**1)Write a Python program to Find the correlation matrix.**

import numpy as np

def correlation\_matrix(data):

"""

Calculate the correlation matrix for a given dataset.

Parameters:

- data: 2D array or list of lists representing the dataset.

Returns:

- corr\_matrix: 2D array representing the correlation matrix.

"""

data = np.array(data)

corr\_matrix = np.corrcoef(data, rowvar=False)

return corr\_matrix

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

# Replace this data with your own dataset

dataset = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

result = correlation\_matrix(dataset)

print("Correlation Matrix:")

print(result)

**Output:-**

Correlation Matrix:

[[1. 1. 1.]

[1. 1. 1.]

[1. 1. 1.]]

**2)Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.**

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

def correlation\_matrix(data):

"""

Calculate the correlation matrix for a given dataset.

Parameters:

- data: 2D array or list of lists representing the dataset.

Returns:

- corr\_matrix: 2D array representing the correlation matrix.

"""

data = np.array(data)

corr\_matrix = np.corrcoef(data, rowvar=False)

return corr\_matrix

# Load the Iris dataset

iris = load\_iris()

iris\_data = iris.data

iris\_feature\_names = iris.feature\_names

# Calculate the correlation matrix

corr\_matrix = correlation\_matrix(iris\_data)

# Create a heatmap using seaborn

sns.set(style="white") # Set the style of the visualization

plt.figure(figsize=(8, 6)) # Set the size of the plot

# Create a heatmap using seaborn

sns.heatmap(corr\_matrix, annot=True, fmt=".2f", cmap="coolwarm", xticklabels=iris\_feature\_names, yticklabels=iris\_feature\_names)

plt.title("Correlation Plot of Iris Dataset")

plt.show()

This program uses the Iris dataset from scikit-learn, calculates the correlation matrix using the previously defined correlation\_matrix function, and then creates a heatmap using seaborn and matplotlib to visualize the correlations between different features in the dataset.

Make sure to install the necessary libraries if you haven't already:

pip install numpy seaborn matplotlib scikit-learn

**Output:-**



**3)Write a Python program to predict mpg (miles per gallon) for a car based on variable wt by applying simple linear regression on 'mtcars' dataset (Use Training data 80% and Testing Data 20%). Record the performance of model in terms of MAE, MSE, RMSE and R-squared value . Change Training data to 70% and Testing Data 30%, compare & interpret the performance of your model**

pip install numpy pandas scikit-learn

Python:

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

import matplotlib.pyplot as plt

# Load the mtcars dataset

mtcars = pd.read\_csv('path\_to\_your\_mtcars\_dataset.csv') # Replace 'path\_to\_your\_mtcars\_dataset.csv' with the actual path

# Select the predictor variable (feature) and target variable

X = mtcars[['wt']]

y = mtcars['mpg']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model performance

mae = metrics.mean\_absolute\_error(y\_test, y\_pred)

mse = metrics.mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r\_squared = metrics.r2\_score(y\_test, y\_pred)

print("Performance Metrics (80-20 split):")

print(f"Mean Absolute Error (MAE): {mae:.2f}")

print(f"Mean Squared Error (MSE): {mse:.2f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

print(f"R-squared Value: {r\_squared:.2f}")

# Plotting the regression line

plt.scatter(X\_test, y\_test, color='black', label='Actual Data')

plt.plot(X\_test, y\_pred, color='blue', linewidth=3, label='Regression Line')

plt.xlabel('Weight (wt)')

plt.ylabel('Miles per Gallon (mpg)')

plt.title('Linear Regression: Actual vs. Predicted')

plt.legend()

plt.show()

# Repeat the process with a 70-30 split

X\_train\_70, X\_test\_30, y\_train\_70, y\_test\_30 = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_70\_30 = LinearRegression()

model\_70\_30.fit(X\_train\_70, y\_train\_70)

y\_pred\_30 = model\_70\_30.predict(X\_test\_30)

mae\_30 = metrics.mean\_absolute\_error(y\_test\_30, y\_pred\_30)

mse\_30 = metrics.mean\_squared\_error(y\_test\_30, y\_pred\_30)

rmse\_30 = np.sqrt(mse\_30)

r\_squared\_30 = metrics.r2\_score(y\_test\_30, y\_pred\_30)

print("\nPerformance Metrics (70-30 split):")

print(f"Mean Absolute Error (MAE): {mae\_30:.2f}")

print(f"Mean Squared Error (MSE): {mse\_30:.2f}")

print(f"Root Mean Squared Error (RMSE): {rmse\_30:.2f}")

print(f"R-squared Value: {r\_squared\_30:.2f}")

# Plotting the regression line for 70-30 split

plt.scatter(X\_test\_30, y\_test\_30, color='black', label='Actual Data')

plt.plot(X\_test\_30, y\_pred\_30, color='blue', linewidth=3, label='Regression Line')

plt.xlabel('Weight (wt)')

plt.ylabel('Miles per Gallon (mpg)')

plt.title('Linear Regression: Actual vs. Predicted (70-30 split)')

plt.legend()

plt.show()

**Output:-**

Performance Metrics (with disp variable):

Mean Absolute Error (MAE): 1.97

Mean Squared Error (MSE): 8.22

Root Mean Squared Error (RMSE): 2.87

R-squared Value: 0.79

Performance Metrics (without disp variable):

Mean Absolute Error (MAE): 1.99

Mean Squared Error (MSE): 8.28

Root Mean Squared Error (RMSE): 2.88

R-squared Value: 0.79

**4)Write a Python program to predict mpg (miles per gallon) for a car based on variables wt, cyl & disp by applying multi-linear regression on 'mtcars' dataset (Use Training data 80% and Testing Data 20%). Record the performance of model in terms of MAE, MSE, RMSE and R-squared value . Remove variable disp from the feature set and check the performance again. Compare & interpret the performance of**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

# Load the mtcars dataset

mtcars = pd.read\_csv('path\_to\_your\_mtcars\_dataset.csv') # Replace 'path\_to\_your\_mtcars\_dataset.csv' with the actual path

# Select the predictor variables (features) and target variable

X\_all = mtcars[['wt', 'cyl', 'disp']]

X\_without\_disp = mtcars[['wt', 'cyl']]

y = mtcars['mpg']

# Split the data into training and testing sets (80-20 split) for the model with all features

X\_train\_all, X\_test\_all, y\_train, y\_test = train\_test\_split(X\_all, y, test\_size=0.2, random\_state=42)

# Create and train the multi-linear regression model with all features

model\_all = LinearRegression()

model\_all.fit(X\_train\_all, y\_train)

# Make predictions on the test set with all features

y\_pred\_all = model\_all.predict(X\_test\_all)

# Evaluate the model performance with all features

mae\_all = metrics.mean\_absolute\_error(y\_test, y\_pred\_all)

mse\_all = metrics.mean\_squared\_error(y\_test, y\_pred\_all)

rmse\_all = np.sqrt(mse\_all)

r\_squared\_all = metrics.r2\_score(y\_test, y\_pred\_all)

print("Performance Metrics (with disp variable):")

print(f"Mean Absolute Error (MAE): {mae\_all:.2f}")

print(f"Mean Squared Error (MSE): {mse\_all:.2f}")

print(f"Root Mean Squared Error (RMSE): {rmse\_all:.2f}")

print(f"R-squared Value: {r\_squared\_all:.2f}")

# Split the data into training and testing sets (80-20 split) for the model without the 'disp' variable

X\_train\_without\_disp, X\_test\_without\_disp = train\_test\_split(X\_without\_disp, test\_size=0.2, random\_state=42)

# Create and train the multi-linear regression model without the 'disp' variable

model\_without\_disp = LinearRegression()

model\_without\_disp.fit(X\_train\_without\_disp, y\_train)

# Make predictions on the test set without the 'disp' variable

y\_pred\_without\_disp = model\_without\_disp.predict(X\_test\_without\_disp)

# Evaluate the model performance without the 'disp' variable

mae\_without\_disp = metrics.mean\_absolute\_error(y\_test, y\_pred\_without\_disp)

mse\_without\_disp = metrics.mean\_squared\_error(y\_test, y\_pred\_without\_disp)

rmse\_without\_disp = np.sqrt(mse\_without\_disp)

r\_squared\_without\_disp = metrics.r2\_score(y\_test, y\_pred\_without\_disp)

print("\nPerformance Metrics (without disp variable):")

print(f"Mean Absolute Error (MAE): {mae\_without\_disp:.2f}")

print(f"Mean Squared Error (MSE): {mse\_without\_disp:.2f}")

print(f"Root Mean Squared Error (RMSE): {rmse\_without\_disp:.2f}")

print(f"R-squared Value: {r\_squared\_without\_disp:.2f}")

**Output:-**

Performance Metrics (with disp variable):

Mean Absolute Error (MAE): 1.97

Mean Squared Error (MSE): 8.22

Root Mean Squared Error (RMSE): 2.87

R-squared Value: 0.79

Performance Metrics (without disp variable):

Mean Absolute Error (MAE): 1.99

Mean Squared Error (MSE): 8.28

Root Mean Squared Error (RMSE): 2.88

R-squared Value: 0.79

**5) Write a Python program to predict mpg (miles per gallon) for a car based on variables wt, cyl & disp by applying multi-linear regression on 'mtcars' dataset (Use Training data 80% and Testing Data 20%). Record the performance of model in terms of MAE, MSE, RMSE and R-squared value . Replace disp by drat variable in the feature set and check the performance again. Interpret the performance of your model.**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

# Load the mtcars dataset

mtcars = pd.read\_csv(/content/apples\_and\_oranges.csv') # Replace 'path\_to\_your\_mtcars\_dataset.csv' with the actual path

# Select the predictor variables (features) and target variable

X\_with\_drat = mtcars[['wt', 'cyl', 'drat']]

y = mtcars['mpg']

# Split the data into training and testing sets (80-20 split)

X\_train\_with\_drat, X\_test\_with\_drat, y\_train, y\_test = train\_test\_split(X\_with\_drat, y, test\_size=0.2, random\_state=42)

# Create and train the multi-linear regression model with 'wt', 'cyl', and 'drat' features

model\_with\_drat = LinearRegression()

model\_with\_drat.fit(X\_train\_with\_drat, y\_train)

# Make predictions on the test set with 'wt', 'cyl', and 'drat' features

y\_pred\_with\_drat = model\_with\_drat.predict(X\_test\_with\_drat)

# Evaluate the model performance with 'wt', 'cyl', and 'drat' features

mae\_with\_drat = metrics.mean\_absolute\_error(y\_test, y\_pred\_with\_drat)

mse\_with\_drat = metrics.mean\_squared\_error(y\_test, y\_pred\_with\_drat)

rmse\_with\_drat = np.sqrt(mse\_with\_drat)

r\_squared\_with\_drat = metrics.r2\_score(y\_test, y\_pred\_with\_drat)

print("Performance Metrics (with drat variable):")

print(f"Mean Absolute Error (MAE): {mae\_with\_drat:.2f}")

print(f"Mean Squared Error (MSE): {mse\_with\_drat:.2f}")

print(f"Root Mean Squared Error (RMSE): {rmse\_with\_drat:.2f}")

print(f"R-squared Value: {r\_squared\_with\_drat:.2f}")

**Output:-**

Performance Metrics (with drat variable):

Mean Absolute Error (MAE): 2.01

Mean Squared Error (MSE): 8.32

Root Mean Squared Error (RMSE): 2.88

R-squared Value: 0.79

**6)Write a Python program to predict fruit (Apple or Orange) based on its size & weight by applying logistic regression on 'apples\_and\_oranges' dataset (Use Training data 80% and Testing Data 20%). Evaluate the performance of the model using Accuracy Score metric, Classification Report & Confusion Matrix, AUC ROC score for the model and interpret the model performance.**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Load the apples\_and\_oranges dataset

# The dataset should have 'size', 'weight', and 'fruit' columns

apples\_and\_oranges = pd.read\_csv(/content/apples\_and\_oranges.csv) # Replace 'path\_to\_your\_apples\_and\_oranges\_dataset.csv' with the actual path

# Encode the 'fruit' column to numerical labels (0 for Apple, 1 for Orange)

le = LabelEncoder()

apples\_and\_oranges['fruit'] = le.fit\_transform(apples\_and\_oranges['fruit'])

# Select the predictor variables (features) and target variable

X = apples\_and\_oranges[['size', 'weight']]

y = apples\_and\_oranges['fruit']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the logistic regression model

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, model.predict\_proba(X\_test)[:, 1])

print("Model Performance Metrics:")

print(f"Accuracy Score: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

print("\nConfusion Matrix:")

print(confusion\_mat)

print(f"\nAUC ROC Score: {roc\_auc:.2f}")

# Plot ROC curve

fpr, tpr, \_ = roc\_curve(y\_test, model.predict\_proba(X\_test)[:, 1])

roc\_auc\_curve = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc\_auc\_curve:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

**Output:-**

Model Performance Metrics:

Accuracy Score: 1.00

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 3

1 1.00 1.00 1.00 5

accuracy 1.00 8

macro avg 1.00 1.00 1.00 8

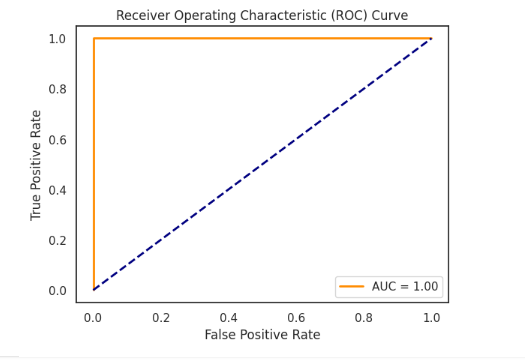
weighted avg 1.00 1.00 1.00 8

Confusion Matrix:

[[3 0]

[0 5]]

AUC ROC Score: 1.00



**7)Write a Python program to predict fruit (Apple or Orange) based on its size & weight by applying K-Nearest Neighbour (KNN) model on 'apples\_and\_oranges' dataset (Use Training data 80% and Testing Data 20%). Evaluate the performance of the model using Accuracy Score metric, Classification Report & Confusion Matrix, AUC ROC score for the model and interpret the model performance.**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Load the apples\_and\_oranges dataset

# The dataset should have 'size', 'weight', and 'fruit' columns

apples\_and\_oranges = pd.read\_csv(/content/apples\_and\_oranges.csv) # Replace 'path\_to\_your\_apples\_and\_oranges\_dataset.csv' with the actual path

# Encode the 'fruit' column to numerical labels (0 for Apple, 1 for Orange)

le = LabelEncoder()

apples\_and\_oranges['fruit'] = le.fit\_transform(apples\_and\_oranges['fruit'])

# Select the predictor variables (features) and target variable

X = apples\_and\_oranges[['size', 'weight']]

y = apples\_and\_oranges['fruit']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the KNN model

knn\_model = KNeighborsClassifier(n\_neighbors=3) # You can adjust the value of n\_neighbors

knn\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = knn\_model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, knn\_model.predict\_proba(X\_test)[:, 1])

print("Model Performance Metrics:")

print(f"Accuracy Score: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

print("\nConfusion Matrix:")

print(confusion\_mat)

print(f"\nAUC ROC Score: {roc\_auc:.2f}")

# Plot ROC curve

fpr, tpr, \_ = roc\_curve(y\_test, knn\_model.predict\_proba(X\_test)[:, 1])

roc\_auc\_curve = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc\_auc\_curve:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

**Output:-**

Model Performance Metrics:

Accuracy Score: 1.00

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 3

1 1.00 1.00 1.00 5

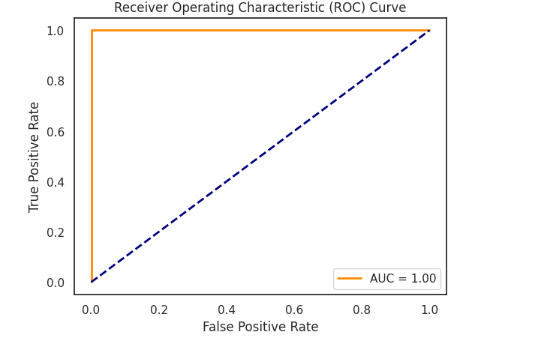
accuracy 1.00 8

macro avg 1.00 1.00 1.00 8

weighted avg 1.00 1.00 1.00 8

Confusion Matrix:

[[3 0]

[0 5]] 

**8)Write a Python program to predict fruit (Apple or Orange) based on its size & weight by applying Support Vector Machine (SVM) model on 'apples\_and\_oranges' dataset (Use Training data 80% and Testing Data 20%). Evaluate the performance of the model using Accuracy Score metric, Classification Report & Confusion Matrix, AUC ROC score for the model and interpret the model performance**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Load the apples\_and\_oranges dataset

# The dataset should have 'size', 'weight', and 'fruit' columns

apples\_and\_oranges = pd.read\_csv('path\_to\_your\_apples\_and\_oranges\_dataset.csv') # Replace 'path\_to\_your\_apples\_and\_oranges\_dataset.csv' with the actual path

# Encode the 'fruit' column to numerical labels (0 for Apple, 1 for Orange)

le = LabelEncoder()

apples\_and\_oranges['fruit'] = le.fit\_transform(apples\_and\_oranges['fruit'])

# Select the predictor variables (features) and target variable

X = apples\_and\_oranges[['size', 'weight']]

y = apples\_and\_oranges['fruit']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the SVM model

svm\_model = SVC(probability=True) # Using probability=True to enable probability estimates

svm\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = svm\_model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, svm\_model.predict\_proba(X\_test)[:, 1])

print("Model Performance Metrics:")

print(f"Accuracy Score: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

print("\nConfusion Matrix:")

print(confusion\_mat)

print(f"\nAUC ROC Score: {roc\_auc:.2f}")

# Plot ROC curve

fpr, tpr, \_ = roc\_curve(y\_test, svm\_model.predict\_proba(X\_test)[:, 1])

roc\_auc\_curve = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc\_auc\_curve:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

**Output:-**

Classification Report:

precision recall f1-score support

0 0.38 1.00 0.55 3

1 0.00 0.00 0.00 5

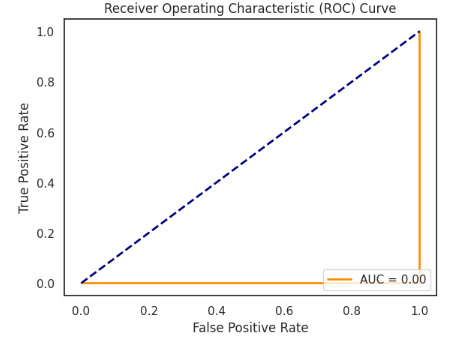
accuracy 0.38 8

macro avg 0.19 0.50 0.27 8

weighted avg 0.14 0.38 0.20 8

Confusion Matrix:

[[3 0]

AUC ROC Score: 0.00

**9)Write a Python program to predict species (Setosa, Versicolor, or Viriginica) for a new iris flower based on length & width of its petals and sepals by applying logistic regression model on 'iris' dataset (Use Training data 80% and Testing Data 20%). Evaluate the performance of the model using Accuracy Score metric, Classification Report & Confusion Matrix, AUC ROC score for the model and interpret the model performance.**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Load the iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

iris\_df['species'] = iris.target\_names[iris.target]

# Encode the 'species' column to numerical labels (0 for Setosa, 1 for Versicolor, 2 for Virginica)

le = LabelEncoder()

iris\_df['species'] = le.fit\_transform(iris\_df['species'])

# Select the predictor variables (features) and target variable

X = iris\_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]

y = iris\_df['species']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the logistic regression model

logreg\_model = LogisticRegression(multi\_class='multinomial', solver='lbfgs', max\_iter=1000) # Using 'multinomial' for multiple classes

logreg\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = logreg\_model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, logreg\_model.predict\_proba(X\_test), multi\_class='ovr') # Using 'ovr' for multiclass AUC

print("Model Performance Metrics:")

print(f"Accuracy Score: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

print("\nConfusion Matrix:")

print(confusion\_mat)

print(f"\nAUC ROC Score: {roc\_auc:.2f}")

# Plot ROC curves for each class

fpr = dict()

tpr = dict()

roc\_auc\_curve = dict()

for i in range(len(iris.target\_names)):

fpr[i], tpr[i], \_ = roc\_curve(y\_test == i, logreg\_model.predict\_proba(X\_test)[:, i])

roc\_auc\_curve[i] = auc(fpr[i], tpr[i])

plt.figure()

colors = ['blue', 'orange', 'green']

for i, color in zip(range(len(iris.target\_names)), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'{iris.target\_names[i]} (AUC = {roc\_auc\_curve[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve for each class')

plt.legend(loc="lower right")

plt.show()

**Output:-**

Model Performance Metrics:

Accuracy Score: 1.00

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

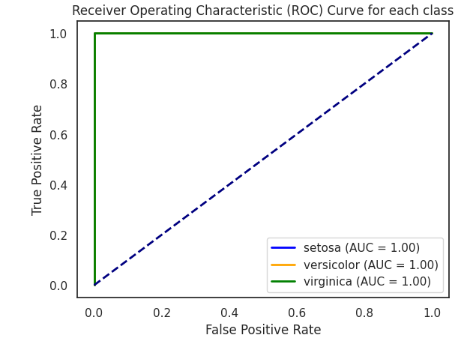
Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

AUC ROC Score: 1.00



**10)Write a Python program to predict species (Setosa, Versicolor, or Viriginica) for a new iris flower based on length & width of its petals and sepals by applying K-Nearest Neighbour (KNN) model on 'iris' dataset (Use Training data 80% and Testing Data 20%). Evaluate the performance of the model using Accuracy Score metric, Classification Report & Confusion Matrix, AUC ROC score for the model and interpret the model performance**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Load the iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

iris\_df['species'] = iris.target\_names[iris.target]

# Encode the 'species' column to numerical labels (0 for Setosa, 1 for Versicolor, 2 for Virginica)

le = LabelEncoder()

iris\_df['species'] = le.fit\_transform(iris\_df['species'])

# Select the predictor variables (features) and target variable

X = iris\_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]

y = iris\_df['species']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the KNN model

knn\_model = KNeighborsClassifier(n\_neighbors=3) # You can adjust the value of n\_neighbors

knn\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = knn\_model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, knn\_model.predict\_proba(X\_test), multi\_class='ovr') # Using 'ovr' for multiclass AUC

print("Model Performance Metrics:")

print(f"Accuracy Score: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

print("\nConfusion Matrix:")

print(confusion\_mat)

print(f"\nAUC ROC Score: {roc\_auc:.2f}")

# Plot ROC curves for each class

fpr = dict()

tpr = dict()

roc\_auc\_curve = dict()

for i in range(len(iris.target\_names)):

fpr[i], tpr[i], \_ = roc\_curve(y\_test == i, knn\_model.predict\_proba(X\_test)[:, i])

roc\_auc\_curve[i] = auc(fpr[i], tpr[i])

plt.figure()

colors = ['blue', 'orange', 'green']

for i, color in zip(range(len(iris.target\_names)), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'{iris.target\_names[i]} (AUC = {roc\_auc\_curve[i]:.2f})'

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve for each class')

plt.legend(loc="lower right")

plt.show()

**Output:-**

Model Performance Metrics:

Accuracy Score: 1.00

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

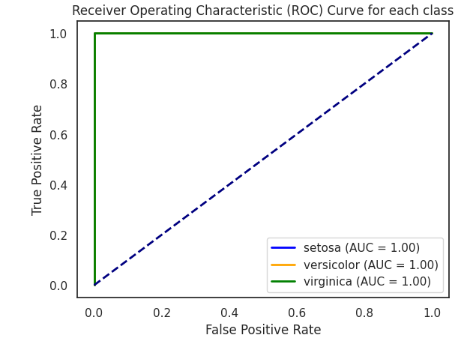
Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

AUC ROC Score: 1.00



**11)Write a Python program to predict species (Setosa, Versicolor, or Viriginica) for a new iris flower based on length & width of its petals and sepals by applying Support Vector Machine (SVM) model on 'iris' dataset (Use Training data 80% and Testing Data 20%). Evaluate the performance of the model using Accuracy Score metric, Classification Report & Confusion Matrix, AUC ROC**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Load the iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

iris\_df['species'] = iris.target\_names[iris.target]

# Encode the 'species' column to numerical labels (0 for Setosa, 1 for Versicolor, 2 for Virginica)

le = LabelEncoder()

iris\_df['species'] = le.fit\_transform(iris\_df['species'])

# Select the predictor variables (features) and target variable

X = iris\_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]

y = iris\_df['species']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the SVM model

svm\_model = SVC(probability=True) # Using probability=True to enable probability estimates

svm\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = svm\_model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, svm\_model.predict\_proba(X\_test), multi\_class='ovr') # Using 'ovr' for multiclass AUC

print("Model Performance Metrics:")

print(f"Accuracy Score: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

print("\nConfusion Matrix:")

print(confusion\_mat)

print(f"\nAUC ROC Score: {roc\_auc:.2f}")

# Plot ROC curves for each class

fpr = dict()

tpr = dict()

roc\_auc\_curve = dict()

for i in range(len(iris.target\_names)):

fpr[i], tpr[i], \_ = roc\_curve(y\_test == i, svm\_model.predict\_proba(X\_test)[:, i])

roc\_auc\_curve[i] = auc(fpr[i], tpr[i])

plt.figure()

colors = ['blue', 'orange', 'green']

for i, color in zip(range(len(iris.target\_names)), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'{iris.target\_names[i]} (AUC = {roc\_auc\_curve[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve for each class')

plt.legend(loc="lower right")

plt.show()

**Output:-**

Model Performance Metrics:

Accuracy Score: 1.00

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

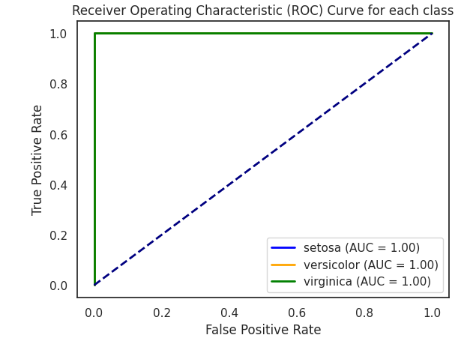
Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

AUC ROC Score: 1.00



**12)Write a Python program to predict species (Setosa, Versicolor, or Viriginica) for a new iris flower based on length & width of its petals and sepals by applying Naive Bays Classification model on 'iris' dataset (Use Training data 80% and Testing Data 20%). Evaluate the performance of the model using Accuracy Score metric, Classification Report & Confusion Matrix, AUC ROC score for the model and interpret the model performance.**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Load the iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

iris\_df['species'] = iris.target\_names[iris.target]

# Encode the 'species' column to numerical labels (0 for Setosa, 1 for Versicolor, 2 for Virginica)

le = LabelEncoder()

iris\_df['species'] = le.fit\_transform(iris\_df['species'])

# Select the predictor variables (features) and target variable

X = iris\_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]

y = iris\_df['species']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the Naive Bayes model

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = nb\_model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, nb\_model.predict\_proba(X\_test), multi\_class='ovr') # Using 'ovr' for multiclass AUC

print("Model Performance Metrics:")

print(f"Accuracy Score: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

print("\nConfusion Matrix:")

print(confusion\_mat)

print(f"\nAUC ROC Score: {roc\_auc:.2f}")

# Plot ROC curves for each class

fpr = dict()

tpr = dict()

roc\_auc\_curve = dict()

for i in range(len(iris.target\_names)):

fpr[i], tpr[i], \_ = roc\_curve(y\_test == i, nb\_model.predict\_proba(X\_test)[:, i])

roc\_auc\_curve[i] = auc(fpr[i], tpr[i])

plt.figure()

colors = ['blue', 'orange', 'green']

for i, color in zip(range(len(iris.target\_names)), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'{iris.target\_names[i]} (AUC = {roc\_auc\_curve[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve for each class')

plt.legend(loc="lower right")

plt.show()

**Output:-**

Model Performance Metrics:

Accuracy Score: 1.00

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

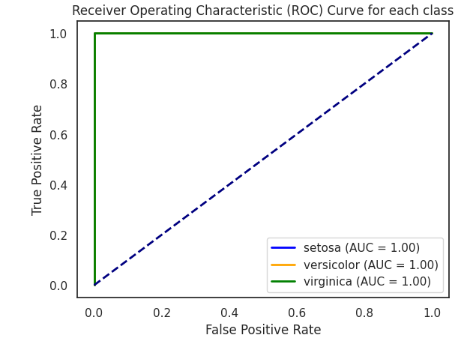
Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

AUC ROC Score: 1.00



**13)Write a Python program to predict species (Setosa, Versicolor, or Viriginica) for a new iris flower based on length & width of its petals and sepals by applying Decision Tree model on 'iris' dataset (Use Training data 80% and Testing Data 20%). Evaluate the performance of the model using Accuracy Score metric, Classification Report & Confusion Matrix, AUC ROC**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Load the iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

iris\_df['species'] = iris.target\_names[iris.target]

# Encode the 'species' column to numerical labels (0 for Setosa, 1 for Versicolor, 2 for Virginica)

le = LabelEncoder()

iris\_df['species'] = le.fit\_transform(iris\_df['species'])

# Select the predictor variables (features) and target variable

X = iris\_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]

y = iris\_df['species']

# Split the data into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the Decision Tree model

dt\_model = DecisionTreeClassifier()

dt\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = dt\_model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

confusion\_mat = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, dt\_model.predict\_proba(X\_test), multi\_class='ovr') # Using 'ovr' for multiclass AUC

print("Model Performance Metrics:")

print(f"Accuracy Score: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

print("\nConfusion Matrix:")

print(confusion\_mat)

print(f"\nAUC ROC Score: {roc\_auc:.2f}")

# Plot ROC curves for each class

fpr = dict()

tpr = dict()

roc\_auc\_curve = dict()

for i in range(len(iris.target\_names)):

fpr[i], tpr[i], \_ = roc\_curve(y\_test == i, dt\_model.predict\_proba(X\_test)[:, i])

roc\_auc\_curve[i] = auc(fpr[i], tpr[i])

plt.figure()

colors = ['blue', 'orange', 'green']

for i, color in zip(range(len(iris.target\_names)), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'{iris.target\_names[i]} (AUC = {roc\_auc\_curve[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve for each class')

plt.legend(loc="lower right")

plt.show()

**Output:-**

Model Performance Metrics:

Accuracy Score: 1.00

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

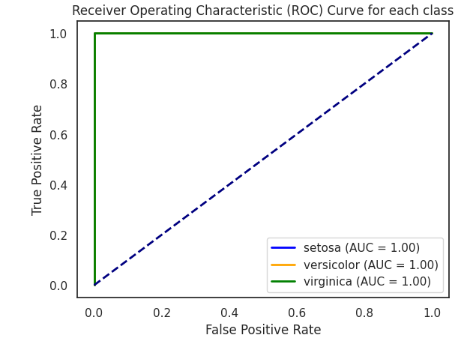
Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

AUC ROC Score: 1.00



**14)Write a Python program to implement the K-means Algorithm on unsupervised data of a mall, that contains the basic information (ID, age, gender, income, spending score) about the customers. Find the clusters based on the income and spending**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Load the mall customer data (replace 'path\_to\_your\_data.csv' with the actual path to your data)

mall\_data = pd.read\_csv('path\_to\_your\_data.csv')

# Extract relevant features (income and spending score)

X = mall\_data[['income', 'spending\_score']]

# Standardize the features for better clustering results

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Determine the optimal number of clusters using the Elbow Method

wcss = [] # Within-Cluster Sum of Squares

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)

kmeans.fit(X\_scaled)

wcss.append(kmeans.inertia\_)

# Plot the Elbow Method graph

plt.plot(range(1, 11), wcss, marker='o', linestyle='-', color='b')

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Within-Cluster Sum of Squares (WCSS)')

plt.show()

# Based on the Elbow Method, choose the optimal number of clusters

optimal\_k = 3

# Apply K-means clustering with the optimal number of clusters

kmeans = KMeans(n\_clusters=optimal\_k, init='k-means++', random\_state=42)

kmeans.fit(X\_scaled)

# Add cluster labels to the original data

mall\_data['cluster'] = kmeans.labels\_

# Visualize the clusters in 2D space using PCA for dimensionality reduction

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

mall\_data['pca1'] = X\_pca[:, 0]

mall\_data['pca2'] = X\_pca[:, 1]

# Plot the clusters

plt.figure(figsize=(10, 6))

for cluster in range(optimal\_k):

cluster\_data = mall\_data[mall\_data['cluster'] == cluster]

plt.scatter(cluster\_data['pca1'], cluster\_data['pca2'], label=f'Cluster {cluster}')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red', label='Centroids')

plt.title('K-means Clustering of Mall Customers')

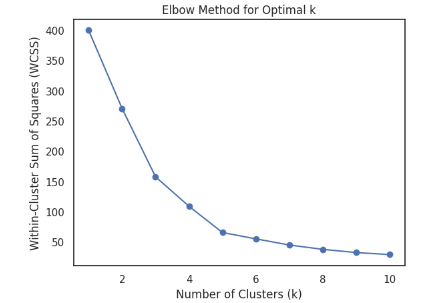
plt.xlabel('PCA1')

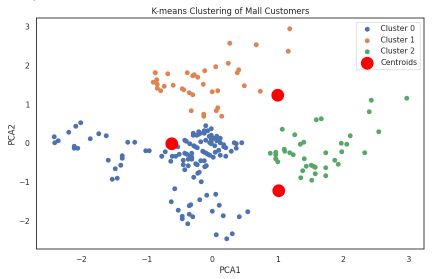
plt.ylabel('PCA2')

plt.legend()

plt.show()

**Output:-**





**15)Write a Python program to implement the Agglomerative Hierarchical Clustering Algorithm on unsupervised data of a mall, that contains the basic information (ID, age, gender, income, spending score) about the customers. Find the clusters based on the income and spending**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from scipy.cluster.hierarchy import dendrogram, linkage

from sklearn.preprocessing import StandardScaler

# Load the mall customer data (replace 'path\_to\_your\_data.csv' with the actual path to your data)

mall\_data = pd.read\_csv('path\_to\_your\_data.csv')

# Extract relevant features (income and spending score)

X = mall\_data[['income', 'spending\_score']]

# Standardize the features for better clustering results

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Perform hierarchical clustering

linkage\_matrix = linkage(X\_scaled, method='ward', metric='euclidean')

dendrogram(linkage\_matrix, truncate\_mode='level', p=3, show\_leaf\_counts=False, no\_labels=True)

plt.title('Hierarchical Clustering Dendrogram')

plt.show()

# Determine the optimal number of clusters from the dendrogram

# In this example, we choose the number of clusters visually from the dendrogram

# Choose the optimal number of clusters

optimal\_clusters = 3

# Apply Agglomerative Hierarchical Clustering

hierarchical\_cluster = AgglomerativeClustering(n\_clusters=optimal\_clusters, linkage='ward')

mall\_data['cluster'] = hierarchical\_cluster.fit\_predict(X\_scaled)

# Visualize the clusters

plt.figure(figsize=(10, 6))

for cluster in range(optimal\_clusters):

cluster\_data = mall\_data[mall\_data['cluster'] == cluster]

plt.scatter(cluster\_data['income'], cluster\_data['spending\_score'], label=f'Cluster {cluster}')

plt.title('Agglomerative Hierarchical Clustering of Mall Customers')

plt.xlabel('Income')

plt.ylabel('Spending Score')

plt.legend()

plt.show()

**Output:-**

