In this project, we're delving into the online retail sector using a transactional dataset from a UK-based retailer (UCI Machine Learning Repository, 2010-2011). Our main aim is to optimize marketing and boost sales through customer segmentation. By transforming transactional data into a customer-centric dataset and employing K-means clustering, we'll identify distinct customer groups with unique profiles and preferences. Building on this segmentation, we'll develop a recommendation system to suggest top-selling products to customers in each segment who haven't purchased them yet, ultimately enhancing marketing effectiveness and driving increased sales.

STEP-1: SETUP AND INITIALIZATION

```python
#Importing all the necessary modules and libraries
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import plotly.graph_objects as go
from matplotlib.colors import LinearSegmentedColormap
from matplotlib import colors as mcolors
from scipy.stats import linregress
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from sklearn.metrics import silhouette_score, calinski_harabasz_score,
davies_bouldin_score
from sklearn.cluster import KMeans
from tabulate import tabulate
from collections import Counter

%matplotlib inline

# Initialize Plotly for use in the notebook
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)

sns.set(rc={'axes.facecolor': '#fcf0dc'}, style='darkgrid')
```

STEP-2: LOADING THE DATASET

```python
df = pd.read_csv('/kaggle/input/ecommerce-data/data.csv',
encoding="ISO-8859-1")
```

Variable	Description
InvoiceNo	Code representing each unique transaction. If

Variable	Description
	this code starts with letter 'c', it indicates a cancellation.
StockCode	Code uniquely assigned to each distinct product.
Description	Description of each product.
Quantity	The number of units of a product in a transaction.
InvoiceDate	The date and time of the transaction.
UnitPrice	The unit price of the product in sterling.
CustomerID	Identifier uniquely assigned to each customer.
Country	The country of the customer.

PRELIMINARY ANALYSIS

```
df.head(20)
```

```
  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   

5    536365     22752         SET 7 BABUSHKA NESTING BOXES         2   

6    536365     21730    GLASS STAR FROSTED T-LIGHT HOLDER         6   

7    536366     22633               HAND WARMER UNION JACK         6   

8    536366     22632            HAND WARMER RED POLKA DOT         6   

9    536367     84879        ASSORTED COLOUR BIRD ORNAMENT        32   

10   536367     22745           POPPY'S PLAYHOUSE BEDROOM          6   

11   536367     22748            POPPY'S PLAYHOUSE KITCHEN         6   

12   536367     22749    FELTCRAFT PRINCESS CHARLOTTE DOLL         8   

13   536367     22310              IVORY KNITTED MUG COSY          6   
```

```
14    536367     84969   BOX OF 6 ASSORTED COLOUR TEASPOONS         6

15    536367     22623        BOX OF VINTAGE JIGSAW BLOCKS          3

16    536367     22622       BOX OF VINTAGE ALPHABET BLOCKS         2

17    536367     21754             HOME BUILDING BLOCK WORD         3

18    536367     21755             LOVE BUILDING BLOCK WORD         3

19    536367     21777          RECIPE BOX WITH METAL HEART         4


       InvoiceDate  UnitPrice  CustomerID         Country
0   12/1/2010 8:26       2.55     17850.0  United Kingdom
1   12/1/2010 8:26       3.39     17850.0  United Kingdom
2   12/1/2010 8:26       2.75     17850.0  United Kingdom
3   12/1/2010 8:26       3.39     17850.0  United Kingdom
4   12/1/2010 8:26       3.39     17850.0  United Kingdom
5   12/1/2010 8:26       7.65     17850.0  United Kingdom
6   12/1/2010 8:26       4.25     17850.0  United Kingdom
7   12/1/2010 8:28       1.85     17850.0  United Kingdom
8   12/1/2010 8:28       1.85     17850.0  United Kingdom
9   12/1/2010 8:34       1.69     13047.0  United Kingdom
10  12/1/2010 8:34       2.10     13047.0  United Kingdom
11  12/1/2010 8:34       2.10     13047.0  United Kingdom
12  12/1/2010 8:34       3.75     13047.0  United Kingdom
13  12/1/2010 8:34       1.65     13047.0  United Kingdom
14  12/1/2010 8:34       4.25     13047.0  United Kingdom
15  12/1/2010 8:34       4.95     13047.0  United Kingdom
16  12/1/2010 8:34       9.95     13047.0  United Kingdom
17  12/1/2010 8:34       5.95     13047.0  United Kingdom
18  12/1/2010 8:34       5.95     13047.0  United Kingdom
19  12/1/2010 8:34       7.95     13047.0  United Kingdom

df.info()
#rows and columns: 541909 x 8

<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
```

Upon initial examination, it appears there are missing values in the Description and CustomerID columns that require attention. The InvoiceDate column is already formatted as datetime, simplifying subsequent time series analysis. Additionally, it's evident that a single customer can engage in multiple transactions, as indicated by the recurrence of CustomerID in the initial rows

SUMMARY STATISTICS

```
df.describe().T

               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

df.describe(include='object').T

             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                    10/31/2011 14:41    1114
Country      541909     38                      United Kingdom  495478
```

Quantity:

- Average: 9.55
- Range: -80995 to 80995 (negative values shows cancelled orders)(includes returns/cancellations)
- Large standard deviation indicates data spread and outliers.

UnitPrice:

- Average: 4.61
- Range: -11062.06 to 38970 (includes errors/noise)
- Presence of outliers indicated by a large difference from the 75th percentile.

CustomerID:

- 406,829 non-null entries, addressing missing values is necessary.
- Range: 12346 to 18287, identifying unique customers.

InvoiceNo:

- 25,900 unique invoices, with the most frequent (573585) appearing 1114 times.

StockCode:

- 4,070 unique stock codes, with the most frequent (85123A) appearing 2313 times.

Description:

- 4,223 unique product descriptions.
- Most frequent: "WHITE HANGING HEART T-LIGHT HOLDER" (2369 times).
- Missing values need attention.

Country:

- Transactions from 38 countries, with around 91.4% from the United Kingdom.

```python
# What is the time period covered by this dataset?
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
start_date = df['InvoiceDate'].min()
end_date = df['InvoiceDate'].max()
time_period = end_date - start_date
print("Start Date:", start_date)
print("End Date:", end_date)
print("Time Period:", time_period)
```

```
Start Date: 2010-12-01 08:26:00
End Date: 2011-12-09 12:50:00
Time Period: 373 days 04:24:00
```

```python
# Filter out the rows with InvoiceNo starting with "C" and create a
new column indicating the transaction status
df['Transaction_Status'] =
np.where(df['InvoiceNo'].astype(str).str.startswith('C'), 'Cancelled',
'Completed')

# Analyze the characteristics of these rows (considering the new
column)
cancelled_transactions = df[df['Transaction_Status'] == 'Cancelled']
cancelled_transactions.describe().drop('CustomerID', axis=1)
```

	Quantity	InvoiceDate	UnitPrice
count	9288.000000	9288	9288.000000
mean	-29.885228	2011-06-26 03:42:05.943152640	48.393661
min	-80995.000000	2010-12-01 09:41:00	0.010000
25%	-6.000000	2011-03-21 16:15:00	1.450000
50%	-2.000000	2011-07-07 17:33:30	2.950000

```
75%       -1.000000           2011-10-06 20:36:00      5.950000
max       -1.000000           2011-12-09 11:58:00  38970.000000
std     1145.786965                           NaN    666.600430
```

Customer Analysis

```
#How many unique customers are there in the dataset?
unique_customers = df['CustomerID'].nunique()
print("Number of unique customers:", unique_customers)

Number of unique customers: 4372

#What is the distribution of the number of orders per customer?
orders_per_customer = df.groupby('CustomerID')['InvoiceNo'].nunique()
print("Distribution of orders per customer:")
print(orders_per_customer.describe())

Distribution of orders per customer:
count    4372.000000
mean        5.075480
std         9.338754
min         1.000000
25%         1.000000
50%         3.000000
75%         5.000000
max       248.000000
Name: InvoiceNo, dtype: float64

#Can you identify the top 5 customers who have made the most purchases
by order count?
top_customers = df.groupby('CustomerID')
['InvoiceNo'].nunique().sort_values(ascending=False).head(5)
print("Top 5 customers with the most purchases:")
print(top_customers)

Top 5 customers with the most purchases:
CustomerID
14911.0    248
12748.0    224
17841.0    169
14606.0    128
13089.0    118
Name: InvoiceNo, dtype: int64
```

Returns and Refunds

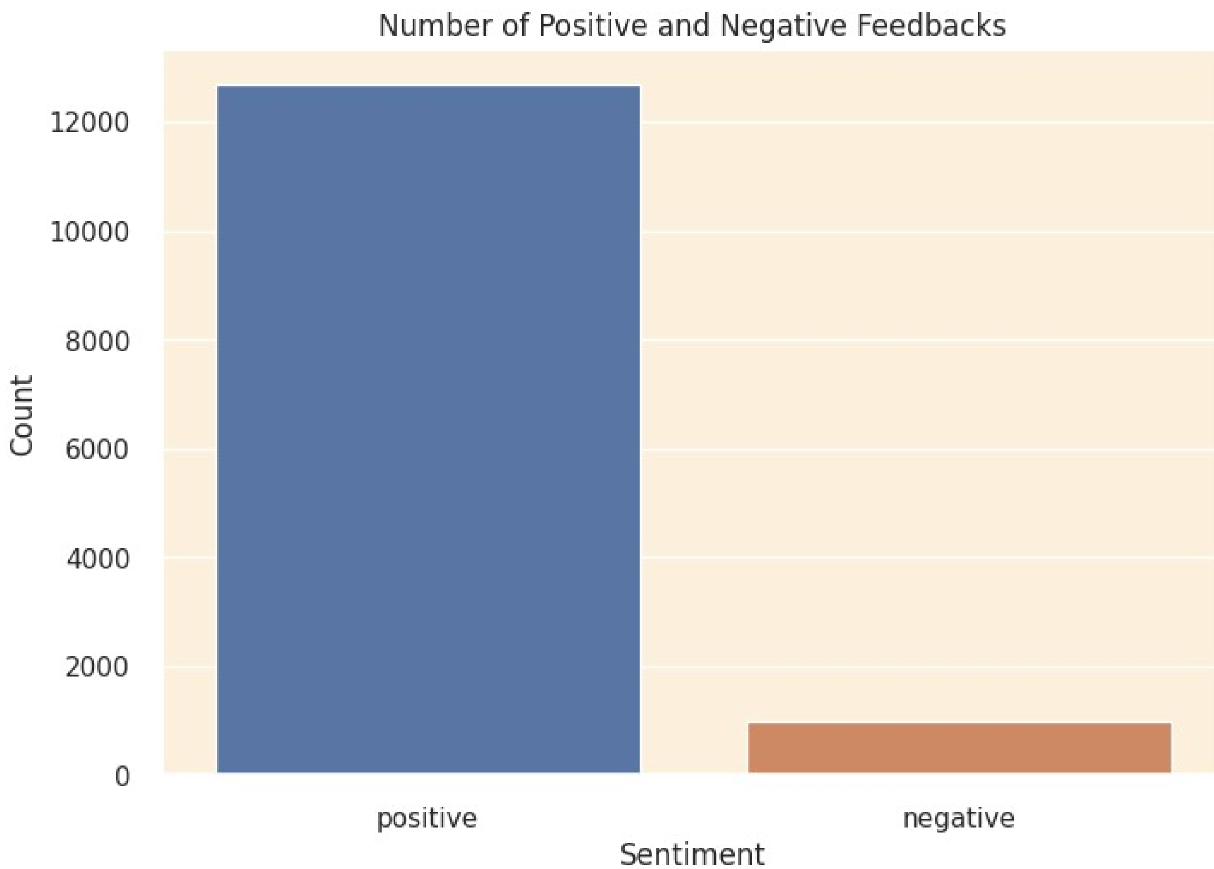
```
df['Cancelled'] = (df['Quantity'] < 0).astype(int)
df.head()
```

```
  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

  Transaction_Status  Cancelled
0          Completed          0
1          Completed          0
2          Completed          0
3          Completed          0
4          Completed          0

# Group by product description and calculate cancellation rates
cancellation_rates = df.groupby('Description')
['Cancelled'].mean().reset_index()

cancellation_rates

                             Description  Cancelled
0      4 PURPLE FLOCK DINNER CANDLES   0.000000
1      50'S CHRISTMAS GIFT BAG LARGE   0.007692
2                  DOLLY GIRL BEAKER   0.011050
3        I LOVE LONDON MINI BACKPACK   0.000000
4        I LOVE LONDON MINI RUCKSACK   0.000000
...                                ...        ...
4218     wrongly marked carton 22804   1.000000
4219    wrongly marked. 23343 in box   1.000000
4220    wrongly sold (22719) barcode   0.000000
4221             wrongly sold as sets   1.000000
4222                wrongly sold sets   1.000000

[4223 rows x 2 columns]
```

SENTIMENT ANALYSIS

As I do not have the feedback data so what I am doing is if there are more than 2 cancelled products than the feedback is negative else the feedback is positive and create this kind of data to do sentiment analysis

```
# Assuming you have a column 'transaction_status' in your dataframe

# Create a new dataset for sentiment analysis
sentiment_data = df.groupby(['InvoiceDate','CustomerID',
'Transaction_Status']).size().unstack(fill_value=0)

# Calculate the total count of cancelled products for each customer
sentiment_data['total_cancelled'] = sentiment_data['Cancelled']

# Label customers based on the count of cancelled products
sentiment_data['sentiment'] =
np.where(sentiment_data['total_cancelled'] > 2, 'negative',
'positive')

# Extract relevant columns for the sentiment analysis dataset
sentiment_analysis_dataset = sentiment_data[['total_cancelled',
'sentiment']].reset_index()

# Display a sample of the new dataset
print(sentiment_analysis_dataset.head())

Transaction_Status         InvoiceDate  CustomerID  total_cancelled
sentiment
0                  2010-12-01 08:26:00     17850.0                0
positive
1                  2010-12-01 08:28:00     17850.0                0
positive
2                  2010-12-01 08:34:00     13047.0                0
positive
3                  2010-12-01 08:35:00     13047.0                0
positive
4                  2010-12-01 08:45:00     12583.0                0
positive

# Assuming you have 'sentiment' column in sentiment_analysis_by_month
dataset

# Count the number of positive and negative feedbacks
sentiment_counts =
sentiment_analysis_by_month['sentiment'].value_counts()

# Plot the bar chart
sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values)
plt.title('Number of Positive and Negative Feedbacks')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```




```
print(sentiment_analysis_dataset.head())

Transaction_Status         InvoiceDate  CustomerID  total_cancelled
sentiment
0                  2010-12-01 08:26:00     17850.0                0
positive
1                  2010-12-01 08:28:00     17850.0                0
positive
2                  2010-12-01 08:34:00     13047.0                0
positive
3                  2010-12-01 08:35:00     13047.0                0
positive
4                  2010-12-01 08:45:00     12583.0                0
positive

print(sentiment_analysis_dataset.dtypes)

Transaction_Status
InvoiceDate        datetime64[ns]
CustomerID                float64
total_cancelled             int64
sentiment                  object
dtype: object
```

```
sentiment_analysis_dataset['MonthYear'] =
sentiment_analysis_dataset['InvoiceDate'].dt.to_period('M')
sentiment_analysis_dataset
```

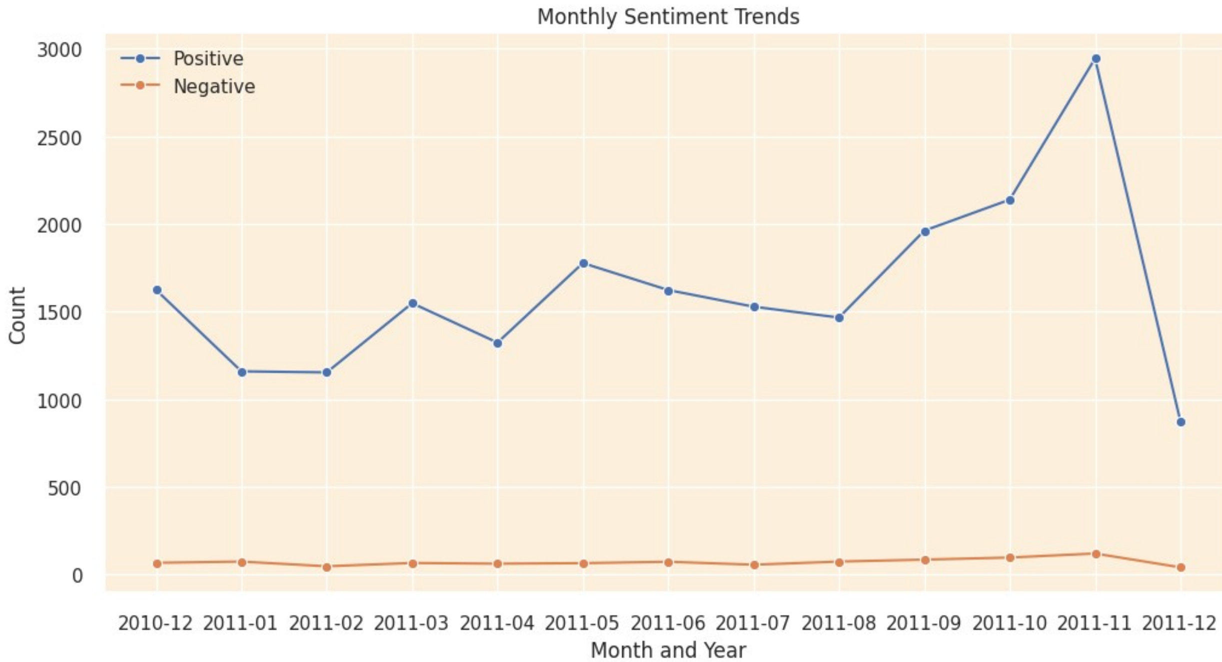
```
Transaction_Status         InvoiceDate  CustomerID  total_cancelled
sentiment  \
0                  2010-12-01 08:26:00     17850.0                0
positive
1                  2010-12-01 08:28:00     17850.0                0
positive
2                  2010-12-01 08:34:00     13047.0                0
positive
3                  2010-12-01 08:35:00     13047.0                0
positive
4                  2010-12-01 08:45:00     12583.0                0
positive
...                                ...         ...              ...
...
22029              2011-12-09 12:23:00     13777.0                0
positive
22030              2011-12-09 12:25:00     13777.0                0
positive
22031              2011-12-09 12:31:00     15804.0                0
positive
22032              2011-12-09 12:49:00     13113.0                0
positive
22033              2011-12-09 12:50:00     12680.0                0
positive

Transaction_Status MonthYear
0                    2010-12
1                    2010-12
2                    2010-12
3                    2010-12
4                    2010-12
...                      ...
22029                2011-12
22030                2011-12
22031                2011-12
22032                2011-12
22033                2011-12

[22034 rows x 5 columns]
```

```
print(sentiment_analysis_dataset.dtypes)
```

```
Transaction_Status
InvoiceDate        datetime64[ns]
CustomerID                float64
total_cancelled             int64
```

```
sentiment                          object
MonthYear                       period[M]
dtype: object
```

```python
# Assuming 'MonthYear' is already of type Period and 'positive' and
'negative' are numeric columns

# Group data by Month and Year
monthly_sentiment_data =
sentiment_analysis_dataset.groupby(['MonthYear',
'sentiment']).size().unstack(fill_value=0)

# Visualize Trends (line chart)
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

# Create a line chart
sns.lineplot(data=monthly_sentiment_data,
x=monthly_sentiment_data.index.astype(str), y='positive',
label='Positive', marker='o')
sns.lineplot(data=monthly_sentiment_data,
x=monthly_sentiment_data.index.astype(str), y='negative',
label='Negative', marker='o')

plt.title('Monthly Sentiment Trends')
plt.xlabel('Month and Year')
plt.ylabel('Count')
plt.legend()
plt.show()
```




```
# Extract Month and Year from InvoiceDate in df
df['Month'] = df['InvoiceDate'].dt.to_period('M')
df
```

```
       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      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
...                    ...        ...         ...             ...
541904 2011-12-09 12:50:00       0.85     12680.0          France
541905 2011-12-09 12:50:00       2.10     12680.0          France
541906 2011-12-09 12:50:00       4.15     12680.0          France
541907 2011-12-09 12:50:00       4.15     12680.0          France
541908 2011-12-09 12:50:00       4.95     12680.0          France

       Transaction_Status  Cancelled    Month
0               Completed          0  2010-12
1               Completed          0  2010-12
2               Completed          0  2010-12
3               Completed          0  2010-12
4               Completed          0  2010-12
...                   ...        ...      ...
541904          Completed          0  2011-12
541905          Completed          0  2011-12
541906          Completed          0  2011-12
541907          Completed          0  2011-12
541908          Completed          0  2011-12

[541909 rows x 11 columns]
```

sentiment_analysis_dataset

```
Transaction_Status         InvoiceDate  CustomerID  total_cancelled
sentiment  \
0                  2010-12-01 08:26:00     17850.0                0
positive
1                  2010-12-01 08:28:00     17850.0                0
positive
2                  2010-12-01 08:34:00     13047.0                0
positive
3                  2010-12-01 08:35:00     13047.0                0
positive
4                  2010-12-01 08:45:00     12583.0                0
positive
...                                ...         ...              ...
...
22029              2011-12-09 12:23:00     13777.0                0
positive
22030              2011-12-09 12:25:00     13777.0                0
positive
22031              2011-12-09 12:31:00     15804.0                0
positive
```

```
22032                2011-12-09 12:49:00     13113.0                0
positive
22033                2011-12-09 12:50:00     12680.0                0
positive

Transaction_Status MonthYear
0                    2010-12
1                    2010-12
2                    2010-12
3                    2010-12
4                    2010-12
...                      ...
22029                2011-12
22030                2011-12
22031                2011-12
22032                2011-12
22033                2011-12

[22034 rows x 5 columns]
```

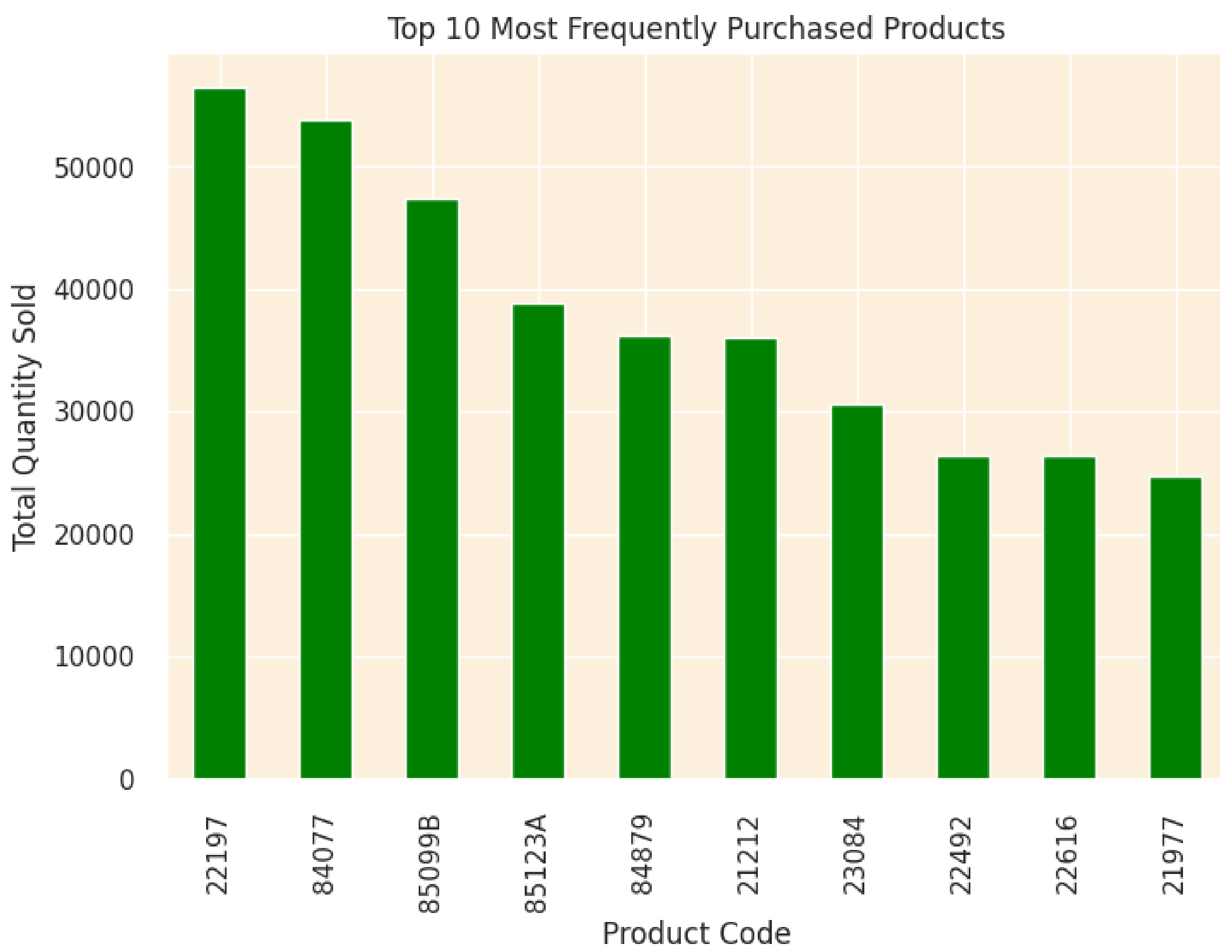
Product Analysis

```
#What are the top 10 most frequently purchased products?
top_products = df.groupby('StockCode')
['Quantity'].sum().sort_values(ascending=False).head(10)
print("Top 10 most frequently purchased products:")
print(top_products)

Top 10 most frequently purchased products:
StockCode
22197     56450
84077     53847
85099B    47363
85123A    38830
84879     36221
21212     36039
23084     30646
22492     26437
22616     26315
21977     24753
Name: Quantity, dtype: int64

#What is the average price of products in the dataset?
average_price = df['UnitPrice'].mean()
print("Average price of products:", average_price)

Average price of products: 4.611113626088513

#Can you find out which product category generates the highest
revenue?
```

```
df['TotalRevenue'] = df['Quantity'] * df['UnitPrice']
highest_revenue_category = df.groupby('StockCode')
['TotalRevenue'].sum().sort_values(ascending=False).idxmax()
print("Product category generating the highest revenue:",
highest_revenue_category)
```

```
Product category generating the highest revenue: DOT
```

```
#What are the top 10 most frequently purchased products?
import matplotlib.pyplot as plt

# Group by StockCode, sum quantities, and get the top 10 products
top_products = df.groupby('StockCode')
['Quantity'].sum().sort_values(ascending=False).head(10)

# Plotting
top_products.plot(kind='bar', color='green')
plt.title('Top 10 Most Frequently Purchased Products')
plt.xlabel('Product Code')
plt.ylabel('Total Quantity Sold')
plt.show()
```



Time Analysis

```
# Is there a specific day of the week or time of day when most orders
are placed?
# Extract day of the week and hour from the InvoiceDate
df['DayOfWeek'] = df['InvoiceDate'].dt.dayofweek
df['HourOfDay'] = df['InvoiceDate'].dt.hour

# Count the number of orders for each day of the week and hour
most_orders_day = df['DayOfWeek'].value_counts().idxmax()
most_orders_hour = df['HourOfDay'].value_counts().idxmax()

print("Day of the week with the most orders:", most_orders_day)
print("Hour of the day with the most orders:", most_orders_hour)
```

```
Day of the week with the most orders: 3
Hour of the day with the most orders: 12
```

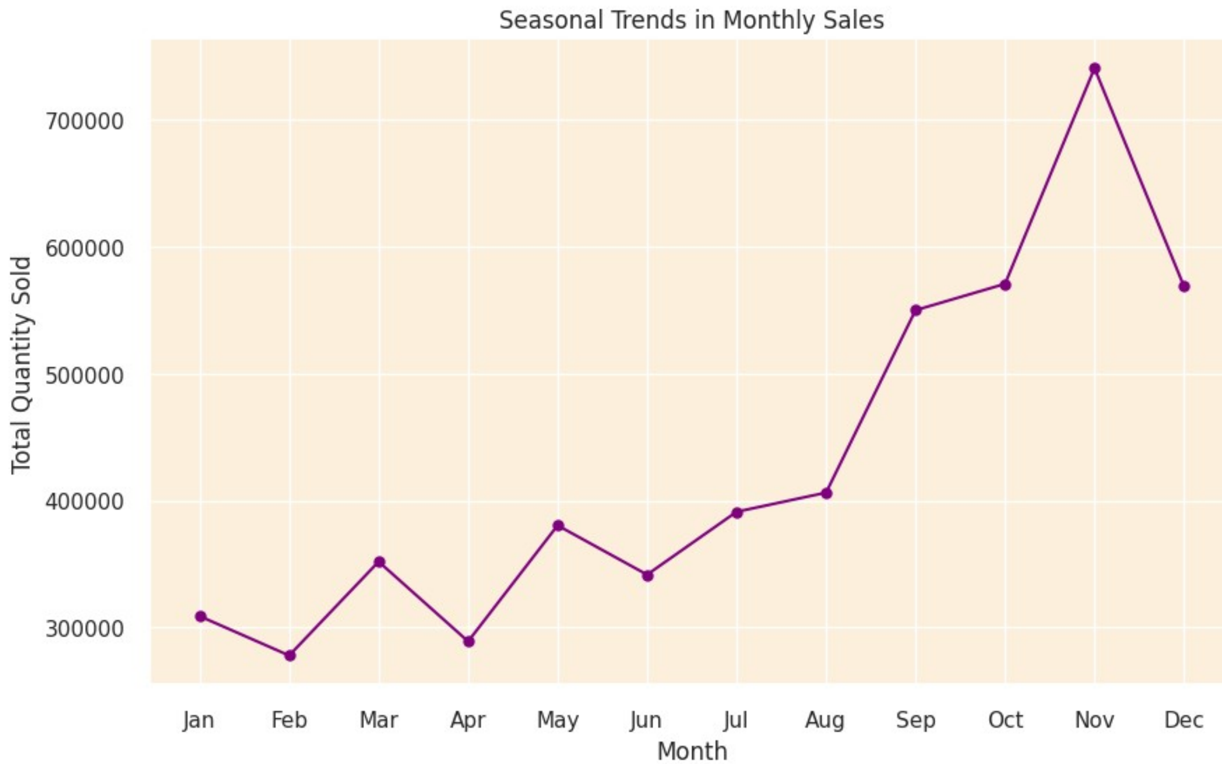
```
# What is the average order processing time?
# Calculate the time difference between InvoiceDate and InvoiceDate of
the previous order
df['OrderProcessingTime'] = df.groupby('CustomerID')
['InvoiceDate'].diff()

# Calculate the average order processing time
average_processing_time = df['OrderProcessingTime'].mean()
print("Average order processing time:", average_processing_time)
```

```
Average order processing time: 1 days 10:51:53.479651242
```

```
# Are there any seasonal trends in the dataset?
# Extract month and quarter from the InvoiceDate
df['Month'] = df['InvoiceDate'].dt.month
df['Quarter'] = df['InvoiceDate'].dt.quarter

# Plot monthly and quarterly sales
import matplotlib.pyplot as plt

monthly_sales = df.groupby('Month')['Quantity'].sum()
quarterly_sales = df.groupby('Quarter')['Quantity'].sum()

# Plotting
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plt.plot(monthly_sales, marker='o')
plt.title('Monthly Sales')
plt.xlabel('Month')
plt.ylabel('Quantity')

plt.subplot(2, 1, 2)
```

```
plt.plot(quarterly_sales, marker='o')
plt.title('Quarterly Sales')
plt.xlabel('Quarter')
plt.ylabel('Quantity')

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


```
import matplotlib.pyplot as plt

# Extract month from the InvoiceDate
df['Month'] = df['InvoiceDate'].dt.month

# Plotting
monthly_sales = df.groupby('Month')['Quantity'].sum()

plt.figure(figsize=(10, 6))
plt.plot(monthly_sales, marker='o', color='purple')
plt.title('Seasonal Trends in Monthly Sales')
plt.xlabel('Month')
plt.ylabel('Total Quantity Sold')
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.show()
```

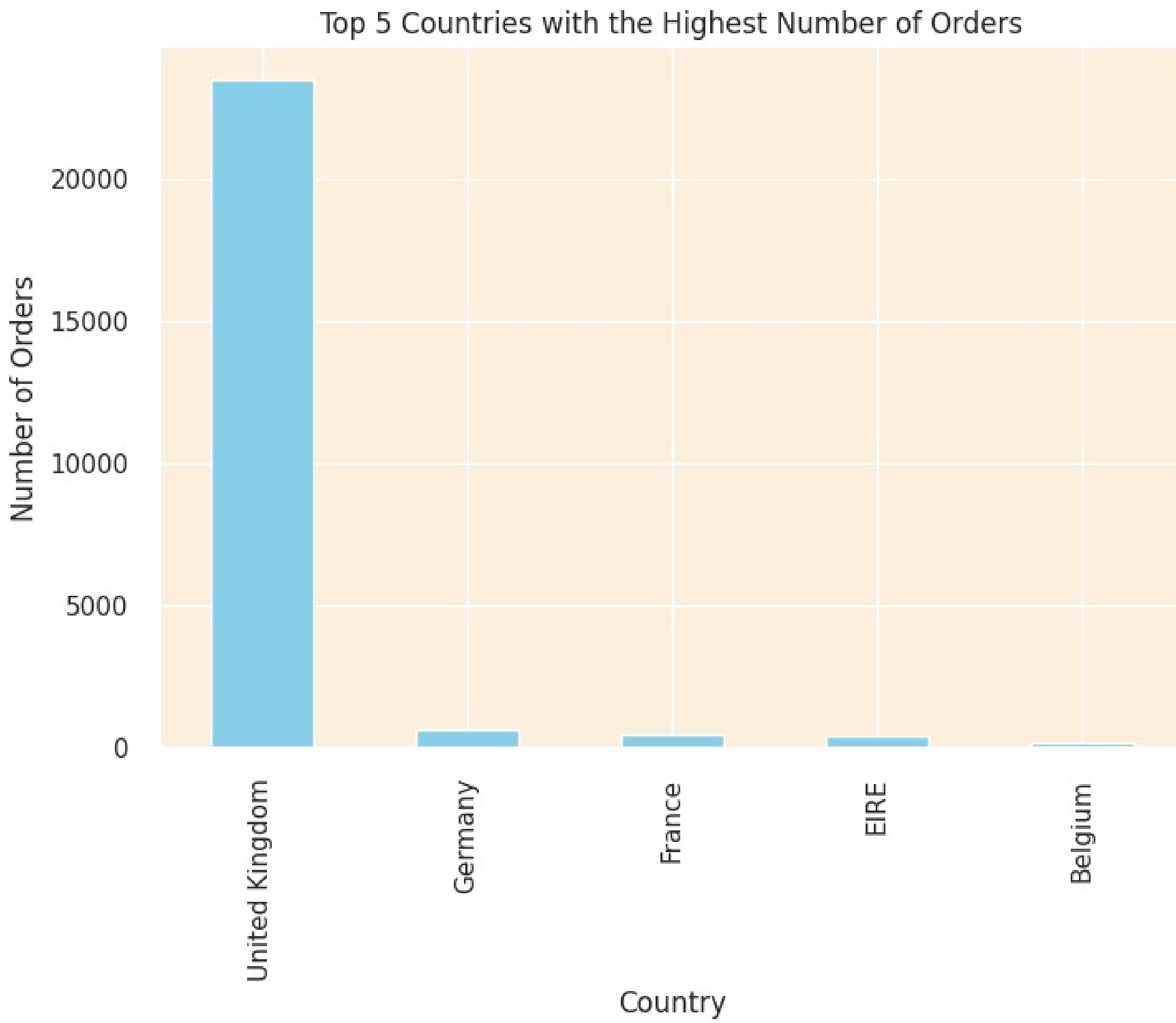


So as we can see that the amount of quanity sold in November is highest, mostly due to the Black friday deals and festivals like Christmas, Hallowen etc

Geographical Analysis

```
# Can you determine the top 5 countries with the highest number of
orders?
top_countries = df.groupby('Country')
['InvoiceNo'].nunique().sort_values(ascending=False).head(5)
print("Top 5 countries with the highest number of orders:")
print(top_countries)

Top 5 countries with the highest number of orders:
Country
United Kingdom    23494
Germany             603
France              461
EIRE                360
Belgium             119
Name: InvoiceNo, dtype: int64

import matplotlib.pyplot as plt

# Plotting
top_countries.plot(kind='bar', color='skyblue')
plt.title('Top 5 Countries with the Highest Number of Orders')
plt.xlabel('Country')
```

```
plt.ylabel('Number of Orders')
plt.show()
```



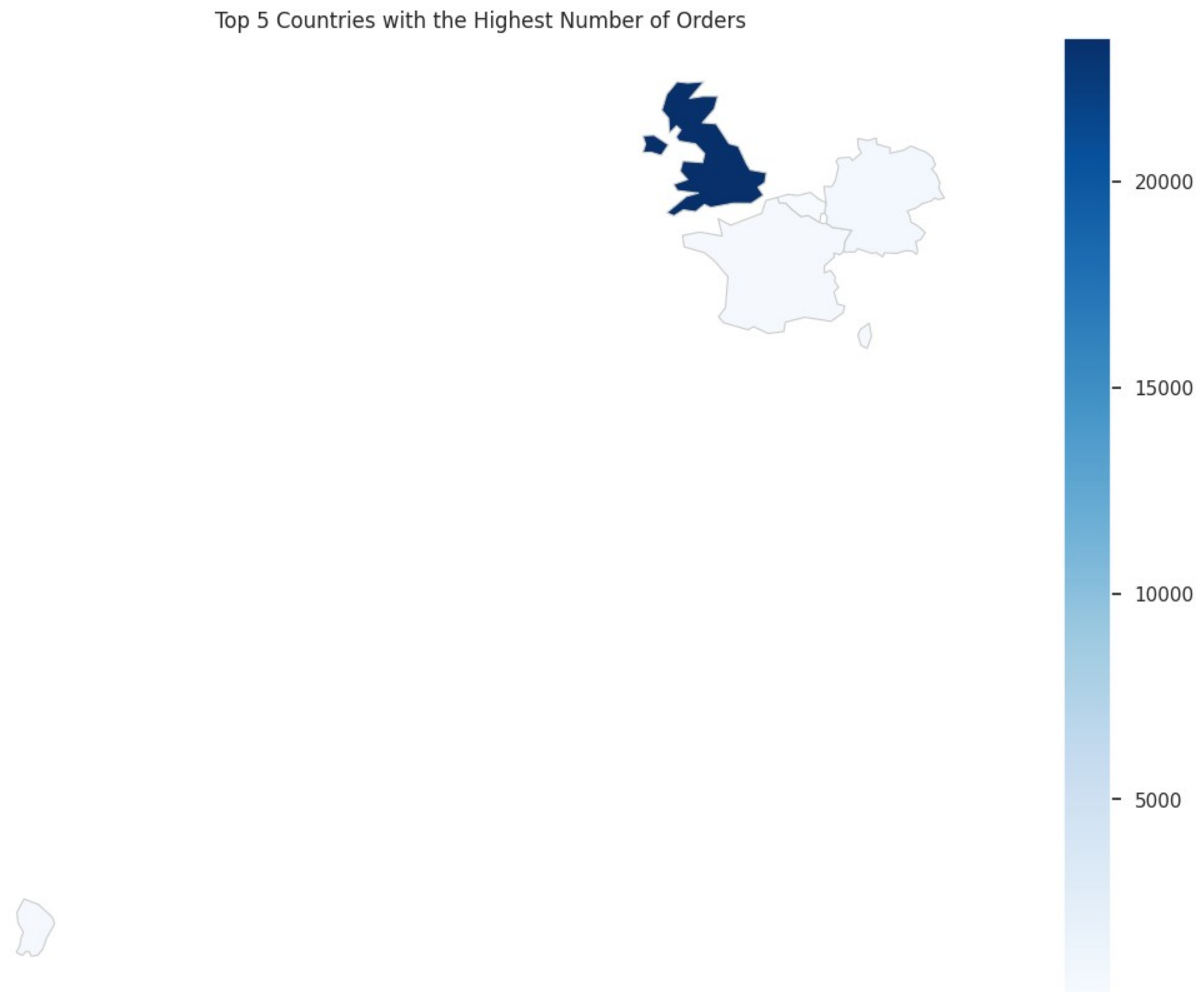

```
!pip install geopandas matplotlib
```

```
Requirement already satisfied: geopandas in
/opt/conda/lib/python3.10/site-packages (0.14.0)
Requirement already satisfied: matplotlib in
/opt/conda/lib/python3.10/site-packages (3.7.3)
Requirement already satisfied: fiona>=1.8.21 in
/opt/conda/lib/python3.10/site-packages (from geopandas) (1.9.5)
Requirement already satisfied: packaging in
/opt/conda/lib/python3.10/site-packages (from geopandas) (21.3)
Requirement already satisfied: pandas>=1.4.0 in
/opt/conda/lib/python3.10/site-packages (from geopandas) (2.0.3)
Requirement already satisfied: pyproj>=3.3.0 in
/opt/conda/lib/python3.10/site-packages (from geopandas) (3.6.1)
```

```
Requirement already satisfied: shapely>=1.8.0 in
/opt/conda/lib/python3.10/site-packages (from geopandas) (1.8.5.post1)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib) (1.1.0)
Requirement already satisfied: cycler>=0.10 in
/opt/conda/lib/python3.10/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy<2,>=1.20 in
/opt/conda/lib/python3.10/site-packages (from matplotlib) (1.24.3)
Requirement already satisfied: pillow>=6.2.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib) (10.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.10/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: attrs>=19.2.0 in
/opt/conda/lib/python3.10/site-packages (from fiona>=1.8.21-
>geopandas) (23.1.0)
Requirement already satisfied: certifi in
/opt/conda/lib/python3.10/site-packages (from fiona>=1.8.21-
>geopandas) (2023.7.22)
Requirement already satisfied: click~=8.0 in
/opt/conda/lib/python3.10/site-packages (from fiona>=1.8.21-
>geopandas) (8.1.7)
Requirement already satisfied: click-plugins>=1.0 in
/opt/conda/lib/python3.10/site-packages (from fiona>=1.8.21-
>geopandas) (1.1.1)
Requirement already satisfied: cligj>=0.5 in
/opt/conda/lib/python3.10/site-packages (from fiona>=1.8.21-
>geopandas) (0.7.2)
Requirement already satisfied: six in /opt/conda/lib/python3.10/site-
packages (from fiona>=1.8.21->geopandas) (1.16.0)
Requirement already satisfied: setuptools in
/opt/conda/lib/python3.10/site-packages (from fiona>=1.8.21-
>geopandas) (68.1.2)
Requirement already satisfied: pytz>=2020.1 in
/opt/conda/lib/python3.10/site-packages (from pandas>=1.4.0-
>geopandas) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in
/opt/conda/lib/python3.10/site-packages (from pandas>=1.4.0-
>geopandas) (2023.3)
```

```python
import geopandas as gpd
import matplotlib.pyplot as plt

# Create a GeoDataFrame with country geometries
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
```

```
# Merge the world GeoDataFrame with the top countries DataFrame
merged = world.merge(top_countries.reset_index(), left_on='name',
right_on='Country')

# Plotting
fig, ax = plt.subplots(1, 1, figsize=(15, 10))
merged.plot(column='InvoiceNo', cmap='Blues', linewidth=0.8, ax=ax,
edgecolor='0.8', legend=True)
ax.set_title('Top 5 Countries with the Highest Number of Orders')
ax.set_axis_off()
plt.show()
```




```
# Is there a correlation between the country of the customer and the
average order value?
# Calculate total revenue for each order
df['TotalRevenue'] = df['Quantity'] * df['UnitPrice']

# Calculate average order value for each country
```

```
avg_order_value_by_country = df.groupby('Country')
['TotalRevenue'].mean()

# Calculate correlation
correlation =
avg_order_value_by_country.corr(df['Country'].astype('category').cat.c
odes)
print("Correlation between the country and average order value:",
correlation)
```

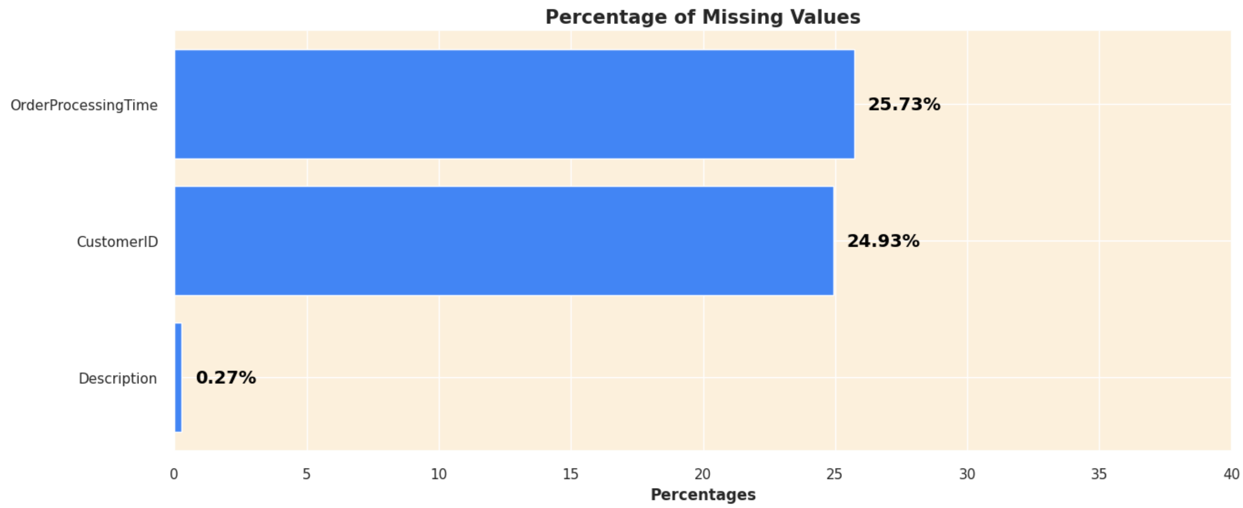
```
Correlation between the country and average order value: nan
```

```
# Customer Behavior?
# Profitability Analysis?
# Customer Satisfaction?
```

STEP-3: DATA CLEANING AND TRASFORMATION

3.1: Handling Missing Values

```
missing_data = df.isnull().sum()
missing_percentage = (missing_data[missing_data > 0] / df.shape[0]) *
100

missing_percentage.sort_values(ascending=True, inplace=True)

fig, ax = plt.subplots(figsize=(15, 6))
ax.barh(missing_percentage.index, missing_percentage, color='#4285F4')

for i, (value, name) in enumerate(zip(missing_percentage,
missing_percentage.index)):
    ax.text(value + 0.5, i, f"{value:.2f}%", ha='left', va='center',
fontweight='bold', fontsize=14, color='black')
ax.set_xlim([0, 40])
plt.title("Percentage of Missing Values",   fontweight='bold',
fontsize=15)
plt.xlabel('Percentages',fontweight='bold', fontsize=12)
plt.show()
```




```
#dropping all the missing rows with CustomerID and Description(Reason:
would create noise and bias also the project is of customer
segmentation so without custmerID makes no sense to work on tht rows)
df = df.dropna(subset=['CustomerID', 'Description'])

df.isnull().sum().sum()

4372
```

3.2: Handling Duplicates

```
duplicated_entries = df[df.duplicated(keep=False)]

sorted_duplicates = duplicated_entries.sort_values(by=['InvoiceNo',
'StockCode', 'Description', 'CustomerID', 'Quantity'])

sorted_duplicates.head(10)

    InvoiceNo StockCode                        Description  Quantity  \
494    536409     21866        UNION JACK FLAG LUGGAGE TAG         1
517    536409     21866        UNION JACK FLAG LUGGAGE TAG         1
485    536409     22111       SCOTTIE DOG HOT WATER BOTTLE         1
539    536409     22111       SCOTTIE DOG HOT WATER BOTTLE         1
489    536409     22866      HAND WARMER SCOTTY DOG DESIGN         1
527    536409     22866      HAND WARMER SCOTTY DOG DESIGN         1
521    536409     22900    SET 2 TEA TOWELS I LOVE LONDON          1
537    536409     22900    SET 2 TEA TOWELS I LOVE LONDON          1
578    536412     21448          12 DAISY PEGS IN WOOD BOX         1
598    536412     21448          12 DAISY PEGS IN WOOD BOX         1

            InvoiceDate  UnitPrice  CustomerID         Country  \
494 2010-12-01 11:45:00       1.25     17908.0  United Kingdom
517 2010-12-01 11:45:00       1.25     17908.0  United Kingdom
485 2010-12-01 11:45:00       4.95     17908.0  United Kingdom
```

```
539 2010-12-01 11:45:00      4.95     17908.0  United Kingdom
489 2010-12-01 11:45:00      2.10     17908.0  United Kingdom
527 2010-12-01 11:45:00      2.10     17908.0  United Kingdom
521 2010-12-01 11:45:00      2.95     17908.0  United Kingdom
537 2010-12-01 11:45:00      2.95     17908.0  United Kingdom
578 2010-12-01 11:49:00      1.65     17920.0  United Kingdom
598 2010-12-01 11:49:00      1.65     17920.0  United Kingdom

    Transaction_Status  Cancelled  Month  TotalRevenue  DayOfWeek
HourOfDay  \
494          Completed          0     12          1.25          2
11
517          Completed          0     12          1.25          2
11
485          Completed          0     12          4.95          2
11
539          Completed          0     12          4.95          2
11
489          Completed          0     12          2.10          2
11
527          Completed          0     12          2.10          2
11
521          Completed          0     12          2.95          2
11
537          Completed          0     12          2.95          2
11
578          Completed          0     12          1.65          2
11
598          Completed          0     12          1.65          2
11

    OrderProcessingTime  Quarter
494              0 days        4
517              0 days        4
485              0 days        4
539              0 days        4
489              0 days        4
527              0 days        4
521              0 days        4
537              0 days        4
578              0 days        4
598              0 days        4
```

```
sorted_duplicates
```

```
       InvoiceNo StockCode                          Description
Quantity  \
494       536409     21866         UNION JACK FLAG LUGGAGE TAG
1
517       536409     21866         UNION JACK FLAG LUGGAGE TAG
```

```
1
485       536409     22111         SCOTTIE DOG HOT WATER BOTTLE
1
539       536409     22111         SCOTTIE DOG HOT WATER BOTTLE
1
489       536409     22866        HAND WARMER SCOTTY DOG DESIGN
1
...          ...       ...                                  ...
...
411644   C572226     85066          CREAM SWEETHEART MINI CHEST
-1
436250   C574095     22326  ROUND SNACK BOXES SET OF4 WOODLAND
-1
436251   C574095     22326  ROUND SNACK BOXES SET OF4 WOODLAND
-1
461407   C575940     23309  SET OF 60 I LOVE LONDON CAKE CASES
-24
461408   C575940     23309  SET OF 60 I LOVE LONDON CAKE CASES
-24

               InvoiceDate  UnitPrice  CustomerID         Country  \
494    2010-12-01 11:45:00       1.25     17908.0  United Kingdom
517    2010-12-01 11:45:00       1.25     17908.0  United Kingdom
485    2010-12-01 11:45:00       4.95     17908.0  United Kingdom
539    2010-12-01 11:45:00       4.95     17908.0  United Kingdom
489    2010-12-01 11:45:00       2.10     17908.0  United Kingdom
...                    ...        ...         ...             ...
411644 2011-10-21 13:58:00      12.75     15321.0  United Kingdom
436250 2011-11-03 09:54:00       2.95     12674.0          France
436251 2011-11-03 09:54:00       2.95     12674.0          France
461407 2011-11-13 11:38:00       0.55     17838.0  United Kingdom
461408 2011-11-13 11:38:00       0.55     17838.0  United Kingdom

       Transaction_Status  Cancelled  Month  TotalRevenue
DayOfWeek  \
494             Completed          0     12          1.25          2

517             Completed          0     12          1.25          2

485             Completed          0     12          4.95          2

539             Completed          0     12          4.95          2

489             Completed          0     12          2.10          2

...                   ...        ...    ...           ...        ...

411644          Cancelled          1     10        -12.75          4

436250          Cancelled          1     11         -2.95          3
```

```
436251         Cancelled         1     11         -2.95         3

461407         Cancelled         1     11        -13.20         6

461408         Cancelled         1     11        -13.20         6


        HourOfDay OrderProcessingTime  Quarter
494            11              0 days        4
517            11              0 days        4
485            11              0 days        4
539            11              0 days        4
489            11              0 days        4
...           ...                 ...      ...
411644         13              0 days        4
436250          9              0 days        4
436251          9              0 days        4
461407         11              0 days        4
461408         11              0 days        4

[9726 rows x 16 columns]
```

```
# Removing all the dupliacte entries as it seems the errorneos data
since it is impossible to have same transictions again and again
df.drop_duplicates(inplace=True)

df.shape

(401778, 16)
```

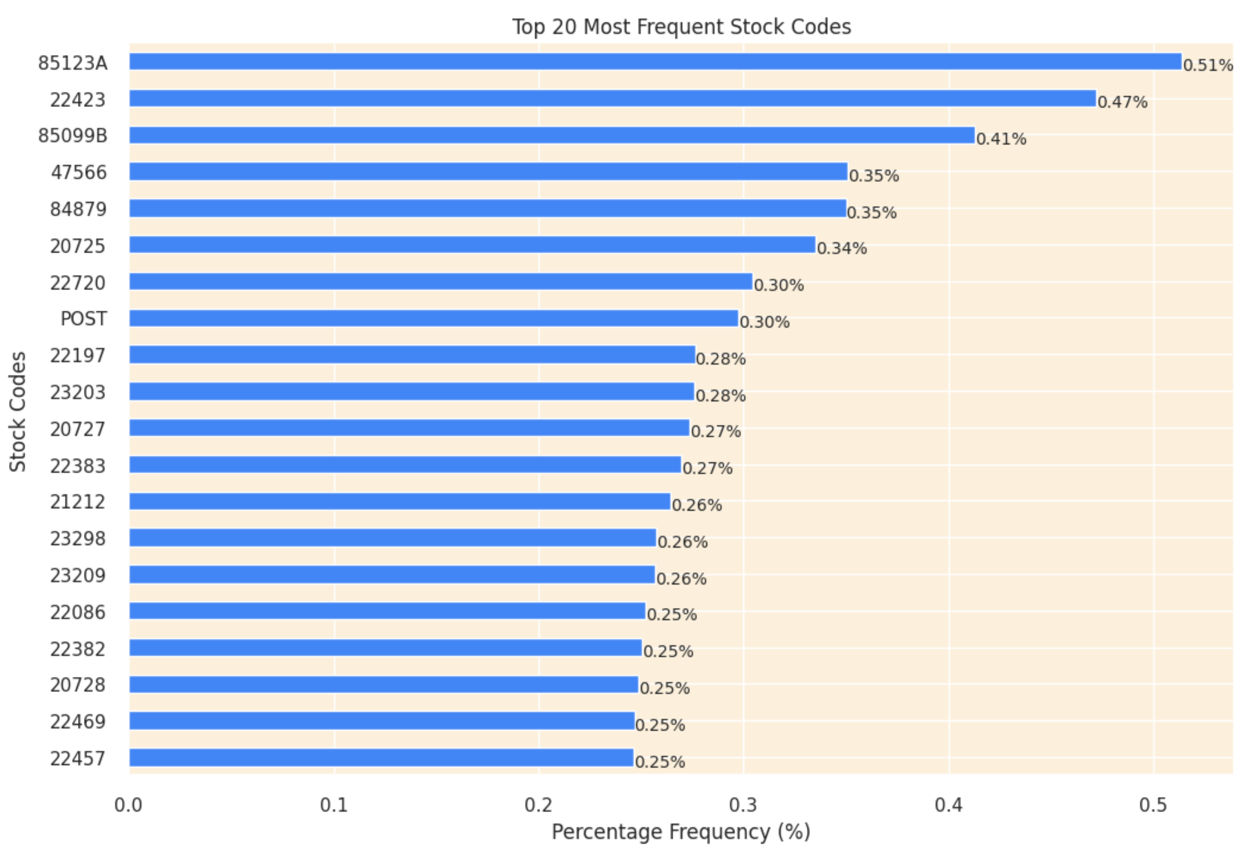
3.3: Handling Cancelled Transcitions

```
df['Transaction_Result'] =
np.where(df['InvoiceNo'].astype(str).str.startswith('C'), 'Cancelled',
'Successful')
cancelled_transactions = df[df['Transaction_Result'] == 'Cancelled']

# What is the percentage of orders that have experienced returns or
refunds?
cancelled_percentage = (cancelled_transactions.shape[0] / df.shape[0])
* 100
print(f"Cancelled Percentage: {cancelled_percentage:.2f}%")

Cancelled Percentage: 2.21%
```

3.4: Dealing with Stock Code Anamolies

```
unique_stock_codes = df['StockCode'].nunique()
print(f"Unique Stock Codes: {unique_stock_codes}")
```

```
Unique Stock Codes: 3684

top_20_stock_codes =
df['StockCode'].value_counts(normalize=True).head(20) * 100

plt.figure(figsize=(12, 8))
top_20_stock_codes.plot(kind='barh', color='#4285F4')

for index, value in enumerate(top_20_stock_codes):
    plt.text(value, index + 0.25, f'{value:.2f}%', fontsize=10)

plt.title('Top 20 Most Frequent Stock Codes')
plt.xlabel('Percentage Frequency (%)')
plt.ylabel('Stock Codes')
plt.gca().invert_yaxis()
plt.show()

print("Visualization for the Top 20 Most Frequent Stock Codes")
```




```
Visualization for the Top 20 Most Frequent Stock Codes

unique_stock_codes = df['StockCode'].unique()
numeric_char_counts_in_unique_codes =
pd.Series(unique_stock_codes).apply(lambda x: sum(c.isdigit() for c in
```

```
str(x))).value_counts()

print("Value counts of numeric character frequencies in unique stock
codes:")
print("-"*70)
print(numeric_char_counts_in_unique_codes)
```

```
Value counts of numeric character frequencies in unique stock codes:
----------------------------------------------------------------------
5    3676
0       7
1       1
Name: count, dtype: int64
```

```
# Checking for all the codes that does not have digit which might be
the errorneous stock code
anomalous_stock_codes = [code for code in unique_stock_codes if
sum(c.isdigit() for c in str(code)) in (0, 1)]

print("Anomalous stock codes:")
print("-"*22)
for code in anomalous_stock_codes:
    print(code)
```

```
Anomalous stock codes:
----------------------
POST
D
C2
M
BANK CHARGES
PADS
DOT
CRUK
```

```
# All the stock codes anomalies are removed which are just alphabets
df = df[~df['StockCode'].isin(anomalous_stock_codes)]

df.shape[0]
```

```
399863
```

3.5: Dealing with Zero Unit Prices

```
df['UnitPrice'].describe()
```

```
count    399863.000000
mean          2.907701
std           4.451412
min           0.000000
25%           1.250000
```

```
50%             1.950000
75%             3.750000
max           649.500000
Name: UnitPrice, dtype: float64

# Minimum value is 0 which means that it costs nothing but it is not
possible
# Removing all the rows with 0 unit price
# They are probably the wrong
df = df[df['UnitPrice'] > 0]
```

STEP-4: FEATURE ENGINEERING

RFM FEATURES

4.1: Recency

This metric indicates how recently a customer has made a purchase. A lower recency value means the customer has purchased more recently, indicating higher engagement with the brand.

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
df['InvoiceDay'] = df['InvoiceDate'].dt.date
customer_data = df.groupby('CustomerID')
['InvoiceDay'].max().reset_index()
most_recent_date = df['InvoiceDay'].max()
customer_data['InvoiceDay'] =
pd.to_datetime(customer_data['InvoiceDay'])
most_recent_date = pd.to_datetime(most_recent_date)
customer_data['Recency'] = (most_recent_date -
customer_data['InvoiceDay']).dt.days
customer_data.drop(columns=['InvoiceDay'], inplace=True)
```

Named the customer-centric dataframe as customer_last_purchase, which will eventually contain all the customer-based features we plan to create.

```
customer_data.head()

   CustomerID  Recency
0     12346.0      325
1     12347.0        2
2     12348.0       75
3     12349.0       18
4     12350.0      310
```

4.2: Frequency

This metric signifies how often a customer makes a purchase within a certain period. A higher frequency value indicates a customer who interacts with the business more often, suggesting higher loyalty or satisfaction.

```
total_transactions = (
    df.groupby('CustomerID')['InvoiceNo']
    .nunique()
    .reset_index()
    .rename(columns={'InvoiceNo': 'Frequency'})
)
customer_data = pd.merge(customer_data, total_transactions,
on='CustomerID')
#customer_data = pd.merge(customer_data, total_products_purchased,
on='CustomerID')
customer_data.head()
```

	CustomerID	Recency	Frequency
0	12346.0	325	2
1	12347.0	2	7
2	12348.0	75	4
3	12349.0	18	1
4	12350.0	310	1

4.3: Monetary

This metric represents the total amount of money a customer has spent over a certain period. Customers who have a higher monetary value have contributed more to the business, indicating their potential high lifetime value.

```
df['Total_Spend'] = df['UnitPrice'] * df['Quantity']
total_spend = (
    df.groupby('CustomerID')['Total_Spend']
    .sum()
    .reset_index()
    .rename(columns={'Total_Spend': 'Monetary'})
)


""""
    average_transaction_value = (
    total_spend.merge(total_transactions, on='CustomerID')
    .assign(Average_Transaction_Value=lambda x: x['Total_Spend'] /
x['Total_Transactions'])
)"""
customer_data = pd.merge(customer_data, total_spend, on='CustomerID')
#customer_data = pd.merge(customer_data,
average_transaction_value[['CustomerID',
'Average_Transaction_Value']], on='CustomerID')
customer_data.head()
```

	CustomerID	Recency	Frequency	Monetary
0	12346.0	325	2	0.00
1	12347.0	2	7	4310.00
2	12348.0	75	4	1437.24

```
3      12349.0        18           1   1457.55
4      12350.0       310           1    294.40
```

4.4: RFM Quartiles

```
#uncomment and do run
customer_data.set_index('CustomerID', inplace=True)

customer_data.head()
```

	Recency	Frequency	Monetary
CustomerID			
12346.0	325	2	0.00
12347.0	2	7	4310.00
12348.0	75	4	1437.24
12349.0	18	1	1457.55
12350.0	310	1	294.40

```
quantiles = customer_data.quantile(q=[0.25,0.5,0.75])
quantiles
```

	Recency	Frequency	Monetary
0.25	16.0	1.0	292.3725
0.50	50.0	3.0	642.5450
0.75	143.0	5.0	1584.9300

```
quantiles.to_dict()
```

```
{'Recency': {0.25: 16.0, 0.5: 50.0, 0.75: 143.0},
 'Frequency': {0.25: 1.0, 0.5: 3.0, 0.75: 5.0},
 'Monetary': {0.25: 292.3725, 0.5: 642.5450000000001, 0.75: 1584.93}}
```

4.5: RFM Segments

```
# Arguments (x = value, p = recency, monetary_value, frequency, d =
quartiles dict)
def RScore(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
# Arguments (x = value, p = recency, monetary_value, frequency, k =
quartiles dict)
def FMScore(x,p,d):
    if x <= d[p][0.25]:
        return 1
```

```
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4

#create rfm segmentation table
rfm_segmentation = customer_data
rfm_segmentation['R_Quartile'] =
rfm_segmentation['Recency'].apply(RScore, args=('Recency',quantiles,))
rfm_segmentation['F_Quartile'] =
rfm_segmentation['Frequency'].apply(FMScore,
args=('Frequency',quantiles,))
rfm_segmentation['M_Quartile'] =
rfm_segmentation['Monetary'].apply(FMScore,
args=('Monetary',quantiles,))

rfm_segmentation.head()

            Recency  Frequency  Monetary  R_Quartile  F_Quartile
M_Quartile
CustomerID

12346.0         325          2      0.00           1           2
1
12347.0           2          7   4310.00           4           4
4
12348.0          75          4   1437.24           2           3
3
12349.0          18          1   1457.55           3           1
3
12350.0         310          1    294.40           1           1
2

rfm_segmentation['RFMScore'] = rfm_segmentation.R_Quartile.map(str) \
                            + rfm_segmentation.F_Quartile.map(str) \
                            + rfm_segmentation.M_Quartile.map(str)
rfm_segmentation.head()

            Recency  Frequency  Monetary  R_Quartile  F_Quartile
M_Quartile  \
CustomerID

12346.0         325          2      0.00           1           2
1
12347.0           2          7   4310.00           4           4
4
12348.0          75          4   1437.24           2           3
3
12349.0          18          1   1457.55           3           1
```

```
3
12350.0         310           1    294.40          1           1
2

         RFMScore
CustomerID
12346.0       121
12347.0       444
12348.0       233
12349.0       313
12350.0       112
```

Best Recency score = 4: most recently purchase. Best Frequency score = 4: most quantity purchase. Best Monetary score = 4: spent the most.

```
#best customers
rfm_segmentation[rfm_segmentation['RFMScore']=='444'].sort_values('Monetary', ascending=False).head(10)
```

```
            Recency  Frequency   Monetary  R_Quartile  F_Quartile
M_Quartile  \
CustomerID

14646.0           1         73  278778.02           4           4
4
18102.0           0         60  259657.30           4           4
4
17450.0           8         49  189575.53           4           4
4
14911.0           1        242  128768.24           4           4
4
14156.0           9         64  113685.77           4           4
4
17511.0           2         45   88138.20           4           4
4
16684.0           4         30   65920.12           4           4
4
13694.0           3         57   62924.10           4           4
4
15311.0           0        118   59284.19           4           4
4
13089.0           2        118   57339.83           4           4
4

           RFMScore
CustomerID
14646.0       444
18102.0       444
17450.0       444
```

```
14911.0          444
14156.0          444
17511.0          444
16684.0          444
13694.0          444
15311.0          444
13089.0          444
```

```
print("Best Customers:
",len(rfm_segmentation[rfm_segmentation['RFMScore']=='444']))
print('Loyal Customers:
',len(rfm_segmentation[rfm_segmentation['F_Quartile']==4]))
print("Big Spenders:
",len(rfm_segmentation[rfm_segmentation['M_Quartile']==4]))
print('Almost Lost: ',
len(rfm_segmentation[rfm_segmentation['RFMScore']=='244']))
print('Lost Customers:
',len(rfm_segmentation[rfm_segmentation['RFMScore']=='144']))
print('Lost Cheap Customers:
',len(rfm_segmentation[rfm_segmentation['RFMScore']=='111']))

Best Customers:  482
Loyal Customers:  1067
Big Spenders:  1091
Almost Lost:  87
Lost Customers:  15
Lost Cheap Customers:  404
```

STEP-5: K-Means Clustering

```
customer_data

            Recency  Frequency  Monetary  R_Quartile  F_Quartile
M_Quartile  \
CustomerID

12346.0         325          2      0.00           1           2
1
12347.0           2          7   4310.00           4           4
4
12348.0          75          4   1437.24           2           3
3
12349.0          18          1   1457.55           3           1
3
12350.0         310          1    294.40           1           1
2
...             ...        ...       ...         ...         ...
...
18280.0         277          1    180.60           1           1
1
```

```
18281.0         180           1      80.82            1            1
1
18282.0           7           3     176.60            4            2
1
18283.0           3          16    2041.23            4            4
4
18287.0          42           3    1837.28            3            2
4

           RFMScore
CustomerID
12346.0         121
12347.0         444
12348.0         233
12349.0         313
12350.0         112
...             ...
18280.0         111
18281.0         111
18282.0         421
18283.0         444
18287.0         324

[4362 rows x 7 columns]
```

```
customer_data.reset_index(inplace=True)

customer_data
```

```
      CustomerID  Recency  Frequency  Monetary  R_Quartile  F_Quartile
\
0        12346.0      325          2      0.00           1           2

1        12347.0        2          7   4310.00           4           4

2        12348.0       75          4   1437.24           2           3

3        12349.0       18          1   1457.55           3           1

4        12350.0      310          1    294.40           1           1

...          ...      ...        ...       ...         ...         ...

4357     18280.0      277          1    180.60           1           1

4358     18281.0      180          1     80.82           1           1

4359     18282.0        7          3    176.60           4           2

4360     18283.0        3         16   2041.23           4           4
```

```
4361     18287.0       42            3   1837.28           3           2


      M_Quartile RFMScore
0              1      121
1              4      444
2              3      233
3              3      313
4              2      112
...          ...      ...
4357           1      111
4358           1      111
4359           1      421
4360           4      444
4361           4      324

[4362 rows x 8 columns]

customer_data.rename(columns={'Recency': 'last_purchased'},
inplace=True)
customer_data.rename(columns={'Frequency': 'quantity'}, inplace=True)
customer_data.rename(columns={'Monetary': 'price'}, inplace=True)

customer_data.rename(columns={'Price': 'Total_Money_Spent'},
inplace=True)

customer_data.head()

   CustomerID  last_purchased  quantity    price  R_Quartile
F_Quartile  \
0     12346.0             325         2     0.00           1
2
1     12347.0               2         7  4310.00           4
4
2     12348.0              75         4  1437.24           2
3
3     12349.0              18         1  1457.55           3
1
4     12350.0             310         1   294.40           1
1

   M_Quartile RFMScore
0           1      121
1           4      444
2           3      233
3           3      313
4           2      112

customer_segmentation=customer_data[['CustomerID','last_purchased','qu
antity','price']]
```

```
customer_segmentation.head()
```

	CustomerID	last_purchased	quantity	price
0	12346.0	325	2	0.00
1	12347.0	2	7	4310.00
2	12348.0	75	4	1437.24
3	12349.0	18	1	1457.55
4	12350.0	310	1	294.40

Number of Different Products Purchased: This metric reflects the variety of products a customer has bought. A higher count suggests a diverse taste, covering a broad range of products. Conversely, a lower count indicates a more specific or focused preference. Analyzing this diversity aids in categorizing customers based on their buying habits, providing valuable insights for tailoring personalized product recommendations.

```
unique_products_purchased = df.groupby('CustomerID')
['StockCode'].nunique().reset_index()
unique_products_purchased.rename(columns={'StockCode':
'Unique_Products'}, inplace=True)

customer_segmentation = pd.merge(customer_segmentation,
unique_products_purchased, on='CustomerID')

customer_segmentation.head()
```

	CustomerID	last_purchased	quantity	price	Unique_Products
0	12346.0	325	2	0.00	1
1	12347.0	2	7	4310.00	103
2	12348.0	75	4	1437.24	21
3	12349.0	18	1	1457.55	72
4	12350.0	310	1	294.40	16

In this stage, our goal is to comprehend and record the shopping patterns and habits of customers. These attributes will provide valuable insights into when customers prefer to shop, offering crucial information for tailoring a personalized shopping experience.

```
df['Day_Of_Week'] = df['InvoiceDate'].dt.dayofweek
df['Hour'] = df['InvoiceDate'].dt.hour
days_between_purchases = df.groupby('CustomerID')['InvoiceDay'].apply(
    lambda x: (x.diff().dropna()).apply(lambda y: y.days)
)
average_days_between_purchases =
days_between_purchases.groupby('CustomerID').mean().reset_index()
average_days_between_purchases.rename(columns={'InvoiceDay':
'Average_Days_Between_Purchases'}, inplace=True)

favorite_shopping_day = df.groupby(['CustomerID',
'Day_Of_Week']).size().reset_index(name='Count')
favorite_shopping_day =
```

```
favorite_shopping_day.loc[favorite_shopping_day.groupby('CustomerID')
['Count'].idxmax()][['CustomerID', 'Day_Of_Week']]

favorite_shopping_hour = df.groupby(['CustomerID',
'Hour']).size().reset_index(name='Count')
favorite_shopping_hour =
favorite_shopping_hour.loc[favorite_shopping_hour.groupby('CustomerID'
)['Count'].idxmax()][['CustomerID', 'Hour']]

customer_segmentation = pd.merge(customer_segmentation,
average_days_between_purchases, on='CustomerID')
customer_segmentation = pd.merge(customer_segmentation,
favorite_shopping_day, on='CustomerID')
customer_segmentation = pd.merge(customer_segmentation,
favorite_shopping_hour, on='CustomerID')

customer_segmentation.head()
```

```
   CustomerID  last_purchased  quantity    price  Unique_Products  \
0     12346.0             325         2     0.00                1
1     12347.0               2         7  4310.00              103
2     12348.0              75         4  1437.24               21
3     12349.0              18         1  1457.55               72
4     12350.0             310         1   294.40               16

   Average_Days_Between_Purchases  Day_Of_Week  Hour
0                        0.000000            1    10
1                        2.016575            1    14
2                       10.884615            3    19
3                        0.000000            0     9
4                        0.000000            2    16
```

In this phase, we will incorporate a geographical feature indicating the location of customers. Recognizing the geographic distribution of customers is crucial for various reasons:

Country: This attribute specifies the country of each customer, offering insights into region-specific buying patterns and preferences. Varied regions may exhibit distinct preferences and purchasing behaviors, crucial for tailoring marketing strategies and optimizing inventory. Additionally, it plays a significant role in logistics and supply chain optimization, particularly for online retailers where shipping and delivery are vital factors.

```
df['Country'].value_counts(normalize=True).head()
```

```
Country
United Kingdom    0.891036
Germany           0.022710
France            0.020389
EIRE              0.018428
Spain             0.006158
Name: proportion, dtype: float64
```

Inference: Given that a substantial portion (89%) of transactions are originating from the United Kingdom, we might consider creating a binary feature indicating whether the transaction is from the UK or not. This approach can potentially streamline the clustering process without losing critical geographical information, especially when considering the application of algorithms like K-means which are sensitive to the dimensionality of the feature space.

```
customer_country = df.groupby(['CustomerID',
'Country']).size().reset_index(name='Number_of_Transactions')

customer_main_country =
customer_country.sort_values('Number_of_Transactions',
ascending=False).drop_duplicates('CustomerID')

customer_main_country['Is_UK'] =
customer_main_country['Country'].apply(lambda x: 1 if x == 'United
Kingdom' else 0)

customer_segmentation = pd.merge(customer_segmentation,
customer_main_country[['CustomerID', 'Is_UK']], on='CustomerID',
how='left')

customer_segmentation.head()
```

```
   CustomerID  last_purchased  quantity    price  Unique_Products  \
0     12346.0             325         2     0.00                1
1     12347.0               2         7  4310.00              103
2     12348.0              75         4  1437.24               21
3     12349.0              18         1  1457.55               72
4     12350.0             310         1   294.40               16

   Average_Days_Between_Purchases  Day_Of_Week  Hour  Is_UK
0                        0.000000            1    10      1
1                        2.016575            1    14      0
2                       10.884615            3    19      0
3                        0.000000            0     9      0
4                        0.000000            2    16      0
```

In this phase, I'll explore customer cancellation patterns to refine our segmentation model. I'll introduce two key metrics:

1. **Cancellation Frequency:** This indicates how often a customer cancels transactions, helping identify those more likely to cancel, potentially signaling dissatisfaction.

2. **Cancellation Rate:** This is the proportion of canceled transactions out of all transactions, offering a normalized view. A high rate suggests potential dissatisfaction. Incorporating these insights will provide a more thorough understanding of customer behavior for improved segmentation.

```
# Calculate the total number of transactions made by each customer
total_transactions = df.groupby('CustomerID')
```

```
['InvoiceNo'].nunique().reset_index()

# Calculate the number of cancelled transactions for each customer
cancelled_transactions = df[df['Transaction_Status'] == 'Cancelled']
cancellation_frequency = cancelled_transactions.groupby('CustomerID')
['InvoiceNo'].nunique().reset_index()
cancellation_frequency.rename(columns={'InvoiceNo':
'Cancellation_Frequency'}, inplace=True)

# Merge the Cancellation Frequency data into the customer_data
dataframe
customer_segmentation = pd.merge(customer_segmentation,
cancellation_frequency, on='CustomerID', how='left')

# Replace NaN values with 0 (for customers who have not cancelled any
transaction)
customer_segmentation['Cancellation_Frequency'].fillna(0,
inplace=True)

# Calculate the Cancellation Rate
customer_segmentation['Cancellation_Rate'] =
customer_segmentation['Cancellation_Frequency'] /
total_transactions['InvoiceNo']

# Display the first few rows of the customer_data dataframe
customer_segmentation.head()
```

	CustomerID	last_purchased	quantity	price	Unique_Products	\
0	12346.0	325	2	0.00	1	
1	12347.0	2	7	4310.00	103	
2	12348.0	75	4	1437.24	21	
3	12349.0	18	1	1457.55	72	
4	12350.0	310	1	294.40	16	

	Average_Days_Between_Purchases	Day_Of_Week	Hour	Is_UK	\
0	0.000000	1	10	1	
1	2.016575	1	14	0	
2	10.884615	3	19	0	
3	0.000000	0	9	0	
4	0.000000	2	16	0	

	Cancellation_Frequency	Cancellation_Rate
0	1.0	0.5
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

Monthly_Spending_Mean: This is the average amount a customer spends monthly. It helps us gauge the general spending habit of each customer. A higher mean indicates a customer who spends more, potentially showing interest in premium products, whereas a lower mean might indicate a more budget-conscious customer.

Spending_Trend: This reflects the trend in a customer's spending over time, calculated as the slope of the linear trend line fitted to their spending data. A positive value indicates an increasing trend in spending, possibly pointing to growing loyalty or satisfaction. Conversely, a negative trend might signal decreasing interest or satisfaction, highlighting a need for re-engagement strategies. A near-zero value signifies stable spending habits. Recognizing these trends can help in developing strategies to either maintain or alter customer spending patterns, enhancing the effectiveness of marketing campaigns.

```
from scipy.stats import linregress

df['Year'] = df['InvoiceDate'].dt.year
df['Month'] = df['InvoiceDate'].dt.month

monthly_spending = df.groupby(['CustomerID', 'Year', 'Month'])
['Total_Spend'].sum().reset_index()

seasonal_buying_patterns = monthly_spending.groupby('CustomerID')
['Total_Spend'].agg(['mean', 'std']).reset_index()
seasonal_buying_patterns.rename(columns={'mean':
'Monthly_Spending_Mean', 'std': 'Monthly_Spending_Std'}, inplace=True)

seasonal_buying_patterns['Monthly_Spending_Std'].fillna(0,
inplace=True)

def calculate_trend(spend_data):
    # If there are more than one data points, we calculate the trend
using linear regression
    if len(spend_data) > 1:
        x = np.arange(len(spend_data))
        slope, _, _, _, _ = linregress(x, spend_data)
        return slope
    else:
        return 0

spending_trends = monthly_spending.groupby('CustomerID')
['Total_Spend'].apply(calculate_trend).reset_index()
spending_trends.rename(columns={'Total_Spend': 'Spending_Trend'},
inplace=True)

customer_segmentation = pd.merge(customer_segmentation,
seasonal_buying_patterns[['CustomerID', 'Monthly_Spending_Mean']],
on='CustomerID')
customer_segmentation = pd.merge(customer_segmentation,
spending_trends[['CustomerID', 'Spending_Trend']], on='CustomerID')

customer_segmentation.head()
```

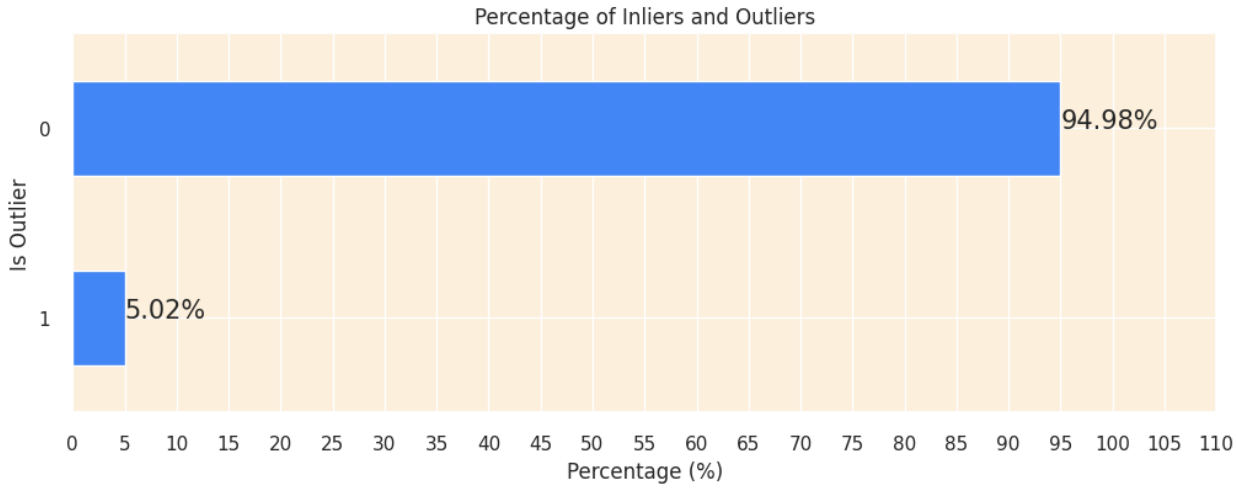
```
   CustomerID  last_purchased  quantity    price  Unique_Products  \
0     12346.0             325         2     0.00                1
1     12347.0               2         7  4310.00              103
2     12348.0              75         4  1437.24               21
3     12349.0              18         1  1457.55               72
4     12350.0             310         1   294.40               16

   Average_Days_Between_Purchases  Day_Of_Week  Hour  Is_UK  \
0                        0.000000            1    10      1
1                        2.016575            1    14      0
2                       10.884615            3    19      0
3                        0.000000            0     9      0
4                        0.000000            2    16      0

   Cancellation_Frequency  Cancellation_Rate  Monthly_Spending_Mean  \
0                     1.0                0.5               0.000000
1                     0.0                0.0             615.714286
2                     0.0                0.0             359.310000
3                     0.0                0.0            1457.550000
4                     0.0                0.0             294.400000

   Spending_Trend
0        0.000000
1        4.486071
2     -100.884000
3        0.000000
4        0.000000

# Changing the data type of 'CustomerID' to string as it is a unique
identifier and not used in mathematical operations
customer_segmentation['CustomerID'] =
customer_segmentation['CustomerID'].astype(str)

# Convert data types of columns to optimal types
customer_segmentation = customer_segmentation.convert_dtypes()
```

STEP-6: Outlier Detection

In this step, I'll address outliers in our dataset, which are data points significantly different from the majority. Outliers can distort clustering results, especially in k-means clustering. To handle this, I'll employ the Isolation Forest algorithm, suitable for multi-dimensional data. This algorithm isolates outliers by randomly selecting features and split values, providing a computationally efficient solution.

```
model = IsolationForest(contamination=0.05, random_state=0)
customer_segmentation['Outlier_Scores'] =
model.fit_predict(customer_segmentation.iloc[:, 1:].to_numpy())
customer_segmentation['Is_Outlier'] = [1 if x == -1 else 0 for x in
customer_segmentation['Outlier_Scores']]
customer_segmentation.head()
```

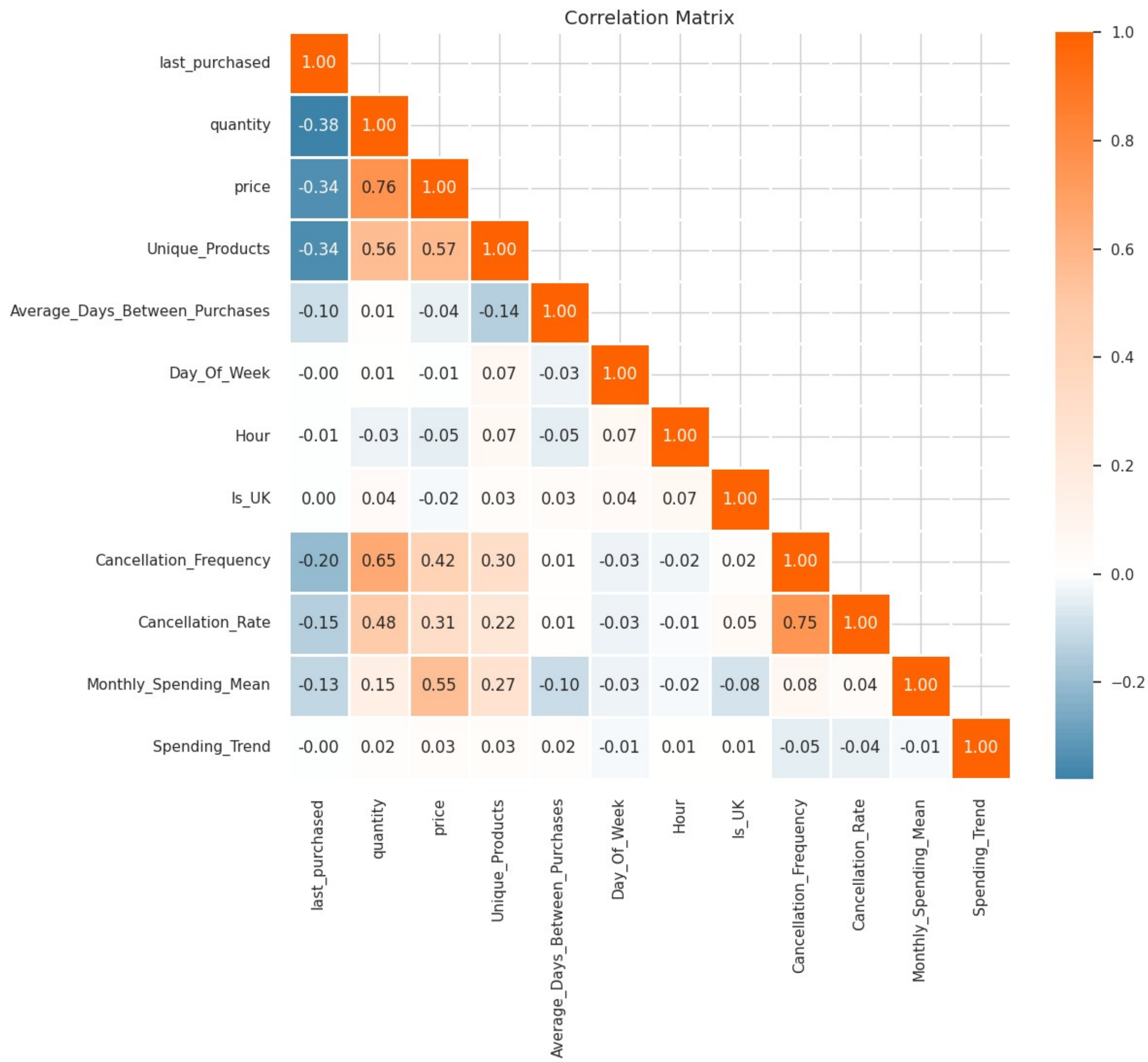
```
   CustomerID  last_purchased  quantity    price  Unique_Products  \
0     12346.0             325         2      0.0                1   
1     12347.0               2         7   4310.0              103   
2     12348.0              75         4  1437.24               21   
3     12349.0              18         1  1457.55               72   
4     12350.0             310         1    294.4               16   

   Average_Days_Between_Purchases  Day_Of_Week  Hour  Is_UK  \
0                             0.0            1    10      1   
1                        2.016575            1    14      0   
2                       10.884615            3    19      0   
3                             0.0            0     9      0   
4                             0.0            2    16      0   

   Cancellation_Frequency  Cancellation_Rate  Monthly_Spending_Mean  \
0                       1                0.5                    0.0   
1                       0                0.0             615.714286   
2                       0                0.0                 359.31   
3                       0                0.0                1457.55   
4                       0                0.0                  294.4   

   Spending_Trend  Outlier_Scores  Is_Outlier  
0             0.0               1           0  
1        4.486071               1           0  
2        -100.884              -1           1  
3             0.0               1           0  
4             0.0               1           0  

#Visualization
outlier_percentage =
customer_segmentation['Is_Outlier'].value_counts(normalize=True) * 100

plt.figure(figsize=(12, 4))
outlier_percentage.plot(kind='barh', color='#4285F4')

for index, value in enumerate(outlier_percentage):
    plt.text(value, index, f'{value:.2f}%', fontsize=15)


plt.title('Percentage of Inliers and Outliers')
plt.xticks(ticks=np.arange(0, 115, 5))
plt.xlabel('Percentage (%)')
plt.ylabel('Is Outlier')
plt.gca().invert_yaxis()
plt.show()
```




```
outliers_data =
customer_segmentation[customer_segmentation['Is_Outlier'] == 1]
customer_data_cleaned =
customer_segmentation[customer_segmentation['Is_Outlier'] == 0]
customer_data_cleaned =
customer_data_cleaned.drop(columns=['Outlier_Scores', 'Is_Outlier'])
customer_data_cleaned.reset_index(drop=True, inplace=True)

customer_data_cleaned.shape[0]
```

```
4070
```

```
sns.set_style('whitegrid')
corr = customer_data_cleaned.drop(columns=['CustomerID']).corr()
colors = ['#004c6d', '#005b8e', 'white', '#ffcaa8', '#ff6200']
my_cmap = LinearSegmentedColormap.from_list('custom_map', colors,
N=256)
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, k=1)] = True
plt.figure(figsize=(12, 10))
sns.heatmap(corr, mask=mask, cmap=my_cmap, annot=True, center=0,
fmt='.2f', linewidths=2)
plt.title('Correlation Matrix', fontsize=14)
plt.show()
```



Looking at the heatmap, we can see that there are some pairs of variables that have high correlations, for instance:

- Monthly_Spending_Mean and Average_Transaction_Value
- Total_Spend and Total_Products_Purchased
- Total_Transactions and Total_Spend
- Total_Transactions and Total_Products_Purchased

Step-7: Feature Scaling

```
scaler = StandardScaler()
columns_to_exclude = ['CustomerID', 'Is_UK', 'Day_Of_Week']
columns_to_scale =
customer_data_cleaned.columns.difference(columns_to_exclude)
customer_data_scaled = customer_data_cleaned.copy()
```

```
customer_data_scaled[columns_to_scale] =
scaler.fit_transform(customer_data_scaled[columns_to_scale])
customer_data_scaled.head()
customer_data_scaled.set_index('CustomerID', inplace=True)
customer_data_scaled.head()
```

```
            last_purchased  quantity     price  Unique_Products  \
CustomerID
12346.0           2.365508 -0.488356 -0.807586        -0.923777
12347.0          -0.907055  0.759515  2.300435         0.836626
12349.0          -0.744947 -0.737930  0.243480         0.301602
12350.0           2.213531 -0.737930 -0.595289        -0.664894
12352.0          -0.562575  1.009089  0.104924         0.042719

            Average_Days_Between_Purchases  Day_Of_Week      Hour
Is_UK  \
CustomerID

12346.0                          -0.335862            1 -1.090209
1
12347.0                          -0.132136            1  0.646955
0
12349.0                          -0.335862            0 -1.524500
0
12350.0                          -0.335862            2  1.515537
0
12352.0                          -0.019396            1  0.646955
0

            Cancellation_Frequency  Cancellation_Rate
Monthly_Spending_Mean  \
CustomerID

12346.0                   0.437337           0.414129              -
1.100976
12347.0                  -0.545525          -0.430724
0.758902
12349.0                  -0.545525          -0.430724
3.301822
12350.0                  -0.545525          -0.430724              -
0.211687
12352.0                   0.437337          -0.219511              -
0.145375

            Spending_Trend
CustomerID
12346.0           0.087764
12347.0           0.112453
12349.0           0.087764
```

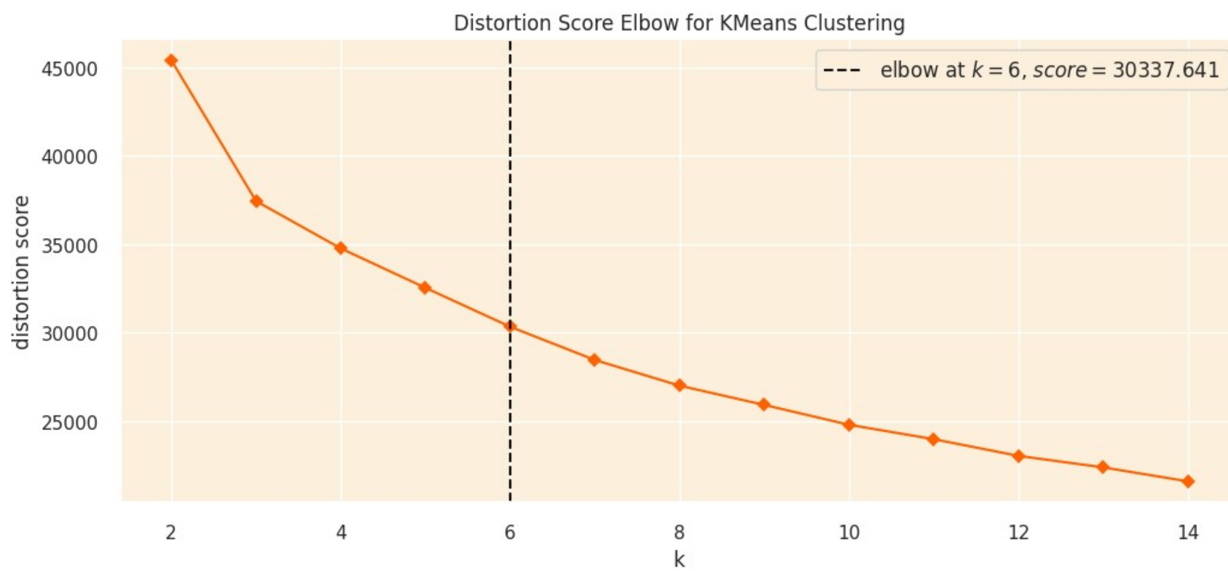
```
12350.0            0.087764
12352.0            0.139227
```

Step-8: K-Means Clustering

8.1: Elbow Method

The Elbow Method is a technique for identifying the ideal number of clusters in a dataset. It involves iterating through the data, generating clusters for various values of k. The k-means algorithm calculates the sum of squared distances between each data point and its assigned cluster centroid, known as the inertia or WCSS score. By plotting the inertia score against the k value, we create a graph that typically exhibits an elbow shape, hence the name "Elbow Method". The elbow point represents the k-value where the reduction in inertia achieved by increasing k becomes negligible, indicating the optimal stopping point for the number of clusters.

```python
sns.set(style='darkgrid', rc={'axes.facecolor': '#fcf0dc'})
sns.set_palette(['#ff6200'])
km = KMeans(init='k-means++', n_init=10, max_iter=100, random_state=0)
fig, ax = plt.subplots(figsize=(12, 5))
visualizer = KElbowVisualizer(km, k=(2, 15), timings=False, ax=ax)
visualizer.fit(customer_data_scaled)
visualizer.show()
```




```
<Axes: title={'center': 'Distortion Score Elbow for KMeans
Clustering'}, xlabel='k', ylabel='distortion score'>
```

Determining the ideal k value for the KMeans clustering algorithm involves identifying the elbow point. Using the YellowBrick library, the Elbow method suggests k=5 as optimal, though the elbow is not very distinct—a common scenario in real-world data. The inertia consistently decreases up to k=5, hinting at an optimal value within the range of 3 to 7. To refine this further, we'll leverage silhouette analysis, a cluster quality assessment method. Business insights can also play a role in selecting a practical k value within this range.

8.2: Sillehoute Analysis

The Silhouette Method is an approach to find the optimal number of clusters in a dataset by evaluating the consistency within clusters and their separation from other clusters. It computes the silhouette coefficient for each data point, which measures how similar a point is to its own cluster compared to other clusters.

```
def silhouette_analysis(df, start_k, stop_k, figsize=(15, 16)):
    """
    Perform Silhouette analysis for a range of k values and visualize
the results.
    """

    # Set the style and context for a cleaner appearance
    sns.set(style="whitegrid")
    sns.set_context("notebook")

    plt.figure(figsize=figsize)

    grid = gridspec.GridSpec(stop_k - start_k + 1, 2)

    first_plot = plt.subplot(grid[0, :])

    # New color palette
    sns.set_palette(['#FF5733', '#FFBF00', '#33FF57', '#5733FF',
'#FF3366'])

    silhouette_scores = []

    # Iterate through the range of k values
    for k in range(start_k, stop_k + 1):
        km = KMeans(n_clusters=k, init='k-means++', n_init=10,
max_iter=100, random_state=0)
        km.fit(df)
        labels = km.predict(df)
        score = silhouette_score(df, labels)
        silhouette_scores.append(score)

    best_k = start_k + silhouette_scores.index(max(silhouette_scores))

    plt.plot(range(start_k, stop_k + 1), silhouette_scores,
marker='o')
    plt.xticks(range(start_k, stop_k + 1))
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Silhouette score')
    plt.title('Average Silhouette Score for Different k Values',
fontsize=15)
```

```
    optimal_k_text = f'The k value with the highest Silhouette score
is: {best_k}'
    plt.text(10, 0.23, optimal_k_text, fontsize=12,
verticalalignment='bottom',
             horizontalalignment='left',
bbox=dict(facecolor='#fcc36d', edgecolor='#ff6200', boxstyle='round,
pad=0.5'))

    # New color palette for silhouette plots
    colors = sns.set_palette(['#FF5733', '#FFBF00', '#33FF57',
'#5733FF', '#FF3366'])

    for i in range(start_k, stop_k + 1):
        km = KMeans(n_clusters=i, init='k-means++', n_init=10,
max_iter=100, random_state=0)
        row_idx, col_idx = divmod(i - start_k, 2)
        ax = plt.subplot(grid[row_idx + 1, col_idx])
        visualizer = SilhouetteVisualizer(km, colors=colors, ax=ax)
        visualizer.fit(df)
        score = silhouette_score(df, km.labels_)
        ax.text(0.97, 0.02, f'Silhouette Score: {score:.2f}',
fontsize=12, \
                ha='right', transform=ax.transAxes, color='red')
        ax.set_title(f'Silhouette Plot for {i} Clusters', fontsize=15)

    plt.tight_layout()
    plt.show()

silhouette_analysis(customer_data_scaled, 2, 12, figsize=(20, 50))
```


The k value with the highest Silhouette score is: 3
Average Silhouette Score for Different k Values
Silhouette score
Number of clusters (k)
Silhouette Plot for 2 Clusters
Silhouette Score: 0.19
Silhouette Plot for 3 Clusters
Silhouette Score: 0.21
Silhouette Plot for 4 Clusters
Silhouette Score: 0.17
Silhouette Plot for 5 Clusters
Silhouette Score: 0.17
Silhouette Plot for 6 Clusters
Silhouette Score: 0.18
Silhouette Plot for 7 Clusters
Silhouette Score: 0.18
Silhouette Plot for 8 Clusters
Silhouette Score: 0.18
Silhouette Plot for 9 Clusters
Silhouette Score: 0.17
Silhouette Plot for 10 Clusters
Silhouette Plot for 11 Clusters

```python
#thus we choose value of k as 3 as it has the best score
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10,
max_iter=100, random_state=0)
kmeans.fit(customer_data_scaled)
cluster_frequencies = Counter(kmeans.labels_)
label_mapping = {label: new_label for new_label, (label, _) in
                 enumerate(cluster_frequencies.most_common())}
label_mapping = {v: k for k, v in {2: 1, 1: 0, 0: 2}.items()}
new_labels = np.array([label_mapping[label] for label in
kmeans.labels_])
customer_data_cleaned['cluster'] = new_labels
customer_data_scaled['cluster'] = new_labels

customer_data_cleaned.head()
```

```
  CustomerID  last_purchased  quantity    price  Unique_Products  \
0    12346.0             325         2      0.0                1
1    12347.0               2         7   4310.0              103
2    12349.0              18         1  1457.55               72
3    12350.0             310         1    294.4               16
4    12352.0              36         8  1265.41               57

   Average_Days_Between_Purchases  Day_Of_Week  Hour  Is_UK  \
0                             0.0            1    10      1
1                        2.016575            1    14      0
2                             0.0            0     9      0
3                             0.0            2    16      0
4                         3.13253            1    14      0

   Cancellation_Frequency  Cancellation_Rate  Monthly_Spending_Mean  \
0                       1                0.5                    0.0
1                       0                0.0             615.714286
2                       0                0.0                1457.55
3                       0                0.0                  294.4
4                       1              0.125               316.3525

   Spending_Trend  cluster
0             0.0        2
1        4.486071        2
2             0.0        2
3             0.0        2
4           9.351        2
```

Step-9: Evaluation

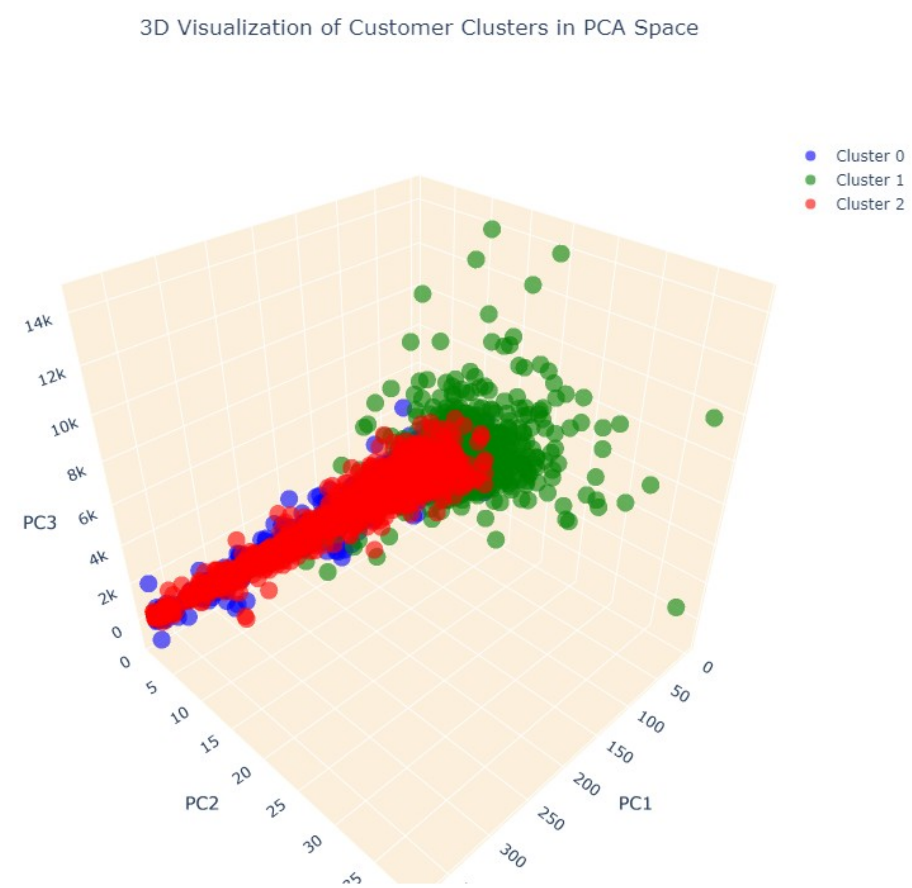
9.1: 3D Visualization of Top Principal Components

```python
customer_data_scaled.head()
```

```
               last_purchased  quantity     price  Unique_Products  \
CustomerID
12346.0              2.365508 -0.488356 -0.807586        -0.923777
12347.0             -0.907055  0.759515  2.300435         0.836626
12349.0             -0.744947 -0.737930  0.243480         0.301602
12350.0              2.213531 -0.737930 -0.595289        -0.664894
12352.0             -0.562575  1.009089  0.104924         0.042719

               Average_Days_Between_Purchases  Day_Of_Week      Hour
Is_UK  \
CustomerID

12346.0                             -0.335862            1 -1.090209
1
12347.0                             -0.132136            1  0.646955
0
12349.0                             -0.335862            0 -1.524500
0
12350.0                             -0.335862            2  1.515537
0
12352.0                             -0.019396            1  0.646955
0

               Cancellation_Frequency  Cancellation_Rate
Monthly_Spending_Mean  \
CustomerID

12346.0                      0.437337           0.414129              -
1.100976
12347.0                     -0.545525          -0.430724
0.758902
12349.0                     -0.545525          -0.430724
3.301822
12350.0                     -0.545525          -0.430724              -
0.211687
12352.0                      0.437337          -0.219511              -
0.145375

               Spending_Trend  cluster
CustomerID
12346.0              0.087764        2
12347.0              0.112453        2
12349.0              0.087764        2
12350.0              0.087764        2
12352.0              0.139227        2
```

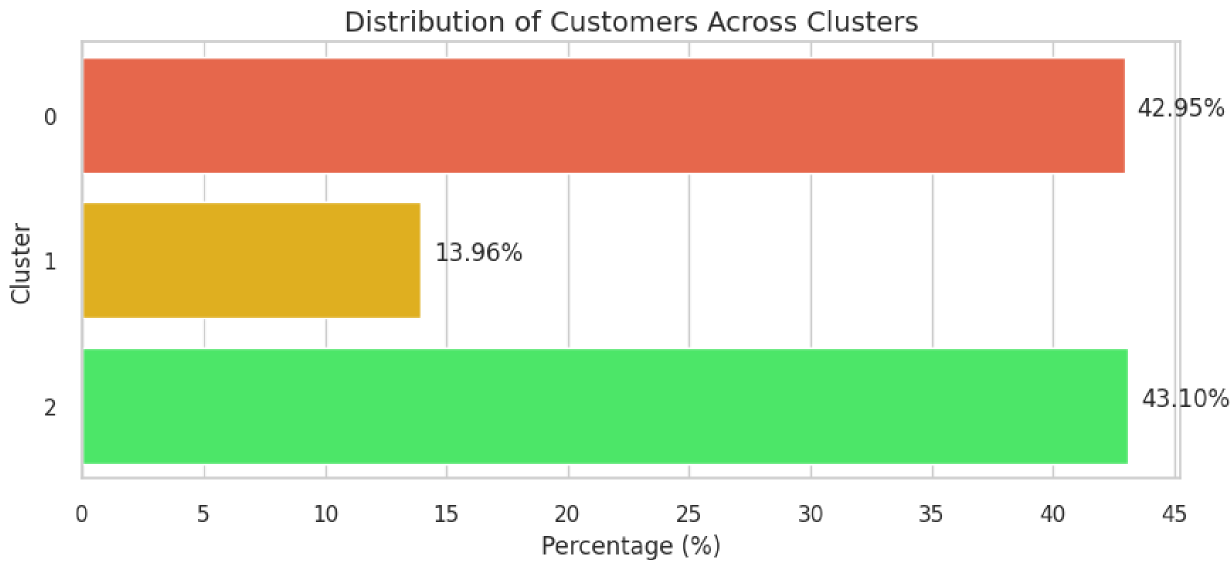
```
import plotly.graph_objects as go

# Create separate data frames for each cluster
cluster_0 = customer_data_cleaned[customer_data_cleaned['cluster'] ==
```

```
0]
cluster_1 = customer_data_cleaned[customer_data_cleaned['cluster'] ==
1]
cluster_2 = customer_data_cleaned[customer_data_cleaned['cluster'] ==
2]

# Create a 3D scatter plot
fig = go.Figure()

# Add data points for each cluster separately and specify the color
fig.add_trace(go.Scatter3d(
    x=cluster_0['last_purchased'], y=cluster_0['quantity'],
z=cluster_0['price'],
    mode='markers', marker=dict(color='blue', size=8, opacity=0.6),
name='Cluster 0'
))

fig.add_trace(go.Scatter3d(
    x=cluster_1['last_purchased'], y=cluster_1['quantity'],
z=cluster_1['price'],
    mode='markers', marker=dict(color='green', size=8, opacity=0.6),
name='Cluster 1'
))

fig.add_trace(go.Scatter3d(
    x=cluster_2['last_purchased'], y=cluster_2['quantity'],
z=cluster_2['price'],
    mode='markers', marker=dict(color='red', size=8, opacity=0.6),
name='Cluster 2'
))

# Set the title and layout details
fig.update_layout(
    title=dict(text='3D Visualization of Customer Clusters in PCA
Space', x=0.5),
    scene=dict(
        xaxis=dict(backgroundcolor="#fcf0dc", gridcolor='white',
title='PC1'),
        yaxis=dict(backgroundcolor="#fcf0dc", gridcolor='white',
title='PC2'),
        zaxis=dict(backgroundcolor="#fcf0dc", gridcolor='white',
title='PC3'),
    ),
    width=900,
    height=800,
    legend=dict(x=0.85, y=0.95)
)

# Show the plot
fig.show()
```




```
# Calculate the percentage of customers in each cluster
cluster_percentage =
(customer_data_cleaned['cluster'].value_counts(normalize=True) *
100).reset_index()
cluster_percentage.columns = ['Cluster', 'Percentage']
cluster_percentage.sort_values(by='Cluster', inplace=True)

# Create a horizontal bar plot
plt.figure(figsize=(10, 4))
sns.barplot(x='Percentage', y='Cluster', data=cluster_percentage,
orient='h')

# Adding percentages on the bars
for index, value in enumerate(cluster_percentage['Percentage']):
    plt.text(value+0.5, index, f'{value:.2f}%')

plt.title('Distribution of Customers Across Clusters', fontsize=14)
plt.xticks(ticks=np.arange(0, 50, 5))
plt.xlabel('Percentage (%)')
```

```python
# Show the plot
plt.show()
```




```python
num_observations = len(customer_data_scaled)

X = customer_data_scaled.drop('cluster', axis=1)
clusters = customer_data_scaled['cluster']

sil_score = silhouette_score(X, clusters)
calinski_score = calinski_harabasz_score(X, clusters)
davies_score = davies_bouldin_score(X, clusters)

table_data = [
    ["Number of Observations", num_observations],
    ["Silhouette Score", sil_score],
    ["Calinski Harabasz Score", calinski_score],
    ["Davies Bouldin Score", davies_score]
]

print(tabulate(table_data, headers=["Metric", "Value"],
tablefmt='pretty'))
```

```
+-------------------------+--------------------+
|         Metric          |       Value        |
+-------------------------+--------------------+
| Number of Observations  |        4070        |
|    Silhouette Score     | 0.2125150405732648 |
| Calinski Harabasz Score | 955.1164407844061  |
|  Davies Bouldin Score   | 1.5206604567569066 |
+-------------------------+--------------------+
```

To further scrutinize the quality of our clustering, I will employ the following metrics:

- Silhouette Score: A measure to evaluate the separation distance between the clusters. Higher values indicate better cluster separation. It ranges from -1 to 1.
- Calinski Harabasz Score: This score is used to evaluate the dispersion between and within clusters. A higher score indicates better defined clusters.
- Davies Bouldin Score: It assesses the average similarity between each cluster and its most similar cluster. Lower values indicate better cluster separation.

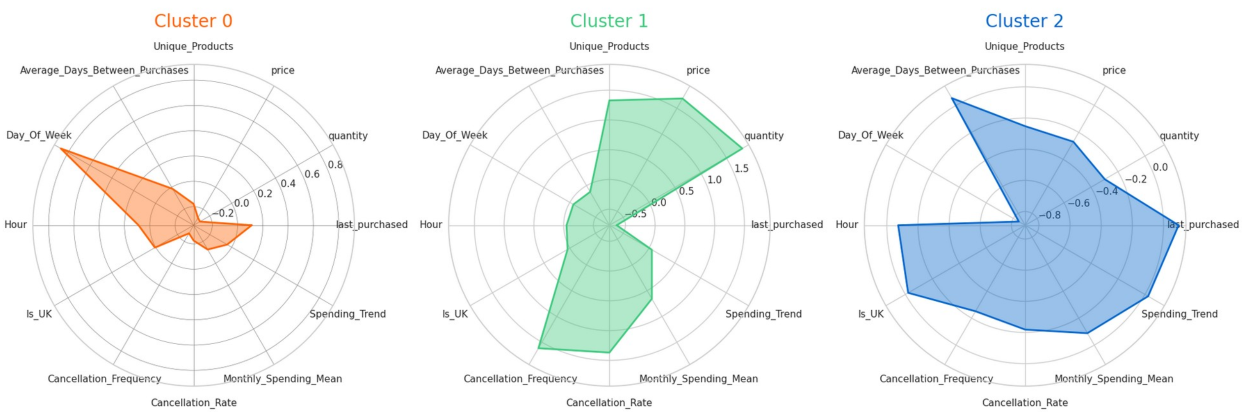
Step-9: Cluster Analysis

9.1: Radar Chart Approach

We created radar charts to visualize the centroid values of each cluster across different features. This can give a quick visual comparison of the profiles of different clusters. To construct the radar charts, it's essential to first compute the centroid for each cluster. This centroid represents the mean value for all features within a specific cluster. Subsequently, I will display these centroids on the radar charts, facilitating a clear visualization of the central tendencies of each feature across the various clusters

```
df_customer = customer_data_scaled

scaler = StandardScaler()
df_customer_standardized =
scaler.fit_transform(df_customer.drop(columns=['cluster'], axis=1))

df_customer_standardized = pd.DataFrame(df_customer_standardized,
columns=df_customer.columns[:-1], index=df_customer.index)
df_customer_standardized['cluster'] = df_customer['cluster']

cluster_centroids = df_customer_standardized.groupby('cluster').mean()

def create_radar_chart(ax, angles, data, color, cluster):
    ax.fill(angles, data, color=color, alpha=0.4)
    ax.plot(angles, data, color=color, linewidth=2, linestyle='solid')
    ax.set_title(f'Cluster {cluster}', size=20, color=color, y=1.1)

labels = np.array(cluster_centroids.columns)
num_vars = len(labels)
angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()

labels = np.concatenate((labels, [labels[0]]))
angles += angles[:1]

fig, ax = plt.subplots(figsize=(20, 10), subplot_kw=dict(polar=True),
nrows=1, ncols=len(cluster_centroids))

colors = ['#ff6200', '#3fca7f', '#0066cc']
for i, color in enumerate(colors):
    if i < len(cluster_centroids):
        data = cluster_centroids.loc[i].tolist()
```

```python
        data += data[:1]
        create_radar_chart(ax[i], angles, data, color, i)

for axis in ax:
    axis.set_xticks(angles[:-1])
    axis.set_xticklabels(labels[:-1])

ax[0].grid(color='grey', linewidth=0.5)

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



9.2 Histogram Approach

To validate the characteristics outlined in the radar charts, we can generate histograms for each feature categorized by cluster labels. These histograms enable a visual examination of the distribution of feature values within each cluster. This process aids in confirming or refining the identified profiles from the radar charts.

```python
features = customer_data_cleaned.columns[1:-1]
clusters = customer_data_cleaned['cluster'].unique()
clusters.sort()

n_rows = len(features)
n_cols = len(clusters)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 3*n_rows))

for i, feature in enumerate(features):
    for j, cluster in enumerate(clusters):
        data = customer_data_cleaned[customer_data_cleaned['cluster'] == cluster][feature]
        axes[i, j].hist(data, bins=20, color=colors[j], edgecolor='w', alpha=0.7)
        axes[i, j].set_title(f'Cluster {cluster} - {feature}', fontsize=15)
        axes[i, j].set_xlabel('')
        axes[i, j].set_ylabel('')
```

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


Cluster 0 - last_purchased
Cluster 1 - last_purchased
Cluster 2 - last_purchased
Cluster 0 - quantity
Cluster 1 - quantity
Cluster 2 - quantity
Cluster 0 - price
Cluster 1 - price
Cluster 2 - price
Cluster 0 - Unique_Products
Cluster 1 - Unique_Products
Cluster 2 - Unique_Products
Cluster 0 - Average_Days_Between_Purchases
Cluster 1 - Average_Days_Between_Purchases
Cluster 2 - Average_Days_Between_Purchases
Cluster 0 - Day_Of_Week
Cluster 1 - Day_Of_Week
Cluster 2 - Day_Of_Week
Cluster 0 - Hour
Cluster 1 - Hour
Cluster 2 - Hour
Cluster 0 - Is_UK
Cluster 1 - Is_UK
Cluster 2 - Is_UK
Cluster 0 - Cancellation_Frequency
Cluster 1 - Cancellation_Frequency
Cluster 2 - Cancellation_Frequency
Cluster 0 - Cancellation_Rate
Cluster 1 - Cancellation_Rate
Cluster 2 - Cancellation_Rate

Cluster 0 (Red Chart): Customer Type: Casual Weekend Shoppers

- People in this group don't spend a lot and don't shop frequently. They buy a small number of items during their visits.
- They seem to like shopping on weekends, as they do it more on those days.
- Their spending doesn't change much from month to month; it stays somewhat steady and is not very high.
- These customers rarely cancel their purchases; they don't do it often.
- When they do buy something, they usually don't spend a lot each time.

Cluster 1 (Green Chart): Customer Type: Early Rise Shoppers

- People in this group love to spend a lot and buy many different things.
- They shop a lot, but they also cancel their purchases frequently.
- They don't wait much between purchases; they like shopping early in the day.
- Their spending from month to month changes a lot, meaning it's not very consistent.
- Even though they spend a lot, their trend of spending doesn't seem to be going up; it might even be going down over time.

Cluster 2 (Blue Chart): Customer Type: Occasional Big Spenders

- People in this group spend a decent amount, but they don't shop very often. There's a longer gap between their purchases.
- They've been spending more and more as time goes on; their spending trend is going up.
- These customers like shopping later in the day, and most of them live in the UK.
- They cancel some of their purchases, but not too many. It happens every now and then.
- When they do buy something, they usually spend a good amount each time.

STEP-10: RECOMMENDATION SYSTEM

Firstly, we identify and extract the CustomerIDs associated with outliers, removing their transactions from the main dataframe. Ensuring consistent data types for CustomerID across both dataframes, we proceed to merge the transaction data with customer data, incorporating cluster information for each transaction. Subsequently, we determine the top 10 best-selling products in each cluster based on total quantity sold. Building on this, we create a record of products purchased by each customer in each cluster. With this information, we generate personalized recommendations for each customer within their respective clusters. The recommendation process involves identifying products already purchased by the customer and suggesting the top 3 products in the best-selling list that the customer hasn't bought yet. Finally, we compile these recommendations into a dataframe and merge it with the original customer data, providing a comprehensive view that includes cluster information and personalized product suggestions.

```
# Filter out outliers from the original data
outlier_customer_ids = 
outliers_data['CustomerID'].astype('float').unique()
df_filtered = df[~df['CustomerID'].isin(outlier_customer_ids)]

# Ensure CustomerID is in float format for merging
```

```
customer_data_cleaned['CustomerID'] =
customer_data_cleaned['CustomerID'].astype('float')

# Merge filtered data with customer data
merged_data = df_filtered.merge(customer_data_cleaned[['CustomerID',
'cluster']], on='CustomerID', how='inner')

# Find the best-selling products for each cluster
best_selling_products = merged_data.groupby(['cluster', 'StockCode',
'Description'])['Quantity'].sum().reset_index()
best_selling_products =
best_selling_products.sort_values(by=['cluster', 'Quantity'],
ascending=[True, False])

# Select the top 10 products per cluster
top_products_per_cluster =
best_selling_products.groupby('cluster').head(10)

# Group customer purchases
customer_purchases = merged_data.groupby(['CustomerID', 'cluster',
'StockCode'])['Quantity'].sum().reset_index()

# Generate recommendations for each customer in each cluster
recommendations = []
for cluster in top_products_per_cluster['cluster'].unique():
    top_products =
top_products_per_cluster[top_products_per_cluster['cluster'] ==
cluster]
    customers_in_cluster =
customer_data_cleaned[customer_data_cleaned['cluster'] == cluster]
['CustomerID']

    for customer in customers_in_cluster:
        customer_purchased_products =
customer_purchases[(customer_purchases['CustomerID'] == customer) &

(customer_purchases['cluster'] == cluster)]['StockCode'].tolist()

        top_products_not_purchased =
top_products[~top_products['StockCode'].isin(customer_purchased_produc
ts)]
        top_3_products_not_purchased =
top_products_not_purchased.head(3)

        recommendations.append([customer, cluster] +
top_3_products_not_purchased[['StockCode',
'Description']].values.flatten().tolist())

# Create a DataFrame for recommendations
recommendations_df = pd.DataFrame(recommendations,
```

```
columns=['CustomerID', 'cluster', 'Rec1_StockCode',
'Rec1_Description', \
                                                 'Rec2_StockCode',
'Rec2_Description', 'Rec3_StockCode', 'Rec3_Description'])

# Merge recommendations with customer data
customer_data_with_recommendations =
customer_data_cleaned.merge(recommendations_df, on=['CustomerID',
'cluster'], how='right')

# Display a sample of the final DataFrame
customer_data_with_recommendations.set_index('CustomerID').iloc[:, -
6:].sample(10, random_state=0)
```

```
           Rec1_StockCode                   Rec1_Description
Rec2_StockCode  \
CustomerID

16212.0             16014        SMALL CHINESE STYLE SCISSOR
84077
15089.0             16014        SMALL CHINESE STYLE SCISSOR
84077
17674.0             16014        SMALL CHINESE STYLE SCISSOR
84077
14130.0             16014        SMALL CHINESE STYLE SCISSOR
84077
14670.0             84077  WORLD WAR 2 GLIDERS ASSTD DESIGNS
84879
14419.0             16014        SMALL CHINESE STYLE SCISSOR
84077
17588.0             16014        SMALL CHINESE STYLE SCISSOR
84077
13000.0             84077  WORLD WAR 2 GLIDERS ASSTD DESIGNS
84879
15058.0             16014        SMALL CHINESE STYLE SCISSOR
84077
14158.0             84077  WORLD WAR 2 GLIDERS ASSTD DESIGNS
84879

                             Rec2_Description Rec3_StockCode  \
CustomerID
16212.0     WORLD WAR 2 GLIDERS ASSTD DESIGNS          17003
15089.0     WORLD WAR 2 GLIDERS ASSTD DESIGNS          17003
17674.0     WORLD WAR 2 GLIDERS ASSTD DESIGNS          17003
14130.0     WORLD WAR 2 GLIDERS ASSTD DESIGNS          17003
14670.0         ASSORTED COLOUR BIRD ORNAMENT          15036
14419.0     WORLD WAR 2 GLIDERS ASSTD DESIGNS          17003
17588.0     WORLD WAR 2 GLIDERS ASSTD DESIGNS          84879
13000.0         ASSORTED COLOUR BIRD ORNAMENT          15036
15058.0     WORLD WAR 2 GLIDERS ASSTD DESIGNS          17003
```

```
14158.0         ASSORTED COLOUR BIRD ORNAMENT         15036

                               Rec3_Description
CustomerID
16212.0                      BROCADE RING PURSE
15089.0                      BROCADE RING PURSE
17674.0                      BROCADE RING PURSE
14130.0                      BROCADE RING PURSE
14670.0               ASSORTED COLOURS SILK FAN
14419.0                      BROCADE RING PURSE
17588.0           ASSORTED COLOUR BIRD ORNAMENT
13000.0               ASSORTED COLOURS SILK FAN
15058.0                      BROCADE RING PURSE
14158.0               ASSORTED COLOURS SILK FAN
```