**PRACTICAL : 1**

**AIM : Given the following vectors:**

**A = [1, 2, 3, 4, 5, 6, 7, 8, 9 10]**

**B = [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]**

**C = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]**

**Ex. 1: Find the arithmetic mean of vector A, B and C**

**Ex. 2: Find the variance of the vector A, B and C**

**Ex. 3: Find the euclidean distance between vector A and B**

**Ex. 4: Find the correlation between vectors A & B and A & C**

**CODE :**

Ex. 1: Find the arithmetic mean of vector A, B and C

import numpy as np

# Assuming A, B, and C are NumPy arrays

A = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

B = np.array([4, 8, 12, 16, 20, 24, 28, 32, 36, 40])

C = np.array( [10, 9, 8, 7, 6, 5, 4, 3, 2, 1])

mean\_A = np.mean(A)

mean\_B = np.mean(B)

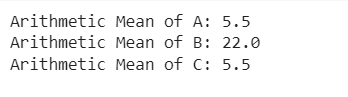
mean\_C = np.mean(C)

print("Arithmetic Mean of A:", mean\_A)

print("Arithmetic Mean of B:", mean\_B)

print("Arithmetic Mean of C:", mean\_C)

**OUTPUT :**



Ex. 2: Find the variance of the vector A, B and C

import numpy as np

# Assuming A, B, and C are NumPy arrays

A = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

B = np.array([4, 8, 12, 16, 20, 24, 28, 32, 36, 40])

C = np.array( [10, 9, 8, 7, 6, 5, 4, 3, 2, 1])

# Calculate variance

variance\_A = np.var(A)

variance\_B = np.var(B)

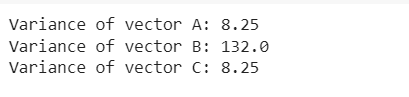
variance\_C = np.var(C)

print("Variance of vector A:", variance\_A)

print("Variance of vector B:", variance\_B)

print("Variance of vector C:", variance\_C)

**OUTPUT :**



Ex. 3: Find the euclidean distance between vector A and B

import numpy as np

# Assuming A, B, and C are NumPy arrays

A = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

B = np.array([4, 8, 12, 16, 20, 24, 28, 32, 36, 40])

# Calculate Euclidean distance

euclidean\_distance = np.linalg.norm(A - B)

print("Euclidean distance between A and B:", euclidean\_distance)

**OUTPUT :**



Ex. 4: Find the correlation between vectors A & B and A & C

import numpy as np

# Assuming A, B, and C are NumPy arrays

A = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

B = np.array([4, 8, 12, 16, 20, 24, 28, 32, 36, 40])

C = np.array( [10, 9, 8, 7, 6, 5, 4, 3, 2, 1])

# Calculate correlations

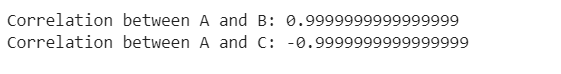
correlation\_AB = np.corrcoef(A, B)[0, 1]

correlation\_AC = np.corrcoef(A, C)[0, 1]

print("Correlation between A and B:", correlation\_AB)

print("Correlation between A and C:", correlation\_AC)

**OUTPUT :**



**PRACTICAL : 2**

**AIM : Load breast cancer dataset and perform classification using Euclidean distance. Use 70% data as training and 30% for testing.**

**CODE :**

import numpy as np

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

# Load breast cancer dataset

data = load\_breast\_cancer()

X = data.data

y = data.target

# Split the data into training and testing sets (70% training, 30% testing)

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

# Standardize the features (important for distance-based methods)

scaler = StandardScaler()

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

X\_test = scaler.transform(X\_test)

def euclidean\_distance(x1, x2):

return np.sqrt(np.sum((x1 - x2)\*\*2))

def predict(X\_train, y\_train, x\_test, k=3):

distances = [euclidean\_distance(x\_test, x) for x in X\_train]

k\_neighbors\_indices = np.argsort(distances)[:k]

k\_neighbor\_labels = [y\_train[i] for i in k\_neighbors\_indices]

most\_common = np.bincount(k\_neighbor\_labels).argmax()

return most\_common

# Make predictions on the testing data

y\_pred = [predict(X\_train, y\_train, x, k=3) for x in X\_test]

# Calculate accuracy

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

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

**OUTPUT :**



**PRACTICAL : 3**

**AIM : Repeat the above experiment with 10-fold cross validation and find the standard deviation in accuracy.**

**CODE :**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, KFold

from sklearn.metrics import accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

# Load the breast cancer dataset

data = load\_breast\_cancer()

X, y = data.data, data.target

# Split the data into training (70%) and testing (30%) sets

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

# Function to perform classification using Euclidean distance

def euclidean\_distance\_classification(X\_train, y\_train, X\_test):

knn\_classifier = KNeighborsClassifier(metric='euclidean')

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

predictions = knn\_classifier.predict(X\_test)

return predictions

# Perform classification on the test set

predictions = euclidean\_distance\_classification(X\_train, y\_train, X\_test)

# Calculate and print accuracy on the test set

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

print(f"Accuracy on the test set: {accuracy:.2f}")

# Now, let's repeat the experiment with 10-fold cross-validation and find the standard deviation in accuracy

kf = KFold(n\_splits=10, shuffle=True, random\_state=42)

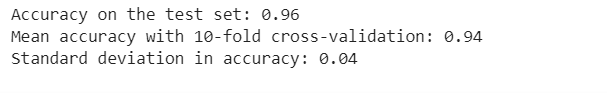
accuracies = cross\_val\_score(KNeighborsClassifier(metric='euclidean'), X, y, cv=kf)

# Calculate and print the mean and standard deviation of accuracies

print(f"Mean accuracy with 10-fold cross-validation: {np.mean(accuracies):.2f}")

print(f"Standard deviation in accuracy: {np.std(accuracies):.2f}")

**OUTPUT :**



**PRACTICAL : 4**

**AIM : Repeat the experiment 2 and build the confusion matrix. Also derive Precision, Recall and Specificity of the algorithm.**

**CODE :**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

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

def specificity\_score(y\_true, y\_pred):

tn, fp, \_, \_ = confusion\_matrix(y\_true, y\_pred).ravel()

return tn / (tn + fp)

from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

# Load the breast cancer dataset

data = load\_breast\_cancer()

X, y = data.data, data.target

# Split the data into training (70%) and testing (30%) sets

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

# Function to perform classification using Euclidean distance

def euclidean\_distance\_classification(X\_train, y\_train, X\_test):

knn\_classifier = KNeighborsClassifier(metric='euclidean')

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

predictions = knn\_classifier.predict(X\_test)

return predictions

# Perform classification on the test set

predictions = euclidean\_distance\_classification(X\_train, y\_train, X\_test)

# Build the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, predictions)

# Calculate precision, recall, and specificity

precision = precision\_score(y\_test, predictions)

recall = recall\_score(y\_test, predictions)

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

specificity = specificity\_score(y\_test, predictions)

# Print the confusion matrix, precision, recall, and specificity

print("Confusion Matrix:")

print(conf\_matrix)

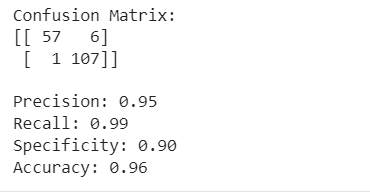
print(f"\nPrecision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"Specificity: {specificity:.2f}")

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

**OUTPUT :**



**PRACTICAL : 5**

**AIM : Predict the class for X = < Sunny, Cool, High, Strong > using Naïve Bayes Classifier for given data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Outlook** | **Temp.** | **Humidity** | **Windy** | **Play** |
| D1 | Sunny | Hot | High | False | No |
| D2 | Sunny | Hot | High | True | No |
| D3 | Overcast | Hot | High | False | Yes |
| D4 | Rainy | Mild | High | False | Yes |
| D5 | Rainy | Cool | Normal | False | Yes |
| D6 | Rainy | Cool | Normal | True | No |
| D7 | Overcast | Cool | Normal | True | Yes |
| D8 | Sunny | Mild | High | False | No |
| D9 | Sunny | Cool | Normal | False | Yes |
| D10 | Rainy | Mild | Normal | False | Yes |
| D11 | Sunny | Mild | Normal | True | Yes |
| D12 | Overcast | Mild | High | True | Yes |
| D13 | Overcast | Hot | Normal | False | Yes |
| D14 | Rainy | Mild | High | True | No |

**CODE :**

**OUTPUT :**

**PRACTICAL : 6**

**AIM : For the data given in Exercise 5, find the splitting attribute at first level:**

**Information Gain: 𝐼(𝑃, 𝑁) = − 𝑃 log2 𝑃 − 𝑁 log2 𝑁 = 0.940**

**𝑆 𝑆 𝑆 𝑆**

**𝑣**

**𝐸𝑛𝑡𝑟𝑜𝑝𝑦: 𝐸(𝑂𝑢𝑡𝑙𝑜𝑜𝑘) = ∑ 𝑃𝑖 + 𝑁𝑖 𝐼(𝑃 , 𝑁 )) = 0.694**

**𝑃 + 𝑁 𝑖 𝑖**

**𝑖=1**

**𝐺𝑎𝑖𝑛 (𝑂𝑢𝑡𝑙𝑜𝑜𝑘) = 𝐼(𝑃, 𝑁) − 𝐸(𝑂𝑢𝑡𝑙𝑜𝑜𝑘) = 0.246**

**CODE :**

import pandas as pd

import numpy as np

from sklearn.feature\_selection import mutual\_info\_classif

# Define the dataset

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],

'Temp.': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],

'Windy': [False, True, False, False, False, True, True, False, False, False, True, True, False, True],

'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}

df = pd.DataFrame(data)

# Encode categorical features using Label Encoding

for column in df.columns:

df[column] = pd.factorize(df[column])[0]

# Extract features and target variable

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

y = df['Play']

# Calculate Information Gain for each feature

information\_gains = mutual\_info\_classif(X, y)

# Find the splitting attribute with the highest Information Gain

splitting\_attribute = X.columns[np.argmax(information\_gains)]

# Output the result

print(f"The splitting attribute at the first level is: {splitting\_attribute}")

**OUTPUT :**



**PRACTICAL : 7**

**AIM : Generate and test decision tree for the dataset.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Outlook** | **Temp.** | **Humidity** | **Windy** | **Play** |
| **D1** | **Sunny** | **Hot** | **High** | **False** | **No** |
| **D2** | **Sunny** | **Hot** | **High** | **True** | **No** |
| **D3** | **Overcast** | **Hot** | **High** | **False** | **Yes** |
| **D4** | **Rainy** | **Mild** | **High** | **False** | **Yes** |
| **D5** | **Rainy** | **Cool** | **Normal** | **False** | **Yes** |
| **D6** | **Rainy** | **Cool** | **Normal** | **True** | **No** |
| **D7** | **Overcast** | **Cool** | **Normal** | **True** | **Yes** |
| **D8** | **Sunny** | **Mild** | **High** | **False** | **No** |
| **D9** | **Sunny** | **Cool** | **Normal** | **False** | **Yes** |
| **D10** | **Rainy** | **Mild** | **Normal** | **False** | **Yes** |
| **D11** | **Sunny** | **Mild** | **Normal** | **True** | **Yes** |
| **D12** | **Overcast** | **Mild** | **High** | **True** | **Yes** |
| **D13** | **Overcast** | **Hot** | **Normal** | **False** | **Yes** |
| **D14** | **Rainy** | **Mild** | **High** | **True** | **No** |

**CODE :**

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

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

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

from sklearn import preprocessing

# Given dataset

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],

'Temp': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],

'Windy': [False, True, False, False, False, True, True, False, False, False, True, True, False, True],

'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}

df = pd.DataFrame(data)

# Convert categorical variables to numerical for scikit-learn

le = preprocessing.LabelEncoder()

df['Outlook'] = le.fit\_transform(df['Outlook'])

df['Temp'] = le.fit\_transform(df['Temp'])

df['Humidity'] = le.fit\_transform(df['Humidity'])

df['Play'] = le.fit\_transform(df['Play'])

# Features and target variable

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

y = df['Play']

# 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)

# Create a decision tree classifier

dt\_classifier = DecisionTreeClassifier()

# Fit the classifier to the training data

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

# Make predictions on the test set

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

# Evaluate the performance

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

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

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

print(f"Accuracy: {accuracy:.4f}")

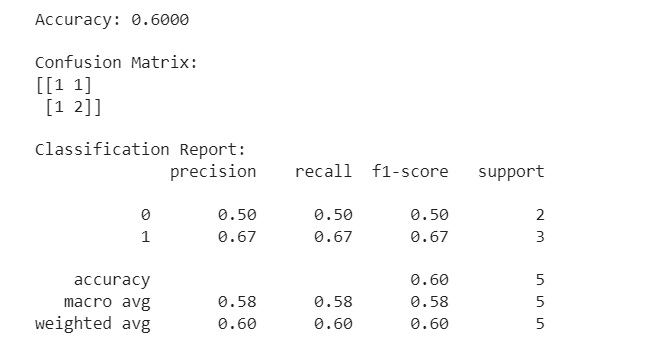
print("\nConfusion Matrix:")

print(conf\_matrix)

print("\nClassification Report:")

print(classification\_rep)

**OUTPUT:**



**PRACTICAL : 8**

**AIM : Find the clusters for following data with k = 2: Start with points 1 and 4 as two separate clusters. i**

|  |  |  |  |
| --- | --- | --- | --- |
| **i** | **A** | **B** |  |
| **1** | **1.0** | **1.0** |  |
| **2** | **1.5** | **2.0** |  |
| **3** | **3.0** | **4.0** |  |
| **4** | **5.0** | **7.0** |  |
| **5** | **3.5** | **5.0** |  |
| **6** | **4.5** | **5.0** |  |
| **7** | **3.5** | **4.5** |  |

**CODE :**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

# Given data

data = {

'A': [1.0, 1.5, 3.0, 5.0, 3.5, 4.5, 3.5],

'B': [1.0, 2.0, 4.0, 7.0, 5.0, 5.0, 4.5]

}

df = pd.DataFrame(data)

# Initial clusters with points 1 and 4

initial\_clusters = pd.DataFrame({

'A': [1.0, 5.0],

'B': [1.0, 7.0]

})

# Plot initial clusters

plt.scatter(df['A'], df['B'], label='Data Points')

plt.scatter(initial\_clusters['A'], initial\_clusters['B'], marker='X', s=200, color='red', label='Initial Clusters')

plt.title('Initial Clusters')

plt.xlabel('A')

plt.ylabel('B')

plt.legend()

plt.show()

# KMeans clustering with k=2

kmeans = KMeans(n\_clusters=2, init=initial\_clusters.values, n\_init=1, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(df)

# Plot final clusters

plt.scatter(df['A'], df['B'], c=df['Cluster'], cmap='viridis', label='Data Points')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], marker='X', s=200, color='red', label='Final Clusters')

plt.title('Final Clusters')

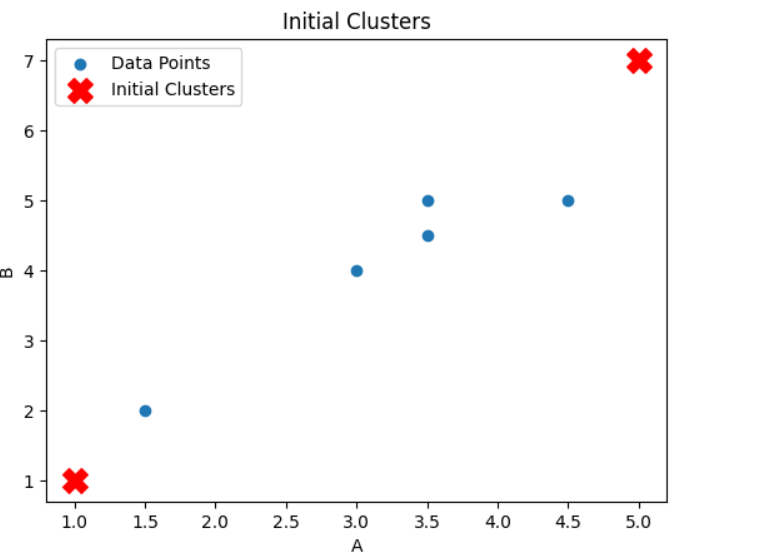
plt.xlabel('A')

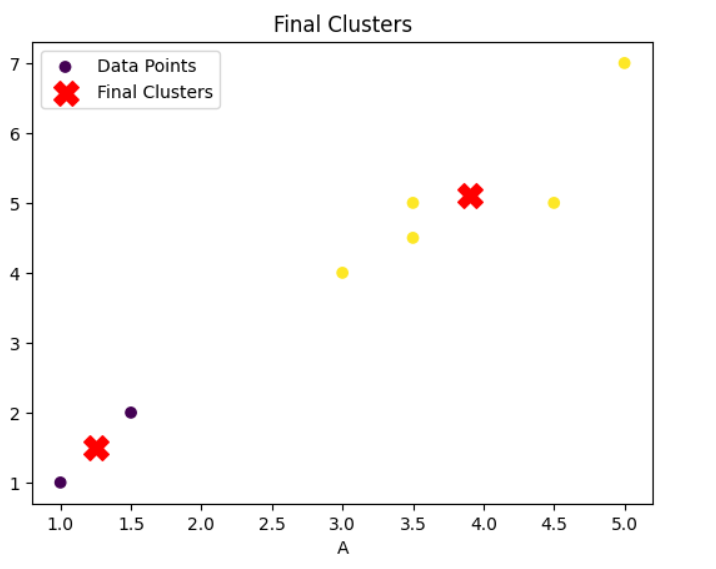
plt.ylabel('B')

plt.legend()

plt.show()

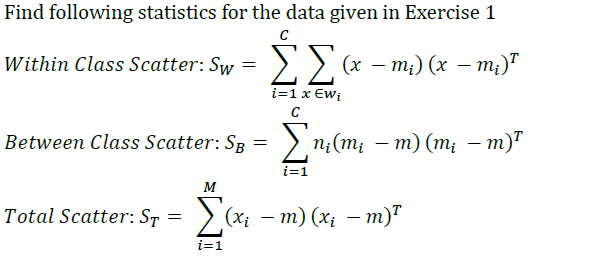
**OUTPUT:**





**PRACTICAL : 9**

**AIM :**



**CODE :**

import numpy as np

# Given data

A = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

B = np.array([4, 8, 12, 16, 20, 24, 28, 32, 36, 40])

C = np.array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])

# Calculate mean vectors

mean\_A = np.mean(A)

mean\_B = np.mean(B)

mean\_C = np.mean(C)

# Within-Class Scatter Matrix (S\_W)

S\_W\_A = np.sum((A - mean\_A)[:, np.newaxis] @ (A - mean\_A)[:, np.newaxis].T, axis=0)

S\_W\_B = np.sum((B - mean\_B)[:, np.newaxis] @ (B - mean\_B)[:, np.newaxis].T, axis=0)

S\_W\_C = np.sum((C - mean\_C)[:, np.newaxis] @ (C - mean\_C)[:, np.newaxis].T, axis=0)

S\_W = S\_W\_A + S\_W\_B + S\_W\_C

# Between-Class Scatter Matrix (S\_B)

S\_B\_A = len(A) \* (mean\_A - np.mean([mean\_A, mean\_B, mean\_C])) \* (mean\_A - np.mean([mean\_A, mean\_B, mean\_C]))

S\_B\_B = len(B) \* (mean\_B - np.mean([mean\_A, mean\_B, mean\_C])) \* (mean\_B - np.mean([mean\_A, mean\_B, mean\_C]))

S\_B\_C = len(C) \* (mean\_C - np.mean([mean\_A, mean\_B, mean\_C])) \* (mean\_C - np.mean([mean\_A, mean\_B, mean\_C]))

S\_B = S\_B\_A + S\_B\_B + S\_B\_C

# Total Scatter Matrix (S\_T)

S\_T\_A = (A - np.mean([mean\_A, mean\_B, mean\_C]))[:, np.newaxis] @ (A - np.mean([mean\_A, mean\_B, mean\_C]))[:, np.newaxis].T

S\_T\_B = (B - np.mean([mean\_A, mean\_B, mean\_C]))[:, np.newaxis] @ (B - np.mean([mean\_A, mean\_B, mean\_C]))[:, np.newaxis].T

S\_T\_C = (C - np.mean([mean\_A, mean\_B, mean\_C]))[:, np.newaxis] @ (C - np.mean([mean\_A, mean\_B, mean\_C]))[:, np.newaxis].T

S\_T = S\_T\_A + S\_T\_B + S\_T\_C

# Output the results

print("Within-Class Scatter Matrix (S\_W):")

print(S\_W)

print("\n")

print("Between-Class Scatter Matrix (S\_B):")

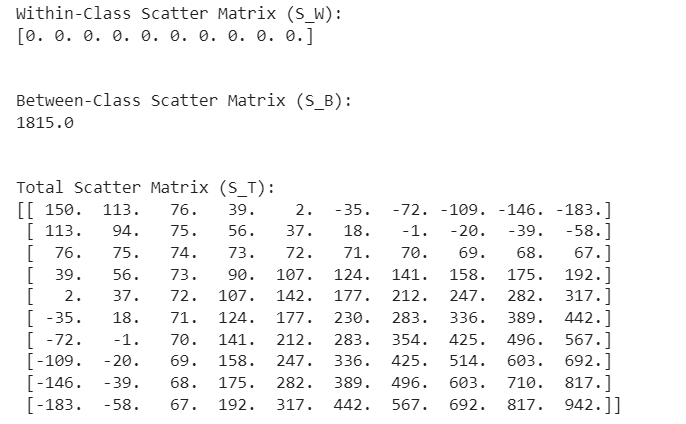
print(S\_B)

print("\n")

print("Total Scatter Matrix (S\_T):")

print(S\_T)

**OUTPUT :**



**PRACTICAL : 10**

**AIM : Given the following vectors:**

**X = [340, 230, 405, 325, 280, 195, 265, 300, 350, 310]; %sale**

**Y = [71, 65, 83, 74, 67, 56, 57, 78, 84, 65];**

**Ex. 1: Find the Linear Regression model for independent variable X and dependent variable Y.**

**Ex. 2: Predict the value of y for x = 250. Also find the residual for y4.**

**CODE :**

**Ex. 1: Find the Linear Regression model for independent variable X and dependent variable Y.**

import numpy as np

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Given data

X = np.array([340, 230, 405, 325, 280, 195, 265, 300, 350, 310])

Y = np.array([71, 65, 83, 74, 67, 56, 57, 78, 84, 65])

# Reshape X to a 2D array

X\_reshaped = X.reshape(-1, 1)

# Create and fit the linear regression model

model = LinearRegression()

model.fit(X\_reshaped, Y)

# Coefficients of the linear regression model

slope = model.coef\_[0]

intercept = model.intercept\_

# Display the linear regression model

print(f"Linear Regression Model: Y = {slope:.4f} \* X + {intercept:.4f}")

# Plot the data and the linear regression line

plt.scatter(X, Y, label='Data Points')

plt.plot(X, model.predict(X\_reshaped), color='red', label='Linear Regression Line')

plt.title('Linear Regression Model')

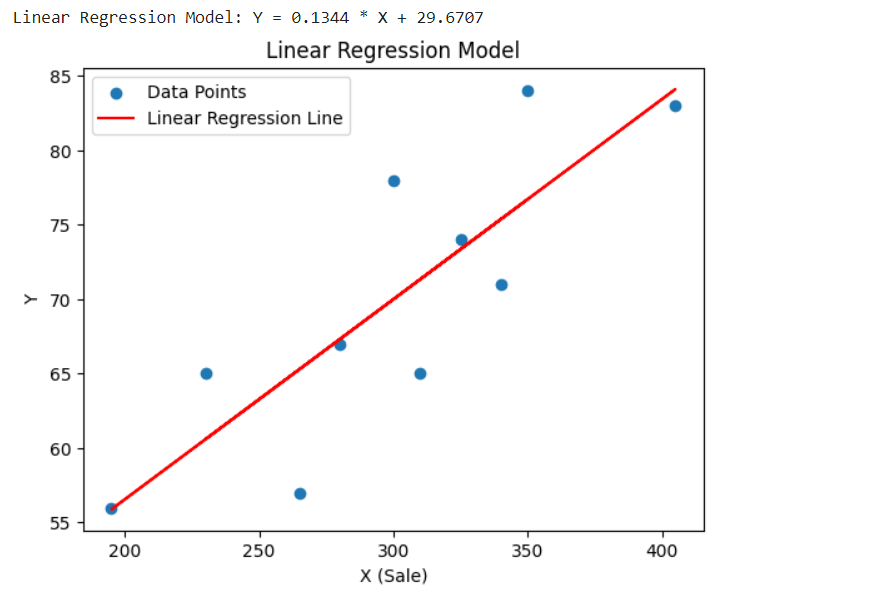
plt.xlabel('X (Sale)')

plt.ylabel('Y')

plt.legend()

plt.show()

**OUTPUT :**



Ex. 2: Predict the value of y for x = 250. Also find the residual for y4.

**CODE :**

# Predict the value of y for x = 250

x\_to\_predict = np.array([[250]])

predicted\_y = model.predict(x\_to\_predict)

print(f"Predicted value of Y for X = 250: {predicted\_y[0]:.4f}")

# Find the residual for y4 (corresponding to X[3])

residual\_y4 = Y[3] - model.predict(X\_reshaped)[3]

print(f"Residual for Y4: {residual\_y4:.4f}")

**OUTPUT:**

