AIM

Program to implement k-means clustering technique using any standard dataset available in the public domain.

Programming code:

Dataset used: GENERAL.csv

```
# importing the libraries
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt

dataset= pd.read_csv('./CC GENERAL.csv')

# checking the presence of null values
print(dataset.isnull().sum())
 #CREDIT_LIMIT                           1
 #MINIMUM_PAYMENTS                     313
```

OUTPUT:

```
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
```

Programming code:

```
dataset['CREDIT_LIMIT'].fillna(dataset.CREDIT_LIMIT.mean(), inplace
 = True)
dataset['MINIMUM_PAYMENTS'].fillna(dataset.MINIMUM_PAYMENTS.mean(),
 inplace = True) # unfilled vaues replaced using mean

print(dataset.isnull().sum())

print(dataset.describe())
```

OUTPUT:

```
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
            BALANCE  BALANCE_FREQUENCY  ...  PRC_FULL_PAYMENT       TENURE
count   8950.000000         8950.000000  ...       8950.000000  8950.000000
mean    1564.474828            0.877271  ...          0.153715    11.517318
std     2081.531879            0.236904  ...          0.292499     1.338331
min        0.000000            0.000000  ...          0.000000     6.000000
25%      128.281915            0.888889  ...          0.000000    12.000000
50%      873.385231            1.000000  ...          0.000000    12.000000
75%     2054.140036            1.000000  ...          0.142857    12.000000
max    19043.138560            1.000000  ...          1.000000    12.000000
```

Programming code:

```
dataset.drop(['CUST_ID'], axis= 1, inplace = True) #no relevance fo
r custid

# No Categorical Values found
X = dataset.iloc[:,:].values
```

```
# Using standard scaler
from sklearn.preprocessing import StandardScaler
standardscaler= StandardScaler()
X = standardscaler.fit_transform(X)  #scaling the values
print(X)
```

OUTPUT:

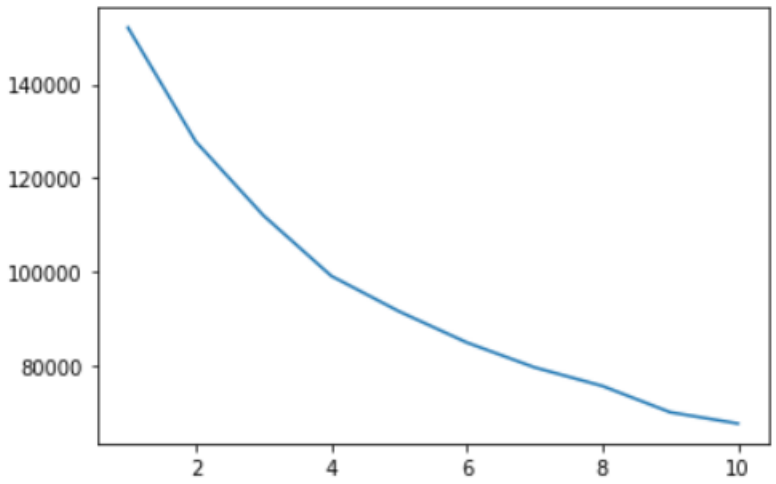
```
[[-0.73198937 -0.24943448 -0.42489974 ... -0.31096755 -0.52555097
   0.36067954]
 [ 0.78696085  0.13432467 -0.46955188 ...  0.08931021  0.2342269
   0.36067954]
 [ 0.44713513  0.51808382 -0.10766823 ... -0.10166318 -0.52555097
   0.36067954]
 ...
 [-0.7403981  -0.18547673 -0.40196519 ... -0.33546549  0.32919999
  -4.12276757]
 [-0.74517423 -0.18547673 -0.46955188 ... -0.34690648  0.32919999
  -4.12276757]
 [-0.57257511 -0.88903307  0.04214581 ... -0.33294642 -0.52555097
  -4.12276757]]
```

Programming code:

```
"""K MEANS CLUSTERING """
#Inertia, or the within-
cluster sum of squares criterion, can be recognized as a measure of
 how internally coherent clusters are
from sklearn.cluster import KMeans
wss= []
for i in range(1, 11):
    kmeans= KMeans(n_clusters = i, init = 'k-
means++', random_state = 0)
    kmeans.fit(X)
    wss.append(kmeans.inertia_)
plt.plot(range(1,11), wss) # selecting 4
```

OUTPUT:

```
[<matplotlib.lines.Line2D at 0x7f74661e8a90>]
```



Programming code:

```
wss_mean=np.array(wss).mean()
print(wss)
print(wss_mean)
print([abs(wss_mean-x) for x in wss])
k=np.argmin([abs(wss_mean-x) for x in wss])+1
```

OUTPUT:

```
[152149.99999999983, 127784.92103208725, 111986.41162208859,
99073.93826774803, 91502.98328256077, 84851.13240432573,
79532.40237691796, 75568.97609993909, 69954.91393943134,
67546.56302862825]
95995.22420537268
[56154.775794627145, 31789.69682671457, 15991.187416715911,
3078.714062375351, 4492.240922811907, 11144.091801046947,
16462.82182845472, 20426.248105433595, 26040.31026594134,
28448.661176744426]
```

Programming code:

```
kmeans = KMeans(n_clusters = k, init= 'k-
means++', random_state = 0)
kmeans.fit(X)
```

```
Y_pred_K= kmeans.predict(X)
print(Y_pred_K)



#showing the clusters of first 100 persons
plt.figure(figsize=(16,4))
plt.plot(range(1,100+1),Y_pred_K[:100],'ro')
```

OUTPUT:

```
[3 1 0 ... 0 3 3]
[<matplotlib.lines.Line2D at 0x7f745836e290>]
```