Segmentation clientèle

Zaccaria Amillou

Sommaire

- Contexte
- Données
- Requêtes SQL
- EDA
- Modélisation
- Simulation

Contexte

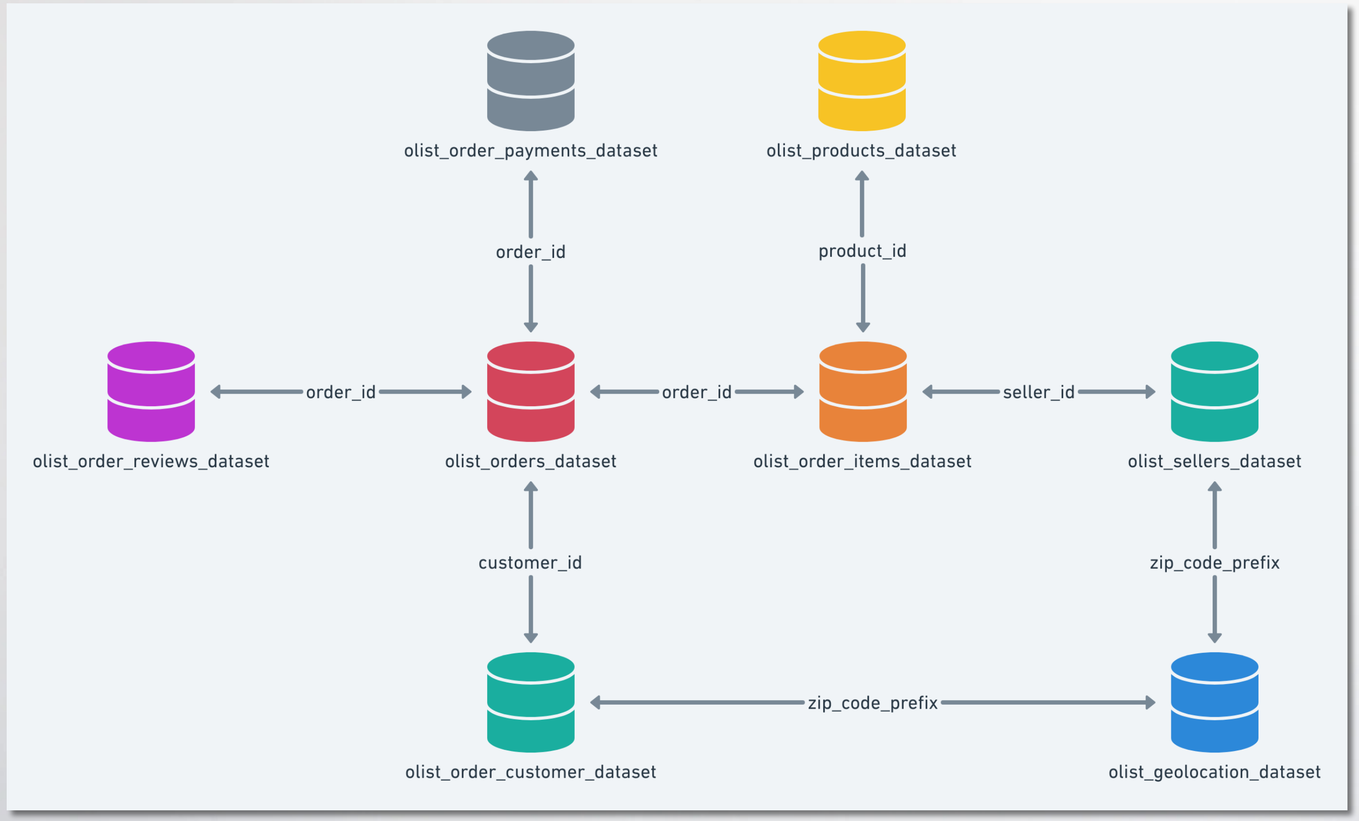
Entreprise brésilienne

Propose solution de vente sur marketplace en ligne

Objectifs:

Définir la segmentation de la clientèle

Fichiers utilisées



Requêtes SQL

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 ?

```sql
SELECT
    o.order_id as 'id ordre',
    (julianday(order_estimated_delivery_date) -
julianday(order_delivered_customer_date)) as 'j_diff'
FROM
    orders o
WHERE
    (julianday(order_estimated_delivery_date) -
julianday(order_delivered_customer_date)) >= 3
AND
    STRFTIME('%m', order_approved_at) >= 3
AND
    order_status NOT LIKE 'canceled'
```

Requêtes SQL

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

```sql
SELECT
    oi.seller_id,
    COUNT(oi.product_id),
    SUM(op.payment_installments * op.payment_value) AS total
FROM order_items oi
LEFT JOIN orders o ON oi.order_id = o.order_id
LEFT JOIN order_pymts op ON o.order_id = op.order_id
GROUP BY oi.seller_id
HAVING SUM(op.payment_installments * op.payment_value) > 100000;
```

Requêtes SQL

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) ?

```sql
SELECT
    seller_id,
    COUNT(oi.product_id) AS 'nombre produits',
    order_purchase_timestamp
FROM
    order_items oi
JOIN orders o ON o.order_id = oi.order_id
WHERE strftime('%Y-%m-%d %H:%M:%S', o.order_purchase_timestamp) <
strftime('%Y-%m-%d %H:%M:%S', '2018-10-17 17:30:18', '-3 months')
GROUP BY
    seller_id
HAVING
    COUNT(oi.product_id) >= 30
ORDER BY "nombre product" DESC;
```

Requêtes SQL

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

```sql
SELECT
    c.customer_zip_code_prefix as ZIP,
    AVG(review_score)
FROM order_reviews or2
JOIN orders o ON or2.order_id = o.order_id
JOIN main.customers c on o.customer_id = c.customer_id
GROUP BY c.customer_zip_code_prefix
ORDER BY AVG(review_score) ASC;
```

EDA

4 81

Add Code Cell | Add Markdown Cell | Add SQL Cell

In 7

```
# affichage dataframe
display(customers.head(3))
```

Executed at 2024.02.22 08:40:06 in 7ms

3 rows | 3 rows × 5 columns | Static Output

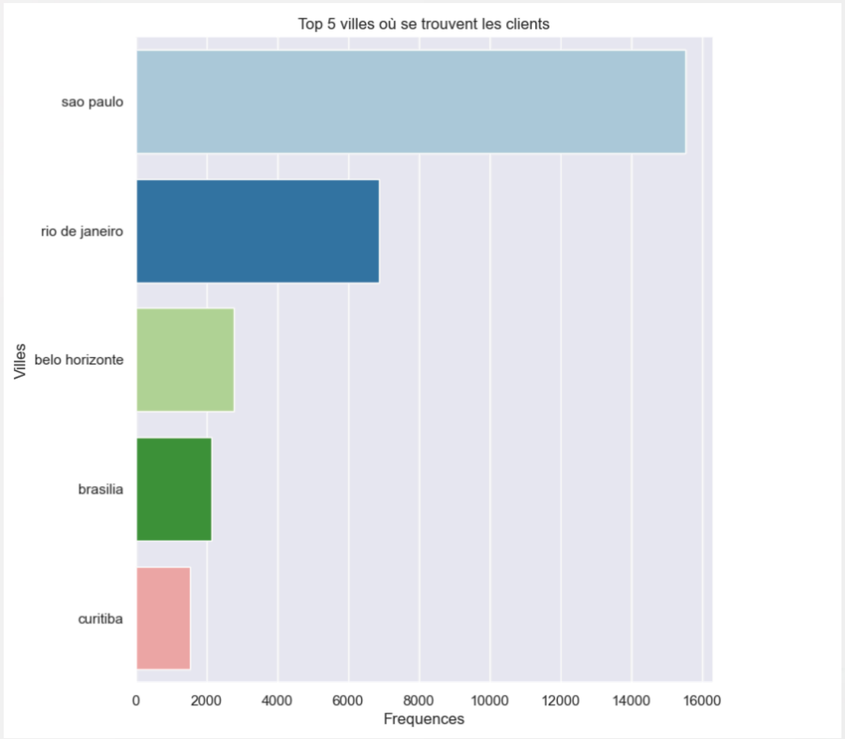
	customer_id	customer_unique_id	customer_zip_code_prefix
0	06b8999e2fba1a1fbc88172c00ba8bc	861eff4711a542e4b93843c6dd7febb	144
1	18955e83d337fd6b2def6b18a428ac7	290c77bc529b7ac935b93aa66c333dc	97
2	4e7b3e00288586ebd08712fdd0374a0	060e732b5b29e8181a18229c7b0b2b5	11

Le df `customers` contient les informations à propos des clients

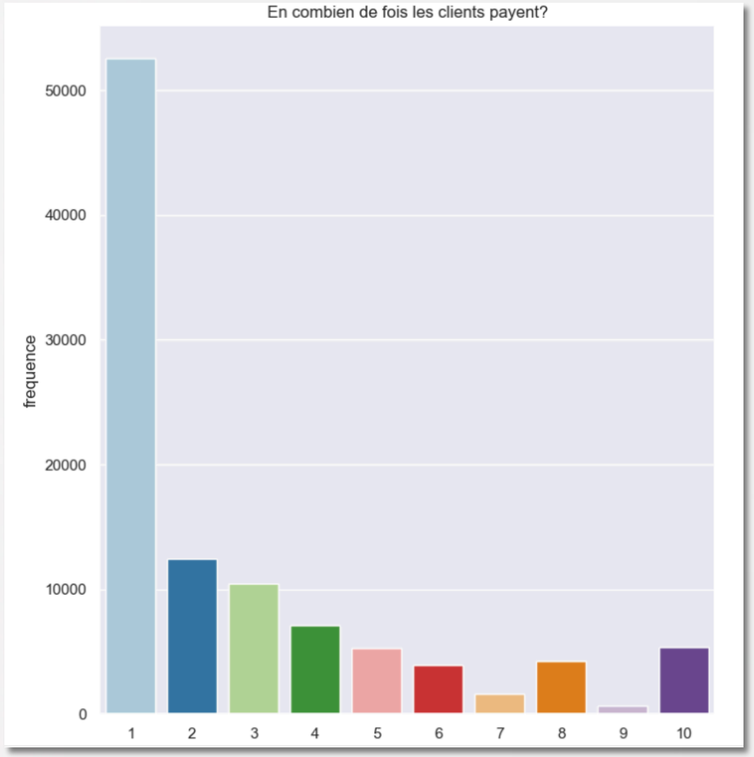
- `customer_id` : id donnée aux clients pour chaque ordre
- `customer_unique_id` : id unique pour identifier chaque client
- `customer_zip_code_prefix` : code postale du client
- `customer_city` : la ville du client au moment de l'ordre
- `customer_state` : l'état du client au moment de l'ordre

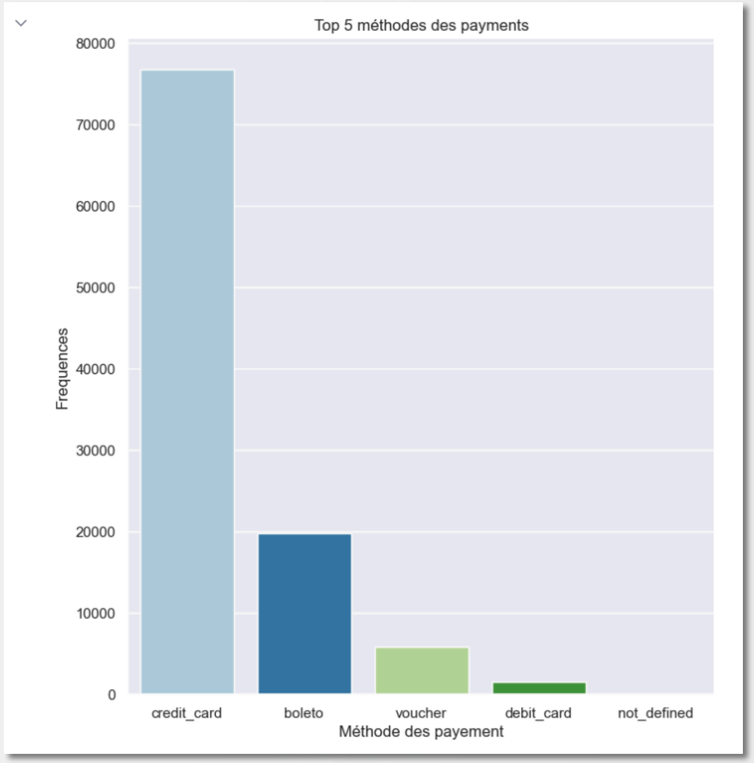
EDA - Analyses

In 36
```python
def graphique_top_count(data, colonne):
    ''' Creation bar plot des top 10 des iteration d'une colonne données
    ARGS:
        data: dataframe
        colonne: colonne a itérer'''
    # count d'itérations des typologies de payments
    data_count = data[colonne].value_counts().reset_index()
    data_count.columns = [colonne, 'count']
    # top 10
    top_10_count = data_count.sort_values(by='count', ascending=False).head(10)
    # creation graphique
    sns.barplot(
        top_10_count,
        x=colonne,
        y='count',
    palette='Paired')
```
Executed at 2024.02.22 09:26:50 in 2ms

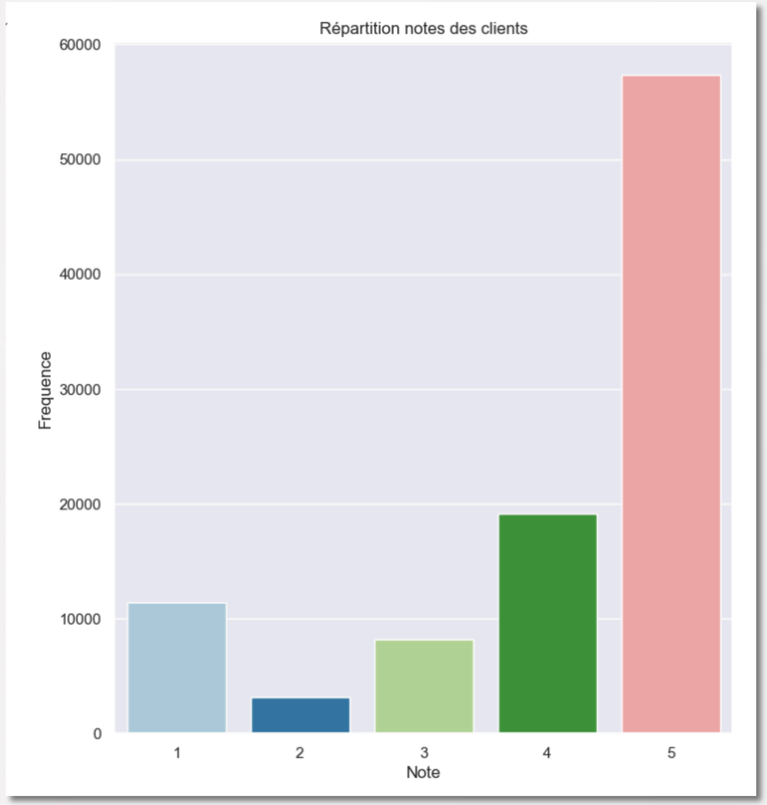


EDA - Analyses





EDA - Analyses



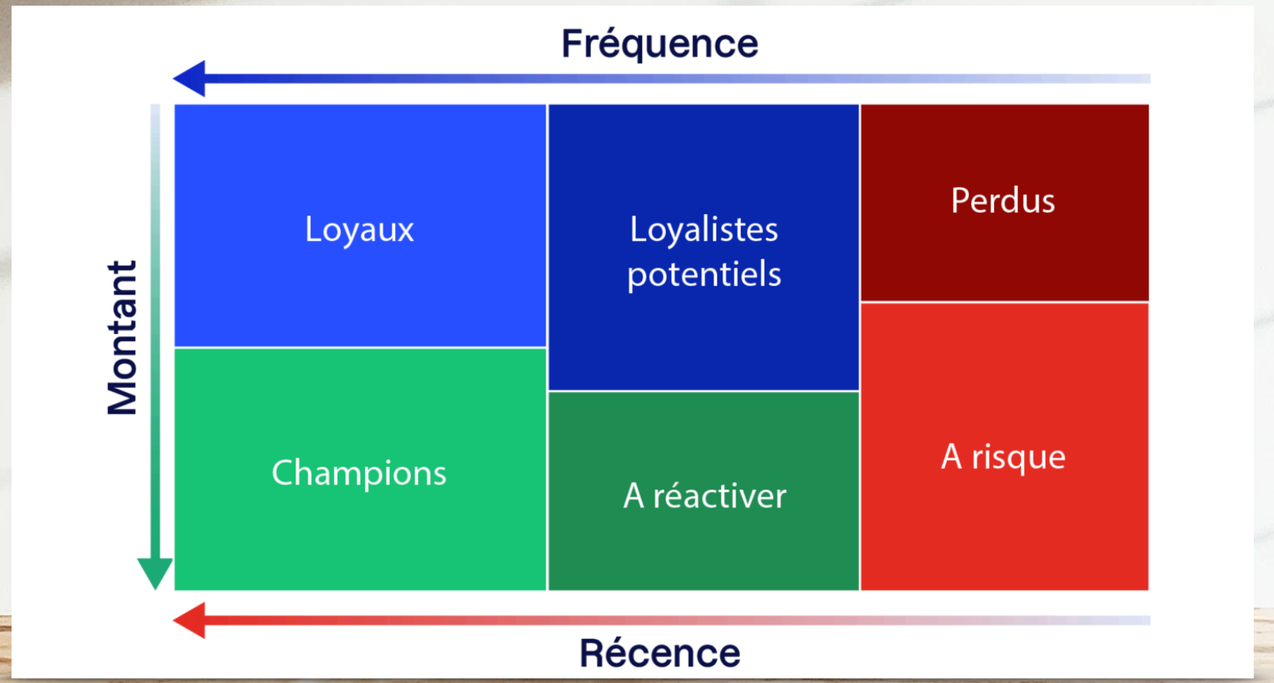
EDA

In 53

```python
# jointures entre variables
df = (
    customers[['customer_id','customer_city', 'customer_unique_id', 'customer_state']]
    .merge(orders, on='customer_id', how='left')
    .merge(payments, on='order_id', how='left' )
    .merge(items,on='order_id',how='left')
    .merge(sellers[['seller_id','seller_city','seller_state']], on='seller_id', how='left')
    .merge(reviews, on='order_id', how='left')
)
```

Executed at 2024.02.22 09:26:53 in 384ms

Modélisation



Modélisation - RFM

In 9
```
# calcul de la date à utiliser
ajourdhui = df['order_purchase_timestamp'].max() + dt.timedelta(days=2)
ajourdhui
```
Executed at 2024.02.22 08:40:58 in 3ms

Out 9
```
Timestamp('2018-10-19 17:30:18')
```

In 12
```
# Calculs de jours passés à partir d'ajourdhui (variable) et derniere commande
recence_df['Recence']= recence_df['order_purchase_timestamp'].apply(lambda x: (ajourdhui - x).days)
display(recence_df.head(3))
```
Executed at 2024.02.22 08:40:58 in 160ms

In 13
```
frequence_df = df.groupby('customer_unique_id').size().reset_index(name='Frequence')
# Sort du df par frequence
frequence_df = frequence_df.sort_values(by='Frequence')
```

In 14
```
# création df pour le montant
montant_df = df.groupby('customer_unique_id', as_index=False)['payment_value'].sum()
montant_df.columns = ['customer_unique_id', 'Montant']
```

Modélisation - RFM

In 15

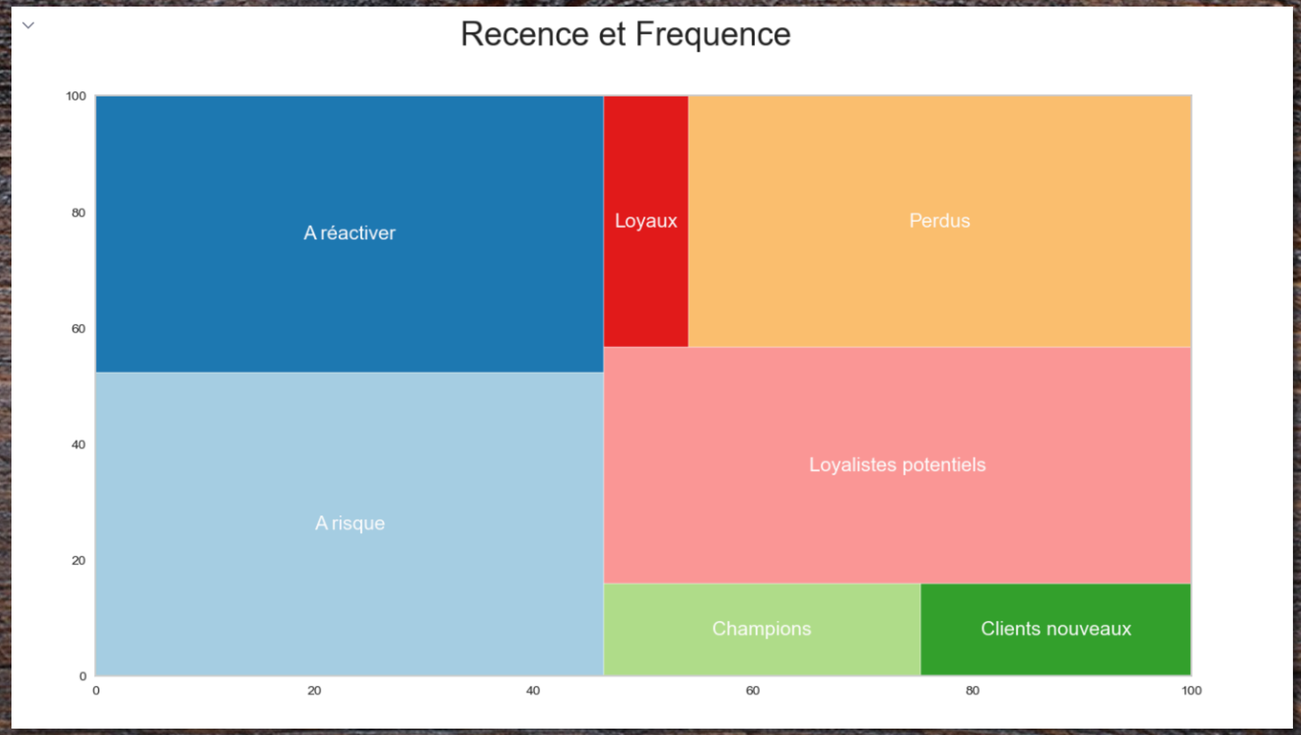
```
# Merge entre le 3 df
RF_df = recence_df.merge(frequence_df, on='customer_unique_id')
RFM_df = RF_df.merge(montant_df, on='customer_unique_id').drop(columns='order_purchase_timestamp')
display(RFM_df.head(3))
```

Executed at 2024.02.22 08:40:58 in 40ms

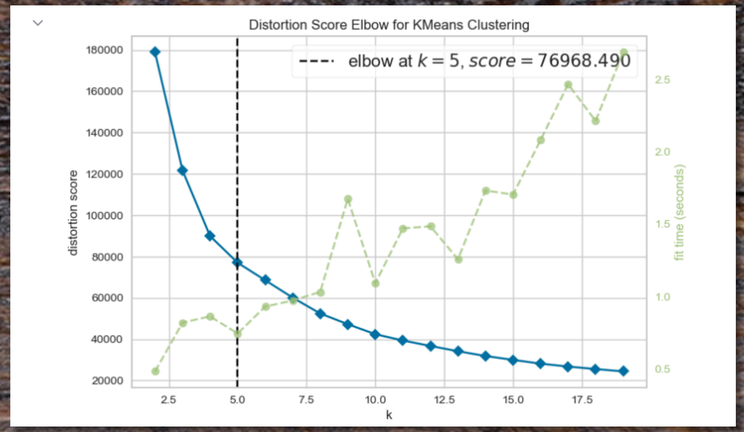
3 rows | 3 rows × 4 columns | Static Output

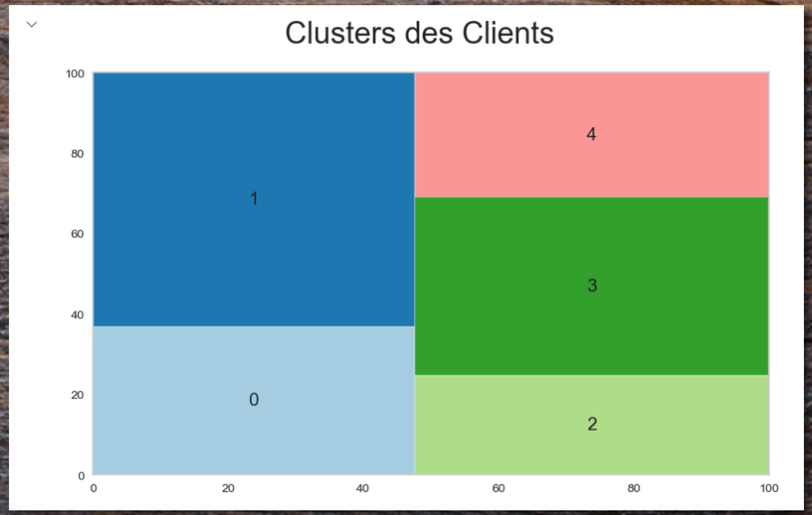
	customer_unique_id	Recence	Frequence	Montant
0	0000366f3b9a7992bf8c76cfdf3221e2	162	1	141.90
1	0000b849f77a49e4a4ce2b2a4ca5be3f	165	1	27.19
2	0000f46a3911fa3c0805444483337064	587	1	86.22

Modélisation - Cluster manuel



Modélisation - K Means

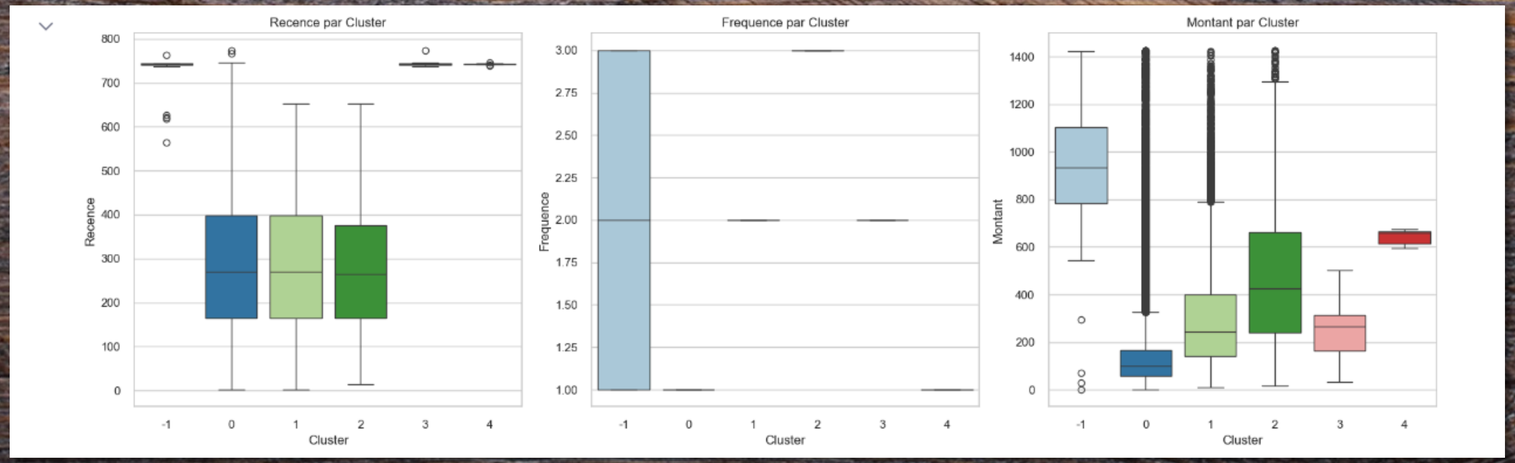




Modélisation - DB Scan

```
In 39  1  # Init DBScan
       2  dbscan = DBSCAN(eps=0.5, min_samples=5)
       3  # Fitting DBSCAN
       4  clusters = dbscan.fit_predict(X_scaled)
       5  # Ajout label cluster dans le df
       6  RFM_df_k_1['Cluster'] = clusters
```

Executed at 2024.02.22 08:42:13 in 45s 782ms



Simulation

In 8

```python
def prep_df(df, ratio):
    '''Fonction qui divise le dataframe selon un ratio sélectionné
    ARGS:
        df: dataframe
        ratio: ratio de division du df'''

    df_sort = df.sort_values('Recence').reset_index()
    num_lignes = round(df_sort.shape[0] * ratio)
    df1 = df_sort.iloc[:num_lignes, :]
    return df1.drop('index', axis=1)

```

Executed at 2024.02.22 08:42:24 in 6ms

In 9

```python
def div_pas (df,t1,  semaine):
    '''Fonction pour diviser le df en semaines
    ARGS:
        df: dataframe
        t1 : date limite
        seamaines : numéro des semaines à considerer'''

    # Calculer le pas en jours en fonction du pas en semaines spécifié
    step = 7 * semaine
    # Trouver la date la plus ancienne et la plus récente dans la colonne 'Recence'
    ancienne, nouv = min(df['Recence']), max(df['Recence'])
    # Calculer la durée totale en jours
    nb = nouv - ancienne
    # Init liste pour stocker les différents pas de temps
    T = []
    t = t1
    # Générer les pas de temps jusqu'à la date la plus récente
    while t < nouv:
        T.append(t)
        t += step
    # Supprimer le premier élément de la liste car il est égal à t1
    T.pop(0)
    return T
```

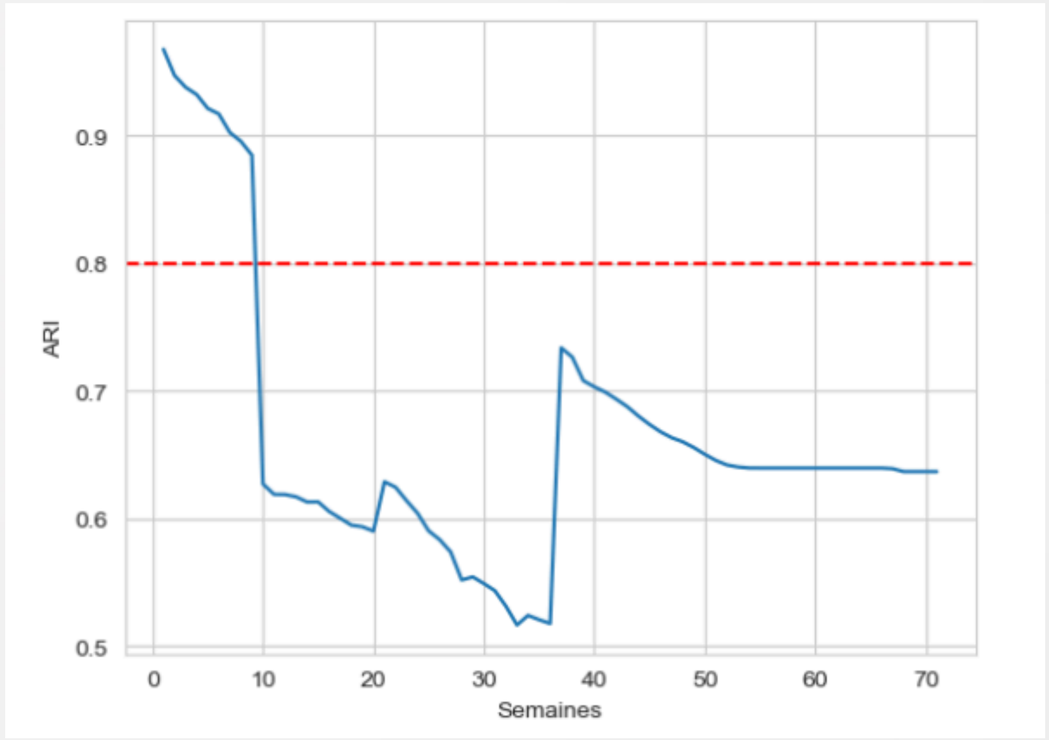
Executed at 2024.02.22 08:42:24 in 4ms

Simulation

ARI - Adjusted Rand Index

Une mesure utilisée en analyse de clustering pour évaluer la similarité entre deux ensembles de regroupements. Il compare les regroupements réels avec les regroupements prédits par un algorithme de clustering, prenant en compte le hasard.

Simulation



Graphique ARI

Merci pour votre attention.