customer-segmentation-using-rando

April 1, 2024

Step 1.1 | Importing Necessary Libraries   Tabel of Contents

First of all, I will import all the necessary libraries that we will use throughout the project. This generally includes libraries for data manipulation, data visualization, and others based on the specific needs of the project:

[1]:
```
# Ignore warnings
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
```

[2]:
```
# Initialize Plotly for use in the notebook
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

[3]:
```
# Configure Seaborn plot styles: Set background color and use dark grid
sns.set(rc={'axes.facecolor': '#fcf0dc'}, style='darkgrid')
```

Step 1.2 | Loading the Dataset   Tabel of Contents

Next, I will load the dataset into a pandas DataFrame which will facilitate easy manipulation and analysis:

[4]:
```
df = pd.read_csv('/kaggle/input/ecommerce-data/data.csv', encoding="ISO-8859-1")
```

Step 2.1 | Dataset Overview   Tabel of Contents

First I will perform a preliminary analysis to understand the structure and types of data columns:

[5]:
```
df.head(10)
```

[5]:
```
  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

      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
```

[6]:
```
df.info()
```

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

```
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

Step 2.2 | Summary Statistics Tabel of Contents

[7]:
```
# Summary statistics for numerical variables
df.describe().T
```

[7]:
```
               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
```

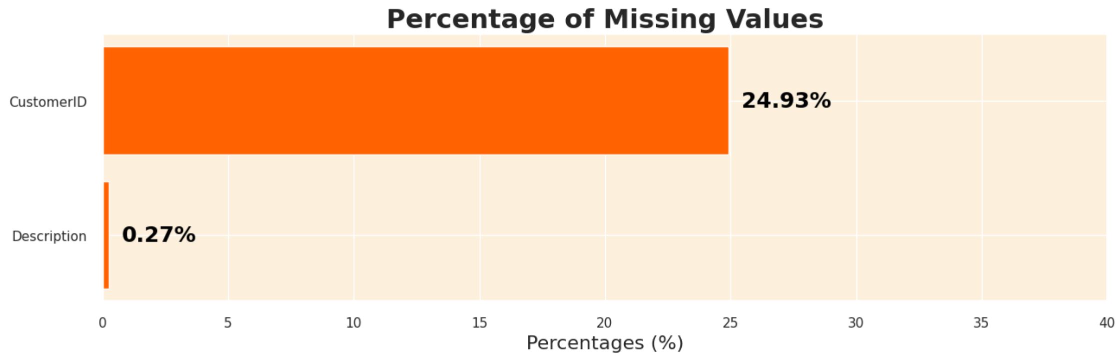
[8]:
```
# Summary statistics for categorical variables
df.describe(include='object').T
```

[8]:
```
              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
```

Step 3.1 | Handling Missing Values Tabel of Contents

[9]:
```
# Calculating the percentage of missing values for each column
missing_data = df.isnull().sum()
missing_percentage = (missing_data[missing_data > 0] / df.shape[0]) * 100

# Prepare values
missing_percentage.sort_values(ascending=True, inplace=True)

# Plot the barh chart
fig, ax = plt.subplots(figsize=(15, 4))
ax.barh(missing_percentage.index, missing_percentage, color='#ff6200')

# Annotate the values and indexes
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=18, color='black')
```

```python
# Set x-axis limit
ax.set_xlim([0, 40])

# Add title and xlabel
plt.title("Percentage of Missing Values", fontweight='bold', fontsize=22)
plt.xlabel('Percentages (%)', fontsize=16)
plt.show()
```



[10]:
```python
# Extracting rows with missing values in 'CustomerID' or 'Description' columns
df[df['CustomerID'].isnull() | df['Description'].isnull()].head()
```

[10]:
```
      InvoiceNo StockCode                      Description  Quantity  \
622      536414     22139                              NaN        56
1443     536544     21773  DECORATIVE ROSE BATHROOM BOTTLE         1
1444     536544     21774  DECORATIVE CATS BATHROOM BOTTLE         2
1445     536544     21786               POLKADOT RAIN HAT          4
1446     536544     21787            RAIN PONCHO RETROSPOT         2

         InvoiceDate  UnitPrice  CustomerID         Country
622   12/1/2010 11:52       0.00         NaN  United Kingdom
1443  12/1/2010 14:32       2.51         NaN  United Kingdom
1444  12/1/2010 14:32       2.51         NaN  United Kingdom
1445  12/1/2010 14:32       0.85         NaN  United Kingdom
1446  12/1/2010 14:32       1.66         NaN  United Kingdom
```

[11]:
```python
# Removing rows with missing values in 'CustomerID' and 'Description' columns
df = df.dropna(subset=['CustomerID', 'Description'])
```

[12]:
```python
# Verifying the removal of missing values
df.isnull().sum().sum()
```

[12]: 0

Step 3.2 | Handling Duplicates   Tabel of Contents

[13]:
```python
# Finding duplicate rows (keeping all instances)
duplicate_rows = df[df.duplicated(keep=False)]

# Sorting the data by certain columns to see the duplicate rows next to each␣
↪other
duplicate_rows_sorted = duplicate_rows.sort_values(by=['InvoiceNo',␣
↪'StockCode', 'Description', 'CustomerID', 'Quantity'])

# Displaying the first 10 records
duplicate_rows_sorted.head(10)
```

[13]:
```
    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  12/1/2010 11:45       1.25     17908.0  United Kingdom
517  12/1/2010 11:45       1.25     17908.0  United Kingdom
485  12/1/2010 11:45       4.95     17908.0  United Kingdom
539  12/1/2010 11:45       4.95     17908.0  United Kingdom
489  12/1/2010 11:45       2.10     17908.0  United Kingdom
527  12/1/2010 11:45       2.10     17908.0  United Kingdom
521  12/1/2010 11:45       2.95     17908.0  United Kingdom
537  12/1/2010 11:45       2.95     17908.0  United Kingdom
578  12/1/2010 11:49       1.65     17920.0  United Kingdom
598  12/1/2010 11:49       1.65     17920.0  United Kingdom
```

[14]:
```python
# Displaying the number of duplicate rows
print(f"The dataset contains {df.duplicated().sum()} duplicate rows that need␣
↪to be removed.")

# Removing duplicate rows
df.drop_duplicates(inplace=True)
```

```
The dataset contains 5225 duplicate rows that need to be removed.
```

[15]:
```python
# Getting the number of rows in the dataframe
df.shape[0]
```

[15]: 401604

Step 3.3 | Treating Cancelled Transactions   Tabel of Contents

[16]:
```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)
```

[16]:

	Quantity	UnitPrice
count	8872.000000	8872.000000
mean	-30.774910	18.899512
std	1172.249902	445.190864
min	-80995.000000	0.010000
25%	-6.000000	1.450000
50%	-2.000000	2.950000
75%	-1.000000	4.950000
max	-1.000000	38970.000000

[17]:
```python
# Finding the percentage of cancelled transactions
cancelled_percentage = (cancelled_transactions.shape[0] / df.shape[0]) * 100

# Printing the percentage of cancelled transactions
print(f"The percentage of cancelled transactions in the dataset is:␣
↪{cancelled_percentage:.2f}%")
```

```
The percentage of cancelled transactions in the dataset is: 2.21%
```

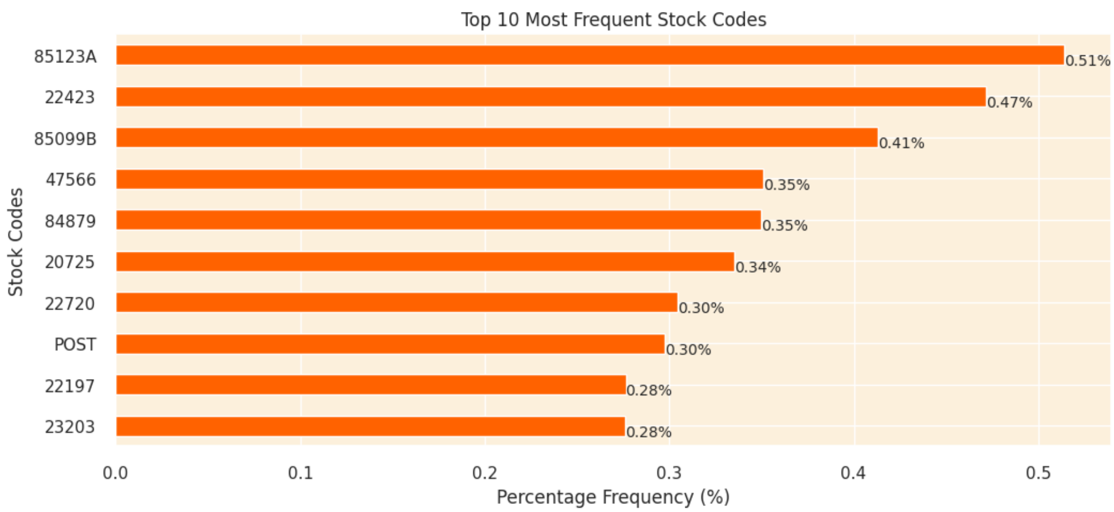
Step 3.4 | Correcting StockCode Anomalies   Tabel of Contents

[18]:
```python
# Finding the number of unique stock codes
unique_stock_codes = df['StockCode'].nunique()

# Printing the number of unique stock codes
print(f"The number of unique stock codes in the dataset is:␣
↪{unique_stock_codes}")
```

```
The number of unique stock codes in the dataset is: 3684
```

[19]:
```python
# Finding the top 10 most frequent stock codes
top_10_stock_codes = df['StockCode'].value_counts(normalize=True).head(10) * 100

# Plotting the top 10 most frequent stock codes
plt.figure(figsize=(12, 5))
```

```
top_10_stock_codes.plot(kind='barh', color='#ff6200')

# Adding the percentage frequency on the bars
for index, value in enumerate(top_10_stock_codes):
    plt.text(value, index+0.25, f'{value:.2f}%', fontsize=10)

plt.title('Top 10 Most Frequent Stock Codes')
plt.xlabel('Percentage Frequency (%)')
plt.ylabel('Stock Codes')
plt.gca().invert_yaxis()
plt.show()
```



[20]:

```
# Finding the number of numeric characters in each unique stock code
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()

# Printing the value counts for unique stock codes
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
```

[21]:
```
# Finding and printing the stock codes with 0 and 1 numeric characters
anomalous_stock_codes = [code for code in unique_stock_codes if sum(c.isdigit()␣
↪for c in str(code)) in (0, 1)]

# Printing each stock code on a new line
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
```

[22]:
```
# Calculating the percentage of records with these stock codes
percentage_anomalous = (df['StockCode'].isin(anomalous_stock_codes).sum() /␣
↪len(df)) * 100

# Printing the percentage
print(f"The percentage of records with anomalous stock codes in the dataset is:␣
↪{percentage_anomalous:.2f}%")
```

```
The percentage of records with anomalous stock codes in the dataset is: 0.48%
```

[23]:
```
# Removing rows with anomalous stock codes from the dataset
df = df[~df['StockCode'].isin(anomalous_stock_codes)]
```

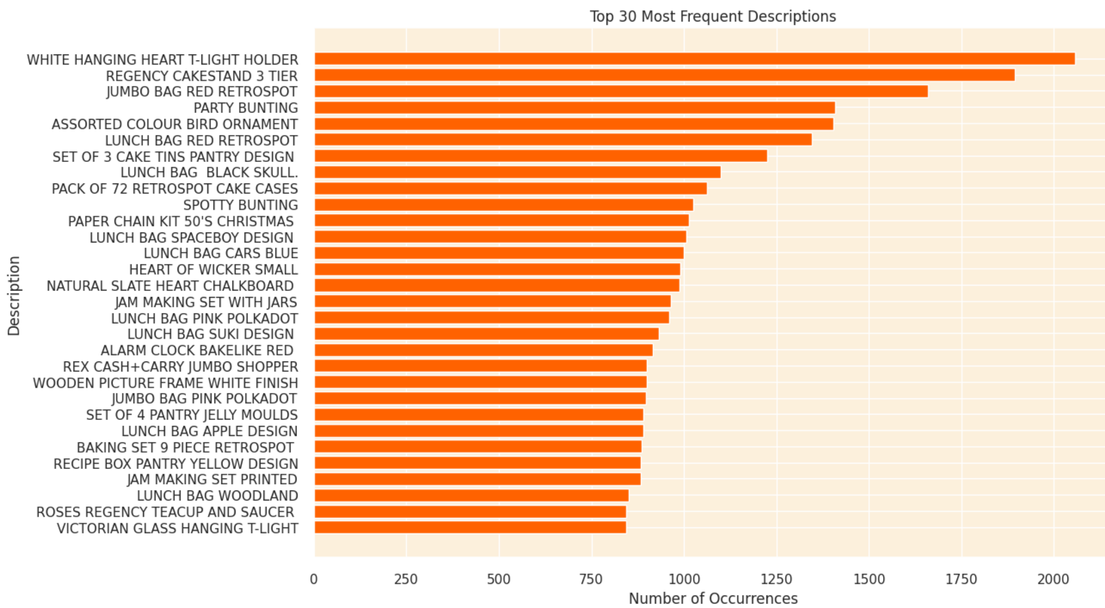
[24]:
```
# Getting the number of rows in the dataframe
df.shape[0]
```

[24]: 399689

Step 3.5 | Cleaning Description Column   Tabel of Contents

[25]:
```
# Calculate the occurrence of each unique description and sort them
description_counts = df['Description'].value_counts()

# Get the top 30 descriptions
top_30_descriptions = description_counts[:30]
```

```
# Plotting
plt.figure(figsize=(12,8))
plt.barh(top_30_descriptions.index[::-1], top_30_descriptions.values[::-1],␣
↪color='#ff6200')

# Adding labels and title
plt.xlabel('Number of Occurrences')
plt.ylabel('Description')
plt.title('Top 30 Most Frequent Descriptions')

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



[26]:
```
# Find unique descriptions containing lowercase characters
lowercase_descriptions = df['Description'].unique()
lowercase_descriptions = [desc for desc in lowercase_descriptions if any(char.
↪islower() for char in desc)]

# Print the unique descriptions containing lowercase characters
print("The unique descriptions containing lowercase characters are:")
print("-"*60)
for desc in lowercase_descriptions:
    print(desc)
```

```
The unique descriptions containing lowercase characters are:
------------------------------------------------------------
BAG 500g SWIRLY MARBLES
```

```
POLYESTER FILLER PAD 45x45cm
POLYESTER FILLER PAD 45x30cm
POLYESTER FILLER PAD 40x40cm
FRENCH BLUE METAL DOOR SIGN No
BAG 250g SWIRLY MARBLES
BAG 125g SWIRLY MARBLES
3 TRADITIONAl BISCUIT CUTTERS  SET
NUMBER TILE COTTAGE GARDEN No
FOLK ART GREETING CARD,pack/12
ESSENTIAL BALM 3.5g TIN IN ENVELOPE
POLYESTER FILLER PAD 65CMx65CM
NUMBER TILE VINTAGE FONT No
POLYESTER FILLER PAD 30CMx30CM
POLYESTER FILLER PAD 60x40cm
FLOWERS HANDBAG blue and orange
Next Day Carriage
THE KING GIFT BAG 25x24x12cm
High Resolution Image
```

[27]:
```python
service_related_descriptions = ["Next Day Carriage", "High Resolution Image"]

# Calculate the percentage of records with service-related descriptions
service_related_percentage = df[df['Description'].
 ↪isin(service_related_descriptions)].shape[0] / df.shape[0] * 100

# Print the percentage of records with service-related descriptions
print(f"The percentage of records with service-related descriptions in the␣
 ↪dataset is: {service_related_percentage:.2f}%")

# Remove rows with service-related information in the description
df = df[~df['Description'].isin(service_related_descriptions)]

# Standardize the text to uppercase to maintain uniformity across the dataset
df['Description'] = df['Description'].str.upper()
```

```
The percentage of records with service-related descriptions in the dataset is:
0.02%
```

[28]:
```python
# Getting the number of rows in the dataframe
df.shape[0]
```

[28]: 399606

Step 3.6 | Treating Zero Unit Prices   Tabel of Contents

[29]:
```python
df['UnitPrice'].describe()
```

[29]:
```
count    399606.000000
mean          2.904957
std           4.448796
min           0.000000
25%           1.250000
50%           1.950000
75%           3.750000
max         649.500000
Name: UnitPrice, dtype: float64
```

[30]:
```
df[df['UnitPrice']==0].describe()[['Quantity']]
```

[30]:
```
           Quantity
count     33.000000
mean     420.515152
std     2176.713608
min        1.000000
25%        2.000000
50%       11.000000
75%       36.000000
max    12540.000000
```

[31]:
```
# Removing records with a unit price of zero to avoid potential data entry␣
↪errors
df = df[df['UnitPrice'] > 0]
```

[32]:
```
# Resetting the index of the cleaned dataset
df.reset_index(drop=True, inplace=True)
```

[33]:
```
# Getting the number of rows in the dataframe
df.shape[0]
```

[33]: 399573

Step 4.1.1 | Recency (R)   Tabel of Contents

[34]:
```
# Convert InvoiceDate to datetime type
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

# Convert InvoiceDate to datetime and extract only the date
df['InvoiceDay'] = df['InvoiceDate'].dt.date

# Find the most recent purchase date for each customer
customer_data = df.groupby('CustomerID')['InvoiceDay'].max().reset_index()

# Find the most recent date in the entire dataset
most_recent_date = df['InvoiceDay'].max()
```

```
# Convert InvoiceDay to datetime type before subtraction
customer_data['InvoiceDay'] = pd.to_datetime(customer_data['InvoiceDay'])
most_recent_date = pd.to_datetime(most_recent_date)

# Calculate the number of days since the last purchase for each customer
customer_data['Days_Since_Last_Purchase'] = (most_recent_date -␣
↪customer_data['InvoiceDay']).dt.days

# Remove the InvoiceDay column
customer_data.drop(columns=['InvoiceDay'], inplace=True)
```

[35]:
```
customer_data.head()
```

[35]:

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

[36]:
```
# Calculate the total number of transactions made by each customer
total_transactions = df.groupby('CustomerID')['InvoiceNo'].nunique().
↪reset_index()
total_transactions.rename(columns={'InvoiceNo': 'Total_Transactions'},␣
↪inplace=True)

# Calculate the total number of products purchased by each customer
total_products_purchased = df.groupby('CustomerID')['Quantity'].sum().
↪reset_index()
total_products_purchased.rename(columns={'Quantity':␣
↪'Total_Products_Purchased'}, inplace=True)

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, total_transactions, on='CustomerID')
customer_data = pd.merge(customer_data, total_products_purchased,␣
↪on='CustomerID')

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

[36]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	\
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

   Total_Products_Purchased
0                         0
1                      2458
2                      2332
3                       630
4                       196
```

[37]:

```
# Calculate the total spend by each customer
df['Total_Spend'] = df['UnitPrice'] * df['Quantity']
total_spend = df.groupby('CustomerID')['Total_Spend'].sum().reset_index()

# Calculate the average transaction value for each customer
average_transaction_value = total_spend.merge(total_transactions,␣
↪on='CustomerID')
average_transaction_value['Average_Transaction_Value'] =␣
↪average_transaction_value['Total_Spend'] /␣
↪average_transaction_value['Total_Transactions']

# Merge the new features into the customer_data dataframe
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')

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

[37]:

```
   CustomerID  Days_Since_Last_Purchase  Total_Transactions  \
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

   Total_Products_Purchased  Total_Spend  Average_Transaction_Value
0                         0         0.00                   0.000000
1                      2458      4310.00                 615.714286
2                      2332      1437.24                 359.310000
3                       630      1457.55                1457.550000
4                       196       294.40                 294.400000
```

[38]:

```
# Calculate the number of unique products purchased by each customer
unique_products_purchased = df.groupby('CustomerID')['StockCode'].nunique().
↪reset_index()
```

```
unique_products_purchased.rename(columns={'StockCode':
  'Unique_Products_Purchased'}, inplace=True)

# Merge the new feature into the customer_data dataframe
customer_data = pd.merge(customer_data, unique_products_purchased,
  on='CustomerID')

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

[38]:

```
   CustomerID  Days_Since_Last_Purchase  Total_Transactions  \
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

   Total_Products_Purchased  Total_Spend  Average_Transaction_Value  \
0                         0         0.00                   0.000000
1                      2458      4310.00                 615.714286
2                      2332      1437.24                 359.310000
3                       630      1457.55                1457.550000
4                       196       294.40                 294.400000

   Unique_Products_Purchased
0                          1
1                        103
2                         21
3                         72
4                         16
```

[39]:

```
# Extract day of week and hour from InvoiceDate
df['Day_Of_Week'] = df['InvoiceDate'].dt.dayofweek
df['Hour'] = df['InvoiceDate'].dt.hour

# Calculate the average number of days between consecutive purchases
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)

# Find the favorite shopping day of the week
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']]

# Find the favorite shopping hour of the day
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']]

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, average_days_between_purchases,␣
 ↪on='CustomerID')
customer_data = pd.merge(customer_data, favorite_shopping_day, on='CustomerID')
customer_data = pd.merge(customer_data, favorite_shopping_hour, on='CustomerID')

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

[39]:

```
   CustomerID  Days_Since_Last_Purchase  Total_Transactions  \
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

   Total_Products_Purchased  Total_Spend  Average_Transaction_Value  \
0                         0         0.00                   0.000000
1                      2458      4310.00                 615.714286
2                      2332      1437.24                 359.310000
3                       630      1457.55                1457.550000
4                       196       294.40                 294.400000

   Unique_Products_Purchased  Average_Days_Between_Purchases  Day_Of_Week  \
0                          1                        0.000000            1
1                        103                        2.016575            1
2                         21                       10.884615            3
3                         72                        0.000000            0
4                         16                        0.000000            2

   Hour
0    10
1    14
2    19
3     9
4    16
```

[40]:

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

[40]:
```
Country
United Kingdom    0.890971
Germany           0.022722
France            0.020402
EIRE              0.018440
Spain             0.006162
Name: proportion, dtype: float64
```

[41]:
```python
# 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_data = pd.merge(customer_data, cancellation_frequency,␣
↪on='CustomerID', how='left')

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

# Calculate the Cancellation Rate
customer_data['Cancellation_Rate'] = customer_data['Cancellation_Frequency'] /␣
↪total_transactions['InvoiceNo']

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

[41]:
```
   CustomerID  Days_Since_Last_Purchase  Total_Transactions  \
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

   Total_Products_Purchased  Total_Spend  Average_Transaction_Value  \
0                         0         0.00                   0.000000
1                      2458      4310.00                 615.714286
2                      2332      1437.24                 359.310000
3                       630      1457.55                1457.550000
4                       196       294.40                 294.400000
```

```
   Unique_Products_Purchased  Average_Days_Between_Purchases  Day_Of_Week  \
0                          1                        0.000000            1
1                        103                        2.016575            1
2                         21                       10.884615            3
3                         72                        0.000000            0
4                         16                        0.000000            2

   Hour  Cancellation_Frequency  Cancellation_Rate
0    10                     1.0                0.5
1    14                     0.0                0.0
2    19                     0.0                0.0
3     9                     0.0                0.0
4    16                     0.0                0.0
```

[42]:

```
# Extract month and year from InvoiceDate
df['Year'] = df['InvoiceDate'].dt.year
df['Month'] = df['InvoiceDate'].dt.month

# Calculate monthly spending for each customer
monthly_spending = df.groupby(['CustomerID', 'Year', 'Month'])['Total_Spend'].
 ↪sum().reset_index()

# Calculate Seasonal Buying Patterns: We are using monthly frequency as a proxy␣
 ↪for seasonal buying patterns
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)

# Replace NaN values in Monthly_Spending_Std with 0, implying no variability␣
 ↪for customers with single transaction month
seasonal_buying_patterns['Monthly_Spending_Std'].fillna(0, inplace=True)

# Calculate Trends in Spending
# We are using the slope of the linear trend line fitted to the customer's␣
 ↪spending over time as an indicator of spending trends
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
    # If there is only one data point, no trend can be calculated, hence we␣
 ↪return 0
```

```
    else:
        return 0

# Apply the calculate_trend function to find the spending trend for each␣
 ↪customer
spending_trends = monthly_spending.groupby('CustomerID')['Total_Spend'].
 ↪apply(calculate_trend).reset_index()
spending_trends.rename(columns={'Total_Spend': 'Spending_Trend'}, inplace=True)

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, seasonal_buying_patterns,␣
 ↪on='CustomerID')
customer_data = pd.merge(customer_data, spending_trends, on='CustomerID')

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

[42]:

```
   CustomerID  Days_Since_Last_Purchase  Total_Transactions  \
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

   Total_Products_Purchased  Total_Spend  Average_Transaction_Value  \
0                         0         0.00                   0.000000
1                      2458      4310.00                 615.714286
2                      2332      1437.24                 359.310000
3                       630      1457.55                1457.550000
4                       196       294.40                 294.400000

   Unique_Products_Purchased  Average_Days_Between_Purchases  Day_Of_Week  \
0                          1                        0.000000            1
1                        103                        2.016575            1
2                         21                       10.884615            3
3                         72                        0.000000            0
4                         16                        0.000000            2

   Hour  Cancellation_Frequency  Cancellation_Rate  Monthly_Spending_Mean  \
0    10                     1.0                0.5               0.000000
1    14                     0.0                0.0             615.714286
2    19                     0.0                0.0             359.310000
3     9                     0.0                0.0            1457.550000
4    16                     0.0                0.0             294.400000

   Monthly_Spending_Std  Spending_Trend
0              0.000000        0.000000
```

```
1          341.070789        4.486071
2          203.875689     -100.884000
3            0.000000        0.000000
4            0.000000        0.000000
```

[43]:
```
# Changing the data type of 'CustomerID' to string as it is a unique identifier␣
 ↪and not used in mathematical operations
customer_data['CustomerID'] = customer_data['CustomerID'].astype(str)

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

[44]:
```
customer_data.head(10)
```

[44]:
```
  CustomerID  Days_Since_Last_Purchase  Total_Transactions  \
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
5    12352.0                        36                   8
6    12353.0                       204                   1
7    12354.0                       232                   1
8    12355.0                       214                   1
9    12356.0                        22                   3

   Total_Products_Purchased  Total_Spend  Average_Transaction_Value  \
0                         0          0.0                        0.0
1                      2458       4310.0                 615.714286
2                      2332      1437.24                     359.31
3                       630      1457.55                    1457.55
4                       196        294.4                      294.4
5                       463      1265.41                  158.17625
6                        20         89.0                       89.0
7                       530       1079.4                     1079.4
8                       240        459.4                      459.4
9                      1573      2487.43                 829.143333

   Unique_Products_Purchased  Average_Days_Between_Purchases  Day_Of_Week  \
0                          1                             0.0            1
1                        103                        2.016575            1
2                         21                       10.884615            3
3                         72                             0.0            0
4                         16                             0.0            2
5                         57                         3.13253            1
6                          4                             0.0            3
7                         58                             0.0            3
```

```
8                        13                          0.0            0
9                        52                     5.315789            1

   Hour  Cancellation_Frequency  Cancellation_Rate  Monthly_Spending_Mean  \
0    10                       1                0.5                    0.0
1    14                       0                0.0             615.714286
2    19                       0                0.0                 359.31
3     9                       0                0.0                1457.55
4    16                       0                0.0                  294.4
5    14                       1              0.125               316.3525
6    17                       0                0.0                   89.0
7    13                       0                0.0                 1079.4
8    13                       0                0.0                  459.4
9     9                       0                0.0             829.143333

   Monthly_Spending_Std  Spending_Trend
0                   0.0             0.0
1            341.070789        4.486071
2            203.875689        -100.884
3                   0.0             0.0
4                   0.0             0.0
5            134.700629           9.351
6                   0.0             0.0
7                   0.0             0.0
8                   0.0             0.0
9            991.462585        -944.635
```

[45]:

```
customer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4282 entries, 0 to 4281
Data columns (total 15 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   CustomerID                      4282 non-null   string
 1   Days_Since_Last_Purchase        4282 non-null   Int64
 2   Total_Transactions              4282 non-null   Int64
 3   Total_Products_Purchased        4282 non-null   Int64
 4   Total_Spend                     4282 non-null   Float64
 5   Average_Transaction_Value       4282 non-null   Float64
 6   Unique_Products_Purchased       4282 non-null   Int64
 7   Average_Days_Between_Purchases  4282 non-null   Float64
 8   Day_Of_Week                     4282 non-null   Int32
 9   Hour                            4282 non-null   Int32
 10  Cancellation_Frequency          4282 non-null   Int64
 11  Cancellation_Rate               4282 non-null   Float64
 12  Monthly_Spending_Mean           4282 non-null   Float64
```

```
 13  Monthly_Spending_Std            4282 non-null   Float64
 14  Spending_Trend                  4282 non-null   Float64
dtypes: Float64(7), Int32(2), Int64(5), string(1)
memory usage: 527.0 KB
```