GR20 Regulations

III B.Tech I Semester

Machine Learning

with

R Programming Lab

(GR20A3079)

Department of Artificial Intelligence and Machine Learning

Engineering

(CSBS)

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

SYLLABUS

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

MACHINE LEARNING WITH R PROGRAMMING LAB

CourseCode:GR20A3079 **L/T/P/C: 0/0/3/1.5**

III Year I Semester

Course Objectives:

1. To used R and Weka tools for experimenting machine learning techniques
2. To introduce basic concepts of data preprocessing techniques
3. To understand the Supervised and Unsupervised learning techniques
4. To study the various statistical techniques
5. To understand the performance of the various machine learning techniques

Course Outcomes:

1. Distinguish between, supervised, unsupervised and semi-supervised learning
2. Apply the appropriate machine learning strategy for any given problem
3. Ability to get the skill to apply machine learning techniques to address the real time problems in different areas
4. Modify existing machine learning algorithms to improve classification efficiency
5. Compare the performance of various machine learning algorithms

TASK-1

(a): Introduction to WEKA

(b): Reading ARFF, CSV files and apply preprocessing techniques in Weka (UCI datasets)

TASK-2

(a): Introduction to R Programming

(b): Read dataset and practice basic programming concepts

TASK-3

(a): Perform Data exploration and pre-processing in R (use any UCI dataset)

(b): Perform Feature Engineering and Feature Selection Methods in R

TASK-4

(a): Implement regularized Linear regression in R (use any UCI dataset)

(b): Implement regularized logistic regression in R

TASK-5:

Implement Apriori association rule mining algorithm in R and Weka

TASK-6:

Implement K- means clustering algorithm in R and Weka

TASK-7:

Implement Decision Tree classification algorithm in R and Weka

TASK-8:

Implement Distance and density-based anomaly detection (K-NN) algorithm

TASK-9:

Implement Normal distribution in R programming for any dataset.

TASK-10:

Implement Expectation Maximization in R programming to solve exponential

distribution

TASK-11:

Apply EM algorithm to cluster a set of data stored in a .CSV file and use the same data set for clustering using k-Means algorithm in R. Compare the results of these two algorithms and comment on the quality of clustering

TASK-12:

Write R program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

TEXTBOOKS:

1. A First Course in Statistical Programming with R, W Braun (You?)|2016.

2. Data Analytics Using R Paperback – 28 April 2018 by Seema Acharya (Author).

3. Machine Learning with R: Expert techniques for predictive modeling, 3rd Edition Paperback – Import, 15 April 2019 by Brett Lantz (Author).

REFERENCE BOOKS:

1. Beyond Spreadsheets with R: A beginner's guide to R and RStudio 1st Edition by Dr Jonathan Carroll (Author)

2. Data Science and Machine Learning with R Paperback – 30 July 2021 by Reema Thareja (Author)

3. R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, 2/e, Wiley, 2001.

4. C. Bishop, Pattern Recognition and Machine Learning, Springer, 2007.

5. E. Alpaydin, Introduction to Machine Learning, 3/e, Prentice-Hall, 2014.

6. A. Rostamizadeh, A. Talwalkar, M. Mohri, Foundations of Machine Learning, MIT Press.

7. A. Webb, Statistical Pattern Recognition, 3/e, Wiley, 2011.

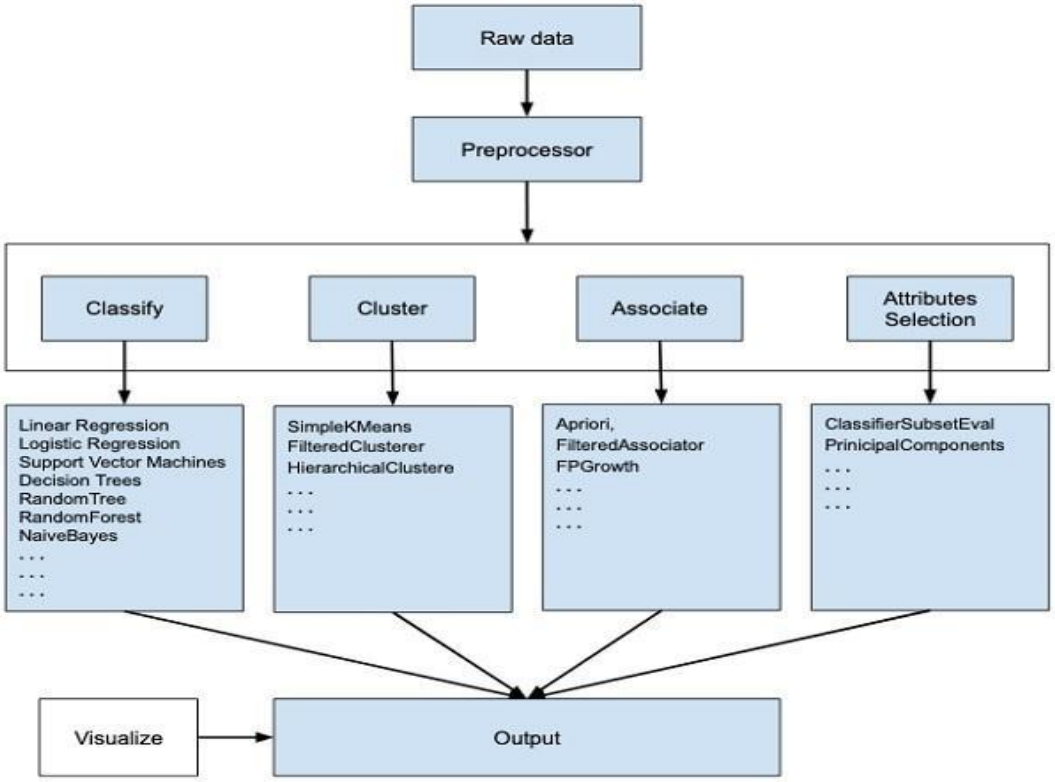
INDEX

TASK 1

(a): Introduction to WEKA.
(b): Reading ARFF, CSV files and apply preprocessing techniques in Weka (UCI datasets).

(a): Introduction to WEKA.

WEKA - an open-source software provides tools for data preprocessing, implementation of several Machine Learning algorithms, and visualization tools so that you can develop machine learning techniques and apply them to real-world data mining problems. What WEKA offers is summarized in the following diagram –



If you observe the beginning of the flow of the image, you will understand that there are many stages in dealing with Big Data to make it suitable for machine learning −

First, you will start with the raw data collected from the field. This data may contain several null values and irrelevant fields. You use the data preprocessing tools provided in WEKA to cleanse the data.

Then, you would save the preprocessed data in your local storage for applying ML algorithms.

Next, depending on the kind of ML model that you are trying to develop you would select one of the options such as Classify, Cluster, or Associate. The Attributes Selection allows the automatic selection of features to create a reduced dataset.

Note that under each category, WEKA provides the implementation of several algorithms. You would select an algorithm of your choice, set the desired parameters and run it on the dataset.

Then, WEKA would give you the statistical output of the model processing. It provides you a visualization tool to inspect the data.

The various models can be applied on the same dataset. You can then compare the outputs of

different models and select the best that meets your purpose.

To install WEKA on your machine, visit WEKA's official website and download the installation file. WEKA supports installation on Windows, Mac OS X and Linux.

You just need to follow the instructions on this page to install WEKA for your OS.

The steps for installing on Windows are as follows −

Download the Windows installation file.

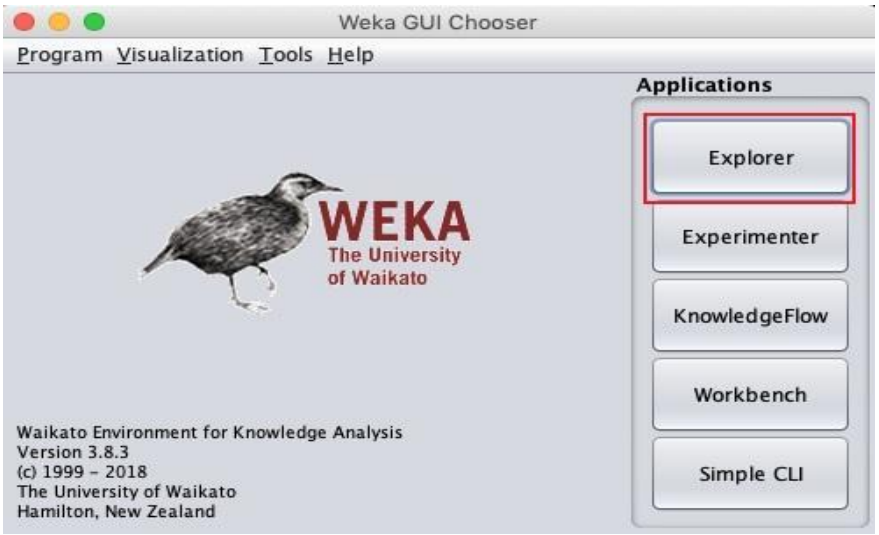
Double click on the downloaded weka-3-8-3-corretto-jvm.dmg file.

You will see the following screen on successful installation.

Click on the weak-3-8-3-corretto-jvm icon to start Weka.

Optionally you may start it from the command line −

java -jar weka.jar

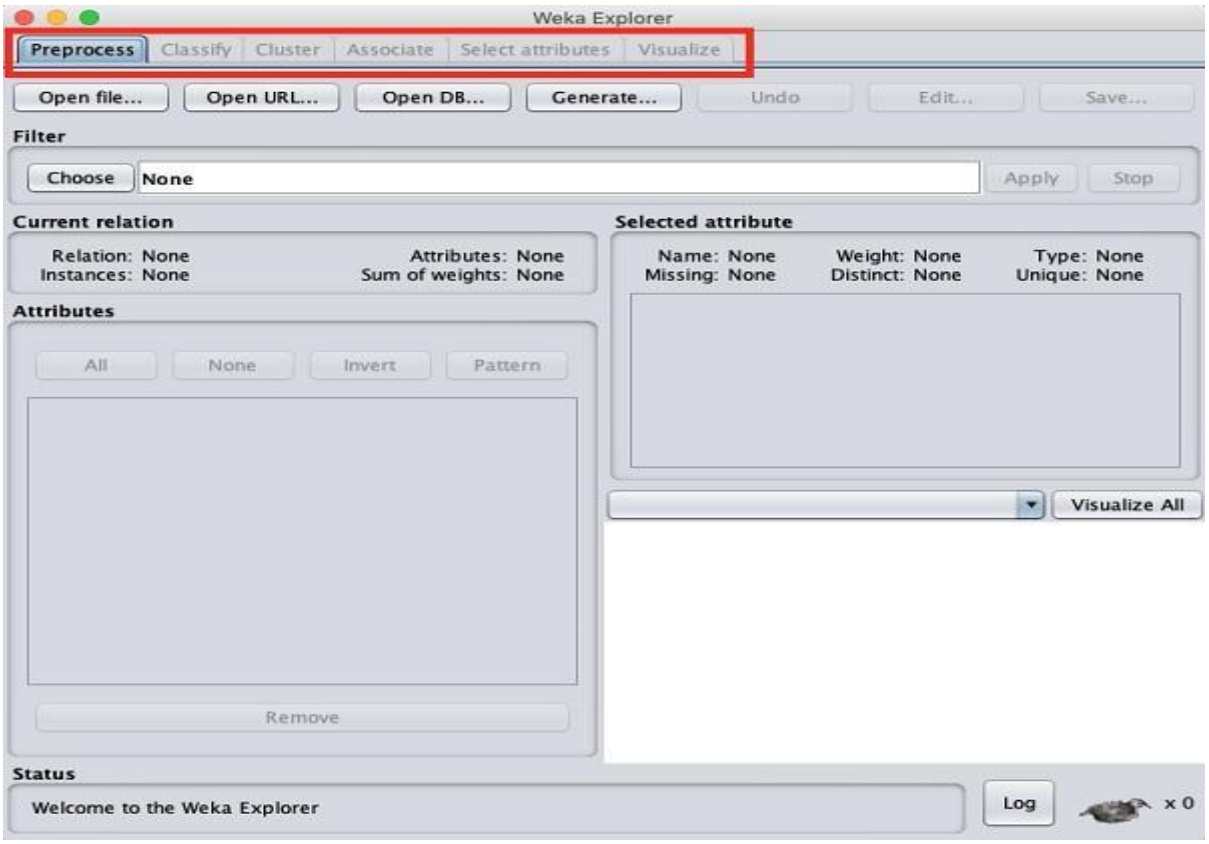


The GUI Chooser application allows you to run five different types of applications as listed

here

a. Explorer
b. Experimenter
c. KnowledgeFlow
d. Workbench
e. Simple CLI

When you click on the Explorer button in the Applications selector, it opens the following screen.



On the top, you will see several tabs as listed here −

- Preprocess
- Classify
- Cluster
- Associate
- Select Attributes
- Visualize

Under these tabs, there are several pre-implemented machine learning algorithms.

Preprocess Tab

Initially as you open the explorer, only the **Preprocess** tab is enabled. The first step in machine learning is to preprocess the data. Thus, in the **Preprocess** option, you will select the data file, process it and make it fit for applying the various machine learning algorithms.

Classify Tab

The **Classify** tab provides you several machine learning algorithms for the classification of your data. To list a few, you may apply algorithms such as Linear Regression, Logistic Regression, Support Vector Machines, Decision Trees, RandomTree, RandomForest, NaiveBayes, and so on. The list is very exhaustive and provides both supervised and unsupervised machine learning algorithms.

Cluster Tab

Under the **Cluster** tab, there are several clustering algorithms provided - such as SimpleKMeans, FilteredClusterer, HierarchicalClusterer, and so on.

Associate Tab

Under the **Associate** tab, you would find Apriori, FilteredAssociator and FPGrowth.

Select Attributes Tab

Select Attributes allows you feature selections based on several algorithms such as ClassifierSubsetEval, PrinicipalComponents, etc.

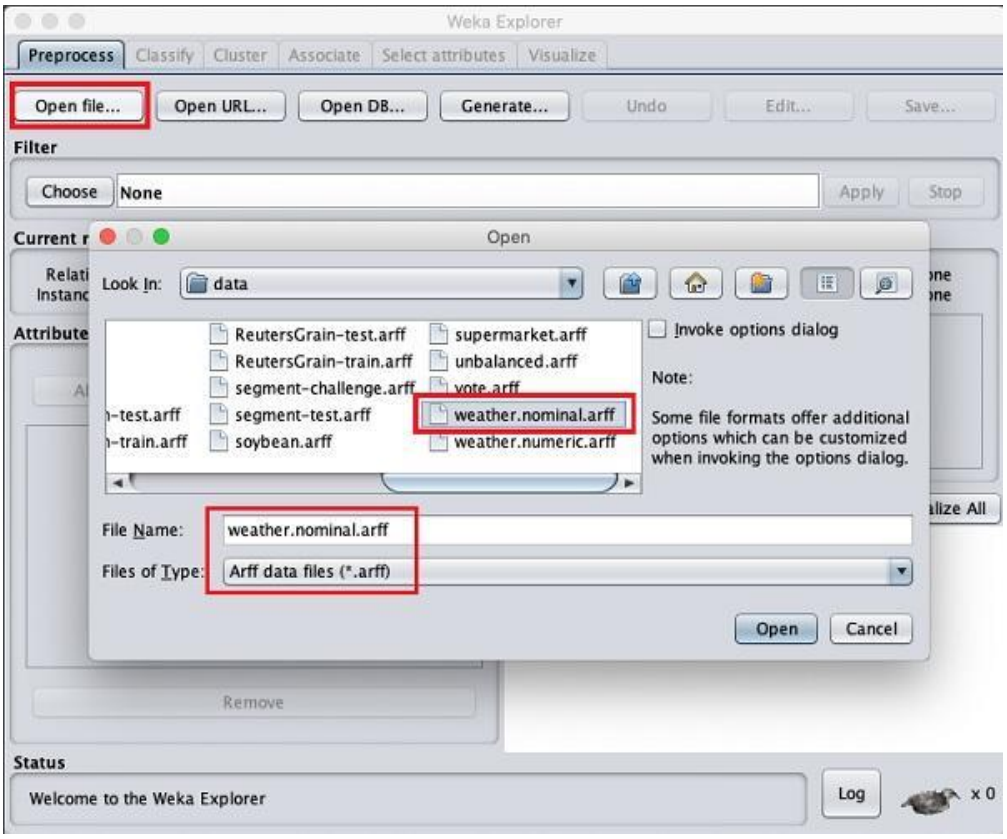
Visualize Tab

Lastly, the **Visualize** option allows you to visualize your processed data for analysis.

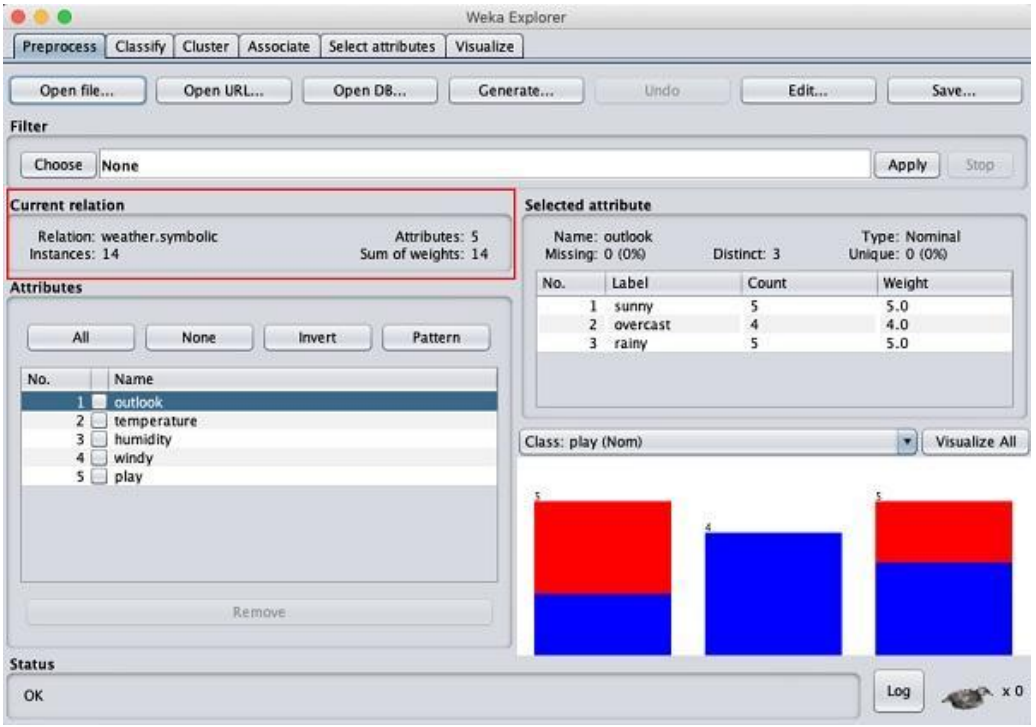
(b): Reading ARFF, CSV files and apply preprocessing techniques in Weka (UCI datasets)

To demonstrate the available features in preprocessing, we will use the Weather database that is provided in the installation.

Using the Open file ... option under the Preprocess tag select the weather-nominal.arff file.



When you open the file, your screen looks like as shown here −



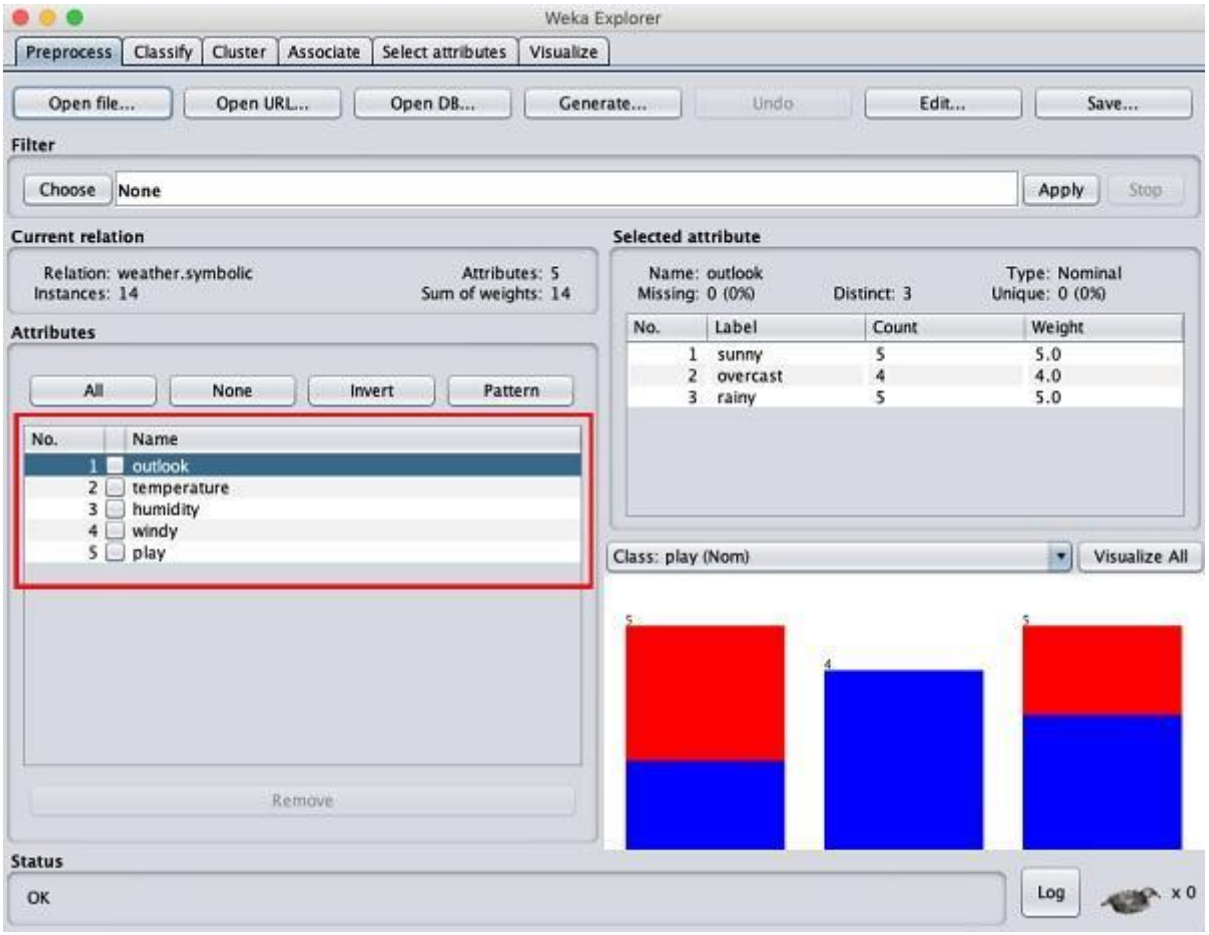
Understanding Data

Let us first look at the highlighted Current relation sub window. It shows the name of the database that is currently loaded. You can infer two points from this sub window −

There are 14 instances - the number of rows in the table.

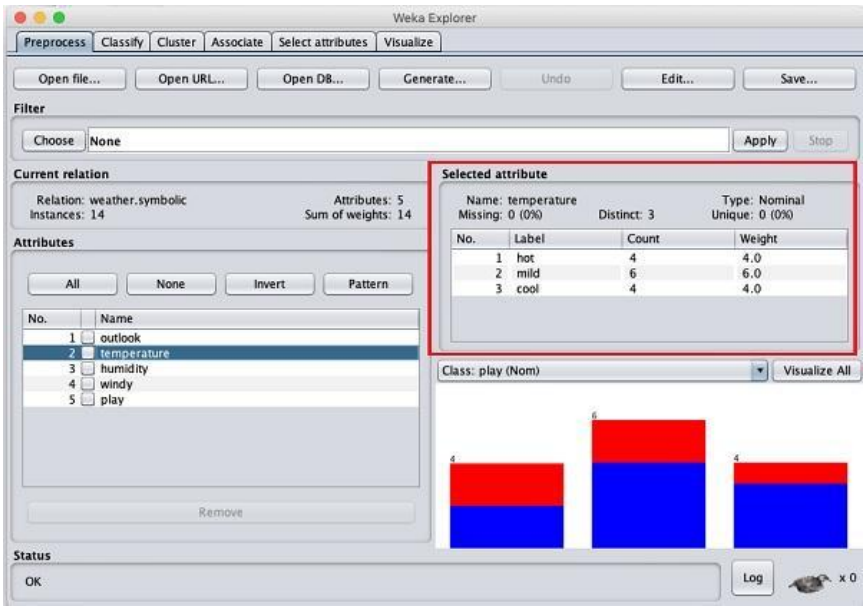
The table contains 5 attributes - the fields, which are discussed in the upcoming sections.

On the left side, notice the Attributes sub window that displays the various fields in the database.



The weather database contains five fields - outlook, temperature, humidity, windy and play. When you select an attribute from this list by clicking on it, further details on the attribute itself are displayed on the right hand side.

Let us select the temperature attribute first. When you click on it, you would see the following screen



In the Selected Attribute sub window, you can observe the following −

- The name and the type of the attribute are displayed.
- The type for the temperature attribute is Nominal.
- The number of Missing values is zero.

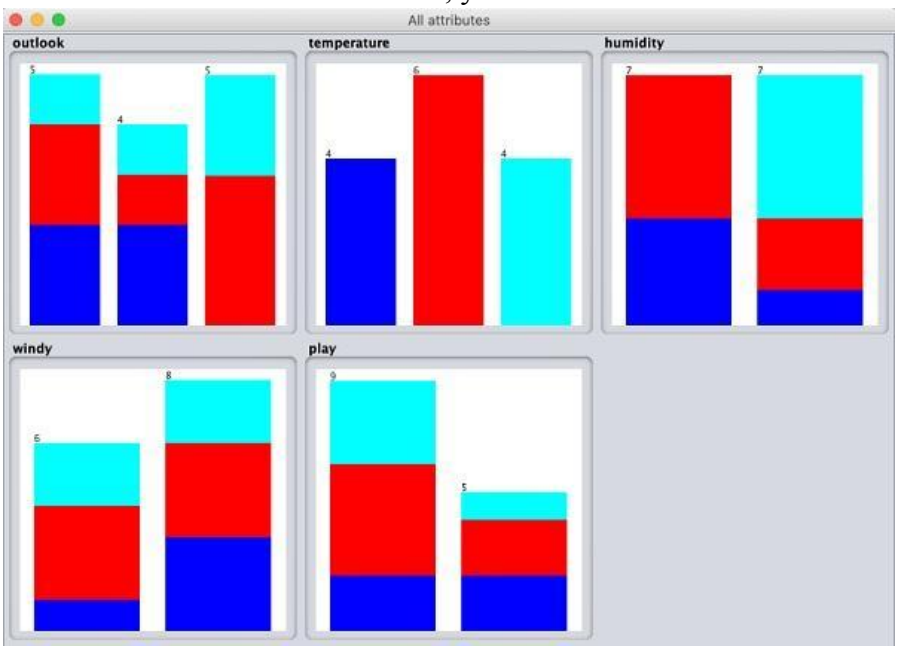
There are three distinct values with no unique value.

The table underneath this information shows the nominal values for this field as hot, mild and cold.

It also shows the count and weight in terms of a percentage for each nominal value.

At the bottom of the window, you see the visual representation of the class values.

If you click on the Visualize All button, you will be able to see all features in one single window as shown here



Removing Attributes

Many a time, the data that you want to use for model building comes with many irrelevant fields. For example, the customer database may contain his mobile number which is relevant in analysing his credit rating.

To remove Attribute/s select them and click on the Remove button at the bottom.

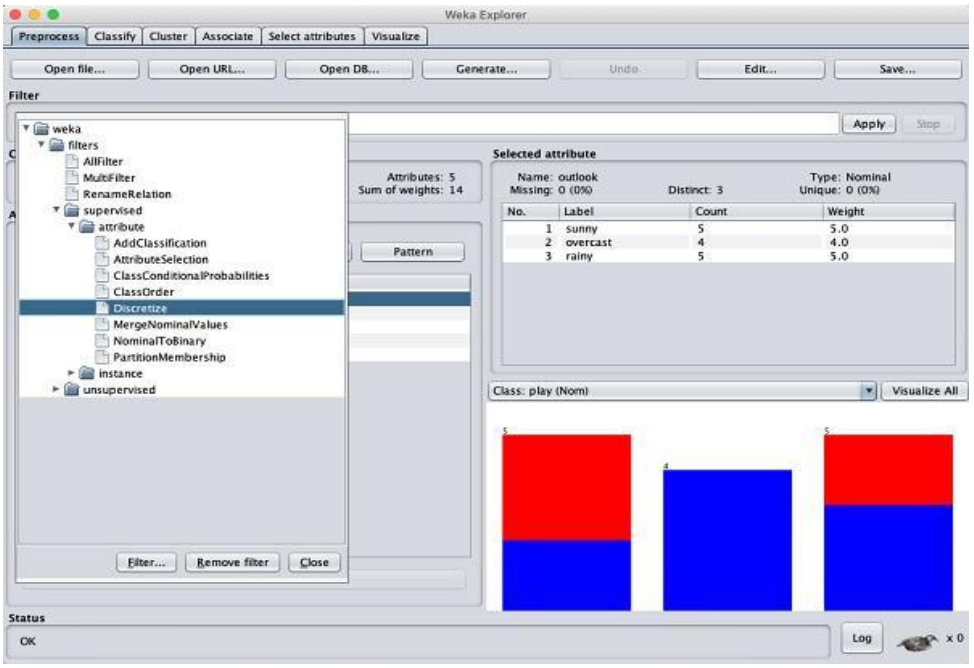
The selected attributes would be removed from the database. After you fully preprocess the data, you can save it for model building.

Applying Filters

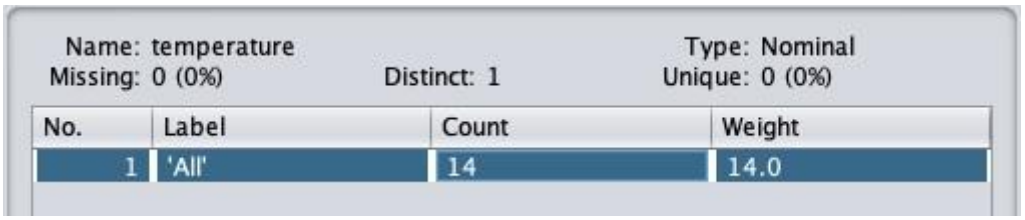
Some of the machine learning techniques such as association rule mining requires categorical data. To illustrate the use of filters, we will use weather-numeric.arff database that contains two numeric attributes - temperature and humidity.

We will convert these to nominal by applying a filter on our raw data. Click on the Choose button in the Filter subwindow and select the following filter −

weka→filters→supervised→attribute→Discretize



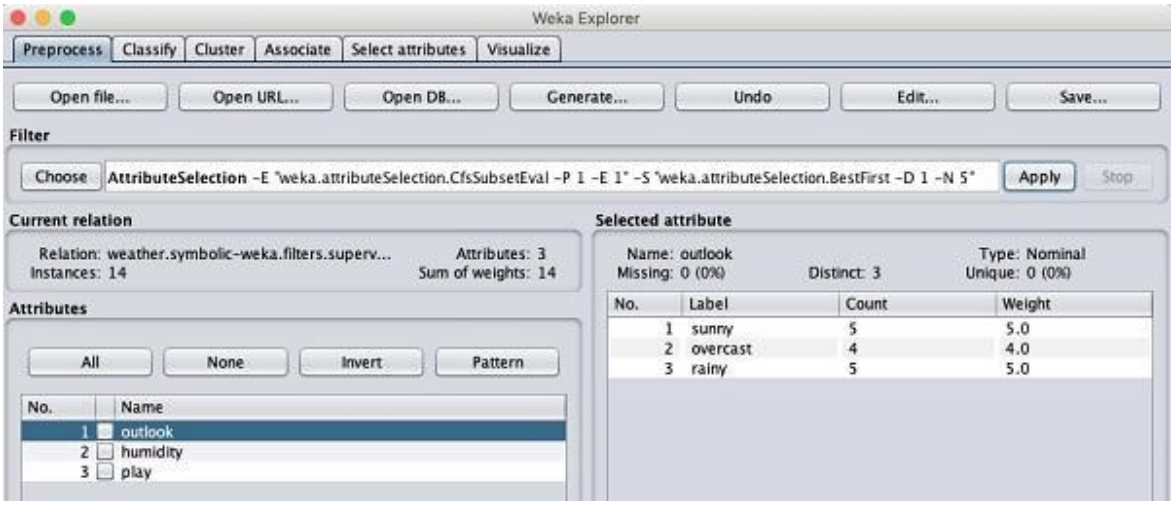
Click on the Apply button and examine the temperature and/or humidity attribute. You will notice that these have changed from numeric to nominal types.



Suppose you want to select the best attributes for deciding the play. Select and apply the following filter −

weka→filters→supervised→attribute→AttributeSelection

You will notice that it removes the temperature and humidity attributes from the database.



After you are satisfied with the preprocessing of your data, save the data by clicking the Save ... button. You will use this saved file for model building.

TASK 2

(a): Introduction to R Programming.

(b): Read dataset and practice basic programming concepts.

(a): Introduction to R Programming.

R is a programming language and an analytics tool that was developed in 1993 by Robert Gentleman and Ross Ihaka at the University of Auckland, Auckland, New Zealand. It is extensively used by Software Programmers, Statisticians, Data Scientists, and Data Miners. It is one of the most popular Data analytics tools used in Data Analytics and Business Analytics. It has numerous applications in domains like healthcare, academics, consulting, finance, media, and many more. Its vast applicability in Statistics, Data Visualization, and Machine Learning have given rise to the demand for certified trained professionals in R.

Some important features of R are as follows:

- It is a free and open-source programming language issued under GNU (General Public License).
- It has cross-platform interoperability which means that it has distributions running on Windows, Linux, and Mac. R code can easily be ported from one platform to another.
- It uses an interpreter instead of a compiler, which makes the development of code easier.
- It effectively associates different databases, and it does well in bringing in information from Microsoft Excel, as well as, Microsoft Access, MySQL, SQLite, Oracle, etc.
- It is a flexible language that bridges the gap between Software Development and Data Analysis.
- It provides a wide variety of packages with a diversity of codes, functions, and features tailored for data analysis, statistical modeling, visualization, Machine Learning, and importing and manipulating data.
- It integrates various powerful tools to communicate reports in different forms like CSV, XML, HTML, and pdf, and also through interactive websites, with the help of R packages.

Steps to perform Data Analysis in R

Import: The first step is to import data into the R environment. It means that you take the data stored in files, databases, HTML tables, etc., and load it into an R data frame to perform data analysis on it.

Transform: In this step, first, we make our data tidy by making each column a variable, and each row an observation. Once we have tidy data, we narrow down on it to find observations of our interest, create new variables that are functions of existing variables, and find summary statistics of the observations.

To Install R:

Open an internet browser and go to www.r-project.org.

Click the "download R" link in the middle of the page under "Getting Started."

Select a CRAN location (a mirror site) and click the corresponding link.

Click on the "Download R for Windows" link at the top of the page.

Click on the "install R for the first time" link at the top of the page.

Click "Download R for Windows" and save the executable file somewhere on your computer. Run the .exe file and follow the installation instructions.

Now that R is installed, you need to download and install RStudio.

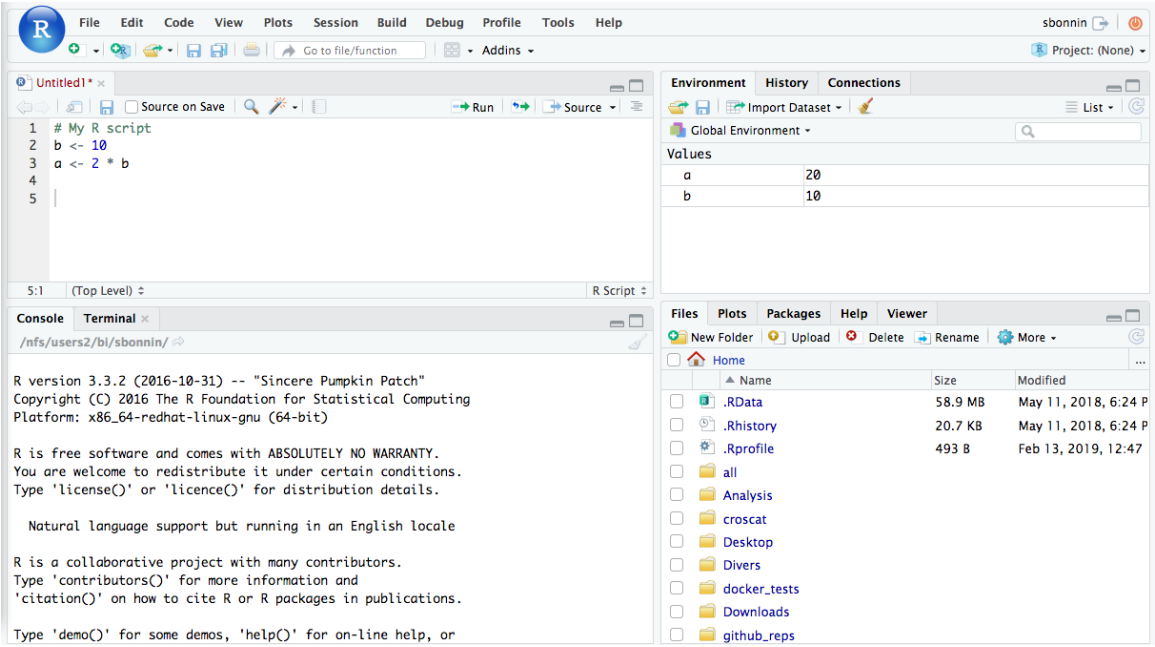
To Install RStudio

Go to www.rstudio.com and click on the "Download RStudio" button.

Click on "Download RStudio Desktop."

Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.

R Studio Interface:



(b): Read dataset and practice basic programming concepts.

Reading CSV Files

CSV (Comma Separated Values) is a text file in which the values in columns are separated by a comma.

For importing data in the R programming environment, we have to set our working directory with the setwd() function.

For example:

setwd("C:/Users/HP/Desktop/BLOG/files")

To read a csv file, we use the in-built function **read.csv()** that outputs the data from the file as a data frame.

For example:

read.data <- read.csv("file1.csv")

print(read.data)

Output:

Sl. No.	empid	empname	empdept	empsalary	empstart_date
1	1	Sam	IT	25000	03-09-2005
2	2	Rob	HR	30000	03-05-2005
3	3	Max	Marketing	29000	05-06-2007
4	4	John	R&D	35000	01-03-1999
5	5	Gary	Finance	32000	05-09-2000
6	6	Alex	Tech	20000	09-05-2005
7	7	Ivar	Sales	36000	04-04-1999
8	8	Robert	Finance	34000	06-08-2008

Analyzing a CSV File

#To print number of columns

print(ncol(read.data))

Output:

[1] 5

#To print number of rows

print(nrow(read.data))

Output:

[1] 8

#To print the range of salary packages

range.sal <- range(read.data$empsalary)

print(range.sal)

Output:

[1] 20000 36000

#To print the details of a person with the highest salary, we use the subset() function to extract variables and observations

max.sal <- subset(read.data, empsalary == max(empsalary))

print(max.sal)

Output:

Output:

Sl. No.	empid	empname	empdept	empsalary	empstart_date
7	7	Ivar	Sales	36000	04-04-1999

#To print the details of all people working in Finance department

fin.per <- subset(read.data, empdept == "Finance")

print(fin.per)

fin.per <- subset(read.data, empdept == "Finance")

print(fin.per)

Output:

Sl. No.	empid	empname	empdept	empsalary	empstart_date
5	5	Gary	Finance	36000	05-09-2000
8	8	Robert	Finance	34000	06-08-2008

TASK 3

(a): Perform Data exploration and pre-processing in R (use any UCI dataset)

(b): Perform Feature Engineering and Feature Selection Methods in R

(a): Perform Data exploration and pre-processing in R (use any UCI dataset)

It is very important to explore data before starting to build a predictive model. It gives an idea about the structure of the dataset like number of continuous or categorical variables and number of observations (rows).

Dataset

We have five variables - Q1, Q2, Q3, Q4 and Age. The variables Q1-Q4 represents survey responses of a questionnaire. The response lies between 1 and 6. The variable Age represents age groups of the respondents. It lies between 1 to 3. 1 represents Generation Z, 2 represents Generation X and Y, 3 represents Baby Boomers.

Q1	Q2	Q3	Q4	Age
1	2	6	1	1
6	3	1	1	3
2	4	3	4	3
6	5	6	4	2
3	6	6	5	2
2	4	1	1	3
6	5	1	1	1
4	5	2	6	3
1	6	5	5	3
1	5	5	6	2
1	4	3	5	1
2	5	6	6	3
6	5	3	5	2
2	4	1	6	3

Sample Data

Import data into R

The **read.csv()** function is used to import CSV file into R. The header = TRUE tells R that header is included in the data that we are going to import.

```
mydata <- read.csv("C:/Users/Deepanshu/Documents/Book1.csv", header=TRUE)
```

You can also create **sample data** which would be used further to demonstrate data exploration techniques. The program below creates random observations with replacement.

```
mydata = data.frame(Q1 = sample(1:6, 100, replace = TRUE),Q2 = sample(1:6, 100, replace = TRUE),Q3 = sample(1:6, 100, replace = TRUE), Q4 = sample(1:6, 100, replace = TRUE), Age = sample(1:3, 100, replace = TRUE))
```

1. Calculate basic descriptive statistics

```
summary(mydata)
```

```
> mydata <- read.csv("C:/Users/Deepanshu/Documents/Book1.csv", header=TRUE)
> summary(mydata)
       Q1              Q2              Q3              Q4              Age
 Min.   :1.000   Min.   :2.0   Min.   :1.00   Min.   :1.00   Min.   :1.000
 1st Qu.:1.250   1st Qu.:4.0   1st Qu.:1.25   1st Qu.:1.75   1st Qu.:2.000
 Median :2.000   Median :5.0   Median :3.00   Median :5.00   Median :2.500
 Mean   :3.071   Mean   :4.5   Mean   :3.50   Mean   :4.00   Mean   :2.286
 3rd Qu.:5.500   3rd Qu.:5.0   3rd Qu.:5.75   3rd Qu.:5.75   3rd Qu.:3.000
 Max.   :6.000   Max.   :6.0   Max.   :6.00   Max.   :6.00   Max.   :3.000
```

Data Exploration with R

To calculate summary of a particular column, say third column, you can use the following syntax :

```
summary( mydata[3])
```

To calculate summary of a particular column by its name, you can use the following syntax :

```
summary( mydata$Q1)
```

2. Lists name of variables in a dataset

```
names(mydata)
> names(mydata)
[1] "Q1"  "Q2"  "Q3"  "Q4"  "Age"
```

3. Calculate number of rows in a dataset

```
nrow(mydata)
> nrow(mydata)
[1] 100
```

4. Calculate number of columns in a dataset

```
ncol(mydata)
> ncol(mydata)
[1] 5
```

5. List structure of a dataset

```
str(mydata)
> str(mydata)
'data.frame': 100 obs. of  5 variables:
 $ Q1 : int  1 5 3 1 6 2 2 1 4 1 ...
 $ Q2 : int  3 3 3 1 1 4 2 2 6 1 ...
 $ Q3 : int  4 2 1 4 3 6 1 4 4 4 ...
 $ Q4 : int  3 5 1 1 3 4 2 2 5 1 ...
 $ Age: int  3 1 1 1 1 1 3 1 2 3 ...
```

6. See first 6 rows of dataset

```
head(mydata)
  Q1 Q2 Q3 Q4 Age
1  1  3  4  3   3
2  5  3  2  5   1
3  3  3  1  1   1
4  1  1  4  1   1
5  6  1  3  3   1
6  2  4  6  4   1
```

7. First n rows of dataset

In the code below, we are selecting first 5 rows of dataset.

```
head(mydata, n=5)
```

8. All rows but the last row

```
head(mydata, n= -1)
```

9. Last 6 rows of dataset

```
tail(mydata)
```

10. Last n rows of dataset

In the code below, we are selecting last 5 rows of dataset.

```
tail(mydata, n=5)
```

11. All rows but the first row

```
tail(mydata, n= -1)
```

12. Select random rows from a dataset

```
library(dplyr)
sample_n(mydata, 5)
```

If dplyr package is not already installed, make sure you install it before running the above script.

13. Selecting N% random rows

```
library(dplyr)
sample_frac(mydata, 0.1)
```

In this case, it selects 10% random rows from mydata data frame.

14. Number of missing values

The function below returns number of missing values in each variable of a dataset.

colSums(is.na(mydata))
It can also be written like -

sapply(mydata, function(y) sum(is.na(y)))

15. Number of missing values in a single variable

sum(is.na(mydata$Q1))

Data Pre-processing:

Importing The Dataset

dataset = read.csv('dataset.csv')

	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

The highlighted cells with value ‘NA’ denotes missing values in the dataset.

Dealing With Missing Values

dataset$age = ifelse(is.na(dataset$age),ave(dataset$age, FUN = function(x) mean(x, na.rm = 'TRUE')),dataset$age)

dataset$salary = ifelse(is.na(dataset$salary), ave(dataset$salary, FUN = function(x) mean(x, na.rm = 'TRUE')), dataset$salary)

Output :

	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	51000
5	Germany	Yes	40.00000	60000
6	India	Yes	35.00000	58000
7	Russia	No	39.58333	52000
8	India	Yes	48.00000	51000
9	Germany	No	50.00000	83000
10	India	Yes	37.00000	51000
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

dataset$age = as.numeric(format(round(dataset$age, 0)))

Since we are not interested in having decimal places for age we will round it up using the above code. The argument 0 in the round function means no decimal places.After executing the above code block the dataset would look like what's shown below :

	nation	purchased_item	age	salary
1	India	No	25	35000
2	Russia	Yes	40	40000
3	Germany	No	50	54000
4	Russia	No	35	51000
5	Germany	Yes	40	60000
6	India	Yes	35	58000
7	Russia	No	40	52000
8	India	Yes	48	51000
9	Germany	No	50	83000
10	India	Yes	37	51000
11	Germany	No	21	24000
12	India	Yes	40	60000
13	Russia	No	63	70000
14	Germany	yes	26	36000
15	India	No	45	40000

Dealing With Categorical Data

In our dataset, we have two categorical features, nation, and purchased_item. In R we can use the factor method to convert texts into numerical codes.

```
dataset$nation = factor(dataset$nation, levels = c('India','Germany','Russia'), labels = c(1,2,3))

dataset$purchased_item = factor(dataset$purchased_item, levels = c('No','Yes'),  labels = c(0,1))
```

Output:

	nation	purchased_item	age	salary
1	1	0	25	35000
2	3	1	40	40000
3	2	0	50	54000
4	3	0	35	51000
5	2	1	40	60000
6	1	1	35	58000
7	3	0	40	52000
8	1	1	48	51000
9	2	0	50	83000
10	1	1	37	51000
11	2	0	21	24000
12	1	1	40	60000
13	3	0	63	70000
14	2	1	26	36000
15	1	0	45	40000

Splitting The Dataset Into Training And Testing Sets

We will use the caTools library in R to split our dataset to training_set and test_set

```
install.packages('caTools') #install once
library(caTools) # importing caTools library
set.seed(123)
split = sample.split(dataset$purchased_item, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

Scaling The Features

```
training_set[,3:4] = scale(training_set[,3:4])
test_set[,3:4] = scale(test_set[,3:4])
```

The scale method in R can be used to scale the features in the dataset. Here we are only scaling the non-factors which are the age and the salary.

Output:

Training_set:

	nation	purchased_item	age	salary
1	1	0	-1.1813252	-1.0687684
2	3	1	0.1073932	-0.6777556
3	2	0	0.9665388	0.4170803
4	3	0	-0.3221796	0.1824727
5	2	1	0.1073932	0.8862957
7	3	0	0.1073932	0.2606752
8	1	1	0.7947097	0.1824727
10	1	1	-0.1503505	0.1824727
11	2	0	-1.5249835	-1.9289966
12	1	1	0.1073932	0.8862957
13	3	0	2.0834281	1.6683214
14	2	1	-1.0954107	-0.9905658

Test_set:

	nation	purchased_item	age	salary
6	1	1	-1.0910895	-0.1080509
9	2	0	0.8728716	1.0496377
15	1	0	0.2182179	-0.9415868

(b): Perform Feature Engineering and Feature Selection Methods in R

Feature Engineering in R:

Let's start by importing the dataset Bank Churn Model from Kaggle and visualize the first six rows [1].

```
df <- read.csv('Bank_churn_modelling.csv',header = T)
head(df)
```

```
  RowNumber CustomerId  Surname CreditScore Geography Gender Age Tenure   Balance NumOfProducts HasCrCard IsActiveMember EstimatedSalary Exited
1         1   15634602 Hargrave         619    France Female  42      2      0.00             1         1              1       101348.88      1
2         2   15647311     Hill         608     Spain Female  41      1  83807.86             1         0              1       112542.58      0
3         3   15619304     Onio         502    France Female  42      8 159660.80             3         1              0       113931.57      1
4         4   15701354     Boni         699    France Female  39      1      0.00             2         0              0        93826.63      0
5         5   15737888 Mitchell         850     Spain Female  43      2 125510.82             1         1              1        79084.10      0
6         6   15574012      Chu         645     Spain   Male  44      8 113755.78             2         1              0       149756.71      1
> |
```

```
summary(df)
```

```
   RowNumber      CustomerId       Surname           CreditScore     Geography          Gender              Age             Tenure          Balance       
 Min.   :    1   Min.   :15565701   Length:10000       Min.   :350.0   Length:10000       Length:10000       Min.   :18.00   Min.   : 0.000   Min.   :     0  
 1st Qu.: 2501   1st Qu.:15628528   Class :character   1st Qu.:584.0   Class :character   Class :character   1st Qu.:32.00   1st Qu.: 3.000   1st Qu.:     0  
 Median : 5000   Median :15690738   Mode  :character   Median :652.0   Mode  :character   Mode  :character   Median :37.00   Median : 5.000   Median : 97199  
 Mean   : 5000   Mean   :15690941                      Mean   :650.5                                         Mean   :38.92   Mean   : 5.013   Mean   : 76486  
 3rd Qu.: 7500   3rd Qu.:15753234                      3rd Qu.:718.0                                         3rd Qu.:44.00   3rd Qu.: 7.000   3rd Qu.:127644  
 Max.   :10000   Max.   :15815690                      Max.   :850.0                                         Max.   :92.00   Max.   :10.000   Max.   :250898  
 NumOfProducts     HasCrCard      IsActiveMember   EstimatedSalary        Exited      
 Min.   :1.00   Min.   :0.0000   Min.   :0.0000   Min.   :    11.58   Min.   :0.0000  
 1st Qu.:1.00   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 51002.11   1st Qu.:0.0000  
 Median :1.00   Median :1.0000   Median :1.0000   Median :100193.91   Median :0.0000  
 Mean   :1.53   Mean   :0.7055   Mean   :0.5151   Mean   :100090.24   Mean   :0.2037  
 3rd Qu.:2.00   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:149388.25   3rd Qu.:0.0000  
 Max.   :4.00   Max.   :1.0000   Max.   :1.0000   Max.   :199992.48   Max.   :1.0000  
```

summary is a function used to print the descriptive statistics of all the features. From the output, it's worth noticing that:

```
str(df)
```

```
'data.frame':   10000 obs. of  14 variables:
 $ RowNumber      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ CustomerId     : int  15634602 15647311 15619304 15701354 15737888 15574012 15592531 15656148 15792365 15592389 ...
 $ Surname        : chr  "Hargrave" "Hill" "Onio" "Boni" ...
 $ CreditScore    : int  619 608 502 699 850 645 822 376 501 684 ...
 $ Geography      : chr  "France" "Spain" "France" "France" ...
 $ Gender         : chr  "Female" "Female" "Female" "Female" ...
 $ Age            : int  42 41 42 39 43 44 50 29 44 27 ...
 $ Tenure         : int  2 1 8 1 2 8 7 4 4 2 ...
 $ Balance        : num  0 83808 159661 0 125511 ...
 $ NumOfProducts  : int  1 1 3 2 1 2 2 4 2 1 ...
 $ HasCrCard      : int  1 0 1 0 1 1 1 1 0 1 ...
 $ IsActiveMember : int  1 1 0 0 1 0 1 0 1 1 ...
 $ EstimatedSalary: num  101349 112543 113932 93827 79084 ...
 $ Exited         : int  1 0 1 0 0 1 0 1 0 0 ...
```

str provides a more compact output that summarizes the structure of the dataset. It tells us that there are 1000 rows and 14 columns. Most of the variables contain numeric/integer values, while the remaining features are character variables.

1. Convert to factors

As we have seen in the previous paragraph, we have to change to convert some variables into factors. You are probably wondering what factors are. In the R language, factors are types of vectors specialized in grouping elements into categories. In other words, factors are categorical variables, that can have two or more levels. These levels can contain both strings and integers.

```
df$Geography <- as.factor(Geography)
df$Gender <- as.factor(Gender)
df$HasCrCard <- as.factor(HasCrCard)
```

```
df$IsActiveMember <- as.factor(IsActiveMember)
df$Exited <- as.factor(Exited)
str(df)
```

```
'data.frame':   10000 obs. of  14 variables:
 $ RowNumber      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ CustomerId     : int  15634602 15647311 15619304 15701354 15737888 15574012 15592531 15656148 15792365 15592389 ...
 $ Surname        : chr  "Hargrave" "Hill" "Onio" "Boni" ...
 $ CreditScore    : int  619 608 502 699 850 645 822 376 501 684 ...
 $ Geography      : Factor w/ 3 levels "France","Germany",..: 1 3 1 1 3 3 1 2 1 1 ...
 $ Gender         : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 2 2 1 2 2 ...
 $ Age            : int  42 41 42 39 43 44 50 29 44 27 ...
 $ Tenure         : int  2 1 8 1 2 8 7 4 4 2 ...
 $ Balance        : num  0 83808 159661 0 125511 ...
 $ NumOfProducts  : int  1 1 3 2 1 2 2 4 2 1 ...
 $ HasCrCard      : Factor w/ 2 levels "0","1": 2 1 2 1 2 2 2 2 1 2 ...
 $ IsActiveMember : Factor w/ 2 levels "0","1": 2 2 1 1 2 1 2 1 2 2 ...
 $ EstimatedSalary: num  101349 112543 113932 93827 79084 ...
 $ Exited         : Factor w/ 2 levels "0","1": 2 1 2 1 1 2 1 2 1 1 ...
```

We can check easily if we converted well the variables using the `str` function again. To visualize the levels of the factors, we can use the levels function:

```
levels(Geography)
# "France"  "Germany" "Spain"
```

The dataset contains only customers from France, Germany and Spain.

2. Add and Delete a column

The easier way to add a new column is to assign a single value to the entire new variable. For example, let's create a new column filled with NA, which is a logical constant that indicates missing values in R.

```
df$newcol <- NA
df[1:5,8:15]
```

	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	newcol
1	2	0.00	1	1	1	101348.88	1	NA
2	1	83807.86	1	0	1	112542.58	0	NA
3	8	159660.80	3	1	0	113931.57	1	NA
4	1	0.00	2	0	0	93826.63	0	NA
5	2	125510.82	1	1	1	79084.10	0	NA

To have an easier visualization, a subset of the dataset is selected, the first five rows and the last eight columns. An alternative for adding a column is the following syntax:

```
df['newcol'] <- NA
```

So, we can specify the variable into the square brackets, instead of using the $ operator. To delete a column, we can set it to NULL, which represents the null object in R:

```
df$newcol <- NULL
# alternative df['newcol'] <- NULL
head(df)
```

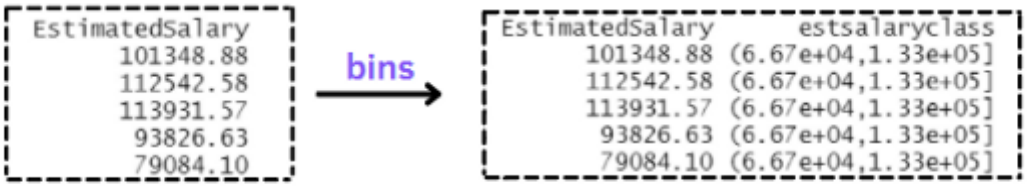
	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
1	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1
2	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0
3	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1
4	4	15701354	Boni	699	France	Female	39	1	0.00	2	0
5	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1
6	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1

	IsActiveMember	EstimatedSalary	Exited
1	1	101348.88	1
2	1	112542.58	0
3	0	113931.57	1
4	0	93826.63	0
5	1	79084.10	0
6	0	149756.71	1

3. Recode continuous variable to a categorical variable

Let's suppose that I want to create a new variable, based on the values of EstimatedSalary. The goal is to visualize the range of the estimated salary into intervals. This is possible using the cut function:

```
df$estsalaryclass<-cut(df$EstimatedSalary,breaks =
seq(0,max(df$EstimatedSalary)+1,len=4))
levels(df$estsalaryclass)
# [1] "(11.6,6.67e+04]"     "(6.67e+04,1.33e+05]" "(1.33e+05,2e+05]"
```



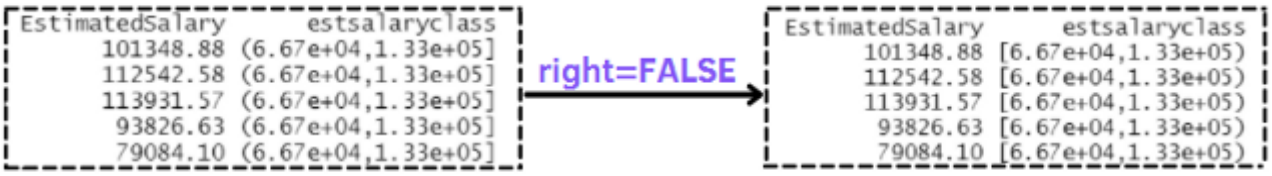
To create the intervals, the seq function is applied to create three different levels. It takes in as input the start value, the end value and the length of the sequence.

We can also add labels to bins specifying the parameter labels:

```
df$estsalaryclass<-cut(df$EstimatedSalary,breaks = seq(0,max(df$EstimatedSalary)+1,len=4),
                       labels = c('poor','middle','rich'))
levels(df$estsalaryclass)
# [1] "poor"   "middle" "rich"
df[1:5,c('EstimatedSalary','estsalaryclass')]
```

From the illustration, you can deduce that the names of the factor levels are created after specifying the bounds.We can also notice that the *intervals are by default open on the left and closed on the right*. This means that only the higher value is included on the bound, while the lowest value is excluded.If we want the intervals to be closed on the left and open on the right, we need to set the parameter right equal to FALSE:

```
df$estsalaryclass<-cut(df$EstimatedSalary,breaks =
seq(0,max(df$EstimatedSalary)+1,len=4),right = FALSE)
levels(df$estsalaryclass)
[1] "[11.6,6.67e+04)"     "[6.67e+04,1.33e+05)" "[1.33e+05,2e+05)"
```



4. Find missing values

R language provides is.na function to check if there are missing values in the dataset. To count the missing values of the dataset, we can sum them:

```
sum(is.na(df))
# [1] 0
```

Since the data frame doesn't contain missing values, we can manually add it as an exercise:

```
df[5,'Age'] = NA
sum(is.na(df))
# [1] 1
```

You are probably thinking about what `is.na` function is really doing. Let's print the output of this function:

```
head(is.na(df))
```

```
     RowNumber CustomerId Surname CreditScore Geography Gender   Age Tenure Balance NumOfProducts HasCrCard IsActiveMember
[1,]     FALSE      FALSE   FALSE       FALSE     FALSE  FALSE FALSE  FALSE   FALSE         FALSE     FALSE          FALSE
[2,]     FALSE      FALSE   FALSE       FALSE     FALSE  FALSE FALSE  FALSE   FALSE         FALSE     FALSE          FALSE
[3,]     FALSE      FALSE   FALSE       FALSE     FALSE  FALSE FALSE  FALSE   FALSE         FALSE     FALSE          FALSE
[4,]     FALSE      FALSE   FALSE       FALSE     FALSE  FALSE FALSE  FALSE   FALSE         FALSE     FALSE          FALSE
[5,]     FALSE      FALSE   FALSE       FALSE     FALSE  FALSE  TRUE  FALSE   FALSE         FALSE     FALSE          FALSE
[6,]     FALSE      FALSE   FALSE       FALSE     FALSE  FALSE FALSE  FALSE   FALSE         FALSE     FALSE          FALSE
     EstimatedSalary Exited estsalaryclass
[1,]           FALSE  FALSE          FALSE
[2,]           FALSE  FALSE          FALSE
[3,]           FALSE  FALSE          FALSE
[4,]           FALSE  FALSE          FALSE
[5,]           FALSE  FALSE          FALSE
[6,]           FALSE  FALSE          FALSE
```

It returns a data frame containing boolean values that represent the missing values, where TRUE indicates that we have a NA value. To return the number of missing values for each column, we can use the `for` and the `cat` function:

```
for (i in 1:length(df))
{cat(c(colnames(df[i]),':',sum(is.na(df[,i])),"\n"))}
```

```
RowNumber : 0
CustomerId : 0
Surname : 0
CreditScore : 0
Geography : 0
Gender : 0
Age : 1
Tenure : 0
Balance : 0
NumOfProducts : 0
HasCrCard : 0
IsActiveMember : 0
EstimatedSalary : 0
Exited : 0
estsalaryclass : 0
```

5. Handle Missing values

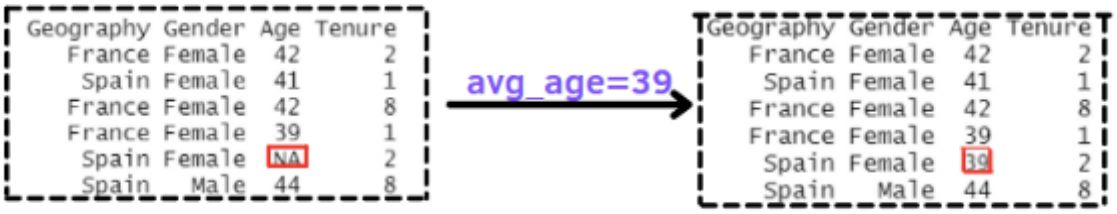
The easier way to handle missing values would be to drop the rows containing NA values:

```
df_notna<-na.omit(df)
sum(is.na(df_notna))
# [1] 0
```

`na.omit` deletes easily the rows with NA and returns a data frame without missing values.

But removing missing values isn't always the best solution. Another method could be to replace the NA values with the column's mean:

```
avg_age <- round(mean(df$Age,na.rm=TRUE))
df$Age<-ifelse(is.na(df$Age),avg_age,df$Age)
```



Feature Selection Methods in R

- Random Forest Method, Relative Importance, MARS, Step-wise Regression and Boruta

Import Data

For illustrating the various methods, we will use the 'Ozone' data from 'mlbench' package, except for Information value method which is applicable for binary categorical response variables.

inputData <- read.csv("http://rstatistics.net/wp-content/uploads/2015/09/ozone1.csv", stringsAsFactors=F)

Random Forest Method

Random forest can be very effective to find a set of predictors that best explains the variance in the response variable.

```
library(party)
cf1 <- cforest(ozone_reading ~ . , data= inputData, control=cforest_unbiased(mtry=2,ntree=50)) # fit the random forest
varimp(cf1) # get variable importance, based on mean decrease in accuracy
#=>              Month       Day_of_month        Day_of_week
#=>        0.689167598        0.115937291       -0.004641633
#=>     pressure_height         Wind_speed           Humidity
#=>        5.519633507        0.125868789        3.474611356
#=> Temperature_Sandburg  Temperature_ElMonte Inversion_base_height
#=>       12.878794481       14.175901506        4.276103121
#=>  Pressure_gradient Inversion_temperature          Visibility
```

```
#=>         3.234732558         11.738969777          2.283430842
varimp(cf1, conditional=TRUE)  # conditional=True, adjusts for correlations between predictors
#=>               Month        Day_of_month         Day_of_week
#=>          0.08899435          0.19311805          0.02526252
#=>     pressure_height          Wind_speed            Humidity
#=>          0.35458493         -0.19089686          0.14617239
#=>  Temperature_Sandburg   Temperature_ElMonte Inversion_base_height
#=>          0.74640367          1.19786882          0.69662788
#=>    Pressure_gradient Inversion_temperature          Visibility
#=>          0.58295887          0.65507322          0.05380003
varimpAUC(cf1)  # more robust towards class imbalance.
#=>               Month        Day_of_month         Day_of_week
#=>          1.12821259         -0.04079495          0.07800158
#=>     pressure_height          Wind_speed            Humidity
#=>          5.85160593          0.11250973          3.32289714
#=>  Temperature_Sandburg   Temperature_ElMonte Inversion_base_height
#=>         11.97425093         13.66085973          3.70572939
#=>    Pressure_gradient Inversion_temperature          Visibility
#=>          3.05169171         11.48762432          2.04145930
```

TASK 4

(a): Implement regularized Linear regression in R (use any UCI dataset).

(b): Implement regularized logistic regression in R

(a): Implement regularized Linear regression in R (use any UCI dataset).

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is −

```
y = ax + b
```

Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()** function in R.

Input Data

Below is the sample data representing the observations −

```
# Values of height
151, 174, 138, 186, 128, 136, 179, 163, 152, 131

# Values of weight.
63, 81, 56, 91, 47, 57, 76, 72, 62, 48
```

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in linear regression is −

```
lm(formula,data)
```

Following is the description of the parameters used −

- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
```

```
# Apply the lm() function.
relation <- lm(y~x)
print(relation)
```

When we execute the above code, it produces the following result −

```
Call:
lm(formula = y ~ x)
Coefficients:
(Intercept)           x
   -38.4551          0.6746
```

Get the Summary of the Relationship

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)

print(summary(relation))
```

When we execute the above code, it produces the following result −

```
Call:
lm(formula = y ~ x)
Residuals:
    Min      1Q     Median      3Q     Max
-6.3002    -1.6629  0.0412    1.8944  3.9775
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509    8.04901  -4.778  0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,    Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

predict() Function

Syntax

The basic syntax for predict() in linear regression is −

```
predict(object, newdata)
```

Following is the description of the parameters used −

- **object** is the formula which is already created using the lm() function.
- **newdata** is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
# The resposne vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)
# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <-  predict(relation,a)
print(result)
```

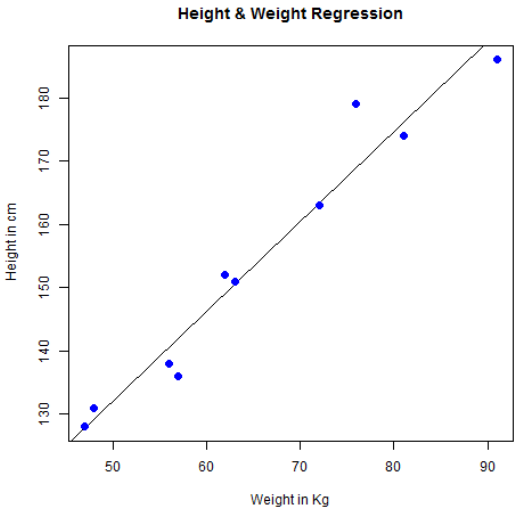
When we execute the above code, it produces the following result −

```
       1
76.22869
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)
# Give the chart file a name.
png(file = "linearregression.png")
# Plot the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression",
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")
# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −


Height & Weight Regression
Height in cm
130
140
150
160
170
180
50
60
70
80
90
Weight in Kg

(b): Implement regularized logistic regression in R

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is −

y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))

Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are the coefficients which are numeric constants.

The function used to create the regression model is the **glm()** function.

Syntax

The basic syntax for **glm()** function in logistic regression is −

```
glm(formula,data,family)
```

Following is the description of the parameters used −

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is binomial for logistic regression.

Example

The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

```
# Select some columns form mtcars.
input <- mtcars[,c("am","cyl","hp","wt")]
print(head(input))
```

When we execute the above code, it produces the following result −

```
                  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
```

Create Regression Model

We use the **glm()** function to create the regression model and get its summary for analysis.

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

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

When we execute the above code, it produces the following result −

```
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
```

Conclusion

In the summary as the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", we consider them to be insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.

TASK 5

Implement Apriori association rule mining algorithm in R and Weka

Apriori Algorithm Implementation in R

RStudio provides popular open source and enterprise-ready professional software for the R statistical computing environment. R is a language that is developed to support statistical calculations and graphical computing/ visualizations. It has an in-built library function called **arules** which implements the Apriori algorithm for **Market Basket Analysis** and computes the strong rules through Association Rule Mining, once we specify the minimum support and minimum confidence, according to our needs. Given below are the required code

and corresponding output for the Apriori algorithm. The **Groceries** dataset has been used for the same, which is available in the default database of R. It contains 9,835 transactions/ records, each having 'n' number of items that were bought together from the grocery store.

Example:

Step 1: Load required library

'**arules**' package provides the infrastructure for representing, manipulating, and analyzing transaction data and patterns.

```
library(arules)
```

'**arulesviz**' package is used for visualizing Association Rules and Frequent Itemsets. It extends the package 'arules' with various visualization techniques for association rules and itemsets. The package also includes several interactive visualizations for rule exploration.

```
library(arulesViz)
```

'**RColorBrewer**' is a ColorBrewer Palette which provides color schemes for maps and other graphics.

```
library(RColorBrewer)
```

Step 2: Import the dataset

'**Groceries**' dataset is predefined in the R package. It is a set of 9835 records/ transactions, each having 'n' number of items, which were bought together from the grocery store.

```
data("Groceries")
```

Step 3: Applying apriori() function

'**apriori()**' function is in-built in R to mine frequent itemsets and association rules using the Apriori algorithm. Here, 'Groceries' is the transaction data. 'parameter' is a named list that specifies the minimum support and confidence for finding the association rules. The default behavior is to mine the rules with minimum support of 0.1 and 0.8 as the minimum confidence. Here, we have specified the minimum support to be 0.01 and the minimum confidence to be 0.2.

```
rules <- apriori(Groceries, parameter = list(supp = 0.01, conf = 0.2))
```

Step 4: Applying inspect() function

inspect() function prints the internal representation of an R object or the result of an expression. Here, it displays the first 10 strong association rules.

```
inspect(rules[1:10])
```

Step 5: Applying itemFrequencyPlot() function

itemFrequencyPlot() creates a bar plot for item frequencies/ support. It creates an item frequency bar plot for inspecting the distribution of objects based on the transactions. The items are plotted ordered by descending support. Here, 'topN=20' means that 20 items with the highest item frequency/ lift will be plotted.

```
arules::itemFrequencyPlot(Groceries, topN = 20,
                          col = brewer.pal(8, 'Pastel2'),
                          main = 'Relative Item Frequency Plot',
                          type = "relative",
                          ylab = "Item Frequency (Relative)")
```

The complete R code is given below.

```
# Loading Libraries
```

```
library(arules)
library(arulesViz)
library(RColorBrewer)

# import dataset
data("Groceries")

# using apriori() function
rules <- apriori(Groceries,
parameter = list(supp = 0.01, conf = 0.2))

# using inspect() function
inspect(rules[1:10])

# using itemFrequencyPlot() function
arules::itemFrequencyPlot(Groceries, topN = 20,
col = brewer.pal(8, 'Pastel2'),
  main = 'Relative Item Frequency Plot',
    type = "relative",
     ylab = "Item Frequency (Relative)")
```

Output:

```
Console  Terminal
~/
> rules <- apriori(Groceries,parameter = list(supp = 0.01, conf = 0.2))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen
        0.2    0.1    1 none FALSE            TRUE       5    0.01      1     10
 target   ext
  rules FALSE

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

Absolute minimum support count: 98

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
sorting and recoding items ... [88 item(s)] done [0.00s].
creating transaction tree ... done [0.01s].
checking subsets of size 1 2 3 4 done [0.01s].
writing ... [232 rule(s)] done [0.00s].
creating S4 object  ... done [0.01s].
> inspect(rules[1:10])
     lhs                rhs                    support    confidence lift     count
[1]  {}              => {whole milk}       0.25551601 0.2555160  1.000000 2513
[2]  {hard cheese}   => {whole milk}       0.01006609 0.4107884  1.607682   99
[3]  {butter milk}   => {other vegetables} 0.01037112 0.3709091  1.916916  102
[4]  {butter milk}   => {whole milk}       0.01159126 0.4145455  1.622385  114
[5]  {ham}           => {whole milk}       0.01148958 0.4414062  1.727509  113
[6]  {sliced cheese} => {whole milk}       0.01077783 0.4398340  1.721356  106
[7]  {oil}           => {whole milk}       0.01128622 0.4021739  1.573968  111
[8]  {onions}        => {other vegetables} 0.01423488 0.4590164  2.372268  140
[9]  {onions}        => {whole milk}       0.01209964 0.3901639  1.526965  119
[10] {berries}       => {yogurt}           0.01057448 0.3180428  2.279848  104
> arules::itemFrequencyPlot(Groceries,topN=20,col=brewer.pal(8,'Pastel2'),main='Relativ
e Item Frequency Plot',type="relative",ylab="Item Frequency (Relative)")
```

Strong Rules:

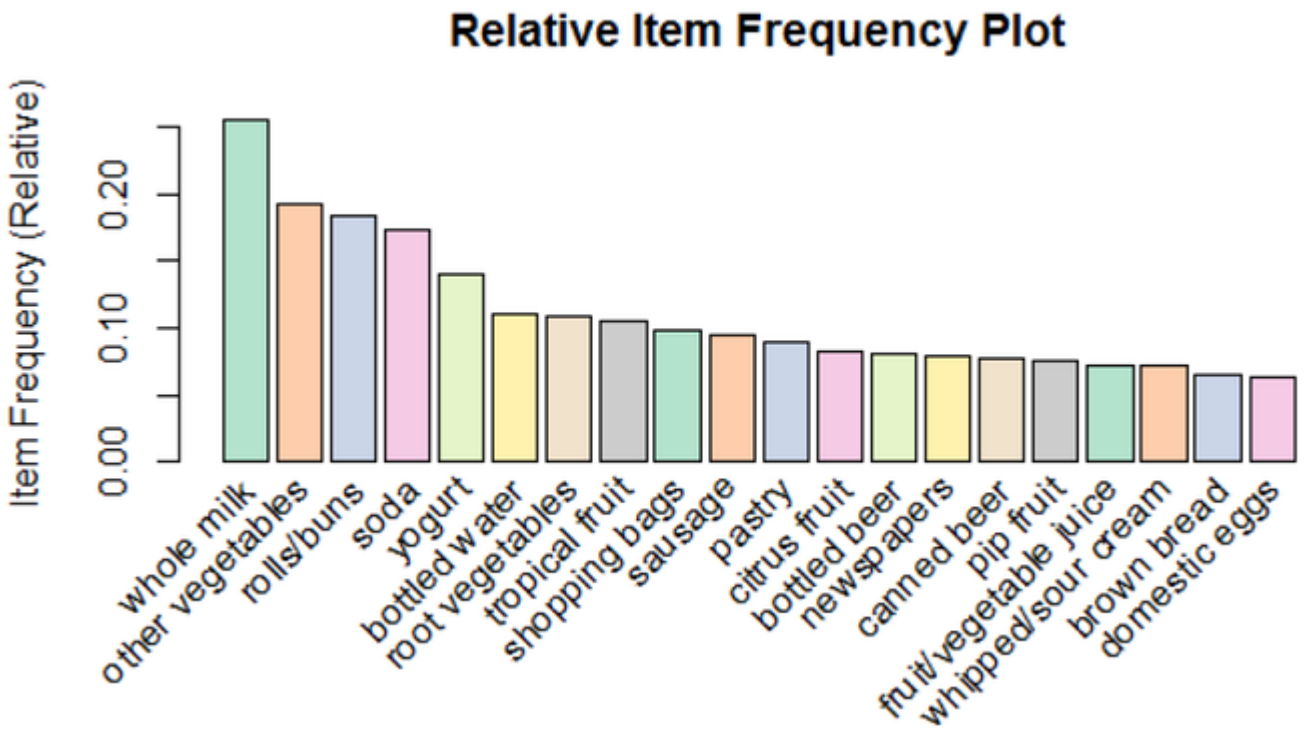
Strong Rules obtained after applying the Apriori Algorithm is as follows

```
     lhs                rhs                    support    confidence lift     count
[1]  {}              => {whole milk}       0.25551601 0.2555160  1.000000 2513
[2]  {hard cheese}   => {whole milk}       0.01006609 0.4107884  1.607682   99
[3]  {butter milk}   => {other vegetables} 0.01037112 0.3709091  1.916916  102
[4]  {butter milk}   => {whole milk}       0.01159126 0.4145455  1.622385  114
[5]  {ham}           => {whole milk}       0.01148958 0.4414062  1.727509  113
[6]  {sliced cheese} => {whole milk}       0.01077783 0.4398340  1.721356  106
[7]  {oil}           => {whole milk}       0.01128622 0.4021739  1.573968  111
[8]  {onions}        => {other vegetables} 0.01423488 0.4590164  2.372268  140
[9]  {onions}        => {whole milk}       0.01209964 0.3901639  1.526965  119
[10] {berries}       => {yogurt}           0.01057448 0.3180428  2.279848  104
```

After running the above code for the Apriori algorithm, we can see the following output, specifying the first 10 strongest Association rules, based on the support (minimum support of 0.01), confidence (minimum confidence of 0.2), and lift, along with mentioning the count of times the products occur together in the transactions.

Visualization:

Box Plot of the Top 20 Items having the Highest Item Frequency (Relative) using Lift as a Parameter

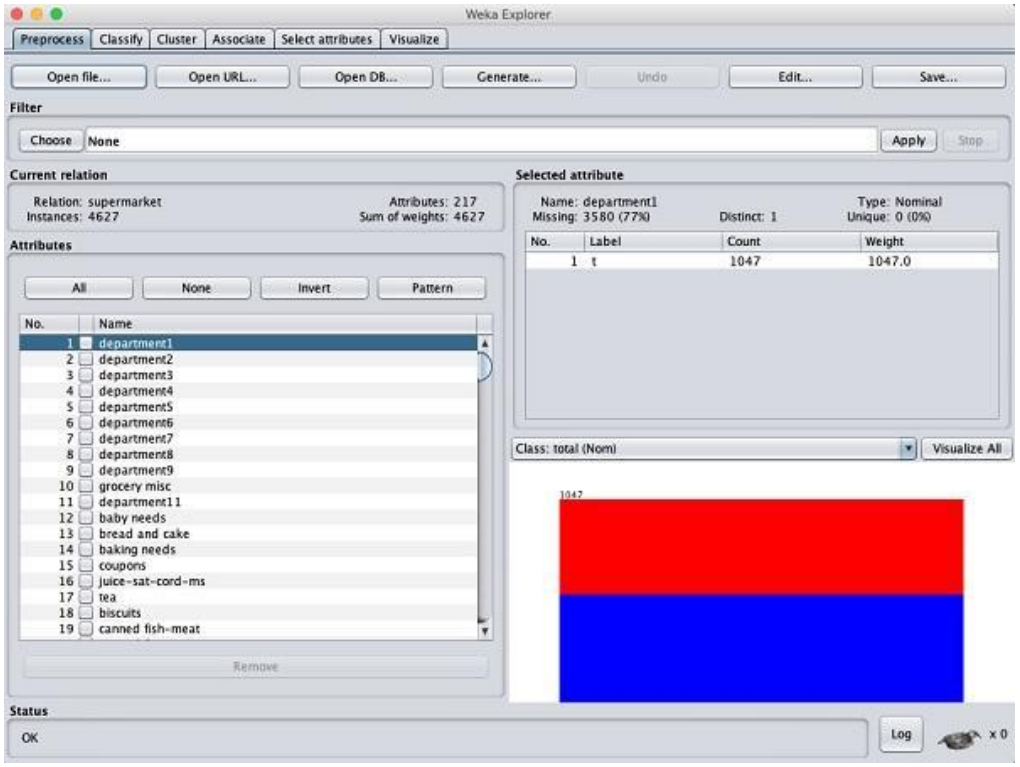


Implement Apriori association rule mining algorithm in Weka

WEKA provides the implementation of the Apriori algorithm. You can define the minimum support and an acceptable confidence level while computing these rules. You will apply the **Apriori** algorithm to the **supermarket** data provided in the WEKA installation.

Loading Data

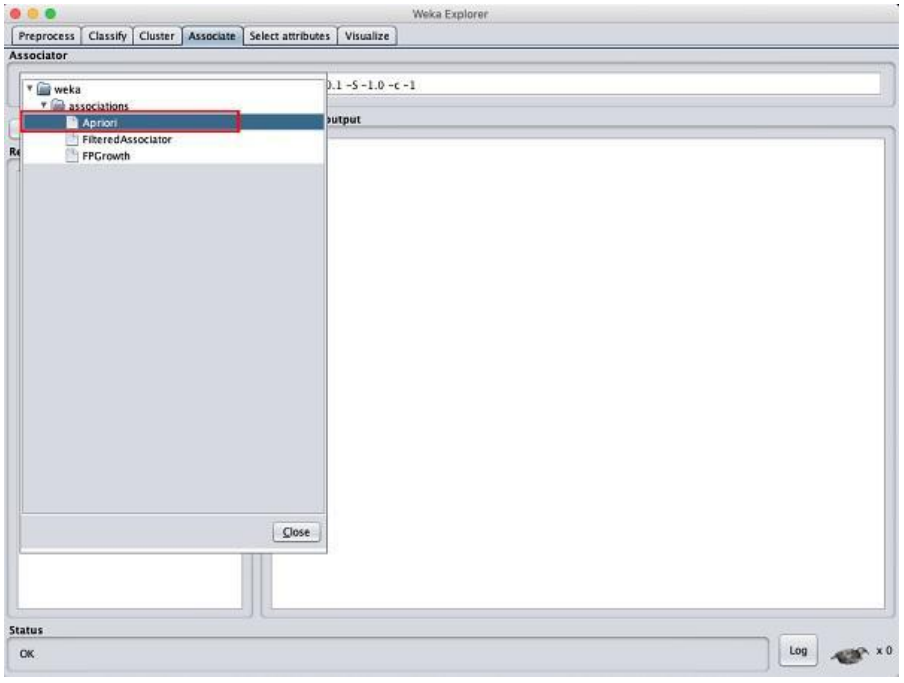
In the WEKA explorer, open the **Preprocess** tab, click on the **Open file** ... button and select **supermarket.arff** database from the installation folder. After the data is loaded you will see the following screen −



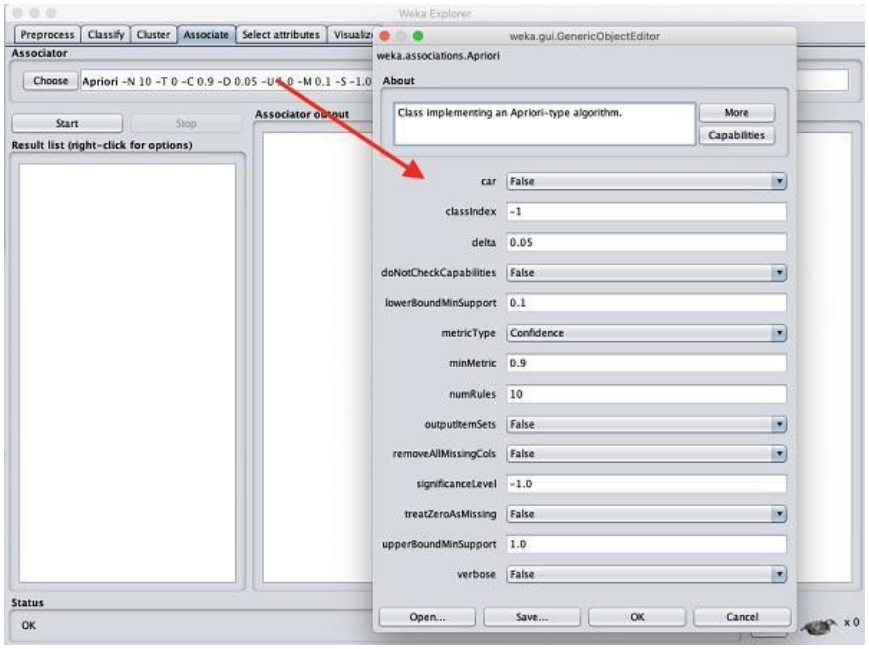
The database contains 4627 instances and 217 attributes. You can easily understand how difficult it would be to detect the association between such a large number of attributes. Fortunately, this task is automated with the help of Apriori algorithm.

Associator

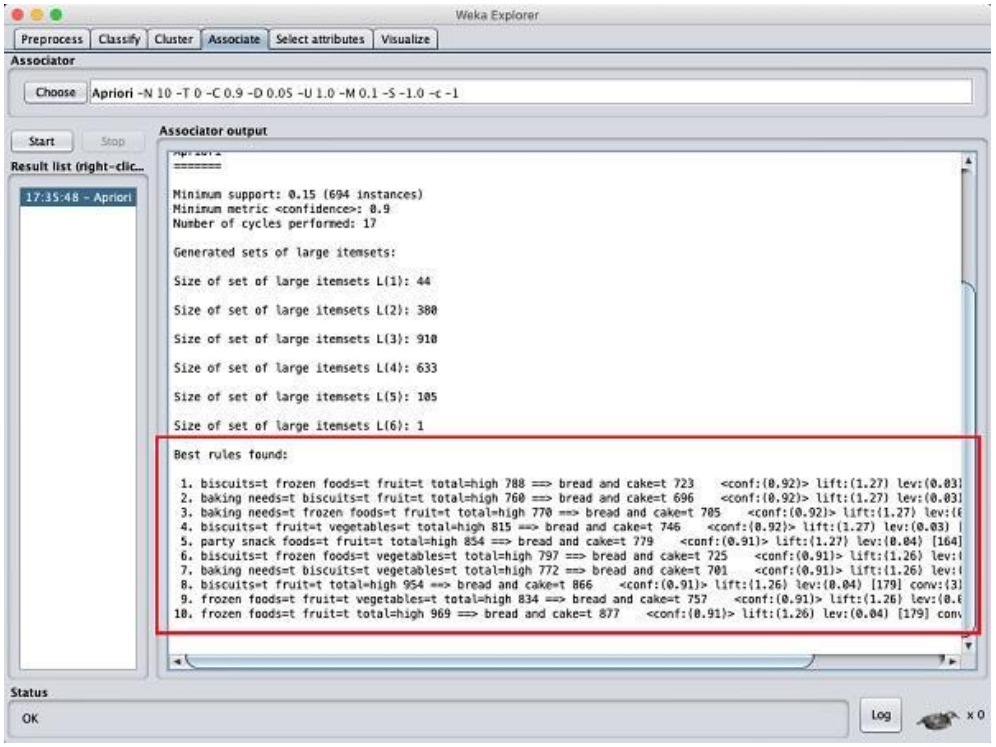
Click on the **Associate** TAB and click on the **Choose** button. Select the **Apriori** association as shown in the screenshot −



To set the parameters for the Apriori algorithm, click on its name, a window will pop up as shown below that allows you to set the parameters −



After you set the parameters, click the **Start** button. After a while you will see the results as shown in the screenshot below −



At the bottom, you will find the detected best rules of associations. This will help the supermarket in stocking their products in appropriate shelves.

TASK 6

Implement K- means clustering algorithm in R and Weka

K- means clustering algorithm in R:

K Means Clustering in R Programming is an Unsupervised Non-linear algorithm that cluster data based on similarity or similar groups. It seeks to partition the observations into a pre-specified number of clusters. Segmentation of data takes place to assign each training example to a segment called a cluster. In the

unsupervised algorithm, high reliance on raw data is given with large expenditure on manual review for review of relevance is given. It is used in a variety of fields like Banking, healthcare, retail, Media, etc.

The Dataset

Iris dataset consists of 50 samples from each of 3 species of Iris(Iris setosa, Iris virginica, Iris versicolor) and a multivariate dataset introduced by British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems. Four features were measured from each sample i.e length and width of the sepals and petals and based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

```
# Loading data
data(iris)
# Structure
str(iris)
```

Performing K-Means Clustering on Dataset

Using K-Means Clustering algorithm on the dataset which includes 11 persons and 6 variables or attributes

```
# Installing Packages
install.packages("ClusterR")
install.packages("cluster")
  # Loading package
library(ClusterR)
library(cluster)
  # Removing initial label of
# Species from original dataset
iris_1 <- iris[, -5]

# Fitting K-Means clustering Model
# to training dataset
set.seed(240) # Setting seed
kmeans.re <- kmeans(iris_1, centers = 3, nstart = 20)
kmeans.re
  # Cluster identification for
# each observation
kmeans.re$cluster
  # Confusion Matrix
cm <- table(iris$Species, kmeans.re$cluster)
```

```
  # Model Evaluation and visualization
plot(iris_1[c("Sepal.Length", "Sepal.Width")])
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
     col = kmeans.re$cluster)
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
     col = kmeans.re$cluster,
     main = "K-means with 3 clusters")
  ## Plotiing cluster centers
kmeans.re$centers
kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")]
  # cex is font size, pch is symbol
points(kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")],
       col = 1:3, pch = 8, cex = 3)
  ## Visualizing clusters
y_kmeans <- kmeans.re$cluster
clusplot(iris_1[, c("Sepal.Length", "Sepal.Width")],
         y_kmeans,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = paste("Cluster iris"),
         xlab = 'Sepal.Length',
         ylab = 'Sepal.Width')
```

Output:

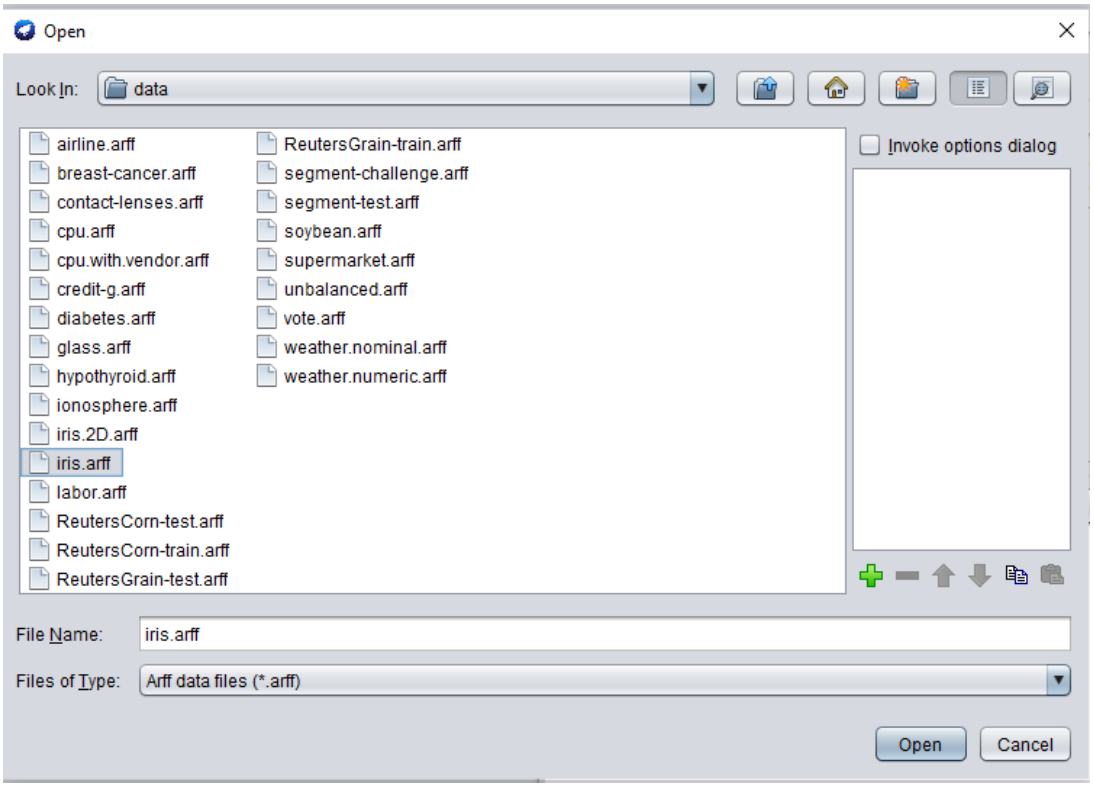
```
> kmeans.re
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

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
 [44] 1 1 1 1 1 1 1 1 2 2 3 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 2
 [87] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3 2 2 3 3 3 2 3 2 3 2 3 3
[130] 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

The 3 clusters are made which are of 50, 62, and 38 sizes respectively. Within the cluster, the sum of squares is 88.4%.

K- means clustering algorithm in Weka

Steps to be followed:

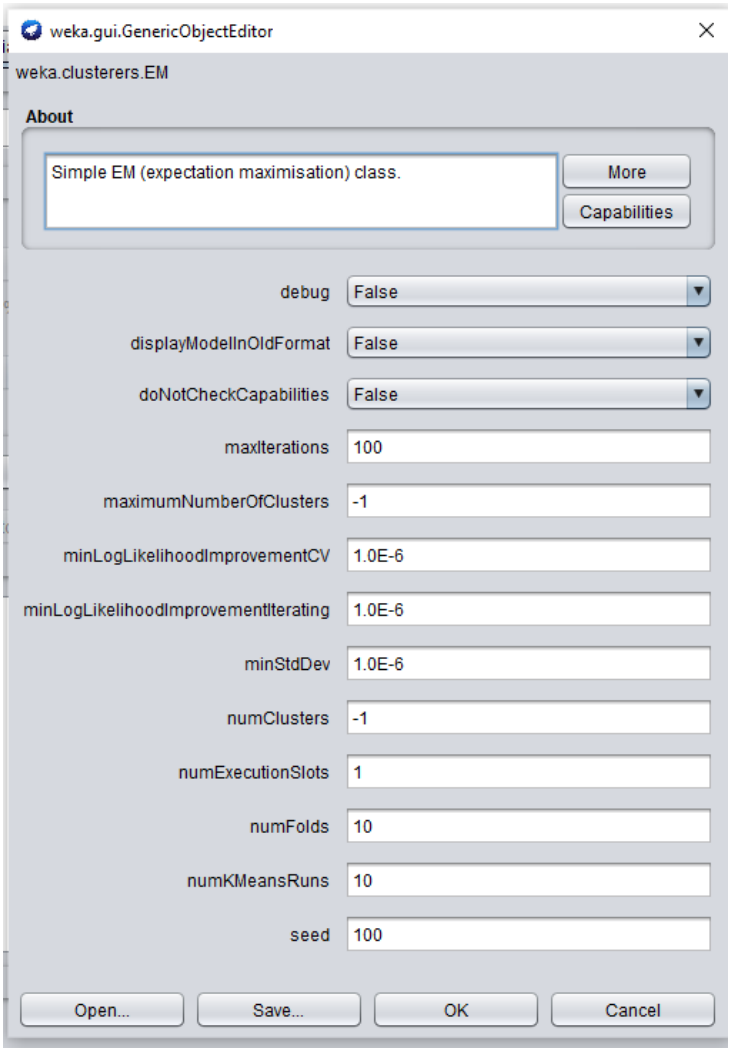
Step 1: In the preprocessing interface, open the Weka Explorer and load the required dataset, and we are taking the iris.arff dataset.



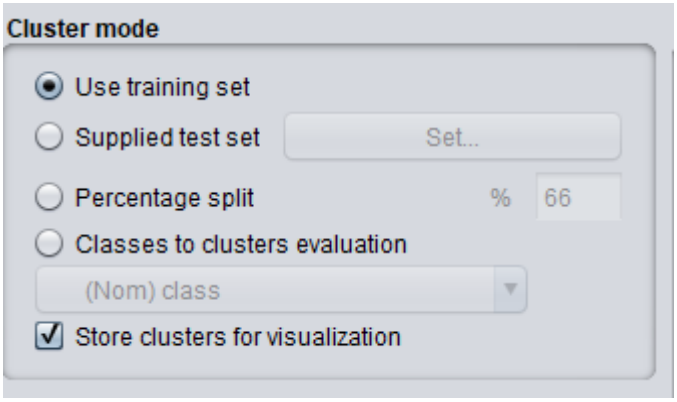
Step 2: Find the 'cluster' tab in the explorer and press the choose button to execute clustering. A dropdown list of available clustering algorithms appears as a result of this step and selects the simple-k means algorithm.

Step 3: Then, to the right of the choose icon, press the text button to bring up the popup window shown in the screenshots. We enter three for the number of clusters in this window and leave the seed value alone.

The seed value is used to generate a random number that is used to make internal assignments of instances of clusters.



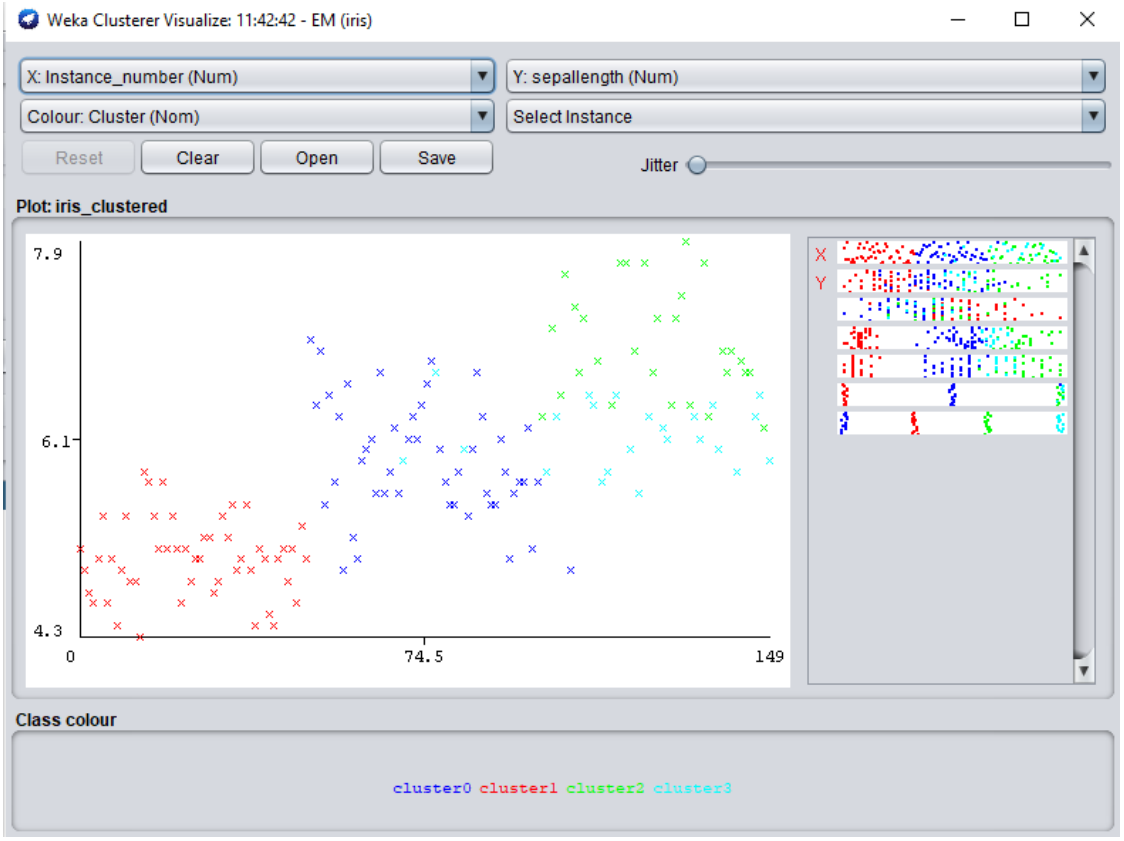
Step 4: One of the choices has been chosen. We must ensure that they are in the 'cluster mode' panel before running the clustering algorithm. The choice to use a training set is selected, and then the 'start' button is pressed. The screenshots below display the process and the resulting window.



Step 5: The centroid of each cluster is shown in the result window, along with statistics on the number and percent of instances allocated to each cluster. Each cluster centroid is represented by a mean vector. This cluster can be used to describe a cluster.

```
Number of clusters selected by cross validation: 4
Number of iterations performed: 16


                     Cluster
Attribute                 0       1       2       3
                     (0.32)  (0.33)   (0.2)  (0.14)
=========================================================
sepallength
  mean                 5.897   5.006  6.9426  6.1304
  std. dev.           0.5279  0.3489   0.498  0.2943

sepalwidth
  mean                2.7519   3.418  3.1103  2.8088
  std. dev.           0.3103  0.3772  0.2952  0.2361

petallength
  mean                4.2267   1.464  5.8559  5.0993
  std. dev.            0.445  0.1718  0.4626  0.2462

petalwidth
  mean                1.3134   0.244  2.1495  1.8254
  std. dev.           0.1864  0.1061   0.232  0.2152

class
  Iris-setosa              1      51       1       1
  Iris-versicolor    48.1125       1  1.0182  3.8693
  Iris-virginica      2.0983       1 31.0375 19.8641
  [total]            51.2108      53 33.0557 24.7335
```

Step 6: Another way to grasp the characteristics of each cluster is to visualize them. To do so, right-click the result set on the result. Selecting to visualize cluster assignments from the list column.



TASK 6

Implement Decision Tree classification algorithm in R and Weka

Decision Tree classification algorithm in R :

Import the data

```
set.seed(678)
path <- 'https://raw.githubusercontent.com/guru99-edu/R-Programming/master/titanic_data.csv'
titanic <-read.csv(path)
head(titanic)
```

Output:

```
##   X pclass survived                                            name    sex
## 1 1      1        1                   Allen, Miss. Elisabeth Walton female
## 2 2      1        1                  Allison, Master. Hudson Trevor   male
## 3 3      1        0                    Allison, Miss. Helen Loraine female
## 4 4      1        0            Allison, Mr. Hudson Joshua Creighton   male
## 5 5      1        0 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6 6      1        1                         Anderson, Mr. Harry   male
##       age sibsp parch ticket     fare   cabin embarked
## 1 29.0000     0     0  24160 211.3375      B5        S
## 2  0.9167     1     2 113781 151.5500 C22 C26        S
## 3  2.0000     1     2 113781 151.5500 C22 C26        S
## 4 30.0000     1     2 113781 151.5500 C22 C26        S
## 5 25.0000     1     2 113781 151.5500 C22 C26        S
## 6 48.0000     0     0  19952  26.5500     E12        S
##                        home.dest
## 1                    St Louis, MO
## 2 Montreal, PQ / Chesterville, ON
## 3 Montreal, PQ / Chesterville, ON
## 4 Montreal, PQ / Chesterville, ON
## 5 Montreal, PQ / Chesterville, ON
## 6    New York, NY
```

Step 2) Clean the dataset

```
library(dplyr)
# Drop variables
clean_titanic <- titanic % > %
select(-c(home.dest, cabin, name, X, ticket)) % > %
#Convert to factor level
            mutate(pclass = factor(pclass, levels = c(1, 2, 3), labels = c('Upper', 'Middle', 'Lower')),
            survived = factor(survived, levels = c(0, 1), labels = c('No', 'Yes'))) % > %
na.omit()
glimpse(clean_titanic)
```

Step 3) Create train/test set

```
create_train_test(df, size = 0.8, train = TRUE)
arguments:
```

-df: Dataset used to train the model.
-size: Size of the split. By default, 0.8. Numerical value
-train: If set to `TRUE`, the function creates the train set, otherwise the test set. Default value sets to `TRUE`. Boolean value.You need to add a Boolean parameter because R does not allow to return two data frames simultaneously.

```
create_train_test <- function(data, size = 0.8, train = TRUE) {
    n_row = nrow(data)
    total_row = size * n_row
    train_sample < - 1: total_row
    if (train == TRUE) {
        return (data[train_sample, ])
    } else {
        return (data[-train_sample, ])
    }
}
```

Install rpart.plot

rpart.plot is not available from conda libraries. You can install it from the console:

```
install.packages("rpart.plot")
```

Step 4) Build the model

You are ready to build the model. The syntax for Rpart decision tree function is:

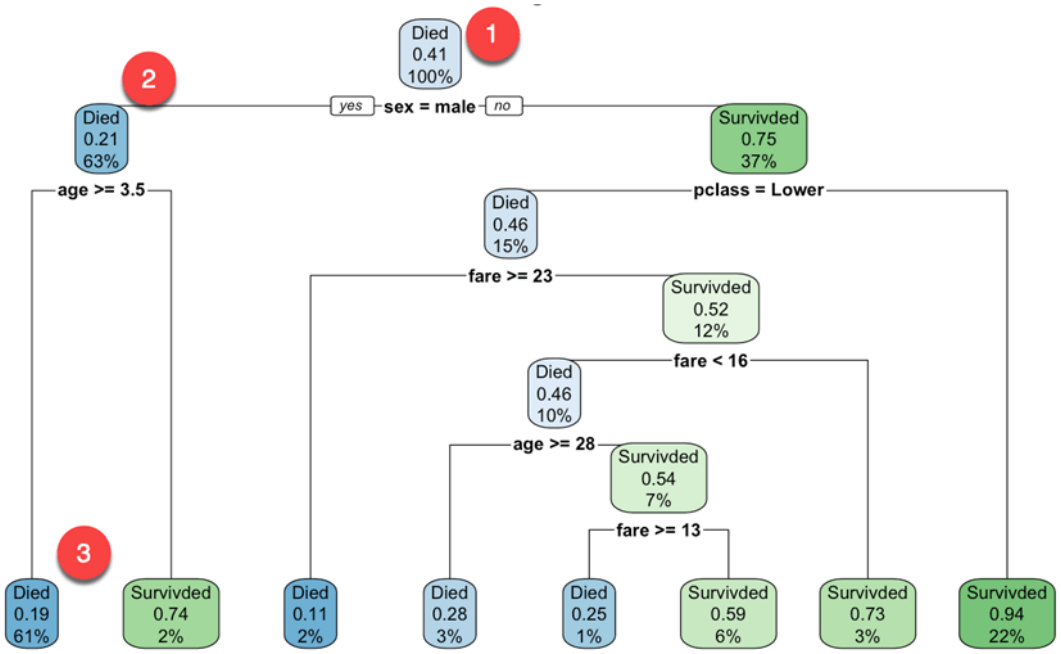
```
rpart(formula, data=, method='')
```

arguments:
- formula: The function to predict
- data: Specifies the data frame- method:
- "class" for a classification tree
- "anova" for a regression tree

You use the class method because you predict a class.

```
library(rpart)
library(rpart.plot)
fit <- rpart(survived~., data = data_train, method = 'class')
rpart.plot(fit, extra = 106
```

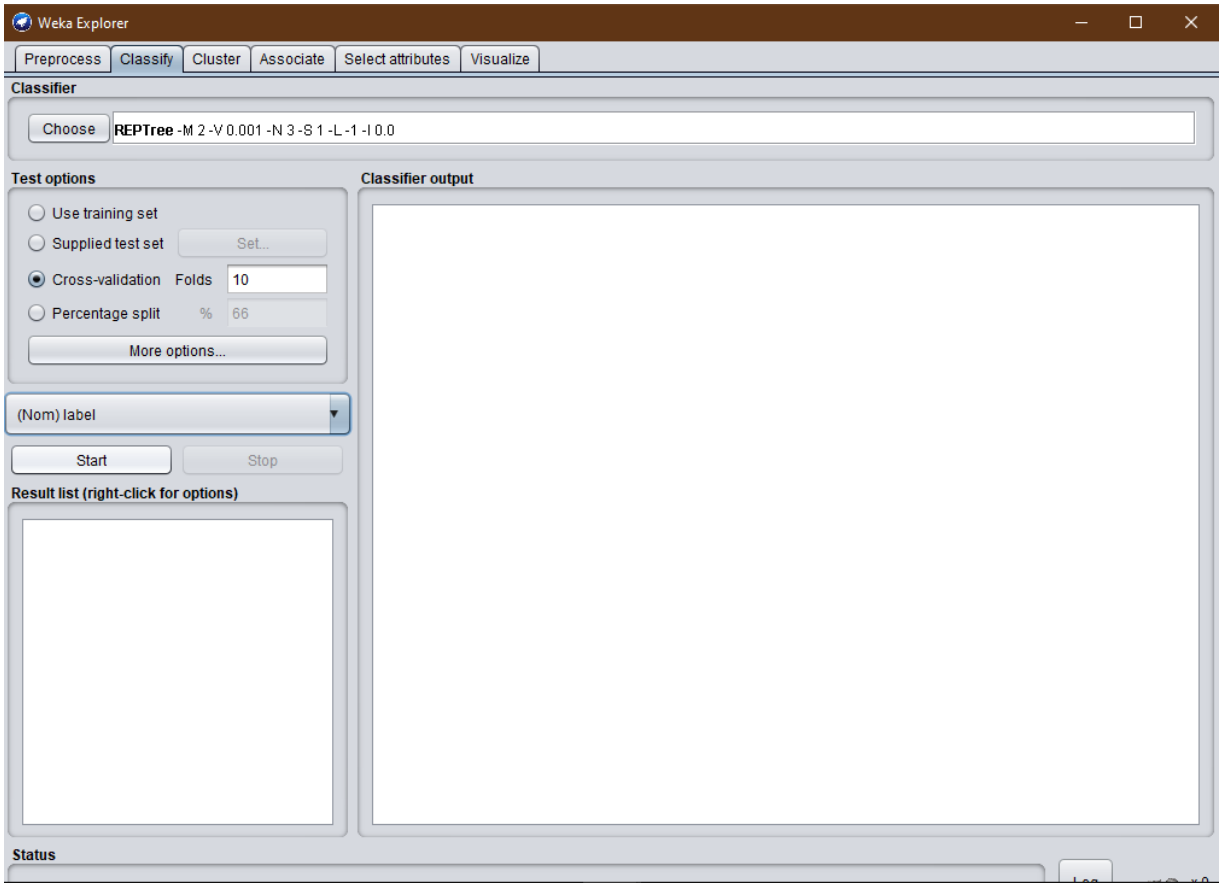
Output:



Decision Tree classification algorithm in Weka

Implementing a decision tree in Weka is pretty straightforward. Just complete the following steps:

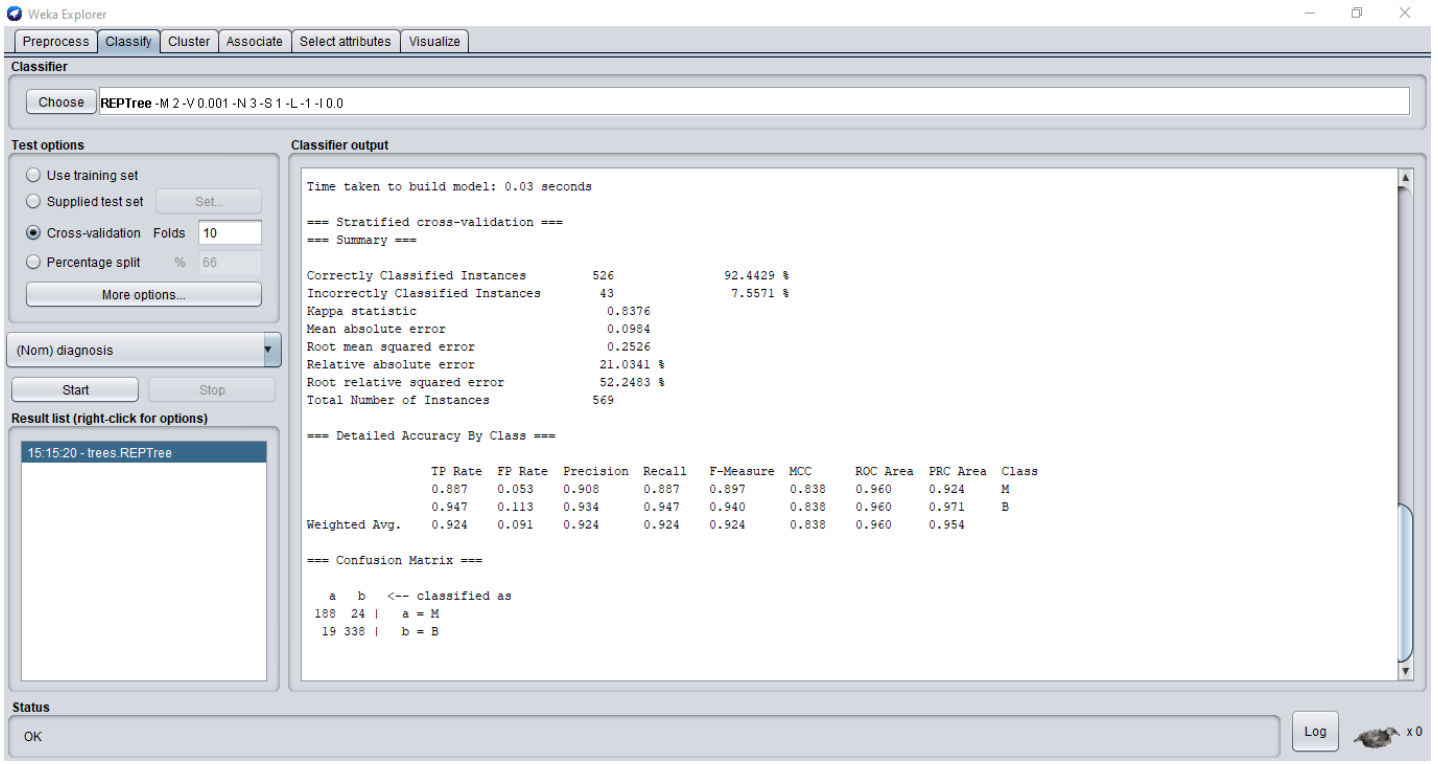
1. Click on the **"Classify"** tab on the top
2. Click the **"Choose"** button
3. From the drop-down list, select **"trees"** which will open all the tree algorithms
4. Finally, select **the "RepTree"** decision tree



You can select your target feature from the drop-down just above the **"Start"** button. If you don't do that, WEKA automatically selects the last feature as the target for you.

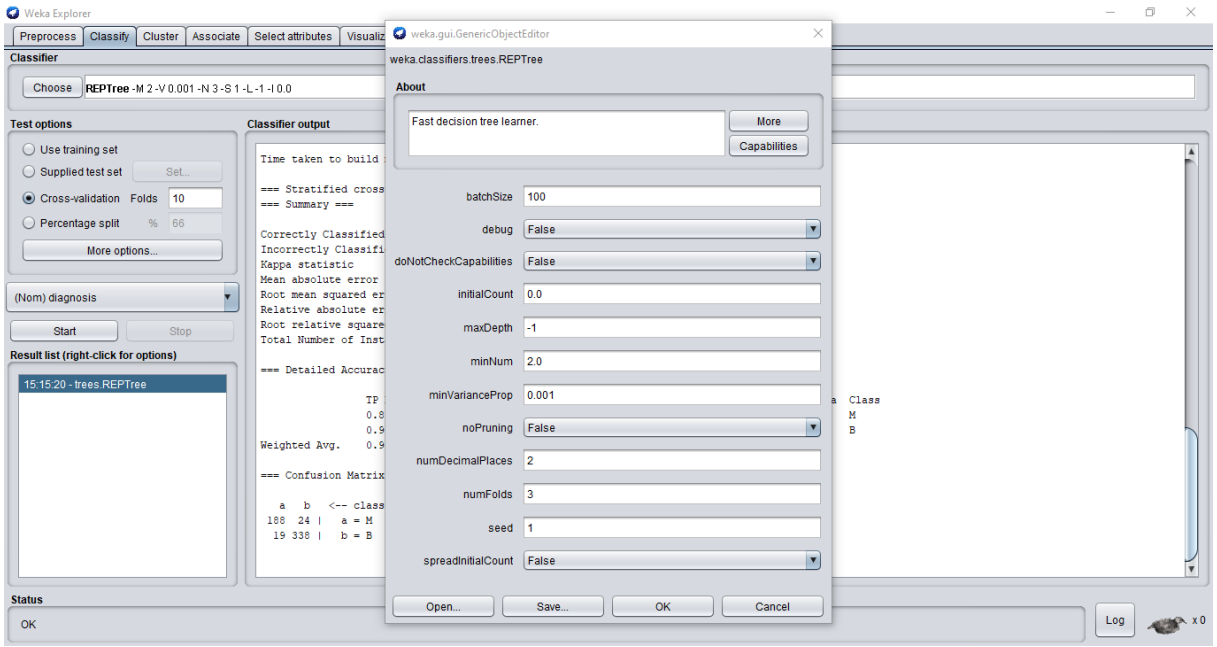
The **"Percentage split"** specifies how much of your data you want to keep for training the classifier. The rest of the data is used during the testing phase to calculate the accuracy of the model. With **"Cross-validation Fold"** you can create multiple samples (or folds) from the training dataset. If you decide to create N folds, then the model is iteratively run N times. And each time one of the folds is held back for validation while the remaining N-1 folds are used for training the model. The result of all the folds is averaged to give the result of cross-validation.

The greater the number of cross-validation folds you use, the better your model will become. This makes the model train on randomly selected data which makes it more robust.



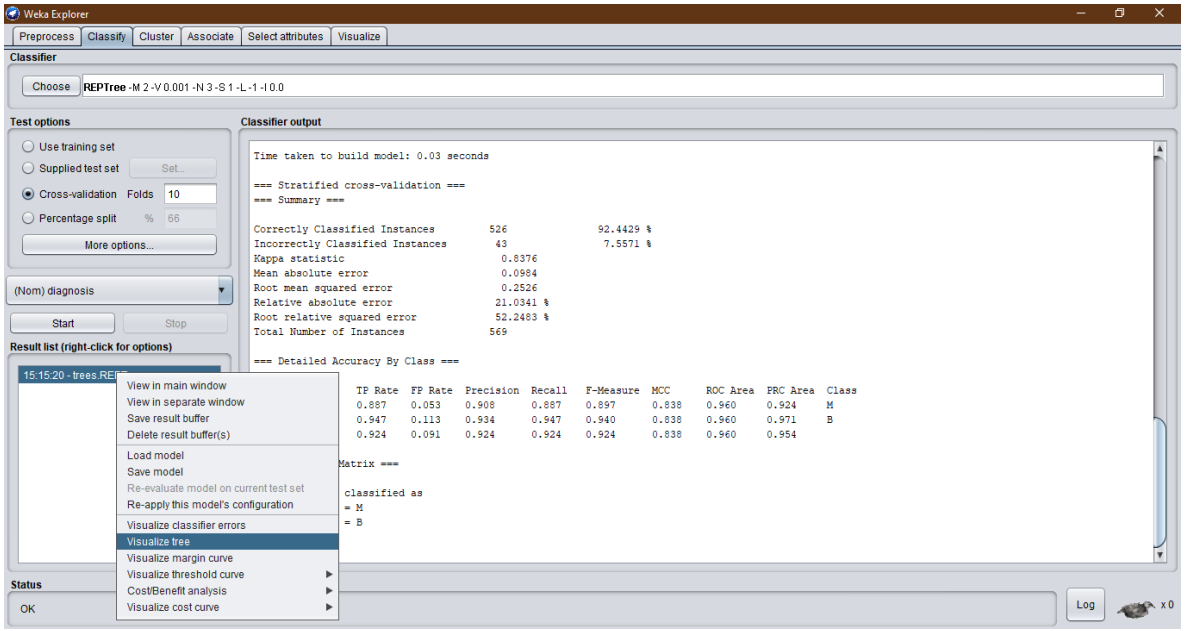
Our classifier has got an accuracy of 92.4%. Weka even prints the **Confusion matrix** for you which gives different metrics.

Decision Tree Parameters in Weka Decision trees have a lot of parameters. We can tune these to improve our model's overall performance. This is where a working knowledge of decision trees really plays a crucial role.

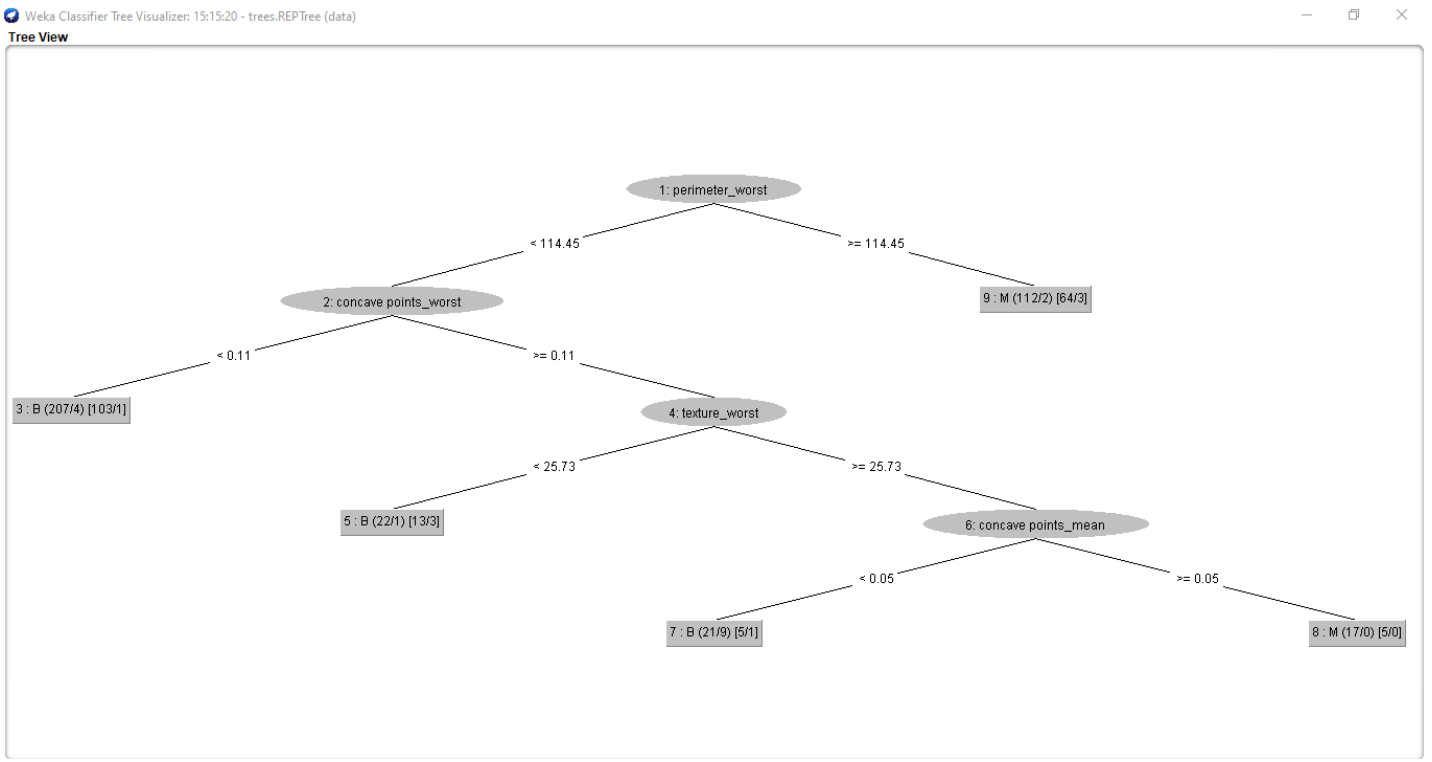


Visualizing your Decision Tree in Weka Weka even allows you to easily visualize the decision tree built on your dataset:

1. Go to the **"Result list"** section and right-click on your trained algorithm
2. Choose the **"Visualise tree"** option



Decision tree will look like below:



TASK 8

Implement Distance and density-based anomaly detection (K-NN) algorithm

Machine learning finds extensive usage in pharmaceutical industry especially in detection of oncogenic (cancer cells) growth. R finds application in machine learning to build models to predict the abnormal growth of cells thereby helping in detection of cancer and benefiting the health system.

Let's see the process of building this model using kNN algorithm in R Programming.

Step 1- Data collection

We will use a data set of 100 patients (created solely for the purpose of practice) to implement the knn algorithm and thereby interpreting results .

The data set consists of 100 observations and 10 variables (out of which 8 numeric variables and one categorical variable and is ID) which are as follows:

1. Radius
2. Texture
3. Perimeter
4. Area
5. Smoothness
6. Compactness
7. Symmetry
8. Fractal dimension

Here's how the data set looks like:

	A	B	C	D	E	F	G	H	I	J	K
1	id	diagnosis_result	radius	texture	perimeter	area	smoothness	compactness	symmetry	fractal_dimension	
2	1	M	23	12	151	954	0.143	0.278	0.242	0.079	
3	2	B	9	13	133	1326	0.143	0.079	0.181	0.057	
4	3	M	21	27	130	1203	0.125	0.16	0.207	0.06	
5	4	M	14	16	78	386	0.07	0.284	0.26	0.097	
6	5	M	9	19	135	1297	0.141	0.133	0.181	0.059	
7	6	B	25	25	83	477	0.128	0.17	0.209	0.076	
8	7	M	16	26	120	1040	0.095	0.109	0.179	0.057	
9	8	M	15	18	90	578	0.119	0.165	0.22	0.075	
10	9	M	19	24	88	520	0.127	0.193	0.235	0.074	
11	10	M	25	11	84	476	0.119	0.24	0.203	0.082	
12	11	M	24	21	103	798	0.082	0.067	0.153	0.057	
13	12	M	17	15	104	781	0.097	0.129	0.184	0.061	
14	13	B	14	15	132	1123	0.097	0.246	0.24	0.078	
15	14	M	12	22	104	783	0.084	0.1	0.185	0.053	
16	15	M	12	13	94	578	0.113	0.229	0.207	0.077	
17	16	M	22	19	97	659	0.114	0.16	0.23	0.071	
18	17	M	10	16	95	685	0.099	0.072	0.159	0.059	
19	18	M	15	14	108	799	0.117	0.202	0.216	0.074	
20	19	M	20	14	130	1260	0.098	0.103	0.158	0.054	

Prostate_Cancer

Step 2- Preparing and exploring the data

```
[Previously saved workspace restored]

> setwd("C:/Users/Payal/Desktop/KNN")
> prc <- read.csv("Prostrate_Cancer.csv",stringsAsFactors = FALSE)
> str(prc)
'data.frame':   100 obs. of  10 variables:
 $ id                : int  1 2 3 4 5 6 7 8 9 10 ...
 $ diagnosis_result : chr  "M" "B" "M" "M" ...
 $ radius            : int  23 9 21 14 9 25 16 15 19 25 ...
 $ texture           : int  12 13 27 16 19 25 26 18 24 11 ...
 $ perimeter         : int  151 133 130 78 135 83 120 90 88 84 ...
 $ area              : int  954 1326 1203 386 1297 477 1040 578 520 476 ...
 $ smoothness        : num  0.143 0.143 0.125 0.07 0.141 0.128 0.095 0.119 0.127 0.119 ...
 $ compactness       : num  0.278 0.079 0.16 0.284 0.133 0.17 0.109 0.165 0.193 0.24 ...
 $ symmetry          : num  0.242 0.181 0.207 0.26 0.181 0.209 0.179 0.22 0.235 0.203 ...
 $ fractal_dimension: num  0.079 0.057 0.06 0.097 0.059 0.076 0.057 0.075 0.074 0.082 ...
>
```

setwd("C:/Users/Payal/Desktop/KNN") #Using this command, we've imported the 'Prostate_Cancer.csv' data file. This command is used to point to the folder containing the required file. Do keep in mind, that it's a common mistake to use "\" instead of "/" after the setwd command.

prc <- read.csv("Prostate_Cancer.csv",stringsAsFactors = FALSE) #This command imports the required data set and saves it to the prc data frame.

stringsAsFactors = FALSE #This command helps to convert every character vector to a factor wherever it makes sense.

str(prc) #We use this command to see whether the data is structured or not.

We find that the data is structured with 10 variables and 100 observations. If we observe the data set, the first variable 'id' is unique in nature and can be removed as it does not provide useful information.

```
> prc <- prc[-1]
> head(prc)
  diagnosis_result radius texture perimeter area smoothness compactness symmetry fractal_dimension
1                M     23      12       151  954      0.143       0.278    0.242              0.079
2                B      9      13       133 1326      0.143       0.079    0.181              0.057
3                M     21      27       130 1203      0.125       0.160    0.207              0.060
4                M     14      16        78  386      0.070       0.284    0.260              0.097
5                M      9      19       135 1297      0.141       0.133    0.181              0.059
6                B     25      25        83  477      0.128       0.170    0.209              0.076
>
```

prc <- prc[-1] #removes the first variable(id) from the data set.

The data set contains patients who have been diagnosed with either Malignant (M) or Benign (B) cancer

table(prc$diagnosis_result) # it helps us to get the numbers of patients

(The variable diagnosis_result is our target variable i.e. this variable will determine the results of the diagnosis based on the 8 numeric variables)

prc$diagnosis <- factor(prc$diagnosis_result, levels = c("B", "M"), labels = c("Benign", "Malignant"))

round(prop.table(table(prc$diagnosis)) * 100, digits = 1) # it gives the result in the percentage form rounded of to 1 decimal place(and so it's digits = 1)

```
> prc$diagnosis <- factor(prc$diagnosis_result, levels = c("B", "M"), labels = c("Benign", "Malignant"))
> round(prop.table(table(prc$diagnosis)) * 100, digits = 1)

   Benign Malignant
       38        62
>
```

Normalizing numeric data

This feature is of paramount importance since the scale used for the values for each variable might be different. The best practice is to normalize the data and transform all the values to a common scale.

normalize <- function(x) {

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

Once we run this code, we are required to normalize the numeric features in the data set. Instead of normalizing each of the 8 individual variables we use:

prc_n <- as.data.frame(lapply(prc[2:9], normalize))

The first variable in our data set (after removal of id) is 'diagnosis_result' which is not numeric in nature. So, we start from 2nd variable. The function lapply() applies normalize() to each feature in the data frame. The final result is stored to prc_n data frame using as.data.frame() function

Let's check using the variable 'radius' whether the data has been normalized.

summary(prc_n$radius)

```
> normalize <- function(x) {
+  return ((x - min(x)) / (max(x) - min(x)))
+  }
> prc_n <- as.data.frame(lapply(prc[2:9], normalize))
> summary(prc_n$radius)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.1875  0.5000  0.4906  0.7500  1.0000
> |
```

Creating training and test data set

The kNN algorithm is applied to the training data set and the results are verified on the test data set.For this, we would divide the data set into 2 portions in the ratio of 65: 35 (assumed) for the training and test data set respectively. We shall divide the prc_n data frame into prc_train and prc_test data frames

prc_train <- prc_n[1:65,]

prc_test <- prc_n[66:100,]

A blank value in each of the above statements indicate that all rows and columns should be included.Our target variable is 'diagnosis_result' which we have not included in our training and test data sets.

prc_train_labels <- prc[1:65, 1]

prc_test_labels <- prc[66:100, 1] #This code takes the diagnosis factor in column 1 of the prc data frame and on turn creates prc_train_labels and prc_test_labels data frame.

Step 3 – Training a model on data

The knn () function needs to be used to train a model for which we need to install a package 'class'. The knn() function identifies the k-nearest neighbors using Euclidean distance where k is a user-specified number.You need to type in the following commands to use knn()

install.packages("class")

library(class)

Now we are ready to use the knn() function to classify test data

prc_test_pred <- knn(train = prc_train, test = prc_test,cl = prc_train_labels, k=10). The value for k is generally chosen as the square root of the number of observations.

knn() returns a factor value of predicted labels for each of the examples in the test data set which is then assigned to the data frame prc_test_pred

```
>  prc_train <- prc_n[1:65, ]
>  prc_test <- prc_n[66:100, ]
> prc_test_labels <- prc[66:100, 1]
> prc_train_labels <- prc[1:65, 1]
> prc_test_labels <- prc[66:100, 1]
> prc_test_pred <- knn(train = prc_train, test = prc_test,cl = prc_train_labels, k=10)
Error: could not find function "knn"
> install.packages("class")
Installing package into 'C:/Users/Payal/Documents/R/win-library/3.1'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'http://mirrors.xmu.edu.cn/CRAN/bin/windows/contrib/3.1/class_7.3-13.zip'
Content type 'application/zip' length 100211 bytes (97 Kb)
opened URL
downloaded 97 Kb

package 'class' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Payal\AppData\Local\Temp\RtmpkPNcE8\downloaded_packages
> library(class)
Warning message:
package 'class' was built under R version 3.1.3
> prc_test_pred <- knn(train = prc_train, test = prc_test,cl = prc_train_labels, k=10)
>
```

TASK 9

Implement Normal distribution in R programming for any dataset.

R has four in built functions to generate normal distribution. They are described below.

dnorm(x, mean, sd)

pnorm(x, mean, sd)

qnorm(p, mean, sd)

rnorm(n, mean, sd)

Following is the description of the parameters used in above functions −

x is a vector of numbers.

p is a vector of probabilities.

n is number of observations(sample size).

mean is the mean value of the sample data. It's default value is zero.

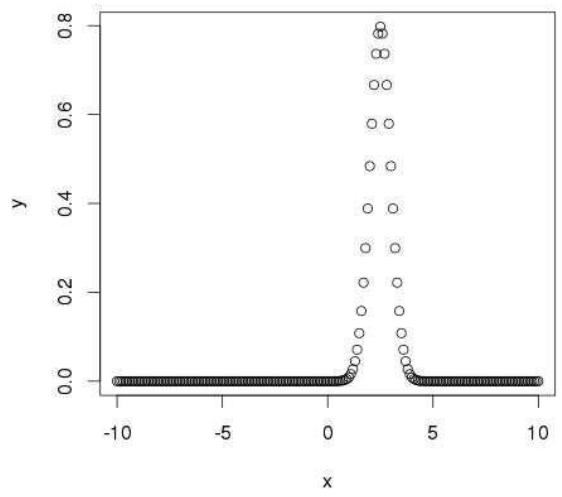
sd is the standard deviation. It's default value is 1.

dnorm()

This function gives height of the probability distribution at each point for a given mean and standard deviation.

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.1.

x <- seq(-10, 10, by = .1)

# Choose the mean as 2.5 and standard deviation as 0.5.

y <- dnorm(x, mean = 2.5, sd = 0.5)

# Give the chart file a name.

png(file = "dnorm.png")

plot(x,y)

# Save the file.

dev.off()
```

When we execute the above code, it produces the following result −



pnorm()

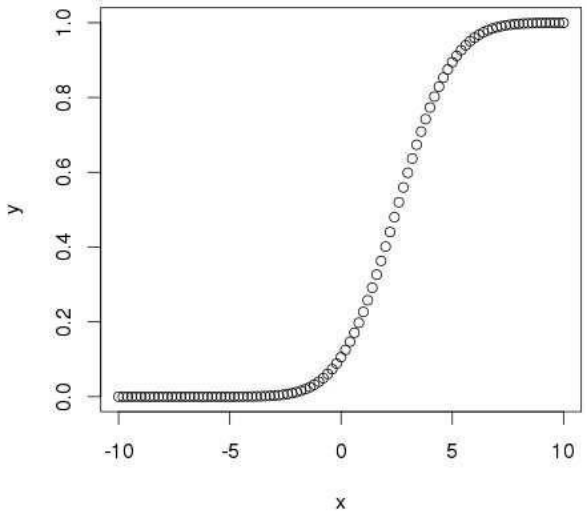
This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

Live Demo

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.2.

x <- seq(-10,10,by = .2)



# Choose the mean as 2.5 and standard deviation as 2.
```

```
y <- pnorm(x, mean = 2.5, sd = 2)

# Give the chart file a name.

png(file = "pnorm.png")

# Plot the graph.

plot(x,y)

# Save the file.

dev.off()
```

When we execute the above code, it produces the following result −



qnorm()

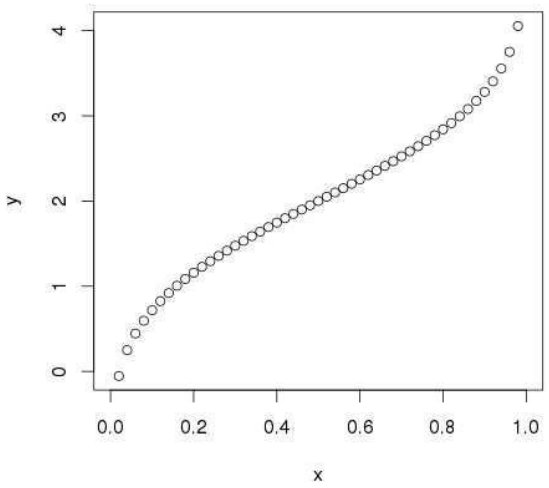
This function takes the probability value and gives a number whose cumulative value matches the probability value.

Live Demo

```
# Create a sequence of probability values incrementing by 0.02.

x <- seq(0, 1, by = 0.02)

# Choose the mean as 2 and standard deviation as 3.

y <- qnorm(x, mean = 2, sd = 1)

# Give the chart file a name.

png(file = "qnorm.png")

# Plot the graph.
```

```
plot(x,y)

# Save the file.

dev.off()
```

When we execute the above code, it produces the following result −



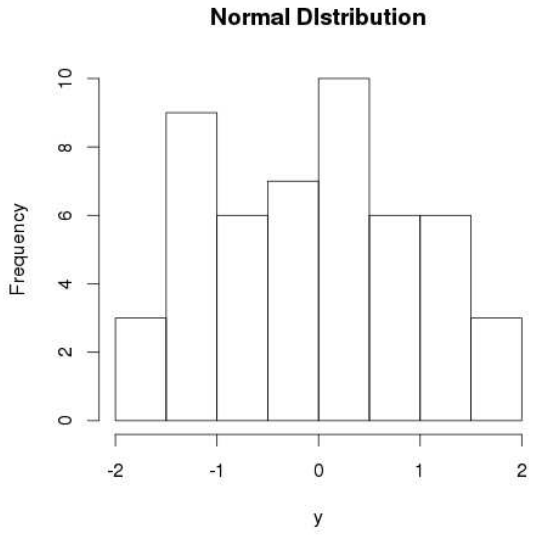
rnorm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

Live Demo

```
# Create a sample of 50 numbers which are normally distributed.

y <- rnorm(50)

# Give the chart file a name.

png(file = "rnorm.png")

# Plot the histogram for this sample.

hist(y, main = "Normal DIstribution")

# Save the file.

dev.off()
```

When we execute the above code, it produces the following result −



TASK 10

Implement Expectation Maximization in R programming to solve exponential distribution

```
#------------------------------- Expectation ----------------------------

expectation <- function(sample,p,a,b)

  {

  p_expectation <- (p*dbinom(sample,1,a)) / ( p*dbinom(sample,1,a) + (1-p)*dbinom(sample,1,b) )

  return(p_expectation)

}

#------------------------------- Maximization   -------------------------------

maximization <- function(sample,epart){

    # estimate p

   p_temp <- mean(epart)

    # estimate a and b

    a_temp <- sum(sample*epart) / sum(epart)

  b_temp <- sum(sample*(1-epart)) / sum(1-epart)

    list(p_temp,a_temp,b_temp)

}

#--------------------------- Expectation Maximization Algorithm  ------------------------
```

```
EM <- function(sample,p_inits,a_inits,b_inits,maxit=1000,tol=1e-6)
  {
  # Estimation of parameter(Initial)
  flag <- 0
  p_cur <- p_inits; a_cur <- a_inits; b_cur <- b_inits
    # Iterate between expectation and maximization parts
    for(i in 1:maxit){
    cur <- c(p_cur,a_cur,b_cur)
    new <- maximization(sample,expectation(sample, p_cur, a_cur, b_cur))
    p_new <- new[[1]]; a_new <- new[[2]]; b_new <- new[[3]]
    new_step <- c(p_new,a_new,b_new)
        # Stop iteration if the difference between the current and new estimates is less than a tolerance level
    if( all(abs(cur - new_step) < tol) ){ flag <- 1; break}
          # Otherwise continue iteration
    p_cur <- p_new; a_cur <- a_new; b_cur <- b_new
  }
  if(!flag) warning("Didn't converge\n")
    list(p_cur, a_cur, b_cur)
}




#------------------------------ Calculating Information matrix ----------------------------
Info.Mat.function <- function(sample, p.est, a.est, b.est){
  expectation.est <- expectation(sample,p.est, a.est, b.est)
  info.mat <- matrix(rep(0,9),3,3)
  info.mat[1,1] <- - sum(expectation.est)/(p.est^2)  - sum((1-expectation.est))/((1-p.est)^2)
  info.mat[2,2] <- - sum(expectation.est*sample)/(a.est^2) - sum(expectation.est*(1-sample))/((1-a.est)^2)
```

```
                info.mat[3,3] <- - sum((1-expectation.est)*sample)/(b.est^2) - sum((1-expectation.est)*(1-sample))/((1-b.est)^2)

  return(-info.mat)

}

#------------------ Now Generate sample data -------------------------------

n <- 10000

p_true <- 0.85 # prob of using first coin

a_true <-  0.50 # the first coin has P(heads) = 0.50

b_true <-  0.70 # the second coin has P(heads) = 0.70

true <- c(p_true,a_true,b_true)

u <- ifelse(runif(n)<p_true, rbinom(n,1,a_true),rbinom(n,1,b_true))

# Set parameter estimates

p_init = 0.70; a_init = 0.70; b_init = 0.60

#--------------------Return EM Algorithm function and calculate Confidence Interval----------------------------

output <- EM(u,p_init,a_init,b_init)

# Confidence Intervals

sd.out <- sqrt(diag(solve(Info.Mat.function(u,output[[1]],output[[2]],output[[3]]))))

data.frame("Truth" = true, "EM Estimate" = unlist(output), "Lower CI" = unlist(output) - qnorm(.975)*sd.out, "Upper CI" = unlist(output) + qnorm(.975)*sd.out)

##   Truth EM.Estimate  Lower.CI  Upper.CI

## 1  0.85   0.6863229 0.6772290 0.6954169

## 2  0.50   0.5605037 0.5487615 0.5722460

## 3  0.70   0.4505061 0.4330945 0.4679177
```

TASK 11

Apply EM algorithm to cluster a set of data stored in a .CSV file and use the same data set for clustering using k-Means algorithm in R. Compare the results of these two algorithms and comment on the quality of clustering

K-Means clustering groups the data on similar groups. The algorithm is as follows:

1. Choose the number **K** clusters.

2. Select at random K points, the centroids(Not necessarily from the given data).
3. Assign each data point to closest centroid that forms K clusters.
4. Compute and place the new centroid of each centroid.
5. Reassign each data point to new cluster.

After final reassignment, name the cluster as Final cluster.

The Dataset

`Iris` dataset consists of 50 samples from each of 3 species of Iris(Iris setosa, Iris virginica, Iris versicolor) and a multivariate dataset introduced by British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems. Four features were measured from each sample i.e length and width of the sepals and petals and based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

```
# Loading data
data(iris)

# Structure
str(iris)
```

Performing K-Means Clustering on Dataset

Using K-Means Clustering algorithm on the dataset which includes 11 persons and 6 variables or attributes

```
# Installing Packages
install.packages("ClusterR")
install.packages("cluster")

# Loading package
library(ClusterR)
library(cluster)

# Removing initial label of
# Species from original dataset
iris_1 <- iris[, -5]

# Fitting K-Means clustering Model
# to training dataset
set.seed(240) # Setting seed
kmeans.re <- kmeans(iris_1, centers = 3, nstart = 20)
kmeans.re

# Cluster identification for
# each observation
kmeans.re$cluster

# Confusion Matrix
cm <- table(iris$Species, kmeans.re$cluster)
cm
```

```
# Model Evaluation and visualization
plot(iris_1[c("Sepal.Length", "Sepal.Width")])
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
    col = kmeans.re$cluster)
plot(iris_1[c("Sepal.Length", "Sepal.Width")],
    col = kmeans.re$cluster,
    main = "K-means with 3 clusters")

## Plotiing cluster centers
kmeans.re$centers
kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")]

# cex is font size, pch is symbol
points(kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")],
      col = 1:3, pch = 8, cex = 3)

## Visualizing clusters
y_kmeans <- kmeans.re$cluster
clusplot(iris_1[, c("Sepal.Length", "Sepal.Width")],
        y_kmeans,
        lines = 0,
        shade = TRUE,
        color = TRUE,
        labels = 2,
        plotchar = FALSE,
        span = TRUE,
        main = paste("Cluster iris"),
        xlab = 'Sepal.Length',
        ylab = 'Sepal.Width')
```

Output:

```
> kmeans.re
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

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
 [44] 1 1 1 1 1 1 1 1 2 2 3 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 2 2 2
 [87] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3
[130] 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

The 3 clusters are made which are of 50, 62, and 38 sizes respectively. Within the cluster, the sum of squares is 88.4%.

Cluster identification:

```
> confusionMatrix(cm)
Confusion Matrix and Statistics


            setosa versicolor virginica
  setosa         20          0         0
  versicolor      0         20         0
  virginica       0          0        20

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9404, 1)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            1.0000           1.0000
Specificity                 1.0000            1.0000           1.0000
Pos Pred Value              1.0000            1.0000           1.0000
Neg Pred Value              1.0000            1.0000           1.0000
Prevalence                  0.3333            0.3333           0.3333
Detection Rate              0.3333            0.3333           0.3333
Detection Prevalence        0.3333            0.3333           0.3333
Balanced Accuracy           1.0000            1.0000           1.0000
```

The model achieved an accuracy of 100% with a p-value of less than 1. This indicates the model is good.

Run the EM algorithm for clustering.

Usage

EM(d, clusters, model = "VVV", ...)

Arguments

D The dataset (matrix or data.frame).

clusters Either an integer (the number of clusters) or a (vector) indicating the cluster to which each point is initially allocated.

model A character string indicating the model.

... Other parameters.

Value

A clustering model obtained by EM.

Examples

require (datasets)
data (iris)
EM (iris [, -5], 3) # Default initialization
km = KMEANS (iris [, -5], k = 3)
EM (iris [, -5], km$cluster) # Initialization with another clustering method

The accuracy we get from **`K Mean algorithm`** is 0.24 which is less than that of accuracy we get from **`EM algorithm`** that is 0.33 on same data-set (iris data-set).

TASK 12

Write R program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

Title: Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. ca = number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. Heartdisease: It is integer valued from 0 (no presence) to 4. Diagnosis of heart disease

(angiographic disease status)

Some instance from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Program:

```
import numpy as np import
csv
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.inference import
VariableElimination

#read Cleveland Heart Disease data

heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)


#display the data

print('Few examples from the dataset are given below') print(heartDisease.head())


#Model Bayesian Network

Model=BayesianModel([('age','trestbps'),('age','fbs'),
('sex','trestbps'),('exang','trestbps'),('trestbps','heartdise
ase'),('fbs','heartdisease'),('heartdisease','restecg'),
('heartdisease','thalach'),('heartdisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators

print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network
```

```
print('\n Inferencing with Bayesian Network:') HeartDisease_infer = VariableElimination(model)
```

```
#computing the Probability of HeartDisease given Age

print('\n 1. Probability of HeartDisease given Age=30')
q=HeartDisease_infer.query(variables=['heartdisease'],evidence
={'age':28})
print(q['heartdisease'])

#computing the Probability of HeartDisease given cholesterol print('\n 2. Probability of HeartDisease
given cholesterol=100') q=HeartDisease_infer.query(variables=['heartdisease'],evidence
={'chol':100})
print(q['heartdisease'])
```

Output:

Few examples from the dataset are given below

	age	sex	cp	trestbps	...slope	ca	thal	heartdisease
0	63	1	1	145	...	3	0	6	0
1	67	1	4	160	...	2	3	3	2
2	67	1	4	120	...	2	2	7	1
3	37	1	3	130	...	3	0	3	0
4	41	0	2	130	...	1	0	3	0

[5 rows x 14 columns]

Learning CPD using Maximum likelihood estimators Inferencing with

Bayesian Network:

1. Probability of HeartDisease given Age=28

heartdisease	phi(heartdisease)
heartdisease_0	0.6791
heartdisease_1	0.1212
heartdisease_2	0.0810
heartdisease_3	0.0939
heartdisease_4	0.0247

2. Probability of HeartDisease given cholesterol=100

heartdisease	phi(heartdisease)
heartdisease_0	0.5400
heartdisease_1	0.1533
heartdisease_2	0.1303
heartdisease_3	0.1259
heartdisease_4	0.0506