
- title: Customer Personality Analysis
- author: KAINAT NAQVI
- date: 2024-12-28
- description: This project aims to analyze customer data to understand their personalities, preferences, and behaviors.
- tags: [Customer Analysis, Machine Learning, Customer Segmentation, Marketing]

Customer Personality Analysis

Introduction

In today's competitive business landscape, understanding customer behavior is paramount for success. This project delves into the realm of **Customer Personality Analysis**, a crucial endeavor that seeks to unravel the intricate tapestry of customer preferences, needs, and behaviors. By analyzing customer data, we aim to gain deeper insights into their personalities, enabling businesses to tailor their marketing strategies, product offerings, and customer service to better resonate with their target audience.

Objectives

This project aims to achieve the following objectives:

1. **Understand Customer Demographics and Behavior:** Analyze customer data to identify key demographic trends, spending patterns, and purchase history.
2. **Segment Customers:** Group customers into distinct segments based on their characteristics and behaviors, enabling targeted marketing efforts.
3. **Predict Customer Churn:** Develop predictive models to identify customers at risk of churning and implement proactive retention strategies.
4. **Optimize Marketing Campaigns:** Analyze campaign performance and identify the most effective channels and messages for different customer segments.

Methodology

This project will employ a combination of data analysis and machine learning techniques, including:

- **Data Cleaning and Preparation:** Handling missing values, data type conversions, and feature engineering.
- **Exploratory Data Analysis (EDA):** Visualizing data distributions, identifying trends, and calculating descriptive statistics.
- **Customer Segmentation:** Utilizing clustering algorithms to group customers with similar characteristics.
- **Predictive Modeling:** Building machine learning models to predict customer churn and campaign response.

Data

- **Dataset:**
  https://raw.githubusercontent.com/amankharwal/Website-data/master/marketing_campaign.csv
- **Variables:**
  - `ID`: Unique customer identifier
  - `Year_Birth`: Year of birth
  - `Education`: Educational level
  - `Marital_Status`: Marital status
  - `Income`: Annual income
  - `Kidhome`: Number of children at home
  - `Teenhome`: Number of teenagers at home
  - `Dt_Customer`: Date of customer enrollment
  - `Recency`: Number of days since last purchase
  - `MntWines`: Amount spent on wine
  - `MntFruits`: Amount spent on fruits
  - `MntMeatProducts`: Amount spent on meat products
  - `MntFishProducts`: Amount spent on fish products
  - `MntSweetProducts`: Amount spent on sweets
  - `MntGoldProds`: Amount spent on gold products
  - `NumDealsPurchases`: Number of purchases made with a discount
  - `NumWebPurchases`: Number of purchases made online
  - `NumCatalogPurchases`: Number of purchases made through catalog
  - `NumStorePurchases`: Number of purchases made in-store
  - `NumWebVisitsMonth`: Number of visits to the website per month
  - `AcceptedCmp3`: Whether the customer accepted campaign 3
  - `AcceptedCmp4`: Whether the customer accepted campaign 4
  - `AcceptedCmp5`: Whether the customer accepted campaign 5
  - `AcceptedCmp1`: Whether the customer accepted campaign 1
  - `AcceptedCmp2`: Whether the customer accepted campaign 2
  - `Complain`: Whether the customer has filed a complaint
  - `Z_CostContact`: Cost of contact
  - `Z_Revenue`: Revenue generated by the customer
  - `Response`: Whether the customer responded to the last campaign

Expected Outcomes

This project is expected to deliver the following outcomes:

- A comprehensive understanding of customer personalities and preferences.
- Actionable insights for targeted marketing campaigns and customer retention strategies.
- Predictive models to identify at-risk customers and optimize marketing spend.

```
import pandas as pd

url =
"https://raw.githubusercontent.com/amankharwal/Website-data/master/
marketing_campaign.csv"
data = pd.read_csv(url, sep=';')

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

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\869169758.py:1:
DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major
release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type,
and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at
https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

```
     ID  Year_Birth   Education Marital_Status   Income  Kidhome
Teenhome  \
0  5524        1957  Graduation         Single  58138.0        0
0
1  2174        1954  Graduation         Single  46344.0        1
1
2  4141        1965  Graduation       Together  71613.0        0
0
3  6182        1984  Graduation       Together  26646.0        1
0
4  5324        1981         PhD        Married  58293.0        1
0

  Dt_Customer  Recency  MntWines  ...  NumWebVisitsMonth  AcceptedCmp3
\
0  2012-09-04       58       635  ...                  7             0

1  2014-03-08       38        11  ...                  5             0

2  2013-08-21       26       426  ...                  4             0

3  2014-02-10       26        11  ...                  6             0

4  2014-01-19       94       173  ...                  5             0
```

```
   AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  \
0             0             0             0             0         0
1             0             0             0             0         0
2             0             0             0             0         0
3             0             0             0             0         0
4             0             0             0             0         0

   Z_CostContact  Z_Revenue  Response
0              3         11         1
1              3         11         0
2              3         11         0
3              3         11         0
4              3         11         0

[5 rows x 29 columns]
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
 19  NumWebVisitsMonth    2240 non-null   int64  
 20  AcceptedCmp3         2240 non-null   int64  
 21  AcceptedCmp4         2240 non-null   int64  
 22  AcceptedCmp5         2240 non-null   int64  
 23  AcceptedCmp1         2240 non-null   int64  
 24  AcceptedCmp2         2240 non-null   int64  
 25  Complain             2240 non-null   int64  
 26  Z_CostContact        2240 non-null   int64  
```

```
 27  Z_Revenue            2240 non-null   int64
 28  Response             2240 non-null   int64
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

```
data.describe()
```

```
                ID   Year_Birth         Income      Kidhome
Teenhome  \
count   2240.000000  2240.000000    2216.000000  2240.000000
2240.000000
mean    5592.159821  1968.805804   52247.251354     0.444196
0.506250
std     3246.662198    11.984069   25173.076661     0.538398
0.544538
min        0.000000  1893.000000    1730.000000     0.000000
0.000000
25%     2828.250000  1959.000000   35303.000000     0.000000
0.000000
50%     5458.500000  1970.000000   51381.500000     0.000000
0.000000
75%     8427.750000  1977.000000   68522.000000     1.000000
1.000000
max    11191.000000  1996.000000  666666.000000     2.000000
2.000000

           Recency     MntWines    MntFruits  MntMeatProducts  \
count  2240.000000  2240.000000  2240.000000      2240.000000
mean     49.109375   303.935714    26.302232       166.950000
std      28.962453   336.597393    39.773434       225.715373
min       0.000000     0.000000     0.000000         0.000000
25%      24.000000    23.750000     1.000000        16.000000
50%      49.000000   173.500000     8.000000        67.000000
75%      74.000000   504.250000    33.000000       232.000000
max      99.000000  1493.000000   199.000000      1725.000000

       MntFishProducts  ...  NumWebVisitsMonth  AcceptedCmp3
AcceptedCmp4  \
count      2240.000000  ...        2240.000000   2240.000000
2240.000000
mean         37.525446  ...           5.316518      0.072768
0.074554
std          54.628979  ...           2.426645      0.259813
0.262728
min           0.000000  ...           0.000000      0.000000
0.000000
25%           3.000000  ...           3.000000      0.000000
0.000000
50%          12.000000  ...           6.000000      0.000000
0.000000
```

```
75%          50.000000  ...            7.000000      0.000000
0.000000
max         259.000000  ...           20.000000      1.000000
1.000000

       AcceptedCmp5  AcceptedCmp1  AcceptedCmp2     Complain
Z_CostContact  \
count   2240.000000   2240.000000   2240.000000  2240.000000
2240.0
mean       0.072768      0.064286      0.013393     0.009375
3.0
std        0.259813      0.245316      0.114976     0.096391
0.0
min        0.000000      0.000000      0.000000     0.000000
3.0
25%        0.000000      0.000000      0.000000     0.000000
3.0
50%        0.000000      0.000000      0.000000     0.000000
3.0
75%        0.000000      0.000000      0.000000     0.000000
3.0
max        1.000000      1.000000      1.000000     1.000000
3.0

       Z_Revenue     Response
count     2240.0  2240.000000
mean        11.0     0.149107
std          0.0     0.356274
min         11.0     0.000000
25%         11.0     0.000000
50%         11.0     0.000000
75%         11.0     0.000000
max         11.0     1.000000

[8 rows x 26 columns]
```

Key Observations:

- Shape: The dataset contains 2240 rows and 29 columns.

Missing Values:

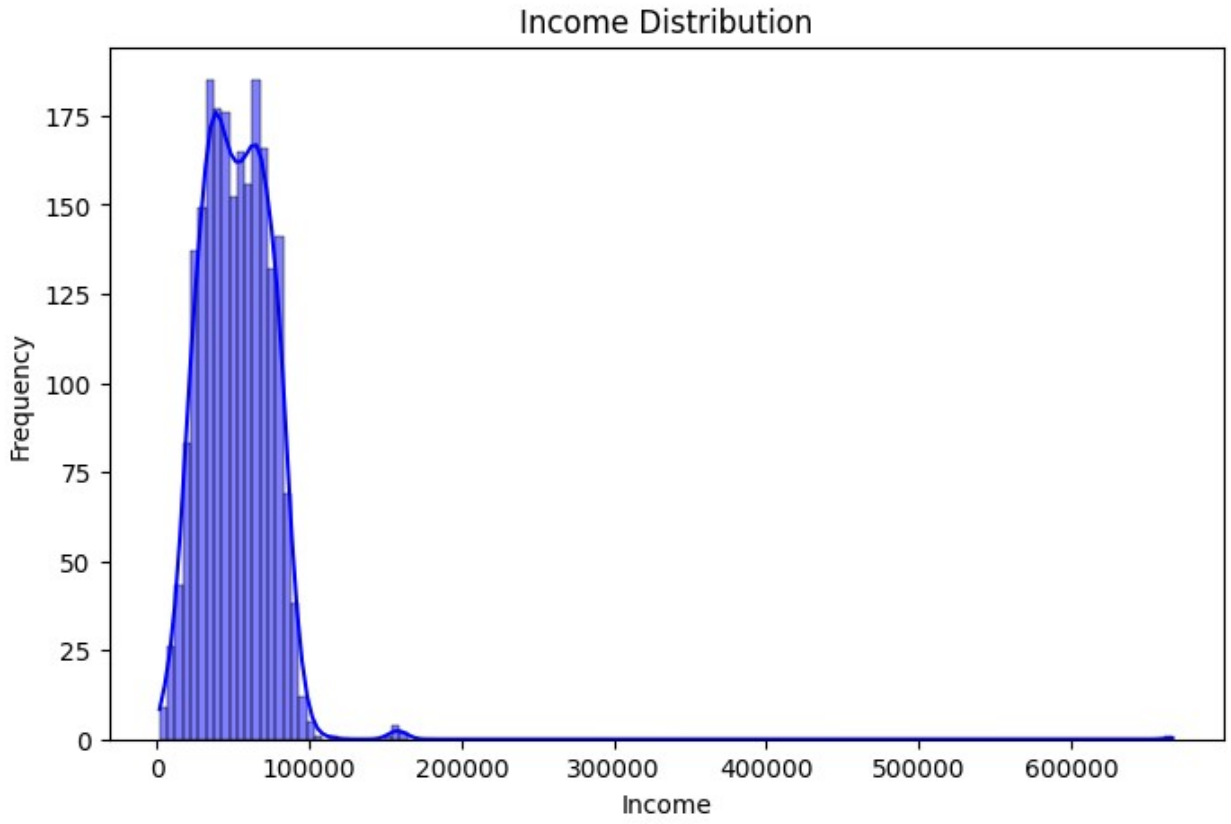
- The `Income` column has 24 missing values.
- No other columns have missing data.

Data Types:

- 3 categorical columns: `Education`, `Marital_Status`, `Dt_Customer`.
- 25 numerical columns: Mostly integers, with `Income` being a float.

```
# Check distribution of Income to decide handling method
import matplotlib.pyplot as plt
```

```
import seaborn as sns

plt.figure(figsize=(8, 5))
sns.histplot(data['Income'], kde=True, color='blue')
plt.title('Income Distribution')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.show()
```



Income Distribution Insights

- **Right-skewed distribution:** Majority of customers have lower incomes, with a smaller portion having very high incomes.
- **Peak:** Around 60,000 to 80,000, indicating the most common income bracket.
- **Wide spread:** Significant income variation across customers.

Implications:

- **Target marketing:** Focus on lower to moderate income segments, but also consider the potential of high-income customers.
- **Product development:** Offer tiered products to cater to different income levels.
- **Customer segmentation:** Use income as a key variable for segmentation.

Further analysis: Investigate potential outliers and segment customers based on income and other attributes.

Conclusion:

For this dataset, imputing with the median is the best choice as it ensures no significant data loss while handling skewness effectively.

Handling Missing Values :

```
# Replace missing values with median
data['Income'].fillna(data['Income'].median(), inplace=True)

# Verify if missing values are handled
print("Missing values in Income after handling:",
data['Income'].isnull().sum())
```

```
Missing values in Income after handling: 0
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\3357389572.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['Income'].fillna(data['Income'].median(), inplace=True)
```

```
# Convert Dt_Customer to datetime format
data['Dt_Customer'] = pd.to_datetime(data['Dt_Customer'], format='%Y-
%m-%d')

# Create a new feature: Customer_Since
data['Customer_Since'] = (pd.Timestamp.now() -
data['Dt_Customer']).dt.days

# Display the changes
print(data[['Dt_Customer', 'Customer_Since']].head())
```

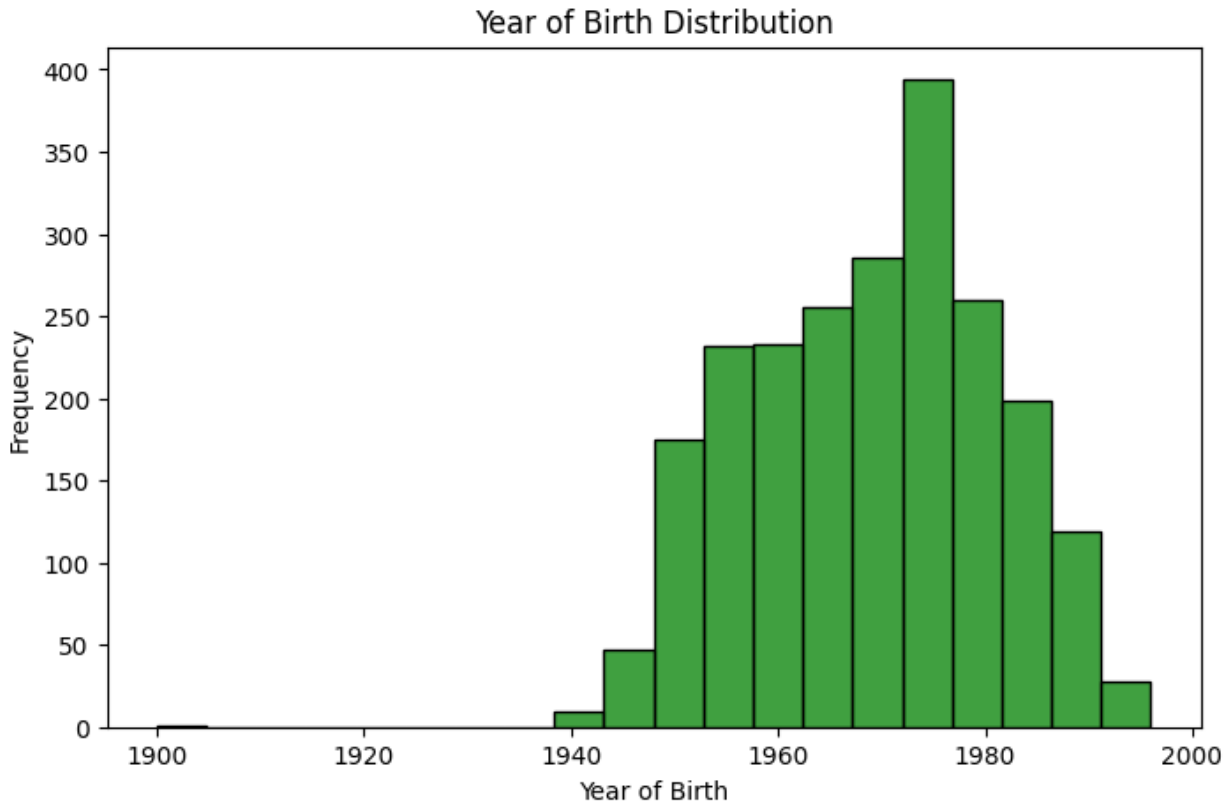
```
  Dt_Customer  Customer_Since
0  2012-09-04            4498
1  2014-03-08            3948
2  2013-08-21            4147
3  2014-02-10            3974
4  2014-01-19            3996
```

The Dt_Customer column represents the date when the customer enrolled. We need to:

Convert it to a datetime format for easier analysis. Create a new feature, such as Customer_Since (number of days since enrollment).

```
# Check for outliers in Year_Birth
print("Unique Year_Birth values:", data['Year_Birth'].unique())

# Plot distribution
plt.figure(figsize=(8, 5))
sns.histplot(data['Year_Birth'], bins=20, kde=False, color='green')
plt.title('Year of Birth Distribution')
plt.xlabel('Year of Birth')
plt.ylabel('Frequency')
plt.show()

# Filter out unrealistic birth years (e.g., before 1900 or after 2020)
data = data[(data['Year_Birth'] >= 1900) & (data['Year_Birth'] <=
2020)]

# Verify changes
print("Dataset shape after Year_Birth filtering:", data.shape)
```

```
Unique Year_Birth values: [1957 1954 1965 1984 1981 1967 1971 1985
1974 1950 1983 1976 1959 1952
 1987 1946 1980 1949 1982 1979 1951 1969 1986 1989 1963 1970 1973 1943
 1975 1996 1968 1964 1977 1978 1955 1966 1988 1948 1958 1972 1960 1945
 1991 1962 1953 1961 1956 1992 1900 1990 1947 1993 1994 1941 1944 1995
 1940]
```




```
Dataset shape after Year_Birth filtering: (2238, 30)
```

The dataset appears to have been successfully cleaned for the Year_Birth column. You've removed outliers like 1893, 1899, and other unrealistic birth years. We will now handle categorical columns such as Education and Marital_Status using one-hot encoding or label encoding and address the rest of the numerical columns for scaling or transformation.

```
# Check unique values in 'Education' and 'Marital_Status'
print("Unique values in Education:", data['Education'].unique())
print("Unique values in Marital_Status:",
data['Marital_Status'].unique())

# Simplify and clean up 'Education' categories
data['Education'] = data['Education'].replace({
    '2n Cycle': 'Other',
    'Basic': 'Other',
    'Master': 'Postgraduate',
    'PhD': 'Postgraduate',
    'Graduation': 'Undergraduate'
})

# Simplify and clean up 'Marital_Status' categories
data['Marital_Status'] = data['Marital_Status'].replace({
    'Together': 'Married',
```

```
    'Alone': 'Single',
    'Absurd': 'Other',
    'YOLO': 'Other'
})

# Perform one-hot encoding on 'Education' and 'Marital_Status'
data_encoded = pd.get_dummies(data, columns=['Education',
'Marital_Status'], drop_first=True)

# Display the shape of the dataset after encoding
print("Dataset shape after encoding:", data_encoded.shape)

# Preview encoded columns
encoded_columns = [col for col in data_encoded.columns if 'Education_'
in col or 'Marital_Status_' in col]
print(data_encoded[encoded_columns].head())
```

```
Unique values in Education: ['Undergraduate' 'Postgraduate' 'Other']
Unique values in Marital_Status: ['Single' 'Married' 'Divorced'
'Widow' 'Other']
Dataset shape after encoding: (2238, 34)
   Education_Postgraduate  Education_Undergraduate
Marital_Status_Married  \
0                   False                     True
False
1                   False                     True
False
2                   False                     True
True
3                   False                     True
True
4                    True                    False
True

   Marital_Status_Other  Marital_Status_Single  Marital_Status_Widow
0                 False                   True                 False
1                 False                   True                 False
2                 False                  False                 False
3                 False                  False                 False
4                 False                  False                 False
```

```
# Drop irrelevant columns
data_encoded = data_encoded.drop(columns=['Z_CostContact',
'Z_Revenue'], errors='ignore')

# Check the shape of the dataset after dropping columns
print(f"Dataset shape after dropping irrelevant columns:
{data_encoded.shape}")
```

```
Dataset shape after dropping irrelevant columns: (2238, 32)
```

```
# Create derived columns in data_encoded
data_encoded['MntTotal'] = (data_encoded['MntWines'] +
data_encoded['MntFruits'] + data_encoded['MntMeatProducts'] +
                            data_encoded['MntFishProducts'] +
data_encoded['MntSweetProducts'] + data_encoded['MntGoldProds'])


data_encoded['AvgSpendingPerVisit'] = data_encoded['MntTotal'] / (
    data_encoded['NumWebPurchases'] +
data_encoded['NumCatalogPurchases'] +
data_encoded['NumStorePurchases']
).replace(0, 1)  # Avoid division by zero

data_encoded['Customer_Age'] = pd.Timestamp.now().year -
data_encoded['Year_Birth']

# Display the new derived columns
print(data_encoded[['MntTotal', 'AvgSpendingPerVisit', 'Customer_Age',
'Customer_Since']].head())
```

```
   MntTotal  AvgSpendingPerVisit  Customer_Age  Customer_Since
0      1617            73.500000            67            4498
1        27             6.750000            70            3948
2       776            38.800000            59            4147
3        53             8.833333            40            3974
4       422            30.142857            43            3996
```

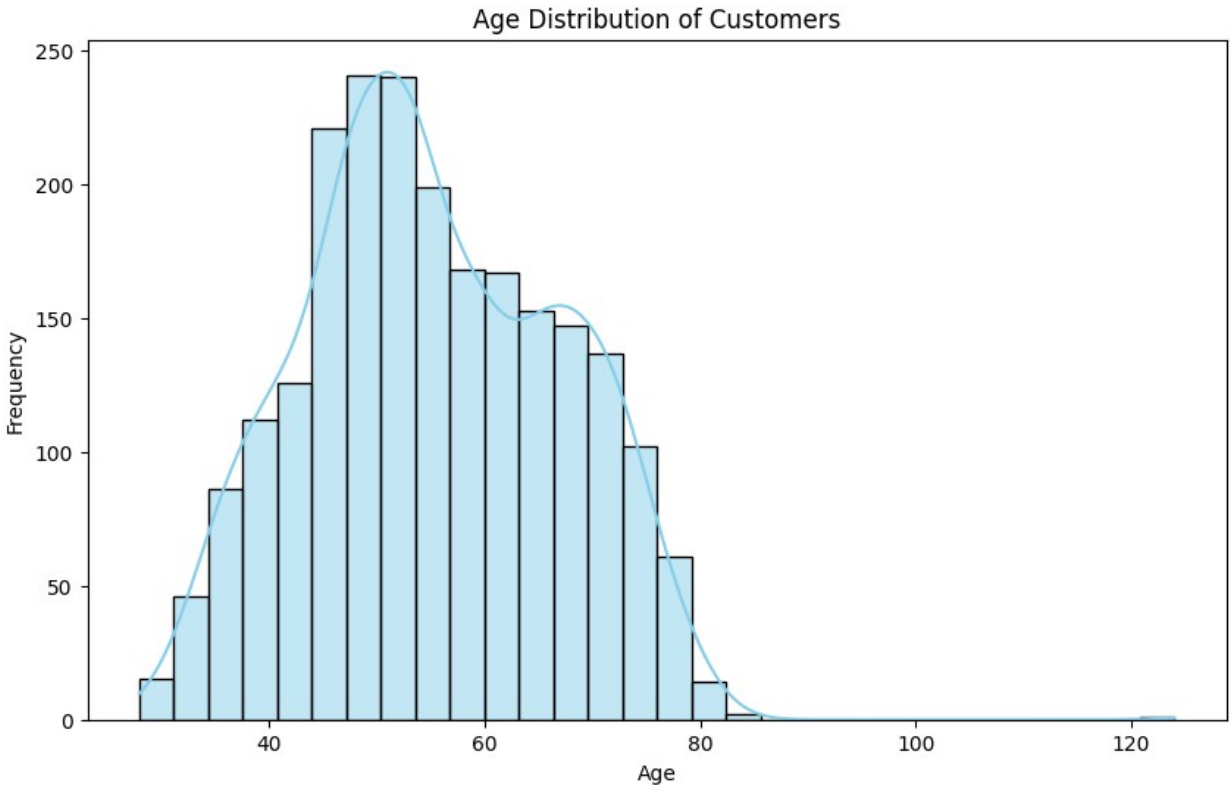
```
data_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2238 entries, 0 to 2239
Data columns (total 35 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   ID                      2238 non-null   int64
 1   Year_Birth              2238 non-null   int64
 2   Income                  2238 non-null   float64
 3   Kidhome                 2238 non-null   int64
 4   Teenhome                2238 non-null   int64
 5   Dt_Customer             2238 non-null   datetime64[ns]
 6   Recency                 2238 non-null   int64
 7   MntWines                2238 non-null   int64
 8   MntFruits               2238 non-null   int64
 9   MntMeatProducts         2238 non-null   int64
 10  MntFishProducts         2238 non-null   int64
 11  MntSweetProducts        2238 non-null   int64
 12  MntGoldProds            2238 non-null   int64
 13  NumDealsPurchases       2238 non-null   int64
 14  NumWebPurchases         2238 non-null   int64
```

```
 15  NumCatalogPurchases      2238 non-null   int64
 16  NumStorePurchases        2238 non-null   int64
 17  NumWebVisitsMonth        2238 non-null   int64
 18  AcceptedCmp3             2238 non-null   int64
 19  AcceptedCmp4             2238 non-null   int64
 20  AcceptedCmp5             2238 non-null   int64
 21  AcceptedCmp1             2238 non-null   int64
 22  AcceptedCmp2             2238 non-null   int64
 23  Complain                 2238 non-null   int64
 24  Response                 2238 non-null   int64
 25  Customer_Since           2238 non-null   int64
 26  Education_Postgraduate   2238 non-null   bool
 27  Education_Undergraduate  2238 non-null   bool
 28  Marital_Status_Married   2238 non-null   bool
 29  Marital_Status_Other     2238 non-null   bool
 30  Marital_Status_Single    2238 non-null   bool
 31  Marital_Status_Widow     2238 non-null   bool
 32  MntTotal                 2238 non-null   int64
 33  AvgSpendingPerVisit      2238 non-null   float64
 34  Customer_Age             2238 non-null   int64
dtypes: bool(6), datetime64[ns](1), float64(2), int64(26)
memory usage: 537.6 KB
```

Exploratory Data Analysis (EDA)

```
# Age Distribution
plt.figure(figsize=(10, 6))
sns.histplot(data_encoded['Customer_Age'], bins=30, kde=True,
color='skyblue')
plt.title('Age Distribution of Customers')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



Insights:

- **Right-skewed distribution:** The age distribution is positively skewed, indicating that a majority of customers are younger, with a smaller portion being older.
- **Peak:** The peak of the distribution is around 40-50 years old, suggesting this is the age group with the highest concentration of customers.
- **Long tail:** The distribution has a long tail, indicating the presence of a significant number of customers in the older age groups.

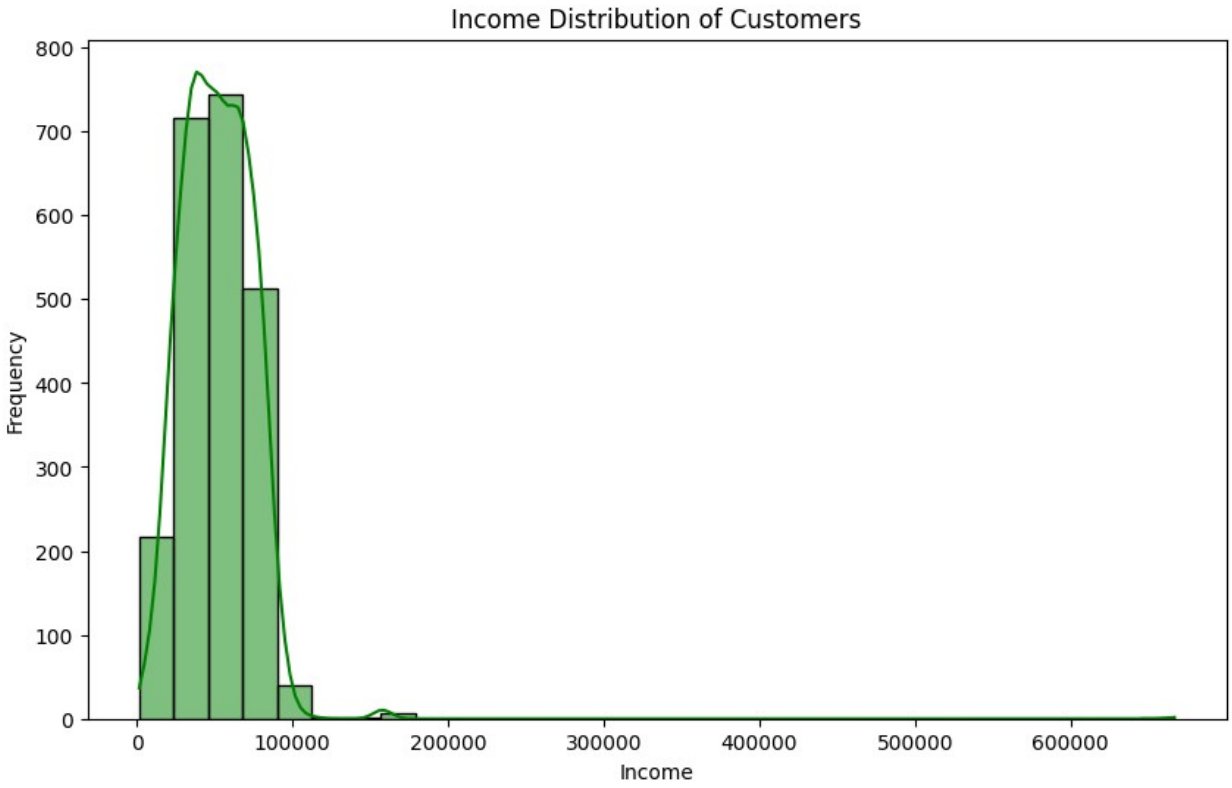
Recommendations:

- **Targeted marketing:** Focus marketing efforts on the younger customer segment (40-50 years old) while also considering the needs and preferences of older customers.
- **Age-specific product offerings:** Tailor products and services to different age groups.
- **Loyalty programs:** Implement loyalty programs to reward repeat customers, especially in younger age groups.
- **Customer experience:** Ensure a positive customer experience for all age groups through personalized recommendations, excellent service, and accessible platforms.

By leveraging these insights and implementing targeted strategies, you can effectively engage with customers of all ages and drive business growth.

```
# Income Distribution
plt.figure(figsize=(10, 6))
sns.histplot(data_encoded['Income'], bins=30, kde=True, color='green')
```

```
plt.title('Income Distribution of Customers')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.show()
```



Income Distribution Analysis

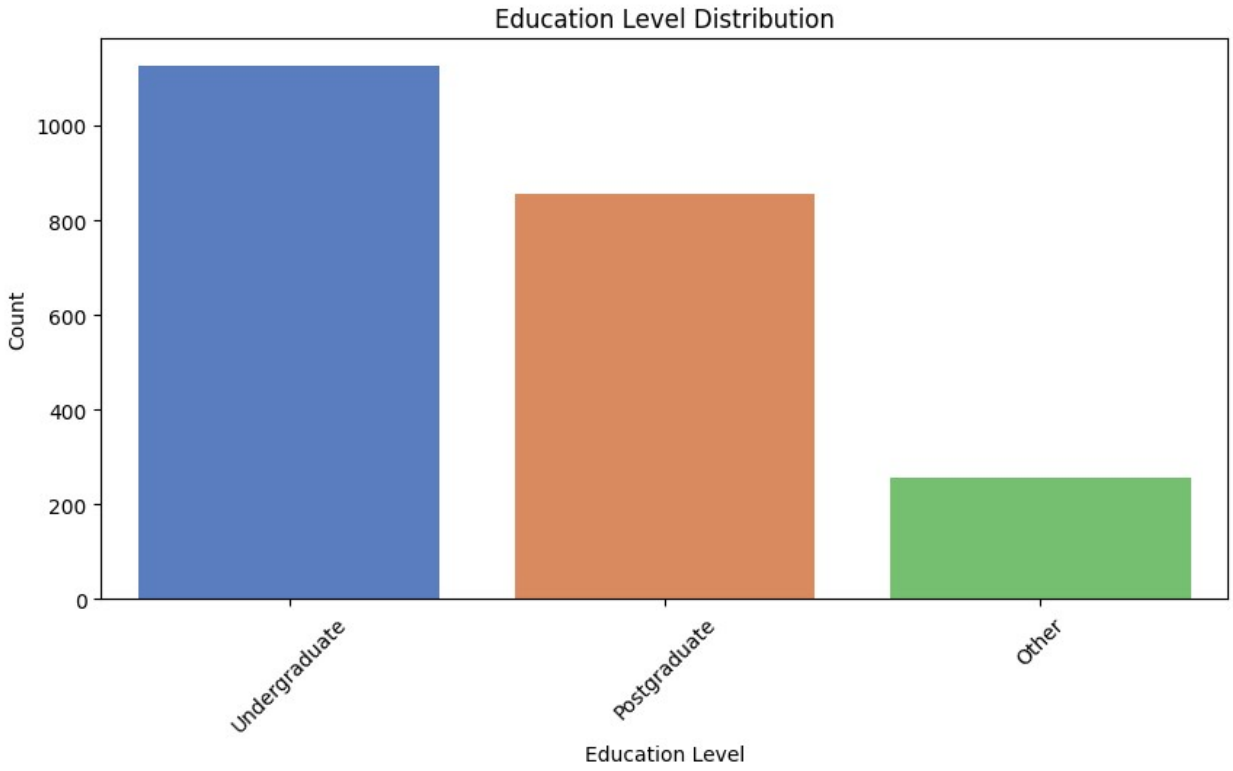
Strategic Recommendations

- **Targeted Marketing:**
  - **Focus on lower- to middle-income segments:** Tailor marketing campaigns and offers to appeal to the majority of customers who fall within these income brackets. This could involve targeted promotions, discounts, and loyalty programs designed to attract and retain these customers.
  - **Develop strategies for high-income customers:** While representing a smaller segment, high-income customers can contribute significantly to overall revenue. Implement strategies to identify and engage with these customers through exclusive offers, personalized services, and premium product offerings.
- **Product and Service Differentiation:**
  - **Tiered pricing and product offerings:** Consider offering a range of products and services at different price points to cater to the diverse income levels within the customer base. This allows customers to choose options that align with their budgets and preferences.

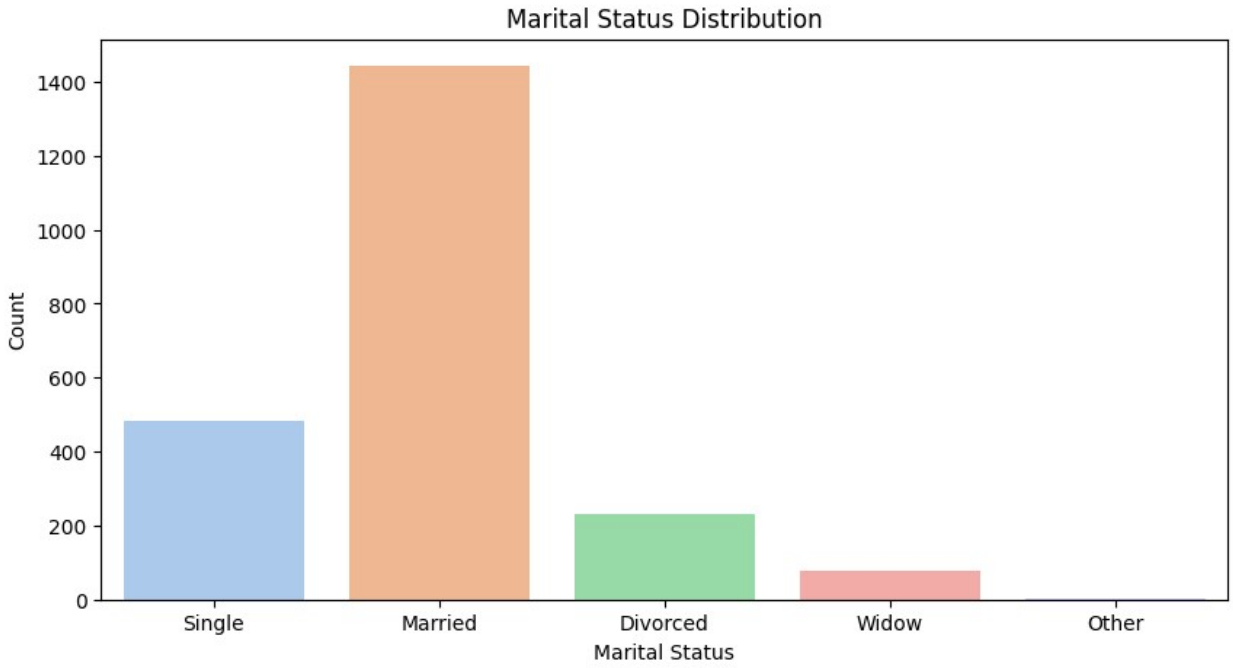
- **Value-driven offerings:** Emphasize value and affordability in marketing messages and product descriptions to appeal to price-sensitive customers in the lower-income segments.

```
# Education Breakdown
plt.figure(figsize=(10, 5))
sns.countplot(x='Education', data=data, palette='muted')
plt.title('Education Level Distribution')
plt.xlabel('Education Level')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()

# Marital Status Breakdown
plt.figure(figsize=(10, 5))
sns.countplot(x='Marital_Status', data=data, palette='pastel')
plt.title('Marital Status Distribution')
plt.xlabel('Marital Status')
plt.ylabel('Count')
plt.show()
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\388669412.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Education', data=data, palette='muted')
```




```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\388669412.py:12:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Marital_Status', data=data, palette='pastel')
```



Education Level Distribution Analysis

Insights:

- Undergraduates form the largest customer segment.
- A significant portion of customers hold postgraduate degrees.
- Customers with "Other" educational backgrounds constitute a smaller segment.

Recommendations:

- **Target Undergraduates:** Focus marketing efforts on this key segment.
- **Engage Postgraduates:** Develop strategies to attract and retain this high-potential group.
- **Consider "Other" Segment:** Tailor offerings to meet their specific needs.

Marital Status Distribution Analysis

Key Insights

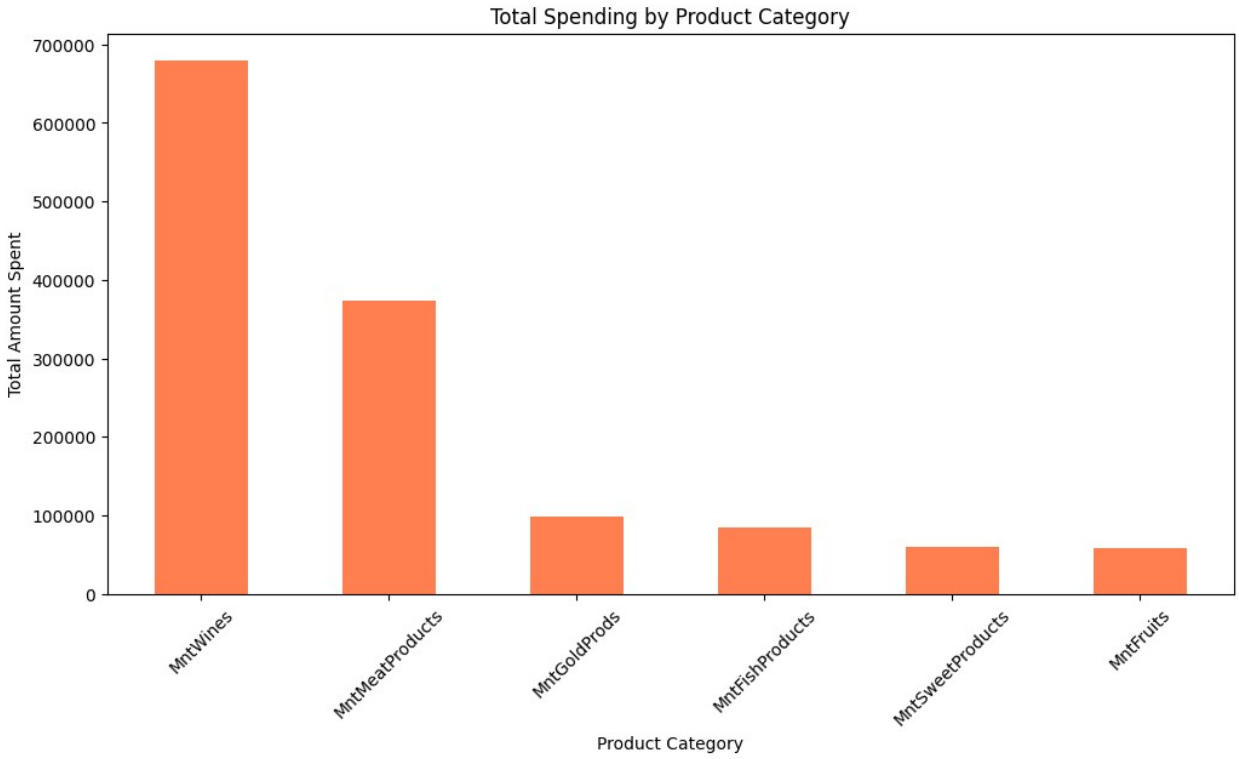
- **Married individuals constitute the largest segment:** The chart reveals that "Married" is the most dominant marital status among customers, indicating a significant market opportunity within this group.
- **Singles form a substantial segment:** "Single" individuals represent a considerable portion of the customer base, suggesting a significant market to target.
- **Smaller segments:** "Divorced," "Widow," and "Other" marital statuses represent smaller segments of the customer base. However, these segments should not be overlooked, as they still constitute a valuable portion of the market.

Strategic Recommendations

- **Targeted Marketing:**

  - **Focus on Married Couples:** Leverage targeted marketing campaigns and offers to engage with the substantial married customer segment. This could involve family-oriented promotions, special offers for couples, and targeted advertising.
  - **Attract Single Customers:** Develop strategies to attract and retain single customers. This could involve personalized offers, exclusive events, and content that resonates with their lifestyle and needs.
  - **Consider the Smaller Segments:** While smaller, the "Divorced," "Widow," and "Other" segments still present market opportunities. Tailor marketing messages and product offerings to cater to the specific needs and preferences of these groups.
- **Product and Service Differentiation:**
  - **Offer diverse product lines:** Provide a range of products and services that cater to the varying needs and preferences of different marital statuses. This could include family-sized packages, individual-sized options, and personalized recommendations based on marital status.

```
# Spending Categories
spending_features = ['MntWines', 'MntFruits', 'MntMeatProducts',
'MntFishProducts', 'MntSweetProducts', 'MntGoldProds']
total_spending =
data[spending_features].sum().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
total_spending.plot(kind='bar', color='coral')
plt.title('Total Spending by Product Category')
plt.xlabel('Product Category')
plt.ylabel('Total Amount Spent')
plt.xticks(rotation=45)
plt.show()
```



Total Spending by Product Category Analysis

Insights:

- **Wines are the most popular category:** Wine purchases account for the highest total spending, indicating significant customer demand.
- **Meat and Gold products follow:** Meat products and Gold products show the next highest spending levels, suggesting a strong market for these categories.
- **Other categories have lower spending:** Spending on fruits, fish, and sweets is considerably lower compared to wine, meat, and gold products.

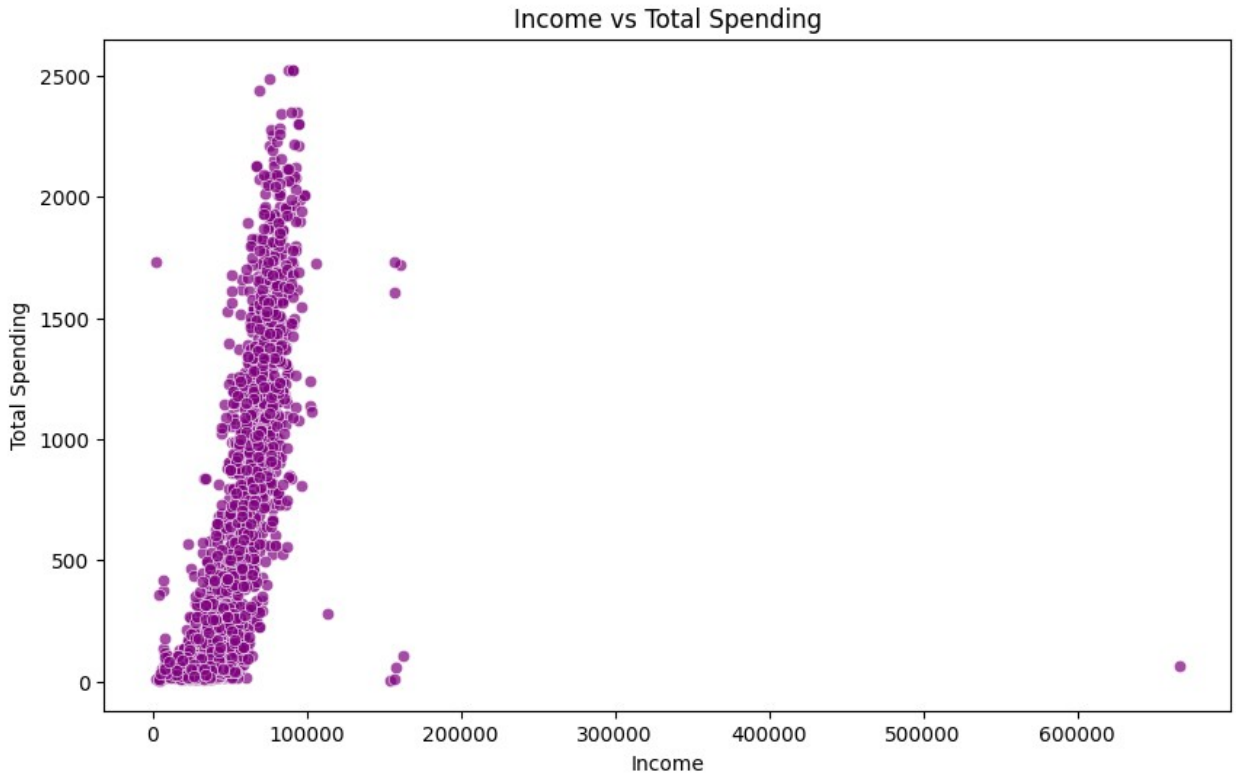
Recommendations:

- **Focus on Wine:** Leverage the popularity of wine with targeted promotions and exclusive offers.
- **Explore Meat and Gold Products:** Capitalize on the demand for meat and gold products by expanding offerings and exploring new product lines.
- **Promote other categories:** Implement strategies to increase spending on fruits, fish, and sweets, potentially through cross-selling or bundled offers.
- **Data-driven approach:** Continuously monitor spending trends across categories to adjust strategies and optimize product offerings.

Key Takeaway: Understanding spending patterns across product categories enables businesses to optimize inventory, tailor marketing efforts, and drive sales growth.

```
# Income vs Total Spending
plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(data=data_encoded, x='Income', y='MntTotal',
alpha=0.7, color='purple')
plt.title('Income vs Total Spending')
plt.xlabel('Income')
plt.ylabel('Total Spending')
plt.show()
```



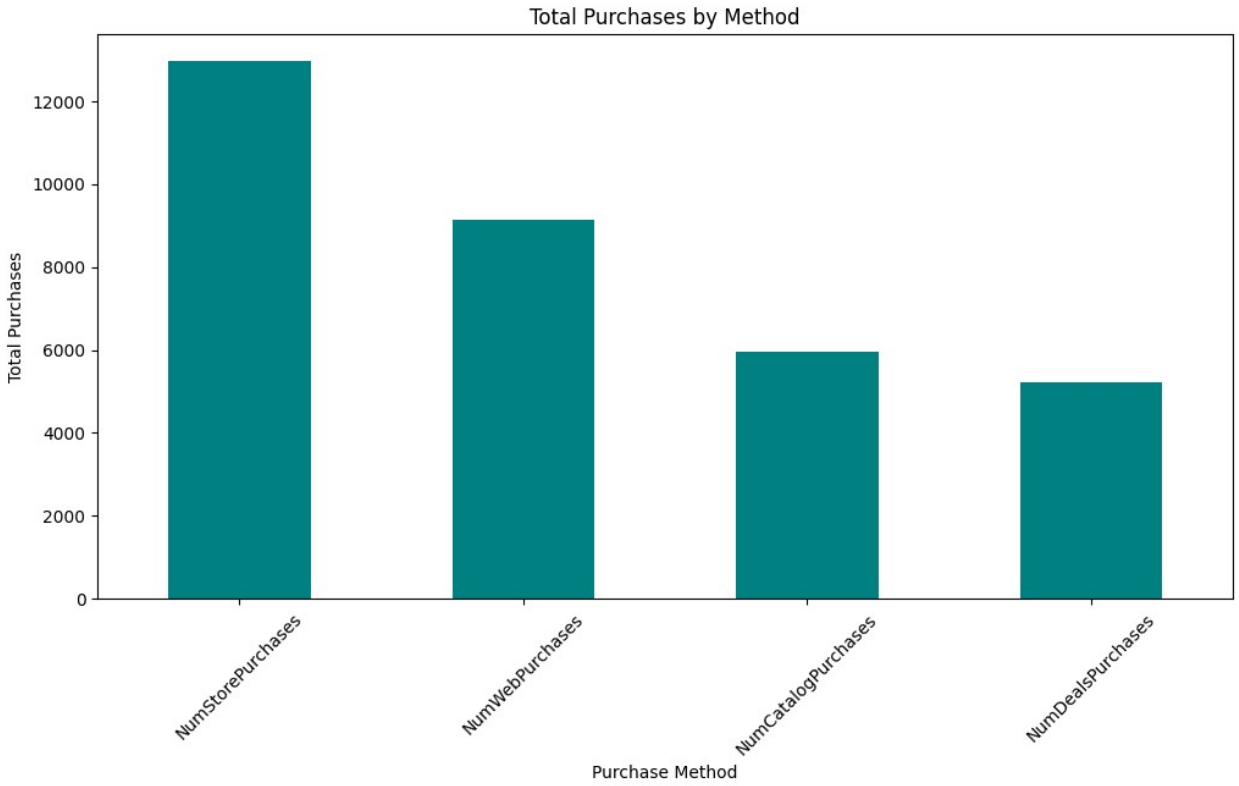
Income vs Total Spending Analysis

Insights:

- **Positive Correlation:** The scatter plot reveals a positive correlation between income and total spending. This indicates that customers with higher incomes tend to spend more on products across categories.
- **Heterogeneous Spending:** While there's a general trend of higher spending with increasing income, there's significant variability within each income bracket. This suggests that factors beyond income influence individual spending behavior.

```
# Purchase Methods Distribution
purchase_features = ['NumDealsPurchases', 'NumWebPurchases',
'NumCatalogPurchases', 'NumStorePurchases']
total_purchases =
data[purchase_features].sum().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
total_purchases.plot(kind='bar', color='teal')
```

```
plt.title('Total Purchases by Method')
plt.xlabel('Purchase Method')
plt.ylabel('Total Purchases')
plt.xticks(rotation=45)
plt.show()
```



Total Purchases by Method Analysis

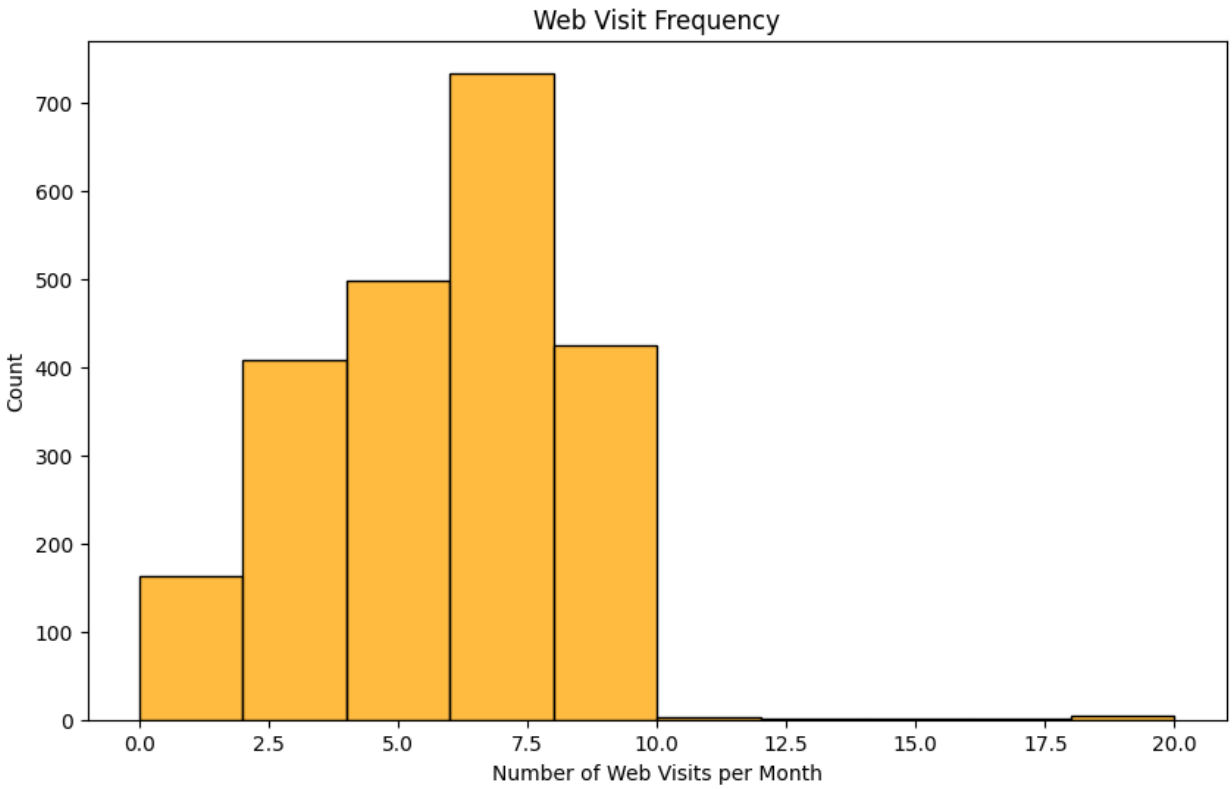
Insights:

- **Store Purchases Dominate:** Store purchases account for the highest number of transactions, indicating a strong preference for in-store shopping among customers.
- **Web Purchases Second:** Online purchases are the second most popular method, reflecting a growing online presence and e-commerce adoption.
- **Catalog and Deal Purchases Lower:** Catalog and deal purchases have lower frequencies compared to store and web purchases.

Recommendations:

- **Optimize In-Store Experience:** Focus on enhancing the in-store shopping experience to maintain its popularity.
- **Strengthen Online Presence:** Invest in online channels to further capitalize on the growing demand for web purchases.
- **Promote Catalog and Deal Purchases:** Explore strategies to increase the frequency of catalog and deal purchases, such as targeted promotions and personalized offers.

```
# Web Visit Frequency
plt.figure(figsize=(10, 6))
sns.histplot(data['NumWebVisitsMonth'], bins=10, kde=False,
color='orange')
plt.title('Web Visit Frequency')
plt.xlabel('Number of Web Visits per Month')
plt.ylabel('Count')
plt.show()
```



Web Visit Frequency Analysis

Insights:

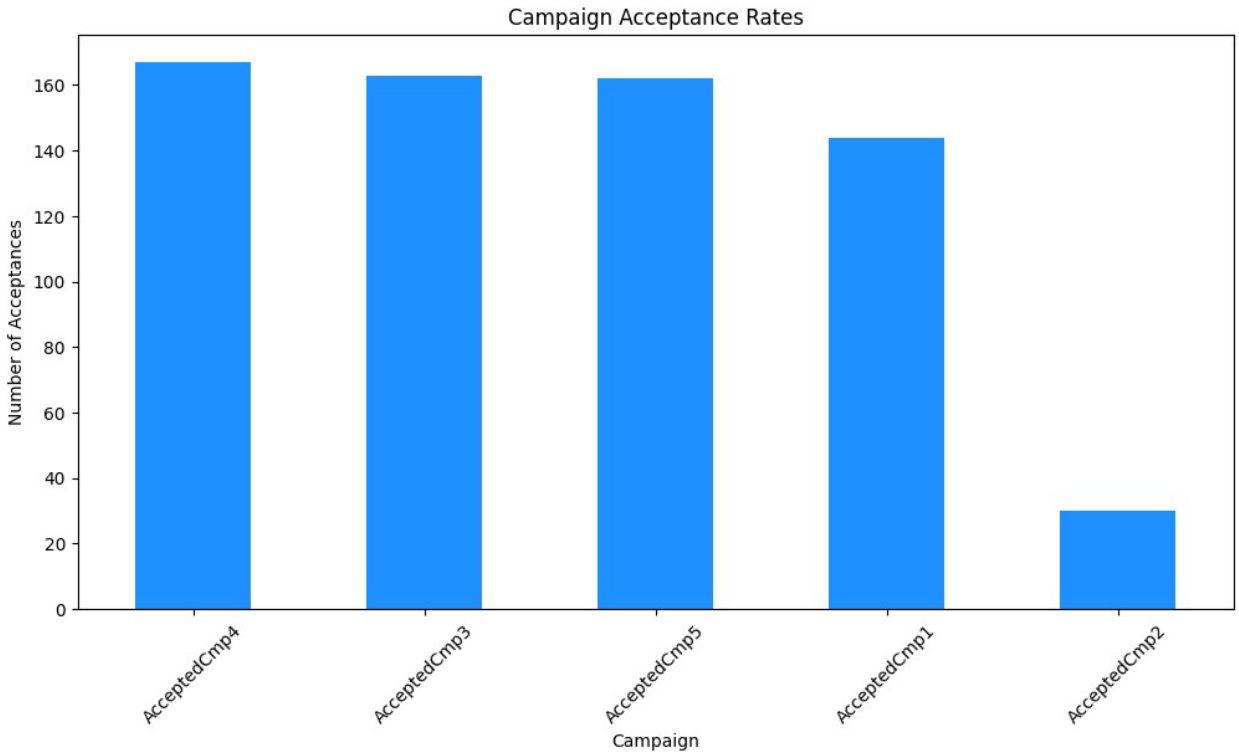
- **Most customers visit the website infrequently:** The distribution shows a peak around 5-7 web visits per month, indicating that a majority of customers do not visit the website frequently.
- **Long tail:** A small portion of customers visit the website very frequently (more than 15 times per month).

Recommendations:

- **Improve website engagement:** Implement strategies to increase website engagement for the majority of customers who visit infrequently. This could include personalized recommendations, targeted promotions, and interactive content.

- **Leverage high-frequency visitors:** Identify and engage with the small group of highly engaged customers. This could involve exclusive offers, loyalty programs, and personalized communication.

```
# Campaign Acceptance Rates
campaign_features = ['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3',
'AcceptedCmp4', 'AcceptedCmp5']
campaign_responses =
data[campaign_features].sum().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
campaign_responses.plot(kind='bar', color='dodgerblue')
plt.title('Campaign Acceptance Rates')
plt.xlabel('Campaign')
plt.ylabel('Number of Acceptances')
plt.xticks(rotation=45)
plt.show()
```



Campaign Acceptance Rates Analysis

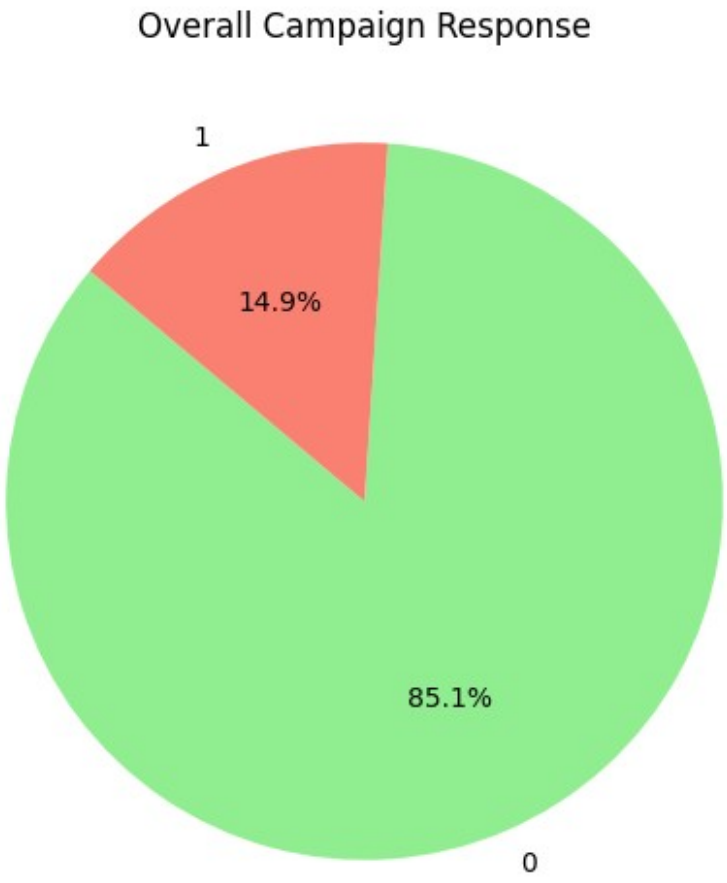
Insights:

- **Campaigns 3, 4, and 5 show high acceptance rates:** These campaigns have consistently higher acceptance rates compared to campaigns 1 and 2.
- **Campaign 2 has the lowest acceptance rate:** This campaign has the lowest acceptance rate, suggesting potential issues with its messaging, targeting, or execution.

Recommendations:

- **Analyze successful campaigns (3, 4, 5):** Identify key elements (e.g., messaging, targeting, offers) that contributed to their success.
- **Review and improve Campaign 2:** Analyze the reasons for its low acceptance rate and implement necessary changes to improve performance in future campaigns.
- **A/B testing:** Conduct A/B testing to compare different campaign variations and identify the most effective elements.

```
# Overall Campaign Response
plt.figure(figsize=(6, 6))
data['Response'].value_counts().plot.pie(autopct='%1.1f%%',
startangle=140, colors=['lightgreen', 'salmon'])
plt.title('Overall Campaign Response')
plt.ylabel('')
plt.show()
```



Overall Campaign Response Analysis

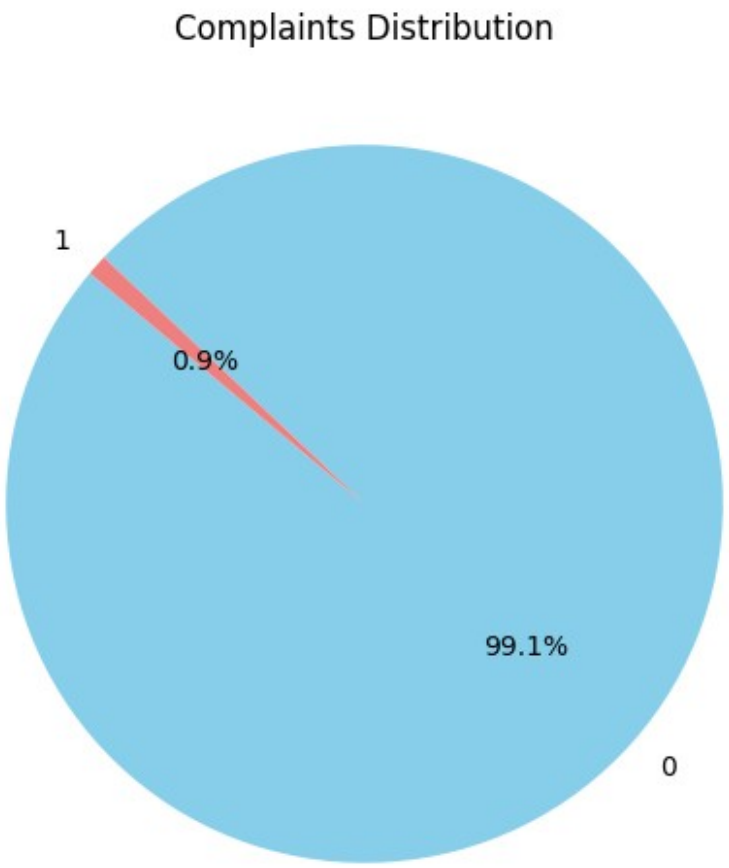
Insights:

- **Low Overall Response Rate:** The pie chart reveals that only 14.9% of customers responded positively to the campaigns, indicating a low overall response rate.

Recommendations:

- **Analyze Campaign Effectiveness:** Conduct a thorough analysis of campaign performance, identifying areas for improvement in targeting, messaging, and offers.
- **Improve Campaign Targeting:** Refine customer segmentation and targeting strategies to reach the most receptive audience segments.
- **Enhance Campaign Messaging:** Optimize campaign messaging to increase relevance, clarity, and appeal to the target audience.
- **Personalize Offers:** Implement personalized offers and communication strategies to increase engagement and response rates.
- **A/B Testing:** Conduct A/B testing to compare different campaign variations and identify the most effective elements.

```
# Complaints Distribution
plt.figure(figsize=(6, 6))
data['Complain'].value_counts().plot.pie(autopct='%1.1f%%',
startangle=140, colors=['skyblue', 'lightcoral'])
plt.title('Complaints Distribution')
plt.ylabel('')
plt.show()
```



Complaints Distribution Analysis

Insights:

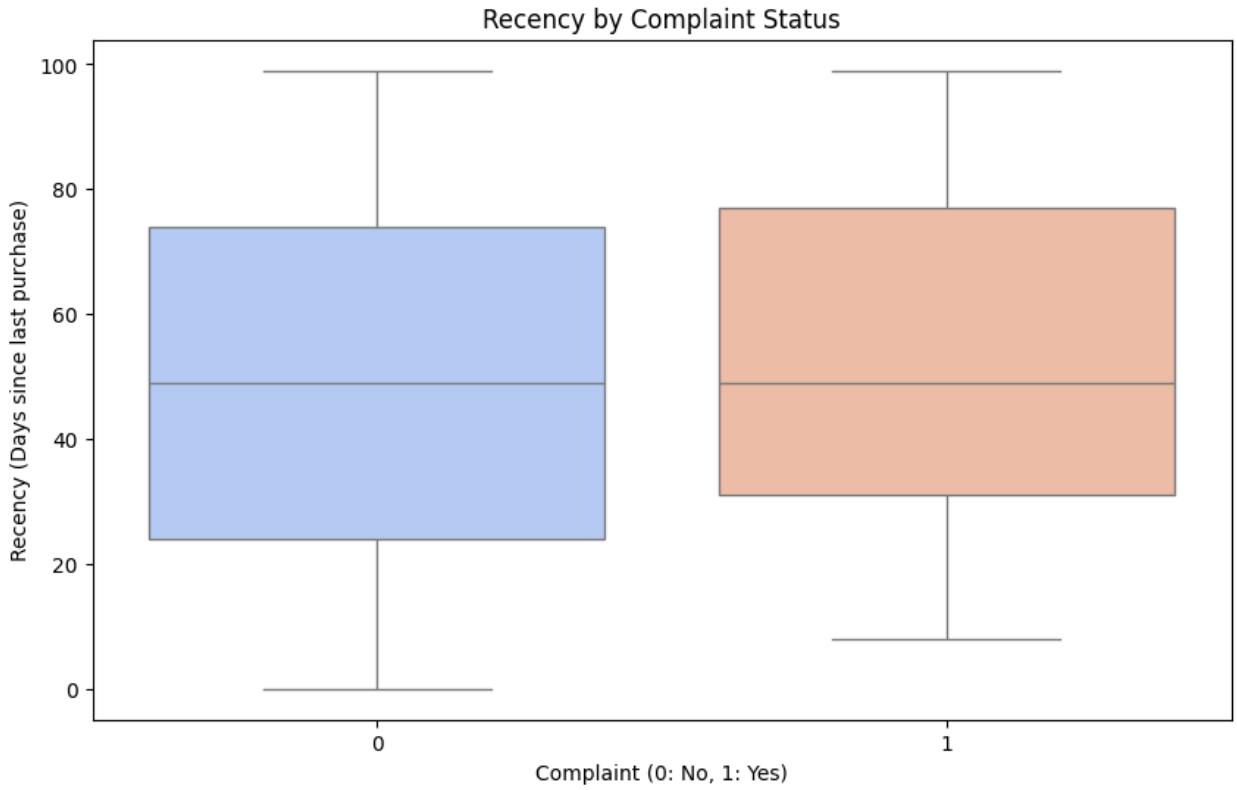
- **Low Complaint Rate:** The pie chart shows a very low complaint rate, with only 0.9% of customers having filed a complaint.
- **Class Imbalance:** This distribution represents a significant class imbalance, which can pose challenges for predictive modeling tasks related to complaint prediction.

Recommendations:

- **Focus on Root Cause Analysis:** Investigate the root causes of the existing complaints to identify areas for improvement in customer service and product quality.
- **Proactive Customer Service:** Implement proactive customer service strategies to prevent complaints from occurring in the first place. This could include proactive communication, proactive issue resolution, and proactive customer support.
- **Data-Driven Approach:** Leverage customer feedback and data analytics to identify patterns and trends in customer complaints. This information can be used to improve products, services, and overall customer experience.
- **Class Imbalance Handling:** If building predictive models for complaint prediction, address the class imbalance issue through techniques such as oversampling, undersampling, or using appropriate loss functions.

```
# Complaints vs Recency
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Complain', y='Recency', palette='coolwarm')
plt.title('Recency by Complaint Status')
plt.xlabel('Complaint (0: No, 1: Yes)')
plt.ylabel('Recency (Days since last purchase)')
plt.show()
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\3703302215.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=data, x='Complain', y='Recency',
palette='coolwarm')
```



Recency by Complaint Status Analysis

Insights:

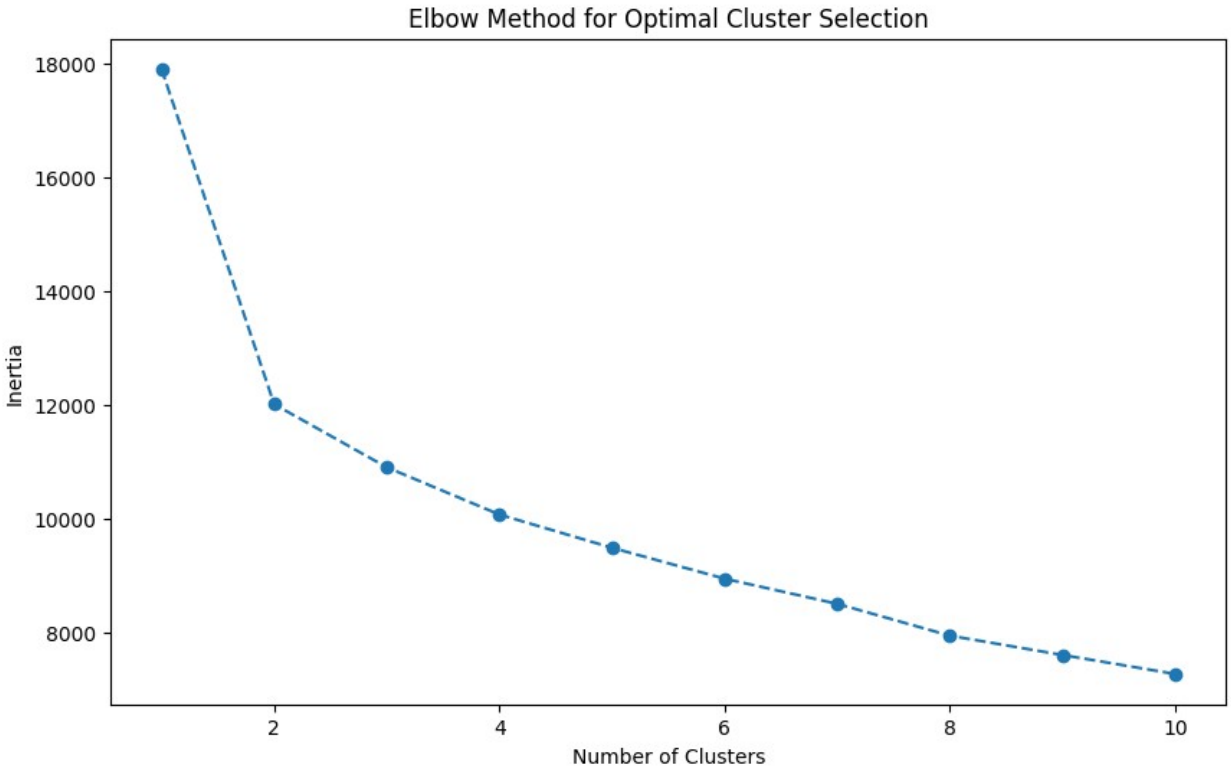
- **Higher Recency for Customers with Complaints:** The box plot shows that customers who have filed a complaint tend to have higher recency values (i.e., more days since their last purchase) compared to customers who have not filed a complaint.

Recommendations:

- **Proactive Engagement:** Focus on proactively engaging with customers who have been inactive for an extended period, especially those who have previously filed complaints. This could involve personalized outreach, targeted offers, and proactive customer support.
- **Investigate Complaint Causes:** Conduct a thorough analysis of the reasons behind complaints and address the underlying issues to improve customer satisfaction and prevent future complaints.
- **Monitor Customer Behavior:** Continuously monitor customer behavior and identify patterns associated with increased complaint likelihood. This could involve tracking recency, purchase history, and other relevant metrics.

```
from sklearn.preprocessing import StandardScaler

# Select relevant features for clustering
features = ['Income', 'Customer_Age', 'MntTotal', 'NumWebPurchases',
            'NumCatalogPurchases', 'NumStorePurchases', 'Recency',
```

```
'Customer_Since']

# Select only the relevant features from the encoded DataFrame
data_segmentation = data_encoded[features].copy()

# Handle any missing values if still present
data_segmentation =
data_segmentation.fillna(data_segmentation.median())

# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_segmentation)

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

# Find optimal number of clusters using Elbow Method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

# Plot Elbow Curve
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal Cluster Selection')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```



Elbow Method for Optimal Cluster Selection

This plot illustrates the Elbow Method, a technique used to determine the optimal number of clusters for K-Means clustering.

Key Observations:

- **Inertia:** The y-axis represents "Inertia," which measures the within-cluster sum of squares. In essence, it quantifies how tightly the data points are clustered around their respective centroids.
- **Number of Clusters:** The x-axis represents the number of clusters (K) used in the K-Means algorithm.

Interpretation:

As the number of clusters (K) increases, the inertia generally decreases. This is because with more clusters, data points are closer to their assigned centroids. However, at some point, the decrease in inertia starts to slow down, forming an "elbow" in the plot.

Identifying the Optimal Number of Clusters:

The "elbow point" typically indicates the optimal number of clusters. In this plot, the elbow appears to be around **K = 3 or K = 4**. Choosing these values as the number of clusters is likely to strike a balance between minimizing within-cluster variation and avoiding overfitting.

```
optimal_clusters = 4
```

```
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42,
n_init=10)
data_segmentation['Cluster'] = kmeans.fit_predict(data_scaled)

# Preview clusters
print(data_segmentation.head())
```

```
    Income  Customer_Age  MntTotal  NumWebPurchases
NumCatalogPurchases  \
0  58138.0            67      1617                8
10
1  46344.0            70        27                1
1
2  71613.0            59       776                8
2
3  26646.0            40        53                2
0
4  58293.0            43       422                5
3

   NumStorePurchases  Recency  Customer_Since  Cluster
0                  4       58            4498        0
1                  2       38            3948        2
2                 10       26            4147        1
3                  4       26            3974        2
4                  6       94            3996        3
```

Cluster Profiling and Insights

```
# Group by Cluster and calculate key statistics
cluster_summary = data_segmentation.groupby('Cluster').mean().round(2)
cluster_summary['Count'] = data_segmentation['Cluster'].value_counts()
print(cluster_summary)
```

```
            Income  Customer_Age  MntTotal  NumWebPurchases  \
Cluster
0        77220.37         54.89   1433.85             4.98
1        59126.71         59.88    745.77             6.79
2        34446.30         51.00     96.43             2.03
3        37605.97         54.43    135.99             2.37

         NumCatalogPurchases  NumStorePurchases  Recency
Customer_Since  \
Cluster

0                       6.43               8.61    52.91
4188.64
1                       2.93               7.74    42.85
4251.49
2                       0.53               3.21    25.16
```

```
4143.03
3                       0.74                3.48    75.76
4167.71

         Count
Cluster
0          551
1          586
2          548
3          553
```

Cluster Summary

The `cluster_summary` DataFrame reveals distinct customer segments:

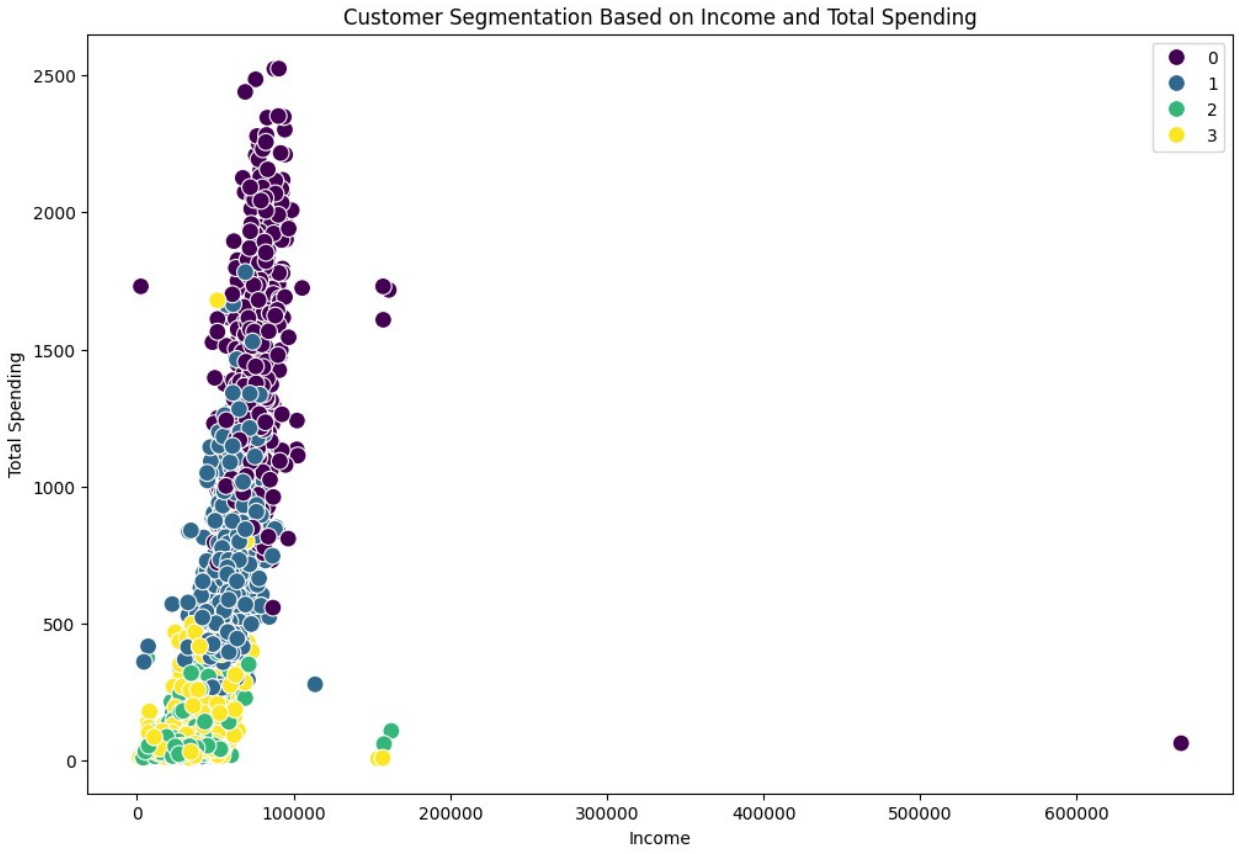
- **Cluster 0:** High-spending, high-income customers with moderate recency.
- **Cluster 1:** High-spending, moderate-income customers with moderate recency.
- **Cluster 2:** Low-spending, low-income customers with low recency.
- **Cluster 3:** Moderate-spending, moderate-income customers with high recency.

Implications:

- **Tailor marketing:** Implement targeted strategies for each cluster (e.g., exclusive offers for Cluster 0, re-engagement campaigns for Cluster 3).
- **Optimize resource allocation:** Allocate resources effectively based on the potential value of each segment.
- **Enhance customer experience:** Provide personalized experiences to meet the unique needs of each customer segment.

This analysis enables data-driven decision-making to improve customer satisfaction and business growth.

```
import seaborn as sns

# Plot clusters using two main features (e.g., Income vs MntTotal)
plt.figure(figsize=(12, 8))
sns.scatterplot(data=data_segmentation, x='Income', y='MntTotal',
hue='Cluster', palette='viridis', s=100)
plt.title('Customer Segmentation Based on Income and Total Spending')
plt.xlabel('Income')
plt.ylabel('Total Spending')
plt.legend()
plt.show()
```



Customer Segmentation Based on Income and Total Spending

This scatter plot visualizes customer segmentation based on their Income and Total Spending. Each point represents an individual customer, and the color of the point indicates the cluster to which the customer belongs.

Insights:

- **Cluster 0 (Purple):** This cluster is characterized by customers with high incomes and high total spending. They are likely the most valuable customers for the business.
- **Cluster 1 (Blue):** This cluster comprises customers with moderate to high incomes and moderate total spending. They represent a significant segment of the customer base.
- **Cluster 2 (Green):** This cluster consists of customers with low incomes and low total spending. They may have lower purchasing power and require different marketing strategies.
- **Cluster 3 (Yellow):** This cluster includes customers with high incomes but relatively low total spending. They may be interested in premium products or services but have not yet made significant purchases.

Implications:

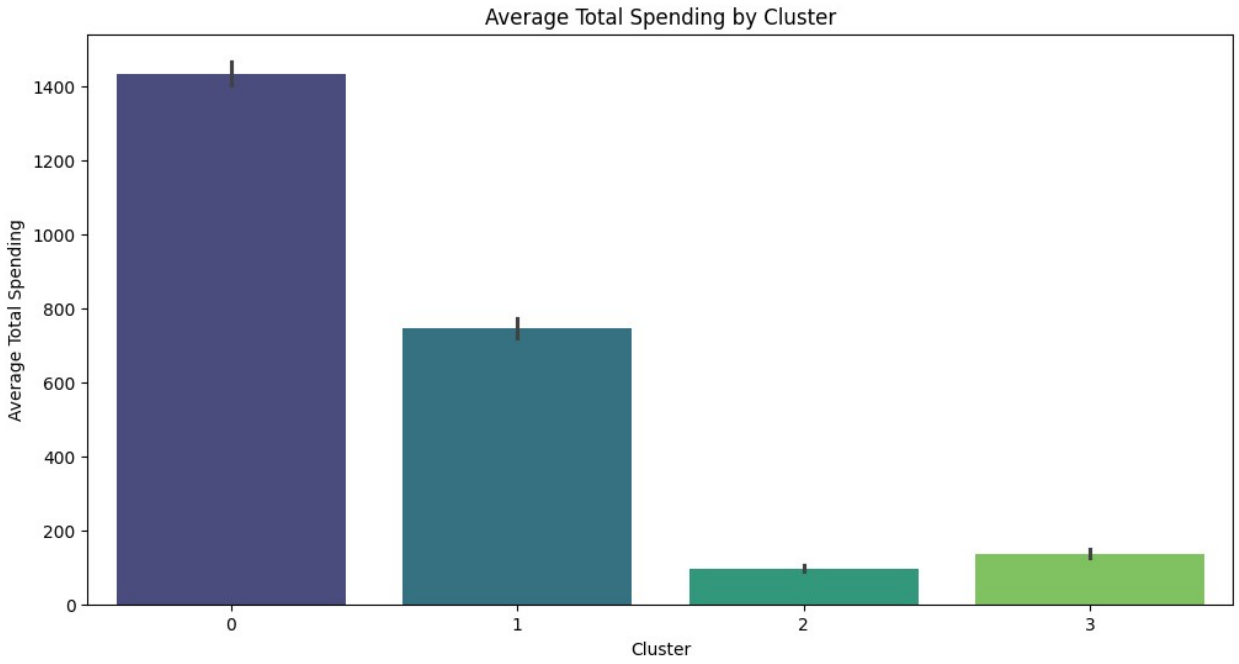
- **Product Development:** The segmentation can inform product development strategies. By understanding the spending patterns of different customer clusters, businesses can develop and offer products and services that cater to the specific needs and preferences of each segment.
- **Customer Relationship Management:** The segmentation can be used to prioritize customer interactions and allocate resources effectively. For example, businesses can focus on building stronger relationships with high-value customers in Cluster 0 while implementing strategies to reactivate inactive customers in Cluster 2.

3. Visualize Cluster Characteristics

3.1 Cluster Spending Behavior

```
# Total Spending by Cluster
plt.figure(figsize=(12, 6))
sns.barplot(data=data_segmentation, x='Cluster', y='MntTotal',
palette='viridis')
plt.title('Average Total Spending by Cluster')
plt.xlabel('Cluster')
plt.ylabel('Average Total Spending')
plt.show()
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\3258182260.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=data_segmentation, x='Cluster', y='MntTotal',
palette='viridis')
```



Average Total Spending by Cluster Analysis

Insights:

- **Cluster 0** has the highest average total spending, indicating this segment is the most valuable to the business.
- **Clusters 1, 2, and 3** have progressively lower average total spending.

Recommendations:

- **Focus on Cluster 0:** Tailor marketing efforts to retain and further engage customers in this high-spending segment.
- **Re-engage Cluster 3:** Implement strategies to reactivate customers in Cluster 3, who have lower spending levels.

3.2 Cluster Distribution by Age and Income

```
# Cluster Distribution by Age and Income
plt.figure(figsize=(12, 8))
sns.scatterplot(data=data_segmentation, x='Income', y='Customer_Age',
hue='Cluster', palette='deep', s=100)
plt.title('Customer Segmentation by Age and Income')
plt.xlabel('Income')
plt.ylabel('Customer Age')
plt.legend()
plt.show()
```

Customer Segmentation by Age and Income

Insights:

- **Cluster 0 (Blue):** This cluster comprises older customers with high incomes.
- **Cluster 1 (Orange):** This cluster includes customers with a wide range of ages and moderate incomes.
- **Cluster 2 (Green):** This cluster consists of younger customers with varying income levels.
- **Cluster 3 (Red):** This cluster includes a mix of ages with very high incomes.

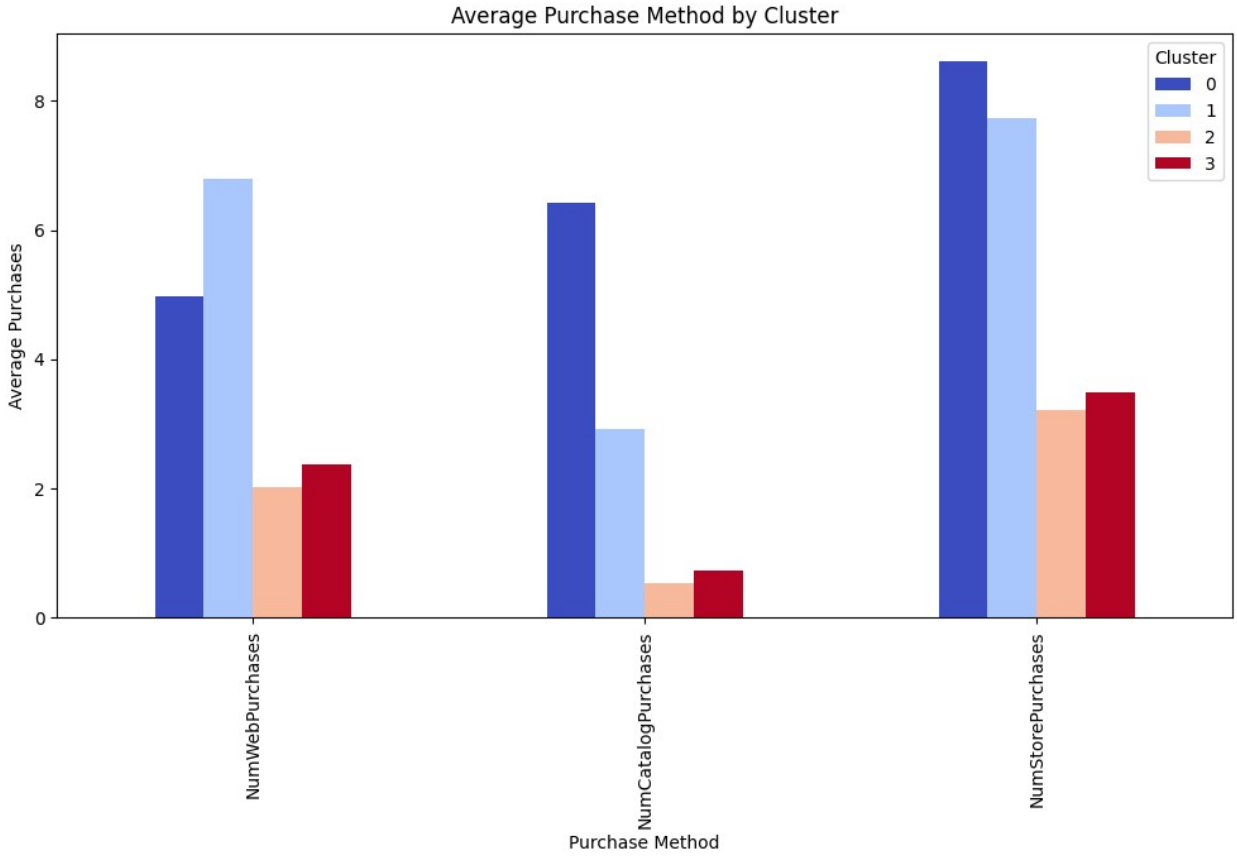
Recommendations:

- **Customer Relationship Management:** Focus on building strong relationships with high-value customers in Clusters 0 and 3.

3.3 Purchase Channel Preferences by Cluster

```
# Average Purchases by Cluster
purchase_features = ['NumWebPurchases', 'NumCatalogPurchases',
'NumStorePurchases']
cluster_purchase = data_segmentation.groupby('Cluster')
[purchase_features].mean().T

cluster_purchase.plot(kind='bar', figsize=(12, 6),
```

```
colormap='coolwarm')
plt.title('Average Purchase Method by Cluster')
plt.ylabel('Average Purchases')
plt.xlabel('Purchase Method')
plt.legend(title='Cluster')
plt.show()
```



Average Purchase Method by Cluster Analysis

Insights:

- **Cluster 0: (Blue)** This cluster exhibits a strong preference for store purchases, followed by web and catalog purchases.
- **Cluster 1: (Light Blue)** This cluster shows a balanced preference across all purchase methods.
- **Cluster 2: (Orange)** This cluster primarily relies on store purchases, with significantly lower usage of web and catalog channels.
- **Cluster 3: (Red)** This cluster heavily favors store purchases, with minimal usage of web and catalog channels.

Recommendations:

- **Targeted Marketing:** Tailor marketing strategies across different channels based on cluster preferences. For example, focus on online promotions for Cluster 1 and in-store promotions for Clusters 2 and 3.
- **Inventory Management:** Optimize inventory levels across different channels based on cluster purchase behavior.
- **Channel Expansion:** Encourage customers in Clusters 2 and 3 to explore online and catalog channels through targeted promotions and incentives.

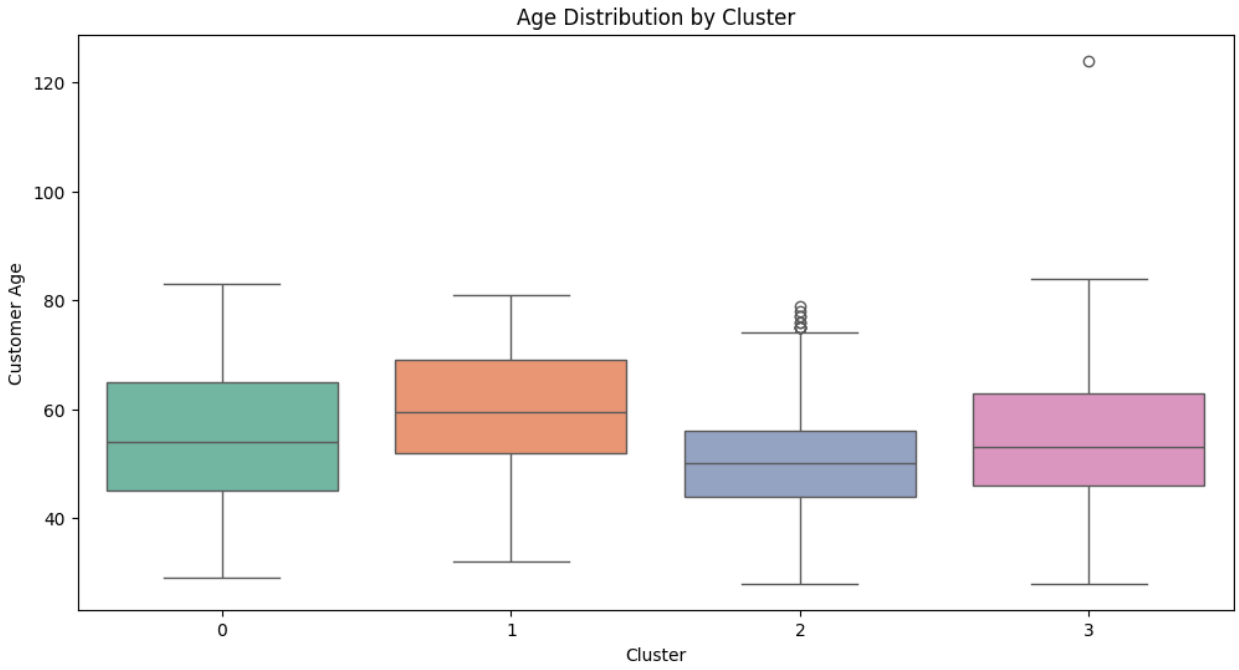
Cluster Demographic Analysis:

3.4 Age Distribution by Cluster

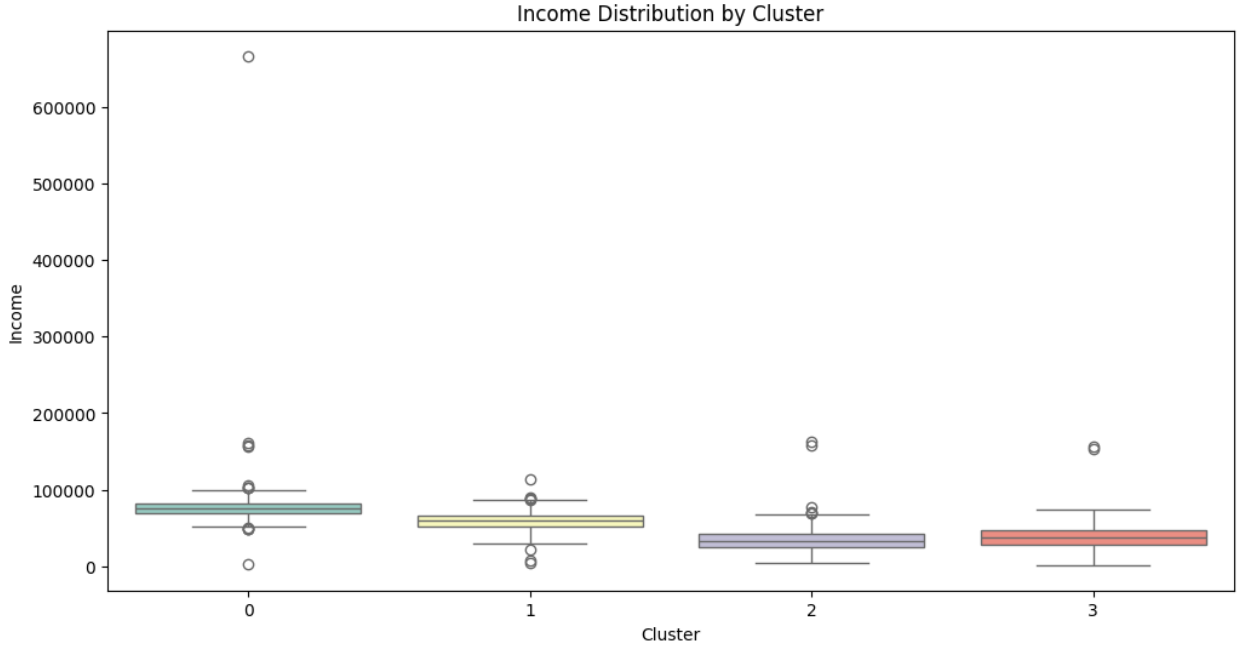
3.5 Income Distribution by Cluster

```
# Age Distribution by Cluster
plt.figure(figsize=(12, 6))
sns.boxplot(x='Cluster', y='Customer_Age', data=data_segmentation,
palette='Set2')  # Changed data to data_segmentation
plt.title('Age Distribution by Cluster')
plt.xlabel('Cluster')
plt.ylabel('Customer Age')
plt.show()

# Income Distribution by Cluster
plt.figure(figsize=(12, 6))
sns.boxplot(x='Cluster', y='Income', data=data_segmentation,
palette='Set3')  # Changed data to data_segmentation
plt.title('Income Distribution by Cluster')
plt.xlabel('Cluster')
plt.ylabel('Income')
plt.show()
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\563044142.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='Cluster', y='Customer_Age', data=data_segmentation,
palette='Set2')  # Changed data to data_segmentation
```




```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\563044142.py:11:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='Cluster', y='Income', data=data_segmentation,
palette='Set3')  # Changed data to data_segmentation
```



Age Distribution by Cluster Analysis

Insights:

- **Cluster 0:** This cluster has the youngest customer base with a median age below 50.
- **Cluster 1:** This cluster has a slightly older customer base compared to Cluster 0, with a median age around 60.
- **Cluster 2:** This cluster has the oldest customer base with a median age above 60.
- **Cluster 3:** This cluster has a moderate age distribution with a median age around 60.

Recommendations:

- **Product Differentiation:** Offer products and services that cater to the age preferences of each cluster.
- **Customer Relationship Management:** Focus on building strong relationships with younger customers in Cluster 0 and older customers in Clusters 1 and 2.

Income Distribution by Cluster Analysis

Insights:

- **Cluster 0:** This cluster has the lowest median income.
- **Cluster 1:** This cluster has a moderate median income.
- **Cluster 2:** This cluster has the highest median income.
- **Cluster 3:** This cluster has a slightly lower median income compared to Cluster 2.

Recommendations:

- **Targeted Marketing:** Tailor marketing campaigns and offers to the income level of each cluster.
- **Customer Relationship Management:** Focus on building strong relationships with high-income customers in Clusters 2 and 3.

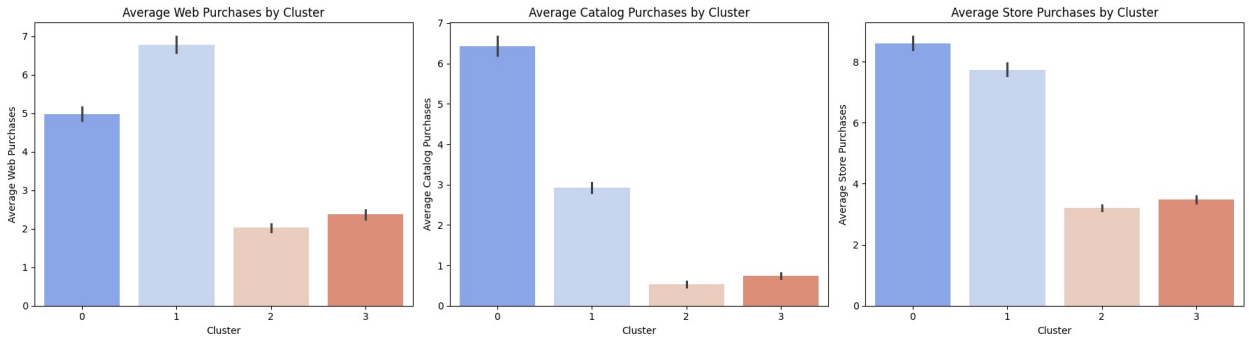
3.6 Cluster Engagement Behavior Analysis

```
# Web, Catalog, and Store Purchases by Cluster
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Web Purchases
sns.barplot(x='Cluster', y='NumWebPurchases', data=data_segmentation,
ax=axes[0], palette='coolwarm') # Changed data to data_segmentation
axes[0].set_title('Average Web Purchases by Cluster')
axes[0].set_xlabel('Cluster')
axes[0].set_ylabel('Average Web Purchases')

# Catalog Purchases
sns.barplot(x='Cluster', y='NumCatalogPurchases',
data=data_segmentation, ax=axes[1], palette='coolwarm') # Changed data
to data_segmentation
axes[1].set_title('Average Catalog Purchases by Cluster')
axes[1].set_xlabel('Cluster')
axes[1].set_ylabel('Average Catalog Purchases')
```

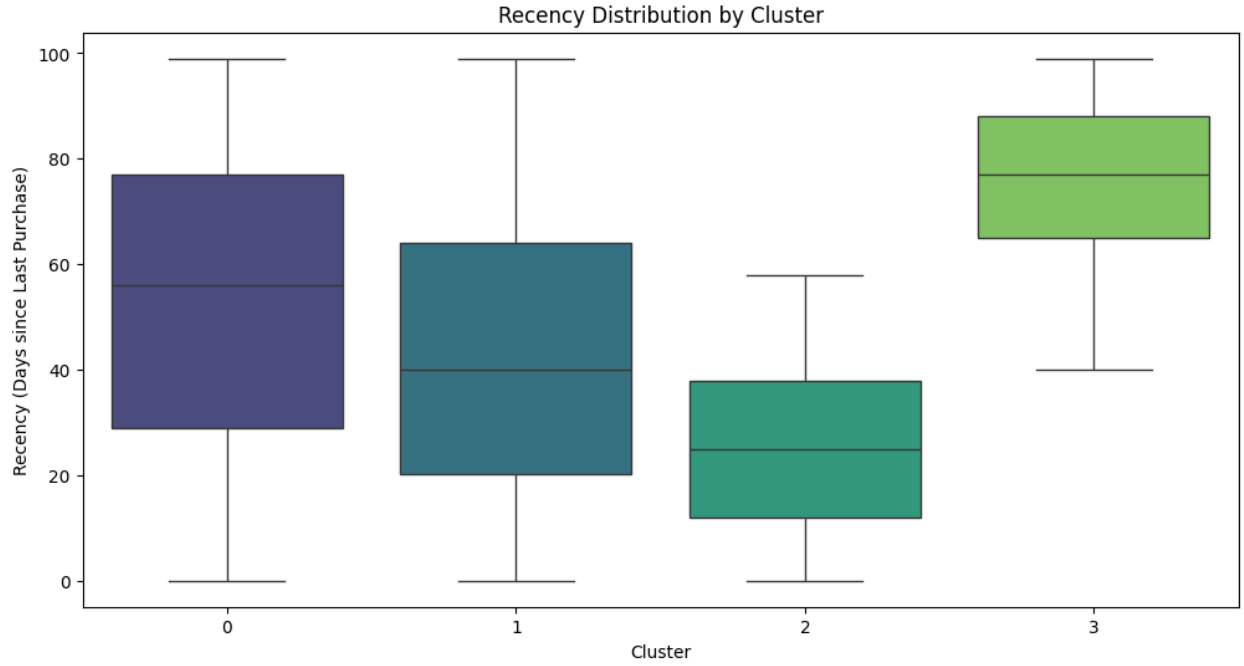
```
# Store Purchases
sns.barplot(x='Cluster', y='NumStorePurchases',
data=data_segmentation, ax=axes[2], palette='coolwarm') # Changed data
to data_segmentation
axes[2].set_title('Average Store Purchases by Cluster')
axes[2].set_xlabel('Cluster')
axes[2].set_ylabel('Average Store Purchases')

plt.tight_layout()
plt.show()

# Recency Trends by Cluster
plt.figure(figsize=(12, 6))
sns.boxplot(x='Cluster', y='Recency', data=data_segmentation,
palette='viridis') # Changed data to data_segmentation
plt.title('Recency Distribution by Cluster')
plt.xlabel('Cluster')
plt.ylabel('Recency (Days since Last Purchase)')
plt.show()
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\3679764269.py:5:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='Cluster', y='NumWebPurchases',
data=data_segmentation, ax=axes[0], palette='coolwarm') # Changed data
to data_segmentation
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\3679764269.py:11:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='Cluster', y='NumCatalogPurchases',
data=data_segmentation, ax=axes[1], palette='coolwarm') # Changed data
to data_segmentation
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\3679764269.py:17:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='Cluster', y='NumStorePurchases',
```

```
data=data_segmentation, ax=axes[2], palette='coolwarm') # Changed data to data_segmentation
```




```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\3679764269.py:27: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='Cluster', y='Recency', data=data_segmentation, palette='viridis') # Changed data to data_segmentation
```



Cluster Analysis Insights

1. Purchasing Behavior:

- **Web:** Cluster 1 leads, followed by Cluster 0. Clusters 2 and 3 have low web purchase activity.
- **Catalog:** Cluster 0 dominates, with Clusters 1 and 2 showing moderate activity. Cluster 3 has minimal catalog purchases.
- **Store:** Cluster 0 leads, followed by Cluster 1. Clusters 2 and 3 have significantly lower store purchases.

2. Recency:

- Cluster 2 has the highest recency (most recent purchases).
- Cluster 3 shows the longest recency (least recent purchases).
- Clusters 0 and 1 have moderate recency levels.

Preliminary Insights:

- **Cluster 0:** High spenders with high engagement across all channels.
- **Cluster 1:** Moderate spenders with a focus on web purchases.
- **Cluster 2:** Low spenders but recently active.
- **Cluster 3:** Low spenders and least active customers.

Cluster-Based Recommendations and Next Steps

1. Cluster 0: High-Value, High-Engagement Customers

- **Profile:** High spenders with significant engagement across web, catalog, and store channels.
- **Strategy:**
  - Implement loyalty programs to retain these customers.
  - Offer exclusive deals and early access to products.
  - Encourage referrals through referral bonuses.
  - Use personalized recommendations to increase spending further.

2. Cluster 1: Moderate Spenders, Web-Focused

- **Profile:** Moderate spenders, primarily engaged through online channels.
- **Strategy:**
  - Launch targeted email campaigns with personalized offers.
  - Improve online shopping experience with intuitive UI/UX enhancements.
  - Provide discounts on bulk web purchases.
  - Introduce loyalty rewards for frequent online transactions.

3. Cluster 2: Low Spenders, Recently Active

- **Profile:** Recently active but low spenders across all channels.
- **Strategy:**
  - Offer limited-time promotions to encourage purchases.
  - Send follow-up reminders and cart abandonment emails.
  - Provide entry-level membership incentives to increase engagement.
  - Conduct surveys to understand barriers to higher spending.

4. Cluster 3: Low Spenders, Inactive Customers

- **Profile:** Low engagement and high recency values, indicating churn risk.
- **Strategy:**
  - Launch a win-back campaign with compelling offers.
  - Use re-engagement email sequences with personalized deals.
  - Conduct customer satisfaction surveys to understand reasons for churn.
  - Highlight new product launches or company updates to regain interest.

Next Steps:

- **Continuous Monitoring:** Monitor customer behavior within each cluster over time to identify evolving trends and adjust strategies accordingly.
- **A/B Testing:** Conduct A/B testing on different marketing strategies to determine the most effective approaches for each cluster.
- **Data Analysis:** Utilize advanced analytics techniques to gain deeper insights into customer behavior and preferences within each cluster.

 RFM Analysis Results Summary

```
import pandas as pd

# RFM Analysis
data_segmentation['Recency'] = data_segmentation['Recency'] # Use
data_segmentation which has 'Cluster' column
data_segmentation['Frequency'] = data_segmentation['NumWebPurchases']
+ data_segmentation['NumCatalogPurchases'] +
data_segmentation['NumStorePurchases']
data_segmentation['Monetary'] = data_segmentation['MntTotal']

rfm = data_segmentation.groupby('Cluster').agg({ # Use
data_segmentation for groupby
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': 'mean'
}).reset_index()

# Display RFM analysis results
print(rfm)
```

```
   Cluster    Recency  Frequency     Monetary
0        0  52.907441  20.023593  1433.847550
1        1  42.848123  17.453925   745.767918
2        2  25.158759   5.762774    96.425182
3        3  75.764919   6.594937   135.989150
```

RFM Analysis Results Summary

Cluster	Insights
0	High spenders with frequent purchases and moderate recency. Valuable customers.
1	Moderate spenders with good frequency and relatively recent activity.

Cluster	Insights
2	Low spenders with low frequency but recent activity. Potential to grow.
3	Low spenders with infrequent purchases and long recency. At risk of churn.

This table summarizes the key insights from the RFM analysis, providing a clear and concise overview of each customer cluster.

Sentiment Analysis

```
# Check if all required columns exist
required_columns = ['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3',
'AcceptedCmp4', 'AcceptedCmp5', 'Response', 'Complain']
missing_columns = [col for col in required_columns if col not in
data.columns]

if missing_columns:
    print(f"Missing columns: {missing_columns}")
else:
    # Sentiment score calculation
    data['Sentiment_Score'] = (
        data['AcceptedCmp1'] +
        data['AcceptedCmp2'] +
        data['AcceptedCmp3'] +
        data['AcceptedCmp4'] +
        data['AcceptedCmp5'] +
        data['Response'] -
        data['Complain']
    )

    # Classify sentiment
    data['Sentiment'] = pd.cut(
        data['Sentiment_Score'],
        bins=[-float('inf'), 0, 2, float('inf')],
        labels=['Negative', 'Neutral', 'Positive']
    )

    # Display sentiment distribution
    sentiment_distribution = data['Sentiment'].value_counts()
    print(sentiment_distribution)
```

```
Sentiment
Negative    1631
Neutral      511
Positive      96
Name: count, dtype: int64
```

Sentiment Analysis Results

- **Negative:** 74.37%

- **Neutral:** 23.28%
- **Positive:** 4.35%

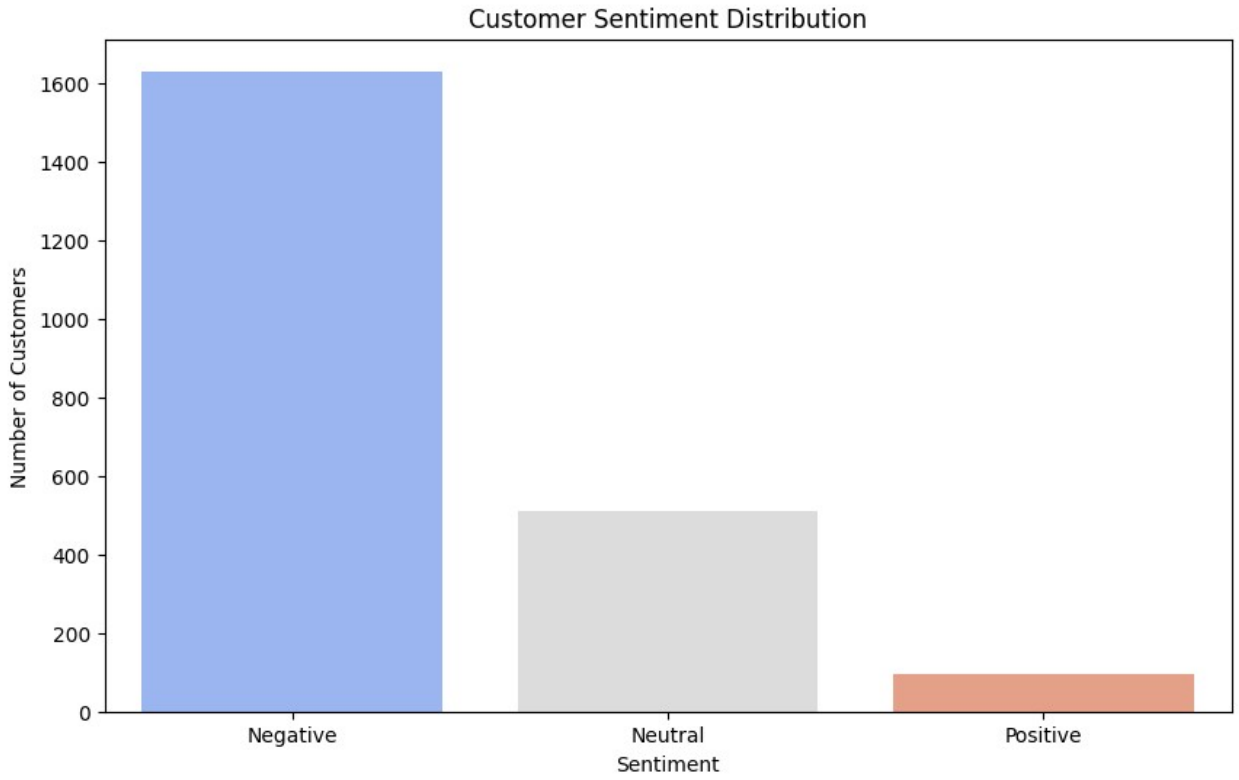
Interpretation:

The analysis reveals a predominantly negative sentiment, with a significant majority of the sentiments falling into the negative category.

Visualization of Sentiment Distribution

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sentiment Distribution Plot
plt.figure(figsize=(10, 6))
sns.barplot(x=sentiment_distribution.index,
y=sentiment_distribution.values, palette='coolwarm')
plt.title('Customer Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Number of Customers')
plt.show()
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\843916327.py:6:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=sentiment_distribution.index,
y=sentiment_distribution.values, palette='coolwarm')
```




```
# Ensure both datasets have a common identifier
data_segmentation = data_segmentation[['Cluster']]
data_combined = pd.concat([data, data_segmentation], axis=1)

# Verify the merge
print(data_combined.columns)
```

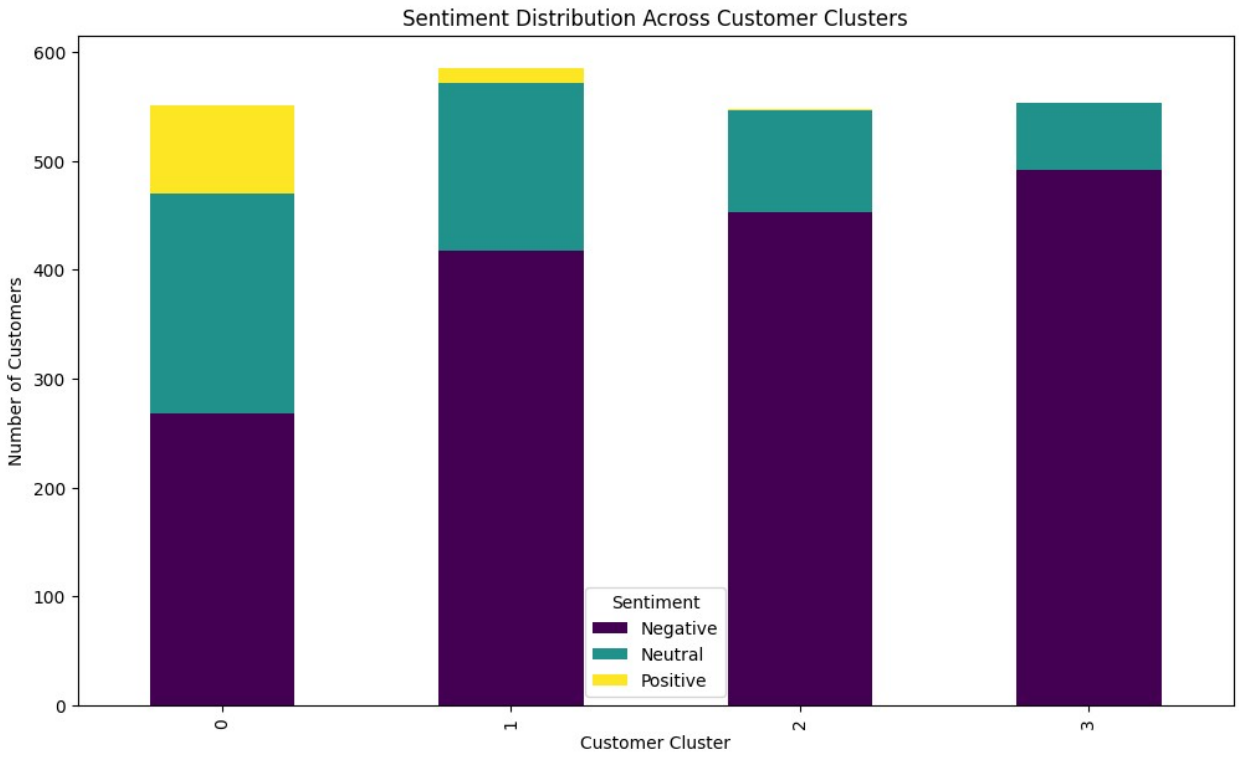
```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income',
'Kidhome',
       'Teenhome', 'Dt_Customer', 'Recency', 'MntWines', 'MntFruits',
       'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases',
'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Revenue',
'Response',
       'Customer_Since', 'Sentiment_Score', 'Sentiment', 'Cluster'],
      dtype='object')
```

Sentiment Analysis by Cluster:

```
# Group data by Cluster and Sentiment
sentiment_by_cluster = data_combined.groupby(['Cluster',
'Sentiment']).size().unstack(fill_value=0)
```

```
# Plot Sentiment Distribution by Cluster
sentiment_by_cluster.plot(kind='bar', stacked=True, figsize=(12, 7),
colormap='viridis')
plt.title('Sentiment Distribution Across Customer Clusters')
plt.xlabel('Customer Cluster')
plt.ylabel('Number of Customers')
plt.legend(title='Sentiment')
plt.show()
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\2276972371.py:2:
FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to
retain current behavior or observed=True to adopt the future default
and silence this warning.
  sentiment_by_cluster = data_combined.groupby(['Cluster',
'Sentiment']).size().unstack(fill_value=0)
```



Predictive Modeling

1. Churn Prediction Model

```
# Assuming 'Recency' is in days since last purchase
data_encoded['Churn'] = (data_encoded['Recency'] > 180).astype(int) #
Churn if no purchase in last 6 months
```

```
# Add the missing columns to data_encoded
# Use the original data_segmentation or rfm DataFrame that contains
'Frequency'
# Assuming 'data_segmentation' was originally used for calculating
'Frequency' before being reassigned
data_encoded['Frequency'] = data['NumWebPurchases'] +
data['NumCatalogPurchases'] + data['NumStorePurchases']
data_encoded['Sentiment_Score'] = data['Sentiment_Score']
data_encoded['Cluster'] = data_segmentation['Cluster']  # Assuming
data_segmentation now only has 'Cluster'

# Now, select features for X
features = ['MntTotal', 'Recency', 'Frequency', 'Customer_Age',
'Income',
            'Sentiment_Score', 'Cluster', 'Education_Postgraduate',
            'Education_Undergraduate', 'Marital_Status_Married',
            'Marital_Status_Other', 'Marital_Status_Single']
X = data_encoded[features]
y = data_encoded['Churn']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


data_encoded['Churn'] = (data_encoded['Recency'] > 90).astype(int)

# Check the distribution again:
print(data_encoded['Churn'].value_counts())


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

# Now, select features for X
features = ['MntTotal', 'Recency', 'Frequency', 'Customer_Age',
'Income',
            'Sentiment_Score', 'Cluster', 'Education_Postgraduate',
            'Education_Undergraduate', 'Marital_Status_Married',
            'Marital_Status_Other', 'Marital_Status_Single']
X = data_encoded[features]
y = data_encoded['Churn']

# Apply SMOTE to oversample the minority class:
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)

# 3. Proceed with Model Training:

# Now you can proceed with splitting the data and training the model:
from sklearn.model_selection import train_test_split
```

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

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)
```

```
Churn
0    2040
1     198
Name: count, dtype: int64
```

```
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.13.0-py3-none-any.whl.metadata (8.8
kB)
Requirement already satisfied: numpy<3,>=1.24.3 in c:\users\kaina\
appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy<2,>=1.10.1 in c:\users\kaina\
appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.14.0)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in c:\users\
kaina\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.5.1)
Collecting sklearn-compat<1,>=0.1 (from imbalanced-learn)
  Downloading sklearn_compat-0.1.3-py3-none-any.whl.metadata (18 kB)
Requirement already satisfied: joblib<2,>=1.1.1 in c:\users\kaina\
appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in c:\users\
kaina\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (3.5.0)
Downloading imbalanced_learn-0.13.0-py3-none-any.whl (238 kB)
   ---------------------------------------- 0.0/238.4 kB ? eta -:--:--
   - -------------------------------------- 10.2/238.4 kB ? eta
-:--:--
   ---------- ---------------------------- 71.7/238.4 kB 787.7 kB/s
eta 0:00:01
   ------------------------ --------------- 143.4/238.4 kB 1.2 MB/s
eta 0:00:01
   ---------------------------------------  235.5/238.4 kB 1.4 MB/s
eta 0:00:01
   ---------------------------------------- 238.4/238.4 kB 1.2 MB/s
eta 0:00:00
Downloading sklearn_compat-0.1.3-py3-none-any.whl (18 kB)
Installing collected packages: sklearn-compat, imbalanced-learn
Successfully installed imbalanced-learn-0.13.0 sklearn-compat-0.1.3


[notice] A new release of pip is available: 24.0 -> 24.3.1
```

```
[notice] To update, run: python.exe -m pip install --upgrade pip
c:\Users\kaina\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

LogisticRegression(random_state=42)
```

Addressing Churn in Customer Segmentation Analysis

This section outlines the steps taken to address the imbalanced class distribution in the churn prediction task:

1. Identifying Churn: A new binary variable 'Churn' is created. Customers with recency exceeding 90 days (considered inactive) are assigned a churn value of 1, while active customers are assigned 0.

2. Verifying Imbalance: The distribution of the 'Churn' variable is checked to confirm the class imbalance, where the number of churning customers might be significantly lower than non-churning customers.

3. SMOTE Oversampling: SMOTE (Synthetic Minority Over-sampling Technique) is an oversampling technique that creates synthetic samples for the minority class to address class imbalance.

4. Feature Selection: Relevant features such as customer demographics, purchase behavior, sentiment score, and cluster membership are selected for model training.

5. Train-Test Split: The data is split into training and testing sets for model training and evaluation.

6. Logistic Regression Model: A Logistic Regression model is chosen for churn prediction, suitable for binary classification tasks.

By addressing the class imbalance and following these steps, the churn prediction model can be trained on a more balanced dataset, potentially leading to improved accuracy in identifying customers at risk of churn.

- **Non-churning customers (0)** are heavily overrepresented with 2040 instances.
- **Churning customers (1)** are underrepresented with only 198 instances.

This imbalance poses a challenge for machine learning models as they may be biased towards the majority class (non-churning customers) and struggle to accurately predict the minority class (churning customers).

Addressing Class Imbalance:

To address this issue, techniques like SMOTE (Synthetic Minority Over-sampling Technique) can be used to generate synthetic samples for the minority class (churning customers), thereby balancing the class distribution and improving model performance.

```
from collections import Counter

# ... (Previous code for data preparation, SMOTE oversampling, and
feature selection) ...

# Train-Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train Logistic Regression Model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Make Predictions
y_pred = model.predict(X_test)

# Evaluate Model Performance
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
print("AUC-ROC:", roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.9154411764705882
Precision: 0.8805620608899297
Recall: 0.9543147208121827
F1-score: 0.9159561510353228
AUC-ROC: 0.9167308201217312

c:\Users\kaina\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

Predict Purchase Behavior

```
import pandas as pd
# data_combined = pd.concat([data, data_segmentation], axis=1)

# Verify target variable
print(data_combined['Response'].value_counts())

Response
0    1904
1     334
Name: count, dtype: int64

# Install imbalanced-learn if not already installed
!pip install imbalanced-learn

from imblearn.over_sampling import SMOTE
import pandas as pd

# Define features and target
features = ['MntTotal', 'Recency', 'Frequency', 'Customer_Age',
'Income',
            'Sentiment_Score', 'Cluster', 'Education_Postgraduate',
            'Education_Undergraduate', 'Marital_Status_Married',
            'Marital_Status_Other', 'Marital_Status_Single']

# Instead of using data_combined directly, select features from
data_encoded
# and then merge with the 'Cluster' column from data_segmentation or
data_combined.
X = data_encoded[features]
# Assuming data_combined has the 'Cluster' and 'Response' columns
y = data_combined['Response']

# Check class distribution
print("Before SMOTE:")
print(y.value_counts())

# Split the data before applying SMOTE to avoid data leakage
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize the features
```

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

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

# Check class distribution after SMOTE
print("After SMOTE:")
print(pd.Series(y_train_resampled).value_counts())
```

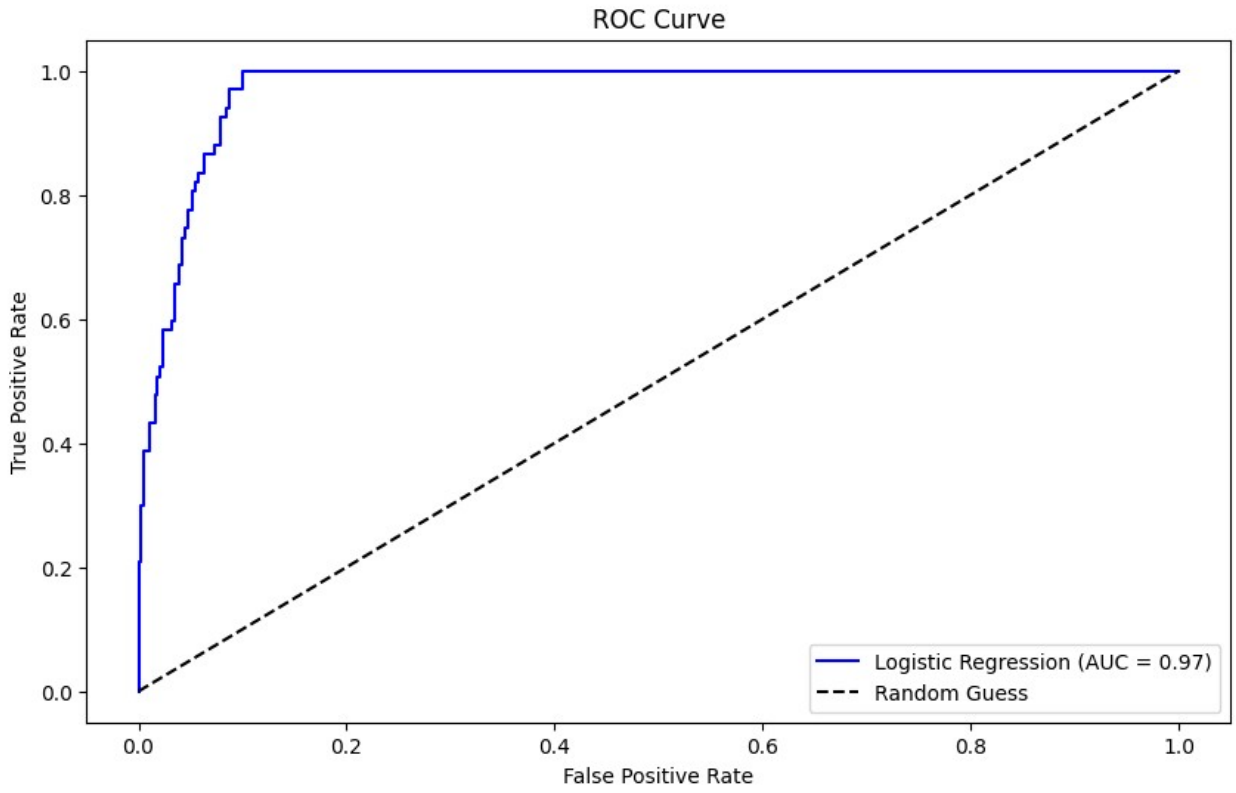
```
Requirement already satisfied: imbalanced-learn in c:\users\kaina\
appdata\local\programs\python\python312\lib\site-packages (0.13.0)
Requirement already satisfied: numpy<3,>=1.24.3 in c:\users\kaina\
appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy<2,>=1.10.1 in c:\users\kaina\
appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.14.0)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in c:\users\
kaina\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.5.1)
Requirement already satisfied: sklearn-compat<1,>=0.1 in c:\users\
kaina\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (0.1.3)
Requirement already satisfied: joblib<2,>=1.1.1 in c:\users\kaina\
appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in c:\users\
kaina\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (3.5.0)
Before SMOTE:
Response
0    1904
1     334
Name: count, dtype: int64
After SMOTE:
Response
0    1523
1    1523
Name: count, dtype: int64


[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
log_reg = LogisticRegression(random_state=42, solver='liblinear',
max_iter=1000)

# Train the model on the resampled training data
log_reg.fit(X_train_resampled, y_train_resampled)
```

```
LogisticRegression(max_iter=1000, random_state=42, solver='liblinear')
```

```
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, roc_curve, accuracy_score

# Make predictions on the test set
y_pred = log_reg.predict(X_test_scaled)
y_proba = log_reg.predict_proba(X_test_scaled)[:, 1]

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Classification Report
cr = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(cr)

# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# ROC AUC Score
roc_auc = roc_auc_score(y_test, y_proba)
print(f"ROC AUC Score: {roc_auc:.2f}")

# Plot ROC Curve
import matplotlib.pyplot as plt
import seaborn as sns

fpr, tpr, thresholds = roc_curve(y_test, y_proba)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc:.2f})',
color='blue')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

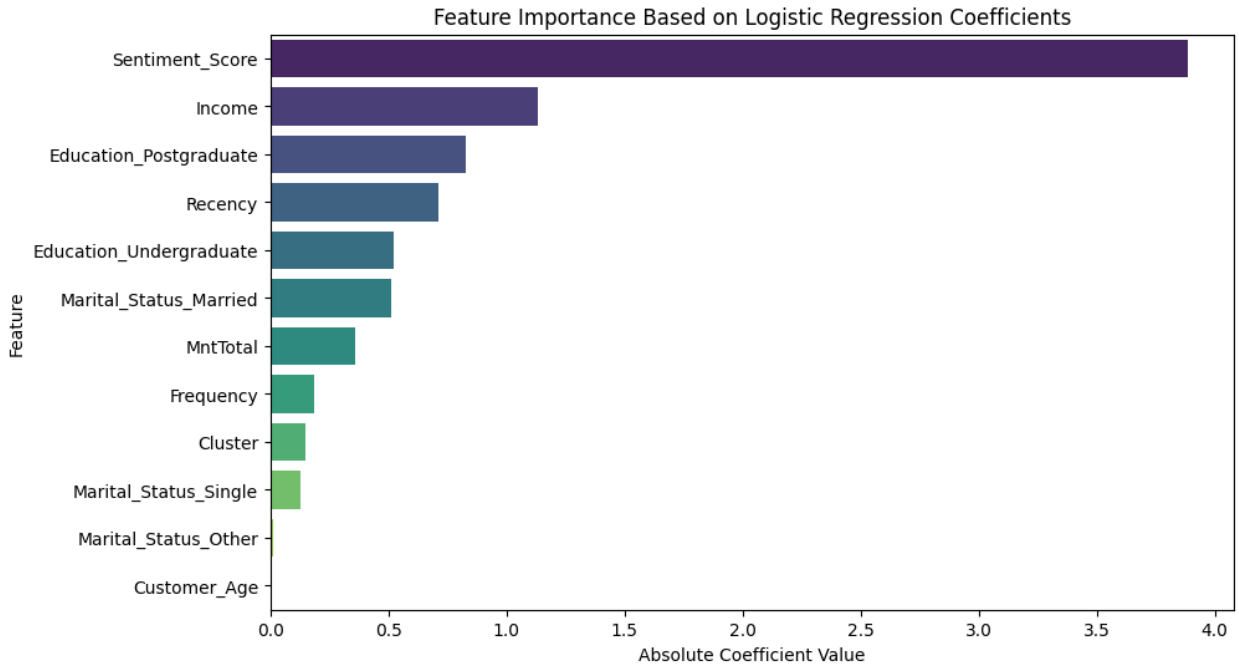
```
Confusion Matrix:
[[348  33]
 [  3  64]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.91      0.95       381
           1       0.66      0.96      0.78        67

    accuracy                           0.92       448
   macro avg       0.83      0.93      0.87       448
weighted avg       0.94      0.92      0.93       448

Accuracy: 0.92
ROC AUC Score: 0.97
```




```python
import numpy as np

# Extract feature names and their corresponding coefficients
feature_names = X.columns
coefficients = log_reg.coef_[0]

# Create a DataFrame for better visualization
feature_importance = pd.DataFrame({
```

```
    'Feature': feature_names,
    'Coefficient': coefficients
})

# Calculate absolute values for importance
feature_importance['Absolute_Coefficient'] =
feature_importance['Coefficient'].abs()

# Sort by absolute coefficient
feature_importance =
feature_importance.sort_values(by='Absolute_Coefficient',
ascending=False)

# Plot Feature Importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Absolute_Coefficient', y='Feature',
data=feature_importance, palette='viridis')
plt.title('Feature Importance Based on Logistic Regression
Coefficients')
plt.xlabel('Absolute Coefficient Value')
plt.ylabel('Feature')
plt.show()
```

```
C:\Users\kaina\AppData\Local\Temp\ipykernel_16736\2540169956.py:21:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='Absolute_Coefficient', y='Feature',
data=feature_importance, palette='viridis')
```



Feature Importance for Churn Prediction

Key Features:

- **Sentiment Score:** Highest impact, indicating negative sentiment strongly influences churn.
- **Income:** Lower income customers may be more prone to churn.
- **Recency:** Longer recency periods increase churn risk.

Other Influential Factors:

- **Education:** Postgraduate education may have a positive impact on retention.
- **Marital Status:** Marital status shows some influence on churn risk.

Recommendations:

- **Sentiment Analysis:** Prioritize improving customer sentiment.
- **Targeted Campaigns:** Tailor campaigns based on income and education.
- **Retention Strategies:** Focus on retaining customers with longer recency.
- **Loyalty Programs:** Incentivize frequent purchases to reduce churn risk.