Experiment-8

November 26, 2023

Name: Adwait Purao UID: 2021300101 Batch: B2

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[3]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

[64]:
```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/CC GENERAL.csv")
```

[10]:
```
df.head()
```

[10]:

	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

[11]:
```
df.tail()
```

[11]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	\
8945	C19186	28.493517	1.000000	291.12	0.00	
8946	C19187	19.183215	1.000000	300.00	0.00	
8947	C19188	23.398673	0.833333	144.40	0.00	
8948	C19189	13.457564	0.833333	0.00	0.00	
8949	C19190	372.708075	0.666667	1093.25	1093.25	

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
8945	291.12	0.000000	1.000000	
8946	300.00	0.000000	1.000000	
8947	144.40	0.000000	0.833333	
8948	0.00	36.558778	0.000000	
8949	0.00	127.040008	0.666667	

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	\
8945	0.000000	0.833333	
8946	0.000000	0.833333	
8947	0.000000	0.666667	
8948	0.000000	0.000000	
8949	0.666667	0.000000	

	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	\
8945	0.000000	0	6	1000.0	
8946	0.000000	0	6	1000.0	
8947	0.000000	0	5	1000.0	
8948	0.166667	2	0	500.0	
8949	0.333333	2	23	1200.0	

	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
8945	325.594462	48.886365	0.50	6

8946	275.861322	NaN	0.00	6
8947	81.270775	82.418369	0.25	6
8948	52.549959	55.755628	0.25	6
8949	63.165404	88.288956	0.00	6

[12]:
```
df.describe()
```

[12]:

	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

[40]:
```
# Finding missing values
missing_data = df.isna()
missing_counts = missing_data.sum()
print(missing_counts)
```

```
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
```

[65]:
```
df = df.drop("CUST_ID", axis=1)

# Filling NaN with mean of the values
df = df.fillna(df.mean())

# For scaling the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(df)
```

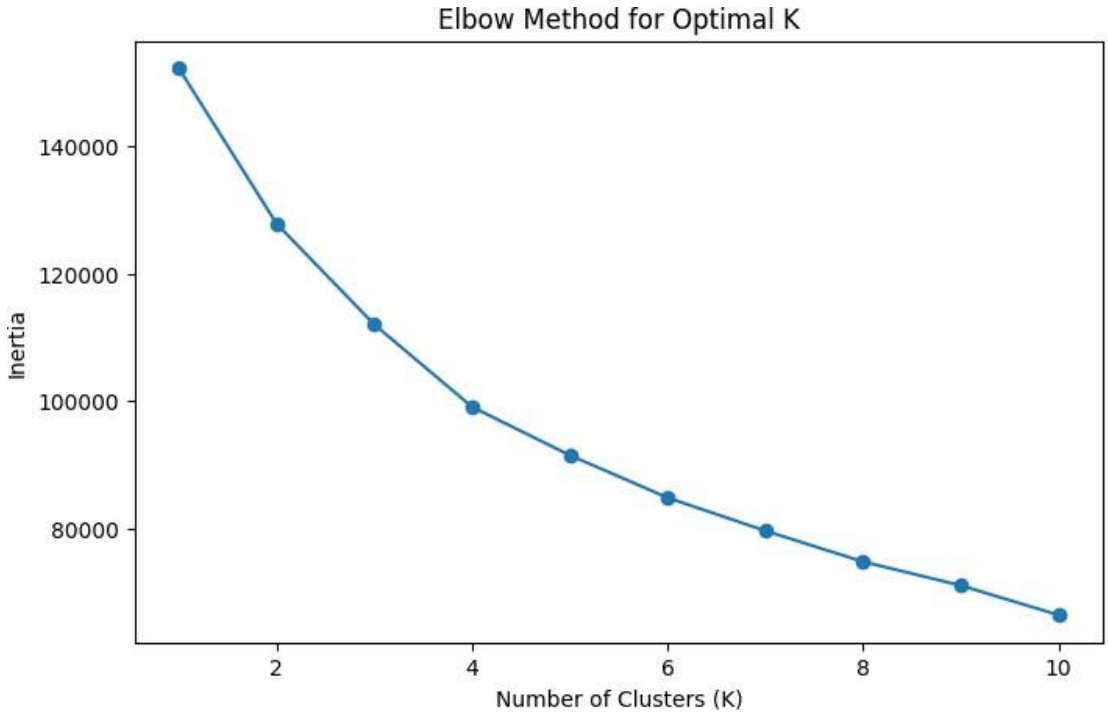
[46]:
```
missing_data = df.isna()
missing_counts = missing_data.sum()
print(missing_counts)
```

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
Cluster	0

dtype: int64

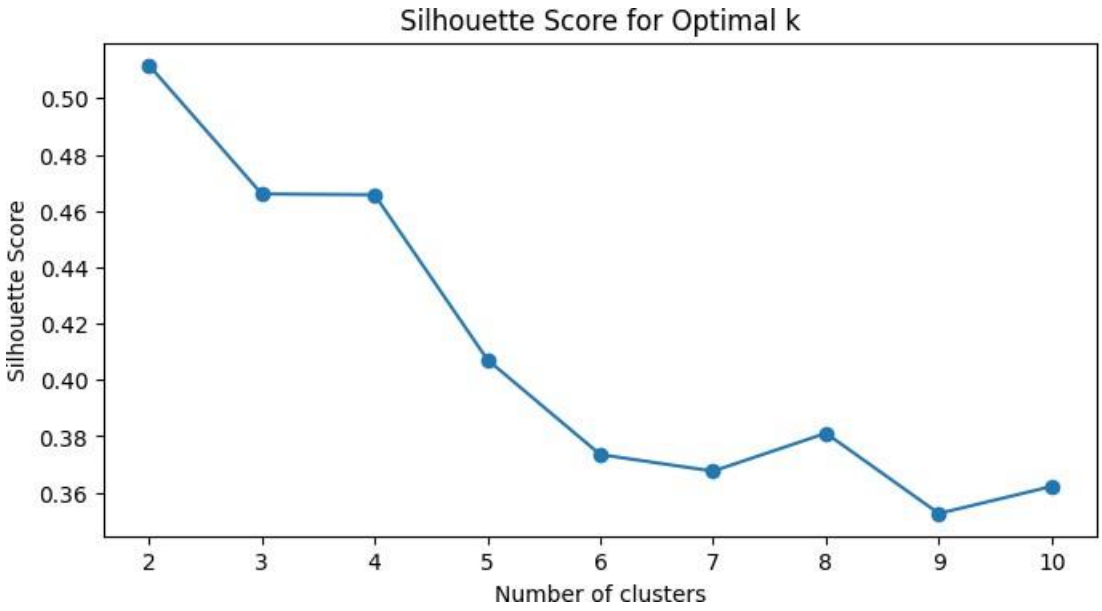
Using Elbow method to find optimal k

[66]:
```
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=101, n_init=10)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow curve plt.figure(figsize=(8,
5)) plt.plot(range(1, 11), inertia,
marker="o") plt.title("Elbow Method for
Optimal K") plt.xlabel("Number of Clusters
(K)") plt.ylabel("Inertia")
plt.show()
```



Verifing Silhoute Score to get the optimal K

[67]:
```
from sklearn.metrics import silhouette_score

silhouette_scores = []
K_range = range(2, 11)
for k in K_range:
    kmeanModel = KMeans(n_clusters=k,random_state=101,n_init = 10)
    kmeanModel.fit(df)
    labels = kmeanModel.labels_
    silhouette_avg = silhouette_score(df, labels)
    silhouette_scores.append(silhouette_avg)

# Plot silhouette scores
plt.figure(figsize=(8, 4))
plt.plot(K_range, silhouette_scores, marker="o")
plt.title("Silhouette Score for Optimal k")
plt.xlabel("Number of clusters")
plt.ylabel("Silhouette Score")
plt.show()
```



Since we the silhoute score for k = 2 is the highest, we choose k = 2 as the optimal k.

[59]:
```
optimal_k = 2
kmeans = KMeans(n_clusters=optimal_k, random_state=101, n_init=10)
df["Cluster"] = kmeans.fit_predict(data_scaled)
```

[60]:
```
# Analyze the characteristics of each cluster
cluster_means = df.groupby("Cluster").mean()
cluster_means
```

[60]:

Cluster	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	\
0	1378.130586	0.945460	2024.075484	1141.435320	
1	1697.139951	0.828725	276.410541	201.586115	

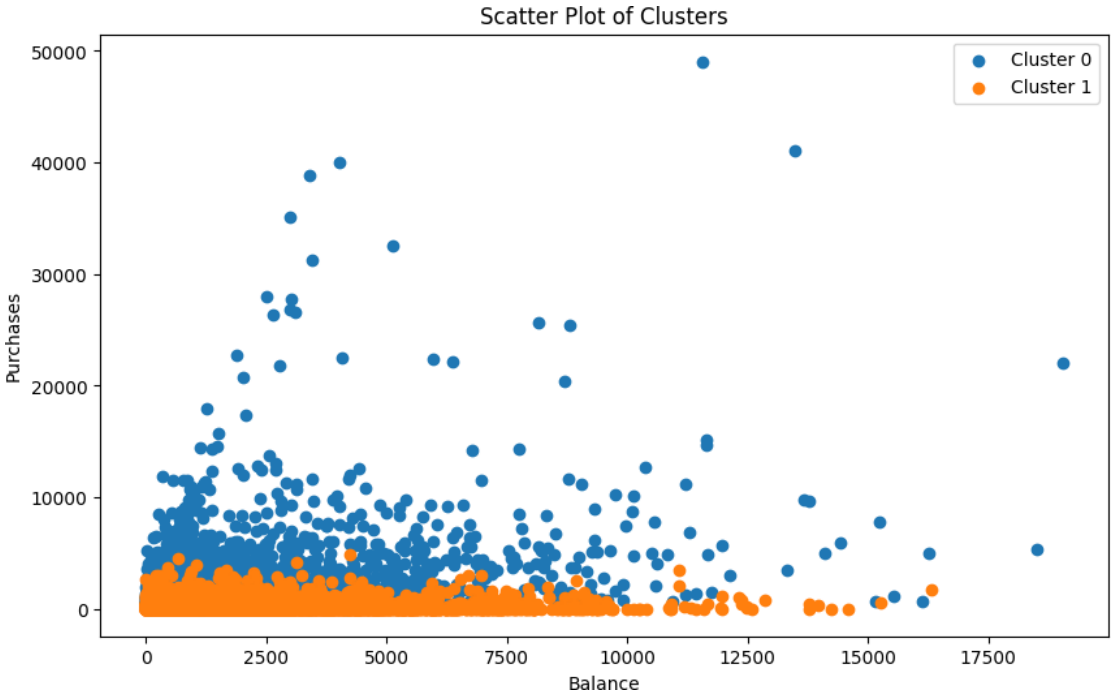
Cluster	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
0	882.969151	435.925695	0.904712	
1	75.104101	1365.413355	0.195352	

Cluster	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	\
0	0.362118	0.730082	
1	0.088790	0.104122	

Cluster	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	\

```
0                    0.058397          1.411069      30.657174
1                    0.189783          4.557192       3.356350

         CREDIT_LIMIT     PAYMENTS   MINIMUM_PAYMENTS   PRC_FULL_PAYMENT  \
Cluster
0         5095.773845  2184.020580        872.555375           0.270892
1         4066.345128  1412.148599        858.262710           0.070292

               TENURE
Cluster
0           11.674637
1           11.405318
```

[61]:
```
selected_clusters = [0, 1]

plt.figure(figsize=(10, 6))
for cluster in selected_clusters:
    cluster_data = df[df["Cluster"] == cluster]
    plt.scatter(
        cluster_data["BALANCE"], cluster_data["PURCHASES"], label=f"Cluster␣
 ↪{cluster}"
    )

plt.title("Scatter Plot of Clusters")
plt.xlabel("Balance")
plt.ylabel("Purchases")
plt.legend()
plt.show()
```



Analyzing the Cluster Produced

The presented plot demonstrates the segmentation of data points into two distinct clusters. In Cluster 1, individuals exhibit low spending scores, while in Cluster 0, individuals are characterized by high spending scores. This binary clustering suggests a clear division between those with relatively conservative financial behaviors (Cluster 1) and those with more extravagant spending habits (Cluster 0).