



Segmentation des clients

Sujet : Segmentation des clients

Type : Clustering

Target : Segmentation des clients selon RFM (Recency, Frequency & Monetary value)

Reference : <https://archive.ics.uci.edu/ml/datasets/online+retail>

Dataset : [https://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online Retail.xlsx](https://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online+Retail.xlsx)

the more we understand about our customers the more efficient actions we can consider to apply.



Customer segmentation is the practice of dividing a customer base into groups of individuals that are similar in specific ways relevant to marketing, such as age, gender, interests and spending habits."

RFM analysis readily answers these questions for your business...(Reference from Putler)

- Who are my best customers?
- Which customers are at the verge of churning?
- Who has the potential to be converted in more profitable customers?
- Who are lost customers that you don't need to pay much attention to?
- Which customers you must retain?
- Who are your loyal customers?
- Which group of customers is most likely to respond to your current campaign?

Compréhension de la problématique

La problématique de la segmentation des clients en se basant sur l'analyse RFM (Récence, Fréquence, Montant) est une question centrale dans le domaine du marketing et de la gestion de la relation client. Cette méthodologie de segmentation vise à diviser une base de clients en groupes homogènes en fonction de leur comportement d'achat, ce qui permet aux entreprises de mieux comprendre leurs clients et de personnaliser leurs stratégies de marketing et de fidélisation. Le rôle du Machine Learning (ML) dans l'étude de ce thème est essentiel.

L'analyse RFM repose sur trois critères clés :

1. **Récence (Recency)** : Il s'agit de la mesure de la dernière interaction ou transaction d'un client avec l'entreprise. Les clients récents sont souvent plus actifs et engagés, tandis que les clients inactifs

depuis longtemps peuvent être en voie de désengagement.

2. **Fréquence (Frequency)** : Ce critère évalue la fréquence des interactions ou transactions d'un client sur une période donnée. Les clients fréquents sont souvent plus rentables et loyaux.
3. **Montant (Monetary)** : Il mesure la valeur monétaire totale des transactions effectuées par un client. Les clients à fort montant contribuent davantage aux revenus de l'entreprise.

Approche

Pour traiter la problématique de la segmentation des clients en utilisant l'analyse RFM et le Machine Learning, plusieurs approches et algorithmes peuvent être utilisés. Voici une synthèse des différentes étapes et méthodes :

1. **Calcul des Coefficients RFM** : Après EDA et Preprocessing La première étape consiste à calculer les coefficients RFM pour chaque client. Cela implique de déterminer la récence, la fréquence et le montant des transactions pour chaque client, généralement sur une période donnée.
2. **Segmentation avec K-Means** : Une fois que les coefficients RFM sont calculés, l'algorithme de clustering K-Means peut être utilisé pour regrouper les clients en segments homogènes en fonction de ces coefficients. K-Means attribue automatiquement les clients à des groupes en minimisant la variance intra-segment. Le nombre de clusters (K) doit être défini a priori.
3. **Algorithmes de Classification** : Après avoir effectué la segmentation K-Means, vous pouvez utiliser des algorithmes de classification pour attribuer des labels à ces segments. Il existe plusieurs algorithmes de classification, tels que la régression logistique, les arbres de décision, les forêts aléatoires ou les machines à vecteurs de support (SVM). Ces algorithmes attribuent vont utiliser la classe donnée par K-means comme la target-variable et les coeff RFM comme features variables
4. **Validation et Évaluation** : Il est essentiel d'évaluer la qualité de la segmentation et de la catégorisation des clients. Des métriques telles que la méthode d'Elbow pour K-Means, la précision, le rappel, et la F-mesure pour la classification peuvent être utilisées pour mesurer la performance du modèle.
5. Application Streamlit qui sert à catégoriser un client en tant que Silver, Gold ou Platinum en se basant sur ses coefficients RFM.

Analyse exploratoire des données (EDA) & Preprocessing

1. Importation des bibliothèques nécessaires :
 - `pandas` pour la manipulation et l'analyse des données.
 - `numpy` pour les opérations numériques.
 - `matplotlib.pyplot` pour la création de graphiques et de diagrammes.
 - `seaborn` pour la visualisation des données.
 - `warnings` pour la gestion des messages d'avertissement.
 - `datetime` pour travailler avec les dates et les heures.
2. Ignorer les avertissements :
 - Vous avez utilisé `warnings.filterwarnings('ignore')` pour supprimer les messages d'avertissement qui pourraient être générés pendant l'exécution de votre code.

3. Lire les données à partir d'un fichier Excel :

- Vous utilisez la fonction `pd.read_excel("Online Retail.xlsx")` pour lire les données à partir du fichier Excel "Online Retail.xlsx" et les stocker dans le DataFrame `df`.

4. Afficher les 10 premières lignes de l'ensemble de données :

- Vous utilisez `df.head(10)` pour afficher les 10 premières lignes du DataFrame afin d'avoir un aperçu rapide des données.

5. Vérifier la forme de l'ensemble de données :

- Vous utilisez `df.shape` pour obtenir les dimensions du DataFrame, ce qui vous donne le nombre de lignes et de colonnes.

6. Obtenir des informations générales sur l'ensemble de données :

- Vous utilisez `df.info()` pour obtenir un aperçu de l'ensemble de données, y compris les types de données de chaque colonne, le nombre de valeurs non nulles et l'utilisation de la mémoire.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import datetime as dt

warnings.filterwarnings('ignore')
color = sns.color_palette()

df=pd.read_excel("Online Retail.xlsx")

df.head(10)

# Let's discover the shape of our dataset
df.shape

# getting general infos about our dataset
df.info()
```

1. Trouver la dernière date disponible dans votre ensemble de données :

```
df['InvoiceDate'].max()
```

- Cette ligne renvoie la date la plus récente présente dans la colonne "InvoiceDate" de votre DataFrame.

2. Trouver la première date disponible dans votre ensemble de données :

```
df['InvoiceDate'].min()
```

- Cette ligne renvoie la date la plus ancienne présente dans la colonne "InvoiceDate" de votre DataFrame.

3. Compter le nombre de valeurs nulles dans chaque colonne :

```
df.isnull().sum().sort_values(ascending=False)
```

- Cette ligne affiche le nombre de valeurs nulles dans chaque colonne, triées par ordre décroissant.

4. Créer un nouveau DataFrame sans les valeurs nulles :

```
df_new = df.dropna()
```

- Cette ligne crée un nouveau DataFrame appelé `df_new` en supprimant toutes les lignes contenant des valeurs nulles.

5. Vérifier que le nouveau DataFrame n'a plus de valeurs manquantes :

```
df_new.isnull().sum().sort_values(ascending=False)
```

- Cette ligne vérifie que le DataFrame `df_new` ne contient plus de valeurs nulles après la suppression.

6. Changer le type de données de la colonne "CustomerID" de float64 à int64 :

```
df_new['CustomerID'] = df_new['CustomerID'].astype('int64')
```

- Cette ligne convertit le type de données de la colonne "CustomerID" de float64 à int64, en supprimant les décimales.

7. Décrire les colonnes numériques du DataFrame :

```
df_new.describe().round(3)
```

- Cette ligne génère des statistiques descriptives pour les colonnes numériques du DataFrame `df_new`, arrondies à trois décimales.

8. Éliminer les lignes avec des quantités négatives :

```
df_new = df_new[df_new.Quantity > 0]
```

- Cette ligne filtre le DataFrame pour ne conserver que les lignes où la colonne "Quantity" a une valeur strictement positive, éliminant ainsi les lignes avec des quantités négatives.

9. Diviser la colonne "Date" en mois, jour et heure pour une analyse ultérieure :

```
df_new.insert(loc=2, column='year_month', value=df_new['InvoiceDate'].map(lambda x: 100*x.year + x.month))
df_new.insert(loc=3, column='month', value=df_new.InvoiceDate.dt.month)
df_new.insert(loc=4, column='day', value=(df_new.InvoiceDate.dt.dayofweek) + 1)
df_new.insert(loc=5, column='hour', value=df_new.InvoiceDate.dt.hour)
```

- Ces lignes ajoutent de nouvelles colonnes au DataFrame `df_new`, extrayant l'année et le mois de la colonne "InvoiceDate" (dans la colonne "year_month"), le mois (dans la colonne "month"), le jour de la semaine (dans la colonne "day"), et l'heure de la journée (dans la colonne "hour") pour une analyse plus détaillée.

10. Créer une nouvelle colonne "amount_spent" :

```
df_new['amount_spent'] = df_new['Quantity'] * df_new['UnitPrice']
```

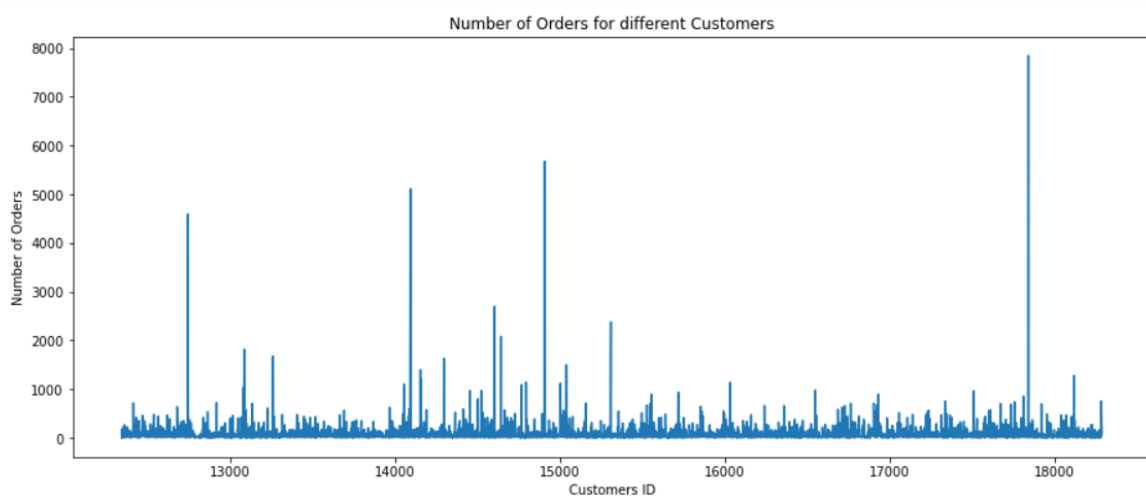
- Cette ligne calcule le montant dépensé pour chaque transaction en multipliant la quantité (column "Quantity") par le prix unitaire (column "UnitPrice") et stocke le résultat dans une nouvelle colonne "amount_spent".

number of orders made by the customers

Out[189]:

	CustomerID	Country	InvoiceNo
0	12346	United Kingdom	1
1	12347	Iceland	182
2	12348	Finland	31
3	12349	Italy	73
4	12350	Norway	17

Number of Orders for different Customers

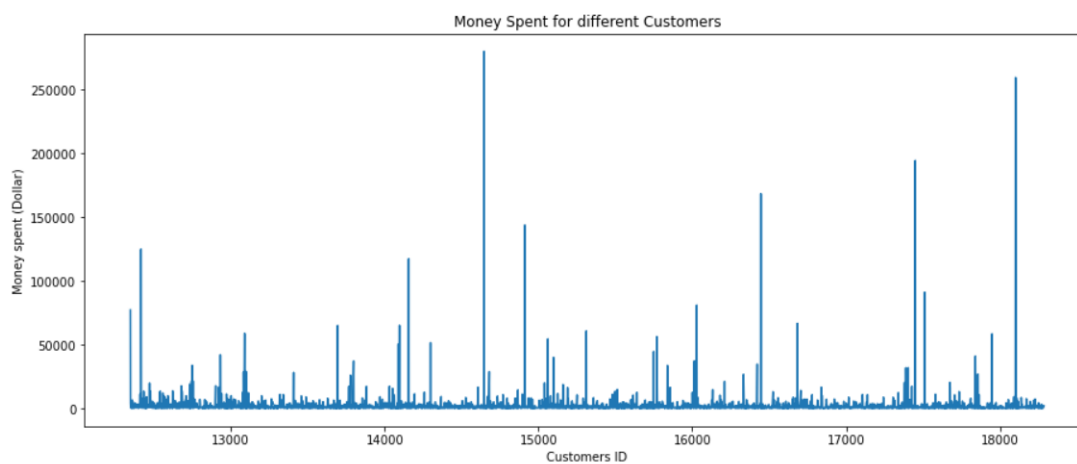


top 10 most number of orders

Out[191]:

	CustomerID	Country	InvoiceNo
4019	17841	United Kingdom	7847
1888	14911	EIRE	5677
1298	14096	United Kingdom	5111
334	12748	United Kingdom	4596
1670	14606	United Kingdom	2700
2185	15311	United Kingdom	2379
1698	14646	Netherlands	2080
570	13089	United Kingdom	1818
699	13263	United Kingdom	1677
1443	14298	United Kingdom	1637

money spent by the customers

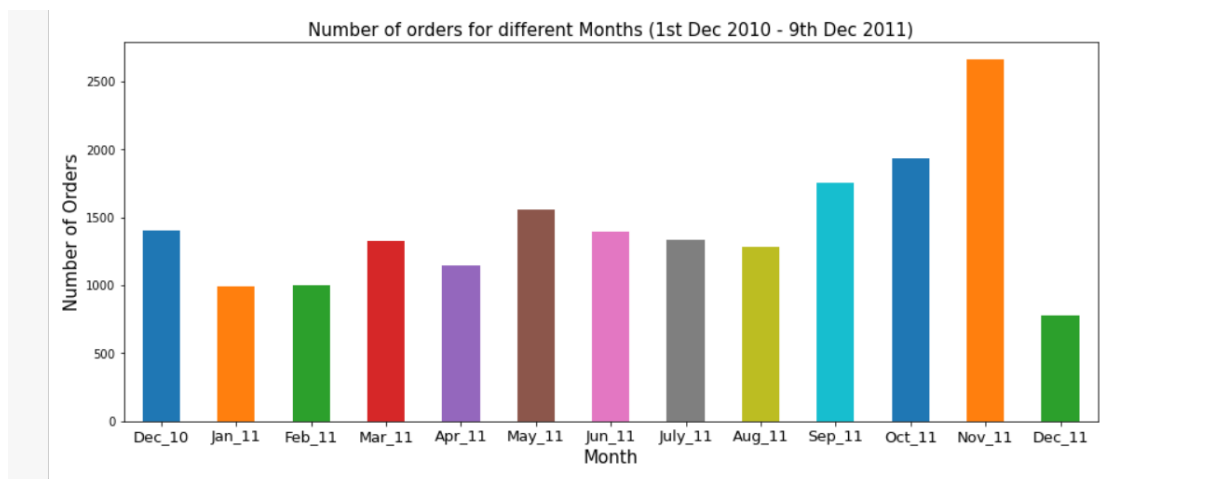


top 10 highest money spent

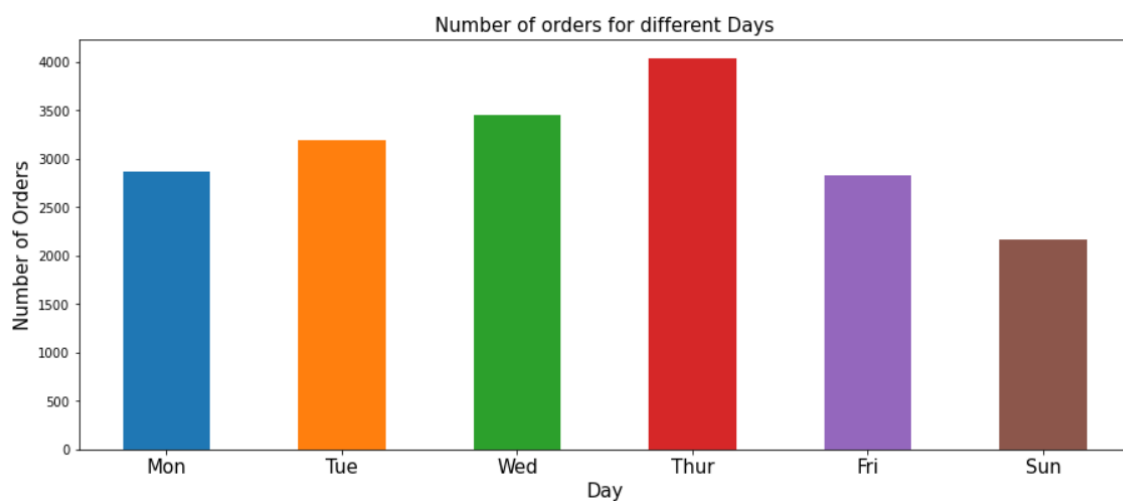
]:

	CustomerID	Country	amount_spent
1698	14646	Netherlands	280206.02
4210	18102	United Kingdom	259657.30
3737	17450	United Kingdom	194550.79
3017	16446	United Kingdom	168472.50
1888	14911	EIRE	143825.06
57	12415	Australia	124914.53
1342	14156	EIRE	117379.63
3780	17511	United Kingdom	91062.38
2711	16029	United Kingdom	81024.84
0	12346	United Kingdom	77183.60

number of orders for different months



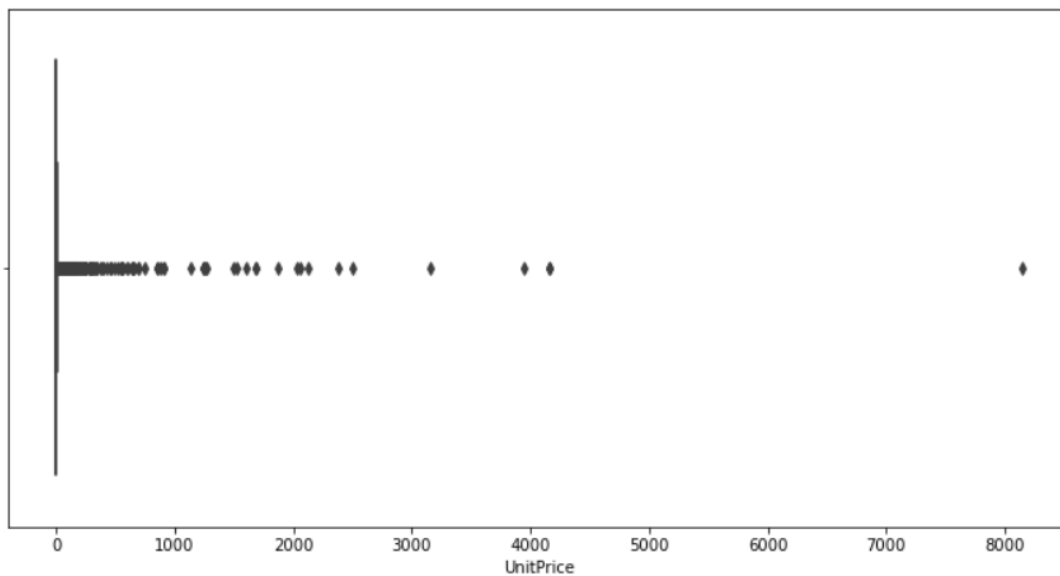
number of orders for different days



Description of UnitPrice Column

```
Out[196]: count    397924.000000
          mean       3.116174
          std       22.096788
          min        0.000000
          25%        1.250000
          50%        1.950000
          75%        3.750000
          max       8142.750000
          Name: UnitPrice, dtype: float64
```

UnitPrice Distribution:



Construction des modèles ML : Clustering

RFM (Recency Frequency Monetary) Analysis

Category 0 : Silver Customer

Category 1 : Platinum Customer

Category 2 : Gold Customer

1. Créer une date de référence :

```
now = dt.date(2011, 12, 9).
```

2. Extraire la date à partir de la colonne "InvoiceDate" :

```
df_new['date'] = pd.DatetimeIndex(df_new.InvoiceDate).date
```

3. Grouper les clients par leur dernière date d'achat :


```
recency_df = df_new.groupby(['CustomerID'], as_index=False)['date'].max()
```

4. Renommer les colonnes du DataFrame `recency_df` :

```
recency_df.columns = ['CustomerID', 'LastPurchaseDate']
```

5. Afficher les premières lignes du DataFrame `recency_df` :

```
recency_df.head()
```

Out[200]:

	CustomerID	LastPurchaseDate
0	12346	2011-01-18
1	12347	2011-12-07
2	12348	2011-09-25
3	12349	2011-11-21
4	12350	2011-02-02

Calculer la récence pour chaque client :

[201]:

	CustomerID	LastPurchaseDate	Recency
0	12346	2011-01-18	325
1	12347	2011-12-07	2
2	12348	2011-09-25	75
3	12349	2011-11-21	18
4	12350	2011-02-02	310

on calcule aussi la frequency et la Monetary

Out[203]:

	CustomerID	Frequency
0	12346	1
1	12347	7
2	12348	4
3	12349	1
4	12350	1

out[204]:

	CustomerID	Monetary
0	12346	77183.60
1	12347	4310.00
2	12348	1797.24
3	12349	1757.55
4	12350	334.40

on regroupe finalement toutes les resultats dans un seul tableau

[205]:

	Recency	Frequency	Monetary
CustomerID			
12346	325	1	77183.60
12347	2	7	4310.00
12348	75	4	1797.24
12349	18	1	1757.55
12350	310	1	334.40

1. Création d'une copie du DataFrame `rfm` pour la segmentation :

```
rfm_segmentation = rfm.copy()
```

- on créer une copie du DataFrame `rfm` que vous nommez `rfm_segmentation` . Cette copie sera utilisée pour la segmentation.

2. Importation de la classe KMeans depuis scikit-learn :

```
from sklearn.cluster import KMeann
```

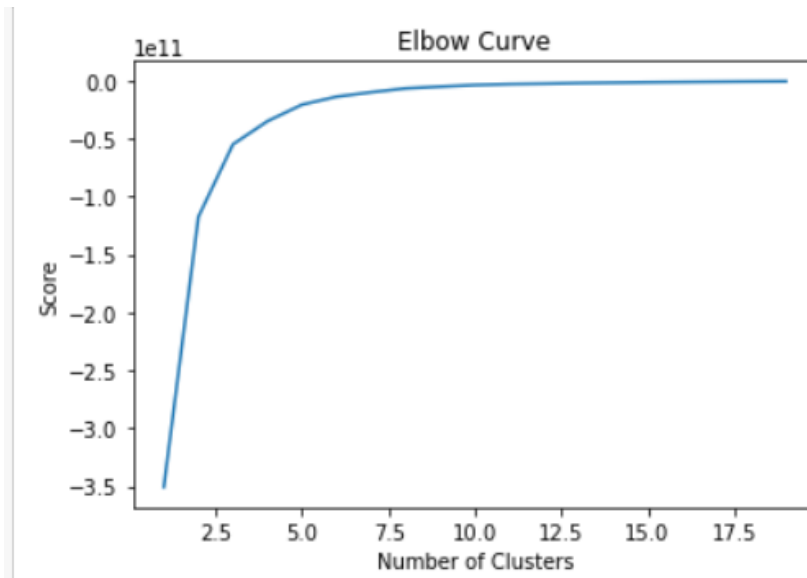
- on importe la classe KMeans à partir de la bibliothèque scikit-learn, qui sera utilisée pour effectuer le clustering K-Means.

3. Détermination du nombre optimal de clusters en utilisant la méthode du coude :

- on exécute une boucle pour différents nombres de clusters (de 1 à 19) en utilisant K-Means, et on calcule le score (inertie) pour chaque nombre de clusters.
- Le score est la somme des carrés des distances entre les points et les centres des clusters. Plus le score est faible, meilleure est la performance du clustering.
- on trace ensuite la courbe du coude en fonction du nombre de clusters et examinez le graphique pour déterminer le nombre optimal de clusters.

4. Tracé de la courbe du coude :

```
plt.plot(Nc, score)
plt.xlabel('Nombre de clusters')
plt.ylabel('Score')
plt.title('Courbe du coude')
plt.show()
```



1. Appliquer K-Means sur les données de segmentation `rfm_segmentation` :

```
kmeans = KMeans(n_clusters=3, random_state=0).fit(rfm_segmentation)
```

- on utilise la classe KMeans de scikit-learn pour effectuer une analyse de clustering K-Means avec 3 clusters (vous pouvez ajuster ce nombre en fonction de vos besoins). L'argument `random_state` est utilisé pour initialiser le générateur de nombres aléatoires, garantissant ainsi que les résultats sont reproductibles.

2. Attribuer les labels de cluster à chaque observation :

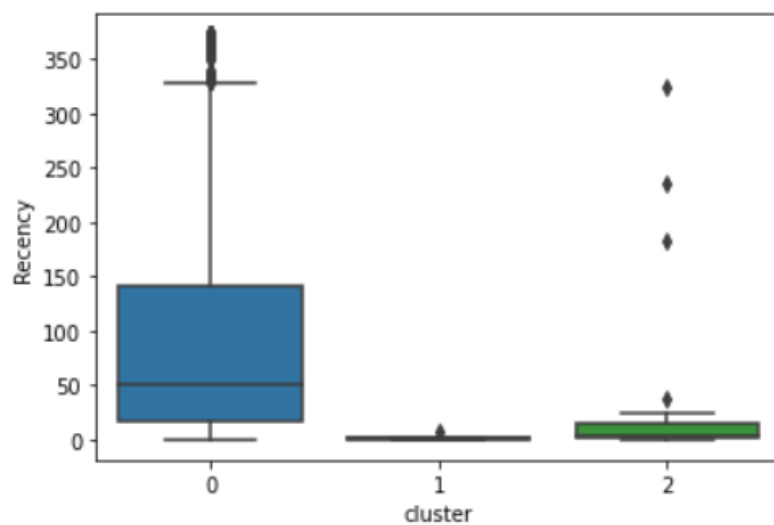
```
rfm_segmentation['cluster'] = kmeans.labels_
```

- on attribue les labels de cluster générés par l'algorithme K-Means à chaque observation dans le DataFrame `rfm_segmentation`. Chaque ligne du DataFrame est désormais associée à un cluster.

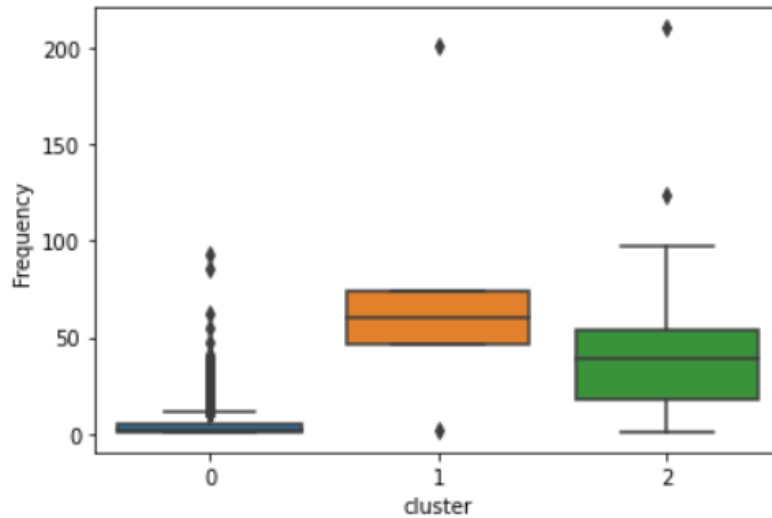
ut[290]:

CustomerID	Recency	Frequency	Monetary	cluster
12346	325	1	77183.60	2
12347	2	7	4310.00	0
12348	75	4	1797.24	0
12349	18	1	1757.55	0
12350	310	1	334.40	0
12352	36	8	2506.04	0
12353	204	1	89.00	0
12354	232	1	1079.40	0
12355	214	1	459.40	0
12356	22	3	2811.43	0

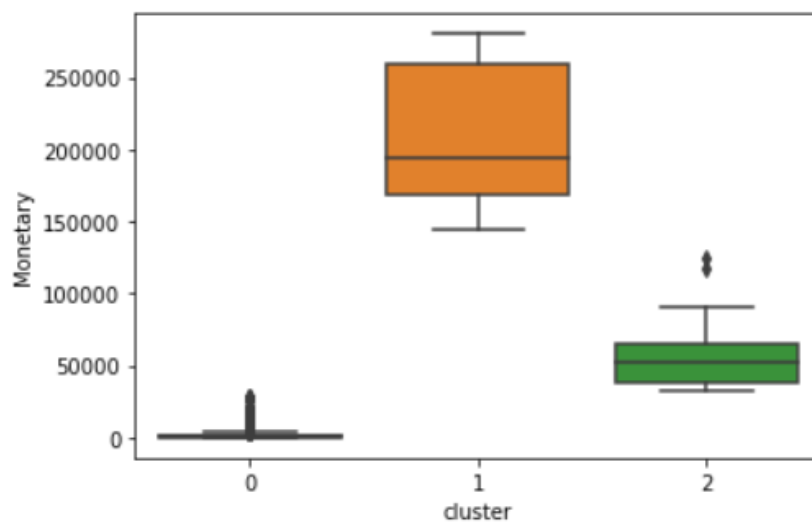
Let's do some visualisation on this data



cluster 0 have high recency rate which is bad. cluster 1 and cluster 2 having low so they are in race of platinum and gold customer.



cluster 0 have low frequency rate which is bad. cluster 1 and cluster 2 having high so they are in race of platinum and gold customer.



cluster 0 have low Monetary rate which is bad. cluster 1 have highest Montary (money spend) platinum where as cluster 2 have medium level(Gold) and cluster 0 is silver customer.

Model & Segmentation

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import f1_score, confusion_matrix, classification_report
from sklearn.model_selection import learning_curve
```

Importation de bibliothèques et modules nécessaires :

- Vous avez importé plusieurs bibliothèques et modules, y compris des classifieurs (Logistic Regression, Decision Tree, Random Forest, AdaBoost, SVM, K-Nearest Neighbors), des métriques d'évaluation (f1_score, confusion_matrix, classification_report, learning_curve), et des fonctions pour diviser les données en ensembles d'entraînement et de test (train_test_split).

```
target = rfm_segmentation["cluster"]
rfm_segmentation.drop("cluster", axis=1, inplace=True)

trainset, testset, ytrain, ytest = train_test_split(rfm_segmentation, target, test_size=0.2, random_state=0)
```

Préparation des données d'entraînement et de test :

- Vous avez séparé la cible (variable à prédire, "cluster") du reste des données dans le DataFrame `rfm_segmentation`, puis divisé l'ensemble de données en ensembles d'entraînement (`trainset`, `ytrain`) et de test (`testset`, `ytest`) en utilisant `train_test_split`.

```
from sklearn.linear_model import LogisticRegression

LR_classification = LogisticRegression()
LR_classification.fit(trainset, ytrain)
Predictions = LR_classification.predict(trainset)
test_pred = LR_classification.predict(testset)

Predictions = Predictions.reshape(-1, 1)
test_pred = test_pred.reshape(-1, 1)

print("the accuracy of the model in train set: {:.2f}%".format(LR_classification.score(trainset, ytrain)*100))

print("the accuracy of the model in test set: {:.2f}%".format(LR_classification.score(testset, ytest)*100))

print(confusion_matrix(ytest, test_pred))
print(classification_report(ytest, test_pred))
```

1. Création d'un modèle de régression logistique et entraînement :

- Vous avez créé un modèle de régression logistique (`LR_classification`) et entraîné ce modèle sur les données d'entraînement (`trainset` et `ytrain`) en utilisant `fit`.

2. Prédictions et évaluation du modèle :

- Vous avez obtenu des prédictions pour l'ensemble d'entraînement et l'ensemble de test en utilisant `predict`. Ensuite, vous avez calculé et affiché la précision du modèle sur ces ensembles, ainsi que la matrice de confusion et le rapport de classification pour l'ensemble de test.

```
In [301]: print("the accuracy of the model in train set: {:.2f}%".format(LR_classification.score(trainset, ytrain)*100))
the accuracy of the model in train set: 99.71%

In [302]: print("the accuracy of the model in test set: {:.2f}%".format(LR_classification.score(testset, ytest)*100))
the accuracy of the model in test set: 99.77%
```

```

[[860  0  0]
 [  0  0  0]
 [  1  1  6]]

```

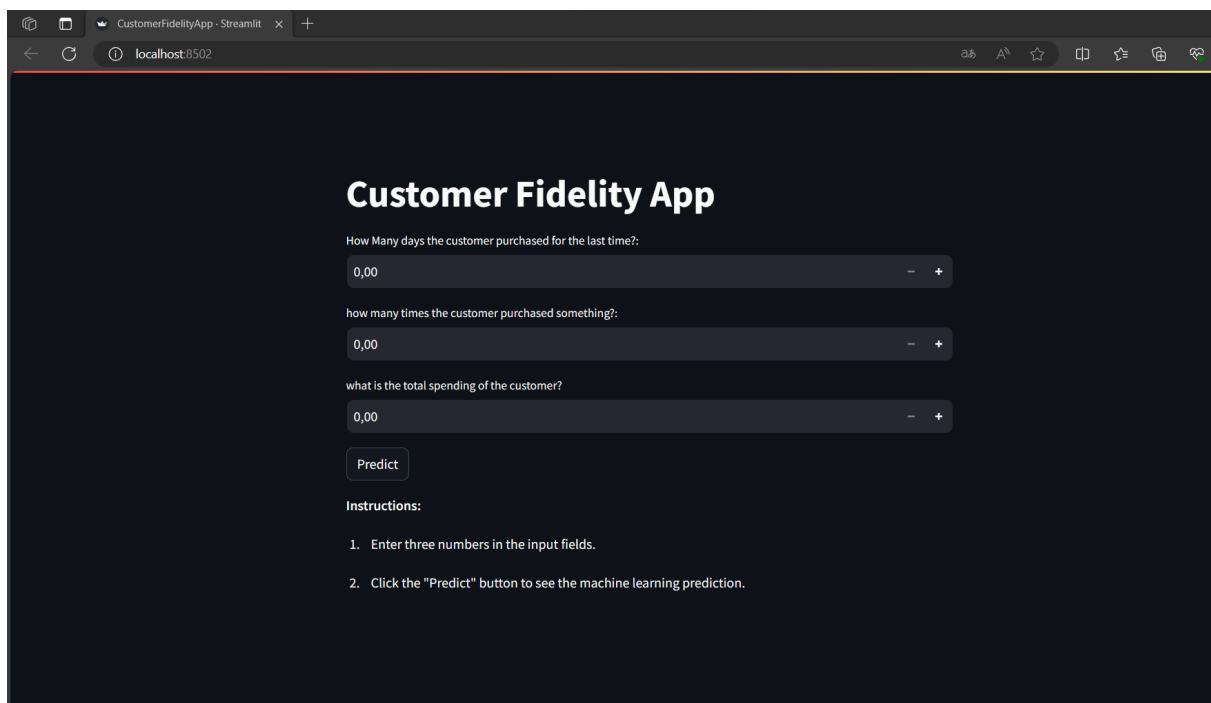
	precision	recall	f1-score	support
0	1.00	1.00	1.00	860
1	0.00	0.00	0.00	0
2	1.00	0.75	0.86	8
accuracy			1.00	868
macro avg	0.67	0.58	0.62	868
weighted avg	1.00	1.00	1.00	868

the same process can be applied to DecisionTreeClassifier, RandomForestClassifier, AdaBoostClassifier, SVC and KNeighborsClassifier


the problem of these algorithms resides in the imbalanced categories of the clustering (most of categories are 0)

the algorithms tend to classify most of the customers as class 0

Déploiement du modèle



Deploy



Customer Fidelity App

How Many days the customer purchased for the last time?:

20,00 - +

how many times the customer purchased something?:

100,00 - +

what is the total spending of the customer?

20000,00 - +

Predict

the client is a: Silver Customer

Instructions:

1. Enter three numbers in the input fields.
2. Click the "Predict" button to see the machine learning prediction.

Deploy

Customer Fidelity App

How Many days the customer purchased for the last time?:

1,00 - +

how many times the customer purchased something?:

100,00 - +

what is the total spending of the customer?

100000,00 - +

Predict

the client is a: Platinum Customer

Instructions:

1. Enter three numbers in the input fields.
2. Click the "Predict" button to see the machine learning prediction.

← localhost:8502

Customer Fidelity App

How Many days the customer purchased for the last time?:

50,00 - +

how many times the customer purchased something?:

15,00 - +

what is the total spending of the customer?

100000,00 - +

Predict

the client is a: Gold Customer

Instructions:

1. Enter three numbers in the input fields.
2. Click the "Predict" button to see the machine learning prediction.