Segmentez des clients d'un site e-commerce

Projet 5: Moustapha ABDELLAHI

OPENCLASSROOMS

Sommaire

- **Présentation de la problématique**
- **Découverte des données**
- **Préparation des données**
- **Approches de modélisation**
- **Conclusions**

PROBLEMATIQUE DE L'ENTREPRISE OLIST

- **Mission :**

- **Consultant pour Olist, une entreprise brésilienne de vente en ligne sur les marketplaces.**

- **Objectifs :**

- **Fournir aux équipes du e-commerce une segmentation pour les campagnes de communication quotidiennes.**

- **Comprendre les différents types d'utilisateurs grâce à leur comportement**

- **Fournir à l'équipe marketing une description actionnable de la segmentation**

- **Proposer un contrat de maintenance**

ANALYSE DE LA PROBLEMATIQUE

- **Explorer les données et choisir les features pertinentes**
- **Problème de machine learning non supervisé (clustering)**
- **Les clusters doivent être faciles à expliquer et utilisables**

DECOUVERTE DES DONNEES

- **Base de données (Olist) anonymisée téléchargeable :**

(https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce)

- **9 datasets :’olist_geolocation_dataset.csv’/'olist_customers_dataset/ ‘olist_orders_dataset’/olist_order_items_dataset’/ olist_order_payments_dataset’/‘olist_order_reviews_dataset’/ ‘olist_products_dataset’/ ‘olist_sellers_dataset'/'product_category_name_translation'**

- **7 datasets : sont utilisés pour créer la table RFM**

PREPARATION DES DONNEES

Fusion et Nettoyage des données

- **Jointure des différents datasets**

order_id	order_status	order_purchase_timestamp	order_delivered_customer_date	order_item_id	product_id	price
e1b050b8499577073aeb2a297a1	delivered	2017-05-16 15:05:35	2017-05-25 10:35:35	1	a9516a079e37a9c9c36b9b78b10169e8	124.99
34febe9cd269e378117d6681172	delivered	2017-11-09 00:50:13	2017-11-28 00:09:50	1	a9516a079e37a9c9c36b9b78b10169e8	112.99
34febe9cd269e378117d6681172	delivered	2017-11-09 00:50:13	2017-11-28 00:09:50	2	a9516a079e37a9c9c36b9b78b10169e8	112.99
5365d330d10485e0203d54ab9e8	delivered	2017-05-07 20:11:26	2017-05-26 09:54:04	1	a9516a079e37a9c9c36b9b78b10169e8	124.99
b3614664aa66867856dba7e61b7	delivered	2018-02-03 19:45:40	2018-02-28 21:09:00	1	a9516a079e37a9c9c36b9b78b10169e8	106.99

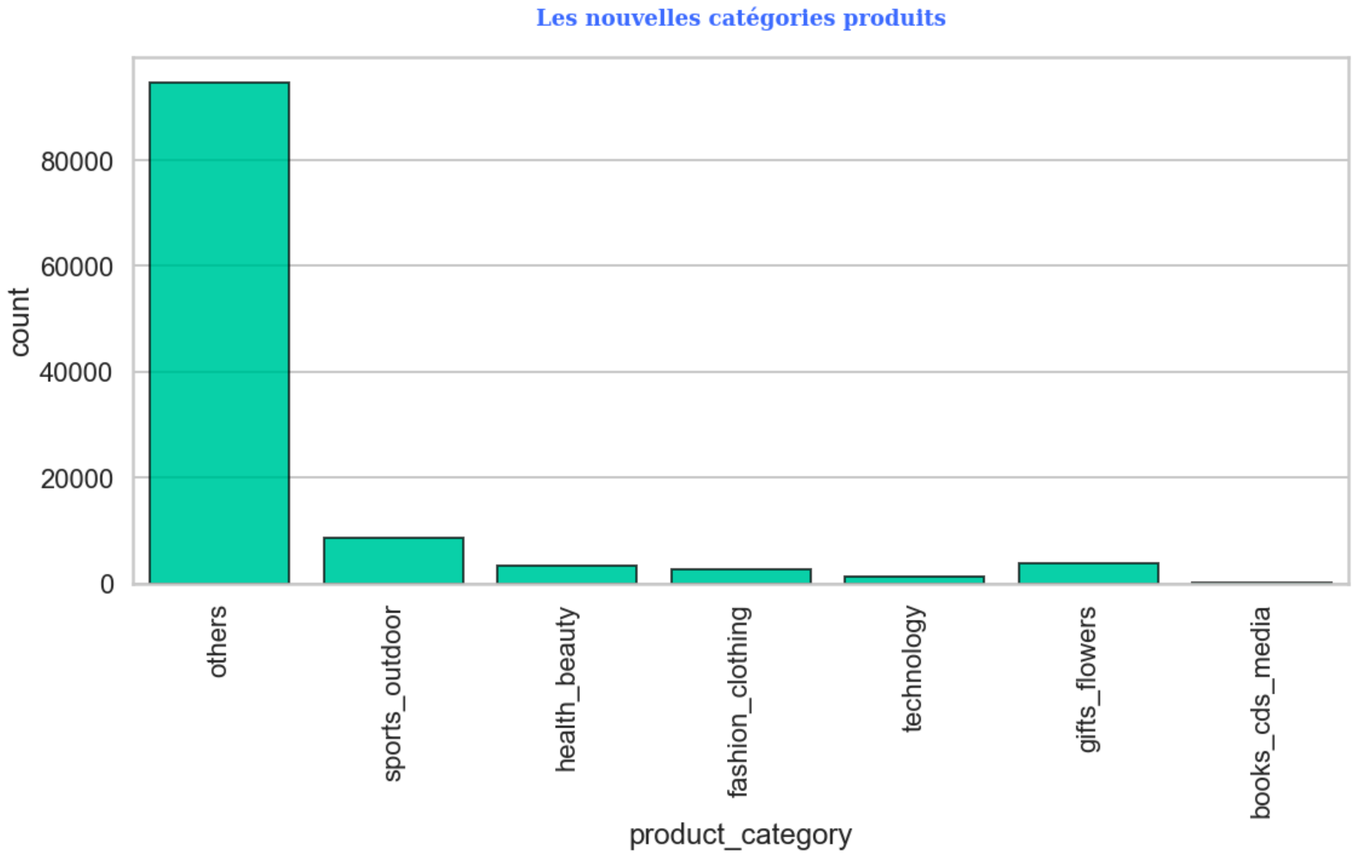
- **Filtrage des données pertinentes**

```
# Filtrage pour ne conserver que les lignes "delivered"

df6 = df6[df6["order_status"] == "delivered"]
```

```
#Check if there is any Price = 0 or <0

df6[(df6['price']==0)|(df6['price']<0)].head()
```

Features Engineering

- **Catégories du dataset résultant trop nombreuses:** → **regroupement en 7 catégories principales en vente en ligne sur le site http://statista.com**

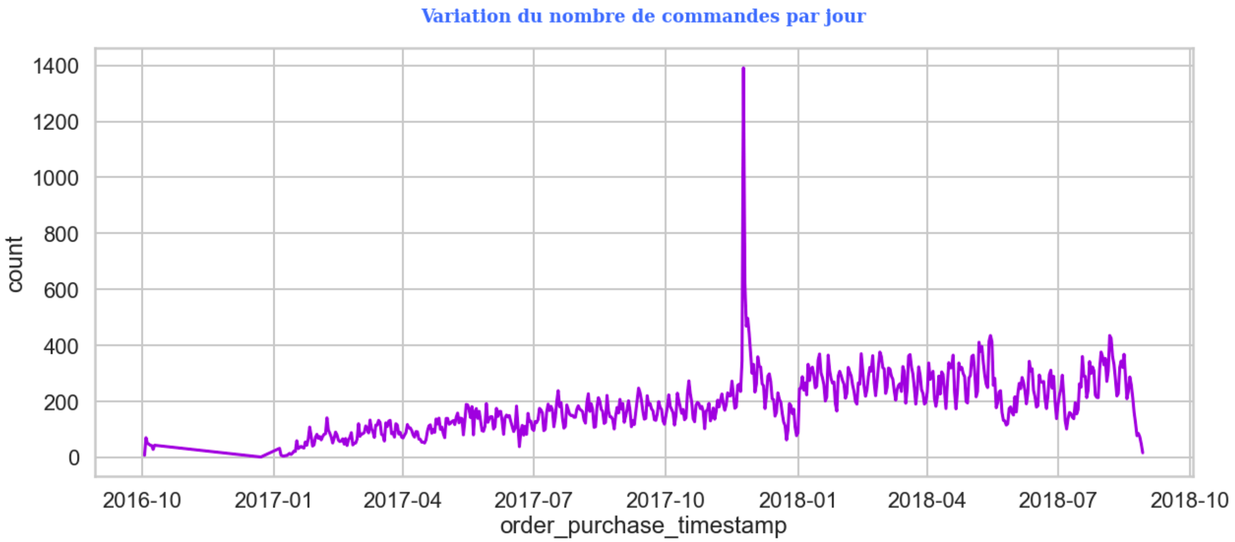


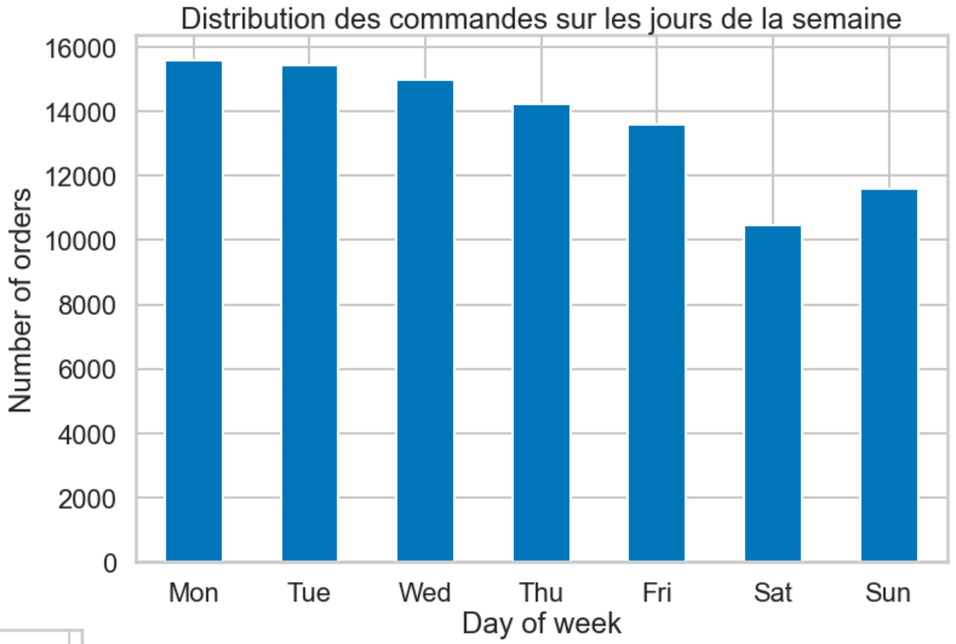
- **Création de nouvelles features :** → **'total_cost'= 'price' +'freight_value', 'mean_score'= mean(review_score), total_orders'.**

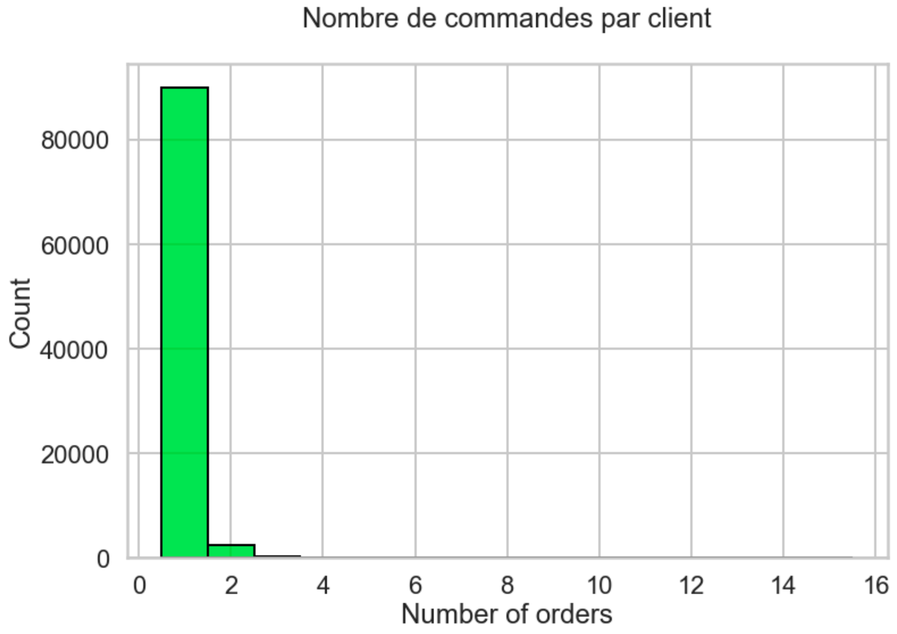
```
#----------------------------------------
total_orders = df6[['customer_unique_id', 'order_id']].groupby(['customer_unique_id']).agg('count').reset_index().\
                        sort_values('order_id', ascending=False)
total_orders.head()
```

PREPARATION DES DONNEES

Exploration



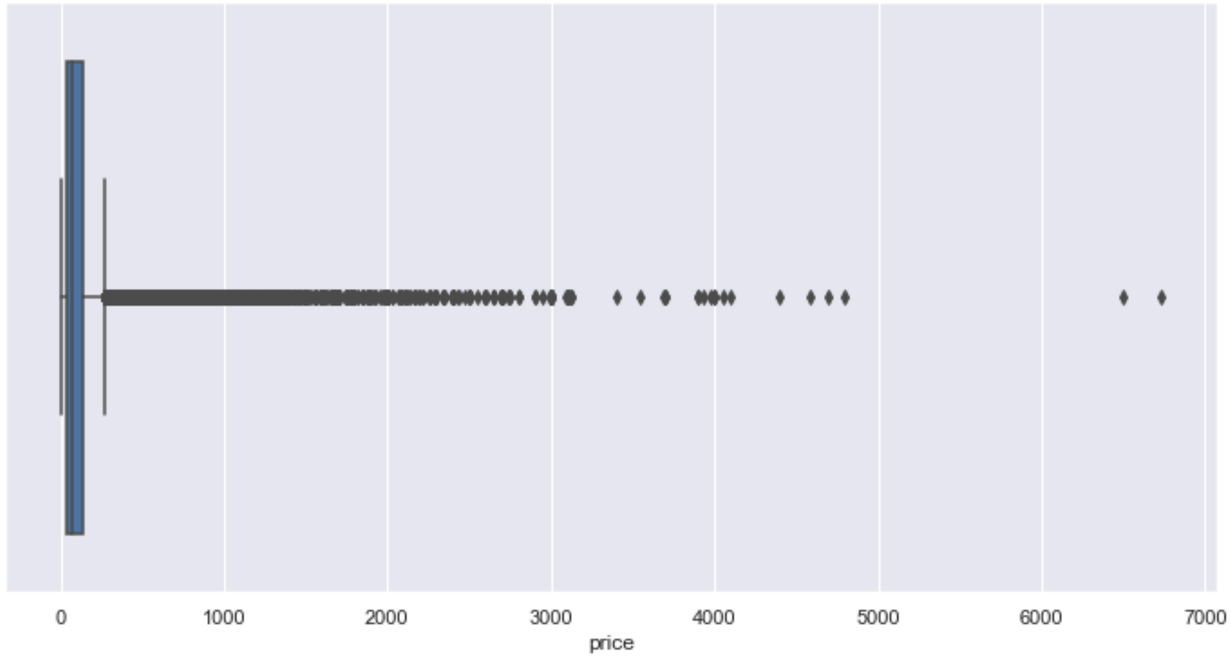


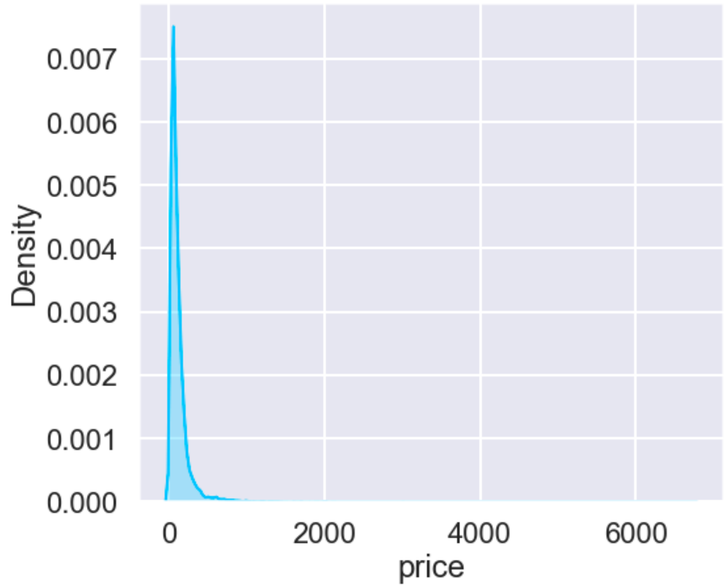


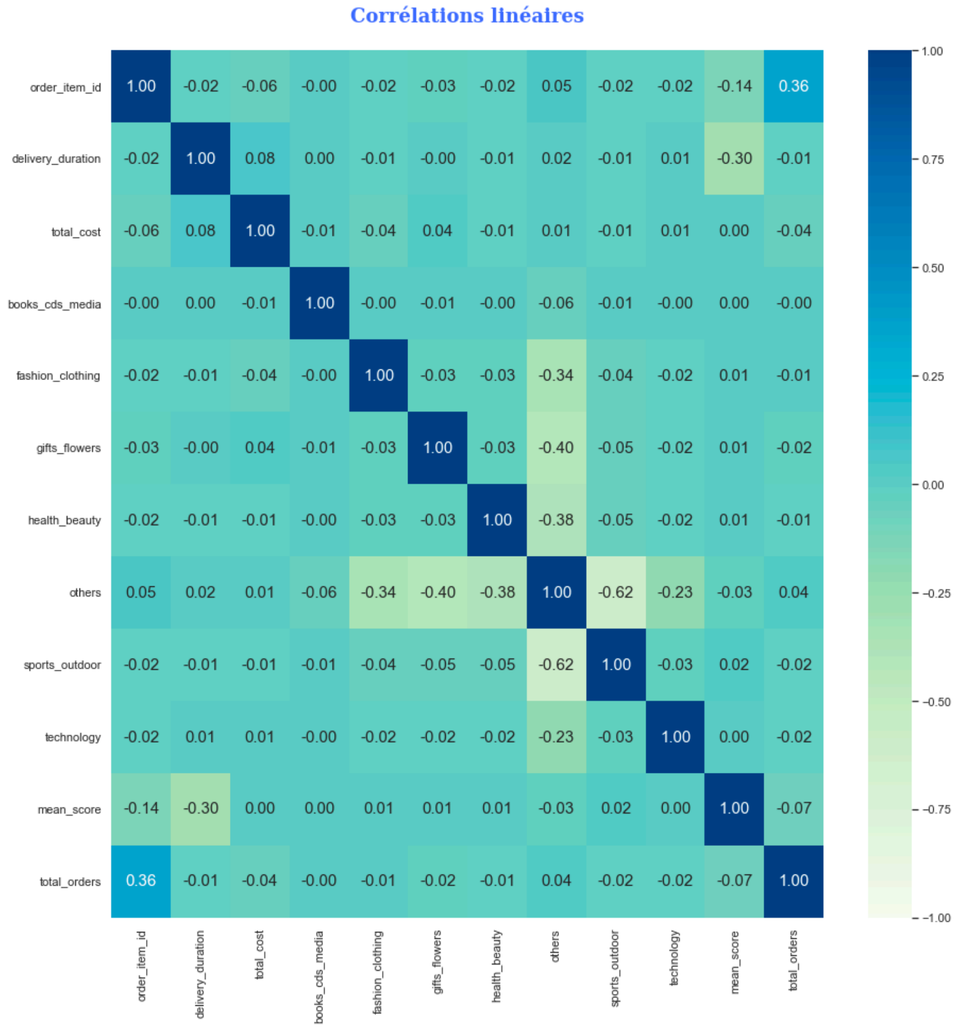
Exploration

- **Dataset final**

dim= (91521 lin., 20 col.)







CREATION DE LA TABLE RFM

- **Récence : nombre de jours depuis le dernier achat d'un client**
- **Fréquence : nombre total d'achat d'un client**
- **Montant : somme totale dépensée par un client**

La table obtenue:

```
rfm=df9.groupby('customer_unique_id').\
agg({'order_purchase_timestamp':lambda x: (today-x.max()).days,'order_id':lambda x: x.nunique(),
                                    'total_cost':'mean'})
rfm
```

```
rfm['order_purchase_timestamp'] = rfm['order_purchase_timestamp'].astype(int)

rfm.rename(columns={'order_purchase_timestamp': 'Recency',
                            'order_id': 'Frequency',
                            'total_cost': 'Monetary'}, inplace=True)
```

```
rfm.head()
```

	customer_unique_id	Recency	Frequency	Monetary
0	0000366f3b9a7992bf8c76cfdf3221e2	111	1	141.90
1	0000b849f77a49e4a4ce2b2a4ca5be3f	114	1	27.19
2	0000f46a3911fa3c0805444483337064	537	1	86.22
3	0000f6ccb0745a6a4b88665a16c9f078	321	1	43.62
4	0004aac84e0df4da2b147fca70cf8255	288	1	196.89

CALCUL DU SCORE RFM

- **Utilisation des quartiles : pour regrouper les clients selon leur comportement**

```
# Split into 4 segments using quartiles

quartiles = rfm.quantile(q=[0.25,0.5,0.75])
quartiles = quartiles.to_dict()
```

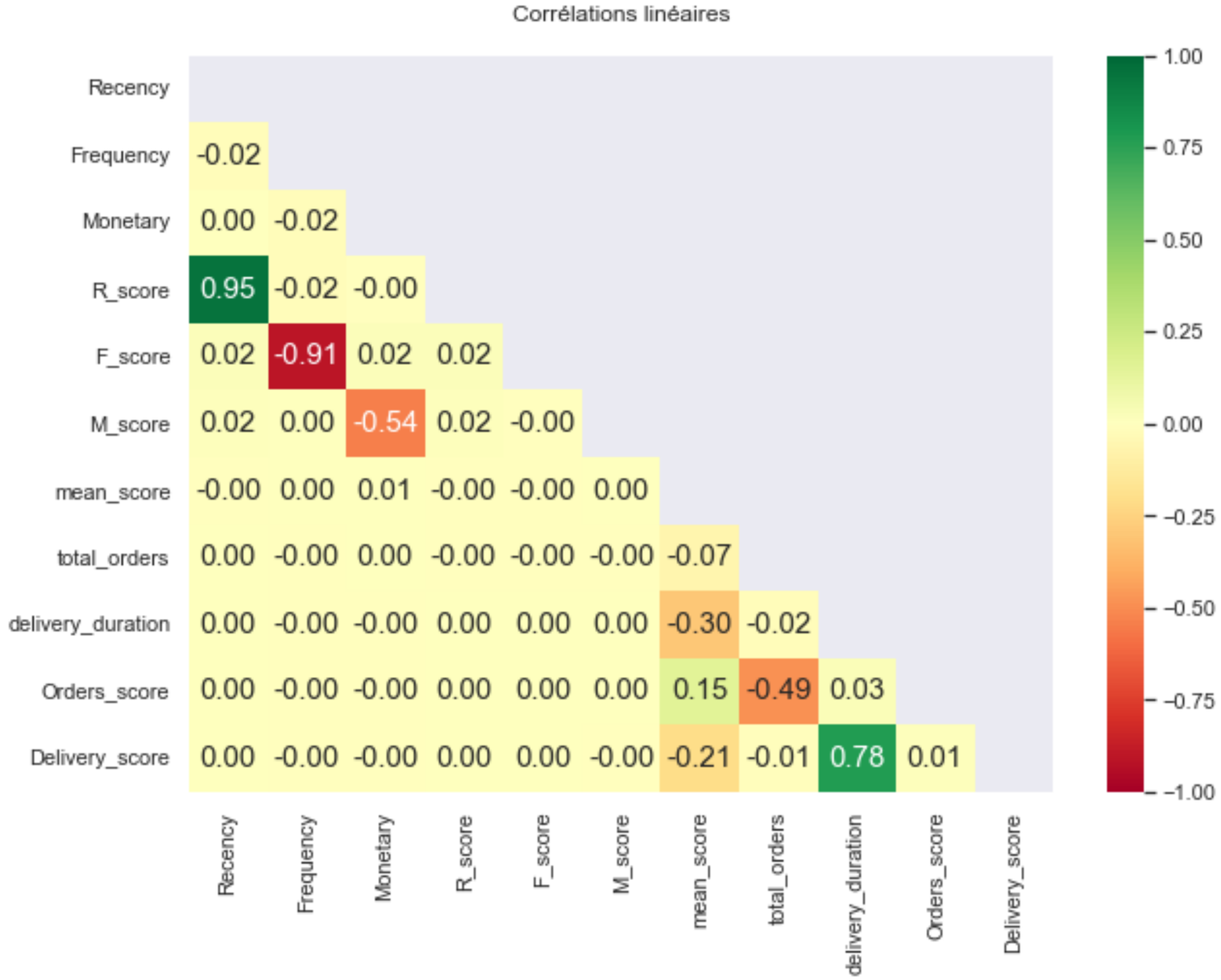
- **Attribution des valeurs : pour la segmentation des clients**

```
#Ajout des colonnes  R, F et M dans le dataset

rfm['R_score'] = rfm['Recency'].apply(RScoring, args=('Recency',quartiles,))
rfm['F_score'] = rfm['Frequency'].apply(FMScoring, args=('Frequency',quartiles,))
rfm['M_score'] = rfm['Monetary'].apply(FMScoring, args=('Monetary',quartiles,))
rfm.head(5)
```

	RFM_score	Customer_segment	mean_score	total_orders	delivery_duration	Number of customers	activity
0	1-1-1	Champion Customers	4.0	1	9.0	167	Bought recently, buy always and spend a lot of money
1	X-1-X	Loyal Customers	1.0	2	19.0	1355	Buy reguralery and responsive to promotion
2	X-X-1	Potential loyalist	1.0	2	19.0	11500	Recent shoppers with high frequency
3	3-1-1	Promising customers	3.0	1	19.0	142	Recent shoppers but don't spend much
4	4-1-1	Dormant Customers	4.0	1	25.0	109	Customers used to be active, but could be lost if not reactivated
5	4-4-4	At risk Customers	4.0	1	19.0	5820	Make purchases somtimes but long time ago. Must bring them back
6	others	Lost customers	4.0	1	13.0	79899	Last buy since long time and low number of orders

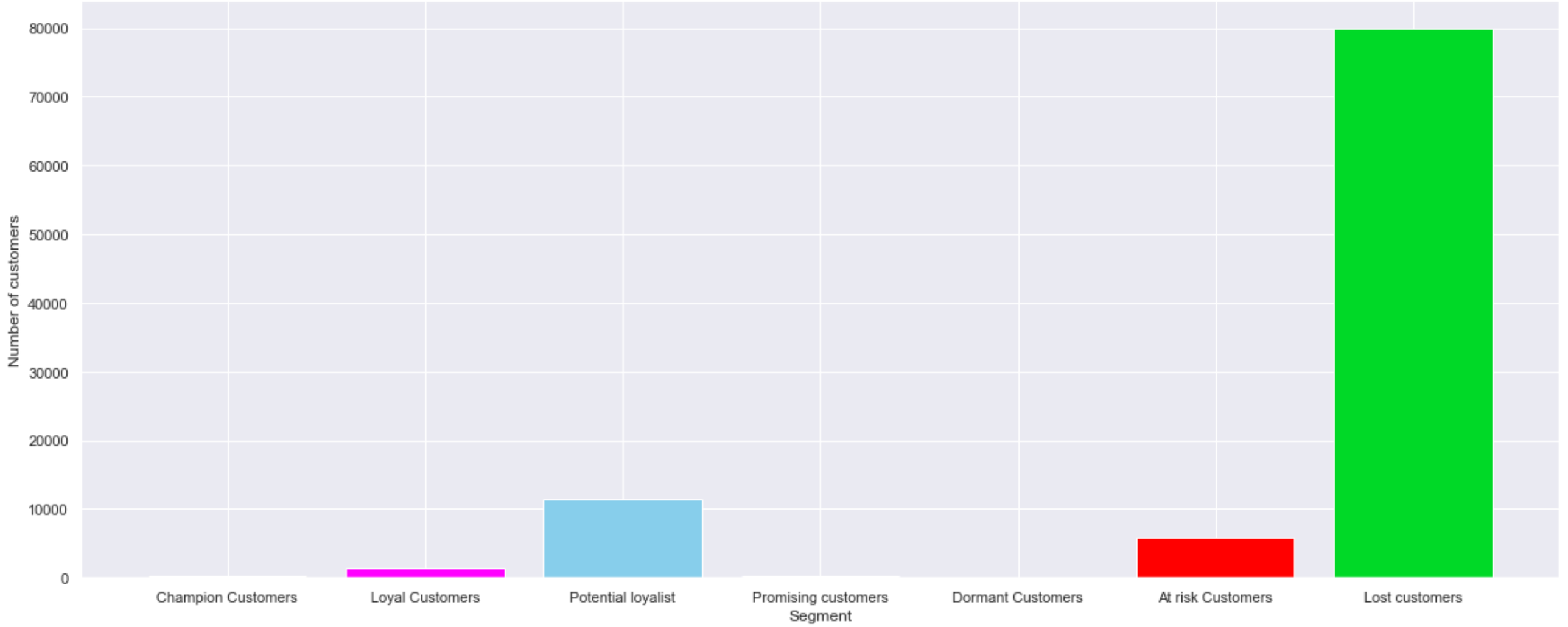
CORRELATION DANS LA TABLE RFM

- **Pas de corrélation entre les variables RFM.**



VISUALISATION DE LA SEGMENTATION RFM

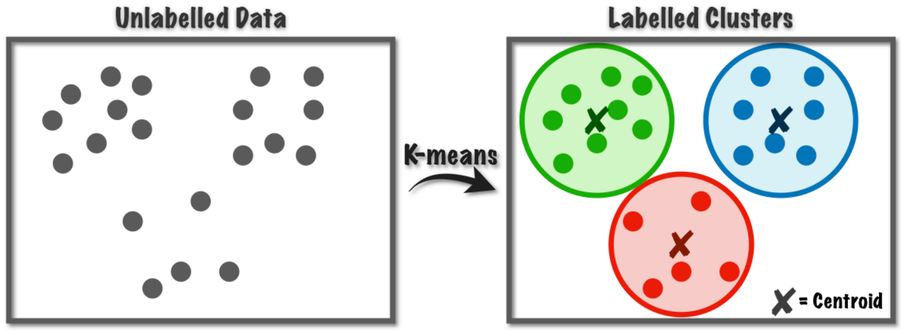
- **Baucoup de clients sont perdus malgré qu'il y a des clients qui sont des loyalistes potentiels.**



KMeans Clustering

Soit $X = \{x_1, x_2, \ldots, x_n\}$ un dataset, on cherche à partitionner les n points en k clusters $S = \{S_1, S_2, \ldots, S_k\}$, $k \leq n$ en minimisant la distance entre les points à l'intérieur de chaque partition :

$$argmin_S \sum_{i=1}^{k} \sum_{x_j \in S_i} \| x_j - c_i \|^2$$



- **Algorithme:**

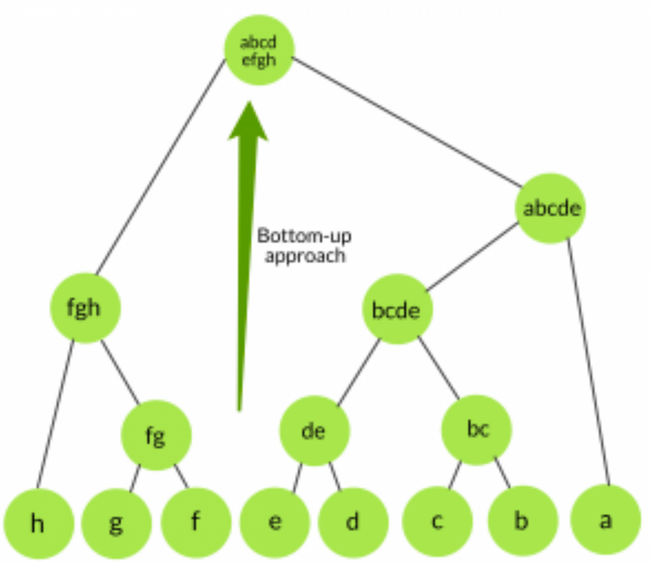
Étape 1 : Tout d'abord, choisir les centres de cluster ou le nombre de clusters.
Étape 2 : Affecter chaque point à son centre de cluster le plus proche en calculant la distance euclidienne.
Étape 3 : Les centroïdes du cluster seront optimisés en fonction de la moyenne des points attribués à ce cluster.
Étape 4 : Répéter les étapes 1, 2 et 3 jusqu'à ce que les centroïdes du cluster ne se déplacent plus, c-à-d que K-means a convergé.

KMedoids Clustering

Dans l'algorithme des *K-medoids*, le seul changement par rapport à *K-means* est le remplacement des centroides par des médoïdes. Le médoïde d'un groupe est l'élément pour lequel la somme des distances aux autres éléments du groupe est la plus faible, c-à-d le plus « central » du groupe.

ALGORITHMES DE MODELISATION : Agglomerative Clustering

Soit $X = \{x_1, x_2, \ldots, x_n\} \subset E$, un jeu de données. Le but est de réduire le nombre de clusters à $n_{cl} < n$, de façon itérative en fusionnant les deux clusters les plus proches,



Hierarchical agglomerative clustering

c-à-d la dissimilarité (distance en espace euclidien) entre eux est minimale.
De multiples critères permettent de calculer la dissimilarité, par exemple le *Clustering de Ward* où la distance entre deux clusters est celle qui minimise la variance intercluster donnée par:

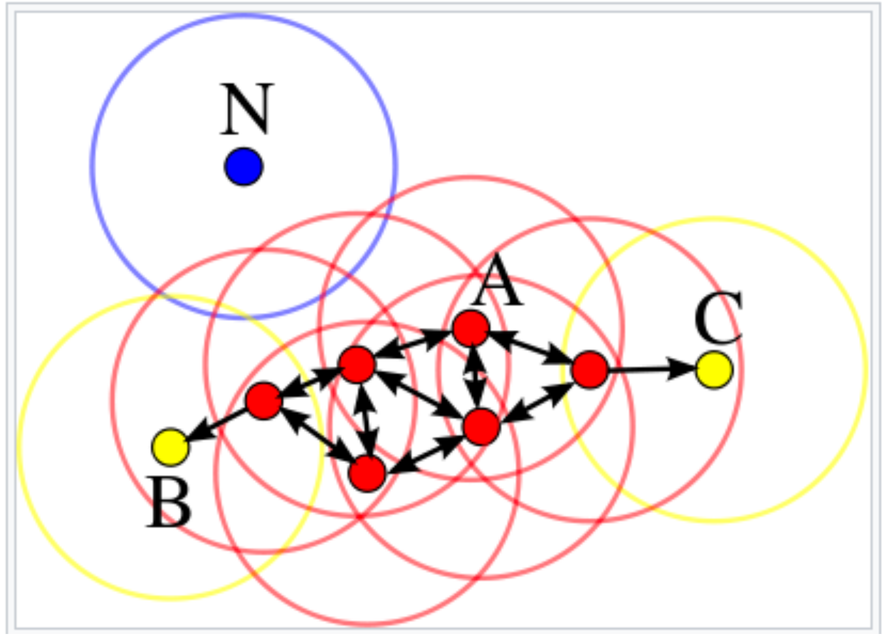
$$V = \sum_k \frac{1}{|C_k|} \sum_{x \in C_k} ||x - \mu||^2$$

- **Algorithme:**

Étape 1 : Considérer chaque point du jeu de données comme un cluster
Étape 2 : Calculer la dissimilarité (ou la similarité) d'un cluster avec tous les autres clusters (la matrice de proximité)
Étape 3 : Fusionner les clusters qui sont similaires ou proches les uns des autres.
Étape 4 : Recalculer la matrice de proximité pour chaque cluster
Étape 5 : Répéter les étapes 3 et 4 jusqu'à ce que tous les points appartiennent à un seul cluster, en agglomérant tous les clusters initiaux.

ALGORITHMES DE MODELISATION : DBSCAN Clustering

Soit E un espace doté d'une métrique d et $X = \{x_1, x_2, \ldots, x_n\} \subset E$, un jeu de données. Le ϵ-voisinage $V_\epsilon(x)$ du point $x \in E$ est : $V_\epsilon(x) = \{u \in E \,|\, d(x,u) < \epsilon\}$ **.**



DBSCAN requiert deux paramètres principaux :

- **ϵ : qui définit la taille du voisinage (le rayon de la boule)**
- **m = min_samples : qui définit la densité minimale à dépasser (le nombre de voisins dans la boule pour être considéré un point central).**
- **Algorithme:**

Étape 1 : DBSCAN sélectionne un point arbitraire aléatoire dans le jeu de données, puis se déplace vers tous les points d jeu de données.

Étape 2 : Si l'algorithme trouve qu'il y a des "min_samples" à une distance de eps (ϵ) du point choisi, l'algorithme considère que tous ces points font partie du même cluster.

Étape 3 : L'algorithme est ensuite répété pour les points de voisinage et les clusters sont ainsi élargis.

RESULTATS DE KMEANS CLUSTERING

RFM: les clients du cluster 2 sont ceux les plus dépensiers, mais ceux du cluster 0 ont acheté plus récemment.

```
cluster0.describe()
```

	Recency	Frequency	Monetary
count	49735.000000	49735.0	49735.000000
mean	127.704856	1.0	121.710404
std	72.361556	0.0	93.626466
min	0.000000	1.0	9.590000
25%	65.000000	1.0	56.930000
50%	129.000000	1.0	94.850000
75%	189.000000	1.0	157.300000
max	258.000000	1.0	604.730000

```
cluster1.describe()
```

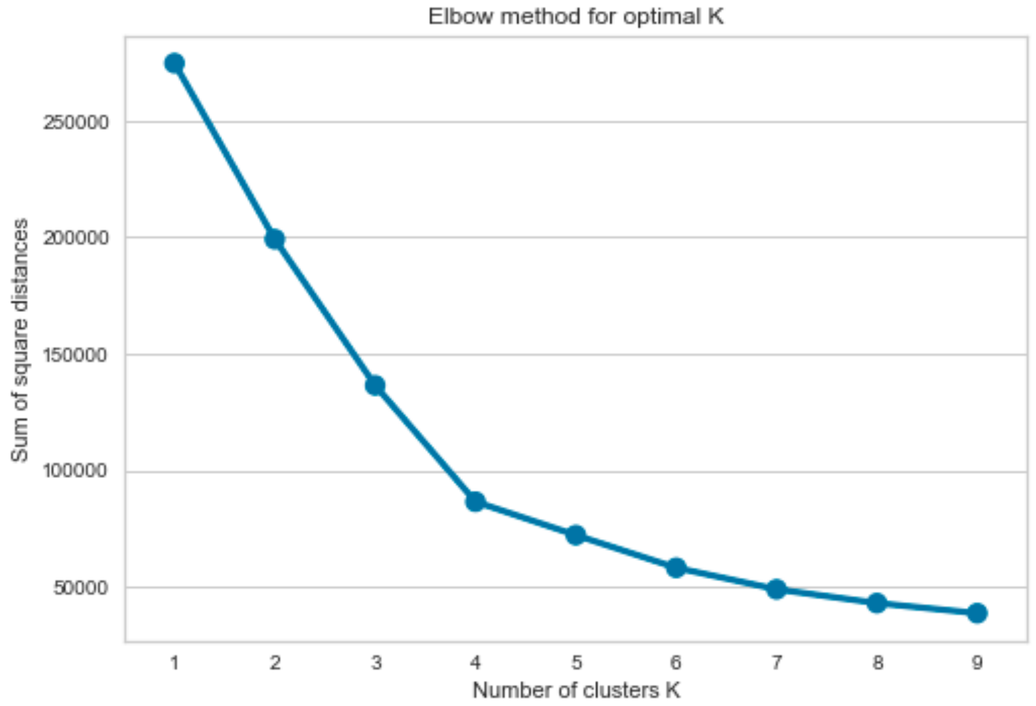
	Recency	Frequency	Monetary
count	36578.000000	36578.0	36578.000000
mean	387.182268	1.0	120.206667
std	95.940391	0.0	92.751923
min	258.000000	1.0	9.341429
25%	299.000000	1.0	56.750000
50%	374.000000	1.0	91.230000
75%	461.000000	1.0	153.017500
max	695.000000	1.0	671.310000

```
cluster2.describe()
```

	Recency	Frequency	Monetary
count	2487.000000	2487.000000	2487.000000
mean	240.581021	1.002010	1027.601267
std	151.523039	0.044802	549.658225
min	1.000000	1.000000	564.890000
25%	115.000000	1.000000	683.760000
50%	225.000000	1.000000	839.400000
75%	349.500000	1.000000	1172.450000
max	694.000000	2.000000	6929.310000

```
cluster3.describe()
```

	Recency	Frequency	Monetary
count	2721.000000	2721.000000	2721.000000
mean	219.969129	2.113561	121.155116
std	144.334422	0.507267	96.538446
min	1.000000	2.000000	14.976667
25%	104.000000	2.000000	64.098000
50%	198.000000	2.000000	94.577500
75%	317.000000	2.000000	146.365000
max	691.000000	15.000000	1034.340000

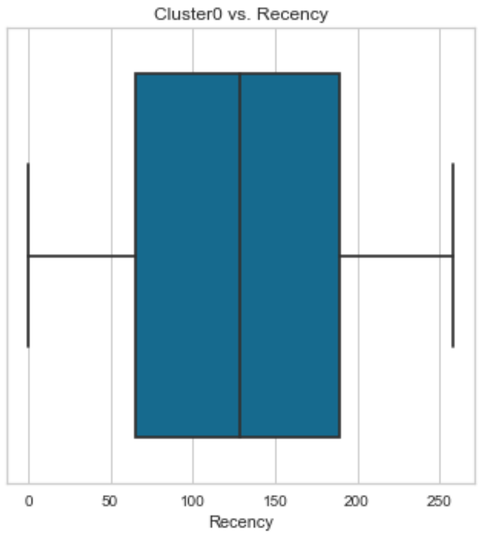


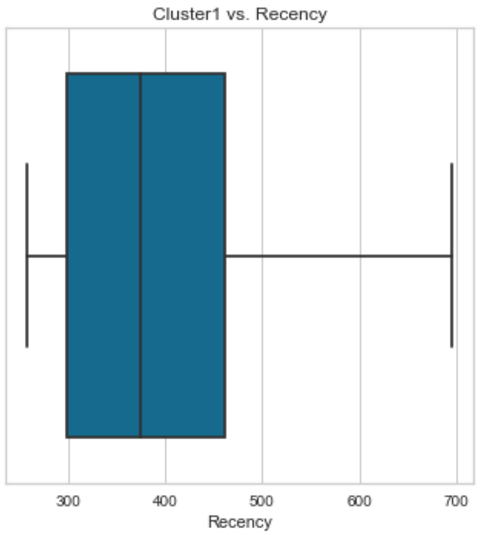

```
score = silhouette_score(rfm_scaled, kmeans.labels_, metric='euclidean')
#
kmeans_labels = kmeans.labels_

# Print the score
#
print('Silhouette Score: %.3f' % score)
```

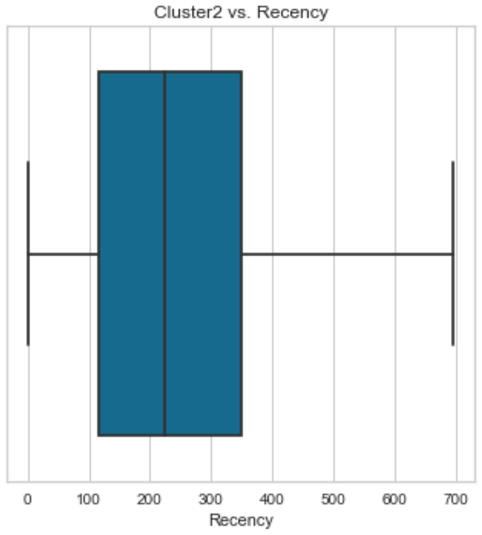
```
Silhouette Score: 0.489
```

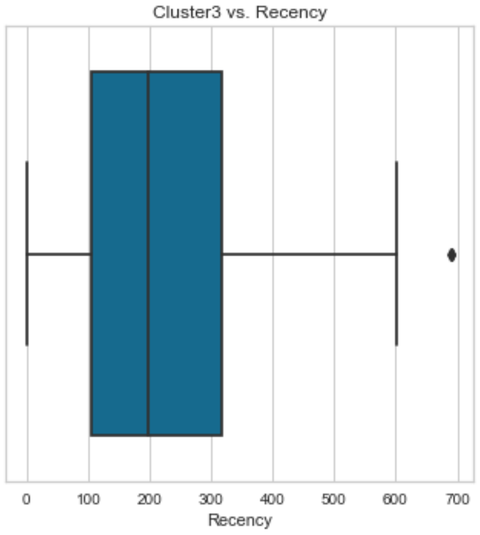
RESULTATS DE KMEANS CLUSTERING

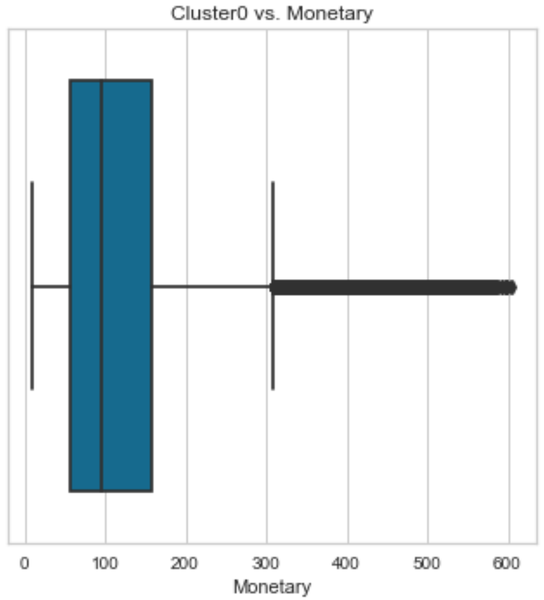
RFM: Des clients dépensiers apparaissent comme outliers dans tous les clusters.

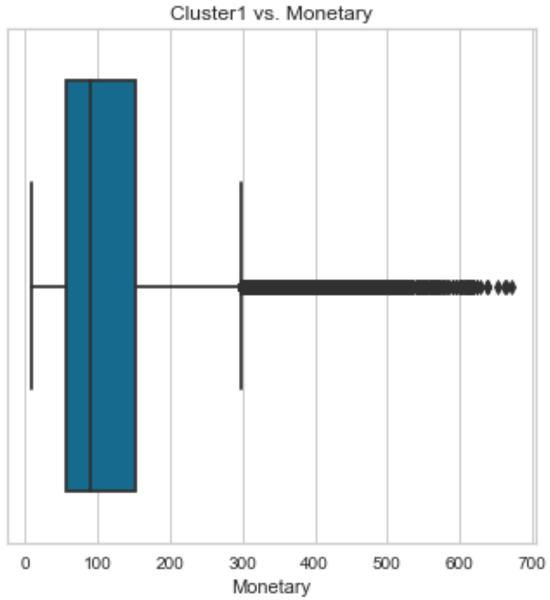


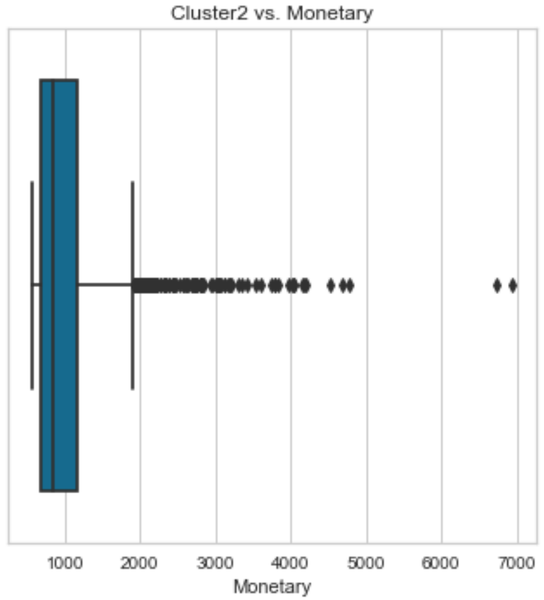


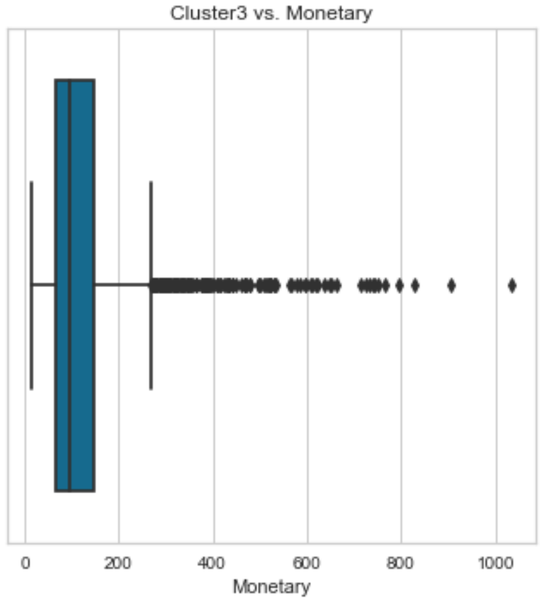








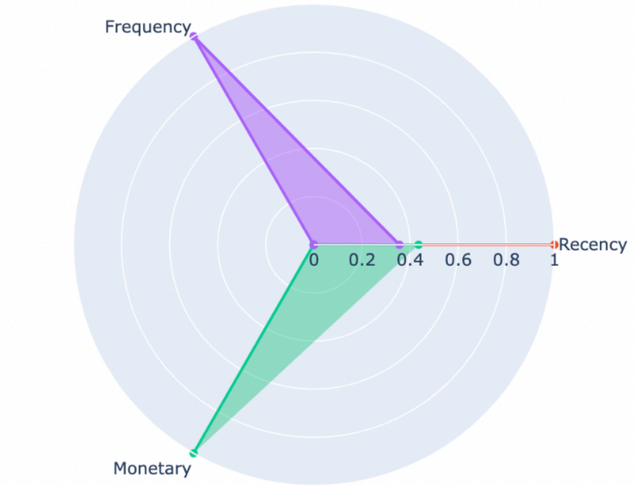




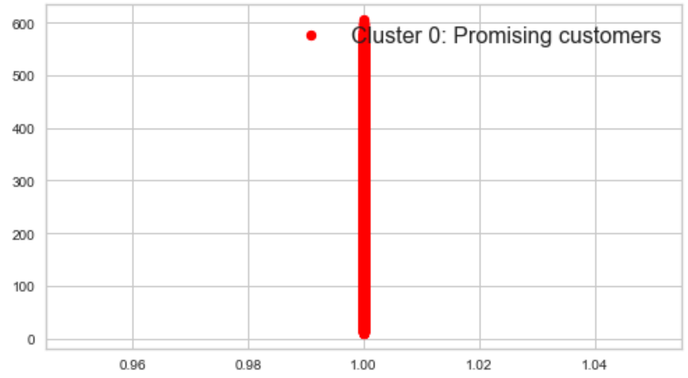
RESULTATS DE KMEANS CLUSTERING

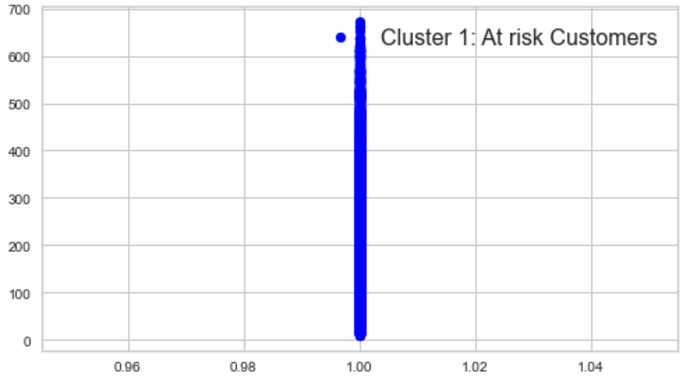
RFM: **les clients du cluster 3 font des achats plus fréquemment que les autres. alors que les clients du cluster 2 dépensent plus que les autres.**

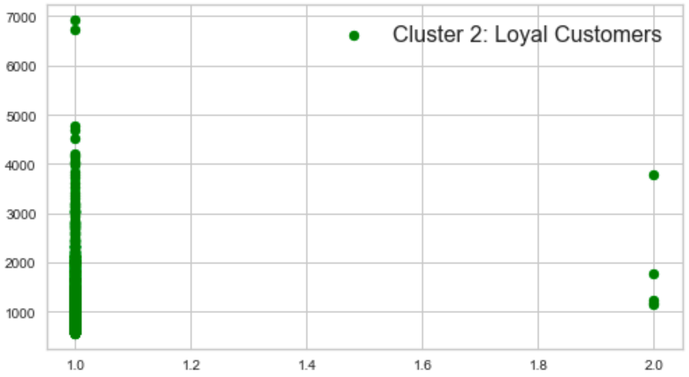
Comparaison des moyennes par variable des clusters

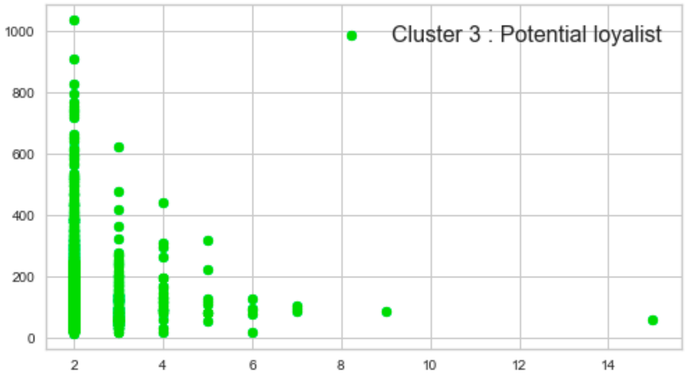


Frequency vs Monetary









COMPARAISON DES ALGORITHMES

Features considérées:

'Recency','Frequency','Monetary', 'mean_score','total_orders', 'delivery_duration'

KMeans	Agglomerative clustring	DBSCAN	KMedoids
n_clusters=4, X(90312, 6)	n_clusters = 4, sample(n=10000)	n_clusters = 8, X(90312, 6)	n_clusters = 4, sample(n=10000)
init='k-means++'	linkage = 'ward'	eps=1, min_samples=30	init='heuristic'
Silhouette score : 0.258	Silhouette score: 0.355	Silhouette score: 0.337	Silhouette score: 0.195

COMPARAISON DES ALGORITHMES

KMeans: les clients du cluster 3 font des achats plus fréquemment que les autres alors que les clients du cluster 0 dépensent plus que les autres.

```
]: cluster0.describe()
```

	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	30855.000000	30855.0	30855.000000	30855.000000	30855.000000	30855.000000
mean	391.272468	1.0	146.932641	4.584179	1.635618	10.902706
std	96.101063	0.0	205.277943	0.664987	1.656863	6.261009
min	254.000000	1.0	9.341429	1.000000	1.000000	1.000000
25%	304.000000	1.0	57.595000	4.000000	1.000000	6.000000
50%	379.000000	1.0	94.350000	5.000000	1.000000	10.000000
75%	465.000000	1.0	161.455000	5.000000	2.000000	14.000000
max	695.000000	1.0	6929.310000	5.000000	26.000000	59.000000

```
]: cluster1.describe()
```

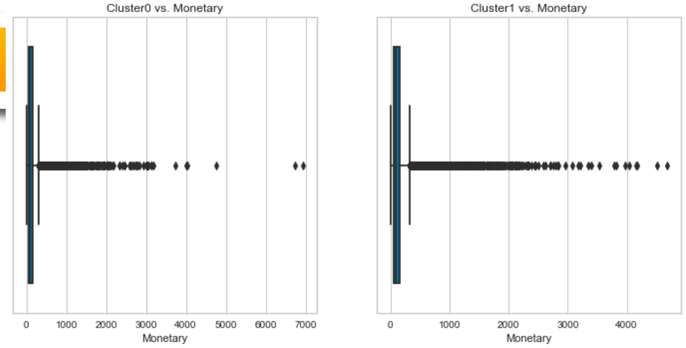
	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	42222.000000	42222.0	42222.000000	42222.000000	42222.000000	42222.000000
mean	127.666051	1.0	146.831299	4.610945	1.612406	10.682843
std	72.917033	0.0	196.121942	0.627048	1.553542	6.068746
min	0.000000	1.0	9.590000	2.000000	1.000000	1.000000
25%	64.000000	1.0	57.722500	4.000000	1.000000	6.000000
50%	128.000000	1.0	97.775000	5.000000	1.000000	10.000000
75%	189.000000	1.0	163.790000	5.000000	2.000000	14.000000
max	262.000000	1.0	4681.780000	5.000000	26.000000	45.000000

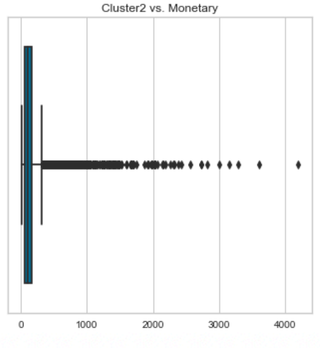
```
]: cluster2.describe()
```

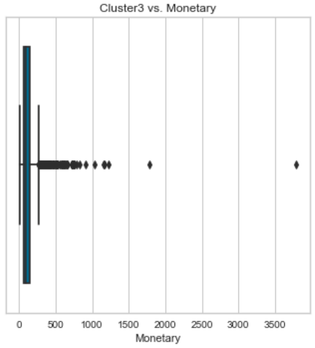
	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	15719.000000	15719.000000	15719.00000	15719.000000	15719.000000	15719.000000
mean	232.053566	1.000064	144.01460	1.622214	2.866404	20.445385
std	143.640170	0.007976	186.19911	0.912754	6.057301	15.318518
min	0.000000	1.000000	11.85000	1.000000	1.000000	1.000000
25%	119.000000	1.000000	57.38500	1.000000	1.000000	10.000000
50%	218.000000	1.000000	96.62000	1.000000	1.000000	17.000000
75%	326.000000	1.000000	162.95500	2.000000	2.000000	28.000000
max	694.000000	2.000000	4194.76000	5.000000	75.000000	208.000000

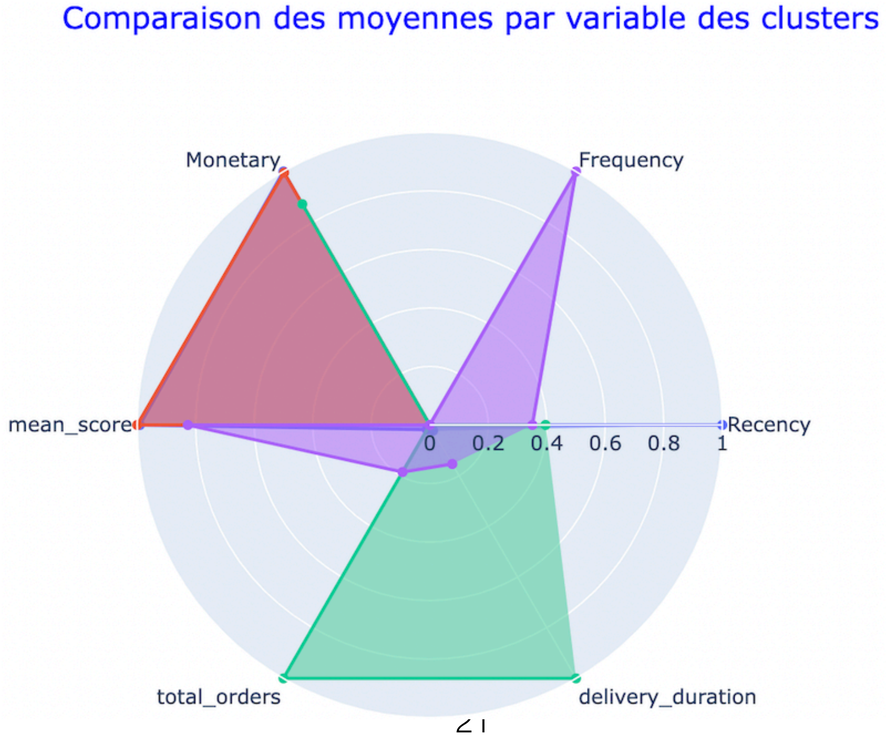
```
]: cluster3.describe()
```

	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	2725.000000	2725.000000	2725.000000	2725.000000	2725.000000	2725.000000
mean	220.160367	2.113394	124.260767	4.092364	1.845505	12.187890
std	144.409165	0.506913	128.357769	1.316369	2.867994	8.850898
min	1.000000	2.000000	14.976667	1.000000	1.000000	1.000000
25%	104.000000	2.000000	64.110000	4.000000	1.000000	6.000000
50%	198.000000	2.000000	94.705000	5.000000	1.000000	10.000000
75%	318.000000	2.000000	146.480000	5.000000	2.000000	15.000000
max	691.000000	15.000000	3785.815000	5.000000	75.000000	142.000000









COMPARAISON DES ALGORITHMES

Agglomerative clustering: **les différents clusters ont la même fréquence des achats mais les clients du cluster 1 dépensent un peu plus que les autres.**

```
cluster0.describe()
```

	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	7797.000000	7797.000000	7797.000000	7797.000000	7797.000000	7797.000000
mean	238.385148	1.037066	139.986603	4.070340	1.873669	12.531102
std	151.751388	0.265197	176.003510	1.337527	3.384512	9.486953
min	0.000000	1.000000	9.590000	1.000000	1.000000	1.000000
25%	115.000000	1.000000	57.850000	4.000000	1.000000	7.000000
50%	221.000000	1.000000	96.010000	5.000000	1.000000	10.000000
75%	343.000000	1.000000	160.760000	5.000000	2.000000	16.000000
max	695.000000	15.000000	4016.910000	5.000000	75.000000	145.000000

```
cluster1.describe()
```

	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	1923.000000	1923.000000	1923.000000	1923.000000	1923.000000	1923.000000
mean	234.693708	1.028081	143.933178	4.060668	1.910036	12.248570
std	152.757823	0.174438	191.213004	1.341348	3.509126	9.592726
min	1.000000	1.000000	10.070000	1.000000	1.000000	1.000000
25%	107.000000	1.000000	54.525000	4.000000	1.000000	7.000000
50%	219.000000	1.000000	94.720000	5.000000	1.000000	10.000000
75%	338.000000	1.000000	160.650000	5.000000	2.000000	15.000000
max	694.000000	3.000000	2787.870000	5.000000	75.000000	186.000000

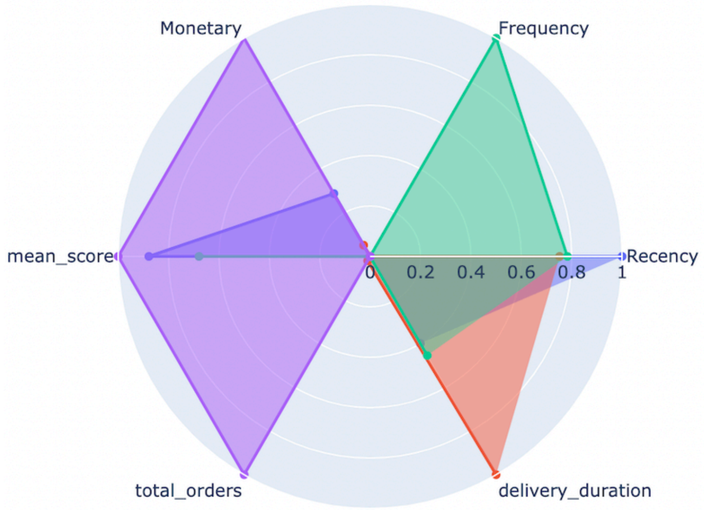
```
cluster2.describe()
```

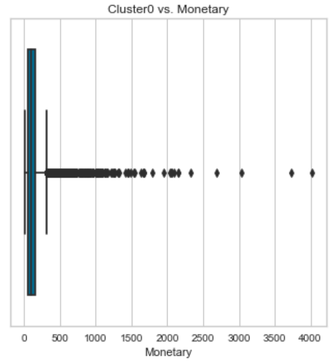
	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	269.000000	269.000000	269.000000	269.000000	269.000000	269.000000
mean	226.063197	1.018587	126.962776	4.149274	2.334572	12.334572
std	148.113358	0.135314	117.897888	1.262127	6.719844	9.804029
min	3.000000	1.000000	17.380000	1.000000	1.000000	1.000000
25%	111.000000	1.000000	56.050000	4.000000	1.000000	7.000000
50%	199.000000	1.000000	93.270000	5.000000	1.000000	9.000000
75%	310.000000	1.000000	157.060000	5.000000	2.000000	15.000000
max	692.000000	2.000000	958.390000	5.000000	75.000000	86.000000

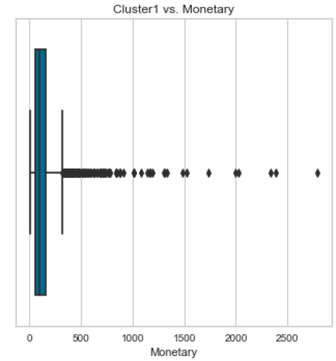
```
cluster3.describe()
```

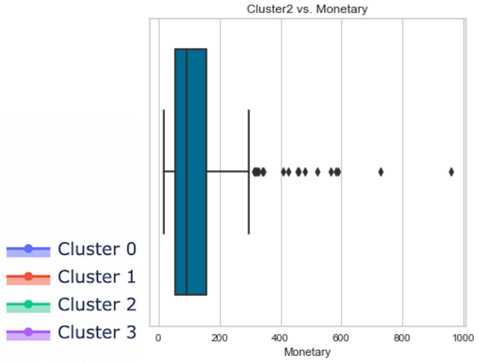
	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	11.000000	11.000000	11.000000	11.000000	11.000000	11.000000
mean	231.000000	1.090909	99.253636	4.090909	1.363636	13.454545
std	169.419007	0.301511	60.122159	1.578261	0.674200	12.250417
min	35.000000	1.000000	23.680000	1.000000	1.000000	4.000000
25%	81.500000	1.000000	61.375000	4.000000	1.000000	6.500000
50%	227.000000	1.000000	68.610000	5.000000	1.000000	10.000000
75%	331.500000	1.000000	132.755000	5.000000	1.500000	16.500000
max	564.000000	2.000000	213.520000	5.000000	3.000000	47.000000

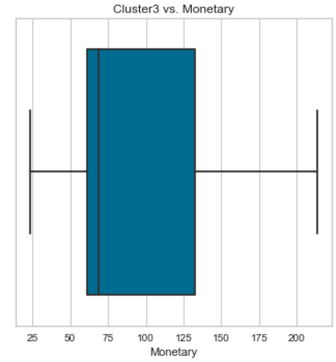
Comparaison des moyennes par variable des clusters











COMPARAISON DES ALGORITHMES

DBSCAN clustering: **les clients du cluster 3 ont des achats plus fréquents mais les clients du cluster 0 dépensent un peu plus que les autres.**

Estimated number of noise points: 1534

```
: cluster0.describe()
```

	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	42004.000000	42004.0	42004.00000	42004.000000	42004.000000	42004.000000
mean	127.747167	1.0	140.00547	4.610761	1.581230	10.674055
std	72.910294	0.0	154.59808	0.627117	1.360062	6.062512
min	0.000000	1.0	9.59000	2.000000	1.000000	1.000000
25%	64.000000	1.0	57.61375	4.000000	1.000000	6.000000
50%	128.000000	1.0	97.38500	5.000000	1.000000	10.000000
75%	189.000000	1.0	163.06000	5.000000	2.000000	14.000000
max	262.000000	1.0	1839.05000	5.000000	15.000000	45.000000

```
]: cluster1.describe()
```

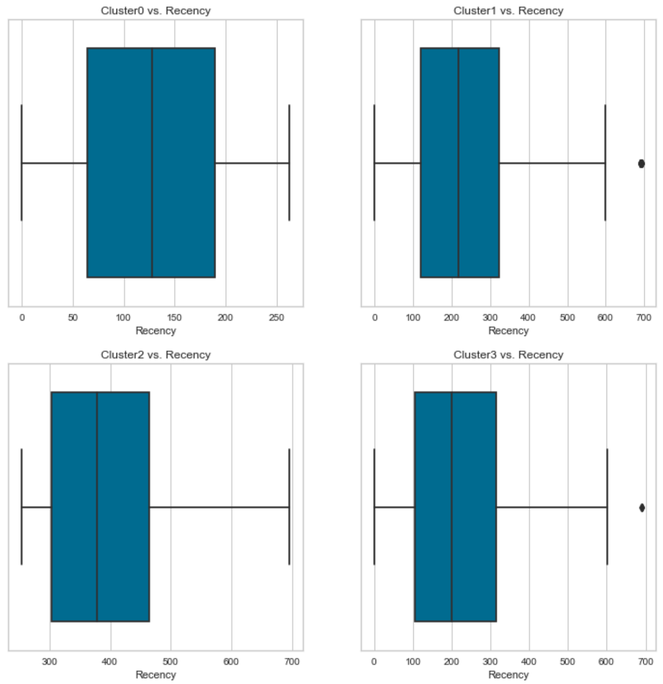
	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	14857.000000	14857.0	14857.000000	14857.000000	14857.000000	14857.000000
mean	231.207983	1.0	124.545994	1.568086	2.121222	19.775863
std	142.350877	0.0	103.567600	0.830962	1.959311	13.009435
min	0.000000	1.0	11.850000	1.000000	1.000000	1.000000
25%	119.000000	1.0	56.770000	1.000000	1.000000	10.000000
50%	218.000000	1.0	94.310000	1.000000	1.000000	17.000000
75%	324.000000	1.0	157.410000	2.000000	2.000000	27.000000
max	694.000000	1.0	836.700000	5.000000	15.000000	84.000000

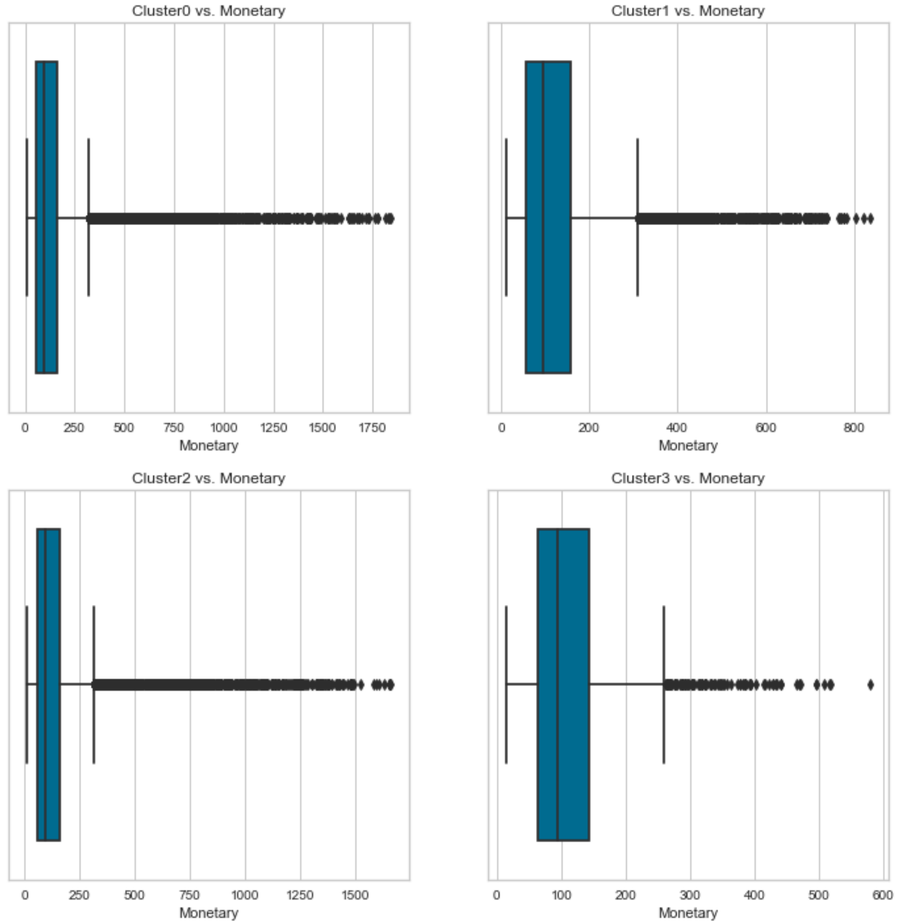
```
]: cluster2.describe()
```

	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	30635.000000	30635.0	30635.000000	30635.000000	30635.000000	30635.000000
mean	390.987629	1.0	138.589951	4.584977	1.591807	10.884054
std	95.922012	0.0	153.946082	0.663412	1.401188	6.237399
min	254.000000	1.0	9.341429	1.000000	1.000000	1.000000
25%	304.000000	1.0	57.510000	4.000000	1.000000	6.000000
50%	379.000000	1.0	94.050000	5.000000	1.000000	10.000000
75%	465.000000	1.0	159.560000	5.000000	2.000000	14.000000
max	695.000000	1.0	1657.110000	5.000000	15.000000	52.000000

```
: cluster3.describe()
```

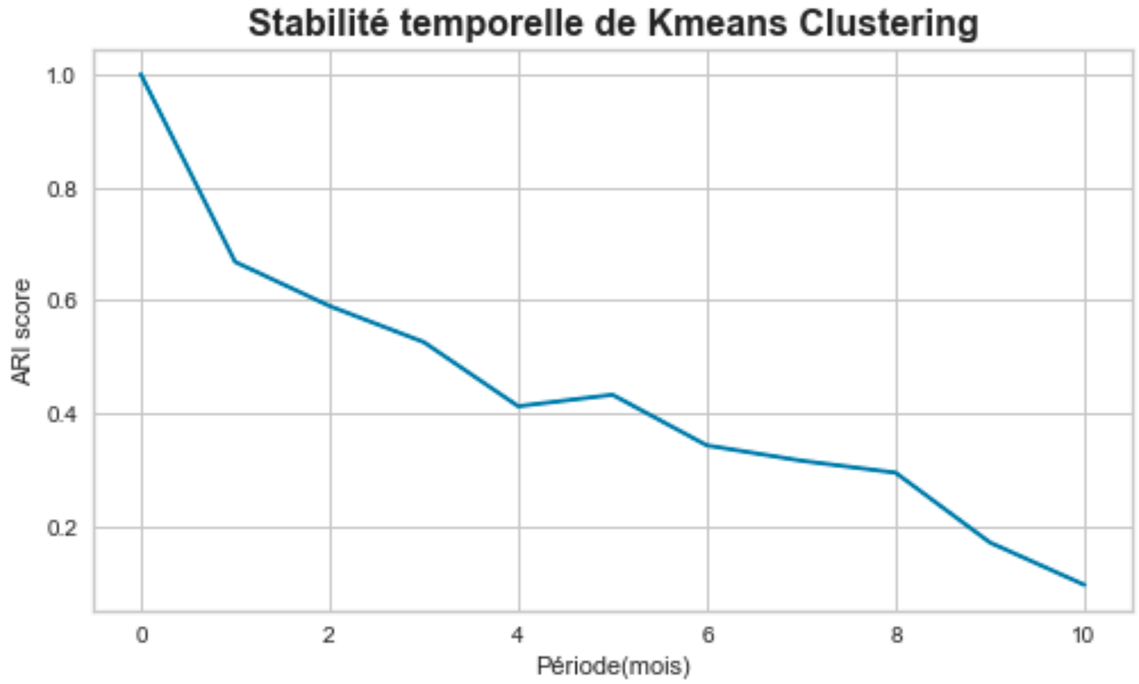
	Recency	Frequency	Monetary	mean_score	total_orders	delivery_duration
count	2294.000000	2294.0	2294.000000	2294.000000	2294.000000	2294.000000
mean	218.668265	2.0	112.693490	4.202623	1.524847	11.061465
std	141.063815	0.0	72.650076	1.225604	1.028186	6.396082
min	1.000000	2.0	14.976667	1.000000	1.000000	1.000000
25%	106.000000	2.0	63.675000	4.000000	1.000000	6.000000
50%	199.000000	2.0	93.760000	5.000000	1.000000	10.000000
75%	314.000000	2.0	142.542500	5.000000	2.000000	14.000000
max	691.000000	2.0	579.293333	5.000000	8.000000	38.000000

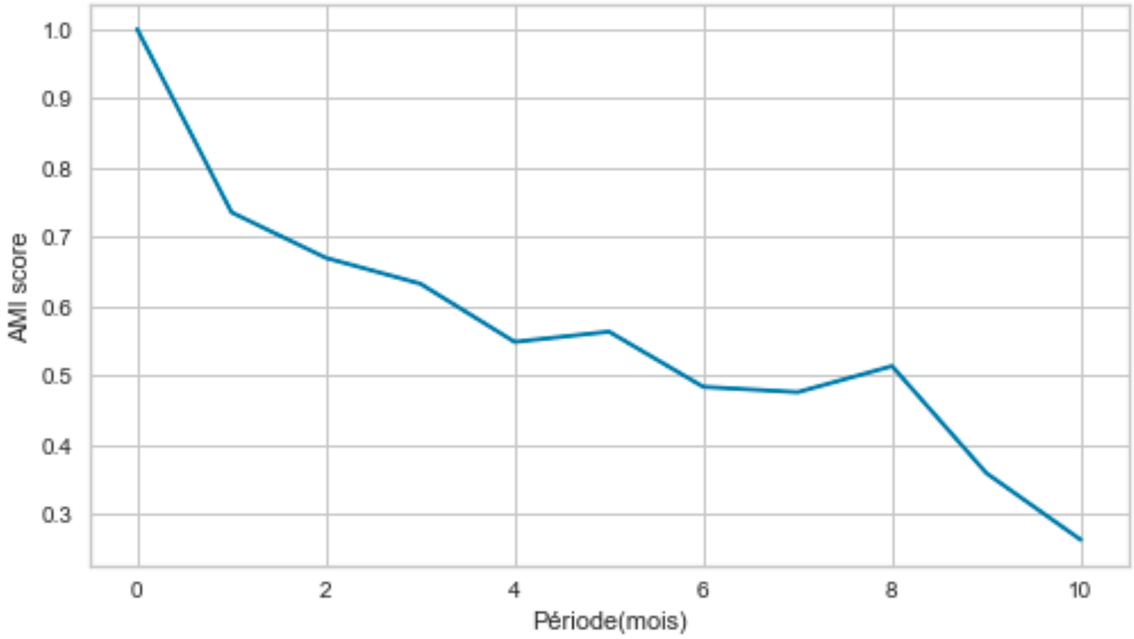




ETUDE TEMPORELLE DE KMEANS

KMeans clustering: **on note qu'après un mois environ il y a une chute du nombre de clients faisant des achats en ligne sur le site de Olist et ARI score est environ 0.67. Donc, un maintenance mensuelle est recommandée.**





CONCLUSIONS

Approche analytique:

- La segmentation RFM permet d'identifier les différents types de clients, mais son analyse est essentiellement basée sur la satisfaction des clients
- L'application de la méthode RFM serait plus pertinente avec un dataset plus réaliste où la majorité des clients font plus qu'une commande.

Approche automatique:

- Clustering avec variables RFM: Dans le cas où les variables considérées sont R, F, M, l'algorithme Kmeans donne un bon partitionnement des clients en 4 clusters et un silhouette score d'environs 0.49

- Clustering avec augmentation des variables : en ajoutant les variables mean_score, total_orders, delivery_duration aux variables précédentes, nous avons remarqué:
- Kmeans: ses résultats sont meilleurs que les autres algorithmes puisqu'il permet de partitionner le dataset global en 4 clusters de tailles raisonnables mais avec un silhouette score en baisse d'environs 0.26.
- DBSCAN : partitionne le dataset global en 8 clusters avec certaines tailles très petite et un cluster vide. Il présente des clients comme outliers n'appartenant à aucun cluster.
- Agglomerative clustering et KMedoids : ne marchent que pour un échantillon des clients (10000). Les deux algorithmes donnent un clustering en 4 clusters avec des silhouette score d'environs 0.36 et 0.20 respectivement.
- Etude temporelle de Kmeans: pour la stabilité temporelle, une maintenance mensuelle est conseillée.