CHAPTER 5

STATISTICAL ANALYSIS USING R

❖ OBJECTIVES

On completion of this Chapter you will be able to:

- obtain the basic statistical measures like mean, median, mode, standard deviation, variation etc., using R
- obtain the summary statistics of a given data
- understand and plot the normal distribution of data using R functions
- understand and plot the binomial distribution of data using R functions
- perform correlation analysis on the given data using R
- perform regression analysis on the given data using R
- perform ANOVA, ANCOVA on the given data using R
- perform chi-square and hypothesis testing on the given data using R

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 other important R package for statistical analysis is the *stats* package.

5.1. Basic Statistical Measures

Any dataset available in R or that is been imported into R for further analysis will have both categorical data as well as numeric data. So, we can apply the statistical functions available in R on the numeric data and understand the statistical measures of the fields. The basic statistical measures are the minimum, maximum, mean and median represented by the functions *min()*, *max()*, *mean()* and *median()*

respectively. Let us use the dataset named *mtcars* that is available in R by default to understand these statistical measures.

```
> data(mtcars)
> colnames(mtcars)
[1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear" "carb"
> min(mtcars$cyl)
[1] 4
> max(mtcars$cyl)
[1] 8
> mean(mtcars$cyl)
[1] 6.1875
> median(mtcars$cyl)
[1] 6
```

All the above results can also be obtained by one function *summary()* and this can also be applied on all the fields of the dataset at one shot. The *range()* function gives the minimum and maximum values of a numeric field at one go.

```
> summary(mtcars$cyl)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   4.000   4.000   6.000   6.188   8.000   8.000
> range(mtcars$cyl)
[1] 4 8
```

5.1.1. Mean

Mean 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. The basic syntax for calculating mean in R is given below along with its parameters.

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

trim - to drop some observations from both end of the sorted vector

na.rm - to remove the missing values from the input vector

```
> x <- c(45, 56, 78, 12, 3, -91, -45, 15, 1, 24)
> mean(x)
[1] 9.8
```

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 = 0.2, 2 values from each end will be dropped from the calculations to find mean. In this case the sorted vector is (-91, -45, 1, 3, 12, 15, 24, 45, 56, 78) and the values removed from the vector for calculating mean are (−91, −45) from left and (56, 78) from right.

```
> mean(x, trim = 0.2)
[1] 16.66667
```

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.

```
> x <- c(45, 56, 78, 12, 3, -91, NA, -45, 15, 1, 24, NA)
> mean(x)
[1] NA
> mean(x, na.rm = TRUE)
[1] 9.8
```

5.1.2. Median

The middle most value in a data series is called the median. The *median()* function is used in R to calculate this value. The basic syntax for calculating median in R is given below along with its parameters.

median(x, na.rm = FALSE)

x - numeric vector

na.rm - to remove the missing values from the input vector

```
> x <- c(45, 56, 78, 12, 3, -91, -45, 15, 1, 24)
> median(x)
[1] 13.5
```

5.1.3. Mode

The mode is the value that has highest number of occurrences in a set of data. Unlike 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.

```
Mode <- function(x)
{
  y <- unique(x)
  y[which.max(tabulate(match(x, y)))]
}

> x <- c(1,2,3,4,5,5,5)
> Mode(x)
[1] 5
> ch <- c("a", "e", "i", "o", "u", "u", "a", "a")
> Mode(ch)
[1] "a"
```

The function *unique()* returns a vector, data frame or array like *x* but with duplicate elements/rows removed. The function *match()* returns a vector of the positions of (first) matches of its first argument in its second. The function *tabulate()* takes the integer-valued vector bin and counts the number of times each integer occurs in it. The function *which.max()* determines the location, i.e., index of the (first) maximum of a numeric (or logical) vector.

5.1.4. Standard Deviation and Variance

The functions to calculate the standard deviation, variance and the mean absolute deviation are *sd()*, *var()* and *mad()* respectively.

```
> sd(mtcars$cyl)
[1] 1.785922
> var(mtcars$cyl)
[1] 3.189516
> mad(mtcars$cyl)
[1] 2.9652
```

5.1.5. Quartile Ranges

The *quantile()* function provides the quartiles of the numeric values. An alternative function for quartiles is *fivenum()*. The *IQR()* function provides the inter quartile range of the numeric fields.

```
> quantile(mtcars$cyl)
  0%  25%  50%  75% 100%
   4    4    6    8    8
> fivenum(mtcars$cyl)
[1] 4 4 6 8 8
> IQR(mtcars$cyl)
[1] 4
```

5.1.6. Other Statistical Functions

The function *cor()* and *cov()* are used to find the correlation and covariance between two numeric fields respectively. In the below example the value shows that there is negative correlation between the two numeric fields.

```
> cor(mtcars$mpg, mtcars$cyl)
[1] -0.852162
```

```
> cov(mtcars$mpg, mtcars$cyl)
[1] -9.172379
```

There are other statistics functions such as *pmin()*, *pmax()* [parallel equivalents of *min()* and *max()* respectively], *cummin()* [cumulative minimum value], *cummax()* [cumulative maximum value], *cumsum()* [cumulative sum] and *cumprod()* [cumulative product].

```
> nrow(mtcars)
[1] 32
> mtcars$cyl
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
> pmin(mtcars$cyl)
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
> pmax(mtcars$cyl)
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
> cummin(mtcars$cyl)
 [1] 6 6 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
> cummax(mtcars$cyl)
 [1] 6 6 6 6 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
> cumsum(mtcars$cyl)
 [1]   6  12  16  22  30  36  44  48  52  58  64  72  80  88  96 104 112 116 120 124
[21] 128 136 144 152 160 164 168 172 180 186 194 198
> cumprod(mtcars$cyl)
 [1] 6.000000e+00 3.600000e+01 1.440000e+02 8.640000e+02 6.912000e+03
                                                    4.147200e+04
 [7] 3.317760e+05 1.327104e+06 5.308416e+06 3.185050e+07 1.911030e+08
                                                    1.528824e+09
[13] 1.223059e+10 9.784472e+10 7.827578e+11 6.262062e+12 5.009650e+13
                                                    2.003860e+14
```

```
[19] 8.015440e+14 3.206176e+15 1.282470e+16 1.025976e+17 8.207810e+17
                                                        6.566248e+18
[25] 5.252999e+19 2.101199e+20 8.404798e+20 3.361919e+21 2.689535e+22
                                                        1.613721e+23
[31] 1.290977e+24 5.163908e+24
```

5.2. Summary Statistics

Thus the *summary()* function can be applied on the entire dataset to get all the statistical values of all the numeric fields.

```
> summary(mtcars)
      mpg             cyl             disp             hp             drat
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080
 Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930
       wt             qsec             vs              am              gear
 Min.   :1.513   Min.   :14.50   Min.   :0.0000   Min.   :0.0000   Min.   :3.000
 1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000
 Median :3.325   Median :17.71   Median :0.0000   Median :0.0000   Median :4.000
 Mean   :3.217   Mean   :17.85   Mean   :0.4375   Mean   :0.4062   Mean   :3.688
 3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000
 Max.   :5.424   Max.   :22.90   Max.   :1.0000   Max.   :1.0000   Max.   :5.000
      carb
 Min.   :1.000
 1st Qu.:2.000
 Median :2.000
```

```
Mean   :2.812
3rd Qu.:4.000
Max.   :8.000
```

5.3. Normal Distribution

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve. The centre of the curve represents the mean of the data set. In the graph, half of values lie to the left of the mean and the other half lie to the right of the graph. This is referred as normal distribution in statistics. R has four in-built functions to generate normal distribution. They are described below.

```
dnorm(x, mean, sd)
pnorm(x, mean, sd)
qnorm(p, mean, sd)
rnorm(n, mean, sd)
```

```
x - vector of numbers
p - vector of probabilities
n - sample size
mean - mean (default value is 0)
sd - standard deviation (default value is 1)
```

5.3.1. dnorm()

For a given mean and standard deviation, this function gives the height of the probability distribution. Below is an example in which the result of the *dnorm()* function is plotted in a graph in *Fig. 5.1*.

```
> x <- seq(-5,5, by = 0.05)
> y <- dnorm(x, mean = 1.5, sd = 0.5)
> plot(x, y)
```

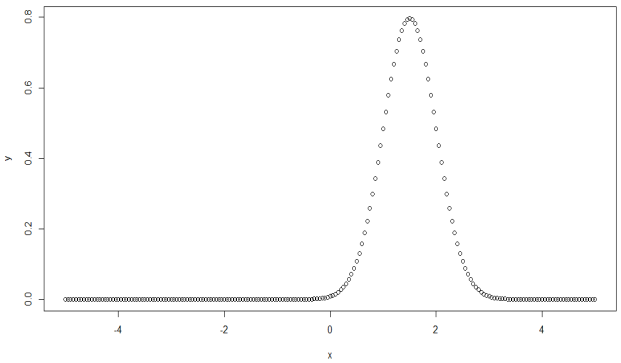

Figure 5.1 Plot of dnorm()

5.3.2. pnorm()

The *pnorm()* function returns the probability of a normally distributed random number which is less than the value of a given number. The other name for this is "*Cumulative Distribution Function*". Below is an example in which the result of the *pnorm()* function is plotted in a graph as in *Fig.* 5.2.

```
> x <- seq(-5,5, by = 0.05)
> y <- pnorm(x, mean = 1.5, sd = 1)
> plot(x, y)
```

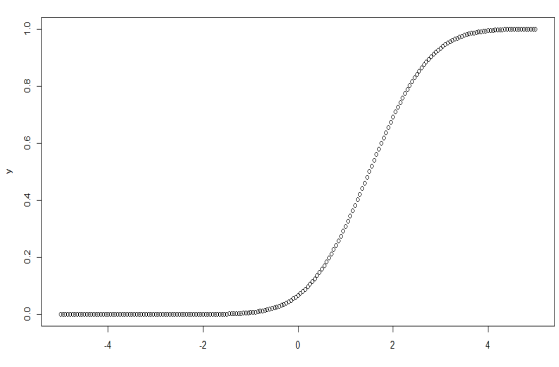

Figure 5.2 Plot of pnorm()

5.3.3. qnorm()

The *qnorm()* function takes the probability value as input and returns a cumulative value that matches the probability value. Below is an example in which the result of the *qnorm()* function is plotted in a graph as in *Fig. 5.3*.

```
> x <- seq(0, 1, by = 0.02)
> y <- qnorm(x, mean = 2, sd = 1)
> plot(x, y)
```

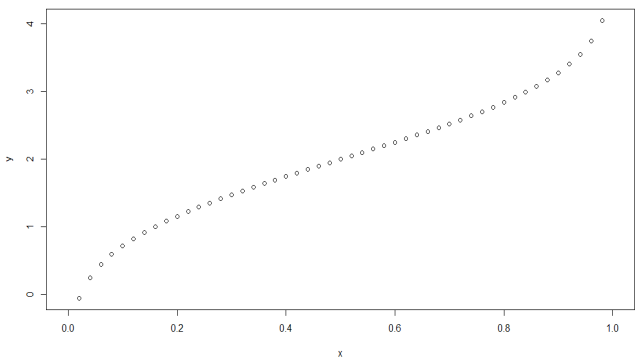

Figure 5.3 Plot of qnorm()

5.3.4. rnorm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers as in *Fig. 5.4*.

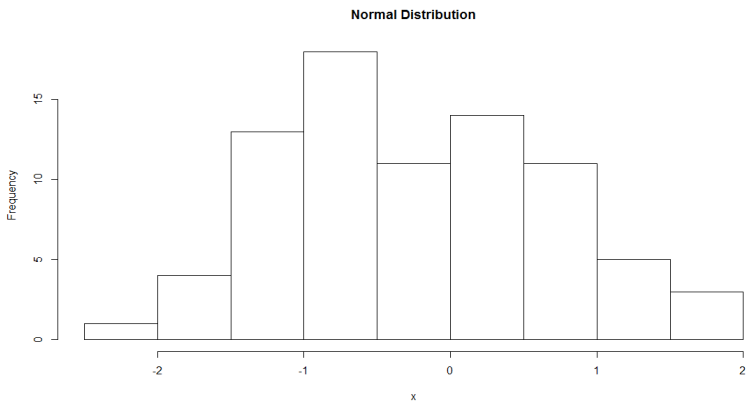

Figure 5.4 Histogram Using rnorm()

```
> x <- rnorm(80)
> hist(x, main = "Normal Distribution")
```

5.4. Binomial Distribution

The probability of success of an event is found by the binomial distribution model and this has only two possible outcomes in a series of experiments. For example, tossing of a coin always gives a head or a tail. During the binomial distribution, the probability of finding exactly 3 heads when tossing a coin for 10 times is estimated. R has four in-built functions to generate binomial distribution. They are described below.

dbinom(x, size, prob)

pbinom(x, size, prob)

qbinom(p, size, prob)

rbinom(n, size, prob)

x - vector of numbers

p - vector of probabilities

n - sample size

size – number of trials

prob – probability of success of each trial

5.4.1. dbinom()

This function gives the probability density distribution at each point. Below is an example in which the result of the *dbinom()* function is plotted in a graph as in *Fig. 5.5*.

```
> x <- seq(0, 25, by = 1)
> y <- dbinom(x,25,0.5)
> plot(x, y)
```

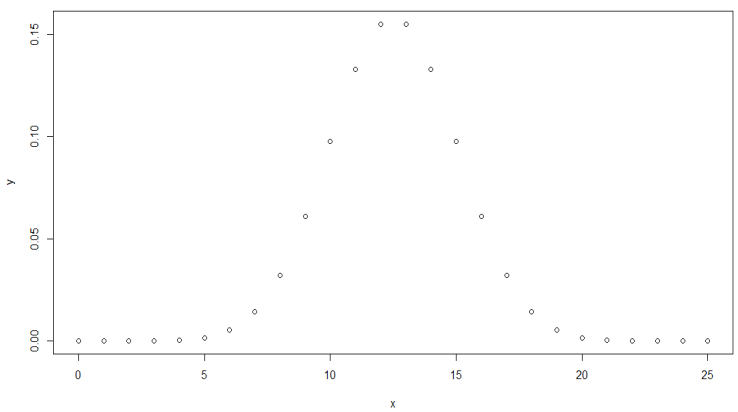

Figure 5.5 Plot Using dbinorm()

5.4.2. pbinom()

This function gives the cumulative probability of an event. It is a single value representing the probability. The probability of getting 25 or less heads from a 50 tosses of a coin is given by the below code.

```
> x <- pbinom(25,50,0.5)
> x
[1] 0.5561376
```

5.4.3. qbinom()

The function *qbinom()* takes the probability value as input and returns a number whose cumulative value matches the probability value. The below example finds how many heads will have a probability of 0.5 will come out when a coin is tossed 50 times.

```
> x <- qbinom(0.5, 50, 1/2)
> x
[1] 25
```

5.4.4. rbinom()

The function *rbinom()* returns the required number of random values of the given probability from a given sample. The below code is to find 5 random values from a sample of 50 with a probability of 0.5.

```
> x <- rbinom(5,50,0.5)
> x
[1] 24 21 22 29 32
```

5.5. Correlation Analysis

To evaluate the relation between two or more variables, the correlation test is used. Correlation coefficient in R can be computed using the functions *cor()* or *cor.test()*. The basic syntax for the correlation functions in R are as below.

```
cor(x, y, method)

cor.test(x, y, method)

x, y - numeric vectors with the same length
method - correlation method ("pearson" or "kendall" or "spearman")
```

Consider the data set "*mtcars*" available in the R environment. Let us first find the correlation between the horse power ("*hp*") and the mileage per gallon ("*mpg*") of the cars and then between the horse power ("*hp*") and the cylinder displacement ("*disp*") of the cars. From the test we find that the horse power ("*hp*") and the

mileage per gallon (“*mpg*”) of the cars have *negative correlation* (*-0.7761684*) and the horse power (“*hp*”) and the cylinder displacement (“*disp*”) of the cars have positive correlation (*0.7909486*).

```
> cor(mtcars$hp, mtcars$mpg, method = “pearson”)
[1] -0.7761684

> cor.test(mtcars$hp, mtcars$mpg, method = “pearson”)
Pearson’s product-moment correlation
data:  mtcars$hp and mtcars$mpg
t = -6.7424, df = 30, p-value = 1.788e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
-0.8852686 -0.5860994
sample estimates:
  cor
  -0.7761684

> cor(mtcars$hp, mtcars$disp, method = “pearson”)
[1] 0.7909486
> cor.test(mtcars$hp, mtcars$disp, method = “pearson”)

Pearson’s product-moment correlation
data:  mtcars$hp and mtcars$disp
t = 7.0801, df = 30, p-value = 7.143e-08
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
  0.6106794 0.8932775
sample estimates:
     cor
  0.7909486
```

The correlation results can also be viewed graphically as in Fig. 5.6. The corrplot() function can be used to analyze the correlation between the various columns of a dataset, say mtcars. After this, the correlation between individual columns can be compared by plotting it in separate graphs as in Fig. 5.7 and Fig. 5.8.

```
> library(corrplot)
> M <- cor(mtcars)
> corrplot(M, method = "number")
```

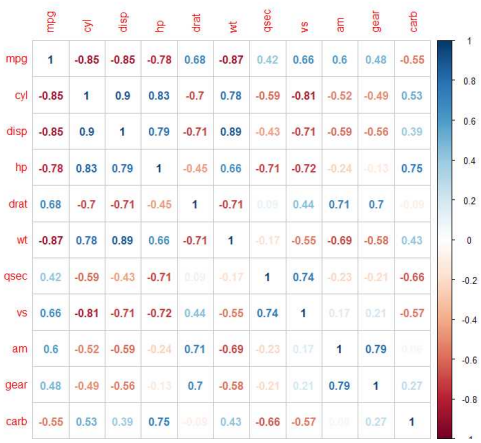

Figure 5.6 Cor Plot of mtcars Dataset

```
> plot(mtcars$hp, mtcars$mpg, xlab="Horse Power of the Cars",
                              ylab="Mileage per Gallon of the Cars", pch=21)
```

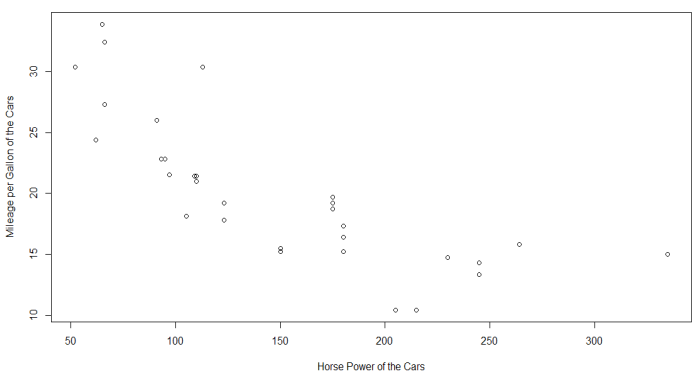

Figure 5.7 Scatter Plot of Negative Correlation

```
> plot(mtcars$hp, mtcars$disp, xlab="Horse Power of the Cars",
                    ylab="Cylinder Displacement of the Cars", pch=21)
```

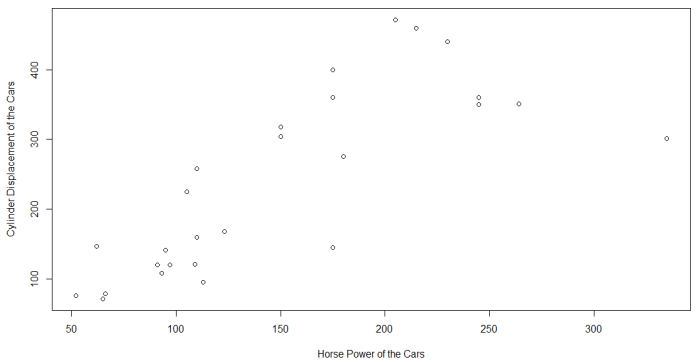

Figure 5.8 Scatter Plot of Positive Correlation

It can be noted that the graph with negative correlation (*Fig. 5.7*) has the dots from top left corner to bottom right corner and the graph with positive correlation (*Fig. 5.8*) has the dots from the bottom left corner to the top right corner.

5.6. Regression Analysis

5.6.1. Linear Regression

Regression analysis is a widely used statistical tool. A relationship model is established between the two variables used in the regression analysis. 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. Linear Regression of the two variables are related through an equation. The exponent of both the variables in this equation is 1. A linear relationship is represented by 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: $y = ax + b$, where y is the response variable, x is the predictor variable and a and b are constants which are called the coefficients.

A simple example of regression is predicting *income* of a person when his *expenditure* is known. To do this we need to have the relationship between income and expenditure of a person. First, carry out the experiment of gathering a sample of observed values of income and corresponding expenditures. Then create a relationship model using the *lm()* function in R. Find the coefficients from the model created and create the mathematical equation using these values. Now, get a summary of the relationship model to know the average error in prediction, which is also called *residuals*. Finally, to predict the income of the new persons, use the *predict()* function in R.

The function *lm()* creates the relationship model between the predictor and the response variable. The basic syntax for *lm()* function in linear regression is as given below.

```
lm(formula,data)

formula - relation between x and y
data - numeric vector

> x <- c(1510, 1740, 1380, 1860, 1280, 1360, 1790, 1630, 1520, 1310)
> y <- c(6300, 8100, 5600, 9100, 4700, 5700, 7600, 7200, 6200, 4800)
> model <- lm(y~x)
> model

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)          x
  -3845.509       6.746

> summary(model)

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

```
Residuals:

   Min     1Q  Median     3Q    Max
-630.02 -166.29    4.12  189.44  397.75

Coefficients:
     Estimate Std. Error t value Pr(>|t|)
(Intercept) -3845.5087   804.9013  -4.778  0.00139 **
x              6.7461     0.5191  12.997 1.16e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 325.3 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
```

The basic syntax for the function *predict()* in linear regression is as given below.

```
predict(object, newdata)

object - model created using lm() function
newdata - new numeric vector for predictor variable
> expense <- data.frame(x = 4700)
> income <-  predict(model, expense)
> income
      1
27861.18
```

The graphical representation of this linear regression is drawn by the below code and *Fig. 5.9*.

```
> plot(y,x,col = "blue",main = "Income & Expenditure Regression",
                                          abline(lm(x~y)),cex = 1.3,pch = 16,
                              xlab = "Income in Rs.",ylab = "Expenditure in Rs.")
```

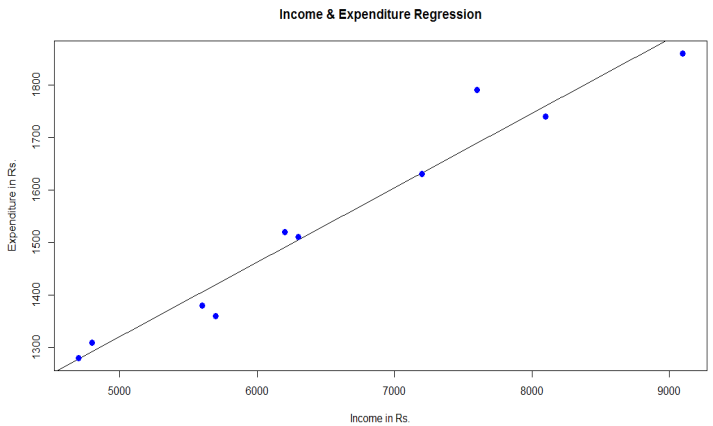

Figure 5.9 Plot of Linear Regression

5.6.2. Multiple Regressions

Multiple regressions 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 regressions we have *more than one predictor* variable and *one response* variable.

The equation for multiple regressions consists of the below variables.

$$y = a + b_1x_1 + b_2x_2 + ...b_nx_n$$

In this equation y is the response variable, a, b_1, $b_2...b_n$ are the coefficients and x_1, x_2, ...x_n are the predictor variables.

In R, the *lm()* function is used to create the regression model. 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. The relationship model is built between the predictor variables and the response variables. The basic syntax for *lm()* function in multiple regression is as given below.

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

Consider the data set "*mtcars*" available in the R environment. This dataset presents the data of 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. This model establishes 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.

```
> model2 <- lm(mpg~disp+hp+wt, data = mtcars[, c("mpg", "disp", "hp", "wt")])
> model2

Call:
lm(formula = mpg ~ disp + hp + wt, data = mtcars[, c("mpg", "disp",
    "hp", "wt")])

Coefficients:
(Intercept)        disp          hp          wt
  37.105505    -0.000937    -0.031157    -3.800891
> a <- coef(model2)[1]
> a
(Intercept)
   37.10551
> b1 <- coef(model2)[2]
> b2 <- coef(model2)[3]
> b3 <- coef(model2)[4]
> b1
        disp
-0.0009370091
> b2
         hp
-0.03115655
> b3
```

```
    wt
-3.800891
```

We create the mathematical equation below, from the above intercept and coefficient values.

$$Y = a+b_1*x_1+b_2*x_2+b_3*x_3$$

$$Y = (37.10551)+(-0.0009370091)*x_1+(-0.03115655)*x_2+(-3.800891)*x_3$$

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 = 160, hp = 110 and wt = 2.620 the predicted mileage is given by:

$$Y = (37.10551)+(-0.0009370091)*160+(-0.03115655)*110+(-3.800891)*2.620$$
$$= 23.57003$$

The above value can also be calculated using the function *predict()* for the given new value.

```
> newdata <- data.frame(disp = 160, hp = 110, wt = 2.620)
> mileage <-  predict(model2, newdata)
> mileage
    1
23.57003
```

The graphical representation of this multiple regression is drawn by the below code and in *Fig. 5.10*.

```
> plot(mtcars$mpg, mtcars$disp+mtcars$hp+mtcars$wt, col = "blue",
    main = "Mileage Per Gallons & Other Factors Multiple Regression",
    abline(lm(mtcars$disp+mtcars$hp+mtcars$wt~mtcars$mpg)),
    cex = 1.5, pch = 15,
    xlab = "Other Facors (mtcars$disp+mtcars$hp+mtcars$wt)",
    ylab = "Mileage Per Gallon")
```

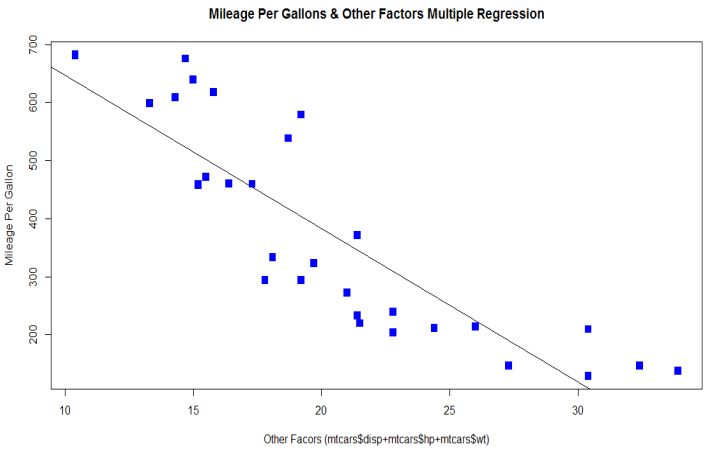

Figure 5.10 Plot of Multiple Regressions

5.6.3. Logistic Regression

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables. The general mathematical equation for logistic regression is as given below.

$$y = 1 / (1 + e^{\wedge} -(a + b_1x_1 + b_2x_2 + b_3x_3 + ... + b_nx_n))$$

In this equation y is the response variable, a, b_1, b_2...b_n are the coefficients and x_1, x_2, ...x_n are the predictor variables.

The function used to create the logical regression model is the *glm()* function. The basic syntax for *glm()* function in logistic regression is as given below.

glm(formula,data,family)

formula - relation between the variables x_1, x_2, ...x_n and y

data - numeric vectors of data

family - takes the value "binomial" for logistic regression

The in-built data set "*mtcars*" describes different models of a car with their various engine specifications. In "*mtcars*" data set, the column *am* describes the transmission mode with a binary value. A logistic regression model is built between the columns "*am*" and 3 other columns - *hp*, *wt* and *cyl*.

```
> model3 <- glm(am ~ cyl + hp + wt, data = mtcars[, c("am", "cyl", "hp", "wt")],
                                                        family = binomial)
> model3

Call:  glm(formula = am ~ cyl + hp + wt, family = binomial, data = mtcars[
    c("am", "cyl", "hp", "wt")])

Coefficients:
(Intercept)         cyl          hp          wt
   19.70288     0.48760     0.03259    -9.14947

Degrees of Freedom: 31 Total (i.e. Null);  28 Residual
Null Deviance:          43.23
Residual Deviance: 9.841    AIC: 17.84

> summary(model3)

Call:
glm(formula = am ~ cyl + hp + wt, family = binomial, data = mtcars[,
                                                        c("am", "cyl", "hp", "wt")])

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.17272  -0.14907  -0.01464   0.14116   1.27641

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) 19.70288    8.11637   2.428   0.0152 *
```

```
cyl          0.48760    1.07162   0.455   0.6491
hp           0.03259    0.01886   1.728   0.0840 .
wt          -9.14947    4.15332  -2.203   0.0276 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.2297  on 31  degrees of freedom
Residual deviance:  9.8415  on 28  degrees of freedom
AIC: 17.841

Number of Fisher Scoring iterations: 8
```

The *p-value* in the summary is greater than *0.05* for the variables "*cyl*" (*0.6491*) and "*hp*" (*0.0840*). This value is considered to be insignificant in contributing to the value of the variable "*am*". Only weight "*wt*" (*0.0276*) impacts the "*am*" value in this regression model.

5.6.4. Poisson Regression

Poisson Regression involves regression models in which the response variable is in the form of counts and not fractional numbers. For example, the count of number of births or number of wins in a football match series. Also the values of the response variables follow a Poisson distribution. The Poisson Regression is represented by the below equation.

$$log(y) = a + b_1x_1 + b_2x_2 + b_nx_n.....$$

In this equation y is the response variable, a, b_1, b_2...b_n are the coefficients and x_1, x_2, ...x_n are the predictor variables.

The function used to create the Poisson regression model is the *glm()* function. The basic syntax for *glm()* function in logistic regression is as given below.

```
glm(formula, data, family)
```

formula - relation between the variables x_1, x_2, ...x_n and y

data - numeric vectors of data

family - takes the value "poisson" for poisson regression

The data set "*warpbreaks*" describes the effect of wool type and tension on the number of warp breaks per loom. Let's consider "*breaks*" as the response variable which is a count of number of breaks. The wool "*type*" and "*tension*" are taken as predictor variables. The model so built shows the below results.

```
> model4 <- glm(formula = breaks ~ wool + tension, data = warpbreaks,
                                                    family = poisson)
> summary(model4)

Call:
glm(formula = breaks ~ wool + tension, family = poisson, data = warpbreaks
Deviance Residuals:
    Min      1Q  Median      3Q     Max
-3.6871 -1.6503 -0.4269  1.1902  4.2616
Coefficients:
  Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.69196    0.04541  81.302  < 2e-16 ***
woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
tensionM    -0.32132    0.06027  -5.332 9.73e-08 ***
tensionH    -0.51849    0.06396  -8.107 5.21e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 297.37  on 53  degrees of freedom
  Residual deviance: 210.39  on 50  degrees of freedom
  AIC: 493.06
```

Number of Fisher Scoring iterations: 4

In the summary we look for the p-value in the last column to be less than 0.05 to consider an impact of the predictor variable on the response variable. As seen the wooltype B having tension type M and H have impact on the count of breaks.

p-value of woolB = 6.49e-05 = 0.0000649 < 0.05

p-value of tensionM = 9.73e-08 = 0.0000000973 < 0.05

p-value of tensionH = 5.21e-16 = 0.000000000000000521 < 0.05

5.6.5. Non-Linear Least Square Regression

When modelling real world data for regression analysis, we observe that it is rarely the case that the equation of the model is a linear equation giving a linear graph. The equation of the model of real world data involves mathematical functions of higher degree. In such a scenario, the plot of the model gives a curve rather than a line. Both linear and non-linear regression aims to adjust the values of the model's parameters. This is to find the line or curve that comes nearer to the data. On finding these values we will be able to estimate the response variable with good accuracy.

In Least Square regression, a regression model is established. In this regression model, the sum of the squares of the vertical distances of different points from the regression curve is minimized. We generally start with a defined model and assume some values for the coefficients. The *nls()* function of R is then apllied to get the more accurate values and the confidence intervals. The basic syntax for creating a nonlinear least square test in R is as below.

nls(formula, data, start)

formula - a nonlinear model formula including variables and parameters

data – a data frame

start - list or vector of starting estimates

We will consider a nonlinear model with assumption of initial values of its coefficients. Next we will see what the confidence intervals of these assumed values so that we can judge how well these values fit into the model. Consider the below equation.

$a = b_1 * x \wedge 2 + b_2$

Let us assume the initial coefficients to be 1 and 3 and fit these values into *nls()* function.

```
> x <- c(1.6, 2.1, 2, 2.23, 3.71, 3.25, 3.4, 3.86, 1.19, 2.21)
> y <- c(5.19, 7.43, 6.94, 8.11, 18.75, 14.88, 16.06, 19.12, 3.21, 7.58)
> plot(x, y)
> model <- nls(y ~ b1*x^2+b2, start = list(b1 = 1,b2 = 3))
> new <- data.frame(x = seq(min(x), max(x), len = 100))
> lines(new$x, predict(model, newdata = new))
> res1 <- sum(resid(model)^2)
> res1
[1] 1.081935
> res2 <- confint(model)
> res2
      2.5%    97.5%
b1 1.137708 1.253135
b2 1.497364 2.496484
```

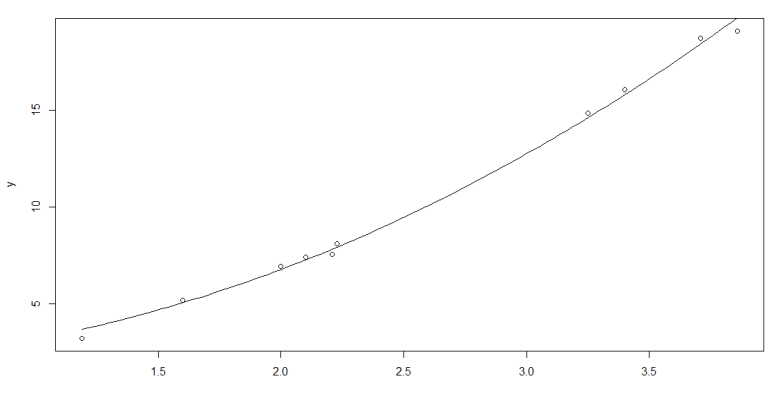

Figure 5.11 Plot of Non Linear Regressions

We can conclude that the value of b_1 is more close to 1 (*1.253135*) while the value of b_2 is more close to 2 (*2.496484*) and not 3.

5.7. Analysis of Variance (ANOVA)

Analysis of Variance (ANOVA) is a statistical measure that is used for investigating data by comparing the means of subsets of the data. The base case is the *one-way* ANOVA. In *one-way* ANOVA the data is sub-divided into groups based on a single classification factor. The one-way ANOVA is used to verify if the means of many groups are equals. But this analysis may not be very useful for complex problems. For example, it may be necessary to take into account two factors of variability to determine if the averages between the groups depend on the group classification or the second variable that is to consider. In this case the two-way analysis of variance (*two-way* ANOVA) should be used.

Consider the dataset *PlantGrowth* available in R for performing *one-way* ANOVA using R. This dataset has two columns, the control group / treatment and the weight of the plant indicating its growth. We want to check if the hypothesis that the control group / treatment has effect on the plant weight / plant growth. The below code does the same.

```
> plant = lm(PlantGrowth$weight ~ PlantGrowth$group)
> anova(plant)
Analysis of Variance Table

Response: PlantGrowth$weight
    Df  Sum Sq Mean Sq F value  Pr(>F)
PlantGrowth$group  2  3.7663  1.8832  4.8461 0.01591 *
Residuals         27 10.4921  0.3886
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The result shows that the *F-value* is *4.8461* and the *p-value* is *0.01591* which is less than *0.05* (*5% level of significance*). This shows that the null hypothesis is rejected, that is the control group / treatment has effect on the plant growth / plant weight.

For *two-way* ANOVA, consider the below example of *revenues* collected for *5 years* in *each month*. We want to see if the revenue depends on the Year and / or Month or if they are independent of these two factors.

```
> revenue = c(15,18,22,23,24, 22,25,15,15,14, 18,22,15,19,21,
+         23,15,14,17,18, 23,15,26,18,14, 12,15,11,10,8, 26,12,23,15,18,
+         19,17,15,20,10, 15,14,18,19,20, 14,18,10,12,23, 14,22,19,17,11,
+         21,23,11,18,14)

> months = gl(12,5)
> years = gl(5, 1, length(revenue))
> fit = aov(revenue ~ months + years)

> anova(fit)
Analysis of Variance Table

Response: revenue
    Df Sum Sq Mean Sq F value Pr(>F)
months   11 308.45  28.041  1.4998 0.1660
years     4  44.17  11.042  0.5906 0.6712
Residuals 44 822.63  18.696
```

The significance of the difference between months is: $F = 1.4998$. This value is lower than the value tabulated and indeed $p\text{-}value > 0.05$. So we cannot reject the *null hypothesis*: the means of revenue evaluated according to the months are not proven to be not equal, hence we remain in our belief that the variable *"months" has no effect on revenue*.

The significance of the difference between years is: $F = 0.5906$. This value is lower than the value tabulated and indeed $p\text{-}value > 0.05$. So we fail to reject the *null hypothesis*: the means of revenue evaluated according to the years are not found to be un-equal, then the variable *"years" has no effect on revenue*.