UNDERSTANDING CONSUMER BEHAVIOUR ANALYSIS. A MACHINE LEARNING APPROACH FOR E-COMMERCE TRENDS.

Student Name: Cynthia Wanyeki

Technical Mentor: Nikita Njoroge, Diana Mong'ina, Lucille Kaleha.

Phase: Phase 3 Project

Deadline: 1st December 2023.

1.) INTRODUCTION.

The aim of this project is to analyse consumer behaviour on E-Commerce platform and build a model that can improve engagement on these platforms in order to make more sales.

The online market is the largest platform and E-Commerce when tapped into really well is a amazing way to make more sales.

This analysis uses a real-time project dataset from Kaggle for Amazon Customer Behavior Survey and Shopping Behavior Survey.

1.1) BUSINESS UDERSTANDING

The e-commerce industry has witnessed unprecedented growth in recent years, with a surge in online shopping platforms providing consumers with a plethora of choices. Understanding customer behavior on these platforms is crucial for businesses to enhance user experience, optimize marketing strategies, and ultimately boost sales. This project aims to delve into the nuances of customer behavior on e-commerce platforms, uncovering patterns and insights that can inform strategic decision-making.

1.2) PROBLEM STATEMENT.

Despite the rapid growth of the e-commerce sector, businesses face challenges in comprehensively understanding customer behavior. The lack of detailed insights into user preferences, navigation patterns, and purchase decision factors hinders the ability to tailor services and offerings effectively. This project aims to address this gap by conducting a thorough analysis of customer behavior on e-commerce platforms, identifying pain points, and proposing solutions for a more personalized and seamless user experience.

1.3) OBJECTIVES.

Customer Segmentation: Identify and classify different customer segments based on their behavior, preferences, and buying patterns.

User Journey Analysis: Map out the typical user journey on the e-commerce platform, highlighting key touchpoints, drop-offs, and areas of improvement.

Product Affinity Analysis: Understand the relationships between products, analyzing which items are often purchased together or in succession.

Conversion Rate Optimization: Identify factors influencing conversion rates and propose strategies to optimize the conversion funnel.

Predictive Analytics: Utilize predictive modeling to forecast future trends in customer behavior and anticipate potential challenges.

2.) LIBRARIES.

2.1) Importing Libraries

```
#import libraries
import warnings
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import tree
from sklearn.svm import SVC
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import RFE
pd.set_option('display.max_columns', None)
from sklearn.naive_bayes import GaussianNB
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree,
export_text
from sklearn.preprocessing import LabelEncoder, StandardScaler,
OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score
from sklearn.metrics import accuracy_score, classification_report,\
confusion_matrix, ConfusionMatrixDisplay, roc_curve, roc_auc_score,
auc, f1_score
from sklearn.ensemble import RandomForestClassifier,
BaggingClassifier, AdaBoostClassifier,\
GradientBoostingClassifier
import xgboost as xgb
```

2.2) LOADING DATASETS

```
#load the Amazon_Customer_Behavior_Survey dataset
df_Amazon= pd.read_csv('/content/Amazon Customer Behavior Survey.csv')
df_Amazon.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 Add_to_Cart_Browsing
Cart_Completion_Frequency  \
0                            1                  Yes
Sometimes
1                            1                  Yes
Often
2                            2                  Yes
Sometimes
3                            5                Maybe
Sometimes
4                            1                  Yes
Sometimes

         Cart_Abandonment_Factors Saveforlater_Frequency
Review_Left  \
0  Found a better price elsewhere              Sometimes         Yes

1              High shipping costs                 Rarely          No

2  Found a better price elsewhere                 Rarely          No

3  Found a better price elsewhere              Sometimes         Yes

4              High shipping costs                 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
```

3. DATA UNDERSTANDING

3.1) Understanding column names, Data types and Summary Statistics for the Amazon Dataset

```
# Display basic information about the Amazon dataset
df_Amazon.info()

# Check for missing values
df_Amazon.isnull().sum()

# Explore statistics of numerical columns
df_Amazon.describe()

# Explore unique values in categorical columns
for column in df_Amazon.select_dtypes(include='object').columns:
    print(f"{column}: {df_Amazon[column].unique()}")
```

```
<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
Timestamp: ['2023/06/04 1:28:19 PM GMT+5:30' '2023/06/04 2:30:44 PM
GMT+5:30'
 '2023/06/04 5:04:56 PM GMT+5:30' '2023/06/04 5:13:00 PM GMT+5:30'
 '2023/06/04 5:28:06 PM GMT+5:30' '2023/06/04 6:01:59 PM GMT+5:30'
 '2023/06/04 6:31:41 PM GMT+5:30' '2023/06/04 7:13:12 PM GMT+5:30'
 '2023/06/04 7:23:21 PM GMT+5:30' '2023/06/04 7:33:12 PM GMT+5:30'
 '2023/06/04 7:45:33 PM GMT+5:30' '2023/06/04 7:48:31 PM GMT+5:30'
 '2023/06/04 8:00:11 PM GMT+5:30' '2023/06/04 8:01:45 PM GMT+5:30'
 '2023/06/04 8:02:28 PM GMT+5:30' '2023/06/04 8:20:42 PM GMT+5:30'
 '2023/06/04 8:39:53 PM GMT+5:30' '2023/06/04 8:45:45 PM GMT+5:30'
 '2023/06/04 8:46:02 PM GMT+5:30' '2023/06/04 8:48:59 PM GMT+5:30'
 '2023/06/04 8:49:49 PM GMT+5:30' '2023/06/04 8:52:49 PM GMT+5:30'
 '2023/06/04 8:54:36 PM GMT+5:30' '2023/06/04 9:10:08 PM GMT+5:30'
 '2023/06/04 9:10:51 PM GMT+5:30' '2023/06/04 9:18:12 PM GMT+5:30'
 '2023/06/04 9:22:58 PM GMT+5:30' '2023/06/04 9:29:59 PM GMT+5:30'
 '2023/06/04 9:40:54 PM GMT+5:30' '2023/06/04 9:48:15 PM GMT+5:30'
 '2023/06/04 10:13:28 PM GMT+5:30' '2023/06/04 10:24:47 PM GMT+5:30'
 '2023/06/04 10:37:00 PM GMT+5:30' '2023/06/04 11:09:27 PM GMT+5:30'
 '2023/06/04 11:12:15 PM GMT+5:30' '2023/06/04 11:24:41 PM GMT+5:30'
 '2023/06/04 11:27:53 PM GMT+5:30' '2023/06/04 11:38:30 PM GMT+5:30'
 '2023/06/04 11:57:10 PM GMT+5:30' '2023/06/05 12:02:56 AM GMT+5:30'
 '2023/06/05 12:05:20 AM GMT+5:30' '2023/06/05 12:36:02 AM GMT+5:30'
 '2023/06/05 12:37:54 AM GMT+5:30' '2023/06/05 12:39:14 AM GMT+5:30'
 '2023/06/05 12:57:05 AM GMT+5:30' '2023/06/05 2:56:10 AM GMT+5:30'
 '2023/06/05 4:26:25 AM GMT+5:30' '2023/06/05 7:16:40 AM GMT+5:30'
 '2023/06/05 8:43:26 AM GMT+5:30' '2023/06/05 9:20:17 AM GMT+5:30'
 '2023/06/05 9:56:58 AM GMT+5:30' '2023/06/05 10:36:38 AM GMT+5:30'
 '2023/06/05 10:37:36 AM GMT+5:30' '2023/06/05 10:53:16 AM GMT+5:30'
 '2023/06/05 11:20:41 AM GMT+5:30' '2023/06/05 11:54:52 AM GMT+5:30'
 '2023/06/05 1:10:59 PM GMT+5:30' '2023/06/05 1:27:08 PM GMT+5:30'
 '2023/06/05 1:28:44 PM GMT+5:30' '2023/06/05 1:32:57 PM GMT+5:30'
 '2023/06/05 1:35:41 PM GMT+5:30' '2023/06/05 1:35:44 PM GMT+5:30'
```

```
'2023/06/05 1:37:01 PM GMT+5:30' '2023/06/05 1:43:00 PM GMT+5:30'
'2023/06/05 1:45:30 PM GMT+5:30' '2023/06/05 1:54:28 PM GMT+5:30'
'2023/06/05 2:10:33 PM GMT+5:30' '2023/06/05 2:16:16 PM GMT+5:30'
'2023/06/05 2:17:45 PM GMT+5:30' '2023/06/05 2:21:37 PM GMT+5:30'
'2023/06/05 2:24:22 PM GMT+5:30' '2023/06/05 2:25:28 PM GMT+5:30'
'2023/06/05 2:28:10 PM GMT+5:30' '2023/06/05 2:46:25 PM GMT+5:30'
'2023/06/05 3:28:01 PM GMT+5:30' '2023/06/05 4:46:52 PM GMT+5:30'
'2023/06/05 4:49:43 PM GMT+5:30' '2023/06/05 4:50:33 PM GMT+5:30'
'2023/06/05 4:50:44 PM GMT+5:30' '2023/06/05 4:52:12 PM GMT+5:30'
'2023/06/05 6:49:25 PM GMT+5:30' '2023/06/05 7:04:21 PM GMT+5:30'
'2023/06/05 8:06:04 PM GMT+5:30' '2023/06/05 8:07:52 PM GMT+5:30'
'2023/06/05 8:44:09 PM GMT+5:30' '2023/06/05 8:46:09 PM GMT+5:30'
'2023/06/05 8:47:42 PM GMT+5:30' '2023/06/05 8:49:12 PM GMT+5:30'
'2023/06/05 8:50:24 PM GMT+5:30' '2023/06/05 8:52:01 PM GMT+5:30'
'2023/06/05 9:56:02 PM GMT+5:30' '2023/06/05 9:56:08 PM GMT+5:30'
'2023/06/05 9:58:17 PM GMT+5:30' '2023/06/05 10:04:32 PM GMT+5:30'
'2023/06/05 10:05:25 PM GMT+5:30' '2023/06/05 10:06:15 PM GMT+5:30'
'2023/06/05 10:06:46 PM GMT+5:30' '2023/06/05 10:07:31 PM GMT+5:30'
'2023/06/05 10:07:33 PM GMT+5:30' '2023/06/05 10:08:15 PM GMT+5:30'
'2023/06/05 10:09:03 PM GMT+5:30' '2023/06/06 8:46:03 AM GMT+5:30'
'2023/06/06 9:37:33 AM GMT+5:30' '2023/06/06 9:39:32 AM GMT+5:30'
'2023/06/06 9:45:55 AM GMT+5:30' '2023/06/06 9:47:31 AM GMT+5:30'
'2023/06/06 9:59:35 AM GMT+5:30' '2023/06/06 10:03:52 AM GMT+5:30'
'2023/06/06 10:07:39 AM GMT+5:30' '2023/06/06 10:17:06 AM GMT+5:30'
'2023/06/06 10:19:05 AM GMT+5:30' '2023/06/06 10:22:47 AM GMT+5:30'
'2023/06/06 10:36:14 AM GMT+5:30' '2023/06/06 10:46:31 AM GMT+5:30'
'2023/06/06 11:20:53 AM GMT+5:30' '2023/06/06 11:49:21 AM GMT+5:30'
'2023/06/06 12:00:16 PM GMT+5:30' '2023/06/06 12:52:14 PM GMT+5:30'
'2023/06/06 1:06:16 PM GMT+5:30' '2023/06/06 2:07:12 PM GMT+5:30'
'2023/06/06 3:02:38 PM GMT+5:30' '2023/06/06 4:25:49 PM GMT+5:30'
'2023/06/06 4:46:25 PM GMT+5:30' '2023/06/06 5:16:58 PM GMT+5:30'
'2023/06/06 6:32:00 PM GMT+5:30' '2023/06/06 6:32:36 PM GMT+5:30'
'2023/06/06 6:33:23 PM GMT+5:30' '2023/06/06 6:34:44 PM GMT+5:30'
'2023/06/06 6:35:14 PM GMT+5:30' '2023/06/06 6:35:33 PM GMT+5:30'
'2023/06/06 6:36:16 PM GMT+5:30' '2023/06/06 6:37:19 PM GMT+5:30'
'2023/06/06 6:37:26 PM GMT+5:30' '2023/06/06 6:38:03 PM GMT+5:30'
'2023/06/06 6:38:45 PM GMT+5:30' '2023/06/06 6:38:49 PM GMT+5:30'
'2023/06/06 6:39:26 PM GMT+5:30' '2023/06/06 6:40:27 PM GMT+5:30'
'2023/06/06 6:40:30 PM GMT+5:30' '2023/06/06 6:41:11 PM GMT+5:30'
'2023/06/06 6:42:00 PM GMT+5:30' '2023/06/06 6:42:07 PM GMT+5:30'
'2023/06/06 6:43:02 PM GMT+5:30' '2023/06/06 6:43:32 PM GMT+5:30'
'2023/06/06 6:44:04 PM GMT+5:30' '2023/06/06 6:44:50 PM GMT+5:30'
'2023/06/06 6:45:07 PM GMT+5:30' '2023/06/06 6:45:33 PM GMT+5:30'
'2023/06/06 6:46:14 PM GMT+5:30' '2023/06/06 6:46:21 PM GMT+5:30'
'2023/06/06 6:48:26 PM GMT+5:30' '2023/06/06 6:51:13 PM GMT+5:30'
'2023/06/06 6:57:14 PM GMT+5:30' '2023/06/06 7:02:21 PM GMT+5:30'
'2023/06/06 7:08:47 PM GMT+5:30' '2023/06/06 7:10:14 PM GMT+5:30'
'2023/06/06 7:10:55 PM GMT+5:30' '2023/06/06 7:11:13 PM GMT+5:30'
'2023/06/06 7:11:46 PM GMT+5:30' '2023/06/06 7:12:54 PM GMT+5:30'
```

```
'2023/06/06 7:15:48 PM GMT+5:30' '2023/06/06 7:16:28 PM GMT+5:30'
'2023/06/06 7:17:17 PM GMT+5:30' '2023/06/06 7:18:01 PM GMT+5:30'
'2023/06/06 7:18:52 PM GMT+5:30' '2023/06/06 7:19:43 PM GMT+5:30'
'2023/06/06 7:20:35 PM GMT+5:30' '2023/06/06 7:21:26 PM GMT+5:30'
'2023/06/06 7:22:31 PM GMT+5:30' '2023/06/06 7:23:25 PM GMT+5:30'
'2023/06/06 7:31:34 PM GMT+5:30' '2023/06/06 7:39:44 PM GMT+5:30'
'2023/06/06 7:40:11 PM GMT+5:30' '2023/06/06 7:40:53 PM GMT+5:30'
'2023/06/06 7:41:36 PM GMT+5:30' '2023/06/06 7:42:13 PM GMT+5:30'
'2023/06/06 7:42:51 PM GMT+5:30' '2023/06/06 7:46:27 PM GMT+5:30'
'2023/06/06 7:54:19 PM GMT+5:30' '2023/06/06 8:01:47 PM GMT+5:30'
'2023/06/06 8:08:19 PM GMT+5:30' '2023/06/06 8:55:08 PM GMT+5:30'
'2023/06/06 8:56:17 PM GMT+5:30' '2023/06/06 8:57:23 PM GMT+5:30'
'2023/06/06 9:08:20 PM GMT+5:30' '2023/06/06 9:08:39 PM GMT+5:30'
'2023/06/06 11:06:30 PM GMT+5:30' '2023/06/06 11:28:31 PM GMT+5:30'
'2023/06/07 12:58:28 AM GMT+5:30' '2023/06/07 3:08:03 AM GMT+5:30'
'2023/06/07 6:09:58 AM GMT+5:30' '2023/06/07 9:19:23 AM GMT+5:30'
'2023/06/07 9:20:34 AM GMT+5:30' '2023/06/07 9:24:31 AM GMT+5:30'
'2023/06/07 9:25:35 AM GMT+5:30' '2023/06/07 9:27:03 AM GMT+5:30'
'2023/06/07 9:28:09 AM GMT+5:30' '2023/06/07 9:32:38 AM GMT+5:30'
'2023/06/07 10:56:11 AM GMT+5:30' '2023/06/07 10:57:14 AM GMT+5:30'
'2023/06/07 10:58:27 AM GMT+5:30' '2023/06/07 11:41:56 AM GMT+5:30'
'2023/06/07 11:44:55 AM GMT+5:30' '2023/06/07 11:46:52 AM GMT+5:30'
'2023/06/07 11:47:07 AM GMT+5:30' '2023/06/07 11:47:44 AM GMT+5:30'
'2023/06/07 11:48:25 AM GMT+5:30' '2023/06/07 11:54:03 AM GMT+5:30'
'2023/06/07 12:01:15 PM GMT+5:30' '2023/06/07 12:13:39 PM GMT+5:30'
'2023/06/07 12:23:07 PM GMT+5:30' '2023/06/07 12:27:12 PM GMT+5:30'
'2023/06/07 12:29:31 PM GMT+5:30' '2023/06/07 12:30:22 PM GMT+5:30'
'2023/06/07 12:36:04 PM GMT+5:30' '2023/06/07 12:39:08 PM GMT+5:30'
'2023/06/07 12:44:27 PM GMT+5:30' '2023/06/07 12:50:13 PM GMT+5:30'
'2023/06/07 1:00:08 PM GMT+5:30' '2023/06/07 1:00:09 PM GMT+5:30'
'2023/06/07 1:04:07 PM GMT+5:30' '2023/06/07 1:05:41 PM GMT+5:30'
'2023/06/07 1:25:26 PM GMT+5:30' '2023/06/07 1:33:51 PM GMT+5:30'
'2023/06/07 1:40:37 PM GMT+5:30' '2023/06/07 1:49:06 PM GMT+5:30'
'2023/06/07 2:09:49 PM GMT+5:30' '2023/06/07 2:11:50 PM GMT+5:30'
'2023/06/07 2:16:43 PM GMT+5:30' '2023/06/07 2:19:05 PM GMT+5:30'
'2023/06/07 2:22:14 PM GMT+5:30' '2023/06/07 2:23:02 PM GMT+5:30'
'2023/06/07 2:33:56 PM GMT+5:30' '2023/06/07 2:40:30 PM GMT+5:30'
'2023/06/07 3:13:52 PM GMT+5:30' '2023/06/07 3:56:01 PM GMT+5:30'
'2023/06/07 3:58:52 PM GMT+5:30' '2023/06/07 4:10:33 PM GMT+5:30'
'2023/06/07 4:46:12 PM GMT+5:30' '2023/06/07 5:26:20 PM GMT+5:30'
'2023/06/07 5:58:12 PM GMT+5:30' '2023/06/07 6:15:10 PM GMT+5:30'
'2023/06/07 6:16:51 PM GMT+5:30' '2023/06/07 6:17:42 PM GMT+5:30'
'2023/06/07 6:18:37 PM GMT+5:30' '2023/06/07 6:20:00 PM GMT+5:30'
'2023/06/07 6:20:53 PM GMT+5:30' '2023/06/07 6:28:35 PM GMT+5:30'
'2023/06/07 6:29:14 PM GMT+5:30' '2023/06/07 6:29:53 PM GMT+5:30'
'2023/06/07 6:30:43 PM GMT+5:30' '2023/06/07 6:31:23 PM GMT+5:30'
'2023/06/07 7:17:13 PM GMT+5:30' '2023/06/07 8:20:23 PM GMT+5:30'
'2023/06/07 8:41:53 PM GMT+5:30' '2023/06/07 9:16:59 PM GMT+5:30'
'2023/06/07 9:17:50 PM GMT+5:30' '2023/06/07 9:18:29 PM GMT+5:30'
```

```
 '2023/06/07 9:18:55 PM GMT+5:30' '2023/06/07 9:19:32 PM GMT+5:30'
 '2023/06/07 9:21:10 PM GMT+5:30' '2023/06/07 9:21:50 PM GMT+5:30'
 '2023/06/07 9:22:24 PM GMT+5:30' '2023/06/07 9:23:39 PM GMT+5:30'
 '2023/06/07 9:23:41 PM GMT+5:30' '2023/06/07 9:35:08 PM GMT+5:30'
 '2023/06/07 9:35:23 PM GMT+5:30' '2023/06/07 9:53:52 PM GMT+5:30'
 '2023/06/07 9:54:24 PM GMT+5:30' '2023/06/07 9:54:56 PM GMT+5:30'
 '2023/06/07 10:07:42 PM GMT+5:30' '2023/06/07 10:09:17 PM GMT+5:30'
 '2023/06/07 10:09:54 PM GMT+5:30' '2023/06/07 10:10:26 PM GMT+5:30'
 '2023/06/07 10:10:58 PM GMT+5:30' '2023/06/07 10:11:28 PM GMT+5:30'
 '2023/06/07 10:16:22 PM GMT+5:30' '2023/06/07 10:16:54 PM GMT+5:30'
 '2023/06/07 10:17:40 PM GMT+5:30' '2023/06/07 10:20:09 PM GMT+5:30'
 '2023/06/07 10:21:00 PM GMT+5:30' '2023/06/07 10:35:06 PM GMT+5:30'
 '2023/06/07 10:38:51 PM GMT+5:30' '2023/06/07 10:39:21 PM GMT+5:30'
 '2023/06/07 10:40:10 PM GMT+5:30' '2023/06/07 10:41:10 PM GMT+5:30'
 '2023/06/07 10:42:24 PM GMT+5:30' '2023/06/07 10:44:42 PM GMT+5:30'
 '2023/06/07 10:58:04 PM GMT+5:30' '2023/06/07 10:59:06 PM GMT+5:30'
 '2023/06/07 11:00:15 PM GMT+5:30' '2023/06/07 11:01:42 PM GMT+5:30'
 '2023/06/07 11:02:45 PM GMT+5:30' '2023/06/07 11:03:50 PM GMT+5:30'
 '2023/06/07 11:05:19 PM GMT+5:30' '2023/06/07 11:06:40 PM GMT+5:30'
 '2023/06/07 11:07:55 PM GMT+5:30' '2023/06/07 11:08:52 PM GMT+5:30'
 '2023/06/07 11:10:09 PM GMT+5:30' '2023/06/08 3:23:21 AM GMT+5:30'
 '2023/06/08 3:23:36 AM GMT+5:30' '2023/06/08 3:23:58 AM GMT+5:30'
 '2023/06/08 3:24:37 AM GMT+5:30' '2023/06/08 3:25:14 AM GMT+5:30'
 '2023/06/08 3:25:32 AM GMT+5:30' '2023/06/08 3:26:03 AM GMT+5:30'
 '2023/06/08 3:26:38 AM GMT+5:30' '2023/06/08 3:27:23 AM GMT+5:30'
 '2023/06/08 3:27:57 AM GMT+5:30' '2023/06/08 3:28:23 AM GMT+5:30'
 '2023/06/08 3:29:26 AM GMT+5:30' '2023/06/08 3:30:05 AM GMT+5:30'
 '2023/06/08 3:30:19 AM GMT+5:30' '2023/06/08 3:32:42 AM GMT+5:30'
 '2023/06/08 3:33:39 AM GMT+5:30' '2023/06/08 7:00:57 AM GMT+5:30'
 '2023/06/08 7:33:15 AM GMT+5:30' '2023/06/08 8:59:45 AM GMT+5:30'
 '2023/06/08 9:16:26 AM GMT+5:30' '2023/06/08 9:54:40 AM GMT+5:30'
 '2023/06/08 10:54:00 AM GMT+5:30' '2023/06/08 11:59:14 AM GMT+5:30'
 '2023/06/08 12:22:05 PM GMT+5:30' '2023/06/08 2:21:41 PM GMT+5:30'
 '2023/06/08 4:29:56 PM GMT+5:30' '2023/06/08 5:06:30 PM GMT+5:30'
 '2023/06/08 5:07:19 PM GMT+5:30' '2023/06/08 5:07:35 PM GMT+5:30'
 '2023/06/08 5:08:27 PM GMT+5:30' '2023/06/08 5:09:31 PM GMT+5:30'
 '2023/06/08 5:11:27 PM GMT+5:30' '2023/06/08 5:13:33 PM GMT+5:30'
 '2023/06/08 5:14:30 PM GMT+5:30' '2023/06/08 5:15:27 PM GMT+5:30'
 '2023/06/08 5:16:17 PM GMT+5:30' '2023/06/08 5:17:10 PM GMT+5:30'
 '2023/06/08 5:19:23 PM GMT+5:30' '2023/06/08 5:19:41 PM GMT+5:30'
 '2023/06/08 5:19:56 PM GMT+5:30' '2023/06/08 5:20:43 PM GMT+5:30'
 '2023/06/08 5:20:57 PM GMT+5:30' '2023/06/08 5:22:08 PM GMT+5:30'
 '2023/06/08 5:23:05 PM GMT+5:30' '2023/06/08 5:23:08 PM GMT+5:30'
 '2023/06/08 5:24:42 PM GMT+5:30' '2023/06/08 5:25:27 PM GMT+5:30'
 '2023/06/08 5:25:40 PM GMT+5:30' '2023/06/08 5:27:22 PM GMT+5:30'
 '2023/06/08 5:28:09 PM GMT+5:30' '2023/06/08 5:28:47 PM GMT+5:30'
 '2023/06/08 5:29:01 PM GMT+5:30' '2023/06/08 5:30:12 PM GMT+5:30'
 '2023/06/08 5:30:34 PM GMT+5:30' '2023/06/08 5:31:03 PM GMT+5:30'
 '2023/06/08 5:32:09 PM GMT+5:30' '2023/06/08 5:32:12 PM GMT+5:30'
```

```
 '2023/06/08 5:33:01 PM GMT+5:30' '2023/06/08 5:33:12 PM GMT+5:30'
 '2023/06/08 5:34:27 PM GMT+5:30' '2023/06/08 5:35:01 PM GMT+5:30'
 '2023/06/08 5:35:25 PM GMT+5:30' '2023/06/08 5:36:14 PM GMT+5:30'
 '2023/06/08 5:37:13 PM GMT+5:30' '2023/06/08 5:37:58 PM GMT+5:30'
 '2023/06/08 5:38:01 PM GMT+5:30' '2023/06/08 5:38:57 PM GMT+5:30'
 '2023/06/08 5:39:54 PM GMT+5:30' '2023/06/08 5:39:57 PM GMT+5:30'
 '2023/06/08 5:40:49 PM GMT+5:30' '2023/06/08 5:41:46 PM GMT+5:30'
 '2023/06/08 5:42:23 PM GMT+5:30' '2023/06/08 5:42:27 PM GMT+5:30'
 '2023/06/08 5:43:45 PM GMT+5:30' '2023/06/08 5:44:57 PM GMT+5:30'
 '2023/06/08 5:45:08 PM GMT+5:30' '2023/06/08 5:45:59 PM GMT+5:30'
 '2023/06/08 5:46:49 PM GMT+5:30' '2023/06/08 5:47:32 PM GMT+5:30'
 '2023/06/08 5:48:14 PM GMT+5:30' '2023/06/08 5:48:55 PM GMT+5:30'
 '2023/06/08 5:49:56 PM GMT+5:30' '2023/06/08 5:49:59 PM GMT+5:30'
 '2023/06/08 5:52:26 PM GMT+5:30' '2023/06/08 5:52:44 PM GMT+5:30'
 '2023/06/08 5:53:11 PM GMT+5:30' '2023/06/08 5:54:05 PM GMT+5:30'
 '2023/06/08 5:54:49 PM GMT+5:30' '2023/06/08 5:55:15 PM GMT+5:30'
 '2023/06/08 5:55:54 PM GMT+5:30' '2023/06/08 5:57:03 PM GMT+5:30'
 '2023/06/08 5:57:34 PM GMT+5:30' '2023/06/08 5:58:03 PM GMT+5:30'
 '2023/06/08 5:58:43 PM GMT+5:30' '2023/06/08 5:59:20 PM GMT+5:30'
 '2023/06/08 5:59:25 PM GMT+5:30' '2023/06/08 6:00:06 PM GMT+5:30'
 '2023/06/08 6:00:20 PM GMT+5:30' '2023/06/08 6:00:50 PM GMT+5:30'
 '2023/06/08 6:01:31 PM GMT+5:30' '2023/06/08 6:01:33 PM GMT+5:30'
 '2023/06/08 6:39:59 PM GMT+5:30' '2023/06/08 7:46:31 PM GMT+5:30'
 '2023/06/08 7:50:55 PM GMT+5:30' '2023/06/08 8:26:58 PM GMT+5:30'
 '2023/06/08 9:17:38 PM GMT+5:30' '2023/06/08 9:39:30 PM GMT+5:30'
 '2023/06/08 9:39:55 PM GMT+5:30' '2023/06/08 9:40:24 PM GMT+5:30'
 '2023/06/08 9:40:50 PM GMT+5:30' '2023/06/08 10:13:41 PM GMT+5:30'
 '2023/06/08 10:21:20 PM GMT+5:30' '2023/06/08 10:24:19 PM GMT+5:30'
 '2023/06/08 10:38:16 PM GMT+5:30' '2023/06/08 10:38:51 PM GMT+5:30'
 '2023/06/08 10:39:27 PM GMT+5:30' '2023/06/08 10:39:59 PM GMT+5:30'
 '2023/06/08 10:40:33 PM GMT+5:30' '2023/06/08 10:41:11 PM GMT+5:30'
 '2023/06/09 3:47:38 AM GMT+5:30' '2023/06/09 8:10:05 AM GMT+5:30'
 '2023/06/09 9:31:57 AM GMT+5:30' '2023/06/09 9:32:44 AM GMT+5:30'
 '2023/06/09 9:33:35 AM GMT+5:30' '2023/06/09 9:34:07 AM GMT+5:30'
 '2023/06/09 9:34:44 AM GMT+5:30' '2023/06/09 9:35:19 AM GMT+5:30'
 '2023/06/09 9:35:57 AM GMT+5:30' '2023/06/09 9:37:15 AM GMT+5:30'
 '2023/06/09 9:37:44 AM GMT+5:30' '2023/06/09 9:38:31 AM GMT+5:30'
 '2023/06/09 9:39:04 AM GMT+5:30' '2023/06/09 10:22:03 AM GMT+5:30'
 '2023/06/09 10:22:35 AM GMT+5:30' '2023/06/09 10:23:05 AM GMT+5:30'
 '2023/06/09 10:23:45 AM GMT+5:30' '2023/06/09 10:24:58 AM GMT+5:30'
 '2023/06/09 10:25:27 AM GMT+5:30' '2023/06/09 10:26:01 AM GMT+5:30'
 '2023/06/09 10:26:26 AM GMT+5:30' '2023/06/09 10:26:51 AM GMT+5:30'
 '2023/06/09 10:28:23 AM GMT+5:30' '2023/06/09 10:28:57 AM GMT+5:30'
 '2023/06/09 10:30:39 AM GMT+5:30' '2023/06/09 10:31:15 AM GMT+5:30'
 '2023/06/09 10:58:13 AM GMT+5:30' '2023/06/09 10:58:44 AM GMT+5:30'
 '2023/06/09 10:59:38 AM GMT+5:30' '2023/06/09 12:22:47 PM GMT+5:30'
 '2023/06/09 2:22:25 PM GMT+5:30' '2023/06/09 2:23:03 PM GMT+5:30'
 '2023/06/09 2:23:45 PM GMT+5:30' '2023/06/09 2:34:43 PM GMT+5:30'
 '2023/06/09 2:35:29 PM GMT+5:30' '2023/06/09 2:36:09 PM GMT+5:30'
```

```
 '2023/06/09 2:36:46 PM GMT+5:30' '2023/06/09 2:39:16 PM GMT+5:30'
 '2023/06/09 2:39:47 PM GMT+5:30' '2023/06/09 2:40:21 PM GMT+5:30'
 '2023/06/09 2:41:53 PM GMT+5:30' '2023/06/09 2:42:24 PM GMT+5:30'
 '2023/06/09 2:42:57 PM GMT+5:30' '2023/06/09 2:43:37 PM GMT+5:30'
 '2023/06/09 2:44:09 PM GMT+5:30' '2023/06/09 2:49:20 PM GMT+5:30'
 '2023/06/09 2:51:07 PM GMT+5:30' '2023/06/09 3:00:03 PM GMT+5:30'
 '2023/06/09 3:06:37 PM GMT+5:30' '2023/06/09 3:07:36 PM GMT+5:30'
 '2023/06/09 3:08:38 PM GMT+5:30' '2023/06/09 3:09:41 PM GMT+5:30'
 '2023/06/09 3:10:36 PM GMT+5:30' '2023/06/09 3:16:30 PM GMT+5:30'
 '2023/06/09 3:17:31 PM GMT+5:30' '2023/06/09 3:18:25 PM GMT+5:30'
 '2023/06/09 3:19:53 PM GMT+5:30' '2023/06/09 3:22:15 PM GMT+5:30'
 '2023/06/09 3:23:38 PM GMT+5:30' '2023/06/09 3:24:36 PM GMT+5:30'
 '2023/06/09 3:25:33 PM GMT+5:30' '2023/06/09 3:26:38 PM GMT+5:30'
 '2023/06/09 4:28:30 PM GMT+5:30' '2023/06/09 4:28:58 PM GMT+5:30'
 '2023/06/09 4:29:23 PM GMT+5:30' '2023/06/09 4:29:50 PM GMT+5:30'
 '2023/06/09 4:30:15 PM GMT+5:30' '2023/06/09 4:30:40 PM GMT+5:30'
 '2023/06/09 4:31:06 PM GMT+5:30' '2023/06/09 4:31:33 PM GMT+5:30'
 '2023/06/09 4:32:00 PM GMT+5:30' '2023/06/09 4:32:28 PM GMT+5:30'
 '2023/06/09 4:32:56 PM GMT+5:30' '2023/06/09 4:33:19 PM GMT+5:30'
 '2023/06/09 7:22:11 PM GMT+5:30' '2023/06/09 7:23:09 PM GMT+5:30'
 '2023/06/09 7:24:06 PM GMT+5:30' '2023/06/09 7:24:57 PM GMT+5:30'
 '2023/06/09 7:26:02 PM GMT+5:30' '2023/06/09 7:27:00 PM GMT+5:30'
 '2023/06/09 7:28:09 PM GMT+5:30' '2023/06/10 10:49:42 AM GMT+5:30'
 '2023/06/10 2:40:34 PM GMT+5:30' '2023/06/10 2:41:51 PM GMT+5:30'
 '2023/06/10 11:21:59 PM GMT+5:30' '2023/06/11 9:31:41 AM GMT+5:30'
 '2023/06/11 9:53:31 AM GMT+5:30' '2023/06/11 2:07:39 PM GMT+5:30'
 '2023/06/11 9:15:38 PM GMT+5:30' '2023/06/11 10:42:48 PM GMT+5:30'
 '2023/06/11 10:43:19 PM GMT+5:30' '2023/06/11 10:44:01 PM GMT+5:30'
 '2023/06/11 10:44:46 PM GMT+5:30' '2023/06/11 10:45:34 PM GMT+5:30'
 '2023/06/11 10:47:42 PM GMT+5:30' '2023/06/11 10:48:27 PM GMT+5:30'
 '2023/06/11 10:49:05 PM GMT+5:30' '2023/06/11 10:49:49 PM GMT+5:30'
 '2023/06/11 10:50:44 PM GMT+5:30' '2023/06/11 10:51:27 PM GMT+5:30'
 '2023/06/11 10:52:06 PM GMT+5:30' '2023/06/11 10:52:47 PM GMT+5:30'
 '2023/06/11 10:53:44 PM GMT+5:30' '2023/06/11 10:54:33 PM GMT+5:30'
 '2023/06/11 10:55:58 PM GMT+5:30' '2023/06/11 10:56:58 PM GMT+5:30'
 '2023/06/11 10:57:34 PM GMT+5:30' '2023/06/11 10:58:09 PM GMT+5:30'
 '2023/06/11 10:58:52 PM GMT+5:30' '2023/06/11 10:59:30 PM GMT+5:30'
 '2023/06/11 11:00:11 PM GMT+5:30' '2023/06/11 11:01:04 PM GMT+5:30'
 '2023/06/11 11:02:11 PM GMT+5:30' '2023/06/11 11:02:55 PM GMT+5:30'
 '2023/06/11 11:03:34 PM GMT+5:30' '2023/06/11 11:04:17 PM GMT+5:30'
 '2023/06/11 11:04:55 PM GMT+5:30' '2023/06/11 11:05:38 PM GMT+5:30'
 '2023/06/11 11:06:16 PM GMT+5:30' '2023/06/11 11:06:52 PM GMT+5:30'
 '2023/06/11 11:07:31 PM GMT+5:30' '2023/06/11 11:08:35 PM GMT+5:30'
 '2023/06/11 11:09:33 PM GMT+5:30' '2023/06/11 11:10:14 PM GMT+5:30'
 '2023/06/11 11:10:59 PM GMT+5:30' '2023/06/11 11:15:03 PM GMT+5:30'
 '2023/06/11 11:15:37 PM GMT+5:30' '2023/06/11 11:16:18 PM GMT+5:30'
 '2023/06/11 11:16:58 PM GMT+5:30' '2023/06/11 11:17:41 PM GMT+5:30'
 '2023/06/11 11:18:35 PM GMT+5:30' '2023/06/11 11:19:12 PM GMT+5:30'
 '2023/06/11 11:19:45 PM GMT+5:30' '2023/06/11 11:20:18 PM GMT+5:30'
```

```
 '2023/06/11 11:20:57 PM GMT+5:30' '2023/06/11 11:21:30 PM GMT+5:30'
 '2023/06/11 11:22:19 PM GMT+5:30' '2023/06/11 11:22:55 PM GMT+5:30'
 '2023/06/11 11:24:21 PM GMT+5:30' '2023/06/11 11:25:09 PM GMT+5:30'
 '2023/06/11 11:25:48 PM GMT+5:30' '2023/06/11 11:26:21 PM GMT+5:30'
 '2023/06/11 11:26:54 PM GMT+5:30' '2023/06/11 11:27:25 PM GMT+5:30'
 '2023/06/11 11:27:55 PM GMT+5:30' '2023/06/11 11:28:26 PM GMT+5:30'
 '2023/06/12 9:40:22 AM GMT+5:30' '2023/06/12 2:41:06 PM GMT+5:30'
 '2023/06/12 2:41:58 PM GMT+5:30' '2023/06/12 2:43:30 PM GMT+5:30'
 '2023/06/12 2:44:47 PM GMT+5:30' '2023/06/12 2:48:55 PM GMT+5:30'
 '2023/06/12 2:49:47 PM GMT+5:30' '2023/06/12 2:51:24 PM GMT+5:30'
 '2023/06/12 2:54:13 PM GMT+5:30' '2023/06/12 3:27:59 PM GMT+5:30'
 '2023/06/12 3:29:20 PM GMT+5:30' '2023/06/12 3:33:53 PM GMT+5:30'
 '2023/06/12 3:35:24 PM GMT+5:30' '2023/06/12 3:37:14 PM GMT+5:30'
 '2023/06/12 3:38:48 PM GMT+5:30' '2023/06/12 3:40:25 PM GMT+5:30'
 '2023/06/12 3:42:05 PM GMT+5:30' '2023/06/12 3:44:16 PM GMT+5:30'
 '2023/06/12 3:45:27 PM GMT+5:30' '2023/06/12 3:47:01 PM GMT+5:30'
 '2023/06/12 3:48:11 PM GMT+5:30' '2023/06/12 3:49:27 PM GMT+5:30'
 '2023/06/12 3:51:31 PM GMT+5:30' '2023/06/12 3:52:33 PM GMT+5:30'
 '2023/06/12 3:53:28 PM GMT+5:30' '2023/06/12 3:54:44 PM GMT+5:30'
 '2023/06/12 3:55:53 PM GMT+5:30' '2023/06/12 3:56:57 PM GMT+5:30'
 '2023/06/12 3:57:52 PM GMT+5:30' '2023/06/12 3:59:10 PM GMT+5:30'
 '2023/06/12 3:59:59 PM GMT+5:30' '2023/06/12 4:00:56 PM GMT+5:30'
 '2023/06/12 4:02:02 PM GMT+5:30' '2023/06/12 4:02:53 PM GMT+5:30'
 '2023/06/12 4:03:59 PM GMT+5:30' '2023/06/12 9:57:20 PM GMT+5:30'
 '2023/06/16 9:16:05 AM GMT+5:30']
Gender: ['Female' 'Prefer not to say' 'Male' 'Others']
Purchase_Frequency: ['Few times a month' 'Once a month' 'Less than
once a month'
 'Multiple times a week' 'Once a week']
Purchase_Categories: ['Beauty and Personal Care' 'Clothing and
Fashion'
 'Groceries and Gourmet Food;Clothing and Fashion'
 'Beauty and Personal Care;Clothing and Fashion;others'
 'Beauty and Personal Care;Clothing and Fashion'
 'Beauty and Personal Care;Clothing and Fashion;Home and Kitchen'
 'Clothing and Fashion;Home and Kitchen' 'others'
 'Clothing and Fashion;others' 'Beauty and Personal Care;Home and
Kitchen'
 'Groceries and Gourmet Food'
 'Groceries and Gourmet Food;Clothing and Fashion;others'
 'Groceries and Gourmet Food;Beauty and Personal Care;Clothing and
Fashion;Home and Kitchen'
 'Groceries and Gourmet Food;Beauty and Personal Care;Clothing and
Fashion;Home and Kitchen;others'
 'Home and Kitchen' 'Beauty and Personal Care;others'
 'Beauty and Personal Care;Home and Kitchen;others'
 'Home and Kitchen;others' 'Groceries and Gourmet Food;Home and
Kitchen'
 'Beauty and Personal Care;Clothing and Fashion;Home and
```

```
Kitchen;others'
 'Groceries and Gourmet Food;Beauty and Personal Care;Home and
Kitchen'
 'Groceries and Gourmet Food;Home and Kitchen;others'
 'Groceries and Gourmet Food;Clothing and Fashion;Home and
Kitchen;others'
 'Groceries and Gourmet Food;Beauty and Personal Care'
 'Clothing and Fashion;Home and Kitchen;others'
 'Groceries and Gourmet Food;Beauty and Personal Care;Clothing and
Fashion'
 'Groceries and Gourmet Food;Clothing and Fashion;Home and Kitchen'
 'Groceries and Gourmet Food;Beauty and Personal Care;others'
 'Groceries and Gourmet Food;Beauty and Personal Care;Clothing and
Fashion;others']
Personalized_Recommendation_Frequency: ['Yes' 'No' 'Sometimes']
Browsing_Frequency: ['Few times a week' 'Few times a month' 'Rarely'
'Multiple times a day']
Product_Search_Method: ['Keyword' 'Filter' 'categories' 'others' nan]
Search_Result_Exploration: ['Multiple pages' 'First page']
Add_to_Cart_Browsing: ['Yes' 'Maybe' 'No']
Cart_Completion_Frequency: ['Sometimes' 'Often' 'Rarely' 'Never'
'Always']
Cart_Abandonment_Factors: ['Found a better price elsewhere' 'High
shipping costs'
 'Changed my mind or no longer need the item' 'others']
Saveforlater_Frequency: ['Sometimes' 'Rarely' 'Never' 'Often'
'Always']
Review_Left: ['Yes' 'No']
Review_Reliability: ['Occasionally' 'Heavily' 'Moderately' 'Never'
'Rarely']
Review_Helpfulness: ['Yes' 'No' 'Sometimes']
Recommendation_Helpfulness: ['Yes' 'Sometimes' 'No']
Service_Appreciation: ['Competitive prices' 'Wide product selection'
 'User-friendly website/app interface' '.' 'Customer service '
 'Product recommendations' 'Customer service' 'Quick delivery'
 'All the above']
Improvement_Areas: ['Reducing packaging waste' 'Product quality and
accuracy'
 'Shipping speed and reliability' 'Customer service responsiveness'
'.'
 'Nothing' 'better app interface and lower shipping charges' 'Nil'
 'Add more familiar brands to the list' 'UI'
 'Scrolling option would be much better than going to next page'
 'Quality of product is very poor according to the big offers'
 'I have no problem with Amazon yet. But others tell me about the
refund issues '
 'User interface ' 'Irrelevant product suggestions'
 'User interface of app' "I don't have any problem with Amazon"
 'No problems with Amazon']
```

25 Columns

601 Rows

3.3) Check info.

```
#Check info for Amazon dataset
print('Amazon Dataset')
df_Amazon.info()
```

```
Amazon Dataset
<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
```

AMAZON DATASET

Rows=601

Columns=23

Numerical variables = 320

Categorical variables = 18

dtypes: category(1), float64(8), object(16)

```
column_names= df_Amazon.columns
column_names

Index(['Timestamp', 'age', 'Gender', 'Purchase_Frequency',
       'Purchase_Categories', 'Personalized_Recommendation_Frequency',
       'Browsing_Frequency', 'Product_Search_Method',
       'Search_Result_Exploration', 'Customer_Reviews_Importance',
       'Add_to_Cart_Browsing', 'Cart_Completion_Frequency',
       'Cart_Abandonment_Factors', 'Saveforlater_Frequency',
'Review_Left',
       'Review_Reliability', 'Review_Helpfulness',
       'Personalized_Recommendation_Frequency ',
'Recommendation_Helpfulness',
       'Rating_Accuracy ', 'Shopping_Satisfaction',
'Service_Appreciation',
       'Improvement_Areas'],
      dtype='object')
```

4. DATA CLEANING

```
import pandas as pd
from sklearn.impute import SimpleImputer

# Assuming df_Amazon is your DataFrame

# Handle Missing Values
numerical_cols = ['age', 'Rating_Accuracy ', 'Shopping_Satisfaction']
categorical_cols = ['Personalized_Recommendation_Frequency ',
'Service_Appreciation', 'Improvement_Areas']

# Impute missing values in numerical features with mean
imputer_numeric = SimpleImputer(strategy='mean')
df_Amazon[numerical_cols] =
imputer_numeric.fit_transform(df_Amazon[numerical_cols])

# Impute missing values in categorical features with most frequent
category
imputer_categorical = SimpleImputer(strategy='most_frequent')
df_Amazon[categorical_cols] =
imputer_categorical.fit_transform(df_Amazon[categorical_cols])

# Remove unnecessary columns
columns_to_drop = ['Timestamp', 'Personalized_Recommendation_Frequency
', 'Rating_Accuracy ', 'Improvement_Areas']
```

```
df_Amazon_cleaned = df_Amazon.drop(columns=columns_to_drop, axis=1)
df_Amazon_cleaned
```

```
      age             Gender      Purchase_Frequency  \
0    23.0             Female       Few times a month   
1    23.0             Female            Once a month   
2    24.0  Prefer not to say       Few times a month   
3    24.0             Female            Once a month   
4    22.0             Female  Less than once a month   
..    ...                ...                     ...   
597  23.0             Female             Once a week   
598  23.0             Female             Once a week   
599  23.0             Female            Once a month   
600  23.0             Female       Few times a month   
601  23.0             Female             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  \
0                 Keyword            Multiple pages   
1                 Keyword            Multiple pages   
2                 Keyword            Multiple pages   
3                 Keyword                First page   
4                  Filter            Multiple pages   
..                    ...                       ...   
597            categories            Multiple pages   
```

```
598                Filter            Multiple pages
599            categories            Multiple pages
600               Keyword            Multiple pages
601               Keyword            Multiple pages

     Customer_Reviews_Importance Add_to_Cart_Browsing  \
0                              1                  Yes
1                              1                  Yes
2                              2                  Yes
3                              5                Maybe
4                              1                  Yes
..                           ...                  ...
597                            4                Maybe
598                            3                Maybe
599                            3                Maybe
600                            1                  Yes
601                            3                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  Shopping_Satisfaction
Service_Appreciation
0                          Yes                    1.0
Competitive prices
1                    Sometimes                    2.0   Wide product
selection
2                           No                    3.0
Competitive prices
3                    Sometimes                    4.0
Competitive prices
4                          Yes                    2.0
Competitive prices
..                         ...                    ...
...
597                  Sometimes                    4.0
Competitive prices
598                  Sometimes                    3.0  Product
recommendations
599                  Sometimes                    3.0   Wide product
selection
600                        Yes                    2.0   Wide product
selection
601                  Sometimes                    3.0  Product
recommendations

[602 rows x 19 columns]

# Print the first few rows of the cleaned dataset
print(df_Amazon_cleaned.head())

# Check for missing values
print("Missing values in the cleaned dataset:")
print(df_Amazon_cleaned.isnull().sum())

    age             Gender      Purchase_Frequency  \
0  23.0             Female       Few times a month
1  23.0             Female            Once a month
2  24.0  Prefer not to say       Few times a month
3  24.0             Female            Once a month
4  22.0             Female  Less than once a month

                                 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

  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 Add_to_Cart_Browsing
Cart_Completion_Frequency  \
0                            1                  Yes
Sometimes
1                            1                  Yes
Often
2                            2                  Yes
Sometimes
3                            5                Maybe
Sometimes
4                            1                  Yes
Sometimes

         Cart_Abandonment_Factors Saveforlater_Frequency
Review_Left  \
0  Found a better price elsewhere              Sometimes         Yes

1             High shipping costs                 Rarely          No

2  Found a better price elsewhere                 Rarely          No

3  Found a better price elsewhere              Sometimes         Yes

4             High shipping costs                 Rarely          No


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

```
   Shopping_Satisfaction    Service_Appreciation
0                    1.0      Competitive prices
1                    2.0  Wide product selection
2                    3.0      Competitive prices
3                    4.0      Competitive prices
4                    2.0      Competitive prices
Missing values in the cleaned dataset:
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
Recommendation_Helpfulness                  0
Shopping_Satisfaction                       0
Service_Appreciation                        0
dtype: int64
```

```python
#Handling missing values
# Impute missing values with the mode
mode_value = df_Amazon_cleaned['Product_Search_Method'].mode()[0]
df_Amazon_cleaned['Product_Search_Method'].fillna(mode_value,
inplace=True)
print(df_Amazon_cleaned.isnull().sum())
```

```
age                                         0
Gender                                      0
Purchase_Frequency                          0
Purchase_Categories                         0
Personalized_Recommendation_Frequency       0
Browsing_Frequency                          0
Product_Search_Method                       0
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
Recommendation_Helpfulness                        0
Shopping_Satisfaction                             0
Service_Appreciation                              0
dtype: int64
```

5. EXPLORATORY DATA ANALYSIS.

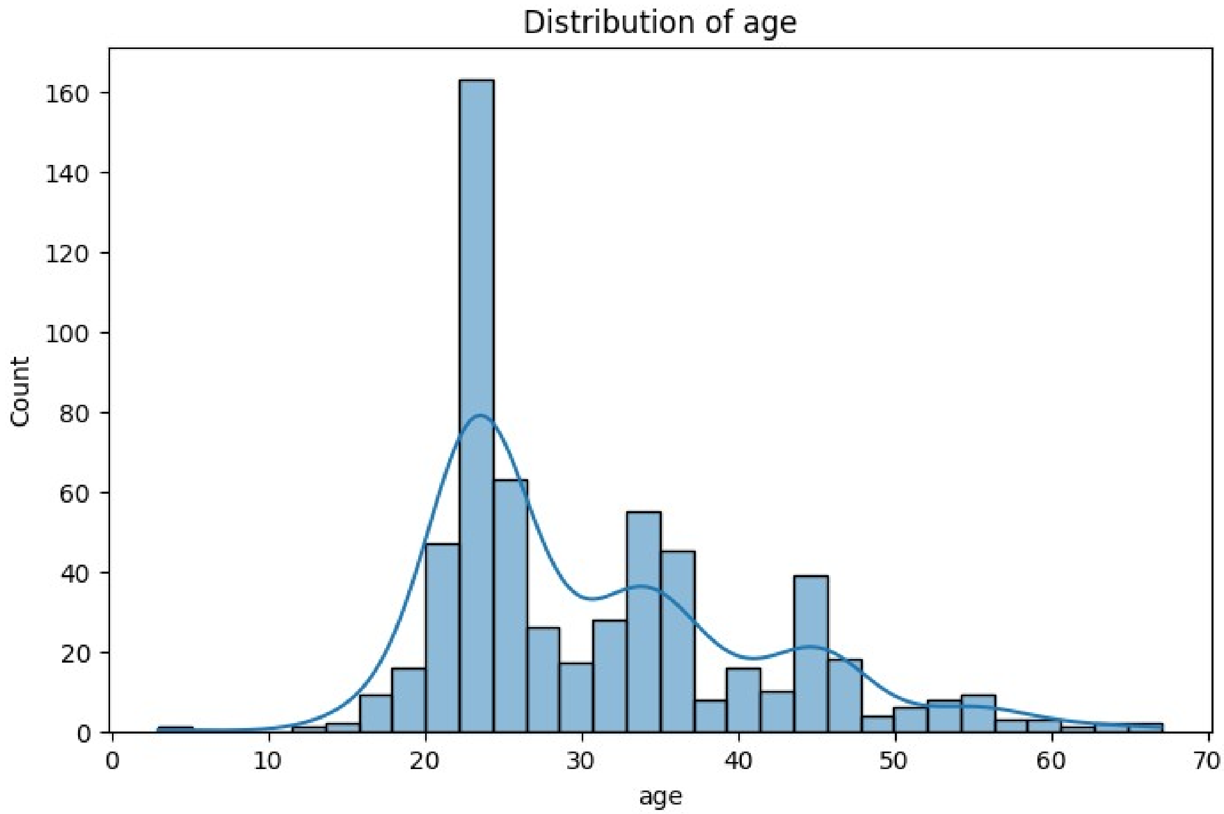
Perform EDA to explore the datasets and understand their distribution, relationships and patterns.

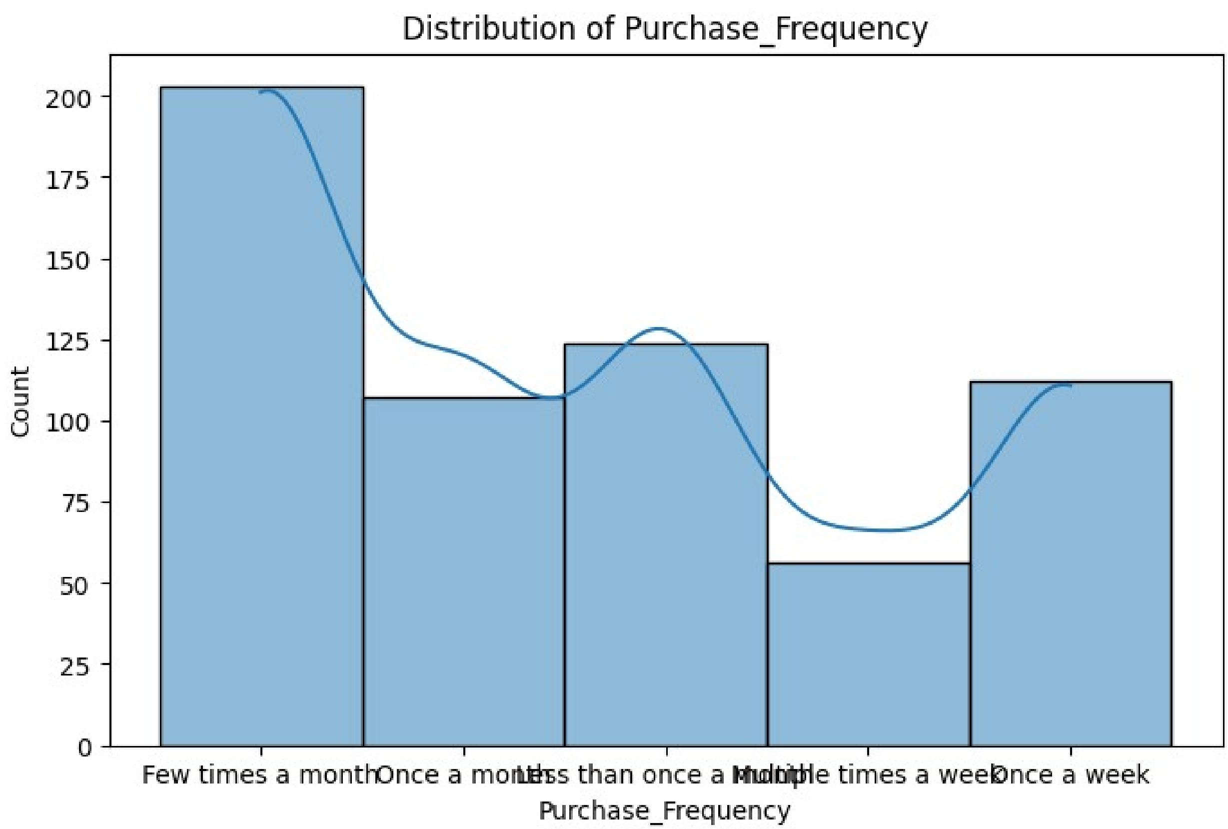
Visualizations are essential for summary statistics in order to gain insights.

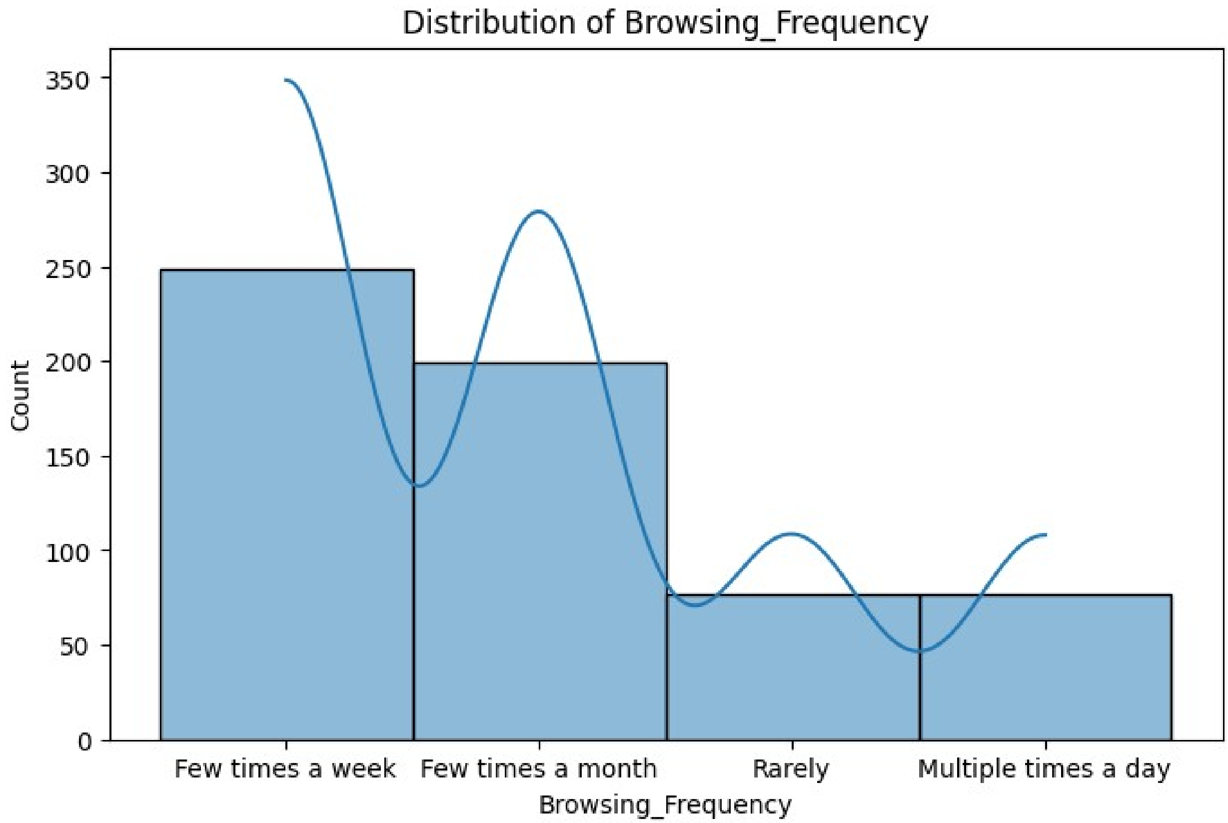
5.1 UNIVARIATE ANALYSIS

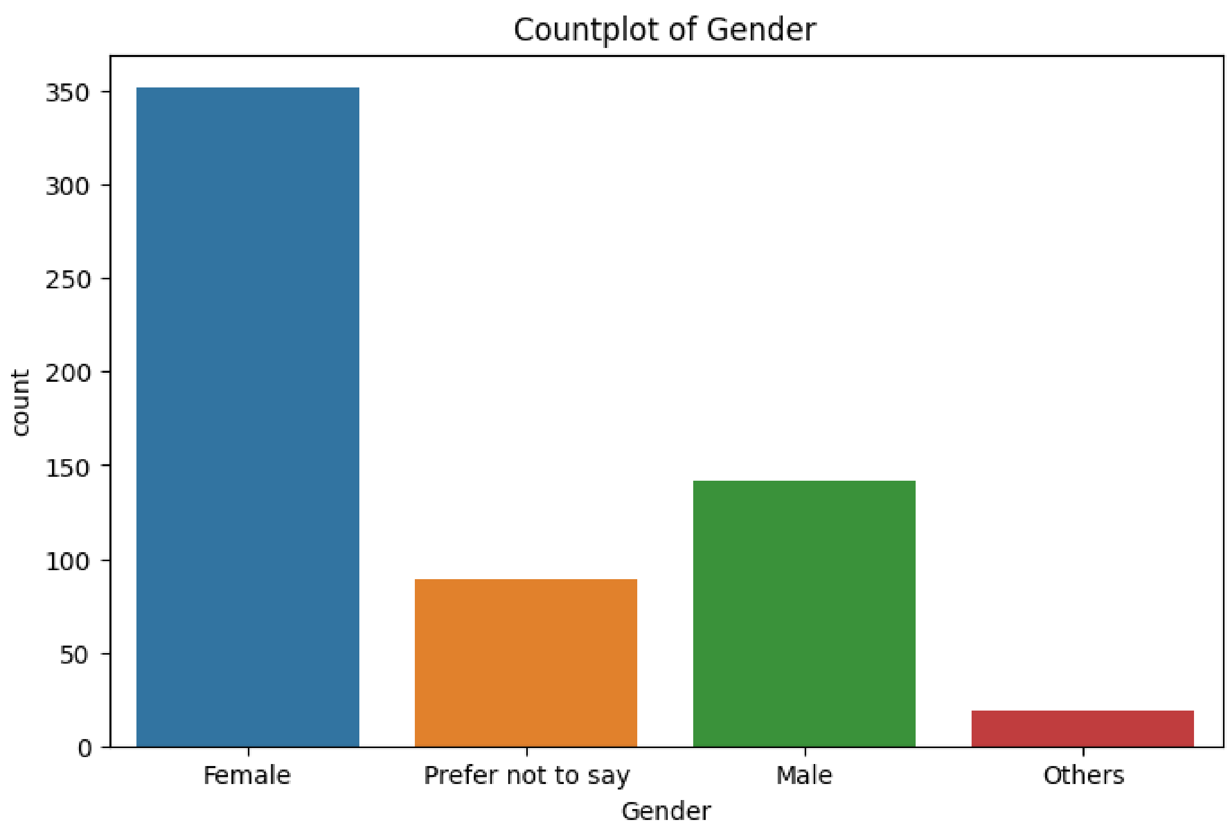
The purpose of a univariate analysis is to Understand the distribution and characteristics of individual variables.

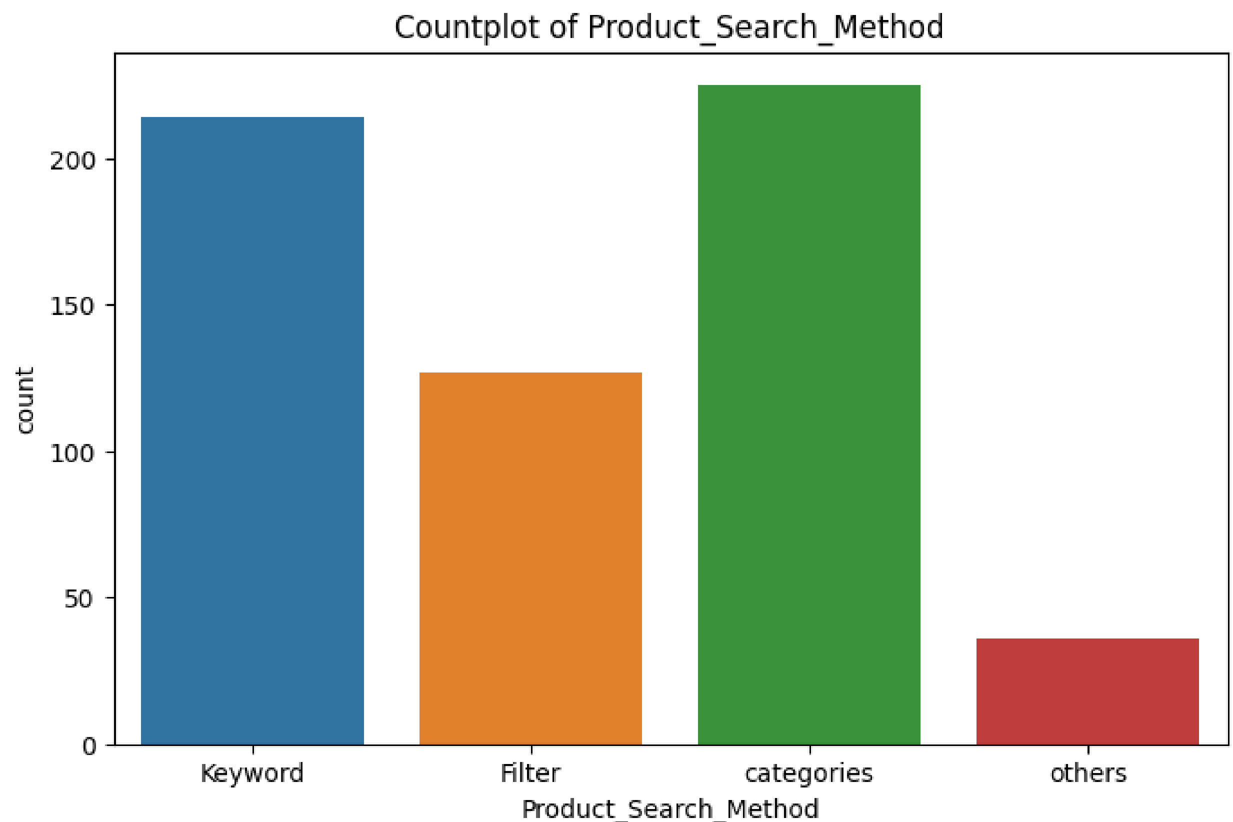
```
# Univariate analysis for numerical features
numerical_features = ['age', 'Purchase_Frequency',
'Browsing_Frequency']

# Histograms for numerical features
for feature in numerical_features:
    plt.figure(figsize=(8, 5))
    sns.histplot(df_Amazon_cleaned[feature], bins=30, kde=True)
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.show()

# Univariate analysis for categorical features
categorical_features = ['Gender', 'Product_Search_Method']

# Countplots for categorical features
for feature in categorical_features:
    plt.figure(figsize=(8, 5))
    sns.countplot(x=feature, data=df_Amazon_cleaned)
    plt.title(f'Countplot of {feature}')
    plt.show()
```


Distribution of age
Count
age
0
20
40
60
80
100
120
140
160
0
10
20
30
40
50
60
70


Distribution of Purchase_Frequency
Count
Purchase_Frequency
0
25
50
75
100
125
150
175
200
Few times a month
Once a month
Less than once a month
Multiple times a week
Once a week


Distribution of Browsing_Frequency
Count
350
300
250
200
150
100
50
0
Few times a week
Few times a month
Rarely
Multiple times a day
Browsing_Frequency


Countplot of Gender
count
350
300
250
200
150
100
50
0
Female
Prefer not to say
Male
Others
Gender



-The highest age group of consumers are between 20-30 yrs

-Thee purchase frequency is relatively equal with most consumers making a purchase once a month.

-The browsing frequency is few times a week and few times a month.

-Females make up the largest number of consumers in terms of gender distribution.

-Most consumers search for products via keywords and categories.

5.3) BIVARIATE ANALYSIS.

The purpose of bivariate analysis is to explore the relationship and associations between pairs of variables.

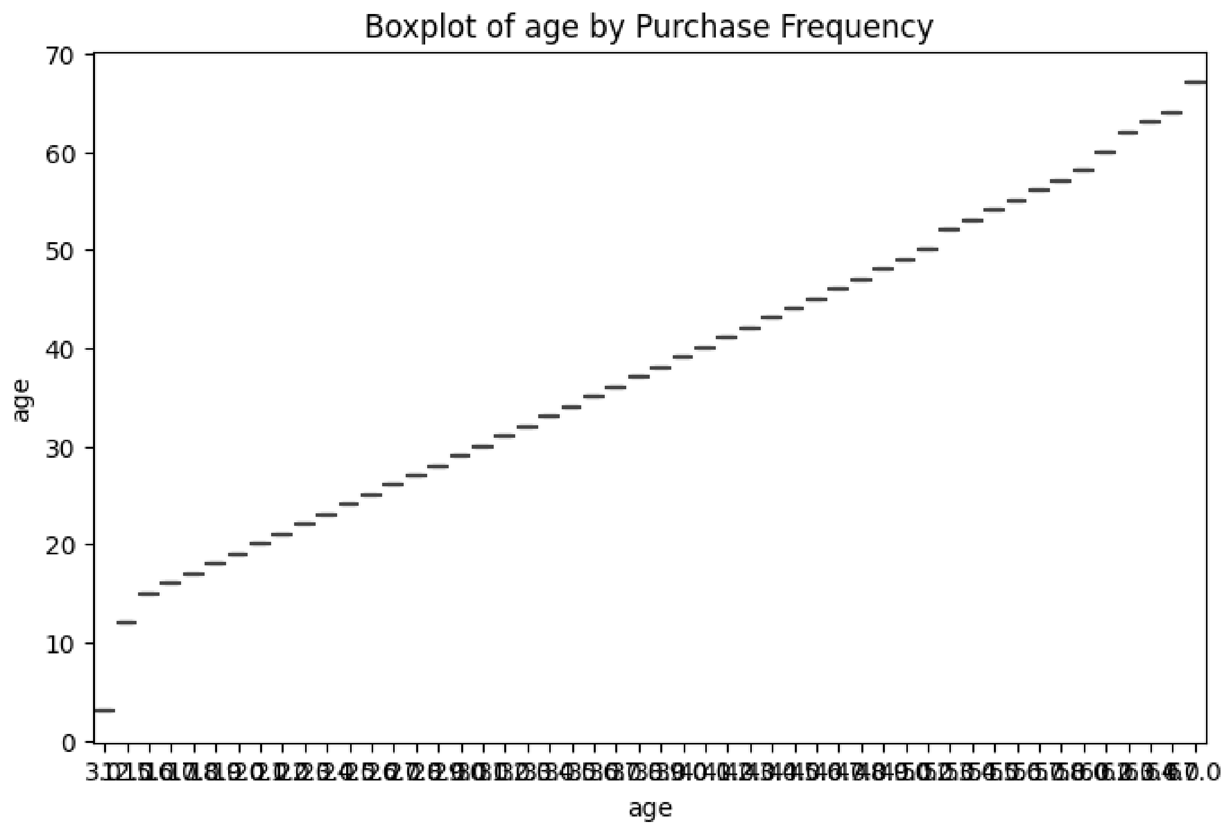
```
# Select numerical variables
numerical_vars = df_Amazon_cleaned.select_dtypes(include=['float64',
'int64']).columns

# Select categorical variables
categorical_vars =
df_Amazon_cleaned.select_dtypes(include=['object']).columns

print("Numerical Variables:")
print(numerical_vars)
```

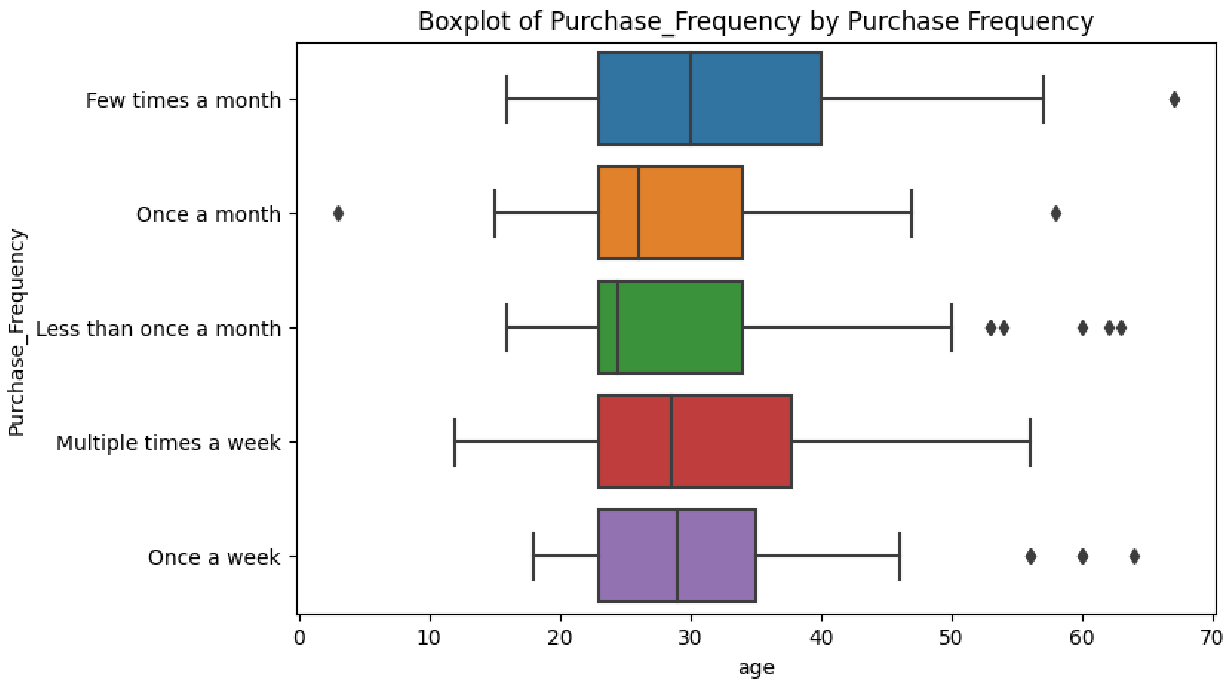
```
print("\nCategorical Variables:")
print(categorical_vars)
```

```
Numerical Variables:
Index(['age', 'Customer_Reviews_Importance', 'Shopping_Satisfaction'],
dtype='object')

Categorical Variables:
Index(['Gender', 'Purchase_Frequency', 'Purchase_Categories',
       'Personalized_Recommendation_Frequency', 'Browsing_Frequency',
       'Product_Search_Method', 'Search_Result_Exploration',
       'Add_to_Cart_Browsing', 'Cart_Completion_Frequency',
       'Cart_Abandonment_Factors', 'Saveforlater_Frequency',
'Review_Left',
       'Review_Reliability', 'Review_Helpfulness',
       'Recommendation_Helpfulness', 'Service_Appreciation'],
      dtype='object')
```

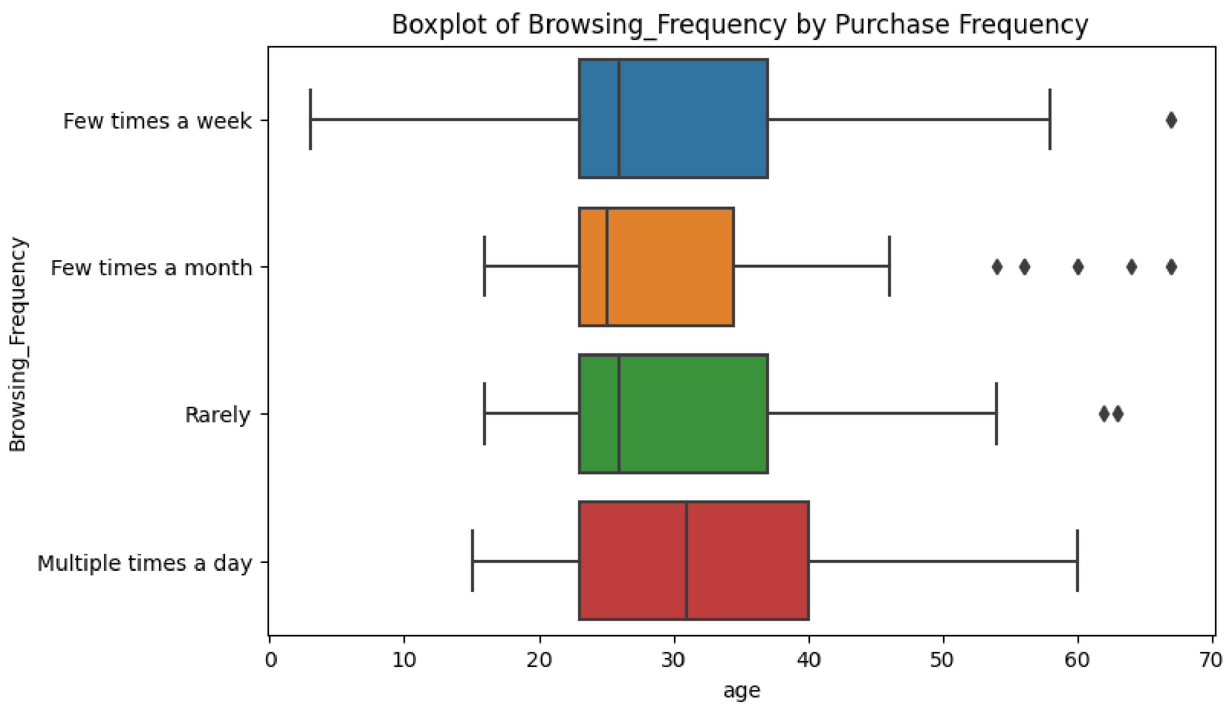
```
import matplotlib.pyplot as plt
import seaborn as sns
# Remove whitespaces from column names
df_Amazon_cleaned.columns = df_Amazon_cleaned.columns.str.strip()


# Numerical features
Numerical_features = ['age', 'Customer_Reviews_Importance',
'Rating_Accuracy ',
       'Shopping_Satisfaction']



# Categorical features
categorical_features = ['Gender', 'Purchase_Frequency',
'Purchase_Categories',
        'Browsing_Frequency',
       'Product_Search_Method', 'Search_Result_Exploration',
       'Add_to_Cart_Browsing', 'Cart_Completion_Frequency',
       'Cart_Abandonment_Factors', 'Saveforlater_Frequency',
'Review_Left',
       'Review_Reliability', 'Review_Helpfulness',
        'Recommendation_Helpfulness',
       'Service_Appreciation']

# Bivariate analysis for numerical features vs. target variable
for feature in numerical_features:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x='age', y=feature, data=df_Amazon_cleaned)
    plt.title(f'Boxplot of {feature} by Purchase Frequency')
    plt.show()
```

```
# Bivariate analysis for categorical features vs. target variable
for feature in categorical_features:
    plt.figure(figsize=(8, 5))
    sns.countplot(x=feature, hue='Browsing_Frequency',
data=df_Amazon_cleaned)
    plt.title(f'Countplot of {feature} by Browsing Frequency')
    plt.show()
```

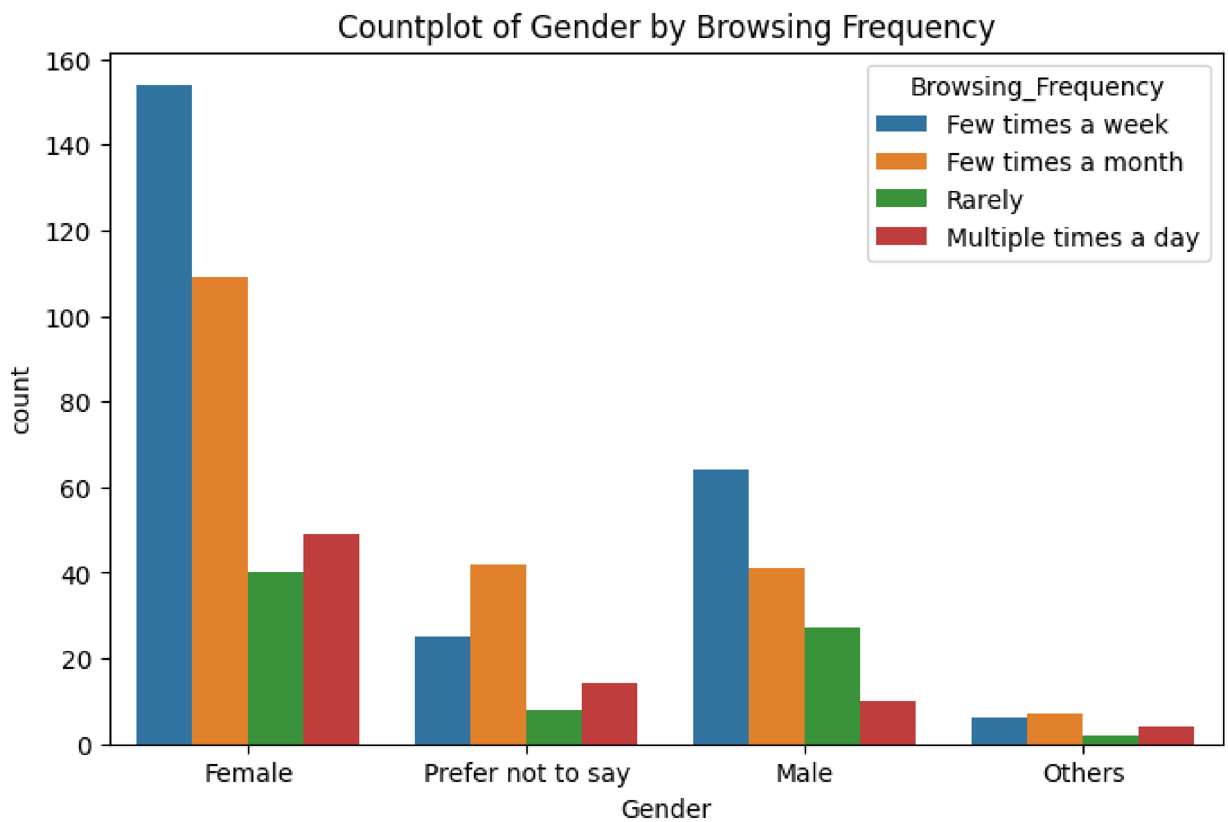




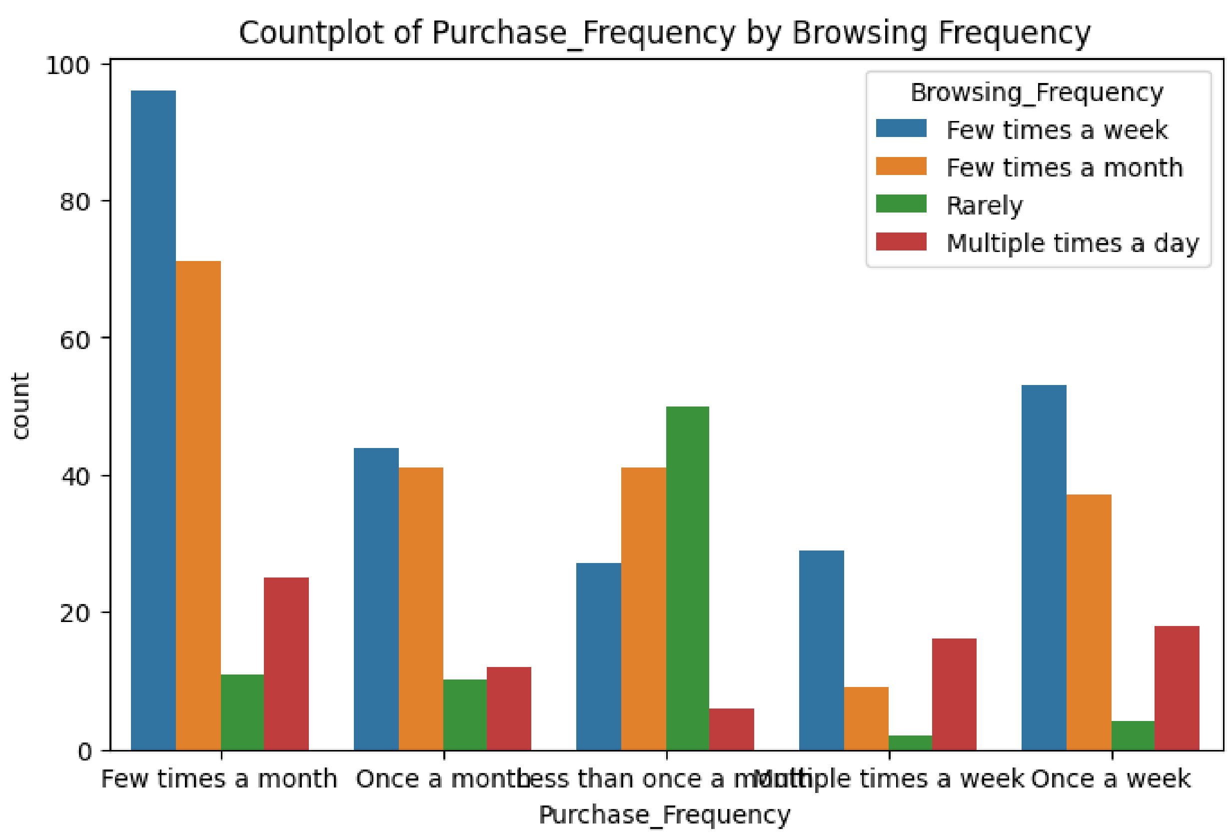
Boxplot of Purchase_Frequency by Purchase Frequency
Purchase_Frequency
Few times a month
Once a month
Less than once a month
Multiple times a week
Once a week
0
10
20
30
40
50
60
70
age



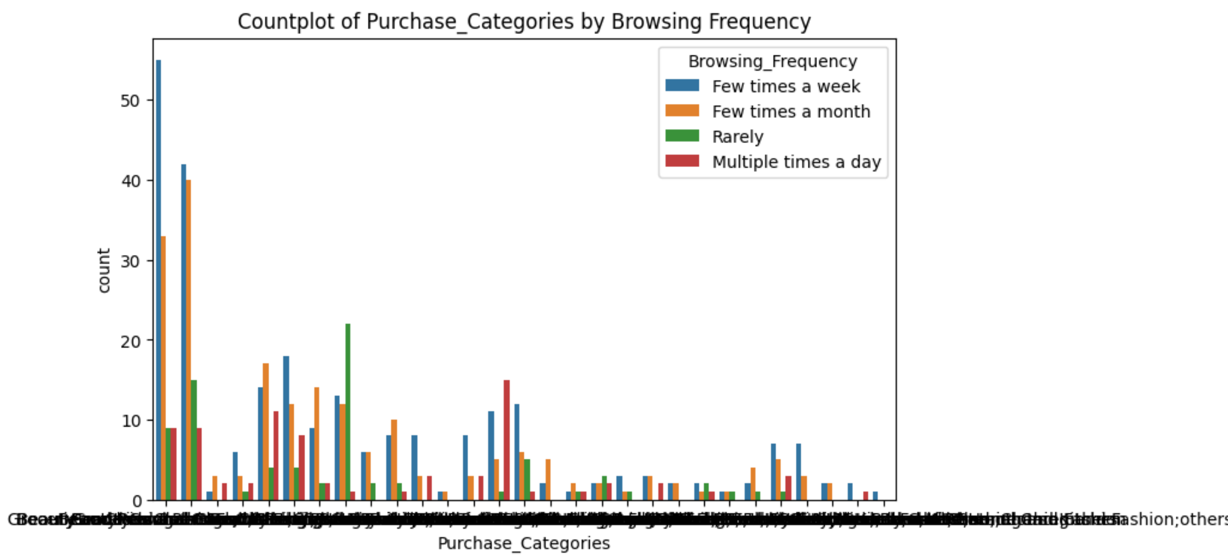
Boxplot of Browsing_Frequency by Purchase Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
0
10
20
30
40
50
60
70
age

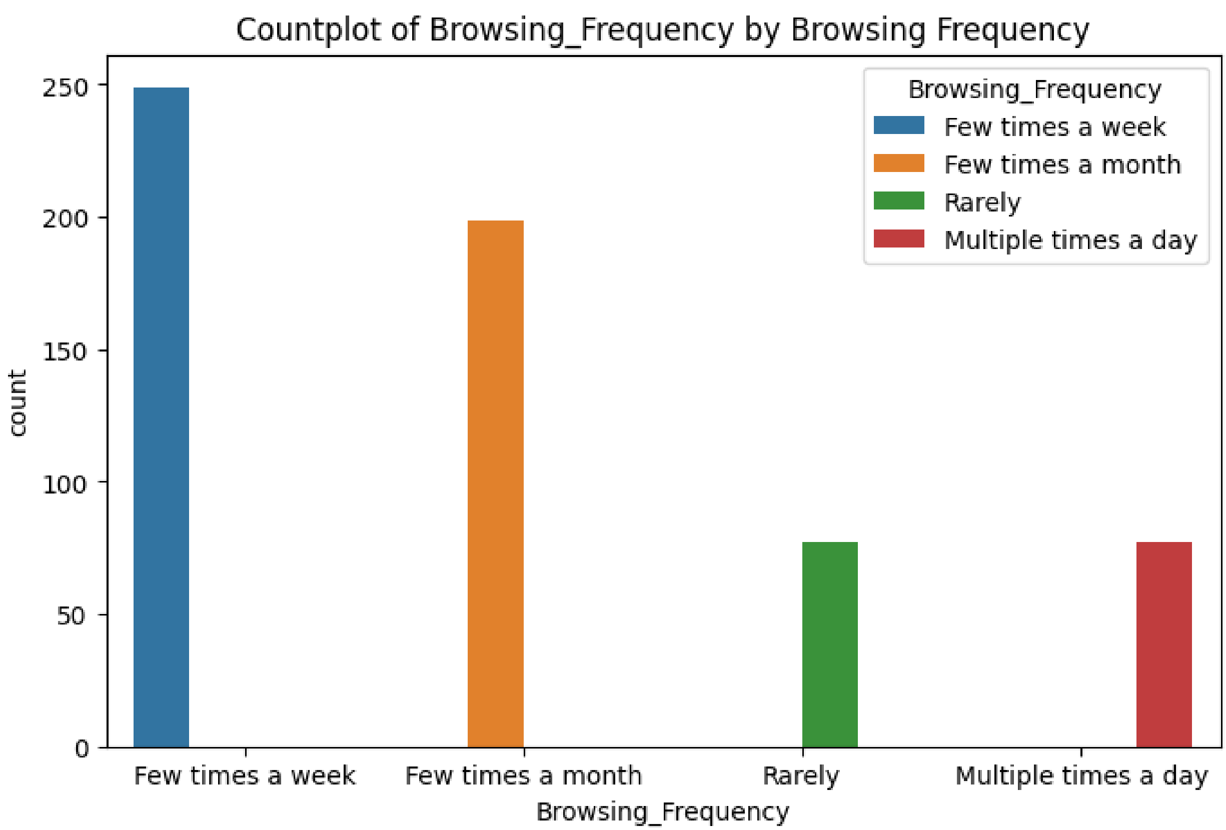


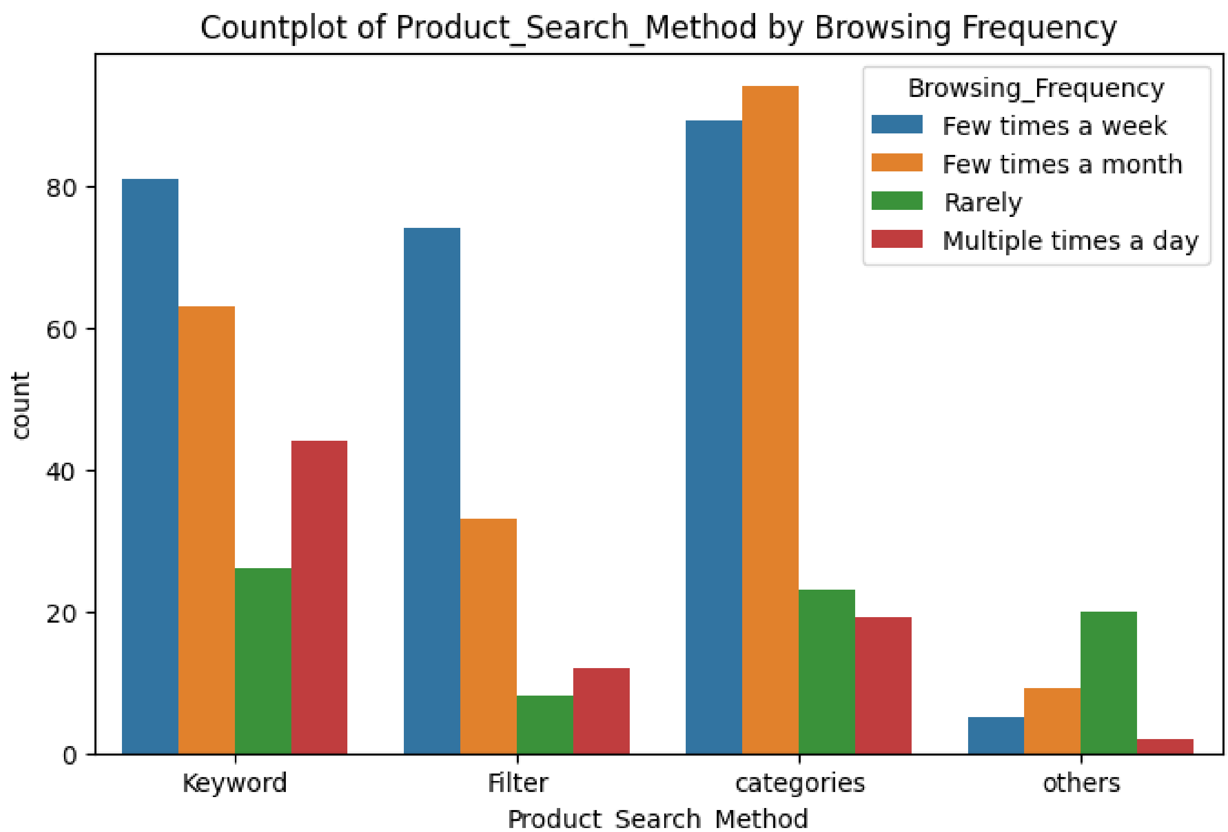
Countplot of Gender by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
160
140
120
100
80
60
40
20
0
Female
Prefer not to say
Male
Others
Gender



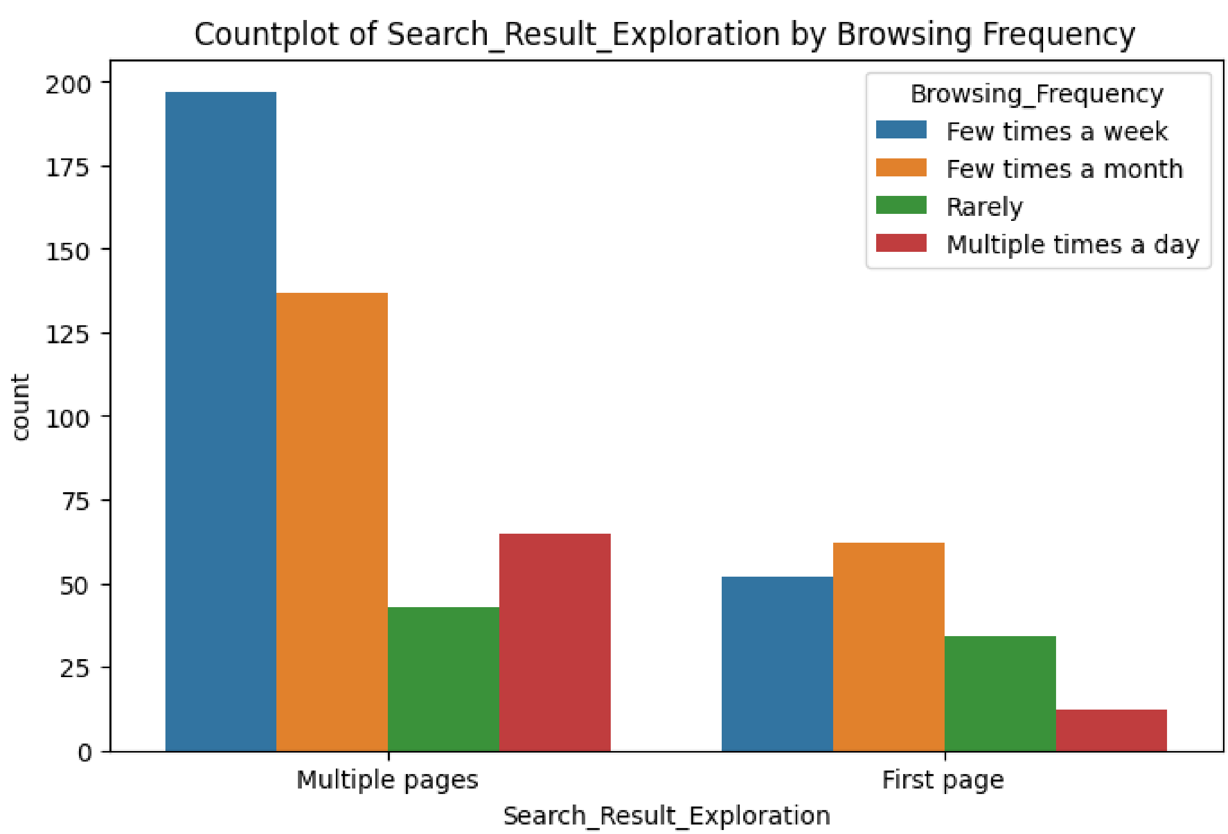
Countplot of Purchase_Frequency by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
100
80
60
40
20
0
Few times a month
Once a month
Less than once a month
Multiple times a week
Once a week
Purchase_Frequency


Countplot of Purchase_Categories by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
Purchase_Categories
0
10
20
30
40
50

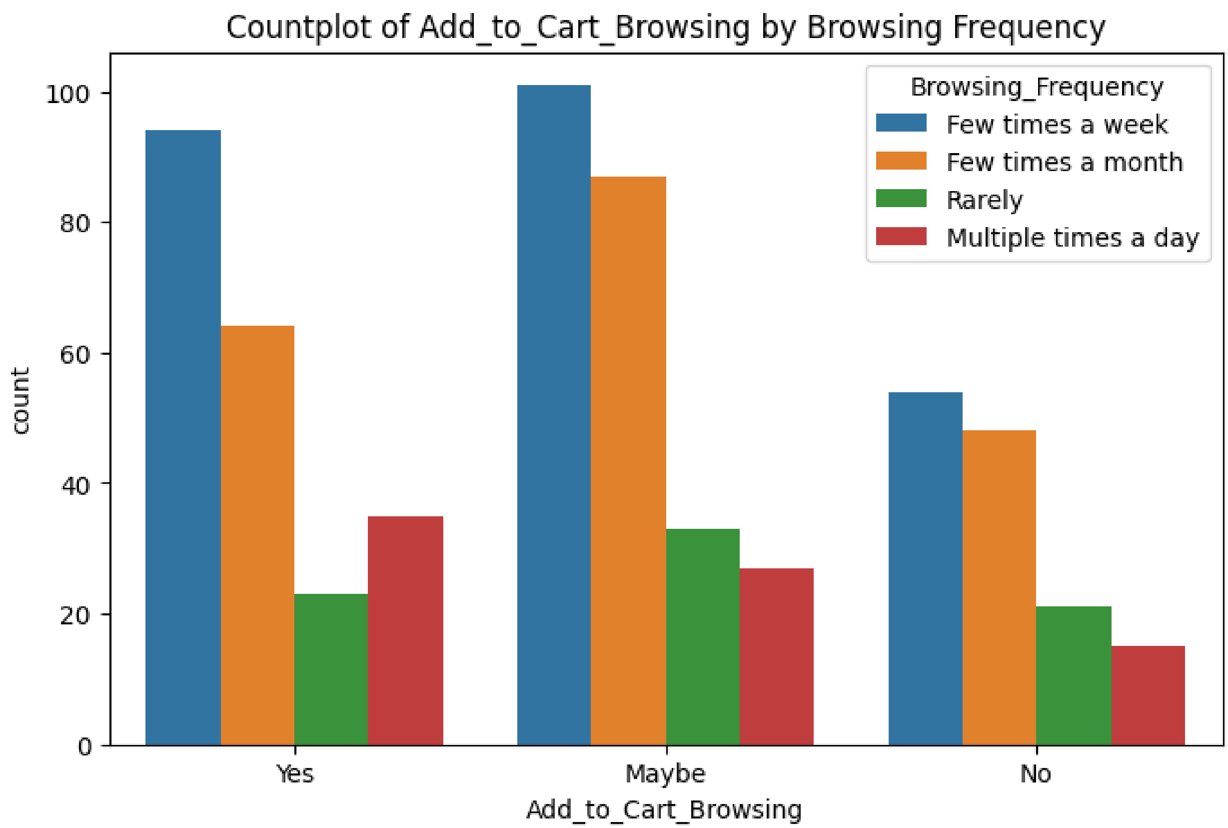

Countplot of Browsing_Frequency by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
50
100
150
200
250
Few times a week
Few times a month
Rarely
Multiple times a day
Browsing_Frequency

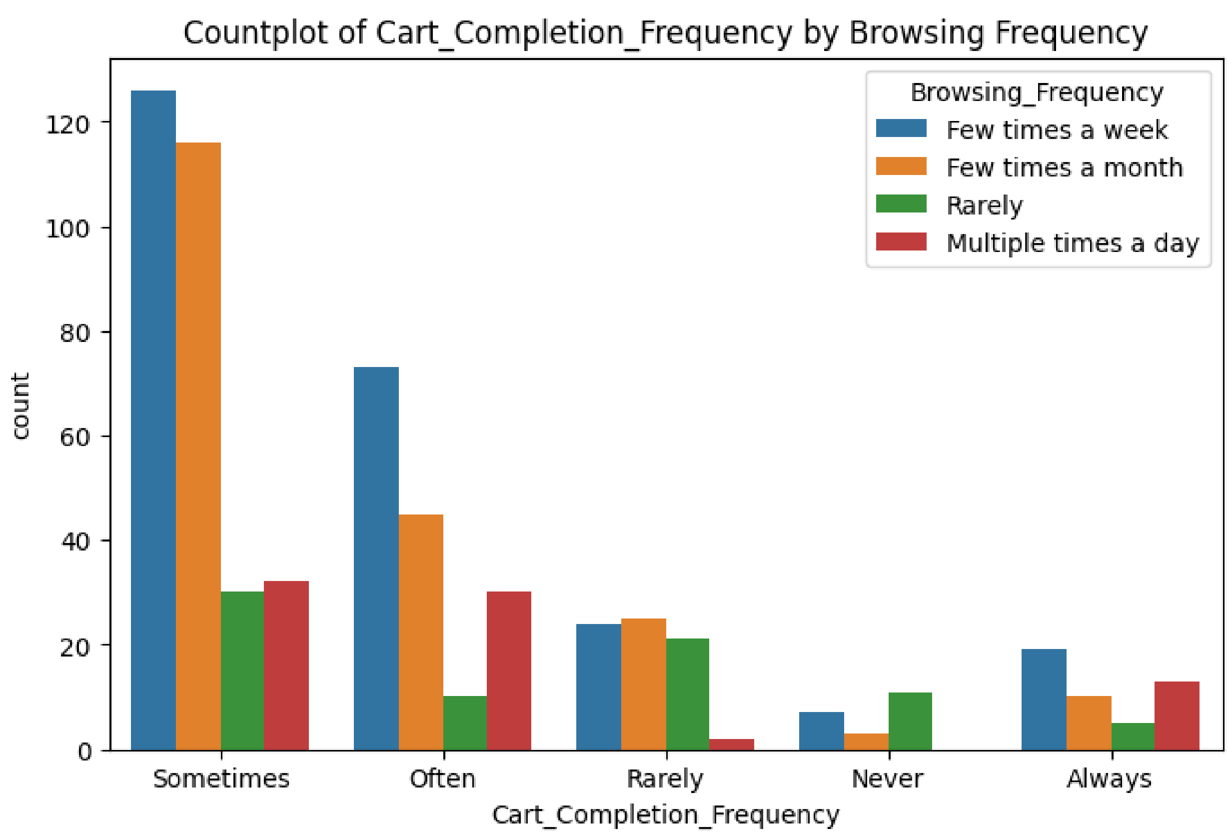


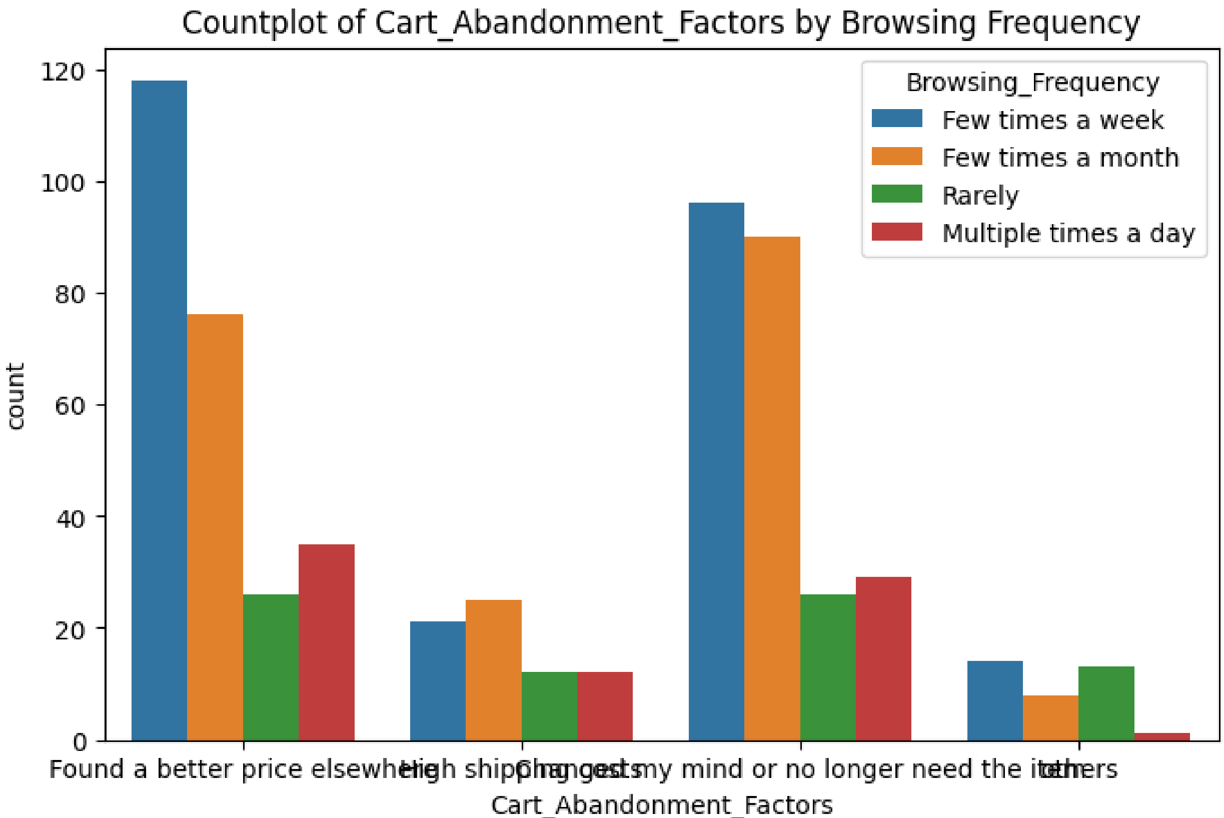
Countplot of Product_Search_Method by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
20
40
60
80
Keyword
Filter
categories
others
Product_Search_Method

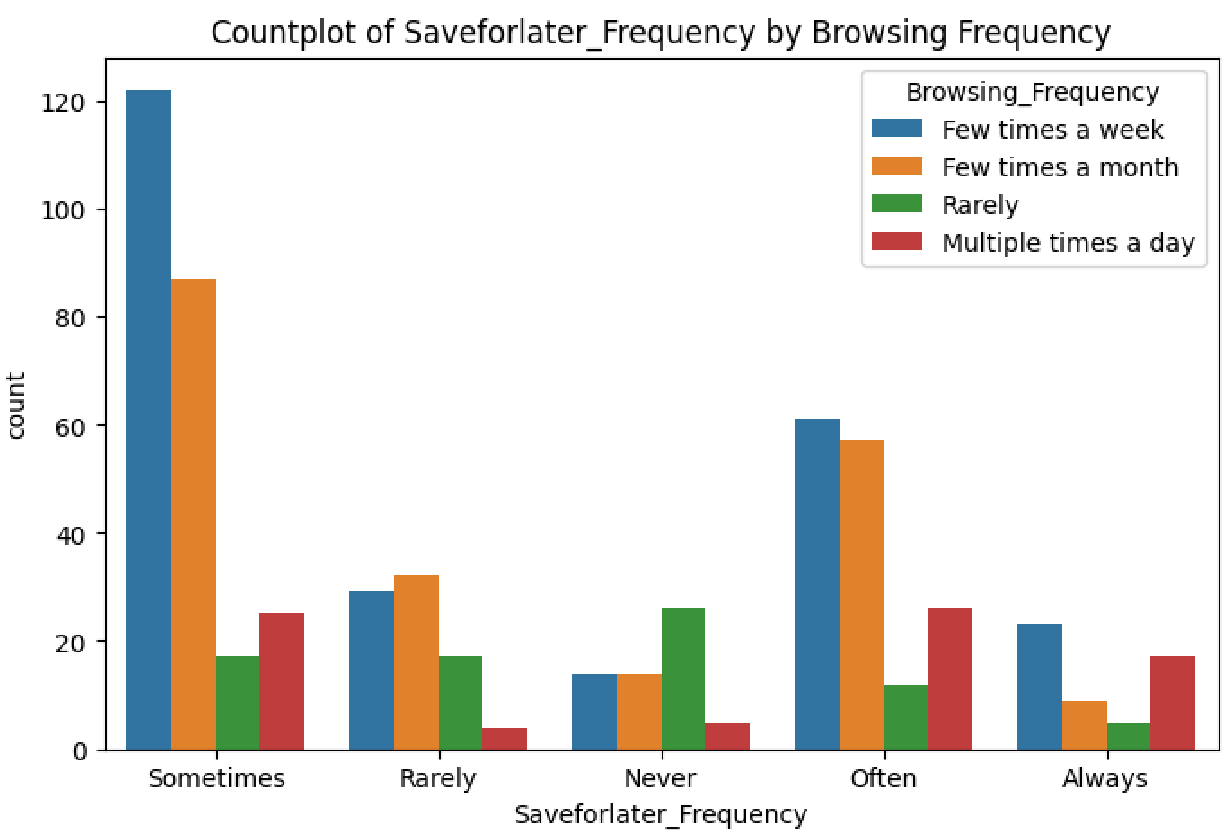


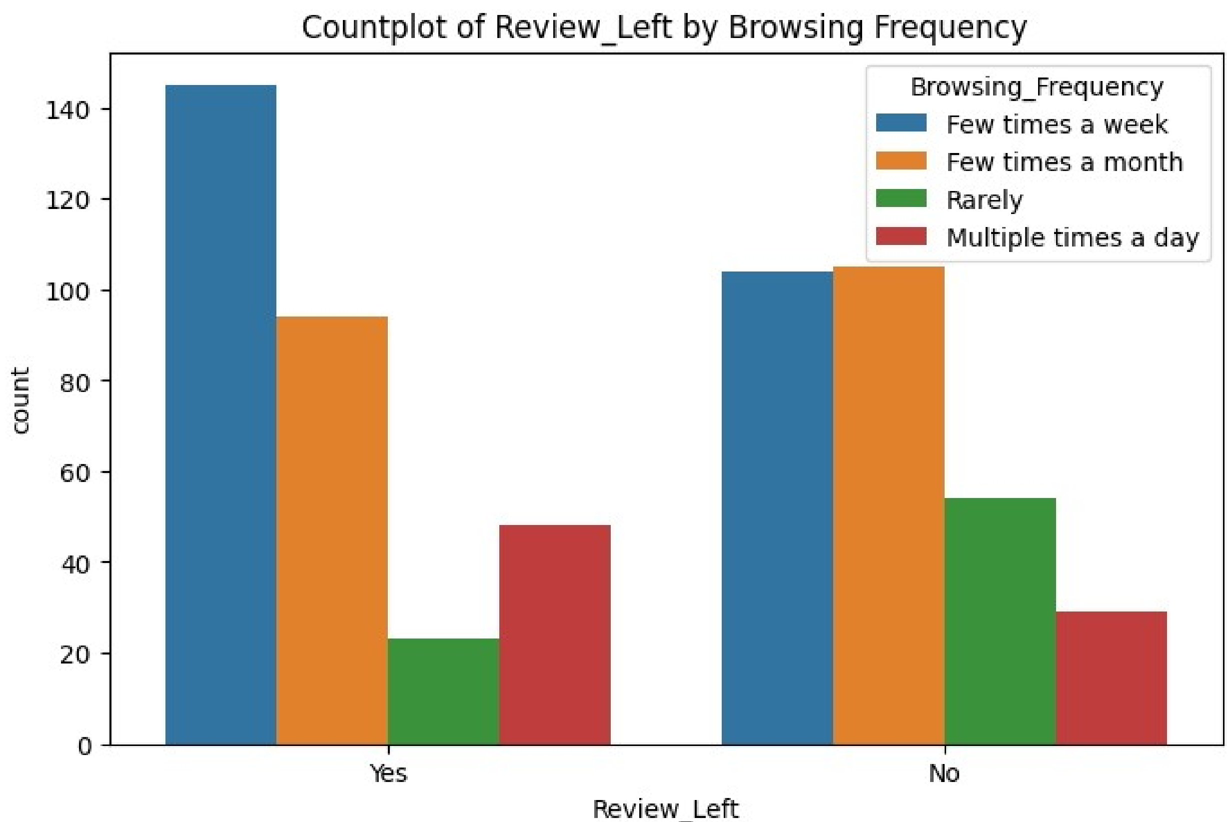
Countplot of Search_Result_Exploration by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
25
50
75
100
125
150
175
200
Multiple pages
First page
Search_Result_Exploration


Countplot of Add_to_Cart_Browsing by Browsing Frequency
count
0
20
40
60
80
100
Yes
Maybe
No
Add_to_Cart_Browsing
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day

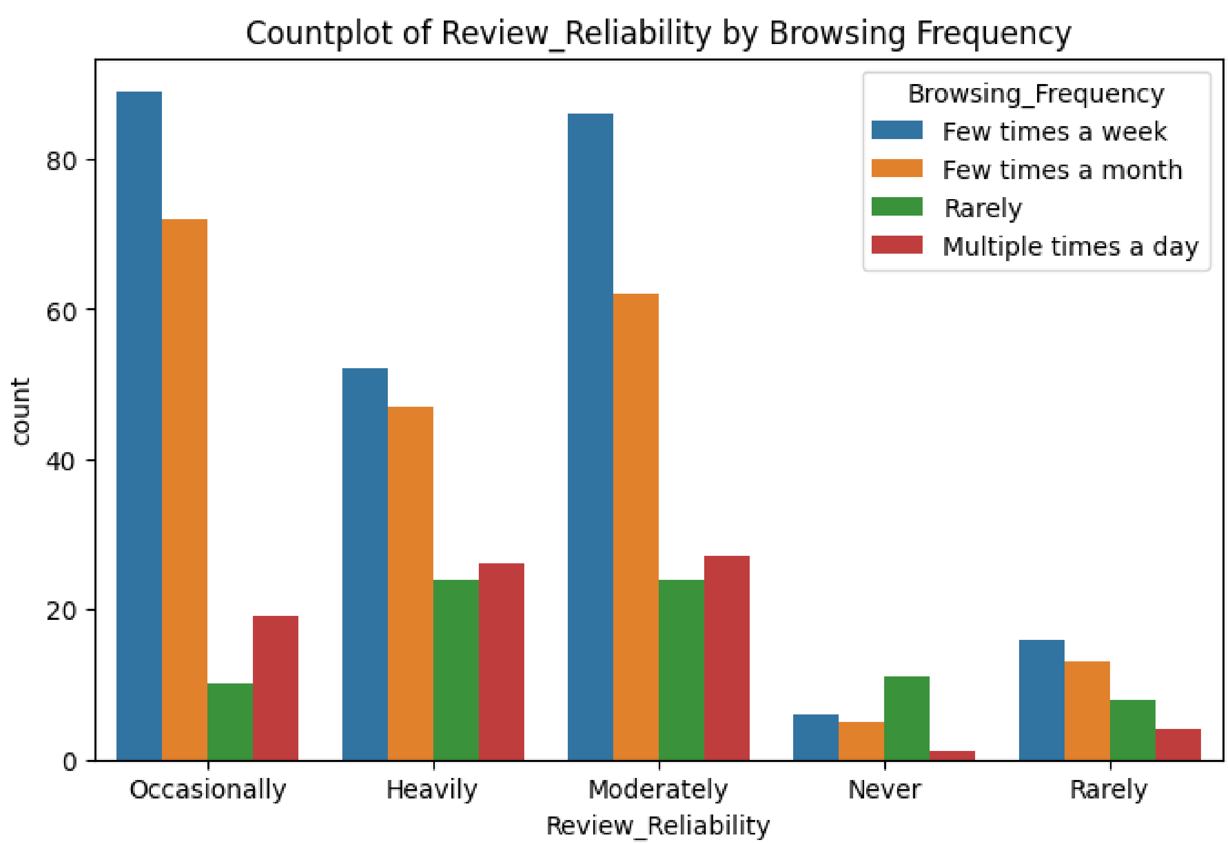

Countplot of Cart_Completion_Frequency by Browsing Frequency
count
0
20
40
60
80
100
120
Sometimes
Often
Rarely
Never
Always
Cart_Completion_Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day


Countplot of Cart_Abandonment_Factors by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
20
40
60
80
100
120
Found a better price elsewhere
High shipping costs
Changed my mind or no longer need the item
others
Cart_Abandonment_Factors

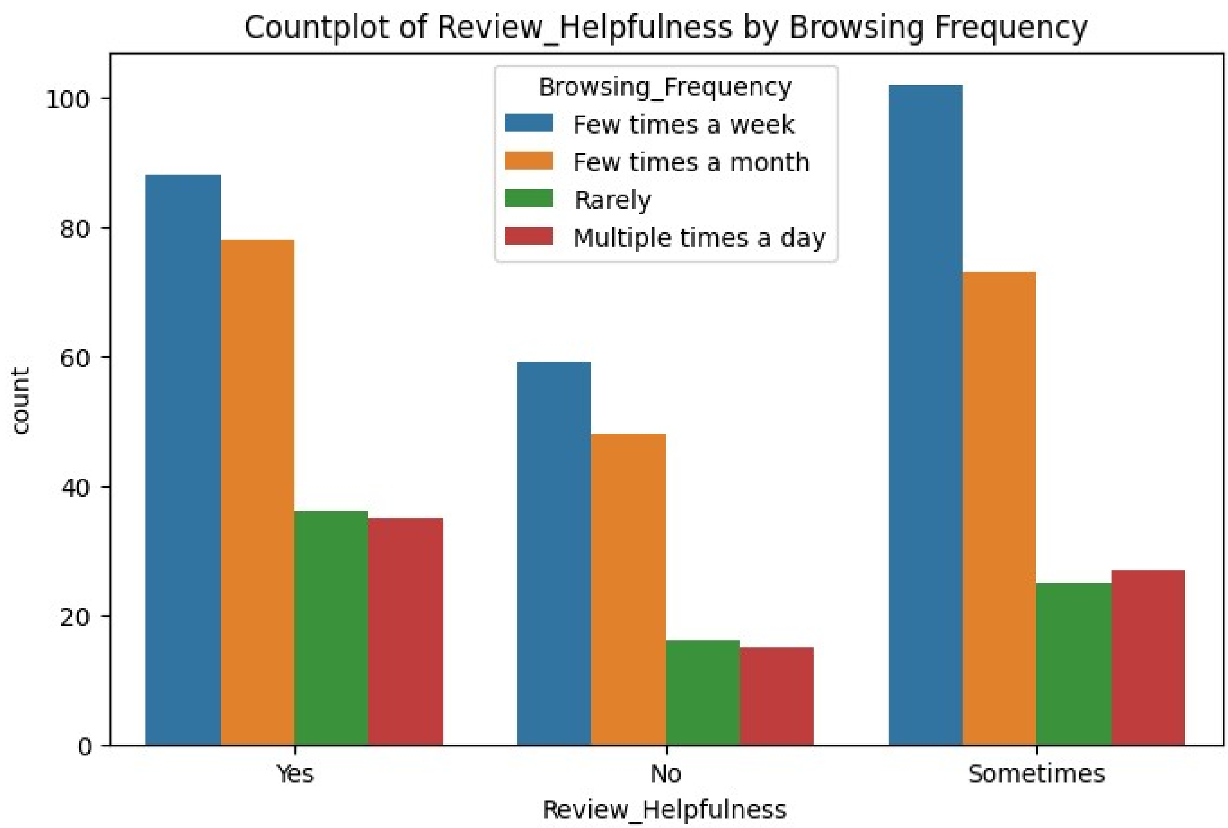

Countplot of Saveforlater_Frequency by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
20
40
60
80
100
120
Sometimes
Rarely
Never
Often
Always
Saveforlater_Frequency

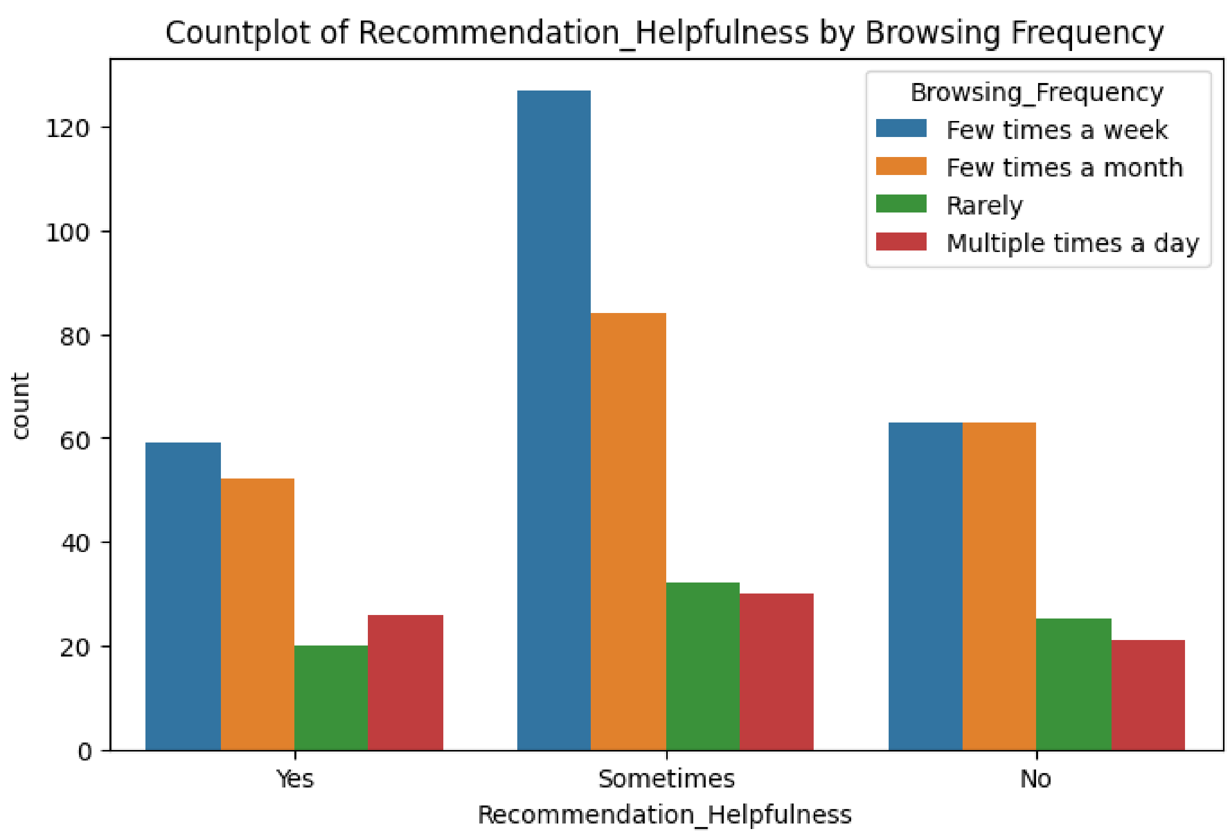


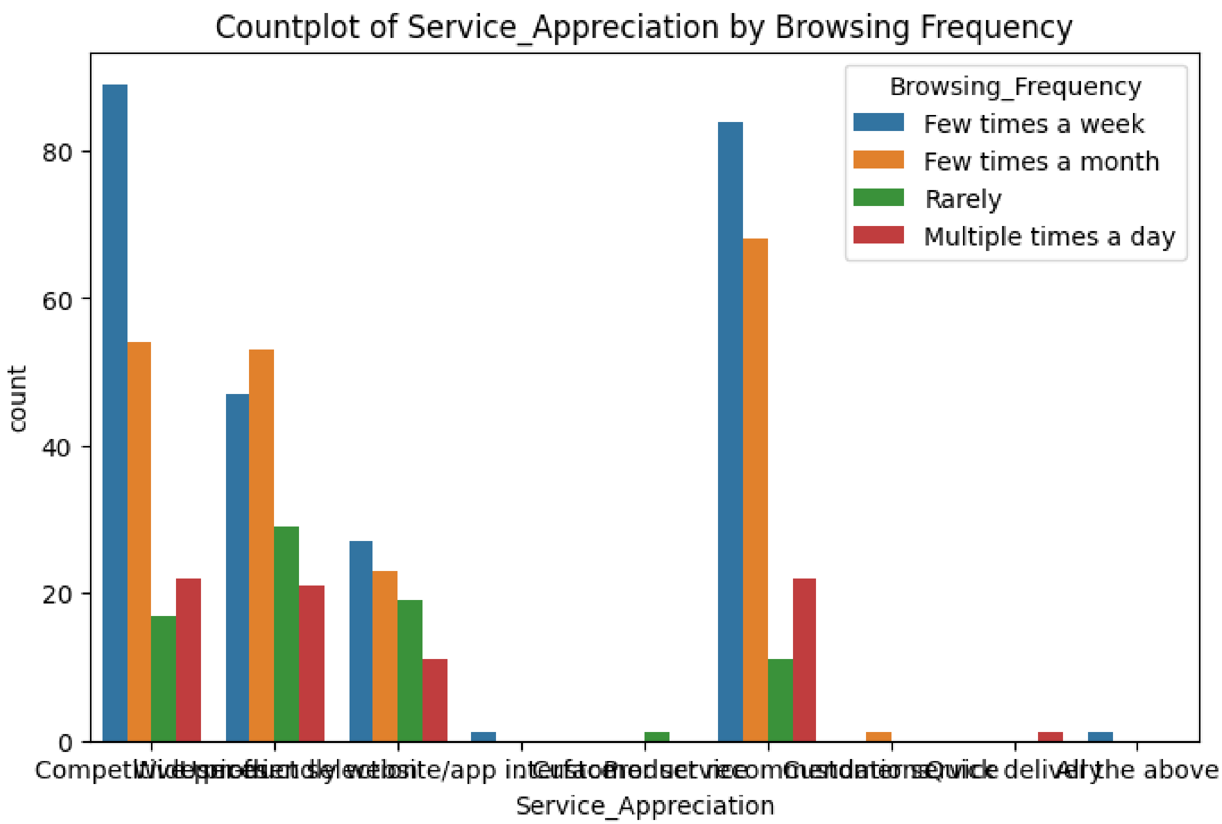
Countplot of Review_Left by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
20
40
60
80
100
120
140
Yes
No
Review_Left



Countplot of Review_Reliability by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
20
40
60
80
Occasionally
Heavily
Moderately
Never
Rarely
Review_Reliability


Countplot of Review_Helpfulness by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
20
40
60
80
100
Yes
No
Sometimes
Review_Helpfulness


Countplot of Recommendation_Helpfulness by Browsing Frequency
Browsing_Frequency
Few times a week
Few times a month
Rarely
Multiple times a day
count
0
20
40
60
80
100
120
Yes
Sometimes
No
Recommendation_Helpfulness



-The purchase frequency increases with age.

-Most consumers will purchase few times a month or multiple times a week.

-Personalized recommendation frequency sometimes affects the purchase frequency.

-Most consumers purchase frequently when reviews are left.

-Most consumers purchase frequently when the rating is 5 stars.

-The gender distribution shows that femaes lead and they purchase few times a week or few times a month.

-The purchase frequency increases when the shopping satisfaction is between 4-5 stars.

-The purchase frequency increases when the browsing frequency is high,ie, few times a week.

-The purchase categories are affected when the purchase frequency is few times a week.

The personalized recommendation frequency sometimes affects the browsing frequency.

-The highest rate for browsing is few times a week.

-The frequetly browsed product search method is categories which occurs few times a month.

-Consumers tend to browse multiple pages frequently during search result exploration

-The browsing frequency is high when the add to cart browsing is maybe.

-The browsing frequency is high when the cart completion is sometimes.

-The browsing frequency increases when the cart abandonment factors are found a better price elsewhere.

-When the save for later frequency is sometimes, the browsing frequency increases.

-The browsing frequency increases when reviews are left

-When the review reliability is moderate and occassional, the browsing frequency is high

-The review helpfullness sometimes affects the browsing frequency.

-The recommendation frequency sometimes affects the browsing frequency

-When the service appreciation is positive the browsing frequency is high.

-Improvement areas encourage browsing frequency.

```
column_names=df_Amazon_cleaned.columns
column_names_list=df_Amazon_cleaned.columns.tolist()
column_names
```

```
Index(['age', 'Gender', 'Purchase_Frequency', 'Purchase_Categories',
       'Personalized_Recommendation_Frequency', 'Browsing_Frequency',
       'Product_Search_Method', 'Search_Result_Exploration',
       'Customer_Reviews_Importance', 'Add_to_Cart_Browsing',
       'Cart_Completion_Frequency', 'Cart_Abandonment_Factors',
       'Saveforlater_Frequency', 'Review_Left', 'Review_Reliability',
       'Review_Helpfulness', 'Recommendation_Helpfulness',
       'Shopping_Satisfaction', 'Service_Appreciation'],
      dtype='object')
```

6. FEATURE ENGINEERING.

Feature engineering is a crucial step in preparing data for machine learning models.The goal is to extract meaningful information and create features that can potentially contribute to the understanding of consumer behavior.

```
from sklearn.pipeline import Pipeline

# Separate features and target variable
X = df_Amazon_cleaned.drop('Browsing_Frequency', axis=1)
y = df_Amazon_cleaned['Browsing_Frequency']

# Encode the target variable if it's categorical
le = LabelEncoder()
y = le.fit_transform(y)

# Split the data into training and testing sets
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define numerical and categorical features
numerical_features = X.select_dtypes(include=['int64',
'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Build the preprocessing pipeline
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Fit and transform the training data
X_train_preprocessed = preprocessor.fit_transform(X_train)

# Transform the test data using the trained transformer
X_test_preprocessed = preprocessor.transform(X_test)

#print the results
print('X_train_preprocessed_data', X_train_preprocessed)
print('X_test_preprocessed_data', X_test_preprocessed)
print('y_test_preprocessed_data',y_test)
print('y_train_preprocessed_data', y_train)
```

```
X_train_preprocessed_data   (0, 0)    -0.7498616728265725
  (0, 1)   -1.2502350100704027
  (0, 2)   1.542208046712811
  (0, 3)   1.0
  (0, 8)   1.0
  (0, 13)  1.0
  (0, 41)  1.0
  (0, 45)  1.0
  (0, 49)  1.0
  (0, 52)  1.0
  (0, 56)  1.0
  (0, 59)  1.0
  (0, 65)  1.0
  (0, 67)  1.0
  (0, 69)  1.0
```

```
  (0, 76)	1.0
  (0, 79)	1.0
  (0, 87)	1.0
  (1, 0)	0.4645607670036675
  (1, 1)	-0.3935925031703121
  (1, 2)	-0.43066980027884333
  (1, 3)	1.0
  (1, 7)	1.0
  (1, 14)	1.0
  (1, 42)	1.0
  :	:
  (479, 58)	1.0
  (479, 66)	1.0
  (479, 67)	1.0
  (479, 72)	1.0
  (479, 75)	1.0
  (479, 78)	1.0
  (479, 82)	1.0
  (480, 0)	-0.7498616728265725
  (480, 1)	-1.2502350100704027
  (480, 2)	-1.4171087237746705
  (480, 3)	1.0
  (480, 8)	1.0
  (480, 19)	1.0
  (480, 42)	1.0
  (480, 45)	1.0
  (480, 49)	1.0
  (480, 52)	1.0
  (480, 57)	1.0
  (480, 58)	1.0
  (480, 65)	1.0
  (480, 67)	1.0
  (480, 69)	1.0
  (480, 76)	1.0
  (480, 79)	1.0
  (480, 87)	1.0
X_test_preprocessed_data   (0, 0)	-0.4462560628690124
  (0, 1)	-1.2502350100704027
  (0, 2)	-1.4171087237746705
  (0, 3)	1.0
  (0, 8)	1.0
  (0, 40)	1.0
  (0, 41)	1.0
  (0, 45)	1.0
  (0, 49)	1.0
  (0, 52)	1.0
  (0, 57)	1.0
  (0, 60)	1.0
  (0, 64)	1.0
```

```
  (0, 68)	1.0
  (0, 70)	1.0
  (0, 76)	1.0
  (0, 79)	1.0
  (0, 84)	1.0
  (1, 0)	0.5657626369895208
  (1, 1)	0.46305000372977856
  (1, 2)	-0.43066980027884333
  (1, 3)	1.0
  (1, 10)	1.0
  (1, 38)	1.0
  (1, 41)	1.0
  :	:
  (119, 59)	1.0
  (119, 64)	1.0
  (119, 68)	1.0
  (119, 72)	1.0
  (119, 74)	1.0
  (119, 78)	1.0
  (119, 82)	1.0
  (120, 0)	1.4765794668622008
  (120, 1)	-0.3935925031703121
  (120, 2)	-0.43066980027884333
  (120, 3)	1.0
  (120, 11)	1.0
  (120, 27)	1.0
  (120, 41)	1.0
  (120, 46)	1.0
  (120, 49)	1.0
  (120, 51)	1.0
  (120, 55)	1.0
  (120, 58)	1.0
  (120, 64)	1.0
  (120, 67)	1.0
  (120, 70)	1.0
  (120, 74)	1.0
  (120, 77)	1.0
  (120, 82)	1.0
y_test_preprocessed_data [0 1 1 3 2 2 0 0 3 0 3 1 1 1 1 1 1 2 1 1 0 0 0 1 2 1 3 2 1 1 1 2 2 1 1 0 1
 0 1 0 0 0 3 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 3 0 1 3 1 1 1 0 0 1 1 1 0 3 2 2
 0 1 2 0 0 1 1 0 0 1 0 1 0 1 3 3 1 3 1 1 1 1 1 1 1 1 2 2 1 0 2 0 0 1 1 1 1
 1 1 1 1 0 2 1 1 0 2]
y_train_preprocessed_data [3 2 0 3 0 0 2 1 0 3 1 1 1 0 0 0 3 0 1 3 3 1 2 0 3 2 0 2 1 1 0 2 2 1 0 1 1
 2 3 2 1 0 0 1 1 1 1 1 2 0 1 2 1 0 0 1 0 3 1 1 1 2 0 0 1 2 1 1 1 0 0 0 0 2
```

```
 1 1 1 2 2 0 0 1 0 1 1 1 0 1 2 3 1 0 0 3 0 2 3 0 1 0 0 1 2 1 1 2 3 0 2
1 1
 1 1 2 0 3 0 3 1 0 0 3 2 1 2 1 0 1 1 3 0 1 1 0 0 2 1 0 1 3 1 1 0 2 0 1
0 0
 0 2 3 3 1 0 0 1 0 1 1 1 0 2 0 2 1 3 0 1 0 2 0 3 3 1 1 1 0 0 0 1 1 1 2
1 1
 3 1 3 0 3 0 1 0 0 1 1 3 1 2 1 1 1 1 3 2 0 3 3 0 2 2 2 3 3 1 0 3 3 2 1
0 1
 3 1 1 1 1 0 0 0 1 3 2 0 1 2 0 0 3 3 3 0 2 0 1 0 0 3 1 1 0 1 1 0 0 2 0
1 1
 0 0 1 1 3 1 0 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 3 0 0 0 0
3 0
 3 1 3 1 2 1 0 0 1 2 1 2 1 0 1 0 1 0 0 1 3 3 0 0 1 2 1 1 1 1 0 0 2 0 0
2 1
 1 0 0 1 0 2 1 1 1 1 2 1 0 0 3 3 3 2 1 0 1 1 0 0 3 1 0 0 2 0 0 3 3 0 1
0 1
 1 3 1 1 1 0 0 0 3 0 0 1 1 2 0 3 1 3 1 0 3 3 2 1 0 3 0 0 1 1 0 0 1 0 0
1 2
 0 1 0 1 1 0 1 0 1 0 0 0 3 1 2 0 0 1 1 3 1 3 1 2 0 1 1 1 1 0 3 0 3 0 2
0 1
 1 1 1 1 0 0 2 1 1 1 0 0 0 0 2 1 2 0 0 1 1 2 1 0 1 0 0 3 1 0 0 0 2 0 1
0 0]
```

The above process includes:

Dropping unnecessary columns.

Performing train-test split.

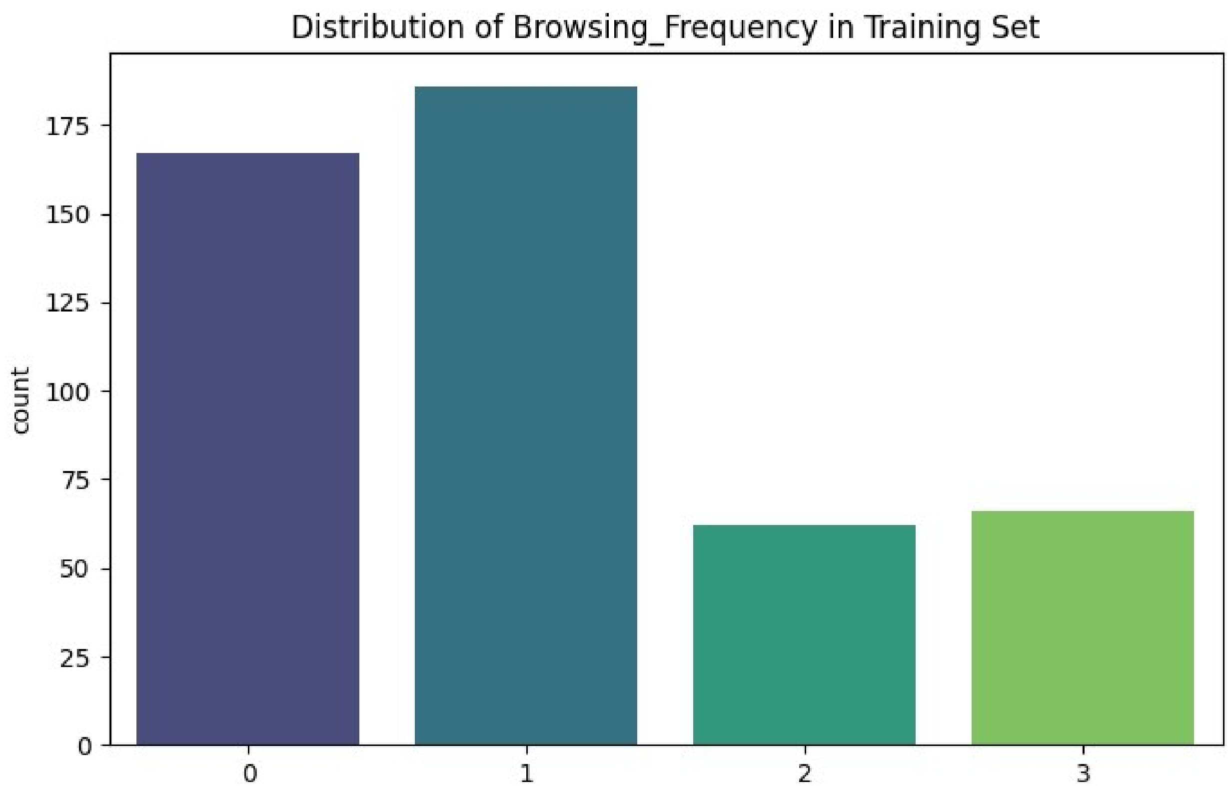
Handling missing values for numerical and categorical features.

Scaling numerical features.

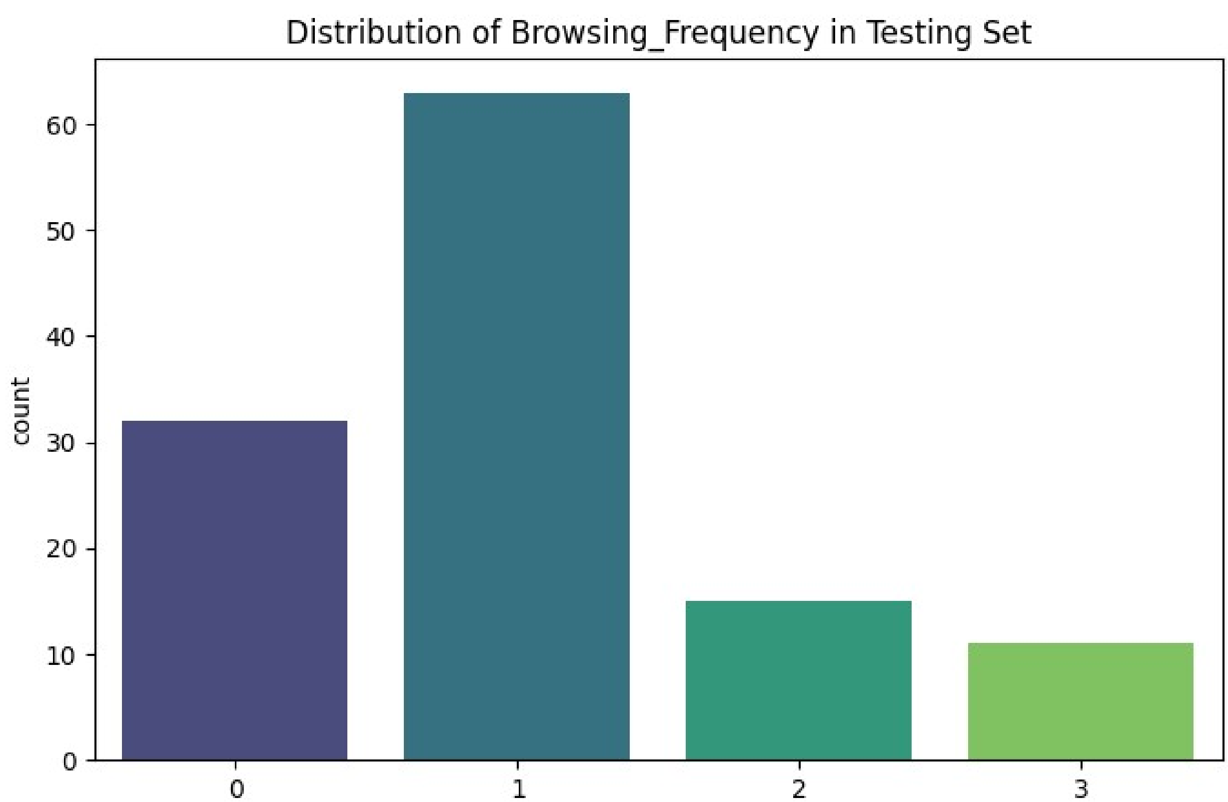
Encoding categorical features.

CHECK FOR IMBALANCED CLASSES.

```
# Plot the distribution of the target variable in the training set
plt.figure(figsize=(8, 5))
sns.countplot(x=y_train, palette='viridis')
plt.title('Distribution of Browsing_Frequency in Training Set')
plt.show()

# Plot the distribution of the target variable in the testing set
plt.figure(figsize=(8, 5))
sns.countplot(x=y_test, palette='viridis')
plt.title('Distribution of Browsing_Frequency in Testing Set')
plt.show()
```



Distribution of Browsing_Frequency in Training Set
count
0
25
50
75
100
125
150
175
0
1
2
3



Distribution of Browsing_Frequency in Testing Set
count
0
10
20
30
40
50
60
0
1
2
3

This has created two bar plots showing the count of each class in the Browsing_Frequency variable for both the training and testing sets. It helps identify if there is a significant class imbalance.

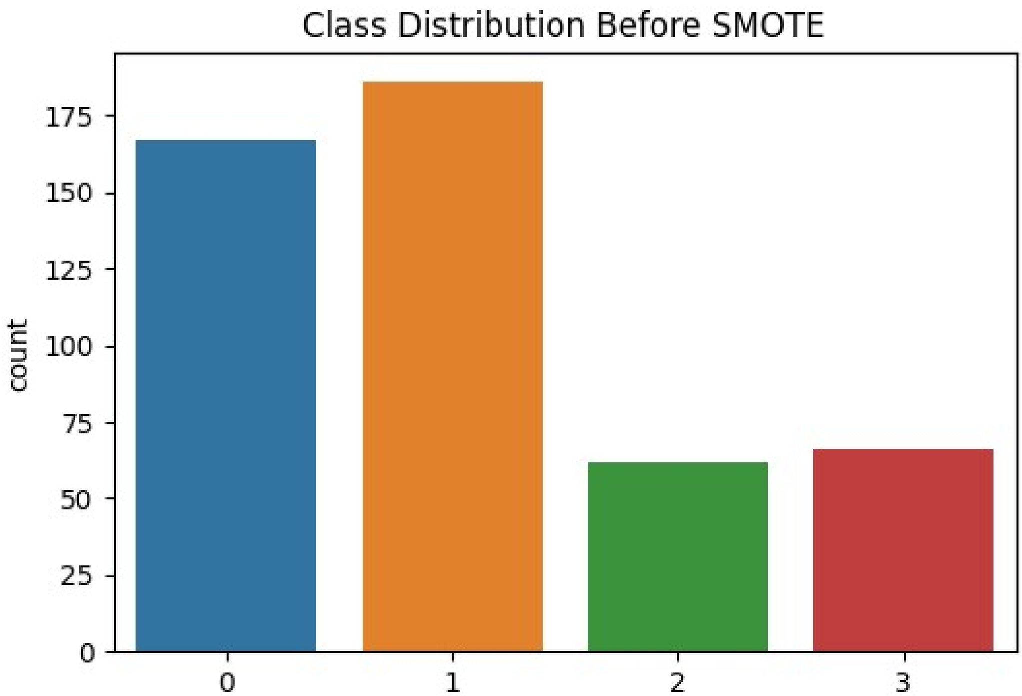
If there is a severe imbalance,consider techniques like oversampling the minority class, undersampling the majority class, or using different evaluation metrics that are robust to imbalanced classes (e.g., precision, recall, F1-score).

If the classes are reasonably balanced,proceed with training the machine learning model.

```
!pip install imbalanced-learn
from imblearn.over_sampling import SMOTE

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled =
smote.fit_resample(X_train_preprocessed, y_train)
```

```
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.10/dist-packages (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.23.5)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.2.2)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(3.2.0)
```

```
#Target distribution before SMOTE.
import matplotlib.pyplot as plt
import seaborn as sns

#  y_train is the target variable
plt.figure(figsize=(6, 4))
sns.countplot(x=y_train)
plt.title('Class Distribution Before SMOTE')
plt.show()
```

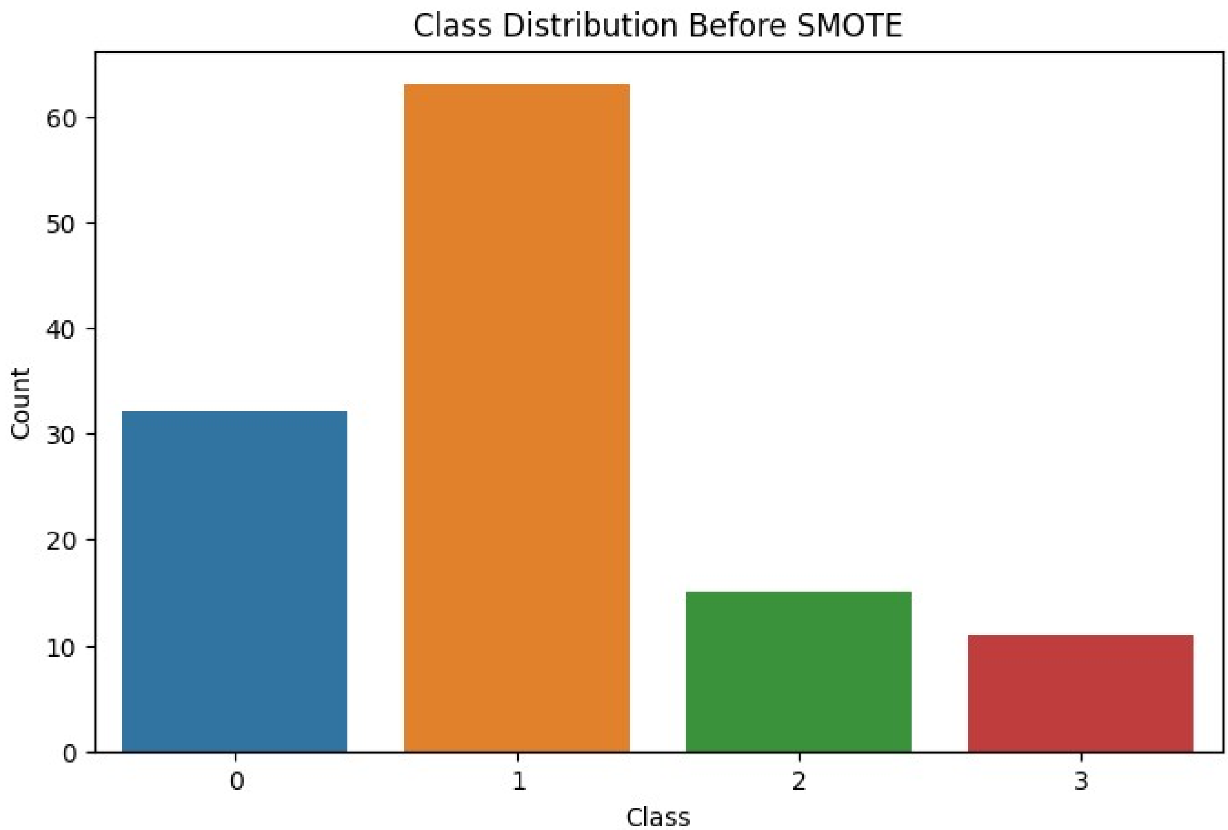


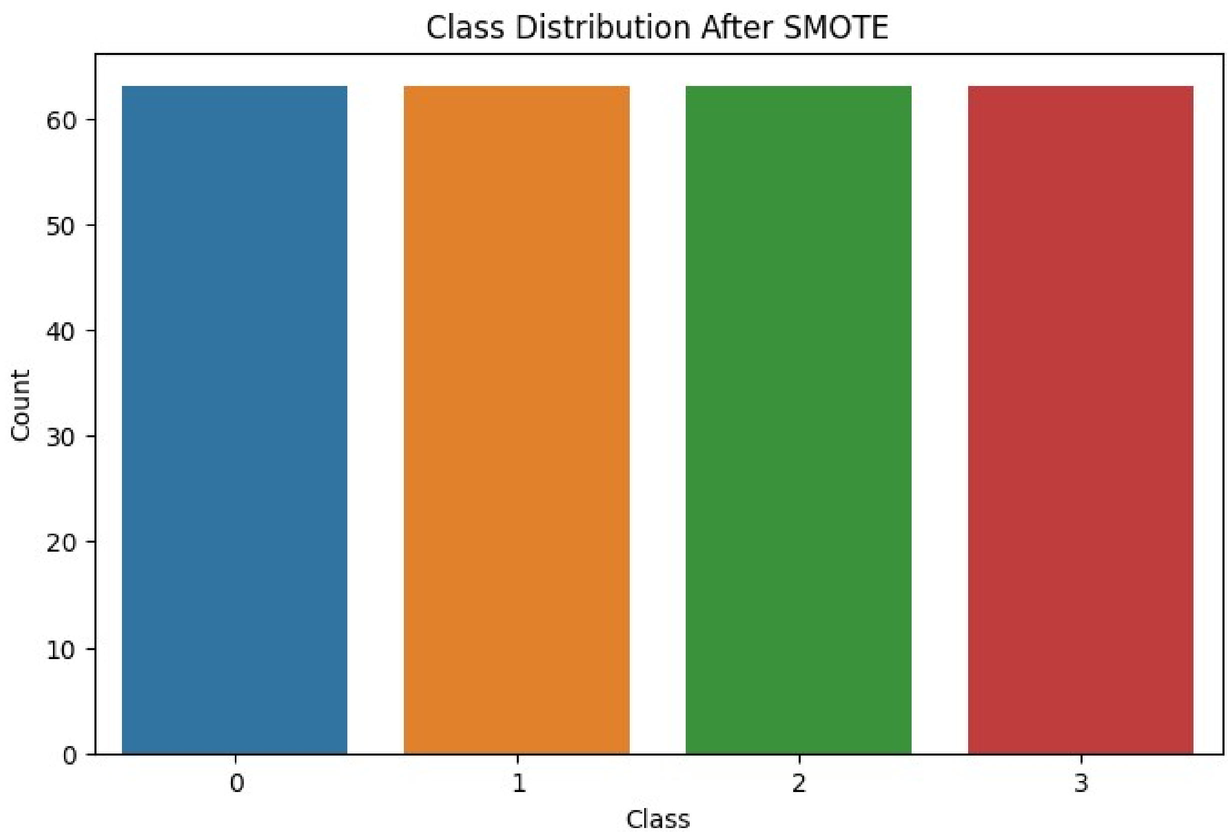

```
#Apply SMOTE
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled =
smote.fit_resample(X_train_preprocessed, y_train)

#After SMOTE
plt.figure(figsize=(6, 4))
sns.countplot(x=y_train_resampled)
plt.title('Class Distribution After SMOTE')
plt.show()
```

SMOTE has sccessfully Oversampled the minority class.

```
from imblearn.over_sampling import SMOTE
# Create a SMOTE object
smote = SMOTE(random_state=42)

# Apply SMOTE to X_test and y_test
X_test_resampled, y_test_resampled =
smote.fit_resample(X_test_preprocessed, y_test)

import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

# Function to plot class distribution
def plot_class_distribution(y, title):
    class_counts = Counter(y)
    classes = list(class_counts.keys())
    counts = list(class_counts.values())

    plt.figure(figsize=(8, 5))
    sns.barplot(x=classes, y=counts)
    plt.title(title)
    plt.xlabel('Class')
    plt.ylabel('Count')
    plt.show()

# Plot class distribution before SMOTE
```

```
plot_class_distribution(y_test, title='Class Distribution Before SMOTE')

# Plot class distribution after SMOTE
plot_class_distribution(y_test_resampled, title='Class Distribution After SMOTE')
```





Now X-test and y_test are balanced.

7.) MODELLING.

Given my objectives, I will incorporate classification, clustering (for customer segmentation), and potentially regression for predictive analytics.

Here's a breakdown:

Customer Segmentation (Clustering):

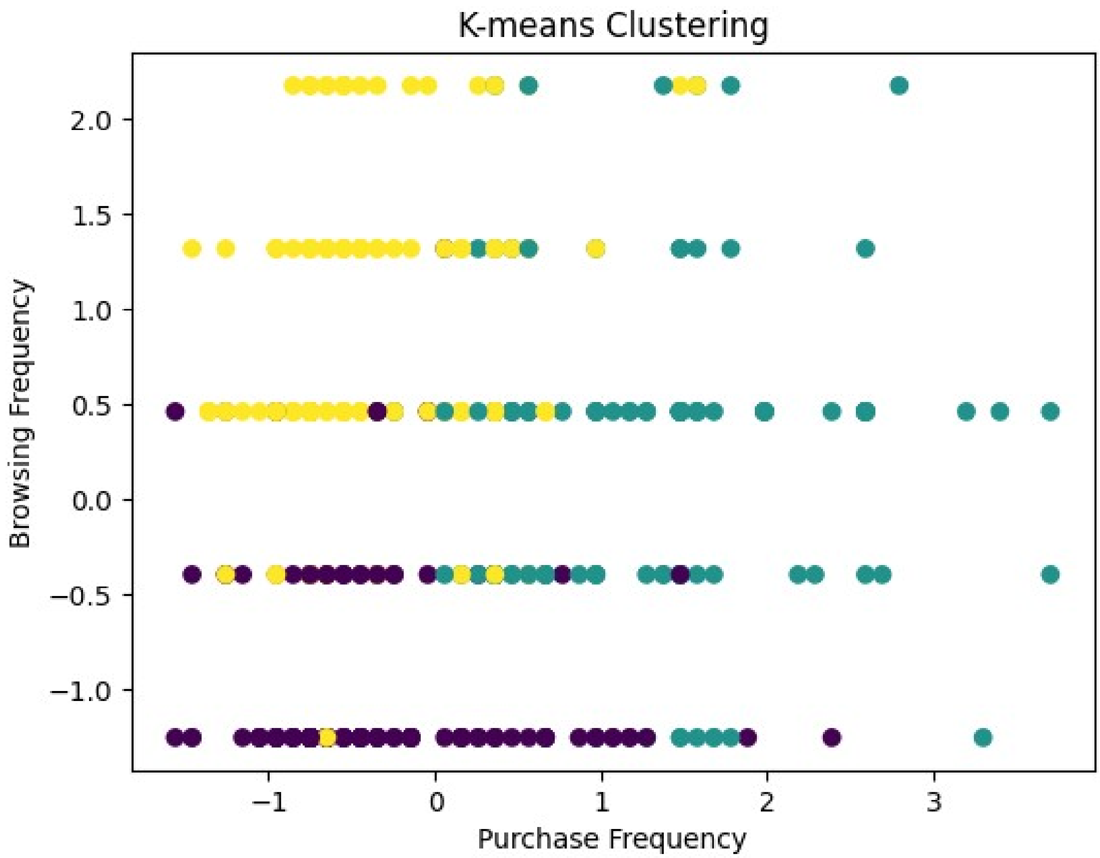
clustering algorithms like K-means or hierarchical clustering to group customers based on their behavior and preferences.

Conversion Rate Optimization (Binary Classification):

If you're predicting whether a user will convert or not, this becomes a binary classification problem. Logistic Regression, Decision Trees, or Random Forests are common choices.

7.1) K-MEANS

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

# X_train_preprocessed is the preprocessed training data (sparse matrix)
# Convert sparse matrix to dense format
X_train_dense = X_train_preprocessed.toarray()

# Choose the number of clusters (K)
k = 3

# Apply K-means clustering
kmeans = KMeans(n_clusters=k, random_state=42)
clusters = kmeans.fit_predict(X_train_dense)

# Visualize clusters (for the first two features)
plt.scatter(X_train_dense[:, 0], X_train_dense[:, 1], c=clusters, cmap='viridis')
plt.title('K-means Clustering')
plt.xlabel('Purchase Frequency')
plt.ylabel('Browsing Frequency')
plt.show()
```



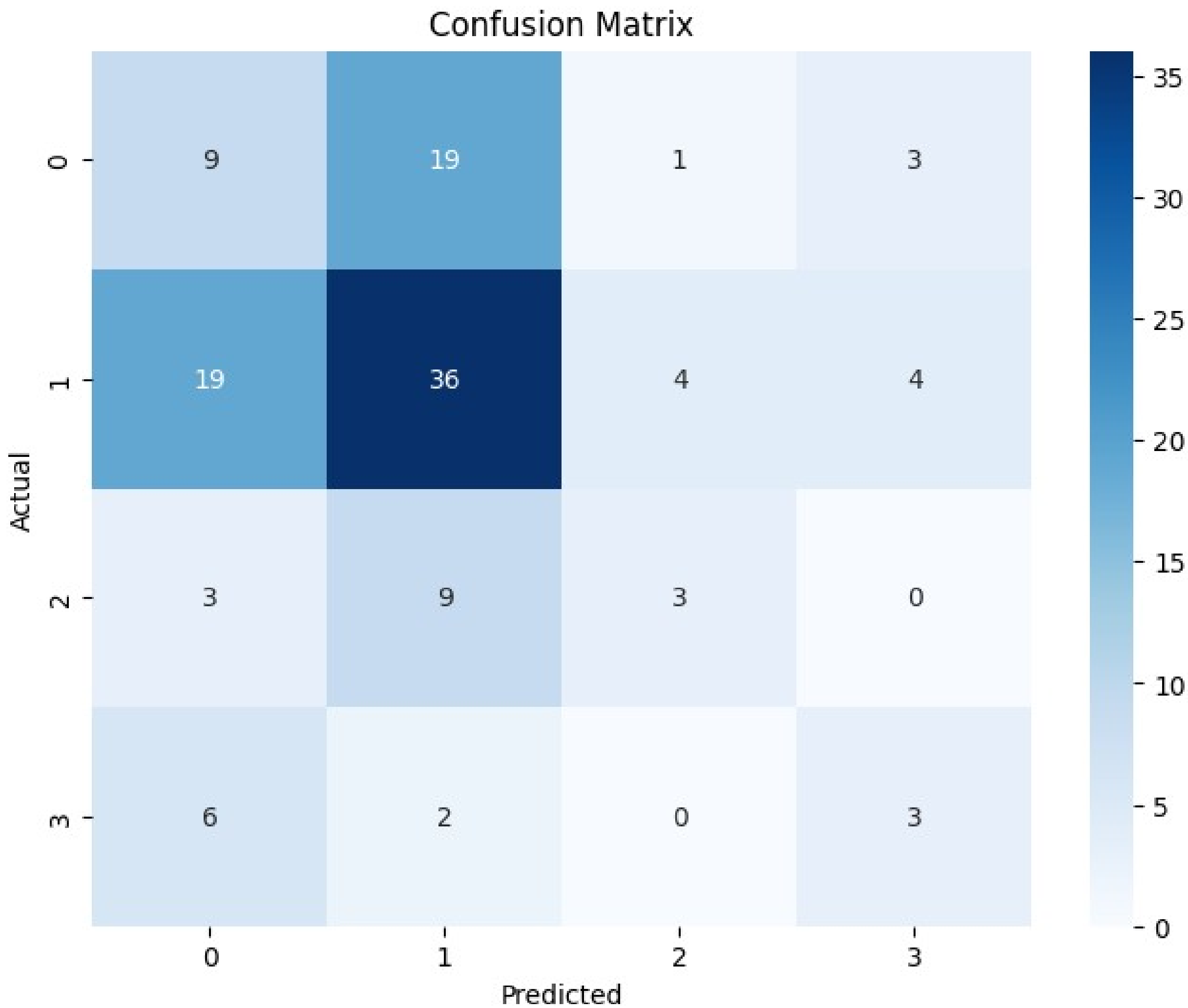
The above model has clustered consumers into two categories; Purchase_Frequency and Browsing_Frequency. The graph shows that the two features are higly correlated in that a high browsing frequency leads to high purchase frequency.

7.2) LOGISTIC REGRESSION.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

#X_train_preprocessed and X_test_preprocessed are the feature matrices
# and y_train and y_test are the target variables

# Create a logistic regression model
logreg_model = LogisticRegression(random_state=42)

# Fit the model tothe training data
logreg_model.fit(X_train_preprocessed, y_train)

# Make predictions on the test set
predictions = logreg_model.predict(X_test_preprocessed)
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
conf_matrix = confusion_matrix(y_test, predictions)
classification_rep = classification_report(y_test, predictions)

# Print the evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_rep)

# Calculate the confusion matrix
# Predict the labels for the test set
y_pred = logreg_model.predict(X_test_preprocessed)

conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=[0, 1, 2, 3], yticklabels=[0, 1, 2, 3])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy: 0.4215

Confusion Matrix:
[[ 9 19  1  3]
 [19 36  4  4]
 [ 3  9  3  0]
 [ 6  2  0  3]]

Classification Report:
              precision    recall  f1-score   support

           0       0.24      0.28      0.26        32
           1       0.55      0.57      0.56        63
           2       0.38      0.20      0.26        15
           3       0.30      0.27      0.29        11

    accuracy                           0.42       121
   macro avg       0.37      0.33      0.34       121
weighted avg       0.42      0.42      0.42       121
```



Accuracy: 0.4215

Accuracy is the proportion of correctly classified instances out of the total instances. In this case, the model is correctly predicting the class labels for approximately 42.15% of the samples in the test set.

Confusion Matrix

Each row in the matrix represents the actual class, and each column represents the predicted class. For example, the entry in the first row and first column (9) indicates that 9 instances of class 0 were correctly predicted as class 0. The entry in the second row and third column (4) indicates that 4 instances of class 1 were incorrectly predicted as class 2.

Classification Report.

Precision: Indicates the proportion of true positive predictions among the instances predicted as a particular class.

Recall: Represents the proportion of true positive predictions among all actual instances of a particular class. The recall for class 1 is 0.57, indicating that 57% of all actual class 1 instances were correctly predicted.

F1-score: The harmonic mean of precision and recall. It provides a balance between precision and recall.

7.3) DECISION TREE.

```
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Create a Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)

# Train the model
decision_tree_model.fit(X_train_dense, y_train)

# Predict the labels for the test set
y_pred_decision_tree =
decision_tree_model.predict(X_test_preprocessed)

# Evaluate the model
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)
conf_matrix_decision_tree = confusion_matrix(y_test,
y_pred_decision_tree)
classification_report_decision_tree = classification_report(y_test,
y_pred_decision_tree)

# Print the results
print(f"Decision Tree Accuracy: {accuracy_decision_tree:.4f}\n")
print("Confusion Matrix:")
print(conf_matrix_decision_tree)
print("\nClassification Report:")
print(classification_report_decision_tree)
```

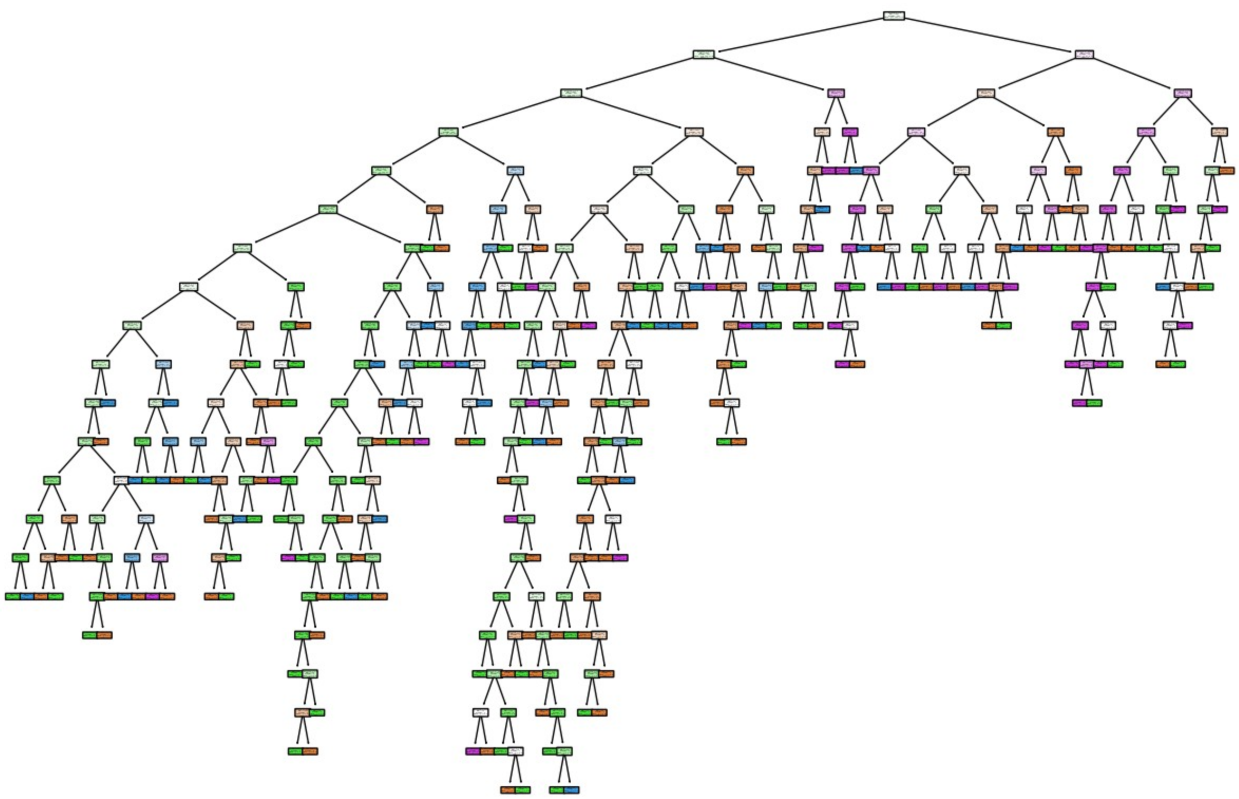
```
Decision Tree Accuracy: 0.3802

Confusion Matrix:
[[10 13  3  6]
 [19 26 11  7]
 [ 4  8  3  0]
 [ 1  0  3  7]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.29      0.31      0.30        32
           1       0.55      0.41      0.47        63
           2       0.15      0.20      0.17        15
           3       0.35      0.64      0.45        11

    accuracy                           0.38       121
   macro avg       0.34      0.39      0.35       121
weighted avg       0.42      0.38      0.39       121
```

```python
#feature names.
num_features = X_train_dense.shape[1]
feature_names = [f'Feature{i}' for i in range(num_features)]

#class names.
import numpy as np

class_names = np.unique(y_train).tolist()
# Print Feature Names
print("Feature Names:")
print(feature_names)

# Print Class Names
print("\nClass Names:")
print(class_names)
# Convert class names to strings
class_names = list(map(str, class_names))



# Plot the Decision Tree
plt.figure(figsize=(15, 10))
plot_tree(decision_tree_model, feature_names=feature_names,
class_names=class_names, filled=True, rounded=True)
plt.show()
```

```
Feature Names:
['Feature0', 'Feature1', 'Feature2', 'Feature3', 'Feature4',
'Feature5', 'Feature6', 'Feature7', 'Feature8', 'Feature9',
'Feature10', 'Feature11', 'Feature12', 'Feature13', 'Feature14',
'Feature15', 'Feature16', 'Feature17', 'Feature18', 'Feature19',
'Feature20', 'Feature21', 'Feature22', 'Feature23', 'Feature24',
'Feature25', 'Feature26', 'Feature27', 'Feature28', 'Feature29',
'Feature30', 'Feature31', 'Feature32', 'Feature33', 'Feature34',
'Feature35', 'Feature36', 'Feature37', 'Feature38', 'Feature39',
'Feature40', 'Feature41', 'Feature42', 'Feature43', 'Feature44',
'Feature45', 'Feature46', 'Feature47', 'Feature48', 'Feature49',
```

```
'Feature50', 'Feature51', 'Feature52', 'Feature53', 'Feature54',
'Feature55', 'Feature56', 'Feature57', 'Feature58', 'Feature59',
'Feature60', 'Feature61', 'Feature62', 'Feature63', 'Feature64',
'Feature65', 'Feature66', 'Feature67', 'Feature68', 'Feature69',
'Feature70', 'Feature71', 'Feature72', 'Feature73', 'Feature74',
'Feature75', 'Feature76', 'Feature77', 'Feature78', 'Feature79',
'Feature80', 'Feature81', 'Feature82', 'Feature83', 'Feature84',
'Feature85', 'Feature86', 'Feature87']

Class Names:
[0, 1, 2, 3]
```



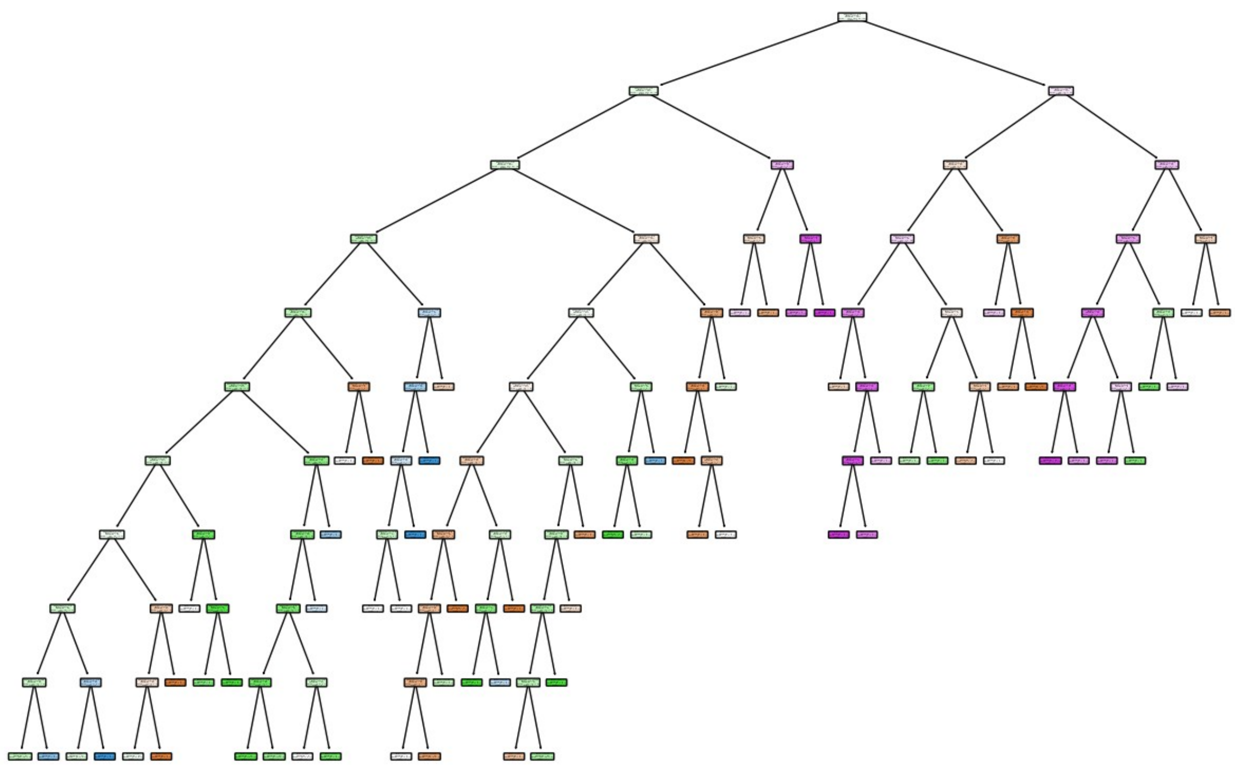
Gridsearch CV

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
# Create a Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)

# Use GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(decision_tree_model, param_grid, cv=5,
scoring='accuracy')
grid_search.fit(X_train_dense, y_train)

# Get the best parameters and retrain the model
best_params = grid_search.best_params_
best_decision_tree_model = grid_search.best_estimator_

# Print the best parameters
print("Best Parameters:", best_params)

# Retrain the model with the best parameters
best_decision_tree_model.fit(X_train_dense, y_train)

# Predict the labels for the test set
y_pred_decision_tree =
best_decision_tree_model.predict(X_test_preprocessed)

# Evaluate the model
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)
conf_matrix_decision_tree = confusion_matrix(y_test,
y_pred_decision_tree)
classification_report_decision_tree = classification_report(y_test,
y_pred_decision_tree)

# Print the results
print(f"Decision Tree Accuracy: {accuracy_decision_tree:.4f}\n")
print("Confusion Matrix:")
print(conf_matrix_decision_tree)
print("\nClassification Report:")
print(classification_report_decision_tree)

# Plot the Decision Tree with the best parameters
plt.figure(figsize=(15, 10))
plot_tree(best_decision_tree_model, feature_names=feature_names,
class_names=class_names, filled=True, rounded=True)
plt.show()
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'best'}
Decision Tree Accuracy: 0.4545

Confusion Matrix:
[[11 16  0  5]
 [18 36  4  5]
```

```
 [ 5  8  2  0]
 [ 0  4  1  6]]

Classification Report:
              precision    recall  f1-score   support

           0       0.32      0.34      0.33        32
           1       0.56      0.57      0.57        63
           2       0.29      0.13      0.18        15
           3       0.38      0.55      0.44        11

    accuracy                           0.45       121
   macro avg       0.39      0.40      0.38       121
weighted avg       0.45      0.45      0.45       121
```



7.4) ENSEMBLE METHODS

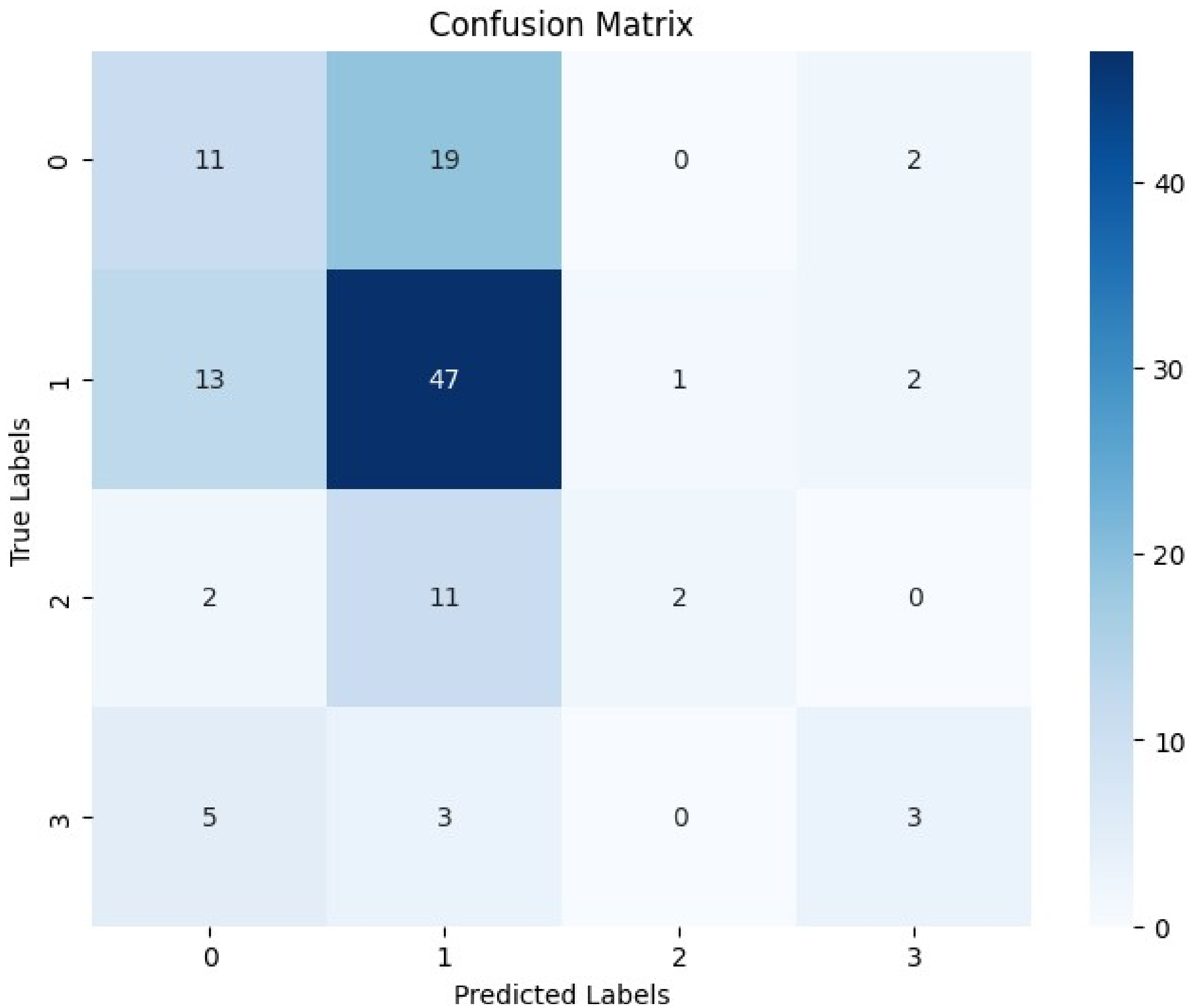
```
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid for hyperparameter tuning (adjust as
needed)
param_grid = {
```

```
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Create a Random Forest model
random_forest_model = RandomForestClassifier(random_state=42)

# Use GridSearchCV for hyperparameter tuning
grid_search_rf = GridSearchCV(random_forest_model, param_grid, cv=5,
scoring='accuracy')
grid_search_rf.fit(X_train_dense, y_train)

# Get the best parameters and retrain the model
best_params_rf = grid_search_rf.best_params_
best_random_forest_model = grid_search_rf.best_estimator_

# Print the best parameters
print("Best Parameters for Random Forest:", best_params_rf)

# Retrain the Random Forest model with the best parameters
best_random_forest_model.fit(X_train_dense, y_train)

# Predict the labels for the test set
y_pred_random_forest =
best_random_forest_model.predict(X_test_preprocessed)

# Evaluate the Random Forest model
accuracy_random_forest = accuracy_score(y_test, y_pred_random_forest)
conf_matrix_random_forest = confusion_matrix(y_test,
y_pred_random_forest)
classification_report_random_forest = classification_report(y_test,
y_pred_random_forest)

# Print the results for Random Forest
print(f"Random Forest Accuracy: {accuracy_random_forest:.4f}\n")
print("Confusion Matrix:")
print(conf_matrix_random_forest)
print("\nClassification Report:")
print(classification_report_random_forest)
```

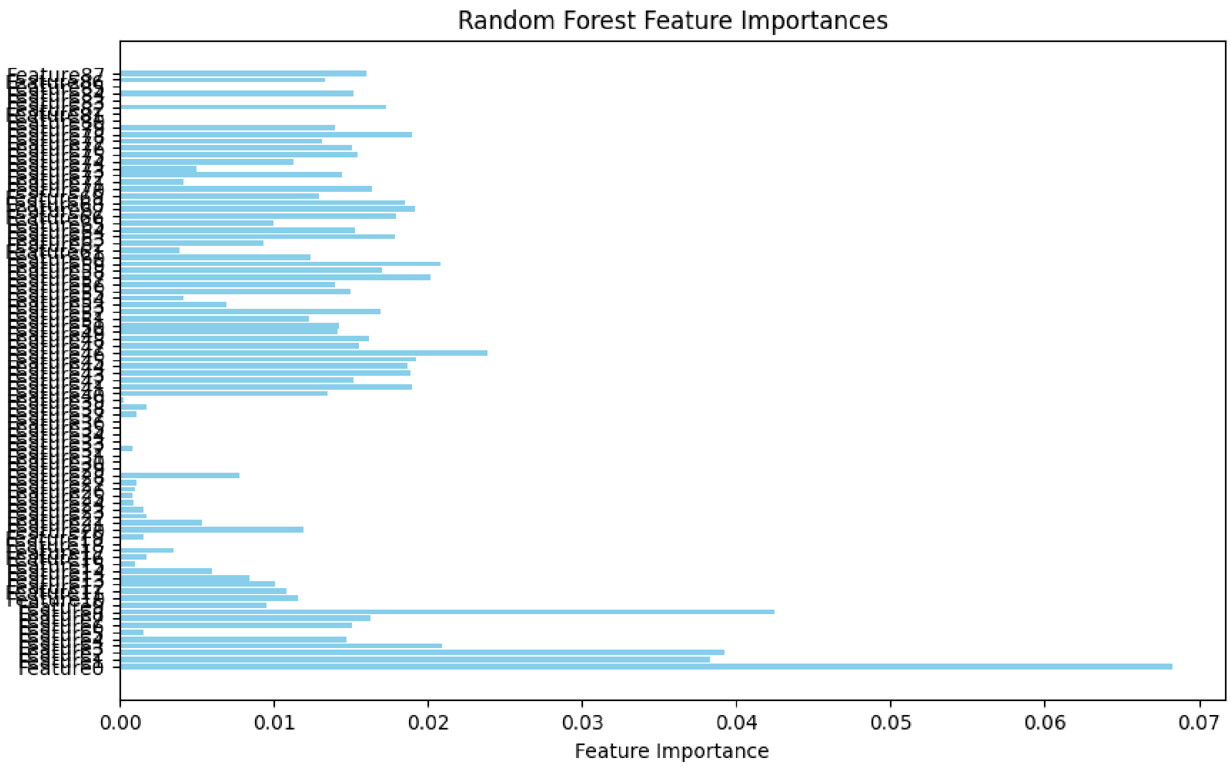
```
Best Parameters for Random Forest: {'bootstrap': True, 'max_depth':
None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators':
200}
Random Forest Accuracy: 0.5207

Confusion Matrix:
```

```
[[11 19  0  2]
 [13 47  1  2]
 [ 2 11  2  0]
 [ 5  3  0  3]]

Classification Report:
              precision    recall  f1-score   support

           0       0.35      0.34      0.35        32
           1       0.59      0.75      0.66        63
           2       0.67      0.13      0.22        15
           3       0.43      0.27      0.33        11

    accuracy                           0.52       121
   macro avg       0.51      0.37      0.39       121
weighted avg       0.52      0.52      0.49       121
```

```
import seaborn as sns

# Define a function to plot a confusion matrix
def plot_confusion_matrix(conf_matrix, class_names):
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()

# Plot the Confusion Matrix for Random Forest
plot_confusion_matrix(conf_matrix_random_forest, class_names)
```




```
# Get feature importances from the trained Random Forest model
feature_importances = best_random_forest_model.feature_importances_

# Create a bar plot to visualize feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_names, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importances')
plt.show()
```



7.5) XG BOOST.

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for XGBoost hyperparameter tuning
param_grid_xgb = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
}

# Create an XGBoost model
xgb_model = xgb.XGBClassifier(objective='multi:softmax',
num_class=len(class_names), random_state=42)

# Use GridSearchCV for XGBoost hyperparameter tuning
grid_search_xgb = GridSearchCV(xgb_model, param_grid_xgb, cv=5,
scoring='accuracy')
grid_search_xgb.fit(X_train_dense, y_train)

# Get the best parameters and retrain the model
```

```python
best_params_xgb = grid_search_xgb.best_params_
best_xgb_model = grid_search_xgb.best_estimator_

# Print the best parameters
print("Best Parameters for XGBoost:", best_params_xgb)

# Retrain the XGBoost model with the best parameters
best_xgb_model.fit(X_train_dense, y_train)

# Predict the labels for the test set
y_pred_xgb = best_xgb_model.predict(X_test_preprocessed)

# Evaluate the XGBoost model
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)
classification_report_xgb = classification_report(y_test, y_pred_xgb)

# Print the results for XGBoost
print(f"XGBoost Accuracy: {accuracy_xgb:.4f}\n")
print("Confusion Matrix:")
print(conf_matrix_xgb)
print("\nClassification Report:")
print(classification_report_xgb)
```

```
Best Parameters for XGBoost: {'colsample_bytree': 0.8,
'learning_rate': 0.01, 'max_depth': 5, 'min_child_weight': 3,
'n_estimators': 100, 'subsample': 0.8}
XGBoost Accuracy: 0.0826

Confusion Matrix:
[[ 0  0 17 15]
 [ 0  0 24 39]
 [ 0  0  3 12]
 [ 0  0  4  7]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        32
           1       0.00      0.00      0.00        63
           2       0.06      0.20      0.10        15
           3       0.10      0.64      0.17        11

    accuracy                           0.08       121
   macro avg       0.04      0.21      0.07       121
weighted avg       0.02      0.08      0.03       121
```

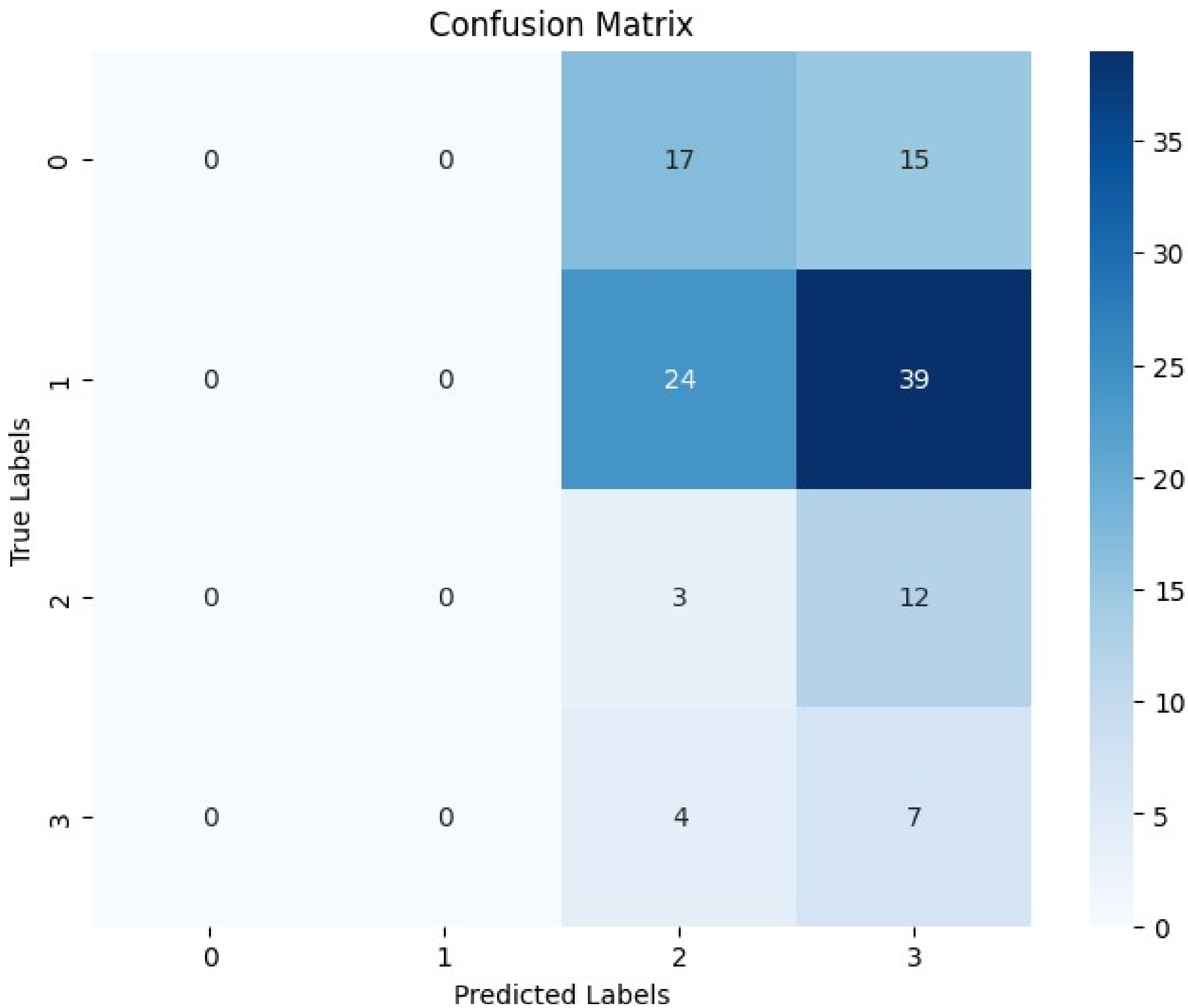
An accuracy of 8.26% suggests that there is room for improvement, and you may need to revisit the data, features, and model configuration to enhance performance.

8.) EVALUATION.

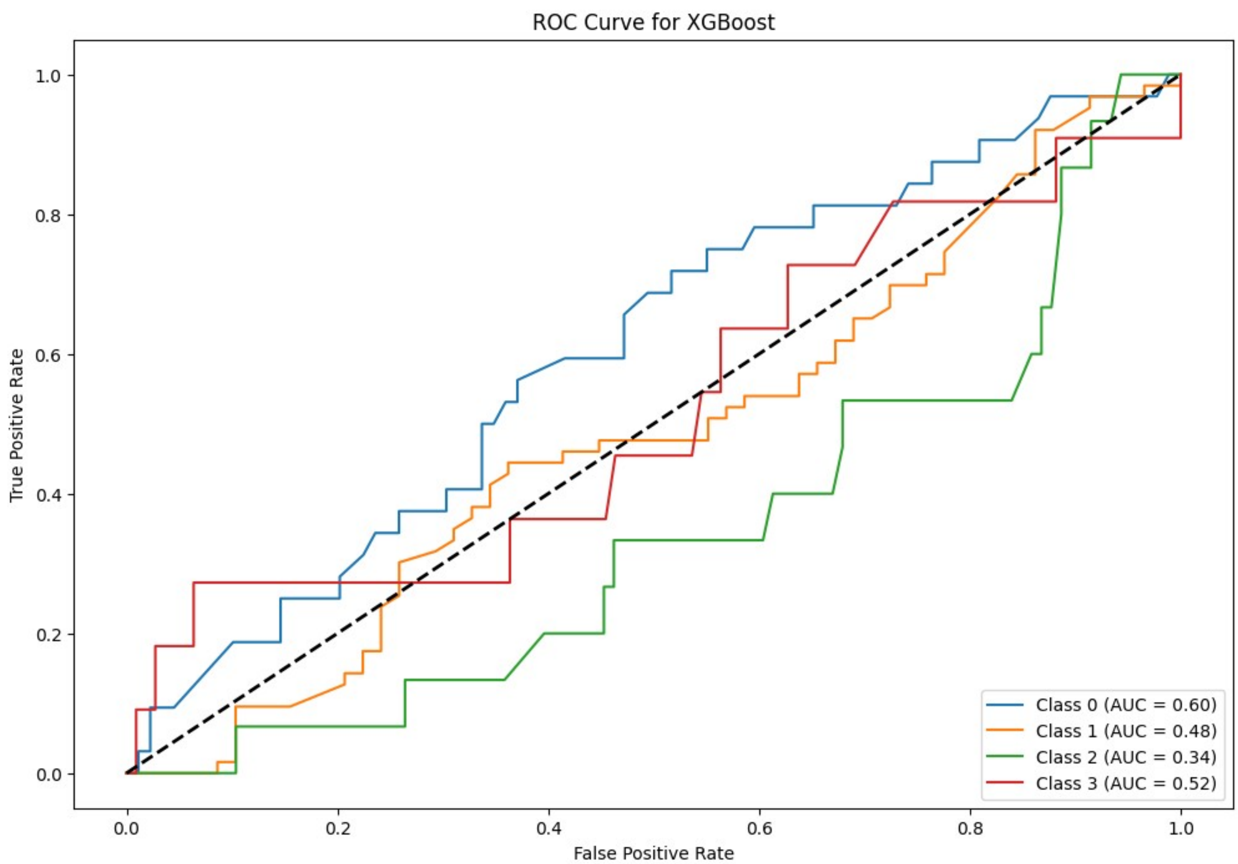
8.1) Evaluate XGBoost Model.

```
from sklearn.metrics import roc_curve, auc, precision_recall_curve,
average_precision_score

# Evaluate XGBoost model
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)
classification_report_xgb = classification_report(y_test, y_pred_xgb)

# Plot the Confusion Matrix
plot_confusion_matrix(conf_matrix_xgb, class_names)

# Calculate and plot ROC curve and AUC for each class
plt.figure(figsize=(12, 8))

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(class_names)):
    fpr[i], tpr[i], _ = roc_curve(y_test == i,
best_xgb_model.predict_proba(X_test_preprocessed)[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves
for i in range(len(class_names)):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC =
{roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', linewidth=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for XGBoost')
plt.legend(loc='lower right')
plt.show()

# Calculate and plot precision-recall curve and AUC for each class
plt.figure(figsize=(12, 8))

precision = dict()
recall = dict()
pr_auc = dict()

for i in range(len(class_names)):
    precision[i], recall[i], _ = precision_recall_curve(y_test == i,
```

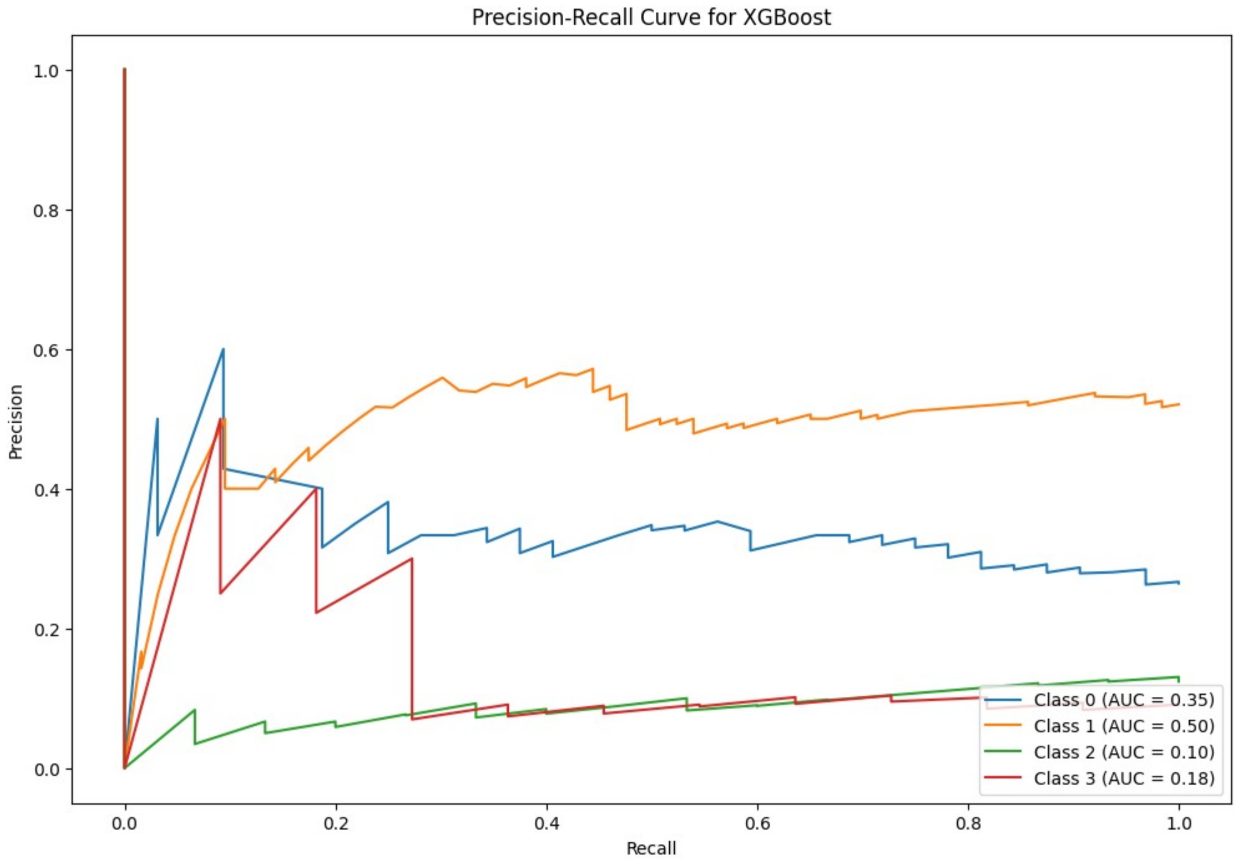
```
best_xgb_model.predict_proba(X_test_preprocessed)[:, i])
    pr_auc[i] = average_precision_score(y_test == i,
best_xgb_model.predict_proba(X_test_preprocessed)[:, i])

# Plot precision-recall curves
for i in range(len(class_names)):
    plt.plot(recall[i], precision[i], label=f'Class {i} (AUC =
{pr_auc[i]:.2f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for XGBoost')
plt.legend(loc='lower right')
plt.show()

# Print Classification Report and Accuracy
print(f"XGBoost Accuracy: {accuracy_xgb:.4f}\n")
print("Confusion Matrix:")
print(conf_matrix_xgb)
print("\nClassification Report:")
print(classification_report_xgb)
```



Confusion Matrix
True Labels
Predicted Labels
0
1
2
3
0
0
17
15
0
0
24
39
0
0
3
12
0
0
4
7
0
5
10
15
20
25
30
35



ROC Curve for XGBoost
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 (AUC = 0.60)
Class 1 (AUC = 0.48)
Class 2 (AUC = 0.34)
Class 3 (AUC = 0.52)



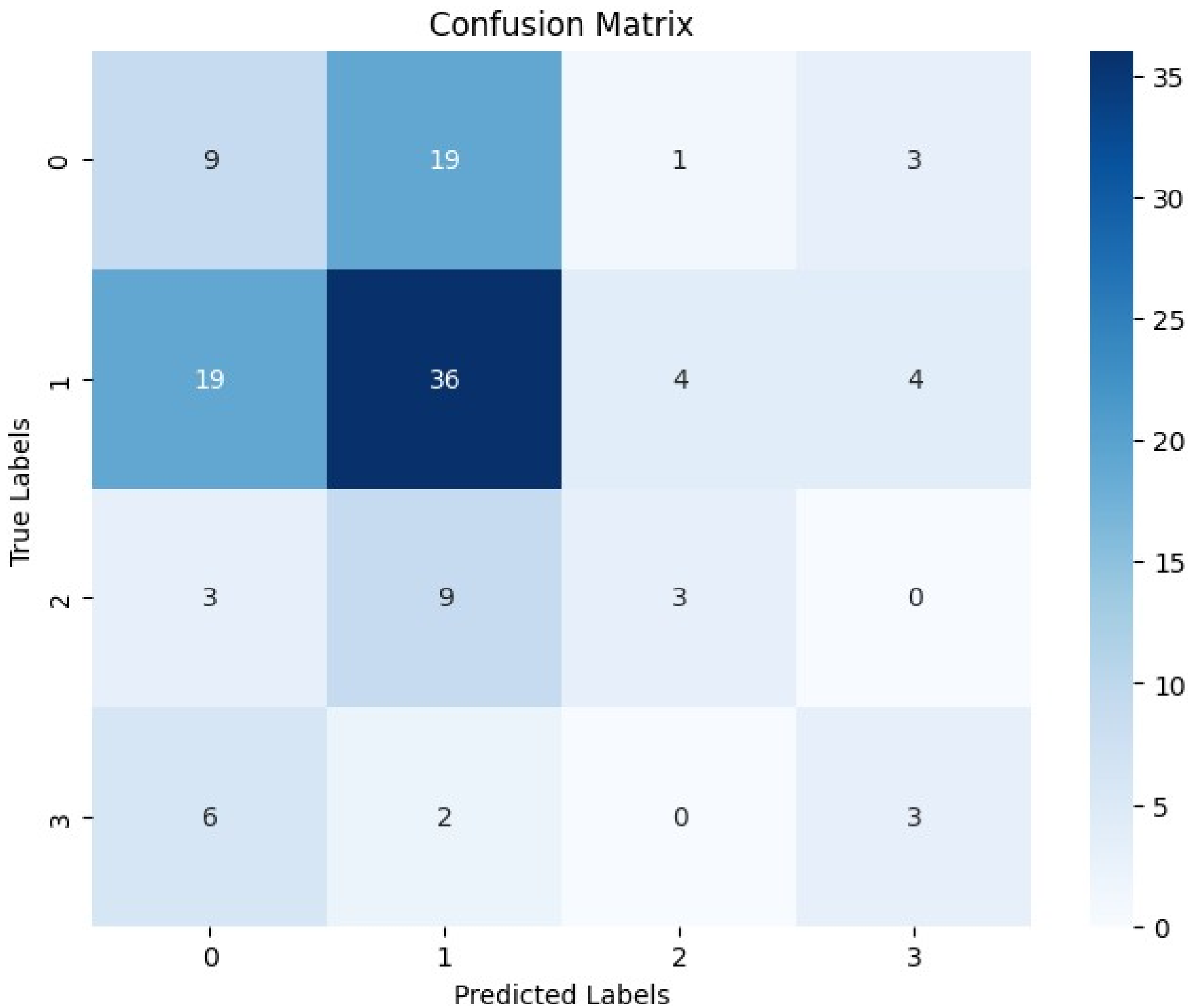

```
XGBoost Accuracy: 0.0826

Confusion Matrix:
[[ 0  0 17 15]
 [ 0  0 24 39]
 [ 0  0  3 12]
 [ 0  0  4  7]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        32
           1       0.00      0.00      0.00        63
           2       0.06      0.20      0.10        15
           3       0.10      0.64      0.17        11

    accuracy                           0.08       121
   macro avg       0.04      0.21      0.07       121
weighted avg       0.02      0.08      0.03       121
```

Evaluated XGBoost model using various metrics. consider metrics like precision, recall, and F1-score, along with the confusion matrix. Additionally, we'll create a ROC curve and calculate the AUC (Area Under the Curve) for a multiclass classification problem.

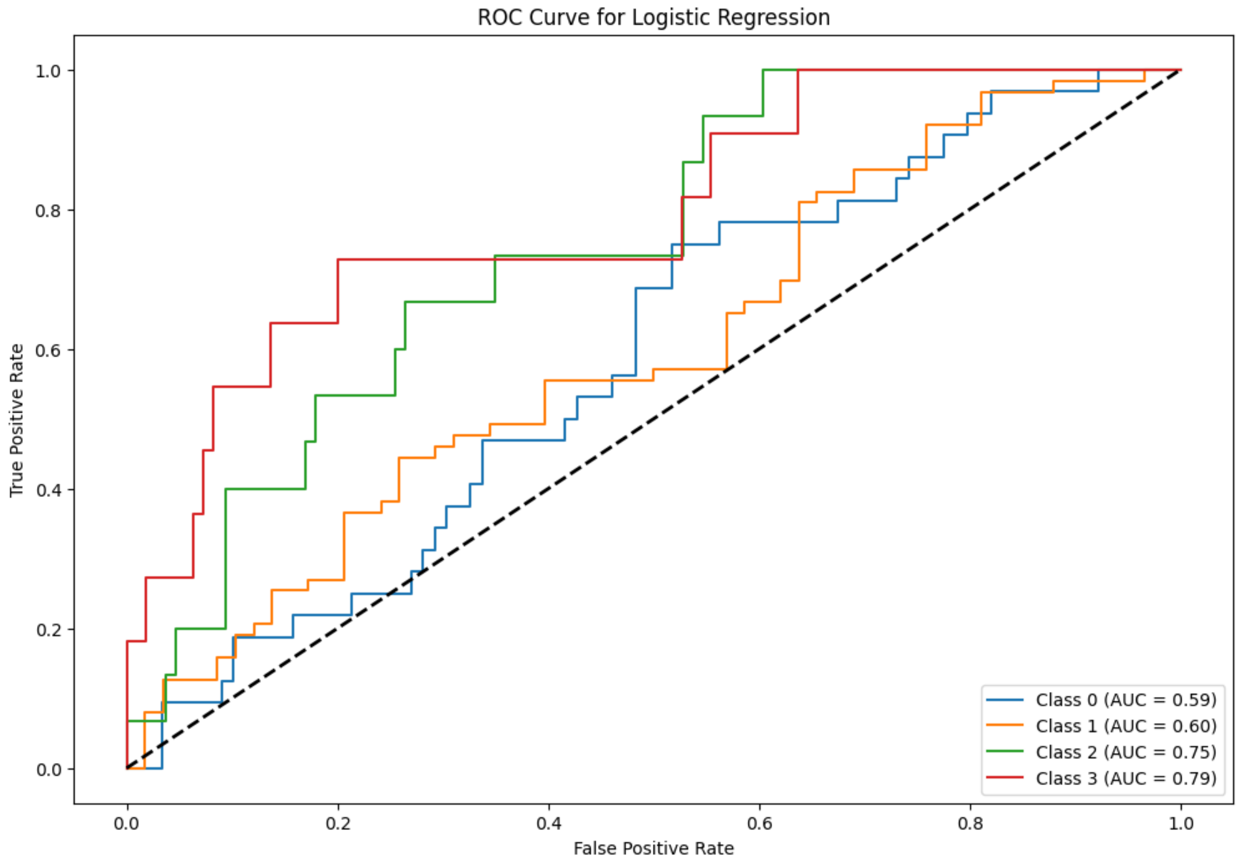
8.2) Evaluate Logistic Regression Model

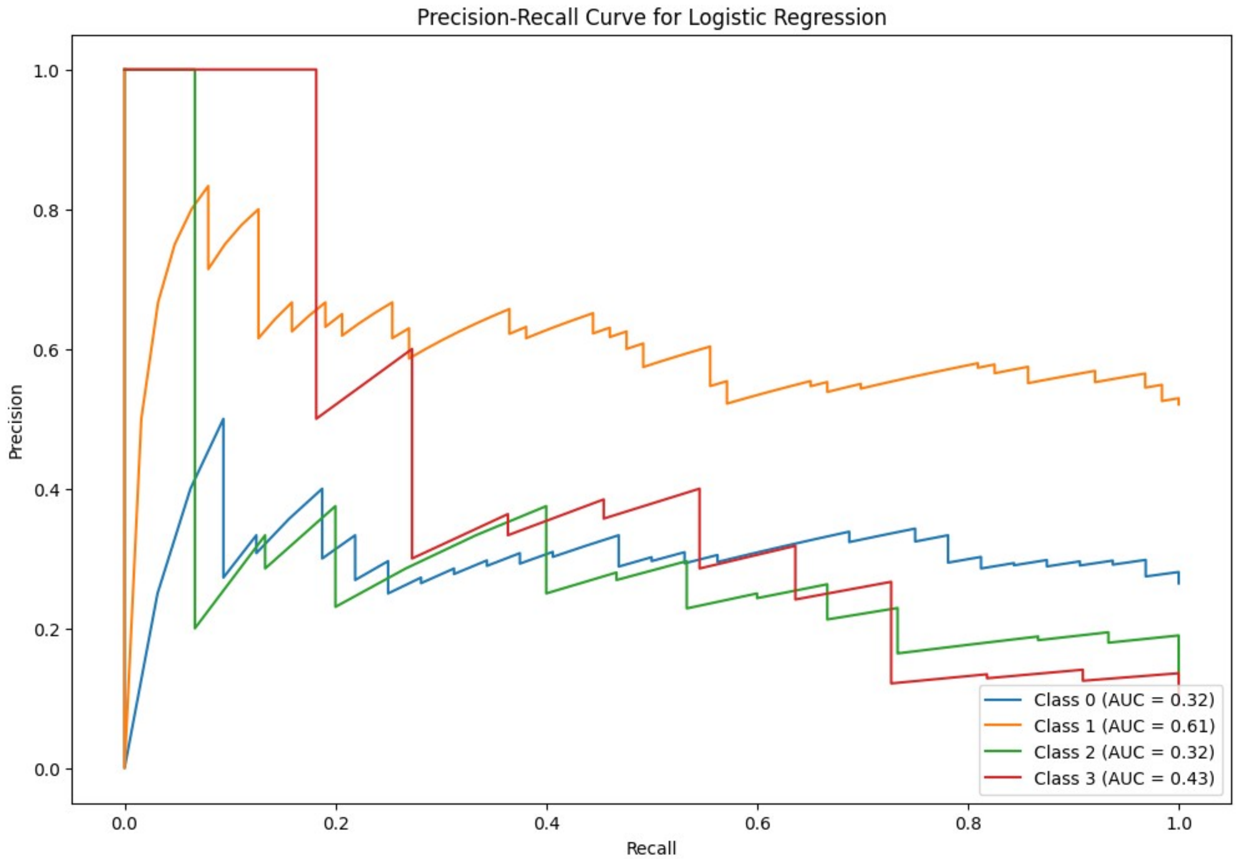
```
from sklearn.linear_model import LogisticRegression

# Create a Logistic Regression model
logistic_model = LogisticRegression(random_state=42)

# Train the model
logistic_model.fit(X_train_dense, y_train)

# Predict the labels for the test set
y_pred_logistic = logistic_model.predict(X_test_preprocessed)

# Evaluate Logistic Regression model
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
conf_matrix_logistic = confusion_matrix(y_test, y_pred_logistic)
classification_report_logistic = classification_report(y_test,
y_pred_logistic)



# Plot the Confusion Matrix
plot_confusion_matrix(conf_matrix_logistic, class_names)

# Calculate and plot ROC curve and AUC for each class
plt.figure(figsize=(12, 8))

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(class_names)):
    fpr[i], tpr[i], _ = roc_curve(y_test == i,
logistic_model.predict_proba(X_test_preprocessed)[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves
for i in range(len(class_names)):
    plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC =
{roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', linewidth=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.legend(loc='lower right')
plt.show()

# Calculate and plot precision-recall curve and AUC for each class
plt.figure(figsize=(12, 8))
```

```
precision = dict()
recall = dict()
pr_auc = dict()

for i in range(len(class_names)):
    precision[i], recall[i], _ = precision_recall_curve(y_test == i,
logistic_model.predict_proba(X_test_preprocessed)[:, i])
    pr_auc[i] = average_precision_score(y_test == i,
logistic_model.predict_proba(X_test_preprocessed)[:, i])

# Plot precision-recall curves
for i in range(len(class_names)):
    plt.plot(recall[i], precision[i], label=f'Class {i} (AUC =
{pr_auc[i]:.2f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Logistic Regression')
plt.legend(loc='lower right')
plt.show()

# Print Classification Report and Accuracy
print(f"Logistic Regression Accuracy: {accuracy_logistic:.4f}\n")
print("Confusion Matrix:")
print(conf_matrix_logistic)
print("\nClassification Report:")
print(classification_report_logistic)
```



Confusion Matrix
True Labels
Predicted Labels
0
1
2
3
9
19
1
3
19
36
4
4
3
9
3
0
6
2
0
3
35
30
25
20
15
10
5
0


ROC Curve for Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 (AUC = 0.59)
Class 1 (AUC = 0.60)
Class 2 (AUC = 0.75)
Class 3 (AUC = 0.79)



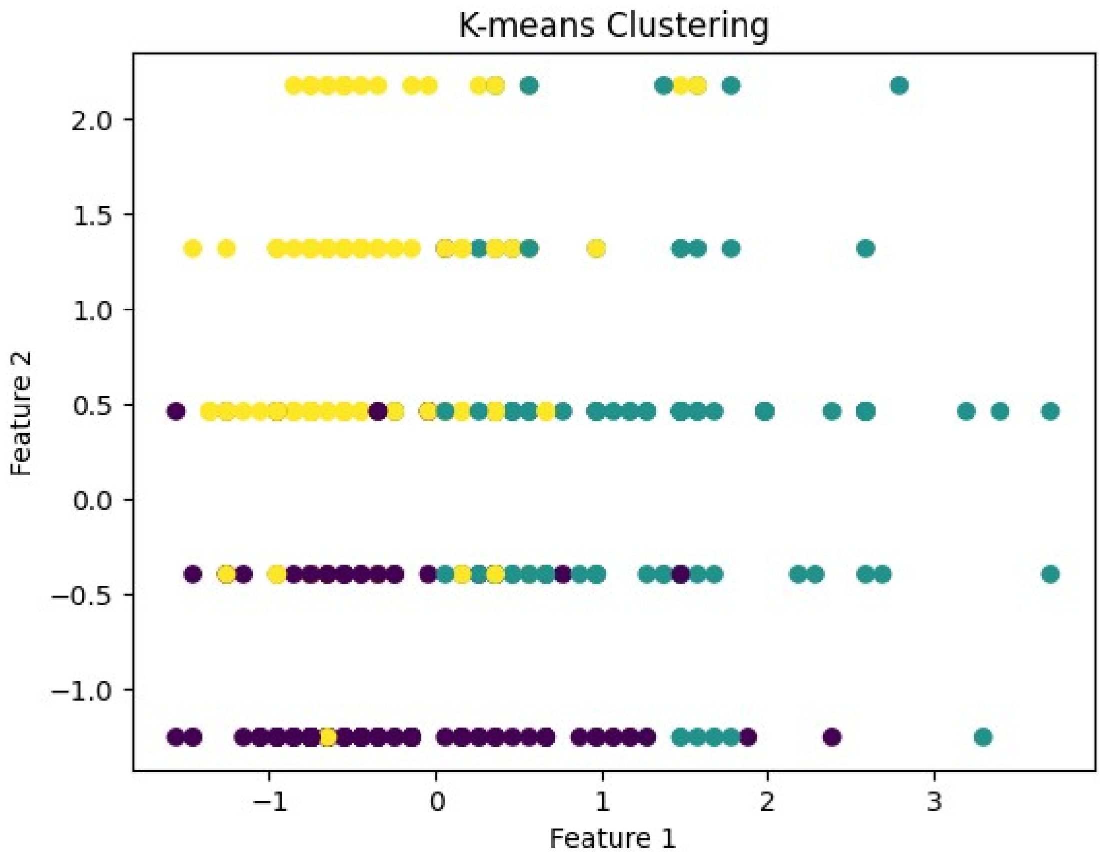

```
Logistic Regression Accuracy: 0.4215

Confusion Matrix:
[[ 9 19  1  3]
 [19 36  4  4]
 [ 3  9  3  0]
 [ 6  2  0  3]]

Classification Report:
              precision    recall  f1-score   support

           0       0.24      0.28      0.26        32
           1       0.55      0.57      0.56        63
           2       0.38      0.20      0.26        15
           3       0.30      0.27      0.29        11

    accuracy                           0.42       121
   macro avg       0.37      0.33      0.34       121
weighted avg       0.42      0.42      0.42       121
```

Logistic Regression Model:

The logistic regression model has an accuracy of 0.4215, providing a moderate level of performance. Precision, recall, and F1-score vary across classes, indicating potential challenges in certain categories. Consider exploring more sophisticated models or feature engineering to enhance predictive capabilities

8.3) Evaluate K-Means.

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Assuming X_train_preprocessed is the preprocessed training data
(sparse matrix)

# Convert sparse matrix to dense format
X_train_dense = X_train_preprocessed.toarray()

# Choose the number of clusters (K)
k = 3

# Apply K-means clustering
kmeans = KMeans(n_clusters=k, random_state=42)
clusters = kmeans.fit_predict(X_train_dense)

# Evaluate K-means clustering
silhouette_avg = silhouette_score(X_train_dense, clusters)
print(f"Silhouette Score for K-means: {silhouette_avg:.4f}")

# Visualize clusters (for the first two features)
plt.scatter(X_train_dense[:, 0], X_train_dense[:, 1], c=clusters,
cmap='viridis')
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

Silhouette Score for K-means: 0.0852
```



K-Means Clustering:

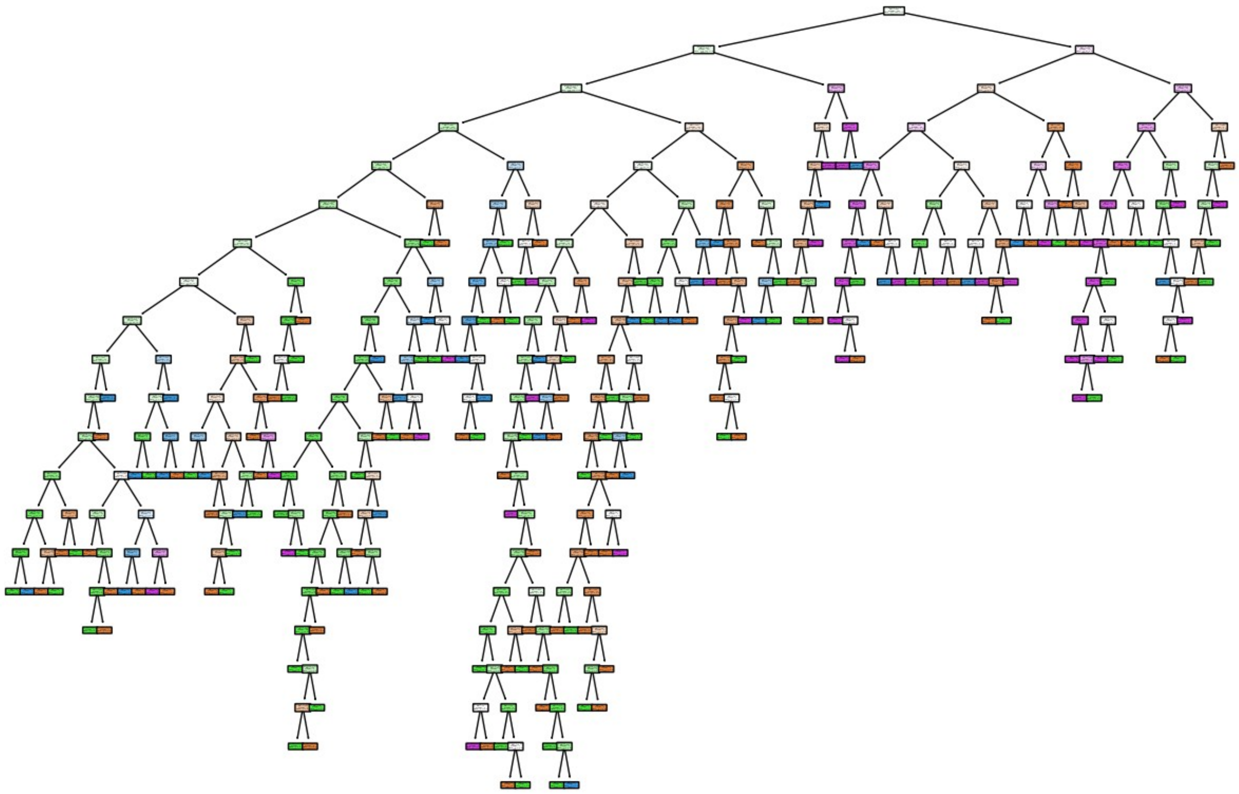
Clustering may not be suitable for the given task as it is an unsupervised method. Clustering is more appropriate for tasks where the data naturally groups into clusters, whereas classification models are designed for labeled data.

8.4) Evaluate Decision Trees Model

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Assuming X_train_preprocessed and y_train are the preprocessed
training data and labels
# Similarly, X_test_preprocessed and y_test for the test set

# Create a Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)

# Train the model
```

```
decision_tree_model.fit(X_train_preprocessed, y_train)

# Predict the labels for the test set
y_pred_decision_tree =
decision_tree_model.predict(X_test_preprocessed)

# Evaluate the model
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)
conf_matrix_decision_tree = confusion_matrix(y_test,
y_pred_decision_tree)
classification_report_decision_tree = classification_report(y_test,
y_pred_decision_tree)

# Print the results
print(f"Decision Tree Accuracy: {accuracy_decision_tree:.4f}\n")
print("Confusion Matrix:")
print(conf_matrix_decision_tree)
print("\nClassification Report:")
print(classification_report_decision_tree)

# Visualize the Decision Tree
plt.figure(figsize=(15, 10))
plot_tree(decision_tree_model, feature_names=feature_names,
class_names=class_names, filled=True, rounded=True)
plt.show()
```

```
Decision Tree Accuracy: 0.3802

Confusion Matrix:
[[10 13  3  6]
 [19 26 11  7]
 [ 4  8  3  0]
 [ 1  0  3  7]]

Classification Report:
              precision    recall  f1-score   support

           0       0.29      0.31      0.30        32
           1       0.55      0.41      0.47        63
           2       0.15      0.20      0.17        15
           3       0.35      0.64      0.45        11

    accuracy                           0.38       121
   macro avg       0.34      0.39      0.35       121
weighted avg       0.42      0.38      0.39       121
```



Decision Tree Model:

The decision tree model has a relatively low accuracy of 0.38, indicating that it may not perform well on the given dataset. The precision, recall, and F1-score for each class vary, suggesting that the model struggles with classifying certain categories. The decision tree visualization can provide insights into how the model makes decisions, but the complexity may affect its generalization.

9) DEPLOYMENT.

9.1) Save model using joblib.

```
#Save the model
import joblib
from sklearn.linear_model import LogisticRegression

# Example: train and save a Logistic Regression model
model = LogisticRegression(random_state=42)
# ... Train the model ...

# Save the model
joblib.dump(model, 'logistic_model.joblib')
```

```
['logistic_model.joblib']
```

9.2) Deploy using Streamlit

```
#Create a streamlit app for deployment
!pip install streamlit
```

```
Requirement already satisfied: streamlit in
/usr/local/lib/python3.10/dist-packages (1.29.0)
Requirement already satisfied: altair<6,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (4.2.2)
Requirement already satisfied: blinker<2,>=1.0.0 in
/usr/lib/python3/dist-packages (from streamlit) (1.4)
Requirement already satisfied: cachetools<6,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (5.3.2)
Requirement already satisfied: click<9,>=7.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (8.1.7)
Requirement already satisfied: importlib-metadata<7,>=1.4 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (6.8.0)
Requirement already satisfied: numpy<2,>=1.19.3 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (1.23.5)
Requirement already satisfied: packaging<24,>=16.8 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (23.2)
Requirement already satisfied: pandas<3,>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (1.5.3)
Requirement already satisfied: pillow<11,>=7.1.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (9.4.0)
Requirement already satisfied: protobuf<5,>=3.20 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (3.20.3)
Requirement already satisfied: pyarrow>=6.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (9.0.0)
Requirement already satisfied: python-dateutil<3,>=2.7.3 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (2.8.2)
Requirement already satisfied: requests<3,>=2.27 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (2.31.0)
Requirement already satisfied: rich<14,>=10.14.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (13.7.0)
Requirement already satisfied: tenacity<9,>=8.1.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (8.2.3)
Requirement already satisfied: toml<2,>=0.10.1 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.3.0 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (4.5.0)
Requirement already satisfied: tzlocal<6,>=1.1 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (5.2)
Requirement already satisfied: validators<1,>=0.2 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (0.22.0)
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in
```

```
/usr/local/lib/python3.10/dist-packages (from streamlit) (3.1.40)
Requirement already satisfied: pydeck<1,>=0.8.0b4 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (0.8.1b0)
Requirement already satisfied: tornado<7,>=6.0.3 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (6.3.2)
Requirement already satisfied: watchdog>=2.1.5 in
/usr/local/lib/python3.10/dist-packages (from streamlit) (3.0.0)
Requirement already satisfied: entrypoints in
/usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0-
>streamlit) (0.4)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0-
>streamlit) (3.1.2)
Requirement already satisfied: jsonschema>=3.0 in
/usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0-
>streamlit) (4.19.2)
Requirement already satisfied: toolz in
/usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0-
>streamlit) (0.12.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in
/usr/local/lib/python3.10/dist-packages (from gitpython!
=3.1.19,<4,>=3.0.7->streamlit) (4.0.11)
Requirement already satisfied: zipp>=0.5 in
/usr/local/lib/python3.10/dist-packages (from importlib-
metadata<7,>=1.4->streamlit) (3.17.0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas<3,>=1.3.0-
>streamlit) (2023.3.post1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-
dateutil<3,>=2.7.3->streamlit) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27-
>streamlit) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27-
>streamlit) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27-
>streamlit) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27-
>streamlit) (2023.11.17)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich<14,>=10.14.0-
>streamlit) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from rich<14,>=10.14.0-
>streamlit) (2.16.1)
```

```
Requirement already satisfied: smmap<6,>=3.0.1 in
/usr/local/lib/python3.10/dist-packages (from gitdb<5,>=4.0.1-
>gitpython!=3.1.19,<4,>=3.0.7->streamlit) (5.0.1)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->altair<6,>=4.0-
>streamlit) (2.1.3)
Requirement already satisfied: attrs>=22.2.0 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0-
>altair<6,>=4.0->streamlit) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0-
>altair<6,>=4.0->streamlit) (2023.11.2)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0-
>altair<6,>=4.0->streamlit) (0.31.1)
Requirement already satisfied: rpds-py>=0.7.1 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0-
>altair<6,>=4.0->streamlit) (0.13.2)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0-
>rich<14,>=10.14.0->streamlit) (0.1.2)
```

```python
#Create a Streamlit app in a file named app_streamlit.py:
import streamlit as st
import joblib
import numpy as np
from scipy import sparse


# Load the trained model
model = joblib.load('logistic_model.joblib')

def preprocess_input(data):
    # Convert the input data to a CSR matrix if needed
    input_csr = sparse.csr_matrix(data)  # Assuming data is a dense
array

    # Implement any additional preprocessing logic if required

    return input_csr


def main():
    st.title('Logistic Regression Predictor')

    # Collect user input features
    feature1 = st.slider('Feature 1', min_value=0.0, max_value=1.0,
value=0.5)
    feature2 = st.slider('Feature 2', min_value=0.0, max_value=1.0,
value=0.5)
```

```
    # Create a user input array
    user_input = np.array([[feature1, feature2]])

    # Preprocess the user input
    processed_input = preprocess_input(user_input)

    if st.button('Predict'):
        # Make predictions
        prediction = model.predict(processed_input)

        # Display the prediction
        st.success(f'Prediction: {prediction[0]}')

if __name__ == '__main__':
    main()

2023-12-05 09:45:05.997
  Warning: to view this Streamlit app on a browser, run it with the
following
  command:

    streamlit run
/usr/local/lib/python3.10/dist-packages/colab_kernel_launcher.py
[ARGUMENTS]
```

11) RECOMMENDATIONS.

Personalized Recommendations: Implement a robust recommendation engine based on user preferences and historical data to enhance the personalized shopping experience.

User Interface Optimization: Improve the user interface based on insights gained from the user journey analysis, making navigation more intuitive and user-friendly.

Targeted Marketing Campaigns: Develop targeted marketing campaigns tailored to specific customer segments, optimizing advertising efforts and increasing ROI.

Dynamic Pricing Strategies: Explore dynamic pricing models based on customer behavior, demand patterns, and competitor analysis.

Customer Engagement Initiatives: Introduce loyalty programs, special promotions, and interactive content to enhance customer engagement and retention.

10) CONCLUSION

In conclusion, this project provides a comprehensive understanding of customer behavior on e-commerce platforms, offering valuable insights that can drive strategic decision-making. By implementing the recommended strategies, businesses can not only enhance the user experience but also increase customer satisfaction, loyalty, and ultimately, revenue. As the e-commerce landscape continues to evolve, staying attuned to customer behavior is pivotal for maintaining a competitive edge in the market.