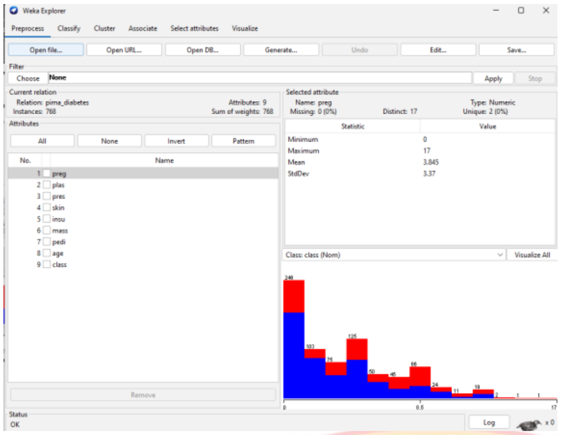
6)Demonstrate ZeroR technique on Iris dataset (by using necessary preprocessing technique(s)) and share your observations (using WEKA)

Process:

Load the any dataset into the weka.



Current relation
Relation: pima_diabetes Attributes: 9
Instances: 768 Sum of weights: 768

Selected attribute
Name: preg Type: Numeric
Missing: 0 (0%) Distinct: 17 Unique: 2 (0%)

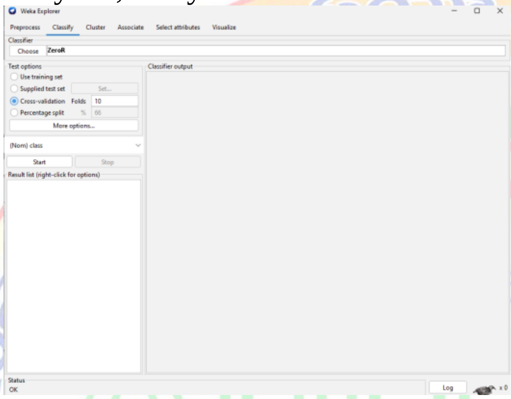
Statistic	Value
Minimum	0
Maximum	17
Mean	3.845
StdDev	3.37

No.	Name
1	preg
2	plas
3	pres
4	skin
5	insu
6	mass
7	pedi
8	age
9	class

Class: class (Nom)

Status
OK

click on the Classify tab, and you would see the following screen



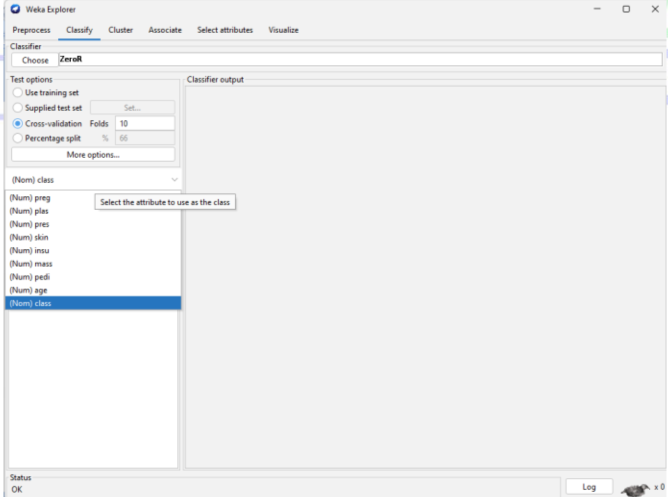
Classifier
Choose ZeroR

Test options
○ Use training set
○ Supplied test set
● Cross-validation Folds 10
○ Percentage split % 66

(Nom) class

Status
OK

the default class option for the output class



Classifier
Choose ZeroR

Test options
○ Use training set
○ Supplied test set
● Cross-validation Folds 10
○ Percentage split % 66

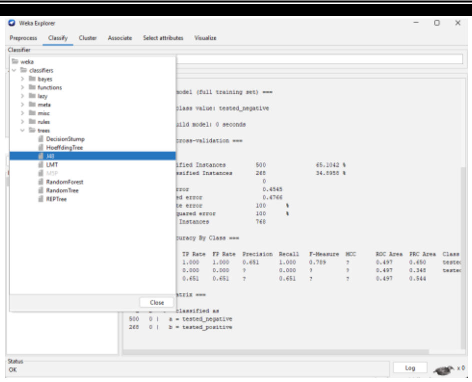
(Nom) class

(Num) preg
(Num) plas
(Num) pres
(Num) skin
(Num) insu
(Num) mass
(Num) pedi
(Num) age
(Nom) class

Select the attribute to use as the class

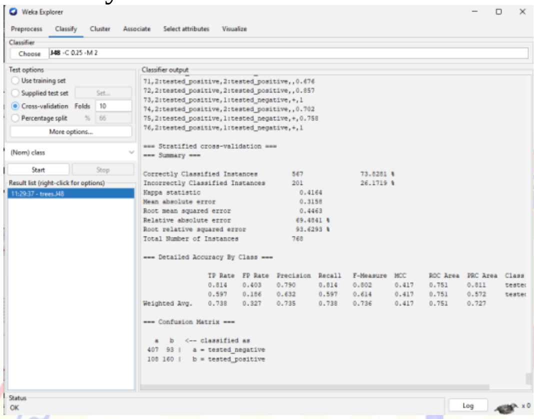
Status
OK

Click on the Choose button and select the following classifier − weka→classifiers>trees>J48



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

class value: tested_negative

uild model: 0 seconds

cross-validation ===

ified Instances        500               65.1042 %
ssified Instances      268               34.8958 %
                         0
rror                     0.4545
ed error                 0.4766
te error               100      %
quared error           100      %
 Instances             768

curacy By Class ===

 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
 1.000    1.000    0.651      1.000    0.789      ?        0.497     0.650     testec
 0.000    0.000    ?          0.000    ?          ?        0.497     0.348     testec
 0.651    0.651    ?          0.651    ?          ?        0.497     0.544

atrix ===

            classified as
 500   0 |   a = tested_negative
 268   0 |   b = tested_positive
```

Click on the Start button to start the classification process. After a while, the classification results would be presented on your screen as shown here



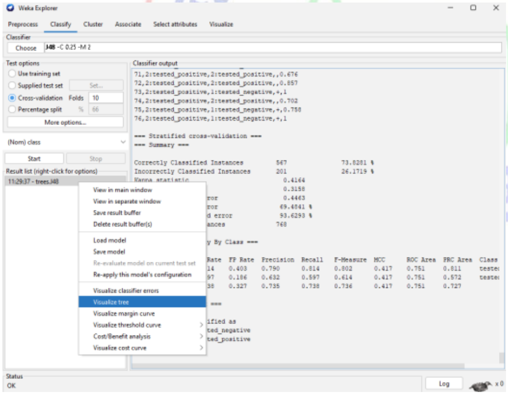
Classifier: J48 -C 0.25 -M 2

Folds: 10

%: 66

```
71,2:tested_positive,2:tested_positive,,0.676
72,2:tested_positive,2:tested_positive,,0.857
73,2:tested_positive,1:tested_negative,+,1
74,2:tested_positive,2:tested_positive,,0.702
75,2:tested_positive,1:tested_negative,+,0.758
76,2:tested_positive,1:tested_negative,+,1

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

Correctly Classified Instances         567               73.8281 %
Incorrectly Classified Instances       201               26.1719 %
Kappa statistic                          0.4164
Mean absolute error                      0.3158
Root mean squared error                  0.4463
Relative absolute error                 69.4841 %
Root relative squared error             93.6293 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.814    0.403    0.790      0.814    0.802      0.417    0.751     0.811     teste
                 0.597    0.186    0.632      0.597    0.614      0.417    0.751     0.572     teste
Weighted Avg.    0.738    0.327    0.735      0.738    0.736      0.417    0.751     0.727

=== Confusion Matrix ===

   a   b   <-- classified as
 407  93 |   a = tested_negative
 108 160 |   b = tested_positive
```

Select Visualize tree to get a visual representation of the traversal tree as seen in the screenshot below

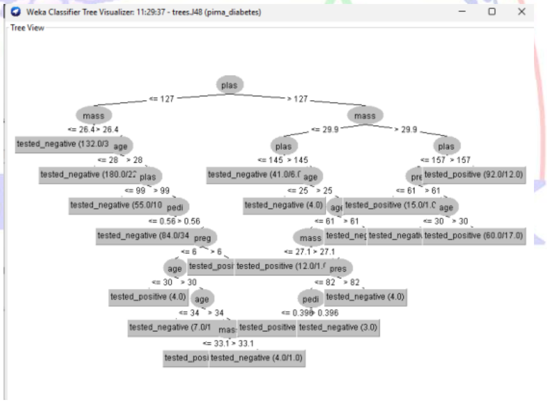


Classifier: J48 -C 0.25 -M 2

Folds: 10

%: 66

```
71,2:tested_positive,2:tested_positive,,0.676
72,2:tested_positive,2:tested_positive,,0.857
73,2:tested_positive,1:tested_negative,+,1
74,2:tested_positive,2:tested_positive,,0.702
75,2:tested_positive,1:tested_negative,+,0.758
76,2:tested_positive,1:tested_negative,+,1

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

Correctly Classified Instances         567               73.8281 %
Incorrectly Classified Instances       201               26.1719 %
Kappa statistic                          0.4164
                                         0.3158
ror                                      0.4463
ror                                     69.4841 %
d error                                 93.6293 %
ances                                  768

y By Class ===

Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
14    0.403    0.790      0.814    0.802      0.417    0.751     0.811     teste
97    0.186    0.632      0.597    0.614      0.417    0.751     0.572     teste
38    0.327    0.735      0.738    0.736      0.417    0.751     0.727

 ===

ified as
ted_negative
ted_positive
```

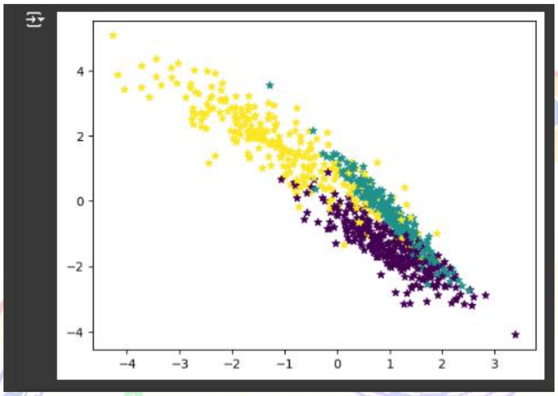


7)Write a program of Naive Bayesian classification using Python programming language

Program:

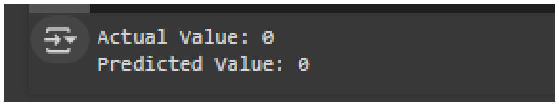
```
from sklearn.datasets import make_classification
        X, y = make_classification(
          n_features=6,
          n_classes=3,
          n_samples=800,
          n_informative=2,
          random_state=1,
          n_clusters_per_class=1,
      )
      import matplotlib.pyplot as plt
      plt.scatter(X[:, 0], X[:, 1], c=y, marker="*");
```

Output:



```
from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.33, random_state=125
      )
from sklearn.naive_bayes import GaussianNB
      model = GaussianNB()
      model.fit(X_train, y_train)
      predicted = model.predict([X_test[6]])
      print("Actual Value:", y_test[6])
      print("Predicted Value:", predicted[0])
```

Output:

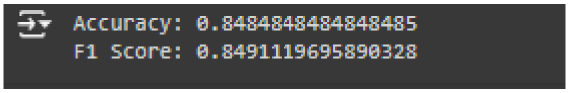


```
Actual Value: 0
Predicted Value: 0
```

```
from sklearn.metrics import (
          accuracy_score,
          confusion_matrix,
          ConfusionMatrixDisplay,
          f1_score,
      )
y_pred = model.predict(X_test)
```

```
        accuray = accuracy_score(y_pred, y_test)
        f1 = f1_score(y_pred, y_test, average="weighted")
        print("Accuracy:", accuray)
        print("F1 Score:", f1)
```

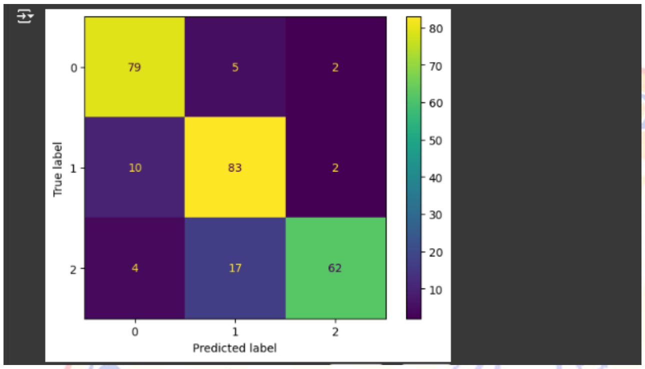
Output:



```
Accuracy: 0.8484848484848485
F1 Score: 0.8491119695890328
```

```
labels = [0,1,2]
        cm = confusion_matrix(y_test, y_pred, labels=labels)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
        disp.plot();
```

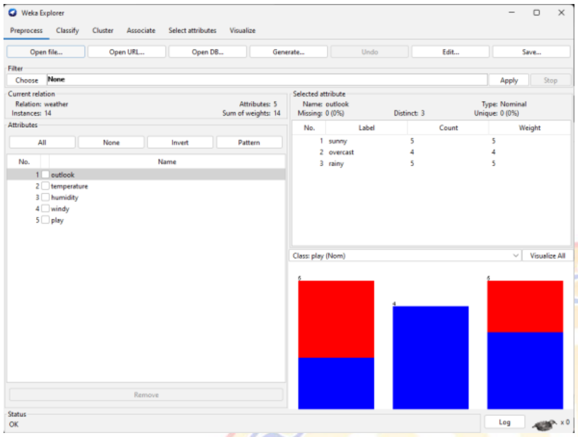
Output:



8) Demonstrate performing clustering of data sets Load each dataset into Weka and run simple k-means clustering algorithm with different values of k (number of desired clusters). Study the clusters formed. Observe the sum of squared errors and centroids, and derive insights. Explore other clustering techniques available in Weka. Explore visualization features of Weka to visualize the clusters. Derive interesting insights and explain

Process:

Load the any dataset into the weka.



Filter: Choose None

Current relation
Relation: weather
Instances: 14
Attributes: 5
Sum of weights: 14

Attributes

No.	Name
1	outlook
2	temperature
3	humidity
4	windy
5	play

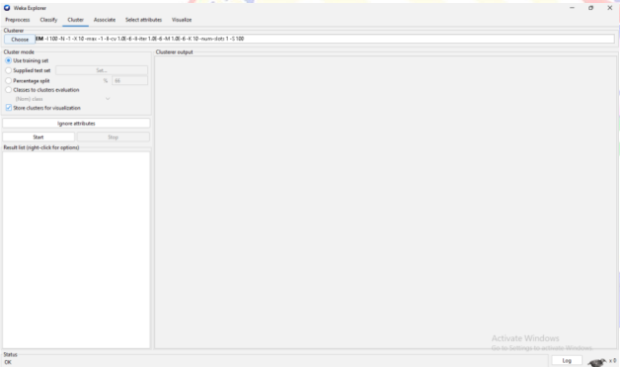
Selected attribute
Name: outlook
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	sunny	5	5
2	overcast	4	4
3	rainy	5	5

Class: play (Nom)

Status
OK

Click on cluster tab and Choose k-mean and select use training set test option.



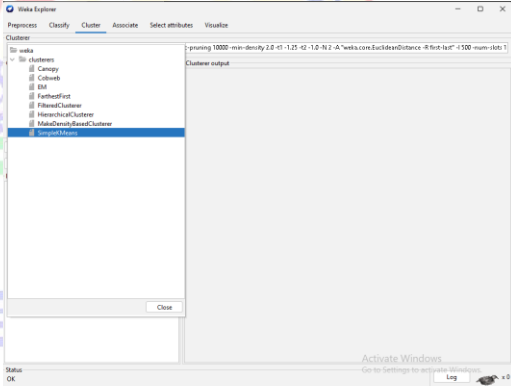
Clusterer: Choose EM -I 100 -N -1 -X 10 -max -1 -ll-cv 1.0E-6 -ll-iter 1.0E-6 -M 1.0E-6 -K 10 -num-slots 1 -S 100

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)

Clusterer output

Status
OK

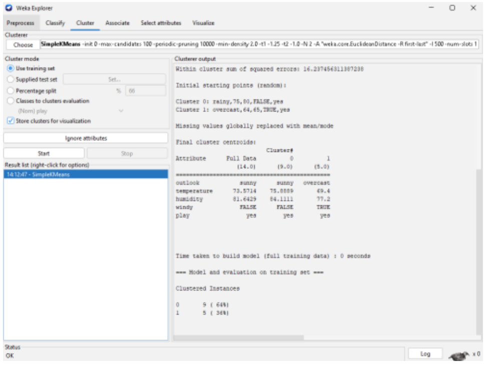


-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1

Clusterer output

Status
OK

Click on start button



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-slots 1

Cluster mode
● Use training set
○ Supplied test set
○ Percentage split % 66
○ Classes to clusters evaluation
(Nom) play
☑ Store clusters for visualization

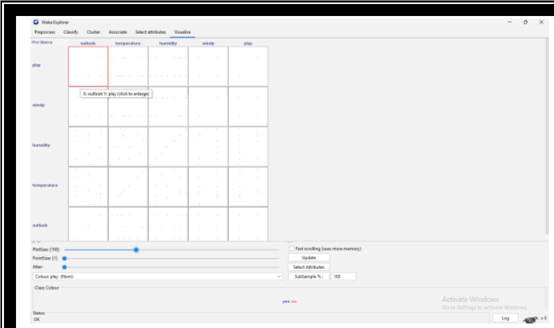
Result list (right-click for options)
16:12:47 - SimpleKMeans

Clusterer output

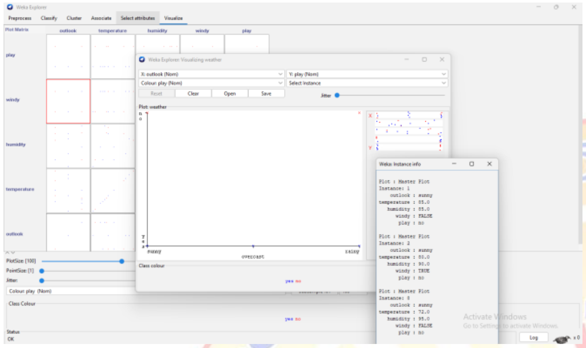
```
Within cluster sum of squared errors: 16.237456311387238

Initial starting points (random):

Cluster 0: rainy,75,80,FALSE,yes
Cluster 1: overcast,64,65,TRUE,yes

Missing values globally replaced with mean/mode

Final cluster centroids:
                         Cluster#
Attribute     Full Data         0          1
                 (14.0)     (9.0)      (5.0)
============================================
outlook           sunny     sunny   overcast
temperature     73.5714   75.8889       69.4
humidity        81.6429   84.1111       77.2
windy             FALSE     FALSE       TRUE
play                yes       yes        yes




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

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

Clustered Instances

0       9 ( 64%)
1       5 ( 36%)
```

Status
OK

To open Visualization screen, click ‘Visualize’ tab



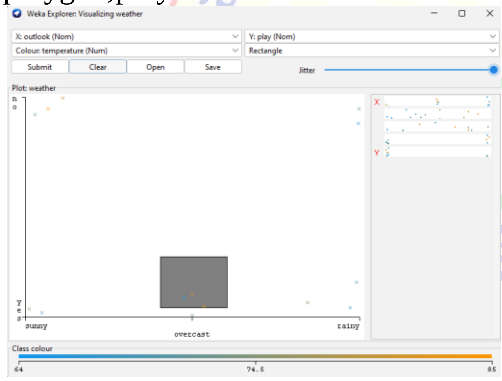
Select Instance. Click on an individual data point. It brings up a window listing attributes of the point. If more than one point will appear at the same location, more than one set of attributes will be shown



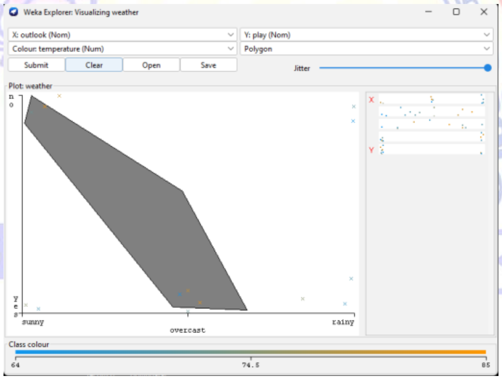
X: outlook (Nom)
Y: play (Nom)
Colour: play (Nom)
Select Instance

```
Plot : Master Plot
Instance: 1
    outlook : sunny
temperature : 85.0
   humidity : 85.0
      windy : FALSE
       play : no

Plot : Master Plot
Instance: 2
    outlook : sunny
temperature : 80.0
   humidity : 90.0
      windy : TRUE
       play : no

Plot : Master Plot
Instance: 8
    outlook : sunny
temperature : 72.0
   humidity : 95.0
      windy : FALSE
       play : no
```

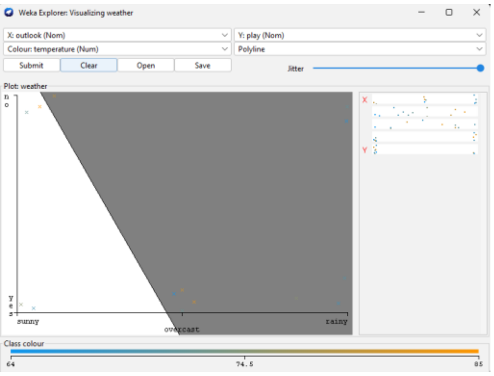
Rectangle .You can create a rectangle by dragging it around the point.do the same process for polygon,polyline



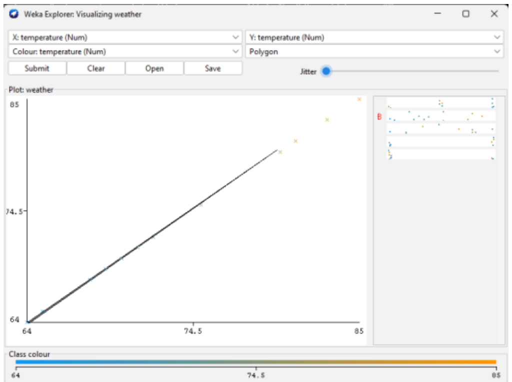
X: outlook (Nom)
Y: play (Nom)
Colour: temperature (Num)
Rectangle



X: outlook (Nom)
Y: play (Nom)
Colour: temperature (Num)
Polygon



X: outlook (Nom)
Y: play (Nom)
Colour: temperature (Num)
Polyline

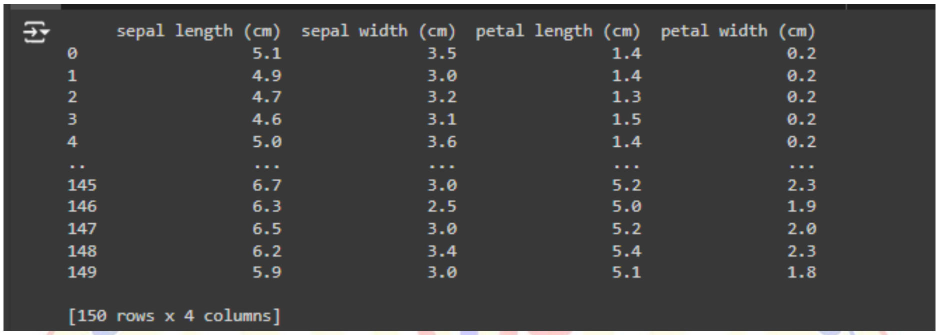


X: temperature (Num)
Y: temperature (Num)
Colour: temperature (Num)
Polygon

9) Write a program of cluster analysis using simple k-means algorithm Python programming language.

PROGRAMS:

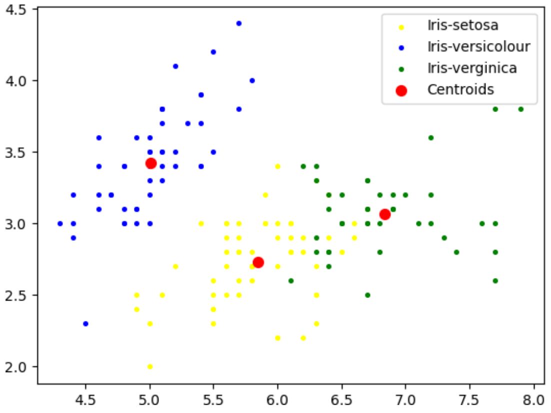
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.datasets
import seaborn as sns
from sklearn.cluster import KMeans
rom sklearn.datasets import load_iris # Import the load_iris function directly
iris = load_iris() # Call load_iris() directly
Data = pd.DataFrame(iris.data, columns=iris.feature_names)
print(Data)
```



```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
```

```
x = Data.iloc[:, 0:3].values
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=100, n_init=10, random_state=0) y_means
= kmeans.fit_predict(x)
plt.scatter(x[y_means==0,0],x[y_means==0,1],s=7,c='yellow',label='Iris-setosa') # Changed y_kmeans to
y_means
plt.scatter(x[y_means==1,0],x[y_means==1,1],s=7,c='blue',label='Iris-versicolour') # Changed y_kmeans
to y_means
plt.scatter(x[y_means==2,0],x[y_means==2,1],s=7,c='green',label='Iris-verginica')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=50,c='red',label='Centroids)
plt.legend()
```

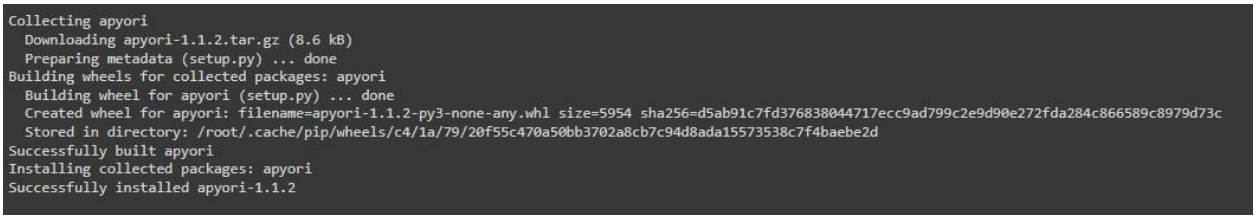
OUTPUT:



10) Write a Python program to generate frequent item sets / association rules using Apriori algorithm.

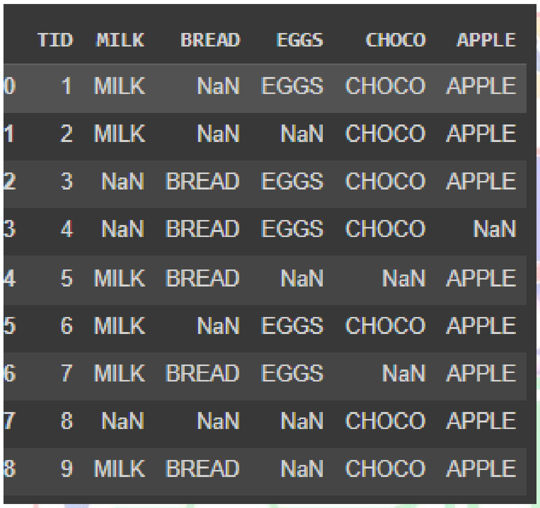
PROGRAMS:

pip install apyori



```
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5954 sha256=d5ab91c7fd376838044717ecc9ad799c2e9d90e272fda284c866589c8979d73c
  Stored in directory: /root/.cache/pip/wheels/c4/1a/79/20f55c470a50bb3702a8cb7c94d8ada15573538c7f4baebe2d
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

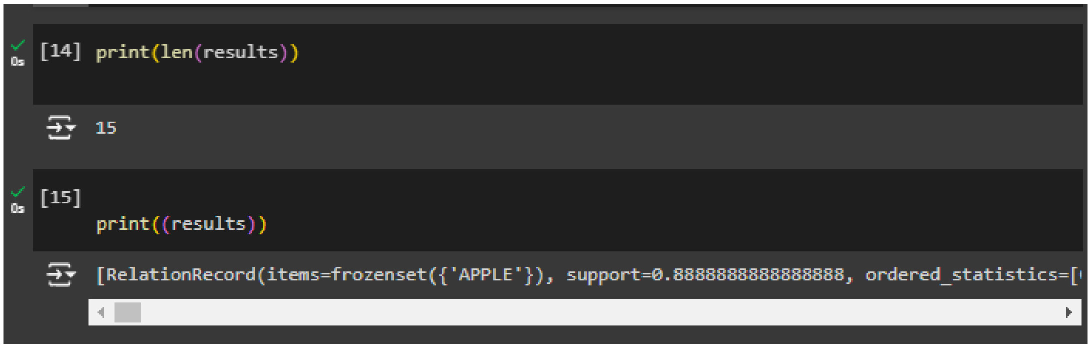
```
import numpy as np
import pandas as pd
from apyori import apriori
data = pd.read_csv('/content/trans_db.csv')
data
```



	TID	MILK	BREAD	EGGS	CHOCO	APPLE
0	1	MILK	NaN	EGGS	CHOCO	APPLE
1	2	MILK	NaN	NaN	CHOCO	APPLE
2	3	NaN	BREAD	EGGS	CHOCO	APPLE
3	4	NaN	BREAD	EGGS	CHOCO	NaN
4	5	MILK	BREAD	NaN	NaN	APPLE
5	6	MILK	NaN	EGGS	CHOCO	APPLE
6	7	MILK	BREAD	EGGS	NaN	APPLE
7	8	NaN	NaN	NaN	CHOCO	APPLE
8	9	MILK	BREAD	NaN	CHOCO	APPLE

```
records=[]
num_rows = data.shape[0] # Now data should be defined in this scope
for i in range(num_rows):
  records.append([str(data.values[i,j]) for j in range(0, min(data.shape[1],6))])

ass_rules=apriori(records,min_support=0.5,confidence=0.7)
results=list(ass_rules)
```

OUTPUT:

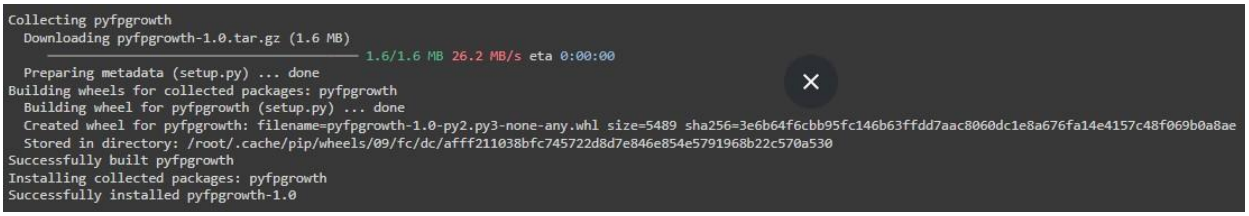


```
[14] print(len(results))

15

[15]
     print((results))

[RelationRecord(items=frozenset({'APPLE'}), support=0.8888888888888888, ordered_statistics=[
```

11) Write a Python program to generate frequent item sets / association rules using FP-growth Tree algorithm.

PROGRAMS:

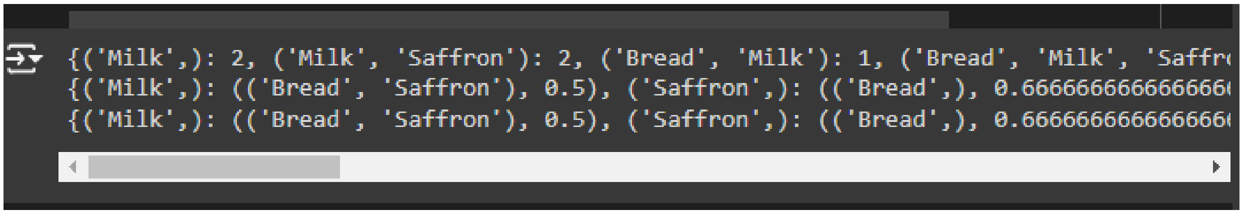
```
pip install pyfpgrowth
```



```
Collecting pyfpgrowth
  Downloading pyfpgrowth-1.0.tar.gz (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 26.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: pyfpgrowth
  Building wheel for pyfpgrowth (setup.py) ... done
  Created wheel for pyfpgrowth: filename=pyfpgrowth-1.0-py2.py3-none-any.whl size=5489 sha256=3e6b64f6cbb95fc146b63ffdd7aac8060dc1e8a676fa14e4157c48f069b0a8ae
  Stored in directory: /root/.cache/pip/wheels/09/fc/dc/afff211038bfc745722d8d7e846e854e5791968b22c570a530
Successfully built pyfpgrowth
Installing collected packages: pyfpgrowth
Successfully installed pyfpgrowth-1.0
```

```
import pyfpgrowth
transactions=[
['Milk','Bread','Saffron'],
 ['Milk','Saffron'],
 ['Bread','Saffron','Wafer'],
 ['Bread','Wafer'],
]

FrequentPatterns = pyfpgrowth.find_frequent_patterns(transactions=transactions,
support_threshold=0.5)
print(FrequentPatterns)
Rules = pyfpgrowth.generate_association_rules(patterns=FrequentPatterns,
confidence_threshold=0.5)
print(Rules)
Rules=pyfpgrowth.generate_association_rules(patterns=FrequentPatterns,confidence_threshold=
0.5)
print(Rules)
```

OUTPUT:



```
{('Milk',): 2, ('Milk', 'Saffron'): 2, ('Bread', 'Milk'): 1, ('Bread', 'Milk', 'Saffr
{('Milk',): (('Bread', 'Saffron'), 0.5), ('Saffron',): (('Bread',), 0.66666666666666
{('Milk',): (('Bread', 'Saffron'), 0.5), ('Saffron',): (('Bread',), 0.66666666666666
```