Customer Segmentation & Recommendation System

By Vaibhav Rai

What is customer segmentation?

Customer segmentation is the process by which you divide your customers up based on common characteristics – such as demographics or behaviours, so your marketing team or sales team can reach out to those customers more effectively.

What is Recommendation System?

A recommendation system is a technology that suggests items or content to users based on their preferences, past behavior, and similarities with other users. It's like having a knowledgeable friend who knows your tastes and can suggest movies, books, or products you might enjoy. These systems use algorithms to analyze data and make personalized recommendations, helping users discover new things they might like without having to search for them manually.

Dataset Description

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

Problem

In this Project, we go deep into the growing field of online retail by examining a transactional dataset from a UK-based business. This dataset includes all transactions between 2010 and 2011. Our primary goal is to improve the effectiveness of marketing tactics and increase sales through consumer segmentation. We intend to transform transactional data into a customer-centric dataset by developing new features that will aid in the segmentation of customers into various groups using the K-means clustering technique. This segmentation will help us grasp the unique profiles and preferences of various client groups. Building on this, we want to construct a recommendation system that will suggest top-selling products to customers within each sector who haven't purchased those items yet, thereby improving marketing efficacy and fostering increased sales

Objectives:

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

Data Cleaning & Transformation

This step includes a thorough cleaning and processing process to improve the dataset. It involves addressing missing values, eliminating duplicate entries, correcting anomalies in product codes and descriptions, and making other necessary adjustments to prepare the data for in-depth analysis and modeling.

- Handling Missing Values
- Handling Duplicates
- Treating Cancelled Transactions
- Correcting StockCode Anomalies
- Outlier Treatment

Data Cleaning & Transformation

Handling Missing Values

```
# Extracting rows with missing values in 'CustomerID' or 'Description' columns
df[df['CustomerID'].isnull() | df['Description'].isnull()].head(2)
```

	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

CustomerID = 24.93% and Description = 0.27%

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

```
# Verifying the removal of missing values
df.isnull().sum().sum()
```

```
0
```

Handling Duplicates

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

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

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

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

```
401604
```

Treating Cancelled Transactions

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

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

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

Correcting StockCode Anomalies

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

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

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

```
In [181]: # Removing rows with anomalous stock codes from the dataset
          df = df[~df['StockCode'].isin(anomalous_stock_codes)]
```

```
In [182]: # Getting the number of rows in the dataframe
          df.shape[0]

Out[182]: 399689
```

Outlier Treatment

```
In [210]: # Separate the outliers for analysis
          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)
```

```
In [211]: # Getting the number of rows in the cleaned customer dataset
          customer_data_cleaned.shape[0]

Out[211]: 4067
```

Feature Engineering

In this Step to generate a comprehensive customer-centric dataset for clustering and recommendation, the following attributes can be engineered from the existing data:

- RFM Features
  - Recency (R)
  - Frequency (F)
  - Monetary (M)
- Product Diversity
- Behavioral Features
- Geographic Features
- Cancellation Insights
- Seasonality & Trends

Feature Engineering

Recency (R)

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

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

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

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

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

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

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

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

Frequency (F)

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

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

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

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

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

Monetary (M)

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

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

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

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

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

Product Diversity

```
# Calculate the number of unique products purchased by each customer
unique_products_purchased = df.groupby('CustomerID')['StockCode'].nunique().reset_index()
unique_products_purchased.rename(columns={'StockCode': 'Unique_Products_Purchased'}, inplace=True)

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

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

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.

Behavioral Features

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

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

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

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

# 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()
```

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

Seasonality & Trends

In [203]:
```
# Extract month and year from InvoiceDate
df['Year'] = df['InvoiceDate'].dt.year
df['Month'] = df['InvoiceDate'].dt.month

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

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

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

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

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

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

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

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.

Geographic Features

```
# Group by CustomerID and Country to get the number of transactions per country for each customer
customer_country = df.groupby(['CustomerID', 'Country']).size().reset_index(name='Number_of_Transactions')

# Get the country with the maximum number of transactions for each customer (in case a customer has transactions from multiple co
customer_main_country = customer_country.sort_values('Number_of_Transactions', ascending=False).drop_duplicates('CustomerID')

# 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 == 'United Kingdom' else 0)

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

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

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

Cancellation Insights

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

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

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

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

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

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

I am going to delve deeper into the cancellation patterns of customers to gain insights that can enhance our customer segmentation model.

Customer Dataset Description

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

Data Preprocessing

Use feature scaling and dimensionality reduction to simplify the data and improve clustering efficiency.

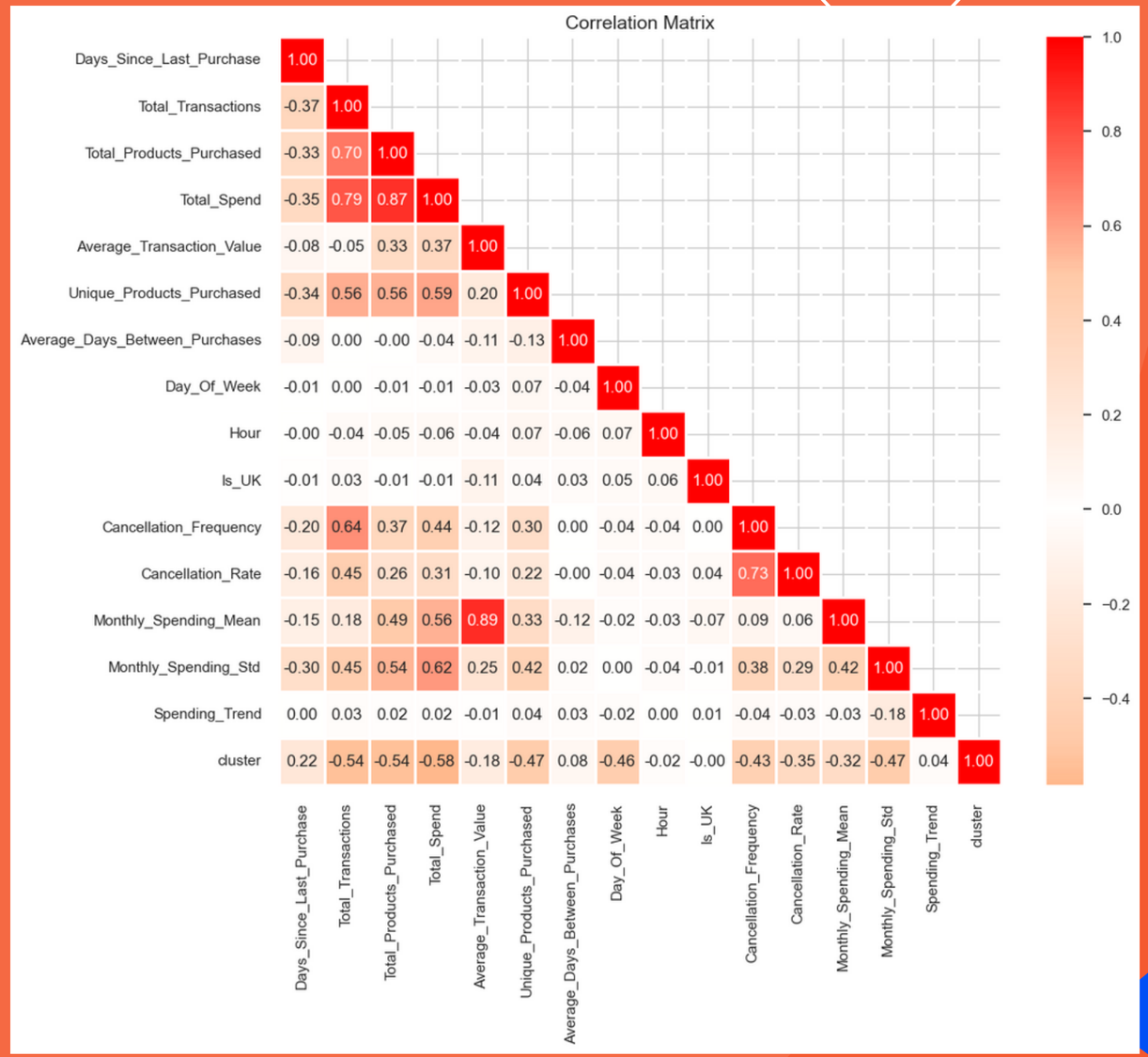
- Correlation Analysis
- Feature Scaling
- Dimensionality Reduction

Correlation Analysis

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

- Monthly_Spending_Mean and Average_Transaction_Value
- Total_Spend and Total_Products_Purchased
- Total_Transactions and Total_Spend
- Cancellation_Rate and Cancellation_Frequency
- Total_Transactions and Total_Products_Purchased

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



Scaling

Before we begin clustering and dimensionality reduction, we must scale our features. This phase is critical, particularly in the context of distance-based algorithms such as K-means and dimensionality reduction methods like PCA.

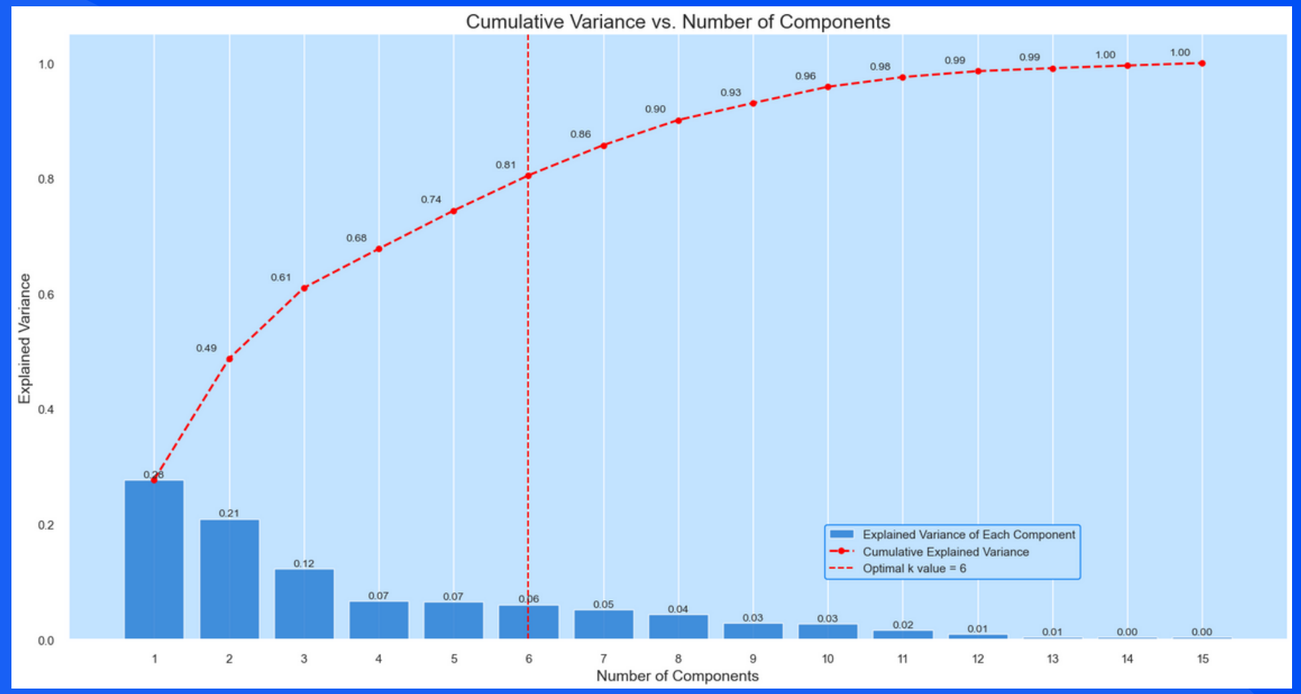
Therefore, to ensure a balanced influence on the model and to reveal the true patterns in the data, I am going to standardize our data, meaning transforming the features to have a mean of 0 and a standard deviation of 1.

```
# Initialize the StandardScaler
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_scale])

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

Dimensionality Reduction

I will apply PCA on all the available components and plot the cumulative variance explained by them. This process will allow me to visualize how much variance each additional principal component can explain, thereby helping me to pinpoint the optimal number of components to retain for the analysis

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.

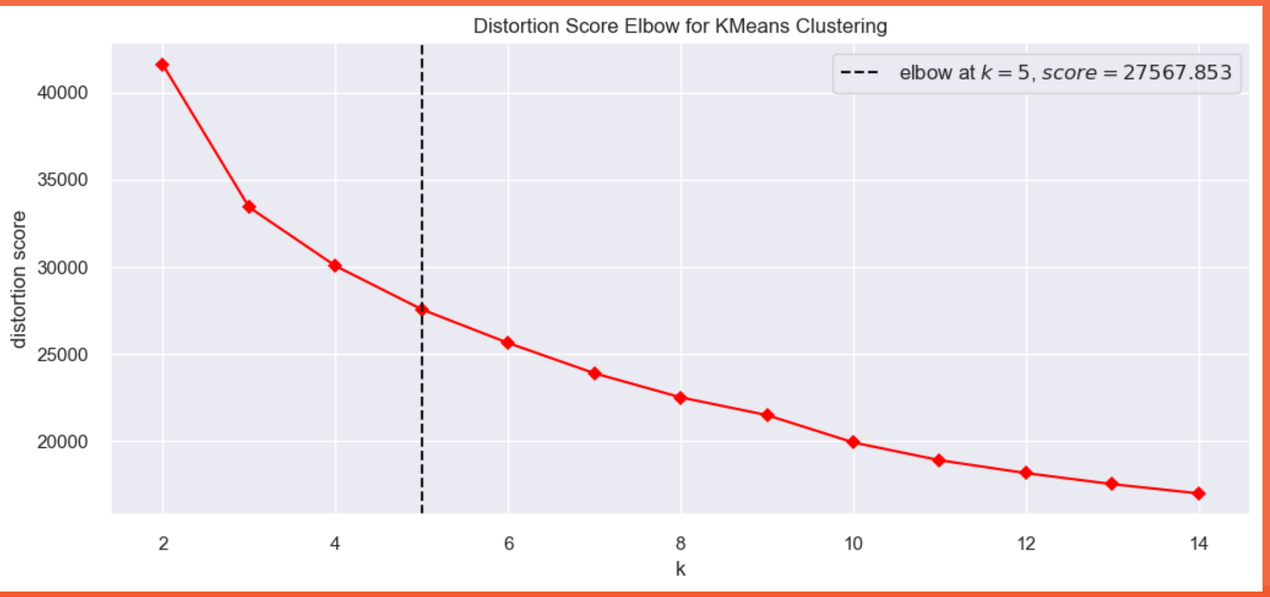
Customer Segmentation using K-Means Clustering

K-Means is an unsupervised machine learning algorithm that divides data into a set number of groups (K) by minimizing the within-cluster sum-of-squares (WCSS), also called inertia. The algorithm iteratively allocates each data point to the nearest centroid before updating the centroids by finding the mean of all allocated points. The process continues until convergence or a stopping requirement is met.

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

- Elbow Method
- Silhouette Method

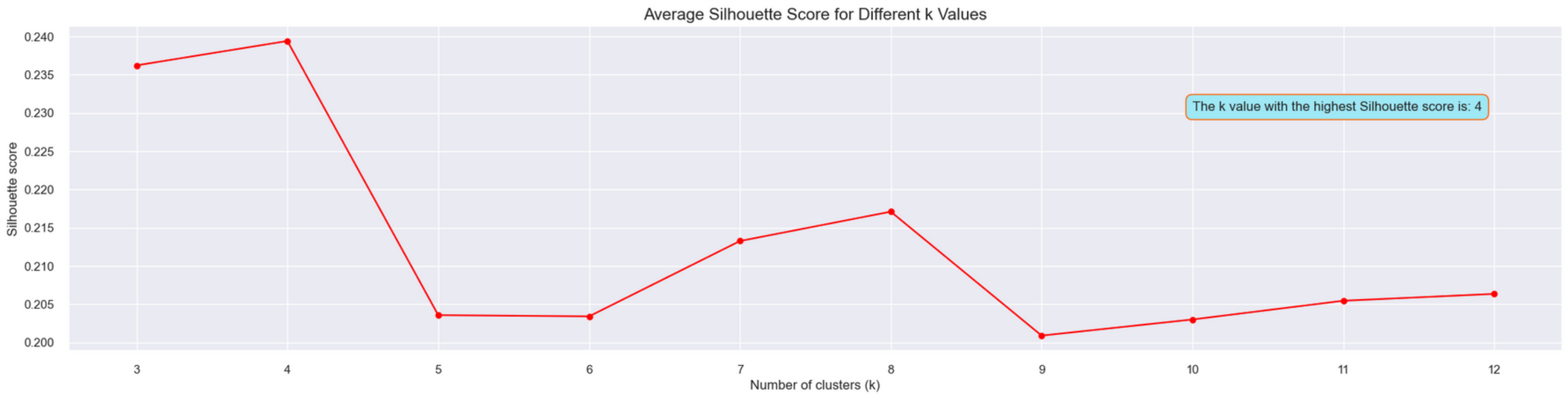
It's common to utilize both methods in practice to corroborate the results.



Elbow Method

The elbow point represents the ideal value of k for the KMeans clustering algorithm. Using the YellowBrick library for the Elbow technique, we find that the best k number is 5. However, there is no clear elbow point in this situation, which is common in real-world data. The plot shows that the inertia decreases dramatically up to k=5, indicating that the optimal value of k is between 3 and 7. To select the best k within this range, we can use silhouette analysis, which is another cluster quality evaluation technique. Furthermore, using business knowledge can aid in calculating a practical k value.

Silhouette Method



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

Cluster Analysis & Evaluation

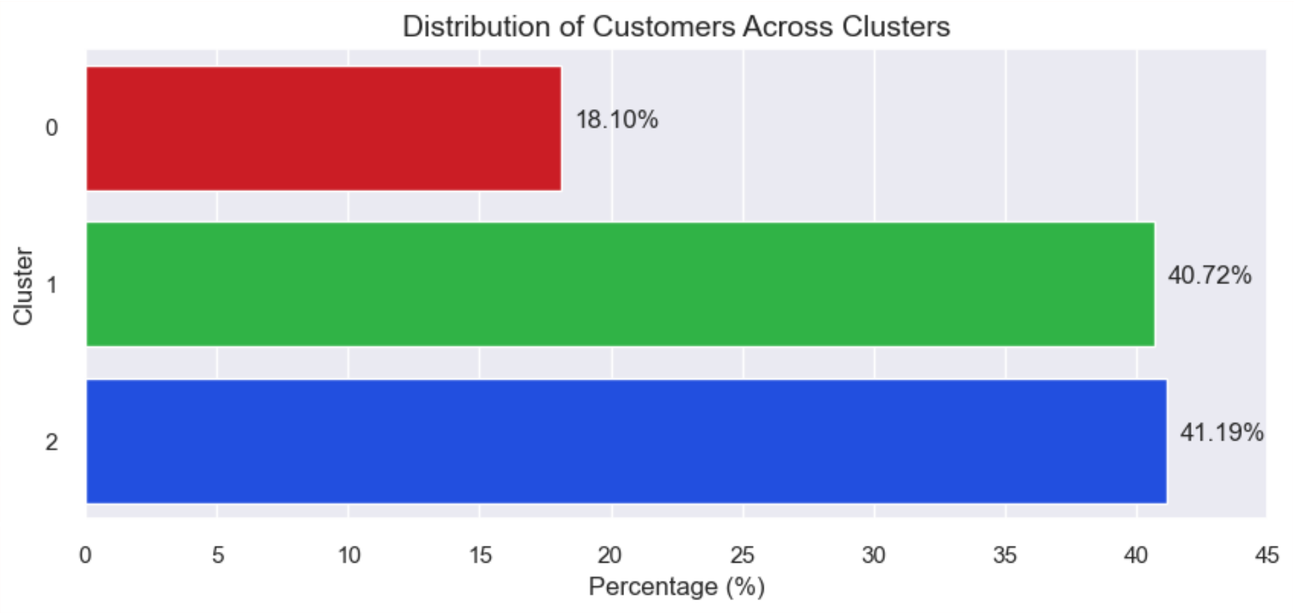
After determining the optimal number of clusters (which is 3 in our case) using elbow and silhouette analyses, I move onto the evaluation step to assess the quality of the clusters formed. This step is essential to validate the effectiveness of the clustering and to ensure that the clusters are coherent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

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:

Cluster Distribution Visualization

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

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

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



Evaluation Metrics

Metric	Value
Number of Observations	4067
Silhouette Score	0.23622848017098863
Calinski Harabasz Score	1257.1747766540632
Davies Bouldin Score	1.3682695376074667

- **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.**

Cluster 0 - Casual Weekend Shoppers:

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

Cluster 1 - Occasional Big Spenders:

- Customers in this cluster don't shop frequently but tend to spend a considerable amount when they do,buying a variety of products.
- Their spending has been on the rise, indicating a growing interest or investment in their purchases.
- They prefer to shop later in the day, possibly after work hours, and are mainly based in the UK.
- They have a moderate tendency to cancel transactions, which might be due to their higher spending; they perhaps reconsider their purchases more often.
- Their purchases are generally substantial, indicating a preference for quality or premium products.

Cluster 2 - Eager Early-Bird Shoppers:

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

Recommendation System

In the final phase of this project, I am set to develop a recommendation system to enhance the online shopping experience. This system will suggest products to customers based on the purchasing patterns prevalent in their respective clusters.

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.

Snapshot of Recommendation System

Out[233]:

CustomerID	Rec1_StockCode	Rec1_Description	Rec2_StockCode	Rec2_Description	Rec3_StockCode	Rec3_Description
13243.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	ASSORTED COLOURS SILK FAN
13232.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	ASSORTED COLOURS SILK FAN
14997.0	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	84879	ASSORTED COLOUR BIRD ORNAMENT	17003	BROCADE RING PURSE
14948.0	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	84879	ASSORTED COLOUR BIRD ORNAMENT	17003	BROCADE RING PURSE
12596.0	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	84879	ASSORTED COLOUR BIRD ORNAMENT	17003	BROCADE RING PURSE
16686.0	22616	PACK OF 12 LONDON TISSUES	85099B	JUMBO BAG RED RETROSPOT	84879	ASSORTED COLOUR BIRD ORNAMENT
17101.0	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	84879	ASSORTED COLOUR BIRD ORNAMENT	17003	BROCADE RING PURSE
14954.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSORTED COLOUR BIRD ORNAMENT	15036	ASSORTED COLOURS SILK FAN
18123.0	18007	ESSENTIAL BALM 3.5G TIN IN ENVELOPE	84879	ASSORTED COLOUR BIRD ORNAMENT	17003	BROCADE RING PURSE
14051.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	85099B	JUMBO BAG RED RETROSPOT	84879	ASSORTED COLOUR BIRD ORNAMENT

Thank You