UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Master 2 - Traitement de l'Information et Data-science en Entreprise (TIDE)

# Analyse en Grande Dimension

Amandine ABENA
Hugo COÏC
Oceane TALEB

Année 2022 - 2023

La collaboration sur ce projet a été très enrichissante et positive. Chacun d'entre nous a apporté ses propres compétences, et ensemble, nous avons travaillé en harmonie pour mener à bout notre projet. Les échanges entre nous ont été fructueux, permettant à chacun d'apprendre de l'autre et de se développer dans le respect et la coopération. Nous avons chacun apporté notre contribution de manière équitable. Ainsi, plus précisément, Amandine s'est occupée de la visualisation et du modèle CAH, Hugo s'est occupé de l'ACP et du modèle K-Means et Oceane s'est occupé du modèle GMM et de la rédaction du rapport.

# Table des matières

# 1 Introduction

L'analyse en grande dimension correspond à l'analyse de de données qui comportent un nombre de variables important par rapport au nombre d'observations. Lorsqu'on se trouve dans ce cadre la, les modèles "classiques" d'analyse échouent systématiquement car ils sont dépassés et pas adaptés. C'est notamment le cas avec nos données. En effet, la base de données continent les informations sur le comportement de près de 9 000 détenteurs actifs de cartes de crédit au cours des 6 derniers mois, au niveau individuel, et contient également 18 variables décrivant leur comportement. Notre objectif est donc de diviser le marché en groupes ayant des comportements similaires à des fins marketing comme par exemple la personnalisation d'offre, ou plus globalement la mise en place de stratégies plus efficaces.

Il était donc question dans ce projet de mettre en oeuvre certaines stratégies que nous avons vu en cours d'analyse en grande dimension afin d'opérer une segmentation de la clientèle.

Afin d'évaluer la performance de chacun de nos algorithmes, nous avons choisi d'associer comme métriques :

— Silhouette score : c'est une métrique qui prend un point, mesure la moyenne des distances entre ce point et ceux de son cluster, puis compare avec la moyenne des distances entre ce point et ceux des autres clusters. Il est compris entre -1 et 1, 1 étant le cluster optimal et -1 le plus mauvais cluster.

— Calinski score : il compare la variance intra-cluster et la variance inter-cluster. Lorsque ce score est élevé, cela veut dire que les clusters sont bien définis et distincts.

# 2 Analyse exploratoire des données et pré-traitements

Nous avons commencé ce travail par une analyse exploratoire des données afin de mieux comprendre celles-ci et détecter d'éventuelles anomalies ou valeurs manquantes à traiter avant de passer aux différents modèles.

## 2.1 Présentation des données et statistiques descriptives

La première partie a consisté en l'importation des données dans un dataframe `df` et l'observation de celles-ci. Notre base de données continent 8950 observations et 18 variables. Nous avons ensuite vérifié la présence de doublons ainsi que de valeurs manquantes, puis nous avons observé les informations générales des données, comme le type des variables.

[6]:
```
#valeurs manquantes
df.isnull().values.any()
df.isna().sum().to_frame()
```

[6]:

| | 0 |
|---|---|
| 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 |

Bien qu'il n'y ait pas de doublons, on a observé 1 valeur manquante pour CREDIT_LIMIT et 313 pour MINIMUM_PAYMENTS. Nous nous sommes alors concentrés sur ces deux variables pour essayer de comprendre la nature de ces données manquantes.

[7]:
```
df[df['CREDIT_LIMIT'].isna()].T
### on aurait pu penser qu'il n'y a pas de limite pour le crédit et qu'il␣
 ↪s'agit d'un client VIP,
### visiblement non il manque juste l'info
```

[7]:

| | 5203 |
|---|---|
| CUST_ID | C15349 |
| BALANCE | 18.400472 |
| BALANCE_FREQUENCY | 0.166667 |

| | |
|---|---|
| PURCHASES | 0.0 |
| ONEOFF_PURCHASES | 0.0 |
| INSTALLMENTS_PURCHASES | 0.0 |
| CASH_ADVANCE | 186.853063 |
| PURCHASES_FREQUENCY | 0.0 |
| ONEOFF_PURCHASES_FREQUENCY | 0.0 |
| PURCHASES_INSTALLMENTS_FREQUENCY | 0.0 |
| CASH_ADVANCE_FREQUENCY | 0.166667 |
| CASH_ADVANCE_TRX | 1 |
| PURCHASES_TRX | 0 |
| CREDIT_LIMIT | NaN |
| PAYMENTS | 9.040017 |
| MINIMUM_PAYMENTS | 14.418723 |
| PRC_FULL_PAYMENT | 0.0 |
| TENURE | 6 |

Pour CREDIT_LIMIT, on pourrait penser, à priori, qu'il s'agit d'un client qui ne possède pas de limite, donc VIP. Mais après avoir observé attentivement ses caractéristiques, nous n'avons pas trouvé d'explication particulière, et avons simplement conclu à une information manquante.

[8]:
```
df[df['MINIMUM_PAYMENTS'].isna()].describe().T
```

[8]:

| | count | mean | std | min |
|---|---|---|---|---|
| BALANCE | 313.0 | 555.441321 | 1292.687887 | 0.0 |
| BALANCE_FREQUENCY | 313.0 | 0.389403 | 0.408341 | 0.0 |
| PURCHASES | 313.0 | 393.087284 | 757.905701 | 0.0 |
| ONEOFF_PURCHASES | 313.0 | 250.433387 | 624.453991 | 0.0 |
| INSTALLMENTS_PURCHASES | 313.0 | 142.653898 | 311.289043 | 0.0 |
| CASH_ADVANCE | 313.0 | 559.136698 | 1185.132567 | 0.0 |
| PURCHASES_FREQUENCY | 313.0 | 0.336043 | 0.372517 | 0.0 |
| ONEOFF_PURCHASES_FREQUENCY | 313.0 | 0.107886 | 0.227370 | 0.0 |
| PURCHASES_INSTALLMENTS_FREQUENCY | 313.0 | 0.244670 | 0.359756 | 0.0 |
| CASH_ADVANCE_FREQUENCY | 313.0 | 0.067169 | 0.130087 | 0.0 |
| CASH_ADVANCE_TRX | 313.0 | 1.460064 | 3.182359 | 0.0 |

| | | | | |
|---|---|---|---|---|
| PURCHASES_TRX | 313.0 | 5.833866 | 9.600908 | 0.0 |
| CREDIT_LIMIT | 313.0 | 3731.789137 | 2924.606153 | 500.0 |
| PAYMENTS | 313.0 | 322.286168 | 1996.658905 | 0.0 |
| MINIMUM_PAYMENTS | 0.0 | NaN | NaN | NaN |
| PRC_FULL_PAYMENT | 313.0 | 0.000000 | 0.000000 | 0.0 |
| TENURE | 313.0 | 11.063898 | 1.869734 | 6.0 |

| | 25% | 50% | 75% | \ |
|---|---|---|---|---|
| BALANCE | 0.187069 | 16.848358 | 286.686616 | |
| BALANCE_FREQUENCY | 0.090909 | 0.181818 | 1.000000 | |
| PURCHASES | 1.400000 | 130.400000 | 399.950000 | |
| ONEOFF_PURCHASES | 0.000000 | 0.000000 | 176.030000 | |
| INSTALLMENTS_PURCHASES | 0.000000 | 0.000000 | 152.280000 | |
| CASH_ADVANCE | 0.000000 | 0.000000 | 480.104401 | |
| PURCHASES_FREQUENCY | 0.083333 | 0.166667 | 0.583333 | |
| ONEOFF_PURCHASES_FREQUENCY | 0.000000 | 0.000000 | 0.083333 | |
| PURCHASES_INSTALLMENTS_FREQUENCY | 0.000000 | 0.000000 | 0.416667 | |
| CASH_ADVANCE_FREQUENCY | 0.000000 | 0.000000 | 0.083333 | |
| CASH_ADVANCE_TRX | 0.000000 | 0.000000 | 1.000000 | |
| PURCHASES_TRX | 1.000000 | 2.000000 | 8.000000 | |
| CREDIT_LIMIT | 1500.000000 | 3000.000000 | 5000.000000 | |
| PAYMENTS | 0.000000 | 0.000000 | 0.000000 | |
| MINIMUM_PAYMENTS | NaN | NaN | NaN | |
| PRC_FULL_PAYMENT | 0.000000 | 0.000000 | 0.000000 | |
| TENURE | 12.000000 | 12.000000 | 12.000000 | |

| | max |
|---|---|
| BALANCE | 9164.724752 |
| BALANCE_FREQUENCY | 1.000000 |
| PURCHASES | 7597.090000 |
| ONEOFF_PURCHASES | 6761.290000 |
| INSTALLMENTS_PURCHASES | 2959.240000 |
| CASH_ADVANCE | 7616.064965 |
| PURCHASES_FREQUENCY | 1.000000 |
| ONEOFF_PURCHASES_FREQUENCY | 1.000000 |

| | |
|---|---|
| PURCHASES_INSTALLMENTS_FREQUENCY | 1.000000 |
| CASH_ADVANCE_FREQUENCY | 1.000000 |
| CASH_ADVANCE_TRX | 21.000000 |
| PURCHASES_TRX | 77.000000 |
| CREDIT_LIMIT | 19500.000000 |
| PAYMENTS | 29272.486070 |
| MINIMUM_PAYMENTS | NaN |
| PRC_FULL_PAYMENT | 0.000000 |
| TENURE | 12.000000 |

Néanmoins pour MINIMUM_PAYMENTS, le nombre de valeurs manquantes est plus important. Nous avons observé une grande variabilité entres ces observations. Il est donc nécessaire de traiter ces valeurs avant de passer à ma modélisation.

## 2.2 Traitements préliminaires

### 2.2.1 Données manquantes

Nous avons ainsi fait le choix de traiter les données manquantes comme suit :

— Suppression de l'observation pour CREDIT_LIMIT.

— Imputation des valeurs manquantes grâce au KNN pour MINIMUM_PAYMENTS.

Nous avons fait le choix du KNN Imputer car il permet de conserver les relations entre les variables. Ce qui, pour notre sujet, est important car il préserve les différences inter-classe. De plus, il est plus précis que la médiane ou la moyenne car les observations avec des données manquantes ont des répartitions très différentes de celles sans données manquantes.

Le KNN Imputer est une méthode de remplacement des données manquantes qui prend un nombre de voisins déterminés en entrée, qui sont les plus proches, et effectue la moyenne sur ces observations et remplace les données manquantes par la moyenne de leurs plus proches voisins.

Avant d'appliquer le KNN Imputer nous avons d'abord standardisé nos données afin d'éviter les éventuelles disparités d'échelles des variables qui pourraient impacter profondément les résultats du KNN Imputer. Ainsi, nous remplaçons les valeurs manquantes de manière plus précise. C'est ce qu'on observe en sortie. En effet, les statistiques descriptives de la

variable MINIMUM_PAYMENTS après imputation, sont très proches de celles après standardisation. Donc le remplacement des valeurs manquantes n'a pas altéré la distribution de la variable, ce qui est positif.

### 2.2.2 Outliers

[11]:
```
c=0
for i,row in df_wo_nan.iterrows():
    if float(df_wo_nan.loc[i,["MINIMUM_PAYMENTS"]])>=float(df_wo_na.
 ↪loc[i,["PAYMENTS"]]) :
        c=c+1
c
```

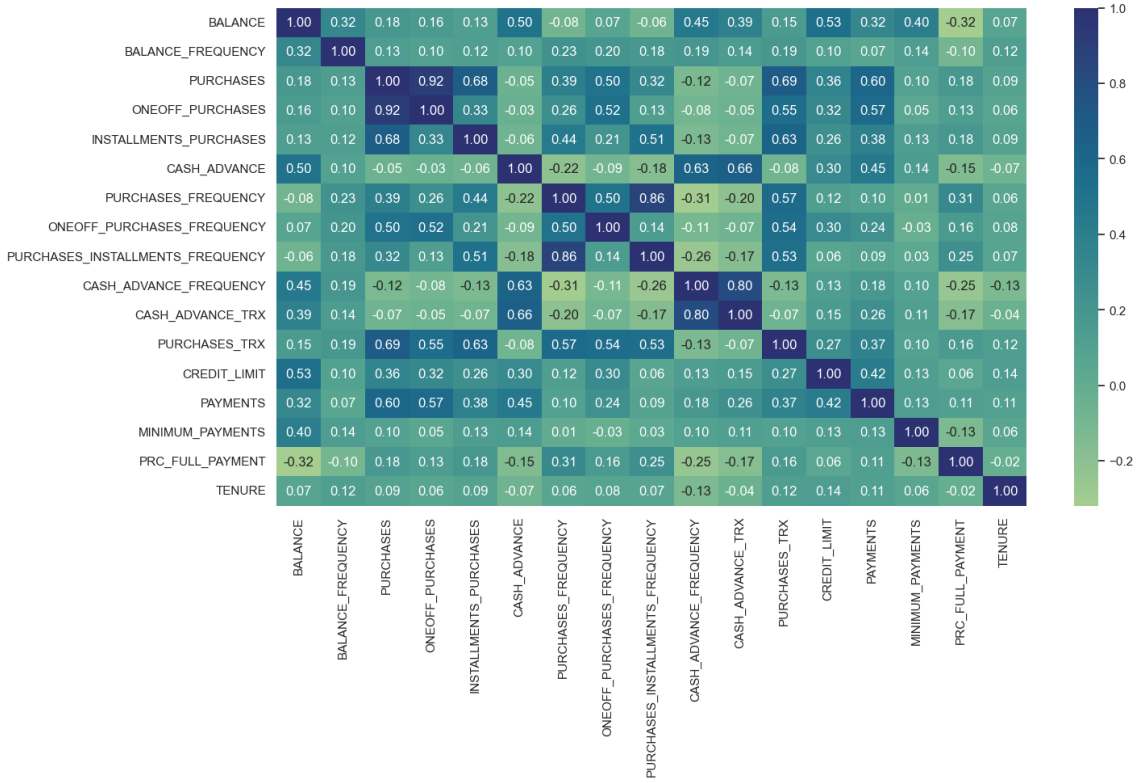
[11]:
```
5929
```

Par la suite, nous nous sommes intéressés aux outliers. Nous avons remarqué une incohérence entres les données. Pour certaines observations, le MINIMUM_PAYMENT est supérieur au PAYMENT. Cependant, ce problème étant étendu sur 5929 observations, soit environ les 2/3 des données, nous avons fait le choix de ne pas traiter ces outliers et d'utiliser les données telles quelles.

## 2.3 Corrélations et distributions des variables

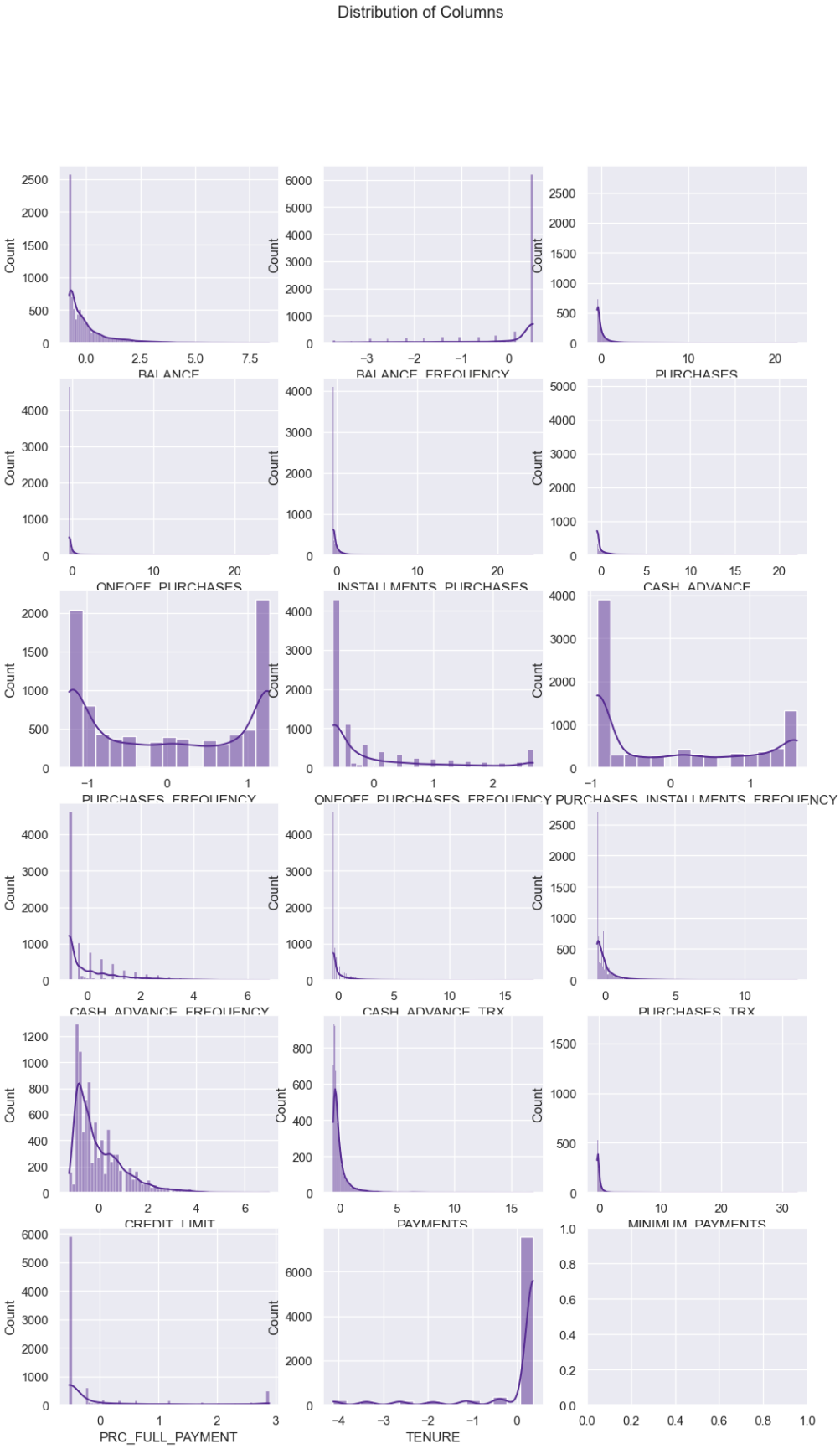
[13]:
```
sns.set(rc = {'figure.figsize':(15,8)})
sns.heatmap(df_wo_na.corr(),annot=True, fmt=".2f", cmap="crest")
```

[13]:
```
<AxesSubplot:>
```




Après avoir appliqué ces traitements, nous avons à nouveau standardisé nos données, puis avons observé les corrélations des différentes variables. On remarque qu'il y a énormément de corrélation entres les différentes variables, ce qui voudrait dire que plusieurs d'entre elles expliqueraient les mêmes profils de client. Il serait donc intéressant d'effectuer une réduction de dimension afin d'améliorer la performance des modèles de clustering en les simplifiant.

[14]:
```
#Distribution des colonnes
sns.set_palette('Purples_r')
fig, ax = plt.subplots(6,3,figsize=(12,20))
for i, col in enumerate(df_wo_na):
    sns.histplot(df_wo_na[col], kde=True, ax=ax[i//3, i%3])
fig.suptitle('Distribution of Columns')
plt.show()
```


Distribution of Columns
Count
BALANCE
BALANCE_FREQUENCY
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
PRC_FULL_PAYMENT
TENURE

# 3 Clustering : K-Means avec ACP

## 3.1 ACP

Suite à l'étude des corrélations et des distributions des variables, nous avons choisi de faire une réduction de dimension grâce à une ACP, afin de simplifier les données en ne conservant que les dimensions les plus importantes et donc d'avoir des résultats plus précis à nos algorithmes de clustering.

L'ACP est un algorithme qui permet de réduire la dimension des données. Il identifie tout d'abord les directions les plus récurrentes des données et projette ensuite celles-ci sur le ou les sous-ensembles qui maximisent leurs variances. Ces sous-ensembles sont appelés composantes principales.

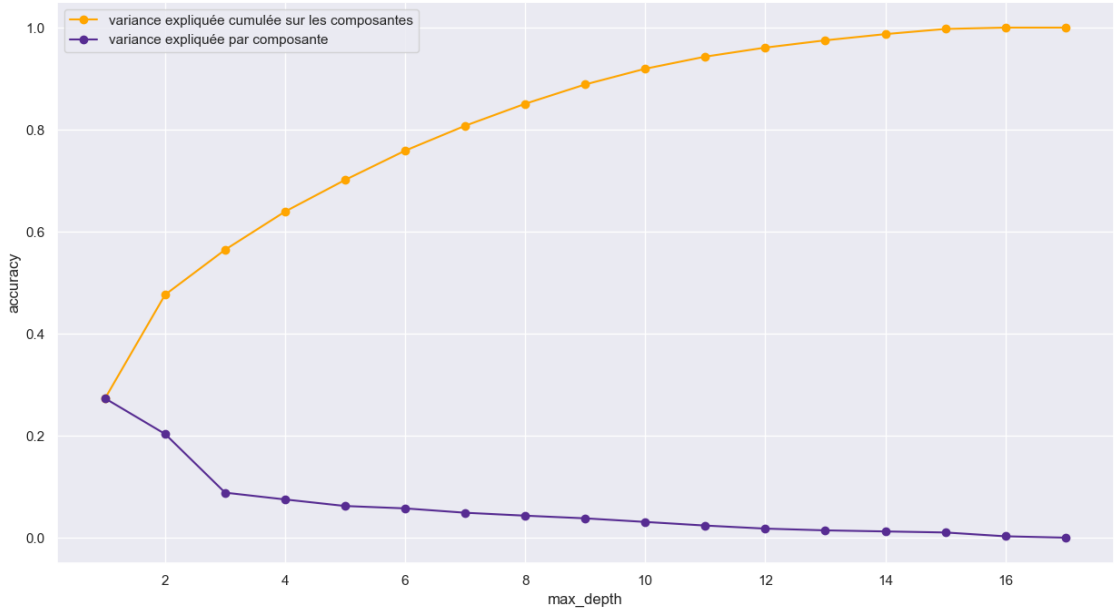
[15]:
```
pca=PCA()
pca=pca.fit(df_wo_na)
```

[16]:
```
pca.explained_variance_ratio_
```

[16]:
```
array([2.73010064e-01, 2.03241866e-01, 8.82218368e-02, 7.48510696e-02,
       6.20469072e-02, 5.73573525e-02, 4.88190145e-02, 4.31003139e-02,
       3.78615418e-02, 3.07994481e-02, 2.37235281e-02, 1.77228258e-02,
       1.42735677e-02, 1.21704661e-02, 1.01289279e-02, 2.67058510e-03,
       6.85227936e-07])
```

[17]:
```
plt.plot(range(1,18),pca.explained_variance_ratio_.cumsum(),␣
 ↪marker='o',label='variance expliquée cumulée sur les composantes'
       ,color='orange' )

plt.plot(range(1,18),pca.explained_variance_ratio_,␣
 ↪marker='o',label='variance expliquée par composante')
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.legend()
plt.grid(True)
```




Nous avons représenté graphiquement la variance expliquée cumulée sur les composantes ainsi que la variance expliquée par composante afin de déterminer le nombre de composantes principales optimal pour obtenir nos données réduites en dimensionnalité.

Le premier critère de sélection sur lequel nous nous appuyons fait référence à une heuristique connue qui indique qu'il faut conserver un minimum de 80% d'explication de la variance pour le choix du nombre de composantes. Selon ce critère, le nombre retenu devrait être 7.

Le second critère repose sur la présence d'un "coude" sur la représentation graphique de la variance expliquée. Ici, on l'observe pour 3 composantes.

Or avec 3 composantes on explique un peu moins de 60% de la variance, ce qui est insuffisant. On choisira donc de conserver 7 composantes pour l'ACP.

## 3.2 K-Means

Pour tenter de répondre à notre problématique de clustering, nous avons essayé en premier lieu l'algorithme des K-Means. L'objectif de ce dernier est de regrouper les données en clusters.

Pour cela, il prend d'abord comme paramètre le nombre de clusters et fixe des centres aléatoires pour chacun d'entre eux. Il assigne ensuite à ces derniers les points pour lesquels la distance avec le centre du cluster est la plus petite. Autrement dit, chaque cluster sera composé des points les plus proches de son centre. Ensuite, les centres sont réajustés et calculés à partir de la moyenne de tous les points du cluster et le processus recommence

jusqu'à ce que les centres des clusters ne bougent plus. Ainsi, il est primordial de bien sélectionner le nombre de clusters pour obtenir les résultats les plus précis possibles.

Nous avons alors cherché le nombre de clusters optimal pour notre premier modèle à partir de nos données réduites.

[18]:
```
pca=PCA(n_components=7)
pca=pca.fit(df_wo_na)
```

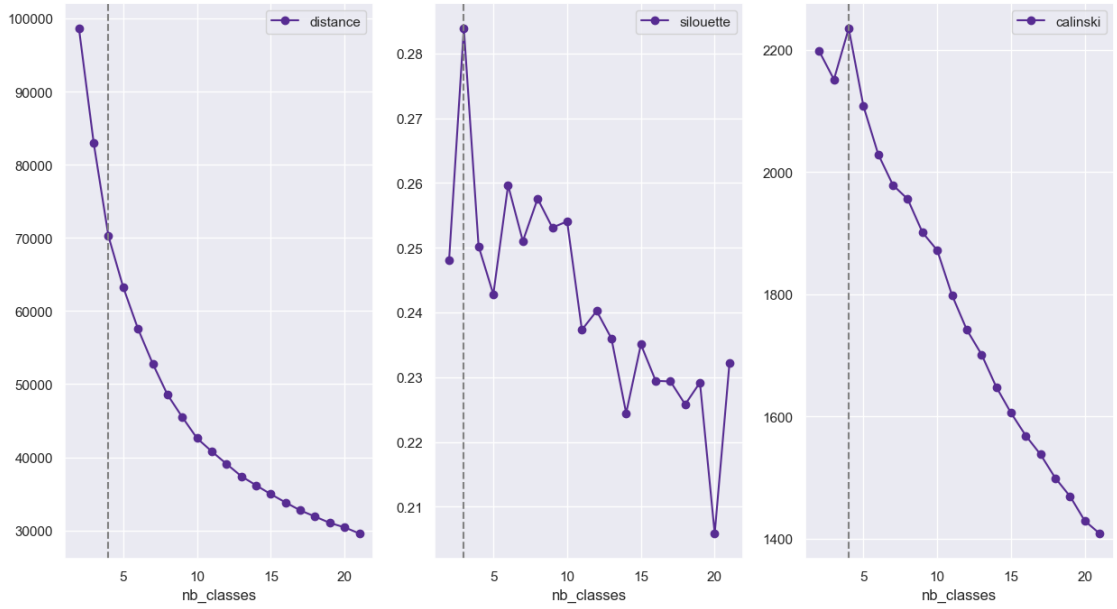
[19]:
```
score_pca=pca.transform(df_wo_na)
```

[20]:
```
distance=[] ### la distance (sum of squared) des points par rapport au␣
 ↪barycentre de la classe (à minimiser)
nb_classes=[]
silouette=[]### Mesure de la distance entre les clusters (à maximiser)
calinski=[] ### rapport entre la dispersion intra et inter-groupes (à␣
 ↪maximiser)
for i in range(20):
    kmeans = KMeans(n_clusters=i+2, random_state=0).fit(score_pca)
    distance.append(kmeans.inertia_)
    predictions=kmeans.predict(score_pca)
    silouette.append(silhouette_score(score_pca, predictions))
    calinski.append(calinski_harabasz_score(score_pca, predictions))
    nb_classes.append(i+2)
```

[21]:
```
plt.subplot(131)
plt.plot(nb_classes,distance,marker='o',label="distance")
plt.axvline(x = 4, color = 'grey',ls='--')
plt.xlabel('nb_classes')
plt.legend()
plt.subplot(132)
plt.plot(nb_classes,silouette,marker='o',label="silouette")
plt.axvline(x = 3, color = 'grey',ls='--')
plt.xlabel('nb_classes')
plt.legend()
plt.subplot(133)
plt.plot(nb_classes,calinski,marker='o',label="calinski")
```

```
plt.axvline(x = 4, color = 'grey',ls='--')
plt.xlabel('nb_classes')
plt.legend()
```

[21] : <matplotlib.legend.Legend at 0x28719a953d0>




Pour cela nous avons considéré et représenté graphiquement 3 indicateurs en fonction du nombre de clusters. La distance, le silhouette score ainsi que le calinski score.

Pour le critère basé sur les distances, nous utilisons la technique du coude pour trouver le nombre de clusters optimal. Ici, on remarque sur le graphique, un coude pour 4 clusters (qui est plus visible sur une image plus grande). Ensuite, pour le silhouette score et le calinski score, on retient le nombre de clusters pour lesquels ces scores sont maximisé. Ainsi, on observe que le silhouette score est maximisé pour 3 clusters et le calinski pour 4 clusters.

Nous avons alors choisi de conserver 4 clusters, car 2 de nos 3 critères nous indiquent que c'est la meilleure option.

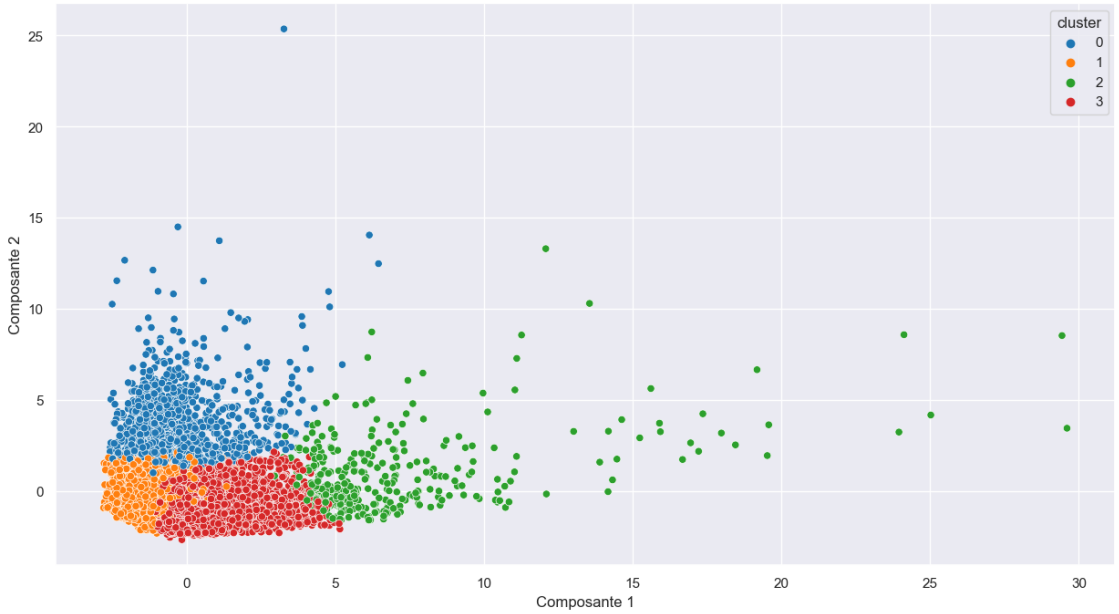
[22] :
```
pca_to_kmeans=KMeans(n_clusters=4,random_state=0)
pca_to_kmeans=pca_to_kmeans.fit(score_pca)
```

[23] :
```
tab_acp=pd.concat([df_wo_na.reset_index(drop=True),pd.
↪DataFrame(score_pca)],axis=1)
```

```
tab_acp.columns.values[-7:]=['Composante 1','Composante 2','Composante␣
 ↪3','Composante 4','Composante 5',
                              'Composante 6','Composante 7']
tab_acp['cluster']=pca_to_kmeans.labels_
```

[24]:
```
sns.scatterplot(x="Composante 1",y="Composante 2",␣
 ↪hue="cluster",palette="tab10",data=tab_acp)
### graphe des clusters en fonctions des deux composantes principales
```

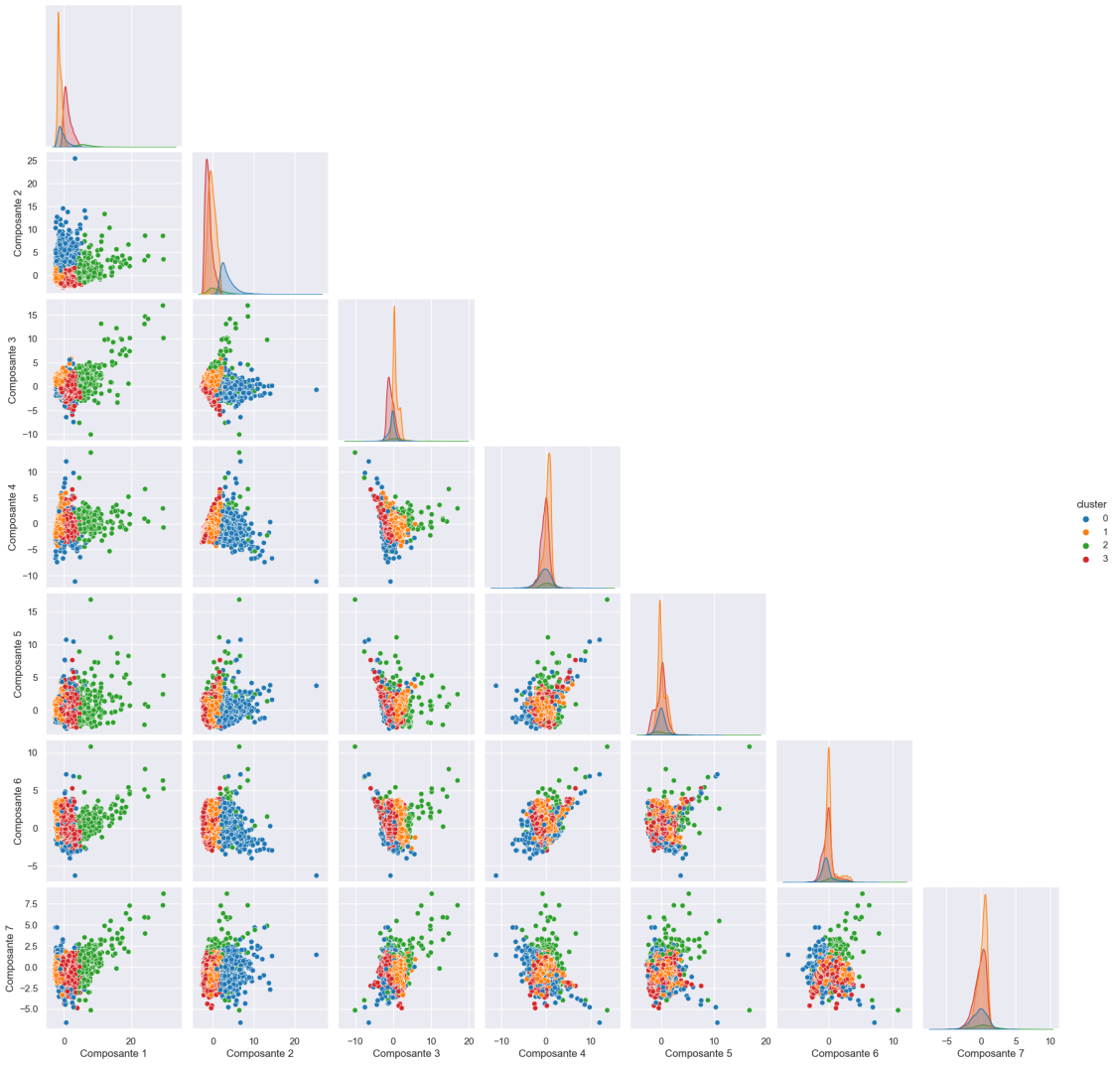
[24]:
```
<AxesSubplot:xlabel='Composante 1', ylabel='Composante 2'>
```




Sur la représentation graphique en 2 dimensions des composantes 1 et 2, on voit nettement les 4 clusters formés. Cependant pour les cluster 0 et 2, respectivement représentés par les couleurs bleu et vert sur le graphique, on observe une dispersion assez importante des observations, comparé aux clusters 1 et 3 (orange et rouge) qui s ont plus compacts. Cela pourrait s'expliquer par le fait que nous n'ayons pas traiter les outliers et que ces observations correspondent aux outliers.

[26]:
```
sns.pairplot(tab_acp.iloc[:,-8:
 ↪],hue="cluster",corner=True,palette="tab10")
## plus gobalement répartitions des classes en fonctions des composantes
```

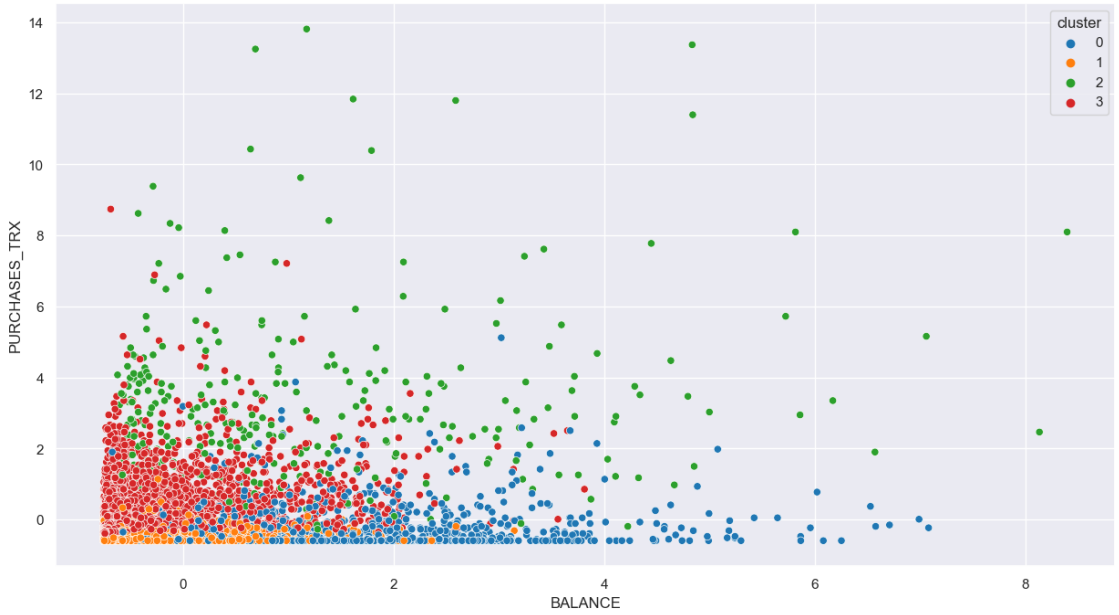
[26]:
```
<seaborn.axisgrid.PairGrid at 0x28719adc970>
```




Nous avons par la suite réalisé un pairplot, qui nous permet d'étudier les relations par paire des composantes de l'ACP. Ce qui est observé en général c'est qu'il y a de bons résultats sur les croisements avec la première composante, mais cela se dégrade à partir de la seconde composante, il est plus compliqué, voire pas possible, de statuer sur la nature des clusters. On voit que le cluster 0 est mieux représenté sur la composante 2 que sur toutes les autres. On remarque aussi sur les courbes de densité, que le cluster 1 a une kurtosis plus élevée que les autres clusters, ce qui voudrait dire qu'il y a une plus grande concentration de points extrêmes dans cette distribution.

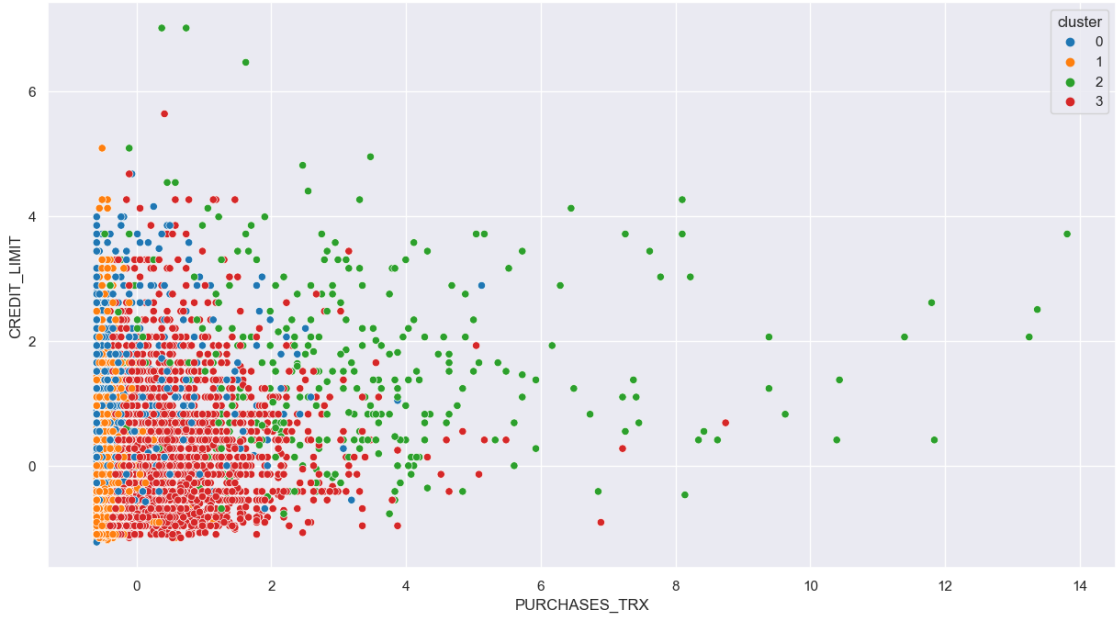
[28]:
```
sns.scatterplot(data=r_, x="BALANCE",y="PURCHASES_TRX", hue="cluster",␣
↪palette="tab10")
```

[28]:
```
<AxesSubplot:xlabel='BALANCE', ylabel='PURCHASES_TRX'>
```




[29]:
```
sns.scatterplot(data=r_, x="PURCHASES_TRX",y="CREDIT_LIMIT",␣
↪hue="cluster", palette="tab10")
```

[29]: <AxesSubplot:xlabel='PURCHASES_TRX', ylabel='CREDIT_LIMIT'>



[30]:
```
tab_acp.cluster.value_counts()
```

[30]:
```
1    4031
3    3365
0    1219
2     335
Name: cluster, dtype: int64
```

Nous avons par la suite étudié les clusters en fonction de certaines variables afin de déterminer le profils des clients de chaque cluster. On peut donc faire les hypothèses suivantes :

- Il semble y avoir deux groupes bien distincts : les comptes avec plus du revenu (clusters 0 et 2) et ceux avec moins de revenus (clusters 1 et 3). En effet on remarque sur les graphiques que les clusters 0 et 2 possèdent des valeurs plus élevées pour les variables BALANCE et CREDIT_LIMIT. De plus, si on associe cela aux fréquences obtenues pour chaque cluster, on pourrait supposer que le cluster 2 est plus riche que le cluster 0, car il y a moins d'individus, les plus riches représentant une petite partie de la population uniquement.
- Au sein du groupe formé par les clusters 0 et 2 on remarque une différence de comportement. Le cluster 0 adopte un "comportement de fourmis" car la variable CASH_ADVANCE est importante. Le cluster 2, à l'inverse, adopte un "comportement de cigale" car il admet un nombre de transactions élevés, observés à travers la variable PURCHASES_TRX.
- En comparant les mêmes variables (CASH_ADVANCE et PURCHASES_TRX) au sein du groupe formé par les clusters 1 et 3, on remarque également une différence. Dans cette configuration, le cluster 1 adopte un "comportement de fourmis" tandis que le cluster 3 a un "comportement de cigale".

# 4 Clustering : CAH avec ACP

Le second algorithme que nous avons utilisé afin d'agréger nos données en clusters est le CAH. Il s'agit d'un algorithme hiérarchique qui détermine les clusters de manière verticale. Au début, il considère chaque point comme un seul cluster. Ensuite, il assemble les points les plus similaires entre eux pour former un nouveau cluster plus grand. Puis il fusionne les clusters qui sont les plus proches en termes de distance entre eux pour former de nouveaux clusters

supérieurs hiérarchiquement. Cette opération est répétée plusieurs fois jusqu'à atteindre le nombre de clusters saisis lors de l'initialisation de l'algorithme.

[31]:
```
from scipy.cluster.hierarchy import linkage, fcluster

# Générer la matrice de liaison avec la distance euclidienne
Z = linkage(score_pca, method='ward', metric='euclidean')

# Définir le nombre de clusters
num_clusters = 4

# Effectuer la classification hiérarchique
clusters = fcluster(Z, num_clusters, criterion='maxclust')
```

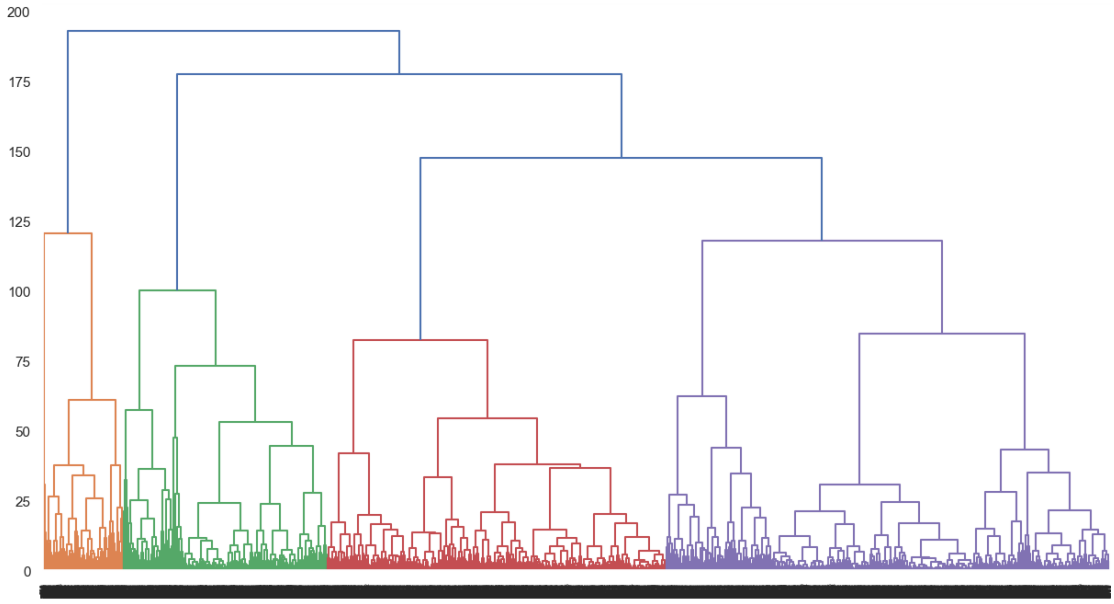
[32]:
```
pd.DataFrame(clusters).value_counts()
```

[32]:
```
4    3722
3    2839
2    1709
1     680
dtype: int64
```

Ici, nous avons fait fonctionner le CAH avec 4 clusters et nous avons obtenu une répartition dans les clusters différente de celle obtenue par les K-Means.

[33]:
```
from scipy.cluster.hierarchy import dendrogram

# Afficher le dendrogramme
dendrogram(Z)
plt.show()
```




[34]:
```
(silhouette_score(score_pca, clusters),
 calinski_harabasz_score(score_pca, clusters))
```

[34]:
```
(0.1746594177333603, 1738.3473167082711)
```

Nous avons représenté ces clusters par un dendrogramme. On observe que l'agrégation des données en 4 groupes a assez bien fonctionné, car les distances entre chaque branche au sein d'un groupe sont plutôt courtes, donc les observations sont proches. Lorsqu'on remonte dans la hiérarchie les distances sont plus espacées, donc le choix de 4 clusters semble optimal. Néanmoins, il est important de noter que les scores obtenus pour cet algorithme sont relativement faibles.

# 5 Clustering : GMM avec ACP

Le dernier algorithme non supervisé de clustering que nous avons utilisé est le GMM (Gaussian Mixture Melange). Le GMM est un algorithme de clustering itératif, qui suppose que les données sont issues de plusieurs distributions gaussiennes et qui estime à chaque itération la probabilité qu'une observation appartienne à une distribution.

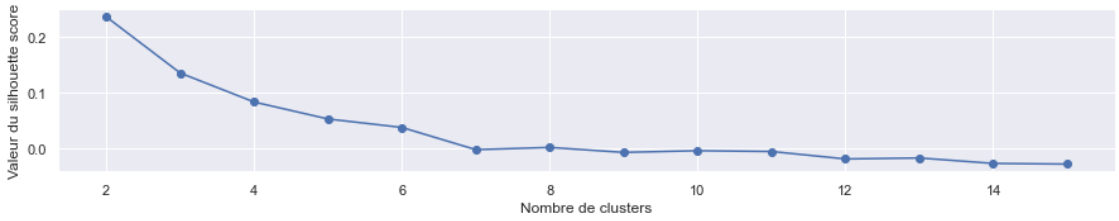
Ce modèle suppose des distributions gaussiennes, or comme nous n'avons pas traité les outliers nous ne savons pas réellement comment se comportent les distributions de nos variables.

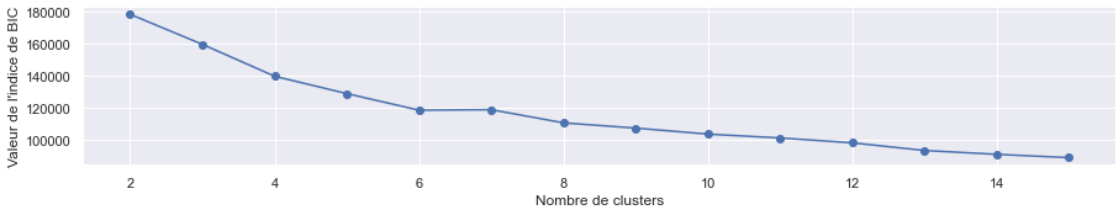
La standardisation effectuée ne nous garanti pas des distributions gaussiennes. Néanmoins, l'algorithme GMM peut faire des prédictions en trouvant des combinaisons de distributions gaussiennes qui se rapprochent de la distribution de nos données. De plus, il est intéressant car fréquemment utilisé pour de la segmentation de client. Ainsi, nous avons utilisé les données réduites par l'ACP afin d'appliquer cet algorithme.

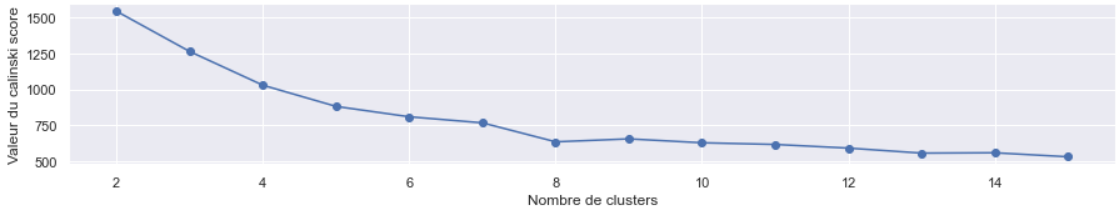
[133]:
```
#Valeur de l'indice de BIC en fonction du nombre de clusters
plt.subplot(3,1,1)
plt.plot(range(2, 16), sil, marker='o')
plt.xlabel("Nombre de clusters")
plt.ylabel("Valeur du silhouette score")
plt.show()

plt.subplot(3,1,2)
plt.plot(range(2, 16), bics, marker='o')
plt.xlabel("Nombre de clusters")
plt.ylabel("Valeur de l'indice de BIC")
plt.show()

plt.subplot(3,1,3)
plt.plot(range(2, 16), cal, marker='o')
plt.xlabel("Nombre de clusters")
plt.ylabel("Valeur du calinski score")
plt.show()
```






Nous avons représenté graphiquement l'évolution du silhouette score, de l'indice de BIC et le calinski score afin de déterminer le nombre de clusters optimal pour cet algorithme. Ainsi, nous devons sélectionner le nombre de clusters qui maximise les score silhouette et calinski et qui minimise l'indice de BIC. Ici , le silhouette ainsi que le calinski sont maximisé pour 2 clusters, mais l'indice du BIC est le plus bas pour 15 clusters. Nous avons choisi de conserver 2 clusters pour l'algorithme GMM.

[145]:
```
gmm = GaussianMixture(n_components=2)
gmm.fit(score_pca)
clusters = gmm.predict(score_pca)
print(silhouette_score(score_pca,␣
 ↪clusters),calinski_harabasz_score(score_pca, clusters))
```

```
0.2363252799744664 1543.2855163282625
```

Ainsi, avec ce paramètre, nous obtenons un silhouette score de 0.2363252799744664 et un calinski score de 1543.2855163282625. Il s'agit la du plus mauvais résultat que nous ayons obtenu. Le GMM ne semble pas correspondre à notre problématique et à nos données. Nous le voyons également dans la répartition dans les classes : la catégorie 0 contient presque le double des observations de la catégorie 1.

[146]:
```
pd.DataFrame(clusters).value_counts()
```

[146]:
```
0    5932
1    3018
dtype: int64
```

# 6 Modèle sélectionné et prédictions

Synthèse des résultats obtenus pour les différents modèles testés :

- Performance des modèles :

| | K-Means | CAH | GMM |
|---|---|---|---|
| Silhouette Score | 0.2502182024179765 | 0.1746594177333603 | 0.2363252799744664 |
| Calinski Score | 2234.8904201017244 | 1738.3473167082714 | 1543.2855163282625 |

- Prédictions des classes :

| Clusters | K-Means | CAH | Clusters |
|---|---|---|---|
| 1 | 4031 | 3722 | 4 |
| 3 | 3365 | 2839 | 3 |
| 0 | 1219 | 1709 | 2 |
| 2 | 335 | 680 | 1 |

Après avoir testé différents algorithmes afin de trouver celui qui répond le mieux à notre problématique de clustering et qui décrit le mieux nos données. Nous avons conclu à l'algorithme des K-Means, qui présente les meilleurs scores et qui est donc le plus performant. Nous avons alors effectué nos prédictions à l'aide de ce dernier sur tous les clients de la base.

[35]:
```
r_.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 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   float64
 11  PURCHASES_TRX                     8950 non-null   float64
 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   float64
 17  cluster                           8950 non-null   int32
dtypes: float64(17), int32(1)
memory usage: 1.2 MB
```

[36]:
```
df=df.reset_index(drop=True)
r_=r_.reset_index(drop=True)
predictions=pd.concat([df["CUST_ID"],r_["cluster"]],axis=1)
```

[37]:
```
predictions
```

[37]:

| | CUST_ID | cluster |
|---|---|---|
| 0 | C10001 | 1 |
| 1 | C10002 | 0 |
| 2 | C10003 | 3 |
| 3 | C10004 | 1 |
| 4 | C10005 | 1 |
| ... | ... | ... |
| 8945 | C19186 | 3 |
| 8946 | C19187 | 3 |
| 8947 | C19188 | 3 |
| 8948 | C19189 | 1 |
| 8949 | C19190 | 1 |

[8950 rows x 2 columns]