Customer Segmentation Analysis and Metrics using Python

Group 939

ITMD 522

Srujan Shetty

Alliance Tedonfouet

Karthik Gande

Sri Bhavya Chaganti

Shilpitha Reddy Peruvula

Introduction

Customer segmentation is a crucial aspect of marketing strategy, aiming to divide a customer base into groups with similar characteristics such as purchasing behavior, preferences, and demographics. By understanding the distinct needs and behaviors of different customer segments, businesses can tailor their marketing efforts more effectively, improve customer satisfaction, and ultimately drive higher revenues.

Introduction

Overview of the project: Customer Segmentation using Clustering

Objectives: Clean data, perform feature engineering, select a model, and evaluate the segmentation

Dataset Overview

Target variable:

- In this, the target variable may vary depending on the specific task being performed

Columns:

- InvoiceNo: Unique identifier for each invoice
- StockCode: Identifier for items in each invoice
- Description: Textual description of items
- Quantity: Quantity of each item purchased
- InvoiceDate: Date and time of purchase
- UnitPrice: Price of each item
- CustomerID: Identifier for the customer
- Country: Country of the customer

Data Shape:

```
print(df1.shape)
display(HTML(df1.head(7).to_html()))

#Printing column names
print(df1.columns)
```

```
Size of our Data:
(541909, 8)
```

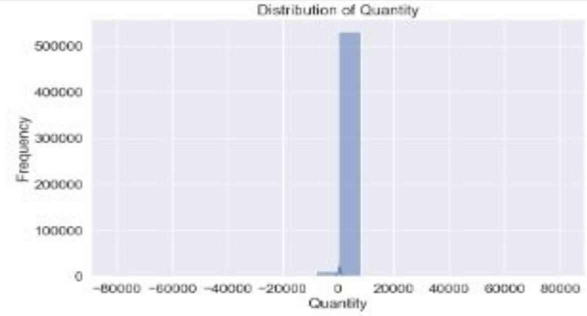
	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850.0	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850.0	United Kingdom

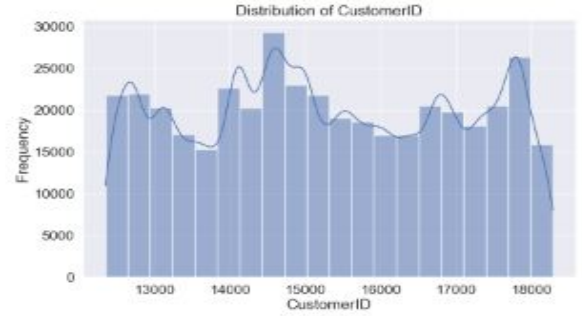
```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

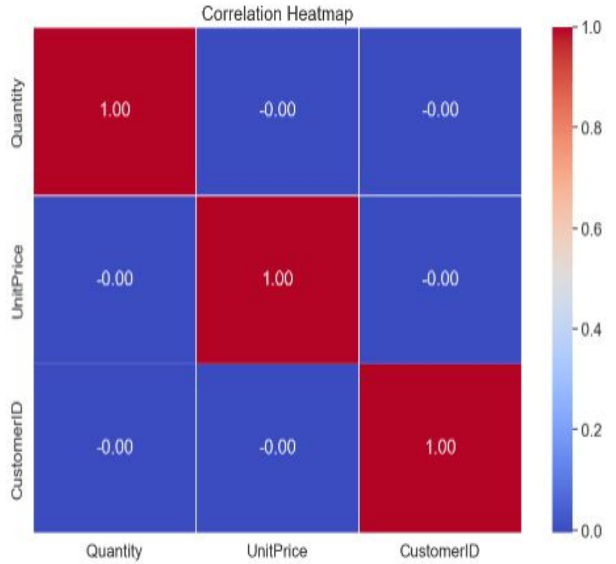
Data Source Analysis:

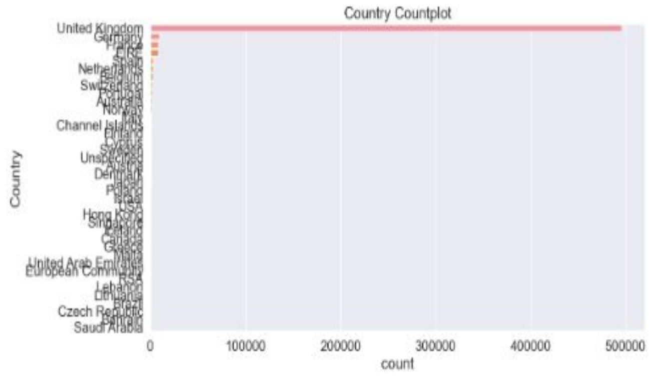
```python
## Checking for missing values
print('ColumnName, DataType, MissingValues')
for col in cols:
    print(f"{col}, {df1[col].dtype}, {df1[col].isnull().sum()}")

# Describing the dataset
print("\nDataset Description:")
print(df1.describe())


# Check and handle numerical features
numerical_features = df1.select_dtypes(include=[np.number]).columns.tolist()

# Check if there are numerical features to plot
if numerical_features:
    for col in numerical_features:
        plt.figure(figsize=(10, 6))
        sns.histplot(df1[col], bins=20, kde=True)
        plt.title(f"Distribution of {col}")
        plt.xlabel(col)
        plt.ylabel("Frequency")
        plt.show()
else:
    print("No numerical features to plot.")

# Correlation heatmap
if numerical_features:
    correlation_matrix = df1[numerical_features].corr()
    plt.figure(figsize=(12, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
    plt.title("Correlation Heatmap")
    plt.show()
else:
    print("No numerical features to compute correlation.")

# Countplot for categorical features
col = 'Country'
plt.figure(figsize=(12, 6))
sns.countplot(data=df1, y=col, order=df1[col].value_counts().index)
plt.title(f"{col} Countplot")
plt.show()
```

```
ColumnName, DataType, MissingValues
InvoiceNo, object, 0
StockCode, object, 0
Description, object, 1454
Quantity, int64, 0
InvoiceDate, datetime64[ns], 0
UnitPrice, float64, 0
CustomerID, float64, 135080
Country, object, 0

Dataset Description:
            Quantity                    InvoiceDate      UnitPrice  \
count  541909.000000                         541909  541909.000000
mean        9.552250  2011-07-04 13:34:57.156386048       4.611114
min    -80995.000000            2010-12-01 08:26:00  -11062.060000
25%         1.000000            2011-03-28 11:34:00       1.250000
50%         3.000000            2011-07-19 17:17:00       2.080000
75%        10.000000            2011-10-19 11:27:00       4.130000
max     80995.000000            2011-12-09 12:50:00   38970.000000
```









Negative values scenario:

```
# Display unique Customer IDs for records with non-positive quantity and price
print("\nCustomer IDs for the above records:",
      df1.loc[(df1['Quantity'] <= 0) & (df1['UnitPrice'] <= 0), 'CustomerID'].unique())

# Calculate the percentage of records with negative quantity
percent_negative_quantity = df1[df1['Quantity'] < 0].shape[0] / df1.shape[0] * 100
print("\nPercentage of records with negative quantity: {:.2f}%".format(percent_negative_quantity))

# Examples of negative quantity transactions
print("\nExamples of transactions with negative quantities:")
display(df1[df1['Quantity'] < 0].head(3))

# Examples of transactions with negative prices
print("\nExamples of transactions with negative prices:")
display(df1[df1['UnitPrice'] < 0].head(3))
```

```
Are there records with both negative quantity and prices? = No

Number of records where either quantity is negative or price is zero (or vice-versa): 11805

Customer IDs for the above records: [nan]

Percentage of records with negative quantity: 1.96%

Examples of transactions with negative quantities:
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
141	C536379	D	Discount	-1	2010-12-01 09:41:00	27.50	14527.0	United Kingdom
154	C536383	35004C	SET OF 3 COLOURED FLYING DUCKS	-1	2010-12-01 09:49:00	4.65	15311.0	United Kingdom
235	C536391	22556	PLASTERS IN TIN CIRCUS PARADE	-12	2010-12-01 10:24:00	1.65	17548.0	United Kingdom

```
Examples of transactions with negative prices:
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
299983	A563186	B	Adjust bad debt	1	2011-08-12 14:51:00	-11062.06	NaN	United Kingdom
299984	A563187	B	Adjust bad debt	1	2011-08-12 14:52:00	-11062.06	NaN	United Kingdom

Fixing negative values

```
[8]: # Remove records without a CustomerID
df1 = df1.dropna(subset=['CustomerID'])

# Fixing negative Quantity and Zero Unit Price
df1 = df1[~(df1['Quantity'] < 0)]
df1 = df1[df1['UnitPrice'] > 0]

# Display the new size of the cleaned data
print('New Size of our Data:')
print(df1.shape)

def summarize_dataframe(df):
    # Number of observations
    obs = df.shape[0]

    # Create a DataFrame to hold the summary information
    summary = pd.DataFrame({
        "Type": df.dtypes,
        "Count": df.count(),
        "Uniques": df.apply(lambda x: len(x.unique())),
        "Nulls": df.isnull().sum(),
        "Distinct": df.nunique(),
        "Missing_ratio": (df.isnull().sum() / obs) * 100
    })

    return summary

# Generate summary
df1_summary = summarize_dataframe(df1)
print("\nDataFrame Summary:")
print(df1_summary)
```

```
New Size of our Data:
(397884, 8)

DataFrame Summary:
                       Type   Count  Uniques  Nulls  Distinct  Missing_ratio
InvoiceNo            object  397884    18532      0     18532            0.0
StockCode            object  397884     3665      0      3665            0.0
Description          object  397884     3877      0      3877            0.0
Quantity              int64  397884      301      0       301            0.0
InvoiceDate  datetime64[ns]  397884    17282      0     17282            0.0
UnitPrice           float64  397884      440      0       440            0.0
CustomerID          float64  397884     4338      0      4338            0.0
Country              object  397884       37      0        37            0.0
```

```
[ ]: '''
```

Fixing stock code and description

•The different descriptions associated with the same StockCode suggest inconsistencies or variations, in how the product is labeled or described in our dataset. These variations could indicate data quality issues such as duplicate entries where two or more descriptions refer to the same product but are written differently. To address these issues, we will consolidate similar descriptions into a single standardized format.

In [9]:
```
# Identify StockCodes with multiple descriptions
multi_desc_stockcodes = df1.groupby(["StockCode"])['Description'].nunique()
multi_desc_stockcodes = multi_desc_stockcodes[multi_desc_stockcodes > 1].index.tolist()

# Display one example of StockCode with multiple descriptions
if multi_desc_stockcodes:
    example_stockcode = multi_desc_stockcodes[0]
    unique_descriptions = df1[df1['StockCode'] == example_stockcode]['Description'].unique()

    print(f"StockCode {example_stockcode} has multiple descriptions:")
    print(unique_descriptions)
```

```
StockCode 20622 has multiple descriptions:
['VIPPASSPORT COVER ' 'VIP PASSPORT COVER ']
```

[11]:
```
#Fixing StockCode and Description discrepancy

from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())


unique_desc = df1[["StockCode", "Description"]].groupby(by=["StockCode"]).\
                apply(pd.DataFrame.mode).reset_index(drop=True)
q = '''
select df.InvoiceNo, df.StockCode, un.Description, df.Quantity, df.InvoiceDate,
       df.UnitPrice, df.CustomerID, df.Country
from df1 as df INNER JOIN
     unique_desc as un on df.StockCode = un.StockCode
'''

df1 = pysqldf(q)

# Generate summary
df1_summary = summarize_dataframe(df1)
print(df1_summary)
```

```
                Type   Count  Uniques  Nulls  Distinct  Missing_ratio
InvoiceNo      int64  397884    18532      0     18532            0.0
StockCode     object  397884     3665      0      3665            0.0
Description   object  397884     3647      0      3647            0.0
Quantity       int64  397884      301      0       301            0.0
InvoiceDate   object  397884    17282      0     17282            0.0
UnitPrice    float64  397884      440      0       440            0.0
CustomerID   float64  397884     4338      0      4338            0.0
Country       object  397884       37      0        37            0.0
```

Feature Engineering

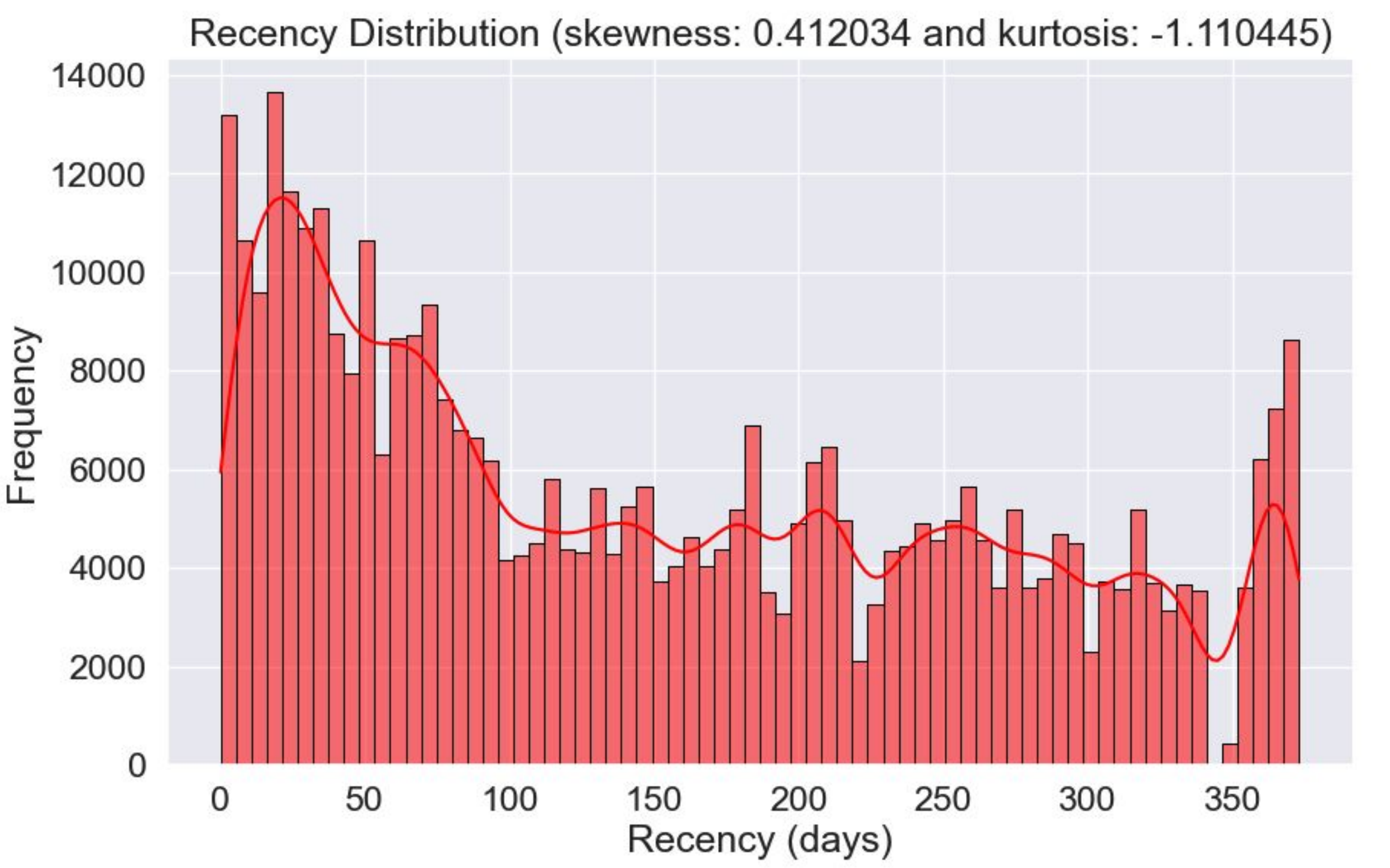
- Frequency: Number of purchases made by each customer
- Recency: Time since the last purchase
- Monetary Value: Total amount spent by each customer

Exploratory Data Analysis (EDA)

1. Visualize purchase frequency, recency, and monetary value distributions
2. Identify patterns and insights from the data
3. Check for correlations between features

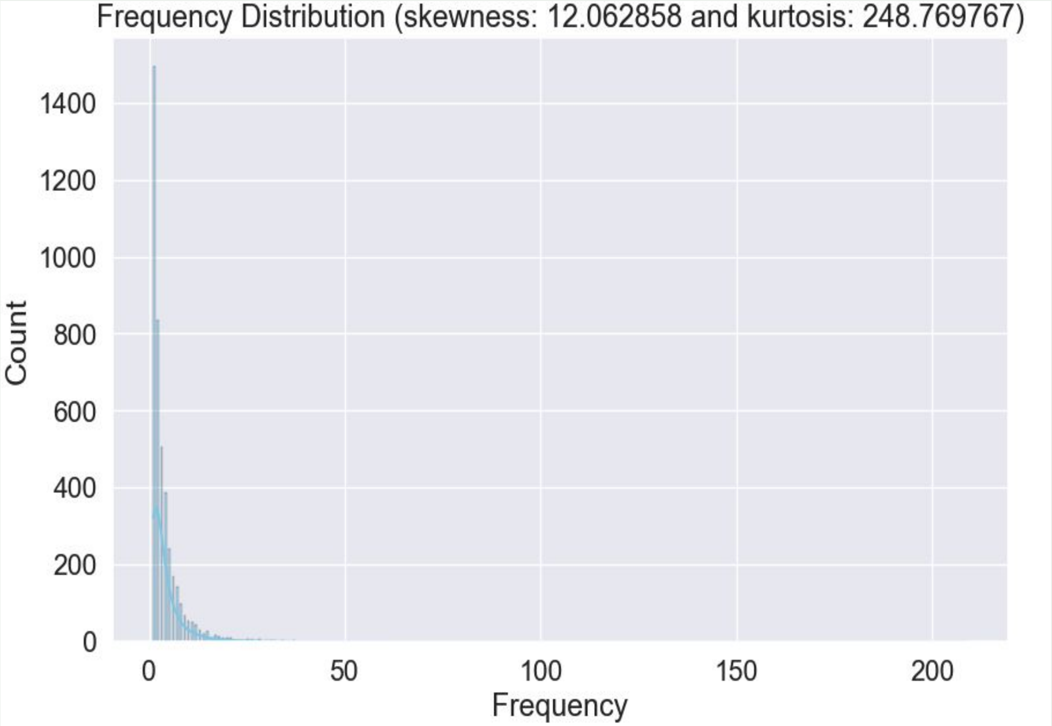
Recency

```
### 1. Recency

import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats

# Convert 'InvoiceDate' column to datetime format
df1['InvoiceDate'] = pd.to_datetime(df1['InvoiceDate'])

# Determine the reference date
reference_date = df1['InvoiceDate'].max()

# Calculate Recency
df1['Recency'] = (reference_date - df1['InvoiceDate']).dt.days
```



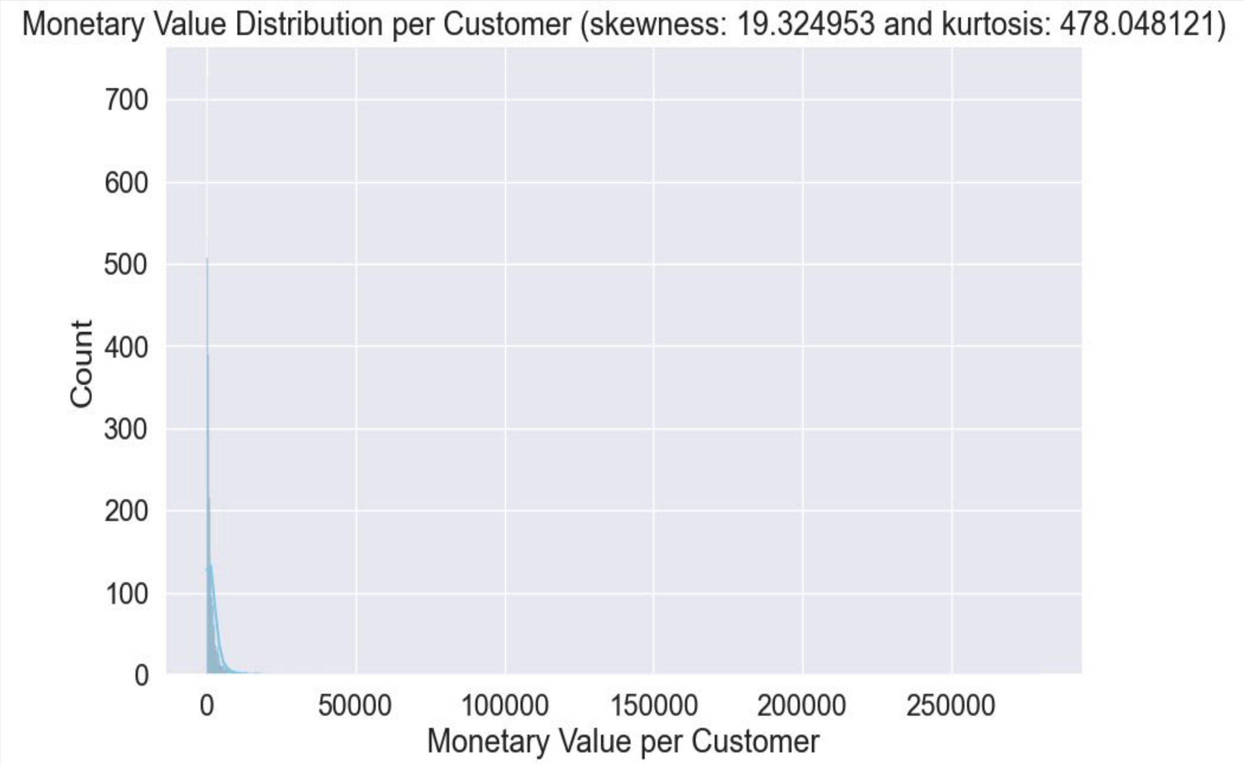
Frequency

```
### 2. Frequency

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats

# Calculate Frequency per Customer
frequency = df1.groupby('CustomerID')['InvoiceNo'].nunique().reset_index()
frequency.columns = ['CustomerID', 'Frequency']
```



Monetary value

```
### 3. Monetary Value

# Calculate Monetary Value per Transaction
df1['MonetaryValue'] = df1['Quantity'] * df1['UnitPrice']

# Calculate Total Monetary Value per Customer
monetary_value_per_customer = df1.groupby('CustomerID')['MonetaryValue'].sum().reset_index()
```



Model Selection

Clustering Algorithms:

K-means

Hierarchical

Gaussian Mixture Models

Density Based Spatial (DBSCAN)

Evaluation Metrics

SILHOUETTE SCORE: MEASURE THE SIMILARITY OF DATA POINTS WITHIN CLUSTERS

VISUAL INSPECTION: EVALUATE CLUSTER SEPARATION THROUGH VISUALIZATIONS LIKE SCATTER PLOTS OR DENDROGRAMS

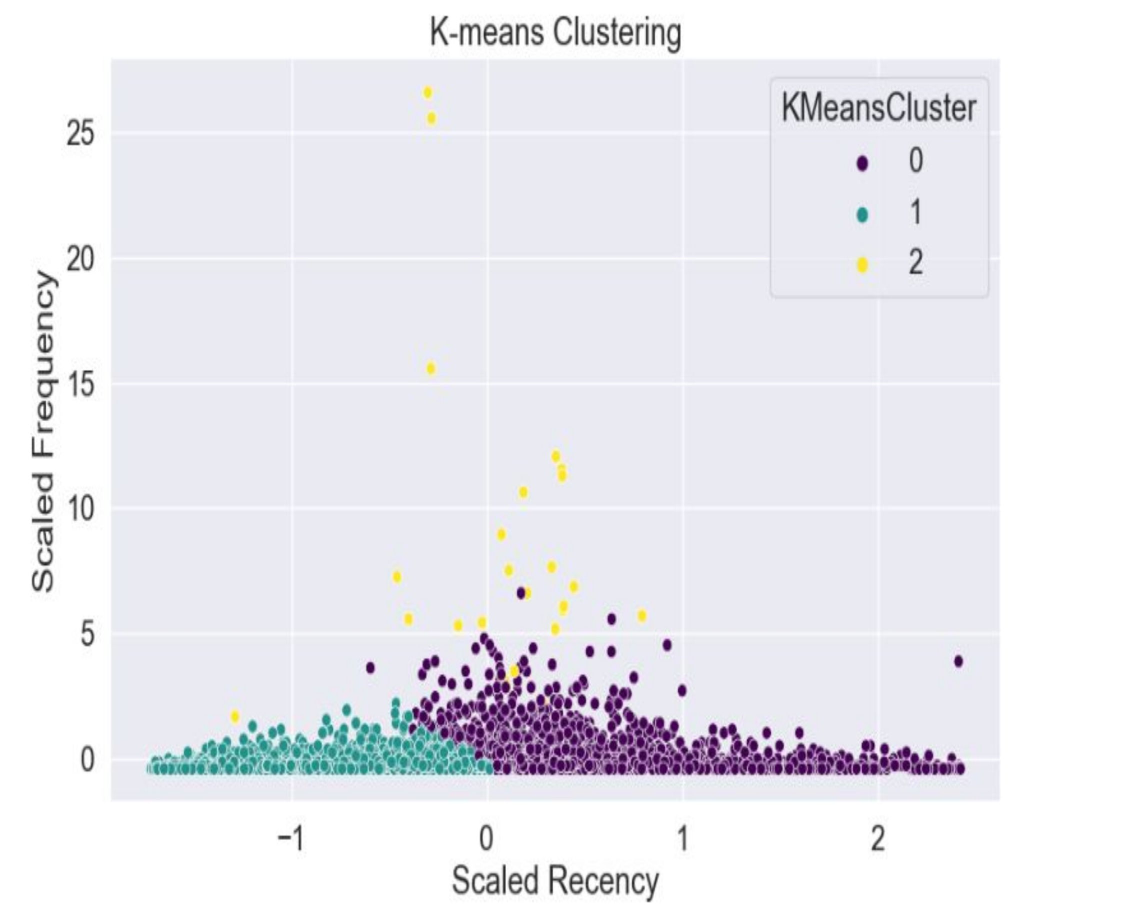
OPTIMAL NUMBER OF CLUSTERS: DETERMINE THE BEST NUMBER OF CLUSTERS FOR SEGMENTATION

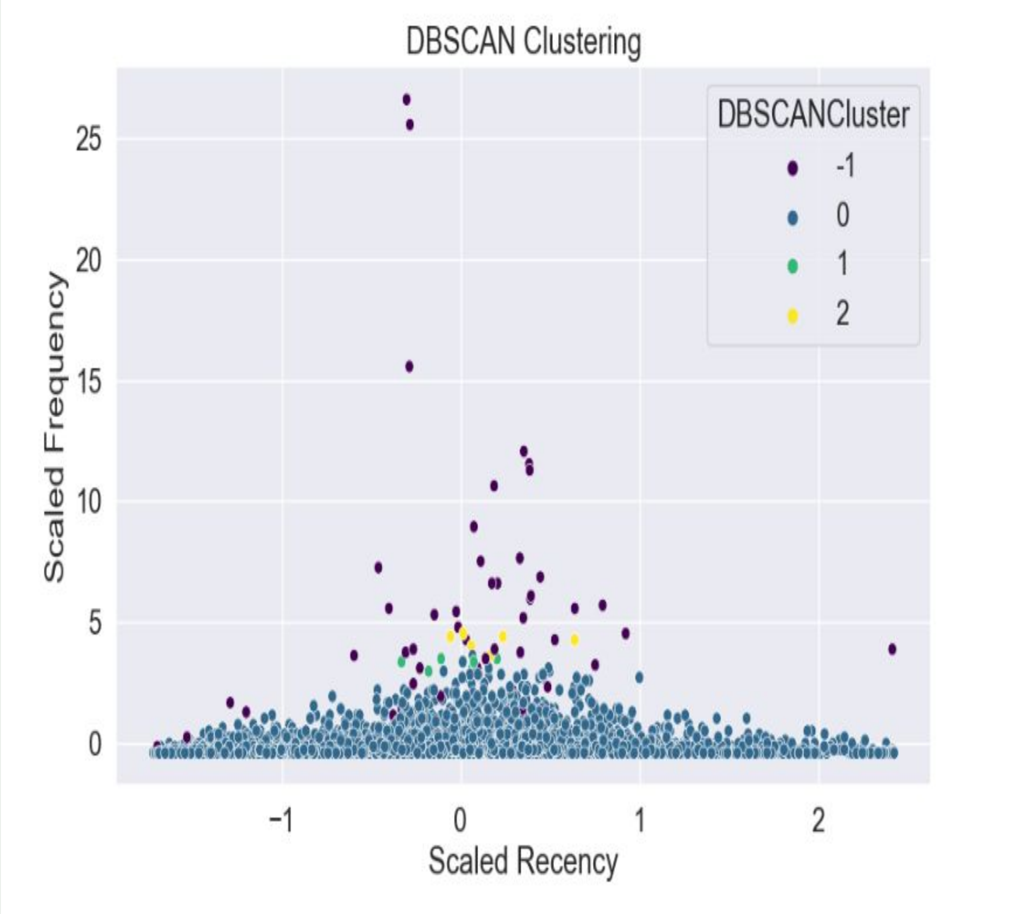
Evaluation Metrics against respective clusters

```
K-means Clustering:
Silhouette Score: 0.4430213985126717
Davies-Bouldin Index: 0.8192987971958634
Calinski-Harabasz Score: 2466.380074400315


Hierarchical Clustering:
Silhouette Score: 0.40831836189281423
Davies-Bouldin Index: 0.7003267015200608
Calinski-Harabasz Score: 1913.2309559374262


Gaussian Mixture Model:
Silhouette Score: 0.08245841517339414
Davies-Bouldin Index: 2.477241637207202
Calinski-Harabasz Score: 591.1298181626729


DBSCAN Clustering:
Silhouette Score for DBSCAN: 0.6197545420897675
```

```
#K-means Clustering: Moderate silhouette score, moderate Davies-Bouldin index, and high Calinski-Harabasz score;
#suggests balanced clusters.

#Hierarchical Clustering: Slightly low silhouette score, better Davies-Bouldin index, and seems moderate Calinski-Harabasz score;
#captures hierarchical structures but with less defined clusters than K-means.

#Gaussian Mixture Model (GMM): Low silhouette score,high Davies-Bouldin index,low Calinski-Harabasz score so not suitable to given dataset.
#DBSCAN Clustering: High silhouette score; seems effective in detecting clusters with varying shapes and sizes, includes outliers.
```

Overall K-Means seems good with balanced clusters





Results and Interpretation

Present	Present the clusters obtained from the selected model
Analyze	Analyze the characteristics of each customer segment
Discuss	Discuss the actionable insights provided by each segment

Cluster wise details

- Cluster 0.0: Occasional high spenders with infrequent purchases.
- Cluster 1.0: Regular but moderate spenders who purchase more frequently.
- Cluster 2.0: High-value customers who purchase very frequently, representing the most valuable segment.

```
for index, row in cluster_stats.iterrows():
    cluster = row['KMeansCluster']
    avg_purchase_amount = row['MonetaryValue']
    avg_recency = row['Recency']
    avg_frequency = row['Frequency']

    print(f"For Cluster {cluster}:")
    print(f"  - The average purchase amount is around ${avg_purchase_amount:.2f}")
    print(f"  - The recency is approximately {int(avg_recency)} days")
    print(f"  - The frequency is about {int(avg_frequency)}\n")
```

```
For Cluster 0.0:
  - The average purchase amount is around $2018.80
  - The recency is approximately 225 days
  - The frequency is about 4

For Cluster 1.0:
  - The average purchase amount is around $1008.06
  - The recency is approximately 78 days
  - The frequency is about 2

For Cluster 2.0:
  - The average purchase amount is around $85581.36
  - The recency is approximately 166 days
  - The frequency is about 64
```

```
#Cluster 0.0: Occasional high spenders with infrequent purchases.
#Cluster 1.0: Regular but moderate spenders who purchase more frequently.
#Cluster 2.0: High-value customers who purchase very frequently, representing the most valuable segment.
```

Overall Cluster details

Cluster Sizes:

- Cluster 0: 2231 customers
- Cluster 1: 2080 customers
- Cluster 2: 27 customers

Cluster Profiles:

- Cluster 0: Less frequent purchases, average spend $2018.80
- Cluster 1: More recent purchases, average spend $1008.06
- Cluster 2: Very frequent purchases, high average spend $85581.36

Average Order Value (AOV) per Cluster:

- Cluster 0: $370.33
- Cluster 1: $378.56
- Cluster 2: $7582.40'''

```
Cluster Sizes:
KMeansCluster
0    2231
1    2080
2      27
Name: count, dtype: int64
--------------------------------------------------
Cluster Profiles:
              Recency                                  Frequency                    \
                 mean  median     min    max    std       mean median min  max
KMeansCluster
0              225.19  209.74  101.10  373.0  61.14       4.98    3.0   1   55
1               78.92   76.25    0.00  158.0  42.58       2.73    2.0   1   21
2              166.51  167.29   38.78  325.0  48.25      64.00   51.0   1  209

                    MonetaryValue
                std          mean    median       min         max       std
KMeansCluster
0              5.93       2018.80    806.41      3.75    50491.81   3726.24
1              2.38       1008.06    598.88      6.20    16569.50   1280.53
2             49.79      85581.36  60767.90  11189.91   280206.02  69457.26
--------------------------------------------------
Average Order Value (AOV) per Cluster:
KMeansCluster
0     370.33
1     378.56
2    7582.40
Name: AverageOrderValue, dtype: float64
```

Conclusion

<u>Application :</u>

Cluster 2: Very frequent purchases.Then actions must be taken to raise their frequency and reduce the chances of them migrating to cluster 0 by staying longer without purchasing products.

Cluster 0: Less frequent purchases and a reasonable frequency, but this is a long time without buying. This group should be sensible to promotions and activations so that they do not get lost and make their next purchase.

Further Exploration

- Cross-selling: By examining a customer's past purchases as well as general trends and patterns that coincide with the customer's purchasing habits, cross-selling experts can offer more products to them. These suggested products would almost certainly be highly tempting.
- New metrics depending on the date of the client's first purchase, like customer relationship time and whether the customer is from a foreign region or not.
- External data providers, utilize it, and so forth.

Thank You