iation-rule-mining-minor-project-2

November 23, 2024

1 Customer Segmentation and Purchase Pattern Analysis

Objective Analyze customer behavior through both segmentation (using clustering) and purchase pattern analysis (using association rules) to derive meaningful business insights.

Dataset Requirements - Download the "Online Retail" dataset from UCI Machine Learning Repository or a similar e-commerce dataset from Kaggle

- Dataset should contain:
  - Customer information for clustering
  - Transaction/purchase information for association rule mining

1.1 Task 1:- Data Preprocessing (2 marks)

- Load and clean the dataset
- Handle missing values and duplicates
- Perform outlier detection and removal
- Feature scaling/normalization
- Create relevant features for both clustering and association analysis

1.1.1 (a) Load and clean the dataset

[297]:
```
# Importing required packages
import numpy as np
import pandas as pd
import warnings as war
war.filterwarnings("ignore")
```

[298]:
```
# Defining dataset csv Path
dataSetPath="C:\\Users\\ASUS\\jupyterworkspace\\Assignment & Mini␣
↪Project\Module_04_Unsupervised Learning and Association Rule␣
↪Mining\\MiniProject_02\\Online Retail.csv"
# Loading dataSet
dataSetRead=pd.read_csv(dataSetPath)
```

1.1.2 Inspect structure

[299]:
```
# Displaying first 5 records to confirming data loading
print("*****************************************************Displaying below␣
↪first 5 records*****************************************************")
dataSetRead.head()
```

```
*****************************************************Displaying below first 5
records*****************************************************
```

[299]:
```
  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  01-12-2010 08:26       2.55     17850.0  United Kingdom
1  01-12-2010 08:26       3.39     17850.0  United Kingdom
2  01-12-2010 08:26       2.75     17850.0  United Kingdom
3  01-12-2010 08:26       3.39     17850.0  United Kingdom
4  01-12-2010 08:26       3.39     17850.0  United Kingdom
```

[300]:
```
# Displaying last 5 records to confirming data loading
print("*****************************************************Displaying below␣
↪last 5 records*****************************************************")
dataSetRead.tail()
```

```
*****************************************************Displaying below last 5
records*****************************************************
```

[300]:
```
       InvoiceNo StockCode                      Description  Quantity  \
541904    581587     22613      PACK OF 20 SPACEBOY NAPKINS        12
541905    581587     22899     CHILDREN'S APRON DOLLY GIRL          6
541906    581587     23254    CHILDRENS CUTLERY DOLLY GIRL          4
541907    581587     23255  CHILDRENS CUTLERY CIRCUS PARADE         4
541908    581587     22138    BAKING SET 9 PIECE RETROSPOT          3

             InvoiceDate  UnitPrice  CustomerID Country
541904  09-12-2011 12:50       0.85     12680.0  France
541905  09-12-2011 12:50       2.10     12680.0  France
541906  09-12-2011 12:50       4.15     12680.0  France
541907  09-12-2011 12:50       4.15     12680.0  France
541908  09-12-2011 12:50       4.95     12680.0  France
```

```
[301]: # Displaying all records to confirming data loading
print("*****************************************************Displaying below␣
 ↪all records**********************************************************")
dataSetRead
```

```
*****************************************************Displaying below all
records**********************************************************
```

```
[301]:        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
…             …       …                                  …        …
541904     581587     22613          PACK OF 20 SPACEBOY NAPKINS        12
541905     581587     22899         CHILDREN'S APRON DOLLY GIRL          6
541906     581587     23254        CHILDRENS CUTLERY DOLLY GIRL          4
541907     581587     23255      CHILDRENS CUTLERY CIRCUS PARADE         4
541908     581587     22138        BAKING SET 9 PIECE RETROSPOT          3

              InvoiceDate  UnitPrice  CustomerID         Country
0        01-12-2010 08:26       2.55     17850.0  United Kingdom
1        01-12-2010 08:26       3.39     17850.0  United Kingdom
2        01-12-2010 08:26       2.75     17850.0  United Kingdom
3        01-12-2010 08:26       3.39     17850.0  United Kingdom
4        01-12-2010 08:26       3.39     17850.0  United Kingdom
…                      …        …          …              …
541904   09-12-2011 12:50       0.85     12680.0          France
541905   09-12-2011 12:50       2.10     12680.0          France
541906   09-12-2011 12:50       4.15     12680.0          France
541907   09-12-2011 12:50       4.15     12680.0          France
541908   09-12-2011 12:50       4.95     12680.0          France

[541909 rows x 8 columns]
```

```
[302]: # Displaying dimension of dataSet
print("Dimention of Dataset:- {}".format(dataSetRead.shape[0:2]))
print("Total number of rows in Dataset:- {}".format(dataSetRead.shape[0]))
print("Total number of columns in Dataset:- {}".format(dataSetRead.shape[1]))
```

```
Dimention of Dataset:- (541909, 8)
Total number of rows in Dataset:- 541909
Total number of columns in Dataset:- 8
```

```
[303]: # Displaying description & statistcal summary of the dataSet
dataSetRead.describe().T
```

[303]:

```
                count          mean          std       min       25%       50%  \
Quantity    541909.0      9.552250   218.081158 -80995.00      1.00      3.00
UnitPrice   541909.0      4.611114    96.759853 -11062.06      1.25      2.08
CustomerID  406829.0  15287.690570  1713.600303  12346.00  13953.00  15152.00

                 75%      max
Quantity       10.00  80995.0
UnitPrice       4.13  38970.0
CustomerID  16791.00  18287.0
```

[304]:

```python
# Displaying description & statistcal summary of the categorical variables
dataSetRead.describe(include='object').T
```

[304]:

```
              count unique                                 top    freq
InvoiceNo    541909  25900                              573585    1114
StockCode    541909   4070                              85123A    2313
Description  540455   4223  WHITE HANGING HEART T-LIGHT HOLDER    2369
InvoiceDate  541909  23260                    31-10-2011 14:41    1114
Country      541909     38                      United Kingdom  495478
```

[305]:

```python
# Displaying the columns and their respective data types
dataSetRead.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

1.1.3 (b) Handle missing values and duplicates

[306]:

```python
# Calculating the number of missing values in each column
missingValue_Count=dataSetRead.isnull().sum()
print(missingValue_Count)
```

```
InvoiceNo         0
StockCode         0
Description    1454
```

```
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

[307]:
```
# finding missing values attribues with counts
missingValue_attributes=missingValue_Count[missingValue_Count.
 ↪where(missingValue_Count.values>0).notnull()]
print(missingValue_attributes)
#Finding the attributes's key which have missing values
print("Below is the list of  missing values attributes:- ")
print(missingValue_attributes.keys())
```

```
Description      1454
CustomerID     135080
dtype: int64
Below is the list of  missing values attributes:-
Index(['Description', 'CustomerID'], dtype='object')
```

[308]:
```
# Finding total no. of missing values for dataSet
total_missingvalue_Count=missingValue_Count.sum()
print("total no. of missing values is :- {} ".format(total_missingvalue_Count))
```

```
total no. of missing values is :- 136534
```

[309]:
```
# Removing missing data from dataSet
datasetRead = dataSetRead.dropna(subset=['CustomerID','Description'],␣
 ↪inplace=True)
```

[310]:
```
# Calculating again the number of missing values in each column
missingValue_Count=dataSetRead.isnull().sum()
print(missingValue_Count)
```

```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

[311]:
```
# Checking for duplicate records
duplicateValue_Count=dataSetRead.duplicated().sum()
print("Total no of duplicate records count:- {}".format(duplicateValue_Count))
```

```
Total no of duplicate records count:- 5225
```

[312]:
```
# Removing duplicates data from dataSet
dataSetRead=dataSetRead.drop_duplicates()
```

[313]:
```
# Checking again for duplicate records
duplicateValue_Count=dataSetRead.duplicated().sum()
print("Total no of duplicate records count:- {}".format(duplicateValue_Count))
```

```
Total no of duplicate records count:- 0
```

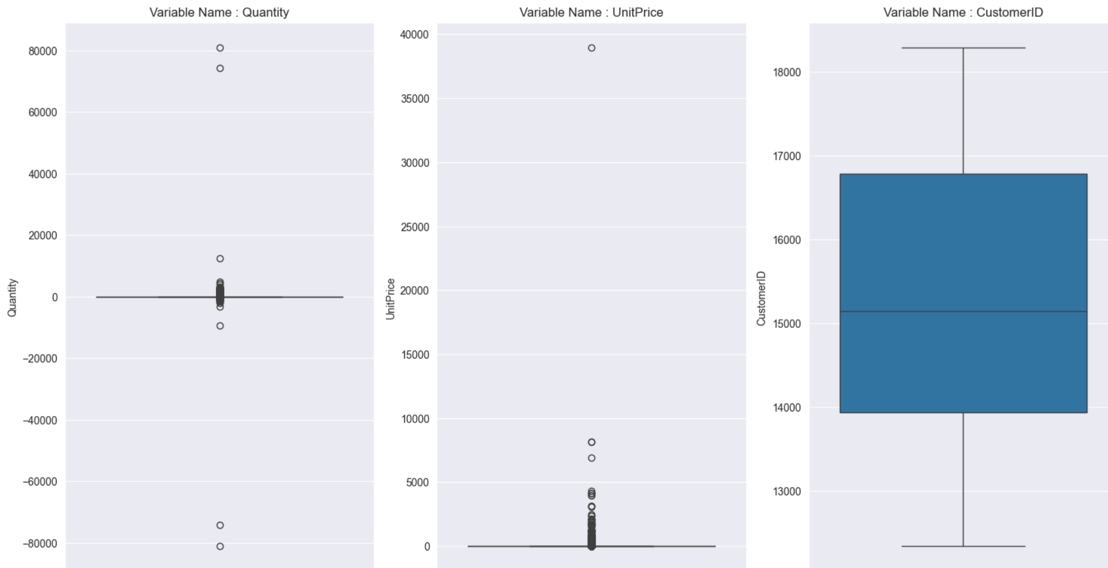
[314]:
```
# Displaying dimension of dataSet
print("Dimention of Dataset:- {}".format(dataSetRead.shape[0:2]))
print("Total number of rows in Dataset:- {}".format(dataSetRead.shape[0]))
print("Total number of columns in Dataset:- {}".format(dataSetRead.shape[1]))
```

```
Dimention of Dataset:- (401604, 8)
Total number of rows in Dataset:- 401604
Total number of columns in Dataset:- 8
```

1.1.4 (c) Perform outlier detection and removal

[315]:
```
# Importing required package
import seaborn as sbn
import matplotlib.pyplot as plt
# taking list of all numerocal attributes
numerical_attributes = ['Quantity', 'UnitPrice','CustomerID']
dataSetRead_new = dataSetRead[numerical_attributes]
# Set the parameters that control the general style of the plots.
sbn.set_style("darkgrid")
# Selecting all numerical attributes columns names
cols = dataSetRead_new.columns
#Plotting Box plot for each numerical attributes
plt.figure(figsize=(15, 30) )
for count, item in enumerate(cols, 1):
    plt.subplot(4, 3, count)
    sbn.boxplot(dataSetRead_new[item])
    plt.title(f"Variable Name : {item}")

plt.tight_layout()
plt.show()
```



[316]:

```
# Calculating IQR for Quantity and UnitPrice
Q1 = dataSetRead[['Quantity', 'UnitPrice','CustomerID']].quantile(0.25)
Q3 = dataSetRead[['Quantity', 'UnitPrice','CustomerID']].quantile(0.75)
IQR = Q3 - Q1

# Identify outliers for each attribute
outliers_quantity = ((dataSetRead['Quantity'] < (Q1['Quantity'] - 1.5 *␣
↪IQR['Quantity'])) |
                     (dataSetRead['Quantity'] > (Q3['Quantity'] + 1.5 *␣
↪IQR['Quantity']))).sum()

outliers_unitprice = ((dataSetRead['UnitPrice'] < (Q1['UnitPrice'] - 1.5 *␣
↪IQR['UnitPrice'])) |
                      (dataSetRead['UnitPrice'] > (Q3['UnitPrice'] + 1.5 *␣
↪IQR['UnitPrice']))).sum()

outliers_CustomerID = ((dataSetRead['CustomerID'] < (Q1['CustomerID'] - 1.5 *␣
↪IQR['CustomerID'])) |
                      (dataSetRead['CustomerID'] > (Q3['CustomerID'] + 1.5 *␣
↪IQR['CustomerID']))).sum()

# Displaying results for each attribute
print(f"Total outliers in 'Quantity': {outliers_quantity}")
print(f"Total outliers in 'UnitPrice': {outliers_unitprice}")
print(f"Total outliers in 'CustomerID': {outliers_CustomerID}")
```

```
Total outliers in 'Quantity': 26646
Total outliers in 'UnitPrice': 35802
```

```
Total outliers in 'CustomerID': 0
```

Observation:- From the above box plots it shows that there is no outliers in the CustomerID attribute

[317]:
```python
# Removing outliers from dataSet
# Identify outliers
outlier_mask = (dataSetRead[['Quantity', 'UnitPrice','CustomerID']] < (Q1 - 1.5␣
 ↪* IQR)) | \
               (dataSetRead[['Quantity', 'UnitPrice','CustomerID']] > (Q3 + 1.5␣
 ↪* IQR))

# Remove outliers
dataSetRead = dataSetRead[~outlier_mask.any(axis=1)]
```

[318]:
```python
# Displaying dimension of dataSet
print("Dimention of Dataset:- {}".format(dataSetRead.shape[0:2]))
print("Total number of rows in Dataset:- {}".format(dataSetRead.shape[0]))
print("Total number of columns in Dataset:- {}".format(dataSetRead.shape[1]))
```

```
Dimention of Dataset:- (339453, 8)
Total number of rows in Dataset:- 339453
Total number of columns in Dataset:- 8
```

1.1.5 (d) Feature scaling/normalization

[319]:
```python
# Importing required packages
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
dataSetRead[['Quantity', 'UnitPrice']] = scaler.
 ↪fit_transform(dataSetRead[['Quantity', 'UnitPrice']])
```

[320]:
```python
# Displaying all records
print("*****************************************************Displaying below␣
 ↪all records**********************************************************")
dataSetRead
```

```
*****************************************************Displaying below all
records**********************************************************
```

[320]:

	InvoiceNo	StockCode	Description	Quantity \
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	0.475
1	536365	71053	WHITE METAL LANTERN	0.475
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	0.525
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	0.475
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	0.475
…	…	…	…	…
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	0.625

```
541905    581587     22899        CHILDREN'S APRON DOLLY GIRL      0.475
541906    581587     23254       CHILDRENS CUTLERY DOLLY GIRL      0.425
541907    581587     23255     CHILDRENS CUTLERY CIRCUS PARADE     0.425
541908    581587     22138       BAKING SET 9 PIECE RETROSPOT      0.400

             InvoiceDate  UnitPrice  CustomerID         Country
0       01-12-2010 08:26   0.340000     17850.0  United Kingdom
1       01-12-2010 08:26   0.452000     17850.0  United Kingdom
2       01-12-2010 08:26   0.366667     17850.0  United Kingdom
3       01-12-2010 08:26   0.452000     17850.0  United Kingdom
4       01-12-2010 08:26   0.452000     17850.0  United Kingdom
…                    …        …          …             …
541904  09-12-2011 12:50   0.113333     12680.0          France
541905  09-12-2011 12:50   0.280000     12680.0          France
541906  09-12-2011 12:50   0.553333     12680.0          France
541907  09-12-2011 12:50   0.553333     12680.0          France
541908  09-12-2011 12:50   0.660000     12680.0          France

[339453 rows x 8 columns]
```

1.1.6 (e) Create relevant features for both clustering and association analysis

*** For Clustering:** RFM features:

–> Recency: Days since last purchase.

–> Frequency: Number of transactions per customer.

–> Monetary: Total spend per customer.

[321]:
```python
# Importing required packages
from datetime import datetime
from datetime import timedelta
```

[322]:
```python
# Creating Recency, Frequency, Monetary features
dataSetRead['InvoiceDate'] = pd.
 ↪to_datetime(dataSetRead['InvoiceDate'],format="%d-%m-%Y %H:%M")
snapshot_date = max(dataSetRead['InvoiceDate']) + timedelta(days=1)
rfm = dataSetRead.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (snapshot_date - x.max()).days,
    'InvoiceNo': 'count',
    'UnitPrice': 'sum'
}).rename(columns={
    'InvoiceDate': 'Recency',
    'InvoiceNo': 'Frequency',
    'UnitPrice': 'Monetary'
})
```

*** For Association Rule Mining:** Create a basket format:

```
--> Group data by InvoiceNo and pivot StockCode or Description into a binary matrix (0/1).

--> Use this format for market basket analysis.
```

[323]:
```
# Basket format
basket = dataSetRead.groupby(['InvoiceNo', 'Description'])['Quantity'].sum().
↪unstack().fillna(0)
basket = basket.applymap(lambda x: 1 if x > 0 else 0)
```

1.2 Task 2:- Customer Segmentation using Clustering (5 marks)

- Apply Elbow method to determine optimal number of clusters (1 mark)
- Implement three clustering algorithms:
  - K-means clustering (1.5 marks)
  - Hierarchical clustering (1.5 marks)
  - DBSCAN (1 mark)
- Calculate and compare Silhouette coefficients for each method

1.2.1 (a) Apply Elbow method to determine optimal number of clusters

The Elbow method helps determine the optimal number of clusters (k) for clustering algorithms like K-Means by evaluating the Within-Cluster-Sum-of-Squares (WCSS) for different k values.

Steps:

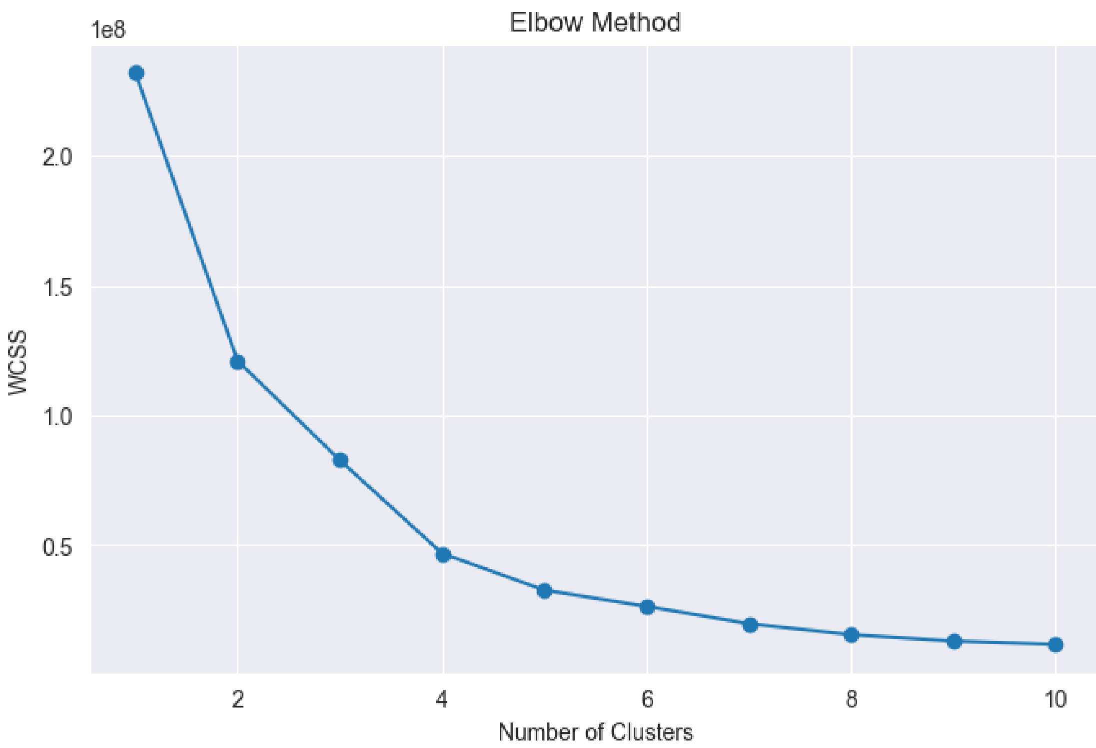
1- Compute K-Means for a range of k values (e.g., 1 to 10).

2- Plot k vs WCSS.

3- Look for the "elbow point" where adding clusters doesn't reduce WCSS significantly.

[324]:
```
# Importing required package
from sklearn.cluster import KMeans
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(rfm)
    wcss.append(kmeans.inertia_)

# Plotting the Elbow graph
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



From the Elbow method above graph , the optimal number of clusters k can typically be identified at the "elbow" point of the curve, where the rate of decrease in the Within-Cluster Sum of Squares (WCSS) starts to slow down.

Looking at the plot, it seems that the "elbow" occurs at k = 3, as the WCSS decreases significantly at first and then levels off after 3 clusters. This is where the curve starts to flatten out, indicating that adding more clusters doesn't result in substantial improvements.

Thus, based on this graph, the optimal value of k is 3.

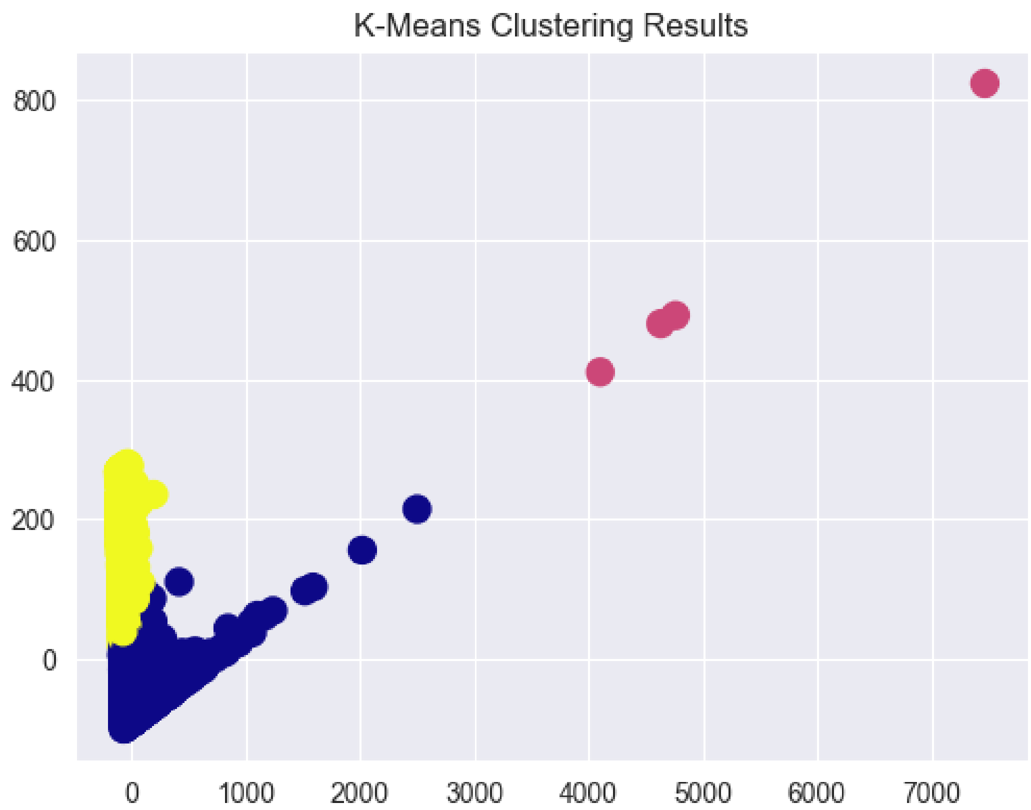
1.2.2 (b) Implement three clustering algorithms:

(b.1) K-Means Clustering –> Use the optimal identified from the Elbow method.

–> Apply the KMeans algorithm.

[325]:
```
# K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
#rfm['KMeans_Cluster'] = kmeans.fit_predict(rfm)
kmeans_labels = kmeans.fit_predict(rfm)
kmeans_silhouette = silhouette_score(rfm, kmeans_labels)
# Visualize clusters (optional, using 2D or 3D PCA)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
rfm_pca = pca.fit_transform(rfm)
```

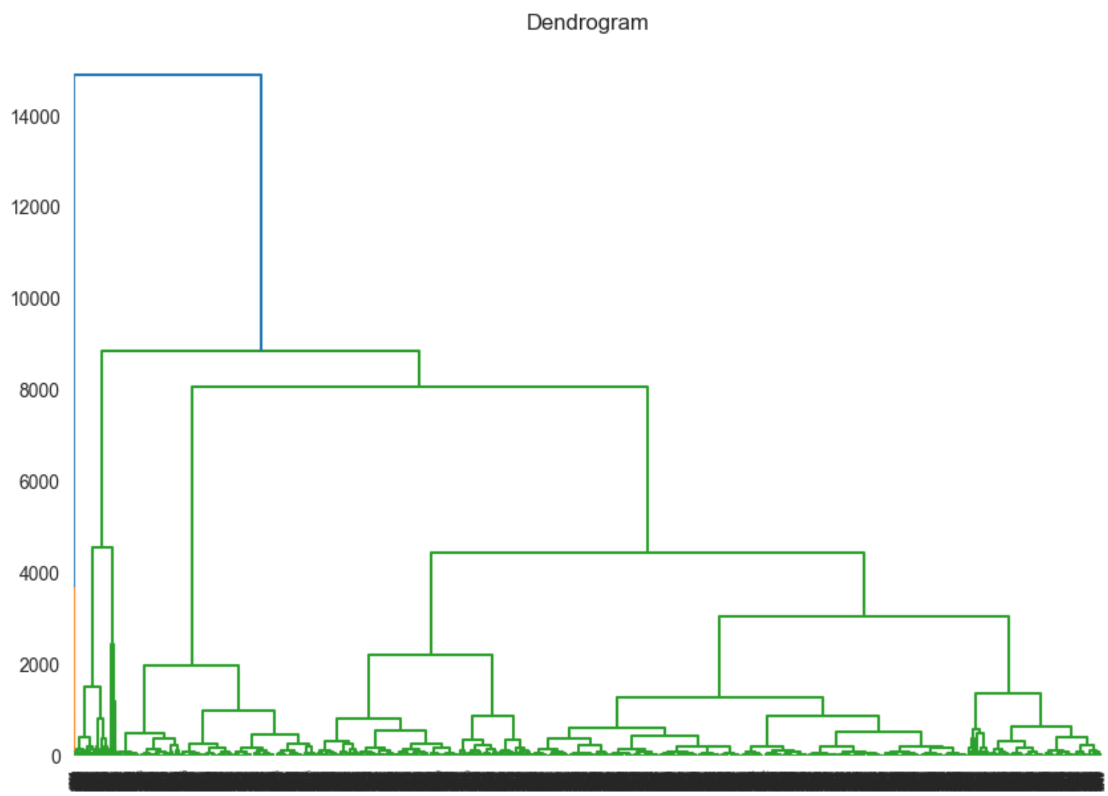
```
plt.scatter(rfm_pca[:, 0], rfm_pca[:, 1], c=kmeans_labels, cmap="plasma", s=100)
plt.title('K-Means Clustering Results')
plt.show()
```



(b.2) Hierarchical Clustering –> Use Agglomerative Clustering.

–> Visualize the dendrogram to validate clusters.

[326]:
```
# Importing required packages
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
# Dendrogram
linked = linkage(rfm, method='ward')
plt.figure(figsize=(10, 7))
dendrogram(linked)
plt.title('Dendrogram')
plt.show()
```



[327]:
```
# Apply Agglomerative Clustering
hierarchical_clustering = AgglomerativeClustering(n_clusters=3,␣
↪metric='euclidean', linkage='ward')
rfm['Hierarchical_Cluster'] = hierarchical_clustering.
↪fit_predict(rfm[['Recency', 'Frequency', 'Monetary']])
```

(b.3) DBSCAN (Density-Based Spatial Clustering of Applications with Noise) –> DBSCAN identifies clusters based on density rather than predetermined

–> Use eps (radius) and min_samples (min points per cluster) to define density.

[328]:
```
from sklearn.cluster import DBSCAN

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=3)
rfm['DBSCAN_Cluster'] = dbscan.fit_predict(rfm[['Recency', 'Frequency',␣
↪'Monetary']])
```

1.2.3 (c) Calculate and compare Silhouette coefficients for each method

The Silhouette coefficient measures how similar each point is to its own cluster compared to other clusters. A high silhouette score indicates good clustering.

[329]:
```python
# Importing required package
from sklearn.metrics import silhouette_score

# Silhouette scores
kmeans_silhouette = silhouette_score(rfm[['Recency', 'Frequency', 'Monetary']],␣
 ↪kmeans_labels)
hierarchical_silhouette = silhouette_score(rfm[['Recency', 'Frequency',␣
 ↪'Monetary']], rfm['Hierarchical_Cluster'])
dbscan_silhouette = silhouette_score(rfm[['Recency', 'Frequency', 'Monetary']],␣
 ↪rfm['DBSCAN_Cluster'])

# Print results
print(f"K-Means Silhouette Score: {kmeans_silhouette}")
print(f"Hierarchical Clustering Silhouette Score: {hierarchical_silhouette}")
print(f"DBSCAN Silhouette Score: {dbscan_silhouette}")
```

```
K-Means Silhouette Score: 0.4998255169836387
Hierarchical Clustering Silhouette Score: 0.6959473543829896
DBSCAN Silhouette Score: -0.5909900158153845
```

Interpretation of Scores K-Means Clustering:

Silhouette Score: 0.4998

This indicates that the clustering is moderately well-formed, with a decent separation between clusters but some degree of overlap or points near

cluster boundaries.

Hierarchical Clustering:

Silhouette Score: 0.6959

This is a high score, suggesting that the clusters are well-separated, compact, and appropriately represent the underlying structure of the data.

It outperforms K-Means in this case.

DBSCAN:

Silhouette Score: -0.5909

A negative score indicates that the clustering is poor, with many points assigned to incorrect clusters or classified as noise. This is likely due

to the dataset not being well-suited to DBSCAN or suboptimal tuning of its eps and min_samples parameters.

1.2.4 Task 3:-Purchase Pattern Analysis using Association Rules (5 marks)

- Prepare transaction data for association rule mining
- Define support and confidence thresholds

- Apply Apriori algorithm to discover frequent itemsets
- Generate and analyze association rules
- Compare rules based on different metrics (support, confidence, lift)
- Provide business insights from discovered rules

(a) Prepare transaction data for association rule mining Association rule mining requires transactional data in the format of InvoiceID and associated Items.

Steps: –> Transform the Dataset:

```
Group data by InvoiceNo and aggregate items into lists.

Exclude invalid transactions, such as cancellations (InvoiceNo starting with "C") and outliers
```

–> Create a Transaction-Item Matrix:

```
Use one-hot encoding to transform item transactions into a binary matrix.
```

[330]:
```
# Filter valid transactions
transactions = dataSetRead[dataSetRead['Quantity'] > 0]
transactions = transactions[~transactions['InvoiceNo'].str.contains('C',␣
↪na=False)]

# Group by InvoiceNo and aggregate items
transaction_data = transactions.groupby('InvoiceNo')['Description'].apply(list)

# Convert to a transactional matrix for one-hot encoding

from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()
transaction_matrix = te.fit_transform(transaction_data)
df_transactions = pd.DataFrame(transaction_matrix, columns=te.columns_)
```

(b) Define support and confidence thresholds Support: Minimum proportion of transactions containing an itemset.

Confidence: Conditional probability of consequent given antecedent.

Example:

Set support = 0.01 (1% of all transactions).

Set confidence = 0.2 (20% likelihood).

(c) Apply Apriori algorithm to discover frequent itemsets The Apriori algorithm identifies frequent itemsets based on the minimum support threshold.

[331]:
```python
from mlxtend.frequent_patterns import apriori

# Generate frequent itemsets
frequent_itemsets = apriori(df_transactions, min_support=0.01,␣
↪use_colnames=True)

# Add the length of itemsets for analysis
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(len)

print(frequent_itemsets)
```

```
      support                                          itemsets  length
0    0.010811                 ( SET 2 TEA TOWELS I LOVE LONDON )       1
1    0.012652                            (10 COLOUR SPACEBOY PEN)       1
2    0.010573                   (12 MESSAGE CARDS WITH ENVELOPES)       1
3    0.015800                     (12 PENCIL SMALL TUBE WOODLAND)       1
4    0.016810                (12 PENCILS SMALL TUBE RED RETROSPOT)       1
..        …                                                 …     …
929  0.014137  (LUNCH BAG RED RETROSPOT, LUNCH BAG WOODLAND, …       3
930  0.012058  (LUNCH BAG RED RETROSPOT, LUNCH BAG SUKI DESIG…       3
931  0.010039  (LUNCH BAG SUKI DESIGN , LUNCH BAG WOODLAND, L…       3
932  0.010692  (POPPY'S PLAYHOUSE LIVINGROOM , POPPY'S PLAYHO…       3
933  0.010395  (LUNCH BAG RED RETROSPOT, LUNCH BAG PINK POLKA…       4

[934 rows x 3 columns]
```

(d) Generate and analyze association rules Use the frequent itemsets to generate association rules with metrics like support, confidence, and lift.

[332]:
```python
from mlxtend.frequent_patterns import association_rules

# Generate rules
rules = association_rules(frequent_itemsets,metric="confidence",␣
↪min_threshold=0.2,num_itemsets=len(transaction_data))

# Add additional metrics for ranking
rules['lift'] = rules['lift']
rules = rules.sort_values(by='lift', ascending=False)

print(rules.head())
```

```
                antecedents              consequents  antecedent support  \
147      (HERB MARKER THYME)  (HERB MARKER ROSEMARY)            0.010811
146   (HERB MARKER ROSEMARY)     (HERB MARKER THYME)            0.011108
144   (HERB MARKER ROSEMARY)   (HERB MARKER PARSLEY)            0.011108
145    (HERB MARKER PARSLEY)  (HERB MARKER ROSEMARY)            0.010930
140   (HERB MARKER ROSEMARY)     (HERB MARKER BASIL)            0.011108
```

```
     consequent support   support  confidence       lift  representativity  \
147            0.011108  0.010217    0.945055  85.080214               1.0
146            0.010811  0.010217    0.919786  85.080214               1.0
144            0.010930  0.010157    0.914439  83.666153               1.0
145            0.011108  0.010157    0.929348  83.666153               1.0
140            0.010930  0.010039    0.903743  82.687602               1.0

     leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski
147  0.010097   17.997838       0.999047  0.873096   0.944438    0.932421
146  0.010097   12.331892       0.999347  0.873096   0.918909    0.932421
144  0.010036   11.559760       0.999146  0.855000   0.913493    0.921893
145  0.010036   13.996628       0.998966  0.855000   0.928554    0.921893
140  0.009917   10.275342       0.999003  0.836634   0.902680    0.911111
```

Metrics for Rule Evaluation: 1- Support: Measures how frequently the itemset occurs.

2- Confidence: Measures the likelihood of seeing the consequent given the antecedent.

3- Lift: Measures the strength of the rule compared to random chance (lift > 1 indicates a strong rule).

(d) Compare rules based on different metrics (support, confidence, lift)

[333]:
```
# Sorting rules by support in descending order
rules_sorted_by_support = rules.sort_values('support', ascending=False)
print("Rules Sorted by Support:")
print(rules_sorted_by_support[['antecedents', 'consequents', 'support']].head())
```

```
Rules Sorted by Support:
                           antecedents                            consequents  \
110  (ROSES REGENCY TEACUP AND SAUCER )  (GREEN REGENCY TEACUP AND SAUCER)
111   (GREEN REGENCY TEACUP AND SAUCER)  (ROSES REGENCY TEACUP AND SAUCER )
25          (ALARM CLOCK BAKELIKE RED )        (ALARM CLOCK BAKELIKE GREEN)
26         (ALARM CLOCK BAKELIKE GREEN)         (ALARM CLOCK BAKELIKE RED )
373           (LUNCH BAG RED RETROSPOT)           (LUNCH BAG PINK POLKADOT)

      support
110  0.030710
111  0.030710
25   0.030591
26   0.030591
373  0.028631
```

[334]:
```
# Sorting rules by confidence in descending order
rules_sorted_by_confidence = rules.sort_values('confidence', ascending=False)
print("Rules Sorted by Confidence:")
print(rules_sorted_by_confidence[['antecedents', 'consequents', 'confidence']].
 ↪head())
```

```
Rules Sorted by Confidence:
                antecedents             consequents  confidence
147     (HERB MARKER THYME)  (HERB MARKER ROSEMARY)    0.945055
145   (HERB MARKER PARSLEY)  (HERB MARKER ROSEMARY)    0.929348
146  (HERB MARKER ROSEMARY)     (HERB MARKER THYME)    0.919786
141     (HERB MARKER BASIL)  (HERB MARKER ROSEMARY)    0.918478
144  (HERB MARKER ROSEMARY)   (HERB MARKER PARSLEY)    0.914439
```

[335]:
```python
# Sorting rules by lift in descending order
rules_sorted_by_lift = rules.sort_values('lift', ascending=False)
print("Rules Sorted by Lift:")
print(rules_sorted_by_lift[['antecedents', 'consequents', 'lift']].head())
```

```
Rules Sorted by Lift:
                antecedents             consequents       lift
147     (HERB MARKER THYME)  (HERB MARKER ROSEMARY)  85.080214
146  (HERB MARKER ROSEMARY)     (HERB MARKER THYME)  85.080214
144  (HERB MARKER ROSEMARY)   (HERB MARKER PARSLEY)  83.666153
145   (HERB MARKER PARSLEY)  (HERB MARKER ROSEMARY)  83.666153
140  (HERB MARKER ROSEMARY)     (HERB MARKER BASIL)  82.687602
```

Comparison of Rules Based on Metrics Support

Definition: Proportion of transactions containing both antecedent and consequent.

Top Rule:

```
--> Antecedent: (ROSES REGENCY TEACUP AND SAUCER)

--> Consequent: (GREEN REGENCY TEACUP AND SAUCER)

--> Support: 0.030710 (Appears in 3.07% of transactions).
```

Insights:

```
--> These are frequent combinations, suitable for promotions targeting large customer segmen

-->  Useful for bulk purchasing or display optimization.
```

Confidence
Definition: Likelihood of consequent being purchased given the antecedent.
Top Rule:
–> Antecedent: (HERB MARKER THYME) –> Consequent: (HERB MARKER ROSEMARY) –> Confidence: 0.945055 (94.5% of transactions with THYME also include ROSEMARY).
Insights:
–> High confidence indicates strong predictability.
–> Use for targeted recommendations, e.g., recommending ROSEMARY if THYME is in a customer's cart. .

Lift

Definition: Strength of the association compared to random co-occurrence.

Top Rule:

```
--> Antecedent: (HERB MARKER THYME)

--> Consequent: (HERB MARKER ROSEMARY)

--> Lift: 85.08 (Occurs 85 times more than expected by chance).
```

Insights:

```
--> High lift suggests a meaningful, non-random relationship.

--> Useful for strategic bundling or cross-selling of products.-
```

1.2.5 (e) Provide business insights from discovered rules

1.2.6 Based on the discovered rules:

1- Cross-Selling Opportunities:

–> Items frequently purchased together (e.g., {Gift Bag} → {Wrapping Paper}) can be bundled to increase sales.

2- Inventory Management:

–> High-lift rules highlight products with strong associations, helping optimize stock placement.

3- Promotional Strategies:

–> Create promotions targeting items in high-confidence rules to drive sales (e.g., discounts on {Paper Garland} with {Paper Lantern}).

4- Personalized Recommendations:

–> Use the rules to recommend related products in e-commerce platforms.