ERODE SENGUNTHAR
ENGINEERING COLLEGE

(APPROVED BY AICTE, NEW DELHI & PERMANANENTLY AFFLICATED TO ANNA UNIVERSITY,CHENNAI
ACCREDITED BY NBA, NEW DELHI, NAAC WITH GRADE “A” & IE(I), KOLKATA)

PERUNDURAI, ERODE – 638 057.

An Autonomous Institution

BONAFIDE CERTIFICATE

Register No. ……………………………………………………………………………………………

Certified that this is the Bonafide Record of Work Done

Name of the Student : ……………………………………………………………………

Branch : ………………………………………………………………………

Lab Code/Name : ……………………………………………………………………

Semester : ……………………………………………………………………

Faculty Incharge   Head of the Department

Submitted for the End semester Practical Examination held on………………………………………………

Internal Examiner   External Examiner

INDEX

Ex.No:1 **Hypothesis Test using R**

Date :

Aim:

To write an R program of Hypothesis Test operations

Source code:

Two sample t-test :

```
s.pool <- sqrt((sd_db ^ 2 + sd_ndb ^ 2) / 2)

n <- nrow(diabetes_sim)

test.stat <- (mean_db - mean_ndb)/(sqrt( 2 * s.pool ^ 2 / n))

test.stat
```

output:

```
##  Two Sample t-test
##
## data:  diabetes_sim[, 1] and diabetes_sim[, 2]
## t = 2.2704, df = 40, p-value = 0.02864
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   1.299436 22.366364
## sample estimates:
## mean of x mean of y
##  152.7791  140.9462
```

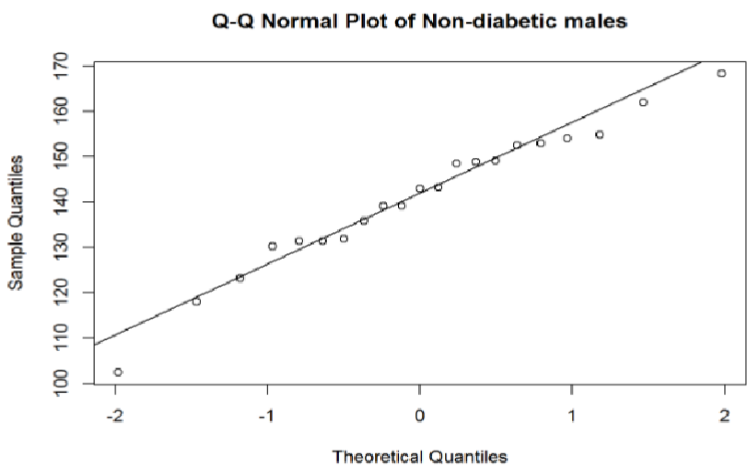
Model validation :

```
var.test(diabetes_sim[, 1], diabetes_sim[, 2])
##
##  F test to compare two variances
##
## data:  diabetes_sim[, 1] and diabetes_sim[, 2]
## F = 1.3934, num df = 20, denom df = 20, p-value = 0.4648
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.5653781 3.4339273
## sample estimates:
## ratio of variances
##           1.393365
t.test(diabetes_sim[, 1], diabetes_sim[, 2], var.equal=FALSE)
##
## Welch Two Sample t-test
##
## data:  diabetes_sim[, 1] and diabetes_sim[, 2]
## t = 2.2704, df = 38.948, p-value = 0.02879
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   1.290565 22.375234
## sample estimates:
## mean of x mean of y
##  152.7791  140.9462
```

Assessing Normality:

```
qqnorm(diabetes_sim[,2], main ="Q-Q Normal Plot of Non-diabetic males")
qqline(diabetes_sim[,2])
shapiro.test(diabetes_sim[,2])
##
## Shapiro-Wilk normality test
##
## data:  diabetes_sim[, 2]
## W = 0.97142, p-value = 0.7643
```

Output :



Checking the data:

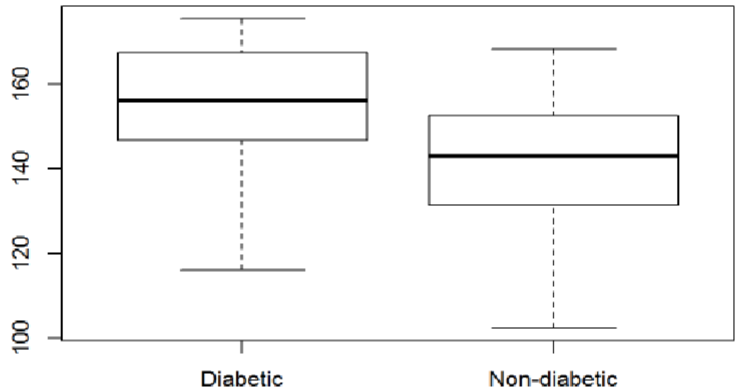
```
diabetes_sim[, 1] # Inspect first column of data - SBP for diabetic men
##  [1] 156.0744 170.1177 152.5676 116.1372 162.6186 169.5206 143.2227
##  [8] 167.4009 167.9799 164.4992 154.4027 167.2837 175.3905 119.2637
## [15] 169.0121 125.4921 159.9883 121.8443 146.9799 146.6104 151.9541
diabetes_sim[1:5, ] # First five entries only of both columns
##   Diabetes Non.diabetes
## 1 156.0744     162.0524
## 2 170.1177     168.3068
## 3 152.5676     153.9854
## 4 116.1372     148.8356
## 5 162.6186     131.3819
boxplot(diabetes_sim[,1], diabetes_sim[, 2], names = c("Diabetic", "Non-diabetic"))
stripchart(diabetes_sim, method="jitter",
vertical = TRUE, group.names = c("Diabetic", "Non-diabetic"))
```

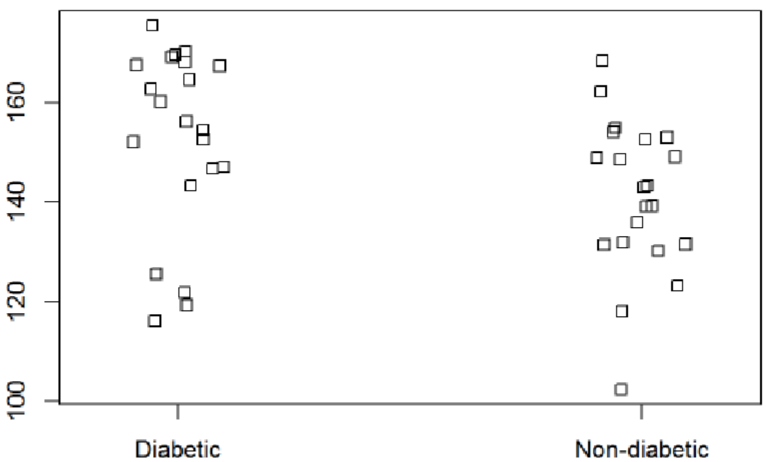
We can also summarise basic aspects of the data using the mean and sd functions.

```
mean_db <- mean(diabetes_sim[, 1])
mean_ndb <- mean(diabetes_sim[, 2])

mean_db
## [1] 152.7791
mean_ndb
## [1] 140.9462
```

```
sd_db <- sd(diabetes_sim[, 1])
sd_ndb <- sd(diabetes_sim[, 2])
```

OUT PUT:





Result:

Thus the R program for the Hypothesis Test operations has been implementedand executed successfully.

Ex.No:2 **Implementation of K-means Clustering using R**

Date:

Aim:

To write an program of K-means Clustering using R Language.

PROBLEM DEFINATION:

CLUSTERING MODEL
e. Clustering algorithms for unsupervised classification.Plot the cluster data using R visualizations

SOURCE CODE :

1. **Clustering algorithms for unsupervised**

```
classification.library(cluster)

> set.seed(20)
> irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
# nstart = 20. This means that R will try 20 different random starting assignments and then
select the onewith the lowest within cluster variation.
> irisCluster
```

OUTPUT:

```
  Petal.Length
 Petal.Width1
      1.462000
      0.246000
2     4.269231   1.342308
3     5.595833   2.037500
```

Clustering vector:

```
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [42] 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2
 [83] 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3
[124] 3 3 3 2 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3
```

Within cluster sum of squares by cluster:

```
[1] 2.02200 13.05769 16.29167
 (between_SS / total_SS =  94.3 %)
```

Available components:

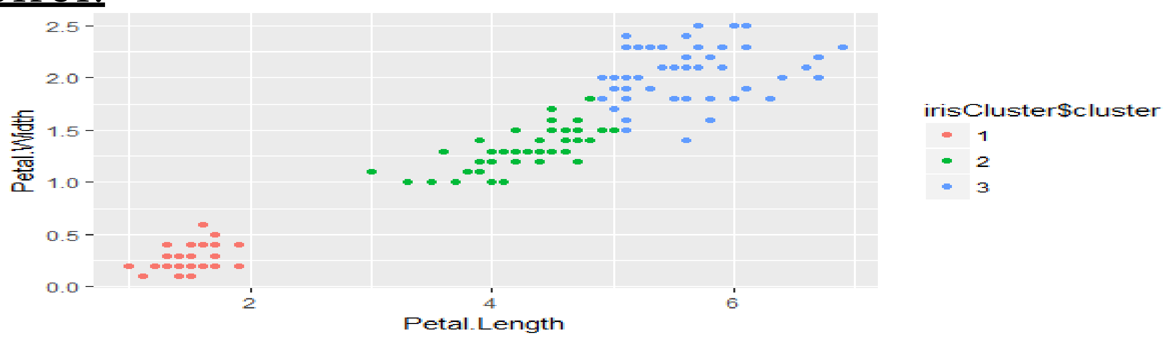
```
[1] "cluster"    "centers"    "totss"      "withinss"6 "tot.withinss"
```

[6] "betweenss" "size"   "iter"   "ifault"

SOURCE CODE:

```
> irisCluster$cluster <- as.factor(irisCluster$cluster)
> ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_point()
```

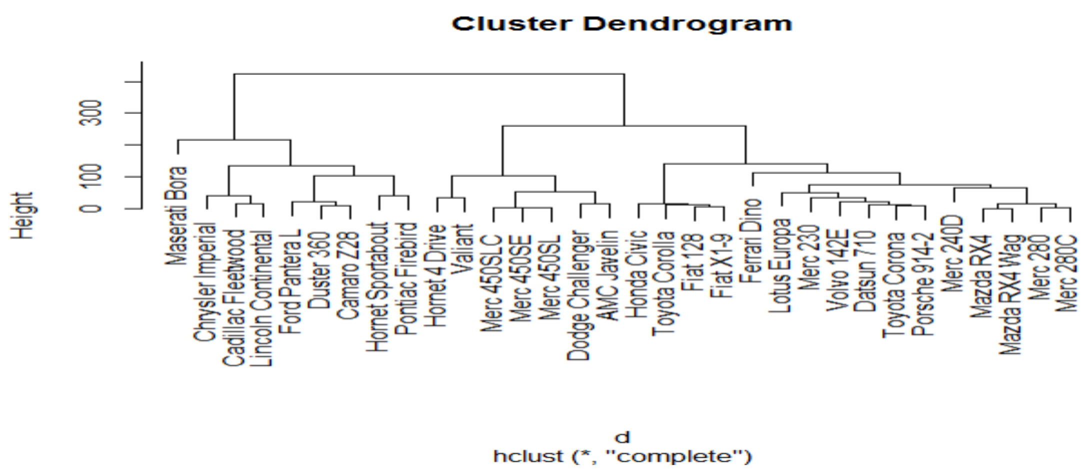
OUTPUT:



SOURCE CODE:

```
> d <- dist(as.matrix(mtcars))  # find distance matrix
> hc <- hclust(d)              # apply hirarchical clustering
> plot(hc)                     # plot the dendrogram
```

OUTPUT:

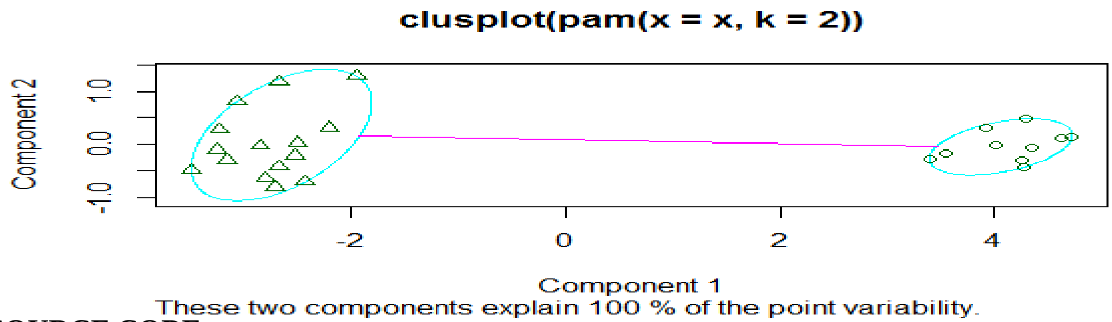


2. **Plot the cluster data using R visualizations.**

SOURCE CODE:

```
## generate 25 objects, divided into 2 clusters.
x <- rbind(cbind(rnorm(10,0,0.5), rnorm(10,0,0.5)),
cbind(rnorm(15,5,0.5),
rnorm(15,5,0.5)))clusplot(pam(x, 2))
```

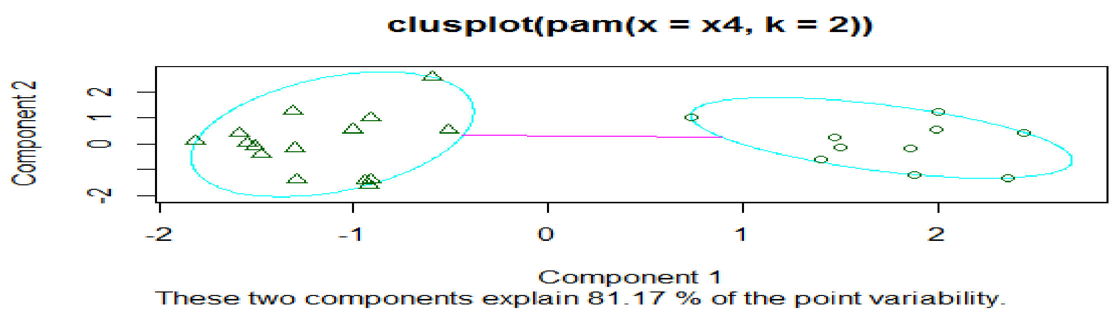
OUTPUT:



SOURCE CODE:

```
## add noise, and try again :
x4 <- cbind(x, rnorm(25),
rnorm(25))clusplot(pam(x4,
2))
```

OUTPUT:



Result:

Thus the program of K-means Clustering using R Language.

has been verified that successfully.

Ex.No:3 Implementation of Linear & Logistic Regression

Aim:

To write an R program to Implementation of Linear & Logistic Regression

PROGRAM:

****SIMPLE LINEAR REGRESSION****

```
dataset = read.csv("data-marketing-budget-12mo.csv", header=T,
colClasses = c("numeric", "numeric", "numeric"))
head(dataset,5)
#/////Simple Regression/////
simple.fit = lm(Sales~Spend,data=dataset)
summary(simple.fit)
```

OUTPUT:

```
Call:
lm(formula = Sales ~ Spend, data = dataset)

Residuals:
   Min     1Q Median     3Q    Max
 -3385  -2097    258   1726   3034

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1383.4714  1255.2404   1.102    0.296
Spend         10.6222     0.1625  65.378 1.71e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2313 on 10 degrees of freedom
Multiple R-squared:  0.9977, Adjusted R-squared:  0.9974
F-statistic:  4274 on 1 and 10 DF,  p-value: 1.707e-14
```

****MULTIPLE LINEAR REGRESSION ****

```
multi.fit = lm(Sales~Spend+Month, data=dataset)
summary(multi.fit)
```

MULTIPLE LINEAR REGRESSION OUTPUT:

```
Call:
lm(formula = Sales ~ Spend + Month, data = dataset)

Residuals:
     Min       1Q   Median       3Q      Max
-1793.73 -1558.33    -1.73  1374.19  1911.58

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -567.6098  1041.8836  -0.545  0.59913
Spend         10.3825     0.1328  78.159 4.65e-14 ***
Month        541.3736   158.1660   3.423  0.00759 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1607 on 9 degrees of freedom
Multiple R-squared:  0.999, Adjusted R-squared:  0.9988
F-statistic:  4433 on 2 and 9 DF,  p-value: 3.368e-14
```

******Logistic Regression ******

```
#selects some column from mtcars

input<- mtcars [,c("am","cyl","hp","wt")]

print(head(input))

input<- mtcars [,c("am","cyl","hp","wt")]

am.data =glm(formula = am ~ cyl+hp+wt,data = input,family = binomial)

print(summary(am.data))
```

OUTPUT:

```
> print(head(input))
                  am cyl  hp    wt
Mazda RX4          1   6 110 2.620
Mazda RX4 Wag      1   6 110 2.875
Datsun 710         1   4  93 2.320
Hornet 4 Drive     0   6 110 3.215
Hornet Sportabout  0   8 175 3.440
Valiant            0   6 105 3.460
```

```
Call:
glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)

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
```

Result:

Thus the R program to Implementation of Linear & Logistic Regression has been implemented and executed successfully

Ex.No:4 Implementation and perform the Time-series Analysis using R

Aim:

To write a Program Time-series Analysis using R

Reading Time Series Data:

```
> kings  <-  scan("http://robjhyndman.com/tsdldata/misc/kings.dat",skip=3)
  Read 42 items
> kings
  [1] 60 43 67 50 56 42 50 65 68 43 65 34 47 34 49 41 13 35 53 56 16 43 69 59 48
  [26] 59 86 55 68 51 33 49 67 77 81 67 71 81 68 70 77 56
```

```
> kingstimeseries <- ts(kings)
> kingstimeseries
  Time Series:
  Start = 1
  End = 42
  Frequency = 1
  [1] 60 43 67 50 56 42 50 65 68 43 65 34 47 34 49 41 13 35 53 56 16 43 69 59 48
  [26] 59 86 55 68 51 33 49 67 77 81 67 71 81 68 70 77 56
> births  <-  scan("http://robjhyndman.com/tsdldata/data/nybirths.dat")
  Read 168 items
> birthstimeseries <- ts(births, frequency=12, start=c(1946,1))
> birthstimeseries
    Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct    Nov    Dec
  1946 26.663 23.598 26.931 24.740 25.806 24.364 24.477 23.901 23.175 23.227 21.672
21.870
  1947 21.439 21.089 23.709 21.669 21.752 20.761 23.479 23.824 23.105 23.110 21.759
22.073
  1948 21.937 20.035 23.590 21.672 22.222 22.123 23.950 23.504 22.238 23.142 21.059
21.573
  1949 21.548 20.000 22.424 20.615 21.761 22.874 24.104 23.748 23.262 22.907 21.519
22.025
  1950 22.604 20.894 24.677 23.673 25.320 23.583 24.671 24.454 24.122 24.252 22.084
22.991
  1951 23.287 23.049 25.076 24.037 24.430 24.667 26.451 25.618 25.014 25.110 22.964
23.981
  1952 23.798 22.270 24.775 22.646 23.988 24.737 26.276 25.816 25.210 25.199 23.162
24.707
  1953 24.364 22.644 25.565 24.062 25.431 24.635 27.009 26.606 26.268 26.462 25.246
25.180
  1954 24.657 23.304 26.982 26.199 27.210 26.122 26.706 26.878 26.152 26.379 24.712
25.688
  1955 24.990 24.239 26.721 23.475 24.767 26.219 28.361 28.599 27.914 27.784 25.693
26.881
```

```
  1956 26.217  24.218  27.914  26.975  28.527  27.139  28.982  28.169  28.056  29.136  26.291
26.987
  1957 26.589  24.848  27.543  26.896  28.878  27.390  28.065  28.141  29.048  28.484  26.634
27.735
  1958 27.132  24.924  28.963  26.589  27.931  28.009  29.229  28.759  28.405  27.945  25.912
26.619
  1959 26.076  25.286  27.660  25.951  26.398  25.565  28.865  30.000  29.261  29.012  26.992
27.897
```

```
> souvenir <- scan("http://robjhyndman.com/tsdldata/data/fancy.dat")
  Read 84 items
> souvenirtimeseries  <-  ts(souvenir,  frequency=12,  start=c(1987,1))
> souvenirtimeseries
  Jan        Feb        Mar        Apr        May        Jun        Jul        Aug        Sep
Oct        Nov        Dec
  1987    1664.81    2397.53    2840.71    3547.29    3752.96    3714.74    4349.61
3566.34    5021.82    6423.48    7600.60   19756.21
  1988    2499.81    5198.24    7225.14    4806.03    5900.88    4951.34    6179.12
4752.15    5496.43    5835.10   12600.08   28541.72
  1989    4717.02    5702.63    9957.58    5304.78    6492.43    6630.80    7349.62
8176.62    8573.17    9690.50   15151.84   34061.01
  1990    5921.10    5814.58   12421.25    6369.77    7609.12    7224.75    8121.22
7979.25    8093.06    8476.70   17914.66   30114.41
  1991    4826.64    6470.23    9638.77    8821.17    8722.37   10209.48   11276.55
12552.22   11637.39   13606.89   21822.11   45060.69
  1992    7615.03    9849.69   14558.40   11587.33    9332.56   13082.09   16732.78
19888.61   23933.38   25391.35   36024.80   80721.71
  1993   10243.24   11266.88   21826.84   17357.33   15997.79   18601.53   26155.15
28586.52   30505.41   30821.33   46634.38  104660.67
```

Result:

Thus the program of K-means Clustering using R Language. has been verified that successfully.

Ex.No:5 **Data Analysis-Visualization using R**

Aim:

To implement *Data visualization* is to provide an efficient graphical display for summarizingand reasoning about quantitative information using R .

1. Histogram:

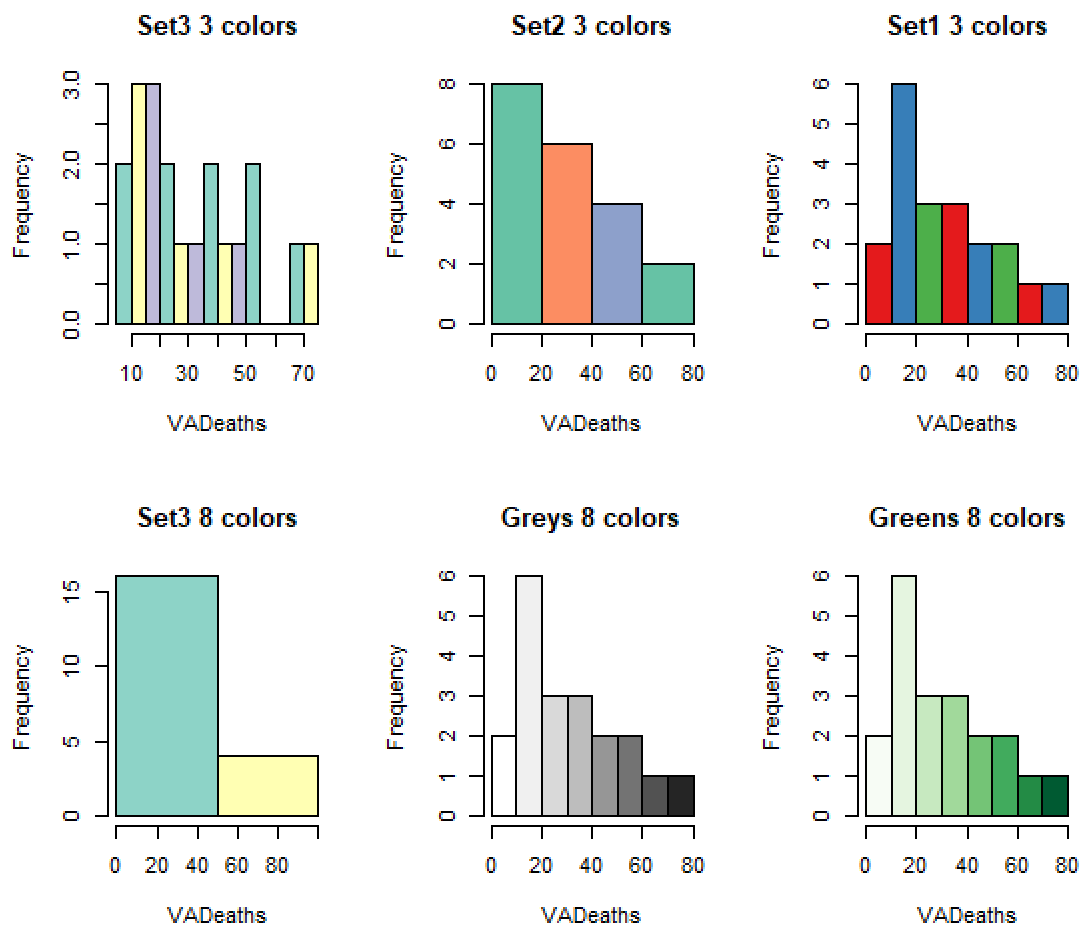
Histogram is basically a plot that breaks the data into bins (or breaks) and shows frequencydistribution of these bins. You can change the breaks also and see the effect it has data visualization in terms of understandability.

Note: We have used par(mfrow=c(2,5)) command to fit multiple graphs in same page for sake of clarity(see the code below).

PROGRAM:

```
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")
```

OUTPUT:



Line Chart

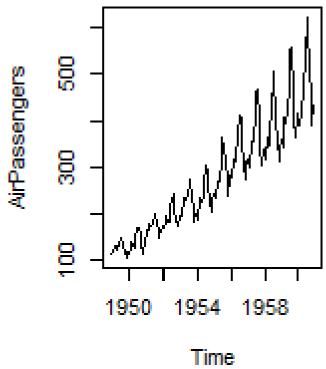
Below is the line chart showing the increase in air passengers over given time period. Line Chartsare commonly preferred when we are to analyses a trend spread over a time period. Furthermore,line plot is also suitable to plots where we need to compare relative changes in quantities across some variable (like time). Below is the code:

PROGRAM:

data(AirPassengers) plot(AirPassengers,type="l") #Simple

Line Plot

OUTPUT:



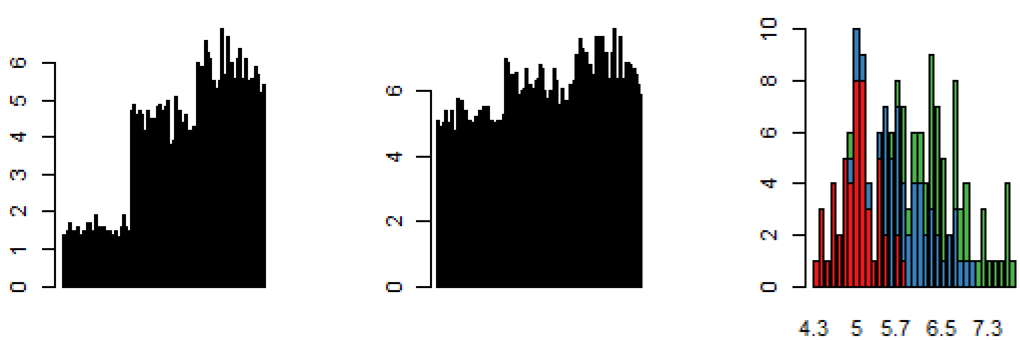
Bar Chart

Bar Plots are suitable for showing comparison between cumulative totals across several groups.Stacked Plots are used for bar plots for various categories. Here's the code:

PROGRAM:

```
data("iris")
barplot(iris$Petal.Length) #Creating simple Bar Graph
barplot(iris$Sepal.Length,col = brewer.pal(3,"Set1"))
barplot(table(iris$Species,iris$Sepal.Length),col = brewer.pal(3,"Set1")) #Stacked Plot
```

OUTPUT:



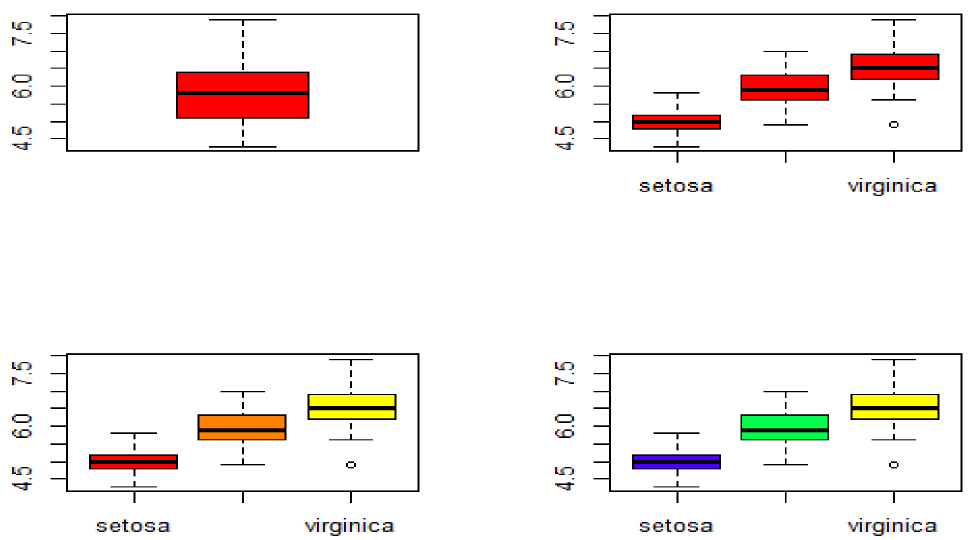
3. Box Plot

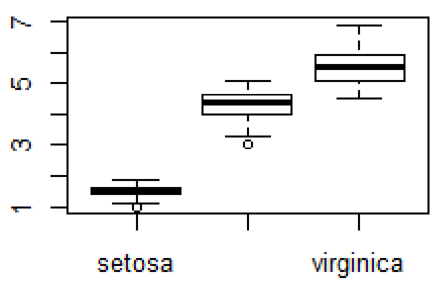
Box Plot shows 5 statistically significant numbers the minimum, the 25th percentile, the median, the 75th percentile and the maximum. It is thus useful for visualizing the spread of the data is and deriving inferences accordingly.

PROGRAM:

```
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) #Creating Box Plot between two variable
```

OUTPUT:





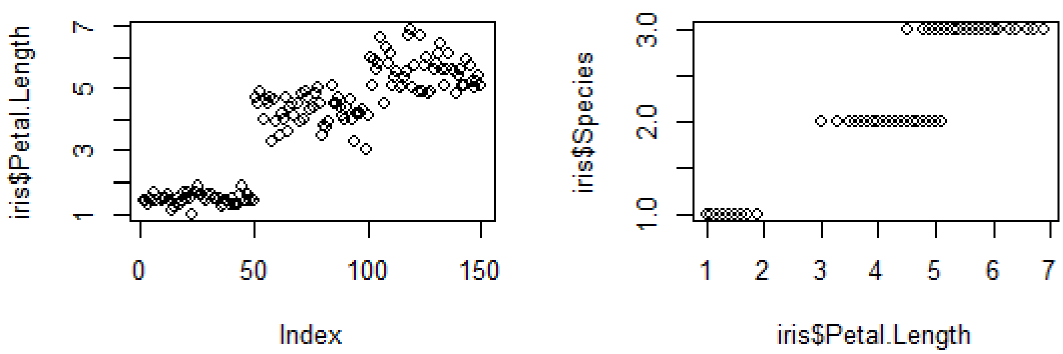
3. Scatter Plot (including 3D and other features)

Scatter plots help in visualizing data easily and for simple data inspection. Here's the code forsimple scatter and multivariate scatter plot:

PROGRAM:

plot(x=iris$Petal.Length) #Simple Scatter Plot

plot(x=iris$Petal.Length,y=iris$Species) #Multivariate Scatter Plot**OUTPUT:**



4. Heat Map

One of the most innovative data visualizations in R, the heat map emphasizes color intensity to visualize relationships between multiple variables. The result is an attractive 2D imagethat is easy to interpret. As a basic example, a heat map highlights the popularity of competing items by ranking them according to their original market launch date. It breaks it down further byproviding sales statistics and figures over the course of time.

PROGRAM:

 # simulate a dataset of 10 points x<-

rnorm(10,mean=rep(1:5,each=2),sd=0.7) y<-

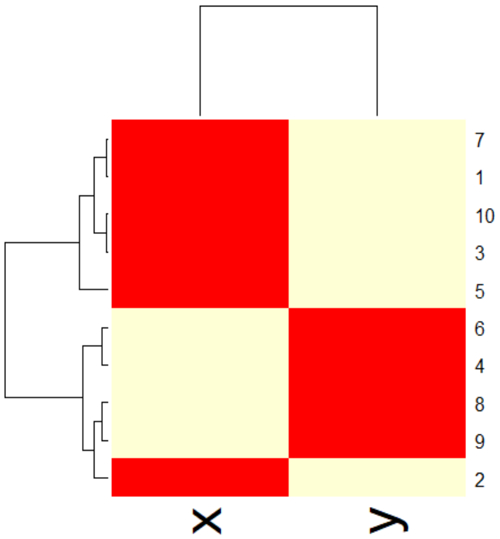
rnorm(10,mean=rep(c(1,9),each=5),sd=0.1)dataFrame<-

data.frame(x=x,y=y) set.seed(143)

dataMatrix<-as.matrix(dataFrame)[sample(1:10),] *# convert to class 'matrix', then shuffle therows of the matrix*

heatmap(dataMatrix) *# visualize hierarchical clustering via a heatmap*

OUTPUT:



3. Correlogram

Correlated data is best visualized through corrplot. The 2D format is similar to a heat map, but ithighlights statistics that are directly related.

Most correlograms highlight the amount of correlation between datasets at various points in time.Comparing sales data between different months or years is a basic example.

PROGRAM:

```
#data("mtcars") corr_matrix <-
cor(mtcars)# with circles
corrplot(corr_matrix)
# with numbers and lower
```

corrplot(corr_matrix,method = 'number',type = "lower")

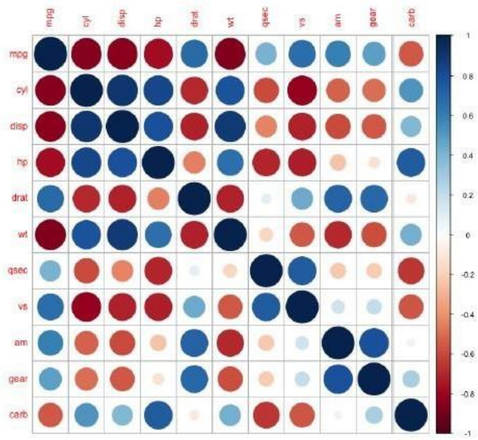


Fig 6. Correlogram with circles (courtesy of Abdul Majed Raja)

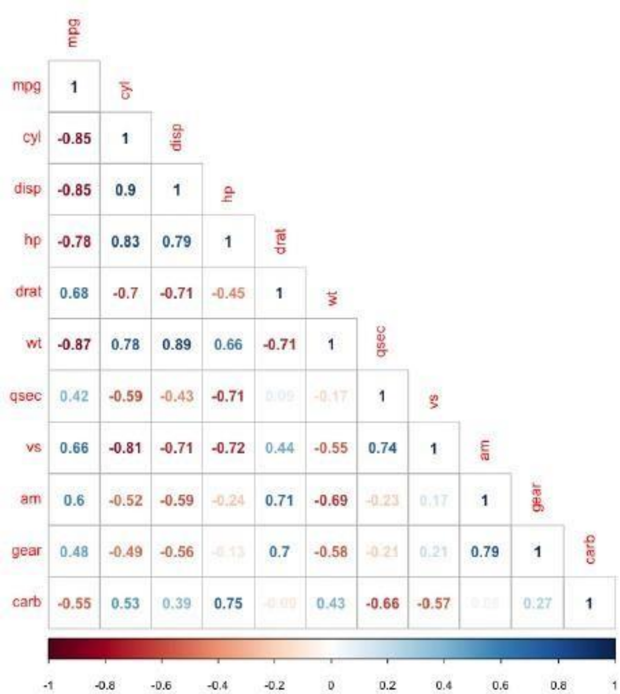


Fig 7. Correlogram with numbers (courtesy of Abdul Majed Raja)

3. Area Chart

Area charts express continuity between different variables or data sets. It's akin to the traditionalline chart you know from grade school and is used in a similar fashion.

Most area charts highlight trends and their evolution over the course of time, making them highlyeffective when trying to expose underlying trends whether they're positive or negative.

PROGRAM:

```
data("airquality") #dataset usedairquality
%>% 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:

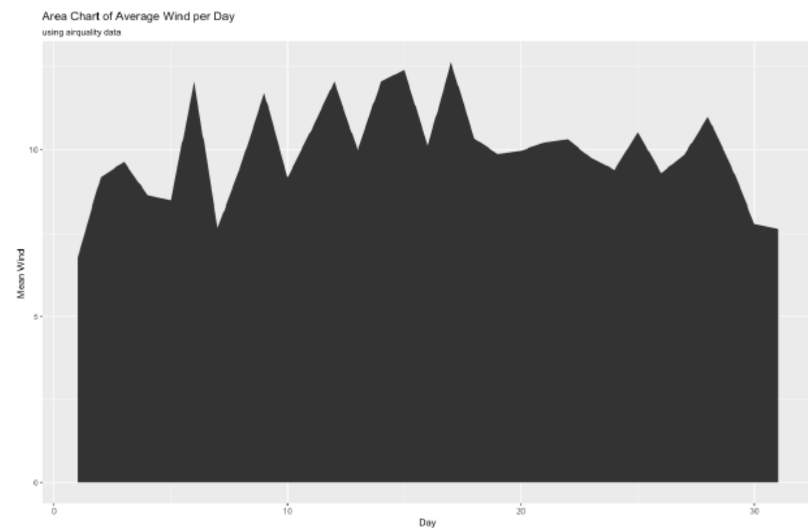


Fig 8. Area chart (courtesy of Abdul Majed Raja)

Result:

Thus the *Data visualization* is to provide an efficient graphical display for summarizing and reasoning about quantitative information using R

Ex.No:6 **Install and Configure Hadoop**

Date:

Aim:

To Install and Configure Hadoop

Step by step Hadoop 2.8.0 installation on Windows 10 Prepare:

These software's should be prepared to install Hadoop 2.8.0 on window 10 64 bits.

1) Download Hadoop 2.8.0
   (Link: http://wwweu.apache.org/dist/hadoop/common/hadoop-2.8.0/hadoop-2.8.0.tar.gz OR http://archive.apache.org/dist/hadoop/core//hadoop-2.8.0/hadoop-2.8.0.tar.gz)
2) Java JDK 1.8.0.zip
   (Link: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)

Set up:

1) Check either Java 1.8.0 is already installed on your system or not, use "Javac -version" to check Java version

2) If Java is not installed on your system then first install java under "C:\JAVA" Javasetup

3) Extract files Hadoop 2.8.0.tar.gz or Hadoop-2.8.0.zip and place under "C:\Hadoop-2.8.0" hadoop

4) Set the path HADOOP_HOME Environment variable on windows 10(see Step 1,2, 3 and 4 below) hadoop

5) Set the path JAVA_HOME Environment variable on windows 10(see Step 1, 2,3 and 4 below) java

6) Next we set the Hadoop bin directory path and JAVA bin directory path

Configuration

a) File C:/Hadoop-2.8.0/etc/hadoop/core-site.xml, paste below xml paragraph andsave this file.

```
<configuration>
     <property>
   <name>fs.defaultFS</name>
   <value>hdfs://localhost:9000</value>
     </property>
</configuration>
```

b)Rename "mapred-site.xml.template" to "mapred-site.xml" and edit this file C:/Hadoop-2.8.0/etc/hadoop/mapred-site.xml, paste below xml paragraph and savethis file.

```
    <configuration>
 <property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
 </property>
    </configuration>
```

c) Create folder "data" under "C:\Hadoop-2.8.0"

1) Create folder "datanode" under "C:\Hadoop-2.8.0\data"

2) Create folder "namenode" under "C:\Hadoop-2.8.0\data" data

d) Edit file C:\Hadoop-2.8.0/etc/hadoop/hdfs-site.xml, paste below xml paragraphand save this file.

```
<configuration>
 <property>
```

```
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>C:\hadoop-2.8.0\data\namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>C:\hadoop-2.8.0\data\datanode</value>
    </property>
</configuration>
```

e) Edit file C:/Hadoop-2.8.0/etc/hadoop/yarn-site.xml, paste below xml paragraphand save this file.

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
</configuration>
```

f) Edit file C:/Hadoop-2.8.0/etc/hadoop/hadoop-env.cmd by closing the command line "JAVA_HOME=%JAVA_HOME%" instead of set "JAVA_HOME=C:\Java" (On C:\java this is path to file jdk.18.0)

Hadoop Configuration

7) Download file Hadoop Configuration.zip (Link: https://github.com/MuhammadBilalYar/HADOOP-INSTALLATION-ON- WINDOW-10/blob/master/Hadoop%20Configuration.zip)

8) Delete file bin on C:\Hadoop-2.8.0\bin, replaced by file bin on file just download(from Hadoop Configuration.zip).

9) Open cmd and typing command "hdfs namenode –format" .You will see hdfsnamenode –format

Testing

10) Open cmd and change directory to "C:\Hadoop-2.8.0\sbin" and type "start-all.cmd" to start apache.

11) Make sure these apps are running.

a) Name node

b)Hadoop data node

c) YARN Resource Manager

d)YARN Node Manager hadoop nodes

12) Open: http://localhost:8088

13) Open: http://localhost:50070

Result:

Thus the Install and Configure Hadoop successfully.

Ex.No:7 **Implementation of word count programs using Map Reduce**

Date:

Aim:

To Implementation of word count programs using Map Reduce

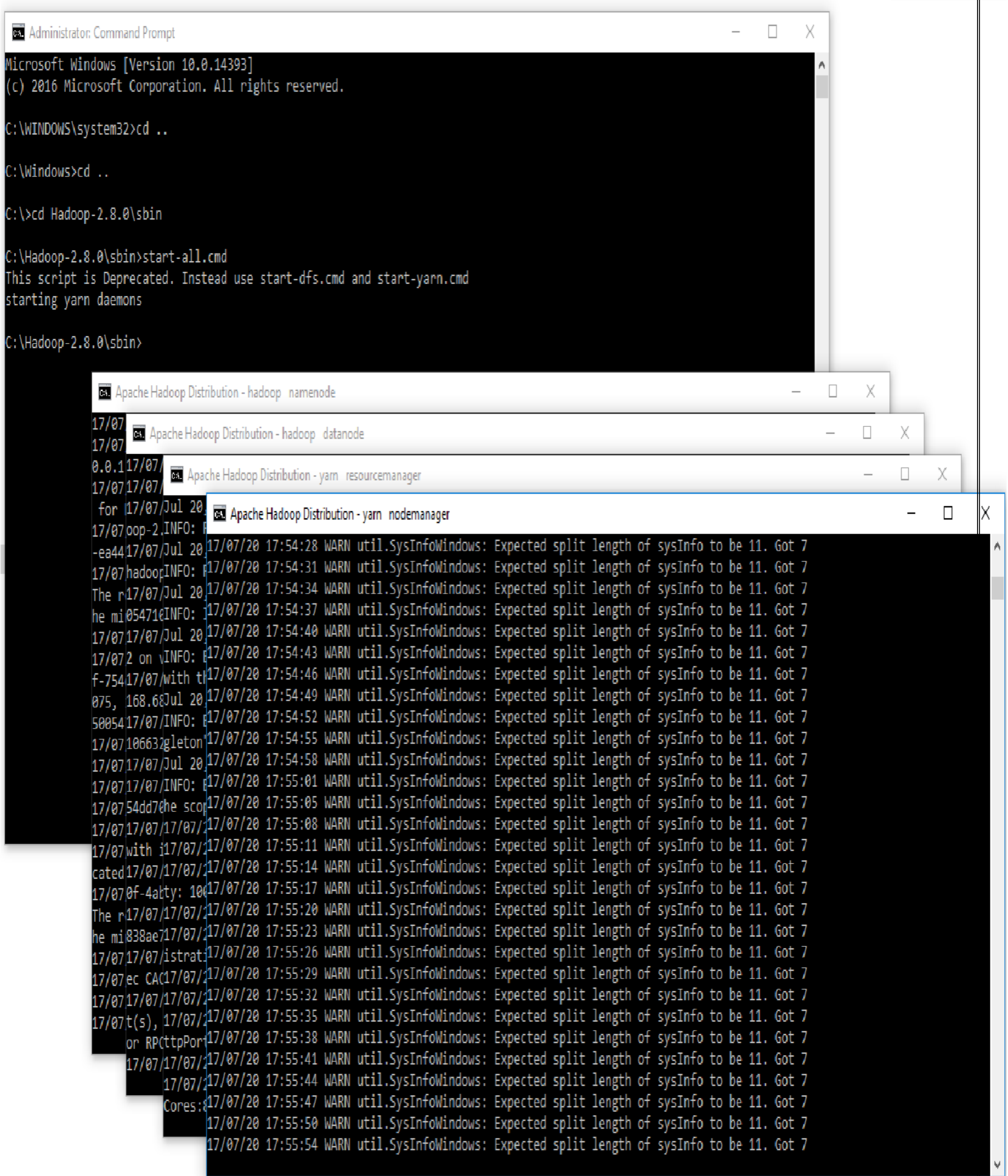
Prepare:

1. Download MapReduceClient.jar
   (Link: https://github.com/MuhammadBilalYar/HADOOP- INSTALLATION-ON-WINDOW-10/blob/master/MapReduceClient.jar)
2. Download Input_file.txt
   (Link: https://github.com/MuhammadBilalYar/HADOOP-INSTALLATION-ON-WINDOW-10/blob/master/input_file.txt)

Place both files in "C:/"

Hadoop Operation:

1. Open cmd in Administrative mode and move to "C:/Hadoop-2.8.0/sbin" andstart cluster
2. **Start-all.cmd**


Administrator: Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\WINDOWS\system32>cd ..
C:\Windows>cd ..
C:\>cd Hadoop-2.8.0\sbin
C:\Hadoop-2.8.0\sbin>start-all.cmd
This script is Deprecated. Instead use start-dfs.cmd and start-yarn.cmd
starting yarn daemons
C:\Hadoop-2.8.0\sbin>
Apache Hadoop Distribution - hadoop namenode
Apache Hadoop Distribution - hadoop datanode
Apache Hadoop Distribution - yarn resourcemanager
Apache Hadoop Distribution - yarn nodemanager
17/07/20 17:54:28 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:31 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:34 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:37 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:40 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:43 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:46 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:49 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:52 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:55 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:54:58 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:01 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:05 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:08 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:11 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:14 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:17 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:20 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:23 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:26 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:29 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:32 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:35 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:38 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:41 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:44 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:47 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:50 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7
17/07/20 17:55:54 WARN util.SysInfoWindows: Expected split length of sysInfo to be 11. Got 7

3. Create an input directory in HDFS.

hadoop fs -mkdir /input_dir

4. Copy the input text file named input_file.txt in the input directory (input_dir)of HDFS.

hadoop fs -put C:/input_file.txt /input_dir

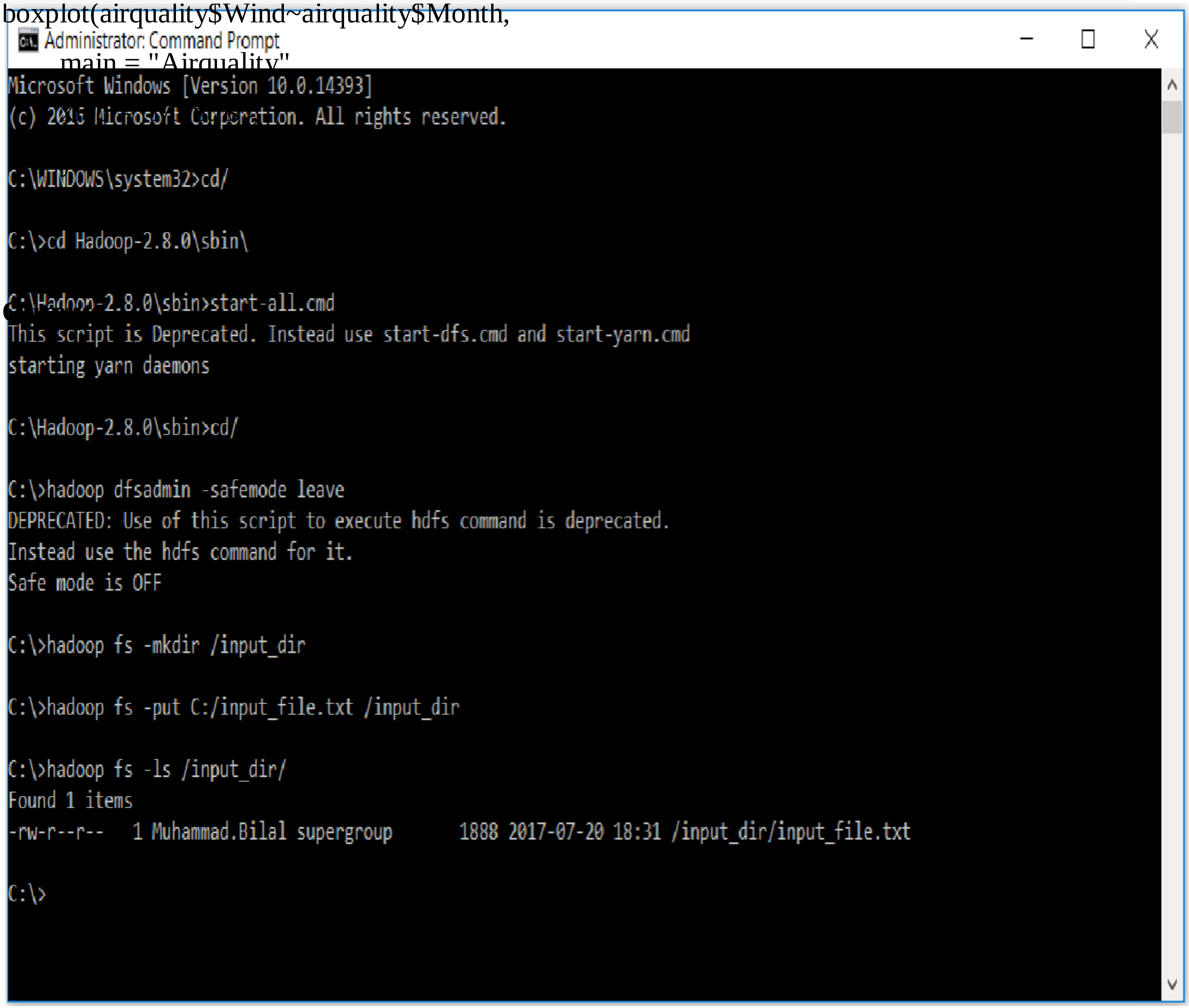
Verify input_file.txt available in HDFS input director

Assigning names to Boxplot in R Programming

hadoop fs -ls /input_dir/

boxplot(airquality$Wind~airquality$Month,

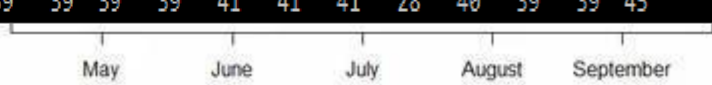
main = "Airquality"



1. Verify content of the copied file.

hadoop dfs -cat /input_dir/input_file.txt

```
C:\>hadoop fs -ls /input_dir/
Found 1 items
-rw-r--r--   1 Muhammad.Bilal supergroup       1888 2017-07-20 18:31 /input_dir/input_file.txt

C:\>hadoop dfs -cat /input_dir/input_file.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
23   23   27  43   24   25   26   26   26   26   25   26   25
26   27   28  28   28   30   31   31   31   30   30   30   29
31   32   32  32   33   34   35   36   36   34   34   34   34
39   38   39  39   39   41   42   43   40   39   38   38   40
38   39   39  39   39   41   41   41   28   40   39   39   45
23   23   27  43   24   25   26   26   26   26   25   26   25
26   27   28  28   28   30   31   31   31   30   30   30   29
31   32   32  32   33   34   35   36   36   34   34   34   34
39   38   39  39   39   41   42   43   40   39   38   38   40
38   39   39  39   39   41   41   41   28   40   39   39   45
23   23   27  43   24   25   26   26   26   26   25   26   25
26   27   28  28   28   30   31   31   31   30   30   30   29
31   32   32  32   33   34   35   36   36   34   34   34   34
39   38   39  39   39   41   42   43   40   39   38   38   40
38   39   39  39   39   41   41   41   28   40   39   39   45
23   23   27  43   24   25   26   26   26   26   25   26   25
26   27   28  28   28   30   31   31   31   30   30   30   29
31   32   32  32   33   34   35   36   36   34   34   34   34
39   38   39  39   39   41   42   43   40   39   38   38   40
38   39   39  39   39   41   41   41   28   40   39   39   45
23   23   27  43   24   25   26   26   26   26   25   26   25
26   27   28  28   28   30   31   31   31   30   30   30   29
31   32   32  32   33   34   35   36   36   34   34   34   34
39   38   39  39   39   41   42   43   40   39   38   38   40
38   39   39  39   39   41   41   41   28   40   39   39   45
23   23   27  43   24   25   26   26   26   26   25   26   25
26   27   28  28   28   30   31   31   31   30   30   30   29
31   32   32  32   33   34   35   36   36   34   34   34   34
39   38   39  39   39   41   42   43   40   39   38   38   40
38   39   39  39   39   41   41   41   28   40   39   39   45
```



5. Run MapReduceClient.jar and also provide input and out directories.

hadoop jar C:/MapReduceClient.jar wordcount /input_dir /output_dir

Removing Outlines of Boxplot in R

```
Administrator: Command Prompt
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1999
                HDFS: Number of bytes written=120
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=2180
                Total time spent by all reduces in occupied slots (ms)=2442
                Total time spent by all map tasks (ms)=2180
                Total time spent by all reduce tasks (ms)=2442
                Total vcore-milliseconds taken by all map tasks=2180
                Total vcore-milliseconds taken by all reduce tasks=2442
                Total megabyte-milliseconds taken by all map tasks=2232320
                Total megabyte-milliseconds taken by all reduce tasks=2500608
        Map-Reduce Framework
                Map input records=30
                Map output records=390
                Map output bytes=2730
                Map output materialized bytes=195
                Input split bytes=111
                Combine input records=390
                Combine output records=21
                Reduce input groups=21
                Reduce shuffle bytes=195
                Reduce input records=21
                Reduce output records=21
                Spilled Records=42
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=70
                CPU time spent (ms)=764
                Physical memory (bytes) snapshot=471478272
                Virtual memory (bytes) snapshot=619429888
                Total committed heap usage (bytes)=353894400
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1888
        File Output Format Counters
                Bytes Written=120

C:\>
```

6. Verify content for generated output file.

hadoop dfs -cat /output_dir/*

```
C:\>hadoop dfs -cat /output_dir/*
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
23      12
24      6
25      18
26      36
27      12
28      24
29      6
30      24
31      24
32      18
33      6
34      30
35      6
36      12
38      24
39      66
40      18
41      24
42      6
43      12
45      6

C:\>
```

Some Other useful commands

8) To leave Safe mode

hadoop dfsadmin –safemode leave

9) To delete file from HDFS directory

hadoop fs -rm -r /iutput_dir/input_file.txt

10) To delete directory from HDFS directory

hadoop fs -

rm -r

/iutput_dir

```
C:\>hadoop dfsadmin -safemode leave
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
Safe mode is OFF

C:\>hadoop fs -rm -r /input_dir/input_file.txt
Deleted /input_dir/input_file.txt



C:\>hadoop fs -rm -r /input_dir
Deleted /input_dir

C:\>
```

Result:

Thus the word count programs using Map Reduce successfully Executed and verified.

Ex.No:8 **Implement an application that stores big data in Hbase /MongoDB / Pig using Hadoop / R/Cassandra**

Date:

Aim:

To Implement an application that stores big data in Hbase /MongoDB / Pig using Hadoop / R/Cassandra

MongoDB with R

1) To use MongoDB with R, first, we have to download and install MongoDBNext, start MongoDB. We can start MongoDB like so:

mongod

2) Inserting data

Let's insert the crimes data from data.gov to MongoDB. The dataset reflects reported incidents of crime (with the exception of murders where data exists for eachvictim) that occurred in the City of Chicago since 2001.

library (ggplot2) library (dplyr)

library (maps) library (ggmap)

library (mongolite)library

(lubridate) library (gridExtra)

crimes=data.table::fread("Crimes_2001_to_present.csv")names

(crimes)

Output:

ID' 'Case Number' 'Date' 'Block' 'IUCR' 'Primary Type' 'Description' 'Location Description' 'Arrest''Domestic' 'Beat' 'District' 'Ward' 'Community Area' 'FBI Code' 'XCoordinate' 'Y Coordinate' 'Year' 'Updated On' 'Latitude' 'Longitude' 'Location'

1) Let’s remove spaces in the column names to avoid any problems when we query itfrom MongoDB.

```
names(crimes) = gsub(" ","",names(crimes))

names(crimes)
```

'ID' 'CaseNumber' 'Date' 'Block' 'IUCR' 'PrimaryType' 'Description 'LocationDescription' 'Arrest' 'Domestic' 'Beat' 'District' 'Ward' 'CommunityArea 'FBICode' 'XCoordinate' 'YCoordinate' 'Year' 'UpdatedOn' 'Latitude' 'Longitude 'Location'

2) Let’s use the insert function from the mongolite package to insert rows to a collectionin MongoDB.Let’s create a database called Chicago and call the collection crimes.

```
my_collection = mongo(collection = "crimes", db = "Chicago") # createconnection, database and collection
```

my_collection$insert(crimes)

1) Let’s check if we have inserted the “crimes” data.

```
my_collection$count()
```

6261148

We see that the collection has 6261148 records.

2) First, let’s look what the data looks like by displaying one record:

my_collection$iterate()$one()

```
$ID

1454164

$Case Number

 G185744'

$Date

 04/01/2001 06:00:00 PM'

$Block
```

 0910'

$Primary Type

 MOTOR VEHICLE THEFT'

$Description

 AUTOMOBILE'

$Location DescriptionSTREET'

$Arrestfalse'

$Domesticfalse'

$Beat1622

$District16

$FBICode07'

$XCoordinate1136545

YCoordinate1932203

$Year2001

$Updated On

08/17/2015 03:03:40 PM'

$Latitude 41.970129962

$Longitude 87.773302309

$Location

(41.970129962, -87.773302309)'

1) How many distinct “Primary Type” do we have?

length(my_collection$distinct("PrimaryType"))

35

As shown above, there are 35 different crime primary types in the database. We willsee the patterns of the most common crime types below.

1) Now, let’s see how many domestic assaults there are in the collection.

my_collection$count('{"PrimaryType":"ASSAULT", "Domestic" : "true" }')

82470

2) To get the filtered data and we can also retrieve only the columns of interest.

query1= my_collection$find('{"PrimaryType" : "ASSAULT", "Domestic" :"true" }')

query2= my_collection$find('{"PrimaryType" : "ASSAULT", "Domestic" :"true" }',

fields = '{"_id":0, "PrimaryType":1, "Domestic":1}')ncol(query1)

 # with all the columns

ncol(query2) # only the selected columns

10)We can explore any patterns of domestic crimes. For example, are they common incertain days/hours/months?

domestic=my_collection$find('{"Domestic":"true"}', fields = '{"_id":0, "Domestic":1,"Date":1}')

domestic$Date= mdy_hms(domestic$Date) domestic$Weekday =

weekdays(domestic$Date) domestic$Hour =

hour(domestic$Date) domestic$month =

month(domestic$Date,label=TRUE)

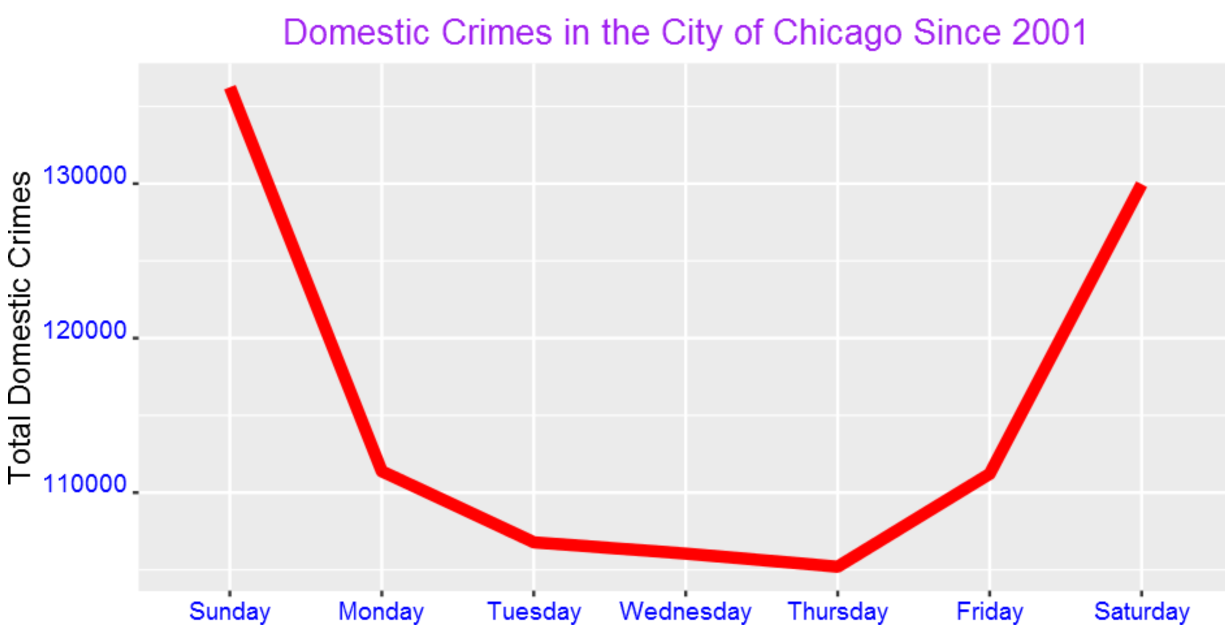
WeekdayCounts = as.data.frame(table(domestic$Weekday))

WeekdayCounts$Var1 = factor(WeekdayCounts$Var1, ordered=TRUE, levels=c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday","Saturday"))

ggplot(WeekdayCounts, aes(x=Var1, y=Freq)) + geom_line(aes(group=1),size=2,color="red") + xlab("Day of the Week") + ylab("Total Domestic Crimes")+

ggtitle("Domestic Crimes in the City of Chicago Since 2001")+

```
theme(axis.title.x=element_blank(),axis.text.y =
element_text(color="blue",size=11,angle=0,hjust=1,vjust=0),
axis.text.x = element_text(color="blue",size=11,angle=0,hjust=.5,vjust=.5),
axis.title.y = element_text(size=14),
plot.title=element_text(size=16,color="purple",hjust=0.5))
```



1) Domestic crimes are common over the weekend than in weekdays? What could bethe reason?

We can also see the pattern for each day by hour:

```
DayHourCounts = as.data.frame(table(domestic$Weekday, domestic$Hour))
DayHourCounts$Hour = as.numeric(as.character(DayHourCounts$Var2))
ggplot(DayHourCounts, aes(x=Hour, y=Freq)) + geom_line(aes(group=Var1,color=Var1),
size=1.4)+ylab("Count")+
ylab("Total Domestic Crimes")+ggtitle("Domestic Crimes in the City of ChicagoSince 2001")+
```

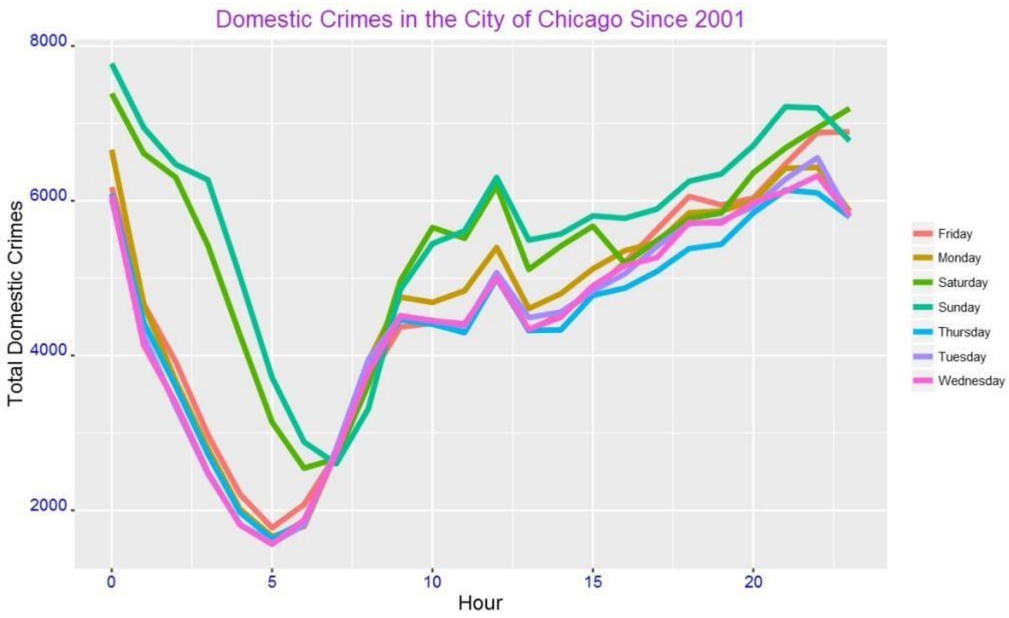
theme(axis.title.x=element_text(size=14),axis.text.y =

element_text(color="blue",size=11,angle=0,hjust=1,vjust=0),

axis.text.x = element_text(color="blue",size=11,angle=0,hjust=.5,vjust=.5),axis.title.y =

element_text(size=14),

legend.title=element_blank(), plot.title=element_text(size=16,color="purple",hjust=0.5))



11)The crimes peak mainly around mid-night. We can also use one color forweekdays and another color for weekend as shown below.

DayHourCounts$Type = ifelse((DayHourCounts$Var1 == "Sunday") |

(DayHourCounts$Var1 == "Saturday"), "Weekend", "Weekday")

ggplot(DayHourCounts, aes(x=Hour, y=Freq)) + geom_line(aes(group=Var1,color=Type), size=2, alpha=0.5) +

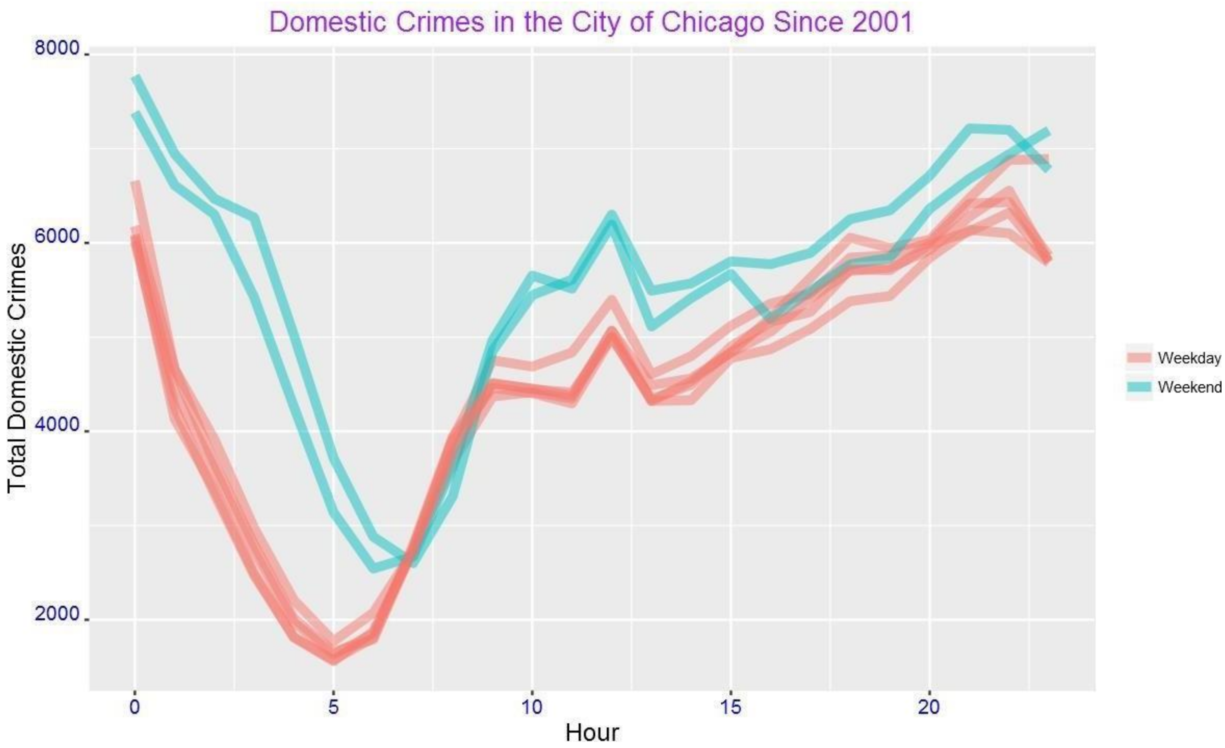
ylab("Total Domestic Crimes")+ggtitle("Domestic Crimes in the City of ChicagoSince 2001")+

theme(axis.title.x=element_text(size=14),axis.text.y =

element_text(color="blue",size=11,angle=0,hjust=1,vjust=0),

axis.text.x = element_text(color="blue",size=11,angle=0,hjust=.5,vjust=.5),axis.title.y = element_text(size=14),

legend.title=element_blank(), plot.title=element_text(size=16,color="purple",hjust=0.5))



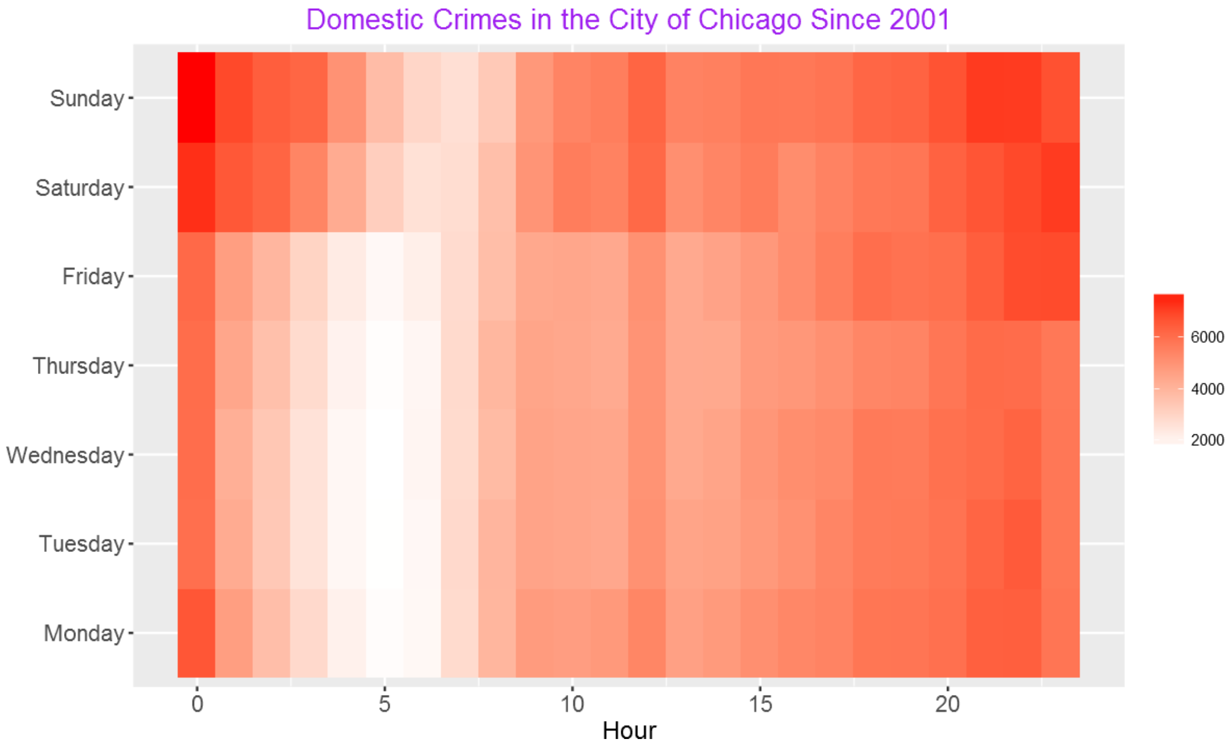
3) The difference between weekend and weekdays are clearer from this figure than from the previous plot. We can also see the above pattern from a heat map.

DayHourCounts$Var1 = factor(DayHourCounts$Var1, ordered=TRUE, levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))

ggplot(DayHourCounts, aes(x = Hour, y = Var1)) + geom_tile(aes(fill = Freq)) + scale_fill_gradient(name="Total MV Thefts", low="white", high="red") +

ggtitle("Domestic Crimes in the City of Chicago Since 2001")+theme(axis.title.y = element_blank())+ylab("")+theme(axis.title.x=element_text(size=14),axis.text.y = element_text(size=13),axis.text.x = element_text(size=13), axis.title.y = element_text(size=14),

legend.title=element_blank(),plot.title=element_text(size=16,color="purple",hjus t=0.5))



1) Let's see the pattern of other crime types. Let's focus on four of the most commonones.

```
crimes=my_collection$find('{}', fields = '{"_id":0, "PrimaryType":1,"Year":1}')
```

```
crimes%>%group_by(PrimaryType)%>%summarize(Count=n())%>%arrange
(desc(Count))%>%head(4)
```

THEFT	**1301434**
BATTERY	**1142377**
CRIMINAL DAMAGE	**720143**
NARCOTICS	**687790**

12) As shown in the table above, the most common crime type is theft followed by battery. Narcotics is fourth most common while criminal damage is the third most common crime type in the city of Chicago. Now, let's generate plots by day and hour.

```
four_most_common=crimes%>%group_by(PrimaryType)%>%summarize(Cou
nt=n())%>%arrange(desc(Count))%>%head(4)

four_most_common=four_most_common$PrimaryType crimes=my_collection$find('{}', fields =
'{"_id":0, "PrimaryType":1,"Date":1}')crimes=filter(crimes,PrimaryType
%in%four_most_common)

crimes$Date= mdy_hms(crimes$Date) crimes$Weekday =
weekdays(crimes$Date) crimes$Hour = hour(crimes$Date)
crimes$month=month(crimes$Date,label = TRUE)

g = function(data){WeekdayCounts = as.data.frame(table(data$Weekday))

WeekdayCounts$Var1 = factor(WeekdayCounts$Var1, ordered=TRUE, levels=c("Sunday", "Monday",
"Tuesday", "Wednesday", "Thursday", "Friday","Saturday"))

ggplot(WeekdayCounts, aes(x=Var1, y=Freq)) +
geom_line(aes(group=1),size=2,color="red") + xlab("Day of the Week") +
theme(axis.title.x=element_blank(),axis.text.y =
```

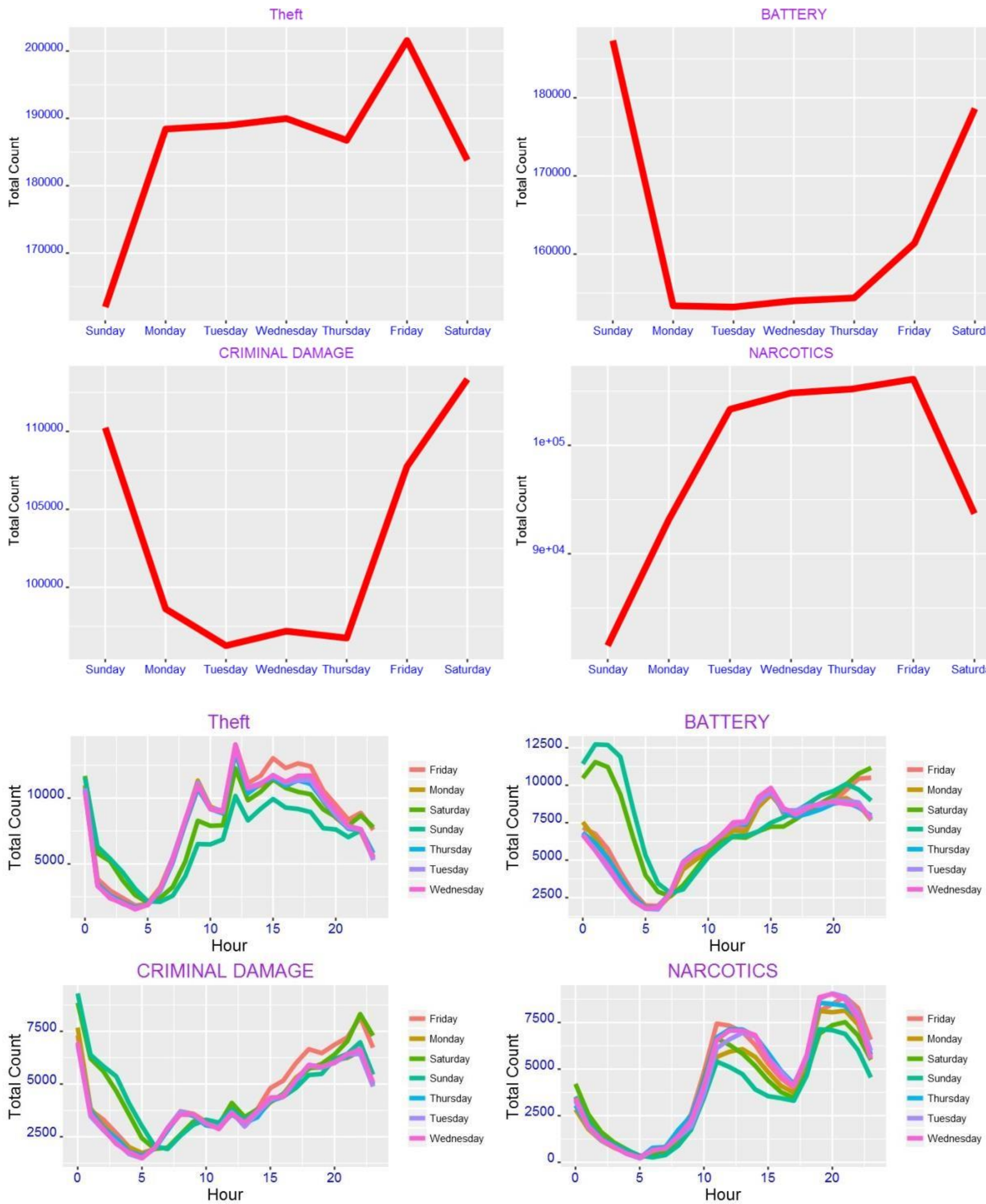
```
element_text(color="blue",size=10,angle=0,hjust=1,vjust=0),
 axis.text.x = element_text(color="blue",size=10,angle=0,hjust=.5,vjust=.5),axis.title.y = element_text(size=11),
plot.title=element_text(size=12,color="purple",hjust=0.5)) }
g1=g(filter(crimes,PrimaryType=="THEFT"))+ggtitle("Theft")+ylab("Total Count")
	g2=g(filter(crimes,PrimaryType=="BATTERY"))+ggtitle("BATTERY")+ylab(" Total Count")
g3=g(filter(crimes,PrimaryType=="CRIMINAL DAMAGE"))+ggtitle("CRIMINAL
DAMAGE")+ylab("Total Count")
	g4=g(filter(crimes,PrimaryType=="NARCOTICS"))+ggtitle("NARCOTICS")+ylab("Total Count")
grid.arrange(g1,g2,g3,g4,ncol=2)
```

From the plots above, we see that theft is most common on Friday. Battery and criminal damage, on the other hand, are highest at weekend. We also observe thatnarcotics decreases over weekend.

We can also see the pattern of the above four crime types by hour:

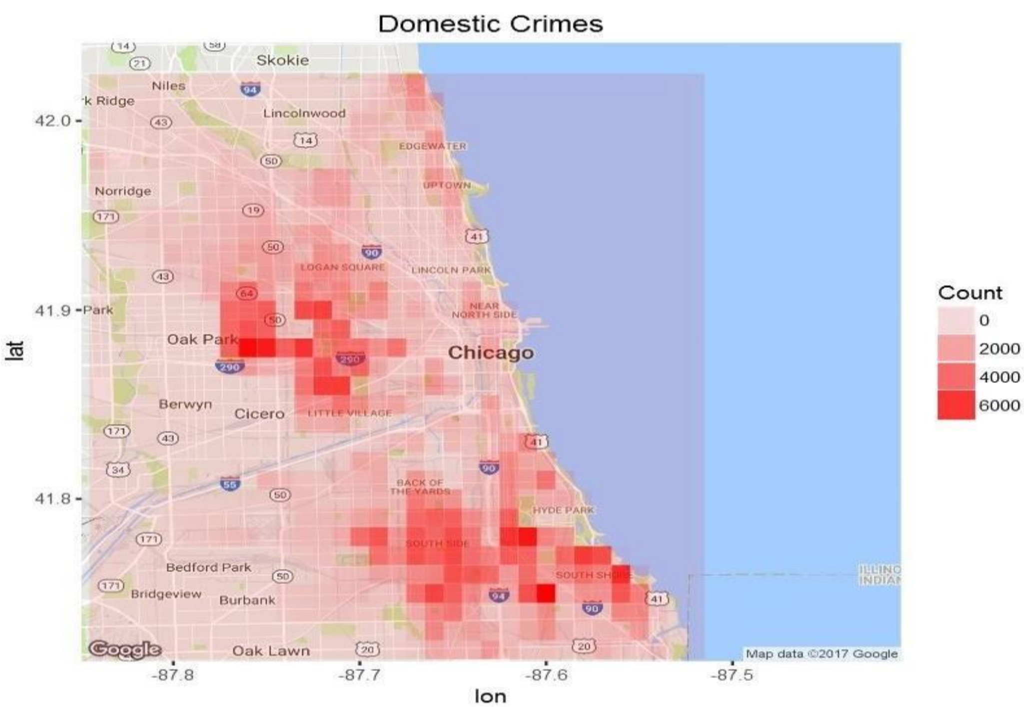
From the plots above, we see that theft is most common on Friday. Battery and criminal damage, on the other hand, are highest at weekend. We also observe thatnarcotics decreases over weekend.

We can also see the pattern of the above four crime types by hour:

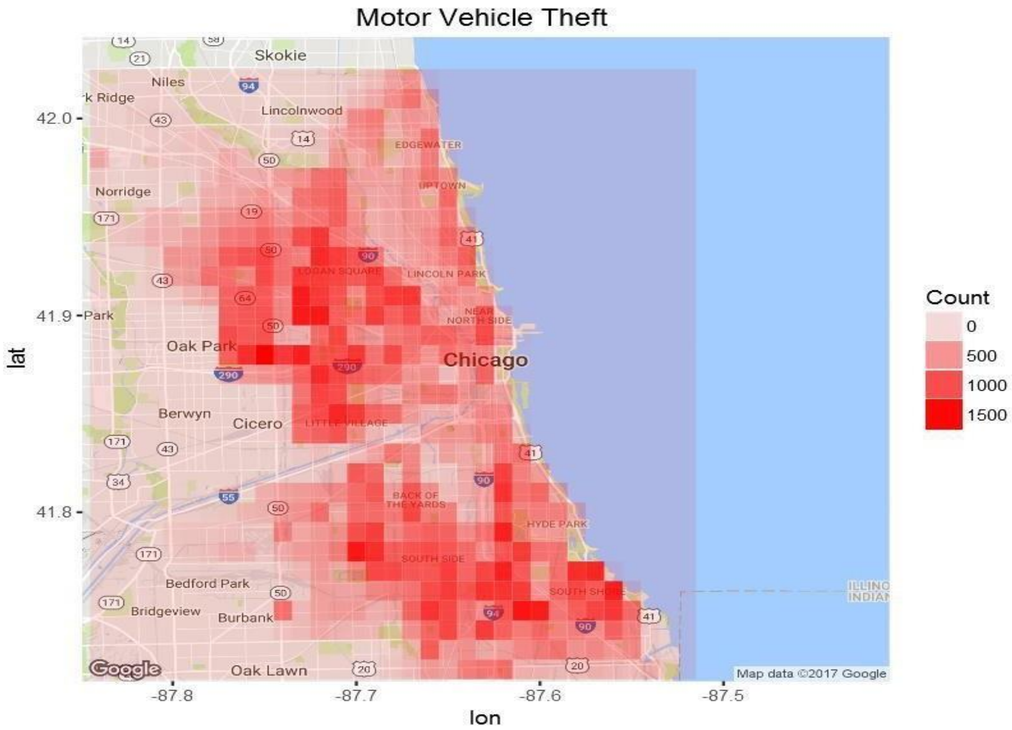

Theft
Total Count
200000
190000
180000
170000
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
BATTERY
Total Count
180000
170000
160000
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
CRIMINAL DAMAGE
Total Count
110000
105000
100000
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
NARCOTICS
Total Count
1e+05
9e+04
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
Theft
Total Count
10000
5000
0 5 10 15 20
Hour
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
BATTERY
Total Count
12500
10000
7500
5000
2500
0 5 10 15 20
Hour
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
CRIMINAL DAMAGE
Total Count
7500
5000
2500
0 5 10 15 20
Hour
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
NARCOTICS
Total Count
7500
5000
2500
0
0 5 10 15 20
Hour
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday

13) We can also see a map for domestic crimes only:

```
domestic=my_collection$find('{"Domestic":"true"}', fields = '{"_id":0,"Latitude":1,
"Longitude":1,"Year":1}')

LatLonCounts=as.data.frame(table(round(domestic$Longitude,2),round(domest ic$Latitude,2)))

LatLonCounts$Long = as.numeric(as.character(LatLonCounts$Var1))LatLonCounts$Lat =
as.numeric(as.character(LatLonCounts$Var2))

ggmap(chicago) + geom_tile(data = LatLonCounts, aes(x = Long, y = Lat, alpha
= Freq), fill="red")+
ggtitle("Domestic Crimes")+labs(alpha="Count")+theme(plot.title =element_text(hjust=0.5))
```



14)Let’s see where motor vehicle theft is common:



Domestic crimes show concentration over two areas whereas motor vehicle theft iswide spread over large part of the city of Chicago.

Result:

Thus the Implement an application that stores big data in Hbase /MongoDB / Pig using Hadoop / R/Cassandra has been executed successfully

Exp No: 09 **Using Apache Spark for Data Analytics**

Date:

Aim:

To Write a program to use the apache spark for Data Analytics

Program:

Creating Spark Session

For this, we need to set up the spark in our system and after we log into the Spark console, the following packages need to be imported to perform the examples.

```
from pyspark.sql import SparkSession
from pyspark.sql.types  import  *
from pyspark.sql.functions import *
from pyspark.sql.types import Row
from datetime import datetime
```

After the necessary imports, we have to initialize the spark session by the following command:

```
spark = SparkSession.builder.appName("Python Spark SQL basic
example").config("spark.some.config.option", "some-value").getOrCreate()
```

Then we will create a Spark RDD using the parallelize function. This RDD contains two rows for two students and the values are self-explanatory.

```
student_records = sc.parallelize
([Row(roll_no=1,name='JohnDoe',passed=True,
marks={'Math':89,'Physics':87,'Chemistry':81},
sports =['chess','football'], DoB=datetime(2012,5,1,13,1,5)),
Row(roll_no=2,name='John Smith',passed=False,
marks={'Math':29,'Physics':31,'Chemistry':36},
sports =['volleyball','tabletennis'], DoB=datetime(2012,5,12,14,2,5))])
```

Creating DataFrame

Let’s create a DataFrame from this RDD and show the resulting DataFrame by following the command.

```
student_records_df = student_records.toDF()
student_records_df.show()
```

```
>>> student_records_df.show()
+-------+----------+------+--------------------+--------------------+-------------------+
|roll_no|      name|passed|               marks|              sports|                DoB|
+-------+----------+------+--------------------+--------------------+-------------------+
|      1|  John Doe|  true|[Chemistry -> 81,...|   [chess, football]|2012-05-01 13:01:05|
|      2|John Smith| false|[Chemistry -> 36,...|[volleyball, tabl...|2012-05-12 14:02:05|
+-------+----------+------+--------------------+--------------------+-------------------+
```

Now, as we can see the content of column ‘marks’ has been truncated. To view the full content we can run the following command:

```
>>> student_records_df.show(truncate=False)
+-------+----------+------+--------------------------------------------+-------------------------+-------------------+
|roll_no|name      |passed|marks                                       |sports                   |DoB                |
+-------+----------+------+--------------------------------------------+-------------------------+-------------------+
|1      |John Doe  |true  |[Chemistry -> 81, Math -> 89, Physics -> 87]|[chess, football]        |2012-05-01 13:01:05|
|2      |John Smith|false |[Chemistry -> 36, Math -> 29, Physics -> 31]|[volleyball, tabletennis]|2012-05-12 14:02:05|
+-------+----------+------+--------------------------------------------+-------------------------+-------------------+
```

Creating Temporary View

The above DataFrame can be treated as a relational table. For that, by using the following command we can create a relational view named ‘records’ which is valid for the created spark session.

```
student_records_df.createOrReplaceTempView('records')
```

It is time for us to now run a SQL query against this view and show the results.

```
spark.sql("SELECT * FROM records").show()
```

```
>>> spark.sql("SELECT * FROM records").show()
+-------+----------+------+--------------------+--------------------+-------------------+
|roll_no|      name|passed|               marks|              sports|                DoB|
+-------+----------+------+--------------------+--------------------+-------------------+
|      1|  John Doe|  true|[Chemistry -> 81,...|   [chess, football]|2012-05-01 13:01:05|
|      2|John Smith| false|[Chemistry -> 36,...|[volleyball, tabl...|2012-05-12 14:02:05|
+-------+----------+------+--------------------+--------------------+-------------------+
```

Here we can verify that the spark.sql returns Spark DataFrame.

```
>>> result = spark.sql("SELECT * FROM records")
>>> type(result)
<class 'pyspark.sql.dataframe.DataFrame'>
```

Accessing Elements of List or Dictionary within DataFrame

```
spark.sql('SELECT roll_no, marks["Physics"], sports[1] FROM
records').show()
```

```
>>> spark.sql('SELECT roll_no, marks["Physics"], sports[1] FROM records').show()
+-------+-------------+-----------+
|roll_no|marks[Physics]|  sports[1]|
+-------+-------------+-----------+
|      1|           87|   football|
|      2|           31|tabletennis|
+-------+-------------+-----------+
```

Where Clause

```
spark.sql("SELECT * FROM records where passed = True").show()
```

```
>>> spark.sql("SELECT * FROM records where passed = True").show()
+-------+--------+------+--------------------+-----------------+-------------------+
|roll_no|    name|passed|               marks|           sports|                DoB|
+-------+--------+------+--------------------+-----------------+-------------------+
|      1|John Doe|  true|[Chemistry -> 81,...|[chess, football]|2012-05-01 13:01:05|
+-------+--------+------+--------------------+-----------------+-------------------+
```

In the above example, we have selected the row for which the 'passed' column has the boolean value True.

We can write where clause using the values from the data structure field also. In the following example, we are using the key ‘Chemistry’ from the marks dictionary.

```
spark.sql('SELECT * FROM records WHERE marks["Chemistry"] < 40').show()
```

```
>>> spark.sql('SELECT * FROM records WHERE marks["Chemistry"] < 40').show()
+-------+----------+------+--------------------+--------------------+-------------------+
|roll_no|      name|passed|               marks|              sports|                DoB|
+-------+----------+------+--------------------+--------------------+-------------------+
|      2|John Smith| false|[Chemistry -> 36,...|[volleyball, tabl...|2012-05-12 14:02:05|
+-------+----------+------+--------------------+--------------------+-------------------+
```

Creating Global View

The view ‘records’ we have created above has the scope only for the current session. Once the session disappears, the view will be terminated, and it will not be accessible. However, if we want other sessions which were initiated in the same application to be able to access the view even if the session that created the view ends, then we make a global view by using the following command:

```
student_records_df.createGlobalTempView('global_record')
```

The scope of this view will be at the application level rather than the session-level. Now, let’s run a select query on this global view:

```
spark.sql("SELECT * FROM global_temp.global_records").show()
```

```
>>> spark.sql("SELECT * FROM global_temp.global_records").show()
+-------+----------+------+--------------------+--------------------+-------------------+
|roll_no|      name|passed|               marks|              sports|                DoB|
+-------+----------+------+--------------------+--------------------+-------------------+
|      1|  John Doe|  true|[Chemistry -> 81,...|   [chess, football]|2012-05-01 13:01:05|
|      2|John Smith| false|[Chemistry -> 36,...|[volleyball, tabl...|2012-05-12 14:02:05|
+-------+----------+------+--------------------+--------------------+-------------------+
```

All the global views are preserved in the database called: global_temp.

Dropping Columns from DataFrame

If we want to see only the columns of our DataFrame, we can use the following command:

```
student_records_df.columns
```

```
>>> student_records_df.columns
['roll_no', 'name', 'passed', 'marks', 'sports', 'DoB']
```

If we want to drop any column, then we can use the drop command. In our dataset, let's try to drop the 'passed' column.

```
student_records_df = student_records_df.drop('passed')
```

```
>>> student_records_df = student_records_df.drop('passed')
>>> student_records_df.columns
['roll_no', 'name', 'marks', 'sports', 'DoB']
```

Now, we can see that we don't have the column 'passed' anymore in our DataFrame.

Few More Queries

Let's create a column that shows the average marks for each student:

```
spark.sql("SELECT round( (marks.Physics+marks.Chemistry+marks.Math)/3)
avg_marks FROM records").show()
```

```
>>> spark.sql("SELECT round( (marks.Physics+marks.Chemistry+marks.Math)/3) avg_marks FROM records").show()
+---------+
|avg_marks|
+---------+
|     86.0|
|     32.0|
+---------+
```

Now, we will add this column to our existing DataFrame.

```
student_records_df=spark.sql("SELECT *, round(
(marks.Physics+marks.Chemistry+marks.Math)/3) avg_marks FROM records")

student_records_df.show()
```

```
>>> student_records_df=spark.sql("SELECT *, round( (marks.Physics+marks.Chemistry+marks.Math)/3) avg_marks FROM records"
)
>>> student_records_df.show()
+-------+----------+--------------------+--------------------+-------------------+---------+
|roll_no|      name|               marks|              sports|                DoB|avg_marks|
+-------+----------+--------------------+--------------------+-------------------+---------+
|      1|  John Doe|[Chemistry -> 81,...|   [chess, football]|2012-05-01 13:01:05|     86.0|
|      2|John Smith|[Chemistry -> 36,...|[volleyball, tabl...|2012-05-12 14:02:05|     32.0|
+-------+----------+--------------------+--------------------+-------------------+---------+
```

We had dropped the column 'passed' earlier. We can derive a new column named 'status', where we will put the status 'passed' or 'failed' after calculating the average marks and we will check if the average marks are greater than 40 percent.

To perform that, first, we must update the view again.

```
student_records_df.createOrReplaceTempView('records')
```

We can achieve this by the following query:

```
student_records_df =
student_records_df.withColumn('status',(when(col('avg_marks')>=40,
'passed')).otherwise('failed'))student_records_df.show()
```

```
>>> student_records_df  = student_records_df.withColumn('status',(when(col('avg_marks')>=40, 'passed')).otherwise('faile
d'))
>>> student_records_df.show()
+-------+----------+--------------------+--------------------+-------------------+---------+------+
|roll_no|      name|               marks|              sports|                DoB|avg_marks|status|
+-------+----------+--------------------+--------------------+-------------------+---------+------+
|      1|  John Doe|[Chemistry -> 81,...|   [chess, football]|2012-05-01 13:01:05|     86.0|passed|
|      2|John Smith|[Chemistry -> 36,...|[volleyball, tabl...|2012-05-12 14:02:05|     32.0|failed|
+-------+----------+--------------------+--------------------+-------------------+---------+------+
```

Group by and Aggregation

Let's look into some more functionalities of Spark SQL. For that, we have to take a new DataFrame. Let's create a new DataFrame with employee records.

```
employeeData =(('John','HR','NY',90000,34,10000),
('Neha','HR','NY',86000,28,20000),
('Robert','Sales','CA',81000,56,22000),
('Maria','Sales','CA',99000,45,15000),
('Paul','IT','NY',98000,38,14000),
('Jen','IT','CA',90000,34,20000),
('Raj','IT','CA',93000,28,28000),
('Pooja','IT','CA',95000,31,19000))
columns = ('employee_name','department','state','salary','age','bonus')
employeeDf = spark.createDataFrame(employeeData, columns)
```

```
>>> employeeDf.show()
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|         John|        HR|   NY| 90000| 34|10000|
|         Neha|        HR|   NY| 86000| 28|20000|
|       Robert|     Sales|   CA| 81000| 56|22000|
|        Maria|     Sales|   CA| 99000| 45|15000|
|         Paul|        IT|   NY| 98000| 38|14000|
|          Jen|        IT|   CA| 90000| 34|20000|
|          Raj|        IT|   CA| 93000| 28|28000|
|        Pooja|        IT|   CA| 95000| 31|19000|
+-------------+----------+-----+------+---+-----+
```

If we wish to query the department wise total salary, we can achieve that in the following way:

employeeDf.groupby(col('department')).agg(sum(col('salary'))).show()

```
>>> employeeDf.groupby(col('department')).agg(sum(col('salary'))).show()
+----------+-----------+
|department|sum(salary)|
+----------+-----------+
|     Sales|     180000|
|        HR|     176000|
|        IT|     376000|
+----------+-----------+
```

The result shows the department-wise total salary. If we want to see the total salary in an ordered way we can achieve by following way.

employeeDf.groupby(col('department')).agg(sum(col('salary')).alias ('total_sal')).orderBy('total_sal').show()

```
>>> employeeDf.groupby(col('department')).agg(sum(col('salary')).alias('total_sal')).orderBy('total_sal').show()
+----------+---------+
|department|total_sal|
+----------+---------+
|        HR|   176000|
|     Sales|   180000|
|        IT|   376000|
+----------+---------+
```

Windowing in Spark

```
from pyspark.sql.window import Window
windowSpec =Window.partitionBy("department").orderBy(col("salary")
.desc())employeeDf = employeeDf.withColumn("rank",dense_rank()
.over(windowSpec))employeeDf.filter(col('rank') == 2).show()
```

```
>>> from pyspark.sql.window import Window
>>> windowSpec = Window.partitionBy("department").orderBy(col("salary").desc())
>>> employeeDf = employeeDf.withColumn("rank",dense_rank().over(windowSpec))
>>> employeeDf.filter(col('rank') == 2).show()
+-------------+----------+-----+------+---+-----+----+
|employee_name|department|state|salary|age|bonus|rank|
+-------------+----------+-----+------+---+-----+----+
|       Robert|     Sales|   CA| 81000| 56|22000|   2|
|         Neha|        HR|   NY| 86000| 28|20000|   2|
|        Pooja|        IT|   CA| 95000| 31|19000|   2|
+-------------+----------+-----+------+---+-----+----+
```

Joins in Spark

To perform join let's create another dataset containing managers of each department.

```
managers = (('Sales','Maria'),('HR','John'),('IT','Pooja'))
mg_columns = ('department', 'manager')
managerDf = spark.createDataFrame(managers, mg_columns)
managerDf.show()
```

```
>>> managers = (('Sales','Maria'),('HR','John'),('IT','Pooja'))
>>> mg_columns = ('department', 'manager')
>>> managerDf = spark.createDataFrame(managers, mg_columns)
>>> managerDf.show()
+----------+-------+
|department|manager|
+----------+-------+
|     Sales|  Maria|
|        HR|   John|
|        IT|  Pooja|
+----------+-------+
```

Now, if we want to view the name of managers of each employee, we can run the following command:

```
employeeDf.join(managerDf, employeeDf['department'] ==
managerDf['department'], how='inner').select(col('employee_name'),
col('manager')).show()
```

```
>>> employeeDf.join(managerDf, employeeDf['department']==managerDf['department'], how='inner').select(col('employee_name
'),col('manager')).show()
+-------------+-------+
|employee_name|manager|
+-------------+-------+
|       Robert|  Maria|
|        Maria|  Maria|
|         John|   John|
|         Neha|   John|
|         Paul|  Pooja|
|          Jen|  Pooja|
|          Raj|  Pooja|
|        Pooja|  Pooja|
+-------------+-------+
```

Result:

Thus the program has been written and used for data analytics using apache spark and verified successfully.