INTRODUCTION TO R-PROGRAMMING

Topics:

- What is R
- Basic Commands
- Charts and graphs in R (Histogram)
- Standard deviation
- Covariance and Correlation coefficient
- Linear models in R: Simple linear Regression, Multiple Regression.

R - INTRODUCTION:

R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by **Ross Ihaka and Robert Gentleman** at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.

Features of R

As stated earlier, R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R −

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

R Command Prompt

Once you have R environment setup, then it's easy to start your R command prompt by just clicking on R, This will launch R interpreter and you will get a prompt > where you can start typing your program as follows −

```
# My first program in R Programming

> myString <- "Hello, World!"

> print ( myString)

[1] "Hello, World!"
```

Here first statement defines a string variable myString, where we assign a string "Hello, World!" and then next statement print() is being used to print the value stored in variable myString.

Basic R commands

Command	Purpose
help()	Obtain documentation for a given R command
example()	View some examples on the use of a command
c(), scan()	Enter data manually to a vector in R
seq()	Make arithmetic progression vector
rep()	Make vector of repeated values
data()	Load (often into a data.frame) built-in dataset
View()	View dataset in a spreadsheet-type format
dim()	See dimensions (# of rows/cols) of data.frame
length()	Give length of a vector
rm()	Removes an item from memory

1

Basic R commands

Command	Purpose
hist()	Command for producing a histogram
histogram()	Lattice command for producing a histogram
mean(), median()	Identify “center” of distribution
var(), sd()	Find variance, sd of values in vector
sum()	Add up all values in a vector
quantile()	Find the position of a quantile in a dataset
barplot()	Produces a bar graph
barchart()	Lattice command for producing bar graphs
plot()	Produces a scatterplot
xyplot()	Lattice command for producing a scatterplot

2

Basic R commands

Command	Purpose
lm()	Determine the least-squares regression line
anova()	Analysis of variance (can use on results of lm())
predict()	Obtain predicted values from linear model
nls()	estimate parameters of a nonlinear model
sample()	take a sample from a vector of data
replicate()	repeat some process a set number of times
dbinom(), etc.	tools for Binomial distributions
dpois(), etc.	tools for Poisson distributions
pnorm(), etc.	tools for normal distributions
qt(), etc.	tools for student t distributions

3

Basic R commands

Command	Purpose
pchisq(), etc.	tools for chi-square distributions
binom.test()	hypothesis test and confidence interval for 1 proportion
chisq.test()	carries out a chi-square test

14

Comments

Comments are like helping text in your R program and they are ignored by the interpreter while executing your actual program. Single comment is written using # in the beginning of the statement as follows −

My first program in R Programming

R does not support multi-line comments but you can perform a trick which is something as follows −

```
if(FALSE) {
   "This is a demo for multi-line comments and it should be put inside either a
      single OR double quote"
}

myString <- "Hello, World!"
print ( myString)
```

[1] "Hello, World!"

Though above comments will be executed by R interpreter, they will not interfere with your actual program. You should put such comments inside, either single or double quote.

Data Types

Generally, while doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are −

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

Data Type	Example	Verify

Logical	TRUE, FALSE	`v <- TRUE`
		`print(class(v))`
		it produces the following result −
		[1] "logical"

Numeric	12.3, 5, 999	v <- 23.5 print(class(v)) it produces the following result − [1] "numeric"
Integer	2L, 34L, 0L	v <- 2L print(class(v)) it produces the following result − [1] "integer"
Complex	3 + 2i	v <- 2+5i print(class(v)) it produces the following result − [1] "complex"
Character	'a' , '"good", "TRUE", '23.4'	v <- "TRUE" print(class(v)) it produces the following result − [1] "character"
Raw	"Hello" is stored as 48 65 6c 6c 6f	v <- charToRaw("Hello") print(class(v)) it produces the following result − [1] "raw"

Vectors

When you want to create vector with more than one element, you should use **c()** function which means to combine the elements into a vector.

```
# Create a vector.
apple <- c('red','green',"yellow")
print(apple)

# Get the class of the vector.
print(class(apple))
```

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

[1] "red" "green" "yellow"

[1] "character"

R – Data Types

- The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are:
- Vectors , Lists, Matrices, Arrays, Factors, Data Frames
- The simplest of these objects is the vector object and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

```
# Create the data frame.
BMI <-   data.frame(
   gender = c("Male", "Male","Female"),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)
print(BMI)
```

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

```
  gender height weight Age
1   Male  152.0     81  42
2   Male  171.5     93  38
3 Female  165.0     78  26
```

R-Charts and Graphs

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.

Bar chart

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.

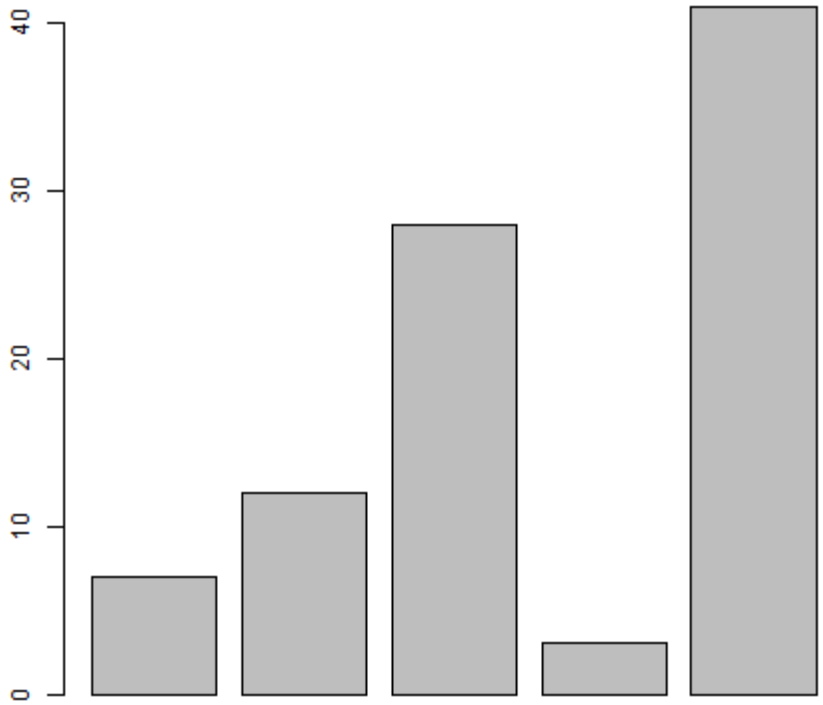
Example

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.

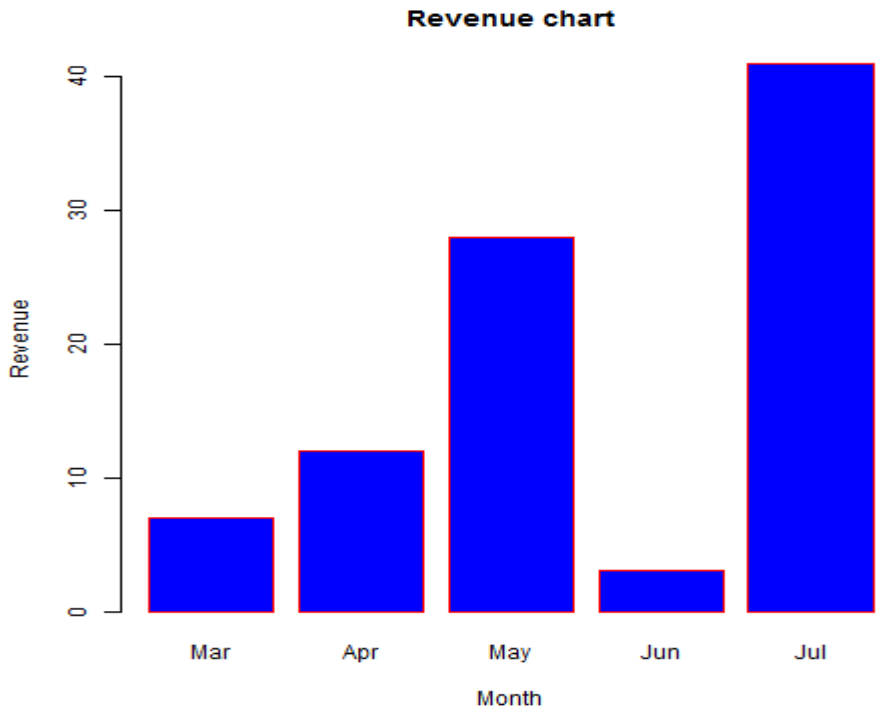
Example

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 –



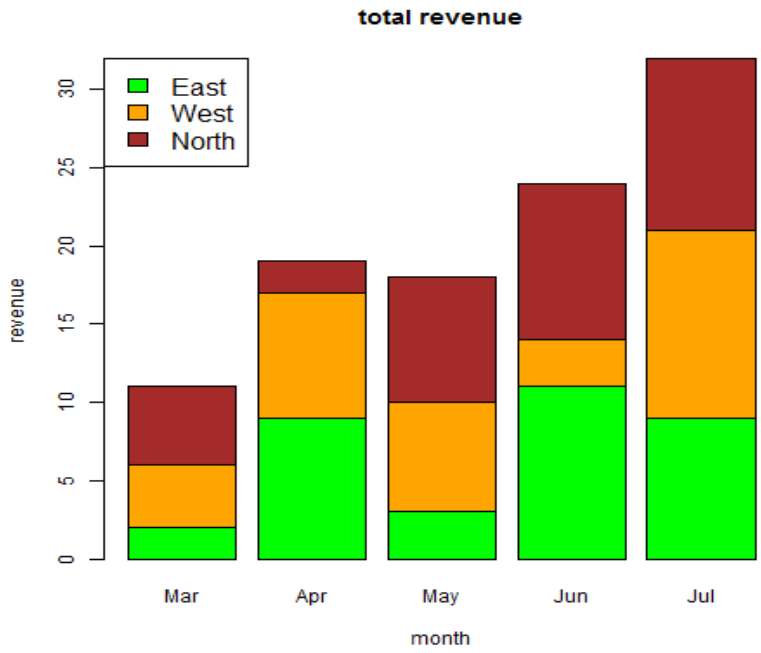
Group Bar Chart and Stacked Bar Chart

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()
```



Histogram

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.

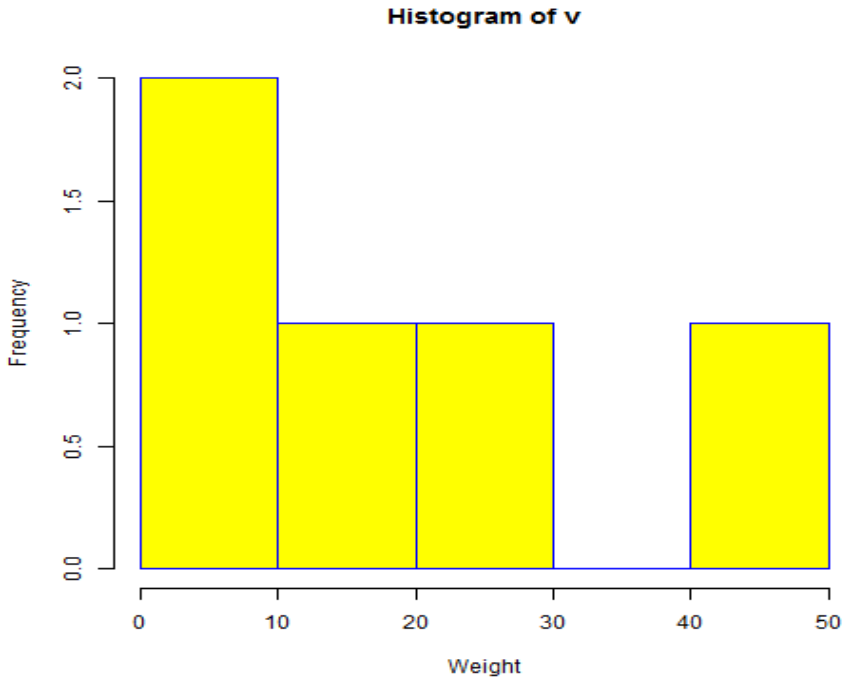
Example

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 −



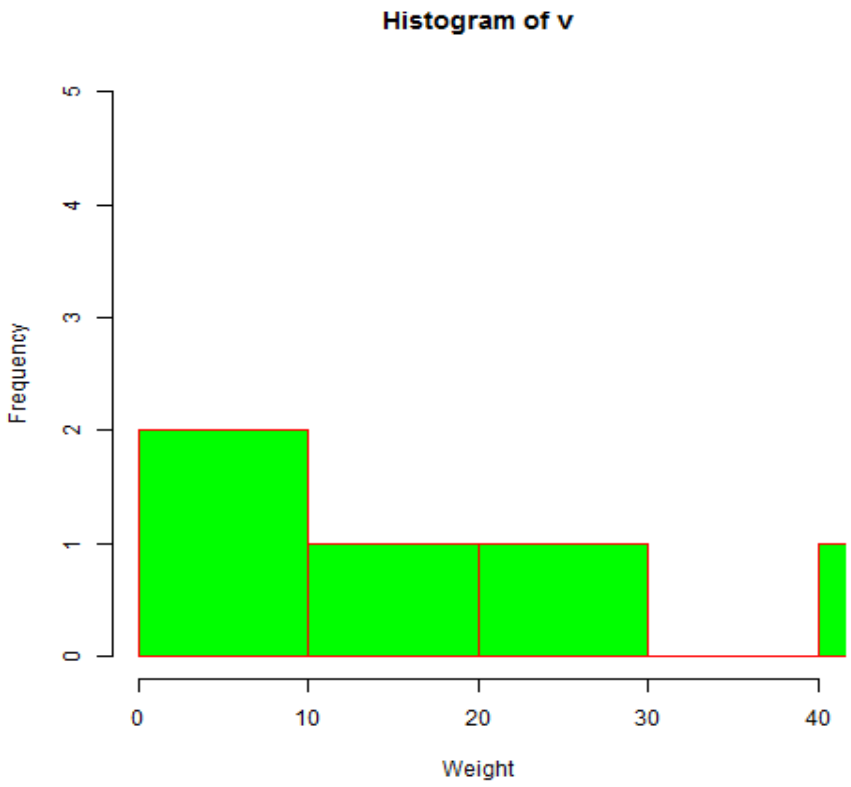
Range of X and Y values

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-Statistics

Statistical analysis in R is performed by using many in-built functions. Most of these functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result.

The functions we are discussing in this chapter are mean, median and mode.

Measures of Central Tendency

1. Mean : Average
2. Median : Middle Value
3. Mode : Most Often

Mean

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
```

Median

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
```

Mode

The mode is the value that has highest number of occurrences in a set of data. Unike mean and median, mode can have both numeric and character data.

R does not have a standard in-built function to calculate mode. So we create a user function to calculate mode of a data set in R. This function takes the vector as input and gives the mode value as output.

Example

```
# Create the function.
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

# 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"
```

Variance: How far a set of data values are spread out from their mean.

```
> variance.result = var(x) # calculate variance
> print (variance.result)
```

Standard Deviation: A measure that is used to quantify the amount of variation or dispersion of a set of data values.

```
> sd.result = sqrt(var(x)) # calculate standard deviation
> print (sd.result)
  [1] 1.576138
```

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

```
# Create the data frame.
BMI <-   data.frame(
   gender = c("Male", "Male","Female"),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)
print(BMI)
```

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

```
  gender height weight Age
1   Male  152.0     81  42
2   Male  171.5     93  38
3   Female  165.0     78  26
```

Correlation and Covariance

You can use the **cor( )** function to produce correlations and the **cov( )** function to produces covariances.

Correlation test is used to evaluate the association between two or more variables.

Correlation coefficient can be computed using the functions **cor()** or **cor.test()**:

- **cor()** computes the **correlation coefficient**
- **cor.test()** test for association/correlation between paired samples. It returns both the **correlation coefficient** and the **significance level**(or p-value) of the correlation .

A simplified format is **cor(x, use=, method=)** where

Option	Description
x	Matrix or data frame
use	Specifies the handling of missing data. Options are **all.obs** (assumes no missing data - missing data will produce an error), **complete.obs** (listwise deletion), and **pairwise.complete.obs** (pairwise deletion)
method	Specifies the type of correlation. Options are **pearson**, **spearman** or **kendall**.

Example

```
# Create the function.
> x <- mtcars[1:3]
> y <- mtcars[4:6]
```

```
> cor(x, y)
>df<-data.frame(x,y)
> cor(df, use=, method="pearson" )
> cor(df[,1:3],method="spearman")
> cov(x, y)
```

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

	hp	drat	wt
mpg	-0.7761684	0.6811719	-0.8676594
cyl	0.8324475	-0.6999381	0.7824958
disp	0.7909486	-0.7102139	0.8879799

	x	y
x	1.0000000	0.9622683
y	0.9622683	1.0000000

	x	y
x	1.0000000	0.9669173
y	0.9669173	1.0000000

[1] 42.07355

Regression analysis

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is $y = ax + b$

Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()** function in R.

Input Data

Below is the sample data representing the observations

```
# Values of height
151, 174, 138, 186, 128, 136, 179, 163, 152, 131

# Values of weight.
63, 81, 56, 91, 47, 57, 76, 72, 62, 48
```

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in linear regression is

```
lm(formula,data)
```

Following is the description of the parameters used

- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)
print(relation)
```

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

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   -38.4551        0.6746
```

Get the Summary of the Relationship

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)
print(summary(relation))
```

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

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-6.3002  -1.6629  0.0412   1.8944  3.9775

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509    8.04901  -4.778  0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,    Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

Predict() Function

Syntax

The basic syntax for predict() in linear regression is −

predict(object, newdata)

Following is the description of the parameters used −

- **object** is the formula which is already created using the lm() function.
- **newdata** is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)
# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <-  predict(relation,a)
print(result)
```

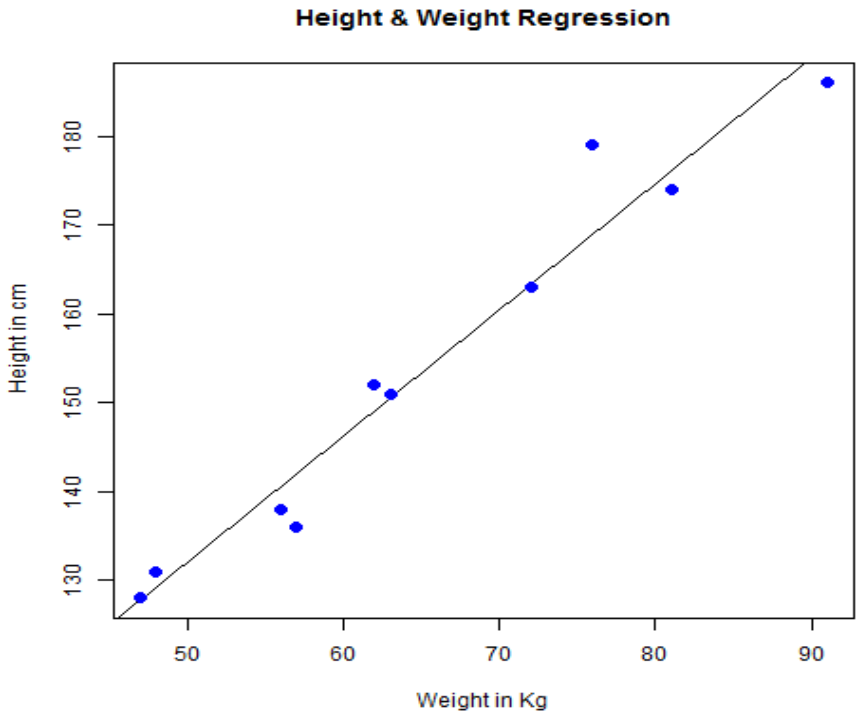
When we execute the above code, it produces the following result −

```
       1
76.22869
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)

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

# Plot the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression",
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

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

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



Multiple regression

Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.

The general mathematical equation for multiple regression is −

$y = a + b1x1 + b2x2 + ...bnxn$

Following is the description of the parameters used −

- **y** is the response variable.
- **a, b1, b2...bn** are the coefficients.
- **x1, x2, ...xn** are the predictor variables.

We create the regression model using the **lm()** function in R. The model determines the value of the coefficients using the input data. Next we can predict the value of the response variable for a given set of predictor variables using these coefficients.

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in multiple regression is −

lm(y ~ x1+x2+x3...,data)

Following is the description of the parameters used −

- **formula** is a symbol presenting the relation between the response variable and predictor variables.
- **data** is the vector on which the formula will be applied.

Example

Input Data

Consider the data set "mtcars" available in the R environment. It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"), horse power("hp"), weight of the car("wt") and some more parameters.

The goal of the model is to establish the relationship between "mpg" as a response variable with "disp","hp" and "wt" as predictor variables. We create a subset of these variables from the mtcars data set for this purpose.

```
input <- mtcars[,c("mpg","disp","hp","wt")]
print(head(input))
```

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

```
                  mpg   disp   hp    wt
Mazda RX4         21.0  160    110   2.620
Mazda RX4 Wag     21.0  160    110   2.875
Datsun 710        22.8  108     93   2.320
Hornet 4 Drive    21.4  258    110   3.215
Hornet Sportabout 18.7  360    175   3.440
Valiant           18.1  225    105   3.460
```

Create Relationship Model & get the Coefficients

```
input <- mtcars[,c("mpg","disp","hp","wt")]

# Create the relationship model.
model <- lm(mpg~disp+hp+wt, data = input)
# Show the model.
print(model)
# Get the Intercept and coefficients as vector elements.
cat("# # # # The Coefficient Values # # # ","\n")
a <- coef(model)[1]
print(a)

Xdisp <- coef(model)[2]
Xhp <- coef(model)[3]
Xwt <- coef(model)[4]

print(Xdisp)
```

```
print(Xhp)
print(Xwt)
```

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

```
Call:
lm(formula = mpg ~ disp + hp + wt, data = input)

Coefficients:
(Intercept)        disp          hp          wt
  37.105505    -0.000937    -0.031157    -3.800891

# # # # The Coefficient Values # # #
(Intercept)
   37.10551
        disp
-0.0009370091
         hp
-0.03115655
       wt
-3.800891
```

Create Equation for Regression Model

Based on the above intercept and coefficient values, we create the mathematical equation.

Y = a+Xdisp.x1+Xhp.x2+Xwt.x3
or
Y = 37.15+(-0.000937)*x1+(-0.0311)*x2+(-3.8008)*x3

Apply Equation for predicting New Values

We can use the regression equation created above to predict the mileage when a new set of values for displacement, horse power and weight is provided.

For a car with disp = 221, hp = 102 and wt = 2.91 the predicted mileage is −

Y = 37.15+(-0.000937)*221+(-0.0311)*102+(-3.8008)*2.91 = 22.7104