**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp. No.** | **Date** | **Experiment Name** | **Page No.** | **Signature** |
| 1. |  | Install, configure and run Hadoop and R | 2 |  |
| 2. |  | Implement word count / frequency programs using MapReduce | 6 |  |
| 3. |  | Implement an MR program that processes a Weather Dataset | 8 |  |
| 4 a. |  | R – Programming Introduction and Basics | 11 |  |
| 4 b. |  | Implement Linear and Logistic Regression | 30 |  |
| 5 a. |  | Implement SVM Classification Techniques | 32 |  |
| 5 b. |  | Implement Decision Tree Classification Techniques | 35 |  |
| 6. |  | Implement Clustering Techniques | 37 |  |
| 7. |  | Visualize data using any plotting framework | 39 |  |
| 8. |  | Implement an application that stores big data in Hbase / MongoDB / Pig using Hadoop / R | 45 |  |

|  |  |
| --- | --- |
| **Exp No: 1** | **Install, configure and run Hadoop and R** |
| **Date:** |

**AIM**

To install, configure and run Hadoop and R.

**PROCEDURE**

* Hadoop Installation

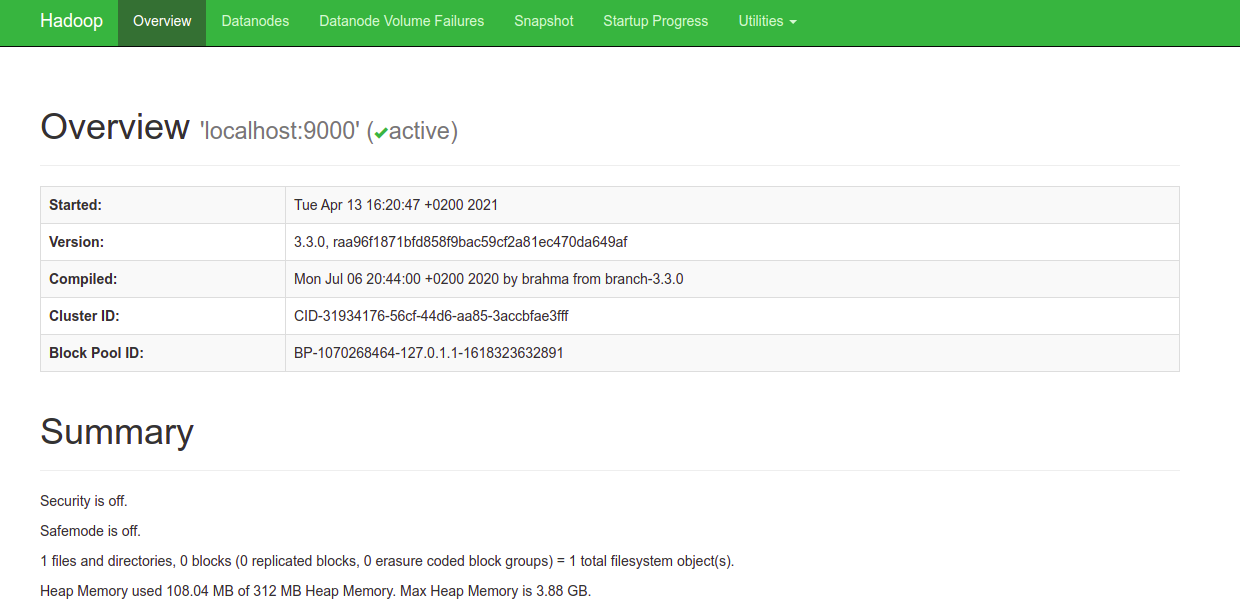
Run the following command on ubuntu terminal:

1. Install Hadoop on ubuntu.
   1. sudo apt update
   2. sudo apt install openjdk-8-jdk -y
2. Check if java is installed
   1. java -version
   2. javac -version
3. Install SSH server
   1. sudo apt install openssh-server openssh-client -y
4. Create a new user in ubuntu
   1. sudo adduser hdoop
   2. sudo adduser hdoop sudo
   3. su – hdoop
5. Create rsa file
   1. ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa
   2. cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys
   3. chmod 0600 ~/.ssh/authorized\_keys
6. Open ssh server
   1. ssh localhost
7. Download Hadoop
   1. wget https://downloads.apache.org/hadoop/common/hadoop-3.2.3/hadoop-3.2.3.tar.gz
   2. tar xzf hadoop-3.2.3.tar.gz
8. Edit bashrc
   1. sudo nano .bashrc
   2. Add the following lines at the end of the file
      1. export HADOOP\_HOME=/home/hdoop/hadoop-3.2.3
      2. export HADOOP\_INSTALL=$HADOOP\_HOME
      3. export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME
      4. export HADOOP\_COMMON\_HOME=$HADOOP\_HOME
      5. export HADOOP\_HDFS\_HOME=$HADOOP\_HOME
      6. export YARN\_HOME=$HADOOP\_HOME
      7. export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native
      8. export PATH=$PATH:$HADOOP\_HOME/sbin:$HADOOP\_HOME/bin
      9. export HADOOP\_OPTS"-Djava.library.path=$HADOOP\_HOME/lib/nativ"
   3. source ~/.bashrc
9. Edit JAVA\_HOME
   1. sudo nano $HADOOP\_HOME/etc/hadoop/hadoop-env.sh
   2. Add the following line at the end of the file
      1. export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64
10. Edit core-site
    1. sudo nano $HADOOP\_HOME/etc/hadoop/core-site.xml
    2. Add the following lines
       1. <property>
       2. <name>hadoop.tmp.dir</name>
       3. <value>/home/hdoop/tmpdata</value>
       4. <description>A base for other temporary directories.</description>
       5. </property>
       6. <property>
       7. <name>fs.default.name</name>
       8. <value>hdfs://localhost:9000</value>
       9. <description>The name of the default file system></description>
       10. </property>
11. Edit hdfs-site
    1. sudo nano $HADOOP\_HOME/etc/hadoop/hdfs-site.xml
    2. Add the following lines
       1. <property>
       2. <name>dfs.data.dir</name>
       3. <value>/home/hdoop/dfsdata/namenode</value>
       4. </property>
       5. <property>
       6. <name>dfs.data.dir</name>
       7. <value>/home/hdoop/dfsdata/datanode</value>
       8. </property>
       9. <property>
       10. <name>dfs.replication</name>
       11. <value>1</value>
       12. </property>
12. Edit mapred-site
    1. sudo nano $HADOOP\_HOME/etc/hadoop/mapred-site.xml
    2. Add the following lines
       1. <property>
       2. <name>mapreduce.framework.name</name>
       3. <value>yarn</value>
       4. </property>
13. Edit yarn-site
    1. sudo nano $HADOOP\_HOME/etc/hadoop/yarn-site.xml
    2. Add the following lines
       1. <property>
       2. <name>yarn.nodemanager.aux-services</name>
       3. <value>mapreduce\_shuffle</value>
       4. </property>
       5. <property>
       6. <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
       7. <value>org.apache.hadoop.mapred.ShuffleHandler</value>
       8. </property>
       9. <property>
       10. <name>yarn.resourcemanager.hostname</name>
       11. <value>127.0.0.1</value>
       12. </property>
       13. <property>
       14. <name>yarn.acl.enable</name>
       15. <value>0</value>
       16. </property>
       17. <property>
       18. <name>yarn.nodemanager.env-whitelist</name>
       19. <value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_CONF\_DIR,CLAS
       20. SPATH\_PERPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_MAPRED\_HOME</value>
       21. </property>
14. Launch Hadoop
    1. hdfs namenode -format
    2. cd $HADOOP\_HOME/sbin
    3. start-all.sh
15. Go to browser
    1. localhost:8088
    2. localhost:9870

* Install R on windows

Download and install R for windows from - <https://cran.r-project.org/bin/windows/base/>

**OUTPUT**

****

**RESULT**

Thus, the installation, configuration of Hadoop and R has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 2** | **Implement word count / frequency programs using MapReduce** |
| **Date:** |

**AIM**

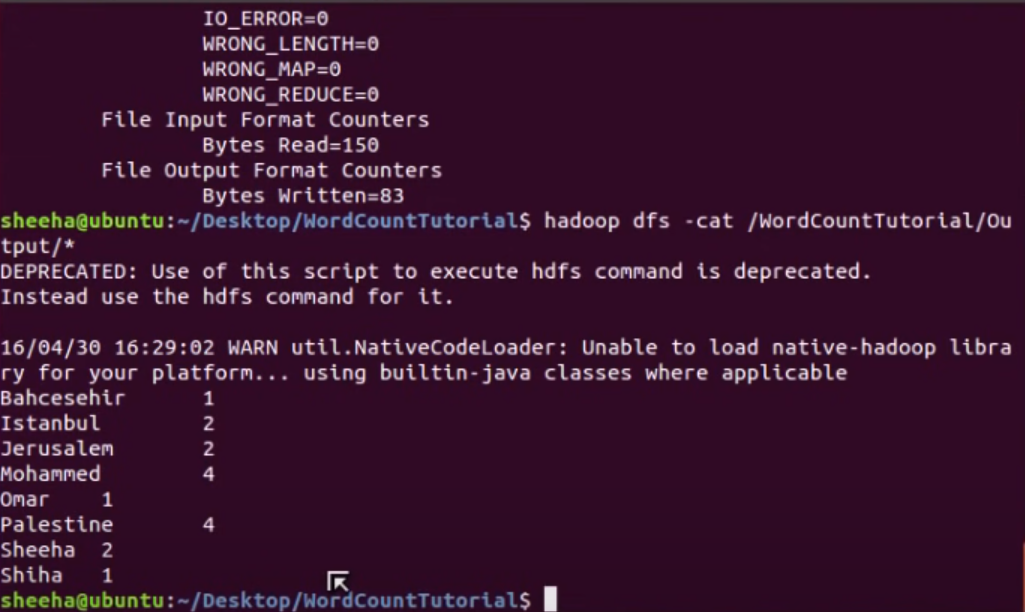
To implement word count program using MapReduce.

**PROCEDURE**

Run the following command on ubuntu terminal.

1. Create a directory on the Desktop named Lab and inside it create two folders; one called “Input” and the other called “tutorial\_classes”.
   1. cd Desktop
   2. mkdir Lab
   3. mkdir Lab/Input
   4. mkdir Lab/tutorial\_classes
2. Add the file attached with this document “WordCount.java” in the directory Lab
3. Add the file attached with this document “input.txt” in the directory Lab/Input.
4. Type the following command to export the hadoop classpath into bash.
   1. export HADOOP\_CLASSPATH=$(hadoop classpath)
5. Make sure it is now exported.
   1. echo $HADOOP\_CLASSPATH
6. It is time to create these directories on HDFS rather than locally. Type the following commands.
   1. hadoop fs -mkdir /WordCountTutorial
   2. hadoop fs -mkdir /WordCountTutorial/Input
   3. hadoop fs -put Lab/Input/input.txt /WordCountTutorial/Input
7. Go to localhost:9870 from the browser, Open “Utilities → Browse File System” and you should see the directories and files we placed in the file system.
8. Then, back to local machine where we will compile the WordCount.java file. Assuming we are currently in the Desktop directory.
   1. cd Lab
   2. javac -classpath $HADOOP\_CLASSPATH -d tutorial\_classes WordCount.javaPut the output files in one jar file (There is a dot at the end)
   3. jar -cvf WordCount.jar -C tutorial\_classes .
9. Now, we run the jar file on Hadoop.
   1. hadoop jar WordCount.jar WordCount /WordCountTutorial/Input /WordCountTutorial/Output
10. Output the result:
    1. hadoop dfs -cat /WordCountTutorial/Output/\*

**OUTPUT**



**RESULT**

Thus, the implementation of word count using MapReduce has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 3** | **Implement an MR program that processes a Weather Dataset** |
| **Date:** |

**AIM**

To implement an MR program that processes a weather dataset.

**PROCEDURE**

Run the following commands on ubuntu terminal.

1. Download dataset from - <ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01>
2. Create a java class as MyMaxMin in eclipse IDE

**MyMaxMin.java**

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.conf.Configuration;

public class MyMaxMin {

public static class MaxTemperatureMapper extends

Mapper<LongWritable, Text, Text, Text> {

public static final int MISSING = 9999;

@Override

public void map(LongWritable arg0, Text Value, Context context)

throws IOException, InterruptedException {

String line = Value.toString();

if (!(line.length() == 0)) {

String date = line.substring(6, 14);

float temp\_Max = Float.parseFloat(line.substring(39, 45).trim());

float temp\_Min = Float.parseFloat(line.substring(47, 53).trim());

if (temp\_Max > 30.0) {

context.write(new Text("The Day is Hot Day :" + date),

new Text(String.valueOf(temp\_Max)));

}

if (temp\_Min < 15) {

context.write(new Text("The Day is Cold Day :" + date),

new Text(String.valueOf(temp\_Min)));

}

}

}

}

public static class MaxTemperatureReducer extends

Reducer<Text, Text, Text, Text> {

public void reduce(Text Key, Iterator<Text> Values, Context context)

throws IOException, InterruptedException {

String temperature = Values.next().toString();

context.write(Key, new Text(temperature));

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = new Job(conf, "weather example");

job.setJarByClass(MyMaxMin.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setMapperClass(MaxTemperatureMapper.class);

job.setReducerClass(MaxTemperatureReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

Path OutputPath = new Path(args[1]);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

OutputPath.getFileSystem(conf).delete(OutputPath);

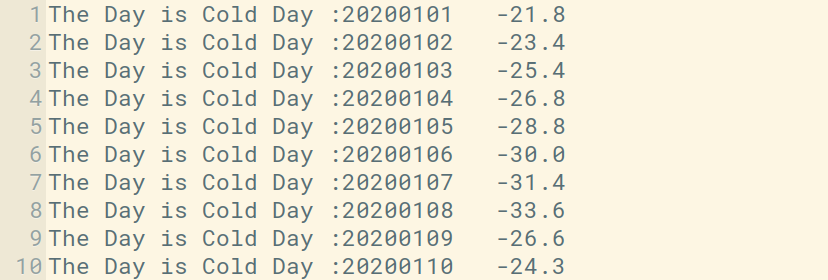
System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

1. Now we add these external jars to our MyProject. Right Click on MyProject -> then select Build Path-> Click on Configure Build Path and select Add External jars…. and add jars from it’s download location then click -> Apply and Close.
2. Now export the project as jar file. Right-click on MyProject choose Export.. and go to Java -> JAR file click -> Next and choose your export destination then click -> Next.
3. Choose Main Class as MyMaxMin by clicking -> Browse and then click -> Finish -> Ok.
4. Start Hadoop
   1. start-all.sh
5. Move dataset to Hadoop HDFS
   1. hdfs dfs -put /file\_path /destination
   2. hdfs dfs -put /home/hadoop/Downloads/CRND0103-2020-AK\_Fairbanks\_11\_NE.txt /
   3. hdfs dfs -ls /
6. Now Run your Jar File with below command and produce the output in MyOutput File.
   1. hadoop jar /home/hadoop/Documents/Project.jar /CRND0103-2020-AK\_Fairbanks\_11\_NE.txt /MyOutput
7. Go to browser – localhost:9870

**OUTPUT**



**RESULT**

Thus, the implementation of MR program that processes a weather dataset has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 4 a** | **R – Programming Introduction and Basics** |
| **Date:** |

**AIM**

To learn and execute R – Programming basics.

**PROCEDURE**

* WHAT IS R?

R is a programming language developed by Ross Ihaka and Robert Gentleman in 1993. R possesses an extensive catalogue of statistical and graphical methods. It includes machine learning algorithm, linear regression, time series, statistical inference to name a few. Most of the R libraries are written in R, but for heavy computational task, C, C++ and Fortran codes are preferred. R is not only entrusted by academic, but many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on.

Data analysis with R is done in a series of steps; programming, transforming, discovering, modelling and communicate the results

* + Program: R is a clear and accessible programming tool
  + Transform: R is made up of a collection of libraries designed specifically for data science
  + Discover: Investigate the data, refine your hypothesis and analyze them
  + Model: R provides a wide array of tools to capture the right model for your data
  + Communicate: Integrate codes, graphs, and outputs to a report with R Markdown or build
  + Shiny apps to share with the world What is R used for?
  + Statistical inference
  + Data analysis
  + Machine learning algorithm
* R DATA TYPES & OPERATOR

Basic data types

* R works with numerous data types, including
* Scalars
* Vectors (numerical, character, logical)
* Matrices
* Data frames
* Lists

Basics types

* 4.5 is a decimal value called numerics.
* 4 is a natural value called integers. Integers are also numerics.
* TRUE or FALSE is a Boolean value called logical.
* The value inside " " or ' ' are text (string). They are called characters.
* We can check the type of a variable with the class function

|  |  |  |
| --- | --- | --- |
| **Data Types** | **R Code** | **Output** |
| # Numeric | x <‐ 28 class(x) | ## [1] "numeric" |
| # String | y <‐ "R is Fantastic" class(y) | ## [1] "character" |
| # Boolean | z <‐ TRUE  class(z) | ## [1] "logical" |

* Variables

Variables store values and are an important component in programming, especially for a data scientist. A variable can store a number, an object, a statistical result, vector, dataset, a model prediction basically anything R outputs. We can use that variable later simply by calling the name of the variable. To declare a variable, we need to assign a variable name. The name should not have space. We can use \_ to connect to words. To add a value to the variable, use <- or =.

* SYNTAX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SYNTAX** | **RCODE** | **OUTPUT** | | |
| # First way to declare a variable: use the `<‐` name\_of\_variable <‐ value  # Second way to declare a variable: use the  `=`  name\_of\_variable = value | # Print variable x x <‐ 42  x | ## | [1] | 42 |
| y <‐ 10 y | ## | [1] | 10 |
| # We call x and y and apply a subtraction  x‐y | ## | [1] | 32 |

* Vectors

A vector is a one-dimensional array. We can create a vector with all the basic data type we learnt before. The simplest way to build a vector in R, is to use the c command.

|  |  |  |
| --- | --- | --- |
| **Vectors** | **Type – Rcode** | **OUTPUT** |
| # Numerical | vec\_num <‐ c(1, 10, 49) vec\_num | ## [1] 1 10 49 |
| # Character | vec\_chr <‐ c("a", "b", "c") vec\_chr | ## [1] "a" "b" "c" |
| # Boolean | vec\_bool <‐ c(TRUE, FALSE, TRUE) vec\_bool | ##[1] TRUE FALSE TRUE |
| # Create the vectors | vect\_1 <‐ c(1, 3, 5)  vect\_2 <‐ c(2, 4, 6)  # Take the sum of A\_vector and B\_vector  sum\_vect <‐ vect\_1 + vect\_2 # Print out total\_vector sum\_vect | [1] 3 7 11 |
| # Slice the  first 5 rows of the vector | slice\_vector <‐ c(1,2,3,4,5,6,7,8,9,10)  slice\_vector[1:5] | ## [1] 1 2 3 4 5 |
| # Faster way  to create adjacent values | c(1:10) | ##  [1] 1 2 3 4 5 6 7 8 9 10 |

|  |  |  |
| --- | --- | --- |
| **Operator Description** | **RCode** | **OUTPUT** |
| **Arithmetic** |  |  |
| + Addition | 3 + 4 | ## [1] 7 |
| - Subtraction | 4-2 | ## [1] 2 |
| \* Multiplication | 3\*5 | ## [1] 15 |
| / Division  ^ or \*\* Exponentiation | (5+5)/2  2^5 | ## [1] 5  ## [1] 32 |
| **Logical** | logical\_vector <‐ c(1:10) logical\_vector>5  logical\_vector[(logical\_vector>5)] | ## [1]FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE  ## [1] 6 7 8 9 10 |
| # Print 5 and 6 logical\_vector <‐ c(1:10)  logical\_vector[(logical\_vector>4) & (logical\_vector<7)] | ## [1] 5 6 |

* R MATRIX TUTORIAL: CREATE, PRINT, ADD COLUMN, SLICE

What is a Matrix?

A matrix is a 2-dimensional array that has m number of rows and n number of columns. In other words, matrix is a combination of two or more vectors with the same data type. Note: It is possible to create more than two dimensions arrays with R.

SYNTAX: matrix(data, nrow, ncol, byrow = FALSE) Arguments:

* + ‐ data: The collection of elements that R will arrange into the rows and columns of the matrix
  + ‐ nrow: Number of rows
  + ‐ ncol: Number of columns
  + ‐ byrow: The rows are filled from the left to the right. We use `byrow = FALSE` (default values), if we want the matrix to be filled by the columns i.e. the values are filled top to bottom.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # Construct a matrix with 5 rows that contain the numbers 1 up to 10 and byrow  = TRUE | matrix\_a <‐matrix(1:10, byrow = TRUE, nrow = 5)  matrix\_a |  | | | | |
| # Print dimension of the matrix | dim(matrix\_a) | ## | [1] | 5 2 |  |  |
| # Construct a matrix | matrix\_b <‐matrix(1:10, byrow = |  | | | | |
| with 5 rows that | FALSE, nrow = 5) |
| contain the numbers | matrix\_b |
| 1 up to 10 and byrow |  |
| = FALSE |  |
| matrix\_c <‐matrix(1:12, byrow = | ## |  | [,1] | [,2] | [,3] |
|  | FALSE, ncol = 3) | ## | [1,] | 1 | 5 | 9 |
|  | matrix\_c | ## | [2,] | 2 | 6 | 10 |
|  |  | ## | [3,] | 3 | 7 | 11 |
|  |  | ## | [4,] | 4 | 8 | 12 |
| # concatenate c(1:5) to the matrix\_a | matrix\_a1 <‐ cbind(matrix\_a, c(1:5)) # Check the dimension dim(matrix\_a1) | ## | [1] | 5 3 |  |  |

* Slice a Matrix

We can select elements one or many elements from a matrix by using the square brackets [ ]. This is where slicing comes into the picture.

For example:

* + matrix\_c[1,2] selects the element at the first row and second column.
  + matrix\_c[1:3,2:3] results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3.
  + matrix\_c[,1] selects all elements of the first column.
  + matrix\_c[1,] selects all elements of the first row.
* WHAT IS FACTOR IN R? CATEGORICAL & CONTINUOUS

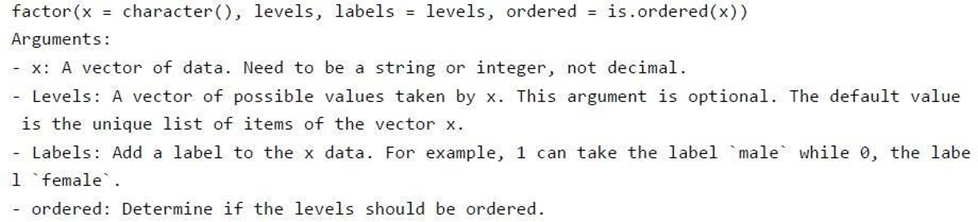
What is Factor in R?

Factors are variables in R which take on a limited number of different values; such variables are often referred to as categorical variables. In a dataset, we can distinguish two types of variables: categorical and continuous. In a categorical variable, the value is limited and usually based on a particular finite group. For example, a categorical variable can be countries, year, gender, occupation. A continuous variable, however, can take any values, from integer to decimal. For example, we can have the revenue, price of a share, etc..

* Categorical variables

R stores categorical variables into a factor. Let's check the code below to convert a character variable into a factor variable. Characters are not supported in machine learning algorithm, and the only way is to convert a string to an integer.

* SYNTAX



* RCODE

# Create gender vector

gender\_vector <‐ c("Male", "Female", "Female", "Male", "Male") class(gender\_vector)

# Convert gender\_vector to a factor

factor\_gender\_vector <‐factor(gender\_vector) class(factor\_gender\_vector)

* OUTPUT:

## [1] "character"

## [1] "factor"

* Nominal categorical variable

A categorical variable has several values but the order does not matter. For instance, male or female categorical variable do not have ordering.

* RCODE:

# Create a color vector

color\_vector <‐ c('blue', 'red', 'green', 'white', 'black', 'yellow') # Convert the vector to factor

factor\_color <‐ factor(color\_vector) factor\_color

* OUTPUT:

## [1] blue red green white black yellow

## Levels: black blue green red white yellow

* Ordinal categorical variable

Ordinal categorical variables do have a natural ordering. We can specify the order, from the lowest to the highest with order = TRUE and highest to lowest with order = FALSE. We can use summary to count the values for each factor.

* RCODE:

# Create Ordinal categorical vector

day\_vector <‐ c('evening', 'morning', 'afternoon', 'midday', 'midnight', 'evening') # Convert `day\_vector` to a factor with ordered level

factor\_day <‐ factor(day\_vector, order = TRUE, levels =c('morning', 'midday', 'afternoon', 'evening', 'midnight'))

# Print the new variable factor\_day

* OUTPUT:

## [1] evening morning afternoon midday midnight evening

## Levels: morning < midday < afternoon < evening < midnight # Append the line to above code

# Count the number of occurence of each level summary(factor\_day)

## Morning midday afternoon evening midnight ## 1 1 1 2 1

* Continuous variables

Continuous class variables are the default value in R. They are stored as numeric or integer. We can see it from the dataset below. mtcars is a built-in dataset. It gathers information on different types of car. We can import it by using mtcars and check the class of the variable mpg, mile per gallon. It returns a numeric value, indicating a continuous variable.

* RCODE:

dataset <‐ mtcars class(dataset)

* OUTPUT:

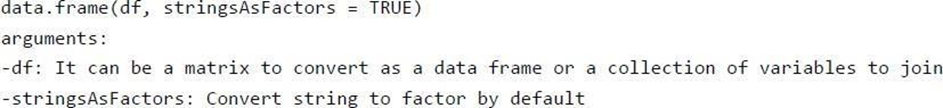
## [1] "numeric"

* R DATA FRAMES: CREATE, APPEND, SELECT, SUBSET

What is a Data Frame?

A data frame is a list of vectors which are of equal length. A matrix contains only one type of data, while a data frame accepts different data types (numeric, character, factor, etc.).

* Syntax



* Create a data frame RCODE:

# Create a, b, c, d variables a <‐ c(10,20,30,40)

b <‐ c('book', 'pen', 'textbook', 'pencil\_case') c <‐ c(TRUE,FALSE,TRUE,FALSE)

d <‐ c(2.5, 8, 10, 7)

# Join the variables to create a data frame df <‐ data.frame(a,b,c,d)

df

* OUTPUT:

## a b c d

## 1 1 book TRUE 2.5

## 2 2 pen TRUE 8.0

## 3 3 textbook TRUE 10.0

## 4 4 pencil\_case FALSE 7.0

# Name the data frame

names(df) <‐ c('ID', 'items', 'store', 'price') df

* OUTPUT:

## ID items store price

## 1 10 book TRUE 2.5

## 2 20 pen FALSE 8.0

## 3 30 textbook TRUE 10.0

## 4 40 pencil\_case FALSE 7.0

# Print the structure str(df)

* OUTPUT:

## 'data.frame': 4 obs. of 4 variables:

## $ ID : num 10 20 30 40

## $ items: Factor w/ 4 levels "book","pen","pencil\_case",..: 1 2 4 3 ## $ store: logi TRUE FALSE TRUE FALSE

## $ price: num 2.5 8 10 7

* Slice Data Frame

It is possible to SLICE values of a Data Frame. We select the rows and columns to return into bracket precede by the name of the data frame.

RCODE:

## Select Rows 1 to 3 and columns 3 to 4

df[1:3, 3:4]

* OUTPUT:

## store price ## 1 TRUE 2.5

## 2 FALSE 8.0

## 3 TRUE 10.0

* Append a Column to Data Frame

You can also append a column to a Data Frame. You need to use the symbol $ to append a new Variable.

* RCODE:

# Create a new vector quantity <‐ c(10, 35, 40, 5)

# Add `quantity` to the `df` data frame df$quantity <‐ quantity

df

* OUTPUT:

## ID items store price quantity ## 1 10 book TRUE 2.5 10

## 2 20 pen FALSE 8.0 35

## 3 30 textbook TRUE 10.0 40

## 4 40 pencil\_case FALSE 7.0 5

Note: The number of elements in the vector has to be equal to the no of elements in data frame. Executing the following statement

* RCODE

quantity <‐ c(10, 35, 40)

# Add `quantity` to the `df` data frame df$quantity <‐ quantity

* Select a column of a data frame

Sometimes, we need to store a column of a data frame for future use or perform operation on a column. We can use the $ sign to select the column from a data frame.

* RCODE:

# Select the column ID df$ID

* OUTPUT:

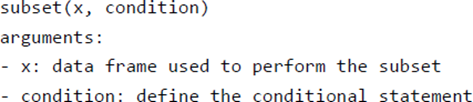
## [1] 1 2 3 4

* Subset a data frame

In the previous section, we selected an entire column without condition. It is possible to subset based on whether or not a certain condition was true.

We use the subset() function.

* SYNTAX



We want to return only the items with price above 10, we can do

* RCODE:

# Select price above 5 subset(df, subset = price >

* OUTPUT:

ID items store price

20 pen FALSE 8

30 textbook TRUE 10

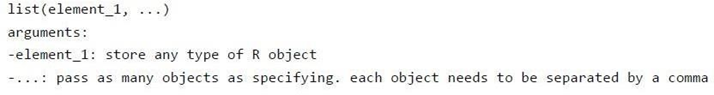
40 pencil\_case FALSE 7

* LISTS IN R: CREATE, SELECT [EXAMPLE]
* What is a List?

A list is a great tool to store many kinds of object in the order expected. We can include matrices, vectors data frames or lists. We can imagine a list as a bag in which we want to put many different items. When we need to use an item, we open the bag and use it. A list is similar; we can store a collection of objects and use them when we need them.

We can use list() function to create a list.

* SYNTAX



* RCODE:

# Vector with numeric from 1 up to 5 vect <‐ 1:5

# A 2x 5 matrix

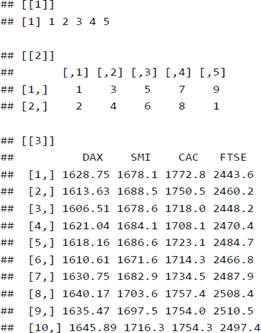
mat <‐ matrix(1:9, ncol = 5) dim(mat)

# select the 10th row of the built‐in R data set EuStockMarkets df <‐ EuStockMarkets[1:10,]

# Construct list with these vec, mat, and df: my\_list <‐ list(vect, mat, df)

my\_list

* OUTPUT



* IF, ELSE, ELIF STATEMENT IN R

The if, else, ELIF statement

An if-else statement is a great tool for the developer trying to return an output based on a condition.

* SYNTAX:

if (condition1) { expr1

} else it (condition2) { expr2

} else if (condition3) { expr3

} else { expr4

}

VAT has different rate according to the product purchased. Imagine we have three different kind of products with different VAT applied:

Categories Products VAT

A Book, magazine, newspaper, etc.. 8%

B Vegetable, meat, beverage, etc.. 10%

C Tee-shirt, jean, pant, etc.. 20%

We can write a chain to apply the correct VAT rate to the product a customer bought.

* RCODE:

category <‐ 'A' price <‐ 10

if (category =='A'){

cat('A vat rate of 8% is applied.','The total price is',price \*1.08)

} else if (category =='B'){

cat('A vat rate of 10% is applied.','The total price is',price \*1.10)

} else {

cat('A vat rate of 20% is applied.','The total price is',price \*1.20)

}

* OUTPUT:

# A vat rate of 8% is applied. The total price is 10.8

* FOR LOOP SYNTAX AND EXAMPLES

# Create fruit vector

fruit <‐ c('Apple', 'Orange', 'Passion fruit', 'Banana') # Create the for statement

for ( i in fruit){ print(i)

}

* OUTPUT:

## [1] "Apple"

## [1] "Orange"

## [1] "Passion fruit"

## [1] "Banana"

* For Loop over a matrix

A matrix has 2-dimension, rows and columns. To iterate over a matrix, we have to define two for loop, namely one for the rows and another for the column.

* SYNTAX:

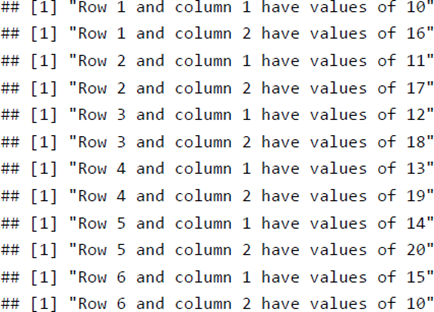
# Create a matrix

mat <‐ matrix(data = seq(10, 20, by=1), nrow = 6, ncol =2) # Create the loop with r and c to iterate over the matrix

for (r in 1:nrow(mat)) for (c in 1:ncol(mat))

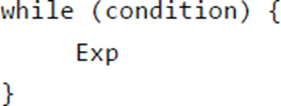
print(paste("Row", r, "and column",c, "have values of", mat[r,c]))

* OUTPUT



* WHILE LOOP IN R WITH EXAMPLE

A loop is a statement that keeps running until a condition is satisfied. The syntax for a while loop is the following:



* SYNTAX:

#Create a variable with value 1 begin <‐ 1

#Create the loop while (begin <= 10){ #See which we are

cat('This is loop number',begin)

#add 1 to the variable begin after each loop begin <‐ begin+1

print(begin)

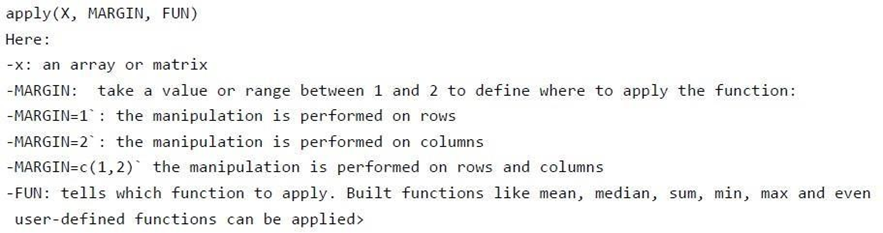
}

* APPLY(), SAPPLY(), TAPPLY() IN R WITH EXAMPLES

apply() function

We use apply() over a matrice. This function takes 5 arguments:

* SYNTAX:



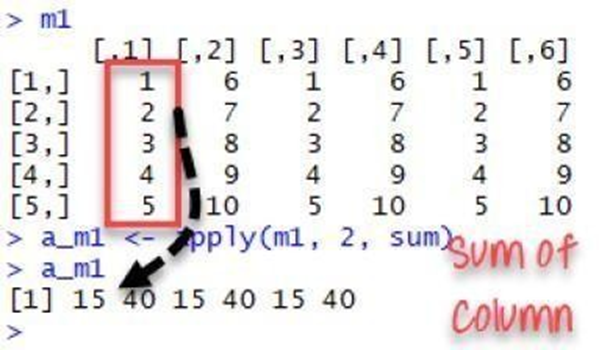
The simplest example is to sum a matrices over all the columns. The code apply(m1, 2, sum) will apply the sum function to the matrix 5x6 and return the sum of each column accessible in the dataset.

* RCODE:

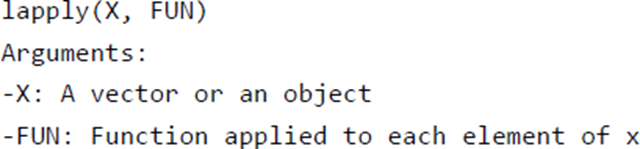
m1 <‐ matrix(C<‐(1:10),nrow=5, ncol=6) m1

a\_m1 <‐ apply(m1, 2, sum) a\_m1

* OUTPUT:



* lapply() function
* SYNTAX



l in lapply() stands for list. The difference between lapply() and apply() lies between the output return. The output of lapply() is a list. lapply() can be used for other objects like data frames and lists. lapply() function does not need MARGIN.

A very easy example can be to change the string value of a matrix to lower case with to lower function. We construct a matrix with the name of the famous movies. The name is in upper case format.

* RCODE:

movies <‐ c("SPYDERMAN","BATMAN","VERTIGO","CHINATOWN")

movies\_lower <‐lapply(movies, tolower) str(movies\_lower)

We can use unlist() to convert the list into a vector. movies\_lower <‐unlist(lapply(movies,tolower)) str(movies\_lower)

* OUTPUT:

## List of 4

## $:chr"spyderman"

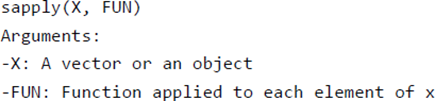
## $:chr"batman"

## $:chr"vertigo"

## $:chr"chinatown"

## chr [1:4] "spyderman" "batman" "vertigo" "chinatown"

* sapply() function
* SYNTAX



* RCODE:

dt <‐ cars

lmn\_cars <‐ lapply(dt, min) smn\_cars <‐ sapply(dt, min) lmn\_cars

smn\_cars

lmxcars <‐ lapply(dt, max) smxcars <‐ sapply(dt, max) lmxcars

smxcars

We can use a user built-in function into lapply() or sapply(). We create a function named avg to compute the average of the minimum and maximum of the Vector.

avg <‐ function(x) {

( min(x) + max(x) ) / 2} fcars <‐ sapply(dt, avg) fcars

* OUTPUT:

## $speed

## [1] 4

## $dist

## [1] 2

## speed dist

## 4 2

## $speed

## [1] 25

## $dist

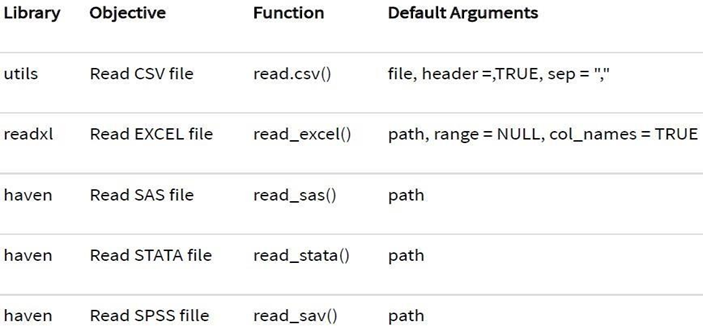
## [1] 120

## speed dist

## 25 120

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Arguments** | **Objective** | **Input** | **Output** |
| apply | apply(x, MARGIN, FUN) | Apply a function to the rows or columns or both | Data frame or matrix | vector, list, array |
| lapply | lapply(X, FUN) | Apply a function to all the  elements of the input | List, vector or data frame | list |
| sapply | sappy(X FUN) | Apply a function to all the  elements of the input | List, vector or data frame | vector or matrix |

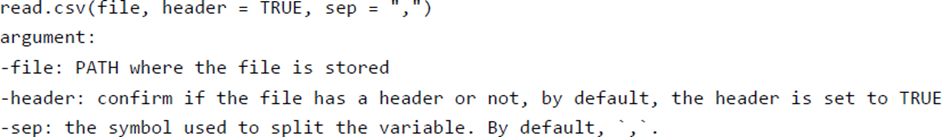
* IMPORT DATA INTO R: READ CSV, EXCEL, SPSS, STATA, SAS FILES



* Read CSV

One of the most widely data store is the .csv (comma-separated values) file formats. R loads an array of libraries during the start-up, including the utils package. This package is convenient to open csv files combined with the reading.csv () function.

* SYNTAX:



* RCODE:

PATH <‐

'https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/datasets/m tc

ars.csv'

df <‐ read.csv(PATH, header = TRUE, sep = ',', stringsAsFactors =FALSE) length(df)

class(df$X)

* OUTPUT:

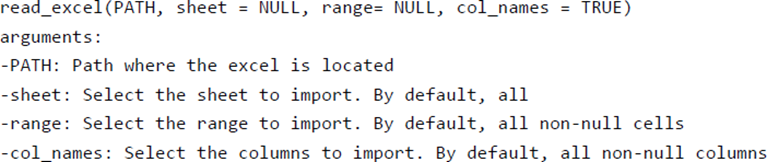
## [1] 12

## [1] "factor"

* Read Excel files

Excel files are very popular among data analysts. Spreadsheets are easy to work with and flexible. R is equipped with a library readxl to import Excel spreadsheet.

* SYNTAX



* RCODE:

require(readxl)

library(readxl)

readxl\_example()

readxl\_example("geometry.xls")

We can import the spreadsheets from the readxl library and count the number of columns in the first sheet. # Store the path of `datasets.xlsx`

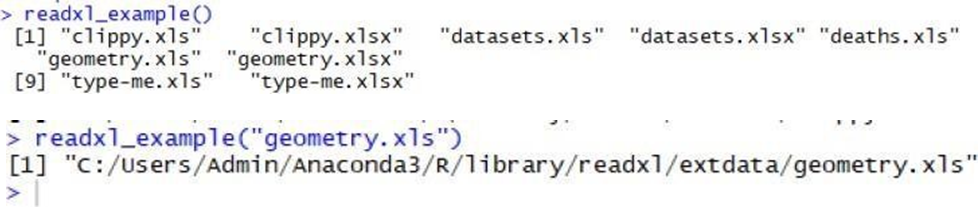
example <‐ readxl\_example("datasets.xlsx") #

Import the spreadsheet

df <‐ read\_excel(example)

# Count the number of columns length(df)

* OUTPUT



* Read excel\_sheets()

The file datasets.xlsx is composed of 4 sheets. We can find out which sheets are available in the workbook by using excel\_sheets() function

* RCODE:

example <‐ readxl\_example("datasets.xlsx")

excel\_sheets(example)

If a worksheet includes many sheets, it is easy to select a particular sheet by using the sheet arguments. We can specify the name of the sheet or the sheet index. We can verify if both function returns the same output with identical().

example <‐ readxl\_example("datasets.xlsx") quake <‐ read\_excel(example, sheet = "quakes") quake\_1 <‐read\_excel(example, sheet = 4) identical(quake, quake\_1)

* OUTPUT:

[1] "iris" "mtcars" "chickwts" "quakes"

## [1] TRUE

* R EXPORTING DATA TO CSV, EXCEL, SAS, STATA, AND TEXT FILE

Export to Hard drive

To begin with, you can save the data directly into the working directory. The following code prints the path of your working directory:

* RCODE:

directory <‐getwd() directory OUTPUT:

## [1] "/Users/15\_Export\_to\_do"

Create data frame

First of all, let's import the mtcars dataset and get the mean of mpg and disp grouped by

Gear

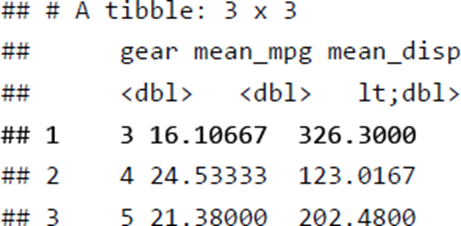
* RCODE:

library(dplyr)

df <‐mtcars % > % select(mpg, disp, gear) % > % group\_by(gear) % > %

summarize(mean\_mpg = mean(mpg), mean\_disp = mean(disp)) df

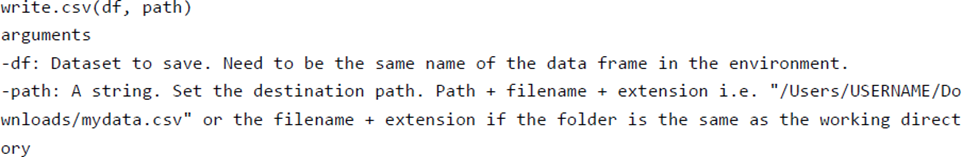
* OUTPUT



The table contains three rows and three columns. You can create a CSV file with the function write.csv().

Export CSV

* SYNTAX



* RCODE:

write.csv(df, "table\_car.csv")

Code Explanation

write.csv(df, "table\_car.csv"): Create a CSV file in the hard drive: df: name of the data frame in the environment

"table\_car.csv": Name the file table\_car and store it as csv

Note: You can use the function write.csv2() to separate the rows with a semicolon.

write.csv2(df, "table\_car.csv")

* Export to Excel file

Export data to Excel is trivial for Windows users and trickier for Mac OS user. Both users will use the library xlsx to create an Excel file. The slight difference comes from the installation of the library. Indeed, the library xlsx uses Java to create the file. Java needs to be installed if not present in your machine.

* Windows users

If you are a Windows user, you can install the library directly with conda:

conda install ‐c r r‐xlsx library(xlsx)

write.xlsx(df, "table\_car.xlsx") library(haven)

write\_sav(df, "table\_car.sav" ## spss file write\_sas(df, "table\_car.sas7bdat") write\_dta(df, "table\_car.dta") ## STATA File save(df, file ='table\_car.RData')

* R SELECT(), FILTER(), ARRANGE()

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Objective | Code | Explanation |
| glimpse | check the structure of a df | glimpse(df) | Identical to str() |
| select() | Select/exclude the variables | select(df, A, B,C) select(df, A:C) select(df, ‐C) | Select the variables A, B and C Select all variables from A to C Exclude C |
| arrange() | Sort the dataset with one or many variables | arrange(A) arrange(desc (A), B) | Descending sort of variable A and ascending sort of B |

**RESULT**

Thus, basics programs of R – programming has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 4 b** | **Implement Linear and Logistic Regression** |
| **Date:** |

**AIM**

To implement linear and logistic regression.

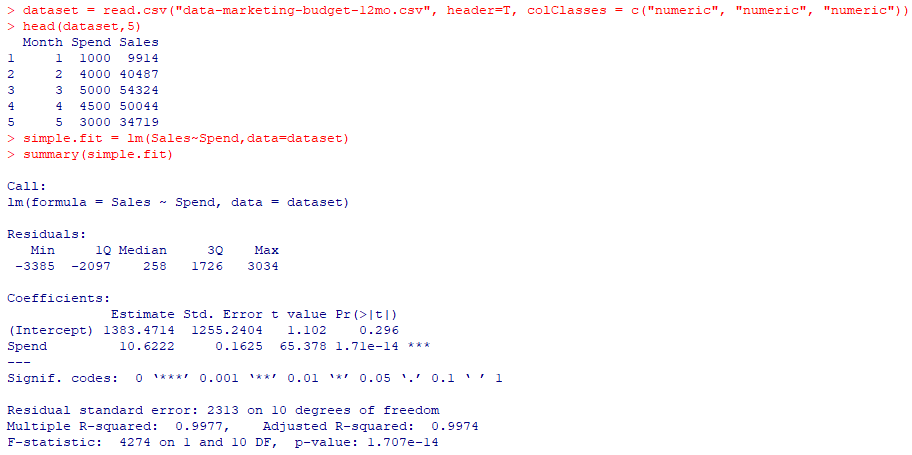
**PROCEDURE**

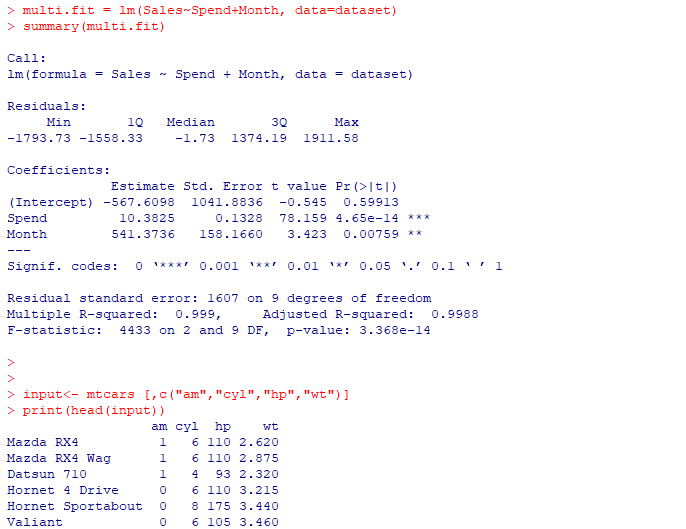
1. Open R on windows.
2. Create a new workspace.
3. Create a new script file.
4. Type the code in the script file.
5. Run the script file.
6. Close R.

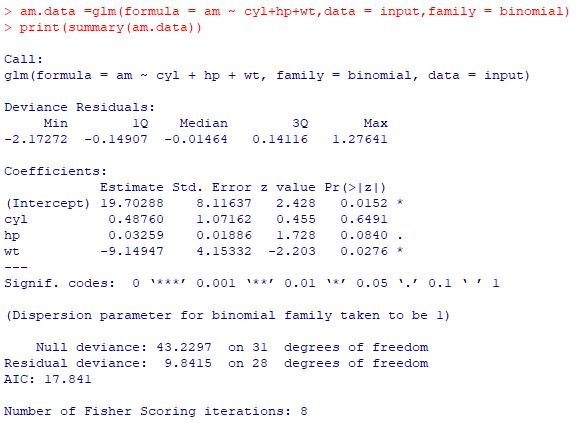
**PROGRAM**

* dataset = read.csv("data-marketing-budget-12mo.csv", header=T, colClasses = c("numeric", "numeric", "numeric"))
* head(dataset,5)
* simple.fit = lm(Sales~Spend,data=dataset)
* summary(simple.fit)
* multi.fit = lm(Sales~Spend+Month, data=dataset)
* summary(multi.fit)
* input<- mtcars [,c("am","cyl","hp","wt")]
* print(head(input))
* am.data =glm(formula = am ~ cyl+hp+wt,data = input,family = binomial)
* print(summary(am.data))

**OUTPUT**







**RESULT**

Thus, the implementation of linear and logistic regression has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 5 a** | **Implement SVM Classification Techniques** |
| **Date:** |

**AIM**

To implement SVM Classification technique.

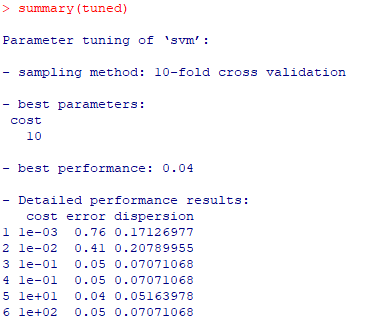
**PROCEDURE**

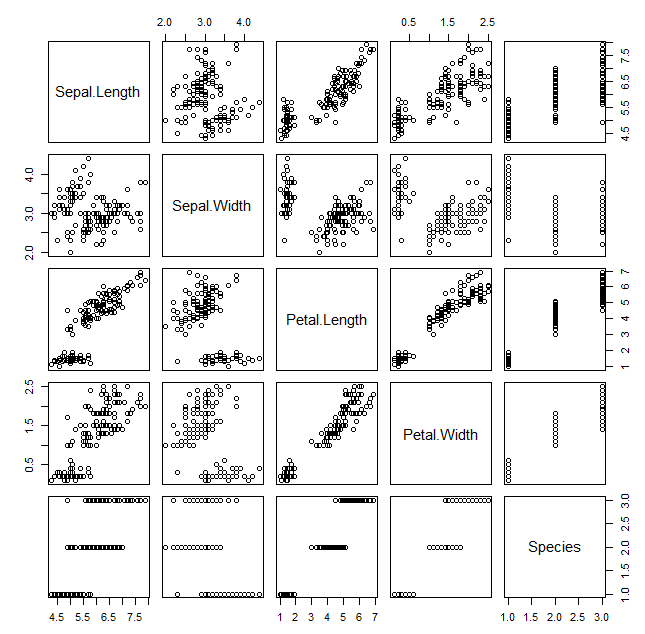
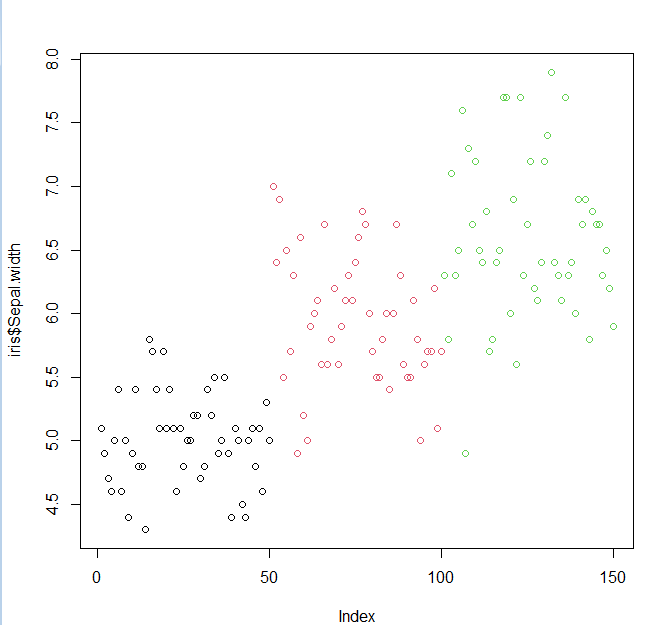
1. Open R on windows.
2. Create a new workspace.
3. Create a new script file.
4. Type the code in the script file.
5. Run the script file.
6. Close R.

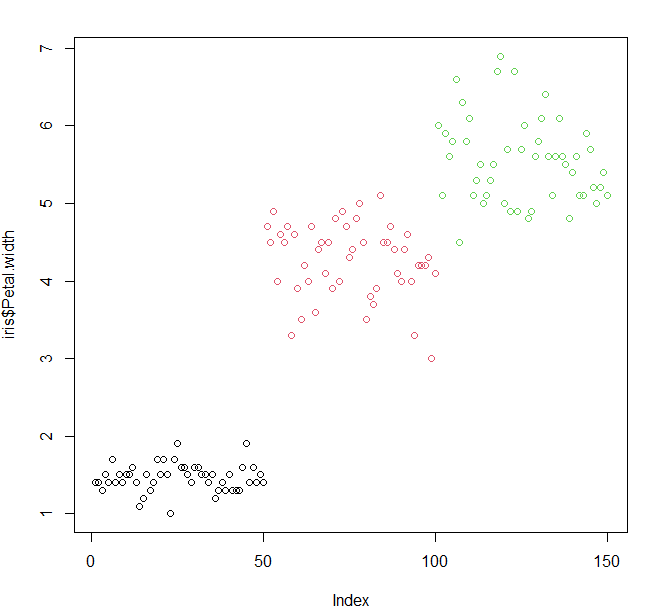
**PROGRAM**

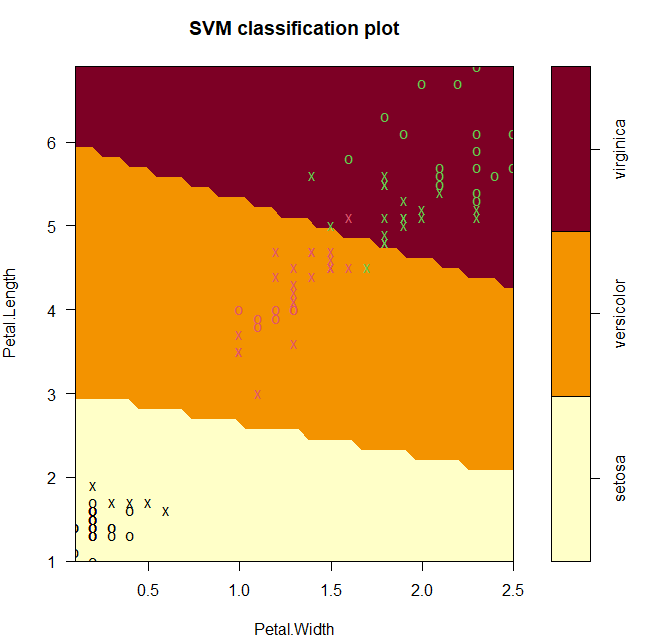
* library(e1071)
* plot(iris)
* plot(iris$Sepal.Length, iris$Sepal.width, col=iris$Species)
* plot(iris$Petal.Length, iris$Petal.width, col=iris$Species)
* s<-sample(150,100)
* col<- c("Petal.Length", "Petal.Width", "Species")
* iris\_train<- iris[s,col]
* iris\_test<- iris[-s,col]
* svmfit<- svm(Species ~., data = iris\_train, kernel = "linear", cost = .1, scale = FALSE)
* print(svmfit)
* plot(svmfit, iris\_train[,col])
* tuned <- tune(svm, Species~., data = iris\_train, kernel = "linear", ranges= list(cost=c(0.001,0.01,.1,.1,10,100)))
* summary(tuned)
* p<-predict(svmfit, iris\_test[,col], type="class")
* plot(p)
* table(p,iris\_test[,3] )
* mean(p== iris\_test[,3])

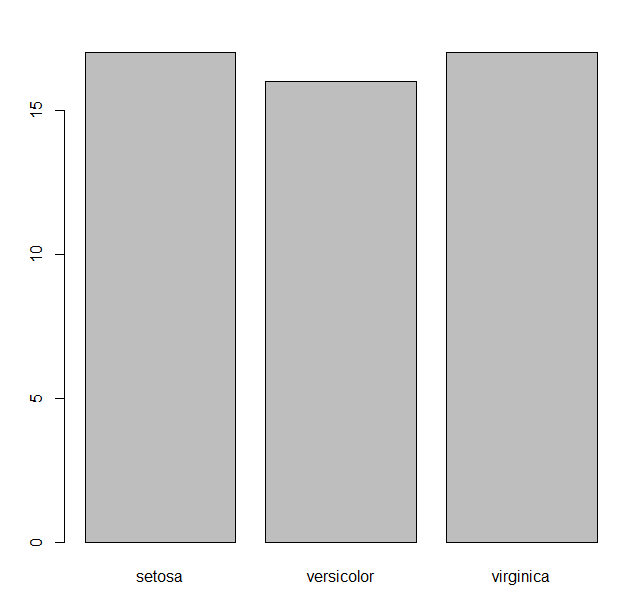
**OUTPUT**

****







**RESULT**

Thus, the implementation of SVM Classification technique has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 5 b** | **Implement Decision Tree Classification Techniques** |
| **Date:** |

**AIM**

To implement decision tree classification technique.

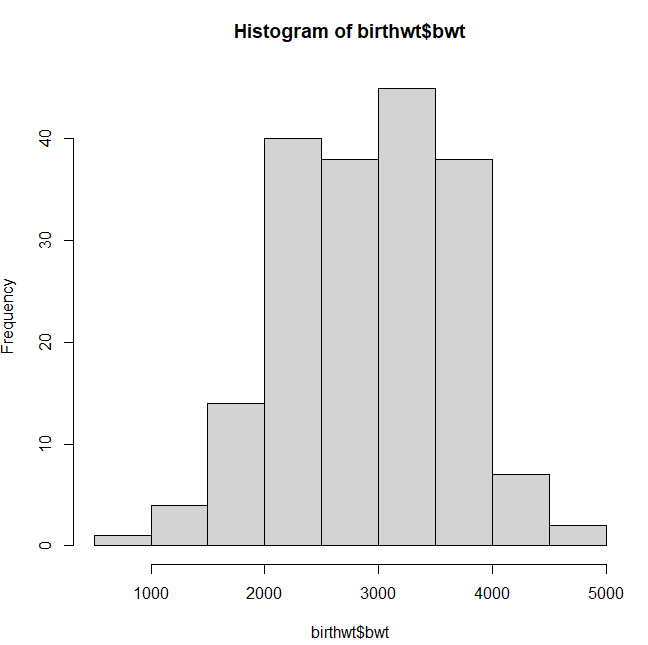
**PROCEDURE**

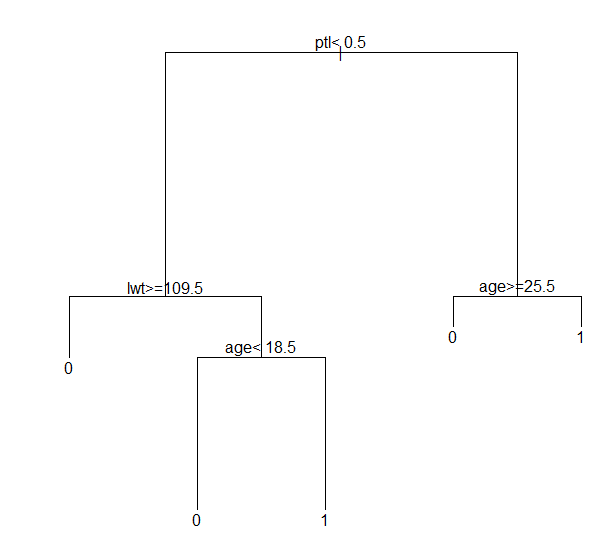
1. Open R on windows.
2. Create a new workspace.
3. Create a new script file.
4. Type the code in the script file.
5. Run the script file.
6. Close R.

**PROGRAM**

* library(MASS)
* library(rpart)
* head(birthwt)
* hist(birthwt$bwt)
* table(birthwt$low)
* cols <- c('low', 'race', 'smoke', 'ht', 'ui')
* birthwt[cols] <- lapply(birthwt[cols], as.factor)
* set.seed(1)
* train<- sample(1:nrow(birthwt), 0.75 \* nrow(birthwt))
* birthwtTree<- rpart(low ~ . - bwt, data = birthwt[train, ], method = 'class')
* plot(birthwtTree)
* text(birthwtTree, pretty = 0)
* summary(birthwtTree)
* birthwtPred<- predict(birthwtTree, birthwt[-train, ], type = 'class')
* table(birthwtPred, birthwt[-train, ]$low)

**OUTPUT**





**RESULT**

Thus, the implementation of decision tree classification technique has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 6** | **Implement Clustering Techniques** |
| **Date:** |

**AIM**

To implement clustering techniques.

**PROCEDURE**

1. Open R on windows.

2. Create a new workspace.

3. Create a new script file.

4. Type the code in the script file.

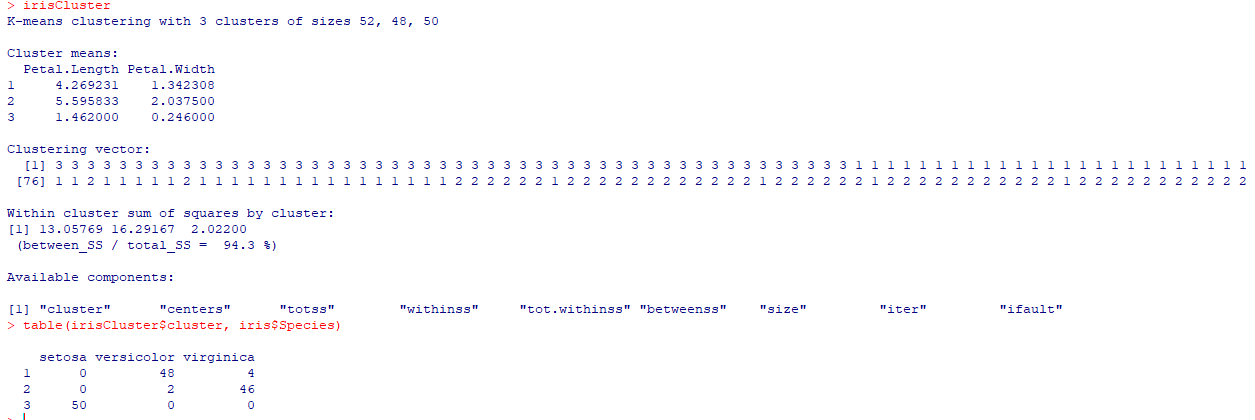
5. Run the script file.

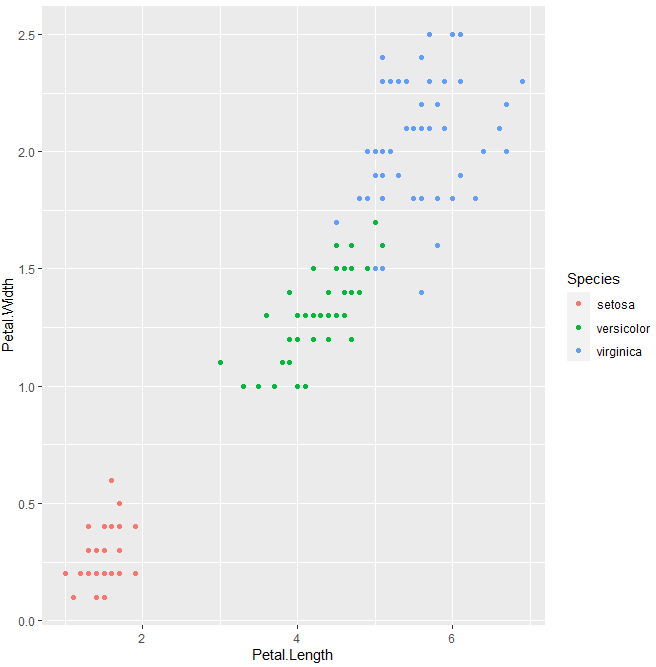
6. Close R.

**PROGRAM**

* library(datasets)
* head(iris)
* library(ggplot2)
* ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom\_point()
* set.seed(20)
* irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
* irisCluster
* table(irisCluster$cluster, iris$Species)

**OUTPUT**

****



**RESULT**

Thus, the implementation of clustering techniques has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 7** | **Visualize data using any plotting framework** |
| **Date:** |

**AIM**

To visualize data using any plotting framework in R.

**PROCEDURE**

1. Open R on windows.

2. Create a new workspace.

3. Create a new script file.

4. Type the code in the script file.

5. Run the script file.

6. Close R.

**PROGRAM**

1. Histogram

* library(RColorBrewer)
* data(VADeaths)
* par(mfrow=c(2,3))
* hist(VADeaths,breaks=10, col=brewer.pal(3,"Set3"),main="Set3 3 colors")
* hist(VADeaths,breaks=3 ,col=brewer.pal(3,"Set2"),main="Set2 3 colors")
* hist(VADeaths,breaks=7, col=brewer.pal(3,"Set1"),main="Set1 3 colors")
* hist(VADeaths,,breaks= 2, col=brewer.pal(8,"Set3"),main="Set3 8 colors")
* hist(VADeaths,col=brewer.pal(8,"Greys"),main="Greys 8 colors")
* hist(VADeaths,col=brewer.pal(8,"Greens"),main="Greens 8 colors")\

1. Line Chart

* data(AirPassengers)
* plot(AirPassengers,type="l")

1. Bar Chart

* data("iris")
* barplot(iris$Petal.Length)
* barplot(iris$Sepal.Length,col = brewer.pal(3,"Set1"))
* barplot(table(iris$Species,iris$Sepal.Length),col = brewer.pal(3,"Set1"))

1. Box Plot

* data(iris)
* par(mfrow=c(2,2))
* boxplot(iris$Sepal.Length,col="red")
* boxplot(iris$Sepal.Length~iris$Species,col="red")
* boxplot(iris$Sepal.Length~iris$Species,col=heat.colors(3))
* boxplot(iris$Sepal.Length~iris$Species,col=topo.colors(3))
* boxplot(iris$Petal.Length~iris$Species)

1. Scatter Plot

* plot(x=iris$Petal.Length)
* plot(x=iris$Petal.Length,y=iris$Species)

1. Heat Map

* x <- rnorm(10,mean=rep(1:5,each=2),sd=0.7)
* y <- rnorm(10,mean=rep(c(1,9),each=5),sd=0.1)
* dataFrame<- data.frame(x=x,y=y)
* set.seed(143)
* dataMatrix<-as.matrix(dataFrame)[sample(1:10),]
* heatmap(dataMatrix)

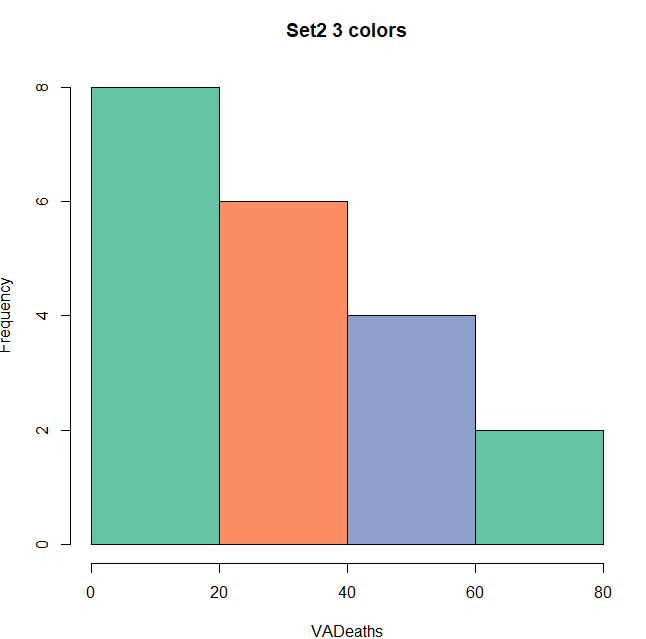
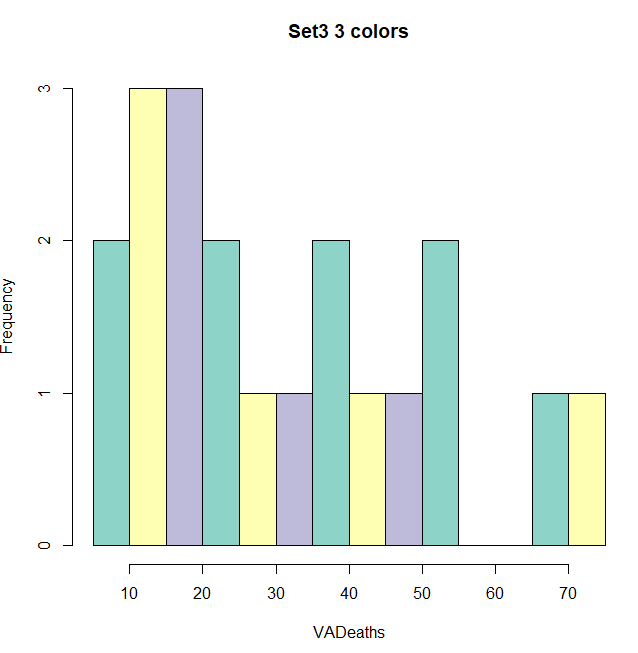
1. Correlogram

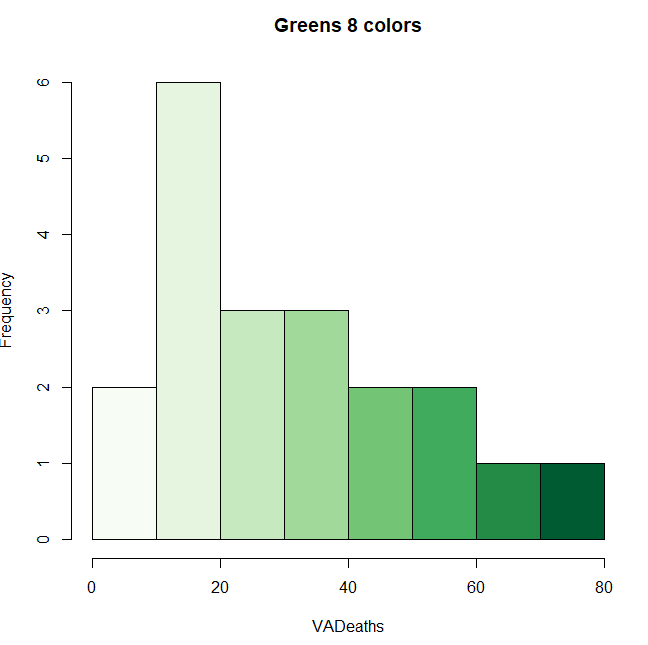
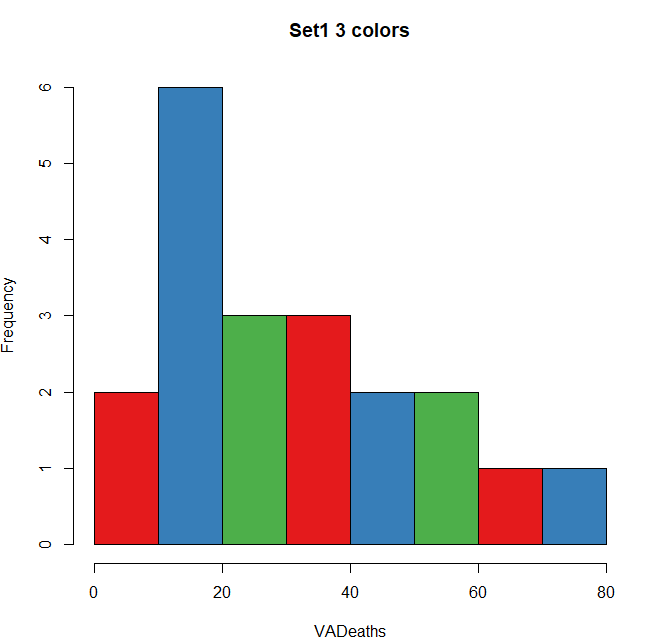
* library("corrplot")
* data("mtcars")
* corr\_matrix <- cor(mtcars)
* corrplot(corr\_matrix)
* corrplot(corr\_matrix,method = 'number',type = "lower")

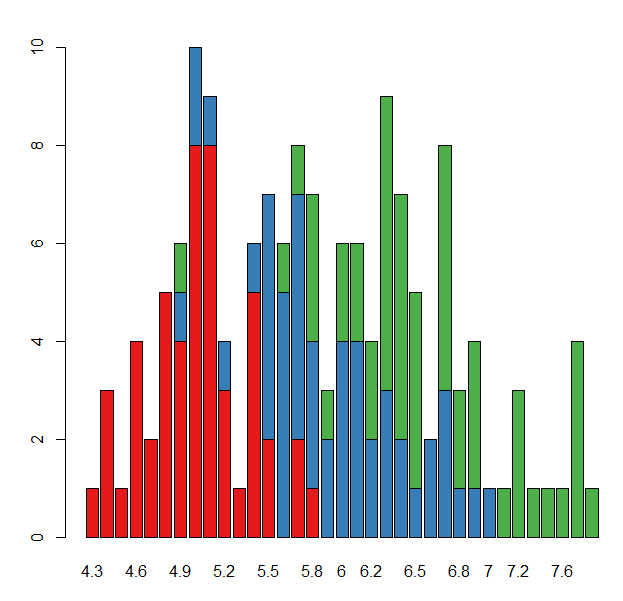
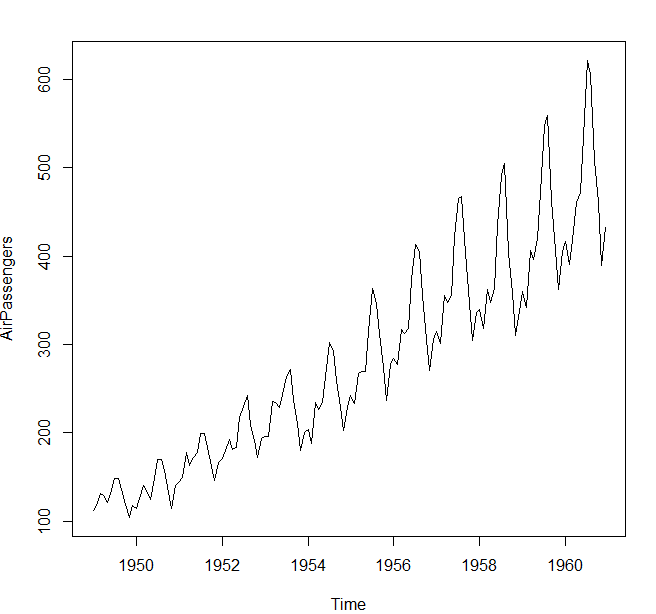
1. Area Chart

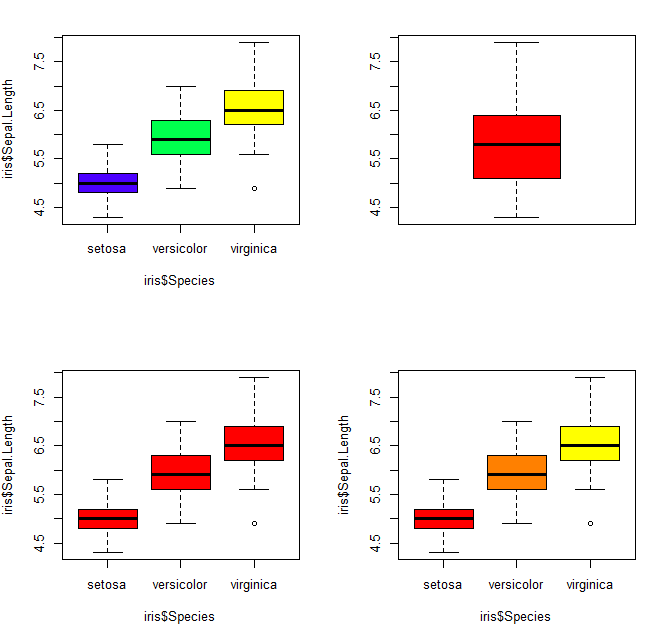
* library(dplyr)
* library(ggplot2)
* airquality %>%
  + group\_by(Day) %>%
  + summarise(mean\_wind = mean(Wind)) %>%
  + ggplot() +
  + geom\_area(aes(x = Day, y = mean\_wind)) +
  + labs(title = "Area Chart of Average Wind per Day",
    - subtitle = "using airquality data",
    - y = "Mean Wind")

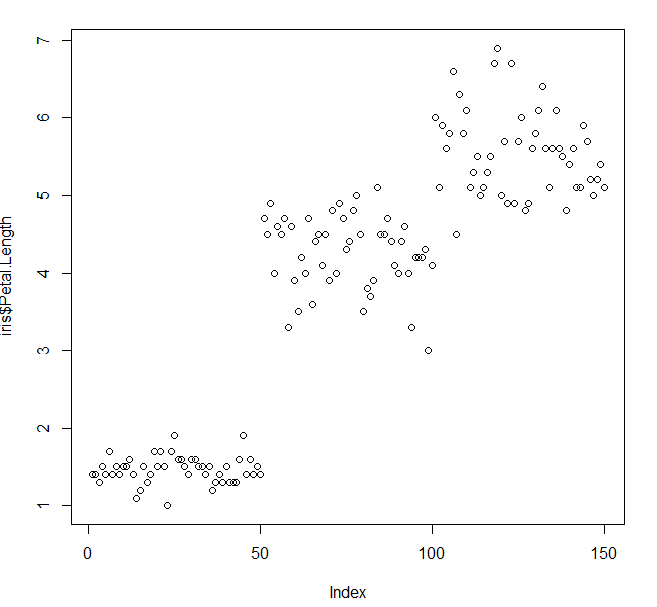
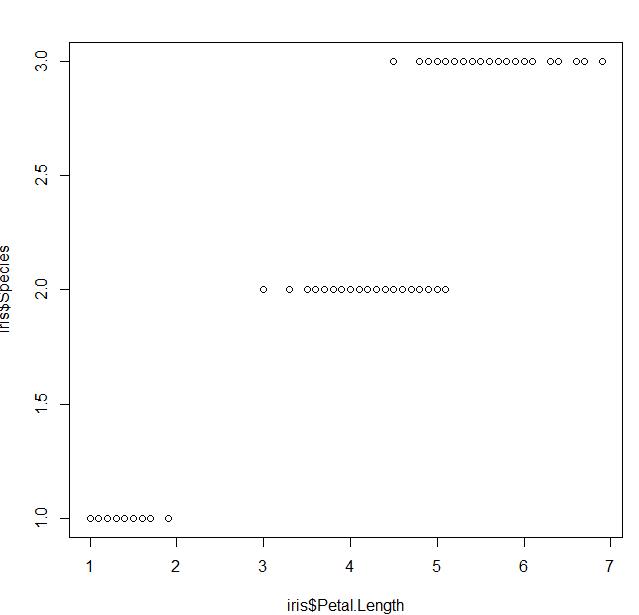
**OUTPUT**

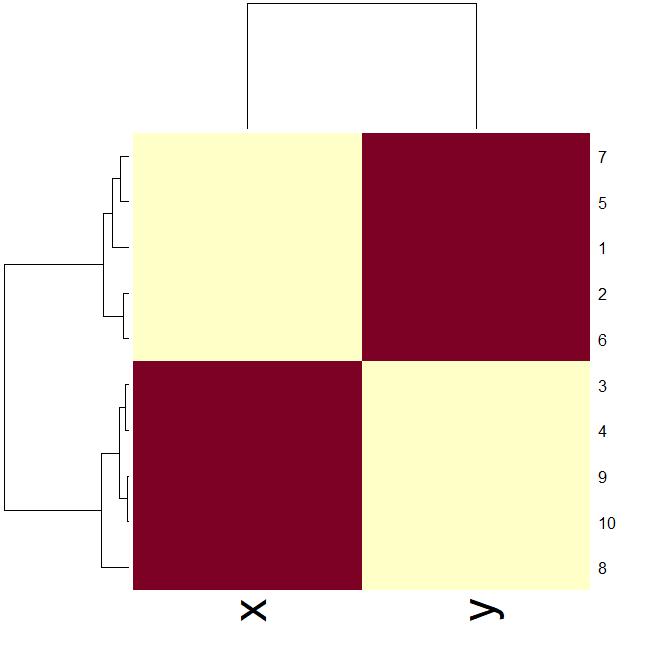


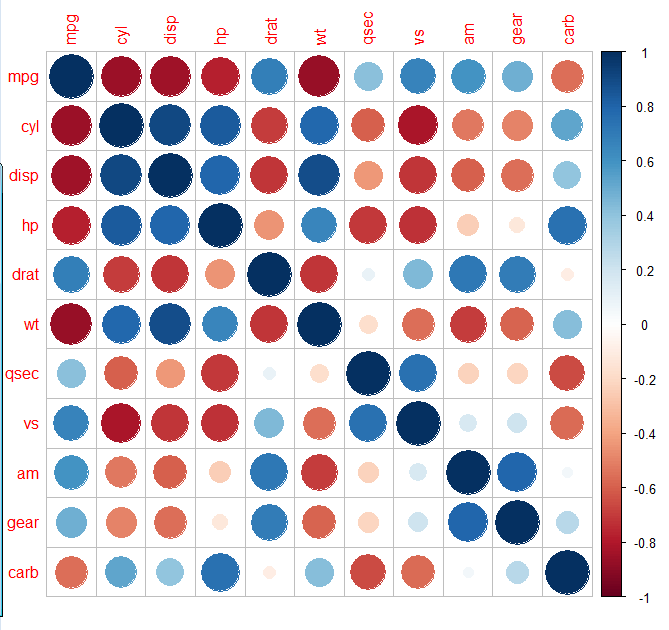
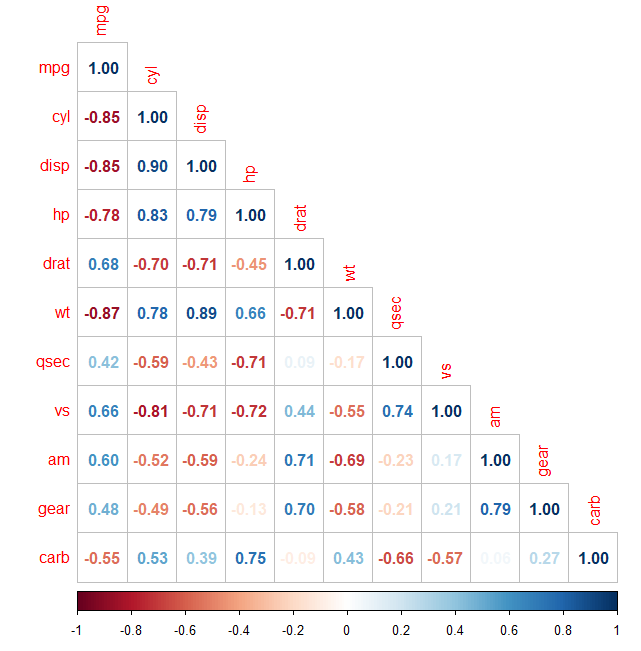


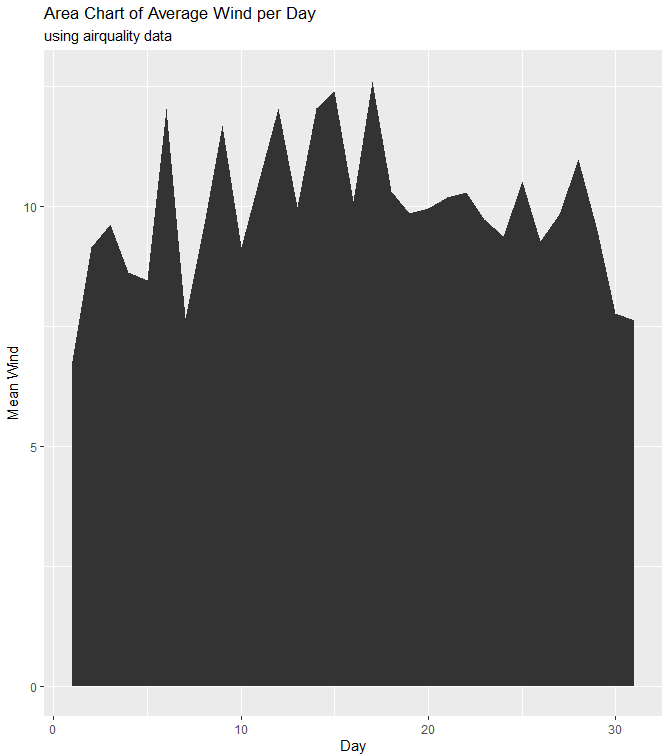








**RESULT**

Thus, the visualization of data using plotting framework has been executed successfully.

|  |  |
| --- | --- |
| **Exp No: 8** | **Implement an application that stores big data in Hbase / MongoDB / Pig using Hadoop / R** |
| **Date:** |

**AIM**

To implement an application that stores big data in mongoDB using R.

**PROCEDURE**

1. Open R on windows.

2. Create a new workspace.

3. Create a new script file.

4. Type the code in the script file.

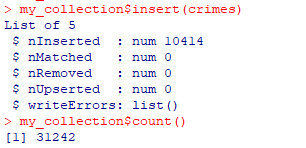
5. Run the script file.

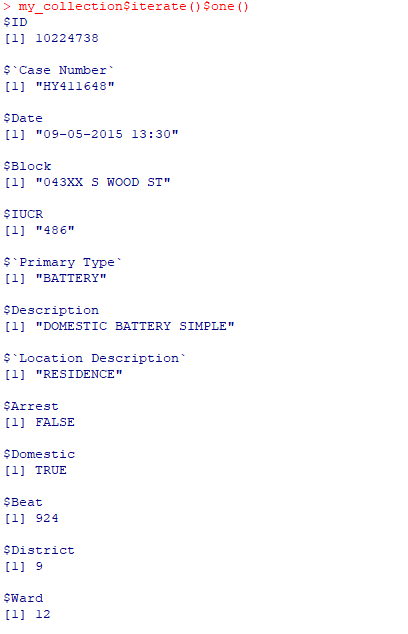
6. Close R.

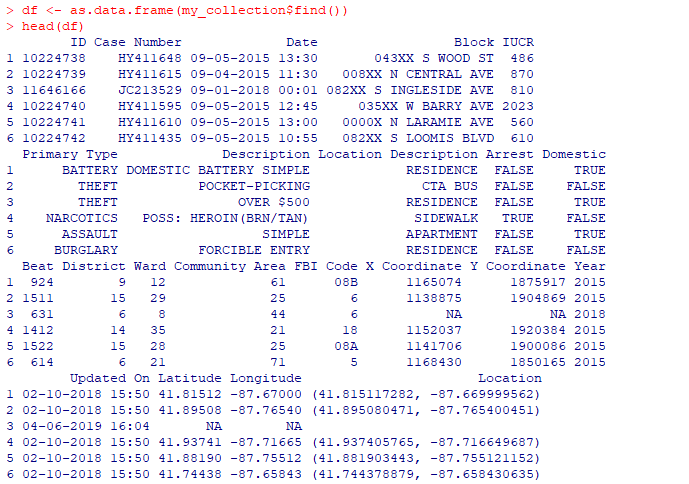
**PROGRAM**

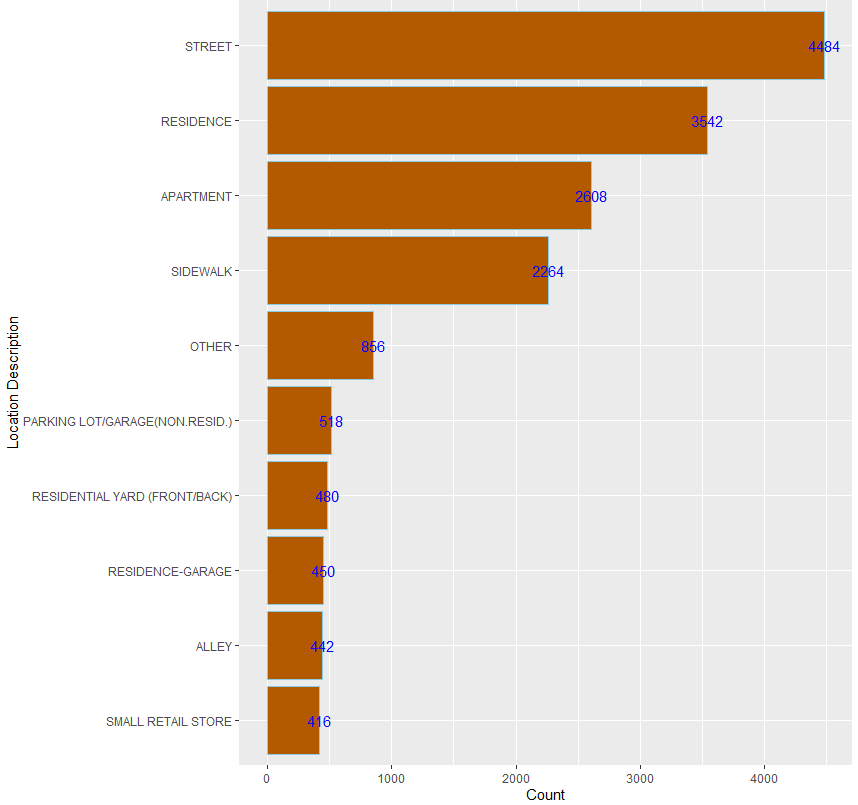
* library(ggplot2)
* library(mongolite)
* library(dplyr)
* crimes=data.table::fread("crimes.csv")
* connection\_string="mongodb://localhost:27017/?tls=false&readPreference=primary"
* my\_collection = mongo(collection = "crimes", db = "chicago",url=connection\_string)
* my\_collection$insert(crimes)
* my\_collection$count()
* my\_collection$iterate()$one()
* df <- as.data.frame(my\_collection$find())
* head(df)
* length(my\_collection$distinct("Primary Type"))
* my\_collection$aggregate('[{"$group":{"\_id":"$Location Description", "Count": {"$sum":1}}}]')%>%na.omit()%>%
* arrange(desc(Count))%>%head(10)%>%
* ggplot(aes(x=reorder(`\_id`,Count),y=Count))+
* geom\_bar(stat="identity",color='skyblue',fill='#b35900')+geom\_text(aes(label = Count), color = "blue") +coord\_flip()+xlab("Location Description")
* crimes=my\_collection$find('{}', fields = '{"\_id":0, "Primary Type":1,"Year":1}')
* crimes%>%group\_by("Primary Type")%>%summarize(Count=n())%>%arrange(desc(Count))%>%head(4)

**OUTPUT**









**RESULT**

Thus, the implementation of application to store big data in mongoDB using R has been executed successfully.