# Part-A: Data Mining

## 1. Demonstration of preprocessing on dataset student.arff

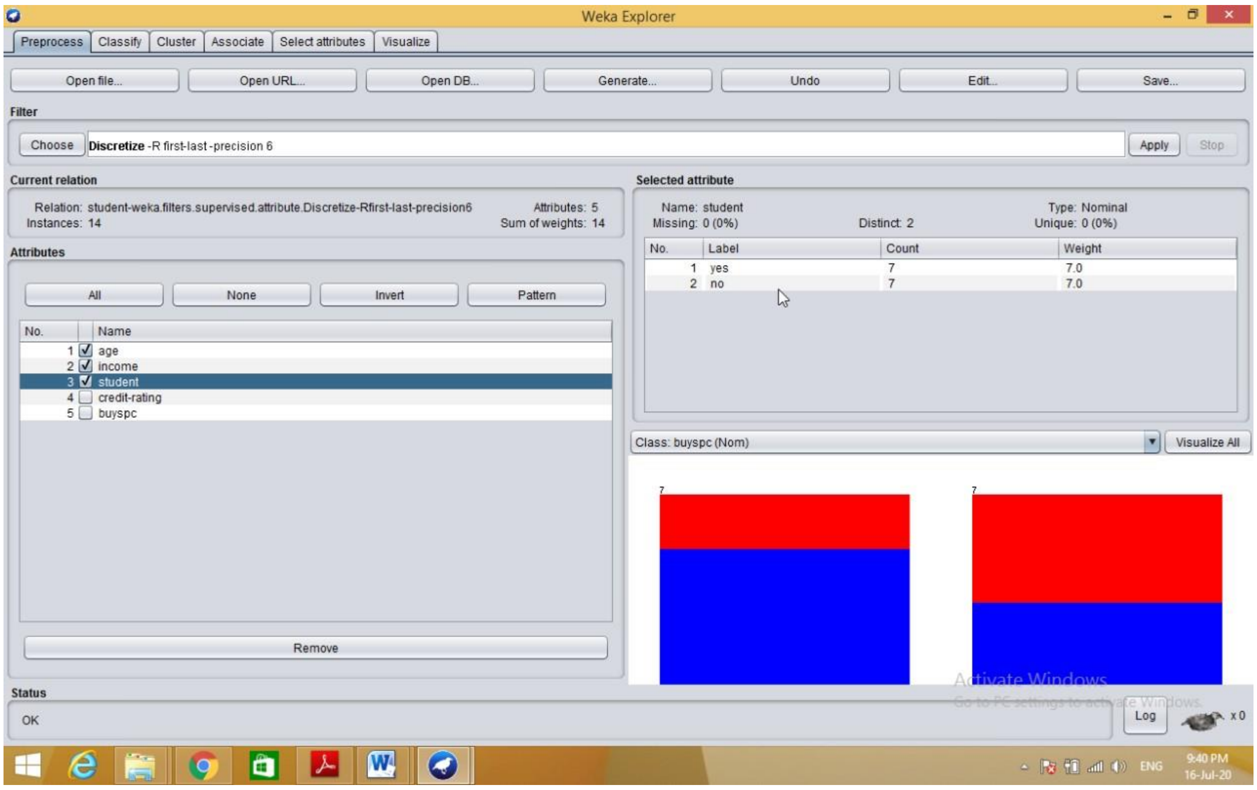
**Aim:** This experiment illustrates some of the basic data preprocessing operations that can be performed using WEKA-Explorer. The sample dataset used for this example is the student data available in arff format.

**Weka - Preprocessing** the Data. The data that is collected from the field contains many unwanted things that leads to wrong analysis. For example, the data may contain null fields, it may contain columns that are irrelevant to the current analysis, and so on.

**Step1:** Loading the data. We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.
**Step2:** Once the data is loaded, weka will recognize the attributes and during the scan of the data weka will compute some basic strategies on each attribute. The left panel in the above figure shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).
**Step3:** Clicking on an attribute in the left panel will show the basic statistics on the attributes for the categorical attributes the frequency of each attribute value is shown, while for continuous attributes we can obtain min, max, mean, standard deviation and deviation etc.,
**Step4:** The visualization in the right button panel in the form of cross-tabulation across two attributes.
Note:we can select another attribute using the dropdown list.
Step5:Selecting or filtering attributes Removing an attribute-When we need to remove an attribute,we can do this by using the attribute filters in weka.In the filter model panel,click on choose button,This will show a popup window with a list of available filters. Scroll down the list and select the "weka.filters.unsupervised.attribute.remove" filters.
Step 6:a)Next click the textbox immediately to the right of the choose button.In the resulting dialog box enter the index of the attribute to be filtered out.
b) Make sure that invert selection option is set to false.The click OK now in the filter box.you will see "Remove-R-7".
c) Click the apply button to apply filter to this data.This will remove the attribute and create new working relation.
d) Save the new working relation as an arff file by clicking save button on the top(button)panel.(student.arff)

**Procedure:**

Choose Tab→Weka→filters→Supervised→attribute→Discretize



Filter: Discretize -R first-last -precision 6

Current relation
Relation: student-weka.filters.supervised.attribute.Discretize-Rfirst-last-precision6 Attributes: 5
Instances: 14 Sum of weights: 14

Attributes

| No. | | Name |
|---|---|---|
| 1 | ✓ | age |
| 2 | ✓ | income |
| 3 | ✓ | student |
| 4 | | credit-rating |
| 5 | | buyspc |

Selected attribute
Name: student Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

| No. | Label | Count | Weight |
|---|---|---|---|
| 1 | yes | 7 | 7.0 |
| 2 | no | 7 | 7.0 |

Class: buyspc (Nom)

Status
OK

**Dataset student .arff**

```
@relation student
@attribute age {<30,30-40,>40}
@attribute income {low, medium, high}
@attribute student {yes, no}
@attribute credit-rating {fair, excellent}
@attribute buyspc {yes, no}
@data
%
<30, high, no, fair, no
<30, high, no, excellent, no
30-40, high, no, fair, yes
>40, medium, no, fair, yes
>40, low, yes, fair, yes
>40, low, yes, excellent, no
30-40, low, yes, excellent, yes
<30, medium, no, fair, no
<30, low, yes, fair, no
>40, medium, yes, fair, yes
<30, medium, yes, excellent, yes
30-40, medium, no, excellent, yes
30-40, high, yes, fair, yes
>40, medium, no, excellent, no
%
```

## 2. Demonstration of Association rule process on dataset contactlenses.arff using apriori algorithm.

**Aim:** This experiment illustrates some of the basic elements of asscociation rule mining using WEKA. The sample dataset used for this example is contactlenses.arff
Step1: Open the data file in Weka Explorer. It is presumed that the required data fields have been discretized. In this example it is age attribute.
Step2: Clicking on the associate tab will bring up the interface for association rule algorithm.
Step3: We will use apriori algorithm. This is the default algorithm.
Step4: Inorder to change the parameters for the run (example support, confidence etc) we click on the text box immediately to the right of the choose button.
**Description:**

**The Apriori Algorithm:** Finding Frequent Itemsets Using Candidate Generation

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties. Apriori employs an iterative approach known as a *level-wise* search, where *k*-itemsets are used to explore (*k*+1)-itemsets.

**Apriori property:** All nonempty subsets of a frequent itemset must also be frequent.
It has two steps.

1) Join step

2) Prune step

**Procedure:**

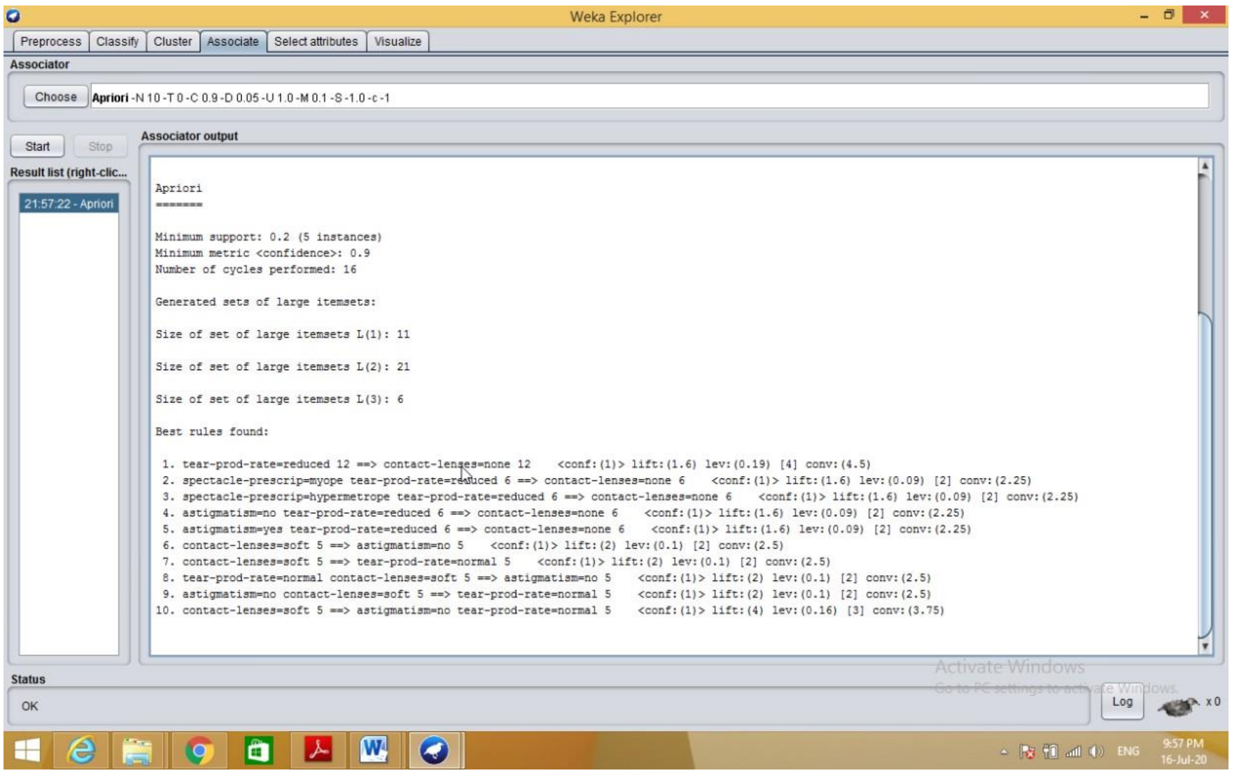
Step 1: Load appropriate dataset into weka

Step 2: Select associate tab and select apriori algorithm

Dataset contactlenses.arff

```
@relation contact-lenses
@attribute age {young,pre-presbyopic,presbyopic}
@attribute spectacle-prescrip {myope,hypermetrope}
@attribute astigmatism {no,yes}
@attribute tear-prod-rate {reduced,normal}
@attribute contact-lenses {soft,hard,none}
@data
%
% 24 instances
%
young,myope,no,reduced,none
young,myope,no,normal,soft
```

```
young,myope,yes,reduced,none
young,myope,yes,normal,hard
young,hypermetrope,no,reduced,none
young,hypermetrope,no,normal,soft
young,hypermetrope,yes,reduced,none
young,hypermetrope,yes,normal,hard
pre-presbyopic,myope,no,reduced,none
pre-presbyopic,myope,no,normal,soft
pre-presbyopic,myope,yes,reduced,none
pre-presbyopic,myope,yes,normal,hard
pre-presbyopic,hypermetrope,no,reduced,none
pre-presbyopic,hypermetrope,no,normal,soft
pre-presbyopic,hypermetrope,yes,reduced,none
pre-presbyopic,hypermetrope,yes,normal,none
presbyopic,myope,no,reduced,none
presbyopic,myope,no,normal,none
presbyopic,myope,yes,reduced,none
presbyopic,myope,yes,normal,hard
presbyopic,hypermetrope,no,reduced,none
presbyopic,hypermetrope,no,normal,soft
presbyopic,hypermetrope,yes,reduced,none
presbyopic,hypermetrope,yes,normal,none

%
%
```



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Associator output

```
Apriori
=======

Minimum support: 0.2 (5 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 16

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 21

Size of set of large itemsets L(3): 6

Best rules found:

 1. tear-prod-rate=reduced 12 ==> contact-lenses=none 12    <conf:(1)> lift:(1.6) lev:(0.19) [4] conv:(4.5)
 2. spectacle-prescrip=myope tear-prod-rate=reduced 6 ==> contact-lenses=none 6    <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
 3. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6 ==> contact-lenses=none 6    <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
 4. astigmatism=no tear-prod-rate=reduced 6 ==> contact-lenses=none 6    <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
 5. astigmatism=yes tear-prod-rate=reduced 6 ==> contact-lenses=none 6    <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
 6. contact-lenses=soft 5 ==> astigmatism=no 5    <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
 7. contact-lenses=soft 5 ==> tear-prod-rate=normal 5    <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
 8. tear-prod-rate=normal contact-lenses=soft 5 ==> astigmatism=no 5    <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
 9. astigmatism=no contact-lenses=soft 5 ==> tear-prod-rate=normal 5    <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
10. contact-lenses=soft 5 ==> astigmatism=no tear-prod-rate=normal 5    <conf:(1)> lift:(4) lev:(0.16) [3] conv:(3.75)
```

## 3. Demonstration of classification rule process on dataset employee.arff using j48 algorithm.

**Aim:** This experiment illustrates the use of j-48 classifier in weka. The sample data set used in this experiment is “student” data available at arff format. This document assumes that appropriate data pre processing has been performed.

Steps involved in this experiment:

Step-1: We begin the experiment by loading the data (student.arff)into weka.

Step2: Next we select the “classify” tab and click “choose” button t o select the “j48”classifier.

Step3: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values. The default version does perform some pruning but does not perform error pruning.

Step4: Under the “text” options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don’t have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

Step-5: We now click ”start” to generate the model .the Ascii version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.

Step-6: Note that the classification accuracy of model is about 69%.this indicates that we may find more work. (Either in preprocessing or in selecting current parameters for the classification)

Step-7: Now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting “visualize tree” from the pop-up menu.

Step-8: We will use our model to classify the new instances.

Step-9: In the main panel under “text” options click the “supplied test set” radio button and then click the “set” button. This wills pop-up a window which will allow you to open the file containing test instances.

**Description:**

**Classification** is a **data mining** function that assigns items in a collection to target categories or classes. The goal of **classification** is to accurately predict the target class for each case in the **data**. For example, a **classification** model could be used to identify loan applicants as low, medium, or high credit risks.

**Entropy** characterizes the (im) purity of an arbitrary collection of examples

Information Gain is the expected reduction in entropy caused by partitioning the examples according to a given attribute

If we have a set with k different values in it, we can calculate the entropy as follows:

$$entropy(Set) = I(Set) = -\sum_{i=1}^{k} P(value_i) \cdot \log_2(P(value_i))$$

Where P(valuei) is the probability of getting the ith value when randomly selecting one from the set. So, for the set R = {a,a,a,b,b,b,b,b}

$$entropy\ (R) = I(R) = -\left[\underbrace{\left(\frac{3}{8}\right)\log_2\left(\frac{3}{8}\right)}_{\text{a-values}} + \underbrace{\left(\frac{5}{8}\right)\log_2\left(\frac{5}{8}\right)}_{\text{b-values}}\right]$$

**Kappa statistic** is a measure of how closely the instances classified by the machine learning classifier matched the **data** labeled as ground truth, controlling for the accuracy of a random classifier as measured by the expected accuracy.

**Decision tree** builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**.

**Classifier Accuracy Measures**

The ***confusion matrix*** is a useful tool for analyzing how well your classifier can recognize tuples of different classes.

| | | **Predicted class** | |
|---|---|---|---|
| | | $C_1$ | $C_2$ |
| **Actual class** | $C_1$ | true positives | false negatives |
| | $C_2$ | false positives | true negatives |

$$sensitivity = \frac{t\_pos}{pos}$$

$$specificity = \frac{t\_neg}{}$$

$prec$

$$accuracy = sensitivity\frac{pos}{(pos+neg)} + specificity\frac{neg}{(pos+neg)}.$$

**F measure** (F1 score or **F** score) is a **measure** of a test's accuracy and is defined as the weighted harmonic mean of the precision and recall of the test.

**TP Rate**: **rate** of true positives (instances correctly classified as a given class)

FP**Rate**: **rate** of false positives (instances falsely classified as a given class)

**Precision** is the fraction of retrieved data that are relevant to the query:

**Recall** is the fraction of the relevant data that are successfully retrieved.

**Cross-validation:** In ***k***-fold **cross-validation**, the initial data are randomly partitioned into *k* mutually exclusive subsets or "folds," *D*1, *D*2: *Dk*, each of approximately equal size.

**Predictor Error Measures**

Absolute error : $|y_i - y_i'|$

Squared error : $(y_i - y_i')^2$

Relative absolute error : $\frac{\sum_{i=1}^{d}|y_i - y_i'|}{\sum_{i=1}^{d}|y_i - \bar{y}|}$

Relative squared error : $\frac{\sum_{i=1}^{d}(y_i - y_i')^2}{\sum_{i=1}^{d}(y_i - \bar{y})^2}$

**Procedure:**

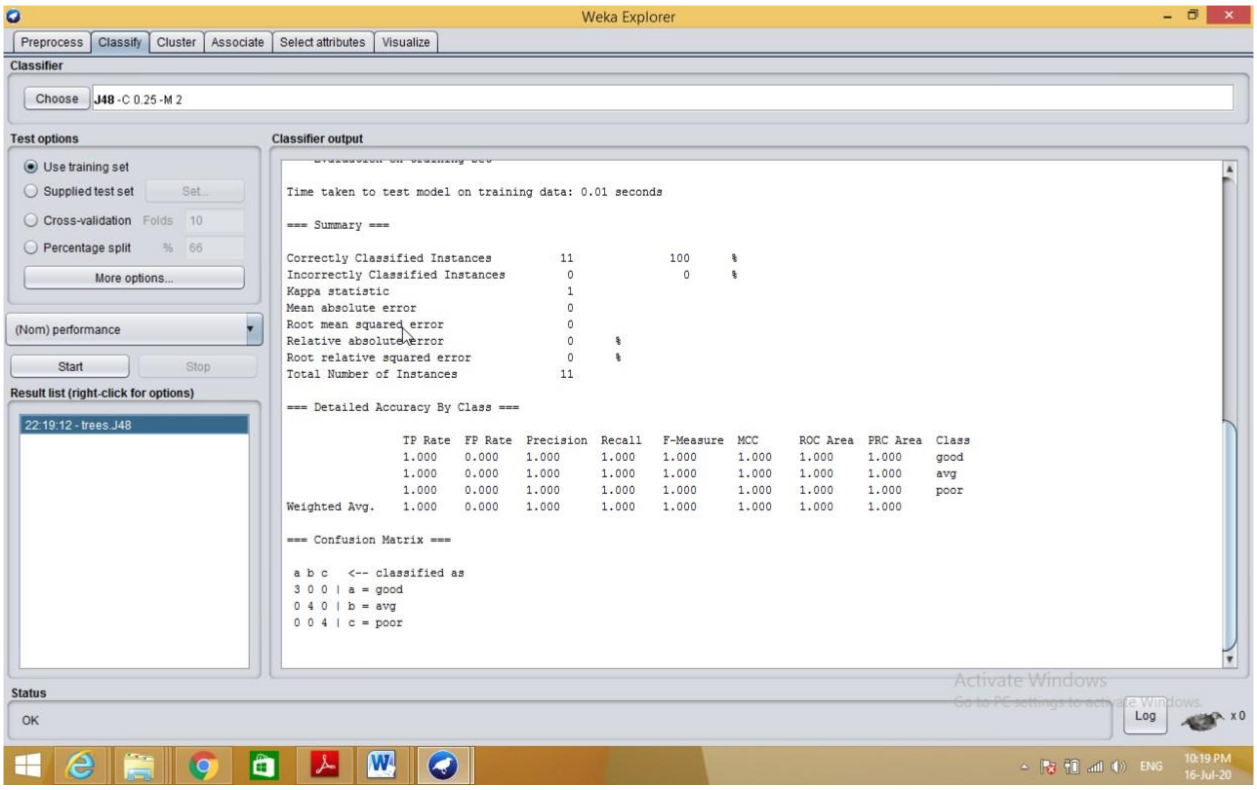
Step 1: Load appropriate dataset into WEKA.

Step 2: Select classify TAB and select J48 algorithm.

**Data set employee.arff:**

```
@relation employee
@attribute age {25, 27, 28, 29, 30, 35, 48}
@attribute salary{10k,15k,17k,20k,25k,30k,35k,32k}
@attribute performance {good, avg, poor}
@data
%
25, 10k, poor
27, 15k, poor
27, 17k, poor
28, 17k, poor
29, 20k, avg
30, 25k, avg
29, 25k, avg
30, 20k, avg
35, 32k, good
```

48, 35k, good
48, 32k,good
%



Classifier: J48 -C 0.25 -M 2

Test options: [✓ Use training set] Cross-validation Folds 10; Percentage split % 66

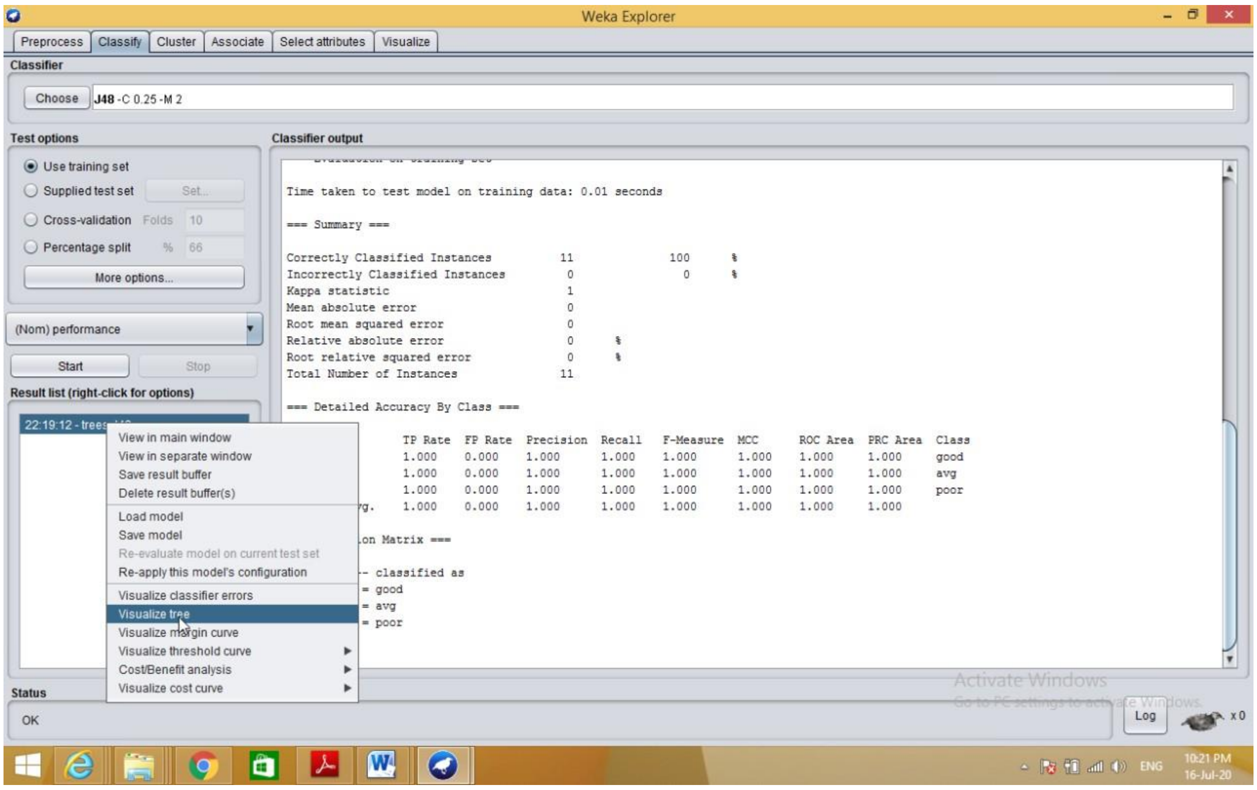
(Nom) performance

Result list: 22:19:12 - trees.J48

```
Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances          11              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances               11

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     good
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     avg
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     poor
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

 a b c   <-- classified as
 3 0 0 | a = good
 0 4 0 | b = avg
 0 0 4 | c = poor
```

## Generating Decision Tree:-

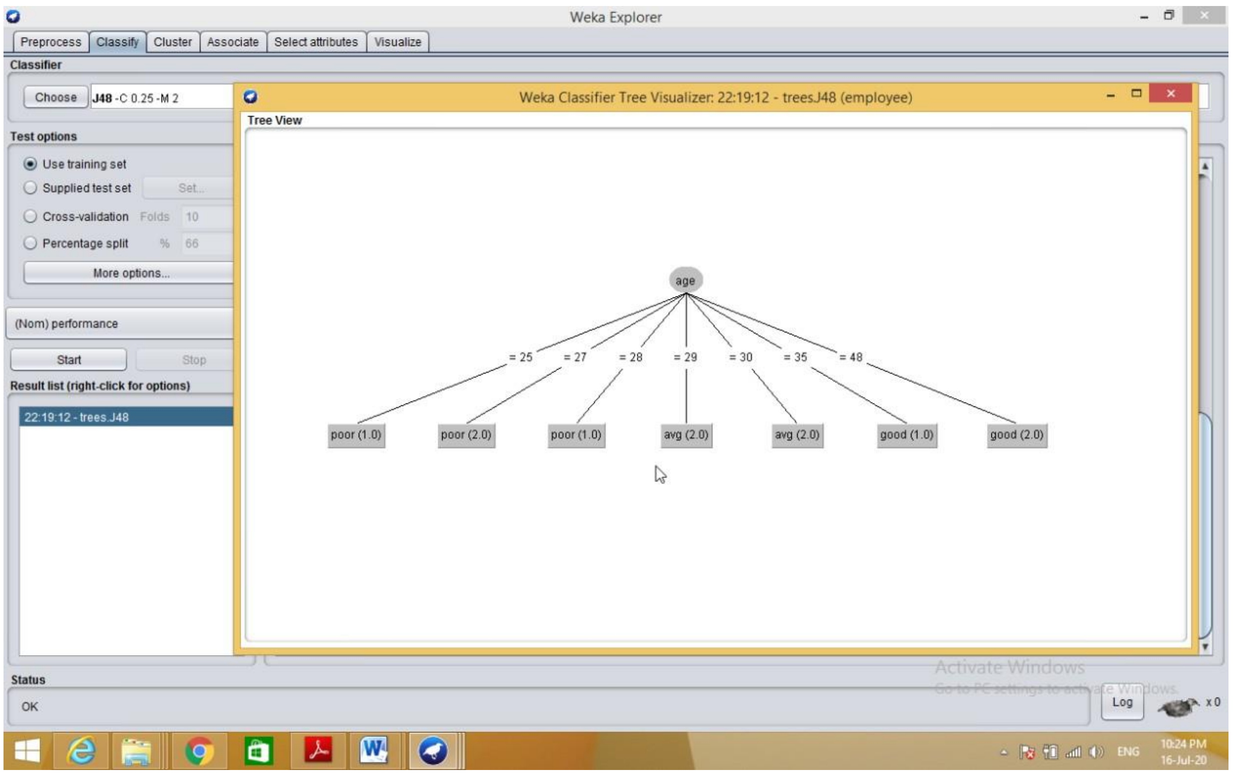
Right click on result list→select visualize tree option



J48 -C 0.25 -M 2

Classifier output

```
Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances          11              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances               11

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     good
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     avg
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     poor
             vg. 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

    on Matrix ===

    -- classified as
     = good
     = avg
     = poor
```



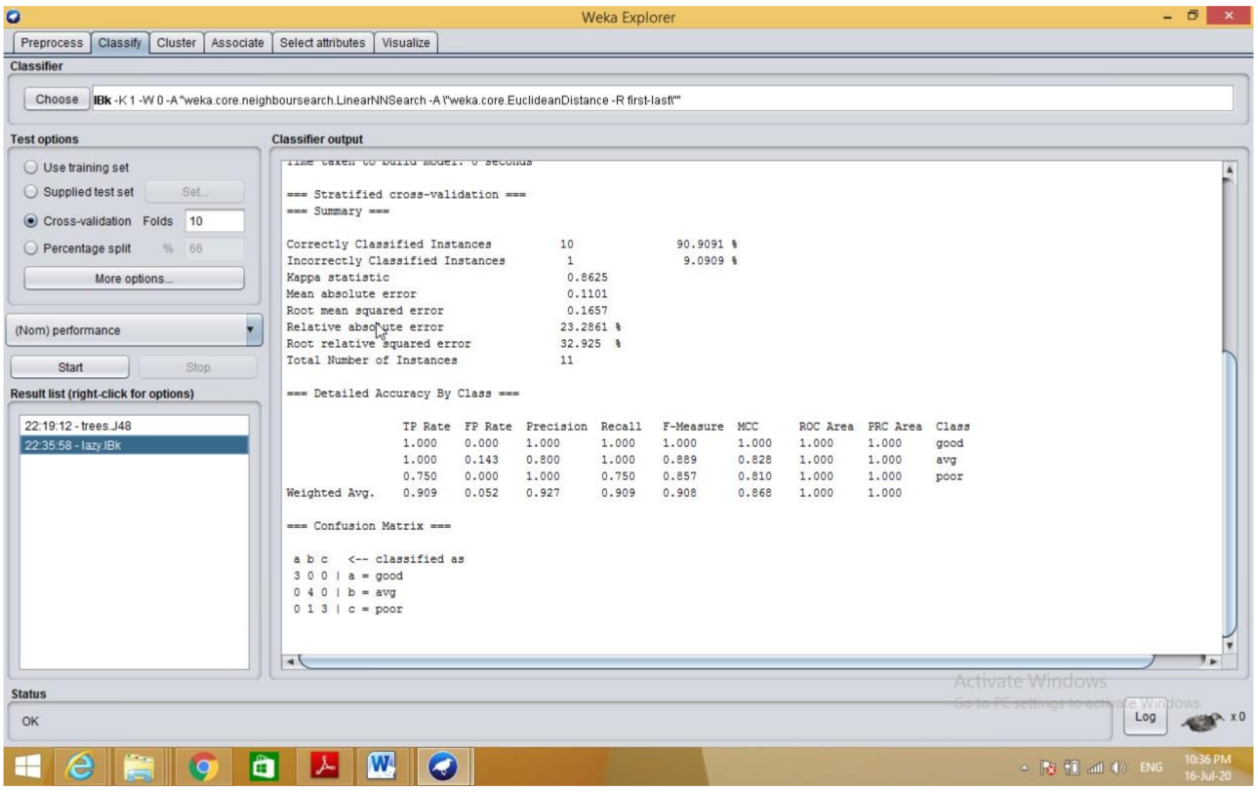
## 4. Demonstration of classification rule process on dataset employee.arff using id3 algorithm.

**Aim:** This experiment illustrates the use of j-48 classifier in weka.the sample data set used in this experiment is “employee”data available at arff format. This document assumes that appropriate data pre processing has been performed.
Steps involved in this experiment:
Step 1: We begin the experiment by loading the data (employee.arff) into weka.
Step2: Next we select the “classify” tab and click “choose” button to select the “j48”classifier.
Step3: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values the default version does perform some pruning but does not perform error pruning.
Step4: Under the “text “options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don’t have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.
Step-5: We now click ”start” to generate the model .the ASCII version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.
Step-6: Note that the classification accuracy of model is about 69%.this indicates that we may find more work. (Either in preprocessing or in selecting current parameters for the classification)
Step-7: Now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting “visualize tree” from the pop-up menu.
Step-8: We will use our model to classify the new instances.
Step-9: In the main panel under “text “options click the “supplied test set” radio button and then click the “set” button. This wills pop-up a window which will allow you to open the file containing test instances.

**Description: Classification** is a **data mining** function that assigns items in a collection to target categories or classes. The goal of **classification** is to accurately predict the target class for each case in the **data**. For example, a **classification** model could be used to identify loan applicants as low, medium, or high credit risks.

**Data set employee.arff:**
@relation employee

```
@attribute age {25, 27, 28, 29, 30, 35, 48}
@attribute salary{10k,15k,17k,20k,25k,30k,35k,32k}
@attribute performance {good, avg, poor}
@data
%
25, 10k, poor
27, 15k, poor
27, 17k, poor
28, 17k, poor
29, 20k, avg
30, 25k, avg
29, 25k, avg
30, 20k, avg
35, 32k, good
48, 35k, good
48, 32k,good
%
```



IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""

Classifier output

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          10               90.9091 %
Incorrectly Classified Instances         1                9.0909 %
Kappa statistic                          0.8625
Mean absolute error                      0.1101
Root mean squared error                  0.1657
Relative absolute error                 23.2861 %
Root relative squared error             32.925  %
Total Number of Instances               11

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     good
                 1.000    0.143    0.800      1.000    0.889      0.828    1.000     1.000     avg
                 0.750    0.000    1.000      0.750    0.857      0.810    1.000     1.000     poor
Weighted Avg.    0.909    0.052    0.927      0.909    0.908      0.868    1.000     1.000

=== Confusion Matrix ===

 a b c   <-- classified as
 3 0 0 | a = good
 0 4 0 | b = avg
 0 1 3 | c = poor
```

## 5. Demonstration of classification rule process on dataset employee.arff using naïve bayes algorithm.

**Aim:** This experiment illustrates the use of naïve bayes classifier in weka. The sample data set used in this experiment is “employee”data available at arff format. This document assumes that appropriate data pre processing has been performed.
Steps involved in this experiment:
1. We begin the experiment by loading the data (employee.arff) into weka.
Step2: next we select the “classify” tab and click “choose” button to select the “id3”classifier.
Step3: now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values his default version does perform some pruning but does not perform error pruning.
Step4: under the “text “options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don’t have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.
Step-5: we now click”start”to generate the model .the ASCII version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.
Step-6: note that the classification accuracy of model is about 69%.this indicates that we may find more work. (Either in preprocessing or in selecting current parameters for the classification)
Step-7: now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting “visualize tree” from the pop-up menu.
Step-8: we will use our model to classify the new instances.
Step-9: In the main panel under “text “options click the “supplied test set” radio button and then click the “set” button. This will show pop-up window which will allow you to open the file containing test instances.

**Description:**
**Bayesian classification** is based on Baye’s Theorem. Bayesian classifiers are the statistical classifiers. Bayesian classifiers can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.

**Baye's Theorem**
Bayes' Theorem is named after Thomas Bayes. There are two types of probabilities

- Posterior Probability [P(H/X)]
- Prior Probability [P(H)]

where X is data tuple and H is some hypothesis.
According to Bayes' Theorem,

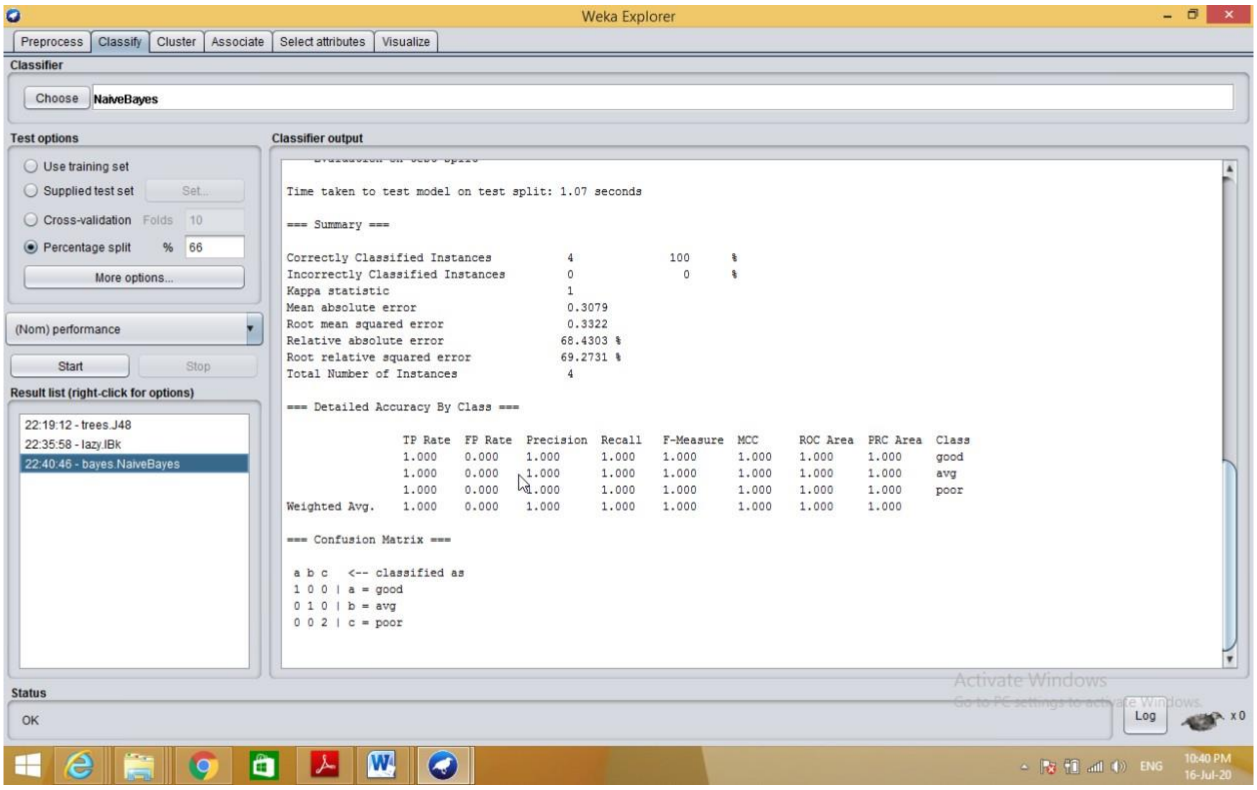
$P(H/X)= P(X/H)P(H) / P(X)$

**Procedure:**

Step 1: Load appropriate dataset into WEKA
Step 2: Go to Classify tab→Bayes→NaiveBayes

Data set : **classification rule process on dataset employee.arff using naïve bayes algorithm.**

```
@relation employee

@attribute age {25,27,28,29,30,35,48,39,45}

@attribute salary {10k,15k,17k,20k,25k,30k,34k,35k,32k,33k,40k}

@attribute performance {good,avg,poor}

@data

%

25,10k,poor

27,15k,poor

27,17k,poor

28,17k,poor

29,20k,avg

30,25k,avg

29,20k,avg

30,20k,avg

35,32k,good

39,33k,good

45,40k,good

48,34k,good

48,32k,good

35,20k,poor

%
```



Classifier: NaiveBayes

Test options: Use training set / Supplied test set / Cross-validation Folds 10 / [✓ Percentage split] % 66

(Nom) performance

Result list (right-click for options)
22:19:12 - trees.J48
22:35:58 - lazy.IBk
22:40:46 - bayes.NaiveBayes

Classifier output

```
Time taken to test model on test split: 1.07 seconds

=== Summary ===

Correctly Classified Instances           4              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.3079
Root mean squared error                  0.3322
Relative absolute error                 68.4303 %
Root relative squared error             69.2731 %
Total Number of Instances                4

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     good
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     avg
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     poor
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

 a b c   <-- classified as
 1 0 0 | a = good
 0 1 0 | b = avg
 0 0 2 | c = poor
```

Status: OK

## 6. Demonstration of clustering rule process on dataset iris.arff using simple k-means.

**Aim:** This experiment illustrates the use of simple k-mean clustering with Weka explorer. The sample data set used for this example is based on the iris data available in ARFF format. This document assumes that appropriate preprocessing has been performed. This iris dataset includes 150 instances.

Steps involved in this Experiment

Step 1: Run the Weka explorer and load the data file iris.arff in preprocessing interface.

Step 2: Inorder to perform clustering select the 'cluster' tab in the explorer and click on the choose button. This step results in a dropdown list of available clustering algorithms.

Step 3 : In this case we select 'simple k-means'.

Step 4: Next click in text button to the right of the choose button to get popup window shown in the screenshots. In this window we enter six on the number of clusters and we leave the value of the seed on as it is. The seed value is used in generating a random number which is used for making the internal assignments of instances of clusters.

Step 5 : Once of the option have been specified. We run the clustering algorithm there we must make sure that they are in the 'cluster mode' panel. The use of training set option is selected and then we click 'start' button. This process and resulting window are shown in the following screenshots.

Step 6 : The result window shows the centroid of each cluster as well as statistics on the number and the percent of instances assigned to different clusters. Here clusters centroid are means vectors for each clusters. This clusters can be used to characterized the cluster.For eg, the centroid of cluster1 shows the class iris.versicolor mean value of the sepal length is 5.4706, sepal width 2.4765, petal width 1.1294, petal length 3.7941.

Step 7: Another way of understanding characterstics of each cluster through visualization ,we can do this, try right clicking the result set on the result. List panel and selecting the visualize cluster assignments.

**Description:**

**Clustering** is a process of partitioning a set of **data** (or objects) into a set of meaningful sub-classes, called **clusters**.

**K-Means clustering** intends to partition *n* objects into *k* clusters in which each object belongs to the cluster with the nearest mean.

**Procedure:**

Step 1: Load appropriate dataset into WEKA

Step 2: Go to Cluster tab→choose→SimpleKmeans

Dataset :

@relation iris @attribute sepallength REAL @attribute sepalwidth REAL @attribute petallength REAL @attribute petalwidth REAL @attribute class {Iris-setosa,Iris-versicolor,Iris-virginica} @data 5.1,3.5,1.4,0.2,Iris-setosa 4.9,3.0,1.4,0.2,Iris-setosa 4.7,3.2,1.3,0.2,Iris-setosa 4.6,3.1,1.5,0.2,Iris-setosa 5.0,3.6,1.4,0.2,Iris-setosa

| | | |
|---|---|---|
| 5.4,3.9,1.7,0.4,Iris-setosa | 4.6,3.4,1.4,0.3,Iris-setosa | 5.0,3.4,1.5,0.2,Iris-setosa |
| 4.4,2.9,1.4,0.2,Iris-setosa | 4.9,3.1,1.5,0.1,Iris-setosa | 5.4,3.7,1.5,0.2,Iris-setosa |
| 4.8,3.4,1.6,0.2,Iris-setosa | 4.8,3.0,1.4,0.1,Iris-setosa | 4.3,3.0,1.1,0.1,Iris-setosa |
| 5.8,4.0,1.2,0.2,Iris-setosa | 5.7,4.4,1.5,0.4,Iris-setosa | 5.4,3.9,1.3,0.4,Iris-setosa |
| 5.1,3.5,1.4,0.3,Iris-setosa | 5.7,3.8,1.7,0.3,Iris-setosa | 5.1,3.8,1.5,0.3,Iris-setosa |
| 5.4,3.4,1.7,0.2,Iris-setosa | 5.1,3.7,1.5,0.4,Iris-setosa | 4.6,3.6,1.0,0.2,Iris-setosa |
| 5.1,3.3,1.7,0.5,Iris-setosa | 4.8,3.4,1.9,0.2,Iris-setosa | 5.0,3.0,1.6,0.2,Iris-setosa |
| 5.0,3.4,1.6,0.4,Iris-setosa | 5.2,3.5,1.5,0.2,Iris-setosa | 5.2,3.4,1.4,0.2,Iris-setosa |
| 4.7,3.2,1.6,0.2,Iris-setosa | 4.8,3.1,1.6,0.2,Iris-setosa | 5.4,3.4,1.5,0.4,Iris-setosa |
| 5.2,4.1,1.5,0.1,Iris-setosa | 5.5,4.2,1.4,0.2,Iris-setosa | 4.9,3.1,1.5,0.1,Iris-setosa |
| 5.0,3.2,1.2,0.2,Iris-setosa | 5.5,3.5,1.3,0.2,Iris-setosa | 4.9,3.1,1.5,0.1,Iris-setosa |
| 4.4,3.0,1.3,0.2,Iris-setosa | 5.1,3.4,1.5,0.2,Iris-setosa | 5.0,3.5,1.3,0.3,Iris-setosa |
| 4.5,2.3,1.3,0.3,Iris-setosa | 4.4,3.2,1.3,0.2,Iris-setosa | 5.0,3.5,1.6,0.6,Iris-setosa |
| 5.1,3.8,1.9,0.4,Iris-setosa | 4.8,3.0,1.4,0.3,Iris-setosa | 5.1,3.8,1.6,0.2,Iris-setosa |
| 4.6,3.2,1.4,0.2,Iris-setosa | 5.3,3.7,1.5,0.2,Iris-setosa | 5.0,3.3,1.4,0.2,Iris-setosa |
| 7.0,3.2,4.7,1.4,Iris-versicolor | 6.4,3.2,4.5,1.5,Iris-versicolor | 6.9,3.1,4.9,1.5,Iris-versicolor |
| 5.5,2.3,4.0,1.3,Iris-versicolor | 6.5,2.8,4.6,1.5,Iris-versicolor | 5.7,2.8,4.5,1.3,Iris-versicolor |
| 6.3,3.3,4.7,1.6,Iris-versicolor | 4.9,2.4,3.3,1.0,Iris-versicolor | 6.6,2.9,4.6,1.3,Iris-versicolor |
| 5.2,2.7,3.9,1.4,Iris-versicolor | 5.0,2.0,3.5,1.0,Iris-versicolor | 5.9,3.0,4.2,1.5,Iris-versicolor |
| 6.0,2.2,4.0,1.0,Iris-versicolor | 6.1,2.9,4.7,1.4,Iris-versicolor | 5.6,2.9,3.6,1.3,Iris-versicolor |
| 6.7,3.1,4.4,1.4,Iris-versicolor | 5.6,3.0,4.5,1.5,Iris-versicolor | 5.8,2.7,4.1,1.0,Iris-versicolor |
| 6.2,2.2,4.5,1.5,Iris-versicolor | 5.6,2.5,3.9,1.1,Iris-versicolor | 5.9,3.2,4.8,1.8,Iris-versicolor |
| 6.1,2.8,4.0,1.3,Iris-versicolor | 6.3,2.5,4.9,1.5,Iris-versicolor | 6.1,2.8,4.7,1.2,Iris-versicolor |
| 6.4,2.9,4.3,1.3,Iris-versicolor | 6.6,3.0,4.4,1.4,Iris-versicolor | 6.8,2.8,4.8,1.4,Iris-versicolor |
| 6.7,3.0,5.0,1.7,Iris-versicolor | 6.0,2.9,4.5,1.5,Iris-versicolor | 5.7,2.6,3.5,1.0,Iris-versicolor |
| 5.5,2.4,3.8,1.1,Iris-versicolor | 5.5,2.4,3.7,1.0,Iris-versicolor | 5.8,2.7,3.9,1.2,Iris-versicolor |
| 6.0,2.7,5.1,1.6,Iris-versicolor | 5.4,3.0,4.5,1.5,Iris-versicolor | 6.0,3.4,4.5,1.6,Iris-versicolor |
| 6.7,3.1,4.7,1.5,Iris-versicolor | 6.3,2.3,4.4,1.3,Iris-versicolor | 5.6,3.0,4.1,1.3,Iris-versicolor |
| 5.5,2.5,4.0,1.3,Iris-versicolor | 5.5,2.6,4.4,1.2,Iris-versicolor | 6.1,3.0,4.6,1.4,Iris-versicolor |
| 5.8,2.6,4.0,1.2,Iris-versicolor | 5.0,2.3,3.3,1.0,Iris-versicolor | 5.6,2.7,4.2,1.3,Iris-versicolor |
| 5.7,3.0,4.2,1.2,Iris-versicolor | 5.7,2.9,4.2,1.3,Iris-versicolor | 6.2,2.9,4.3,1.3,Iris-versicolor |
| 5.1,2.5,3.0,1.1,Iris-versicolor | 5.7,2.8,4.1,1.3,Iris-versicolor | 6.3,3.3,6.0,2.5,Iris-virginica |
| 5.8,2.7,5.1,1.9,Iris-virginica | 7.1,3.0,5.9,2.1,Iris-virginica | 6.3,2.9,5.6,1.8,Iris-virginica |
| 6.5,3.0,5.8,2.2,Iris-virginica | 7.6,3.0,6.6,2.1,Iris-virginica | 4.9,2.5,4.5,1.7,Iris-virginica |
| 7.3,2.9,6.3,1.8,Iris-virginica | 6.7,2.5,5.8,1.8,Iris-virginica | 7.2,3.6,6.1,2.5,Iris-virginica |
| 6.5,3.2,5.1,2.0,Iris-virginica | 6.4,2.7,5.3,1.9,Iris-virginica | 6.8,3.0,5.5,2.1,Iris-virginica |
| 5.7,2.5,5.0,2.0,Iris-virginica | 5.8,2.8,5.1,2.4,Iris-virginica | 6.4,3.2,5.3,2.3,Iris-virginica |
| 6.5,3.0,5.5,1.8,Iris-virginica | 7.7,3.8,6.7,2.2,Iris-virginica | 7.7,2.6,6.9,2.3,Iris-virginica |
| 6.0,2.2,5.0,1.5,Iris-virginica | 6.9,3.2,5.7,2.3,Iris-virginica | 5.6,2.8,4.9,2.0,Iris-virginica |
| 7.7,2.8,6.7,2.0,Iris-virginica | 6.3,2.7,4.9,1.8,Iris-virginica | 6.7,3.3,5.7,2.1,Iris-virginica |
| 7.2,3.2,6.0,1.8,Iris-virginica | 6.2,2.8,4.8,1.8,Iris-virginica | 6.1,3.0,4.9,1.8,Iris-virginica |
| 6.4,2.8,5.6,2.1,Iris-virginica | 7.2,3.0,5.8,1.6,Iris-virginica | 7.4,2.8,6.1,1.9,Iris-virginica |
| 7.9,3.8,6.4,2.0,Iris-virginica | 6.4,2.8,5.6,2.2,Iris-virginica | 6.3,2.8,5.1,1.5,Iris-virginica |

6.1,2.6,5.6,1.4,Iris-virginica 7.7,3.0,6.1,2.3,Iris-virginica 6.3,3.4,5.6,2.4,Iris-virginica

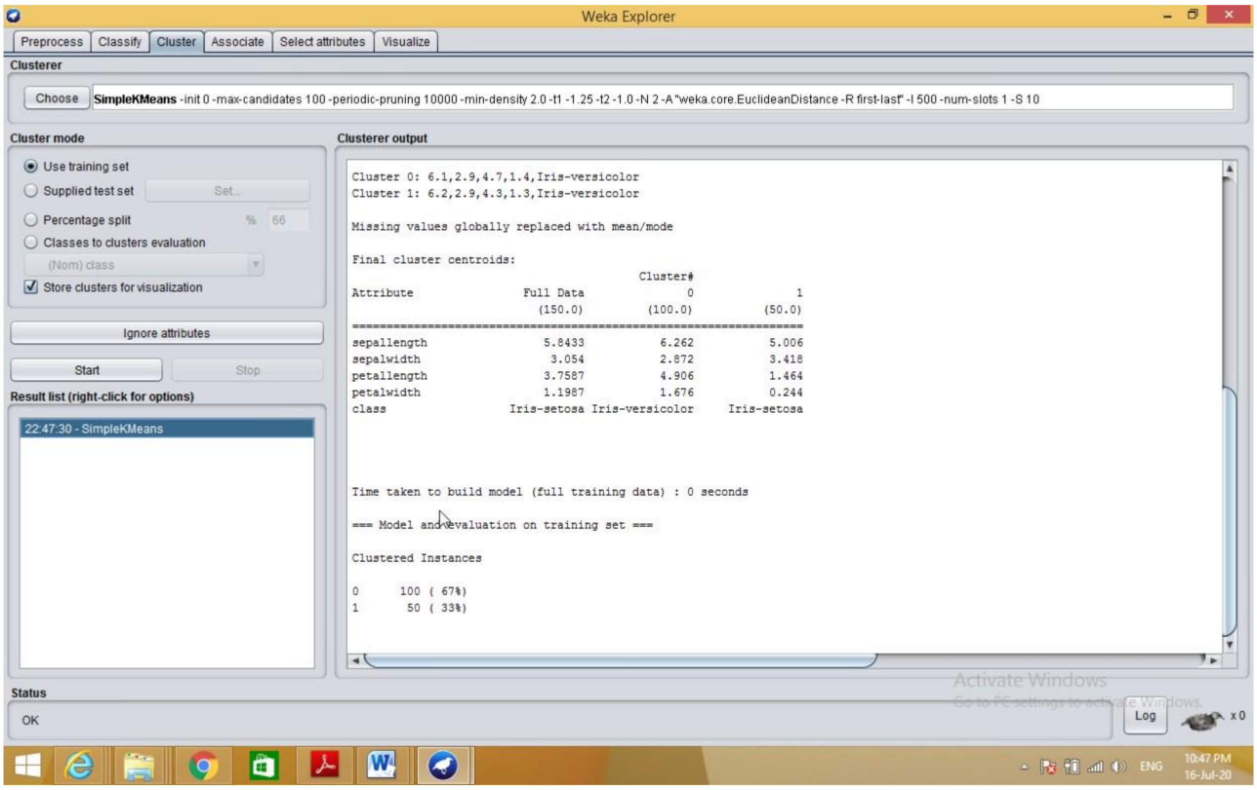
6.4,3.1,5.5,1.8,Iris-virginica 6.0,3.0,4.8,1.8,Iris-virginica 6.9,3.1,5.4,2.1,Iris-virginica
6.7,3.1,5.6,2.4,Iris-virginica 6.9,3.1,5.1,2.3,Iris-virginica 5.8,2.7,5.1,1.9,Iris-virginica
6.8,3.2,5.9,2.3,Iris-virginica 6.7,3.3,5.7,2.5,Iris-virginica 6.7,3.0,5.2,2.3,Iris-virginica
6.3,2.5,5.0,1.9,Iris-virginica 6.5,3.0,5.2,2.0,Iris-virginica 6.2,3.4,5.4,2.3,Iris-virginica

5.9,3.0,5.1,1.8,Iris-virginica

%

%

%



Clusterer

SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10

Cluster mode

(●) Use training set
( ) Supplied test set
( ) Percentage split % 66
( ) Classes to clusters evaluation
(Nom) class
[✓] Store clusters for visualization

Result list (right-click for options)

22:47:30 - SimpleKMeans

Clusterer output

```
Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor

Missing values globally replaced with mean/mode

Final cluster centroids:
                                  Cluster#
Attribute            Full Data           0             1
                       (150.0)     (100.0)        (50.0)
=========================================================
sepallength             5.8433       6.262         5.006
sepalwidth               3.054       2.872         3.418
petallength             3.7587       4.906         1.464
petalwidth              1.1987       1.676         0.244
class              Iris-setosa Iris-versicolor Iris-setosa




Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      100 ( 67%)
1       50 ( 33%)
```

Status

OK

## 7. Demonstration of clustering rule process on dataset student.arff using hierarchical clustering.

**Aim:** This experiment illustrates the use of hierarchical clustering with Weka explorer. The sample data set used for this example is based on the student data available in ARFF format. This document assumes that appropriate preprocessing has been performed. This istudent dataset includes 14 instances.
Steps involved in this Experiment
Step 1: Run the Weka explorer and load the data file student.arff in preprocessing interface.
Step 2: Inorder to perform clustering select the 'cluster' tab in the explorer and click on the choose button. This step results in a dropdown list of available clustering algorithms.
Step 3 : In this case we select 'hierarchical'.
Step 4: Next click in text button to the right of the choose button to get popup window shown in the screenshots. In this window we enter six on the number of clusters and we leave the value of the seed on as it is. The seed value is used in generating a random number which is used for making the internal assignments of instances of clusters.
Step 5 : Once of the option have been specified. We run the clustering algorithm there we must make sure that they are in the 'cluster mode' panel. The use of training set option is selected and then we click 'start' button. This process and resulting window are shown in the following screenshots.
Step 6 : The result window shows the centroid of each cluster as well as statistics on the number and the percent of instances assigned to different clusters. Here clusters centroid are means vectors for each clusters. This clusters can be used to characterized the cluster.
Step 7: Another way of understanding characteristics of each cluster through visualization ,we can do this, try right clicking the result set on the result. List panel and selecting the visualize cluster assignments. Interpretation of the above visualization From the above visualization, we can understand the distribution of age and instance number in each cluster. For instance, for each cluster is dominated by age. In this case by changing the color dimension to other attributes we can see their distribution with in each of the cluster.
Step 8: We can assure that resulting dataset which included each instance along with its assign cluster. To do so we click the save button in the visualization window and save the result student hierarchical .The top portion of this file is shown in the following figure.

**Description:**
**Hierarchical clustering** involves creating **clusters** that have a predetermined ordering from top to bottom. For example, all files and folders on the hard disk are organized in a **hierarchy**. There are two types of **hierarchical clustering**, Divisive and Agglomerative.

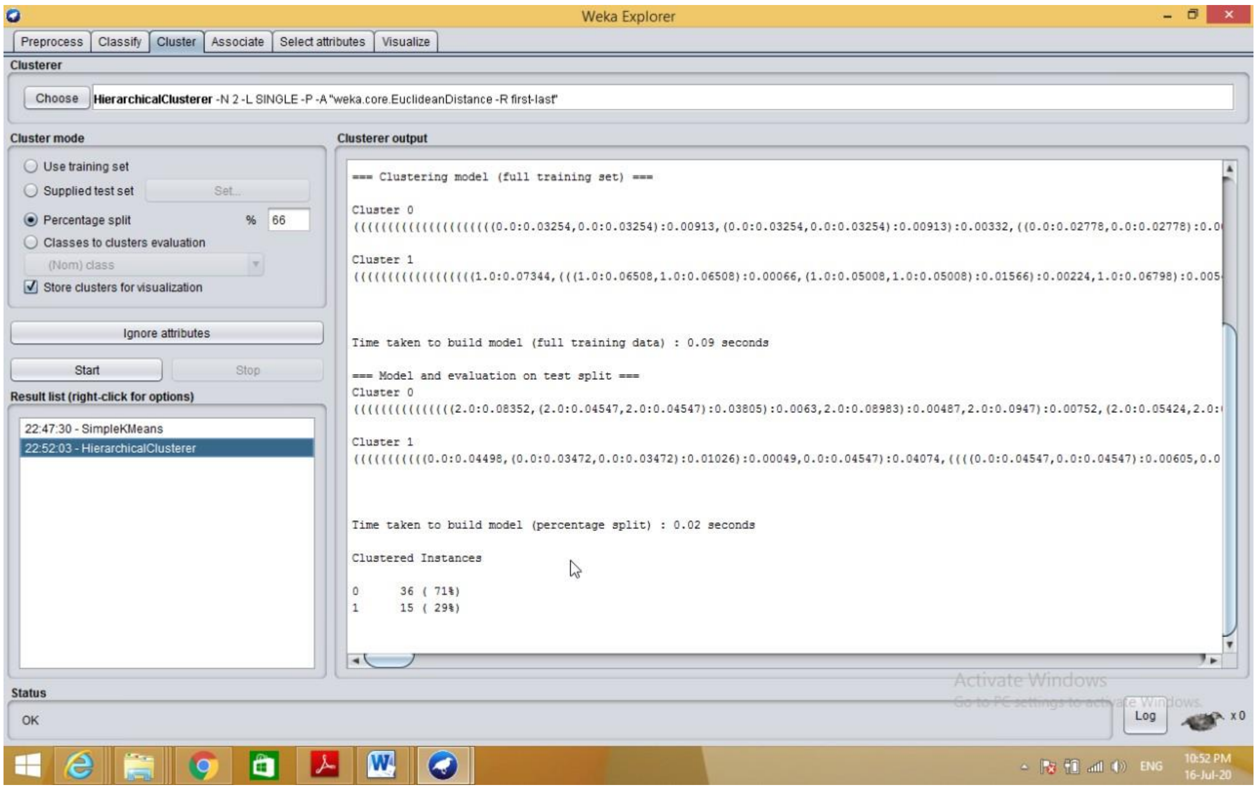
**Procedure:**

Step 1: Load appropriate dataset into WEKA

Step 2: Go to Cluster tab→choose→Hierarchical clustering

**Dataset student .arff**

```
@relation student
@attribute age {<30,30-40,>40}
@attribute income {low, medium, high}
@attribute student {yes, no}
@attribute credit-rating {fair, excellent}
@attribute buyspc {yes, no}
@data
%
<30, high, no, fair, no
<30, high, no, excellent, no
30-40, high, no, fair, yes
>40, medium, no, fair, yes
>40, low, yes, fair, yes
>40, low, yes, excellent, no
30-40, low, yes, excellent, yes
<30, medium, no, fair, no
<30, low, yes, fair, no
>40, medium, yes, fair, yes
<30, medium, yes, excellent, yes
30-40, medium, no, excellent, yes
30-40, high, yes, fair, yes
>40, medium, no, excellent, no
%
```



HierarchicalClusterer -N 2 -L SINGLE -P -A "weka.core.EuclideanDistance -R first-last"

Percentage split % 66

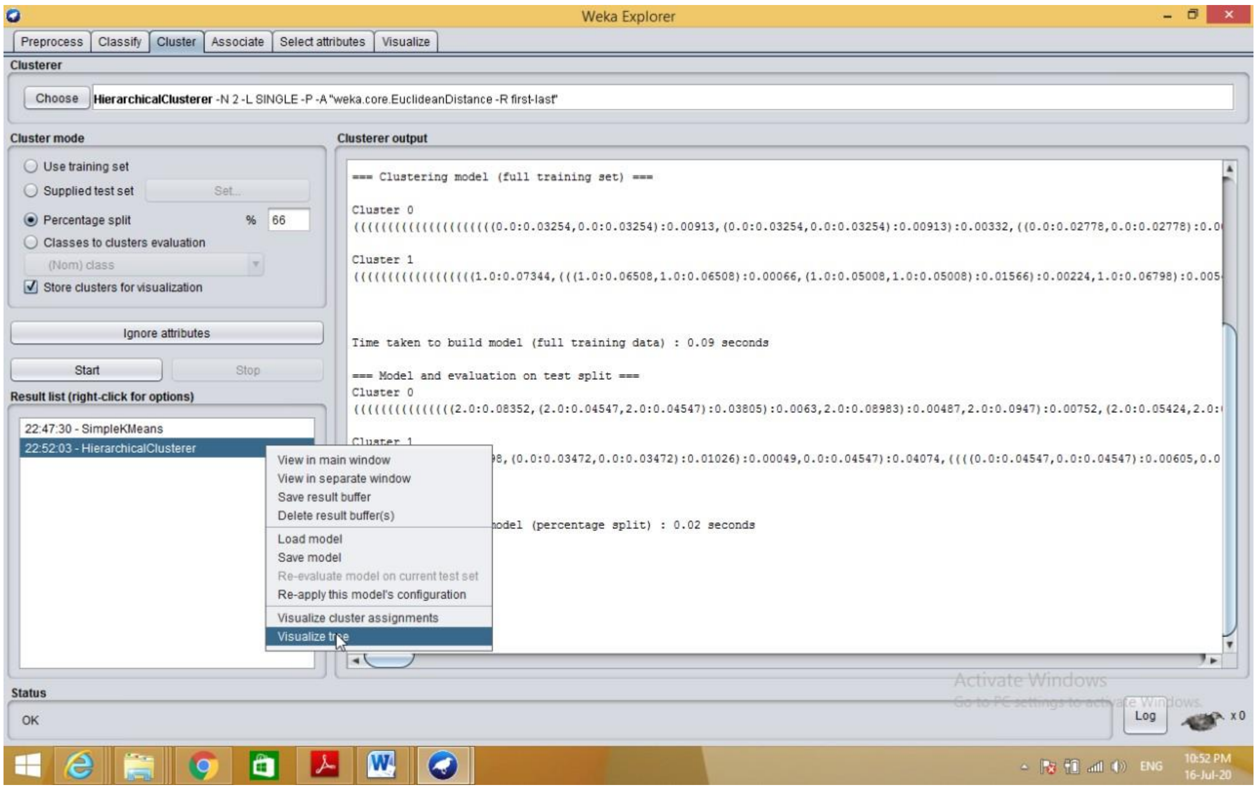
☑ Store clusters for visualization

22:47:30 - SimpleKMeans
22:52:03 - HierarchicalClusterer

```
=== Clustering model (full training set) ===

Cluster 0
(((((((((((((((((((0.0:0.03254,0.0:0.03254):0.00913,(0.0:0.03254,0.0:0.03254):0.00913):0.00332,((0.0:0.02778,0.0:0.02778):0.0

Cluster 1
((((((((((((((((1.0:0.07344,(((1.0:0.06508,1.0:0.06508):0.00066,(1.0:0.05008,1.0:0.05008):0.01566):0.00224,1.0:0.06798):0.005



Time taken to build model (full training data) : 0.09 seconds

=== Model and evaluation on test split ===
Cluster 0
(((((((((((((2.0:0.08352,(2.0:0.04547,2.0:0.04547):0.03805):0.0063,2.0:0.08983):0.00487,2.0:0.0947):0.00752,(2.0:0.05424,2.0:

Cluster 1
((((((((((0.0:0.04498,(0.0:0.03472,0.0:0.03472):0.01026):0.00049,0.0:0.04547):0.04074,((((0.0:0.04547,0.0:0.04547):0.00605,0.0



Time taken to build model (percentage split) : 0.02 seconds

Clustered Instances

0      36 ( 71%)
1      15 ( 29%)
```

OK

ENG 10:52 PM 16-Jul-20



Clusterer: HierarchicalClusterer -N 2 -L SINGLE -P -A "weka.core.EuclideanDistance -R first-last"

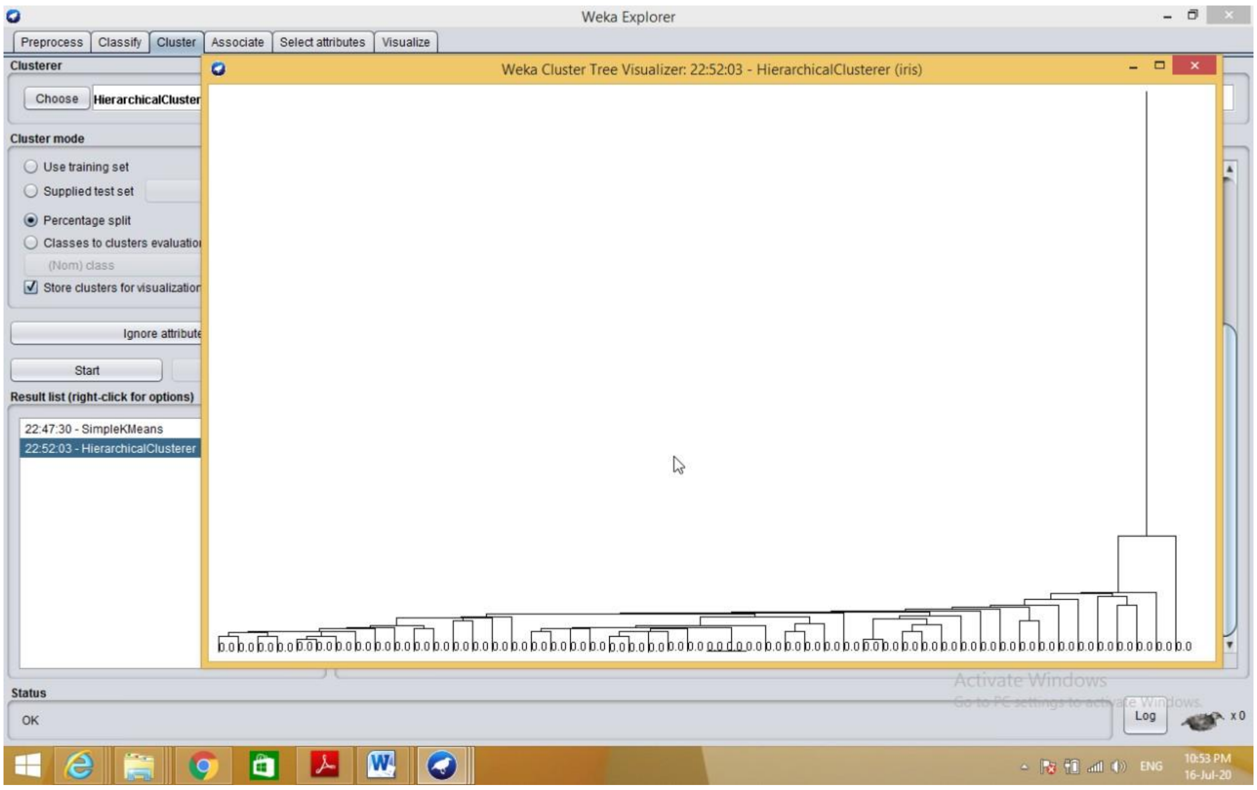
Cluster mode:
- Use training set
- Supplied test set
- [✓ Percentage split] % 66
- Classes to clusters evaluation
- (Nom) class
- [✓] Store clusters for visualization

Result list (right-click for options)
- 22:47:30 - SimpleKMeans
- 22:52:03 - HierarchicalClusterer

Clusterer output

```
=== Clustering model (full training set) ===

Cluster 0
(((((((((((((((((((0.0:0.03254,0.0:0.03254):0.00913,(0.0:0.03254,0.0:0.03254):0.00913):0.00332,((0.0:0.02778,0.0:0.02778):0.0

Cluster 1
((((((((((((((((((1.0:0.07344,(((1.0:0.06508,1.0:0.06508):0.00066,(1.0:0.05008,1.0:0.05008):0.01566):0.00224,1.0:0.06798):0.005



Time taken to build model (full training data) : 0.09 seconds

=== Model and evaluation on test split ===
Cluster 0
((((((((((((((2.0:0.08352,(2.0:0.04547,2.0:0.04547):0.03805):0.0063,2.0:0.08983):0.00487,2.0:0.0947):0.00752,(2.0:0.05424,2.0:

Cluster 1
                     98,(0.0:0.03472,0.0:0.03472):0.01026):0.00049,0.0:0.04547):0.04074,((((0.0:0.04547,0.0:0.04547):0.00605,0.0



                     odel (percentage split) : 0.02 seconds
```

Status
OK


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
HierarchicalCluster
Cluster mode
Use training set
Supplied test set
Percentage split
Classes to clusters evaluation
(Nom) class
Store clusters for visualization
Ignore attribute
Start
Result list (right-click for options)
22:47:30 - SimpleKMeans
22:52:03 - HierarchicalClusterer
Weka Cluster Tree Visualizer: 22:52:03 - HierarchicalClusterer (iris)
Status
OK
Log
x 0

## Part-B: Data Analytics

**1. a) Write R program to find R-Mean, Median & Mode with the sample data.**

**Source code:**

```
Mean:
# Create a vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find Mean.
result.mean <- mean(x)
print(result.mean)
0utput: 8.22

Applying Trim Option:
# Create a vector.
x &lt;- c(12,7,3,4.2,18,2,54,-21,8,-5)
# Find Mean.
result.mean &lt;- mean(x,trim = 0.3)
print(result.mean)
output:5.55
Applying NA Option:
```

```
# Create a vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5,NA)

# Find mean.
result.mean <-  mean(x)
print(result.mean)

# Find mean dropping NA values.
result.mean <-  mean(x,na.rm = TRUE)
print(result.mean)
```

output: NA  8.22

```
# Create the vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find the median.
median.result <- median(x)
print(median.result)
```

**output:** 5.6

```
# Create the function.
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

# Create the vector with numbers.
```

```
v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)

# Calculate the mode using the user function.
result <- getmode(v)
print(result)

# Create the vector with characters.
charv <- c("o","it","the","it","it")

# Calculate the mode using the user function.
result <- getmode(charv)
print(result)
```

**output:2 "it"**

**b) Write R program to find Analysis and Covariance with the sample data and visualize the regression graphically.**

**Source code:**

```
input <- mtcars[,c("am","mpg","hp")]
print(head(input))
# Get the dataset.
input <- mtcars
# Create the regression model.
result <- aov(mpg~hp*am,data = input)
print(summary(result))
# Create the regression model.
result <- aov(mpg~hp+am,data = input)
print(summary(result))
# Create the regression models.
result1 <- aov(mpg~hp*am,data = input)
result2 <- aov(mpg~hp+am,data = input)
# Compare the two models.
print(anova(result1,result2))
```

**output:**

```
result <- aov(mpg~hp*am,data = input)
> print(summary(result))
            Df Sum Sq Mean Sq F value   Pr(>F)
hp           1  678.4   678.4  77.391 1.50e-09 ***
am           1  202.2   202.2  23.072 4.75e-05 ***
hp:am        1    0.0     0.0   0.001    0.981
Residuals   28  245.4     8.8
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
result <- aov(mpg~hp+am,data = input)
> print(summary(result))
            Df Sum Sq Mean Sq F value  Pr(>F)
hp          1  678.4   678.4   80.15 7.63e-10 ***
am          1  202.2   202.2   23.89 3.46e-05 ***
Residuals   29  245.4     8.5
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
result1 <- aov(mpg~hp*am,data = input)
> result2 <- aov(mpg~hp+am,data = input)
> # Compare the two models.
> print(anova(result1,result2))
Analysis of Variance Table

Model 1: mpg ~ hp * am
Model 2: mpg ~ hp + am
  Res.Df    RSS Df  Sum of Sq      F Pr(>F)
1     28 245.43
2     29 245.44 -1 -0.0052515 6e-04 0.9806


>
```

**2. Write R program to find the following Regressions with the sample data and visualize the regressions graphically.**
**Source code:**

**a) Linear Regression**

Creating Relationship Model and Getting the Coefficients

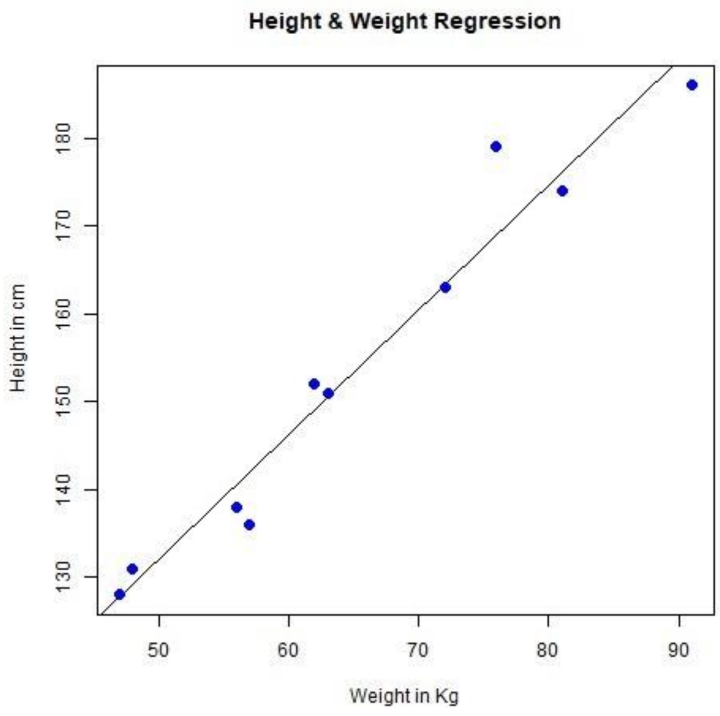
```
#Creating input vector for lm() function
x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)
# Applying the lm() function.
relationship_model<- lm(y~x)
#Printing the coefficient
print(relationship_model)
```

Getting Summary of Relationship Model

```
#Creating input vector for lm() function
x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)
  # Applying the lm() function.
relationship_model<- lm(y~x)
```

```
  #Printing the coefficient
 print(summary(relationship_model))
```

```
x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)
relationship_model<- lm(y~x)
# Giving a name to the chart file.
png(file = "linear_regression.png")
# Plotting the chart.
plot(y,x,col = "red",main = "Height and Weight Regression",abline(lm(x~y)),cex = 1.3,pch = 16,
xlab = "Weight in Kg",ylab = "Height in cm")
# Saving the file.
    dev.off()
```



**b) Multiple Regression**

**Source code:**

```
#Creating input data.
input <- mtcars[,c("mpg","wt","disp","hp")]
# Creating the relationship model.
Model <- lm(mpg~wt+disp+hp, data = input)
# Showing the Model.
print(Model)
```

**output:**

```
data<-mtcars[,c("mpg","wt","disp","hp")]> print(head(input))                mpg    wt disp hp
Mazda RX4         21.0 2.620 160 110
Mazda RX4 Wag     21.0 2.875 160 110
Datsun 710        22.8 2.320 108  93
Hornet 4 Drive    21.4 3.215  258 110
Hornet Sportabout 18.7 3.440  360 175
Valiant           18.1 3.460 225 105
```

```
Call:
lm(formula = mpg ~ wt + disp + hp, data = input)

Coefficients:
(Intercept)          wt         disp           hp
37.105505      -3.800891    -0.000937    -0.031157
```

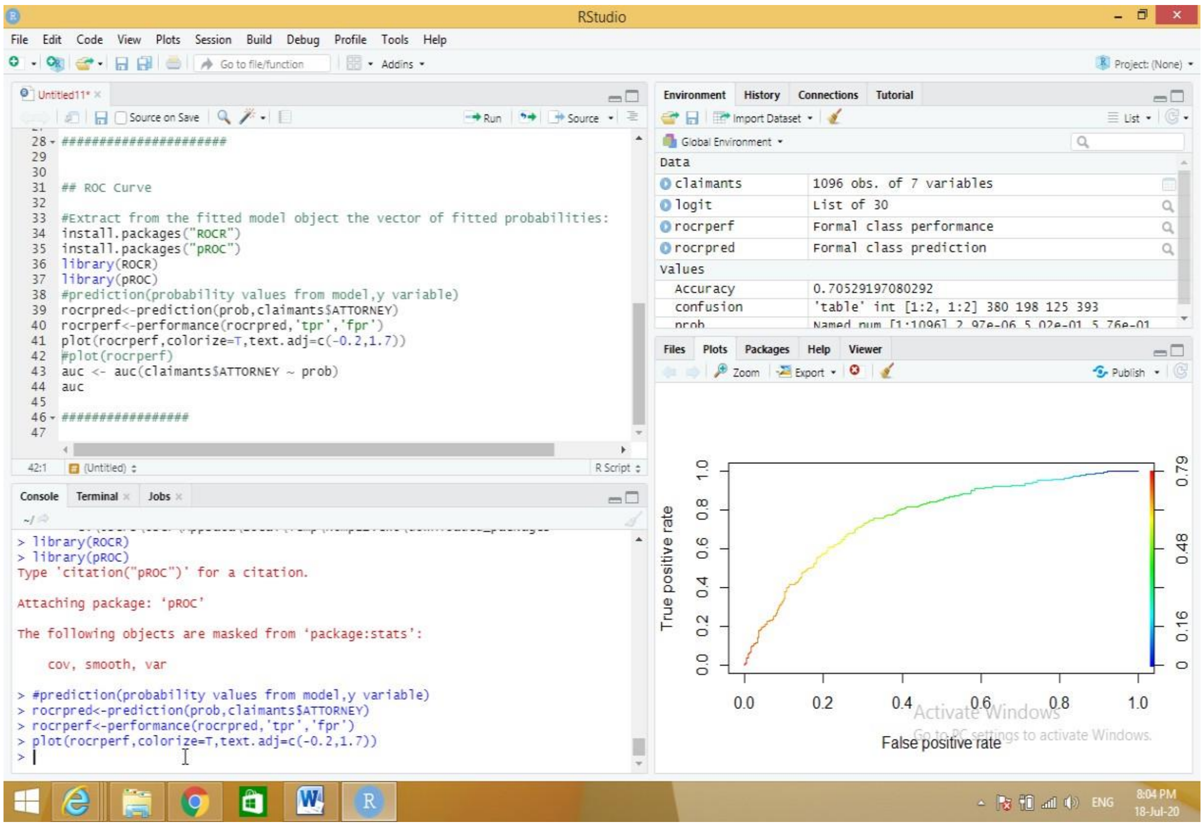
**c) Logistic Regression**

**Source code:**

```
claimants<-read.csv("C:/Users/User/Desktop/EXCELR/logistic regression/claimants.csv")
#Finding null values
sum(is.na(claimants))
#Removing null values- na.omit(dataset)
claimants <- na.omit(claimants)
# Logistic Regression
#glm(y~x,family="bin...)
logit<-glm(ATTORNEY ~ factor(CLMSEX) + factor(CLMINSUR) + factor(SEATBELT)
        + CLMAGE + LOSS,family= "binomial",data=claimants)
summary(logit)

# Confusion Matrix Table
#predict(modelobject,testdataset)
prob=predict(logit,type=c("response"),claimants)
prob

#table(dataframe1,dataframe2) ..to create 2X2 matrix
confusion<-table(prob>0.5,claimants$ATTORNEY)
confusion

# Model Accuracy
#adding diagonal elements in the confusion matrix
Accuracy<-sum(diag(confusion))/sum(confusion)
Accuracy

##############

######################


## ROC Curve

#Extract from the fitted model object the vector of fitted probabilities:
install.packages("ROCR")
install.packages("pROC")
library(ROCR)
library(pROC)
#prediction(probability values from model,y variable)
rocrpred<-prediction(prob,claimants$ATTORNEY)
rocrperf<-performance(rocrpred,'tpr','fpr')
```

```
plot(rocrperf,colorize=T,text.adj=c(-0.2,1.7))
#plot(rocrperf)
auc <- auc(claimants$ATTORNEY ~ prob)
auc

##################
```

**output:**



```
##########################


## ROC Curve

#Extract from the fitted model object the vector of fitted probabilities:
install.packages("ROCR")
install.packages("pROC")
library(ROCR)
library(pROC)
#prediction(probability values from model,y variable)
rocrpred<-prediction(prob,claimants$ATTORNEY)
rocrperf<-performance(rocrpred,'tpr','fpr')
plot(rocrperf,colorize=T,text.adj=c(-0.2,1.7))
#plot(rocrperf)
auc <- auc(claimants$ATTORNEY ~ prob)
auc

##################
```

```
Data
claimants     1096 obs. of 7 variables
logit         List of 30
rocrperf      Formal class performance
rocrpred      Formal class prediction
Values
  Accuracy    0.70529197080292
  confusion   'table' int [1:2, 1:2] 380 198 125 393
  prob        Named num [1:1096] 2.97e-06 5.02e-01 5.76e-01
```

```
> library(ROCR)
> library(pROC)
Type 'citation("pROC")' for a citation.

Attaching package: 'pROC'

The following objects are masked from 'package:stats':

    cov, smooth, var

> #prediction(probability values from model,y variable)
> rocrpred<-prediction(prob,claimants$ATTORNEY)
> rocrperf<-performance(rocrpred,'tpr','fpr')
> plot(rocrperf,colorize=T,text.adj=c(-0.2,1.7))
> 
```

**d) Poisson Regression.**

**Source code:**

```
input <- warpbreaks
print(head(input))

output <-glm(formula = breaks ~ wool+tension, data = warpbreaks,
   family = poisson)
print(summary(output))
```

**output:**

```
input <- warpbreaks
> print(head(input))
breaks wool tension
1 26      A      L
2    30   A      L
3    54   A      L
4    25   A      L
5    70   A      L
6    52   A      L
> output <-glm(formula = breaks ~ wool+tension, data = warpbreaks,
+            family = poisson)
> print(summary(output))

Call:
glm(formula = breaks ~ wool + tension, family = poisson, data = warpbreaks)

Deviance Residuals:
    Min       1Q   Median      3Q      Max
-3.6871  -1.6503  -0.4269  1.1902   4.2616
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.69196    0.04541  81.302  < 2e-16 ***
woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
tensionM    -0.32132    0.06027  -5.332 9.73e-08 ***
tensionH    -0.51849    0.06396  -8.107 5.21e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 297.37 on 53 degrees of freedom
Residual deviance: 210.39 on 50 degrees of freedom
AIC: 493.06

Number of Fisher Scoring iterations: 4
```

**3. a) Write R program to find Time Series Analysis with the sample data and visualize the regression graphically.**

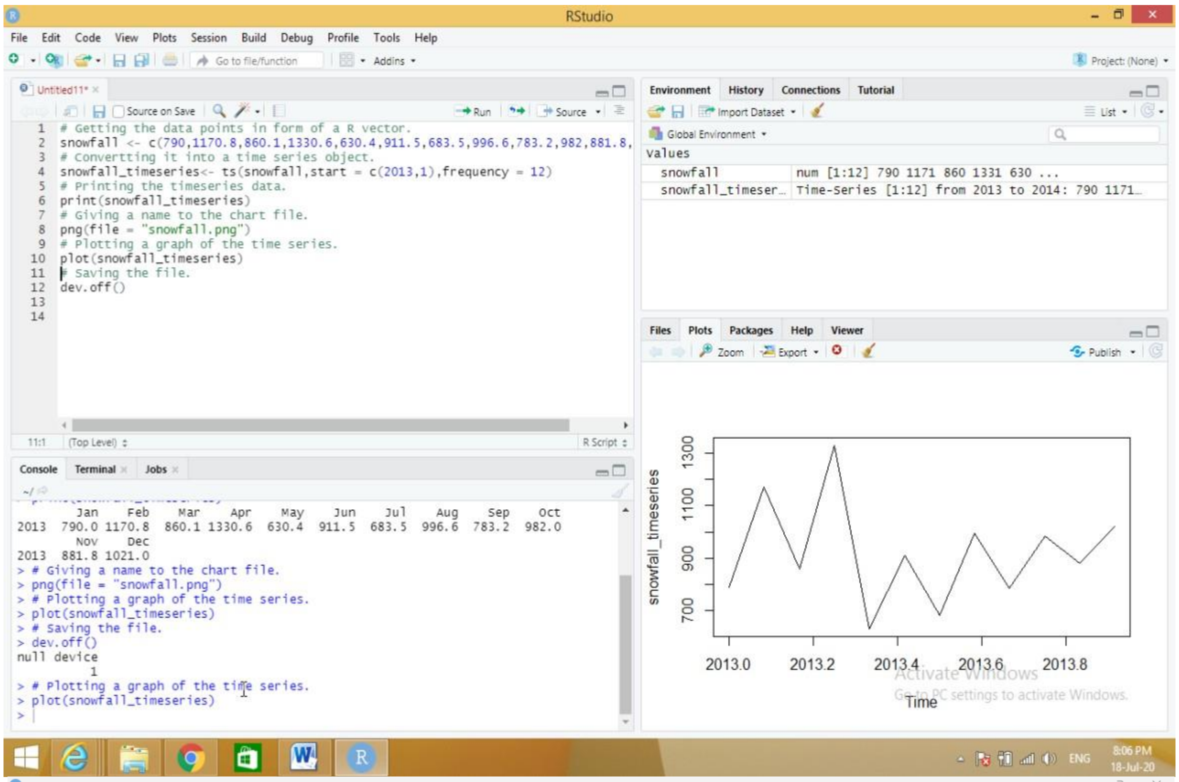
**Source code:**

```
# Getting the data points in form of a R vector.
snowfall <- c(790,1170.8,860.1,1330.6,630.4,911.5,
683.5,996.6,783.2,982,881.8,1021)
# Convertting it into a time series object.
snowfall_timeseries<- ts(snowfall,start = c(2013,1),frequency = 12)
# Printing the timeseries data.
print(snowfall_timeseries)
# Giving a name to the chart file.
png(file = "snowfall.png")
# Plotting a graph of the time series.
plot(snowfall_timeseries)
# Saving the file.
dev.off()
model<-lm(snowfall_timeseries~time(snowfall_timeseries))
png(file = "snowfall_with_trend.png")
# Plotting a graph of the time series.
plot(snowfall_timeseries,main="Snowfall Over Time with Trend Line:",
xlab="Time",ylab="Snowfall",col="blue")
#adding regression
abline(model,col="red")
# Saving the file.
dev.off()
```

**output:**

```
snowfall <- c(790,1170.8,860.1,1330.6,630.4,911.5,683.5,996.6,783.2,982,881.8,1021)
> # Convertting it into a time series object.
> snowfall_timeseries<- ts(snowfall,start = c(2013,1),frequency = 12)
> # Printing the timeseries data.
> print(snowfall_timeseries)
       Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
2013 790.0 1170.8  860.1 1330.6  630.4  911.5  683.5  996.6  783.2  982.0
       Nov    Dec
2013  881.8 1021.0
> # Giving a name to the chart file.
> png(file = "snowfall.png")
> # Plotting a graph of the time series.
> plot(snowfall_timeseries)
> # Saving the file.
> dev.off()
null device
          1
> # Plotting a graph of the time series.
```

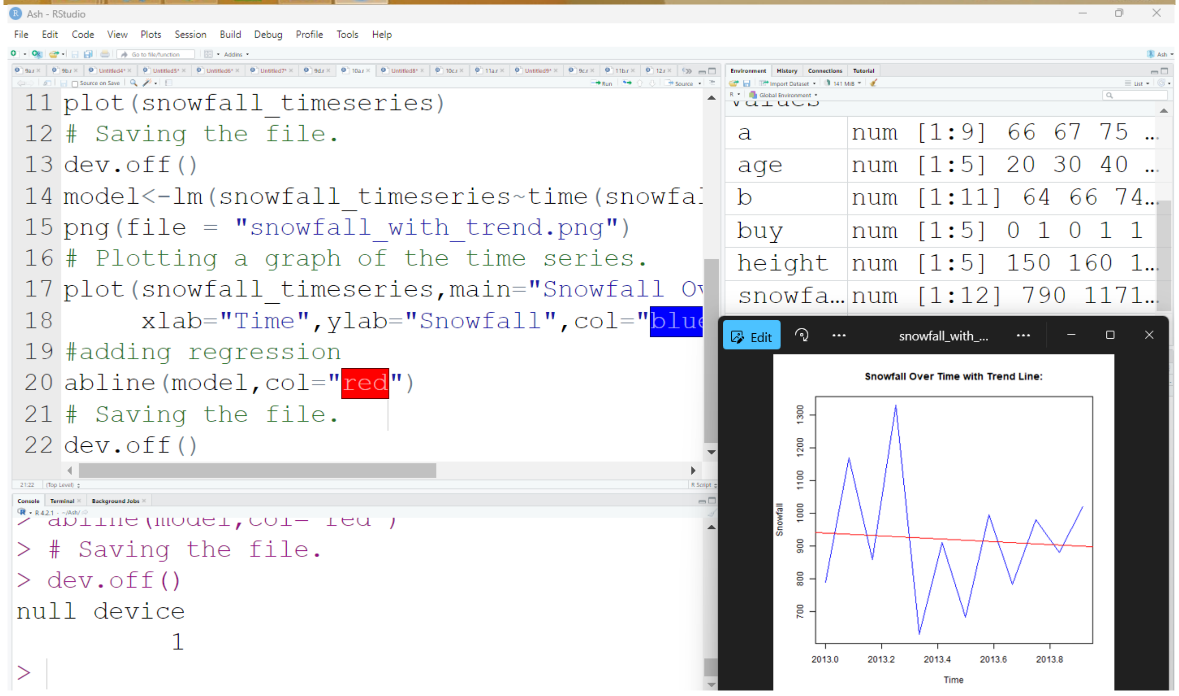
> plot(snowfall_timeseries)



```
# Getting the data points in form of a R vector.
snowfall <- c(790,1170.8,860.1,1330.6,630.4,911.5,683.5,996.6,783.2,982,881.8,
# Convertting it into a time series object.
snowfall_timeseries<- ts(snowfall,start = c(2013,1),frequency = 12)
# Printing the timeseries data.
print(snowfall_timeseries)
# Giving a name to the chart file.
png(file = "snowfall.png")
# Plotting a graph of the time series.
plot(snowfall_timeseries)
# Saving the file.
dev.off()
```

```
        Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
2013  790.0 1170.8  860.1 1330.6  630.4  911.5  683.5  996.6  783.2  982.0
        Nov    Dec
2013  881.8 1021.0
> # Giving a name to the chart file.
> png(file = "snowfall.png")
> # Plotting a graph of the time series.
> plot(snowfall_timeseries)
> # Saving the file.
> dev.off()
null device
          1
> # Plotting a graph of the time series.
> plot(snowfall_timeseries)
>
```

```
values
  snowfall            num [1:12] 790 1171 860 1331 630 ...
  snowfall_timeser…   Time-Series [1:12] from 2013 to 2014: 790 1171…
```



```
plot(snowfall_timeseries)
# Saving the file.
dev.off()
model<-lm(snowfall_timeseries~time(snowfal
png(file = "snowfall_with_trend.png")
# Plotting a graph of the time series.
plot(snowfall_timeseries,main="Snowfall Ov
      xlab="Time",ylab="Snowfall",col="blue
#adding regression
abline(model,col="red")
# Saving the file.
dev.off()
```

```
> # Saving the file.
> dev.off()
null device
          1
>
```

```
a        num [1:9]  66 67 75 …
age      num [1:5]  20 30 40 …
b        num [1:11]  64 66 74…
buy      num [1:5]  0 1 0 1 1
height   num [1:5]  150 160 1…
snowfa…  num [1:12]  790 1171…
```

**b) Write R program to find Non Linear Least Square with the sample data and visualize the regression graphically.**

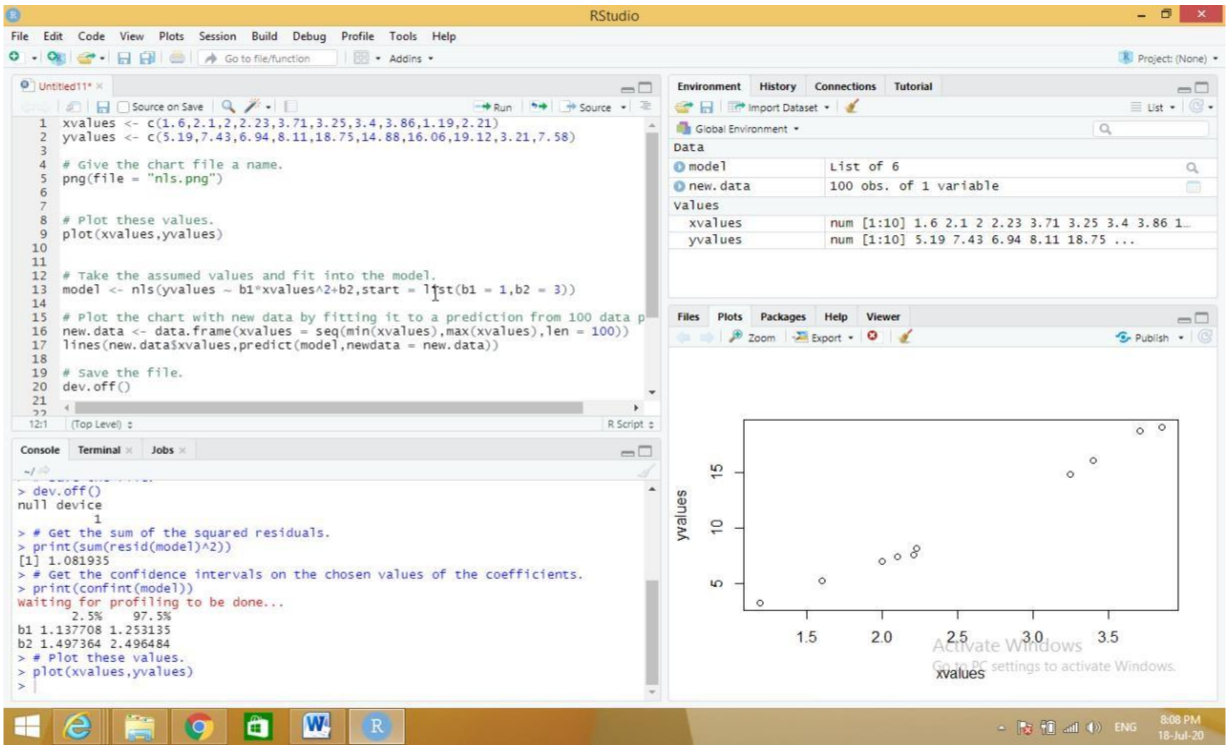
**<u>Source code:</u>**

xvalues <- c(1.6,2.1,2,2.23,3.71,3.25,3.4,3.86,1.19,2.21)

yvalues <- c(5.19,7.43,6.94,8.11,18.75,14.88,16.06,19.12,3.21,7.58)

```
# Give the chart file a name.
png(file = "nls.png")

# Plot these values.
plot(xvalues,yvalues)

# Take the assumed values and fit into the model.
model <- nls(yvalues ~ b1*xvalues^2+b2,start = list(b1 = 1,b2 = 3))

# Plot the chart with new data by fitting it to a prediction from 100 data points.
new.data <- data.frame(xvalues = seq(min(xvalues),max(xvalues),len = 100))
lines(new.data$xvalues,predict(model,newdata = new.data))

# Save the file.
dev.off()
# Get the sum of the squared residuals.
print(sum(resid(model)^2))

# Get the confidence intervals on the chosen values of the coefficients.
print(confint(model))
```

**output:**



```
xvalues <- c(1.6,2.1,2,2.23,3.71,3.25,3.4,3.86,1.19,2.21)
yvalues <- c(5.19,7.43,6.94,8.11,18.75,14.88,16.06,19.12,3.21,7.58)

# Give the chart file a name.
png(file = "nls.png")


# Plot these values.
plot(xvalues,yvalues)


# Take the assumed values and fit into the model.
model <- nls(yvalues ~ b1*xvalues^2+b2,start = list(b1 = 1,b2 = 3))

# Plot the chart with new data by fitting it to a prediction from 100 data p
new.data <- data.frame(xvalues = seq(min(xvalues),max(xvalues),len = 100))
lines(new.data$xvalues,predict(model,newdata = new.data))

# Save the file.
dev.off()
```

```
Data
model        List of 6
new.data     100 obs. of 1 variable
Values
  xvalues    num [1:10] 1.6 2.1 2 2.23 3.71 3.25 3.4 3.86 1…
  yvalues    num [1:10] 5.19 7.43 6.94 8.11 18.75 ...
```

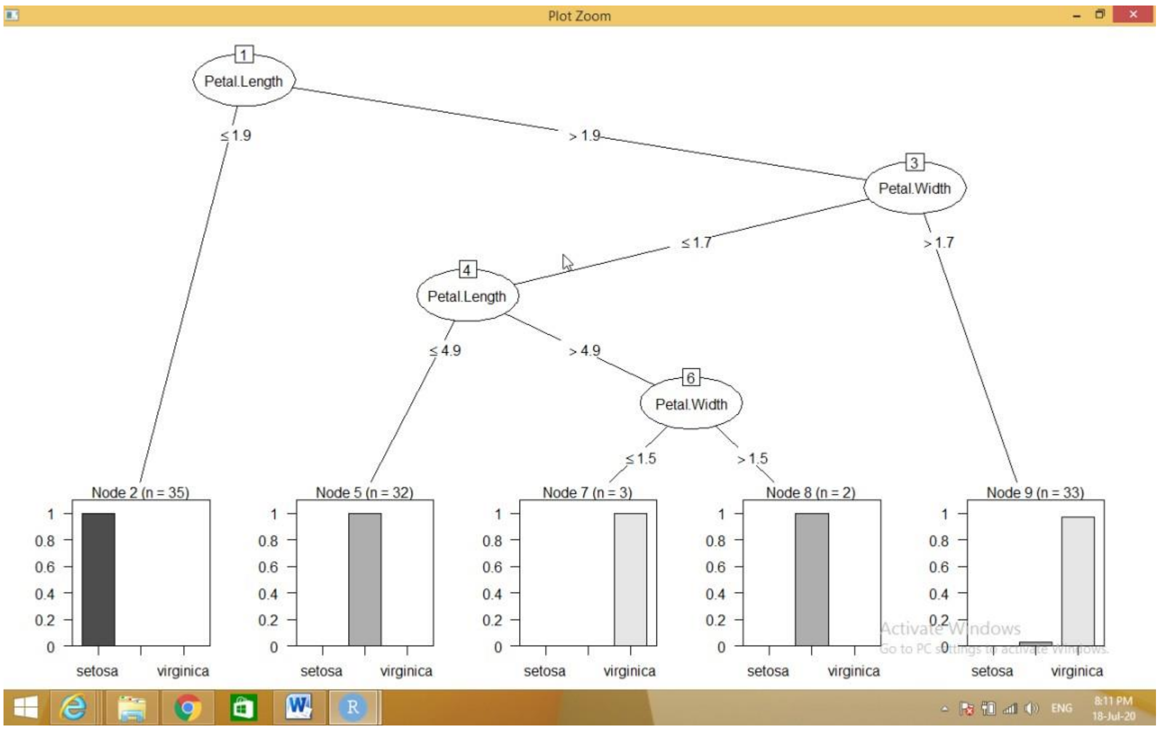
```
> dev.off()
null device
          1
> # Get the sum of the squared residuals.
> print(sum(resid(model)^2))
[1] 1.081935
> # Get the confidence intervals on the chosen values of the coefficients.
> print(confint(model))
Waiting for profiling to be done...
       2.5%    97.5%
b1 1.137708 1.253135
b2 1.497364 2.496484
> # Plot these values.
> plot(xvalues,yvalues)
>
```

**c) Write R program to find Decision Tree with the sample data and visualize the regression graphically.**

**Source code:**

```
#Data Load
data("iris")
#Install the required packages
install.packages("caret")
install.packages("C50")
#Library  invoke
library(caret)
library(C50)
#To make the results consistent across the runs

set.seed(7)
#Data Partition
inTraininglocal<-createDataPartition(iris$Species,p=.70,list = F)
training<-iris[inTraininglocal,]
testing<-iris[-inTraininglocal,]

#Model Building
model<-C5.0(Species~.,data = training)
#Generate the model summary
summary(model)
#Predict for test data set
pred<-predict.C5.0(model,testing[,-5]) #type ="prob"
#Accuracy of the algorithm
a<-table(testing$Species,pred)
sum(diag(a))/sum(a)
#Visualize the decision tree
plot(model)
```

**output:**


Plot Zoom
1
Petal.Length
≤ 1.9
> 1.9
3
Petal.Width
≤ 1.7
> 1.7
4
Petal.Length
≤ 4.9
> 4.9
6
Petal.Width
≤ 1.5
> 1.5
Node 2 (n = 35)
Node 5 (n = 32)
Node 7 (n = 3)
Node 8 (n = 2)
Node 9 (n = 33)
1
0.8
0.6
0.4
0.2
0
setosa
virginica
Activate Windows
Go to PC settings to activate Windows.
ENG
8:11 PM
18-Jul-20

**4. Write R program to find the following Distribution with the sample data and visualize the linear regression graphically.**

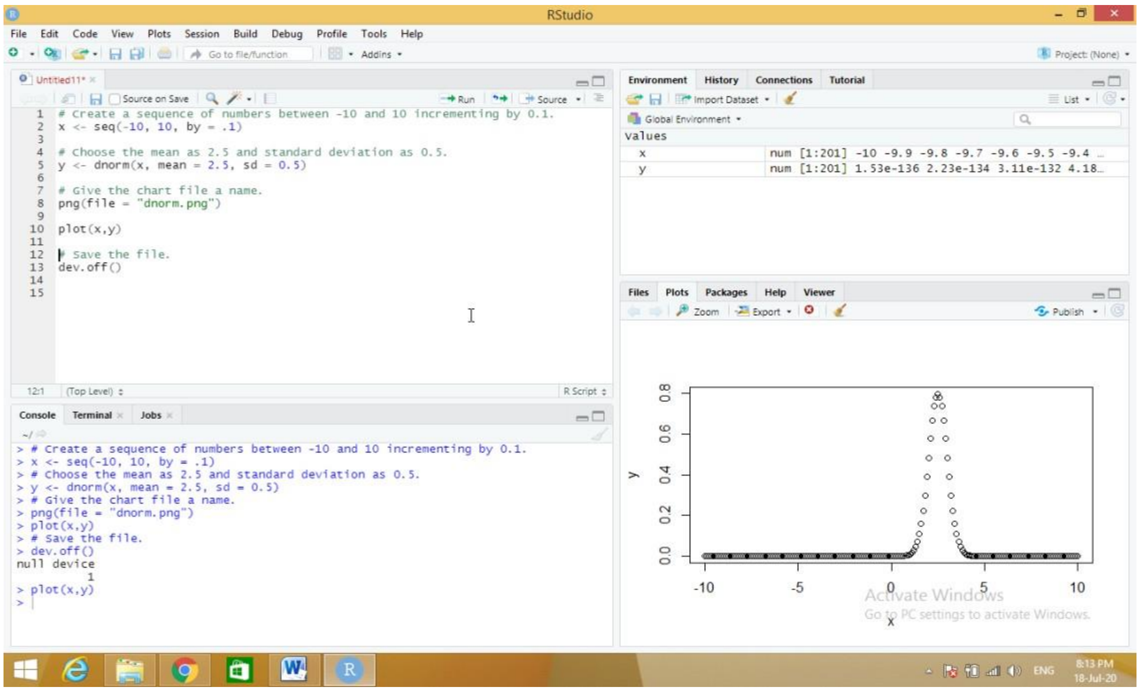
**Source code:**

**a) Normal Distribution**

**dnorm**

```
# 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()
```

**output:**



```
# 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()
```

```
values
  x        num [1:201] -10 -9.9 -9.8 -9.7 -9.6 -9.5 -9.4 ...
  y        num [1:201] 1.53e-136 2.23e-134 3.11e-132 4.18...
```

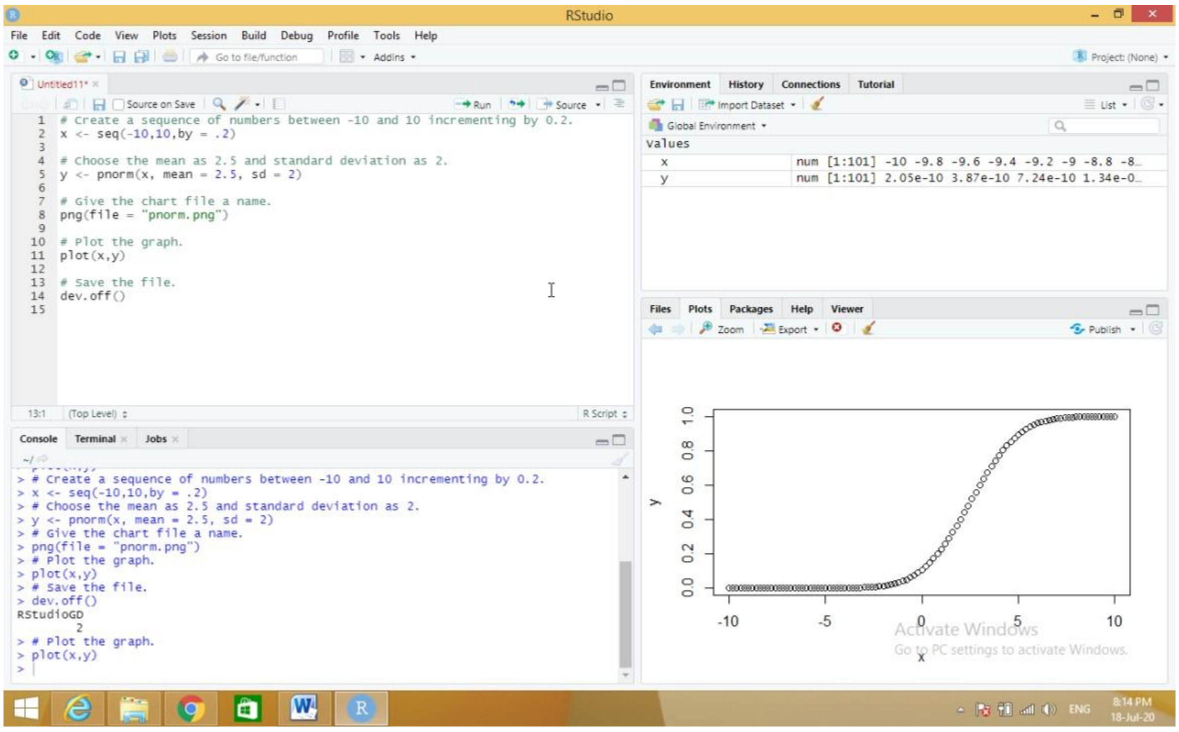
```
> # Create a sequence of numbers between -10 and 10 incrementing by 0.1.
> x <- seq(-10, 10, by = .1)
> # Choose the mean as 2.5 and standard deviation as 0.5.
> y <- dnorm(x, mean = 2.5, sd = 0.5)
> # Give the chart file a name.
> png(file = "dnorm.png")
> plot(x,y)
> # Save the file.
> dev.off()
null device
          1
> plot(x,y)
>
```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Environment History Connections Tutorial
Files Plots Packages Help Viewer
y
0.0 0.2 0.4 0.6 0.8
-10 -5 0 5 10
x
Activate Windows
Go to PC settings to activate Windows.

**Pnorm**

```
# 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()
```

**output:**



```
# 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()
```

13:1 (Top Level) R Script

Console Terminal Jobs

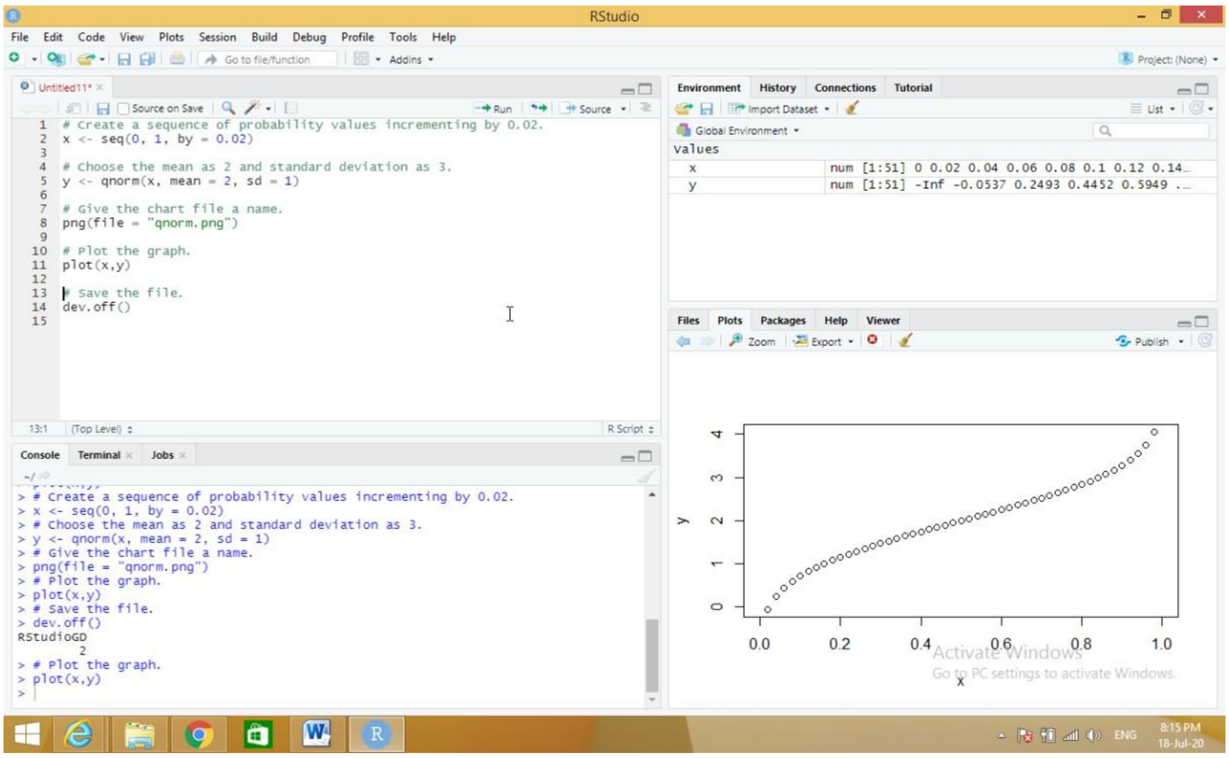
```
> # Create a sequence of numbers between -10 and 10 incrementing by 0.2.
> x <- seq(-10,10,by = .2)
> # Choose the mean as 2.5 and standard deviation as 2.
> y <- pnorm(x, mean = 2.5, sd = 2)
> # Give the chart file a name.
> png(file = "pnorm.png")
> # Plot the graph.
> plot(x,y)
> # Save the file.
> dev.off()
RStudioGD
        2
> # Plot the graph.
> plot(x,y)
>
```

```
Values
  x      num [1:101] -10 -9.8 -9.6 -9.4 -9.2 -9 -8.8 -8…
  y      num [1:101] 2.05e-10 3.87e-10 7.24e-10 1.34e-0…
```

**qnorm**

```
# 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()
```

**output:**



```
# 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()
```

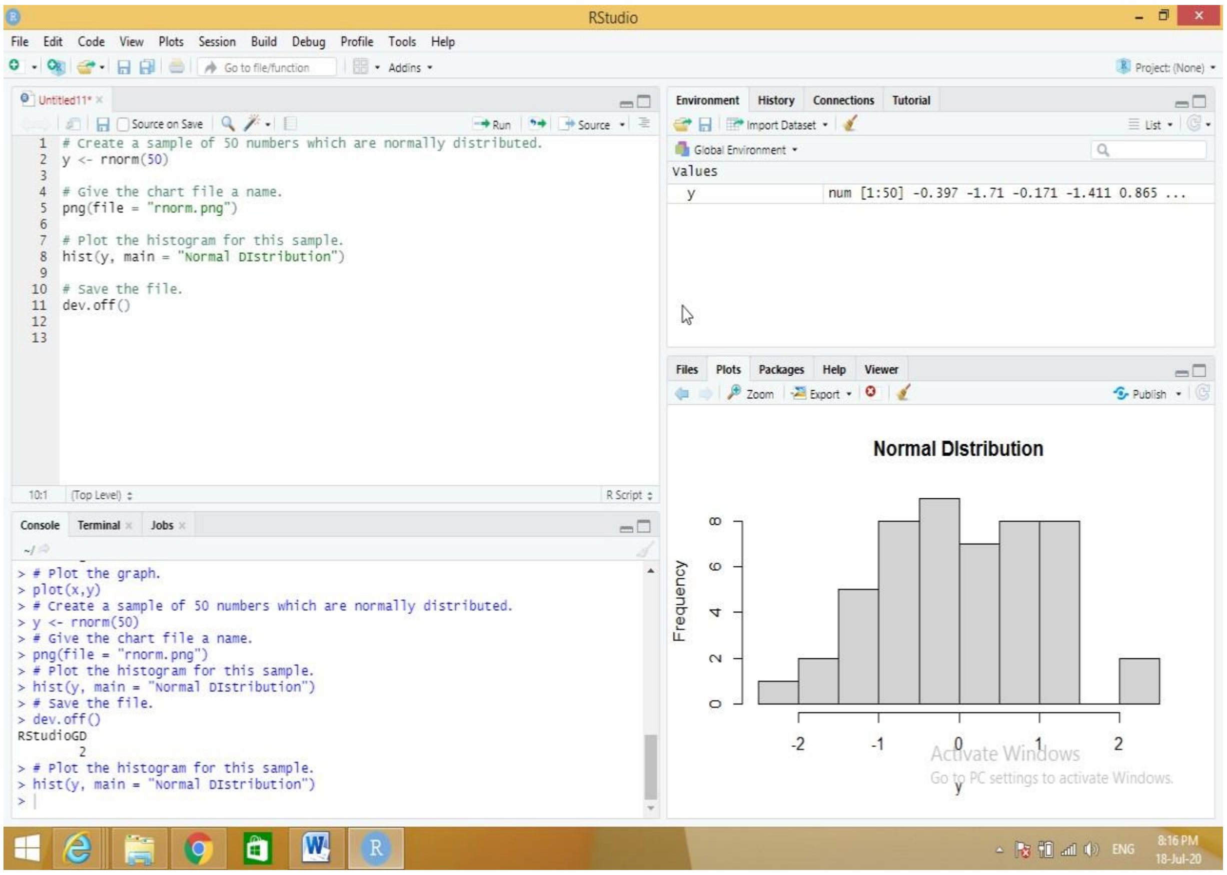
```
Values
x    num [1:51] 0 0.02 0.04 0.06 0.08 0.1 0.12 0.14...
y    num [1:51] -Inf -0.0537 0.2493 0.4452 0.5949 ...
```

```
> # 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()
RStudioGD
        2
> # Plot the graph.
> plot(x,y)
> 
```

**rnorm**

```
# 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()
```

**output:**



```
# 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()
```

```
Values
  y                    num [1:50] -0.397 -1.71 -0.171 -1.411 0.865 ...
```

```
> # Plot the graph.
> plot(x,y)
> # 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()
RStudioGD
        2
> # Plot the histogram for this sample.
> hist(y, main = "Normal DIstribution")
> |
```

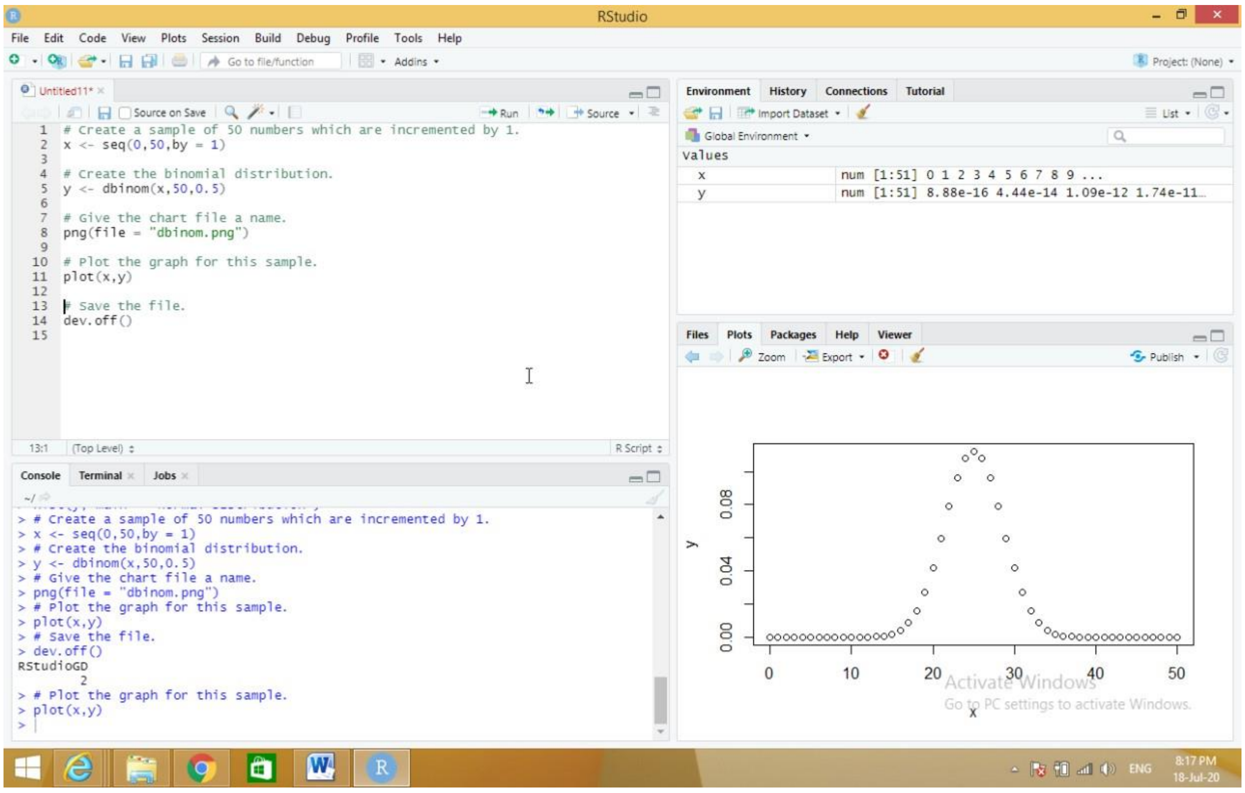
## b) Binomial Distribution

**dbinom**

**<u>Source code:</u>**

```
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)
# Create the binomial distribution.
y <- dbinom(x,50,0.5)
# Give the chart file a name.
png(file = "dbinom.png")
# Plot the graph for this sample.
plot(x,y)
# Save the file.
dev.off()
```

**<u>output:</u>**



```
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)

# Create the binomial distribution.
y <- dbinom(x,50,0.5)

# Give the chart file a name.
png(file = "dbinom.png")

# Plot the graph for this sample.
plot(x,y)

# Save the file.
dev.off()
```

```
Values
  x        num [1:51] 0 1 2 3 4 5 6 7 8 9 ...
  y        num [1:51] 8.88e-16 4.44e-14 1.09e-12 1.74e-11…
```

```
> # Create a sample of 50 numbers which are incremented by 1.
> x <- seq(0,50,by = 1)
> # Create the binomial distribution.
> y <- dbinom(x,50,0.5)
> # Give the chart file a name.
> png(file = "dbinom.png")
> # Plot the graph for this sample.
> plot(x,y)
> # Save the file.
> dev.off()
RStudioGD
        2
> # Plot the graph for this sample.
> plot(x,y)
> 
```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled11*
Source on Save
Run
Source
Environment History Connections Tutorial
Import Dataset
Global Environment
Files Plots Packages Help Viewer
Zoom
Export
Publish
Console Terminal Jobs
y
0.00
0.04
0.08
x
0
10
20
30
40
50
Activate Windows
Go to PC settings to activate Windows.
8:17 PM
18-Jul-20

**Pbinom**

```
# Probability of getting 26 or less heads from a 51 tosses of a coin.
x <- pbinom(26,51,0.5)

print(x)
```

**output:**

```
print(x)
[1] 0.610116
```

**qbinom**

```
# How many heads will have a probability of 0.25 will come out when a coin
# is tossed 51 times.
x <- qbinom(0.25,51,1/2)

print(x)
```

**output:**

```
print(x)
[1] 23
```

**rbinom**

```
# Find 8 random values from a sample of 150 with probability of 0.4.
x <- rbinom(8,150,.4)

print(x)
```

**output:**

```
print(x)
[1] 68 59 55 49 51 59 53 55
```

**5. Write R program to do the following tests with the sample data and visualize the results graphically.**

**Source code:**

**a) χ2-test**

```
library("MASS")
print(str(Cars93))
# Loading the Mass library.

# Creating a data frame from the main data set.
car_data<- data.frame(Cars93$AirBags, Cars93$Type)
# Creating a table with the needed variables.
car_data = table(Cars93$AirBags, Cars93$Type)
print(car_data)
# Performing the Chi-Square test.
print(chisq.test(car_data))
```

**output:**

```
print(str(Cars93))
'data.frame':         93 obs. of  27 variables:
 $ Manufacturer      : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model             : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74 73 35 ...
 $ Type              : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
 $ Min.Price         : num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
 $ Price             : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ Max.Price         : num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
 $ MPG.city          : int  25 18 20 19 22 22 19 16 19 16 ...
 $ MPG.highway       : int  31 25 26 26 30 31 28 25 27 25 ...
 $ AirBags           : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ DriveTrain        : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
 $ Cylinders         : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5 ...
 $ EngineSize        : num 1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
 $ Horsepower        : int  140 200 172 172 208 110 170 180 170 200 ...
 $ RPM               : int  6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
 $ Rev.per.mile      : int  2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
 $ Man.trans.avail   : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
 $ Fuel.tank.capacity: num  13.2 18 16.9 21.1 21.1 16.4 18 23 18.8 18 ...
 $ Passengers        : int  5 5 5 6 4 6 6 6 5 6 ...
 $ Length            : int  177 195 180 193 186 189 200 216 198 206 ...
 $ Wheelbase         : int  102 115 102 106 109 105 111 116 108 114 ...
 $ Width             : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Turn.circle       : int  37 38 37 37 39 41 42 45 41 43 ...
 $ Rear.seat.room    : num 26.5 30 28 31 27 28 30.5 30.5 26.5 35 ...
 $ Luggage.room      : int  11 15 14 17 13 16 17 21 14 18 ...
 $ Weight            : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
 $ Origin            : Factor w/ 2 levels "USA","non-USA": 2 2 2 2 2 1 1 1 1 1 ...
 $ Make              : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
```

```
NULL
> # Creating a data frame from the main data set.
> car_data<- data.frame(Cars93$AirBags, Cars93$Type)
> # Creating a table with the needed variables.
> car_data = table(Cars93$AirBags, Cars93$Type)
> print(car_data)

                  Compact Large Midsize Small Sporty Van
  Driver & Passenger      2    4      7     0     3 0
  Driver only           9    7     11     5     8  3
  None                5    0      4   16     3  6
> # Performing the Chi-Square test.
> print(chisq.test(car_data))

            Pearson's Chi-squared test

data: car_data
X-squared = 33.001, df = 10, p-value = 0.0002723
```

**b) t-test**

```
x  <- c(0.593, 0.142, 0.329, 0.691, 0.231, 0.793, 0.519, 0.392, 0.418)
t.test(x, alternative="greater", mu=0.3)
```

**output:**

```
t.test(x, alternative="greater", mu=0.3)

            One Sample t-test

data:  x
t = 2.2051, df = 8, p-value = 0.02927
alternative hypothesis: true mean is greater than 0.3
95 percent confidence interval:
 0.3245133       Inf
sample estimates:
mean of x
0.4564444
```

**c) F-test**

```
#f test

x<-c(18,19,22,25,27,28,41,45,51,55)
y<-c(14,15,15,17,18,22,25,25,27,34)
print(var.test(x,y))
```

**output:**

```
F test to compare two variances

data:  x and y
F = 4.3871, num df = 9, denom df =
9, p-value = 0.03825
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
  1.089699 17.662528
sample estimates:
ratio of variances
          4.387122
```