🧪 Projet Python : ScrapBooking & Co.

🎯 Objectifs :

Créer une application complète qui :

- Scrape l'ensemble des livres du site https://books.toscrape.com
- Stocke les données dans deux CSV
- Charge les données dans des DataFrames
- Fait des analyses statistiques (prix, disponibilité, catégories...)
- Créer des visualisations grâce à matplotlib
- Catégoriser les données par rapport aux prix, grâce à l'algorithme K-Means de sklearn
- Reproduire toutes ces étapes pour un scraping de produits sur des sites commerciaux

📦 Structure du projet

```
product_analysis_project/
├── scraper/
│   ├── __init__.py
│   ├── book.py
│   ├── scraper_books.py
│   ├── scraper_custom.py
│   └── exporter.py
├── analysis/
│   ├── __init__.py
│   └── stats.py
├── data/
│   ├── books.csv
│   ├── clustering_price.png
```

```
│   ├── histogram_price.png
│   ├── boxplot_price.png
│   └── custom_products.csv
├── main.py
└── README.md
```

🧩 Étapes et questions du projet (10 parties)

✅ 1. Créer la classe `Book` (`book.py`)

Question : définir une classe avec les attributs suivants :

- `title` : str
- `price` : float
- `availability` : str
- `rating` : str

Ajouter une méthode `.to_dict()` pour exporter les données au format dictionnaire.

✅ 2. Classe `BookScraper` (`scraper.py`)

Question :

Créer une classe `BookScraper` qui contient :

- Une méthode `.scrape_page(url)` → retourne une liste d'objets `Book`
- Une méthode `.scrape_all(pages=50)` → itère sur toutes les pages valides du site

Conseil :

- Gérer `index.html` à part
- Arrêter si une page retourne un code différent de 200

✅ 3. Export CSV (`exporter.py`)

Question :

Créer une fonction `export_books_to_csv(books, filepath)` qui écrit les données dans un fichier CSV.

→ Utiliser la méthode `.to_dict()` de chaque `Book`.

✅ 4. Script principal (`main.py`)

Question :

Créer un script principal qui :

- Lance le scraping
- Stocke les livres dans un CSV (`data/books.csv`)
- Affiche le nombre total de livres récupérés

✅ 5. Lecture et nettoyage des données (`stats.py`)

Créer une fonction `load_books(filepath)` qui lit `books.csv` dans un DataFrame `pandas`.

→ Vérifier que `price` est bien de type float (faire un nettoyage si nécessaire)

✅ 6. Statistiques univariées (`stats.py`)

Créer les fonctions statistiques suivantes :

- `describe_prices(df)` → Calcul de moyenne, médiane, écart-type, quartiles, min/max pour les prix
- `availability_counts(df)` → qui affiche le nombre de livres :
  - disponibles vs non disponibles
  - par niveau de rating (`One` , `Two` , etc.)
- `summary_by_rating(df)` → calcule le prix moyen par rating

✅ 7. Visualisations (`stats.py`)

- Grâce à matplotlib.pyplot, implémenter 4 fonctions à appeler depuis `main.py` :
  1. `plot_price_histogram(df)` : histogramme des prix (20 classes).
  2. `plot_price_boxplot(df)` : boxplot simple des prix.

3. `plot_price_clusters(df)` : scatter plot index/prix coloré par cluster.
4. `plot_cluster_distribution(df)` : boxplot des prix par cluster.

Sortie attendue :

- 4 fichiers PNG dans `data/` :
  - `histogram_price.png`
  - `boxplot_price.png`
  - `clustering_price.png`
  - `cluster_boxplot.png`

✅ 8. Clustering des prix avec algorithme “K-Means” (`stats.py`)

Objectif : regrouper les produits par gamme de prix (bas/moyen/haut) en utilisant une méthode de clustering (algorithme de classification) provenant de la librairie `sklearn` (à installer si nécessaire)

Consignes :

- Utilise `sklearn.cluster.KMeans` avec `n_clusters=3` , vous pourrez vous inspirer de cela :

```
    X = df[['price']]
model = KMeans(n_clusters=n_clusters, random_state=42, n_init='auto')
df['price_cluster'] = model.fit_predict(X)
```

- Implémente dans `stats.py` :
  1. `price_clustering(df)` → ajoute une colonne `price_cluster`
  2. `summary_by_cluster(df)` → retourne `df.groupby('price_cluster')['price'].describe()`
- Créer la visualisation du clustering des prix, via matplotlib.pyplot.

✅ 9. Scraping de produits personnalisés sur plateformes réelles (`scraper_custom.py`)

Question : créer une base de `ProductScraper` personnalisable selon la plateforme :

```
python
CopierModifier
class Product:
    def __init__(self, title, price, source):
        self.title = title
        self.price = price
        self.source = source

    def to_dict(self):
        return {"title": self.title, "price": self.price, "source": self.source}
```

Objectif :

- Scraper environ 250 produits sur la plateforme donné selon les demandes de M.ROSARI (bien se référer aux astuces données et dont nous avons parlé hier en fin de journée)
- Exporter vers `custom_products.csv`

⚠️ À faire **avec des headers d'agent simulés**. Utiliser `requests` et `BeautifulSoup`.

✅ 10. Analyse des produits scrappés

Questions :

- Répéter les étapes 3 à 8 pour les produits scrappés.

📦 Livrables attendus

Projet sur Github avec les accès nécessaire, à rendre aujourd'hui ce soir (23h59 au plus tard)

🧠 Astuces et conseils

1. **Utiliser Headers User-Agent** pour scraper Amazon, eBay ou LeBonCoin :

```
headers = {"User-Agent": "Mozilla/5.0"}
```

2. **Inspecter le HTML** via clic droit > Inspecter → identifier les classes HTML cibles (ex: `.a-price` sur Amazon).
3. Utiliser un **sleep aléatoire** entre les requêtes pour éviter d'être bloqué :

```
import time, random
time.sleep(random.uniform(1, 3))
```

4. Protéger les scrapers via `try/except` pour capturer les erreurs réseau.
5. `BeautifulSoup(html, 'html.parser')` suffit, inutile de parser du JavaScript ici.
6. Pour debug : `print(soup.prettify())` ou `print(soup.select("div.nom_classe"))`