BÉQUET Clément
NOËL Sébastien

Le 03/05/2023

PROJET BDDR

• Nettoyage des données et mise en forme

Dans un premier temps, il faut savoir que plusieurs données sont à nettoyer en effet le noms des thématiques et des sous-thématiques se trouvent dans un dossier qui est composé de fichiers inutiles et afin de coder la récupération de ces données il était important que nous supprimions ces derniers.

Par la suite, le fichier Excel « metadata.csv » qui nous a été mis à disposition devait également être nettoyé car ce dernier est la base de nos récupérations de données. Nous avons remarqué que certains titres d'articles étaient doublés et nous avions comme information que le « doi » (Digital Object Identifier) de chaque article était unique, nous avons donc réalisé une suppression de données en ne gardant qu'un exemplaire de chaque « doi ». Cependant, après une vérification de notre travail, nous avons compris que certains articles ne contenaient pas de « doi », c'est-à-dire que la valeur nulle du « doi » était considéré un grand nombre de fois et qu'ainsi tous les articles qui n'avaient pas de « doi » étaient considérés comme les mêmes alors que ce n'était pas le cas. Nous avons donc mis de côté les articles qui ne possédaient pas de « doi » pour ensuite les remettre dans le fichier Excel après avoir supprimé les doublons des titres qui en possèdent.

L'étape finale consistait à créer un fichier Excel nommé « data_final.csv » qui contenait toutes les valeurs précédemment nettoyées ainsi que l'appartenance ou non à une thématique et à une sous-thématique. Pour cela, nous avons ajouté deux colonnes à ce fichier et si l'article était présent dans les fichiers mis à disposition, nous y insérions le nom de sa thématique et de sa sous-thématique et une valeur « Inconnu » dans le cas inverse.

• Peuplement :

➢ Création des tables Thématique et Sous-thématique

Afin de créer ces deux tables, nous avons simplement utiliser la bibliothèque « os » et ensuite sommes allés récupérer les noms des fichiers qui se trouvaient dans le dossier « target_tables ». Pour des soucis d'esthétique, nous avons de plus mis le nom retourné sous forme de titre afin d'avoir des majuscules ainsi que remplacer les tirets du bas par des espaces. Nous avons de plus retiré les chiffres qui pouvaient se trouver au début d'un nom de thématique et également l'extension « .csv » qui était récurrente à chaque fin de nom de sous-thématique. Bien entendu, lors du peuplement de la table « Sous-thématique », nous avons incrémenté un compteur lors que les sous-thématiques faisaient parti de la même thématique.

➢ Création des tables Auteur et Article

Afin de créer ces tables Auteur et Article, il a fallu dans un premier temps extraire les données nécessaires. Nous allons vous expliquer ces extractions d'une manière générale.

Premièrement nous traversons le fichier csv nettoyé ligne par ligne en réitérant les étapes suivantes : on récupère le fichier pdf qui donne le plus d'informations sur les auteurs de l'article, de même pour le fichier pmc, on compare ces nombres d'informations et nous choisissons lequel est utilisé lors du peuplement.

Voici le déroulement de la première étape : nous créons une liste de fichiers pdf initialisée à vide, puis nous nous sommes aperçus que plusieurs fichiers pdf pouvaient être référencés pour un même article, il était donc important de connaître le fichier qui nous donnait le plus d'informations, c'est donc pour cela que nous avons « split » la valeur de la cellule en fonction des points virgules puis, pour chaque fichier pdf, déterminer la quantité d'informations donnée par ce dernier. Nous avons ensuite sauvegardé le fichier pdf concerné dans la liste des fichiers pdf créée. Nous avons fait de même avec les fichiers pmc que nous avons sauvegardé dans la liste des fichiers pmc sauf que nous nous sommes aperçus qu'un seul fichier pmc était renseigné pour un article donc il n'y avait pas de comparaison de la quantité d'informations donnée par chacun à faire.

Le fichier pdf qui donne le plus d'informations :

```
for k in range (1,len(L)):
    if (L['pdf'][k] != ""):
        if ';' in L['pdf'][k]:
            nb_info_pdf_0 = 0
            for i in range(len(L['pdf'][k].split(';'))):
                nb_info_pdf = 0
                with open('C:/Users/Sébastien/Desktop/Projet BDDR 2/' + L['pdf'][k].split(';')[i].strip() , encoding="utf8") as file:
                    reader = json.load(file)

                    if (len(reader["metadata"]["authors"]) > 0):

                        for j in range(len(reader["metadata"]["authors"])):
                            if (reader["metadata"]["authors"][j]["first"] != ""):
                                nb_info_pdf += 1
                            if (reader["metadata"]["authors"][j]["middle"] != ""):
                                nb_info_pdf += 1
                            if (reader["metadata"]["authors"][j]["last"] != ""):
                                nb_info_pdf += 1
                            if (reader["metadata"]["authors"][j]["email"] != "" and reader["metadata"]["authors"][j]["email"] != None):
                                nb_info_pdf += 1
                            if (len(reader["metadata"]["authors"][j]["affiliation"]) >= 1 ):
                                if (reader["metadata"]["authors"][j]["affiliation"]["institution"] != ""):
                                    nb_info_pdf += 1
                                if (reader["metadata"]["authors"][j]["affiliation"]["laboratory"] != ""):
                                    nb_info_pdf += 1
                    else:
                        nb_info_pdf = 0

                if (nb_info_pdf >= nb_info_pdf_0):
                    nb_info_pdf_0 = nb_info_pdf
                    if (i == 0):
                        liste_fichiers_pdf.append('C:/Users/Sébastien/Desktop/Projet BDDR 2/' + L['pdf'][k].split(';')[i].strip())
                    elif (i >= 1):
                        liste_fichiers_pdf.pop()
                        liste_fichiers_pdf.append('C:/Users/Sébastien/Desktop/Projet BDDR 2/' + L['pdf'][k].split(';')[i].strip())

        else :
            liste_fichiers_pdf.append('C:/Users/Sébastien/Desktop/Projet BDDR 2/' + L['pdf'][k].strip())
    else :
        liste_fichiers_pdf.append('')
```

Concernant la deuxième étape, le but est ici de déterminer s'il s'agit du fichier pdf qui donne plus d'informations que le fichier pmc ou l'inverse. Donc pour un même indice de liste, nous récupérons le fichier pdf et le fichier pmc (s'ils existent), nous vérifions s'ils existent dans la base de fichiers qui nous a été transmise, nous calculons la quantité d'informations donnée par chacun puis en fonction du fichier choisi (celui qui est le plus informatif), nous ouvrons ce dernier afin de récupérer les informations souhaitées (les auteurs même s'ils sont plusieurs, leurs emails, leurs affiliations, la date de publication de l'article, le journal dans lequel il est paru, l'url, sa thématique et sa sous-thématique). Dans le cas où le fichier pdf donne la même quantité d'informations que le fichier pmc, nous avons choisi de conserver le fichier pmc. Si toutefois, la quantité d'informations des deux fichiers est nulle (c'est-à-dire que soit les fichiers existent mais ne donnent aucune information ou alors les fichiers n'existent pas) alors nous récupérons les auteurs non pas dans les fichiers mais cette fois-ci dans la colonne de notre fichier Excel correspondante, les emails et affiliations ne sont donc pas récupérables, et de même que précédemment, nous récupérons la date, le journal, l'url, la thématique et la sous-thématique dans les colonnes correspondantes.

Incrémentation d'une sous liste avec les informations :

```
elif (nb_info_pmc > nb_info_pdf):
    with open(liste_fichiers_pmc[k] , encoding="utf8") as file :
        reader = json.load(file)
        if (len(reader["metadata"]["authors"]) > 0):
            for j in range(len(reader["metadata"]["authors"])):
                liste_1_titre = []
                liste_1_titre.append(L['title'][k+1])
                liste_1_titre.append(reader["metadata"]["authors"][j]["first"])
                if (len(reader["metadata"]["authors"][j]["middle"]) >= 1):
                    liste_1_titre.append(''.join(reader["metadata"]["authors"][j]["middle"][0]))
                else:
                    liste_1_titre.append('')
                liste_1_titre.append(reader["metadata"]["authors"][j]["last"])
                if (reader["metadata"]["authors"][j]["email"] == None):
                    liste_1_titre.append('')
                else:
                    liste_1_titre.append(reader["metadata"]["authors"][j]["email"])
                if (reader["metadata"]["authors"][j]["affiliation"] != {}):
                    liste_1_titre.append(reader["metadata"]["authors"][j]["affiliation"]["institution"])
                    liste_1_titre.append(reader["metadata"]["authors"][j]["affiliation"]["laboratory"])
                else:
                    liste_1_titre.append('')
                    liste_1_titre.append('')
                liste_1_titre.append(L['date'][k+1])
                liste_1_titre.append(L['journal'][k+1])
                liste_1_titre.append(L['url'][k+1])
                if (L['thematique'][k+1]== 'Inconnu'):
                    liste_1_titre.append('')
                else:
                    liste_1_titre.append(L['thematique'][k+1])
                if (L['sous-thematique'][k+1]== 'Inconnu'):
                    liste_1_titre.append('')
                else:
                    liste_1_titre.append(L['sous-thematique'][k+1])
                liste_auteurs.append(liste_1_titre)
```

Finalement, nous avons toutes les informations nécessaires pour peupler les tables Auteur et Article. Nous commençons par créer deux tables subsidiaires Auteur2 et Article2 qui seront les tables mères des tables finales. Nous initialisons un compteur titre qui nous permettra d'avoir le même identifiant d'article pour tous les auteurs d'un même article.

Dans un premier temps, nous vérifions qu'un auteur existe pour l'article étudié, si c'est le cas et si le titre correspond au suivant (au moins deux auteurs pour cet article), alors nous insérons dans les deux tables les données voulues ainsi que le compteur et si ce n'est pas le cas nous incrémentons le compteur pour que le peuplement du prochain titre ait un identifiant d'article incrémenté d'une unité. Si on se situe à la fin de notre grande liste, nous comparons le dernier titre avec l'avant dernier au lieu de regarder le suivant qui n'existe pas. Si aucun auteur n'existe pour l'article étudié, nous réalisons tout de même un peuplement dans les deux tables en ne peuplant que le titre de l'article et son identifiant pour la table Auteur2 et les autres informations dans la table Article2.

La remarque à se faire maintenant est que même si un auteur paraît plusieurs fois car il a écrit plusieurs articles différents, son identifiant ne sera pas le même. Nous avons donc réglé ce problème en triant par nom puis prénom notre table Auteur2 et ensuite nous avons comparé s'il s'agissait du même auteur d'un article au suivant, si c'était le cas l'identifiant de l'auteur restait le même sinon le compteur auteur était incrémenté d'un à la même manière que le compteur des titres. Il nous restait plus qu'à créer la table Article en triant la table Article2 par titre, ce tri était un choix de notre part. Nous avons également supprimer les lignes avec des titres vides dans ces deux tables afin de les rendre moins volumineuses.

➢ Création des tables subsidiaires (journaux, dates, laboratoires, institutions)

La création de ces tables est due à la volonté de créer des histogrammes nous donnant le nombre de publications par journaux, laboratoires, institutions et dates. Dans cette partie, un seul exemple de peuplement de ces tables sera expliqué car la méthode est similaire pour les autres tables. Considérons le peuplement de la table Journaux.

Premièrement, nous créons un DataFrame nommé Journaux qui récupère l'entièreté de la table Article, puis comme les titres d'articles sont répétés autant de fois que le nombre d'auteurs qu'il existe pour chaque article, nous supprimons les doublons en fonction des titres afin de considérer un seul titre dans le calcul du nombre de publications par journal. Par la suite, nous créons un DataSeries qui sera composé de chaque nom de journal et du nombre de fois qu'il paraît dans la table Article et donc du nombre de publications qu'il a enregistré. Comme les DataSeries ne sont pas des données qui sont itérables (nous ne pouvions pas créer une boucle sur la longueur du DataSeries), nous le transformons en DataFrame. Finalement, nous peuplons une table Journaux précédemment créée avec un identifiant unique, le nom du journal qui sera l'index du DataFrame crée juste avant, et sa quantité de publications qui sera référencé dans la colonne journal automatiquement nommée lors de la création du DataSeries.

Peuplement de la table Journaux :

```
journaux = pd.read_sql('SELECT * FROM Article', conn)
journaux.drop_duplicates(subset=['titre'],inplace=True)

journaux = journaux['journal'].value_counts()
journaux = journaux.to_frame()

cur.execute("""DROP TABLE IF EXISTS Journaux CASCADE;
            CREATE TABLE Journaux ( Journal_id INT PRIMARY KEY, Journal VARCHAR(10485760), Quantite INT ) ;""")

for i in range(len(journaux)):
    cur.execute("""INSERT INTO Journaux
                (Journal_id, Journal, Quantite)
                VALUES
                (%s,%s,%s)
                """,
                (i+1,journaux.index[i],journaux['journal'][i]))
```

Exemple d'un DataSeries :

```
sebnoel=# select date,count(*) from Article group by date order by count desc;
    date    |  count
------------+---------
 2021       | 1124604
 2020       |  993372
 2022       |  196551
 2021-03-09 |   37680
 2015       |   27339
 2016       |   27275
 2017       |   26452
 2018       |   26121
 2019       |   24321
 2014       |   23343
 2021-03-24 |   21627
 2021-08-24 |   20976
 2013       |   19494
 2012       |   15631
 2021-05-14 |   13531
 2011       |   13094
 2021-11-17 |   12819
 2021-04-29 |   12725
```

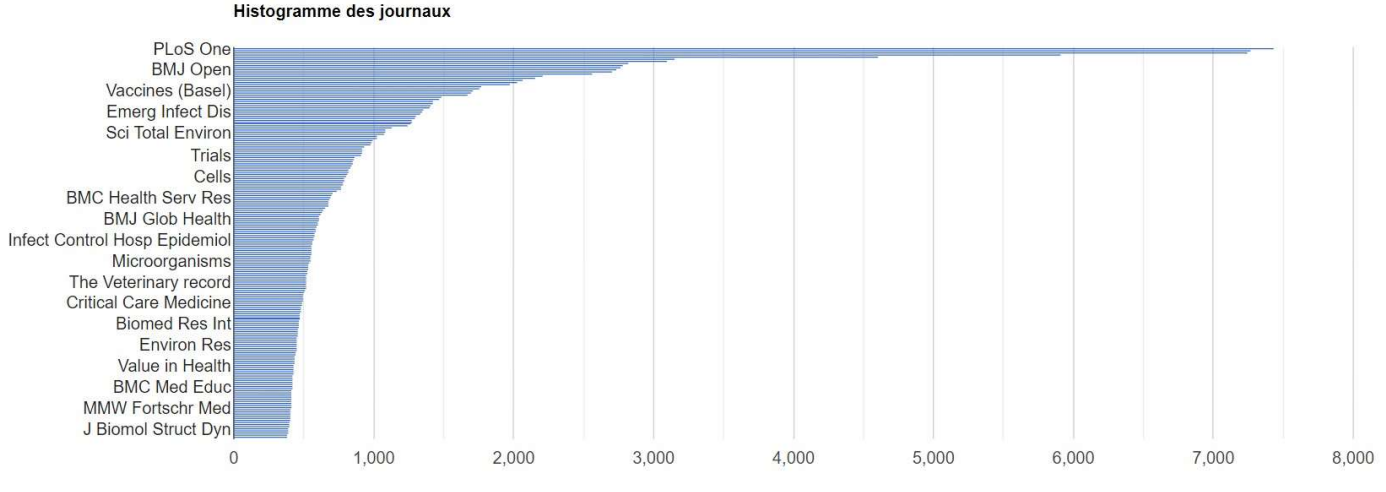
Afin de réaliser l'histogramme du nombre de publications par journaux, nous avons dû réaliser un fichier texte nommé journaux.txt sera sous la forme souhaitée, c'est-à-dire celle qui correspond à l'incrémentation des données dans la création d'un histogramme sur une page html. Nous voulions simplement avoir une succession de listes qui contiennent le nom du journal et sa quantité. Afin de rendre l'histogramme plus lisible, nous avons omis la valeur nulle (les articles qui n'ont pas de journal référencé) c'est pourquoi notre boucle commence à l'indice 1. Ensuite, nous avons rencontré un problème car certains noms de journaux était composé d'une apostrophe donc nous avons réglé cela en « splitant » le nom du journal en fonction des apostrophes et les avons remplacé par un espace.

Création du fichier texte :

```
data_journaux = pd.read_sql('SELECT * FROM Journaux', conn)

if os.path.exists("C:/Users/Sébastien/Desktop/journaux.txt"):
    os.remove("C:/Users/Sébastien/Desktop/journaux.txt")

journaux = open("C:/Users/Sébastien/Desktop/journaux.txt","a",encoding="utf-8")

for i in range(1,len(data_journaux)):
    if (i == len(data_journaux)-1):
        if ("'" in data_journaux['journal'][i]):
            nouveau_texte = ""
            for j in range(len(data_journaux['journal'][i].split("'"))):
                nouveau_texte = nouveau_texte + data_journaux['journal'][i].split("'")[j] + " "
            journaux.write(f"['nouveau_texte',{data_journaux['quantite'][i]}]")
        else:
            journaux.write(f"['{data_journaux['journal'][i]}',{data_journaux['quantite'][i]}]")
    else:
        if ("'" in data_journaux['journal'][i]):
            nouveau_texte = ""
            for j in range(len(data_journaux['journal'][i].split("'"))):
                nouveau_texte = nouveau_texte + data_journaux['journal'][i].split("'")[j] + " "
            journaux.write(f"['nouveau_texte',{data_journaux['quantite'][i]}],")
            journaux.write(f"\n")
        else:
            journaux.write(f"['{data_journaux['journal'][i]}',{data_journaux['quantite'][i]}],")
            journaux.write(f"\n")

journaux.close()
```

Lorsque notre fichier texte était prêt, il nous suffisait de réaliser un copier-coller de ce dernier vers la page html correspondante. Pour cette étape, j'ai essayé d'utiliser une méthode plus subtile en indiquant dans la page html le fichier dans lequel se trouvait les données mais un histogramme vide apparaissait. Le dernier problème que j'ai rencontré était le fait que le chargement de la page web concernant l'histogramme prenait environ 20 secondes (ce qui est bien entendu long et désagréable), j'ai donc remarqué que l'histogramme ne prenait pas en considération toutes les valeurs et j'ai finalement supprimé les valeurs dans ma page html qui n'était pas affichées dans l'histogramme. Cette fois-ci le chargement de la page web était instantané. Toutes ces méthodes ont été réitérées pour les histogrammes des publications par laboratoire ou institution. Le résultat final était le suivant :

Histogramme :



- # Projet Django

Concernant le projet réalisé avec le framework Web Django et codé sur Visual Studio Code, nous avons rencontré peu de problèmes. En effet, l'expérience acquise l'année dernière m'a permis de gagner un temps précieux.

➢ Création de fichiers html

Afin de rendre une page Web lisible, la création d'un fichier html relié à une url est primordiale. Dans l'exemple ci-dessous, nous voyons ce fichier est lié à la vue « index22 » qui récupère les entrées du formulaire et les comparent aux données présentes dans la table Auteur, la méthode employée dans le formulaire est appelée.

Ensuite, pour chaque attribut pour lequel l'utilisateur peut rechercher un auteur et les articles qu'il a écrit, on crée une « division » dans laquelle nous retrouvons un champ de recherche (balise « input ») et son nom (balise « label »), l'entrée faite par l'utilisateur est stockée dans la variable « form.nom » par exemple. Nous répétons cette action pour l'ensemble des attributs proposés à la recherche. Un bouton « Rechercher » est crée afin de valider les entrées de l'utilisateur et donc passer à la recherche.

Exemple du contenu d'un fichier html gérant les entrées d'un formulaire :

```
<body>
  <h2>Recherche d'un auteur précis ou des auteurs d'un article.</h2>
  {% if rien %}
  Aucun auteur n'est enregistré
  {% else %}
  <form action = "{% url 'index22' %}", method="get", name="search_form">
    {% csrf_token %}
    <div>
      <label for="nom">Nom de l'auteur : </label>
      <input type="text" name="nom">
      {{form.nom}}
    </div>
    <div>
      <label for="prenom">Prénom de l'auteur : </label>
      <input type="text" name="prenom">
      {{form.prenom}}
    </div>
    <div>
      <label for="titre">Titre d'un article : </label>
      <input type="text" name="titre">
      {{form.titre}}
    </div>
    <h2> </h2>
    <button type="submit" value="Rechercher">Rechercher</button>
  </form>
  {% endif %}
  <h6> </h6>
  <button onclick="window.location.href = 'http://127.0.0.1:8000/accueil_auteur';">Retour à la page d'accueil des auteurs </button>
  <button onclick="window.location.href = 'http://127.0.0.1:8000';">Retour à la page d'accueil </button>
</body>
```

Prenons l'exemple de la création d'une vue dans le but de gérer les réponses données par l'utilisateur dans un formulaire permettant la recherche d'un auteur selon certains attributs. Dans un premier temps, nous récupérons les données de la table Auteur, puis en utilisant les filtres Django, nous comparons l'entrée faite par l'utilisateur (enregistrée dans la variable nom par exemple) puis dans la variable « auteur_qs » (variable qui contiendra uniquement les lignes de la table Auteur correspondants à la recherche), nous y modifions ses données afin de ne garder que les valeurs avec le même nom d'auteur par exemple. Voici un exemple de vue gérant les entrées d'un formulaire :

Exemple d'une vue :

```
def index22(request):
    all_auteur = auteur.objects.all()
    auteur_qs = all_auteur
    for i in dict(request.GET):
        if i == 'nom':
            nom = request.GET.get('nom')
            if nom != "":
                auteur_qs = auteur_qs.filter(Q(nom=nom))
        if i == 'prenom':
            prenom = request.GET.get('prenom')
            if prenom != "":
                auteur_qs = auteur_qs.filter(Q(prenom=prenom))
        if i == 'titre':
            titre = request.GET.get('titre')
            if titre != "":
                auteur_qs = auteur_qs.filter(Q(titre=titre))
    return render(request,'resultat_recherche_auteur.html',{
        'all_auteur' : all_auteur,
        'auteur_qs' : auteur_qs
    })
```

➢ Gestion des url

Pour chaque fichier html rédigé, il est nécessaire d'y associer une url. Rien de plus simple, nous créions la fin de l'url souhaitée puis nous l'associons à une vue qui elle était rattachée au fichier html correspondant. Voici un exemple :

Création d'url :

```
path('resultat_recherche_auteur',views.index22,name="index22"),
```