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

<u>CODE :</u>

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 :

```
Arithmetic Mean of A: 5.5
Arithmetic Mean of B: 22.0
Arithmetic Mean of C: 5.5
```

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 :

```
Variance of vector A: 8.25
Variance of vector B: 132.0
Variance of vector C: 8.25
```

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 :

```
Euclidean distance between A and B: 58.86425061104575
```

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 :

```
Correlation between A and B: 0.9999999999999999
Correlation between A and C: -0.9999999999999999
```

PRACTICAL : 2

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

<u>CODE :</u>

```
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 :

```
Accuracy: 0.96
```

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
```

```python
print(f"Mean accuracy with 10-fold cross-validation: {np.mean(accuracies):.2f}")
print(f"Standard deviation in accuracy: {np.std(accuracies):.2f}")
```

OUTPUT :

```
Accuracy on the test set: 0.96
Mean accuracy with 10-fold cross-validation: 0.94
Standard deviation in accuracy: 0.04
```

PRACTICAL : 4

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

<u>CODE :</u>

```
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 :

```
Confusion Matrix:
[[ 57   6]
 [  1 107]]

Precision: 0.95
Recall: 0.99
Specificity: 0.90
Accuracy: 0.96
```

PRACTICAL : 5

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

$$\mathbf{P(C/X) = \frac{P(X/C).P(C)}{P(X)}}$$

#	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: $I(P, N) = -\frac{P}{S}\log_2\frac{P}{S} - \frac{N}{S}\log_2\frac{N}{S} = 0.940$

$Entropy: E(Outlook) = \sum_{i=1}^{v}\frac{P_i + N_i}{P + N} I(P_i, N_i) = 0.694$

$Gain\ (Outlook) = I(P, N) - E(Outlook) = 0.246$

<u>CODE :</u>

```
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 :

```
The splitting attribute at the first level is: Windy
```

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:

```
Accuracy: 0.6000

Confusion Matrix:
[[1 1]
 [1 2]]

Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.50      0.50         2
           1       0.67      0.67      0.67         3

    accuracy                           0.60         5
   macro avg       0.58      0.58      0.58         5
weighted avg       0.60      0.60      0.60         5
```

PRACTICAL : 8

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

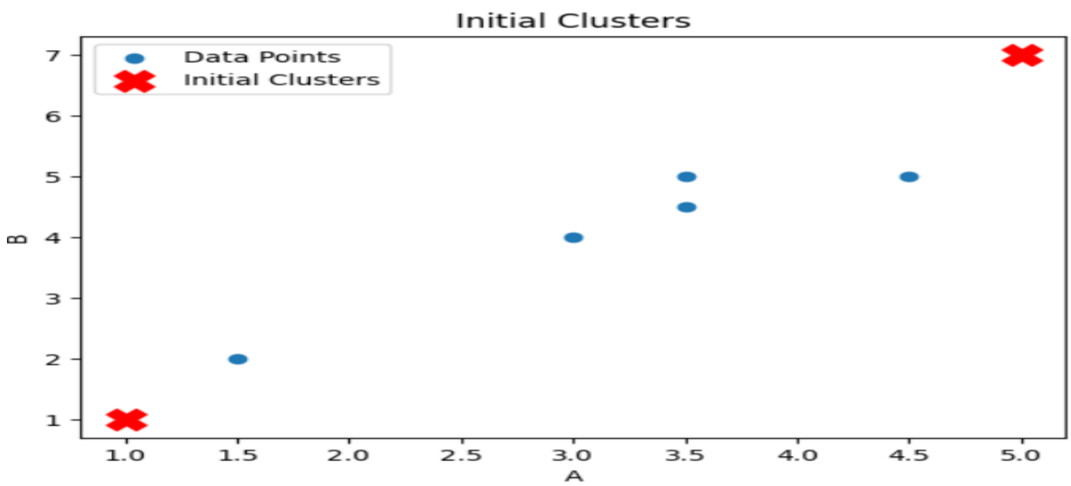
i	A	B
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

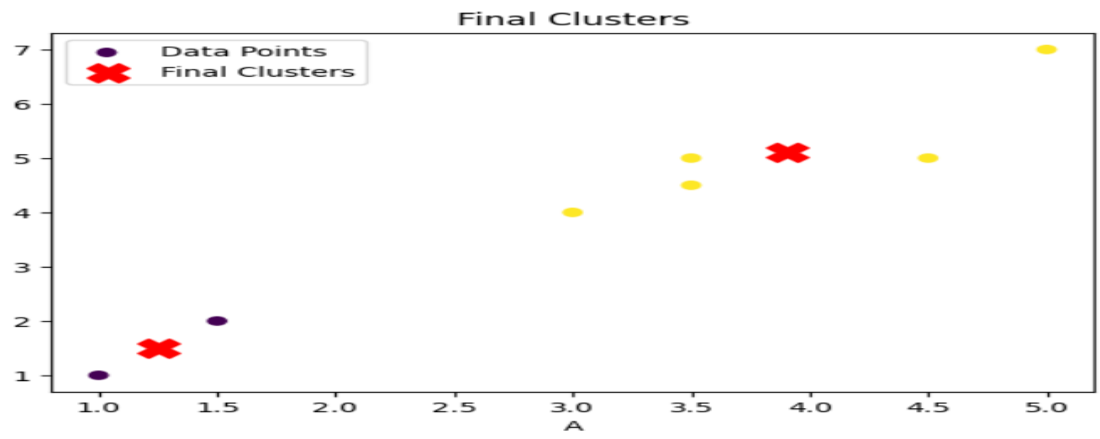
CODE :

```
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:




Final Clusters
Data Points
Final Clusters
7
6
5
4
3
2
1
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
A

PRACTICAL : 9

AIM :

Find following statistics for the data given in Exercise 1

$$Within\ Class\ Scatter: S_W = \sum_{i=1}^{C} \sum_{x \in w_i} (x - m_i)(x - m_i)^T$$

$$Between\ Class\ Scatter: S_B = \sum_{i=1}^{C} n_i (m_i - m)(m_i - m)^T$$

$$Total\ Scatter: S_T = \sum_{i=1}^{M} (x_i - m)(x_i - m)^T$$

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 :

```
Within-Class Scatter Matrix (S_W):
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


Between-Class Scatter Matrix (S_B):
1815.0


Total Scatter Matrix (S_T):
[[ 150.  113.   76.   39.    2.  -35.  -72. -109. -146. -183.]
 [ 113.   94.   75.   56.   37.   18.   -1.  -20.  -39.  -58.]
 [  76.   75.   74.   73.   72.   71.   70.   69.   68.   67.]
 [  39.   56.   73.   90.  107.  124.  141.  158.  175.  192.]
 [   2.   37.   72.  107.  142.  177.  212.  247.  282.  317.]
 [ -35.   18.   71.  124.  177.  230.  283.  336.  389.  442.]
 [ -72.   -1.   70.  141.  212.  283.  354.  425.  496.  567.]
 [-109.  -20.   69.  158.  247.  336.  425.  514.  603.  692.]
 [-146.  -39.   68.  175.  282.  389.  496.  603.  710.  817.]
 [-183.  -58.   67.  192.  317.  442.  567.  692.  817.  942.]]
```

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 y_4.

<u>CODE :</u>

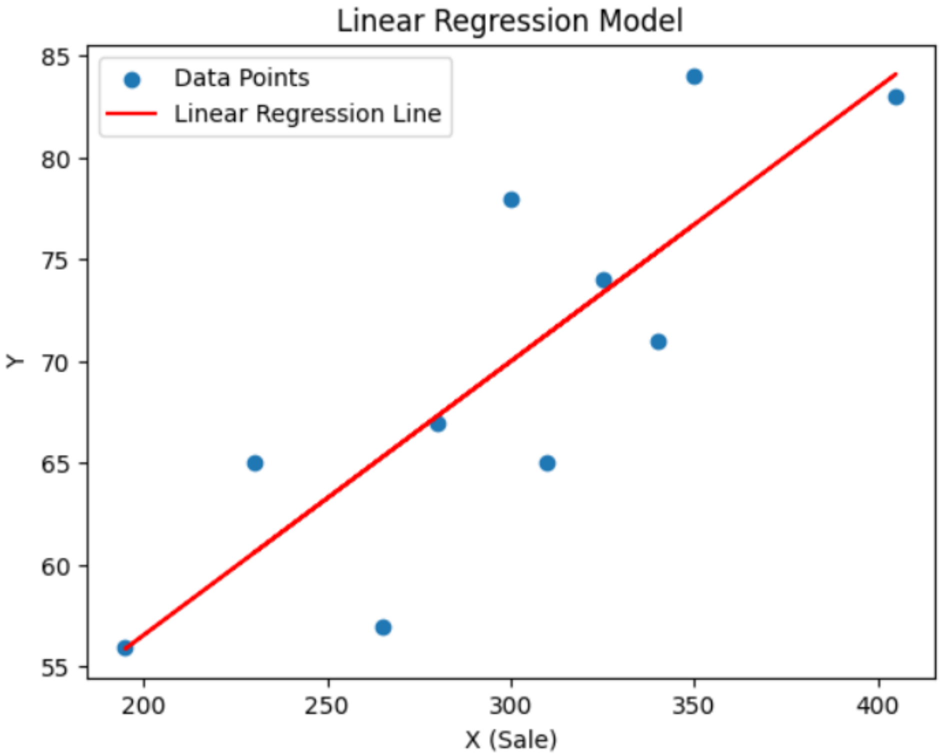
Ex. 1: Find the Linear Regression model for independent variable X and dependent variable Y.

```
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 :

```
Linear Regression Model: Y = 0.1344 * X + 29.6707
```



Ex. 2: Predict the value of y for x = 250. Also find the residual for y_4.

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:

```
Predicted value of Y for X = 250: 63.2784
Residual for Y4: 0.6392
```