Experiment 10 ML (K Means Clustering Technique)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('CreditCard.csv')
df.head()
```

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

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.shape
```

```
(8950, 18)
```

```
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
```

```
df.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
count	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
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371
min	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
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000

We observe that two columns contains null or blank values these columns are 'CREDIT LIMIT' and 'MINIMUM PAYMENTS'. We assume that these columns may contain outliers. Therefore we replace missing values in these columns with median value of column.

```
# Filling the missing values with their column median

df['CREDIT_LIMIT'] = df['CREDIT_LIMIT'].fillna(df['CREDIT_LIMIT'].median())
df['MINIMUM_PAYMENTS'] = df['MINIMUM_PAYMENTS'].fillna(df['MINIMUM_PAYMENTS'].median())
```

```
# dropping  first column
new_df = df.drop('CUST_ID', axis=1)
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 17 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   BALANCE                           8950 non-null   float64
 1   BALANCE_FREQUENCY                 8950 non-null   float64
 2   PURCHASES                         8950 non-null   float64
 3   ONEOFF_PURCHASES                  8950 non-null   float64
 4   INSTALLMENTS_PURCHASES            8950 non-null   float64
 5   CASH_ADVANCE                      8950 non-null   float64
 6   PURCHASES_FREQUENCY               8950 non-null   float64
 7   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 8   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 9   CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 10  CASH_ADVANCE_TRX                  8950 non-null   int64  
 11  PURCHASES_TRX                     8950 non-null   int64  
 12  CREDIT_LIMIT                      8950 non-null   float64
 13  PAYMENTS                          8950 non-null   float64
 14  MINIMUM_PAYMENTS                  8950 non-null   float64
 15  PRC_FULL_PAYMENT                  8950 non-null   float64
 16  TENURE                            8950 non-null   int64  
dtypes: float64(14), int64(3)
memory usage: 1.2 MB
```

```
# scaling all numerical columns
col_names = new_df.columns
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(new_df), columns = col_names)
df_scaled.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCH
0	-0.731989	-0.249434	-0.424900	-0.356934	-0.349079	-0.466786	-0.806490	
1	0.786961	0.134325	-0.469552	-0.356934	-0.454576	2.605605	-1.221758	
2	0.447135	0.518084	-0.107668	0.108889	-0.454576	-0.466786	1.269843	
3	0.049099	-1.016953	0.232058	0.546189	-0.454576	-0.368653	-1.014125	
4	-0.358775	0.518084	-0.462063	-0.347294	-0.454576	-0.466786	-1.014125	

Next steps: Generate code with df_scaled View recommended plots New interactive sheet

```
# k means clustering algorithm

from sklearn.cluster import KMeans
```

```
km = KMeans(n_clusters=3, random_state=4)
# we start by guessing number of clusters to be 3
```

```
kmeans_fit = km.fit(df_scaled)
```

```
km.labels_ # shows unique labels assigned by the algorithm
```

```
array([2, 1, 2, ..., 2, 2, 2], dtype=int32)
```

```
from sklearn.metrics import silhouette_score, silhouette_samples
```

```
silhouette_score(df_scaled, km.labels_)
```

```
np.float64(0.25098792290537314)
```

```
# Silhouette score is ranging between -1 to 1
# Silhouette score = 1 ideal case (perfect fit for selected k)
# Silhouette score = 0 data points are close to boundaries
# Silhouette score = -1 data points are assigned to wrong group
```

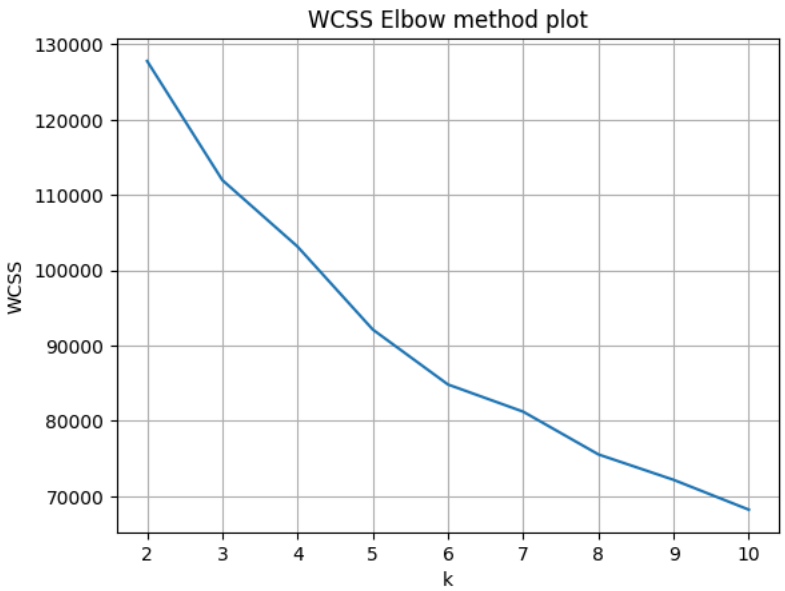
```
# determination of best no.of clusters for guven dataset
wcss =[] # wcss = within clusted sum of squares
s1 =[] # empty list for storing silhouette
k=10 # initialize total no. of clusters to test
for i in range(2,k+1):
  kmeans = KMeans(n_clusters=i,random_state=4)
  kmeans.fit(df_scaled)
  wcss.append(kmeans.inertia_)
  score = silhouette_score(df_scaled,kmeans.labels_)
  s1.append(score)
```

```
# for each value of k display WCSS and Silhouette score
optimal_k = pd.DataFrame({'k': range(2,k+1),'WCSS':wcss,'Silhouette_score':s1})
optimal_k
```

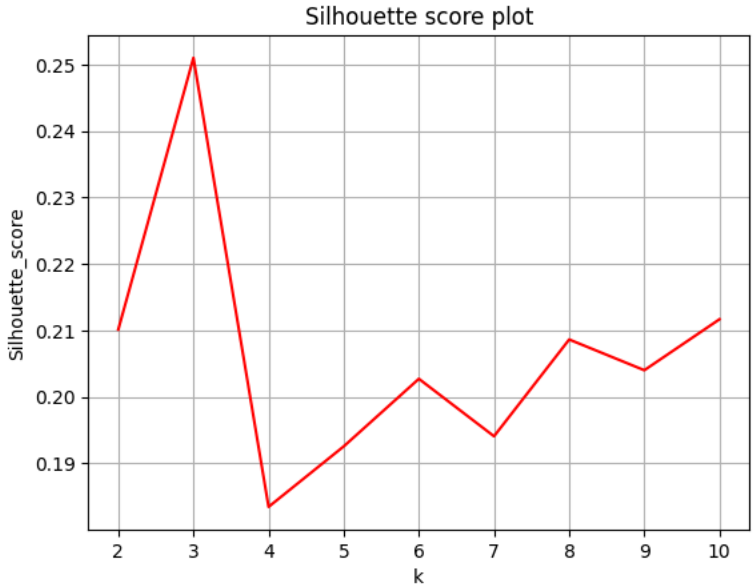
	k	WCSS	Silhouette_score
0	2	127784.842986	0.210132
1	3	111975.043593	0.250988
2	4	103145.612421	0.183471
3	5	92131.465545	0.192566
4	6	84829.997103	0.202724
5	7	81221.525853	0.194086
6	8	75545.268317	0.208639
7	9	72168.365836	0.204017
8	10	68206.537103	0.211677

Next steps: Generate code with optimal_k View recommended plots New interactive sheet

```
# plotting the scores

sns.lineplot(x='k', y='WCSS', data=optimal_k)
plt.title('WCSS Elbow method plot')
plt.grid(True)
plt.show()
```




```
sns.lineplot(x = 'k', y = 'Silhouette_score', data = optimal_k, color='red')
plt.title('Silhouette score plot')
plt.grid(True)
plt.show()
```



- Silhouette score is maximum at k = 3 in this case
- Elbow

```
# Redesigning clusters with k = 2
kmeans = KMeans(n_clusters=3, random_state=4)
kmeans.fit(df_scaled)
```

```
KMeans
KMeans(n_clusters=3, random_state=4)
```

```
kmeans.cluster_centers_
```

```
array([[ 2.96205438e-01,  4.40360296e-01,  1.48850271e+00,
         1.24656973e+00,  1.22895441e+00, -2.52583955e-01,
         1.14091898e+00,  1.54867389e+00,  9.44426689e-01,
        -3.63280444e-01, -2.55290565e-01,  1.64732717e+00,
         8.63409159e-01,  8.08195752e-01,  1.63758763e-01,
         4.94330807e-01,  2.98653259e-01],
       [ 1.18245092e+00,  3.45711499e-01, -2.87338902e-01,
        -2.04809823e-01, -3.03202813e-01,  1.40224430e+00,
        -6.38601820e-01, -3.03635778e-01, -5.50248107e-01,
         1.58030175e+00,  1.36361348e+00, -3.64901585e-01,
         6.15299732e-01,  4.56793855e-01,  3.94840016e-01,
        -4.10350125e-01, -1.22632222e-01],
       [-3.63631243e-01, -1.80214303e-01, -2.37619707e-01,
        -2.08141046e-01, -1.79351936e-01, -3.05079568e-01,
        -7.53696535e-02, -2.46030548e-01, -5.68699223e-02,
        -3.27394340e-01, -2.94655811e-01, -2.51004470e-01,
        -3.37386075e-01, -2.85402321e-01, -1.34969796e-01,
         1.45358322e-03, -3.10933985e-02]])
```

```
kmeans_fit.inertia_
```

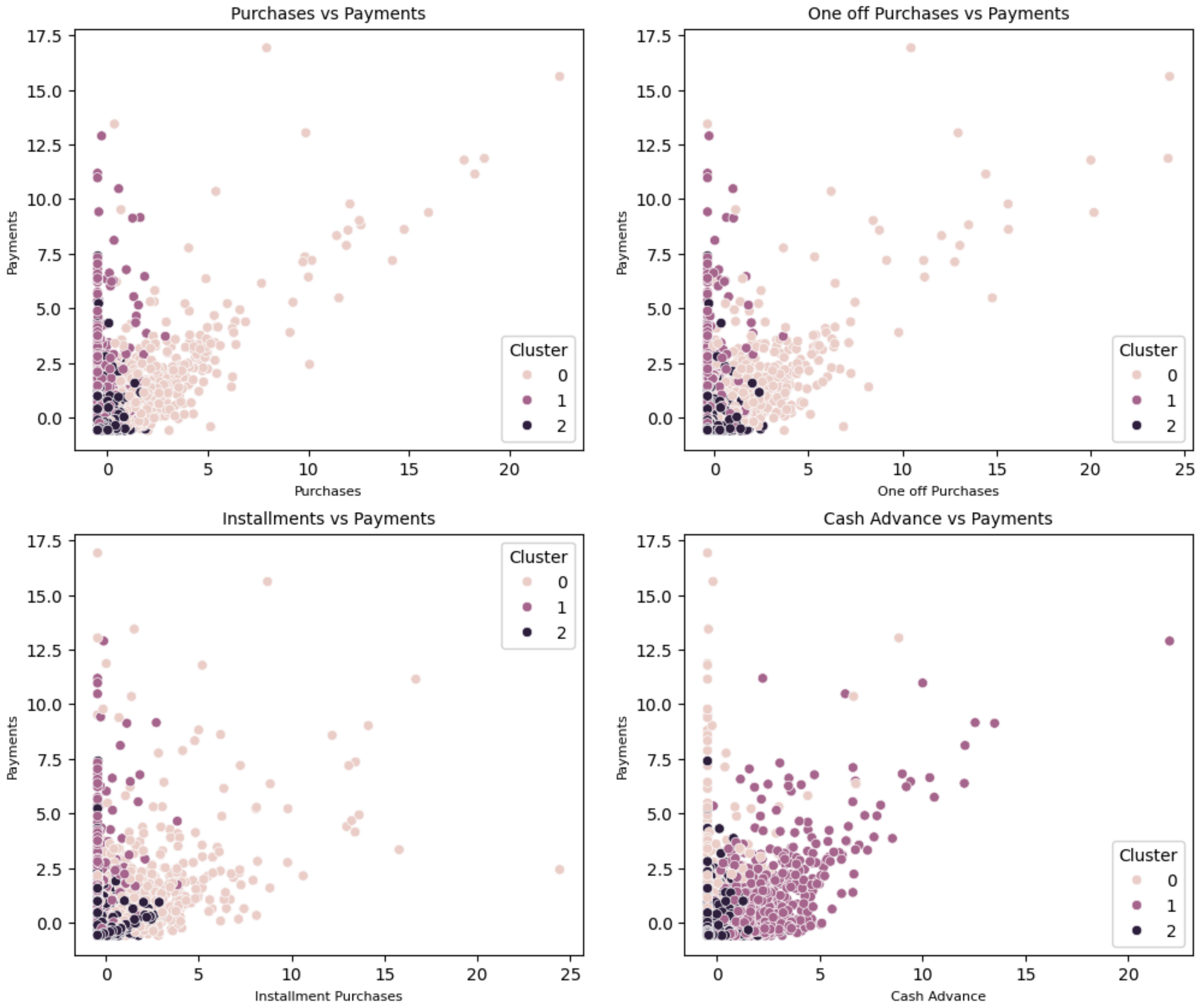
```
111975.0435932569
```

```
# adding a new column to the original dataset for label
df_scaled['Cluster'] = kmeans.labels_
df_scaled.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCH
0	-0.731989	-0.249434	-0.424900	-0.356934	-0.349079	-0.466786	-0.806490	
1	0.786961	0.134325	-0.469552	-0.356934	-0.454576	2.605605	-1.221758	
2	0.447135	0.518084	-0.107668	0.108889	-0.454576	-0.466786	1.269843	
3	0.049099	-1.016953	0.232058	0.546189	-0.454576	-0.368653	-1.014125	
4	-0.358775	0.518084	-0.462063	-0.347294	-0.454576	-0.466786	-1.014125	

Next steps: Generate code with df_scaled | View recommended plots | New interactive sheet

```
# plotting final graph of results

plt.figure(figsize=(12, 10))

plt.subplot(2,2,1)
sns.scatterplot(x=df_scaled['PURCHASES'], y=df_scaled['PAYMENTS'], hue = df_scaled['Cluster'])
plt.xlabel('Purchases', fontsize=8)
plt.ylabel('Payments', fontsize=8)
plt.title('Purchases vs Payments', fontsize=10)

plt.subplot(2,2,2)
sns.scatterplot(x=df_scaled['ONEOFF_PURCHASES'], y=df_scaled['PAYMENTS'], hue = df_scaled['Cluster'])
plt.xlabel('One off Purchases', fontsize=8)
plt.ylabel('Payments', fontsize=8)
plt.title('One off Purchases vs Payments', fontsize=10)

plt.subplot(2,2,3)
sns.scatterplot(x=df_scaled['INSTALLMENTS_PURCHASES'], y=df_scaled['PAYMENTS'], hue = df_scaled['Cluster'])
plt.xlabel('Installment Purchases', fontsize=8)
plt.ylabel('Payments', fontsize=8)
plt.title('Installments vs Payments', fontsize=10)

plt.subplot(2,2,4)
sns.scatterplot(x=df_scaled['CASH_ADVANCE'], y=df_scaled['PAYMENTS'], hue = df_scaled['Cluster'])
plt.xlabel('Cash Advance', fontsize=8)
plt.ylabel('Payments', fontsize=8)
plt.title('Cash Advance vs Payments', fontsize=10)

plt.show()
```



Start coding or generate with AI.