Projet 5 - Segmentez des clients d'un site e-commerce

Fournir à l'équipe marketing une description actionable de la segmentation et de sa logique sous-jacente pour une utilisation optimale, ainsi qu'une proposition de contrat de maintenance basée sur une analyse de la stabilité des segments au cours du temps.

Sommaire

Problématique & Données

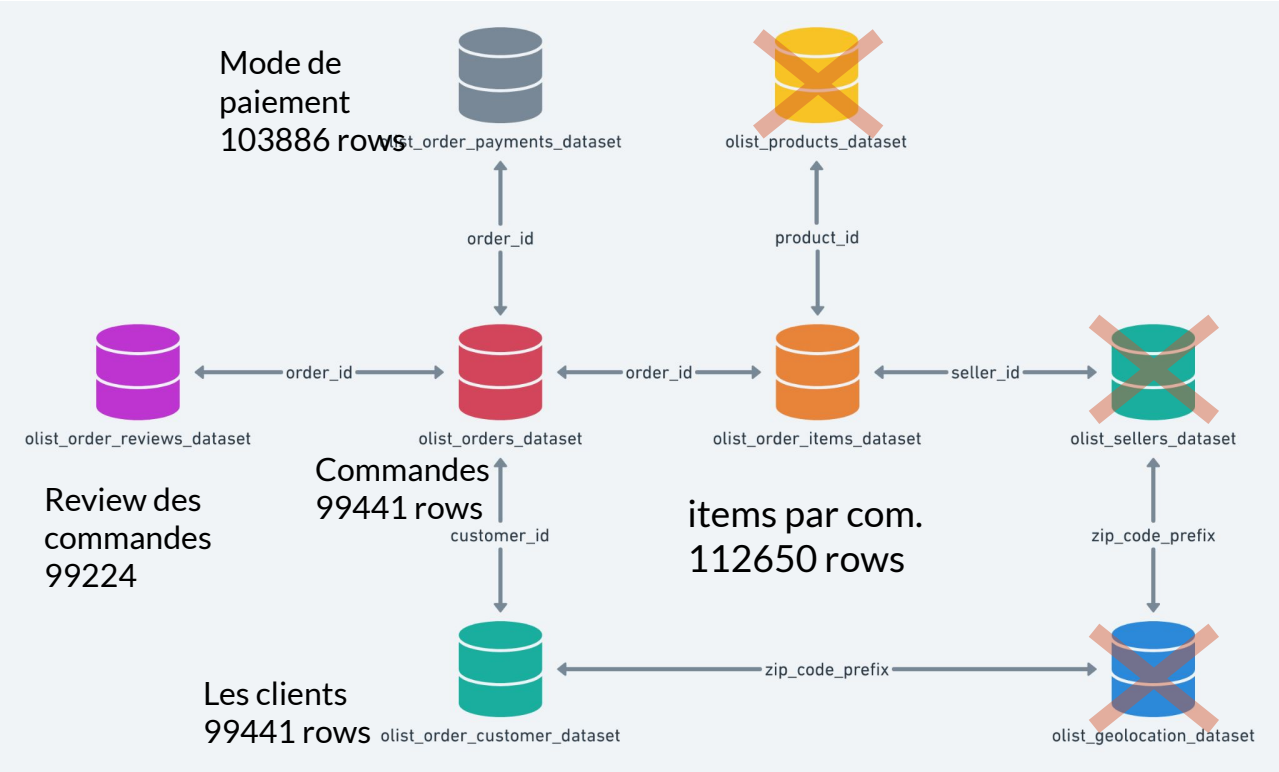
La problématique

Contexte : Olist, marketplace brésilienne, a besoin d'une segmentation de ses clients pour ses campagnes de communication.

Objectif : Comprendre les différents types d'utilisateurs et proposer une description actionnable de la segmentation ainsi qu'une proposition de contrat de maintenance.

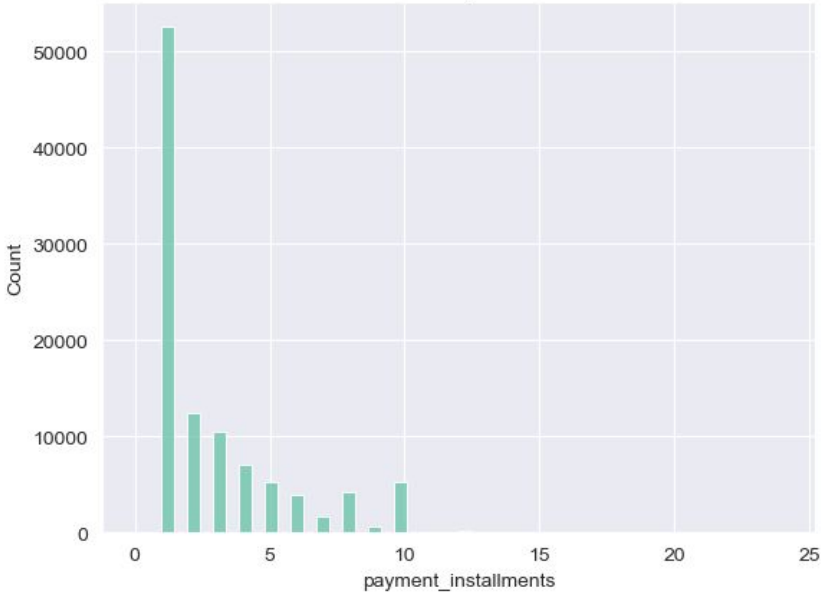
Moyens : Une base de données anonymisée fournie par Olist

La base de données - Schéma global

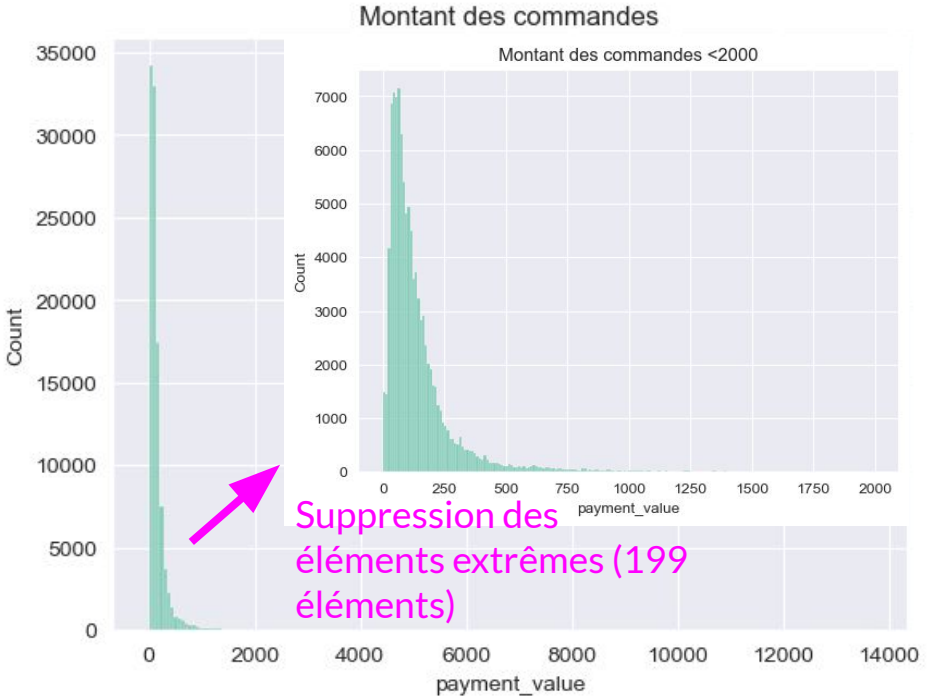


La base de données - Les paiements

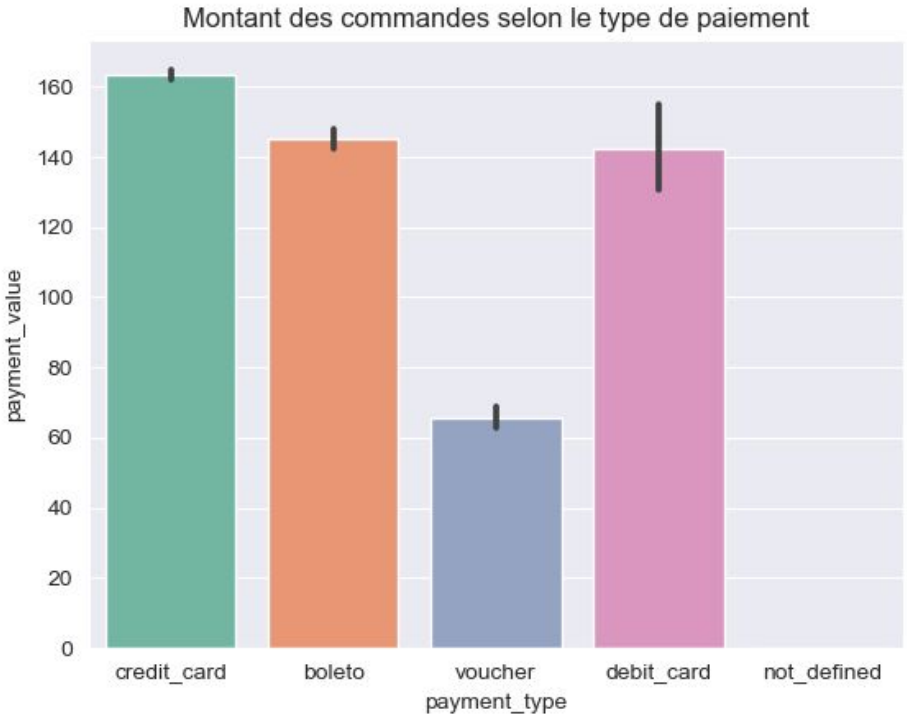
Nombre de paiements

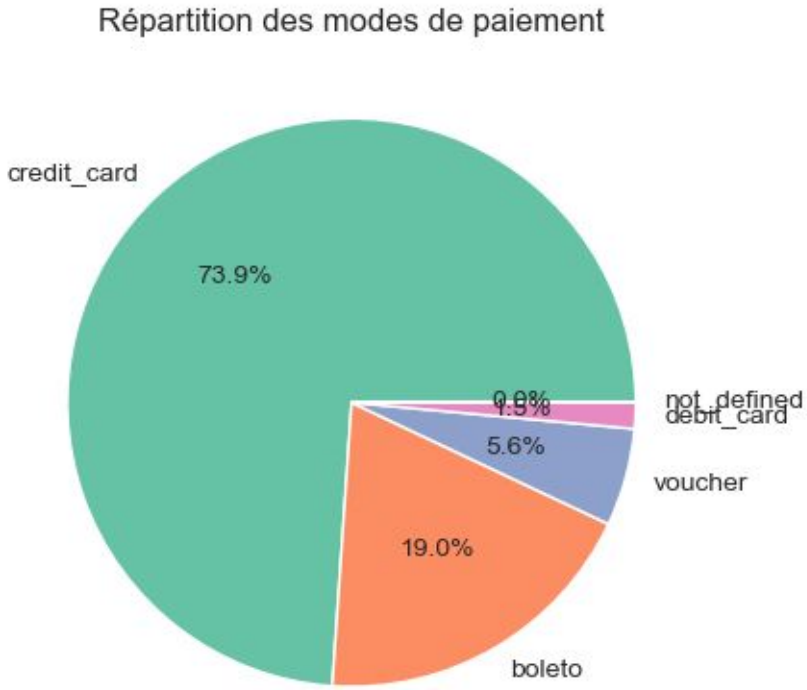


Montant des paiements

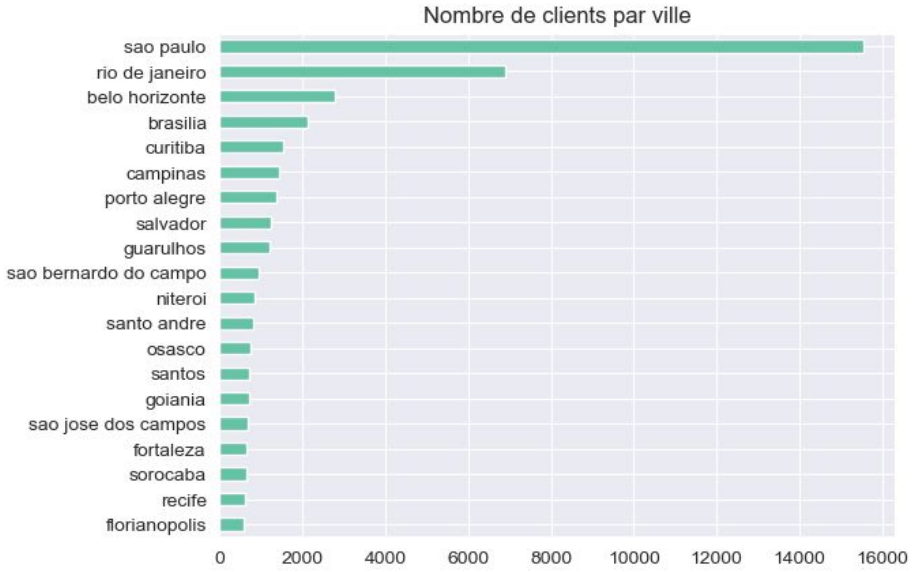


La base de données - Les types de paiements

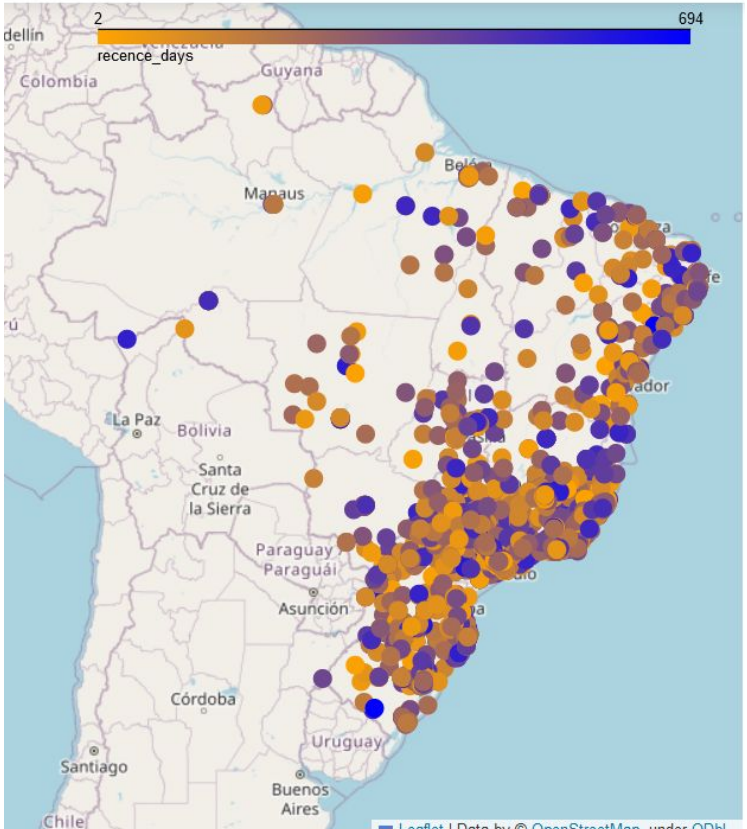




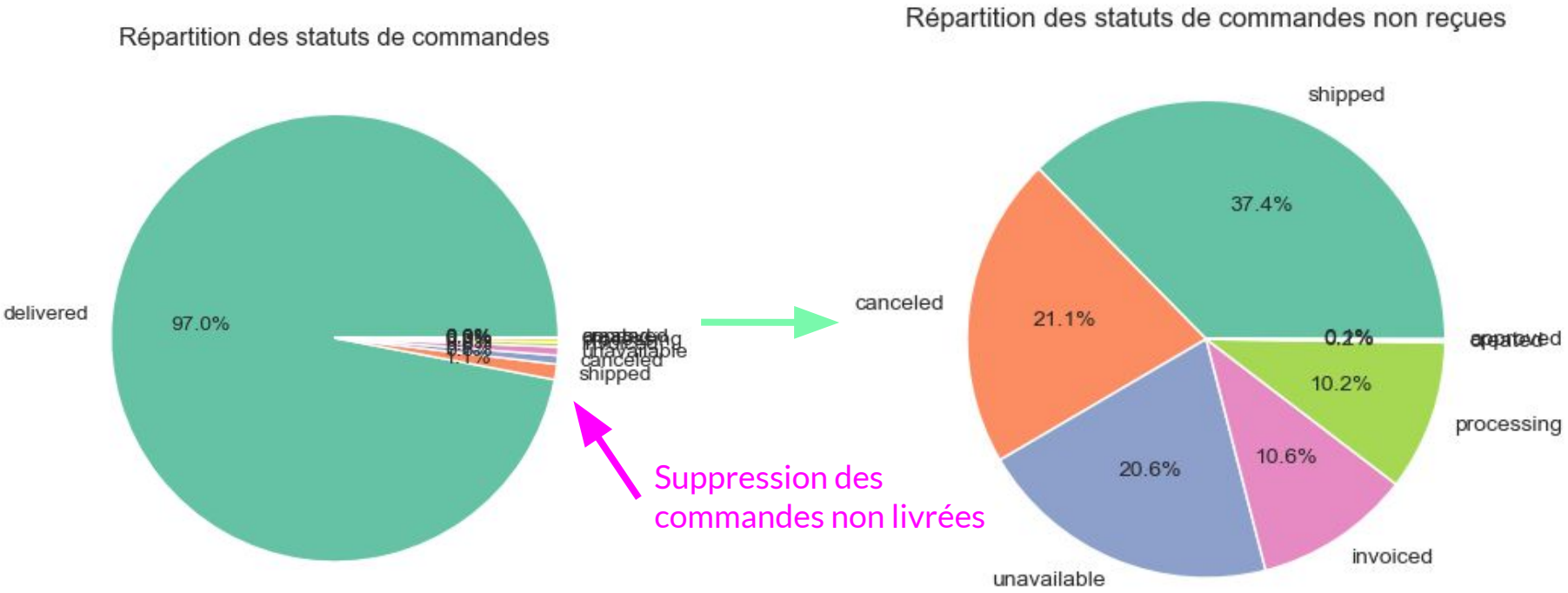
La base de données - Géolocalisations



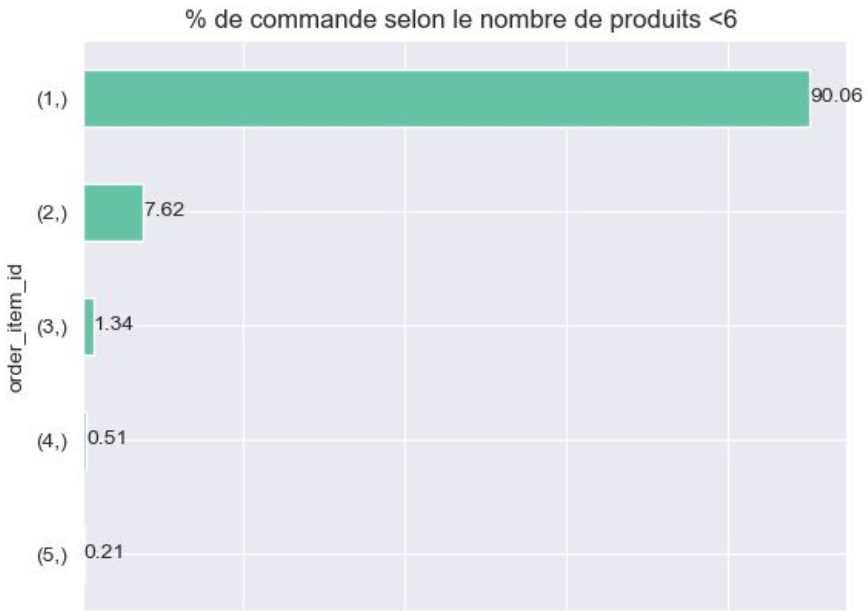
Pas d'utilisation des données de géolocalisation



La base de données - Statuts des commandes

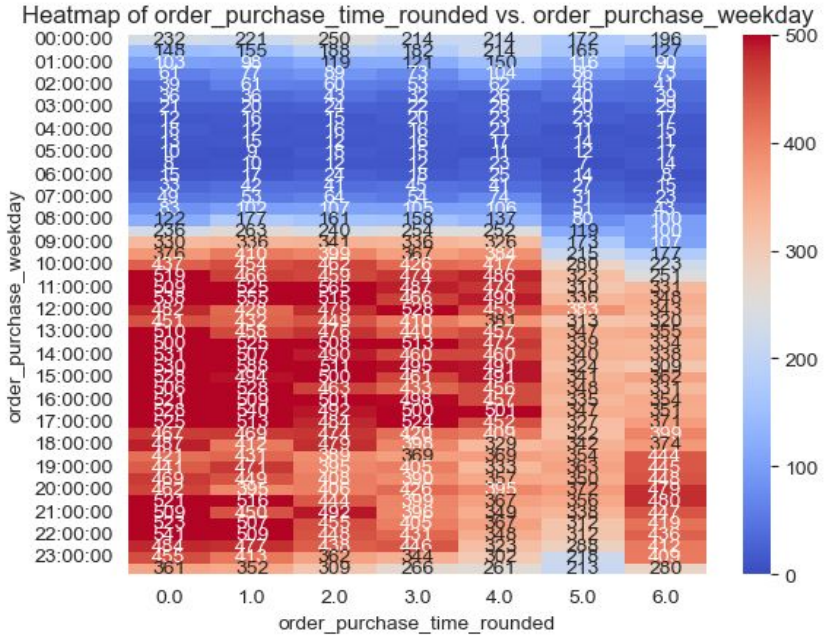


La base de données - Nb de produits par commande et nb de commandes par client



La plupart des commandes n'ont qu'un seul produit, et la plupart des clients n'ont fait qu'une seule commande. Abandon des catégories de produits.
Binarisation avec création de la variable 'more_than_one_order'

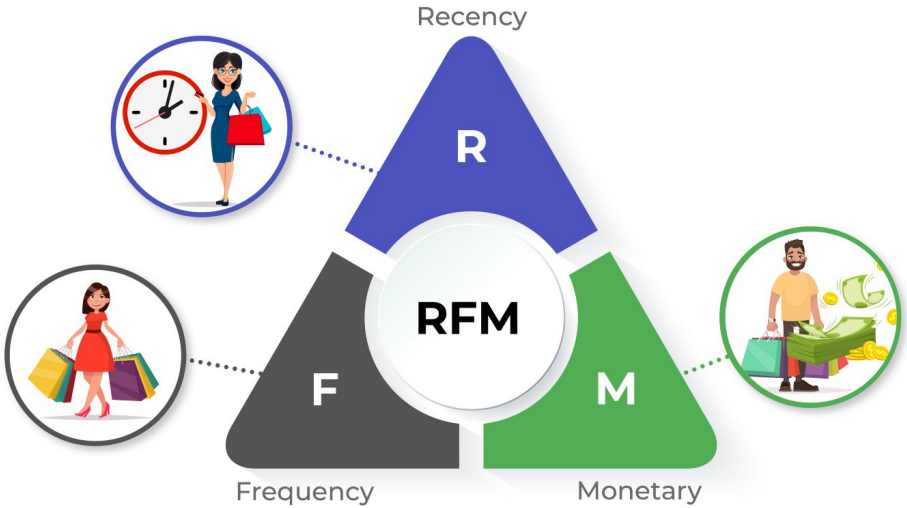
La base de données - Heures et Jours des commandes



Abandon de l'idée de conserver une segmentation en fonction des habitudes d'heures et jours de commande.

Segmentation manuelle - RFM

Segmentation classique - RFM



La segmentation RFM est une technique qui permet de classer les clients en fonction de leur comportement d'achat, en utilisant les notions de **Récence, Fréquence et Montant.** Cette méthode permet d'optimiser les campagnes de marketing et la fidélisation client.

recence	nb_of_orders	total_payment	R	F	M	RFM
470 days	1	146.87	1	1	4	114
229 days	1	335.48	3	1	5	315
102 days	1	157.73	4	1	4	414
169 days	1	173.30	4	1	4	414
31 days	1	252.25	5	1	5	515
...	...	...	...	...	...	...
144 days	1	88.78	4	1	3	413
147 days	1	129.06	4	1	4	414
143 days	1	56.04	4	1	2	412
299 days	1	711.07	2	1	5	215
253 days	1	21.77	3	1	1	311

```
rfm['R']=pd.qcut(rfm.recence, q=5,
                 labels=[5,4,3,2,1])
rfm['F']=pd.cut(rfm.nb_of_orders, bins=5,
                 labels=[1,2,3,4,5])
rfm['M']=pd.qcut(rfm.total_payment, q=5,
                 labels=[1,2,3,4,5])
```

nb de combinaisons différentes : 73

Segmentation classique - RFM

moy de R 3.004
moy de F 1.003
moy de M : 2.999

```
rfm_['R']=rfm.R.apply(lambda x: 0 if x < 3 else 1)
rfm_['F']=rfm.F.apply(lambda x: 0 if x < 2 else 1)
rfm_['M']=rfm.M.apply(lambda x: 0 if x < 3 else 1)
```

```
rfm_dict= {'111':'Champions',
           '011':'Client_fidèles',
           '110':'Client_fidèles',
           '101':'Clients_récents',
           '100':'Clients_récents',
           '000':'Clients perdus',
           '001':'Clients en hibernation',
           '010':'Clients en hibernation'}
```

	Segments	Proportions
0	Clients_récents	0.600282
1	Clients en hibernation	0.233566
2	Clients perdus	0.164265
3	Client_fidèles	0.001151
4	Champions	0.000736

- **Champions** : Clients qui ont acheté plusieurs fois, récemment et avec un gros panier
- **Clients fidèles** : clients qui ont acheté plusieurs fois, soit récemment, soit un gros panier
- **Clients Récents** : clients ayant acheté récemment, une seule fois, soit un gros panier soit un petit peu importe.
- **Clients perdus** : clients n'ayant acheté qu'une seule fois, il y a longtemps, et un petit panier..
- **Client en hibernation** : client ayant acheté une fois un gros panier, ou plusieurs fois un petit panier.

Feature Engineering

Le Feature Engineering

Créer les variables

Stockage dernière date de la BDD

Stockage des fichiers de la BDD nécessaires à la construction des variables

Création de la liste des clients

'more_than_one_order'
= 0 si un seul order, 1 si plus d'un order

'recence_days'
jours depuis le dernier achat

'average_payment'
moyenne des paiements par commande

'credit_card_percentage'
info sur le mode de paiement avec le pourcentage du paiement effectué par CB

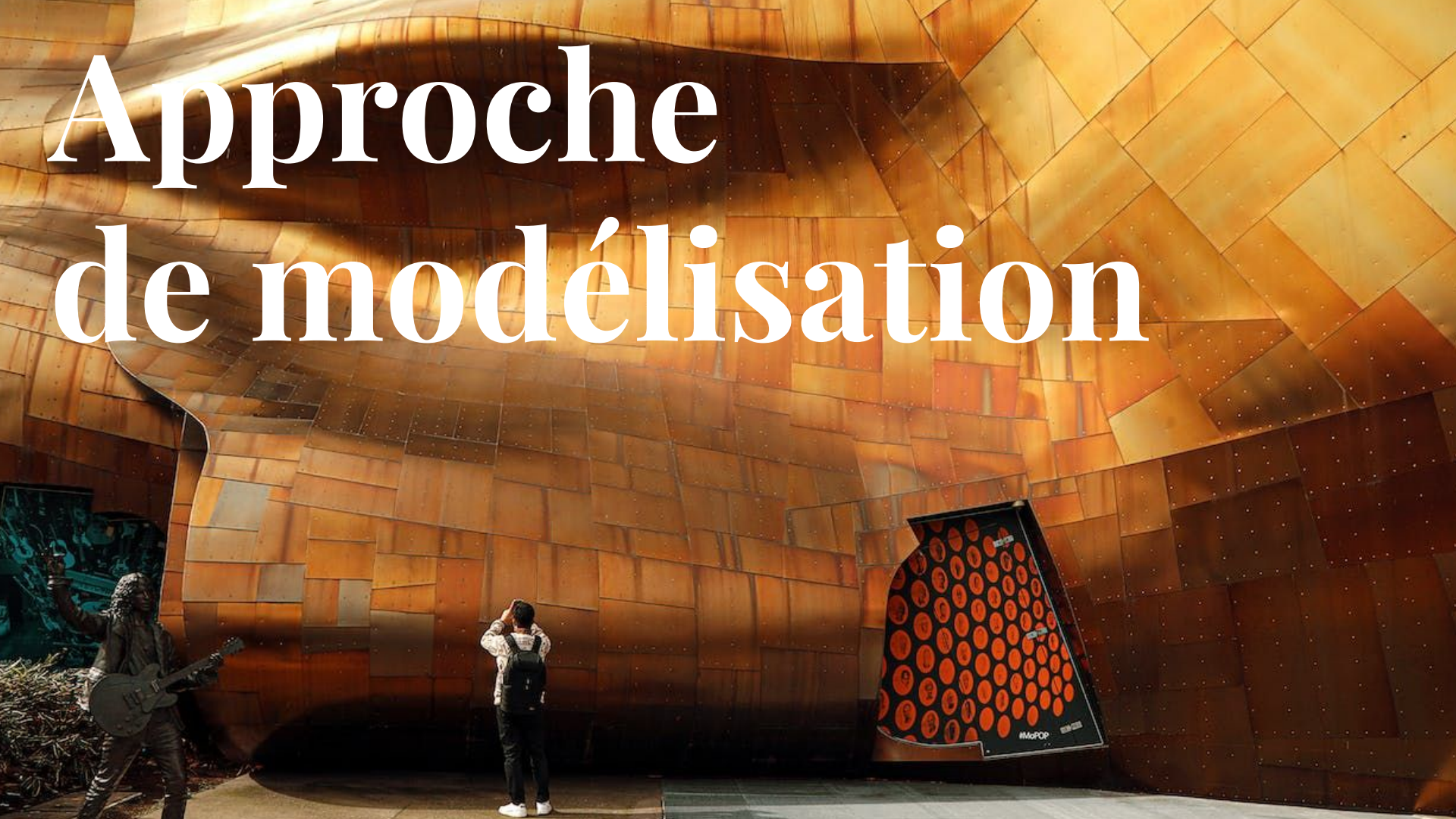
'review_fillna'
note attribuée dans la review ou absence de review (moyenne)

On enlève les éléments extrêmes

```
lastpurchasedata = pd.to_datetime(orders_import['order_purchase_timestamp'])
lastpurchasedata = lastpurchasedata.dt.date
last_purchase_date=lastpurchasedata.max()
```

```
def calcul_var(data) :
    orders=data[3]
    order_payments=data[1]
    order_reviews=data[2]
    customers=data[0]
    orders = orders[orders.order_status == 'delivered']
    listofcustomers=orders.customer_id.unique()
    customers = customers[customers.customer_id.isin(listofcustomers)]
    nborderbycustomer=orders['customer_id'].value_counts()
    nborderbycustomer = pd.DataFrame(nborderbycustomer)
    nborderbycustomer.reset_index(inplace=True)
    nborderbycustomer.rename(columns={'customer_id':'nb_of_orders','index':'customer_id'},inplace=True)
    nborderbycustomer['more_than_one_order']=nborderbycustomer['nb_of_orders'].apply(lambda x: 0 if x == 1 else 1)
    datacustomers=customers.customer_id.copy().to_frame()
    datacustomers = pd.merge(datacustomers,
                        nborderbycustomer[['customer_id','more_than_one_order']],
                        on = 'customer_id', how = 'left')
    dataorders=orders[['order_id','customer_id','order_purchase_timestamp']]
    dataorders['order_purchase_timestamp'] = pd.to_datetime(dataorders['order_purchase_timestamp'])
    dataorders['order_purchase_date']=dataorders['order_purchase_timestamp'].dt.date
    dataorders['recence']=last_purchase_date-dataorders['order_purchase_date']
    recenceparclient=dataorders[['customer_id','recence']].groupby(['customer_id']).min()
    recenceparclient.reset_index(inplace=True)
    datacustomers = pd.merge(datacustomers,
                        recenceparclient[['customer_id','recence']],
                        on = 'customer_id', how = 'left')
    order_payments=order_payments[order_payments.payment_type != 'not_defined']
    order_payments.replace('debit_card','credit_card',inplace=True)
    order_payments_ = pd.pivot_table(order_payments, values='payment_value', index='order_id', columns='payment_type', aggfunc='su
    order_payments_['total_payment'] = order_payments_.sum(axis=1)
    order_payments_['credit_card_percentage'] = order_payments_['credit_card'] / order_payments_['total_payment']
    order_payments_.reset_index(inplace=True)
    dataorders = pd.merge(dataorders,
                order_payments_,
                on = 'order_id', how = 'left')
    infocommoyparclient=dataorders[['customer_id','total_payment','credit_card_percentage']].groupby(['customer_id']).mean()
    infocommoyparclient.reset_index(inplace=True)
    datacustomers = pd.merge(datacustomers,
                        infocommoyparclient,
                        on = 'customer_id', how = 'left')
    datacustomers.rename(columns={'total_payment':'average_payment'},inplace=True)
    dataorders = pd.merge(dataorders,
                order_reviews[['order_id','review_score']],
                on = 'order_id', how = 'left')
    inforeviewparclient=dataorders[['customer_id','review_score']].groupby(['customer_id']).mean()
    datacustomers = pd.merge(datacustomers,
                        inforeviewparclient,
                        on = 'customer_id', how = 'left')
    datacustomers['recence_days'] = round(datacustomers['recence'].dt.days)
    datacustomers['review_fillna']=datacustomers['review_score'].fillna(datacustomers.review_score.mean())
    data_return=datacustomers[['customer_id','more_than_one_order','recence_days','average_payment','review_fillna','credit_card_p
    data_return.dropna(inplace=True)
    data_return=data_return[data_return['average_payment']<2000]
    data_return.set_index('customer_id',inplace=True)
    return data_return
```

Approche de modélisation



Approche de modélisation - Méthodes & Contraintes

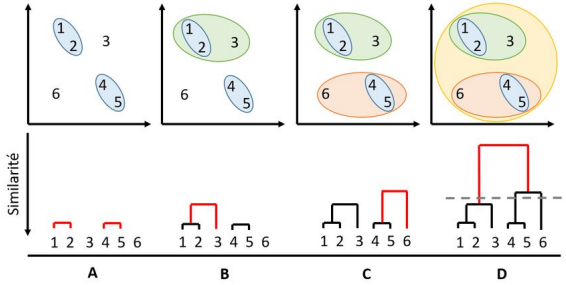
- Utilisation du feature engineering présenté
- Algorithme de clustering non supervisé
- Comparaison des modèles via le silhouette score
- Temps d'entraînement du modèle à prendre en compte
- Réutilisation du modèle à prendre en compte
- Le nombre de clusters doit être suffisamment élevé pour être actionnable
- Trouver les meilleurs hyperparamètres

Approche de modélisation

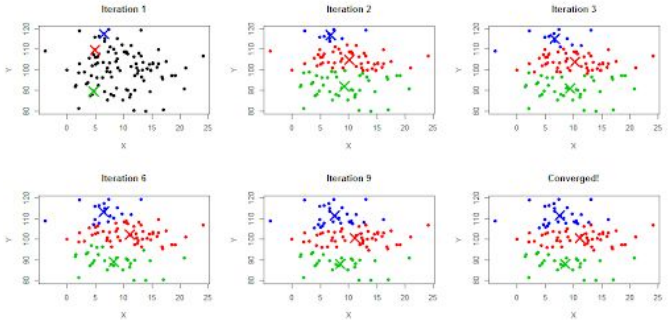
Méthodes testées pour segmenter :

MESURE DE DISTANCE ENTRE LES INDIVIDUS

CAH (approche hiérarchique , tous les clusters sont représentés)

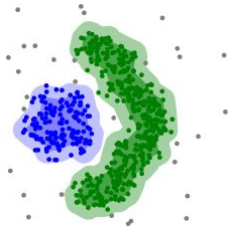


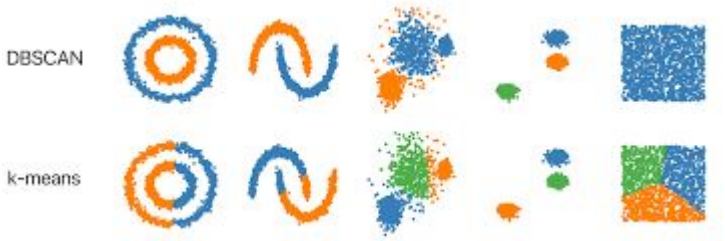
Kmeans (partitionnement, nombre de clusters fixés)



MESURE DE DENSITÉ DES INDIVIDUS DANS L'ESPACE

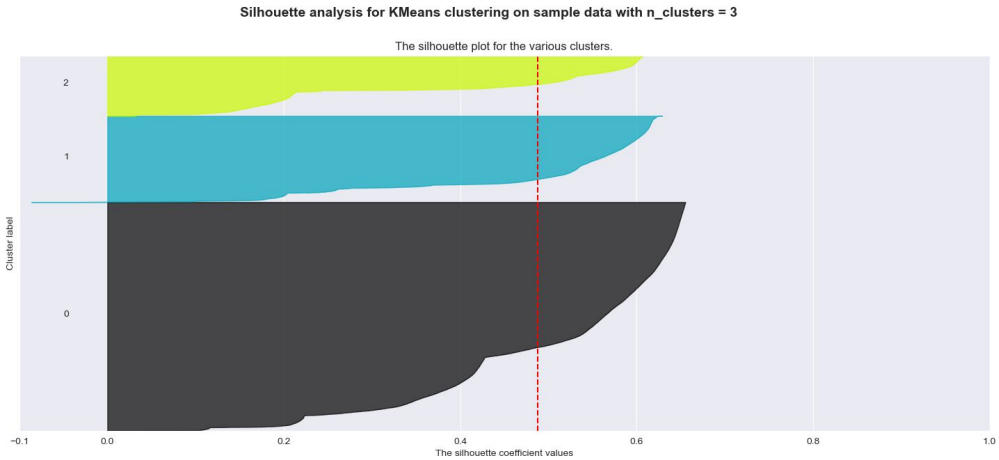
DBScan

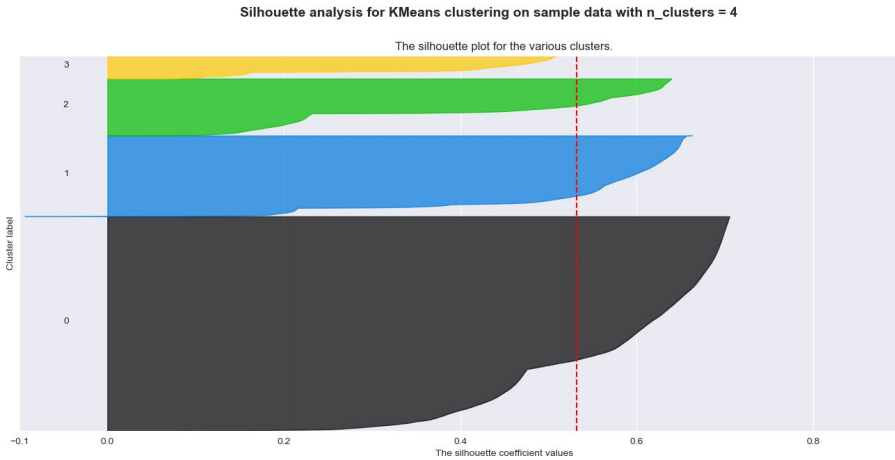




k-means - recherche de l'hyperpar. : n_clusters

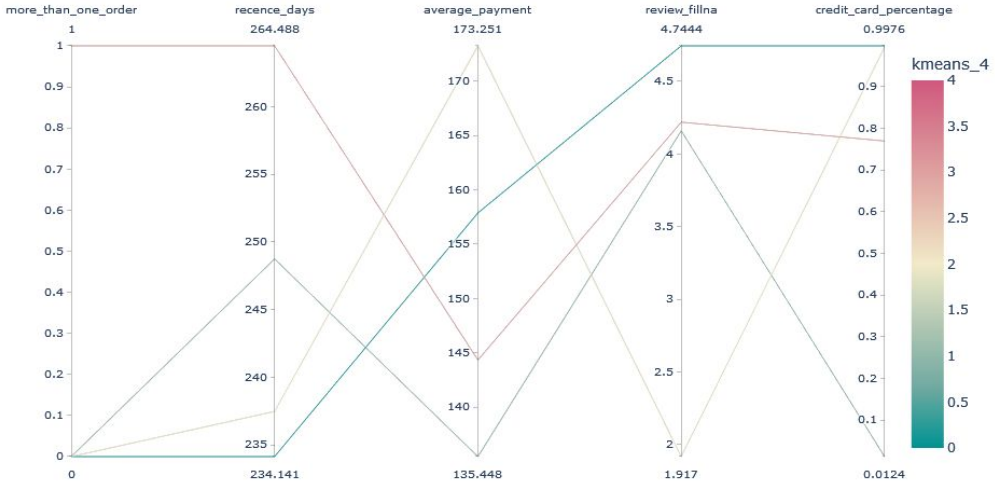
For n_clusters = 3 The average silhouette_score is : 0.4879
For n_clusters = 4 The average silhouette_score is : 0.5317
For n_clusters = 5 The average silhouette_score is : 0.4466
For n_clusters = 6 The average silhouette_score is : 0.4472

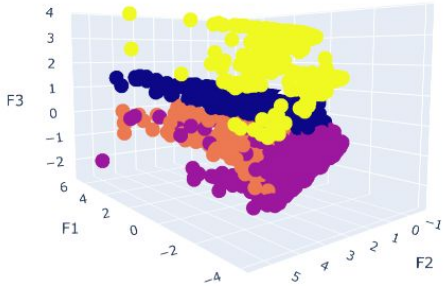




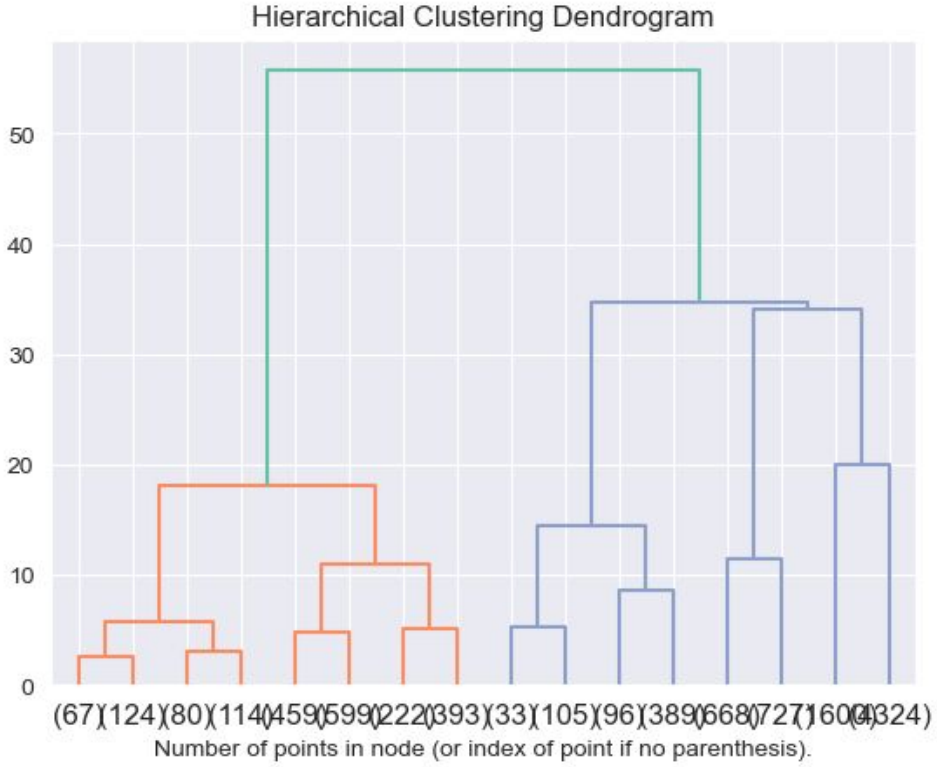
k-means - n_clusters = 4

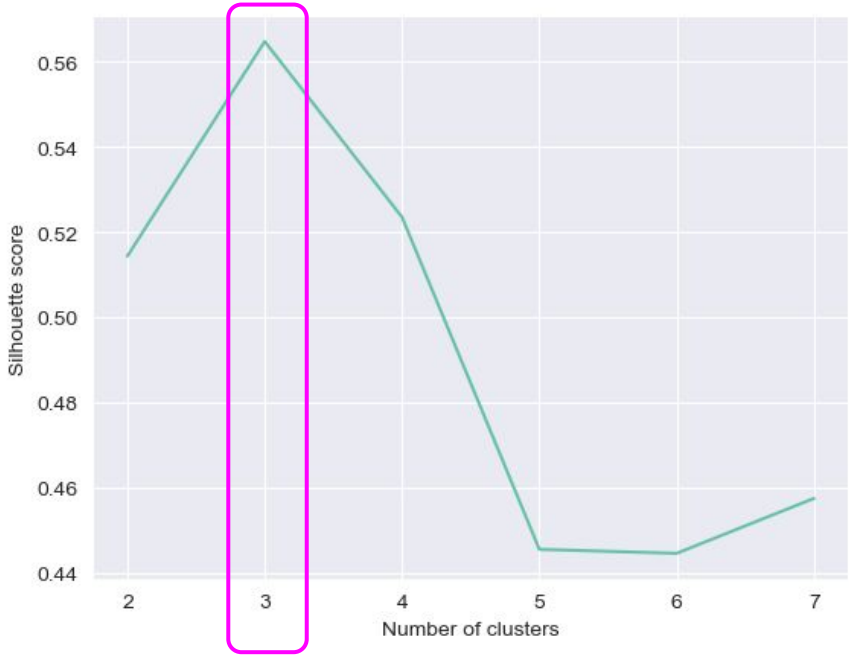
	kmeans_4	more_than_one_order	recence_days	average_payment	review_fillna	credit_card_percentage
0	0	0.0	234.140941	157.874224	4.744350	0.997600
1	1	0.0	248.742863	135.447514	4.159618	0.012419
2	2	0.0	237.477386	173.251067	1.916969	0.994295
3	3	1.0	264.488003	144.332462	4.218957	0.769313





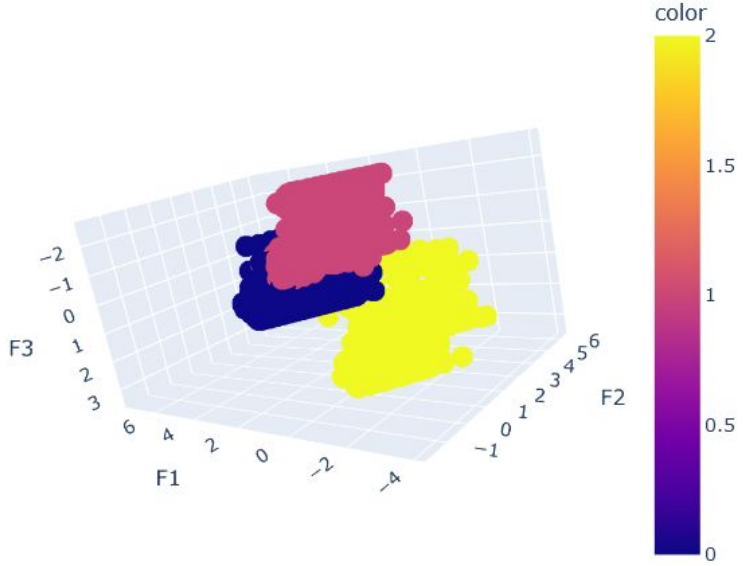
CAH - recherche de l'hyperpar. : n_clusters

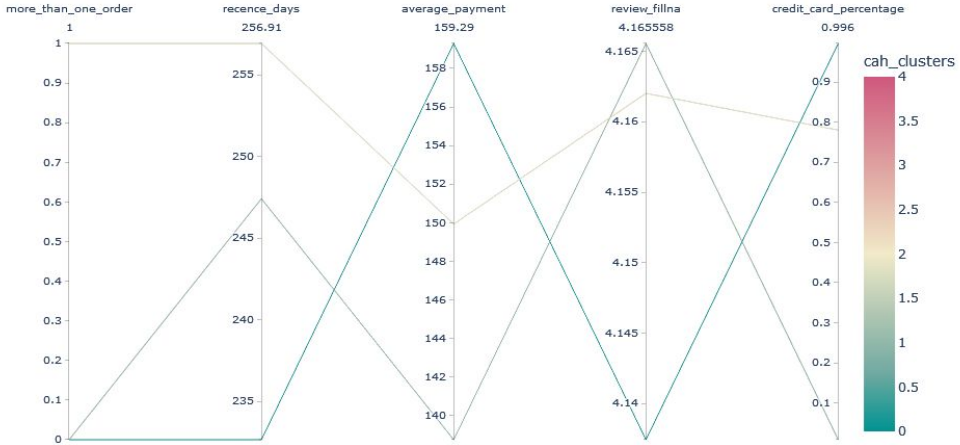




CAH - n_clusters = 3

cah_clusters	0	1	2
average_payment	159.29	138.75	149.93
credit_card_percentage	1.00	0.01	0.78
more_than_one_order	0.00	0.00	1.00
recence_days	232.65	247.41	256.91
review_fillna	4.14	4.17	4.16





DBScan - recherche des hyperparamètres & résultats

eps =0.1, min_samples =100
nb de clusters = 7
 bruit ? 20.17%
silhouette score = 0.18908478311952423

---

eps =0.25, min_samples =5
nb de clusters = 14
 bruit ? 0.26%
silhouette score = 0.2665197218405494

---

eps =0.25, min_samples =10
nb de clusters = 11
 bruit ? 0.51%
silhouette score = 0.2982272452196968

---

eps =0.25, min_samples =50
nb de clusters = 9
 bruit ? 2.24%
silhouette score = 0.5197694620480122

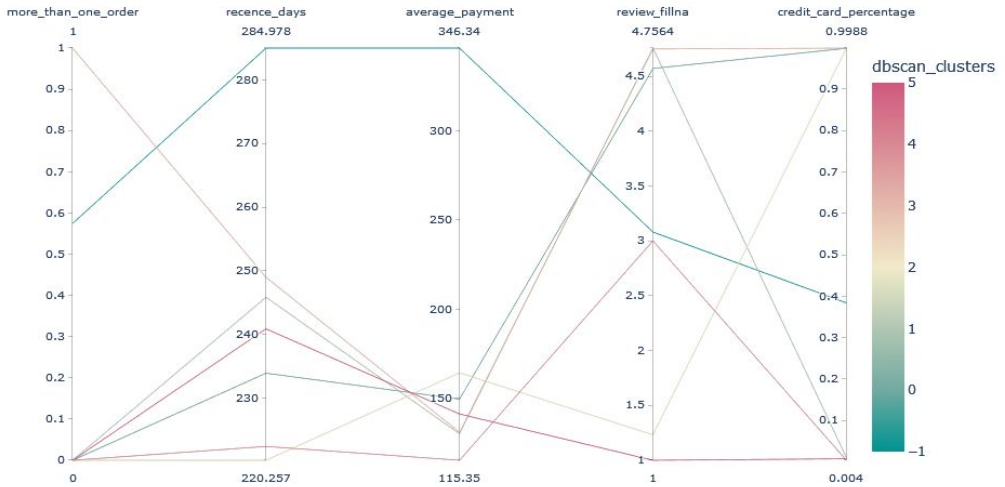
---

eps =0.25, min_samples =100
nb de clusters = 7
bruit ? 4.09%
silhouette score = 0.5155105232504489

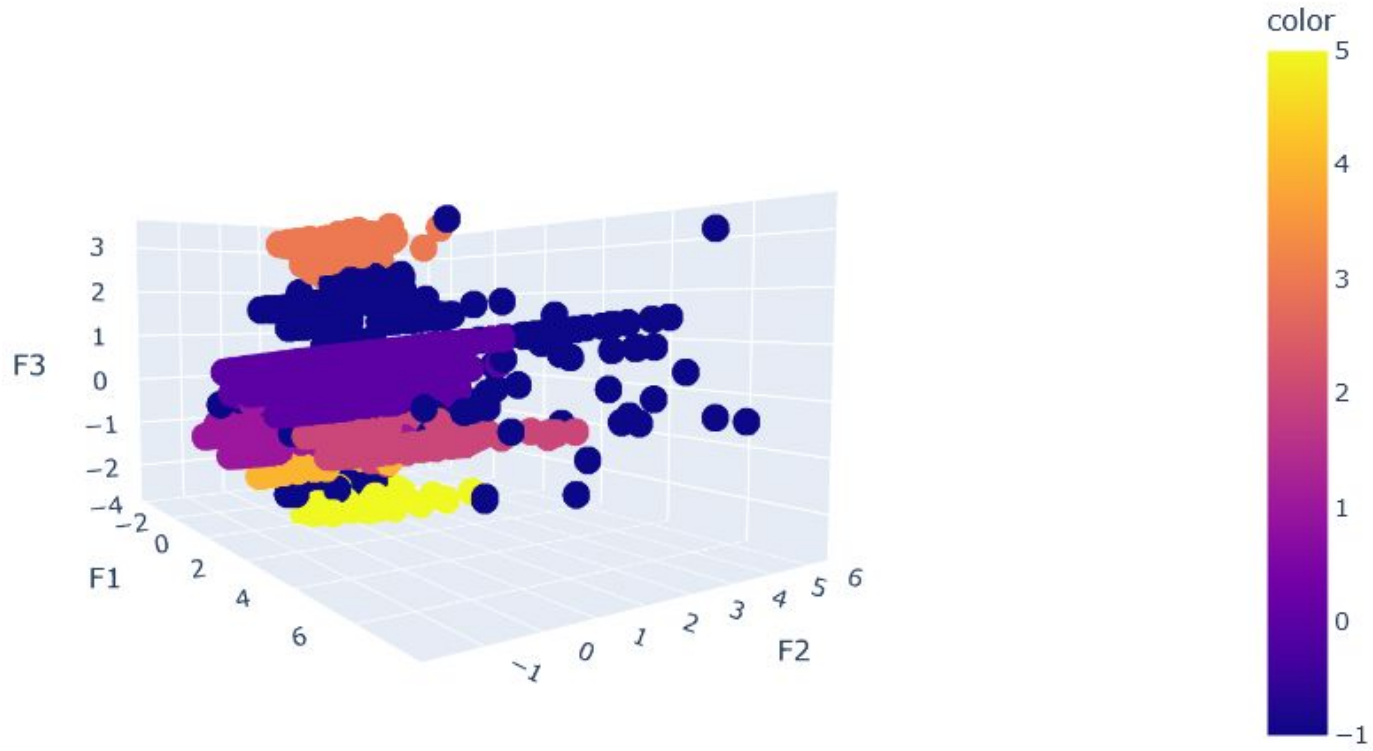
---

eps =0.5, min_samples =5
nb de clusters = 3
 bruit ? 0.01%
silhouette score = 0.4463297995889052

	dbscan_clusters	more_than_one_order	recence_days	average_payment	review_fillna	credit_card_percentage
0	-1	0.57	284.98	346.34	3.08	0.38
1	0	0.00	233.93	149.61	4.57	1.00
2	1	0.00	245.85	130.21	4.76	0.01
3	2	0.00	220.26	164.35	1.23	1.00
4	3	1.00	248.98	131.04	4.75	1.00
5	4	0.00	222.43	115.35	3.00	0.00
6	5	0.00	240.89	141.29	1.00	0.01



DBScan - eps =0.25, min_samples =100 , nb de clusters = 7 - Visualisation



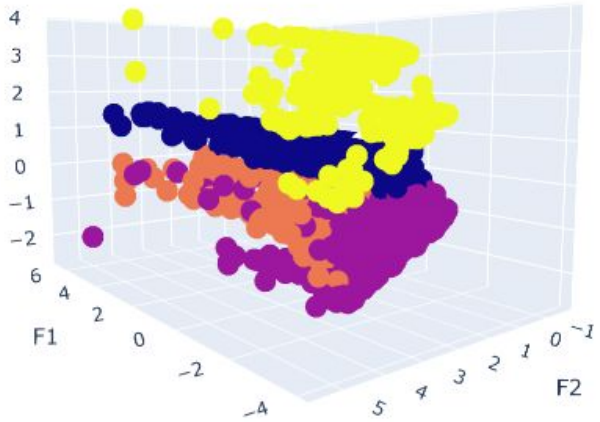
Choix du modèle & Segmentation

Choix du modèle - Comparatif

Type	Hyperpar.	Nb de clusters (actionnable ?)	Silhouette Score	Temps d'entraînement du modèle	Modèle réutilisable ?
K-Means	n_clusters = 4	4	0.53	Ok	Oui
CAH	n_clusters = 3	3	0.56	Plutôt long (recours à l'échantillonnage)	Non
DBScan	eps =0.25, min_samples =100 nb de clusters = 7	6 (car '-1'= bruit)	0.52 bruit : 4.09%	Plutôt long (recours à l'échantillonnage)	Non

	kmeans_4	more_than_one_order	recence_days	average_payment	review_fillna	credit_card_percentage
0	0	0.0	234.140941	157.874224	4.744350	0.997600
1	1	0.0	248.742863	135.447514	4.159618	0.012419
2	2	0.0	237.477386	173.251067	1.916969	0.994295
3	3	1.0	264.488003	144.332462	4.218957	0.769313

Choix du modèle - Segmentation



Segment 0: 🌟 Les VIPs 🌟

- **Portrait** : Récents, bonne review, panier moyen haut en carte et une seule commande
- **Actions**: Offrir une promotion spéciale ou un programme de fidélité pour encourager ces clients à revenir et à dépenser plus.

Segment 1: 🤔 Les Indécis 🤔

- **Portrait** : Plus anciens, bonne review, panier le plus bas sans carte et une seule commande
- **Actions**: Envoyer des e-mails de relance pour inciter ces clients à revenir et à effectuer un nouvel achat, offrir un coupon avec minimum d'achat pour les encourager à revenir en augmentant le panier.

Segment 2: 😔 Les Déçus 😔

- **Portrait** : Plutôt récents, mauvaise review, panier le plus haut en carte et une seule commande
- **Actions**: Contacter ces clients pour leur demander des commentaires et des suggestions d'amélioration, et offrir un coupon pour s'excuser de la gêne occasionnée.

Segment 3: 🎉 Les Réguliers 🎉

- **Portrait** : Les plus anciens, bonne review, panier moyen bas, méthodes de paiement mixte, et plus d'une commande.
- **Actions**: Offrir des avantages exclusifs ou un programme de récompenses pour remercier ces clients fidèles, ainsi que des promotions pour les inciter à dépenser plus et à parrainer leurs amis et leur famille.

Contrat de Maintenance

Maintenance - Simuler une màj de la BDD

Nb de commandes par mois

2016-10	236
2016-11	29
2016-12	2
2017-01	601
2017-02	1497
2017-03	2668
2017-04	2098
2017-05	3654
2017-06	3209
2017-07	3658
2017-08	4287
2017-09	4095
2017-10	4445
2017-11	6575
2017-12	6019
2018-01	6965
2018-02	6507
2018-03	7042
2018-04	6895
2018-05	6925
2018-06	6068
2018-07	5989
2018-08	7010
2018-09	2

2017 :
Année de base

2018 :
Test de màj
mois par mois

```python
def select_period (data,start_date,end_date):
    orders=data[3]
    order_payments=data[1]
    order_reviews=data[2]
    # star_date et end_date au format 'YYYY-MM-DD'
    filter_data=pd.to_datetime(orders['order_delivered_carrier_date']).dt.floor('d') #création de la date pour filtrer
    start_date=pd.to_datetime(start_date).floor('d') #mise en forme des dates début
    end_date=pd.to_datetime(end_date).floor('d') #et fin
    orders = orders.loc[filter_data.between(start_date, end_date)] #on filtre les orders avec les dates
    orders_id_list=orders['order_id'].unique()
    order_payments=order_payments[order_payments['order_id'].isin(orders_id_list)]
    order_reviews=order_reviews[order_reviews['order_id'].isin(orders_id_list)]
    data_return=[data[0],order_payments,order_reviews,orders]
    return data_return
```

```python
#période de référence
start_date='2017-01-01'
end_date='2017-12-31'
b0_data=select_period(data_import,start_date,end_date)
b0_data=calcul_var(b0_data)
b0_c0=MinMax_train_kmeans(b0_data)
s0=b0_c0[0]
c0=b0_c0[1]
b0_by_c0=c0.predict(s0.transform(b0_data))
```

La séparation des données par date pour b1 à bn

```python
dates=[('2017-01-01', '2018-01-31'),
 ('2017-01-01', '2018-02-28'),
 ('2017-01-01', '2018-03-31'),
 ('2017-01-01', '2018-04-30'),
 ('2017-01-01', '2018-05-31'),
 ('2017-01-01', '2018-06-30'),
 ('2017-01-01', '2018-07-31'),
 ('2017-01-01', '2018-08-31')]
```

```python
data=data_import.copy()
for i,dates in enumerate(dates) :
    start_date, end_date = dates
    bx_data=select_period(data,start_date,end_date)
    bx_data=calcul_var(bx_data)
    bx_cx=MinMax_train_kmeans(bx_data)
    sx=bx_cx[0]
    cx=bx_cx[1]
    exec(f'b{i+1}_data = bx_data')
    exec(f'c{i+1} = cx')
    exec(f's{i+1} = sx')
```

Maintenance - Comparer les segmentations avec/sans màj

7. Boucle pour segmenter Bn avec clustering C0

SOMMAIRE

```
for i in range(8) :
    exec(f'bx_data = b{i+1}_data')
    bx_by_c0 = c0.predict(s0.transform(bx_data))
    exec(f'b{i+1}_by_c0 = bx_by_c0')
```

8. Boucle pour segmenter Bn avec clustering Cn

SOMMAIRE

```
for i in range(8) :
    exec(f'bx_data = b{i+1}_data')
    exec(f'sx = s{i+1}')
    exec(f'cx = c{i+1}')
    bx_by_cx = cx.predict(sx.transform(bx_data))
    exec(f'b{i+1}_by_c{i+1} = bx_by_cx')
```

9. Comparaison des segmentations avec ARI (indice RAND

SOMMAIRE

```
from sklearn.metrics import adjusted_rand_score as ARI
```

```
resultats = pd.DataFrame(columns=['Comparaison_c0_cn_sur', 'indice_rand'])
```

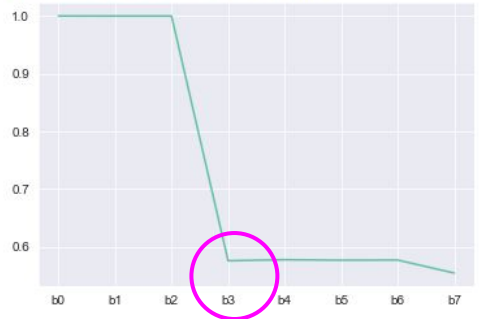
```
for i in range(9) :
    exec(f'bx_by_c0 = b{i}_by_c0')
    exec(f'bx_by_cx = b{i}_by_c{i}')
    indice_rand=ARI(bx_by_c0,bx_by_cx)
    text='b'+str(i)
    resultats.loc[len(resultats)] = [str(text), round(indice_rand,5)]
```

```
resultats
```

	Comparaison_c0_cn_sur	indice_rand
0	b0	1.00000
1	b1	0.99994
2	b2	0.99995
3	b3	0.57549
4	b4	0.57731
5	b5	0.57644
6	b6	0.57669
7	b7	0.55415

```
plt.plot(resultats['Comparaison_c0_cn_sur'],resultats['indice_rand'])
```

```
[<matplotlib.lines.Line2D at 0x1f6386f42b0>]
```



⇒ Màj obligatoire dès le 3ème mois

Conclusions & Remarques

Ce que nous proposons au client :

- Une segmentation des clients via la méthode k-means avec l'hyper paramètre n_clusters = 4, réutilisable 3 mois (m0, m+1 , m+2) et à mettre à jour à m+3
- Une typologie/nomenclature des clients avec rapide description et une proposition d'actions à titre d'exemple

Si c'était à refaire :

- Je ferai des essais avec d'autres variables, pour essayer d'améliorer encore plus le silhouette score (temps de réception de la commande ? utilisation des boleto ?)

Une idée :

- Peut-être que l'on pourrait utiliser les centroids du modèles dbscan pour initier un kmeans à 6 clusters ? Car cest peut être mieux d'avoir + de clusters

Le plus important : j'ai gagné en compétences