Projet 4

Brazilian E-Commerce Public Dataset by Olist

Jeudi 21 mars 2024

Sommaire

Positionnement

Partie I - Requêtes SQL

Requête 1

En excluant les commandes annulées, quelles sont les commandes récentes de moins de 3 mois que les clients ont reçues avec au moins 3 jours de retard ?

```
SELECT *, julianday(order_delivered_customer_date) - julianday(order_estimated_delivery_date) AS days_too_late
FROM orders
WHERE
    order_status <> 'canceled'
  AND order_purchase_timestamp >= DATE((SELECT max(order_purchase_timestamp)
                                        from orders), '-3 months')
  AND order_purchase_timestamp IS NOT NULL
  AND order_delivered_customer_date IS NOT NULL
  AND days_too_late > 3
ORDER BY days_too_late DESC;
```

Requête 2

Qui sont les vendeurs ayant généré un chiffre d'affaires de plus de 100 000 Real sur des commandes livrées via Olist ?

```
SELECT seller_id, SUM(price) as total_revenue
FROM order_items
GROUP BY seller_id
HAVING total_revenue > 100000;
```

Requête 3

Qui sont les nouveaux vendeurs (moins de 3 mois d'ancienneté) qui sont déjà très engagés avec la plateforme (ayant déjà vendu plus de 30 produits) ?

```
SELECT
    oi.seller_id,
    COUNT(DISTINCT o.order_id) as total_orders_per_seller,
    min(order_purchase_timestamp) as date_premiere_commande
FROM
    order_items oi
        JOIN
    orders o ON oi.order_id = o.order_id
GROUP BY
    oi.seller_id
HAVING
    total_orders_per_seller > 30
    AND date_premiere_commande >= DATE((SELECT max(order_purchase_timestamp)
                                        from orders), '-3 months');
```

Requête 4

Quels sont les 5 codes postaux, enregistrant plus de 30 commandes, avec le pire review score moyen sur les 12 derniers mois ?

```
SELECT customer_zip_code_prefix, COUNT(o.order_id) as nombre_commande, AVG(review_score) as moyenne_score
FROM orders
JOIN customers c on orders.customer_id = c.customer_id
JOIN order_reviews o on orders.order_id = o.order_id
WHERE
    order_purchase_timestamp >= DATE((SELECT max(order_purchase_timestamp)
                                      from orders), '-12 months')
GROUP BY customer_zip_code_prefix
HAVING
    nombre_commande > 30
ORDER BY moyenne_score
LIMIT 5;
```

Partie II - Modélisations

Cleaning et analyse exploratoire

```
# Analyse prix des commandes
mean_price_df_order_items = df_order_items["price"].mean()
min_price_df_order_items = df_order_items["price"].min()
max_price_df_order_items = df_order_items["price"].max()

print(f"Mean price of orders: {mean_price_df_order_items}")
print(f"Minimum price of orders: {min_price_df_order_items}")
print(f"Maximum price of orders: {max_price_df_order_items}")
```

Executed at 2024.03.18 10:28:26 in 42ms

```
Mean price of orders: 120.65373901464716
Minimum price of orders: 0.85
Maximum price of orders: 6735.0
```

```
# Analyse type de paiement
payment_type_counts = df_order_pymts['payment_type'].value_counts()

print(payment_type_counts)
```

Executed at 2024.03.18 10:28:26 in 42ms

```
payment_type
credit_card     76795
boleto          19784
voucher          5775
debit_card       1529
not_defined         3
Name: count, dtype: int64
```

Modèles utilisés et features

Features :

- Récence
- Fréquence
- Montant
- Average review score

Modèles

- K-means
- DBSCAN

Features

Récence

```
def creation_analyse_rfm (dataframe_periode, end_date):

    # Pour la récence
    # Trouver la date de la commande la plus récente pour chaque customer_unique_id
    latest_order_date_df = dataframe_periode.groupby('customer_unique_id')
     ['order_purchase_timestamp_datetime'].max().reset_index()
    latest_order_date_df.columns = ['customer_unique_id', 'latest_order_date']

    # Calculer la récence en jours jusqu'à la fin de la période spécifiée
    latest_order_date_df['recency'] = (pd.to_datetime(end_date) -
     latest_order_date_df['latest_order_date']).dt.days
```

Features

Fréquence

```
# Pour la fréquence
# Calculer la fréquence d'achat par client
frequency_df = dataframe_periode.groupby('customer_unique_id')['order_id'].nunique()
 .reset_index()
frequency_df.columns = ['customer_unique_id', 'frequency']
```

Features

Montant

```
# Pour le montant
# Calculer le montant total dépensé par commande pour chaque client_unique_id
total_amount_spent = dataframe_periode.groupby('customer_unique_id')['payment_value']
 .sum()

# Calculer le nombre total de commandes pour chaque client_unique_id
total_orders = dataframe_periode.groupby('customer_unique_id')['order_id'].nunique()

# Calculer la moyenne par commande pour chaque client_unique_id
average_spending_per_order = total_amount_spent / total_orders

# Créer un DataFrame avec les résultats
average_spending_df = pd.DataFrame({'customer_unique_id': average_spending_per_order
 .index, 'average_spending_per_order': average_spending_per_order.values})
```

Features

Average review score

```
# Pour la moyenne du review_score par client
average_score_per_customer = merged_df.groupby('customer_unique_id')['review_score']
 .mean().reset_index()
```

K-means

- Standardisation
- Optimisation du nombre de clusters :
    - Elbow curve
    - Score de silhouette
- Entraînement
- Visualisation des clusters

K-means

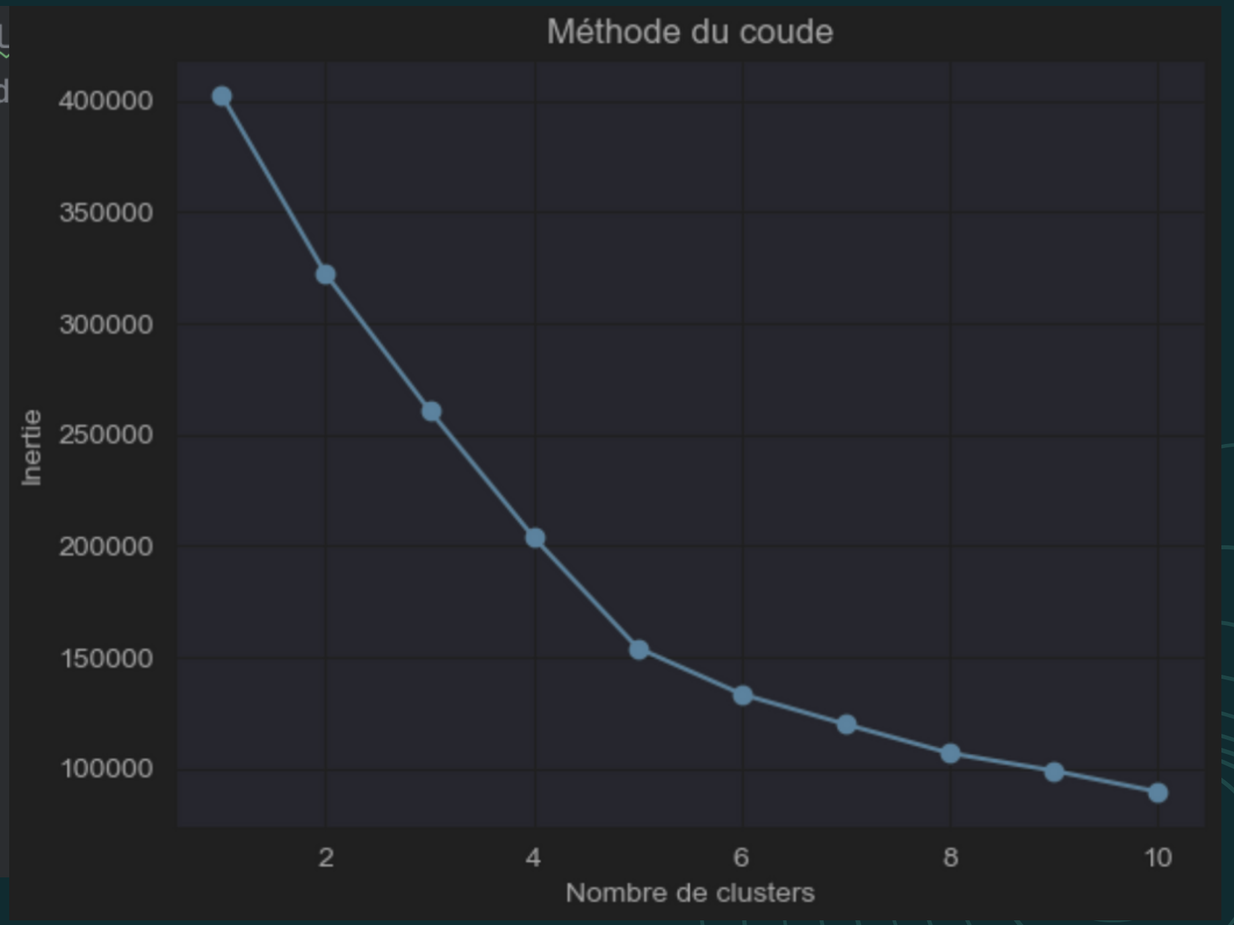
Standardisation

```
# Créer un objet StandardScaler
scaler = StandardScaler()

# Standardisation de filtered_combined_df
rfm_df_2016_2018_filtered_standardized = pd.DataFrame(scaler.fit_transform
 (rfm_df_2016_2018_filtered), columns=rfm_df_2016_2018_filtered.columns)

rfm_df_2016_2018_filtered_standardized.head()
```

K-means

Optimisation du nombre de clusters : elbow curve

```python
# Optimisation du nombre de clusters pour l'entraînement du modèl
# Initialise une liste pour stocker l'inertie (somme des carrés d
inertia = []

# Test des différents nombres de clusters
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(rfm_df_2016_2018_filtered_standardized)
    inertia.append(kmeans.inertia_)

# Elbow curve
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Nombre de clusters')
plt.ylabel('Inertie')
plt.title('Méthode du coude')
plt.show()
```



K-means

Optimisation du nombre de clusters : score de silhouette

```
# Calcul du score de silhouette kmeans_2016_2018
silhouette_avg_kmeans_2016_2018 = silhouette_score
 (rfm_df_2016_2018_filtered_standardized, cluster_labels_2016_2018)
print("Score de silhouette :", silhouette_avg_kmeans_2016_2018)
```

Executed at 2024.03.08 15:55:08 in 2m 29s 303ms

```
 Score de silhouette : 0.5206395329986259
```

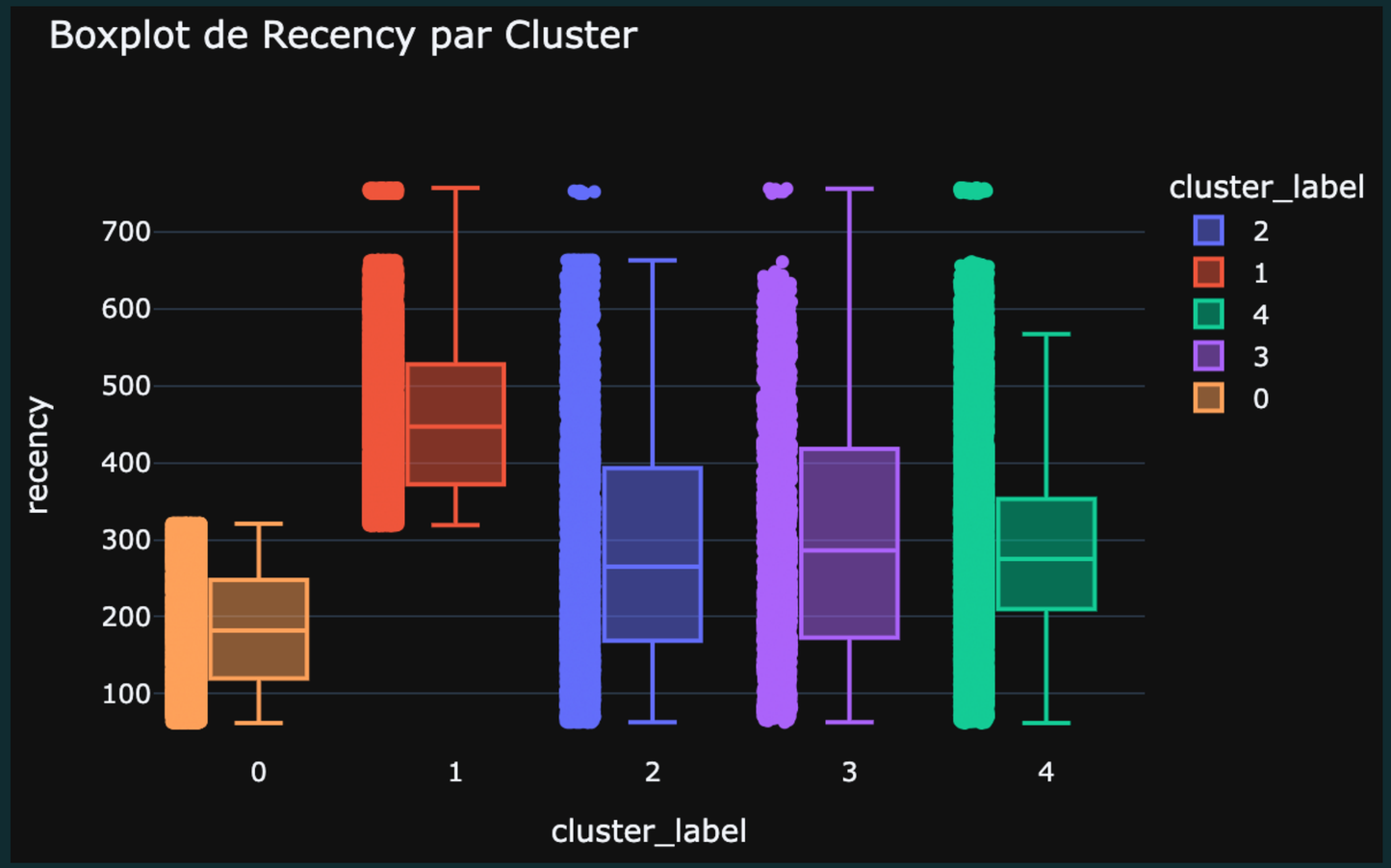
K-means

Entraînement

```
# Entraînement modèle kmeans
# Nombre de clusters
n_clusters = 5

# Instanciation d'un objet KMeans
kmeans_2016_2018 = KMeans(n_clusters=n_clusters, random_state=42)

# Clustering sur les données
cluster_labels_2016_2018 = kmeans_2016_2018.fit_predict
 (rfm_df_2016_2018_filtered_standardized)

# Ajout des étiquettes de cluster au DataFrame
rfm_df_2016_2018_filtered_standardized['cluster_label'] = cluster_labels_2016_2018

rfm_df_2016_2018_filtered_standardized.head()
```

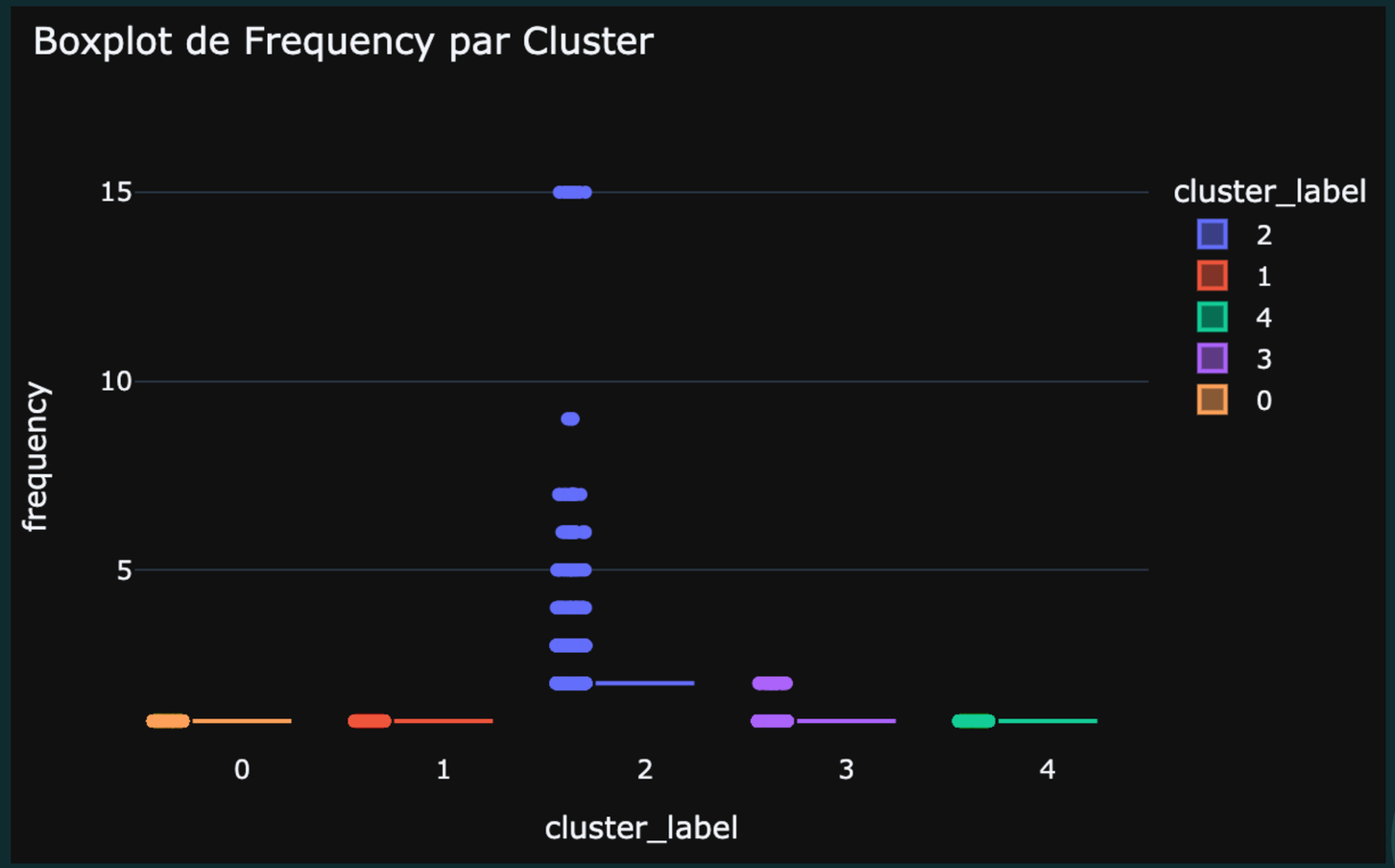
K-means

Visualisation des clusters



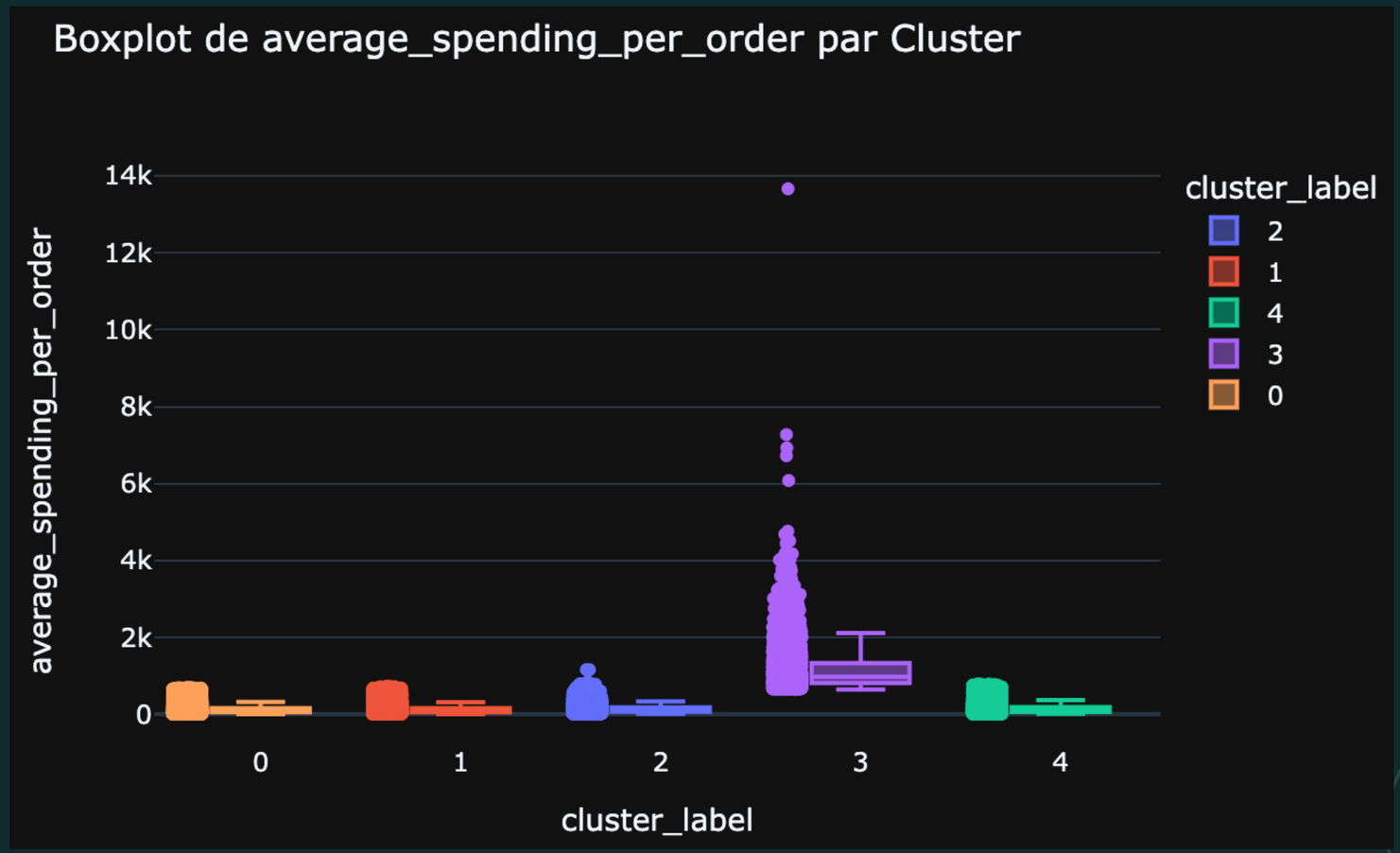
K-means

Visualisation des clusters



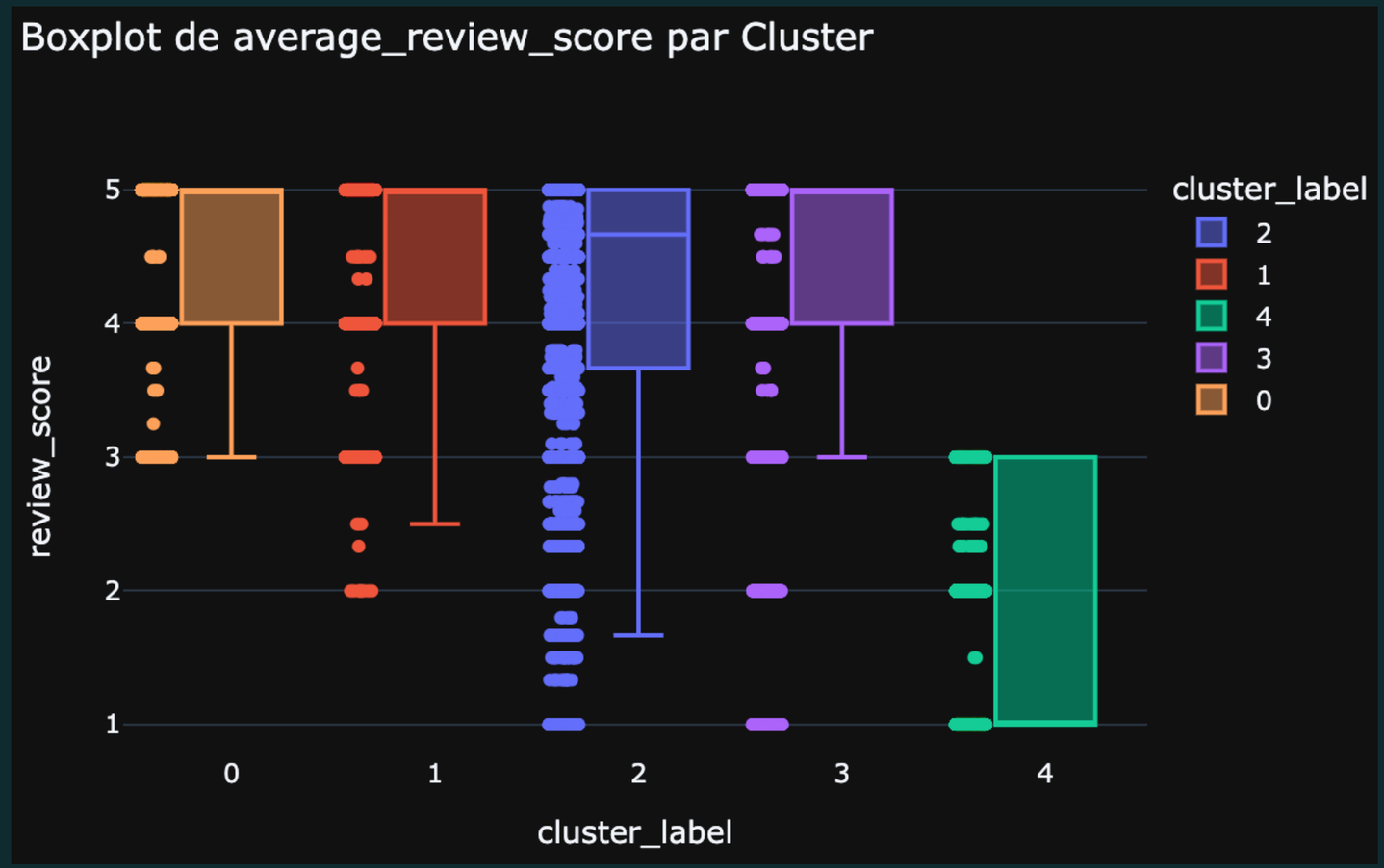
K-means

Visualisation des clusters



K-means

Visualisation des clusters



DBSCAN

- Standardisation
- Optimisation du nombre de clusters :
    - eps
    - min_samples
- Entraînement
- Visualisation des clusters

DBSCAN

Optimisation des hyperparamètres : score de silhouette

```python
from sklearn.cluster import DBSCAN
from sklearn.model_selection import ParameterGrid

# Grille d'hyperparamètres
param_grid = {
    'eps': [0.1, 0.5, 1.0],
    'min_samples': [5, 10, 20]
}

best_score = -1
best_params = None

# Gridsearch
for params in ParameterGrid(param_grid):
    dbscan = DBSCAN(**params)
    dbscan.fit(rfm_df_2016_2018_filtered_standardized)
    labels = dbscan.labels_
    score = silhouette_score(rfm_df_2016_2018_filtered_standardized, labels)

    if score > best_score:
        best_score = score
        best_params = params

print("Hyperparamètres:", best_params)
```

Executed at 2024.03.08 16:34:40 in 33m 15s 919ms

```
Hyperparamètres: {'eps': 1.0, 'min_samples': 10}
```

DBSCAN

Entraînement

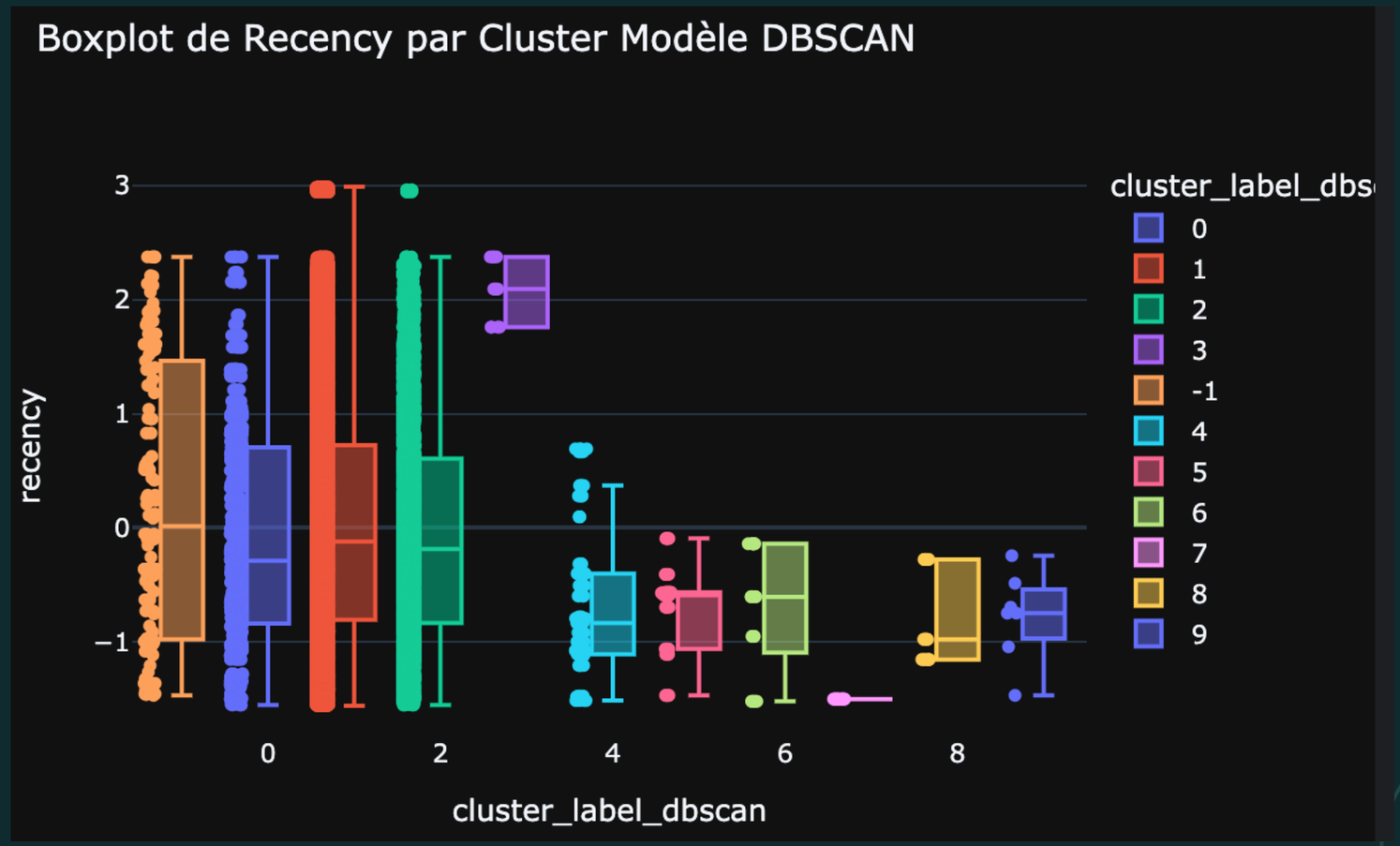
```
# Entraînement
dbscan = DBSCAN(eps=1.0, min_samples=10)

# Clustering
cluster_label_dbscan = dbscan.fit_predict(rfm_df_2016_2018_filtered_standardized_dbscan)

# Etiquettes de cluster
rfm_df_2016_2018_filtered_standardized_dbscan['cluster_label_dbscan'] =
 cluster_label_dbscan

# Calcul du score de silhouette
silhouette_avg = silhouette_score(rfm_df_2016_2018_filtered_standardized_dbscan,
 cluster_label_dbscan)
print("Score de silhouette :", silhouette_avg)
```

Executed at 2024.03.08 16:45:03 in 6m 31s 685ms

```
Score de silhouette : 0.4351535908387644
```

DBSCAN

Visualisation des clusters



Partie III - Maintenance

Fonctions utilisées

```
# Partie maintenance du modèle
def creation_data_frame_par_periode (start_date, end_date, merged_df_cleaned) :
    """Crée un dataframe pour la période spécifiée :
    start_date : début de la période souhaitée
    end_date : fin de la période souhaitée
     merge_df_cleaned = dataframe contenant les données sur lesquels on souhaite
      travailler"""

    # Crée un dataframe pour la période spécifiée
    filtered_df = merged_df_cleaned[
     (merged_df_cleaned['order_purchase_timestamp_datetime'] >= start_date) &
     (merged_df_cleaned['order_purchase_timestamp_datetime'] <= end_date)].copy()

    return filtered_df
```

Fonctions utilisées

```python
def creation_analyse_rfm (dataframe_periode, end_date):
    """Fonction réalisant une analyse rfm + average review score sur le dataframe spécifié
    dataframe_periode : dataframe de la période souhaitée sur lequel on souhaite réaliser
     l'analyse
    end_date : dernier jour de la période souhaitée
     """
    # Pour la récence
    # Trouve la date de la commande la plus récente pour chaque customer_unique_id
    latest_order_date_df = dataframe_periode.groupby('customer_unique_id')
     ['order_purchase_timestamp_datetime'].max().reset_index()
    latest_order_date_df.columns = ['customer_unique_id', 'latest_order_date']

    # Calcul la récence en jours jusqu'à la fin de la période spécifiée
    latest_order_date_df['recency'] = (pd.to_datetime(end_date) -
     latest_order_date_df['latest_order_date']).dt.days

    # Pour la fréquence
    # Calcul la fréquence d'achat par client
    frequency_df = dataframe_periode.groupby('customer_unique_id')['order_id'].nunique()
     .reset_index()
    frequency_df.columns = ['customer_unique_id', 'frequency']

    # Pour le montant
    # Calcul le montant total dépensé par commande pour chaque client_unique_id
    total_amount_spent = dataframe_periode.groupby('customer_unique_id')['payment_value']
     .sum()

    # Calcul le nombre total de commandes pour chaque client_unique_id
    total_orders = dataframe_periode.groupby('customer_unique_id')['order_id'].nunique()

    # Calcul la moyenne par commande pour chaque client_unique_id
    average_spending_per_order = total_amount_spent / total_orders
```

```python
    # Crée un DataFrame avec les résultats
    average_spending_df = pd.DataFrame({'customer_unique_id': average_spending_per_order
     .index, 'average_spending_per_order': average_spending_per_order.values})

    # Pour la moyenne du review_score par client
    average_score_per_customer = merged_df.groupby('customer_unique_id')['review_score']
     .mean().reset_index()

    # Fusion des DataFrames sur 'customer_unique_id'
    rfm_df = pd.merge(frequency_df, latest_order_date_df, on='customer_unique_id',
     how='left')
    rfm_df = pd.merge(rfm_df, average_spending_df, on='customer_unique_id', how='left')
    rfm_df = pd.merge(rfm_df, average_score_per_customer, on='customer_unique_id',
     how='left')

    # Ajoute la colonne 'order_purchase_timestamp_datetime'
    rfm_df = pd.merge(rfm_df, dataframe_periode[['customer_unique_id',
     'order_purchase_timestamp_datetime']], on='customer_unique_id', how='left')

    # Trie le dataframe en fonction de la colonne 'order_purchase_timestamp_datetime'
    rfm_df.sort_values(by='order_purchase_timestamp_datetime', inplace=True)

    return rfm_df
```

Maintenance

Entraînement du modèle de référence sur les données de 2016 à 2018

```python
# Maintenance
from datetime import datetime, timedelta
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score

# Date d'initialisation
init_date = datetime(2018,1,1)

# Création du dataframe pour les données de 2016 à 2017
df_1 = creation_data_frame_par_periode(start_date="2015-01-01", end_date=init_date,
 merged_df_cleaned=merged_df_cleaned)

# Création des analyses rfm pour le dataframe de référence
rfm_1 = creation_analyse_rfm(dataframe_periode=df_1, end_date=init_date)

# Suppression des colonnes non pertinentes
columns_to_exclude = ['customer_unique_id', "latest_order_date",
 "order_purchase_timestamp_datetime"]

# Entraînement du modèle Kmeans pour le dataframe de référence
kmeans_1 = KMeans(n_clusters=5, random_state=42).fit(rfm_1.drop(columns=columns_to_exclude))
```

Boucle for pour comparer les clusterings par rapport au clustering fait par le modèle de référence

```python
# Initialisation de la liste pour stocker les scores ARI
ari_scores = []

# Boucle créant un dataframe par mois jusqu'au mois d'août 2018 (date de la dernière
 commande), puis création des analyses rfm, entraînement des modèles et comparaison de la
 similarité des modèles de 2016-2017
for i in range(8):
    end_date = init_date + timedelta(days=i*30)
    df_2 = creation_data_frame_par_periode(start_date="2015-01-01", end_date=end_date,
     merged_df_cleaned=merged_df_cleaned)

    # Création des analyses rfm pour le dataframe actuel
    rfm_2 = creation_analyse_rfm(dataframe_periode=df_2, end_date=end_date)

    # Entraînement du modèle Kmeans pour le dataframe actuel
    kmeans_2 = KMeans(n_clusters=5, random_state=42).fit(rfm_2.drop
     (columns=columns_to_exclude))

    # Prédictions du modèle sur les nouvelles données
    predict_reference = kmeans_1.predict(rfm_2.drop(columns=columns_to_exclude))
    predict_2 = kmeans_2.predict(rfm_2.drop(columns=columns_to_exclude))

    # Comparaison des modèles grâce à l'Adjusted Rand Score et stockage du score dans la
     liste
    ari = adjusted_rand_score(predict_reference, predict_2)
    ari_scores.append(ari)

# Affichage des scores ARI pour chaque itération
for i, ari in enumerate(ari_scores):
    print(f"Iteration {i+1} - Adjusted Rand Score entre le modèle de référence et le modèle
     {i+1}: {ari}")
```

Comparaison des clusterings par période

Score ARI

Au bout d'un mois	Au bout de deux mois	Au bout de trois mois	Au bout de quatre mois	Au bout de cinq mois	Au bout de six mois	Au bout de sept mois
1.0	0.91	0.73	0.73	0.57	0.47	0.40

Conclusion