```python
#naive bayes age, salary, purchased 1
import pandas as pd
import numpy as np
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
data = pd.read_csv('NaiveBayes.csv')
data.head()
X = data[['Age', 'Salary']]
y = data['Purchased']
X_train , X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
nb_classifier = GaussianNB()
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy :{accuracy*100:.2f}")
cm = confusion_matrix( y_test, y_pred)
print("Confusion Matrix\n", cm)
cr = classification_report(y_test, y_pred)
print("Classification Report\n", cr)
```

```python
#naive bayes prima indians diabetes 2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
column_names = [
    'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
    'BMI', 'DiabetesPedigree', 'Age', 'Outcome']
df = pd.read_csv('pima-indians-diabetes.csv', names = column_names , header= None, skiprows = 9)
#removing the unwanted values by changing to numeric values
df = df[pd.to_numeric(df['Pregnancies'], errors='coerce').notnull()]
df.reset_index(drop=True, inplace=True)
print(df.head())
# Features (X) - All columns except the last one (Outcome)
X = df.iloc[:, :-1]
# Target (y) - The last column (Outcome)
y = df.iloc[:, -1]
# Check the first few rows of features and target
print(X.head())
print(y.head())
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.4f}")
```

```python
#social media naive bayes 3 4
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
```

```
import matplotlib.pyplot as plt
df = pd.read_csv('data/Social_Network_Ads.csv')
print(df.head())
df = df.drop(columns=['User ID'])
df['Gender'] = LabelEncoder().fit_transform(df['Gender'])
X = df[['Gender', 'Age', 'EstimatedSalary']]  # Features (Gender, Age, EstimatedSalary)
y = df['Purchased']  # Target variable (Purchased)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

#diabetes regression 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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
df = pd.read_csv('data/diabetes.csv')
df = df.dropna()
X = df[['Glucose']]  # Use 'Glucose' as the predictor feature
y = df['Outcome']    # 'Outcome' is the target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Coefficient (Slope):", model.coef_)
print("Intercept:", model.intercept_)
rss = np.sum((y_test - y_pred) ** 2)
print("Residual Sum of Squares (RSS):", rss)
r2_score = model.score(X_test, y_test)
print("Coefficient of Determination (R²):", r2_score)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
accuracy = r2_score * 100  # In percentage terms
print("Accuracy (R² score in percentage):", accuracy)
# Plot the regression line
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.title('Simple Linear Regression')
plt.xlabel('Glucose')
plt.ylabel('Outcome')
plt.legend()
plt.show()
```

#sat vs gpa regression 2

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
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
df = pd.read_csv('data/1.01. Simple linear regression.csv')
X = df[['SAT']]  # Independent variable (SAT score)
y = df['GPA']   # Dependent variable (GPA)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Coefficient (Slope):", model.coef_)
print("Intercept:", model.intercept_)
# Calculate Residual Sum of Squares (RSS)
rss = np.sum((y_test - y_pred) ** 2)
print("Residual Sum of Squares (RSS):", rss)
# Coefficient of Determination (R²)
r2 = model.score(X_test, y_test)
print("Coefficient of Determination (R²):", r2)
# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)
# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)
# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
# Plot the regression line
plt.scatter(X, y, color='blue', label='Data Points')  # Scatter plot of the data
plt.plot(X, model.predict(X), color='red', label='Regression Line')  # Plot the regression line
plt.title('Linear Regression: SAT vs GPA')
plt.xlabel('SAT Score')
plt.ylabel('GPA')
plt.legend()
plt.show()
```

#advertising regression 345

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
df = pd.read_csv('data/advertising.csv')
y = df['Sales']
def perform_regression(feature_name):
   # Define the independent variable (X)
   X = df[[feature_name]]  # We are using only one feature (TV, Radio, or Newspaper)
   # Split the data into training and testing sets (80-20 split)
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
   # Initialize the Linear Regression model
   model = LinearRegression()
   # Train the model on the training data
   model.fit(X_train, y_train)
   # Make predictions on the test data
   y_pred = model.predict(X_test)
   # Display the coefficients
   print(f"\n{feature_name} Model Coefficients:")
   print("Coefficient (Slope):", model.coef_)
   print("Intercept:", model.intercept_)
   # Model Evaluation Metrics
   print(f"\nModel Evaluation for {feature_name}:")
   # Coefficient of Determination (R²)
   r2 = r2_score(y_test, y_pred)
```

```
    print("Coefficient of Determination (R²):", r2)
    # Mean Absolute Error (MAE)
    mae = mean_absolute_error(y_test, y_pred)
    print("Mean Absolute Error (MAE):", mae)
    # Mean Squared Error (MSE)
    mse = mean_squared_error(y_test, y_pred)
    print("Mean Squared Error (MSE):", mse)
    # Root Mean Squared Error (RMSE)
    rmse = np.sqrt(mse)
    print("Root Mean Squared Error (RMSE):", rmse)
    # Plot the regression line
    plt.figure(figsize=(4, 4))
    plt.scatter(X, y, color='blue', label='Data Points')  # Scatter plot of the data
    plt.plot(X, model.predict(X), color='red', label=f'Regression Line ({feature_name})')  # Plot the regression line
    plt.title(f'Linear Regression: {feature_name} vs Sales')
    plt.xlabel(feature_name)
    plt.ylabel('Sales')
    plt.legend()
    plt.show()
perform_regression('TV')
perform_regression('Radio')
perform_regression('Newspaper')
```

#cities hierarchical clustering 1

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import fcluster
df = pd.read_csv('data/cities_r2.csv')
# We are interested in the 'effective_literacy_rate_total' column for clustering
df_literacy = df[['effective_literacy_rate_total']]
print(df_literacy.isnull().sum())
scaler = StandardScaler()
df_literacy_scaled = scaler.fit_transform(df_literacy)
Z = linkage(df_literacy_scaled, method='ward')
# Step 4: Plot the Dendrogram to decide the number of clusters
plt.figure(figsize=(5, 5))
dendrogram(Z)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Cities')
plt.ylabel('Euclidean Distance')
plt.show()
# Fit the model with 3 clusters
hc = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')
# Add the cluster labels to the original dataframe
df['Cluster'] = hc.fit_predict(df_literacy_scaled)
# Step 6: Visualizing the clusters
# Here we plot the cities based on their effective literacy rate and color them by clusters
sns.scatterplot(x=df['effective_literacy_rate_total'],
                y=np.zeros_like(df['effective_literacy_rate_total']),
                hue=df['Cluster'],
                palette='viridis',
                s=100, alpha=0.7)
plt.title('Hierarchical Clustering - Literacy Rate')
plt.xlabel('Effective Literacy Rate Total')
plt.ylabel('Cluster')
plt.show()
print(df[['name_of_city', 'effective_literacy_rate_total', 'Cluster']])
```

#hitters hierarchical clustering 2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
import seaborn as sns
df = pd.read_csv('data/hitters.csv')
df_cruns = df[['CRuns']]
scaler = StandardScaler()
df_cruns_scaled = scaler.fit_transform(df_cruns)
Z = linkage(df_cruns_scaled, method='ward')
#dendogram
plt.figure(figsize=(4,4))
dendrogram(Z)
plt.title('Dendrogram for Hierarchical Clustering (CRuns)')
plt.xlabel('Players')
plt.ylabel('Euclidean Distance')
plt.show()
hc = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')
# Assign cluster labels to the dataframe
df['Cluster'] = hc.fit_predict(df_cruns_scaled)
# Plot the 'CRuns' values and color them based on their cluster
plt.figure(figsize=(4,4))
sns.scatterplot(x=df['CRuns'],
        y=np.zeros_like(df['CRuns']),
        hue=df['Cluster'],
        palette='viridis',
        s=50, alpha=0.7)
plt.title('Hierarchical Clustering - CRuns')
plt.xlabel('CRuns (Clustered)')
plt.ylabel('Cluster')
plt.show()
#cluster count
print(df[['AtBat', 'Hits', 'HmRun', 'CRuns', 'Cluster']])
```

#startups hierarchical 3

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('data/50_Startups.csv')
abel_encoder = LabelEncoder()
df['STATE'] = label_encoder.fit_transform(df['STATE'])
# Select the numerical columns for scaling
numerical_columns = ['RND', 'ADMIN', 'MKT', 'PROFIT']
scaler = StandardScaler()
# Scale the data
df_scaled = df[numerical_columns]
df_scaled = scaler.fit_transform(df_scaled)
Z = linkage(df_scaled, method='ward')
# Step 4: Plot the Dendrogram to decide the number of clusters
plt.figure(figsize=(4,4))
dendrogram(Z)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Startups')
plt.ylabel('Euclidean Distance')
plt.show()
clusters = fcluster(Z, t=3, criterion='maxclust')
df['Cluster'] = clusters
print(df[['RND', 'ADMIN', 'MKT', 'PROFIT', 'Cluster']].head())
plt.figure(figsize=(5,5))
sns.scatterplot(x=df['RND'], y=df['PROFIT'], hue=df['Cluster'], palette='viridis', s=50, alpha=0.7)
```

```python
plt.title('Hierarchical Clustering - RND vs PROFIT')
plt.xlabel('RND')
plt.ylabel('PROFIT')
plt.show()
```

#loan eligibility decision tree 1

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
from sklearn import tree
df = pd.read_csv('data/madfhantr.csv')
print(df.isnull().sum())
df['LoanAmount'].fillna(df['LoanAmount'].median(), inplace=True)
df.dropna(subset=['Loan_Status'], inplace=True)
df['Dependents'] = df['Dependents'].replace('3+', 3)
# Handle missing values in 'Dependents' column
df['Dependents'] = df['Dependents'].fillna(0)  # Replace NaN with 0 or another reasonable value
# Convert 'Dependents' column to integer
df['Dependents'] = df['Dependents'].astype(int)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].median(), inplace=True)
df.dropna(subset=['Loan_Amount_Term'], inplace=True)
df['Credit_History'].fillna(df['Credit_History'].median(), inplace=True)
df.dropna(subset=['Credit_History'], inplace=True)
for column in ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area', 'Loan_Status']:
    df[column] = LabelEncoder().fit_transform(df[column])
X = df[['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome',
        'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area']]
y = df['Loan_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train_scaled, y_train)
y_pred = dt_classifier.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy Score of the Decision Tree Classifier:", accuracy)
# Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(9,9))
tree.plot_tree(dt_classifier, filled=True, feature_names=X.columns, class_names=['N', 'Y'], rounded=True, proportion=False,
precision=2)
plt.title("Decision Tree - Loan Eligibility")
plt.show()
```

#diabetes decision tree 2

```python
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
column_names = [
    'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
    'BMI', 'DiabetesPedigree', 'Age', 'Outcome']
df = pd.read_csv('data/pima-indians-diabetes.csv', names = column_names , header= None, skiprows = 9)
#removing the unwanted values by changing to numeric values
df = df[pd.to_numeric(df['Pregnancies'], errors='coerce').notnull()]
```

```
df.reset_index(drop=True, inplace=True)
print(df.head())
X = df.drop('Outcome', axis=1)  # Features (all columns except 'Outcome')
y = df['Outcome']  # Target variable (Outcome)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
y_pred = dt_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
# Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(12, 8))
tree.plot_tree(dt_classifier, filled=True, feature_names=X.columns, class_names=['No Diabetes', 'Diabetes'], rounded=True)
plt.title("Decision Tree - Diabetes Prediction")
plt.show()
#random sample test
random_sample = X_test.sample(1)  # Pick a random sample from the test set
random_sample_prediction = dt_classifier.predict(random_sample)  # Predict the outcome for this sample
# Print the random sample and its prediction
print("\nRandom Sample Prediction:")
print(random_sample)
print(f"Prediction (1 = Diabetes, 0 = No Diabetes): {random_sample_prediction[0]}")
```

#cities r2 kmeans 1

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder, StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('data/cities_r2.csv')
print(df.isnull().sum())  # Check for missing values
# For simplicity, let's drop rows with missing values. Alternatively, we can fill with mean or mode
df = df.dropna()
abel_encoder = LabelEncoder()
# Encode categorical columns
df['state_name'] = label_encoder.fit_transform(df['state_name'])
df['location'] = label_encoder.fit_transform(df['location'])
X = df[['total_graduates']]  # We're focusing on total_graduates for clustering
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)
plt.figure(figsize=(5,5))
sns.scatterplot(x=df.index, y='total_graduates', hue='Cluster', palette='viridis', data=df, s=100)
plt.title('K-Means Clustering based on Total Graduates')
plt.xlabel('City Index')
plt.ylabel('Total Graduates')
plt.legend(title='Cluster')
plt.show()
print(df[['name_of_city', 'total_graduates', 'Cluster']].sort_values(by='Cluster'))
```

#social media advertise kmeans 2

```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder, StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('data/Social_Network_Ads.csv')
```

```
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
X = df[['Age', 'EstimatedSalary']]  # We can include 'Age' as well if desired
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)
plt.figure(figsize=(5,5))
sns.scatterplot(x='Age', y='EstimatedSalary', hue='Cluster', palette='viridis', data=df, s=100)
plt.title('K-Means Clustering based on Age and Estimated Salary')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend(title='Cluster')
plt.show()
print(df[['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Cluster']].sort_values(by='Cluster'))
```

#k-means clustering same with 'effective_literacy_rate_total' instead of total graduates 3

with 8 initial centroids kmeans 4

```
import numpy as np
import matplotlib.pyplot as plt
points = np.array([
    [0.1, 0.6],  # P1
    [0.15, 0.71],  # P2
    [0.08, 0.9],  # P3
    [0.16, 0.85],  # P4
    [0.2, 0.3],  # P5
    [0.25, 0.5],  # P6
    [0.24, 0.1],  # P7
    [0.3, 0.2]   # P8
])
m1 = np.array([0.1, 0.6])  # P1
m2 = np.array([0.3, 0.2])  # P8
def euclidean_distance(p1, p2):
    return np.sqrt(np.sum((p1 - p2) ** 2))
def assign_clusters(points, m1, m2):
    cluster_1 = []
    cluster_2 = []
    for point in points:
        d1 = euclidean_distance(point, m1)
        d2 = euclidean_distance(point, m2)
        if d1 < d2:
            cluster_1.append(point)
        else:
            cluster_2.append(point)
    return np.array(cluster_1), np.array(cluster_2)
def calculate_new_centroids(cluster_1, cluster_2):
    new_m1 = np.mean(cluster_1, axis=0) if len(cluster_1) > 0 else m1
    new_m2 = np.mean(cluster_2, axis=0) if len(cluster_2) > 0 else m2
    return new_m1, new_m2
def k_means_clustering(points, m1, m2):
    # Step 1: Assign points to the nearest centroid
    cluster_1, cluster_2 = assign_clusters(points, m1, m2)
    # Step 2: Calculate new centroids
    new_m1, new_m2 = calculate_new_centroids(cluster_1, cluster_2)
    return cluster_1, cluster_2, new_m1, new_m2
cluster_1, cluster_2, new_m1, new_m2 = k_means_clustering(points, m1, m2)
# Question 1: Which cluster does P6 belong to?
p6 = np.array([0.25, 0.5])
d1 = euclidean_distance(p6, m1)
d2 = euclidean_distance(p6, m2)
p6_cluster = 1 if d1 < d2 else 2
# Question 2: What is the population of the cluster around m2?
population_cluster_2 = len(cluster_2)
# Question 3: What are the updated values of m1 and m2?
```

```
updated_m1 = new_m1
updated_m2 = new_m2
# Output the results
print(f"Cluster 1 (C1):\n{cluster_1}")
print(f"Cluster 2 (C2):\n{cluster_2}")
print(f"Updated m1: {updated_m1}")
print(f"Updated m2: {updated_m2}")
# Answer to the questions:
print(f"\nAnswer to the questions:")
print(f"1. P6 belongs to Cluster {p6_cluster}")
print(f"2. The population of Cluster C2 (around m2) is: {population_cluster_2}")
print(f"3. The updated centroids are m1: {updated_m1} and m2: {updated_m2}")
plt.figure(figsize=(8, 6))
# Plot Cluster 1 in red
plt.scatter(cluster_1[:, 0], cluster_1[:, 1], color='red', label='Cluster 1 (C1)', s=100)
# Plot Cluster 2 in blue
plt.scatter(cluster_2[:, 0], cluster_2[:, 1], color='blue', label='Cluster 2 (C2)', s=100)
# Plot the centroids (m1 and m2)
plt.scatter(m1[0], m1[1], color='black', marker='X', s=200, label='Centroid m1 (C1)', linewidth=3)
plt.scatter(m2[0], m2[1], color='green', marker='X', s=200, label='Centroid m2 (C2)', linewidth=3)
# Plot the updated centroids
plt.scatter(updated_m1[0], updated_m1[1], color='orange', marker='X', s=200, label='Updated Centroid m1', linewidth=3)
plt.scatter(updated_m2[0], updated_m2[1], color='purple', marker='X', s=200, label='Updated Centroid m2', linewidth=3)
# Labeling the points and adding legend
plt.title("K-Means Clustering (1 Iteration)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.show()
```

mobile prices svm 1

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
df = pd.read_csv('data/test.csv')
# Drop the 'id' column as it's just an identifier
#df = df.drop(columns=['id'])
# Creating a target variable based on 'ram' (you can modify this to suit your needs)
def classify_price_range(row):
    if row['ram'] < 1000:
        return 0  # Low
    elif row['ram'] < 2000:
        return 1  # Medium
    else:
        return 2  # High
# Apply the classification function to create the 'price_range' column
df['price_range'] = df.apply(classify_price_range, axis=1)
X = df[['ram', 'battery_power']]
y = df['price_range']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm = SVC(kernel='linear')  # Using linear kernel for simplicity
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the SVM model: {accuracy * 100:.2f}%')
```

#universal bank SVM 2

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
df = pd.read_csv('data/UniversalBank.csv')
X = df.drop(columns=['ID', 'Personal Loan'])  # Features
y = df['Personal Loan']  # Target (Personal Loan)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Step 5: Train and test the SVM model (using linear kernel for simplicity)
svm = SVC(kernel='linear')
svm.fit(X_train_scaled, y_train)
svm_pred = svm.predict(X_test_scaled)
svm_accuracy = accuracy_score(y_test, svm_pred)
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)
log_reg_pred = log_reg.predict(X_test_scaled)
log_reg_accuracy = accuracy_score(y_test, log_reg_pred)
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)
print(f'Accuracy of SVM: {svm_accuracy * 100:.2f}%')
print(f'Accuracy of Logistic Regression: {log_reg_accuracy * 100:.2f}%')
print(f'Accuracy of Random Forest: {rf_accuracy * 100:.2f}%')

# Step 9: Plot the comparison of model accuracies
models = ['SVM', 'Logistic Regression', 'Random Forest']
accuracies = [svm_accuracy, log_reg_accuracy, rf_accuracy]
plt.bar(models, accuracies, color=['blue', 'green', 'orange'])
plt.ylabel('Accuracy')
plt.title('Model Comparison')
plt.show()
```

#user behaviour svm 3

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
df = pd.read_csv('data/user_behavior_dataset.csv')
le = LabelEncoder()

df['Device Model'] = le.fit_transform(df['Device Model'])
df['Operating System'] = le.fit_transform(df['Operating System'])
df['Gender'] = le.fit_transform(df['Gender'])
X = df.drop(columns=['User ID', 'User Behavior Class'])
y = df['User Behavior Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm = SVC(kernel='linear')
svm.fit(X_train_scaled, y_train)
svm_pred = svm.predict(X_test_scaled)
```

```
svm_accuracy = accuracy_score(y_test, svm_pred)
print(f'Accuracy of SVM model: {svm_accuracy * 100:.2f}%')
plt.bar(['SVM'], [svm_accuracy], color='blue')
plt.ylabel('Accuracy')
plt.title('SVM Model Accuracy')
plt.show()
```

#user behavior svm 4

```
# Step 1: Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
df = pd.read_csv('data/bank_transactions_data_2.csv')
df.dropna(inplace=True)
label_encoder = LabelEncoder()
# Columns that need encoding
categorical_columns = ['TransactionType', 'Location', 'DeviceID', 'MerchantID', 'Channel', 'CustomerOccupation']
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])
features = ['TransactionAmount', 'CustomerAge', 'TransactionDuration', 'LoginAttempts', 'AccountBalance']
X = df[features]
# Target variable: let's predict 'TransactionType' (Debit=0, Credit=1)
y = df['TransactionType']
#Step 3: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 4: Standardize the features (important for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_scaled, y_train)
y_pred_linear = svm_linear.predict(X_test_scaled)
accuracy_linear = accuracy_score(y_test, y_pred_linear)
print("SVM with Linear Kernel Accuracy: ", accuracy_linear)
print("Classification Report for Linear Kernel:")
print(classification_report(y_test, y_pred_linear))
svm_poly = SVC(kernel='poly', degree=3)
svm_poly.fit(X_train_scaled, y_train)
# Make predictions for Polynomial Kernel
y_pred_poly = svm_poly.predict(X_test_scaled)
accuracy_poly = accuracy_score(y_test, y_pred_poly)
print("SVM with Polynomial Kernel Accuracy: ", accuracy_poly)
print("Classification Report for Polynomial Kernel:")
print(classification_report(y_test, y_pred_poly))
if accuracy_linear > accuracy_poly:
    print("The Linear Kernel model performs better.")
else:
    print("The Polynomial Kernel model performs better.")
```

#apriori market basket analysis 1

```
# Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
# Step 1: Load the dataset
data = pd.read_csv("data/Order1.csv")
# Step 2: Data preprocessing
# Renaming columns for clarity (optional)
data.rename(columns={"Member_number": "TransactionID", "itemDescription": "Item"}, inplace=True)
# Step 3: Group transactions by TransactionID
```

```
# Each transaction will be a list of items
transactions = data.groupby("TransactionID")["Item"].apply(list)
# Step 4: Create a one-hot encoded DataFrame for Apriori
# Flatten the transactions to create a binary matrix
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_data = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_data, columns=te.columns_)
# Step 5: Apply the Apriori algorithm
frequent_itemsets = apriori(df_encoded, min_support=0.09, use_colnames=True)
print("Frequent Itemsets:")
print(frequent_itemsets)
# Step 6: Generate association rules
from mlxtend.frequent_patterns import association_rules
# Generate association rules from frequent itemsets
rules = association_rules(frequent_itemsets,num_itemsets=2, metric="lift", min_threshold=1.0)
# Display association rules
print("Association Rules:")
print(rules)
plt.figure(figsize=(10, 6))
sns.barplot(x='support', y='antecedents', data=rules)
plt.title('Association Rules Support')
plt.xlabel('Support')
plt.ylabel('Antecedents')
plt.show()
# Visualization 2: Plot the lift of the association rules
plt.figure(figsize=(10, 6))
sns.barplot(x='lift', y='antecedents', data=rules)
plt.title('Association Rules Lift')
plt.xlabel('Lift')
plt.ylabel('Antecedents')
plt.show()
# Visualization 3: Scatter plot of support vs. lift for the association rules
plt.figure(figsize=(10, 6))
sns.scatterplot(x='support', y='lift', data=rules)
plt.title('Support vs. Lift for Association Rules')
plt.xlabel('Support')
plt.ylabel('Lift')
plt.show()
```

#apriori order 2

```
#data shaping
transactions = []
for i in range(data.shape[0]):  # Iterate through rows
    transaction = [str(data.iloc[i, j]) for j in range(data.shape[1]) if pd.notnull(data.iloc[i, j])]
    transactions.append(transaction)
```