Ex.No. : 1 Vectors and List operations in R

Creating a vector

```
X <- c(1, 4, 5, 2, 6, 7)
print('using c function')
print(X)
Y <- seq(1, 10, length.out = 5)
print('using seq() function')
print(Y)
Z <- 5:10
print('using colon')
print(Z)
```

Accessing vector elements

```
X <- c(2, 5, 8, 1, 2)
print('using Subscript operator')
print(X[2])
Y <- c(4, 5, 2, 1, 7)
print('using c function')
print(Y[c(4, 1)])
Z <- c(5, 2, 1, 4, 4, 3)
print('Logical indexing')
print(Z[Z>3])
```

Modifying a vector

```
X <- c(2, 5, 1, 7, 8, 2)

# modify a specific element
X[3] <- 11
print('Using subscript operator')
print(X)

# Modify using different logics.
X[X>9] <- 0
print('Logical indexing')
print(X)

# Modify by specifying the position or elements.
X <- X[c(5, 2, 1)]
print('using c function')
print(X)
```

Deleting a vector

```
Creating a vector
X <- c(5, 2, 1, 6)
# Deleting a vector
X <- NULL
```

```
print('Deleted vector')
print(X)
```

Arithmetic operations

```
#Creating Vectors
X <- c(5, 2, 5, 1, 51, 2)
Y <- c(7, 9, 1, 5, 2, 1)

# Addition
Z <- X + Y
print('Addition')
print(Z)

# Subtraction
S <- X - Y
print('Subtraction')
print(S)

# Multiplication
M <- X * Y
print('Multiplication')
print(M)

# Division
D <- X / Y
print('Division')
print(D)
```

Sorting of Vectors

```
#Creating a Vector
X <- c(5, 2, 5, 1, 51, 2)

# Sort in ascending order
A <- sort(X)
print('sorting done in ascending order')
print(A)

# sort in descending order.
B <- sort(X, decreasing = TRUE)
print('sorting done in descending order')
print(B)
#II Creating List

#Creating Vectors
vec1 <- c(1, 2, 3)
vec2 <- c(TRUE, FALSE)
```

```
# Creating a list of Vectors
listt = list(vec1, vec2)

# Printing List
print (listt)
```

Adding elements to a list

```
# Creating Vectors
vec1 <- c(1, 2, 3)
vec2 <- c(TRUE, FALSE)

# Creating list of Vectors
lst = list(vec1, vec2)

# Creating a new Vector
vec3 <- c(1 + 3i)

# Adding Vector to list
lst[[3]]<- vec3

# Printing List
print (lst)


# determine the length of list
len <- length(lst)

# Creating new Vector
vec3 <- c(0.5, 2 + 2i)

# Using for loop to add elements
for( i in 1:2)
{

   # Adding vec to list
   lst[[len + i]]<- vec3
}
print (lst)
```

Removing elements from a list

```
lst[[2]]<-NULL
print ("Modified List")
print (lst)
```

Modifying elements in a list

```
# Creating Vectors
vec1 <- c(1, 2, 3)
vec2 <- c(TRUE, FALSE)

# Creating list of Vectors
lst = list(vec1, vec2)
print ("original list")
print (lst)

# Modifying List element
lst[[2]]<-c("TEACH", "CODING")
print ("Modified List")
print (lst)
```

Merging two lists

```
# R program to merge two lists of Vectors

# Creating 1st list
list_data1 <- list(c(1:3), c(TRUE, FALSE))

# Creating 2nd list
list_data2 <- list(c(0.1, 3.4))
print("First List")
print (list_data1)
print ("Second List")
print (list_data2)
print("Merged List")

# Merging Lists
merged_list <- c(list_data1, list_data2)
print (merged_list)
```

OUTPUT

#Creating Vector

```
[1] 1 4 5 2 6 7

[1]  1.00  3.25  5.50  7.75 10.00

[1]  5  6  7  8  9 10
```

#Accessing Vector Element

```
[1] 5

[1] 1 4

[1] 5 4 4
```

#Modifying Vector

```
[1]  2  5 11  7  8  2

[1] 2 5 0 7 8 2

[1] 8 5 2
```

#Deleting Vector

```
NULL
```

#Arithmetic Operation

```
[1] "Addition"
[1] 12 11  6  6 53  3

[1] "Subtraction"
[1] -2 -7  4 -4 49  1

[1] "Multiplication"
[1]  35  18   5   5 102   2

[1] "Division"
[1]  0.7142857  0.2222222  5.0000000  0.2000000 25.5000000  2.0000000
```

#Sorting Value

```
[1] "sorting done in ascending order"
[1]  1  2  2  5  5 51

[1] "sorting done in descending order"
[1] 51  5  5  2  2  1
```

#Adding element to the list

```
[[1]]
[1] 1 2 3

[[2]]
[1]  TRUE FALSE

[[3]]
[1] 1+3i
```

[[4]]
[1] 0.5+0i 2.0+2i

[[5]]
[1] 0.5+0i 2.0+2i

#Modifying element in the list

[1] "Modified List"
[[1]]
[1] 1 2 3

[[2]]
[1] "TEACH" "CODING"

#Merging Two List

```
[1] "First List"
[[1]]
[1] 1 2 3

[[2]]
[1]  TRUE FALSE

[1] "Second List"
[[1]]
[1] 0.1 3.4

[1] "Merged List"
[[1]]
[1] 1 2 3

[[2]]
[1]  TRUE FALSE

[[3]]
[1] 0.1 3.4
```

Ex.No. : 2 Matrices and Array operations in R

#Creation of an Array

```
vector1 <- c(1, 2, 3)
vector2 <- c(10, 15, 3, 11, 16, 12)
result <- array(c(vector1, vector2), dim = c(3, 3, 2))
print(result)
```

Operations on Arrays

Naming columns and rows

```
vector1 <- c(1, 2, 3)
vector2 <- c(10, 15, 3, 11, 16, 12)
column.names <- c("COL1", "COL2", "COL3")
row.names <- c("R1", "R2", "R3")
matrix.names <- c("Matrix.NO1", "Matrix.NO2")
result <- array(c(vector1, vector2), dim = c(3, 3, 2),
                dimnames = list(row.names, column.names, matrix.names))
print(result)
```

Manipulating array elements

```
vector1 <- c(1, 2, 3)
vector2 <- c(4, 6, 8, 0, 2, 4)
array1 <- array(c(vector1, vector2), dim = c(3, 3, 2))
vector3 <- c(3, 2, 1)
vector4 <- c(2, 4, 6, 8, 3, 5)
 array2 <- array(c(vector3, vector4), dim = c(3, 3, 2))
matrix1 <- array1[,,2]
matrix2 <- array2[,,2]
result <- matrix1 + matrix2
print(result)
result<-matrix1-matrix2
print(result)
result<-matrix1*matrix2
result<-matrix1/matrix2
```

Accessing Array elements

```
vector1 <- c(1, 2, 3)
vector2 <- c(10, 15, 3, 11, 16, 12)
column.names <- c("COLUMN1", "COLUMN2", "COLUMN3")
row.names <- c("ROW1", "ROW2", "ROW3")
matrix.names <- c("Matrix.NO1", "Matrix.NO2")
# taking vector as input
result <- array(c(vector1, vector2), dim = c(3, 3, 2),
                dimnames = list(row.names, column.names, matrix.names))
print(result)
```

```
# print third row of second matrix
print(result[3,,2])
```

Calculation across array element

```
vector1 <- c(3, 2, 1)
vector2 <- c(2, 4, 6, 8, 0, 1)
new.array <- array(c(vector1, vector2), dim = c(3, 3, 2))
print(new.array)

# using apply and calculate the sum of rows in matrices
result <- apply(new.array, c(1), sum)
print(result)

#Creation of Matrices:
# Elements are arranged sequentially by row.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)

# Elements are arranged sequentially by column.
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)

# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```

Operations on Matrices

Matrices Addition

```
# Creating 1st Matrix
B = matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

# Creating 2nd Matrix
C = matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)

# Getting number of rows and columns
num_of_rows = nrow(B)
num_of_cols = ncol(B)

# Creating matrix to store results
```

```
sum = matrix(, nrow = num_of_rows, ncol = num_of_cols)

# Printing Original matrices
print(B)
print(C)

# Creating 1st Matrix
B = matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

# Creating 2nd Matrix
C = matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)

# Getting number of rows and columns
num_of_rows = nrow(B)
num_of_cols = ncol(B)

# Creating matrix to store results
diff = matrix(, nrow = num_of_rows, ncol = num_of_cols)

# Printing Original matrices
print(B)
print(C)

# Calculating diff of matrices
for(row in 1:num_of_rows)
{
   for(col in 1:num_of_cols)
   {
      diff[row, col] <- B[row, col] - C[row, col]
   }
}

# Printing resultant matrix
print(diff)
# Calculating product of matrices
for(row in 1:num_of_rows)
{
   for(col in 1:num_of_cols)
   {
      prod[row, col] <- B[row, col] * C[row, col]
   }
}

# Printing resultant matrix
print(prod)


# R program for matrix multiplication
```

```
# using '*' operator

# Creating 1st Matrix
B = matrix(c(1, 2 + 3i, 5.4), nrow = 1, ncol = 3)

# Creating 2nd Matrix
C = matrix(c(2, 1i, 0.1), nrow = 1, ncol = 3)

# Printing the resultant matrix
print (B * C)
```

Matrices Division

```
# Creating 1st Matrix
B = matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

# Creating 2nd Matrix
C = matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)

# Getting number of rows and columns
num_of_rows = nrow(B)
num_of_cols = ncol(B)

# Creating matrix to store results
div = matrix(, nrow = num_of_rows, ncol = num_of_cols)

# Printing Original matrices
print(B)
print(C)

# Calculating product of matrices
for(row in 1:num_of_rows)
{
   for(col in 1:num_of_cols)
   {
      div[row, col] <- B[row, col] / C[row, col]
   }
}
# Printing resultant matrix
print(div)
# Creating 1st Matrix
B = matrix(c(4, 6i, -1), nrow = 1, ncol = 3)
# Creating 2nd Matrix
C = matrix(c(2, 2i, 0), nrow = 1, ncol = 3)
# Printing the resultant matrix
print (B / C)
```

OUTPUT

#Naming Column and Rows

Matrix.NO1

```
   COL1 COL2 COL3
R1    1   10   11
R2    2   15   16
R3    3    3   12
```

, , Matrix.NO2

```
   COL1 COL2 COL3
R1    1   10   11
R2    2   15   16
R3    3    3   12
```

#Manipulating array Elements

```
   [,1] [,2] [,3]
[1,]    4    6    8
[2,]    4   10    5
[3,]    4   14    9

     [,1] [,2] [,3]
[1,]   -2    2   -8
[2,]    0    2   -1
[3,]    2    2   -1
```

#Accessing Array Elements

print third row of second matrix

```
COLUMN1 COLUMN2 COLUMN3
      3       3      12
```

#Calculation across array elements

using apply and calculate the sum of rows in matrices

[1] 26 12 16

#Matrices Operation

#Addition

```
[,1] [,2] [,3]
[1,]    8   12   16
[2,]   10   14   18
```

#Subtraction

```
[,1] [,2] [,3]
[1,]    6    6    6
[2,]    6    6    6
```

#Multiplication

```
  [,1] [,2] [,3]
[1,]    7   27   55
```

```
[2,]   16   40   72
```

#Division

```
        [,1]      [,2]      [,3]
[1,] 0.1428571 0.3333333 0.4545455
[2,] 0.2500000 0.4000000 0.5000000
```

```
[,1] [,2]      [,3]
[1,] 2+0i 3+0i -Inf+NaNi
```

Ex.No. : 3 Saving, loading and removing R datastructes

#Method 1: Using save. image and load method

#Syntax:
save.image(file = “.RData”)

Arguments :
file – name of the file where the R object is saved to or read from.
obj1<-c(1:15)
obj2<-FALSE
Obj3<-"Welcome to R Environment"
save.image("ex1.Rdata")

#Syntax:
Load(path)
load("ex1.Rdata")

#Method 2: Using saveRDS and readRDS method

Syntax:
saveRDS(object, file = “”)

obj1<-c(1:15)
obj2<-FALSE
Obj3<-"Welcome to R Environment"
saveRDS(obj1,file="ex1int.Rdata")
print("Data object")
readRDS("ex1int.Rdata")

#Method 3: Using the save and load method

#Syntax:
save(objects, file)

save(obj1, obj3, file ="tempworkspaceobj.RData")
load("tempworkspaceobj.RData")

OUTPUT

[1] "Data Object"

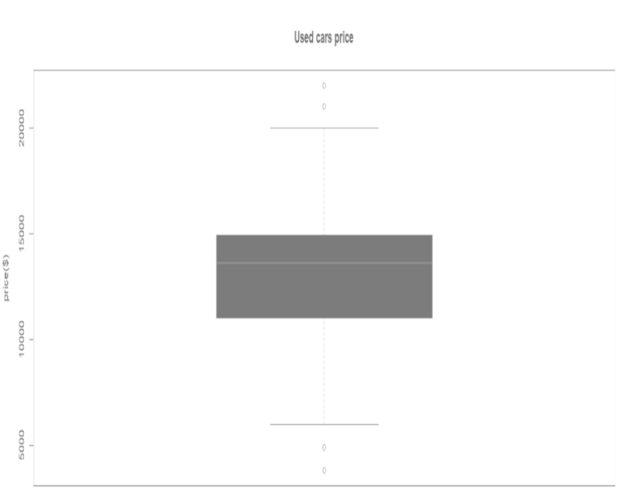
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Ex.No. : 4 Visualizing numeric data – scatterplot, boxplot, piechart, histograms

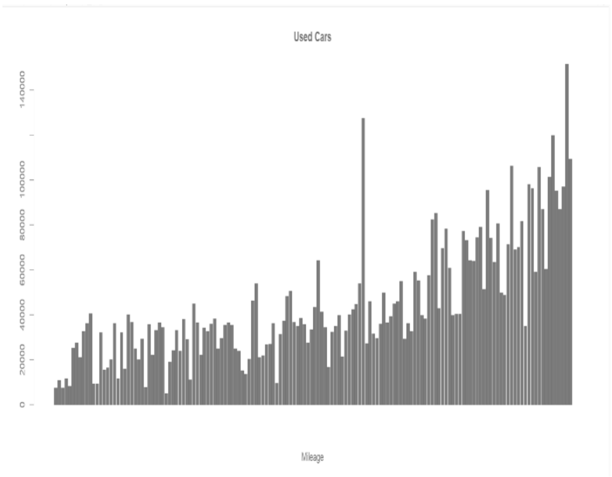
```
us_cars<-read.csv("S:/ML_Dataset/usedcars.csv",stringsAsFactors = FALSE)
str(us_cars)

#Box Plot
boxplot(us_cars$price,main="Used cars price",ylab="price($)",col="blue",border="red")
boxplot(us_cars$mileage,main="Used cars mileage", ylab=" ",col="red",border="brown")
boxplot(us_cars[,0:6],main='Used Cars')

#Bar plot
barplot(us_cars$mileage,main='Used Cars',
        xlab="Mileage",horiz=FALSE,col="blue")

#Histogram
hist(us_cars$mileage,main="Used Cars Mileage",
     xlab='Mileage',col="yellow",border="red")
hist(us_cars$price,main="Used Cars Price",
     xlab="price",col="Green", border="red")

#scatter plot
plot(us_cars$price,us_cars$mileage,main="Scatterplot",
     xlab="Price",ylab="Mileage",pch=12,col='green')

#pie
n=12
pie(rep(1,n),col=rainbow(n))
pie(rep(1,n),col=heat.colors(n))
pie(rep(1,n),col=terrain.colors(n))

#3d Graphs
cone<-function(x,y)
{
  sqrt(x^2 + y^2)
}
#prepare variables.
x<-y<-seq(-1,1,length=30)
z<-outer(x,y,cone)
persp(x,y,z,main="Perspective Plot of a Cone",
      zlab="Height",
      theta=30,phi=15,
      col="green",shade=0.4)
```

OUTPUT

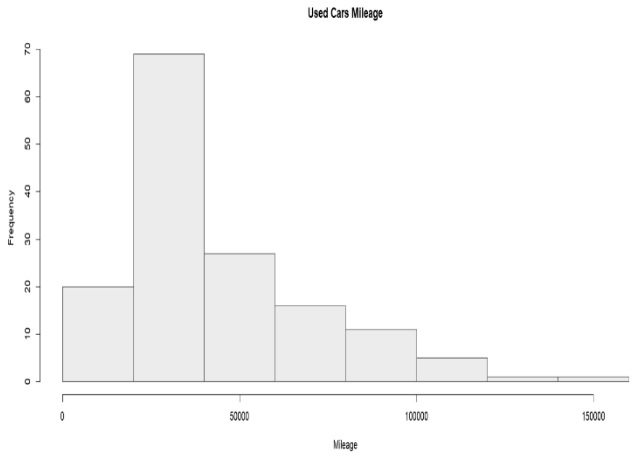
BoxPlot



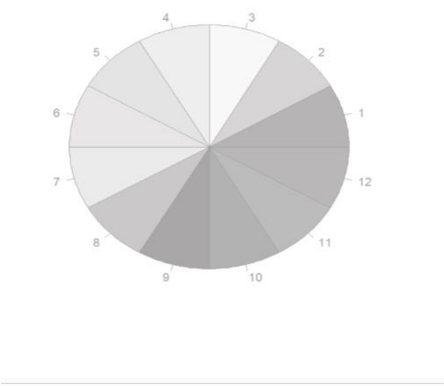
BarPlot



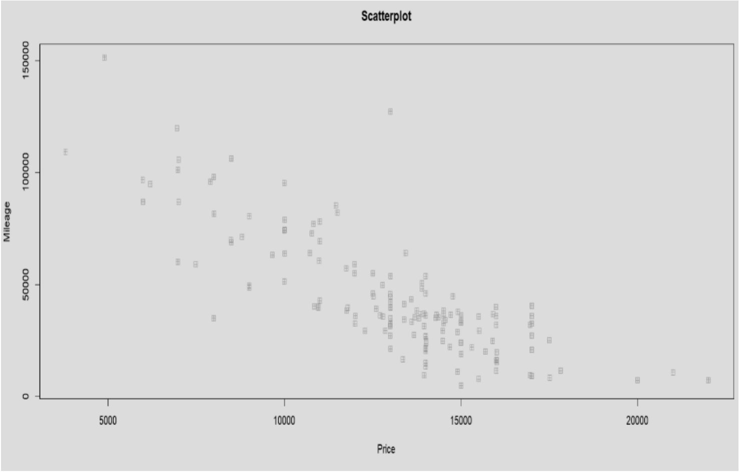
Histogram



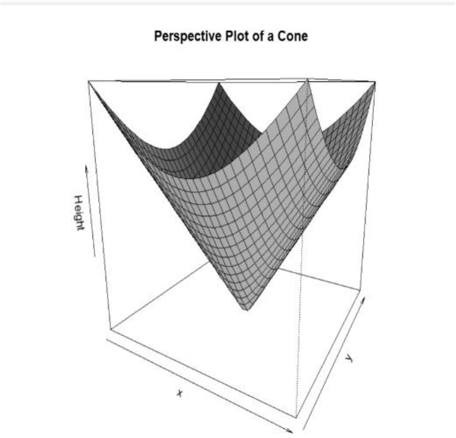
Piechart



ScatterPlot



3D Graphs



Ex.No. : 5 Measuring Central Tendency

```
usedcars <- read.csv("D:/RPractical_progrmas/usedcars.csv", stringsAsFactors = FALSE)

str(usedcars)
print(head(usedcars))
max1=max(usedcars$price)
print(max1)

min1=min(usedcars$price)
print(min1)

range=max1-min1
print(range)
r=range(usedcars$price)
print(r)

variance=var(usedcars$price)
print(variance)

std=sd(usedcars$price)


quartiles=quantile(usedcars$price)
print(quartiles)

IQR=IQR(usedcars$price)
print(IQR)

  summary=summary(usedcars$price)
  print(summary)

  summary(usedcars)
```

OUTPUT

```
year   model  price   mileage  color   transmission
1 2011   SEL  21992    7413   Yellow     AUTO
2 2011   SEL  20995   10926    Gray      AUTO
3 2011   SEL  19995    7351   Silver     AUTO
4 2011   SEL  17809   11613   Gray       AUTO
5 2012    SE  17500    8367   White      AUTO
6 2010   SEL  17495   25125   Silver     AUTO
```

#Max

```
[1] 21992
```

#Min
[1] 3800

#Range
[1] 18192

[1] 3800 21992

#variance
[1] 9749892

#Standard Deviation
[1] 3122.482

#Quartiles

```
    0%      25%      50%      75%     100%
 3800.0 10995.0 13591.5 14904.5 21992.0
```

#IQR
[1] 3909.5

#Summary

```
   Min.  1st Qu.  Median    Mean  3rd Qu.  Max.
   3800   10995   13592   12962   14904   21992
```

#Summary used Cars

```
      year          model               price          mileage
 Min.    :2000   Length:150         Min.   : 3800   Min.   :  4867
 1st Qu.:2008   Class :character   1st Qu.:10995   1st Qu.: 27200
 Median :2009   Mode  :character   Median :13592   Median : 36385
 Mean   :2009                      Mean   :12962   Mean   : 44261
 3rd Qu.:2010                      3rd Qu.:14904   3rd Qu.: 55125
 Max.   :2012                      Max.   :21992   Max.   :151479
    color           transmission
 Length:150         Length:150
 Class :character   Class :character
 Mode  :character   Mode  :character
```

Ex.No. : 6 Data Pre-Processing Methods

```
mydata<-read.csv("S:/ML_Dataset/mysampledata.csv")
mydata


#dealing with missing values
mydata$age<-ifelse(is.na(mydata$age),ave(mydata$age,FUN = function(x)
mean(x,na.rm=TRUE)),mydata$age)
mydata
mydata$salary<-ifelse(is.na(mydata$salary),ave(mydata$salary, FUN = function(x)
mean(x,na.rm=TRUE)),mydata$salary)
mydata
mydata$age<-as.numeric(format(round(mydata$age,0)))


#Dealing with categorical Data
mydata$nation<-factor(mydata$nation,levels=c('India','Russia','Germany'),labels=c(1,2,3))
mydata$purchased_item<-factor(mydata$purchased_item,levels = c('No','Yes'),labels=c(0,1))
mydata


install.Packages("caTools")
library(caTools)
set.seed(123)
Split<-sample.split(mydata$purchased_item,SplitRatio=0.8)
training_set<-subset(mydata,Split==TRUE)
test_set<-subset(mydata,Split==FALSE)
training_set[,3:4]<-scale(training_set[,3:4])
test_set[,3:4]<-scale(test_set[,3:4])
```

OUTPUT

#Dealing with missing values

```
    nation purchased_item age salary
1    India             No  25  35000
2   Russia            Yes  NA  40000
3  Germany             No  50  54000
4   Russia             No  35     NA
5  Germany            Yes  40  60000
6    India            Yes  35  58000
7   Russia             No  NA  52000
8    India            Yes  48     NA
9  Germany             No  50  83000
10   India            Yes  37     NA
11 Germany             No  21  24000
12   India            Yes  NA  60000
13  Russia             No  63  70000
14 Germany            Yes  26  36000
15   India             No  45  40000
> mydata$age<-ifelse(is.na(mydata$age),ave(m
> mydata
    nation purchased_item      age salary
1    India             No 25.00000  35000
2   Russia            Yes 39.58333  40000
3  Germany             No 50.00000  54000
4   Russia             No 35.00000     NA
5  Germany            Yes 40.00000  60000
6    India            Yes 35.00000  58000
7   Russia             No 39.58333  52000
8    India            Yes 48.00000     NA
9  Germany             No 50.00000  83000
10   India            Yes 37.00000     NA
11 Germany             No 21.00000  24000
12   India            Yes 39.58333  60000
13  Russia             No 63.00000  70000
14 Germany            Yes 26.00000  36000
15   India             No 45.00000  40000
```

#Dealing with categorical Data

```
   nation purchased_item age salary
1       1              0  25  35000
2       2              1  40  40000
3       3              0  50  54000
4       2              0  35     NA
5       3              1  40  60000
6       1              1  35  58000
7       2              0  40  52000
8       1              1  48     NA
9       3              0  50  83000
10      1              1  37     NA
11      3              0  21  24000
12      1              1  40  60000
13      2              0  63  70000
14      3              1  26  36000
15      1              0  45  40000
> 
```

Ex.No. : 7 Build model using K-Nearest Neighbour

```
loan <- read.csv("credit_data.csv")

str(loan)

 #Data Cleaning

loan.subset <-
loan[c('Creditability','Age..years.','Sex...Marital.Status','Occupation','Account.Balance','Credit
.Amount','Length.of.current.employment','Purpose')]

str(loan.subset)

#Data Normalization

head(loan.subset)

#Normalization

normalize <- function(x) {

return ((x - min(x)) / (max(x) - min(x))) }

loan.subset.n <- as.data.frame(lapply(loan.subset[,2:8], normalize))

head(loan.subset.n)

#Data Splicing

set.seed(123)

dat.d <- sample(1:nrow(loan.subset.n),size=nrow(loan.subset.n)*0.7,replace = FALSE)

train.loan <- loan.subset[dat.d,] # 70% training data

test.loan <- loan.subset[-dat.d,] # remaining 30% test data

#Creating seperate dataframe for 'Creditability' feature which is our target.

train.loan_labels <- loan.subset[dat.d,1]

test.loan_labels <-loan.subset[-dat.d,1]

#Building a Machine Learning model

install.packages('class')

library(class)

NROW(train.loan_labels)

knn.26 <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=26)

knn.27 <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=27)
```

#Model Evaluation

ACC.26 <- 100 * sum(test.loan_labels == knn.26)/NROW(test.loan_labels)

ACC.27 <- 100 * sum(test.loan_labels == knn.27)/NROW(test.loan_labels)

ACC.26

ACC.27

table(knn.26 ,test.loan_labels)

table(knn.27 ,test.loan_labels)

install.packages('caret')

library(caret)

confusionMatrix(table(knn.26 ,test.loan_labels))

#Optimization

i=1

k.optm=1

for (i in 1:28){

knn.mod <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=i)

k.optm[i] <- 100 * sum(test.loan_labels == knn.mod)/NROW(test.loan_labels)

k=i

cat(k,'=',k.optm[i],'')

}

#Accuracy plot

plot(k.optm, type="b", xlab="K- Value",ylab="Accuracy level")

OUTPUT

#DataCleaning

```
> str(loan.subset)
'data.frame':   1000 obs. of  8 variables:
 $ Creditability              : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Age..years.                : int  21 36 23 39 38 48 39 40 65 23 ...
 $ Sex...Marital.Status       : int  2 3 2 3 3 3 3 3 2 2 ...
 $ Occupation                 : int  3 3 2 2 2 2 2 2 1 1 ...
 $ Account.Balance            : int  1 1 2 1 1 1 1 1 4 2 ...
 $ Credit.Amount              : int  1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
 $ Length.of.current.employment: int  2 3 4 3 3 2 4 2 1 1 ...
 $ Purpose                    : int  2 0 9 0 0 0 0 0 3 3 ...
```

Data Normalization

```
> head(loan.subset)
  Creditability Age..years. Sex...Marital.Status Occupation Account.Balance Credit.Amount Length.of.current.employment Purpose
1             1          21                    2          3               1          1049                            2       2
2             1          36                    3          3               1          2799                            3       0
3             1          23                    2          2               2           841                            4       9
4             1          39                    3          2               1          2122                            3       0
5             1          38                    3          2               1          2171                            3       0
6             1          48                    3          2               1          2241                            2       0
> normalize<-function(x){
+   return((x-min(x))/(max(x)-min(x)))
+ }
> loan.subset.n<-as.data.frame(lapply(loan.subset[,2:8],normalize))
> head(loan.subset.n)
  Age..years. Sex...Marital.Status Occupation Account.Balance Credit.Amount Length.of.current.employment Purpose
1  0.03571429            0.3333333  0.6666667       0.0000000    0.04396390                         0.25     0.2
2  0.30357143            0.6666667  0.6666667       0.0000000    0.14025531                         0.50     0.0
3  0.07142857            0.3333333  0.3333333       0.3333333    0.03251898                         0.75     0.9
4  0.35714286            0.6666667  0.3333333       0.0000000    0.10300429                         0.50     0.0
5  0.33928571            0.6666667  0.3333333       0.0000000    0.10570045                         0.50     0.0
6  0.51785714            0.6666667  0.3333333       0.0000000    0.10955211                         0.25     0.0
```

Building a Machine Learning model

[1] 700

#Model Evaluation

[1] 69

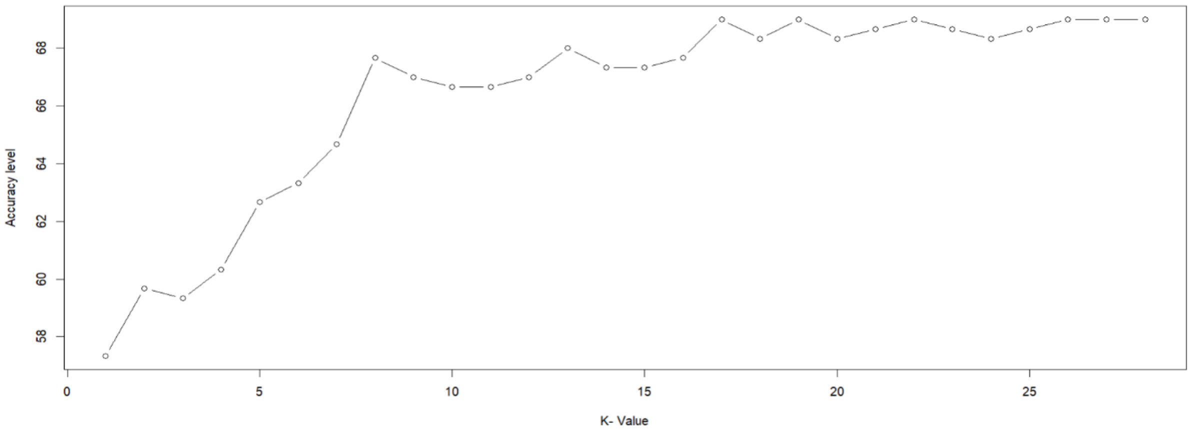
[1] 69

```
test.loan_labels
knn.26   0   1
     0   8   6
     1  87 199
test.loan_labels
knn.27   0   1
     0   8   6
     1   87 199
     2
```

#Optimization

1 = 57.33333 2 = 59.66667 3 = 59.33333 4 = 60.33333 5 = 62.66667 6 = 63.33333 7 = 64.66667 8 = 67.66667 9 = 67 10 = 66.66667 11 = 66.66667 12 = 67 13 = 68 14 = 67.33333 15 = 67.33333 16 = 67.66667 17 = 69 18 = 68.33333 19 = 69 20 = 68.33333 21 = 68.66667 22 = 69 23 = 68.66667 24 = 68.33333 25 = 68.66667 26 = 69 27 = 69 28 = 69

#Accuracy Plot

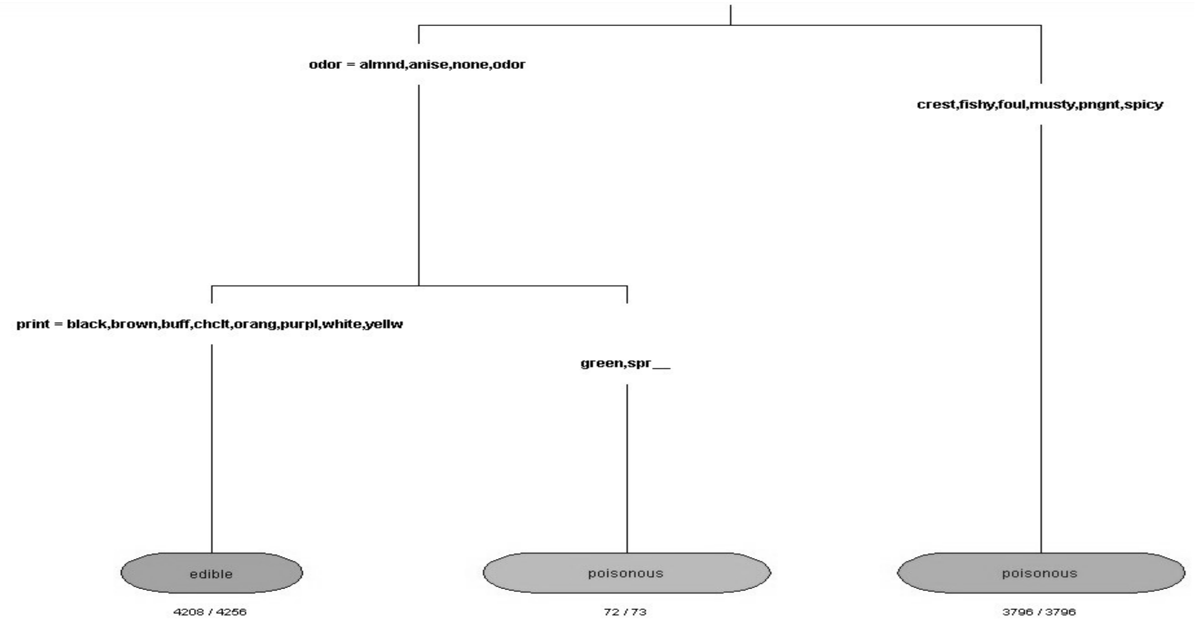


Ex.No. : 8 Decision Tree

```
mushrooms <- read.csv("mushroom.csv", header = FALSE)

head(mushrooms)

colnames(mushrooms) <-
c("Class","cap.shape","cap.surface","cap.color","bruises","odor","gill.attachment","gill.spacing","gill.size","gill.color","stalk.shape","stalk.root","stalk.surface.above.ring","stalk.surface.below.ring","stalk.color.above.ring","stalk.color.below.ring","veil.type","veil.color","ring.number","ring.type","print","population","habitat")

head(mushrooms)

# Define the factor names for "Class"

levels(mushrooms$Class) <- c("Edible","Poisonous")

# Define the factor names for "odor"

levels(mushrooms$odor) <-
c("Almonds","Anise","Creosote","Fishy","Foul","Musty","None","Pungent","Spicy")


# Define the factor names for "print"

levels(mushrooms$print) <-
c("Black","Brown","Buff","Chocolate","Green","Orange","Purple","White","Yellow")

head(mushrooms)

# Import our required libraries

library(rpart)

library(rpart.plot)

# Create a classification decision tree using "Class" as the variable we want to predict and everything else as its predictors.

myDecisionTree <- rpart(Class ~ ., data = mushrooms, method = "class")


# Print out a summary of our created model.

print(myDecisionTree)

rpart.plot(myDecisionTree, type = 3, extra = 2, under = TRUE, faclen=5, cex = .75)

newCase  <- mushrooms[10,-1]

newCase

predict(myDecisionTree, newCase, type = "class")
```

```
train_ind <- sample(c(1:nrow(mushrooms)), size = 10)
## 75% of the sample size
n <- nrow(mushrooms)
smp_size <- floor(0.75 * n)

## set the seed to make your partition reproductible
set.seed(123)
train_ind <- sample(c(1:n), size = smp_size)

mushrooms_train <- mushrooms[train_ind, ]
mushrooms_test <- mushrooms[-train_ind, ]
newDT <- rpart(Class ~ ., data = mushrooms_train, method = "class")
result <- predict(newDT, mushrooms_test[,-1], type = "class")
head(result)
head(mushrooms_test$Class)
table(mushrooms_test$Class, result)
```

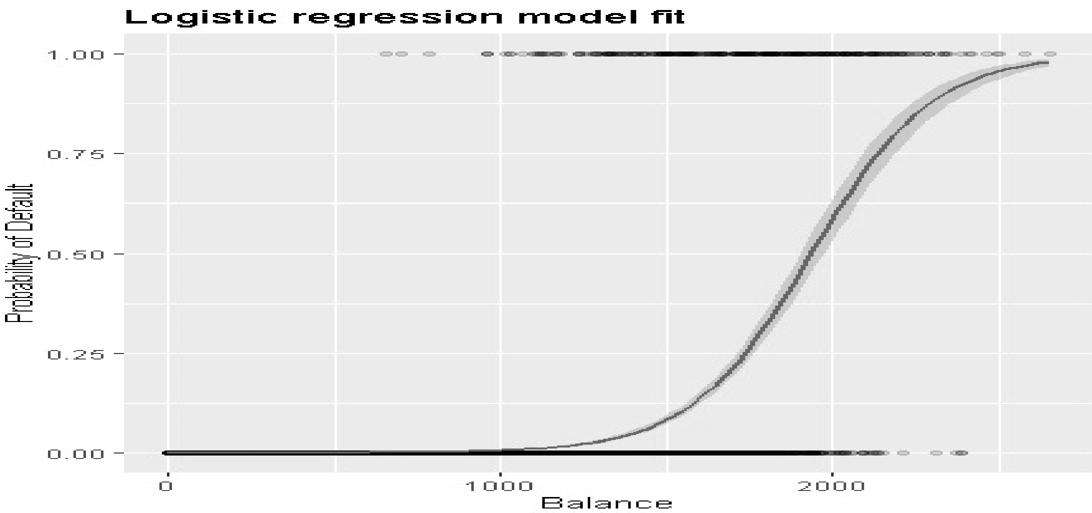
OUTPUT



Ex.No. : 9 Develop a model using logistic regression

```
library(tidyverse)
library(modelr)
library(broom)

#Install ISLR Package
install.packages('ISLR')

#Load ISLR Package
library('ISLR')

# Load data
(mydata <- as_tibble(ISLR::Default))

#Checking for NA values
sum(is.na(mydata))

#Creating the Training and Testing data set
sample <- sample(c(TRUE, FALSE), nrow(mydata), replace = T, prob = c(0.6,0.4))
train <- mydata[sample, ]
test <- mydata[!sample, ]

#Fitting a logistic regression model
logmodel <- glm(default ~ balance, family = "binomial", data = train)
#Plotting a graph: Probability of default Vs Balance
mydata %>%
mutate(prob = ifelse(default == "Yes", 1, 0)) %>%
ggplot(aes(balance, prob)) +
geom_point(alpha = .15) +
geom_smooth(method = "glm", method.args = list(family = "binomial")) +
ggtitle("Logistic regression model fit") +
xlab("Balance") +
ylab("Probability of Default")

#Summary of the Logistic Regression
summary(logmodel)
```

OUTPUT



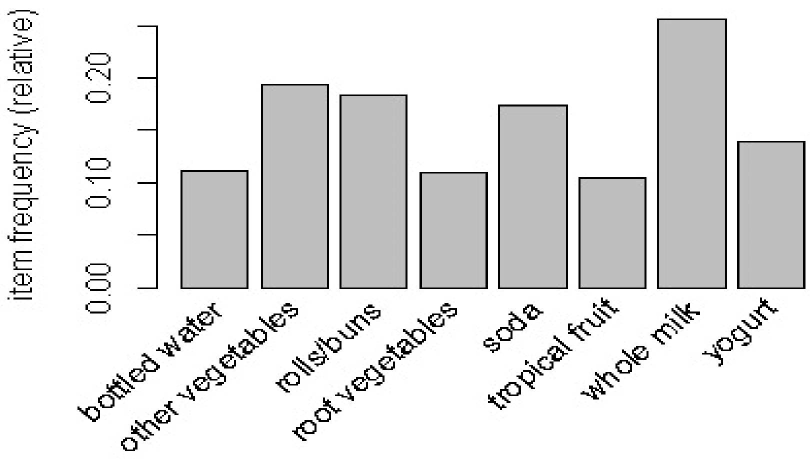
Logistic regression model fit
1.00
0.75
0.50
0.25
0.00
Probability of Default
0
1000
2000
Balance

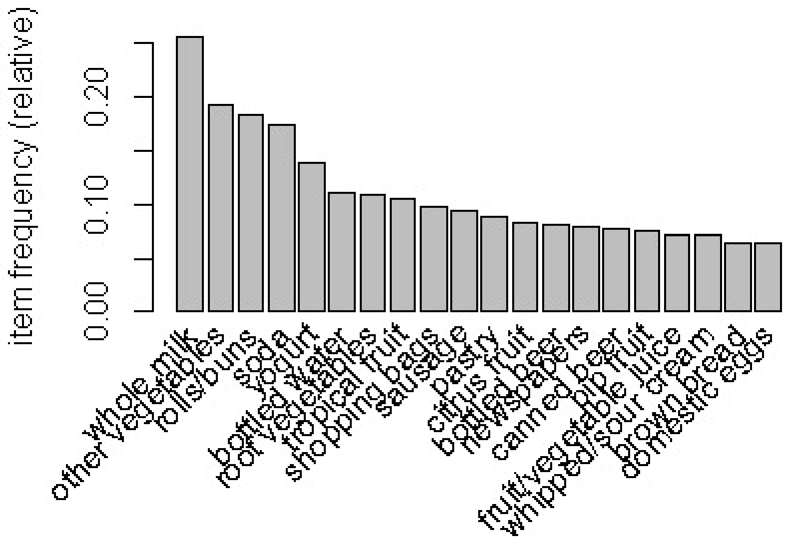
Ex.No. : 10 Identify patterns using Association Rules

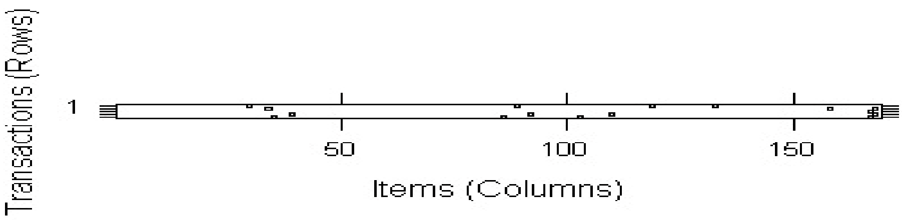
```
# load the grocery data into a sparse matrix
library(arules)
groceries <- read.transactions("groceries.csv", sep = ",")
summary(groceries)

# look at the first five transactions
inspect(groceries[1:5])

# examine the frequency of items
itemFrequency(groceries[, 1:3])

# plot the frequency of items
itemFrequencyPlot(groceries, support = 0.1)
itemFrequencyPlot(groceries, topN = 20)

# a visualization of the sparse matrix for the first five transactions
image(groceries[1:5])

# visualization of a random sample of 100 transactions
image(sample(groceries, 100))

## Step 3: Training a model on the data ----
library(arules)

# default settings result in zero rules learned
apriori(groceries)

# set better support and confidence levels to learn more rules
groceryrules <- apriori(groceries, parameter = list(support =
                    0.006, confidence = 0.25, minlen = 2))
groceryrules

## Step 4: Evaluating model performance ----
# summary of grocery association rules
summary(groceryrules)

# look at the first three rules
inspect(groceryrules[1:3])

## Step 5: Improving model performance ----

# sorting grocery rules by lift
inspect(sort(groceryrules, by = "lift")[1:5])

# finding subsets of rules containing any berry items
berryrules <- subset(groceryrules, items %in% "berries")
inspect(berryrules)
```

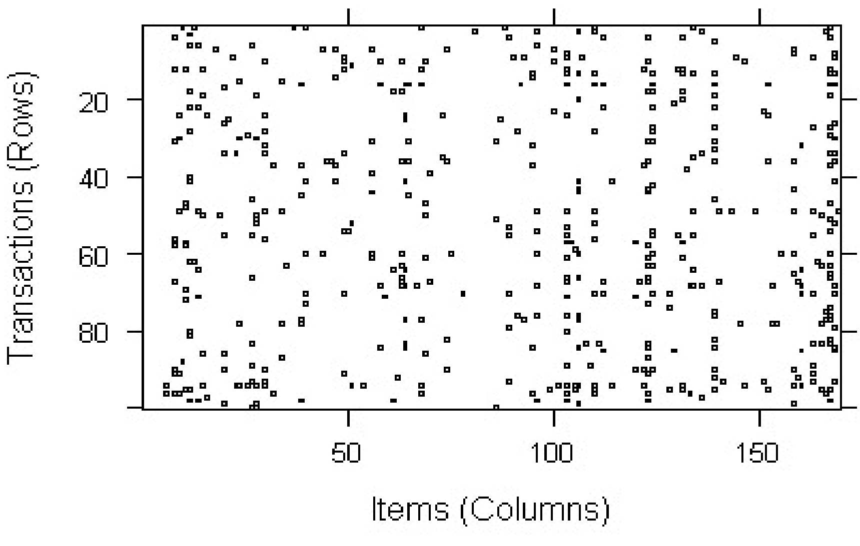
```
# writing the rules to a CSV file
write(groceryrules, file = "groceryrules.csv",
     sep = ",", quote = TRUE, row.names = FALSE)

# converting the rule set to a data frame
groceryrules_df <- as(groceryrules, "data.frame")
str(groceryrules_df)
```

OUTPUT






Transactions (Rows)
1
50
100
150
Items (Columns)


Transactions (Rows)
20
40
60
80
50
100
150
Items (Columns)

Ex.No. : 11 Identify patterns using Association Rules