Credit-Card Customer Segmentation and Behavior Analysis

Business Problem:"*A credit card company noticed that despite having a large customer base, retention rates were falling and default rates were climbing. Marketing campaigns were sending the same offers to all customers, with mixed results. This project aims to change that*".

Goal: Segment credit card customers based on spending patterns and repayment behavior to help businesses design targeted offers, improve retention, and detect risky customers.

In [ ]:
```
#Importing all necessary libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
from sklearn.preprocessing import StandardScaler
from scipy import stats
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

In [433…
```
# Downloading the dataset from Kaggle
# ! kaggle datasets download arjunbhasin2013/ccdata
```

In [434…
```
# Extracting the all files from the zip folder
try:
    with zipfile.ZipFile('ccdata.zip','r') as file:
        file.extractall()
        print('Executed')
except :
    print(FileNotFoundError)
```

```
Executed
```

In [435…
```
DATA_PATH='CC GENERAL.csv'
```

In [436…
```
#Function to check the source path does exist or not. If path exist the it wil
def load_data(path=DATA_PATH):
    if not os.path.exists(path):
        raise FileNotFoundError(f"Data file not found at {path}.")
    df = pd.read_csv(path)
    print(f"Loaded data: {df.shape[0]} rows, {df.shape[1]} columns")
    return df
```

In [437…
```
# Reading the dataset with pandas
df=load_data(DATA_PATH)
```

```
Loaded data: 8950 rows, 18 columns
```

In [438…

```
df.shape
```

Out[438…

```
(8950, 18)
```

In [439…

```
df.columns
```

Out[439…

```
Index(['CUST_ID', 'BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES',
       'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'CREDIT_LIMIT', 'PAYMENTS',
       'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT', 'TENURE'],
      dtype='object')
```

Our data represents 8,950 credit card users, each with unique spending habits, repayment behaviors, and credit limits. Think of them as 8,950 stories waiting to be understood.

In [440…

```
#Extracting the besic information of the dataset
def data_report(dataframe):
    print(dataframe.info())
    print("\nFirst rows:\n", dataframe.head())
```

In [441…

```
data_report(df)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   CUST_ID                           8950 non-null   object 
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64  
 12  PURCHASES_TRX                     8950 non-null   int64  
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
None

First rows:
   CUST_ID      BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  \
0  C10001    40.900749           0.818182      95.40              0.00   
1  C10002  3202.467416           0.909091       0.00              0.00   
2  C10003  2495.148862           1.000000     773.17            773.17   
3  C10004  1666.670542           0.636364    1499.00           1499.00   
4  C10005   817.714335           1.000000      16.00             16.00   

   INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
0                    95.4      0.000000             0.166667   
1                     0.0   6442.945483             0.000000   
2                     0.0      0.000000             1.000000   
3                     0.0    205.788017             0.083333   
4                     0.0      0.000000             0.083333   

   ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
0                    0.000000                          0.083333   
1                    0.000000                          0.000000   
2                    1.000000                          0.000000   
3                    0.083333                          0.000000   
4                    0.083333                          0.000000   

   CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
0                0.000000                 0              2        1000.0   
1                0.250000                 4              0        7000.0   
2                0.000000                 0             12        7500.0   
3                0.083333                 1              1        7500.0   
```

```
4                0.000000                0             1       1200.0

      PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT  TENURE
0   201.802084        139.509787          0.000000      12
1  4103.032597       1072.340217          0.222222      12
2   622.066742        627.284787          0.000000      12
3     0.000000               NaN          0.000000      12
4   678.334763        244.791237          0.000000      12
```

Data Set Explanation:

The dataset contains the information of custoemr's credit card usages. It's contains total 18 columns and 8950 rows. The columns are as follows:

- **CUST_ID**: Unique identifier for each customer.
- **BALANCE**: Current balance on the credit card.
- **BALANCE_FREQUENCY**: Frequency of balance updates on-go. If the value is 1 then it updated frequently, if 0 then not updated frequently.
- **PURCHASES**: Total amount of purchases made by the customer.
- **ONEOFF_PURCHASES**: The highest amount of perchase in one-go.
- **INSTALLMENTS_PURCHASES**: The perchase amount on installments.
- **CASH_ADVANCE**: Advanced cash given by the customer.
- **PURCHASES_FREQUENCY**: How freuently the custmer did perchase any product. If it's 1 then the coustomer purchase frequently, if it's 0 then don't.
- **ONEOFF_PURCHASES_FREQUENCY** : How frequently the customer make one time purchase. If it's 1 then the customer do frequently, if it's 0 then he don't.
- **PURCHASES_INSTALLMENTS_FREQUENCY**: The frequency of purchesing product on installments. If it's 1 then the customer do frequently, if it's 0 then he don't.
- **CASH_ADVANCE_FREQUENCY**: The frequency of purchesing product on advanced payment. If it's 1 then the customer do frequently, if it's 0 then he don't.
- **CASH_ADVANCE_TRX**: Number of transactions done with "Cash in Advanced" .
- **PURCHASES_TRX** :The number of transactions made with peoduct purchase.
- **CREDIT_LIMIT**: Maximum limit can be purchas through credit card.
- **PAYMENTS**: Amount of payment done by user.
- **MINIMUM_PAYMENTS**: The minimum amount of purchase done by user.
- **PRC_FULL_PAYMENT** : Percentage of full Payment paid by user.

- **TENURE** :The duration within which user must repay the borrowed amount along with interest

In [442…

```
#Statistical summary of the dataset
df.describe().T.style.background_gradient(cmap="YlGnBu")
```

Out[442…

	count	mean	std
BALANCE	8950.000000	1564.474828	2081.531879
BALANCE_FREQUENCY	8950.000000	0.877271	0.236904
PURCHASES	8950.000000	1003.204834	2136.634782
ONEOFF_PURCHASES	8950.000000	592.437371	1659.887917
INSTALLMENTS_PURCHASES	8950.000000	411.067645	904.338115
CASH_ADVANCE	8950.000000	978.871112	2097.163877
PURCHASES_FREQUENCY	8950.000000	0.490351	0.401371
ONEOFF_PURCHASES_FREQUENCY	8950.000000	0.202458	0.298336
PURCHASES_INSTALLMENTS_FREQUENCY	8950.000000	0.364437	0.397448
CASH_ADVANCE_FREQUENCY	8950.000000	0.135144	0.200121
CASH_ADVANCE_TRX	8950.000000	3.248827	6.824647
PURCHASES_TRX	8950.000000	14.709832	24.857649
CREDIT_LIMIT	8949.000000	4494.449450	3638.815725
PAYMENTS	8950.000000	1733.143852	2895.063757
MINIMUM_PAYMENTS	8637.000000	864.206542	2372.446607
PRC_FULL_PAYMENT	8950.000000	0.153715	0.292499
TENURE	8950.000000	11.517318	1.338331

This makes it visually clear which columns have high variance, outliers, or skewness.

In [443…

```
# Checking for missing values
df.isna().sum()
```

Out[443…

```
CUST_ID                              0
BALANCE                              0
BALANCE_FREQUENCY                    0
PURCHASES                            0
ONEOFF_PURCHASES                     0
INSTALLMENTS_PURCHASES               0
CASH_ADVANCE                         0
PURCHASES_FREQUENCY                  0
ONEOFF_PURCHASES_FREQUENCY           0
PURCHASES_INSTALLMENTS_FREQUENCY     0
CASH_ADVANCE_FREQUENCY               0
CASH_ADVANCE_TRX                     0
PURCHASES_TRX                        0
CREDIT_LIMIT                         1
PAYMENTS                             0
MINIMUM_PAYMENTS                   313
PRC_FULL_PAYMENT                     0
TENURE                               0
dtype: int64
```

In our dataset, 313 values are missing in the MINIMUM_PAYMENTS column, and 1 value is missing in the CREDIT_LIMIT column.

In [444…

```python
# Filling the empty cells with the median value
df['MINIMUM_PAYMENTS'].fillna(df['MINIMUM_PAYMENTS'].median(), inplace=True)
df['CREDIT_LIMIT'].fillna(df['CREDIT_LIMIT'].median(), inplace=True)
```

The data contains outlier. So, it's better to fill the empty values with median insted mean. Because, the outliers does't effect the median.

In [445…

```python
# Rechecking that all null values got removed or not
df.isna().sum()
```

Out[445…

```
CUST_ID                             0
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        0
PAYMENTS                            0
MINIMUM_PAYMENTS                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

In [446…
```
#Droping duplicates if exits
df['CUST_ID'].drop_duplicates()
```

Out[446…
```
0       C10001
1       C10002
2       C10003
3       C10004
4       C10005
         ...
8945    C19186
8946    C19187
8947    C19188
8948    C19189
8949    C19190
Name: CUST_ID, Length: 8950, dtype: object
```

Exploratory Data Analysis

In [447…
```
#Getting idea about the maximum values of each features
df.max()
```

Out[447…
```
CUST_ID                                  C19190
BALANCE                             19043.13856
BALANCE_FREQUENCY                           1.0
PURCHASES                              49039.57
ONEOFF_PURCHASES                       40761.25
INSTALLMENTS_PURCHASES                  22500.0
CASH_ADVANCE                        47137.21176
PURCHASES_FREQUENCY                         1.0
ONEOFF_PURCHASES_FREQUENCY                  1.0
PURCHASES_INSTALLMENTS_FREQUENCY            1.0
CASH_ADVANCE_FREQUENCY                      1.5
CASH_ADVANCE_TRX                            123
PURCHASES_TRX                               358
CREDIT_LIMIT                            30000.0
PAYMENTS                            50721.48336
MINIMUM_PAYMENTS                    76406.20752
PRC_FULL_PAYMENT                            1.0
TENURE                                       12
dtype: object
```

Interesting Insights:

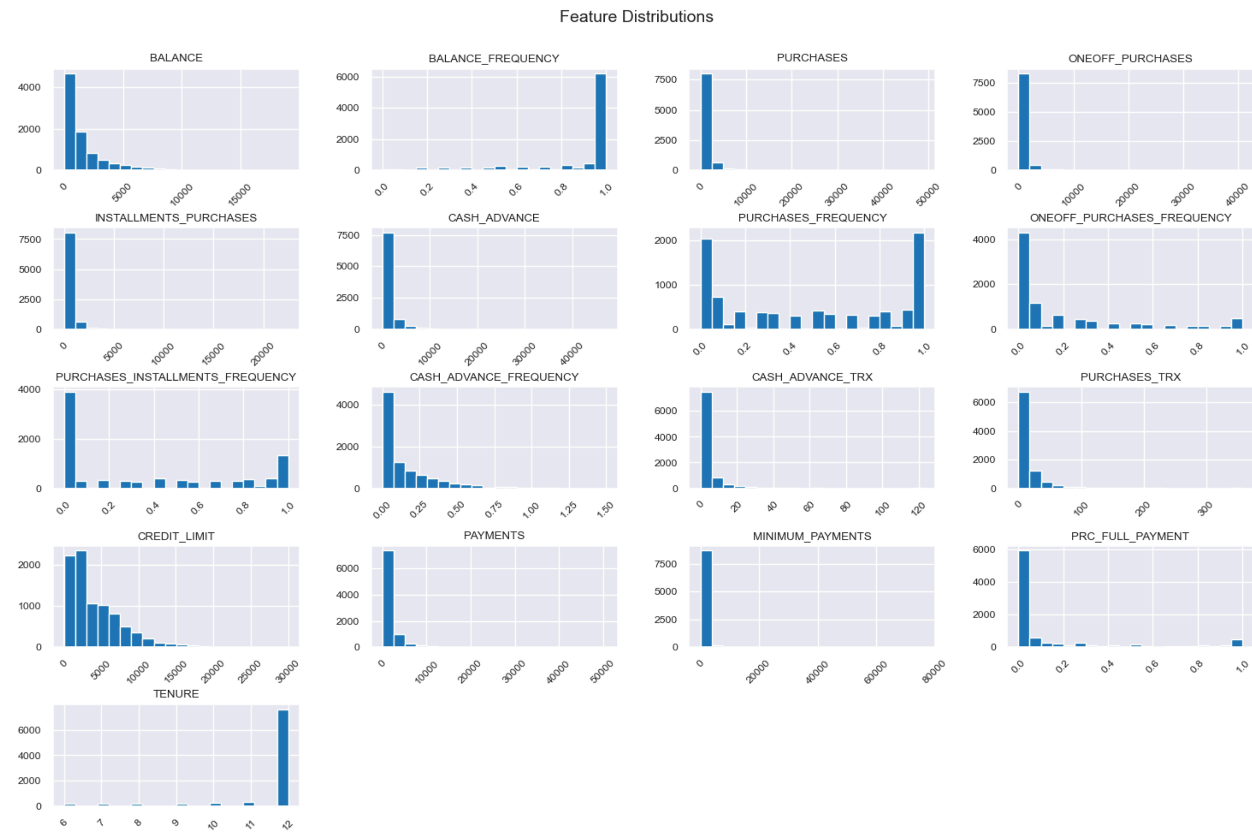
- The highest Balance is 19,043.14.
- The maximum Purchase Amount is 49,039.57.
- The maximum One-off Purchase Amount is 40,761.25.
- The maximum Installment Purchase Amount is 22,500.00.

- The maximum Cash Advance is 47,137.21.
- The maximum Credit Limit is 30,000.00.

In [448…
```
df.min()
```

Out[448…
```
CUST_ID                                  C10001
BALANCE                                     0.0
BALANCE_FREQUENCY                           0.0
PURCHASES                                   0.0
ONEOFF_PURCHASES                            0.0
INSTALLMENTS_PURCHASES                      0.0
CASH_ADVANCE                                0.0
PURCHASES_FREQUENCY                         0.0
ONEOFF_PURCHASES_FREQUENCY                  0.0
PURCHASES_INSTALLMENTS_FREQUENCY            0.0
CASH_ADVANCE_FREQUENCY                      0.0
CASH_ADVANCE_TRX                              0
PURCHASES_TRX                                 0
CREDIT_LIMIT                               50.0
PAYMENTS                                    0.0
MINIMUM_PAYMENTS                       0.019163
PRC_FULL_PAYMENT                            0.0
TENURE                                        6
dtype: object
```

In [449…
```
# Exploring the distribution of all features
axes = df.hist(bins=20, figsize=(15, 10), color='#1f77b4')
plt.suptitle("Feature Distributions", fontsize=14)
plt.tight_layout(pad=2)
for ax in axes.flatten():
    ax.tick_params(axis='x', rotation=45)
    ax.tick_params(axis='y', rotation=0)
plt.show()
```



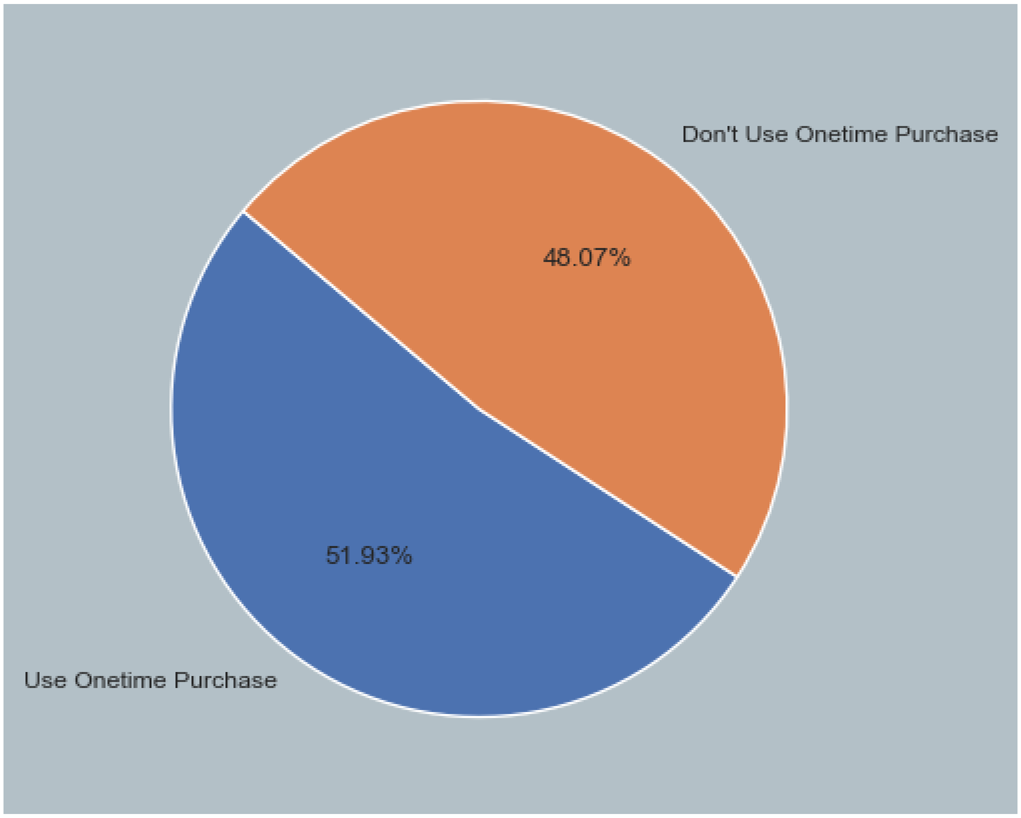
In [450…

```
# Customers who have done ONEOFF_PURCHASES purchase and who don't
total_customers=df['CUST_ID'].count()
num_onetimepurchese=df[df['ONEOFF_PURCHASES']>0]['CUST_ID'].count()
print(f'Total number of customers who have done one time purchese is {num_onet
num_dont_onetimepurchese=total_customers-num_onetimepurchese
print(f'Total number of customers who have done one time purchese is {num_dont
```

```
Total number of customers who have done one time purchese is 4648
Total number of customers who have done one time purchese is 4302
```

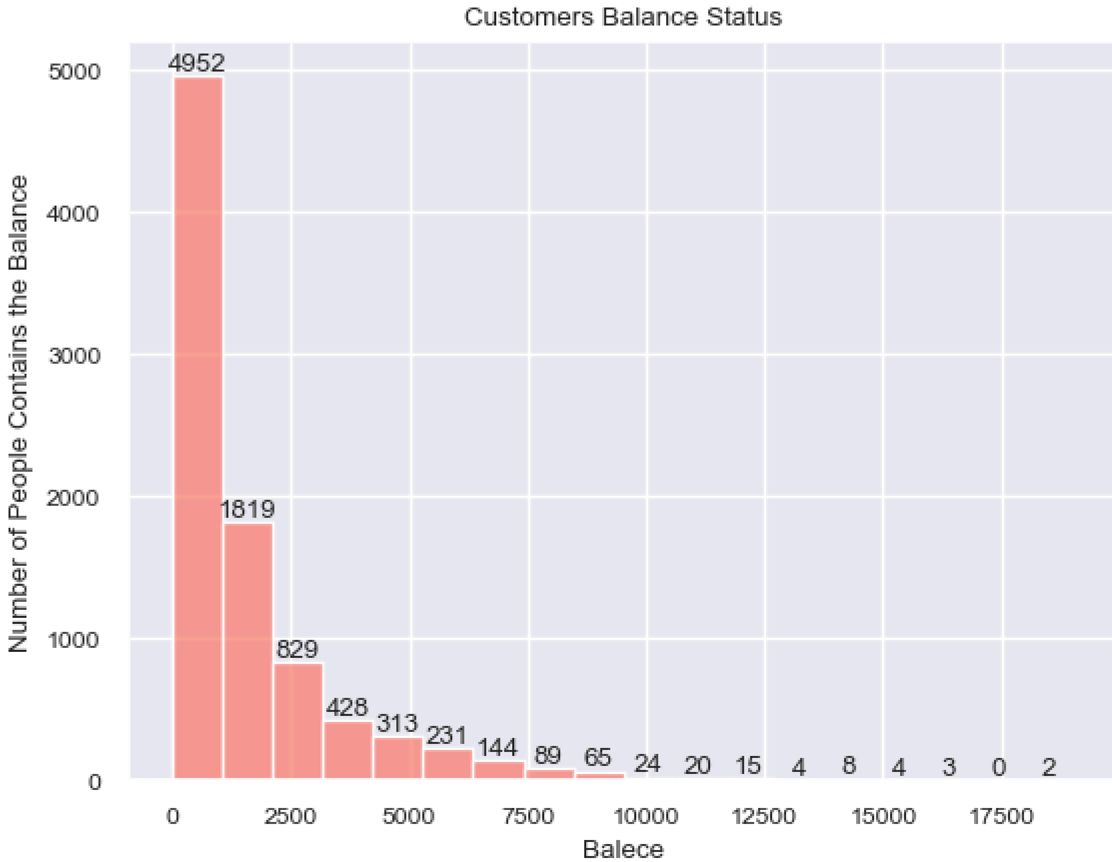
In [451…

```
#Ploting a pie chart of the customer's who have done and who never done ONEOFF
onetime_purchese_cnt=[num_onetimepurchese,num_dont_onetimepurchese]
plt.figure(figsize=(6,5),facecolor='#b3c0c7')
labels=['Use Onetime Purchase',"Don't Use Onetime Purchase"]
color=['#fcb103','#252729']
plt.pie(onetime_purchese_cnt, autopct='%1.2f%%',labels=labels,startangle=140)
plt.show()
```



In [452…

```
# Ploting a histogram
cnt_ax=sns.histplot(data=df,x='BALANCE',binwidth=1050,color='salmon')
for container in cnt_ax.containers:#type: ignore
    cnt_ax.bar_label(container)
plt.title('Customers Balance Status')
plt.ylabel('Number of People Contains the Balance')
plt.xlabel('Balece')
plt.show()
```




```
In [453…
# Just Want to see the customers who have done one time purchase OR Installmen
dataset_onetimeoff=df[df['ONEOFF_PURCHASES']>0]
dataset_installment=df[df['INSTALLMENTS_PURCHASES']>0]
```

```
In [454…
# Oneoff purchases
sns.scatterplot(
    data=dataset_onetimeoff,
    x='BALANCE',
    y='ONEOFF_PURCHASES',
    color='#1f77b4',  # Deep blue
    s=100,
    edgecolor='black',
    marker='o',
    alpha=0.9,
    label='One-off Purchases'
)
#Installment purchases
sns.scatterplot(
    data=dataset_installment,
    x='BALANCE',
    y='INSTALLMENTS_PURCHASES',
    s=80,
    color='#ff7f0e',  # Vibrant orange
    marker='o',
    edgecolor='black',
    alpha=0.9,
```

```
    label='Installment Purchases'
)


plt.title('Balance vs Purchase type')
plt.xlabel('Balance of Customer')
plt.ylabel('Purchase value')

legend = plt.legend(
    loc='upper right',
    prop={'size': 6, 'weight': 'bold'},
    title_fontsize=6
)


plt.show()
```

Key insights:

- We can see that if the customer balence is 0 or near 0 they do prefer installment purchase.
- In most of the cases when Purchase value is high then customers preffer oneoff purchase.

In [455…

```
#Credit limit distribution
```

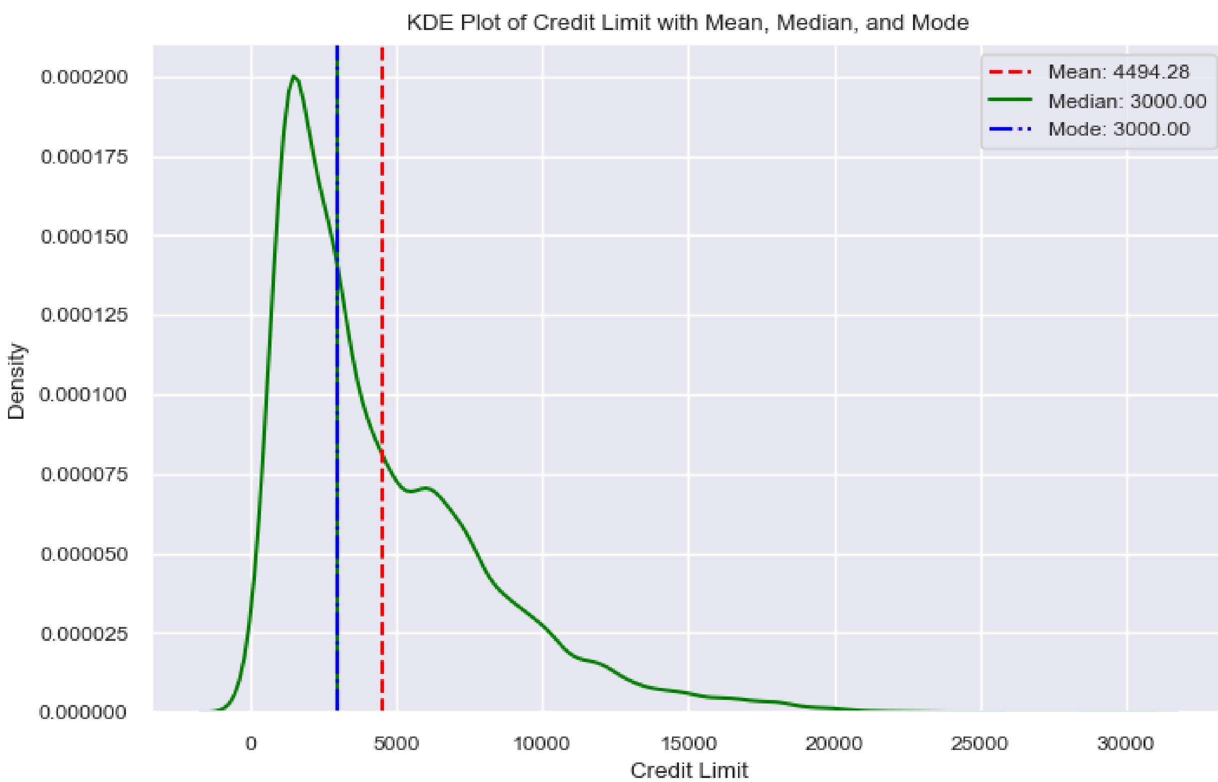
```
credit_limit = df['CREDIT_LIMIT']
mean_val = credit_limit.mean()
median_val = credit_limit.median()
mode_val = credit_limit.mode()[0]

plt.figure(figsize=(8, 5))  # Create figure first

# KDE plot
sns.kdeplot(credit_limit, color='green') #type:ignore

# Add lines for mean, median, and mode
plt.axvline(mean_val, color='red', linestyle='--', label=f'Mean: {mean_val:.2f
plt.axvline(median_val, color='green', linestyle='-', label=f'Median: {median_
plt.axvline(mode_val, color='blue', linestyle='-.', label=f'Mode: {mode_val:.2

# Add legend and labels
plt.legend()
plt.title('KDE Plot of Credit Limit with Mean, Median, and Mode')
plt.xlabel('Credit Limit')
plt.ylabel('Density')
plt.grid(True)
```



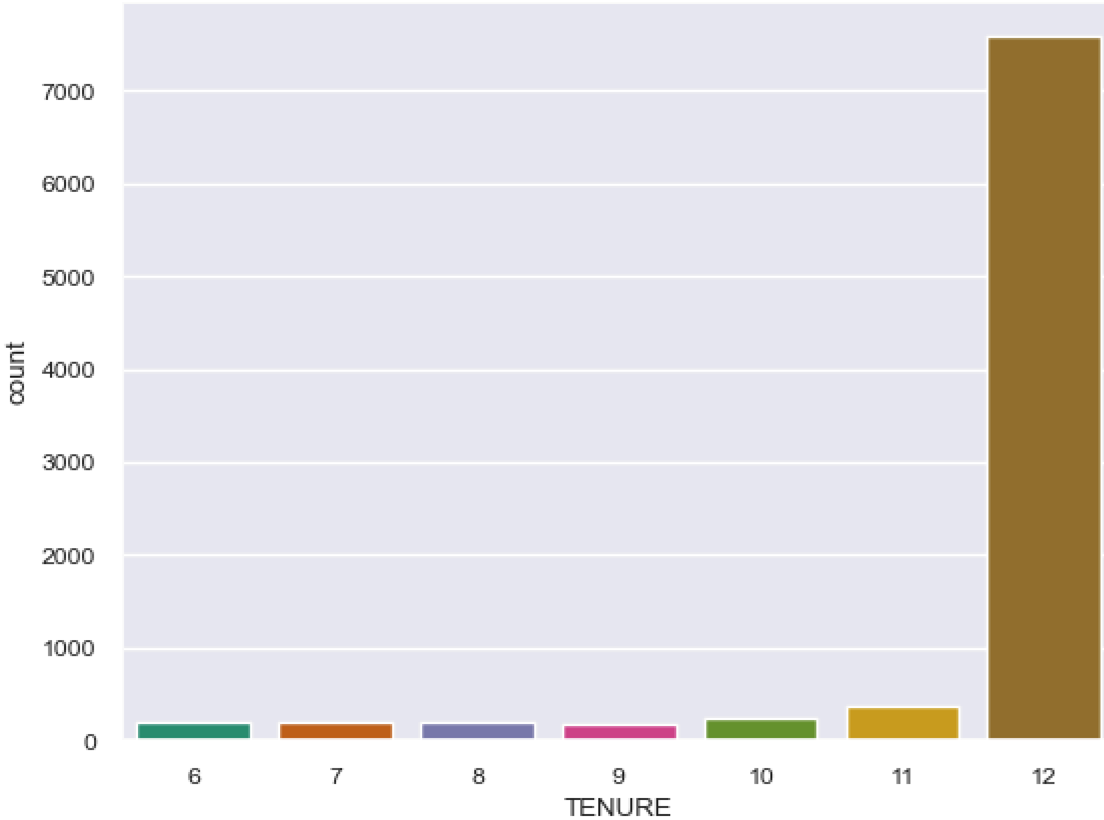
In this distribution, the median and mode are both 3000.0, while the mean is 4494.28. Since the mean is greater than the median, the data is positively skewed (right-skewed), indicating a longer tail on the right side of the distribution. This suggests that a small number of high values are pulling the mean upward compared to the median. So, there exits some outliers.

In [456…

```
sns.countplot(data=df,x='TENURE',palette='Dark2',hue='TENURE',legend=False)
```

Out[456…

```
<Axes: xlabel='TENURE', ylabel='count'>
```



Key Insights:

The majority of customers have a TENURE of 12 months, indicating that most people are given — or choose — the maximum available repayment period. In contrast, very few customers have a tenure of 6, 7, 8, or 9 months. This suggests that the common repayment preference or policy favors a full 12-month period, possibly because it provides greater flexibility in managing repayments, reduces monthly payment amounts, and makes it easier for customers to meet their obligations without financial strain.

In [457…

```
#Ploting the CREDIT_LIMIT and PRC_FULL_PAYMENT
sns.scatterplot(data=df,x='CREDIT_LIMIT',y='PRC_FULL_PAYMENT',color='darkblue'
plt.xlabel('Credit Limit')
plt.ylabel('Percentage of Full Payment')
```

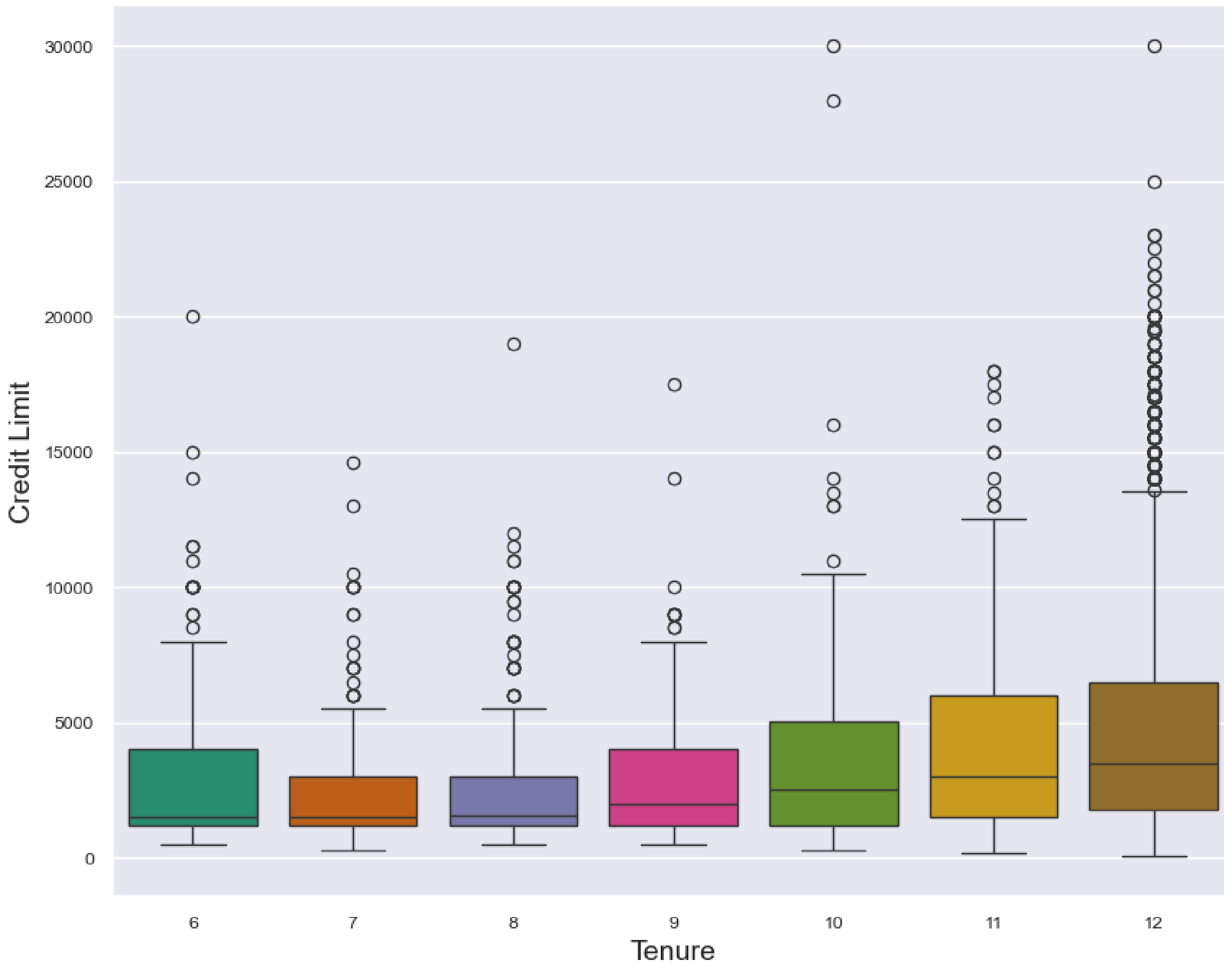
Out[457…

```
Text(0, 0.5, 'Percentage of Full Payment')
```

Plotting the relationship between CREDIT_LIMIT and PRC_FULL_PAYMENT — This visualization helps in understanding how the percentage of full payments made by customers varies with their assigned credit limit. By observing the trend, we can identify whether customers with higher credit limits tend to make full payments more frequently or if the repayment behavior is independent of the credit limit.

In [458…

```
#Box plot of TENURE and CREDIT_LIMIT
plt.figure(figsize=(10,8))
sns.boxplot(x = 'TENURE', y = 'CREDIT_LIMIT', data = df,hue='TENURE',legend=Fa
plt.ylabel('Credit Limit',fontsize=14)
plt.xlabel('Tenure',fontsize=14)
plt.show()
```



Buisness Insights

Insight	Why It Matters
Higher tenure → Higher limits	Shows customer loyalty is rewarded
High variance in long tenure	Bank trusts long-term customers with variable needs
Targeted offers possible	Banks can create tenure-based reward programs
Risk watch for high-limit outliers	Important for fraud detection or credit risk modeling

In [459…

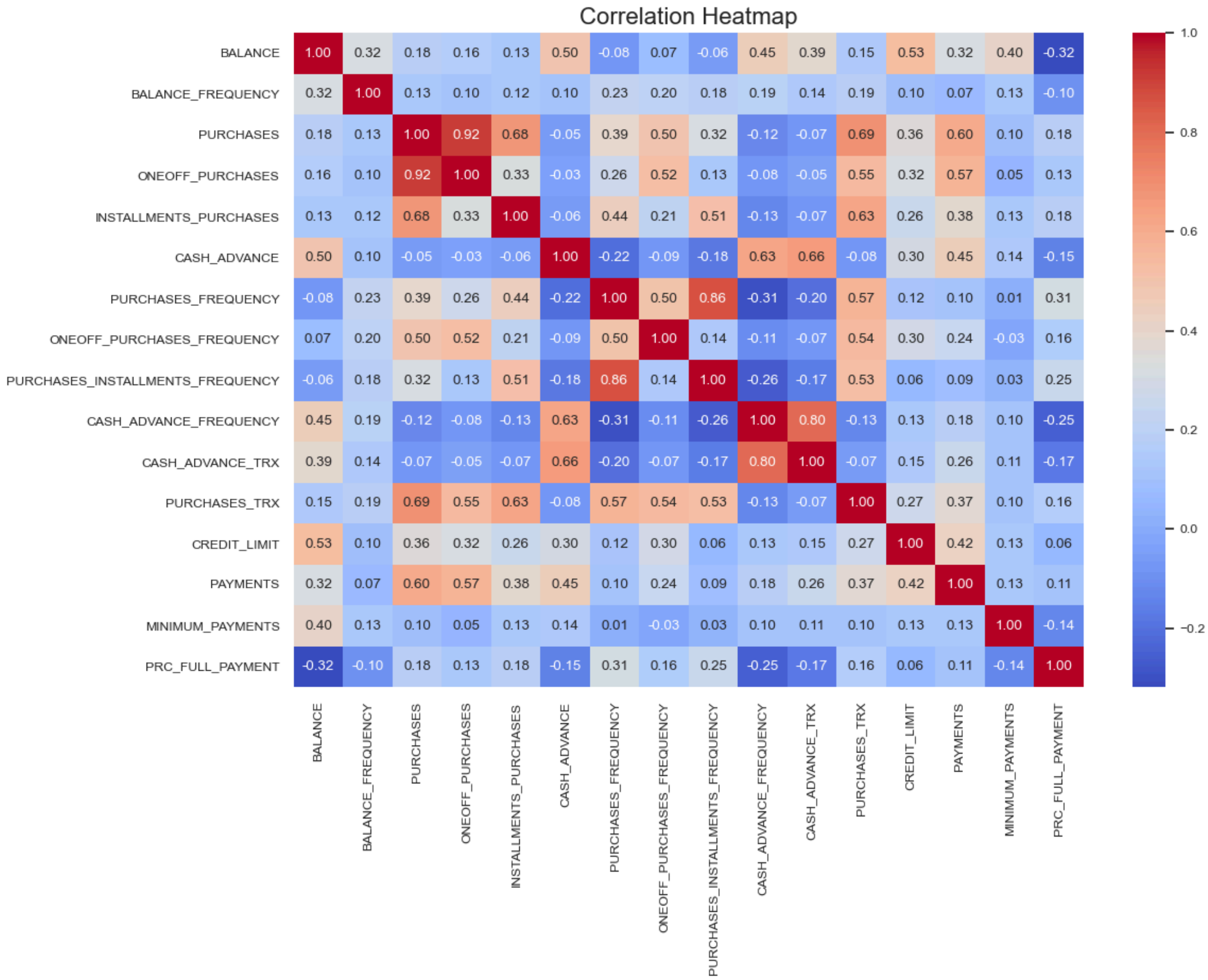
```
numeric_cols=df.columns[1:17] # Extracting the numerical columns
```

In [460…

```
# Coorelation heat map to better understand the relation between the features
data=df[numeric_cols]
plt.figure(figsize=(12,8))
corr = data.corr()
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
plt.title("Correlation Heatmap", fontsize=16)
```

Out[460…

```
Text(0.5, 1.0, 'Correlation Heatmap')
```



We can observe several very ***interesting correlations*** from the dataset:

- CREDIT_LIMIT and BALANCE (0.53) — A moderate positive relationship, indicating that customers with higher credit limits tend to maintain higher balances.
- CASH_ADVANCE and BALANCE (0.50) — Suggests that customers who take more cash advances generally have higher outstanding balances.
- PURCHASES and INSTALLMENTS_PURCHASES (0.69) — Shows a strong positive correlation, but it is still lower than the relationship between PURCHASES and ONEOFF_PURCHASES (0.92). This indicates that customers tend to prefer one-off purchases over installment purchases.
- PURCHASE_FREQUENCY and PURCHASE_INSTALLMENT_FREQUENCY (0.68) — Implies that customers who purchase more frequently also tend to use installment purchases more often.
- BALANCE and PRC_FULL_PAYMENT (-0.32) — A moderate negative correlation, suggesting that customers with higher balances are less likely to make full payments, possibly due to financial constraints or payment habits.

- PURCHASE_FREQUENCY and CASH_ADVANCE_FREQUENCY(-0.31) — Another moderate negative correlation, suggesting that customers pays advance cash frequently have less tendency to purchase product frequently.

Let's find some more insights

In [461…

```
# customers with high balence
highbalance_cst=df[df['BALANCE']>10000].loc[:,['CUST_ID','BALANCE']]
print(f'Total number of people who have higher balance is {highbalance_cst.sha
print(highbalance_cst.sort_values(by='BALANCE',ascending=False))
zero_balance=df[df['BALANCE']==0][['CUST_ID','BALANCE']]
print(zero_balance)
print(f'Total number of people who zero balance is {zero_balance.shape[0]}')
```

```
Total number of people who have higher balance is 66
     CUST_ID      BALANCE
138   C10144  19043.13856
4140  C14256  18495.55855
5488  C15642  16304.88925
6629  C16812  16259.44857
5281  C15429  16115.59640
...      ...          ...
5737  C15897  10243.14763
853   C10884  10131.00055
3510  C13610  10124.47214
4102  C14218  10116.70899
3491  C13588  10092.23573

[66 rows x 2 columns]
     CUST_ID  BALANCE
99    C10104      0.0
181   C10187      0.0
654   C10680      0.0
860   C10891      0.0
1131  C11171      0.0
...      ...      ...
8191  C18411      0.0
8329  C18550      0.0
8404  C18629      0.0
8484  C18714      0.0
8500  C18731      0.0

[80 rows x 2 columns]
Total number of people who zero balance is 80
```

Data Preprocessing

We will select all numerical features to train the model (**Kmean**)

In [462…

```
clusture_df=df[numeric_cols]  #Extarcting the required features
```

```
clusture_df
```

Out[462…

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	IN
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	
...	...	...	...	...	
8945	28.493517	1.000000	291.12	0.00	
8946	19.183215	1.000000	300.00	0.00	
8947	23.398673	0.833333	144.40	0.00	
8948	13.457564	0.833333	0.00	0.00	
8949	372.708075	0.666667	1093.25	1093.25	

8950 rows × 16 columns

Outliers Detection and Removal

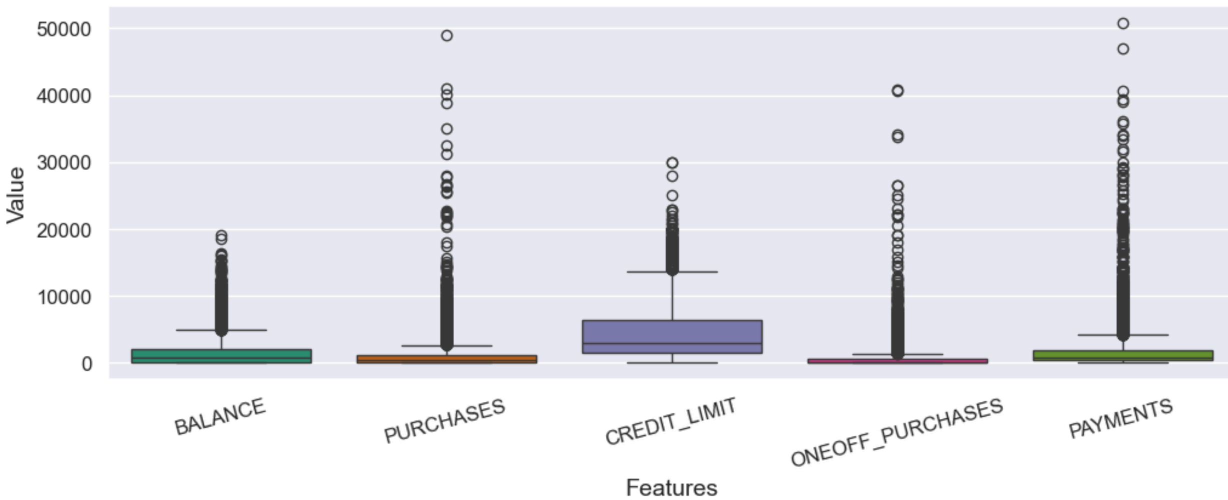
Visualizing the Outliers: To visualize the outliers, we use a boxplot.

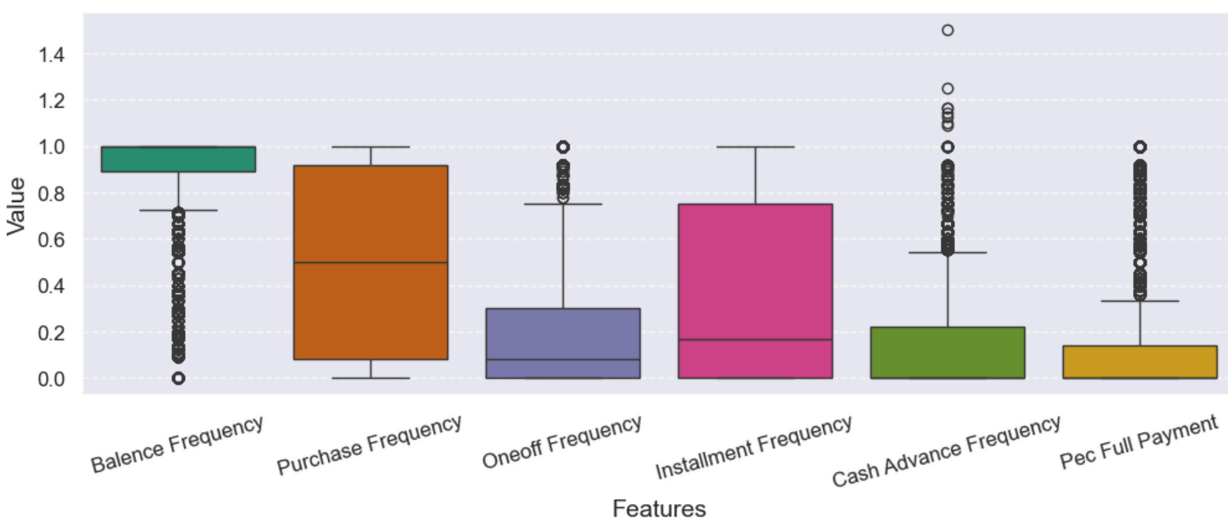
Concept: Outliers are data points whose values are significantly higher or lower than most of the observations in the dataset. In a boxplot, any data point that lies outside the whiskers (beyond 1.5 × IQR from the first or third quartile) is considered an outlier. These points appear as individual dots or markers, clearly indicating values that deviate from the general data distribution.

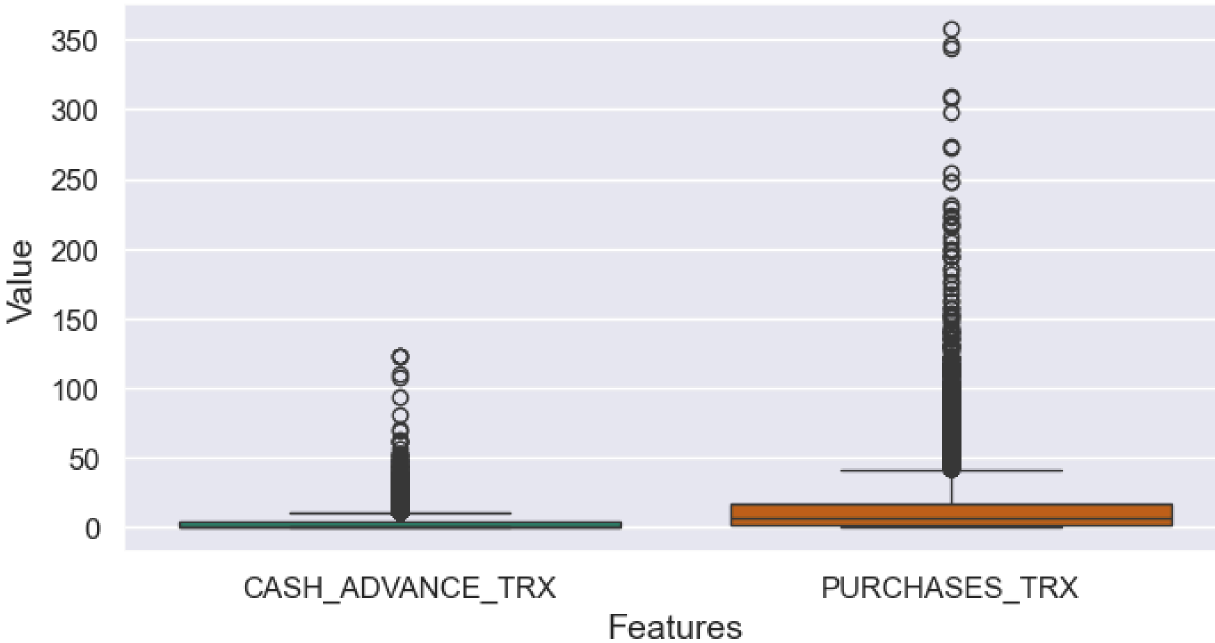
In [463…

```
'''Here We can't Plot all features in a single plot because the have significa
   frequency have range between 0 to 1. While Transaction related features hav
'''
# Boxplot of features related to Balence and Cash
plt.figure(figsize=(12,4))
sns.boxplot(data=clusture_df[['BALANCE', 'PURCHASES', 'CREDIT_LIMIT','ONEOFF_P
plt.xticks(fontsize=12,rotation=15)
plt.yticks(fontsize=12)
plt.ylabel("Value", fontsize=14)
plt.xlabel("Features", fontsize=14)
plt.show()
# Boxplot of features related to  frequency
plt.figure(figsize=(12,4))
sns.boxplot(data=clusture_df[['BALANCE_FREQUENCY','PURCHASES_FREQUENCY','ONEOF
labels=['Balence Frequency','Purchase Frequency','Oneoff Frequency','Installme
plt.xticks(ticks=range(len(labels)), labels=labels, fontsize=12, rotation=15)
```

```
plt.yticks(fontsize=12)
plt.ylabel("Value", fontsize=14)
plt.xlabel("Features", fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()
# Boxplot of features related to transaction
plt.figure(figsize=(8,4))
sns.boxplot(data=clusture_df[['CASH_ADVANCE_TRX', 'PURCHASES_TRX']],palette='D
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.ylabel("Value", fontsize=14)
plt.xlabel("Features", fontsize=14)
plt.show()
```







Removal of Outliers: To remove outliers we will use z-score method. Z-score method is a very standard method to calculate the distance of any point from mean.

concept: We calculate the z-score as Z-score=(x−μ)/σ, here μ is the mean and σ is the standard deviation. If Z-score is >3 then We generally consider as an outlier.

In [464…
```
z_scors=np.abs(stats.zscore(clusture_df))
z_scors
```

Out[464…
```
array([[0.73198937, 0.24943448, 0.42489974, ..., 0.52897879, 0.3024    ,
        0.52555097],
       [0.78696085, 0.13432467, 0.46955188, ..., 0.81864213, 0.09749953,
        0.2342269 ],
       [0.44713513, 0.51808382, 0.10766823, ..., 0.38380474, 0.0932934 ,
        0.52555097],
       ...,
       [0.7403981 , 0.18547673, 0.40196519, ..., 0.5706145 , 0.32687479,
        0.32919999],
       [0.74517423, 0.18547673, 0.46955188, ..., 0.58053567, 0.33830497,
        0.32919999],
       [0.57257511, 0.88903307, 0.04214581, ..., 0.57686873, 0.3243581 ,
        0.52555097]])
```

In [465…
```
filtered_zscore=(z_scors<3).all(axis=1)
filtered_df=clusture_df[filtered_zscore]
filtered_df
```

Out[465…

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	IN
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	
...	...	...	...	...	
8945	28.493517	1.000000	291.12	0.00	
8946	19.183215	1.000000	300.00	0.00	
8947	23.398673	0.833333	144.40	0.00	
8948	13.457564	0.833333	0.00	0.00	
8949	372.708075	0.666667	1093.25	1093.25	

7786 rows × 16 columns

Data Standarization

The process of converting data into a consistent format to improve data quality, enable easier analysis, and facilitate integration across different systems.

Here, we will use StandardScaler module from Scikit-learn library to standarize our data.

In [466…

```
features=StandardScaler().fit(clusture_df.values)
scaled=features.transform(clusture_df.values)
scaled_features=pd.DataFrame(scaled,columns=clusture_df.columns)
scaled_features
```

Out[466…

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INST
0	-0.731989	-0.249434	-0.424900	-0.356934	
1	0.786961	0.134325	-0.469552	-0.356934	
2	0.447135	0.518084	-0.107668	0.108889	
3	0.049099	-1.016953	0.232058	0.546189	
4	-0.358775	0.518084	-0.462063	-0.347294	
...	...	...	...	...	
8945	-0.737950	0.518084	-0.333293	-0.356934	
8946	-0.742423	0.518084	-0.329136	-0.356934	
8947	-0.740398	-0.185477	-0.401965	-0.356934	
8948	-0.745174	-0.185477	-0.469552	-0.356934	
8949	-0.572575	-0.889033	0.042146	0.301732	

8950 rows × 16 columns

Dimensional Reduction

Dimensionality reduction is a technique used to reduce the number of features or variables in a dataset while preserving its most important information. Here, we will use Principle Component Analysis technique to reuduce our 17 numerical features into 2 dimensions.

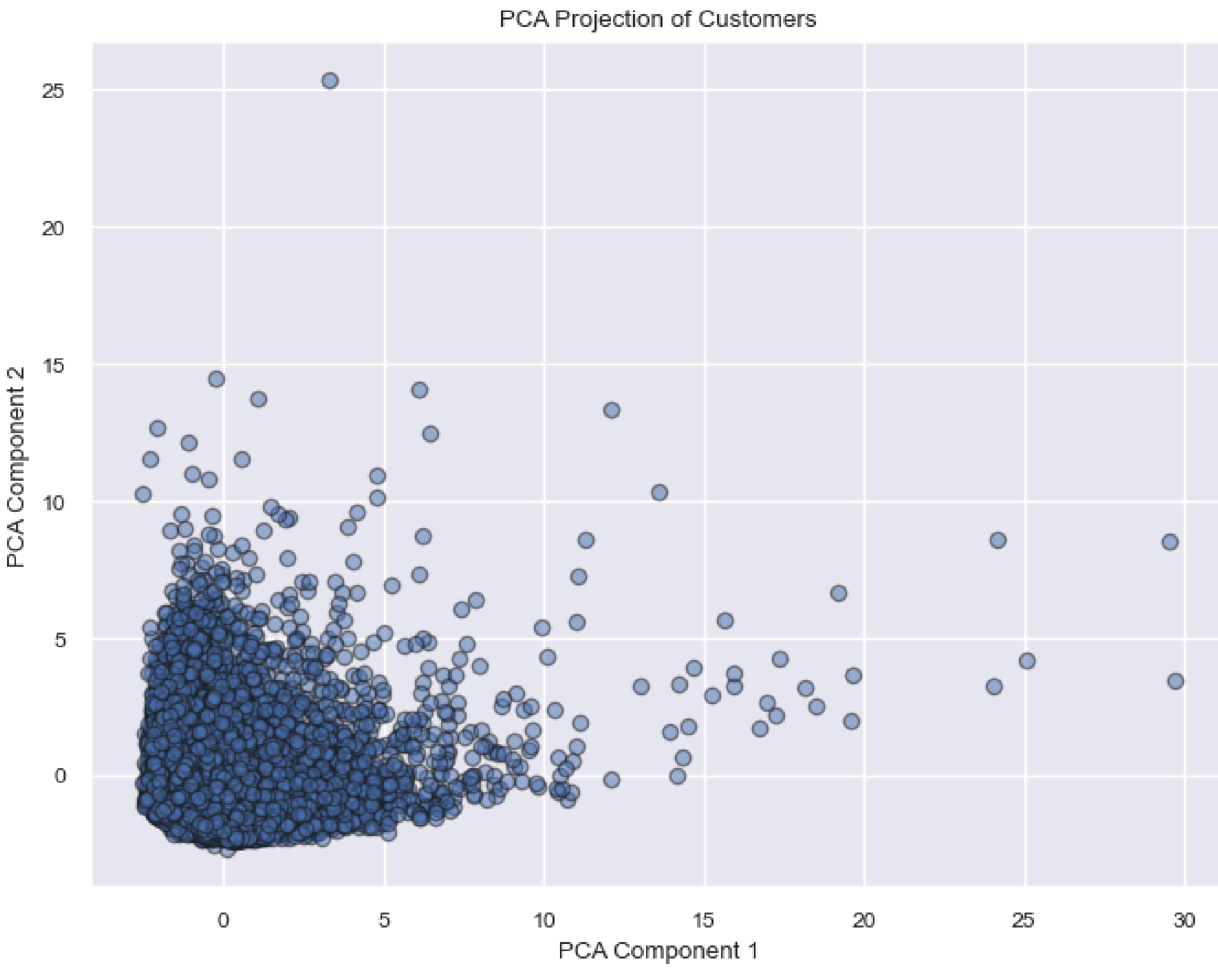
In [467…

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_features)

plt.figure(figsize=(8,6))
plt.scatter(pca_result[:,0], pca_result[:,1],edgecolor='k', alpha=0.5)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.title("PCA Projection of Customers")
```

Out[467…

```
Text(0.5, 1.0, 'PCA Projection of Customers')
```



Clustering the Data

Algorithm: To determine the optimal number of clusters, we use the K-Means algorithm.

Concept : K-Means is one of the most widely used techniques for clustering large datasets. The process begins by randomly selecting a set of k centroids. Each data point is then assigned to the cluster with the nearest centroid. The centroid of each cluster is recalculated as the mean of all points in that cluster. This process of assignment and centroid update continues until the centroids and clusters remain stable without significant changes.

Before clustering, we need to determine the optimal number of clusters. To achieve this, we use the Elbow method. In the Elbow graph, the point where the rate of decrease in the within-cluster variance slows down noticeably — forming an “elbow” shape — is considered the optimal number of clusters.

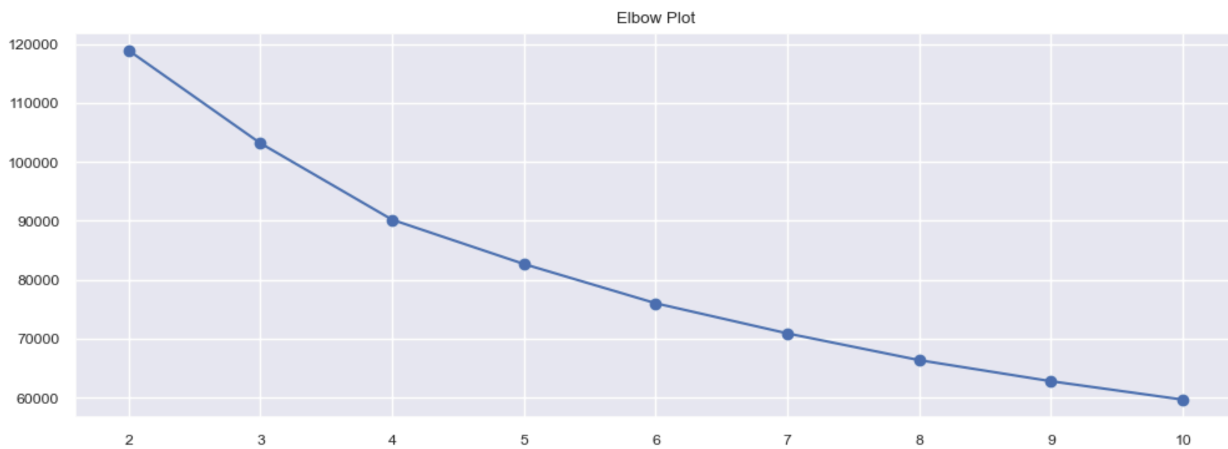
In [468…

```
#function to plot Elbow grarh
# for our dataset it is not logical to make more than 10 clustres. So, the ran
```

```
def find_k(df_scaled, k_range=range(2,11)):
    inertias=[]
    for k in k_range:
        kmeans = KMeans(n_clusters=k, n_init=10)
        labels = kmeans.fit_predict(df_scaled)
        inertias.append(kmeans.inertia_)
    plt.figure(figsize=(12,4))
    plt.plot(k_range, inertias, marker='o')
    plt.title('Elbow Plot')
    plt.show()
    return pd.DataFrame({'k': list(k_range), 'inertia': inertias})
```

In [469…

```
#calling the function
wcss=find_k(scaled_features)
wcss
```



Out[469…

	k	inertia
0	2	118899.221345
1	3	103156.464468
2	4	90184.520984
3	5	82666.408273
4	6	76009.424103
5	7	70896.416863
6	8	66361.200957
7	9	62785.397867
8	10	59666.483564

In our Elbow graph we can notice the sharp drop at 4. So, the optimal number of clustre will be 4.

In [470…

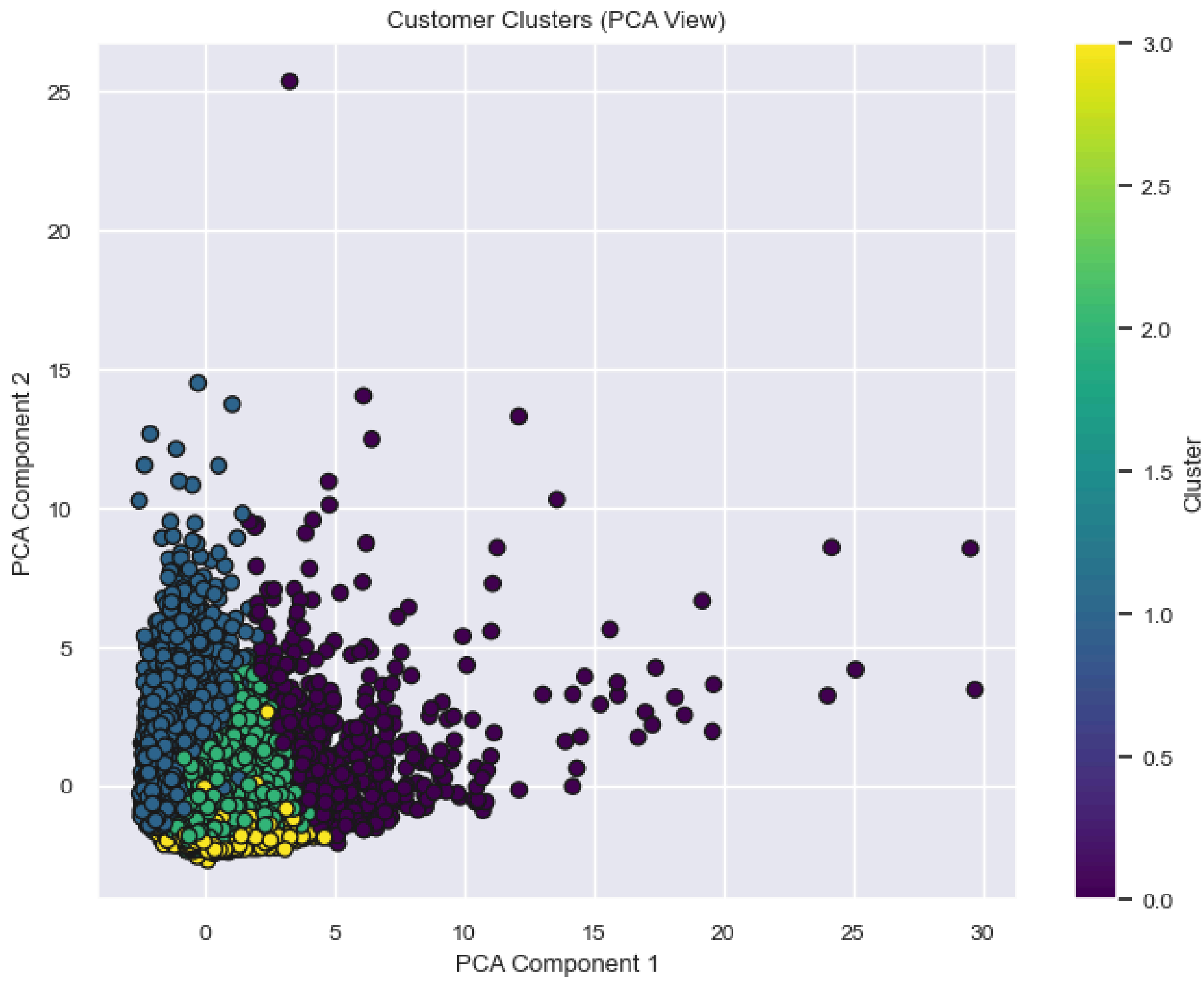
```
# Fit the standard data into the model
kmeans = KMeans(n_clusters=4, random_state=42)
```

```python
df['Cluster'] = kmeans.fit_predict(scaled_features)
df['Cluster']
```

Out[470…
```
0       1
1       1
2       2
3       1
4       1
       ..
8945    3
8946    2
8947    2
8948    1
8949    2
Name: Cluster, Length: 8950, dtype: int32
```

In [471…
```python
#Plotting the clusters
plt.figure(figsize=(8,6))
plt.scatter(pca_result[:,0], pca_result[:,1], c=df['Cluster'], cmap='viridis',
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.title("Customer Clusters (PCA View)")
plt.colorbar(label="Cluster")
```

Out[471…
```
<matplotlib.colorbar.Colorbar at 0x1f2dad02550>
```



In [472…

```
cluster_summary = df[numeric_cols].groupby(df['Cluster']).mean()
cluster_summary
```

Out[472…

Cluster	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
0	4171.811246	0.987145	6867.332195	4504.861408
1	1909.868290	0.846203	229.748585	188.098225
2	1240.828922	0.932336	1117.398118	562.706274
3	114.247971	0.813402	1138.700474	543.905534

Explanation of Clusters

=> Here, I am considering the currency is $.

Cluster 0:

- High average balance (~$4,172)
- High purchase amount ($6,867), mostly one-off purchases ($4,505)
- High purchase frequency (~94%), both one-off and installments
- Credit limit is highest (~$9,949)
- Payments are large ($7,612), with decent minimum payments ($2,382)
- Full payment ratio ~21% — they sometimes carry balances

Conclusion: Premium customers, high-value spenders, mix of one-off and installment purchases, strong revenue generators for the bank.

Cluster 1:

- Low balance (~$1,910)
- Very low purchases (~$230) and purchase frequency (~13%)
- Low credit limit (~$4,199)
- Cash advance is significant (~$1,566) relative to their spend
- Full payment ratio is lowest (~3%)

Conclusion: Low-engagement customers, primarily using cash advances, minimal card spending. Possibly a credit risk group.

Cluster 2:

Moderate balance (~$1,241)

- Purchases ($1,117), almost evenly split between one-off ($563) and installments (~$555)
- Purchase frequency (~84%) is high
- Low cash advance (~$360)
- Credit limit (~$3,955)
- Full payment ratio (~5.6%)

Conclusion: Regular users who make moderate purchases, balanced payment style, mostly safe customers.

Cluster 3:

- Very low balance (~$114)
- Purchases ($1,139), mostly one-off ($544) and installments (~$595)
- High full payment ratio (~78%) — they pay off almost everything
- Credit limit (~$4,661)
- Very low cash advance (~$62)

Conclusion: Low debt customers, make purchases and clear dues fully, very low risk.

Final conclusion : We have analized the data and segregate our customers into four different clustures according to past, present, future conditions.

Cluster Summary

- **Cluster 0** : High spenders with balanced usage, Premium customers. We can call them ***The Big Spenders***.
- **Cluster 1** : Low activity, low spend,Low-engagement customers. We can call them ***The Dormants***.
- **Cluster 2** : Moderate spenders with installment preference,regular users. We can call them ***The Potential***.
- **Cluster 3** : Low balance but high one-off purchases. We can call them ***The Cash Advance Seekers***.

Business Insights

- ***The Big Spenders*** focuses for premium offers, loyalty programs, and high-value rewards — they’re the most profitable.
- ***The Dormants*** need to educate about card benefits, encourage purchases instead of cash advances, possibly reduce risk exposure.
- ***The Potential*** need to encourage more one-off spending through targeted promotions and rewards.
- ***The Cash Advance Seekers*** maintains engagement with cashback offers, but profitability is lower since they pay in full.