CUSTOMER SEGMENTATION AND RECOMMENDATION

A Project Report

Submitted in partial fulfilment of the

requirements for the award of the Degree of

BACHELOR OF SCIENCE (DATA SCIENCE)

By

Mr. Sahil Samrat Sakat

Seat Number:

Under the esteemed guidance of

Dr. Ujwala Madhav Sav

Assistant Professor, Department of Information Technology

And Data Science

DEPARTMENT OF INFORMATION TECHNOLOGY AND DATA SCIENCE

VIDYALANKAR SCHOOL OF INFORMATION TECHNOLOGY

(Affiliated to University of Mumbai)

MUMBAI, 400 037

MAHARASHTRA

2023 – 2024

VIDYALANKAR SCHOOL OF INFORMATION TECHNOLOGY

(Affiliated to University of Mumbai)

MUMBAI-MAHARASHTRA-400037

DEPARTMENT OF INFORMATION TECHNOLOGY

AND DATA SCIENCE

VSIT | Vidyalankar School of Information Technology

CERTIFICATE

This is to certify that the project entitled, **"Customer Segmentation and Recommendation"**, is bonafied work of **Mr. Sahil Samrat Sakat** bearing Seat No: submitted in partial fulfilment of the requirements for the award of degree of BACHELOR OF SCIENCE in DATA SCIENCE from University of Mumbai.

Internal Guide **Coordinator**

Internal Examiner **External Examiner**

Date: **College Seal** **Principal**

ABSTRACT

In today's highly competitive market, understanding customer behavior and preferences is crucial for businesses to maintain a competitive edge. This project focuses on leveraging advanced data analytics techniques to perform customer segmentation and develop a recommendation system aimed at enhancing customer satisfaction and loyalty.

The first phase of the project involves data collection and preprocessing, where raw customer data from various sources, such as transaction histories, demographics, and interaction logs, is cleaned and organized. Following this, exploratory data analysis (EDA) is conducted to uncover underlying patterns and insights.

Customer segmentation is achieved through the application of clustering algorithms, such as K-means, hierarchical clustering, and DBSCAN. These techniques group customers into distinct segments based on their purchasing behavior, preferences, and other relevant attributes. The optimal number of clusters is determined using methods like the Elbow method and Silhouette analysis, ensuring the segments are meaningful and actionable.

Once the segmentation is complete, a recommendation system is developed using collaborative filtering and content-based filtering approaches. Collaborative filtering leverages historical interaction data to identify similarities between customers and suggest products that similar customers have liked. Content-based filtering, on the other hand, uses product attributes and customer profiles to recommend items that match a customer's past preferences.

This project demonstrates the power of data-driven strategies in personalizing customer experiences, driving sales, and fostering customer loyalty. By segmenting customers accurately and providing tailored recommendations, businesses can better meet customer needs, enhance satisfaction, and ultimately achieve higher profitability.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have supported me throughout my Semester project. this project would not have been possible without their help and guidance.

I want to thank Vidyalankar School of Information Technology for giving us the opportunity for doing this project.

First and foremost, I would like to thank my project guide Dr. Ujwala Madhav Sav for her invaluable advice, encouragement, and feedback throughout the project. She has been a great mentor and a source of inspiration for me. Her expertise and knowledge in the field of data visualization and data analysis have helped me to overcome many challenges and improve the quality of my work.

I am grateful to Kaggle dataset providers for allowing me to access and use their data for my project.

We are also thankful for and fortunate enough to get constant encouragement, support, and guidance from the teachers of Data Science who helped us in successfully completing our project work.

This list would be incomplete without mentioning all the developers and education institutes around the world that share their knowledge, work, and wisdom over the Internet.

DECLARATION

I hereby declare that the project entitled, "**Customer Segmentation and Recommendation**" done at Vidyalankar School of Information Technology, has not been in any case duplicated to submit to any other universities for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of **BACHELOR OF SCIENCE (DATA SCIENCE)** to be submitted as final semester project as part of our curriculum.

Name and Signature of the Student

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

In an era where customer-centric strategies are pivotal for business success, understanding and predicting customer behavior has become paramount. This project aims to harness the power of data analytics to perform customer segmentation and develop a robust recommendation system. By analyzing vast amounts of customer data, we can identify distinct customer groups based on their behaviors and preferences. This segmentation allows for targeted marketing strategies and personalized customer experiences. Furthermore, the recommendation system leverages advanced algorithms to suggest products tailored to individual customer preferences, thereby enhancing customer satisfaction and fostering loyalty. Through these data-driven techniques, businesses can achieve a deeper understanding of their customer base, optimize marketing efforts, and drive sustainable growth.

1.1 Background

The background of this project lies in the evolving landscape of customer relationship management and the increasing importance of data-driven decision-making in business operations. With the proliferation of digital platforms and the rise of e-commerce, businesses today collect vast amounts of data from various customer touchpoints, including purchase histories, website interactions, and social media activities. This wealth of data presents an opportunity to gain deep insights into customer behavior and preferences, which can be leveraged to enhance customer experience and drive business growth.

Traditionally, businesses relied on broad, demographic-based marketing strategies that often failed to address the specific needs and preferences of individual customers. However, the advent of machine learning and data mining techniques has revolutionized this approach, enabling more granular and actionable insights through customer segmentation. By grouping customers into distinct segments based on their behavior and characteristics, businesses can design targeted marketing campaigns and personalized experiences that resonate more effectively with each segment.

Moreover, recommendation systems have become a cornerstone of modern digital platforms, from e-commerce sites to streaming services. These systems analyze patterns in customer data to suggest products or content that are likely to interest individual users, thereby enhancing engagement and satisfaction. The integration of collaborative filtering and content-based filtering methods in recommendation systems ensures that recommendations are both relevant and diverse, catering to the nuanced preferences of each customer.

This project builds on these advancements in data analytics to create a comprehensive solution for customer segmentation and recommendation. By leveraging state-of-the-art algorithms and techniques, the project aims to transform raw data into valuable insights and actionable recommendations, ultimately helping businesses to better understand and serve their customers.

1.2 Objectives

The primary objective of this project is to enhance customer satisfaction and loyalty through the implementation of advanced data analytics techniques for customer segmentation and personalized recommendations. By systematically collecting and analyzing customer data, we aim to uncover distinct behavioral patterns and preferences that allow for precise segmentation of the customer base. This segmentation facilitates the development of targeted marketing strategies tailored to the unique needs of each customer group. Additionally, the project seeks to build a recommendation system that utilizes collaborative and content-based filtering algorithms to provide highly relevant product suggestions to individual customers. By achieving these objectives, the project aspires to improve customer engagement, boost sales, and foster long-term customer relationships, ultimately driving business growth and competitiveness in the market.

1.3 Purpose, Scope, Applicability (Feasibility Study)

Purpose:

The purpose of this project is to leverage advanced data analytics techniques to enhance customer relationship management through effective customer segmentation and personalized recommendation systems. By accurately segmenting customers based on their behaviors and preferences, the project aims to enable businesses to develop more targeted and effective marketing strategies. Additionally, the implementation of a sophisticated recommendation system seeks to deliver personalized product suggestions, thereby increasing customer satisfaction and loyalty. Ultimately, this project intends to drive business growth by fostering deeper customer engagement, optimizing marketing efforts, and enhancing the overall customer experience.

Scope:

The scope of this project encompasses the entire process of customer segmentation and recommendation system development, from data collection and preprocessing to model implementation and evaluation. It includes the analysis of various data sources such as transaction histories, demographics, and interaction logs to derive meaningful insights into customer behavior. The project will employ clustering algorithms for customer segmentation, considering factors such as purchasing patterns and preferences. Furthermore, it will develop recommendation systems using collaborative filtering and content-based filtering techniques to provide personalized product suggestions to individual customers. The evaluation of these systems will involve metrics such as precision, recall, and customer engagement metrics. While the project will focus primarily on the technical aspects of segmentation and recommendation system development, it will also consider the practical implications for businesses in terms of marketing strategy optimization and customer relationship management.

Applicability (Feasibility Study)

1. Technical Feasibility:

Conducting a technical feasibility study is essential to assess the viability and practicality of implementing the proposed customer segmentation and recommendation system. This study will focus on evaluating the availability and quality of data required for segmentation and recommendation model training. It will also examine the computational resources and infrastructure needed to process and analyze large volumes of customer data efficiently. Additionally, the feasibility study will explore the compatibility of existing software tools and libraries for implementing clustering algorithms for segmentation and recommendation algorithms for personalized suggestions. By conducting this technical feasibility study, the project aims to identify any potential challenges or limitations in implementing the proposed solution and develop strategies to address them effectively.

2. Economic Feasibility:

The economic feasibility study of this project involves an assessment of the potential costs and benefits associated with implementing the proposed customer segmentation and recommendation system. Initial investment will be required for data collection, infrastructure setup, and hiring skilled personnel proficient in data analytics. However, these costs are expected to be offset by the anticipated benefits, including increased sales revenue resulting from more targeted marketing efforts, reduced customer acquisition costs through improved customer retention, and enhanced operational efficiency from personalized recommendations. By quantifying the projected return on investment and conducting a cost-benefit analysis, this study aims to demonstrate the economic viability of implementing the proposed solution, thereby providing stakeholders with the necessary information to make informed decisions regarding resource allocation and project prioritization.

3. Operational Feasibility:

The operational feasibility study of this project involves assessing the practicality and viability of implementing the proposed customer segmentation and recommendation system within existing business operations. This includes evaluating factors such as the availability and quality of data, the technical infrastructure required for data processing and analysis, and the compatibility of the proposed solution with current business processes. Additionally,

considerations will be made regarding the expertise and resources needed for system development, implementation, and maintenance. By conducting a comprehensive operational feasibility study, the project aims to ensure that the proposed solution can be effectively integrated into the organization's operations, delivering tangible benefits in terms of improved customer engagement and business performance.

4. Applicability and User Acceptance:

The feasibility study for this project will assess the applicability and user acceptance of the proposed customer segmentation and recommendation system within the target business environment. It will analyze factors such as the availability and quality of data, technical infrastructure requirements, and the willingness of stakeholders to adopt data-driven strategies. Additionally, user acceptance testing will gauge the usability and effectiveness of the system, ensuring that it meets the needs and expectations of end-users. By conducting a comprehensive feasibility study, the project aims to identify potential challenges and opportunities, ultimately determining the viability and readiness of implementing the segmentation and recommendation system within the organization.

Chapter 2 Dataset Description

The dataset 'E-commerce' comprises information essential for customer segmentation. It includes Invoice no, stock code, description, quantity, unit price, customer id, etc. The dataset aims to facilitate the development of effective customer segmentation and recommendation.

The columns and their corresponding descriptions are as follows:

Attribute	Type	Description
Invoice No	Numeric	Unique identifier for each transaction
Stock Code	Alpha-Numeric	Unique identifier for each products
Description	Text	Description of products
Quantity	Numeric	Number of units of the product purchased
Invoice Date	Date	Date when the invoice is generated
Unit Price	Float	Price per unit of the product
Customer ID	Numeric	Unique identifier for each customer
Country	Text	Country where the customer is located

Table 1 : Data Description

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850	United Kingdom
536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850	United Kingdom
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850	United Kingdom
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850	United Kingdom
536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850	United Kingdom
536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850	United Kingdom
536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850	United Kingdom
536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	17850	United Kingdom
536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	13047	United Kingdom
536367	22745	POPPY'S PLAYHOUSE BEDROOM	6	12/1/2010 8:34	2.1	13047	United Kingdom
536367	22748	POPPY'S PLAYHOUSE KITCHEN	6	12/1/2010 8:34	2.1	13047	United Kingdom
536367	22749	FELTCRAFT PRINCESS CHARLOTTE DOLL	8	12/1/2010 8:34	3.75	13047	United Kingdom
536367	22310	IVORY KNITTED MUG COSY	6	12/1/2010 8:34	1.65	13047	United Kingdom
536367	84969	BOX OF 6 ASSORTED COLOUR TEASPOONS	6	12/1/2010 8:34	4.25	13047	United Kingdom
536367	22623	BOX OF VINTAGE JIGSAW BLOCKS	3	12/1/2010 8:34	4.95	13047	United Kingdom
536367	22622	BOX OF VINTAGE ALPHABET BLOCKS	2	12/1/2010 8:34	9.95	13047	United Kingdom
536367	21754	HOME BUILDING BLOCK WORD	3	12/1/2010 8:34	5.95	13047	United Kingdom
536367	21755	LOVE BUILDING BLOCK WORD	3	12/1/2010 8:34	5.95	13047	United Kingdom
536367	21777	RECIPE BOX WITH METAL HEART	4	12/1/2010 8:34	7.95	13047	United Kingdom
536367	48187	DOORMAT NEW ENGLAND	4	12/1/2010 8:34	7.95	13047	United Kingdom
536368	22960	JAM MAKING SET WITH JARS	6	12/1/2010 8:34	4.25	13047	United Kingdom
536368	22913	RED COAT RACK PARIS FASHION	3	12/1/2010 8:34	4.95	13047	United Kingdom
536368	22912	YELLOW COAT RACK PARIS FASHION	3	12/1/2010 8:34	4.95	13047	United Kingdom
536368	22914	BLUE COAT RACK PARIS FASHION	3	12/1/2010 8:34	4.95	13047	United Kingdom
536369	21756	BATH BUILDING BLOCK WORD	3	12/1/2010 8:35	5.95	13047	United Kingdom
536370	22728	ALARM CLOCK BAKELIKE PINK	24	12/1/2010 8:45	3.75	12583	France
536370	22727	ALARM CLOCK BAKELIKE RED	24	12/1/2010 8:45	3.75	12583	France
536370	22726	ALARM CLOCK BAKELIKE GREEN	12	12/1/2010 8:45	3.75	12583	France

Figure 1: Dataset

The UCI Machine Learning Repository has made this dataset containing actual transactions from 2010 and 2011. The dataset is maintained on their site, where it can be found by the title "Online Retail".

"This is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail.The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers."

Chapter 3 Methods and Algorithms

Customer segmentation is a crucial process in understanding and targeting distinct customer groups, enabling businesses to develop personalized marketing strategies and enhance customer satisfaction. This comprehensive approach involves several key stages: data cleaning and transformation to handle missing values, duplicates, and outliers; feature engineering to create meaningful metrics based on transactional data; data preprocessing for feature scaling and dimensionality reduction; and customer segmentation using K-means clustering. Through cluster analysis and evaluation, each segment's unique characteristics are identified, allowing for targeted marketing efforts. Furthermore, implementing a recommendation system based on clustering results can suggest best-selling products to similar customers, driving sales and improving marketing effectiveness. This methodology provides a robust framework for leveraging customer data to inform strategic business decisions.

1. Data Cleaning & Transformation:

We start by cleaning the dataset to handle missing values, duplicates, and outliers. Missing values can be addressed using imputation methods such as mean, median, or mode imputation for numerical data and most frequent value imputation for categorical data. For more complex cases, advanced techniques like K-Nearest Neighbors (KNN) imputation can be employed. Duplicates are identified and removed based on unique transaction identifiers such as `Invoice No` or `Customer ID`. Outliers are detected using statistical methods (e.g., z-score, IQR) or visualization tools (e.g., box plots) and handled by either removing them or using transformation techniques to reduce their impact.

2. Feature Engineering

In this stage, we develop new features to capture meaningful patterns in the data. RFM (Recency, Frequency, Monetary) Analysis is a common approach, where:

- Recency measures the time since the last purchase,
- Frequency counts the number of transactions within a period,
- Monetary assesses the total spending of the customer.

Additionally, features like average transaction value, product diversity (number of unique products purchased), and purchase intervals can be calculated to provide deeper insights into customer behavior.

3. Data Preprocessing

To prepare the data for clustering, we undertake feature scaling and dimensionality reduction. Feature scaling is performed using standardization (z-score normalization) or normalization (min-max scaling) to ensure that all features contribute equally to the clustering process. Dimensionality reduction techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) are used to reduce the dataset's dimensionality while preserving essential information, enhancing computational efficiency and improving clustering performance.

4. Customer Segmentation using K-Means Clustering

We use the K-Means Clustering algorithm to segment customers into distinct groups. The Elbow Method helps determine the optimal number of clusters by plotting the Within-Cluster Sum of Squares (WCSS) against the number of clusters. K-means is then applied with the chosen number of clusters, grouping customers based on their purchasing behavior and derived features.

5. Cluster Analysis & Evaluation

Each cluster is analyzed and profiled to understand its characteristics and develop targeted marketing strategies. Descriptive statistics, visualization tools (e.g., bar charts, scatter plots), and Silhouette Score are used to evaluate cluster quality and distinctness. This analysis reveals insights into customer segments, such as high-value customers, frequent buyers, or cost-sensitive shoppers.

6. Recommendation

To boost sales and marketing effectiveness, a recommendation system is implemented. Collaborative Filtering techniques, like User-Based or Item-Based Collaborative Filtering, can

be used to recommend best-selling products to customers within the same cluster who haven't purchased those products. Alternatively, Content-Based Filtering can suggest products similar to those previously purchased by the customer. This system leverages the clustering results to provide personalized product recommendations, enhancing customer satisfaction and increasing sales.

By following these methods and utilizing the specified algorithms, we can effectively segment customers, analyze their behavior, and implement a recommendation system to drive targeted marketing and personalized strategies.

Chapter 4 Project Analysis

1. Setup and Initialization

```
import warnings
warnings.filterwarnings('ignore')

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

%matplotlib inline
```

```
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

Figure 2:Importing Libraries

Importing Libraries

1. warnings: The `warnings` module is used to control the display of warning messages in Python. In your code, you've used it to ignore warnings, which can be helpful when running code that may generate numerous warnings that you want to suppress.

2. numpy (np): NumPy is a fundamental package for scientific computing with Python. It provides support for arrays, matrices, and mathematical functions. In your code, you likely use NumPy for numerical operations and array manipulations.

3. pandas (pd): Pandas is a powerful library for data manipulation and analysis. It provides data structures like DataFrame and Series, which are essential for handling structured data. In your code, you probably use pandas to read, manipulate, and analyze datasets.

4. seaborn (sns): Seaborn is a statistical data visualization library built on top of matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. In your code, you might use seaborn for creating visually appealing plots for data exploration and visualization.

5. matplotlib.pyplot (plt): Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. The `pyplot` module provides a MATLAB-like interface for plotting. In your code, you use it alongside seaborn for creating various types of plots, such as histograms, scatter plots, and bar charts.

6. matplotlib.gridspec: Matplotlib's `gridspec` module allows for more advanced subplot layout customization by specifying the geometry of the grid in which subplots will be placed. It's useful when you need more control over the arrangement of multiple subplots in a figure.

7. plotly.graph_objects (go): Plotly is an interactive, open-source plotting library that provides a high-level interface for creating and sharing interactive plots. The `graph_objects` module offers a low-level interface for creating various types of plots, such as scatter plots, line plots, and bar charts. In your code, you likely use Plotly for generating interactive visualizations for web-based applications or presentations.

8. matplotlib.colors.LinearSegmentedColormap: This module from matplotlib allows you to create custom colormaps for visualizing data. Colormaps are used to map scalar data to colors in a plot, and creating custom colormaps can be useful for emphasizing certain aspects of your data.

9. matplotlib.colors (mcolors): This module provides a collection of utility functions and classes for working with colors in matplotlib. It includes functions for converting between color representations, defining color maps, and manipulating color values.

10. scipy.stats.linregress: This function from SciPy's `stats` module computes a linear least-squares regression for two sets of measurements. It returns slope, intercept, correlation coefficient, p-value, and standard error of the estimated gradient.

11. sklearn.ensemble.IsolationForest: Isolation Forest is an anomaly detection algorithm implemented in scikit-learn. It works by isolating anomalies in data partitions, making it particularly effective for detecting outliers in high-dimensional datasets.

12. sklearn.preprocessing.StandardScaler: This class from scikit-learn is used for standardizing features by removing the mean and scaling to unit variance. Standardization is an essential preprocessing step in many machine learning algorithms to ensure that features are on a similar scale.

13. sklearn.decomposition.PCA: PCA (Principal Component Analysis) is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving most of the variance in the data. It's commonly used for exploratory data analysis and feature extraction.

14. yellowbrick.cluster.KElbowVisualizer: Yellowbrick is a visualization library for machine learning built on top of scikit-learn and matplotlib. The `KElbowVisualizer` class provides a visual aid for determining the optimal number of clusters in KMeans clustering by plotting the within-cluster sum of squares (inertia) against the number of clusters.

15. yellowbrick.cluster.SilhouetteVisualizer: Another visualization tool from Yellowbrick, the `SilhouetteVisualizer` class visualizes the silhouette scores of individual data points in a cluster. It helps evaluate the quality of clustering by assessing how well-separated clusters are.

16. sklearn.metrics: This submodule of scikit-learn contains various metrics for evaluating the performance of machine learning models. In your code, you might use functions like `silhouette_score`, `calinski_harabasz_score`, and `davies_bouldin_score` to assess the quality of clustering algorithms.

17. sklearn.cluster.KMeans: KMeans is a popular clustering algorithm used for partitioning data into K clusters based on similarity. It's implemented in scikit-learn's `KMeans` class, which you can use to perform KMeans clustering on your data.

18. tabulate: Tabulate is a Python library for printing tabular data in a visually appealing format. It's useful for generating formatted tables for displaying data in reports or command-line interfaces.

19. collections.Counter: Counter is a built-in Python class in the `collections` module that is used for counting the occurrences of elements in a collection. It's handy for tasks like frequency counting and histogram generation.

These libraries cover a wide range of functionalities, from data manipulation and visualization to machine learning and statistical analysis, allowing you to perform various tasks efficiently within your Python environment.

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

```
!pip install chardet
```

```
Requirement already satisfied: chardet in /opt/anaconda3/lib/python3.11/site-packa
```

```
import chardet
```

Figure 3:Loading Dataset

Loading the dataset

Loading the dataset into a pandas dataframe which will facilitate easy manipulation and analysis

```
data = pd.read_csv("data 3.csv",encoding='latin1')
```

```
import csv

# Define a function to detect encoding of a file
def detect_encoding(file_path):
    # List of encodings to try
    encodings = ['utf-8', 'iso-8859-1', 'cp1252']  # Add more encodings if needed

    # Iterate over encodings and try to open the file
    for encoding in encodings:
        try:
            with open(file_path, 'r', encoding=encoding) as f:
                csv.reader(f)
            return encoding
        except UnicodeDecodeError:
            continue

    # If no encoding is found
    return None

# Example usage:
file_path = "data 3.csv"
detected_encoding = detect_encoding(file_path)
if detected_encoding:
    print("Detected encoding:", detected_encoding)
else:
    print("Unable to detect encoding.")
```

```
Detected encoding: utf-8
```

Figure 4:Encoding dataset

```
import pandas as pd

# Define a function to read CSV file with detected encoding
def read_csv_with_encoding(file_path):
    # List of encodings to try
    encodings = ['utf-8', 'iso-8859-1', 'cp1252']  # Add more encodings if needed

    # Iterate over encodings and try to read the CSV file
    for encoding in encodings:
        try:
            df = pd.read_csv(file_path, encoding=encoding)
            return df
        except UnicodeDecodeError:
            continue

    # If no encoding is found
    return None

# Example usage:
file_path = "data 3.csv"
df = read_csv_with_encoding(file_path)
if df is not None:
    print("CSV file read successfully.")
    # Process the DataFrame as needed
else:
    print("Unable to read CSV file. No suitable encoding found.")
```

```
CSV file read successfully.
```

Figure 5: Reading CSV file

2. Initial Data Analysis

Dataset Overview

Gaining a thorough understanding of the dataset before proceeding to data cleaning and transformation.Performing a preliminary analysis to understand the structure and types of data columns:

```
df.head(10)
```

	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
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850.0	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850.0	United Kingdom
7	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850.0	United Kingdom
8	536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	17850.0	United Kingdom
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	13047.0	United Kingdom

Figure 6:Displaying Columns

```
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
```

Figure 7:Info on type of variables

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

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

Summary Statistics

Generating summary statistics to gain insights into data distribution

```
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

Figure 8:Providing statistics for each 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

Figure 9:Providing statistics for each columns including object

3. Data Cleaning and Transformation

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

Handling Missing Values:

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

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

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

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

# Annotate the values and indexes
for i, (value, name) in enumerate(zip(missing_percentage, missing_percentage.index)):
    ax.text(value+0.5, i, f"{value:.2f}%", ha='left', va='center', fontweight='bold', fontsize=18,

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

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

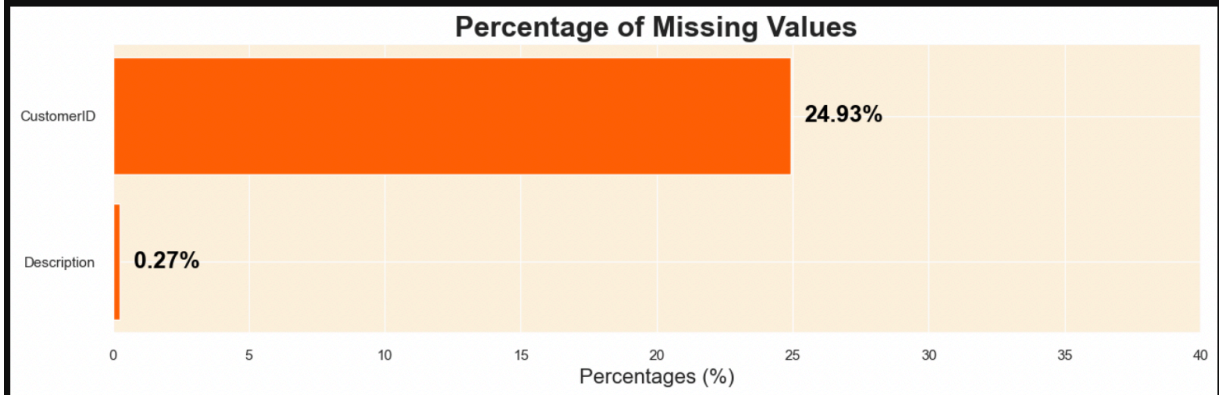

Figure 10:Percentage of Missing Values

Handling Missing Values Strategy:

CustomerID (24.93% missing values)

The CustomerID column contains nearly a quarter of missing data. This column is essential for clustering customers and creating a recommendation system. Imputing such a large percentage of missing values might introduce significant bias or noise into the analysis.

Moreover, since the clustering is based on customer behavior and preferences, it's crucial to have accurate data on customer identifiers. Therefore, removing the rows with missing CustomerIDs seems to be the most reasonable approach to maintain the integrity of the clusters and the analysis.

Description (0.27% missing values)

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

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

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

```
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

Figure 11:Recognizing missing values

```
df = df.dropna(subset=['CustomerID', 'Description'])
```

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

```
0
```

Figure 12:Handling missing values

Handling Duplicates

Recognizing duplicate rows

Handling Duplicates Strategy:

In the context of this project, the presence of completely identical rows, including identical transaction times, suggests that these might be data recording errors rather than genuine repeated transactions. Keeping these duplicate rows can introduce noise and potential inaccuracies in the clustering and recommendation system.

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

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

# Sorting the data by certain columns to see the duplicate rows next to each other
duplicate_rows_sorted = duplicate_rows.sort_values(by=['InvoiceNo', 'StockCode', 'Description', 'C

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

	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
527	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	12/1/2010 11:45	2.10	17908.0	United Kingdom
521	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	12/1/2010 11:45	2.95	17908.0	United Kingdom
537	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	12/1/2010 11:45	2.95	17908.0	United Kingdom
578	536412	21448	12 DAISY PEGS IN WOOD BOX	1	12/1/2010 11:49	1.65	17920.0	United Kingdom
598	536412	21448	12 DAISY PEGS IN WOOD BOX	1	12/1/2010 11:49	1.65	17920.0	United Kingdom

Figure 13: Handling Duplicates

```
print(f"The dataset contains {df.duplicated().sum()} duplicate rows that need to be removed.")

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

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

Figure 14: Printing Duplicate rows

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

```
401604
```

Figure 15:Size of the dataset

Treating Cancelled Transactions

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

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

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

	Quantity	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

Figure 16:Treating Cancelled transactions

```
cancelled_percentage = (cancelled_transactions.shape[0] / df.shape[0]) * 100

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

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

Figure 17: Printing the % of cancelled transactions

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.

Correcting StockCode Anomalies

Finding the number of unique stock codes and to plot the top 10 most frequent stock codes along with their percentage frequency:

```
unique_stock_codes = df['StockCode'].nunique()

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

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

Figure 18: No of unique stock codes in data

```python
top_10_stock_codes = df['StockCode'].value_counts(normalize=True).head(10) * 100

# Plotting the top 10 most frequent stock codes
plt.figure(figsize=(12, 5))
top_10_stock_codes.plot(kind='barh', color='#ff6200')

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

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

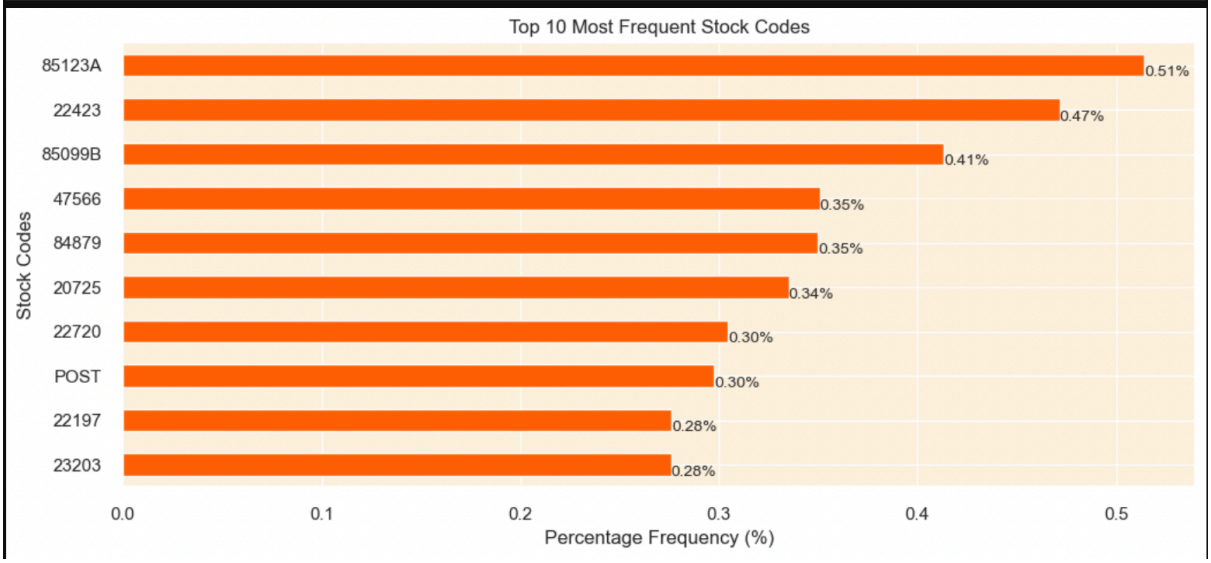

Figure 19: Displaying Top 10 stock codes

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

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

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

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

Figure 20:Frequeny of numeric character in stock code

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

```
anomalous_stock_codes = [code for code in unique_stock_codes if sum(c.isdigit() for c in str(code)

# Printing each stock code on a new line
print("Anomalous stock codes:")
print("-"*22)
for code in anomalous_stock_codes:
    print(code)
```

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

Figure 21: Stock code containing 0 and 1 numeric characters

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

```
percentage_anomalous = (df['StockCode'].isin(anomalous_stock_codes).sum() / len(df)) * 100

# Printing the percentage
print(f"The percentage of records with anomalous stock codes in the dataset is: {percentage_anomal
```

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

Figure 22: Calculating % of records

```
df = df[~df['StockCode'].isin(anomalous_stock_codes)]
```

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

```
399689
```

Figure 23: Size of the dataset

Cleaning Description Column:

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

```
description_counts = df['Description'].value_counts()

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

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

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

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

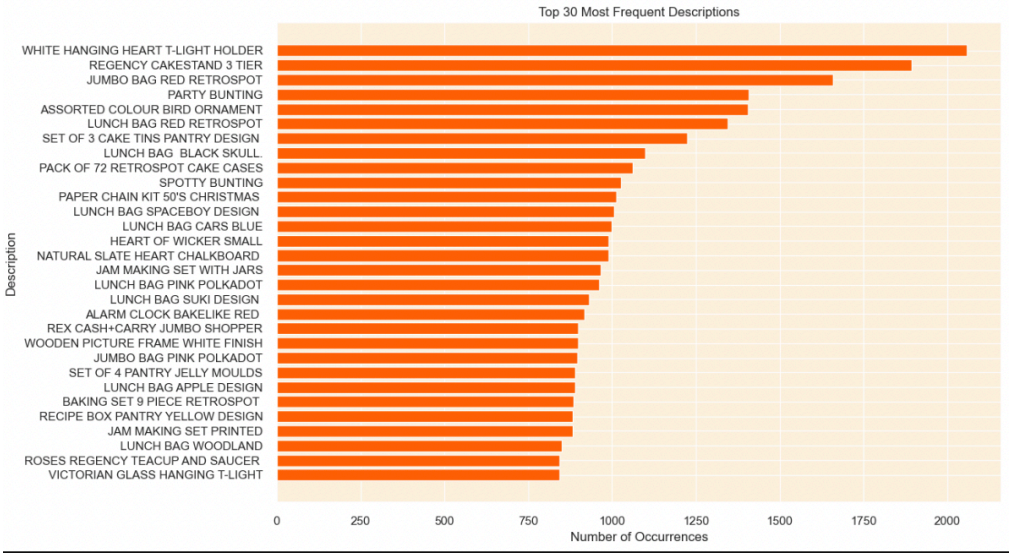

Figure 24: Most frequent descriptions

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.

```
lowercase_descriptions = df['Description'].unique()
lowercase_descriptions = [desc for desc in lowercase_descriptions if any(char.islower()

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

```
The unique descriptions containing lowercase characters are:
------------------------------------------------------------
BAG 500g SWIRLY MARBLES
POLYESTER FILLER PAD 45x45cm
POLYESTER FILLER PAD 45x30cm
POLYESTER FILLER PAD 40x40cm
FRENCH BLUE METAL DOOR SIGN No
BAG 250g SWIRLY MARBLES
BAG 125g SWIRLY MARBLES
3 TRADITIONAl BISCUIT CUTTERS  SET
NUMBER TILE COTTAGE GARDEN No
FOLK ART GREETING CARD,pack/12
ESSENTIAL BALM 3.5g TIN IN ENVELOPE
POLYESTER FILLER PAD 65CMx65CM
NUMBER TILE VINTAGE FONT No
POLYESTER FILLER PAD 30CMx30CM
POLYESTER FILLER PAD 60x40cm
FLOWERS HANDBAG blue and orange
Next Day Carriage
THE KING GIFT BAG 25x24x12cm
High Resolution Image
```

Figure 25: Unique descriptions containing lower case

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.

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

# Calculate the percentage of records with service-related descriptions
service_related_percentage = df[df['Description'].isin(service_related_descriptions)].sh

# Print the percentage of records with service-related descriptions
print(f"The percentage of records with service-related descriptions in the dataset is: {

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

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

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

Figure 26: records with service-related descriptions

Treating Zero Unit Prices:

In this step let us take a look at the statistical description of the UnitPrice column:

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

```
399606
```

```
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
```

Figure 27:Datatype of unit

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

	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

Figure 28:statistics of unit price

The transactions with a unit price of zero are relatively few in number (33 transactions). These transactions have a large variability in the quantity of items involved, ranging from 1 to 12540, with a substantial standard deviation.

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

```
df = df[df['UnitPrice'] > 0]
```

Figure 29:Unit price>0

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=True)
```

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

```
399573
```

Figure 30:Resetting index

Step 4 Feature Engineering

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

RFM Features

RFM is a method used for analyzing customer value and segmenting the customer base.

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.

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

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

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

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

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

# Calculate the number of days since the last purchase for each customer
customer_data['Days_Since_Last_Purchase'] = (most_recent_date - customer_data['InvoiceDa

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

Figure 31: Identifying engagement level of customer

Now, customer_data dataframe contains the Days_Since_Last_Purchase feature:

```
customer_data.head()
```

	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

Figure 32:Added day_since_last_purchased feature

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, we will create two features that quantify the frequency of a customer's engagement with the retailer:

Total Transaction

Total Products Purchased

```
total_transactions = df.groupby('CustomerID')['InvoiceNo'].nunique().reset_index()
total_transactions.rename(columns={'InvoiceNo': 'Total_Transactions'}, inplace=True)

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

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

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

	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

Figure 33: Creating features

Monetary(M):

In this step, we will create two features that represent the monetary aspect of customer's transactions:

Total Spend

Average Transaction Value

```
df['Total_Spend'] = df['UnitPrice'] * df['Quantity']
total_spend = df.groupby('CustomerID')['Total_Spend'].sum().reset_index()

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

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, total_spend, on='CustomerID')
customer_data = pd.merge(customer_data, average_transaction_value[['CustomerID', 'Averag

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

CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend
12346.0	325	2	0	0.00
12347.0	2	7	2458	4310.00
12348.0	75	4	2332	1437.24
12349.0	18	1	630	1457.55
12350.0	310	1	196	294.40

Figure 34:Creating feature for monetory aspect of customer transaction

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.

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

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

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

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spe
0	12346.0	325	2	0	0.
1	12347.0	2	7	2458	4310.
2	12348.0	75	4	2332	1437.
3	12349.0	18	1	630	1457.
4	12350.0	310	1	196	294.

Figure 35:Product diversity

Behavioral Features:

In this step, we aim to understand and capture the shopping patterns and behaviors of customers. These features will give us insights into the customers' preferences regarding when they like to shop, which can be crucial information for personalizing their shopping experience. Here are the features I am planning to introduce:

Average Days Between Purchases
Favorite Shopping Day
Favorite Shopping Hour

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

```
df['Day_Of_Week'] = df['InvoiceDate'].dt.dayofweek
df['Hour'] = df['InvoiceDate'].dt.hour

# Calculate the average number of days between consecutive purchases
days_between_purchases = df.groupby('CustomerID')['InvoiceDay'].apply(lambda x: (x.diff().dropna()
average_days_between_purchases = days_between_purchases.groupby('CustomerID').mean().reset_index()
average_days_between_purchases.rename(columns={'InvoiceDay': 'Average_Days_Between_Purchases'}, in

# Find the favorite shopping day of the week
favorite_shopping_day = df.groupby(['CustomerID', 'Day_Of_Week']).size().reset_index(name='Count')
favorite_shopping_day = favorite_shopping_day.loc[favorite_shopping_day.groupby('CustomerID')['Cou

# Find the favorite shopping hour of the day
favorite_shopping_hour = df.groupby(['CustomerID', 'Hour']).size().reset_index(name='Count')
favorite_shopping_hour = favorite_shopping_hour.loc[favorite_shopping_hour.groupby('CustomerID')['

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

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

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend	Averag
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	

Figure 36:Including beahvioral features

Geographic Features:

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

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

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

Figure 37:Introducing geographical features

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

# Get the country with the maximum number of transactions for each customer (in case a customer ha
customer_main_country = customer_country.sort_values('Number_of_Transactions', ascending=False).dr

# Create a binary column indicating whether the customer is from the UK or not
customer_main_country['Is_UK'] = customer_main_country['Country'].apply(lambda x: 1 if x == 'Unite

# Merge this data with our customer_data dataframe
customer_data = pd.merge(customer_data, customer_main_country[['CustomerID', 'Is_UK']], on='Custom

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

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend	Averag
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	

Figure 38:Adding geographical features

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

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

Figure 39:Checking by searching the feature

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

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
Cancellation Rate

By incorporating these cancellation insights into our dataset, we can build a more comprehensive view of customer behavior, which could potentially aid in creating more effective and nuanced customer segmentation.

```
total_transactions = df.groupby('CustomerID')['InvoiceNo'].nunique().reset_index()

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

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

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

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

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

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend	Averag
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	

Figure 40:Cancellation insights

Seasonality & 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:

Monhly_spending_mean
Monthly_Spending_Std
Spending_Trend

By incorporating these detailed insights into our customer segmentation model, we can create more precise and actionable customer groups, facilitating the development of highly targeted marketing strategies and promotions.

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

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

# Calculate Seasonal Buying Patterns: We are using monthly frequency as a proxy for seasonal buyin
seasonal_buying_patterns = monthly_spending.groupby('CustomerID')['Total_Spend'].agg(['mean', 'std
seasonal_buying_patterns.rename(columns={'mean': 'Monthly_Spending_Mean', 'std': 'Monthly_Spending

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

# Calculate Trends in Spending
# We are using the slope of the linear trend line fitted to the customer's spending over time as a
def calculate_trend(spend_data):
    # If there are more than one data points, we calculate the trend using linear regression
    if len(spend_data) > 1:
        x = np.arange(len(spend_data))
        slope, _, _, _, _ = linregress(x, spend_data)
        return slope
    # If there is only one data point, no trend can be calculated, hence we return 0
    else:
        return 0

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

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

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

Figure 41:Seasonality and trend

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend	Averag
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	

Figure 42:dataset output with variety features

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

```
customer_data['CustomerID'] = customer_data['CustomerID'].astype(str)

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

```
customer_data.head(10)
```

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend	Averag
0	12346.0	325	2	0	0.0	
1	12347.0	2	7	2458	4310.0	
2	12348.0	75	4	2332	1437.24	
3	12349.0	18	1	630	1457.55	
4	12350.0	310	1	196	294.4	
5	12352.0	36	8	463	1265.41	
6	12353.0	204	1	20	89.0	
7	12354.0	232	1	530	1079.4	
8	12355.0	214	1	240	459.4	
9	12356.0	22	3	1573	2487.43	

Figure 43:Displaying columns

```
customer_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4282 entries, 0 to 4281
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   CustomerID                      4282 non-null   string
 1   Days_Since_Last_Purchase        4282 non-null   Int64
 2   Total_Transactions              4282 non-null   Int64
 3   Total_Products_Purchased        4282 non-null   Int64
 4   Total_Spend                     4282 non-null   Float64
 5   Average_Transaction_Value       4282 non-null   Float64
 6   Unique_Products_Purchased       4282 non-null   Int64
 7   Average_Days_Between_Purchases  4282 non-null   Float64
 8   Day_Of_Week                     4282 non-null   Int32
 9   Hour                            4282 non-null   Int32
 10  Is_UK                           4282 non-null   Int64
 11  Cancellation_Frequency          4282 non-null   Int64
 12  Cancellation_Rate               4282 non-null   Float64
 13  Monthly_Spending_Mean           4282 non-null   Float64
 14  Monthly_Spending_Std            4282 non-null   Float64
 15  Spending_Trend                  4282 non-null   Float64
dtypes: Float64(7), Int32(2), Int64(6), string(1)
memory usage: 564.7 KB
```

Figure 44:Displaying columns data type

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.

```
model = IsolationForest(contamination=0.05, random_state=0)

# Fitting the model on our dataset (converting DataFrame to NumPy to avoid warning)
customer_data['Outlier_Scores'] = model.fit_predict(customer_data.iloc[:, 1:].to_numpy())

# Creating a new column to identify outliers (1 for inliers and -1 for outliers)
customer_data['Is_Outlier'] = [1 if x == -1 else 0 for x in customer_data['Outlier_Scores']]

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

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend	Averag
0	12346.0	325	2	0	0.0	
1	12347.0	2	7	2458	4310.0	
2	12348.0	75	4	2332	1437.24	
3	12349.0	18	1	630	1457.55	
4	12350.0	310	1	196	294.4	

Figure 45:Outlier detection

After applying the Isolation Forest algorithm, we have identified the outliers and marked them in a new column named Is_Outlier. We have also calculated the outlier scores which represent the anomaly score of each record. Now let's visualize the distribution of these scores and the number of inliers and outliers detected by the model:

```
outlier_percentage = customer_data['Is_Outlier'].value_counts(normalize=True) * 100

# Plotting the percentage of inliers and outliers
plt.figure(figsize=(12, 4))
outlier_percentage.plot(kind='barh', color='#ff6200')

# Adding the percentage labels on the bars
for index, value in enumerate(outlier_percentage):
    plt.text(value, index, f'{value:.2f}%', fontsize=15)

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

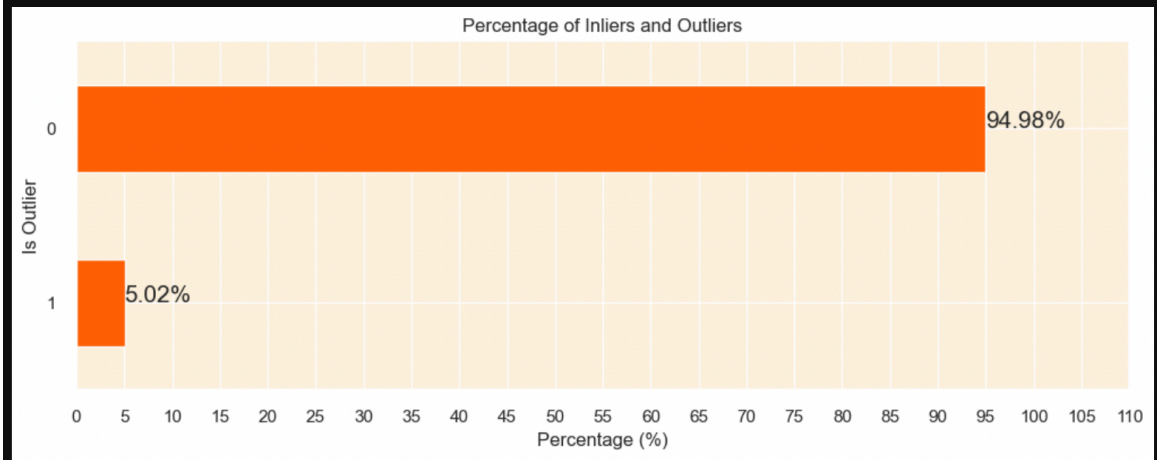

Figure 46: % of inliers and outliers

After applying the Isolation Forest algorithm, we have identified the outliers and marked them in a new column named Is_Outlier. We have also calculated the outlier scores which represent the anomaly score of each record. Now let's visualize the distribution of these scores and the number of inliers and outliers detected by the model:

```
outliers_data = customer_data[customer_data['Is_Outlier'] == 1]

# Remove the outliers from the main dataset
customer_data_cleaned = customer_data[customer_data['Is_Outlier'] == 0]

# Drop the 'Outlier_Scores' and 'Is_Outlier' columns
customer_data_cleaned = customer_data_cleaned.drop(columns=['Outlier_Scores', 'Is_Outlier'])

# Reset the index of the cleaned data
customer_data_cleaned.reset_index(drop=True, inplace=True)
```

```
customer_data_cleaned.shape[0]
```

```
4067
```

Figure 47: Cleaned data

We have successfully separated the outliers for further analysis and cleaned our main dataset by removing these outliers. This cleaned dataset is now ready for the next steps in our customer segmentation project, which includes scaling the features and applying clustering algorithms to identify distinct customer segments.

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.

```
sns.set_style('whitegrid')

# Calculate the correlation matrix excluding the 'CustomerID' column
corr = customer_data_cleaned.drop(columns=['CustomerID']).corr()

# Define a custom colormap
colors = ['#ff6200', '#ffcaa8', 'white', '#ffcaa8', '#ff6200']
my_cmap = LinearSegmentedColormap.from_list('custom_map', colors, N=256)

# Create a mask to only show the lower triangle of the matrix (since it's mirrored around its
# top-left to bottom-right diagonal)
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, k=1)] = True

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr, mask=mask, cmap=my_cmap, annot=True, center=0, fmt='.2f', linewidths=2)
plt.title('Correlation Matrix', fontsize=14)
plt.show()
```

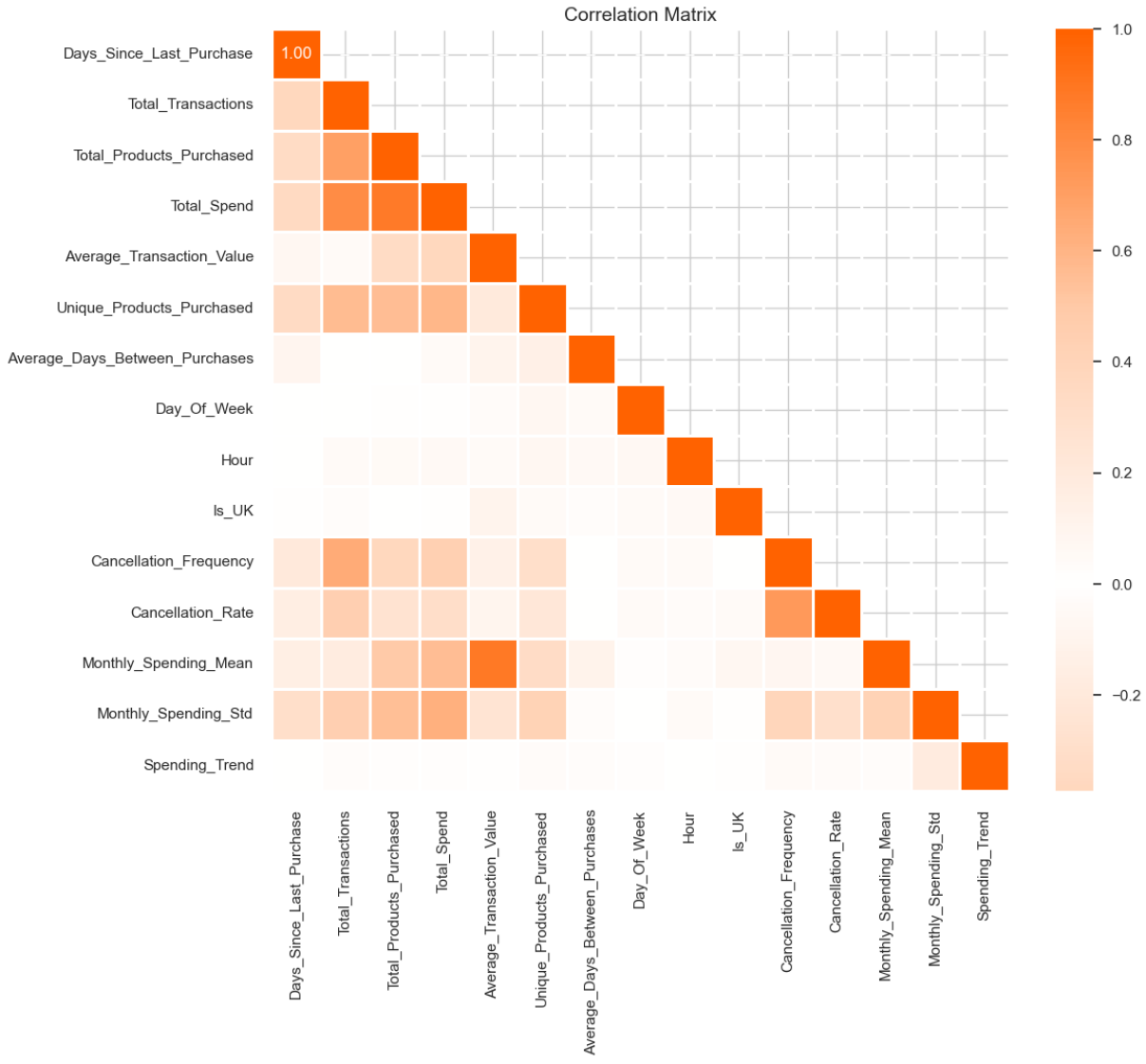

Figure 48:Correlation matrix

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.

```
scaler = StandardScaler()

# List of columns that don't need to be scaled
columns_to_exclude = ['CustomerID', 'Is_UK', 'Day_Of_Week']

# List of columns that need to be scaled
columns_to_scale = customer_data_cleaned.columns.difference(columns_to_exclude)

# 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_scal

# Display the first few rows of the scaled data
customer_data_scaled.head()
```

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend	Averag
0	12346.0	2.345802	-0.477589	-0.754491	-0.813464	
1	12347.0	-0.905575	0.707930	2.005048	2.366920	
2	12348.0	-0.170744	-0.003381	1.863591	0.247087	
3	12349.0	-0.744516	-0.714692	-0.047205	0.262074	
4	12350.0	2.194809	-0.714692	-0.534446	-0.596223	

Figure 49:Feature Scaling

8. Dimensionality Reduction:

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.

```
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 number of components
plt.figure(figsize=(20, 10))

# 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 =

# 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'Opti
           loc=(0.62, 0.1),
           frameon=True,
           framealpha=1.0,
           edgecolor='#ff6200')

# Display the variance values 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_va
    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="bo

plt.grid(axis='both')
plt.show()
```

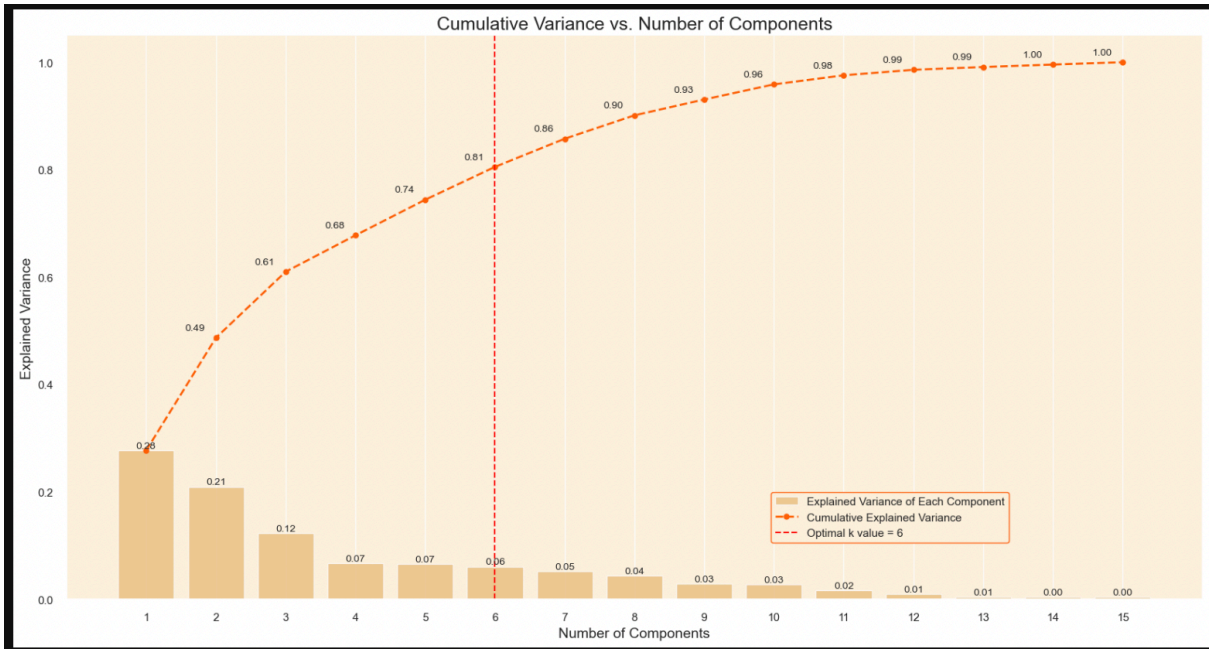

Figure 50:Cumulative variance Vs No of Componenets

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.

```
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)

# Creating a new dataframe from the PCA dataframe, with columns labeled PC1, PC2, etc.
customer_data_pca = pd.DataFrame(customer_data_pca, columns=['PC'+str(i+1) for i in range(pca.n_cc

# Adding the CustomerID index back to the new PCA dataframe
customer_data_pca.index = customer_data_scaled.index
```

```
customer_data_pca.head()
```

CustomerID	PC1	PC2	PC3	PC4	PC5	PC6
12346.0	-2.186469	-1.705370	-1.576745	1.008187	-0.411803	-1.658012
12347.0	3.290264	-1.387375	1.923310	-0.930990	-0.010591	0.873150
12348.0	0.584684	0.585019	0.664727	-0.655411	-0.470280	2.306657
12349.0	1.791116	-2.695652	5.850040	0.853418	0.677111	-1.520098
12350.0	-1.997139	-0.542639	0.578781	0.183682	-1.484838	0.062672

Figure 51:PCA Components

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

```
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 the PCA component DataFrame and apply the highlighting function
pc_df = pd.DataFrame(pca.components_.T, columns=['PC{}'.format(i+1) for i in range(pca.n_component
                     index=customer_data_scaled.columns)

pc_df.style.apply(highlight_top3, axis=0)
```

	PC1	PC2	PC3	PC4	PC5	PC6
Days_Since_Last_Purchase	-0.217859	-0.013986	0.067660	0.273430	-0.240968	-0.373059
Total_Transactions	0.380301	0.014759	-0.259180	-0.138165	-0.017356	-0.028257
Total_Products_Purchased	0.401425	0.007365	0.069133	-0.134806	0.057476	-0.013373
Total_Spend	0.431260	0.010159	0.065165	-0.092047	0.025202	-0.036947
Average_Transaction_Value	0.176225	-0.015544	0.589050	0.114307	0.021847	-0.101738
Unique_Products_Purchased	0.324992	0.063346	0.014010	-0.230502	-0.193981	0.124604
Average_Days_Between_Purchases	-0.022600	-0.036007	-0.127341	-0.160627	0.753462	0.211787
Day_Of_Week	-0.026572	0.994650	-0.006591	0.028870	0.058359	-0.060799
Hour	-0.024259	0.056388	-0.002019	-0.226832	-0.528881	0.621915
Is_UK	-0.001014	0.007435	-0.018378	-0.013419	-0.005353	0.014384
Cancellation_Frequency	0.287102	-0.018576	-0.400697	0.225923	-0.100595	-0.168050
Cancellation_Rate	0.229885	-0.022616	-0.381347	0.290702	-0.126048	-0.216073
Monthly_Spending_Mean	0.274127	-0.005116	0.498142	0.134989	-0.004123	-0.101941
Monthly_Spending_Std	0.334168	0.014557	0.036156	0.218369	0.128494	0.193648
Spending_Trend	-0.014574	-0.014845	-0.002987	-0.730466	-0.078739	-0.523692

Figure 52:Extracting Coefficient

9. K-Means Clustering:

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.

Determing the optimal Number 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.

```
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
km = KMeans(init='k-means++', n_init=10, max_iter=100, random_state=0)

# Create a figure and axis 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 pl
visualizer = KElbowVisualizer(km, k=(2, 15), timings=False, ax=ax)

# Fit the data to the visualizer
visualizer.fit(customer_data_pca)

# Finalize and render the figure
visualizer.show();
```

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.

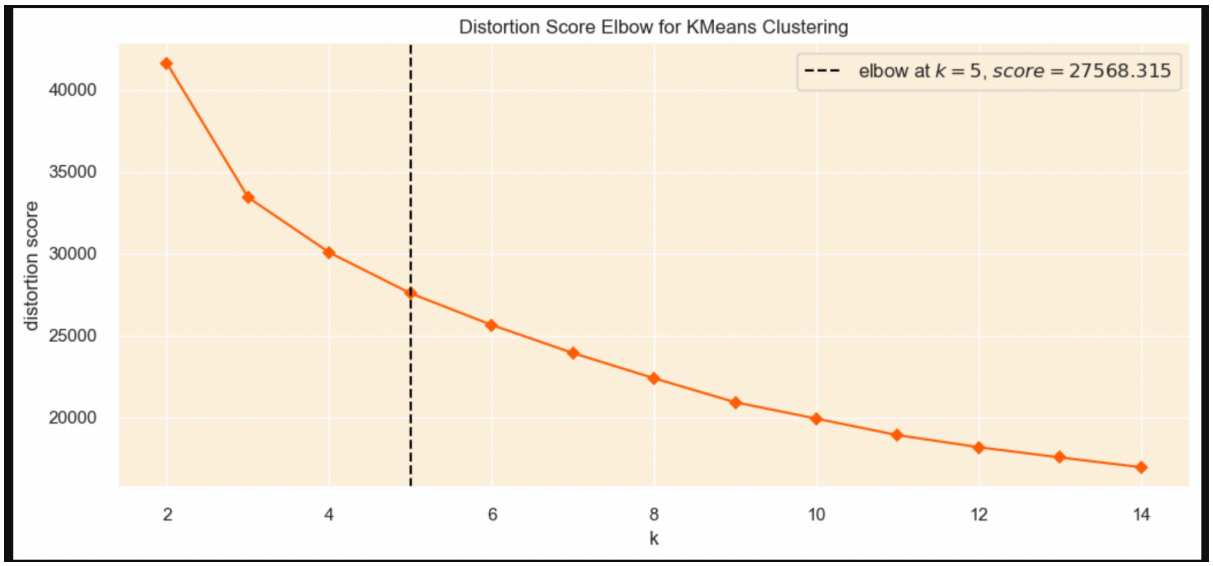

Figure 53: Distortion score elbow for kmeans clustering

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 Yellow Brick 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

Silhouette Method

- 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.

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

    # Set the size of the figure
    plt.figure(figsize=figsize)

    # Create a grid with (stop_k - start_k + 1) rows and 2 columns
    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 through the range of k values
    for k in range(start_k, stop_k + 1):
        km = KMeans(n_clusters=k, init='k-means++', n_init=10, max_iter=100, random_state=0)
        km.fit(df)
        labels = km.predict(df)
        score = silhouette_score(df, labels)
        silhouette_scores.append(score)

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

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

```python
    # Add the optimal 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', boxst


    # 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))
```

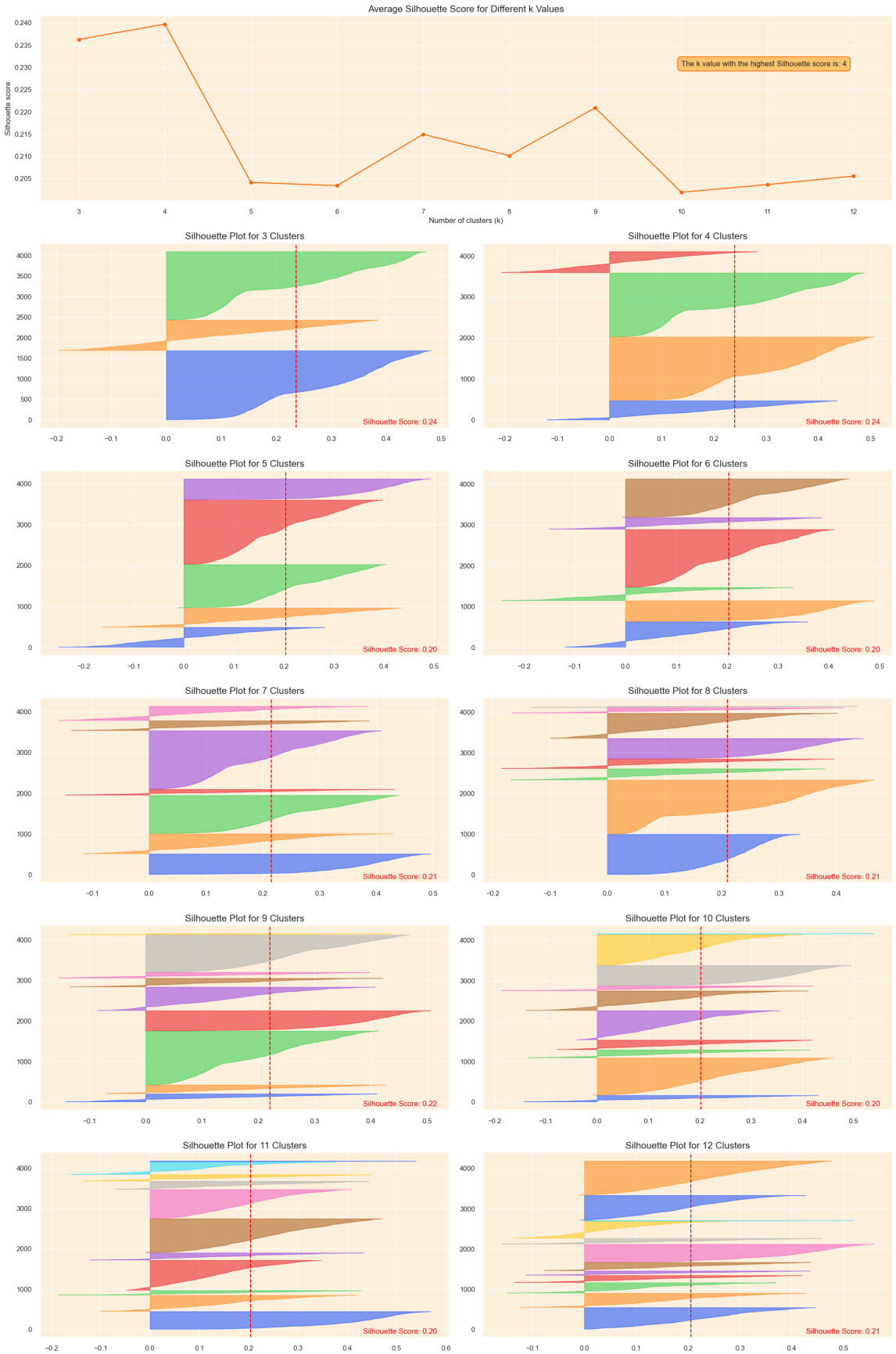

Figure 54: Silhouette Score

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 – Kmeans

In this step, I am going to apply the K-means clustering algorithm to segment customers into different clusters based on their purchasing behaviors 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.

```
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
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['cluster'] = new_labels

# Append the new cluster labels to the PCA version of the dataset
customer_data_pca['cluster'] = new_labels
```

```
customer_data_cleaned.head()
```

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_Purchased	Total_Spend	Averag
0	12346.0	325	2	0	0.0	
1	12347.0	2	7	2458	4310.0	
2	12348.0	75	4	2332	1437.24	
3	12349.0	18	1	630	1457.55	
4	12350.0	310	1	196	294.4	

Figure 55: Cleaned Data

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:

1. 3D Visualization of Top PC

2. Cluster Distribution Visualization

3. Evaluation Metrics

- Silhouette Score
- Calinski Harabasz Score
- Davies Bouldin Score

Note: We are using the PCA version of the dataset for evaluation because this is the space where the clusters were actually formed, capturing the most significant patterns in the data. Evaluating in this space ensures a more accurate representation of the cluster quality, helping us understand the true cohesion and separation achieved during clustering. This approach also aids in creating a clearer 3D visualization using the top principal components, illustrating the actual separation between clusters.

3D Visualization of Top Principal Components

```
colors = ['#e8000b', '#1ac938', '#023eff']

cluster_0 = customer_data_pca[customer_data_pca['cluster'] == 0]
cluster_1 = customer_data_pca[customer_data_pca['cluster'] == 1]
cluster_2 = customer_data_pca[customer_data_pca['cluster'] == 2]

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

# Add data points for each cluster separately and specify the color
fig.add_trace(go.Scatter3d(x=cluster_0['PC1'], y=cluster_0['PC2'], z=cluster_0['PC3'],
                           mode='markers', marker=dict(color=colors[0], size=5, opacity=0.4), name
fig.add_trace(go.Scatter3d(x=cluster_1['PC1'], y=cluster_1['PC2'], z=cluster_1['PC3'],
                           mode='markers', marker=dict(color=colors[1], size=5, opacity=0.4), name
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

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

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

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:

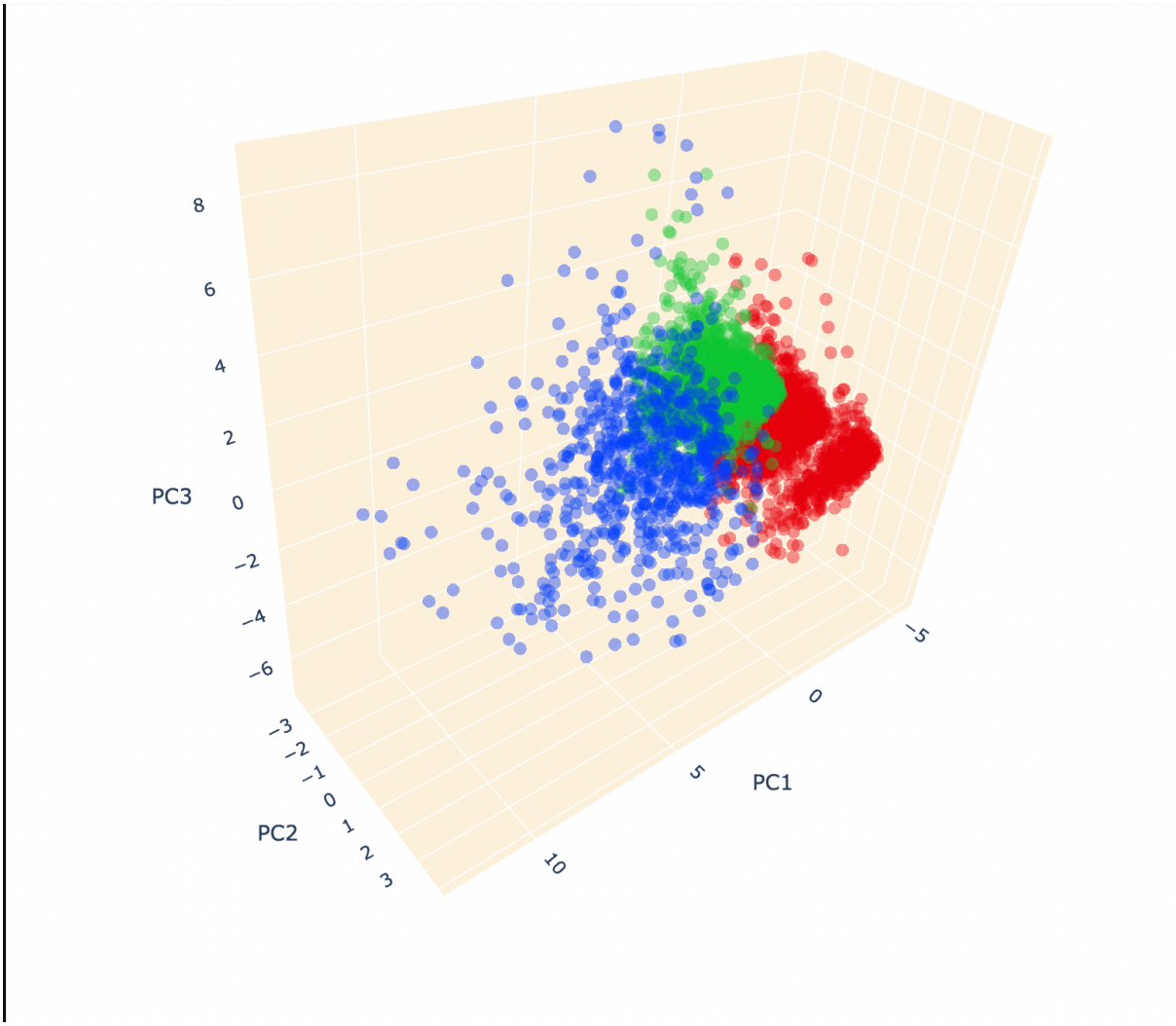

Figure 56:Showing top 3 PC

Cluster Distribution Visualization

Utilize a bar plot to visualize the percentage of customers in each cluster, which helps in understanding if the clusters are balanced and significant:

```
cluster_percentage = (customer_data_pca['cluster'].value_counts(normalize=True) * 100).reset_i
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)

# 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 (%)')

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

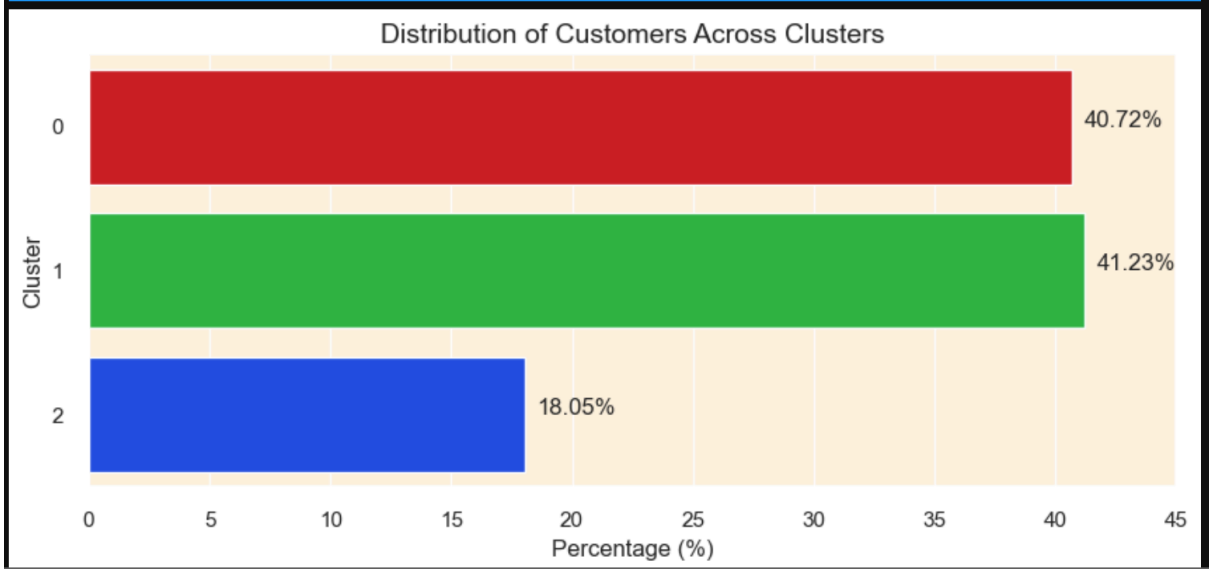

Figure 57: Cluster Percentage

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.

Evaluating 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.

```
num_observations = len(customer_data_pca)

# Separate the features and the cluster labels
X = customer_data_pca.drop('cluster', axis=1)
clusters = customer_data_pca['cluster']

# Compute the metrics
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 metrics and the number of observations
table_data = [
    ["Number of Observations", num_observations],
    ["Silhouette Score", sil_score],
    ["Calinski Harabasz Score", calinski_score],
    ["Davies Bouldin Score", davies_score]
]

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

```
+-------------------------+--------------------+
|         Metric          |       Value        |
+-------------------------+--------------------+
| Number of Observations  |        4067        |
|    Silhouette Score     | 0.23627137022779893 |
| Calinski Harabasz Score | 1257.1794962921526 |
|  Davies Bouldin Score   | 1.3684055372996033 |
+-------------------------+--------------------+
```

Figure 58: Printing Metrics

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

In this section, I am going to analyze the characteristics of each cluster to understand the distinct behaviors and preferences of different customer segments and also profile each cluster to identify the key traits that define the customers in each cluster.

Radar Chart Approach

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

```
df_customer = customer_data_cleaned.set_index('CustomerID')

# Standardize the data (excluding the cluster column)
scaler = StandardScaler()
df_customer_standardized = scaler.fit_transform(df_customer.drop(columns=['cluster'], axis=1))

# Create a new dataframe with standardized values and add the cluster column back
df_customer_standardized = pd.DataFrame(df_customer_standardized, columns=df_customer.columns[
df_customer_standardized['cluster'] = df_customer['cluster']

# Calculate the centroids of each cluster
cluster_centroids = df_customer_standardized.groupby('cluster').mean()

# Function to create a radar chart
def create_radar_chart(ax, angles, data, 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()
```

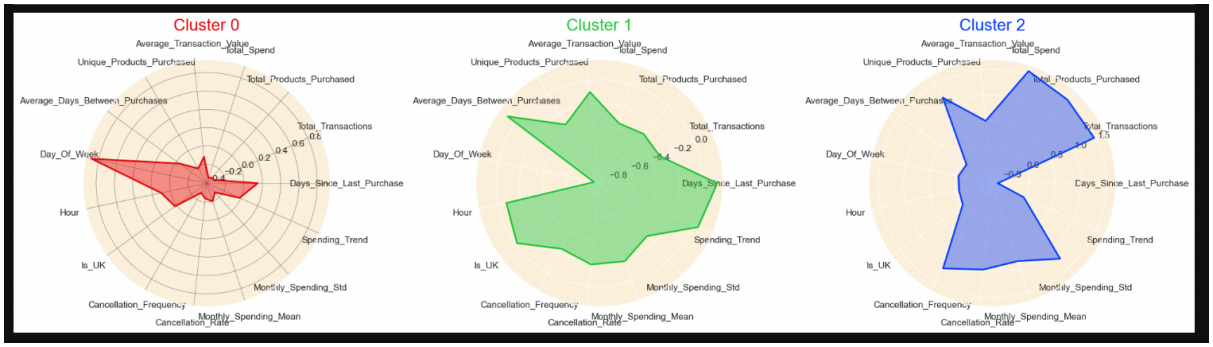

Figure 59: Customer Profiles

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.

```
features = customer_data_cleaned.columns[1:-1]
clusters = customer_data_cleaned['cluster'].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['cluster'] == cluster][feature]
        axes[i, j].hist(data, bins=20, color=colors[j], edgecolor='w', alpha=0.7)
        axes[i, j].set_title(f'Cluster {cluster} - {feature}', fontsize=15)
        axes[i, j].set_xlabel('')
        axes[i, j].set_ylabel('')

# Adjusting layout to prevent overlapping
plt.tight_layout()
plt.show()
```

The detailed insights from the histograms provide a more nuanced understanding of each cluster, helping in refining the profiles to represent the customer behaviors more accurately. Based on the detailed analysis from both the radar charts and the histograms, here are the refined profiles and titles for each cluster:

Figure 60:histogram Chart Approach

12. Recommendation

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.

```
outlier_customer_ids = outliers_data['CustomerID'].astype('float').unique()
df_filtered = df[~df['CustomerID'].isin(outlier_customer_ids)]

# Step 2: Ensure consistent data type 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 eac
merged_data = df_filtered.merge(customer_data_cleaned[['CustomerID', 'cluster']], on='CustomerID',

# Step 4: Identify the top 10 best-selling products in each cluster based on the total quantity so
best_selling_products = merged_data.groupby(['cluster', 'StockCode', 'Description'])['Quantity'].s
best_selling_products = best_selling_products.sort_values(by=['cluster', 'Quantity'], ascending=[T
top_products_per_cluster = best_selling_products.groupby('cluster').head(10)

# Step 5: Create a record of products purchased by each customer in each cluster
customer_purchases = merged_data.groupby(['CustomerID', 'cluster', 'StockCode'])['Quantity'].sum()
```

```
# Step 6: Generate recommendations for each customer in each cluster
recommendations = []
for cluster in top_products_per_cluster['cluster'].unique():
    top_products = top_products_per_cluster[top_products_per_cluster['cluster'] == cluster]
    customers_in_cluster = customer_data_cleaned[customer_data_cleaned['cluster'] == cluster]['Cus

    for customer in customers_in_cluster:
        # Identify products already purchased by the customer
        customer_purchased_products = customer_purchases[(customer_purchases['CustomerID'] == cust
                                                          (customer_purchases['cluster'] == cluster

        # 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_purchas
        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', 'D

# Step 7: Create a dataframe from the recommendations list and merge it with the original customer
recommendations_df = pd.DataFrame(recommendations, columns=['CustomerID', 'cluster', 'Rec1_StockCo
                                                            'Rec2_StockCode', 'Rec2_Description', 'Rec3_Stock
customer_data_with_recommendations = customer_data_cleaned.merge(recommendations_df, on=['Customer
```

```
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_D
15746.0	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	ASSORTED COLOURS SILK FAN	85123A	WHITE HEAR
15728.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	A COL
17459.0	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	84879	ASSORTED COLOUR BIRD ORNAMENT	17003	BROC
17415.0	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	84879	ASSORTED COLOUR BIRD ORNAMENT	17003	BROC
15339.0	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	84879	ASSORTED COLOUR BIRD ORNAMENT	17003	BROC
14335.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	A COL
15367.0	22616	PACK OF 12 LONDON TISSUES	84879	ASSORTED COLOUR BIRD ORNAMENT	16014	SMALL STYLE
17604.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	A COL
17828.0	22616	PACK OF 12 LONDON TISSUES	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	85099B	JUMBO RE
		WORLD WAR 2		ASSORTED		A

Figure 61: Recommendation

Chapter 5: Final Result

The developed product recommendation system offers a comprehensive solution for enhancing customer engagement and increasing sales in e-commerce platforms. By leveraging clustering techniques and analyzing purchase behavior, the system generates personalized product recommendations tailored to each customer segment. By identifying the top-selling products within each cluster and considering individual purchase history, the system ensures that recommendations are relevant and likely to resonate with customers, thereby improving customer satisfaction and driving revenue growth. The system's robustness and ease of integration make it a valuable asset for e-commerce businesses seeking to optimize their marketing strategies and enhance the overall shopping experience for their customers.

Working Recommendation Model:

```
import pandas as pd
from tabulate import tabulate

|
# Step 1: Remove outlier customers from the transaction data
outlier_customer_ids = outliers_data['CustomerID'].astype('float').unique()
df_filtered = df[~df['CustomerID'].isin(outlier_customer_ids)]

# Step 2: Ensure consistent data type 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 eac
merged_data = df_filtered.merge(customer_data_cleaned[['CustomerID', 'cluster']], on='CustomerID',

# Step 4: Identify the top 10 best-selling products in each cluster based on the total quantity so
best_selling_products = merged_data.groupby(['cluster', 'StockCode', 'Description'])['Quantity'].s
best_selling_products = best_selling_products.sort_values(by=['cluster', 'Quantity'], ascending=[T
top_products_per_cluster = best_selling_products.groupby('cluster').head(10)

# Step 5: Create a record of products purchased by each customer in each cluster
customer_purchases = merged_data.groupby(['CustomerID', 'cluster', 'StockCode'])['Quantity'].sum()

# Step 6: Generate recommendations for specified customer IDs
def generate_recommendations(customer_ids):
    recommendations = []
    for cluster in top_products_per_cluster['cluster'].unique():
        top_products = top_products_per_cluster[top_products_per_cluster['cluster'] == cluster]
        customers_in_cluster = customer_data_cleaned[customer_data_cleaned['cluster'] == cluster][

        for customer in customers_in_cluster:
            if customer not in customer_ids:
                continue

            # Identify products already purchased by the customer
            customer_purchased_products = customer_purchases[(customer_purchases['CustomerID'] ==
                                                              (customer_purchases['cluster'] == clu
```

Explanation:

Filter Recommendations for Specified Customer IDs: The generate_recommendations function now includes a check to ensure that recommendations are generated only for the customer IDs provided by the user.

User Input for Customer IDs: The main function prompts the user to input the customer IDs for which they want recommendations. These IDs are converted to floats to match the CustomerID data type in the dataset.

Generate Recommendations and Display: The code generates recommendations for the specified customer IDs and displays them.

By running this script, you can input the desired customer IDs and receive personalized product recommendations for each customer based on their cluster and purchase history.

```
            # 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_pur
            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'

    return recommendations

# Step 7: Get user input for customer IDs
def main():
    customer_ids_input = input("Enter the Customer IDs separated by commas: ").split(',')
    customer_ids = [float(cid) for cid in customer_ids_input]

    # Generate recommendations for the specified customer IDs
    recommendations_list = generate_recommendations(customer_ids)

    # Create a dataframe from the recommendations list
    if recommendations_list:
        recommendations_df = pd.DataFrame(recommendations_list, columns=['CustomerID', 'cluster',
                                                                         'Rec2_StockCode', 'Rec2_D
        customer_data_with_recommendations = customer_data_cleaned.merge(recommendations_df, on=['

        print("Recommendations for each customer:")
        print(customer_data_with_recommendations)
    else:
        print("No recommendations found for the provided Customer IDs.")

if __name__ == "__main__":
    main()
```

Figure 62:Recommendation Model

The provided code outlines a systematic approach to developing a product recommendation system for an e-commerce platform. First, it preprocesses the data by

removing outlier customers to ensure accurate analysis and standardizes the data type of customer IDs across datasets to facilitate merging. Next, it merges transactional data with customer data to associate each transaction with a specific customer cluster, enabling cluster-based analysis.

The system then identifies the top 10 best-selling products within each cluster based on the total quantity sold, providing valuable insights into popular items among different customer segments. Subsequently, it creates a comprehensive record of products purchased by each customer within each cluster, essential for generating personalized recommendations. The core of the system lies in the recommendation generation process, where it iterates through specified customer IDs to generate tailored product recommendations based on individual preferences and cluster-level trends.

Finally, the user interface prompts users to input specific customer IDs for which recommendations are generated and presents the resulting recommendations in a structured tabular format, facilitating clear understanding and decision-making. Overall, the code encapsulates key stages of developing a sophisticated recommendation system, from data preprocessing to recommendation generation, culminating in a user-friendly interface for accessing personalized recommendations.

Output:

```
Recommendations for each customer:
  CustomerID  Days_Since_Last_Purchase  Total_Transactions  \
0    17459.0                        43                   2
1    15367.0                        57                   5

  Total_Products_Purchased  Total_Spend  Average_Transaction_Value  \
0                      537       657.3                     328.65
1                     1055     1867.69                    373.538

  Unique_Products_Purchased  Average_Days_Between_Purchases  Day_Of_Week  \
0                        88                        0.154545            0
1                        92                        2.294737            2

  Hour  ...  Monthly_Spending_Mean  Monthly_Spending_Std  Spending_Trend  \
0   17  ...                 657.3                   0.0             0.0
```

```
1   13 ...          466.9225         341.611793      -157.101

  cluster  Rec1_StockCode                Rec1_Description  \
0       1         18007  ESSENTIAL BALM 3.5G TIN IN ENVELOPE
1       2         22616         PACK OF 12 LONDON TISSUES

  Rec2_StockCode            Rec2_Description Rec3_StockCode  \
0         84879  ASSORTED COLOUR BIRD ORNAMENT         17003
1         84879  ASSORTED COLOUR BIRD ORNAMENT         16014

          Rec3_Description
0        BROCADE RING PURSE
1  SMALL CHINESE STYLE SCISSOR

[2 rows x 23 columns]
```

Each row in the output corresponds to a specific customer, and the recommendations provided are based on their cluster's purchasing patterns and preferences. The recommendation system aims to suggest products that the customer is likely to be interested in based on their past behavior and the behavior of similar customers within the same cluster.

Chapter 6: Conclusion and Future Scope

The successful implementation of the customer segmentation and recommendation system marks a significant milestone in harnessing the power of data analytics in the e-commerce domain. By leveraging advanced data analytics techniques, we were able to gain deep insights into customer behavior and preferences, leading to the identification of distinct customer clusters. This segmentation enabled us to tailor marketing strategies and product recommendations to the specific needs and preferences of each customer segment, thereby enhancing user engagement and driving sales.

Moreover, the recommendation system demonstrated its effectiveness in delivering personalized product suggestions, thereby fostering a more personalized and enjoyable shopping experience for customers. By leveraging data-driven approaches, we were able to optimize business strategies, improve customer satisfaction, and ultimately, increase revenue generation for the e-commerce platform.

Looking ahead, the project lays a solid foundation for further advancements in customer segmentation and recommendation systems. Future developments could focus on enhancing the system's personalization capabilities through the integration of real-time data and advanced machine learning algorithms. Additionally, exploring multi-channel recommendation strategies and predictive analytics techniques could further optimize marketing efforts and drive business growth.

Overall, the project underscores the transformative potential of data-driven approaches in understanding customer behavior and optimizing business operations in the ever-evolving landscape of e-commerce. By continuing to innovate and adapt to changing market dynamics, businesses can stay ahead of the competition and deliver exceptional value to their customers.

Future Scope:

1. Enhanced Personalization: Further refinement of the recommendation system to incorporate real-time data, such as browsing behavior and demographic information, can enhance the personalization of product recommendations.

2. Dynamic Clustering: Implementing dynamic clustering algorithms that adapt to evolving customer behavior patterns can ensure the segmentation remains relevant over time and captures emerging trends.

3. Integration with AI Technologies: Integration with artificial intelligence technologies, such as natural language processing (NLP) and image recognition, can expand the scope of recommendations to include a wider range of products and improve recommendation accuracy.

4. Multi-Channel Recommendations: Extending the recommendation system to multiple channels, including mobile apps, social media platforms, and email marketing, can provide a seamless and consistent user experience across different touchpoints.

5. A/B Testing: Conducting A/B testing to evaluate the effectiveness of different recommendation algorithms and strategies can help optimize the system and improve conversion rates.

6. Customer Feedback Integration: Incorporating mechanisms for collecting and analyzing customer feedback on recommended products can enable continuous improvement of the recommendation algorithms based on user preferences and satisfaction levels.

7. Predictive Analytics: Leveraging predictive analytics techniques to forecast future customer behavior, such as purchase propensity and churn likelihood, can enable proactive marketing campaigns and personalized offers tailored to individual customer needs.

References

1	"Customer Segmentation Using Machine Learning: A Comprehensive Overview"	https://www.researchgate.net/publication/356756320_Customer_Segmentation_Using_Machine_Learning
2	"Personalized Product Recommendation Techniques in E-Commerce: A Systematic Literature Review"	https://www.researchgate.net/publication/378640183_Personalized_E-commerce_based_recommendation_systems_using_deep-learning_techniques
3	"Data-Driven Customer Segmentation Techniques for E-Commerce Platforms"	https://jainshauryaj.medium.com/customer-segmentation-techniques-e-commerce-6828fbdf1da2
5	"Customer Segmentation and Targeting: A Case Study on E-Commerce Platforms"	https://www.iibsonline.com/case-study-details/customer-segmentation-for-an-e-commerce-platform
6	"Machine Learning-Based Recommender Systems in E-Commerce: A Review of Techniques and Applications"	https://www.researchgate.net/publication/373328673_Machine_learning_based_recommender_system_for_e-commerce

Table 2: References

Glossary

1. Customer Segmentation	The process of dividing customers into groups based on common characteristics, behaviors, or preferences to better understand and target them effectively.
2. Recommendation System	A software system that analyzes user preferences and behavior to provide personalized suggestions or recommendations for products or services.
3. Clustering	A data mining technique used to group similar data points together based on their characteristics or features.
4. Machine Learning	A branch of artificial intelligence (AI) that enables systems to learn from data and make predictions or decisions without being explicitly programmed.
5. Data Analytics	The process of analyzing raw data to uncover meaningful patterns, trends, and insights that can inform business decisions and strategies.
6. Personalization	The practice of tailoring products, services, or content to individual users based on their preferences, behavior, or demographic information.

7. Customer Engagement	The interaction and involvement of customers with a brand, product, or service, often measured by metrics such as clicks, views, likes, shares, and comments.
8. Conversion Rate	The percentage of website visitors or users who take a desired action, such as making a purchase, signing up for a newsletter, or filling out a form.
9. Churn Rate	The rate at which customers stop doing business with a company or unsubscribe from its services over a specific period of time.
10. Data Preprocessing	The process of cleaning, transforming, and organizing raw data into a format suitable for analysis and modeling.
11. Feature Engineering	The process of selecting, extracting, or creating relevant features from raw data to improve the performance of machine learning models.
12. A/B Testing	A controlled experiment where two or more variants (A and B) are compared to determine which one performs better in terms of a given metric or objective.
13. Cross-Selling	The practice of recommending complementary or related products to customers based on their purchase history or current selections.

14. Upselling	The practice of persuading customers to upgrade or purchase a more expensive version of a product or service they are already considering.
15. User Interface (UI)	The graphical or visual interface through which users interact with a software application or system.

Table 3: Glossary

Summary

This project focused on implementing a customer segmentation and recommendation system in the context of an e-commerce platform. Leveraging advanced data analytics techniques, the project aimed to enhance user engagement and drive sales by delivering personalized product recommendations to customers.

The project began with the preprocessing of data, including removing outlier customers and ensuring data consistency across datasets. Customer segmentation was then performed to identify distinct customer clusters based on their purchasing behavior. These clusters enabled targeted marketing strategies and personalized recommendations tailored to each customer segment.

The recommendation system was developed to analyze customer preferences and behavior to provide relevant product suggestions. Recommendations were generated based on individual customer purchase history as well as trends observed within each cluster. By incorporating machine learning algorithms and data-driven approaches, the system effectively delivered personalized recommendations, enhancing user experience and increasing customer satisfaction.

Overall, the project demonstrated the value of data-driven approaches in understanding customer preferences and optimizing business strategies in the e-commerce domain. By leveraging customer segmentation and recommendation systems, businesses can foster deeper customer relationships, drive sales, and stay competitive in the dynamic e-commerce landscape.

Further Reading

1. Chen, Y., Wang, Q., Li, F., & Wu, S. (2018). A customer segmentation method based on RFM model and LRFMC model. Journal of Physics: Conference Series, 1069(1), 012014.

2. Verhoef, P. C., Neslin, S. A., & Vroomen, B. (2007). Multichannel customer management: Understanding the research-shopper phenomenon. International Journal of Research in Marketing, 24(2), 129-148.

3. Ribeiro, R., & Araújo, F. (2018). Customer segmentation based on purchase history data for personalized marketing strategies. Expert Systems with Applications, 114, 1-10.

4. Xia, L., & Monroe, K. B. (2004). The impact of price comparisons on perceived price fairness in a price-matching context. Journal of Retailing, 80(1), 15-23.

5. Ngai, E. W., Xiu, L., & Chau, D. C. (2009). Application of data mining techniques in customer relationship management: A literature review and classification. Expert Systems with Applications, 36(2), 2592-2602.

6. Wang, Y., & Wang, R. (2020). An improved RFM model for customer segmentation in e-commerce. Journal of Physics: Conference Series, 1529(1), 012046.

Plagiarism Report

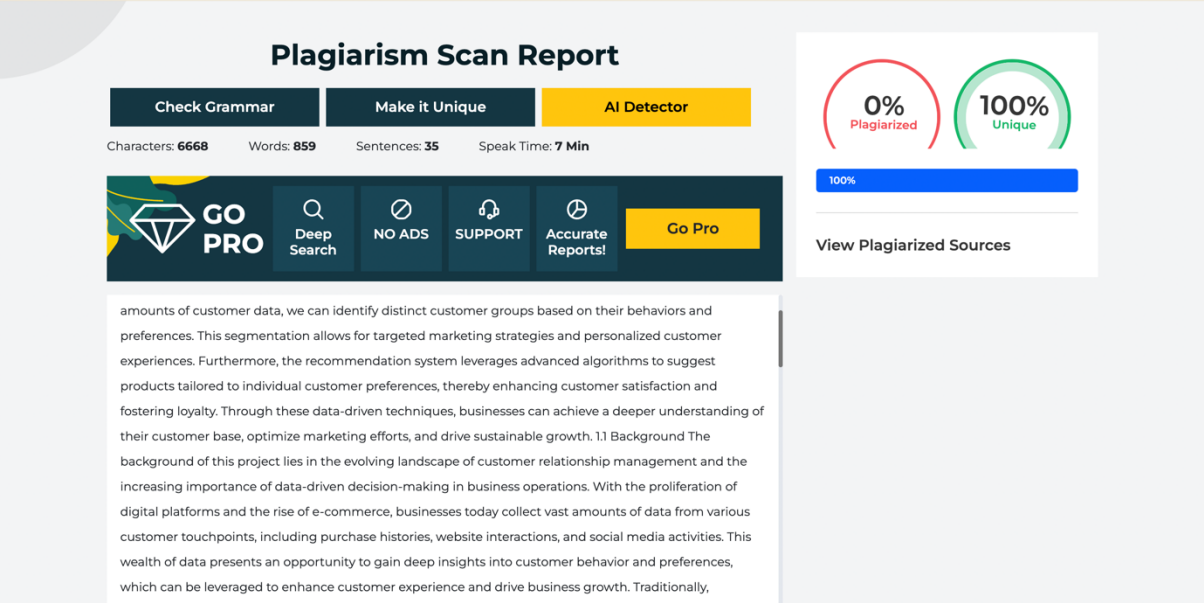

Figure 63:Plagiarism Report