Mélanie Donne

Rapport Etape 1

Analyse de l'immobilier à Paris

Mélanie Donne

Introduction

Ce rapport détaille les progrès réalisés dans le cadre du projet de l’analyse de l'immobilier à Paris.

L'objectif principal est de collecter des données brutes à partir de différentes sources d'API, puis de les nettoyer, de les prétraiter et de les préparer pour une analyse ultérieure.

Dans cette première étape, nous avons concentré nos efforts sur trois principaux aspects : la préparation des données foncières, l'extraction des données via des requêtes API et le nettoyage des résultats obtenus. À cette fin, nous avons développé trois scripts Python distincts : data-cleaning.py pour les données foncières, api-requests.py pour l'extraction des données et api-cleaning-results.py pour leur nettoyage et leur préparation.

Dans ce rapport, nous détaillerons les différentes étapes impliquées dans le processus de nettoyage et de prétraitement des données, en fournissant des explications détaillées sur le fonctionnement des scripts et en illustrant chaque étape avec des exemples concrets.

Mélanie Donne

1. Le nettoyage des données

Pour commencer, j'ai codé le script `data-cleaning.py` qui effectue plusieurs opérations de nettoyage et de prétraitement sur un jeu de données relatives aux valeurs foncières de l'année 2022.

Voici les différentes étapes réalisées par ce script :

Chargement des Données :

Le script utilise la librairie Pandas pour charger les données du fichier valeursfoncieres-2022.txt dans un DataFrame, en spécifiant que les colonnes sont séparées par le caractère "|".

```
# Chargement des Données
vf2022 = pd.read_csv("./valeursfoncieres-2022.txt", sep="|", low_memory=False)
```

Nettoyage des Colonnes avec des Données Manquantes :

Les colonnes contenant des valeurs manquantes sont identifiées, puis celles dont le pourcentage de valeurs manquantes dépasse 50% sont supprimées du DataFrame. Cette opération vise à éliminer les colonnes avec des données insuffisantes pour une analyse significative.

```
# Nettoyage des Colonnes avec des Données Manquantes
columns_with_nan = vf2022.columns[vf2022.isna().any()]
for column in columns_with_nan:
        percentage_missing = vf2022[column].isna().sum() * 100.0 / vf2022.shape[0]
    if percentage_missing > 50:
        vf2022.drop(column, axis=1, inplace=True)
```

Suppression des Colonnes Inutiles :

Certaines colonnes jugées inutiles pour l'analyse sont supprimées du DataFrame. Ces colonnes comprennent des informations telles que la section, le numéro de plan, le nombre de lots, etc.

```
# Suppression des Colonnes Inutiles
useless_columns = ['Section', 'No plan', 'Nombre de lots', 'Code type local', 'Nature culture', 'No voie', 'No disposition']
vf2022.drop(useless_columns, axis=1, inplace=True)
```

Nettoyage des Lignes avec des Données Manquantes :

Les lignes contenant au moins 14 valeurs manquantes sont supprimées du DataFrame. Cette opération vise à éliminer les lignes ayant un nombre significatif de valeurs manquantes.

```
# Nettoyage des Lignes avec des Données Manquantes
vf2022 = vf2022.dropna(axis=0, thresh=14)
```

Conversion du Format de Données :

La colonne "Valeur fonciere" est nettoyée en remplaçant les virgules par des points pour assurer un format numérique approprié.

```
# Conversion du Format de Données
vf2022["Valeur fonciere"] = vf2022["Valeur fonciere"].str.replace(',', '.')
```

Conversion des Types de Données :

Certaines colonnes sont converties dans des types de données appropriés. Par exemple, la colonne "Valeur fonciere" est convertie en type numérique, la colonne "Surface reelle bati" est également convertie en type numérique, et la colonne "Code postal" est convertie en type entier (int32). De plus, la colonne "Date mutation" est convertie en type de données datetime.

```
# Conversion des Types de Données
vf2022["Valeur fonciere"] = pd.to_numeric(vf2022["Valeur fonciere"])
vf2022["Surface reelle bati"] = pd.to_numeric(vf2022["Surface reelle bati"])
vf2022["Code postal"] = vf2022["Code postal"].astype({'Code postal':
'int32'})
vf2022["Date mutation"] = pd.to_datetime(vf2022["Date mutation"])
```

Filtrage des Données pour Paris :

Les données sont filtrées pour inclure uniquement les codes postaux de Paris, définis dans la liste paris_postal_codes.

```
# Filtrage des Données pour Paris
paris_postal_codes = [75001, 75002, 75003, 75004, 75005, 75006, 75007,
75008, 75009, 75010,
                      75011, 75012, 75013, 75014, 75015, 75016, 75017,
75018, 75019, 75020]
vf_paris2022 = vf2022[vf2022['Code postal'].isin(paris_postal_codes)]
```

Affichage des données de Paris :

Les cinq premières lignes des données de Paris sont affichées pour inspection.

```
# Affichage des Données de Paris
print(vf_paris2022.head())
```

Enregistrement des Données dans un Fichier CSV :

Les données nettoyées et filtrées sont enregistrées dans un fichier CSV nommé "valeursfoncieres_paris_2022.csv", avec l'index des lignes omis et le format numérique des valeurs défini pour une précision décimale de 1, et en utilisant une virgule comme séparateur de champ et un retour à la ligne comme terminateur de ligne.

```
# Enregistrer les données organisées dans un fichier CSV
vf_paris2022.to_csv("valeursfoncieres_paris_2022.csv",       index=False,
float_format='%.1f', sep=',', line_terminator='\n')
```

Ce script permet donc de nettoyer et de préparer les données relatives aux valeurs foncières de l'année 2022 pour une analyse ultérieure, en particulier en se concentrant sur les propriétés localisées à Paris.

Mélanie Donne

2. La récupération et la sauvegarde de données

Dans le cadre de l'analyse de l'immobilier à Paris, nous avons exploité plusieurs sources de données disponibles, notamment les données gouvernementales et municipales, ainsi que des API fournissant des informations pertinentes sur les biens immobiliers et leur environnement.

Nous commençons avec api_request.py qui permet de récupérer des données à partir de plusieurs API en utilisant des requêtes HTTP, puis de sauvegarder ces données dans des fichiers locaux au format JSON.

Voici les différentes étapes réalisées par ce script :

Définition de la fonction fetch_and_save_data :

Cette fonction est définie pour récupérer des données à partir d'une API et les sauvegarder dans un fichier JSON. Elle prend deux arguments en entrée : l'URL de l'API (api_url) et le chemin vers le fichier de sortie (output_file).
À l'aide de la bibliothèque requests, elle envoie une requête GET à l'URL spécifiée.
Si la requête aboutit (code de statut HTTP 200), elle convertit la réponse en format JSON et la sauvegarde dans le fichier spécifié et si la requête échoue (code de statut HTTP différent de 200), un message d'erreur est affiché.

```
def fetch_and_save_data(api_url, output_file):
    response = requests.get(api_url)
    if response.status_code == 200:
        data = response.json()  # Convertir la réponse en JSON

        # Créer le dossier raw_api_data dans le répertoire fetching_api_data
s'il n'existe pas
                    raw_data_dir   =   os.path.join(os.path.dirname(__file__),
'raw_api_data')
        if not os.path.exists(raw_data_dir):
            os.makedirs(raw_data_dir)

        with open(output_file, 'w') as json_file:
             json.dump(data, json_file)  # Enregistrer les données JSON dans
un fichier
         print(f"Les données ont été récupérées avec succès depuis {api_url}
et enregistrées dans {output_file}.")
    else:
        print(f"Erreur lors de la récupération des données depuis {api_url}.
Statut de la requête : {response.status_code}")
```

Mélanie Donne

Définition de la liste api_urls :

Cette liste contient les URL des différentes API à interroger ainsi que les chemins des fichiers de sortie dans lesquels sauvegarder les données récupérées.
Chaque élément de la liste est un tuple contenant l'URL de l'API et le chemin du fichier de sortie.

```
# Liste des API à interroger avec les URLs correspondantes et les noms des fichiers de sortie
api_urls = [
("https://opendata.paris.fr/api/explore/v2.1/catalog/datasets/plu-espaces-libres-a-vegetaliser-elv/records",
"fetching_api_data/raw_api_data/plu-espaces-libres-a-vegetaliser-elv.json"),
("https://opendata.paris.fr/api/explore/v2.1/catalog/datasets/plu-secteurs-de-risques-delimites-par-le-ppri/records",
"fetching_api_data/raw_api_data/plu-secteurs-de-risques-delimites-par-le-ppri.json"),
("https://opendata.paris.fr/api/explore/v2.1/catalog/datasets/arrondissements/records",          "fetching_api_data/raw_api_data/arrondissements.json"),
("https://parisdata.opendatasoft.com/api/explore/v2.1/catalog/datasets/quartier_paris/records",
"fetching_api_data/raw_api_data/quartiers-administratifs.json"),
("https://opendata.paris.fr/api/explore/v2.1/catalog/datasets/logement-encadrement-des-loyers/records","fetching_api_data/raw_api_data/logement-encadrement-des-loyers.json"),
("https://opendata.paris.fr/api/explore/v2.1/catalog/datasets/espaces_verts/records","fetching_api_data/raw_api_data/espaces-verts-et-assimiles.json")
]
```

Boucle de traitement des API :

Une boucle for est utilisée pour parcourir chaque tuple dans la liste api_urls. Pour chaque tuple, la fonction fetch_and_save_data est appelée avec l'URL de l'API et le chemin du fichier de sortie comme arguments. Ainsi, chaque API est interrogée et ses données sont sauvegardées dans un fichier JSON correspondant.

```
# Parcourir la liste des API et exécuter la fonction fetch_and_save_data pour chaque API
for api_url, output_file in api_urls:
    fetch_and_save_data(api_url, output_file)
```

Après avoir utilisé `api-requests.py` pour récupérer les données à partir des différentes API, nous avons besoin de `api-cleaning-results.py` pour effectuer plusieurs opérations de nettoyage et de prétraitement sur ces données. Pour faire simple, `api-cleaning-results.py` est nécessaire pour transformer les données brutes extraites des API en un format propre, structuré et prêt à être utilisé pour des analyses ultérieures. Cela permet d'obtenir des résultats plus précis et significatifs à partir des données.

Mélanie Donne

Voici les étapes :

Définition de la fonction format_data :

Cette fonction prend plusieurs arguments en entrée : le chemin du fichier d'entrée au format JSON (input_file), le chemin du fichier de sortie (output_file), un dictionnaire de mapping pour renommer les clés des données (keys_mapping), et un argument optionnel pour spécifier le répertoire de sortie (output_directory).
Elle charge les données depuis le fichier d'entrée au format JSON. Ensuite, elle parcourt chaque entrée dans les données et crée une nouvelle entrée formatée en utilisant le mapping des clés fourni.
Les données formatées sont stockées dans une liste. Si le répertoire de sortie spécifié n'existe pas, il est créé. Enfin, les données formatées sont écrites dans un nouveau fichier JSON dans le répertoire de sortie.

```
def format_data(input_file, output_file, keys_mapping, output_directory="fetching_api_data/formatted_json"):
    # Charger les données depuis le fichier d'entrée au format JSON
    with open(input_file, 'r', encoding='utf-8') as file:
        data = json.load(file)

    # Initialiser une liste pour stocker les données formatées
    formatted_data = []

    # Parcourir chaque entrée dans les données
    for entry in data.get('results', []):
        # Créer une entrée formatée en utilisant le mapping des clés
            formatted_entry = {keys_mapping[key]: entry[key] for key in keys_mapping if key in entry}
        formatted_data.append(formatted_entry)

    # Créer le répertoire de sortie s'il n'existe pas déjà
    os.makedirs(output_directory, exist_ok=True)

    # Écrire les données formatées dans un nouveau fichier JSON
            with open(os.path.join(output_directory, output_file), 'w', encoding='utf-8') as outfile:
        json.dump(formatted_data, outfile, indent=2)
```

Formatage des données pour chaque API :

Pour chaque jeu de données extrait à partir d'une API, la fonction format_data est appelée avec les arguments appropriés.
Chaque appel spécifie le chemin du fichier d'entrée JSON brut, le nom du fichier de sortie formaté, et un mapping des clés pour renommer les attributs selon des noms plus significatifs. Les données sont ainsi nettoyées et préparées pour une analyse ultérieure.

```
# Formatage des données pour les espaces libres à végétaliser
format_data('fetching_api_data/raw_api_data/plu-espaces-libres-a-vegetaliser
```

```
-elv.json',
            'plu-espaces-libres-a-vegetaliser-elv-formatted.json',
            {'n_sq_elv': 'Identifiant',
             'st_area_shape': 'Superficie',
             'st_perimeter_shape': 'Périmètre',
             'c_sec': 'Section',
             'c_asp': 'Aspect',
             'geo_point_2d': 'Coordonnées géographiques'})
```

3. Les APIs

Les API sélectionnées ont été choisies en fonction de leur pertinence par rapport aux objectifs du projet.

En effet, ces dernières fournissent des données essentielles liées à l'urbanisme, à l'aménagement du territoire, au logement et à d'autres aspects de la vie urbaine qui sont au cœur de l'objet du projet. Par exemple, les données sur les arrondissements, les quartiers administratifs, les secteurs de risques, les espaces verts et les logements encadrés par des loyers sont tous des éléments cruciaux à prendre en compte lors de l'analyse de la dynamique urbaine de Paris.

De plus, ces informations sont mises à disposition par le gouvernement et sont accessibles publiquement, ce qui garantit la fiabilité et l'authenticité des données.

En résumé, ces API ont été choisies en raison de leur pertinence thématique, de la disponibilité de leurs données, de leur compatibilité avec les objectifs de l'analyse et de leur potentiel pour générer des insights significatifs pour le projet.

PLU - Espaces Libres à Végétaliser (ELV)

Description : Cette source de données fournit des informations sur les espaces libres à végétaliser prescrits par le Plan Local d'Urbanisme (PLU) de Paris.

Exemple de Données :


```
{
    "Identifiant": 4604,
    "Superficie": 52.77880947163109,
    "P\u00e9rim\u00e8tre": 33.05862591876806,
    "Section": null,
    "Aspect": null,
    "Coordonn\u00e9es g\u00e9ographiques": {
      "lon": 2.323241978305356,
      "lat": 48.83377330996698
    }
  }
```


PLU - Secteurs de Risques Délimités par le PPRI

Description : Cette source fournit des informations sur les secteurs de risques délimités par le Plan de Prévention du Risque d'Inondation (PPRI) mentionnés dans le PLU de Paris.

Exemple de Données :


```
{
    "zonage": "B",
    "n_sq_pprizone": 46,
    "st_area_shape": 540.2740820561355,
```

```
    "st_perimeter_shape": 102.2926853236986,
    "geo_shape": {
      "type": "Feature",
      "geometry": {
        "coordinates": [...]
}
```

Arrondissements

Cette source fournit des informations sur les arrondissements municipaux de Paris.

Exemple de Données :

```
{
    "n_sq_ar": 750000005,
    "c_ar": 5,
    "c_arinsee": 75105,
    "l_ar": "5\u00e8me Ardt",
    "l_aroff": "Panth\u00e9on",
    "surface": 2539374.62284532,
    "perimetre": 6239.19539614,
    "geom_x_y": {
      "lon": 2.3507146095752587,
      "lat": 48.844443150532705
    }
  }
```

Quartiers Administratifs

Description : Cette source fournit des informations sur les quartiers administratifs de Paris.

Exemple de Données :

```
{
    "n_sq_qu": "750000022",
    "c_qu": "22",
    "c_quinsee": "7510602",
    "l_qu": "Od\u00e9on",
    "c_ar": 6,
    "n_sq_ar": "750000006",
    "perimetre": 3516.31446441,
    "surface": 716148.35103094,
    "geom_x_y": {
      "lon": 2.3363388275876775,
      "lat": 48.847800629292664
    },
    "geom": {
```

```
      "type": "Feature",
      "geometry": {
        "coordinates": [...]
}
```


Logement - Encadrement des Loyers

Description : Cette source fournit des informations sur les loyers de référence par quartier à Paris depuis la mise en place de l'encadrement des loyers en 2019.

Exemple de Données :


```
{
    "annee": "2019",
    "id_zone": 10,
    "id_quartier": 49,
    "nom_quartier": "Salp\u00eatri\u00e8re",
    "piece": 1,
    "epoque": "Apres 1990",
    "meuble_txt": "meubl\u00e9",
    "ref": 32.2,
    "max": 38.6,
    "min": 22.5,
    "ville": "PARIS",
    "code_grand_quartier": 7511349,
    "geo_shape": {
      "type": "Feature",
      "geometry": {
        "coordinates": [...]
}
```


Espaces Verts et Assimilés

Description : Cette source fournit des informations sur les espaces verts publics gérés par la Ville de Paris.

Exemple de Données :


```
{
    "Identifiant": 12176,
    "Nom": "JARDINIERES DE LA RUE HENRI RIBIERE",
    "Type": "D\u00e9corations sur la voie publique",
    "Cat\u00e9gorie": "Jardiniere",
    "Num\u00e9ro": 24,
    "Compl\u00e9ment": null,
    "Type de voie": "RUE",
    "Libell\u00e9 de voie": "HENRI RIBIERE",
    "Code postal": "75019",
```

```
    "Superficie": 118,
    "Surface totale r\u00e9elle": 120,
    "Surface horticole": 120,
    "Pr\u00e9sence de cl\u00f4ture": "Non",
    "P\u00e9rim\u00e8tre": 295.03370964,
    "Ann\u00e9e d'ouverture": "2019",
    "Ann\u00e9e de r\u00e9novation": null,
    "Ancien nom": null,
    "Ann\u00e9e de changement de nom": null,
    "Nombre d'entit\u00e9s": 7,
    "Ouvert/Ferm\u00e9": null,
    "ID de division": 119,
    "ID d'atelier horticole": 43,
    "ID 3D ENB": "JDE11688",
    "Site Villes": "SV",
    "ID \u00c9quipement": "11688",
    "Comp\u00e9tence": "CA",
    "G\u00e9om\u00e9trie": {
      "type": "Feature",
      "geometry": {
        "coordinates": [...]
}
```