Tribhuvan University

NEPALAYA COLLEGE
Kalanki, Kathmandu

Lab Report Of:
Data Warehousing and Data Mining

Submitted to:
Narayan Chalise

Submitted by
Raju Chaudhary(24201/076)

Under the Supervision of
CSIT Department Nepalaya College

In partial fulfillment of the requirement for Bachelor's Degree in Computer Science and Information Technology (B.Sc. CSIT), 7th Sem
March 2024

Content

1) **Understand the features of WEKA tool kit such as Explorer, Knowledge flow interface, Experimenter, command-line interface.**

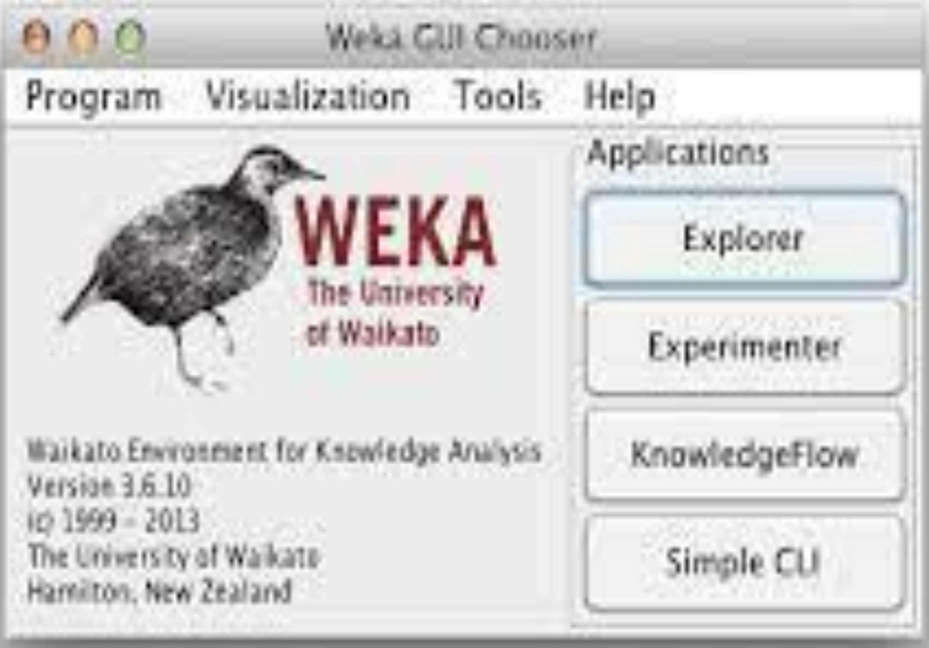


Fig: Weka

WEKA (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. It provides a wide range of tools for data pre-processing, classification, regression, clustering, association rules mining, and visualization. Here are the main features of WEKA:

Explorer: The Explorer is a graphical user interface (GUI) that allows users to interactively explore datasets, build and evaluate machine learning models, and visualize results. It provides various tabs and panels for loading datasets, selecting classifiers, configuring options, running experiments, and viewing results.

Knowledge Flow Interface: The Knowledge Flow interface provides a visual programming environment for constructing machine learning workflows. Users can create complex data processing and modeling pipelines by connecting components such as data sources, filters, classifiers, and evaluators in a graphical manner. This interface is particularly useful for building automated data analysis workflows and experimenting with different combinations of algorithms and parameters.

Experimenter: The Experimenter allows users to design and conduct systematic experiments to evaluate the performance of machine learning algorithms on different datasets. It provides capabilities for defining experimental setups, running multiple trials with various configurations, and analyzing results statistically. The Experimenter helps researchers and practitioners compare the effectiveness of different algorithms and techniques across multiple datasets in a controlled environment.

Command-Line Interface (CLI): WEKA also provides a command-line interface for users who prefer working with text-based commands or need to automate tasks. The command-line interface allows users to execute WEKA functionalities from the terminal or script files, enabling batch processing, integration with other software tools, and deployment in production environments.

2) Navigate the options available in the WEKA such as select attributes panel, preprocess panel, classify panel, associate panel and visualize.

In WEKA's Explorer interface, you'll find several panels that allow you to perform various tasks related to data preprocessing, attribute selection, classification, association rule mining, and visualization. Here's a brief overview of each panel and the options they provide:

1. Preprocess Panel:
   - ➢ This panel is used for data preprocessing tasks such as cleaning noisy data, handling missing values, transforming attributes, and filtering instances.
   - ➢ Options available include:
   - Loading datasets from various file formats.
   - Applying filters for data preprocessing, such as removing attributes, replacing missing values, discretizing numeric attributes, and transforming data.
   - Configuring filter options and parameters.
   - Saving preprocessed data to a file.

2. Select Attributes Panel:

- ➢ This panel is used for selecting a subset of attributes from the dataset or transforming existing attributes.
- ➢ Options available include:
- ➢ Selecting attributes based on various criteria such as information gain, correlation, and principal components analysis (PCA).
- ➢ Removing irrelevant or redundant attributes.
- ➢ Transforming attributes using attribute selection algorithms.
- ➢ Visualizing attribute distributions and correlations.

3. Classify Panel:

- ➢ This panel is used for building and evaluating classification models on the dataset.
- ➢ Options available include:
- ➢ Selecting classification algorithms such as decision trees, support vector machines (SVM), k-nearest neighbors (k-NN), and Naive Bayes.
- ➢ Configuring algorithm-specific parameters.
- ➢ Evaluating model performance using cross-validation, training/test splits, or other evaluation methods.
- ➢ Visualizing classification results, such as confusion matrices and ROC curves.

4. Associate Panel:

- ➢ This panel is used for discovering association rules and frequent itemsets in transactional data.
- ➢ Options available include:
- ➢ Selecting association rule mining algorithms such as Apriori and FP-growth.
- ➢ Specifying minimum support and confidence thresholds.
- ➢ Generating association rules and analyzing their significance.
- ➢ Visualizing frequent itemsets and association rules.

5. Visualize Panel:
- This panel is used for visualizing datasets, models, and evaluation results.
- Options available include:
- Visualizing attribute distributions, scatter plots, and histograms.
- Visualizing decision trees, rule sets, and other classification models.
- Visualizing evaluation metrics such as ROC curves, precision-recall curves, and lift charts.
- Interactively exploring data visualizations for insights and patterns.

These panels in WEKA's Explorer interface provide a comprehensive set of options for performing various data analysis tasks, from preprocessing and feature selection to classification, association rule mining, and visualization. Users can explore and manipulate their data effectively to derive meaningful insights and build predictive models.

3) Study the ARFF file format.

The ARFF (Attribute-Relation File Format) is a plain text file format commonly used to describe datasets for machine learning tasks. It was developed as part of the WEKA software package but has gained popularity beyond WEKA due to its simplicity and flexibility. ARFF files consist of two main sections: the header section and the data section.

Here's an overview of the structure and features of the ARFF file format:

Header Section:

- The header section defines the metadata and attributes of the dataset.
- It begins with the @relation keyword followed by the name of the dataset.
- Each attribute is defined using the @attribute keyword followed by the attribute name and its type.
    - Attribute types can be numeric, nominal, string, date, or relational.
    - Numeric attributes are specified as numeric.
    - Nominal attributes list their possible values within curly braces {}.
    - String attributes are specified as string.
    - Date attributes have the format date "yyyy-MM-dd HH:mm:ss".
- Optionally, the header section may include additional metadata such as comments and information about the source of the dataset.

Data Section:

- The data section contains the actual instances or examples of the dataset.
- It starts with the @data keyword followed by the data instances, one per line.
- Each instance consists of attribute values separated by commas (,), respecting the order of attributes defined in the header section.
- Missing values are represented as a question mark (?) or left blank.

Here's a simple example of an ARFF file representing a dataset with two attributes (temperature and outlook) and three instances:

```
less
@relation weather
@attribute temperature numeric
@attribute outlook {sunny, overcast, rainy}
@data
30,sunny
25,overcast
20,rainy
```

In this example:

The dataset is named "weather".

It has two attributes: "temperature" (numeric) and "outlook" (nominal with three possible values: sunny, overcast, rainy).

The data section contains three instances, each with values for the two attributes.

ARFF files are widely supported by various machine learning libraries and tools, making them a convenient and interoperable format for sharing and working with datasets across different platforms and applications.

4) Explore the available data sets in WEKA.

In WEKA, there are several datasets available for users to explore and use for experimentation and learning purposes. These datasets cover various domains such as classification, regression, clustering, and association rule mining. Here's how you can access the available datasets in WEKA:

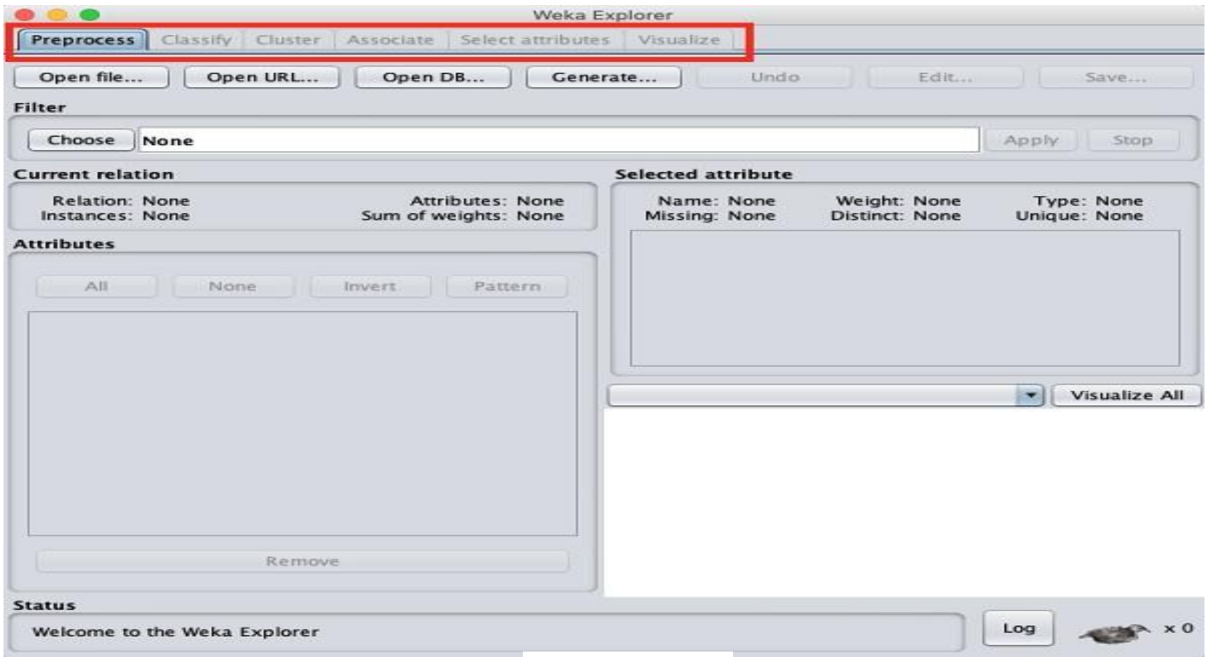


Fig: Explorer

1. **Using the Explorer Interface:**

- Open WEKA and launch the Explorer interface.
- Go to the "Preprocess" panel.
- Click on the "Open file" button.
- Navigate to the "data" directory within the WEKA installation directory.
- You'll find a variety of datasets categorized into different folders based on their domain or type (e.g., classification, regression, clustering).
- Select the dataset you're interested in, and it will be loaded into WEKA for analysis.

2. Using the Knowledge Flow Interface:

- Open WEKA and launch the Knowledge Flow interface.
- Drag the "ArffLoader" operator onto the canvas.
- Double-click on the "ArffLoader" operator to configure it.
- In the "File" parameter, browse to the "data" directory within the WEKA installation directory.
- Select the dataset you want to load.
- Connect the output port of the "ArffLoader" operator to other operators for further processing or analysis.

3. Directly Accessing the Dataset Files:

- Navigate to the "data" directory within the WEKA installation directory using a file browser or command line.
- You'll find various dataset files in ARFF format (*.arff) stored in this directory.
- You can directly open these files using any text editor to view the dataset attributes and instances.

Some of the commonly used datasets available in WEKA include:

- ➢ Iris dataset: A classic dataset for classification, containing measurements of iris flowers.
- ➢ Breast Cancer Wisconsin (Diagnostic) dataset: A dataset for binary classification of breast cancer diagnoses.
- ➢ Housing dataset: A dataset for regression analysis, containing housing prices and attributes.
- ➢ Chess dataset: A dataset for clustering, containing chess games data.
- ➢ Weather dataset: A dataset for association rule mining, containing weather conditions.

These datasets, along with many others available in WEKA, serve as valuable resources for learning and practicing machine learning techniques and algorithms. They cover a wide range of scenarios and can be used for educational purposes, research, and experimentation.

5) Explore various options in WEKA for preprocessing data and apply Discretization filter and Resample filter etc. in each dataset.

In WEKA, the "Preprocess" panel provides various options for preprocessing data before applying machine learning algorithms. Here's how you can apply the Discretization filter and Resample filter to a dataset using WEKA:

Discretization:

- Open WEKA and launch the Explorer interface.
- Go to the "Preprocess" panel.
- Click on the "Open file" button to load the dataset you want to preprocess.
- Once the dataset is loaded, select the "Filters" tab in the "Preprocess" panel.
- Scroll down or use the search bar to find the "unsupervised.attribute.Discretize" filter.
- Double-click on the filter to configure it.
- In the filter configuration window, you can adjust parameters such as the number of bins for discretization and the range of attributes to apply the filter to.
- Click "OK" to apply the Discretization filter to the dataset.
- Optionally, you can preview the discretized dataset by clicking on the "Apply" button in the "Preprocess" panel.

Resampling:

- After applying the Discretization filter or any other preprocessing steps, you can further preprocess the dataset by applying the Resample filter for data sampling.
- In the "Filters" tab of the "Preprocess" panel, scroll down or use the search bar to find the "filters.unsupervised.instance.Resample" filter.
- Double-click on the filter to configure it.
- In the filter configuration window, you can specify options such as the percentage of instances to sample, whether to use with or without replacement, and seed for randomization.
- Click "OK" to apply the Resample filter to the dataset.
- Optionally, you can preview the resampled dataset by clicking on the "Apply" button in the "Preprocess" panel.

These steps demonstrate how to apply the Discretization filter and Resample filter to preprocess a dataset using WEKA's Explorer interface. These preprocessing techniques are commonly used to prepare data for machine learning tasks by transforming continuous attributes into discrete ones and adjusting the distribution of instances in the dataset.

6) Extract the if-then rule from the decision tree generated by the classifier.

In WEKA, you can extract if-then rules from a decision tree generated by a classifier using the "ClassifierTree" option. Here's how you can do it:

Generate the Decision Tree:

- Open WEKA and load the dataset you want to work with.
- Go to the "Classify" panel.
- Choose a decision tree classifier such as J48 (C4.5) or RandomForest.
- Configure the classifier options as needed.
- Click on the "Start" button to build the decision tree model.

View the Decision Tree:

- Once the decision tree model is built, you can view it by clicking on the "More options" button (three dots) next to the classifier in the "Result list" panel.
- Select "Visualize tree" to see the graphical representation of the decision tree.

Extract If-Then Rules:

- After visualizing the decision tree, you can extract if-then rules directly from the tree structure.
- Analyze the branches and nodes of the decision tree to identify the rules.
- Each path from the root node to a leaf node corresponds to a rule.
- For each rule, you can extract the conditions (if-part) and the predicted class (then-part).

Example:

- Suppose you have a decision tree with the following structure:

```
if (attribute1 <= value1) and (attribute2 > value2) then class A

else if (attribute1 > value3) and (attribute3 <= value4) then class B

else class C
```

- In this example, each if-then rule corresponds to a path from the root node to a leaf node in the decision tree.
- For instance, the rule "if (attribute1 <= value1) and (attribute2 > value2) then class A" represents one path in the decision tree.

Export Rules:

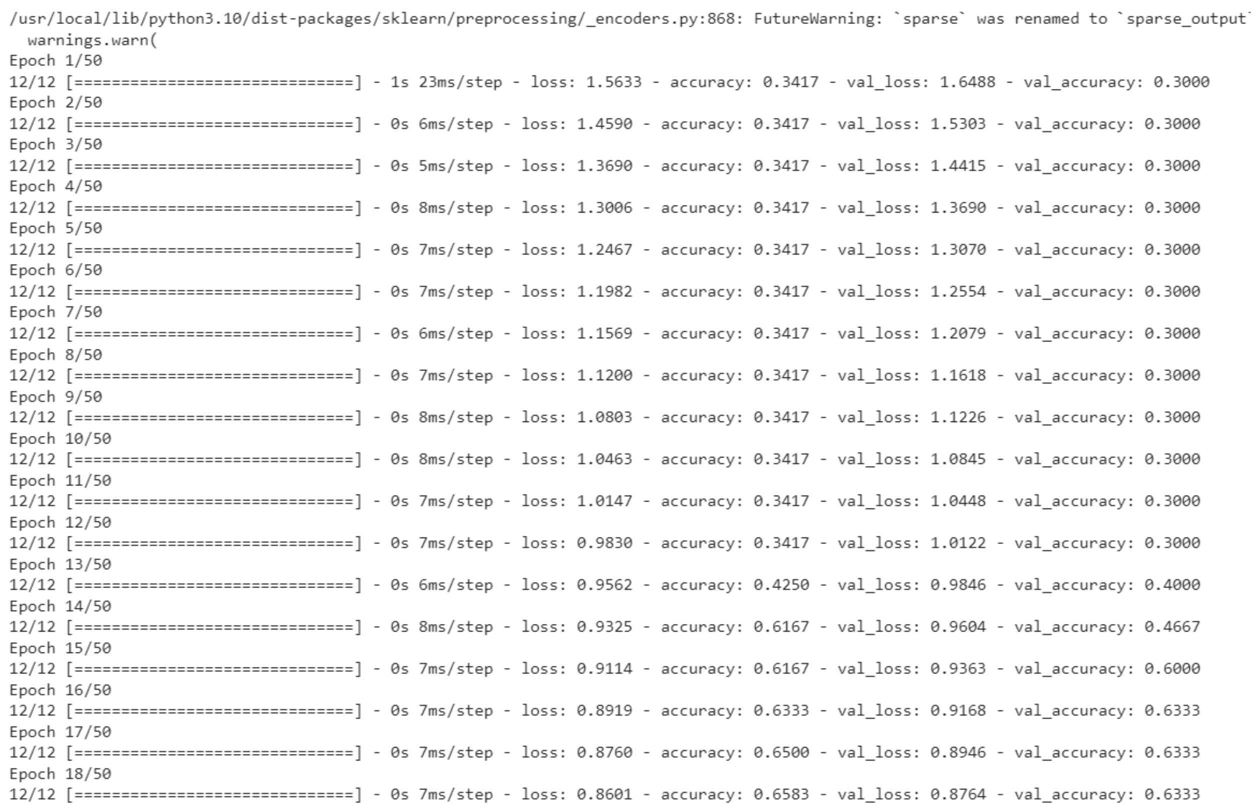
- If you need to export the extracted rules for further analysis or use, you may need to manually transcribe them or use third-party tools to convert the decision tree into a rule-based representation.

By following these steps, you can extract if-then rules from a decision tree generated by a classifier in WEKA. This process allows you to interpret and understand the underlying decision-making logic of the model.

7) Train the model using Back Propagation.

➢
```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target.reshape(-1, 1)

# One-hot encode the target variable
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform(y)

# 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)
```

```
# Define the neural network architecture
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(4,)),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model using backpropagation
model.fit(X_train, y_train, epochs=50, batch_size=10, validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Loss: {loss}, Accuracy: {accuracy}")
```



```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output`
  warnings.warn(
Epoch 1/50
12/12 [==============================] - 1s 23ms/step - loss: 1.5633 - accuracy: 0.3417 - val_loss: 1.6488 - val_accuracy: 0.3000
Epoch 2/50
12/12 [==============================] - 0s 6ms/step - loss: 1.4590 - accuracy: 0.3417 - val_loss: 1.5303 - val_accuracy: 0.3000
Epoch 3/50
12/12 [==============================] - 0s 5ms/step - loss: 1.3690 - accuracy: 0.3417 - val_loss: 1.4415 - val_accuracy: 0.3000
Epoch 4/50
12/12 [==============================] - 0s 8ms/step - loss: 1.3006 - accuracy: 0.3417 - val_loss: 1.3690 - val_accuracy: 0.3000
Epoch 5/50
12/12 [==============================] - 0s 7ms/step - loss: 1.2467 - accuracy: 0.3417 - val_loss: 1.3070 - val_accuracy: 0.3000
Epoch 6/50
12/12 [==============================] - 0s 7ms/step - loss: 1.1982 - accuracy: 0.3417 - val_loss: 1.2554 - val_accuracy: 0.3000
Epoch 7/50
12/12 [==============================] - 0s 6ms/step - loss: 1.1569 - accuracy: 0.3417 - val_loss: 1.2079 - val_accuracy: 0.3000
Epoch 8/50
12/12 [==============================] - 0s 7ms/step - loss: 1.1200 - accuracy: 0.3417 - val_loss: 1.1618 - val_accuracy: 0.3000
Epoch 9/50
12/12 [==============================] - 0s 8ms/step - loss: 1.0803 - accuracy: 0.3417 - val_loss: 1.1226 - val_accuracy: 0.3000
Epoch 10/50
12/12 [==============================] - 0s 8ms/step - loss: 1.0463 - accuracy: 0.3417 - val_loss: 1.0845 - val_accuracy: 0.3000
Epoch 11/50
12/12 [==============================] - 0s 7ms/step - loss: 1.0147 - accuracy: 0.3417 - val_loss: 1.0448 - val_accuracy: 0.3000
Epoch 12/50
12/12 [==============================] - 0s 7ms/step - loss: 0.9830 - accuracy: 0.3417 - val_loss: 1.0122 - val_accuracy: 0.3000
Epoch 13/50
12/12 [==============================] - 0s 6ms/step - loss: 0.9562 - accuracy: 0.4250 - val_loss: 0.9846 - val_accuracy: 0.4000
Epoch 14/50
12/12 [==============================] - 0s 8ms/step - loss: 0.9325 - accuracy: 0.6167 - val_loss: 0.9604 - val_accuracy: 0.4667
Epoch 15/50
12/12 [==============================] - 0s 7ms/step - loss: 0.9114 - accuracy: 0.6167 - val_loss: 0.9363 - val_accuracy: 0.6000
Epoch 16/50
12/12 [==============================] - 0s 7ms/step - loss: 0.8919 - accuracy: 0.6333 - val_loss: 0.9168 - val_accuracy: 0.6333
Epoch 17/50
12/12 [==============================] - 0s 7ms/step - loss: 0.8760 - accuracy: 0.6500 - val_loss: 0.8946 - val_accuracy: 0.6333
Epoch 18/50
12/12 [==============================] - 0s 7ms/step - loss: 0.8601 - accuracy: 0.6583 - val_loss: 0.8764 - val_accuracy: 0.6333
```

```
Epoch 19/50
12/12 [==============================] - 0s 6ms/step - loss: 0.8450 - accuracy: 0.6667 - val_loss: 0.8601 - val_accuracy: 0.6333
Epoch 20/50
12/12 [==============================] - 0s 6ms/step - loss: 0.8320 - accuracy: 0.6667 - val_loss: 0.8440 - val_accuracy: 0.6333
Epoch 21/50
12/12 [==============================] - 0s 6ms/step - loss: 0.8201 - accuracy: 0.6667 - val_loss: 0.8289 - val_accuracy: 0.6333
Epoch 22/50
12/12 [==============================] - 0s 7ms/step - loss: 0.8078 - accuracy: 0.6750 - val_loss: 0.8145 - val_accuracy: 0.6333
Epoch 23/50
12/12 [==============================] - 0s 15ms/step - loss: 0.7968 - accuracy: 0.6750 - val_loss: 0.8017 - val_accuracy: 0.6333
Epoch 24/50
12/12 [==============================] - 0s 16ms/step - loss: 0.7853 - accuracy: 0.6750 - val_loss: 0.7908 - val_accuracy: 0.6333
Epoch 25/50
12/12 [==============================] - 0s 12ms/step - loss: 0.7753 - accuracy: 0.6750 - val_loss: 0.7799 - val_accuracy: 0.6333
Epoch 26/50
12/12 [==============================] - 0s 15ms/step - loss: 0.7657 - accuracy: 0.6833 - val_loss: 0.7668 - val_accuracy: 0.6333
Epoch 27/50
12/12 [==============================] - 0s 13ms/step - loss: 0.7563 - accuracy: 0.6917 - val_loss: 0.7559 - val_accuracy: 0.7333
Epoch 28/50
12/12 [==============================] - 0s 12ms/step - loss: 0.7462 - accuracy: 0.7250 - val_loss: 0.7451 - val_accuracy: 0.7667
Epoch 29/50
12/12 [==============================] - 0s 20ms/step - loss: 0.7371 - accuracy: 0.7417 - val_loss: 0.7344 - val_accuracy: 0.8333
Epoch 30/50
12/12 [==============================] - 0s 24ms/step - loss: 0.7283 - accuracy: 0.8167 - val_loss: 0.7254 - val_accuracy: 0.8000
Epoch 31/50
12/12 [==============================] - 0s 12ms/step - loss: 0.7196 - accuracy: 0.8333 - val_loss: 0.7155 - val_accuracy: 0.8000
Epoch 32/50
12/12 [==============================] - 0s 8ms/step - loss: 0.7105 - accuracy: 0.8167 - val_loss: 0.7064 - val_accuracy: 0.7333
Epoch 33/50
12/12 [==============================] - 0s 10ms/step - loss: 0.7022 - accuracy: 0.7500 - val_loss: 0.6968 - val_accuracy: 0.7333
Epoch 34/50
12/12 [==============================] - 0s 9ms/step - loss: 0.6935 - accuracy: 0.7417 - val_loss: 0.6878 - val_accuracy: 0.7333
Epoch 35/50
12/12 [==============================] - 0s 12ms/step - loss: 0.6852 - accuracy: 0.7000 - val_loss: 0.6787 - val_accuracy: 0.7333
Epoch 36/50
12/12 [==============================] - 0s 9ms/step - loss: 0.6767 - accuracy: 0.6917 - val_loss: 0.6717 - val_accuracy: 0.7333
Epoch 37/50
12/12 [==============================] - 0s 8ms/step - loss: 0.6693 - accuracy: 0.7083 - val_loss: 0.6609 - val_accuracy: 0.7333
Epoch 38/50
12/12 [==============================] - 0s 9ms/step - loss: 0.6607 - accuracy: 0.6917 - val_loss: 0.6530 - val_accuracy: 0.7333
Epoch 39/50
12/12 [==============================] - 0s 10ms/step - loss: 0.6533 - accuracy: 0.6833 - val_loss: 0.6457 - val_accuracy: 0.7333
Epoch 40/50
12/12 [==============================] - 0s 8ms/step - loss: 0.6449 - accuracy: 0.7167 - val_loss: 0.6384 - val_accuracy: 0.7333
Epoch 41/50
12/12 [==============================] - 0s 10ms/step - loss: 0.6381 - accuracy: 0.7250 - val_loss: 0.6301 - val_accuracy: 0.7333
Epoch 42/50
12/12 [==============================] - 0s 12ms/step - loss: 0.6300 - accuracy: 0.7000 - val_loss: 0.6227 - val_accuracy: 0.7333
Epoch 43/50
12/12 [==============================] - 0s 9ms/step - loss: 0.6231 - accuracy: 0.6667 - val_loss: 0.6145 - val_accuracy: 0.7333
Epoch 44/50
12/12 [==============================] - 0s 9ms/step - loss: 0.6157 - accuracy: 0.6833 - val_loss: 0.6084 - val_accuracy: 0.7333
Epoch 45/50
12/12 [==============================] - 0s 8ms/step - loss: 0.6084 - accuracy: 0.7000 - val_loss: 0.6011 - val_accuracy: 0.7333
Epoch 46/50
12/12 [==============================] - 0s 8ms/step - loss: 0.6019 - accuracy: 0.7000 - val_loss: 0.5935 - val_accuracy: 0.7333
Epoch 47/50
12/12 [==============================] - 0s 9ms/step - loss: 0.5955 - accuracy: 0.6917 - val_loss: 0.5863 - val_accuracy: 0.7333
Epoch 48/50
12/12 [==============================] - 0s 8ms/step - loss: 0.5889 - accuracy: 0.6917 - val_loss: 0.5801 - val_accuracy: 0.7333
Epoch 49/50
12/12 [==============================] - 0s 7ms/step - loss: 0.5825 - accuracy: 0.7083 - val_loss: 0.5745 - val_accuracy: 0.7333
Epoch 50/50
12/12 [==============================] - 0s 9ms/step - loss: 0.5766 - accuracy: 0.7083 - val_loss: 0.5680 - val_accuracy: 0.7333
1/1 [==============================] - 0s 311ms/step - loss: 0.5680 - accuracy: 0.7333
Loss: 0.5680186152458191, Accuracy: 0.7333333492279053
```

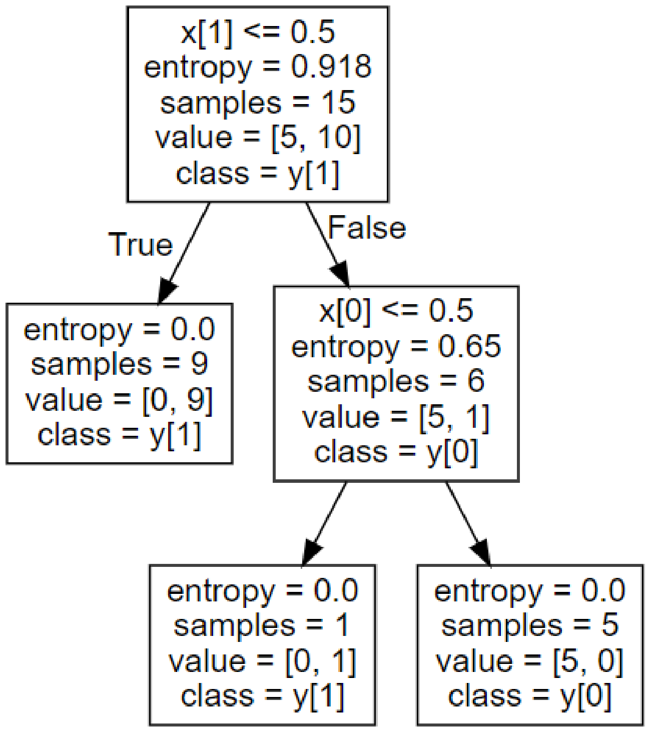
8) Train the model using ID3 Algorithm.

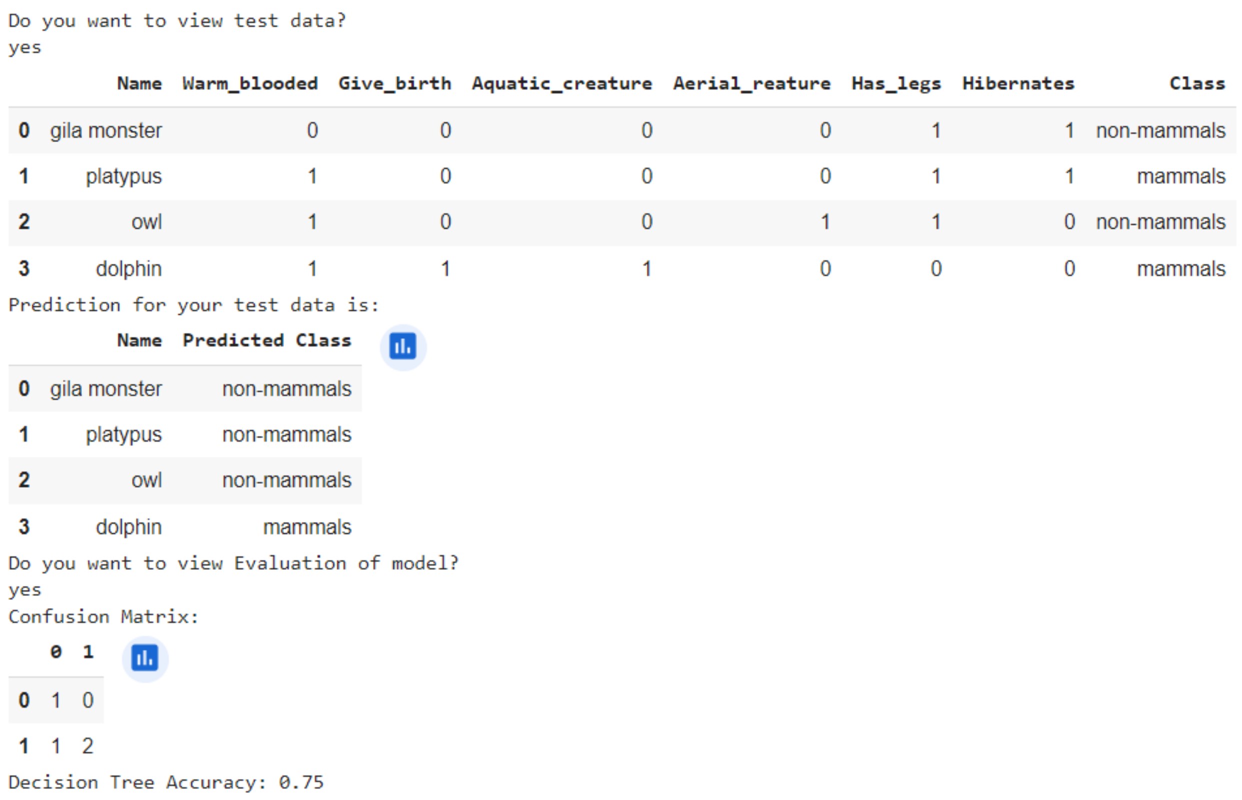
➢
```
import pandas as pd
from IPython.display import display
from sklearn import tree
from sklearn.tree import export_graphviz
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

def toy_dataset():
  animal=[['human',1, 1, 0, 0,1,0,'mammals'], ['python',0,0,0,0,0,1,'reptiles'],
  ['salmon',0,0,1,0,0,0,'fishes'], ['whale',1,1,1,0,0,0,'mammals'],
['frog',0,0,1,0,1,1,'amphibians'], ['komodo',0,0,0,0,1,0,'reptiles'], ['bat',1,1,0,1,1,1,'mammals'],
['pigeon',1,0,0,1,1,0,'birds'], ['cat',1,1,0,0,1,0,'mammals'], ['leopard shark',0,1,1,0,0,0,'fishes'],
  ['turtle',0,0,1,0,1,0,'reptiles'], ['penguin',1,0,1,0,1,0,'birds'],
  ['porcupine',1,1,0,0,1,1,'mammals'], ['eel',0,0,1,0,0,0,'fishes'],
['salamander',0,0,1,0,1,1,'amphibians']]
  titles=['Name','Warm_blooded','Give_birth','Aquatic_creature','Aerial_reature',
'Has_legs','Hibernates','Class']
  data=pd.DataFrame(animal,columns=titles)
  data['Class'] = data['Class'].replace(['fishes','birds','amphibians','reptiles'],'non-mammals')
  print("Do you want to view data?")
  choice=input()
  if choice=='yes':
    display(data)
  return data

  def build_model(data):
    Y = data['Class']
    X= data.drop(['Name', 'Class'],axis=1)
    clf = tree.DecisionTreeClassifier (criterion='entropy',max_depth=3)
    clf.fit(X, Y)
    return clf

def prediction_using_model(clf):
  testData = [['gila monster',0,0,0,0,1,1,'non-
mammals'],['platypus',1,0,0,0,1,1,'mammals'],['owl',1,0,0,1,1,0,'non-
mammals'],['dolphin',1,1,1,0,0,0,'mammals']]
  titles = ['Name', 'Warm_blooded','Give_birth','Aquatic_creature','Aerial_reature','Has_legs',
'Hibernates','Class']
  testData = pd.DataFrame(testData, columns=titles)
  print("Do you want to view test data?")
  choice=input()
  if choice=='yes':
    display(testData)
  #Splitting test data
  y_test=testData['Class']
  x_test=testData.drop(['Name','Class'],axis=1)
  y_pred = clf.predict(x_test)
  predictions = pd.concat([testData['Name'],pd.Series(y_pred,name='Predicted Class')],
axis=1)
  print("Prediction for your test data is:")
  display(predictions)
  #Model evaluation
  print("Do you want to view Evaluation of model?")
  choice=input()
  if choice=='yes':
    model_evaluation(y_pred,y_test)
  else:
    quit()

def model_evaluation(y_pred,y_test):
  print("Confusion Matrix:")
  report=(confusion_matrix(y_test, y_pred))
  cf=pd.DataFrame(report).transpose()
  display(cf)
  score =accuracy_score(y_test,y_pred)
  print('Decision Tree Accuracy:',score)
  print("Classification report:")
  report=(classification_report(y_test, y_pred, output_dict=True))
  df = pd.DataFrame(report).transpose()
  display(df[['precision', 'recall','f1-score']].head(2))

def main():
  data=toy_dataset()
  model= build_model(data)
  # to visualize tree install both graphviz and pydotplus
  dot_data = tree.export_graphviz(model,'tree.dot',class_names=True)
  !dot-Tpng tree.dot -o tree.png"
  print("Your decision tree constructed successfully, check the current directory for tree.png")
  prediction_using_model(model)
main()
```

```
Do you want to view data?
yes
```

	Name	Warm_blooded	Give_birth	Aquatic_creature	Aerial_reature	Has_legs	Hibernates	Class
0	human	1	1	0	0	1	0	mammals
1	python	0	0	0	0	0	1	non-mammals
2	salmon	0	0	1	0	0	0	non-mammals
3	whale	1	1	1	0	0	0	mammals
4	frog	0	0	1	0	1	1	non-mammals
5	komodo	0	0	0	0	1	0	non-mammals
6	bat	1	1	0	1	1	1	mammals
7	pigeon	1	0	0	1	1	0	non-mammals
8	cat	1	1	0	0	1	0	mammals
9	leopard shark	0	1	1	0	0	0	non-mammals
10	turtle	0	0	1	0	1	0	non-mammals
11	penguin	1	0	1	0	1	0	non-mammals
12	porcupine	1	1	0	0	1	1	mammals
13	eel	0	0	1	0	0	0	non-mammals
14	salamander	0	0	1	0	1	1	non-mammals

```
/bin/bash: -c: line 1: unexpected EOF while looking for matching `"'
/bin/bash: -c: line 2: syntax error: unexpected end of file
Your decision tree constructed successfully, check the current directory for tree.png
```





```
Do you want to view test data?
yes
```

	Name	Warm_blooded	Give_birth	Aquatic_creature	Aerial_reature	Has_legs	Hibernates	Class
0	gila monster	0	0	0	0	1	1	non-mammals
1	platypus	1	0	0	0	1	1	mammals
2	owl	1	0	0	1	1	0	non-mammals
3	dolphin	1	1	1	0	0	0	mammals

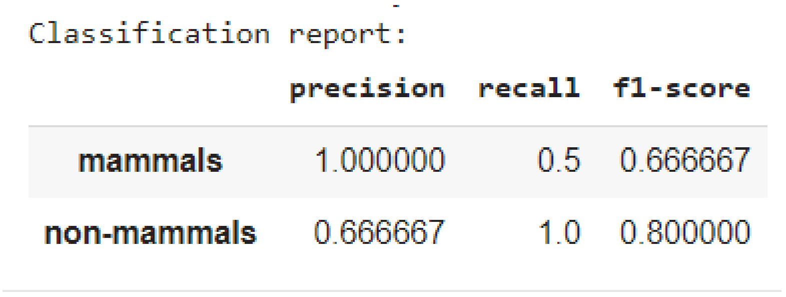
```
Prediction for your test data is:
```

	Name	Predicted Class
0	gila monster	non-mammals
1	platypus	non-mammals
2	owl	non-mammals
3	dolphin	mammals

```
Do you want to view Evaluation of model?
yes
Confusion Matrix:
```

	0	1
0	1	0
1	1	2

```
Decision Tree Accuracy: 0.75
```

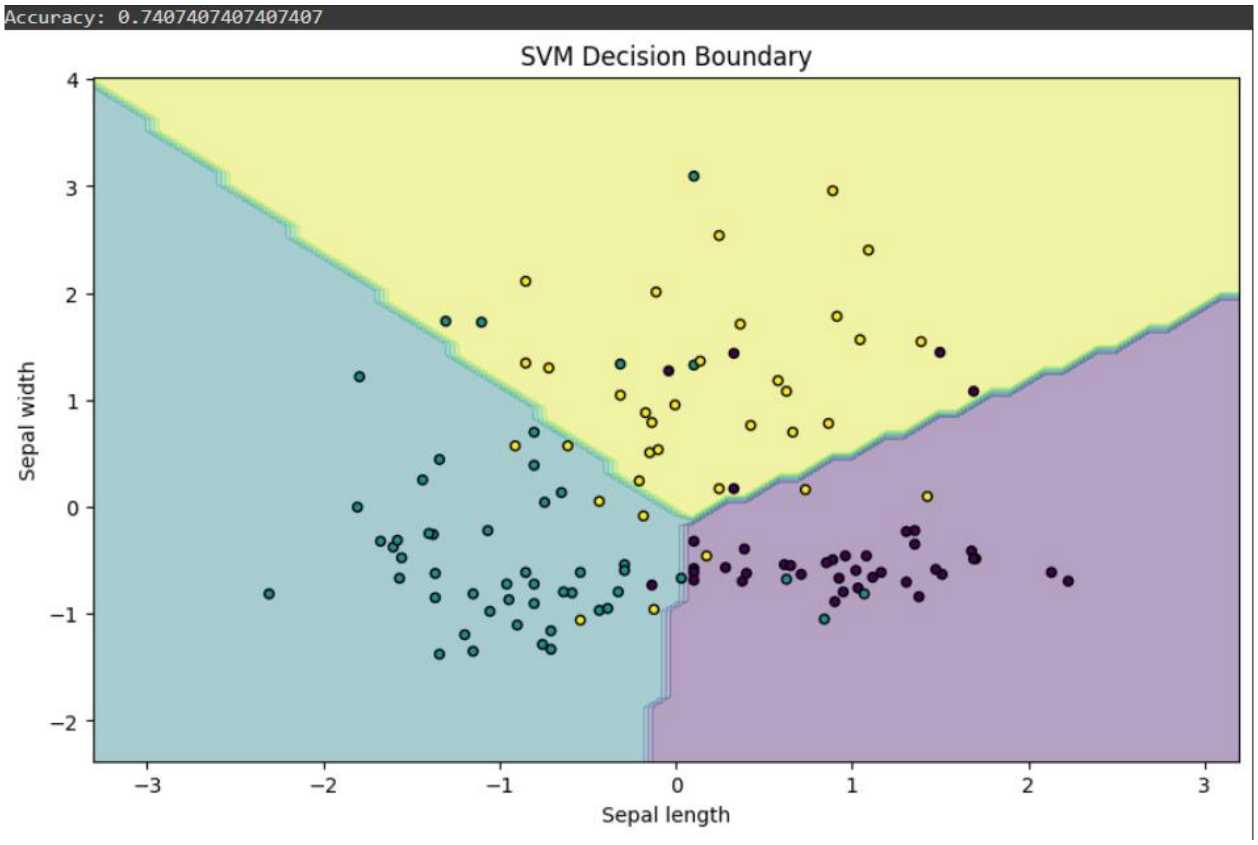


```
Classification report:
```

	precision	recall	f1-score
mammals	1.000000	0.5	0.666667
non-mammals	0.666667	1.0	0.800000

9) Train the model using Support Vector Machine.

➢
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Wine dataset
wine = datasets.load_wine()
X = wine.data[:, :2]  # We only take the first two features for visualization purposes
y = wine.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize features by removing the mean and scaling to unit variance
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Train the Support Vector Machine (SVM) model
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train, y_train)

# Predict the labels for test set
y_pred = svm_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Visualize the decision boundary
def plot_decision_boundary(X, y, model):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.title('SVM Decision Boundary')

# Plot decision boundary
plt.figure(figsize=(10, 6))
plot_decision_boundary(X_train, y_train, svm_classifier)
plt.show()
```

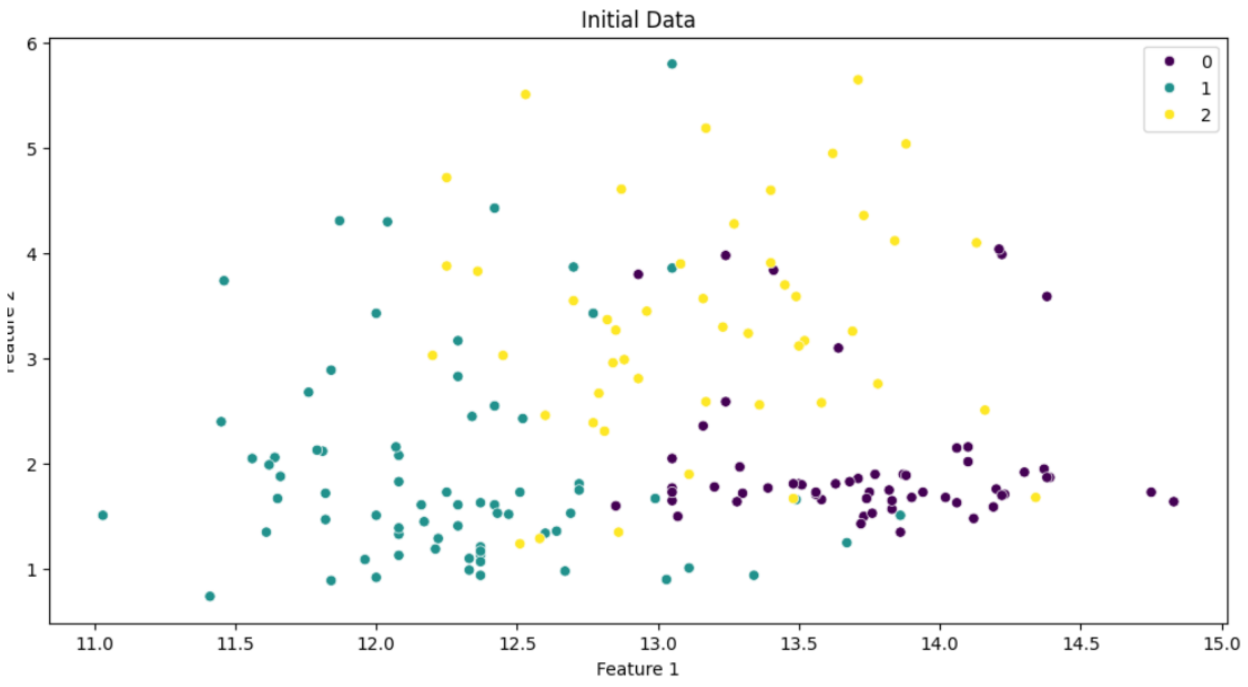


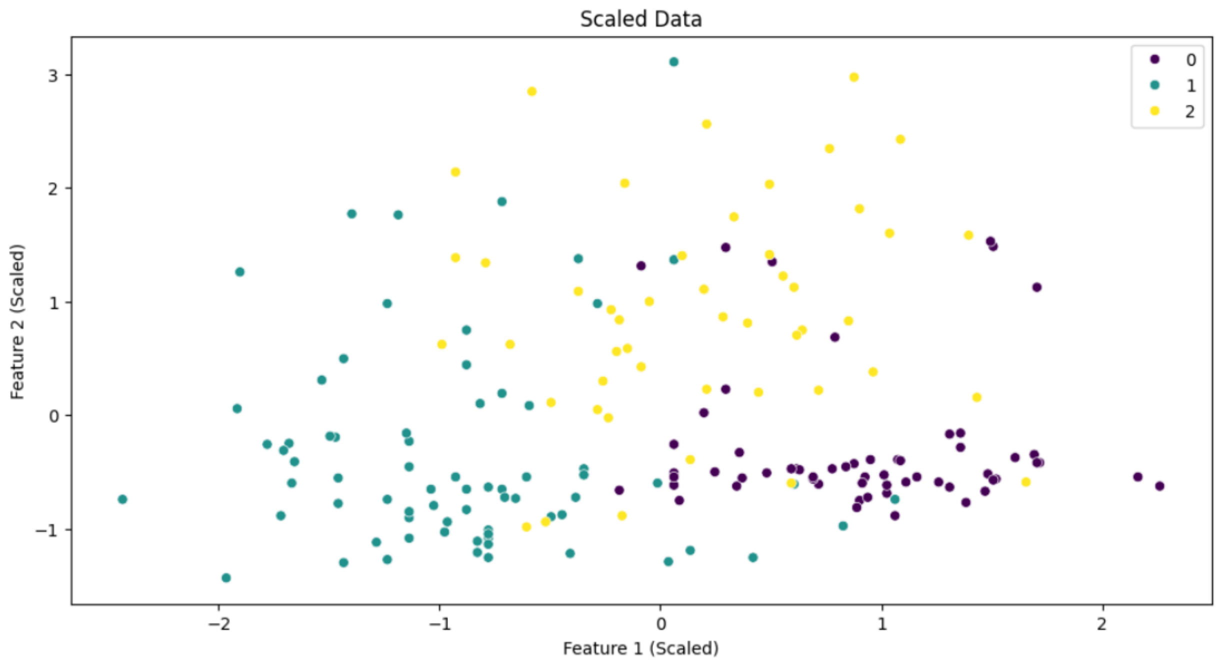
```
Accuracy: 0.7407407407407407
```

10) WAP to preprocess the dataset for model training.

```
# //Data Preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

# Load the Wine dataset
wine = datasets.load_wine()
X = wine.data[:, :2]  # We only take the first two features for visualization purposes
y = wine.target

# Convert initial data to DataFrame for visualization
df_initial = pd.DataFrame(X, columns=['Feature 1', 'Feature 2'])

# Plot the initial data
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Feature 1', y='Feature 2', data=df_initial, hue=y, palette='viridis')
plt.title('Initial Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

```
# Perform scaling (standardization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert to DataFrame for visualization
df_scaled = pd.DataFrame(X_scaled, columns=['Feature 1', 'Feature 2'])

# Plot the scaled data
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Feature 1', y='Feature 2', data=df_scaled, hue=y, palette='viridis')
plt.title('Scaled Data')
plt.xlabel('Feature 1 (Scaled)')
plt.ylabel('Feature 2 (Scaled)')
plt.show()
```

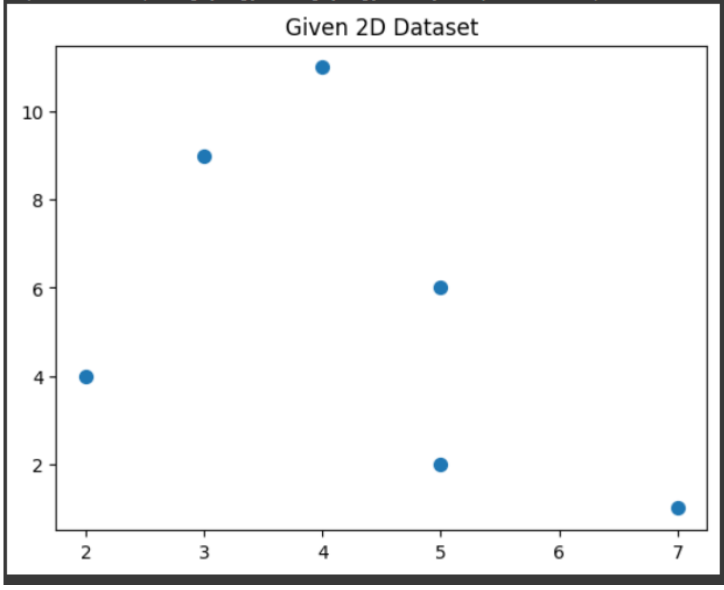




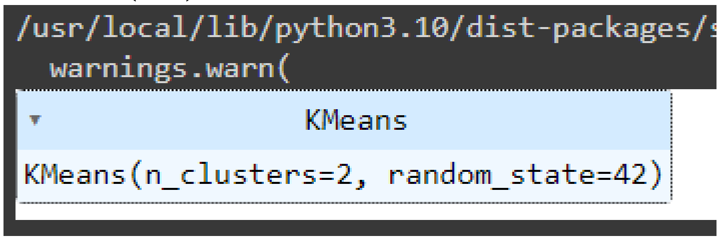
11) WAP to make a cluster using K-Mean algorithm.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Given dataset
data = np.array([[2, 4], [5, 6], [3, 9], [4, 11], [5, 2], [7, 1]])

# Visualize the dataset
plt.scatter(data[:, 0], data[:, 1], s=50, cmap='viridis')
plt.title('Given 2D Dataset')
plt.show()
```

```
<ipython-input-3-d6785daa1f32>:2: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(data[:, 0], data[:, 1], s=50, cmap='viridis')
```



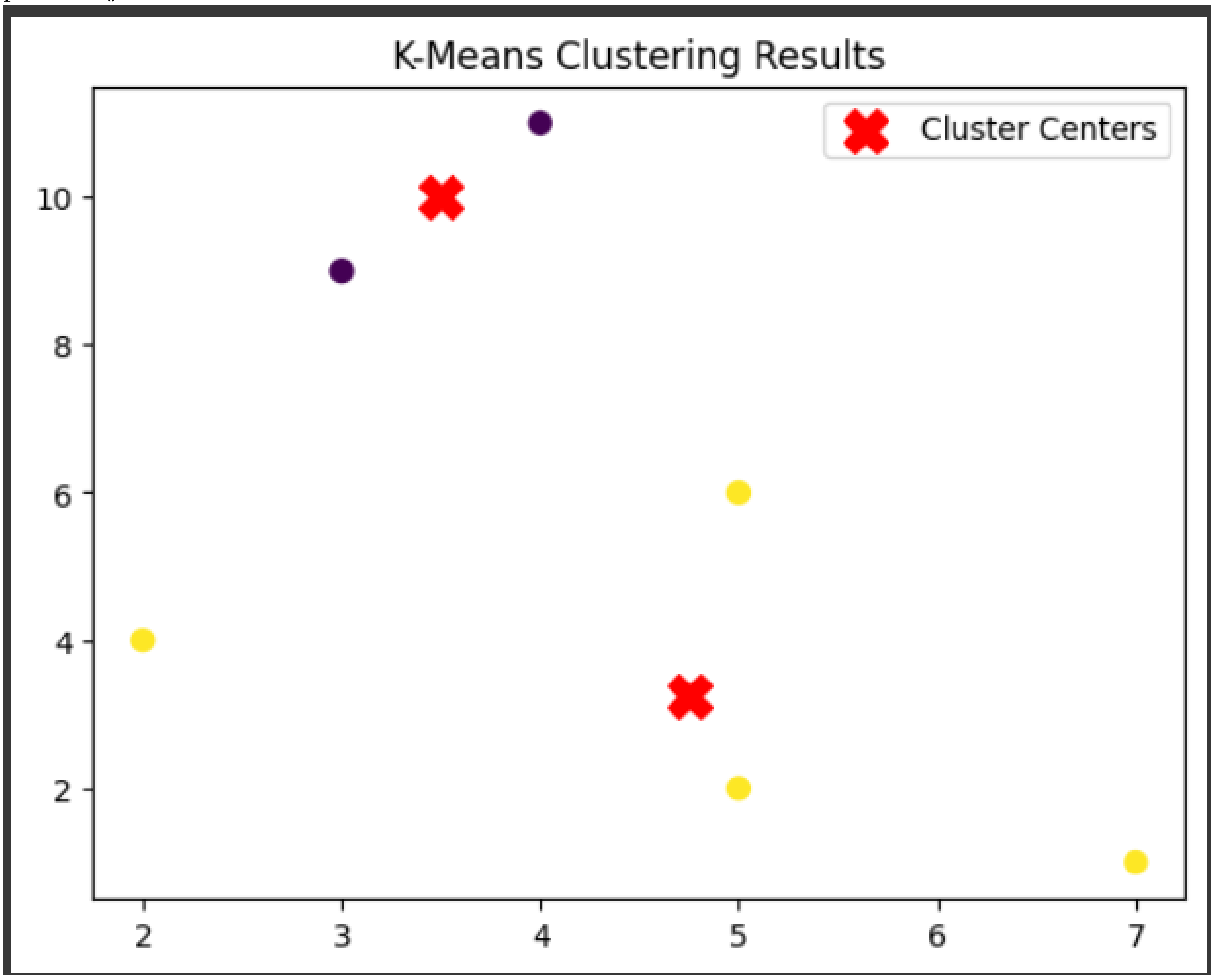
```
# Apply k-means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(data)
```



```
/usr/local/lib/python3.10/dist-packages/
  warnings.warn(
```

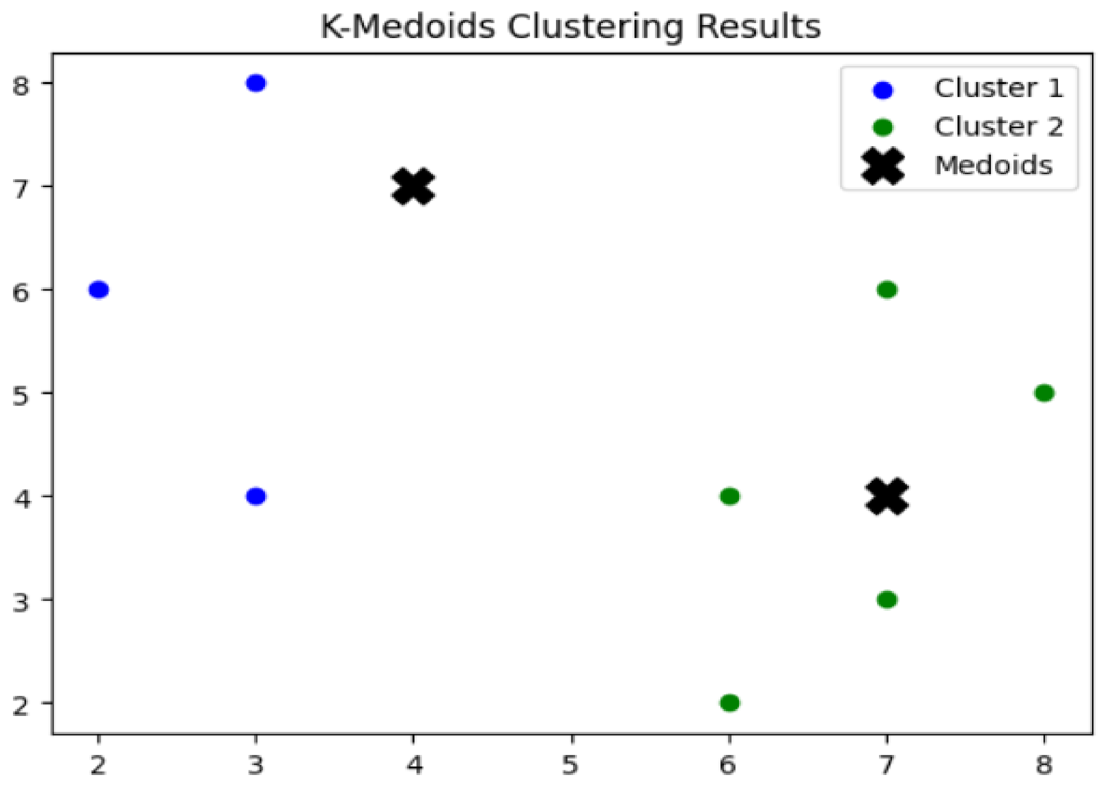
```
▾ KMeans
KMeans(n_clusters=2, random_state=42)
```

```
# Get cluster centers and labels
centers = kmeans.cluster_centers_
labels = kmeans.labels_

# Visualize the clustered data
plt.scatter(data[:, 0], data[:, 1], c=labels, s=50, cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X', s=200, label='Cluster Centers')
plt.title('K-Means Clustering Results')
plt.legend()
plt.show()
```



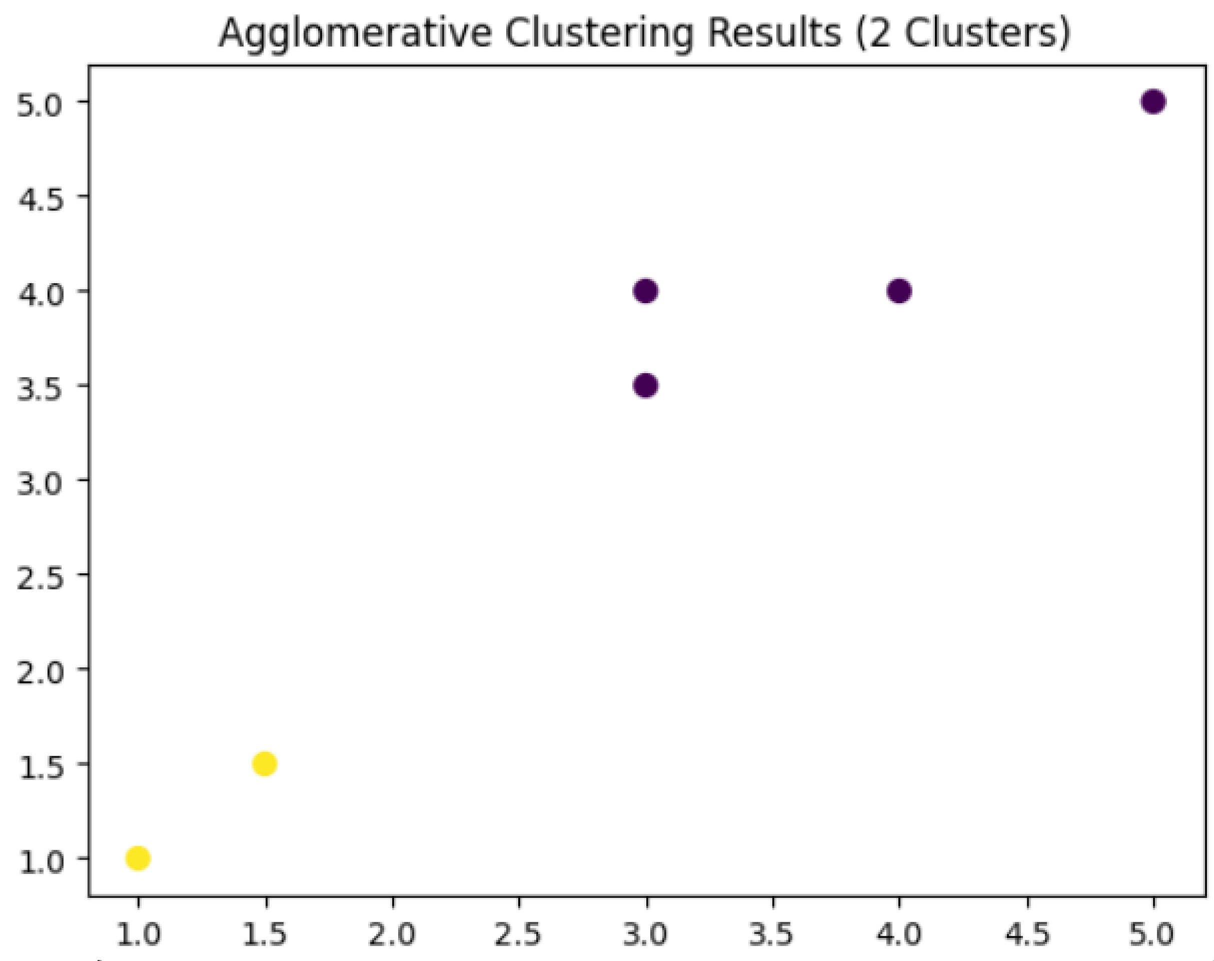
12) WAP to make a cluster using K-Medoid algorithm.

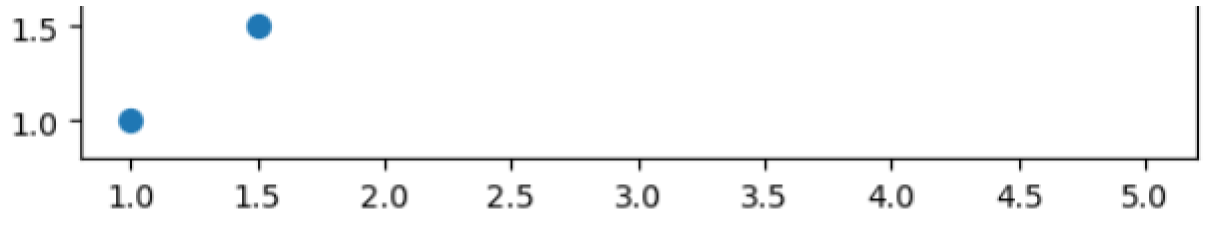
➢
```
import numpy as np
import matplotlib.pyplot as plt
from pyclustering.cluster import kmedoids
from pyclustering.cluster import cluster_visualizer

# Given dataset
data = np.array([[2, 6], [3, 4], [3, 8], [4, 7], [6, 2], [6, 4], [7, 3], [7, 4], [8, 5], [7, 6]])

k = 2
initial_medoids = [0, 1]
kmedoids_instance = kmedoids.kmedoids(data, initial_medoids)

kmedoids_instance.process()
clusters = kmedoids_instance.get_clusters()
medoids = kmedoids_instance.get_medoids()
# Visualize the clustered data
colors = ['blue', 'green', 'red', 'cyan', 'magenta', 'yellow']
for cluster_index, cluster in enumerate(clusters):
    cluster_points = data[cluster]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], c=colors[cluster_index], label=f'Cluster
{cluster_index + 1}')

plt.scatter(data[medoids, 0], data[medoids, 1], c='black', marker='X', s=200, label='Medoids')
plt.title('K-Medoids Clustering Results')
plt.legend()
plt.show()
```



1. WAP to make a cluster using Agglomerative and divisive algorithm.

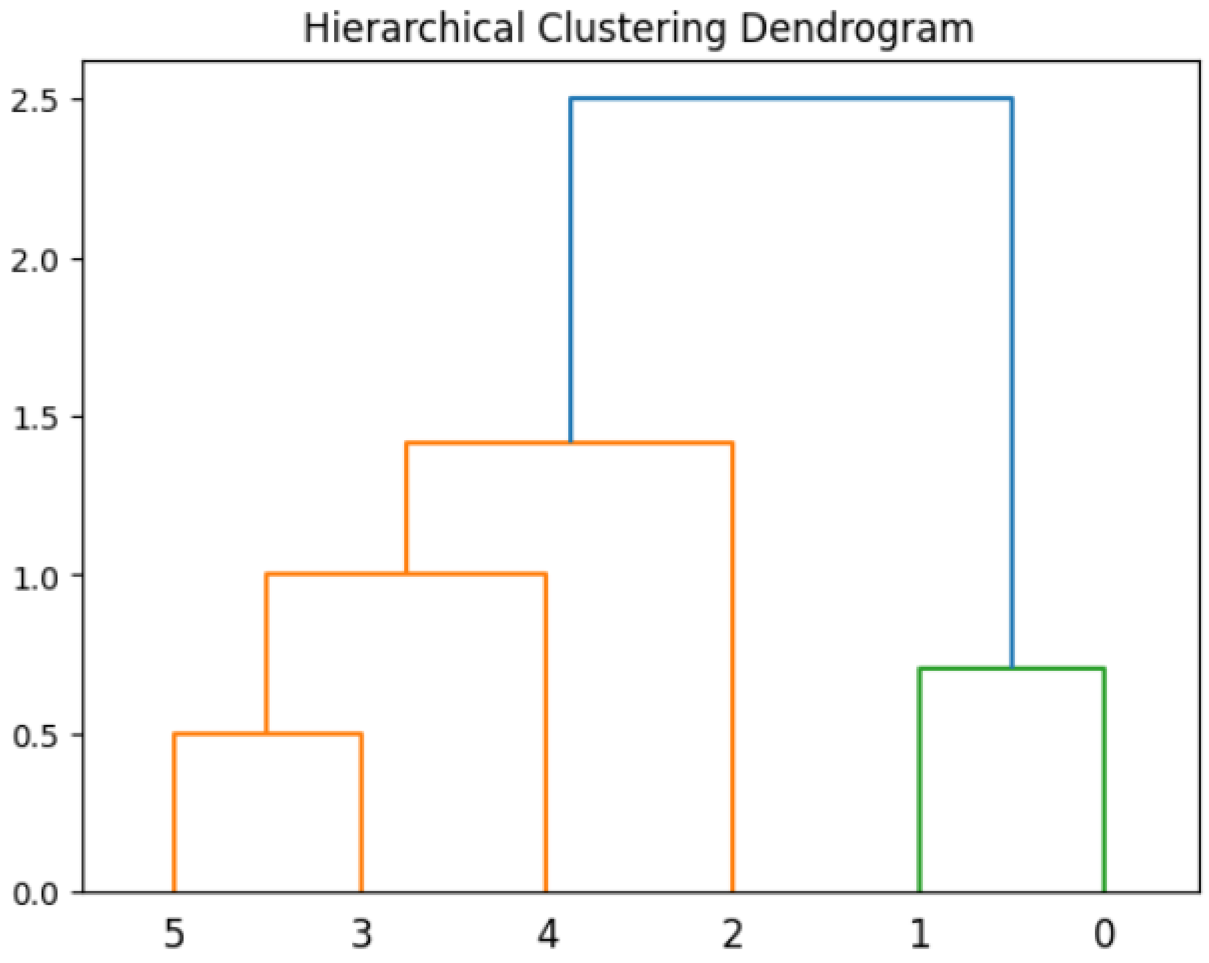
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Given dataset
```





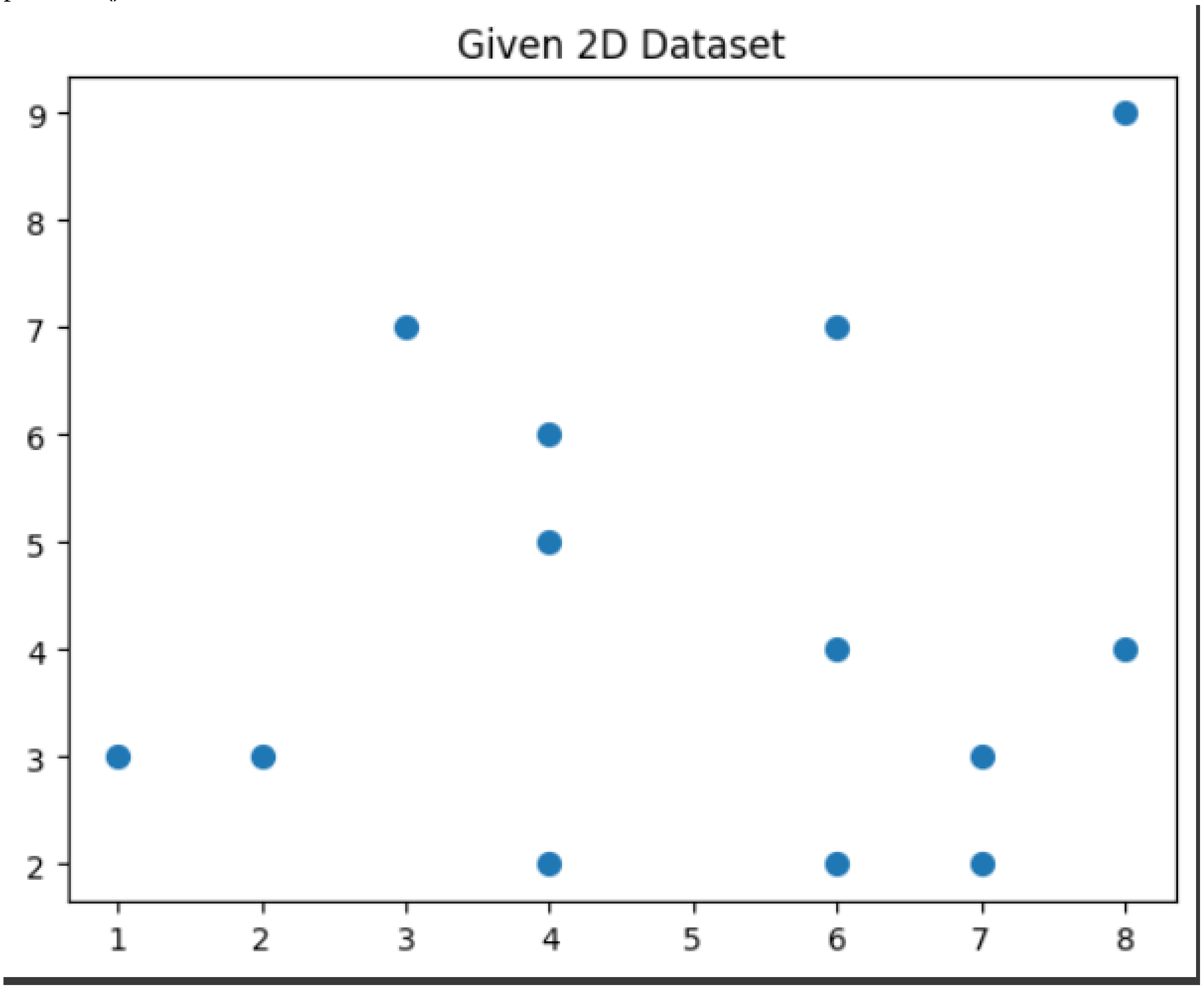
```
# Apply agglomerative clustering with 2 clusters
n_clusters = 2
agg_cluster = AgglomerativeClustering(n_clusters=n_clusters)
labels = agg_cluster.fit_predict(data)

# Visualize the clustered data
plt.scatter(data[:, 0], data[:, 1], c=labels, s=50, cmap='viridis')
plt.title('Agglomerative Clustering Results (2 Clusters)')
plt.show()

# Plot dendrogram for hierarchical structure (optional)
```

```
linked = linkage(data, 'single')  # You can change the linkage method if needed
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.show()
```



13) WAP to make a cluster using DB scan algorithm.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import DBSCAN

data = np.array([[3, 7], [4, 6], [6, 4], [7, 3], [6, 2], [7, 2], [8, 4],[1,3], [4, 2], [2, 3], [6, 7], [8, 9], [4, 5]])# importing the dataset

# Visualize the dataset
plt.scatter(data[:, 0], data[:, 1], s=50, cmap='viridis')
plt.title('Given 2D Dataset')
plt.show()
```

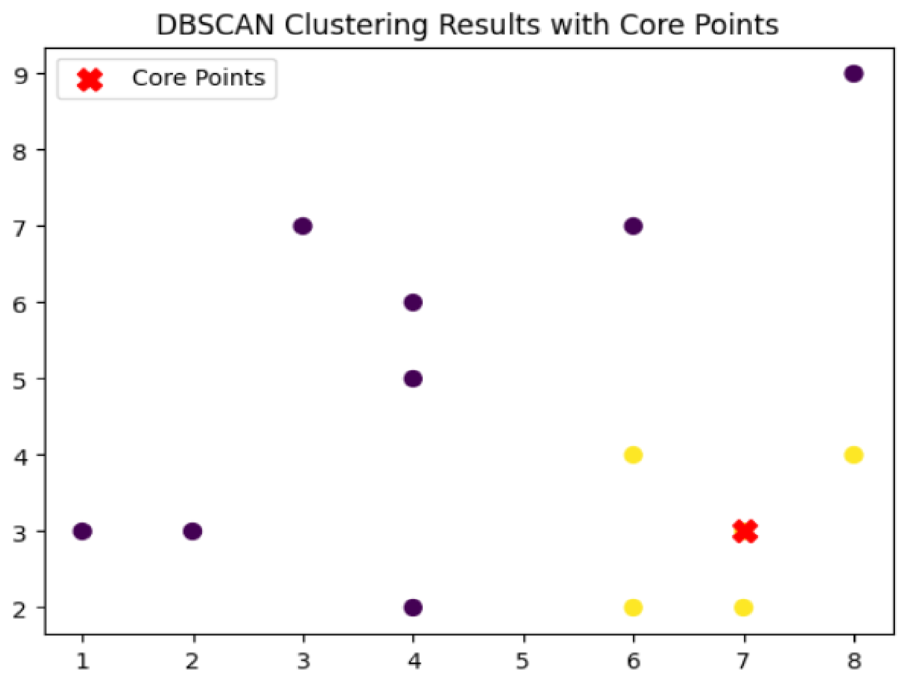


```
# Apply DBSCAN clustering
eps = 1.9  # epsilon, the maximum distance between two samples for one to be considered as in the neighborhood of the other
min_samples = 4  # the number of samples (or total weight) in a neighborhood for a point to be considered as a core point
dbscan = DBSCAN(eps=eps, min_samples=min_samples)
labels = dbscan.fit_predict(data)
```

```
# Identify and print core points
core_points_indices = dbscan.core_sample_indices_
print(f"Core Points Indices: {core_points_indices}")
# Display cluster assignments (clusters are represented by numbers, and noise is labeled as -1)
for i, label in enumerate(labels):
    print(f'Data point {i+1} is assigned to Cluster {label}')
```

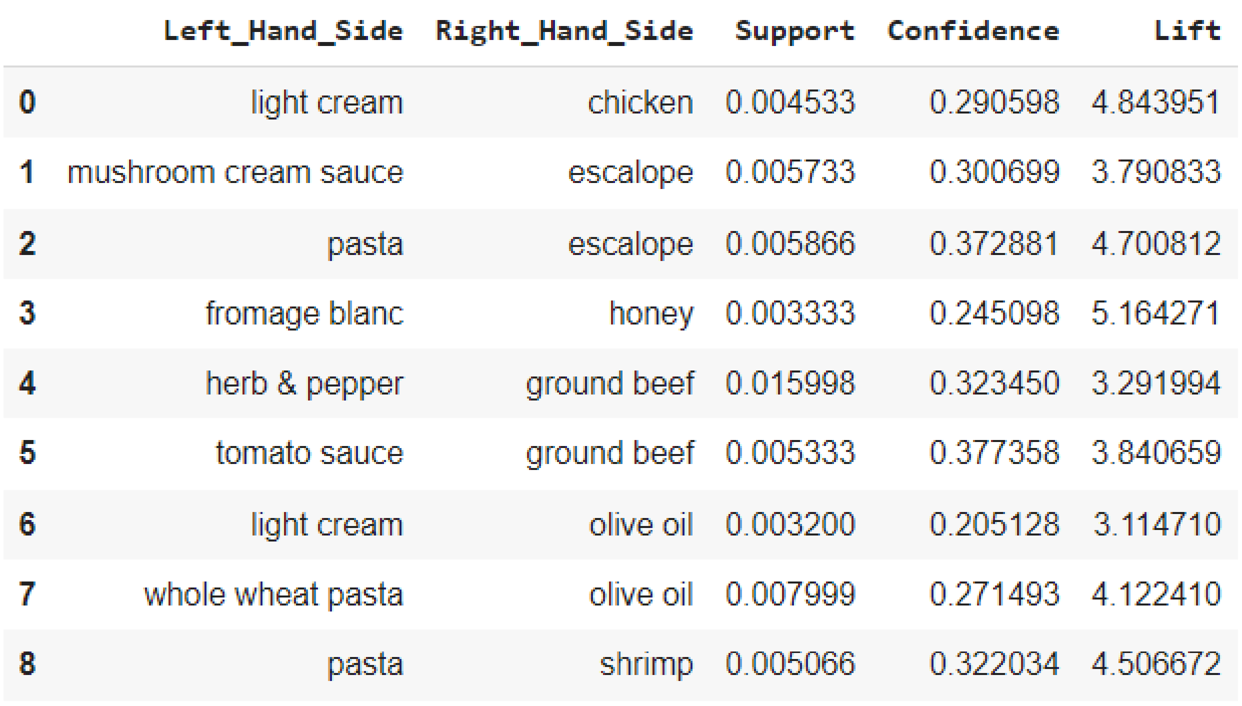
```
Core Points Indices: [3]
Data point 1 is assigned to Cluster -1
Data point 2 is assigned to Cluster -1
Data point 3 is assigned to Cluster 0
Data point 4 is assigned to Cluster 0
Data point 5 is assigned to Cluster 0
Data point 6 is assigned to Cluster 0
Data point 7 is assigned to Cluster 0
Data point 8 is assigned to Cluster -1
Data point 9 is assigned to Cluster -1
Data point 10 is assigned to Cluster -1
Data point 11 is assigned to Cluster -1
Data point 12 is assigned to Cluster -1
Data point 13 is assigned to Cluster -1
```

```
# Visualize the clustered data with core points
plt.scatter(data[:, 0], data[:, 1], c=labels, s=50, cmap='viridis', marker='o')
plt.scatter(data[core_points_indices, 0], data[core_points_indices, 1], c='red', s=100,
marker='X', label='Core Points')
plt.title('DBSCAN Clustering Results with Core Points')
plt.legend()
plt.show()
```



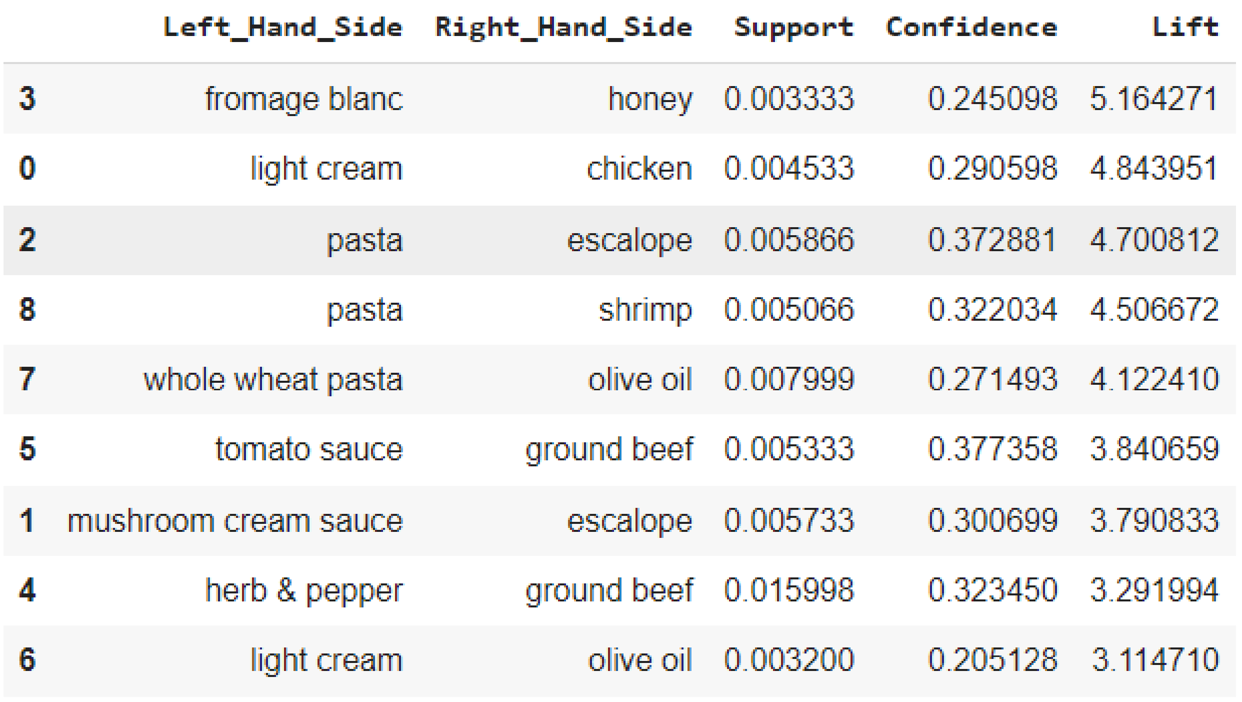
14) WAP to demonstrate apriori algorithm.

➢
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

Data = pd.read_csv('/content/drive/MyDrive/DWDM/Market_Basket_Optimisation .csv',
header = None)

# Intializing the list
transacts = []
# populating a list of transactions
for i in range(0, 7501):
  transacts.append([str(Data.values[i,j]) for j in range(0, 20)])

from apyori import apriori
rule = apriori(transactions = transacts, min_support = 0.003, min_confidence = 0.2, min_lift =
3, min_length = 2, max_length = 2)

output = list(rule)# returns a non-tabular output
# putting output into a pandas dataframe
def inspect(output):
    lhs         = [tuple(result[2][0][0])[0] for result in output]
    rhs         = [tuple(result[2][0][1])[0] for result in output]
    support    = [result[1] for result in output]
    confidence = [result[2][0][2] for result in output]
    lift       = [result[2][0][3] for result in output]
    return list(zip(lhs, rhs, support, confidence, lift))
output_DataFrame = pd.DataFrame(inspect(output), columns = ['Left_Hand_Side',
'Right_Hand_Side', 'Support', 'Confidence', 'Lift'])

output_DataFrame
```



	Left_Hand_Side	Right_Hand_Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
2	pasta	escalope	0.005866	0.372881	4.700812
3	fromage blanc	honey	0.003333	0.245098	5.164271
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
6	light cream	olive oil	0.003200	0.205128	3.114710
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
8	pasta	shrimp	0.005066	0.322034	4.506672

```
output_DataFrame.nlargest(n = 10, columns = 'Lift')
```

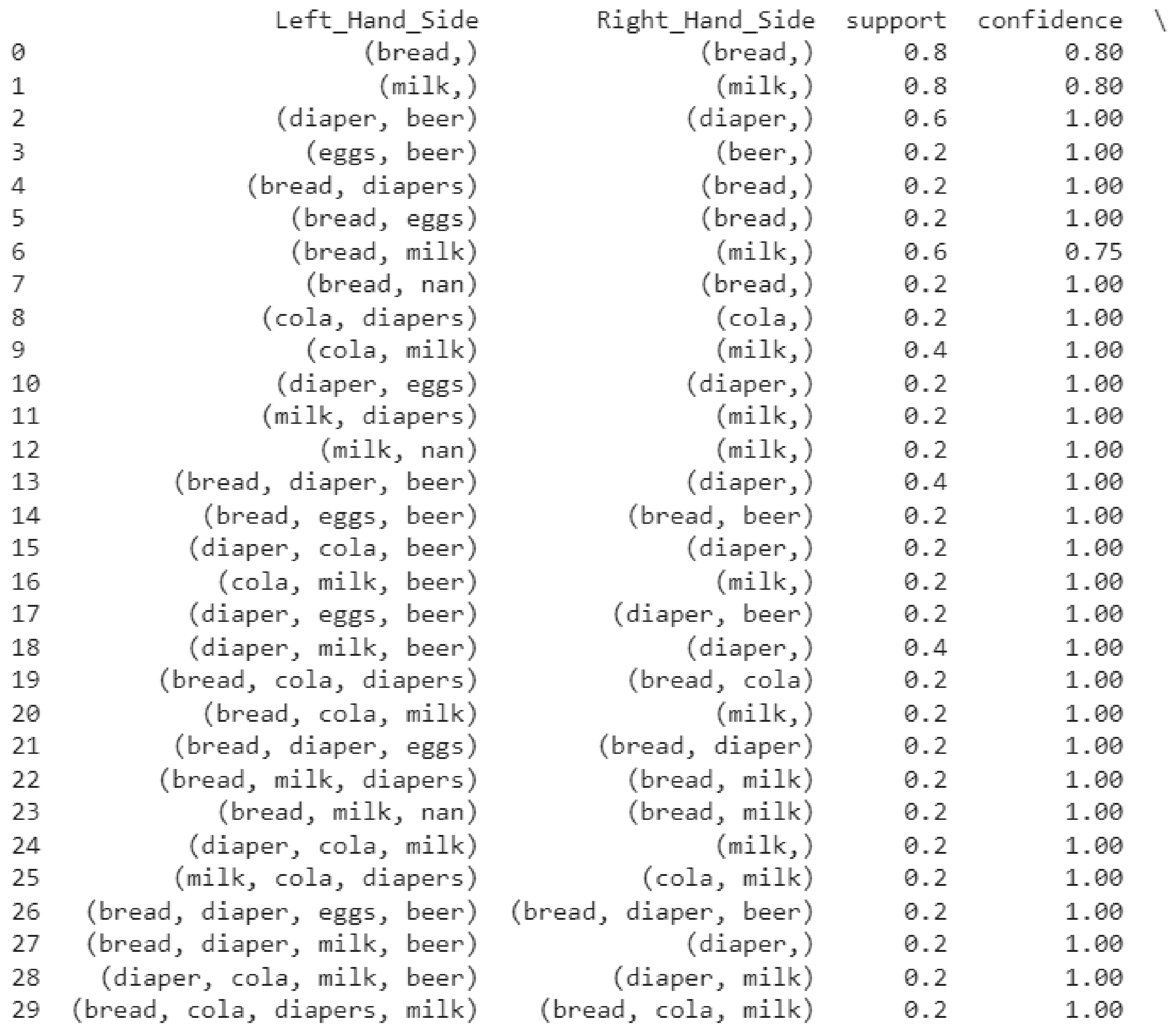


	Left_Hand_Side	Right_Hand_Side	Support	Confidence	Lift
3	fromage blanc	honey	0.003333	0.245098	5.164271
0	light cream	chicken	0.004533	0.290598	4.843951
2	pasta	escalope	0.005866	0.372881	4.700812
8	pasta	shrimp	0.005066	0.322034	4.506672
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
6	light cream	olive oil	0.003200	0.205128	3.114710

```
from apyori import apriori
import pandas as pd

# Define your own dataset (list of transactions)
transactions = [
    ['bread', 'milk','nan','nan'],
    ['bread', 'diaper', 'beer', 'eggs'],
    ['milk', 'diaper', 'beer', 'cola'],
    ['bread', 'milk', 'diaper', 'beer'],
    ['bread', 'milk', 'diapers', 'cola'],]
```

15)# Apply Apriori algorithm

```
rules = apriori(transactions, min_support=0.2, min_confidence=0.7)

# Convert the rules to a list and then to a DataFrame for better visualization
rules_list = list(rules)
df_rules = pd.DataFrame([{
    'Left_Hand_Side': tuple(rule.items),
    'Right_Hand_Side': tuple(rule.ordered_statistics[0].items_add),
    'support': rule.support,
    'confidence': rule.ordered_statistics[0].confidence,
    'lift': rule.ordered_statistics[0].lift
} for rule in rules_list])

# Print the DataFrame
print(df_rules)
```

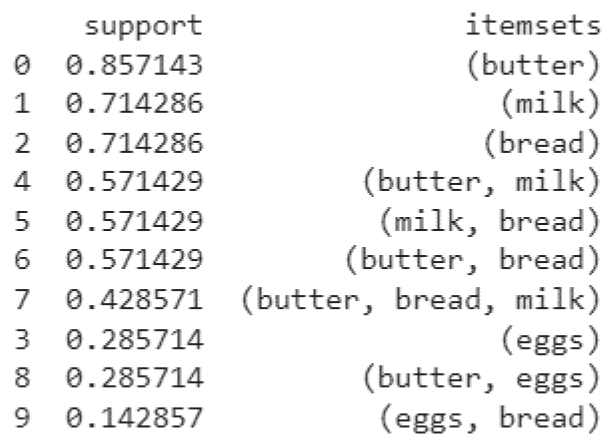


```
                 Left_Hand_Side       Right_Hand_Side  support  confidence  \
0                      (bread,)              (bread,)      0.8        0.80
1                       (milk,)               (milk,)      0.8        0.80
2                (diaper, beer)             (diaper,)      0.6        1.00
3                  (eggs, beer)               (beer,)      0.2        1.00
4              (bread, diapers)              (bread,)      0.2        1.00
5                 (bread, eggs)              (bread,)      0.2        1.00
6                 (bread, milk)               (milk,)      0.6        0.75
7                  (bread, nan)              (bread,)      0.2        1.00
8               (cola, diapers)               (cola,)      0.2        1.00
9                  (cola, milk)               (milk,)      0.4        1.00
10               (diaper, eggs)             (diaper,)      0.2        1.00
11              (milk, diapers)               (milk,)      0.2        1.00
12                  (milk, nan)               (milk,)      0.2        1.00
13         (bread, diaper, beer)             (diaper,)      0.4        1.00
14           (bread, eggs, beer)         (bread, beer)      0.2        1.00
15          (diaper, cola, beer)             (diaper,)      0.2        1.00
16            (cola, milk, beer)               (milk,)      0.2        1.00
17          (diaper, eggs, beer)        (diaper, beer)      0.2        1.00
18          (diaper, milk, beer)             (diaper,)      0.4        1.00
19         (bread, cola, diapers)         (bread, cola)      0.2        1.00
20           (bread, cola, milk)               (milk,)      0.2        1.00
21         (bread, diaper, eggs)       (bread, diaper)      0.2        1.00
22        (bread, milk, diapers)         (bread, milk)      0.2        1.00
23            (bread, milk, nan)         (bread, milk)      0.2        1.00
24          (diaper, cola, milk)               (milk,)      0.2        1.00
25         (milk, cola, diapers)          (cola, milk)      0.2        1.00
26   (bread, diaper, eggs, beer)  (bread, diaper, beer)      0.2        1.00
27   (bread, diaper, milk, beer)             (diaper,)      0.2        1.00
28    (diaper, cola, milk, beer)        (diaper, milk)      0.2        1.00
29 (bread, cola, diapers, milk)    (bread, cola, milk)      0.2        1.00
```

```
        lift
0   1.000000
1   1.000000
2   1.666667
3   1.666667
4   1.250000
5   1.250000
6   0.937500
7   1.250000
8   2.500000
9   1.250000
10  1.666667
11  1.250000
12  1.250000
13  1.666667
14  2.500000
15  1.666667
16  1.250000
17  1.666667
18  1.666667
19  5.000000
20  1.250000
21  2.500000
22  1.666667
23  1.666667
24  1.250000
```

16) WAP to demonstrate FP growth algorithm.

➢
```
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd

dataset =
[['milk','bread','butter'],['milk','bread'],['milk','butter'],['milk','bread','butter'],['bread','butter'],['
milk','bread','butter','eggs'],['eggs','butter']]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns= te.columns_)

frequent_itemsets = fpgrowth(df, min_support = 0.05, use_colnames = True)
frequent_itemsets = frequent_itemsets.sort_values(by='support', ascending=False)

print(frequent_itemsets.head(10))
```



```
    support                itemsets
0  0.857143                (butter)
1  0.714286                  (milk)
2  0.714286                 (bread)
4  0.571429          (butter, milk)
5  0.571429           (milk, bread)
6  0.571429         (butter, bread)
7  0.428571  (butter, bread, milk)
3  0.285714                  (eggs)
8  0.285714          (butter, eggs)
9  0.142857           (eggs, bread)
```