Machine Learning Lab Manual

BCA 6th Sem

LIST OF PROGRAMS
1. Install and set up Python and essential libraries like NumPy and pandas
2. Introduce scikit-learn as a machine learning library. 3. Install and set up scikit-learn and other necessary tools.
4. Write a program to Load and explore the dataset of .CVS and excel files using pandas.
5. Write a program to Visualize the dataset to gain insights using Matplotlib or Seaborn by plotting scatter plots, bar charts.
6. Write a program to Handle missing data, encode categorical variables, and perform feature scaling.
7. Write a program to implement a k-Nearest Neighbours (k-NN) classifier using scikit-learn and Train the classifier on the dataset and evaluate its performance.
8. Write a program to implement a linear regression model for regression tasks and Train the model on a dataset with continuous target variables.
9. Write a program to implement a decision tree classifier using scikit-learn and visualize the decision tree and understand its splits.
10. Write a program to Implement K-Means clustering and Visualize clusters.

1. **install and set up Python and essential libraries like NumPy and Pandas**.

Install Python: If you haven't already installed Python, you can download it from the official website:

To verify (terminal)

python --version

Install pip: pip is a package manager for Python that allows you to easily install and manage libraries. Most recent versions of Python come with pip pre-installed. You can verify if pip is installed by running the following command in your terminal or command prompt:

pip --version

Install NumPy and pandas: Once you have Python and pip installed, you can use pip to install NumPy and pandas by running the following commands in your terminal or command prompt:

#In terminal

pip install numpy

pip install pandas

This will download and install NumPy and Pandas along with any dependencies they require.

Verify installation: After installing NumPy and pandas, you can verify that they were installed correctly by running the following commands in Python's interactive mode or a Python script:

```
import numpy
import pandas
print(numpy.__version__)
print(pandas.__version__)
```

These commands should print the versions of NumPy and pandas that were installed.

Output:

```
(/opt/anaconda3) apple@Apples-MacBook-Pro ~ % python --version
Python 3.9.7
(/opt/anaconda3) apple@Apples-MacBook-Pro ~ % pip --version
pip 21.2.4 from /opt/anaconda3/lib/python3.9/site-packages/pip (python 3.9)
(/opt/anaconda3) apple@Apples-MacBook-Pro ~ % pip install numpy

Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.9/site-packages (1.22.4)
(/opt/anaconda3) apple@Apples-MacBook-Pro ~ % pip install pandas
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.9/site-packages (1.3.4)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/anaconda3/lib/python3.9/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/python3.9/site-packages (from pandas) (2021.3)
Requirement already satisfied: numpy>=1.17.3 in /opt/anaconda3/lib/python3.9/site-packages (from pandas) (1.22.4)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.9/site-packages (from python-dateutil>=2.7.3->pandas
) (1.16.0)
(/opt/anaconda3) apple@Apples-MacBook-Pro ~ %
```

```
import numpy
import pandas

print(numpy.__version__)
print(pandas.__version__)
```

```
1.22.4
1.3.4
```

2. **Introduce sci-kit-learn as a machine learning library.**

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon **NumPy, SciPy and Matplotlib.**

Installation

If you already installed NumPy and Scipy, the following are the two easiest ways to install scikit-learn −

Using pip

The following command can be used to install sci-kit-learn via pip

pip install -U scikit-learn

Features

Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows −

Supervised Learning algorithms − Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.

Unsupervised Learning algorithms − On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

Clustering − This model is used for grouping unlabeled data.

Cross Validation − It is used to check the accuracy of supervised models on unseen data.

3. **Install and set up scikit-learn and other necessary tools.**

scikit-learn, a powerful Python library for machine learning. Here are the steps to set it up:
Install Python: If you haven't already installed Python, download and install the latest version of Python 3 from the official Python website.
Install scikit-learn using pip: Open your terminal or command prompt and run the following command:

```
pip install -U scikit-learn
```

To verify your installation, you can use the following commands:

```
python -m pip show scikit-learn
```

To see which version and where scikit-learn is installed

```
python -m pip freeze
```

To see all packages installed in the active virtual environment

```
import sklearn

import numpy

import pandas

import matplotlib

print(sklearn.__version__)

print(numpy.__version__)

print(pandas.__version__)

print(matplotlib.__version__)
```

These commands should print the versions of scikit-learn and other libraries that were installed.

Output:

```
]: import sklearn
   import numpy
   import pandas
   import matplotlib
   print(sklearn.__version__)
   print(numpy.__version__)
   print(pandas.__version__)
   print(matplotlib.__version__)
```

```
1.0.2
1.22.4
1.3.4
3.4.3
```

4. **Write a program to Load and explore the dataset of .CVS and excel files using pandas.**

```
import pandas as pd

def explore_dataset(file_path):

 # Check if the file is a CSV or Excel file

   if file_path.endswith('.csv'):

 # Load CSV file into a pandas DataFrame

      df = pd.read_csv(file_path)

   elif file_path.endswith('.xlsx'):

    # Load Excel file into a pandas DataFrame

      df = pd.read_excel(file_path)

   else:

      print("Unsupported file format. Please provide a CSV or Excel file.")

      return

   # Display basic information about the DataFrame

  print("Dataset information:")

   print(df.info())

   # Display the first few rows of the DataFrame

   print("\nFirst few rows of the dataset:")

   print(df.head())
```

```
    # Display summary statistics for numerical columns
    print("\nSummary statistics:")
    print(df.describe())

    # Display unique values for categorical columns
    print("\nUnique values for categorical columns:")
    for column in df.select_dtypes(include='object').columns:
        print(f"{column}: {df[column].unique()}")

# Example usage
file_path = 'IRIS.csv'
 # Change this to the path of your CSV or Excel file
explore_dataset(file_path)
```

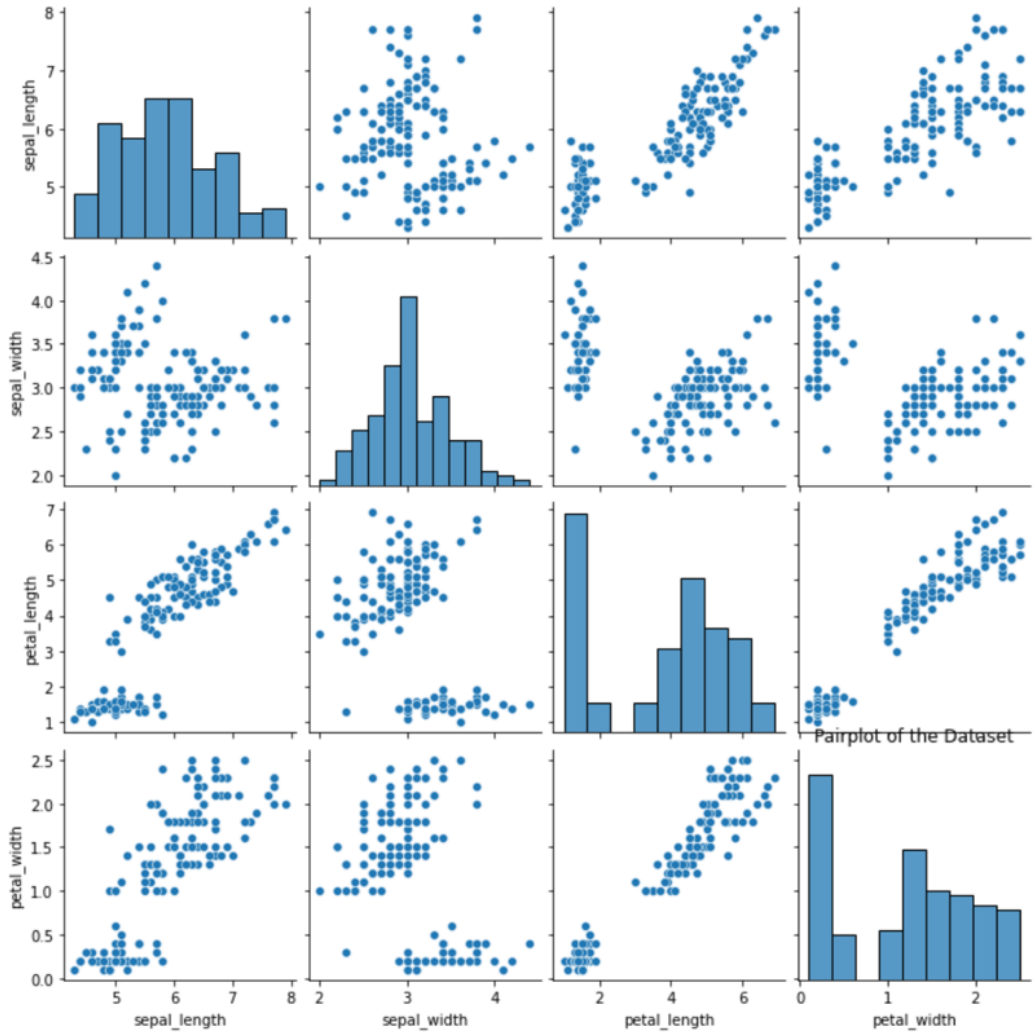
Output:

```
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None

First few rows of the dataset:
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa

Summary statistics:
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000

Unique values for categorical columns:
species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

5. **Write a program to Visualize the dataset to gain insights using Matplotlib or Seaborn by plotting scatter plots, and bar charts.**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def visualize_dataset(file_path):

    # Load the dataset into a pandas DataFrame
    df = pd.read_csv(file_path)

    # Assuming it's a CSV file, change accordingly if it's an Excel file

    # Plot scatter plots

    sns.pairplot(df)
    plt.title("Pairplot of the Dataset")
    plt.show()

    # Plot bar chart for categorical column (assuming the first column is categorical)

  if df.iloc[:, 0].dtype == 'object':
      sns.countplot(x=df.columns[0], data=df)
      plt.title("Bar Chart of Categorical Column")
      plt.xlabel(df.columns[0])
      plt.ylabel("Count")
      plt.show()
  else:
      print("No categorical column found to plot bar chart.")

# Example usage
file_path = 'IRIS.csv'  # Change this to the path of your CSV file
visualize_dataset(file_path)
```

Output:




```
No categorical column found to plot bar chart.
```

6. **Write a program to Handle missing data, encode categorical variables, and perform feature scaling.**

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder, StandardScaler


# Load Iris dataset

iris = load_iris()

iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

iris_df['target'] = iris.target


def preprocess_dataset(df):

# Handle missing data (Iris dataset doesn't have missing values, but we'll simulate some)

 df.iloc[::10, 0] = float('NaN')

# Simulate missing values in the first column

 imputer = SimpleImputer(strategy='mean')

 df[df.columns] = imputer.fit_transform(df[df.columns])

 # Encode categorical variable (if applicable)
```

```
 # Since Iris dataset doesn't have categorical variables, we'll skip this step

  # Perform feature scaling

scaler = StandardScaler()

df[df.columns[:-1]] = scaler.fit_transform(df[df.columns[:-1]])

 return df

# Preprocess Iris dataset

preprocessed_df = preprocess_dataset(iris_df)

# Display preprocessed dataset

print("Preprocessed dataset:")

print(preprocessed_df.head())
```

Output:

```
Preprocessed dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0       1.132764e-15          1.019004          -1.340227         -1.315444
1      -1.196022e+00         -0.131979          -1.340227         -1.315444
2      -1.451098e+00          0.328414          -1.397064         -1.315444
3      -1.578636e+00          0.098217          -1.283389         -1.315444
4      -1.068484e+00          1.249201          -1.340227         -1.315444

   target
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0
```

[ ]:

7. **Write a program to implement a k-Nearest Neighbours (k-NN) classifier using scikit-learn and Train the classifier on the dataset and evaluate its performance.**

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report


# Load Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# 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)


# Initialize the k-NN classifier

k = 3    # Number of neighbors
```

```
knn_classifier = KNeighborsClassifier(n_neighbors=k)


# Train the classifier

knn_classifier.fit(X_train, y_train)


# Make predictions on the testing set

y_pred = knn_classifier.predict(X_test)

# Evaluate the classifier's performance

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)


# Display classification report

print("Classification Report:")

print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

Output:

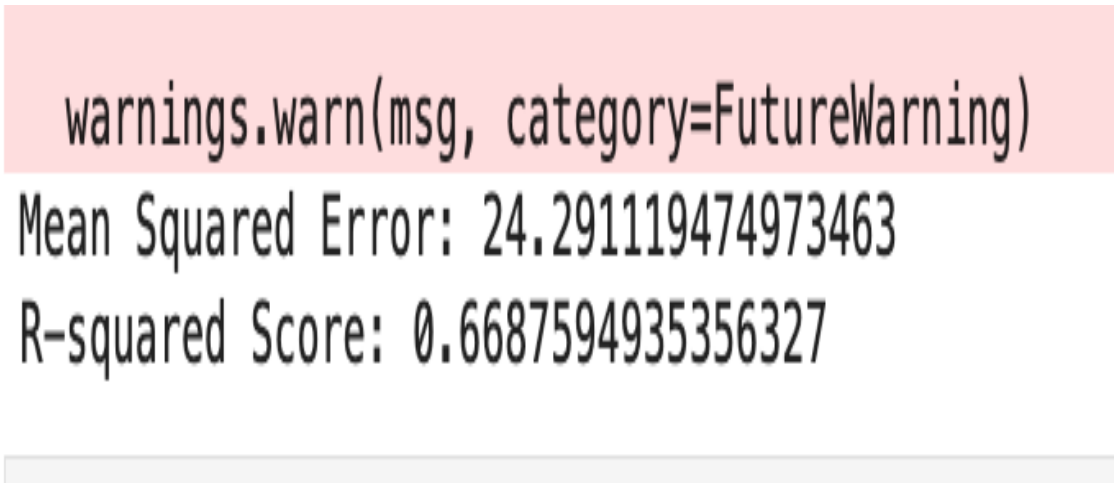
```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

8. **Write a program to implement a linear regression model for regression tasks and Train the model on a dataset with continuous target variables.**

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_boston

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

# Load Boston Housing dataset

boston = load_boston()

X = boston.data

y = boston.target

# Convert the data to a pandas DataFrame for easier manipulation

boston_df = pd.DataFrame(data=X, columns=boston.feature_names)

boston_df['target'] = 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)

# Initialize Linear Regression model

linear_regression = LinearRegression()

# Train the model
```

```
linear_regression.fit(X_train, y_train)

# Make predictions on the testing set

y_pred = linear_regression.predict(X_test)


# Evaluate the model's performance

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)

print("R-squared Score:", r2)
```

Output:

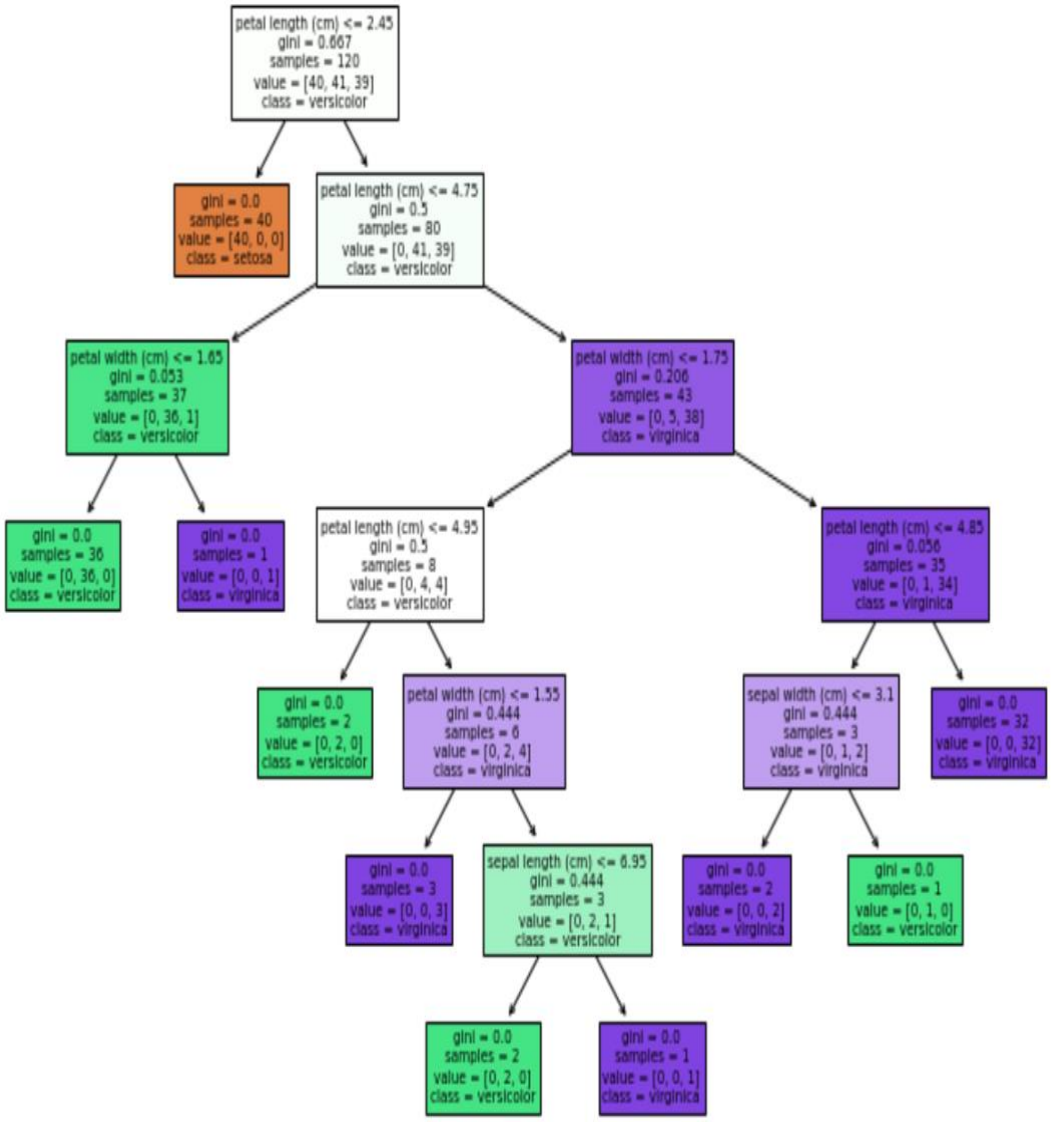


9. Write a program to implement a decision tree classifier using scikit-learn and visualize the decision tree and understand its splits.

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier, plot_tree

import matplotlib.pyplot as plt

# Load Iris dataset

iris = load_iris()

X = iris.data

y = iris.target

# 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)

# Initialize Decision Tree classifier

decision_tree = DecisionTreeClassifier()

# Train the classifier

decision_tree.fit(X_train, y_train)
```

```
# Visualize the decision tree

plt.figure(figsize=(12, 8))

plot_tree(decision_tree, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)

plt.show()
```

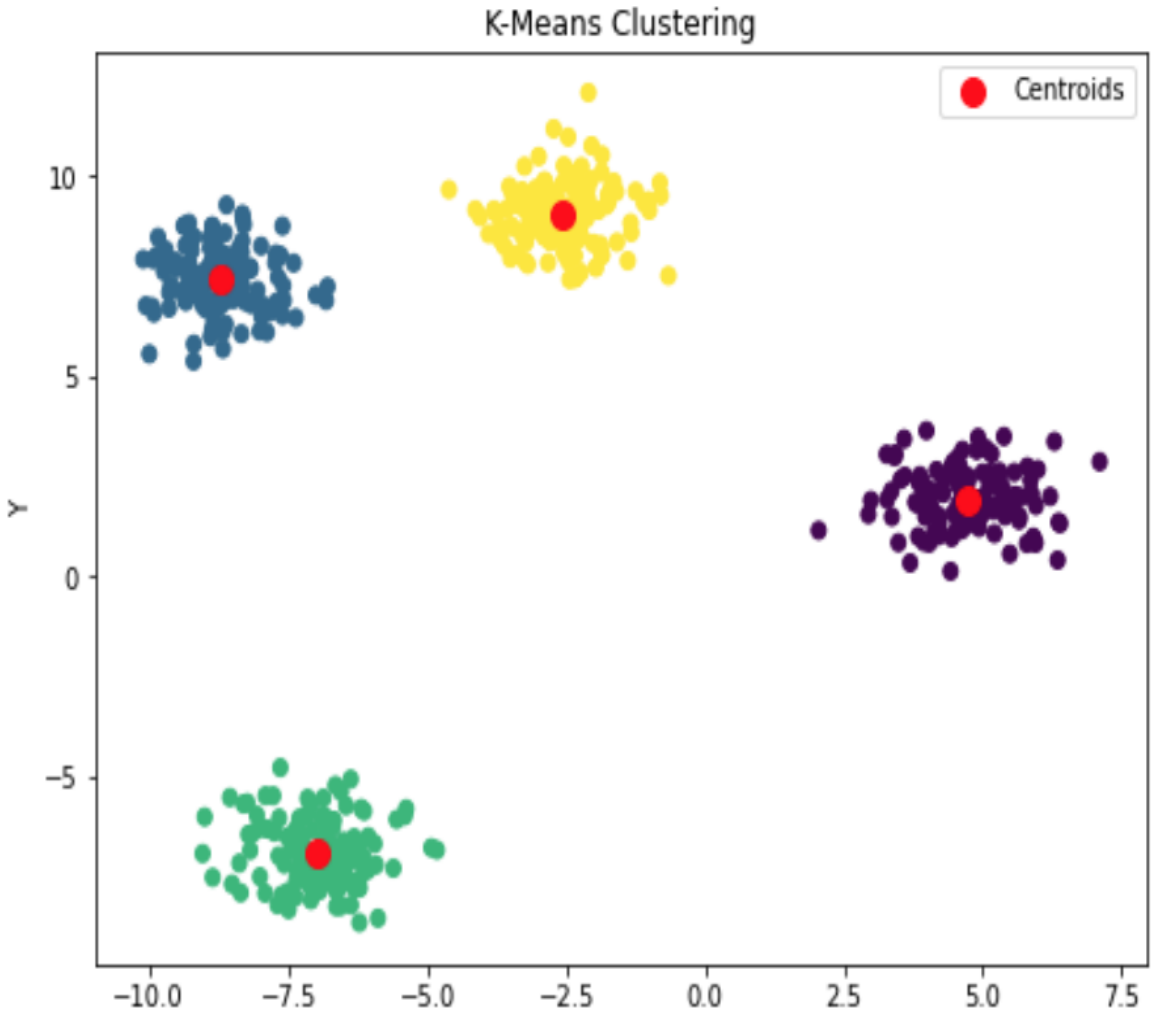
Output:


petal length (cm) <= 2.45
gini = 0.667
samples = 120
value = [40, 41, 39]
class = versicolor
gini = 0.0
samples = 40
value = [40, 0, 0]
class = setosa
petal length (cm) <= 4.75
gini = 0.5
samples = 80
value = [0, 41, 39]
class = versicolor
petal width (cm) <= 1.65
gini = 0.053
samples = 37
value = [0, 36, 1]
class = versicolor
petal width (cm) <= 1.75
gini = 0.206
samples = 43
value = [0, 5, 38]
class = virginica
gini = 0.0
samples = 36
value = [0, 36, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
petal length (cm) <= 4.95
gini = 0.5
samples = 8
value = [0, 4, 4]
class = versicolor
petal length (cm) <= 4.85
gini = 0.056
samples = 35
value = [0, 1, 34]
class = virginica
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
petal width (cm) <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
class = virginica
sepal width (cm) <= 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 32
value = [0, 0, 32]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
sepal length (cm) <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

10. Write a program to Implement K-Means clustering and Visualize clusters.

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

# Generate sample data
X, y = make_blobs(n_samples=500, centers=4, cluster_std=0.8, random_state=42)

# Create a K-Means clusterer with 4 clusters
kmeans = KMeans(n_clusters=4, random_state=42)

# Fit the data
kmeans.fit(X)

# Get cluster labels
labels = kmeans.labels_

# Plot the data with cluster labels

plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=100,
c='red', label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Output:


K-Means Clustering
Centroids
Y
10
5
0
−5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5