Chapter 3

R Overview

3.1 Introduction to R and Installation

R is a language and environment, which was developed by the Bell Laboratories and is primarily used for statistical computing and graphics. R environment comes with huge computation and data analysis capabilities which include variety of statistical methods like linear and non-linear modelling, statistical tests, time-series analysis methods, classification techniques and clustering techniques, conjoint analysis and multi-dimensional scaling etc. R is an open-source, freely available powerful software environment which has made it a primary attraction among the users.

As mentioned in the site (https://www.r-project.org/about.html), the R environment includes –

- An effective data handling and storage facility,
- A suite of operators for calculations on arrays, in particular matrices,
- A large, coherent, integrated collection of intermediate tools for data analysis,
- Graphical facilities for data analysis and display either on-screen or on hardcopy, and
- A well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

The R environment contains the base installation and many more additional packages (to be downloaded and installed as and when required) which provide necessary capability to work on large dataset and perform necessary analysis using advanced statistical and machine learning techniques.

As mentioned earlier, R environment is freely available over internet and the packages (a collection of R functions) which provide the capability to perform advanced analytics using the

environment can be accessed by the users through downloading and installing in the environment.

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. R suite has the following advantages –

- R has an effective data handling and storage facility
- R has a suite of operators for calculation on arrays, in particular matrices
- R has graphical facilities for data analysis and display
- R language has conditionals, loops and user-defined recursive functions and I/O facilities

R-Installation

The R basic environment can be downloaded from the following link –

https://cran.r-project.org/bin/windows/base/

The users may also use the R-Studio package which can be installed after basic R environment installation. The R-Studio comes with many advantages above the basic R environment. Some of the advantages are delineated below –

- Integrated view where datasets can be viewed directly
- Real-time Code suggestion
- Better visibility and accessibility

R-studio can be downloaded from the following link –

https://www.rstudio.com/products/rstudio/download/

The R-studio needs to be installed after the basic installation of R environment.

Installing R Packages

The basic functionalities in R comes with the BASE package, however, true power of R

environment can only be utilized if other libraries are installed in R. R can provide numerous functionalities through plenty of useful libraries. The way of downloading and calling each library during each run-time session is as follows –

```
# Downloading the 'kernlab' library
>install.packages("kernlab")
# Calling the 'kernlab' package during run-time
>library(kernlab)
```

3.2 Basics of Programming in R

In order to start using R effectively, it is necessary to learn basics of working in R. Few important concepts in terms of R data structure will be introduced in this section.

Vectors – The fundamental R data structure is Vector, which is supposed to store an ordered set of values called elements. The vector can store any type of values, however, the values stored in a particular vector should be of the same type.

Any vector can be defined in R in the following manner –

```
# Defining a vector only with strings
>state_capitals <- c('Kolkata', 'Bengaluru', 'Chennai')
# Defining a vector with numeric values
>Height <- c(170.5, 176, 182.4)
```

A particular element in the vector can be using the following code –

```
# Calling the 2nd element in vector 'state_capitals'
>state_capitals[2]
[1] "Bengaluru"
```

Few elements from a vector can be called using the following code –

```
# Calling the 1st two elements in vector 'state_capitals'
>state_capitals[1:2]
[1] "Kolkata" "Bengaluru"
```

Factors - A factor is a special case of vector that is solely used to represent categorical or ordinal variables. The factors can be represented in the following manner –

```
# Defining a factor describing 5 individuals who are married or unmarried
>mar_status <- factor(c('Married', 'Unmarried', 'Married', 'Married', 'Unmarried'))
>mar_status
[1] Married   Unmarried Married   Married   Unmarried
Levels: Married Unmarried
```

Lists – A list is a data structure like vector. Vector can hold elements of only one type at a time. However, lists can hold elements of any type simultaneously.

```
# Defining a list containing name, age, address and employer details of an individual
>list_example <- list('Harris', 24, 'London', 'Cognizant')mar_status
```

The output for displaying a list will be as follows –

```
# Displaying a list
>list_example
[[1]]
[1] "Harris"


[[2]]
[1] 24


[[3]]
[1] "London"


[[4]]
[1] "Cognizant"
```

Data Frames – Data Frame is the most important R Data Structure and is used mostly. Data frame is the structure which is equivalent to a database or a spread-sheet (since it has both rows and columns). A data frame can be created by combining a number of individual vectors. The following example will help in understanding the creation of data frame from multiple vectors in better manner.

Suppose, there are 4 vectors – name, age, company and city. Each of these 4 vectors contain details about 4 individuals. It is needed to join these vectors to get a data frame ‘emp_details’

which will be having all information about the individual at one place.

```
>name <- c('Harris', 'John', 'Mac', 'Steffy')
>age <- c(24, 26, 27, 22)
>company <- c('IBM', 'Cognizant', 'IBM', 'HCL')
>city <- c('London', 'Paris', 'Bengaluru', 'Tokyo')
>emp_details <- data.frame(name, age, company, city, stringsAsFactors = FALSE)
```

The output of data frame ‘emp_details’ will be visible as follows –

```
>emp_details
   name age   company      city
1 Harris  24       IBM    London
2   John  26 Cognizant     Paris
3    Mac  27       IBM Bengaluru
4 Steffy  22       HCL     Tokyo
```

Data frames are mostly used in R environment for analysis of data.

Matrix – Matrix is another type of data structure which is used in R. A matrix is a data structure that represents a two-dimensional table with rows and columns of data. Like vectors, R matrix can contain any one type of data, although they are most often used for mathematical operations and hence mostly store the numerical values.

Creation of matrix can be understood from the following set of codes.

```
>mat1 <- matrix(c(12,34,56,87), nrow = 2)
>mat1
     [,1] [,2]
[1,]   12   56
[2,]   34   87
```

```
>mat2 <- matrix(c(12,34,56,87), ncol = 2)
>mat2
     [,1] [,2]
[1,]   12   56
[2,]   34   87
```

3.3 Importing Data in R

In the real-life scenario, it is needed to work with large datasets which are generally collected as Excel or CSV file. CSV file can be created using the Excel application and one of the commonest format for uploading data into the R environment.

```
# Importing CSV file 'Purchase.csv' into R
>test_data <- read.csv('D:/R_FOLDER/imdb.csv', stringAsFactors = FALSE)
# Checking whether the file uploaded successfully or not
>test  data
```

The excel file can be uploaded into the R environment using ‘readxl’ library.

```
# Downloading ‘readxl’ library
>install.packages(“readxl”)
# Loading ‘readxl’ in the R environment
>library(readxl)
# Importing xls file into the system
>test_data <- read_excel('D:/R_FOLDER/imdb.xls')
# Importing xlsx file into the system
>test_data <- read_excel('D:/R_FOLDER/imdb.xlsx')
```

The data is uploaded as data frame. By default, R assumes that the CSV, XLS, XLSX files include a header line listing the names of the features in the dataset.

3.4 Data Manipulation in R

After uploading the data, it is needed to understand whether the data has been uploaded correctly or not. Before getting insights from the data, it is needed to understand the structure of the data.

```
# Reading a CSV file into a dataframe
>letters <- read.csv("D:/Documents/ML/letterdata.csv")
# Checking the structure of the data frame
>str(letters)
'data.frame':   20000 obs. of  5 variables:
 $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...
 $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...
 $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...
 $ width : int  3 3 6 6 3 5 5 3 4 13 ...
 $ height: int  5 7 8 6 1 8 4 2 4 9 ...
```

The structure of the data mainly tells – (a) how many fields are there in the dataset, (b) what are the fields in the dataset and (c) what are the datatypes of different fields. The structure of an imported dataset can be seen using the 'str()' command.

The str() function provides a method to display the structure of the R data frames, vectors or lists.

The data frame 'letters' contain 5 variables – letter, xbox, ybox, width and height. The properties of each variable are mentioned against each variable in the output.

Creating a new variable from the original variable using transformation can be done using the following code.

```
> letters$log_ybox <- log(letters$ybox)
> str(letters)
'data.frame':   20000 obs. of  18 variables:
 $ letter  : Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...
 $ xbox    : int  2 5 4 7 2 4 4 1 2 11 ...
 $ ybox    : int  8 12 11 11 1 11 2 1 2 15 ...
 $ width   : int  3 3 6 6 3 5 5 3 4 13 ...
 $ height  : int  5 7 8 6 1 8 4 2 4 9 ...
 $ log_ybox : num  2.08 2.48 2.4 2.4 0 ...
```

3.5 Performing Data Analysis using R

After uploading the data, the major step that needs to be performed is analyzing the data for getting some insights from it. This section is on how to perform basic analysis in a data set. The most common insight about any dataset is enquiring about mean, median and quantiles. The 'summary()' function in R gives this information in a dataset.

The 'iris' is a standard dataset available in base R. The following code will give summary of the dataset 'iris'. The summary

```
> summay(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

mmary will include information like minimum, 1st quantile, median, mean, 3rd quantile and maximum.

It is possible to get mean and median individually also and for particular field in the dataset. The following code can be used for that.

```
> mean(iris$Sepal.Length)
[1] 5.843333
> median(iris$Sepal.Length)
[1] 5.8
```

It is possible to find interquartile range (3rd Quartile – 1st Quartile) using the following code –

```
> IQR(iris$Sepal.Length)
[1] 1.3
```

It is possible to find out the quartiles separately using the following codes –

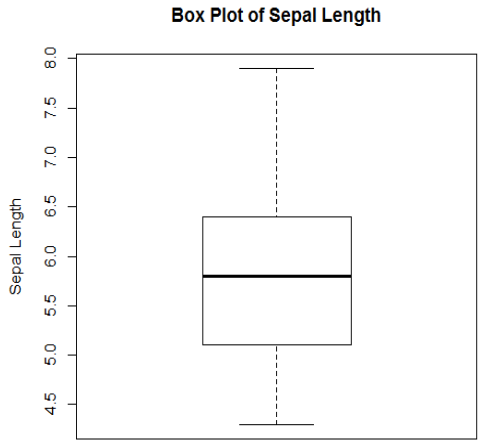
```
> quantile(iris$Sepal.Length)
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
> quantile(iris$Sepal.Length, seq(from = 0, to = 1, by = 0.2))
  0%  20%  40%  60%  80% 100%
4.30 5.00 5.60 6.10 6.52 7.90
```

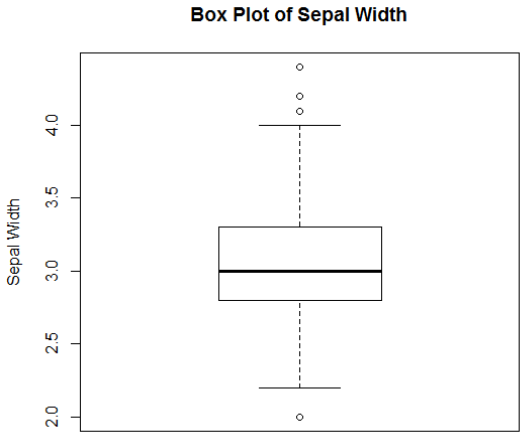
The following code is used to get spread or variation in a dataset.

```
# Finding Standard Deviation
> sd(iris$Sepal.Length)
[1] 0.8280661
# Finding Variance
> var(iris$Sepal.Length)
[1] 0.6856935
# Finding Range (minimum and maximum)
> range(iris$Sepal.Length)
[1] 4.3 7.9
```

Box-plot reveals how the dataset is stretched and also reveals whether there is any outlier in the dataset. The presence of outlier in the dataset increases variability in the dataset. The following code is used to create box-plot. In the 2nd plot, the outliers are present and are denoted by the dotted circles.

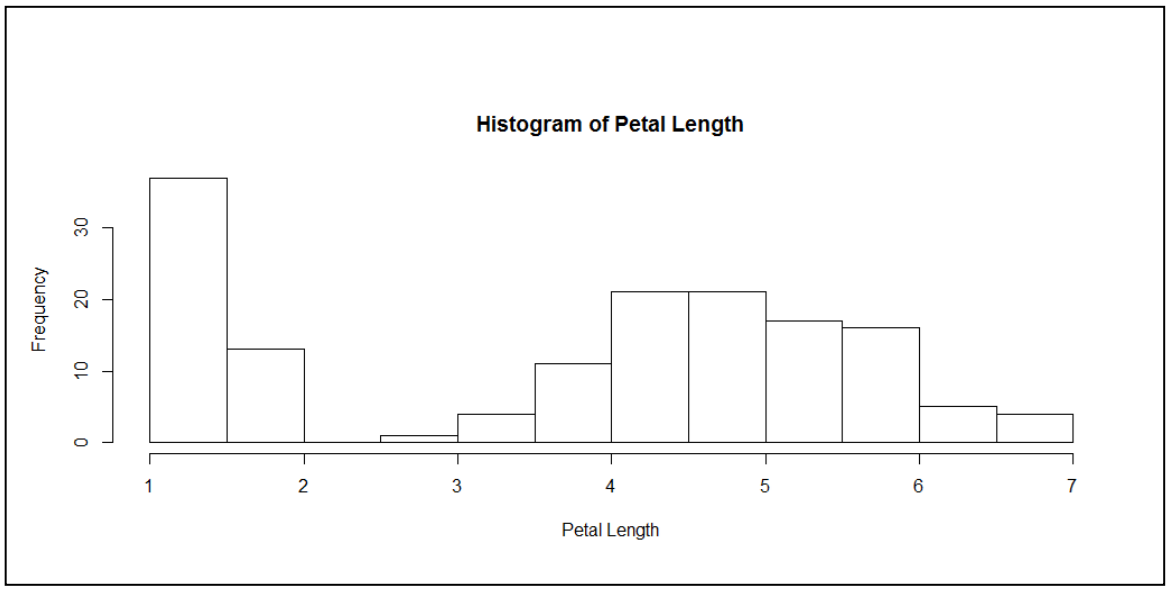
```
# Box Plot
> boxplot(iris$Sepal.Length, main = "Box Plot of Sepal Length", ylab = "Sepal Length")
> boxplot(iris$Sepal.Width, main = "Box Plot of Sepal Width", ylab = "Sepal Width")
```





Histogram is also used for depicting the spread of a numeric variable. It divides the variable's values into a predefined number of portions or bins that act as containers for values. The following code is used for creation of histogram.

```
> hist(iris$Petal.Length, main = "Histogram of Petal Length", xlab = "Petal Length")
```



How to create basic plots in R is demonstrated in the current section. In the next section, discussion is done on how R can be used for elementary data analysis on a real-life dataset to get elementary insights.

3.6 Problem-Oriented Data Analysis in R Environment

Working with customer data reveals many useful insights which can lead to better decisions related to marketing strategy. In this section, analysis will be performed on an e-commerce portal data. The data (in CSV format) can be downloaded from the following link - https://drive.google.com/file/d/1D4EjSvMbFSOTBAM_7VaVSh3Yj63-6WPi/view?usp=sharing.

The dataset contains the information about users' engagement in an e-commerce portal which include –

- Gender of the user
- Marital Status of the user
- Price of the item the user has viewed
- Number of items the user has purchased before
- Total purchase value of all the items the user has purchased previously
- Income of the user
- Whether the user has purchased the current item

The data is imported in R using the following code –

```
> purchase <- read.csv("C:/Users/subhabahap/Documents/Python Scripts/Purchase2.csv")
> View(purchase)
```

	USER_GENDER	MARITAL	PRICE	NO_ITEMS	PUR_VALUE	USER_INCOME	PURCHASE
1	1	M	25000	0	0	39171	0
2	1	U	20000	2	21866	249	1
3	0	U	30000	1	16090	1249	0
4	0	U	15000	0	0	7247	1
5	1	U	28000	2	26888	33314	1
6	0	M	14000	1	2800	0	0
7	1	M	20000	1	4035	27436	1
8	1	U	23000	1	15125	31514	0
9	0	U	14000	2	500000	8745	1
10	0	M	30000	2	5281	0	1
11	0	M	14000	0	0	0	0
12	1	M	9000	0	0	40087	0
13	0	U	32000	0	0	17116	0
14	0	U	14000	1	6371	0	1
15	0	U	15000	1	19751	5850	1
16	0	M	16000	0	0	0	0
17	0	M	32000	1	4043	0	1
18	1	U	32000	1	12368	1254	0
19	0	U	14000	1	10101	1968	1
20	1	U	10000	2	12545	38461	1
21	1	U	23000	1	9461	479	0
22	1	U	12000	0	0	25280	0
23	1	U	14000	0	0	280	1
24	1	U	30000	1	11896	25871	0
25	0	U	19000	0	0	0	0
26	1	U	13000	2	8028	431	1
27	1	U	15000	2	13067	32615	1
28	0	M	20000	0	0	0	0
29	0	M	16000	0	0	15054	0
30	1	M	15000	1	16248	39640	1
31	1	U	10000	1	5478	4230	0

In order to check the structure of the data uploaded, the following code can be used –

```
> str(purchase)
'data.frame':   11157 obs. of  7 variables:
 $ USER_GENDER: int  1 1 0 0 1 0 1 1 0 0 ...
 $ MARITAL    : Factor w/ 2 levels "M","U": 1 2 2 2 2 1 1 2 2 1 ...
 $ PRICE      : int  25000 20000 30000 15000 28000 14000 20000 23000 14000 30000 ...
 $ NO_ITEMS   : int  0 2 1 0 2 1 1 1 2 2 ...
 $ PUR_VALUE  : int  0 21866 16090 0 26888 2800 4035 15125 500000 5281 ...
 $ USER_INCOME: int  39171 249 1249 7247 33314 0 27436 31514 8745 0 ...
 $ PURCHASE   : int  0 1 0 1 1 0 1 0 1 1 ...
```

After upload of the data it is needed to check whether there is any missing value in the data. If there is any missing value, it is needed to impute new data in place of the null values.

```
> sum(is.na(purchase))
[1] 15
> sum(is.na(purchase$USER_GENDER))
[1] 0
> sum(is.na(purchase$MARITAL))
[1] 0
> sum(is.na(purchase$PRICE))
[1] 12
> sum(is.na(purchase$NO_ITEMS))
[1] 0
> sum(is.na(purchase$PUR_VALUE))
[1] 2
> sum(is.na(purchase$USER_INCOME))
[1] 1
> sum(is.na(purchase$PURCHASE))
[1] 0
```

It is evident that in the whole dataset there are 15 null values. In the price variable, there are 12 null values in the 'PRICE' field, 2 missing values in the 'PUR_VALUE' field and 1 missing value in the 'USER_INCOME' field.

There are few ways to impute data in place of missing values.

1. Imputing column mean in place of missing values –

```
> purchase$PRICE [is.na(purchase$PRICE)] <- round(mean(purchase$PRICE, na.rm = TRUE))
```

2. Imputing column median in place of missing values –

```
> purchase$PRICE [is.na(purchase$PRICE)] <- round(median(purchase$PRICE, na.rm = TRUE))
```

3. Imputing values of the previous records in place of missing values –

```
> purchase$PUR_VALUE <- na.locf(purchase$PUR_VALUE)
```

4. Imputing interpolated value based on the previous and the next record values in place of the missing values –

```
> purchase$PUR_VALUE <- na.approx(purchase$PUR_VALUE)
```

After the missing values are imputed, it is needed to check basic statistics like mean, median etc. of the whole dataset.

```
> summary(purchase)
```

```
 USER_GENDER     MARITAL      PRICE          NO_ITEMS        PUR_VALUE       USER_INCOME          PURCHASE
Min.   :0.0000   M:3086   Min.   : 9000   Min.   :0.0000   Min.   :     0   Min.   :     0.0   Min.   :0.0000
1st Qu.:0.0000   U:8071   1st Qu.:13000   1st Qu.:0.0000   1st Qu.:     0   1st Qu.:   560.8   1st Qu.:0.0000
Median :0.0000            Median :16000   Median :1.0000   Median :  3505   Median :  5723.0   Median :0.0000
Mean   :0.4991            Mean   :18239   Mean   :0.6665   Mean   :  6717   Mean   : 11197.8   Mean   :0.3313
3rd Qu.:1.0000            3rd Qu.:24000   3rd Qu.:1.0000   3rd Qu.: 12616   3rd Qu.: 18018.8   3rd Qu.:1.0000
Max.   :1.0000            Max.   :32000   Max.   :2.0000   Max.   :700000   Max.   :900000.0   Max.   :1.0000
                          NA's   :12                                        NA's   :1
```

The summary gives an overview of the data.

It is needed to check whether there is anomaly in the dataset. For that we will be needing to check whether there is any outlier in the dataset. The observations are generally treated as outliers if any of the following conditions are satisfied –

1. Observation > 3rd Quartile + 1.5 * (Inter-quartile Range)

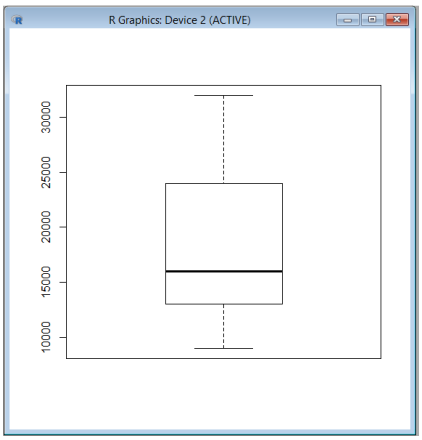
   Observation < 1st Quartile – 1.5 * (Inter-quartile Range)

2. Observation > Mean + 3 * (Standard Deviation)

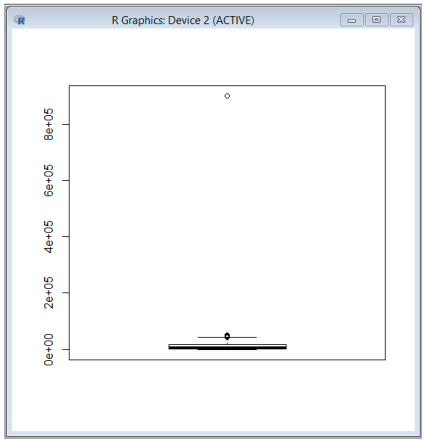
   Observation < Mean – 3 * (Standard Deviation)

Box-plot reveals whether there is any outlier in the data.

```
> boxplot(purchase$PUR_VALUE)
```




```
> boxplot(purchase$USER_INCOME)
```

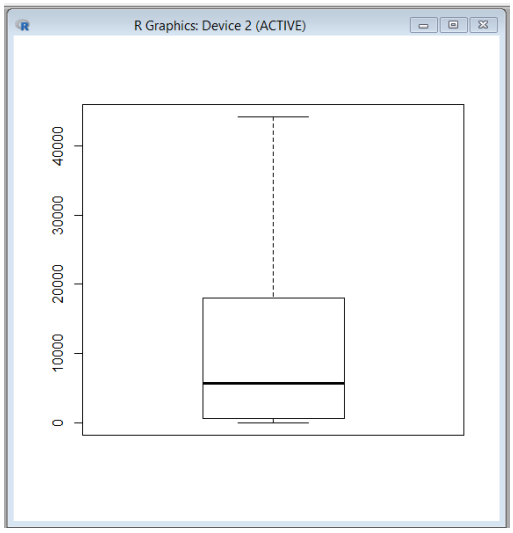


The boxplot for the field 'USER_INCOME' shows that there are outliers in the data.

The outliers in the data can be capped at the upper limit (3rd Quartile + 1.5 * (Inter-quartile Range)) as well as the lower limit (1st Quartile – 1.5 * (Inter-quartile Range)).

In order to clip the outliers, the following code can be used.

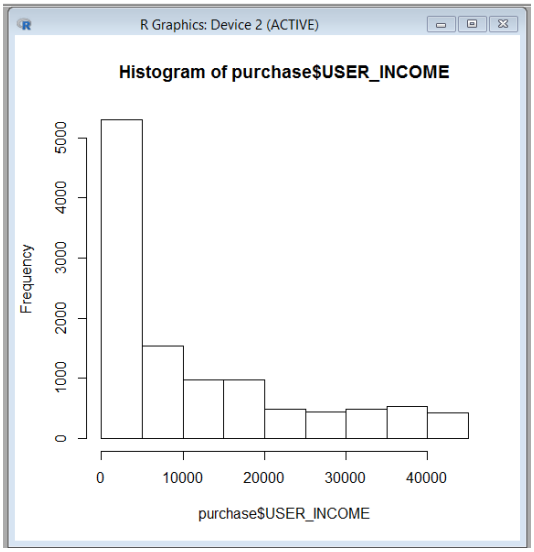
```
> q <- quantile(purchase$USER_INCOME)
> q
   0%   25%   50%   75%  100%
    0   560  5722 18018 900000
> LL = q[2] - 1.5*(q[4] - q[2])
> UL = q[4] + 1.5*(q[4] - q[2])
> purchase$USER_INCOME[purchase$USER_INCOME < LL] <- LL
> purchase$USER_INCOME[purchase$USER_INCOME > UL] <- UL
> boxplot(purchase$USER_INCOME)
```



Checking of the boxplot reveals that there is no more outlier in the data.

Histogram gives an insight on how data points are spread.

```
> hist(purchase$USER_INCOME)
```



From the histogram, it is revealed that the 'USER_INCOME' data is rightly skewed.

The R-codes used for exploratory data analysis have been discussed in this chapter. The basic insights that are obtained from the preliminary study are the stepping stones for the more detailed analysis afterwards with appropriate machine learning techniques which will be discussed in the next chapter.