PROJECT TITLE: CREDIT CARD USAGE SEGMENTATION

Project Overview:

This project focuses on developing unsupervised learning models for customer segmentation based on credit card usage data. The models aim to provide insights into distinct customer segments, improve credit risk assessment, and optimize marketing strategies. Deliverables include segmentation models, comprehensive data visualizations, and a comparative analysis of various algorithms.

STEP 1: ALL THE NECESSARY LIBRARIES REQUIRED FOR THE PROJECT ARE IMPORTED HERE

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

STEP 2: FILE INGESTION AND DATASET INSPECTION IS PERFORMED

Here we are reading the 'Customer Data' csv file and storing the data to a new dataframe "data".

```
data = pd.read_csv('Customer Data.csv')
```

Visualized dataset for determining the number of rows and columns.

```
data.shape

(8950, 18)
```

Visualizing dataset for the first 10 rows.

```
data.head(10)
```

```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 8950,\n  \"fields\":
[\n    {\n      \"column\": \"CUST_ID\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 8950,\n
\"samples\": [\n          \"C17875\",\n          \"C16296\",\n
\"C17219\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BALANCE\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 2081.531879456554,\n        \"min\": 0.0,\n
\"max\": 19043.13856,\n        \"num_unique_values\": 8871,\n
\"samples\": [\n          325.024091,\n          965.514081,\n
203.499251\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BALANCE_FREQUENCY\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.23690400268475698,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 43,\n
\"samples\": [\n          0.428571,\n          0.8,\n          0.2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"PURCHASES\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2136.6347818728423,\n        \"min\": 0.0,\n        \"max\":
49039.57,\n        \"num_unique_values\": 6203,\n        \"samples\":
[\n          1361.65,\n          2485.54,\n          2580.63\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ONEOFF_PURCHASES\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1659.887917437827,\n        \"min\":
0.0,\n        \"max\": 40761.25,\n        \"num_unique_values\":
4014,\n        \"samples\": [\n          25.62,\n          13007.07,\n
185.63\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"INSTALLMENTS_PURCHASES\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 904.3381151753365,\n
\"min\": 0.0,\n        \"max\": 22500.0,\n
\"num_unique_values\": 4452,\n        \"samples\": [\n
228.56,\n          255.58,\n          729.6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CASH_ADVANCE\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2097.1638766431465,\n        \"min\": 0.0,\n        \"max\":
47137.21176,\n        \"num_unique_values\": 4323,\n
\"samples\": [\n          4473.3497,\n          520.844673,\n
3968.684047\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PURCHASES_FREQUENCY\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.40137074736905376,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 47,\n
\"samples\": [\n          0.8,\n          0.555556,\n          0.2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"ONEOFF_PURCHASES_FREQUENCY\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.2983360651847212,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 47,\n        \"samples\": [\n
0.909091,\n          0.625,\n          0.181818\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\":
\"PURCHASES_INSTALLMENTS_FREQUENCY\",\n      \"properties\": {\n
```

```
\"dtype\": \"number\",\n        \"std\": 0.39744777974542483,\n
\"min\": 0.0,\n        \"max\": 1.0,\n        \"num_unique_values\":
47,\n        \"samples\": [\n          0.857143,\n          0.222222,\
n          0.142857\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CASH_ADVANCE_FREQUENCY\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.20012138814749122,\n
\"min\": 0.0,\n        \"max\": 1.5,\n        \"num_unique_values\":
54,\n        \"samples\": [\n          0.222222,\n          0.818182,\
n          0.7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CASH_ADVANCE_TRX\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6,\n        \"min\": 0,\n
\"max\": 123,\n        \"num_unique_values\": 65,\n
\"samples\": [\n          47,\n          61,\n          0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"PURCHASES_TRX\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
24,\n        \"min\": 0,\n        \"max\": 358,\n
\"num_unique_values\": 173,\n        \"samples\": [\n          162,\n
216,\n          79\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CREDIT_LIMIT\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3638.8157254985426,\n        \"min\":
50.0,\n        \"max\": 30000.0,\n        \"num_unique_values\": 205,\
n        \"samples\": [\n          9000.0,\n          3000.0,\n
300.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PAYMENTS\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2895.063756904579,\n        \"min\":
0.0,\n        \"max\": 50721.48336,\n        \"num_unique_values\":
8711,\n        \"samples\": [\n          810.671862,\n
5943.975673,\n          7079.1781\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MINIMUM_PAYMENTS\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2372.446606583975,\n        \"min\": 0.019163,\n        \"max\":
76406.20752,\n        \"num_unique_values\": 8636,\n
\"samples\": [\n          173.484575,\n          185.120378,\n
163.711273\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PRC_FULL_PAYMENT\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.29249919623387977,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 47,\n
\"samples\": [\n          0.583333,\n          0.272727,\n
0.1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"TENURE\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 6,\n        \"max\": 12,\n
```

```
\"num_unique_values\": 7,\n        \"samples\": [\n          12,\n
8,\n          7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data"}
```

Visualizing dataset for determining the Column data type and not-null value count of columns.

```
data.info()

<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
```

Visualized dataset for determining the summary statistical parameters like mean, standard deviation, percentile, count, minimum and maximum values for numerical columns in our Dataset.

```
data.describe()
```

```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 8,\n  \"fields\": [\
n    {\n      \"column\": \"BALANCE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 6590.634765473159,\n
\"min\": 0.0,\n        \"max\": 19043.13856,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
1564.4748276781006,\n          873.385231,\n          8950.0\
n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BALANCE_FREQUENCY\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3164.0501764110654,\n        \"min\":
0.0,\n        \"max\": 8950.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          8950.0,\n          0.8772707255865921,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PURCHASES\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16909.167916503804,\n        \"min\":
0.0,\n        \"max\": 49039.57,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          1003.2048335195531,\n          361.28,\n
8950.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ONEOFF_PURCHASES\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 14139.379689427924,\n        \"min\":
0.0,\n        \"max\": 40761.25,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          8950.0,\n          592.4373709497207,\n
577.405\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"INSTALLMENTS_PURCHASES\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 8006.815798601111,\n
\"min\": 0.0,\n        \"max\": 22500.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          8950.0,\n
411.0676446927374,\n          468.6375\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CASH_ADVANCE\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16276.409312979034,\n        \"min\": 0.0,\n        \"max\":
47137.21176,\n        \"num_unique_values\": 6,\n        \"samples\":
[\n          8950.0,\n          978.8711124654749,\n
47137.21176\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PURCHASES_FREQUENCY\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3164.1315573244246,\n        \"min\":
0.0,\n        \"max\": 8950.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          0.49035054837988823,\n          0.5,\n
8950.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ONEOFF_PURCHASES_FREQUENCY\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3164.2076992871343,\n
\"min\": 0.0,\n        \"max\": 8950.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          8950.0,\n
0.202457683575419,\n          0.3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\":
\"PURCHASES_INSTALLMENTS_FREQUENCY\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3164.1675778237154,\n
\"min\": 0.0,\n        \"max\": 8950.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          8950.0,\n
0.3644373415642458,\n          0.75\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CASH_ADVANCE_FREQUENCY\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3164.1989665645915,\n        \"min\": 0.0,\n        \"max\": 8950.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          8950.0,\n
0.13514420033519556,\n          1.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CASH_ADVANCE_TRX\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3157.6627679219728,\n        \"min\": 0.0,\n        \"max\": 8950.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          8950.0,\n
3.2488268156424582,\n          123.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PURCHASES_TRX\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3145.318735437323,\n        \"min\": 0.0,\n        \"max\": 8950.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
14.709832402234637,\n          7.0,\n          8950.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CREDIT_LIMIT\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9587.667437964345,\n        \"min\": 50.0,\n        \"max\": 30000.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
4494.449450364621,\n          3000.0,\n          8949.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PAYMENTS\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
17320.986188971423,\n        \"min\": 0.0,\n        \"max\":
50721.48336,\n        \"num_unique_values\": 8,\n        \"samples\":
[\n          1733.1438520248046,\n          856.901546,\n
8950.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MINIMUM_PAYMENTS\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 26501.924945218896,\n        \"min\":
0.019163,\n        \"max\": 76406.20752,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
864.2065423050828,\n          312.343947,\n          8637.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PRC_FULL_PAYMENT\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3164.222602967969,\n        \"min\":
0.0,\n        \"max\": 8950.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          8950.0,\n          0.15371464849162012,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"TENURE\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3160.928564179653,\n        \"min\": 1.3383307693673308,\n
\"max\": 8950.0,\n        \"num_unique_values\": 5,\n
```

```
\"samples\": [\n          11.51731843575419,\n          12.0,\n
1.3383307693673308\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

Visualized dataset for determining the null/missing values in the respective columns.

```
data.isnull().sum()

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
```

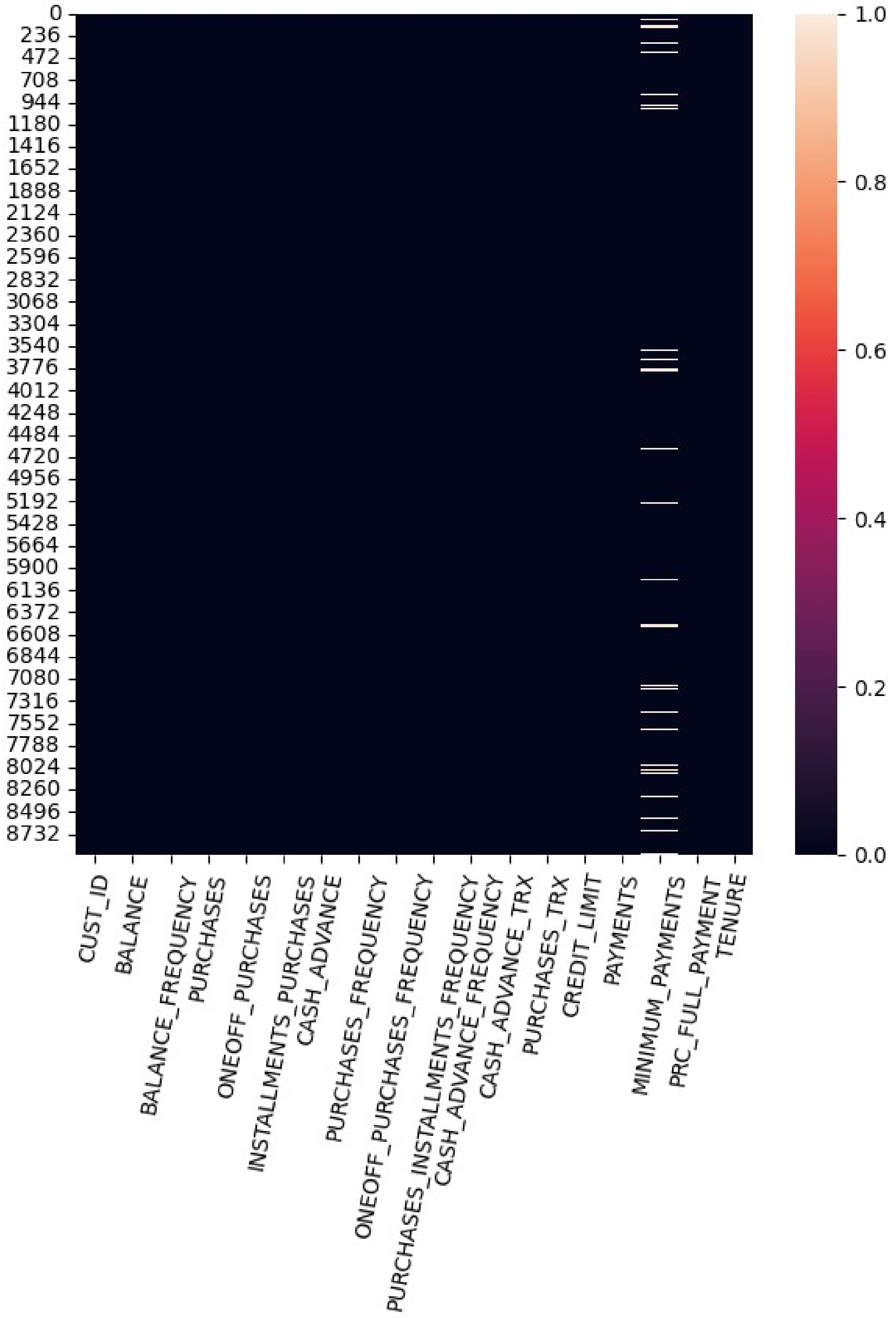
Checked the dataset for identifying any duplicate values or duplicate rows.

```
print(data.duplicated().sum())
data[data.duplicated()]

0

{"repr_error":"Out of range float values are not JSON compliant:
nan","type":"dataframe"}
```

STEP 3: PERFORMING EXPLORATORY DATA ANALYSIS (EDA)

Visualized dataset for representing the null values using a SNS Heatmap.

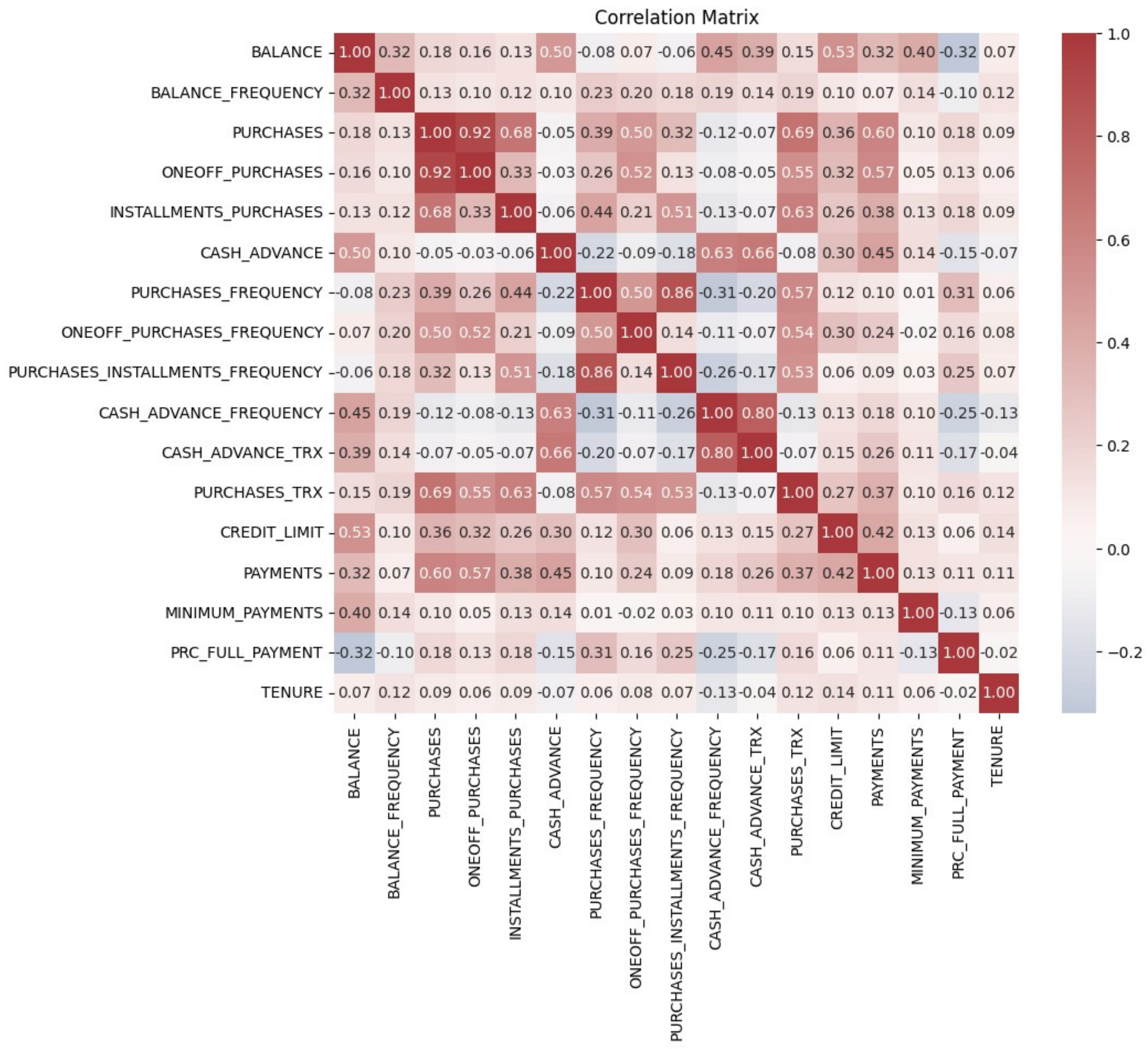
```
plt.figure(figsize=(7,7))
sns.heatmap(data.isnull())
plt.xticks(rotation=80)
plt.show()
```


0
236
472
708
944
1180
1416
1652
1888
2124
2360
2596
2832
3068
3304
3540
3776
4012
4248
4484
4720
4956
5192
5428
5664
5900
6136
6372
6608
6844
7080
7316
7552
7788
8024
8260
8496
8732
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
1.0
0.8
0.6
0.4
0.2
0.0

The heatmap above shows that we have very minimal missing values in the dataset 'Customer Data'.

Visualizing dataset further by utilizing Correlation Heatmap.

```
#Function is defined here

def correlation_heatmap(data, numeric_col1):
 corr = data[numeric_col1].corr()
 plt.figure(figsize=(10, 8))
 sns.heatmap(corr, annot=True, fmt='.2f', cmap='vlag', center=0)

 #Here cmap decides what colour and shapes we want in the Matrix.

 plt.title('Correlation Matrix')
 plt.show()

numeric_col1 = 
data.select_dtypes(include=[np.number]).columns.tolist()
if 'CUST_ID' in numeric_col1:
   numeric_col1.remove('CUST_ID')

correlation_heatmap(data, numeric_col1)
```



	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
BALANCE	1.00	0.32	0.18	0.16	0.13	0.50	-0.08	0.07	-0.06	0.45	0.39	0.15	0.53	0.32	0.40	-0.32	0.07
BALANCE_FREQUENCY	0.32	1.00	0.13	0.10	0.12	0.10	0.23	0.20	0.18	0.19	0.14	0.19	0.10	0.07	0.14	-0.10	0.12
PURCHASES	0.18	0.13	1.00	0.92	0.68	-0.05	0.39	0.50	0.32	-0.12	-0.07	0.69	0.36	0.60	0.10	0.18	0.09
ONEOFF_PURCHASES	0.16	0.10	0.92	1.00	0.33	-0.03	0.26	0.52	0.13	-0.08	-0.05	0.55	0.32	0.57	0.05	0.13	0.06
INSTALLMENTS_PURCHASES	0.13	0.12	0.68	0.33	1.00	-0.06	0.44	0.21	0.51	-0.13	-0.07	0.63	0.26	0.38	0.13	0.18	0.09
CASH_ADVANCE	0.50	0.10	-0.05	-0.03	-0.06	1.00	-0.22	-0.09	-0.18	0.63	0.66	-0.08	0.30	0.45	0.14	-0.15	-0.07
PURCHASES_FREQUENCY	-0.08	0.23	0.39	0.26	0.44	-0.22	1.00	0.50	0.86	-0.31	-0.20	0.57	0.12	0.10	0.01	0.31	0.06
ONEOFF_PURCHASES_FREQUENCY	0.07	0.20	0.50	0.52	0.21	-0.09	0.50	1.00	0.14	-0.11	-0.07	0.54	0.30	0.24	-0.02	0.16	0.08
PURCHASES_INSTALLMENTS_FREQUENCY	-0.06	0.18	0.32	0.13	0.51	-0.18	0.86	0.14	1.00	-0.26	-0.17	0.53	0.06	0.09	0.03	0.25	0.07
CASH_ADVANCE_FREQUENCY	0.45	0.19	-0.12	-0.08	-0.13	0.63	-0.31	-0.11	-0.26	1.00	0.80	-0.13	0.13	0.18	0.10	-0.25	-0.13
CASH_ADVANCE_TRX	0.39	0.14	-0.07	-0.05	-0.07	0.66	-0.20	-0.07	-0.17	0.80	1.00	-0.07	0.15	0.26	0.11	-0.17	-0.04
PURCHASES_TRX	0.15	0.19	0.69	0.55	0.63	-0.08	0.57	0.54	0.53	-0.13	-0.07	1.00	0.27	0.37	0.10	0.16	0.12
CREDIT_LIMIT	0.53	0.10	0.36	0.32	0.26	0.30	0.12	0.30	0.06	0.13	0.15	0.27	1.00	0.42	0.13	0.06	0.14
PAYMENTS	0.32	0.07	0.60	0.57	0.38	0.45	0.10	0.24	0.09	0.18	0.26	0.37	0.42	1.00	0.13	0.11	0.11
MINIMUM_PAYMENTS	0.40	0.14	0.10	0.05	0.13	0.14	0.01	-0.02	0.03	0.10	0.11	0.10	0.13	0.13	1.00	-0.13	0.06
PRC_FULL_PAYMENT	-0.32	-0.10	0.18	0.13	0.18	-0.15	0.31	0.16	0.25	-0.25	-0.17	0.16	0.06	0.11	-0.13	1.00	-0.02
TENURE	0.07	0.12	0.09	0.06	0.09	-0.07	0.06	0.08	0.07	-0.13	-0.04	0.12	0.14	0.11	0.06	-0.02	1.00

Customer Identity or CUST_ID column in our dataset is a unique ID and thats'why not very significant and for this reason, it's removed from column list before calling the correlation_heatmap function.

INFERENCE FROM CORRELATION MATRIX: PURCHASE AND ONEOFF_PURCHASES ARE CLOSELY RELATED OR POSITIVELY CORRELATED WHEREAS PRC_FULL_PAYMENT AND BALANCE ARE VERY POORELY RELATED OR NEGATIVELY COORELATED

STEP 4: DATA PRE-PROCESSING IS PERFORMED HERE TO REMOVE ANY INCONSISTANCIES IN THE RAW DATA

CUST_ID column is dropped as it's insignificant in the dataset but a copy of it has to be created so that we can trace the original customer after clustering.

```
cust_identity = data['CUST_ID']
data = data.drop(columns=['CUST_ID'])
data.head(5)
```

```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 8950,\n  \"fields\":
[\n    {\n      \"column\": \"BALANCE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2081.531879456554,\n
\"min\": 0.0,\n        \"max\": 19043.13856,\n
\"num_unique_values\": 8871,\n        \"samples\": [\n
325.024091,\n          965.514081,\n          203.499251\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BALANCE_FREQUENCY\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.23690400268475698,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 43,\n        \"samples\": [\n
0.428571,\n          0.8,\n          0.2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PURCHASES\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2136.6347818728423,\n        \"min\": 0.0,\n        \"max\":
49039.57,\n        \"num_unique_values\": 6203,\n        \"samples\":
[\n          1361.65,\n          2485.54,\n          2580.63\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ONEOFF_PURCHASES\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1659.887917437827,\n        \"min\":
0.0,\n        \"max\": 40761.25,\n        \"num_unique_values\":
4014,\n        \"samples\": [\n          25.62,\n          13007.07,\n
185.63\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"INSTALLMENTS_PURCHASES\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 904.3381151753365,\n
\"min\": 0.0,\n        \"max\": 22500.0,\n
\"num_unique_values\": 4452,\n        \"samples\": [\n
228.56,\n          255.58,\n          729.6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CASH_ADVANCE\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2097.1638766431465,\n        \"min\": 0.0,\n        \"max\":
47137.21176,\n        \"num_unique_values\": 4323,\n
\"samples\": [\n          4473.3497,\n          520.844673,\n
3968.684047\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PURCHASES_FREQUENCY\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.40137074736905376,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 47,\n
\"samples\": [\n          0.8,\n          0.555556,\n          0.2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"ONEOFF_PURCHASES_FREQUENCY\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
0.2983360651847212,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 47,\n        \"samples\": [\n
0.909091,\n          0.625,\n          0.181818\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\":
\"PURCHASES_INSTALLMENTS_FREQUENCY\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.39744777974542483,\n
\"min\": 0.0,\n        \"max\": 1.0,\n        \"num_unique_values\":
47,\n        \"samples\": [\n          0.857143,\n          0.222222,\
n          0.142857\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CASH_ADVANCE_FREQUENCY\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.20012138814749122,\n
\"min\": 0.0,\n        \"max\": 1.5,\n        \"num_unique_values\":
54,\n        \"samples\": [\n          0.222222,\n          0.818182,\
n          0.7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CASH_ADVANCE_TRX\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6,\n        \"min\": 0,\n
\"max\": 123,\n        \"num_unique_values\": 65,\n
\"samples\": [\n          47,\n          61,\n          0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"PURCHASES_TRX\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
24,\n        \"min\": 0,\n        \"max\": 358,\n
\"num_unique_values\": 173,\n        \"samples\": [\n          162,\n
216,\n          79\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CREDIT_LIMIT\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3638.8157254985426,\n        \"min\":
50.0,\n        \"max\": 30000.0,\n        \"num_unique_values\": 205,\
n        \"samples\": [\n          9000.0,\n          3000.0,\n
300.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PAYMENTS\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2895.063756904579,\n        \"min\":
0.0,\n        \"max\": 50721.48336,\n        \"num_unique_values\":
8711,\n        \"samples\": [\n          810.671862,\n
5943.975673,\n          7079.1781\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MINIMUM_PAYMENTS\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2372.446606583975,\n        \"min\": 0.019163,\n        \"max\":
76406.20752,\n        \"num_unique_values\": 8636,\n
\"samples\": [\n          173.484575,\n          185.120378,\n
163.711273\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PRC_FULL_PAYMENT\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.29249919623387977,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 47,\n
```

```
\"samples\": [\n          0.583333,\n          0.272727,\n
0.1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"TENURE\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 6,\n        \"max\": 12,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          12,\n
8,\n          7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data"}
```

Here the missing value of CREDIT_LIMIT is filled with Medium value of that column

```
data['CREDIT_LIMIT'].fillna(data['CREDIT_LIMIT'].median(),
inplace=True)
data.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                     313
PRC_FULL_PAYMENT                       0
TENURE                                 0
dtype: int64
```

Here the missing values of MINIMUM_PAYMENTS columns are filled by establishing a relation between Balance and minimum payment and then determing the median values. Then that ratio is utilised with Balance column data to fill the values in column.

```
ratio = (data['MINIMUM_PAYMENTS'] / data['BALANCE']).median()
data['MINIMUM_PAYMENTS'].fillna(data['BALANCE'] * ratio, inplace=True)

print(data.isnull().sum()[[ 'MINIMUM_PAYMENTS']])

MINIMUM_PAYMENTS    0
dtype: int64
```

STEP 5: PERFORMING FEATURE ENGINEERING

A copy of the dataset is made here for performing feature engineering.

'If statements' are explicitely written here to make the code reusable if different csv files dataset are used. Here the new features are made based on the scores that are obtained from the correlation matrix. Replace function converts any zero value to np.nan value in division so to avoid zero and zero by zero division case.

```
numeric_features = data.copy()


if {'PURCHASES', 'PURCHASES_TRX'}.issubset(numeric_features.columns):
    numeric_features['Average_purchase_amt'] =
numeric_features['PURCHASES'] /
numeric_features['PURCHASES_TRX'].replace(0, np.nan)

if {'CASH_ADVANCE',
'CASH_ADVANCE_TRX'}.issubset(numeric_features.columns):
    numeric_features['Cash_advance_per_trx'] =
numeric_features['CASH_ADVANCE'] /
numeric_features['CASH_ADVANCE_TRX'].replace(0, np.nan)

if {'PAYMENTS', 'CREDIT_LIMIT'}.issubset(numeric_features.columns):
    numeric_features['Payment_to_Crlimit'] =
numeric_features['PAYMENTS'] /
numeric_features['CREDIT_LIMIT'].replace(0, np.nan)

if {'BALANCE', 'CREDIT_LIMIT'}.issubset(numeric_features.columns):
    numeric_features['Utilization'] = numeric_features['BALANCE'] /
numeric_features['CREDIT_LIMIT'].replace(0, np.nan)

if {'ONEOFF_PURCHASES',
'PURCHASES'}.issubset(numeric_features.columns):
    numeric_features['Oneoff_ratio'] =
numeric_features['ONEOFF_PURCHASES'] /
numeric_features['PURCHASES'].replace(0, np.nan)

if {'INSTALLMENTS_PURCHASES',
'PURCHASES'}.issubset(numeric_features.columns):
    numeric_features['Installment_ratio'] =
numeric_features['INSTALLMENTS_PURCHASES'] /
numeric_features['PURCHASES'].replace(0, np.nan)


#Here we are again converting all the NAN values in the
numeric_features dataset into zero.

numeric_features.fillna(0, inplace=True)

print("After creating new columns, shape of the dataset is =",
numeric_features.shape)
```

```
After creating new columns, shape of the dataset is = (8950, 23)
```

```
numeric_features.head(5)
```

{"type":"dataframe","variable_name":"numeric_features"}

STEP 6: PERFORMING SCALING OF DATA AND K MEANS CLUSTERRING

Here 'feature_cols' is the complete list of column data and column names which are strings. 'select_dtypes' here only selects the columns which are numeric so that if the .csv file changes with similar columns, then also it can handle or process the columns without breaking the code.

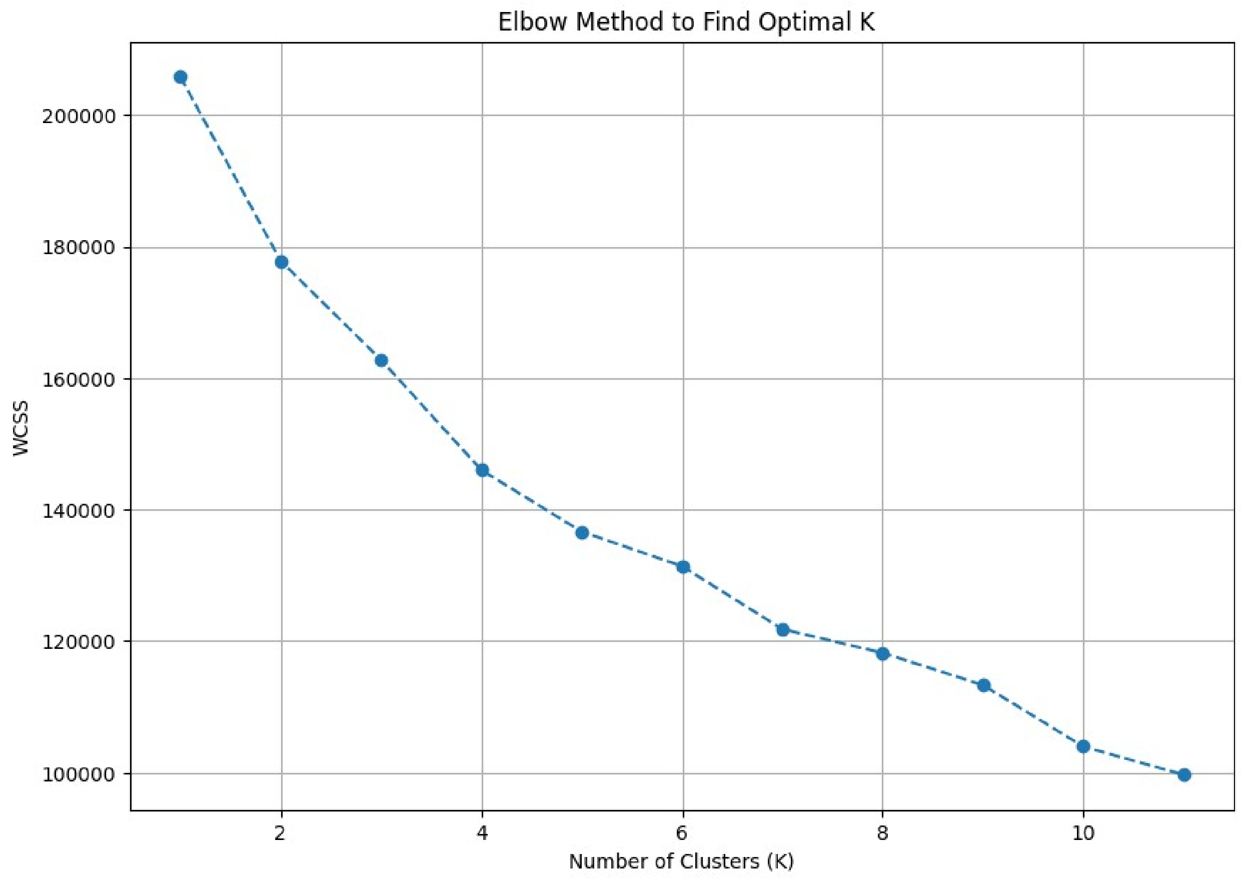
Further, here we have to pass only numeric features to scalar fit transform and that'why we need to extract data or numeric values without labels.

```
feature_cols =
numeric_features.select_dtypes(include=[np.number]).columns.tolist()

X = numeric_features[feature_cols]

scaler = StandardScaler()
scaled_features = pd.DataFrame(scaler.fit_transform(X), columns=
X.columns)

#Here scaled_features is a dataframe that contains all the column data
from standard scalar transformation.

print("The Scaled feature matrix shape is", scaled_features.shape)

The Scaled feature matrix shape is (8950, 23)
```

'wcss' means "Within-Cluster Sum of Squares". So here we will initially create an empty list. A custom function is then defined and called to get the results from KMeans algorithm.

```
def evaluate_kmeans(X, ks=range(1, 12)):
 wcss = []
 for i in ks:
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(scaled_features)
    wcss.append(kmeans.inertia_)
 return wcss

kmeans_results = evaluate_kmeans(scaled_features.values, ks=range(1,
12))


# Plotting the Elbow Method graph between interia or wcss and number
of cluster(k)

plt.figure(figsize=(10, 7))
plt.plot(range(1, 12), kmeans_results, marker='o', linestyle='--')
```

```python
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.title('\n Elbow Method to Find Optimal K')
plt.grid()
plt.show()
```

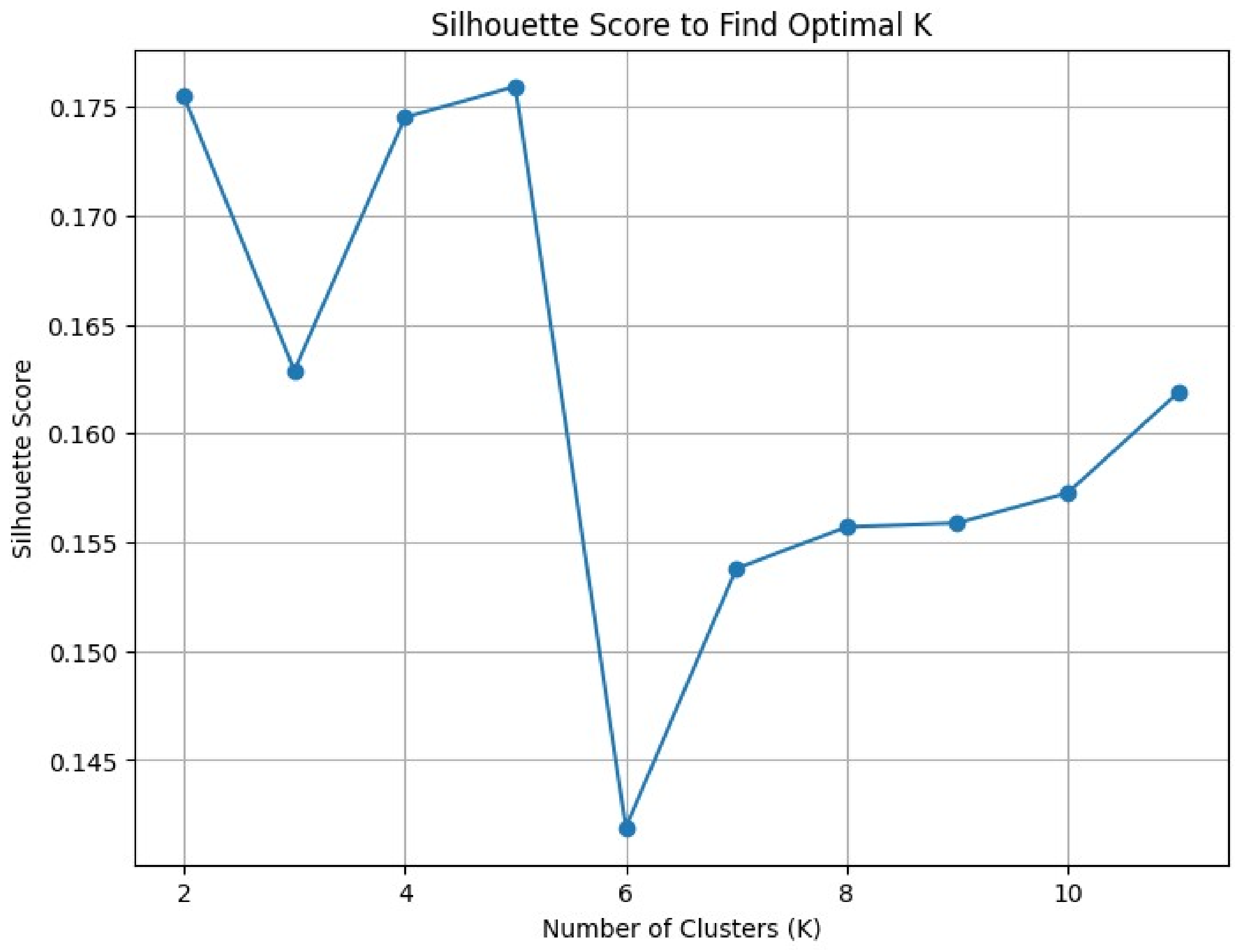


INFERENCE: The optimal number of clusters based on the Elbow Method is K= 6. This is where the 'WCSS' curve bends and begins to flatten. So we can assume K=6 for KMeans clustering and can then evaluate the quality further using silhouette scores.

Here we will perform the silhouette score analysis to find out the value of K for the best quality of cluters. An empty list is created here for storing silhouette score values. Silhouette needs alteast 2 clusters and so it started with 2.

```python
silhouette_scores = []
K_range = range(2, 12)

for K in K_range:
    kmeans = KMeans(n_clusters=K, random_state=42)
    kmeans.fit(scaled_features)
    silhouette = silhouette_score(scaled_features, kmeans.labels_)
    silhouette_scores.append(silhouette)
```

```
#Plotting the Graph for silhouette scores

plt.figure(figsize=(8, 6))
plt.plot(K_range, silhouette_scores, marker='o', linestyle='-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score to Find Optimal K')
plt.grid()
plt.show()
```



INFERENCE : FROM THE ABOVE GRAPH ANALYSIS OF SILHOUETTE SCORE, WE CAN SELECT EITHER FOUR OR FIVE CLUSTER WITH VALUE 0.18 AS THE REQUISITE NUMBER OF CLUSTERS IN OUR TASK AS FROM INITIAL INSIGHTS, CHOOSING EITHER FOUR OR FIVE NUMBER OF CLUSTER HAS THE BEST QUALITY OF SEPERATION.

STEP 7: PERFORMING AGGLOMERATIVE CLUSTERING EVALUATION

Here in Agglomerative clusturing, average linkage is choosen as it minimizes the average distance between all pairs of observations in different clusters.

```
aggm_results = []

for k in range(2, 11):
 aggm = AgglomerativeClustering(n_clusters=k, linkage='average')
 labels = aggm.fit_predict(scaled_features.values)
 sil1 = silhouette_score(scaled_features, labels)
 aggm_results.append({'k': k, 'silhouette': sil1, 'model': aggm})
 print(f'Agg k={k} : silhouette={sil1:.2f}')
```

```
Agg k=2 : silhouette=0.94
Agg k=3 : silhouette=0.86
Agg k=4 : silhouette=0.83
Agg k=5 : silhouette=0.81
Agg k=6 : silhouette=0.80
Agg k=7 : silhouette=0.75
Agg k=8 : silhouette=0.75
Agg k=9 : silhouette=0.74
Agg k=10 : silhouette=0.69
```

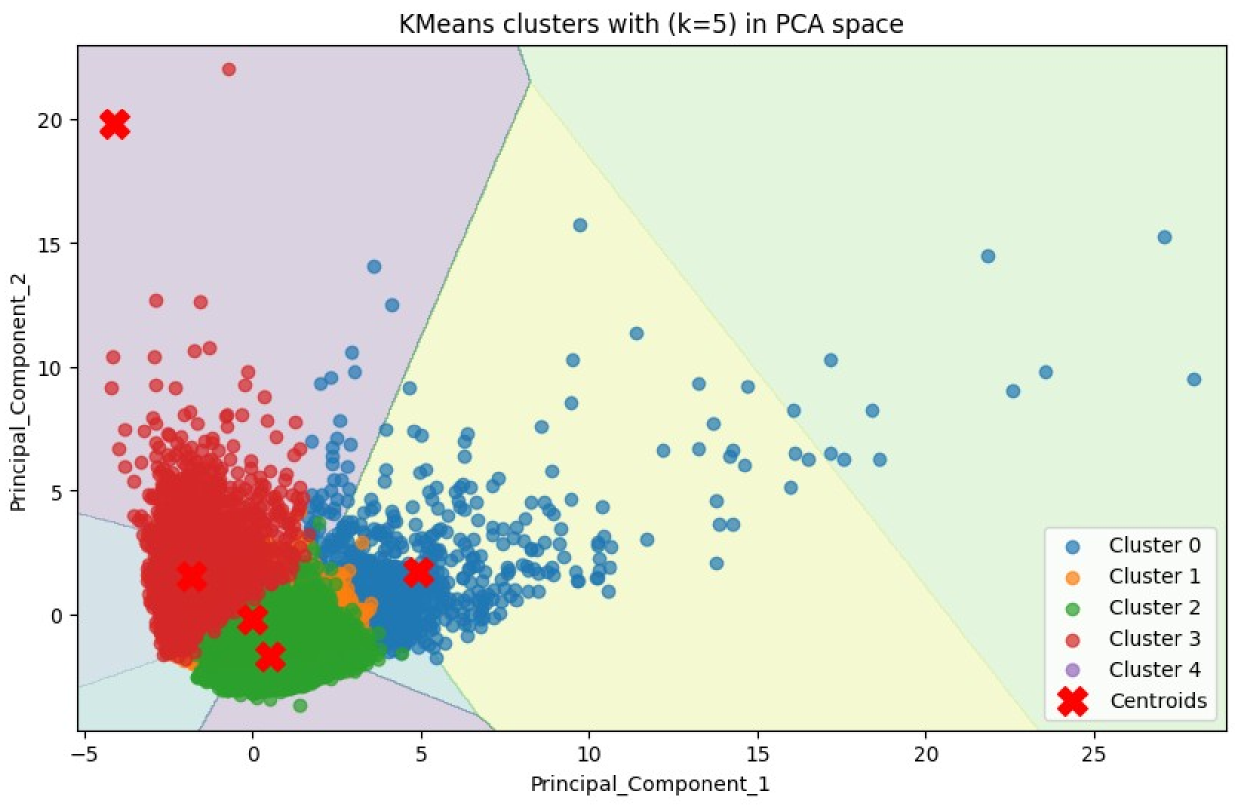
INFERENCE: BASED ON THE SILHOUETTE SCORES OBTAINED USING THE AGGLOMERATIVE CLUSTURING METHOD, THE BEST K IS 2 BUT IF WE COMPARE IT WITH OUR PREVIOUS RESULTS WITH INERTIA AND SILHOUETTE SCORE BASED ON KMEANS, THEN WE CAN CONCLUDE THAT THE OPTIMAL K VALUES ARE 4 AND 5.

STEP 8: PERFORMING PRINCIPAL COMPONENT ANALYSIS IN 2 DIMENSION

Here we have assumed that the best cluster for our dataset is 5 and then PCA is performed. PCA algorith works by determing the covariance matrix and then by calculating the eigen vectors of the covariance matrix. The eigen vectors with largest eigen values are our principal components. Finally a matrix is created to project the data on principal components.

```
best_k = 5
best_kmeans = KMeans(n_clusters=best_k, init='k-means++',
random_state=42)
kmeans_labels = best_kmeans.fit_predict(scaled_features.values)

#Here kmeans_labels is a cluster label array.

pca = PCA(n_components=2)
pca1 = pca.fit_transform(scaled_features.values)


#Projecting the clusters and their centres into PCA 2D space

h = 0.05          # This is step size of the grid created
x_min, x_max = pca1[:, 0].min() - 1, pca1[:, 0].max() + 1
y_min, y_max = pca1[:, 1].min() - 1, pca1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
```

```
# Predicted cluster label for each point in the mesh

Z = kmeans.predict(pca.inverse_transform(np.c_[xx.ravel(),
yy.ravel()]))
Z = Z.reshape(xx.shape)

# Plotted decision boundaries of clusters

plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.2, cmap='viridis')

# Plotted actual points

for cl in np.unique(kmeans_labels):
    ix = kmeans_labels == cl
    plt.scatter(pca1[ix, 0], pca1[ix, 1], label=f'Cluster {cl}',
alpha=0.7)


# Plotted cluster centers in PCA space. This visualisation is only an
approximation.

centers = pca.transform(best_kmeans.cluster_centers_)
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X',
label='Centroids')

plt.legend()
plt.title(f'KMeans clusters with (k={best_k}) in PCA space')
plt.xlabel('Principal_Component_1')
plt.ylabel('Principal_Component_2')
plt.show()

print("pca1 shape:", pca1.shape)
print("kmeans_labels shape:", kmeans_labels.shape)
print("Unique labels:", np.unique(kmeans_labels))
```



```
pca1 shape: (8950, 2)
kmeans_labels shape: (8950,)
Unique labels: [0 1 2 3 4]
```

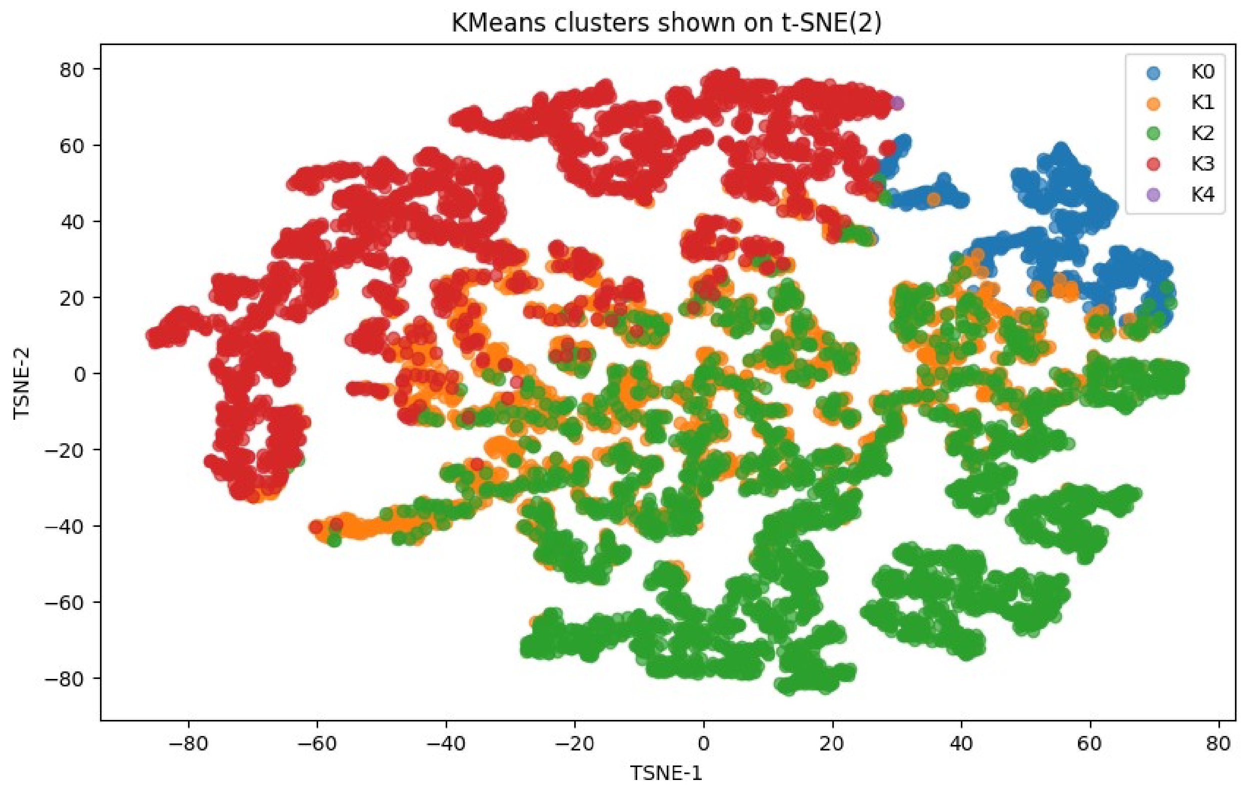
INFERENCE: PCA ANALYSIS AND CLUSTER PROJECTION ON PCA GRAPH SHOWS THAT THE DATAPOINTS OF FOUR CLUSTERS OVERLAPP OVER ONE ANOTHER AS VISIBLE FROM THEIR DATA CENTROIDS BUT THE FIFTH CLUSTER IS AN OUTLIER WITH IT'S CENTROD BEING FAR AWAY. THIS ALSO SUGGESTS THAT THEIR IS LESS ASSOCIATION OF FEATURES IN THE FIFTH CLUSTER WITH THE REST OF THE FOUR CLUSTERS PRESENT AND THE DATASET CAN BE WELL-REPRESENTED BY TAKING FOUR CLUSTURS ONLY.

STEP 9: PERFORMING t-SNE(t-distributed Stochastic Neighbor Embedding) FOR DIMENSIONALITY REDUCTION

Here we are visualizing the different clusters obtained from t-SNE method in 2 Dimension.

```
X_for_tsne = pca1
RANDOM_STATE = 42
tsne = TSNE(n_components=2, perplexity=40, n_iter=1200, random_state=
RANDOM_STATE, init='pca')
X_tsne = tsne.fit_transform(X_for_tsne)

#Here we are plotting the t-SNE Graph

plt.figure(figsize=(10, 6))
```

```
for cl in np.unique(kmeans_labels):
 ix = kmeans_labels == cl
 plt.scatter(X_tsne[ix, 0], X_tsne[ix, 1], label=f'K{cl}', alpha=0.7)
plt.legend()
plt.title('KMeans clusters shown on t-SNE(2)')
plt.xlabel('TSNE-1')
plt.ylabel('TSNE-2')
plt.show()
```



INFERENCE: FROM THE ABOVE GRAPH OBTAINED FROM t-SNE METHOD, WE CAN CONCLUDE THAT THE DATASET 'CUSTOMER DATA' CAN BE BROADLY CLUSTURRED INTO 4 PARTS AS REPRESENTED BY DIFFERENT COLOURS AND THE 5TH CLUSTER(ALTHOUGH PRESENT) IS OVERLAPPED WITH OTHER CLUSTER DATA POSSIBLY DUE TO SIMILAR FEATURES.

STEP 10: LABEL ASSIGNMENT ON ORIGINAL DATAFRAME AND VISUALISATION DATA FOR BUSINESS USE-CASES

Here we are basically informing the KMeans model that we want create 4 clusters, determine the cluster centers and to assign the datapoints to nearest cluster. Here the orginal dataset numeric_features is attached with kmeans_labels that we have obtained by creating a "New Column" in it. Further, the customer identity column has been concatanated with the numeric features with label colums so that the original customer can be traced.

Best number of clusters is finally choosen to be 4 based on t-SNE, silhouette score and PCA Graph analysis.

```
best_k = 4

best_kmeans = KMeans(n_clusters=best_k, init='k-means++',
random_state=42)
kmeans_labels = best_kmeans.fit_predict(scaled_features.values)

numeric_features['cluster_label'] = kmeans_labels
Final_dataset = pd.concat([cust_identity, numeric_features], axis=1)
print(Final_dataset['cluster_label'].value_counts())

cluster_label
2    3043
3    2748
1    2436
0     723
Name: count, dtype: int64
```

INFERENCE:

1. CLUSTER 1 HAS 723 ROWS
2. CLUSTER 2 HAS 2436 ROWS
3. CLUSTER 3 HAS 3043 ROWS
4. CLUSTER 4 HAS 2748 ROWS

As the original dataset is clusturred into 4 parts based on labels, we can now analyse the data based on the simliar features that are useful for obtaining business trends. Here we are extracting the mean data of columns for each of the clusters by using groupby with cluster label.

```
cluster_profile =
Final_dataset.groupby('cluster_label').mean(numeric_only=True)
#numeric only here is True as we have only 1 column with strings
print(cluster_profile)

                   BALANCE  BALANCE_FREQUENCY    PURCHASES
ONEOFF_PURCHASES  \
cluster_label

0              3101.194769           0.985557  5732.974689
3735.366487
1               976.389456           0.837846   918.512849
816.712184
2               652.267868           0.848544   722.739422
119.516122
3              2691.612101           0.915540   144.450218
90.409774

               INSTALLMENTS_PURCHASES  CASH_ADVANCE
PURCHASES_FREQUENCY  \
cluster_label
```

```
0                         1998.438077   1115.891258
0.953404
1                          101.976314    243.793907
0.454252
2                          603.728419    182.533660
0.754780
3                           54.084720   2476.264626
0.107705

               ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY  \
cluster_label

0                                0.717612
0.801896
1                                0.382034
0.143960
2                                0.075589
0.707089
3                                0.048220
0.065351

               CASH_ADVANCE_FREQUENCY  ...     PAYMENTS
MINIMUM_PAYMENTS  \
cluster_label                          ...

0                            0.109770  ...  5967.568057
1621.563184
1                            0.057889  ...  1220.527194
454.970240
2                            0.034802  ...   957.862841
653.757233
3                            0.321418  ...  1931.988683
1191.468827

               PRC_FULL_PAYMENT     TENURE  Average_purchase_amt  \
cluster_label
0                      0.272576  11.910097            120.262671
1                      0.124272  11.584975            142.645623
2                      0.257568  11.542885             52.723500
3                      0.033540  11.325691             24.115849

               Cash_advance_per_trx  Payment_to_Crlimit
Utilization  \
cluster_label

0                        153.750224            0.807761     0.360726

1                         88.959726            0.428229     0.319505
```

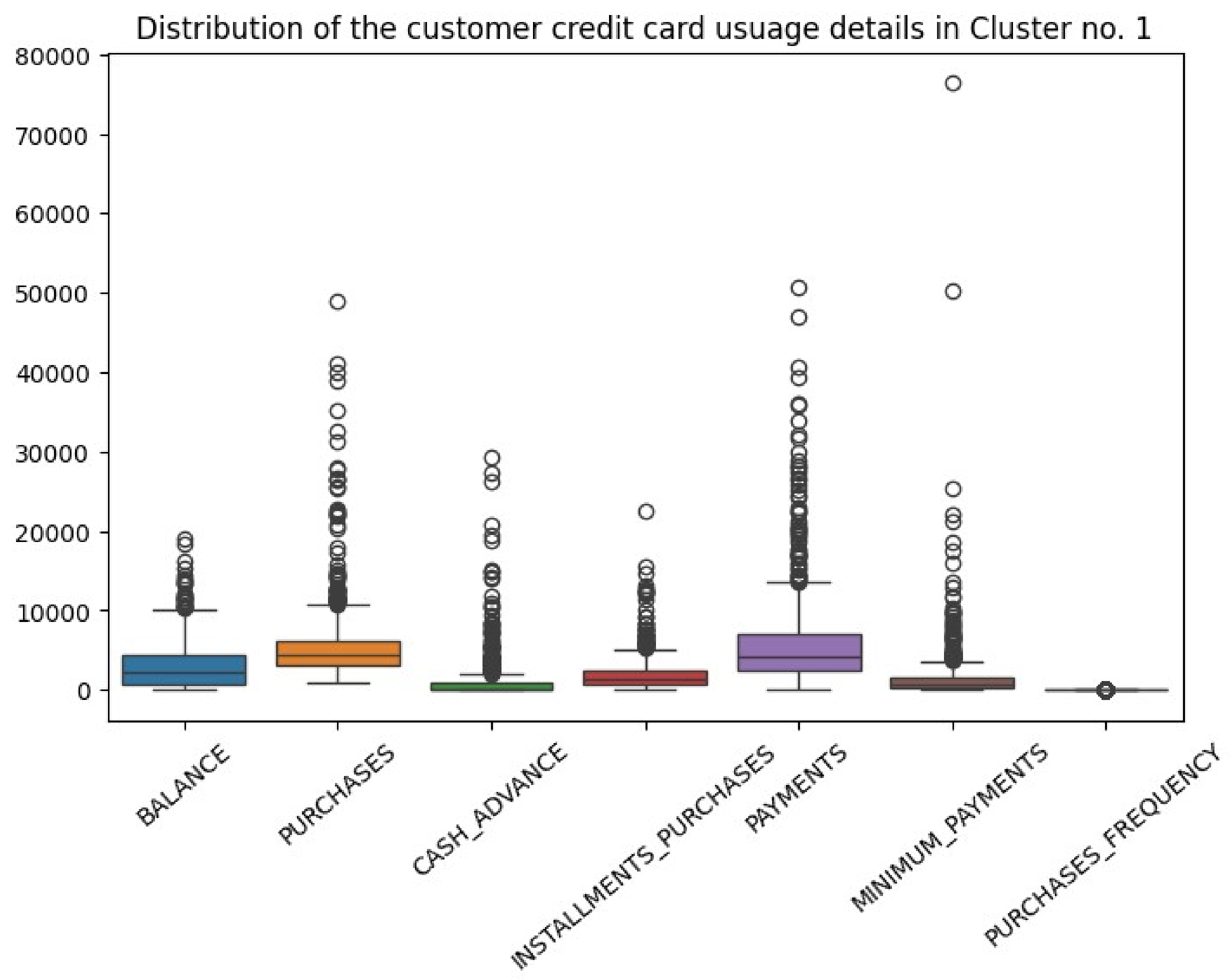
```
2                        63.868522           0.379618     0.257040

3                       490.349919           0.543540     0.603791


              Oneoff_ratio  Installment_ratio
cluster_label
0                 0.613810           0.386337
1                 0.898669           0.084520
2                 0.097969           0.903525
3                 0.170091           0.102008

[4 rows x 23 columns]
```

BUSINESS INSIGHTS BASED ON CUSTOMER CLUSTERS:

1. **CLUSTER 1 HAS PEOPLE WHO NOT ONLY MAINTAIN HIGH BALANCE BUT ARE ALSO HEAVY SPENDERS. THEY POSSES LOW RISK AS THEIR PAYMENT TO CREDIT RATIO IS GOOD.**
2. **CLUSTER 2 HAS PEOPLE WHO POSSES HIGH RISK AS THE CASH ADVANCE PER TRANSACTION IS HIGH AND PURCHASE FULL PAYMENT IS LOW AS COMPARED TO BANK BALANCE MAINTAINED.**
3. **CLUSTER 3 HAS PEOPLE WHO MAINTAINS SIGNIFICANTLY LOWER BALANCE AND ALSO AVERAGE PURCHASE AMOUNT, UTILISATION ARE QUITE LOW, CREDIT LIMIT IS ALSO NOT UTILISED FULLY.**
4. **CLUSTER 4 HAS PEOPLE WHO MAINTAINS A GOOD BALANCE, PAYMENT TO CREDIT RATIO, GOOD CREDIT USERS FROM BANK BUT AVERAGE PURCHASES ARE LOW.**

Here we can change the cluster number and column name manually to view the distribution of parameters.

```
cluster_num = 0
Needed_ft = ['BALANCE', 'PURCHASES', 'CASH_ADVANCE',
'INSTALLMENTS_PURCHASES','PAYMENTS','MINIMUM_PAYMENTS',
'PURCHASES_FREQUENCY']
j = Final_dataset[numeric_features['cluster_label'] == cluster_num]


#Plotted a Graph here using sns boxplot for cluster number 1 for
showcasing only the selected features.

plt.figure(figsize=(8, 5))
sns.boxplot(data=j[Needed_ft])
plt.title(f'Distribution of the customer credit card usuage details in
Cluster no. {cluster_num+1}')
plt.xticks(rotation =40)
plt.show()
```



GRAPH EXPLANATION: From the above boxplot data of cluster 1, we can infer that this cluster or group of people maintains a decent bank balance, spends heavily, payments and installment payments are also and cash advances are well in the margins. Thus this particular segment of people can be considered as low risk and loyal customers.

Finally we can save the file with the name "Final_dataset_clustured" with the exact customer information and to which characteristic cluster they belong to for further business reserach.

```
Final_dataset_clustured = Final_dataset[['CUST_ID', 'cluster_label']]
Final_dataset_clustured.to_csv('Final_dataset_clustured.csv')
print("Cluster profile saved as 'cluster_profile.csv'")

Cluster profile saved as 'cluster_profile.csv'
```

This code finally saves the clusturred data into four sequential csv files so that the segmented datasets now can be utilised further for taking business driven decisons based on similar credit card usage patterns.

```
for k, cluster_data in Final_dataset.groupby('cluster_label'):
    filename = f'cluster_{k}.csv'
    cluster_data.to_csv(filename, index=False)
    print(f"Saved cluster {k} with {len(cluster_data)} rows to
'{filename}'")
```

```
Saved cluster 0 with 723 rows to 'cluster_0.csv'
Saved cluster 1 with 2436 rows to 'cluster_1.csv'
Saved cluster 2 with 3043 rows to 'cluster_2.csv'
Saved cluster 3 with 2748 rows to 'cluster_3.csv'
```

CAPSTONE PROJECT CONCLUSION: IN THIS PROJECT WORK, WE HAVE INITIALLY LOADED OUR NECESSARY LIBRARIES, VISUALISED OUR GIVEN DATASET FOR ANY INCONSISTENCIES, TREATED ACCORINGLY AND CREATED NEW FEATURES FOR KNOWING INSIGHTS AS PER CREDIT CARD USUAGE SEGEMNETS.

THEN WE HAVE APPLIED APPROPRIATE UNSUPERVISED LEARNING MACHINE LEARNING ALGORITHMS AFTER PERFORMING NECESSARY SCALING OF DATA TO DETERMINE THE OPTIMAL NUMBER OF CLUSTER WHICH HAVE SIMILAR FEATURES. FINALLY WE HAVE PERFORMED CLUSTURING OF OUR GIVEN DATASET 'Customer Data' FOR EXTRACTING INFORMATION AND REPRESENTING INSIGHTS WHICH ARE IMPORTANT AND BENEFICIAL FOR SEGMENTING SIMILAR CUSTOMERS BASED ON THEIR CREDIT CARD USUAGE.

Github link: https://github.com/AnuragBhattacharjee-Mainz/Credit-Card-Usage-Segmentation.git