M1 Data Science – Projet BDDR (2022 – 2023)

Binôme Zeynep BALIKCI et Mariama MBAYE

Objectif : A partir d'un jeu de données, nous allons réaliser une application web permettant de passer des requêtes paramétrées et de visualiser les réponses.

1. Contexte

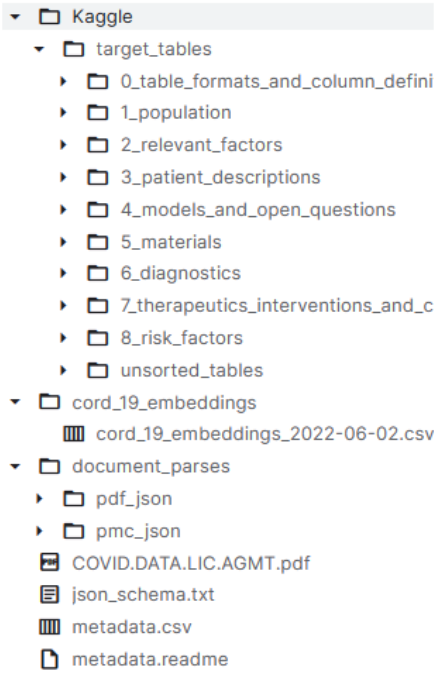
Pendant la pandémie du COVID-19, de nombreux articles de recherche ont été publiés pour mieux comprendre l'impact de la maladie aux différentes échelles. L'urgence de la situation sanitaire nous a aussi mené à une urgence académique : celle de la classification et l'organisation de la recherche (et résultats) générés.

C'est dans ce contexte là que le jeu de données CORD-19 (The COVID-19 Open Research Dataset) est né. Il est conçu pour faciliter le développement de systèmes d'exploration de textes et de recherche d'informations grâce à sa riche collection de métadonnées et de documents structurés en texte intégral.

Sujet :

L'application à développer est destinée aux chercheurs intéressés de trouver rapidement des publications pertinentes dans leur domaine de recherche. Elle doit leur proposer un jeu de formulaires leur permettant d'explorer et de visualiser les données et, si possible, proposer des liens vers des sites web qui pourraient chercher l'article même.

Nous avons à notre disposition un jeu de données installées depuis Kaggle, c'est une arborescence des fichiers CSVs regroupés par thématiques, et de fichiers json:



Socle minimal des requêtes :

1. Liste d'articles par thématique et sous-thématique.
2. Histogramme d'articles publiés par date, semaine, mois.
3. Liste de thématiques
4. Nombre de publications par labo/institution.
5. Liste de journaux par nombre et type de publications.

2. Base de données

2.1 Analyse des données

a) Données nécessaires pour les requêtes

Après étude du jeu de données téléchargé sur Kaggle, nous nous sommes rendus comptes que les données nécessaires à nos requêtes se trouvent dans le fichier metadata.csv, les dossiers /document_parses/pdf_json et /document_parses/pmc_json, et enfin le dossier /Kaggle/target_tables.

Nous avons récupéré les données nécessaires pour les requêtes 1 et 3 à partir du dossier /Kaggle/target_tables : le thème est le nom du dossier et le sous-thème est le nom de chaque fichier .csv se trouvant dans chaque dossier/thème. Et le titre des articles du sous-thème se trouve dans la colonne 'Study' de chaque fichier .csv

```
import pandas as pd
import os
import json

#chemin_archive ="/users/2023/ds1/share/CORD-19"
chemin_archive = "D:/archive"
chemin_tables=f'{chemin_archive}/Kaggle/target_tables'
elements = os.listdir(chemin_tables)
dossiers = [element for element in elements if os.path.isdir(os.path.join(chemin_tables, element))]
for dossier in dossiers[1:-1]:
    theme=(dossier[2:].replace("_"," ")).upper()   # le theme est le nom du dossier
    print(f'{theme=}')
```

```
theme='POPULATION'
theme='RELEVANT FACTORS'
theme='PATIENT DESCRIPTIONS'
theme='MODELS AND OPEN QUESTIONS'
theme='MATERIALS'
theme='DIAGNOSTICS'
theme='THERAPEUTICS INTERVENTIONS AND CLINICAL STUDIES'
theme='RISK FACTORS'
```

Pour la requête 5 (type de publications), nous avons récupérés les données nécessaires dans une colonne des fichiers .csv dans les dossier/thèmes.

Remarque : Il y a très peu d'articles qui ont un sous-thème, et donc il y a très peu d'articles qui ont un type de publication.

La liste de journaux et la date de publication d'un article se trouvent dans le fichier metadata.csv (colonnes publish_time et journal) :

```
DF=pd.read_csv(f'{chemin_archive}/metadata.csv')
DF.columns
```

```
Index(['cord_uid', 'sha', 'source_x', 'title', 'doi', 'pmcid', 'pubmed_id',
       'license', 'abstract', 'publish_time', 'authors', 'journal', 'mag_id',
       'who_covidence_id', 'arxiv_id', 'pdf_json_files', 'pmc_json_files',
       'url', 's2_id'],
      dtype='object')
```

Finalement, pour la requête 4, les affiliations des auteurs ne se trouvent que dans les fichiers du dossier /document_parses/pdf_json Nous avons bien vérifié qu'il n'y avait aucune affiliation dans les fichiers du dossier /document_parses/pmc_json:

```
chemin1 = f'{chemin_archive}/document_parses/pmc_json'
elements1 = os.listdir(chemin1)
no_affiliation_dans_pmc=0
affiliation_dans_pmc=0
for element in elements1:
    with open(f'{chemin1}/{element}', 'r') as f:
        data = json.load(f)
        a=data['metadata']['authors']
        if len(a)!=0:
            for i in range(len(a)):
                if len(data['metadata']['authors'][i]['affiliation'])!=0:
                    affiliation_dans_pmc+=1
                else:
                    no_affiliation_dans_pmc+=1
print(f'{affiliation_dans_pmc=}')
print(f'{no_affiliation_dans_pmc=}')
```

```
affiliation_dans_pmc=0
no_affiliation_dans_pmc=2401600
```

b) Problèmes rencontrés

- Auteurs

Dans les fichiers .json, nous avons la liste des auteurs pour l'article de la forme suivante : [{ 'first' : prénom_auteur1, 'middle' : [...], 'last' : nom_auteur1, 'suffix': … } , { …} , etc...].

Pour un article du fichier metadata.csv le nom/prénom des auteurs (colonne 'authors') sont écrits de la façon suivante : nom_auteur1, prénom_auteur1 suffix/middle_auteur1 ; nom_auteur2, prénom_auteur2 suffixe/middle_auteur2 ; etc... (chaque auteur est séparé pas un " ; " lorsqu'il y a plusieurs auteurs). Nous n'avons pas réussi à déterminer si c'est le suffix ou le middle d'un auteur qui vient en premier dans le fichier metadata.csv. Donc nous avons décidé de récupérer le nom/prénom d'un auteur à partir du fichier metadata.csv car la forme était "plus propre". Mais un problème se soulève : est-ce que nous avons les mêmes informations dans les fichiers .json et dans le metadata.csv ?

Nous avons remarqué les colonnes DF['pmc_json_files'] et DF['pdf_json_files'] et nous nous sommes donc demandé si nous pouvions accéder aux fichiers .json directement à partir du fichier metadata.csv.

```
len(elements1) == len(DF[DF['pmc_json_files'].notnull()])
```

```
True
```

```
no_pmc_files=[]
pmc_files_load=0
pmc_files=DF[DF['pmc_json_files'].notnull()]['pmc_json_files']
for file in pmc_files:
    try:
        with open(f'{chemin_archive}/{file}', 'r') as f:
            data = json.load(f)
            pmc_files_load+=1
    except:
        no_pmc_files.append(file)
print(f'{pmc_files_load=}')
print(f'{no_pmc_files=}')
```

```
pmc_files_load=315742
no_pmc_files=[]
```

Tous les fichiers de la colonne 'pmc_json_files' de metadata.csv existent et sont uniques. Nous avons fait pareil pour les fichiers de la colonnes 'pdf_json_files', mais pour certains articles il y en avait 2 ou même 3 fichiers pdf_json_files séparer par un ' ; '. Mais nous pouvons accéder à tous les fichiers .json du dossier /document_parses/pdf_json à partir du fichier metadata.csv (il y a seulement 2 fichiers .json qui se trouvent dans la colonne 'pdf_json_files' mais qui n'existent pas dans le dossier /document_parses/pdf_json).

Donc nous pouvons accéder aux fichiers .json de chaque article directement à partir du fichier metadata.csv.

Nous avons aussi remarqué que certains articles ont à la fois un fichier pmc_json_files et un fichier pdf_json_files.

```
print("nombre d'article avec un pmc_files et un pdf_files ="
      ,len(DF[DF['pdf_json_files'].notnull() & DF["pmc_json_files"].notnull()]))
print("nombre d'article avec uniquement pmc_files =",len(DF[DF['pdf_json_files'].isnull() & DF["pmc_json_files"].notnull()]))
print("nombre d'article avec uniquement pdf_files =",len(DF[DF['pdf_json_files'].notnull() & DF["pmc_json_files"].isnull()]))
```

```
nombre d'article avec un pmc_files et un pdf_files = 299609
nombre d'article avec uniquement pmc_files = 16133
nombre d'article avec uniquement pdf_files = 74157
```

Nous nous sommes donc demandé si les données dans pdf_json_files et les données dans pmc_json_files sont les mêmes pour un même article ? Et dans le cas où l'article a plusieurs pdf_json_files ?

```
DF_essais=DF[DF['pmc_json_files'].notnull()].reset_index()
for k in range(len(DF_essais)):
    auteurs_metadata=[]
    auteurs_pmc=[]
    ligne=DF_essais['authors'][k]
    file=DF_essais['pmc_json_files'][k]
    if type(ligne)==str:
        for author in ligne.split('; '):
            auteurs_metadata.append(author)
    with open(f'{chemin_archive}/{file}','r') as f:
        data=json.load(f)
        L=data['metadata']['authors']
        if len(L)!=0:
            for i in range(len(L)):
                name=L[i]['last']+', '+L[i]['first']
                auteurs_pmc.append(name)
    if len(auteurs_pmc)!=len(auteurs_metadata):
        print("liste d'auteurs dans metadata.csv pour un article =",'\n',ligne,'\n')
        print("le fichier pmc_json_files =",file,'\n')
        print("liste d'auteurs dans le fichier pmc_json_files pour un article =",'\n',"; ".join(auteurs_pmc))
        break
```

```
liste d'auteurs dans metadata.csv pour un article =
 Froissart, Remy; Roze, Denis; Uzest, Marilyne; Galibert, Lionel; Blanc, Stephane; Michalakis, Yannis

le fichier pmc_json_files = document_parses/pmc_json/PMC1054884.xml.json

liste d'auteurs dans le fichier pmc_json_files pour un article =
 Froissart, Remy; Roze, Denis; Uzest, Marilyne; Galibert, Lionel; Blanc, Stephane; Michalakis, Yannis; Hull, Roger
```

Nous pouvons remarquer qu'il y a un auteur en plus dans le fichier pmc (Hull, Roger), donc nous n'avons pas les mêmes données dans les fichiers .json et le fichier metadata. De même si un article a un fichier pmc_json_files et un fichier pdf_json_files, il n'y a pas les mêmes auteurs dans les 2 fichiers. Et finalement si un article a plusieurs pdf_json_files, il n'y a pas les même données (auteurs, emails, affiliation) dans les ces fichiers. Mais nous allons essayer de récupérer toutes les données disponibles dans notre jeu de données.

- Titre

Mais nous avons rencontré un autre problème avec le fichier metadata.csv car il y a beaucoup de valeurs manquantes DOI et les titres des articles ne sont pas unique :

```
print(len(DF))
print(len(DF['title'].unique()))
```

```
1056660
850367
```

Et il y a 503 articles qui n'ont pas de titre :

```
len(DF[DF['title'].isnull()])
```

```
503
```

Nous avons donc décidé de crée un primary key personnalisé pour la classe Articles, c'est-à-dire, nous allons identifier un article par le numéro de la ligne dans laquelle elle se trouve dans le fichier metadata.csv. Il y aura donc plusieurs articles avec le même titre, mais nous n'avons pas réussi à trouver la raison pour laquelle certains articles avaient le même titre, ou n'avaient pas de titre, donc pour ne pas perdre de donnée nous avons décidé de récupérer tous les articles du fichier metadata.csv.

- Autre découvertes et changement

Nous nous sommes rendus comptes que certains auteurs avaient 2 affiliations (laboratoire et institution), et que certains articles dans les sous_themes avait plusieurs 'Study Type' (types de publications). Nous avons changé le schéma en conséquence. La longueur des nom des affiliations pouvait dépasser les 2000 caractères donc nous avons changé son type en TextField. La date de publication n'était que l'année pour certains articles donc nous avons changé son type en CharField, et avons décidé de le traité comme un datetime après le peuplement.

2.2 Peuplement des tables

- Pour peupler les tables : theme, sous_themes, studytype, journal et affiliation nous avons choisi d'aller chercher directement les données dans les dossiers/fichiers, c'était l'étape la plus simple. Cette étape a été faite avec le module psycopg2.
- Pour peupler les tables : articles, sous_themes_articles et studytype_articles nous avons décidé de reconstruire un autre dataframe à partir du fichier metadata.csv et en allant récupérer les données disponibles dans les dossiers des sous themes pour chaque ligne de metadata.csv de façon à obtenir la liste des studytypes et la liste des sous_themes d'un article/ligne :

```
DF2=pd.DataFrame({'title': DF['title'], 'abstract':DF['abstract'], 'publish_time': DF['publish_time'],'journal': DF['journal'],
                  'url': DF['url'],'studytype': Study_types,'sous_themes': Sous_themes_articles2})
```

```
DF2[DF2['sous_themes']!='NULL']
```

	title	abstract	publish_time	journal	url	studytype	sous_themes
8150	Staff safety during emergency airway managemen...	NaN	2020-02-24	Lancet Respir Med	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7...	[Expert Review]	[What are ways to create hospital infrastructu...
8176	Protecting health-care workers from subclinica...	NaN	2020-02-13	Lancet Respir Med	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7...	[Expert Review]	[What are ways to create hospital infrastructu...
9385	Minimise nosocomial spread of 2019-nCoV when t...	NaN	2020-02-11	Lancet	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7...	[Expert Review]	[What are ways to create hospital infrastructu...
87619	Is COVID-19 the first pandemic that evolves in...	SARS-CoV-2 is a zoonotic virus that has achiev...	2020-04-21	Veterinaria italiana	https://doi.org/10.12834/vetit.2246.12523.1; h...	[Expert Review]	[Evidence that domesticated_farm animals can b...
88228	Current Drugs with Potential for Treatment of ...	PURPOSE SARS-CoV-2 first emerged in China in D...	2020	Journal of pharmacy & pharmaceutical sciences ...	https://doi.org/10.18433/jpps31002; https://ww...	[Expert Review]	[What is the efficacy of novel therapeutics be...

- Pour peupler les tables : authors, articles_authors et affiliation_authors nous avons essayé de construire un autre dataframe de façon à obtenir à la fin 1 ligne = 1 auteur (avec son affiliation1, affiliation2, son email et les id des articles qu'il a écrit). Mais la création du dataframe prenait une journée entière car c'était des listes supplémentaires qu'on créait. Nous avons donc changé notre méthode et nous avons décidé de créer un dataframe à partir d'un dictionnaire tels que dict = { 'un_auteur' : { 'email' : ___ , 'articles_id' : [_,_,_] , 'institution' : ___ , 'labarotory': ___}, 'un_autre_auteur' : {...}, etc...}. Et cette fois, nous avons créé le dataframe seulement pour les auteurs du metadata.csv qui possédaient un fichier pmc_json ou pdf_json. Cela ne prend plus que 2h. Les auteurs restants qui se trouvent sur les lignes du metadata.csv dans laquelle il n'a pas de pmc_json ou de pdf_json seront traité directement pendant le peuplement.

3. Application

L'apparence de notre site est assez brute car nous nous sommes concentrés sur le fait d'afficher les informations dont nous avions besoin plutôt qu'à la façon dont il est affiché.

La requête “liste de thématiques” était la plus simple, et nous avons rendus les noms des thèmes cliquables, c’est-à-dire lorsque vous cliquez sur un thème vous êtes dirigé vers un page dans laquelle vous avez la liste des sous-thèmes de ce thème. En fait cette requête est liée à la requête “liste d’articles par thématique et sous-thématique”, car nous avons aussi rendus les noms des sous-thèmes cliquables, c’est-à-dire lorsque vous cliquez sur un sous-thème vous êtes dirigé vers un page dans laquelle vous avez la liste des articles de ce sous-thème.

Pour la requête “nombre de publications par labo/institution”, nous avons fait plusieurs INNER JOIN, pour obtenir la liste des affiliations ordonnée par leur nombre de publications.

Mais nous n’avons pas réussi à faire de même pour la requête “liste de journaux par nombre et type de publications”. Pour cette requête nous avons décidé de créer un dictionnaire : { ‘un_journal’ : { ‘nb_articles’ : __ , ‘study_types’: [_,_,_] }}. Pour cela nous somme partis de la liste des types de publications car il y a très peu de types de publications, donc la création de ce dictionnaire ne prend pas énormément de temps.

Nous avons bien réussi à récupérer les données pour la création d’un histogramme (nombre d’articles par année, mois, ou semaine sous forme de dictionnaire) mais nous n’avons pas réussi à utiliser l’outil Django Plotly Dash pour la création des histogrammes.

Certains articles avaient plusieurs urls séparé d’un ‘ ; ’ donc nous avons créé un tag personnalisé : split. Elle se trouvent dans le fichier templetags de l’application.

Et pendant la création des vues, nous nous sommes rendus comptes que créent des string ‘NULL’ pour les valeurs manquantes étaient inutiles, c’est mieux si nous n’ajoutons rien dans la base de données si nous n’avons pas le donnée (par exemple les emails manquants, ou bien le journal manquant pour un article). Nous nous sommes rendu compte de cela un peu tard donc nous n’avons pu adapter seulement quelques scripts de peuplement en fonction de cette information.

Requêtes supplémentaires :

1. Liste des affiliations : tous les noms des affiliations sont cliquables, et vous dirige vers une page dans laquelle se trouve la liste des articles liés à un institution/laboratoire.
2. Liste des journaux : tous les noms des journaux sont cliquables, et vous dirige vers une page dans laquelle se trouve la liste des articles publiés dans un journal.
3. Liste de articles : tous les noms des articles sont cliquables, et vous dirige vers une page dans laquelle se trouve toutes les informations d'un article, c’est-à-dire ses auteurs, journal dans laquelle il a été oublié, son type d’études s'il en a une, etc...