P8 Building Customer Segmentation Models using Python

Full Name:Sneha Kisan Kadam

Internship Registraion ID: STU63734d18176301668500760

Problem:

In this project, we delve deep into the thriving sector of **online retail** by analyzing a **transactional dataset** from a UK-based retailer, available at the UCI Machine Learning Repository. This dataset documents all transactions between 2010 and 2011. Our primary objective is to amplify the efficiency of marketing strategies and boost sales through **customer segmentation**. We aim to transform the transactional data into a customer-centric dataset by creating new features that will facilitate the segmentation of customers into distinct groups using the **K-means clustering** algorithm. This segmentation will allow us to understand the

distinct **profiles** and preferences of different customer groups. Building upon this, we intend to develop a **recommendation system** that will suggest top-selling products to customers within each segment who haven't purchased those items yet, ultimately enhancing marketing efficacy and fostering increased sales.

Objectives:

- **Data Cleaning & Transformation**: Clean the dataset by handling missing values, duplicates, and outliers, preparing it for effective clustering.
- **Feature Engineering**: Develop new features based on the transactional data to create a customer-centric dataset, setting the foundation for customer segmentation.
- **Data Preprocessing**: Undertake feature scaling and dimensionality reduction to streamline the data, enhancing the efficiency of the clustering process.
- **Customer Segmentation using K-Means Clustering**: Segment customers into distinct groups using K-means, facilitating targeted marketing and personalized strategies.
- **Cluster Analysis & Evaluation**: Analyze and profile each cluster to develop targeted marketing strategies and assess the quality of the clusters formed.
- **Recommendation System**: Implement a system to recommend best-selling products to customers within the same cluster who haven't purchased those products, aiming to boost sales and marketing effectiveness.

Table of Contents:

Let's get started:

Step 1 | Setup and Initialization

Tabel of Contents

Step 1.1 | Importing Necessary Libraries

Tabel of Contents

First of all, we 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:

In [4]:
```
# 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
from sklearn.cluster import KMeans
from tabulate import tabulate
from collections import Counter

%matplotlib inline
```

In [5]:
```
# Initialize Plotly for use in the notebook
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

In [6]:
```
# 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, we will load the dataset into a pandas DataFrame which will facilitate easy manipulation and analysis:

In [7]:
```
df = pd.read_csv('customer_data_internship.csv', encoding="ISO-8859-1")
```

Dataset Description:

Variable	Description
InvoiceNo	Code representing each unique transaction. If this code starts with letter 'c',

Variable	Description
	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.

Step 2 | Initial Data Analysis

⬆️ Tabel of Contents

Afterward, we are going to gain a thorough understanding of the dataset before proceeding to the data cleaning and transformation stages.

Step 2.1 | Dataset Overview

⬆️ Tabel of Contents

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

In [8]:
```
df.head(10)
```

Out[8]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850.0
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850.0
7	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850.0
8	536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	17850.0
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	13047.0

In [9]:
```
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
```

Inferences:

The dataset consists of 541,909 entries and 8 columns. Here is a brief overview of each column:

- **`InvoiceNo`**: This is an object data type column that contains the invoice number for each transaction. Each invoice number can represent multiple items purchased in a single transaction.
- **`StockCode`**: An object data type column representing the product code for each item.
- **`Description`**: This column, also an object data type, contains descriptions of the products. It has some missing values, with 540,455 non-null entries out of 541,909.
- **`Quantity`**: This is an integer column indicating the quantity of products purchased in each transaction.
- **`InvoiceDate`**: A datetime column that records the date and time of each transaction.
- **`UnitPrice`**: A float column representing the unit price of each product.

- `CustomerID` : A float column that contains the customer ID for each transaction. This column has a significant number of missing values, with only 406,829 non-null entries out of 541,909.
- `Country` : An object column recording the country where each transaction took place.

From a preliminary overview, it seems that there are missing values in the `Description` and `CustomerID` columns which need to be addressed. The `InvoiceDate` column is already in datetime format, which will facilitate further time series analysis. We also observe that a single customer can have multiple transactions as inferred from the repeated `CustomerID` in the initial rows.

The next steps would include deeper data cleaning and preprocessing to handle missing values, potentially erroneous data, and to create new features that can help in achieving the project goals.

Step 2.2 | Summary Statistics

Tabel of Contents

Now, we are going to generate summary statistics to gain initial insights into the data distribution:

In [10]:
```
# Summary statistics for numerical variables
df.describe().T
```

Out[10]:

	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	1

In [11]:
```
# Summary statistics for categorical variables
df.describe(include='object').T
```

Out[11]:

	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

Inferences:

- `Quantity`:
  - The average quantity of products in a transaction is approximately 9.55.
  - The quantity has a wide range, with a minimum value of -80995 and a maximum value of 80995. The negative values indicate returned or cancelled orders, which need to be handled appropriately.
  - The standard deviation is quite large, indicating a significant spread in the data. The presence of outliers is indicated by a large difference between the maximum and the 75th percentile values.

- `UnitPrice`:
  - The average unit price of the products is approximately 4.61.
  - The unit price also shows a wide range, from -11062.06 to 38970, which suggests the presence

of errors or noise in the data, as negative prices don't make sense.
    - Similar to the Quantity column, the presence of outliers is indicated by a large difference between the maximum and the 75th percentile values.

- `CustomerID`:
    - There are 406829 non-null entries, indicating missing values in the dataset which need to be addressed.
    - The Customer IDs range from 12346 to 18287, helping in identifying unique customers.

- `InvoiceNo`:
    - There are 25900 unique invoice numbers, indicating 25900 separate transactions.
    - The most frequent invoice number is 573585, appearing 1114 times, possibly representing a large transaction or an order with multiple items.

- `StockCode`:
    - There are 4070 unique stock codes representing different products.
    - The most frequent stock code is 85123A, appearing 2313 times in the dataset.

- `Description`:
    - There are 4223 unique product descriptions.
    - The most frequent product description is "WHITE HANGING HEART T-LIGHT HOLDER", appearing 2369 times.
    - There are some missing values in this column which need to be treated.

- `Country`:
    - The transactions come from 38 different countries, with a dominant majority of the transactions

(approximately 91.4%) originating from the United Kingdom.

Step 3 | Data Cleaning & Transformation

Tabel of Contents

This step encompasses a comprehensive cleaning and transformation process to refine the dataset. It includes addressing missing values, eliminating duplicate entries, correcting anomalies in product codes and descriptions, and other necessary adjustments to prepare the data for in-depth analysis and modeling.

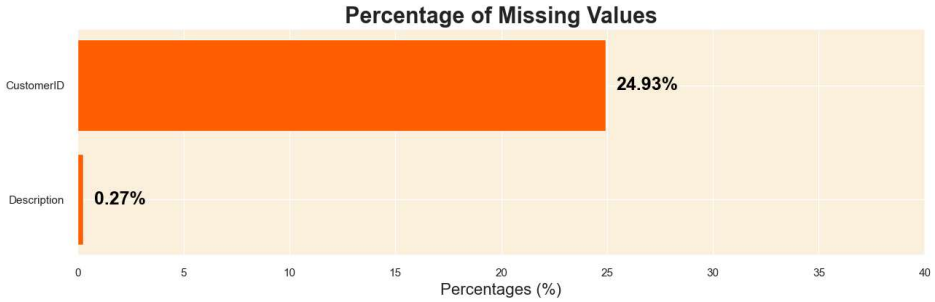
Step 3.1 | Handling Missing Values

Tabel of Contents

Initially, we will determine the percentage of missing values present in each column, followed by selecting the most effective strategy to address them:

In [12]:
```
# Calculating the percentage of missing values for each column
missing_data = df.isnull().sum()
missing_percentage = (missing_data[missing_data > 0] / df.shape[0

# 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='#ff6

# Annotate the values and indexes
for i, (value, name) in enumerate(zip(missing_percentage, missing
    ax.text(value+0.5, i, f"{value:.2f}%", ha='left', va='center'

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

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



Handling Missing Values Strategy:

- **`CustomerID` (24.93% missing values)**
  - The `CustomerID` column contains nearly a quarter of missing data. This column is essential for clustering customers and creating a recommendation system. Imputing such a large percentage of missing values might introduce significant bias or noise into the analysis.
  - Moreover, since the clustering is based on customer behavior and preferences, it's crucial to have accurate data on customer identifiers. Therefore, removing the rows with missing `CustomerID`s seems to be the most reasonable approach to maintain the integrity of the clusters and the analysis.
- **`Description` (0.27% missing values)**

- The `Description` column has a minor percentage of missing values. However, it has been noticed that there are inconsistencies in the data where the same `StockCode` does not always have the same `Description`. This indicates data quality issues and potential errors in the product descriptions.

- Given these inconsistencies, imputing the missing descriptions based on `StockCode` might not be reliable. Moreover, since the missing percentage is quite low, it would be prudent to remove the rows with missing `Description`s to avoid propagating errors and inconsistencies into the subsequent analyses.

By removing rows with missing values in the `CustomerID` and `Description` columns, we aim to construct a cleaner and more reliable dataset, which is essential for achieving accurate clustering and creating an effective recommendation system.

In [13]:
```
# Extracting rows with missing values in 'CustomerID' or 'Descr
df[df['CustomerID'].isnull() | df['Description'].isnull()].head
```

Out[13]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice
622	536414	22139	NaN	56	12/1/2010 11:52	0.00
1443	536544	21773	DECORATIVE ROSE BATHROOM BOTTLE	1	12/1/2010 14:32	2.51
1444	536544	21774	DECORATIVE CATS BATHROOM BOTTLE	2	12/1/2010 14:32	2.51
1445	536544	21786	POLKADOT RAIN HAT	4	12/1/2010 14:32	0.85
1446	536544	21787	RAIN PONCHO RETROSPOT	2	12/1/2010 14:32	1.66

In [14]:
```
# Removing rows with missing values in 'CustomerID' and 'Descri
df = df.dropna(subset=['CustomerID', 'Description'])
```

In [15]:
```
# Verifying the removal of missing values
df.isnull().sum().sum()
```

Out[15]:
```
0
```

Step 3.2 | Handling Duplicates

Tabel of Contents

Next, we are going to recognize duplicate rows in the dataset:

In [16]:
```
# Finding duplicate rows (keeping all instances)
duplicate_rows = df[df.duplicated(keep=False)]

# Sorting the data by certain columns to see the duplicate row
duplicate_rows_sorted = duplicate_rows.sort_values(by=['Invoic

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

Out[16]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice
494	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	12/1/2010 11:45	1.25
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	12/1/2010 11:45	1.25
485	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	12/1/2010 11:45	4.95
539	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	12/1/2010 11:45	4.95
489	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	12/1/2010 11:45	2.10
527	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	12/1/2010 11:45	2.10
521	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	12/1/2010 11:45	2.95
537	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	12/1/2010 11:45	2.95
578	536412	21448	12 DAISY PEGS IN WOOD BOX	1	12/1/2010 11:49	1.65
598	536412	21448	12 DAISY PEGS IN WOOD BOX	1	12/1/2010 11:49	1.65

Handling Duplicates Strategy:

In the context of this project, the presence of completely identical rows, including identical

transaction times, suggests that these might be data recording errors rather than genuine repeated transactions. Keeping these duplicate rows can introduce noise and potential inaccuracies in the clustering and recommendation system.

Therefore, I am going to remove these completely identical duplicate rows from the dataset. Removing these rows will help in achieving a cleaner dataset, which in turn would aid in building more accurate customer clusters based on their unique purchasing behaviors. Moreover, it would help in creating a more precise recommendation system by correctly identifying the products with the most purchases.

In [17]:
```
# Displaying the number of duplicate rows
print(f"The dataset contains {df.duplicated().sum()} duplicat

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

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

In [18]:
```
# Getting the number of rows in the dataframe
df.shape[0]
```

Out[18]:
```
401604
```

Step 3.3 | Treating Cancelled Transactions

Tabel of Contents

To refine our understanding of customer behavior and preferences, we need to take into account the transactions that were cancelled. Initially, we will identify these transactions by filtering the rows where the `InvoiceNo` starts with "C". Subsequently, we will analyze these rows to

understand their common characteristics or patterns:

In [19]:
```
# Filter out the rows with InvoiceNo starting with "C" and
df['Transaction_Status'] = np.where(df['InvoiceNo'].astype(

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

Out[19]:

	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

Inferences from the Cancelled Transactions Data:

- All quantities in the cancelled transactions are negative, indicating that these are indeed orders that were cancelled.
- The `UnitPrice` column has a considerable spread, showing that a variety of products, from low to high value, were part of the cancelled transactions.

Strategy for Handling Cancelled Transactions:

Considering the project's objective to cluster customers based on their purchasing behavior and preferences and to eventually create a recommendation system, it's imperative to understand the cancellation patterns of customers. Therefore, the strategy is to retain these cancelled transactions in the dataset, marking them distinctly to facilitate further analysis. This approach will:

- Enhance the clustering process by incorporating patterns and trends observed in cancellation data, which might represent certain customer behaviors or preferences.
- Allow the recommendation system to possibly prevent suggesting products that have a high likelihood of being cancelled, thereby improving the quality of recommendations.

In [20]:
```
# Finding the percentage of cancelled transactions
cancelled_percentage = (cancelled_transactions.shape[0]

# Printing the percentage of cancelled transactions
print(f"The percentage of cancelled transactions in the
```

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

Step 3.4 | Correcting StockCode Anomalies

Tabel of Contents

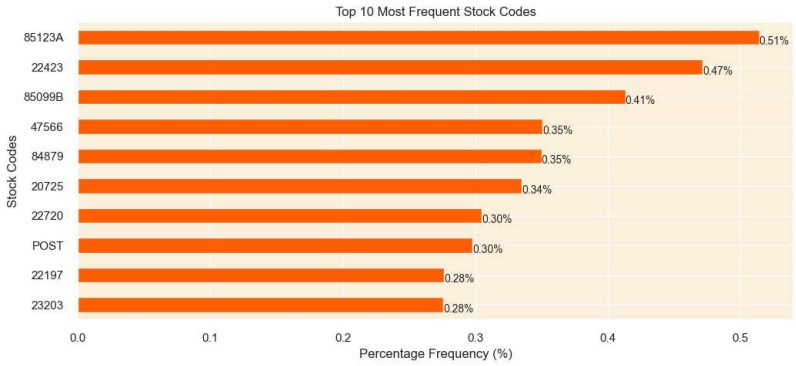
First of all, lets find the number of unique stock codes and to plot the top 10 most frequent

stock codes along with their percentage frequency:

In [21]:
```
# 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
```

```
The number of unique stock codes in the dataset is: 368
4
```

In [22]:
```
# Finding the top 10 most frequent stock codes
top_10_stock_codes = df['StockCode'].value_counts(norma

# 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}%', fontsi

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



Inferences on Stock Codes:

- **Product Variety**: The dataset contains 3684 unique stock codes, indicating a substantial variety of products available in the online retail store. This diversity can potentially lead to the identification of distinct customer clusters, with

preferences for different types of products.

- **Popular Items**: A closer look at the top 10 most frequent stock codes can offer insights into the popular products or categories that are frequently purchased by customers.

- **Stock Code Anomalies**: We observe that while most stock codes are composed of 5 or 6 characters, there are some anomalies like the code '**POST**'. These anomalies might represent services or non-product transactions (perhaps postage fees) rather than actual products. To maintain the focus of the project, which is clustering based on product purchases and creating a recommendation system, these anomalies should be further investigated and possibly treated appropriately to ensure data integrity.

To delve deeper into identifying these anomalies, let's explore the frequency of the number of numeric characters in the stock codes, which can provide insights into the nature of these unusual entries:

In [23]:
```
# Finding the number of numeric characters in each u
unique_stock_codes = df['StockCode'].unique()
numeric_char_counts_in_unique_codes = pd.Series(uniq

# Printing the value counts for unique stock codes
print("Value counts of numeric character frequencies
print("-"*70)
print(numeric_char_counts_in_unique_codes)
```

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

Inference:

The output indicates the following:

- A majority of the unique stock codes (3676 out of 3684) contain exactly 5 numeric characters, which seems to be the standard format for representing product codes in this dataset.
- There are a few anomalies: 7 stock codes contain no numeric characters and 1 stock code contains only 1 numeric character. These are clearly deviating from the standard format and need further investigation to understand their nature and whether they represent valid product transactions.

Now, let's identify the stock codes that contain 0 or 1 numeric characters to further understand these anomalies:

In [24]:
```
# Finding and printing the stock codes with 0 and 1
anomalous_stock_codes = [code for code in unique_st

# 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
```

Let's calculate the percentage of records with these anomalous stock codes:

In [25]:
```
# Calculating the percentage of records with these
percentage_anomalous = (df['StockCode'].isin(anom

# Printing the percentage
print(f"The percentage of records with anomalous
```

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

Inference:

Based on the analysis, we find that a very small proportion of the records, **0.48%**, have anomalous stock codes, which deviate from the typical format observed in the majority of the data. Also, these anomalous codes are just a fraction among all unique stock codes (**only 8 out of 3684**).

These codes seem to represent non-product transactions like "**BANK CHARGES**", "**POST**" (possibly postage fees), etc. Since they do not represent actual products and are a very small proportion of the dataset, including them in the analysis might introduce

noise and distort the clustering and recommendation system.

Strategy:

Given the context of the project, where the aim is to cluster customers based on their product purchasing behaviors and develop a product recommendation system, it would be prudent to exclude these records with anomalous stock codes from the dataset. This way, the focus remains strictly on genuine product transactions, which would lead to a more accurate and meaningful analysis.

Thus, the strategy would be to filter out and remove rows with these anomalous stock codes from the dataset before proceeding with further analysis and model development:

In [26]:
```
# Removing rows with anomalous stock codes fr
df = df[~df['StockCode'].isin(anomalous_stock
```

In [27]:
```
# Getting the number of rows in the dataframe
df.shape[0]
```

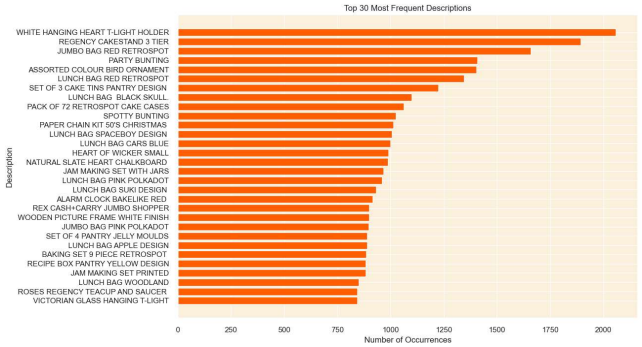
Out[27]:
```
399689
```

Step 3.5 | Cleaning Description Column

Tabel of Contents

First, I will calculate the occurrence count of each unique description in the dataset. Then, I will plot the top 30 descriptions. This visualization will give a clear view of the highest occurring descriptions in the dataset:

In [28]:
```
# Calculate the occurrence of each unique de
description_counts = df['Description'].value

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

# Plotting
plt.figure(figsize=(12,8))
plt.barh(top_30_descriptions.index[::-1], to

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

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



Inferences on Descriptions:

- The most frequent descriptions are generally household items, particularly those associated with kitchenware, lunch bags, and decorative items.

- Interestingly, all the descriptions are in uppercase, which might be a standardized format for entering product descriptions in the database. However, considering the inconsistencies and anomalies encountered in the dataset so far, it would be prudent to check if there are descriptions entered in lowercase or a mix of case styles.

In [29]:
```
# Find unique descriptions containing lowe
lowercase_descriptions = df['Description']
lowercase_descriptions = [desc for desc in

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

```
The unique descriptions containing lowerca
se 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
```

Inference:

- Upon reviewing the descriptions that contain lowercase characters, it is evident that some entries are not product descriptions, such as "**Next Day Carriage**" and "**High Resolution Image**". These entries seem to be unrelated to the actual products and might represent other types of information or service details.

Strategy:

- **Step 1**: Remove the rows where the descriptions contain service-related information like "**Next Day Carriage**" and "**High Resolution Image**", as these do not represent actual products and would not contribute to the clustering and recommendation system we aim to build.
- **Step 2**: For the remaining descriptions with mixed case, standardize the text to uppercase to maintain uniformity across the dataset. This will also assist in reducing the chances of having duplicate entries with different case styles.

By implementing the above strategy, we can enhance the quality of our dataset, making it more suitable for the analysis and modeling phases of our project.

In [30]:
```
service_related_descriptions = ["Next Da

# Calculate the percentage of records wi
service_related_percentage = df[df['Desc

# Print the percentage of records with s
print(f"The percentage of records with s

# Remove rows with service-related infor
df = df[~df['Description'].isin(service_
```

```
# Standardize the text to uppercase to m
df['Description'] = df['Description'].st
```

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

In [31]:
```
# Getting the number of rows in the datc
df.shape[0]
```

Out[31]:
```
399606
```

Step 3.6 | Treating Zero Unit Prices

Tabel of Contents

In this step, first I am going to take a look at the statistical description of the `UnitPrice` column:

In [32]:
```
df['UnitPrice'].describe()
```

Out[32]:
```
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
```

Inference:

The minimum unit price value is zero. This suggests that there are some transactions where the unit price is zero, potentially indicating a free

item or a data entry error. To understand their nature, it is essential to investigate these zero unit price transactions further. A detailed analysis of the product descriptions associated with zero unit prices will be conducted to determine if they adhere to a specific pattern:

In [33]:
```
df[df['UnitPrice']==0].describe()[['Q
```

Out[33]:

	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

Inferences on UnitPrice:

- The transactions with a unit price of zero are relatively few in number (33 transactions).
- These transactions have a large variability in the quantity of items involved, ranging from

1 to 12540, with a substantial standard deviation.

- Including these transactions in the clustering analysis might introduce noise and could potentially distort the customer behavior patterns identified by the clustering algorithm.

Strategy:

Given the small number of these transactions and their potential to introduce noise in the data analysis, the strategy should be to remove these transactions from the dataset. This would help in maintaining a cleaner and more consistent dataset, which is essential for building an accurate and reliable clustering model and recommendation system.

In [34]:
```
# Removing records with a unit pri
df = df[df['UnitPrice'] > 0]
```

Step 3.7 | Outlier Treatment

Tabel of Contents

In K-means clustering, the algorithm is sensitive to both the scale of data and the presence of outliers, as they can significantly influence the position of centroids, potentially leading to incorrect cluster assignments. However, considering the context of this project where the final goal is to understand customer behavior and preferences through K-means clustering, it would be more prudent to address the issue of outliers **after the feature engineering phase** where we create a customer-centric dataset. At this stage, the data is transactional, and removing outliers might eliminate valuable information that could play a crucial role in segmenting customers later on. Therefore, we

will postpone the outlier treatment and proceed to the next stage for now.

In [35]:
```
# Resetting the index of the cle
df.reset_index(drop=True, inplace
```

In [36]:
```
# Getting the number of rows in t
df.shape[0]
```

Out[36]: 399573

Step 4 | Feature Engineerin

Tabel of Contents

In order to create a comprehensive customer-centric dataset for clustering and recommendation, the following features can be engineered from the available data:

Step 4.1 | RFM Features

Tabel of Contents

RFM is a method used for analyzing customer value and segmenting the customer base. It is an acronym that stands for:

- **Recency (R):** 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.
- **Frequency (F):** 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.

- **Monetary (M):** 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.

Together, these metrics help in understanding a customer's buying behavior and preferences, which is pivotal in personalizing marketing strategies and creating a recommendation system.

Step 4.1.1 | Recency (R)

Tabel of Contents

In this step, we focus on understanding

how recently a customer has made a purchase. This is a crucial aspect of customer segmentation as it helps in identifying the engagement level of customers. Here, I am going to define the following feature:

- **Days Since Last Purchas**: This feature represents the number of days that have passed since the customer's last purchase. A lower value indicates that the customer has purchased recently, implying a higher engagement level with the business, whereas a higher value may indicate a lapse or decreased engagement. By understanding

the recency of purchases, businesses can tailor their marketing strategies to re-engage customers who have not made purchases in a while, potentially increasing customer retention and fostering loyalty.

In [37]:
```
# Convert InvoiceDate to date
df['InvoiceDate'] = pd.to_da

# Convert InvoiceDate to date
df['InvoiceDay'] = df['Invoi

# Find the most recent purch
customer_data = df.groupby('

# Find the most recent date
most_recent_date = df['Invoi

# Convert InvoiceDay to date
customer_data['InvoiceDay'] =
most_recent_date = pd.to_date

# Calculate the number of da
customer_data['Days_Since_La

# Remove the InvoiceDay colu
customer_data.drop(columns=[
```

Now, **customer_data** dataframe contains the

`Days_Since_Last_Purc`
feature:

In [38]:
```
customer_data.head()
```

Out[38]:

	CustomerID	Days_Since_Last_
0	12346.0	
1	12347.0	
2	12348.0	
3	12349.0	
4	12350.0	

Note:

- We've named the customer-centric dataframe as **customer_data**, which will eventually contain all the customer-based features we plan to create.

Step 4.1.2 | Frequency (F)

Tabel of Contents

In this step, I am going to create two features that quantify the frequency of a customer's engagement with the retailer:

- **Total Transactions**: This feature represents the total number of transactions made by a customer. It helps in understanding the engagement level of a customer with the retailer.
- **Total Products Purchased**: This feature indicates the total number of products (sum of quantities)

purchased by a customer across all transactions. It gives an insight into the customer's buying behavior in terms of the volume of products purchased.

These features will be crucial in segmenting customers based on their buying frequency, which is a key aspect in determining customer segments for targeted marketing and personalized recommendations.

In [39]:
```
# Calculate the total nu
total_transactions = df.
total_transactions.renam

# Calculate the total nu
total_products_purchased
total_products_purchased

# Merge the new features
```

```
customer_data = pd.merge
customer_data = pd.merge

# Display the first few
customer_data.head()
```

Out[39]:

	CustomerID	Days_Since_La
0	12346.0	
1	12347.0	
2	12348.0	
3	12349.0	
4	12350.0	

Step 4.1.3 | Monetary (M)

Tabel of Contents

In this step, I am going to create two features that represent the monetary aspect of customer's transactions:

- **Total Spend**: This feature represents the total amount of money spent by each customer. It

is calculated as the sum of the product of `UnitPrice` and `Quantity` for all transactions made by a customer. This feature is crucial as it helps in identifying the total revenue generated by each customer, which is a direct indicator of a customer's value to the business.

- **Average Transaction Value**: This feature is calculated as the **Total Spend** divided by

the **Total Transactions** for each customer. It indicates the average value of a transaction carried out by a customer. This metric is useful in understanding the spending behavior of customers per transaction, which can assist in tailoring marketing strategies and offers to different customer segments based on their average spending patterns.

In [40]:
```
# Calculate the total s
df['Total_Spend'] = df[
total_spend = df.groupb

# Calculate the average
average_transaction_val
average_transaction_val

# Merge the new feature
customer_data = pd.merg
customer_data = pd.merg

# Display the first few
customer_data.head()
```

Out[40]:

	CustomerID	Days_Since_
0	12346.0	
1	12347.0	
2	12348.0	
3	12349.0	
4	12350.0	

Step 4.2 | Product Diversity

⬆️ Tabel of Contents

In this step, we are going to understand the diversity in the product purchase behavior of customers. Understanding product diversity can

help in crafting personalized marketing strategies and product recommendations. Here, I am going to define the following feature:

- **Unique Products Purchased**: This feature represents the number of distinct products bought by a customer. A higher value indicates that the customer has a diverse taste or preference, buying a wide

range of products, while a lower value might indicate a focused or specific preference. Understanding the diversity in product purchases can help in segmenting customers based on their buying diversity, which can be a critical input in personalizing product recommendatic

In [41]:
```
# Calculate the numbe
customer_products = d
customer_products['Un

# Merge the new featu
customer_data = pd.me
```

```
# Display the first f
customer_data.head()
```

Out[41]:

	CustomerID	Days_Sinc
0	12346.0	
1	12347.0	
2	12348.0	
3	12349.0	
4	12350.0	

Step 4.3 | Behavioral Features

Tabel of Contents

In this step, we aim to understand and capture the shopping patterns and behaviors of customers. These features will give us insights into the customers' preferences regarding when they like to shop,

which can be crucial information for personalizing their shopping experience. Here are the features I am planning to introduce:

- **Average Days Between Purchases**: This feature represents the average number of days a customer waits before making another purchase. Understanding this can help in predicting when the customer

is likely to make their next purchase, which can be a crucial metric for targeted marketing and personalized promotions.

- **Favorite Shopping Day**: This denotes the day of the week when the customer shops the most. This information can help in identifying the preferred shopping days of different

customer segments, which can be used to optimize marketing strategies and promotions for different days of the week.

- **Favorite Shopping Hour**: This refers to the hour of the day when the customer shops the most. Identifying the favorite shopping hour can aid in optimizing the timing of

marketing campaigns and promotions to align with the times when different customer segments are most active.

By including these behavioral features in our dataset, we can create a more rounded view of our customers, which will potentially enhance the effectiveness of the clustering algorithm, leading to more meaningful

customer segments.

In [42]:
```
# Extract day of wee
df['Day_Of_Week'] =
df['Hour'] = df['Inv

# Calculate the aver
days_between_purchas
average_days_between
average_days_between

# Find the favorite
favorite_shopping_da
favorite_shopping_da

# Find the favorite
favorite_shopping_ho
favorite_shopping_ho

# Merge the new feat
customer_data = pd.m
customer_data = pd.m
customer_data = pd.m

# Display the first
customer_data.head()
```

Out[42]:

	CustomerID	Days_Si
0	12346.0	
1	12347.0	
2	12348.0	
3	12349.0	
4	12350.0	

Step 4.4

Geographi Features

Tabel of Contents

In this step, we will introduce a geographic feature that reflects the geographical location of customers. Understanding the geographic distribution of customers is pivotal for several reasons:

- **Country**: This feature identifies the country where each customer is located. Including the country

data can help us understand region-specific buying patterns and preferences. Different regions might have varying preferences and purchasing behaviors which can be critical in personalizing marketing strategies and inventory planning. Furthermore, it can be instrumental in logistics and

supply chain optimization, particularly for an online retailer where shipping and delivery play a significant role.

In [43]:
```
df['Country'].value
```

Out[43]:
```
Country
United Kingdom
0.890971
Germany
0.022722
France
0.020402
EIRE
0.018440
Spain
0.006162
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.

Methodol

- First, I will group the data by `Customer` and `Country` and calculate the number of transactio per country for each customer.

- Next, I will identify the main country for each customer (the country from which they have the maximum transactio

- Then, I will create a binary column indicating whether the customer is from the UK or not.

- Finally, I will merge this informatic with the `customer` dataframe to include the new feature in our analysis.

In [44]:
```
# Group by Custo
customer_country

# Get the countr
customer_main_cc

# Create a binar
customer_main_cc

# Merge this dat
customer_data =

# Display the fi
customer_data.he
```

Out[44]:

	CustomerID	Da
0	12346.0	
1	12347.0	
2	12348.0	
3	12349.0	
4	12350.0	

In [45]:
```
# Display featur
customer_data['I
```

Out[45]:
```
Is_UK
1    3866
0     416
Name: count, dt
ype: int64
```

Step 4.5 | Cancella Insights

Tabel of Contents

In this step, We are going to delve deeper into the cancellation patterns of customers to gain insights that can enhance our customer segmentatior model. The

features I am planning to introduce are:

- **Cancella Frequen** This metric represen the total number of transacti a custome has canceled Understa the frequenc of cancellat can help us identify custome who are more likely to

cancel
transacti
This
could
be
an
indicator
of
dissatisfa
or
other
issues,
and
understa
this
can
help
us
tailor
strategie
to
reduce
cancellat
and
enhance
custome
satisfacti

- **Cancella Rate**: This represen the proportic of transacti

that a customer has canceled out of all their transacti This metric gives a normaliz view of cancellat behavior A high cancellat rate might be indicative of an unsatisfie customer segment By identifyir these segment we

can develop targeted strategie to improve their shopping experien and potential reduce the cancellat rate.

By incorporating these cancellation insights into our dataset, we can build a more comprehensi view of customer behavior, which could potentially aid in creating more

effective and nuanced customer segmentation

In [46]:
```
# Calculate the
total_transact

# Calculate the
cancelled_trans
cancellation_fr
cancellation_fr

# Merge the Car
customer_data =

# Replace NaN v
customer_data[

# Calculate the
customer_data[

# Display the f
customer_data.h
```

Out[46]:

	CustomerID	D
0	12346.0	
1	12347.0	
2	12348.0	
3	12349.0	
4	12350.0	

Step 4.6 | Seasona & Trends

Tabel of Contents

In this step, I will delve into the seasonality and trends in customers' purchasing behaviors, which can offer invaluable insights for tailoring marketing strategies and enhancing customer satisfaction. Here are the features I am looking

to introduce:

- **Monthl** This is the average amount a custom spends monthly It helps us gauge the general spendir habit of each custom A higher mean indicate a custom who spends more, potenti showing interest

in premiu product whereas a lower mean might indicate a more budget- conscio custom

- **Month** This feature indicate the variabili in a custom monthly spendir A higher value signals that the custom spendir fluctuat significa

month-
to-
month,
perhaps
indicati
sporadi
large
purchas
In
contras
a
lower
value
suggest
more
stable,
consiste
spendir
habits.
Underst
this
variabili
can
help
in
crafting
persona
promot
or
discour
during
periods
they
are
expecte
to

spend more.

- **Spendi** This reflects the trend in a custom spendir over time, calculat as the slope of the linear trend line fitted to their spendir data. A positive value indicate an increasi trend in

spendir
possibly
pointing
to
growing
loyalty
or
satisfac
Convers
a
negativ
trend
might
signal
decreas
interest
or
satisfac
highligh
a
need
for
re-
engage
strategi
A
near-
zero
value
signifies
stable
spendir
habits.
Recogn
these
trends

can help in develop strategi to either maintai or alter custom spendir pattern: enhanci the effectiv of marketi campai

By incorporatin these detailed insights into our customer segmentatic model, we can create more precise and actionable

customer groups, facilitating the developmer of highly targeted marketing strategies and promotions.

In [47]:
```
# Extract mon
df['Year'] =
df['Month'] =

# Calculate m
monthly_spend

# Calculate S
seasonal_buyi
seasonal_buyi

# Replace NaN
seasonal_buyi

# Calculate T
# We are usin
def calculate
    # If ther
    if len(sp
        x = n
        slope
        retur
    # If ther
    else:
        retur

# Apply the c
spending_tren
spending_tren

# Merge the n
customer_data
customer_data

# Display the
customer_data
```

Out[47]:

	CustomerID
0	12346.0
1	12347.0
2	12348.0
3	12349.0
4	12350.0

We've done a great job so far! We have created a dataset that focuses on our customers, using a variety of new features that give us a deeper

understanc
of
their
buying
patterns
and
preference

In [48]:
```
# Changing t
customer_dat

# Convert da
customer_dat
```

In [49]:
```
customer_dat
```

Out[49]:

	CustomerID
0	12346.0
1	12347.0
2	12348.0
3	12349.0
4	12350.0
5	12352.0
6	12353.0
7	12354.0
8	12355.0
9	12356.0

In [50]:
```
customer_dat
```

```
<class 'pan
das.core.fr
ame.DataFra
me'>
RangeIndex:
4282 entrie
s, 0 to 428
1
Data column
s (total 16
columns):
 #   Column
Non-Null Co
unt  Dtype
---  ------
-----------
---  -----
 0   Custom
erID
4282 non-nu
ll   string
 1   Days_S
ince_Last_P
urchase
4282 non-nu
ll   Int64
 2   Total_
Transaction
s
4282 non-nu
ll   Int64
 3   Total_
Products_Pu
rchased
4282 non-nu
ll   Int64
 4   Total_
Spend
4282 non-nu
ll   Float6
4
 5   Averag
e_Transacti
on_Value
4282 non-nu
ll   Float6
4
 6   Unique
_Products_P
urchased
4282 non-nu
ll   Int64
 7   Averag
e_Days_Betw
een_Purchas
es  4282 no
n-null   Fl
oat64
 8   Day_Of
_Week
```

```
4282 non-nu
ll   Int32
 9   Hour
4282 non-nu
ll   Int32
 10  Is_UK
4282 non-nu
ll   Int64
 11  Cancel
lation_Freq
uency
4282 non-nu
ll   Int64
 12  Cancel
lation_Rate
4282 non-nu
ll   Float6
4
 13  Monthl
y_Spending_
Mean
4282 non-nu
ll   Float6
4
 14  Monthl
y_Spending_
Std
4282 non-nu
ll   Float6
4
 15  Spendi
ng_Trend
4282 non-nu
ll   Float6
4
dtypes: Flo
at64(7), In
t32(2), Int
64(6), stri
ng(1)
memory usag
e: 564.7 KB
```

Let's review the descriptic of the columns in our newly

created `custome` dataset:

Cust Data Desc

Variable
Customer
Days_Sinc
Total_Trar
Total_Pro
Total_Spe
Average_

Variable
Unique_P
Average_
Day_Of_W
Hour
Is_UK
Cancellati
Cancellati

Variable
Monthly_
Monthly_
Spending

We've done a great job so far!

We have created a dataset that focuses on our customers using a variety of new features that give us a deeper understanding of their buying patterns and preferences.

Now that our dataset is ready, we

can
move
on
to
the
next
steps
of
our
project
This
include
looking
at
our
data
more
closely
to
find
any
patterr
or
trends,
making
sure
our
data
is
in
the
best
shape
by
checki
for

and
handli
any
outlier
and
prepar
our
data
for
the
cluster
proces
All
of
these
steps
will
help
us
build
a
strong
founda
for
creatin
our
custom
segme
and,
eventu
a
person
recomi
system

Let's dive in!

S
5
I
O
D
ar
Tr

Tabel of Contents

In this sectic I will ident and handl outlie in our datas Outlie are

data point that are signif differ from the majo of other point in the datas These point can poter skew the result of our analy espec in k-mean cluste where they can signif influe the

positi
of
the
cluste
centr
There
it
is
essen
to
ident
and
treat
these
outlie
appro
to
achie
more
accur
and
mean
cluste
result

Given
the
multi
dime
natur
of
the
data,
it
would
be

prude
to
use
algor
that
can
detec
outlie
in
multi
dime
space
I
am
going
to
use
the
Isola
Fores
algor
for
this
task.
This
algor
works
well
for
multi
dime
data
and
is
comp
efficie

It isolat obse by rand select a featu and then rand select a split value betw the maxi and minin value of the select featu

Let's proce with this appr

In [51]:
```
# Init
model

# Fitt
custom
```

```
# Crea
custom

# Disp
custom
```

Out[51]:

Cust
0
1
2
3
4

Afte
app
the
Isol
Fore
algc
we
have
ider
the
outl
and
mar
ther
in
a
new
colu
nam
`Is_`
We
have
also
calc

the outl scor whic repr the ano scor of each recc

Nov let's visu the distı of thes scor and the num of inlie and outl dete by the moc

In [52]:
```
# Cal
outli

# Plo
plt.f
```

```
outli

# Add
for i
    p

plt.t
plt.x
plt.x
plt.y
plt.g
plt.s
```

In

Fro
the
ab
plo
we
car
ob
tha
ab
5%
of
the
cus
ha
be
ide
as
ou
in
ou
da
Th
pe

see
to
be
a
rea
pro
no
too
hig
to
los
a
sig
am
of
da
an
no
too
lov
to
ret
po
no
da
po
It
sug
tha
ou
iso
for
alg
ha
wo
we

in
ide
a
mo
pe
of
ou
wh
wil
be
cri
in
ref
ou
cus
seg

S

C
tl
n
o
tl
p
(c
s
u
c
it
is
c
to

h
tl
o
t
p
tl
fr
a
tl
c
q
s
T
I
w
s
tl
o
f
fu
a
a
r
tl
fr
o
m
d
t
p
it
f
tl
c
a

L
p
w
tl
fo
s

L
ir
tl
s

In [53]:
```
#
ou

#
cu

#
cu

#
cu
```

In [54]:

Out[54]:

In [55]:

In

Out