

1. Scraping des Données

Le processus de scraping consiste à extraire des informations depuis une page web. Pour notre projet, nous avons utilisé la bibliothèque **BeautifulSoup** pour parser le contenu HTML et **requests** pour envoyer des requêtes HTTP à la page contenant les livres.

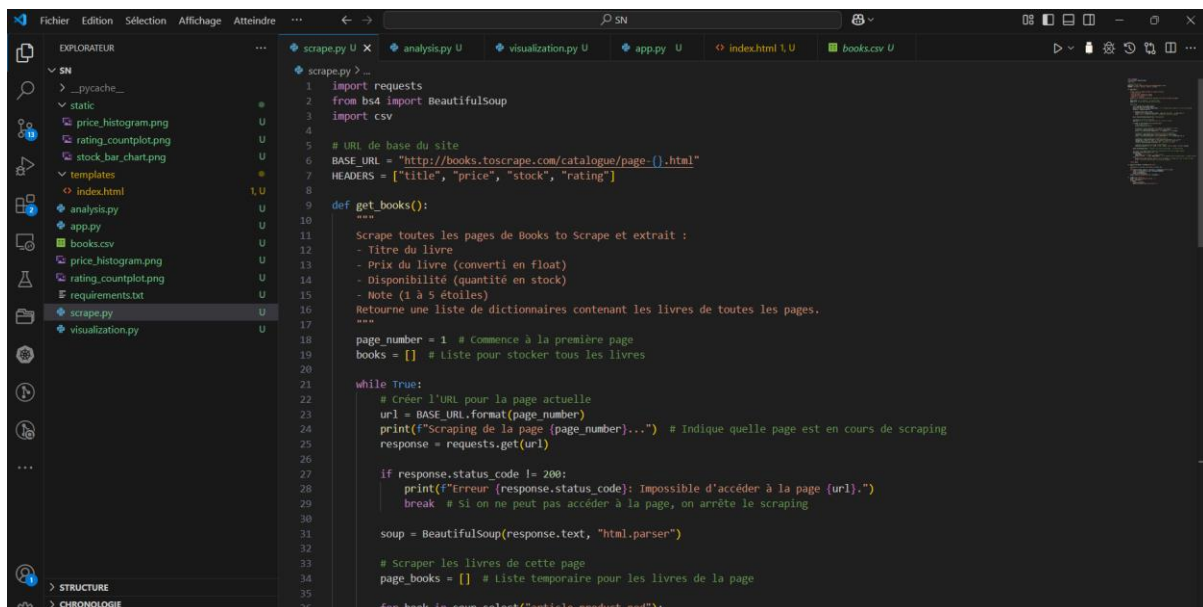
Ce script effectue le web scraping du site **Books to Scrape** en extrayant les **titres**, **prix**, **stocks** et **notes** des livres sur toutes les pages du catalogue. Il utilise la bibliothèque **requests** pour récupérer le HTML des pages et **BeautifulSoup** pour analyser et extraire les données pertinentes. Chaque livre est stocké sous forme de dictionnaire, et les résultats sont enregistrés dans un fichier **CSV**. Le script gère automatiquement la navigation entre les pages et convertit les valeurs extraites (prix en float, stock en int, notes en chiffres). En cas d'erreur ou de page inexistante, il interrompt le scraping proprement.

Voici les étapes détaillées de la récupération des données :

a. Récupération des données

Nous avons visité une page contenant une liste de livres (par exemple, un site comme books.toscrape.com), puis nous avons extrait des informations spécifiques : le **titre**, le **prix** et la **note** de chaque livre.

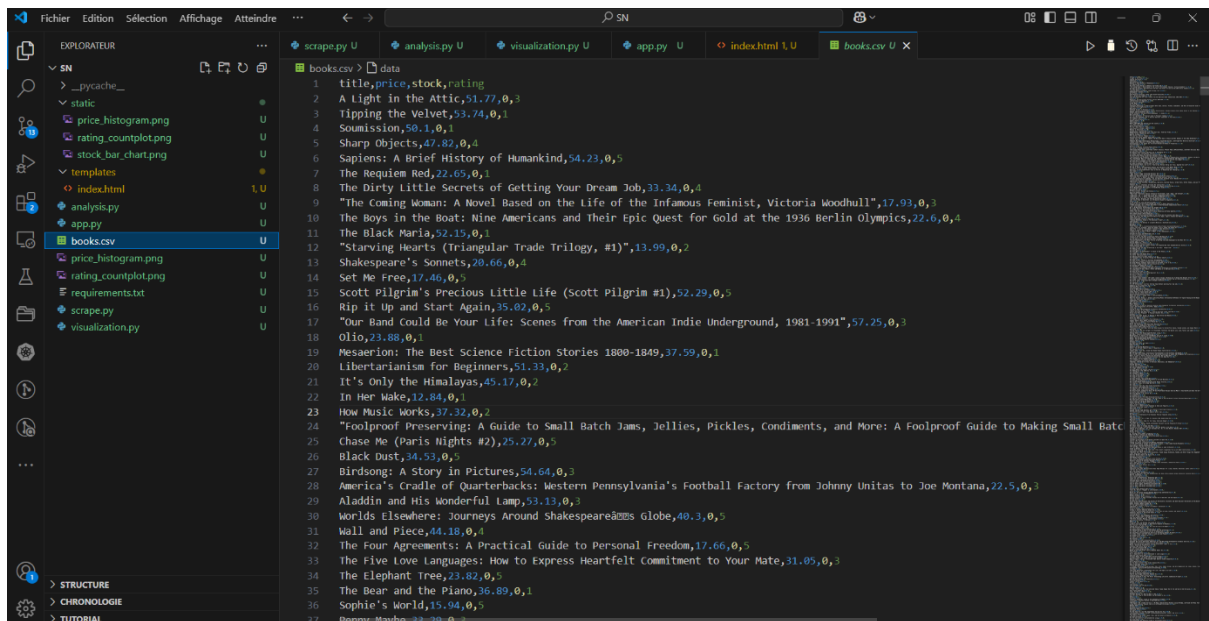
Voici un extrait du code pour récupérer les informations :



```
1 import requests
2 from bs4 import BeautifulSoup
3 import csv
4
5 # URL de base du site
6 BASE_URL = "http://books.toscrape.com/catalogue/page-{}.html"
7 HEADERS = [{"title", "price", "stock", "rating"}]
8
9 def get_books():
10     """
11     Scrape toutes les pages de Books to Scrape et extrait :
12     - Titre du livre
13     - Prix du livre (converti en float)
14     - Disponibilité (quantité en stock)
15     - Note (1 à 5 étoiles)
16     Retourne une liste de dictionnaires contenant les livres de toutes les pages.
17     """
18     page_number = 1 # Commence à la première page
19     books = [] # Liste pour stocker tous les livres
20
21     while True:
22         # Créer l'URL pour la page actuelle
23         url = BASE_URL.format(page_number)
24         print(f"Scraping de la page {page_number}...") # Indique quelle page est en cours de scraping
25         response = requests.get(url)
26
27         if response.status_code != 200:
28             print(f"Erreur {response.status_code}: Impossible d'accéder à la page {url}.")
29             break # Si on ne peut pas accéder à la page, on arrête le scraping
30
31         soup = BeautifulSoup(response.text, "html.parser")
32
33         # Scraper les livres de cette page
34         page_books = [] # Liste temporaire pour les livres de la page
35
36         for book in soup.select("article.product_pod"):
```

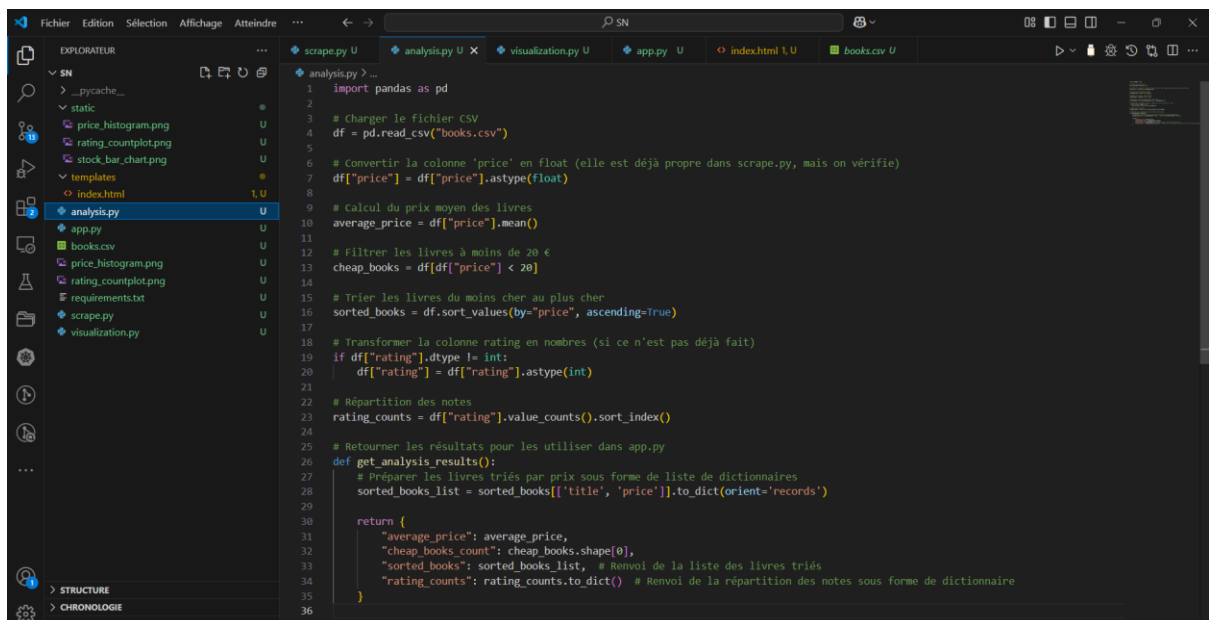
b. Sauvegarde dans un fichier CSV

Une fois les données récupérées, nous avons créé un fichier CSV pour pouvoir les analyser plus tard avec Pandas.



2. Analyse des Données avec Pandas

Une fois les données récupérées et stockées dans un fichier CSV, nous avons utilisé la bibliothèque **Pandas** pour effectuer plusieurs analyses.



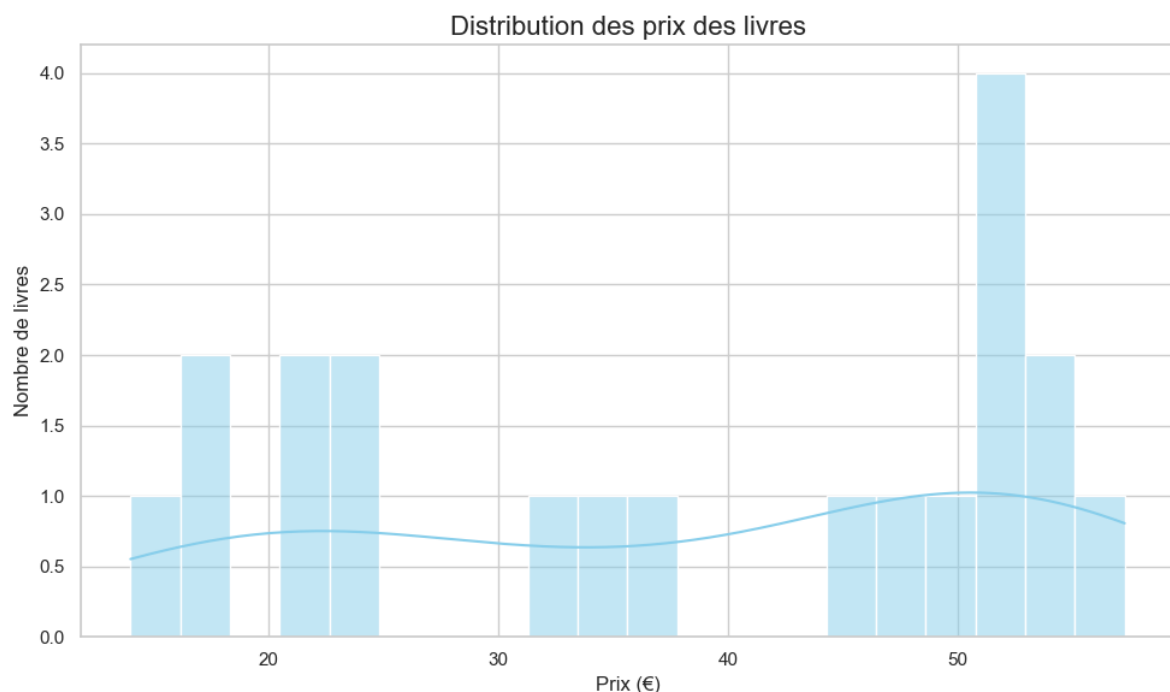
3. Visualisation des Données

Nous avons créé plusieurs graphiques pour mieux visualiser les données, notamment un histogramme des prix et un countplot de la répartition des notes.

```
Fichier Edition Sélection Affichage Atteindre ...
analysis.py U visualization.py U app.py U index.html 1.U books.csv U
EXPLORATEUR
  > _pycache_
  > static
    price_histogram.png U
    rating_countplot.png U
    stock_bar_chart.png U
  > templates
    index.html 1.U
    analysis.py U
    app.py U
    books.csv U
    price_histogram.png U
    rating_countplot.png U
    requirements.txt U
    scrape.py U
    visualization.py U
STRUCTURE
CHRONOLOGIE
TUTORIAL
main.py U Launchpad 0 0 1
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 # Charger le fichier CSV
6 df = pd.read_csv("books.csv")
7
8 # Configurer Seaborn pour une meilleure visualisation
9 sns.set(style="whitegrid")
10
11 # 1. Histogramme des prix
12 plt.figure(figsize=(10, 6))
13 sns.histplot(df["price"], bins=20, kde=True, color="skyblue")
14 plt.title("Distribution des prix des livres", fontsize=16)
15 plt.xlabel("Prix (MAD)", fontsize=12)
16 plt.ylabel("Nombre de livres", fontsize=12)
17 plt.tight_layout()
18 plt.savefig("price_histogram.png") # Sauvegarde le graphique en PNG
19 plt.show()
20
21 # 2. Countplot de la répartition des notes
22 plt.figure(figsize=(8, 6))
23 sns.countplot(x="rating", data=df, palette="viridis")
24 plt.title("Répartition des notes des livres", fontsize=16)
25 plt.xlabel("Note", fontsize=12)
26 plt.ylabel("Nombre de livres", fontsize=12)
27 plt.tight_layout()
28 plt.savefig("rating_countplot.png") # Sauvegarde le graphique en PNG
29 plt.show()
30
31 # 3. Bar chart des stocks
32 plt.figure(figsize=(10, 6))
33 sns.barplot(x="title", y="stock", data=df, palette="blues_d")
34 plt.title("Stocks des livres", fontsize=16)
35 plt.xlabel("Livres", fontsize=12)
36 plt.ylabel("Stock", fontsize=12)
37 plt.xticks(rotation=90)
```

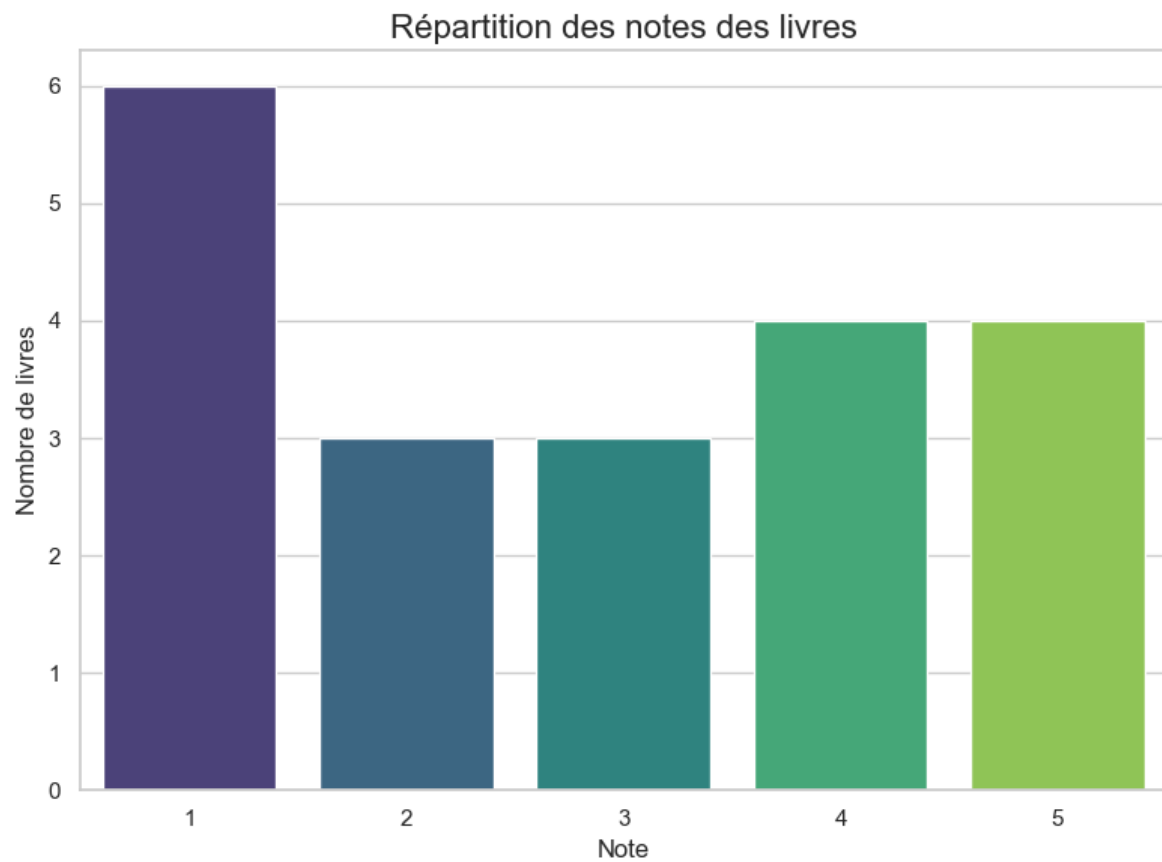
a. Histogramme des prix

Un **histogramme** est utile pour visualiser la distribution des prix des livres. Nous avons utilisé **Seaborn** et **Matplotlib** pour créer cet histogramme :



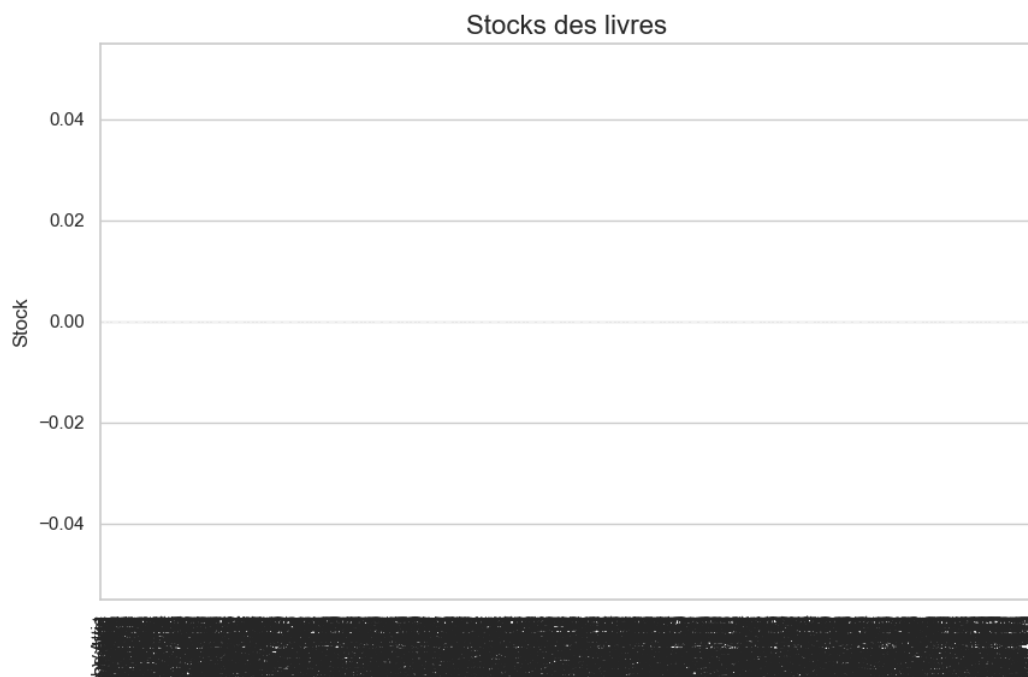
b. Countplot des notes

Un **countplot** est utilisé pour voir la répartition des livres par note :



c. Bar Chart des stocks

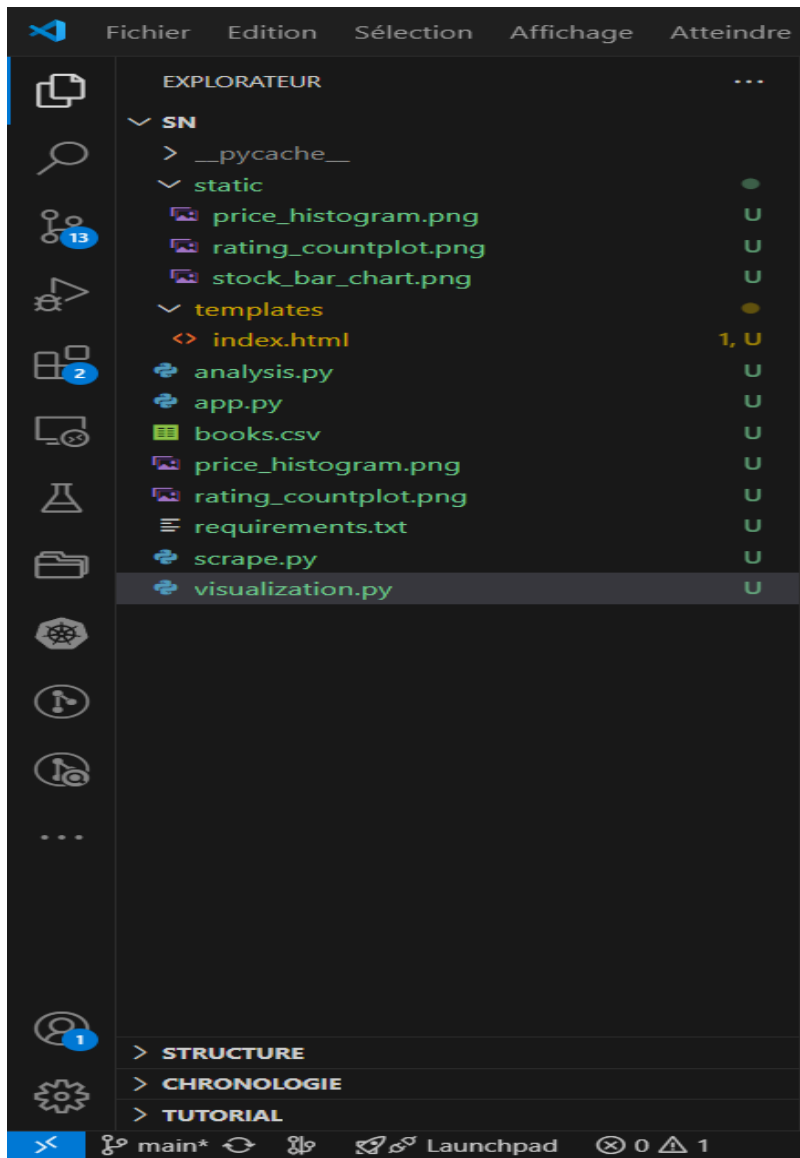
Nous avons ajouté un graphique de type **bar chart** pour visualiser les livres en fonction de leur stock, supposant qu'il existe une colonne stock dans le fichier CSV.



4. Application Flask

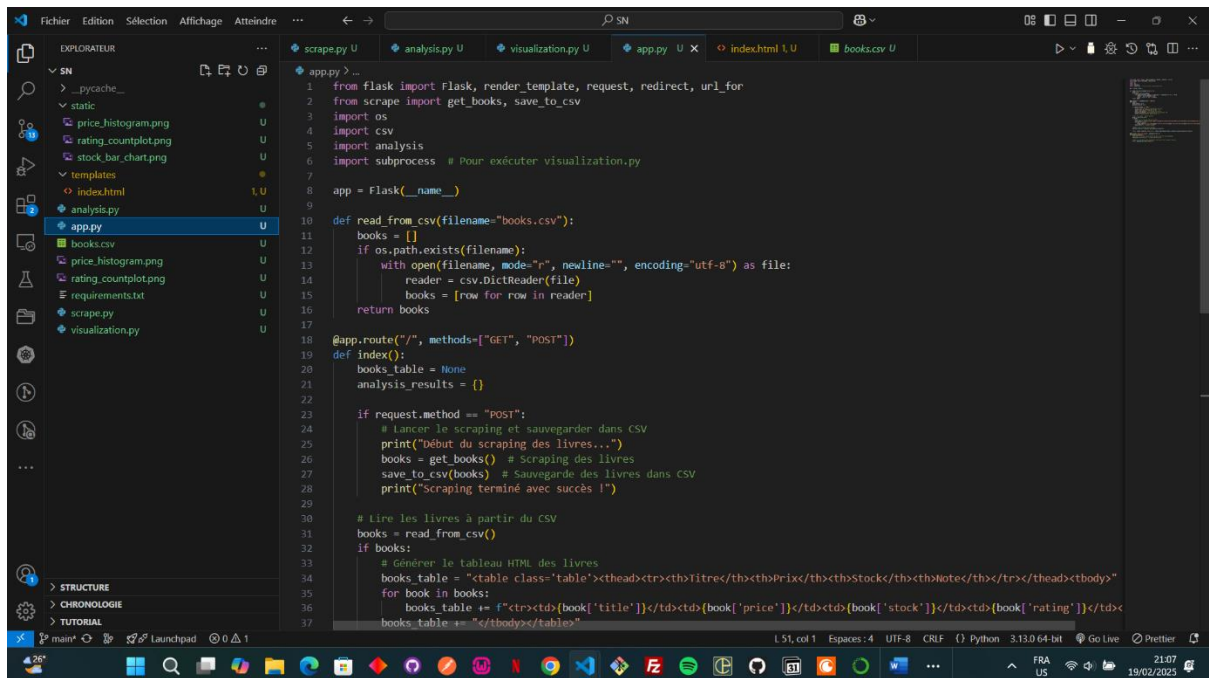
a. Structure de l'App

Nous avons utilisé **Flask** pour créer une mini-application qui permet à l'utilisateur de démarrer le scraping, voir les résultats, et générer les graphiques. L'application Flask suit une structure simple :



b. Code de l'application Flask

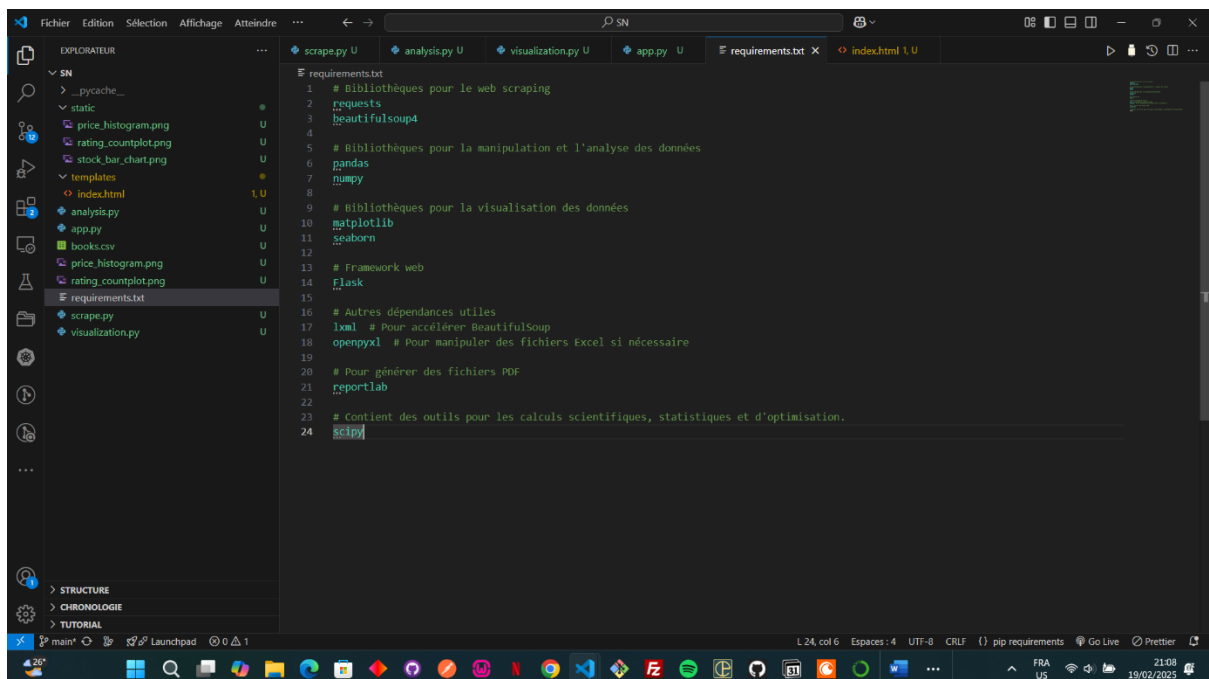
Voici un extrait du code **Flask** pour afficher la page et générer les graphiques :



5. Code Source

Voici les fichiers nécessaires pour le projet :

- **scraping.py** : Code de scraping.
- **visualization.py** : Code de création des graphiques.
- **app.py** : Code Flask.
- **requirements.txt** : Liste des dépendances nécessaires.



En bonus, j'ai scrappé toutes les pages et réalisé d'autres analyses en plus de celles demandées. De plus, au lieu d'exécuter les fichiers un par un, la plateforme permet de lancer directement le scraping et de générer les différents graphiques.