AIM : Demonstration of preprocessing on datasets student.arff and labor.arff using

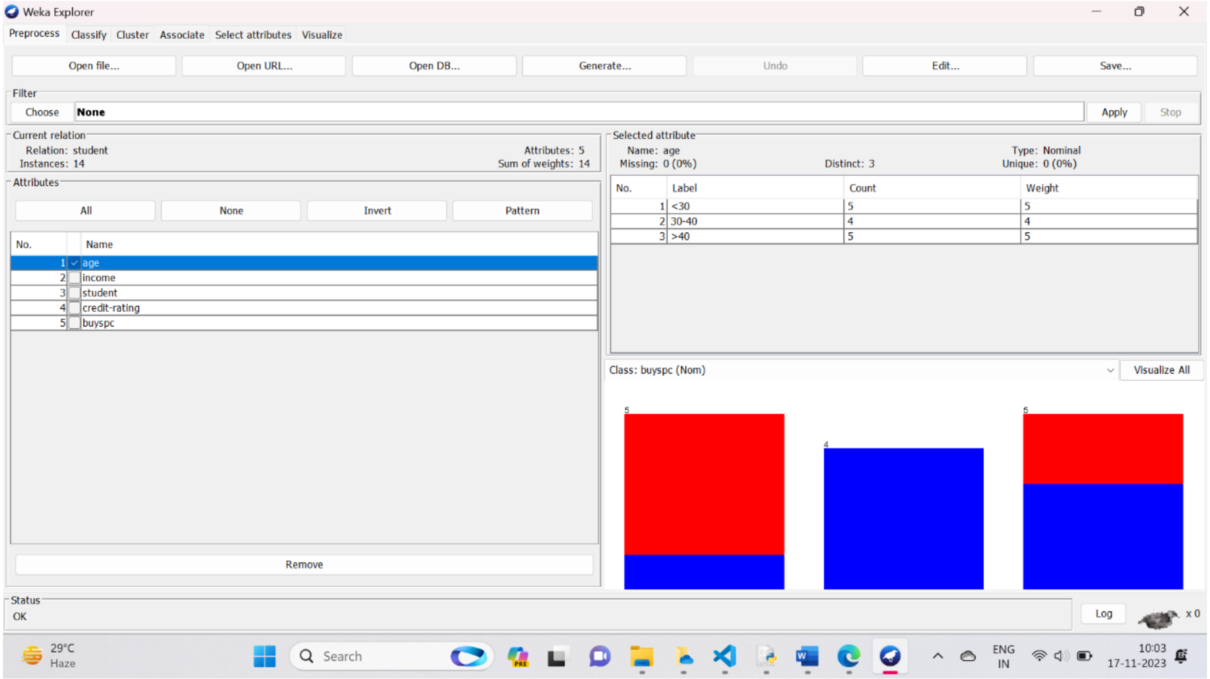
a) WEKA
b) Python

a) WEKA :

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

%
```

Output :



Labor.arff :

@relation 'labor-neg-data'

@attribute 'duration' real

@attribute 'wage-increase-first-year' real

@attribute 'wage-increase-second-year' real

@attribute 'wage-increase-third-year' real

@attribute 'cost-of-living-adjustment' {'none','tcf','tc'}

@attribute 'working-hours' real

@attribute 'pension' {'none','ret_allw','empl_contr'}

@attribute 'standby-pay' real

@attribute 'shift-differential' real

@attribute 'education-allowance' {'yes','no'}

@attribute 'statutory-holidays' real

@attribute 'vacation' {'below_average','average','generous'}

@attribute 'longterm-disability-assistance' {'yes','no'}

@attribute 'contribution-to-dental-plan' {'none','half','full'}

```
@attribute 'bereavement-assistance' {'yes','no'}
@attribute 'contribution-to-health-plan' {'none','half','full'}
@attribute 'class' {'bad','good'}
@data
%
1,5,?,?,?,40,?,?,2,?,11,'average',?,?,'yes',?,'good'
2,4.5,5.8,?,?,35,'ret_allw',?,?,'yes',11,'below_average',?,'full',?,'full','good'
?,?,?,?,?,38,'empl_contr',?,5,?,11,'generous','yes','half','yes','half','good'
3,3.7,4,5,'tc',?,?,?,?,'yes',?,?,?,?,'yes',?,'good'
3,4.5,4.5,5,?,40,?,?,?,?,12,'average',?,'half','yes','half','good'
2,2,2.5,?,?,35,?,?,6,'yes',12,'average',?,?,?,?,'good'
3,4,5,5,'tc',?,'empl_contr',?,?,?,12,'generous','yes','none','yes','half','good'
3,6.9,4.8,2.3,?,40,?,?,3,?,12,'below_average',?,?,?,?,'good'
2,3,7,?,?,38,?,12,25,'yes',11,'below_average','yes','half','yes',?,'good'
1,5.7,?,?,'none',40,'empl_contr',?,4,?,11,'generous','yes','full',?,?,'good'
3,3.5,4,4.6,'none',36,?,?,3,?,13,'generous',?,?,'yes','full','good'
2,6.4,6.4,?,?,38,?,?,4,?,15,?,?,'full',?,?,'good'
2,3.5,4,?,'none',40,?,?,2,'no',10,'below_average','no','half',?,'half','bad'
3,3.5,4,5.1,'tcf',37,?,?,4,?,13,'generous',?,'full','yes','full','good'
1,3,?,?,'none',36,?,?,10,'no',11,'generous',?,?,?,?,'good'
2,4.5,4,?,'none',37,'empl_contr',?,?,?,11,'average',?,'full','yes',?,'good'
1,2.8,?,?,?,35,?,?,2,?,12,'below_average',?,?,?,?,'good'
1,2.1,?,?,'tc',40,'ret_allw',2,3,'no',9,'below_average','yes','half',?,'none','bad'
1,2,?,?,'none',38,'none',?,?,'yes',11,'average','no','none','no','none','bad'
2,4,5,?,'tcf',35,?,13,5,?,15,'generous',?,?,?,?,'good'
2,4.3,4.4,?,?,38,?,?,4,?,12,'generous',?,'full',?,'full','good'
2,2.5,3,?,?,40,'none',?,?,?,11,'below_average',?,?,?,?,'bad'
3,3.5,4,4.6,'tcf',27,?,?,?,?,?,?,?,?,?,?,'good'
2,4.5,4,?,?,40,?,?,4,?,10,'generous',?,'half',?,'full','good'
```

1,6,?,?,?,38,?,8,3,?,9,'generous',?,?,?,?,'good'

3,2,2,2,'none',40,'none',?,?,?,10,'below_average',?,'half','yes','full','bad'

2,4.5,4.5,?,'tcf',?,?,?,?,'yes',10,'below_average','yes','none',?,'half','good'

2,3,3,?,'none',33,?,?,?,'yes',12,'generous',?,?,'yes','full','good'

2,5,4,?,'none',37,?,?,5,'no',11,'below_average','yes','full','yes','full','good'

3,2,2.5,?,?,35,'none',?,?,?,10,'average',?,?,'yes','full','bad'

3,4.5,4.5,5,'none',40,?,?,?,'no',11,'average',?,'half',?,?,'good'

3,3,2,2.5,'tc',40,'none',?,5,'no',10,'below_average','yes','half','yes','full','bad'

2,2.5,2.5,?,?,38,'empl_contr',?,?,?,10,'average',?,?,?,?,'bad'

2,4,5,?,'none',40,'none',?,3,'no',10,'below_average','no','none',?,'none','bad'

3,2,2.5,2.1,'tc',40,'none',2,1,'no',10,'below_average','no','half','yes','full','bad'

2,2,2,?,'none',40,'none',?,?,'no',11,'average','yes','none','yes','full','bad'

1,2,?,?,'tc',40,'ret_allw',4,0,'no',11,'generous','no','none','no','none','bad'

1,2.8,?,?,'none',38,'empl_contr',2,3,'no',9,'below_average','yes','half',?,'none','bad'

3,2,2.5,2,?,37,'empl_contr',?,?,?,10,'average',?,?,'yes','none','bad'

2,4.5,4,?,'none',40,?,?,4,?,12,'average','yes','full','yes','half','good'

1,4,?,?,'none',?,'none',?,?,'yes',11,'average','no','none','no','none','bad'

2,2,3,?,'none',38,'empl_contr',?,?,'yes',12,'generous','yes','none','yes','full','bad'

2,2.5,2.5,?,'tc',39,'empl_contr',?,?,?,12,'average',?,?,'yes',?,'bad'

2,2.5,3,?,'tcf',40,'none',?,?,?,11,'below_average',?,?,'yes',?,'bad'

2,4,4,?,'none',40,'none',?,3,?,10,'below_average','no','none',?,'none','bad'

2,4.5,4,?,?,40,?,?,2,'no',10,'below_average','no','half',?,'half','bad'

2,4.5,4,?,'none',40,?,?,5,?,11,'average',?,'full','yes','full','good'

2,4.6,4.6,?,'tcf',38,?,?,?,?,?,?,'yes','half',?,'half','good'

2,5,4.5,?,'none',38,?,14,5,?,11,'below_average','yes',?,?,'full','good'

2,5.7,4.5,?,'none',40,'ret_allw',?,?,?,11,'average','yes','full','yes','full','good'

2,7,5.3,?,?,?,?,?,?,?,11,?,'yes','full',?,?,'good'

3,2,3,?,'tcf',?,'empl_contr',?,?,'yes',?,?,'yes','half','yes',?,'good'

3,3.5,4,4.5,'tcf',35,?,?,?,?,13,'generous',?,?,'yes','full','good'

3,4,3.5,?,'none',40,'empl_contr',?,6,?,11,'average','yes','full',?,'full','good'

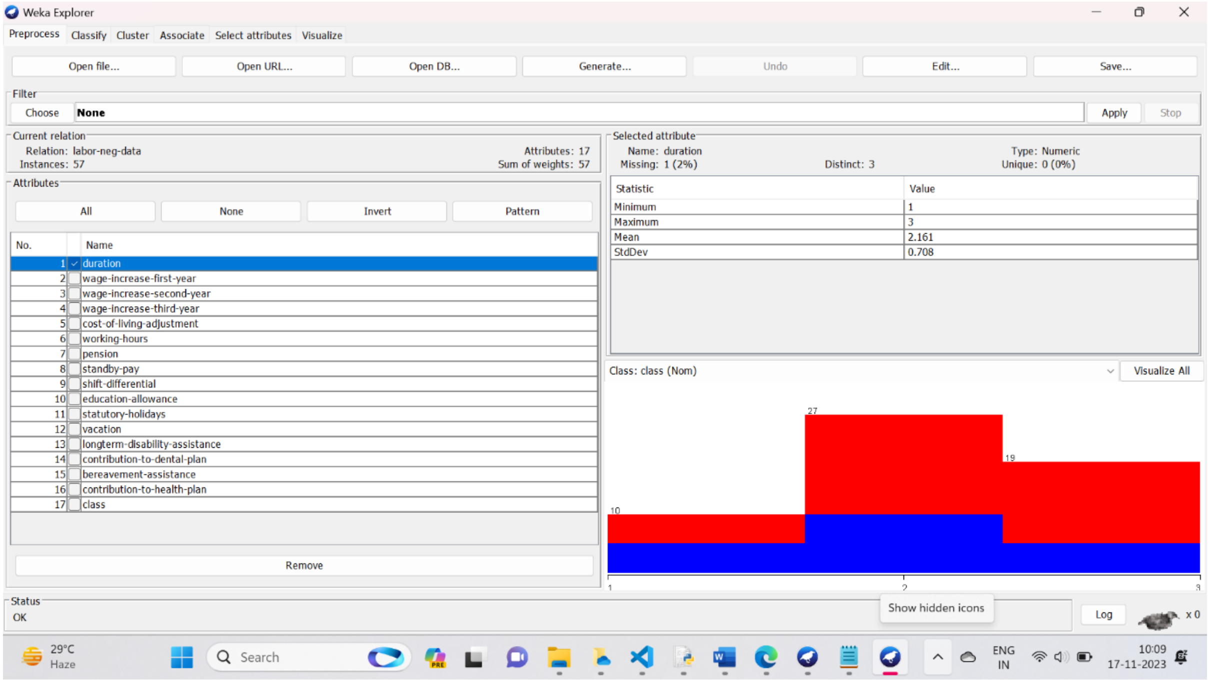
3,5,4.4,?,'none',38,'empl_contr',10,6,?,11,'generous','yes',?,?,'full','good'

3,5,5,5,?,40,?,?,?,?,12,'average',?,'half','yes','half','good'

3,6,6,4,?,35,?,?,14,?,9,'generous','yes','full','yes','full','good'

%

Output :



b) Python :

student.py :

import pandas as pd

from scipy.io import arff

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

Load ARFF file

data, meta = arff.loadarff(r"C:\Users\Rupa\OneDrive\Desktop\DWDM Datasets\Student.arff")

```
# Convert ARFF data to DataFrame
df = pd.DataFrame(data)

# Encode categorical variables using LabelEncoder
categorical_columns = ['age', 'income', 'student', 'credit-rating', 'buyspc']
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Split the dataset into features (X) and the target variable (y)
X = df.drop('buyspc', axis=1)
y = df['buyspc']
print(X)
print(y)
```

Output :

```
    age  income  student  credit-rating
0     1       0        0              1
1     1       0        0              0
2     0       0        0              1
3     2       2        0              1
4     2       1        1              1
5     2       1        1              0
6     0       1        1              0
7     1       2        0              1
8     1       1        1              1
9     2       2        1              1
10    1       2        1              0
11    0       2        0              0
12    0       0        1              1
13    2       2        0              0
0     0
1     0
2     1
3     1
4     1
5     0
6     1
7     0
8     0
9     1
10    1
11    1
12    1
13    0
Name: buyspc, dtype: int32
```

Labor.py :

```
import pandas as pd

from scipy.io import arff

from sklearn.preprocessing import LabelEncoder


# Load ARFF file

data, meta = arff.loadarff(r"C:\Users\Rupa\OneDrive\Desktop\DWDM Datasets\Labor.arff")


# Convert ARFF data to DataFrame

df = pd.DataFrame(data)


# Replace '?' with NaN to handle missing values

df.replace('?', pd.NA, inplace=True)


# Encode categorical variables using LabelEncoder

categorical_columns = ['cost-of-living-adjustment', 'pension', 'education-allowance',
'vacation', 'longterm-disability-assistance', 'contribution-to-dental-plan', 'bereavement-
assistance', 'contribution-to-health-plan', 'class']

label_encoders = {}

for column in categorical_columns:

   le = LabelEncoder()

   df[column] = le.fit_transform(df[column].astype(str))

   label_encoders[column] = le


# Split the dataset into features (X) and target (y)

X = df.drop('class', axis=1)

y = df['class']


# Your dataset is now preprocessed and ready for data mining tasks.

print(X)
```

print(y)

output :

```
    duration  ...  contribution-to-health-plan
0        1.0  ...                            0
1        2.0  ...                            1
2        NaN  ...                            2
3        3.0  ...                            0
4        3.0  ...                            2
5        2.0  ...                            0
6        3.0  ...                            2
7        3.0  ...                            0
8        2.0  ...                            0
9        1.0  ...                            0
10       3.0  ...                            1
11       2.0  ...                            0
12       2.0  ...                            2
13       3.0  ...                            1
14       1.0  ...                            0
15       2.0  ...                            0
16       1.0  ...                            0
17       1.0  ...                            3
18       1.0  ...                            3
19       2.0  ...                            0
20       2.0  ...                            1
21       2.0  ...                            0
22       3.0  ...                            0
23       2.0  ...                            1
24       1.0  ...                            0
25       3.0  ...                            1
26       2.0  ...                            2
27       2.0  ...                            1
28       2.0  ...                            1
29       3.0  ...                            1
30       3.0  ...                            0
31       3.0  ...                            1
32       2.0  ...                            0
33       2.0  ...                            3
34       3.0  ...                            1
35       2.0  ...                            1

35       2.0  ...                            1
36       1.0  ...                            3
37       1.0  ...                            3
38       3.0  ...                            3
39       2.0  ...                            2
40       1.0  ...                            3
41       2.0  ...                            1
42       2.0  ...                            0
43       2.0  ...                            0
44       2.0  ...                            3
45       2.0  ...                            2
46       2.0  ...                            1
47       2.0  ...                            2
48       2.0  ...                            1
49       2.0  ...                            1
50       2.0  ...                            0
51       3.0  ...                            0
52       3.0  ...                            1
53       3.0  ...                            1
54       3.0  ...                            1
55       3.0  ...                            2
56       3.0  ...                            1

[57 rows x 16 columns]
0     1
1     1
2     1
3     1
4     1
5     1
6     1
7     1
8     1
9     1
10    1
11    1
12    0
13    1
14    1
15    1
16    1

18    0
19    1
20    1
21    0
22    1
23    1
24    1
25    0
26    1
27    1
28    1
29    0
30    1
31    0
32    0
33    0
34    0
35    0
36    0
37    0
38    0
39    1
40    0
41    0
42    0
43    0
44    0
45    0
46    1
47    1
48    1
49    1
50    1
51    1
52    1
53    1
54    1
55    1
56    1
Name: class, dtype: int32
```

AIM : Demonstration of Association rule process on dataset contactlenses.arff using apriori algorithm in

a) WEKA

b)Python

a) **WEKA :**

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

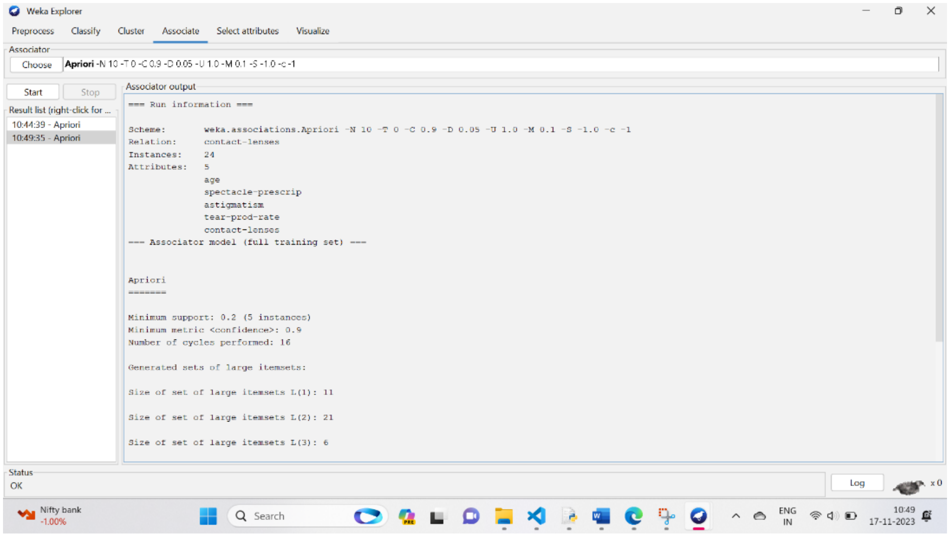
presbyopic,hypermetrope,no,reduced,none

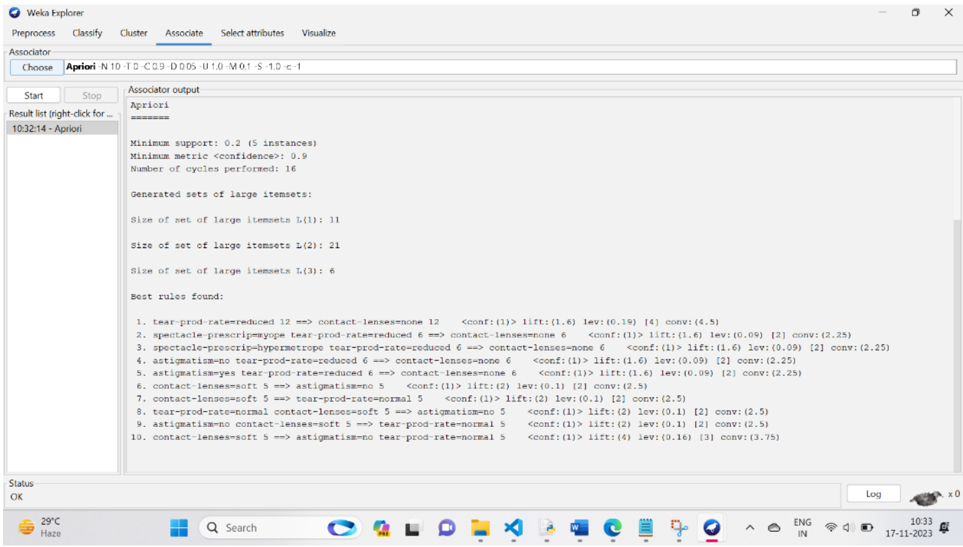
presbyopic,hypermetrope,no,normal,soft

presbyopic,hypermetrope,yes,reduced,none

presbyopic,hypermetrope,yes,normal,none

Output :





b)python :

```
from scipy.io import arff

import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules



# Load the ARFF file

data = arff.loadarff(r"C:\Users\Rupa\OneDrive\Desktop\DWDM Datasets\contactlenses.arff")

df = pd.DataFrame(data[0])



# Convert the nominal attributes to strings

for col in df.columns:

    if pd.api.types.is_categorical_dtype(df[col]):

        df[col] = df[col].str.decode('utf-8')



# Convert the dataset into a one-hot encoded format
```

```
oht = pd.get_dummies(df.iloc[:, :-1], columns=df.columns[:-1], prefix='', prefix_sep='')


# Find frequent itemsets using the Apriori algorithm

min_support = 0.2  # Minimum support threshold (adjust as needed)

frequent_itemsets = apriori(oht, min_support=min_support, use_colnames=True)


# Display frequent itemsets

print("Frequent Itemsets:")

print(frequent_itemsets)


# Find association rules

min_confidence = 0.7  # Minimum confidence threshold (adjust as needed)

association_rules_df = association_rules(frequent_itemsets, metric="lift",
min_threshold=min_confidence)
```

Display association rules

print("\nAssociation Rules:")

print(association_rules_df)

Output :

```
Frequent Itemsets:
     support                        itemsets
0   0.333333           (b'pre-presbyopic')
1   0.333333               (b'presbyopic')
2   0.333333                    (b'young')
3   0.500000           (b'hypermetrope')
4   0.500000                    (b'myope')
5   0.500000                       (b'no')
6   0.500000                      (b'yes')
7   0.500000                   (b'normal')
8   0.500000                  (b'reduced')
9   0.250000        (b'hypermetrope', b'no')
10  0.250000       (b'yes', b'hypermetrope')
11  0.250000    (b'hypermetrope', b'normal')
12  0.250000   (b'hypermetrope', b'reduced')
13  0.250000               (b'myope', b'no')
14  0.250000              (b'myope', b'yes')
15  0.250000           (b'myope', b'normal')
16  0.250000          (b'myope', b'reduced')
17  0.250000              (b'normal', b'no')
18  0.250000             (b'no', b'reduced')
19  0.250000             (b'yes', b'normal')
20  0.250000            (b'yes', b'reduced')

Association Rules:
          antecedents        consequents  ...  conviction  zhangs_metric
0    (b'hypermetrope')            (b'no')  ...         1.0            0.0
1              (b'no')  (b'hypermetrope')  ...         1.0            0.0
2             (b'yes')  (b'hypermetrope')  ...         1.0            0.0
3    (b'hypermetrope')           (b'yes')  ...         1.0            0.0
4    (b'hypermetrope')        (b'normal')  ...         1.0            0.0
5          (b'normal')  (b'hypermetrope')  ...         1.0            0.0
6    (b'hypermetrope')       (b'reduced')  ...         1.0            0.0
7         (b'reduced')  (b'hypermetrope')  ...         1.0            0.0
8           (b'myope')            (b'no')  ...         1.0            0.0
9              (b'no')         (b'myope')  ...         1.0            0.0
10          (b'myope')           (b'yes')  ...         1.0            0.0
11            (b'yes')         (b'myope')  ...         1.0            0.0

12          (b'myope')        (b'normal')  ...         1.0            0.0
13         (b'normal')         (b'myope')  ...         1.0            0.0
14          (b'myope')       (b'reduced')  ...         1.0            0.0
15        (b'reduced')         (b'myope')  ...         1.0            0.0
16         (b'normal')            (b'no')  ...         1.0            0.0
17             (b'no')        (b'normal')  ...         1.0            0.0
18             (b'no')       (b'reduced')  ...         1.0            0.0
19        (b'reduced')            (b'no')  ...         1.0            0.0
20            (b'yes')        (b'normal')  ...         1.0            0.0
21         (b'normal')           (b'yes')  ...         1.0            0.0
22            (b'yes')       (b'reduced')  ...         1.0            0.0
23        (b'reduced')           (b'yes')  ...         1.0            0.0

[24 rows x 10 columns]
```

AIM : Demonstration of Association rule process on dataset supermarket.arff using apriori algorithm in

a) WEKA
b) Python

a) WEKA :

@relation attribute

@attribute bread{y,n}

@attribute jelly{y,n}

@attribute butter{y,n}

@attribute milk{y,n}

@attribute sugar{y,n}

@data

y y y n n

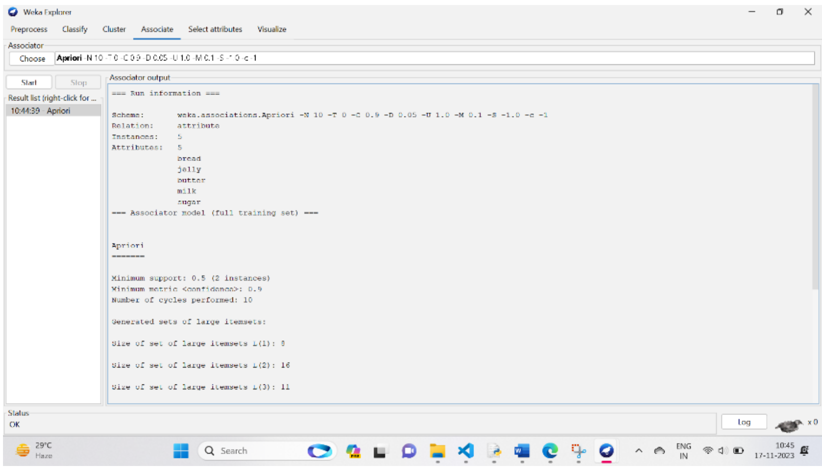
y n y n n

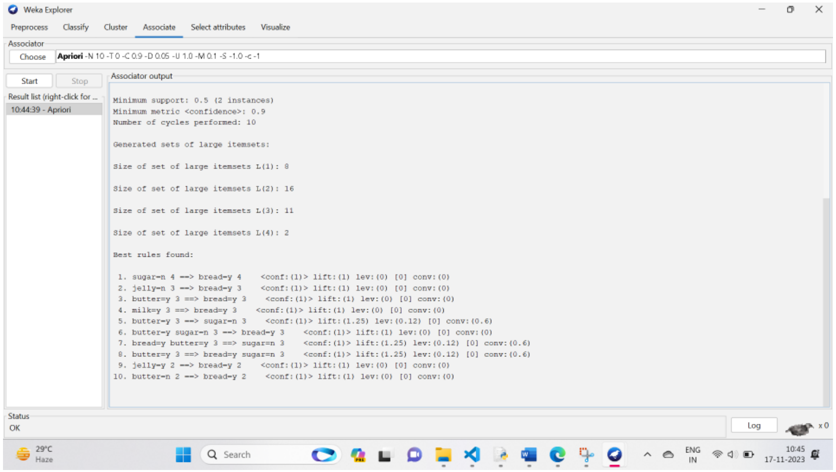
y n y y n

y n n y y

y y n y n

Output :





b)python :

import pandas as pd

from mlxtend.frequent_patterns import apriori

from mlxtend.frequent_patterns import association_rules

Load the dataset

data = pd.read_csv(r"C:\Users\Rupa\OneDrive\Desktop\DWDM Datasets\super market.csv")

Convert 'y' and 'n' to boolean values (True and False)

data = data.applymap(lambda x: True if x == 'y' else False)

Perform one-hot encoding

one_hot_encoded = pd.get_dummies(data)

Find frequent item sets with minimum support

frequent_itemsets = apriori(one_hot_encoded, min_support=0.2, use_colnames=True)

Generate association rules with minimum confidence and compute lift

rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)

```
# Display the association rules

print("Association Rules:")

print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])


# Save the rules to a CSV file if needed

# rules.to_csv("association_rules.csv", index=False)
```

Output :

```
Association Rules:
       antecedents      consequents  support  confidence      lift
0          (bread)          (jelly)      0.4    0.400000  1.000000
1          (jelly)          (bread)      0.4    1.000000  1.000000
2         (butter)          (bread)      0.6    1.000000  1.000000
3          (bread)         (butter)      0.6    0.600000  1.000000
4          (bread)           (milk)      0.6    0.600000  1.000000
5           (milk)          (bread)      0.6    1.000000  1.000000
6          (bread)          (sugar)      0.2    0.200000  1.000000
7          (sugar)          (bread)      0.2    1.000000  1.000000
8          (sugar)           (milk)      0.2    1.000000  1.666667
9           (milk)          (sugar)      0.2    0.333333  1.666667
10  (butter, jelly)         (bread)      0.2    1.000000  1.000000
11         (bread)  (butter, jelly)      0.2    0.200000  1.000000
12    (jelly, milk)         (bread)      0.2    1.000000  1.000000
13         (bread)    (jelly, milk)      0.2    0.200000  1.000000
14   (butter, milk)         (bread)      0.2    1.000000  1.000000
15         (bread)   (butter, milk)      0.2    0.200000  1.000000
16   (bread, sugar)          (milk)      0.2    1.000000  1.666667
17    (bread, milk)         (sugar)      0.2    0.333333  1.666667
18    (sugar, milk)         (bread)      0.2    1.000000  1.000000
19         (bread)    (sugar, milk)      0.2    0.200000  1.000000
20         (sugar)    (bread, milk)      0.2    1.000000  1.666667
21          (milk)   (bread, sugar)      0.2    0.333333  1.666667
```

AIM : Demonstration of classification rule process on dataset employee.arff using id3 algorithm in

a) WEKA
b) Python

b)python :

```
import csv

import math


# Function to calculate entropy

def entropy(data, target_attribute):

    # Count the occurrences of each target value

    target_counts = {}
```

```python
    for row in data:
        target_value = row[target_attribute]
        if target_value not in target_counts:
            target_counts[target_value] = 0
        target_counts[target_value] += 1

    # Calculate entropy using the formula
    entropy_value = 0
    total_instances = len(data)
    for target_value in target_counts:
        probability = target_counts[target_value] / total_instances
        entropy_value -= probability * math.log2(probability)

    return entropy_value

# Function to calculate information gain for an attribute
def information_gain(data, attribute, target_attribute):
    total_entropy = entropy(data, target_attribute)
    total_instances = len(data)

    attribute_values = set([row[attribute] for row in data])
    weighted_entropy = 0

    for value in attribute_values:
        subset = [row for row in data if row[attribute] == value]
        subset_entropy = entropy(subset, target_attribute)
        probability = len(subset) / total_instances
        weighted_entropy += probability * subset_entropy
```

```
    return total_entropy - weighted_entropy


# Function to choose the best attribute to split on
def choose_best_attribute(data, attributes, target_attribute):
    best_attribute = None
    max_info_gain = -1

    for attribute in attributes:
        info_gain = information_gain(data, attribute, target_attribute)
        if info_gain > max_info_gain:
            max_info_gain = info_gain
            best_attribute = attribute

    return best_attribute


# Recursive ID3 algorithm to build the decision tree
def id3(data, attributes, target_attribute):
    target_values = set([row[target_attribute] for row in data])

    # If all instances have the same target value, return that value
    if len(target_values) == 1:
        return target_values.pop()

    # If there are no attributes left to split on, return the majority target value
    if len(attributes) == 0:
        majority_target = max(set([row[target_attribute] for row in data]),
key=[row[target_attribute] for row in data].count)
        return majority_target

    # Choose the best attribute to split on
```

```
    best_attribute = choose_best_attribute(data, attributes, target_attribute)

    # Create a new decision tree node with the best attribute as its label
    tree = {best_attribute: {}}
    attribute_values = set([row[best_attribute] for row in data])

    # Recursively build the subtree for each attribute value
    for value in attribute_values:
        subset = [row for row in data if row[best_attribute] == value]
        subtree = id3(subset, [attr for attr in attributes if attr != best_attribute],
target_attribute)
        tree[best_attribute][value] = subtree

    return tree

# Read data from a CSV file
data = []
with open(r"C:\Users\Rupa\OneDrive\Desktop\DWDM Datasets\employee.csv") as
csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        data.append(row)

# List of attributes (excluding the target attribute)
attributes = ['age', 'salary']

# Target attribute
target_attribute = 'performance'

# Build the decision tree
```

decision_tree = id3(data, attributes, target_attribute)

Print the resulting decision tree

import pprint

pprint.pprint(decision_tree)

Output :

```
{'age': {'%': None,
         '25': ' poor',
         '27': ' poor',
         '28': ' poor',
         '29': ' avg',
         '30': ' avg',
         '35': ' good',
         '48': {'salary': {' 32k': 'good'}}}}
```

AIM : Demonstration of classification rule process on dataset employee.arff using naïve bayes algorithm in

a) WEKA
b) Python

a) WEKA :

@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

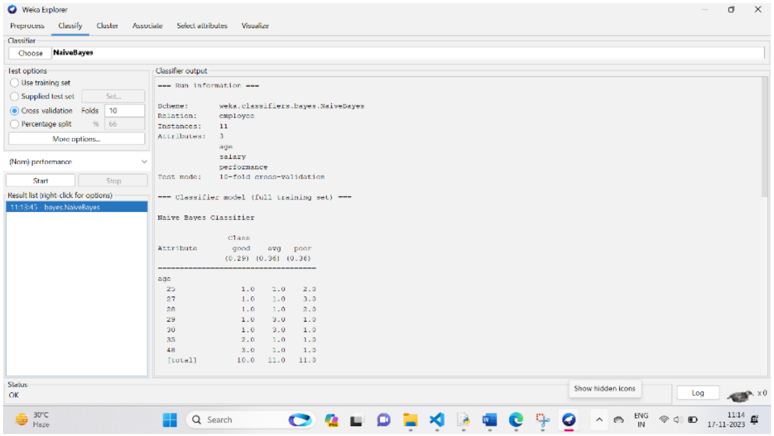
35, 32k, good

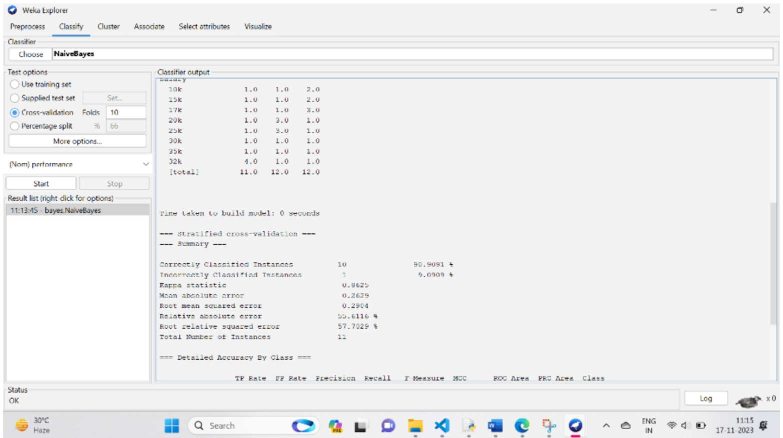
48, 32k, good

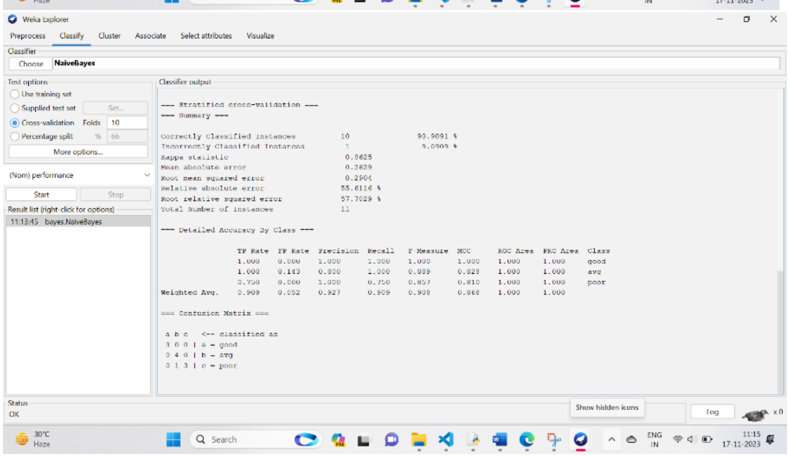
48, 32k,good

%

Output :







b)python :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report

# Read the dataset into a Pandas DataFrame
data = pd.read_csv(r"C:\Users\Rupa\OneDrive\Desktop\DWDM Datasets\employee (2).csv")

# Encode the categorical attributes (age, salary, performance) to numeric values
le = LabelEncoder()
data['age'] = le.fit_transform(data['age'])
data['salary'] = le.fit_transform(data['salary'])
data['performance'] = le.fit_transform(data['performance'])

# Split the dataset into features (X) and target (y)
X = data[['age', 'salary']]
y = data['performance']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train a Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_classifier.predict(X_test)
```

Evaluate the classifier

accuracy = accuracy_score(y_test, y_pred)

classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)

print("Classification Report:\n", classification_rep)

Output :

```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         3

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

AIM : Demonstration of clustering rule process on datasets iris.arff and student.arff using simple K-means in

a) WEKA
b) Python

a) WEKA :

Iris.arff :

@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

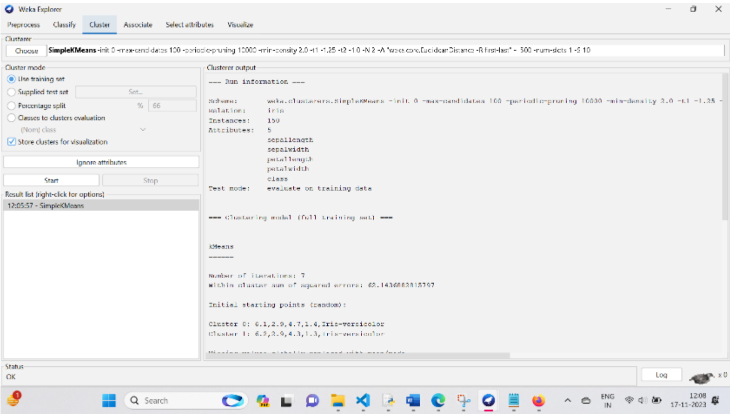
6.5,3.0,5.2,2.0,Iris-virginica

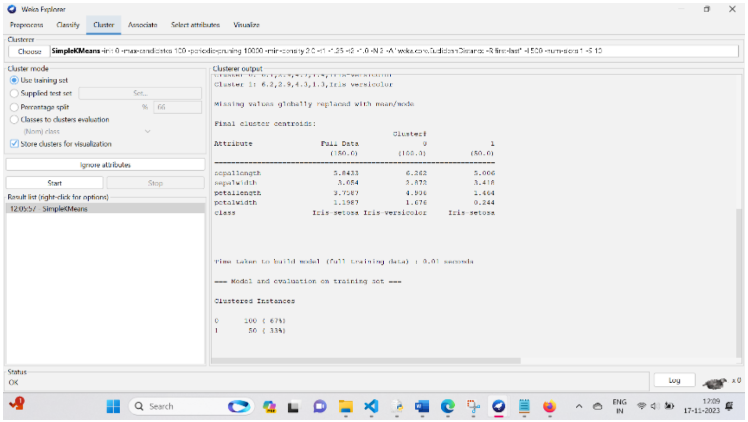
6.2,3.4,5.4,2.3,Iris-virginica

5.9,3.0,5.1,1.8,Iris-virginica

%

Output :





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
%
```

Output :