

Credit Card Dataset:

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import seaborn as sns
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings("ignore")
```

```
In [2]: df = pd.read_csv("CC_GENERAL.csv")
df.head()
```

```
Out[2]:
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.1666
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.0000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.0000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.0833
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.0833

```
In [3]: # Drop any irrelevant columns (e.g., customer IDs)
df.drop('CUST_ID', axis=1, inplace=True)
df.head()
```

Out[3]:	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEO
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	

Handling missing values:

```
In [4]: df.isnull().sum()
```

```
Out[4]: 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 [5]: df['CREDIT_LIMIT'].fillna(df['CREDIT_LIMIT'].median(),inplace=True)
df['MINIMUM_PAYMENTS'].fillna(df['MINIMUM_PAYMENTS'].median(),inplace=True)
print (df.isnull().sum())
```

```
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 [6]: def plot(data):
        for i,j in zip(df.columns, range(len(df.columns))):
            plt.figure(figsize=(8,2))

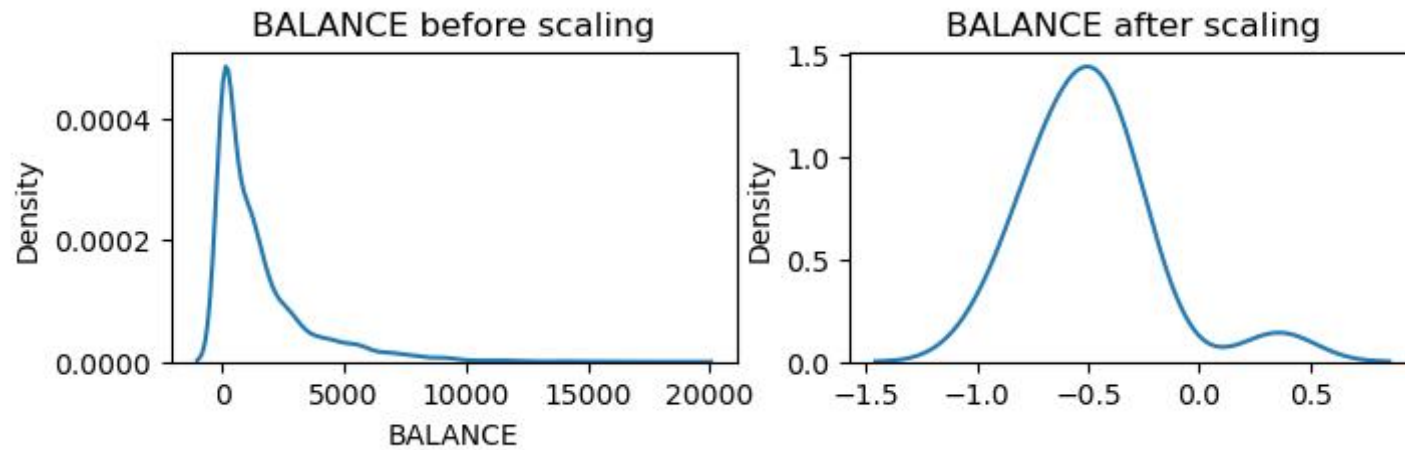
            # 121 for 1 row, 2 columns, 1st position
            plt.subplot(121)
            sns.kdeplot(df[i])
            plt.title(i+' before scaling')

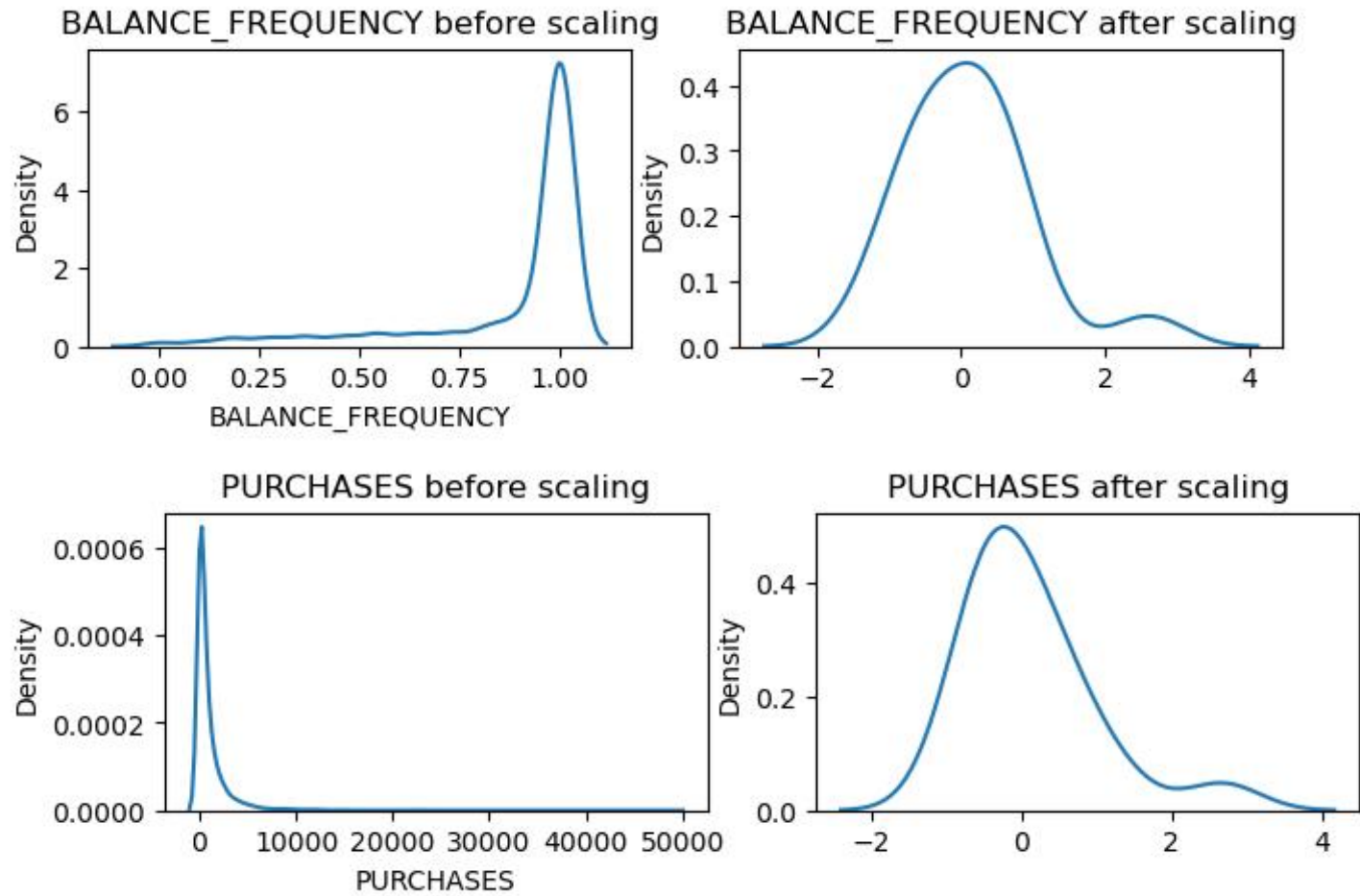
            # 122 for 1 row, 2 columns, 2nd position
            plt.subplot(122)
            sns.kdeplot(data[j])
            plt.title(i+' after scaling')
```

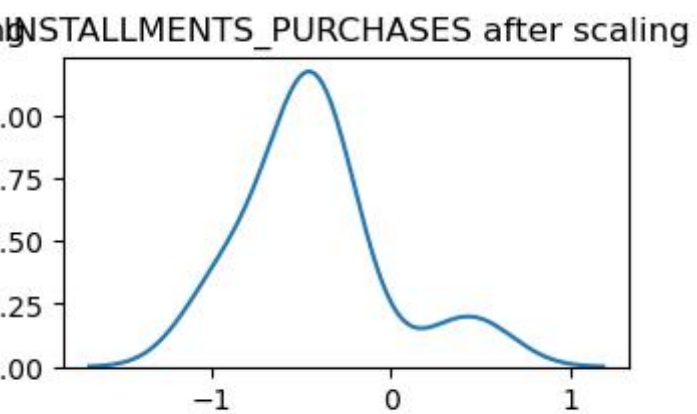
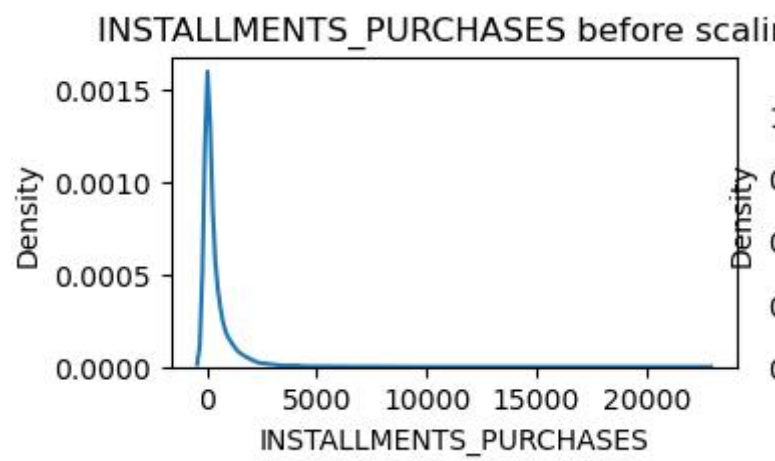
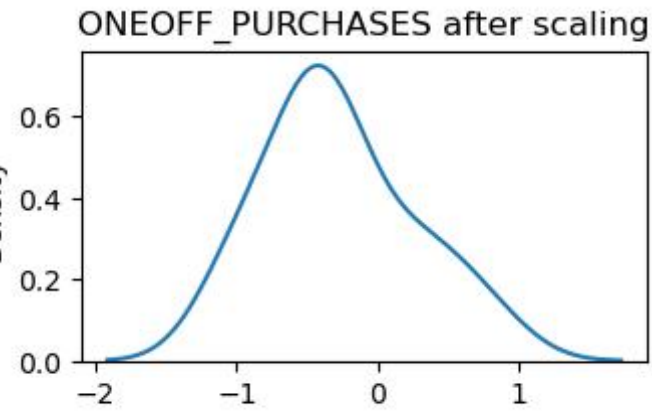
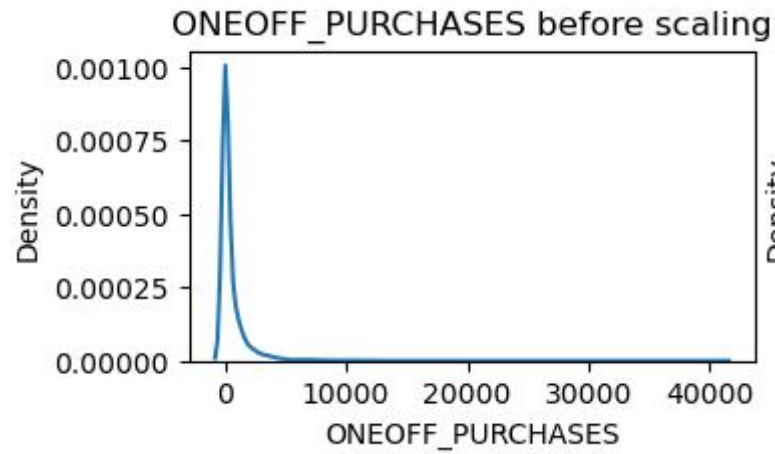
```
In [7]: scaler = StandardScaler()
        scaled_data = scaler.fit_transform(df)
        print(scaled_data)
```

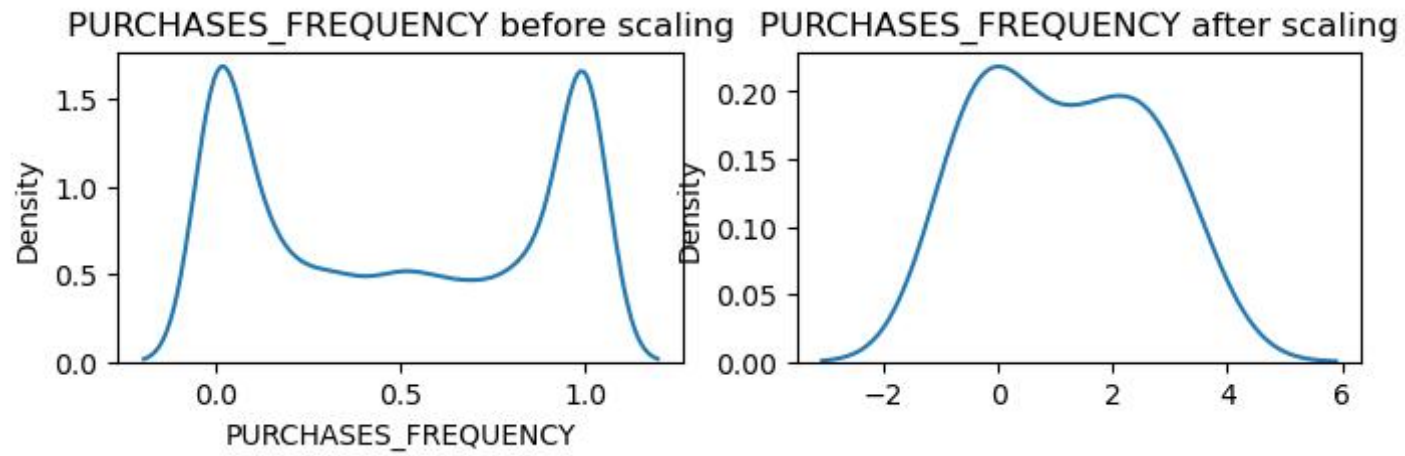
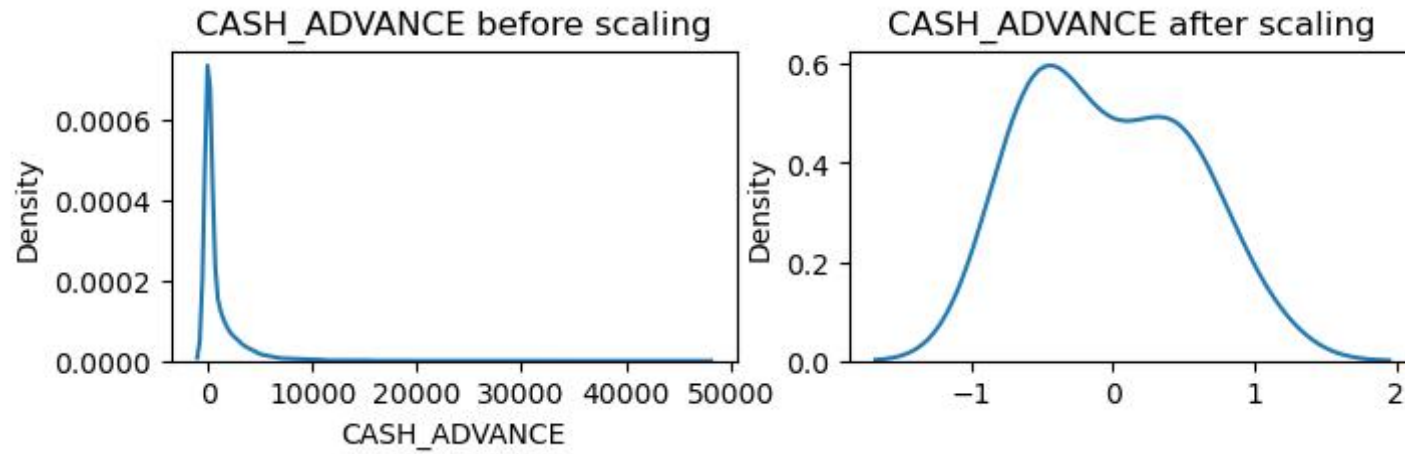
```
[[ -0.73198937 -0.24943448 -0.42489974 ... -0.3024 -0.52555097  
 0.36067954]  
 [ 0.78696085 0.13432467 -0.46955188 ... 0.09749953 0.2342269  
 0.36067954]  
 [ 0.44713513 0.51808382 -0.10766823 ... -0.0932934 -0.52555097  
 0.36067954]  
 ...  
 [-0.7403981 -0.18547673 -0.40196519 ... -0.32687479 0.32919999  
 -4.12276757]  
 [-0.74517423 -0.18547673 -0.46955188 ... -0.33830497 0.32919999  
 -4.12276757]  
 [-0.57257511 -0.88903307 0.04214581 ... -0.3243581 -0.52555097  
 -4.12276757]]
```

```
In [8]: plot(scaled_data)
```

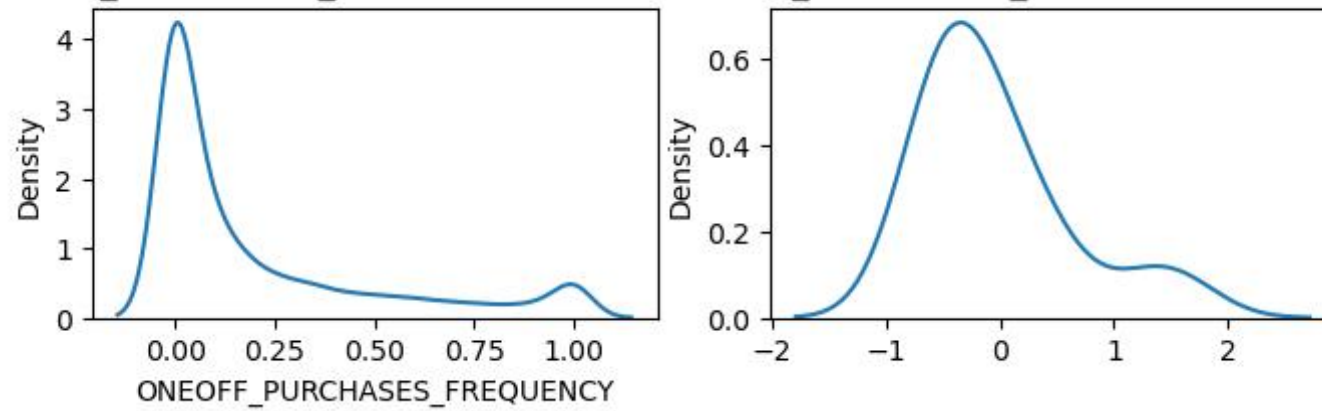




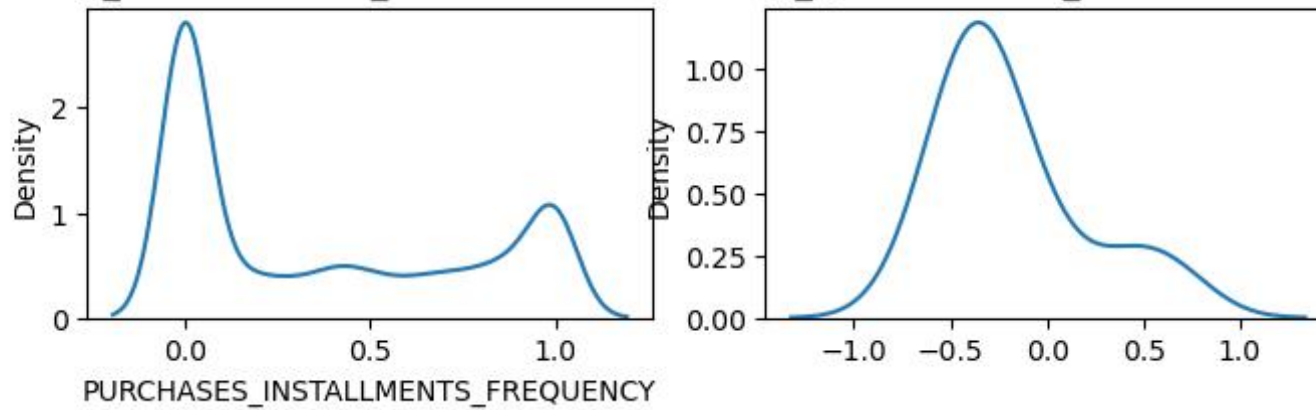




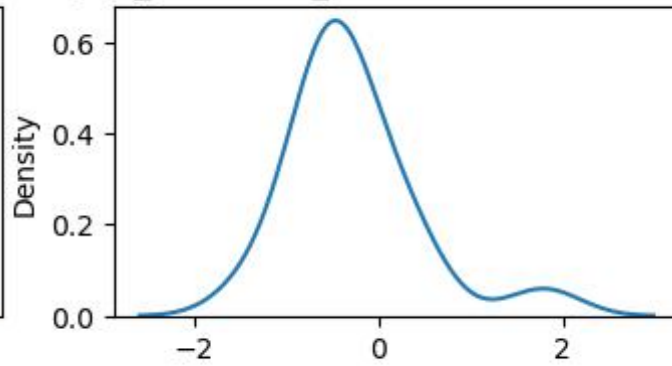
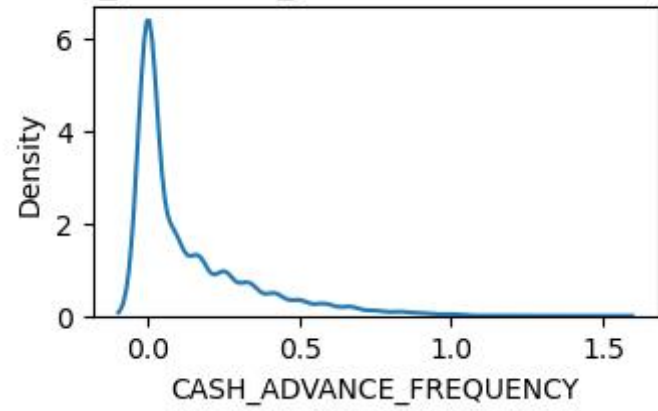
ONEOFF_PURCHASES_FREQUENCY before scaling ONEOFF_PURCHASES_FREQUENCY after scaling



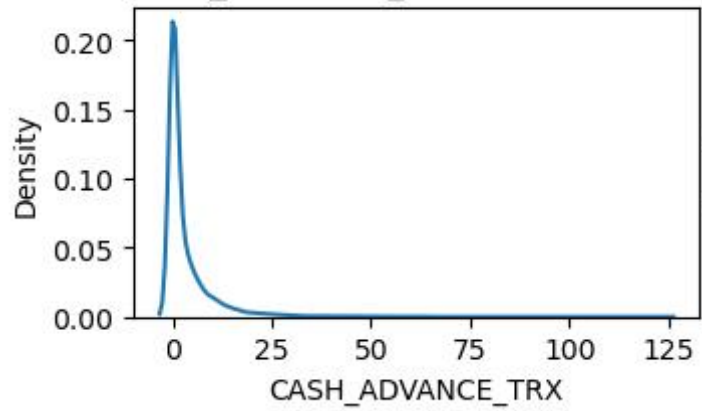
PURCHASES_INSTALLMENTS_FREQUENCY before scaling PURCHASES_INSTALLMENTS_FREQUENCY after scaling



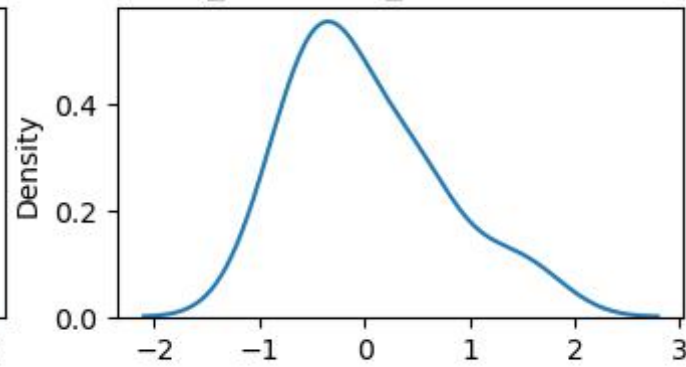
CASH_ADVANCE_FREQUENCY before scaling CASH_ADVANCE_FREQUENCY after scaling

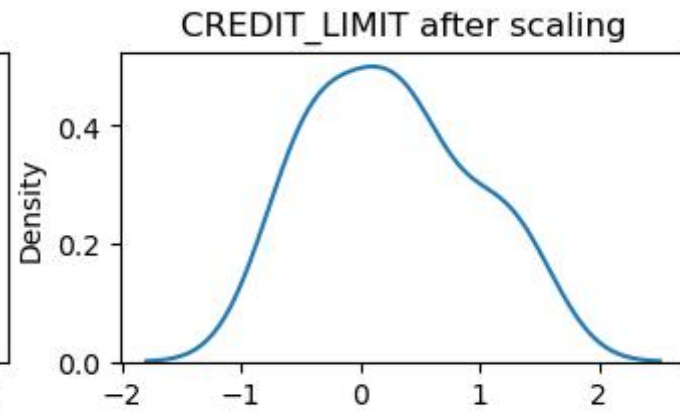
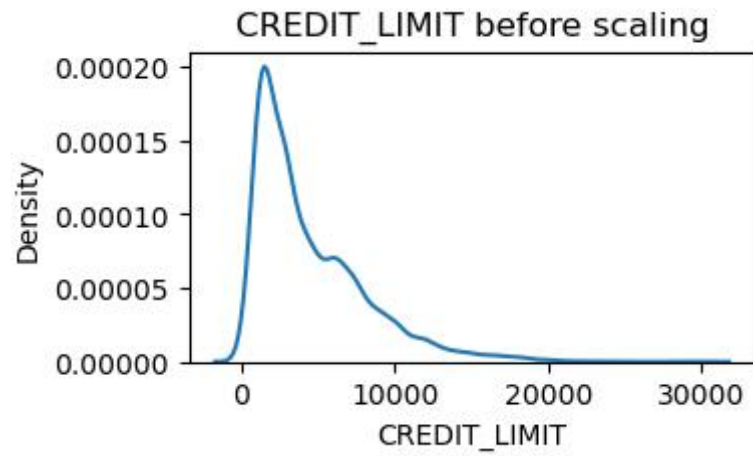
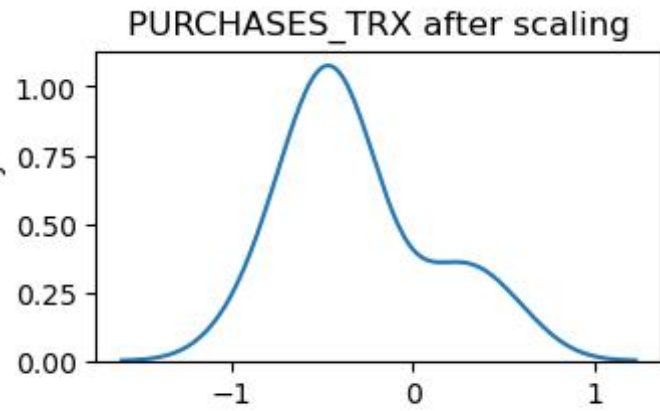
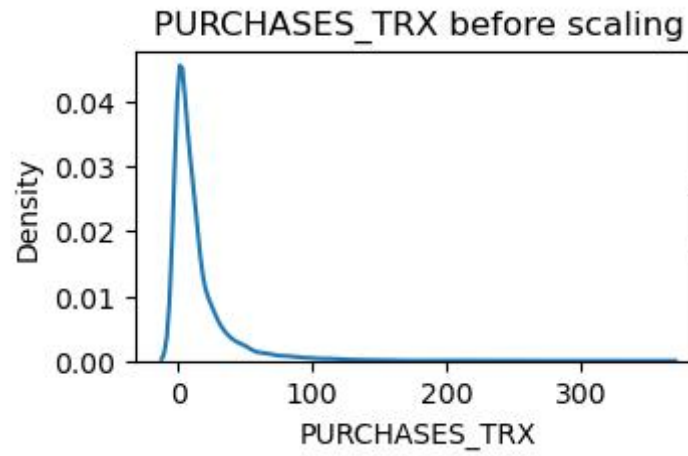


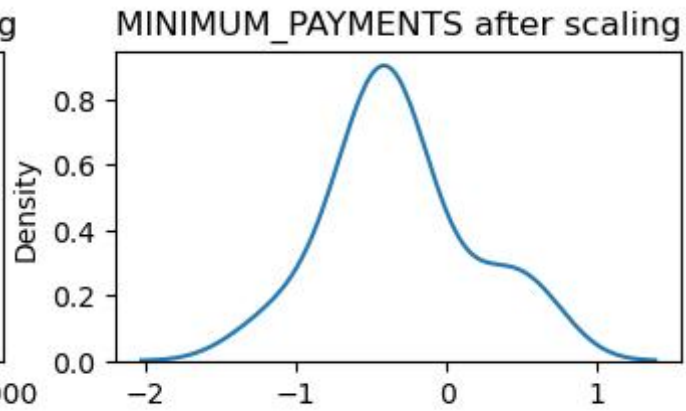
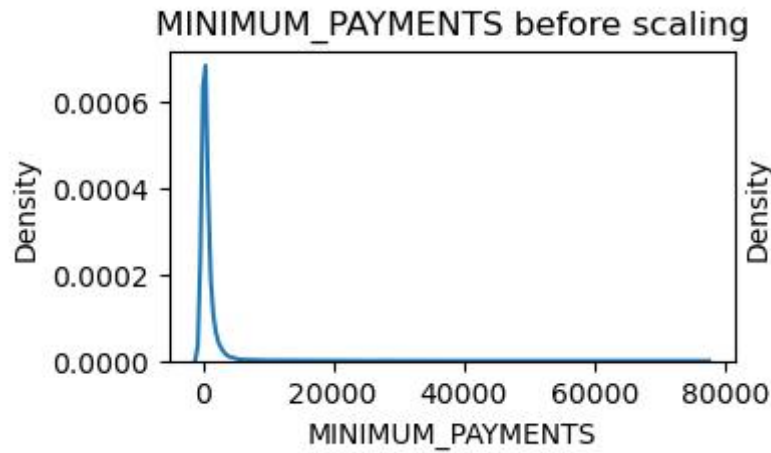
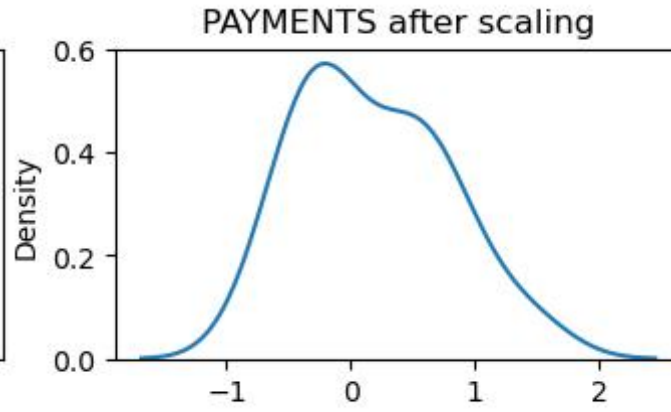
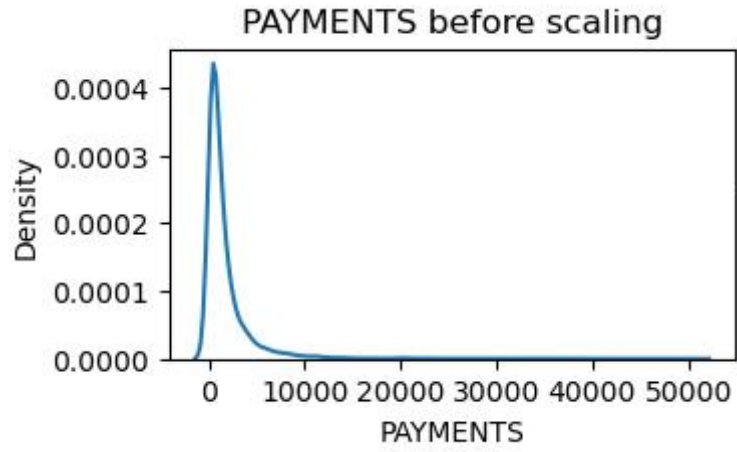
CASH_ADVANCE_TRX before scaling

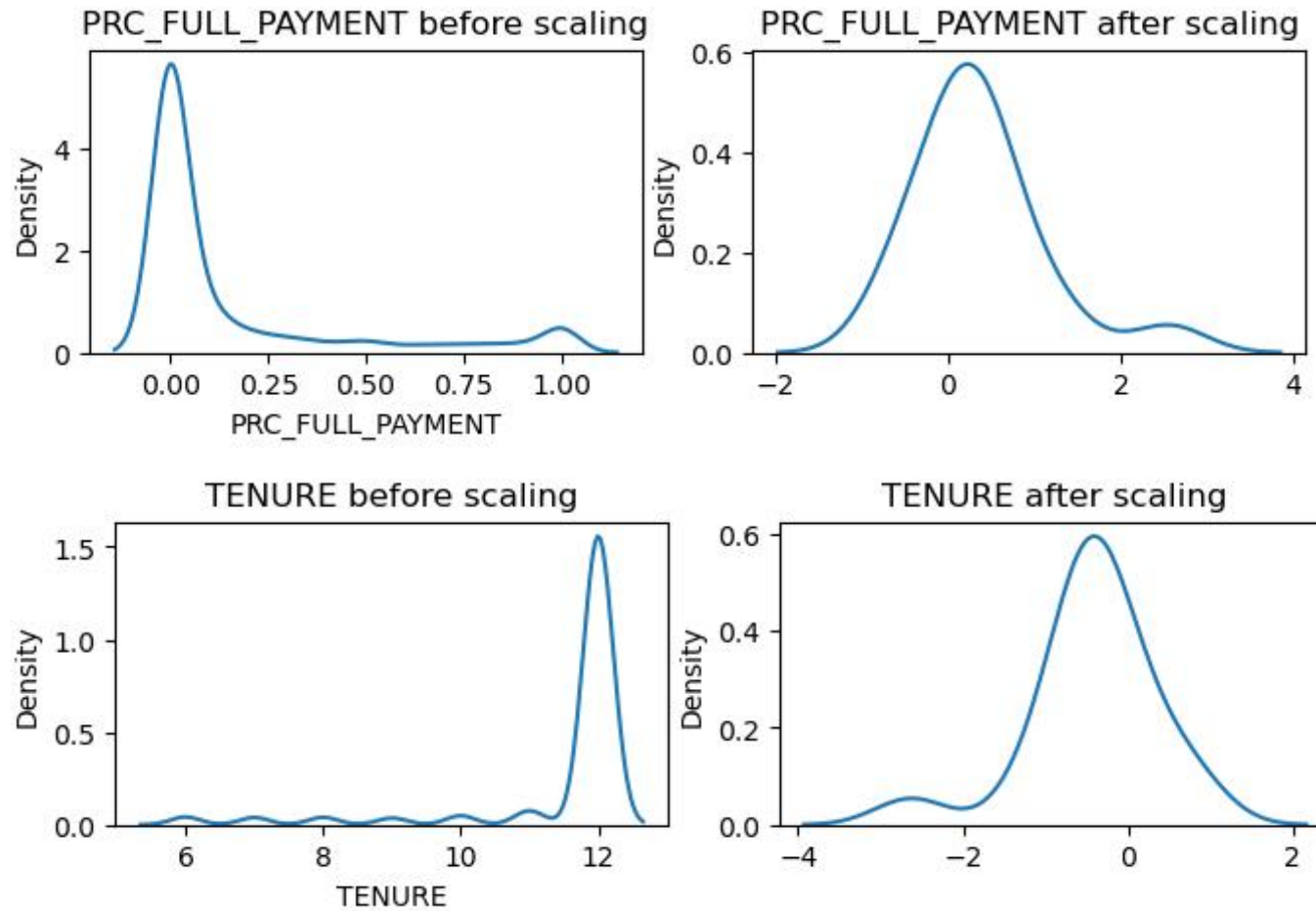


CASH_ADVANCE_TRX after scaling







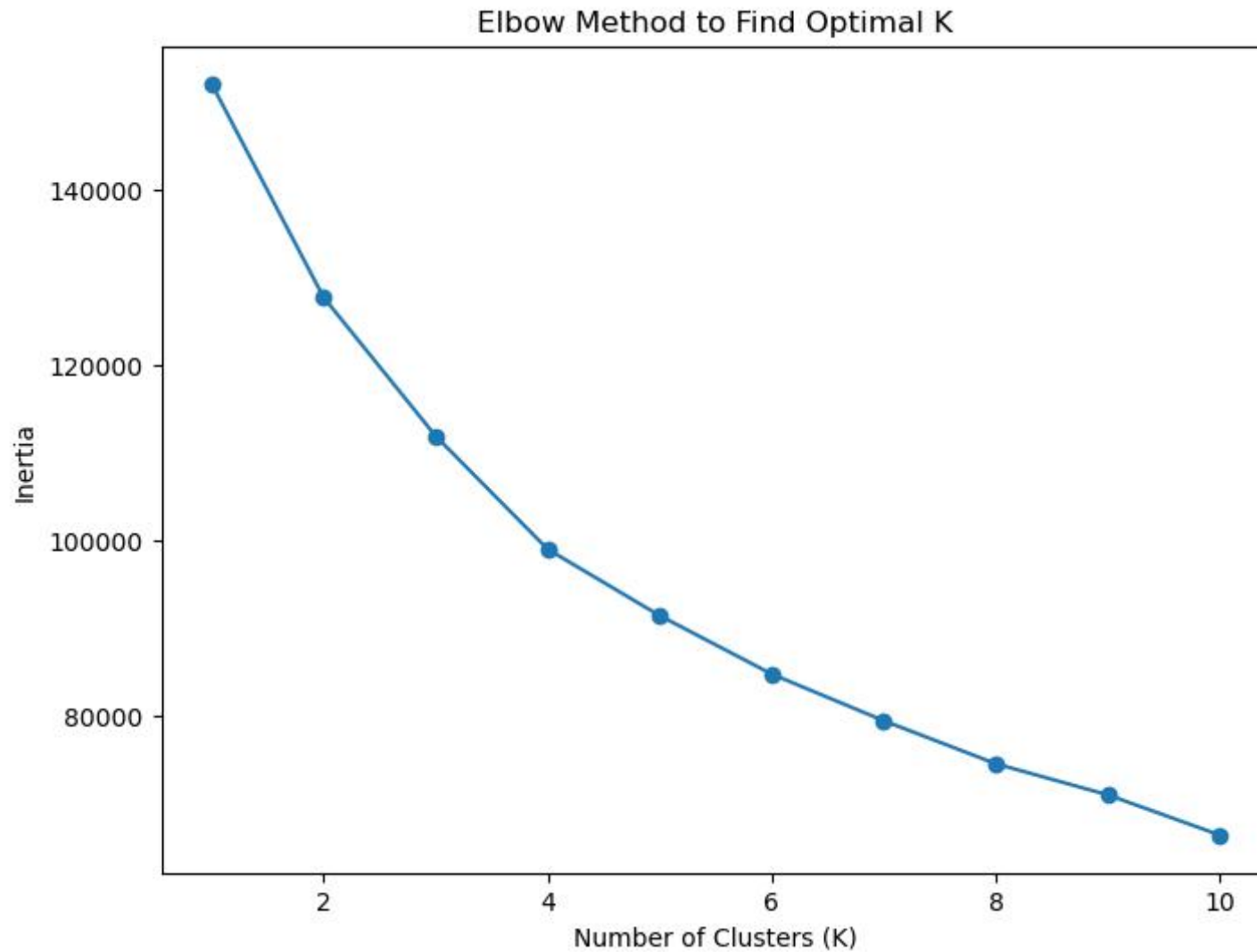


```
In [9]: # To find the optimal number of clusters (K)
# We'll use the 'inertia' attribute of KMeans, which represents the sum of squared distances
# of samples to their closest cluster center
inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)
    inertia.append(kmeans.inertia_)

# Plot the inertia values to find the elbow point
plt.figure(figsize=(8, 6))
```

```
plt.plot(k_range, inertia, marker='o')  
plt.xlabel('Number of Clusters (K)')  
plt.ylabel('Inertia')  
plt.title('Elbow Method to Find Optimal K')  
plt.show()
```



```
In [10]: # Let's say we choose K=4 (as shown by the elbow plot)  
  
# Create and fit the KMeans model with the chosen K value
```

```
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(scaled_data)

# Add the cluster labels back to the original dataframe
df['Cluster'] = kmeans.labels_
```

```
In [11]: # We can now analyze the segments or clusters formed
# For example, we can use groupby to see the average spending behavior of each cluster
cluster_means = df.groupby('Cluster').mean()

# We can also visualize the clusters using scatter plots
# Choose two features for visualization (e.g., 'Spending Score' vs. 'Income')
feature1 = 'PURCHASES'
feature2 = 'CREDIT_LIMIT'

plt.figure(figsize=(10, 6))
for cluster_label in df['Cluster'].unique():
    cluster_data = df[df['Cluster'] == cluster_label]
    plt.scatter(cluster_data[feature1], cluster_data[feature2], label=f'Cluster {cluster_label}')

plt.xlabel(feature1)
plt.ylabel(feature2)
plt.title('Credit Card Data Clusters')
plt.legend()
plt.show()
```

