Partie 1

1. Structure du projet

```
.
├── README.md
├── data
│   ├── input
│   │   ├── clinical_trials.csv
│   │   ├── drugs.csv
│   │   └── pubmed.csv
│   └── output
│       └── drugs_catalog.json
├── requirements.txt
├── src
│   ├── app.py
│   └── utils.py
└── sujet
    └── test_python_de.pdf
```

2. Executer la pipeline

De préférence dans un environnemnt virtuel lancer la requête suivante :

```
pip install requirements.txt
```

Une fois les librairies installées :

```
python3 /path/to/app.py
```

3. A propos du projet

On trouvera dans le fichier utils.py quelques fonctions d'aide.

Le fichier app.py est le fichier qui éxécute notre pipeline. Il a été pensé dans une logique ETL afin de faciliter l'éventuelle intégration dans un scheduler de type DAG

4. Réponse à la partie 4 "traitement ad-hoc" du sujet

Dans notre fichier json en sortie, chaque médicament est associé à la liste des journaux dans lesquels il est cité. Il suffit de récupérer pour chaque médicament la taille de cette liste, et de renvoyer la liste avec la plus grande taille sur l'ensemble de nos données.

5. Réponse à la partie 6 du sujet "Pour aller plus loin"

Les calculs que nous effectuons ici s'exécutent de manière linéaire. Il serait plus intéressant, pour pouvoir gérer de grosses volumétries de données, de paralléliser et distribuer nos calculs.

On pourrait par exemple utiliser un cluster Sparks. De plus, cela permet l'utilisation de la librarie PySpark bien plus efficace que Pandas s

Partie 2

Première requête

```
SELECT
Date,
sum(prod_price*prod_qty) as Ventes
FROM transactions
GROUP BY Date
ORDER BY Date
```

Deuxième requête

```
SELECT
A.client_id,
COUNT(case when B.product_type='meuble' THEN 1 ELSE 0 END ) AS
ventes_meubles,
COUNT(case when B.product_type='deco' THEN 1  ELSE 0 END ) AS ventes_deco
FROM transactions as A left join product_nomenclature AS B
ON A.prod_id=B.product_id
WHERE B.product_type='deco'
GROUP BY A.client_id;
```