

RAPPORT DE SYNTHÈSE : ANALYSE EXPLORATOIRE ET CLUSTERING DES ARTICLES DANS LES TICKETS DE CAISSE

CAROLE YAPI
KILLBILLS

Introduction

Ce rapport de synthèse vise à décrire les grandes étapes de ma démarche allant de l'analyse exploratoire du chargement de données aux tests de ma fonction de catégorisation. J'expliquerai comment j'ai abordé le problème, les choix de traitement effectués et les résultats obtenus. Ce rapport sera organisé en plusieurs sections, chacune couvrant un aspect spécifique de la démarche et des traitements.

Description de la démarche

1. Analyse des exigences :

L'objectif de cet exercice est de proposer une classification ou un clustering des articles trouvés dans les tickets de caisse reçus par KillBills. Pour ma part, j'ai opté pour un modèle de clustering. Pour cela, j'ai commencé par me connecter à la base de données PostgreSQL en utilisant les informations de connexion fournies (host, userName, password, DB name, port).

```
for column in dataset.columns:
    unique_values = dataset[column].unique()
    print(f"Valeurs uniques pour la colonne {column}:")
    print(unique_values)
    print()
```

2. Exploration des items

J'ai exploré la table "items" pour comprendre la structure et les données disponibles. J'ai utilisé Python et le module psycopg2 pour établir la connexion et Pandas pour manipuler les données.

Au niveau de l'observation des items, j'ai fait une visualisation du dataset plus précisément :

Visualisation des valeurs uniques pour les colonnes

. De cette opération, il est ressorti les observations suivantes :

- Certaines colonnes ne contiennent que des valeurs manquantes (NaN, NAT) : "parent quantity" et "date".
- La colonne "amount" contient des valeurs négatives.
- Les colonnes "description", "type" et "itemName" contiennent également des valeurs manquantes.

- Les colonnes "createAt" et "updateAt" contiennent potentiellement les mêmes valeurs.

2.1 vérifications sur createAt et updateAt

```
createdAt_values = dataset["createdAt"]
updatedAt_values = dataset["updatedAt"]

are_equal = createdAt_values.equals(updatedAt_values)

if are_equal:
    print("Les colonnes 'createdAt' et 'updatedAt' ont les mêmes valeurs.")
else:
    print("Les colonnes 'createdAt' et 'updatedAt' ont des valeurs différentes.")
```

Les colonnes 'createdAt' et 'updatedAt' ont les mêmes valeurs.

Cette observation a confirmé que ces deux variables ont les mêmes valeurs.

2.2 types et valeurs manquantes

```
[83] display(dataset.dtypes)
```

id	object
amount	float64
description	object
date	datetime64[ns]
itemName	object
parent	float64
quantity	float64
taxAmount	float64
taxDescription	object
type	object
storeId	object
createdAt	object
updatedAt	object
taxRate	int64
dtype:	object

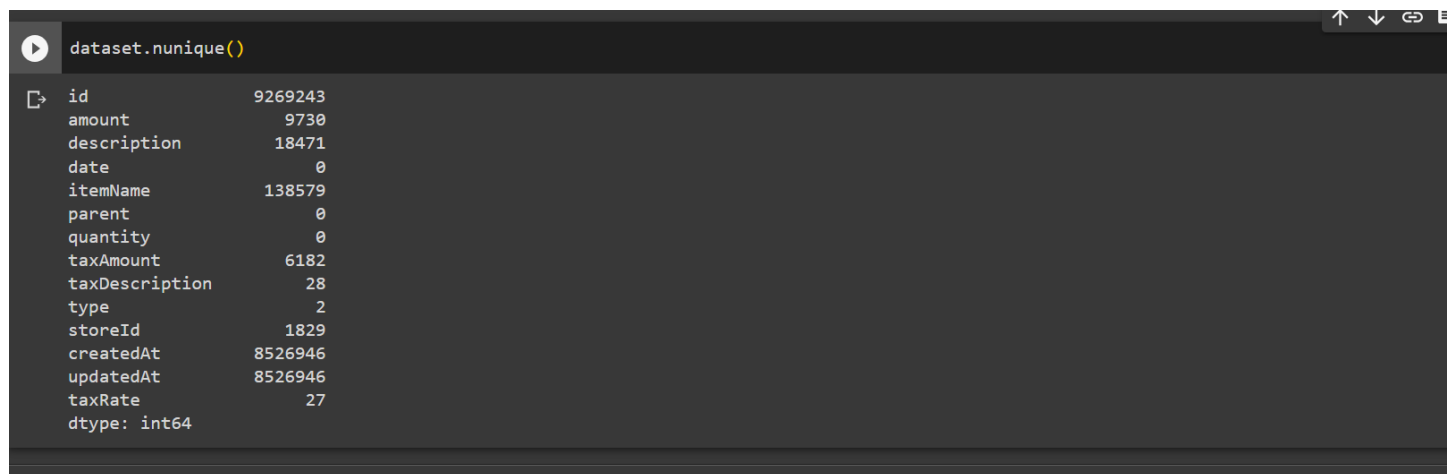
```
(dataset.isnull().sum() / dataset.shape[0]) * 100
```

id	0.000000
amount	0.000000
description	91.693108
date	100.000000
itemName	0.000766
parent	100.000000
quantity	100.000000
taxAmount	0.000000
taxDescription	0.000000
type	94.388301
storeId	0.000000
createdAt	0.000000
updatedAt	0.000000
taxRate	0.000000
dtype:	float64

Il est ressorti des résultats que les types des colonnes "createAt" et "updateAt" nécessitent une correction, et que :

- 91% des descriptions sont manquantes.
- 94% des types sont manquants.
- Les observations précédentes concernant les valeurs manquantes des variables "date", "parent" et "quantity" ont été confirmées (100% de valeurs manquantes).

Étant donné les taux élevés de valeurs manquantes pour certaines variables, j'ai procédé à une étude des valeurs uniques des colonnes afin de déterminer quelles méthodes utiliser pour traiter ces valeurs manquantes.



```
dataset.nunique()
```

id	9269243
amount	9730
description	18471
date	0
itemName	138579
parent	0
quantity	0
taxAmount	6182
taxDescription	28
type	2
storeId	1829
createdAt	8526946
updatedAt	8526946
taxRate	27
dtype: int64	

J'ai ensuite approfondi l'étude des colonnes "type", "description", "itemName", "taxRate" et "amount".

J'ai choisi ces colonnes car les colonnes "date", "quantity" et "parent" ne contiennent que des valeurs manquantes et ne seront donc pas utiles pour notre clustering. Les colonnes "createAt" et "updateAt" n'apportent pas d'informations significatives.

De cette observation, il est ressorti que :

- Les valeurs de description présentes sont peu nombreuses et très variées, et certains articles ont pour description leur itemName.
- La colonne "type" ne contient que 2

valeurs : "menu" et "dish".

2.3 visualisations des données et corrélation

J'ai également effectué des visualisations et des analyses de corrélation entre les variables. J'ai pu déterminer que :

- Il n'y a pas de lien particulier entre les variables.
- Les variables ne sont pas corrélées entre elles.

.

3. Nettoyage des données

3.1 Traitements des valeurs manquantes

- Les valeurs négatives des colonnes numériques sont considérées comme des valeurs manquantes.
- La variable "description" présente énormément de valeurs manquantes et plusieurs valeurs uniques. Étant donné sa nature catégorielle, son imputation par le mode ou toute autre valeur pourrait entraîner d'importantes erreurs de classification. De plus, le nom des articles constitue en lui-même une description des articles et est utilisé comme description dans plusieurs lignes. J'ai donc décidé de supprimer cette variable.
- La variable "itemName" possède très peu de valeurs manquantes (67 sur un jeu de données de plus de 9 000 000 de lignes). Ce nombre étant insignifiant, j'ai décidé de les supprimer.
- J'ai également supprimé les valeurs négatives des colonnes "amount" et "taxAmount", car elles ne correspondent pas à des articles.

```
[104] dataset.drop('description', axis=1, inplace=True)
      dataset.drop('type', axis=1, inplace=True)
```

```
[105] dataset = dataset.dropna(subset=['itemName'])
```

```
dataset.loc[dataset.amount<0]

[107] dataset = dataset.loc[dataset.amount>=0].reset_index()

[108] dataset.loc[dataset.taxAmount<0]

[109] dataset = dataset.loc[dataset.taxAmount>=0].reset_index()
```

J'ai ensuite effectué une nouvelle étude de la corrélation pour vérifier nos variables.

4. Sélection des colonnes utiles

- Les variables avec 100% de valeurs manquantes ont été supprimées, ainsi que les colonnes "createAt" et "updateAt". La colonne "taxDescription" a également été supprimée car elle ne fournissait pas d'informations pertinentes.

```
features_cols = ["amount", "itemName", "taxAmount", "taxRate"]

[112] dataset2 = dataset[features_cols]
```

5. Modeler et clustering

Pour le modèle, j'ai effectué d'autres traitements sur les données :

5.1 suppression des doublons

- Suppression des doublons afin de réduire le nombre de données.

```
dataset2.shape

(9240982, 4)

[115] dataset2 = dataset2.drop_duplicates(subset=["amount", "itemName", "taxAmount", "taxRate"], ignore_index=True)

[116] dataset2.shape

(634417, 4)
```

5.2 sélection des features

j'ai fait la Sélection des features sur mon nouvel ensemble de données en sélectionnant 60 000 lignes pour effectuer le clustering en raison de contraintes de ressources (c'est la valeur maximale qui ne saturait pas ma mémoire vive).

```
# Sélection des colonnes nécessaires pour la classification
features_cols = ["amount", "itemName", "taxAmount", "taxRate"]
```

```
num_rows_to_keep = 60000
# Échantillonnage aléatoire pour sélectionner un sous-ensemble du dataset
sampled_dataset = dataset.sample(n=num_rows_to_keep, random_state=42)
dataset2 = sampled_dataset[features_cols]
```

.

5. 3 Échantillonnage des données et suppression des caractères

```
num_rows_to_keep = 60000
# Échantillonnage aléatoire pour sélectionner un sous-ensemble du dataset
sampled_dataset = dataset.sample(n=num_rows_to_keep, random_state=42)
dataset2 = sampled_dataset[features_cols]
```

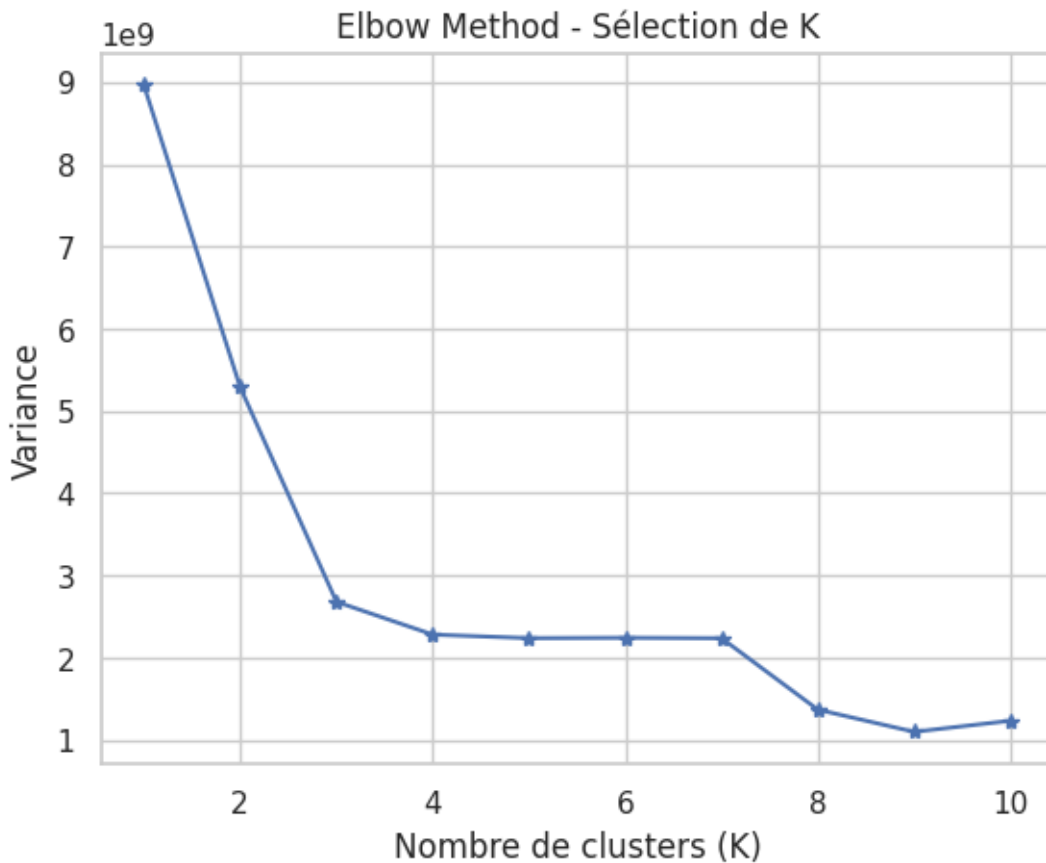
.

Par la suite j'ai fait un Échantillonnage des données et suppression des caractères spéciaux de "itemName" en utilisant une normalisation avec un vectorizer, puis suppression des doublons potentiels résultant du traitement sur "itemName".

. 5. 4 Selection du meilleur k

J'ai aussi sélectionné le meilleur K (nombre de clusters) en utilisant la méthode du coude. J'ai affiché la courbe pour visualiser les variances en fonction du nombre de clusters afin de mieux observer et confirmer, si nécessaire, le meilleur nombre de clusters sélectionné en examinant le point d'inflexion (le "coude") dans la courbe.


```
# Sélection du meilleur K en examinant le coude dans la courbe
best_k = 1
for i in range(1, len(variances)-1):
    diff1 = variances[i] - variances[i-1]
    diff2 = variances[i+1] - variances[i]
    if diff2 < diff1:
        best_k = i + 1
        break
```



5.5 Clustering

Ensuite, j'ai effectué le clustering K-means avec le meilleur nombre de clusters trouvé précédemment et j'ai assigné les clusters aux articles. Les résultats du clustering ont été affichés et des observations ont été faites pour mieux comprendre la répartition des données

```
[150] X_train.loc[X_train.cluster_label==1].taxRate.unique()

array([550, 600, 450, 300, 250, 700])

[149] X_train.loc[X_train.cluster_label==2].taxRate.unique()

array([ 0, 210, 100, 105, 217])

[151] X_train.loc[X_train.cluster_label==3].taxRate.unique()

array([2000, 1600, 2100, 2200, 1500])

[153] X_train.loc[X_train.cluster_label==0].taxRate.unique()

array([1000, 850, 1200, 770, 1300, 1400])
```

Une première remarque importante est que les mêmes valeurs de taux ne se répètent pas dans deux clusters différents. De plus, les taux dans les clusters sont divisés en intervalles qui se chevauchent. Cette observation suggère que le modèle de clustering a réussi à regrouper les articles en fonction de leurs taux, en créant des clusters distincts et en les distinguant les uns des autres.

6. Classify_items

Dans cette partie, j'ai effectué le mappage des clusters précédemment assignés à des catégories spécifiques en utilisant un dictionnaire de correspondance (`category_mapping`). Chaque cluster est associé à une catégorie spécifique.

Ensuite, j'ai défini une fonction `classify_item` pour classer un article donné dans une catégorie. Voici les étapes de la fonction :

1. Le nom de l'article est nettoyé en supprimant les caractères spéciaux à l'aide d'expressions régulières.
2. Les caractéristiques de l'article sont extraites en utilisant le vectorizer TF-IDF préalablement ajusté (`vectorizer.transform`).
3. Les caractéristiques de l'article sont combinées avec les autres variables (montant, montant de taxe, taux de taxe) dans un tableau numpy.

4. Le cluster prédit est obtenu en appliquant le modèle K-means (`kmeans.predict``) sur les caractéristiques de l'article.
5. La catégorie correspondante est obtenue à partir du mappage des clusters (`category_mapping``).
6. La fonction renvoie la catégorie prédite pour l'article donné.

```
# Mapping des clusters aux catégories
category_mapping = {

    0: "Catégorie A",
    1: "Catégorie B",
    2: "Catégorie C",
    3: "Catégorie D",
    # Ajoutez les autres catégories et leurs mappings ici
}

# Fonction de classification d'un item
def classify_item(item_name, amount, tax_amount, tax_rate):
    cleaned_item = re.sub(r'^\w\s\d|_', '', item_name).strip()
    item_features = vectorizer.transform([cleaned_item])
    item_all_features = np.concatenate([item_features.toarray(), [[amount, tax_amount, tax_rate]]], axis=1)
    predicted_cluster = kmeans.predict(item_all_features)
    category = category_mapping.get(predicted_cluster[0], "Catégorie inconnue")
    return category
```

Enfin, des tests ont été effectués pour vérifier la fonction `classify_item``.

```
[136] # Exemple d'utilisation de la fonction de classification
# Test de classificatio
item_name = " pot glace 1b"
amount = 3.50
tax_amount = 0.32
tax_rate = 1000
category = classify_item(item_name, amount, tax_amount, tax_rate)
print("Catégorie associée à l'item :", category)

Catégorie associée à l'item : Catégorie A
```

CONCLUSION

ce rapport décrit les étapes suivies pour résoudre le problème de classification des articles dans les tickets de caisse de KillBills. Nous avons analysé les données, traité les valeurs manquantes et effectué une sélection des variables pertinentes. En utilisant le modèle de clustering K-means, nous avons regroupé les articles en clusters et les avons associés à des catégories spécifiques. En développant une fonction de classification, nous pouvons désormais prédire la catégorie d'un article donné. Ces résultats fournissent une solution efficace pour automatiser la catégorisation des tickets de caisse de KillBills.