Segmentez des clients d'un site e-commerce

Antoine Rota-Nodari

Présentation

📌 Olist, c'est quoi ?

- Olist est une marketplace brésilienne qui connecte des vendeurs indépendants et des boutiques en ligne à des plateformes e-commerce comme Amazon, Mercado Libre et Magalu.

📦 Son fonctionnement :

- Permet aux vendeurs d’accéder à un réseau de marketplaces sans avoir à gérer la logistique ou le marketing.
- Propose des solutions logistiques et analytiques pour optimiser les ventes.

Objectifs et enjeux du projet

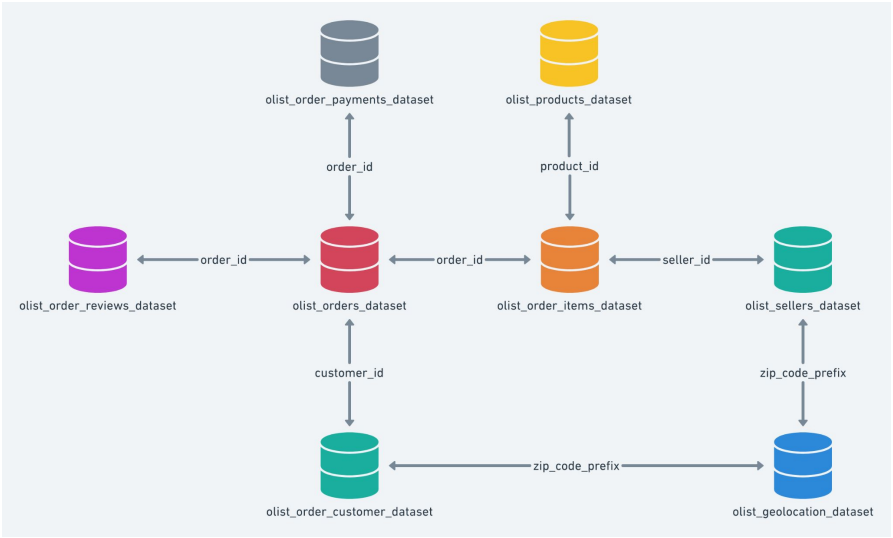
🎯 Objectif :

- Segmenter les clients d'Olist en fonction de leur comportement d'achat pour optimiser les actions marketing.

🚀 Enjeux :

- Améliorer la personnalisation des offres et la fidélisation client.
- Optimiser les coûts d'acquisition et la rentabilité.
- Automatiser la segmentation pour une mise en production efficace.

Schéma des bases de données fournies



DataFrame	Description
customers_df	Informations sur les clients (ID unique)
orders_df	Détails des commandes passées (dates, statuts)
order_items_df	Produits achetés dans chaque commande (prix, vendeurs)
payments_df	Montant et type de paiement utilisé
reviews_df	Avis laissés par les clients (notes, commentaires)

SQL

Requêtes SQL demandées par Fernanda

Contexte :

Fernanda a besoin de plusieurs requêtes SQL pour un dashboard d'analyse des commandes et performances des vendeurs sur Olist. Étant donné que BigQuery est l'outil que j'utilise quotidiennement au travail, les données ont été importées en CSV dans un dataset BigQuery avant d'être exploitées via SQL.

1️⃣ Commandes récentes en retard (hors annulées)

📌 Objectif : Trouver les commandes livrées avec ≥3 jours de retard dans les 3 derniers mois

```
SELECT

*,
DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date, DAY) AS day_dif_delivery

FROM `directed-advice-441114-q3.OCR5_ecommerce_segmentation.orders_dataset`

WHERE order_purchase_timestamp >= TIMESTAMP_SUB((SELECT MAX(order_purchase_timestamp) FROM
`directed-advice-441114-q3.OCR5_ecommerce_segmentation.orders_dataset`), INTERVAL 93 DAY)

AND order_status != 'canceled'

AND DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date, DAY) >= 3
```

2️⃣ Vendeurs générant >100K BRL sur des commandes livrées

📌 Objectif : Identifier les vendeurs les plus performants en termes de chiffre d'affaires

```sql
SELECT
   i.seller_id,
   ROUND(SUM(i.price),2) AS CA
FROM `directed-advice-441114-q3.OCR5_ecommerce_segmentation.order_items_dataset` i
JOIN `directed-advice-441114-q3.OCR5_ecommerce_segmentation.orders_dataset` o
ON i.order_id = o.order_id
WHERE o.order_status = 'delivered'
GROUP BY i.seller_id
HAVING SUM(i.price) > 100000
ORDER BY CA DESC;
```

3️⃣ Nouveaux vendeurs très engagés (ancienneté <3 mois, ≥30 produits vendus)

📌 Objectif : Détecter les vendeurs récemment inscrits ayant déjà un volume de ventes élevé

```sql
SELECT
 oi.seller_id,
 MIN(o.order_purchase_timestamp) AS first_sale_date,
 COUNT(oi.product_id) AS nb_products_sold
FROM `directed-advice-441114-q3.OCR5_ecommerce_segmentation.order_items_dataset` AS oi
JOIN `directed-advice-441114-q3.OCR5_ecommerce_segmentation.orders_dataset` AS o
 ON oi.order_id = o.order_id
GROUP BY oi.seller_id
HAVING
 DATE_DIFF(
   (SELECT MAX(order_purchase_timestamp) FROM `directed-advice-441114-q3.OCR5_ecommerce_segmentation.orders_dataset`),
   MIN(o.order_purchase_timestamp),
   DAY
 ) <= 93
 AND COUNT(oi.product_id) > 30
ORDER BY nb_products_sold DESC;
```

4️⃣ 5 codes postaux avec le pire review score moyen (≥30 reviews, sur 12 derniers mois)

📌 Objectif : Détecter les vendeurs récemment inscrits ayant déjà un volume de ventes élevé

```sql
WITH
 review_customer_order AS (
 SELECT
    o.customer_id,
    r.order_id,
    AVG(review_score) AS review_score,
    COUNT(review_score) AS nb_review,

 FROM
    `directed-advice-441114-q3.OCR5_ecommerce_segmentation.order_reviews_dataset` r
 JOIN
    `directed-advice-441114-q3.OCR5_ecommerce_segmentation.orders_dataset` o
 ON
    r.order_id = o.order_id
 WHERE review_creation_date >= TIMESTAMP_SUB((SELECT MAX(review_creation_date) FROM `directed-advice-441114-q3.OCR5_ecommerce_segmentation.order_reviews_dataset`), INTERVAL 365 DAY)
 GROUP BY
    o.customer_id,
    r.order_id
 )

SELECT
customer_zip_code_prefix,
ROUND(AVG(review_score), 2) AS avg_review,
SUM(nb_review) AS sum_nb_review

FROM review_customer_order r
JOIN `directed-advice-441114-q3.OCR5_ecommerce_segmentation.customers_dataset` c
ON r.customer_id = c.customer_id

GROUP BY customer_zip_code_prefix

HAVING sum_nb_review >= 30

ORDER BY avg_review

LIMIT 5
```

Python

Explication du DataFrame RFM(R) (Récence, Fréquence, Monétaire, Review)

📌 Objectif :

Le modèle RFM(R) est une extension du modèle RFM qui intègre un 4ème indicateur : le score moyen des avis clients (Review). Cette approche permet d'avoir une vision plus complète de l'engagement et de la satisfaction des clients.

Il repose sur quatre indicateurs clés :

1️⃣ **Récence (Recency, R)** → Nombre de jours depuis la dernière commande

2️⃣ **Fréquence (Frequency, F)** → Nombre total de commandes passées

3️⃣ **Monétaire (Monetary, M)** → Montant total dépensé

4️⃣ **Review (R)** → Score moyen des avis laissés

📌 Création du DataFrame RFM(R)

À partir des tables SQL disponibles (customers, orders, payments, reviews), nous avons effectué plusieurs jointures pour obtenir les indicateurs RFM(R) par client.

```
# Étape 1-4 : Jointures successives
merged_1 = pd.merge(customers_df, orders_df, on='customer_id', how='left')
merged_2 = pd.merge(merged_1, order_items_df, on='order_id', how='left')
merged_3 = pd.merge(merged_2, payments_df, on='order_id', how='left')
merged_4 = pd.merge(merged_3, reviews_df, on='order_id', how='left')

# Étape 5 : Agrégation RFM(R)
final_df = merged_4.groupby('customer_unique_id').agg({
    'order_id': pd.Series.nunique,     # Frequency (nb de commandes)
    'payment_value': 'sum',            # Monetary (total dépensé)
    'review_score': 'mean',            # Review (score moyen)
    'order_purchase_timestamp': 'max' # Last Purchase (dernière commande)
}).reset_index()

# Étape 6 : Calcul de la récence (Recency)
final_df['last_purchase'] = pd.to_datetime(final_df['last_purchase'])
reference_date = final_df['last_purchase'].max()  # Date d'analyse
final_df['Recency'] = (reference_date - final_df['last_purchase']).dt.days
```

💾 **Tables utilisées :**

📂 customers_df – Identifiants clients
📂 orders_df – Commandes
📂 order_items_df – Articles commandés
📂 payments_df – Paiements effectués
📂 reviews_df – Avis clients

🛠️ **Étapes :**

1️⃣ **Jointure des clients et commandes**
2️⃣ **Ajout des articles commandés (prix, frais de livraison)**
3️⃣ **Ajout des paiements**
4️⃣ **Ajout du score moyen des avis**
5️⃣ **Agrégation des données par client unique**
6️⃣ **Calcul de la récence (Recency)**

Premières lignes du DataFrame obtenu

customer_unique_id	Recency (Jours)	Frequency (Nb Commandes)	Monetary (BRL)	Review (Score moyen)
0000366f3b9a7992bf8c76cfdf3221e2	160	1	141.9	5
0000b849f77a49e4a4ce2b2a4ca5be3f	163	1	27.19	4
0000f46a3911fa3c0805444483337064	585	1	86.22	3
0000f6ccb0745a6a4b88665a16c9f078	369	1	43.62	4
0004aac84e0df4da2b147fca70cf8255	336	1	196.89	5

Feature Engineering

✅ **Gestion des valeurs manquantes**

```
# Imputation Review manquante par la moyenne
mean_review = final_df['Review'].mean()
final_df['Review'] = final_df['Review'].fillna(mean_review)
```

✅ **Transformation logarithmique (évite la dominance des valeurs extrêmes)**

```
# Log-transform
final_df['Recency_log']    = np.log1p(final_df['Recency'])
final_df['Frequency_log']  = np.log1p(final_df['Frequency'])
final_df['Monetary_log']   = np.log1p(final_df['Monetary'])
final_df['Review_log']     = np.log1p(final_df['Review'])
```

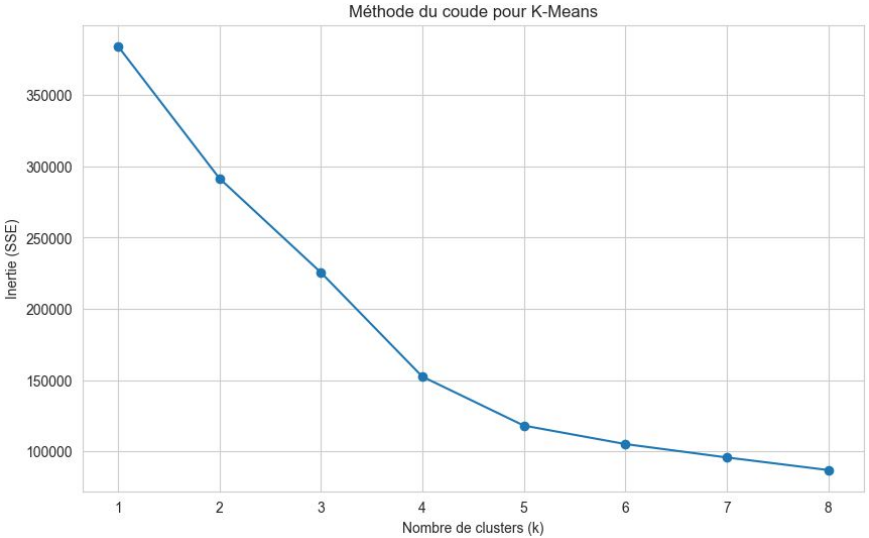
Feature Engineering

✅ **Standardisation (alignement des échelles pour le clustering)**

```
# Sélection pour le clustering
features = ['Recency_log', 'Frequency_log', 'Monetary_log', 'Review_log']
X_raw = final_df[features].copy()

# Normalisation
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)
```

Clustering K-Means : méthode du coude

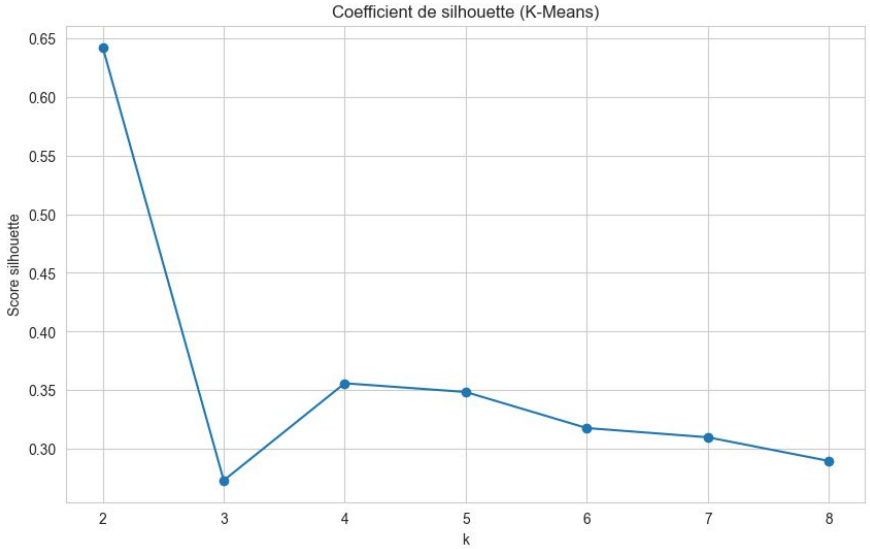


📊 **Objectif :** Trouver le nombre optimal de clusters en analysant l'inertie (SSE).

✅ **Principe :** Lorsque l'inertie commence à stagner, cela indique un bon équilibre entre variance intra-cluster et nombre de clusters.

➡ **Observation :** Le point d'inflexion ("coude") est visible autour de **k = 4**.

Clustering K-Means : coefficient de silhouette

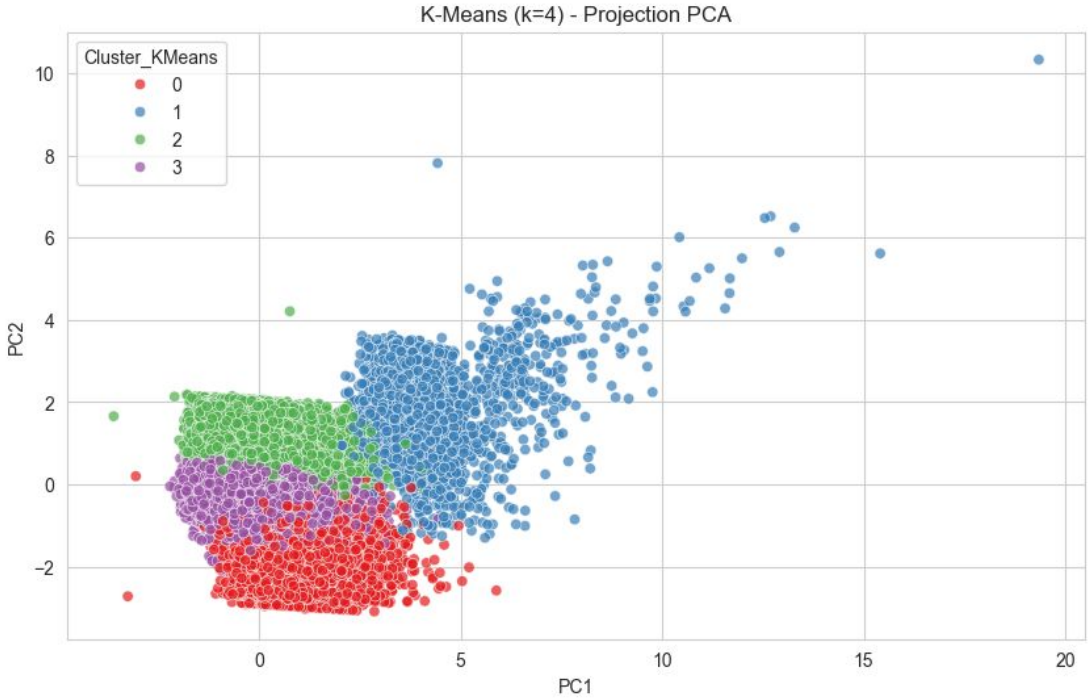


📊 **Objectif :** Mesurer la cohésion et la séparation des clusters.

✅ **Principe :** Un score élevé indique des clusters bien définis et distincts.

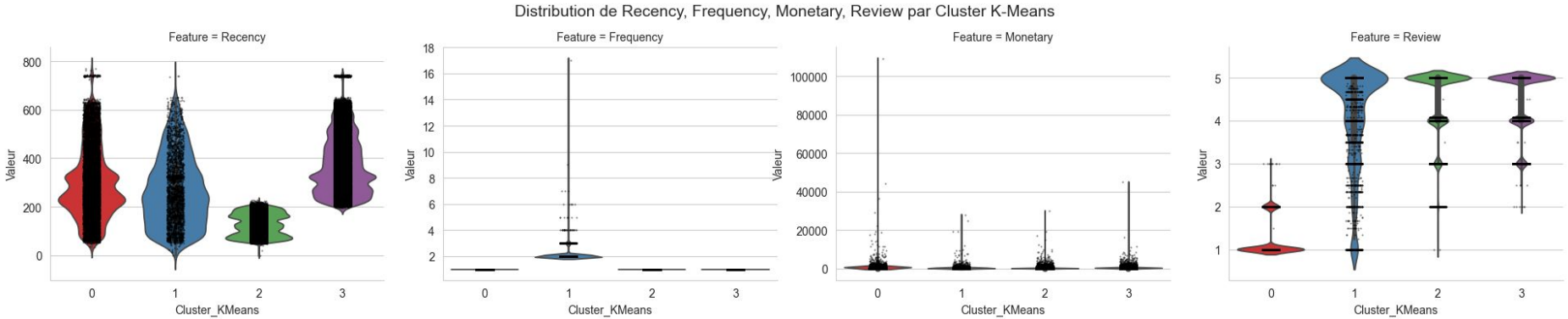
➡ **Observation :** Le meilleur compromis entre séparation et cohésion est atteint avec **k = 4**.

Visualisation 2D avec PC1 et PC2



- **4 clusters distincts** visualisés sur les composantes principales PC1 et PC2.
- **Bonne séparation** des groupes, avec des clusters compacts (0, 2, 3) et un cluster plus dispersé (1).
- Permet d'évaluer la **cohérence des segments** clients et la qualité du clustering.

Visualisation avec volinplots



Cluster	Recency (jours)	Frequency (achats)	Monetary (BRL)	Review (⭐)	Effectif	Profil	Actions
0	299	1	305.8	1.2	13 297	Clients insatisfaits et peu actifs	Campagnes de réengagement & service client
1	268	2.1	487.1	4.1	2 997	Clients fidèles et engagés	Fidélisation via offres premium
2	127	1	203.3	4.6	29 081	Nouveaux clients très satisfaits	Encourager le réachat (cross-selling)
3	377	1	180.2	4.5	50 721	Clients dormants mais satisfaits	Réactivation via offres spéciales

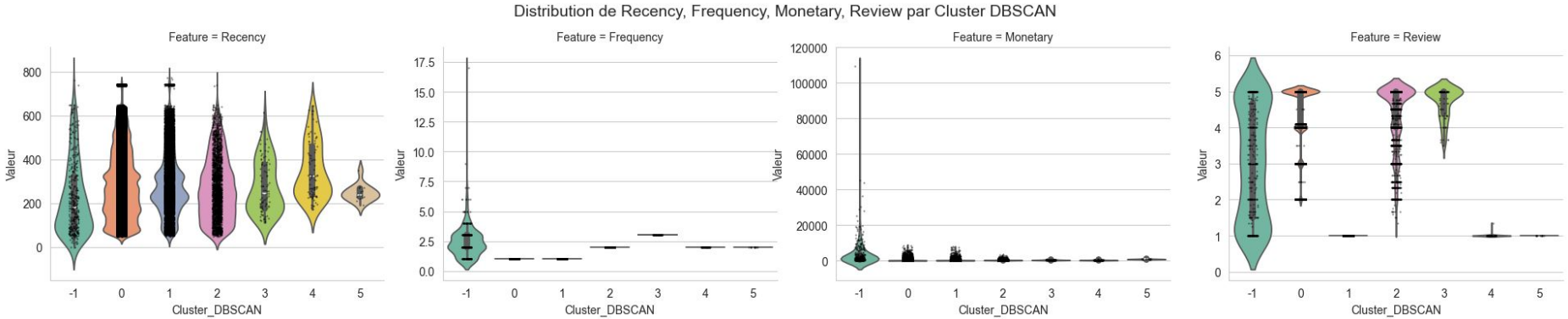
Clustering DBscan

Graphique PCA : Représentation des clusters DBSCAN dans un espace réduit à 2 dimensions.

Clusters détectés :

- **-1 (bruit/anomalies)** : Clients atypiques, très éloignés des autres.
- **0, 1, 2, 3, 4, 5** : Groupes de clients segmentés selon la densité des points.

Analyse des Clusters DBSCAN



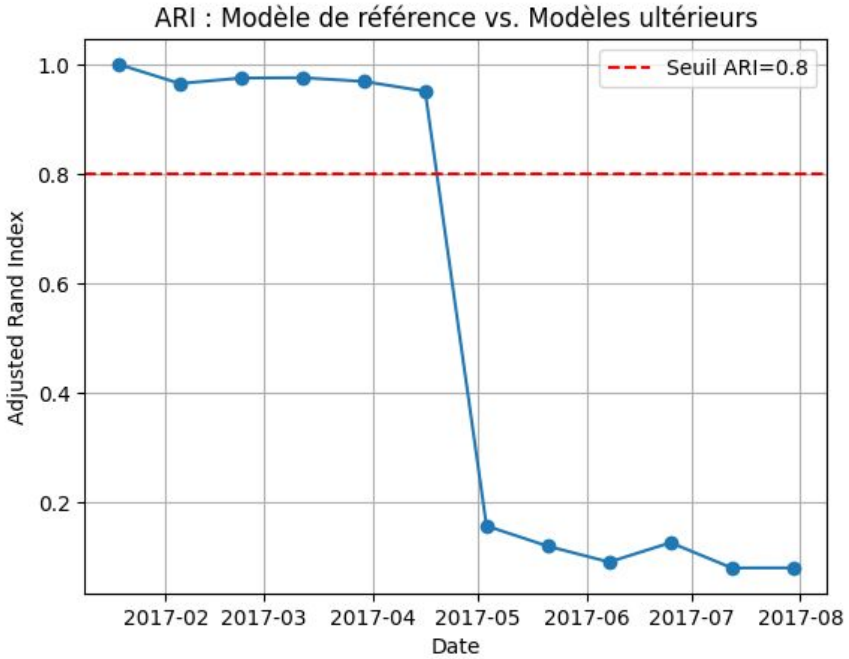
Cluster	Profil	Insights
0	Clients occasionnels satisfaits (82k)	Peu d'engagement mais bonnes notes
1, 4, 5	Clients insatisfaits (10k)	Mauvaise expérience, à analyser
2, 3	Clients fidèles et satisfaits (2.5k)	Potentiel à fidéliser
-1	Anomalies / comportements extrêmes (377)	Outliers

Choix du modèle

➡ **Moins exploitable que K-Means**, DBSCAN crée des clusters plus diffus avec beaucoup de bruit (-1). **K-Means offre une segmentation plus nette et actionnable.**

Maintenance

Slide : Maintenance du Modèle (Sans Review)



📌 **Objectif :** Vérifier la stabilité du clustering dans le temps.

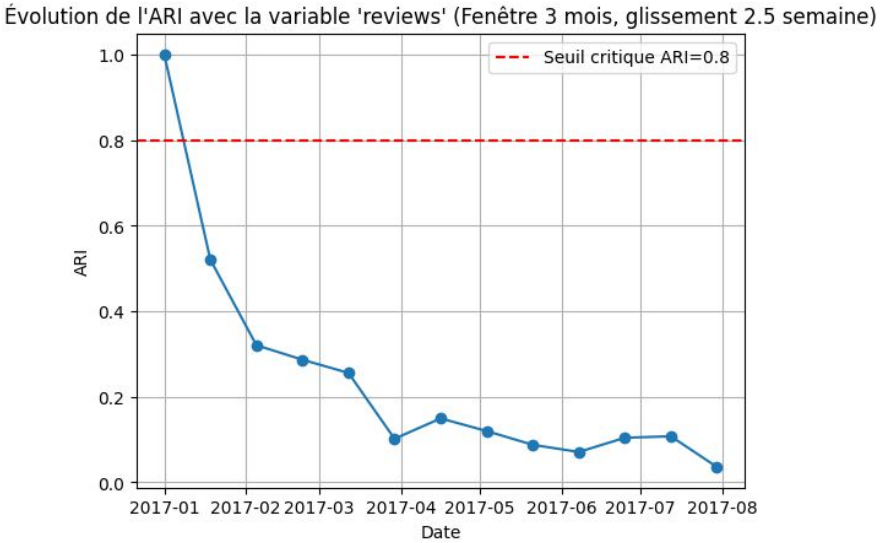
📌 **Méthode :**

- **Fenêtre initiale :** 3 mois de données.
- **Glissement :** 2.5 semaines.
- **Comparaison ARI :** Modèle initial vs. nouveaux modèles.

📌 **Résultats :**

- **ARI < 0.8** après plusieurs mois → recalibrage tous **3-4 mois**.

Maintenance du Modèle (Avec Review) – Non Viable



📌 **Objectif :** Tester l'impact de la variable *reviews* sur la stabilité du clustering.

📌 **Méthode :**

- **Fenêtre initiale :** 3 mois de données.
- **Glissement :** 2.5 semaine.
- **Comparaison ARI :** Modèle initial vs. modèles actualisés.

📌 **Résultats :**

- **ARI chute rapidement** 🔽 → segmentation instable.
- **Impact négatif des reviews**, trop fluctuantes pour être un critère fiable.
- **Conclusion :** Exclure *reviews* pour une segmentation robuste.

Merci