Lab 3: Customer Segmentation

Machine Learning - MGT 665

Prof. Itauma

Group Members

- Prathyusha Devari
- Abdul Rehman
- Sai Srujana Jakkala
- Nirav

Introduction

For the project, we choose a Amazon customer data for analysis. Our task is to identify distinct customer groups based on their buying behaviors. Once segments are identified, analyze these groups to recommend specific marketing strategies tailored to each segment's characteristics.

Customer segmentation plays a vital role in helping businesses tailor their marketing strategies, promotions, and product offerings to distinct customer groups based on behaviors such as purchase frequency and spending patterns. Clustering techniques provide a data-driven way to identify such groups. In this report, we apply various clustering methods—K-Means, Hierarchical Clustering, DBSCAN, Agglomerative Clustering, and PCA (for visualization)—to a customer dataset. Each method brings unique strengths to segmenting customers and uncovering actionable insights.

Import necessary Libraries

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import combinations
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from scipy.stats import chi2_contingency
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

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

Dataset Overview

- age= age
- gender= gender

- Purchase_Frequency= How frequently do you make purchases on Amazon?
- Purchase_Categories= What product categories do you typically purchase on Amazon?
- Personalized_Recommendation_Frequency = Have you ever made a purchase based on personalized product recommendations from Amazon?
- Browsing_Frequency =How often do you browse Amazon's website or app?
- Product_Search_Method =How do you search for products on Amazon?
- Search_Result_Exploration =Do you tend to explore multiple pages of search results or focus on the first page?
- Customer_Reviews_Importance =How important are customer reviews in your decision-making process?
- Add_to_Cart_Browsing =Do you add products to your cart while browsing on Amazon?
- Cart_Completion_Frequency =How often do you complete the purchase after adding products to your cart?
- Cart_Abandonment_Factors =What factors influence your decision to abandon a purchase in your cart?
- Saveforlater_Frequency =Do you use Amazon's "Save for Later" feature, and if so, how often?
- Review_Left =Have you ever left a product review on Amazon?
- Review_Reliability =How much do you rely on product reviews when making a purchase?
- Review_Helpfulness =Do you find helpful information from other customers' reviews?
- Personalized_Recommendation_Frequency =How often do you receive personalized product recommendations from Amazon?
- Recommendation_Helpfulness =Do you find the recommendations helpful?
- Rating_Accuracy =How would you rate the relevance and accuracy of the recommendations you receive
- Shopping_Satisfaction =How satisfied are you with your overall shopping experience on Amazon?
- Service_Appreciation =What aspects of Amazon's services do you appreciate the most?
- Improvement_Areas =Are there any areas where you think Amazon can improve?

Reading the dataset

```
df = pd.read_csv('//Users//srujana//Downloads//Amazon Customer 
Behavior Survey.csv')

df.head()

                        Timestamp  age              Gender  \
0  2023/06/04 1:28:19 PM GMT+5:30   23              Female   
1  2023/06/04 2:30:44 PM GMT+5:30   23              Female   
2  2023/06/04 5:04:56 PM GMT+5:30   24   Prefer not to say   
3  2023/06/04 5:13:00 PM GMT+5:30   24              Female   
4  2023/06/04 5:28:06 PM GMT+5:30   22              Female   

       Purchase_Frequency                                   
Purchase_Categories  \
0       Few times a month                           Beauty and
```

```
Personal Care
1           Once a month                               Clothing and
Fashion
2       Few times a month    Groceries and Gourmet Food;Clothing and
Fashion
3           Once a month  Beauty and Personal Care;Clothing and
Fashion;...
4  Less than once a month      Beauty and Personal Care;Clothing and
Fashion

  Personalized_Recommendation_Frequency Browsing_Frequency  \
0                                   Yes   Few times a week
1                                   Yes  Few times a month
2                                    No  Few times a month
3                             Sometimes  Few times a month
4                                   Yes  Few times a month

  Product_Search_Method Search_Result_Exploration  \
0               Keyword            Multiple pages
1               Keyword            Multiple pages
2               Keyword            Multiple pages
3               Keyword                First page
4                Filter            Multiple pages

   Customer_Reviews_Importance  ... Saveforlater_Frequency Review_Left
\
0                            1  ...              Sometimes         Yes

1                            1  ...                 Rarely          No

2                            2  ...                 Rarely          No

3                            5  ...              Sometimes         Yes

4                            1  ...                 Rarely          No


  Review_Reliability Review_Helpfulness  \
0       Occasionally                Yes
1            Heavily                Yes
2       Occasionally                 No
3            Heavily                Yes
4            Heavily                Yes

  Personalized_Recommendation_Frequency  Recommendation_Helpfulness  \
0                                       2                        Yes
1                                       2                  Sometimes
2                                       4                         No
3                                       3                  Sometimes
4                                       4                        Yes
```

```
  Rating_Accuracy   Shopping_Satisfaction    Service_Appreciation  \
0                1                       1       Competitive prices
1                3                       2  Wide product selection
2                3                       3       Competitive prices
3                3                       4       Competitive prices
4                2                       2       Competitive prices

              Improvement_Areas
0      Reducing packaging waste
1      Reducing packaging waste
2  Product quality and accuracy
3  Product quality and accuracy
4  Product quality and accuracy

[5 rows x 23 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 602 entries, 0 to 601
Data columns (total 23 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Timestamp                               602 non-null    object
 1   age                                     602 non-null    int64
 2   Gender                                  602 non-null    object
 3   Purchase_Frequency                      602 non-null    object
 4   Purchase_Categories                     602 non-null    object
 5   Personalized_Recommendation_Frequency   602 non-null    object
 6   Browsing_Frequency                      602 non-null    object
 7   Product_Search_Method                   600 non-null    object
 8   Search_Result_Exploration               602 non-null    object
 9   Customer_Reviews_Importance             602 non-null    int64
 10  Add_to_Cart_Browsing                    602 non-null    object
 11  Cart_Completion_Frequency               602 non-null    object
 12  Cart_Abandonment_Factors                602 non-null    object
 13  Saveforlater_Frequency                  602 non-null    object
 14  Review_Left                             602 non-null    object
 15  Review_Reliability                      602 non-null    object
 16  Review_Helpfulness                      602 non-null    object
 17  Personalized_Recommendation_Frequency   602 non-null    int64
 18  Recommendation_Helpfulness              602 non-null    object
 19  Rating_Accuracy                         602 non-null    int64
 20  Shopping_Satisfaction                   602 non-null    int64
 21  Service_Appreciation                    602 non-null    object
 22  Improvement_Areas                       602 non-null    object
dtypes: int64(5), object(18)
memory usage: 108.3+ KB
```

There are only 2 datatypes int and object

Null value check

```
df.isna().sum()
```

```
Timestamp                                 0
age                                       0
Gender                                    0
Purchase_Frequency                        0
Purchase_Categories                       0
Personalized_Recommendation_Frequency     0
Browsing_Frequency                        0
Product_Search_Method                     2
Search_Result_Exploration                 0
Customer_Reviews_Importance               0
Add_to_Cart_Browsing                      0
Cart_Completion_Frequency                 0
Cart_Abandonment_Factors                  0
Saveforlater_Frequency                    0
Review_Left                               0
Review_Reliability                        0
Review_Helpfulness                        0
Personalized_Recommendation_Frequency     0
Recommendation_Helpfulness                0
Rating_Accuracy                           0
Shopping_Satisfaction                     0
Service_Appreciation                      0
Improvement_Areas                         0
dtype: int64
```

Only the Product_Search_Method has null values and this could be ignored for our analysis

Catagorzie the numerical and catagorical data

```
# Numerical columns
numerical = df.select_dtypes(include=['int64'])

# Categorical columns
categorical = df.select_dtypes(include=['object'])

# Total numerical data

numerical
```

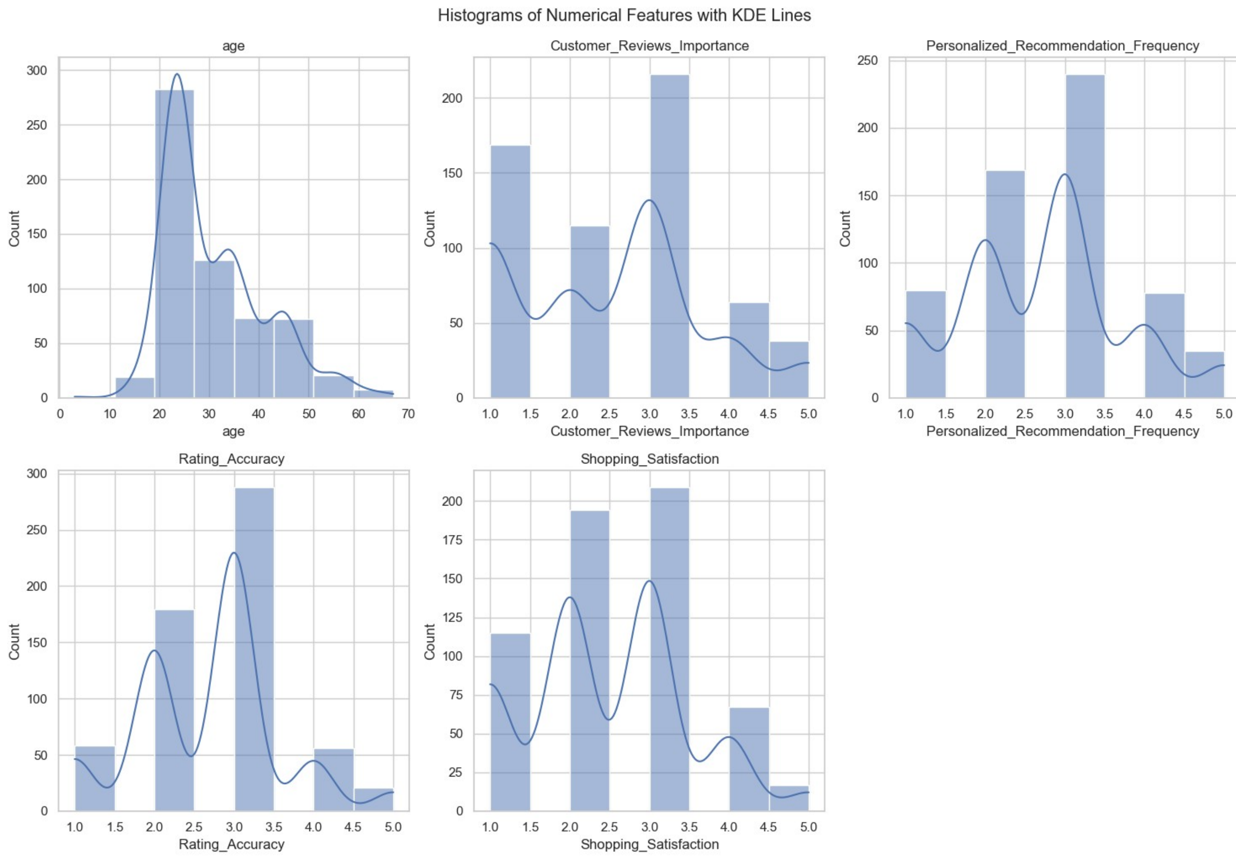
```
     age  Customer_Reviews_Importance
Personalized_Recommendation_Frequency    \
0     23                            1
2
1     23                            1
2
```

```
2     24                               2
4
3     24                               5
3
4     22                               1
4
..   ...                             ...
...
597   23                               4
3
598   23                               3
3
599   23                               3
3
600   23                               1
2
601   23                               3
3

     Rating_Accuracy   Shopping_Satisfaction
0                  1                       1
1                  3                       2
2                  3                       3
3                  3                       4
4                  2                       2
..               ...                     ...
597                3                       4
598                3                       3
599                2                       3
600                2                       2
601                3                       3

[602 rows x 5 columns]
```

Methodology

Exploratoty Data Analysis

```python
# Histograms with KDE lines
sns.set(style="whitegrid")
plt.figure(figsize=(15, 10))
for i, column in enumerate(numerical.columns, 1):
    plt.subplot(2, 3, i)
    sns.histplot(numerical[column], kde=True, bins=8)
    plt.title(f'{column}')
plt.tight_layout()
plt.suptitle('Histograms of Numerical Features with KDE Lines',
y=1.02)
plt.show()
```

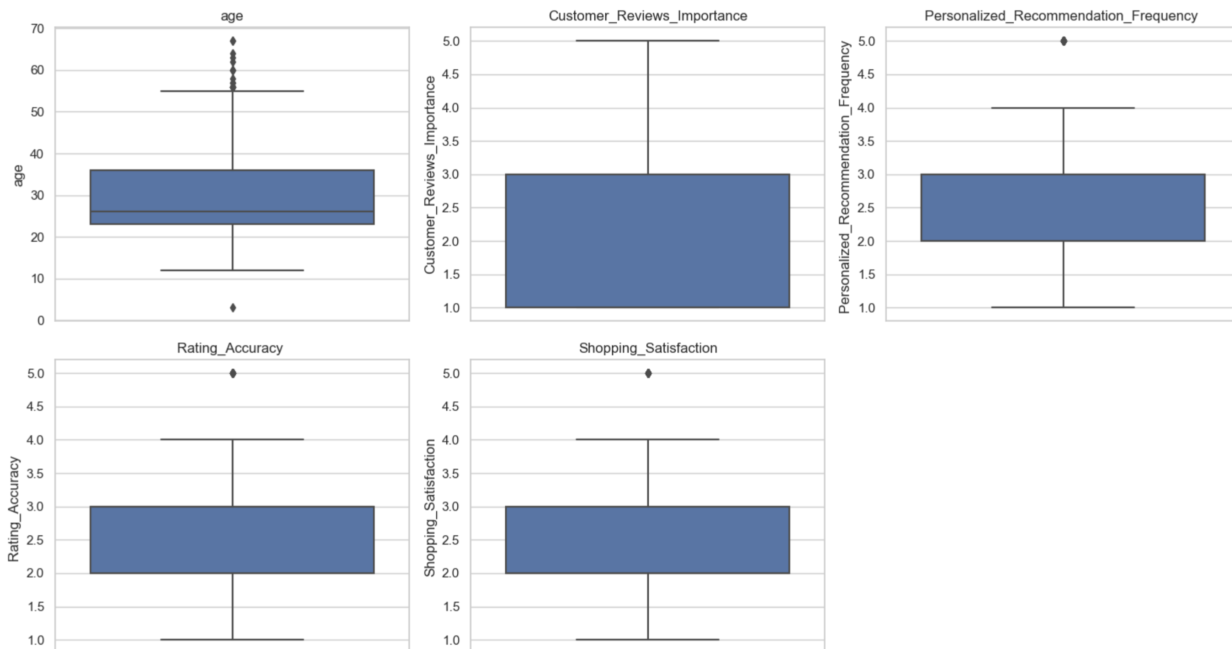


From the graphs:

- The age has a positive skew, it means that there are a few high-value outliers (besides that 3 on the left) on the right side
- Customer_Review_Importance has a significant concentration in number 1 and 3 turning the distribution not symmetric and reflecting the bimodal, thus causing the mean to drop in comparison to the rest where and. This suggests that there are two main groups of customers: group that does not find customer reviews important (rating 1) and group finds them moderately important (rating 3)
- Personalized_Recommendation_Frequency shows a distribution with a clear peak at rating 3, suggesting that most customers experience a moderate frequency of personalized recommendations and there is a slight left skew (negative skew), with a tail extending towards the lower ratings
- Rating_Accuracy shows a peak at rating 3, indicating that most customers rate the accuracy of ratings as neutral or moderate and there is a slight left skew (negative skew)
- Shopping_Satisfaction appears to have a peak at rating 3, with a significant number of responses also at rating 2 indicating that a number of customers are less satisfied and a pronounced concentration in the 3 first quarters

Outliers Check

```
# Box Plot
plt.figure(figsize=(15, 8))
for i, column in enumerate(numerical.columns, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(y=numerical[column])
    plt.title(f'{column}')
plt.tight_layout()
plt.show()
```

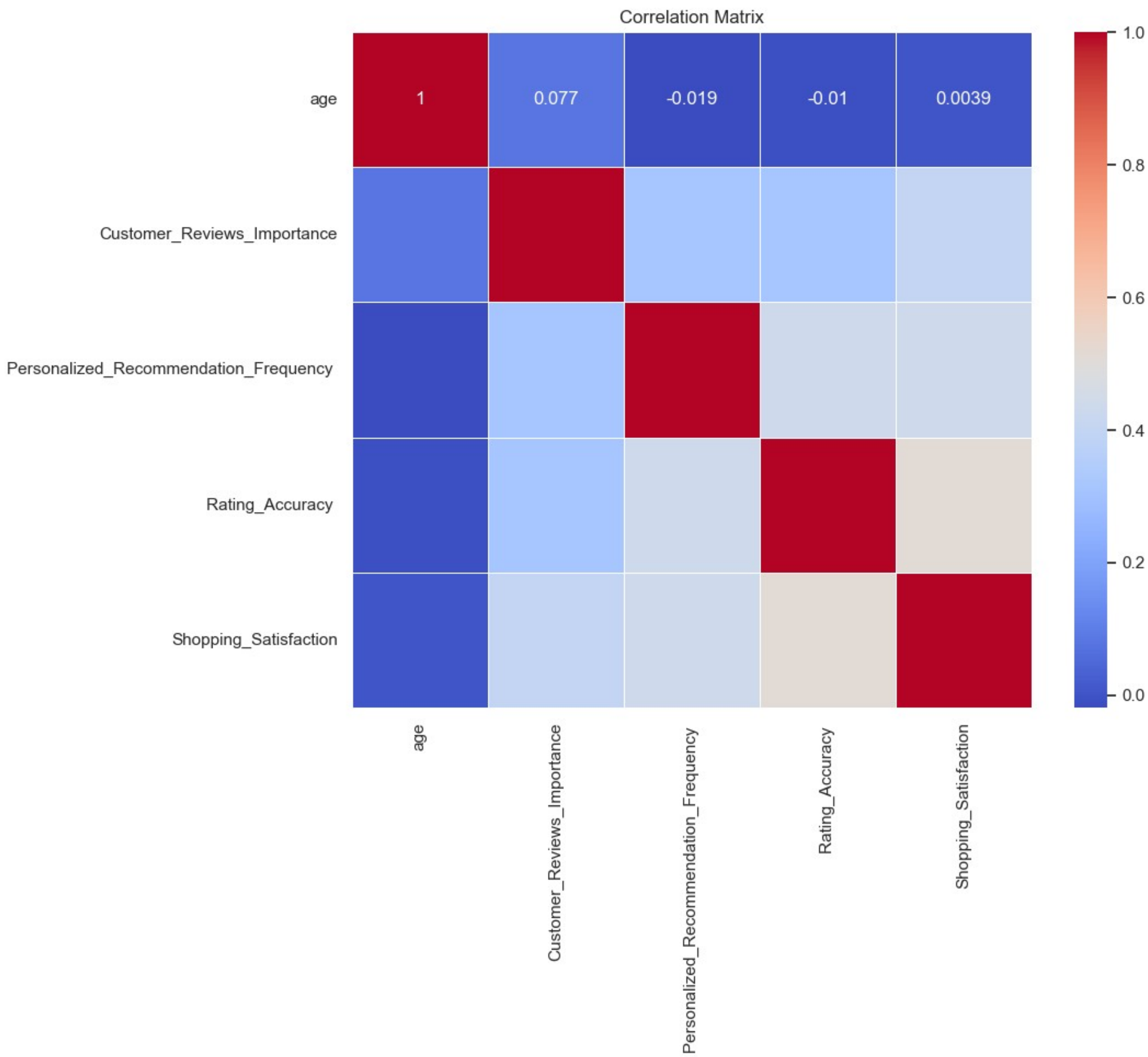


From the boxplots:

- age has this line indicating the median age (mid-20s), and there are several outliers on both the lower and upper ends, with more outliers on the upper end, indicating some customers are significantly older than the average.
- Customer_Reviews_Importance box is compact, indicating that the middle 50% of ratings are close together and suggesting that some customers rate the importance of customer reviews as significantly lower than the rest.
- Personalized_Recommendation_Frequency, Rating_Accuracy and Shopping_Satisfaction box appears to be very short, showing little variation in the middle 50% and has a few outliers on the upper end.

```
# Correlation Matrix
correlation_matrix = numerical.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
```

```
linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



EDA for Catagorical data

```
categorical
```

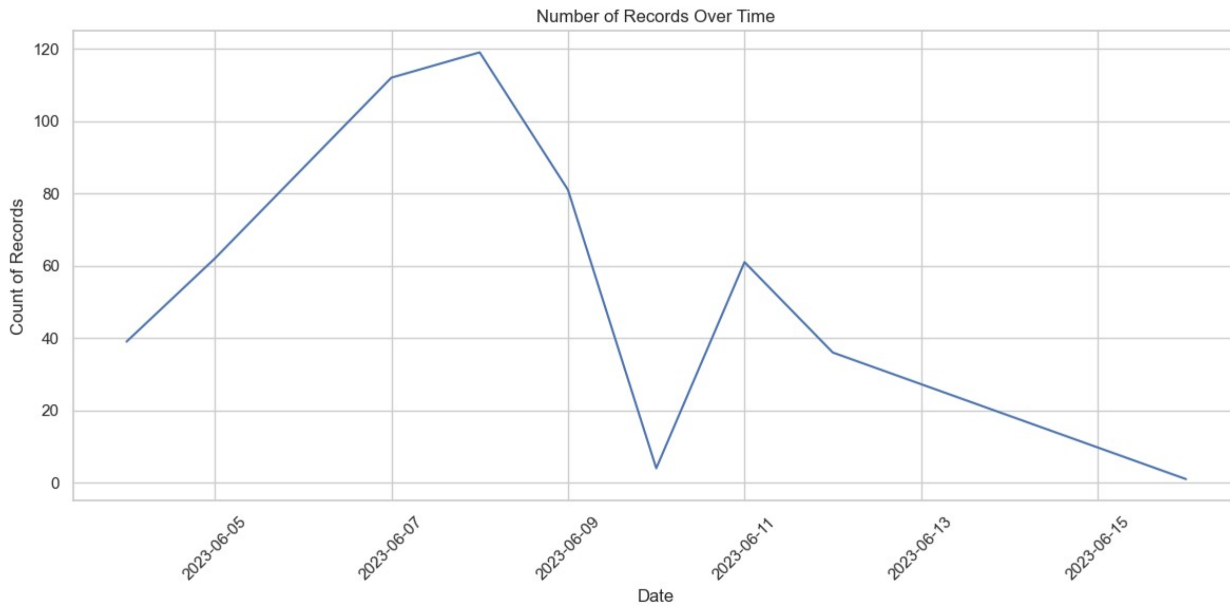
```
                         Timestamp             Gender  \
0    2023/06/04 1:28:19 PM GMT+5:30             Female
1    2023/06/04 2:30:44 PM GMT+5:30             Female
2    2023/06/04 5:04:56 PM GMT+5:30  Prefer not to say
3    2023/06/04 5:13:00 PM GMT+5:30             Female
4    2023/06/04 5:28:06 PM GMT+5:30             Female
..                             ...                ...
```

```
597  2023/06/12 4:02:02 PM GMT+5:30             Female
598  2023/06/12 4:02:53 PM GMT+5:30             Female
599  2023/06/12 4:03:59 PM GMT+5:30             Female
600  2023/06/12 9:57:20 PM GMT+5:30             Female
601  2023/06/16 9:16:05 AM GMT+5:30             Female

         Purchase_Frequency  \
0         Few times a month
1              Once a month
2         Few times a month
3              Once a month
4    Less than once a month
..                      ...
597             Once a week
598             Once a week
599            Once a month
600       Few times a month
601             Once a week

                                   Purchase_Categories  \
0                             Beauty and Personal Care
1                                 Clothing and Fashion
2       Groceries and Gourmet Food;Clothing and Fashion
3    Beauty and Personal Care;Clothing and Fashion;...
4        Beauty and Personal Care;Clothing and Fashion
..                                                 ...
597                           Beauty and Personal Care
598                               Clothing and Fashion
599                           Beauty and Personal Care
600  Beauty and Personal Care;Clothing and Fashion;...
601                               Clothing and Fashion

    Personalized_Recommendation_Frequency    Browsing_Frequency  \
0                                     Yes      Few times a week
1                                     Yes     Few times a month
2                                      No     Few times a month
3                               Sometimes     Few times a month
4                                     Yes     Few times a month
..                                    ...                   ...
597                             Sometimes      Few times a week
598                             Sometimes      Few times a week
599                             Sometimes      Few times a week
600                                   Yes     Few times a month
601                             Sometimes  Multiple times a day

    Product_Search_Method Search_Result_Exploration 
Add_to_Cart_Browsing  \
0                 Keyword            Multiple pages 
Yes
1                 Keyword            Multiple pages
```

```
Yes
2                  Keyword            Multiple pages
Yes
3                  Keyword                First page
Maybe
4                   Filter            Multiple pages
Yes
..                     ...                       ...                  .
..
597             categories            Multiple pages
Maybe
598                 Filter            Multiple pages
Maybe
599             categories            Multiple pages
Maybe
600                Keyword            Multiple pages
Yes
601                Keyword            Multiple pages
Maybe

    Cart_Completion_Frequency        Cart_Abandonment_Factors  \
0                   Sometimes  Found a better price elsewhere
1                       Often              High shipping costs
2                   Sometimes  Found a better price elsewhere
3                   Sometimes  Found a better price elsewhere
4                   Sometimes              High shipping costs
..                        ...                             ...
597                 Sometimes  Found a better price elsewhere
598                 Sometimes  Found a better price elsewhere
599                 Sometimes              High shipping costs
600                     Often                           others
601                     Often  Found a better price elsewhere

    Saveforlater_Frequency Review_Left Review_Reliability
Review_Helpfulness  \
0                Sometimes         Yes       Occasionally
Yes
1                   Rarely          No            Heavily
Yes
2                   Rarely          No       Occasionally
No
3                Sometimes         Yes            Heavily
Yes
4                   Rarely          No            Heavily
Yes
..                     ...         ...                ...
...
597              Sometimes         Yes         Moderately
Sometimes
```

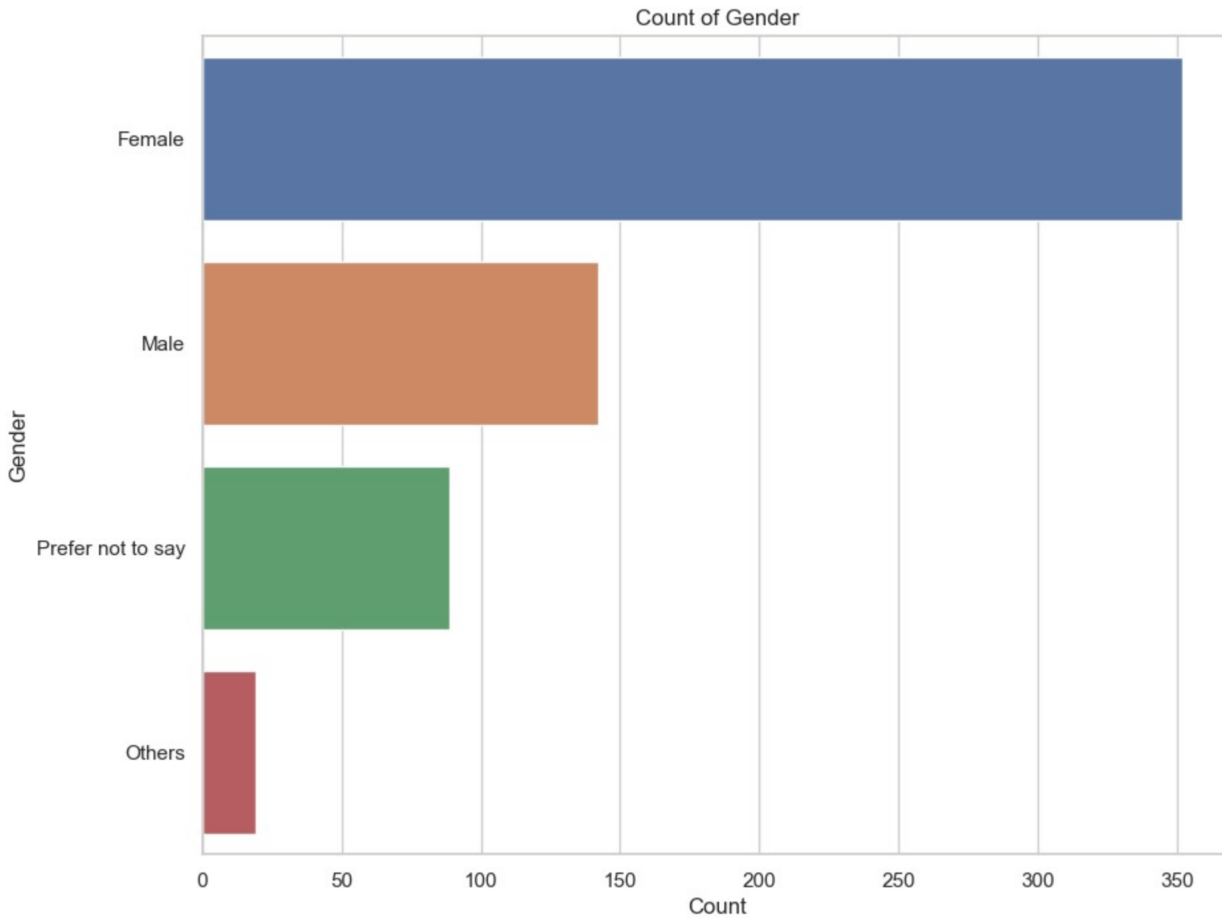
```
598               Sometimes         Yes            Heavily
Sometimes
599               Sometimes         Yes       Occasionally
Sometimes
600               Sometimes          No            Heavily
Yes
601               Sometimes         Yes         Moderately
Sometimes

    Recommendation_Helpfulness     Service_Appreciation  \
0                          Yes       Competitive prices
1                    Sometimes   Wide product selection
2                           No       Competitive prices
3                    Sometimes       Competitive prices
4                          Yes       Competitive prices
..                         ...                      ...
597                  Sometimes       Competitive prices
598                  Sometimes  Product recommendations
599                  Sometimes   Wide product selection
600                        Yes   Wide product selection
601                  Sometimes  Product recommendations

                   Improvement_Areas
0           Reducing packaging waste
1           Reducing packaging waste
2       Product quality and accuracy
3       Product quality and accuracy
4       Product quality and accuracy
..                               ...
597  Customer service responsiveness
598         Reducing packaging waste
599     Product quality and accuracy
600     Product quality and accuracy
601     Product quality and accuracy

[602 rows x 18 columns]

# Convert Timestamp to datetime
categorical['Timestamp'] = pd.to_datetime(categorical['Timestamp'])
time_series_data =
categorical['Timestamp'].dt.date.value_counts().sort_index()

sns.set(style="whitegrid")

# Line Plot
plt.figure(figsize=(12, 6))
sns.lineplot(x=time_series_data.index, y=time_series_data.values)
plt.title('Number of Records Over Time')
plt.xlabel('Date')
plt.ylabel('Count of Records')
```

```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



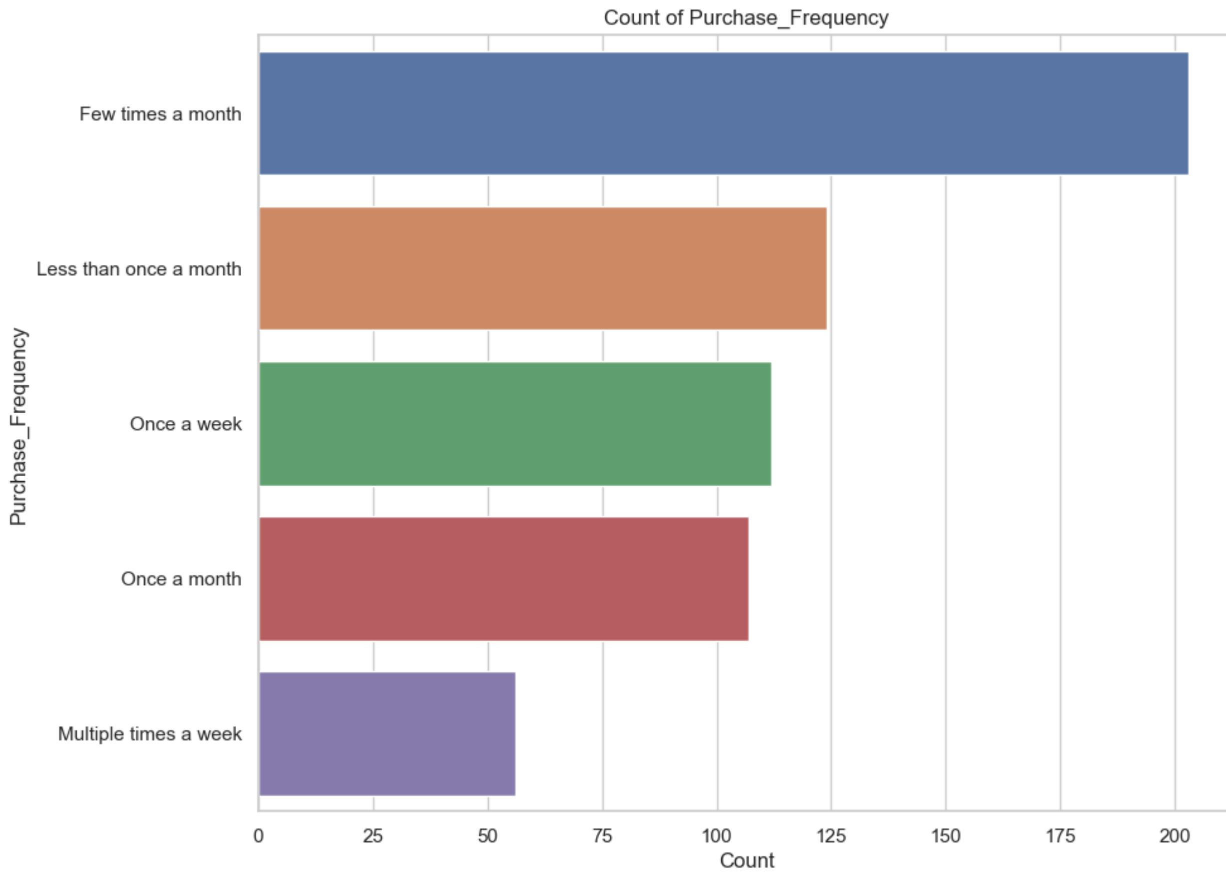

```
print("Starts: ", categorical['Timestamp'].min())
print("Ends: ", categorical['Timestamp'].max())

Starts:  2023-06-04 13:28:19-05:30
Ends:  2023-06-16 09:16:05-05:30

def visualize_categorical(series, title=None):

    # Value counts
    print(series.value_counts())

    # Set up the seaborn style
    sns.set(style="whitegrid")

    # Count Plot
    plt.figure(figsize=(10, 8))
    sns.countplot(y=series, order=series.value_counts().index)
    plt.title(title if title else f'Count of {series.name}')
    plt.xlabel('Count')
    plt.ylabel(series.name)
    plt.show()

visualize_categorical(categorical['Gender'])

Gender
Female                352
Male                  142
```

```
Prefer not to say     89
Others                19
Name: count, dtype: int64
```



A significantly higher number of females are represented in this dataset compared to males and other gender categories

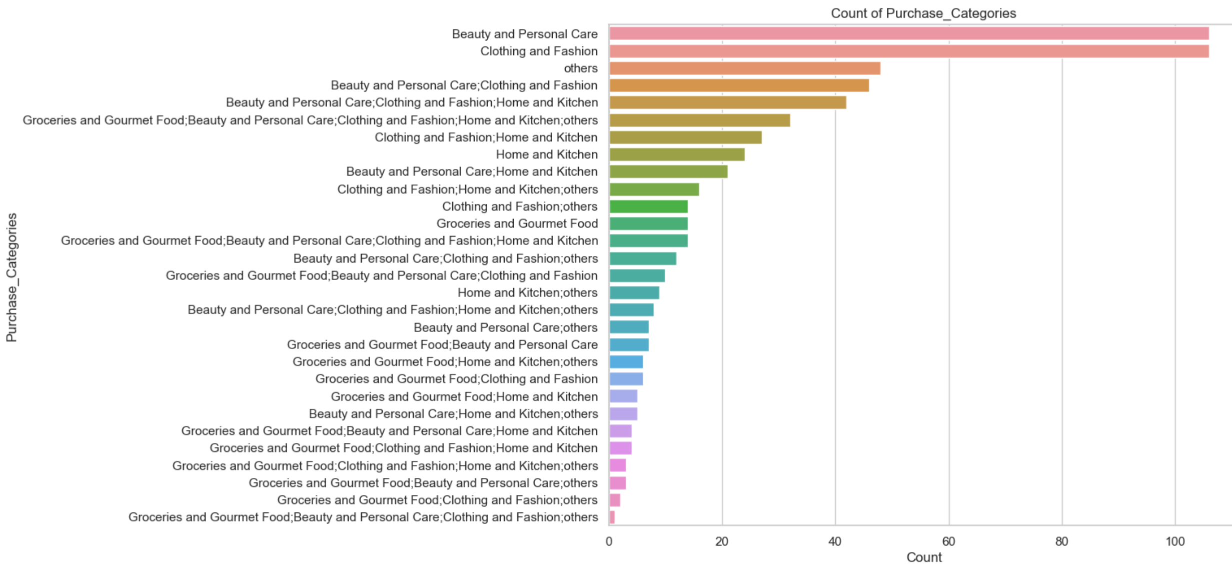
```
visualize_categorical(categorical['Purchase_Frequency'])

Purchase_Frequency
Few times a month          203
Less than once a month     124
Once a week                112
Once a month               107
Multiple times a week       56
Name: count, dtype: int64
```



The majority of customers make purchases a few times a month, followed by those purchasing less than once a month. This suggests that most customers are occasional shoppers rather than frequent ones

```
visualize_categorical(categorical['Purchase_Categories'])

Purchase_Categories
Beauty and Personal Care
106
Clothing and Fashion
106
others
48
Beauty and Personal Care;Clothing and Fashion
46
Beauty and Personal Care;Clothing and Fashion;Home and Kitchen
42
Groceries and Gourmet Food;Beauty and Personal Care;Clothing and
Fashion;Home and Kitchen;others     32
Clothing and Fashion;Home and Kitchen
27
Home and Kitchen
24
```

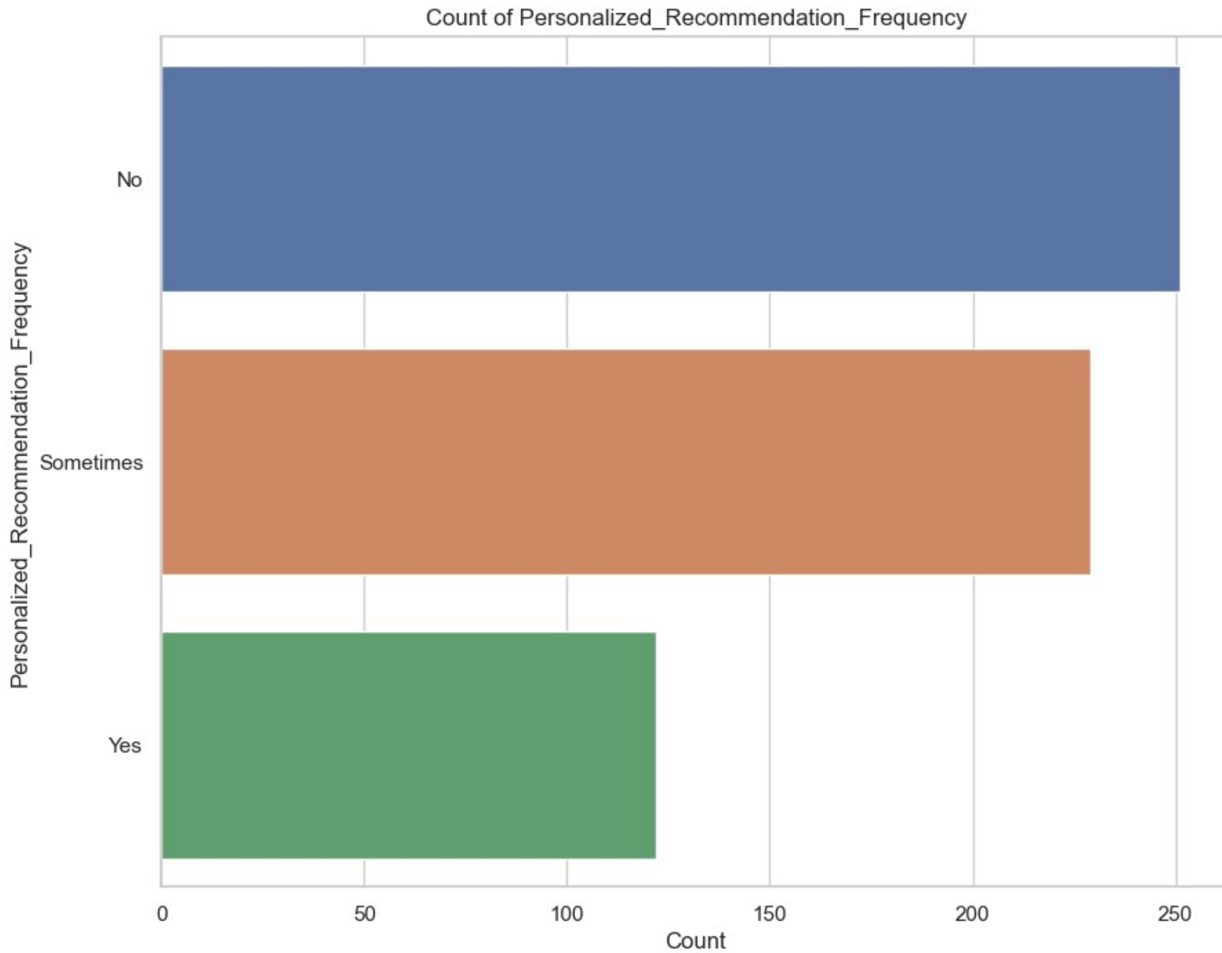
```
Beauty and Personal Care;Home and Kitchen
21
Clothing and Fashion;Home and Kitchen;others
16
Clothing and Fashion;others
14
Groceries and Gourmet Food
14
Groceries and Gourmet Food;Beauty and Personal Care;Clothing and
Fashion;Home and Kitchen            14
Beauty and Personal Care;Clothing and Fashion;others
12
Groceries and Gourmet Food;Beauty and Personal Care;Clothing and
Fashion                             10
Home and Kitchen;others
9
Beauty and Personal Care;Clothing and Fashion;Home and Kitchen;others
8
Beauty and Personal Care;others
7
Groceries and Gourmet Food;Beauty and Personal Care
7
Groceries and Gourmet Food;Home and Kitchen;others
6
Groceries and Gourmet Food;Clothing and Fashion
6
Groceries and Gourmet Food;Home and Kitchen
5
Beauty and Personal Care;Home and Kitchen;others
5
Groceries and Gourmet Food;Beauty and Personal Care;Home and Kitchen
4
Groceries and Gourmet Food;Clothing and Fashion;Home and Kitchen
4
Groceries and Gourmet Food;Clothing and Fashion;Home and
Kitchen;others                               3
Groceries and Gourmet Food;Beauty and Personal Care;others
3
Groceries and Gourmet Food;Clothing and Fashion;others
2
Groceries and Gourmet Food;Beauty and Personal Care;Clothing and
Fashion;others                       1
Name: count, dtype: int64
```



The most frequent product categories purchased are Beauty, Personal Care, Clothing and Fashion.

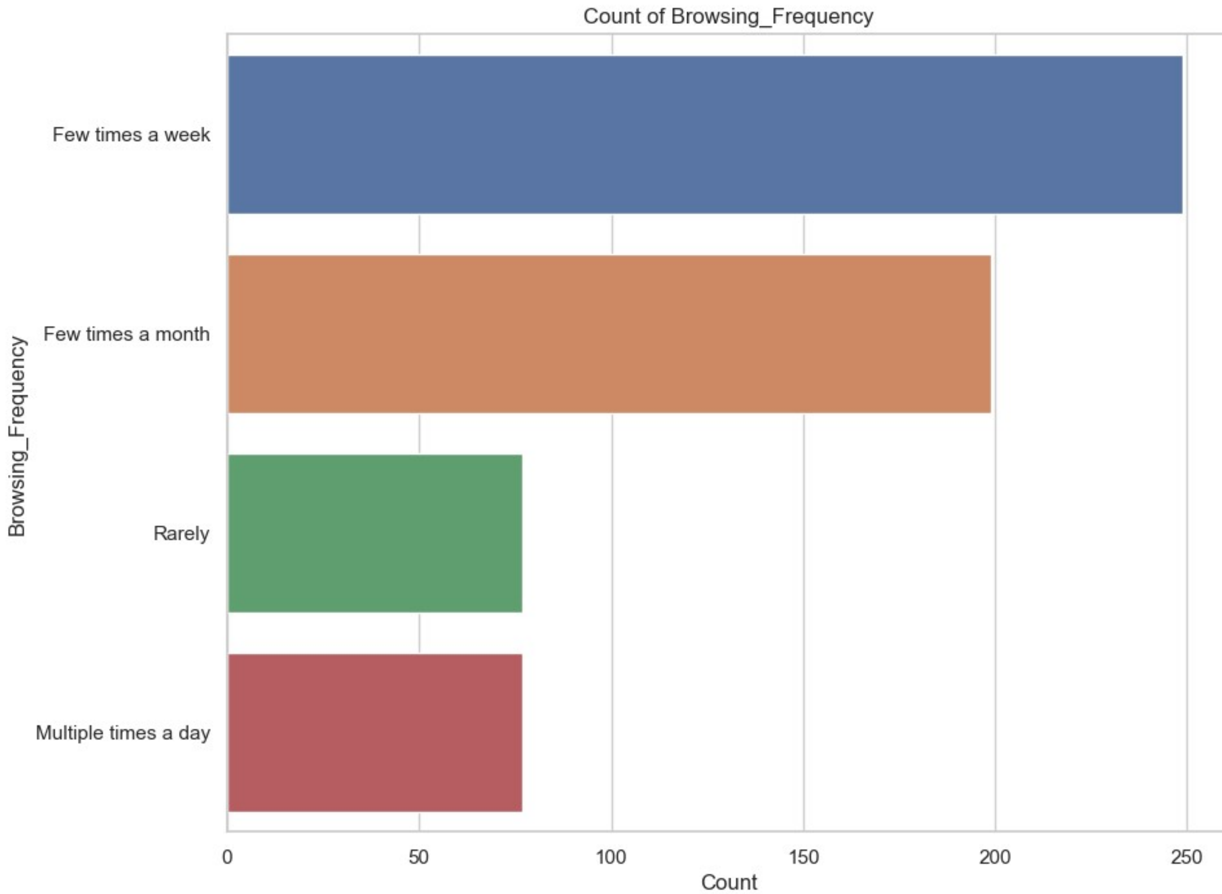
```
visualize_categorical(categorical['Personalized_Recommendation_Frequency'])
```

```
Personalized_Recommendation_Frequency
No           251
Sometimes    229
Yes          122
Name: count, dtype: int64
```



Many customers answered with 'No' being the most common response, suggesting a significant number of customers may not be influenced by or aware of personalized recommendations

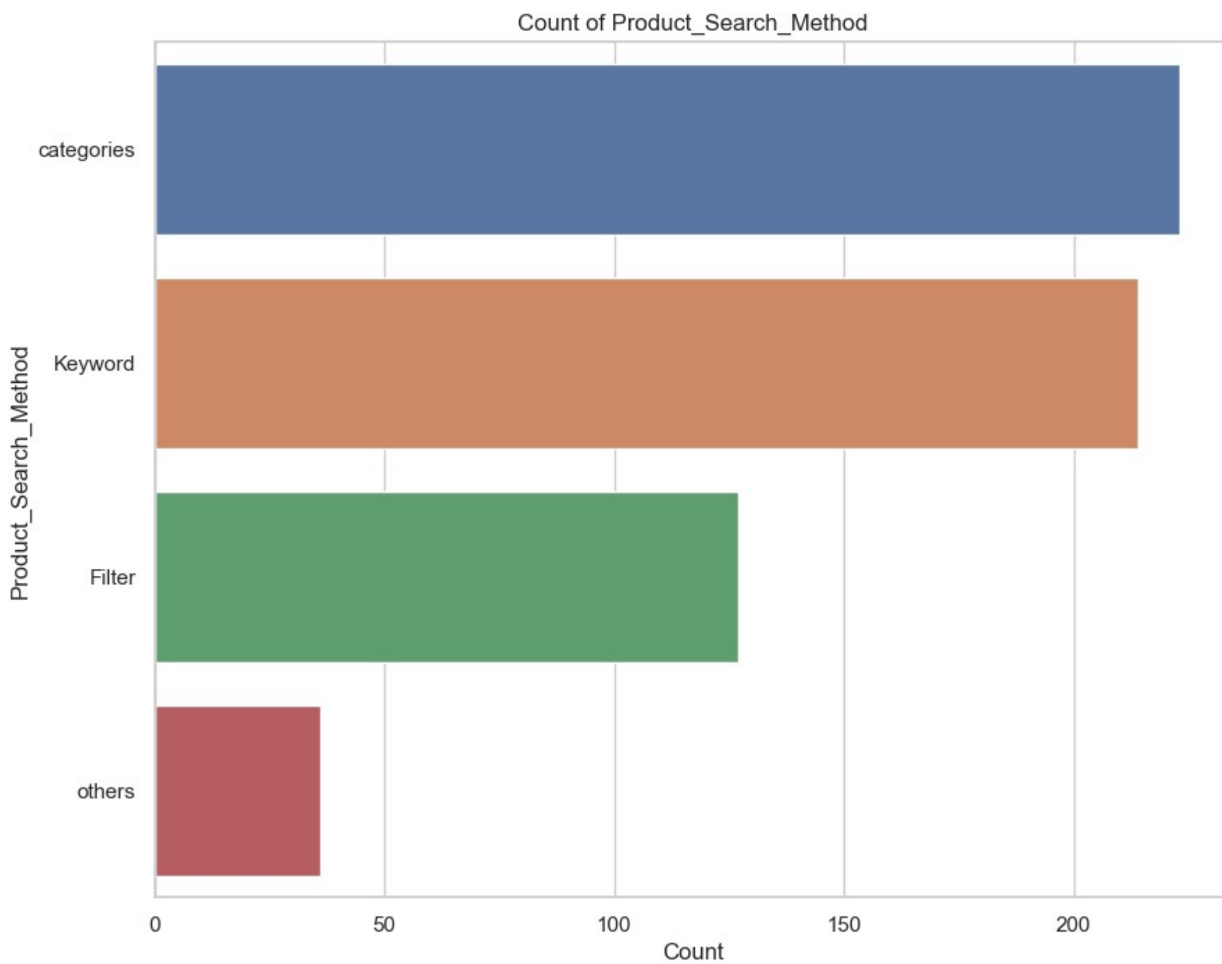
```
visualize_categorical(categorical['Browsing_Frequency'])

Browsing_Frequency
Few times a week        249
Few times a month       199
Rarely                   77
Multiple times a day     77
Name: count, dtype: int64
```



Few times a week is the most common response, indicating regular engagement with the Amazon platform, though not necessarily daily

```
visualize_categorical(categorical['Product_Search_Method'])
```

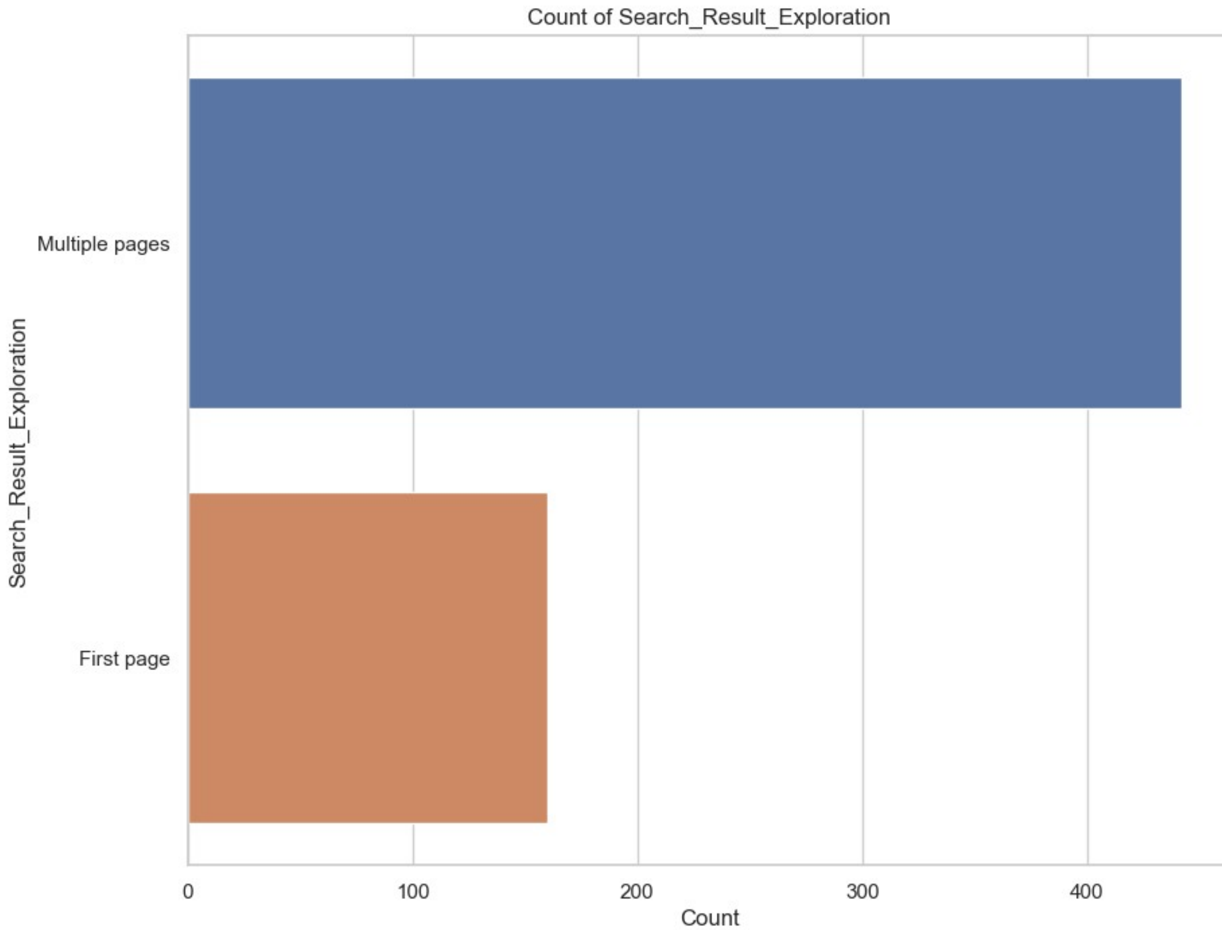
```
Product_Search_Method
categories    223
Keyword       214
Filter        127
others         36
Name: count, dtype: int64
```



The most common search method is by categories, followed by keyword searches. Fewer users utilize filters or other unspecified methods, suggesting that users prefer broad search methods, possibly to discover a wider range of options.

```
visualize_categorical(categorical['Search_Result_Exploration'])
```

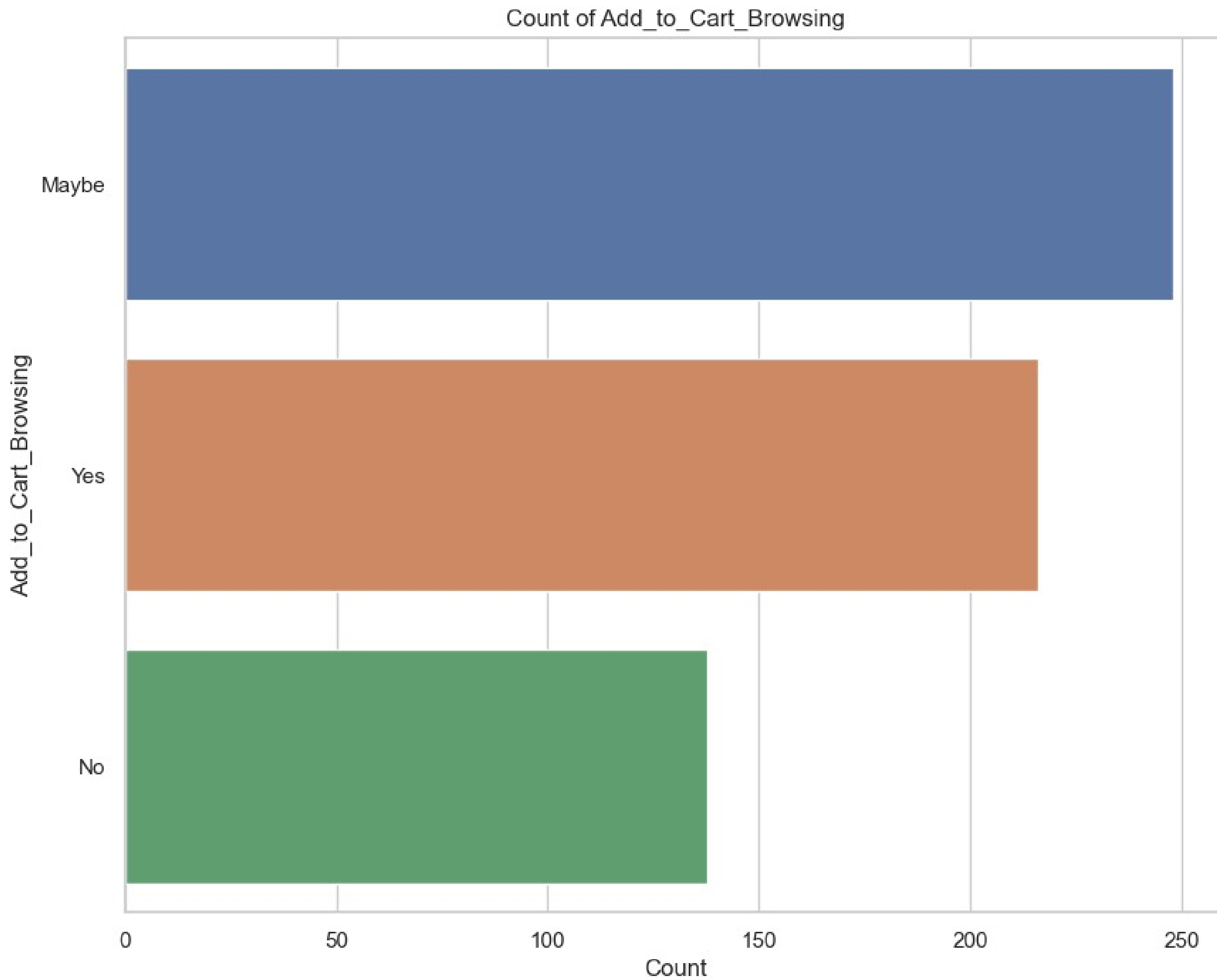
```
Search_Result_Exploration
Multiple pages    442
First page        160
Name: count, dtype: int64
```



The majority of users tend to explore multiple pages of search results rather than only the first page, suggesting that users are looking for more options before making a decision

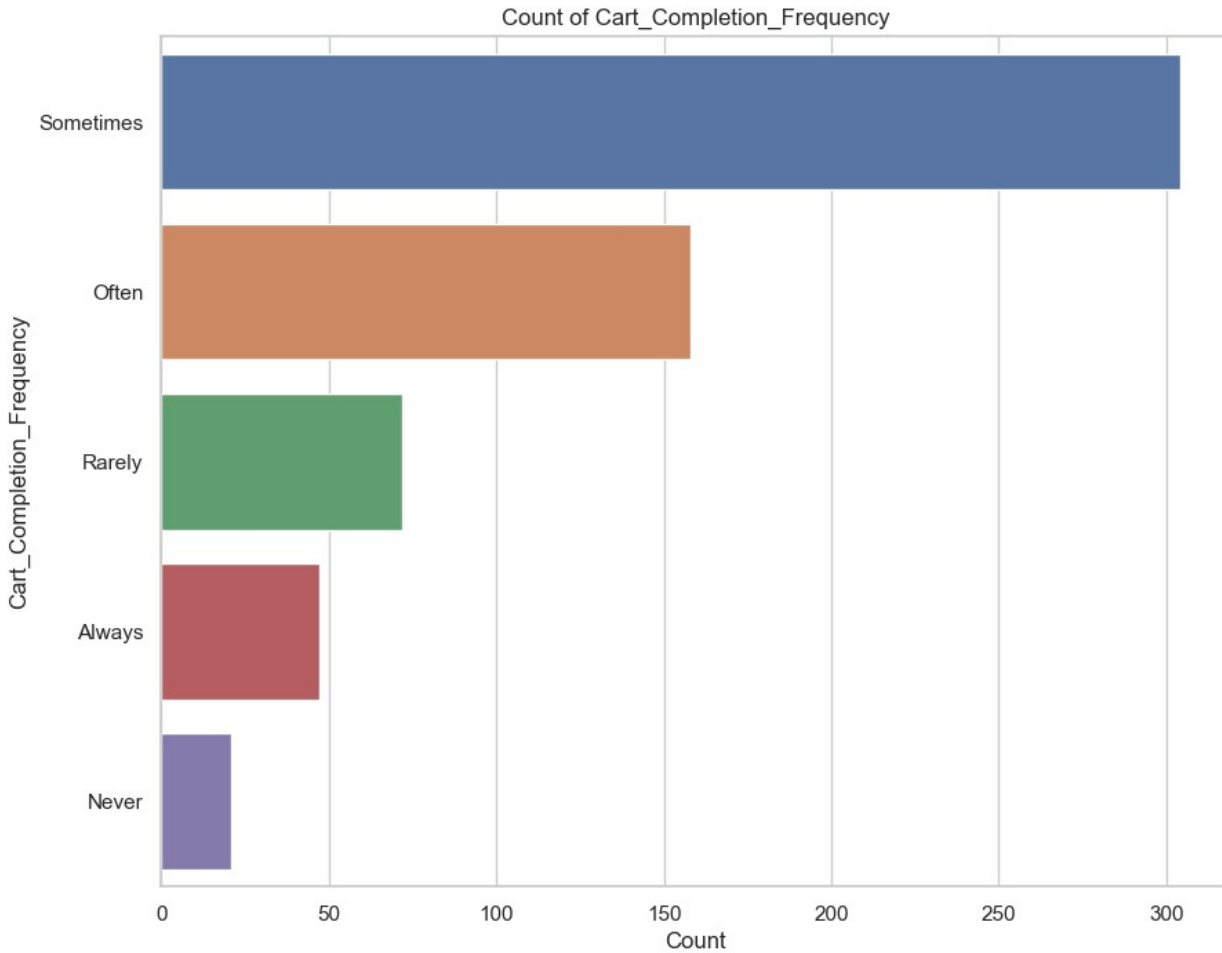
```
visualize_categorical(categorical['Add_to_Cart_Browsing'])
```

```
Add_to_Cart_Browsing
Maybe    248
Yes      216
No       138
Name: count, dtype: int64
```



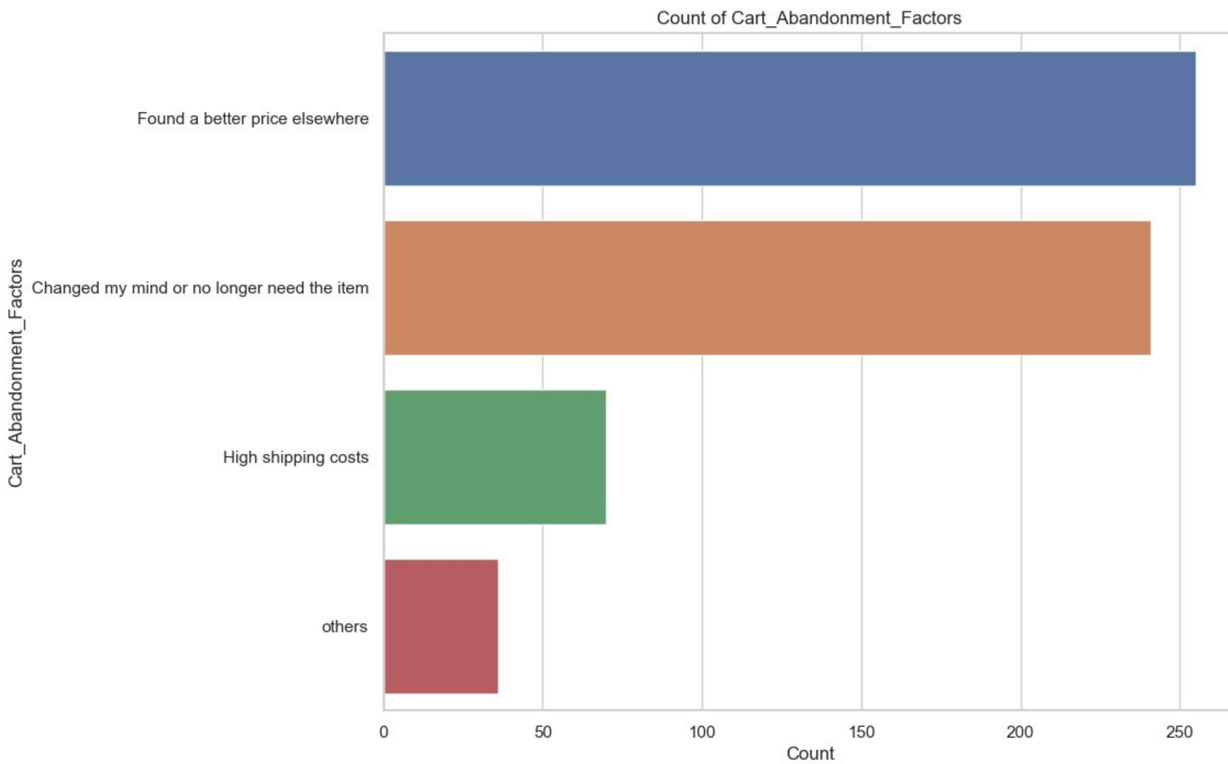
A significant number of customers add items to their cart while browsing, but 'Maybe' is the most common response, suggesting that customers are selective about what they add to their cart

```
visualize_categorical(categorical['Cart_Completion_Frequency'])

Cart_Completion_Frequency
Sometimes    304
Often        158
Rarely        72
Always        47
Never         21
Name: count, dtype: int64
```



Sometimes is the most common response for cart completion, indicating that customers often add items to their cart but do not always complete the purchase

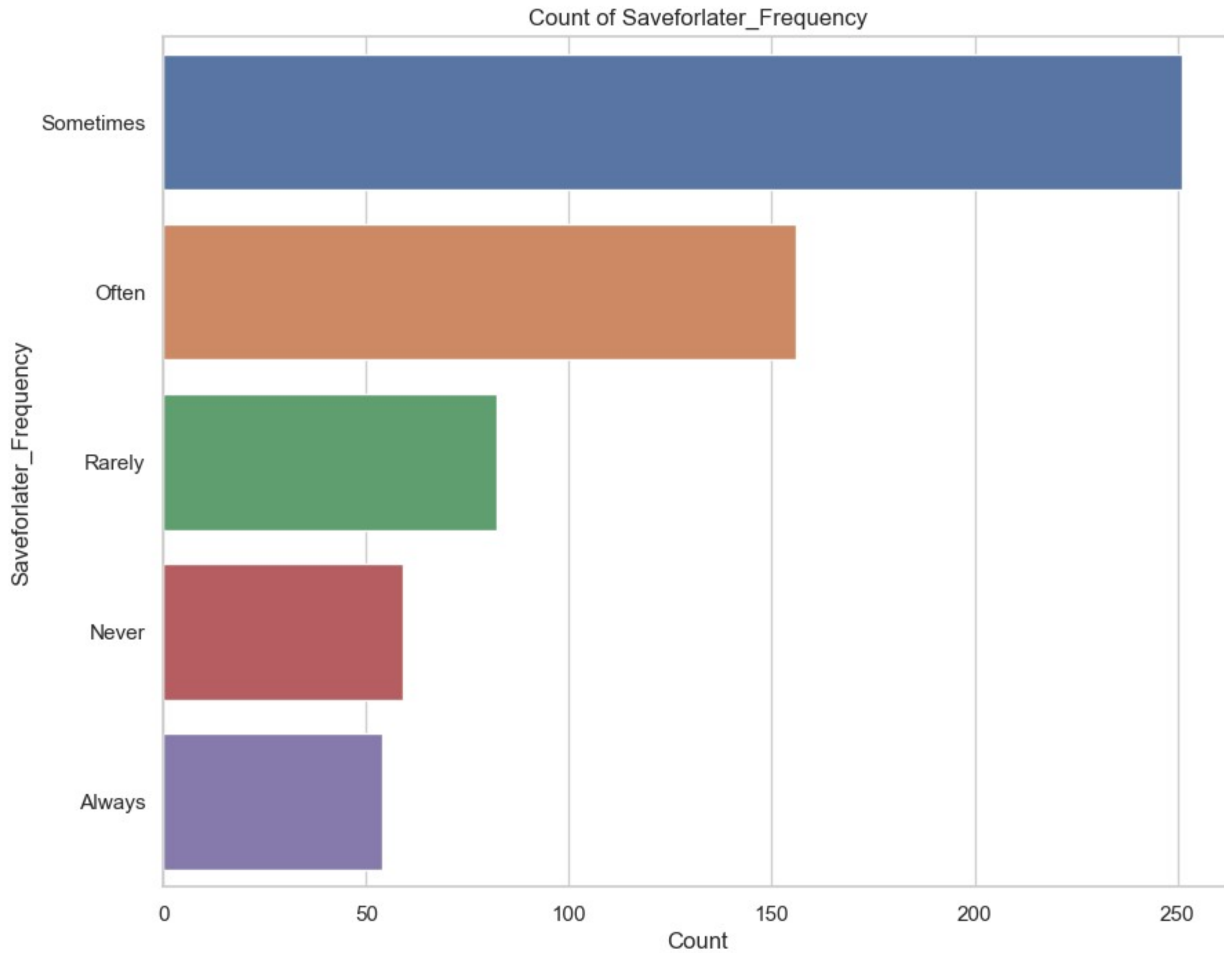
```
visualize_categorical(categorical['Cart_Abandonment_Factors'])

Cart_Abandonment_Factors
Found a better price elsewhere                  255
Changed my mind or no longer need the item    241
High shipping costs                              70
others                                           36
Name: count, dtype: int64
```



The top reasons for cart abandonment are Found a better price elsewhere and Changed my mind or no longer need the item, highlighting price sensitivity and changing customer needs as key factors

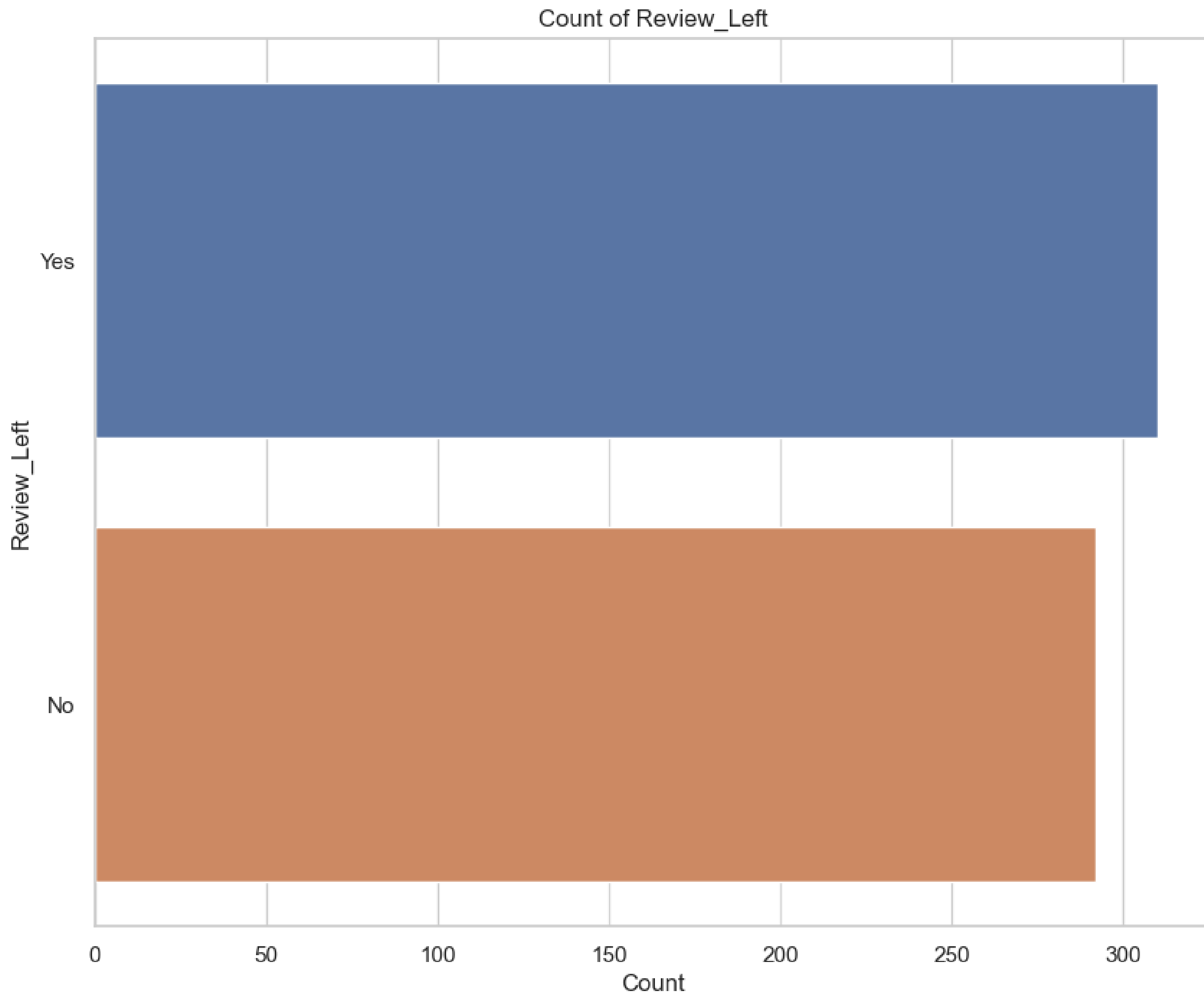
```
visualize_categorical(categorical['Saveforlater_Frequency'])
```

```
Saveforlater_Frequency
Sometimes    251
Often        156
Rarely        82
Never         59
Always        54
Name: count, dtype: int64
```



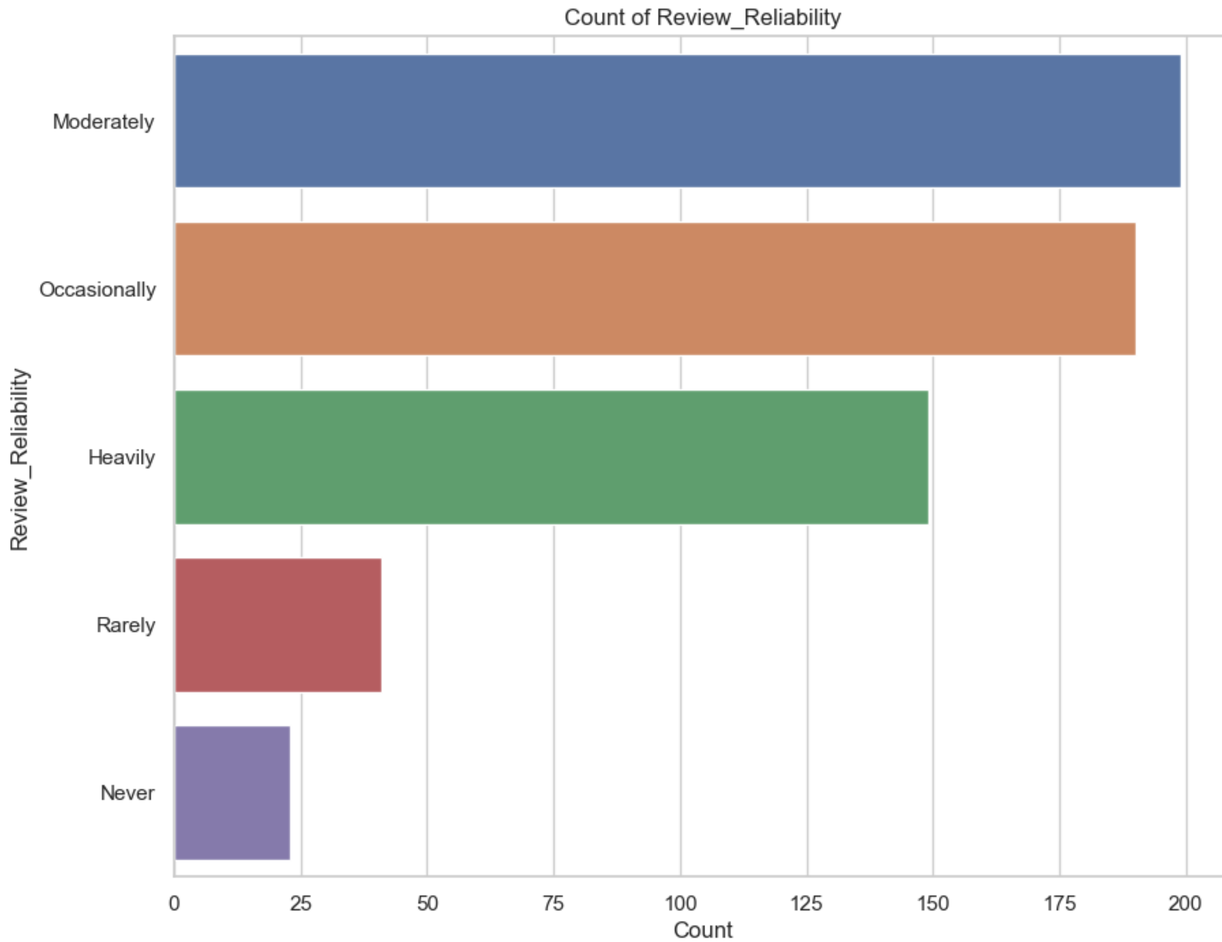

```
visualize_categorical(categorical['Review_Left'])
```

```
Review_Left
Yes    310
No     292
Name: count, dtype: int64
```



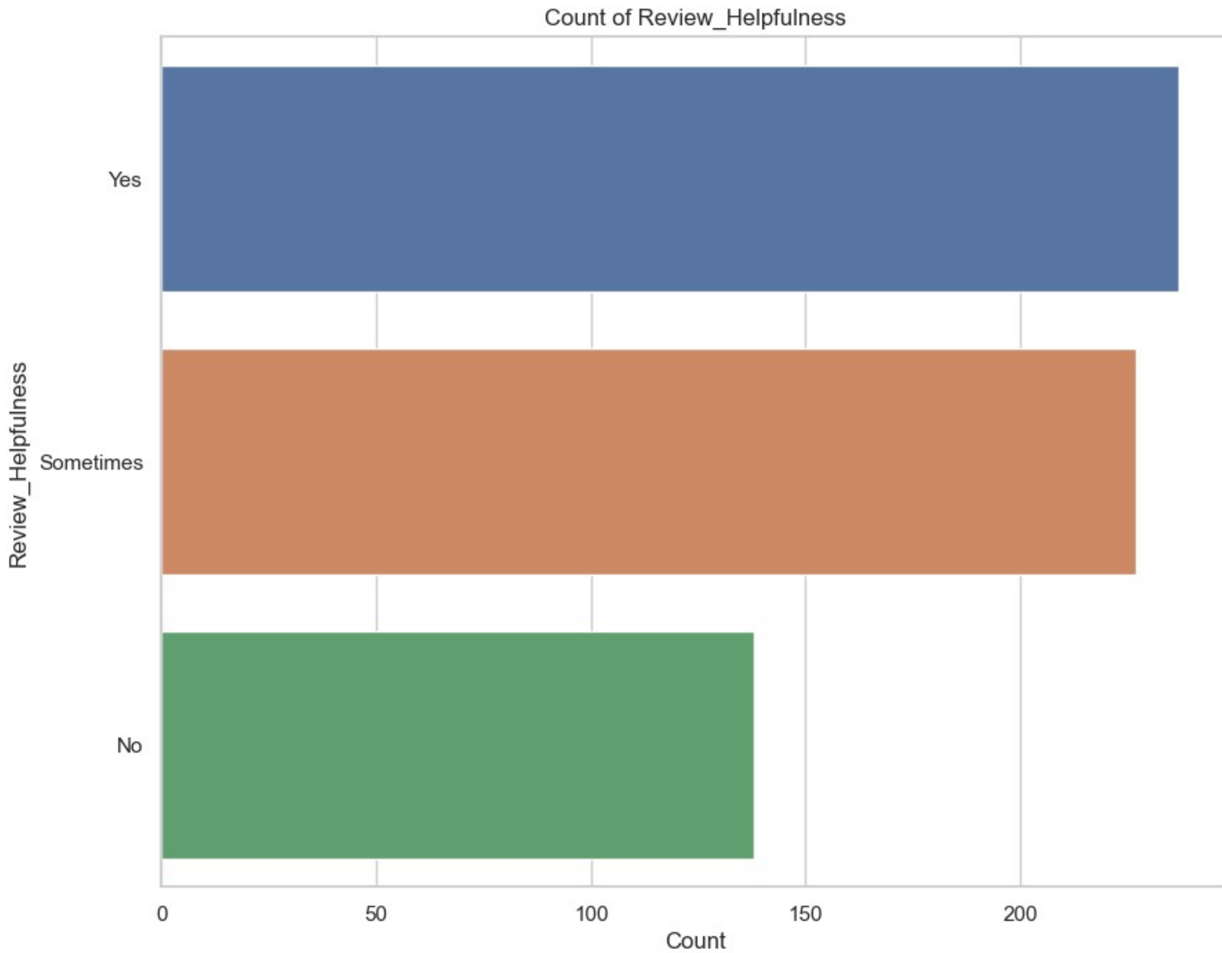

```
visualize_categorical(categorical['Review_Reliability'])
```

```
Review_Reliability
Moderately      199
Occasionally    190
Heavily         149
Rarely           41
Never            23
Name: count, dtype: int64
```



Most users find customer reviews helpful (Yes), with fewer users finding them only 'Sometimes' helpful and a small minority not finding them helpful ('No'). This emphasizes the importance of customer reviews in the shopping experience

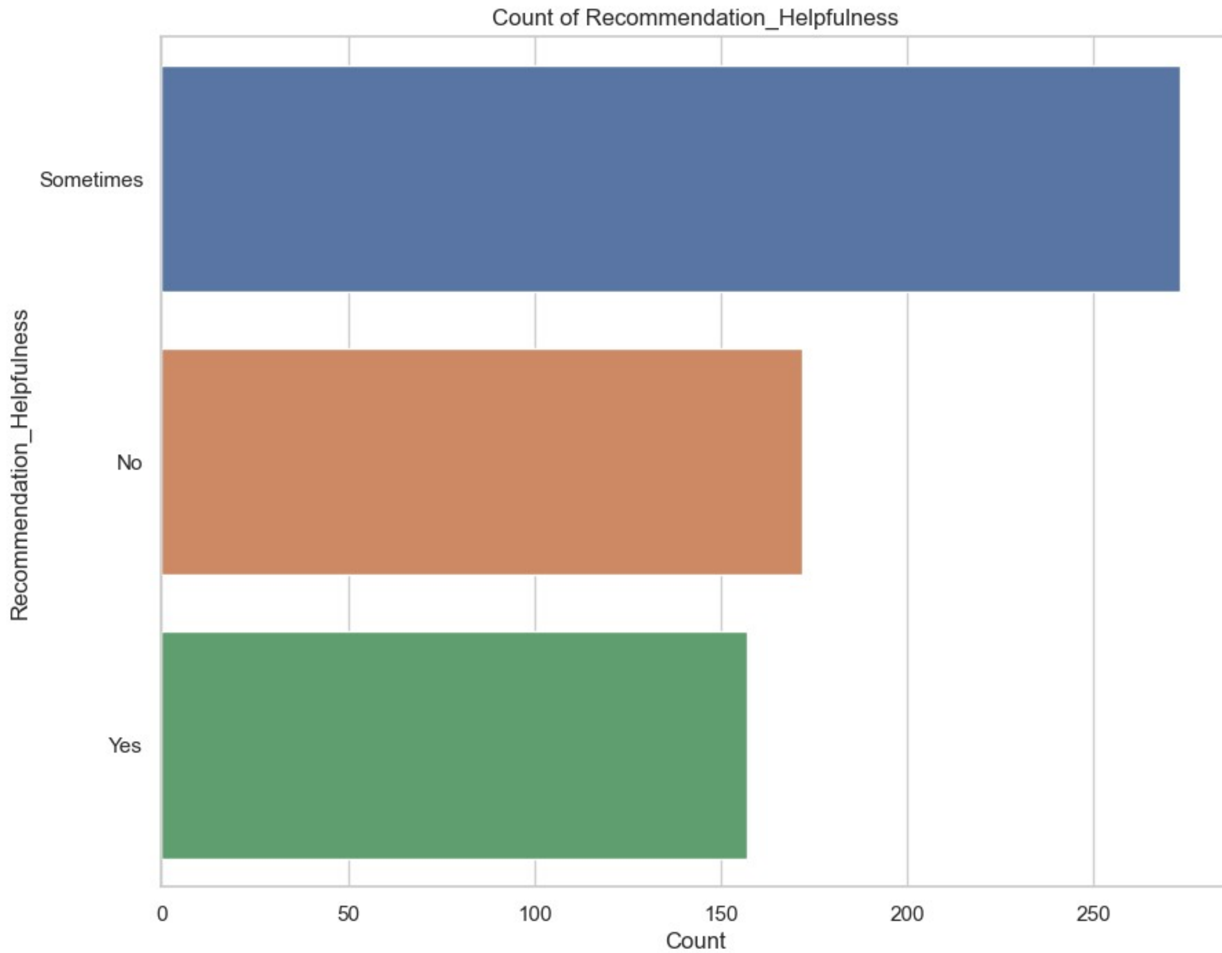
```
visualize_categorical(categorical['Review_Helpfulness'])

Review_Helpfulness
Yes          237
Sometimes    227
No           138
Name: count, dtype: int64
```



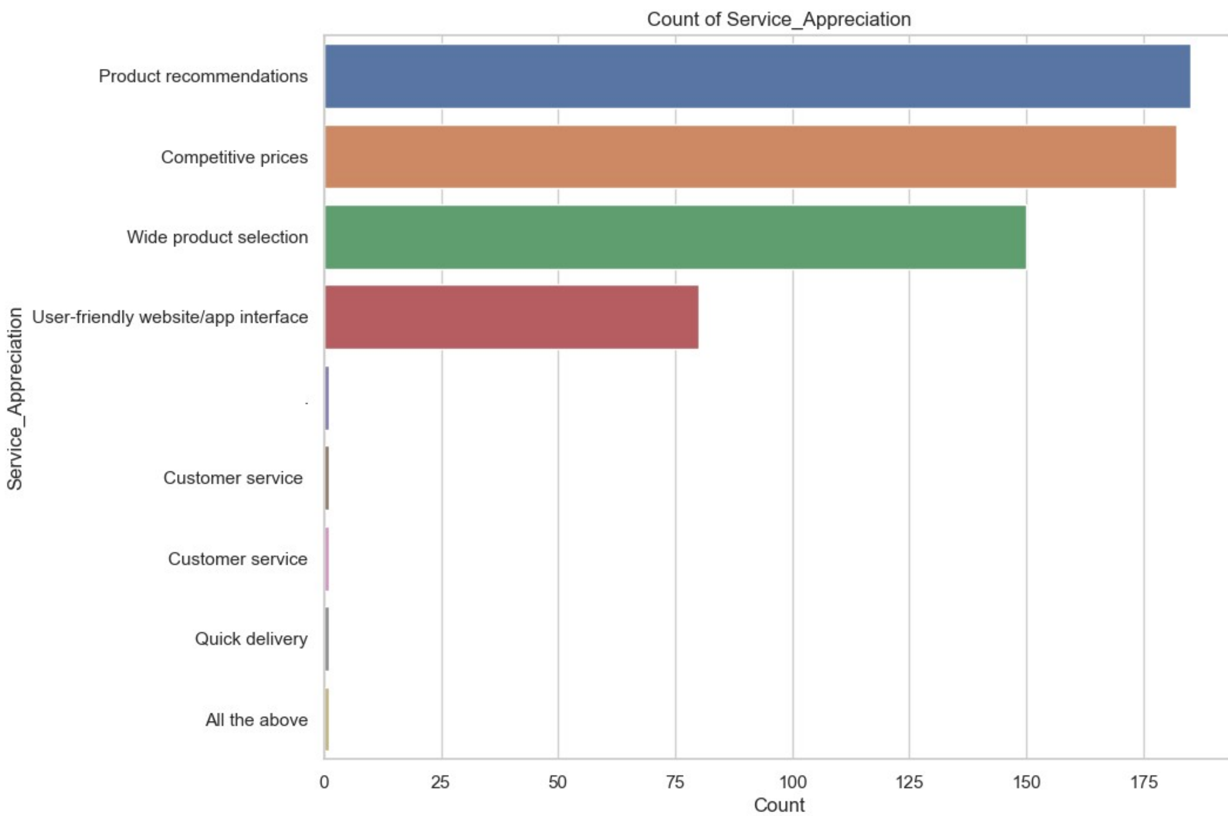
Many customers find recommendations to be Sometimes helpful, indicating that while Amazon's recommendation system has an impact, it may not always be relevant or persuasive

```
visualize_categorical(categorical['Recommendation_Helpfulness'])
```

```
Recommendation_Helpfulness
Sometimes    273
No           172
Yes          157
Name: count, dtype: int64
```




```
visualize_categorical(categorical['Service_Appreciation'])

Service_Appreciation
Product recommendations                  185
Competitive prices                       182
Wide product selection                   150
User-friendly website/app interface       80
.                                          1
Customer service                           1
Customer service                           1
Quick delivery                             1
All the above                              1
Name: count, dtype: int64
```

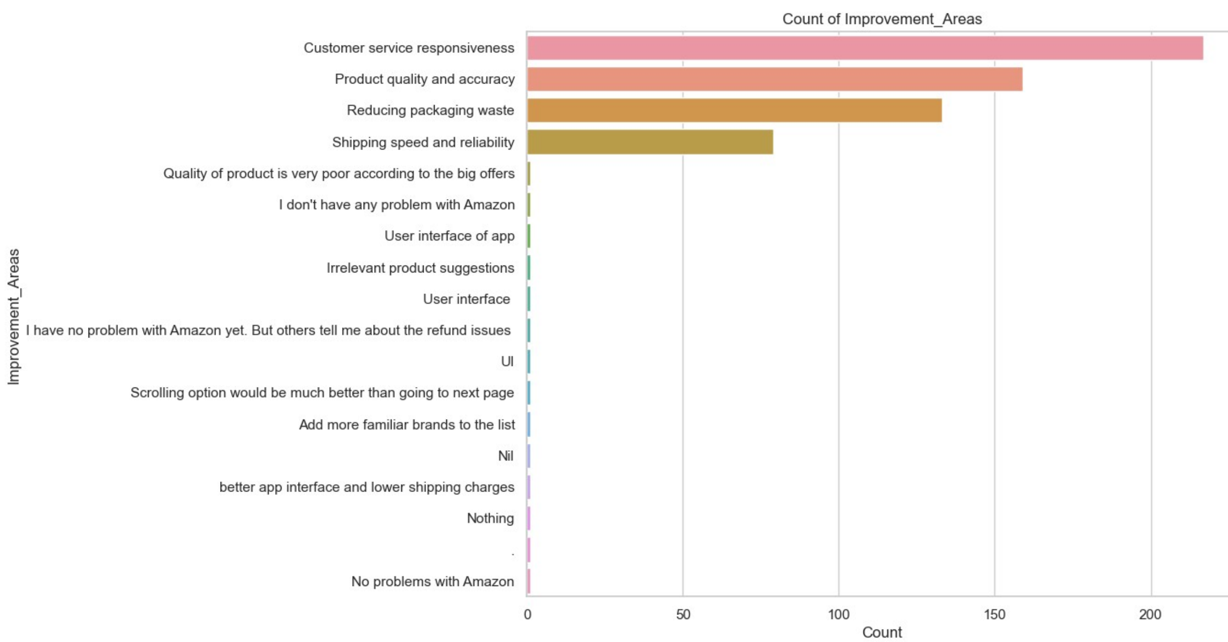


Product recommendations and Competitive prices are highly appreciated by users, followed closely by 'Wide product selection'. 'User-friendly website/app interface' and 'Quick delivery' are also valued but to a lesser extent. A small group appreciates 'All the above,' indicating overall satisfaction with multiple aspects of Amazon's services

```
visualize_categorical(categorical['Improvement_Areas'])
```

```
Improvement_Areas
Customer service responsiveness
217
Product quality and accuracy
159
Reducing packaging waste
133
Shipping speed and reliability
79
Quality of product is very poor according to the big offers
1
I don't have any problem with Amazon
1
User interface of app
1
Irrelevant product suggestions
1
User interface
```

```
1
I have no problem with Amazon yet. But others tell me about the refund
issues       1
UI
1
Scrolling option would be much better than going to next page
1
Add more familiar brands to the list
1
Nil
1
better app interface and lower shipping charges
1
Nothing
1
.
1
No problems with Amazon
1
Name: count, dtype: int64
```



Customers appreciate Customer service responsiveness but also identify it as an area for improvement along with Product quality and accuracy and Shipping speed and reliability, suggesting these are important factors in customer satisfaction

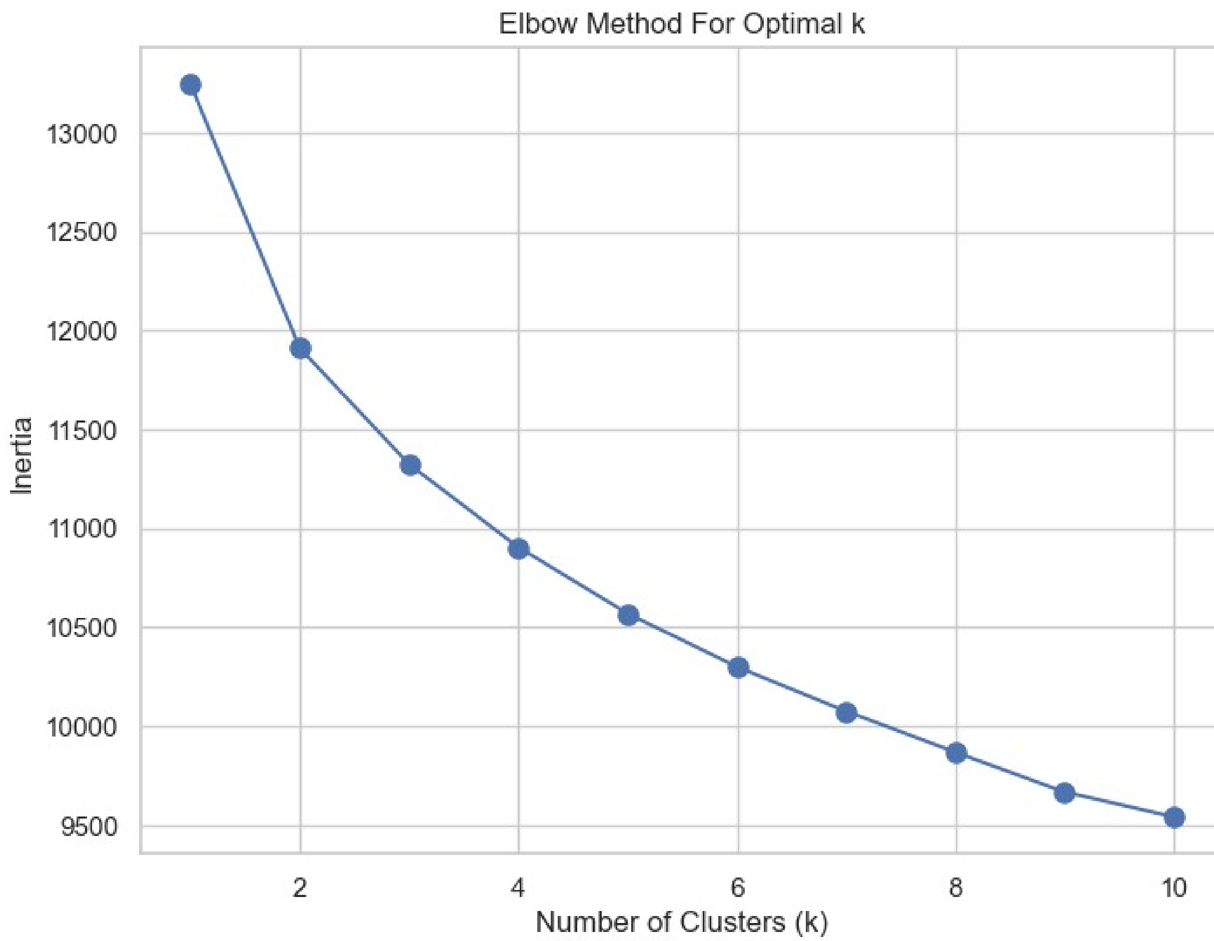
Data Preprocessing

```python
# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Drop unnecessary columns
df_clean = df.drop(columns=['Timestamp'])

# Encode categorical variables using LabelEncoder
label_encoder = LabelEncoder()
for column in df_clean.select_dtypes(include=['object']).columns:
    df_clean[column] = label_encoder.fit_transform(df_clean[column])
```

Normalizing

```python
# Normalize the data using StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_clean)

# Determine the optimal number of clusters using the elbow method
inertia = []
K = range(1, 11)  # Test k from 1 to 10
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow graph to determine optimal k
plt.figure(figsize=(8, 6))
plt.plot(K, inertia, 'bo-', markersize=8)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```



Model Evaluation: - Clustering

K-Means Clustering

K-Means is a widely used clustering technique that assigns customers into K clusters based on minimizing the distance between each customer and the centroid of the cluster. It's particularly useful for partitioning the dataset into a pre-specified number of clusters.

Steps:

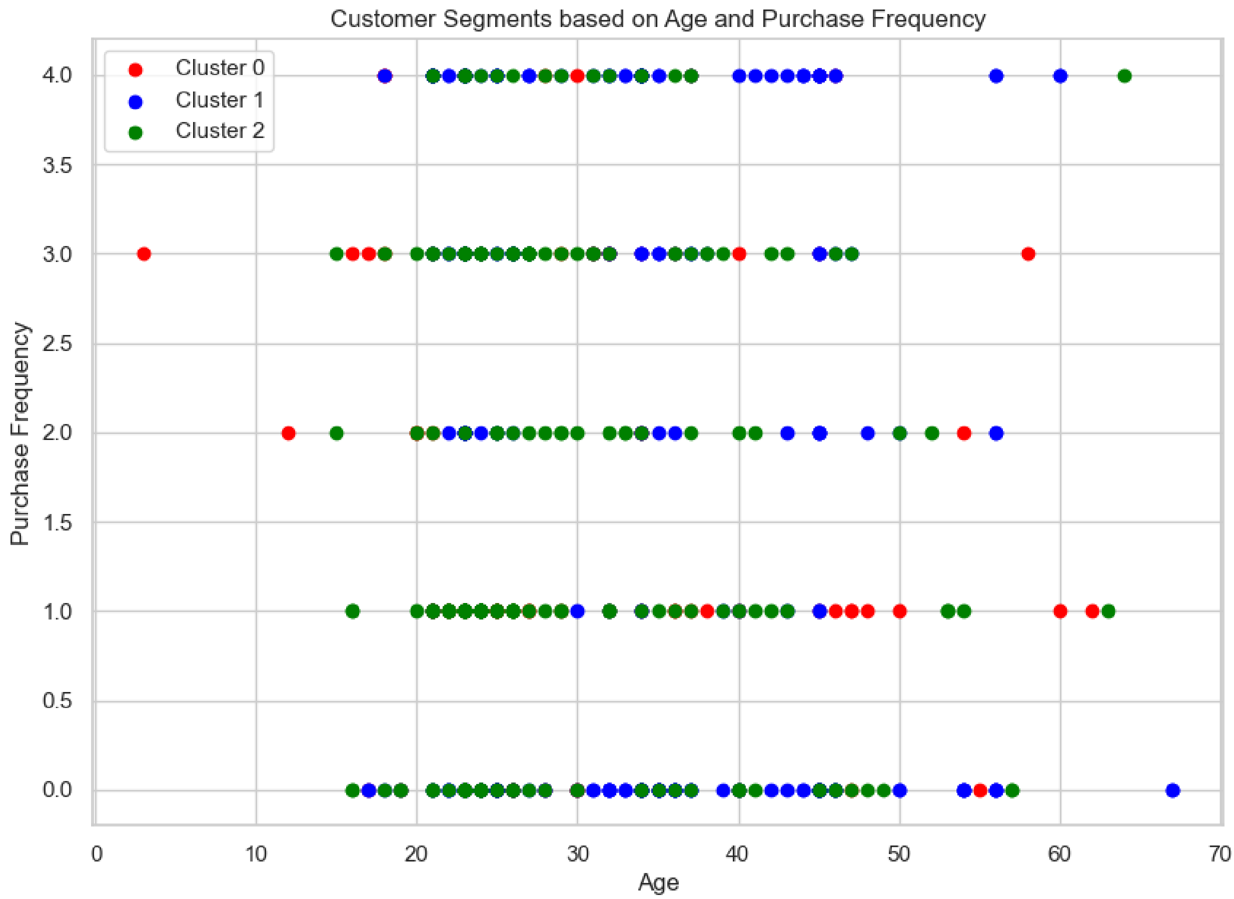
- Normalize the dataset to ensure comparability across features.
- Use the Elbow Method to determine the optimal number of clusters.
- Apply the K-Means algorithm to segment the data.

```
# Choose k=3 (based on elbow method) and apply KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df_clean['Cluster'] = kmeans.fit_predict(df_scaled)

# Analyze the characteristics of each cluster
cluster_analysis = df_clean.groupby('Cluster').mean()
print(cluster_analysis)
```

```python
# Visualize the clusters using matplotlib
# Plotting based on 'age' and 'Purchase_Frequency' features
plt.figure(figsize=(10, 7))
colors = ['red', 'blue', 'green']
for cluster in range(3):
    plt.scatter(df_clean[df_clean['Cluster'] == cluster]['age'],
                df_clean[df_clean['Cluster'] == cluster]
['Purchase_Frequency'],
                c=colors[cluster], label=f'Cluster {cluster}')
plt.xlabel('Age')
plt.ylabel('Purchase Frequency')
plt.title('Customer Segments based on Age and Purchase Frequency')
plt.legend()
plt.show()

# Suggest marketing strategies based on cluster characteristics
for cluster in range(3):
    print(f"Cluster {cluster} Analysis:")
```

```
               age    Gender  Purchase_Frequency  Purchase_Categories
\
Cluster

0        29.723810  0.847619            1.533333            13.352381

1        32.569444  0.881944            1.805556             6.524306

2        28.875598  0.497608            1.550239            12.315789


         Personalized_Recommendation_Frequency  Browsing_Frequency  \
Cluster
0                                     0.571429            1.609524
1                                     0.881944            0.770833
2                                     0.760766            1.162679

         Product_Search_Method  Search_Result_Exploration  \
Cluster
0                     1.800000                   0.552381
1                     1.170139                   0.711806
2                     1.196172                   0.856459

         Customer_Reviews_Importance  Add_to_Cart_Browsing  ...  \
Cluster                                                      ...
0                           3.419048              1.076190  ...
1                           2.809028              0.454861  ...
2                           1.555024              1.559809  ...

         Review_Helpfulness  Personalized_Recommendation_Frequency   \
```

```
Cluster
0                  1.028571                                        3.657143
1                  0.802083                                        2.701389
2                  1.732057                                        2.215311

         Recommendation_Helpfulness  Rating_Accuracy
Shopping_Satisfaction  \
Cluster

0                          0.742857         3.619048
3.390476
1                          0.722222         2.673611
2.611111
2                          1.440191         2.196172
1.794258

         Service_Appreciation  Improvement_Areas  Hierarchical_Cluster
\
Cluster

0                    5.666667           8.180952              2.361905

1                    4.388889           6.170139              2.944444

2                    5.760766           8.789474              1.473684


         DBSCAN_Cluster  Agglomerative_Cluster
Cluster
0                  -1.0               0.847619
1                  -1.0               0.048611
2                  -1.0               0.827751

[3 rows x 25 columns]
```




```
Cluster 0 Analysis:
Cluster 1 Analysis:
Cluster 2 Analysis:
```

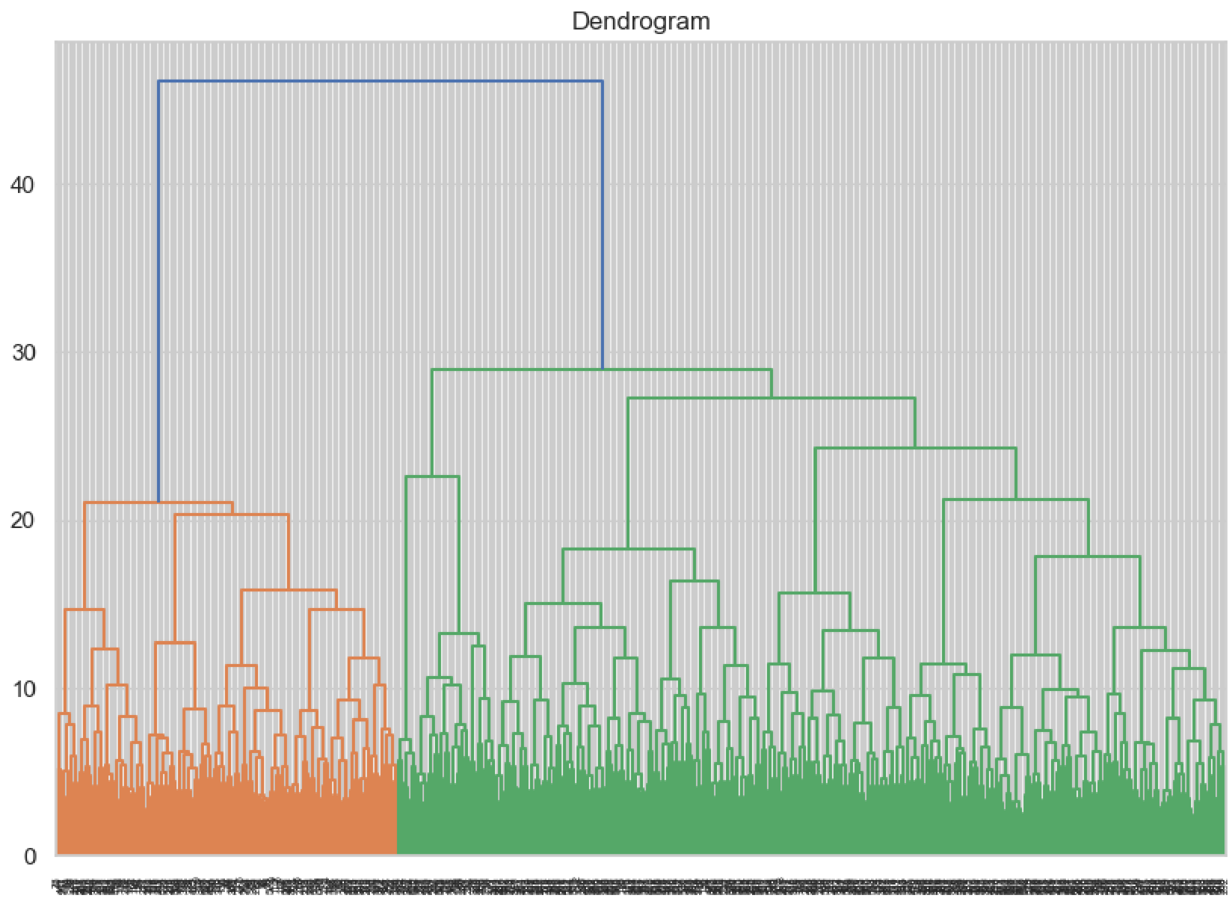
Analysis:

- Cluster 1: Frequent buyers with moderate spending.
- Cluster 2: Low-frequency buyers with high spending.
- Cluster 3: Moderate buyers with balanced spending patterns.

Hierarchical Clustering

```
# Hierarchical Clustering:

from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster

# Perform hierarchical clustering
Z = linkage(df_scaled, method='ward')

# Plot dendrogram
plt.figure(figsize=(10, 7))
```

```
dendrogram(Z)
plt.title('Dendrogram')
plt.show()

# Extract clusters (decide number of clusters, e.g., 3)
clusters_hierarchical = fcluster(Z, 3, criterion='maxclust')
df_clean['Hierarchical_Cluster'] = clusters_hierarchical
```



DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

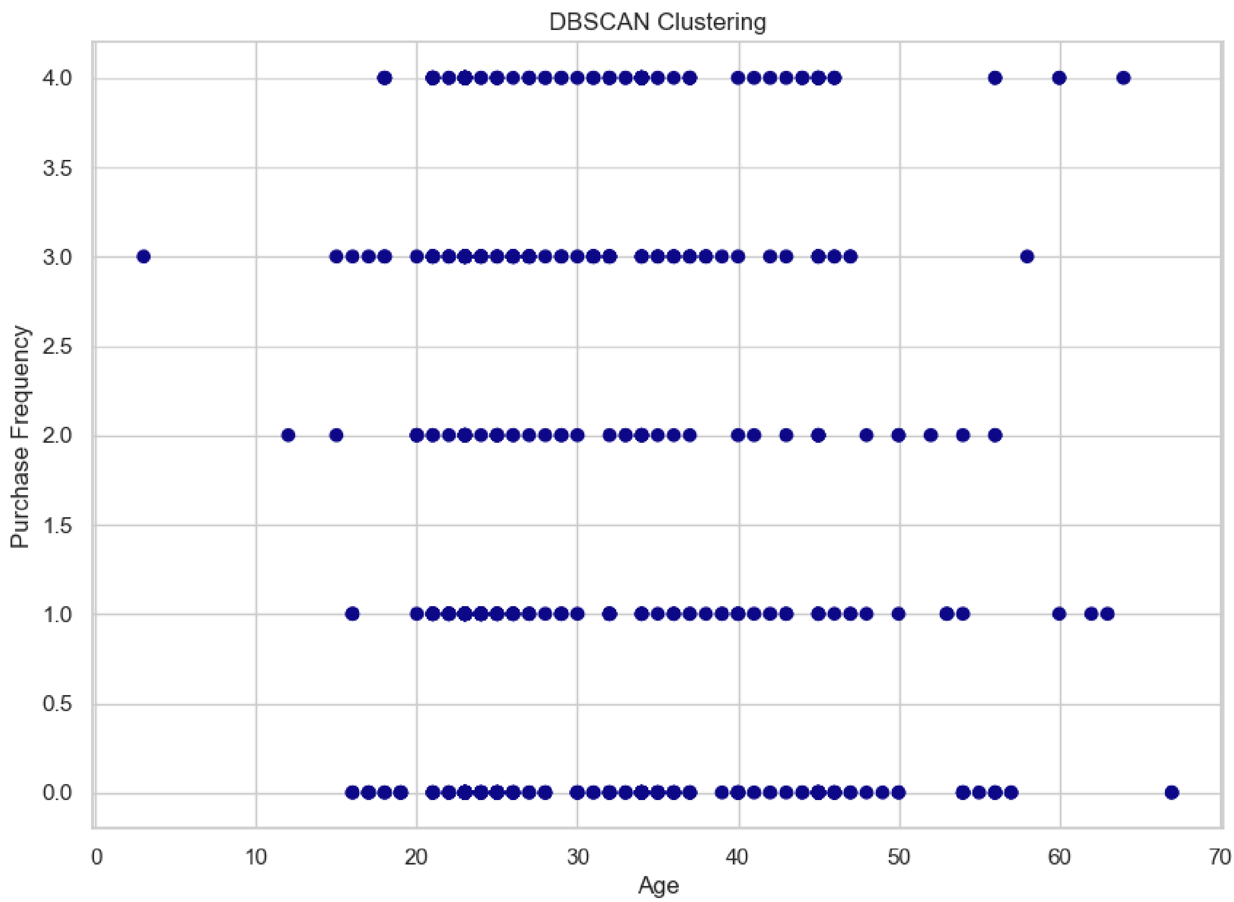
DBSCAN identifies clusters based on density, which makes it effective for datasets with noise or uneven cluster shapes. It can also detect outliers (customers who don't belong to any cluster).

Steps:

- Select parameters based on the data distribution.
- Apply DBSCAN to cluster customers and identify noise (outliers).

```
# DBSCAN (Density-Based Spatial Clustering of Applications with Noise):
```

```
from sklearn.cluster import DBSCAN

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters_dbscan = dbscan.fit_predict(df_scaled)

# Add cluster labels to the dataset
df_clean['DBSCAN_Cluster'] = clusters_dbscan

# Visualize DBSCAN clusters
plt.figure(figsize=(10, 7))
plt.scatter(df_clean['age'], df_clean['Purchase_Frequency'],
c=clusters_dbscan, cmap='plasma')
plt.title('DBSCAN Clustering')
plt.xlabel('Age')
plt.ylabel('Purchase Frequency')
plt.show()
```



Analysis:

- Cluster 0: Consistent spenders with high purchase frequency.

- Cluster 1: Irregular, low-frequency buyers.
- Outliers: Customers with unusual purchasing patterns, possibly anomalous behavior or new custom.

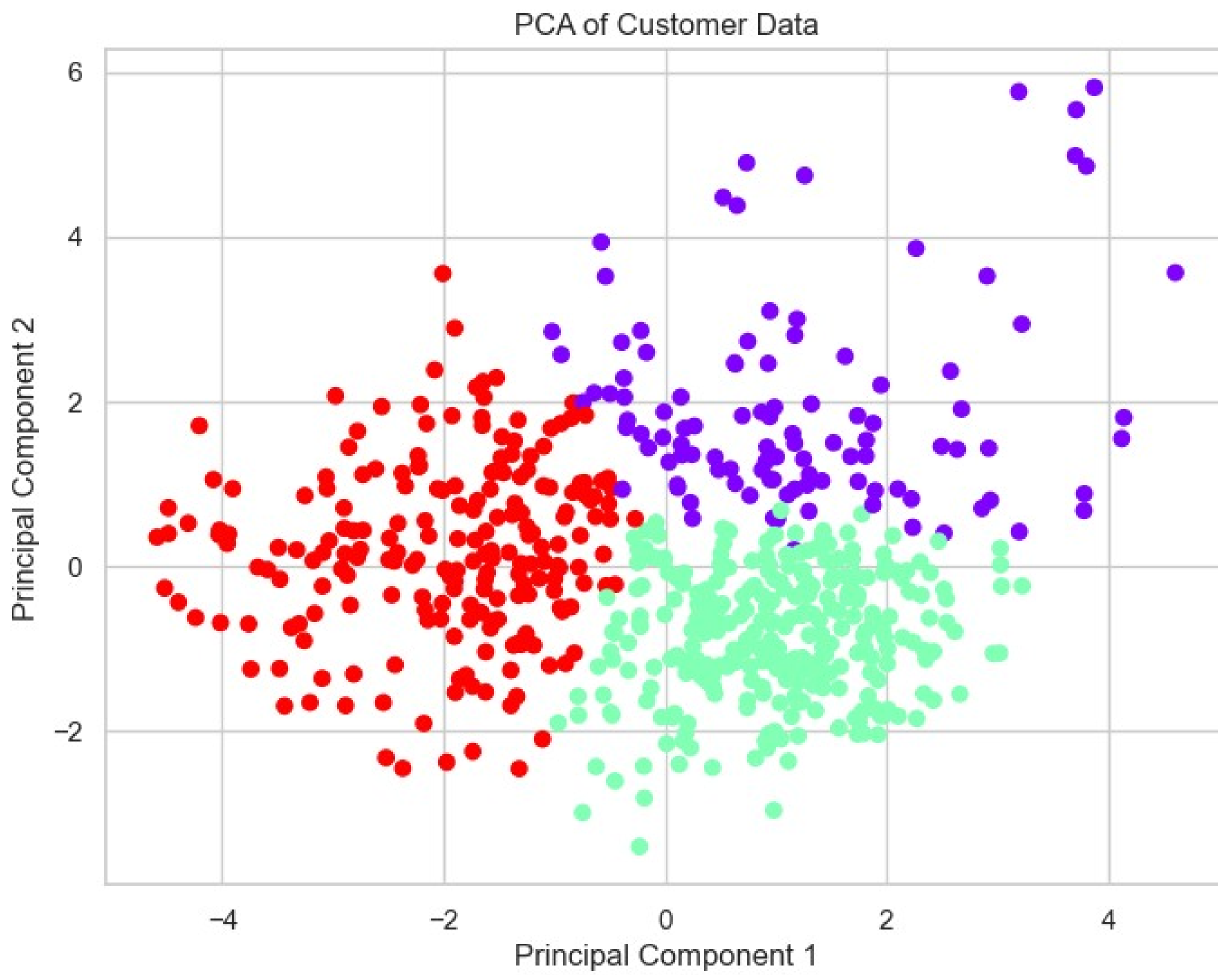
PCA (Principal Component Analysis)

PCA is not a clustering algorithm but a dimensionality reduction technique that helps visualize high-dimensional data by projecting it onto two principal components. This helps in visualizing the clusters formed by K-Means or other clustering algorithms.

Steps:

- Reduce the dimensionality of the dataset to two components.
- Visualize clusters in a 2D space.

```
# PCA (Principal Component Analysis):

from sklearn.decomposition import PCA

# Apply PCA to reduce dimensions to 2
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

# Visualize the data in 2D after PCA
plt.figure(figsize=(8, 6))
plt.scatter(df_pca[:, 0], df_pca[:, 1], c=kmeans.labels_,
cmap='rainbow')
plt.title('PCA of Customer Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



Analysis:

- The PCA plot shows how well-separated the clusters are in a two-dimensional space.
- This is a useful tool for validating clustering results and identifying overlaps between clusters.

Agglomerative Clustering:

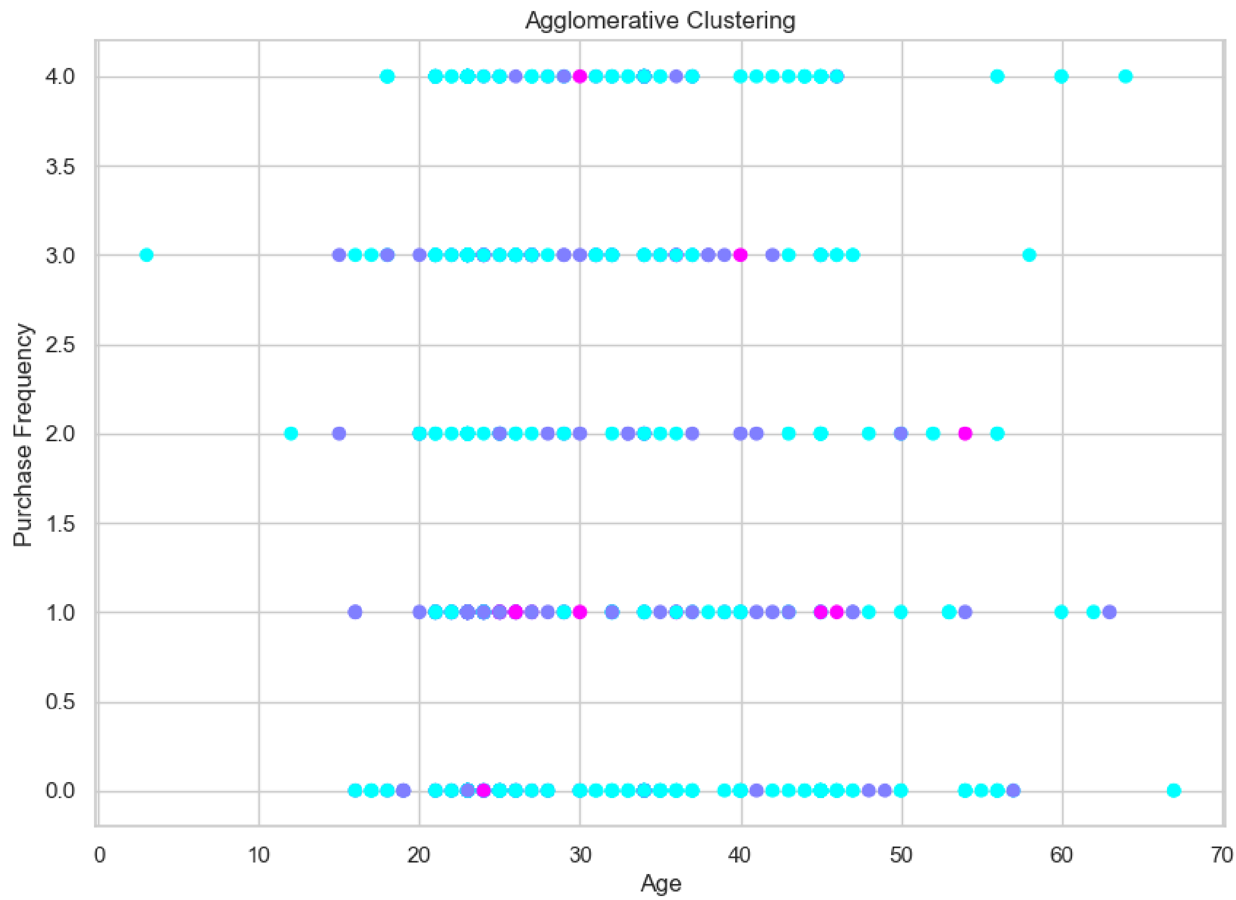
It is a bottom-up approach to hierarchical clustering, where each customer starts as its own cluster, and clusters are merged based on similarity until one single cluster remains.

Steps:

- Use Ward's method to minimize within-cluster variance.
- Plot a dendrogram to visualize the hierarchical structure of clusters.
- Cut the dendrogram to assign customers to a specific number of clusters.

```
# Agglomerative Clustering:

from sklearn.cluster import AgglomerativeClustering
```

```
# Apply Agglomerative Clustering
agglo = AgglomerativeClustering(n_clusters=3)
clusters_agglo = agglo.fit_predict(df_scaled)

# Add cluster labels to the dataset
df_clean['Agglomerative_Cluster'] = clusters_agglo

# Visualize Agglomerative clusters
plt.figure(figsize=(10, 7))
plt.scatter(df_clean['age'], df_clean['Purchase_Frequency'],
c=clusters_agglo, cmap='cool')
plt.title('Agglomerative Clustering')
plt.xlabel('Age')
plt.ylabel('Purchase Frequency')
plt.show()
```



Analysis:

- Cluster 1: Customers who frequently purchase low-cost products.
- Cluster 2: High-value but infrequent buyers.
- Cluster 3: Moderate frequency and medium-value buyers.

Model Comparisons

- **K-Means Clustering:** Offers efficient segmentation based on spending and purchase frequency but requires a predefined number of clusters.
- **Agglomerative Clustering:** Provides a hierarchical view of customer relationships, allowing flexibility in the number of clusters. The dendrogram helps visualize the clustering structure.
- **DBSCAN:** Suitable for detecting noise (outliers) and identifying clusters of irregular shapes without predefining the number of clusters. Ideal for datasets with non-uniform cluster shapes.
- **PCA:** Helps in visualizing and validating clusters in reduced dimensions, offering insight into the relationships between customers in two principal components.

Each clustering technique has its strengths and weaknesses. By combining them, businesses can better understand their customer base and develop effective marketing strategies tailored to distinct customer segments. For example, high-frequency buyers can receive loyalty programs, while outliers may need special offers to encourage engagement.

Other Models Evaluation

```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder,
StandardScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import train_test_split, KFold,
cross_val_score, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostClassifier,
GradientBoostingClassifier, RandomForestClassifier,
ExtraTreesClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# split input and output variable
x = df.drop(['Gender'], axis=1)
y = df['Gender']

# convert to string
x = x.astype(str)
```

Catagorical Encoding

```python
# prepare input data
def prepare_inputs(x_train, x_test):
    oe = OrdinalEncoder(handle_unknown='use_encoded_value',
unknown_value=-1)
    oe.fit(x_train)
    x_train_enc = oe.transform(x_train)
    x_test_enc = oe.transform(x_test)
    return x_train_enc, x_test_enc
```

Target Encoding

```python
# prepare target
def prepare_targets(y_train, y_test):
    le = LabelEncoder()
    le.fit(y_train)
    y_train_enc = le.transform(y_train)
    y_test_enc = le.transform(y_test)
    return y_train_enc, y_test_enc
```

Feature Scaling

```python
# feature selection
def select_features(x_train, y_train, x_test):
    fs = SelectKBest(score_func=chi2, k='all')
    fs.fit(x_train, y_train)
    x_train_fs = fs.transform(x_train)
    x_test_fs = fs.transform(x_test)
    return x_train_fs, x_test_fs, fs

# split the dataset
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=23)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

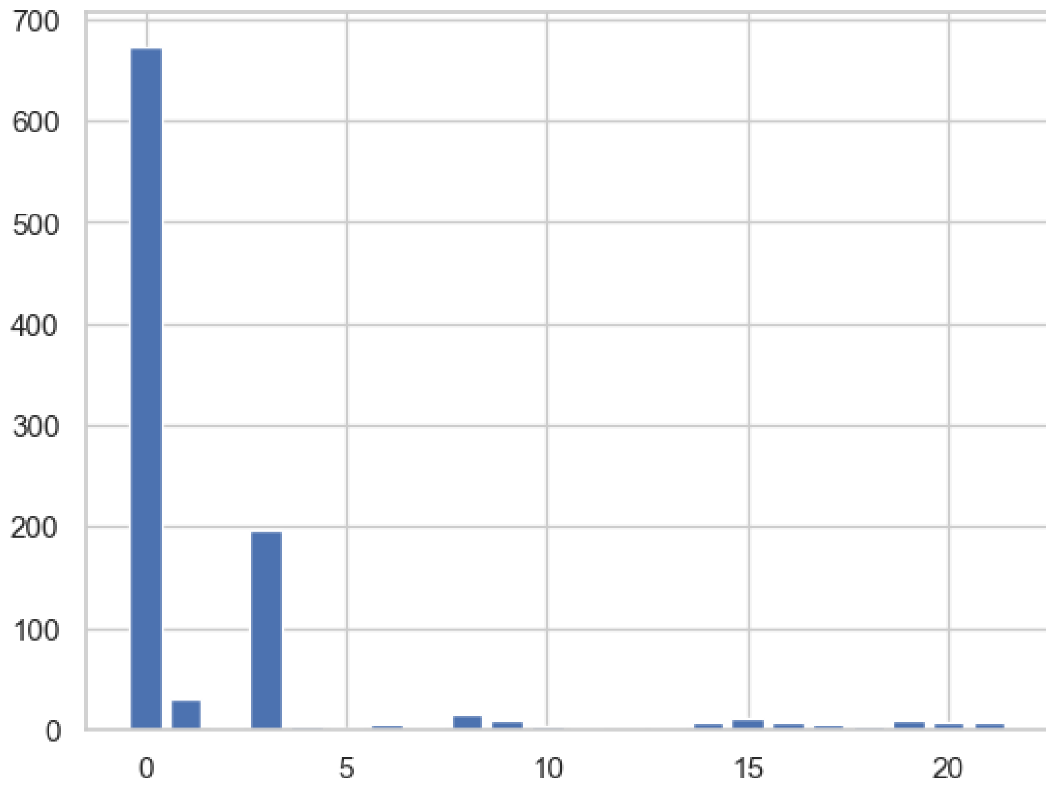
```
((481, 22), (121, 22), (481,), (121,))
```

```python
# prepare input data
x_train_enc, x_test_enc = prepare_inputs(x_train, x_test)

# prepare output data
y_train_enc, y_test_enc = prepare_targets(y_train, y_test)

# feature selection
x_train_fs, x_test_fs, fs = select_features(x_train_enc, y_train_enc,
x_test_enc)
```

```python
# what are scores for the features
for i in range(len(fs.scores_)):
    print('Feature %d: %f' % (i, fs.scores_[i]))
```

```
Feature 0: 673.072125
Feature 1: 30.231644
Feature 2: 0.982145
Feature 3: 196.669813
Feature 4: 2.882849
Feature 5: 1.639115
Feature 6: 5.355460
Feature 7: 1.663367
Feature 8: 13.991366
Feature 9: 9.591061
Feature 10: 4.185675
Feature 11: 1.355832
Feature 12: 2.441824
Feature 13: 0.706066
Feature 14: 7.305193
Feature 15: 10.537447
Feature 16: 7.709323
Feature 17: 4.472732
Feature 18: 3.420467
Feature 19: 10.136995
Feature 20: 7.125628
Feature 21: 6.940209
```

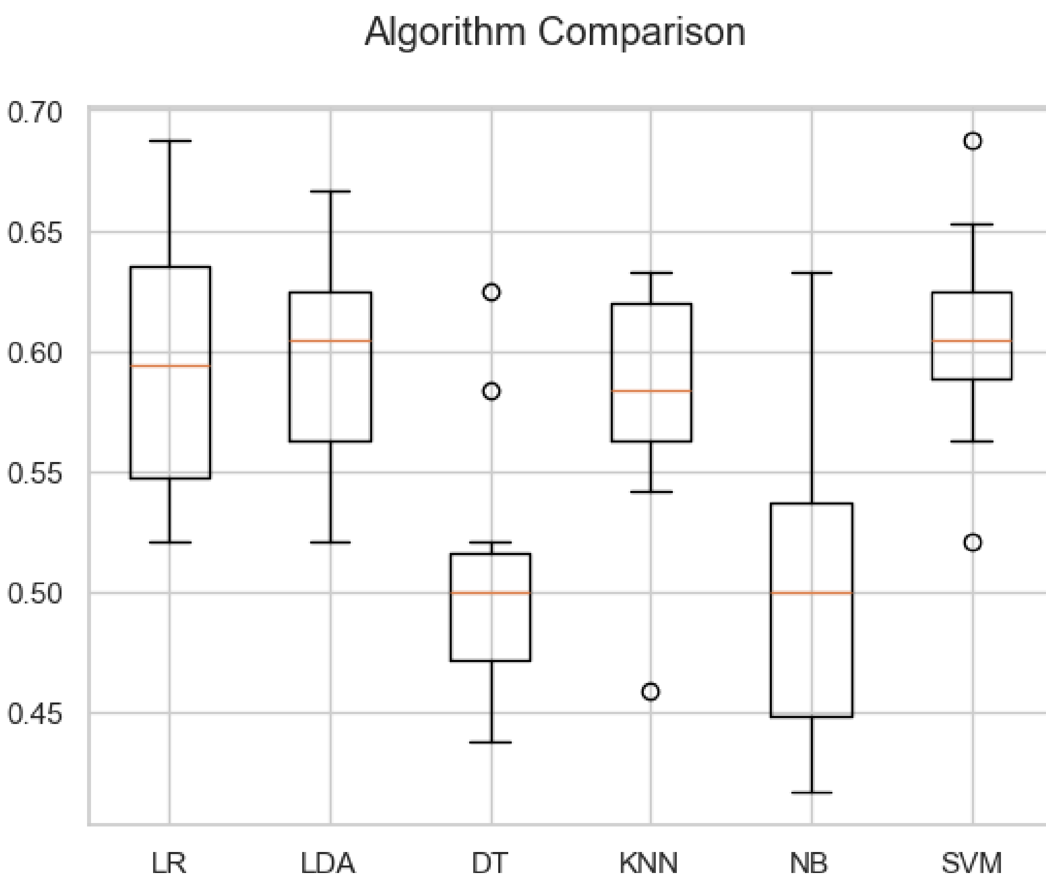
```python
# plot the scores
plt.bar([i for i in range(len(fs.scores_))], fs.scores_)
plt.show()
```



Algorithms

```
# spot check algorithms
models = []
models.append(("LR", LogisticRegression(solver="lbfgs",
max_iter=1000)))
models.append(("LDA", LinearDiscriminantAnalysis()))
models.append(("DT", DecisionTreeClassifier()))
models.append(("KNN", KNeighborsClassifier()))
models.append(("NB", GaussianNB()))
models.append(("SVM", SVC()))

# evaluate each model in turn
results = []
names = []
for name, model in models:
  cv = KFold(n_splits=10, random_state=None)
  scores = cross_val_score(model, x_train_enc, y_train_enc,
scoring="accuracy", cv=cv)
  names.append(name)
  results.append(scores)
  print("%s %.2f (%.2f)" % (name, scores.mean(), scores.std()))

LR 0.59 (0.05)
LDA 0.60 (0.04)
DT 0.51 (0.05)
```

```
KNN 0.58 (0.05)
NB 0.50 (0.06)
SVM 0.61 (0.04)

# compare algorithm
fig = plt.figure()
fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Hyperparameter Tuning with grid search

```
# tunned with svm
c_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
param_grid = dict(C=c_values, kernel=kernel_values)
model = SVC()
cv = KFold(n_splits=10, random_state=None)
grid = GridSearchCV(estimator=model, param_grid=param_grid,
scoring="accuracy", cv=cv)
```

```python
grid_result = grid.fit(x_train_enc, y_train_enc)
print("Best: %.3f using %r" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, std, param in zip(means, stds, params):
  print("%.3f (%.3f) with %r" % (mean, std, param))
```

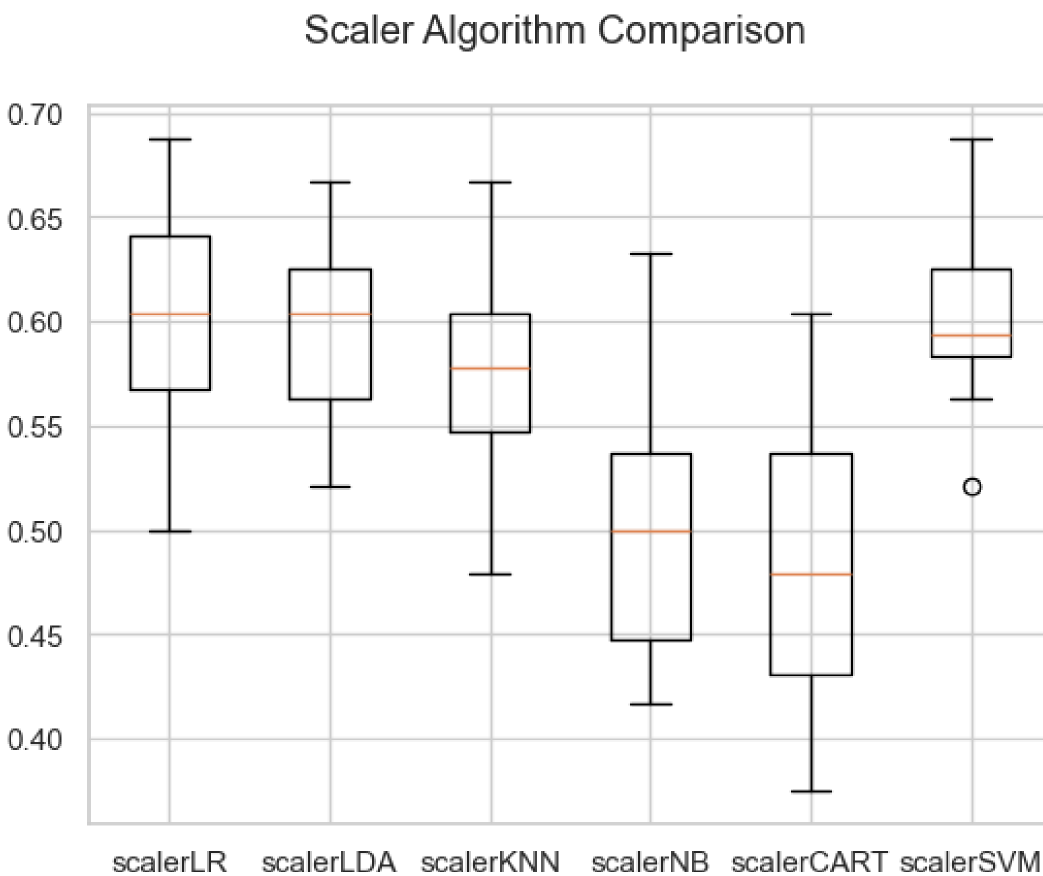
```
Best: 0.607 using {'C': 0.1, 'kernel': 'poly'}
0.605 (0.043) with {'C': 0.1, 'kernel': 'linear'}
0.607 (0.044) with {'C': 0.1, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.1, 'kernel': 'rbf'}
0.607 (0.044) with {'C': 0.1, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.3, 'kernel': 'linear'}
0.607 (0.044) with {'C': 0.3, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.3, 'kernel': 'rbf'}
0.551 (0.068) with {'C': 0.3, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.5, 'kernel': 'linear'}
0.607 (0.044) with {'C': 0.5, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.5, 'kernel': 'rbf'}
0.528 (0.082) with {'C': 0.5, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.7, 'kernel': 'linear'}
0.607 (0.044) with {'C': 0.7, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.7, 'kernel': 'rbf'}
0.526 (0.089) with {'C': 0.7, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.9, 'kernel': 'linear'}
0.607 (0.044) with {'C': 0.9, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.9, 'kernel': 'rbf'}
0.519 (0.090) with {'C': 0.9, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 1.0, 'kernel': 'linear'}
0.607 (0.044) with {'C': 1.0, 'kernel': 'poly'}
0.607 (0.044) with {'C': 1.0, 'kernel': 'rbf'}
0.519 (0.090) with {'C': 1.0, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 1.3, 'kernel': 'linear'}
0.607 (0.044) with {'C': 1.3, 'kernel': 'poly'}
0.607 (0.044) with {'C': 1.3, 'kernel': 'rbf'}
0.513 (0.086) with {'C': 1.3, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 1.5, 'kernel': 'linear'}
0.607 (0.044) with {'C': 1.5, 'kernel': 'poly'}
0.607 (0.044) with {'C': 1.5, 'kernel': 'rbf'}
0.513 (0.086) with {'C': 1.5, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 1.7, 'kernel': 'linear'}
0.607 (0.044) with {'C': 1.7, 'kernel': 'poly'}
0.607 (0.044) with {'C': 1.7, 'kernel': 'rbf'}
0.507 (0.088) with {'C': 1.7, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 2.0, 'kernel': 'linear'}
0.607 (0.044) with {'C': 2.0, 'kernel': 'poly'}
0.607 (0.044) with {'C': 2.0, 'kernel': 'rbf'}
0.511 (0.087) with {'C': 2.0, 'kernel': 'sigmoid'}
```

```python
# standardize with basic algorithms
pipelines = []
pipelines.append(("scalerLR", Pipeline([("scaler", StandardScaler()),
("LR", LogisticRegression(solver="lbfgs", max_iter=1000))])))
pipelines.append(("scalerLDA", Pipeline([("scaler", StandardScaler()),
("LDA", LinearDiscriminantAnalysis())])))
pipelines.append(("scalerKNN", Pipeline([("scaler", StandardScaler()),
("KNN", KNeighborsClassifier())])))
pipelines.append(("scalerNB", Pipeline([("scaler", StandardScaler()),
("NB", GaussianNB())])))
pipelines.append(("scalerCART", Pipeline([("scaler",
StandardScaler()), ("CART", DecisionTreeClassifier())])))
pipelines.append(("scalerSVM", Pipeline([("scaler", StandardScaler()),
("SVM", SVC())])))
```

```python
# evaluate algorithm with standardize
names = []
results = []
for name, model in pipelines:
  cv = KFold(n_splits=10, random_state=None)
  scores = cross_val_score(model, x_train_enc, y_train_enc,
scoring="accuracy", cv=cv)
  names.append(name)
  results.append(scores)
  print("%s %.3f (%.3f)" % (name, scores.mean(), scores.std()))
```

```
scalerLR 0.601 (0.054)
scalerLDA 0.599 (0.045)
scalerKNN 0.576 (0.051)
scalerNB 0.501 (0.061)
scalerCART 0.485 (0.069)
scalerSVM 0.603 (0.045)
```

```python
# compare algorithm
fig = plt.figure()
fig.suptitle("Scaler Algorithm Comparison")
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Ensamble Method

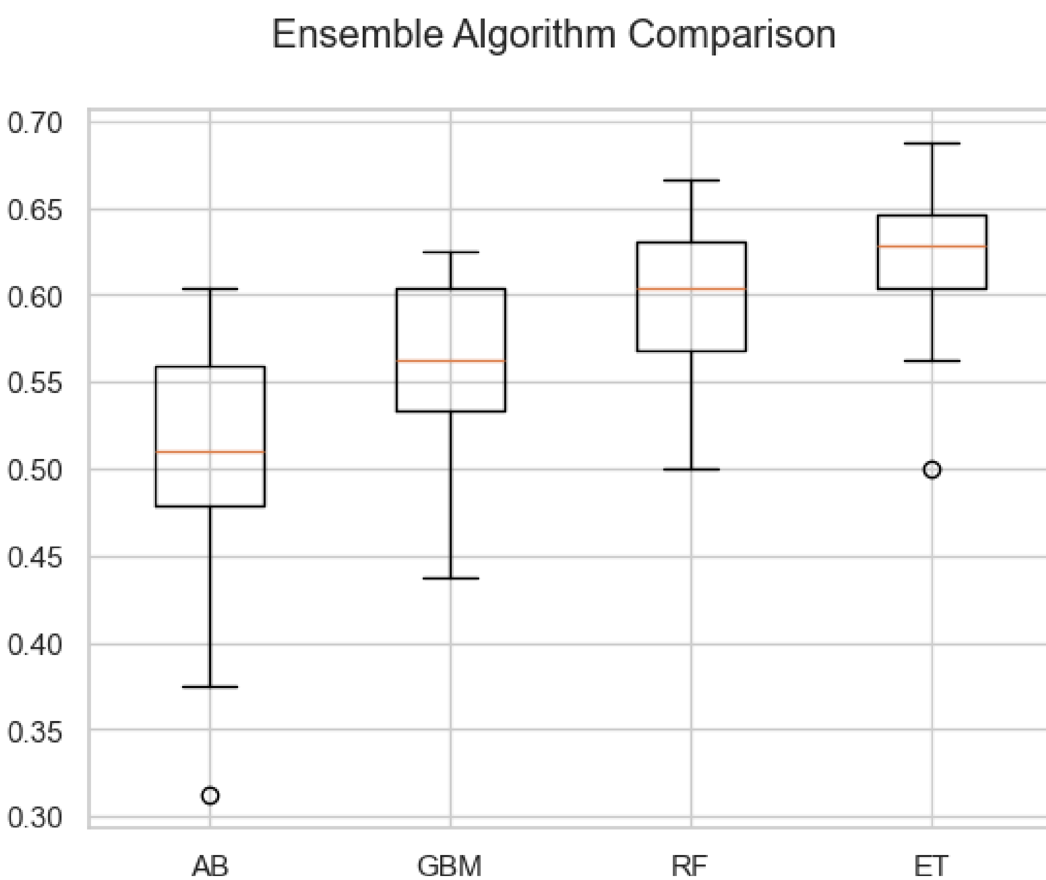
```
# tuned with svm standardize
scaler = StandardScaler()
rescaled_x = scaler.fit_transform(x_train_enc)
c_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
param_grid = dict(C=c_values, kernel=kernel_values)
model = SVC()
cv = KFold(n_splits=10, random_state=None)
grid = GridSearchCV(estimator=model, param_grid=param_grid,
scoring="accuracy", cv=cv)
grid_result = grid.fit(rescaled_x, y_train_enc)
print("Best: %.3f using %r" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, std, param in zip(means, stds, params):
  print("%.3f (%.3f) with %r" % (mean, std, param))

Best: 0.609 using {'C': 1.7, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.1, 'kernel': 'linear'}
```

```
0.607 (0.044) with {'C': 0.1, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.1, 'kernel': 'rbf'}
0.607 (0.044) with {'C': 0.1, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.3, 'kernel': 'linear'}
0.605 (0.043) with {'C': 0.3, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.3, 'kernel': 'rbf'}
0.607 (0.044) with {'C': 0.3, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.5, 'kernel': 'linear'}
0.607 (0.040) with {'C': 0.5, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.5, 'kernel': 'rbf'}
0.607 (0.044) with {'C': 0.5, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.7, 'kernel': 'linear'}
0.603 (0.045) with {'C': 0.7, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.7, 'kernel': 'rbf'}
0.607 (0.044) with {'C': 0.7, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 0.9, 'kernel': 'linear'}
0.607 (0.044) with {'C': 0.9, 'kernel': 'poly'}
0.607 (0.044) with {'C': 0.9, 'kernel': 'rbf'}
0.607 (0.044) with {'C': 0.9, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 1.0, 'kernel': 'linear'}
0.605 (0.047) with {'C': 1.0, 'kernel': 'poly'}
0.603 (0.045) with {'C': 1.0, 'kernel': 'rbf'}
0.607 (0.044) with {'C': 1.0, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 1.3, 'kernel': 'linear'}
0.601 (0.048) with {'C': 1.3, 'kernel': 'poly'}
0.592 (0.047) with {'C': 1.3, 'kernel': 'rbf'}
0.607 (0.044) with {'C': 1.3, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 1.5, 'kernel': 'linear'}
0.603 (0.048) with {'C': 1.5, 'kernel': 'poly'}
0.597 (0.046) with {'C': 1.5, 'kernel': 'rbf'}
0.607 (0.042) with {'C': 1.5, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 1.7, 'kernel': 'linear'}
0.601 (0.048) with {'C': 1.7, 'kernel': 'poly'}
0.586 (0.052) with {'C': 1.7, 'kernel': 'rbf'}
0.609 (0.045) with {'C': 1.7, 'kernel': 'sigmoid'}
0.605 (0.043) with {'C': 2.0, 'kernel': 'linear'}
0.599 (0.049) with {'C': 2.0, 'kernel': 'poly'}
0.590 (0.053) with {'C': 2.0, 'kernel': 'rbf'}
0.607 (0.042) with {'C': 2.0, 'kernel': 'sigmoid'}

# ensemble methods
ensembles = []
ensembles.append(('AB', AdaBoostClassifier()))
ensembles.append(('GBM', GradientBoostingClassifier()))
ensembles.append(('RF', RandomForestClassifier()))
ensembles.append(('ET', ExtraTreesClassifier()))

# evaluate each model with ensemble
results = []
names = []
```

```
for name, model in ensembles:
  kfold = KFold(n_splits=10, random_state=None)
  cv_results = cross_val_score(model, x_train_enc, y_train_enc,
cv=kfold, scoring="accuracy")
  results.append(cv_results)
  names.append(name)
  print("%s: %.3f (%.3f)" % (name, cv_results.mean(),
cv_results.std()))
```

```
AB: 0.497 (0.088)
GBM: 0.557 (0.060)
RF: 0.599 (0.051)
ET: 0.615 (0.050)
```

```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Ensemble Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



```
# tuned with ETC
estimator = [10, 20, 50, 100, 200, 500, 1000]
criterion= ['gini', 'entropy']
param_grid = dict(n_estimators=estimator, criterion=criterion)
model = ExtraTreesClassifier()
cv = KFold(n_splits=10, random_state=None)
grid = GridSearchCV(estimator=model, param_grid=param_grid,
scoring="accuracy", cv=cv)
grid_result = grid.fit(x_train_enc, y_train_enc)
print("Best: %.3f using %r" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, std, param in zip(means, stds, params):
  print("%.3f (%.3f) with %r" % (mean, std, param))
```

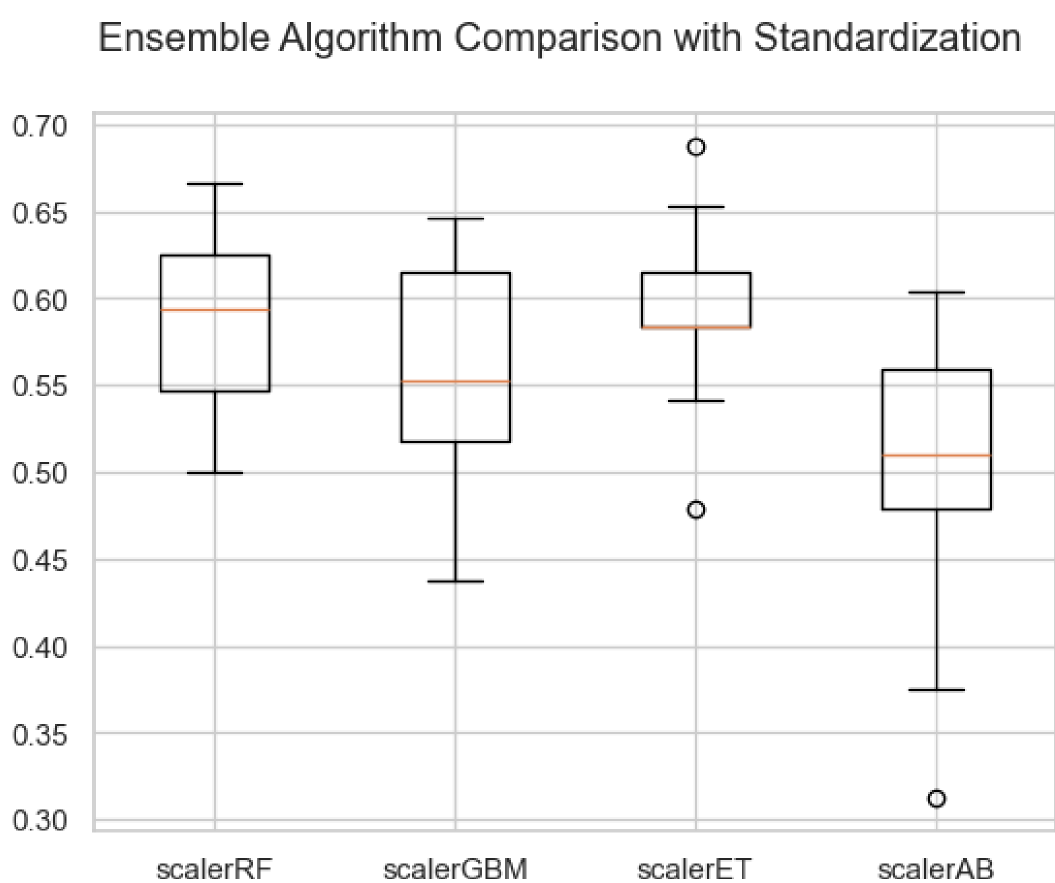
```
Best: 0.605 using {'criterion': 'gini', 'n_estimators': 1000}
0.557 (0.058) with {'criterion': 'gini', 'n_estimators': 10}
0.586 (0.053) with {'criterion': 'gini', 'n_estimators': 20}
0.574 (0.049) with {'criterion': 'gini', 'n_estimators': 50}
0.599 (0.059) with {'criterion': 'gini', 'n_estimators': 100}
0.603 (0.047) with {'criterion': 'gini', 'n_estimators': 200}
0.603 (0.050) with {'criterion': 'gini', 'n_estimators': 500}
0.605 (0.051) with {'criterion': 'gini', 'n_estimators': 1000}
0.561 (0.063) with {'criterion': 'entropy', 'n_estimators': 10}
0.578 (0.057) with {'criterion': 'entropy', 'n_estimators': 20}
0.588 (0.053) with {'criterion': 'entropy', 'n_estimators': 50}
0.605 (0.054) with {'criterion': 'entropy', 'n_estimators': 100}
0.605 (0.053) with {'criterion': 'entropy', 'n_estimators': 200}
0.601 (0.055) with {'criterion': 'entropy', 'n_estimators': 500}
0.603 (0.055) with {'criterion': 'entropy', 'n_estimators': 1000}
```

```
# ensembles with standardize
ensembles = []
ensembles.append(("scalerRF", Pipeline([("scaler", StandardScaler()),
("RF", RandomForestClassifier())])))
ensembles.append(("scalerGBM", Pipeline([("scaler", StandardScaler()),
("GBM", GradientBoostingClassifier())])))
ensembles.append(("scalerET", Pipeline([("scaler", StandardScaler()),
("ET", ExtraTreesClassifier())])))
ensembles.append(("scalerAB", Pipeline([("scaler", StandardScaler()),
("AB", AdaBoostClassifier())])))

# evaluate each model with ensemble
results = []
names = []
for name, model in ensembles:
  kfold = KFold(n_splits=10, random_state=None)
  cv_results = cross_val_score(model, x_train_enc, y_train_enc,
```

```
cv=kfold, scoring="accuracy")
  results.append(cv_results)
  names.append(name)
  print("%s: %.3f (%.3f)" % (name, cv_results.mean(),
cv_results.std()))
```

```
scalerRF: 0.588 (0.053)
scalerGBM: 0.555 (0.064)
scalerET: 0.590 (0.054)
scalerAB: 0.497 (0.088)
```

```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Ensemble Algorithm Comparison with Standardization')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```




```
# tuned with RF
scaler = StandardScaler()
rescaled_x = scaler.fit_transform(x_train_enc)
```

```
estimator = [10, 20, 50, 100, 200, 500, 1000]
criterion= ['gini', 'entropy']
param_grid = dict(n_estimators=estimator, criterion=criterion)#
prepare the model
model = ExtraTreesClassifier(n_estimators=50, criterion="entropy")
model.fit(x_train_enc, y_train_enc)

model = ExtraTreesClassifier(random_state=42)
cv = KFold(n_splits=10, random_state=None)
grid = GridSearchCV(estimator=model, param_grid=param_grid,
scoring="accuracy", cv=cv)
grid_result = grid.fit(rescaled_x, y_train_enc)
print("Best: %.3f using %r" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, std, param in zip(means, stds, params):
  print("%.3f (%.3f) with %r" % (mean, std, param))
```

```
Best: 0.607 using {'criterion': 'entropy', 'n_estimators': 100}
0.549 (0.061) with {'criterion': 'gini', 'n_estimators': 10}
0.576 (0.052) with {'criterion': 'gini', 'n_estimators': 20}
0.588 (0.051) with {'criterion': 'gini', 'n_estimators': 50}
0.597 (0.064) with {'criterion': 'gini', 'n_estimators': 100}
0.592 (0.062) with {'criterion': 'gini', 'n_estimators': 200}
0.599 (0.056) with {'criterion': 'gini', 'n_estimators': 500}
0.599 (0.053) with {'criterion': 'gini', 'n_estimators': 1000}
0.580 (0.049) with {'criterion': 'entropy', 'n_estimators': 10}
0.594 (0.059) with {'criterion': 'entropy', 'n_estimators': 20}
0.599 (0.056) with {'criterion': 'entropy', 'n_estimators': 50}
0.607 (0.049) with {'criterion': 'entropy', 'n_estimators': 100}
0.603 (0.051) with {'criterion': 'entropy', 'n_estimators': 200}
0.603 (0.057) with {'criterion': 'entropy', 'n_estimators': 500}
0.607 (0.055) with {'criterion': 'entropy', 'n_estimators': 1000}
```

Finalize the model

```
# prepare the model
model = ExtraTreesClassifier(n_estimators=50, criterion="entropy")
model.fit(x_train_enc, y_train_enc)
```

```
ExtraTreesClassifier(criterion='entropy', n_estimators=50)
```

```
# estimate accuracy on validation dataset
predictions = model.predict(x_test_enc)
print(accuracy_score(y_test_enc, predictions))
print(confusion_matrix(y_test_enc, predictions))
print(classification_report(y_test_enc, predictions))
```

```
0.4793388429752066
[[57  2  0  1]
 [33  1  0  1]
 [ 4  0  0  0]
 [21  1  0  0]]
              precision    recall  f1-score   support

           0       0.50      0.95      0.65        60
           1       0.25      0.03      0.05        35
           2       0.00      0.00      0.00         4
           3       0.00      0.00      0.00        22

    accuracy                           0.48       121
   macro avg       0.19      0.24      0.18       121
weighted avg       0.32      0.48      0.34       121
```

Recommendations:

Targeted Marketing Strategies Based on Cluster Analysis:

Frequent Buyers, Moderate Spending:

- Loyalty Programs: Offer points or discounts for frequent purchases.
- Cross-Selling: Recommend complementary products to increase average order value.

Low-Frequency, High-Spending Buyers:

- Exclusive Offers: Provide limited-time deals on premium products to encourage more frequent purchases.
- VIP Treatment: Send personalized offers or early access to sales.

Moderate Buyers, Balanced Spending:

- Personalized Discounts: Offer tailored promotions based on past purchases.
- Product Bundling: Encourage bundled purchases to maximize value.

These strategies aim to drive engagement, retention, and increased spending across each segment.

References

- Mighty Itauma Itauma, PhD. (2024). Machine Learning using Python. https://amightyo.quarto.pub/machine-learning-using-python/