Market Segmentation in SBI life Insurance

1. Overview

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 (https://www.kaggle.com/arjunbhasin2013/ccdata)](https://www.kaggle.com/arjunbhasin2013/ccdata)

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

2. Import Libraries:

In [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,SpectralClus
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_samples, silhouette_score
```

3. Load Dataset:

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

Out[2]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALI
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	

4.Exploratory Data Analysis & Data Cleaning:

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

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

In [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
```

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

Out[5]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMEN
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	

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

Out[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
```

In [7]:
```
# find all columns having missing values
missing_var = [var for var in creditcard_df.columns if creditcard_df[var].isnu
missing_var
```

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

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

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

Out[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
```

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

Out[10]:
```
0
```

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

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

Out[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')
```

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

Out[13]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PI
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	

5. Outlier Detection

In [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) &
  print(" outlier in ",i,"is" ,int(((creditcard_df.shape[0]-creditcard_df_no_o
```

```
 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 %
```

In [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_
```

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

Out[16]:

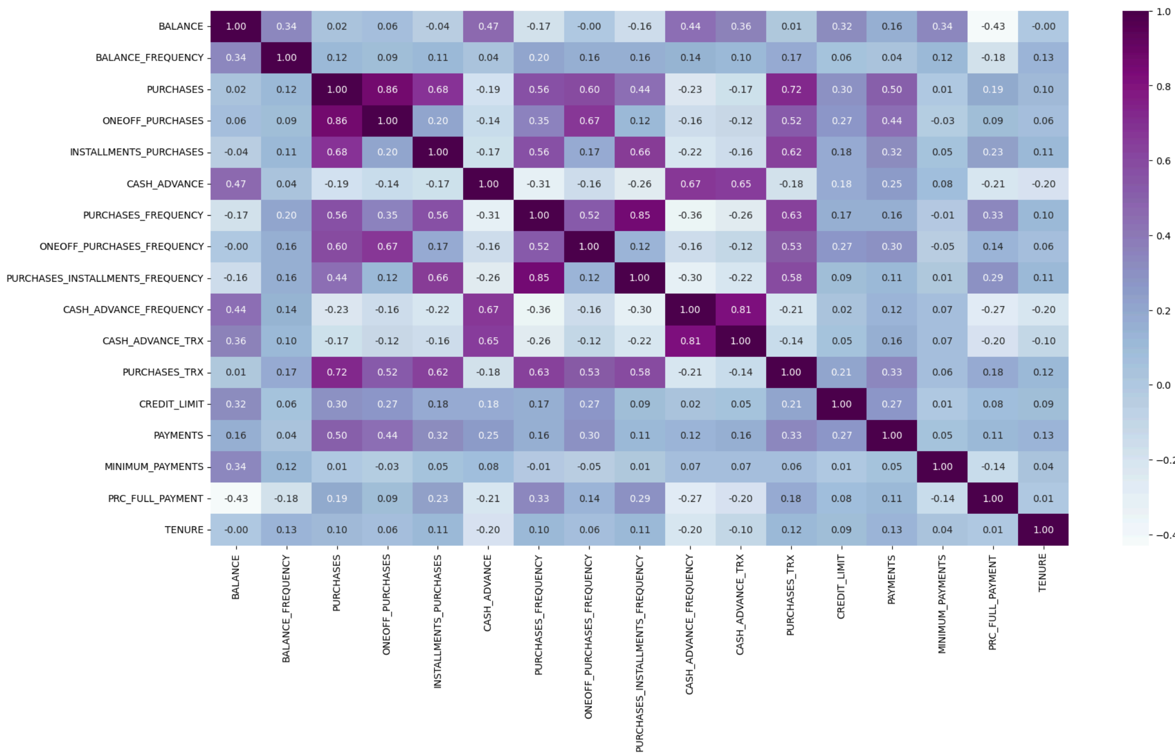
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PU
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
4	817.714335	1.000000	16.00	16.00	
5	1809.828751	1.000000	1333.28	0.00	
7	1823.652743	1.000000	436.20	0.00	

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

Out[17]: (6466, 17)

In [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')
```

Out[18]: <Axes: >



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. 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

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

In [20]:
```
creditcard_scaled_df
```

Out[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. 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.

In [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()
```

Out[21]:

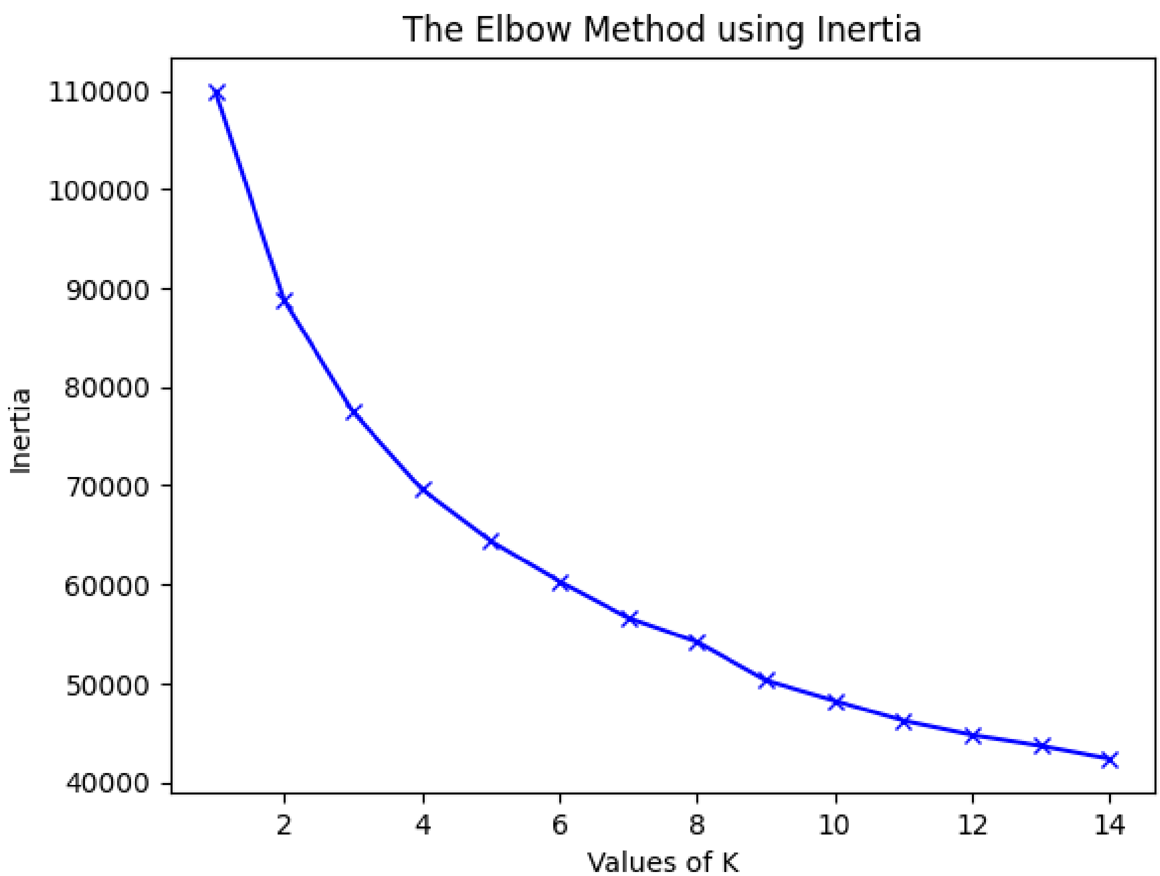
	pca1	pca2
0	-2.286555	3.003828
1	1.134715	0.431969
2	-1.458103	-1.493204
3	0.740689	-0.539416
4	0.648374	-1.077139

8. Hyperparameter tuning

In [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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
```

```
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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 ch
ange from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss 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.

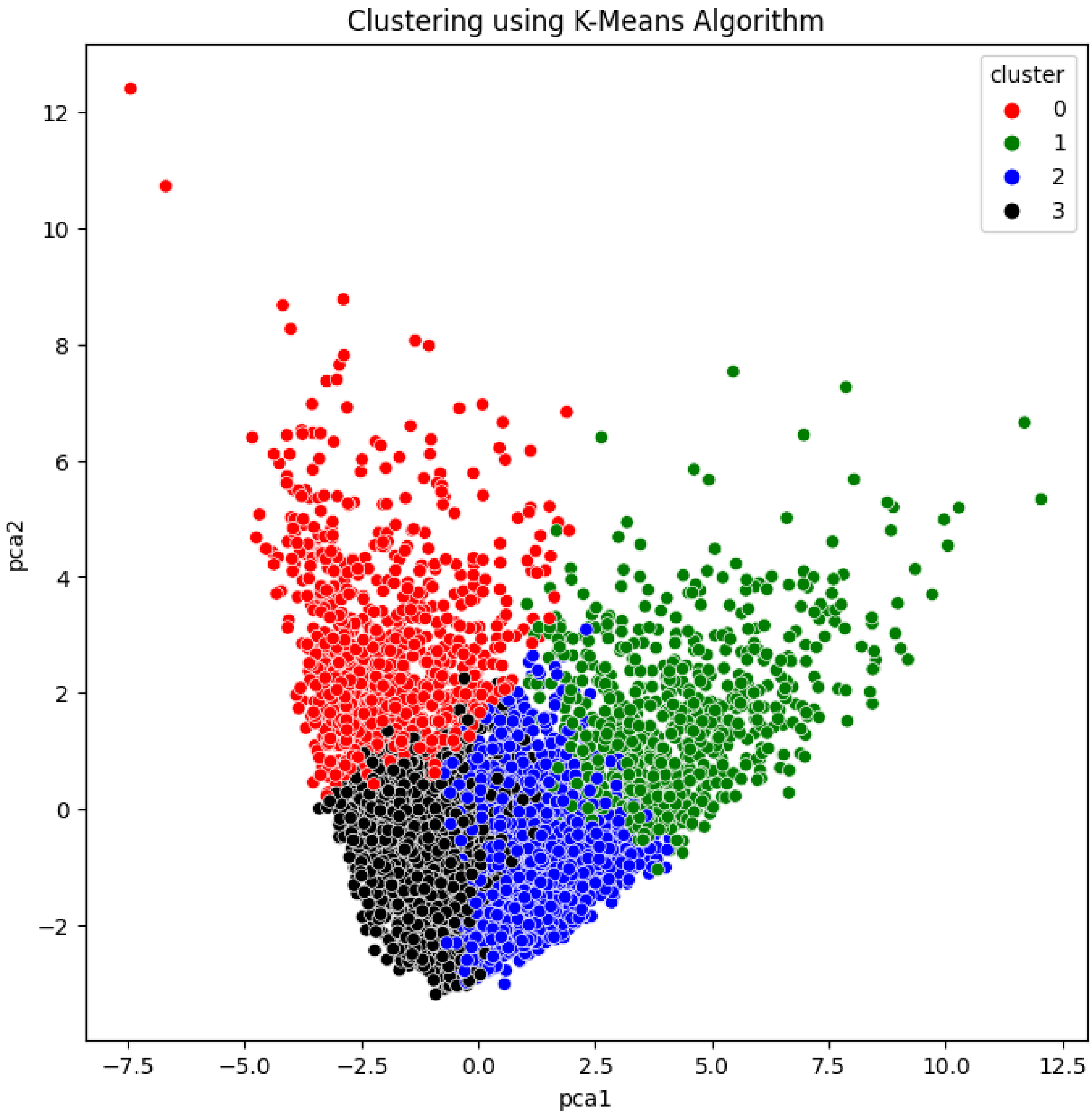
9. Model Building

** K-Means Clustering**

In [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_}
```

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

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



9.1. Analyzing Clustering Output

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

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

Out[25]: (4, 17)

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

Out[26]:

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

In [27]:
```
# create a column as "cluster" & store the respective cluster name that they b
creditcard_cluster_df = pd.concat([creditcard_df,pd.DataFrame({'cluster':kmean
creditcard_cluster_df.head()
```

Out[27]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PU
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	

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

9.3. Analysis of each Cluster

Cluster - 1

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

Out[30]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS
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	

Cluster - 2

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

Out[29]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS
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	

Cluster - 3 (Silver)

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

Out[31]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS
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	

Cluster - 4

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

Out[32]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS
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	

Optional

10. Save The Model

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

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

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

In [ ]: