Customer Segmentation and Purchase Prediction in E-commerce

PROJECT REPORT

Submitted in the partial fulfilment of the requirements

for award of the

Six Months Online Certificate Course

in

Data Science with Python Programming

Course Duration: [22-01-2024 to 21-07-2024]

By

SIRUMALLA

NIKHILESWAR

(Ht.No.2406DS107)

DIRECTORATE OF INNOVATIVE LEARNING & TEACHING
(DILT)
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
(Formerly SCDE_SCHOOL OF CONTINUING AND DISTANCE EDUCATION)
Kukatpally, Hyderabad, Telangana State, INDIA- 500 085

JULY 2024

ABSTRACT

In the highly competitive landscape of e-commerce, understanding customer behaviour is crucial for tailoring marketing strategies and improving customer retention. This project aims to analyse the purchasing behaviour of approximately 4000 customers over a one-year period, using an Ecommerce dataset. The primary objective is to develop a robust model to segment customers based on their purchasing patterns and predict future purchases from new customers starting from their first transaction.

The project will employ the Recency, Frequency, and Monetary (RFM) model to generate key features that describe customer behaviour. These features will be used to perform customer segmentation through K-Means clustering, identifying distinct groups with similar purchasing patterns. Following segmentation, a predictive model will be developed to forecast future purchases using machine learning algorithms, such as K-Means Clustering.

The methodology includes data preprocessing, exploratory data analysis, feature engineering using the RFM model, clustering analysis for customer segmentation, and the development of a predictive model and it will enable the anticipation of future purchases based on initial purchase behaviour.

The outcomes of this project will provide actionable insights for personalized marketing, enhanced customer relationship management, and improved strategic decision-making. By accurately predicting customer behaviour, e-commerce businesses can optimize their marketing efforts, increase customer satisfaction, and ultimately drive higher sales.

TABLE OF CONTENTS

INTRODUCTION

In today's data-driven era, businesses are increasingly focusing on understanding customer behaviour to enhance customer satisfaction and loyalty. Customer segmentation and recommendation systems are pivotal in achieving these objectives.

Customer segmentation

It is the process of dividing a customer base into distinct groups that share similar characteristics. This allows businesses to tailor their marketing strategies, ensuring that each segment receives the most relevant messages and offers. Effective segmentation leads to improved customer engagement, retention, and ultimately, profitability.

Benefits of customer segmentation:

- **Targeted marketing:** By understanding customer segments, businesses can create targeted marketing campaigns that resonate with each group's specific needs and preferences. For example, a business might send targeted email campaigns to customers in a "high-value" segment with different offers than they would send to customers in a "low-value" segment.
- **Increased sales:** Targeted marketing leads to a higher chance of converting leads into sales.
- **Improved customer satisfaction:** Customers are more likely to be satisfied when they receive marketing messages that are relevant to them.

Recommendation systems

These are algorithms designed to suggest products or services to users based on their preferences and behaviours. By analysing past interactions, these systems can predict what a customer might be interested in next, thus enhancing their shopping experience. Personalized recommendations not only boost sales but also increase customer satisfaction by making the shopping experience more intuitive and enjoyable.

- **Collaborative filtering:** is a technique that recommends products to a customer based on the preferences of other customers who have similar preferences. For example, if a customer A buys products X, Y, and Z, and customer B also buys products X and Y, then a collaborative filtering recommendation system might recommend product Z to customer B.
- **Content-based filtering:** is a technique that recommends products to a customer based on the characteristics of the products that the customer has purchased in the past. For example, if a customer has purchased a book on machine learning, then a content-based filtering recommendation system might recommend other books on machine learning to that customer.
- **Hybrid approaches:** combine collaborative filtering and content-based filtering techniques.

Benefits of recommendation systems:

- **Increased sales:** By recommending products that customers are likely to be interested in, recommendation systems can help businesses increase sales.
- **Improved customer experience:** Recommendation systems can help customers discover new products that they might enjoy, which can lead to a more positive customer experience.
- **Reduced customer churn:** By providing customers with relevant recommendations, recommendation systems can help reduce customer churn.

This project aims to develop a robust customer segmentation and recommendation system using Python, leveraging advanced data science techniques. The process involves several critical steps:

1. **Data Collection**: Gathering a dataset that includes customer information such as demographics, purchase history, and browsing behaviour.
2. **Data Preprocessing**: Cleaning the data to handle missing values, outliers, and normalizing the data to ensure consistency.
3. **Customer Segmentation**: Implementing K-Means clustering to segment customers into distinct groups based on their characteristics and behaviours.
4. **Recommendation System**: Developing a collaborative filtering-based recommendation system to suggest products or services to customers based on their preferences and similarities with other customers.

LITERATUE REVIEW

1. Prof. Nikhil Patankar et.al(2021) "**Customer Segmentation Using Machine Learning**" explores the use of the K-means clustering algorithm to classify customers based on behavioural characteristics such as spending habits and annual income. By focusing on these aspects, the study aims to enhance customer segmentation, enabling companies to tailor marketing strategies more effectively. The proposed system involves data gathering, feature extraction, K-means classification, hyperparameter tuning, and data visualization. The results show that this method forms distinct customer groups, allowing for more personalized marketing, although it may increase marketing costs and face challenges with smaller segments. Overall, the study concludes that machine learning-based customer segmentation can significantly improve marketing efficiency and customer satisfaction by addressing individual preferences.

2. Miguel Alves Gomes and Tobias Meisen (2023)"**A Review on Customer Segmentation Methods for Personalized Customer Targeting in E-commerce Use Cases**" A comprehensive review of various customer segmentation methods used for personalized marketing in e-commerce. The study identifies and evaluates 105 publications from 2000 to 2022, outlining a four-phase process: data collection, customer representation, customer analysis through segmentation, and customer targeting. The review highlights that manual feature selection and RFM (Recency, Frequency, Monetary) analysis are predominant in

customer representation, while K-means clustering is the most commonly used segmentation method. The authors also discuss the importance of adapting segmentation strategies to dynamic customer behaviours and legal regulations, emphasizing that personalized marketing based on thorough customer analysis can significantly enhance customer relationship management and business success.

3. Malay Sarkar et.al **"Optimizing E-Commerce Profits: A Comprehensive Machine Learning Framework for Dynamic Pricing and Predicting Online Purchases"** presents a robust framework aimed at enhancing e-commerce profitability through dynamic pricing strategies and purchase prediction models. The study leverages machine learning algorithms to optimize pricing, emphasizing the selection of appropriate purchase prices rather than the lowest possible options. Targeting primarily inventory-led e-commerce companies, the model's applicability extends to online marketplaces without inventories. The framework integrates data from various sources such as visit attributes, visitor details, purchase history, web data, and contextual insights, focusing on predicting purchases within customer segments. The study underlines the significance of web mining, big data technologies, and machine learning in developing effective dynamic pricing and purchase prediction systems, with a logical progression towards personalized adaptive pricing strategies for future research.

4. Kamil Matuszelański et.al(2022) **"Customer Churn in Retail E-Commerce Business: Spatial and Machine Learning Approach"** Explores predicting customer churn in an e-commerce retail store in Brazil using a combination of numerical data on orders, textual reviews, and socio-geo-demographic data. The study involves three stages: pre-processing text reviews with techniques like Latent Dirichlet Allocation to identify topics, spatial analysis using DBSCAN to categorize customer locations, and modelling with machine learning algorithms such as Extreme Gradient Boosting (XGBoost) and logistic regression. Key findings indicate that churn is influenced by factors such as payment value for the first order, number of items bought, shipping cost, product categories, demographic environment, and customer location, while population density and review metrics are non-influential.

PROBLEM STATEMENT

In this project, we aim to transform 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 unique profiles and preferences of various customer groups, thus amplifying the efficiency of marketing strategies and fostering increased sales. Subsequently, we intend to develop a recommendation system that suggests top-selling products to customers within each segment who haven't purchased those items yet, enhancing marketing efficacy and fostering increased sales.

OBJECTIVES

This project has two primary objectives that work in tandem to improve customer experience and potentially increase sales within the retail industry. Here's a breakdown of each objective with additional details:

1. Uncover Distinct Customer Segments

This delves into the world of customer behaviour by uncovering distinct customer segments. We'll meticulously analyse customer data to identify shared characteristics and purchase patterns. Through machine learning techniques like K-Means clustering, we will group customers with similar features into unique segments. Each segment will represent a distinct customer profile, characterized by specific demographics, purchase frequency, and product preferences. By understanding these distinct segments, we gain a deeper grasp of the customer base and their needs.

- Identify Key Customer Characteristics.
- Segment Customers based on Shared Characteristics.
- Analyse and Profile Each Segment.

2. Develop a Personalized Recommendation System

Imagine a customer browsing online and encountering recommendations that feel hand-picked just for them. This personalized touch is precisely what we aim to achieve by developing a recommendation system. Leveraging the insights gleaned from customer segmentation, the system will recommend products tailored to individual customers. This will take into account an

individual's segment membership and past purchase history. By suggesting relevant products that align with their interests and past purchases, we aim to foster a sense of connection, leading to increased customer satisfaction and engagement. Ultimately, these satisfied and engaged customers are more likely to convert, potentially translating into higher sales for the retail company.

- Leverage Customer Segmentation for Targeted Recommendations.

METHODOLOGY

- **Data Cleaning & Transformation**: Clean the dataset by handling missing values, duplicates, and outliers, preparing it for effective clustering.

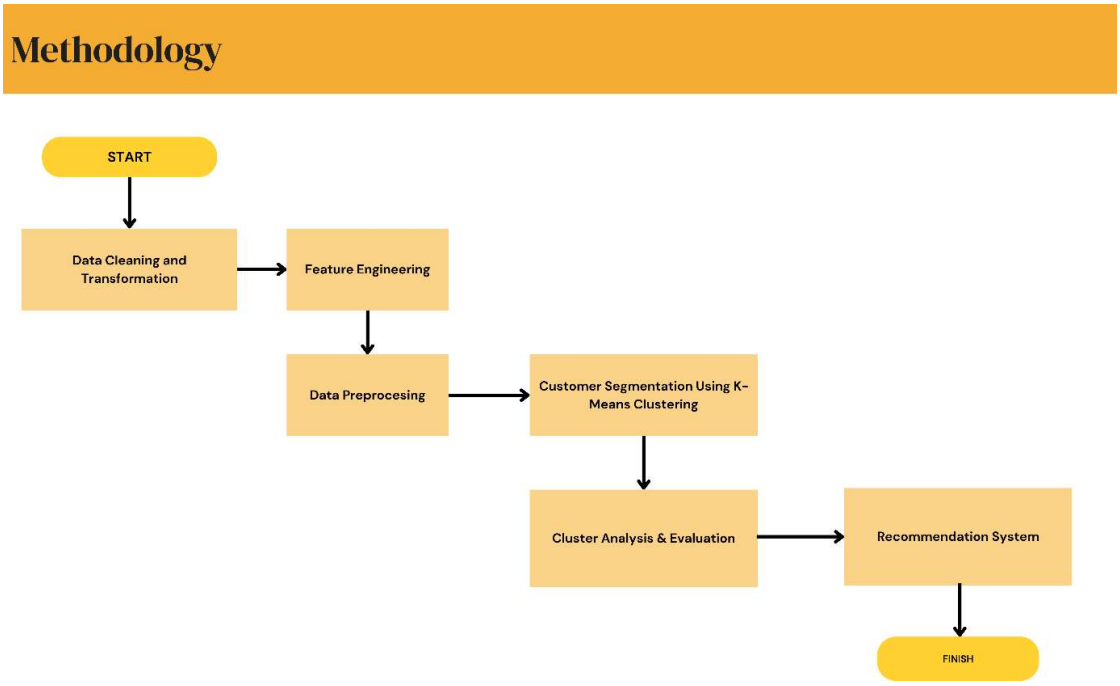
    1. Handling Missing Values
    2. Handling Duplicates
    3. Correcting Anomalies
    4. Outlier Treatment

- **Feature Engineering**: Develop new features based on the transactional data to create a customer-centric dataset, setting the foundation for customer segmentation.

    1. RFM Features
    2. Product Diversity
    3. Behavioural Features
    4. Seasonality and Trends

- **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**: Analyse 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.



ALGORITHM

K-Means Clustering Algorithm

Introduction

The K-Means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the *inertia* or within-cluster sum-of-squares.This algorithm requires the number of clusters to be specified. It scales well to large numbers of samples and has been used across a large range of application areas in many different fields.

K-means clustering is a popular unsupervised machine learning algorithm used to partition a dataset into *k* distinct clusters, where each data point belongs to the cluster with the nearest mean (centroid). The goal is to minimize the variance within each cluster, making the clusters as compact and separated as possible.

Steps of the Algorithm

1. **Initialization**: Select *k* initial centroids randomly from the dataset.
2. **Assignment**: Assign each data point to the nearest centroid, forming *k* clusters.
3. **Update**: Calculate the new centroids as the mean of all data points assigned to each cluster.
4. **Iteration**: Repeat the assignment and update steps until the centroids no longer change significantly (convergence).

Mathematical Formulation

The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster "centroids"; note that they are not, in general, points from X, although they live in the same space.

The K-means algorithm aims to choose centroids that minimise the **inertia**, or **within-cluster sum-of-squares criterion**:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

Inertia

Inertia can be recognized as a measure of how internally coherent clusters are. It suffers from various drawbacks:

- Inertia makes the assumption that clusters are convex and isotropic, which is not always the case. It responds poorly to elongated clusters, or manifolds with irregular shapes.
- Inertia is not a normalized metric: we just know that lower values are better and zero is optimal. But in very high-dimensional spaces, Euclidean distances tend to become inflated (this is an instance of the so-called "curse of dimensionality"). Running a dimensionality reduction algorithm such as Principal Component Analysis(PCA) prior to k-means clustering can alleviate this problem and speed up the computations.

Choosing the Number of Clusters (k)

- **Elbow Method**: Plot the sum of squared distances (inertia) against the number of clusters. The optimal *k* is at the "elbow point" where the inertia decreases sharply.
- **Silhouette Score**: Measures how similar a data point is to its own cluster compared to other clusters. Scores range from -1 (incorrect clustering) to 1 (highly dense clustering).

Advantages

- **Simplicity**: Easy to implement and understand.
- **Scalability**: Efficiently handles large datasets.
- **Speed**: Generally faster convergence compared to other clustering algorithms.

Disadvantages

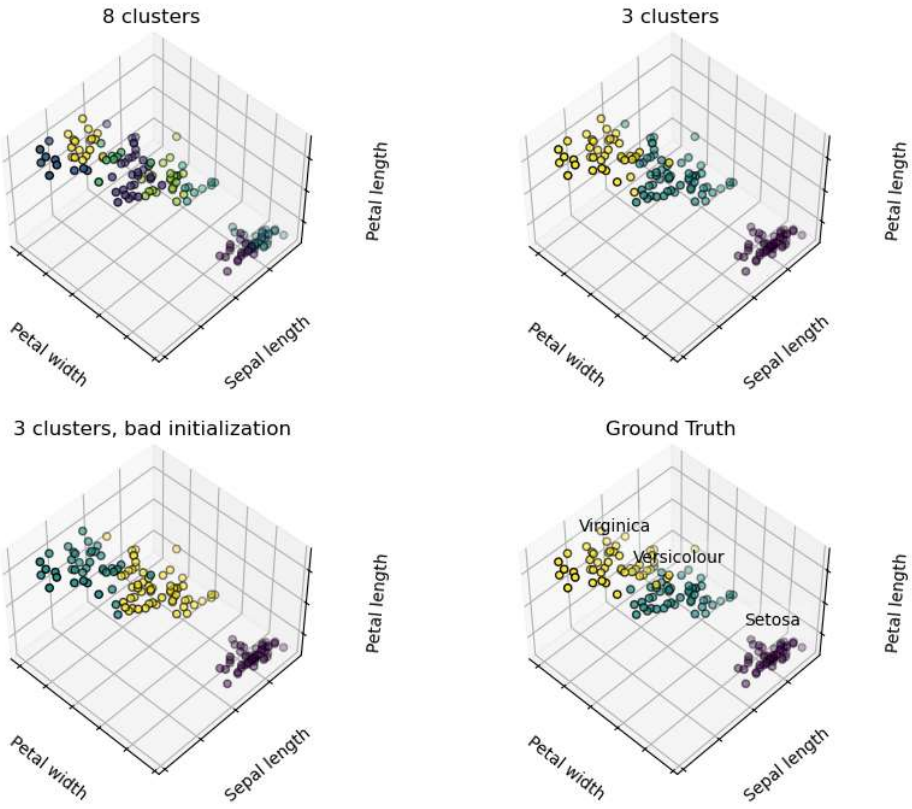
- **Fixed Number of Clusters**: Requires pre-specifying the number of clusters *k.*
- **Sensitivity to Initial Centroids**: Results can vary with different initial centroids.
- **Assumption of Spherical Clusters**: Assumes clusters are spherical and of similar size.

Practical Enhancements

- **K-means++**: Improves centroid initialization by selecting initial points that are distant from each other, leading to faster convergence and better clustering results.
- **Mini-Batch K-means**: Uses small random samples (mini-batches) to update centroids, making the algorithm faster and more efficient for large datasets.

Below Plot Shows:

- Top left: What a K-means algorithm would yield using 8 clusters.
- Top right: What using three clusters would deliver.
- Bottom left: What the effect of a bad initialization is on the classification process: By setting n_init to only 1 (default is 10), the number of times that the algorithm will be run with different centroid seeds is reduced.
- Bottom right: The ground truth.



IMPLEMENTATION

1.CHOOSING DATASET FOR THE PROJECT

Dataset : Online Retail - UCI Machine Learning Repository

Source : UCI Machine Learning Repository

2.SETUP AND INITIALIZATION

- **Importing Necessary Libraries**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

- **Loading the Dataset**

```
df=pd.read_csv("C:/Users/nikhi/Downloads/data1.csv",encoding="ISO-8859-1")
```

```
df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

- **Description of the Columns**

COLUMN	DESCRIPTION OF EACH PRODUCT
InvoiceNo	Code representing each unique transaction. If 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.

```
#description of each column
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
```

```
#Statistics of numerical columns
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Quantity	541909.0	9.552250	218.081158	-80995.00	1.00	3.00	10.00	80995.0
UnitPrice	541909.0	4.611114	96.759853	-11062.06	1.25	2.08	4.13	38970.0
CustomerID	406829.0	15287.690570	1713.600303	12346.00	13953.00	15152.00	16791.00	18287.0

```
#Summary of categorical columns
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

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.

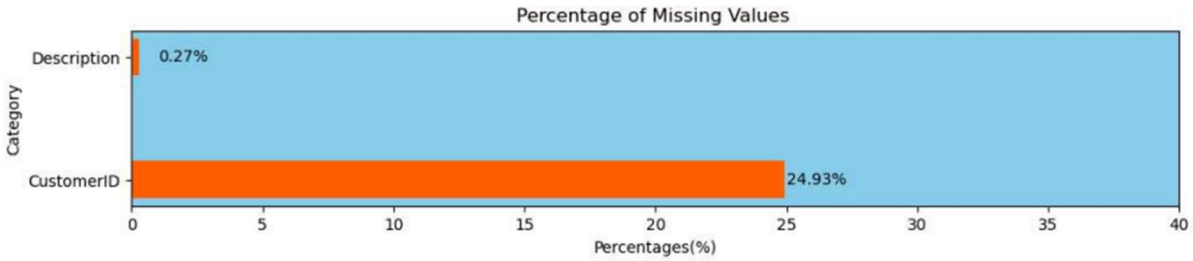
3.DATA CLEANING AND TRANSFORMATION

- **Handling Missing Values**

```
#Knowing the percentage of Blanks in CustomerID and Description
print("Percentage of Empty Blanks in CustomerID is {:.2f} %".format(df['CustomerID']
                                                                   .isnull().sum()/df.shape[0]*100))
print("Percentage of Empty Blanks in Description is {:.2f} %".format(df['Description']
                                                                    .isnull().sum()/df.shape[0]*100))
```

```
Percentage of Empty Blanks in CustomerID is 24.93 %
Percentage of Empty Blanks in Description is 0.27 %
```

```
#Ploting Bar Graph
Categories=["CustomerID","Description"]
Percentages=[(df['CustomerID'].isnull().sum()/df.shape[0]*100),(df['Description']
                                                               .isnull().sum()/df.shape[0]*100)]
fig,ax=plt.subplots(figsize=(12,2))
bars=ax.barh(Categories,Percentages,color='#ff6200',height=0.3)
ax.set_xlim([0,40])
ax.set_facecolor('skyblue')
for bar,percentage in zip(bars,Percentages):
    ax.text(int(percentage)+1,bar.get_y()+bar.get_height()/2,
                                            f"{percentage:.2f}%",ha="left",va="center")
ax.set_xlabel("Percentages(%)")
ax.set_ylabel("Category")
ax.set_title("Percentage of Missing Values ")
plt.show()
```




```
#Extract the Rows which are having missing values in Customer ID and Description
df[df["CustomerID"].isnull() | df["Description"].isnull()].head()
```

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

```
#Removing the rows which are having empty values in CustomerID and Description
df=df.dropna(subset=["CustomerID","Description"])
```

```
#Size of the Dataframe after removing the empty rows
df.shape
```

```
(406829, 8)
```

```
#Verify still there are any other empty rows in the given dataframe
df.isnull().sum().sum()
```

```
0
```

- **Handling Duplicates**

```
#Duplicated rows in the dataset
duplicates=df[df.duplicated(keep=False)]
sorted_duplicates=duplicates.sort_values(by=["InvoiceNo","StockCode","Description",
                                             "CustomerID","Quantity"])
sorted_duplicates
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
494	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	12/1/2010 11:45	1.25	17908.0	United Kingdom
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	12/1/2010 11:45	1.25	17908.0	United Kingdom
485	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	12/1/2010 11:45	4.95	17908.0	United Kingdom
539	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	12/1/2010 11:45	4.95	17908.0	United Kingdom
489	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	12/1/2010 11:45	2.10	17908.0	United Kingdom
...	...	...	...	...	...	...	...	...
440149	C574510	22360	GLASS JAR ENGLISH CONFECTIONERY	-1	11/4/2011 13:25	2.95	15110.0	United Kingdom
461407	C575940	23309	SET OF 60 I LOVE LONDON CAKE CASES	-24	11/13/2011 11:38	0.55	17838.0	United Kingdom
461408	C575940	23309	SET OF 60 I LOVE LONDON CAKE CASES	-24	11/13/2011 11:38	0.55	17838.0	United Kingdom
529980	C580764	22667	RECIPE BOX RETROSPOT	-12	12/6/2011 10:38	2.95	14562.0	United Kingdom
529981	C580764	22667	RECIPE BOX RETROSPOT	-12	12/6/2011 10:38	2.95	14562.0	United Kingdom

```
#Remove the Duplicated Rows
df=df.drop_duplicates()
df
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	France

```
df.shape[0]
```

```
401604
```

- **Treating Cancelled Transactions**

To refine our understanding of customer behaviour 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 analyse these rows to understand their common characteristics or patterns:

```
#Knowing the Cancelled Transactions these are starting with C in InvoiceNo
cancelled_rows=df[df["InvoiceNo"].str.startswith("C")]
cancelled_rows
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
141	C536379	D	Discount	-1	12/1/2010 9:41	27.50	14527.0	United Kingdom
154	C536383	35004C	SET OF 3 COLOURED FLYING DUCKS	-1	12/1/2010 9:49	4.65	15311.0	United Kingdom
235	C536391	22556	PLASTERS IN TIN CIRCUS PARADE	-12	12/1/2010 10:24	1.65	17548.0	United Kingdom
236	C536391	21984	PACK OF 12 PINK PAISLEY TISSUES	-24	12/1/2010 10:24	0.29	17548.0	United Kingdom
237	C536391	21983	PACK OF 12 BLUE PAISLEY TISSUES	-24	12/1/2010 10:24	0.29	17548.0	United Kingdom
...	...	...	...	...	...	...	...	...
540449	C581490	23144	ZINC T-LIGHT HOLDER STARS SMALL	-11	12/9/2011 9:57	0.83	14397.0	United Kingdom
541541	C581499	M	Manual	-1	12/9/2011 10:28	224.69	15498.0	United Kingdom
541715	C581568	21258	VICTORIAN SEWING BOX LARGE	-5	12/9/2011 11:57	10.95	15311.0	United Kingdom
541716	C581569	84978	HANGING HEART JAR T-LIGHT HOLDER	-1	12/9/2011 11:58	1.25	17315.0	United Kingdom
541717	C581569	20979	36 PENCILS TUBE RED RETROSPOT	-5	12/9/2011 11:58	1.25	17315.0	United Kingdom

8872 rows × 8 columns

```
cancelled_rows.shape
```

```
(8872, 8)
```

```
cancelled_rows.describe().drop("CustomerID",axis=1)
```

	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

```
'''Creating a new Column for the given dataset categorizing Transaction_status as
"Cancelled" or "Completed" according to cancelled transactions'''

df["Transaction_Status"]=np.where(df["InvoiceNo"].astype(str).str.startswith("C"),
                                  "Cancelled","Completed")
```

```
#Lets find the no.of unique stock codes
noof_unique_stock_codes=df["StockCode"].nunique()
print(f"The no.of unique stock codes in the given set are : {noof_unique_stock_codes}")
```

```
The no.of unique stock codes in the given set are : 3684
```

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 behaviour 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 behaviours 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.

- **Correcting Stock Code Anomalies**

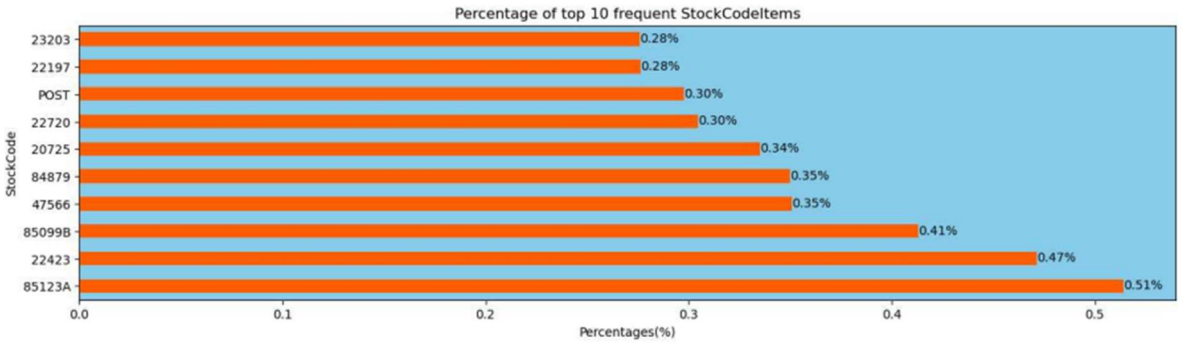
```
#Lets find the no.of unique stock codes
noof_unique_stock_codes=df["StockCode"].nunique()
print(f"The no.of unique stock codes in the given set are : {noof_unique_stock_codes}")
```

```
The no.of unique stock codes in the given set are : 3684
```

```
#Top 10 frequent Stock Codes
frequency=df["StockCode"].value_counts().iloc[:10]/len(df["StockCode"])*100
frequency
```

```
StockCode
85123A    0.514188
22423     0.471609
85099B    0.413093
47566     0.350843
84879     0.349847
20725     0.335156
22720     0.304778
POST      0.297806
22197     0.276392
23203     0.275894
Name: count, dtype: float64
```

```
#Plotting Bar Graph for the percentage of top 10 frequent StockCodeItems
fig,ax=plt.subplots(figsize=(16,4))
frequency.plot(kind='barh', color='#ff6200')
for i,j in enumerate(frequency):
    plt.text(j,i+0.25,f"{j:.2f}%",ha="left",va="top")
ax.set_facecolor("skyblue")
plt.xlabel("Percentages(%)")
plt.title("Percentage of top 10 frequent StockCodeItems")
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.

```
#Finding the no.of numeric characters in each StockCode
unique_stock_codes=df["StockCode"].unique()
pd.Series(unique_stock_codes).apply(lambda x:sum(c.isdigit() for c in str(x))).value_counts()
```

```
5    3676
0       7
1       1
Name: count, dtype: int64
```

```
#Finding the Stock Codes which are with 0 and 1 numeric characters
stock_codes_anamolous=[code for code in unique_stock_codes if
                       sum([i.isdigit() for i in code]) in (0,1)]
print("These are the stock_code_anamolous : ")
print("-"*30)
for i in stock_codes_anamolous:
    print(i)
```

```
These are the stock_code_anamolous :
------------------------------
POST
D
C2
M
BANK CHARGES
PADS
DOT
CRUK
```

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:

```
#Find the percentage of anamolous stock codes in the given dataset
total_noof_anamolous_codes=sum(df["StockCode"].isin(stock_codes_anamolous))
percentage_of_anamolous_codes=total_noof_anamolous_codes/len(df["StockCode"])*100
percentage_of_anamolous_codes
print(f"Total percentage of anamolous codes in the given data set is
            {percentage_of_anamolous_codes:.2f} %")
```

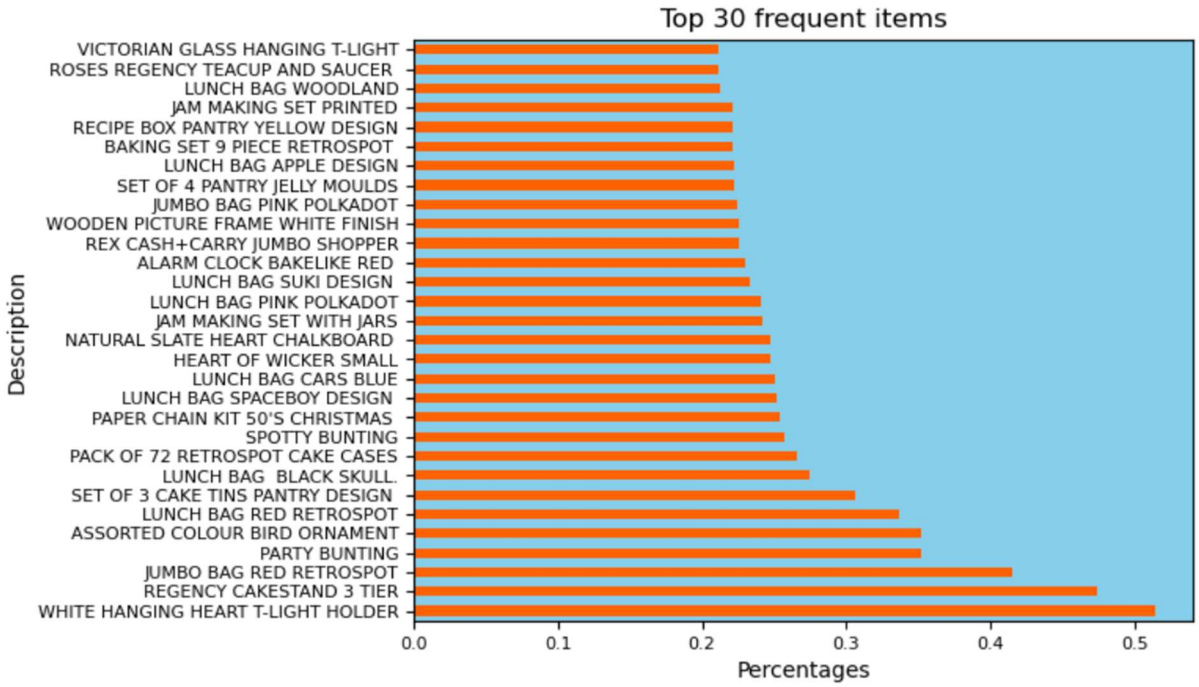
```
Total percentage of anamolous codes in the given data set is 0.48 %
```

```
#Removing the anamolous codes from the data set
df=df[~df["StockCode"].isin(stock_codes_anamolous)]
df.shape[0]
```

```
399689
```

- **Cleaning Description Column**

```
"""Calculate the frequency of top 30 frequencies and then
plot the percentages of them in the whole dataset"""
frequency_of_top_30=df["Description"].value_counts().iloc[:30]
percentage_of_top_30=frequency_of_top_30/len(df["Description"])*100
percentage_of_top_30.plot(kind="barh",color="#ff6200",fontsize=8)
plt.xlabel("Percentages")
plt.title("Top 30 frequent items")
plt.gca().set_facecolor("skyblue")
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.

```
#So description of the items which are not in the Capital letters
unique_description=df["Description"].unique()
anamolous_description=[c for c in unique_description if any(i.islower() for i in c)]
print("Anamolous description in the Dataset is : ")
print("-"*40)
for i in anamolous_description:
    print(i)
```

```
Anamolous description in the Dataset is :
----------------------------------------
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.

```
different_description=["Next Day Carriage","High Resolution Image"]
df=df[~df["Description"].isin(different_description)]
df["Description"]=df["Description"].str.upper()
```

```
df["Description"].shape[0]
```

```
399606
```

- **Treating Zero Unit Prices**

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

```
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

```
#Treating 0's unitprice
df[df["UnitPrice"]==0].describe()["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
Name: Quantity, dtype: float64
```

```
df=df[~(df["UnitPrice"]==0)]
df.shape[0]
```

```
399573
```

- **Outlier Treatment**

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.

```
df.reset_index(drop=True,inplace=False)
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Transaction_Status
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom	Completed
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom	Completed
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom	Completed
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom	Completed
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom	Completed
...	...	...	...	...	...	...	...	...	...
399568	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/2011 12:50	0.85	12680.0	France	Completed
399569	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/2011 12:50	2.10	12680.0	France	Completed
399570	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/2011 12:50	4.15	12680.0	France	Completed
399571	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/2011 12:50	4.15	12680.0	France	Completed
399572	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/2011 12:50	4.95	12680.0	France	Completed

399573 rows × 9 columns

4.FEATURE ENGINEERING

- ## RFM Features

RFM is a method used for analysing 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 behaviour and preferences, which is pivotal in personalizing marketing strategies and creating a recommendation system.

Recency(R)

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.

```
#Recency
#Convert InvoiceData to datetime data
df["InvoiceDate"]=pd.to_datetime(df["InvoiceDate"])
#Extract date from Datetime of InvoiceDate
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 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 of each customer
customer_data["days_since_last_purchase"]=(most_recent_date-customer_data["InvoiceDay"]).dt.days
customer_data.drop(columns=["InvoiceDay"],inplace=True)
customer_data
```

	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
...	...	...
4357	18280.0	277
4358	18281.0	180
4359	18282.0	7
4360	18283.0	3
4361	18287.0	42

4362 rows × 2 columns

Note:

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

Frequency(F)

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 behaviour 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.

```
#Frequency
#Find total transactions made by each customer
total_transactions=df.groupby("CustomerID")["InvoiceNo"].nunique().reset_index()
total_transactions.rename(columns={"InvoiceNo":"Total_Transactions"},inplace=True)
total_transactions
```

	CustomerID	Total_Transactions
0	12346.0	2
1	12347.0	7
2	12348.0	4
3	12349.0	1
4	12350.0	1
...	...	...
4357	18280.0	1
4358	18281.0	1
4359	18282.0	3
4360	18283.0	16
4361	18287.0	3

4362 rows × 2 columns

```
#Calculate the total no.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)
total_products_purchased
```

	CustomerID	Total_Products_Purchased
0	12346.0	0
1	12347.0	2458
2	12348.0	2332
3	12349.0	630
4	12350.0	196
...	...	...
4357	18280.0	45
4358	18281.0	54
4359	18282.0	98
4360	18283.0	1355
4361	18287.0	1586

4362 rows × 2 columns

```
#Merge the new Features into customer_data dataframe
customer_data=pd.merge(customer_data,total_transactions,on="CustomerID")
customer_data=pd.merge(customer_data,total_products_purchased,on="CustomerID")
customer_data
```

	CustomerID	days_since_last_purchase	Total_Transactions	Total_Products_Purchased
0	12346.0	325	2	0
1	12347.0	2	7	2458
2	12348.0	75	4	2332
3	12349.0	18	1	630
4	12350.0	310	1	196
...	...	...	...	...
4357	18280.0	277	1	45
4358	18281.0	180	1	54
4359	18282.0	7	3	98
4360	18283.0	3	16	1355
4361	18287.0	42	3	1586

4362 rows × 4 columns

Monetary(M)

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 behaviour of customers per transaction, which can assist in tailoring marketing strategies and offers to different customer segments based on their average spending patterns.

```
#Monetory
#Calculate the total amount spent by each customer
df["Total_Spent"]=df["Quantity"]*df["UnitPrice"]
total_amount_spent=df.groupby("CustomerID")["Total_Spent"].sum().reset_index()
total_amount_spent.rename(columns={"UnitPrice":"Total_Amount_Spent"},inplace=True)
total_amount_spent
```

	CustomerID	Total_Spent
0	12346.0	0.00
1	12347.0	4310.00
2	12348.0	1437.24
3	12349.0	1457.55
4	12350.0	294.40
...	...	...
4357	18280.0	180.60
4358	18281.0	80.82
4359	18282.0	176.60
4360	18283.0	2039.58
4361	18287.0	1837.28

4362 rows × 2 columns

```
#Merge the new Feature into customer_data dataframe
customer_data=pd.merge(customer_data,total_amount_spent,on="CustomerID")
customer_data
```

	CustomerID	days_since_last_purchase	Total_Transactions	Total_Products_Purchased	Total_Spent
0	12346.0	325	2	0	0.00
1	12347.0	2	7	2458	4310.00
2	12348.0	75	4	2332	1437.24
3	12349.0	18	1	630	1457.55
4	12350.0	310	1	196	294.40
...	...	...	...	...	...
4357	18280.0	277	1	45	180.60
4358	18281.0	180	1	54	80.82
4359	18282.0	7	3	98	176.60
4360	18283.0	3	16	1355	2039.58
4361	18287.0	42	3	1586	1837.28

4362 rows × 5 columns

```
#Calculate the average amount spent by each customer in a transaction
customer_data["Average_Amount_Spent"]=(customer_data["Total_Spent"])/(customer_data["Total_Transactions"])
customer_data
```

	CustomerID	days_since_last_purchase	Total_Transactions	Total_Products_Purchased	Total_Spent	Average_Amount_Spent
0	12346.0	325	2	0	0.00	0.000000
1	12347.0	2	7	2458	4310.00	615.714286
2	12348.0	75	4	2332	1437.24	359.310000
3	12349.0	18	1	630	1457.55	1457.550000
4	12350.0	310	1	196	294.40	294.400000
...	...	...	...	...	...	...
4357	18280.0	277	1	45	180.60	180.600000
4358	18281.0	180	1	54	80.82	80.820000
4359	18282.0	7	3	98	176.60	58.866667
4360	18283.0	3	16	1355	2039.58	127.473750
4361	18287.0	42	3	1586	1837.28	612.426667

4362 rows × 6 columns

- **Product Diversity**

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 recommendations.

```
#Product Diversity
#Calculate the unique products purchased by each customer
unique_products_purchased=df.groupby("CustomerID")["StockCode"].nunique().reset_index()
unique_products_purchased.rename(columns={"StockCode":"Unique_Products"},inplace=True)
#Merge the new Feature with customer_data dataframe
customer_data=pd.merge(customer_data,unique_products_purchased,on="CustomerID")
customer_data
```

	CustomerID	days_since_last_purchase	Total_Transactions	Total_Products_Purchased	Total_Spent	Average_Amount_Spent	Unique_Products
0	12346.0	325	2	0	0.00	0.000000	1
1	12347.0	2	7	2458	4310.00	615.714286	103
2	12348.0	75	4	2332	1437.24	359.310000	21
3	12349.0	18	1	630	1457.55	1457.550000	72
4	12350.0	310	1	196	294.40	294.400000	16
...	...	...	...	...	...	...	...
4357	18280.0	277	1	45	180.60	180.600000	10
4358	18281.0	180	1	54	80.82	80.820000	7
4359	18282.0	7	3	98	176.60	58.866667	12
4360	18283.0	3	16	1355	2039.58	127.473750	262
4361	18287.0	42	3	1586	1837.28	612.426667	59

4362 rows × 7 columns

```
#Extract day of the week and hour from InvoiceDate
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)
customer_data=pd.merge(customer_data,average_days_between_purchases,on="CustomerID")
customer_data
df
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Transaction_Status	InvoiceDay	Total_Spent	Day_Of_Week	Hour
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	Completed	2010-12-01	15.30	2	8
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	Completed	2010-12-01	20.34	2	8
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	Completed	2010-12-01	22.00	2	8
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	Completed	2010-12-01	20.34	2	8
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	Completed	2010-12-01	20.34	2	8

- **Behavioural Features**

  In this step, we aim to understand and capture the shopping patterns and behaviours 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.

  Favourite 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.

  Favourite Shopping Hour: This refers to the hour of the day when the customer shops the most. Identifying the favourite 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.

```
#Find the favourite shopping day in the week of the customer
favourite_shopping_day=df.groupby(["CustomerID","Day_Of_Week"]).size().reset_index(name="Count")
favourite_shopping_day=favourite_shopping_day.loc[favourite_shopping_day.groupby("CustomerID")["Count"].idxmax()][["CustomerID","Day_Of_Week"]]
favourite_shopping_day.rename(columns={"Day_Of_Week":"Favourite_Shopping_Day"},inplace=True)
#Find the favourite shopping hour of the customer
favourite_shopping_hour=df.groupby(["CustomerID","Hour"]).size().reset_index(name="Count")
favourite_shopping_hour=favourite_shopping_hour.loc[favourite_shopping_hour.groupby("CustomerID")["Count"].idxmax()][["CustomerID","Hour"]]
favourite_shopping_hour.rename(columns={"Hour":"Favourite_Shopping_Hour"},inplace=True)
#Merge favourite_shopping_day and favourite_shopping_hour into customer_data
customer_data=pd.merge(customer_data,favourite_shopping_day,on="CustomerID")
customer_data=pd.merge(customer_data,favourite_shopping_hour,on="CustomerID")
customer_data
```

Total_Products_Purchased	Total_Spent	Average_Amount_Spent	Unique_Products	Average_Days_Between_Purchases	Favourite_Shopping_Day	Favourite_Shopping_Hour
0	0.00	0.000000	1	0.000000	1	10
2458	4310.00	615.714286	103	2.016575	1	14
2332	1437.24	359.310000	21	10.884615	3	19
630	1457.55	1457.550000	72	0.000000	0	9
196	294.40	294.400000	16	0.000000	2	16

- **Geographic Features**

```
#Group by CustomerID and Country to get the number of transactions per country for each customer
customer_country=df.groupby(['CustomerID', 'Country']).size().reset_index(name='Number_of_Transactions')
#Get the country with maximum no.of transactions for each customer(in case customer has transactions from multiple countries)
customer_main_country=customer_country.sort_values('Number_of_Transactions',ascending=False).drop_duplicates('CustomerID')
#Creating a binary column whether a customer is from uk or not
customer_main_country['Is_UK']=customer_main_country['Country'].apply(lambda x: 1 if x == 'United Kingdom' else 0)
#Merge Is_UK into customer_data
customer_data = pd.merge(customer_data, customer_main_country[['CustomerID', 'Is_UK']], on='CustomerID', how='left')
customer_data
```

Total_Spent	Average_Amount_Spent	Unique_Products	Average_Days_Between_Purchases	Favourite_Shopping_Day	Favourite_Shopping_Hour	Is_UK
0.00	0.000000	1	0.000000	1	10	1
4310.00	615.714286	103	2.016575	1	14	0
1437.24	359.310000	21	10.884615	3	19	0
1457.55	1457.550000	72	0.000000	0	9	0
294.40	294.400000	16	0.000000	2	16	0

```
customer_data["Is_UK"].value_counts()
```

```
Is_UK
1    3866
0     416
Name: count, dtype: int64
```

- **Cancellation Insights**

  In this step, I am going to delve deeper into the cancellation patterns of customers to gain insights that can enhance our customer segmentation model. The features I am planning to introduce are:

  - Cancellation Frequency: This metric represents the total number of transactions a customer has canceled. Understanding the frequency of cancellations can help us identify customers who are more likely to cancel transactions. This could be an indicator of dissatisfaction or other issues, and understanding this can help us tailor strategies to reduce cancellations and enhance customer satisfaction.
  - Cancellation Rate: This represents the proportion of transactions that a customer has canceled out of all their transactions. This metric gives a normalized view of cancellation behavior. A high cancellation rate might be indicative of an unsatisfied customer segment. By identifying these segments, we can develop targeted strategies to improve their shopping experience and potentially reduce the cancellation rate.

```
#CancellationInsights
#CancellationFrequency
cancellation_df=df[df["Transaction_Status"]=="Cancelled"]
cancellation_frequency=cancellation_df.groupby("CustomerID")["Transaction_Status"].nunique().reset_index()
cancellation_frequency.rename(columns={"Transaction_Status":"Cancellation_Frequency"},inplace=True)
#Merge cancellation_frequency into customer_data
customer_data=pd.merge(customer_data,cancellation_frequency,on="CustomerID",how="left")
#Replace NaN values with 0
customer_data["Cancellation_Frequency"].fillna(0,inplace=True)
#CancellationRate
customer_data["Cancellation_Rate"]=customer_data["Cancellation_Frequency"]/customer_data["Total_Transactions"]
customer_data
```

Unique_Products	Average_Days_Between_Purchases	Favourite_Shopping_Day	Favourite_Shopping_Hour	Is_UK	Cancellation_Frequency	Cancellation_Rate
1	0.000000	1	10	1	1.0	0.500000
103	2.016575	1	14	0	0.0	0.000000
21	10.884615	3	19	0	0.0	0.000000
72	0.000000	0	9	0	0.0	0.000000
16	0.000000	2	16	0	0.0	0.000000

- **Seasonality and Trends**

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:

- **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.
- **Monthly_Spending_Std:** This feature indicates the variability in a customer's monthly spending. A higher value signals that the customer's spending fluctuates significantly month-to-month, perhaps indicating sporadic large purchases. In contrast, a lower value suggests more stable, consistent spending habits. Understanding this variability can help in crafting personalized promotions or discounts during periods they are expected to spend more.

- **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.

Toggle output scrolling

```
port linregress
#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
    else:
        #If there is only one data point no trend can be calculated,hence we return 0
        return 0
#Apply the calculate trend function to calculate the spending trend of each customer
spending_trends=monthly_spending.groupby("CustomerID")["Total_Spent"].apply(calculate_trend).reset_index()
#Rename the Total_Spent in spending_trends to Spending_Trend
spending_trends.rename(columns={"Total_Spent":"Spending_Trend"},inplace=True)
customer_data=pd.merge(customer_data,spending_trends,on="CustomerID")
customer_data
```

Favourite_Shopping_Day	Favourite_Shopping_Hour	Is_UK	Cancellation_Frequency	Cancellation_Rate	Monthly_Spending_Mean	Monthly_Spending_Std	Spending_Trend
1	10	1	1.0	0.500000	0.000000	0.000000	0.000000
1	14	0	0.0	0.000000	615.714286	341.070789	4.486071
3	19	0	0.0	0.000000	359.310000	203.875689	-100.884000
0	9	0	0.0	0.000000	1457.550000	0.000000	0.000000
2	16	0	0.0	0.000000	294.400000	0.000000	0.000000

CUSTOMER DATASET DESCRIPTION

Variable	Description
CustomerID	Identifier uniquely assigned to each customer, used to distinguish individual customers.
Days_Since_Last_Purchase	The number of days that have passed since the customer's last purchase.
Total_Transactions	The total number of transactions made by the customer.
Total_Products_Purchased	The total quantity of products purchased by the customer across all transactions.
Total_Spend	The total amount of money the customer has spent across all transactions.
Average_Transaction_Value	The average value of the customer's transactions, calculated as total spend divided by the number of transactions.
Unique_Products_Purchased	The number of different products the customer has purchased.
Average_Days_Between_Purchases	The average number of days between consecutive purchases made by the customer.
Day_Of_Week	The day of the week when the customer prefers to shop, represented numerically (0 for Monday, 6 for Sunday).
Hour	The hour of the day when the customer prefers to shop, represented in a 24-hour format.
Is_UK	A binary variable indicating whether the customer is based in the UK (1) or not (0).
Cancellation_Frequency	The total number of transactions that the customer has cancelled.
Cancellation_Rate	The proportion of transactions that the customer has cancelled, calculated as cancellation frequency divided by total transactions.
Monthly_Spending_Mean	The average monthly spending of the customer.
Monthly_Spending_Std	The standard deviation of the customer's monthly spending, indicating the variability in their spending pattern.
Spending_Trend	A numerical representation of the trend in the customer's spending over time. A positive value indicates an increasing trend, a negative value indicates a decreasing trend, and a value close to zero indicates a stable trend.

5.OUTLIER DETECTION AND TREATMENT

In this section, I will identify and handle outliers in our dataset. Outliers are data points that are significantly different from the majority of other points in the dataset. These points can potentially skew the results of our analysis, especially in k-means clustering where they can significantly influence the position of the cluster centroids. Therefore, it is essential to identify and treat these outliers appropriately to achieve more accurate and meaningful clustering results.

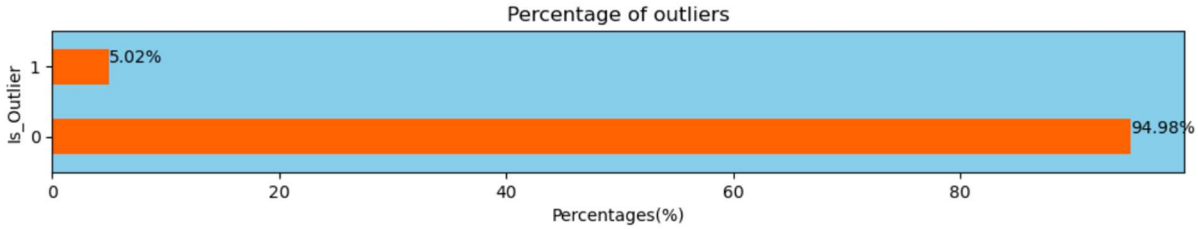
Given the multi-dimensional nature of the data, it would be prudent to use algorithms that can detect outliers in multi-dimensional spaces. I am going to use the Isolation Forest algorithm for this task. This algorithm works well for multi-dimensional data and is computationally efficient. It isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

Let's proceed with this approach:

```
#Outlier Detection and Treatment
#Initializing the IsolationForest Model with a contamination parameter of 0.05
from sklearn.ensemble import IsolationForest
model=IsolationForest(contamination=0.05,random_state=0)
#Fitting the model into the dataset(converting DataFrame to numpy to avoid Warning)
customer_data["Outlier_Scores"]=model.fit_predict(customer_data.iloc[:,1:].to_numpy())
customer_data["Is_Outlier"]=[1 if x==-1 else 0 for x in customer_data["Outlier_Scores"]]
customer_data
```

Favourite_Shopping_Hour	Is_UK	Cancellation_Frequency	Cancellation_Rate	Monthly_Spending_Mean	Monthly_Spending_Std	Spending_Trend	Outlier_Scores	Is_Outlier
10	1	1.0	0.500000	0.000000	0.000000	0.000000	1	0
14	0	0.0	0.000000	615.714286	341.070789	4.486071	1	0
19	0	0.0	0.000000	359.310000	203.875689	-100.884000	1	0
9	0	0.0	0.000000	1457.550000	0.000000	0.000000	1	0
16	0	0.0	0.000000	294.400000	0.000000	0.000000	1	0

```
#Find the percentage of inliers and outliers
outlier_percentage=customer_data["Is_Outlier"].value_counts(normalize=True)*100
#Plotting the percentages of inliers and outliers
plt.figure(figsize=(12,1.5))
outlier_percentage.plot(kind='barh', color='#ff6200')
for i,j in enumerate(outlier_percentage):
    plt.text(j,i+0.25,f"{j:.2f}%",ha="left",va="top")
plt.gca().set_facecolor("skyblue")
plt.xlabel("Percentages(%)")
plt.title("Percentage of outliers")
plt.show()
```



Inference:

From the above plot, we can observe that about 5% of the customers have been identified as outliers in our dataset. This percentage seems to be a reasonable proportion, not too high to lose a significant amount of data, and not too low to retain potentially noisy data points. It suggests that our isolation forest algorithm has worked well in identifying a moderate percentage of outliers, which will be critical in refining our customer segmentation.

Strategy:

Considering the nature of the project (customer segmentation using clustering), it is crucial to handle these outliers to prevent them from affecting the clusters' quality significantly. Therefore, I will separate these outliers for further analysis and remove them from our main dataset to prepare it for the clustering analysis.

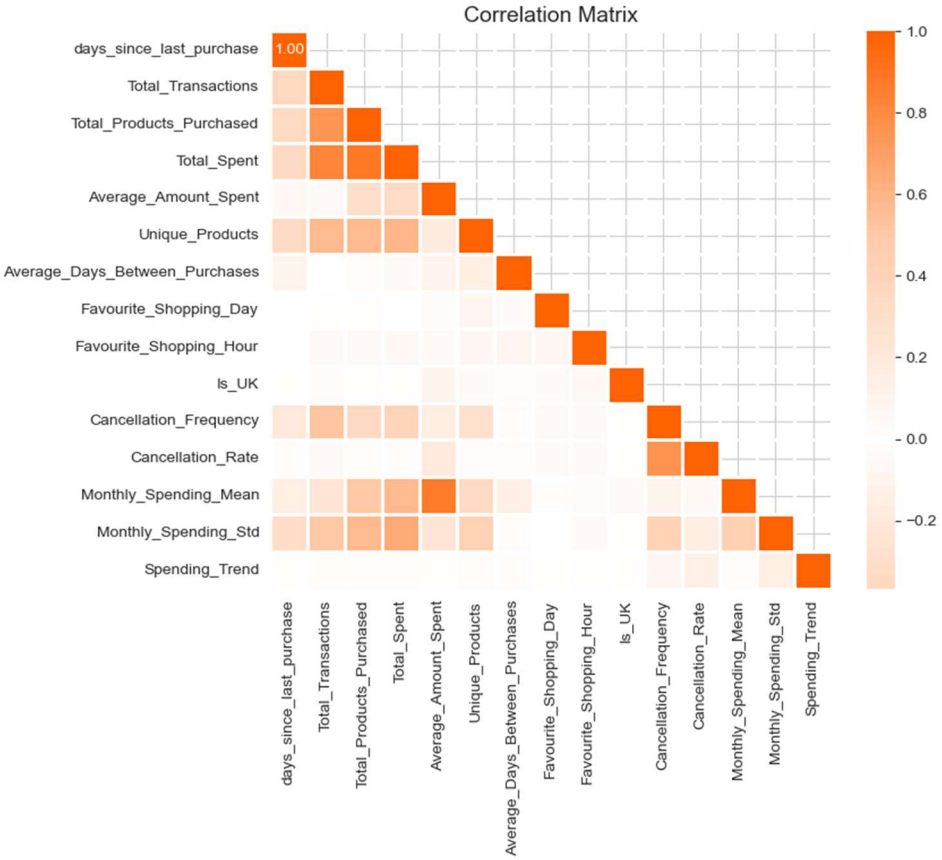
```
#Seperate the outliers data into a seperate data
outliers_data=customer_data[customer_data["Is_Outlier"]==1]
#Remove the outliers from customer_data set
customer_data=customer_data[customer_data["Is_Outlier"]==0]
#Drop Outlier_Scores and Is_Outlier from the customer_data
customer_data_cleaned=customer_data.drop(columns=["Outlier_Scores","Is_Outlier"]).reset_index(drop=True)
customer_data_cleaned
```

6.CORRELATION ANALYSIS

Before we proceed to KMeans clustering, it's essential to check the correlation between features in our dataset. The presence of multicollinearity, where features are highly correlated, can potentially affect the clustering process by not allowing the model to learn the actual underlying patterns in the data, as the features do not provide unique information. This could lead to clusters that are not well-separated and meaningful.

If we identify multicollinearity, we can utilize dimensionality reduction techniques like PCA. These techniques help in neutralizing the effect of multicollinearity by transforming the correlated features into a new set of uncorrelated variables, preserving most of the original data's variance. This step not only enhances the quality of clusters formed but also makes the clustering process more computationally efficient.

```
#CorrelationAnalysis
#Create White Grid
sns.set_style("whitegrid")
#Calculate the corelation matrix excluding the "CustomerID" column
corr=customer_data_cleaned.drop(columns=["CustomerID"]).corr()
#Define the custom colormap
colors=["#ff6200",'#ffcaa8', 'white', '#ffcaa8', '#ff6200']
from matplotlib.colors import LinearSegmentedColormap
my_cmap=LinearSegmentedColormap.from_list("custom_map",colors,N=256)
#Create a mask to only show the lower triangular matrix(since it is a mirror image around its)
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask,k=1)]=True
#Plot the heatmap
plt.figure(figsize=(8,6.4))
sns.heatmap(corr,mask=mask,cmap=my_cmap,annot=True,center=0,fmt=".2f",linewidths=2)
plt.title("Correlation Matrix",fontsize=14)
plt.show()
```



Inference:

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
- Cancellation_Rate and Cancellation_Frequency
- Total_Transactions and Total_Products_Purchased

These high correlations indicate that these variables move closely together, implying a degree of multicollinearity.

Before moving to the next steps, considering the impact of multicollinearity on KMeans clustering, it might be beneficial to treat this multicollinearity possibly through dimensionality reduction techniques such as PCA to create a set of uncorrelated variables. This will help in achieving more stable clusters during the KMeans clustering process.

7.FEATURE SCALING

Before we move forward with the clustering and dimensionality reduction, it's imperative to scale our features. This step holds significant importance, especially in the context of distance-based algorithms like K-means and dimensionality reduction methods like PCA. Here's why:

- **For K-means Clustering**: K-means relies heavily on the concept of 'distance' between data points to form clusters. When features are not on a similar scale, features with larger values can disproportionately influence the clustering outcome, potentially leading to incorrect groupings.

- **For PCA:** PCA aims to find the directions where the data varies the most. When features are not scaled, those with larger values might dominate these components, not accurately reflecting the underlying patterns in the data.

Methodology:

Therefore, to ensure a balanced influence on the model and to reveal the true patterns in the data, I am going to standardize our data, meaning transforming the features to have a mean of 0 and a standard deviation of 1. However, not all features require scaling. Here are the exceptions and the reasons why they are excluded:

- CustomerID: This feature is just an identifier for the customers and does not contain any meaningful information for clustering.
- Is_UK: This is a binary feature indicating whether the customer is from the UK or not. Since it already takes a value of 0 or 1, scaling it won't make any significant difference.
- Day_Of_Week: This feature represents the most frequent day of the week that the customer made transactions. Since it's a categorical feature represented by integers (1 to 7), scaling it would not be necessary.

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
#List of the columns not to be included
columns_to_be_excluded=["CustomerID","Is_UK","Favourite_Shopping_Day","index"]
#List of Colums that need to be scaled
columns_to_scale=customer_data_cleaned.columns.difference(columns_to_be_excluded)
#Copy the cleaned Dataset
customer_data_scaled=customer_data_cleaned.copy()
#Applying the scaler to the necessary columns in the dataset
customer_data_scaled[columns_to_scale]=scaler.fit_transform(customer_data_scaled[columns_to_scale])
customer_data_scaled
```

8.DIMENSIONALITY REDUCTION

Why We Need Dimensionality Reduction?

- **Multicollinearity Detected:** In the previous steps, we identified that our dataset contains multicollinear features. Dimensionality reduction can help us remove redundant information and alleviate the multicollinearity issue.
- **Better Clustering with K-means:** Since K-means is a distance-based algorithm, having a large number of features can sometimes dilute the meaningful underlying patterns in the data. By reducing the dimensionality, we can help K-means to find more compact and well-separated clusters.
- **Noise Reduction:** By focusing only on the most important features, we can potentially remove noise in the data, leading to more accurate and stable clusters.
- **Enhanced Visualization**: In the context of customer segmentation, being able to visualize customer groups in two or three dimensions can provide intuitive insights. Dimensionality reduction techniques can facilitate this by reducing the data to a few principal components which can be plotted easily.
- **Improved Computational Efficiency**: Reducing the number of features can speed up the computation time during the modelling process, making our clustering algorithm more efficient.

Which Dimensionality Reduction Method?

In this step, we are considering the application of dimensionality reduction techniques to simplify our data while retaining the essential information. Among various methods such as KernelPCA, ICA, ISOMAP, TSNE, and UMAP, I am starting with PCA (Principal Component Analysis). Here's why:

PCA is an excellent starting point because it works well in capturing linear relationships in the data, which is particularly relevant given the multicollinearity we identified in our dataset. It allows us to reduce the number of features in our dataset while still retaining a significant amount of the information, thus making our clustering analysis potentially more accurate and interpretable. Moreover, it is computationally efficient, which means it won't significantly increase the processing time.

However, it's essential to note that we are keeping our options open. After applying PCA, if we find that the first few components do not capture a significant amount of variance, indicating a loss of vital information, we might consider exploring other non-linear methods. These methods can potentially provide a more nuanced approach to dimensionality reduction, capturing complex patterns that PCA might miss, albeit at the cost of increased computational time and complexity.

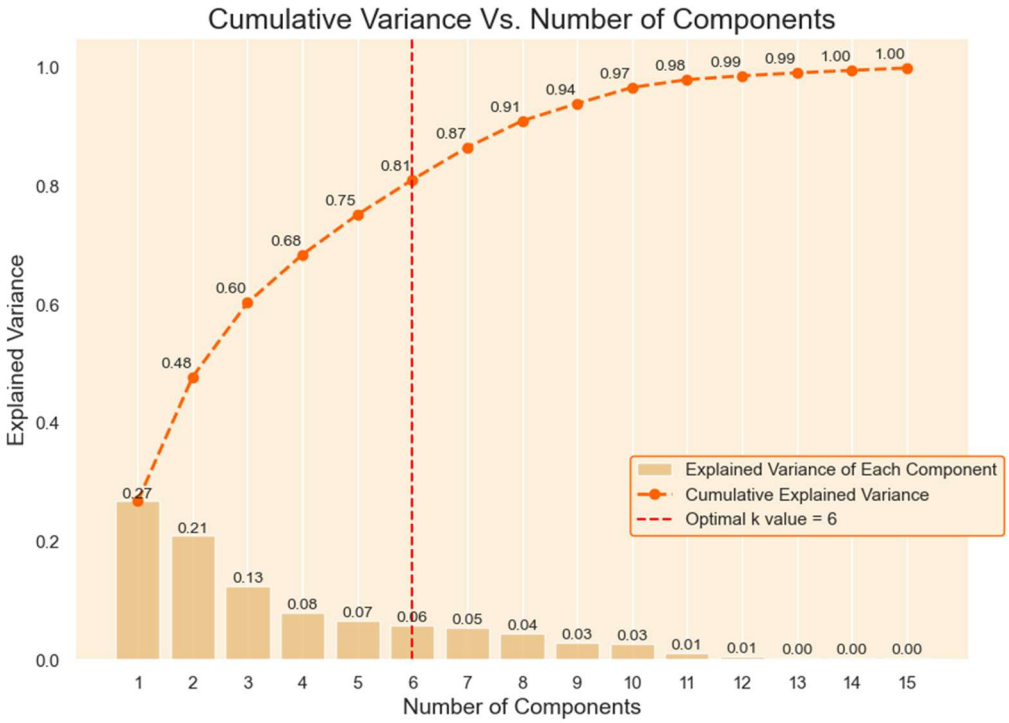
Methodology

I will apply PCA on all the available components and plot the cumulative variance explained by them. This process will allow me to visualize how much variance each additional principal

component can explain, thereby helping me to pinpoint the optimal number of components to retain for the analysis:

```
#Dimensionality Reduction
#Setting CustomerID as the index column
from sklearn.decomposition import PCA
customer_data_scaled.set_index("CustomerID",inplace=True)
#Apply PCA
pca=PCA().fit(customer_data_scaled)
#Calculate the Cumulative Sum of the Explained Variance
explained_variance_ratio=pca.explained_variance_ratio_
cumulative_explained_variance=np.cumsum(explained_variance_ratio)
#Set the optimal k value(based on our analysis,we can choose 6)
optimal_k=6
#Set Seaborn plot style
sns.set(rc={"axes.facecolor":"#fcf0dc"},style="darkgrid")
#Plot the cumulative explained variance against the no.of components
plt.figure(figsize=(10,7))
#Bar chart for the explained variance of each component
barplot=sns.barplot(x=list(range(1,len(cumulative_explained_variance)+1)),y=explained_variance_ratio,color="#fcc36d",alpha=0.8)
#Line Plot for the cumulative explained Variance
lineplot,=plt.plot(range(0,len(cumulative_explained_variance)),cumulative_explained_variance,marker="o",linestyle="--",color="#ff6200",linewidth=2)
#Plot optimal k value line
optimal_k_line=plt.axvline(optimal_k-1,color="red",linestyle="--",label=f"Optimal k value = {optimal_k}")
#set labels and title
plt.xlabel("Number of Components",fontsize=14)
plt.ylabel("Explained Variance",fontsize=14)
plt.title("Cumulative Variance Vs. Number of Components",fontsize=18)
```

```
#Customize ticks and legend
plt.xticks(range(0,len(cumulative_explained_variance)))
plt.legend(handles=[barplot.patches[0],lineplot, optimal_k_line],labels=
       ['Explained Variance of Each Component', 'Cumulative Explained Variance',f"Optimal k value = {optimal_k}"],
          loc=(0.62, 0.2),frameon=True,framealpha=1.0,  edgecolor='#ff6200')
#Display the variance rules for both graphs on the plots
x_offset=-0.3
y_offset=0.01
for i,(ev_ratio,cum_ev_ratio) in enumerate(zip(explained_variance_ratio,cumulative_explained_variance)):
    plt.text(i, ev_ratio, f"{ev_ratio:.2f}", ha="center", va="bottom", fontsize=10)
    if i > 0:
        plt.text(i + x_offset, cum_ev_ratio + y_offset, f"{cum_ev_ratio:.2f}", ha="center", va="bottom", fontsize=10)
plt.grid(axis='both')
plt.show()
```



Conclusion

The plot and the cumulative explained variance values indicate how much of the total variance in the dataset is captured by each principal component, as well as the cumulative variance explained by the first n components.

Here, we can observe that:

- The first component explains approximately 28% of the variance.
- The first two components together explain about 49% of the variance.
- The first three components explain approximately 61% of the variance, and so on.

To choose the optimal number of components, we generally look for a point where adding another component doesn't significantly increase the cumulative explained variance, often referred to as the "elbow point" in the curve.

From the plot, we can see that the increase in cumulative variance starts to slow down after the 6th component (which captures about 81% of the total variance).

Considering the context of customer segmentation, we want to retain a sufficient amount of information to identify distinct customer groups effectively. Therefore, retaining **the first 6 components** might be a balanced choice, as they together explain a substantial portion of the total variance while reducing the dimensionality of the dataset.

```
#Creating a PCA object with 6 components
pca=PCA(n_components=6)
#Fitting and transforming the original data to the new PCA dataframe
customer_data_pca=pca.fit_transform(customer_data_scaled)
customer_data_pca=pd.DataFrame(customer_data_pca,columns=["PC"+str(i+1) for i in range(pca.n_components_)])
customer_data_pca.index=customer_data_scaled.index
customer_data_pca
```

CustomerID	PC1	PC2	PC3	PC4	PC5	PC6
12346.0	-2.061059	1.789632	-3.309594	-1.987881	-0.380436	-1.977625
12347.0	3.232201	1.488895	1.940739	0.426827	-0.212927	0.563307
12348.0	0.593882	-0.571212	0.780029	0.593656	-0.781760	1.999711
12349.0	1.994591	2.733898	5.623730	-3.181078	1.419223	-0.854718
12350.0	-2.006587	0.429607	0.943748	-0.330629	-1.448269	-0.049521

Let's extract the coefficients corresponding to each principal component to better understand the transformation performed by PCA:

```
#Define a function to highlight the top 3 absolute values in each column of a dataframe
def highlight_top3(column):
    top3=column.abs().nlargest(3).index
    return ["background-color: #ffeacc" if i in top3 else "" for i in column.index]
#Create a PCA component DataFrame and apply the highlighting function
pc_df=pd.DataFrame(pca.components_.T,columns=["PC"+str(i+1) for i in range(pca.n_components_)],
                   index=customer_data_scaled.columns)
pc_df.style.apply(highlight_top3,axis=0)
```

	PC1	PC2	PC3	PC4	PC5	PC6
days_since_last_purchase	-0.222178	0.002350	0.061135	-0.285354	-0.159904	-0.328375
Total_Transactions	0.385258	-0.002044	-0.182657	0.274314	-0.039586	-0.049698
Total_Products_Purchased	0.419520	0.005698	0.041876	0.125909	0.012657	-0.047945
Total_Spent	0.443475	0.005034	0.042010	0.096015	0.013863	-0.059100
Average_Amount_Spent	0.182207	0.019369	0.518430	-0.372307	0.096083	-0.032009
Unique_Products	0.336369	-0.055010	0.013608	0.178665	-0.243777	0.041715
Average_Days_Between_Purchases	-0.029240	0.039384	-0.136500	0.196254	0.681576	0.229959
Favourite_Shopping_Day	0.008392	-0.993658	-0.033688	-0.053839	0.077767	-0.034948
Favourite_Shopping_Hour	-0.023207	-0.060320	0.024064	0.102754	-0.638268	0.435662
Is_UK	0.001530	-0.007360	-0.011402	0.016027	-0.008819	0.008518
Cancellation_Frequency	0.247991	0.041938	-0.498902	-0.256185	-0.061111	-0.144741
Cancellation_Rate	0.052207	0.043405	-0.503305	-0.478789	-0.051811	-0.160966
Monthly_Spending_Mean	0.302538	0.015641	0.394879	-0.355699	0.056597	-0.066053
Monthly_Spending_Std	0.354738	0.006678	-0.064369	-0.133376	0.104601	0.159937
Spending_Trend	-0.011987	0.000353	0.087511	0.395360	-0.072332	-0.748575

9.K-MEANS CLUSTERING

K-Means:

K-Means is an unsupervised machine learning algorithm that clusters data into a specified number of groups (K) by minimizing the within-cluster sum-of-squares (WCSS), also known as inertia. The algorithm iteratively assigns each data point to the nearest centroid, then updates the centroids by calculating the mean of all assigned points. The process repeats until convergence or a stopping criterion is reached.

- **Determining the Optimal No. of Clusters**

To ascertain the optimal number of clusters (k) for segmenting customers, I will explore two renowned methods:

- Elbow Method
- Silhouette Method

It's common to utilize both methods in practice to corroborate the results.

Elbow Method

What is the Elbow Method?

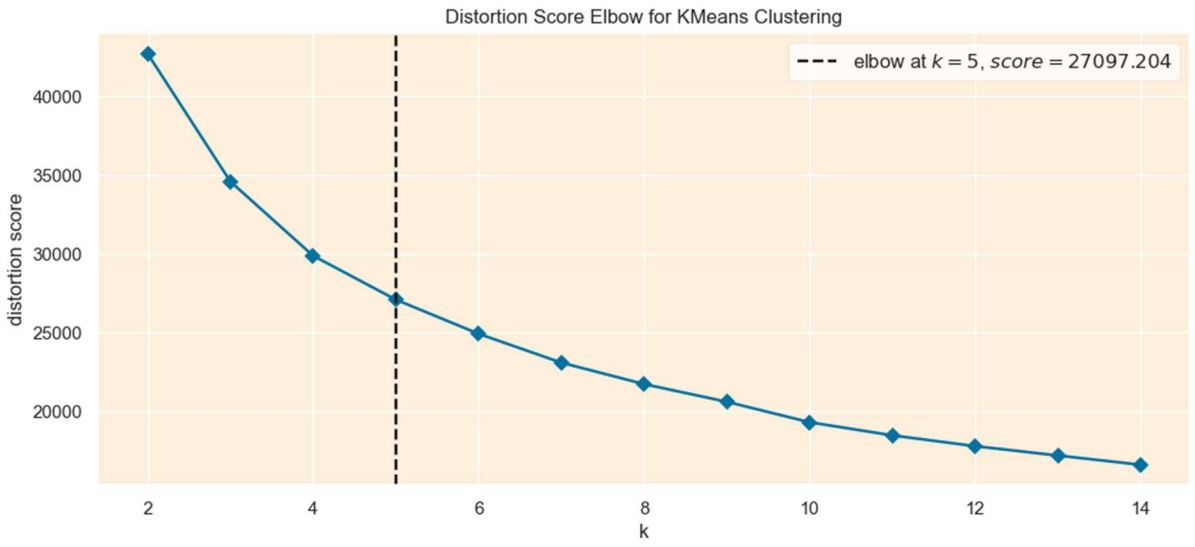
The Elbow Method is a technique for identifying the ideal number of clusters in a dataset. It involves iterating through the data, generating clusters for various values of k. The k-means algorithm calculates the sum of squared distances between each data point and its assigned cluster centroid, known 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.

Utilizing the YellowBrick Library

In this section, I will employ the YellowBrick library to facilitate the implementation of the Elbow method. YellowBrick, an extension of the Scikit-Learn API, is renowned for its ability to rapidly generate insightful visualizations in the field of machine learning.

```
#Determining the optimal no.of Clusters
#Elbow method
sns.set(style="darkgrid",rc={"axes.facecolor":"#fcf0dc"})
#Set the color palette for the plot
sns.set_palette(["#ff6200"])
#Instantiate the clustering model with the specified parameters
from sklearn.cluster import KMeans
km=KMeans(init="k-means++",n_init=10,max_iter=100,random_state=0)
#Create a axis and figure with the desired size
fig,ax=plt.subplots(figsize=(12,5))
#Instantiate the KElbowVisualizer with the model and range of k values, and disable the timing plot
from yellowbrick.cluster import KElbowVisualizer,SilhouetteVisualizer
visualizer=KElbowVisualizer(km,k=(2,15),timings=False,ax=ax)
#Fit the data to the visualizer
visualizer.fit(customer_data_pca)
#Finalize the render the figure
visualizer.show()
```



Optimal k Value: Elbow Method Insights

The optimal value of k for the KMeans clustering algorithm can be found at the elbow point. Using the YellowBrick library for the Elbow method, we observe that the suggested optimal k value is 5. However, we don't have a very distinct elbow point in this case, which is common in real-world data. From the plot, we can see that the inertia continues to decrease significantly up to k=5, indicating that the optimum value of k could be between 3 and 7. To choose the best k within this range, we can employ the silhouette analysis, another cluster quality evaluation method. Additionally, incorporating business insights can help determine a practical k value.

Silhouette Method

What is the Silhouette Method?

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.

What is the Silhouette Score?

The **silhouette score** is the **average silhouette coefficient** calculated for all data points in a dataset. It provides an overall assessment of the clustering quality, taking into account both cohesion within clusters and separation between clusters. A higher silhouette score indicates a better clustering configuration.

What are the Advantages of Silhouette Method over the Elbow Method?

- The **Silhouette Method** evaluates cluster quality by considering **both** the **cohesion within clusters** and their **separation** from other clusters. This provides a more comprehensive measure of clustering performance compared to the **Elbow Method**, which only considers the **inertia** (sum of squared distances within clusters).
- The **Silhouette Method** produces a silhouette score that directly quantifies the quality of clustering, making it easier to compare different values of k. In contrast, the **Elbow Method** relies on the subjective interpretation of the elbow point, which can be less reliable in cases where the plot does not show a clear elbow.
- The **Silhouette Method** generates a visual representation of silhouette coefficients for each data point, allowing for easier identification of fluctuations and outliers within clusters. This helps in determining the optimal number of clusters with higher confidence, as opposed to the **Elbow Method**, which relies on visual inspection of the inertia plot.

Methodology

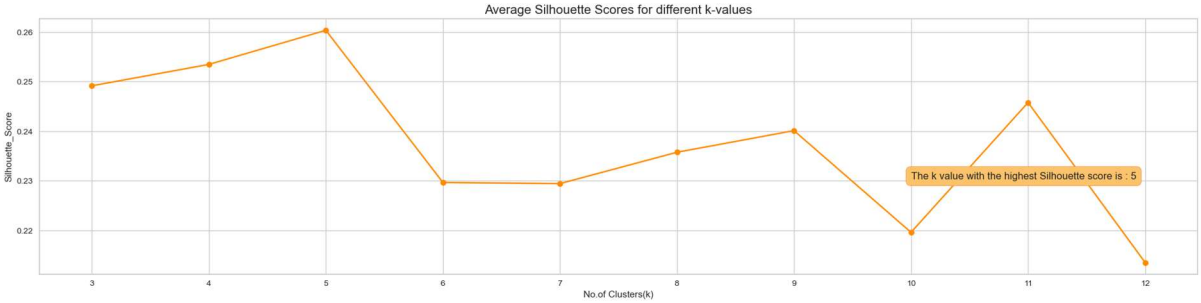
In the following analysis:

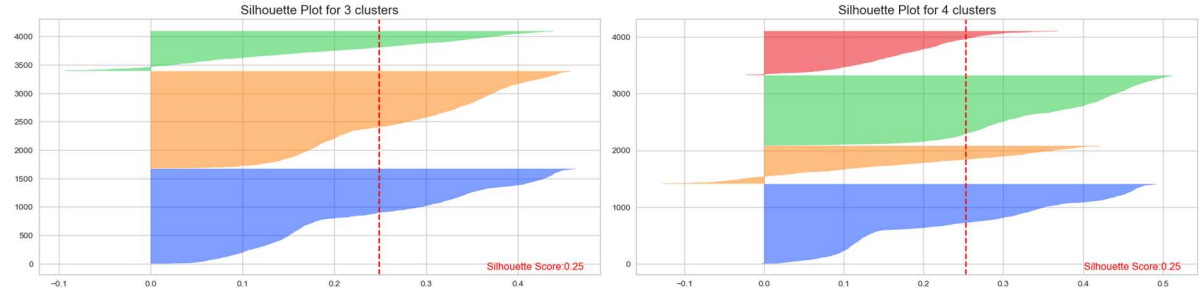
- I will initially choose a range of 2-6 for the number of clusters (k) based on the Elbow method from the previous section. Next, I will plot Silhouette scores for each k value to determine the one with the highest score.

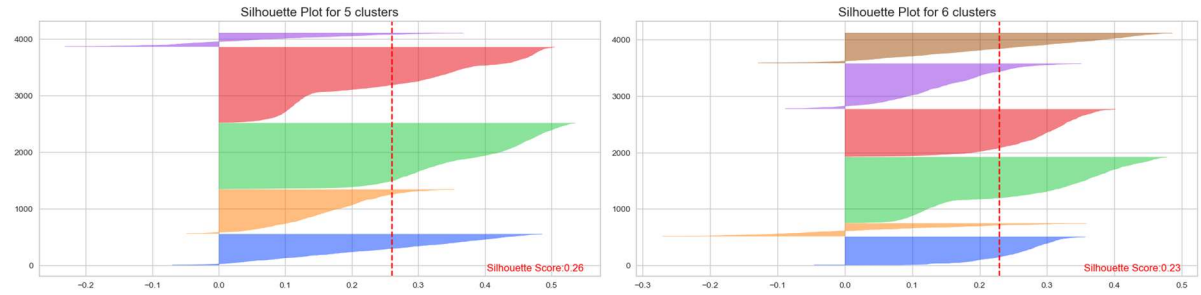
- Subsequently, to fine-tune the selection of the most appropriate k, I will generate Silhouette plots that visually display the silhouette coefficients for each data point within various clusters.

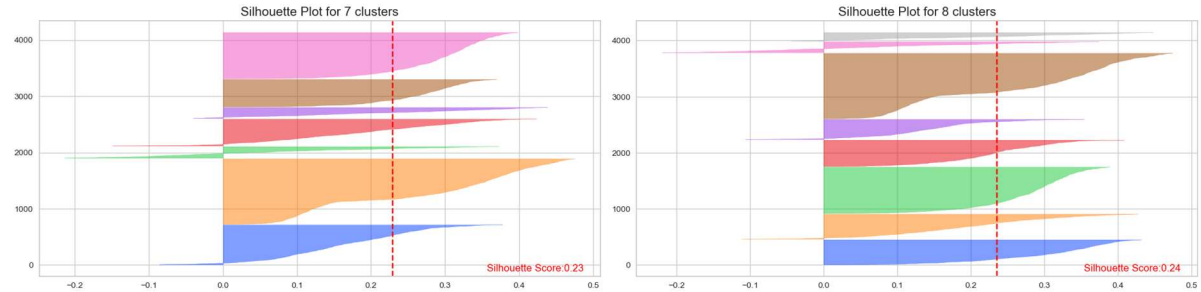
The YellowBrick library will be utilized once again to create these plots and facilitate a comparative analysis.

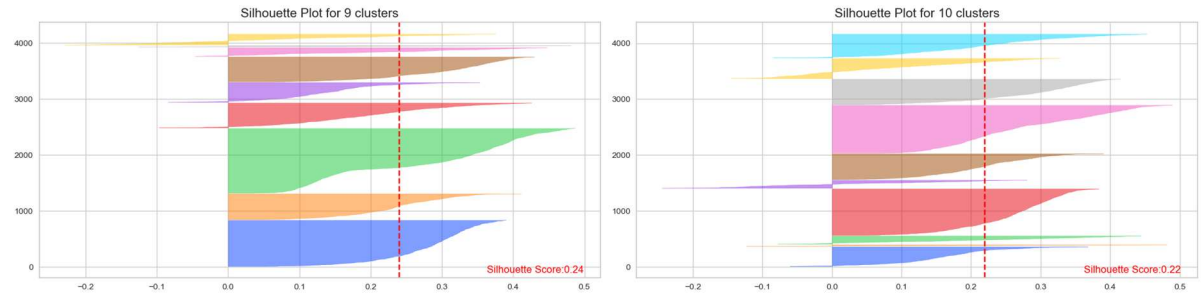
```
#Silhouette Method
def silhouette_analysis(df,start_k,stop_k,figsize=(15,16)):
    """Perfoem Silhoutte analysis for a range of k values and visualize the results"""
    #Set the size of the figure
    plt.figure(figsize=figsize)
    #Create a grid with (stop_k-start_k+1) rows and 2 columns
    import matplotlib.gridspec as gridspec
    grid=gridspec.GridSpec(stop_k-start_k+1,2)
    #Assign the first plot to the first row and both columns
    first_plot=plt.subplot(grid[0,:])
    #First plot:Silhouette scores for different k values
    sns.set_palette(["darkorange"])
    silhouette_scores=[]
    #Iterate throught 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)
        from sklearn.metrics import silhouette_score
        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("No.of Clusters(k)")
    plt.ylabel("Silhouette_Score")
    plt.title("Average Silhouette Scores for different k-values",fontsize=15)
    #Add the optional k value text to the plot
    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"))
    #Second plot (subplot):Silhouette plots for each k value
    colors=sns.color_palette("bright")
    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)
        #Assign the plots to the second,third, and fourth rows
        ax=plt.subplot(grid[row_idx+1,col_idx])
        visualizer=SilhouetteVisualizer(km,colors=colors,ax=ax)
        visualizer.fit(df)
        #Add the silhouette score text to the plot
        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_pca,3,12,figsize=(20,50))
```

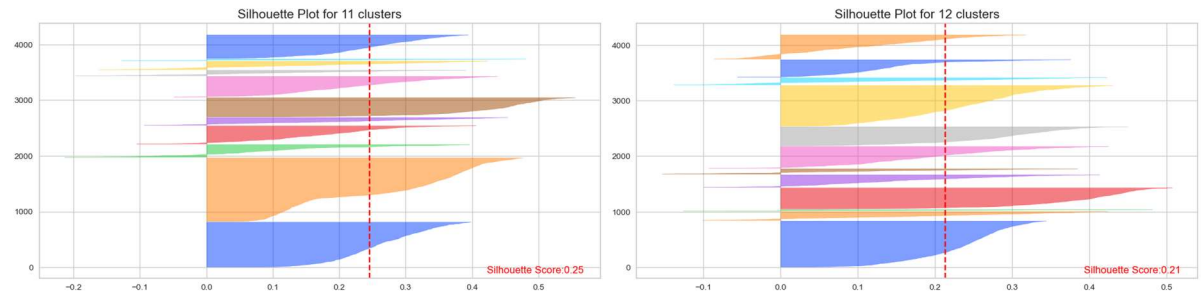

Average Silhouette Scores for different k-values
Silhouette_Score
No.of Clusters(k)
The k value with the highest Silhouette score is : 5


Silhouette Plot for 3 clusters
Silhouette Score:0.25
Silhouette Plot for 4 clusters
Silhouette Score:0.25


Silhouette Plot for 5 clusters
Silhouette Score:0.26
Silhouette Plot for 6 clusters
Silhouette Score:0.23


Silhouette Plot for 7 clusters
Silhouette Score:0.23
Silhouette Plot for 8 clusters
Silhouette Score:0.24


Silhouette Plot for 9 clusters
Silhouette Score:0.24
Silhouette Plot for 10 clusters
Silhouette Score:0.22


Silhouette Plot for 11 clusters
Silhouette Score:0.25
Silhouette Plot for 12 clusters
Silhouette Score:0.21

Optimal k Value: Silhouette Method Insights

Based on above guidelines and after carefully considering the silhouette plots, it's clear that choosing **(k = 3)** is the better option. This choice gives us clusters that are more evenly matched and well-defined, making our clustering solution stronger and more reliable.

- **Clustering Model K-Means**

In this step, I am going to apply the K-means clustering algorithm to segment customers into different clusters based on their purchasing behaviours and other characteristics, using the optimal number of clusters determined in the previous step.

It's important to note that the K-means algorithm might assign different labels to the clusters in each run. To address this, we have taken an additional step to swap the labels based on the frequency of samples in each cluster, ensuring a consistent label assignment across different runs.

```
#Clustering-model-K-Means
#Apply K-Means clustering with the optimal k
kmeans=KMeans(n_clusters=3,init="k-means++",n_init=10,max_iter=100,random_state=0)
kmeans.fit(customer_data_pca)
#Get the frequency of each cluster
from collections import Counter
cluster_frequencies=Counter(kmeans.labels_)
#Create a mapping from old labels to new labels based on frequency
label_mapping={label:new_label for new_label,(label,_) in enumerate(cluster_frequencies.most_common())}
#Reverse the mapping to assign labels as per your criteria
label_mapping={v:k for k,v in {2:1,1:0,0:2}.items()}
#Apply the mapping to get the new labels
new_labels=np.array([label_mapping[label] for label in kmeans.labels_])
#Append the new cluster labels back to the original dataset
customer_data_cleaned["clusters"]=new_labels
#Append the new cluster labels to the PCA version of the dataset
customer_data_pca["clusters"]=new_labels
customer_data_cleaned
```

10.CLUSTERING EVALUATION

After determining the optimal number of clusters (which is 3 in our case) using elbow and silhouette analyses, I move onto the evaluation step to assess the quality of the clusters formed. This step is essential to validate the effectiveness of the clustering and to ensure that the clusters are coherent and well-separated. The evaluation metrics and a visualization technique I plan to use are outlined below:

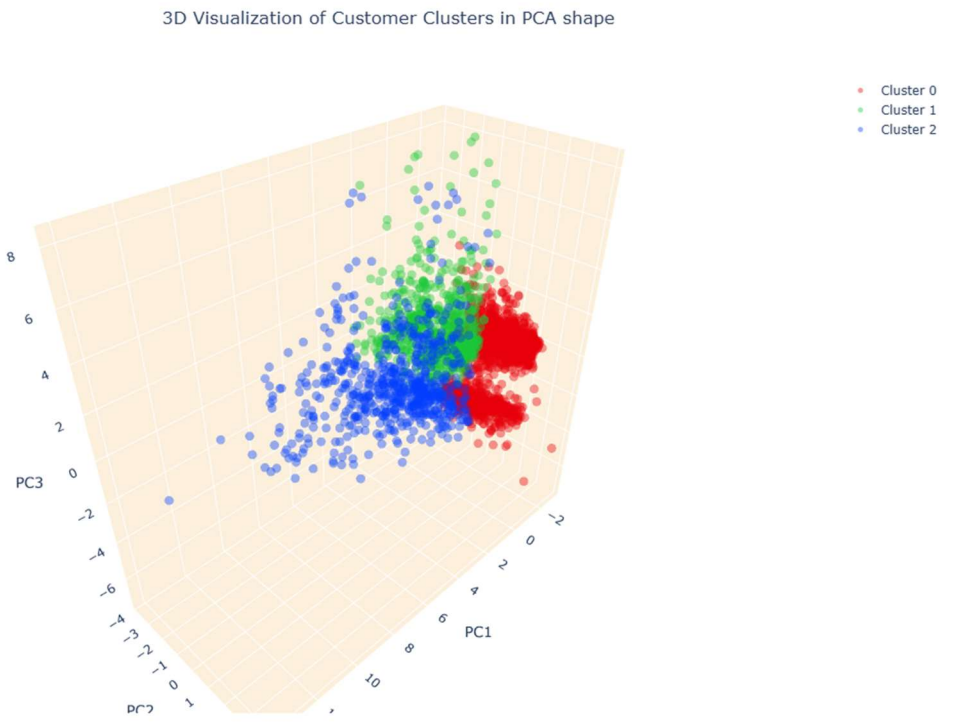
- 3D Visualization of Top PCs
- Cluster Distribution Visualization
- Evaluation Metrics
    - Silhouette Score
    - Calinski Harabasz Score
    - Davies Bouldin Score

- **3D Visualisation of top Principal Components**

  In this part, I am going to choose the top 3 PCs (which capture the most variance in the data) and use them to create a 3D visualization. This will allow us to visually inspect the quality of separation and cohesion of clusters to some extent

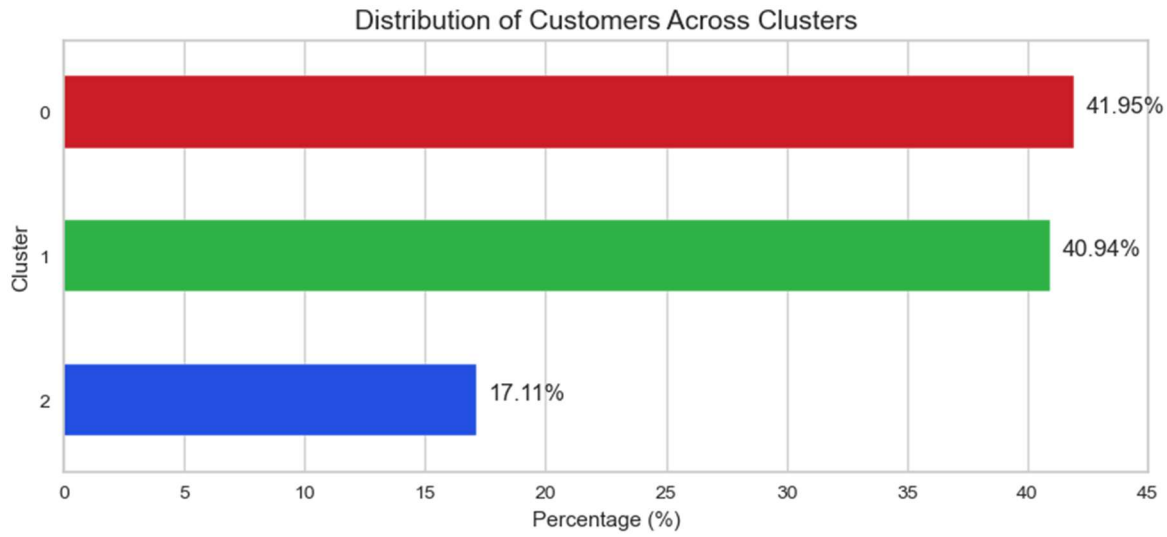
```
#3D Visualization of top Principal components
# Setting up the color scheme for the clusters (RGB order)
colors = ['#e8000b', '#1ac938', '#023eff']
```

```
#Create a seperate dataframe for each cluster
cluster_0=customer_data_pca[customer_data_pca["clusters"]==0]
cluster_1=customer_data_pca[customer_data_pca["clusters"]==1]
cluster_2=customer_data_pca[customer_data_pca["clusters"]==2]
#Create a 3D scatter plot
import plotly.graph_objects as go
fig=go.Figure()
#Add data points for each cluster seperately and specify the color
fig.add_trace(go.Scatter3d(x=cluster_0["PC1"],y=cluster_0["PC2"],z=cluster_0["PC3"],mode="markers",
                           marker=dict(color=colors[0],size=5,opacity=0.4),name="Cluster 0"))
fig.add_trace(go.Scatter3d(x=cluster_1["PC1"],y=cluster_1["PC2"],z=cluster_2["PC3"],mode="markers",
                           marker=dict(color=colors[1],size=5,opacity=0.4),name="Cluster 1"))
fig.add_trace(go.Scatter3d(x=cluster_2["PC1"],y=cluster_2["PC2"],z=cluster_2["PC3"],mode="markers",
                           marker=dict(color=colors[2],size=5,opacity=0.4),name="Cluster 2"))
#Set the title and the layout details
fig.update_layout(title=dict(text="3D Visualization of Customer Clusters in PCA shape",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"))
                            ,height=800,width=900)
fig.show()
```



- **Cluster Distribution Visualization**

```
#Calculate the percentage of each customers in each cluster
cluster_percentage=(customer_data_pca["clusters"].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",palette=colors,width=0.5)
#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 (%)')

plt.show()
```



Inference

The distribution of customers across the clusters, as depicted by the bar plot, suggests a fairly balanced distribution with clusters 0 and 1 holding around 41% of customers each and cluster 2 accommodating approximately 18% of the customers.

This balanced distribution indicates that our clustering process has been largely successful in identifying meaningful patterns within the data, rather than merely grouping noise or outliers.

It implies that each cluster represents a substantial and distinct segment of the customer base, thereby offering valuable insights for future business strategies.

Moreover, the fact that no cluster contains a very small percentage of customers, assures us that each cluster is significant and not just representing outliers or noise in the data. This setup allows for a more nuanced understanding and analysis of different customer segments, facilitating effective and informed decision-making.

- **Evaluation Metrics**

  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.

```python
#Evaluation Metrics
#Compute no.of customers
num_observations=len(customer_data_pca)
#Seperate the features and the cluster labels
X=customer_data_pca.drop("clusters",axis=1)
clusters=customer_data_pca["clusters"]
#Compute the metrics
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score
sil_score=silhouette_score(X,clusters)
calinski_score=calinski_harabasz_score(X,clusters)
davies_score=davies_bouldin_score(X,clusters)
#Create a table to display the no.of observations and the metrics
table_data=[["No.of Observations",num_observations],
            ["Silhouette Score",sil_score],
            ["Calinski Score",calinski_score],
            ["Devies Score",davies_score]]
#Print the table
from tabulate import tabulate
print(tabulate(table_data,headers=["Metric","Value"],tablefmt="pretty"))
```

```
+--------------------+--------------------+
|       Metric       |       Value        |
+--------------------+--------------------+
| No.of Observations |        4067        |
|  Silhouette Score  | 0.2241986417781745 |
|   Calinski Score   | 1167.1215467258985 |
|    Devies Score    | 1.458902764459882  |
+--------------------+--------------------+
```

Clustering Quality Inference

- The **Silhouette Score** of approximately 0.236, although not close to 1, still indicates a fair amount of separation between the clusters. It suggests that the clusters are somewhat distinct, but there might be slight overlaps between them. Generally, a score closer to 1 would be ideal, indicating more distinct and well-separated clusters.
- The **Calinski Harabasz Score** is 1257.17, which is considerably high, indicating that the clusters are well-defined. A higher score in this metric generally signals better cluster definitions, thus implying that our clustering has managed to find substantial structure in the data.
- The **Davies Bouldin Score** of 1.37 is a reasonable score, indicating a moderate level of similarity between each cluster and its most similar one. A lower score is generally

better as it indicates less similarity between clusters, and thus, our score here suggests a decent separation between the clusters.

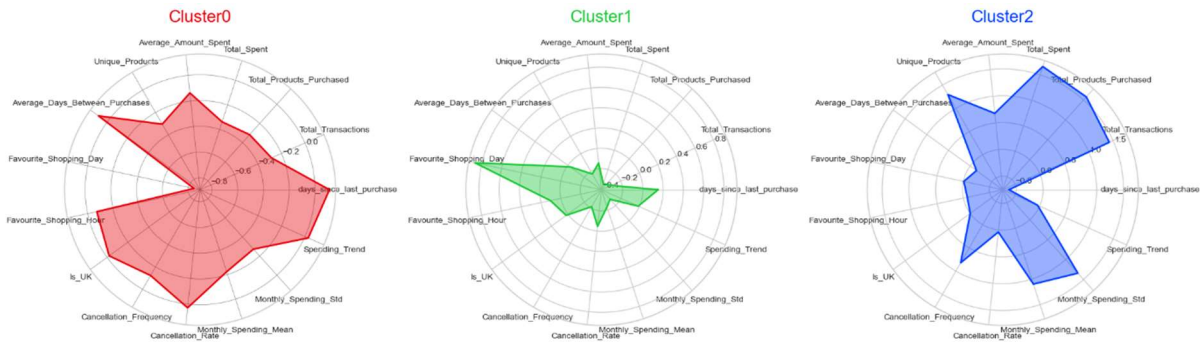
In conclusion, the metrics suggest that the clustering is of good quality, with clusters being well-defined and fairly separated. However, there might still be room for further optimization to enhance cluster separation and definition, potentially by trying other clustering and dimensionality reduction algorithms.

11.CLUSTER ANALYSIS AND PROFILING

- **Radar Chart Approach**

  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:

```
#Cluster analysis and Profiling
#Radar Chart Approach
#Setting "CustomerID" as the index and creating a new dataframe
df_customer=customer_data_cleaned.set_index("CustomerID")
#Standardize the data(excluding the clusters column)
scalar=StandardScaler()
df_customer_standardized=scaler.fit_transform(df_customer.drop(columns=["clusters"],axis=1))
#Create a new dataframe with standardized values and then add the cluster column back
df_customer_standardized = pd.DataFrame(df_customer_standardized, columns=df_customer.columns[:-1], index=df_customer.index)
df_customer_standardized["clusters"]=df_customer["clusters"]
#Calculate the clusters of each cluster
cluster_centroids=df_customer_standardized.groupby("clusters").mean()
#Function to create a radar chart
def create_radar_chart(ax,angles,daa,color,cluster):
    #Plot the data and fill the area
    ax.fill(angles,data,color=color,alpha=0.4)
    ax.plot(angles,data,color=color,linewidth=2,linestyle="solid")
    #Add a title
    ax.set_title(f'Cluster{cluster}',size=20,color=color,y=1.1)

#set data
labels=np.array(cluster_centroids.columns)
num_vars=len(labels)
#Compute angle of each axis
angles=np.linspace(0,2*np.pi, num_vars, endpoint=False).tolist()
# The plot is circular, so we need to "complete the loop" and append the start to the end
labels = np.concatenate((labels, [labels[0]]))
angles += angles[:1]

# Initialize the figure
fig, ax = plt.subplots(figsize=(20, 10), subplot_kw=dict(polar=True), nrows=1, ncols=3)
# Create radar chart for each cluster
for i, color in enumerate(colors):
    data = cluster_centroids.loc[i].tolist()
    data += data[:1]  # Complete the loop
    create_radar_chart(ax[i], angles, data, color, i)

# Add input data
ax[0].set_xticks(angles[:-1])
ax[0].set_xticklabels(labels[:-1])

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

ax[2].set_xticks(angles[:-1])
ax[2].set_xticklabels(labels[:-1])

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

# Display the plot
plt.tight_layout()
plt.show()
```



Customer Profiles Derived from Radar Chart Analysis

Cluster 0 (Red Chart):

Profile: **Sporadic Shoppers with a Preference for Weekend Shopping**

- Customers in this cluster tend to spend less, with a lower number of transactions and products purchased.
- They have a slight tendency to shop during the weekends, as indicated by the very high Day_of_Week value.
- Their spending trend is relatively stable but on the lower side, and they have a low monthly spending variation (low Monthly_Spending_Std).
- These customers have not engaged in many cancellations, showing a low cancellation frequency and rate.
- The average transaction value is on the lower side, indicating that when they do shop, they tend to spend less per transaction.

Cluster 1 (Green Chart):

Profile: **Infrequent Big Spenders with a High Spending Trend**

- Customers in this cluster show a moderate level of spending, but their transactions are not very frequent, as indicated by the high Days_Since_Last_Purchase and Average_Days_Between_Purchases.
- They have a very high spending trend, indicating that their spending has been increasing over time.

- These customers prefer shopping late in the day, as indicated by the high Hour value, and they mainly reside in the UK.
- They have a tendency to cancel a moderate number of transactions, with a medium cancellation frequency and rate.
- Their average transaction value is relatively high, meaning that when they shop, they tend to make substantial purchases.

Cluster 2 (Blue Chart):

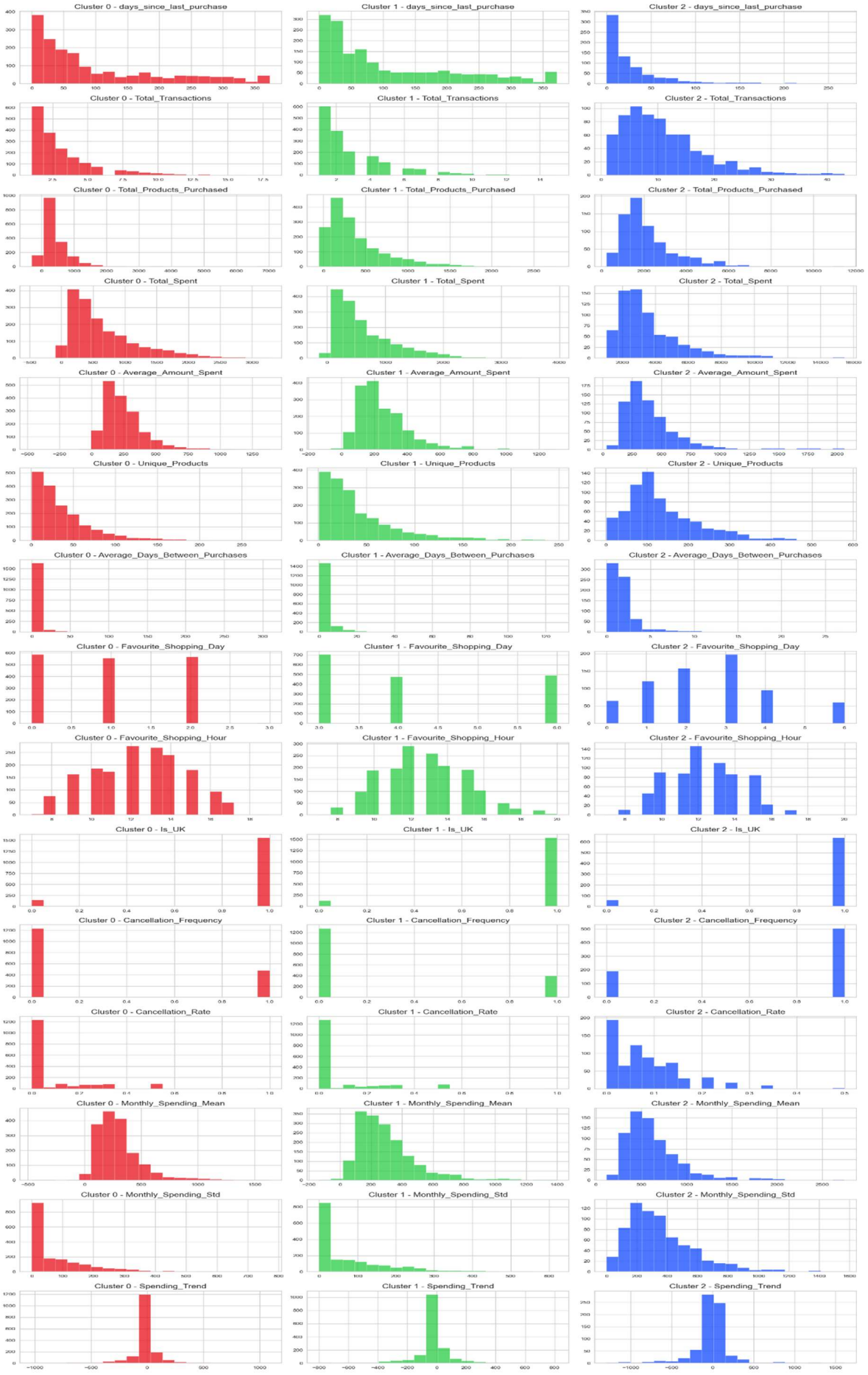
Profile: **Frequent High-Spenders with a High Rate of Cancellations**

- Customers in this cluster are high spenders with a very high total spend, and they purchase a wide variety of unique products.
- They engage in frequent transactions, but also have a high cancellation frequency and rate.
- These customers have a very low average time between purchases, and they tend to shop early in the day (low Hour value).
- Their monthly spending shows high variability, indicating that their spending patterns might be less predictable compared to other clusters.
- Despite their high spending, they show a low spending trend, suggesting that their high spending levels might be decreasing over time.

- **Histogram Chart Approach**

  To validate the profiles identified from the radar charts, we can plot histograms for each feature segmented by the cluster labels. These histograms will allow us to visually inspect the distribution of feature values within each cluster, thereby confirming or refining the profiles we have created based on the radar charts.

```
#Histograph Chart Approach
#Plot histograms for each feature segmented by the clusters
features = customer_data_cleaned.columns[1:-1]
clusters = customer_data_cleaned['clusters'].unique()
clusters.sort()

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

# Plotting histograms
for i, feature in enumerate(features):
    for j, cluster in enumerate(clusters):
        data = customer_data_cleaned[customer_data_cleaned['clusters'] == 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('')
# Adjusting layout to prevent overlapping
plt.tight_layout()
plt.show()
```


Cluster 0 - days_since_last_purchase
Cluster 1 - days_since_last_purchase
Cluster 2 - days_since_last_purchase
Cluster 0 - Total_Transactions
Cluster 1 - Total_Transactions
Cluster 2 - Total_Transactions
Cluster 0 - Total_Products_Purchased
Cluster 1 - Total_Products_Purchased
Cluster 2 - Total_Products_Purchased
Cluster 0 - Total_Spent
Cluster 1 - Total_Spent
Cluster 2 - Total_Spent
Cluster 0 - Average_Amount_Spent
Cluster 1 - Average_Amount_Spent
Cluster 2 - Average_Amount_Spent
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 - Favourite_Shopping_Day
Cluster 1 - Favourite_Shopping_Day
Cluster 2 - Favourite_Shopping_Day
Cluster 0 - Favourite_Shopping_Hour
Cluster 1 - Favourite_Shopping_Hour
Cluster 2 - Favourite_Shopping_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 - Monthly_Spending_Mean
Cluster 1 - Monthly_Spending_Mean
Cluster 2 - Monthly_Spending_Mean
Cluster 0 - Monthly_Spending_Std
Cluster 1 - Monthly_Spending_Std
Cluster 2 - Monthly_Spending_Std
Cluster 0 - Spending_Trend
Cluster 1 - Spending_Trend
Cluster 2 - Spending_Trend

The detailed insights from the histograms provide a more nuanced understanding of each cluster, helping in refining the profiles to represent the customer behaviours more accurately.

12.RECOMMENDATION SYSTEM

In the final phase of this project, I am set to develop a recommendation system to enhance the online shopping experience. This system will suggest products to customers based on the purchasing patterns prevalent in their respective clusters. Earlier in the project, during the customer data preparation stage, I isolated a small fraction (5%) of the customers identified as outliers and reserved them in a separate dataset called outliers_data.

Now, focusing on the core 95% of the customer group, I analyze the cleansed customer data to pinpoint the top-selling products within each cluster. Leveraging this information, the system will craft personalized recommendations, suggesting the top three products popular within their cluster that they have not yet purchased. This not only facilitates targeted marketing strategies but also enriches the personal shopping experience, potentially boosting sales. For the outlier group, a basic approach could be to recommend random products, as a starting point to engage them.

```
#Recommendation System
#Step 1 : Extract the customer ids of the outliers and remove their transactions from the main dataframe
outlier_customer_ids=outliers_data["CustomerID"].astype("float").unique()
df_filtered=df[~df["CustomerID"].isin(outlier_customer_ids)]
#Step 2 : Ensure Consistent dataframe for CustomerID across both dataframes before merging
customer_data_cleaned["CustomerID"]=customer_data_cleaned["CustomerID"].astype("float")
#Step 3 : Merge the transaction data with the customer data to get the cluster information for each transaction
merged_data=df_filtered.merge(customer_data_cleaned[["CustomerID","clusters"]],on="CustomerID",how="inner")
#Step 4 : Identify the top 10 best-selling products in each cluster based on the total quantity sold
best_selling_products=merged_data.groupby(["clusters","StockCode","Description"])["Quantity"].sum().reset_index()
best_selling_products=best_selling_products.sort_values(by=["clusters","Quantity"],ascending=[True,False])
top_products_per_cluster=best_selling_products.groupby("clusters").head(10)
#Step 5 : Create a record of products purchased by each customer in each cluster
customer_purchases=merged_data.groupby(["CustomerID","clusters","StockCode"])["Quantity"].sum().reset_index()
#Step 6 : Generate recommendations for each customer in each cluster
recommendations=[]
for cluster in top_products_per_cluster["clusters"].unique():
    top_products=top_products_per_cluster[top_products_per_cluster["clusters"]==cluster]
    customer_in_cluster=customer_data_cleaned[customer_data_cleaned["clusters"]==cluster]["CustomerID"]
    for customer in customer_in_cluster:
        #Identify products already purchased by the customer
        customer_purchased_products=customer_purchases[(customer_purchases["CustomerID"]==customer) &
        (customer_purchases["clusters"]==cluster)]["StockCode"].tolist()
        #Find top 3 products in the best selling list that the customer hasn't purchased yet
        top_products_not_purchased=top_products[~top_products["StockCode"].isin(customer_purchased_products)]
        top_3_products_not_purchased=top_products_not_purchased.head(3)
        #Append the recommendations to the list
        recommendations.append([customer,cluster]+top_3_products_not_purchased[["StockCode","Description"]]
                               .values.flatten().tolist())
#Step 7 : Create a dataframe from the recommendations list and merge it with the original customer data
recommendations_df=pd.DataFrame(recommendations,columns=["CustomerID","clusters","Rec1_StockCode",
                                                         "Rec1_Description","Rec2_StockCode","Rec2_Description",
                                                         "Rec3_StockCode","Rec3_Description"])
customer_data_with_recommendations=customer_data_cleaned.merge(recommendations_df,on=
                                                               ["CustomerID","clusters"],how="right")
customer_data_with_recommendations
```

```
# Display 10 random rows from the customer_data_with_recommendations dataframe
customer_data_with_recommendations.set_index('CustomerID').iloc[:, -6:].sample(10, random_state=0)
```

CustomerID	Rec1_StockCode	Rec1_Description	Rec2_StockCode	Rec2_Description	Rec3_StockCode	Rec3_Description
15841.0	84879	ASSORTED COLOUR BIRD ORNAMENT	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	17003	BROCADE RING PURSE
15831.0	84879	ASSORTED COLOUR BIRD ORNAMENT	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	85123A	WHITE HANGING HEART T-LIGHT HOLDER
17242.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	ASSORTED COLOURS SILK FAN
17198.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	ASSORTED COLOURS SILK FAN
15019.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	ASSORTED COLOURS SILK FAN
14368.0	84879	ASSORTED COLOUR BIRD ORNAMENT	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	85123A	WHITE HANGING HEART T-LIGHT HOLDER
15179.0	22616	PACK OF 12 LONDON TISSUES	84879	ASSORTED COLOUR BIRD ORNAMENT	22178	VICTORIAN GLASS HANGING T-LIGHT
17513.0	84879	ASSORTED COLOUR BIRD ORNAMENT	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	85123A	WHITE HANGING HEART T-LIGHT HOLDER
17861.0	85099B	JUMBO BAG RED RETROSPOT	84879	ASSORTED COLOUR BIRD ORNAMENT	22178	VICTORIAN GLASS HANGING T-LIGHT

RESULTS

Based on the detailed analysis from both the radar charts and the histograms from K-Means Clustering, here are the refined profiles and titles for each cluster:

Cluster 0 - Casual Weekend Shoppers:

1. Customers in this cluster usually shop less frequently and spend less money compared to the other clusters.
2. They generally have a smaller number of transactions and buy fewer products.
3. These customers have a preference for shopping during the weekends, possibly engaging in casual or window shopping.
4. Their spending habits are quite stable over time, showing little fluctuation in their monthly spending.
5. They rarely cancel their transactions, indicating a more decisive shopping behaviour.
6. When they do shop, their spending per transaction tends to be lower compared to other clusters.

Cluster 1 - Occasional Big Spenders:

1. Customers in this cluster don't shop frequently but tend to spend a considerable amount when they do, buying a variety of products.
2. Their spending has been on the rise, indicating a growing interest or investment in their purchases.

3. They prefer to shop later in the day, possibly after work hours, and are mainly based in the UK.
4. They have a moderate tendency to cancel transactions, which might be due to their higher spending; they perhaps reconsider their purchases more often.
5. Their purchases are generally substantial, indicating a preference for quality or premium products.

Cluster 2 - Eager Early-Bird Shoppers:

1. Customers in this cluster are characterized by their high spending habits. They tend to buy a wide array of unique products and engage in numerous transactions.
2. Despite their high expenditure, they have a tendency to cancel a significant portion of their transactions, possibly indicating impulsive buying behaviours.
3. They usually shop during the early hours of the day, perhaps finding time before their daily commitments or taking advantage of early bird deals.
4. Their spending patterns are quite variable, with high fluctuations in their monthly spending, indicating a less predictable shopping pattern.
5. Interestingly, their spending trend is showing a slight decrease, which might signal a future change in their shopping habits.

CONCLUSIONS

In this project, customer segmentation and recommendation systems were implemented using a combination of data preprocessing, exploratory data analysis (EDA), and machine learning techniques. The data preprocessing involved cleaning and transforming the raw customer data to make it suitable for analysis. The EDA provided insights into customer behaviour and characteristics, which were crucial for effective segmentation.

For customer segmentation, the K-means clustering algorithm was employed to group customers into distinct segments based on their purchasing patterns and other relevant features. This segmentation helped in understanding different customer groups and tailoring marketing strategies to meet their specific needs. Each cluster represented a unique customer segment with similar attributes and behaviours.

In the recommendation system component, collaborative filtering techniques were utilized to build both item-based and user-based recommendation models. These models leveraged historical customer purchase data to recommend products that customers were likely to be interested in. The item-based approach focused on recommending items similar to those a user had interacted with, while the user-based approach recommended items favoured by similar users. The combined use of segmentation and recommendation provided a comprehensive solution for personalized marketing and improved customer satisfaction.

FUTURE SCOPE

1. **Advanced Machine Learning Models:** Incorporate more sophisticated algorithms like neural networks or gradient boosting to enhance the accuracy and effectiveness of both segmentation and recommendation systems. This can provide deeper insights and more personalized recommendations.
2. **Real-time Data Processing**: Implement real-time data collection and processing to keep the customer segments and recommendations up-to-date. This will allow for immediate responses to changing customer behaviours and trends, improving relevance and engagement.
3. **Integration with Multiple Data Sources:** Expand the project to integrate data from various sources, such as social media, web analytics, and customer feedback. This holistic view can enhance the understanding of customer preferences and improve segmentation and recommendation precision.
4. **Personalized Marketing Campaigns:** Use the insights from customer segmentation and recommendations to develop highly targeted marketing campaigns. By leveraging personalized communication strategies, businesses can improve customer retention, satisfaction, and overall sales performance.

REFERENCES

1. **Machine Learning (Scikit Learn) Guide :**

   https://scikit-learn.org/stable/user_guide.html

2. **Python Programming User Guide:**

   https://www.python.org/doc/

3. **Dataset Source**: Access the original dataset from the UCI Machine Learning Repository.

4. **Prof. Nikhil Patankar,** et al., "Customer Segmentation Using Machine Learning" Sanjivani College of Engineering,2023.

5. **Miguel Alves Gomes, Tobias Meisen,** "A Review on Customer Segmentation Methods for Personalized Customer Targeting in E-Commerce Use Cases," Springer,2023.

6. **Malay Sarkar et.al,** "Optimizing E-Commerce Profits: A Comprehensive Machine Learning Framework for Dynamic Pricing and Predicting Online Purchases," 2023.