# 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

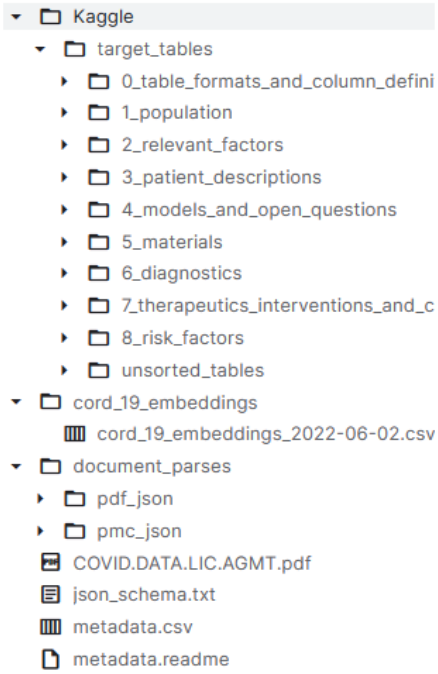
Pendant la pandémie de COVID-19, de nombreux articles de recherche ont été publiés pour mieux comprendre l'impact de la maladie à différentes échelles. L'urgence de la situation sanitaire nous a également menés à une urgence académique : celle de la classification et de l'organisation de la recherche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

C'est dans ce contexte 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 à 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 installé depuis Kaggle, qui est une arborescence de fichiers CSV regroupés par thématiques, ainsi que des 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 installé depuis Kaggle, nous nous sommes rendu compte 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, ainsi que dans 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. 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 récupéré les données nécessaires dans une colonne des fichiers CSV se trouvant dans les dossiers/thèmes.

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

La liste des 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 sous 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 par un " ; " lorsqu'il y a plusieurs auteurs). Nous n'avons pas réussi à déterminer si c'est le suffixe ou le middle d'un auteur qui vient en premier dans le fichier metadata.csv. Par conséquent, nous avons décidé de récupérer le nom/prénom d'un auteur à partir du fichier metadata.csv car la forme était plus propre. Toutefois, un problème se pose : est-ce que nous avons les mêmes informations dans les fichiers .json et dans le metadata.csv ?

Nous avons remarqué les colonnes 'pmc_json_files' et 'pdf_json_files' dans metadata.csv 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 de même pour les fichiers de la colonne 'pdf_json_files', mais pour certains articles il y avait deux ou même trois fichiers pdf_json_files séparés par un ' ; '. Cependant, nous pouvons accéder à tous les fichiers .json du dossier /document_parses/pdf_json à partir du fichier metadata.csv (il y a seulement deux 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 les fichiers pdf_json_files et les données dans les fichiers pmc_json_files sont identiques pour un même article ? Et qu'en est-il dans le cas où l'article a plusieurs fichiers 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 deux fichiers. Et finalement, si un article a plusieurs pdf_json_files, il n'y a pas les mêmes données (auteurs, emails, affiliations) dans ces fichiers. Cependant, 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 pour les DOI et les titres des articles ne sont pas uniques :

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

```
1056660
850367
```

Et il y a 503 articles qui n'ont pas de titre dans le fichier metadata.csv.

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

```
503
```

Nous avons donc décidé de créer une clé primaire personnalisée pour la classe Articles. Nous identifierons ainsi chaque article par le numéro de la ligne correspondante dans le fichier metadata.csv. Il est possible que certains articles aient le même titre et nous n'avons pas réussi à trouver la raison pour laquelle cela se produit, de même que pour les articles sans titre. Afin de ne pas perdre de données, nous avons décidé de récupérer tous les articles du fichier metadata.csv.

- Autres découvertes et changements

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

## 2.2 Peuplement des tables

- Pour peupler les tables : Theme, Sous_Theme, StudyType, Journal et Affiliation, nous avons choisi de récupérer directement les données dans les dossiers/fichiers correspondants, ce qui était l'étape la plus simple. Cette étape a été réalisée à l'aide du module psycopg2.

- Pour peupler les tables Articles, Article_Theme et StudyType_Articles, nous avons décidé de reconstruire un autre dataframe à partir du fichier metadata.csv en allant récupérer les données disponibles dans les dossiers des sous-thèmes pour chaque ligne de metadata.csv. Ainsi, nous obtenons la liste des studytypes et la liste des sous-thèmes pour chaque 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, Author_Article et Author_Affiliation, nous avons d'abord essayé de construire un autre dataframe pour avoir une ligne par auteur avec ses affiliations, son email et les id des articles qu'il a écrit. Cependant, la création du dataframe prenait une journée entière, car nous créions des listes supplémentaires de très grandes tailles en faisant en sorte de ne pas avoir 2 fois le même auteur.

Nous avons alors changé notre méthode et décidé de créer un dataframe à partir d'un dictionnaire contenant les informations pour chaque auteur, tel que dict = { ‘un_auteur’ : { ‘email’ : ___ , ‘articles_id’ : [_,_,_] , ‘institution’ : ___ , ‘labarotory’: ___},

'un_autre_auteur' : {...}, etc...}. Cette fois-ci, nous avons créé le dataframe uniquement pour les auteurs du fichier metadata.csv qui avaient un fichier pmc_json ou pdf_json associé. Cette étape a pris environ 2 heures.

Les auteurs restants, qui n'avaient ni pmc_json ni pdf_json associé, ont été traités directement lors du peuplement.

## 3. Application

L'apparence de notre site est assez brute car nous nous sommes concentrés sur l'affichage des informations dont nous avions besoin plutôt que sur la façon dont elles sont présentées.

La requête "liste de thématiques" était la plus simple, et nous avons rendu les noms des thèmes cliquables. En cliquant sur un thème, vous êtes dirigé vers une page affichant la liste des sous-thèmes correspondants. Cette requête est liée à la requête "liste d'articles par thématique et sous-thématique". Nous avons également rendu les noms des sous-thèmes cliquables, de sorte que vous pouvez cliquer sur un sous-thème pour afficher la liste des articles associés.

Pour la requête "nombre de publications par labo/institution", nous avons effectué plusieurs INNER JOIN pour obtenir la liste des affiliations classées par nombre de publications.

Cependant, nous n'avons pas réussi à effectuer la même opération pour la requête "liste de journaux par nombre et type de publications". Nous avons donc créé un dictionnaire : { 'un_journal' : { 'nb_articles' : __ , 'study_types': [_,_,_] }} à partir de la liste des types de publications. Comme il y a peu de types de publications, la création de ce dictionnaire ne prend pas beaucoup de temps.

Nous avons réussi à récupérer les données nécessaires pour créer 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 créer les histogrammes.

Certains articles avaient plusieurs URL séparés par un ' ; ', donc nous avons créé une balise personnalisée : "split". Elle se trouve dans le fichier templatetags de l'application.

Enfin, pendant la création des vues, nous avons réalisé que la création de chaînes de caractères "NULL" pour les valeurs manquantes était inutile. Il est préférable de ne rien ajouter dans la base de données si nous n'avons pas de données (par exemple, pour les adresses e-mail manquantes ou le journal manquant pour un article). Nous avons réalisé cela un peu tard, donc nous n'avons adapté que quelques scripts de peuplement en fonction de cette information.

### Requêtes supplémentaires :

1. Liste des affiliations : tous les noms des affiliations sont cliquables, et vous dirigent vers une page présentant la liste des articles liés à une institution/laboratoire. Vous pouvez aussi faire une recherche précise par nom d'affiliation.
2. Liste des journaux : tous les noms des journaux sont cliquables, et vous dirigent vers une page présentant la liste des articles publiés dans un journal. Vous pouvez aussi faire une recherche précise par nom de journal.
3. Liste des articles : tous les noms des articles sont cliquables, et vous dirigent vers une page présentant toutes les informations d'un article, telles que ses auteurs, le journal dans lequel il a été publié, son type d'étude s'il en a un, etc... Vous pouvez aussi faire une recherche précise par titre d'article.