**Program:**

import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

# Sample dataset

data = {

'age': [25, np.nan, 28, 35, 40],

'salary': [50000, 60000, np.nan, 80000, 100000],

'city': ['New York', 'San Francisco', np.nan, 'Los Angeles', 'Chicago'],

'purchased': [0, 1, 1, 0, 1]

}

df = pd.DataFrame(data)

print("Original DataFrame:")

print(df)

# Step 1: Handling Missing Data

# Define columns by type

numeric\_features = ['age', 'salary']

categorical\_features = ['city']

# Numeric imputer (mean strategy)

numeric\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='mean')),

('scaler', StandardScaler())

])

# Categorical imputer (most frequent strategy) + One-Hot Encoding

categorical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='most\_frequent')),

('onehot', OneHotEncoder(handle\_unknown='ignore'))

])

# Combine preprocessors in a column transformer

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_features),

('cat', categorical\_transformer, categorical\_features)

]

)

# Step 2: Applying the Preprocessor

# Separate features and target variable

X = df.drop(columns=['purchased'])

y = df['purchased']

# Preprocess the features

X\_preprocessed = preprocessor.fit\_transform(X)

print("\nPreprocessed Features:")

print(X\_preprocessed)

# Step 3: Integrating into a Full Pipeline

# Example: Pipeline including preprocessing and a classifier

from sklearn.ensemble import RandomForestClassifier

model = Pipeline(steps=[

('preprocessor', preprocessor),

('classifier', RandomForestClassifier())

])

# Fitting the pipeline to the data

model.fit(X, y)

# Example Prediction

sample\_data = pd.DataFrame({

'age': [30],

'salary': [70000],

'city': ['Los Angeles']

})

print("\nPrediction for New Sample:")

print(model.predict(sample\_data))

**Output**:

A screenshot of a computer

Description automatically generated

**Program**:

# Import necessary libraries

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier

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

import pandas as pd

# Load the Iris dataset

iris = load\_iris()

# Convert to a DataFrame for better readability

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

data['target'] = iris.target

data['target\_names'] = data['target'].apply(lambda x: iris.target\_names[x])

# Display the first few rows of the dataset

print("First few rows of the dataset:")

print(data.head())

# Separate features (X) and target labels (y)

X = iris.data # Features

y = iris.target # Labels

# Split into training and testing datasets

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

# Initialize and train a Decision Tree Classifier

classifier = DecisionTreeClassifier(random\_state=42)

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

# Make predictions

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

# Evaluate the model

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

print("\nClassification Report:")

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

**output**:

A screenshot of a computer

Description automatically generated

**Program**:

# Import necessary libraries

from sklearn.datasets import load\_iris

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

from sklearn.naive\_bayes import GaussianNB

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

# Load the Iris dataset

iris = load\_iris()

X = iris.data # Features

y = iris.target # Labels

# Split into training and testing datasets

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

# Initialize and train a Naive Bayes classifier

classifier = GaussianNB()

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

# Make predictions

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

# Evaluate the model

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

print("\nClassification Report:")

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

**output**:

A screenshot of a computer program

Description automatically generated

**Program**:

# Import necessary libraries

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

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

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

X = iris.data # Features

y = iris.target # Labels

# Split into training and testing datasets

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

# Initialize and train a Decision Tree classifier

classifier = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=42)

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

# Make predictions

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

# Evaluate the model

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

print("\nClassification Report:")

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

# Visualize the Decision Tree

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

plot\_tree(classifier, feature\_names=iris.feature\_names, class\_names=iris.target\_names, filled=True)

plt.title("Decision Tree Visualization")

plt.show()

**output**:  
A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Program**:

# Import necessary libraries

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.ensemble import IsolationForest

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

# Load the Iris dataset

iris = load\_iris()

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

data['target'] = iris.target

# Display the original dataset shape

print(f"Original Dataset Shape: {data.shape}")

# Step 1: Feature Pruning (Remove irrelevant features)

# Use SelectKBest to keep only the top 2 features based on mutual information

X = data[iris.feature\_names] # Features

y = data['target'] # Target labels

selector = SelectKBest(mutual\_info\_classif, k=2)

X\_pruned = selector.fit\_transform(X, y)

# Update the dataset with pruned features

data\_pruned\_features = pd.DataFrame(X\_pruned, columns=['Feature\_1', 'Feature\_2'])

data\_pruned\_features['target'] = y

print(f"After Feature Pruning (Top 2 Features): {data\_pruned\_features.shape}")

# Step 2: Instance Pruning (Remove outliers)

# Use Isolation Forest to detect outliers

iso = IsolationForest(contamination=0.05, random\_state=42) # Assume 5% contamination

outlier\_predictions = iso.fit\_predict(data\_pruned\_features[['Feature\_1', 'Feature\_2']])

# Keep only the inliers

data\_final = data\_pruned\_features[outlier\_predictions == 1]

print(f"After Instance Pruning (Outlier Removal): {data\_final.shape}")

# Display the final dataset

print("\nFinal Dataset:")

print(data\_final.head())

**output**:

A screenshot of a computer

Description automatically generated

**Program**:

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

X = iris.data # Features

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

# Step 1: Apply K-Means clustering

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

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

# Step 2: Evaluate clustering with Silhouette Score

silhouette\_avg = silhouette\_score(X, data['Cluster'])

print(f"Silhouette Score: {silhouette\_avg:.2f}")

# Step 3: Visualize the clusters (using the first two features for simplicity)

plt.figure(figsize=(8, 6))

plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=data['Cluster'], cmap='viridis', s=50)

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

plt.xlabel(iris.feature\_names[0])

plt.ylabel(iris.feature\_names[1])

plt.title("K-Means Clustering (Iris Dataset)")

plt.legend()

plt.show()

**output**:



A screen shot of a computer screen

Description automatically generated

**Program**:

# Import necessary libraries

import pandas as pd

from mlxtend.frequent\_patterns import apriori, association\_rules

# Sample transactional dataset

data = {

'Transaction': [1, 2, 3, 4, 5],

'Milk': [1, 0, 1, 0, 1],

'Bread': [1, 1, 0, 1, 1],

'Butter': [0, 1, 1, 0, 1],

'Cheese': [0, 0, 1, 1, 0]

}

df = pd.DataFrame(data).set\_index('Transaction')

# Step 1: Apply the Apriori algorithm to find frequent itemsets

frequent\_itemsets = apriori(df, min\_support=0.4, use\_colnames=True)

print("Frequent Itemsets:")

print(frequent\_itemsets)

# Step 2: Generate association rules

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.6)

print("\nAssociation Rules:")

print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

# Step 3: Filter rules for specific conditions (optional)

filtered\_rules = rules[rules['lift'] > 1.2]

print("\nFiltered Rules (Lift > 1.2):")

print(filtered\_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

**output**:

A blue text on a white background

Description automatically generated

**Program**:

# Import necessary libraries

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

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

from sklearn.naive\_bayes import MultinomialNB

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

# Sample dataset

data = {

"Text": [

"I love this product, it is amazing!",

"This is the worst experience I've had.",

"Great quality and fantastic service.",

"Terrible, I will never buy this again.",

"Excellent item, highly recommend it."

],

"Sentiment": [1, 0, 1, 0, 1] # 1 = Positive, 0 = Negative

}

df = pd.DataFrame(data)

# Step 1: Preprocessing and Feature Extraction

# Convert text into a bag of words representation

vectorizer = CountVectorizer()

X\_counts = vectorizer.fit\_transform(df['Text'])

# Transform counts to TF-IDF representation

tfidf\_transformer = TfidfTransformer()

X\_tfidf = tfidf\_transformer.fit\_transform(X\_counts)

# Step 2: Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_tfidf, df['Sentiment'], test\_size=0.3, random\_state=42)

# Step 3: Train a Naive Bayes Classifier

classifier = MultinomialNB()

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

# Step 4: Make Predictions

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

# Step 5: Evaluate the Model

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

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

**output**:

A close up of a text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Program**:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import DBSCAN

from sklearn.preprocessing import StandardScaler

# Example dataset with spatial data (latitude and longitude)

data = {

'Latitude': [37.77, 37.76, 37.78, 37.79, 40.73, 40.74, 40.75, 40.76, 34.05],

'Longitude': [-122.42, -122.43, -122.41, -122.44, -73.99, -73.98, -73.97, -73.96, -118.25]

}

df = pd.DataFrame(data)

# Step 1: Normalize data

scaler = StandardScaler()

coords = scaler.fit\_transform(df[['Latitude', 'Longitude']])

# Step 2: Apply DBSCAN

dbscan = DBSCAN(eps=0.5, min\_samples=2) # eps is the maximum distance for a point to be in a cluster

clusters = dbscan.fit\_predict(coords)

# Add clusters to the dataframe

df['Cluster'] = clusters

# Step 3: Visualize the clusters

plt.figure(figsize=(8, 6))

plt.scatter(df['Longitude'], df['Latitude'], c=df['Cluster'], cmap='viridis', s=100)

plt.xlabel('Longitude')

plt.ylabel('Latitude')

plt.title('Spatial Clustering with DBSCAN')

plt.colorbar(label='Cluster')

plt.show()

# Display results

print(df)

**output**:

A screen shot of a computer

Description automatically generated

**Program**:

# Import necessary libraries

import networkx as nx

import matplotlib.pyplot as plt

# Create a simple undirected graph (social network)

G = nx.Graph()

# Add nodes (individuals)

G.add\_nodes\_from(['Alice', 'Bob', 'Charlie', 'David', 'Eve'])

# Add edges (relationships between individuals)

G.add\_edges\_from([('Alice', 'Bob'), ('Alice', 'Charlie'), ('Bob', 'David'), ('Charlie', 'Eve'), ('David', 'Eve')])

# Step 1: Visualize the Network

plt.figure(figsize=(8, 6))

nx.draw(G, with\_labels=True, node\_size=2000, node\_color='skyblue', font\_size=12, font\_weight='bold', edge\_color='gray')

plt.title("Social Network of Individuals")

plt.show()

# Step 2: Compute Centrality Metrics

degree\_centrality = nx.degree\_centrality(G)

betweenness\_centrality = nx.betweenness\_centrality(G)

closeness\_centrality = nx.closeness\_centrality(G)

# Print centrality measures

print("Degree Centrality:", degree\_centrality)

print("Betweenness Centrality:", betweenness\_centrality)

print("Closeness Centrality:", closeness\_centrality)

# Step 3: Find Communities (using Louvain method for community detection)

from community import community\_louvain

partition = community\_louvain.best\_partition(G)

# Step 4: Visualize Communities

plt.figure(figsize=(8, 6))

pos = nx.spring\_layout(G)

nx.draw(G, pos, with\_labels=True, node\_size=2000, node\_color=[partition[node] for node in G.nodes], font\_size=12, font\_weight='bold', edge\_color='gray', cmap=plt.cm.rainbow)

plt.title("Community Detection in Social Network")

plt.show()

**output**:

Degree Centrality: {'Alice': 0.4, 'Bob': 0.6, 'Charlie': 0.6, 'David': 0.6, 'Eve': 0.6}

Betweenness Centrality: {'Alice': 0.0, 'Bob': 0.2, 'Charlie': 0.2, 'David': 0.0, 'Eve': 0.0}

Closeness Centrality: {'Alice': 0.75, 'Bob': 0.8333333333333334, 'Charlie': 0.75, 'David': 0.75, 'Eve': 0.75}