



## Atelier 3: CLUSTERING

Réalisée par :  
El ktibi El hassane  
Imane Chentouf

Encadrée par :  
Pr . Aachak Lotfi

---

## **Objectif:**

l'objectif principal de cet atelier est de pratiquer les concepts du clustering, en traitant les données d'une Data Sets.

## **Outils :**

### **IDE :**

Jupyter Notebook

### **Language :**

Python

### **Libraries :**

Pandas, Sklearn, matplotlib.pyplot, Seaborn.

### **Data Sets :**

#### **Credit Card:**

Credit Card Data Set: <https://www.kaggle.com/arjunbhasin2013/ccdata>

---

## Partie 1: Data Visualisation:

### Introduction:

À propos de cet ensemble de données

Cette affaire nécessite de développer une segmentation de la clientèle pour définir la stratégie de marketing. L'échantillon de données résume le comportement d'utilisation d'environ 9000 détenteurs de cartes de crédit actifs au cours des 6 derniers mois. Le fichier se situe au niveau du client avec 18 variables comportementales.

Le dictionnaire de données pour les cartes de crédit est le suivant :-

### Ensemble de données

L'ensemble de données comprend les données de 8950 clients avec 17 caractéristiques, en particulier :

- CUSTID : Identification du titulaire de la carte de crédit (Catégorique)
- BALANCE : Solde du montant laissé sur leur compte pour effectuer des achats (
- BALANCE : Fréquence de mise à jour du solde, score entre 0 et 1 (1 = mise à jour fréquente, 0 = pas de mise à jour fréquente)
- ACHATS : Montant des achats effectués sur compte
- ONEOFFPURCHASES : Montant maximum d'achat effectué en une fois
- ACHATS À TEMPÉRAMENT : Montant de l'achat effectué en plusieurs fois
- CASHADVANCE : avance de fonds donnée par l'utilisateur
- FRÉQUENCE DES ACHATS : Fréquence des achats, note entre 0 et 1 (1 = achats fréquents, 0 = achats peu fréquents)
- FRÉQUENCE DES ACHATS : Fréquence des achats en une fois (1 = achats fréquents, 0 = achats peu fréquents)
- FRÉQUENCE DES ACHATS ET DES INSTALLATIONS : Fréquence des achats échelonnés (1 = fréquents, 0 = peu fréquents)
- FRÉQUENCE DE L'ARGENT : Fréquence de paiement de l'avance en espèces
- CASHADVANCETRX : Nombre d'opérations effectuées avec "Cash in Advanced".
- PURCHASESTRX : Nombre de transactions d'achat effectuées
- CREDITLIMIT : Limite de la carte de crédit pour l'utilisateur
- PAIEMENTS : Montant du paiement effectué par l'utilisateur
- PAIEMENTS\_MINIMAUX : Montant minimum des paiements effectués par l'utilisateur
- PRCFULLPAYMENT : Pourcentage du paiement intégral payé par l'utilisateur
- TENURE : Durée du service de carte de crédit pour l'utilisateur

---

## Importation de librairies:

```
Entrée [1]: import numpy as np # linear algebra
import pandas as pd # data processing
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

Après avoir importer les bibliothèques nécessaires pour l'étude de notre dataset ;

On déclare le chemin où se trouve notre fichier csv et on le charge à l'aide de la bibliothèque pandas par "pd.read\_csv".

```
Entrée [2]: cc = pd.read_csv('C:\\Users\\Hp\\Desktop\\ML\\CC GENERAL.csv',
                             index_col='CUST_ID')
```

L'utilité de shape c'est d'avoir une idée sur la taille de notre data set.

```
Entrée [3]: cc.shape
```

```
Out[3]: (8950, 17)
```

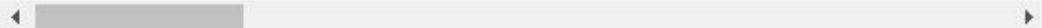
Cela veut dire que notre dataset contient 8950 exemple et 17 caractéristique.

(8950 de lignes et 17 colonnes).

```
Entrée [4]: cc.head()
```

```
Out[4]:
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
CUST_ID				
C10001	40.900749	0.818182	95.40	0.00
C10002	3202.467416	0.909091	0.00	0.00
C10003	2495.148862	1.000000	773.17	773.17
C10004	1666.670542	0.636364	1499.00	1499.00
C10005	817.714335	1.000000	16.00	16.00



---

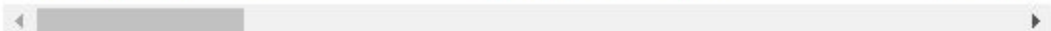
La fonction " describe() " affiche :

- Les nombre de ligne Les moyennes des deux colonnes, l'écart-type, les quartiles, la valeur minimum et la valeur maximum de chaque colonne.
- 

Entrée [5]: `cc.describe()`

Out[5]:

	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



Notre dataset est très grandes donc nous devons vérifier s'il y a des données null dans certaines colonnes;

Entrée [6]: `cc.isnull().sum()`

```
Out[6]: 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
```

Nous remarquons qu'il y a 313 données dans la colonne "MINIMUM\_PAYMENTS" qui manquent.

Donc la Data set contient quelques données nulls.

En les traitant en substituant par des moyens

```
Entrée [7]: cc['MINIMUM_PAYMENTS'].fillna(cc['MINIMUM_PAYMENTS'].mean(),
                                           inplace=True)
            cc['CREDIT_LIMIT'].fillna(cc['CREDIT_LIMIT'].mean(),inplace=True)
```

```
Entrée [8]: cc.isnull().sum()
```

isnull().sum() remplace les valeurs vides par "0".

---

```
Out[8]: 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
```

## Scaling the data

Nous mettons les données à l'échelle parce que cela permet de normaliser les données dans une fourchette particulière et chaque caractéristique se transforme en échelle commune.

Scipy est une **bibliothèque pour les calculs techniques et scientifiques**. Elle contient par exemple des modules pour l'optimisation, l'[algèbre linéaire](#), les [statistiques](#), le [traitement du signal](#) ou encore le [traitement d'images](#).

Il offre également des possibilités avancées de visualisation grâce au module [matplotlib](#).

Afin d'obtenir d'excellentes performances d'exécution.

```
Entrée [10]: from scipy.stats import zscore
```

**Z score**: exprime l'écart par rapport à la valeur moyenne, en déviation standard.

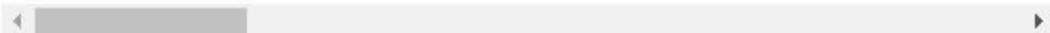


---

```
Entrée [11]: data_scaled=cc.apply(zscore)
              data_scaled.head()
```

Out[11]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
CUST_ID				
C10001	-0.731989	-0.249434	-0.424900	-0.356934
C10002	0.786961	0.134325	-0.469552	-0.356934
C10003	0.447135	0.518084	-0.107668	0.108889
C10004	0.049099	-1.016953	0.232058	0.546189
C10005	-0.358775	0.518084	-0.462063	-0.347294



## Normalisation:

Pour mieux visualiser la data :

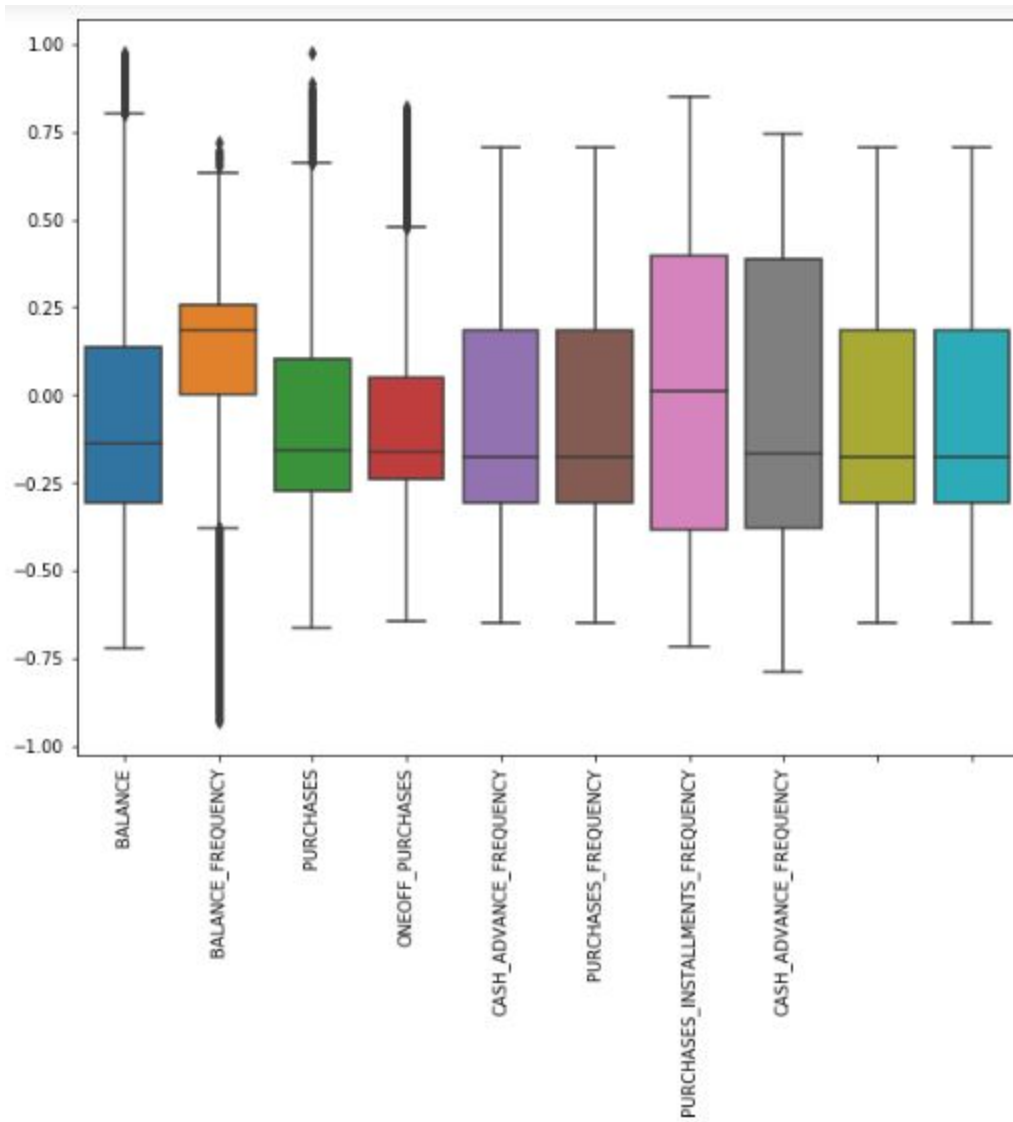
```
Entrée [12]: from sklearn import preprocessing
              from sklearn.preprocessing import StandardScaler, normalize
              data_scaled = normalize(data_scaled)

              data_scaled = pd.DataFrame(data_scaled)
```

```
Entrée [13]: plt.figure(figsize=(8,5))
              sns.boxplot(data=data_scaled)
              plt.xticks(rotation=90)
```

On affiche le diagramme en boîte:





---

## Clustering:

```
Entrée [14]: cluster_range = range(1,15)
cluster_errors=[]
for i in cluster_range:
    clusters=KMeans(i)
    clusters.fit(data_scaled)
    labels=clusters.labels_
    centroids=clusters.cluster_centers_,3
    cluster_errors.append(clusters.inertia_)
clusters_df=pd.DataFrame({'num_clusters':cluster_range,
                           'cluster_errors':cluster_errors})
clusters_df
```

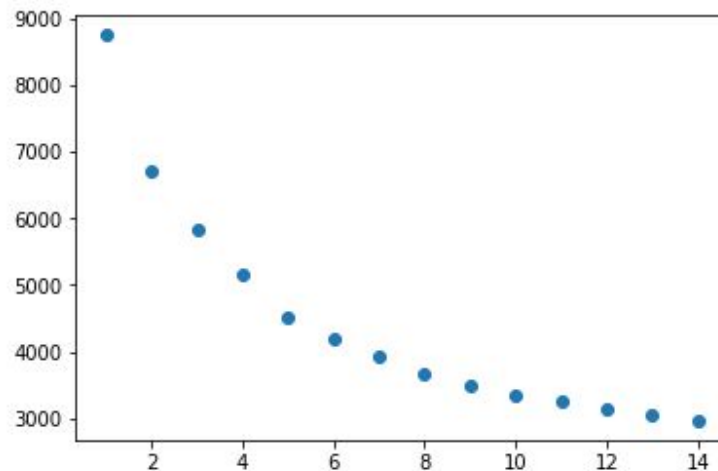
Out[14]:

	num_clusters	cluster_errors
0	1	8754.374981
1	2	6712.707851
2	3	5827.126700
3	4	5145.750341
4	5	4521.798339
5	6	4176.839651
6	7	3914.479117
7	8	3675.558803
8	9	3495.404470
9	10	3348.022342
10	11	3239.511976
11	12	3144.251697
12	13	3052.664453
13	14	2962.522138

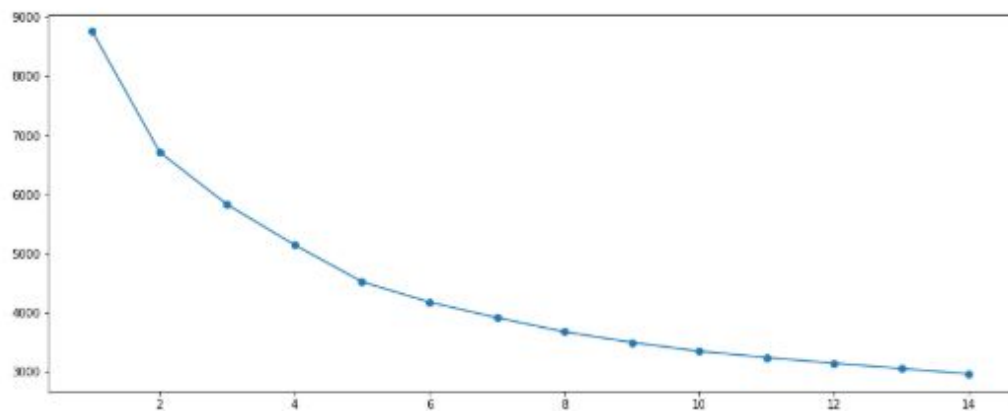
---

```
Entrée [15]: plt.scatter(cluster_range,cluster_errors)
```

```
Out[15]: <matplotlib.collections.PathCollection at 0x24ccee0ff48>
```



```
Entrée [16]: f,ax=plt.subplots(figsize=(15,6))  
plt.plot(clusters_df.num_clusters, clusters_df.cluster_errors,  
         marker='o')  
plt.show()
```



```
Entrée [15]: kmean= KMeans(4)  
kmean.fit(data_scaled)  
labels=kmean.labels_
```

---

## Application de l'APC:

Nous appliquons l'ACP pour transformer les données en 2 dimensions pour la visualisation. Nous ne pourrions pas visualiser les données en 17 dimensions, donc nous réduirons les dimensions avec l'ACP.

L'ACP transforme un grand ensemble de variables en un ensemble plus petit qui contient encore la plupart des informations du grand ensemble. Réduire le nombre de variables d'une donnée.

```
Entrée [17]: pca = PCA(n_components=2)
principalComponents = pca.fit_transform(data_scaled)
principalDf = pd.DataFrame(data = principalComponents
                           , columns = ['BALANCE',
                                       'PURCHASE'])

print(pca.components_)
principalDf.head(2)
```

```
[[ 0.10107779  0.12004337  0.4116414  0.34640697  0.33681405
-0.02334334
  0.32149272  0.2946331  0.27470976 -0.08913898 -0.04865406
  0.39069575
  0.21166275  0.26624912  0.06035121  0.13211246  0.08102049]
 [ 0.40395692  0.13053713  0.041038  0.06287192 -0.01853257
  0.43924159
 -0.19078015 -0.01792374 -0.17860093  0.43408217  0.4197341
 -0.01939452
  0.23867813  0.25747596  0.16696959 -0.19002567 -0.0043264
  9]]
```

Out[17]:

	BALANCE	PURCHASE
0	-1.682221	-1.076450
1	-1.138295	2.506475

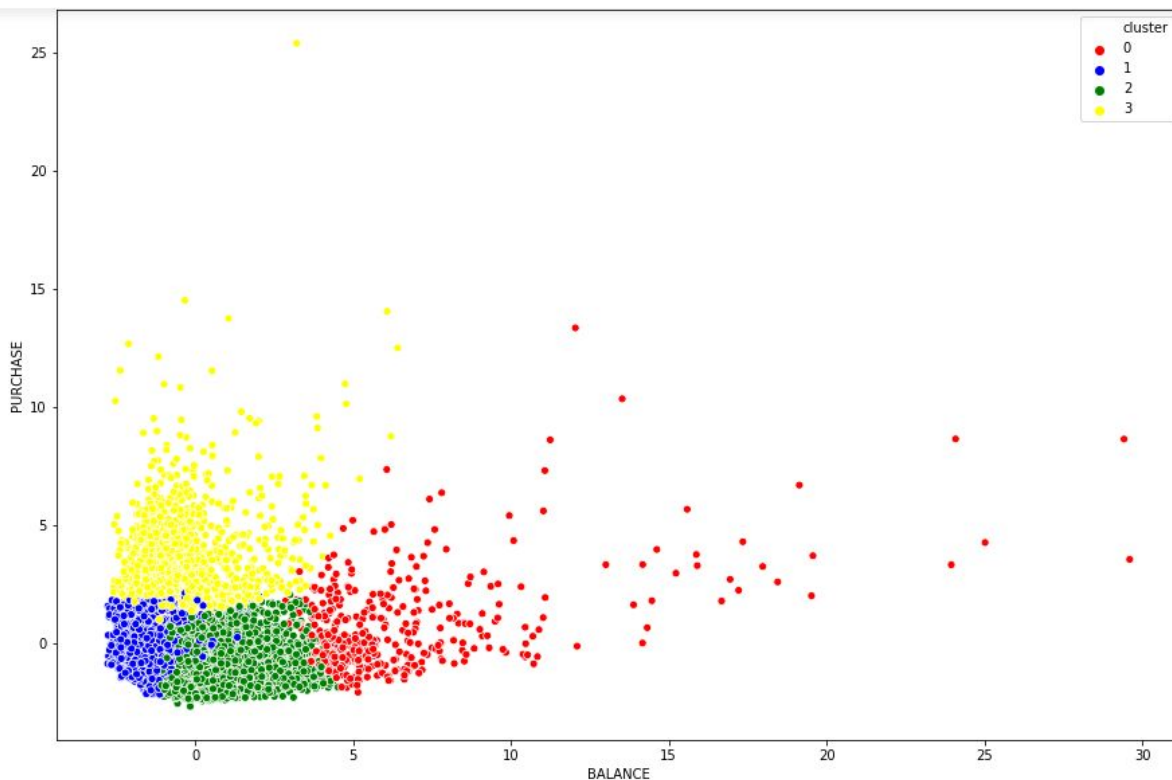
---

```
Entrée [18]: finalDf = pd.concat([principalDf
                                , pd.DataFrame({'cluster':labels})]
                                , axis = 1)
finalDf.head()
```

```
Out[18]:
```

	BALANCE	PURCHASE	cluster
0	-1.682222	-1.076452	1
1	-1.138300	2.506494	3
2	0.969688	-0.383522	2
3	-0.873632	0.043162	1
4	-1.599434	-0.688581	1

```
Entrée [19]: plt.figure(figsize=(15,10))
ax = sns.scatterplot(x="BALANCE"
                    , y="PURCHASE", hue="cluster"
                    , data=finalDf
                    ,palette=['red','blue','green','yellow'])
plt.show()
```



## Interpréter les résultats:

### Cluster 0 :

Solde faible mais le solde est mis à jour fréquemment, c'est-à-dire plus de nombre de transactions. Le nombre d'achats sur le compte est également assez important.

### Cluster 1 :

Solde comparativement élevé, mais le solde n'est pas mis à jour fréquemment, c'est-à-dire moins de transactions. Le nombre d'achats sur le compte est assez faible et les achats en une fois ou par versements sont très faibles.

---

**Cluster 2 :**

L'équilibre est très élevé et il est également mis à jour très fréquemment. Le nombre d'achats est comparativement moins élevé.

**Cluster 3 :**

Le solde est très élevé et il est également mis à jour très fréquemment. Le nombre d'achats est extrêmement élevés.