Netaji Subhas University of Technology

Data Mining

Practical File

Name : Anshuman Rai

Class : COE-3

Roll No. : 2019UCO1679

INDEX

S no.	Topic
1.	Analysis of Data Know the types of data – ordinal, nominal, ratio, interval
2.	Statistical Data Analysis Find the mean, median, variance and standard deviation of data.
3.	Proximity Measures Calculate dissimilarity matrix in any programming language of choice (C, C++, Java, Python).
4.	Data Pre-processing a. Handling missing values using various techniques, finding outliers in data, discretisation. b. Normalization and standardization of data.
5.	Dimensionality Reduction using Principal Components (PCA algorithm)
6.	Implement classification using decision trees
7.	Regression Tasks - Implement linear regression in python
8.	Association Rule Mining – Apriori algorithm
9.	Visualisation Techniques Explore data using various visualization techniques available in Python.
10.	Use Bayesian Learning for classification
11.	KNN algorithm
12.	Implement K-means clustering algorithm.

Q1. Analysis of Data

Know the types of data – ordinal, nominal, ratio, interval .

Find the mean, median, variance and standard deviation of data.

Tool Used: Weka

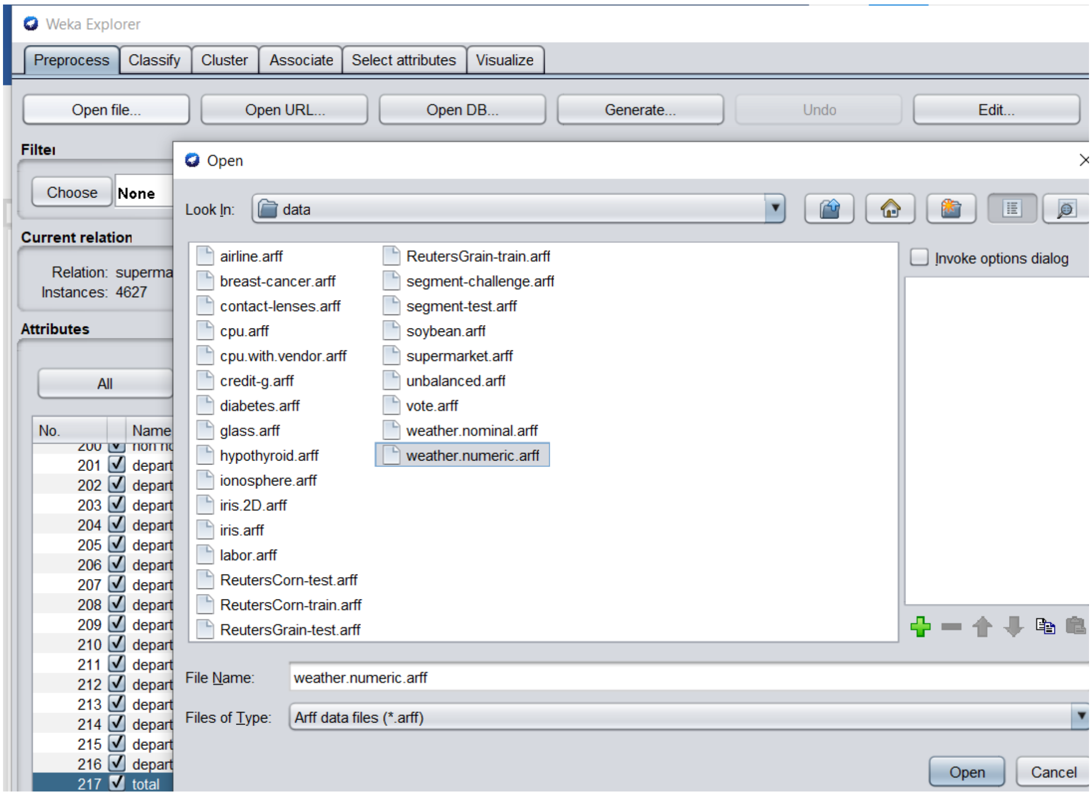
Theory :

Types of data are as follows :

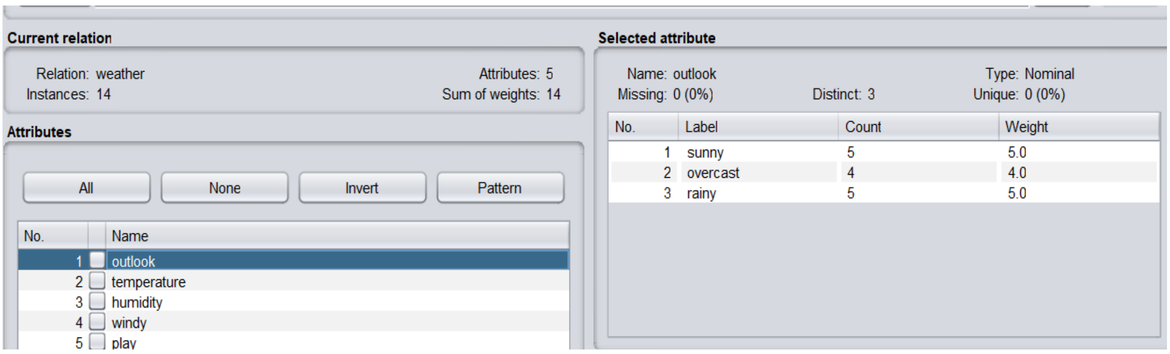
a. Nominal : The values of a nominal attribute are symbols or names of things. Each value represents some kind of category, code, or state, and so nominal attributes are also referred to as categorical. The values do not have any meaningful order.
b. Ordinal : Ordinal attribute is an attribute with possible values that have a meaningful order or ranking among them, but the magnitude between successive values is not known.
c. Interval : They are measured on a scale of equal-size units. The values of interval-scaled attributes have order and can be positive, 0, or negative. Thus, in addition to providing a ranking of values, such attributes allow us to compare and quantify the difference between values. But we cannot say that a value is a multiple of another value since its zero-point is not known. Example temperature in Celsius or Fahrenheit.
d. Ratio : This is a numeric attribute with an inherent zero-point. That is, if a measurement is ratio-scaled, we can speak of a value as being a multiple (or ratio) of another value. In addition, the values are ordered, and we can also compute the difference between values, as well as the mean, median, and mode. Example , temperature in Kelvin.

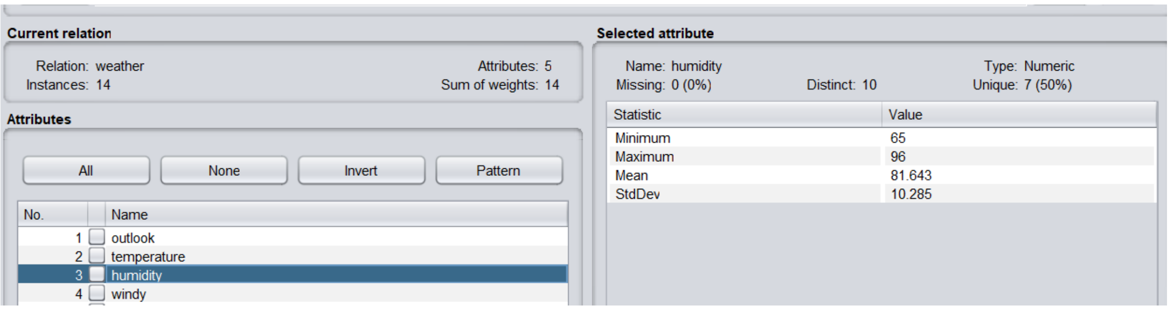
Steps :

1. Open weka Explorer & Select data – Open weka data folder and load the data (here weather_numeric.arff taken)



2. Click on the attributes to view their details. One of the details will the type of the attribute.




Current relation
Relation: weather
Attributes: 5
Instances: 14
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
Selected attribute
Name: humidity
Type: Numeric
Missing: 0 (0%)
Distinct: 10
Unique: 7 (50%)
Statistic
Value
Minimum
65
Maximum
96
Mean
81.643
StdDev
10.285

Q2. Statistical Data Analysis

Find the mean, median, variance and standard deviation of data.

Tool Used: Weka

Theory :

Mean : It is the average of all the given values It is a measure of central tendency of the data is given by

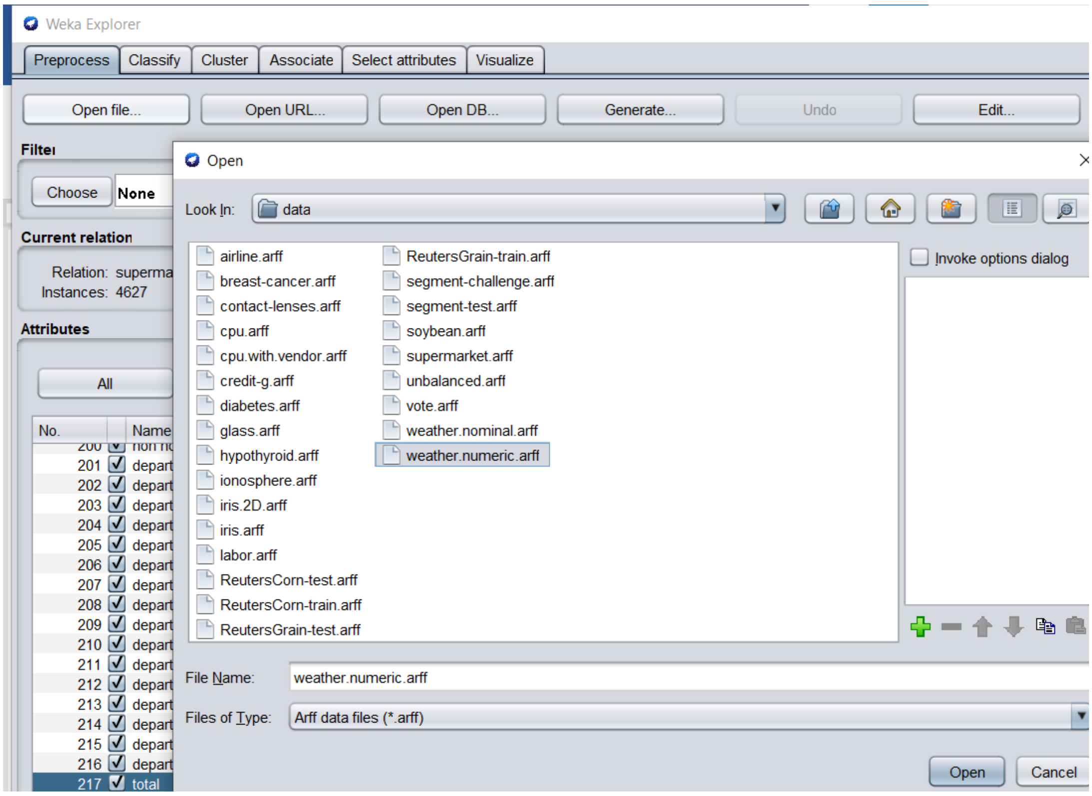
$$\overline{x} = \frac{\sum x}{n}$$

Variance and standard deviation are measures of dispersion or deviation from the mean.

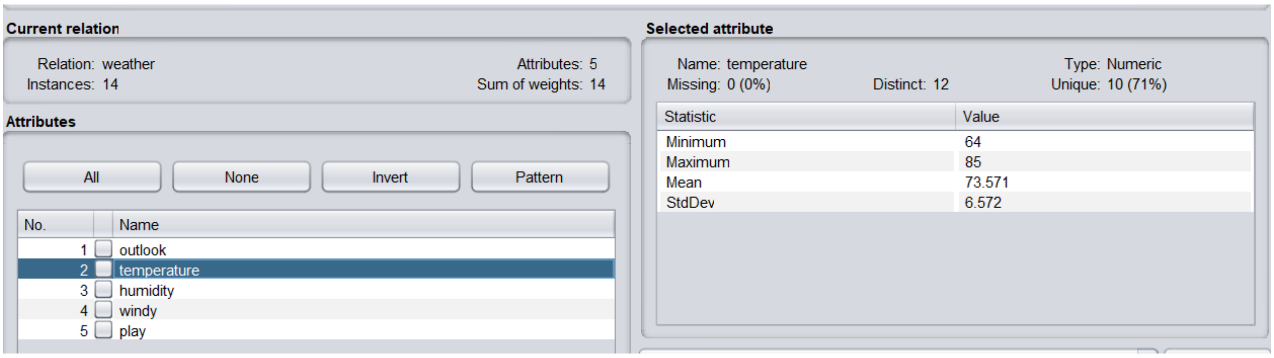
$$Variance, \sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

$$Standard\ Deviation, \sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

1. Open weka Explorer & Select data – Open weka data folder and load the data (here weather_numeric.arff taken)



2. Choose a numerical attribute(here temperature). We can see the mean and standard deviation of the attribute.
Variance = (Standard Deviation)^2
= 43.191


Current relation
Relation: weather
Instances: 14
Attributes: 5
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
Selected attribute
Name: temperature
Missing: 0 (0%)
Distinct: 12
Type: Numeric
Unique: 10 (71%)
Statistic
Value
Minimum
64
Maximum
85
Mean
73.571
StdDev
6.572

Q3. Proximity Measures

Calculate dissimilarity matrix in any programming language of choice (C, C++, Java, Python).

Dissimilarity matrix stores a collection of proximities that are available for all pairs of *n* objects. It is often represented by an n-by-n table:

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix}$$

Source : The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011

where d(i, j) is the measured **dissimilarity** or “difference” between objects i and j. In general, d(i, j) is a non-negative number that is close to 0 when objects i and j are highly similar or “near” each other, and becomes larger the more they differ. Note that d(i,i) = 0; that is, the difference between an object and itself is 0. Also,

d(i,j) = d(j,i)

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
int size;
cout<<"Enter the size of matrix :";
cin>>size;
int arr[size][2];
for(int i=0;i<size;i++)
{
```

```
cout<<"Enter the x coordinate of "<<i<<"th element :";
cin>>arr[i][0];
cout<<"Enter the y coordinate of "<<i<<"th element :";
cin>>arr[i][1];
}
float matrix[size][size];
for(int i=0;i<size;i++)
{
for(int j=0;j<size;j++)
{
if(i==j)
matrix[i][j]=0;
else
matrix[i][j]= sqrt(pow(arr[i][0]-arr[j][0],2)+pow(arr[i][1]-arr[j][1],2));
}
}
cout<<"Through Euclidean distance"<<endl;
for(int i=0;i<size;i++)
{
for(int j=0;j<size;j++)
{
cout<<matrix[i][j]<<" ";
}
cout<<endl;
}
float matrix2[size][size];
for(int i=0;i<size;i++)
{
for(int j=0;j<size;j++)
{
if(i==j)
matrix2[i][j]=0;
else
matrix2[i][j]= abs(arr[i][0]-arr[j][0])+abs(arr[i][1]-arr[j][1]) ;
}
}
cout<<"Through Manhattan distance"<<endl;
for(int i=0;i<size;i++)
{
for(int j=0;j<size;j++)
{
cout<<matrix2[i][j]<<" ";
}
cout<<endl;
}
```

```
    return 0;
}
```

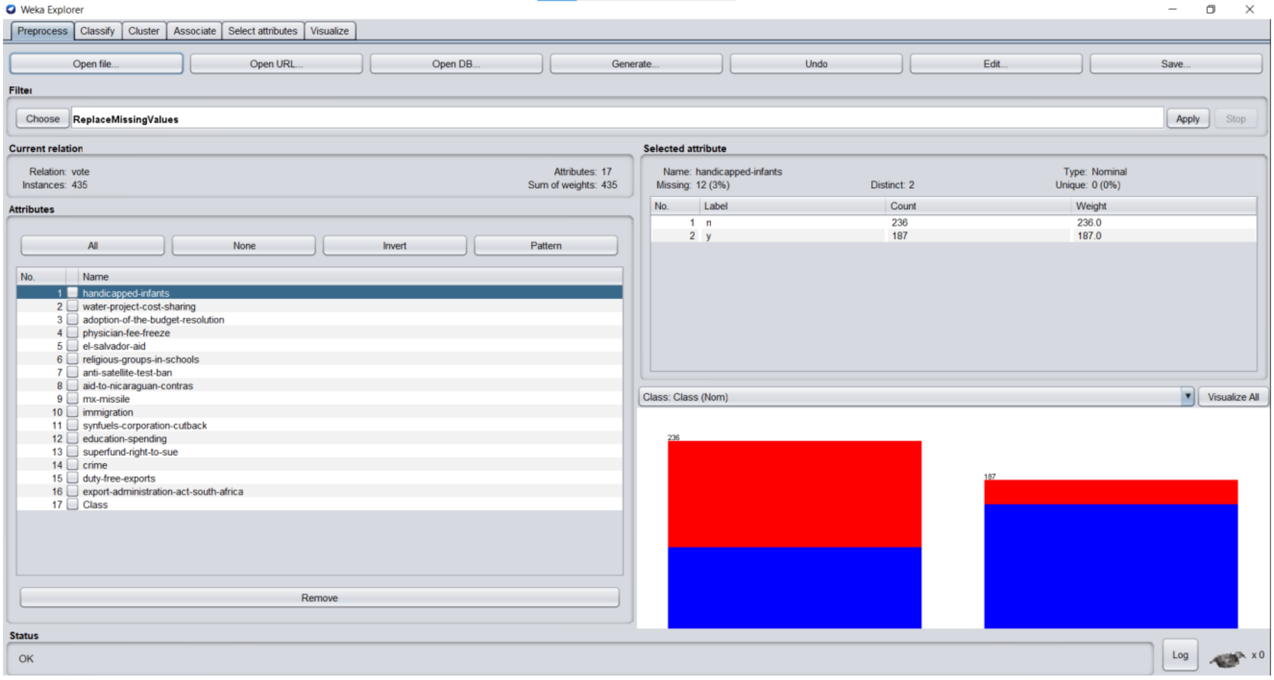
Output :

```
Enter the size of matrix 6
Enter the x coordinate of 0th element 1
Enter the y coordinate of 0th element 2
Enter the x coordinate of 1th element 2
Enter the y coordinate of 1th element 3
Enter the x coordinate of 2th element 10
Enter the y coordinate of 2th element 15
Enter the x coordinate of 3th element 11
Enter the y coordinate of 3th element 20
Enter the x coordinate of 4th element 4
Enter the y coordinate of 4th element 9
Enter the x coordinate of 5th element 12
Enter the y coordinate of 5th element 17
Through Euclidean distance
0 1.41421 15.8114 20.5913 7.61577 18.6011
1.41421 0 14.4222 19.2354 6.32456 17.2047
15.8114 14.4222 0 5.09902 8.48528 2.82843
20.5913 19.2354 5.09902 0 13.0384 3.16228
7.61577 6.32456 8.48528 13.0384 0 11.3137
18.6011 17.2047 2.82843 3.16228 11.3137 0
Through Manhattan distance
0 2 22 28 10 26
2 0 20 26 8 24
22 20 0 6 12 4
28 26 6 0 18 4
10 8 12 18 0 16
26 24 4 4 16 0
```

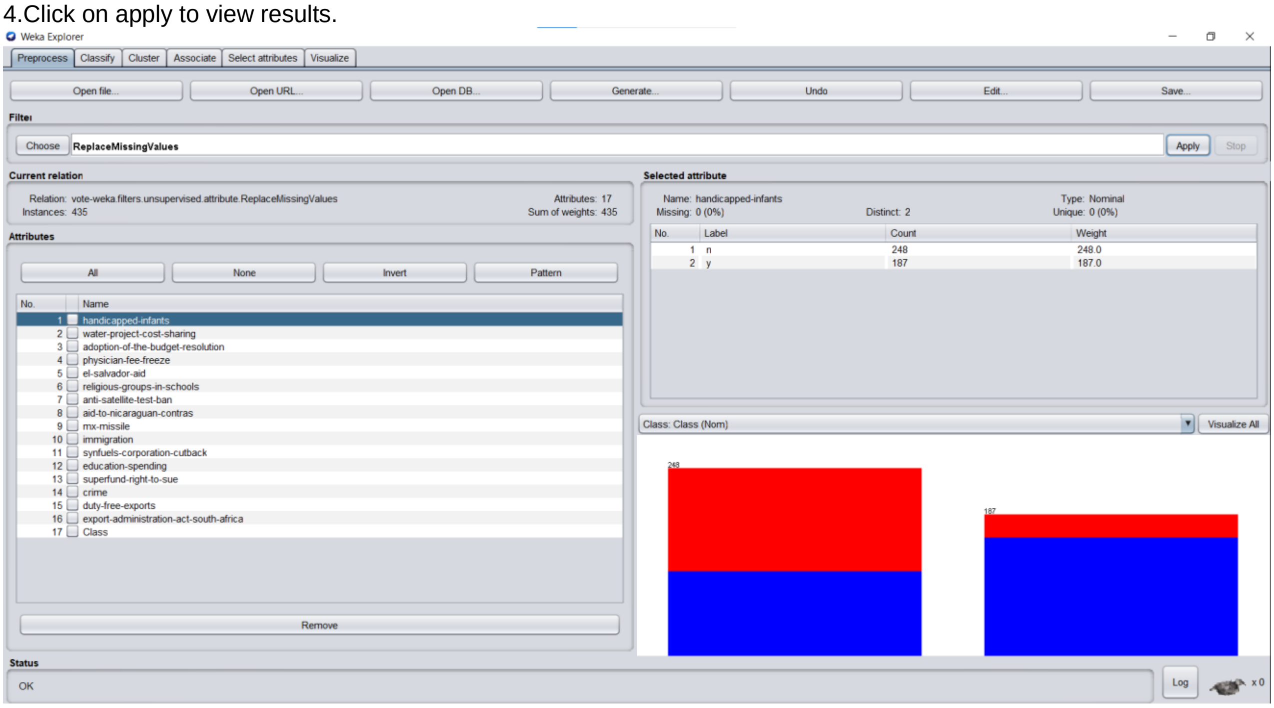
Q4. Data Pre-processing

a. Handling missing values using various techniques, finding outliers in data, discretisation.
Handling missing values:

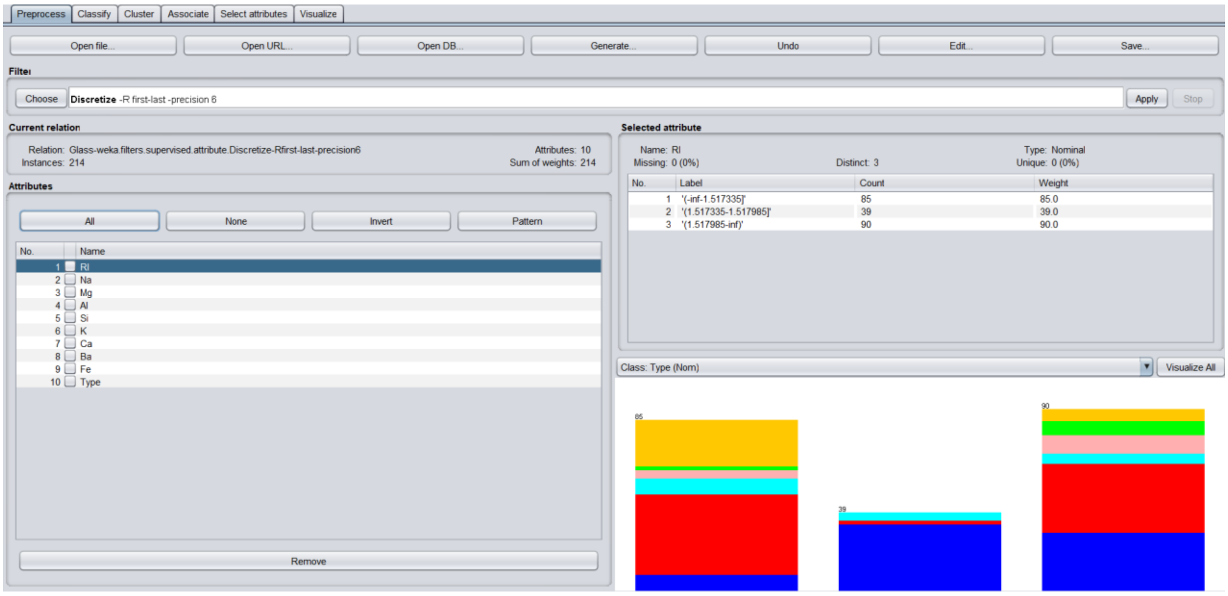
Initial Data:



Replace missing values with mode(nominal) or mean(numeric);
1.Select any dataset.
2.Choose the attribute to discretize.
3.Click on choose tab in filters →unsupervised→attributes→ReplaceMissingValues.
4.Click on apply to view results.

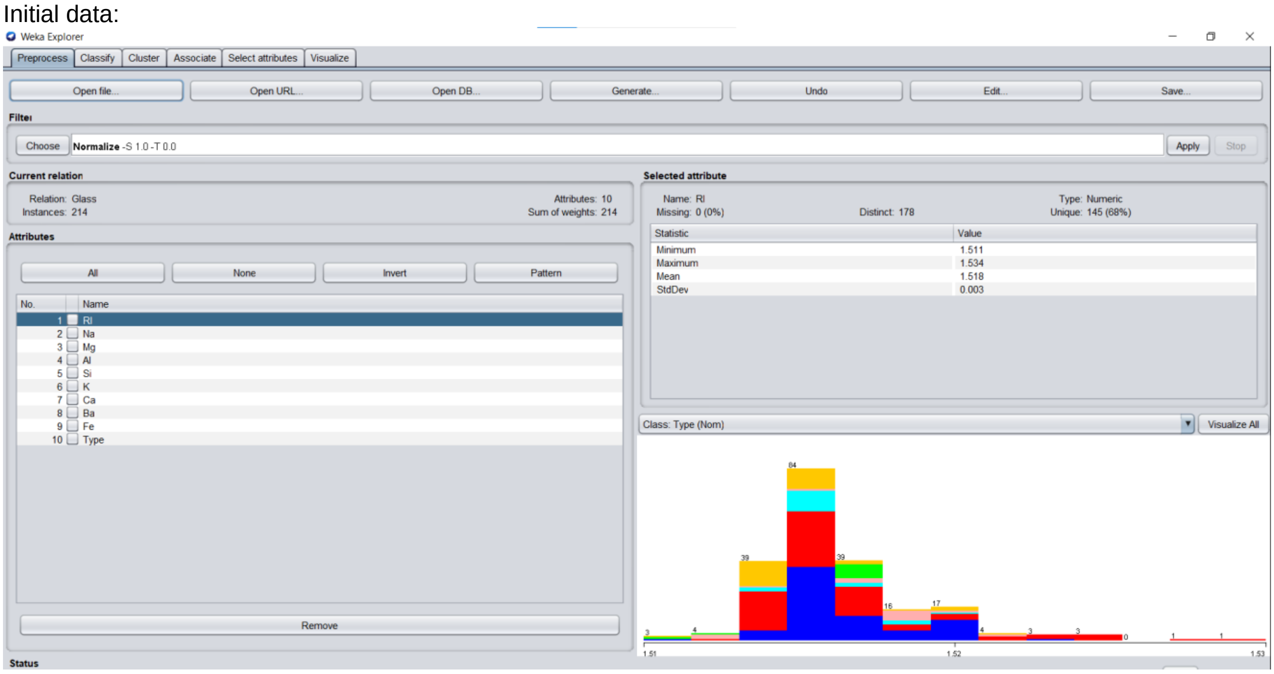


Discretization :
1.Select any dataset.
2.Choose the attribute to discretize.
3.Click on choose tab in filters →supervised→attributes→discretize.
4.Click on apply to view results.

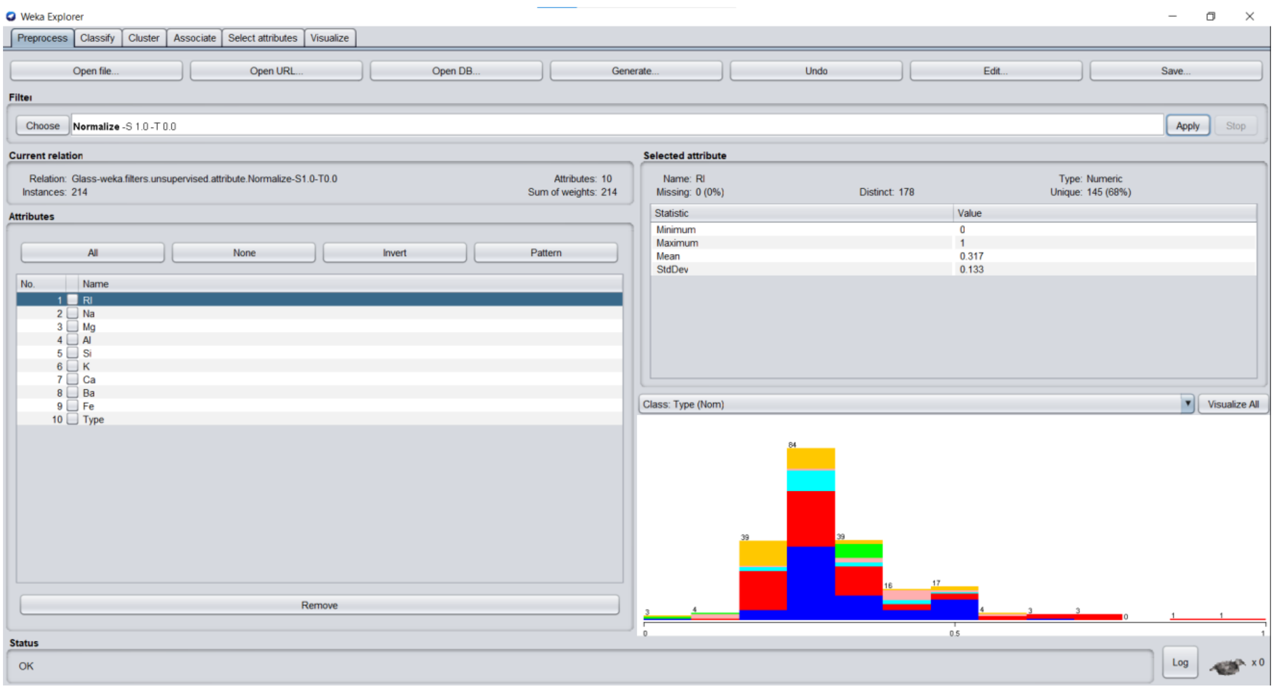


b. Normalization and standardization of data.

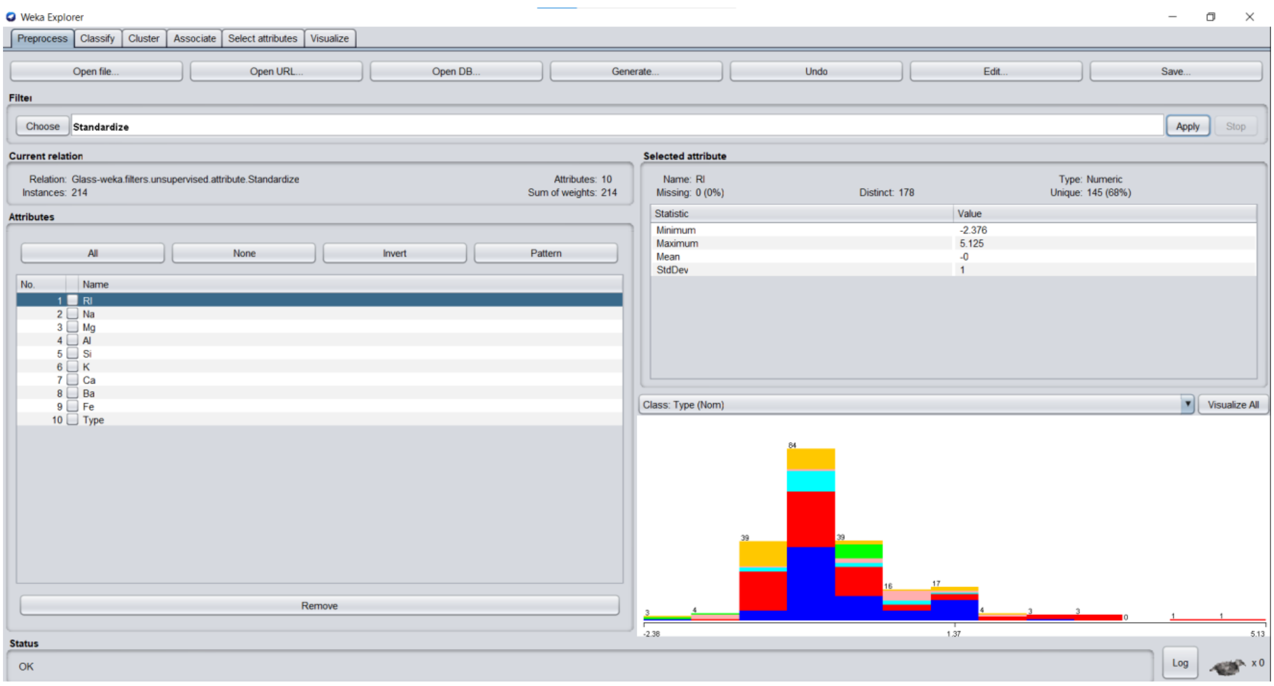
Initial data:



After normalization:
1.Select any dataset.
2.Choose the attribute to normalize.
3.Click on choose tab in filters →unsupervised→attributes→nomalize.
4.Click on apply to view results.



After standardization:
1.Select any dataset.
2.Choose the attribute to discretize.
3.Click on choose tab in filters →unsupervised→attributes→standardize.
4.Click on apply to view results.



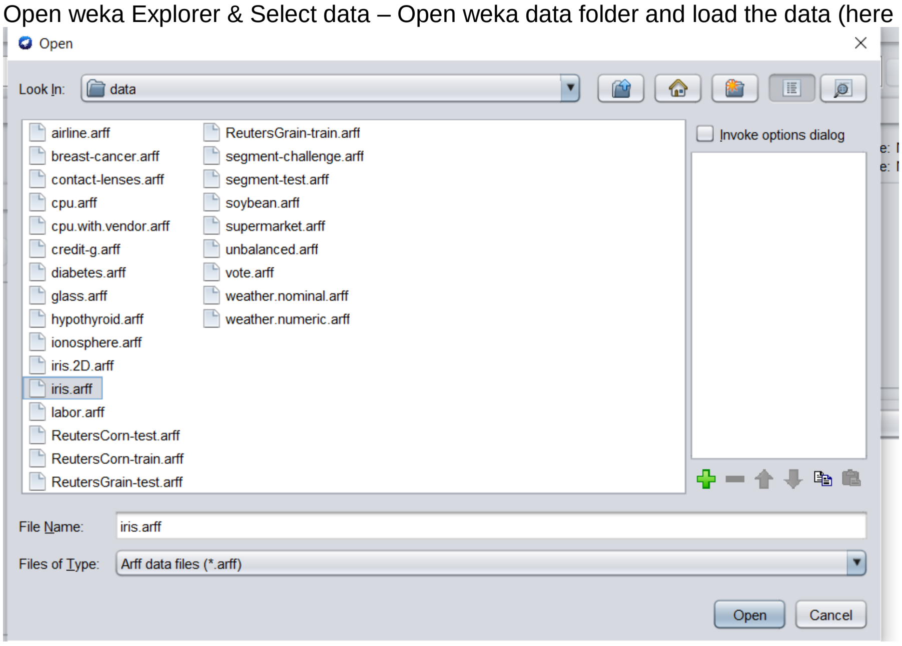
Q5. Implement dimensionality reduction using Principal Components (PCA algorithm)

Theory :

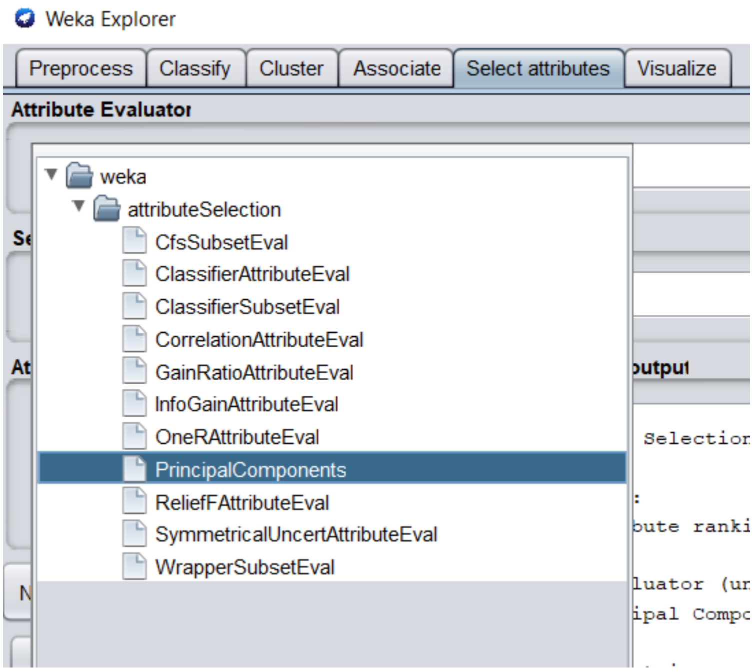
Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation.

Steps:

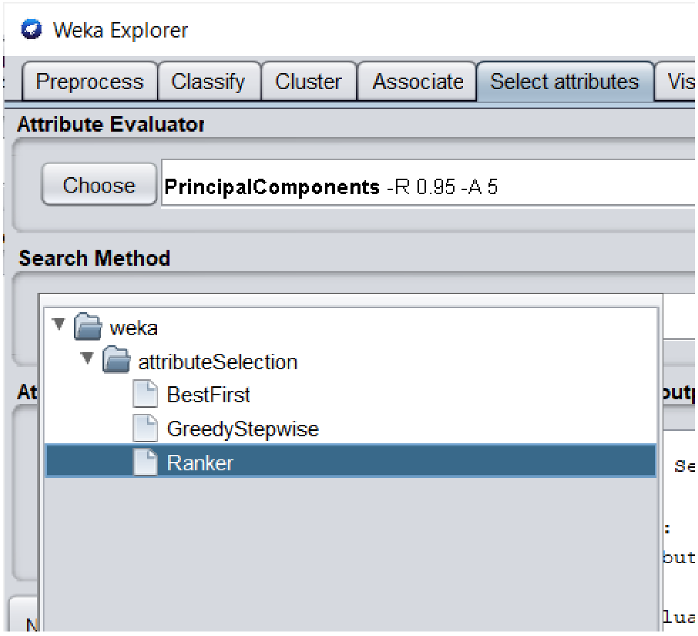
1. Open weka Explorer & Select data – Open weka data folder and load the data (here iris.arff taken)



2. Now from the top dialogue box select: Select Attributes
   a. Attribute evaluator : →Choose→weka→attributeSelection→PrincipleComponents



   b. Search Method : →Choose→weka→attributeSelection→Ranker



Click on the name of algorithm to check the status and set parameters manually.

3. Click on start . The results of PCA will be calculated and displayed.

```
=== Run information ===

Evaluator:    weka.attributeSelection.PrincipalComponents -R 0.95 -A 5
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Evaluation mode:    evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
	Attribute ranking.

Attribute Evaluator (unsupervised):
	Principal Components Attribute Transformer

Correlation matrix
  1     -0.11    0.87    0.82   -0.72    0.08    0.64
 -0.11   1      -0.42   -0.36    0.6    -0.46   -0.13
  0.87  -0.42    1       0.96   -0.92    0.2     0.72
  0.82  -0.36    0.96    1      -0.89    0.12    0.77
 -0.72   0.6    -0.92   -0.89    1      -0.5    -0.5
  0.08  -0.46    0.2     0.12   -0.5     1      -0.5
  0.64  -0.13    0.72    0.77   -0.5    -0.5     1


eigenvalue	proportion	cumulative
  4.34646	  0.62092	  0.62092	0.476petallength+0.465petalwidth-0.451class=Iris-setosa+0.411sepallength+0.349class=Iris-virginica...
  1.76093	  0.25156	  0.87248	0.7  class=Iris-versicolor-0.471class=Iris-virginica-0.451sepalwidth-0.229class=Iris-setosa-0.158sepallength...
  0.68223	  0.09746	  0.96995	-0.75sepalwidth-0.441sepallength-0.36class=Iris-versicolor+0.325class=Iris-virginica-0.073petallength...

Eigenvectors
 V1	 V2	 V3
 0.4107	-0.1575	-0.4409	sepallength
-0.2308	-0.4511	-0.7501	sepalwidth
 0.4755	-0.0219	-0.0733	petallength
 0.4652	-0.0853	-0.0384	petalwidth
-0.451 	-0.2292	 0.0351	class=Iris-setosa
 0.1022	 0.6999	-0.3598	class=Iris-versicolor
 0.3488	-0.4707	 0.3247	class=Iris-virginica

Ranked attributes:
 0.3791  1 0.476petallength+0.465petalwidth-0.451class=Iris-setosa+0.411sepallength+0.349class=Iris-virginica...
 0.1275  2 0.7  class=Iris-versicolor-0.471class=Iris-virginica-0.451sepalwidth-0.229class=Iris-setosa-0.158sepallength...
 0.0301  3 -0.75sepalwidth-0.441sepallength-0.36class=Iris-versicolor+0.325class=Iris-virginica-0.073petallength...

Selected attributes: 1,2,3 : 3
```

Q6. Implement classification using decision trees

Theory:

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

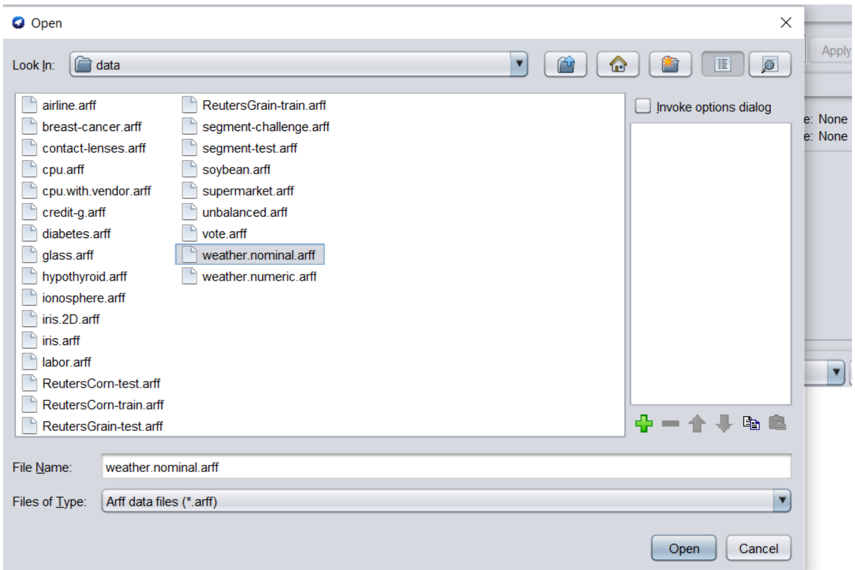
Entropy refers to a common way to measure impurity. In the decision tree, it measures the randomness or impurity in data sets.

Information Gain refers to the decline in entropy after the dataset is split. It is also called Entropy Reduction. Building a decision tree is all about discovering attributes that return the highest data gain.

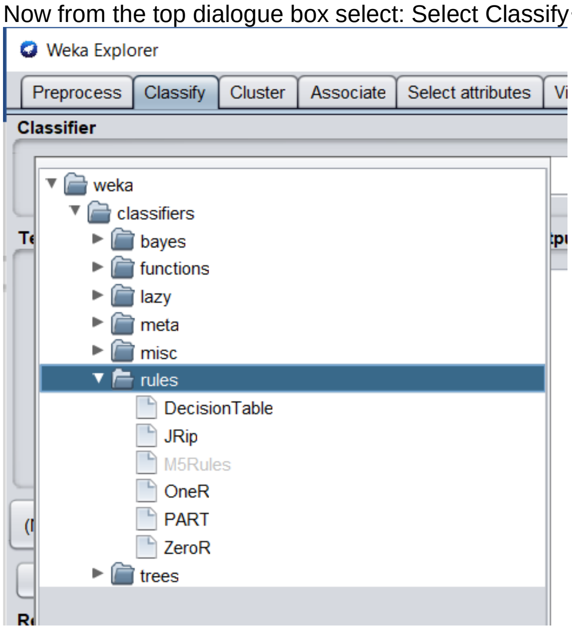
A greedy BFS approach is used to build the tree. At each step ,the attribute with highest Information gain is chosen to be the node.

Steps:

1. Open weka Explorer & Select data – Open weka data folder and load the data (here weather_nominal.arff taken)



2. Now from the top dialogue box select: Select Classify→choose→weka→rules→DecisionTable



Click on the name of algorithm to check the status and set parameters manually.

3. Click start. Results will be displayed after training.

```
Evaluation (for feature selection): CV (leave one out)
Feature set: 5

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances           9               64.2857 %
Incorrectly Classified Instances         5               35.7143 %
Kappa statistic                          0
Mean absolute error                      0.4524
Root mean squared error                  0.4797
Relative absolute error                 97.4359 %
Root relative squared error            100.0539 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.643      1.000    0.783      ?        0.500     0.643     yes
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.357     no
Weighted Avg.    0.643    0.643    ?          0.643    ?          ?        0.500     0.541

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = yes
 5 0 | b = no
```

Q7. Implement linear regression in python

In [4]:
```python
import csv
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pandas as pd
```

Load and Preprocess data

In [ ]:

In [5]:
```python
data = pd.read_csv("imports-85.data" ,na_values='?')
data
```

Out[5]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepow
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

Out[5]:

ke	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
a-ro	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111.0	5000.0	21	27	13495.0
a-ro	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111.0	5000.0	21	27	16500.0
a-ro	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154.0	5000.0	19	26	16500.0
di	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102.0	5500.0	24	30	13950.0
di	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115.0	5500.0	18	22	17450.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
/o	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114.0	5400.0	23	28	16845.0
/o	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	160.0	5300.0	19	25	19045.0
/o	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	134.0	5500.0	18	23	21485.0
/o	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	106.0	4800.0	26	27	22470.0
/o	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114.0	5400.0	19	25	22625.0

In [3]:
```python
data.isnull().sum()
```

Out[3]:
```
symboling             0
normalized-losses    41
make                  0
fuel-type             0
aspiration            0
num-of-doors          2
body-style            0
drive-wheels          0
engine-location       0
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-type           0
num-of-cylinders      0
engine-size           0
fuel-system           0
bore                  4
stroke                4
compression-ratio     0
horsepower            2
peak-rpm              2
city-mpg              0
highway-mpg           0
price                 4
dtype: int64
```

In [191]:
```python
#fill missing values for continuous attributes with mean
continuous = ['normalized-losses', 'bore', 'stroke', 'horsepower', 'peak-rpm', 'price']
```

In [191]:
```python
#fill missing values for continuous attributes with mean
continuous = ['normalized-losses', 'bore', 'stroke', 'horsepower', 'peak-rpm', 'price']
data[continuous]= data[continuous].astype(float)
data[continuous] = data[continuous].fillna(data.mean())
data.isnull().sum()
```

Out[191]:
```
symboling            0
normalized-losses    0
make                 0
fuel-type            0
aspiration           0
num-of-doors         2
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

In [192]:
```python
#fill missing values for categorical data with mode
data = data.fillna({"num-of-doors": data['num-of-doors'].mode().iloc[0]})
data.isnull().sum()
```

Out[192]:
```
symboling            0
normalized-losses    0
make                 0
fuel-type            0
aspiration           0
num-of-doors         0
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
```

In [193]:
```python
#manual label encoding of ordinal attributes
cleanup_nums = {"num-of-doors":     {"four": 4, "two": 2},
                "num-of-cylinders": {"four": 4, "six": 6, "five": 5, "eight": 8,
                                  "two": 2, "twelve": 12, "three":3 }}
data = data.replace(cleanup_nums)
```

In [197]:
```python
#one hot encoding of nominal attributes
nominal = []
price = data['price']
data = data.drop(['price'] , axis=1)
for i,col in enumerate(data.columns):
    if data[col].dtype == object:
        one_hot = pd.get_dummies(data.iloc[:,i])
        print(col , one_hot.columns)
        data[list(one_hot.columns)] = one_hot
        nominal.append(col)
for col in nominal:
    data = data.drop([col] ,axis=1 )
data['price'] = price
```

In [201]:
```python
print(list(data.columns))
print(data.to_numpy().shape)
processed_data = data.to_numpy()
```

```
['symboling', 'normalized-losses', 'num-of-doors', 'wheel-base', 'length', 'width', 'height', 'curb-weight', 'num-of-cylinder
s', 'engine-size', 'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'alfa-romero',
'audi', 'bmw', 'chevrolet', 'dodge', 'honda', 'isuzu', 'jaguar', 'mazda', 'mercedes-benz', 'mercury', 'mitsubishi', 'nissan',
'peugot', 'plymouth', 'porsche', 'renault', 'saab', 'subaru', 'toyota', 'volkswagen', 'volvo', 'diesel', 'gas', 'std', 'turbo',
'convertible', 'hardtop', 'hatchback', 'sedan', 'wagon', '4wd', 'fwd', 'rwd', 'front', 'rear', 'dohc', 'dohcv', 'l', 'ohc', 'oh
cf', 'ohcv', 'rotor', '1bbl', '2bbl', '4bbl', 'idi', 'mfi', 'mpfi', 'spdi', 'spfi', 'price']
(205, 69)
```

In [203]:
```python
X = processed_data[:,:68]
Y = processed_data[:,68:]
print(X.shape , Y.shape)
```

```
(205, 68) (205, 1)
```

In [204]:
```python
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, train_size=0.70 , test_size=0.30)
```

In [205]:
```python
print(X_train.shape , X_test.shape , Y_train.shape ,Y_test.shape)
```

```
(143, 68) (62, 68) (143, 1) (62, 1)
```

In [206]:
```python
def forward_prop(X,w):
    return np.dot(w, X.T)
```

In [254]:
```python
def back_prop(w,res,Y_train ,X_train, lr):
    #print(res.shape,Y_train.T.shape)
    m = res.shape[1]
    diff = res-Y_train.T
    grad = np.reshape(np.sum(X_train.T*(diff)/m, axis=1), (68,1))
    w = w - lr*(grad.T)
    return w

def back_prop_lasso(w,res,Y_train,X_train,lr):
```

In [294]:
```python
def R2(Y , Y_pred):
    y_mean = np.mean(Y)
    return np.sum(np.square(Y_pred-y_mean))/np.sum(np.square(Y-y_mean))
```


In [294]:

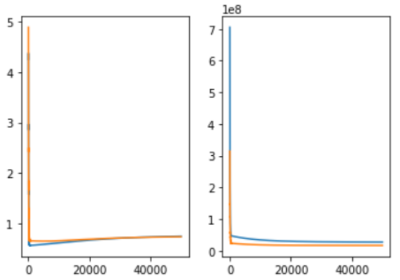
```python
def R2(Y , Y_pred):
    y_mean = np.mean(Y)
    return np.sum(np.square(Y_pred-y_mean))/np.sum(np.square(Y-y_mean))
```


```python
def AdjustedR2(Y , Y_pred,p):
    n = Y.shape[0]
    y_mean = np.mean(Y)
    SSres = np.sum(np.square(Y_pred-Y))/(n-p-1)
    SStot = np.sum(np.square(Y-y_mean))/(n-1)
    return 1-SSres/SStot
```

In [318]:
```python
#without cross validation
w = np.random.randn(1,68) #change 5 according to number of attributes
lr = 0.0000000598
errors_train = []
errors_test = []
R2_train = []
R2_test = []
for i in range(50000):
    m = X_train.shape[1]
    res = forward_prop(X_train,w)
    res_test = forward_prop(X_test,w)
    errors_train.append(np.sum(np.square(res-Y_train.T))/m)
    R2_train.append(R2(Y_train, res))
    if i%1000==0:
        print(errors_train[-1])
    errors_test.append(np.sum(np.square(res_test-Y_test.T))/m)
    R2_test.append(R2(Y_test, res_test))
    w = back_prop(w,res,Y_train,X_train,lr)
ax ,fig = plt.subplots(1,2)
fig[0].plot(R2_train)
fig[0].plot(R2_test)
fig[1].plot(errors_train)
fig[1].plot(errors_test)

print("R2 on training data = ", R2(Y_train, forward_prop(X_train,w)))
print("R2 on test data = ", R2(Y_test, forward_prop(X_test,w)))
```

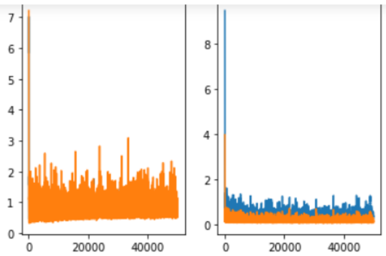
```
705113859.2771521
46288242.858297676
44288113.1165108
42510424.21685141
40930036.14040282
39524701.84334999
38274690.9702568
37162504.801837035
36172623.51102289
35291282.06373185
34506271.516326554
33806762.83019186
33183150.65047616
32626914.78560892
32130497.38091527
31687194.00723212
31291057.087195903
30936810.26074439
30619772.449959293
30335790.523967832
30081179.589266784
29852670.04133925
29647360.61140533
29462676.729009476
29296333.598154932
29146303.45297146
```

```
27725668.76204934
R2 on training data =  0.7482840177632876
R2 on test data =  0.7385614701976351
```



In [319]:
```python
#with cross validation
w = np.random.randn(1,68) #change 5 according to number of attributes
lr = 0.0000000598
errors_train = []
errors_test = []
R2_train = []
R2_test = []
for i in range(50000):
    X_train, X_test, Y_train, Y_test = train_test_split(X,Y, train_size=0.70 , test_size=0.30)
    m = X_train.shape[1]
    res = forward_prop(X_train,w)
    res_test = forward_prop(X_test,w)
    errors_train.append(np.sum(np.square(res-Y_train.T))/m)
    R2_train.append(R2(Y_train, res))
    if i%1000==0:
        print(errors_train[-1])
    errors_test.append(np.sum(np.square(res_test-Y_test.T))/m)
    R2_test.append(R2(Y_test, res_test))
    w = back_prop(w,res,Y_train,X_train,lr)
ax ,fig = plt.subplots(1,2)
fig[0].plot(R2_train)
fig[0].plot(R2_test)
fig[1].plot(errors_train)
fig[1].plot(errors_test)

print("R2 on training data = ", R2(Y_train, forward_prop(X_train,w)))
print("R2 on test data = ", R2(Y_test, forward_prop(X_test,w)))
```

```
945492574.8528987
62781565.66079256
56699259.61267675
52817434.74047402
57142020.572913155
52849849.30881004
45780109.961545974
41354014.0303284
50464132.94841389
43680341.42041526
33453460.810854785
45580186.78937715
41412099.44227666
36503672.87856259
37917202.592694595
38465486.76858988
```



In [320]:
```python
#using sklearn library

from sklearn.linear_model import Lasso ,LinearRegression, Ridge
from sklearn.metrics import mean_squared_error

model = Lasso()
model.fit(X_train, Y_train)
Y_pred1 = model.predict(X_train)
Y_pred2 = model.predict(X_test)
r_sq = model.score(X_train, Y_train)
r_sq2 = model.score(X_test , Y_test)
print("Lasso")
print("R square train: ",r_sq)
print("R square test: ",r_sq2)
print("error_train:", mean_squared_error(Y_train , Y_pred1) )
print("error_test:",mean_squared_error(Y_test , Y_pred2))
print("slopes" , model.coef )
```

```
print("intercept" , model.intercept_)
print("\n")

print("LinearR")
model = LinearRegression()
model.fit(X_train, Y_train)
Y_pred1 = model.predict(X_train)
Y_pred2 = model.predict(X_test)
r_sq = model.score(X_train, Y_train)
r_sq2 = model.score(X_test , Y_test)
print("R square train: ",r_sq)
print("R square test: ",r_sq2)
print("error_train:", mean_squared_error(Y_train , Y_pred1) )
print("error_test:",mean_squared_error(Y_test , Y_pred2))
print("slopes" , model.coef_)
print("intercept" , model.intercept_)
print("\n")

print("Ridge")
model = Ridge()
model.fit(X_train, Y_train)
Y_pred1 = model.predict(X_train)
Y_pred2 = model.predict(X_test)
r_sq = model.score(X_train, Y_train)
r_sq2 = model.score(X_test , Y_test)
print("R square train: ",r_sq)
print("R square test: ",r_sq2)
print("error_train:", mean_squared_error(Y_train , Y_pred1) )
print("error_test:",mean_squared_error(Y_test , Y_pred2))
print("slopes" , model.coef_)
print("intercept" , model.intercept_)
```

```
Lasso
R square train:  0.9558735679723899
R square test:  0.8371005169683412
error_train: 2304443.5804555602
error_test: 12810522.500245448
slopes [-5.19524094e+00  8.56098162e-02  9.75188582e+00  3.07339648e+02
 -1.90585027e+02  8.09482528e+02 -2.78453071e+02  6.96273821e+00
 -9.56754839e+02  1.46839724e+02 -5.68956695e+03 -2.01569208e+03
 -6.78265060e+02 -1.54895740e+01  3.66167600e+00  1.82616056e+02
 -3.41061142e+01  5.49779749e+03 -3.56508104e+02  7.86768543e+03
 -2.82636685e+03 -2.72003348e+03 -2.10458800e+03  2.14219522e+03
  0.00000000e+00  7.36861925e+02  8.76468874e+03  0.00000000e+00
 -2.88216669e+03  1.03564997e+03 -0.00000000e+00 -2.50729220e+03
  4.98671841e+03 -1.14520970e+02  4.94901604e+02 -0.00000000e+00
 -5.08324852e+01  3.56791744e+02 -4.11205767e+02  8.39739475e+03
 -2.65696589e-09 -1.42218244e+03  0.00000000e+00  2.06517829e+03
 -1.15556845e+03 -8.42914668e+02  4.56399855e+02 -0.00000000e+00
 -9.69451788e+02 -0.00000000e+00  7.78391003e+02 -8.58301150e+03
  2.10349374e-09 -9.42284454e+01 -1.38171633e+04 -0.00000000e+00
  3.60461175e+03  5.01937655e+03 -2.50237538e+03  8.70885562e+03
  0.00000000e+00  6.98612207e+02 -1.58315369e+03  3.45968330e+02
 -9.36521534e+02  0.00000000e+00 -9.03068210e+02  0.00000000e+00]
intercept [-37860.64282611]


LinearR
R square train:  0.9560555983800338
R square test:  0.8282389446235208
error_train: 2294937.2871735576
error_test: 13507402.378549373
slopes [[-1.10911276e+01  4.54715867e-03  6.82010241e-01  3.18653003e+02
  -1.94764983e+02  8.52226094e+02 -3.03041344e+02  7.12223666e+00
  -8.50340751e+02  1.46969350e+02 -5.75655153e+03 -1.91784222e+03
   4.50852687e+03 -1.20399412e+03 -4.18662516e-01 -2.37263462e+02
  -9.06987337e+02 -4.24449091e+02 -1.27419430e+03  4.20195318e+03
  -4.20195318e+03 -6.72751819e+02  6.72751819e+02  1.91113653e+03
  -1.27114209e+03 -9.77286448e+02  3.82611464e+02 -4.53194561e+01
  -9.94502084e+02  7.64922431e+01  9.18009841e+02 -4.68293847e+03
   4.68293847e+03 -1.93794054e+02 -1.39469609e+04 -6.41311608e+02
   3.56361407e+03  4.44567501e+03 -2.74370019e+03  9.51647767e+03
   1.04773378e+02  5.96676189e+02 -2.19958663e+03  4.20195318e+03
  -1.34841191e+03 -1.05702887e+02 -1.24970132e+03  0.00000000e+00]]
intercept [-37800.03605439]


Ridge
R square train:  0.9395376041125763
R square test:  0.9039155282451415
error_train: 3157567.3277765964
error_test: 7556146.06278765
slopes [[ 1.07588185e+02 -3.02668778e+00  7.80010690e+01  2.58161358e+02
  -1.40289881e+02  6.51383786e+02 -1.75739353e+02  5.25428310e+00
  -1.26833001e+03  1.72993370e+02 -4.77196437e+03 -2.36966793e+03
  -2.05487836e+02 -4.66556426e+01  3.62009152e+00  1.37529821e+02
  -6.96107464e+01  1.86793523e+03  2.58339929e+02  6.22251023e+03
  -1.32199099e+03 -2.16572373e+03 -1.13756227e+03  1.45199783e+03
  -8.20059905e+02  4.86904379e+02  3.29275595e+03  0.00000000e+00
  -2.19191437e+03 -1.24577028e+02 -1.36391569e+03 -2.17343204e+03
   2.16105816e+03 -2.98802271e+02 -3.34770164e+02 -1.19074192e+03
  -9.98213730e+02 -2.39461526e+02 -1.38033608e+03  7.62845740e+02
  -7.62845740e+02 -1.05957976e+03  1.05957976e+03  2.11401072e+03
  -6.87761980e+02 -9.69554247e+02  1.85538041e+02 -6.42232533e+02
  -7.65288167e+02 -4.63038431e+02  1.22832660e+03 -4.20219676e+03
   4.20219676e+03 -1.05865936e+03 -4.23611372e+03 -1.36391569e+03
   1.43226525e+03  3.01145484e+03 -1.39445011e+03  3.60941878e+03
```

Q8. Association Rule Mining
Use the Apriori algorithm to generate association rules.

Tool Used : Weka
Theory :
Apriori property: All nonempty subsets of a frequent itemset must also be frequent.
Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases.
Some metrics used in the method are:

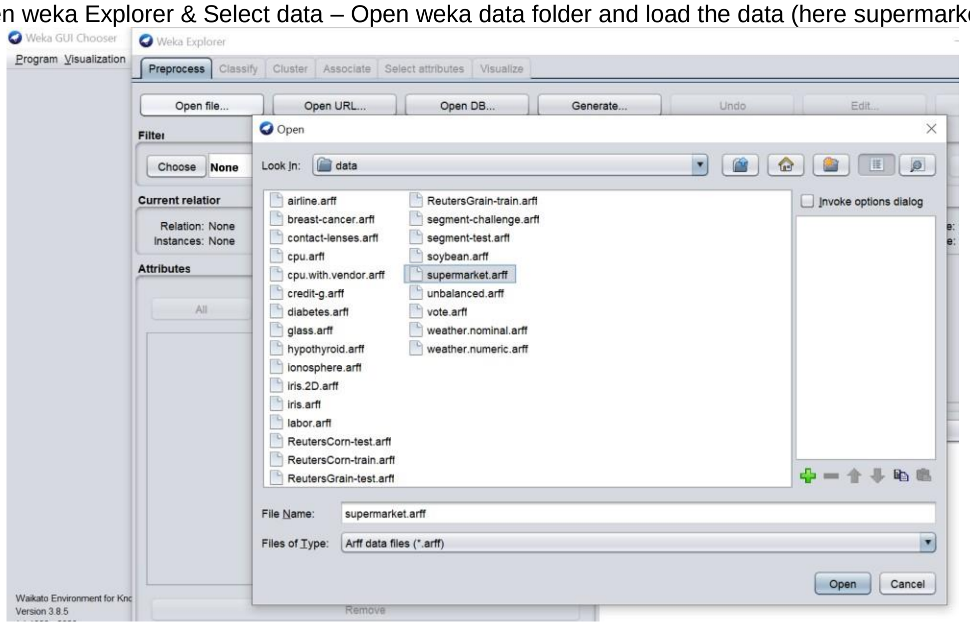
a. Support tells us how frequently an itemset has been bought.
Support(A) = Transactions(A) / Total Transactions
b. Confidence will tell us how confident (based on our data) we can be that an item will be purchased, given that another item has been purchased.
Confidence(A→B) = Support(A U B) / Support(A)

Apriori performs the following sequence of calculations:

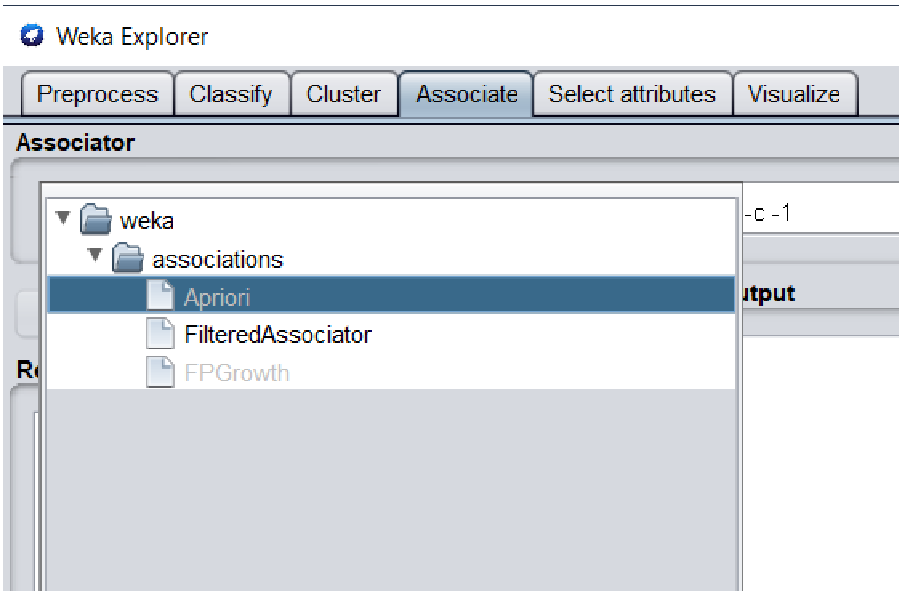
a. Calculate support for item sets of size 1.
b. Apply the minimum support threshold and prune item sets that do not meet the threshold.
c. Move on to item sets of size 2 and repeat steps one and two.
d. Continue the same process until no additional item sets satisfying the minimum threshold can be found.
e. For each frequent(not pruned) itemset l, generate all nonempty subsets of l.
a. For every nonempty subset s of l, output the rule “s →(l – s)” if support count(l)/support count(s) > min conf, where min conf is the minimum confidence threshold.

Steps :

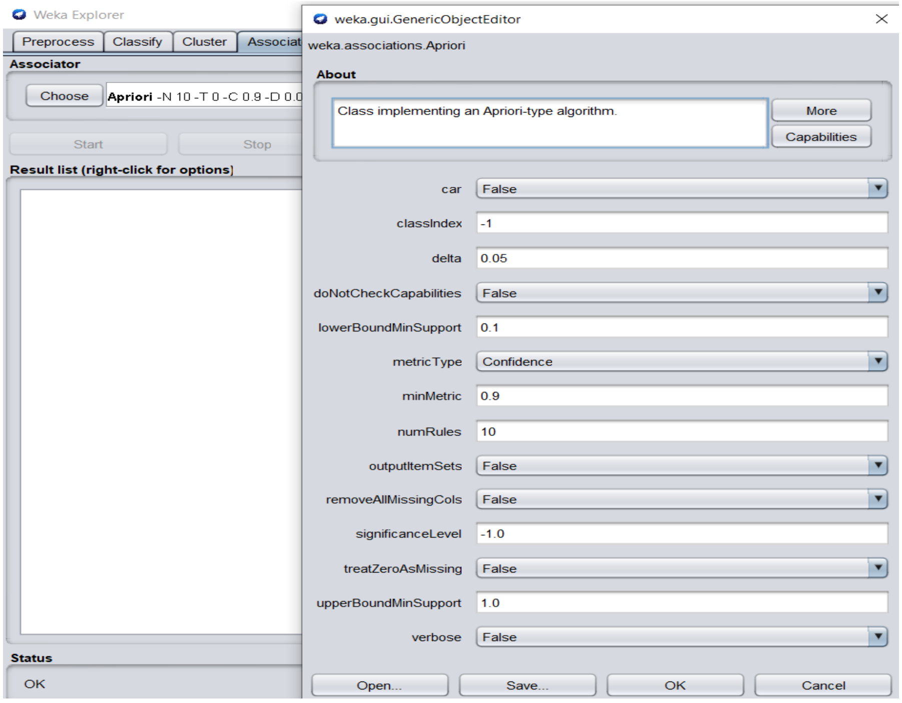
1. Open weka Explorer & Select data – Open weka data folder and load the data (here supermarket.arff taken)



4. Now from the top dialogue box select: Associate→choose→weka→associations→Apriori.



Now, Click on the name of algorithm to check the status and set parameters manually.



5. Click on the Start button. Desired classification will come out.

```
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
```

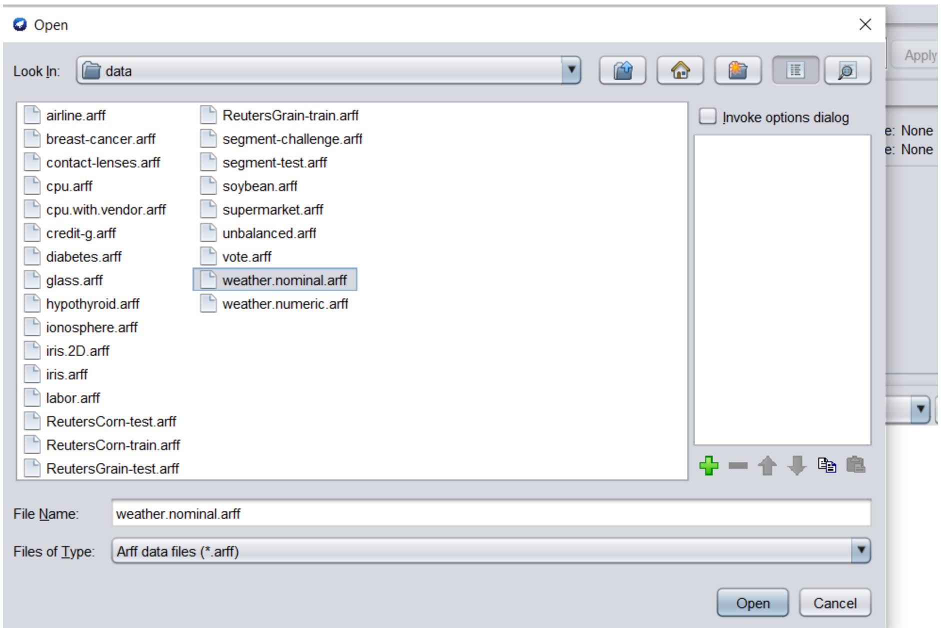
```
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

Q10. Use Bayesian Learning for classification

Theory :
Bayesian classification is based on Bayes' 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.
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)

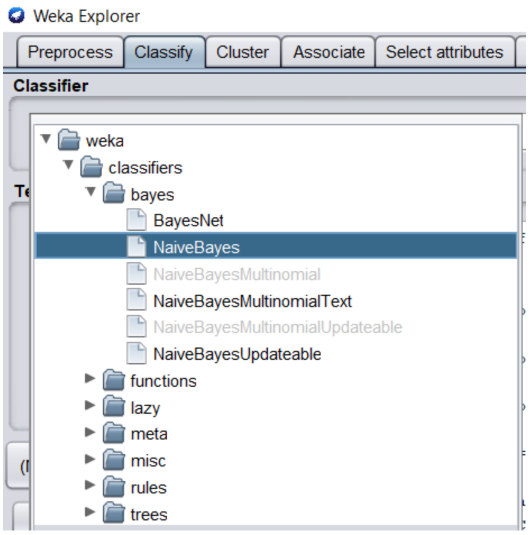
Steps :

1. Open weka Explorer & Select data – Open weka data folder and load the data (here weather_nominal.arff taken)



2. Now in the cluster option of top dialogue menu select

:Classify→choose→weka→classifiers→bayes→NaiveBayes.



Now, Click on the name of algorithm to check the status and set parameters manually.

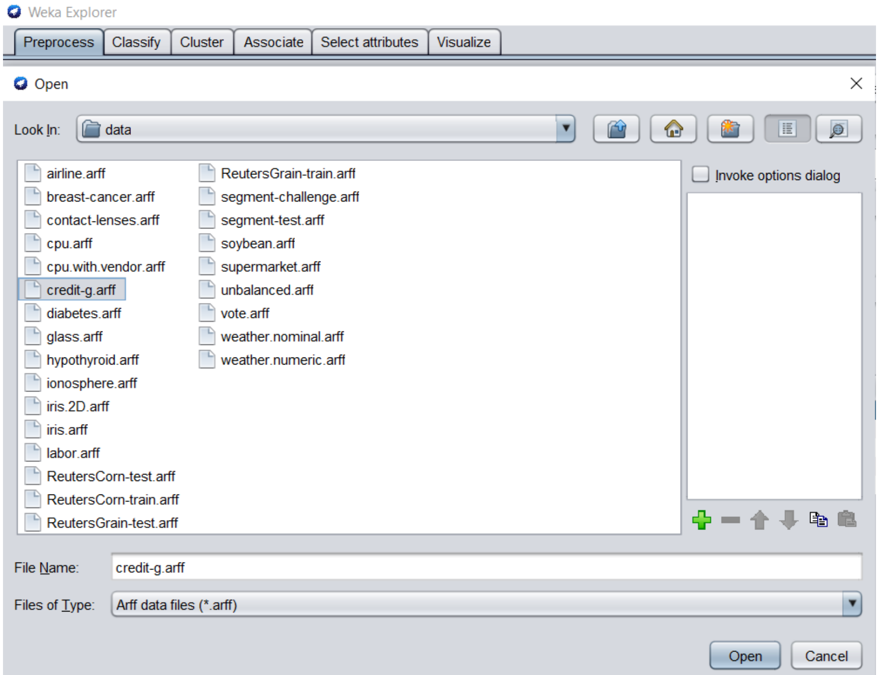
3. Click on the Start button. Results will be displayed .

Classifier output

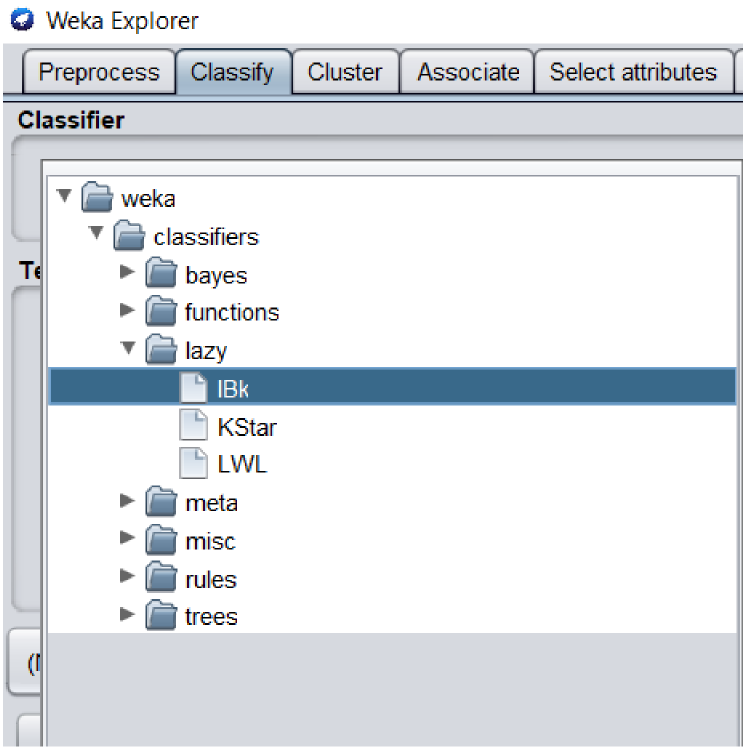
```
Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances          13               92.8571 %
Incorrectly Classified Instances         1                7.1429 %
Kappa statistic                          0.8372
Mean absolute error                      0.2917
Root mean squared error                  0.3392
Relative absolute error                 62.8233 %
Root relative squared error             70.7422 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.200    0.900      1.000    0.947      0.849    0.922     0.947     yes
                 0.800    0.000    1.000      0.800    0.889      0.849    0.911     0.911     no
Weighted Avg.    0.929    0.129    0.936      0.929    0.926      0.849    0.918     0.934

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = yes
 1 4 | b = no
```

Q11. KNN algorithm

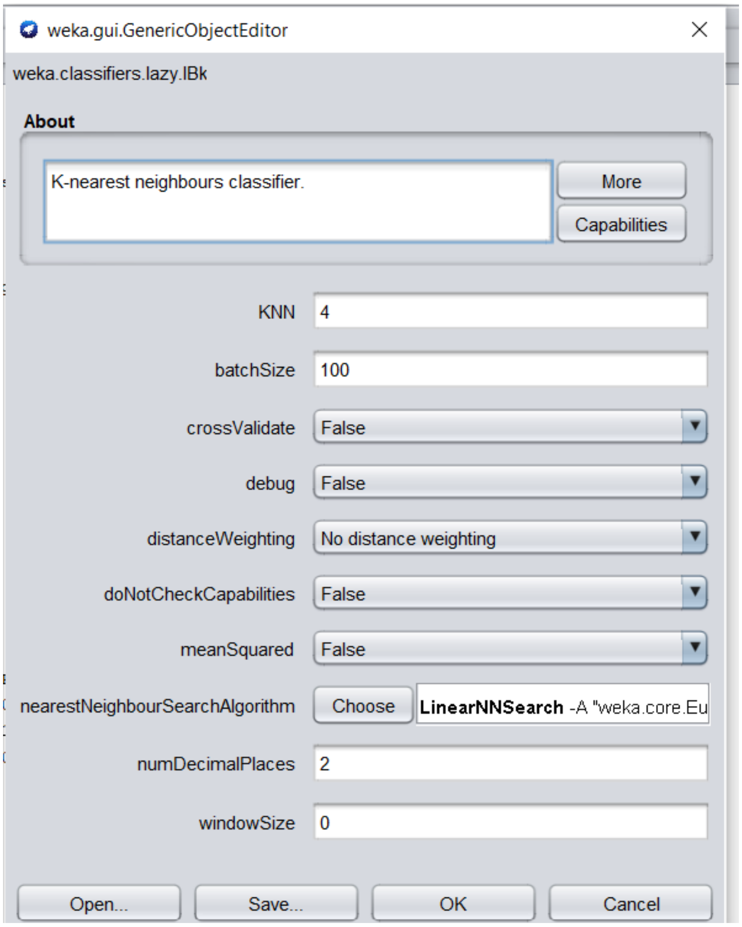
1. Open weka Explorer & Select data – Open weka data folder and load the data (here credit_g.arff taken)



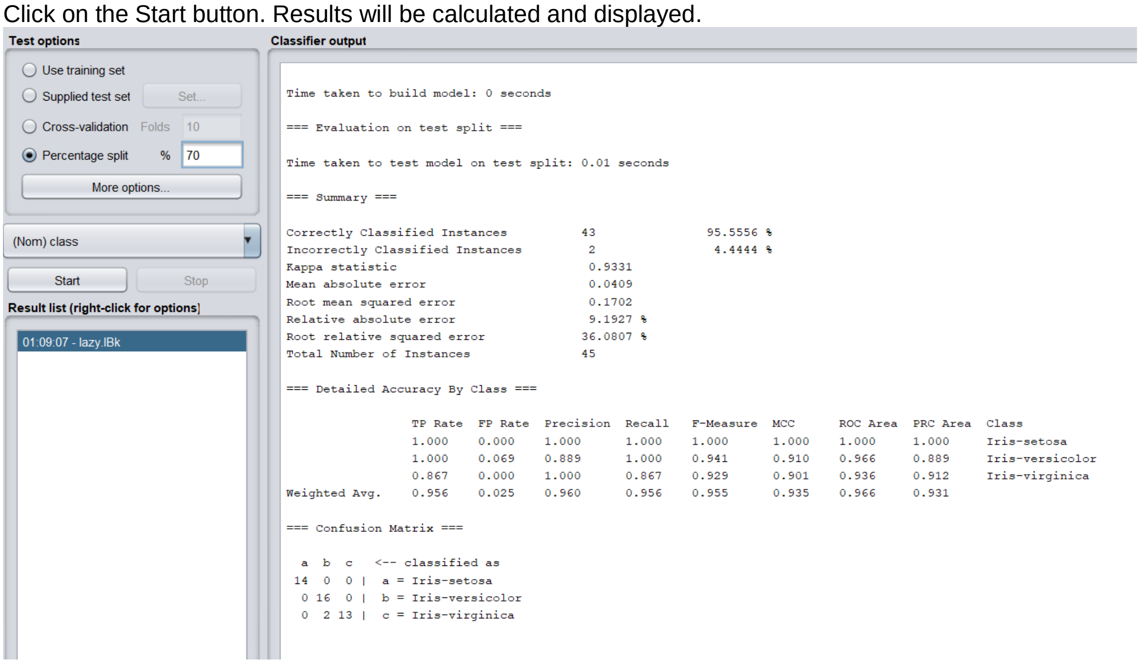
2.Now from the top dialogue box select: Classify→choose→weka→lazy→IBk.



Now, Click on the name of algorithm to check the status and set parameters manually.



3. Click on the Start button. Results will be calculated and displayed.



Q10. Use Bayesian Learning for classification

Theory :

Bayesian classification is based on Bayes' 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.

There are two types of probabilities –

Posterior Probability [P(H/X)]

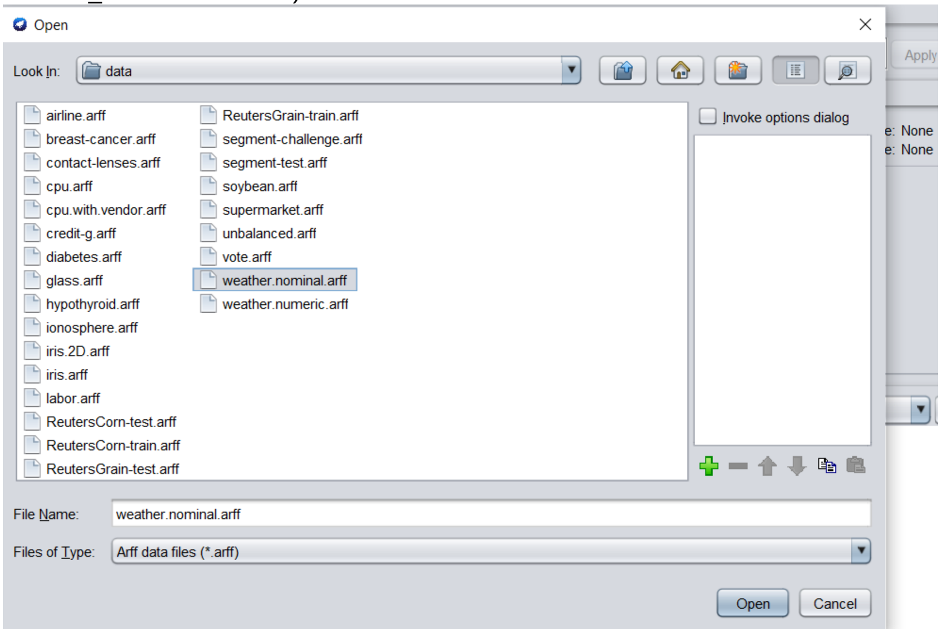
Prior Probability [P(H)]

where X is data tuple and H is some hypothesis.

According to Bayes' Theorem, P(H/X)= P(X/H)P(H) / P(X)

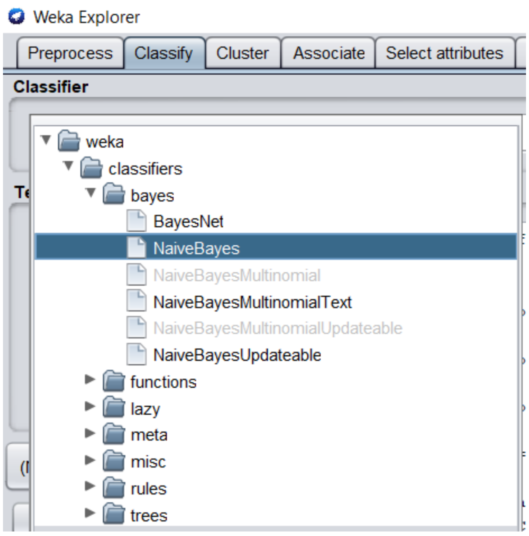
Steps :

1. Open weka Explorer & Select data – Open weka data folder and load the data (here weather_nominal.arff taken)



2. Now in the cluster option of top dialogue menu select

:Classify→choose→weka→classifiers→bayes→NaiveBayes.



Now, Click on the name of algorithm to check the status and set parameters manually.

3. Click on the Start button. Results will be displayed .

Classifier output

```
Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances          13               92.8571 %
Incorrectly Classified Instances         1                7.1429 %
Kappa statistic                          0.8372
Mean absolute error                      0.2917
Root mean squared error                  0.3392
Relative absolute error                 62.8233 %
Root relative squared error             70.7422 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.200    0.900      1.000    0.947      0.849    0.922     0.947     yes
                 0.800    0.000    1.000      0.800    0.889      0.849    0.911     0.911     no
Weighted Avg.    0.929    0.129    0.936      0.929    0.926      0.849    0.918     0.934

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = yes
 1 4 | b = no
```

Q12. Implement K-means clustering algorithm

Tool Used: Weka

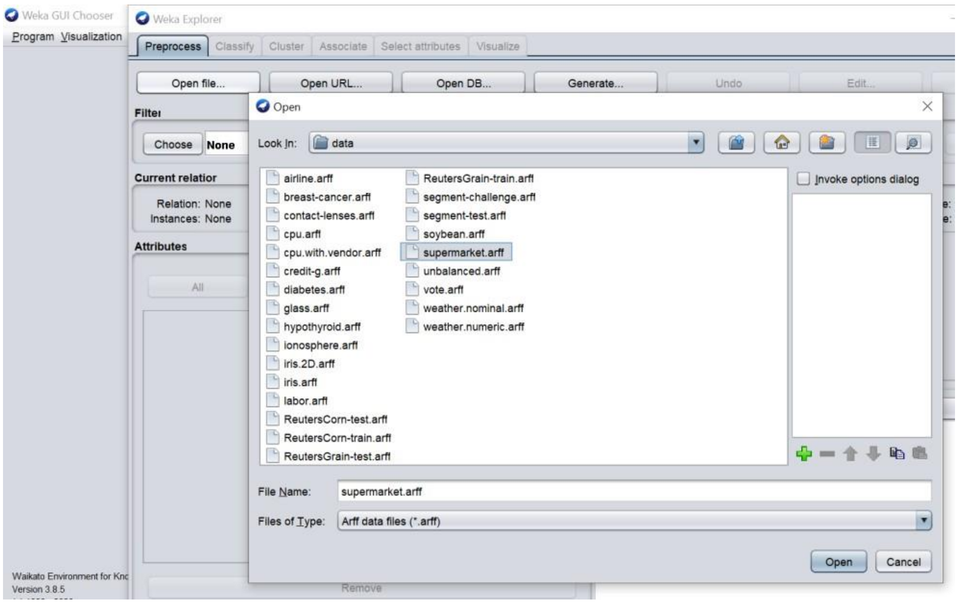
Theory:

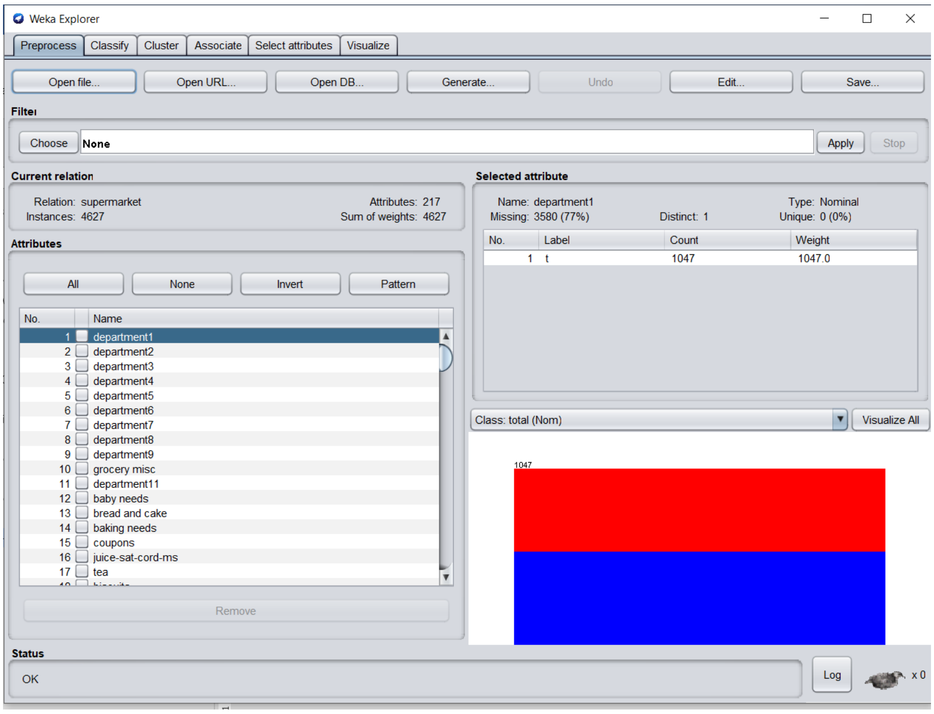
Clustering can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different.In other words, we try to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance. The decision of which similarity measure to use is application-specific.

K-means algorithm is an iterative algorithm that tries to partition the dataset into *K* pre-defined distinct non-overlapping subgroups (clusters) where each datapoint belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

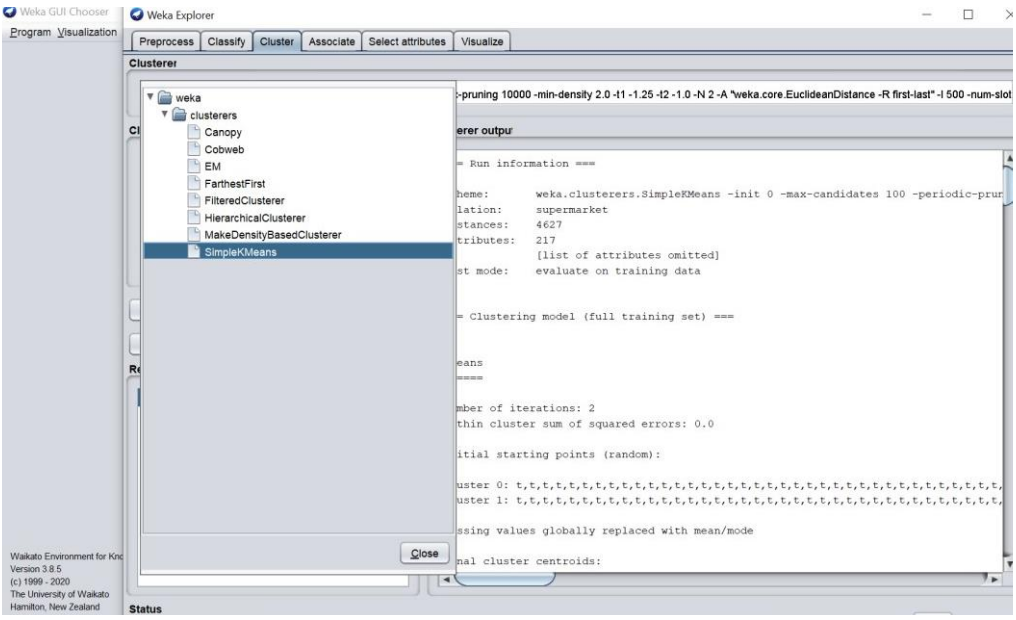
Steps :

1. Open weka Explorer & Select data – Open weka data folder and load the data (here supermarket.arff taken)

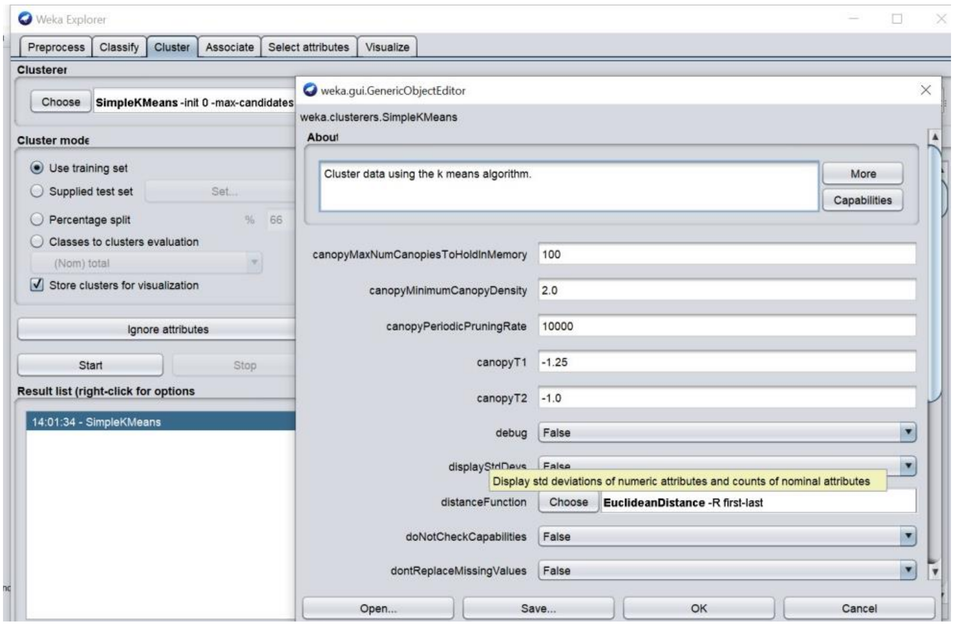




2. Now in the cluster option of top dialogue menu select :Cluster→choose→weka→clusterers→SimpleKMeans.

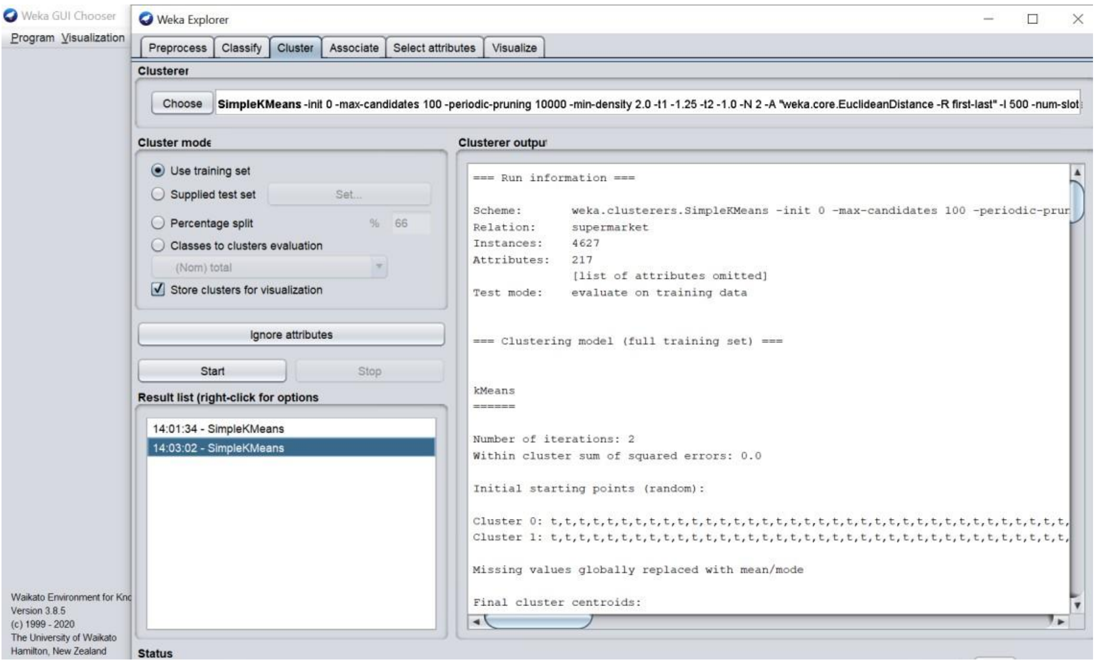


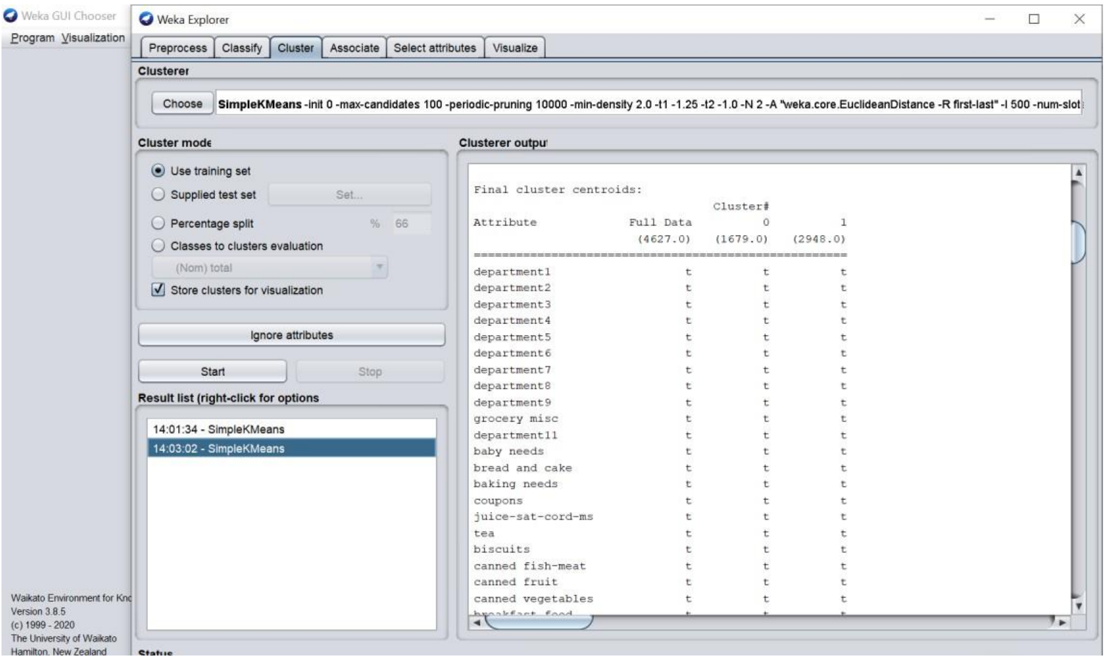
Now, Click on the name of algorithm to check the status and set parameters manually.



3. Click on the Start button.

Desired classification will come out.


Weka GUI Chooser
Program Visualization
Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
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-slot
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
% 66
Classes to clusters evaluation
(Nom) total
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options
14:01:34 - SimpleKMeans
14:03:02 - SimpleKMeans
Clusterer output
=== Run information ===
Scheme: weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-prun
Relation: supermarket
Instances: 4627
Attributes: 217
[list of attributes omitted]
Test mode: evaluate on training data
=== Clustering model (full training set) ===
kMeans
======
Number of iterations: 2
Within cluster sum of squared errors: 0.0
Initial starting points (random):
Cluster 0: t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,
Cluster 1: t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,
Missing values globally replaced with mean/mode
Final cluster centroids:
Waikato Environment for Knc
Version 3.8.5
(c) 1999 - 2020
The University of Waikato
Hamilton, New Zealand
Status


Weka GUI Chooser
Program Visualization
Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
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-slot
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
% 66
Classes to clusters evaluation
(Nom) total
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options
14:01:34 - SimpleKMeans
14:03:02 - SimpleKMeans
Clusterer output
Final cluster centroids:
Cluster#
Attribute Full Data 0 1
(4627.0) (1679.0) (2948.0)
department1 t t t
department2 t t t
department3 t t t
department4 t t t
department5 t t t
department6 t t t
department7 t t t
department8 t t t
department9 t t t
grocery misc t t t
department11 t t t
baby needs t t t
bread and cake t t t
baking needs t t t
coupons t t t
juice-sat-cord-ms t t t
tea t t t
biscuits t t t
canned fish-meat t t t
canned fruit t t t
canned vegetables t t t
Waikato Environment for Knc
Version 3.8.5
(c) 1999 - 2020
The University of Waikato
Hamilton, New Zealand