NAME AND REG NO: ABDUL SUBHAN (FA23-BDS-061) MUHAMMAD IBRAHIM (FA23-BDS-062)

STEP 1 — DATA INGESTION & PREPROCESSING

```
import warnings
warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.utcnow() is deprecated and
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.utcnow() is deprecated and
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.utcnow() is deprecated and
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.utcnow() is deprecated and
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.utcnow() is deprecated and
  return datetime.utcnow().replace(tzinfo=utc)
```

```
!pip install mlxtend
```

```
Requirement already satisfied: mlxtend in /usr/local/lib/python3.12/dist-packages (0.23.4)
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.12/dist-packages (from mlxtend) (1.16.3)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.12/dist-packages (from mlxtend) (2.0.2)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.12/dist-packages (from mlxtend) (2.2.2)
Requirement already satisfied: scikit-learn>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from mlxtend) (1.6.1)
Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.12/dist-packages (from mlxtend) (3.10.0)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.12/dist-packages (from mlxtend) (1.5.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0.0->mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0.0->mlxtend) (4.61.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0.0->mlxtend) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0.0->mlxtend) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0.0->mlxtend) (2.9.0
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.24.2->mlxtend) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.24.2->mlxtend) (2025.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=1.3.1->mlxtend) (3.6
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxte
```

Install & Import Libraries

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import pairwise_distances
from sklearn.decomposition import PCA

from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score

from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, roc_curve, auc, accuracy_score
```

```
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.utcnow() is deprecated and
  return datetime.utcnow().replace(tzinfo=utc)
```

Basic Cleaning

```
df = pd.read_csv('data.csv', encoding='latin1')
df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

```
# Remove missing CustomerID
df = df.dropna(subset=['CustomerID'])

# Remove cancelled invoices
df = df[~df['InvoiceNo'].astype(str).str.startswith('C')]

# Convert InvoiceDate to datetime
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
```

Feature Engineering

```
# Total Amount
df['TotalAmount'] = df['Quantity'] * df['UnitPrice']
```

RFM FEATURE

```
latest_date = df['InvoiceDate'].max()

rfm = df.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (latest_date - x.max()).days,
    'InvoiceNo': 'nunique',
    'TotalAmount': 'sum'
})
```

```
rfm.columns = ['Recency', 'Frequency', 'Monetary']
rfm.head()
```

```
            Recency  Frequency  Monetary

CustomerID
 12346.0        325          1  77183.60
 12347.0          1          7   4310.00
 12348.0         74          4   1797.24
 12349.0         18          1   1757.55
 12350.0        309          1    334.40
```

One-Hot Encoding Country

```
country_encoded = pd.get_dummies(df[['CustomerID','Country']], columns=['Country'])
country_encoded = country_encoded.groupby('CustomerID').sum()

rfm = rfm.join(country_encoded, how='left').fillna(0)
```

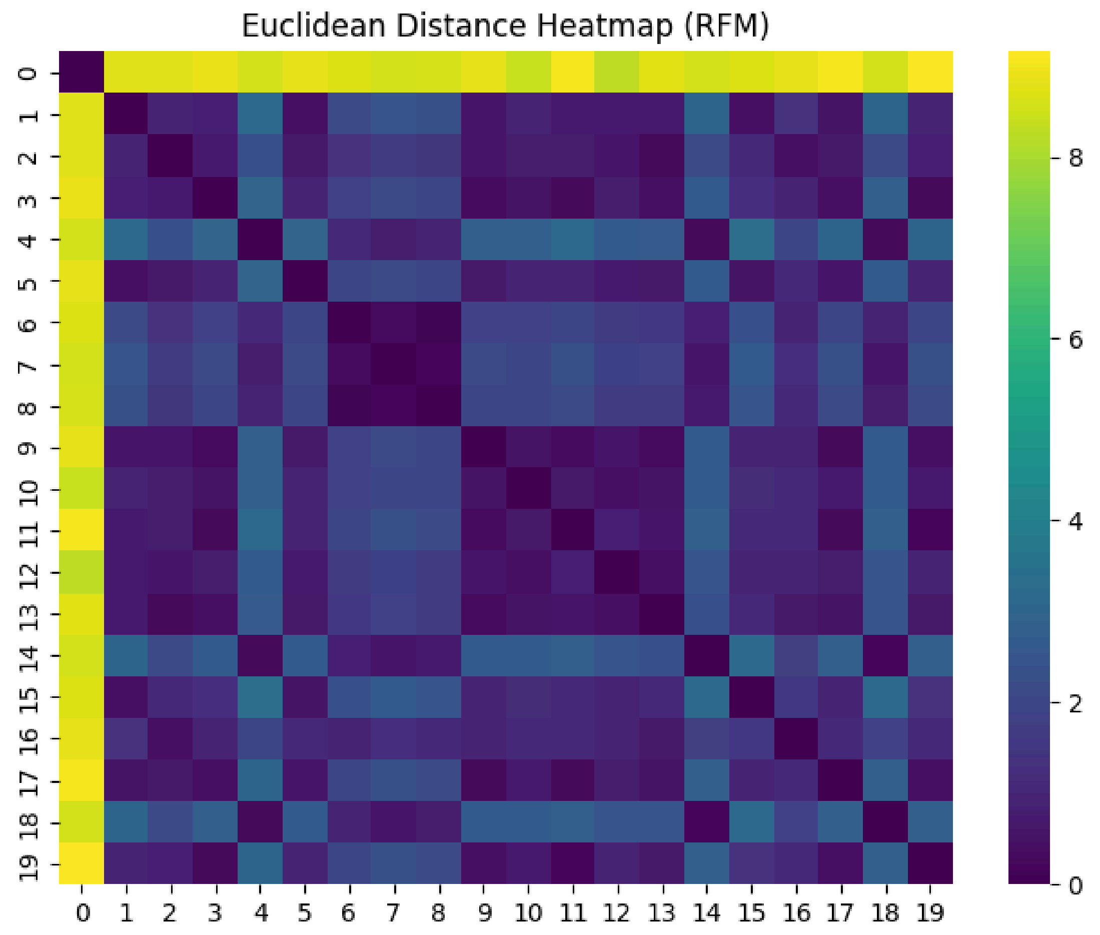
Euclidean Distance (RFM)

```
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm[['Recency','Frequency','Monetary']])

euclidean_dist = pairwise_distances(rfm_scaled, metric='euclidean')
```

Euclidean Heatmap

```
plt.figure(figsize=(8,6))
sns.heatmap(euclidean_dist[:20,:20], cmap='viridis')
plt.title("Euclidean Distance Heatmap (RFM)")
```

```
    plt.show()
```



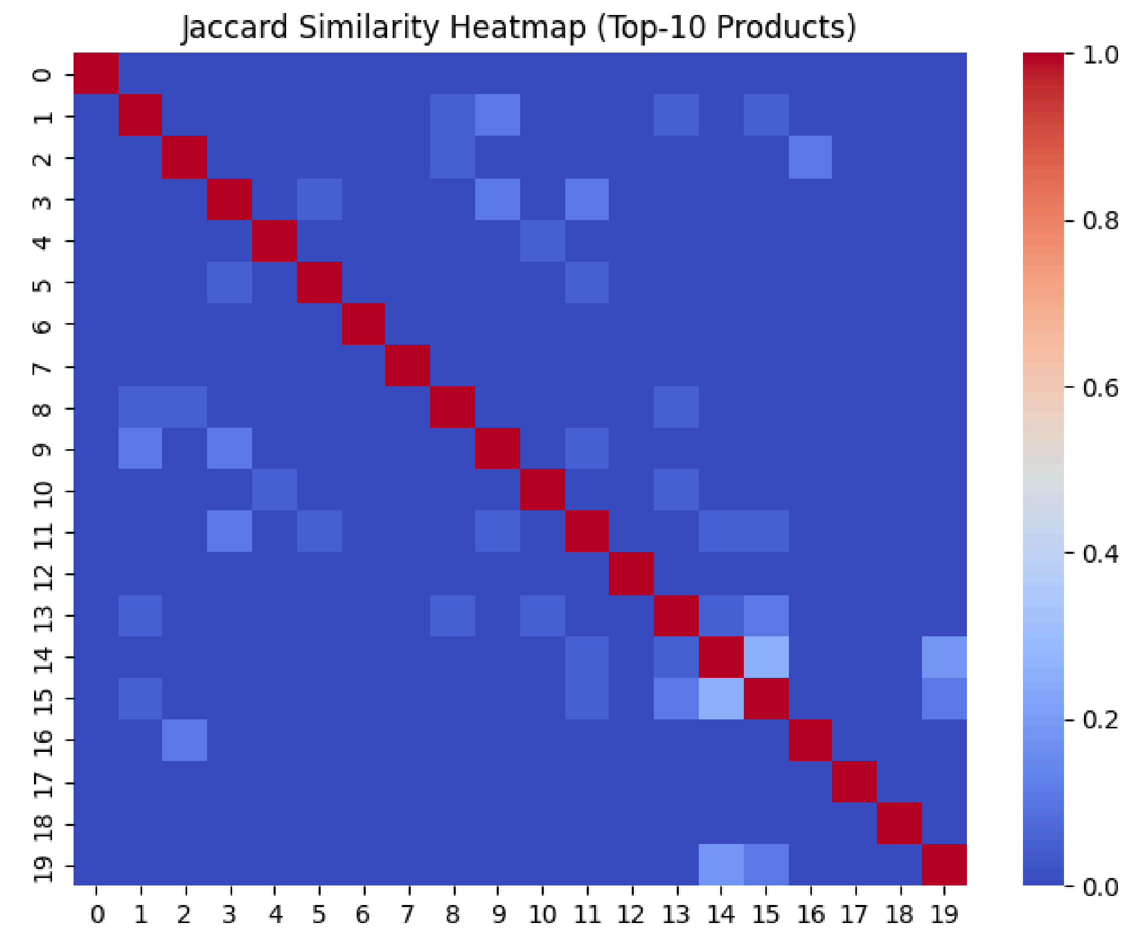
Jaccard Similarity (Top-10 Products)

```
top_products = (
    df.groupby(['CustomerID','StockCode'])['Quantity']
      .sum()
      .reset_index()
)
```

```
top_products = top_products.sort_values(['CustomerID','Quantity'], ascending=False)
top_products = top_products.groupby('CustomerID').head(10)

customer_sets = top_products.groupby('CustomerID')['StockCode'].apply(set)

customers = customer_sets.index[:20]
jaccard_matrix = np.zeros((20,20))

for i, c1 in enumerate(customers):
    for j, c2 in enumerate(customers):
        intersection = len(customer_sets[c1] & customer_sets[c2])
        union = len(customer_sets[c1] | customer_sets[c2])
        jaccard_matrix[i,j] = intersection / union if union != 0 else 0
```

Jaccard Heatmap

```
plt.figure(figsize=(8,6))
sns.heatmap(jaccard_matrix, cmap='coolwarm')
plt.title("Jaccard Similarity Heatmap (Top-10 Products)")
plt.show()
```

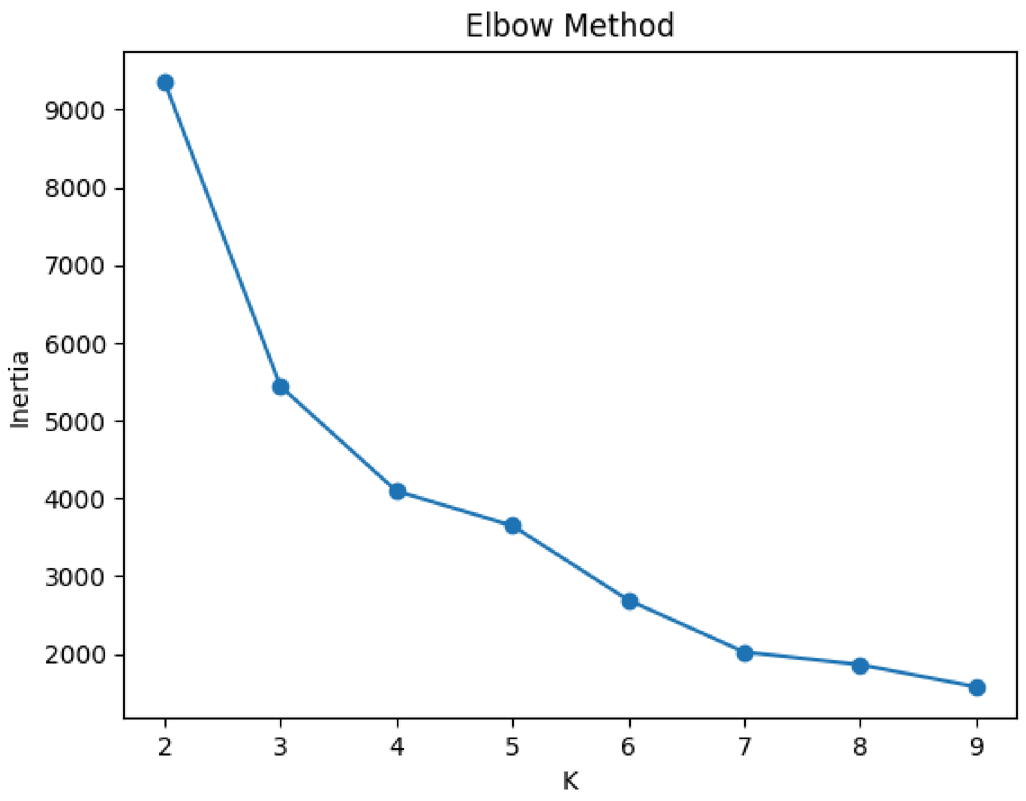


STEP 3 — CLUSTERING KMEANS

Elbow Method

```
inertia = []

for k in range(2,10):
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(rfm_scaled)
    inertia.append(km.inertia_)
```

```
plt.plot(range(2,10), inertia, marker='o')
plt.xlabel("K")
plt.ylabel("Inertia")
plt.title("Elbow Method")
plt.show()
```



KMeans Clustering

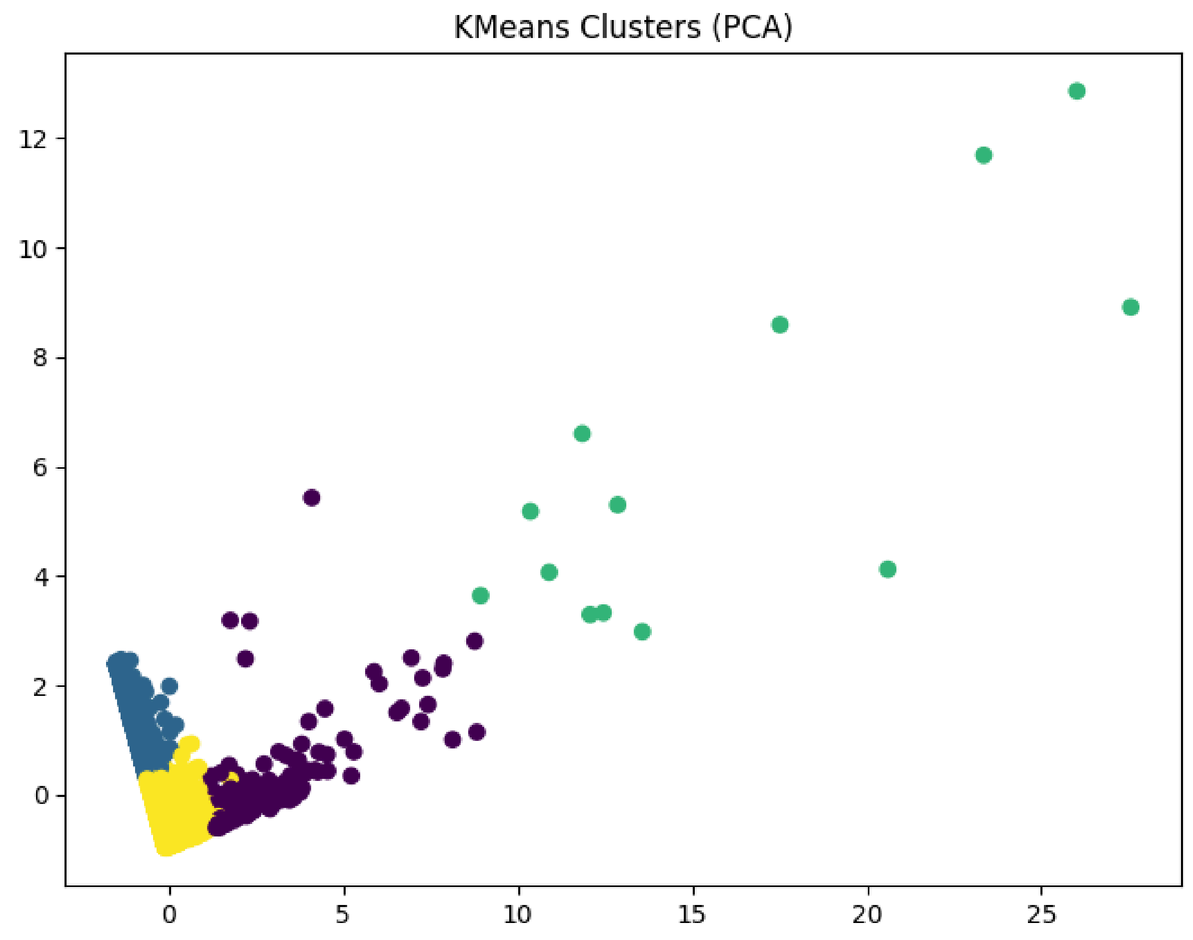
```
kmeans = KMeans(n_clusters=4, random_state=42)
rfm['KMeansCluster'] = kmeans.fit_predict(rfm_scaled)
```

DBSCAN

```
dbscan = DBSCAN(eps=0.8, min_samples=5)
rfm['DBSCANCluster'] = dbscan.fit_predict(rfm_scaled)
```

PCA Visualization

```
pca = PCA(n_components=2)
pca_data = pca.fit_transform(rfm_scaled)

plt.figure(figsize=(8,6))
plt.scatter(pca_data[:,0], pca_data[:,1], c=rfm['KMeansCluster'])
plt.title("KMeans Clusters (PCA)")
plt.show()
```



Basket Creation

```
basket = df.groupby(['InvoiceNo','StockCode'])['Quantity'].sum().unstack().fillna(0)
basket = basket.applymap(lambda x: 1 if x > 0 else 0)
```

Apriori

```
start = pd.Timestamp.now()

freq_items = apriori(basket, min_support=0.02, use_colnames=True)
rules_ap = association_rules(freq_items, metric="confidence", min_threshold=0.6)
rules_ap = rules_ap[rules_ap['lift'] >= 1.2]

apriori_time = pd.Timestamp.now() - start
rules_ap.sort_values('lift', ascending=False).head(10)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_met
19	(22698)	(22697, 22699)	0.029996	0.029186	0.021040	0.701439	24.033032	1.0	0.020165	3.251641	0.988
17	(22697, 22699)	(22698)	0.029186	0.029996	0.021040	0.720887	24.033032	1.0	0.020165	3.475313	0.987
18	(22698, 22699)	(22697)	0.023522	0.037279	0.021040	0.894495	23.994742	1.0	0.020163	9.124923	0.981
5	(22697)	(22698)	0.037279	0.029996	0.024817	0.665702	22.193256	1.0	0.023698	2.901615	0.991
6	(22698)	(22697)	0.029996	0.037279	0.024817	0.827338	22.193256	1.0	0.023698	5.575760	0.984
16	(22697, 22698)	(22699)	0.024817	0.042242	0.021040	0.847826	20.070631	1.0	0.019992	6.293837	0.974
9	(22698)	(22699)	0.029996	0.042242	0.023522	0.784173	18.563760	1.0	0.022255	4.437611	0.975
7	(22697)	(22699)	0.037279	0.042242	0.029186	0.782923	18.534184	1.0	0.027612	4.412071	0.982
8	(22699)	(22697)	0.042242	0.037279	0.029186	0.690932	18.534184	1.0	0.027612	3.114920	0.987
4	(22629)	(22630)	0.037980	0.033233	0.022874	0.602273	18.122934	1.0	0.021612	2.430729	0.982

FP-Growth

```
start = pd.Timestamp.now()

freq_fp = fpgrowth(basket, min_support=0.02, use_colnames=True)
```

```
rules_fp = association_rules(freq_fp, metric="confidence", min_threshold=0.6)
rules_fp = rules_fp[rules_fp['lift'] >= 1.2]

fp_time = pd.Timestamp.now() - start
rules_fp.sort_values('lift', ascending=False).head(10)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_met
15	(22697, 22699)	(22698)	0.029186	0.029996	0.021040	0.720887	24.033032	1.0	0.020165	3.475313	0.987
17	(22698)	(22697, 22699)	0.029996	0.029186	0.021040	0.701439	24.033032	1.0	0.020165	3.251641	0.988
16	(22698, 22699)	(22697)	0.023522	0.037279	0.021040	0.894495	23.994742	1.0	0.020163	9.124923	0.981
11	(22697)	(22698)	0.037279	0.029996	0.024817	0.665702	22.193256	1.0	0.023698	2.901615	0.991
12	(22698)	(22697)	0.029996	0.037279	0.024817	0.827338	22.193256	1.0	0.023698	5.575760	0.984
14	(22697, 22698)	(22699)	0.024817	0.042242	0.021040	0.847826	20.070631	1.0	0.019992	6.293837	0.974
13	(22698)	(22699)	0.029996	0.042242	0.023522	0.784173	18.563760	1.0	0.022255	4.437611	0.975
9	(22697)	(22699)	0.037279	0.042242	0.029186	0.782923	18.534184	1.0	0.027612	4.412071	0.982
10	(22699)	(22697)	0.042242	0.037279	0.029186	0.690932	18.534184	1.0	0.027612	3.114920	0.987
7	(22630)	(22629)	0.033233	0.037980	0.022874	0.688312	18.122934	1.0	0.021612	3.086480	0.977

NAÏVE BAYES CLASSIFICATION

Target Variable

```
threshold = rfm['Monetary'].quantile(0.75)
rfm['HighValue'] = (rfm['Monetary'] > threshold).astype(int)
```

Gaussian Naïve Bayes

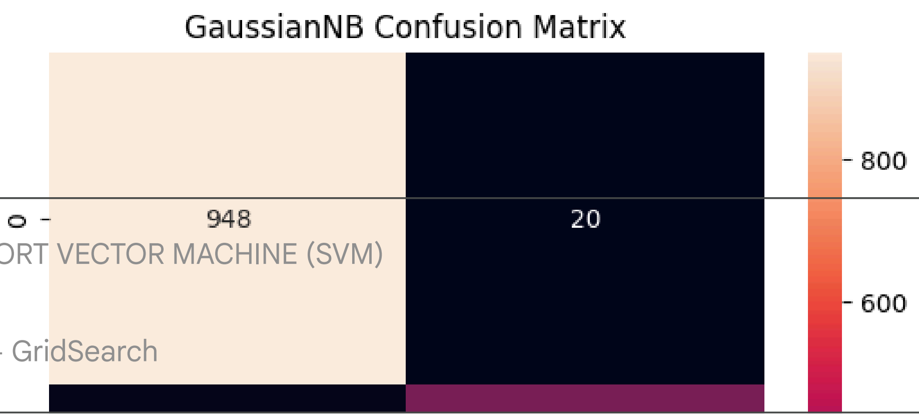
```
X = rfm[['Recency','Frequency','Monetary']]
y = rfm['HighValue']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-1708454712.py in <cell line: 0>()
----> 1 X = rfm[['Recency','Frequency','Monetary']]
      2 y = rfm['HighValue']
      3 
      4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
      5 

NameError: name 'rfm' is not defined
```

Confusion Matrix & ROC

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.title("GaussianNB Confusion Matrix")
plt.show()

y_prob = gnb.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f"AUC={roc_auc:.2f}")
plt.plot([0,1],[0,1],'--')
plt.legend()
plt.title("GaussianNB ROC Curve")
```

```
    plt.show()
```



SUPPORT VECTOR MACHINE (SVM)

SVM + GridSearch

```
param_grid = {'C':[0.1,1,10], 'gamma':[0.01,0.1,1]}

grid = GridSearchCV(SVC(kernel='rbf', probability=True), param_grid, cv=3)
grid.fit(X_train, y_train)
```