customer-segmentation-final

October 15, 2024

Market Segmentation in SBI life Insurance

1 1. Overview

1.0.1 Objective :

This case requires to develop a customer segmentation to give recommendations like saving plans, loans, wealth management, etc. on target customer groups. ### **Data Description :** The sample Dataset summarizes the usage behavior of about 9000 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioral variables. ### **Data :**
Use the below link to download the Data Set:here

1.0.2 Attribute Information :

Following is the Data Dictionary for customer's credit card dataset :-

CUSTID : Identification of Credit Card holder (Categorical) BALANCE : Balance amount left in their account to make purchases BALANCEFREQUENCY : How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated) PURCHASES : Amount of purchases made from account ONEOFFPURCHASES : Maximum purchase amount done in one-go INSTALLMENTSPURCHASES : Amount of purchase done in installment CASHADVANCE : Cash in advance given by the user PURCHASESFREQUENCY : How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased) ONEOFFPURCHASESFREQUENCY : How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased) PURCHASESINSTALLMENTSFREQUENCY : How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done) CASHADVANCEFREQUENCY : How frequently the cash in advance being paid CASHADVANCETRX : Number of Transactions made with "Cash in Advanced" PURCHASESTRX : Numbe of purchase transactions made CREDITLIMIT : Limit of Credit Card for user PAYMENTS : Amount of Payment done by user MINIMUM_PAYMENTS : Minimum amount of payments made by user PRCFULLPAYMENT : Percent of full payment paid by user TENURE : Tenure of credit card service for user

2 2. Import Libraries:

[1]:
```
# import necessary libraries
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import␣
 ↪KMeans,AgglomerativeClustering,DBSCAN,SpectralClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_samples, silhouette_score
```

3 3. Load Dataset:

[2]:
```
# import the dataset
creditcard_df = pd.read_csv("credit_card_dataset.csv")
creditcard_df.head()
```

[2]:

	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
```

4 4.Exploratory Data Analysis & Data Cleaning:

[3]:
```
creditcard_df.shape
```

[3]:
```
(8950, 18)
```

[4]:
```
# information about the data
creditcard_df.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
```

[5]:
```
# Check the statistics summary of the dataframe
creditcard_df.describe()
```

[5]:
```
             BALANCE  BALANCE_FREQUENCY     PURCHASES  ONEOFF_PURCHASES  \
count    8950.000000        8950.000000   8950.000000       8950.000000
mean     1564.474828           0.877271   1003.204834        592.437371
std      2081.531879           0.236904   2136.634782       1659.887917
```

min	0.000000	0.000000	0.000000	0.000000
25%	128.281915	0.888889	39.635000	0.000000
50%	873.385231	1.000000	361.280000	38.000000
75%	2054.140036	1.000000	1110.130000	577.405000
max	19043.138560	1.000000	49039.570000	40761.250000

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
count	8950.000000	8950.000000	8950.000000	
mean	411.067645	978.871112	0.490351	
std	904.338115	2097.163877	0.401371	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.083333	
50%	89.000000	0.000000	0.500000	
75%	468.637500	1113.821139	0.916667	
max	22500.000000	47137.211760	1.000000	

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	\
count	8950.000000	8950.000000	
mean	0.202458	0.364437	
std	0.298336	0.397448	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	0.083333	0.166667	
75%	0.300000	0.750000	
max	1.000000	1.000000	

	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	\
count	8950.000000	8950.000000	8950.000000	8949.000000	
mean	0.135144	3.248827	14.709832	4494.449450	
std	0.200121	6.824647	24.857649	3638.815725	
min	0.000000	0.000000	0.000000	50.000000	
25%	0.000000	0.000000	1.000000	1600.000000	
50%	0.000000	0.000000	7.000000	3000.000000	
75%	0.222222	4.000000	17.000000	6500.000000	
max	1.500000	123.000000	358.000000	30000.000000	

	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
count	8950.000000	8637.000000	8950.000000	8950.000000
mean	1733.143852	864.206542	0.153715	11.517318
std	2895.063757	2372.446607	0.292499	1.338331
min	0.000000	0.019163	0.000000	6.000000
25%	383.276166	169.123707	0.000000	12.000000
50%	856.901546	312.343947	0.000000	12.000000
75%	1901.134317	825.485459	0.142857	12.000000
max	50721.483360	76406.207520	1.000000	12.000000

[6]:
```
# checking for Null values in data frame
creditcard_df.isnull().sum()
```

[6]:
```
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
```

[7]:
```
# find all columns having missing values
missing_var = [var for var in creditcard_df.columns if creditcard_df[var].
 ↪isnull().sum()>0]
missing_var
```

[7]:
```
['CREDIT_LIMIT', 'MINIMUM_PAYMENTS']
```

[8]:
```
# fill mean value in place of missing values
creditcard_df["MINIMUM_PAYMENTS"] = creditcard_df["MINIMUM_PAYMENTS"].
 ↪fillna(creditcard_df["MINIMUM_PAYMENTS"].mean())
creditcard_df["CREDIT_LIMIT"] = creditcard_df["CREDIT_LIMIT"].
 ↪fillna(creditcard_df["CREDIT_LIMIT"].mean())
```

[9]:
```
# Again check for null values
creditcard_df.isnull().sum()
```

[9]:
```
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
```

[10]:
```
# check duplicate entries in the dataset
creditcard_df.duplicated().sum()
```

[10]:
```
0
```

[11]:
```
# drop unnecessary columns
creditcard_df.drop(columns=["CUST_ID"],axis=1,inplace=True)
```

[12]:
```
creditcard_df.columns
```

[12]:
```
Index(['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')
```

[13]:
```
creditcard_df.head()
```

[13]:
```
        BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  \
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

   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        864.206542          0.000000      12
4   678.334763        244.791237          0.000000      12
```

5 5. Outlier Detection

[14]:
```
# find outlier in all columns
for i in creditcard_df.select_dtypes(include=['float64','int64']).columns:
  max_thresold = creditcard_df[i].quantile(0.95)
  min_thresold = creditcard_df[i].quantile(0.05)
  creditcard_df_no_outlier = creditcard_df[(creditcard_df[i] < max_thresold) &␣
 ↪(creditcard_df[i] > min_thresold)].shape
  print(" outlier in ",i,"is" ,int(((creditcard_df.
 ↪shape[0]-creditcard_df_no_outlier[0])/creditcard_df.shape[0])*100),"%")
```

```
outlier in  BALANCE is 10 %
outlier in  BALANCE_FREQUENCY is 75 %
outlier in  PURCHASES is 27 %
outlier in  ONEOFF_PURCHASES is 53 %
outlier in  INSTALLMENTS_PURCHASES is 48 %
outlier in  CASH_ADVANCE is 56 %
outlier in  PURCHASES_FREQUENCY is 47 %
outlier in  ONEOFF_PURCHASES_FREQUENCY is 53 %
outlier in  PURCHASES_INSTALLMENTS_FREQUENCY is 58 %
outlier in  CASH_ADVANCE_FREQUENCY is 57 %
outlier in  CASH_ADVANCE_TRX is 56 %
outlier in  PURCHASES_TRX is 27 %
outlier in  CREDIT_LIMIT is 14 %
outlier in  PAYMENTS is 10 %
outlier in  MINIMUM_PAYMENTS is 10 %
```

```
outlier in  PRC_FULL_PAYMENT is 71 %
outlier in  TENURE is 91 %
```

[15]:
```
# remove outliers from columns having nearly 10% outlier
max_thresold_BALANCE = creditcard_df["BALANCE"].quantile(0.95)
min_thresold_BALANCE = creditcard_df["BALANCE"].quantile(0.05)
max_thresold_CREDIT_LIMIT = creditcard_df["CREDIT_LIMIT"].quantile(0.95)
min_thresold_CREDIT_LIMIT = creditcard_df["CREDIT_LIMIT"].quantile(0.05)
max_thresold_PAYMENTS = creditcard_df["PAYMENTS"].quantile(0.95)
min_thresold_PAYMENTS = creditcard_df["PAYMENTS"].quantile(0.05)
creditcard_df_no_outlier = creditcard_df[(creditcard_df["CREDIT_LIMIT"] <␣
 ↪max_thresold_CREDIT_LIMIT) & (creditcard_df["CREDIT_LIMIT"] >␣
 ↪min_thresold_CREDIT_LIMIT) & (creditcard_df["BALANCE"] <␣
 ↪max_thresold_BALANCE) & (creditcard_df["BALANCE"] > min_thresold_BALANCE) & ␣
 ↪(creditcard_df["PAYMENTS"] < max_thresold_PAYMENTS) &␣
 ↪(creditcard_df["PAYMENTS"] > min_thresold_PAYMENTS)]
```

[16]:
```
# DataFrame having no outlier
creditcard_df_no_outlier.head()
```

[16]:

	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
1	3202.467416	0.909091	0.00	0.00	0.00	6442.945483	0.000000	0.000000	0.000000	0.25	4	0	7000.0
2	2495.148862	1.000000	773.17	773.17	0.00	0.000000	1.000000	1.000000	0.000000	0.00	0	12	7500.0
4	817.714335	1.000000	16.00	16.00	0.00	0.000000	0.083333	0.083333	0.000000	0.00	0	1	1200.0
5	1809.828751	1.000000	1333.28	0.00	1333.28	0.000000	0.666667	0.000000	0.583333	0.00	0	8	1800.0
7	1823.652743	1.000000	436.20	0.00	436.20	0.000000	1.000000	0.000000	1.000000	0.00	0	12	2300.0

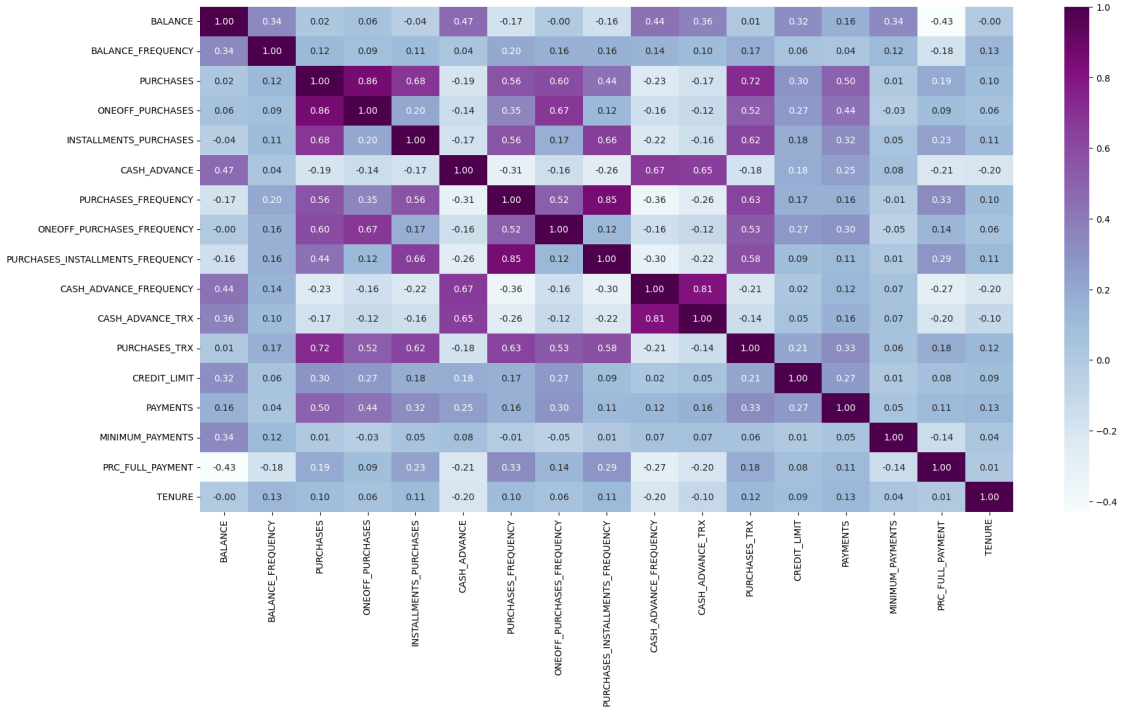
	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
1	4103.032597	1072.340217	0.222222	12
2	622.066742	627.284787	0.000000	12
4	678.334763	244.791237	0.000000	12
5	1400.057770	2407.246035	0.000000	12
7	679.065082	532.033990	0.000000	12

[17]:
```
creditcard_df_no_outlier.shape
```

[17]: (6466, 17)

[18]:
```
# correlation matrix of DataFrame
plt.figure(figsize=(20,10))
corn=creditcard_df_no_outlier.corr()
sns.heatmap(corn,annot=True,cmap="BuPu",fmt='.2f')
```

[18]: <Axes: >



5.1 From the results, we can see 3 pairs of strong correlation

1. "PURCHASES" and "ONEOFF_PURCHASES" – 0.86
2. "PURCHASES_FREQUENCY" and 'PURCHASES_INSTALLMENT_FREQUENCY' – 0.85

3. “CASH_ADVANCE_TRX” and “CASH_ADVANCE_FREQUENCY” –0.81

6 6. Scaling the data

The next step is to scale our values to give them all equal importance. Scaling is also important from a clustering perspective as the distance between points affects the way clusters are formed.

Using the StandardScaler, we transform our dataframe into the following numpy arrays

[19]:
```
# scale the DataFrame
scalar=StandardScaler()
creditcard_scaled_df = scalar.fit_transform(creditcard_df_no_outlier)
```

[20]:
```
creditcard_scaled_df
```

[20]:
```
array([[ 1.35958568, -0.02715353, -0.71136663, …,  0.18339488,
         0.24802861,  0.33969475],
       [ 0.84268315,  0.48108734, -0.05912009, …, -0.04878463,
        -0.51957586,  0.33969475],
       [-0.38317207,  0.48108734, -0.69786902, …, -0.24832644,
        -0.51957586,  0.33969475],
       …,
       [-0.94653953, -0.45069038, -0.51244565, …, -0.32934159,
         1.783241   , -4.58327778],
       [-0.96594456, -0.45069038, -0.61814878, …, -0.34163245,
         1.20753592, -4.58327778],
       [-0.92315108, -2.31424023, -0.71136663, …, -0.36464695,
         0.63183085, -4.58327778]])
```

7 7. Dimensionality reduction

-> Dimensionality reduction is a technique used to reduce the number of features in a dataset while retaining as much of the important information as possible.

-> In other words, it is a process of transforming high-dimensional data into a lower-dimensional space that still preserves the essence of the original data.

-> This can be done for a variety of reasons, such as to reduce the complexity of a model, to reduce the storage space, to improve the performance of a learning algorithm, or to make it easier to visualize the data.

-> There are several techniques for dimensionality reduction, * including principal component analysis (PCA), * singular value decomposition (SVD), * and linear discriminant analysis (LDA).

Each technique uses a different method to project the data onto a lower-dimensional space while preserving important information.

[21]:
```
# convert the DataFrame into 2D DataFrame for visualization
pca = PCA(n_components=2)
```

```
principal_comp = pca.fit_transform(creditcard_scaled_df)
pca_df = pd.DataFrame(data=principal_comp,columns=["pca1","pca2"])
pca_df.head()
```

[21]:

	pca1	pca2
0	-2.286555	3.003828
1	1.134715	0.431969
2	-1.458103	-1.493204
3	0.740689	-0.539416
4	0.648374	-1.077139

8 8. Hyperparameter tuning

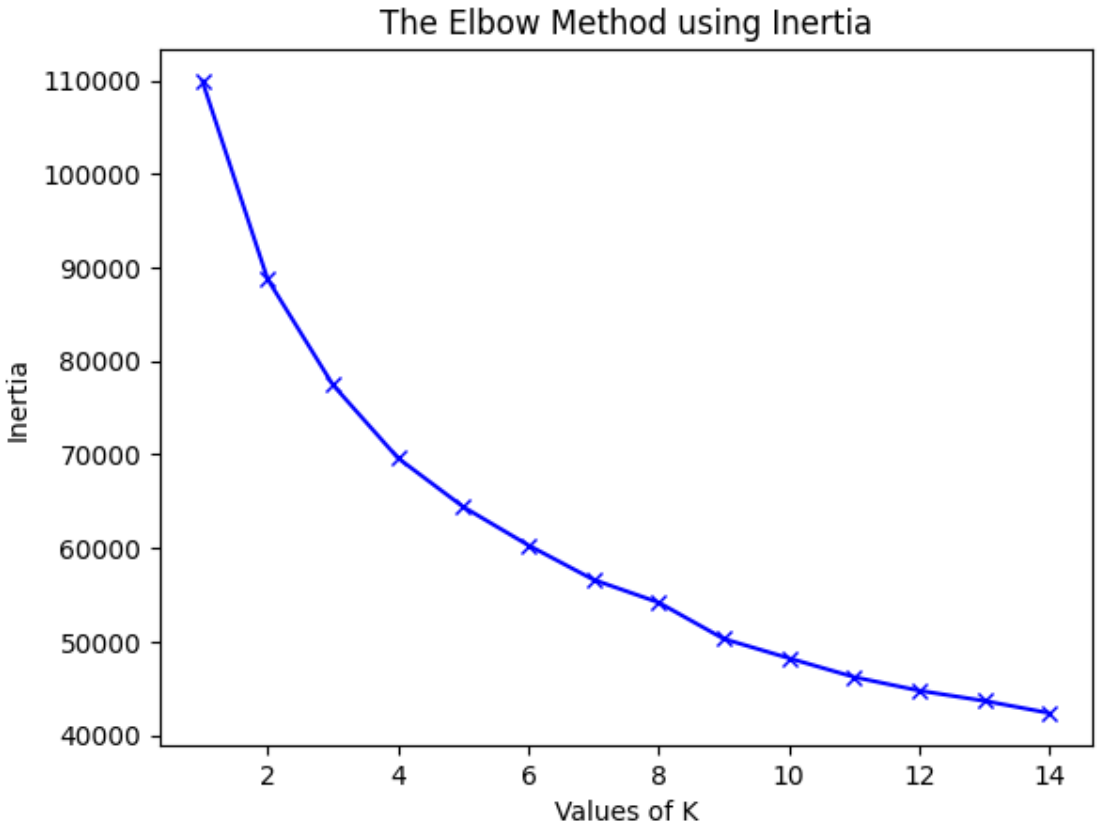
[22]:

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

```
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
```

```
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
```

```
explicitly to suppress the warning
  warnings.warn(
```



From this plot, 4th cluster seems to be the elbow of the curve. However, the values does not reduce to linearly until 8th cluster, so we may consider using 8 clusters in this case.

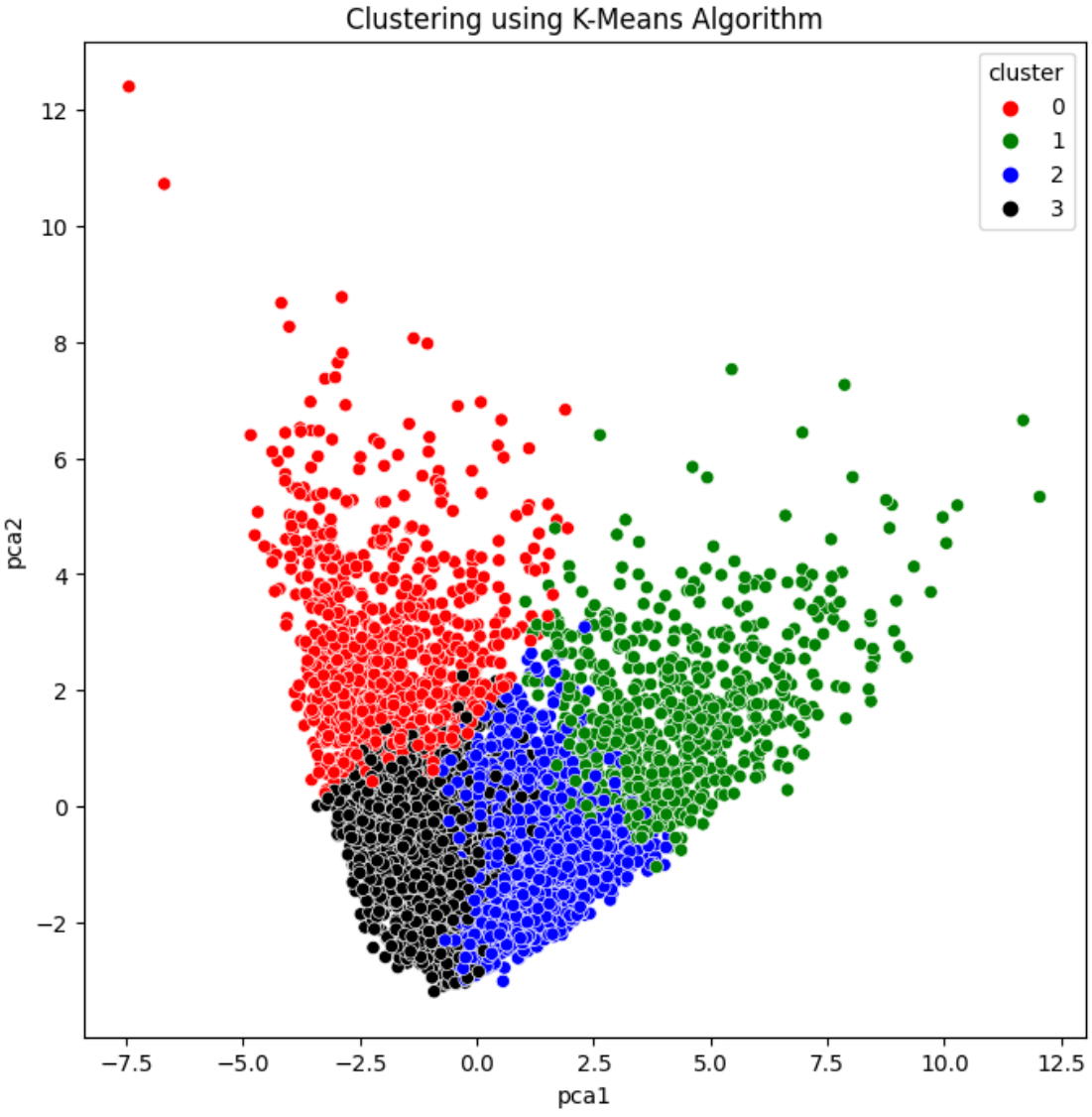
9 9. Model Building

9.1 ** K-Means Clustering**

[23]:
```
# apply kmeans algorithm
kmeans_model=KMeans(4)
kmeans_model.fit_predict(creditcard_scaled_df)
pca_df_kmeans= pd.concat([pca_df,pd.DataFrame({'cluster':kmeans_model.
  ↪labels_})],axis=1)
```

```
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
```

[24]:
```
# visualize the clustered dataframe
# Scatter Plot
plt.figure(figsize=(8,8))
#palette=['dodgerblue','red','green','blue','black','pink','gray','purple','coolwarm']
ax=sns.
 ↪scatterplot(x="pca1",y="pca2",hue="cluster",data=pca_df_kmeans,palette=['red','green','blue
plt.title("Clustering using K-Means Algorithm")
plt.show()
```



9.2 9.1. Analyzing Clustering Output

We've used K-Means model for clustering in this dataset.

[25]:
```
kmeans_model.cluster_centers_.shape
```

[25]:
```
(4, 17)
```

[26]:
```
# find all cluster centers
cluster_centers = pd.DataFrame(data=kmeans_model.
 ↪cluster_centers_,columns=[creditcard_df.columns])
# inverse transfor the data
cluster_centers = scalar.inverse_transform(cluster_centers)
cluster_centers = pd.DataFrame(data=cluster_centers,columns=[creditcard_df.
 ↪columns])
cluster_centers
```

[26]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
0	2883.331725	0.957727	311.733629	206.334852
1	1575.657597	0.975533	3393.301509	2197.577316
2	750.736440	0.936642	896.961919	309.143975
3	1142.858409	0.862488	286.573297	236.849684

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
0	105.433018	3138.561602	0.235644
1	1196.590681	294.622618	0.923300
2	588.108933	154.390396	0.857646
3	49.994545	505.650921	0.174426

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
0	0.116352	0.135821
1	0.676218	0.686154
2	0.220664	0.705934
3	0.102639	0.070199

	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT
0	0.458565	12.202170	5.112426	5107.215648
1	0.053737	1.073431	49.429907	6076.368491
2	0.033873	0.598140	18.399902	3895.678048
3	0.106460	1.861278	3.170677	3191.923559

	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
0	1856.682016	1251.200337	0.031366	11.156805
1	2849.429760	736.837829	0.268146	11.869159
2	1039.543663	594.155004	0.282098	11.724425
3	899.686626	611.358225	0.061514	11.563534

[27]:
```
# create a column as "cluster" & store the respective cluster name that they␣
 ↪belongs to
creditcard_cluster_df = pd.concat([creditcard_df,pd.DataFrame({'cluster':
 ↪kmeans_model.labels_})],axis=1)
```

```
creditcard_cluster_df.head()
```

[27]:

	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	cluster
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.000000	0.083333	0.000000	0	2	1000.0	201.802084	139.509787	0.000000	12	0.0
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.000000	0.000000	0.250000	4	0	7000.0	4103.032597	1072.340217	0.222222	12	2.0
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	1.000000	0.000000	0.000000	0	12	7500.0	622.066742	627.284787	0.000000	12	3.0
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	0.083333	0.000000	0.083333	1	1	7500.0	0.000000	864.206542	0.000000	12	2.0
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	0.083333	0.000000	0.000000	0	1	1200.0	678.334763	244.791237	0.000000	12	2.0

9.3 9.2 Outcome

-> There are 4 clusters (segments)- each clusters are shown below in detail: * First Customers cluster (Transactors): Those are customers who pay least amount of interest charges and careful with their money, Cluster with lowest balance (104 Dollar) and cash advance (303 Dollar), Percentage of full payment = 23%

- Second customers cluster (revolvers) who use credit card as a loan (most lucrative sector): highest balance (5000 Dollar) and cash advance (5000 Dollar), low purchase frequency, high cash advance frequency (0.5), high cash advance transactions (16) and low percentage of full payment (3%)

- Third customer cluster (VIP/Prime): high credit limit 16K Dollar and highest percentage of full payment, target for increase credit limit and increase spending habits
- Fourth customer cluster (low tenure): these are customers with low tenure (7 years), low balance

9.4 9.3. Analysis of each Cluster

9.4.1 Cluster - 1

[30]:
```
cluster_1_df = creditcard_cluster_df[creditcard_cluster_df["cluster"]==0]
cluster_1_df.sort_values(by=['BALANCE'], ascending=False).head()
```

[30]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
2361	15532.33972	1.0	1168.75	0.0
124	14224.11541	1.0	0.00	0.0
4089	13968.47957	1.0	281.71	8.9
723	13774.74154	1.0	404.24	0.0
380	12474.72954	1.0	136.88	0.0

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
2361	1168.75	3183.037625	0.916667
124	0.00	4614.427403	0.000000
4089	272.81	2710.679764	0.416667
723	404.24	3369.474535	0.250000
380	136.88	515.147607	0.166667

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
2361	0.000000	0.916667
124	0.000000	0.000000
4089	0.083333	0.333333
723	0.000000	0.250000
380	0.000000	0.166667

	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT
2361	0.250000	5	11	16500.0
124	0.333333	9	0	19000.0
4089	0.666667	12	9	18500.0
723	0.500000	7	3	14500.0
380	0.166667	2	2	14000.0

	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE	cluster
2361	3906.738592	3379.593046	0.0	12	0.0
124	3066.614272	3406.258999	0.0	12	0.0
4089	3464.441992	3360.086085	0.0	12	0.0
723	3167.870886	3533.464800	0.0	12	0.0
380	3519.008859	3430.627754	0.0	12	0.0

9.4.2 Cluster - 2

[29]:
```
cluster_2_df = creditcard_cluster_df[creditcard_cluster_df["cluster"]==1]
cluster_2_df.sort_values(by=['BALANCE'], ascending=False).head()
```

[29]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	\
501	13479.28821	1.0	41050.4	40624.06	
495	12478.17286	1.0	174.0	174.00	
866	11654.55492	1.0	463.0	74.00	
3210	10871.08518	1.0	0.0	0.00	
755	10397.09989	1.0	0.0	0.00	

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
501	426.34	0.000000	0.833333	
495	0.00	3269.418821	0.250000	
866	389.00	3096.807933	0.583333	
3210	0.00	4822.559803	0.000000	
755	0.00	4045.620171	0.000000	

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	\
501	0.666667	0.416667	
495	0.250000	0.000000	
866	0.083333	0.416667	
3210	0.000000	0.000000	
755	0.000000	0.000000	

	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	\
501	0.000000	0	157	17000.0	
495	0.666667	21	3	14000.0	
866	0.416667	17	7	12500.0	
3210	0.166667	3	0	18000.0	
755	0.250000	6	0	13000.0	

	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE	cluster
501	36066.750680	15914.484620	0.083333	12	1.0
495	3251.190662	3872.099498	0.000000	12	1.0
866	3024.609470	5148.045052	0.000000	12	1.0
3210	2735.624602	2595.765441	0.000000	12	1.0
755	3222.169406	2818.707479	0.000000	12	1.0

9.4.3 Cluster - 3 (Silver)

[31]:
```
cluster_3_df = creditcard_cluster_df[creditcard_cluster_df["cluster"]==2]
cluster_3_df.sort_values(by=['BALANCE'], ascending=False).head()
```

[31]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	\
138	19043.13856	1.0	22009.92	9449.07	

```
5488  16304.88925                1.0    1770.57             0.00
5281  16115.59640                1.0     684.74           105.30
585   15244.74865                1.0    7823.74          7564.81
883   14581.45914                1.0       0.00             0.00

      INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
138                12560.85      0.000000                  1.0
5488                1770.57   7424.094447                  0.5
5281                 579.44   4354.002428                  1.0
585                  258.93   2621.049473                  1.0
883                    0.00  22665.778500                  0.0

      ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
138                     0.750000                          1.000000
5488                    0.000000                          0.416667
5281                    0.083333                          1.000000
585                     1.000000                          1.000000
883                     0.000000                          0.000000

      CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
138                 0.000000                 0            216       18000.0
5488                0.666667                13              9       19000.0
5281                0.583333                15             15       18000.0
585                 0.083333                 2             62       19000.0
883                 0.833333                30              0       18500.0

          PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT  TENURE  cluster
138   23018.575830      18621.013310               0.0      12      2.0
5488   5337.961195       8345.641905               0.0      12      2.0
5281   3546.061550       5743.736444               0.0      12      2.0
585   11123.409180       4467.520244               0.0      12      2.0
883   20941.325510       5433.759888               0.0      12      2.0
```

9.4.4 Cluster - 4

[32]:
```
cluster_4_df = creditcard_cluster_df[creditcard_cluster_df["cluster"] == 3]
cluster_4_df.sort_values(by=['BALANCE'], ascending=False).head()
```

[32]:
```
          BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  \
4140  18495.55855                1.0    5288.28           3657.30
520   15258.22590                1.0     529.30            529.30
4708  15155.53286                1.0     717.24            717.24
5913  13777.37772                1.0       0.00              0.00
153   13673.07961                1.0    9792.23           3959.81

      INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
4140                 1630.98      0.000000                  1.0
```

```
520                    0.00   4100.891579                  0.5
4708                   0.00   4718.274895                  1.0
5913                   0.00   1675.249576                  0.0
153                 5832.42   2444.445738                  1.0

      ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
4140                    0.583333                               1.0
520                     0.500000                               0.0
4708                    1.000000                               0.0
5913                    0.000000                               0.0
153                     0.750000                               1.0

      CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
4140                0.000000                 0             76       22000.0
520                 1.000000                23             10       19000.0
4708                0.500000                 7             24       18000.0
5913                0.666667                11              0       14500.0
153                 0.750000                26            216       20000.0

          PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT  TENURE  cluster
4140   4246.168346       4227.081580               0.0      12      3.0
520    2051.146470       3905.740148               0.0       8      3.0
4708   4002.194556       3843.924668               0.0      12      3.0
5913   3054.844697       3242.471295               0.0      12      3.0
153   11717.307940       6042.391629               0.0      12      3.0
```

9.5 Optional

10 10. Save The Model

[33]:
```
#Saving Scikitlearn models
import joblib
joblib.dump(kmeans_model, "kmeans_model.pkl")
```

[33]:
```
['kmeans_model.pkl']
```

[34]:
```
# save the dataframe in .csv file named as "Clustered_Costumer_Data"
creditcard_cluster_df.to_csv("Clustered_Customer_Data.csv")
```

[ ]:
```

```