Projet New York Times - Bootcamp Data Engineer Février 2023

Mickael Gaspar, Can Baskurt, Clément Guiraud

Github : https://github.com/dst-nynews/dst-nynews
API New York TImes : https://developer.nytimes.com/apis
Doc de suivi projet: Suivi de projet
Documentation des fonctions: Fonctions du projet

Rappel sur les objectifs du projet et les données utilisées

Le projet que nous souhaitons mettre en place se fixe 3 objectifs métiers différents pouvant être, ou non, réalisés en fonction du temps et de notre avancement.

Nous souhaitons tout d'abord proposer un *dashboard* permettant de trier, filtrer et rechercher les éléments essentiels des articles et de la sémantique proposés par le New York Times. Il s'agira de permettre à l'utilisateur d'obtenir facilement des informations sur ce que propose le journal.

Dans un deuxième temps, il nous semblerait intéressant de mettre en relation les données sur le COVID proposées par le New York Times, et si le temps nous le permet par d'autres sources, avec les articles publiés par le New York Times. Cette comparaison pourrait permettre, notamment, d'avoir une meilleure compréhension de comment l'évolution réelle de la pandémie (évaluée par les données covid brutes) et sa retransmission dans un média grand public ont pu, ou pas, évoluer dans le temps.

Enfin, si le temps nous le permet, nous souhaiterions mettre en place une application web sous forme de mini-jeu de prédiction d'articles à succès. Il s'agira de permettre aux utilisateurs de prédire, parmi un set d'articles publiés par le New York Times, lequel sera présent dans les données "most popular" que propose le journal. La proposition de l'utilisateur pourra être comparée avec celle d'un algorithme de *Machine Learning* pour savoir si l'utilisateur est "meilleur ou moins bon qu'une intelligence artificielle".

Rappel sur le livrable 1

A la fin de notre premier livrable nous présentions les suites du projet comme suit :

La prochaine grande étape à mettre en place est le choix du type de stockage qu'il nous faudra utiliser. Ce choix est à la fois technique (BDD SQL ? NoSQL ?) et pratique (stockage en local par chaque membre du groupe ? stockage sur un VM? stockage dans le cloud ?). La solution vers laquelle nous nous orientons est celle d'un stockage en local avec l'utilisation de scripts python assurant la similarité des données exploitées chez chaque membre du groupe.

Ce deuxième livrable se concentrera sur la gestion des données et leur stockage. Il présentera tout d'abord la structure des différentes bases de données du projet , ainsi que les solutions techniques choisies pour leur implantation concrète, et dans un deuxième temps le le *workflow* mis en place du requêtage à *l'upload* dans la base de données.

Structure des bases de données et solutions techniques

Notre projet repose sur deux bases de données distinctes : une SQL, appelée "Covid", pour les données de l'archive Covid et une NoSQL, "NYTimes", pour les données des APIs du New-york Times.

Base de données SQL "Covid"

La base de données SQL : "Covid" contient des données liées au nombre de cas d'infection et de décès du Covid-19 aux Etats-Unis. Elle est composée des 4 tables suivante :

- states
- counts
- counties
- mask_use

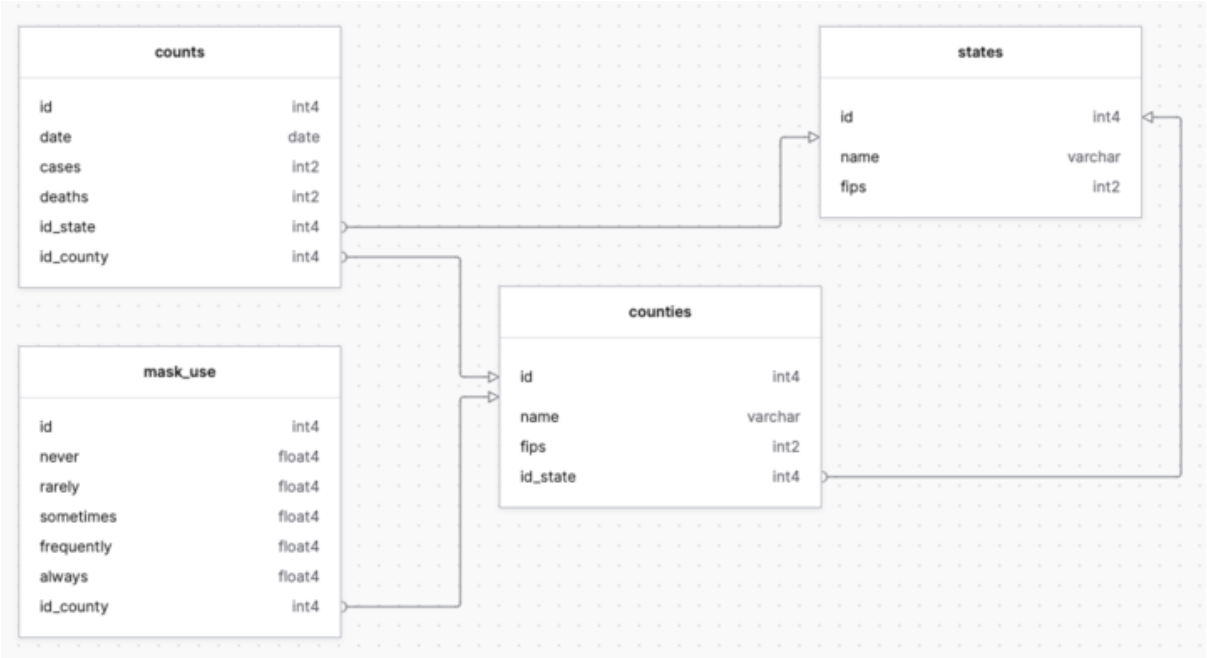

Schéma relationnel des données de la BDD Covid

La table “counts” contient pour chaque entrée : un nombre de cas d’infection, un nombre de décès, une date et un lieu (grâce à 2 relations “one-to-many” via les clés étrangères “id_state” et “id_county”). La table “states” contient le nom d’un état étasunien ainsi que des données de géolocalisation qui lui sont affiliées (“fips”). La table “counties” contient des informations similaires mais non plus au niveau des états mais des comtés (découpage administratif similaire aux circonscriptions électorales en France). Enfin, la table “mask_use” propose des données provenant de sondages sur l’utilisation des masques.

	id	date	state	county	fips	cases	deaths
1	1	2020-01-24	Illinois	Cook	17,031	1	0
2	2	2020-01-24	Washington	Snohomish	53,061	1	0
3	3	2022-05-13	Wyoming	Sweetwater	56,037	11,088	126
4	4	2022-05-13	Wyoming	Teton	56,039	10,074	16

La table “counts” de la base de données “Covid”

D’un point de vue technique, on a d’abord développé une base de données SQLite par souci de commodité. Nous allons la remplacer par une base de données PostgreSQL maintenant que le schéma relationnel des données a été fixé.

La base de données NoSQL

La base de données NoSQL “NYTimes” est composée de 4 collections distinctes :

- Articles
- Concepts
- SearchSemantic
- MostPopular

Articles

La collection “Articles” est composée des données obtenues *via* le requêtage de l’API “Article Search”. Si les *.json* renvoyés par l’API contiennent les données de 10 articles, nous avons fait le choix d’associer à chaque article un document de la base de données. Pour le moment, les articles présents dans la base de données proviennent tous d’une même requête cherchant les articles associés au mot-clé “covid” et publiés entre décembre 2019 et février 2023. Cela représente environ 1400 documents.


```
_id: "nyt://article/3ba0db54-dc3f-53a4-be05-130a74401b0b"
abstract: "Restaurateurs have become de facto public-health officials as Covid-19…"
web_url: "https://www.nytimes.com/2020/06/30/dining/restaurant-risks-coronavirus…"
snippet: "Restaurateurs have become de facto public-health officials as Covid-19…"
lead_paragraph: "Last week, a worker at one of my favorite bakeries in Los Angeles test…"
print_section: "D"
print_page: "1"
▸ headline: Object
▸ keywords: Array
pub_date: "2020-06-30T15:30:40+0000"
document_type: "article"
news_desk: "Dining"
section_name: "Food"
subsection_name: null
word_count: 1059
uri: "nyt://article/3ba0db54-dc3f-53a4-be05-130a74401b0b"
created_at: "28/03/2023 15:23:15"
```


Un document de “Articles”

Concepts

La collection “Concepts” contient les résultats de l’APIs “Semantic” lorsqu’elle est utilisée avec un concept clairement défini. Les documents de cette collection comporte notamment le nom du concept utilisé pour la requête ainsi qu’une liste de 10 articles liés au concept.

Un document de la collection “Concepts”

SearchSemantic

La collection “SearchSemantic” contient des documents provenant de requêtage de l’API Semantic du New-York Times mais, au contraire de la collection “Concepts”, lorsque celle-ci est utilisée avec des concepts flous. L’API permettant de mettre en lien un concept, considéré comme une chaîne de caractère, avec les concepts précis utilisés par le New-York Times. Ainsi, les documents contenus dans la collection Searchsemantic mettent en lien la chaîne de caractère utilisée pour faire la requête avec un concept précis du New-york Times.

```
_id: 100118242
search_name: "Coronavirus"
concept_name: "Coronavirus Reopenings"
concept_created: ""2020-05-15 09:43:34-04:00""
concept_type: "nytd_des"
created_at: "28/03/2023 17:30:23"
```

```
_id: 100145879
search_name: "Coronavirus"
concept_name: "Coronavirus Return to Office"
concept_created: ""2021-12-16 15:52:38-05:00""
concept_type: "nytd_des"
created_at: "28/03/2023 17:30:23"
```

```
_id: 100119718
search_name: "Coronavirus"
concept_name: "Coronavirus Risks and Safety Concerns"
concept_created: ""2020-06-12 13:24:29-04:00""
concept_type: "nytd_des"
created_at: "28/03/2023 17:30:23"
```


3 documents de la collection “SemanticSearch”

MostPopular

La collection MostPopular n’est pas encore peuplée. Elle a pour vocation de contenir les résultats des requêtes effectuées avec l’API MostPopular, c’est-à-dire la liste des articles les plus appréciés sur le New-York Times, les plus transmis par mail et les plus partagés sur Facebook. Cette collection sera peuplée tous les jours, via un script lancé automatiquement par Cron.

Solution Technique mise en place

La base de données NYTimes est stockée sur le Cloud grâce à l’outil MongoDB Atlas dont la solution gratuite est suffisante pour l'hébergement de nos données. Si cette solution venait à ne plus convenir durant l’avancée de notre travail, l’automatisation du peuplement de la basé, tels que nous allons maintenant le présenter, nous permettrait de reconstruire la base de données en local sans difficulté.

Workflow du requêtage au peuplement de la base de données

Le *worklow* que nous avons mis en place est similaire pour l'ensemble des APIs du New-York Times. Pour chaque API a été écrit un script python permettant de facilement récupérer les données d'intérêt. Ce script a pour sortie un fichier .json *raw*, c'est-à-dire non traité. L'ensemble des fichiers .json raw est stocké dans le dossier "data" du repo github (https://github.com/dst-nynews/dst-nynews/tree/main/data).

```python
class ApiArticleSearch:

    def __init__(self, repo_path= None):
        self.KEY_API = os.getenv("KEY_API_NYT")
        self.BASE_URI = "https://api.nytimes.com/svc/search/v2/articlesearch.json?q="
        self.PARAM = {"api-key" : self.KEY_API}
        self.repo_path = repo_path


    def get_articles(self,concept,filter="",page=0):

        """
        Se connecte à l'API "Article Search" du NYT pour la requêter
        Args :
            Concept : le concept que l'on souhaite rechercher dans le NYT

            filter : les filtres que l'on souhaite ajouter à notre requête (notamment la date de publications des articles que l'on souhaite obtenir)

            page : Permet de choisir la page de réponse que l'on souhaite (l'API ne permettant d'obtenir qu'une page de 10 résultats par requête et 100 pages au maximum). Par défautl : la première page (0)
        """

        url_api= self.BASE_URI + concept

        if filter != "":
            filter_api = f"&fq={filter}"
            page_api = "&page=" + str(page)
            url_api = url_api + filter_api + page_api
            req = requests.get(url_api,  params=self.PARAM)
            print(f'Url :' + url_api)

        else:
            req = requests.get(url_api,  params=self.PARAM)
            print(f'Url :' + url_api)

        wb = req.json()

        return wb


    def to_raw_json(self,wb,file_name,page_number):

        """
        Insère le json obtenu après la requête dans un fichier "nom_fichier_page_n".json"
        Args :
            wb : Le json obtenu avec la requête
            file_name : le nom du fichier dans lequel il sera stocké
            page_number : le n° de page qui viendra complet le nom du fichier
```

Une partie du code du script python "request_article_search.py"

Ces fichiers .json *raw* sont ensuite traités, toujours à l'aide de scripts python spécifiques pour chaque API, afin de récupérer seulement les données d'intérêts présentées dans le livrable 1 et de les écrire dans un fichier *.json cleaned*.

```python
def file_name(self,file_path):
    name = os.path.basename(file_path)
    return name

def json_clean(self,file_json):
    json_first_step = file_json["response"]["docs"]
    if json_first_step == []:
        return print("Fichier vide")
    else :
        list_articles_cleaned = []
        for i in json_first_step:
            article={}
            article["abstract"] = i.get("abstract")
            article["web_url"] = i.get("web_url")
            article["snippet"] = i.get("snippet")
            article["lead_paragraph"] = i.get("lead_paragraph")
            article["print_section"] = i.get("print_section")
            article["print_page"] = i.get("print_page")
            article["headline"] = i.get("headline")
            article["keywords"] = i.get("keywords")
            article["pub_date"] = i.get("pub_date")
            article["document_type"] = i.get("document_type")
            article["news_desk"] = i.get("news_desk")
            article["section_name"] =i.get("section_name")
            article["subsection_name"] = i.get("subsection_name")
            article["_id"] = i.get("_id")
            article["word_count"] = i.get("word_count")
            article["uri"] = i.get("uri")

            list_articles_cleaned.append(article)
```

Une partie du script “data_cleaning.py” servant aux données de Article Search

Enfin, un dernier script python, cette fois commun à l’ensemble des API, permet de peupler la base de données MongoDB Atlas à partir des données présentent dans les fichiers *cleaned*.
L’ensemble de ces scripts ont été écrits et pensés en Programmation Orientée Objet afin de faciliter l’automatisation du requêtage et du peuplement de la BDD.

```python
class BddSemantic:
    def __init__(self, repo_path_clean: Optional[str] = None) -> None:
        """Instanciate a connection to fetch data from an API of the NY Times.
        Args:
            repo_path (Optional[str], optional): Path to clean storage directory.
        """
        self.repo_path_clean = repo_path_clean
        self.MongoHost = os.getenv("MONGO_CLIENT_HOST")
        self.Client = MongoClient(host=self.MongoHost, port=27017)

    def import_json(self, file_name):
            with open(self.repo_path_clean+file_name, "r") as file:
                file_json = json.load(file)
            return file_json


    def insert_mongoDB(self, json_cleaned, collection):
            to_stock = self.import_json(json_cleaned)
            if to_stock != None:
                if isinstance(to_stock, list):
                    for document in to_stock:
                        document["created_at"] = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
                        collection.replace_one({"_id": document["_id"]}, document, upsert = True)
                else:
                    #collection.insert_one(to_stock)
                    to_stock["created_at"] = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
                    collection.replace_one({"_id": to_stock["_id"]}, to_stock, upsert = True)
```

Script de peuplement de la BBD Atlas NYTimes

Perspectives

L’étape suivante de notre projet est la digestion des données, c’est-à-dire la valorisation des informations stockées dans nos bases de données. Les objectifs présentés plus haut dans ce livrable restent d’actualité mais il nous faut néanmoins les préciser. Pour cela, nous proposons 5 cas métiers différents, même si liés entre eux.

Cas métier n°1

Objectif métier : voir si le nombre d’articles et le nombre de cas sont corrélés et comment cette corrélation évolue dans le temps

Solution technique : Dashboard représentant le nombre de cas de covid et le nombre de morts comparé au nombre d’articles, de mots par article avec le Keyword “Covid” dans le temps (et dans l’espace à l’aide des données de localisation).

Cas métier n°2

Objectif métier : permettre à un utilisateur de trouver des articles liés à un mot-clé et savoir s’ils ce sont des articles populaires

Solution technique :

1. L'utilisateur cherche un concept n’appartenant pas à la liste officielle “sémantique” du New-York Times.
2. A partir de la recherche est proposée une liste de concepts appartenant aux concepts officiels du New-York Times (c’est ce que renvoie l’API Semantic).
3. Dashboard présentant les informations autour du concept ainsi qu’une liste d'articles liés au concept (ce que renvoie l’API Semantic avec un concept officiel du NYT). Précise si les articles sont “most popular”)
4. Renvoie l’URL de l’article pour qu’il puisse être consulté

Cas métier n°3

Objectif métier : afficher les informations des articles les plus populaires

Solution technique :

1. A partir de la recherche des most popular, on obtient la liste des 20 articles les most viewed, searched, emailed
2. En cliquant sur l’article on obtient les informations sur cet article

Cas métier n°4

Objectif métier : afficher les Keywords les plus associés à un concept sur une période de temps donnée

Solution technique :

1. L'utilisateur propose un concept
2. En retour il obtient quels sont les mots clés associés (information données notamment par l'API article search) à ce mot clé

Cas métier n°5

Objectif métier : Regarder l'évolution des mots associés à un keyword

Solution technique :

1. L'utilisateur propose un keyword
2. Analyse des abstracts des articles liés à ce keyword (fournis par l'API article search)
3. Retourne les mots les plus utilisés sur une période donnée