



PROJECT

CREDIT CARD CLUSTERING

BY MANOGNA MARNENI

Problem statement: Create a clustering model to credit card categorize the dataset into appropriate clusters.

Context: This case requires developing a customer segmentation to define a marketing strategy. The sample Dataset summarises the usage behavior of about 9000 active credit cardholders during the last 6 months. The file is at a customer level with 18 behavioral variables.

Data Understanding

The Data

The credit card data has 18 attributes for each customer which include :

Column Description:

- **CUSTID:** Identification of Credit Cardholder (Categorical)
- **BALANCE:** Balance amount left in their account to make purchases
- **BALANCE FREQUENCY:** 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
- **ONE OFF PURCHASES :** Maximum purchase amount done in one-go
- **INSTALMENT PURCHASES :** Amount of purchase done in instalment
- **CASH ADVANCE :** Cash in advance given by the user
- **PURCHASES FREQUENCY :** How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
- **ONE OFF PURCHASE FREQUENCY :** How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
- **PURCHASE INSTALLMENTS FREQUENCY :** How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)

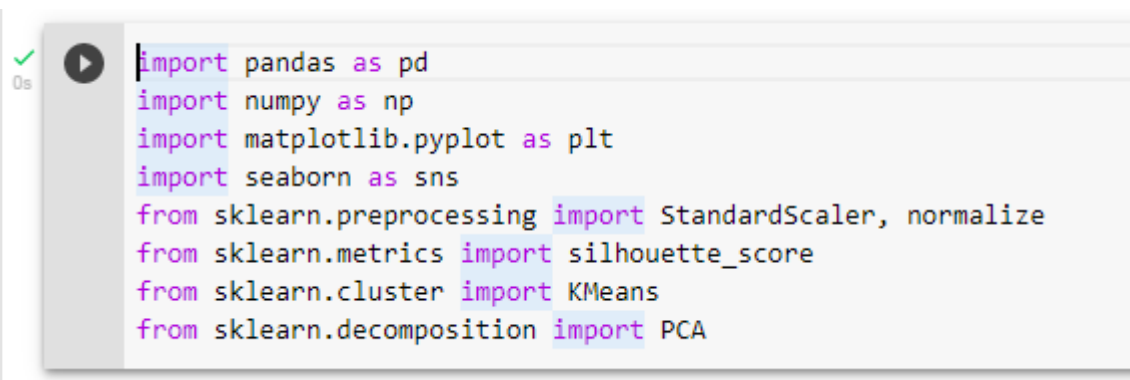
- **CASH ADVANCE FREQUENCY** : How frequently the cash in advance being paid
- **CASH ADVANCE TRX** : Number of Transactions made with "Cash in Advance"
- **PURCHASE TRX** : Number of purchase transactions made
- **CREDIT LIMIT** : Limit of Credit Card for user
- **PAYMENTS** : Amount of Payment done by user
- **MINIMUM_PAYMENTS** : Minimum amount of payments made by user
- **PRC FULL PAYMENT** : Percent of full payment paid by user
- **TENURE** : Tenure of credit card service for user

Pre-Processing Data

This step includes importing needed packages and dataset, checking data summary, handling missing values, checking data types, and selecting the features.

Import library and dataset

Numpy, Pandas and SciKit Learn are some of the inbuilt libraries in Python.

A screenshot of a Jupyter Notebook cell. On the left, there is a green checkmark icon and a play button icon. The code is as follows:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

A little glance at the data :

```
Out[3]:
```

	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	
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	

```
Out[3]:
```

	INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
	0.083333	0.000000	0	2	1000.0	201.802084	139.509787	0.000000	12
	0.000000	0.250000	4	0	7000.0	4103.032597	1072.340217	0.222222	12
	0.000000	0.000000	0	12	7500.0	622.066742	627.284787	0.000000	12
	0.000000	0.083333	1	1	7500.0	0.000000	NaN	0.000000	12
	0.000000	0.000000	0	1	1200.0	678.334763	244.791237	0.000000	12

This is a large data set with 9000 samples. Since the data frame has 18 features with 9000 samples belonging to either of the 18 target classes, our matrix will be:

```
In [5]: df_cc.shape
```

```
Out[5]: (8950, 18)
```

Now going into the mathematics of the dataset, let us find out the standard deviation, mean, minimum value and the four quartile percentile of the data.

```
df_cc.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY
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	0.216667
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371	0.216667
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	0.000000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667	0.333333
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000	1.000000

There are many outliers (look at the max value), but I didn't drop them because they may contain important information, so I treated the outliers as extreme values.

Checking missing values :

```
: df_cc.isnull().sum()
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
```

CREDIT_LIMIT and MINIMUM_PAYMENT content some null values. We will handle the missing values by replacing them by means.

After handling missing values:

```
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
```

There's no more null values.

. Data Normalisation

Normalisation is a technique often applied as part of data preparation for machine learning. The goal of normalisation is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalisation. It is required only when features have different ranges.

```
# Normalizing the Data  
normalized_df = normalize(scaled_df)  
  
# Converting the numpy array into a pandas DataFrame  
normalized_df = pd.DataFrame(normalized_df)
```

✖ Rectangular Snip

. Dimension Reduction using PCA

We apply Principal Component Analysis (PCA) to transform data to 2 dimensions for visualisation because we won't be able to visualise the data in 17 dimensions. PCA transforms a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data.

Applying PCA:

```
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(normalized_df)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
X_principal.head(2)
```

	P1	P2
0	-0.489825	-0.679678
1	-0.518791	0.545011

• Clustering

Clustering is one of the most common exploratory data analysis techniques used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different.

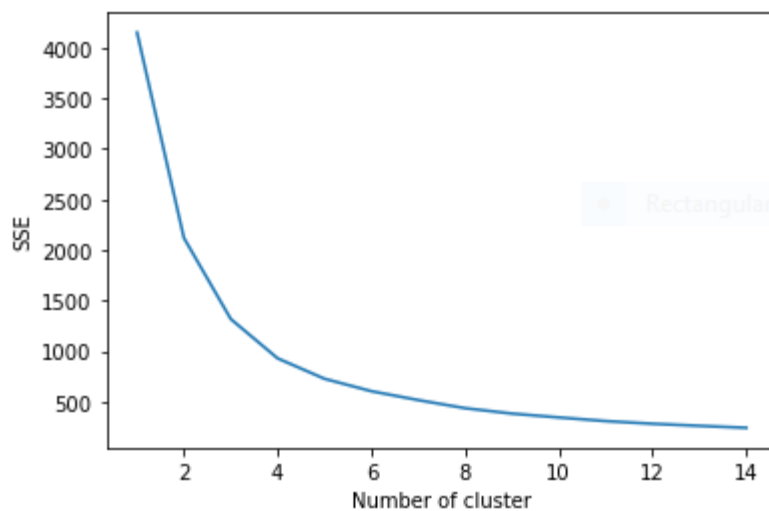
Applying the algorithm

Here I used the K-means algorithm. K-means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**.

To do this, we must first specify the number of clusters K . Here I used the elbow method to specify the best K . Elbow is a very simple method that gives

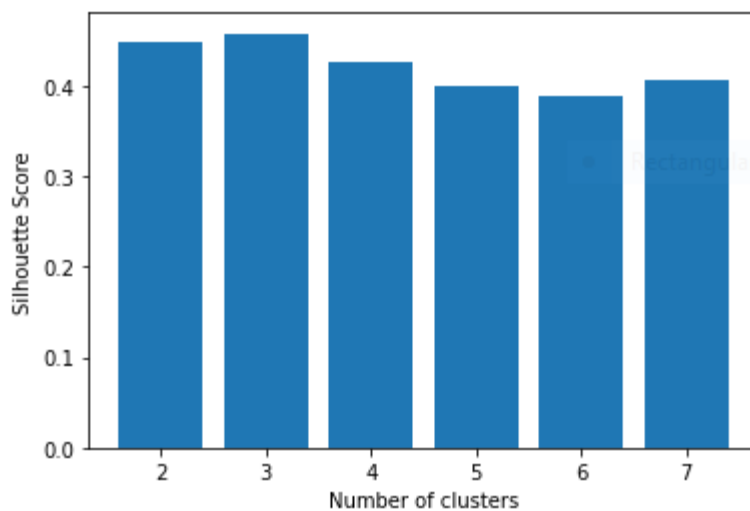
us plots like elbow shape. And we can easily guess the optimal number of K from the plot.

```
sse = {}  
for k in range(1, 15):  
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(X_principal)  
    sse[k] = kmeans.inertia_ # Inertia: Sum of distances of samples to their closest cluster center  
plt.figure()  
plt.plot(list(sse.keys()), list(sse.values()))  
plt.xlabel("Number of cluster")  
plt.ylabel("SSE")  
plt.show()
```



Specifying K using the elbow method.

However, it was hard to find the elbow point of the curve, so I decided to use **silhouette score**. The silhouette method can calculate the silhouette coefficient and easily find the exact number of K .



Specifying K using the silhouette score.

Note : Highest silhouette score is **k = 3**.

And the highest silhouette score is in **k = 3**. It means the best number of clusters, in this case, is 3. Then, I assigned 3 to the KMeans model.

Interpretation of Cluster

After getting the clusters, I interpreted them in the visualisation using

FacetGrid.

```
kmean= KMeans(3)
kmean.fit(scaled_df)
labels=kmean.labels_
clusters=pd.concat([df_cc, pd.DataFrame({'cluster':labels})], axis=1)
```



Clusters visualization using FacetGrid.

And here's the simple results :

Cluster 0

```
Balance           : low
Balance Frequency  : high (updated frequently)
Purchase          : low
Purchase Frequency : low
Cash Advance       : low
Minimum Payment    : low
Credit Limit      : low
```

Rectangular Snip

Cluster 1

```
Balance           : medium
Balance Frequency  : high (updated frequently)
Purchase          : high
Purchase Frequency : high (updated frequently)
Cash Advance       : low
Minimum Payment    : high
Credit Limit      : high
```

Cluster 2

```
Balance           : high
Balance Frequency  : high (updated frequently)
Purchase          : low
Purchase Frequency : very low (does not updated frequently)
Cash Advance       : high
Minimum Payment    : high
Credit Limit      : high
```

Clusters' simple interpretation.

• Visualisation

Finally, I visualised the clusters in a scatter plot.



- **Cluster 0 :** This customer group indicates a large group of customers who have low balances, small spenders (low purchase) with the lowest credit limit.
- **Cluster 1 :** This customer group indicates a small group of customers who have medium balances, spenders (high purchase) with the highest credit limit.
- **Cluster 2 :** This customer group indicates a small group of customers who have high balances and cash advances, low purchase frequency with high credit limit. We can assume that this customer segment uses their credit cards as a loan.

Conclusion

In a couple of minutes we were able to build a clustering model that was able to segment our credit card users into distinctive groups. Some of these were fairly classical such as the prime segment, revolvers and transactors but we were also able to identify inactive users. Understanding the behaviour of customers at this level of granularity is key to tailoring offers which improve customer retention and drive revenues.