1

Classification: Decision Tree Dataset: madfhantr.csv Dream Housing Finance company deals in all kinds of home loans. They have presence across all urban, semi urban and rural areas. Customer first applies for home loan and after that company validates the customer eligibility for loan. Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have provided a dataset to identify the customers segments that are eligible for loan amount so that they can specifically target these customers.

```
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
from sklearn.tree import plot_tree
import matplotlib.pylab as plt
# Load the dataset
data = pd.read_csv('madfhantr.csv')
# Show initial dataset info and check for missing values
print(data.head())
print(data.isnull().sum())  # Check missing values
print(data.info())  # Dataset summary
print(data.describe())  # Statistical summary of numeric columns
# Fill missing values in numerical columns with the median
data['LoanAmount'] = data['LoanAmount'].fillna(data['LoanAmount'].median())
data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].median())
# Fill missing values in categorical columns with the mode
data['Gender'] = data['Gender'].fillna(data['Gender'].mode()[0])
data['Married'] = data['Married'].fillna(data['Married'].mode()[0])
data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0])
data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].mode()[0])
# One-hot encoding for categorical variables
data = pd.get_dummies(data, columns=['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area',
'Dependents'], drop_first=True)
# Define features (X) and target (y)
x = data.drop(['Loan_ID', 'Loan_Status'], axis=1)
y = data['Loan_Status'].map({'Y': 1, 'N': 0})
# Split the data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
# Initialize and train the Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42)
clf.fit(x_train, y_train)
# Predict on the test set
y_pred = clf.predict(x_test)
# Evaluate the model's performance
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
# Visualize the decision tree
plt.figure(figsize=(20, 10))
plot_tree(clf, feature_names=x.columns, class_names=['Not Eligible', 'Eligible'], filled=True)
plt.show()
```

2

Classification: Naïve Bayes Dataset: NaiveBayes.csv Use probabilistic approach to implement Classifier model. Evaluate the performance of the model.

```
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
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("NaiveBayes.csv")

# Handle missing values
data.fillna(data.median(), inplace=True)
for column in data.select_dtypes(include=['object']).columns:
    data[column].fillna(data[column].mode()[0], inplace=True)

# Define features and target variable
X = data.drop(['Purchased'], axis=1)
y = data['Purchased']

# Split the dataset 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)

# Initialize and train the Naïve Bayes classifier
nb = GaussianNB()
nb.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Visualize the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Target', 'Target'], yticklabels=['Not Target',
'Target'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

3

Clustering: K-Means Dataset: Cities_r2.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Apply K-Means clustering algorithms (based on total_graduates) to find the group of customers.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Step 1: Load the dataset
data = pd.read_csv('Cities_r2.csv')
# Step 2: Data Preprocessing
# Check for missing values and handle them if necessary
print("Missing values:\n", data.isnull().sum())
data = data.dropna()  # Drop rows with missing values (or you can fill them if necessary)
# Apply Label Encoding if there are any categorical columns
if 'City' in data.columns:
    le = LabelEncoder()
    data['City'] = le.fit_transform(data['City'])
# Step 3: Feature Selection for Clustering
# We focus on the 'total_graduates' column
X = data[['total_graduates']]
# Standardize the data to improve K-Means performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 4: Determine Optimal Clusters using the Elbow Method
inertia = []
for n in range(1, 11):
    kmeans = KMeans(n_clusters=n, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
# Plot the Elbow graph
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
# Step 5: Apply K-Means with chosen number of clusters
# (Select an appropriate number of clusters based on the elbow plot, e.g., 3)
optimal_clusters = 3  # Adjust this based on the elbow plot
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
data['Cluster'] = kmeans.fit_predict(X_scaled)
# Step 6: Analyze and Visualize the Clusters
print("Cluster Centers:\n", kmeans.cluster_centers_)
print("\nData grouped by clusters:\n", data.groupby('Cluster')['total_graduates'].mean())
# Visualize the clusters
plt.figure(figsize=(8, 5))
plt.scatter(X_scaled, [0]*len(X_scaled), c=data['Cluster'], cmap='viridis')
plt.title('K-Means Clustering based on Total Graduates')
plt.xlabel('Total Graduates (Scaled)')
plt.ylabel('Cluster')
plt.show()
```

4

Clustering: Hierarchical Dataset: Cities_r2.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Apply Hierarchical clustering algorithms (based on effective_literacy_rate_total column) to find the group of customers.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import matplotlib.pyplot as plt
import seaborn as sns
# Step 1: Load and Inspect the Dataset
data = pd.read_csv('Cities_r2.csv')
print("Missing values:\n", data.isnull().sum())
# Drop rows with missing values in `effective_literacy_rate_total`
data = data.dropna(subset=['effective_literacy_rate_total'])
# Step 2: Select and Scale the Feature for Clustering
# Here we use only the 'effective_literacy_rate_total' column
X = data[['effective_literacy_rate_total']]
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 3: Perform Hierarchical Clustering
# Using Ward's method for hierarchical clustering
Z = linkage(X_scaled, method='ward')
# Step 4: Plot the Dendrogram
plt.figure(figsize=(12, 8))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('City Index')
plt.ylabel('Distance')
dendrogram(Z, truncate_mode='lastp', p=12, leaf_rotation=90., leaf_font_size=10., show_contracted=True)
plt.show()
# Step 5: Assign Clusters
# Choose the number of clusters (e.g., 3)
num_clusters = 3
data['Hierarchical_Cluster'] = fcluster(Z, num_clusters, criterion='maxclust')
# Step 6: Visualize the Clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x='effective_literacy_rate_total', y='total_graduates', hue='Hierarchical_Cluster',
palette='Set2')
plt.title('Hierarchical Clustering based on Effective Literacy Rate')
plt.xlabel('Effective Literacy Rate Total')
plt.ylabel('Total Graduates')
plt.show()
```

5

Clustering: K-Means Dataset: Cities_r2.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Apply K-Means clustering algorithms (based on effective_literacy_rate_total column) to find the group of customers.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
# Step 1: Load and Inspect the Dataset
data = pd.read_csv('Cities_r2.csv')
print("Missing values:\n", data.isnull().sum())
# Drop rows with missing values in `effective_literacy_rate_total`
data = data.dropna(subset=['effective_literacy_rate_total'])
# Step 2: Select and Scale the Feature for Clustering
X = data[['effective_literacy_rate_total']]
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 3: Determine the Optimal Number of Clusters Using the Elbow Method
inertia = []
range_n_clusters = range(1, 10)
for k in range_n_clusters:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
# Plot the Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(range_n_clusters, inertia, marker='o')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
# Step 4: Apply K-Means Clustering with the Optimal Number of Clusters (e.g., 3 based on elbow method)
optimal_clusters = 3  # Choose based on the elbow plot
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
data['KMeans_Cluster'] = kmeans.fit_predict(X_scaled)
# Step 5: Visualize the Clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x='effective_literacy_rate_total', y='total_graduates', hue='KMeans_Cluster', palette='Set2')
plt.title('K-Means Clustering based on Effective Literacy Rate')
plt.xlabel('Effective Literacy Rate Total')
plt.ylabel('Total Graduates')
plt.legend(title='Cluster')
plt.show()
```

6

Clustering: Hierarchical Dataset : hitters.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Apply Hierarchical clustering algorithms (based on CRuns column) to find the group of players.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
# Step 1: Load and Inspect the Data
data = pd.read_csv('hitters.csv')
# Display the first few rows to understand the structure
print(data.head())
# Check for missing values
print("Missing values per column:\n", data.isnull().sum())
# Step 2: Handle Missing Values and Encode Categorical Variables
# Fill missing values in numerical columns, e.g., 'Salary'
data['Salary'] = data['Salary'].fillna(data['Salary'].median())
# Label encode categorical columns
label_encoder = LabelEncoder()
data['League'] = label_encoder.fit_transform(data['League'])
data['NewLeague'] = label_encoder.fit_transform(data['NewLeague'])
data['Division'] = label_encoder.fit_transform(data['Division'])
# Step 3: Select 'CRuns' Column for Clustering
# Since the goal is to group players based on their career runs, we’ll use the 'CRuns' column.
X = data[['CRuns']]
# Step 4: Scale the Data
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 5: Plot a Dendrogram to Decide on Clusters
# Plot the dendrogram to help determine the number of clusters
plt.figure(figsize=(10, 7))
dendrogram(linkage(X_scaled, method='ward'))
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Players')
plt.ylabel('Euclidean Distances')
plt.show()
# Step 6: Apply Hierarchical Clustering
# Based on the dendrogram, let’s assume we decide to form 3 clusters.
hierarchical = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
data['Cluster'] = hierarchical.fit_predict(X_scaled)
# Display the first few rows with the assigned clusters
print(data[['CRuns', 'Cluster']].head())
# Step 7: Visualize the Clusters
# Visualize the clusters to see how players are grouped based on CRuns
plt.figure(figsize=(8, 6))
sns.scatterplot(x=data['CRuns'], y=data['Cluster'], hue=data['Cluster'], palette='viridis', s=100, alpha=0.7)
plt.title('Hierarchical Clustering of Players Based on Career Runs')
plt.xlabel('Career Runs (CRuns)')
plt.ylabel('Cluster')
plt.legend(title='Cluster')
plt.show()
```

7

Clustering: K-Means Dataset : Social_Network_Ads.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Apply K-Means clustering algorithms (based on EstimatedSalary column) to find the group of users.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
# Step 1: Load the Data
data = pd.read_csv('Social_Network_Ads.csv')
# Display the first few rows of the data to understand its structure
print(data.head())
# Check for missing values
print("Missing values per column:\n", data.isnull().sum())
# Step 2: Apply Label Encoding if Necessary
# Assuming 'Gender' is a categorical variable that needs encoding
label_encoder = LabelEncoder()
data['Gender'] = label_encoder.fit_transform(data['Gender'])
# Step 3: Select the 'EstimatedSalary' Column for Clustering
# Since the goal is to cluster users based on 'EstimatedSalary', we use this column.
X = data[['EstimatedSalary']]
# Step 4: Scale the Data
# Standardize the data for better clustering performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 5: Determine Optimal Number of Clusters Using Elbow Method
# Plot the Within-Cluster-Sum of Squared Errors (WCSS) to find the optimal number of clusters
wcss = []
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_)
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
# Step 6: Apply K-Means Clustering
# Based on the elbow method, assume we decide to use 3 clusters
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
data['Cluster'] = kmeans.fit_predict(X_scaled)
# Display the first few rows with assigned clusters
print(data[['EstimatedSalary', 'Cluster']].head())
# Step 7: Evaluate Clustering with Silhouette Score
# Calculate the silhouette score to evaluate clustering quality
score = silhouette_score(X_scaled, data['Cluster'])
print(f'Silhouette Score: {score:.2f}')
```

```
# Step 8: Visualize the Clusters
plt.figure(figsize=(8, 6))
plt.scatter(data['EstimatedSalary'], data['Cluster'], c=data['Cluster'], cmap='viridis', s=100, alpha=0.7)
plt.title('K-Means Clustering of Users Based on Estimated Salary')
plt.xlabel('Estimated Salary')
plt.ylabel('Cluster')
plt.colorbar(label='Cluster')
plt.show()
```

8

Clustering: Hierarchical Dataset : 50_Startups.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Apply Hierarchical clustering algorithms (based on PROFIT column) to find the group of start-ups.

```
# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
# Step 1: Load the Data
data = pd.read_csv('50_Startups.csv')
# Display the first few rows of the data to understand its structure
print(data.head())
# Step 2: Check for Missing Values
print("Missing values per column:\n", data.isnull().sum())
# Step 3: Apply Label Encoding if Necessary
# Encode the 'State' column if it's a categorical variable
if 'State' in data.columns:
    label_encoder = LabelEncoder()
    data['State'] = label_encoder.fit_transform(data['State'])
# Step 4: Select 'PROFIT' Column for Clustering
# We only use the 'PROFIT' column as per the instructions
X = data[['PROFIT']]
# Step 5: Standardize the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 6: Apply Hierarchical Clustering
# Create the linkage matrix using the Ward method
linked = linkage(X_scaled, method='ward')
# Plot the Dendrogram to help determine the number of clusters
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Index of Startups')
plt.ylabel('Euclidean Distance')
plt.show()
# Step 7: Apply Agglomerative Clustering Based on the Dendrogram
# Based on the dendrogram, we choose a number of clusters, for example 3
cluster_model = AgglomerativeClustering(n_clusters=3, linkage='ward')
data['Cluster'] = cluster_model.fit_predict(X_scaled)
# Step 8: Display Clustered Data
print(data[['PROFIT', 'Cluster']].head())
# Step 9: Visualize the Clusters
plt.figure(figsize=(10, 6))
plt.scatter(data['PROFIT'], data['Cluster'], c=data['Cluster'], cmap='viridis', s=100, alpha=0.7)
plt.title('Hierarchical Clustering of Startups Based on Profit')
plt.xlabel('PROFIT')
plt.ylabel('Cluster')
plt.colorbar(label='Cluster')
plt.show()
```

9

Regression: Simple Linear Dataset : diabetes.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Use any one feature of the dataset to train and test the regression model. Also calculate coefficients, residual sum of squares and the coefficient of determination

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
# Load the dataset
data = pd.read_csv('diabetes.csv')
# Display basic information about the dataset
print("Dataset preview:")
print(data.head())
print("\nDataset information:")
print(data.info())
# Check for missing values
print("\nMissing values in each column:\n", data.isnull().sum())
# Choose a single feature for simple linear regression (e.g., 'BMI')
X = data[['BMI']]  # Independent variable
y = data['Outcome']  # Dependent variable (target)
# 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)
# Initialize the Linear Regression model
model = LinearRegression()
# Fit the model to the training data
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Display the intercept and coefficient of the model
print("\nRegression Intercept:", model.intercept_)
print("Regression Coefficient:", model.coef_[0])
# Calculate Residual Sum of Squares (RSS) and R-squared (R²)
rss = mean_squared_error(y_test, y_pred) * len(y_test)  # Total squared errors
r2 = r2_score(y_test, y_pred)  # R² score
# Display RSS and R² values
print("\nResidual Sum of Squares (RSS):", rss)
print("Coefficient of Determination (R²):", r2)
# Plot the data points and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Data Points')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel("BMI")
plt.ylabel("Outcome")
plt.title("Relationship between BMI and Outcome (Diabetes)")
plt.legend()
plt.show()
```

10

Regression: Simple Linear Dataset: 1.01. Simple linear regression Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Explore the relationship between students SAT score and GPA using linear regression model. Also display the regression results and plot the regression line.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Load the dataset
data = pd.read_csv('1.01. Simple linear regression.csv')
# Display the first few rows of the data
print("Dataset preview:")
print(data.head())
# Check for missing values
print("\nMissing values in each column:\n", data.isnull().sum())
# Select the feature (SAT) and target (GPA)
X = data[['SAT']]  # Independent variable (SAT score)
y = data['GPA']    # Dependent variable (GPA)
# Initialize the Linear Regression model
model = LinearRegression()
# Fit the model to the data
model.fit(X, y)
# Make predictions
y_pred = model.predict(X)
# Display the intercept and coefficient of the model
print("\nRegression Intercept:", model.intercept_)
print("Regression Coefficient:", model.coef_[0])
# Calculate Residual Sum of Squares (RSS) and R-squared (R²)
rss = mean_squared_error(y, y_pred) * len(y)  # Total squared errors
r2 = r2_score(y, y_pred)  # R² score
# Display RSS and R² values
print("\nResidual Sum of Squares (RSS):", rss)
print("Coefficient of Determination (R²):", r2)
# Plot the data points and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel("SAT Score")
plt.ylabel("GPA")
plt.title("Relationship between SAT Score and GPA")
plt.legend()
plt.show()
```

11

Clustering: K-Means We have given a collection of 8 points. P1=[0.1,0.6] P2=[0.15,0.71] P3=[0.08,0.9] P4=[0.16, 0.85] P5=[0.2,0.3] P6=[0.25,0.5] P7=[0.24,0.1] P8=[0.3,0.2]. Perform the k-mean clustering with initial centroids as m1=P1 =Cluster#1=C1 and m2=P8=cluster#2=C2. Answer the following 1] Which cluster does P6 belongs to? 2] What is the population of a cluster around m2? 3] What is the updated value of m1 and m2?

```
import numpy as np

# Given points
points = {
    'P1': np.array([0.1, 0.6]),
    'P2': np.array([0.15, 0.71]),
    'P3': np.array([0.08, 0.9]),
    'P4': np.array([0.16, 0.85]),
    'P5': np.array([0.2, 0.3]),
    'P6': np.array([0.25, 0.5]),
    'P7': np.array([0.24, 0.1]),
    'P8': np.array([0.3, 0.2])
}
# Initial centroids (m1 = P1, m2 = P8)
m1 = np.array([0.1, 0.6])  # centroid of Cluster 1 (C1)
m2 = np.array([0.3, 0.2])  # centroid of Cluster 2 (C2)
# Function to calculate Euclidean distance
def euclidean_distance(p1, p2):
    return np.sqrt(np.sum((p1 - p2) ** 2))
# Step 1: Assign points to the closest centroid
clusters = {1: [], 2: []}
for label, point in points.items():
    dist_m1 = euclidean_distance(point, m1)
    dist_m2 = euclidean_distance(point, m2)

    # Assign to the nearest centroid
    if dist_m1 < dist_m2:
        clusters[1].append(label)  # Assign to Cluster 1 (C1)
    else:
        clusters[2].append(label)  # Assign to Cluster 2 (C2)
# Display assigned clusters
print(f"Initial clusters:")
print(f"Cluster 1 (C1): {clusters[1]}")
print(f"Cluster 2 (C2): {clusters[2]}")
# Step 2: Update centroids based on the assigned clusters
def calculate_centroid(cluster_points):
    cluster_coords = [points[point] for point in cluster_points]
    return np.mean(cluster_coords, axis=0)
# Calculate new centroids
new_m1 = calculate_centroid(clusters[1])
new_m2 = calculate_centroid(clusters[2])
print(f"\nUpdated centroids:")
print(f"Updated m1 (Cluster 1): {new_m1}")
print(f"Updated m2 (Cluster 2): {new_m2}")
# Step 3: Reassign points based on the updated centroids
clusters = {1: [], 2: []}
for label, point in points.items():
    dist_m1 = euclidean_distance(point, new_m1)
    dist_m2 = euclidean_distance(point, new_m2)

    # Assign to the nearest updated centroid
    if dist_m1 < dist_m2:
        clusters[1].append(label)
    else:
        clusters[2].append(label)
# Display final clusters and updated centroids
print(f"\nFinal clusters after reassignment:")
print(f"Cluster 1 (C1): {clusters[1]}")
print(f"Cluster 2 (C2): {clusters[2]}")
# Answer the questions
# 1) Which cluster does P6 belong to?
p6_cluster = "C1" if "P6" in clusters[1] else "C2"
print(f"\n1. P6 belongs to Cluster {p6_cluster}")
# 2) What is the population of the cluster around m2?
population_m2 = len(clusters[2])
print(f"2. The population of the cluster around m2 (Cluster 2) is: {population_m2}")
# 3) What are the updated values of m1 and m2?
print(f"3. The updated value of m1 is: {new_m1}")
print(f"3. The updated value of m2 is: {new_m2}")
```

12

Regression: Simple Linear Dataset: advertising.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Explore whether TV advertising spending can predict the number of sales for the product. Also display the regression results and plot the regression line.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Load the dataset
data = pd.read_csv("advertising.csv")
# Display the first few rows of the dataset
print(data.head())
# Selecting the feature and target variable
X = data[['TV']]  # Independent variable: TV advertising spending
y = data['Sales']  # Dependent variable: Product sales
# Initialize the Linear Regression model
model = LinearRegression()
# Fit the model
model.fit(X, y)
# Predictions
y_pred = model.predict(X)
# Regression Coefficients and Intercept
print("Regression Coefficients (Slope):", model.coef_)
print("Regression Intercept:", model.intercept_)
# Calculate the residual sum of squares (RSS) and coefficient of determination (R^2)
rss = mean_squared_error(y, y_pred) * len(y)
r2 = r2_score(y, y_pred)
print("Residual Sum of Squares (RSS):", rss)
print("Coefficient of Determination (R^2):", r2)
# Plotting the data points and regression line
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual Sales')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.xlabel("TV Advertising Spending")
plt.ylabel("Product Sales")
plt.title("Linear Regression: TV Advertising Spending vs Product Sales")
plt.legend()
plt.show()
```

13

Regression: Simple Linear Dataset: advertising.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Explore whether Radio advertising spending can predict the number of sales for the product. Also display the regression results and plot the regression line.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv("advertising.csv")
# Display the first few rows of the dataset
print(data.head())
# Selecting the feature and target variable
X = data[['Radio']]  # Independent variable: Radio advertising spending
y = data['Sales']    # Dependent variable: Product sales
# Initialize the Linear Regression model
model = LinearRegression()
# Fit the model
model.fit(X, y)
# Make predictions
y_pred = model.predict(X)
# Regression Coefficients and Intercept
print("Regression Coefficient (Slope):", model.coef_[0])
print("Regression Intercept:", model.intercept_)
# Calculate residual sum of squares (RSS) and coefficient of determination (R²)
rss = np.sum((y - y_pred) ** 2)
r2 = r2_score(y, y_pred)
print("Residual Sum of Squares (RSS):", rss)
print("Coefficient of Determination (R²):", r2)
# Plotting the data points and regression line
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual Sales')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.xlabel("Radio Advertising Spending")
plt.ylabel("Product Sales")
plt.title("Linear Regression: Radio Advertising Spending vs Product Sales")
plt.legend()
plt.show()
```

14

Regression: Simple Linear Dataset: advertising.csv Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. Explore whether Newspaper advertising spending can predict the number of sales for the product. Also display the regression results and plot the regression line.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv("advertising.csv")

# Display the first few rows of the dataset to check the data structure
print(data.head())

# Select the feature and target variable
X = data[['Newspaper']]  # Independent variable: Newspaper advertising spending
y = data['Sales']        # Dependent variable: Product sales

# Initialize the Linear Regression model
model = LinearRegression()

# Fit the model
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)

# Display regression coefficients and intercept
print("Regression Coefficient (Slope):", model.coef_[0])
print("Regression Intercept:", model.intercept_)

# Calculate residual sum of squares (RSS) and coefficient of determination (R²)
rss = mean_squared_error(y, y_pred) * len(y)
r2 = r2_score(y, y_pred)
print("Residual Sum of Squares (RSS):", rss)
print("Coefficient of Determination (R²):", r2)

# Plotting the data points and regression line
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual Sales')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.xlabel("Newspaper Advertising Spending")
plt.ylabel("Product Sales")
plt.title("Linear Regression: Newspaper Advertising Spending vs Product Sales")
plt.legend()
plt.show()
```

15

Market Basket Analysis: Apriori Algorithm Dataset: Order1.csv The dataset has 38765 rows of the purchase orders of people from the grocery stores. These orders can be analysed, and association rules can be generated using Market Basket Analysis by algorithms like Apriori Algorithm. Follow following Steps: a. Data Pre-processing b. Generate the list of transactions from the dataset c. Train Apriori on the dataset d. Visualize the list of datasets

```
# Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("Order1.csv")

# Step 1: Data Preprocessing
basket = data.groupby(['Member_number', 'itemDescription']).size().unstack(fill_value=0)
basket = basket.apply(lambda x: x > 0).astype(int)

# Step 2: Apply Apriori Algorithm
frequent_itemsets = apriori(basket, min_support=0.01, use_colnames=True)

# Calculate num_itemsets for compatibility with older versions of mlxtend
num_itemsets = len(frequent_itemsets)

# Step 3: Generate Association Rules
rules = association_rules(frequent_itemsets, num_itemsets=num_itemsets, metric="confidence", min_threshold=0.2)
print(rules)

# Step 4: Visualize the Top Association Rules
top_rules = rules.sort_values(by="confidence", ascending=False).head(10)
plt.figure(figsize=(10, 6))
plt.scatter(top_rules['support'], top_rules['confidence'], s=top_rules['lift']*100, c='orange', alpha=0.6)
plt.xlabel('Support')
plt.ylabel('Confidence')
plt.title('Top 10 Association Rules (Size ~ Lift)')
plt.show()
```

16

Market Basket Analysis: Apriori Algorithm Dataset: Order2.csv This dataset comprises the list of transactions of a retail company over the period of one week. It contains a total of 7501 transaction records where each record consists of the list of items sold in one transaction. Using this record of transactions and items in each transaction, find the association rules between items. There is no header in the dataset and the first row contains the first transaction, so mentioned header = None here while loading dataset. Follow following steps: a. Data Pre-processing b. Generate the list of transactions from the dataset c. Train Apriori algorithm on the dataset d. Visualize the list of rules

```
# Step 1: Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
from mlxtend.preprocessing import TransactionEncoder

# Step 2: Load the dataset without headers
# The dataset contains transaction data without headers, so we load it with header=None
data = pd.read_csv("Order2.csv", header=None)

# Step 3: Data Pre-processing
# Each transaction (row) contains items as columns, so we need to convert it into a list of transactions
transactions = []
for i in range(len(data)):
    # Extract non-NaN values in each row and store them as a list (each list is a transaction)
    transaction = [str(data.values[i, j]) for j in range(len(data.columns)) if pd.notna(data.values[i, j])]
    transactions.append(transaction)

# Step 4: Convert the list of transactions into a one-hot encoded format
# Use TransactionEncoder to perform the one-hot encoding transformation
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
basket = pd.DataFrame(te_ary, columns=te.columns_)

# Step 5: Apply the Apriori algorithm
# Set a minimum support threshold for frequent itemsets (e.g., 1% support in this case)
frequent_itemsets = apriori(basket, min_support=0.01, use_colnames=True)

# Step 6: Generate association rules
# Generate association rules from the frequent itemsets based on a minimum confidence of 0.2
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.2,num_itemsets=5)
# Step 7: Visualize the List of Rules
# Sort the rules by confidence in descending order to get the top rules
top_rules = rules.sort_values(by="confidence", ascending=False).head(10)
# Plotting the top 10 rules
# The size of each point in the scatter plot represents the lift of the rule
plt.figure(figsize=(10, 6))
plt.scatter(top_rules['support'], top_rules['confidence'], s=top_rules['lift']*100, c='blue', alpha=0.6)
plt.xlabel('Support')
plt.ylabel('Confidence')
plt.title('Top 10 Association Rules (Size ~ Lift)')
plt.show()
# Display the top rules
print(top_rules)
```

17

Market Basket Analysis: Apriori Algorithm Dataset: Order3.csv The dataset has 20507 rows and 5 columns of the purchase orders of people from the bakery. These orders can be analysed, and association rules can be generated using Market Basket Analysis by algorithms like Apriori Algorithm. Follow following steps: a. Data Pre-processing b. Generate the list of transactions from the dataset c. Train Apriori algorithm on the dataset d. Visualize the list of rules

```
# Step 1: Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.preprocessing import TransactionEncoder

# Step 2: Load the dataset
data = pd.read_csv('Order3.csv')

# Step 3: Inspect the dataset
print(data.head())
print(data.info())
# Convert the dataset into a list of transactions
transactions = data.groupby('OrderID')['Product'].apply(list).tolist()
encoder = TransactionEncoder()
encoded_data = encoder.fit(transactions).transform(transactions)
encoded_df = pd.DataFrame(encoded_data, columns=encoder.columns_)
frequent_itemsets = apriori(encoded_df, min_support=0.01, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
# Step 8: Inspect and Visualize the Rules
# Show the first few association rules
print(rules.head())
# Visualize the support vs confidence plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=rules, x='support', y='confidence', size='lift', hue='lift', palette='coolwarm', legend=False)
plt.title('Support vs Confidence (with Lift as size and color)')
plt.xlabel('Support')
plt.ylabel('Confidence')
plt.show()
# Step 9: Visualize the Top 10 Rules based on Lift
# Sorting rules by lift value
top_rules = rules.sort_values('lift', ascending=False).head(10)
# Displaying the top 10 rules
plt.figure(figsize=(10, 6))
sns.barplot(x='lift', y=top_rules['rule'], data=top_rules, palette='viridis')
plt.title('Top 10 Association Rules by Lift')
plt.xlabel('Lift')
plt.ylabel('Rules')
plt.show()
# Optionally, you can save the rules to a CSV file
rules.to_csv('association_rules.csv', index=False)
```

18

Classification: Naïve Bayes Dataset: pima-indians-diabetes.csv Use probabilistic approach to implement Classifier model. Evaluate the performance of the model.

```
# Step 1: Import necessary libraries
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
from sklearn.model_selection import cross_val_score

# Step 2: Load the dataset
data = pd.read_csv('pima-indians-diabetes.csv')

# Step 3: Inspect the dataset
print(data.head())
print(data.info())

# Step 4: Data Preprocessing

# In this dataset, we assume the last column is the target variable "Outcome" (1: diabetes, 0: no diabetes)
# Split the dataset into features (X) and target variable (y)
X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']              # Target variable

# Step 5: Split the dataset 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)

# Step 6: Initialize the Naïve Bayes classifier
nb_classifier = GaussianNB()

# Step 7: Train the model
nb_classifier.fit(X_train, y_train)

# Step 8: Make predictions on the test set
y_pred = nb_classifier.predict(X_test)

# Step 9: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Step 10: Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Step 11: Cross-validation to evaluate the model further
cv_scores = cross_val_score(nb_classifier, X, y, cv=10)  # 10-fold cross-validation
print(f'\n10-Fold Cross-Validation Mean Accuracy: {cv_scores.mean():.4f}')
```

19

Classification: Naïve Bayes Dataset: Social_Network_Ads.csv Use probabilistic approach to implement Classifier model. Evaluate the performance of the model.

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Step 2: Load and inspect the dataset
data = pd.read_csv('Social_Network_Ads.csv')
# Inspect the first few rows of the dataset
print(data.head())
# Check for missing values
print(data.isnull().sum())
# Step 3: Define features and target variable
X = data[['Age', 'EstimatedSalary']]  # Example feature columns
y = data['Purchased']  # Target variable
# Step 4: Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# Step 5: Initialize and train the Naïve Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)
# Step 6: Make predictions on the test set
y_pred = model.predict(X_test)
# Step 7: Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)
# Classification Report
report = classification_report(y_test, y_pred)
print('Classification Report:')
print(report)
# Step 8: Visualize the confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

20

Classification: Naïve Bayes Dataset: Social_Network_Ads.csv Use probabilistic approach to implement Classifier model. Evaluate the performance of the model.

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the dataset
data = pd.read_csv('Social_Network_Ads.csv')
# Step 2: Inspect the dataset
print(data.head())  # Show the first few rows
print(data.isnull().sum())  # Check for missing values
# Step 3: Define feature variables and target variable
# Assuming 'Age' and 'EstimatedSalary' are the features and 'Purchased' is the target
X = data[['Age', 'EstimatedSalary']]  # Features (predictors)
y = data['Purchased']  # Target variable
# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# Step 5: Initialize the Naïve Bayes classifier
model = GaussianNB()
# Step 6: Train the model
model.fit(X_train, y_train)
# Step 7: Make predictions on the test set
y_pred = model.predict(X_test)
# Step 8: Evaluate the performance of the model
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)
# Classification Report
report = classification_report(y_test, y_pred)
print('Classification Report:')
print(report)
# Step 9: Visualize the confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

21

Classification: Decision Tree Dataset: pima-indians-diabetes.csv Create & evaluate the decision tree. Test the decision tree for any random sample.

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
from sklearn import tree

# Step 1: Load the dataset
data = pd.read_csv('pima-indians-diabetes.csv')
# Step 2: Inspect the dataset
print(data.head())  # Show the first few rows
print(data.isnull().sum())  # Check for missing values
# Step 3: Define feature variables and target variable
X = data.drop('Outcome', axis=1)  # Features (all columns except 'Outcome')
y = data['Outcome']  # Target variable ('Outcome')
# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# Step 5: Initialize the Decision Tree classifier
model = DecisionTreeClassifier(random_state=42)
# Step 6: Train the model
model.fit(X_train, y_train)
# Step 7: Make predictions on the test set
y_pred = model.predict(X_test)
# Step 8: Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)
# Classification Report
report = classification_report(y_test, y_pred)
print('Classification Report:')
print(report)
# Step 9: Visualize the Decision Tree
plt.figure(figsize=(12, 8))
tree.plot_tree(model, filled=True, feature_names=X.columns, class_names=['No Diabetes', 'Diabetes'],
rounded=True)
plt.title('Decision Tree for Pima Indians Diabetes Dataset')
plt.show()
# Step 10: Test the model with a random sample
# Let's test with a random sample (e.g., [5, 166, 72, 19, 175, 25.8, 0.587, 51])
random_sample = [[5, 166, 72, 19, 175, 25.8, 0.587, 51]]  # A random test sample
prediction = model.predict(random_sample)
print(f'Prediction for the random sample: {"Diabetic" if prediction[0] == 1 else "Non-Diabetic"}')
```

Classify Mobile Price Range using Support Vector Machine for a dataset and compare the accuracy Dataset : test.csv

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC  # Support Vector Machine
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the dataset
data = pd.read_csv('test.csv')  # Make sure the file is in the same directory or provide full path

# Step 2: Inspect the dataset
print(data.head())  # Show the first few rows
print(data.info())  # Check for missing values and data types

# Step 3: Preprocessing the data
# If necessary, handle missing values (if any)
data = data.dropna()  # Drop rows with missing values, or you can use imputation

# Step 4: Split the data into features (X) and target (y)
X = data.drop('price_range', axis=1)  # Assuming 'price_range' is the target variable
y = data['price_range']  # 'price_range' is the target variable

# Step 5: Standardize the data (important for SVM)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 6: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=42)

# Step 7: Initialize and train the SVM model
model = SVC(kernel='linear', random_state=42)  # You can also try 'rbf' or 'poly' kernels
model.fit(X_train, y_train)

# Step 8: Make predictions
y_pred = model.predict(X_test)

# Step 9: Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)

# Classification Report
report = classification_report(y_test, y_pred)
print('Classification Report:')
print(report)

# Step 10: Plot the confusion matrix using Seaborn heatmap
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[0, 1, 2, 3], yticklabels=[0, 1, 2, 3])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Optional Step 11: Compare with another classifier (e.g., Decision Tree or Logistic Regression)
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

# Initialize and train Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
dt_y_pred = dt_model.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_y_pred)

# Initialize and train Logistic Regression model
lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train, y_train)
lr_y_pred = lr_model.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_y_pred)

# Compare the accuracies
print(f'Accuracy of SVM: {accuracy:.4f}')
print(f'Accuracy of Decision Tree: {dt_accuracy:.4f}')
print(f'Accuracy of Logistic Regression: {lr_accuracy:.4f}')
```

Implement Support Vector Machine for a dataset and compare the accuracy Dataset : UniversalBank.csv

```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Step 2: Load the dataset
data = pd.read_csv('UniversalBank (1).csv')
# Step 3: Inspect the data
print(data.head())  # Check the first few rows of the dataset
print(data.info())  # Check the structure of the data
# Step 4: Data Preprocessing
# Remove 'ID' column since it’s not useful for the model
data = data.drop(['ID'], axis=1)
# Step 6: Define features and target variable
X = data.drop('PersonalLoan', axis=1)  # Drop target column for features
y = data['PersonalLoan']  # 'PersonalLoan' is the target variable
# Step 7: Split the dataset 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)
# Step 8: Scale the features (important for SVM performance)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Step 9: Train the SVM model
svm = SVC(kernel='linear')  # You can use 'linear', 'poly', 'rbf', etc.
svm.fit(X_train_scaled, y_train)
# Step 10: Predict on the test data
y_pred = svm.predict(X_test_scaled)
# Step 11: Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
cm = confusion_matrix(y_test, y_pred)
print(cm)
# Step 12: Visualize the Confusion Matrix
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Implement Support Vector Machine for a dataset. Dataset : user_behavior_dataset.csv

```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Step 2: Load the dataset
data = pd.read_csv('user_behavior_dataset.csv')

# Step 3: Inspect the data
print(data.head())  # Check the first few rows of the dataset
print(data.info())  # Check the structure of the data

# Step 4: Data Preprocessing

# Handle missing values if any
# For simplicity, we drop rows with missing values here, but you can fill them using fillna() if necessary
data = data.dropna()

# If the dataset contains categorical variables, we need to encode them using LabelEncoder or OneHotEncoder
# Example: encoding a column 'Category' (if exists)
# from sklearn.preprocessing import LabelEncoder
# label_encoder = LabelEncoder()
# data['Category'] = label_encoder.fit_transform(data['Category'])

# Step 5: Define features and target variable
# Assuming 'target' is the target column, replace 'target' with the actual column name of your dataset.
X = data.drop('target', axis=1)  # Drop the target column for features
y = data['target']  # The target variable

# Step 6: Split the dataset 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)

# Step 7: Scale the features (important for SVM performance)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 8: Train the SVM model
svm = SVC(kernel='linear')  # You can change the kernel to 'rbf', 'poly', etc., if needed
svm.fit(X_train_scaled, y_train)

# Step 9: Predict on the test data
y_pred = svm.predict(X_test_scaled)

# Step 10: Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Step 11: Visualize the Confusion Matrix
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Implement Support Vector Machine for a dataset and compare the accuracy by applying the following kernel functions: i. Linear ii. Polynomial Dataset : bank_transactions_data_2.csv

```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Step 2: Load the dataset
data = pd.read_csv('bank_transactions_data_2.csv')

# Step 3: Inspect the data
print(data.head())  # Check the first few rows of the dataset
print(data.info())  # Check the structure of the data

# Step 4: Data Preprocessing

# Handle missing values if any
# For simplicity, we drop rows with missing values here, but you can fill them using fillna() if necessary
data = data.dropna()

# If the dataset contains categorical variables, we need to encode them using LabelEncoder or OneHotEncoder
# Example: encoding a column 'Category' (if exists)
# from sklearn.preprocessing import LabelEncoder
# label_encoder = LabelEncoder()
# data['Category'] = label_encoder.fit_transform(data['Category'])

# Step 5: Define features and target variable
# Assuming 'target' is the target column, replace 'target' with the actual column name of your dataset.
X = data.drop('target', axis=1)  # Drop the target column for features
y = data['target']  # The target variable

# Step 6: Split the dataset 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)

# Step 7: Scale the features (important for SVM performance)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 8: Train the SVM model with a Linear Kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_scaled, y_train)

# Step 9: Predict on the test data with Linear Kernel
y_pred_linear = svm_linear.predict(X_test_scaled)

# Step 10: Evaluate the Linear Kernel model
accuracy_linear = accuracy_score(y_test, y_pred_linear)
print(f'Linear Kernel Accuracy: {accuracy_linear:.4f}')
print('Linear Kernel Classification Report:')
print(classification_report(y_test, y_pred_linear))
print('Linear Kernel Confusion Matrix:')
cm_linear = confusion_matrix(y_test, y_pred_linear)
print(cm_linear)

# Step 11: Train the SVM model with a Polynomial Kernel
svm_poly = SVC(kernel='poly', degree=3)  # You can change the degree for experimentation
svm_poly.fit(X_train_scaled, y_train)

# Step 12: Predict on the test data with Polynomial Kernel
y_pred_poly = svm_poly.predict(X_test_scaled)

# Step 13: Evaluate the Polynomial Kernel model
accuracy_poly = accuracy_score(y_test, y_pred_poly)
print(f'Polynomial Kernel Accuracy: {accuracy_poly:.4f}')
print('Polynomial Kernel Classification Report:')
print(classification_report(y_test, y_pred_poly))
print('Polynomial Kernel Confusion Matrix:')
cm_poly = confusion_matrix(y_test, y_pred_poly)
print(cm_poly)

# Step 14: Visualize the Confusion Matrices for both models
plt.figure(figsize=(12, 6))

# Linear Kernel Confusion Matrix
plt.subplot(1, 2, 1)
sns.heatmap(cm_linear, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0',
'Class 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Linear Kernel Confusion Matrix')

# Polynomial Kernel Confusion Matrix
plt.subplot(1, 2, 2)
sns.heatmap(cm_poly, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0',
'Class 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Polynomial Kernel Confusion Matrix')

plt.tight_layout()
plt.show()

# Step 15: Compare the accuracy of both models
print(f'Comparison of Accuracies:\nLinear Kernel Accuracy: {accuracy_linear:.4f}\nPolynomial Kernel Accuracy:
{accuracy_poly:.4f}')
```