IUT de Paris - Rives de Seine
Université Paris Cité

RAPPORT - SAE COLLECTE DE DONNÉES WEB

10/03/2024

Gomis Vincent
Azek Mohamed

SOMMAIRE

INTRODUCTION

L'essence de ce projet résidait dans l'amélioration de la base de données SIRENE, en commençant par le fichier StockEtablissement. Bien que ce fichier soit déjà riche en informations relatives aux établissements, il nécessite l'ajout de données complémentaires. Notre ambition était d'enrichir ces informations avec les coordonnées géographiques et des détails additionnels obtenus via une recherche sur Google Maps.

Pour réaliser ce projet, nous avons adopté une approche structurée basée sur un pseudo-algorithme pour la collecte de données. Dans ce contexte, N représente le nombre total de lignes dans le fichier et P est le nombre de lignes à lire et à traiter à chaque itération, fixé ici à 100 lignes pour l'exemple.
Notre stratégie consiste à traiter le fichier par segments de 100 lignes à la fois. Pour chaque segment, l'algorithme est conçu pour extraire le numéro SIRET de chaque établissement listé, utiliser ce numéro pour obtenir les coordonnées géographiques via l'API Adresse, puis collecter des données supplémentaires via une recherche sur Google Maps.

Nous avons utilisé une combinaison d'outils de programmation et de bibliothèques. Par exemple, nous avons utilisé des bibliothèques Python comme pandas pour la manipulation des données et requests pour faire des appels API. De plus, nous avons utilisé des boucles et des fonctions pour répéter le processus sur chaque segment de données. Cela nous a permis de traiter efficacement de grandes quantités de données de manière automatisée.

Les informations recueillies sont ensuite intégrées au DataFrame initial, enrichissant ainsi la base de données avec des informations et des sites web qui peuvent être associés à chaque SIREN. Ce processus est répété de manière itérative, traitant chaque segment de 100 lignes, jusqu'à ce que l'ensemble du fichier soit parcouru et que la base de données SIRENE soit entièrement mise à jour.

DEMARCHE

Preparation de la base de données:

La base d'un projet DATA commence par préparer et nettoyer la ou les bases de données nécessaires à notre étude. Nous allons voir les différentes étapes de celle-ci.

- Importation du fichier CSV :

Le fichier CSV StockEtablissement_utf8_1000.csv a été importé grâce à la librairie Pandas (import pandas as pd)) avc la commande "pd.read_csv()". Nous avons décidé de traiter l'ensemble des 1000 lignes d'un coup d'où le fait qu'on ait pas utilisé la commande "nrwos". Mais avant cela sur un code d'essai nous avons bien sûr essayer le code par tranche de 10. Nous avons nommé notre fichier Sirene comme dans l'énoncé de la SAE. Nous avons ensuite gardé sur notre ficher uniquement les colonnes utiles pour notre projet afin de rendre le fichier plus lisible. Nous avons donc garder le SIRET et SIREN, le nom des villes, code postal et l'adresses, logiquement nécessaires afin d'effectuer le géocodage.

- Nettoyage des données "contraignantes":

Les champs de données, comme le numéro de voie, ont été purifiés pour prévenir les erreurs. Les valeurs non numériques ont été gérées avec pd.to_numeric(), en utilisant errors='coerce' pour transformer les valeurs non convertibles en NaN, qui ont ensuite été remplacées par zéro. Cette étape était cruciale pour éviter les erreurs lors des opérations ultérieures qui requièrent des formats numériques.

DEMARCHE

Nous avons aussi changé les abréviations qu'on pouvait rencontrer par exemple avec "PL", "RTE" et "AV" par "Place", "Route" et "Avenue". Cette étape avait pour but de faciliter le géocodage des adresse.

Collecte des coordonnées géographiques:

Pour la collecte de données via URL pour les requêtes API, nous avons utilisé OPENCAGEDATA pour géocoder et pas API.gouv. Nous nous sommes aperçu d'une erreur avec api.gouv qui nous empechait de collecter un grand nombres de données d'un coup. En changeant d'API cela ne marchait toujours pas, on s'est donc rendu compte que c'était un problème de PC. En effet, nous avons travaillé sur nos PC d'entreprises qui sont soumis à pleins de blocages. Nous aurions pu donc revenir sur api.gouv mais nous avons décider de continuer sur OPENCAGEDATA.

ETAPE DE LA CREATION D'URL

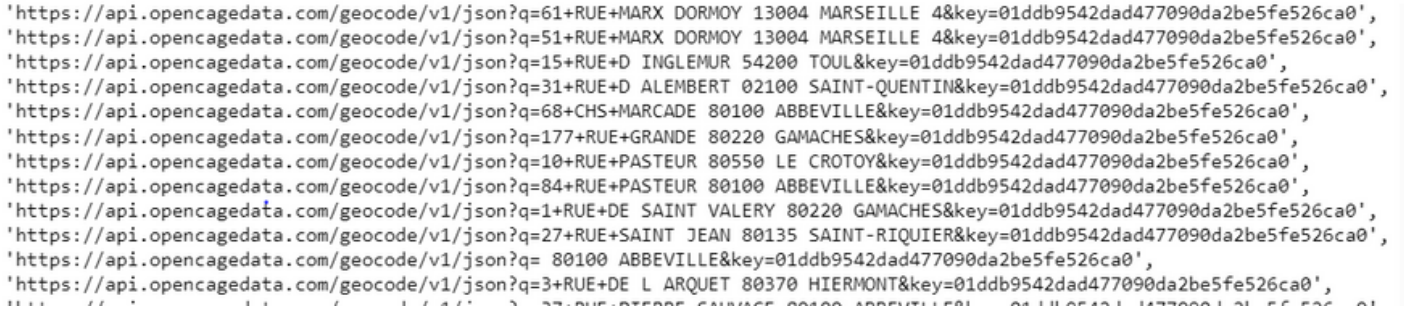
1. INITIALISATION : Pour générer l'URL de chaque adresse, (INITIALISATION) on a créer une liste vide adresse_urls est créée pour stocker les URLs générés. Une clé d'API api_key est définie pour l'API OpenCage (la clé est distincte pour chaque compte). Une liste vide adresse_urls est créée pour stocker les URLs générés. Une clé d'API api_key est définie pour l'API OpenCage.
2. Boucle sur le DataFrame : Le code itère sur chaque ligne du DataFrame sirene en utilisant une boucle for.
3. Vérification de l'adresse : Pour chaque ligne, il vérifie si la valeur de la colonne "Adresse" est "null". Si c'est le cas, il ajoute simplement "null" à la liste adresse_urls.
4. Extraction des valeurs des colonnes : Si l'adresse n'est pas "null", il extrait les valeurs des colonnes "Adresse", "code Postal Etablissement" et "libelle Commune Etablissement", les convertit en chaînes de caractères et les stocke dans les variables adresse, code_postal et ville respectivement. Si une valeur est manquante (NaN), elle est remplacée par une chaîne vide.

DEMARCHE

En utilisant ces 3 colonnes, on permet à l'API d'être très précise, ainsi l'API ne peut pas se tromper et propose à chaque fois une adresse unique qui est forcément l'adresse exacte.

5- Construction de l'URL : notre programme construit ensuite l'URL pour l'API OpenCage en utilisant ces valeurs et la clé d'API, et ajoute l'URL à la liste adresse_urls.

6 - Résultat : À la fin de la boucle, adresse_urls contient tous les URLs générés pour chaque établissement dans le DataFrame **sirene** comme on peut voir ci-dessous



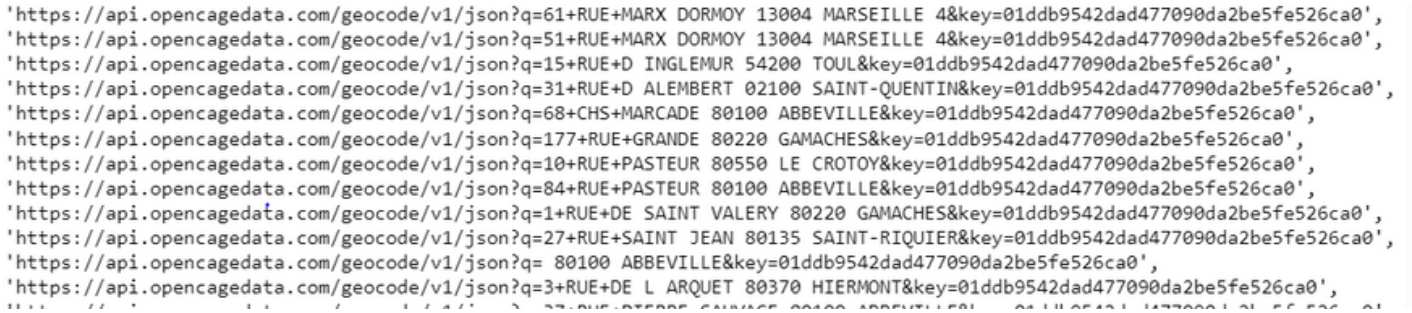
```
'https://api.opencagedata.com/geocode/v1/json?q=61+RUE+MARX DORMOY 13004 MARSEILLE 4&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=51+RUE+MARX DORMOY 13004 MARSEILLE 4&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=15+RUE+D INGLEMUR 54200 TOUL&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=31+RUE+D ALEMBERT 02100 SAINT-QUENTIN&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=68+CHS+MARCADE 80100 ABBEVILLE&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=177+RUE+GRANDE 80220 GAMACHES&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=10+RUE+PASTEUR 80550 LE CROTOY&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=84+RUE+PASTEUR 80100 ABBEVILLE&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=1+RUE+DE SAINT VALERY 80220 GAMACHES&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=27+RUE+SAINT JEAN 80135 SAINT-RIQUIER&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q= 80100 ABBEVILLE&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=3+RUE+DE L ARQUET 80370 HIERMONT&key=01ddb9542dad477090da2be5fe526ca0',
```

Une fois la liste d'URL créée, on utilise cette dernière dans un programme permettant de récupérer les coordonnées géographiques des adresses et en même temps de les stocker dans les colonnes "Latitude adresse" et "Longitude adresse" qu'on vient de creer.

DEMARCHE

En utilisant ces 3 colonnes, on permet à l'API d'être très précise, ainsi l'API ne peut pas se tromper et propose à chaque fois une adresse unique qui est forcément l'adresse exacte.

5- Construction de l'URL : notre programme construit ensuite l'URL pour l'API OpenCage en utilisant ces valeurs et la clé d'API, et ajoute l'URL à la liste adresse_urls.
6 - Résultat : À la fin de la boucle, adresse_urls contient tous les URLs générés pour chaque établissement dans le DataFrame **sirene** comme on peut voir ci-dessous



```
'https://api.opencagedata.com/geocode/v1/json?q=61+RUE+MARX DORMOY 13004 MARSEILLE 4&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=51+RUE+MARX DORMOY 13004 MARSEILLE 4&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=15+RUE+D INGLEMUR 54200 TOUL&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=31+RUE+D ALEMBERT 02100 SAINT-QUENTIN&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=68+CHS+MARCADE 80100 ABBEVILLE&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=177+RUE+GRANDE 80220 GAMACHES&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=10+RUE+PASTEUR 80550 LE CROTOY&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=84+RUE+PASTEUR 80100 ABBEVILLE&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=1+RUE+DE SAINT VALERY 80220 GAMACHES&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=27+RUE+SAINT JEAN 80135 SAINT-RIQUIER&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q= 80100 ABBEVILLE&key=01ddb9542dad477090da2be5fe526ca0',
'https://api.opencagedata.com/geocode/v1/json?q=3+RUE+DE L ARQUET 80370 HIERMONT&key=01ddb9542dad477090da2be5fe526ca0',
```

Une fois la liste d'URL créée, on utilise cette dernière dans un programme permettant de récupérer les coordonnées géographiques des adresses et en même temps de les stocker dans les colonnes "Latitude adresse" et "Longitude adresse" qu'on vient de creer.

DEMARCHE

```python
for i in range(len(sirene)):
    # Vérifier si l'adresse est "null"
    if sirene.iloc[i]["Adresse"] == "null":
        continue

    # Construire l'adresse
    adresse = str(sirene.iloc[i]["Adresse"]) + " " + str(sirene.iloc[i]["code Postal Etablissement"]) + " " + str(sirene.iloc[i]|

    # Construire l'URL pour l'API OpenCage
    url = f"https://api.opencagedata.com/geocode/v1/json?q={adresse}&key={api_key}"

    try:
        # Envoyer une requête GET à l'URL
        response = requests.get(url)
        time.sleep(1)  # Pause de 1 seconde

        # Vérifier si la requête a réussi
        if response.status_code == 200:
            # Extraire les coordonnées de la réponse
            data = response.json()
            if data['results']:
                location = data['results'][0]['geometry']
                sirene.at[i, 'Latitude adresse'] = location['lat']
                sirene.at[i, 'Longitude adresse'] = location['lng']
            else:
                print(f"Pas de résultats pour l'adresse : {adresse}")
        else:
            print(f"Erreur de requête, code d'état : {response.status_code}")
    except Exception as e:
        print(f"Erreur de réseau : {e}")
```

On peut voir ici que pour qu'on utilise un time sleep pour éviter à l'API de faire trop de requêtes à la fois.

Utilisation de chromedriver pour l'extraction des données supplémentaires:

L'objectif de cette étape était d'employer chromedriver en conjonction avec Selenium pour explorer Google Maps et recueillir des informations supplémentaires sur les établissements, y compris les informations et les sites web associés à chaque siret. Cette phase requérait une interaction automatisée avec un navigateur web afin d'accéder à des données qui ne sont pas immédiatement accessibles via une API ou un fichier de données.

Pour cela, on a d'abord installer Selenium sur Jupyter Notebook.

Pour chaque établissement répertorié dans le DataFrame sirene, une URL Google Maps spécifique était générée à partir des données d'adresse. Cette URL servait à diriger le navigateur automatisé vers la page Google Maps correspondant à l'établissement.

DEMARCHE

Voici les étapes clés :

- Construction des URL de recherche : Sur la base des adresses extraits précédemment, des URL de recherche Google Maps étaient créées pour pointer vers l'emplacement spécifique de chaque établissement.
- Navigation et chargement de la page : Chromedriver était utilisé pour ouvrir chaque URL et attendre que la page soit entièrement chargée.
- Extraction des données : Une fois la page chargée, le code HTML de la page était récupéré et des sélecteurs étaient utilisés pour extraire les informations et les URLs des sites web des établissements. Les données extraites étaient stockées dans des listes correspondantes pour une utilisation ultérieure.
- Gestion des exceptions : Des codes de gestion des erreurs étaient en place pour gérer les cas où les informations n'étaient pas disponibles ou lorsque la page ne répondait pas aux attentes, en insérant des valeurs null pour maintenir l'intégrité de la structure des données.
- Intégration des données extraites : Les informations collectées via chromedriver étaient ensuite intégrées au DataFrame sirene. Chaque établissement dans le DataFrame se voyait attribuer ses coordonnées géographiques, un label détaillé de l'adresse, ainsi que les liens vers les sites web et autres informations pertinentes récupérées.

Nous avons essayer d'enrichir notre DataFrame sirene avec des informations supplémentaires (coordonnées géographiques, détails de localisation, et données extraites via chromedriver) puis l'avons sauvegardé en csv. Nous avons utilisé la fonction to_csv de Pandas pour écrire le DataFrame finalisé dans un fichier CSV nommé sirene.csv.

DIFFICULTES RENCONTREES

Pour ce projet, nous avons rencontré pas mal de difficultés, notamment au niveau de Chrome driver. Nous n'avons donc pas réussi a le mettre en marche dans notre code. Il y'avais un problème de version entre chrome et chromeDriver ce qui nous as fait pataugé grandement. Mais notre première et plus grande difficulté a été la collecte des Latitudes et Longitudes avec les problèmes d'accès de nos PC d'entreprise. Nous profitions donc des temps de projet pour nous retrouver à l'IUT et continuer le projet. La structuration du code a aussi été une difficulté. Il nous fallait un plan pour que notre code soit lisible et bien organisé pour être compris par quelqu'un d'autres que nous.

CONCLUSION

Cette SAE a été une expérience riche en apprentissages. Nous avons dû naviguer entre la gestion d'un fichier volumineux, l'acquisition de données géographiques et d'autres informations via Google Maps, tout en surmontant divers défis techniques.
beaucoup de recherches ont été nécessaires à ce travail qui se rapprochait beaucoup de nos missions en entreprise.

Pour le fichier StockEtablissement, nous avons commencé par traiter les données par lots de 50. Une fois que nous avons constaté que cette méthode fonctionnait bien, nous avons augmenté la taille des lots pour traiter 1000 lignes à la fois, comme nous l'avons fait pour le fichier de 6 Go.
Ce projet visant à enrichir la base de données SIRENE a été une expérience riche en apprentissages. Nous avons dû jongler entre la lecture d'un fichier volumineux, la collecte de données géographiques et d'autres détails via Google Maps, et la résolution de divers problèmes techniques.
En suivant notre plan pas à pas, nous avons pu progresser de manière méthodique. Cela a impliqué de lire le fichier StockEtablissement par petits morceaux et d'utiliser ces informations pour rechercher des données géographiques et d'autres détails en ligne. Bien que le processus ait été laborieux, il nous a permis de construire progressivement notre base de données enrichie.

Nous avons rencontré de nombreux défis, notamment la programmation des boucles appropriées pour traiter les données et la gestion des réponses aux requêtes web. La partie la plus difficile a été d'utiliser Chromedriver pour récupérer des informations sur Google Maps. Nous avons souvent obtenu des résultats nuls, ce qui nous a fait tourner en rond (et nous n'avons toujours pas réussi à récupérer ces informations). La taille du fichier de 6,84 Go a également posé problème, nous obligeant à réfléchir à des stratégies pour travailler avec de telles quantités de données sans faire planter notre système.
En résumé, malgré les obstacles, ce projet a été très instructif. Nous avons appris beaucoup de choses sur la gestion des données, le codage pour la collecte d'informations en ligne, et comment s'adapter lorsque les choses ne se passent pas comme prévu (sauf pour la partie Chromedriver, bien sûr).