3. For the Market-Basket dataset, apply Apriori algorithm and identify the best rules based on support and confidence.

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

df=pd.read_csv("basket.csv")
all_items=list(item for sublist in df.values.tolist() for item in sublist if pd.notna(item))
unique_items = list(set(all_items))[:200]
df_subset = df.head(200)
encoded_df=pd.DataFrame(0,index=range(len(df)),columns=unique_items)
for index,transaction in df_subset.iterrows():
    for item in transaction.dropna():
        encoded_df.loc[index,item]=1
min_support=0.3
min_confident=0.7
# Step 4: Apply the Apriori algorithm to find frequent itemsets
frequent_items = apriori(encoded_df, min_support=min_support, use_colnames=True)

# Step 5: Generate association rules based on the frequent itemsets
rules = association_rules(frequent_items, metric="confidence",
min_threshold=min_confident)

# Output the frequent itemsets and association rules
print("Frequent Itemsets:")
print(frequent_items)

print("\nAssociation Rules:")
print(rules)
```

sample dataset:

bread	milk	cookie	eggs	
bread	milk	cookie	soup	
bread	milk	cookie		
turkey	eggs			
eggs	cookies			
milk	diaper	bread		
bread	diaper			
bread	milk	cookie	avocado	
bread	milk	cookie		
bread	milk	cookie	eggs	

Output

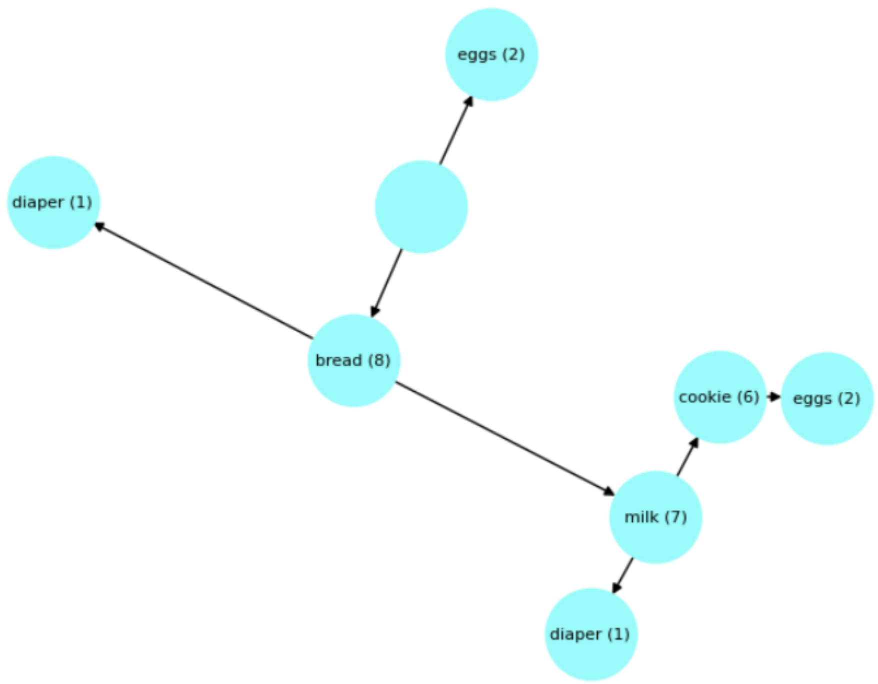
```
Frequent Itemsets:
    support              itemsets
0  0.500000               (bread)
1  0.363636              (cookie)
2  0.454545                (milk)
3  0.409091                (eggs)
4  0.363636       (bread, cookie)
5  0.409091         (bread, milk)
6  0.363636        (milk, cookie)
7  0.363636  (bread, milk, cookie)

Association Rules:
        antecedents      consequents  antecedent support  consequent support
0           (bread)         (cookie)            0.500000            0.363636
1          (cookie)          (bread)            0.363636            0.500000
2           (bread)           (milk)            0.500000            0.454545
3            (milk)          (bread)            0.454545            0.500000
4            (milk)         (cookie)            0.454545            0.363636
5          (cookie)           (milk)            0.363636            0.454545
6     (bread, milk)         (cookie)            0.409091            0.363636
7   (bread, cookie)           (milk)            0.363636            0.454545
8    (milk, cookie)          (bread)            0.363636            0.500000
9           (bread)   (milk, cookie)            0.500000            0.363636
10           (milk)  (bread, cookie)            0.454545            0.363636
11         (cookie)    (bread, milk)            0.363636            0.409091
```

4. For the data set given in Q3, apply FP-tree algorithm, show the tree construction and identify the best rules based on support and confidence.

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth

# Example dataset
data = [
    ['bread', 'milk', 'cookie', 'eggs'],
    ['bread', 'milk', 'cookie', 'soup'],
    ['bread', 'milk', 'cookie'],
    ['turkey', 'eggs'],
    ['eggs', 'cookies'],
    ['milk', 'diaper', 'bread'],
    ['bread', 'diaper'],
    ['bread', 'milk', 'cookie', 'avocado'],
    ['bread', 'milk', 'cookie'],
    ['bread', 'milk', 'cookie', 'eggs']
]

# Create a DataFrame with one-hot encoding
df = pd.DataFrame(data)
df = df.stack().reset_index().pivot_table(index='level_0', columns=0, aggfunc=lambda x: 1,
fill_value=0)

# Apply the FP-growth algorithm
frequent_itemsets = fpgrowth(df, min_support=0.2, use_colnames=True)

# Display frequent itemsets
print(frequent_itemsets)
```

	support	itemsets
0	0.8	((level_1, bread))
1	0.7	((level_1, milk))
2	0.6	((level_1, cookie))
3	0.4	((level_1, eggs))
4	0.2	((level_1, diaper))
5	0.7	((level_1, bread), (level_1, milk))
6	0.6	((level_1, cookie), (level_1, milk))
7	0.6	((level_1, cookie), (level_1, bread))
8	0.6	((level_1, cookie), (level_1, bread), (level_1...
9	0.2	((level_1, cookie), (level_1, eggs))
10	0.2	((level_1, eggs), (level_1, milk))
11	0.2	((level_1, eggs), (level_1, bread))
12	0.2	((level_1, cookie), (level_1, eggs), (level_1,...
13	0.2	((level_1, cookie), (level_1, eggs), (level_1,...
14	0.2	((level_1, eggs), (level_1, bread), (level_1, ...
15	0.2	((level_1, cookie), (level_1, eggs), (level_1,...
16	0.2	((level_1, bread), (level_1, diaper))

FP-tree structure:



5. For the Mall-Customer data set, implement K-means clustering algorithm and visualize the clusters.

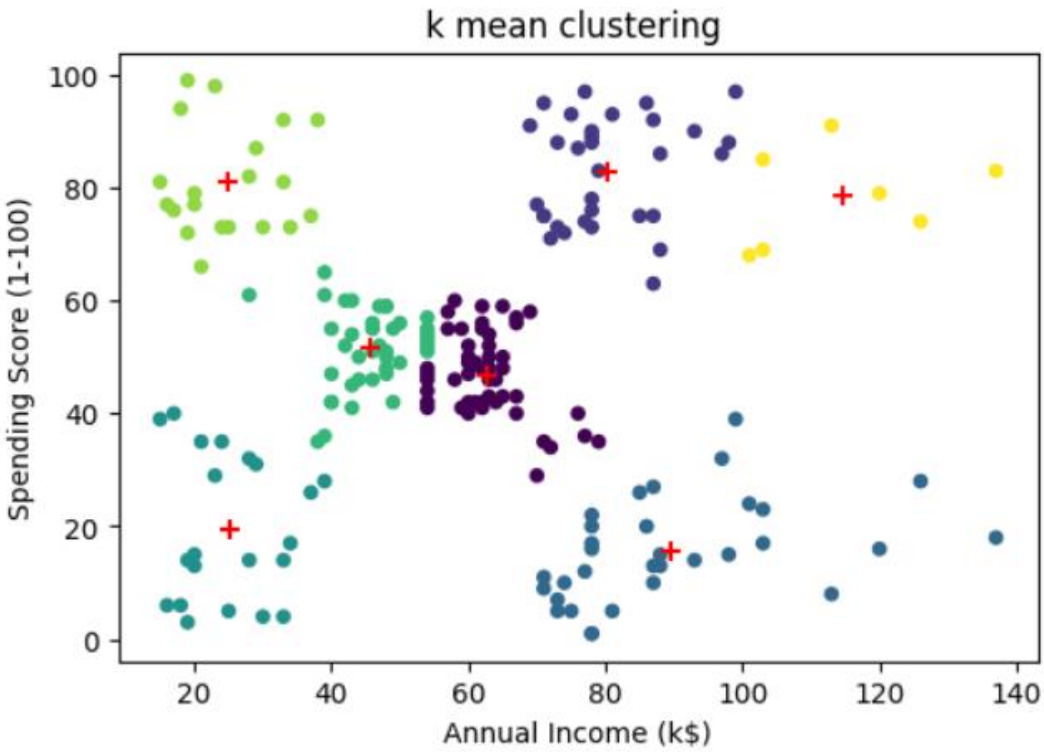
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
data=pd.read_csv("Mall_Customers.csv")
#data.sample(5)
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
22	23	Female	46	25	5
100	101	Female	23	62	41
36	37	Female	42	34	17
53	54	Male	59	43	60
191	192	Female	32	103	69

```
X=data[['Annual Income (k$)','Spending Score (1-100)']]
kmean=KMeans(n_clusters=7,random_state=0)
y_kmeans=kmean.fit_predict(X)
data['Cluster']=y_kmeans
#data.head(5)
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	Male	19	15	39	3
1	2	Male	21	15	81	5
2	3	Female	20	16	6	3
3	4	Female	23	16	77	5
4	5	Female	31	17	40	3

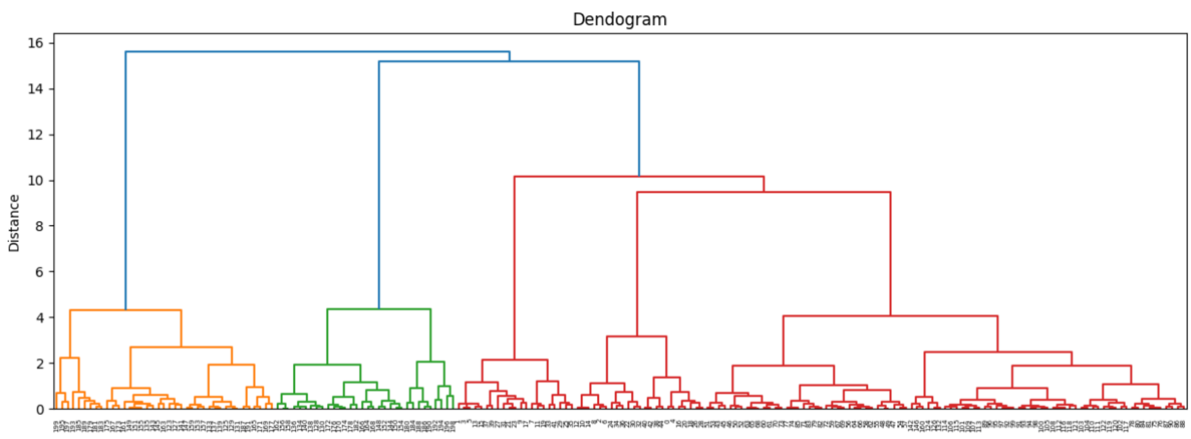
```
plt.figure(figsize=(8,6))
plt.scatter(X.iloc[:,0],X.iloc[:,1],c=y_kmeans,s=50)
centroids = kmean.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='+', label='Centroids')
plt.title('k mean clustering ')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.show()
```


k mean clustering
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
20
40
60
80
100
120
140

6. For the Groceries dataset implement the Agglomerative clustering algorithm and visualize the clusters.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram,linkage
featues=data[['Annual Income (k$)','Spending Score (1-100)']]
#scalling the feature
scaler=StandardScaler()
scaled_feature=scaler.fit_transform(featues)
# Apply agglomerative clustering
agg_clustering=AgglomerativeClustering(n_clusters=5)
data['Cluster']=agg_clustering.fit_predict(scaled_feature)

linkage_matrix=linkage(scaled_feature,method='ward')
plt.figure(figsize=(20,17))
dendrogram(linkage_matrix)
plt.title('Dendogram')
plt.xlabel('levels')
plt.ylabel('Distance')
plt.show()
```

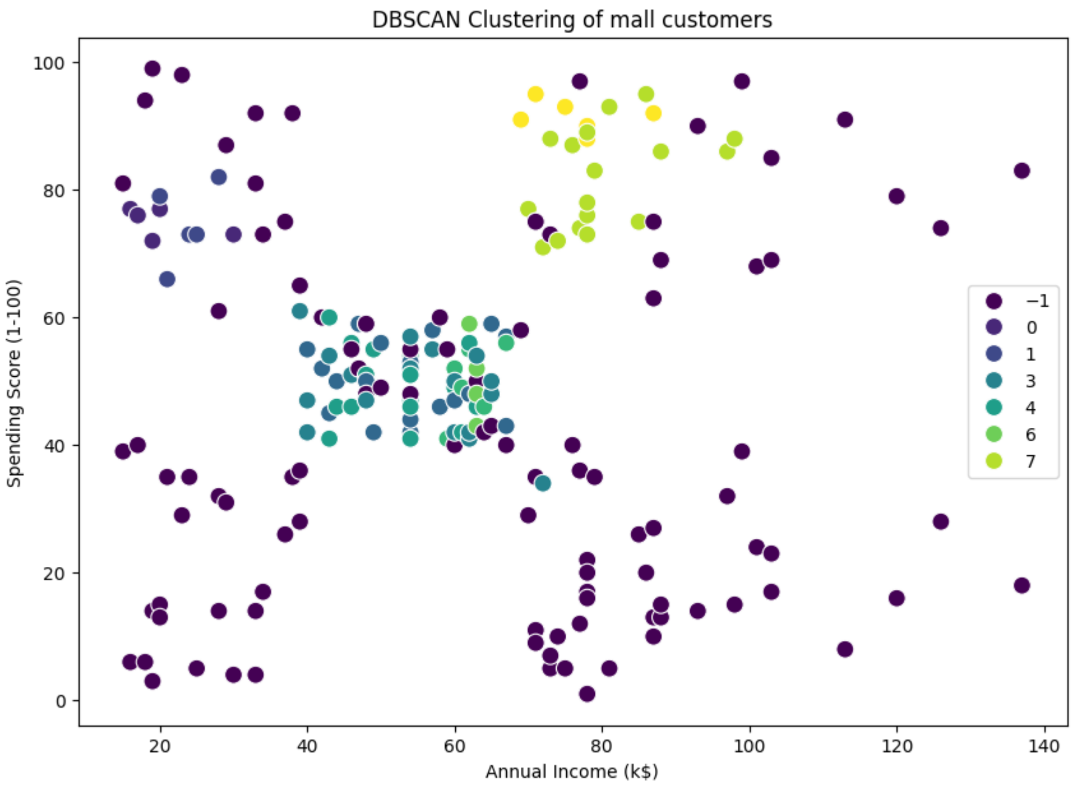
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
5	6	Female	22	17	76	3
11	12	Female	35	19	99	3
150	151	Male	43	78	17	0
67	68	Female	68	48	48	2
155	156	Female	27	78	89	1



7. For the Mall_Customers implement DBScan clustering algorithm and visualize the clusters.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
data=pd.read_csv('Mall_Customers.csv')

data['Gender']=data['Gender'].map({'Male':0,'Female':1})
data.sample(5)

features=data[['Gender','Age','Annual Income (k$)','Spending Score (1-100)']]

#standardizing the features to enure all variables are in same scallable
scalar=StandardScaler()
scaled_features=scalar.fit_transform(features)

#Applay db sacan clustering
dbscan=DBSCAN(eps=0.5,min_samples=5)
clusters=dbscan.fit_predict(scaled_features)

data['Cluster']=clusters
marks=['o','s','D','^','P','*']
plt.figure(figsize=(10,7))
sns.scatterplot(data=data,x="Annual Income (k$)",y="Spending Score (1-
100)",hue="Cluster",palette='viridis',s=100)
plt.title("DBSCAN Clustering of mall customers")
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

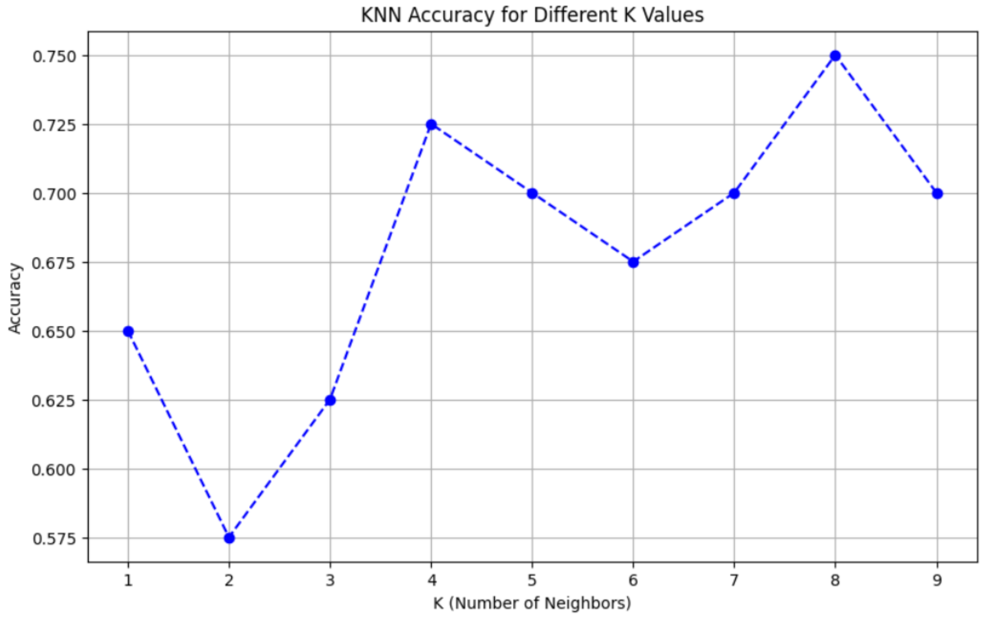
```
Cluster
-1    105
 3     18
 2     18
 7     17
 4     15
 5      7
 8      6
 0      5
 1      5
 6      4
Name: count, dtype: int64
```



8. Implement KNN Classification algorithm on the Mall Customers. Analyse the model using different K values and display the performance of the model.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,accuracy_score

data=pd.read_csv('Mall_Customers.csv')
data['Gender']=data['Gender'].map({'Male':0,'Female':1})

# Define the target variable (e.g., categorize Spending Score into low (0) and high (1))
# Assuming scores <= 50 are "low spenders" and > 50 are "high spenders"
data['Spending_Category'] = data['Spending Score (1-100)'].apply(lambda x: 1 if x > 50 else
0)

X=data[['Gender','Age','Annual Income (k$)']]
y=data['Spending_Category']

#Standardize the feature
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)

X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.2,random_state=42)

# Define the range of K values to tX_test
k_values=range(1,10)
accuracy_scores=[]

for k in k_values:
    # Initialize the KNN classifire
    knn=KNeighborsClassifier(n_neighbors=k)
    # Fit the data model on training data
    knn.fit(X_train,y_train)
    # predict on the testing data
    y_pred=knn.predict(X_test)
    accuracy=accuracy_score(y_test,y_pred)
    accuracy_scores.append(accuracy)
    print(f"\nK={k} Classification Report: ")
    print(classification_report(y_test,y_pred))

print("\nAccuracy scores for different K values:")
# Display the accuracy scores for different K values
for k in k_values:
    print(f"K={k}: {accuracy:.4f}")
```

```
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy_scores, marker='o', linestyle='--', color='b')
plt.title('KNN Accuracy for Different K Values')
plt.xlabel('K (Number of Neighbors)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```




```
# Example custom input for a new customer: Female, 30 years old, Annual Income 70k
custom_data = pd.DataFrame([[1, 20, 20]],columns= ['Gender', 'Age', 'Annual Income (k$)']
)  # Female, Age 30, Annual Income 70k

# Scale the custom data
custom_data_scaled = scaler.transform(custom_data)
# Predict the category for the custom data
predicted_category = knn.predict(custom_data_scaled)

# Output the predicted category
if predicted_category[0] == 1:
   print("Predicted Spending Category: High Spender")
else:
   print("Predicted Spending Category: Low Spender")
```

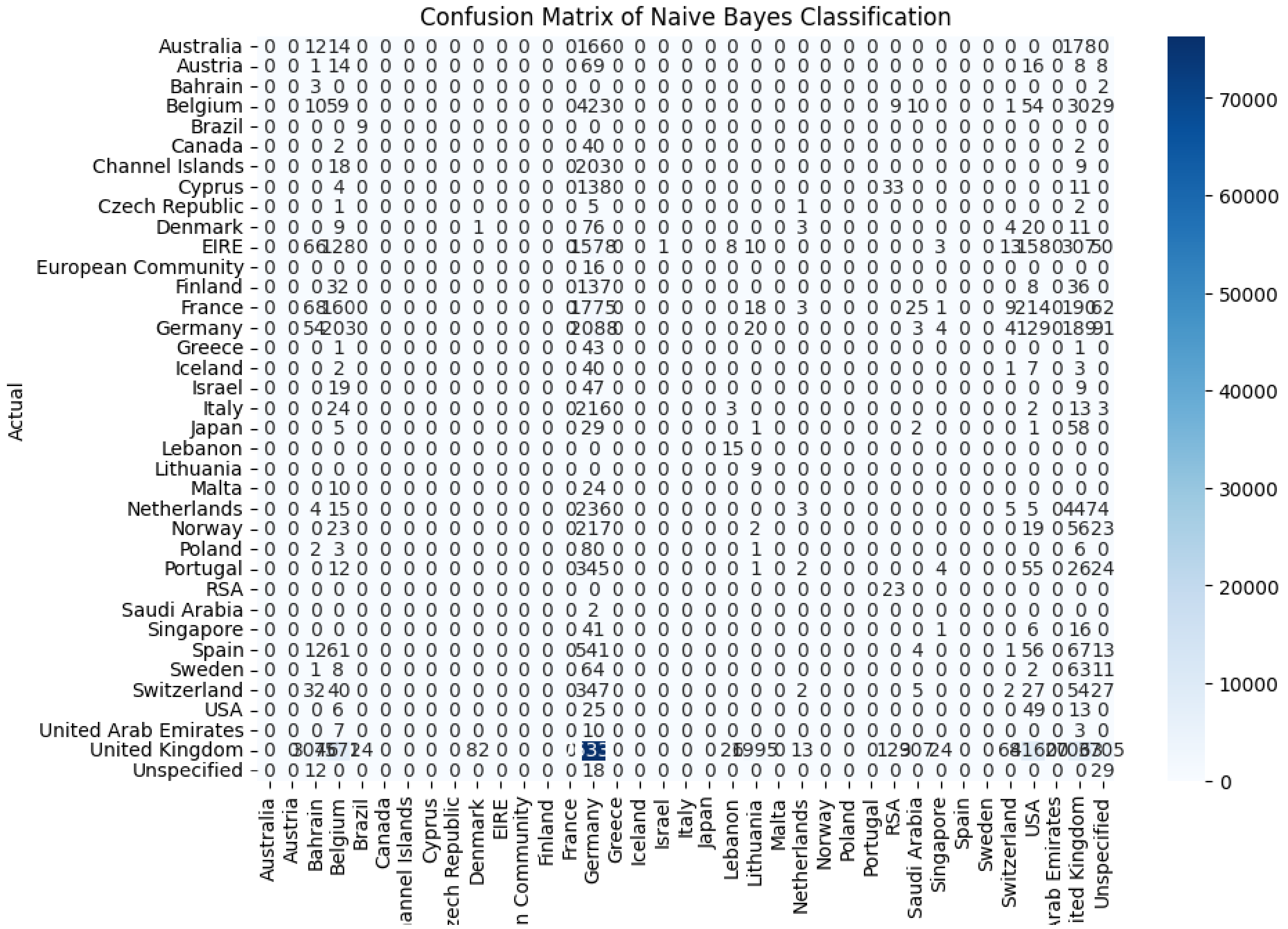
Output:
Predicted Spending Category: High Spender

```
K=1 Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.70      0.70        23
           1       0.59      0.59      0.59        17

    accuracy                           0.65        40
   macro avg       0.64      0.64      0.64        40
weighted avg       0.65      0.65      0.65        40


K=2 Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.78      0.68        23
           1       0.50      0.29      0.37        17

    accuracy                           0.57        40
   macro avg       0.55      0.54      0.52        40
weighted avg       0.56      0.57      0.55        40
```

9. Implement Naïve Bayes Classification algorithm on the Online Retail. Analyse the efficiency of the algorithm using different metrics.

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

dataset = pd.read_csv("Online_Retail.csv", encoding='latin1')

print(dataset.head())

# Dropping rows with missing CustomerID as we need it for classification
dataset = dataset.dropna(subset=['CustomerID'])

dataset['InvoiceDate'] = pd.to_datetime(dataset['InvoiceDate'])
dataset['InvoiceDay'] = dataset['InvoiceDate'].dt.day
dataset['InvoiceMonth'] = dataset['InvoiceDate'].dt.month
dataset['InvoiceHour'] = dataset['InvoiceDate'].dt.hour

X = dataset[['Quantity', 'UnitPrice', 'InvoiceDay', 'InvoiceMonth', 'InvoiceHour']]
y = dataset['Country']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_,
yticklabels=model.classes_)
plt.title('Confusion Matrix of Naive Bayes Classification')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
  InvoiceNo StockCode                          Description  Quantity
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
Accuracy: 0.0105
                      precision    recall  f1-score   support

           Australia       0.00      0.00      0.00       370
             Austria       0.00      0.00      0.00       116
             Bahrain       0.00      0.60      0.00         5
             Belgium       0.01      0.09      0.01       625
              Brazil       0.27      1.00      0.43         9
              Canada       0.00      0.00      0.00        44
     Channel Islands       0.00      0.00      0.00       230
              Cyprus       0.00      0.00      0.00       186
      Czech Republic       0.00      0.00      0.00         9
             Denmark       0.01      0.01      0.01       124
                EIRE       0.00      0.00      0.00      2322
  European Community       0.00      0.00      0.00        16
             Finland       0.00      0.00      0.00       213
              France       0.00      0.00      0.00      2525
             Germany       0.02      0.75      0.05      2785
              Greece       0.00      0.00      0.00        45
             Iceland       0.00      0.00      0.00        53
              Israel       0.00      0.00      0.00        75
               Italy       0.00      0.00      0.00       261
               Japan       0.00      0.00      0.00        96
             Lebanon       0.29      1.00      0.45        15
           Lithuania       0.00      1.00      0.01         9
               Malta       0.00      0.00      0.00        34
         Netherlands       0.11      0.00      0.01       719
              Norway       0.00      0.00      0.00       340
              Poland       0.00      0.00      0.00        92
            Portugal       0.00      0.00      0.00       469
                 RSA       0.12      1.00      0.21        23
```


Confusion Matrix of Naive Bayes Classification
Actual
Australia
Austria
Bahrain
Belgium
Brazil
Canada
Channel Islands
Cyprus
Czech Republic
Denmark
EIRE
European Community
Finland
France
Germany
Greece
Iceland
Israel
Italy
Japan
Lebanon
Lithuania
Malta
Netherlands
Norway
Poland
Portugal
RSA
Saudi Arabia
Singapore
Spain
Sweden
Switzerland
USA
United Arab Emirates
United Kingdom
Unspecified
70000
60000
50000
40000
30000
20000
10000
0