Projet Ecommerce

Efkan TUREDI

Intro

Notre client, Olist, souhaite avoir une segmentation client à des fins de marketings ciblés. Les livrables attendus par le clients sont:

- Une description actionable de notre segmentation et de sa logique
- Contrat de maintenance basée sur une analyse de la stabilité des segments au cours du temps.

Pour faire cette analyse, nous disposons d'un nombre élevé de fichiers .csv qu'il faudra nettoyer et analyser avec l'aide d'algorithme non-supervisés

Le nettoyage

Nous avons des problèmes courants...

- ❏ Certaines données sont en portugais
- ❏ Erreurs d'entrées dans les données (ex: string dans une colonne float)
- ❏ NaNs dans certaines colonnes

Elles représentent toutefois une faible proportion de notre base de données

```
for item in liste_db:
  print(item.isna().sum().sum())
```

```
0
0
0
0
146532
4908
2448
0
0
```

3 de nos fichiers ont des NaNs: *order_reviews*, *orders* et *products*

```
[294]: liste_db = [
         customersX,
         geolocationX,
         order_itemsX,
         order_paymentsX,
         order_reviewsX,
         ordersX,
         productsX,
         sellersX,
         product_category_name_translationX
       ]
```

Quelques lignes de codes pour illustration

Regardons le nombre de NaNs dans nos fichiers CSV

```
for item in liste_db:
  print(item.isna().sum().sum())
```

```
0
0
0
0
146532
4908
2448
0
0
```

Les NaNs sont situés uniquement sur 3 de nos fichiers: order_reviews, orders, et products

```
[294]: liste_db = [
         customersX,
         geolocationX,
         order_itemsX,
         order_paymentsX,
         order_reviewsX,
         ordersX,
         productsX,
         sellersX,
         product_category_name_translationX
       ]
```

Résolutions des problèmes de data

- Traductions des données portugaises vers l'anglais
- Les NaNs sont remplacés par des valeurs valables selon les cas (voir slide suivante)
- Remplacement des quelques strings dans les colonnes float par la moyenne de la colonne float en question

Quelques lignes de codes pour illustration (½)

In [206]:
```
ordersX['order_approved_at'].fillna('N/A', inplace=True)
ordersX['order_delivered_carrier_date'].fillna('N/A', inplace=True)
ordersX['order_delivered_customer_date'].fillna('N/A', inplace=True)
```

In [207]:
```
ordersX.duplicated().sum()
```

Out[207]: 0

In [208]:
```
geolocationX.drop_duplicates(inplace=True)
geolocationX['customer_zip_code_prefix'] = geolocationX['geolocation_zip_code_prefix']
```

In [209]:
```
order_reviewsX['review_comment_title'].fillna('None',inplace=True)
order_reviewsX['review_comment_message'].fillna('None',inplace=True)
```

In [210]:
```
productsX['product_category_name'].fillna('Unknown',inplace=True)
productsX['product_name_lenght'].fillna(0, inplace=True)
productsX['product_description_lenght'].fillna(0,inplace=True)
productsX['product_photos_qty'].fillna(0, inplace=True)
productsX['product_weight_g'].fillna(0, inplace=True)
productsX['product_length_cm'].fillna(0, inplace=True)
productsX['product_height_cm'].fillna(0, inplace=True)
productsX['product_width_cm'].fillna(0, inplace=True)
```

In [211]:
```
order_itemsX['shipping_limit_date'] = order_itemsX['shipping_limit_date'].astype('datetime64')
ordersX['order_purchase_timestamp'] = ordersX['order_purchase_timestamp'].astype('datetime64')
ordersX['order_approved_at'] = pd.to_datetime(ordersX['order_approved_at'], errors='coerce')
ordersX['order_delivered_carrier_date'] = pd.to_datetime(ordersX['order_delivered_carrier_date'], errors='coerce')
ordersX['order_delivered_customer_date'] = pd.to_datetime(ordersX['order_delivered_customer_date'], errors='coerce')
ordersX['order_estimated_delivery_date'] = ordersX['order_estimated_delivery_date'].astype('datetime64')
```

In [212]:
```
products_local = pd.merge(productsX, product_category_name_translationX).drop(['product_category_name'], axis=1)
```

Quelques lignes de codes pour illustration (2/2)

```
master_df1
```

customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	orde
b583a7efe4522c8ce8942bd47f33d487	canceled	2018-04-26 08:13:54	2018-04-26 08:32:52	NaT	NaT	
b583a7efe4522c8ce8942bd47f33d487	canceled	2018-04-26 08:13:54	2018-04-26 08:32:52	NaT	NaT	
65f09de0b77ca07963fde8601c6be1fb	invoiced	2017-11-10 16:27:32	2017-11-10 16:50:47	NaT	NaT	
a979b3cbd898fd8be91a290a667fd0d4	shipped	2017-10-16 21:13:03	2017-10-16 21:28:13	2017-10-26 20:40:20	NaT	
2737211835d5ea370af15ee145f7840c	canceled	2018-08-15 15:13:32	NaT	NaT	NaT	
...	...	...	...	...	...	
b8bd03cdd075b29c82a9c55e5cd8e224	unavailable	2017-06-09 20:25:03	2017-06-09 20:35:17	NaT	NaT	
df7338a04458506a3b2f23056c466e88	shipped	2017-06-07 17:02:27	2017-06-07 17:10:20	2017-06-09 12:03:00	NaT	
cd0a090974c3b64acf613c18f9fcfe83	shipped	2017-11-23 17:11:23	2017-11-24 10:11:48	2017-11-27 18:52:06	NaT	
d8f3aacf5cf5ddf9ceef4ad39e874c98	canceled	2018-03-15 10:06:35	2018-03-15 10:29:39	NaT	NaT	
e4abb5057ec8cfda9759c0dc415a8188	invoiced	2017-11-18 17:27:14	2017-11-18 17:46:05	NaT	NaT	

Les NaNs / NaT touchent des “orders” particuliers: annulé, pas encore recu, pas disponible,...

Ces lignes vont introduire du bruit dans notre base de données. Nous les supprimons de notre BDD (c. 3500 lignes)

Cas particulier de géolocalisation

Dernières corrections de NaNs lors du merge avec les localisations. On

corrige en faisant un Hashmap avec les moyennes des localisations des états

In [333]:
```
geo_hash
```

Out[333]:

	customer_state	geolocation_lat	geolocation_lng
0	AC	-9.940908	-68.043663
1	AL	-9.636357	-36.052805
2	AM	-3.198464	-60.056560
3	AP	0.038680	-51.167226
4	BA	-13.022167	-39.439324
5	CE	-4.310180	-38.966381
6	DF	-15.809172	-47.976446
7	ES	-20.148297	-40.490957
8	GO	-16.613947	-49.329894
9	MA	-3.577111	-44.706294
10	MG	-19.916511	-44.438023

In [334]:
```
zipbObj_lng = zip(geo_hash['customer_state'], geo_hash['geolocation_lng'])
Dict_lng = dict(zipbObj_lng)

zipbObj_lat = zip(geo_hash['customer_state'], geo_hash['geolocation_lat'])
Dict_lat = dict(zipbObj_lat)

master_df['geolocation_lat'].fillna(master_df['customer_state'].map(Dict_lat), inplace = True)
master_df['geolocation_lng'].fillna(master_df['customer_state'].map(Dict_lng), inplace = True)
```

In [335]:
```
master_df.isna().sum().sum()
```

Out[335]: 0

In [330]:
```
master_df.isna().sum()
```

Out[330]:

```
order_id                          0
payment_sequential                0
payment_type                      0
payment_installments              0
payment_value                     0
customer_id                       0
order_status                      0
order_purchase_timestamp          0
order_approved_at                 0
order_delivered_carrier_date      0
order_delivered_customer_date     0
order_estimated_delivery_date     0
order_item_id                     0
product_id                        0
seller_id                         0
shipping_limit_date               0
price                             0
freight_value                     0
review_id                         0
review_score                      0
review_comment_title              0
review_comment_message            0
review_creation_date              0
review_answer_timestamp           0
customer_unique_id                0
customer_zip_code_prefix          0
customer_city                     0
customer_state                    0
product_category_name             0
product_name_lenght               0
product_description_lenght        0
product_photos_qty                0
product_weight_g                  0
product_length_cm                 0
product_height_cm                 0
product_width_cm                  0
seller_zip_code_prefix            0
seller_city                       0
seller_state                      0
geolocation_lat                 303
geolocation_lng                 303
dtype: int64
```

Un aperçu de notre fichier master_df actuel

[336]:
```
master_df
```

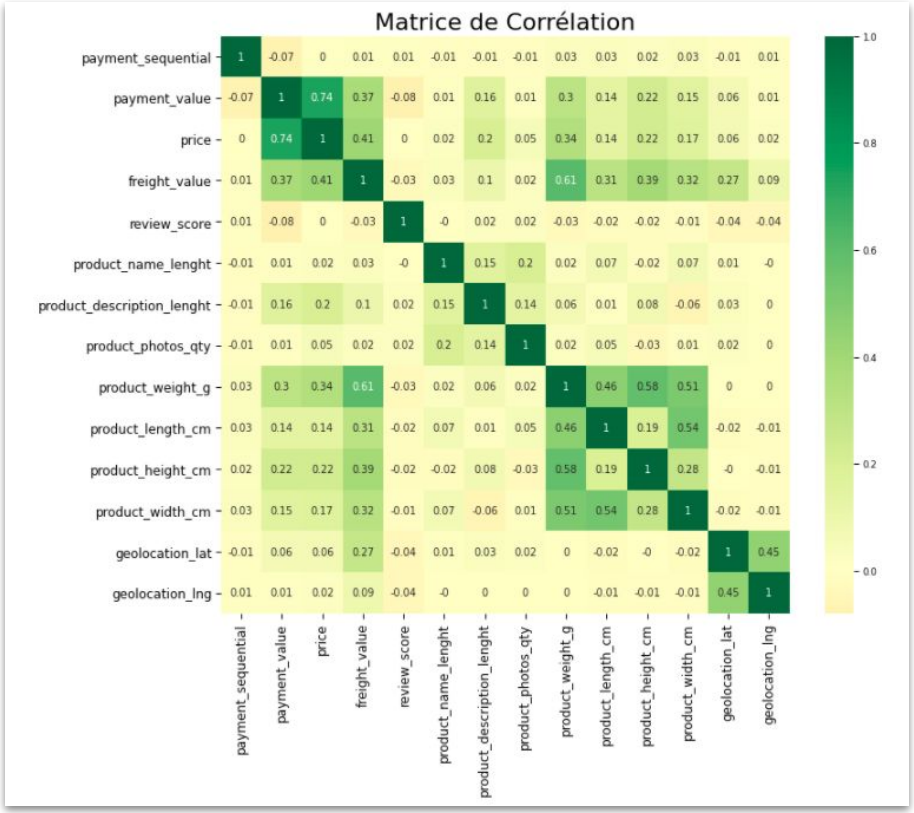
[336]:

	order_id	payment_sequential	payment_type	payment_installments	payment_value	customer_id
0	b81ef226f3fe1789b1e8b2acac839d17	1	credit_card	8	99.33	0a8556ac6be836b46b3e89920d59291c
1	a9810da82917af2d9aefd1278f1dcfa0	1	credit_card	1	24.39	f2c7fc58a9de810828715166c672f10a
2	25e8ea4e93396b6fa0d3dd708e76c1bd	1	credit_card	1	65.71	25b14b69de0b6e184ae6fe2755e478f9
3	ba78997921bbcdc1373bb41e913ab953	1	credit_card	8	107.78	7a5d8efaaa1081f800628c30d2b0728f
4	ba78997921bbcdc1373bb41e913ab953	1	credit_card	8	107.78	7a5d8efaaa1081f800628c30d2b0728f
...	...	...	...	...	...	...
115706	c45067032fd84f4cf408730ff5205568	1	credit_card	2	198.94	0fea3afc6a1510c9db75d349d28af974
115707	0406037ad97740d563a178ecc7a2075c	1	boleto	1	363.31	5d576cb2dfa3bc05612c392a1ee9c654
115708	7b905861d7c825891d6347454ea7863f	1	credit_card	2	96.80	2079230c765a88530822a34a4cec2aa0
115709	b8b61059626efa996a60be9bb9320e10	1	credit_card	5	369.54	5d719b0d300663188169c6560e243f27
115710	28bbae6599b09d39ca406b747b6632b1	1	boleto	1	191.58	4c7f868f43b5cff577b0becb8c8b7860

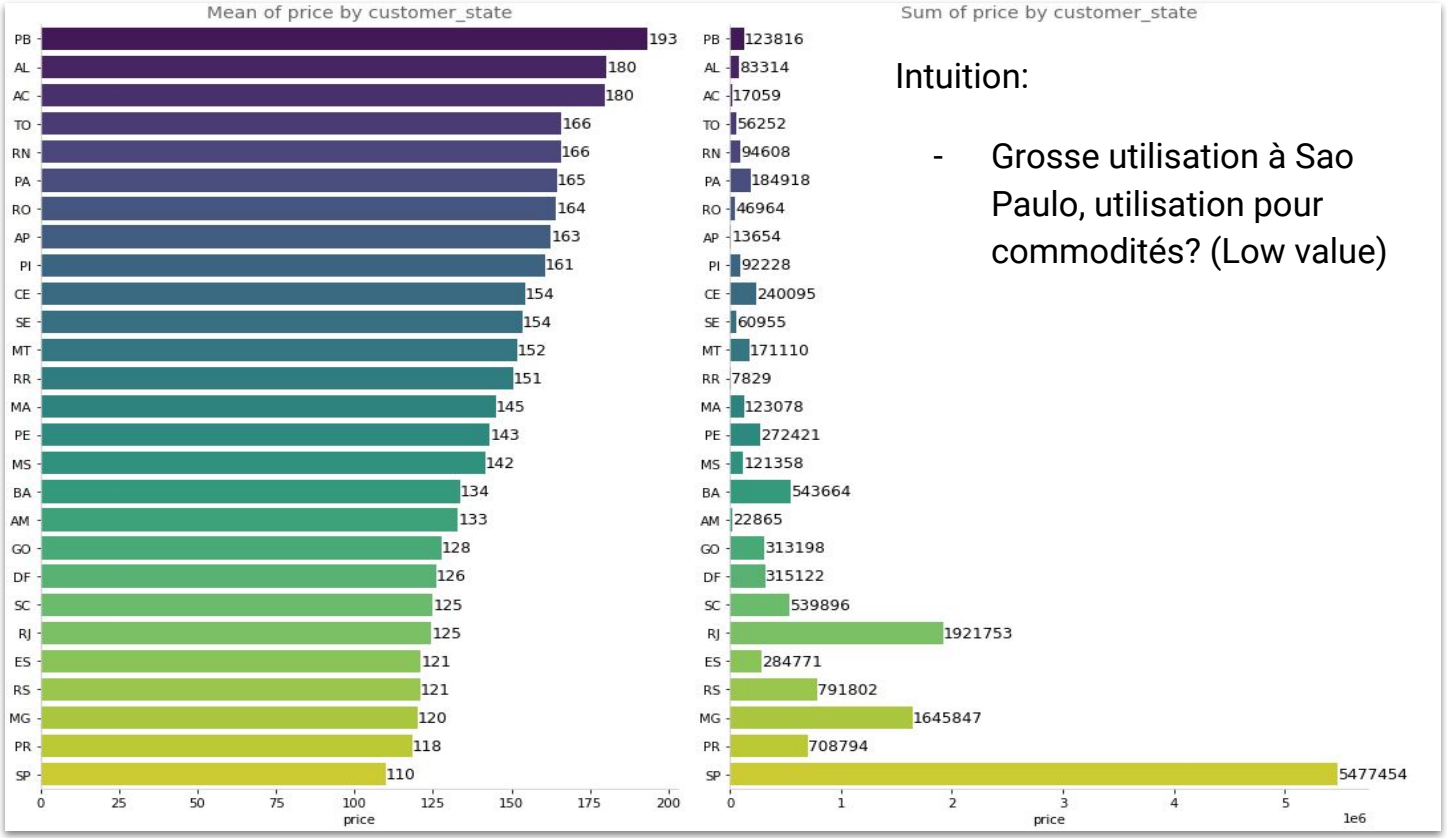
115711 rows × 41 columns

Analyse exploratoire

Matrice de correlation



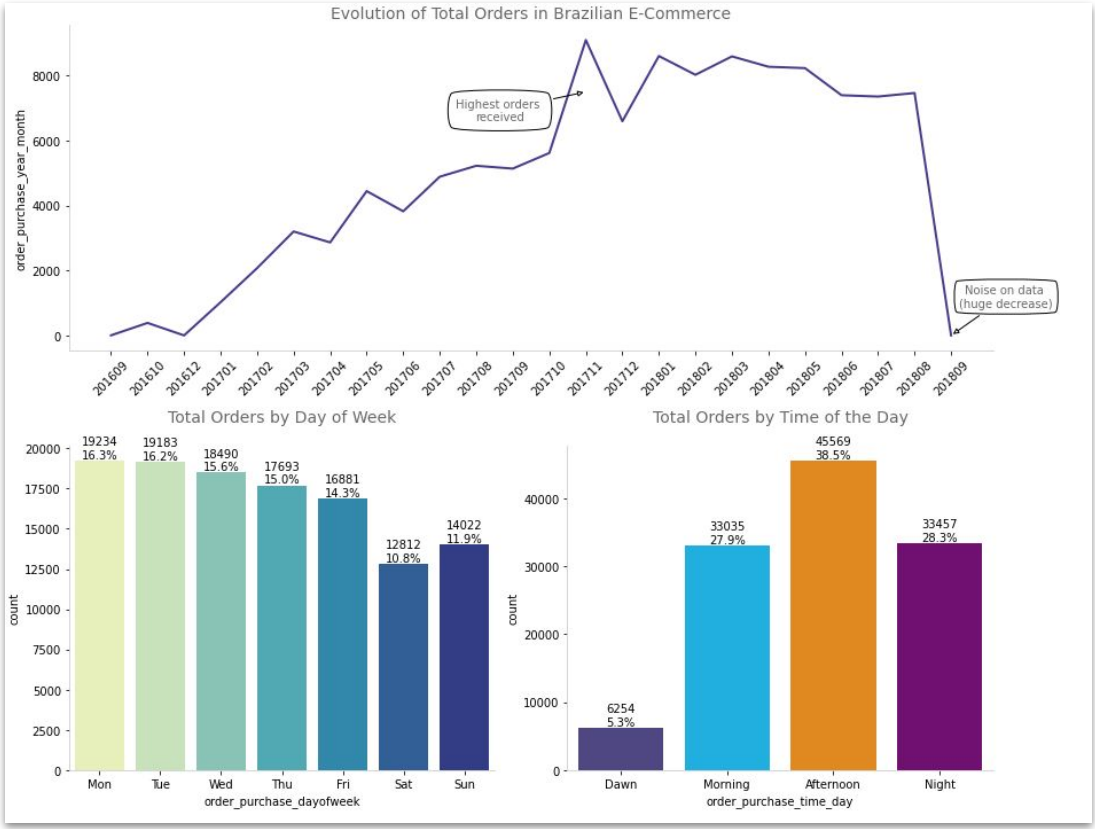
Quelques données sur les clients d'Olist (1/5)



Intuition:

- Grosse utilisation à Sao Paulo, utilisation pour commodités? (Low value)

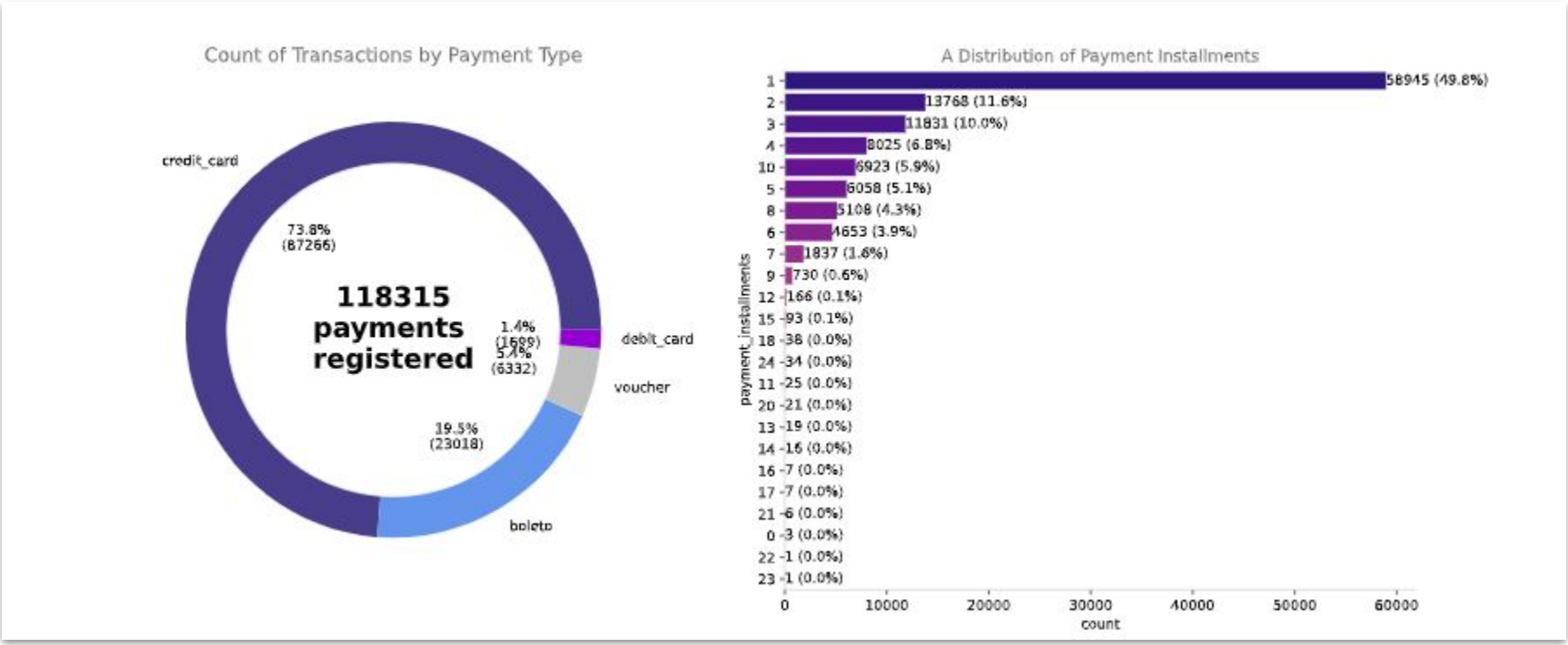
Quelques données sur les clients d'Olist (2/5)



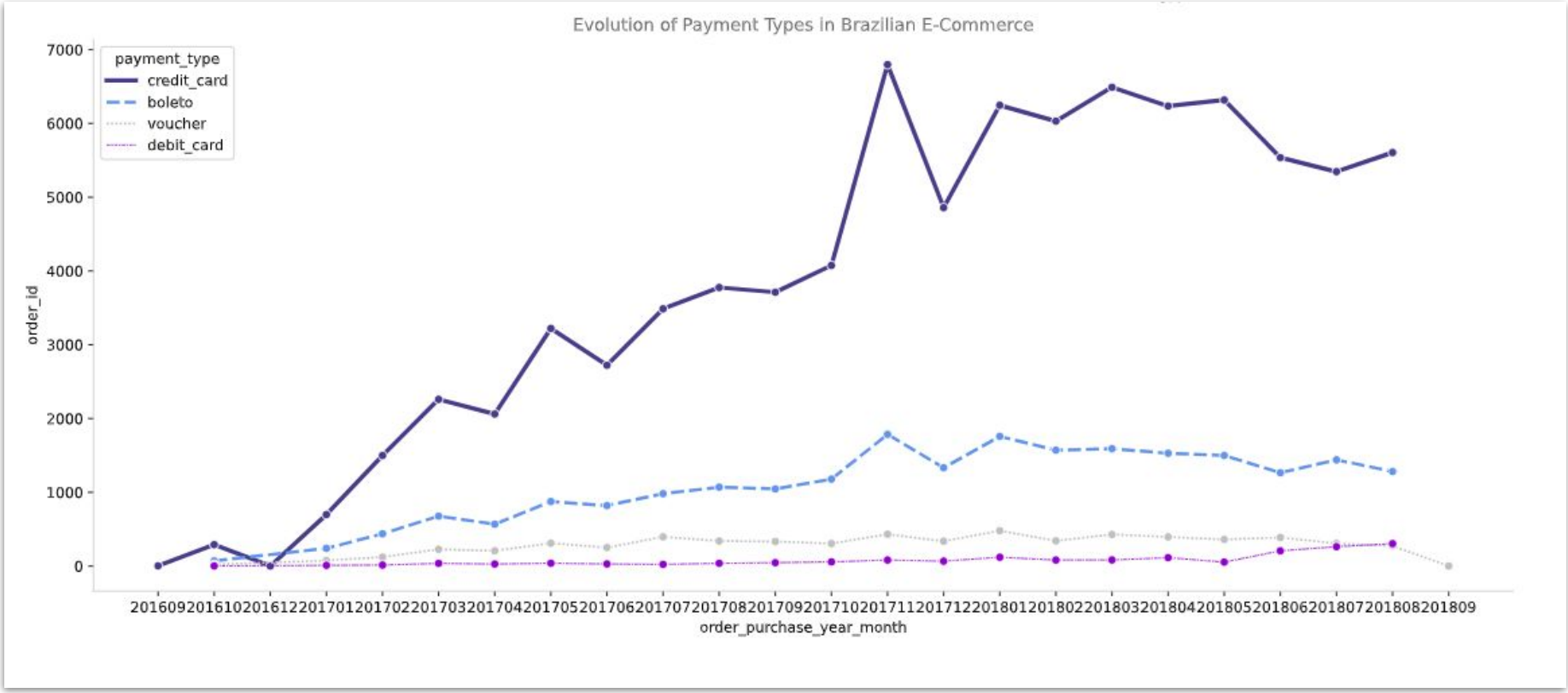
Intuitions:

- Nombre total de commandes drivés essentiellement par l'augmentation du nombre de clients plutôt qu'une aug. du # de commandes par clients
- Le plus gros des utilisations a lieu en début de semaine, dans l'après midi

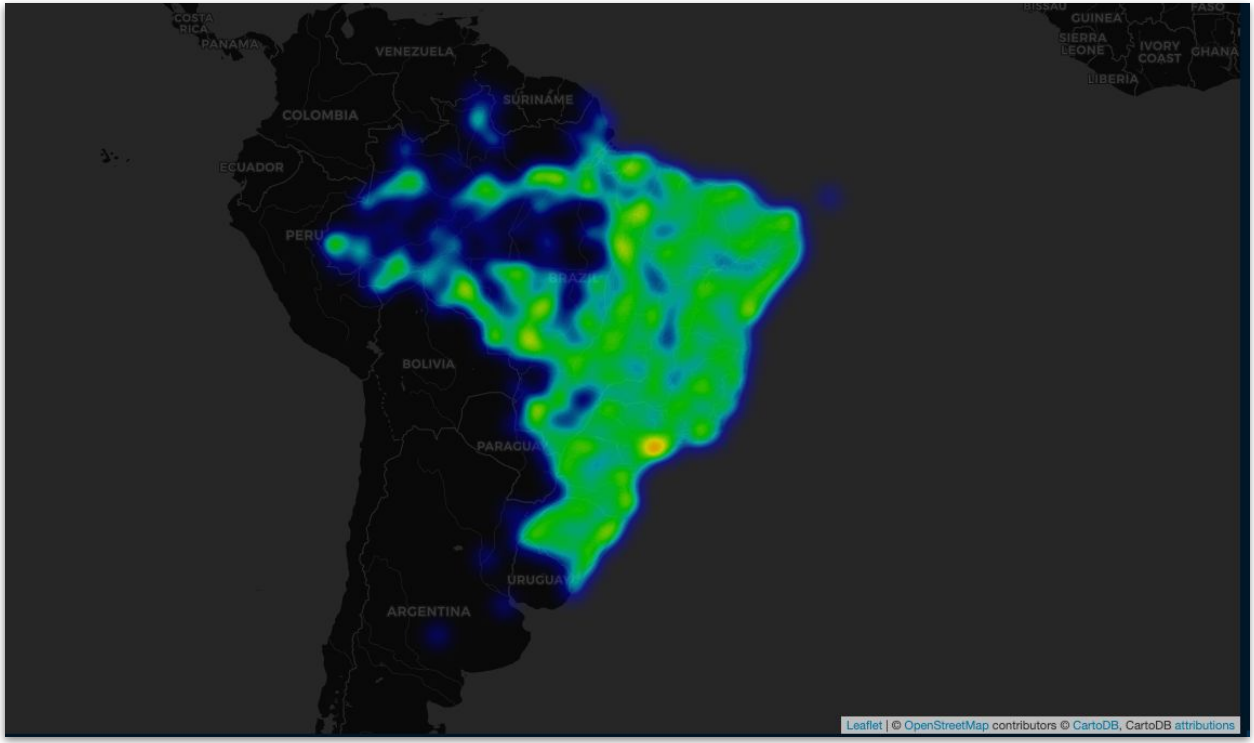
Quelques données sur les clients d'Olist (3/5)



Quelques données sur les clients d'Olist (4/5)



Quelques données sur les clients d'Olist (5/5)



Intuitions:

- Clients largement concentré sur les côtes, très urbanisées
- Le reste du pays est plus compliqué notamment au niveau logistique (Forêt amazonienne)

Feature engineering

Feature engineering (½)

Nous approchons cet exercice avec une volonté de mettre “product_category_name” au coeur de notre clustering. Nous avons donc un travail de feature engineering à mettre en place.

- Passage à une base de données avec customer_id
- Utilisation des fonctions groupby et agg de pandas
- Utilisation de MultilabelBinarizer pour O-H-E de product_category (slide suivante) pour une PCA

```
customer_df = master_df.groupby([
  'customer_unique_id',
  'customer_state',
], as_index=False).agg({
  'payment_value': 'sum',
  'payment_type' : most_frequent,
  'product_category_name' : lister,
  'review_score': 'mean',
  'freight_value':'sum',
  'geolocation_lat':'mean',
  'geolocation_lng':'mean'
})
```

Feature engineering (2/2)

Base de données “autour” de customer ID

	customer_unique_id	customer_state	payment_value	payment_type	product_category_name	review_score	freight_value	geolocation_lat	g
0	0000366f3b9a7992bf8c76cfdf3221e2	SP	141.90	credit_card	[bed_bath_table]	5.0	12.00	-23.340262	
1	0000b849f77a49e4a4ce2b2a4ca5be3f	SP	27.19	credit_card	[health_beauty]	4.0	8.29	-23.559044	
2	0000f46a3911fa3c0805444483337064	SC	86.22	credit_card	[stationery]	3.0	17.22	-27.543010	
3	0000f6ccb0745a6a4b88665a16c9f078	PA	43.62	credit_card	[telephony]	4.0	17.63	-1.312726	
4	0004aac84e0df4da2b147fca70cf8255	SP	196.89	credit_card	[telephony]	5.0	16.89	-23.505588	
...	...	...	...	...	...	...	...	...	
93336	fffcf5a5ff07b0908bd4e2dbc735a684	PE	4134.84	credit_card	[health_beauty]	5.0	497.42	-8.362654	
93337	fffea47cd6d3cc0a88bd621562a9d061	BA	84.58	credit_card	[baby]	4.0	19.69	-12.217900	
93338	ffff371b4d645b6ecea244b27531430a	MT	112.46	credit_card	[auto]	5.0	22.56	-11.834705	
93339	ffff5962728ec6157033ef9805bacc48	ES	133.69	credit_card	[watches_gifts]	5.0	18.69	-21.126170	
93340	ffffd2657e2aad2907e67c3e9daecbeb	PR	71.56	credit_card	[perfumery]	5.0	14.57	-25.445705	

93341 rows × 9 columns

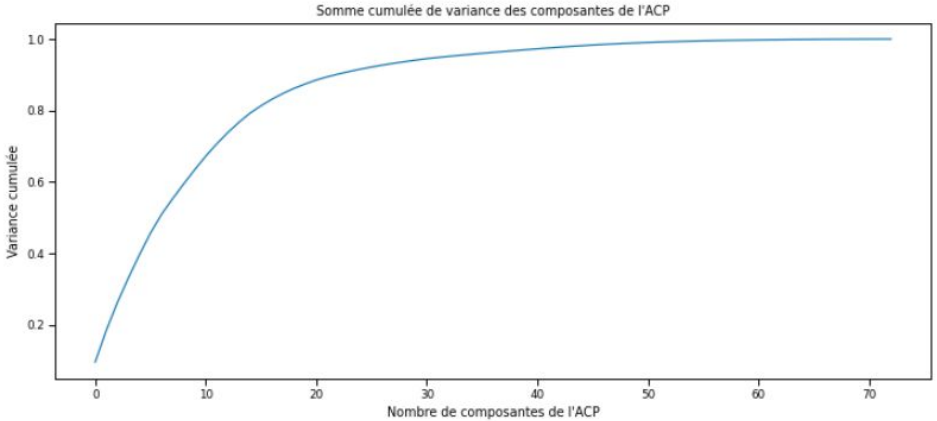
Multilabel Binarizer sur Product Category

	Other	agro_industry_and_commerce	air_conditioning	art	arts_and_craftmanship	audio	auto	baby	bed_bath_table	books_general_interest	books_impo
0	0	0	0	0	0	0	0	0	1	0	
1	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
93082	0	0	0	0	0	0	0	0	0	0	
93083	0	0	0	0	0	0	0	1	0	0	
93084	0	0	0	0	0	0	1	0	0	0	
93085	0	0	0	0	0	0	0	0	0	0	
93086	0	0	0	0	0	0	0	0	0	0	

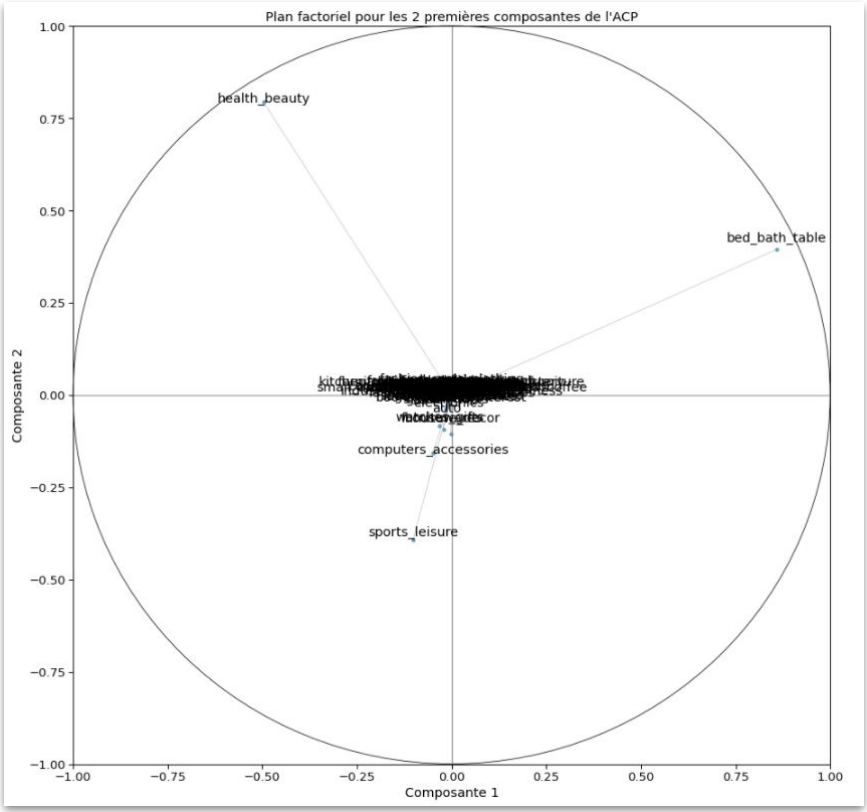
PCA sur product_category

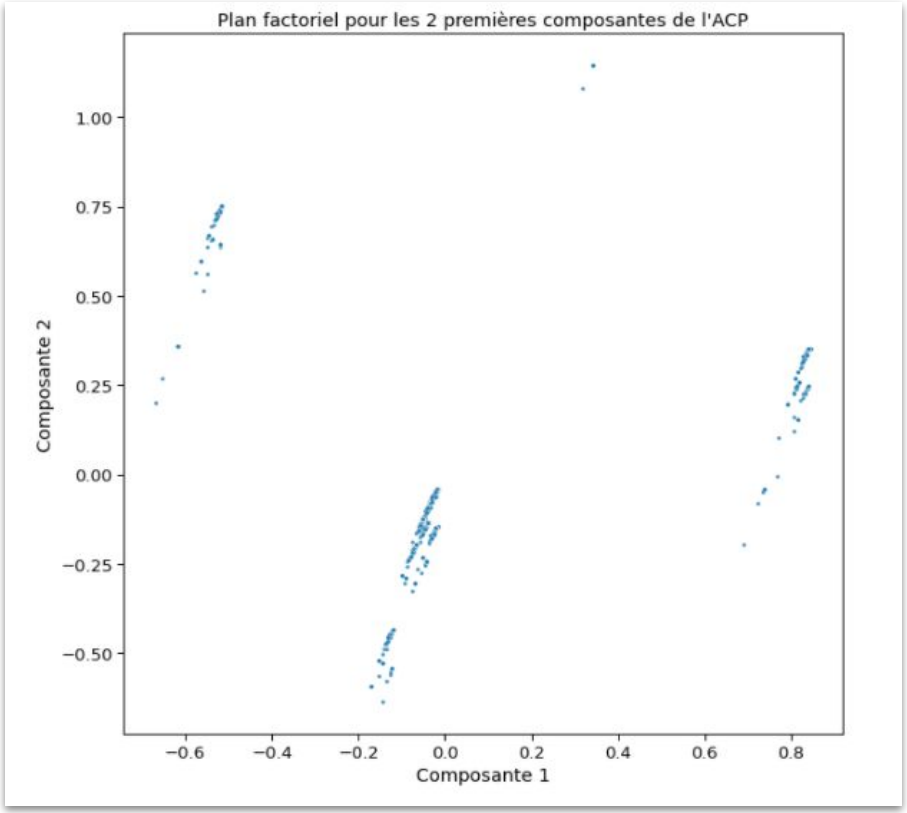
Vu le nombre de colonnes présents dans “product_category” il serait intéressant de faire une réduction de dimension avec PCA pour enlever les features peu utiles.

PCA semble bien fonctionner: 15 composantes (au lieu de >70)contiennent 80% de la variance dans product_category



PCA sur product_category





Création du fichier final pour ML

```
final_df = pd.merge(customer_df_ss,product_pca_df,on='customer_unique_id',how='left')
```

```
final_df
```

	customer_unique_id	customer_state	payment_value	payment_type	product_category_name	review_score	freight_value	geolocation_lat	g
0	0000366f3b9a7992bf8c76cfdf3221e2	SP	-0.109858	credit_card	[bed_bath_table]	0.665746	-0.466776	-0.383089	
1	0000b849f77a49e4a4ce2b2a4ca5be3f	SP	-0.287183	credit_card	[health_beauty]	-0.108041	-0.602544	-0.422063	
2	0000f46a3911fa3c0805444483337064	SC	-0.195931	credit_card	[stationery]	-0.881828	-0.275749	-1.131777	
3	0000f6ccb0745a6a4b88665a16c9f078	PA	-0.261785	credit_card	[telephony]	-0.108041	-0.260745	3.540951	
4	0004aac84e0df4da2b147fca70cf8255	SP	-0.024851	credit_card	[telephony]	0.665746	-0.287825	-0.412540	
...	...	...	...	...	...	...	...	...	
93082	fffcf5a5ff07b0908bd4e2dbc735a684	PE	6.062667	credit_card	[health_beauty]	0.665746	17.297269	2.285060	
93083	fffea47cd6d3cc0a88bd621562a9d061	BA	-0.198466	credit_card	[baby]	-0.108041	-0.185359	1.598276	
93084	ffff371b4d645b6ecea244b27531430a	MT	-0.155368	credit_card	[auto]	0.665746	-0.080330	1.666540	
93085	ffff5962728ec6157033ef9805bacc48	ES	-0.122549	credit_card	[watches_gifts]	0.665746	-0.221954	0.011335	
93086	ffffd2657e2aad2907e67c3e9daecbeb	PR	-0.218594	credit_card	[perfumery]	0.665746	-0.372726	-0.758158	

93087 rows × 24 columns

Machine Learning

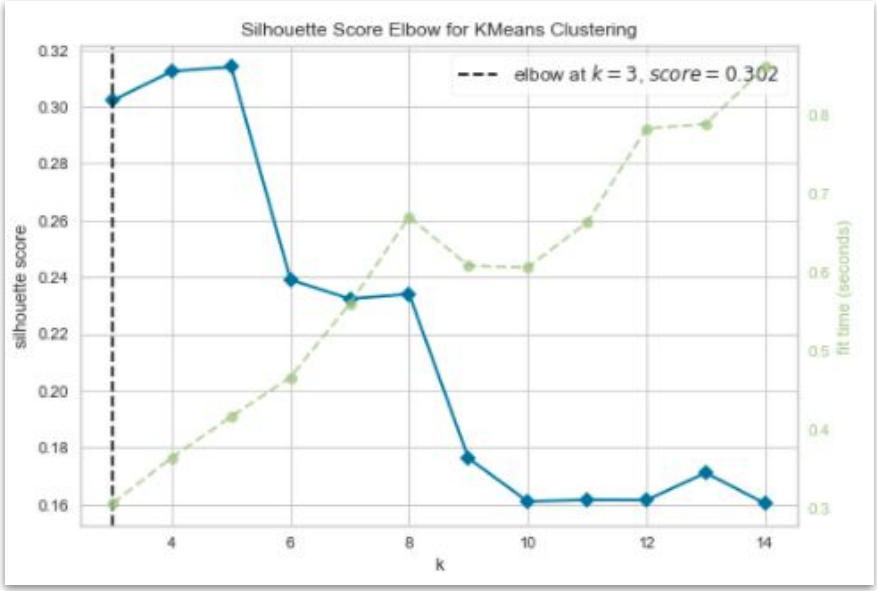
Méthodologies d’optimisation des hyperparamètres

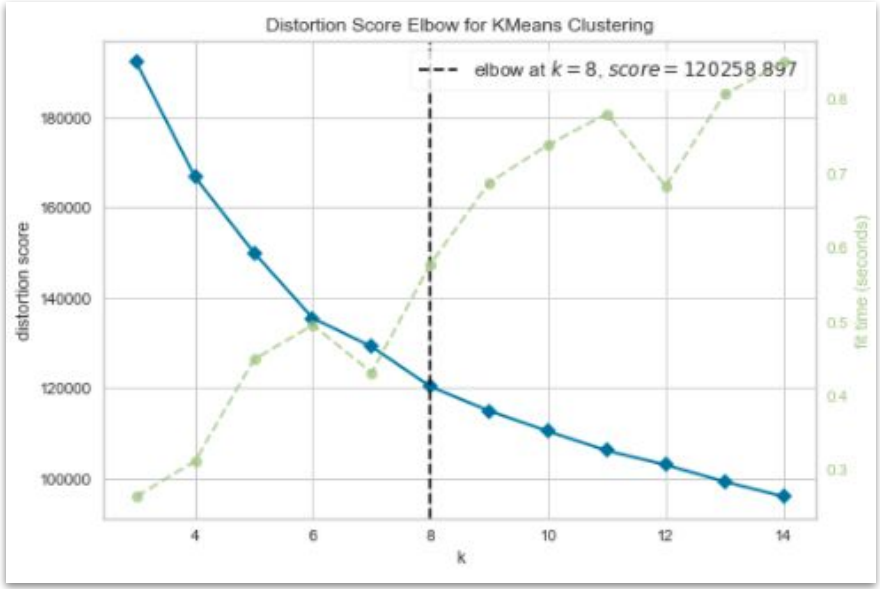
Nous avons utilisé le package Yellowbrick pour voir les paramètres optimaux dans le cas d’un K-means. Nous avons regardé plusieurs indicateurs dont le silhouette score.

Globalement, les paramètres optimaux restent stables au fur et à mesure des essais avec des résultats entre 7 et 9 clusters dans le cadre du Kmeans.

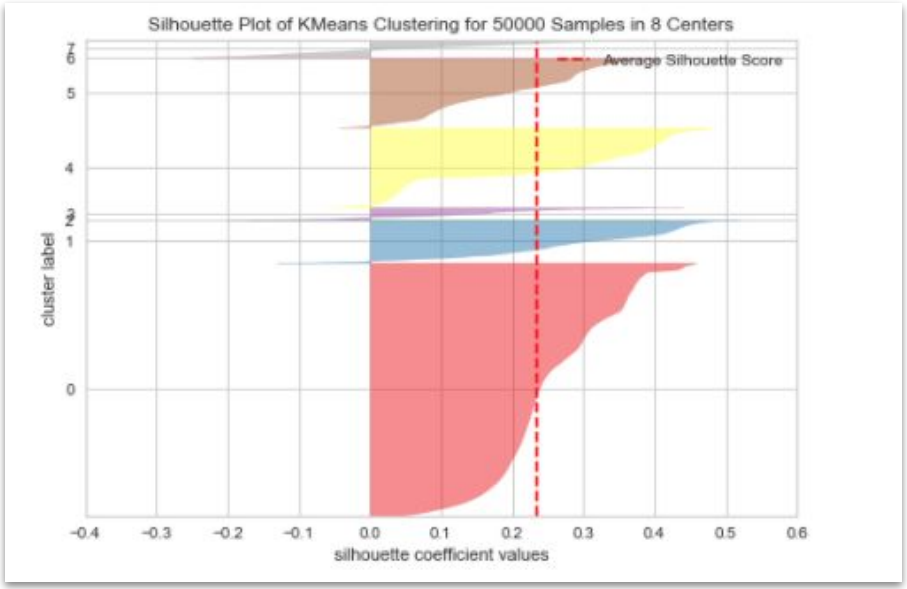
DBScan ne semble pas fonctionner car le nombre de cluster est extrêmement sensible à la moindre variation des paramètres. (Multiplication par 10x du nombre de clusters pour une variation “epsilonesque”)

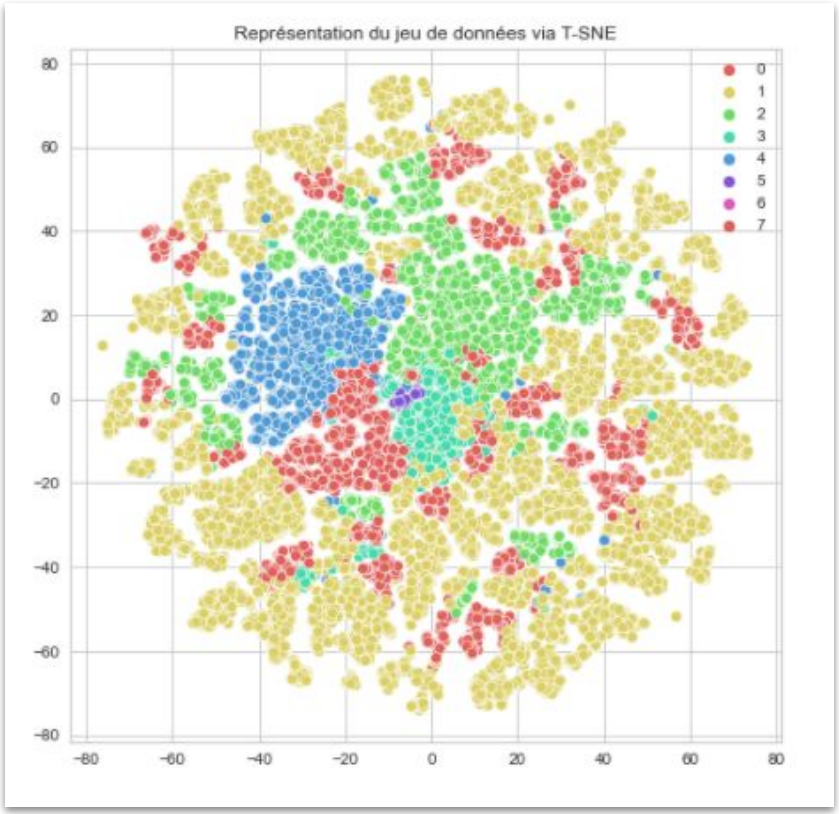
K-Means: recherche des paramètres optimaux



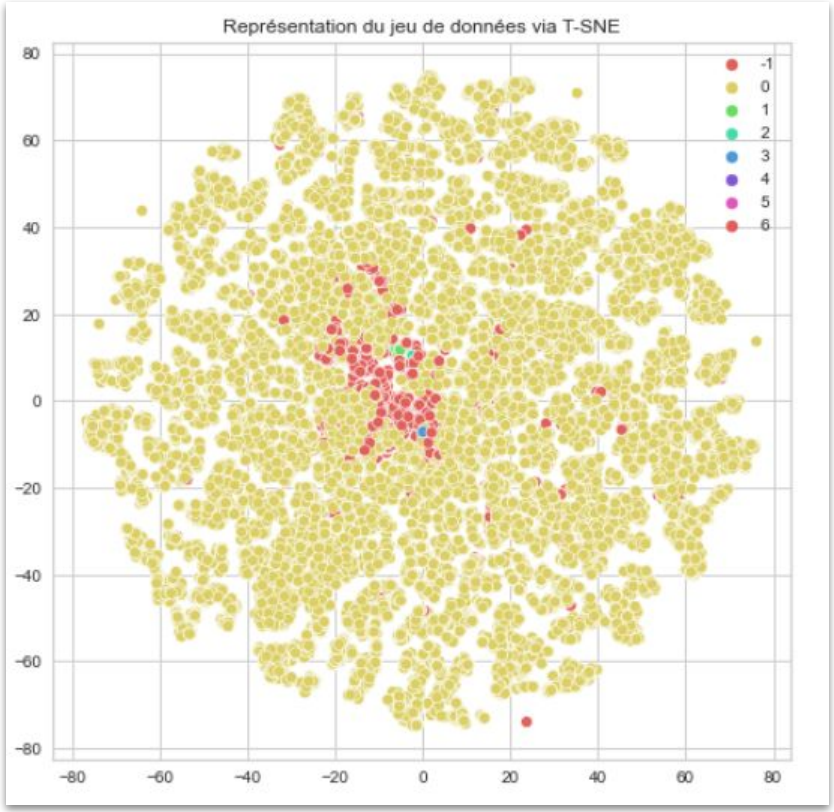


K-Means: profil et projections t-SNE des clusters





DBSCAN: pas efficace avec nos données



Commentaire:

- Semble ne pas bien fonctionner pour notre dataset, car l'algorithme est hautement sensible aux variations dans nos paramètres.

Caractéristiques des clusters K-means

Cluster_k_means	customer_unique_id	payment_value	review_score	freight_value	geolocation_lat	geolocation_lng
0	7251	159.228766	4.627369	22.135478	-26.308888	-51.063167
1	26636	147.598132	4.754631	18.223443	-22.128612	-45.740824
2	8363	187.065919	1.876095	21.620673	-22.593986	-46.195115
3	1790	1096.850425	4.107496	93.746631	-21.386161	-46.296863
4	4465	206.666347	3.998227	34.423489	-8.735784	-38.404434
5	143	3490.662937	3.464948	295.220979	-17.444645	-44.935304
6	12	24602.903333	1.916667	490.819167	-19.075194	-46.357922
7	1340	219.474903	4.142155	36.720366	-7.744368	-53.560371

Remarques concernant les clusters les plus significatifs

Cluster 1: Clients récurrents. Très heureux d'utiliser la plateforme.

Cluster 2: Clients "à travailler". Mauvaise expérience avec la plateforme.

Cluster 0: Clients récurrents. Très heureux d'utiliser la plateforme.

Cluster 4: Clients à fort pouvoir d'achat. Travail de fidélisation.

Actions:

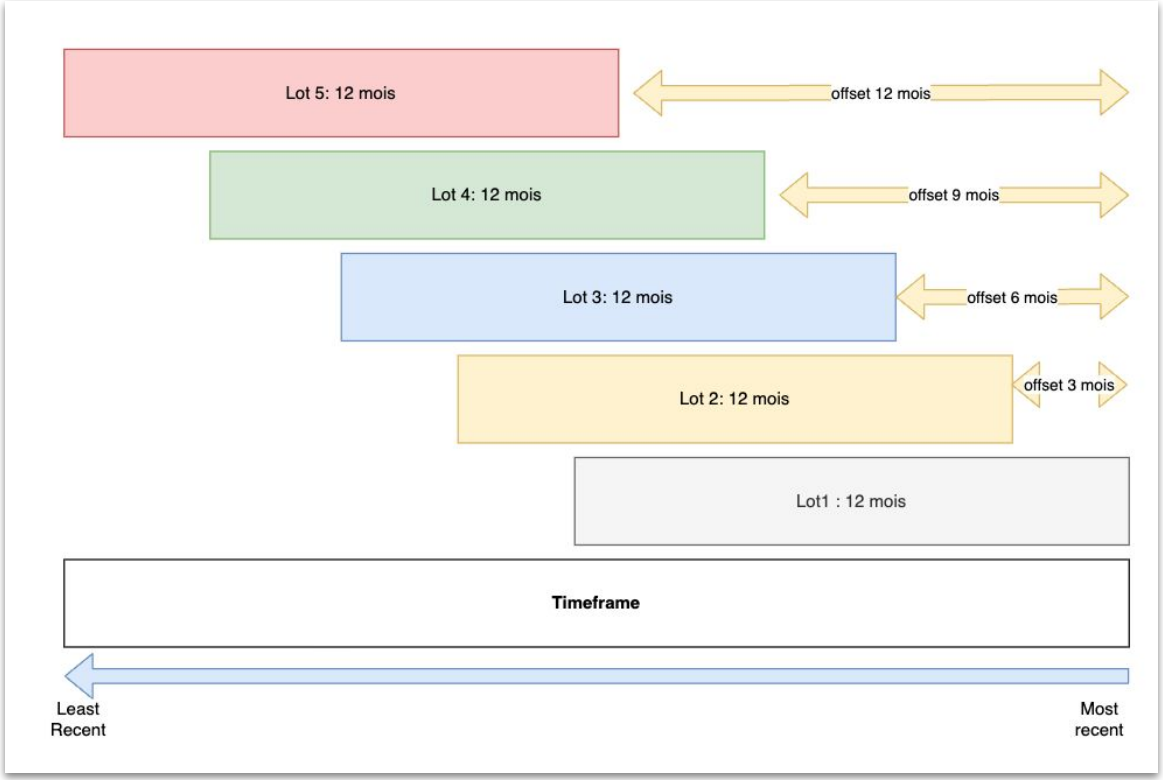
(i) Cluster 0 & 1 : Abonnement type Amazon

(ii) Cluster 2: Proposer des rabais + gratuité des coûts de transport pour compenser la mauvaise expérience

(iii) Cluster 4: Proposer des bundle pour augmenter le volume d’affaires

Etude de sensibilité temporelle: Méthodologie 1/3

- Découpage en lot de 12 mois avec des offsets de 0, 3, 6, 9 et 12 mois
- Les lots de 12 mois permettent d'éviter les problèmes de saisonnalité, et garder des lots de tailles significatives



Etude de sensibilité temporelle: Méthodologie 2/3

- Chaque lot est ensuite pre-procéssé: encodage avec Multilabelbinarizer, PCA, et Scaling des features numériques

In [1079]:

```
def full_prep(df_to_ml):
    """Takes a dataframe to be prepared for Machine Learning. Returns a new dataframe that is full prepared
    for Machine Learning. We use Multilabel binarizer, PCA and Standard Scalers"""
    #1 - Using Multilabel binarizer on product_category for each of the data windows
    mlb = MultiLabelBinarizer()
    mlb.fit_transform(df_to_ml['product_category_name'])
    prod_df = pd.DataFrame(mlb.fit_transform(df_to_ml['product_category_name']),
                           columns=mlb.classes_,
                           index=df_to_ml['product_category_name'].index)

    #2 - PCA on the output binary features of Binarizer
    pca = PCA(n_components = 0.8)
    prod_pca = pca.fit_transform(prod_df)
    n_comp = prod_pca.shape[1]
    print("Number of components is: " + str(n_comp))

    pca_labels =[]
    for x in range(n_comp):
        pca_labels.append('pca_comp'+str(x))
    prod_pca_df = pd.DataFrame(prod_pca, columns=pca_labels)

    #3 - Scaling of numerical features
    numerics = ['payment_value','review_score','freight_value','geolocation_lat','geolocation_lng']
    ss = StandardScaler()
    df_to_ml_ss = df_to_ml.copy()
    df_to_ml_ss[numerics] = ss.fit_transform(df_to_ml_ss[numerics])
    prod_pca_df['customer_unique_id'] = df_to_ml_ss['customer_unique_id']

    #4 - Merging the two outputs
    final_df_to_ml = pd.merge(df_to_ml_ss,prod_pca_df,on='customer_unique_id',how='left')
    return final_df_to_ml
```

Etude de sensibilité temporelle: Méthodologie 3/3

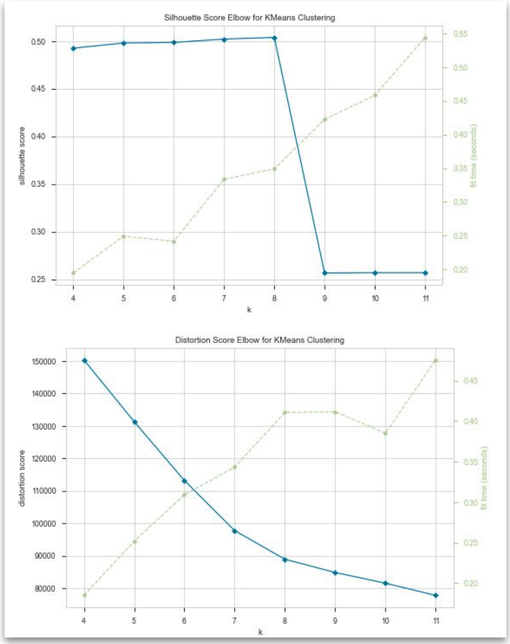
- On fait ensuite tourner les algorithmes K_means combiné avec le package Yellowbrick pour optimiser les paramètres

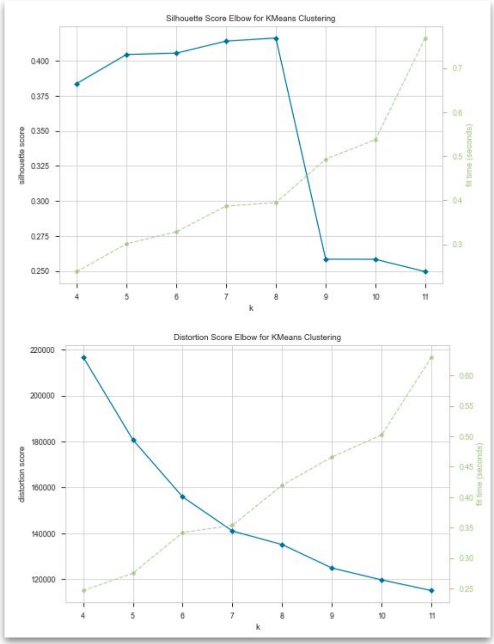
```
def full_viz(df_to_ml):
    """Takes a pre-processed database as input. Returns a database containing the data that is useful
    for clustering only. Prints out the graphs that shows the optimal k_mean parameter"""
    df = df_to_ml.copy()
    n = len(df)

    #We are dropping the useless columns
    df_cleaned = df.drop(columns=['customer_unique_id','customer_state','payment_type','product_category_name', 'order_
    df_cleaned.reset_index(inplace=True,drop=True)

    #Initialising the graphs that allow to find the optimal parameters
    model = KMeans()
    visualizer = KElbowVisualizer(model, k=(4,12),metric='silhouette',locate_elbow=False)

    visualizer.fit(df_cleaned)    # Fit the data to the visualizer
    visualizer.poof()

    visualizer = KElbowVisualizer(model, k=(4,12),locate_elbow=False)
    visualizer.fit(df_cleaned)    # Fit the data to the visualizer
    visualizer.poof()

    return df_cleaned
```

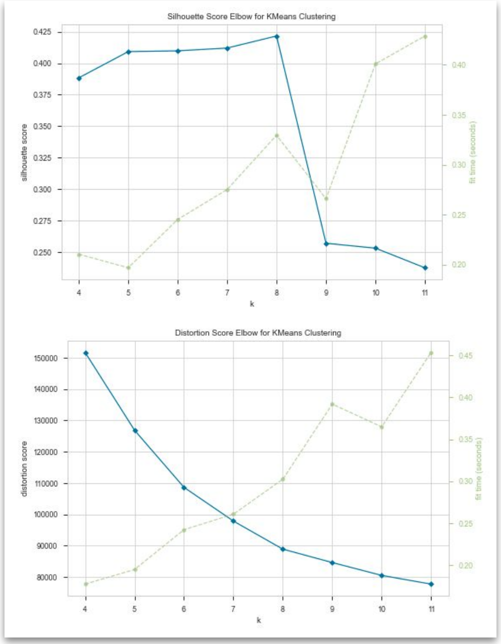
```
ef full_clusters(cleaned_df, opt_k_means, original_df):
    """
    Takes a ready for clustering dataFrame, an optimal k_means parameter, and the original df before preprocessing
    Returns a clustered_df containing the clusters for each ID, and the customer_unique_ID.
    """
    k_means = KMeans(opt_k_means)

    #Showing the silhouette graph for the k_means model
    visualizer = SilhouetteVisualizer(k_means)
    visualizer.fit(cleaned_df)    # Fit the data to the visualizer
    visualizer.poof()

    #We are adding the labels of the cluster to the df, and we add also the customer_unique_id
    k_means.fit(cleaned_df.copy())
    clusters_k_means = k_means.labels_
    clustered_df_clean = cleaned_df.assign(Cluster=clusters_k_means, customer_unique_id=original_df['customer_unique_id'
    return clustered_df_clean
```

Etude de sensibilité temporelle: Résultats 1/2

- **Résultat 1**: Tous les lots ont des nombres composantes de PCA et nombre de clusters optimales similaires.







Etude de sensibilité temporelle: Résultats 2/2

- **Les adj. Rand scores** sont calculés sur les “customer_unique_id” présents sur les deux lots à analyse. Ci dessous la matrice des rand score en fonction des différents lots.
- **Résultat 2:** Les clusters ont une très bonne stabilité au fil du temps, jusqu’à l’offset de 12 mois, où les adj_rand_score se dégradent significativement. Deux explications complémentaires se présentent : la dégradation est due soit à un changement de comportement de la part des clients, soit que l'échantillon de l'intersection est trop petit présentant du bruit. Dans le deuxième cas cela veut dire qu'un nombre important de nouveaux clients s'est ajouté. Les deux hypothèses suggèrent en tout cas de refaire un clustering à ce stade, et un nouveau contrat est préconisé tout les ans.

```
res_matrix
# Adj rand score matrix
```

	df_1_0m	df_2_3m	df_3_6m	df_4_9m	df_5_12m
df_1_0m	1.000000	0.999975	0.999962	0.999942	0.648499
df_2_3m	0.999975	1.000000	0.999990	0.999982	0.676781
df_3_6m	0.999962	0.999990	1.000000	0.999997	0.679097
df_4_9m	0.999942	0.999982	0.999997	1.000000	0.680216
df_5_12m	0.648499	0.676781	0.679097	0.680216	1.000000

Améliorations possibles

- Utilisation de NLP pour les commentaires afin d'évaluer les tendances de celles-ci, en fonction du nombre de commandes, prix moyen, dépenses total, etc…
- Essayer un clustering "manuel" pour comparer avec les résultats de nos algorithmes?
- Un cluster regroupe la grosse majorité des clients: peu d'apports des algorithmes? Nouvelles features nécessaires?

Proposition commerciale contrat de maintenance

Semble raisonnable de faire une mise à jour annuelle (12 mois) de cette base de données, afin d'évaluer l'évolution de la base de clients Olist.

- RDV pour l'année prochaine concernant la mise à jour majeure
- Contrat de maintenance basé sur un forfait à la journée
- Tout travail en dehors de ce périmètre devra être discuté upfront avec le client afin d'éviter des mauvaises surprises avec la facturation

Merci de votre attention!