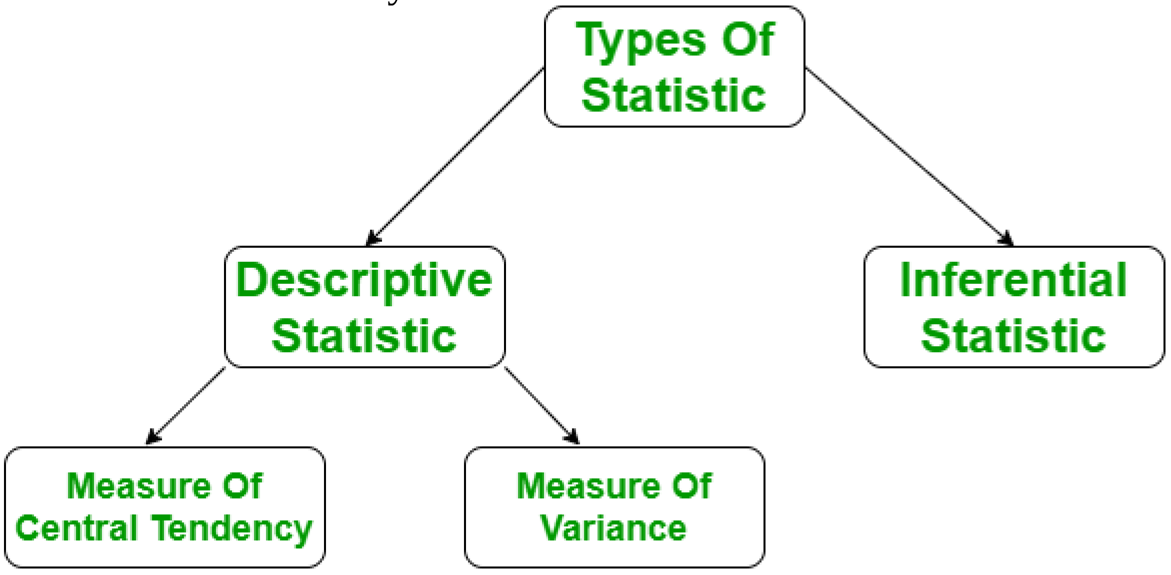
In Descriptive analysis, we are describing our data with the help of various representative methods like using charts, graphs, tables, excel files, etc. In the descriptive analysis, we describe our data in some manner and present it in a meaningful way so that it can be easily understood. Most of the time it is performed on small data sets and this analysis helps us a lot to predict some future trends based on the current findings. Some measures that are used to describe a data set are measures of central tendency and measures of variability or dispersion.

Process of Descriptive Analysis

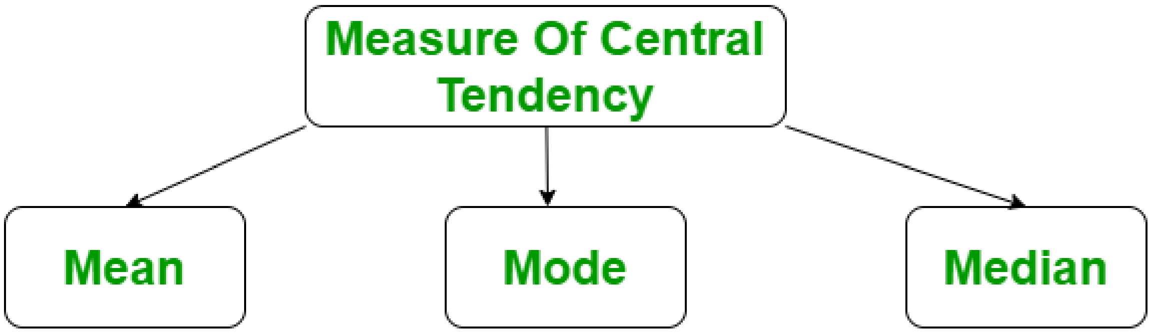
- The measure of central tendency
- Measure of variability



Measure of central tendency

It represents the whole set of data by a single value. It gives us the location of central points. There are three main measures of central tendency:

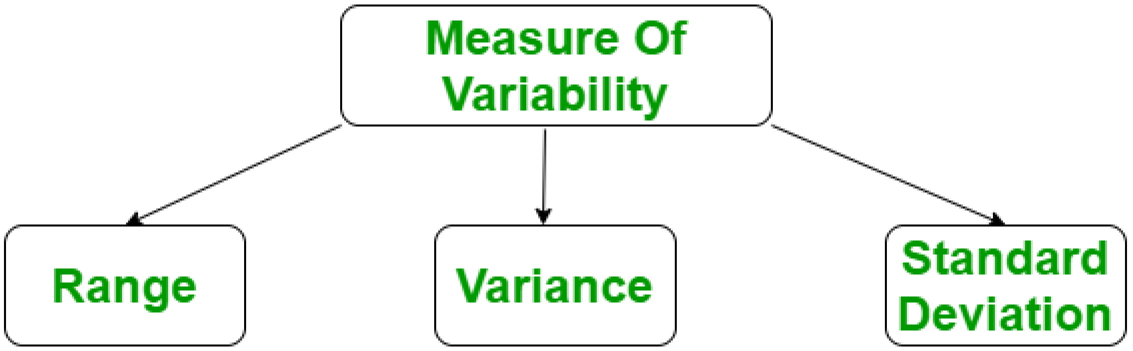
- Mean
- Mode
- Median



Measure of variability

Measure of variability is known as the spread of data or how well is our data is distributed. The most common variability measures are:

- Range
- Variance
- Standard deviation



Need of Descriptive Analysis

Descriptive Analysis helps us to understand our data and is a very important part of Machine Learning. This is due to Machine Learning being all about making predictions. On the other hand, statistics is all about drawing conclusions from data, which is a necessary initial step for Machine Learning. Let's do this descriptive analysis in R.

Descriptive Analysis in R

Descriptive analyses consist of describing simply the data using some summary statistics and graphics. Here, we'll describe how to compute summary statistics using R software.

Import your data into R:

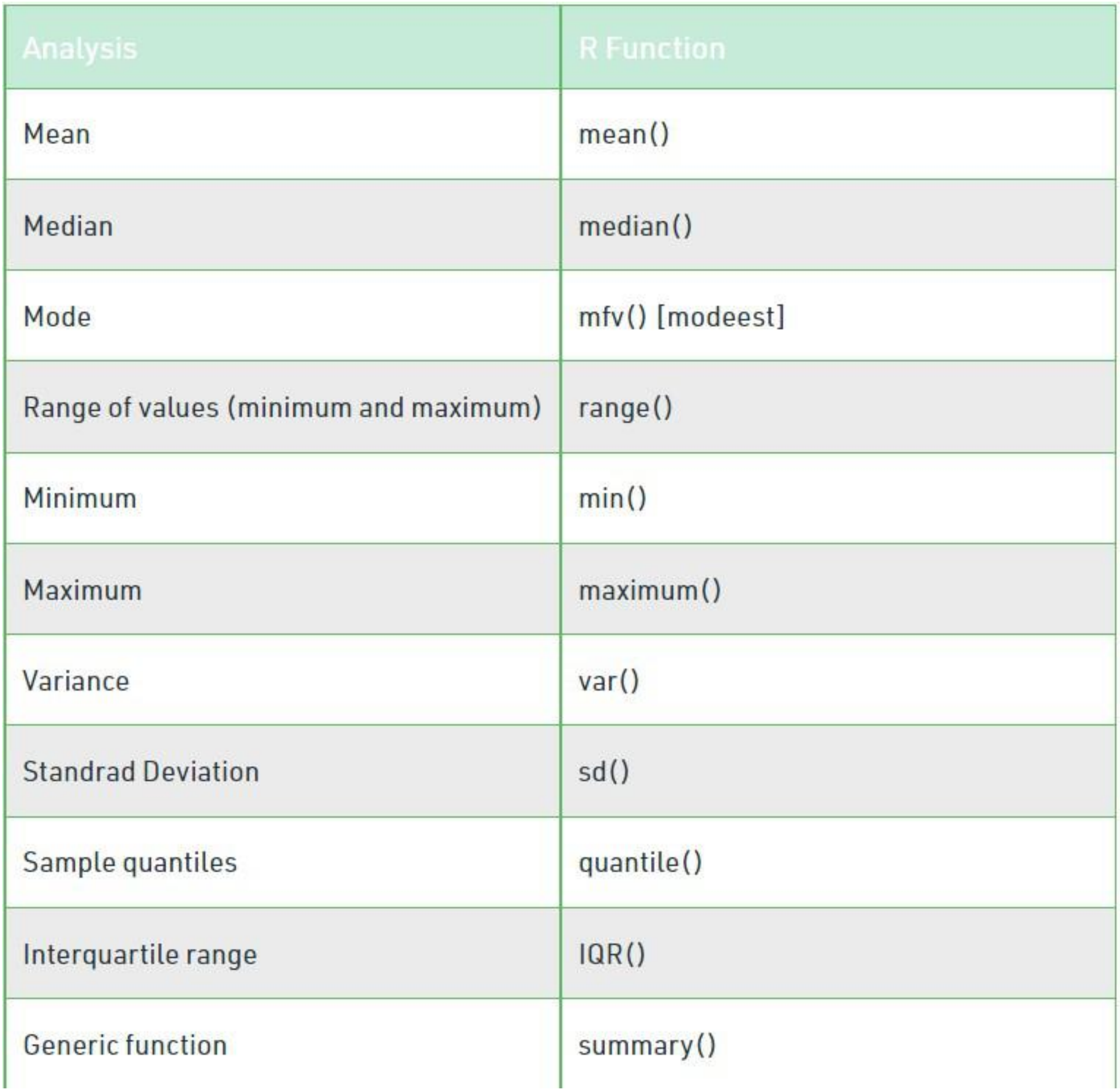
Before doing any computation, first of all, we need to prepare our data, save our data in external .txt or .csv files and it's a best practice to save the file in the current directory. After that import, your data into R as follow:

```
# R program to illustrate

# Descriptive Analysis

# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                stringsAsFactors = F)

# Print the first 6 rows

print(head(myData))
```

Output:

```
   Product Age Gender Education MaritalStatus Usage Fitness Income 
Miles

1   TM195  18   Male        14        Single     3       4  29562 
112

2   TM195  19   Male        15        Single     2       3  31836 
75

3   TM195  19 Female        14     Partnered     4       3  30699 
66

4   TM195  19   Male        12        Single     3       3  32973 
85

5   TM195  20   Male        13     Partnered     4       2  35247 
47

6   TM195  20 Female        14     Partnered     3       3  32973 
66
```

R functions for computing descriptive analysis:



Analysis	R Function
Mean	mean()
Median	median()
Mode	mfv() [modeest]
Range of values (minimum and maximum)	range()
Minimum	min()
Maximum	maximum()
Variance	var()
Standrad Deviation	sd()
Sample quantiles	quantile()
Interquartile range	IQR()
Generic function	summary()

Mean

It is the sum of observations divided by the total number of observations. It is also defined as average which is the sum divided by count.

$$\text{Mean } (\bar{x}) = \frac{\sum x}{n}$$

where n = number of terms

It is calculated by taking the sum of the values and dividing with the number of values in a data series.

The function **mean()** is used to calculate this in R.

Syntax

The basic syntax for calculating mean in R is −

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

Following is the description of the parameters used −

- **x** is the input vector.
- **trim** is used to drop some observations from both end of the sorted vector.
- **na.rm** is used to remove the missing values from the input vector.

Example

```
# Create a vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find Mean.
result.mean <- mean(x)
print(result.mean)
```

When we execute the above code, it produces the following result −

```
[1] 8.22
```

Applying Trim Option

When trim parameter is supplied, the values in the vector get sorted and then the required numbers of observations are dropped from calculating the mean.

When trim = 0.3, 3 values from each end will be dropped from the calculations to find mean.

In this case the sorted vector is (−21, −5, 2, 3, 4.2, 7, 8, 12, 18, 54) and the values removed from the vector for calculating mean are (−21,−5,2) from left and (12,18,54) from right.

```
# Create a vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find Mean.
result.mean <-  mean(x,trim = 0.3)
print(result.mean)
```

When we execute the above code, it produces the following result −

```
[1] 5.55
```

Applying NA Option

If there are missing values, then the mean function returns NA.

To drop the missing values from the calculation use na.rm = TRUE. which means remove the NA values.

```
# Create a vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5,NA)

# Find mean.
result.mean <-  mean(x)
print(result.mean)

# Find mean dropping NA values.
result.mean <-  mean(x,na.rm = TRUE)
print(result.mean)
```

When we execute the above code, it produces the following result −

```
[1] NA
[1] 8.22
```

More Example:

```
# R program to illustrate

# Descriptive Analysis
# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                  stringsAsFactors = F)



# Compute the mean value

mean = mean(myData$Age)

print(mean)
```

Output:

```
[1] 28.78889
```

Median

It is the middle value of the data set. It splits the data into two halves. If the number of elements in the data set is odd then the center element is median and if it is even then the median would be the average of two central elements.

$$\underline{\textbf{Odd}} \qquad \frac{n+1}{2} \qquad\qquad \underline{\textbf{Even}} \qquad \frac{n}{2}, \frac{n}{2}+1$$

where n = number of terms

The middle most value in a data series is called the median. The **median()** function is used in R to calculate this value.

Syntax

The basic syntax for calculating median in R is −

```
median(x, na.rm = FALSE)
```

Following is the description of the parameters used −

- **x** is the input vector.
- **na.rm** is used to remove the missing values from the input vector.

Example

```
# Create the vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find the median.
median.result <- median(x)
print(median.result)
```

When we execute the above code, it produces the following result −

```
[1] 5.6
```

More Example:

```
# R program to illustrate

# Descriptive Analysis

 # Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                 stringsAsFactors = F)

 # Compute the median value

median = median(myData$Age)

print(median)
```

Output:

```
[1] 26
```

Mode

It is the value that has the highest frequency in the given data set. The data set may have no mode if the frequency of all data points is the same. Also, we can have more than one mode if we encounter two or more data points having the same frequency.

```
# Create the vector with numbers.
v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)

# Calculate the mode using the user function.
result <- getmode(v)
print(result)

# Create the vector with characters.
charv <- c("o","it","the","it","it")

# Calculate the mode using the user function.
result <- getmode(charv)
print(result)
```

When we execute the above code, it produces the following result −

```
[1] 2
[1] "it"
```

More Example:

```
# R program to illustrate

# Descriptive Analysis

 # Import the library

library(modeest)

 # Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                   stringsAsFactors = F)
 # Compute the mode value

mode = mfv(myData$Age)

print(mode)
```

Range

The range describes the difference between the largest and smallest data point in our data set. The bigger the range, the more is the spread of data and vice versa.

Range = Largest data value – smallest data value

Example:

```
# R program to illustrate

# Descriptive Analysis



# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                stringsAsFactors = F)

 # Calculate the maximum

max = max(myData$Age)

# Calculate the minimum

min = min(myData$Age)

# Calculate the range

range = max - min
```

```
cat("Range is:\n")

print(range)



# Alternate method to get min and max

r = range(myData$Age)

print(r)
```

Output:

```
Range is:

[1] 32


[1] 18 50
```

Variance

It is defined as an average squared deviation from the mean. It is being calculated by finding the difference between every data point and the average which is also known as the mean, squaring them, adding all of them, and then dividing by the number of data points present in our data set.

$$\sigma^2 = \frac{\sum(\chi - \mu)^2}{N}$$

where,
N = number of terms
u = Mean

Example:

```
# R program to illustrate

# Descriptive Analysis



# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                stringsAsFactors = F)



# Calculating variance

variance = var(myData$Age)

print(variance)
```

Output:

```
[1] 48.21217
```

Standard Deviation

It is defined as the square root of the variance. It is being calculated by finding the Mean, then subtract each number from the Mean which is also known as average and square the result. Adding all the values and then divide by the no of terms followed the square root.

$$\sigma = \sqrt{\frac{\Sigma\,(x - u)^2}{N}}$$

where,
N = number of terms
u = Mean

Example:

- R

```
# R program to illustrate

# Descriptive Analysis



# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv", stringsAsFactors = F)



# Calculating Standard deviation

std = sd(myData$Age)

print(std)
```

Output:

```
[1] 6.943498
```

Some more R function used in Descriptive Analysis:

Quartiles

A quartile is a type of quantile. The first quartile (Q1), is defined as the middle number between the smallest number and the median of the data set, the second quartile (Q2) – the median of the given data set while the third quartile (Q3), is the middle number between the median and the largest value of the data set.

Example:

```
# R program to illustrate

# Descriptive Analysis



# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv", stringsAsFactors = F)



# Calculating Quartiles

quartiles = quantile(myData$Age)

print(quartiles)
```

Output:

```
0%  25%  50%  75% 100%

18   24   26   33   50
```

Interquartile Range

The interquartile range (IQR), also called as midspread or middle 50%, or technically H-spread is the difference between the third quartile (Q3) and the first quartile (Q1). It covers the center of the distribution and contains 50% of the observations.

IQR = Q3 – Q1

Example:

```
# R program to illustrate

# Descriptive Analysis
# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv", stringsAsFactors = F)



# Calculating IQR

IQR = IQR(myData$Age)

print(IQR)
```

Output:

```
[1] 9
```

summary() function in R

The function **summary()** can be used to display several statistic summaries of either one variable or an entire data frame.

Summary of a single variable:

Example:

```
# R program to illustrate

# Descriptive Analysis



# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                stringsAsFactors = F)



# Calculating summary

summary = summary(myData$Age)
```

```
print(summary)
```

Output:

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
18.00   24.00   26.00   28.79   33.00   50.00
```

Summary of the data frame
Example:

```
# R program to illustrate

# Descriptive Analysis



# Import the data using read.csv()

myData = read.csv("CardioGoodFitness.csv",

                stringsAsFactors = F)



# Calculating summary

summary = summary(myData)

print(summary)
```

Output:

```
Product                Age            Gender              Education
 Length:180          Min.   :18.00   Length:180          Min.   :12.00
 Class :character    1st Qu.:24.00   Class :character    1st Qu.:14.00
 Mode  :character    Median :26.00   Mode  :character    Median :16.00
                     Mean   :28.79                       Mean   :15.57
                     3rd Qu.:33.00                       3rd Qu.:16.00
```

```
                    Max.   :50.00                      Max.   :21.00


 MaritalStatus          Usage          Fitness          Income
Miles
 Length:180         Min.   :2.000   Min.   :1.000   Min.   : 29562
Min.   : 21.0
 Class :character   1st Qu.:3.000   1st Qu.:3.000   1st Qu.: 44059
1st Qu.: 66.0
 Mode  :character   Median :3.000   Median :3.000   Median : 50597
Median : 94.0
                    Mean   :3.456   Mean   :3.311   Mean   : 53720
Mean   :103.2
                    3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.: 58668
3rd Qu.:114.8
                    Max.   :7.000   Max.   :5.000   Max.   :104581
Max.   :360.0
```

R - Pie Charts

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

In R the pie chart is created using the **pie()** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

The basic syntax for creating a pie-chart using the R is −

```
pie(x, labels, radius, main, col, clockwise)
```

Following is the description of the parameters used −

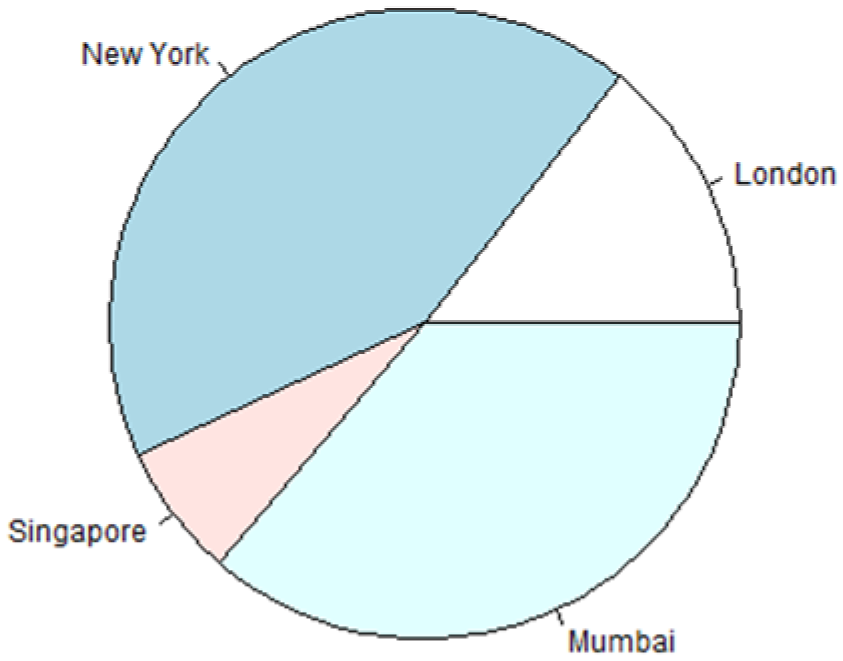
- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.
- **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
- **main** indicates the title of the chart.
- **col** indicates the color palette.
- **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Example

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.
png(file = "city.png")

# Plot the chart.
pie(x,labels)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



Pie Chart Title and Colors

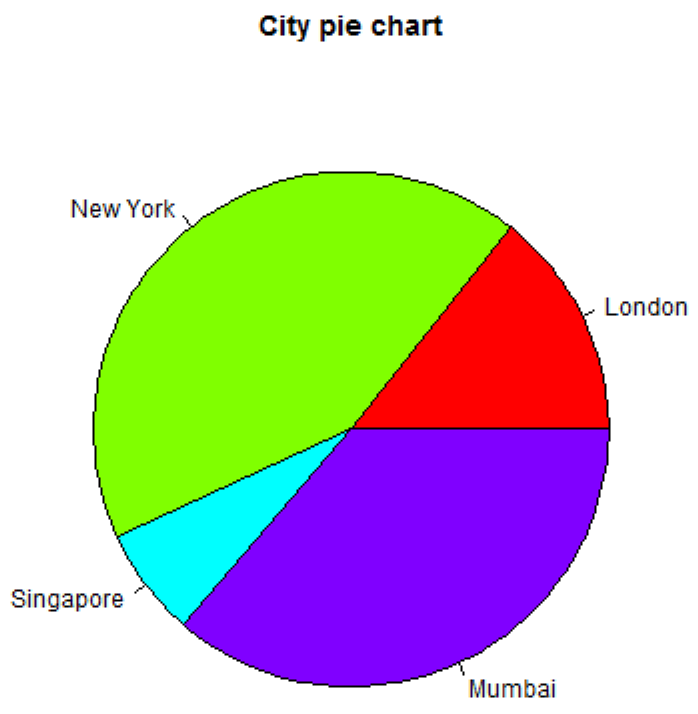
We can expand the features of the chart by adding more parameters to the function. We will use parameter **main** to add a title to the chart and another parameter is **col** which will make use of rainbow colour pallet while drawing the chart. The length of the pallet should be same as the number of values we have for the chart. Hence we use length(x).

Example

The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.
png(file = "city_title_colours.jpg")

# Plot the chart with title and rainbow color pallet.
pie(x, labels, main = "City pie chart", col = rainbow(length(x)))

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −

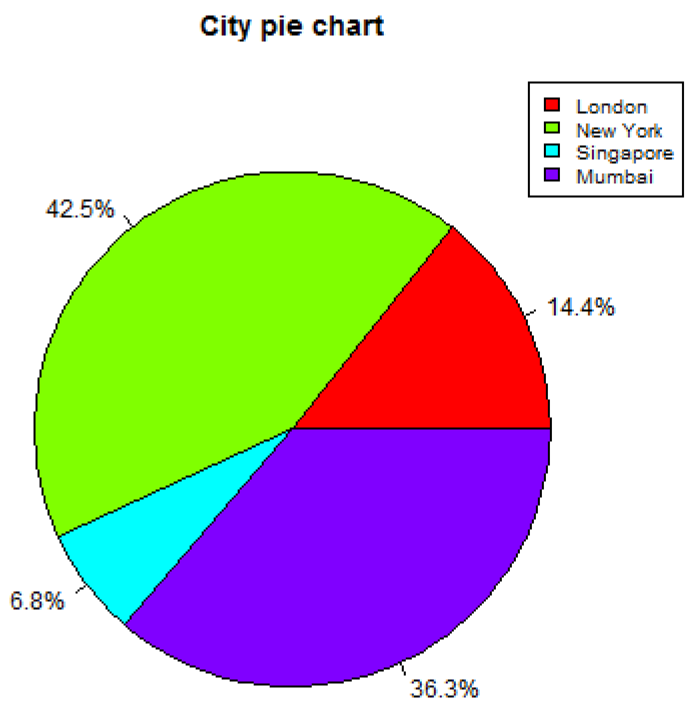


Slice Percentages and Chart Legend

We can add slice percentage and a chart legend by creating additional chart variables.

```
# Create data for the graph.
x <-  c(21, 62, 10,53)
labels <-  c("London","New York","Singapore","Mumbai")

piepercent<- round(100*x/sum(x), 1)

# Give the chart file a name.
png(file = "city_percentage_legends.jpg")

# Plot the chart.
pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))
legend("topright", c("London","New York","Singapore","Mumbai"), cex = 0.8,
   fill = rainbow(length(x)))

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −

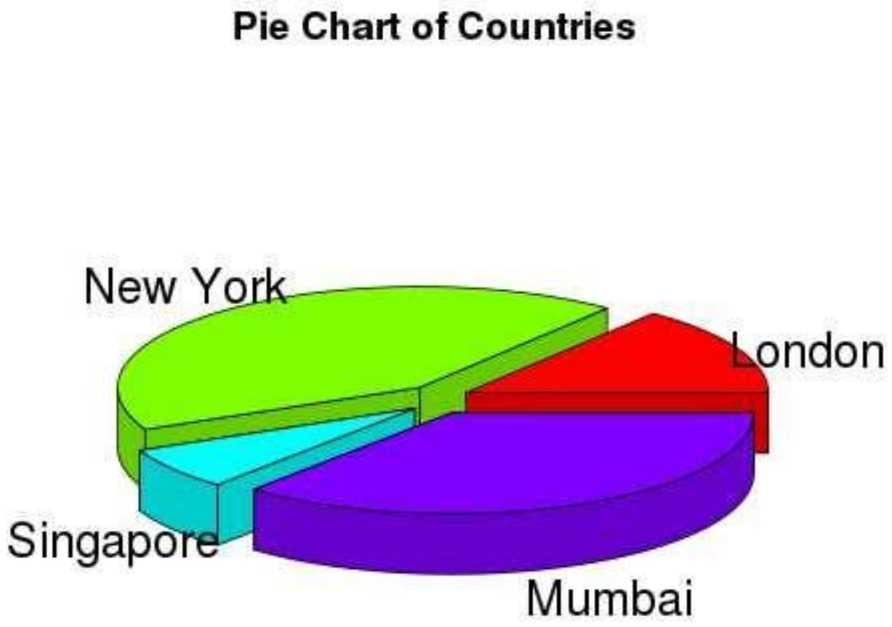


3D Pie Chart

A pie chart with 3 dimensions can be drawn using additional packages. The package **plotrix** has a function called **pie3D()** that is used for this.

```
# Get the library.
library(plotrix)

# Create data for the graph.
x <-  c(21, 62, 10,53)
lbl <-  c("London","New York","Singapore","Mumbai")

# Give the chart file a name.
png(file = "3d_pie_chart.jpg")

# Plot the chart.
pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



R - Bar Charts

Previous Page

Next Page

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function **barplot()** to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax

The basic syntax to create a bar-chart in R is −

```
barplot(H,xlab,ylab,main, names.arg,col)
```

Following is the description of the parameters used −

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.

- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

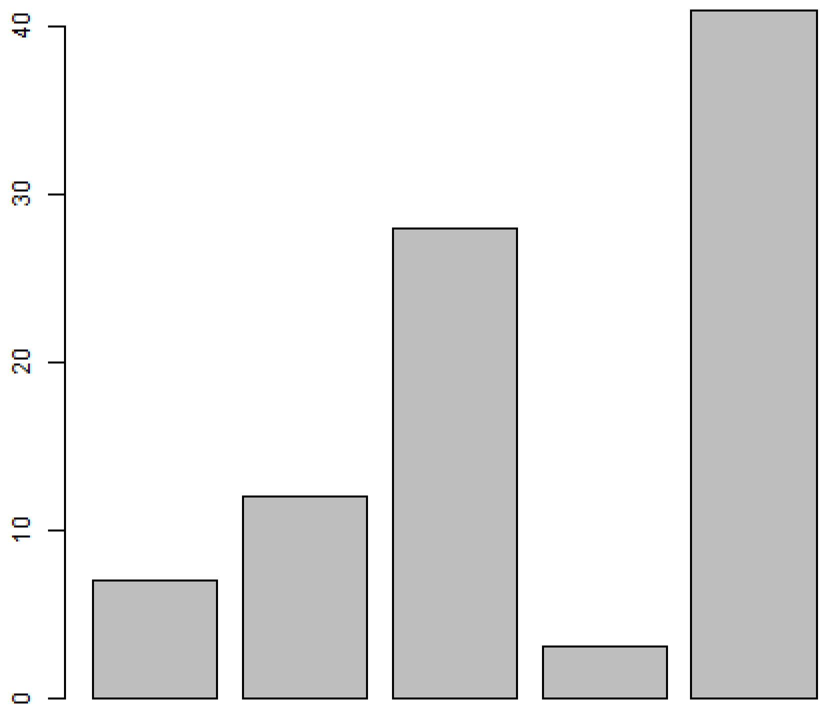
Example

A simple bar chart is created using just the input vector and the name of each bar.

The below script will create and save the bar chart in the current R working directory.

```
# Create the data for the chart
H <- c(7,12,28,3,41)

# Give the chart file a name
png(file = "barchart.png")

# Plot the bar chart
barplot(H)

# Save the file
dev.off()
```

When we execute above code, it produces following result −



Bar Chart Labels, Title and Colors

The features of the bar chart can be expanded by adding more parameters. The **main** parameter is used to add **title**. The **col** parameter is used to add colors to the bars. The **args.name** is a vector having same number of values as the input vector to describe the meaning of each bar.

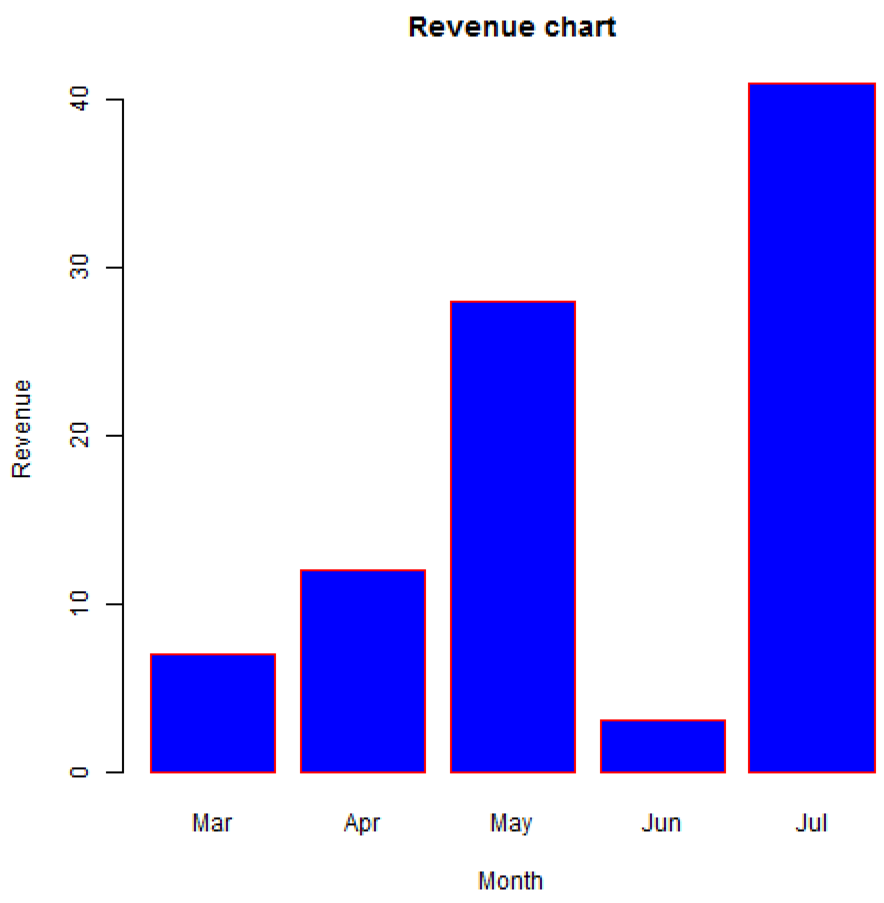
Example

The below script will create and save the bar chart in the current R working directory.

```
# Create the data for the chart
H <- c(7,12,28,3,41)
M <- c("Mar","Apr","May","Jun","Jul")

# Give the chart file a name
png(file = "barchart_months_revenue.png")
```

```
# Plot the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",
main="Revenue chart",border="red")

# Save the file
dev.off()
```

When we execute above code, it produces following result −



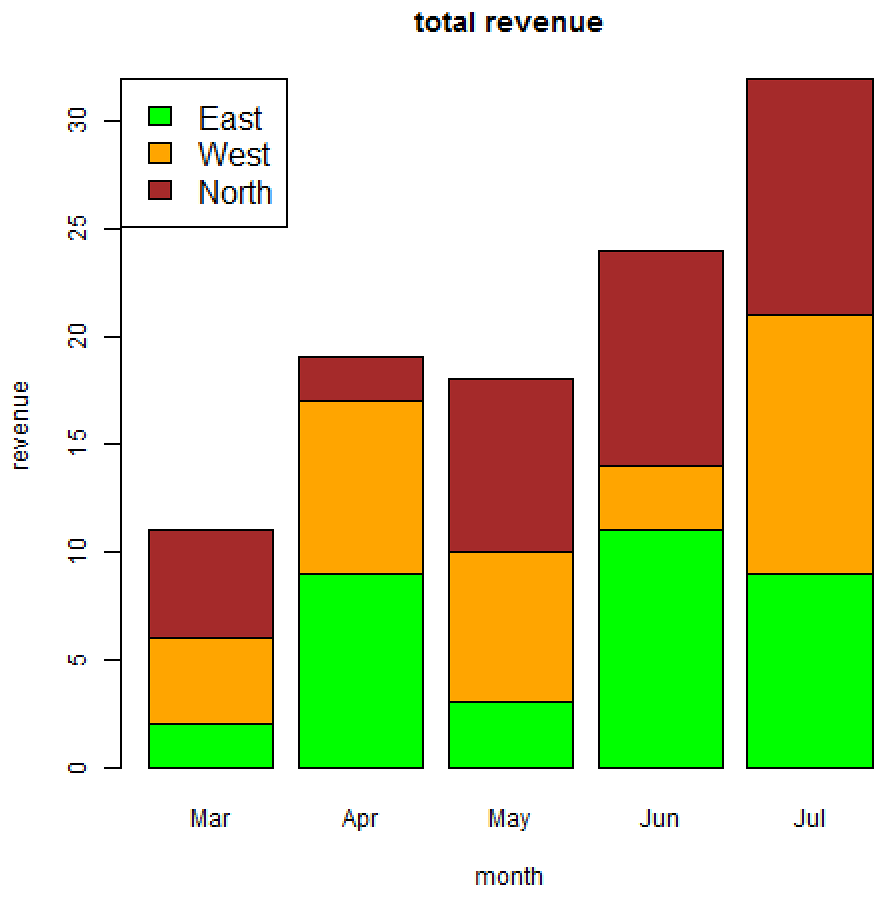
Group Bar Chart and Stacked Bar Chart

We can create bar chart with groups of bars and stacks in each bar by using a matrix as input values.

More than two variables are represented as a matrix which is used to create the group bar chart and stacked bar chart.

```
# Create the input vectors.
colors = c("green","orange","brown")
months <- c("Mar","Apr","May","Jun","Jul")
regions <- c("East","West","North")
```

```
# Create the matrix of the values.
Values <- matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11), nrow = 3, ncol = 5, byrow = TRUE)

# Give the chart file a name
png(file = "barchart_stacked.png")

# Create the bar chart
barplot(Values, main = "total revenue", names.arg = months, xlab = "month", ylab = "revenue", col =
colors)

# Add the legend to the chart
legend("topleft", regions, cex = 1.3, fill = colors)

# Save the file
dev.off()
```



R - Boxplots

Previous Page

Next Page

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the **boxplot()** function.

Syntax

The basic syntax to create a boxplot in R is −

```
boxplot(x, data, notch, varwidth, names, main)
```

Following is the description of the parameters used −

- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.

Example

We use the data set "mtcars" available in the R environment to create a basic boxplot. Let's look at the columns "mpg" and "cyl" in mtcars.

```
input <- mtcars[,c('mpg','cyl')]
print(head(input))
```

When we execute above code, it produces following result −

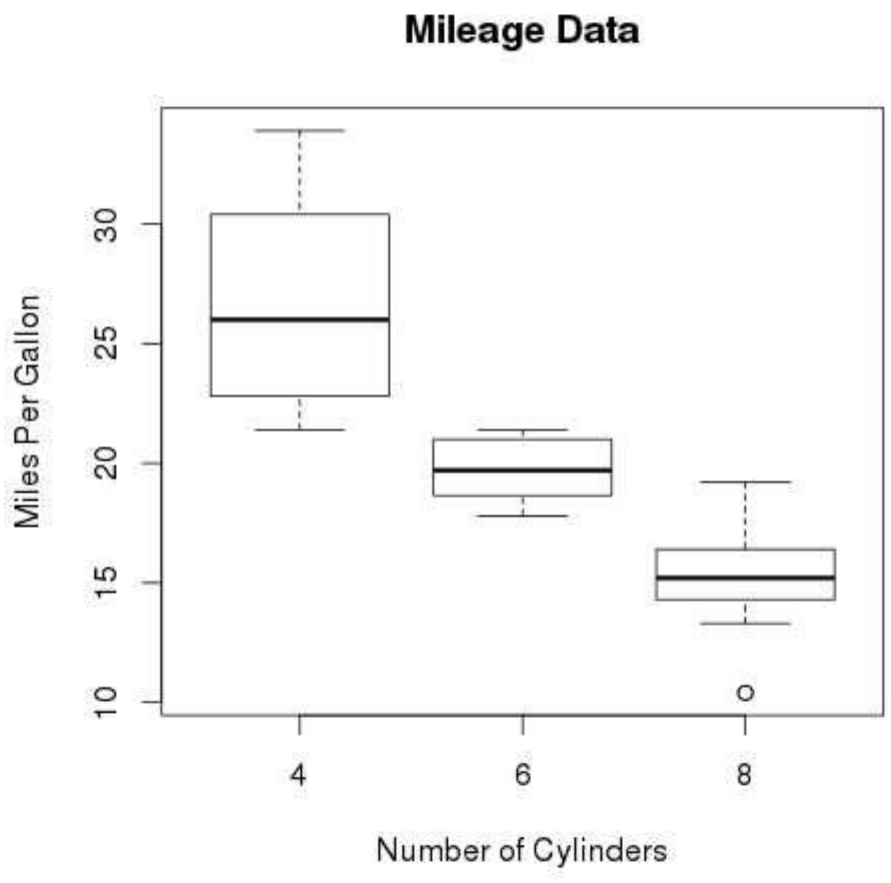
```
                   mpg  cyl
Mazda RX4         21.0   6
Mazda RX4 Wag     21.0   6
Datsun 710        22.8   4
Hornet 4 Drive    21.4   6
Hornet Sportabout 18.7   8
Valiant           18.1   6
```

Creating the Boxplot

The below script will create a boxplot graph for the relation between mpg (miles per gallon) and cyl (number of cylinders).

```
# Give the chart file a name.
png(file = "boxplot.png")

# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
   ylab = "Miles Per Gallon", main = "Mileage Data")

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



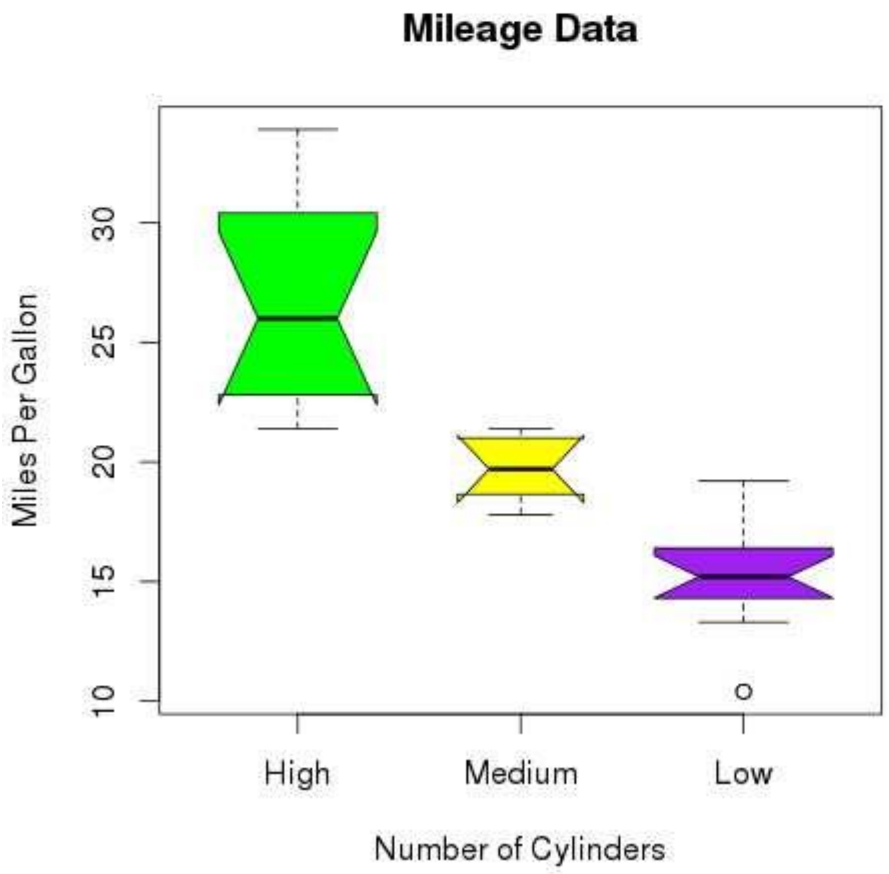
Boxplot with Notch

We can draw boxplot with notch to find out how the medians of different data groups match with each other.

The below script will create a boxplot graph with notch for each of the data group.

```
# Give the chart file a name.
png(file = "boxplot_with_notch.png")

# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars,
   xlab = "Number of Cylinders",
   ylab = "Miles Per Gallon",
   main = "Mileage Data",
   notch = TRUE,
   varwidth = TRUE,
   col = c("green","yellow","purple"),
   names = c("High","Medium","Low")
)
# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



R - Histograms

Previous Page

Next Page

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms.

Syntax

The basic syntax for creating a histogram using R is −

```
hist(v,main,xlab,xlim,ylim,breaks,col,border)
```

Following is the description of the parameters used −

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

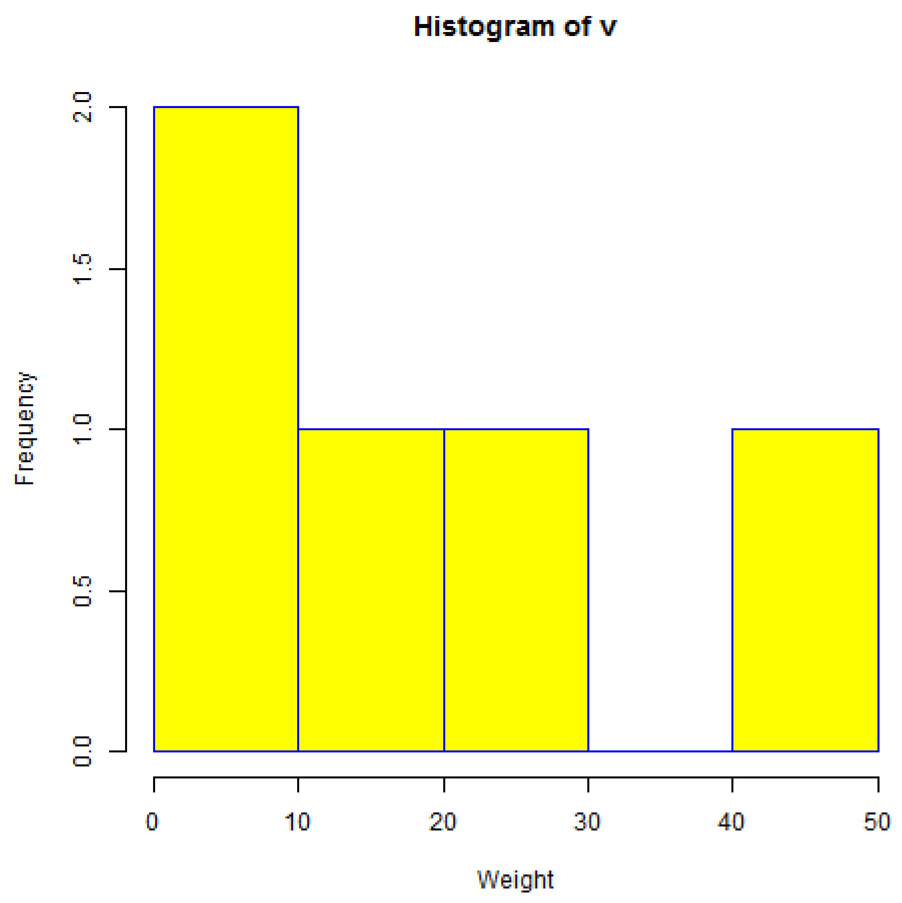
Example

A simple histogram is created using input vector, label, col and border parameters.

The script given below will create and save the histogram in the current R working directory.

```
# Create data for the graph.
v <-  c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.
png(file = "histogram.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "yellow",border = "blue")

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



Range of X and Y values

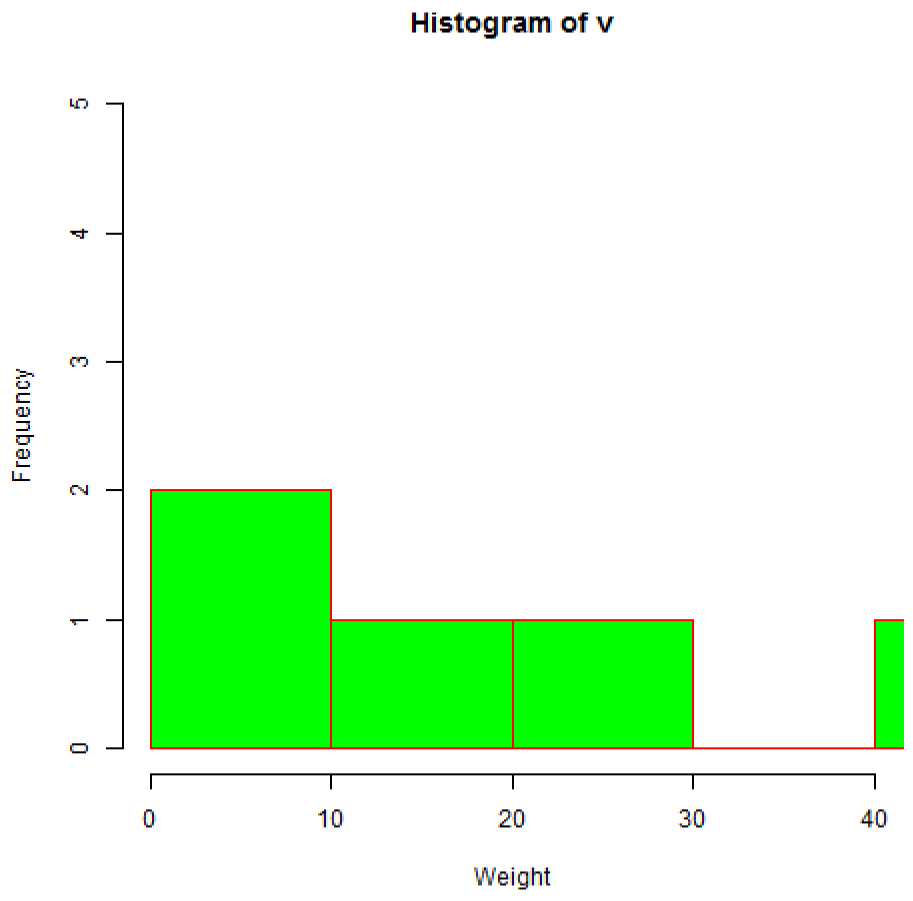
To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.

The width of each of the bar can be decided by using breaks.

```
# Create data for the graph.
v <- c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.
png(file = "histogram_lim_breaks.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim = c(0,5),
  breaks = 5)

# Save the file.
```

```
dev.off()
```

When we execute the above code, it produces the following result −



R - Line Graphs

Previous Page

Next Page

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate (usually the x-coordinate) value. Line charts are usually used in identifying the trends in data.

The **plot()** function in R is used to create the line graph.

Syntax

The basic syntax to create a line chart in R is −

```
plot(v,type,col,xlab,ylab)
```

Following is the description of the parameters used −

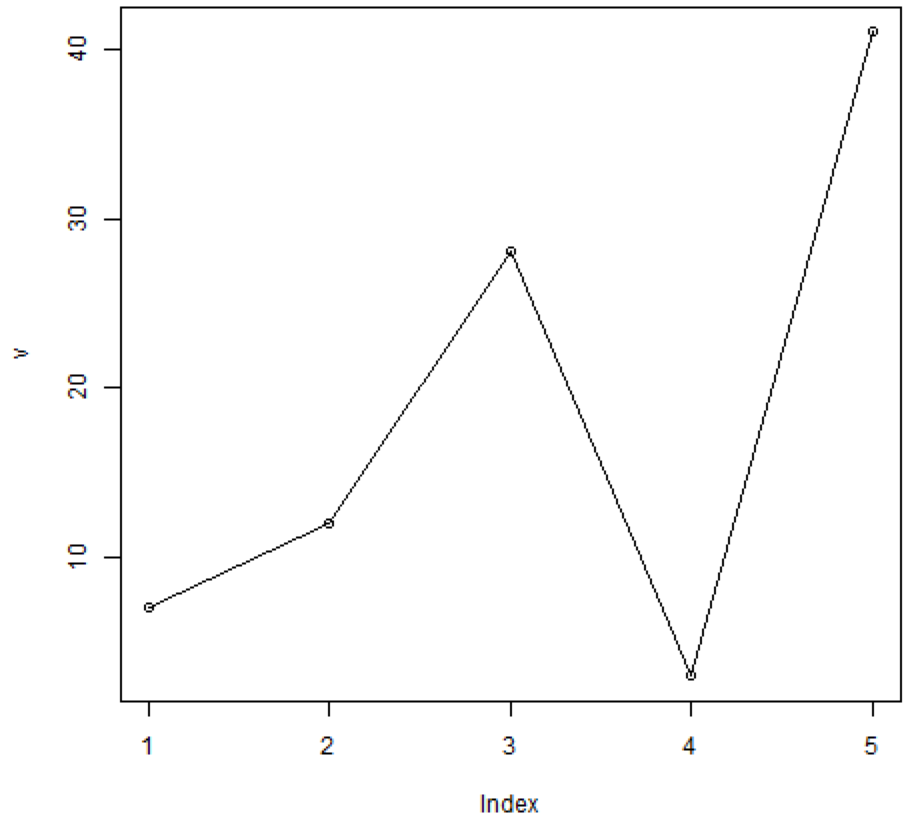
- **v** is a vector containing the numeric values.
- **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the Title of the chart.
- **col** is used to give colors to both the points and lines.

Example

A simple line chart is created using the input vector and the type parameter as "O". The below script will create and save a line chart in the current R working directory.

```
# Create the data for the chart.
v <- c(7,12,28,3,41)

# Give the chart file a name.
png(file = "line_chart.jpg")

# Plot the bar chart.
plot(v,type = "o")

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



Line Chart Title, Color and Labels

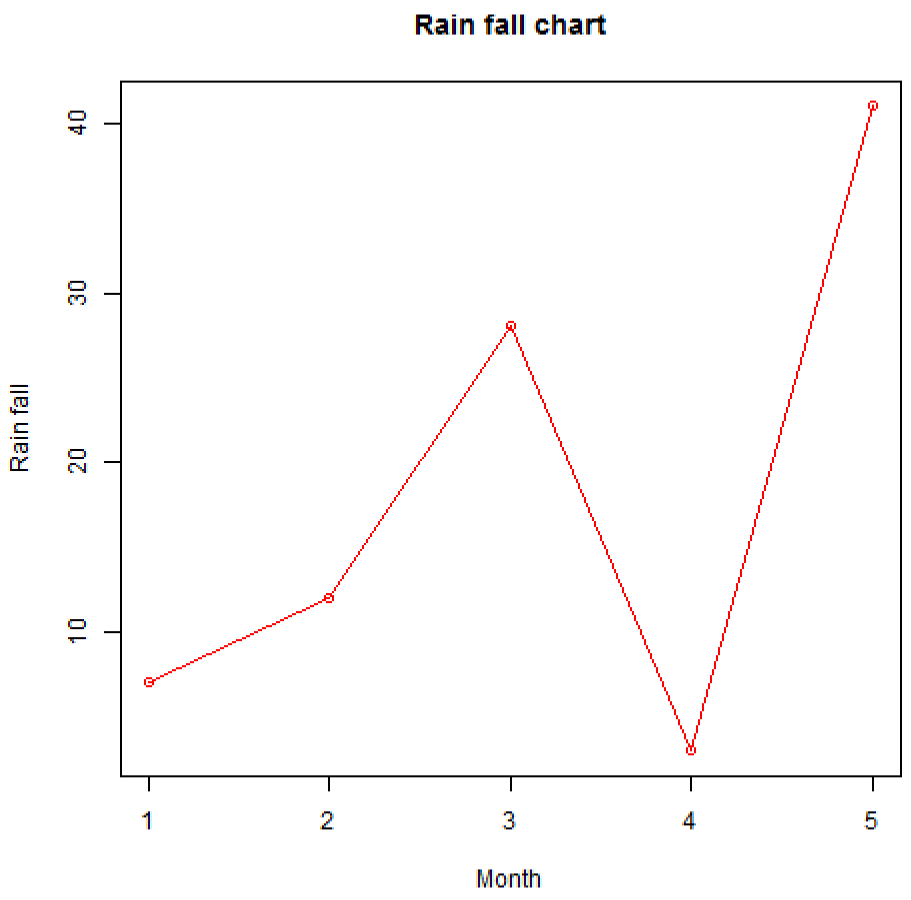
The features of the line chart can be expanded by using additional parameters. We add color to the points and lines, give a title to the chart and add labels to the axes.

Example

```
# Create the data for the chart.
v <- c(7,12,28,3,41)

# Give the chart file a name.
png(file = "line_chart_label_colored.jpg")

# Plot the bar chart.
plot(v,type = "o", col = "red", xlab = "Month", ylab = "Rain fall",
   main = "Rain fall chart")

# Save the file.
```

```
dev.off()
```

When we execute the above code, it produces the following result −



Multiple Lines in a Line Chart

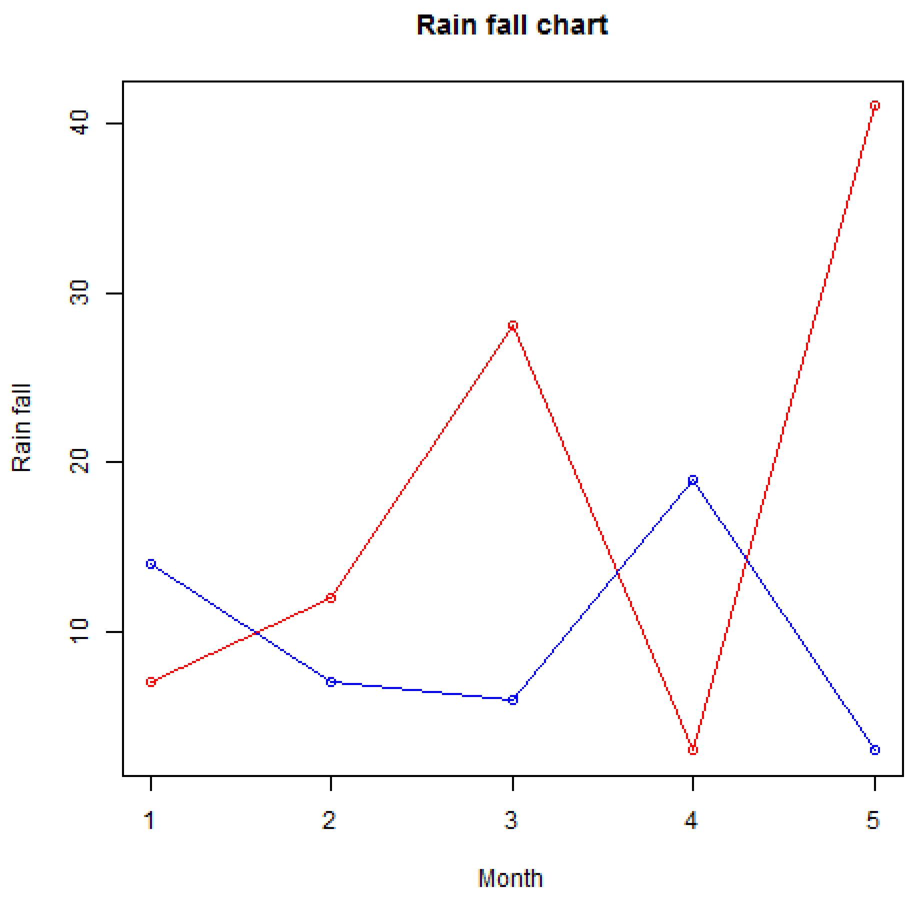
More than one line can be drawn on the same chart by using the **lines()**function.

After the first line is plotted, the lines() function can use an additional vector as input to draw the second line in the chart,

```
# Create the data for the chart.
v <- c(7,12,28,3,41)
t <- c(14,7,6,19,3)

# Give the chart file a name.
png(file = "line_chart_2_lines.jpg")
```

```
# Plot the bar chart.
plot(v,type = "o",col = "red", xlab = "Month", ylab = "Rain fall",
   main = "Rain fall chart")

lines(t, type = "o", col = "blue")

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



R - Scatterplots

Previous Page

Next Page

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

Syntax

The basic syntax for creating scatterplot in R is −

```
plot(x, y, main, xlab, ylab, xlim, ylim, axes)
```

Following is the description of the parameters used −

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

Example

We use the data set **"mtcars"** available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
```

When we execute the above code, it produces the following result −

```
                    wt      mpg
Mazda RX4          2.620   21.0
Mazda RX4 Wag      2.875   21.0
Datsun 710         2.320   22.8
Hornet 4 Drive     3.215   21.4
Hornet Sportabout  3.440   18.7
Valiant            3.460   18.1
```

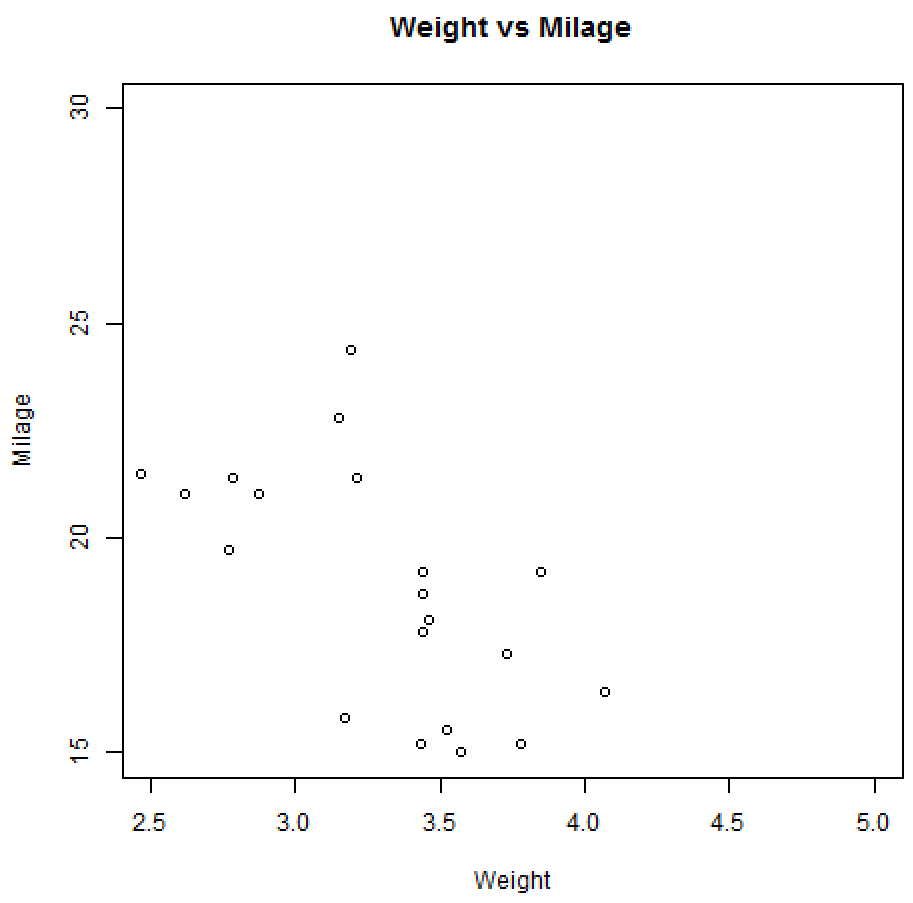
Creating the Scatterplot

The below script will create a scatterplot graph for the relation between wt(weight) and mpg(miles per gallon).

```
# Get the input values.
```

```
input <- mtcars[,c('wt','mpg')]

# Give the chart file a name.
png(file = "scatterplot.png")

# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
   xlab = "Weight",
   ylab = "Milage",
   xlim = c(2.5,5),
   ylim = c(15,30),
   main = "Weight vs Milage"
)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



Scatterplot Matrices

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots.

Syntax

The basic syntax for creating scatterplot matrices in R is −

```
pairs(formula, data)
```

Following is the description of the parameters used −

- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

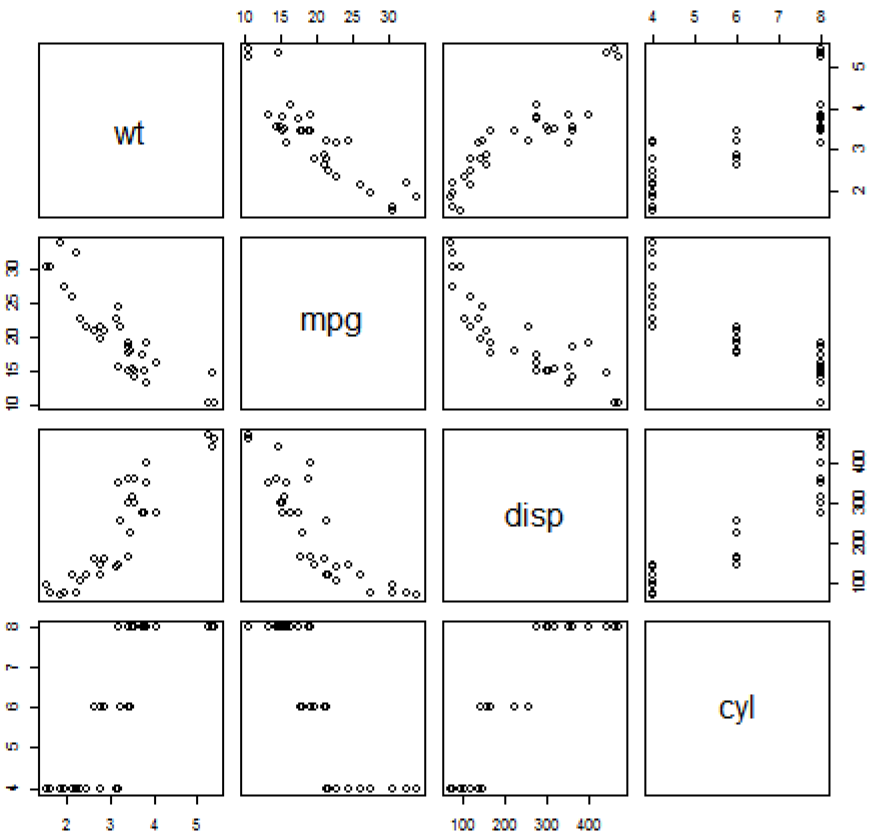
Example

Each variable is paired up with each of the remaining variable. A scatterplot is plotted for each pair.

```
# Give the chart file a name.
png(file = "scatterplot_matrices.png")

# Plot the matrices between 4 variables giving 12 plots.

# One variable with 3 others and total 4 variables.

pairs(~wt+mpg+disp+cyl,data = mtcars,
   main = "Scatterplot Matrix")

# Save the file.
dev.off()
```

When the above code is executed we get the following output.

Scatterplot Matrix


wt
mpg
disp
cyl