**Projet basé sur les recommandations avec le machine learning**

**Ce projet a été réalisé par notre équipe de développeur en intelligence artificielle :**

**Yanice Guigou, Brahiman Koulibaly et Marie Schutz**

**Avant propos :**

1. **présentation du sujet**

**Nous avons développé un moteur de recommandation de livre, à partir d'avis d'utilisateurs.**

**Pour notre projet nous avons disposé d’ une liste d'avis utilisateurs, et de fichiers csv (** [**https://github.com/zygmuntz/goodbooks-10k**](https://github.com/zygmuntz/goodbooks-10k) **).**

**Notre projet est disponible via : (** [**https://github.com/SimplonAI/books/blob/main/main.py**](https://github.com/SimplonAI/books/blob/main/main.py) **)**

**to read.csv : fournit les identifiants des livres marqués "à lire" par chaque utilisateur :**

**user\_id -> identifiant utilisateur**

**book\_id-> identifiant du livre**

**books.csv : données pour chaque livre :**

**book\_id-> identifiant du livre, goodreads book id-> identique à book\_id,**

**best book\_id-> identique goodreads book id, work id-> sans édition**

**books count-> nbre de livre, isbn-> référence**

**isbn13-> référence, authors-> auteur**

**original publication year-> année, original title-> titre original**

**title-> titre, language\_code-> langue**

**average rating-> moyenne des notes, ratings count->nbre de notes**

**work\_ratings\_count-> nbre de notes de travail,**

**work text\_reviews\_count-> critiques de texte**

**ratings\_1,ratings\_2,ratings\_3,ratings\_4,ratings\_5-> note de la presse**

**image\_url-> lien ,mall image\_url-> image**

**Book tags.csv contient des genres attribués par les utilisateurs aux livres ;**

**goodreads\_book\_id-> identique à book\_id, tag\_id-> identifiant du tag, count-> nombre de tag**

**tags.csv contient des tags utilisateurs :**

**tag\_id-> identifiant de l’utilisateur, tag\_name-> les tags**

**ratings.csv contient des évaluations. Il fait 69 Mo**

**user\_id-> identifiant de l’utilisateur, book\_id-> identifiant du livre,**

**rating-> note de l’utilisateur pour le livre**

**Notre solution retourne au moins un article et fournit une recommandation aussi bien à un lecteur novice, qu'à un client habituel qui favorise un certain type ou genre de livre.**

**Nous avons créé un environnement virtuel et travaillons avec vscode.**

**Nous avons conçu une base de données pour stocker les livres, et y importer la liste d'avis fournis. Notre base de données permet de trier ou de filtrer les livres par auteur, éditeur, genre, année de parution.**

**Pour commencer et avoir accès au données, nous vous présentons les étapes :**

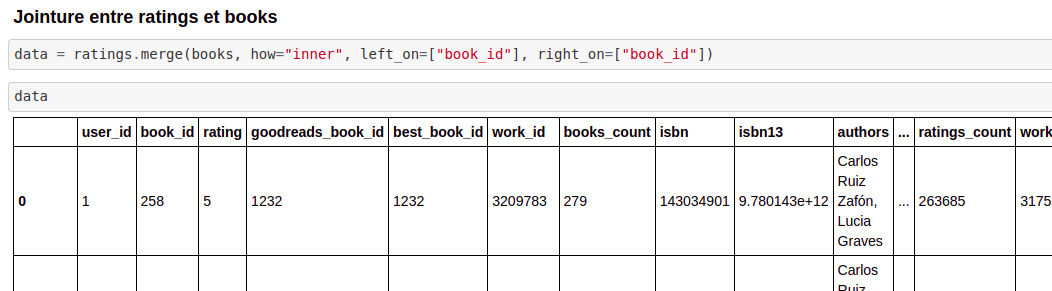
**2 : Import et chargement des données :**

****

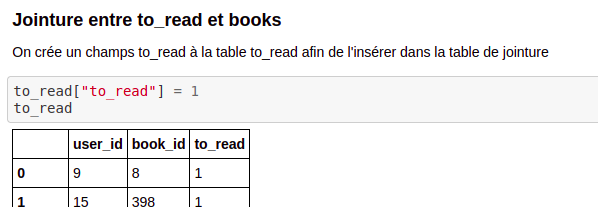
**Avoir un seul csv :**

**Avoir accès aux données de façon plus ordonnée et aussi de pouvoir les visualiser.**

**3 : fusion des csv**

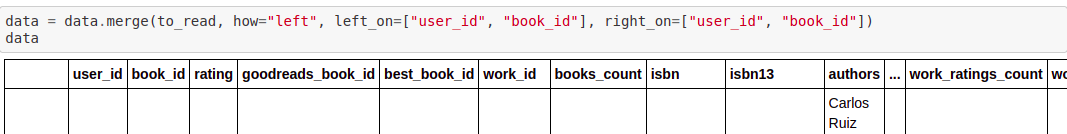
****

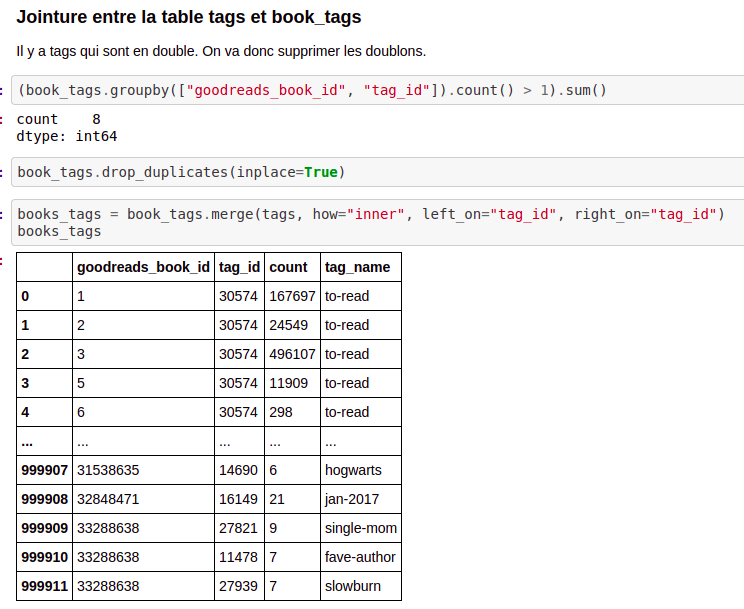
**Afin de pouvoir joindre un csv qui ne contient pas de valeur différente :**

****

**Les correspondances de colonne entre les tables doivent correspondre :**

“user\_id” et “book\_id” et au moins une colonne différente doit être mergé

****

****

**4 : Nettoyage et traitements des colonnes :**

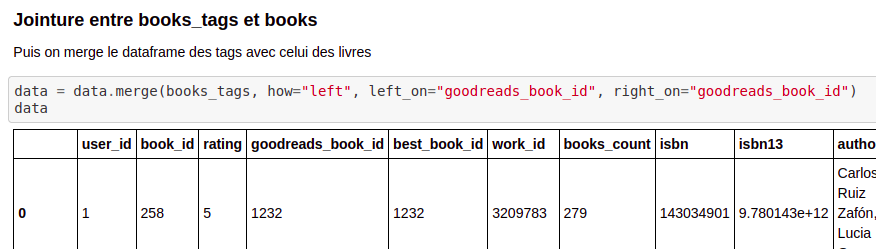
**Pour pouvoir travailler avec les données et qu’elles soient ainsi pertinentes.**

**Les tags :**

Les lignes contiennent de nombreux tags inutiles et/ou mal écrits . De ce fait, nous allons générer un regex afin de nettoyer cette colonne.

****

**Nous ne conservons que les tags ou leurs nombres est supérieurs à 100 .**

****

**On vérifie que tous les livres ont au moins un tag :**(data["tags"] == '').sum() + data["tags"].isna().sum()

Transformation de certaines colonnes pour remplacer les na par des valeurs par défaut :

data['to\_read'].fillna(0, inplace=True)

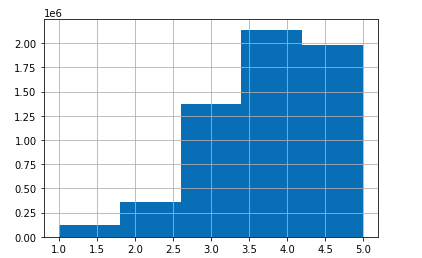
data['isbn13'].fillna('', inplace=True)

data['isbn'].fillna('', inplace=True)

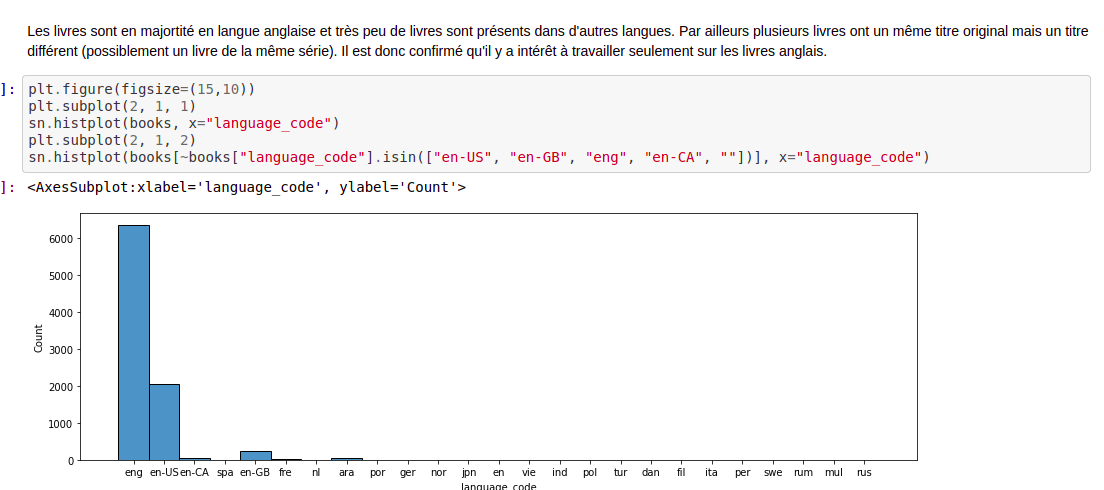
**Traitement de la colonne year:**

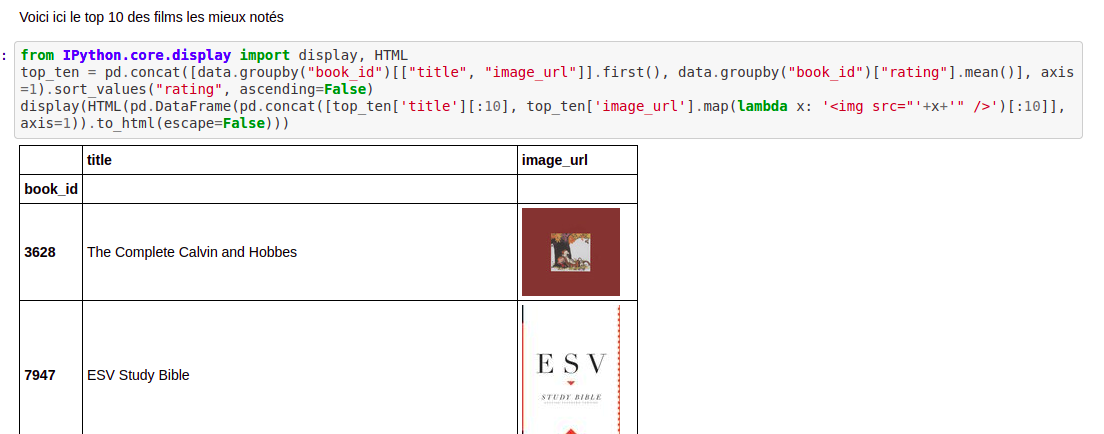
data['original\_publication\_year'] = data['original\_publication\_year'].astype("Int64")

## **5 : Analyse des distributions**

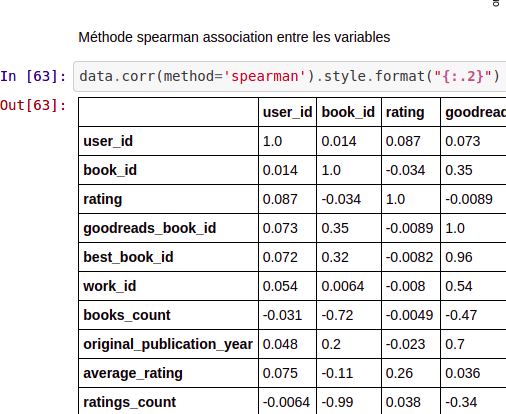
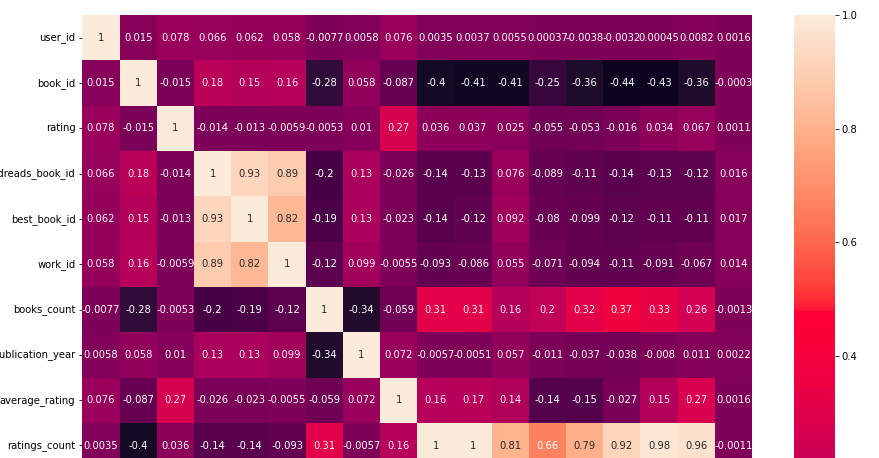
****

Les notes sont le plus souvent comprises entre 3.5 et 5

****

**Les films**

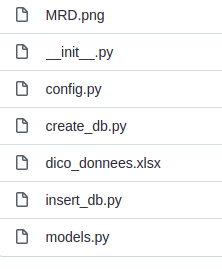
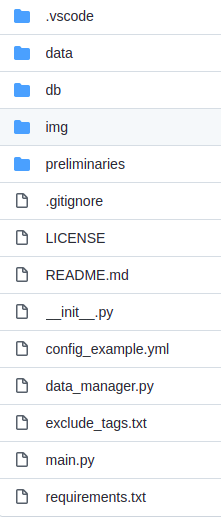
**6 : recherche de corrélations entre variables**

****

**7 : La base de données**

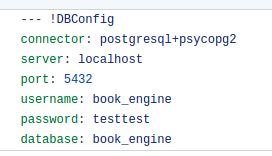
**Pour stocker les nouveaux tags nous avons besoin d’une base de données**

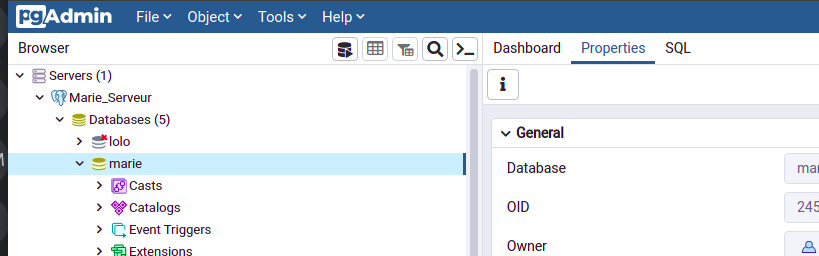
**qui gère les données stockées dans une base de données relationnelle PostgreSQL est supporté avec Python. “ schéma ” :**

****

**Parametre de connection pour la base de données du dossier**

**config\_example.yml**

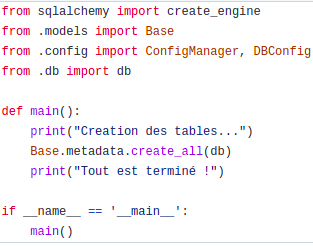
****

****

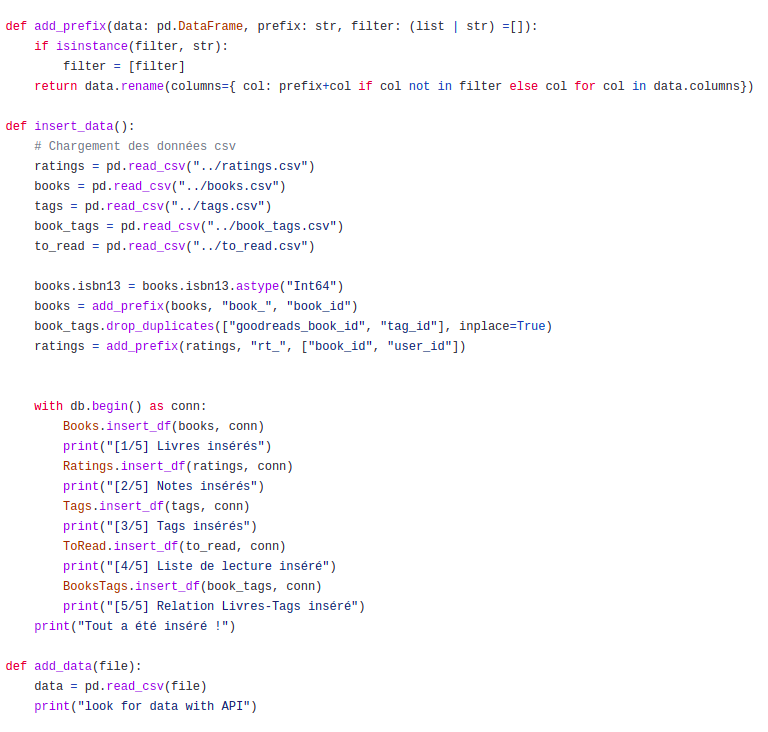
**Connection à la base de données : Via pgAdmin**

****

**Création des tables dans la base de données :**

****

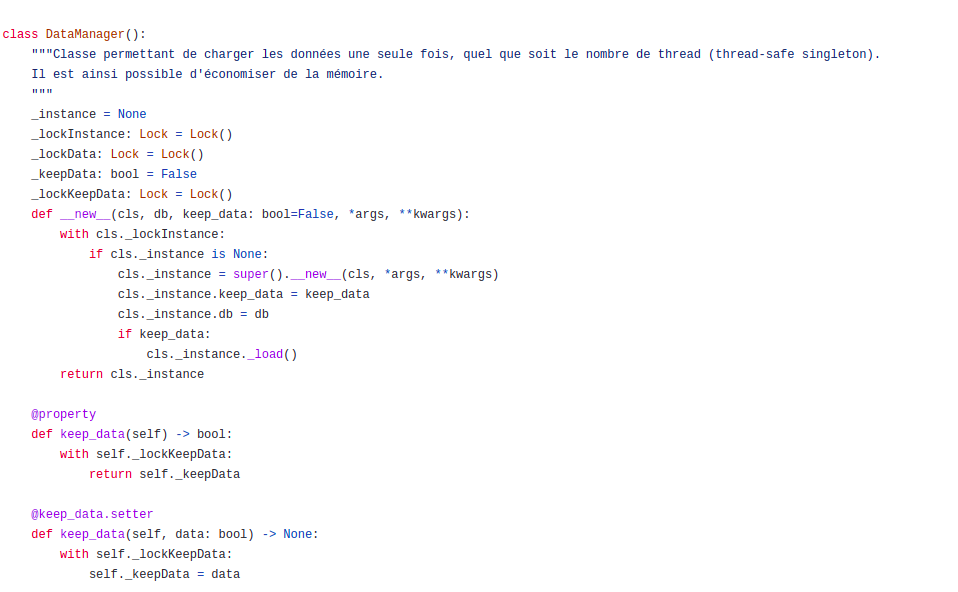
**Fonction d’insertion des tables :**

****

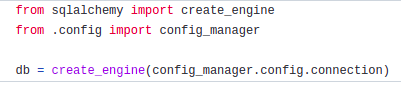
**Exemple avec la classe Tags :**

****

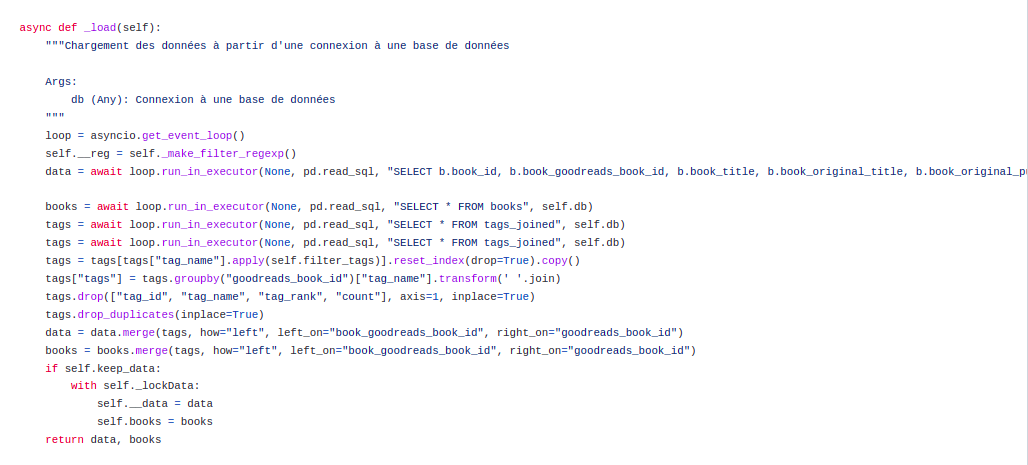
**Fonction de vérification de connection**

****

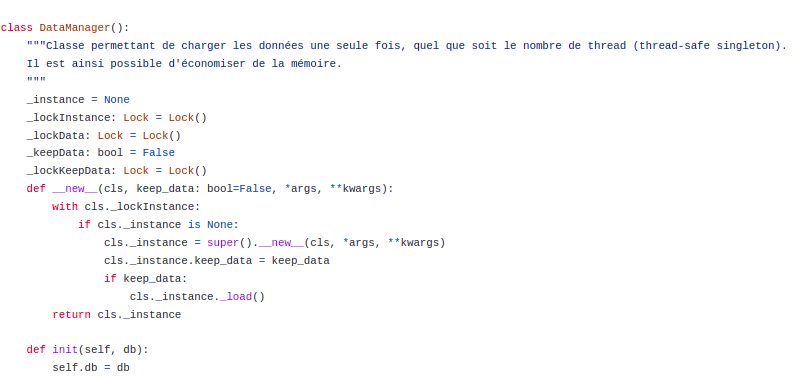
**db :**

****

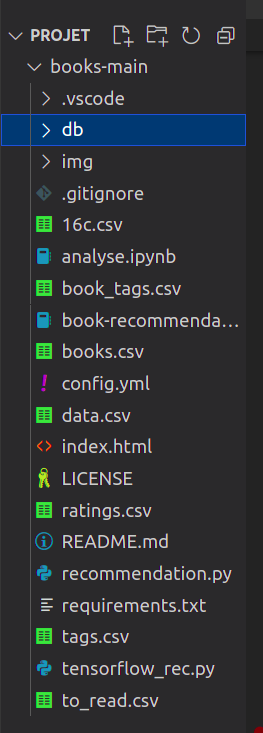
**db :**

****

**classe DataManager :**

****

**8 : Notre architecture avec Flask**

****

**Flask nous permet de créer un application et d’y intégrer une architecture**

**Avec jinja 2 on peut aussi faire une interface graphique.**

**9: Le choix des modèles de recommandation :**

**content based**

Après réflexion il s’est avéré que nous avions besoin des données issues des tags car il fallait générer une recommandation basée sur les avis des clients mais nous avions aussi besoin du titre de l’article ainsi que de sa position à savoir son index. Notre choix s’est donc porté sur un modèle pouvant intégrer des variables de type texte afin d’en ressortir des similitudes.

**Définition shoppé sur le net : )**

**Cet algorithme analyse un ensemble de contenu sans prendre en compte les utilisateurs** (en tout cas, pas dans un premier temps) **et détecte les similarités entre les contenus à des fins de recommandation en inspectant son contenu**. Pour le content-based, l’analyse de contenu consiste à **identifier le tags d’un contenu en répertoriant tous les mots** puis en comparant tous les mots de l’article analysés aux autres articles. Plus un article aura un nombre de mots similaires, plus ces articles seront considérés comme « proches » permettant ainsi de détecter les sujets identiques ou similaires et d’en déduire des recommandations pour le lecteur.

**Notre choix s’est donc porté sur un content base pour le 1er modèle :**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.svm import LinearSVC

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity, linear\_kernel

# Import des librairies

data = pd.read\_csv("data.csv",index\_col=0, dtype={'user\_id':'int64','book\_id':'int64','rating':'int64','goodreads\_book\_id':'int64','best\_book\_id':'int64','work\_id':'int64','books\_count':'int64','isbn':'object','isbn13':'float64','authors':'object','original\_publication\_year':'float64','original\_title':'object','title':'object','language\_code':'object','average\_rating':'float64','ratings\_count':'int64','work\_ratings\_count':'int64','work\_text\_reviews\_count':'int64','ratings\_1':'int64','ratings\_2':'int64','ratings\_3':'int64','ratings\_4':'int64','ratings\_5':'int64','image\_url':'object','small\_image\_url':'object','to\_read':'float64','tags':'object'})

# Lecture du fichier csv regroupant tous les csv nettoyés

#%%

data.drop\_duplicates(subset=['title'],inplace=True)

# Après étude la plupart des livres sont en multiples exemplaires

# s'agissant d'un gros csv très long à charger il faut enlever

# les titres en double.

#%%

data.reset\_index(inplace=True)

# Cela permet de réindexer

def recommandations(title, cosine\_sim, indices):

idx = indices[title]

# Obtenir l'index du film qui correspond au titre

similitude\_scores = list(enumerate(cosine\_sim[idx]))

# Obtenir les scores de similarité

similitude\_scores = sorted(similitude\_scores, key=lambda x: x[1], reverse=True)

# Trier les films en fonction des scores de similarité

similitude\_scores = similitude\_scores[0:5]+ similitude\_scores[40:45]

# Obtenir les scores des films les plus similaires

talk\_indices = [i[0] for i in similitude\_scores]

# Obtenir les index des films

return data['title'].iloc[talk\_indices]

# Renvoie les films les plus similaires

indices = pd.Series(data.index, index=data['title']).drop\_duplicates()

# renvoie une serie avec un index et le titre enleve les doublons

stock = data['tags']

# Tags est stocké dans transcript

tfidf = TfidfVectorizer()

tfidf\_matrix = tfidf.fit\_transform(stock)

# Construire une matrice

cosine\_sim = cosine\_similarity(tfidf\_matrix,tfidf\_matrix)

# Génère une matrice de similarité cosinus

cosine\_sim

#%%

print(recommandations('The Nix', cosine\_sim, indices))

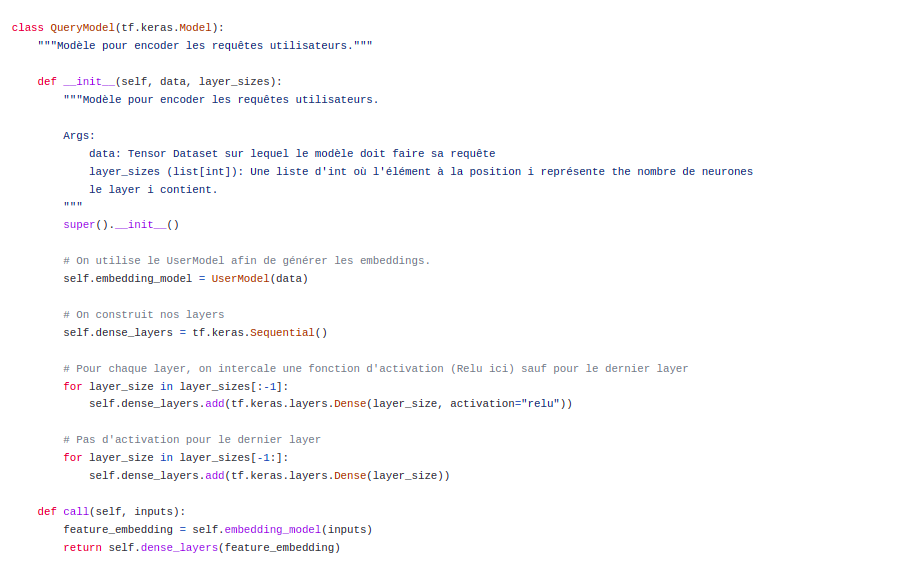
# Contenu des recommandations

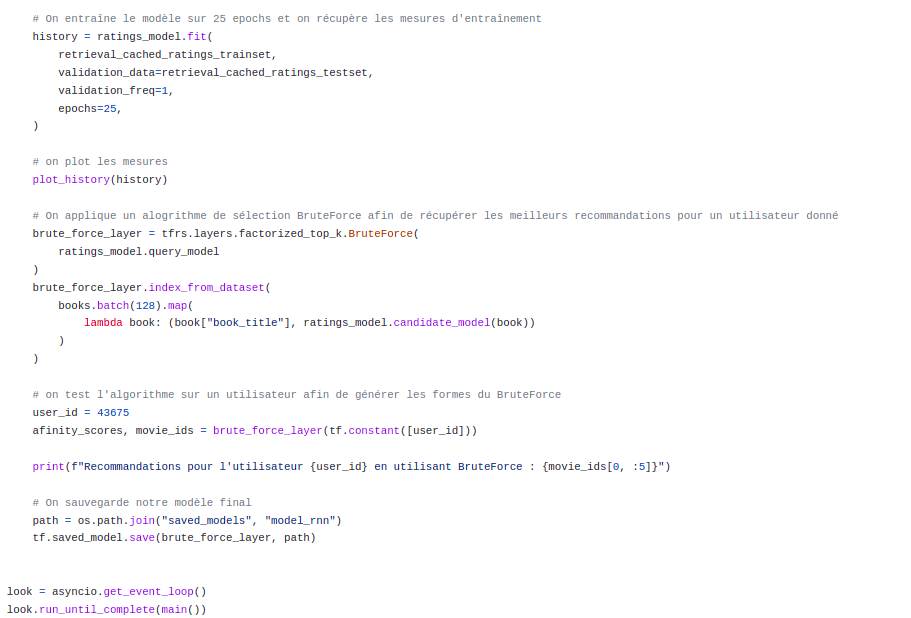
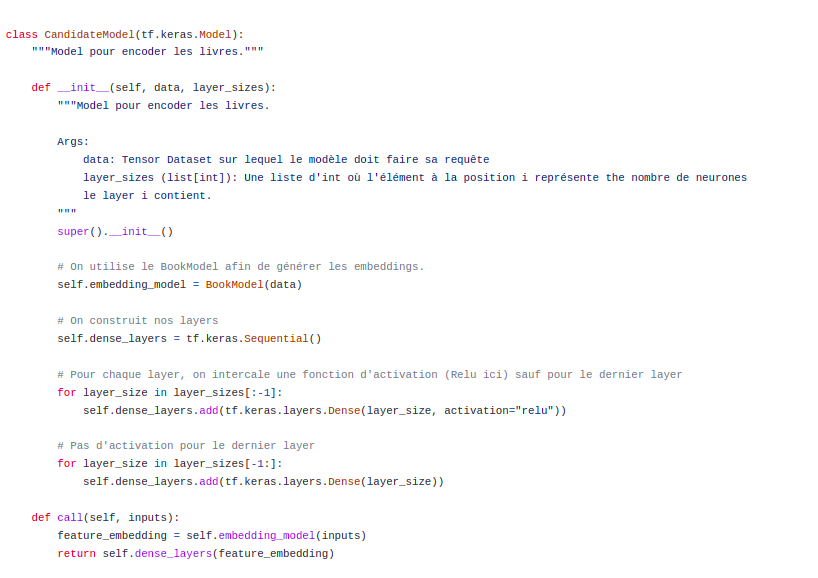
Ce modèle est commenté pour les explications à savoir que le nombre important de données faisait ralentir les divers chargements de codes et que l’astuce était de réduire les livres de façon unique car de 9 millions de données il n’en restait plus que 9000 et ceux afin que le modèle fonctionne facilement :

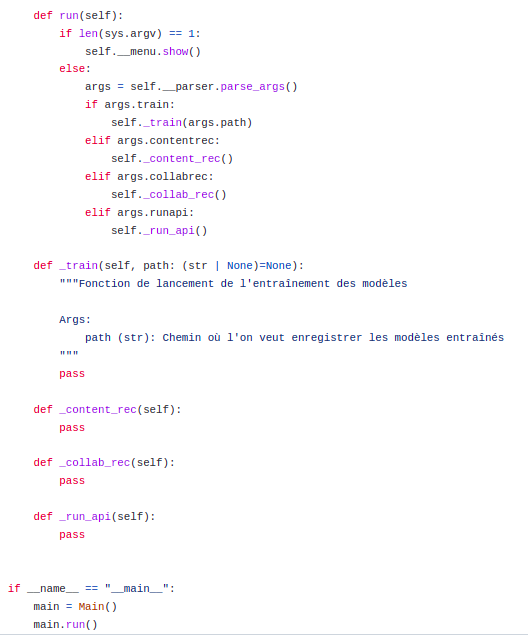
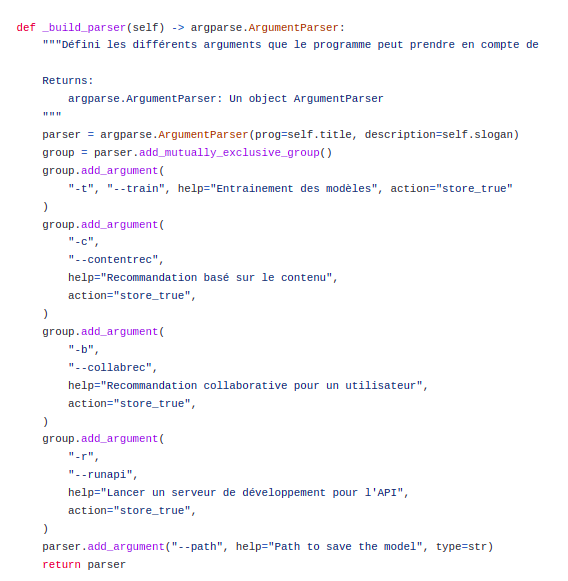
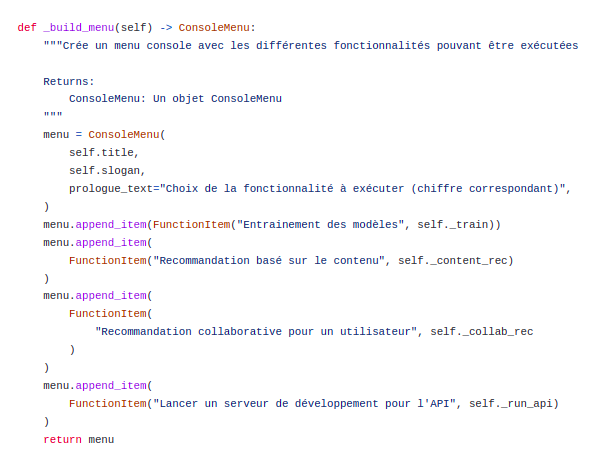
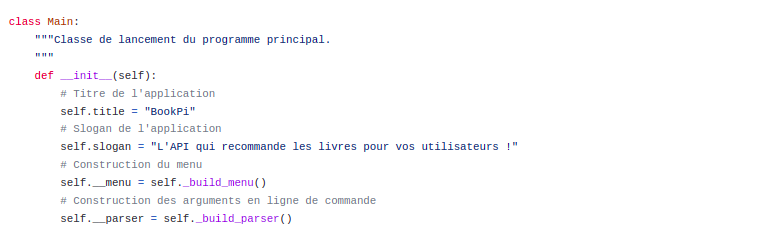
data.drop\_duplicates(subset=[‘title’],inplace=True)

**Un autre modèle … Collaborative filtering**

****

****

****

**10 : main.py**