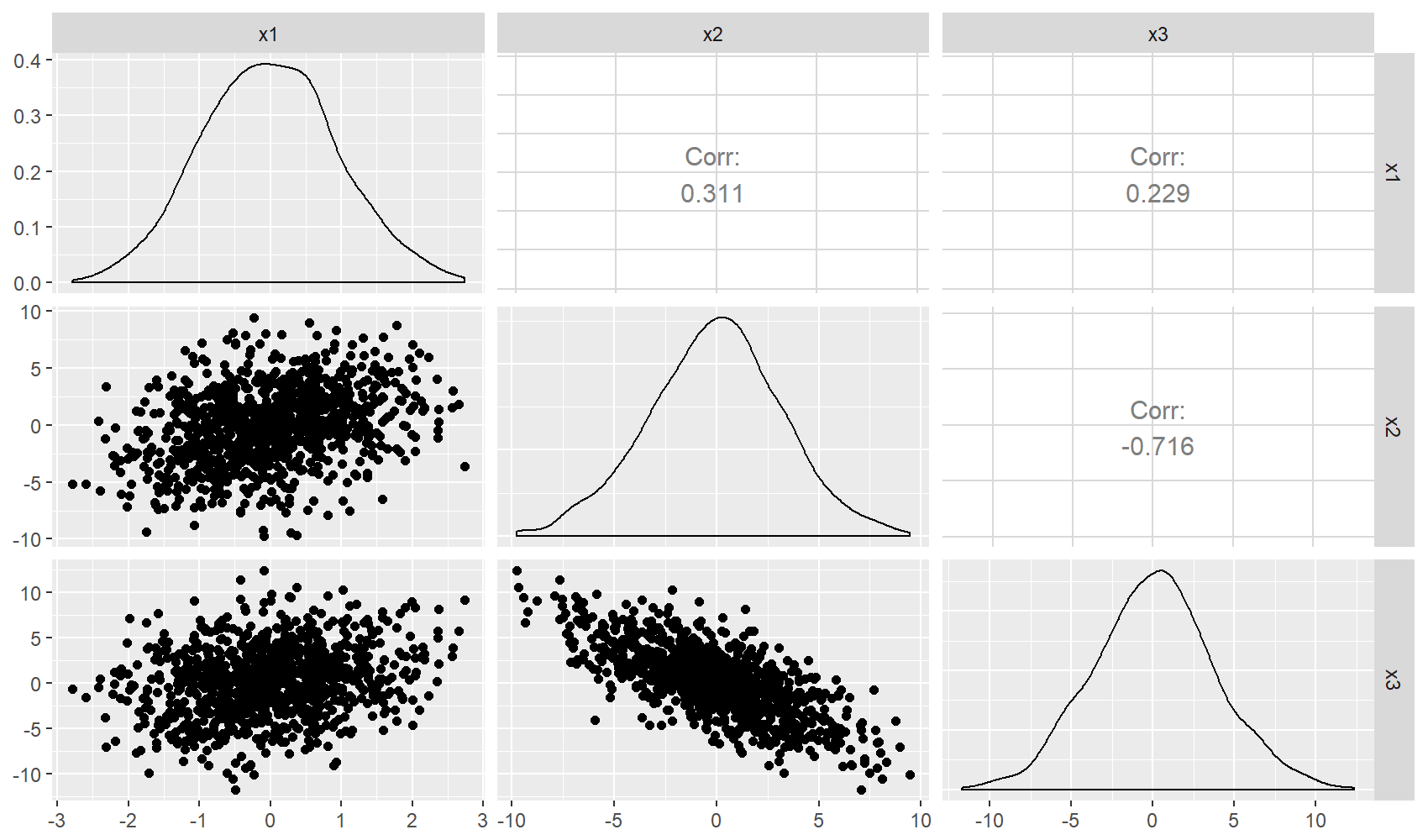
## Example 5: ggpairs R Function [ggplot2 & GGally]

So far, we have only used the pairs function that comes together with the base installation of R. However, the [ggplot2](https://ggplot2.tidyverse.org/) and [GGally](https://statisticsglobe.com/ggally-r-package) packages provide an even more advanced pairs function, which is called ggpairs(). Let’s install and load the packages:

|  |
| --- |
| install.packages("ggplot2") # Packages need to be installed only once  install.packages("GGally")    library("ggplot2") # Load ggplot2 package  library("GGally") # Load GGally package |

The basic application of ggpairs is similar to the pairs function of base R. You simply have to write the following R code:

|  |
| --- |
| ggpairs(data) # Apply ggpairs function |



***Figure 5: ggpairs R Plot via ggplot2 & GGally packages.***

Main difference to the pairs function of base R: The diagonal consists of the densities of the three variables and the upper panels consist of the correlation coefficients between the variables.

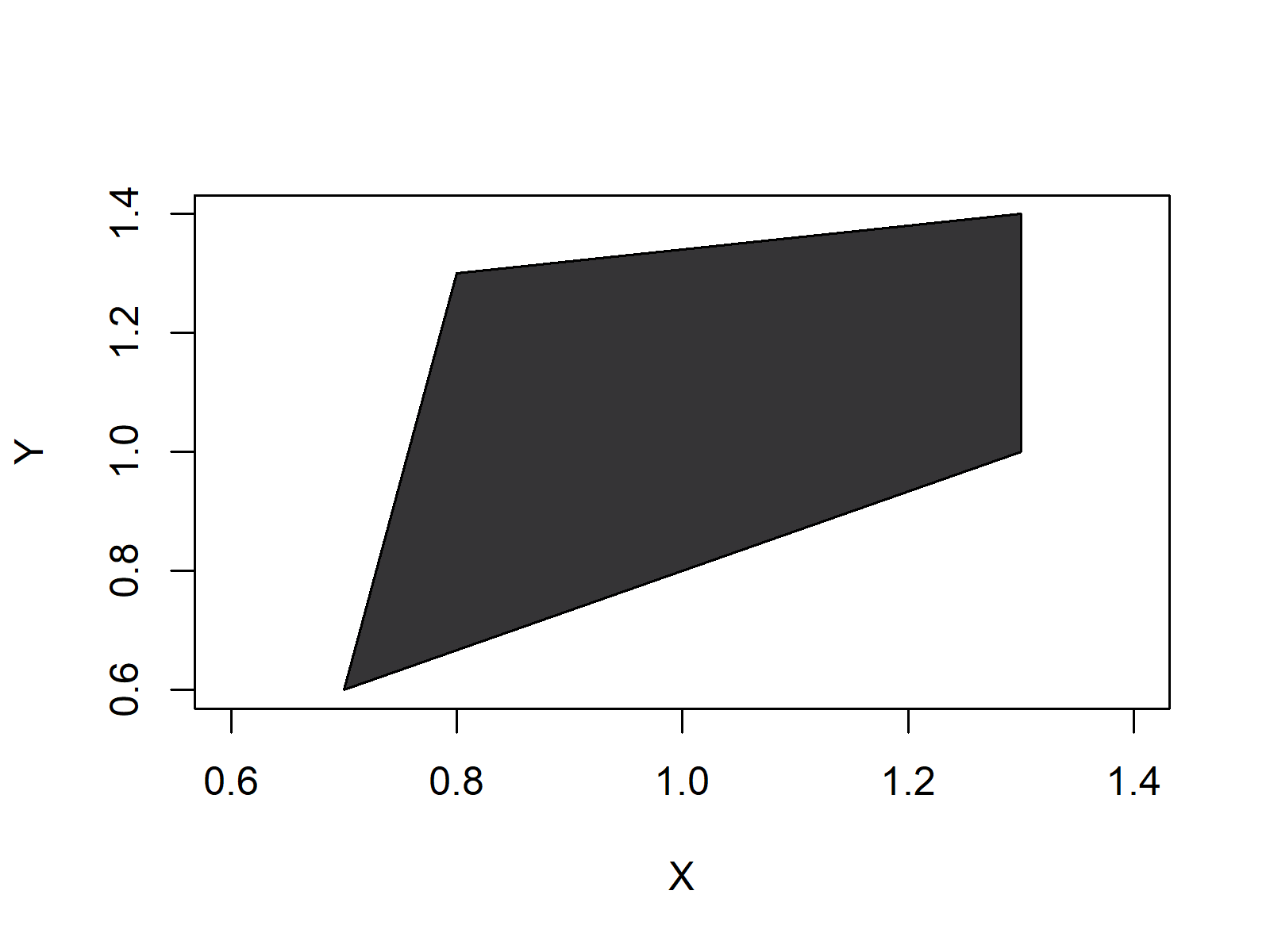
Even better than pairs of base R, isn’t it? However, there is even more to explore. In case, you want to know more about the R ggpairs function, I can recommend the following YouTube video of the channel Dragonfly Statistics:

1. **Polygon Plot**

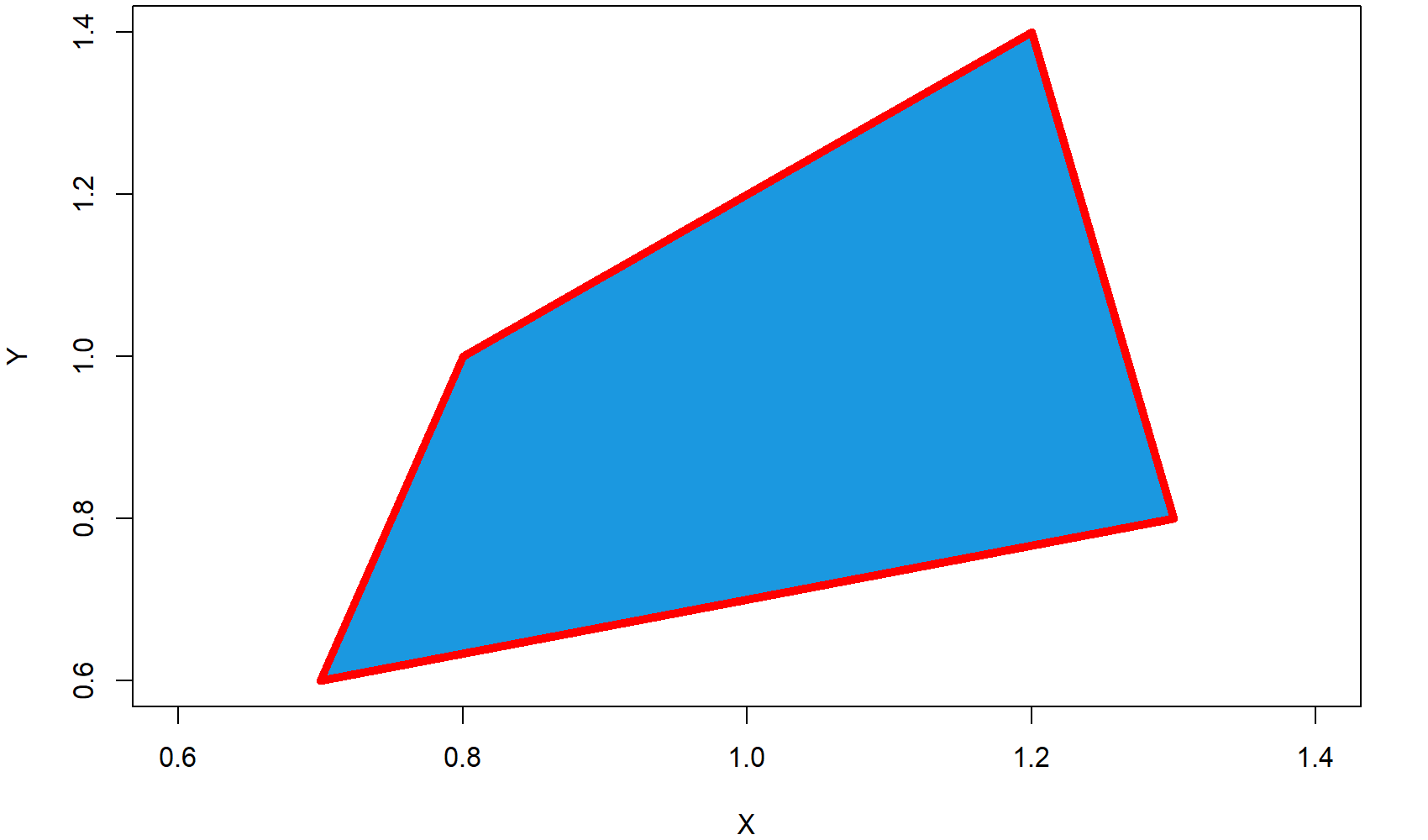
**Polygon Plot Definition:** A polygon plot displays a plane geometric figure (i.e. a [polygon](https://en.wikipedia.org/wiki/Polygon)) within the plot.

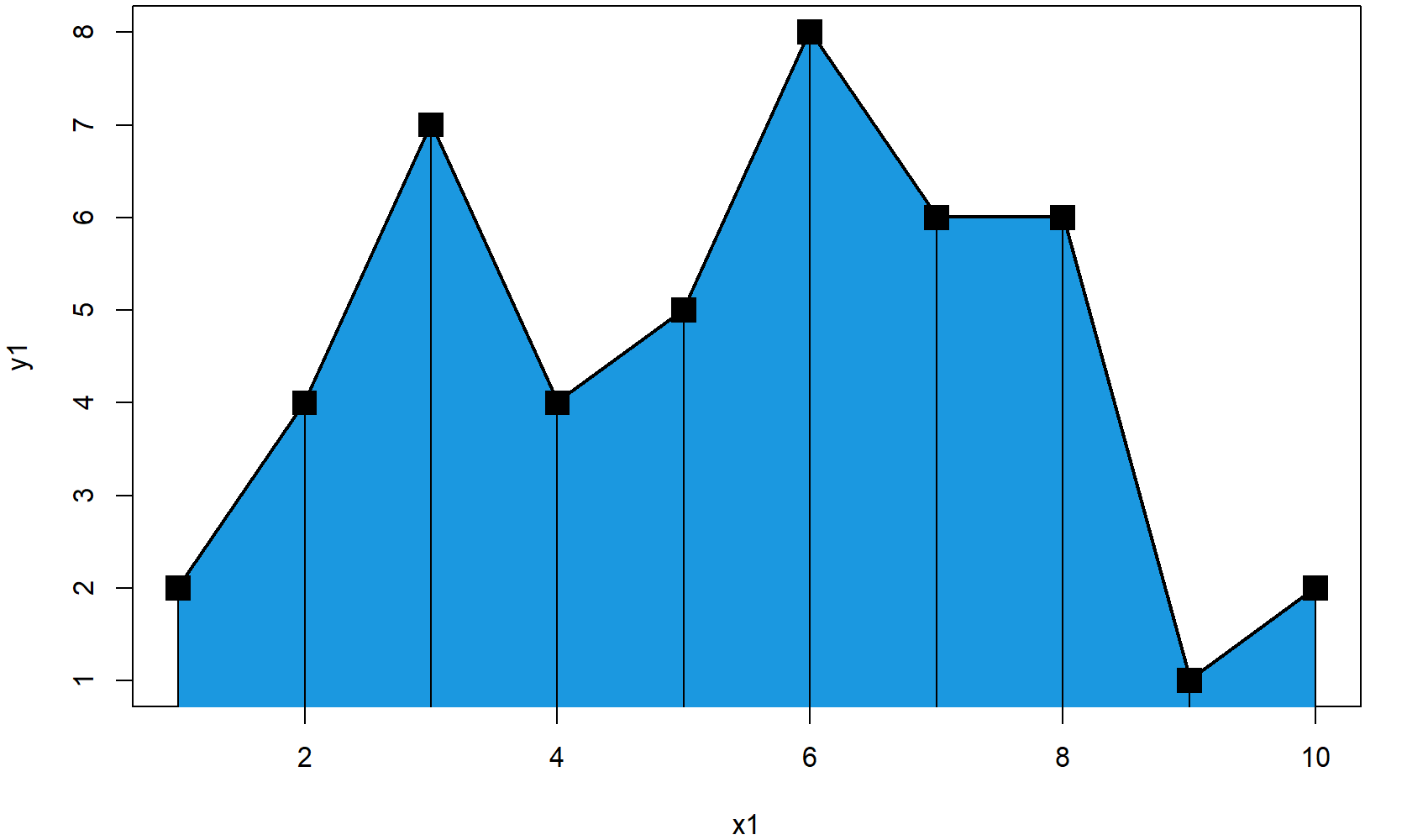
|  |
| --- |
| plot(1, 1, # Draw polygon plot in R  col = "white",  xlab = "X", ylab = "Y")  polygon(x = c(0.7, 1.3, 1.3, 0.8),  y = c(0.6, 1.0, 1.4, 1.3),  col = "#353436") |

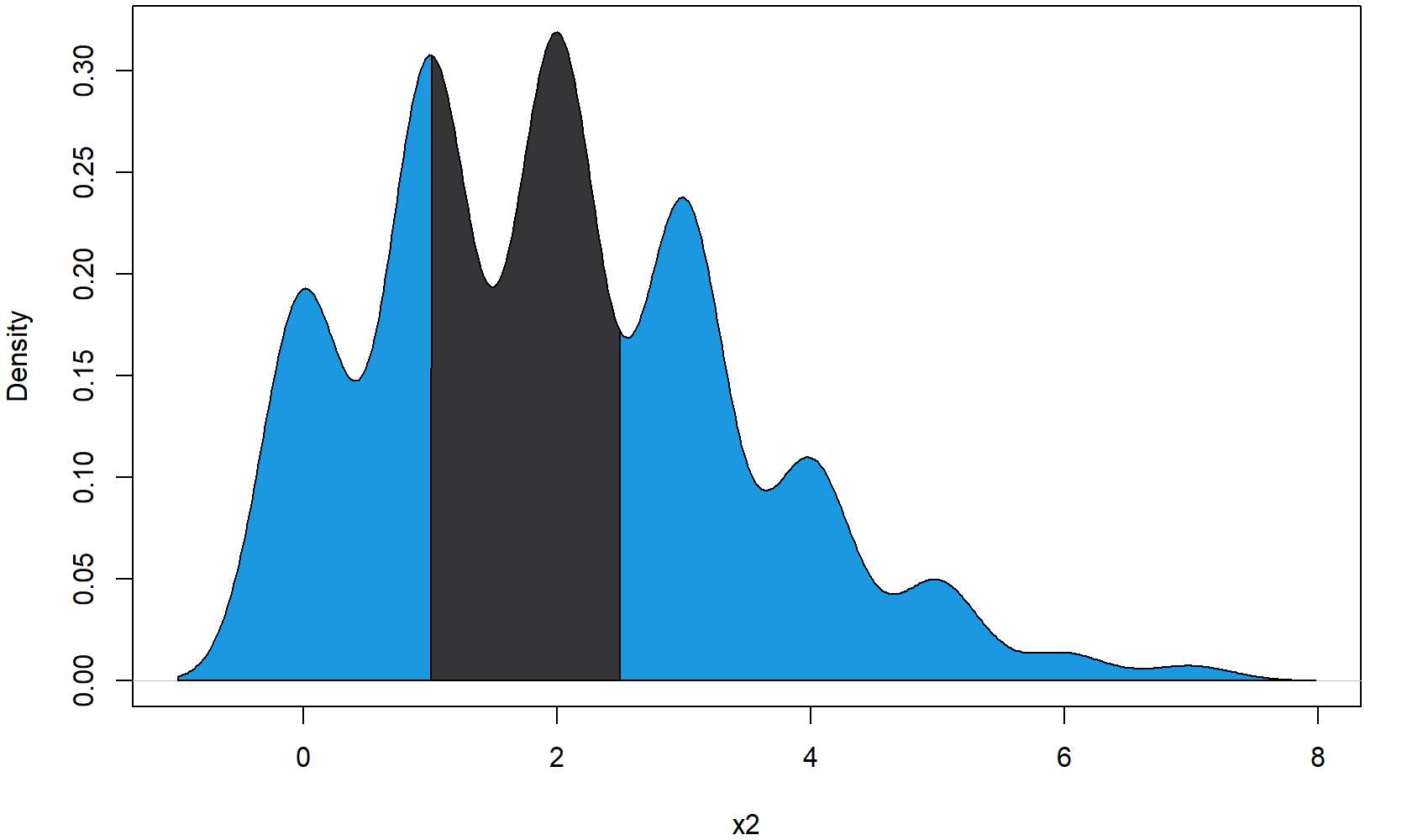
The following R syntax shows how to draw a basic polygon plot in R:

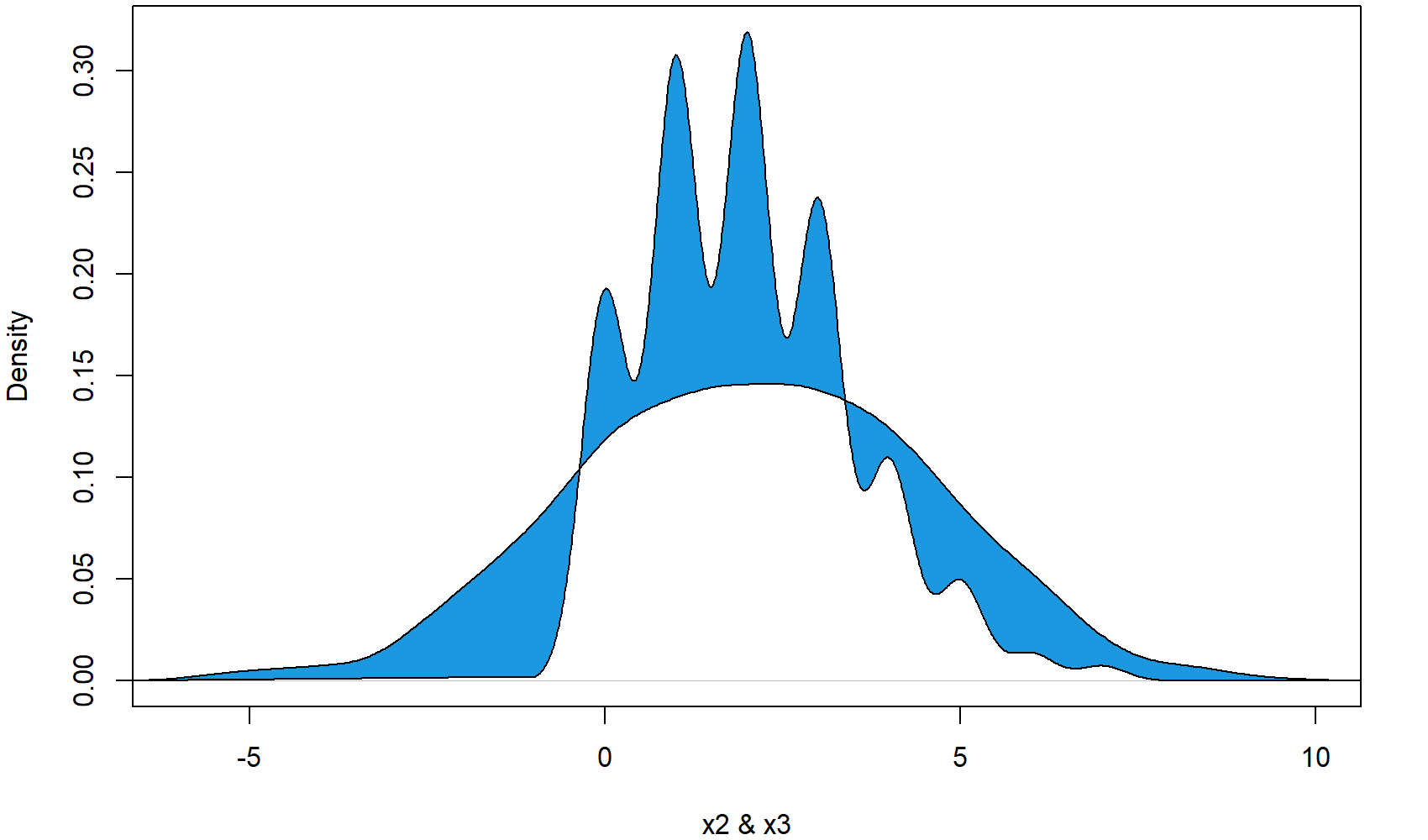


**Advanced Polygon Plots:** Find some advanced polygon plots below. Click on the images to get more information and example R codes for each of the polygon plots.









**Polygon Plot Resources:** Find some further resources on the creation of polygon plots below.

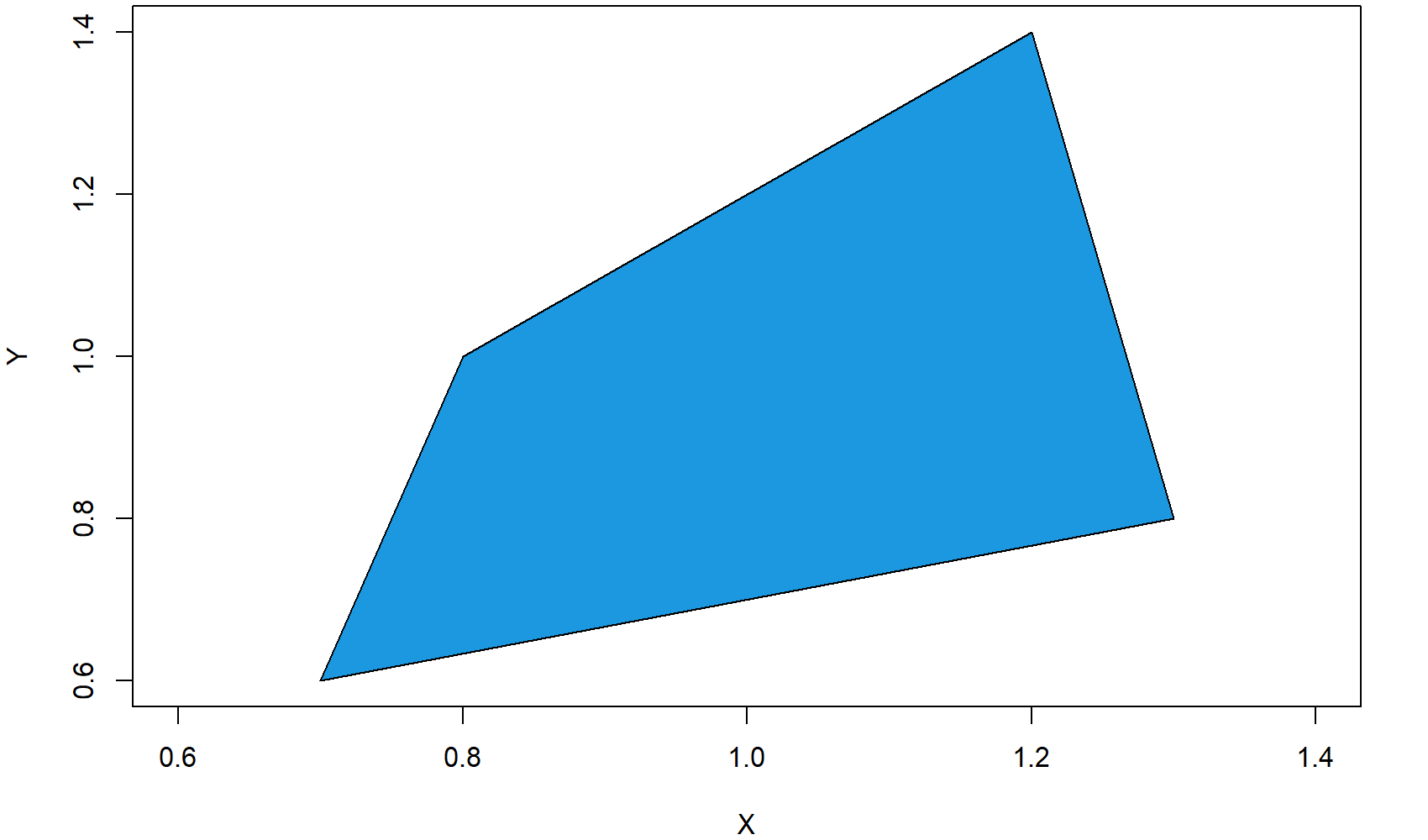
## Example 1: Draw a Square Polygon in an R Plot

Let’s begin with an easy example. In this example, we are going to draw a simple square [polygon](https://en.wikipedia.org/wiki/Polygon) to an empty R plot. Let’s first create an empty plot:

|  |
| --- |
| plot(1, 1, col = "white", xlab = "X", ylab = "Y") # Draw empty plot |

To this plot, we can draw a polygon with the following R code:

|  |
| --- |
| polygon(x = c(0.7, 1.3, 1.2, 0.8), # X-Coordinates of polygon  y = c(0.6, 0.8, 1.4, 1), # Y-Coordinates of polygon  col = "#1b98e0") # Color of polygon |



***Figure 1: Square Polygon in Empty Plot.***

As you can see, the previous polygon() R code consists of three different components:

* x: Here we specify the x-coordinates of each corner of the square polygon.
* y: Here we specify the y-coordinates of each corner of the square polygon.
* color: Here we specify the color of the polygon.

OK, that was easy, but let’s make the graphic a bit prettier…

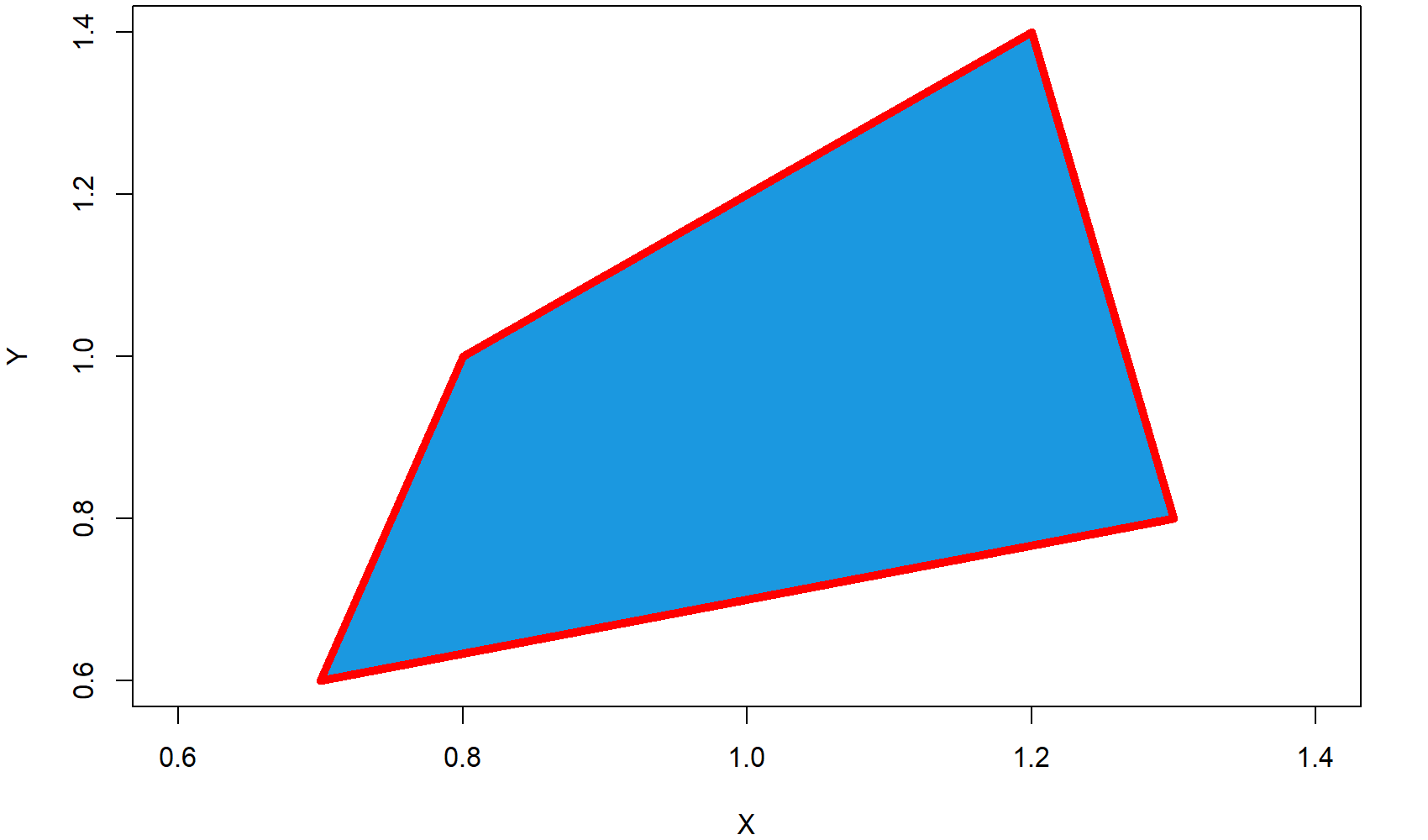
## Example 2: Colored Border of Polygon

For the second example, I’m using exactly the same code as in Example 1, but this time I’m modifying the border color of the polygon. Let’s begin again with an empty graph…

|  |
| --- |
| plot(1, 1, col = "white", xlab = "X", ylab = "Y") # Draw empty plot |

…and then let’s draw a thick red border around the polygon:

|  |
| --- |
| polygon(x = c(0.7, 1.3, 1.2, 0.8), # X-Coordinates of polygon  y = c(0.6, 0.8, 1.4, 1), # Y-Coordinates of polygon  col = "#1b98e0", # Color of polygon  border = "red", # Color of polygon border  lwd = 5) # Thickness of border |



***Figure 2: Square Polygon with Thick & Colored Border.***

We used two further options within the R polygon function:

* border: Here we specify the border color.
* lwd: Here we specify the thickness of the border.

That’s it with the square polygon examples. But as you will see in the next examples, the polygon R function has much more to offer.

## Example 3: Frequency Polygon in R

In the following example, I will show you how to create a [frequency polygon](http://onlinestatbook.com/2/graphing_distributions/freq_poly.html) in R. Let’s create some data for the example:

|  |
| --- |
| x1 <- 1:10 # X values for frequency polygon  y1 <- c(2, 4, 7, 4, 5, 8, 6, 6, 1, 2) # Y values for frequency polygon |

We can draw a frequency polygon plot with the following R code. First, we [draw the line](https://statisticsglobe.com/plot-line-in-r-graph-chart) of the frequency polygon with the plot function:

|  |
| --- |
| plot(x1, y1, # Plot frequency polygon  type = "l", # Set line type to line  lwd = 3) # Thickness of line |

Then, we can add some color below the line with the polygon function:

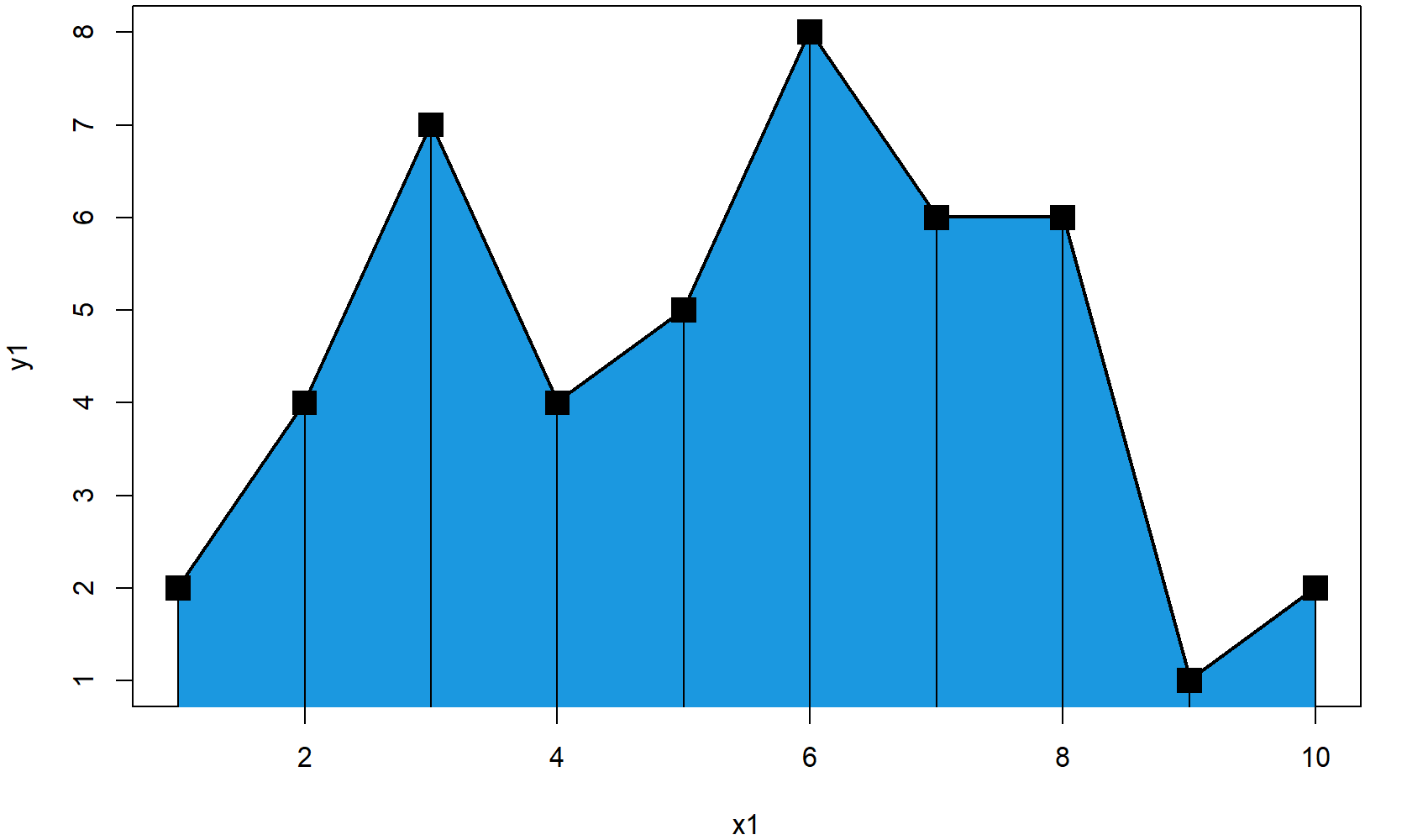
|  |
| --- |
| polygon(c(1, x1, 10), c(0, y1, 0), # X-Y-Coordinates of polygon  col = "#1b98e0") # Color of polygon |

To make the polygon frequency plot a bit prettier, we can also add some squares at the cut-points of our frequency polygon:

|  |
| --- |
| points(x1, y1, # Add squares to frequency polygon  cex = 2, # Size of squares  pch = 15) # Set point type to squares |

And finally, we can add some [line segments](https://statisticsglobe.com/segments-r-example/) to the figure:

|  |
| --- |
| segments(x1, 0, x1, y1) # Add line segments to plot |



***Figure 3: Frequency Polygon in R.***

Looks good. So let’s move on to the next – very popular – application of the polygon R function…

## Example 4: Draw Polygon Below Density

The following application of the polygon function is quite often used to make the plot of a [probability density function (PDF)](https://www.investopedia.com/terms/p/pdf.asp) more visible. With the following R code, you can fill the area below a [density curve](https://statisticsglobe.com/kernel-density-plot-in-base-r) with color (i.e. we are drawing a polygon according to the shape of the density).

Again, let’s begin with some data. For this example, I’m going to use a [poisson distribution](https://brilliant.org/wiki/poisson-distribution/" \t "_blank):

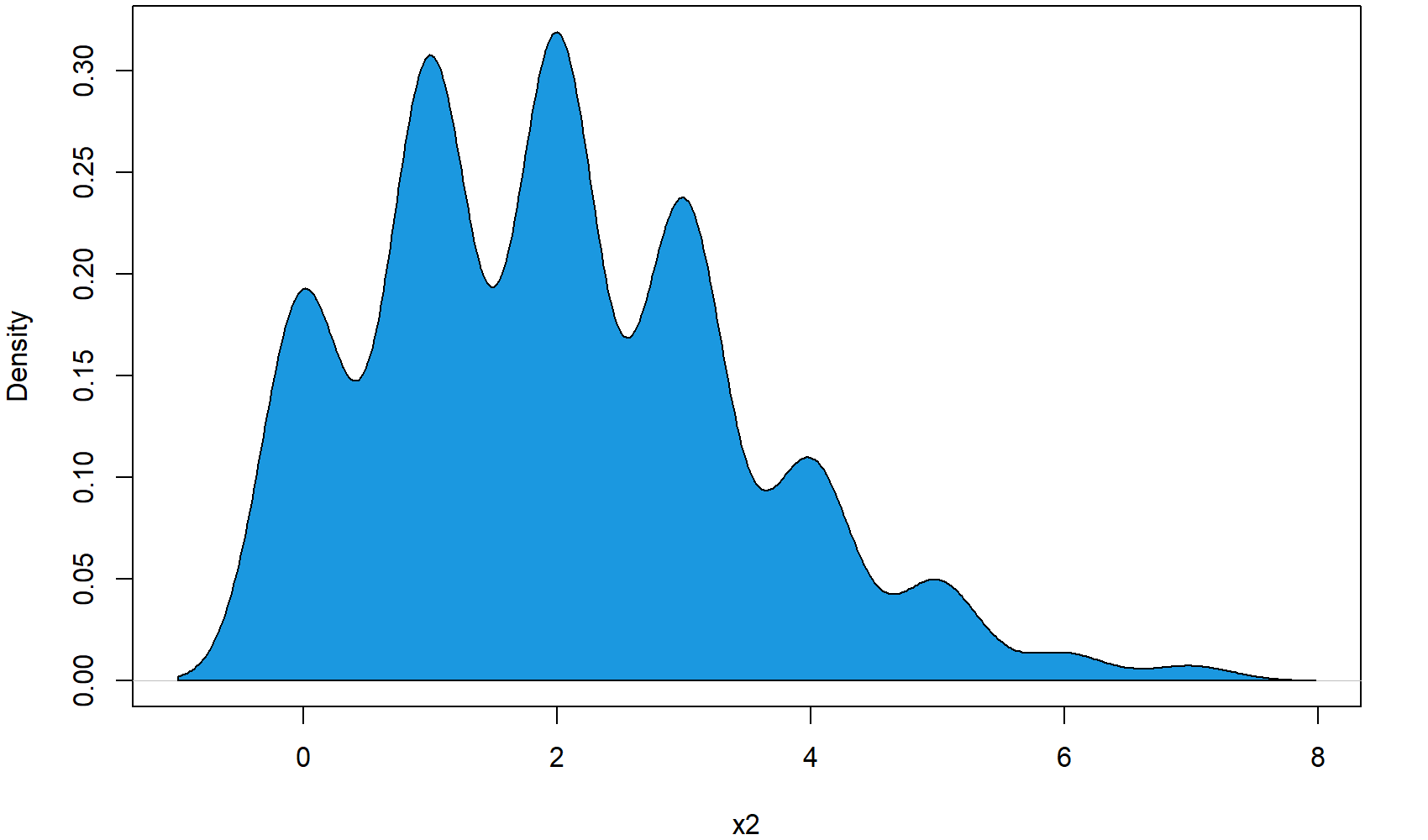
|  |
| --- |
| **set**.seed(15051) # Set seed for reproducibility  N <- 1000 # Sample size  x2 <- rpois(N, 2) # Draw random poisson distribution |

First, we have to draw the density curve to a plot…

|  |
| --- |
| plot(density(x2), # Draw density plot  main = "", # No main title  xlab = "x2") # Set name of x-axis to x2 |

…and then we can add the polygon below this density:

|  |
| --- |
| polygon(c(min(density(x2)$x), density(x2)$x), # X-Coordinates of polygon  c(0, density(x2)$y), # Y-Coordinates of polygon  col = "#1b98e0") # Color of polygon |



***Figure 4: Polygon Below Poisson Distribution.***

## Example 5: Density Polygon for Specific Range

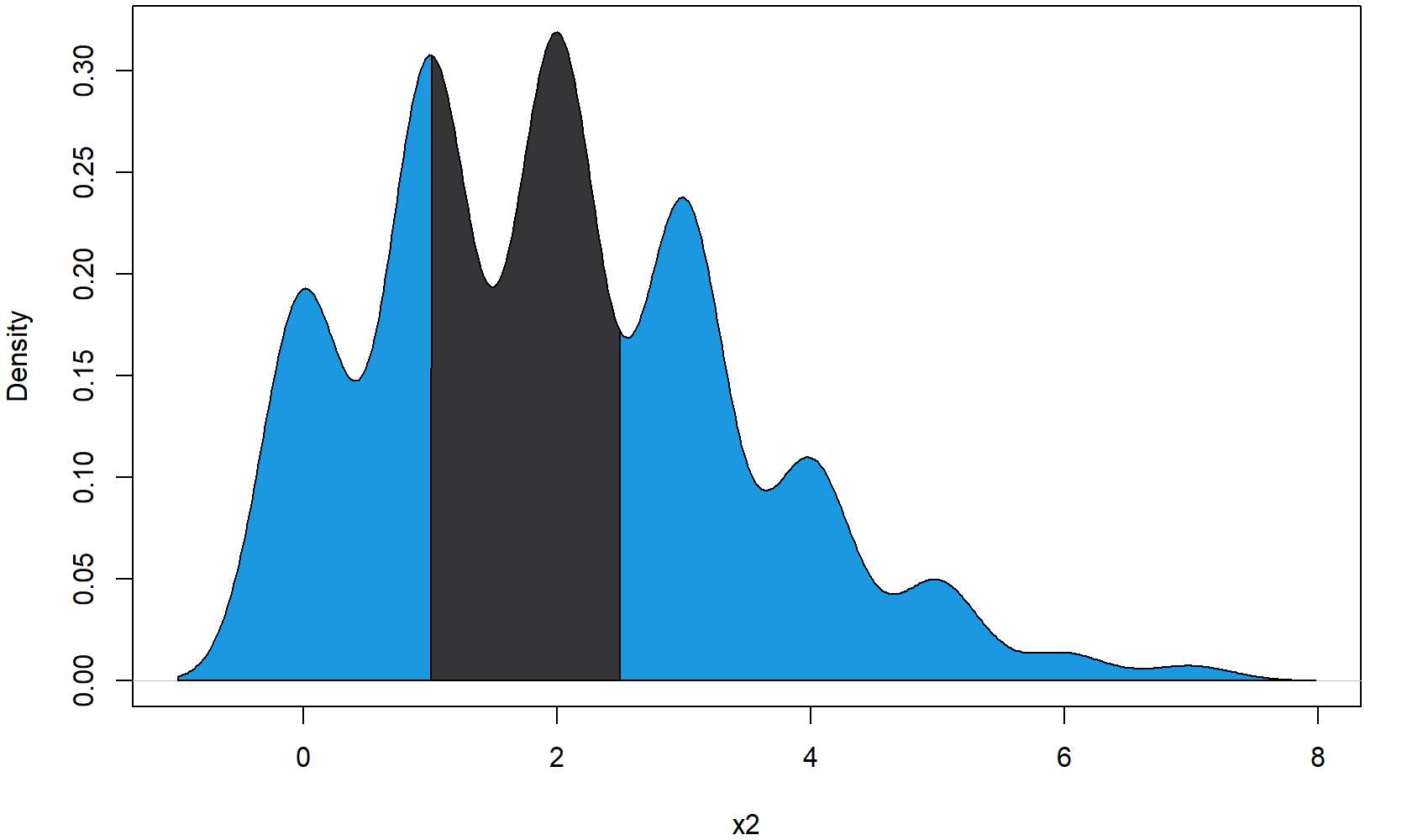
In this example, I’m going to show you how to restrict your density polygon to a certain range of x-values. At this point, I’m just continuing the code of Example 4.

First, we have to define the range, to which we want to restrict our polygon:

|  |
| --- |
| poly\_range <- density(x2)$x > 1 & density(x2)$x < 2.5 # Set polygon x-range |

Then, we can use this range to subset our polygon coordinates:

|  |
| --- |
| polygon(c(1, density(x2)$x[poly\_range], 2.5), # X-Coordinates of polygon range  c(0, density(x2)$y[poly\_range], 0), # Y-Coordinates of polygon range  col = "#353436") # Color of polygon range |



***Figure 5: Density Polygon for Restricted X-Range.***

As you can see, we are just overlaying the blue polygon of Example 4 with our restricted polygon. However, we could of cause just skip the polygon of Example 4 and draw only the restricted polygon.

At this point, you know how to restrict polygons to certain X-values. However, we could also restrict a polygon to certain values on the Y-axis…

## Example 6: Draw Polygon Between Two Densities

We can either restrict our density polygon to a fixed Y-range – or even prettier, we can restrict our polygon to the shapes of two densities. For the first density, I’m using again the x2 data that we created in Example 2. Let’s plot the x2 density:

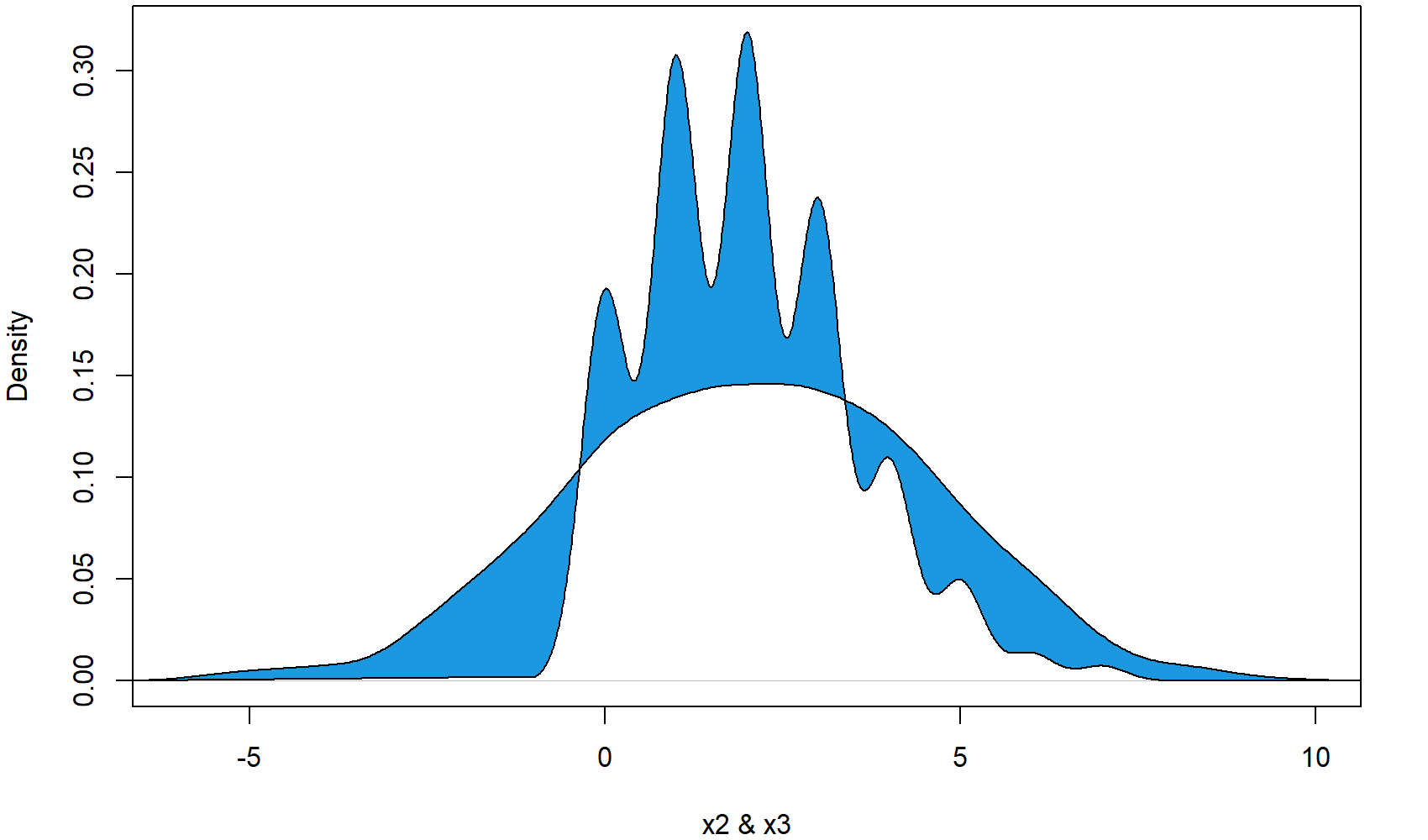
|  |
| --- |
| plot(density(x2), # Draw density plot again  main = "", # No main title  xlab = "x2 & x3", # Set name of x-axis to x2 & x3  xlim = c(- 6, 10)) # Wider xlim |

For the second density, I’m using the following data:

|  |
| --- |
| x3 <- rnorm(N, 2, 2.5) # Draw random normal distribution |

We can now use the values of our first and second density to draw a polygon between the two densities:

|  |
| --- |
| polygon(c(density(x3)$x, rev(density(x2)$x)), # X-Coordinates of polygon  c(density(x3)$y, rev(density(x2)$y)), # Y-Coordinates of polygon  col = "#1b98e0") # Color of polygon |



***Figure 6: Polygon Between Two Densities.***

As you can see in the plot, the second densities follows a [normal distribution](https://statisticsglobe.com/normal-distribution-in-r-dnorm-pnorm-qnorm-rnorm) that is partly lower and partly higher than our first density. Interestingly, the polygon is reflecting this by drawing the polygon sometimes below the first density line and sometimes above the first density line.