PROJET 5

# Segmentation des Clients d'un site E-Commerce

Preparé par: BEDDY EL MOUSTAPHA
Mentor : Thiery silberman
Évaluateur:  Isaac yimgaing

# INTRODUCTION

- Olist souhaite obtenir une segmentation de ses clients utilisable au quotidien par leur équipe marketing dans leurs campagnes de communication. L'objectif est de comprendre les différents types d'utilisateurs grâce à leur comportement et à leurs données personnelles anonymisées.

- Nous utilisée donc des méthodes analytiques et non supervisées pour regrouper ensemble des clients de profils similaires

- Fournir une description actionnable de la segmentation obtenue et proposer une analyse de la stabilité des segments au cours du temps.

# PLAN DE TRAVAILL

1. IMPORTATION DES DONNEES
2. NETTOYAGE DES DONNEES
3. FUSION DES DONNES
4. L’ANALYSE DES VARIABLES
5. SEGMENTATION
6. MAINTENANCE
7. CONCLUSION

# IMPORTATION DES DONNEES

Base de données(olist_customers_dataset.csv) anonymisée du site d'e-commerce **Olist**, téléchargeable (https ://www.kaggle.com/olistbr/brazilian-ecommerce).


```python
#Les clients
customers_df=pd.read_csv("customers.csv")
#Les données de géolocalisation
geolocation_df=pd.read_csv("geolocation.csv")
#Les articles
orders_items_df=pd.read_csv("order_items.csv")
#Les reglements
orders_payment_df=pd.read_csv("order_payments.csv")
#Avis clients
orders_review_df=pd.read_csv("order_reviews.csv")
#Les commandes clients
orders_df=pd.read_csv("orders.csv")
#Categories traduites
products_df=pd.read_csv("products.csv")
#Les produits
category_trans_df=pd.read_csv("product_category.csv")
#Les vendeurs
sellers_df=pd.read_csv("sellers.csv")
```

# NETTOYAGE DES DONNEES

Détection des Valeurs manquantes et doublons

| | datasets | Nom_columns | Nombr_ligne | Nombrl_cols | Nombr_duplic | Nombr_manquent |
|---|---|---|---|---|---|---|
| 0 | customers | customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state | 99441 | 5 | 0 | 0 |
| 1 | geolocation | geolocation_zip_code_prefix, geolocation_lat, geolocation_lng, geolocation_city, geolocation_state | 1000163 | 5 | 261831 | 0 |
| 2 | items | order_id, order_item_id, product_id, seller_id, shipping_limit_date, price, freight_value | 112650 | 7 | 0 | 0 |
| 3 | payments | order_id, payment_sequential, payment_type, payment_installments, payment_value | 103886 | 5 | 0 | 0 |
| 4 | reviews | review_id, order_id, review_score | 99224 | 3 | 0 | 0 |
| 5 | orders | order_id, customer_id, order_status, order_purchase_timestamp, order_approved_at, order_delivered_carrier_date, order_delivered_customer_date, order_estimated_delivery_date | 99441 | 8 | 0 | 4908 |
| 6 | products | product_id, product_category_name, product_weight_g | 32951 | 3 | 0 | 612 |
| 7 | sellers | seller_id, seller_zip_code_prefix, seller_city, seller_state | 3095 | 4 | 0 | 0 |
| 8 | category_translation | product_category_name, product_category_name_english | 71 | 2 | 0 | 0 |

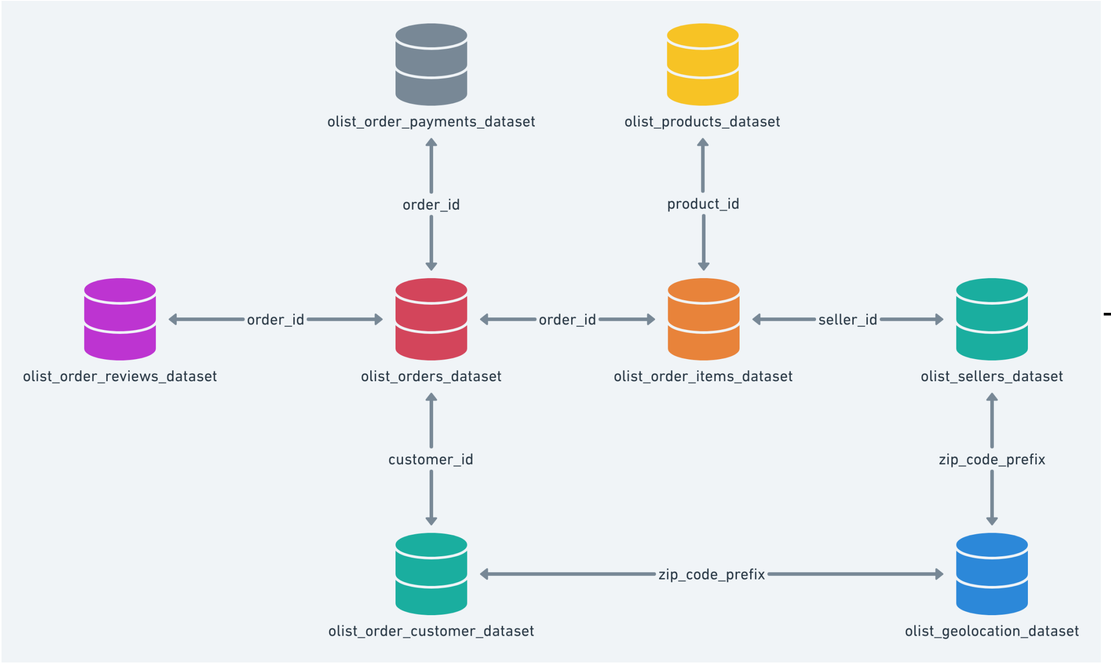
Traitement

- suppression des doublons
- imputation par moyenne
- imputation les colonnes catégorielles
- imputation par les valeurs les plus proches

| | datasets | Nom_columns | Nombr_ligne | Nombrl_cols | Nombr_duplic | Nombr_manquent |
|---|---|---|---|---|---|---|
| 0 | customers | customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state | 99441 | 5 | 0 | 0 |
| 1 | geolocation | geolocation_zip_code_prefix, geolocation_lat, geolocation_lng, geolocation_city, geolocation_state | 738332 | 5 | 0 | 0 |
| 2 | items | order_id, order_item_id, product_id, seller_id, shipping_limit_date, price, freight_value | 112650 | 7 | 0 | 0 |
| 3 | payments | order_id, payment_sequential, payment_type, payment_installments, payment_value | 103886 | 5 | 0 | 0 |
| 4 | reviews | review_id, order_id, review_score | 99224 | 3 | 0 | 0 |
| 5 | orders | order_id, customer_id, order_status, order_purchase_timestamp, order_estimated_delivery_date | 99441 | 5 | 0 | 0 |
| 6 | products | product_id, product_category_name, product_weight_g | 32951 | 3 | 0 | 0 |
| 7 | sellers | seller_id, seller_zip_code_prefix, seller_city, seller_state | 3095 | 4 | 0 | 0 |
| 8 | category_translation | product_category_name, product_category_name_english | 71 | 2 | 0 | 0 |

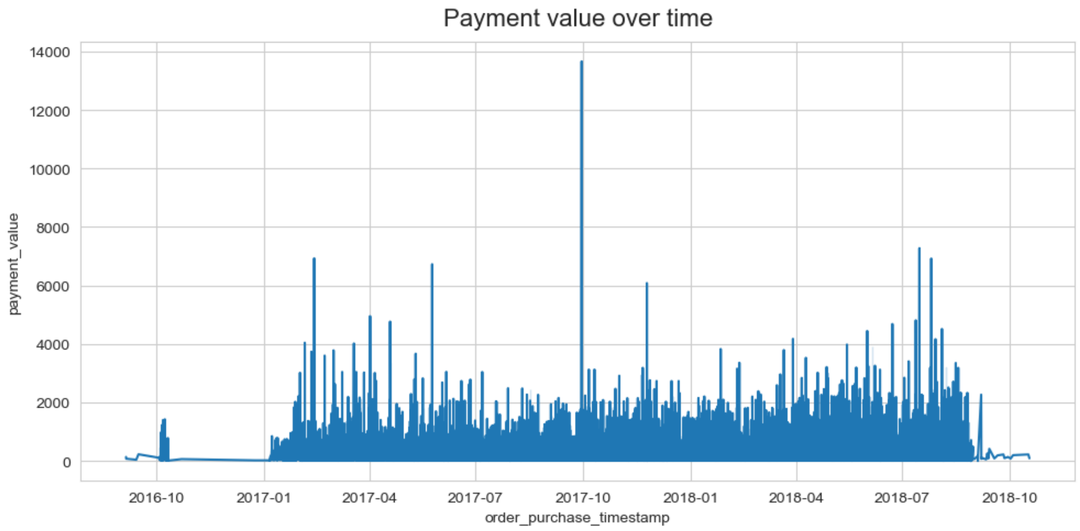
# FUSION DES DONNES

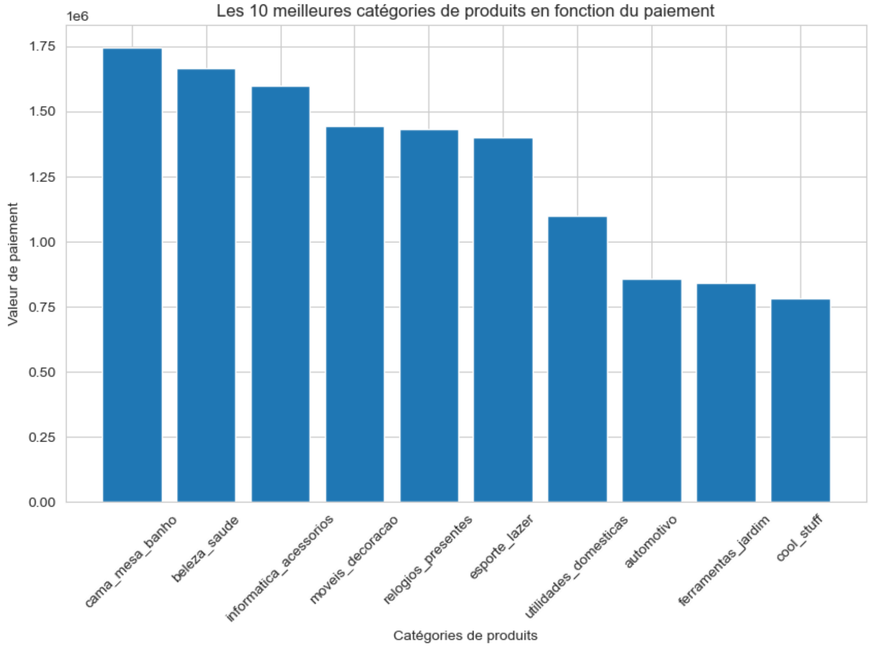
Après avoir nettoyé les données et supprimé les colonnes inutiles, nous allons créer une fusion qui respecte le modèle de données fourni par Olist.

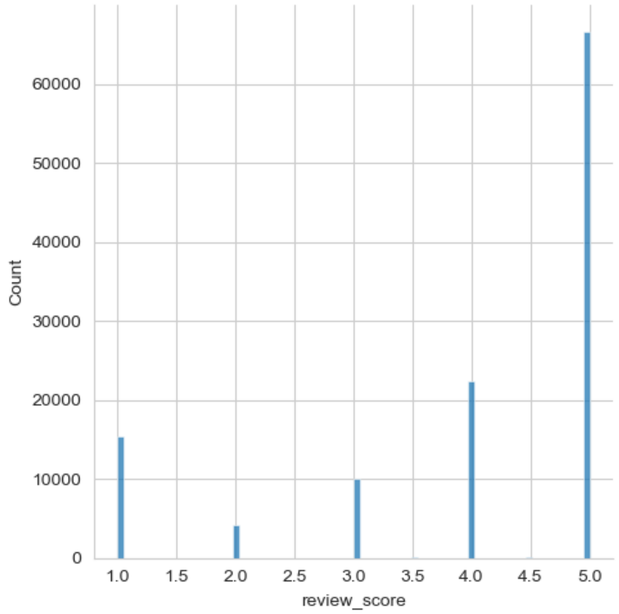


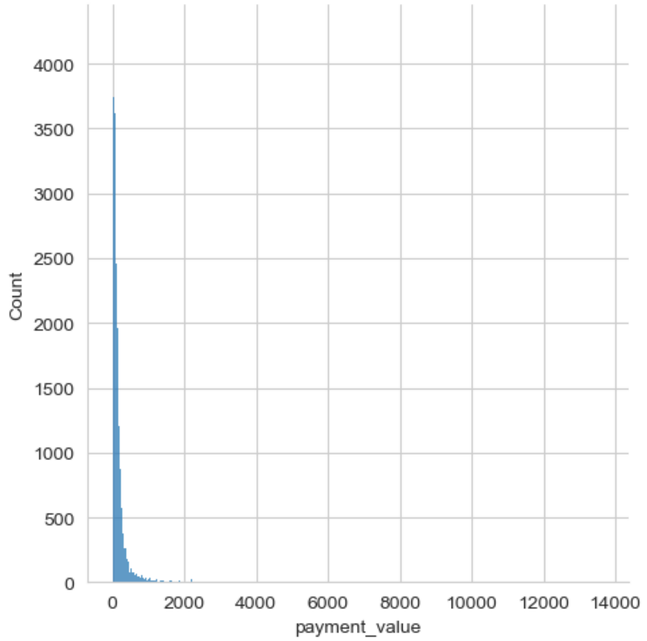

```
merged_df= pd.merge(customers_df, orders_df, on="customer_id", how="left")
merged_df= merged_df.merge(orders_review_df, on="order_id", how="left")
merged_df= merged_df.merge(orders_items_df, on="order_id", how="left")
merged_df= merged_df.merge(products_df, on="product_id", how="left")
merged_df= merged_df.merge(orders_payment_df, on="order_id", how="left")
merged_df= merged_df.merge(sellers_df, on='seller_id', how="left")
merged_df= merged_df.merge(category_trans_df, on='product_category_name', how="left")
```

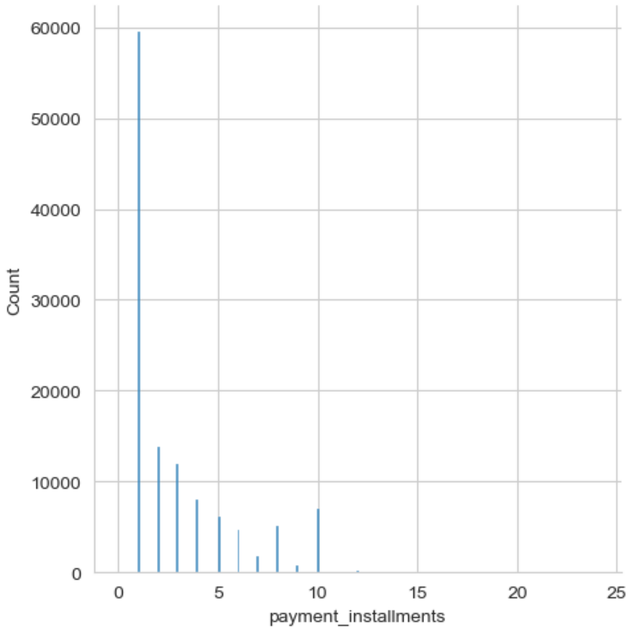
# L'ANALYSE DES VARIABLES



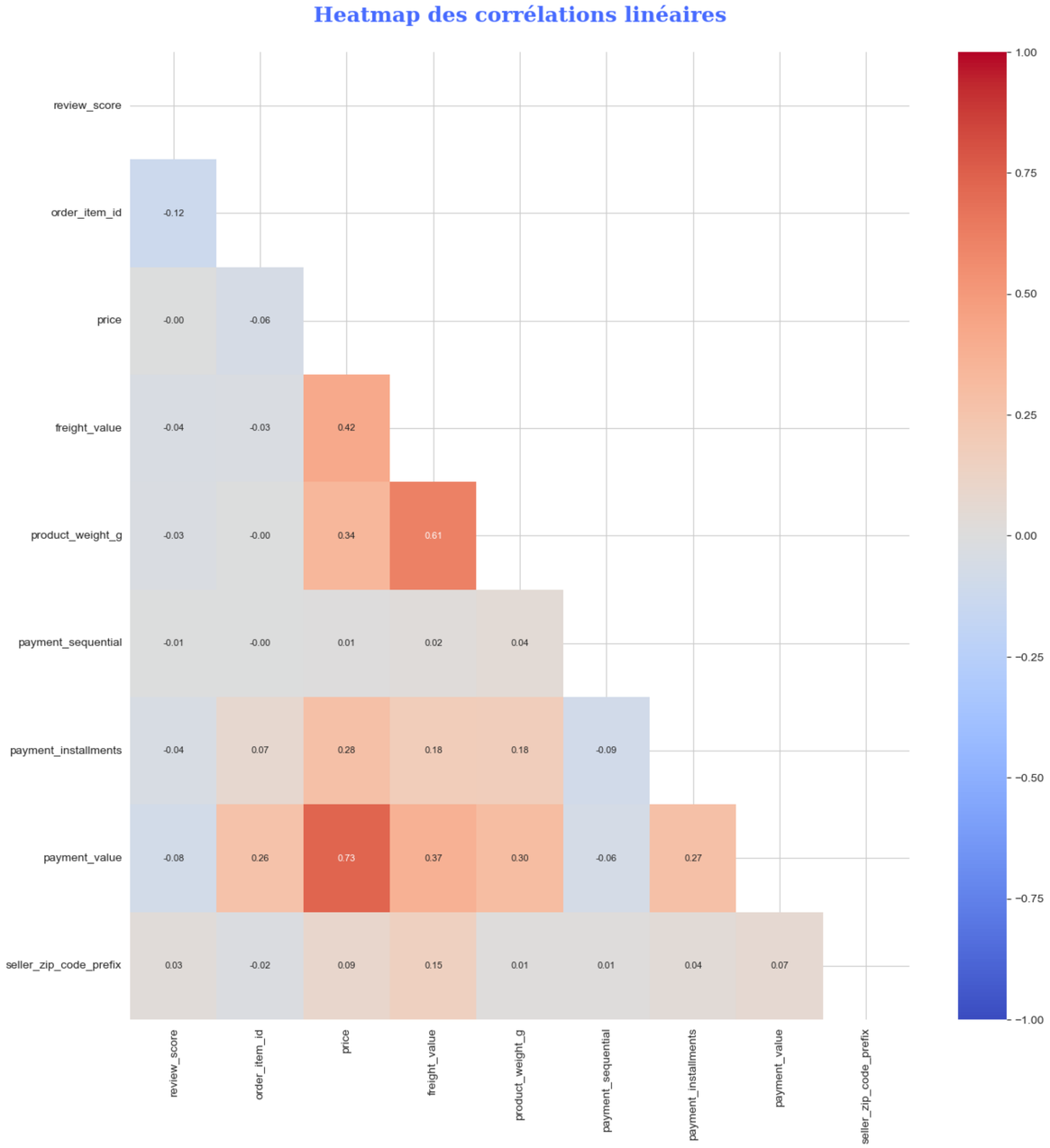



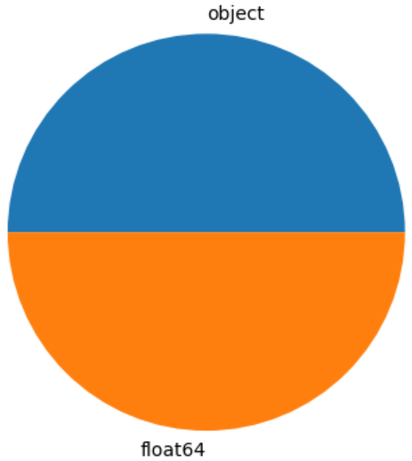



# L'ANALYSE DES VARIABLES




Feature engineering

```
customer_id
customer_unique_id
order_id
order_status
order_purchase_timestamp
order_estimated_delivery_date
review_score
order_item_id
price
freight_value
product_category_name
product_weight_g
payment_sequential
payment_installments
payment_value
seller_zip_code_prefix
seller_city
seller_state
```



# SEGMENTATION Avec RFM

*Récence* : la récence fait référence au temps qui s'est écoulé depuis le dernier achat d'un client. Il s'agit d'un facteur pertinent car les clients qui ont récemment interagi avec une marque sont plus susceptibles de réagir à une nouvelle initiative marketing, ce qui rend leur identification extrêmement précieuse.
*Fréquence* : Il s'agit du nombre de fois qu'un client a effectué un achat. Les consommateurs qui ont acheté plus d'une fois sont plus susceptibles d'acheter à nouveau.
*Monétaire* : ce facteur fait référence à la somme d'argent dépensée par le client sur une période donnée. Les clients qui dépensent beaucoup d'argent sont plus susceptibles de dépenser de l'argent à l'avenir et ont une grande valeur pour une entreprise.

| customer_unique_id | recence | frequence | monetary |
|---|---|---|---|
| 0000366f3b9a7992bf8c76cfdf3221e2 | 161 | 1 | 141.90 |
| 0000b849f77a49e4a4ce2b2a4ca5be3f | 164 | 1 | 27.19 |
| 0000f46a3911fa3c0805444483337064 | 586 | 1 | 86.22 |
| 0000f6ccb0745a6a4b88665a16c9f078 | 370 | 1 | 43.62 |
| 0004aac84e0df4da2b147fca70cf8255 | 337 | 1 | 196.89 |

```
#la date du dernier achat dans une nouvelle dataframe
recence_df = merged_df.groupby(['customer_unique_id'], as_index = False)['order_purchase_time
```

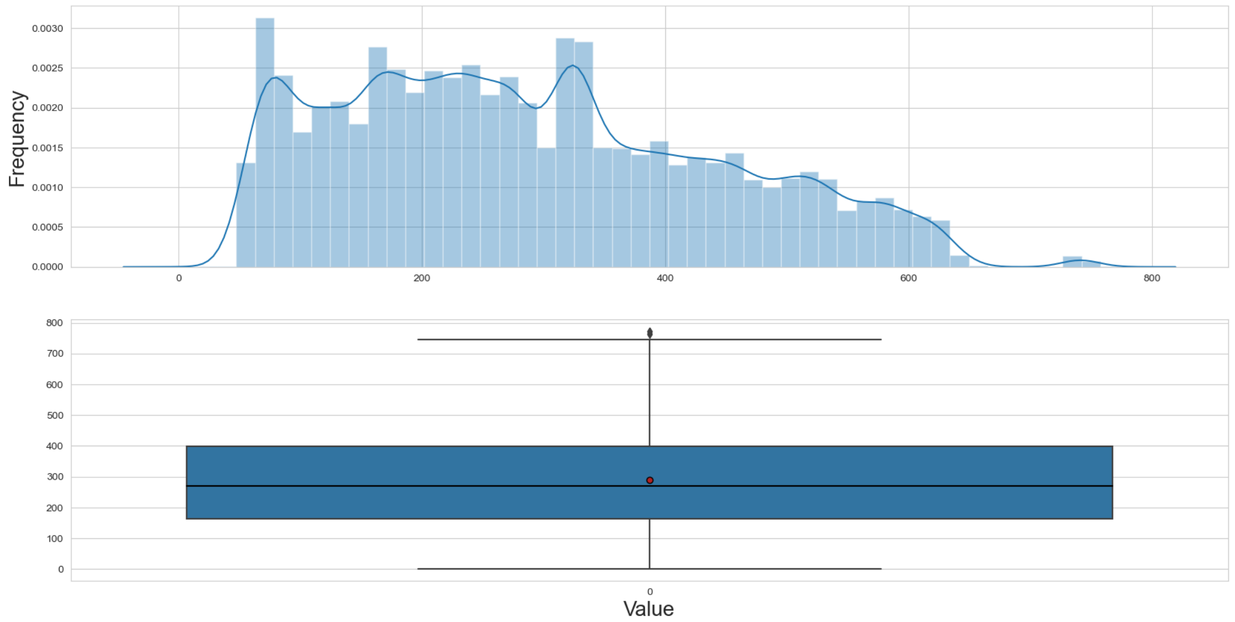
```
#la valeur de récence pour chaque client ajoutant une nouvelle colonne
recence_df['recence'] = recence_df['order_purchase_timestamp'].apply(lambda x: (date_actuell
recence_df.head()
```

```
#Calculate payment value group by customer id in a new dataframe
mont_df = merged_df.groupby('customer_unique_id', as_index=False)['payment_value'].sum()

#Rename 'payment value' column to 'monetary'
mont_df.rename(columns={"payment_value":"monetary"}, inplace=True)

#Show results
mont_df.head()
```

```
#Obtain the number of unique orders by customer in a new dataframe
frequence_df = pd.DataFrame(merged_df.groupby(["customer_unique_id"], as_index = False).agg({

#Rename column to 'frequency'
frequence_df.rename(columns={"order_id":"frequence"}, inplace=True)

#Show results
frequence_df.head()
```

# SEGMENTATION Avec RFM

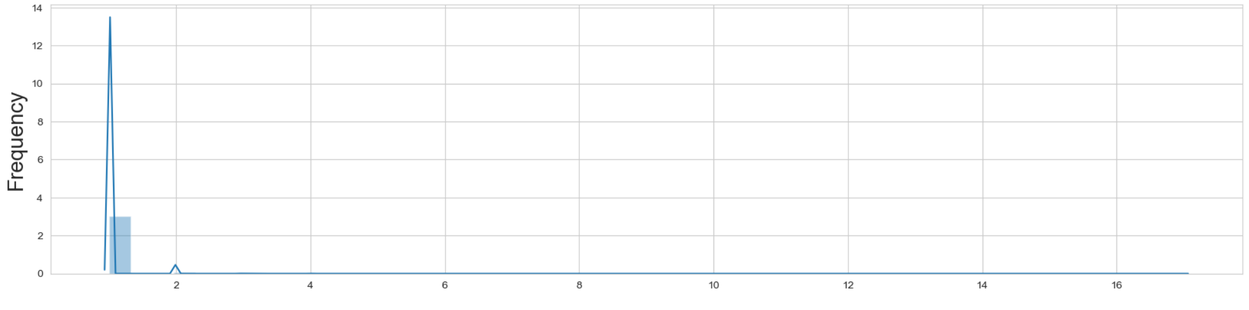
Statistical distribution: recence

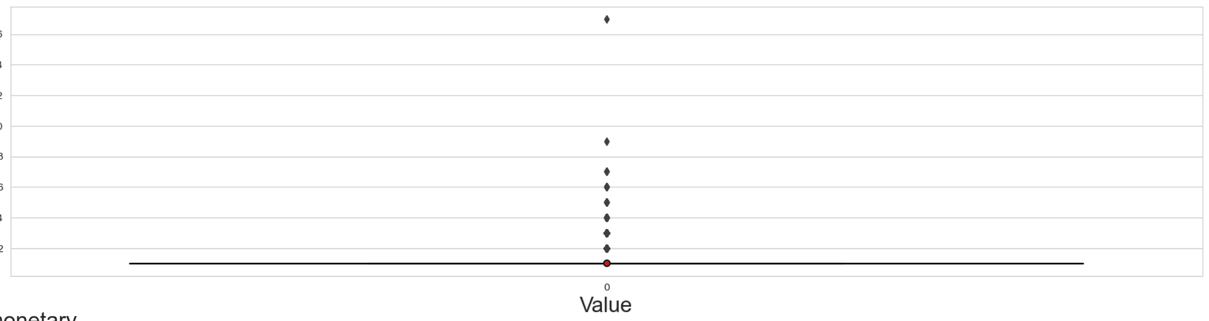




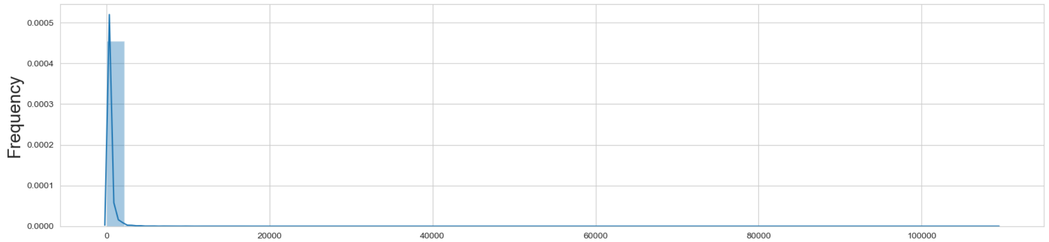


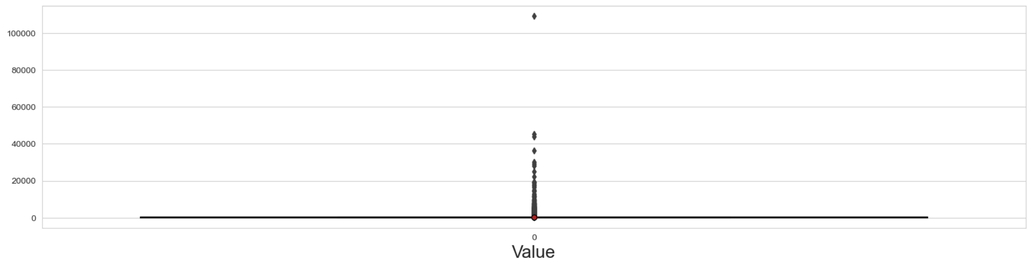
Statistical distribution: frequence




Statistical distribution: monetary

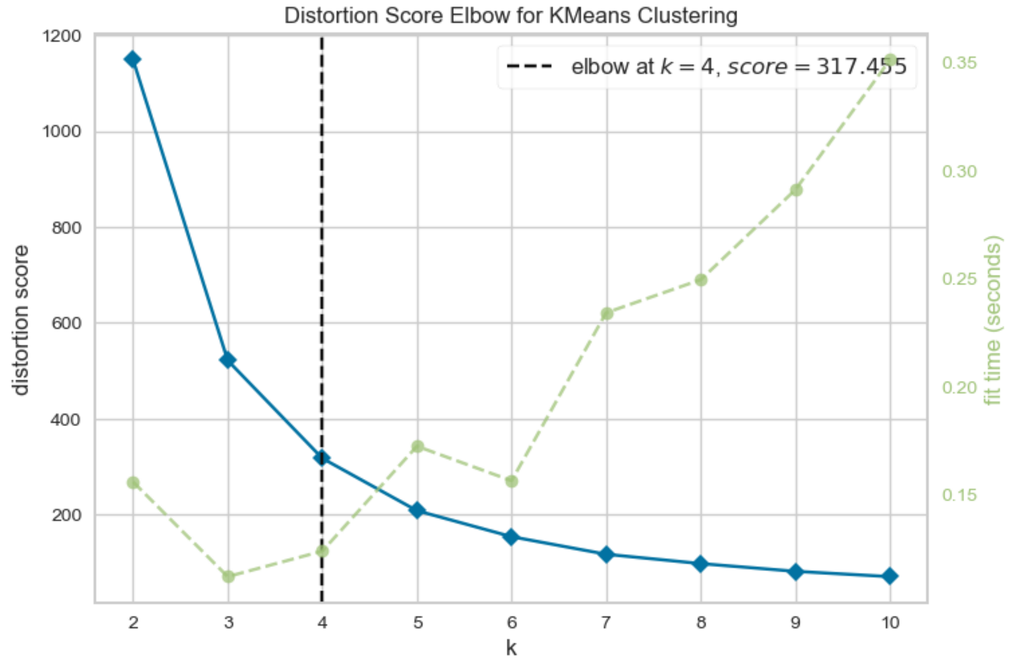


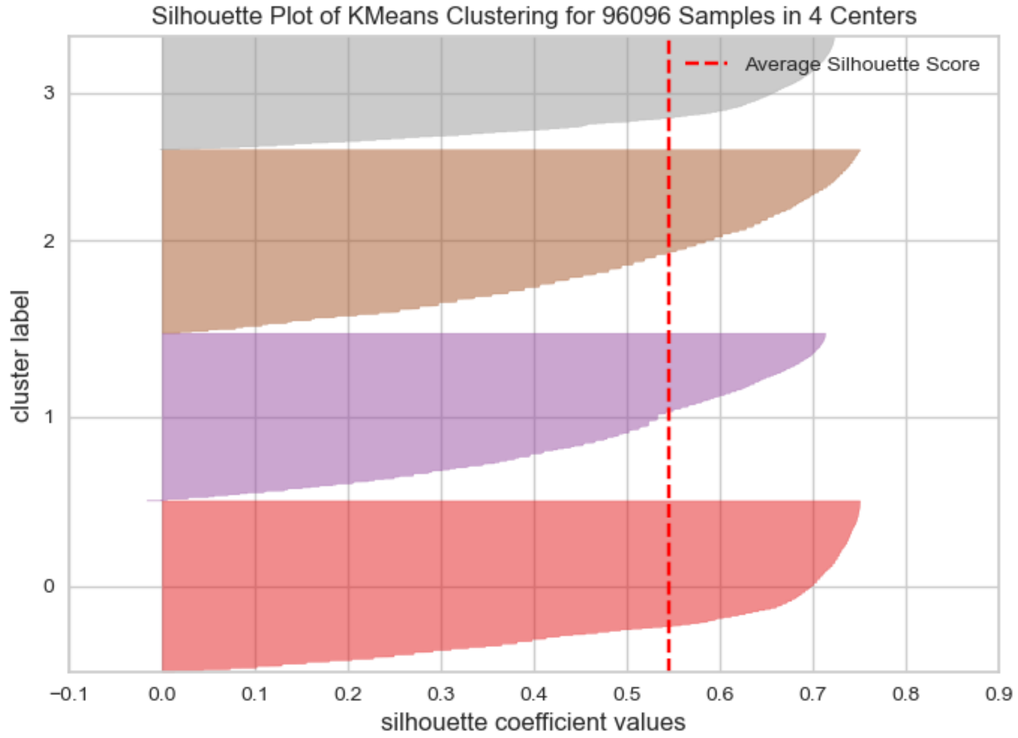
# SEGMENTATION Avec K-means

On commence le clustering avec K-Means pour les colonnes RFM. Ensuite, nous utilisons deux techniques, à savoir la méthode du coude et la méthode de la silhouette, pour déterminer le nombre optimal de clusters dans notre algorithme de clustering.

```
from sklearn.cluster import KMeans
from sklearn.cluster import MeanShift, estimate_bandwidth
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer, InterclusterDistance

X = rfm_df[['recence', 'frequence', 'monetary']]

# méthode du coude
kmeans_visualizer = Pipeline([
    ("preprocessor", preprocessor),
    ("kelbowvisualizer", KElbowVisualizer(KMeans(),K=(2,10)))])
kmeans_visualizer.fit(X)
kmeans_visualizer.named_steps['kelbowvisualizer'].show()
```



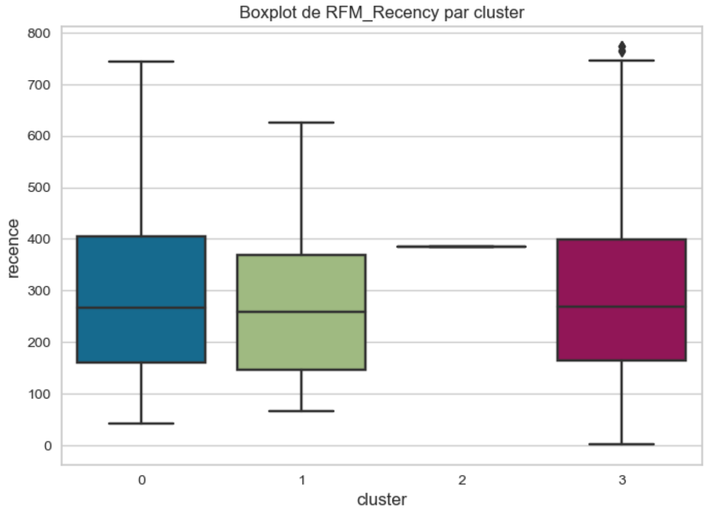

```
K = kmeans_visualizer.named_steps['kelbowvisualizer'].elbow_value_

# Silhouette Visualizer
visualizer_Silh = Pipeline([
    ("preprocessor", preprocessor),
    ("visualizer_Silh", SilhouetteVisualizer(KMeans(K)))])
visualizer_Silh.fit(X)
visualizer_Silh.named_steps['visualizer_Silh'].show()
```

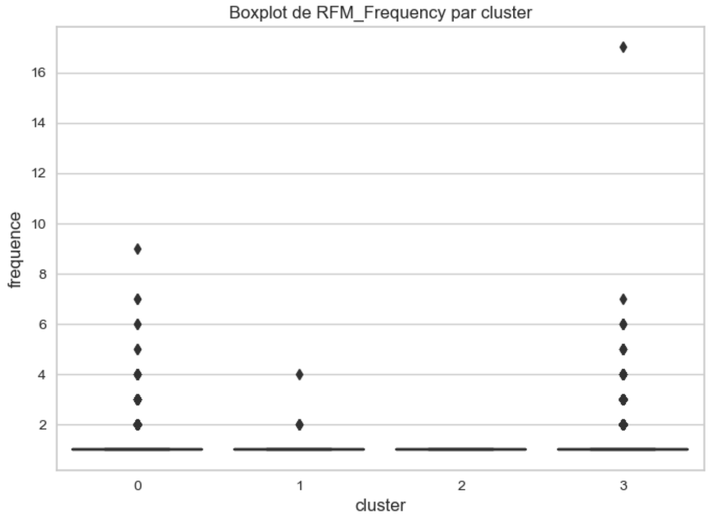


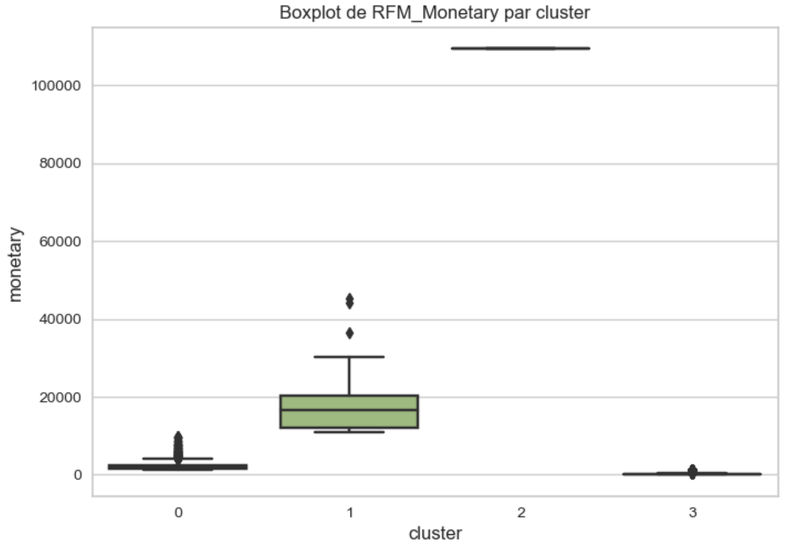
# SEGMENTATION Avec K-means

## Analyse de cluster

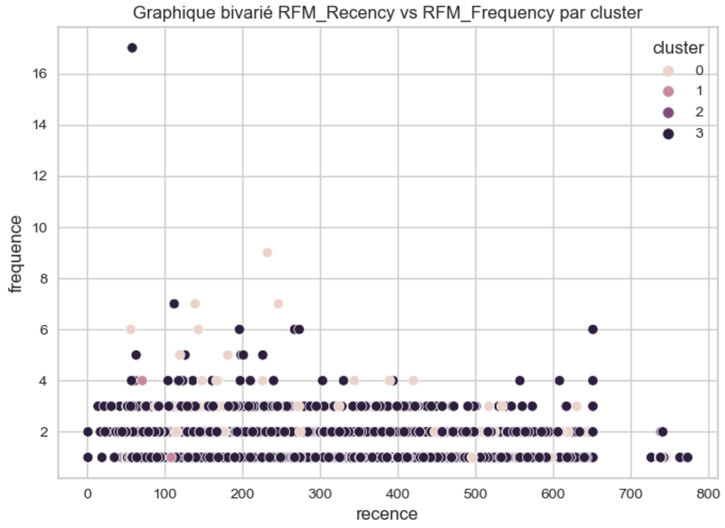
### ANALYSE UNIVARIEE

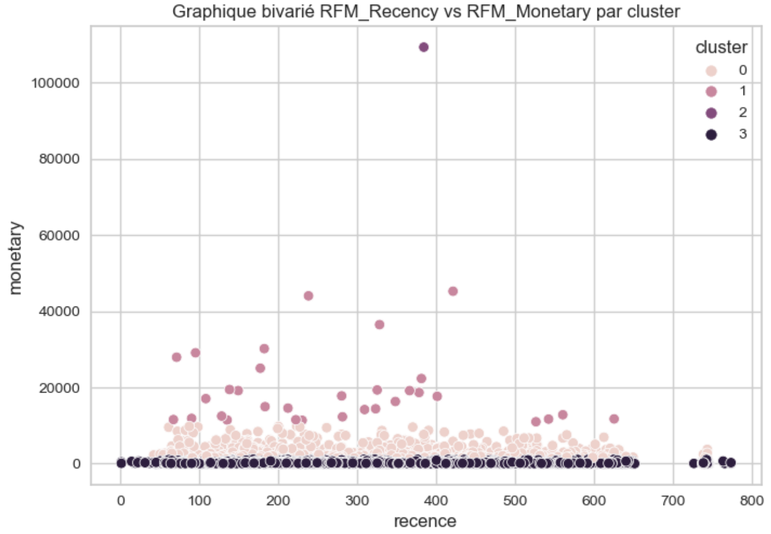


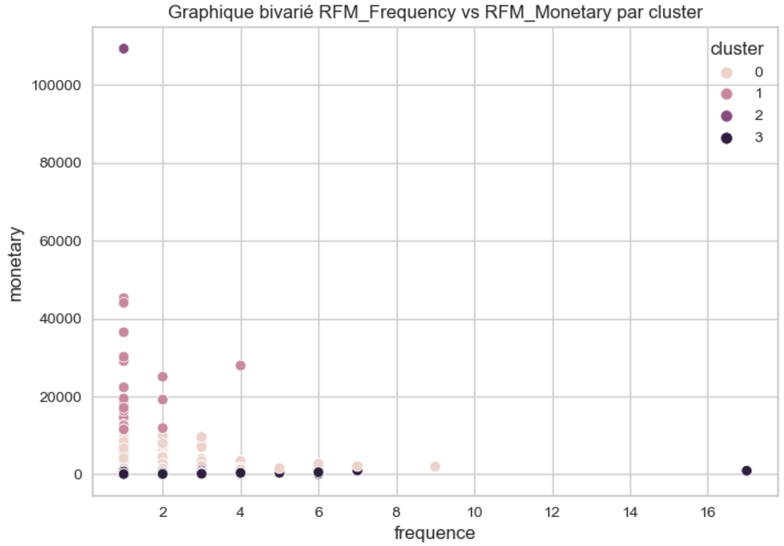




### ANALYSE BIVARIEE



# Analyse de cluster

| | recence | frequence | monetary | |
|---|---|---|---|---|
| | mean | mean | mean | count |
| cluster | | | | |
| **0** | 290.0 | 1.0 | 2270.0 | 1746 |
| **1** | 276.0 | 1.0 | 19159.0 | 32 |
| **2** | 384.0 | 1.0 | 109313.0 | 1 |
| **3** | 289.0 | 1.0 | 169.0 | 94317 |

```python
# compter le nombre de clients dans chaque cluster
count = rfm_df['cluster'].value_counts()

# afficher le résultat
print(count)
```

```
3    94317
0     1746
1       32
2        1
```

**Après avoir examiné ces résultats, nous pouvons tirer les conclusions suivantes :**

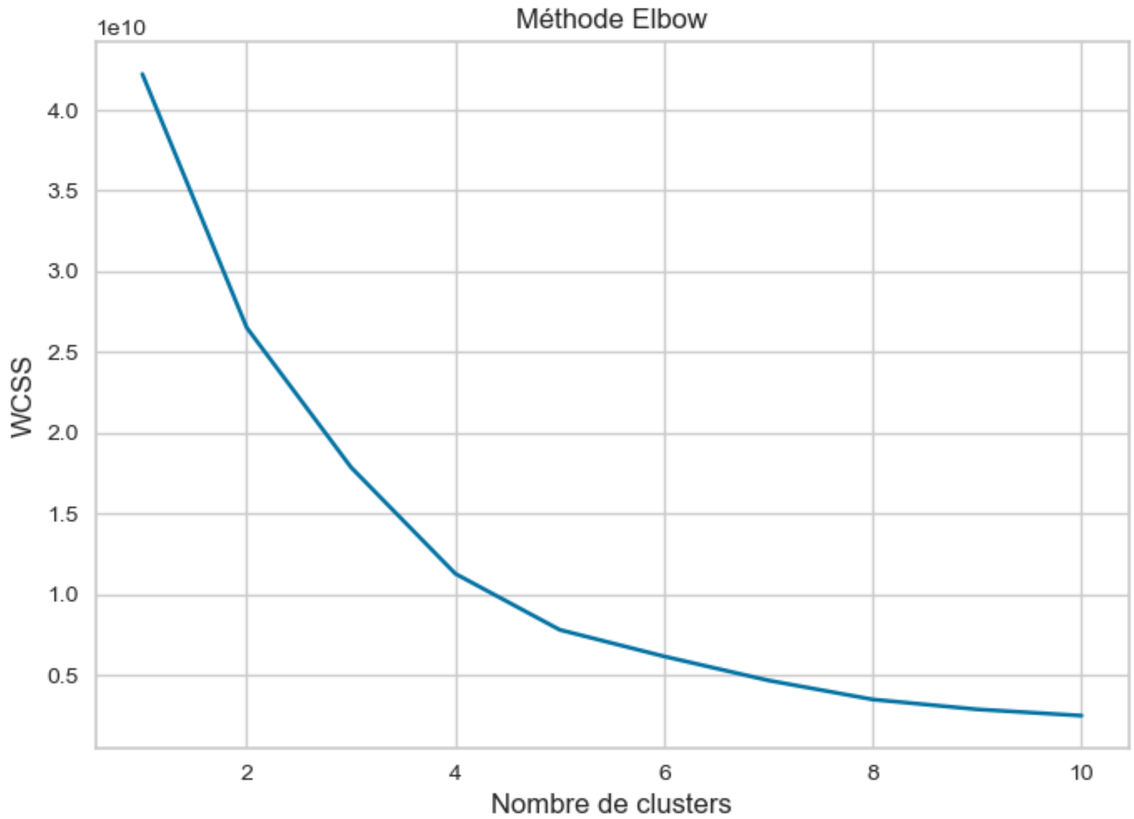
**1) Le grand cluster contenant 94317 clients est de loin le plus important, car il représente la majorité des données. Cela peut suggérer que les clients de ce cluster ont des comportements similaires et qu'ils constituent une part importante de la base de données clients.**
**2) Les clusters plus petits sont moins significatifs en raison de leur taille réduite, mais ils peuvent encore fournir des informations importantes sur les sous-groupes de clients qui ont des comportements distincts.**
**2) Le fait qu'un cluster ne contienne qu'un seul client peut indiquer que ce client a des caractéristiques très différentes des autres clients. Par conséquent, il peut nécessiter une attention particulière dans l'analyse de marketing.**

# SEGMENTATION Avec K-means

## K-Means RFM avec feature "review_score"

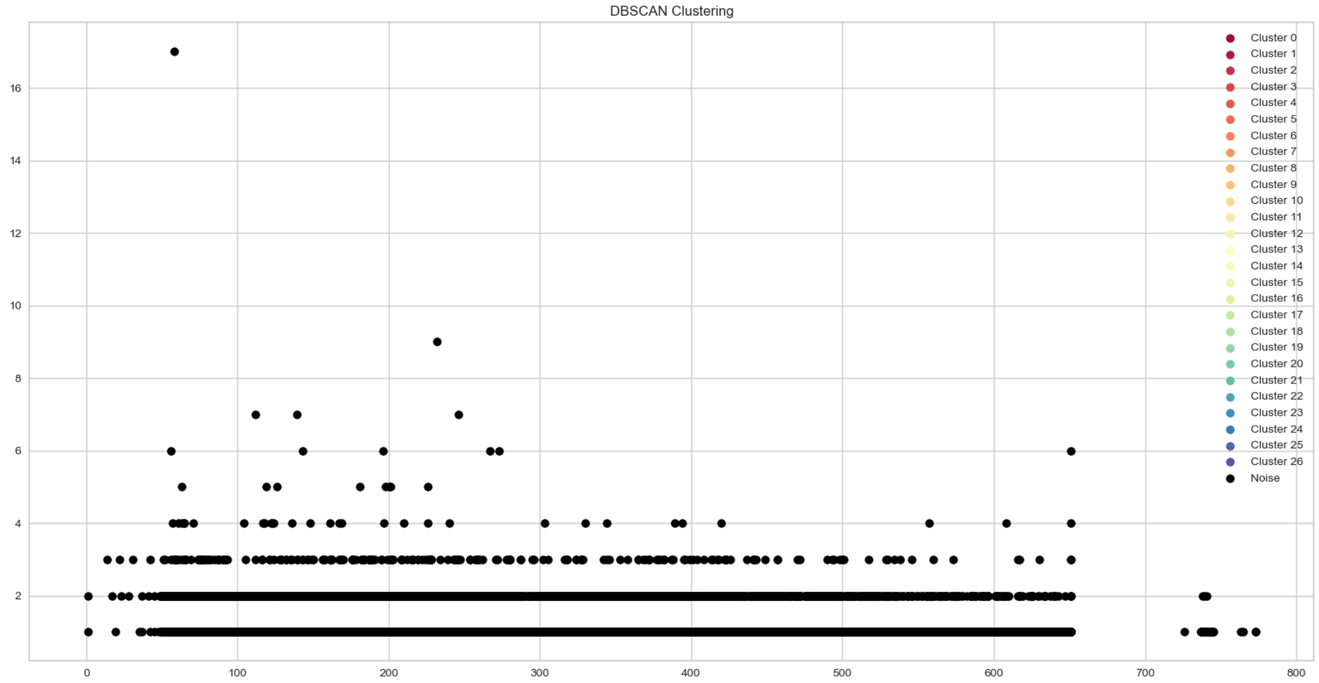
```python
# Ajouter une colonne 'review_score'
rfm_df = rfm_df.assign(review_score=np.random.randint(1, 6, size=len(rfm_df)))

# Afficher le résultat
rfm_df
```

```python
# Sélectionner les features
X = rfm_df[['recence', 'frequence', 'monetary', "review_score"]]

# le nombre optimal de clusters en utilisant la méthode "Elbow"
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Méthode Elbow')
plt.xlabel('Nombre de clusters')
plt.ylabel('WCSS')
plt.show()
```



# SEGMENTATION Avec DEBSCAN

Après avoir obtenu les résultats de K-means, nous avons testé d'autres algorithmes tels que DBSCAN pour obtenir des résultats supplémentaires et parvenir à une conclusion satisfaisante. Cependant, les résultats de DBSCAN ont révélé la présence de 27 clusters et près de 9000 points de bruit qui, d'un point de vue métier, n'ont aucun sens et ne sont pas logiques.




```
# compter le nombre de clusters et le nombre de points de données considérés comme du brui
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)

print('Nombre de clusters : {}'.format(n_clusters))
print('Nombre de points de données considérés comme du bruit : {}'.format(n_noise))
```

```
Nombre de clusters : 27
Nombre de points de données considérés comme du bruit : 95672
```

```
X = rfm_df.loc[:, ['recence', 'frequence', 'monetary']].values
dbscan = DBSCAN(eps=0.5, min_samples=10)
dbscan.fit(X)
```

```
▼ DBSCAN
DBSCAN(min_samples=10)
```

```
labels = dbscan.labels_
labels
```

```
array([-1, -1, -1, ..., -1, -1, -1])
```

# MAINTENANCE

La stabilité temporelle est une mesure importante de la fiabilité et de la validité d'un algorithme de clustering. Si un algorithme de clustering n'est pas stable temporellement, cela peut indiquer qu'il est sensible aux fluctuations aléatoires des données et qu'il peut produire des résultats différents selon le moment où il est exécuté.

Dans le but d'établir un contrat de maintenance de l'algorithme de segmentation client, nous devons tester sa stabilité dans le temps et voir, par exemple, à quel moment les clients changent de Cluster.

## ETAPE 1

nous allons définir une date limite pour effectuer une analyse de segmentation de cluster à l'aide du modèle choisi en utilisant les valeurs RFM à cette date limite. En utilisant les valeurs RFM à la date limite T0, il est possible d'effectuer une analyse de segmentation de cluster avec le modèle choisi. Dans ce cas, le modèle utilisé est la méthode K-Means avec n_clusters=4 clusters.

```
# Nous allons egalement creer un StandardScaler
S0 = StandardScaler()
RFM0_scaled = S0.fit_transform(RFM0[["recency", "frequency", "monetary"]])

#les identifiants de cluster attribués à chaque client.
C0 = M0.fit_predict(RFM0_scaled)
C0
```

```
/Users/sbeddyy/anaconda3/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:8
Warning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
f `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
array([3, 3, 0, ..., 2, 3, 3], dtype=int32)
```

## ETAPE 2

nous allons maintenant répéter le processus en modifiant la date limite de référence en ajoutant quelques jours (T1 = T0 + 15j). Cela nous permettra de voir l'évolution des comportements de la clientèle sur une nouvelle période et d'adapter nos stratégies marketing et commerciales en conséquence.

```
S1 = StandardScaler()
RFM1_scaled = S1.fit_transform(RFM1[["recency", "frequency", "monetary"]])
#Nouvelle liste de cluster
C1 = M1.fit_predict(RFM1_scaled)
C1
```

```
/Users/sbeddyy/anaconda3/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:8
Warning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
f `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
array([2, 0, 2, ..., 1, 2, 2], dtype=int32)
```

## ETAPE 3

Cette étape nous permettra de voir comment la segmentation des clients a évolué entre T0 et T1, en utilisant les mêmes clusters que ceux définis à la date limite T0

```
C2 = M0.predict(RFM1_scaled)
C2
```

```
array([3, 0, 3, ..., 2, 3, 3], dtype=int32)
```

# MAINTENANCE

## ETAPE 4

Pour mesurer la qualité de la segmentation de cluster, nous utiliserons l'indice ARI (Adjusted Rand Index), qui est indépendant de la numérotation des clusters

```
ARI(C1, C2)
```

```
0.9987324632217486
```

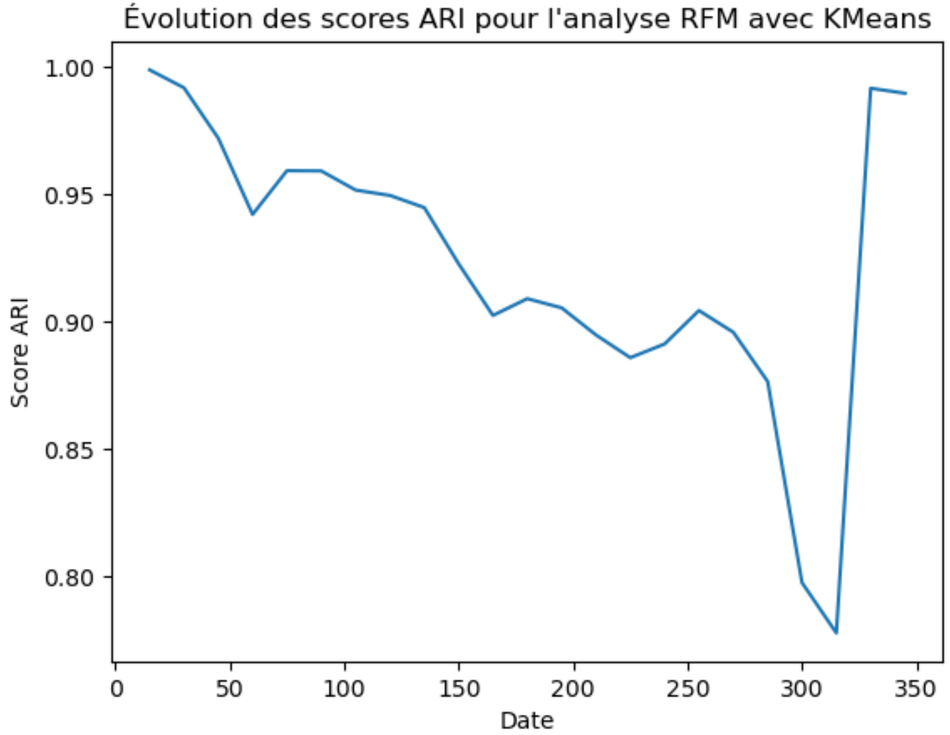
Ainsi, à partir de ce score, deux remarques essentielles peuvent être tirées :

Les comportements des clients n'ont pas beaucoup changé entre T0 et T1, car les clusters obtenus sont très similaires. Cela peut indiquer que les clients ont maintenu les mêmes comportements d'achat au fil du temps, ou que les changements ont été minimes et n'ont pas eu d'impact significatif sur la segmentation de cluster.

Le modèle initial M0 est toujours valable pour la période de temps T1, car les mêmes clusters ont été obtenus pour les deux périodes. Cela suggère que le modèle initial est robuste et peut être utilisé pour segmenter les clients sur des périodes de temps similaires à T0 et T1.

## ETAPE 5

Dans cette étape, nous allons simuler plusieurs périodes et afficher l'évolution du score ARI pour évaluer la stabilité de l'algorithme KMeans au fil du temps.



# CONCLUSION

- L’analyse RFM nous permet d’identifier les différents clients en particulier les meilleurs(best customer).

- Les algorithmes de clustering permettent de regrouper les données proches mais posent néanmoins un problème d’interprétation(par exemple quel est le segment de best customer).

- L’analyse RFM est meilleur que les algorithmes automatiques pour cette tâche. Ces derniers peuvent néanmoins être combinés pour une analyse encore plus pertinentes.