Customer Segmentation Using Clustering Algorithms

`Author:` Prashant Sharma
`Date:` 11.December.2024
`Dataset:` Credit Card Customer Segmentation

About Dataset (Meta data)

Context

This dataset contains a wealth of customer information collected from within a consumer credit card portfolio, with the aim of helping analysts predict customer attrition. It includes comprehensive demographic details such as age, gender, marital status and income category, as well as insight into each customer's relationship with the credit card provider such as the card type, number of months on book and inactive periods. Additionally it holds key data about customers' spending behavior drawing closer to their churn decision such as total revolving balance, credit limit, average open to buy rate and analyzable metrics like total amount of change from quarter 4 to quarter 1, average utilization ratio and Naive Bayes classifier attrition flag (Card category is combined with contacts count in 12months period alongside dependent count plus education level & months inactive). Faced with this set of useful predicted data points across multiple variables capture up-to-date information that can determine long term account stability or an impending departure therefore offering us an equipped understanding when seeking to manage a portfolio or serve individual customers.

Content

Column Descriptions:

- `CLIENTNUM:` Unique identifier for each customer. (Integer).
- `Attrition_Flag:` Flag indicating whether or not the customer has churned out. (Boolean).
- `Customer_Age:` Age of customer. (Integer).
- `Gender:` The text or lyrics that song contain.
- `Dependent_count:` Number of dependents that customer has. (Integer)
- `Education_Level:` Education level of customer. (String)
- `Marital_Status:` Marital status of customer. (String)
- `Income_Category:` Income category of customer. (String)

- `Card_Category:` Type of card held by customer. (String)
- `Months_on_book:` How long customer has been on the books. (Integer)
- `Total_Relationship_Count:` Total number of relationships customer has with the credit card provider. (Integer)
- `Months_Inactive_12_mon:` Number of months customer has been inactive in the last twelve months. (Integer)
- `Contacts_Count_12_mon:` Number of contacts customer has had in the last twelve months. (Integer)
- `Credit_Limit:` Credit limit of customer. (Integer)
- `Total_Revolving_Bal:` Total revolving balance of customer. (Integer)
- `Avg_Open_To_Buy:` Average open to buy ratio of customer. (Integer)
- `Total_Amt_Chng_Q4_Q1:` Total amount changed from quarter 4 to quarter 1. (Integer)
- `Total_Trans_Amt:` Total transaction amount. (Integer)
- `Total_Trans_Ct:` Total transaction count. (Integer)
- `Total_Ct_Chng_Q4_Q1:` Total count changed from quarter 4 to quarter 1. (Integer)
- `Avg_Utilization_Ratio:` Average utilization ratio of customer. (Integer)
- `Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Cou` Naive Bayes classifier for predicting whether or not someone will churn based on characteristics such

Import Libraries

In [1]:
```
# Import libraries

# Data manipulation and analysis
import numpy as np
import pandas as pd

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Machine learning models and utilities
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans,AgglomerativeClustering,DBSCAN,Spectra
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn import tree
from sklearn import metrics
```

Load the Dataset

In [2]:
```
df = pd.read_csv('Data/BankChurners.csv')

# Display the first 10 rows of the dataset
df.head(10)
```

Out[2]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Educatio
0	768805383	Existing Customer	45	M	3	Higl
1	818770008	Existing Customer	49	F	5	(
2	713982108	Existing Customer	51	M	3	(
3	769911858	Existing Customer	40	F	4	Higl
4	709106358	Existing Customer	40	M	3	Une
5	713061558	Existing Customer	44	M	2	(
6	810347208	Existing Customer	51	M	4	l
7	818906208	Existing Customer	32	M	0	Higl
8	710930508	Existing Customer	37	M	3	Une
9	719661558	Existing Customer	48	M	2	(

10 rows × 23 columns

Data Preprocessing

In [3]:
```
# Explore the data types and non-null counts for each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 23 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   CLIENTNUM
10127 non-null  int64
 1   Attrition_Flag
10127 non-null  object
 2   Customer_Age
10127 non-null  int64
 3   Gender
10127 non-null  object
 4   Dependent_count
10127 non-null  int64
 5   Education_Level
10127 non-null  object
 6   Marital_Status
10127 non-null  object
 7   Income_Category
10127 non-null  object
 8   Card_Category
10127 non-null  object
 9   Months_on_book
10127 non-null  int64
 10  Total_Relationship_Count
10127 non-null  int64
 11  Months_Inactive_12_mon
10127 non-null  int64
 12  Contacts_Count_12_mon
10127 non-null  int64
 13  Credit_Limit
10127 non-null  float64
 14  Total_Revolving_Bal
10127 non-null  int64
 15  Avg_Open_To_Buy
10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1
10127 non-null  float64
 17  Total_Trans_Amt
10127 non-null  int64
 18  Total_Trans_Ct
10127 non-null  int64
 19  Total_Ct_Chng_Q4_Q1
10127 non-null  float64
 20  Avg_Utilization_Ratio
10127 non-null  float64
 21  Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12
_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1  10127 non-n
ull  float64
 22  Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12
_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2  10127 non-n
ull  float64
dtypes: float64(7), int64(10), object(6)
memory usage: 1.8+ MB
```

There are some categorical columns too in the dataset and most of the columns are of numerical data

- Categorical columns: Attrition_Flag, Gender, Education_Level, Marital_Status, Income_Category, Card_Category

In [4]:
```
#Checking the data shape
df.shape
```

Out[4]:
```
(10127, 23)
```

In [5]:
```
# Check for missing values in the dataset
df.isnull().sum()
```

Out[5]:
```
CLIENTNUM
0
Attrition_Flag
0
Customer_Age
0
Gender
0
Dependent_count
0
Education_Level
0
Marital_Status
0
Income_Category
0
Card_Category
0
Months_on_book
0
Total_Relationship_Count
0
Months_Inactive_12_mon
0
Contacts_Count_12_mon
0
Credit_Limit
0
Total_Revolving_Bal
0
Avg_Open_To_Buy
0
Total_Amt_Chng_Q4_Q1
0
Total_Trans_Amt
0
Total_Trans_Ct
0
Total_Ct_Chng_Q4_Q1
0
Avg_Utilization_Ratio
0
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mo
n_Dependent_count_Education_Level_Months_Inactive_12_mon_1     0
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mo
n_Dependent_count_Education_Level_Months_Inactive_12_mon_2     0
dtype: int64
```

There are no missing values in the dataset

In [6]:
```
# Check NaN values in the entire dataset
nan_values = df.isna().sum()
print("NaN values in each column:\n", nan_values)
```

```
NaN values in each column:
 CLIENTNUM
0
Attrition_Flag
0
Customer_Age
0
Gender
0
Dependent_count
0
Education_Level
0
Marital_Status
0
Income_Category
0
Card_Category
0
Months_on_book
0
Total_Relationship_Count
0
Months_Inactive_12_mon
0
Contacts_Count_12_mon
0
Credit_Limit
0
Total_Revolving_Bal
0
Avg_Open_To_Buy
0
Total_Amt_Chng_Q4_Q1
0
Total_Trans_Amt
0
Total_Trans_Ct
0
Total_Ct_Chng_Q4_Q1
0
Avg_Utilization_Ratio
0
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_
Dependent_count_Education_Level_Months_Inactive_12_mon_1    0
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_
Dependent_count_Education_Level_Months_Inactive_12_mon_2    0
dtype: int64
```

There are no nan values in the dataset

In [7]:

```python
# Summary statistics of numerical columns
df.describe()
```

Out[7]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Re
count	1.012700e+04	10127.000000	10127.000000	10127.000000	
mean	7.391776e+08	46.325960	2.346203	35.928409	
std	3.690378e+07	8.016814	1.298908	7.986416	
min	7.080821e+08	26.000000	0.000000	13.000000	
25%	7.130368e+08	41.000000	1.000000	31.000000	
50%	7.179264e+08	46.000000	2.000000	36.000000	
75%	7.731435e+08	52.000000	3.000000	40.000000	
max	8.283431e+08	73.000000	5.000000	56.000000	

In [8]:

```
# Check unique values in categorical columns
print("Attrition_Flag:", df['Attrition_Flag'].unique())
print("Gender:", df['Gender'].unique())
print("Education_Level:", df['Education_Level'].unique())
print("Marital_Status:", df['Marital_Status'].unique())
print("Income_Category:", df['Income_Category'].unique())
print("Card_Category:", df['Card_Category'].unique())
```

```
Attrition_Flag: ['Existing Customer' 'Attrited Customer']
Gender: ['M' 'F']
Education_Level: ['High School' 'Graduate' 'Uneducated' 'Unknown' 'College
' 'Post-Graduate'
 'Doctorate']
Marital_Status: ['Married' 'Single' 'Unknown' 'Divorced']
Income_Category: ['$60K - $80K' 'Less than $40K' '$80K - $120K' '$40K - $6
0K' '$120K +'
 'Unknown']
Card_Category: ['Blue' 'Gold' 'Silver' 'Platinum']
```

In [9]:

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

Out[9]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Educatio
0	768805383	Existing Customer	45	M	3	Higl
1	818770008	Existing Customer	49	F	5	(
2	713982108	Existing Customer	51	M	3	(
3	769911858	Existing Customer	40	F	4	Higl
4	709106358	Existing Customer	40	M	3	Une
5	713061558	Existing Customer	44	M	2	(
6	810347208	Existing Customer	51	M	4	l
7	818906208	Existing Customer	32	M	0	Higl
8	710930508	Existing Customer	37	M	3	Une
9	719661558	Existing Customer	48	M	2	(

10 rows × 23 columns

In [10]:

```
df = df.drop(['Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Conta
```

There was no use of these columns for customer segmentation so we have dropped it

In [11]:

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

Out[11]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Mari
0	Existing Customer	45	M	3	High School	
1	Existing Customer	49	F	5	Graduate	
2	Existing Customer	51	M	3	Graduate	
3	Existing Customer	40	F	4	High School	
4	Existing Customer	40	M	3	Uneducated	
5	Existing Customer	44	M	2	Graduate	
6	Existing Customer	51	M	4	Unknown	
7	Existing Customer	32	M	0	High School	
8	Existing Customer	37	M	3	Uneducated	
9	Existing Customer	48	M	2	Graduate	

Exploratory Data Analysis (EDA)

In [12]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Attrition_Flag            10127 non-null  object 
 1   Customer_Age              10127 non-null  int64  
 2   Gender                    10127 non-null  object 
 3   Dependent_count           10127 non-null  int64  
 4   Education_Level           10127 non-null  object 
 5   Marital_Status            10127 non-null  object 
 6   Income_Category           10127 non-null  object 
 7   Card_Category             10127 non-null  object 
 8   Months_on_book            10127 non-null  int64  
 9   Total_Relationship_Count  10127 non-null  int64  
 10  Months_Inactive_12_mon    10127 non-null  int64  
 11  Contacts_Count_12_mon     10127 non-null  int64  
 12  Credit_Limit              10127 non-null  float64
 13  Total_Revolving_Bal       10127 non-null  int64  
 14  Avg_Open_To_Buy           10127 non-null  float64
 15  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 16  Total_Trans_Amt           10127 non-null  int64  
 17  Total_Trans_Ct            10127 non-null  int64  
 18  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 19  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(9), object(6)
memory usage: 1.5+ MB
```

Analysis of Attrition_Flag Column

In [13]:

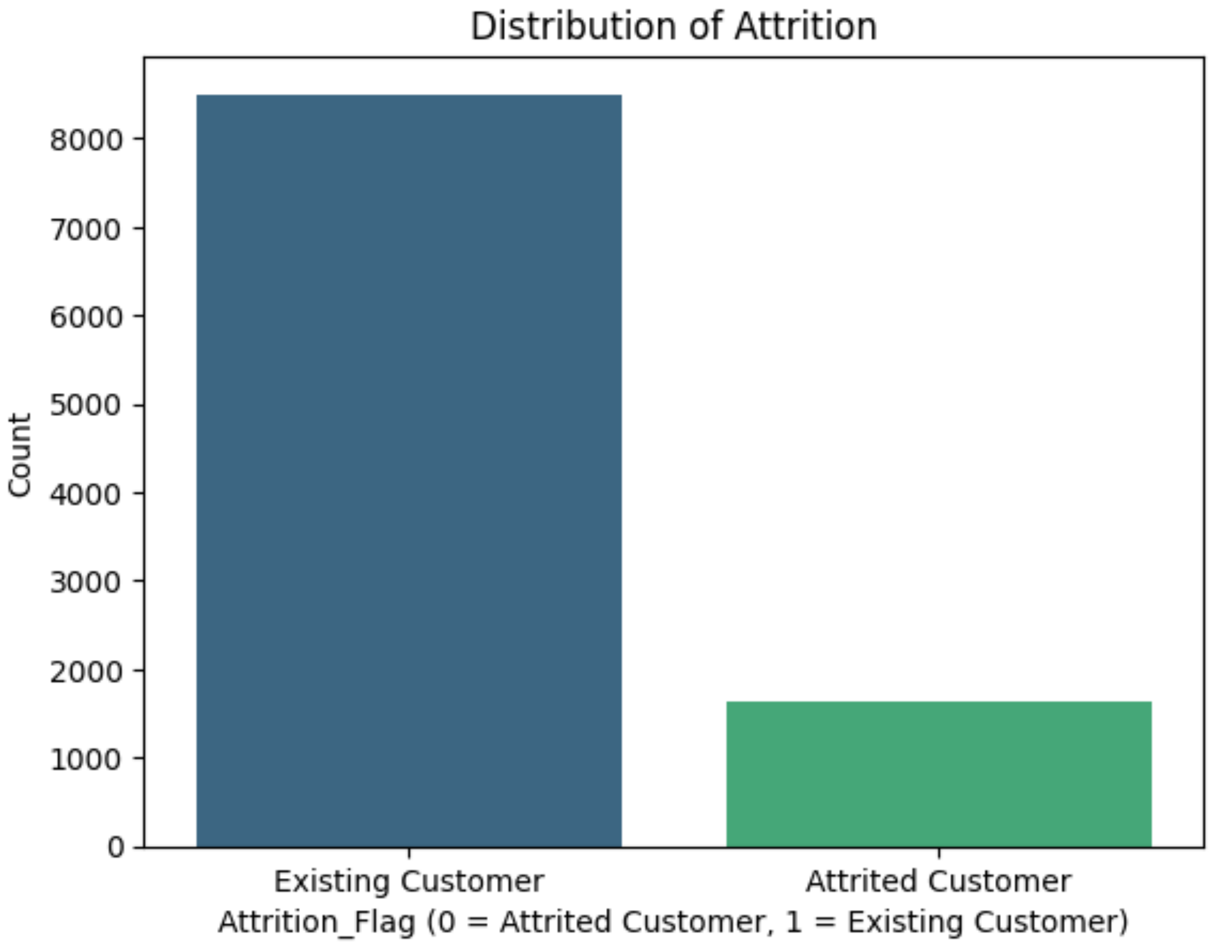
```
# Find the values of atrrition_flag column
df['Attrition_Flag'].value_counts()
```

Out[13]:

```
Attrition_Flag
Existing Customer    8500
Attrited Customer    1627
Name: count, dtype: int64
```

In [14]:

```
sns.countplot(x='Attrition_Flag', hue='Attrition_Flag', data=df, palette=
plt.title('Distribution of Attrition')
plt.xlabel('Attrition_Flag (0 = Attrited Customer, 1 = Existing Customer
plt.ylabel('Count')
plt.show()
```



In [15]:
```
# calculating the percentage fo Existing Customer and Attrited Customer

Existing_count = 8500
Attrited_count = 1627

total_count = Existing_count + Attrited_count

# calculate percentages
Existing_percentage = (Existing_count/total_count)*100
Attrited_percentages = (Attrited_count/total_count)*100

# display the results
print(f'Existing percentage in the data: {Existing_percentage:.2f}%')
print(f'Attrited percentage in the data: {Attrited_percentages:.2f}%')
```

```
Existing percentage in the data: 83.93%
Attrited percentage in the data: 16.07%
```

Existing Customers are way more than Attrited Customers

Analysis of Customer_Age Column

In [16]:
```
print('Age Summary Statistics:')
df['Customer_Age'].describe()
```

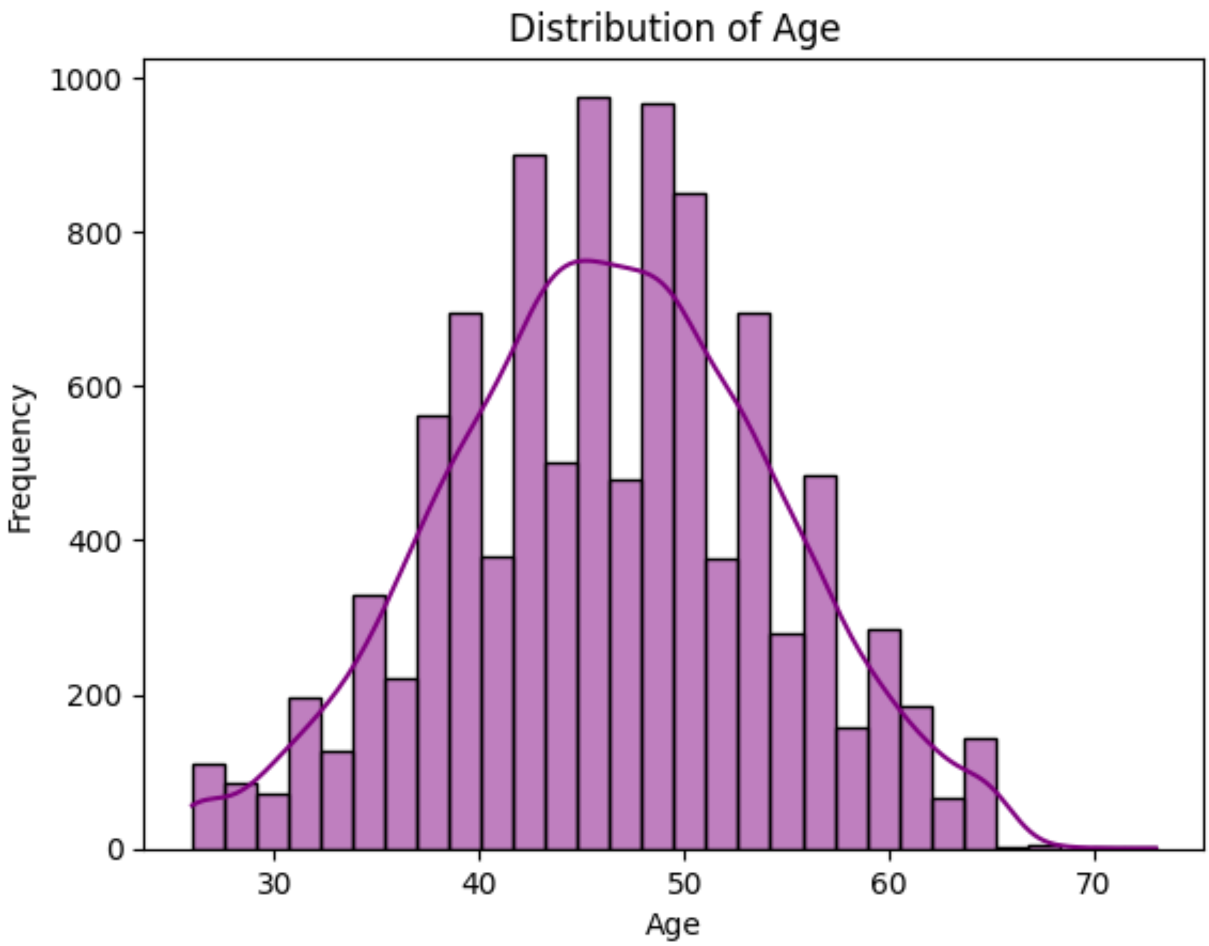
```
Age Summary Statistics:
```

Out[16]:
```
count    10127.000000
mean        46.325960
std          8.016814
min         26.000000
25%         41.000000
50%         46.000000
75%         52.000000
max         73.000000
Name: Customer_Age, dtype: float64
```

In [17]:
```
df['Customer_Age'].min(), df['Customer_Age'].max()
```

Out[17]:
```
(np.int64(26), np.int64(73))
```

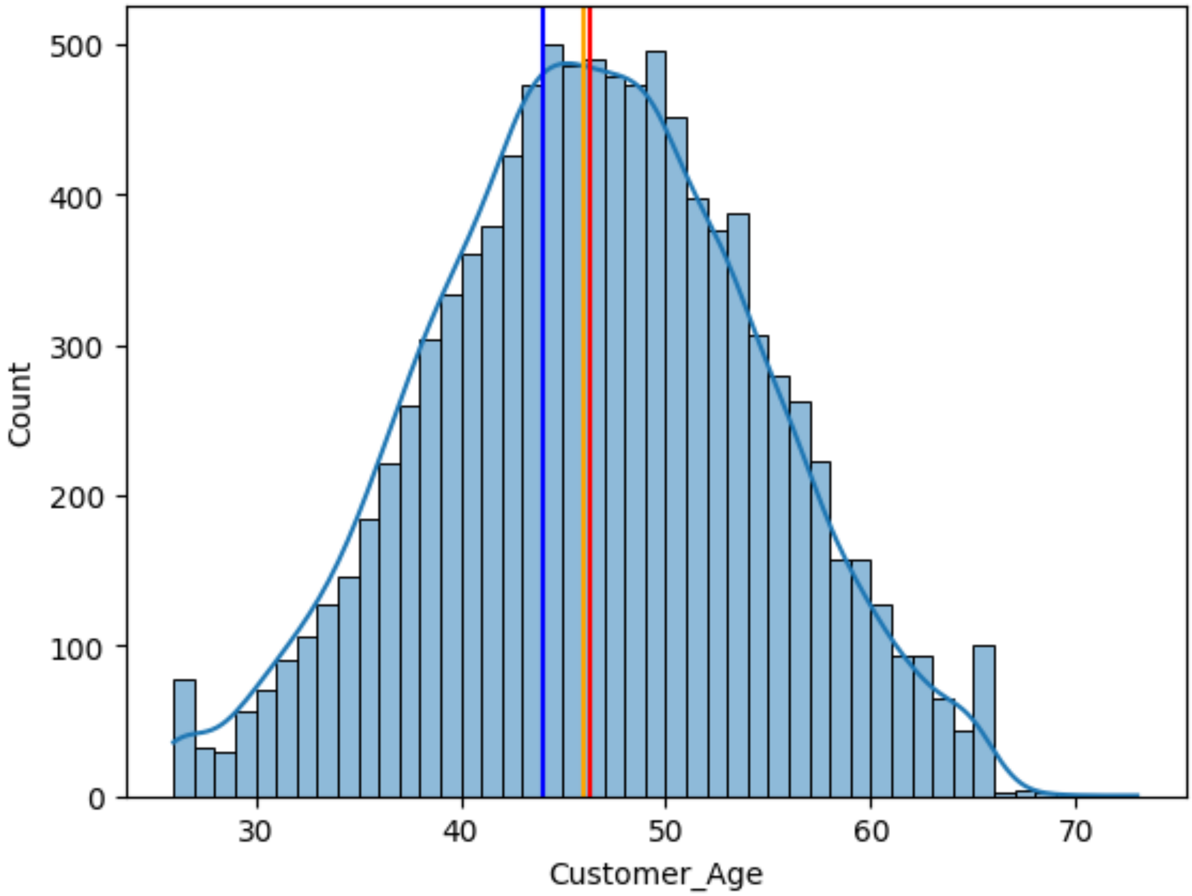
In [18]:
```
sns.histplot(df['Customer_Age'], kde=True, bins=30, color='purple')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



Age Coloumn seems to be normally distributed

In [19]:
```
sns.histplot(df['Customer_Age'], kde=True)
plt.axvline(df['Customer_Age'].mean(), color='Red')
plt.axvline(df['Customer_Age'].median(), color= 'orange')
plt.axvline(df['Customer_Age'].mode()[0], color='Blue')

# print the value of mean, median and mode of age column
print('Mean', df['Customer_Age'].mean())
print('Median', df['Customer_Age'].median())
print('Mode', df['Customer_Age'].mode())
```

```
Mean 46.32596030413745
Median 46.0
Mode 0    44
Name: Customer_Age, dtype: int64
```



The Age Column has a central tendency

Exploring Gender Column

In [20]:
```
# Find the values of sex column
df['Gender'].value_counts()
```

Out[20]:
```
Gender
F    5358
M    4769
Name: count, dtype: int64
```

In [21]:
```
sns.countplot(x='Gender',hue='Gender', data=df, palette='viridis', legend
plt.title('Distribution of Gender')
plt.xlabel('Gender (0 = Female, 1 = Male)')
plt.ylabel('Count')
plt.show()
```

In [22]:
```
# calculating the percentage fo male and female value counts in the data

male_count = 4769
female_count = 5358

total_count = male_count + female_count

# calculate percentages
male_percentage = (male_count/total_count)*100
female_percentages = (female_count/total_count)*100

# display the results
print(f'Male percentage in the data: {male_percentage:.2f}%')
print(f'Female percentage in the data: {female_percentages:.2f}%')
```

```
Male percentage in the data: 47.09%
Female percentage in the data: 52.91%
```

Females are more than 50% in the dataset

Exploring Education_Level Column

In [23]:

```python
# Find count of enducation_level column

df['Education_Level'].value_counts()
```

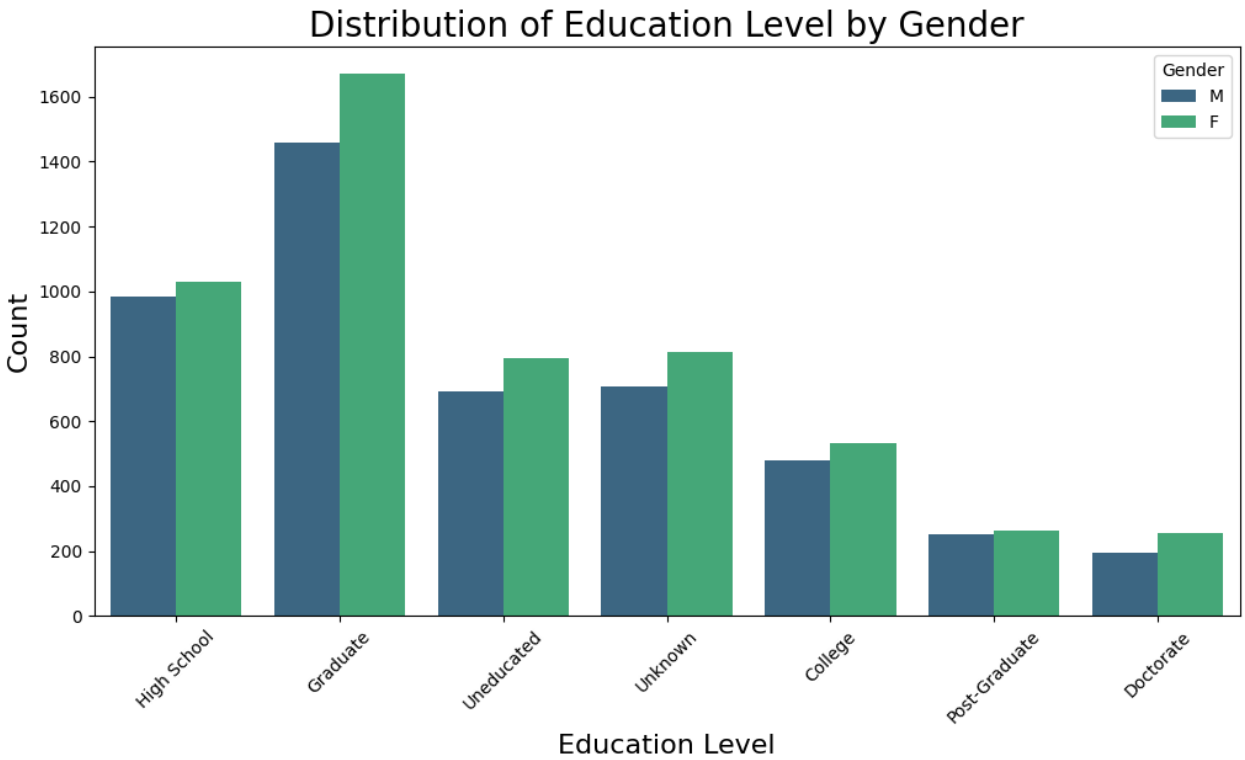
Out[23]:

```
Education_Level
Graduate         3128
High School      2013
Unknown          1519
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: count, dtype: int64
```

Most of the users are Graduates

In [24]:

```python
plt.figure(figsize=(12, 6))

sns.countplot(
    x='Education_Level',
    data=df,
    hue='Gender',
    palette='viridis',
    dodge=True
)
plt.title('Distribution of Education Level by Gender', fontsize=20)
plt.xlabel('Education Level', fontsize=16)
plt.ylabel('Count', fontsize=16)
plt.xticks(rotation=45)
plt.show()
```



At every education level females are more compare to males

Exploring Marital_Status Column

In [25]:

```
# Find the count of marital status column

df['Marital_Status'].value_counts()
```

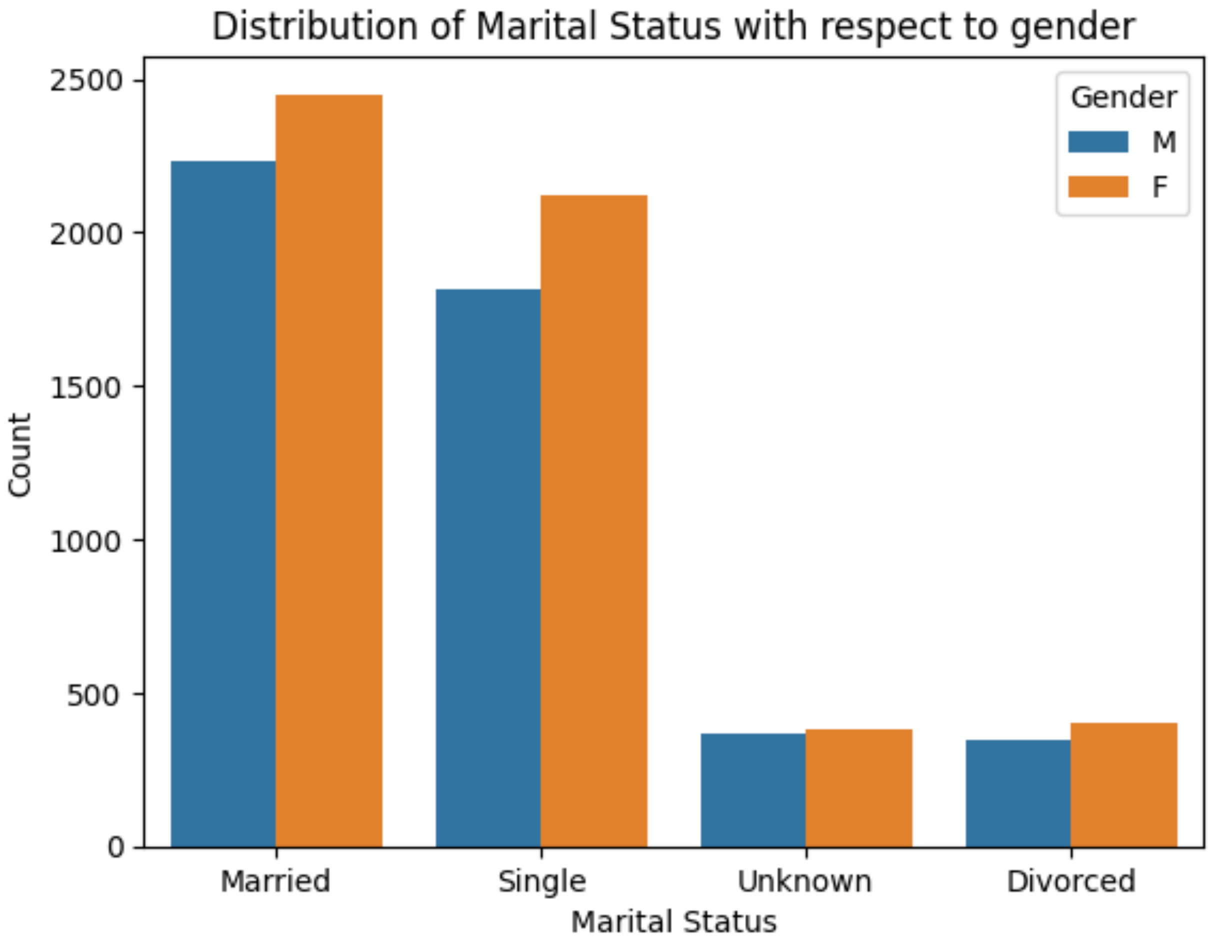
Out[25]:

```
Marital_Status
Married     4687
Single      3943
Unknown      749
Divorced     748
Name: count, dtype: int64
```

Most of the users are married followed by single

In [26]:

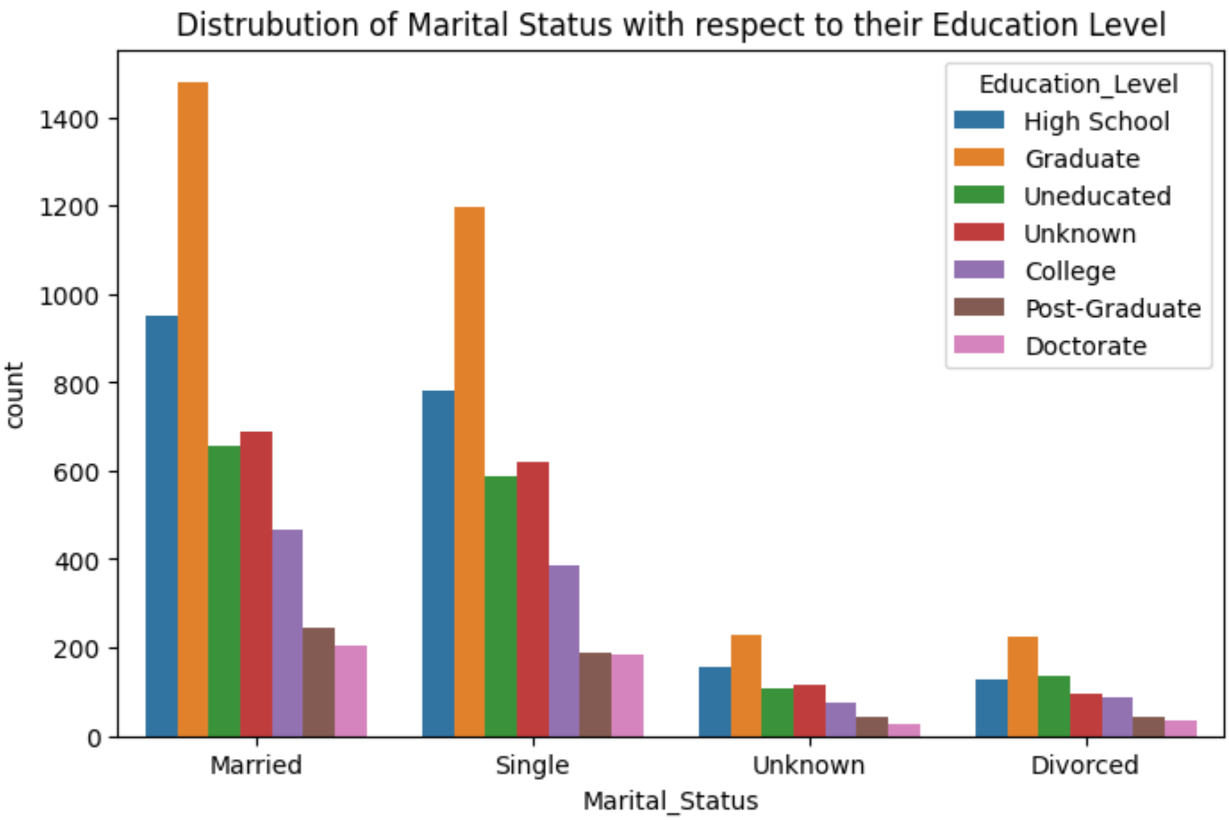
```
sns.countplot(x='Marital_Status', hue='Gender', data=df, dodge=True)
plt.title('Distribution of Marital Status with respect to gender')
plt.xlabel('Marital Status')
plt.ylabel('Count')
plt.show()
```



Again most the Married users are females

In [27]:
```
plt.figure(figsize=(8,5))
sns.countplot(x='Marital_Status', data=df, hue='Education_Level', dodge='
plt.title('Distrubution of Marital Status with respect to their Educatio
```

Out[27]:
```
Text(0.5, 1.0, 'Distrubution of Marital Status with respect to their Edu
cation Level')
```



Graduate Users are higher at every marital status, most of them are married followed by single

Analysis of Income Category Column

In [28]:

```
# Find the count of income category column
df['Income_Category'].value_counts()
```

Out[28]:

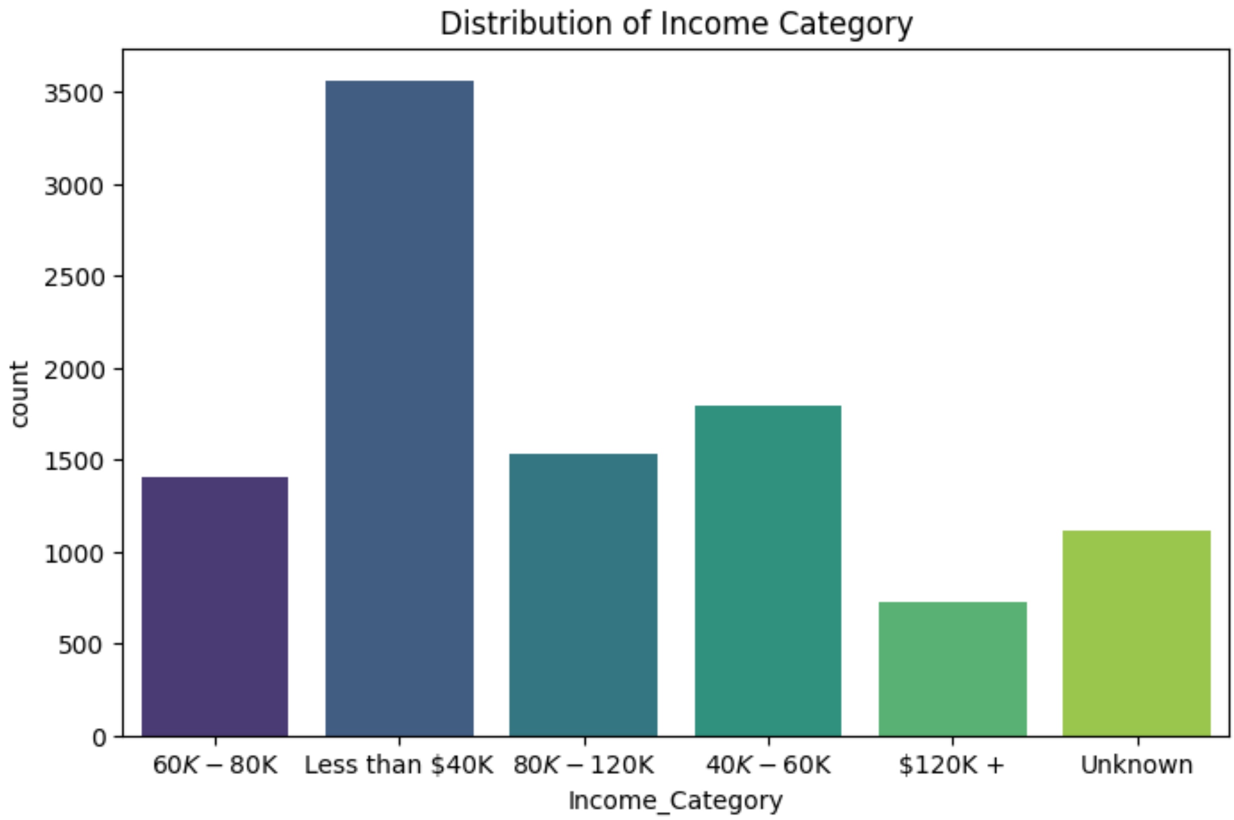
```
Income_Category
Less than $40K     3561
$40K - $60K        1790
$80K - $120K       1535
$60K - $80K        1402
Unknown            1112
$120K +             727
Name: count, dtype: int64
```

Most of the users has less than $40k income

In [29]:

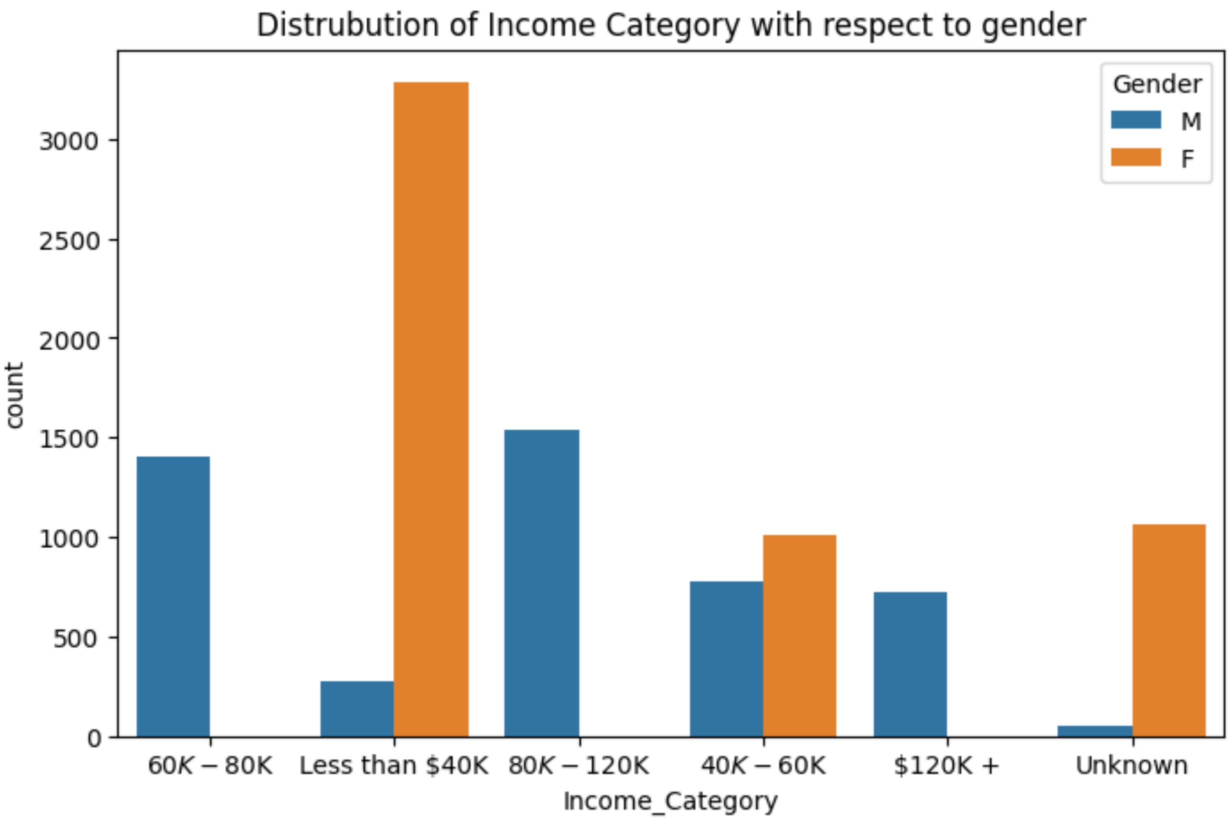
```
plt.figure(figsize=(8,5))
sns.countplot(x='Income_Category', hue='Income_Category', data=df, palet
plt.title('Distribution of Income Category')
```

Out[29]:
```
Text(0.5, 1.0, 'Distribution of Income Category')
```



In [30]:
```
plt.figure(figsize=(8,5))
sns.countplot(x='Income_Category', data=df, hue='Gender', legend=True, d
plt.title('Distrubution of Income Category with respect to gender')
```

Out[30]:
```
Text(0.5, 1.0, 'Distrubution of Income Category with respect to gender')
```



- `Insights:` Most of the females fall under less then $40kincomecategoryandmostofthemalesfallunder$80k-$120k income category

In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Attrition_Flag            10127 non-null  object 
 1   Customer_Age              10127 non-null  int64  
 2   Gender                    10127 non-null  object 
 3   Dependent_count           10127 non-null  int64  
 4   Education_Level           10127 non-null  object 
 5   Marital_Status            10127 non-null  object 
 6   Income_Category           10127 non-null  object 
 7   Card_Category             10127 non-null  object 
 8   Months_on_book            10127 non-null  int64  
 9   Total_Relationship_Count  10127 non-null  int64  
 10  Months_Inactive_12_mon    10127 non-null  int64  
 11  Contacts_Count_12_mon     10127 non-null  int64  
 12  Credit_Limit              10127 non-null  float64
 13  Total_Revolving_Bal       10127 non-null  int64  
 14  Avg_Open_To_Buy           10127 non-null  float64
 15  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 16  Total_Trans_Amt           10127 non-null  int64  
 17  Total_Trans_Ct            10127 non-null  int64  
 18  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 19  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(9), object(6)
memory usage: 1.5+ MB
```

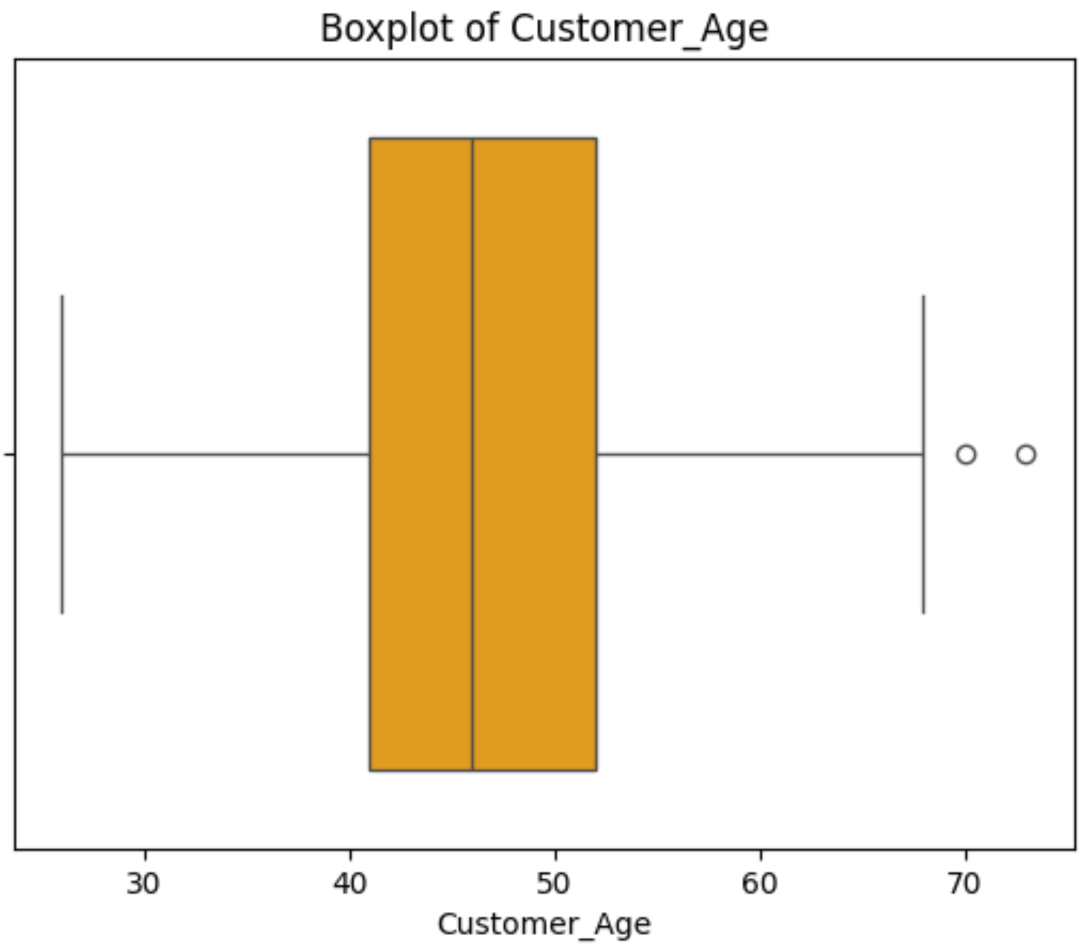
Identifying and Handling Outliers

In [32]:

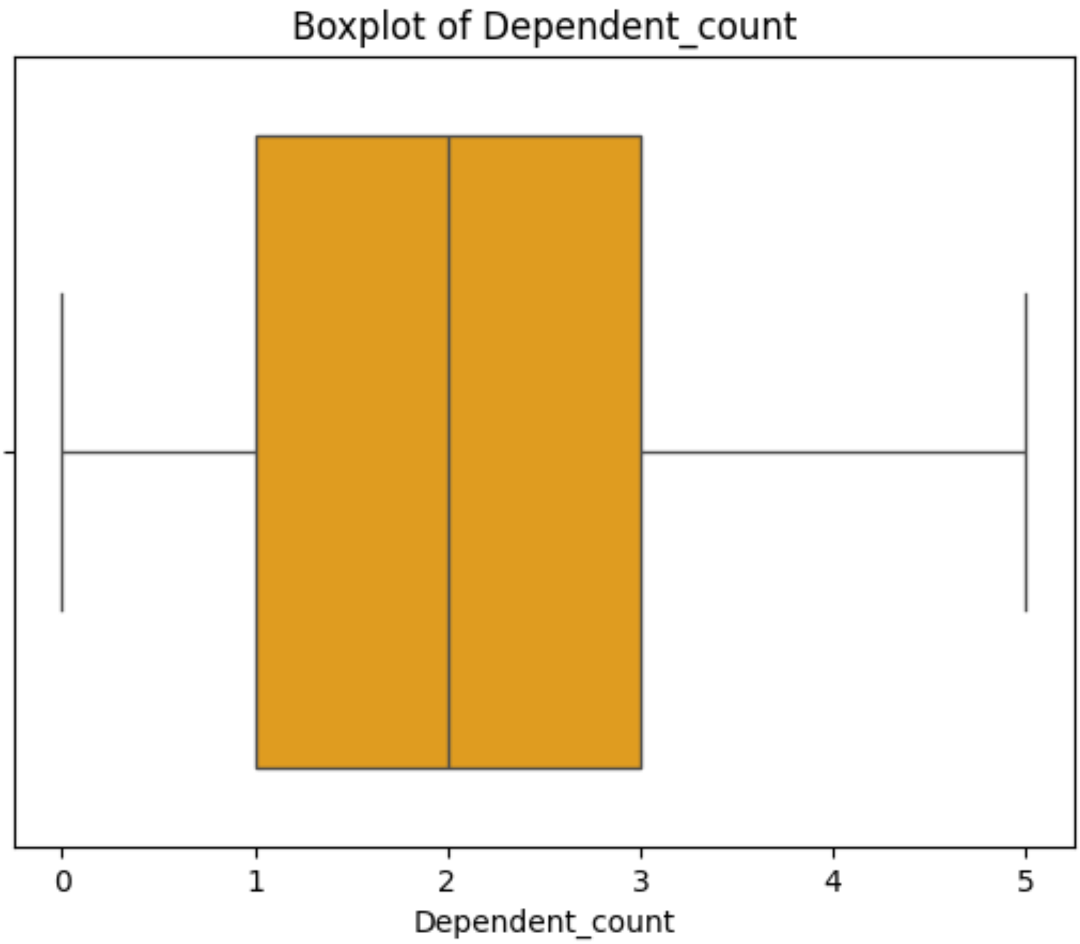
```python
# List of numerical columns
numerical_columns = ['Customer_Age', 'Dependent_count', 'Months_on_book'
for col in numerical_columns:
    # Calculate Q1, Q3, and IQR
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"Column: {col}")
    print(f"Number of outliers: {outliers.shape[0]}")

    # Visualization
    sns.boxplot(x=df[col], color='orange')
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)
    plt.show()
```

```
    # Handle outliers: Remove rows with outliers
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    print(f"After handling outliers, dataset shape: {df.shape}")
```

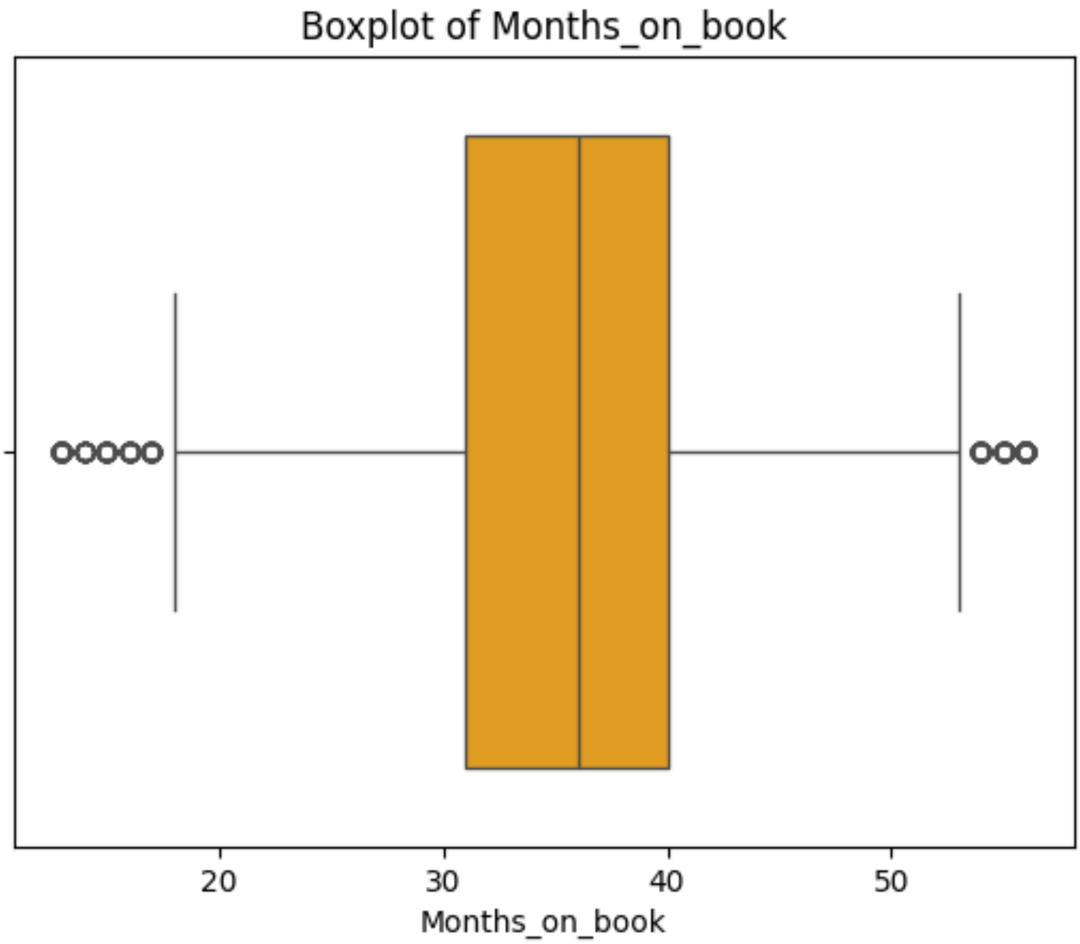
```
Column: Customer_Age
Number of outliers: 2
```



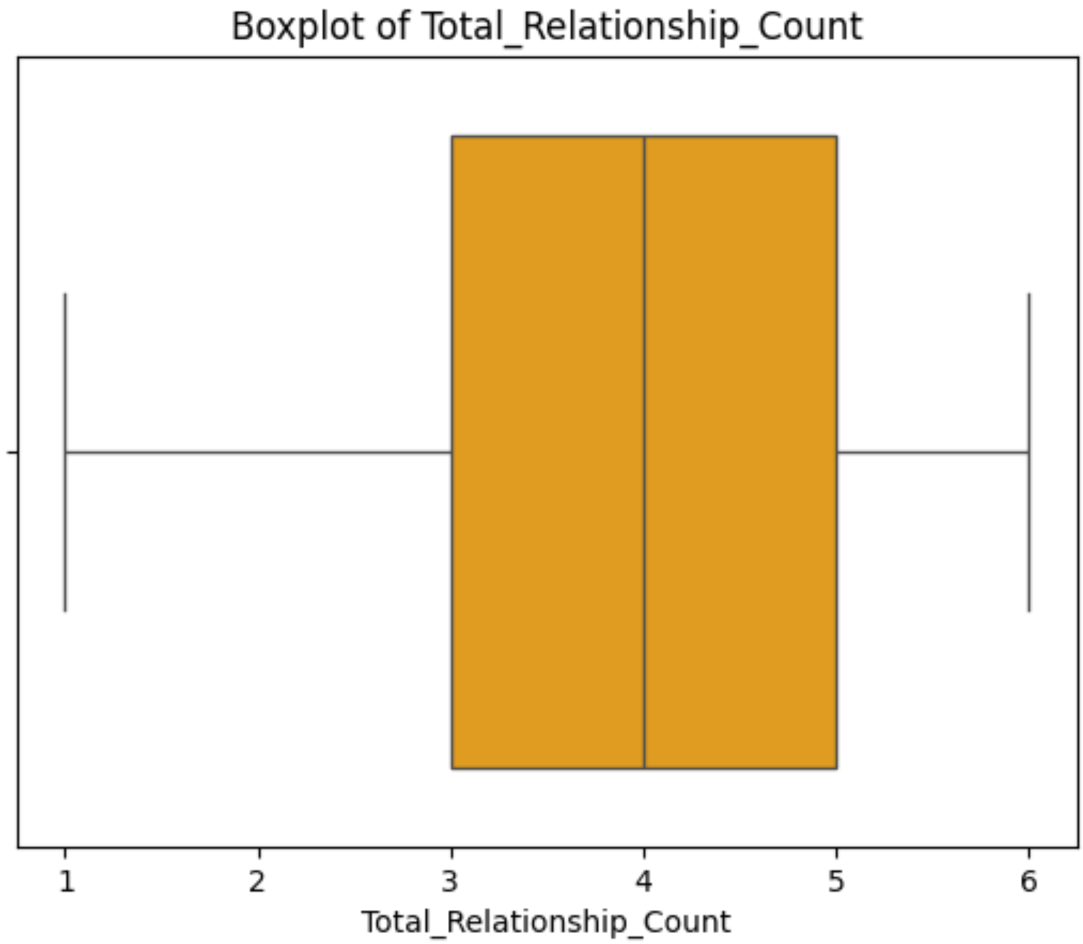

```
After handling outliers, dataset shape: (10125, 20)
Column: Dependent_count
Number of outliers: 0
```



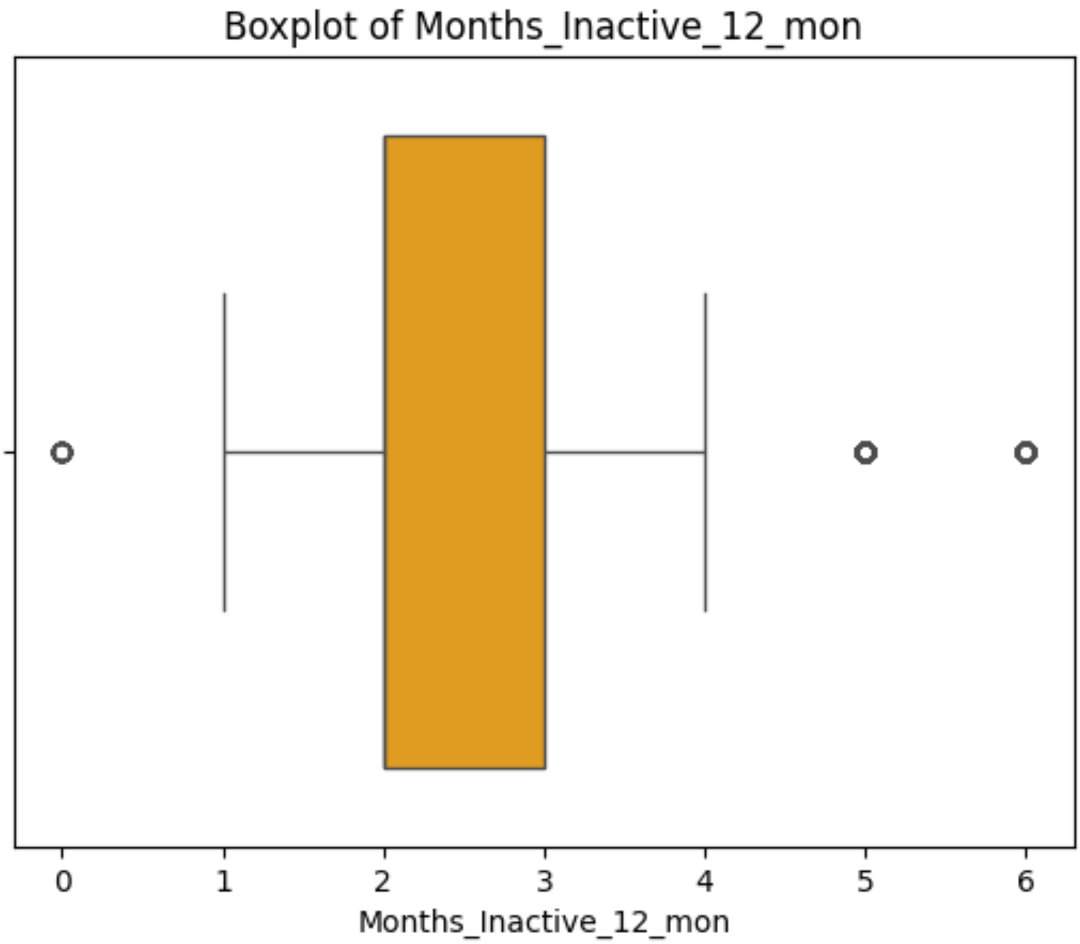

```
After handling outliers, dataset shape: (10125, 20)
Column: Months_on_book
Number of outliers: 385
```



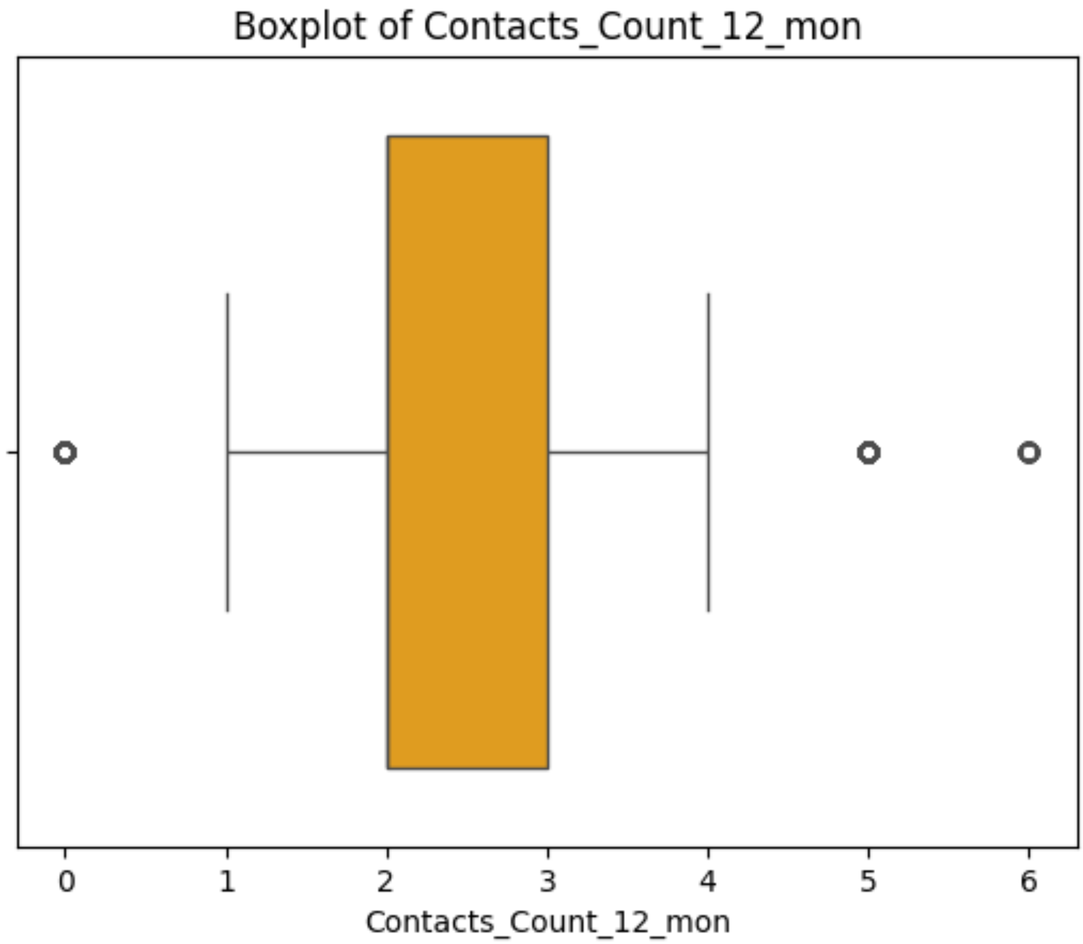

```
After handling outliers, dataset shape: (9740, 20)
Column: Total_Relationship_Count
Number of outliers: 0
```



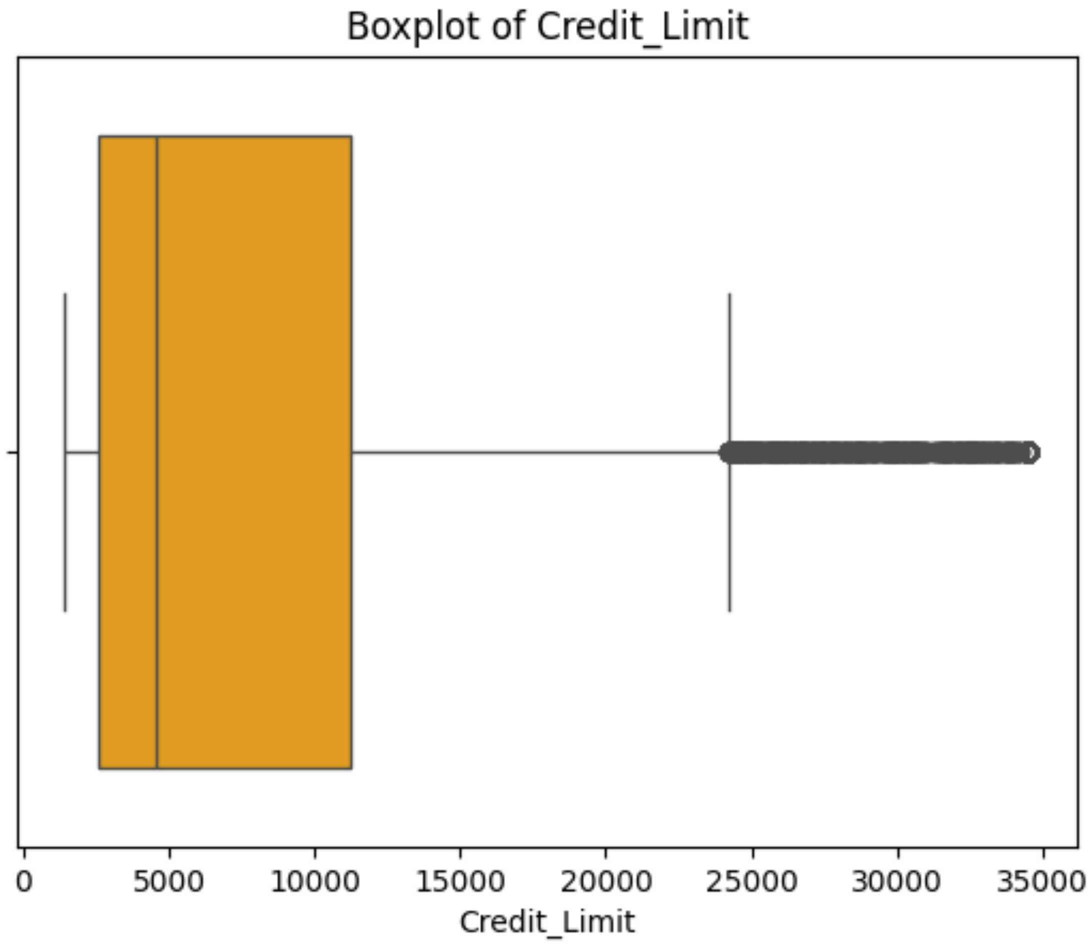

```
After handling outliers, dataset shape: (9740, 20)
Column: Months_Inactive_12_mon
Number of outliers: 308
```



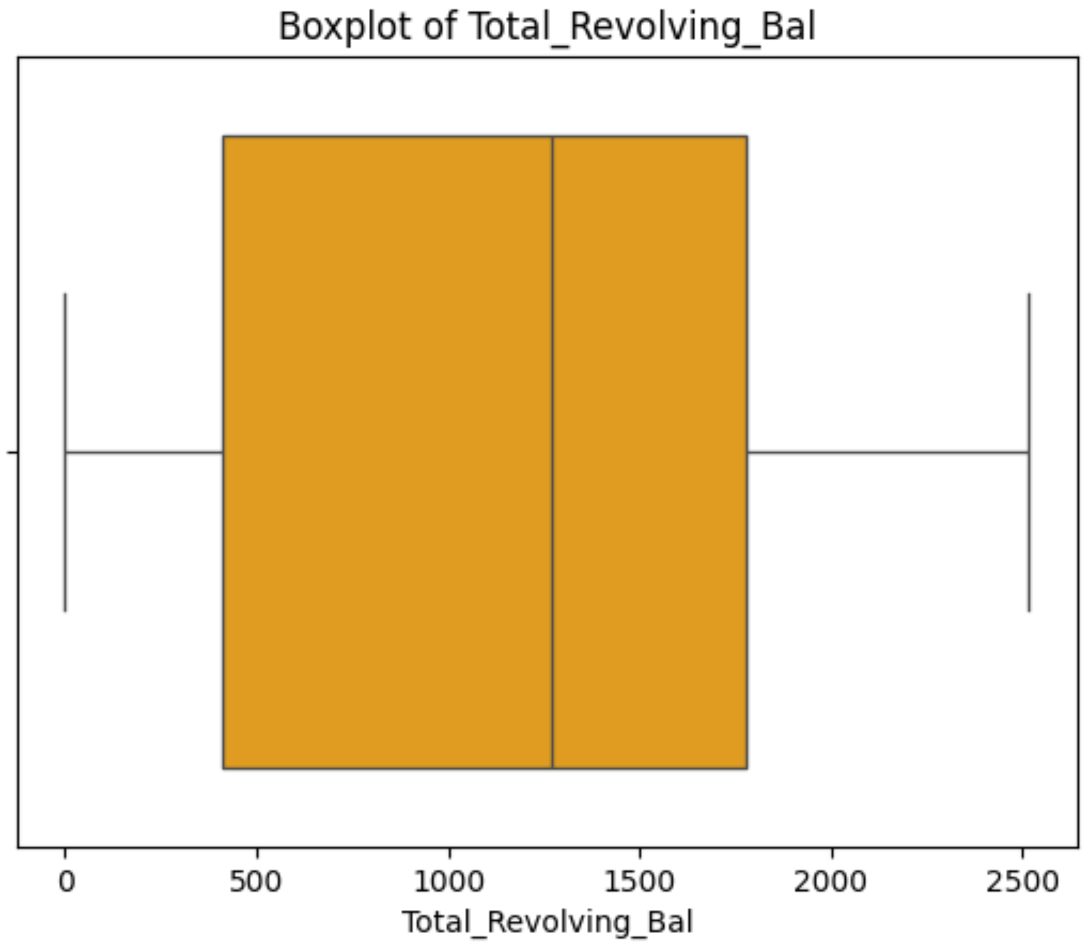

```
After handling outliers, dataset shape: (9432, 20)
Column: Contacts_Count_12_mon
Number of outliers: 584
```



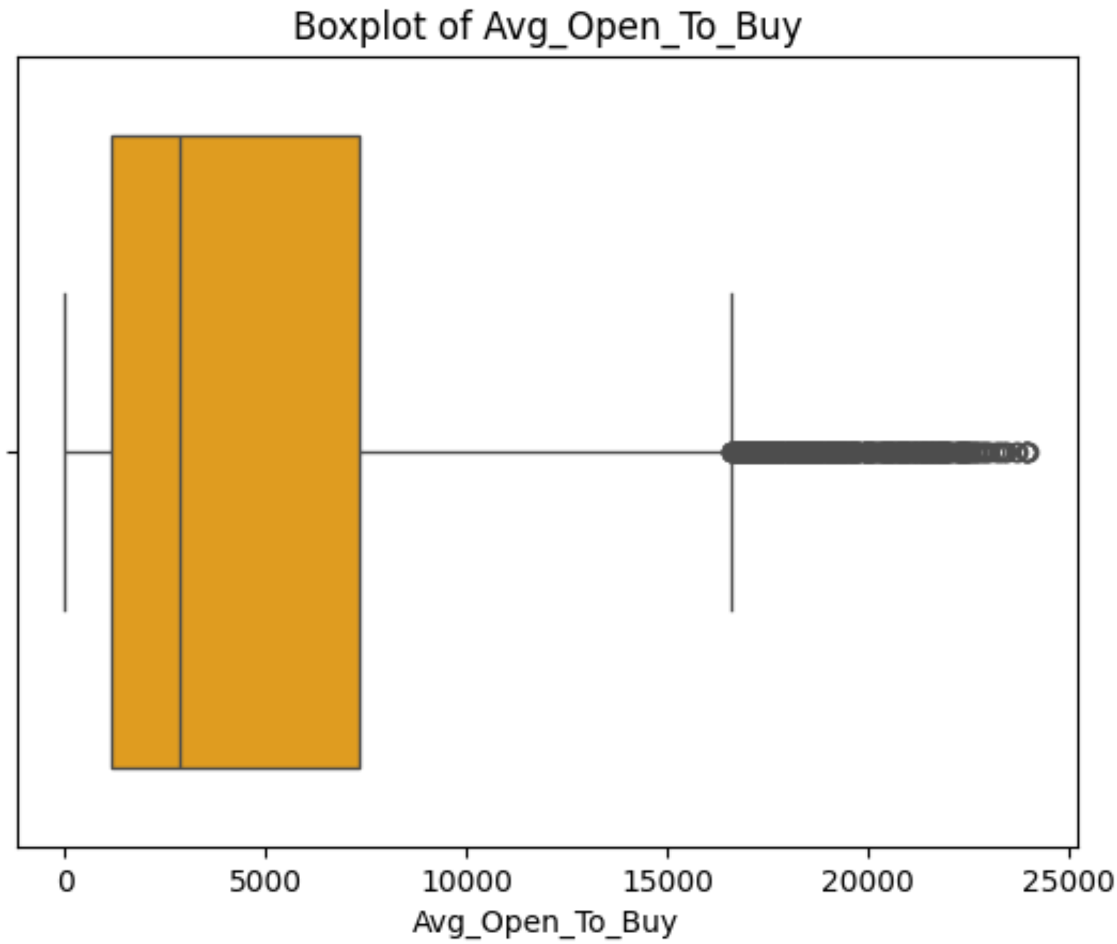

```
After handling outliers, dataset shape: (8848, 20)
Column: Credit_Limit
Number of outliers: 859
```



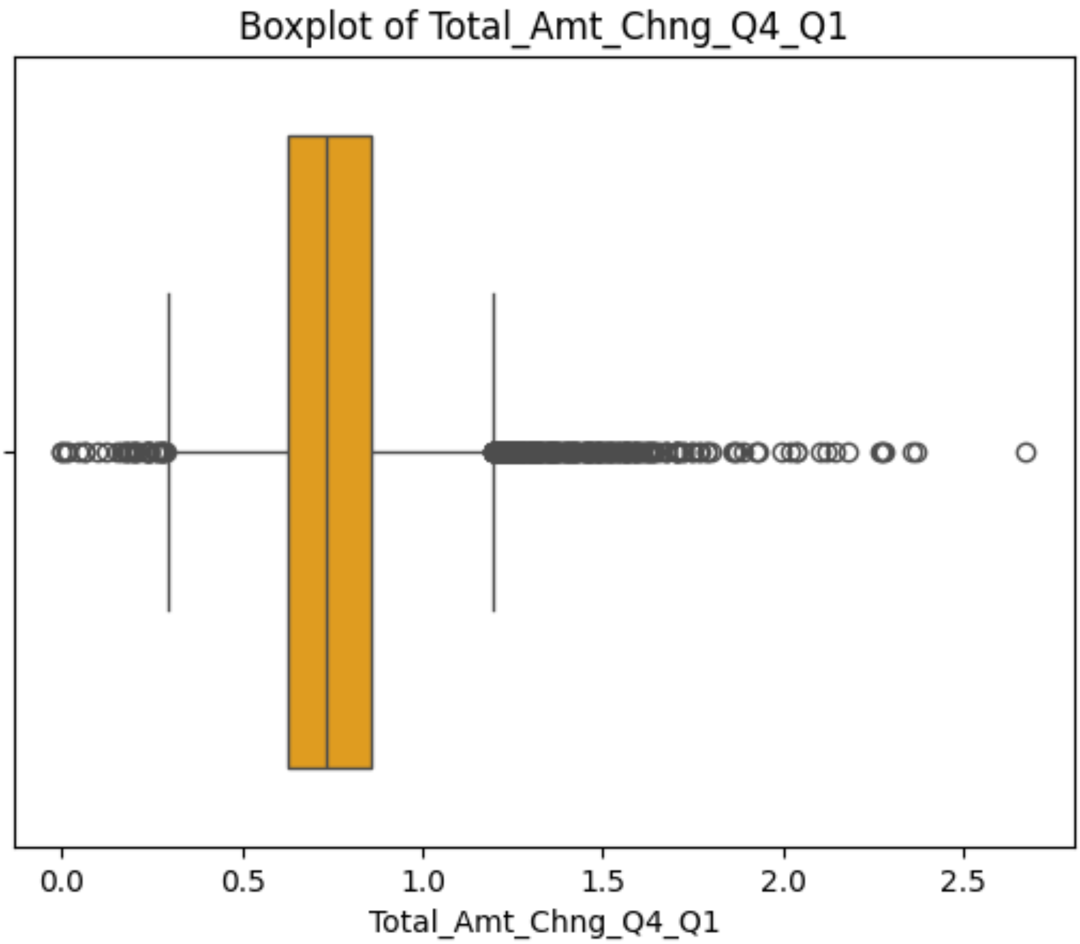

```
After handling outliers, dataset shape: (7989, 20)
Column: Total_Revolving_Bal
Number of outliers: 0
```



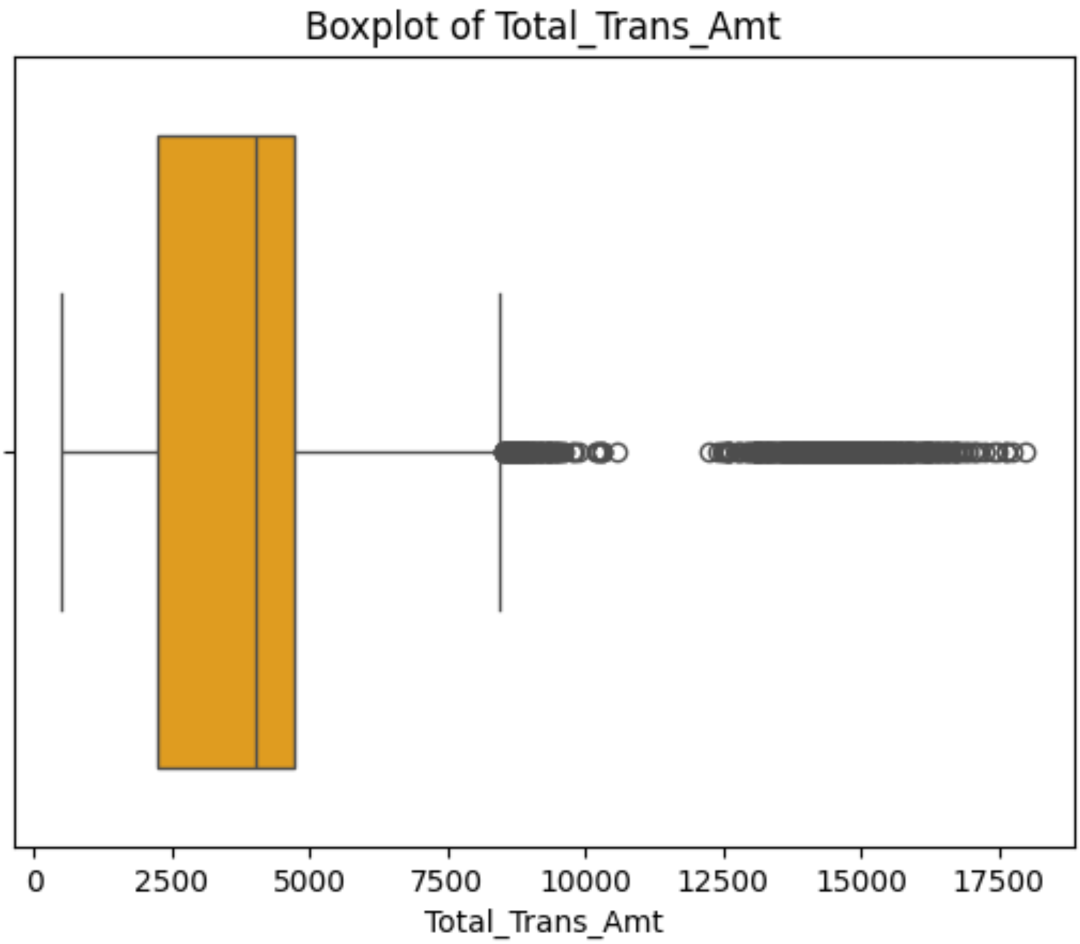

```
After handling outliers, dataset shape: (7989, 20)
Column: Avg_Open_To_Buy
Number of outliers: 467
```



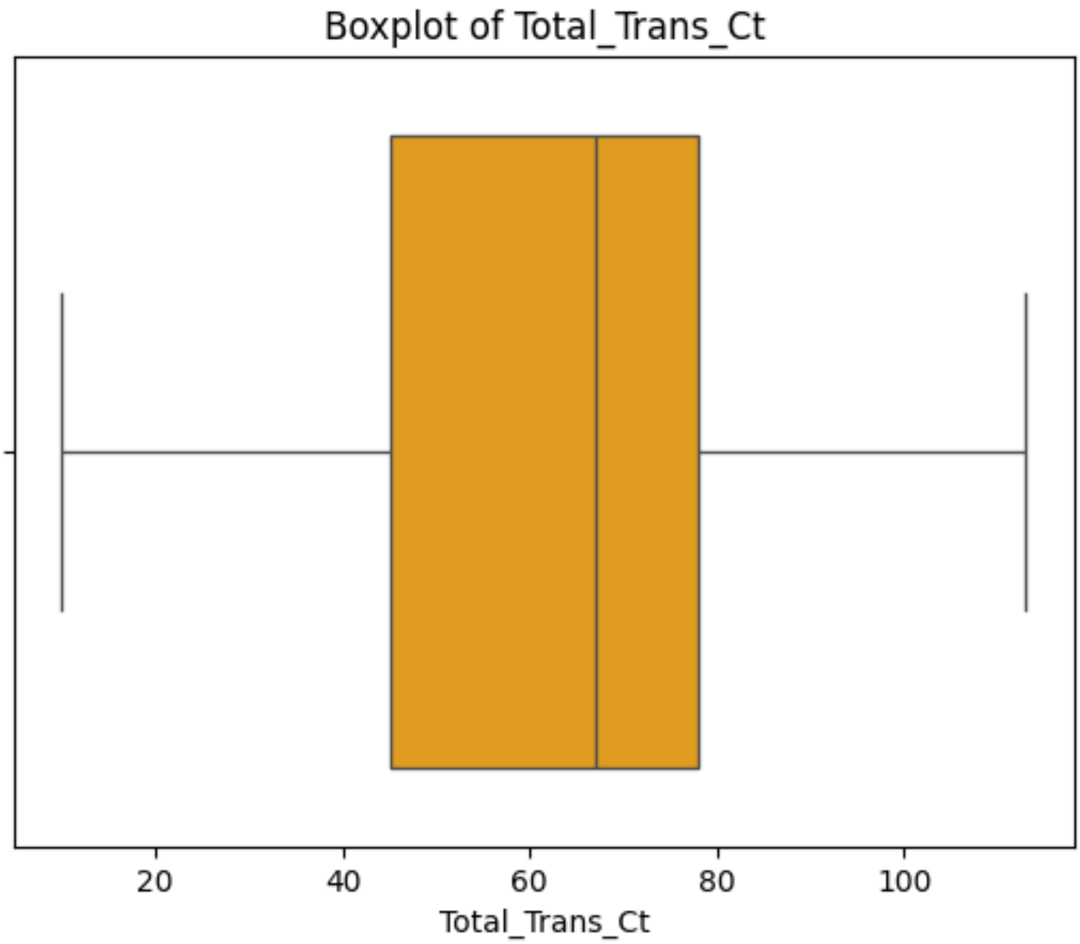

```
After handling outliers, dataset shape: (7522, 20)
Column: Total_Amt_Chng_Q4_Q1
Number of outliers: 289
```



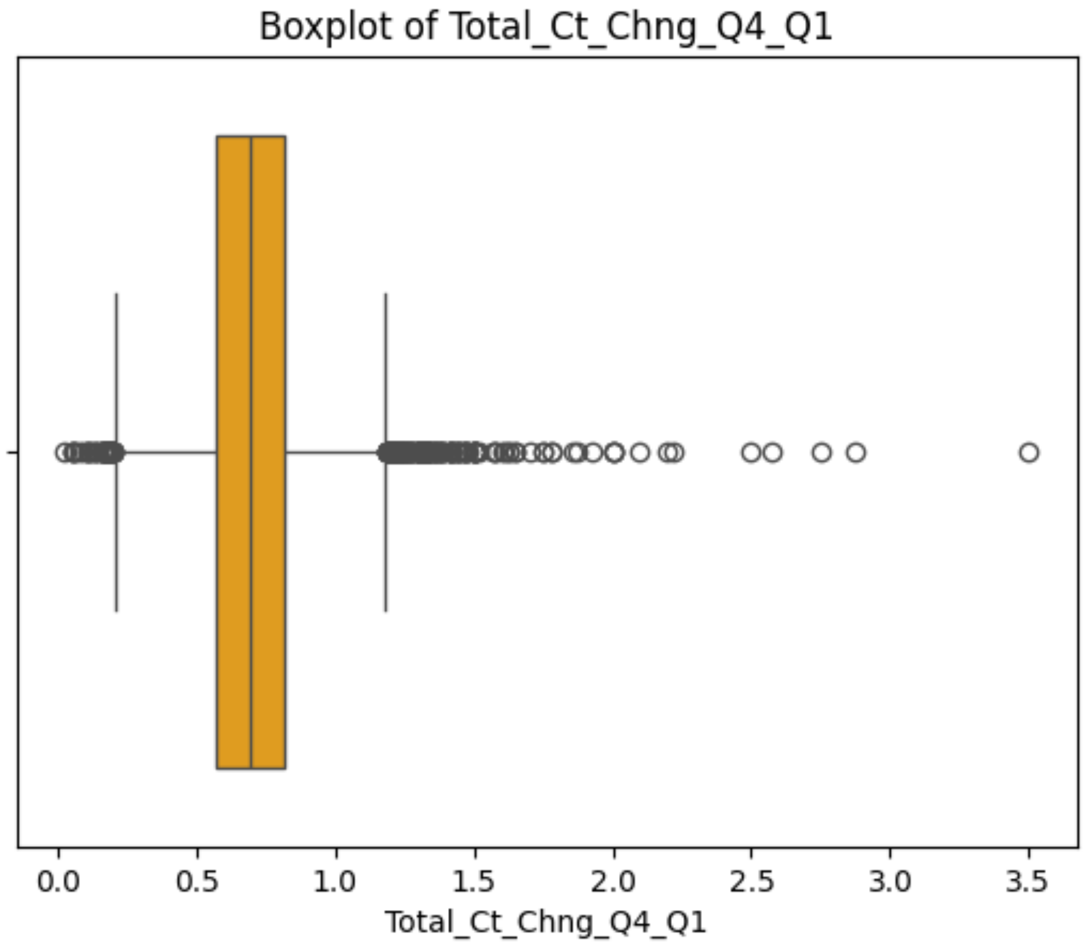

```
After handling outliers, dataset shape: (7233, 20)
Column: Total_Trans_Amt
Number of outliers: 575
```



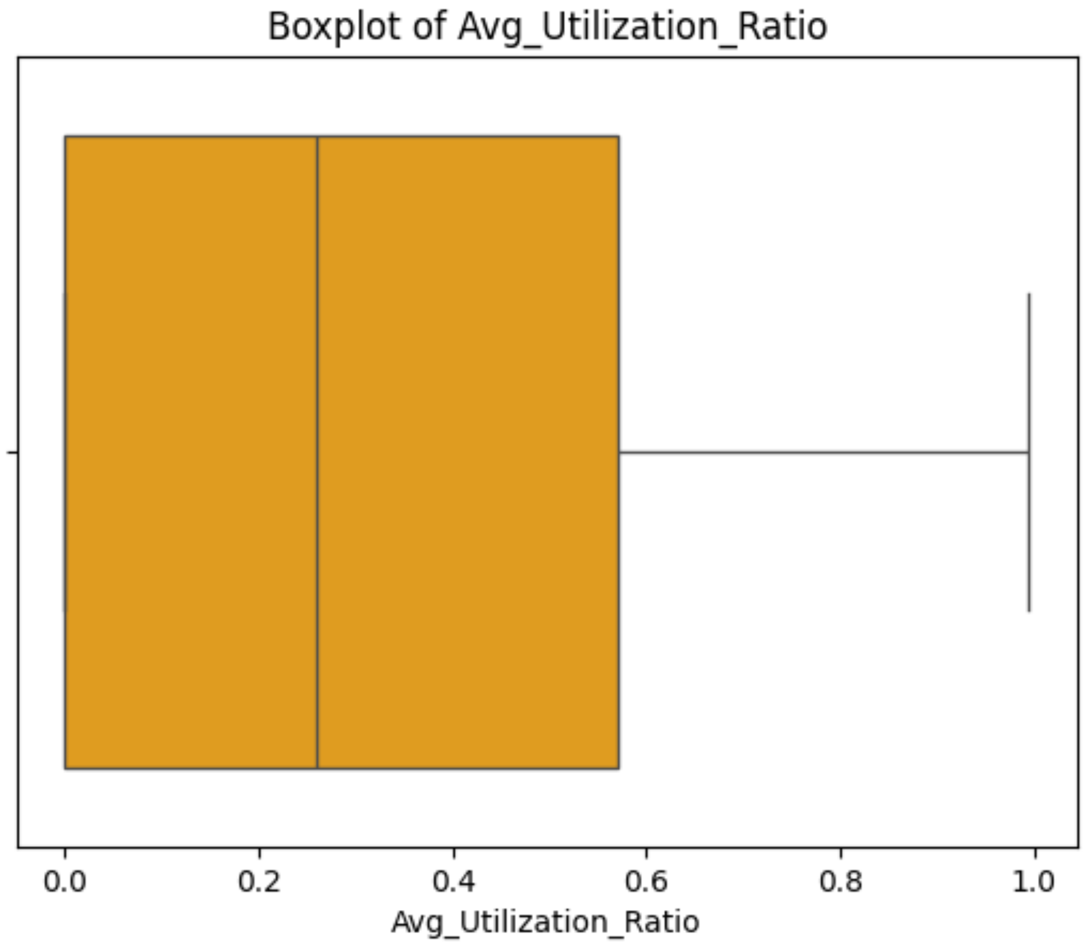

```
After handling outliers, dataset shape: (6658, 20)
Column: Total_Trans_Ct
Number of outliers: 0
```




```
After handling outliers, dataset shape: (6658, 20)
Column: Total_Ct_Chng_Q4_Q1
Number of outliers: 195
```




```
After handling outliers, dataset shape: (6463, 20)
Column: Avg_Utilization_Ratio
Number of outliers: 0
```




```
After handling outliers, dataset shape: (6463, 20)
```

In [33]:
```
df.shape
```

Out[33]:
```
(6463, 20)
```

In [34]:
```
df.head(10)
```

Out[34]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Ma
10	Existing Customer	42	M	5	Uneducated	
14	Existing Customer	57	F	2	Graduate	
19	Existing Customer	45	F	2	Graduate	
21	Attrited Customer	62	F	0	Graduate	
23	Existing Customer	47	F	4	Unknown	
24	Existing Customer	54	M	2	Unknown	
25	Existing Customer	41	F	3	Graduate	
34	Existing Customer	58	M	0	Graduate	
35	Existing Customer	55	F	1	College	
44	Existing Customer	38	F	4	Graduate	

```
Analysis of plotting
```

> - Everything seems fine and there are no outliers in the columns.
> - Columns are cleaned from outliers and also there are no missing values in the dataset.
> - The next step is Feature Scaling but before that we need to encode the categorical columns.

In [35]:

```
# label Encoding

from sklearn.preprocessing import LabelEncoder

# List of categorical columns to encode
categorical_columns = ['Attrition_Flag','Gender', 'Education_Level', 'Ma

# Initialize the LabelEncoder
le = LabelEncoder()

# Apply Label Encoding to each categorical column
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
```

In [36]:
```
df.head(10)
```

Out[36]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Ma
10	1	42	1	5	5	
14	1	57	0	2	2	
19	1	45	0	2	2	
21	0	62	0	0	2	
23	1	47	0	4	6	
24	1	54	1	2	6	
25	1	41	0	3	2	
34	1	58	1	0	2	
35	1	55	0	1	0	
44	1	38	0	4	2	

Now everything is good to go

Feature Scaling

In [37]:
```
scalar=StandardScaler()
scaled_df = scalar.fit_transform(df)
```

Dimentionality Reduction

Converting the DataFrame into 2D DataFrame for visualization

In [38]:
```
pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_df)
pca_df = pd.DataFrame(data=principal_components ,columns=["PCA1","PCA2"]
pca_df
```

Out[38]:

	PCA1	PCA2
0	1.576037	0.188933
1	-0.114062	-1.763207
2	2.366168	0.871814
3	1.560829	-3.771393
4	-0.099488	-2.768796
...	...	...
6458	1.432810	0.200919
6459	1.603994	0.811521
6460	0.177032	0.887172
6461	1.157564	1.442156
6462	1.314080	1.114657

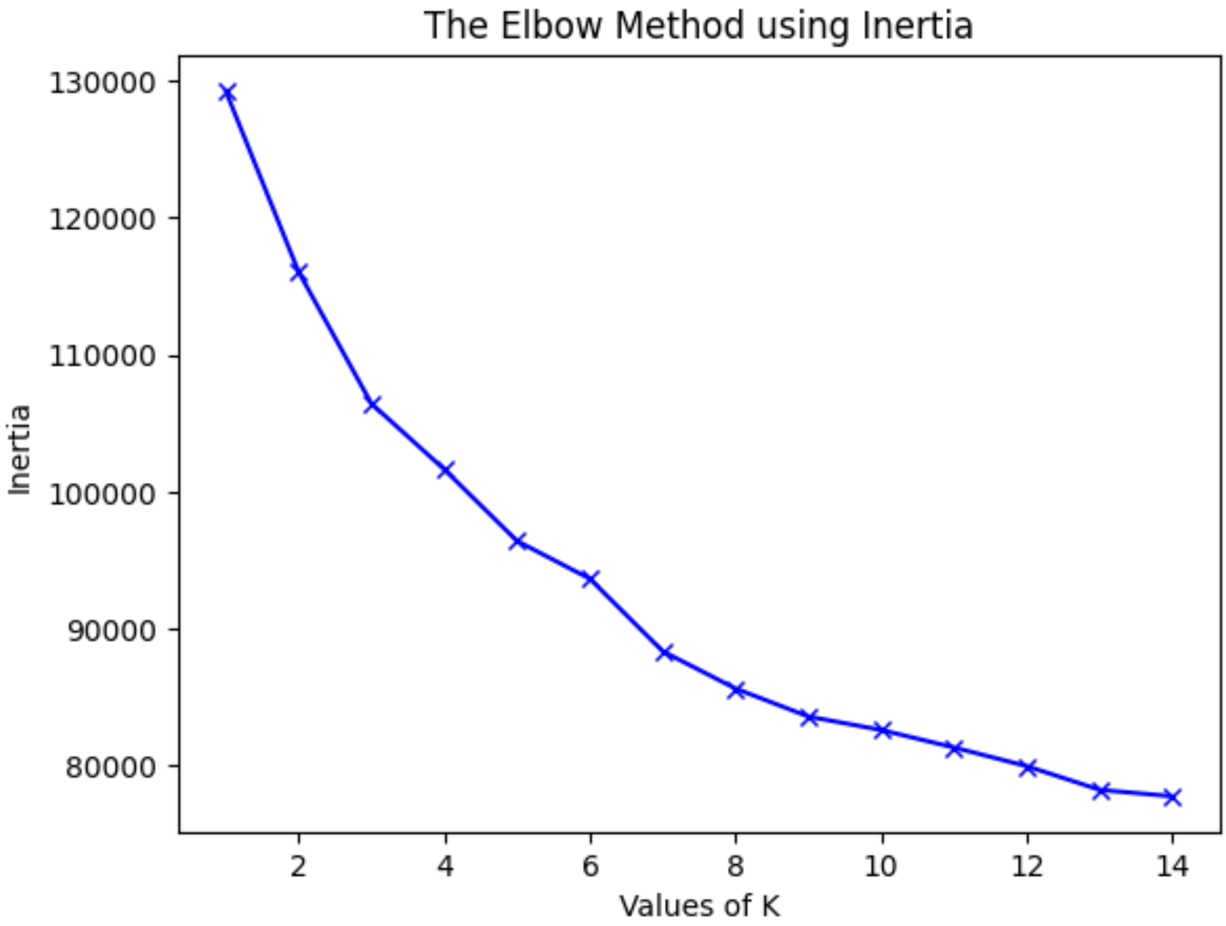
6463 rows × 2 columns

Hyperparameter tuning

Finding 'k' value by Elbow Method

In [39]:

```
inertia = []
range_val = range(1,15)
for i in range_val:
    kmean = KMeans(n_clusters=i)
    kmean.fit_predict(pd.DataFrame(scaled_df))
    inertia.append(kmean.inertia_)
plt.plot(range_val,inertia,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```



Model Building using KMeans

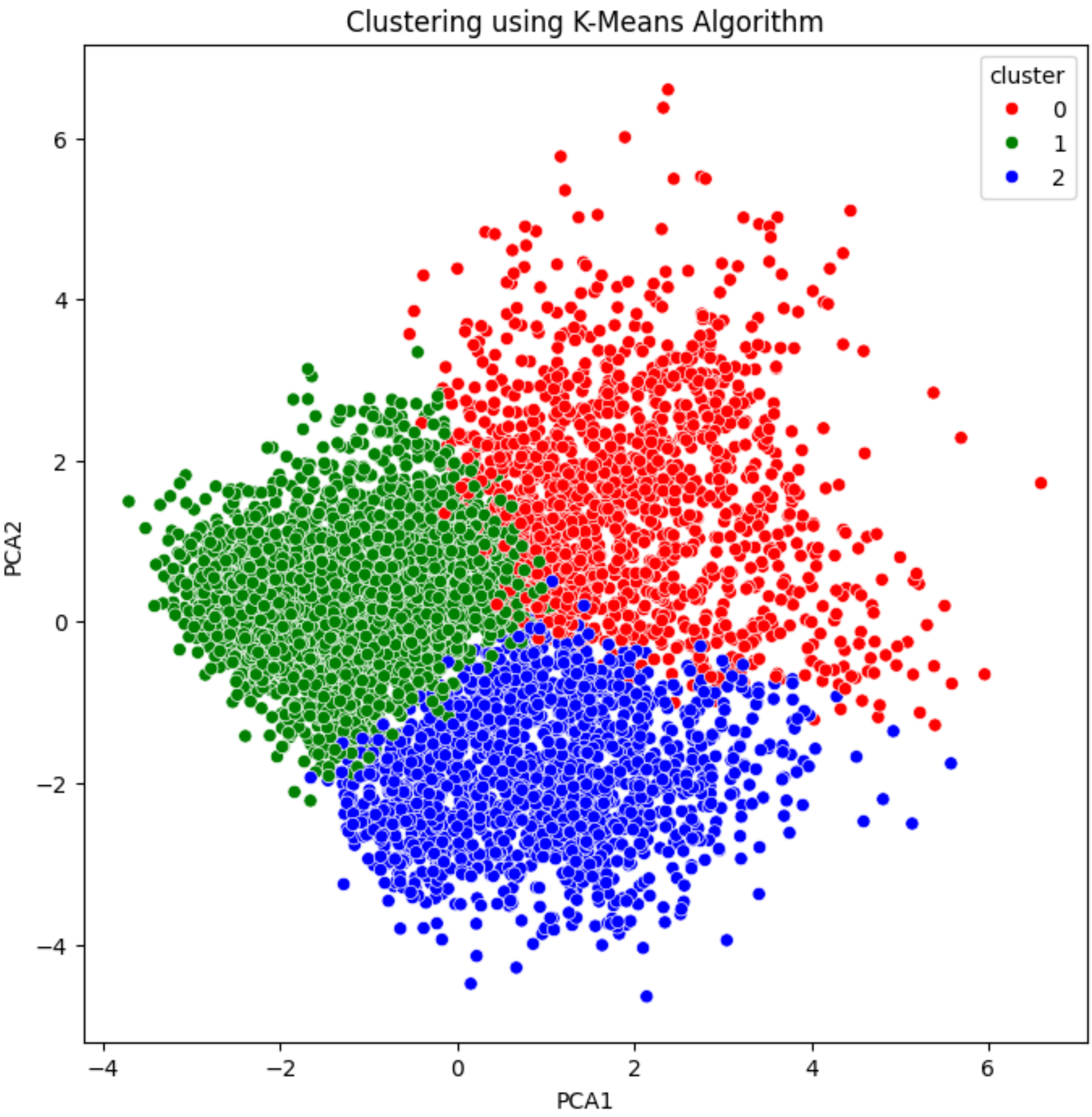
In [70]:

```
kmeans_model=KMeans(3)
kmeans_model.fit_predict(scaled_df)
pca_df_kmeans= pd.concat([pca_df,pd.DataFrame({'cluster':kmeans_model.lal
```

Visualizing the clustered dataframe

In [71]:

```
plt.figure(figsize=(8, 8))
sns.scatterplot(
    x="PCA1",
    y="PCA2",
    hue="cluster",
    data=pca_df_kmeans,
    palette=['red', 'green', 'blue']  # Match the number of clusters
)
plt.title("Clustering using K-Means Algorithm")
plt.show()
```



In [72]:

```
# find all cluster centers
cluster_centers = pd.DataFrame(data=kmeans_model.cluster_centers_,column
# inverse transform the data
cluster_centers = scalar.inverse_transform(cluster_centers)
cluster_centers = pd.DataFrame(data=cluster_centers,columns=[df.columns]
cluster_centers
```

Out[72]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Ma
0	0.901217	45.970651	0.747316	2.457409	3.026485	
1	0.986168	46.106533	0.234550	2.440848	3.114773	
2	0.461631	47.295564	0.380096	2.226619	2.999400	

In [73]:

```
df_reset = df.reset_index(drop=True)
cluster_labels = pd.DataFrame({'Cluster': kmeans_model.labels_}).reset_i

# Concatenate the DataFrames
cluster_df = pd.concat([df_reset, cluster_labels], axis=1)
print(cluster_df.isna().sum())  # Check for NaN values
```

```
Attrition_Flag              0
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
Cluster                     0
dtype: int64
```

In [74]:

```
cluster_df
```

Out[74]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level
0	1	42	1	5	5
1	1	57	0	2	2
2	1	45	0	2	2
3	0	62	0	0	2
4	1	47	0	4	6
...	...	...	...	...	...
6458	0	46	1	3	2
6459	0	48	1	4	0
6460	0	49	0	4	5
6461	0	52	0	5	6
6462	0	30	1	2	2

6463 rows × 21 columns

In [75]:

```
cluster_1_df = cluster_df[cluster_df["Cluster"]==0]
cluster_1_df
```

Out[75]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level
0	1	42	1	5	5
2	1	45	0	2	2
5	1	54	1	2	6
7	1	58	1	0	2
9	1	38	0	4	2
...	...	...	...	...	...
6453	0	52	1	2	0
6457	0	33	1	4	0
6459	0	48	1	4	0
6461	0	52	0	5	6
6462	0	30	1	2	2

1396 rows × 21 columns

In [76]:

```
cluster_2_df = cluster_df[cluster_df["Cluster"]==1]
```

```
cluster_2_df
```

Out[76]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level
22	1	37	0	3	5
35	1	53	0	2	3
66	1	51	0	5	4
73	1	53	0	2	3
79	1	44	0	2	2
...	...	...	...	...	...
6451	0	51	1	3	3
6454	0	39	1	4	2
6455	0	52	1	3	6
6456	0	46	0	3	6
6460	0	49	0	4	5

3394 rows × 21 columns

In [77]:

```
cluster_3_df = cluster_df[cluster_df["Cluster"]==2]
cluster_3_df
```

Out[77]:

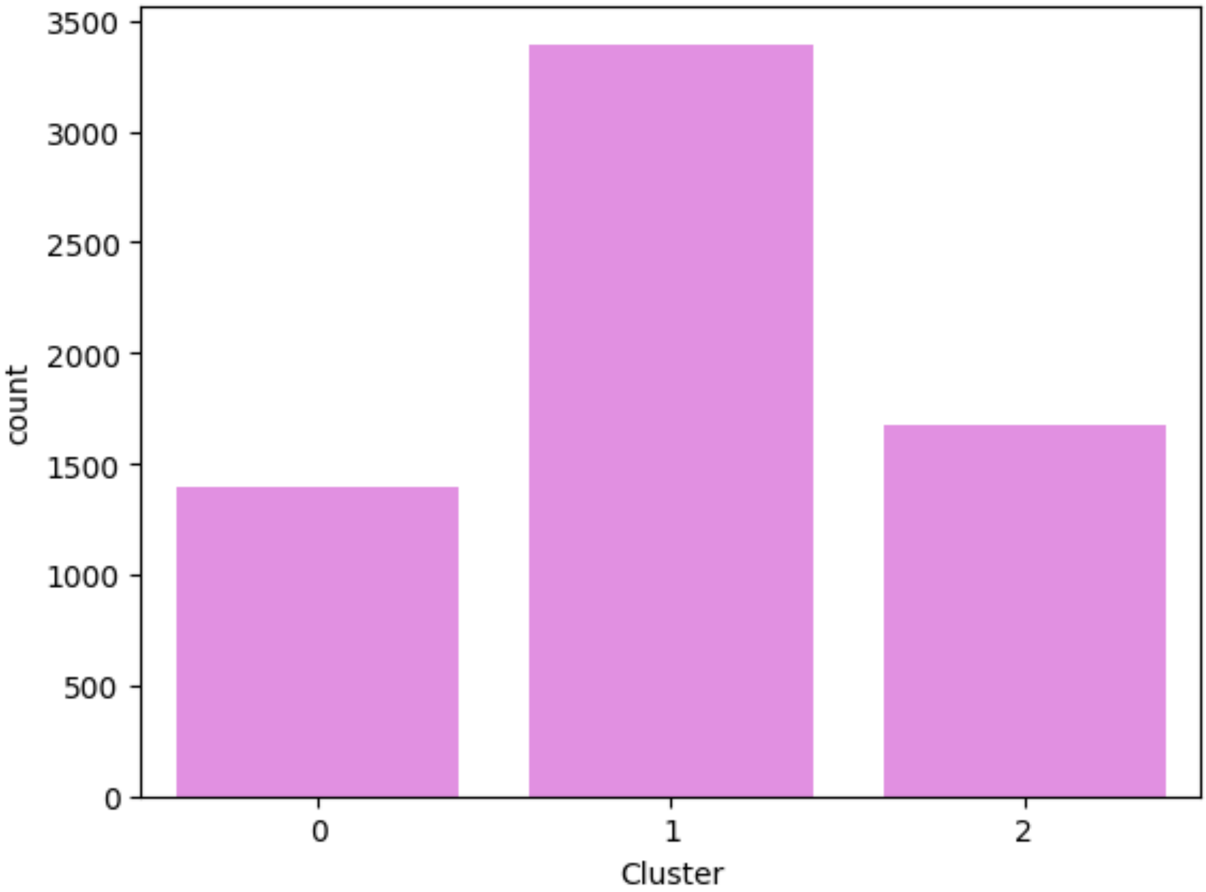
	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level
1	1	57	0	2	2
3	0	62	0	0	2
4	1	47	0	4	6
6	1	41	0	3	2
8	1	55	0	1	0
...	...	...	...	...	...
6394	0	62	0	0	5
6398	0	45	1	5	5
6403	0	40	1	3	3
6420	0	54	1	1	2
6458	0	46	1	3	2

1673 rows × 21 columns

Visualization of Clusters

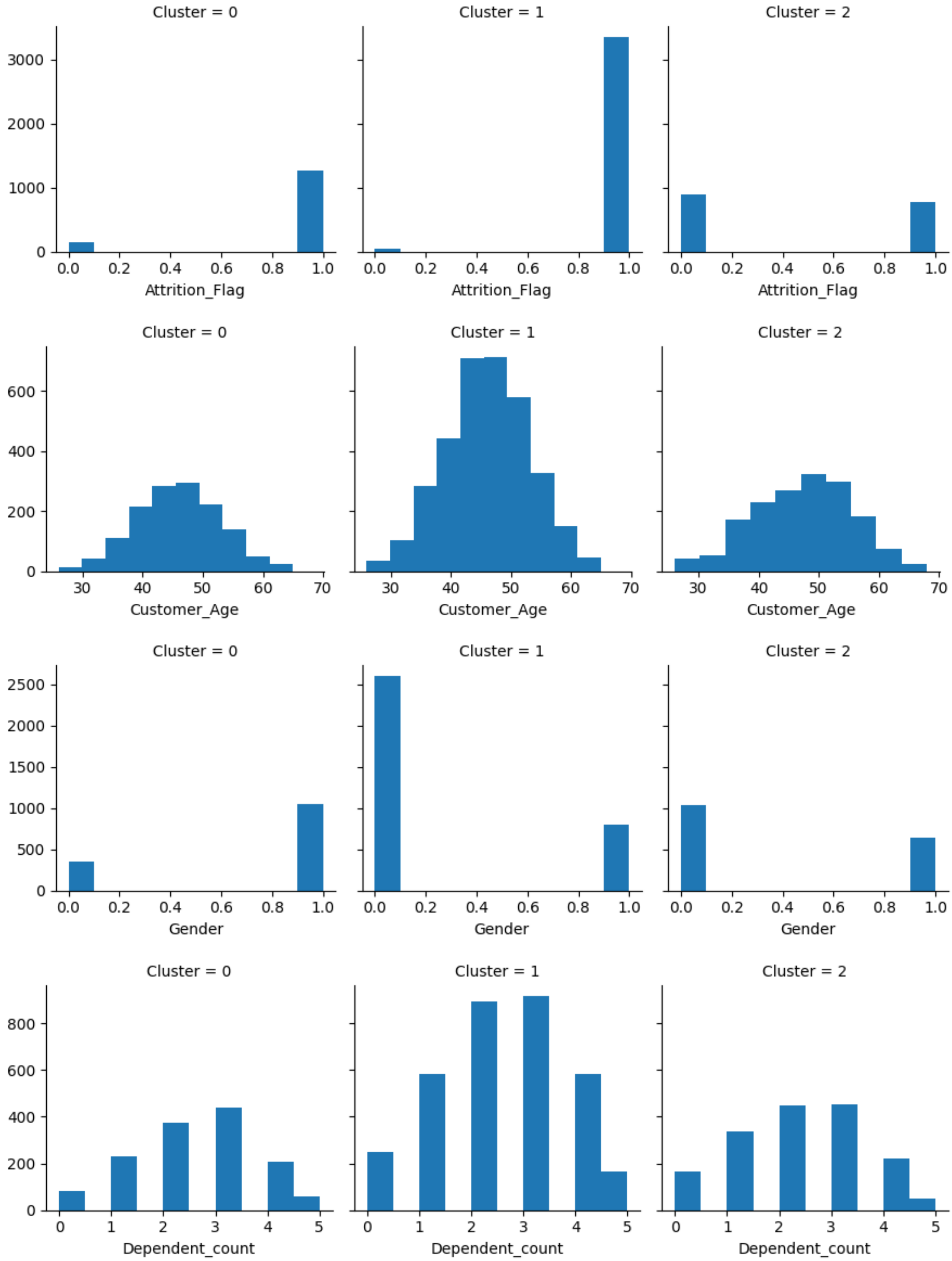
In [78]:
```
#Visualization
sns.countplot(x='Cluster', data=cluster_df, color='violet')
```

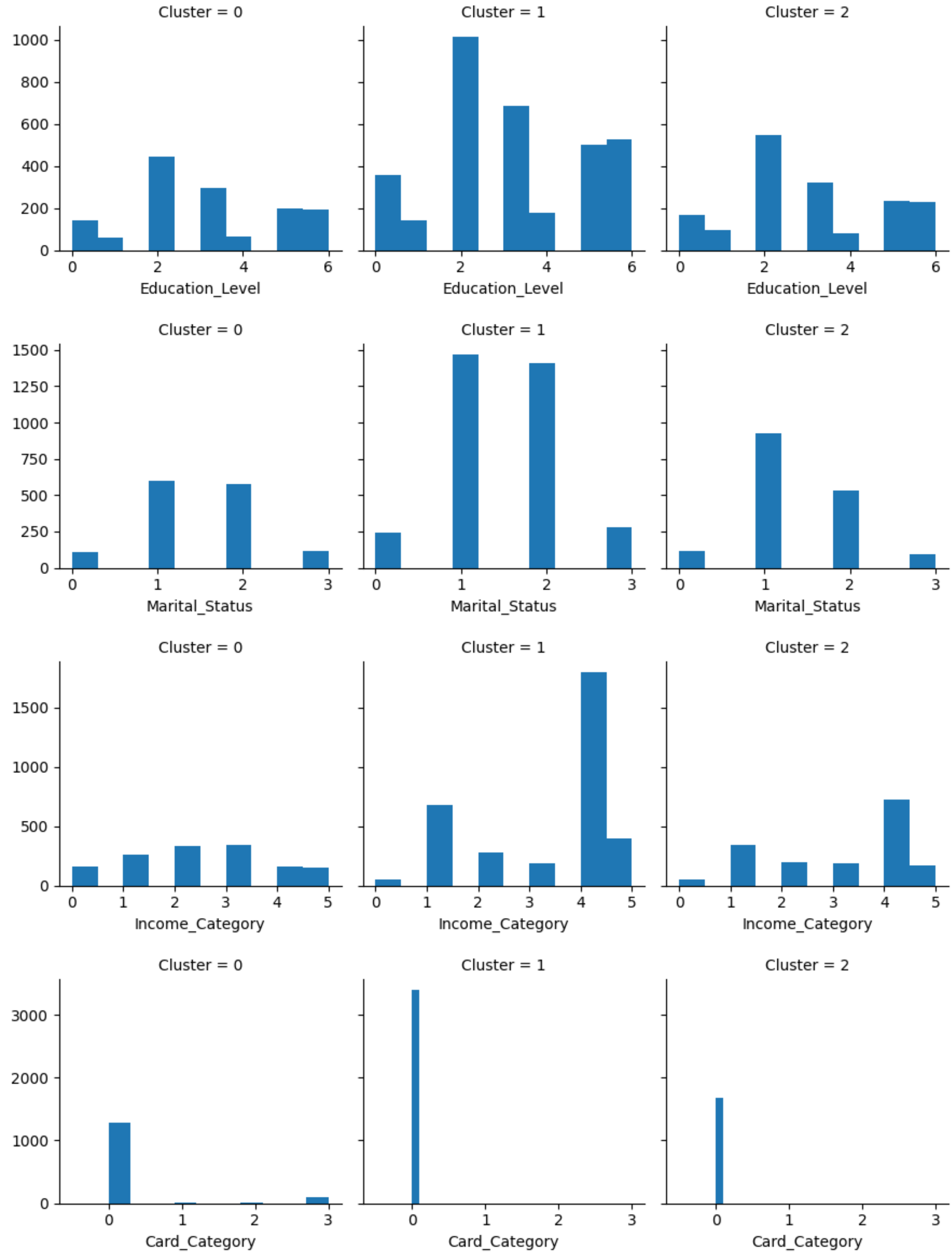
Out[78]: <Axes: xlabel='Cluster', ylabel='count'>

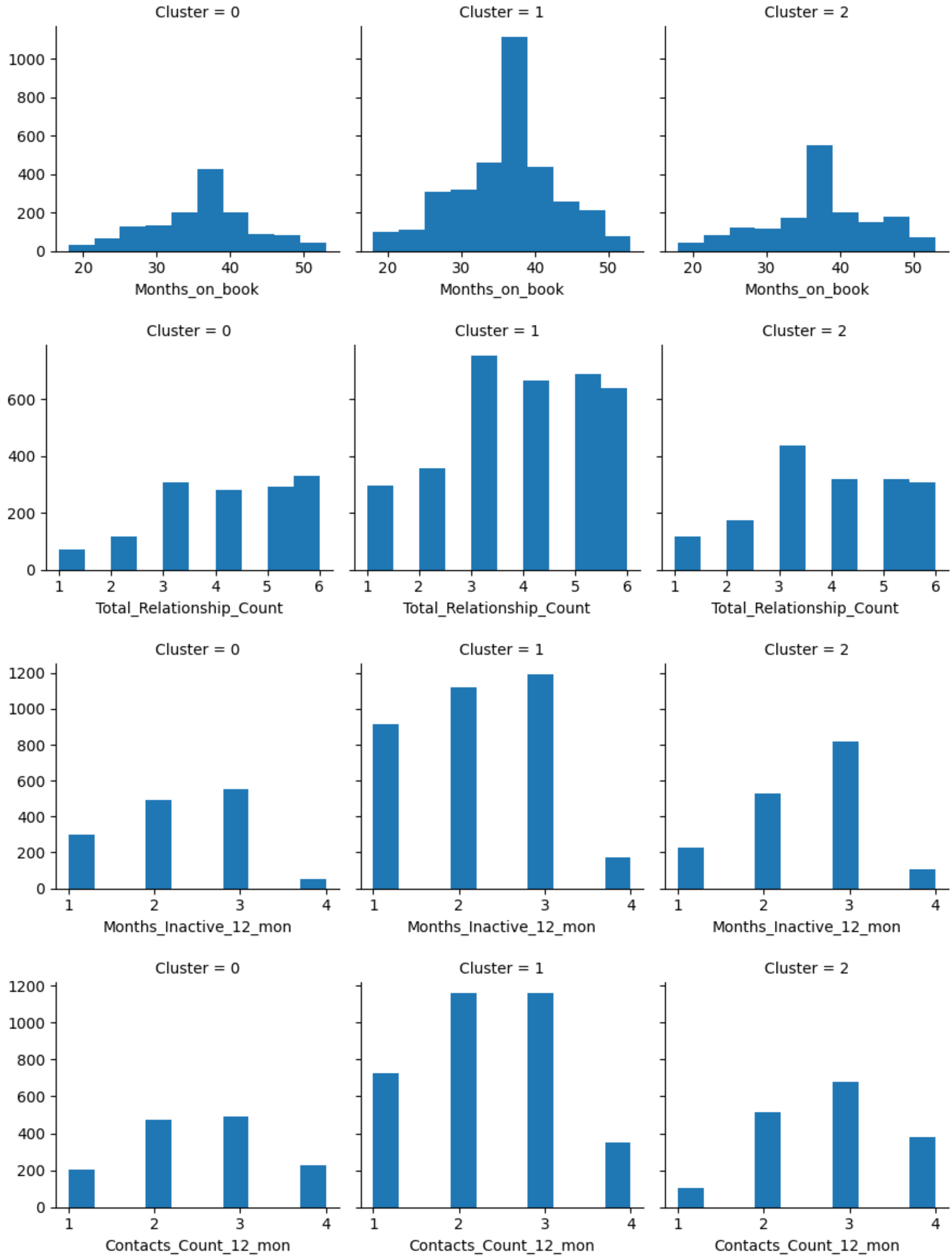


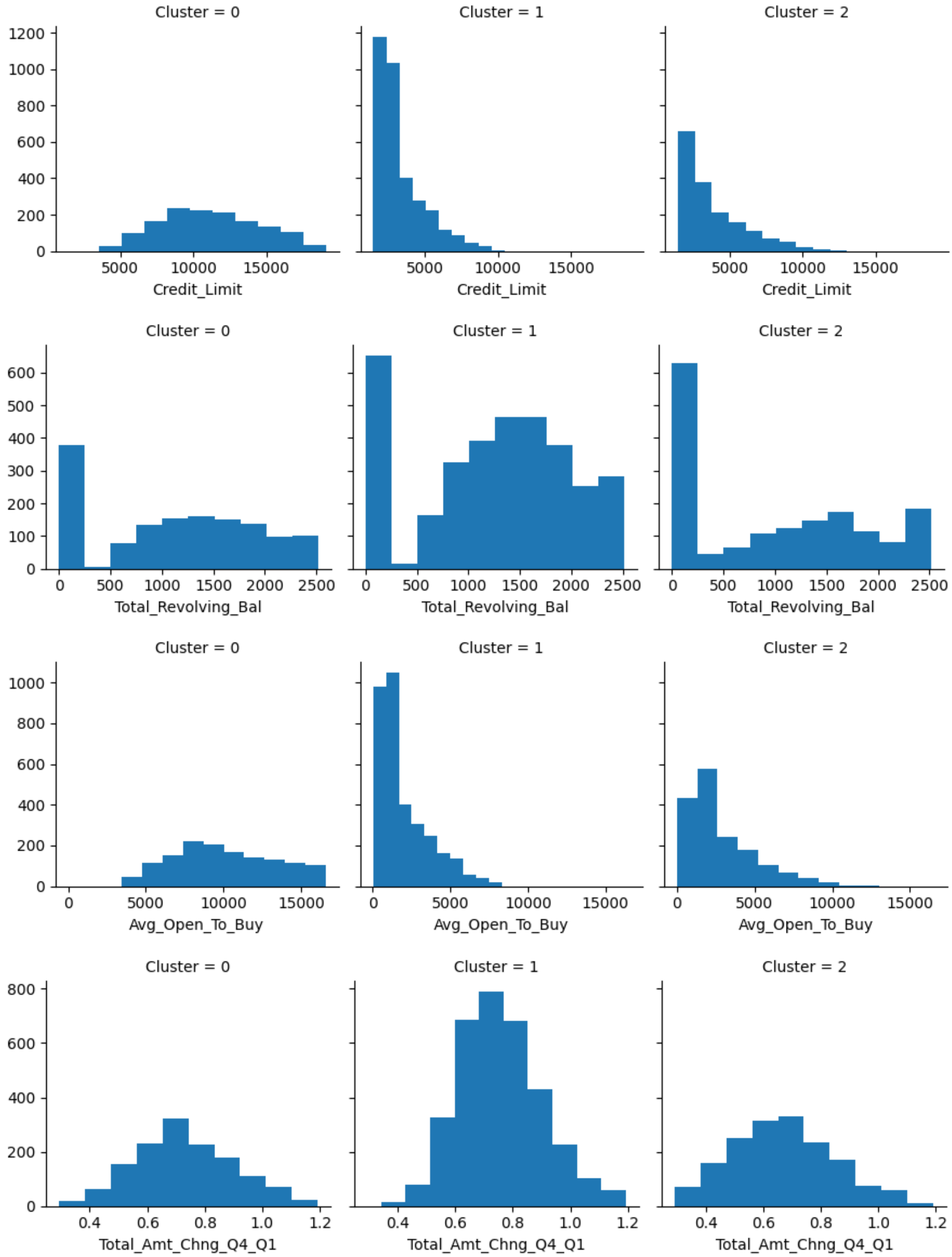
Most of the Data belongs to Cluster 1

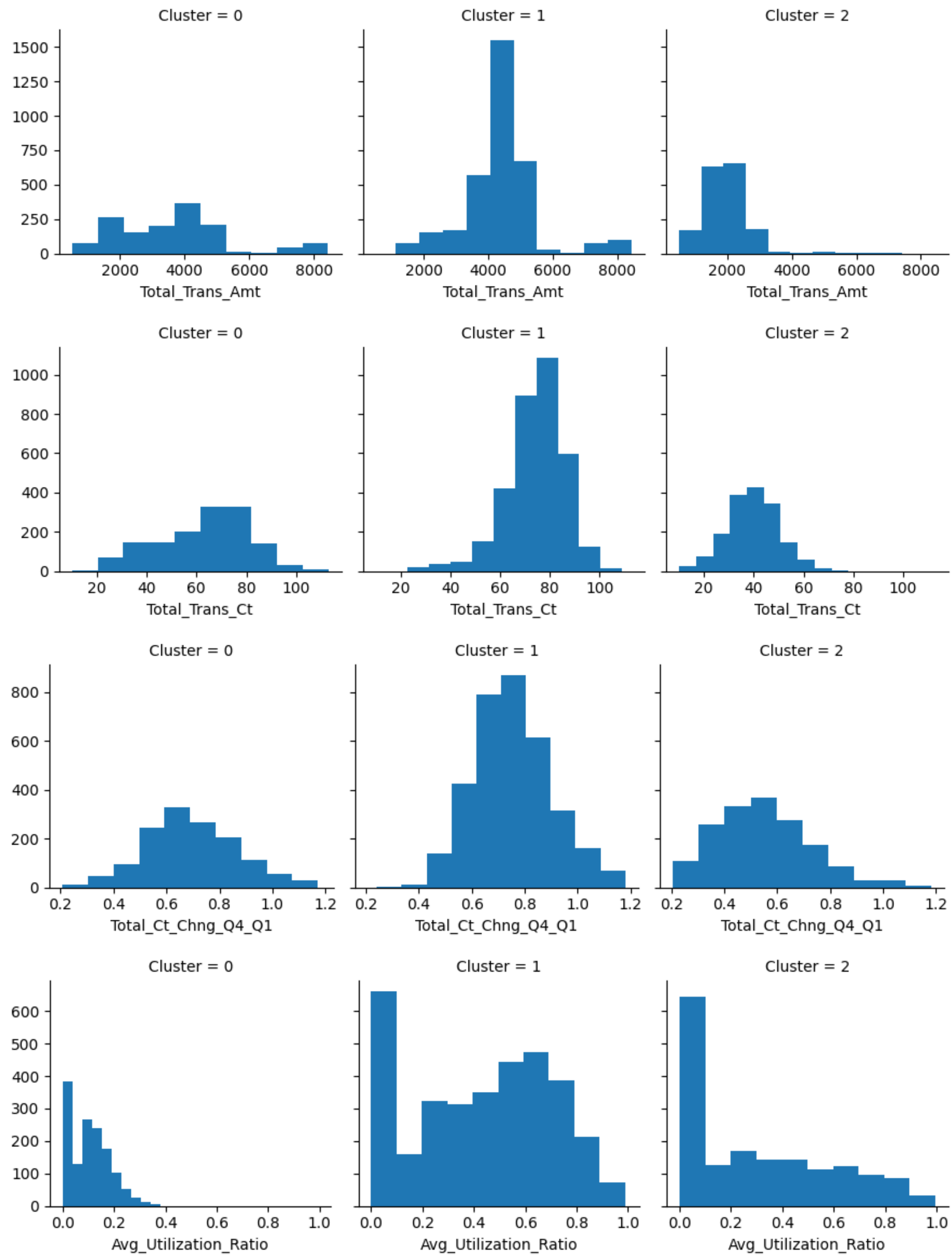
In [79]:
```
for c in cluster_df.drop(['Cluster'],axis=1):
    grid= sns.FacetGrid(cluster_df, col='Cluster')
    grid= grid.map(plt.hist, c)
plt.show()
```


Cluster = 0
Cluster = 1
Cluster = 2
Attrition_Flag
Customer_Age
Gender
Dependent_count


Cluster = 0
Cluster = 1
Cluster = 2
Education_Level
Marital_Status
Income_Category
Card_Category
1000
800
600
400
200
0
1500
1250
750
500
250
3000
2000
0
1
2
3
4
5
6


Cluster = 0
Cluster = 1
Cluster = 2
Months_on_book
Total_Relationship_Count
Months_Inactive_12_mon
Contacts_Count_12_mon


Cluster = 0
Cluster = 1
Cluster = 2
Credit_Limit
Total_Revolving_Bal
Avg_Open_To_Buy
Total_Amt_Chng_Q4_Q1



Saving the kmeans clustering model and the data with cluster

label

In [80]:

```
#Saving Scikitlearn models
import joblib
joblib.dump(kmeans_model, "kmeans_model.pkl")
```

Out[80]: ['kmeans_model.pkl']

In [81]:

```
cluster_df.to_csv("Clustered_Customer_Data.csv")
```

Feature Selection

In [85]:

```
import pandas as pd
from sklearn.feature_selection import SelectKBest, chi2

X = cluster_df.drop('Cluster', axis=1)
y = cluster_df['Cluster']

# Apply SelectKBest with Chi-Square Test
best_features = SelectKBest(score_func=chi2, k=5)
fit = best_features.fit(X, y)

# Get top 5 feature names
selected_features = X.columns[fit.get_support()]
print("Top 5 Features Selected:", selected_features)

# Feature Scores
feature_scores = pd.DataFrame({'Feature': X.columns, 'Score': fit.scores_
feature_scores = feature_scores.sort_values(by='Score', ascending=False)
print("\nFeature Scores:\n", feature_scores)
```

```
Top 5 Features Selected: Index(['Credit_Limit', 'Total_Revolving_Bal', 'Av
g_Open_To_Buy',
       'Total_Trans_Amt', 'Total_Trans_Ct'],
      dtype='object')

Feature Scores:
                     Feature         Score
14            Avg_Open_To_Buy  1.715333e+07
12               Credit_Limit  1.303197e+07
16            Total_Trans_Amt  1.853819e+06
13        Total_Revolving_Bal  7.184135e+04
17             Total_Trans_Ct  2.129077e+04
7               Card_Category  1.192177e+03
2                      Gender  6.795173e+02
0              Attrition_Flag  3.782008e+02
19      Avg_Utilization_Ratio  3.267072e+02
6             Income_Category  2.356321e+02
11      Contacts_Count_12_mon  9.499444e+01
18        Total_Ct_Chng_Q4_Q1  6.941351e+01
10     Months_Inactive_12_mon  4.271503e+01
8              Months_on_book  3.968515e+01
1                Customer_Age  3.900077e+01
3             Dependent_count  2.491556e+01
9    Total_Relationship_Count  1.830483e+01
5              Marital_Status  1.688911e+01
15        Total_Amt_Chng_Q4_Q1  1.200602e+01
4             Education_Level  5.555084e+00
```

In [91]:

```
# Filter dataset with selected features
selected_features = X[selected_features]
```

Training and Testing the model accuracy using decision tree

In [94]:

```
#Split Dataset
X = selected_features
y= cluster_df[['Cluster']]
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.3)
```

In [95]:

```
X_train
```

Out[95]:

	Credit_Limit	Total_Revolving_Bal	Avg_Open_To_Buy	Total_Trans_Amt	Tota
862	6256.0	1530	4726.0	1629	
2662	13883.0	783	13100.0	1871	
2501	1625.0	0	1625.0	2314	
1995	6331.0	1420	4911.0	3527	
5137	4703.0	1555	3148.0	4127	
...	...	...	...	...	
5395	1855.0	907	948.0	4191	
1578	4505.0	1562	2943.0	3968	
3242	2033.0	228	1805.0	2572	
1191	2540.0	1402	1138.0	2384	
4223	3114.0	2208	906.0	5219	

4524 rows × 5 columns

In [96]:

```
X_test
```

Out[96]:

	Credit_Limit	Total_Revolving_Bal	Avg_Open_To_Buy	Total_Trans_Amt	Tota
4144	1623.0	0	1623.0	4965	
1853	1594.0	927	667.0	3851	
6029	12745.0	0	12745.0	4380	
3321	2258.0	1250	1008.0	4368	
4940	1815.0	557	1258.0	4519	
...	...	...	...	...	
1610	11091.0	0	11091.0	1234	
8	3520.0	1914	1606.0	1407	
5207	1569.0	257	1312.0	2288	
2621	2912.0	1501	1411.0	4518	
337	4531.0	1214	3317.0	1414	

1939 rows × 5 columns

In [97]:

```
#Decision_Tree
model= DecisionTreeClassifier(criterion="entropy")
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [98]:

```
#Confusion_Matrix
print(metrics.confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[382  43  25]
 [ 50 924  41]
 [ 45  53 376]]
              precision    recall  f1-score   support

           0       0.80      0.85      0.82       450
           1       0.91      0.91      0.91      1015
           2       0.85      0.79      0.82       474

    accuracy                           0.87      1939
   macro avg       0.85      0.85      0.85      1939
weighted avg       0.87      0.87      0.87      1939
```

Saving the Decision tree model for future prediction

In [99]:

```
import pickle
filename = 'final_model.sav'
pickle.dump(model, open(filename, 'wb'))

# some time later maybe...

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result,'% Acuuracy')
```

```
0.8674574522949974 % Acuuracy
```

In [ ]: