Ex No:1 **Date:**	**Basics of R – data types, vectors, factors, list and data frames**

AIM:

To implement and understand the basics of R programming with its data types, vectors, factors, list and data frames.

ALGORITHM:

1. Start
2. Assign values in logical, numerical, character, complex and character in raw form to a variable v.
3. Print the class of v.
4. Assign a vector for subject Names, temperature and flu_status for three patients using c() function and access the elements.
5. Create a factor using factor() with duplicate values and assign level with distinct values.
6. Display the specific element and check for certain values in factor.
7. Create a list using list() from the patient details and access the multiple elements.
8. Create a data frame using data.frame() with multiple vectors as features. Access the elements.
9. Create a matrix using matrix() with different allocations and access the elements.
10. Stop.

PROGRAM:

```
#Data Types
v<-TRUE
print(class(v))
v<-23.5
print(class(v))
v<-2L
print(class(v))
v<-2+5i
print(class(v))
v<-"TRUE"
print(class(v))
v<-charToRaw("Hello")
print(class(v))

#Vectors
subject_name<-c("John Doe","Jane Doe","Steven Grant")
temperature<-c(98.1,98.6,101.4)
flu_status<-c(FALSE,FALSE,TRUE)
temperature[2]
temperature[2:3]
temperature[-2]

#Factors
gender<-factor(c("MALE","FEMALE","MALE"))
gender
blood<-factor(c("O","AB","A"),levels=c("A","B","AB","O"))
```

```
blood[1:2]
symptoms<-factor(c("SEVERE","MILD","MODERATE"),
                levels=c("MILD","MODERATE","SEVERE"),
                ordered=TRUE)
symptoms>"MODERATE"

#Lists
subject1<-list(fullname=subject_name[1],
              temperature=temperature[1],
              flu_status=flu_status[1],
              gender=gender[1],
              blood=blood[1],
              symptoms=symptoms[1])
subject1
subject1[2]
subject1[[2]]
subject1$temperature
subject1[c("temperature","flu_status")]

#Data Frames
pt_data<-data.frame(subject_name, temperature, flu_status,
                   gender,blood,symptoms)
pt_data
pt_data$subject_name
pt_data[c("temperature","flu_status")]
pt_data[c(1,2),c(2,4)]
pt_data[,1]
pt_data[,]

#Matrices
m<-matrix(c(1,2,3,4),ncol=2)
print(m)
m<-matrix(c(1,2,3,4,5,6),nrow=3)
print(m)
print(m[1,])
print(m[1,])
thismatrix <- matrix(c("apple", "banana", "cherry","orange"), nrow = 2, ncol = 2)
for (rows in 1:nrow(thismatrix)) {
  for (columns in 1:ncol(thismatrix)) {
    print(thismatrix[rows, columns])
  }
}
```

OUTPUT:

```
[1] "logical"
[1] "numeric"
[1] "integer"
[1] "complex"
[1] "character"
[1] "raw"
[1] 98.6
[1]  98.6 101.4
[1]  98.1 101.4
[1] MALE   FEMALE MALE
Levels: FEMALE MALE
[1] O  AB
Levels: A B AB O
[1]  TRUE FALSE FALSE
$fullname
[1] "John Doe"

$temperature
[1] 98.1

$flu_status
[1] FALSE

$gender
[1] MALE
Levels: FEMALE MALE

$blood
[1] O
Levels: A B AB O

$symptoms
[1] SEVERE
Levels: MILD < MODERATE < SEVERE

$temperature
[1] 98.1

[1] 98.1
[1] 98.1
$temperature
[1] 98.1

$flu_status
[1] FALSE

  subject_name temperature flu_status gender blood symptoms
1     John Doe        98.1      FALSE   MALE     O   SEVERE
2     Jane Doe        98.6      FALSE FEMALE    AB     MILD
3 Steven Grant       101.4       TRUE   MALE     A MODERATE
[1] "John Doe"     "Jane Doe"     "Steven Grant"
  temperature flu_status
1        98.1      FALSE
2        98.6      FALSE
3       101.4       TRUE
```

```
2        98.6      FALSE
3       101.4       TRUE
  temperature gender
1        98.1   MALE
2        98.6 FEMALE
[1] "John Doe"     "Jane Doe"     "Steven Grant"
  subject_name temperature flu_status gender blood symptoms
1     John Doe        98.1      FALSE   MALE     O   SEVERE
2     Jane Doe        98.6      FALSE FEMALE    AB     MILD
3 Steven Grant       101.4       TRUE   MALE     A MODERATE
     [,1] [,2]
[1,]    1    3
[2,]    2    4
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
[1] 1 4
[1] 1 4
[1] "apple"
[1] "cherry"
[1] "banana"
[1] "orange"
>
```

Result:

Thus the R Script program to implement various data types, vectors, factors, lists and data frames is executed successfully and the output is verified.

Ex no: 2 Date:	**Diagnosis of Breast Cancer using KNN.**

Aim:

To implement a R program to predict and diagnose Breast Cancer using KNN algorithm.

Algorithm:

1. Start
2. Read the csv file from the directory and store it in bcd variable.
3. Drop the first column id.
4. Change the diagnosis feature with categorical values B and M in a factor
5. Normalize the dataset.
6. Split the dataset for training and testing, with diagnosis as the response variable and the rest as the predictor variables.
7. Import the library "class" for knn classification.
8. Predict the knn model using knn() with 5 clusters with the corresponding training and testing data.
9. Display the confusion matrix and accuracy of the knn model.
10. Stop

PROGRAM:

```
bcd<-read.csv("../input/breast-cancer-dataset/Breast_Cancer.csv", stringsAsFactors=FALSE)
bcd<-bcd[-1]
bcd$diagnosis<-factor(bcd$diagnosis, levels=c("B","M"), labels=c("Benign","Malignant"))
normalize<-function(x){
  return (x-min(x)) / (max(x)- min(x))
}
bcd_n <- as.data.frame(lapply(bcd[2:31], normalize))
x_train <- bcd_n[1:469,]
x_test <- bcd_n[470:569,]
y_train <- bcd[1:469,1]
y_test <- bcd[470:569,1]
library(class)
y_pred<-knn(train=x_train,test=x_test,cl=y_train,k=5)
tbl=table(x=y_test,y=y_pred)
tbl
accuracy = sum(diag(tbl))
```

OUTPUT:

PROBLEMS 37 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
'data.frame':   569 obs. of  32 variables:
 $ id               : int  87139402 8910251 905520 868871 9012568 906539 925291 87880 862989 89827 ...
 $ diagnosis        : chr  "B" "B" "B" "B" ...
 $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
 $ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
 $ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
 $ area_mean        : num  464 346 373 385 712 ...
 $ smoothness_mean  : num  0.1028 0.0969 0.1077 0.1164 0.0796 ...
 $ compactness_mean : num  0.0698 0.1147 0.078 0.1136 0.0693 ...
 $ concavity_mean   : num  0.0399 0.0639 0.0305 0.0464 0.0339 ...
 $ points_mean      : num  0.037 0.0264 0.0248 0.048 0.0266 ...
 $ symmetry_mean    : num  0.196 0.192 0.171 0.177 0.172 ...
 $ dimension_mean   : num  0.0595 0.0649 0.0634 0.0607 0.0554 ...
 $ radius_se        : num  0.236 0.451 0.197 0.338 0.178 ...
 $ texture_se       : num  0.666 1.197 1.387 1.343 0.412 ...
 $ perimeter_se     : num  1.67 3.43 1.34 1.85 1.34 ...
 $ area_se          : num  17.4 27.1 13.5 26.3 17.7 ...
 $ smoothness_se    : num  0.00805 0.00747 0.00516 0.01127 0.00501 ...
 $ compactness_se   : num  0.0118 0.03581 0.00936 0.03498 0.01485 ...
 $ concavity_se     : num  0.0168 0.0335 0.0106 0.0219 0.0155 ...
 $ points_se        : num  0.01241 0.01365 0.00748 0.01965 0.00915 ...
 $ symmetry_se      : num  0.0192 0.035 0.0172 0.0158 0.0165 ...
 $ dimension_se     : num  0.00225 0.00332 0.0022 0.00344 0.00177 ...
 $ radius_worst     : num  13.5 11.9 12.4 11.9 16.2 ...
 $ texture_worst    : num  15.6 22.9 26.4 15.8 15.7 ...
 $ perimeter_worst  : num  87 78.3 79.9 76.5 104.5 ...
 $ area_worst       : num  549 425 471 434 819 ...
 $ smoothness_worst : num  0.139 0.121 0.137 0.137 0.113 ...
 $ compactness_worst: num  0.127 0.252 0.148 0.182 0.174 ...
 $ concavity_worst  : num  0.1242 0.1916 0.1067 0.0867 0.1362 ...
 $ points_worst     : num  0.0939 0.0793 0.0743 0.0861 0.0818 ...
 $ symmetry_worst   : num  0.283 0.294 0.3 0.21 0.249 ...
 $ dimension_worst  : num  0.0677 0.0759 0.0788 0.0678 0.0677 ...
           y
x            Benign Malignant
  Benign        61         0
  Malignant      4        35
[1] "Accuracy 96"
> 
```

Result:

Thus the R Script program to implement diagnosis of Breast Cancer using K-Nearest Neighbour algorithm is executed successfully and the output is verified.

Ex No: 3 Date:	**Filtering Mobile phone spam using Naïve Bayes**

AIM:

To implement a R program to Filter Mobile phone spam using Naïve Bayes.

ALGORITHM:

1. Start
2. Import the csv file and store the dataframe in "Sms". Have a glimpse at the structure of the data frame.
3. Remove the unneccesary columns which is from column 3 to 5.
4. Convert the labels as factors.
5. Remove special characters from the dataset and retain only alpha numeric characters using alnum in str_replace_all() from "stringr" package.
6. Create a volatile corpus VCorpus() for text mining from the source object of "v2" which is extracted using VectorSource() .
7. Create a DocumentTermMatrix() to split the SMS message into individual Components.
8. Create training and testing dataset with the split ratio 0.75.
9. Find the frequent terms which appear for atleast 5 times in DocumentTermMatrix in training and testing dataset respectively.
10. Train the model using naiveBayes() from e1071 library.
11. Evaluate the model Performance.
12. Print the confusion matrix and Accuracy of the model.
13. Stop.

PROGRAM:

```
sms <- read.csv("../input/spam-ham-dataset/spam.csv", stringsAsFactors=FALSE)
str(sms)
sms <-sms[-3:-5]
sms$v1 <- factor(sms$v1)
library(stringr)
sms$v2 = str_replace_all(sms$v2, "[^[:alnum:]]", " ") %>% str_replace_all(.,"[
]+", " ")
library(tm)
sms_corpus <- VCorpus(VectorSource(sms$v2))
```

```
print(sms_corpus)
print(as.character(sms_corpus[[6]]))
sms_dtm <- DocumentTermMatrix(sms_corpus, control = list
(tolower=TRUE,removeNumbers=TRUE,stopwords=TRUE,removePunctuations=TRUE,stemmi
ng=TRUE))
x_train <- sms_dtm[1:4169, ]
x_test <- sms_dtm[4170:5572, ]
y_train <- sms[1:4169, ]$v1
y_test <- sms[4170:5572, ]$v1
sms_freq_word_train <- findFreqTerms(x_train, 5)
sms_freq_word_test <- findFreqTerms(x_test, 5)
x_train<- x_train[ , sms_freq_word_train]
x_test <- x_test[ , sms_freq_word_test]
convert_counts <- function(x) {x <- ifelse(x > 0, "Yes", "No")}
x_train <- apply(x_train, MARGIN = 2,convert_counts)
x_test <- apply(x_test, MARGIN = 2,convert_counts)
library(e1071)
model <- naiveBayes(x_train, y_train,laplace=1)
y_pred <- predict(model, x_test)
cm = table(y_pred, y_test)
print(cm)
acc = sum(diag(cm))/sum(cm)
print(paste("Accuracy: ",acc*100,"%"))
```

OUTPUT:

```
PROBLEMS 73   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER                R Interactive

'data.frame':   5572 obs. of  5 variables:
 $ v1 : chr  "ham" "ham" "spam" "ham" ...
 $ v2 : chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat..." "Ok lar... Joking wif u oni..." "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C "U dun say so early hor... U c already then say..." ...
 $ X  : chr  "" "" "" "" ...
 $ X.1: chr  "" "" "" "" ...
 $ X.2: chr  "" "" "" "" ...
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 5572
[1] "FreeMsg Hey there darling it s been 3 week s now and no word back I d like some fun you up for it still Tb ok XxX std chgs to send 1 50 to rcv"
      y_test
y_pred  ham spam
  ham  1205   10
  spam   16  172
[1] "Accuracy:  98.1468282252316 %"
>
```

RESULT:

Thus the R program to implement filtering of Mobile phone spam using Naïve Bayes is executed successfully and the output is verified.

Ex No:4 Date:	**Risky Bank Loans using Decision Trees**

AIM:

To implement a R program to find Risky Bank loans using Decision Tree.

ALGORITHM:

1. Start
2. Import the dataset credit.csv and display the structure of the dataset.
3. Display the table to find the range of values and find the missing values.
4. Factorise the default column and set seed of 123.
5. Split the dataset for training and testing in the ratio of 0.8, with “default” as the response variable, and the rest as predictor variables.
6. Import the library C5.0 for implementing decision tree.
7. Train the decision tree model using C5.0 function for the training dataset.
8. Test the model to predict using predict(). Print the confusion matrix.
9. Print the accuracy of the decision tree model.
10. Stop

PROGRAM:

```
credit <- read.csv("credit.csv")

str(credit)

table(credit$savings_balance)

summary(credit$amount)

credit$default <- factor(credit$default)

set.seed(123)

train_sample <- sample(1000, 800)

str(train_sample)

x_train <- credit[train_sample, -17]

x_test <- credit[-train_sample, -17]

y_train <- credit[train_sample, 17]

y_test <- credit[-train_sample, 17]

library(C50)

model <- C5.0(x_train,y_train)
```

summary(model)

y_pred <- predict(model,x_test)

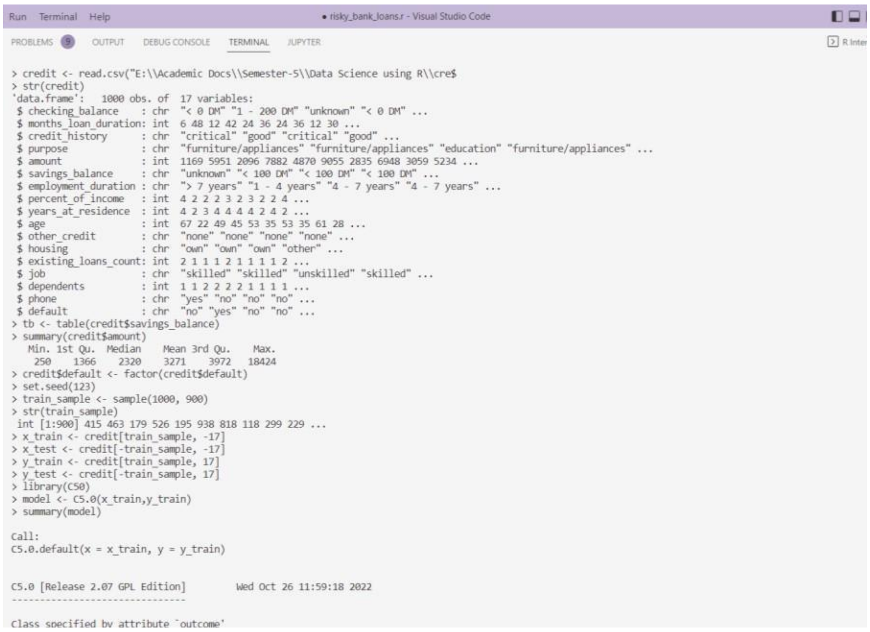
cm = table(y_pred,y_test)

print(cm)

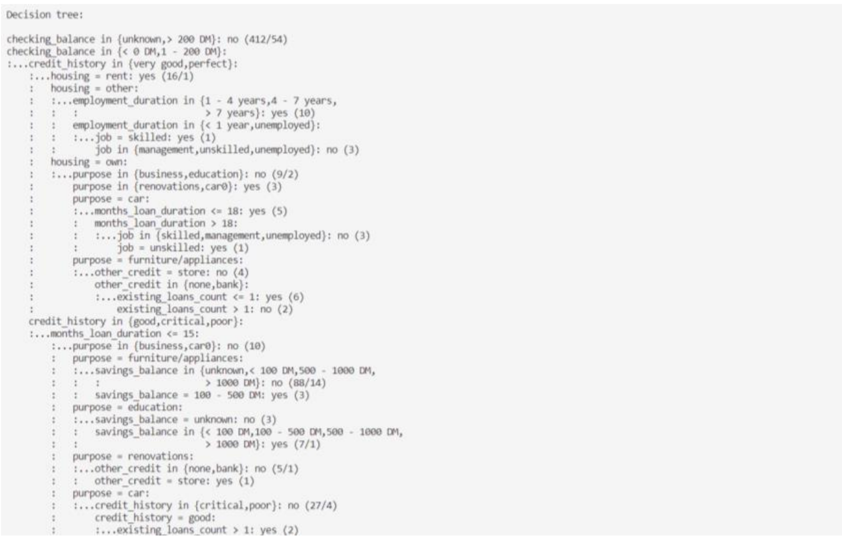
acc=sum(diag(cm))/sum(cm)

print(paste("Accuaracy: ",acc*100,"%"))

OUTPUT:



```
Run   Terminal   Help                    • risky_bank_loans.r - Visual Studio Code
PROBLEMS 9   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

> credit <- read.csv("E:\\Academic Docs\\Semester-5\\Data Science using R\\cre$
> str(credit)
'data.frame':   1000 obs. of  17 variables:
 $ checking_balance    : chr  "< 0 DM" "1 - 200 DM" "unknown" "< 0 DM" ...
 $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
 $ credit_history      : chr  "critical" "good" "critical" "good" ...
 $ purpose             : chr  "furniture/appliances" "furniture/appliances" "education" "furniture/appliances" ...
 $ amount              : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
 $ savings_balance     : chr  "unknown" "< 100 DM" "< 100 DM" "< 100 DM" ...
 $ employment_duration : chr  "> 7 years" "1 - 4 years" "4 - 7 years" "4 - 7 years" ...
 $ percent_of_income   : int  4 2 2 2 3 2 3 2 2 4 ...
 $ years_at_residence  : int  4 2 3 4 4 4 4 2 4 2 ...
 $ age                 : int  67 22 49 45 53 35 53 35 61 28 ...
 $ other_credit        : chr  "none" "none" "none" "none" ...
 $ housing             : chr  "own" "own" "own" "other" ...
 $ existing_loans_count: int  2 1 1 1 2 1 1 1 1 2 ...
 $ job                 : chr  "skilled" "skilled" "unskilled" "skilled" ...
 $ dependents          : int  1 1 2 2 2 2 1 1 1 1 ...
 $ phone               : chr  "yes" "no" "no" "no" ...
 $ default             : chr  "no" "yes" "no" "no" ...
> tb <- table(credit$savings_balance)
> summary(credit$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    250    1366    2320    3271    3972   18424
> credit$default <- factor(credit$default)
> set.seed(123)
> train_sample <- sample(1000, 900)
> str(train_sample)
 int [1:900] 415 463 179 526 195 938 818 118 299 229 ...
> x_train <- credit[train_sample, -17]
> x_test <- credit[-train_sample, -17]
> y_train <- credit[train_sample, 17]
> y_test <- credit[-train_sample, 17]
> library(C50)
> model <- C5.0(x_train,y_train)
> summary(model)

Call:
C5.0.default(x = x_train, y = y_train)


C5.0 [Release 2.07 GPL Edition]        Wed Oct 26 11:59:18 2022
-------------------------------

Class specified by attribute `outcome'
```



```
Decision tree:

checking_balance in {unknown,> 200 DM}: no (412/54)
checking_balance in {< 0 DM,1 - 200 DM}:
:...credit_history in {very good,perfect}:
    :...housing = rent: yes (16/1)
    :   housing = other:
    :   :...employment_duration in {1 - 4 years,4 - 7 years,
    :   :   :                       > 7 years}: yes (10)
    :   :   employment_duration in {< 1 year,unemployed}:
    :   :   :...job = skilled: yes (1)
    :   :       job in {management,unskilled,unemployed}: no (3)
    :   housing = own:
    :   :...purpose in {business,education}: no (9/2)
    :       purpose in {renovations,car0}: yes (3)
    :       purpose = car:
    :       :...months_loan_duration <= 18: yes (5)
    :       :   months_loan_duration > 18:
    :       :   :...job in {skilled,management,unemployed}: no (3)
    :       :       job = unskilled: yes (1)
    :       purpose = furniture/appliances:
    :       :...other_credit = store: no (4)
    :           other_credit in {none,bank}:
    :           :...existing_loans_count <= 1: yes (6)
    :               existing_loans_count > 1: no (2)
    credit_history in {good,critical,poor}:
    :...months_loan_duration <= 15:
        :...purpose in {business,car0}: no (10)
        :   purpose = furniture/appliances:
        :   :...savings_balance in {unknown,< 100 DM,500 - 1000 DM,
        :   :   :                       > 1000 DM}: no (88/14)
        :   :   savings_balance = 100 - 500 DM: yes (3)
        :   purpose = education:
        :   :...savings_balance = unknown: no (3)
        :   :   savings_balance in {< 100 DM,100 - 500 DM,500 - 1000 DM,
        :   :                       > 1000 DM}: yes (7/1)
        :   purpose = renovations:
        :   :...other_credit in {none,bank}: no (5/1)
        :   :   other_credit = store: yes (1)
        :   purpose = car:
        :   :...credit_history in {critical,poor}: no (27/4)
        :       credit_history = good:
        :       :...existing_loans_count > 1: yes (2)
```

```
:   :   :...checking_balance = < 0 DM: yes (4)
:   :       checking_balance = 1 - 200 DM: no (3/1)
:   purpose = furniture/appliances:
:   :...savings_balance in {100 - 500 DM,
:       :                   500 - 1000 DM}: yes (6)
:       savings_balance = < 100 DM:
:       :...months_loan_duration <= 22: yes (12/1)
:           months_loan_duration > 22:
:           :...amount <= 2325: yes (3)
:               amount > 2325: no (6)
employment_duration = 4 - 7 years:
:...savings_balance in {100 - 500 DM,
:   :                   500 - 1000 DM}: no (8)
:   savings_balance = < 100 DM:
:   :...job in {management,unskilled,
:       :       unemployed}: no (6)
:       job = skilled:
:       :...dependents > 1: no (3/1)
:           dependents <= 1:
:           :...months_loan_duration <= 22: no (3)
:               months_loan_duration > 22: yes (8)
employment_duration = > 7 years:
:...other_credit = store: no (2)
    other_credit = bank:
    :...job in {skilled,unemployed}: yes (6)
    :   job in {management,unskilled}: no (4/1)
    other_credit = none:
    :...purpose = business: yes (2)
        purpose in {education,renovations,
        :           car0}: no (1)
        purpose = furniture/appliances:
        :...job in {skilled,unskilled,unemployed}: no (9)
        :   job = management: yes (2)
        purpose = car:
        :...amount <= 6999: no (7/1)
            amount > 6999: [S1]

SubTree [S1]

checking_balance = < 0 DM: no (1)
checking_balance = 1 - 200 DM: yes (3)
```

```
Evaluation on training data (900 cases):

	    Decision Tree
	  ----------------
	  Size      Errors

	    69   99(11.0%)   <<


	   (a)   (b)    <-classified as
	  ----  ----
	   625    10    (a): class no
	    89   176    (b): class yes


	Attribute usage:

	100.00% checking_balance
	 54.22% credit_history
	 48.22% months_loan_duration
	 42.22% savings_balance
	 31.89% purpose
	 22.33% employment_duration
	  9.22% years_at_residence
	  8.78% housing
	  8.44% job
	  6.11% other_credit
```

```
	  5.78% amount
	  4.89% existing_loans_count
	  4.22% phone
	  2.89% percent_of_income
	  1.56% dependents
	  0.78% age


Time: 0.0 secs

> y_pred <- predict(model,x_test)
> cm <- table(y_pred,y_test)
> print(cm)
      y_test
y_pred no yes
   no  55  20
   yes 10  15
> acc<-sum(diag(cm))/sum(cm)
> print(paste("Accuaracy: ",acc*100,"%"))
[1] "Accuaracy:  70 %"
>
```

RESULT:

Thus the R program to find Risky Bank loans using Decision Tree is executed successfully and the output is verified.

Ex No: 5 Date:	**Medical Expense with Linear Regression.**

AIM:

To implement a R program to predict Medical Expense using Linear Regression

ALGORITHM:

1. Start
2. Load the Insurance dataset and analyse the structure of the dataset.
3. Get the summary statistics.Check whether the distribution is right-skewed or left skewed by comapring the mean and median. Verify the same using histogram.
4. Check the distribution of “region” using table.
5. Create a correlation matrix of “age”, “bmi”, “children”, “expenses”.
6. To determine the pattern of the dataset, use scatterplot using pairs() for “age”, “bmi”, “children”, “expenses”.
7. To display a more informative scatterplot use pairs.panel() from “psych” library.
8. Fit the linear regression model using lm() with expenses as the dependent variable.
9. Evaluate the model performance using summary().
10. To improve the model performance, square the age variable as age2 and bmi30 is 1 if bmi>=30 else 0.
11. Train the model with age + age2+bmi30 as also as the independent variables.
12. Evaluate the model performance for model2 using summary().
13. Stop.

PROGRAM:

```
insurance<-read.csv("insurance.csv",stringsAsFactors = TRUE)

str(insurance)

summary(insurance$expenses)

hist(insurance$expenses)

table(insurance$region)

cor(insurance[c("age","bmi","children","expenses")])

pairs(insurance[c("age","bmi","children","expenses")])

library(psych)

pairs.panels(insurance[c("age","bmi","children","expenses")])8

ins_model <- lm(expenses ~ age  + children + bmi +  sex + smoker + region, data =
insurance)

ins_model
```

summary(ins_model)

insurance$age2 <- insurance$age^2

insurance$bmi30 <- ifelse(insurance$bmi >= 30,1,0)

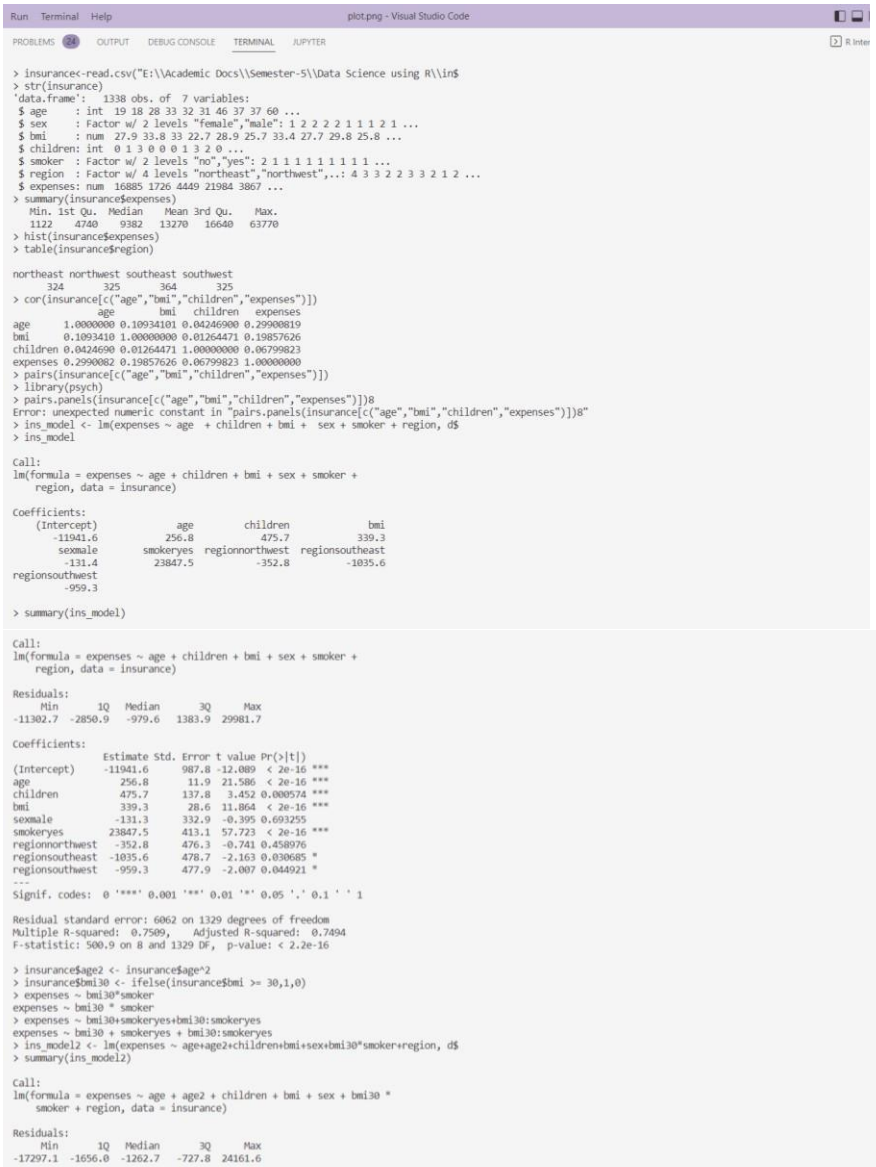
expenses ~ bmi30*smoker

expenses ~ bmi30+smokeryes+bmi30:smokeryes

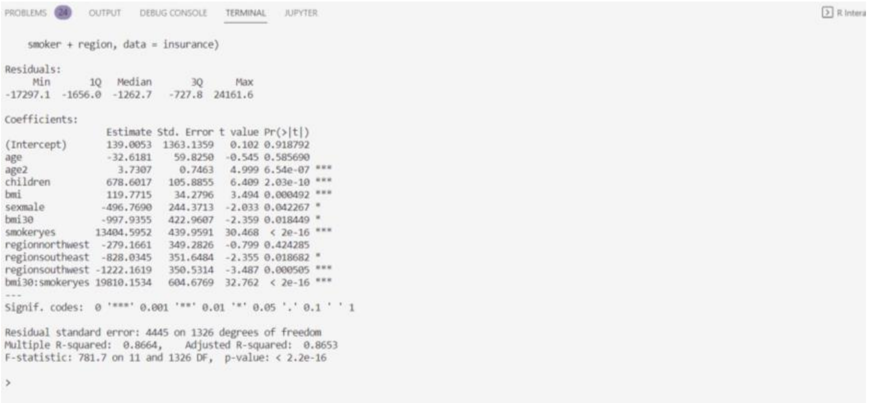
ins_model2 <- lm(expenses ~ age+age2+children+bmi+sex+bmi30*smoker+region, data=insurance)

summary(ins_model2)

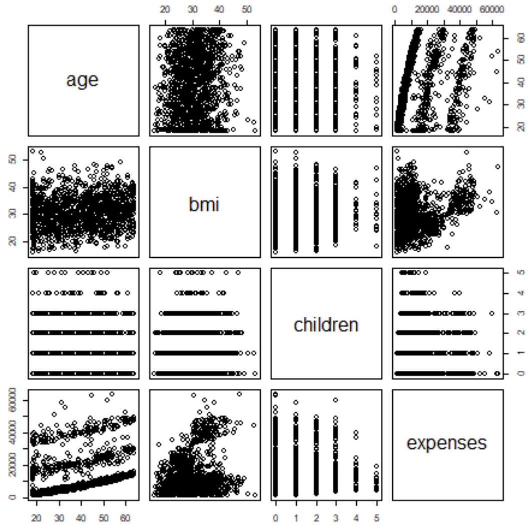
OUTPUT:

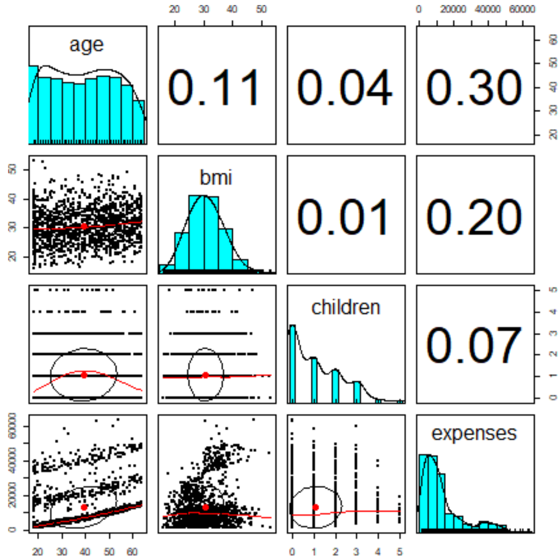


```
Run   Terminal   Help          plot.png - Visual Studio Code
PROBLEMS 24   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER          R Inter

> insurance<-read.csv("E:\\Academic Docs\\Semester-5\\Data Science using R\\in$
> str(insurance)
'data.frame':   1338 obs. of  7 variables:
 $ age     : int  19 18 28 33 32 31 46 37 37 60 ...
 $ sex     : Factor w/ 2 levels "female","male": 1 2 2 2 2 1 1 1 2 1 ...
 $ bmi     : num  27.9 33.8 33 22.7 28.9 25.7 33.4 27.7 29.8 25.8 ...
 $ children: int  0 1 3 0 0 0 1 3 2 0 ...
 $ smoker  : Factor w/ 2 levels "no","yes": 2 1 1 1 1 1 1 1 1 1 ...
 $ region  : Factor w/ 4 levels "northeast","northwest",..: 4 3 3 2 2 3 3 2 1 2 ...
 $ expenses: num  16885 1726 4449 21984 3867 ...
> summary(insurance$expenses)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1122    4740    9382   13270   16640   63770
> hist(insurance$expenses)
> table(insurance$region)

northeast northwest southeast southwest
      324       325       364       325
> cor(insurance[c("age","bmi","children","expenses")])
               age        bmi   children   expenses
age      1.0000000 0.10934101 0.04246900 0.29900819
bmi      0.1093410 1.00000000 0.01264471 0.19857626
children 0.0424690 0.01264471 1.00000000 0.06799823
expenses 0.2990082 0.19857626 0.06799823 1.00000000
> pairs(insurance[c("age","bmi","children","expenses")])
> library(psych)
> pairs.panels(insurance[c("age","bmi","children","expenses")])8
Error: unexpected numeric constant in "pairs.panels(insurance[c("age","bmi","children","expenses")])8"
> ins_model <- lm(expenses ~ age  + children + bmi +  sex + smoker + region, d$
> ins_model

Call:
lm(formula = expenses ~ age + children + bmi + sex + smoker +
    region, data = insurance)

Coefficients:
    (Intercept)              age         children              bmi
       -11941.6            256.8            475.7            339.3
        sexmale        smokeryes  regionnorthwest  regionsoutheast
         -131.4          23847.5           -352.8          -1035.6
regionsouthwest
         -959.3

> summary(ins_model)

Call:
lm(formula = expenses ~ age + children + bmi + sex + smoker +
    region, data = insurance)

Residuals:
     Min       1Q   Median       3Q      Max
-11302.7  -2850.9   -979.6   1383.9  29981.7

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     -11941.6      987.8 -12.089  < 2e-16 ***
age                256.8       11.9  21.586  < 2e-16 ***
children           475.7      137.8   3.452 0.000574 ***
bmi                339.3       28.6  11.864  < 2e-16 ***
sexmale           -131.3      332.9  -0.395 0.693255
smokeryes        23847.5      413.1  57.723  < 2e-16 ***
regionnorthwest   -352.8      476.3  -0.741 0.458976
regionsoutheast  -1035.6      478.7  -2.163 0.030685 *
regionsouthwest   -959.3      477.9  -2.007 0.044921 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6062 on 1329 degrees of freedom
Multiple R-squared:  0.7509,    Adjusted R-squared:  0.7494
F-statistic: 500.9 on 8 and 1329 DF,  p-value: < 2.2e-16

> insurance$age2 <- insurance$age^2
> insurance$bmi30 <- ifelse(insurance$bmi >= 30,1,0)
> expenses ~ bmi30*smoker
expenses ~ bmi30 * smoker
> expenses ~ bmi30+smokeryes+bmi30:smokeryes
expenses ~ bmi30 + smokeryes + bmi30:smokeryes
> ins_model2 <- lm(expenses ~ age+age2+children+bmi+sex+bmi30*smoker+region, d$
> summary(ins_model2)

Call:
lm(formula = expenses ~ age + age2 + children + bmi + sex + bmi30 *
    smoker + region, data = insurance)

Residuals:
     Min       1Q   Median       3Q      Max
-17297.1  -1656.0  -1262.7   -727.8  24161.6
```



```
PROBLEMS 24   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

    smoker + region, data = insurance)

Residuals:
     Min       1Q   Median       3Q      Max
-17297.1  -1656.0  -1262.7   -727.8  24161.6

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       139.0053  1363.1359   0.102 0.918792
age               -32.6181    59.8250  -0.545 0.585690
age2                3.7307     0.7463   4.999 6.54e-07 ***
children          678.6017   105.8855   6.409 2.03e-10 ***
bmi               119.7715    34.2796   3.494 0.000492 ***
sexmale          -496.7690   244.3713  -2.033 0.042267 *
bmi30            -997.9355   422.9607  -2.359 0.018449 *
smokeryes       13404.5952   439.9591  30.468  < 2e-16 ***
regionnorthwest  -279.1661   349.2826  -0.799 0.424285
regionsoutheast  -828.0345   351.6484  -2.355 0.018682 *
regionsouthwest -1222.1619   350.5314  -3.487 0.000505 ***
bmi30:smokeryes 19810.1534   604.6769  32.762  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4445 on 1326 degrees of freedom
Multiple R-squared:  0.8664,    Adjusted R-squared:  0.8653
F-statistic: 781.7 on 11 and 1326 DF,  p-value: < 2.2e-16

>
```





RESULT:

Thus the R program to predict medical expenses using linear regression is executed successfully and the output is verified.

Ex No:6 Date:	**Modeling the strength of concrete with ANNs**

AIM:

To create a R program to implement Modeling steength of concrete with ANNs.

ALGORITHM:

1. Start
2. Read the dataset “Concrete” using read.csv() function and store it in concrete variable.
3. To perform calculations with ease, normalise the data using an user-defined function ‘normalise’ and apply it to the dataset using lapply() and store it to concrete_norm.
4. Compare the strength values of concrete and concrete_norm to check if the data is normalised.
5. Split the dataset for training and testing in the ratio of 75:25.
6. To use ANN, import the library ‘neuralnet’.
7. Train the model using the neuralnet() with strength as the response variable and the rest as predictor variables from the training dataset, with only one hidden node.
8. Visualise the neural network topology by plotting the model.
9. To generate predictions on the test dataset, use the compute() which returns $neurons and $net.result which stores the predicted values.
10. Store the predicted values from model_result$net.result into predicted_strength.
11. To analyse the relationship between predicted strength and the true value, display the correlation matrix using cor().
12. To improve the model performance, develop a new neuralnet() model with 5 hidden nodes.
13. Plot the network again to see a drastic increase in no of connections.
14. Evaluate the second model and display the correlation matrix.
15. Stop.

CODE:

```
concrete<-read.csv(".\Concrete_Data.csv") View(concrete)

str(concrete)

normalize<-function(x){

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

concrete_norm<-as.data.frame(lapply(concrete,normalize))

summary((concrete_norm$strength))

summary(concrete$strength) concrete_train<-

concrete_norm[1:773,] concrete_test<-concrete_norm[774:1030,]

library(neuralnet)
```

concrete_model<-neuralnet(strength~cement+slag+ash+water+superplasticizer

+coarseagg+fineagg+age, data=concrete_train)

plot(concrete_model)

model_results<-compute(concrete_model,concrete_test[1:8]) predicted_strength<-

model_results$net.result cor(predicted_strength,concrete_test$strength)

concrete_model2<-neuralnet(strength~cement+slag+ash+water+superplasticizer

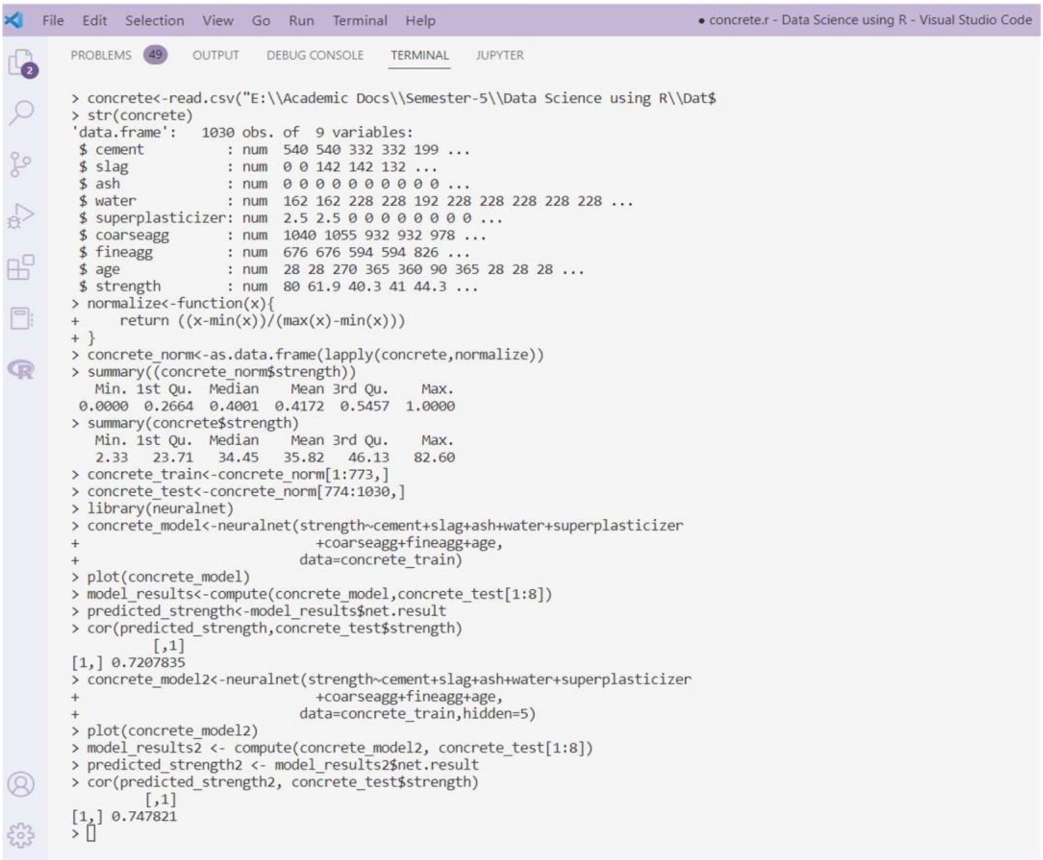
+coarseagg+fineagg+age, data=concrete_train,hidden=5)

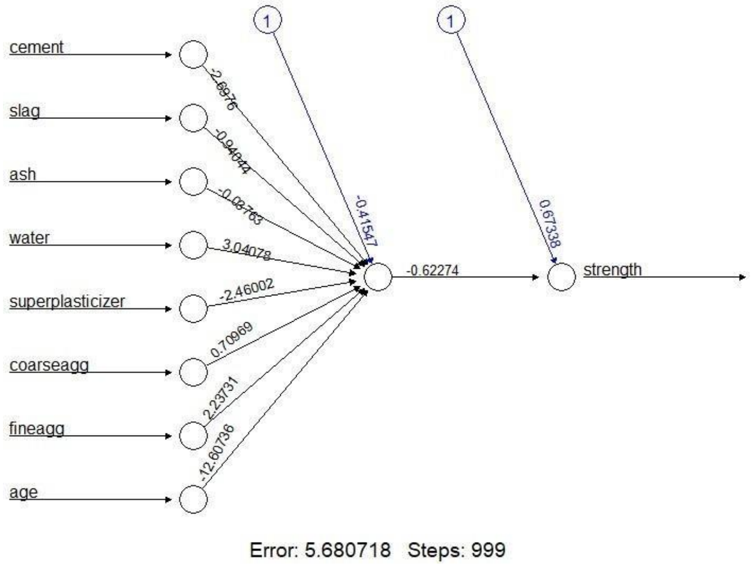
plot(concrete_model2)

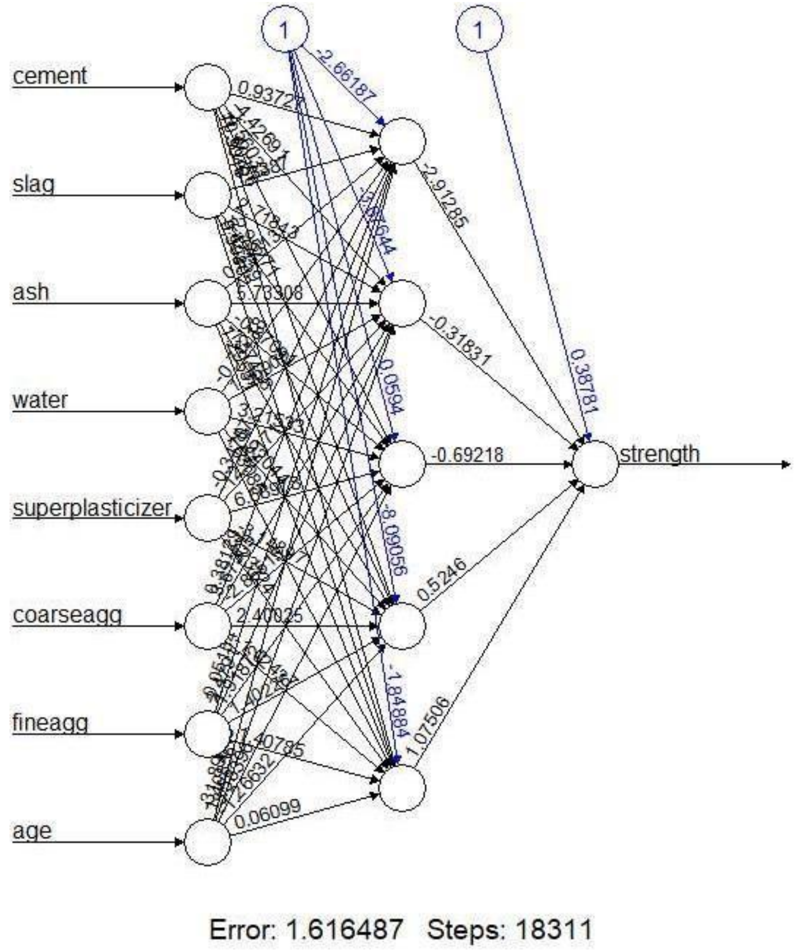
model_results2 <- compute(concrete_model2, concrete_test[1:8])

predicted_strength2 <- model_results2$net.result

cor(predicted_strength2, concrete_test$strength) **<u>OUTPUT:</u>**







RESULT:

Thus the R Program to implement modeling the strength of concrete using ANNs is executed successfully and the output is verified.

Ex No:7 **Date:**	**Identification of frequently Purchased groceries with Apriori**

AIM:

To implement a R program to identify frequently purchased groceries with Apriori algorithm.

ALGORITHM:

1. Start
2. Import the ‘arules’ library to use Apriori algorithm and the Groceries dataset.
3. Import the dataset ‘Groceries’ from arules library using data(). The preloaded dataset is a sparse matrix
4. To do EDA, use the summary() function for the dataset.
5. To see the contents of the sparse matrix, use inspect().
6. To view the proportion of transactions of the items, use itemFrequency().
7. To view the proportion of transactions visually, we use itemFrequencyPlot() with 10 percent support.
8. To limit the plot a specific number of items, use the topN parameter.
9. To visualize the entire sparse matrix we use image() function for random 100 transactions.
10. Train the apriori model using apriori(), with confidence threshold of 25% , set support level of 0.006 which is 60 transactions(2 per day) out of 9835, and to eliminate rules fewer than 2.
11. Summarise the groceryrules from the model.
12. Inspect the first 3 groceryrules and deduce the meaning of it.
13. To inspect which customer is most likely to buy a item relative to average customer we sort the top 5 groceryrules in the order of lift.
14. To find the rules of only berries use the subset() in the transactions, items or rules and save it as a dataframe.
15. Stop

CODE:

```
library(arules) data(Groceries)
summary(Groceries)
inspect(Groceries[1:5])
itemFrequency(Groceries[,1:3])
itemFrequencyPlot(Groceries,
support=0.1)
itemFrequencyPlot(Groceries, topN = 20)
image(Groceries[1:5])
image(sample(Groceries,100))
apriori(Groceries)
groceryrules <- apriori(Groceries, parameter=list(support=0.006, confidence =
0.25, minlen=2)) groceryrules
summary(groceryrules) inspect(groceryrules[1:3])
inspect(sort(groceryrules, by = "lift")[1:5]) berryrules <-
```

subset(groceryrules, items %in% "berries") inspect(berryrules) write(groceryrules, file = "groceryrules.csv") groceryrules_df <- as(groceryrules, "data.frame") str(groceryrules_df)

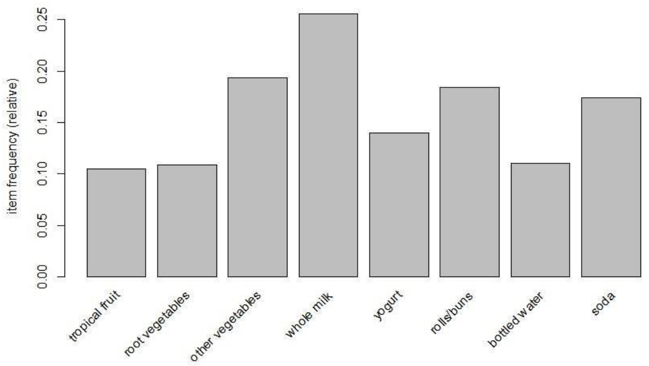
OUTPUT:

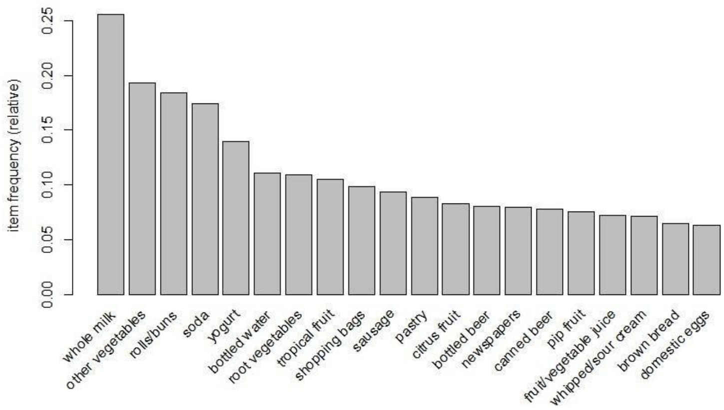
```
PROBLEMS 7   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

> library(arules)
> data(Groceries)
> summary(Groceries)
transactions as itemMatrix in sparse format with
 9835 rows (elements/itemsets/transactions) and
 169 columns (items) and a density of 0.02609146

most frequent items:
      whole milk other vegetables       rolls/buns             soda
            2513             1903             1809             1715
          yogurt          (Other)
            1372            34055

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46
  17   18   19   20   21   22   23   24   26   27   28   29   32
  29   14   14    9   11    4    6    1    1    1    1    3    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   4.409   6.000  32.000

includes extended item information - examples:
       labels  level2           level1
1 frankfurter sausage meat and sausage
2     sausage sausage meat and sausage
3  liver loaf sausage meat and sausage
> inspect(Groceries[1:5])
    items
[1] {citrus fruit,
     semi-finished bread,
     margarine,
     ready soups}
[2] {tropical fruit,
     yogurt,
     coffee}
[3] {whole milk}
[4] {pip fruit,
     yogurt,
     cream cheese ,
     meat spreads}
[5] {other vegetables,
     whole milk,

     long life bakery product}
> itemFrequency(Groceries[,1:3])
frankfurter     sausage  liver loaf
0.058973055 0.093950178 0.005083884
> itemFrequencyPlot(Groceries, support=0.1)
> itemFrequencyPlot(Groceries, topN = 20)
> image(Groceries[1:5])
> image(sample(Groceries,100))
> apriori(Groceries)
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.8    0.1    1 none FALSE            TRUE       5     0.1      1
 maxlen target  ext
     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 983

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
sorting and recoding items ... [8 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [0 rule(s)] done [0.00s].
```

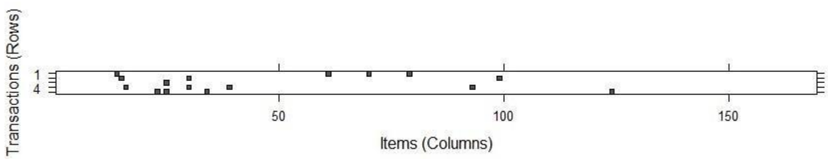
```
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [0 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
set of 0 rules
> groceryrules <- apriori(Groceries, parameter=list(support=0.006, confidence $
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
       0.25    0.1    1 none FALSE            TRUE       5   0.006      2
 maxlen target  ext
     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 59
```

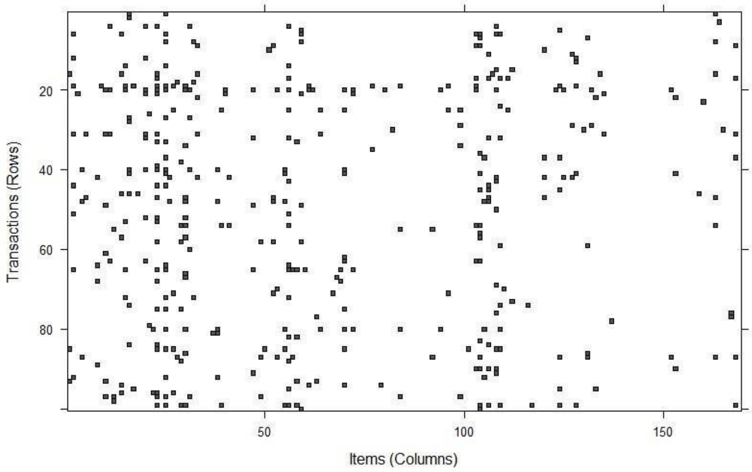
```
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
sorting and recoding items ... [109 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.01s].
writing ... [463 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> groceryrules
set of 463 rules
> summary(groceryrules)
set of 463 rules

rule length distribution (lhs + rhs):sizes
  2   3   4
150 297  16

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   2.000   3.000   2.711   3.000   4.000

summary of quality measures:
    support           confidence        coverage            lift
 Min.   :0.006101   Min.   :0.2500   Min.   :0.009964   Min.   :0.9932
 1st Qu.:0.007117   1st Qu.:0.2971   1st Qu.:0.018709   1st Qu.:1.6229
 Median :0.008744   Median :0.3554   Median :0.024809   Median :1.9332
 Mean   :0.011539   Mean   :0.3786   Mean   :0.032608   Mean   :2.0351
 3rd Qu.:0.012303   3rd Qu.:0.4495   3rd Qu.:0.035892   3rd Qu.:2.3565
 Max.   :0.074835   Max.   :0.6600   Max.   :0.255516   Max.   :3.9565
     count
 Min.   : 60.0
 1st Qu.: 70.0
 Median : 86.0
 Mean   :113.5
 3rd Qu.:121.0
 Max.   :736.0

mining info:
      data ntransactions support confidence
 Groceries          9835   0.006       0.25
                                                                              call
 apriori(data = Groceries, parameter = list(support = 0.006, confidence = 0.25, minlen = 2))
> inspect(groceryrules[1:3])
    lhs              rhs                support     confidence coverage
[1] {pot plants} => {whole milk}       0.006914082 0.4000000  0.01728521
[2] {pasta}      => {whole milk}       0.006100661 0.4054054  0.01504830
[3] {herbs}      => {root vegetables} 0.007015760 0.4312500  0.01626843
```

```
    lift     count
[1] 1.565460 68
[2] 1.586614 60
[3] 3.956477 69
> inspect(sort(groceryrules, by = "lift")[1:5])
    lhs                    rhs                      support confidence    coverage     lift count
[1] {herbs}             => {root vegetables}    0.007015760  0.4312500 0.01626843 3.956477    69
[2] {berries}           => {whipped/sour cream} 0.009049314  0.2721713 0.03324860 3.796886    89
[3] {tropical fruit,
     other vegetables,
     whole milk}        => {root vegetables}    0.007015760  0.4107143 0.01708185 3.768074    69
[4] {beef,
     other vegetables}  => {root vegetables}    0.007930859  0.4020619 0.01972547 3.688692    78
[5] {tropical fruit,
     other vegetables}  => {pip fruit}          0.009456024  0.2634561 0.03589222 3.482649    93
> berryrules <- subset(groceryrules, items %in% "berries")
> inspect(berryrules)
    lhs          rhs                  support     confidence coverage  lift
[1] {berries} => {whipped/sour cream} 0.009049314 0.2721713  0.0332486 3.796886
[2] {berries} => {yogurt}             0.010574479 0.3180428  0.0332486 2.279848
[3] {berries} => {other vegetables}   0.010269446 0.3088685  0.0332486 1.596280
[4] {berries} => {whole milk}         0.011794611 0.3547401  0.0332486 1.388328
    count
[1]  89
[2] 104
[3] 101
[4] 116
> write(groceryrules,  file = "groceryrules.csv")
> groceryrules_df <- as(groceryrules, "data.frame")
> str(groceryrules_df)
'data.frame':   463 obs. of  6 variables:
 $ rules     : chr  "{pot plants} => {whole milk}" "{pasta} => {whole milk}" "{herbs} => {root vegetables}" "{herbs} => {other vegetables}" ...
 $ support   : num  0.00691 0.0061 0.00702 0.00773 0.00773 ...
 $ confidence: num  0.4 0.405 0.431 0.475 0.475 ...
 $ coverage  : num  0.0173 0.015 0.0163 0.0163 0.0163 ...
 $ lift      : num  1.57 1.59 3.96 2.45 1.86 ...
 $ count     : int  68 60 69 76 76 69 70 67 63 88 ...
> 
```









RESULT:

Thus the R program to identify frequently purchased groceries using Apriori algorithm is executed successfully and the output is verified.

Ex. No: 8	**Identification of frequently Purchased groceries with Apriori algorithm**
Date :	

Aim:

To identify frequently purchased groceries using apriori.

Algorithm:

Step 1: Import the 'arules' package.

Step 2: Create a list of transaction datasets. Each transaction is represented as a list of items.

Step 3: Convert the transaction data into a binary format suitable for association rule mining.

Step 4: Use the Apriori algorithm to mine association rules with specified support and confidence thresholds.

Step 5: Display the frequent itemsets and association rules.

Coding:

```
library(arules)

transactions <- list(

  c("bread", "milk", "cereal"),

  c("bread", "milk"),

  c("bread", "cereal"),

  c("bread", "milk", "cereal"),

  c("eggs", "milk", "cereal")

)

transactions <- as(transactions, "transactions")

rules <- apriori(transactions, parameter = list(support = 0.3, confidence = 0.7))

inspect(rules)
```

Output:

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen m
        0.7    0.1    1 none FALSE            TRUE       5     0.3      1

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 1

set item appearances ...[0 item(s)] done [0s].
set transactions ...[5 item(s), 5 transaction(s)] done [0s].
sorting and recoding items ... [4 item(s)] done [0s].
creating transaction tree ... done [0s].
checking subsets of size 1 2 done [0s].
writing ... [3 rule(s)] done [0s].
creating S4 object  ... done [0s].

     lhs         rhs      support confidence lift count
[1]  {bread} => {cereal} 0.4     1          2.5  2
[2]  {cereal} => {bread} 0.4     1          2.5  2
[3]  {milk}  => {bread}  0.6     1          2.5  3
```

Result:

Thus, the identification of frequently purchased groceries is executed successfully.

Ex. No: 9	**Finding Teen Segments of Market**
Date :	

Aim:

To implement Teen segments of market in R.

Algorithm:

Step 1: Load the 'ggplot2' library to enable data visualization.

Step 2: Set a random seed for reproducibility.

Step 3: Generate sample data for demonstration purposes. In this case, create a data frame called 'data' with 100 random values for "Age" and "Spending."

Step 4: Specify the number of clusters (k) for K-Means clustering. In this example, k is set to 2.

Step 5: Perform K-Means clustering on the 'data' using the specified number of clusters (k).

Step 6: Add a new column, "Cluster," to the data frame, representing the cluster assignments for each data point.

Step 7: Display the data in a tabular format. Print the column headers for "Age," "Spending," and "Cluster."

Step 8: Iterate through each row of the data frame and print the "Age," "Spending," and "Cluster" values in a tabular format.

Coding:

```
library(ggplot2)

set.seed(123)

data <- data.frame(

  Age = runif(100, min = 13, max = 19),

  Spending = rnorm(100, mean = 50, sd = 10)

)

k <- 2

kmeans_result <- kmeans(data, centers = k)

data$Cluster <- as.factor(kmeans_result$cluster)
```

```
cat("Age   | Spending | Cluster\n")

cat("----|-------|-----\n")

for (i in 1:nrow(data)) {

  cat(sprintf("%.2f | %.2f      | %s\n", data$Age[i], data$Spending[i], data$Cluster[i]))

}
```

Output:

```
Age   | Spending | Cluster
------|----------|--------
15.27 | 53.05    | 2
15.82 | 47.96    | 2
16.20 | 47.96    | 2
15.66 | 44.66    | 2
15.59 | 49.66    | 2
16.56 | 47.18    | 2
```

Result:

Thus, the R program for teen segment of market is executed successfully.

Ex. No: 10	**TUNING STOCK MODELS**
Date :	

Aim:

To implement tuning stock models for better performance.

Algorithm:

Step 1: Load the 'quantmod' and 'caret' libraries, which are necessary for stock data retrieval and model tuning.

Step 2: Download historical stock data using the `getSymbols` function, specifying the desired stock symbol (e.g., "AAPL").

Step 3: Create a data frame named 'stock_data' to store the historical stock data, including dates and adjusted closing prices.

Step 4: Define the training and test data sets by selecting a specific range of rows from the 'stock_data' data frame.

Step 5: Create time series objects ('train_ts' and 'test_ts') using the 'xts' function to work with the adjusted closing prices of the training and test data.

Step 6: Define a control object ('ctrl') for model tuning, specifying parameters like the resampling method (cross-validation with 5 folds).

Step 7: Tune a stock prediction model (in this example, a simple linear regression) using the 'train' function. The model is trained on the 'train_ts' data, with the dependent variable as 'train_ts' and the independent variable as the index of 'train_ts.' The model tuning is performed under the specified control settings.

Step 8: Print the results of the tuned model, including information about the model's performance, such as root mean squared error (RMSE) and R-squared values, as well as any tuned parameters.

Coding:

```
library(quantmod)

library(caret)

getSymbols("AAPL")

stock_data <- data.frame(Date = index(AAPL), Adjusted = Ad(AAPL))
```

train_data <- stock_data[1:200,]

test_data <- stock_data[201:nrow(stock_data),]

train_ts <- xts(train_data$Adjusted, order.by = train_data$Date)

test_ts <- xts(test_data$Adjusted, order.by = test_data$Date)

ctrl <- trainControl(method = "cv", number = 5)

tune_results <- train(train_ts ~ index(train_ts), data = data.frame(train_ts), method = "lm", trControl = ctrl)

print(tune_results)

Output:

```
Linear Regression

200 samples
  1 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 160, 160, 160, 160, 160
Resampling results:

  RMSE      Rsquared
  10.98207  0.8478949

Tuning parameter 'intercept' was held constant at a value of TRUE
```

Result:

Thus, the implementation of tuning the stocks for better performance is executed successfully.