2306AML117 MAHESH BABU M assignment 10

In [4]:
```
import numpy as np

import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [5]:
```
creditcard=pd.read_csv('creditcard.csv')
```

In [6]:
```
creditcard.head()
```

Out[6]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.000000
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.000000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	1.000000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	0.083333
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	0.083333

DATA PREPROCESSING

In [7]:
```
creditcard.dtypes
```

Out[7]:
```
CUST_ID                            object
BALANCE                           float64
BALANCE_FREQUENCY                 float64
PURCHASES                         float64
ONEOFF_PURCHASES                  float64
INSTALLMENTS_PURCHASES            float64
CASH_ADVANCE                      float64
PURCHASES_FREQUENCY               float64
ONEOFF_PURCHASES_FREQUENCY        float64
PURCHASES_INSTALLMENTS_FREQUENCY  float64
CASH_ADVANCE_FREQUENCY            float64
CASH_ADVANCE_TRX                    int64
PURCHASES_TRX                       int64
CREDIT_LIMIT                      float64
PAYMENTS                          float64
MINIMUM_PAYMENTS                  float64
PRC_FULL_PAYMENT                  float64
TENURE                              int64
dtype: object
```

In [8]:
```
creditcard.isnull().sum()
```

Out[8]:
```
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 [9]:
```
creditcard.describe()
```

Out[9]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PUR
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351	0.202458	
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371	0.298336	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	0.083333	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000	0.083333	
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667	0.300000	
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000	1.000000	

In [10]:
```
creditcard['MINIMUM_PAYMENTS'].fillna(creditcard['MINIMUM_PAYMENTS'].mean(),inplace=True)# fill missing vallue with mean of sample
```

In [11]:
```
creditcard['CREDIT_LIMIT'].fillna(creditcard['CREDIT_LIMIT'].mean(),inplace=True) # fill missing vallue with mean of sample'].fillna(creditcard['''].mean(),inpl
```

In [12]:
```
creditcard.isnull().sum()
```

Out[12]:
```
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 [13]:
```
creditcard1=creditcard.drop(['CUST_ID'],axis=1)
```

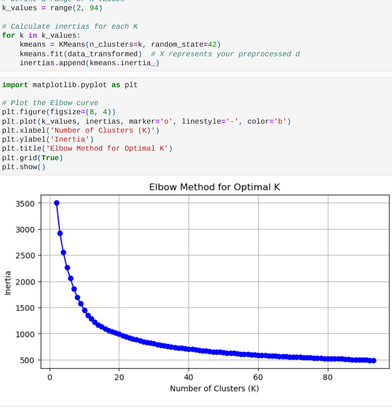
In [14]:
```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
mms.fit(creditcard1)
data_transformed = mms.transform(creditcard1)
data_transformed
```

Out[14]:
```
array([[2.14779454e-03, 8.18182000e-01, 1.94536779e-03, ...,
        1.82564563e-03, 0.00000000e+00, 1.00000000e+00],
       [1.68169097e-01, 9.09091000e-01, 0.00000000e+00, ...,
        1.40344791e-02, 2.22222000e-01, 1.00000000e+00],
       [1.31026136e-01, 1.00000000e+00, 1.57662475e-02, ...,
        8.20961806e-03, 0.00000000e+00, 1.00000000e+00],
       ...,
       [1.22871936e-03, 8.33333000e-01, 2.94456089e-03, ...,
        1.07843629e-03, 2.50000000e-01, 0.00000000e+00],
       [7.0688341e-04, 8.33333000e-01, 0.00000000e+00, ...,
        7.29475795e-04, 2.50000000e-01, 0.00000000e+00],
       [1.95717777e-02, 6.66667000e-01, 2.22932216e-02, ...,
        1.15527021e-03, 0.00000000e+00, 0.00000000e+00]])
```

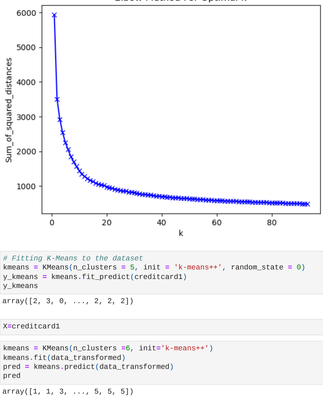
In [15]:
```
Sum_of_squared_distances = []
K = range(1,94)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(data_transformed)
    Sum_of_squared_distances.append(km.inertia_)
```

In [19]:
```
from sklearn.cluster import KMeans

# Initialize an empty list to store inertias
inertias = []

# Define a range of K values
k_values = range(2, 94)

# Calculate inertias for each K
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_transformed)  # X represents your preprocessed d
    inertias.append(kmeans.inertia_)
```

In [20]:
```
import matplotlib.pyplot as plt

# Plot the Elbow curve
plt.figure(figsize=(8, 4))
plt.plot(k_values, inertias, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()
```



In [14]:
```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



In [21]:
```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 0)
y_kmeans = kmeans.fit_predict(creditcard1)
y_kmeans
```

Out[21]:
```
array([2, 3, 0, ..., 2, 2, 2])
```

In [22]:
```
X=creditcard1
```

In [23]:
```
kmeans = KMeans(n_clusters =6, init='k-means++')
kmeans.fit(data_transformed)
pred = kmeans.predict(data_transformed)
pred
```

Out[23]:
```
array([1, 1, 3, ..., 5, 5, 5])
```

In [24]:
```
frame = pd.DataFrame(data_transformed)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Out[24]:
```
1    3271
0    2119
3    1175
4    1069
2     827
5     489
Name: cluster, dtype: int64
```

In [30]:
```
import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.metrics import pairwise_distances

kmeans_model = KMeans(n_clusters=3, random_state=1).fit(X)
labels = kmeans_model.labels_
print(labels)
```

```
[0 2 2 ... 0 0 0]
```

In [31]:
```
metrics.silhouette_score(X, labels, metric='euclidean')
```

Out[31]:
```
0.4653592350091546
```

In [ ]:

In [ ]: