Data visualization in R

Surya Lamichhane

01 March 2025

Contents

By the end of this lesson, you will be able to:

- Visualize data using Using R-Graphics
- Explore the data using Visualization tools in R

What is data visualization? (IBM's Definition)

Data visualization is the representation of data through use of common graphics, such as charts, plots, infographics, and even animations. These visual displays of information communicate complex data relationships and data-driven insights in a way that is easy to understand. This technique mainly use for

- Data exploration,
- Idea generation,
- Idea illustration,
- Visual discovery, and
- Data explanation and story telling.

What is the best way of understanding tools in R?

The help() function in R is essential because it provides immediate access to built-in documentation for functions, packages, and datasets. Here's why it's helpful:

- Quick Reference: Instead of searching online, you can directly access function descriptions, syntax, arguments, and usage examples.
- Understanding Function Arguments: The help page details all arguments a function accepts, their default values, and how to modify them.
  - help(mean) or ?mean shows documentation for the mean function.
  - help(plot) or ?plot() shows documentation for the plot function.
  - help(barplot) or ?barplot() shows documentation for the barplot function.
- Examples for Learning: Many help pages include example code that you can run to understand how a function works.
  - Example: example(plot)
- Accessing Dataset Documentation: If a package includes datasets, you can look up descriptions and structure.
  - Example: help(mtcars)
- Package Documentation: You can explore functions within an installed package.
  - Example: help(package="ggplot2") lists all available functions in ggplot2.
- Finding Related Topics: Using ??keyword (fuzzy search) helps find documentation related to a topic.
  - Example: ??regression lists all help pages related to regression. Debugging Issues: If a function isn't working as expected, the documentation often includes details about error handling and limitations.

Part 1: Data visualization using R Graphics

1. Plot function in R

For more details see [https://r-coder.com/plot-r/]

Syntax

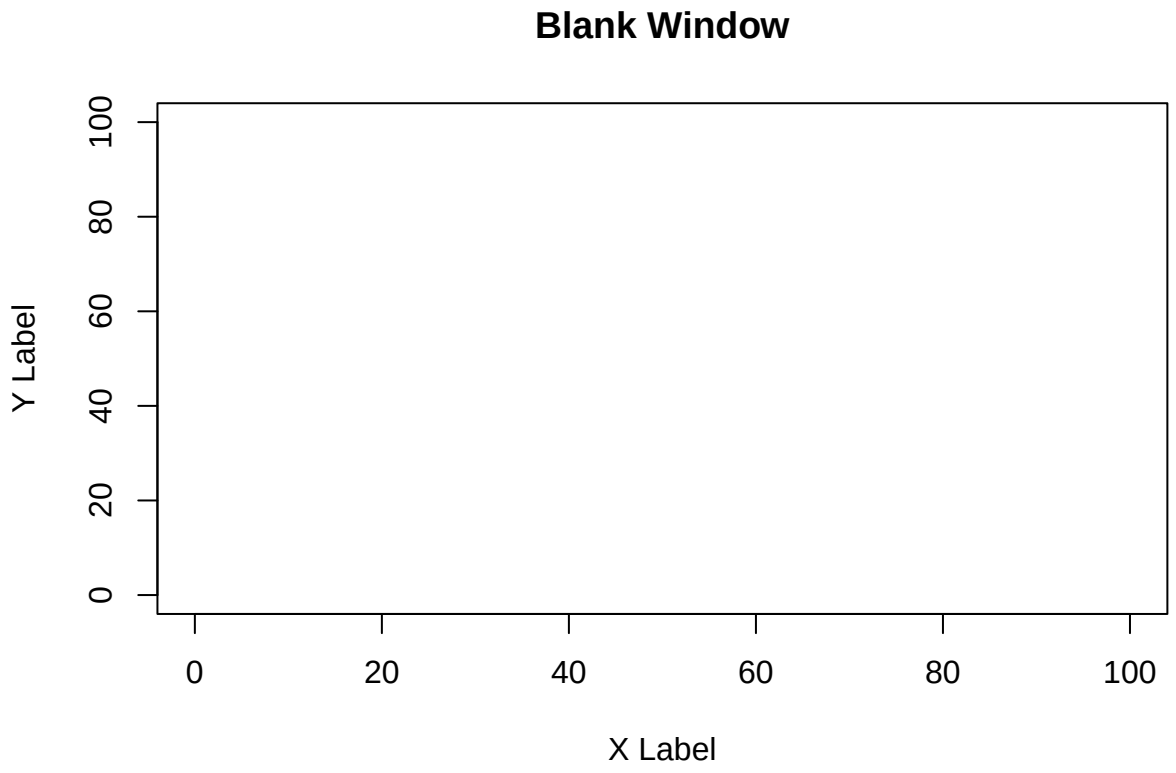
```
plot(x, y, ...)
- the following arguments are optional
for dot plot: type = 'p' (default)
for line chart: type = 'l'
to assign plot title: main = "title", a charactor field
xlab = "Name of X varaible", a charactor field
ylab = "Name of y varaible", a charactor field
xlim = limit of x values, a numerice range
ylim =  limit of y values, a numerice range
```

1.1. Get the documentation of plot function using help function

- use help(plot)

1.2. Creat an empty window in R

```
# Create a blank plotting space
plot(x = 1:10,
     xlab = "X Label",
     ylab = "Y Label",
     xlim = c(0, 100),
     ylim = c(0, 100),
     main = "Blank Window",
     type = "n" # for not ploting points from data x.
    )
```



1.3. Scattor plot

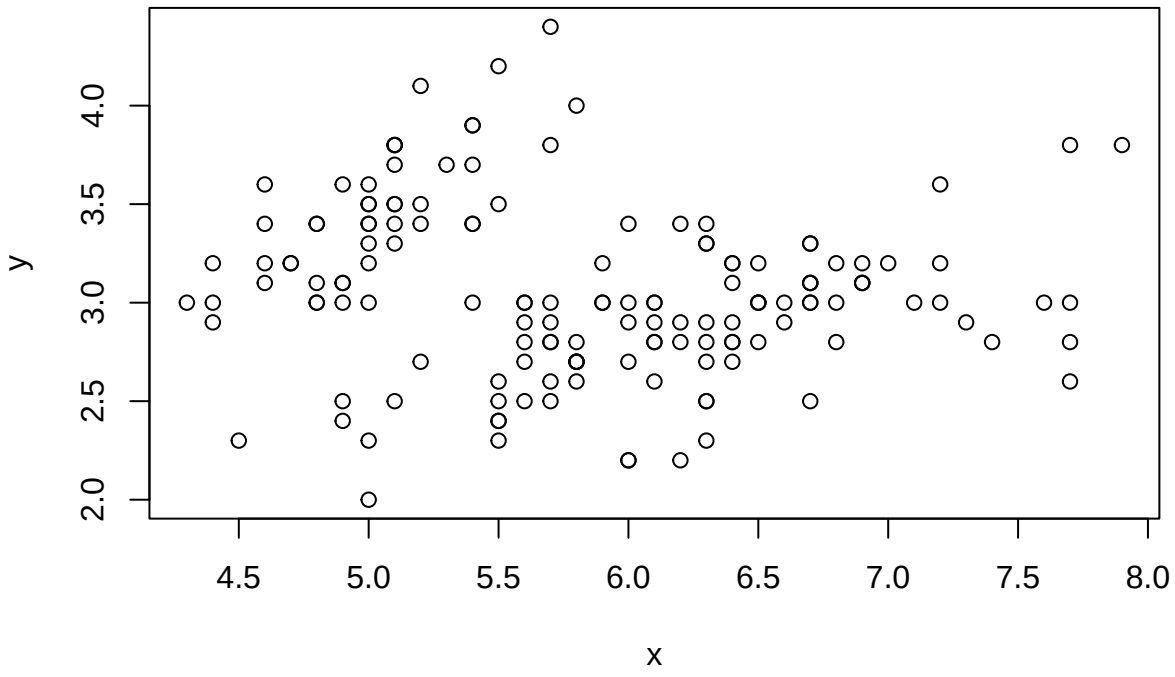
A scatter chart (or a scatter plot) is a chart that shows the relationship between two quantitative variables.

- very powerful techniques to investigate relationship or trend.

```
str(iris)
```

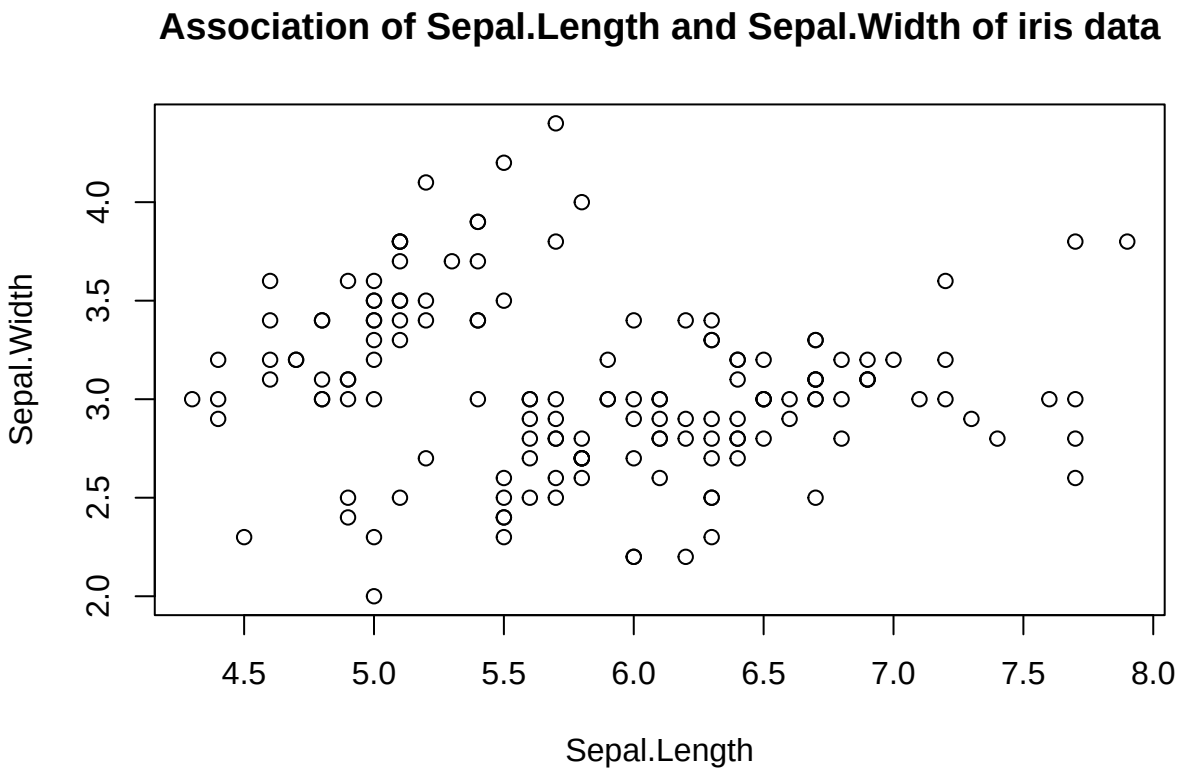
```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
x = iris$Sepal.Length
y= iris$Sepal.Width
plot(x, y)
```



1.4. Label the axes and title of the plot

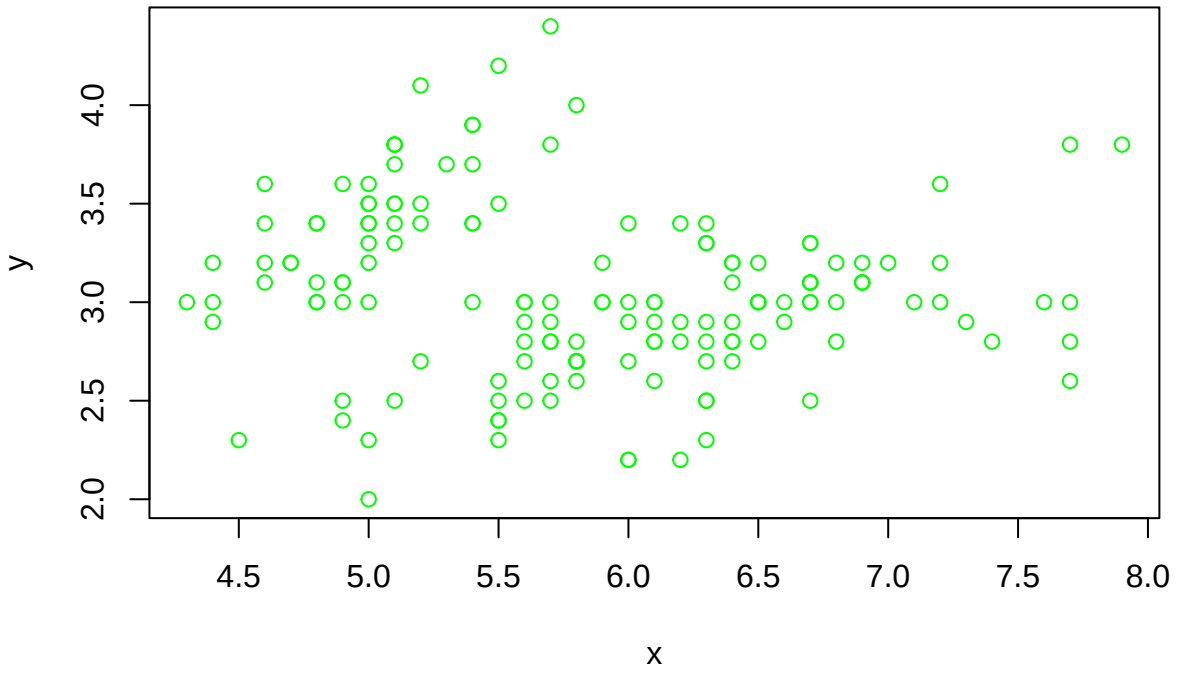
```r
x = iris$Sepal.Length
y= iris$Sepal.Width
plot(x, y, type = 'p', xlim = range(x), ylim = range(y),
     xlab = "Sepal.Length",
     ylab = "Sepal.Width",
     main = "Association of Sepal.Length and Sepal.Width of iris data")
```



1.5. Color of points

For more details see [http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf]

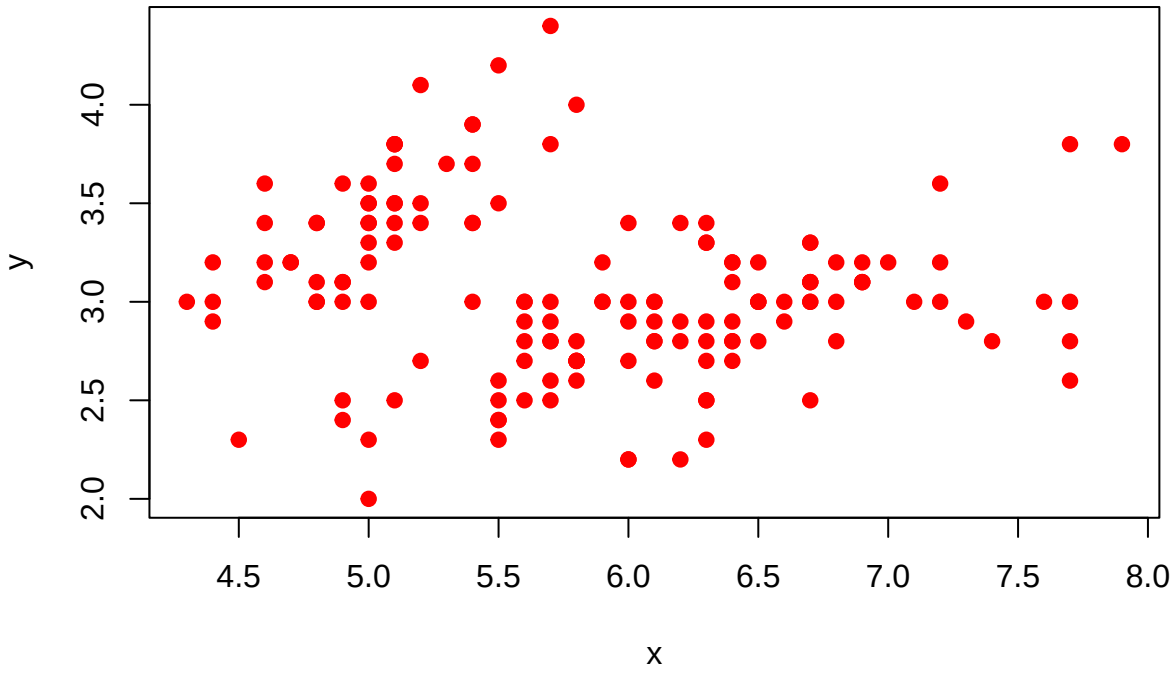
```
x = iris$Sepal.Length
y= iris$Sepal.Width
plot(x, y, col = 'green')
```



1.6. Plot character in R

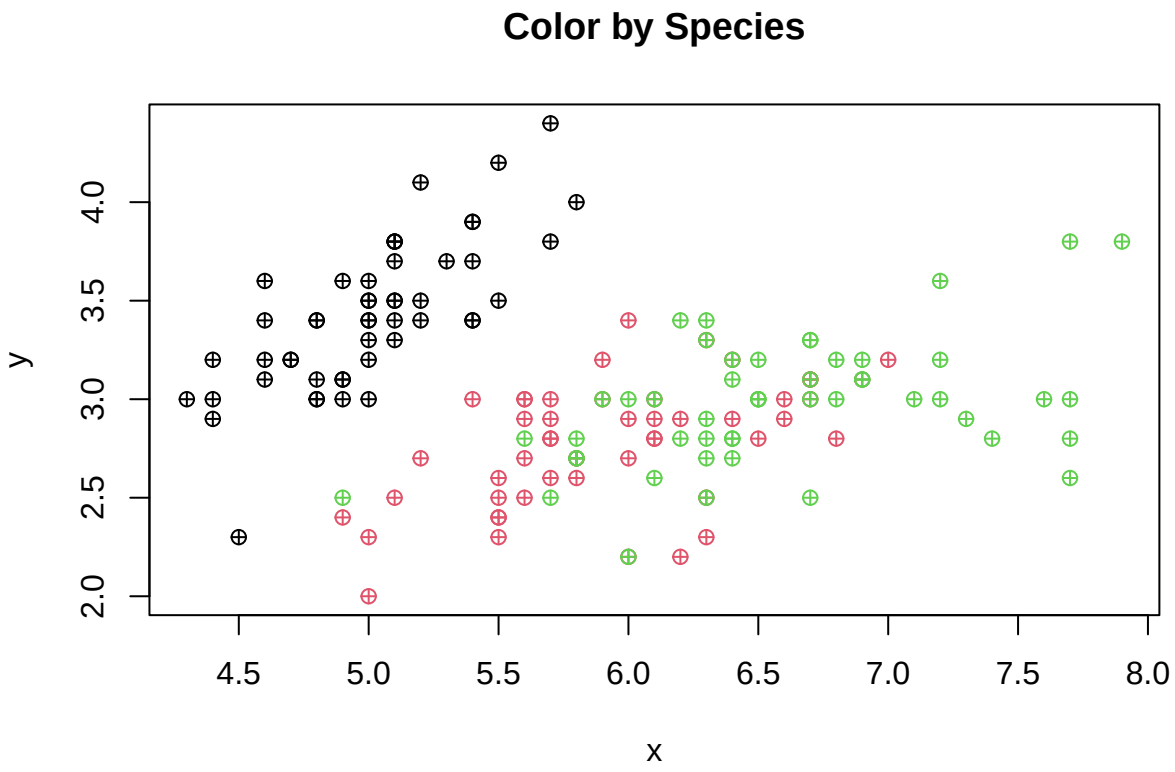
For more details see here [https://www.r-bloggers.com/2021/06/r-plot-pch-symbols-different-point-shapes-in-r/]

```
x = iris$Sepal.Length
y= iris$Sepal.Width
plot(x, y, col = 'red', pch = 19)
```



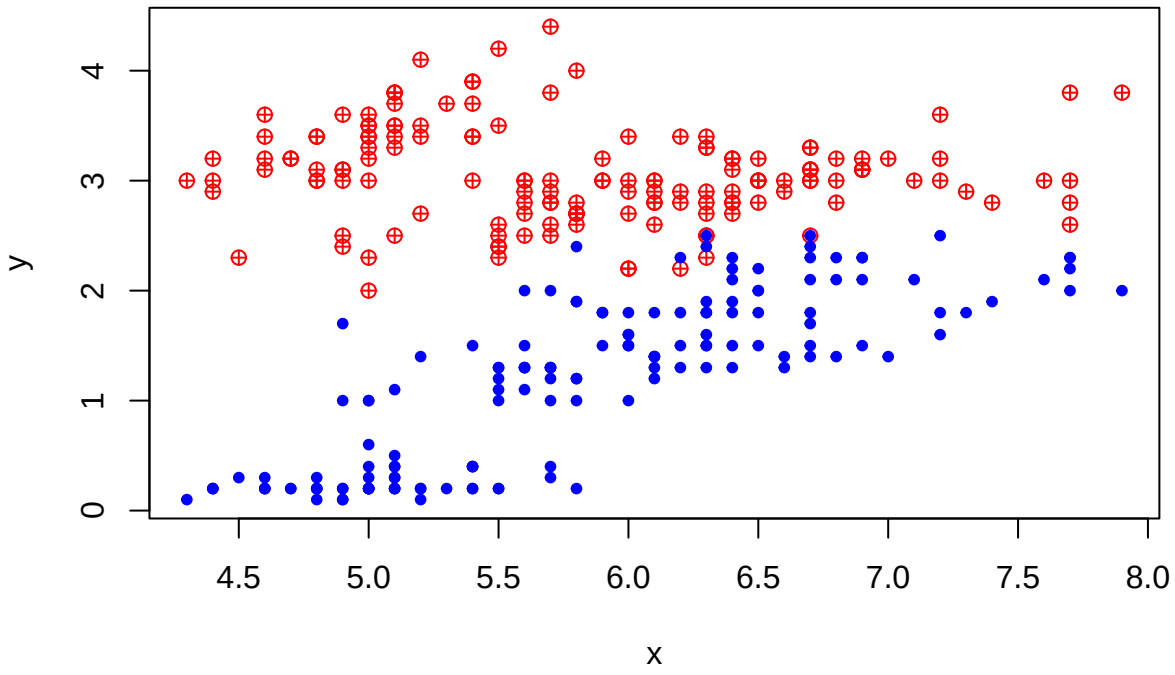
1.7. Plot by category

```
x = iris$Sepal.Length
y= iris$Sepal.Width
plot(x, y, col = iris$Species, pch = 10,
     main = "Color by Species")
```



1.8. Adding another y-variable in the same window (i.e. when your x-varaible is same)

```
x = iris$Sepal.Length
y= iris$Sepal.Width
y1 = iris$Petal.Width
plot(x, y, ylim = range(y, y1), col = 'red', pch = 10)
points(x, y1, col = 'blue', pch = 20)
```



1.9. Saving output files

- Save output files with default size
- figure type png, units = pixcels, length, width are in pixcels resolution
- figure type pdf, default dimension length, width are in inches
- See for more details in link

```
png("~/Desktop/iris.png")
x = iris$Sepal.Length
y= iris$Sepal.Width
y1 = iris$Petal.Width
plot(x, y, ylim = range(y, y1), col = 'red', pch = 10)
points(x, y1, col = 'blue', pch = 20)
dev.off()
```

```
## pdf
##   2
```

```
png("~/Desktop/iris.png", height = 400, width = 500, units = "px")
x = iris$Sepal.Length
y= iris$Sepal.Width
y1 = iris$Petal.Width
plot(x, y, ylim = range(y, y1), col = 'red', pch = 10)
points(x, y1, col = 'blue', pch = 20)
dev.off()
```

1.9.1 Save output files with specific size

```
## pdf
##   2
```

```
pdf("~/Desktop/iris.pdf", height = 6, width = 5)
x = iris$Sepal.Length
y= iris$Sepal.Width
y1 = iris$Petal.Width
plot(x, y, ylim = range(y, y1), col = 'red', pch = 10)
points(x, y1, col = 'blue', pch = 20)
dev.off()
```

```
## pdf
##   2
```

1.9.2 Combining plots

In the previous example we observed the association of Sepal.Width and Petal.Width with x-variable Sepal.Length. Now let's observe the association Sepal.Width and Petal.Width for different flower species in seperate windows.

It is very easy to combine multiple plots into one overall graph in R, using the par(mfrow = c(i, j)) .

```
par(mfrow = c(i, j)): combines the plots,
    i indicates number of rows,
    j indicates number of columns
```

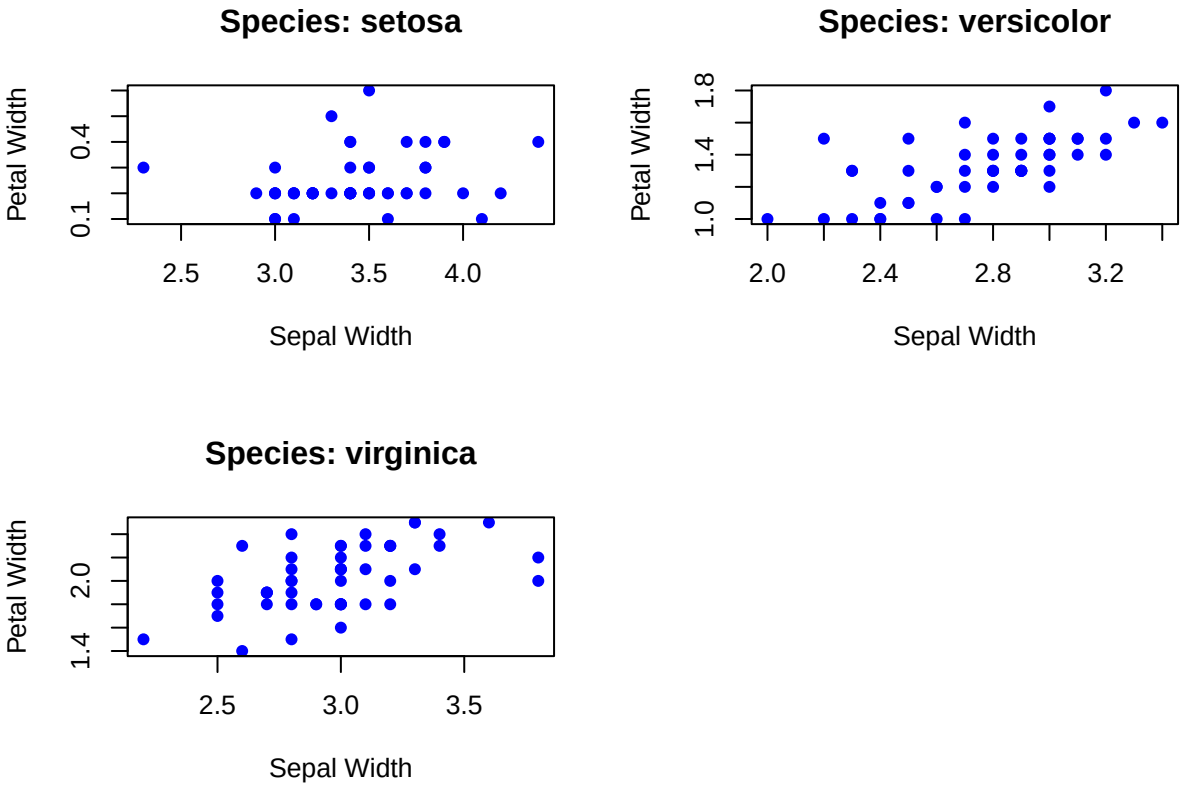
- Find the association of Sepal.Length with other varaibles

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
setosa_df <- iris %>% filter(Species == "setosa")
x1 <- setosa_df$Sepal.Width
y1 <- setosa_df$Petal.Width
versicolor_df <- iris %>% filter(Species == "versicolor")
x2 <- versicolor_df$Sepal.Width
y2 <- versicolor_df$Petal.Width
virginica_df <- iris %>% filter(Species == "virginica")
x3 <- virginica_df$Sepal.Width
y3 <- virginica_df$Petal.Width
```

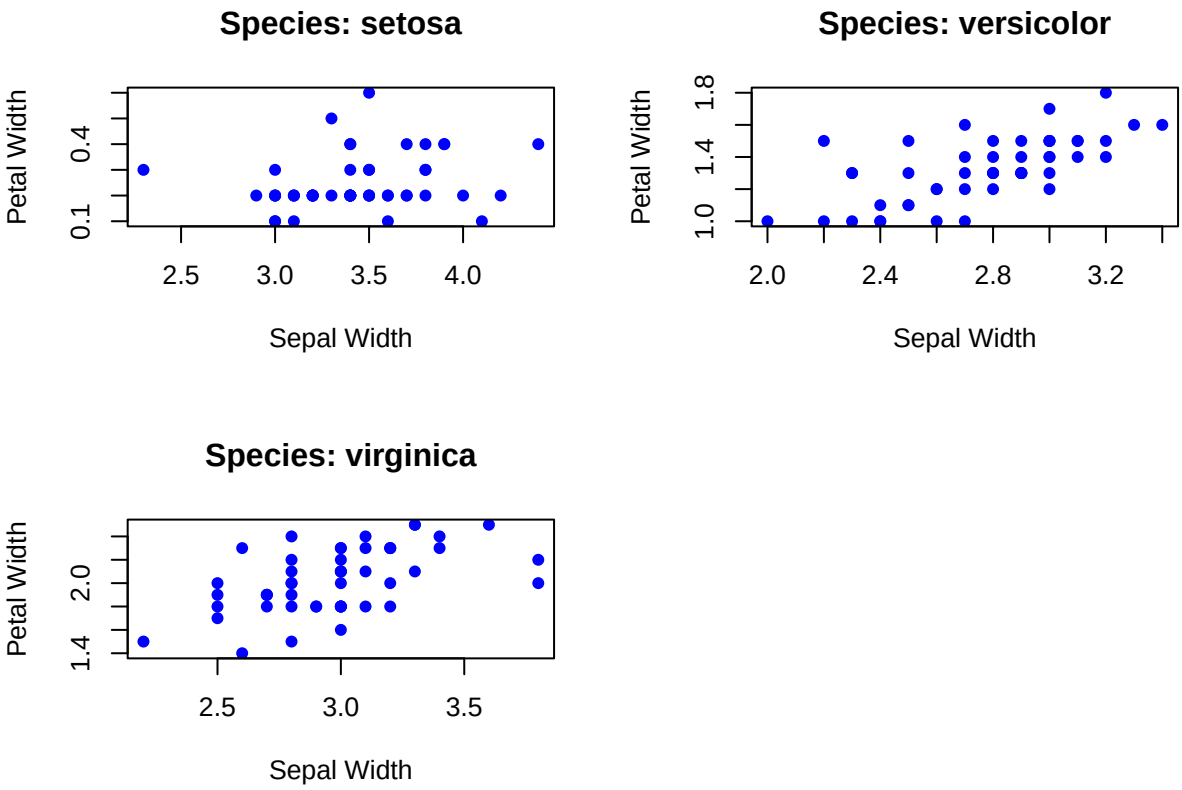
```r
# Set up the plotting window to have 2 rows and 1 column
par(mfrow = c(2, 2))
plot(x1, y1,
     main = "Species: setosa" ,
     xlab = "Sepal Width", ylab = "Petal Width",
       col = "blue", pch = 16)  # Blue dots with solid circles
plot(x2, y2,
     main = "Species: versicolor" ,
     xlab = "Sepal Width", ylab = "Petal Width",
       col = "blue", pch = 16)
plot(x3, y3,
     main = "Species: virginica" ,
     xlab = "Sepal Width", ylab = "Petal Width",
       col = "blue", pch = 16)
```



1.9.2.1 Same plot Using loop We are constructing the above same plot using loop as follows:

```r
# Get unique species names
species_list <- unique(iris$Species)
# Set up the plotting window to have 2 rows and 1 column
par(mfrow = c(2, 2))

# Loop through each species and create separate scatter plots
for (sp in species_list) {
  subset_data <- subset(iris, Species == sp)

  # Create scatter plot
```

```
  plot(subset_data$Sepal.Width, subset_data$Petal.Width,
       main = paste("Species:", sp),
       xlab = "Sepal Width", ylab = "Petal Width",
       col = "blue", pch = 16)  # Blue dots with solid circles
}
```



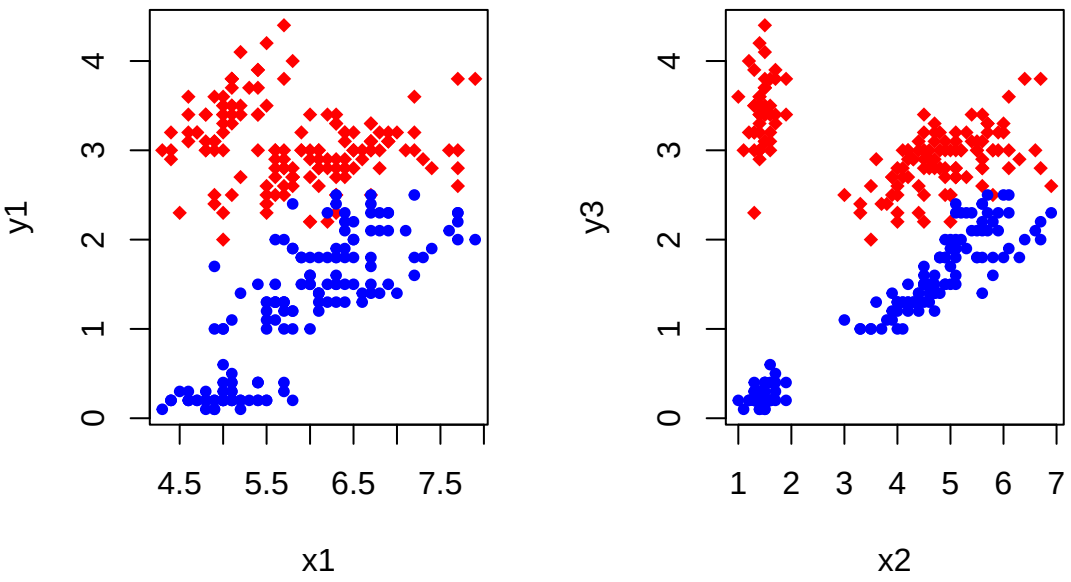
1.9.3 Figure size or window in R

It should be noted that in RStudio the graph will be displayed in the pane layout and figure size can be adjusted in r-chunk by assigning values for fig.width and fig.height such as

```
{r, fig.width=6, fig.height=4}

# Your plotting code here

plot(x, y)
```

```
par(mfrow = c(1, 2))
#plot 1
x1 = iris$Sepal.Length
y1= iris$Sepal.Width
y2 = iris$Petal.Width
plot(x1, y1, ylim = range(c(y1, y2)), col = 'red', pch = 18)
points(x1, y2, col = 'blue', pch = 20)
#plot 2
x2= iris$Petal.Length
```

```r
y3 = iris$Sepal.Width
y4 = iris$Petal.Width
plot(x2, y3, ylim = range(c(y3, y4)), col = 'red', pch = 18)
points(x2, y4, col = 'blue', pch = 20)
```



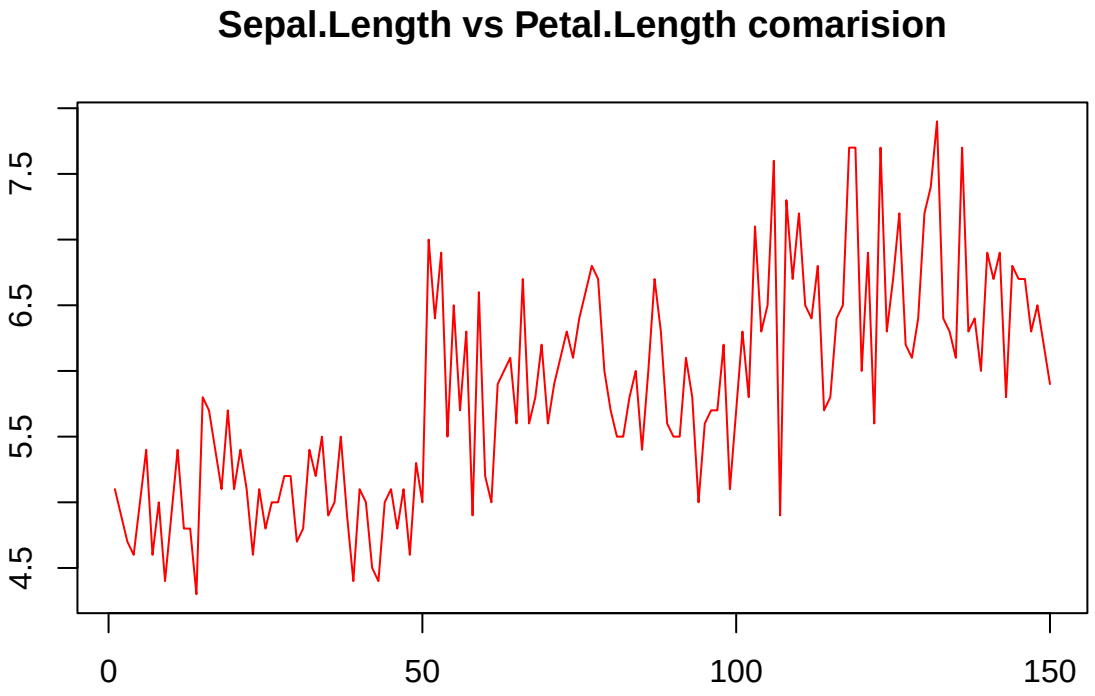
---

2. Line chart or trace plot

- Mostly used for time series data.
- Useful to check the association of variables by the observations
- lty stands for line type such as dotted, solid see [https://r-charts.com/base-r/line-types/]

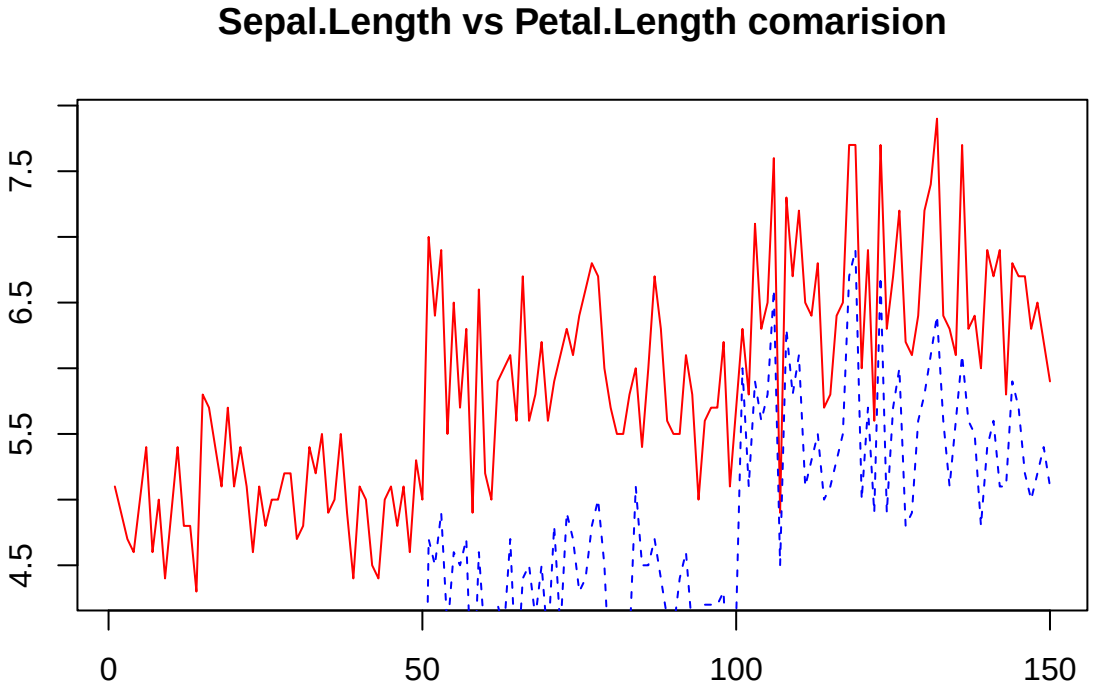
2.1. Line chart of one y-variable

```r
x1 = iris$Sepal.Length
x2 = iris$Petal.Length
idx = 1: length(x1)
plot(idx, x1, type = "l", xlab = "", ylab = "", col = 'red', lty = 1,
     main = "Sepal.Length vs Petal.Length comarision")
```



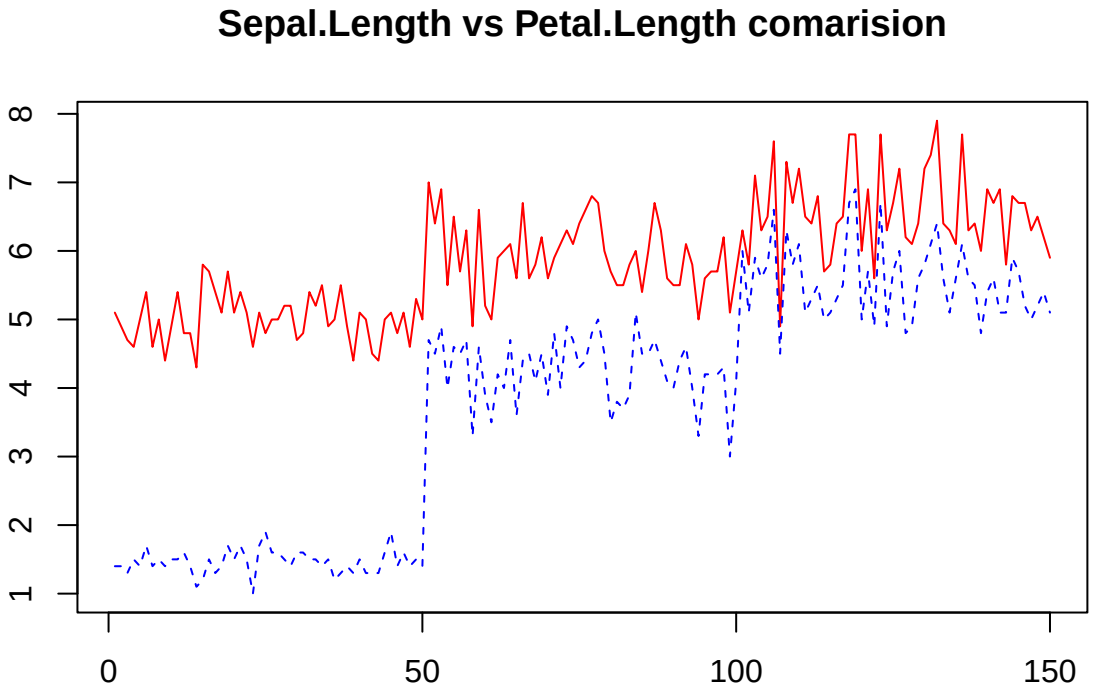
2.2. Line chart of two y-variables comparisions

```
x1 = iris$Sepal.Length
x2 = iris$Petal.Length
idx = 1: length(x1)
plot(idx, x1, type = "l", xlab = "", ylab = "", col = 'red', lty = 1,
     main = "Sepal.Length vs Petal.Length comarision")
lines(idx, x2, type = "l", xlab = "", ylab = "", lty = 2, col = 'blue')
```



- Let's adjust the limit so that we can clearly see Petal.Length

```
x1 = iris$Sepal.Length
x2 = iris$Petal.Length
idx = 1: length(x1)
plot(idx, x1, type = "l", xlab = "", ylab = "", ylim = range(x1, x2),
     lty = 1, col = 'red', main = "Sepal.Length vs Petal.Length comarision")
lines(idx, x2, type = "l", xlab = "", ylab = "", lty = 2, col = 'blue')
```

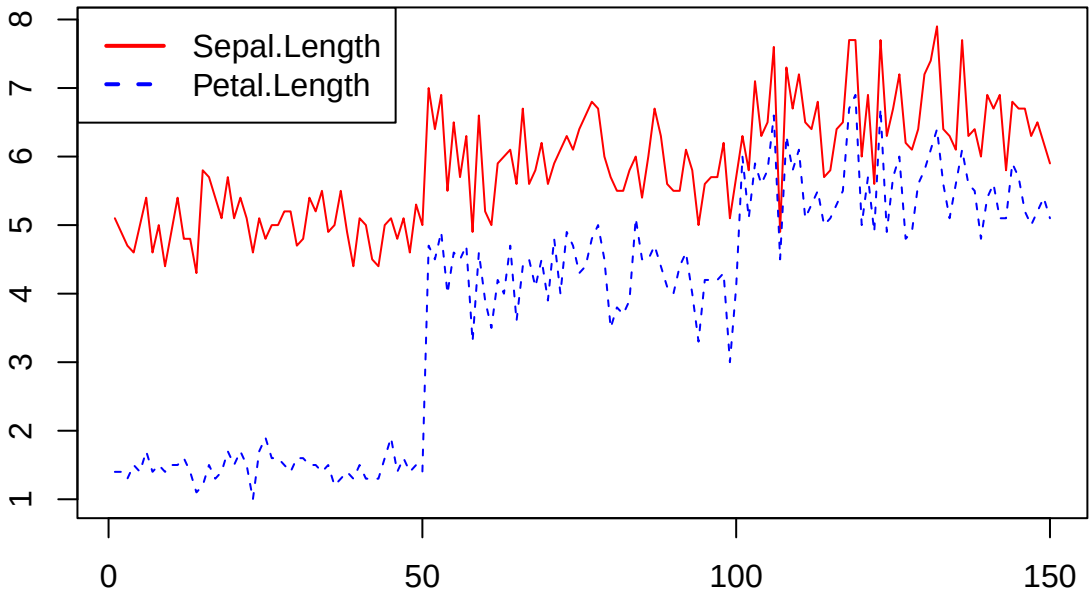


2.3. Define legends in R

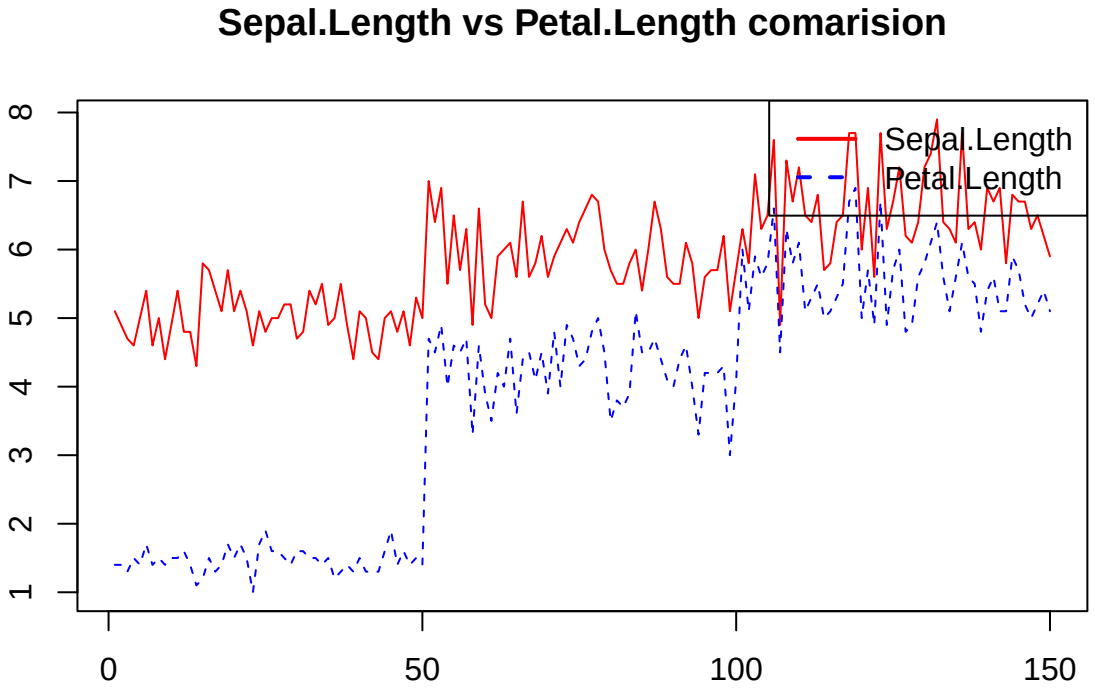
When we are comparing multiple variables using trace plot or scatter plot, it is vary hard to identify the the visual of related variable. So, assigning legend is important in such of cases.

For more details see [https://r-coder.com/add-legend-r/]

```
x1 = iris$Sepal.Length
x2 = iris$Petal.Length
idx = 1: length(x1)
plot(idx, x1, type = "l", xlab = "", ylab = "", ylim = range(x1, x2),
     lty = 1, col = 'red', main = "Sepal.Length vs Petal.Length comarision")
lines(idx, x2, type = "l", xlab = "", ylab = "", lty = 2, col = 'blue')

legend(x = "topleft",          # Position
       legend = c("Sepal.Length", "Petal.Length"),  # Legend texts
       lty = c(1, 2),           # Line types
       col = c('red', 'blue'),           # Line colors
       lwd = 2)                 # Line width
```




```r
# Change the legend to the Right
x1 = iris$Sepal.Length
x2 = iris$Petal.Length
idx = 1: length(x1)
plot(idx, x1, type = "l", xlab = "", ylab = "", ylim = range(x1, x2),
     lty = 1, col = 'red', main = "Sepal.Length vs Petal.Length comarision")
lines(idx, x2, type = "l", xlab = "", ylab = "", lty = 2, col = 'blue')
legend(x = "topright",          # Position
       legend = c("Sepal.Length", "Petal.Length"),  # Legend texts
       inset = c(0, 0),
       lty = c(1, 2),           # Line types
       col = c('red', 'blue'),           # Line colors
       lwd = 2)
```



---

3. Bar plot in R

A bar plot is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to their corresponding values (or count). The bars can be plotted vertically or horizontally.

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

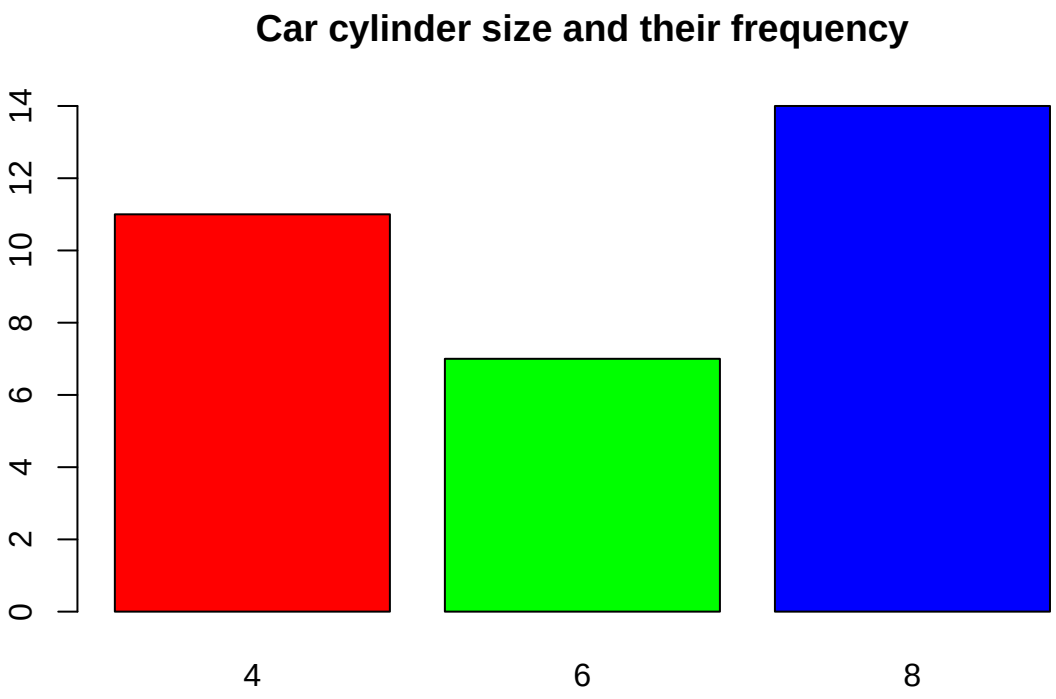
3.1 Bar Plot of one Category

```
cyl_freq_df <-table(mtcars$cyl)
print(cyl_freq_df)
```

```
##
```

```
##  4  6  8
## 11  7 14
```

```r
barplot(cyl_freq_df, col = rainbow(3), main = "Car cylinder size and their frequency")
```



3.1 Absolute frequency vs Relative frequency

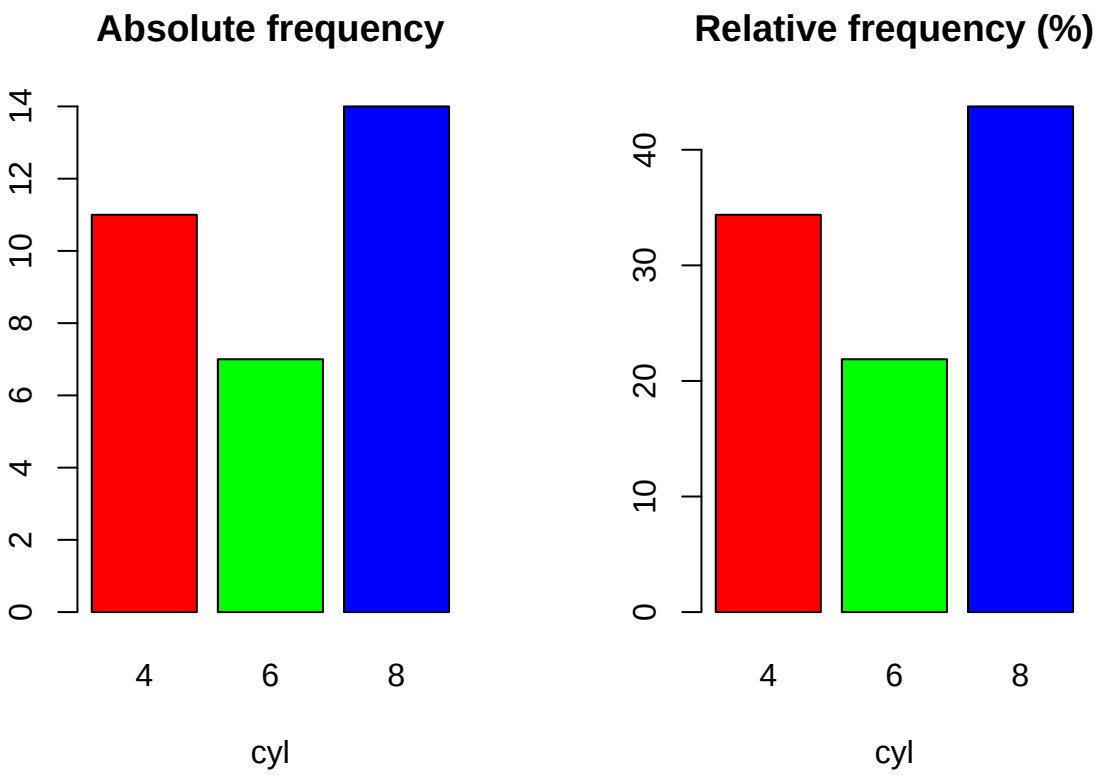
```r
cyl_freq_df <- as.data.frame(cyl_freq_df)
cyl_freq_df
```

```
##   Var1 Freq
## 1    4   11
## 2    6    7
## 3    8   14
```

```r
prop.table(cyl_freq_df$Freq)*100
```

```
## [1] 34.375 21.875 43.750
```

```r
# One row, two columns
par(mfrow = c(1, 2))
# Absolute frequency barplot
barplot(height = cyl_freq_df$Freq, names = cyl_freq_df$Var1, xlab = "cyl",
        main = "Absolute frequency",
        col = rainbow(3))

# Relative frequency barplot
barplot(height = prop.table(cyl_freq_df$Freq)*100, names = cyl_freq_df$Var1,
        xlab = "cyl", main = "Relative frequency (%)",
        col = rainbow(3))
```



3.3 Application in Boston-311 Service data

```
Boston311_2023_data =
read.csv("https://data.boston.gov/dataset/8048697b-ad64-4bfc-b090-ee00169f2323/resource/e6013a93-1321-4
```

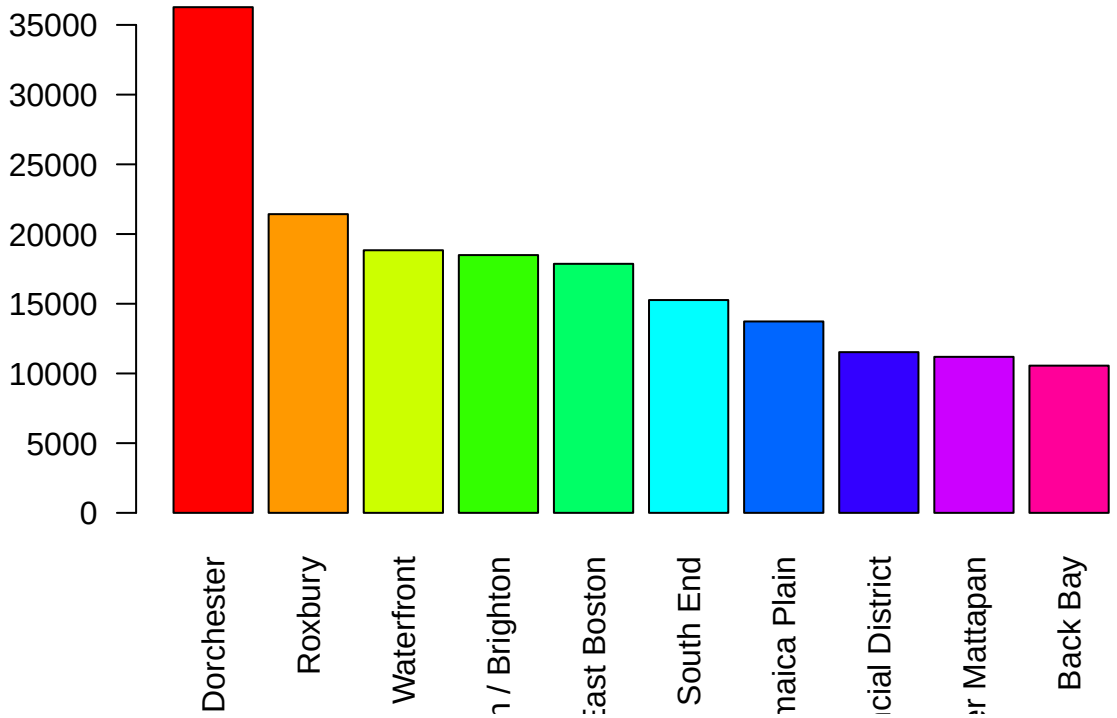
- Which neighborhood has maximum number of complaints of type "Parking Enforcement"?

```
library(stringr)
library(dplyr)
Boston311_2023_data$Parking_Enforcement_status <- str_detect(Boston311_2023_data$case_title,
                                                             regex("\\bParking Enforcement\\b"))
Parking_Enforcement_by_nbd <- Boston311_2023_data %>%
  group_by(neighborhood) %>%
  summarise(nbd_count_Parking_Enforcement = n()) %>%
  arrange(desc(nbd_count_Parking_Enforcement))
head(Parking_Enforcement_by_nbd, 10)
```

```
## # A tibble: 10 x 2
##    neighborhood                          nbd_count_Parking_Enforcement
##    <chr>                                                         <int>
##  1 Dorchester                                                    36272
##  2 Roxbury                                                       21426
##  3 South Boston / South Boston Waterfront                        18835
##  4 Allston / Brighton                                            18490
##  5 East Boston                                                   17862
##  6 South End                                                     15265
```

```
##  7 Jamaica Plain                                  13728
##  8 Downtown / Financial District                  11526
##  9 Greater Mattapan                               11191
## 10 Back Bay                                       10559
```

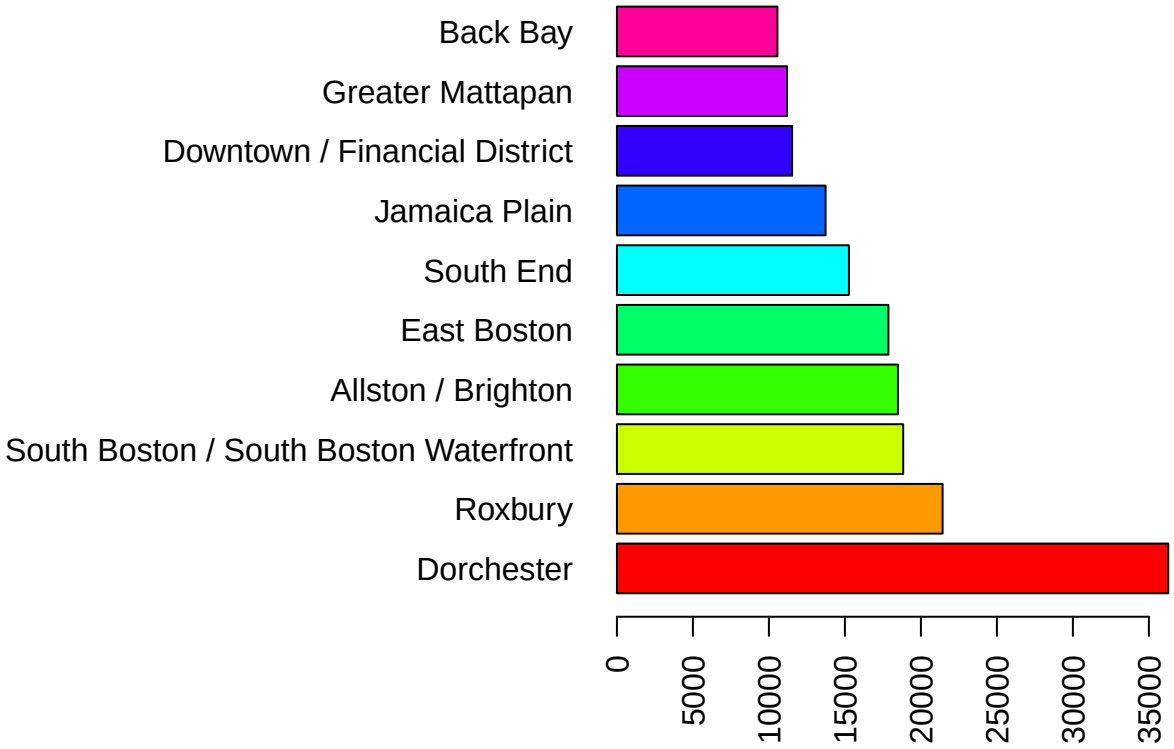
```r
top_10_nbd = Parking_Enforcement_by_nbd[1:10, ]
barplot(names = top_10_nbd$neighborhood, height = top_10_nbd$nbd_count_Parking_Enforcement,
 col = rainbow(10), las = 2)
```




```r
#las = 1, group names printed horizontally
#las = 2, group names printed vertically
```

```
par(mar, mgp, las)
par(mar=c(5.1, 4.1, 4.1, 2.1), mgp=c(3, 1, 0), las=0)
par sets or adjusts plotting parameters. Here we consider the following three parameters: margin size (
mar - A numeric vector of length 4, which sets the margin sizes in the following order: bottom, left, t
mgp - A numeric vector of length 3, which sets the axis label locations relative to the edge of the inn
las - A numeric value indicating the orientation of the tick mark labels and any other text added to a
```

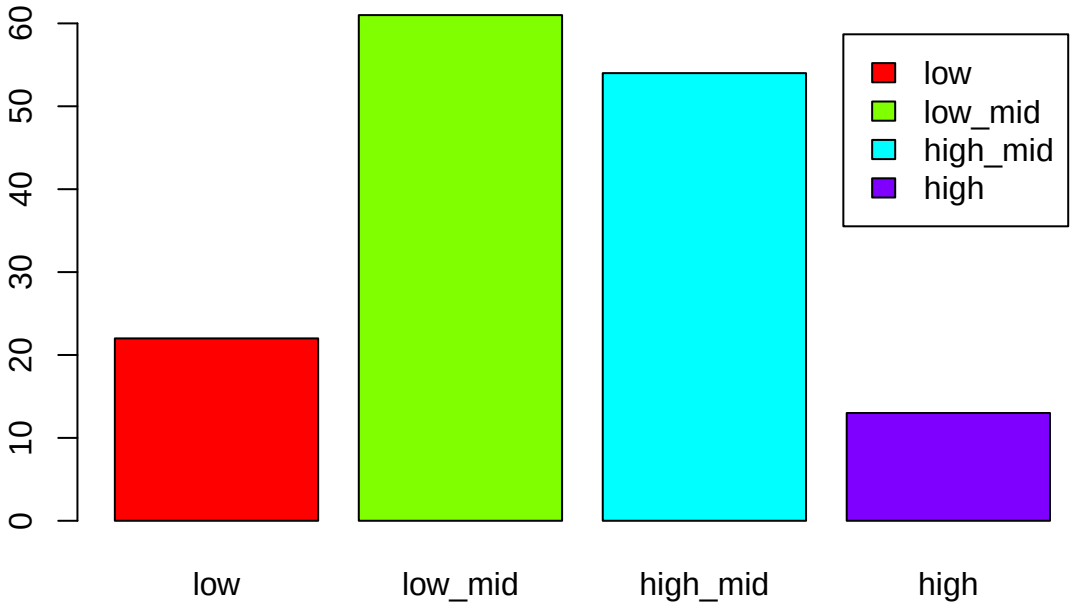
```r
### Horizontal barplot
par(mar = c(4, 16, 2, 2))
top_10_nbd = Parking_Enforcement_by_nbd[1:10, ]
barplot(names = top_10_nbd$neighborhood, height = top_10_nbd$nbd_count_Parking_Enforcement,
 col = rainbow(10), horiz = TRUE, las = 2)
```



3.4. Barplot for continuous variable after converting a new categorical variable

```
### Barplot for continuous variable
var1 = iris$Sepal.Length
cut_off = c(0, 5, 6, 7 , 8)
catgory = c("low", "low_mid", "high_mid", "high")
Sepal_Len_cat1 = cut(var1, breaks = cut_off, include.lowest = TRUE, right = FALSE, labels = catgory)
iris_new = cbind(iris, Sepal_Len_cat1)
```

```
barplot(table(iris_new$Sepal_Len_cat1), col = rainbow(4),
        legend.text = levels(iris_new$Sepal_Len_cat1))# With Legend
```



3.4. Multiple barplot in R

Compute the mean horse power(HP) by transmission type and cyl and then plot them.

```
data(mtcars)
str(mtcars)
```

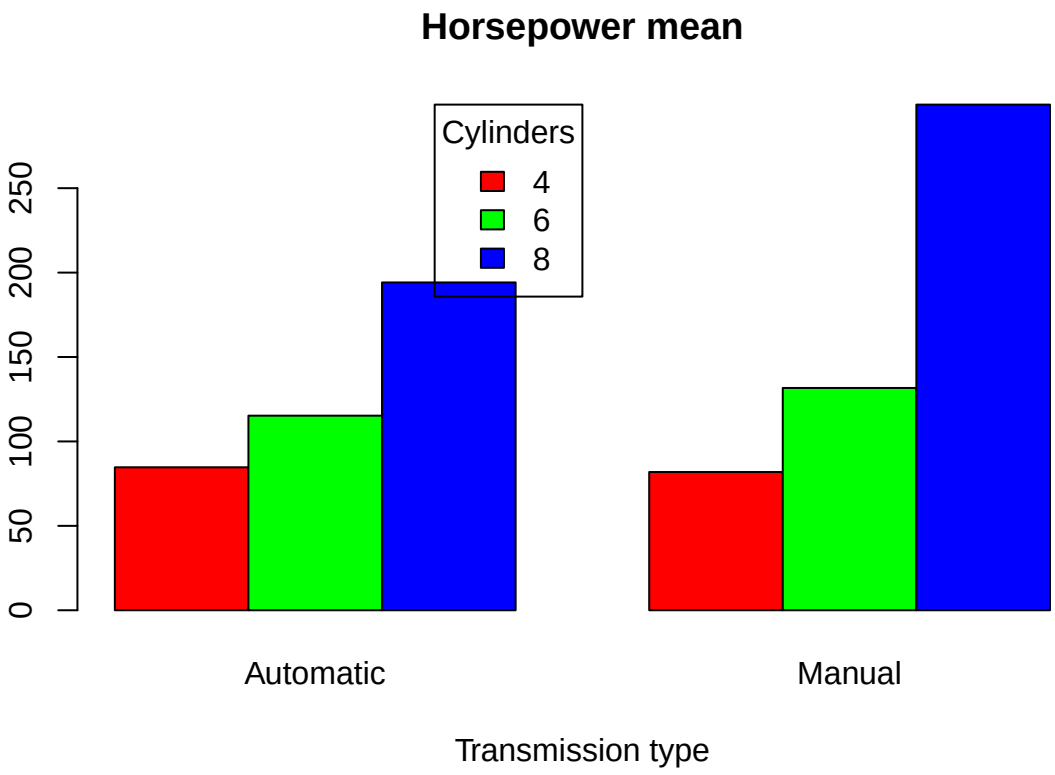
```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
# Convert am to a factor with appropriate labels
mtcars$am <- factor(mtcars$am, levels = c(0, 1), labels = c("Automatic", "Manual"))

# Compute the mean horsepower grouped by cylinders and transmission
summary_data <- tapply(mtcars$hp, list(cylinders = mtcars$cyl, transmission = mtcars$am), mean, na.rm =

# Print the summary table
print(summary_data)
```

```
##          transmission
## cylinders Automatic   Manual
##         4  84.66667  81.8750
```

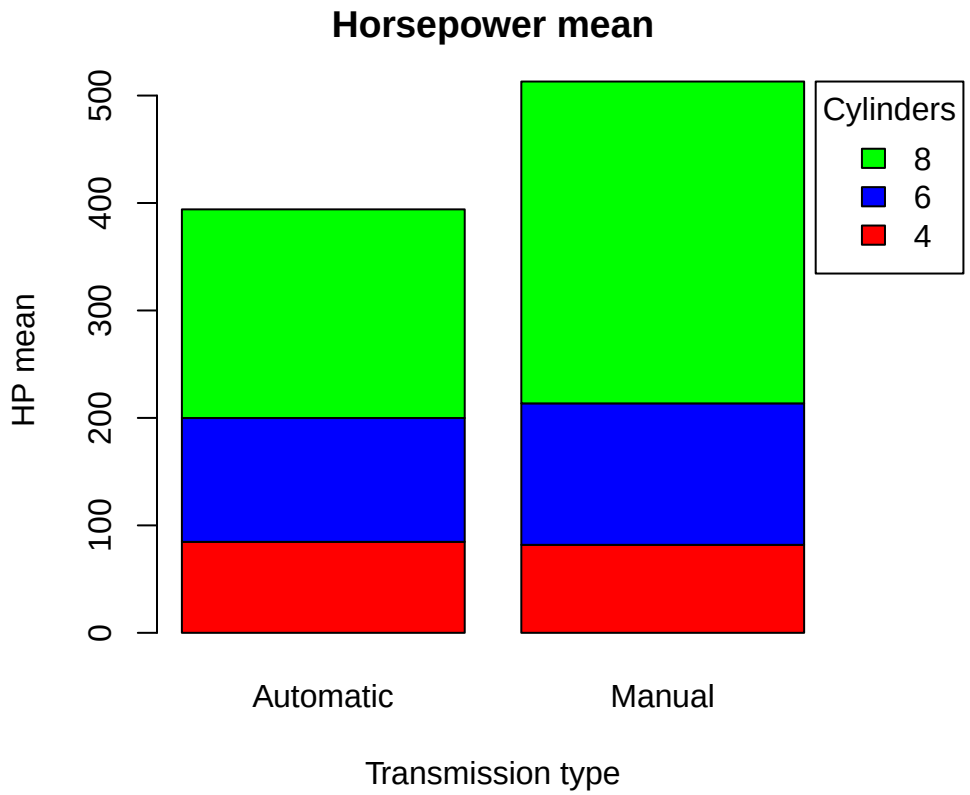
```
##          6 115.25000 131.6667
##          8 194.16667 299.5000
```

```r
barplot(summary_data,  xlab = "Transmission type",
        main = "Horsepower mean",
        col = rainbow(3),
        beside = TRUE,
        legend.text = rownames(summary_data),
        args.legend = list(title = "Cylinders", x = "topright",
                           inset = c(0.5, 0)))
```



3.5 Stacked barplot in R

```r
par(mar = c(5, 5, 3, 10), las = 0)
barplot(summary_data,
        main = "Horsepower mean",
        xlab = "Transmission type", ylab = "HP mean",
        col = c('red', 'blue', 'green'),
        legend.text = rownames(summary_data),
        beside = FALSE, # Stacked bars (default)
        args.legend = list(title = "Cylinders", x = "topright",
                           inset = c(-0.2, 0)))
```



---

4. Pie-chart

A pie chart is used to represent data in numerical proportions. Pie chart in R is created using pie() function.

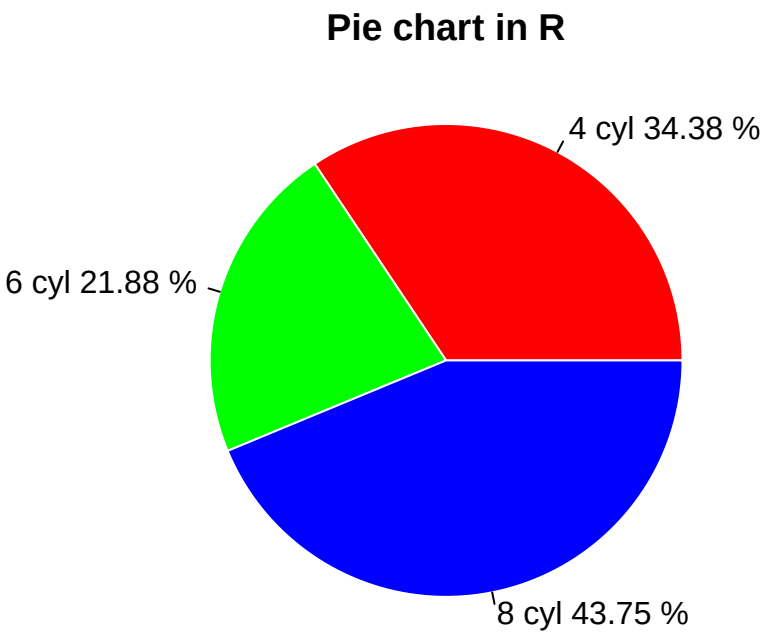
```
# cyl-wise distribution of data using pie-chart
count_cars <- mtcars %>%
  group_by(cyl) %>%
  summarise(count = n())
count_cars
```

```
## # A tibble: 3 x 2
##     cyl count
##   <dbl> <int>
## 1     4    11
## 2     6     7
## 3     8    14
```

For hcl.colors see ["https://blog.r-project.org/2019/04/01/hcl-based-color-palettes-in-grdevices/"]

```
car_type <- paste(count_cars$cyl, "cyl")
count <- count_cars$count

# calculating percentage participation
perc <- round(prop.table(count)*100,2)
```

```r
# add frequency or proportion to country names to create labels
labels <- paste(car_type, perc,'%')
pie(count, labels = labels, radius = 1, col = rainbow(3),
    border = 'white', main = "Pie chart in R")
```



---

5. Histogram

Histogram is the most widely used graph to represent quantitative (or numerical) data mostly for the continuous in nature.

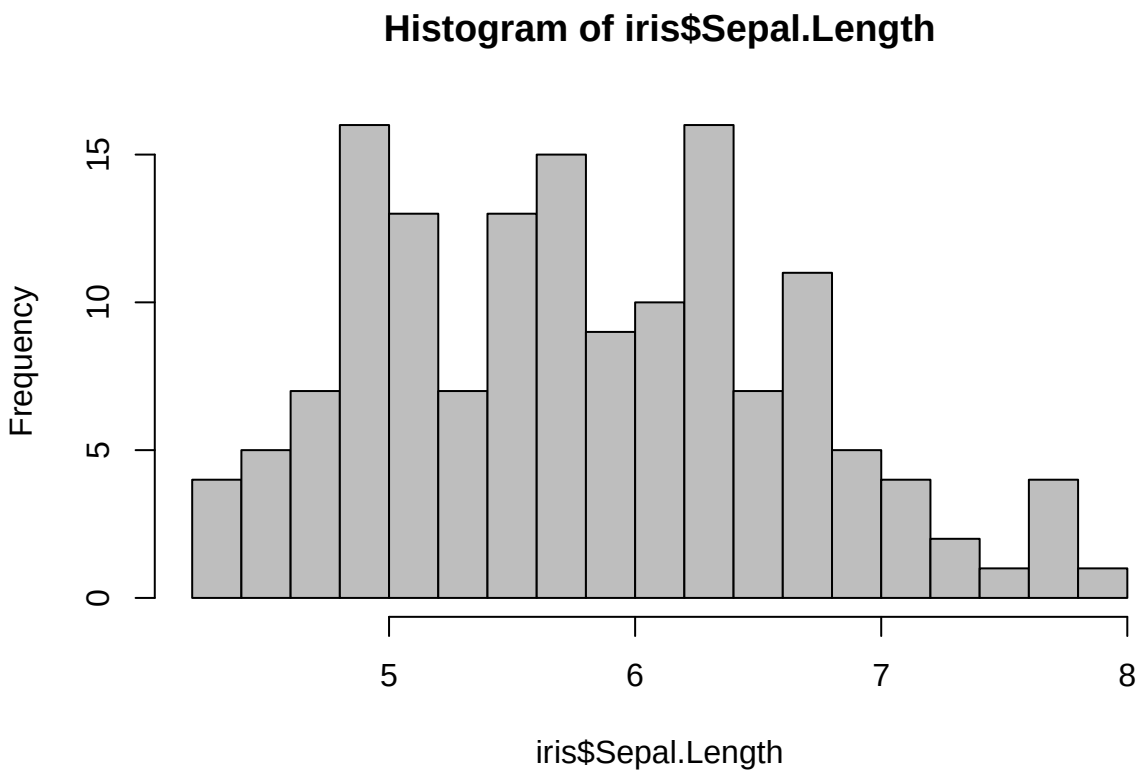
- This type of graph is useful to make an idea of the shape or distribution of the given data
- It provides an idea of the density of the underlying distribution of the data,
- It is often useful for density estimation,

Syntax

```r
hist(x,....)
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = NULL, border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

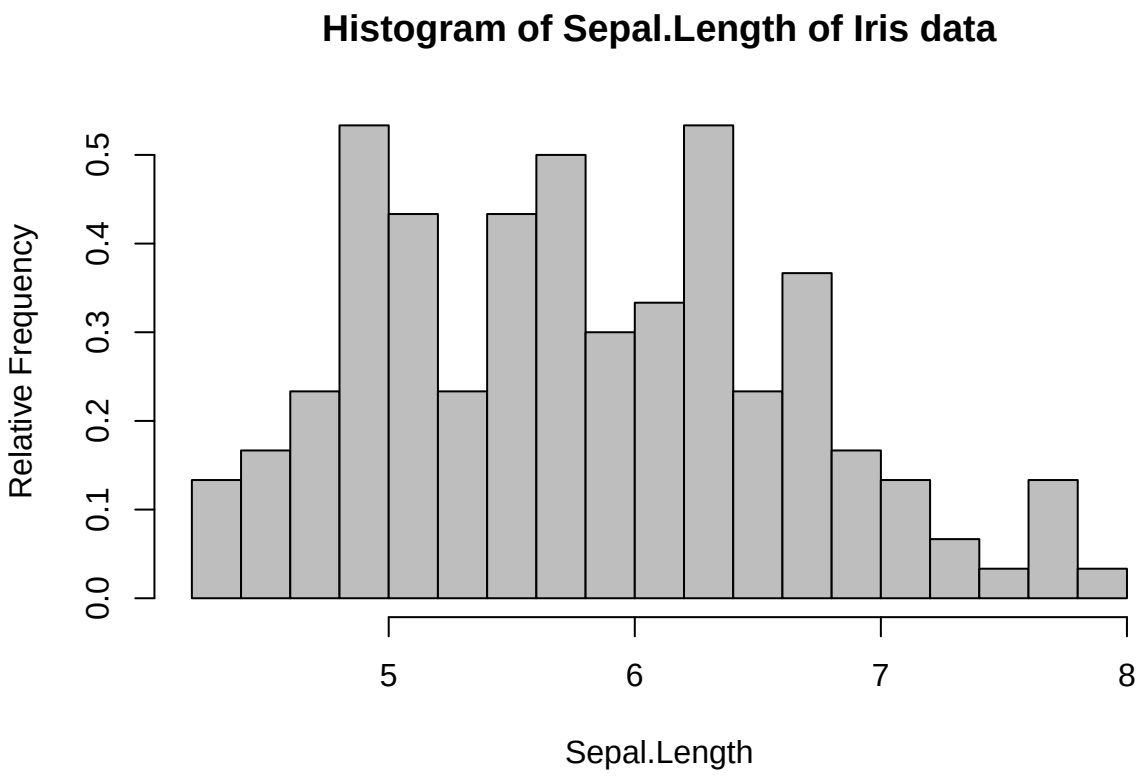
5.1. Histogram of absolute Frequency

```
hist(iris$Sepal.Length, breaks = 20, col = 'gray', probability = FALSE)
```



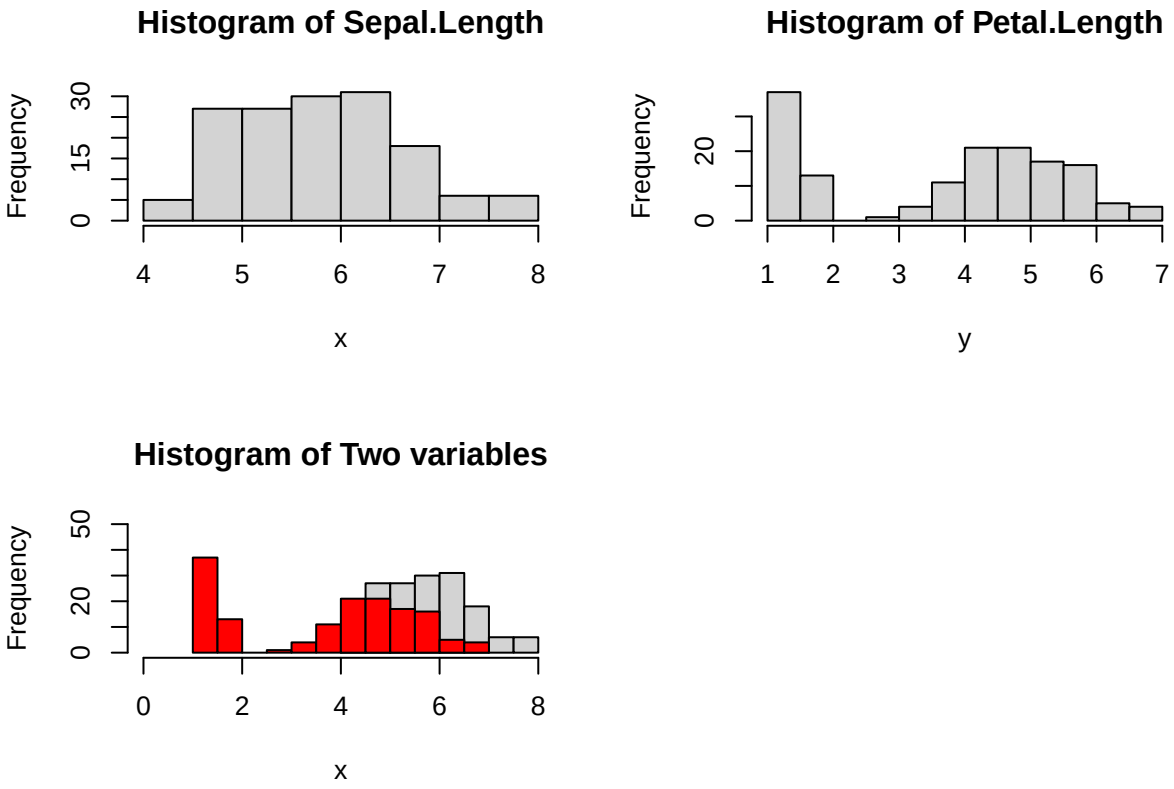
5.2. Histogram with relative frequency

```
hist(iris$Sepal.Length, breaks = 15, xlab = 'Sepal.Length',
     ylab = 'Relative Frequency',probability = TRUE, col = 'gray',
     main = "Histogram of Sepal.Length of Iris data")
```



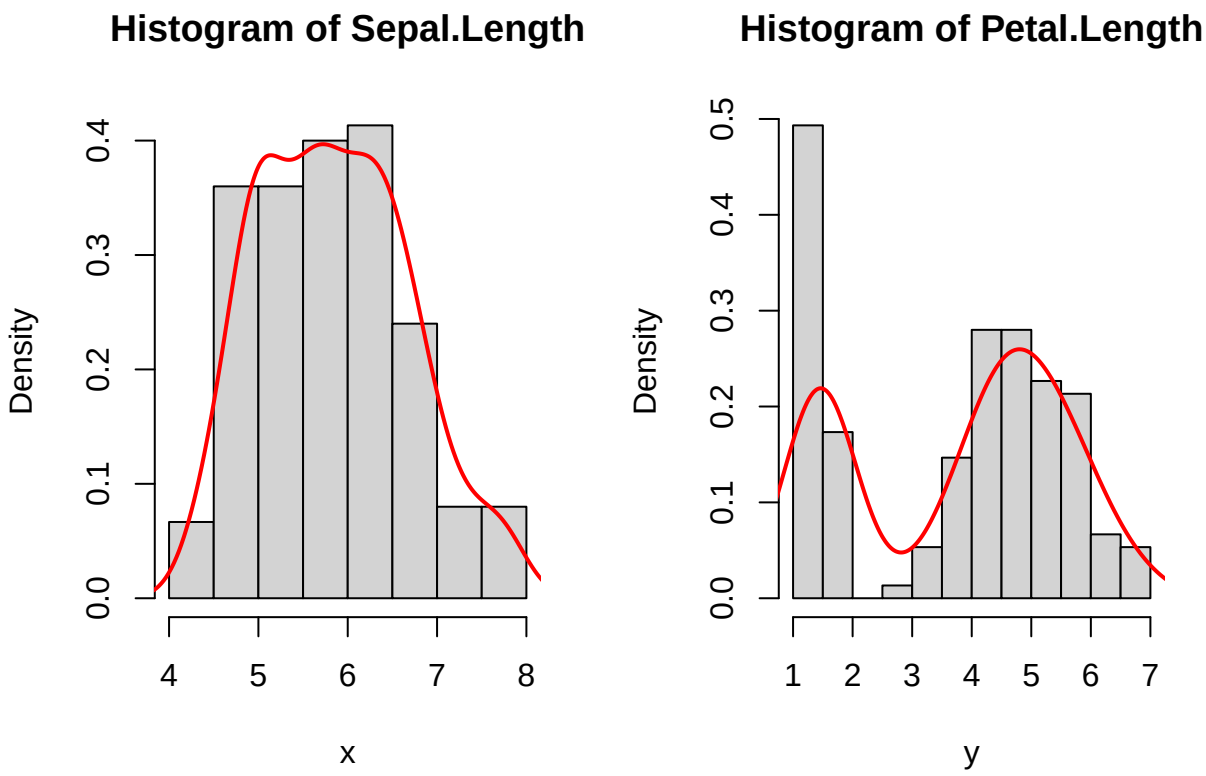
5.3. Histogram with two variables

```r
par(mfrow = c(2, 2))
x <- iris$Sepal.Length    # First group
y <- iris$Petal.Length    # Second group

hist(x,  main = "Histogram of Sepal.Length")
hist(y,  main = "Histogram of Petal.Length")

# Combine plot
hist(x, xlim = c(0, 8),ylim = c(0, 50),  main = "Histogram of Two variables")
hist(y, add = TRUE, col = rgb(1, 0, 0, alpha = 1)) # alpha is the trasparent parameter.
```



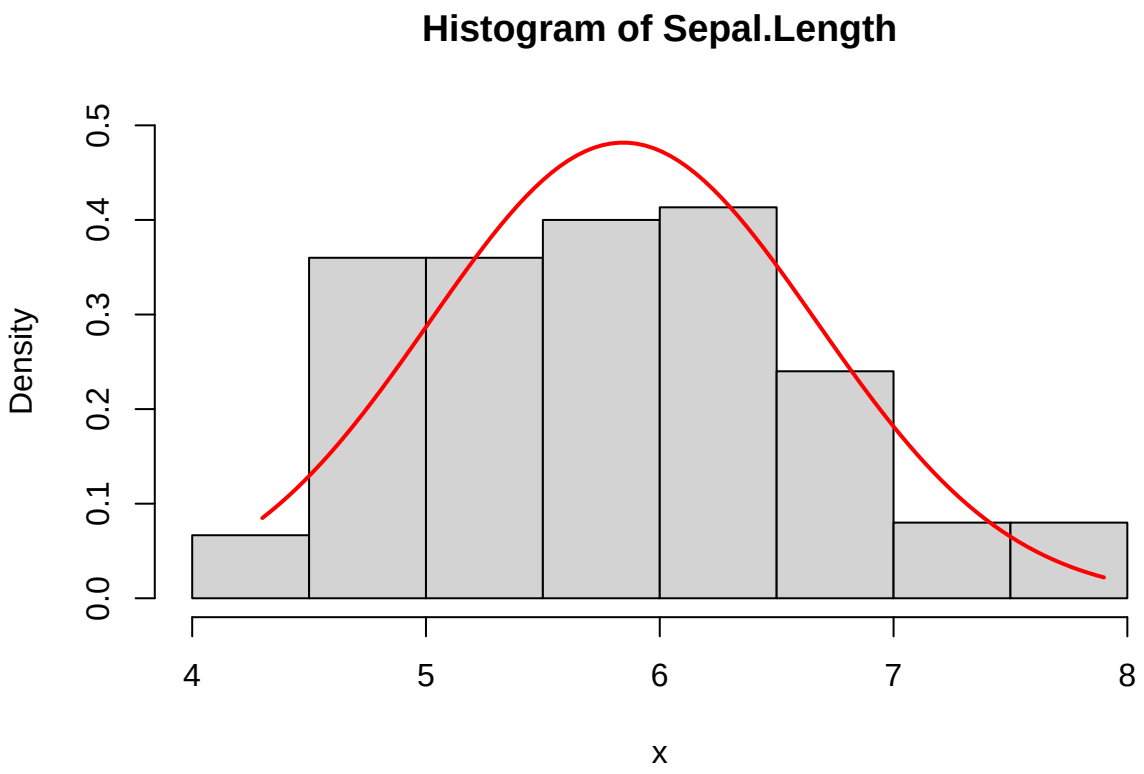
5.4. Add kernel density to histogram (non-parametric curve)

```
par(mfrow = c(1, 2))
x <- iris$Sepal.Length    # First group
y <- iris$Petal.Length    # Second group

hist(x, probability = TRUE,  main = "Histogram of Sepal.Length")
lines(density(x), lwd = 2, col = 'red')
hist(y,  probability = TRUE, main = "Histogram of Petal.Length")
lines(density(y), lwd = 2, col = 'red')
```



5.5. Add normal density to histogram

```
x <- iris$Sepal.Length    # First group
hist(x, ylim = c(0, 0.5), probability = TRUE,
     main = "Histogram of Sepal.Length")
x_val = seq(min(x), max(x), length.out = 100) # create a sequence of 100 numbers between the range of v
f_val = dnorm(x_val, mean = mean(x), sd = sd(x)) # create 100 normal density values corresponding to x_
lines(x_val, f_val, lwd = 2, col = 'red')
```



---

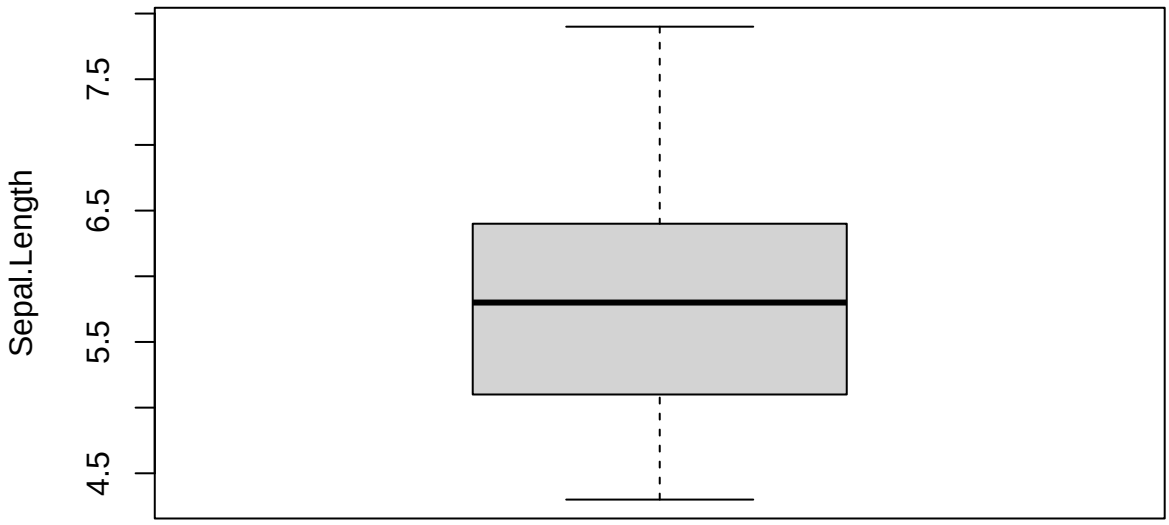
6. Box plot in R

Box plots (Chambers 1983) are an excellent tools for detecting and illustrating location and variation changes between different groups of data.

- It allows us to summarize the main characteristics of the data such as position, dispersion, skewness
- It is a very useful tool to identify potential data outliers.
- Positive Skewness (Right-Skewed):
  - upper whisker is longer in a vertical box plot,
  - right whisker is longer in a horizontal box plot
  - median is closer to other side
- Negative skewness: Longer whisker plot to the down for vertical (and to the left for the horizontal) from the box.
  - lower whisker is longer in a vertical box plot,
  - left whisker is longer in a horizontal box plot
  - median is closer to other side

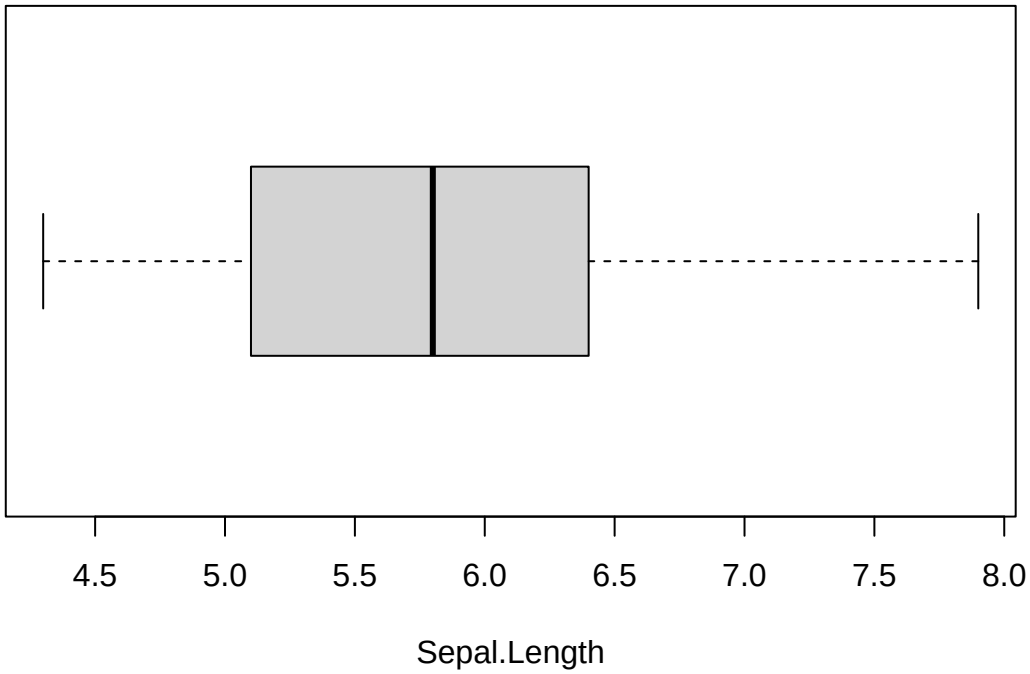
6.1 Vetrical boxplot (Default)

```
boxplot(x, ylab = "Sepal.Length")
```



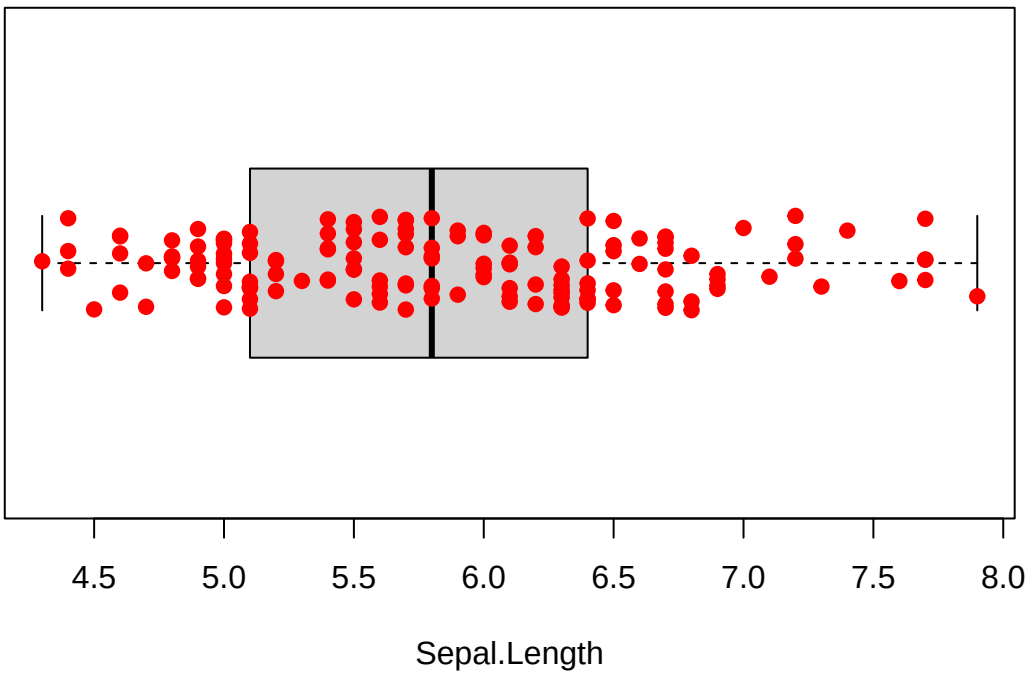
6.2 Horzontal box plot

```
boxplot(x, xlab = "Sepal.Length", horizontal = TRUE)
```



6.3. Box plot with points

```
boxplot(x, xlab = "Sepal.Length", horizontal = TRUE)
stripchart(x, method = "jitter", pch = 19, add = TRUE, col = "red")
```



- When the data have no outliers, two values from end whiskers represents the min and max values, similarly boundaries of the box are Q1 and Q2, and the middle value represents the median.

6.4. Outlier and detecting an outlier using box plot (Simple but widely used method)

Outliers are the data points which are far a way from the vast majority of the data. They may arise due to errors, natural fluctuations, or uncommon occurrences. When extreme outliers are present, they can heavily influence statistical analysis. Handling outliers is challenging and requires careful examination. If they arise due to errors or rare occurrences, removing them may be a good option. However, there are more advanced techniques for dealing with outliers, which are discussed in various research studies.

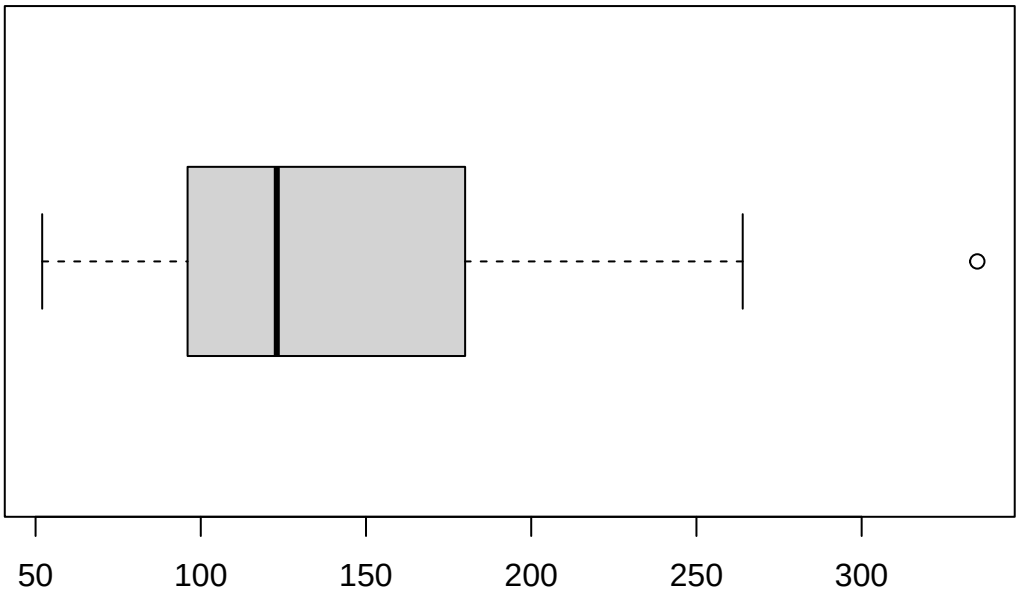
```
IQR = Q3 - Q1
Usual low value, L = Q1 - 1.5*IQR
Usual high value, U = Q3 + 1.5*IQR
Any value outside of the range between L and U considered as outlier
```

```r
# Load dataset
data(mtcars)

# Compute Q1, Q3, and IQR for a variable (e.g., horsepower)
Q1 <- quantile(mtcars$hp, 0.25)
Q3 <- quantile(mtcars$hp, 0.75)
IQR_value <- IQR(mtcars$hp)

# Compute outlier bounds
lower_bound <- Q1 - 1.5 * IQR_value
upper_bound <- Q3 + 1.5 * IQR_value

# Identify outliers
outliers <- mtcars$hp[mtcars$hp < lower_bound | mtcars$hp > upper_bound]
print(outliers)
```

```
## [1] 335
```

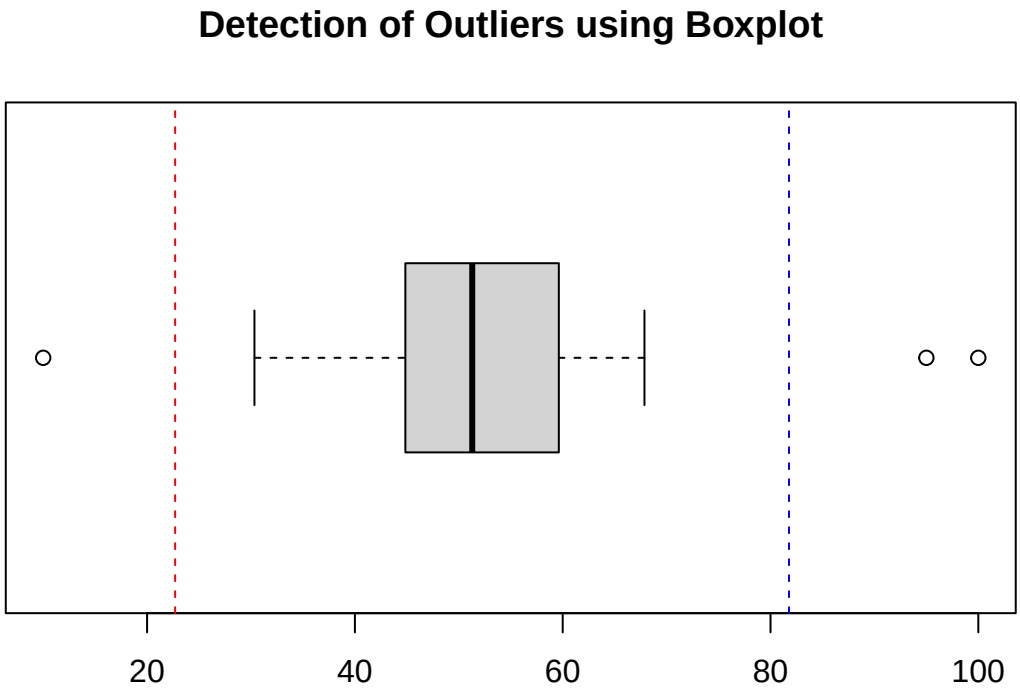
Example of outliers

```
boxplot(mtcars$hp, horizontal = TRUE)
```



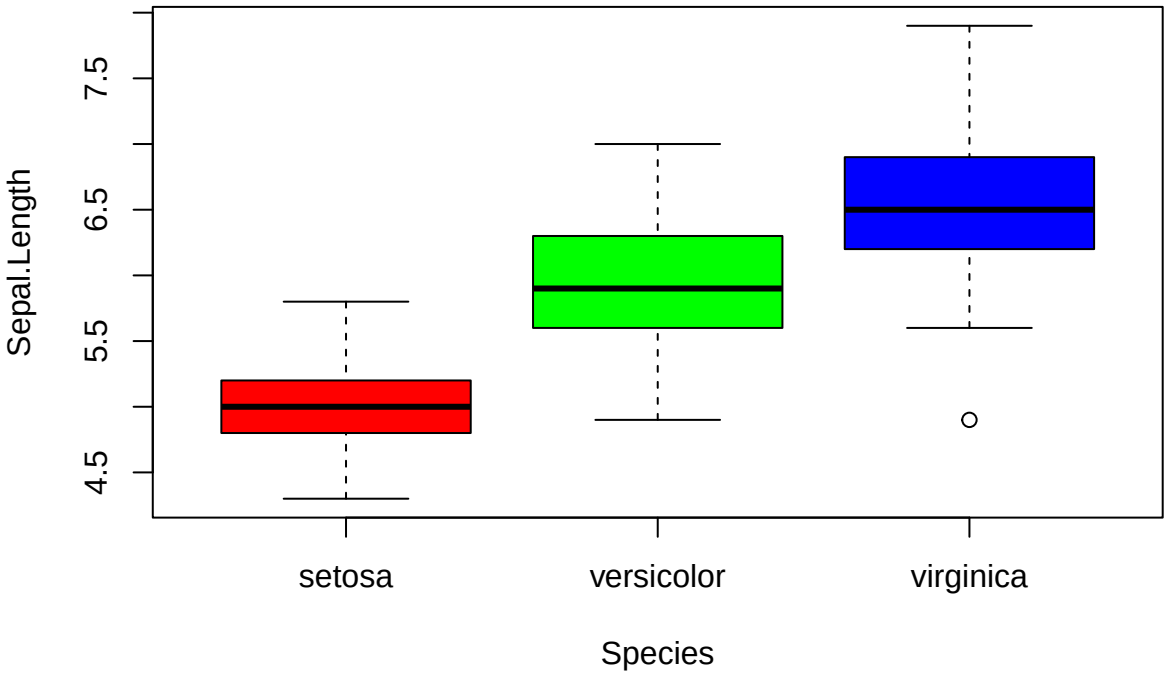
Manual boundry lines for outliers

```
set.seed(123)  # For reproducibility
x <- c(rnorm(20, mean = 50, sd = 10), c(10, 95,100))  # Adding an outlier

# Compute IQR-based bounds
Q1 <- quantile(x, 0.25)
Q3 <- quantile(x, 0.75)
IQR <- Q3 - Q1
L <- Q1 - 1.5 * IQR  # Lower bound
U <- Q3 + 1.5 * IQR  # Upper bound

# Create boxplot
boxplot(x, horizontal = TRUE, main = "Detection of Outliers using Boxplot")

# Add IQR computed boundaries
abline(v = L, col = 'red', lty = 2)  # Dashed red line for lower bound
abline(v = U, col = 'blue', lty = 2)  # Dashed blue line for upper bound
```



6.6. Boxplot of multiple varaibles

```r
boxplot(Sepal.Length ~ Species, data = iris, col = rainbow(3), horizontal = FALSE)
```

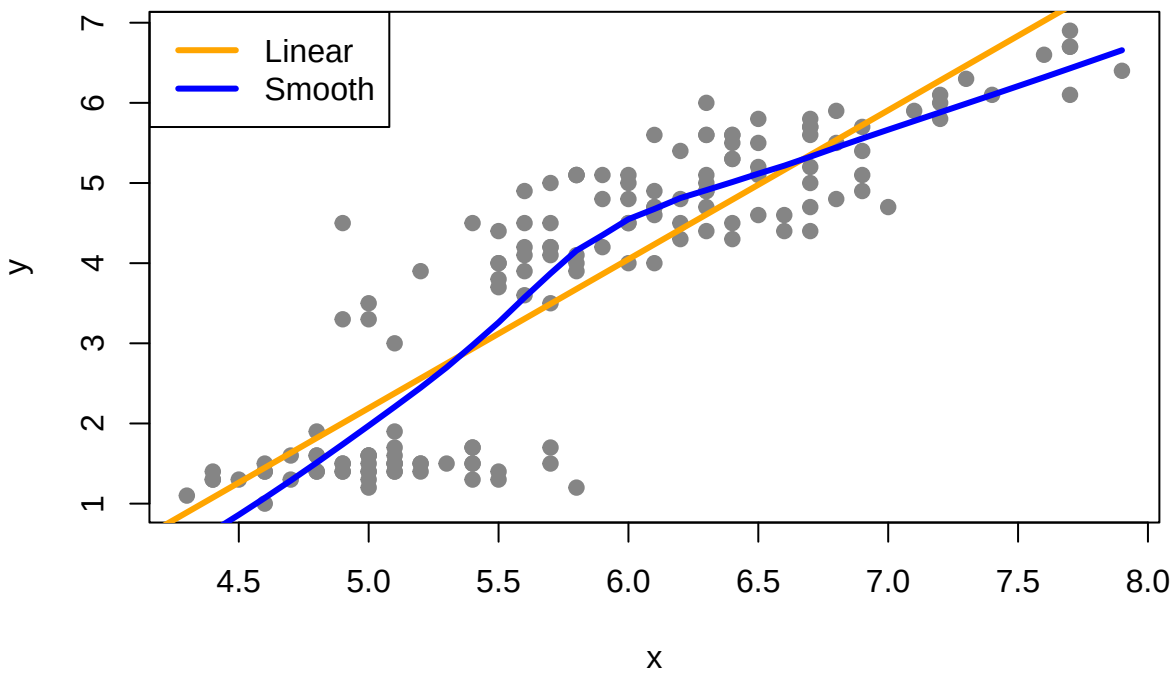


7. Scattor plot

A scatter plot is used to visualize the relationship between two numerical variables. It helps in identifying trends, correlations, and patterns such as linear, non-linear, or no association between variables.

7.1. Scattor plot with regression line

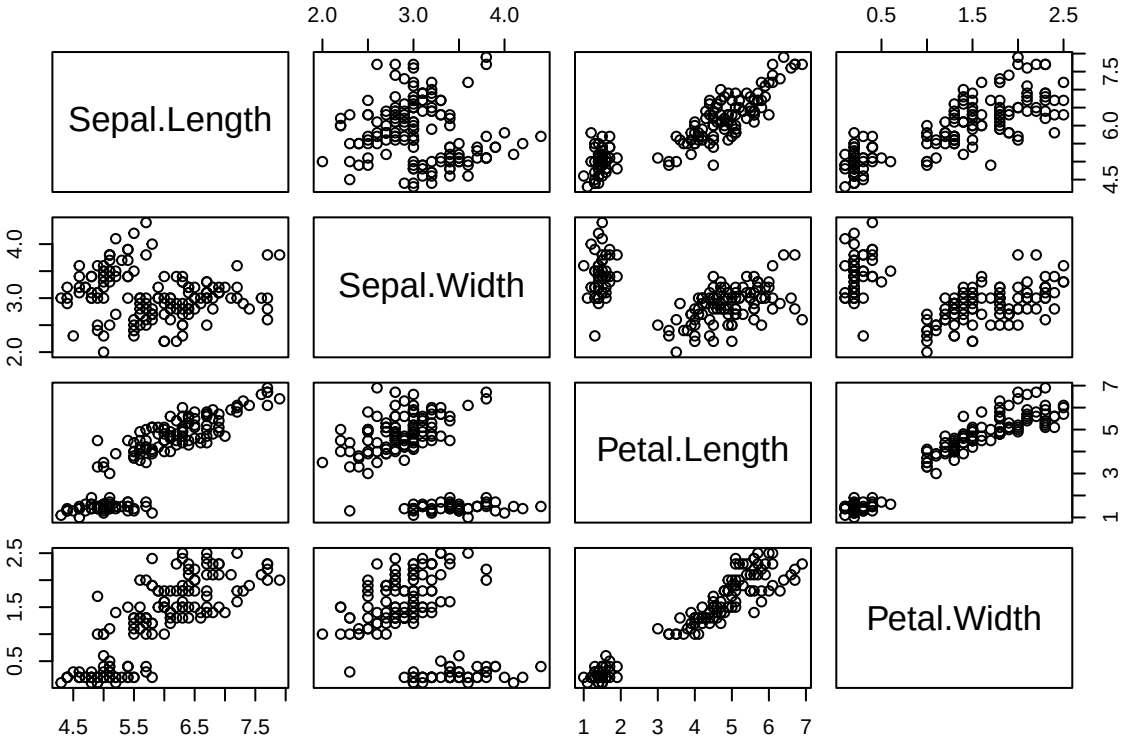
```r
x = iris$Sepal.Length
y = iris$Petal.Length
plot(x, y, pch = 19, col = "gray52")


# Linear fit
abline(lm(y ~ x), col = "orange", lwd = 3)

# Smooth fit
lines(lowess(x, y), col = "blue", lwd = 3)

# Legend
legend("topleft", legend = c("Linear", "Smooth"),
       lwd = 3, lty = c(1, 1), col = c("orange", "blue"))
```



7.2. Scatterplot matrix in R

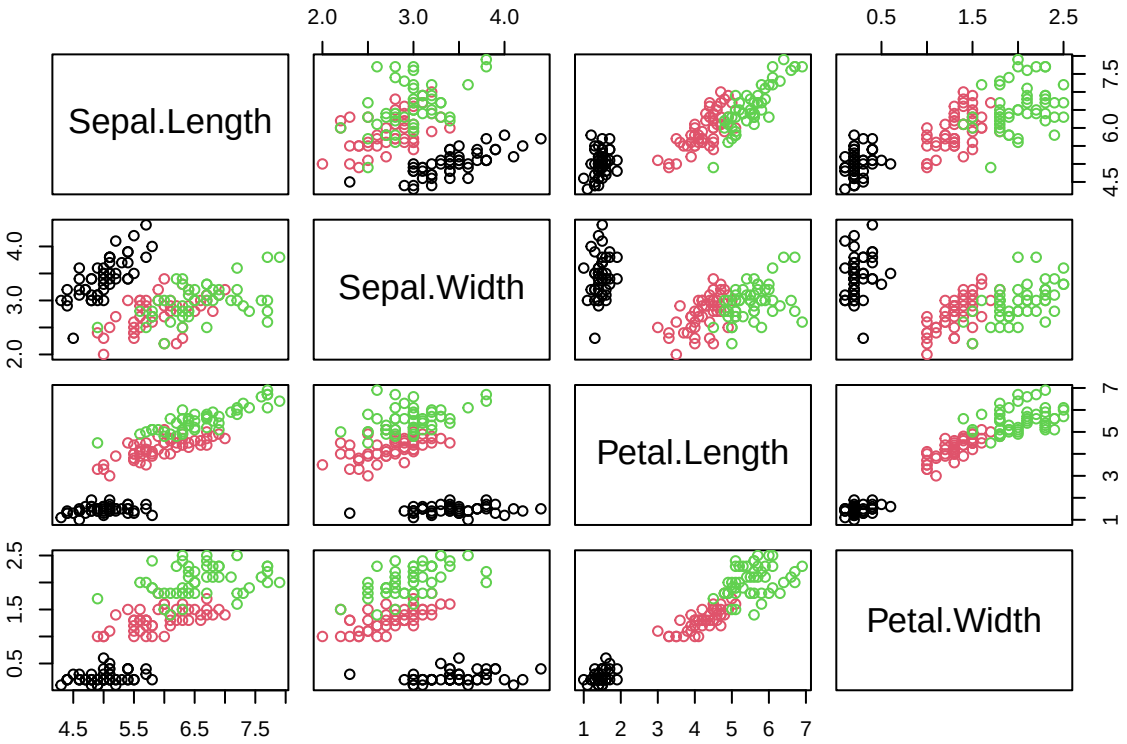
A scatterplot matrix in R is a collection of pairwise scatter plots that display the relationships between multiple variables. It helps visualize the pairwise associations between variables and can be useful in identifying patterns, such as linear or nonlinear relationships, as well as the strength and direction of these associations.

```r
numerical_df <- subset(iris, select = c(Sepal.Length, Sepal.Width,Petal.Length,Petal.Width))
pairs(numerical_df) # Check the association of numerical varaibles.
```



7.3. Pairs plot by category

```r
pairs(~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, col = iris$Species, data = iris)
```


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
2.0
3.0
4.0
0.5
1.5
2.5
4.5
6.0
7.5
1
3
5
7
4.5
5.5
6.5
7.5
1
2
3
4
5
6
7