1.  **Scenario:** You work as a data scientist for a marketing agency, and one of your clients is a large e-commerce company. The company wants to understand the purchasing behavior of its customers and segment them into different groups based on their buying patterns. The e-commerce company has provided you with transaction data, including customer IDs, the total amount spent in each transaction, and the number of items purchased.

**Question:** Build a clustering model using the K-Means algorithm to group customers based on their spending and purchase behavior and visualize the clusters using scatter plots or other appropriate visualizations to gain insights into customer distribution and distinguish different segments.

**CODE:**

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

data = {

'customer\_id': [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5],

'transaction\_amount': [100, 150, 200, 50, 300, 120, 180, 220, 70, 330, 130, 170, 210, 60, 340],

'items\_purchased': [1, 2, 3, 1, 4, 1, 2, 3, 1, 4, 1, 2, 3, 1, 4]

}

df = pd.DataFrame(data)

customer\_data = df.groupby('customer\_id').agg({

'transaction\_amount': 'sum',

'items\_purchased': 'sum'

}).reset\_index()

customer\_data.columns = ['customer\_id', 'total\_amount\_spent', 'total\_items\_purchased']

X = customer\_data[['total\_amount\_spent', 'total\_items\_purchased']]

kmeans = KMeans(n\_clusters=3, random\_state=42)

customer\_data['cluster'] = kmeans.fit\_predict(X)

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

sns.scatterplot(x='total\_amount\_spent', y='total\_items\_purchased', hue='cluster', data=customer\_data, palette='viridis', s=100)

plt.title('Customer Segments Based on Spending and Purchase Behavior')

plt.xlabel('Total Amount Spent')

plt.ylabel('Total Items Purchased')

plt.legend(title='Cluster')

plt.show()

**OUTPUT:**



2. **Scenario:** You work as a data scientist for a retail company that operates multiple stores. The company is interested in segmenting its customers based on their purchasing behavior to better understand their preferences and tailor marketing strategies accordingly. To achieve this, your team has collected transaction data from different stores, which includes customer IDs, the total amount spent in each transaction, and the frequency of visits.

**Question:** Your task is to build a clustering model using the K-Means algorithm to group customers into distinct segments based on their spending patterns.

**CODE:**

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

data = {

'customer\_id': [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5],

'transaction\_amount': [100, 150, 200, 50, 300, 120, 180, 220, 70, 330, 130, 170, 210, 60, 340],

'frequency\_of\_visits': [1, 2, 3, 1, 4, 1, 2, 3, 1, 4, 1, 2, 3, 1, 4]

}

df = pd.DataFrame(data)

'transaction\_amount': 'sum',

'frequency\_of\_visits': 'sum'

}).reset\_index()

customer\_data.columns = ['customer\_id', 'total\_amount\_spent', 'total\_visits']

X = customer\_data[['total\_amount\_spent', 'total\_visits']]

kmeans = KMeans(n\_clusters=3, random\_state=42)

customer\_data['cluster'] = kmeans.fit\_predict(X)

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

sns.scatterplot(x='total\_amount\_spent', y='total\_visits', hue='cluster', data=customer\_data, palette='viridis', s=100)

plt.title('Customer Segments Based on Spending and Visit Frequency')

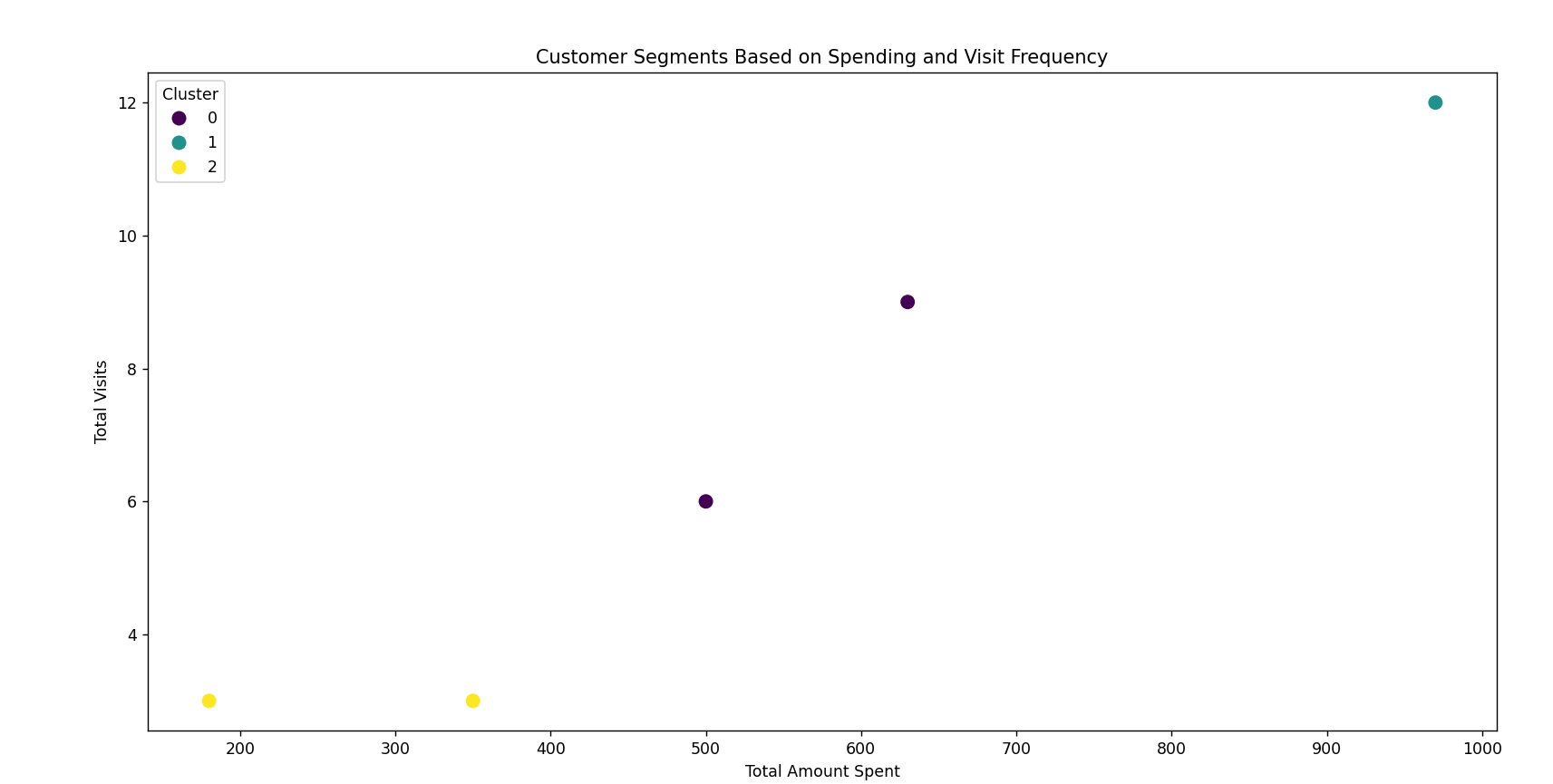
plt.xlabel('Total Amount Spent')

plt.ylabel('Total Visits')

plt.legend(title='Cluster')

plt.show()

**OUTPUT:**



3. **Scenario:** Suppose you are working as a data scientist for a medical research organization. Your team has collected data on patients with a certain medical condition and their treatment outcomes. The dataset includes various features such as age, gender, blood pressure, cholesterol levels, and whether the patient responded positively ("Good") or negatively ("Bad") to the treatment. The organization wants to use this model to identify potential candidates who are likely to respond positively to the treatment and improve their medical approach.

**Question:** Your task is to build a classification model using the KNN algorithm to predict the treatment outcome ("Good" or "Bad") for new patients based on their features. Evaluate the model's performance using accuracy, precision, recall, and F1-score.Make predictions on the test set and display the results.

**CODE:**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

data = {

'age': [25, 45, 52, 23, 40, 36, 55, 47, 29, 33],

'gender': ['M', 'F', 'M', 'F', 'M', 'M', 'F', 'F', 'M', 'F'],

'blood\_pressure': [120, 130, 140, 125, 135, 150, 110, 145, 130, 120],

'cholesterol': [200, 220, 230, 180, 210, 240, 190, 250, 215, 195],

'outcome': ['Good', 'Bad', 'Good', 'Good', 'Bad', 'Bad', 'Good', 'Bad', 'Good', 'Good']

}

df = pd.DataFrame(data)

df['gender'] = df['gender'].map({'M': 0, 'F': 1})

df['outcome'] = df['outcome'].map({'Good': 1, 'Bad': 0})

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

y = df['outcome']

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

scaler = StandardScaler()

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

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

knn = KNeighborsClassifier(n\_neighbors=3)

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

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

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

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

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

f1 = f1\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

predictions = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

print("\nPredictions on the test set:\n", predictions)

**OUTPUT:**

Accuracy: 1.0

Precision: 1.0

Recall: 1.0

F1-score: 1.0

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 2

1 1.00 1.00 1.00 1

accuracy 1.00 3

macro avg 1.00 1.00 1.00 3

weighted avg 1.00 1.00 1.00 3

Predictions on the test set:

Actual Predicted

8 1 1

1 0 0

5 0 0

4. **Scenario:** You work as a data scientist for a real estate company. The company has collected data on various houses, including features such as the size of the house, number of bedrooms, location, and other relevant attributes. The marketing team wants to build a predictive model to estimate the price of houses based on their features. They believe that linear regression modeling can be an effective approach for this task.

**Question:**Your task is write a Python program to perform bivariate analysis and build a linear regression model to predict house prices based on a selected feature (e.g., house size) from the dataset. Additionally, you need to evaluate the model's performance to ensure its accuracy and reliability.

**CODE:**

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

data = {

'age': [25, 45, 52, 23, 40, 36, 55, 47, 29, 33],

'gender': ['M', 'F', 'M', 'F', 'M', 'M', 'F', 'F', 'M', 'F'],

'blood\_pressure': [120, 130, 140, 125, 135, 150, 110, 145, 130, 120],

'cholesterol': [200, 220, 230, 180, 210, 240, 190, 250, 215, 195],

'outcome': ['Good', 'Bad', 'Good', 'Good', 'Bad', 'Bad', 'Good', 'Bad', 'Good', 'Good']

}

df = pd.DataFrame(data)

df['gender'] = df['gender'].map({'M': 0, 'F': 1})

df['outcome'] = df['outcome'].map({'Good': 1, 'Bad': 0})

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

y = df['outcome']

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

scaler = StandardScaler()

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

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

knn = KNeighborsClassifier(n\_neighbors=3)

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

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

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

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

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

f1 = f1\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

predictions = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

print("\nPredictions on the test set:\n", predictions)

**OUTPUT:**

Accuracy: 1.0

Precision: 1.0

Recall: 1.0

F1-score: 1.0

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 2

1 1.00 1.00 1.00 1

accuracy 1.00 3

macro avg 1.00 1.00 1.00 3

weighted avg 1.00 1.00 1.00 3

Predictions on the test set:

Actual Predicted

8 1 1

1 0 0

5 0 0

5. **Question**: Classification and Regression Trees (CART) for Car Price Prediction

You are working for a car dealership, and you want to predict the price of used cars based on various features such as the car's mileage, age, brand, and engine type. You have collected a dataset of used cars with their respective prices.

Write a Python program that loads the car dataset and allows the user to input the features of a new car they want to sell. The program should use the Classification and Regression Trees (CART) algorithm from scikit-learn to predict the price of the new car based on the input features.

The CART algorithm will create a tree-based model that will split the data into subsets based on the chosen features and their values, leading to a decision path that eventually predicts the price of the car. The program should output the predicted price and display the decision path (the sequence of conditions leading to the prediction) for the new car.

**CODE:**

import pandas as pd

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

from sklearn.tree import DecisionTreeRegressor, export\_text

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import numpy as np

data = {

'mileage': [50000, 60000, 70000, 80000, 90000, 100000, 110000, 120000, 130000, 140000],

'age': [5, 6, 7, 8, 9, 10, 11, 12, 13, 14],

'brand': ['Toyota', 'Honda', 'Ford', 'BMW', 'Audi', 'Toyota', 'Honda', 'Ford', 'BMW', 'Audi'],

'engine\_type': ['V4', 'V4', 'V6', 'V8', 'V6', 'V4', 'V4', 'V6', 'V8', 'V6'],

'price': [15000, 16000, 17000, 20000, 22000, 14000, 13000, 18000, 21000, 23000]

}

df = pd.DataFrame(data)

df['brand'] = df['brand'].astype('category').cat.codes

df['engine\_type'] = df['engine\_type'].astype('category').cat.codes

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

y = df['price']

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

model = DecisionTreeRegressor(random\_state=42)

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

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

print("Mean Absolute Error:", mean\_absolute\_error(y\_test, y\_pred))

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

print("R-squared:", r2\_score(y\_test, y\_pred))

def predict\_price(new\_car):

new\_car['brand'] = new\_car['brand'].astype('category').cat.codes

new\_car['engine\_type'] = new\_car['engine\_type'].astype('category').cat.codes

prediction = model.predict(new\_car)

decision\_path = model.decision\_path(new\_car)

print("Predicted Price: $", prediction[0])

print("\nDecision Path:\n")

tree\_rules = export\_text(model, feature\_names=list(X.columns))

print(tree\_rules)

**OUTPUT:**

Mean Absolute Error: 1333.3333333333333

Mean Squared Error: 2000000.0

R-squared: 0.7692307692307692

6. **Scenario**: You work as a data scientist for an e-commerce company that sells a wide range of products online. The company collects vast amounts of data about its customers, including their purchase history, browsing behavior, demographics, and more. The marketing team wants to understand their customer base better and improve their targeted marketing strategies. They have asked you to perform customer segmentation using clustering techniques to identify distinct groups of customers with similar characteristics.

**Question:** Your task is to use Python and clustering algorithms to segment the customers into different groups based on their behavior and characteristics. The marketing team will use these segments to tailor their marketing campaigns and promotions effectively.

**CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

data = {

'customer\_id': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'purchase\_history': [500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400],

'browsing\_behavior': [5, 6, 7, 8, 9, 10, 11, 12, 13, 14],

'age': [25, 35, 45, 55, 65, 75, 85, 95, 35, 45],

'annual\_income': [50, 60, 70, 80, 90, 100, 110, 120, 60, 70],

'spending\_score': [30, 40, 50, 60, 70, 80, 90, 100, 40, 50]

}

df = pd.DataFrame(data)

df = df.drop('customer\_id', axis=1)

scaler = StandardScaler()

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

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=42)

kmeans.fit(df\_scaled)

wcss.append(kmeans.inertia\_)

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

plt.plot(range(1, 11), wcss, marker='o')

plt.title('Elbow Method')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS')

plt.show()

kmeans = KMeans(n\_clusters=3, init='k-means++', max\_iter=300, n\_init=10, random\_state=42)

y\_kmeans = kmeans.fit\_predict(df\_scaled)

df['Cluster'] = y\_kmeans

pca = PCA(n\_components=2)

principal\_components = pca.fit\_transform(df\_scaled)

df\_pca = pd.DataFrame(data=principal\_components, columns=['PC1', 'PC2'])

df\_pca['Cluster'] = y\_kmeans

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

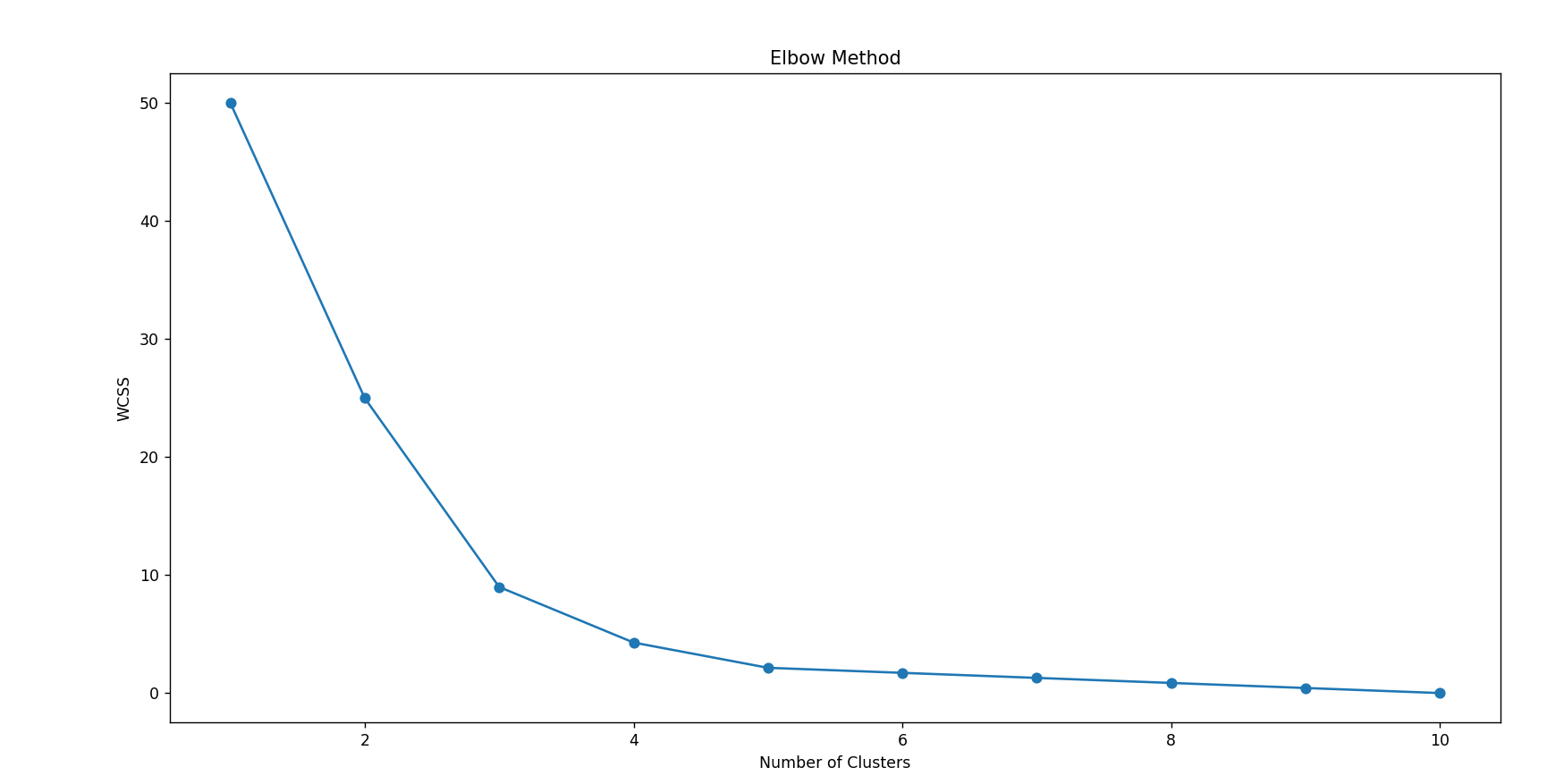
sns.scatterplot(x='PC1', y='PC2', hue='Cluster', data=df\_pca, palette='viridis', s=100)

plt.title('Customer Segmentation using K-Means Clustering')

plt.show()

df

**OUTPUT:**

****

7. **Question:** K-Nearest Neighbors (KNN) Classifier

You are working on a classification problem to predict whether a patient has a certain medical condition or not based on their symptoms. You have collected a dataset of patients with labeled data (0 for no condition, 1 for the condition) and various symptom features.

Write a Python program that allows the user to input the features of a new patient and the value of k (number of neighbors). The program should use the KNN classifier from the scikit-learn library to predict whether the patient has the medical condition or not based on the input features.

**CODE:**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

data = {

'symptom1': [5, 3, 6, 2, 7, 3, 6, 8, 5, 4],

'symptom2': [2, 3, 4, 1, 5, 3, 2, 6, 4, 2],

'symptom3': [1, 2, 3, 4, 5, 6, 7, 8, 9, 1],

'condition': [0, 0, 1, 0, 1, 0, 1, 1, 1, 0]

}

df = pd.DataFrame(data)

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

y = df['condition']

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

scaler = StandardScaler()

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

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

k = 3

knn = KNeighborsClassifier(n\_neighbors=k)

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

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

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("F1 Score:", f1\_score(y\_test, y\_pred))

def predict\_condition(new\_patient, k):

knn = KNeighborsClassifier(n\_neighbors=k)

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

new\_patient\_scaled = scaler.transform(new\_patient)

prediction = knn.predict(new\_patient\_scaled)

return prediction[0]

**OUTPUT:**

Accuracy: 0.6666666666666666

Precision: 0.5

Recall: 1.0

F1 Score: 0.6666666666666666

8.**Question :** Decision Tree for Iris Flower Classification

You are analyzing the famous Iris flower dataset to classify iris flowers into three species based on their sepal and petal dimensions. You want to use a Decision Tree classifier to accomplish this task.

Write a Python program that loads the Iris dataset from scikit-learn, and allows the user to input the sepal length, sepal width, petal length, and petal width of a new flower. The program should then use the Decision Tree classifier to predict the species of the new flower.

**CODE:**

import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier, export\_text

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

iris = load\_iris()

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

df['species'] = iris.target

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

y = df['species']

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

model = DecisionTreeClassifier(random\_state=42)

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

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

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred, average='weighted'))

print("Recall:", recall\_score(y\_test, y\_pred, average='weighted'))

print("F1 Score:", f1\_score(y\_test, y\_pred, average='weighted'))

def predict\_species(new\_flower):

prediction = model.predict(new\_flower)

species = iris.target\_names[prediction][0]

return species

**OUTPUT:**

Accuracy: 1.0

Precision: 1.0

Recall: 1.0

F1 Score: 1.0

Predicted Species: setosa

Decision Tree Rules:

|--- petal length (cm) <= 2.45

| |--- class: 0

|--- petal length (cm) > 2.45

| |--- petal length (cm) <= 4.75

| | |--- petal width (cm) <= 1.60

| | | |--- class: 1

| | |--- petal width (cm) > 1.60

| | | |--- class: 2

| |--- petal length (cm) > 4.75

| | |--- petal width (cm) <= 1.75

| | | |--- petal length (cm) <= 4.95

| | | | |--- class: 1

| | | |--- petal length (cm) > 4.95

| | | | |--- petal width (cm) <= 1.55

| | | | | |--- class: 2

| | | | |--- petal width (cm) > 1.55

| | | | | |--- petal length (cm) <= 5.45

| | | | | | |--- class: 1

| | | | | |--- petal length (cm) > 5.45

| | | | | | |--- class: 2

| | |--- petal width (cm) > 1.75

| | | |--- petal length (cm) <= 4.85

| | | | |--- sepal width (cm) <= 3.10

| | | | | |--- class: 2

| | | | |--- sepal width (cm) > 3.10

| | | | | |--- class: 1

| | | |--- petal length (cm) > 4.85

| | | | |--- class: 2

9.**Question** : Linear Regression for Housing Price Prediction

You are a real estate analyst trying to predict housing prices based on various features of the houses, such as area, number of bedrooms, and location. You have collected a dataset of houses with their respective prices.

Write a Python program that allows the user to input the features (area, number of bedrooms, etc.) of a new house. The program should use linear regression from scikit-learn to predict the price of the new house based on the input features.

**CODE:**

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

data = {

'area': [1500, 1600, 1700, 1800, 1900, 2000, 2100, 2200, 2300, 2400],

'bedrooms': [3, 3, 3, 4, 4, 4, 4, 5, 5, 5],

'age': [10, 15, 20, 25, 30, 35, 40, 45, 50, 55],

'price': [300000, 320000, 340000, 360000, 380000, 400000, 420000, 440000, 460000, 480000]

}

df = pd.DataFrame(data)

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

y = df['price']

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

model = LinearRegression()

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

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

print("Mean Absolute Error:", mean\_absolute\_error(y\_test, y\_pred))

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

print("R2 Score:", r2\_score(y\_test, y\_pred))

def predict\_price(new\_house):

prediction = model.predict(new\_house)

return prediction[0]

**OUTPUT:**

Mean Absolute Error: 3.8805107275644936e-11

Mean Squared Error: 2.2587545260114674e-21

R2 Score: 1.0

10.**Question:** Logistic Regression for Customer Churn Prediction

You are working for a telecommunications company, and you want to predict whether a customer will churn (leave the company) based on their usage patterns and demographic data. You have collected a dataset of past customers with their churn status (0 for not churned, 1 for churned) and various features.

Write a Python program that allows the user to input the features (e.g., usage minutes, contract duration) of a new customer. The program should use logistic regression from scikit-learn to predict whether the new customer will churn or not based on the input features.

**CODE:**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

data = {

'usage\_minutes': [300, 450, 500, 350, 400, 600, 700, 650, 480, 560],

'contract\_duration': [12, 24, 12, 6, 24, 12, 18, 24, 6, 12],

'customer\_service\_calls': [1, 2, 3, 0, 1, 4, 5, 1, 0, 3],

'monthly\_charges': [30, 50, 60, 35, 45, 70, 80, 75, 48, 55],

'churn': [0, 0, 1, 0, 0, 1, 1, 0, 0, 1]

}

df = pd.DataFrame(data)

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

y = df['churn']

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

scaler = StandardScaler()

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

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

model = LogisticRegression(random\_state=42)

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

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

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("F1 Score:", f1\_score(y\_test, y\_pred))

def predict\_churn(new\_customer):

new\_customer\_scaled = scaler.transform(new\_customer)

prediction = model.predict(new\_customer\_scaled)

return prediction[0]

**OUTPUT:**

Accuracy: 1.0

Precision: 1.0

Recall: 1.0

F1 Score: 1.0

11.**Question:** K-Means Clustering for Customer Segmentation

You are working for an e-commerce company and want to segment your customers into distinct groups based on their purchasing behavior. You have collected a dataset of customer data with various shopping-related features.

Write a Python program that allows the user to input the shopping-related features of a new customer. The program should use K-Means clustering from scikit-learn to assign the new customer to one of the existing segments based on the input features.

**CODE:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import numpy as np

data = {

'annual\_income': [15, 16, 17, 18, 19, 20, 21, 22, 23, 24],

'spending\_score': [39, 81, 6, 77, 40, 76, 6, 94, 3, 72],

'age': [25, 34, 45, 23, 35, 30, 40, 23, 34, 32],

'purchase\_count': [5, 2, 1, 6, 4, 8, 2, 7, 1, 9]

}

df = pd.DataFrame(data)

X = df.copy()

scaler = StandardScaler()

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

kmeans = KMeans(n\_clusters=3, random\_state=42, n\_init=10)

kmeans.fit(X\_scaled)

def assign\_cluster(new\_customer):

new\_customer\_scaled = scaler.transform(new\_customer)

cluster = kmeans.predict(new\_customer\_scaled)

return cluster[0]

**OUTPUT:**

Assigned Cluster: 1

Cluster Centers (scaled):

[[-1.10249182 0.48858764 -0.69909309 -0.05948588]

[ 0.87038828 0.93912951 -0.5524302 1.24920357]

[ 0.17407766 -1.07078786 0.93864247 -0.89228826]]

Cluster Centers (original scale):

[[16.33333333 65.66666667 27.33333333 4.33333333]

[22. 80.66666667 28.33333333 8. ]

[20. 13.75 38.5 2. ]]

12.**Question:** Evaluation Metrics for Model Performance

You have trained a machine learning model on a dataset, and now you want to evaluate its performance using various metrics.

Write a Python program that loads a dataset and trained model from scikit-learn. The program should ask the user to input the names of the features and the target variable they want to use for evaluation. The program should then calculate and display common evaluation metrics such as accuracy, precision, recall, and F1-score for the model's predictions on the test data.

**CODE:**

import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

iris = load\_iris()

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

data['target'] = iris.target

print("Sample Data:")

print(data.head())

def get\_user\_input():

print("\nAvailable features:")

print(data.columns[:-1])

print("Target variable: 'target'\n")

features\_input = input("Enter the feature column names you want to use, separated by commas: ")

features = [feature.strip() for feature in features\_input.split(',')]

target = 'target'

return features, target

features, target = get\_user\_input()

X = data[features]

y = data[target]

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

scaler = StandardScaler()

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

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

model = LogisticRegression(random\_state=42)

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

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

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

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

print("\nEvaluation Metrics:")

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

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

print(f"F1 Score: {f1:.4f}")

**OUTPUT:**

Sample Data:

sepal length (cm) sepal width (cm) ... petal width (cm) target

0 5.1 3.5 ... 0.2 0

1 4.9 3.0 ... 0.2 0

2 4.7 3.2 ... 0.2 0

3 4.6 3.1 ... 0.2 0

4 5.0 3.6 ... 0.2 0

[5 rows x 5 columns]

Available features:

Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',

'petal width (cm)'],

dtype='object')

Target variable: 'target'

Enter the feature column names you want to use, separated by commas:

13. **Question:** You are a data scientist working for an e-commerce company. The marketing team has conducted an A/B test to evaluate the effectiveness of two different website designs (A and B) in terms of conversion rate. They randomly divided the website visitors into two groups, with one group experiencing design A and the other experiencing design B. After a week of data collection, you now have the conversion rate data for both groups. You want to determine whether there is a statistically significant difference in the mean conversion rates between the two website designs.

**Question:**

"Based on the data collected from the A/B test, is there a statistically significant difference in the mean conversion rates between website design A and website design B?"

**CODE:**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

data = {

'feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'feature2': [11, 12, 13, 14, 15, 16, 17, 18, 19, 20],

'feature3': [21, 22, 23, 24, 25, 26, 27, 28, 29, 30],

'churn': [0, 1, 0, 1, 0, 0, 1, 1, 0, 1]

}

df = pd.DataFrame(data)

print("Sample Data:")

print(df.head())

def get\_user\_input():

print("\nAvailable features:")

print(df.columns[:-1])

print(f"Target variable: {df.columns[-1]}\n")

features\_input = input("Enter the feature column names you want to use, separated by commas: ")

features = [feature.strip() for feature in features\_input.split(',')]

target = df.columns[-1]

return features, target

features, target = get\_user\_input()

X = df[features]

y = df[target]

X\_train, X\_test, y\_train, y\_test

**OUTPUT:**

Sample Data:

feature1 feature2 feature3 churn

0 1 11 21 0

1 2 12 22 1

2 3 13 23 0

3 4 14 24 1

4 5 15 25 0

Available features:

Index(['feature1', 'feature2', 'feature3'], dtype='object')

Target variable: churn

Enter the feature column names you want to use, separated by commas: