SIMPLON.CO

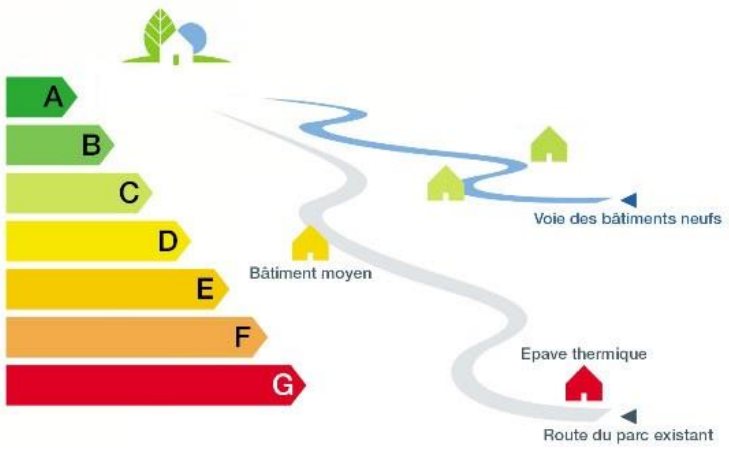
Certification Développeur Data

Rapport projet chef d'ouvre

Abire MAHFOUZE

Trajectoire de la réduction de GES dans le secteur de bâtiment en

LOIRE-ATLANTIQUE



SOMMARE

Contents

INTRODUTION

Le changement climatique est en cours et déjà sensible à l'échelle mondiale. A plus long terme, l'augmentation des épisodes climatiques extrêmes (nombre et durée des vagues de chaleur, des sécheresses, des pluies fortes, …) va avoir des conséquences directes ou in indirectes sur la population, la santé humaine et animale, les activités économiques, la biodiversité, les paysages...

L'augmentation des émissions des gaz à effet de serre (GES) liées aux activités humaines est la cause majeure de ce changement. La réduction de ces émissions est une nécessité pour limiter le plus possible le dérèglement climatique et ses conséquences.

En France les émissions de GES se répartissent entre ces différents secteurs :

- Transports 29%
- Bâtiments résidentiels/Tertiaires 20%
- Agriculture, sylviculture et gestion des sols 19%
- Industries manufacturières 18%
- Transformation de l'énergie 11%
- Déchets 3%

En résumé les secteurs qui émettent le plus de CO2 sont les bâtiments et le transport

Il est donc légitime de se demander quel est le bilan de l'énergie de bâtiment. Ce bilan peut déboucher sur des pistes de solution.

Nous allons répondre à cette question dans le département LOIRE-ATLANTIQUE, à partir des données sur la performance énergétique de logement dans ses différentes communes. Donc, nous allons créer un tableau de bord qui va permettre d'identifier les logements à rénover.

I. L'analyse de demande :

1. L'état de l'art :

Dans l'introduction de ce compte rendu nous avons pu évoquer succinctement les enjeux de la réduction des émissions des gaz à effet de serre dans le secteur de bâtiment. Nous allons donc dans cette partie aborder plus précisément cette thématique pour en définir les contours et les enjeux et plus particulièrement sur comment construire la base de données qui sera le socle des futures évolutions du tableau de bord.

Nécessité et objectifs de la réduction de GES dans le secteur de bâtiment :

Le réchauffement climatique, affectant déjà les écosystèmes, la météo ou les cultures, est majoritairement dû à l'émission de gaz à effets de serre générée par les activités humaines. Or, le secteur de bâtiment est en grande partie responsable. Ses émissions globales sont réparties dans trois secteurs d'activité :

- Le bâtiment, pour les **émissions directes** (scope 1) liées aux **consommations d'énergie** pendant la phase d'usage des bâtiments (gaz, fioul…) et aux fuites de fluides frigorigènes
- La production de l'énergie, pour les émissions indirectes (scope 2) liées à la consommation d'électricité, de chaleur ou de froid via les réseaux urbains
- L'industrie, pour les émissions indirectes (scope 3) liées à la fabrication des matériaux et équipements mis en œuvre dans les constructions neuves ou rénovations

*En 2016, **le secteur du bâtiment représente 26% des émissions nationales** sur ses scopes 1+2 (consommations énergétiques), soit environ 115 MtCO2e. Par ailleurs, la construction neuve (résidentielle et tertiaire) équivaut à environ 30 MtCO2e. Autrement dit, le secteur global représente environ 30% des émissions annuelles nationales.*

Pour lutter contre le changement climatique, la France définie sa feuille de route, La Stratégie Nationale Bas-Carbone (SNBC), Introduite par la Loi de Transition Energétique pour la Croissance Verte (LTECV). Son objectif est la décarbonation quasi-complète des secteurs du transport, de l'énergie et du secteur du bâtiment d'ici 2050.
Les cibles du projet SNBC pour le secteur du bâtiment :

- Scopes 1+2 : 0 MtCO2e en 2050 (hors fuites résiduelles de gaz)
- Scope 3 : non précisé. Le secteur de l'industrie, notamment l'industrie cimentière et sidérurgique, doit réduire de 81% ses émissions entre 2015 et 2050

Pour atteindre ces cibles, le **levier principal reste la massification de la rénovation du parc immobilier français.** La SNBC souhaite porter le nombre annuel de rénovations complètes et performantes dans le logement de 300 000 sur 2015-2030 à 700 000 sur 2030-2050, **en visant en priorité les passoires énergétiques**.
Plus concrètement, ces objectifs signifient que le parc immobilier français doit suivre une trajectoire de réduction d'émissions de CO2 qui tend vers 0 en 2050.

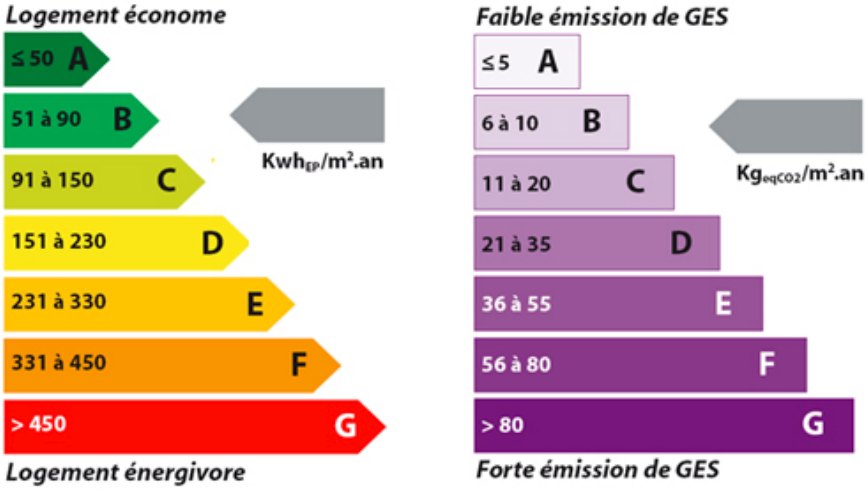
2. La problématique et les attentes du client :

Les contours du projet commencent à être bien plus clairs et le sens de la thématique est aussi très parlant concernant la rénovation énergétique de logement et d'apporter une information sur ce type de rénovation.

Les enjeux liés à la rénovation énergétique nécessitent de disposer d'une bonne connaissance de la consommation énergétique des logements.
Le diagnostic de performance énergétique **(DPE)** est le document qui informe sur la performance énergétique d'un logement ou d'un bâtiment, en mesurant sa consommation d'énergie et son impact en termes d'émission de gaz à effet de serre. Il s'inscrit dans le cadre de la politique énergétique définie au niveau européen afin de réduire la consommation d'énergie des bâtiments et de limiter les émissions de gaz à effet de serre.

Le DPE décrit le bâtiment ou le logement (surface, orientation, murs, fenêtres, matériaux, etc), ainsi que ses équipements de chauffage, de production d'eau chaude sanitaire, de refroidissement et de ventilation. Il indique la consommation annuelle d'énergie, le positionnement en consommation d'énergie et en émission de GES, enfin il propose des recommandations d'améliorations énergétiques.
Pour faciliter la lecture, le DPE comprend deux étiquettes fig(1) :

- L'étiquette « énergie »
- L'étiquette « climat »



Fig(1) : Etiquette DPE

Les étiquettes F et G du DPE, regroupant les logements qualifiés de « passoires thermiques », donc viser par la stratégie de la rénovation.

La rénovation de logement est encadrée par les **réglementation thermique RT**, Il existe
deux types de réglementations pour les bâtiments existants :

La réglementation thermique globale

Pour que la réglementation thermique globale s'applique, il faut que les bâtiments résidentiels ou tertiaires **respectent les trois conditions suivantes :**

- Il faut le bâtiment ait été achevé **après le 1er janvier 1948**
- Il faut la **Surface Hors Œuvre Nette* (SHON) soit supérieure à 1000 m²**
- **Il faut que le coût des travaux de rénovation thermique soit supérieur à 25 % de la valeur du bâtiment (hors foncier).**

La réglementation thermique élément par élément

La réglementation thermique élément par élément **s'applique dans les autres cas où la réglementation thermique globale ne s'applique pas**

Fig(2) : les élément à rénover définir par RT

A Partir de cette problématique, les attentes sont les suivantes :

1) Identifier les logements à rénover, en appuyant sur le DPE.
2) Identifier le type de rénovation en appuyant sur le RT.

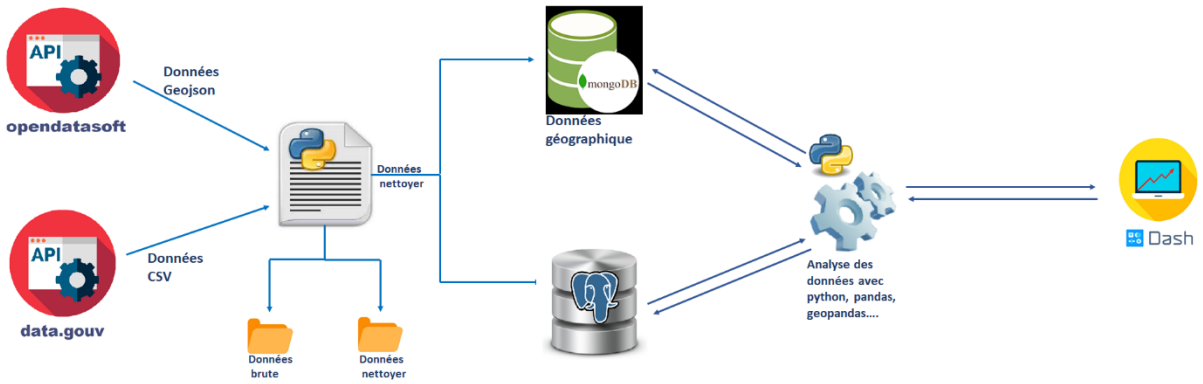
Pour répondre à cette attente, nous allons :

- ✓ Collecter de la donnée sur la performance énergétique de logement des communs du département LOIRE-ATLANTIQUE,
- ✓ Concevoir et développer une base de données
- ✓ Et enfin visualiser l'analyse de ces données sur un tableau de bord dans le but de mettre en évidence des résultats émanant d'une problématique préalablement définie.

3. La traduction métier du besoin client :

Afin de visualiser le déroulement du projet, Nous avons mis en place un schéma fonctionnel sur l'architecture de ce projet. fig(3) On peut y suivre les différentes étapes avec notamment la collecte de la donnée, la préparation, le stockage et la visualisation.

Ces étapes d'un point de vue technique seront détaillées davantage dans la seconde partie appelée la mise en œuvre du projet. Ce schéma a pu être alimenté au fur et à mesure de son déroulement.



Fig(3) : le schéma fonctionnel du projet

II. La mise en œuvre du projet

1. La gestion de projet :

Nous avons commencé par décrire l'organisation générale du projet en définissant les phases suivantes :

Phase 1 : l'initialisation

Dans cette première phase nous avons défini les besoins clients (problème à résoudre).

Phase 2 : cadrage et préparation

Préparation principalement orientée gestion de projet. Nous avons planifié dans le temps les différentes étapes et les différents outils à mobilisés. Dont, le choix de source de données, de base de données (SQL /NOSQL), les différentes technologies à utiliser (python,PostgreSQL, MongoDb….)

Phase 3 : réalisation

Travaille sur les tâches planifiées, trouver des solutions pour satisfaire les objectifs définis. Autocontrôle d'avancement au regard du plan projet initial et l'ajuste si nécessaire

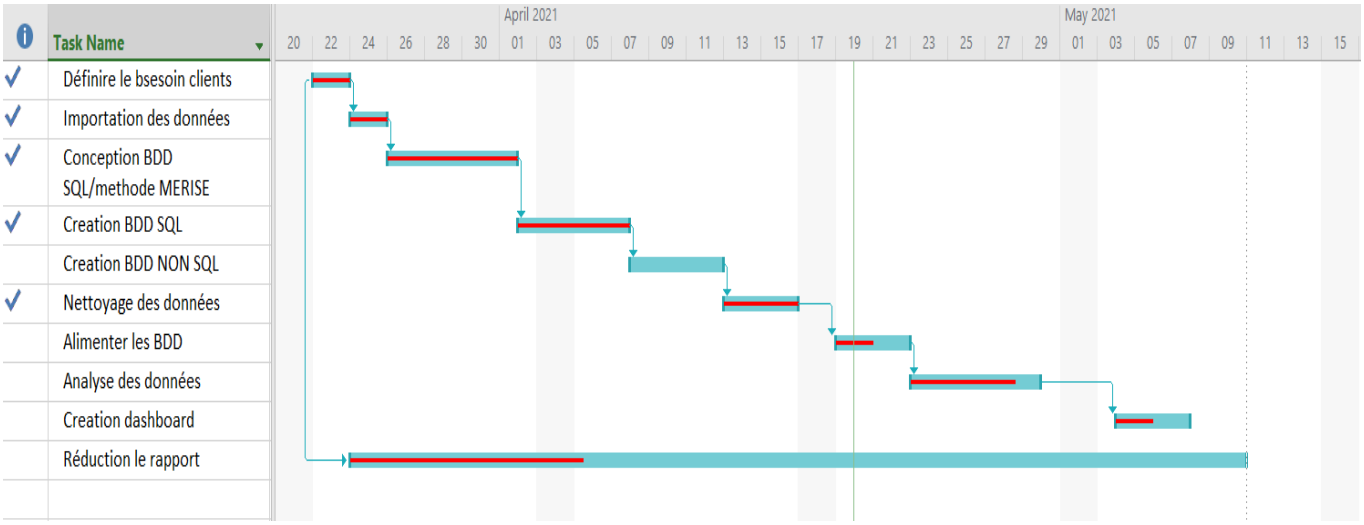
Phase 4 : clôture

Date limite du projet, soit le19 mai 2021.

Après avoir délimité les étapes du projet, pour l'organisation et le suivi nous avons choisi MsProject. Microsoft Project est un outil de gestion de projet permet la planification des projets, la création des tâches, leur hiérarchisation, définir des liens

entre elles et estime de la durée nécessaire à la réalisation de chaque tâche. Il permet aussi de mettre à jour l'avancement de chaque tâche en l'assignant un pourcentage d'avancement.

Microsoft Project propose différentes représentations graphiques du projet. Nous avons choisi le diagramme de Gantt. Il offre aussi la possibilité d'utiliser la méthode de visualisation de Kanban. Qui s'appuyant sur la visualisation du flux de travail (à faire, en cours, termine)



Fig(4) : gestion de projet

2. Les aspects techniques du projet :

Nous avons utilisé les technologies suivantes pour réaliser ce projet :

Le langage Python : tous les tâche sont automatisé avec un script python (importer, sauvegarde, nettoyer et analyse des données), différent bibliothèque python sont utilisé pour compléter le travail.

- pandas et geopandas : Nettoyer et analyser les données .
- psycopg2 : Créer et manipuler la base de données PostgreSQL
- pymongo : Créer et manipuler la base de données MongoDB
- requests : Récupérer des données à partir d'Api
- folium : Visualiser les résultats d'analyse sur une cartes
- plotly.express : Visualiser les résultats d'analyse sur différent type de graphique (ex : pie chart)
- Dash : Créer une dashboard.
- re : Régulière expression, nettoyer des données

pgAdmin : un outil d'administration graphique pour PostgreSQL

MongoDB Compass : un outil d'administration graphique pour MongoDB

Dbeaver : un logiciel permettant l'administration et le requêtage de base de données, nous avons l'utilisé pour visualiser le diagramme d'entité de la base de données SQL

2.1. L'arborescence du projet :

```
projt_certif
|   requirements.txt
|   tree_project.txt
|
+---data
|   +---backup_bd
|   |   \---20210506-100341
|   |           backup.sql
|   |
|   +---data_brute
|   |       44.csv
|   |       correspondance-code-insee-code-postal.geojson
|   |
|   +---data_demo
|   |       donnees_demo.csv
|   |
|   +---data_nettoyee
|   |       data_nettoyee_44.csv
|   |
|   \---metadonnees
|           metadata_donnee_brute.json
|           metadata_donnee_geographique.json
|           metadata_donnee_nettoyer.json
|           metadonnees_bd.json
|
+---documentation
|       ER_diagrame.png
|       MCD.png
|       MLD.png
|       projet_chef_ovre .pptx
|       rapport.docx
|
+---env_projet_certification
|   |   pyvenv.cfg
|   |
|   +---etc
|   +---Include
|   +---Lib
|   |   \---site-packages
|   +---Scripts
|   \---share
\---scripts
    |   analyse_visualisation.py
    |   backup.py
    |   creation_bd_tables.py
    |   importer_nettoyer_inserer.py
    |   map.html
    |   utils.py
    |
    \---assets
            dpe.jpg
            style.css.txt
```

2.2. La collecte de la donnée :

Notre thématique imposée pour ce projet final est donc la réduction l'émission de GES dans le secteur de bâtiment en identifiant la performance énergétique *(logement avec la classe de consommation des énergies G, F)* afin de l'améliore par la rénovation *(globale ou par élément).*

Pour répondre à cet objectif nous avons utilisé deux jeux de données :

1) **Le diagnostic de performance énergétique des logements (DPE)**
   Producteur : ADEME (Agence de la transition écologique)
   Description : Le DPE décrit le bâtiment ou le logement (surface, orientation, murs, fenêtres, matériaux, etc). Il indique, suivant les cas, soit la quantité d'énergie effectivement consommée (sur la base de factures), soit la consommation d'énergie estimée pour une utilisation standardisée du bâtiment ou du logement.
   Licence : Licence ouverte.
   Mise à jour au 26/03/2021

A partir de cette source, les données DPE des logements des communes du département LOIRE-ATLANTIQUE sont récupérer en format csv, ce fichier comporte 263 430 lignes et 131 colonnes, la description détailles de ces données (dictionnaire des données) se trouve sur le site : https://data.ademe.fr/datasets/dpe-44
Nous n'allons pas intéresse par tous ces colonnes, nous allons concentre que sur les données qui décrire le logement, l'années de construction, le surface, la consommation d'énergie…)

Pour automatiser le processus d'importation, Nous avons créé une fonction *fig(5)* avec python (**import_donnees**), il prend en entrée le code de département, afin d'interroger **l'api koumoul.com** et récupérer les données en format csv. Sur cette api les fichier csv sont nommé selon le code de département. Nous avons récupéré les données du département LOIRE-ATLANTIQUE pour ce projet, donc le **code_departement** est **44**. La fonction **import_donnees** va tout d'abord détermine le chemin de dossier où le fichier csv importé va être enregistre, dans notre projet ça sera dans le dossier **data_brute**, puis il donnée un nom au ce fichier selon le code de département (**44.csv**), ensuite il va interroger l'api avec `requests.get` Et enfin sauvegarder le fichier `csv_file.write`

```python
def import_donnees(code_departement):
    """ récupere les données a partire de API koumoul.com et l'enregistre en local dans le fichier data_brute
    """
    try:
        one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))
        data_path = os.path.join(one_directory_up_path, 'data','data_brute')
        csv_file = open(os.path.join(data_path,f'{code_departemet}.csv'), 'wb')
        url=f'https://koumoul.com/s/data-fair/api/v1/datasets/dpe-{code_departement}/data-files/dpe-{code_departement}.csv'
        req = requests.get(url)
        url_content = req.content
        csv_file.write(url_content)
        csv_file.close()
    except import_donnees.Error as e:
        print("Erreur lors de l'importation des données")
        print(e)
        return
    print("Les données importées et sauvegardés  avec succès")
```

Fig(5) : fonction python pour importer les données

2) Correspondance Code INSEE - Code Postal :

Producteur : OpenDataSoft

Description : des informations sur le nom de commune de département LOIRE-ATLANTIQUE, le code postal, le code INSSE et geo shape (type, coordinats)

Licence : Licence ouverte.

Mise à jour au :22 avril 2016

A partir de ce source les données sont récupérer en format geojson*, et utilisé pour faire une visualisation sur une carte, il comporte 221 lignes (communes)*

2.3. Réglementation sur l'utilisation des données :

Les deux jeux de données sont des licences ouvertes donc nous avons la liberté de :

- ✓ Reproduire, copier, publier et transmettre.
- ✓ Diffuser et redistribuer.
- ✓ Adapter, modifier, extraire et transformer, notamment pour créer des informations dérivées

2.4. Le choix de la technologie de la base de données :

Nous avons choisi de créé deux bases de données, afin de répondre à l'objectif de ce projet :

2.4.1. Une base de données SQL pour stoker les données DPE :

Dans ce type de base de données, les données sont organisées en tables et ces tables sont liées par une "relation", les relations entre les tables assurent la **cohérence et l'intégrité** des données à l'aide de contraintes de clés étranges, qui permet d'empêcher de supprimer des données d'une table s'ils ont lié à des autres données dans d'autre table, puis vue que les données sont **bien structurées** ils sont **facilement requêtables**, donc cette solution est bien adapter pour faire l'analyse et formuler des requête complexe

2.4.2. Une base de données NoSQL pour stocker les données géographiques :

NoSQL stocke et manipule des documents qui correspondent à des collections d'objets. Ce type de base de données est très flexibles, chaque collection peut contenir tous types de données, **pas de schéma de tableau fixe** dans lequel les données doivent être définies avant l'enregistrement comme le cas en SQL, donc l'enregistrement de nouvelle collection se fait directement sans passer par des longues procédures de conception, NoSQL peuvent lire et traiter de gros volumes de données à une vitesse record, donc ce type de base de données est bien adapter de stocker les données géographique volumineux (ex chaque conure de commune est représenter par 22 points.

2.5. Conception et développement de la base de données :

2.5.1. Base de données SQL :

Nous avons utilisé La méthode MERISE pour la conception et le développement de la base de données en suivant ses trois niveaux :

1. Le niveau conceptuel, création le model conceptionnel de données (MCD)
2. Le niveau logique, passage du MCD au MLD (Model logique de donnée)
3. Le niveau physique, passage au SQL

Cette étape est réalisée à l'aide de Looping, un logiciel de modélisation conceptuelle de données

A. Elaboration du modèle conceptuel des données (MCD) :

A partir de la fichier CSV importée, nous avons déterminé le contenu de notre base de données, donc nous avons définir neuf entités, chaque entité comporte plusieurs propriétés (attributs) et un identifiant unique, puis nous avons déterminé les associations (relation) entre les entités afin de définir le lien sémantique entre elle. Fig(6)
Les relations de nos tables sont one-to-one entre les entités (logement_description, information_sur_diagnostioque_DPE) et (logement_description, Adresse). One-to-many pour les restes. Par exemple une adresse appartient à une seule commune et une commune peut comporter plusieurs adresses.

B. Passage du MCD au MLD :

Les règles de passage au modèle logique MLD sont comme la suite fig(7) :

- Toute entité du MCD devient une relation, dont la clef est l'identifiant de cette entité. Chaque propriété de l'entité devient un attribut de la relation correspondante

- Les associations ayant au moins une cardinalité de type 1,1 devient une clef étrangère dans la relation qui correspond à l'entité se situant du côté de cette cardinalité 1,1. Cette clef étrangère fera donc référence à la clef de la relation correspondant à la seconde entité reliée par l'association.
- Lorsque deux entités sont toutes deux reliées avec une cardinalité 1,1 par une même association, on choisit de placer la clef étrangère du côté de la relation correspondant à l'entité ayant le plus de liaisons avec les autres (logement description-information sur diagnostique DPE, clé étrange côte de la relation logement description)

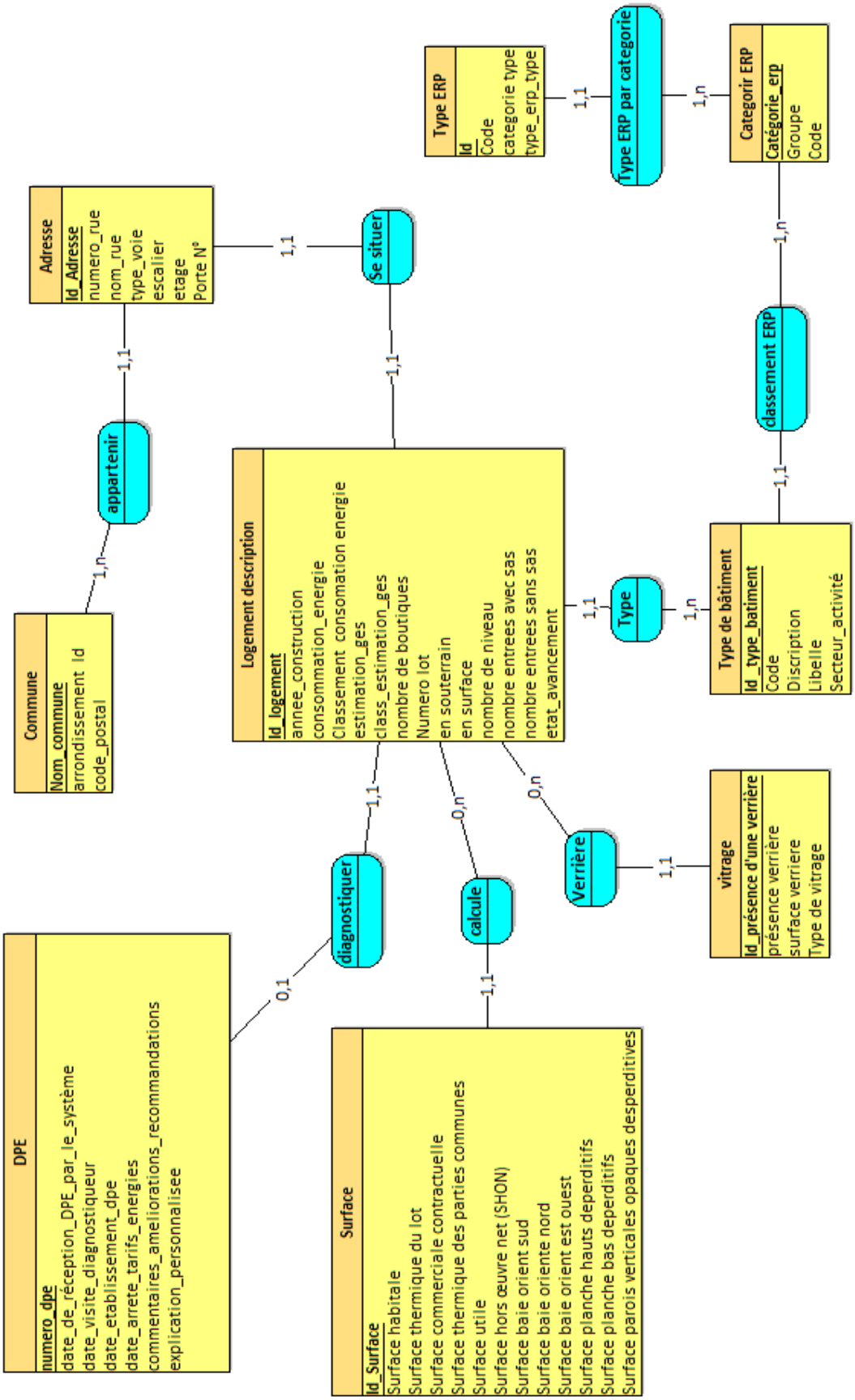
C. Passage au SQL :

A partir de MLD et avec une code python fig(8), nous avons créés la base de données et les tables sous PostgreSQL.

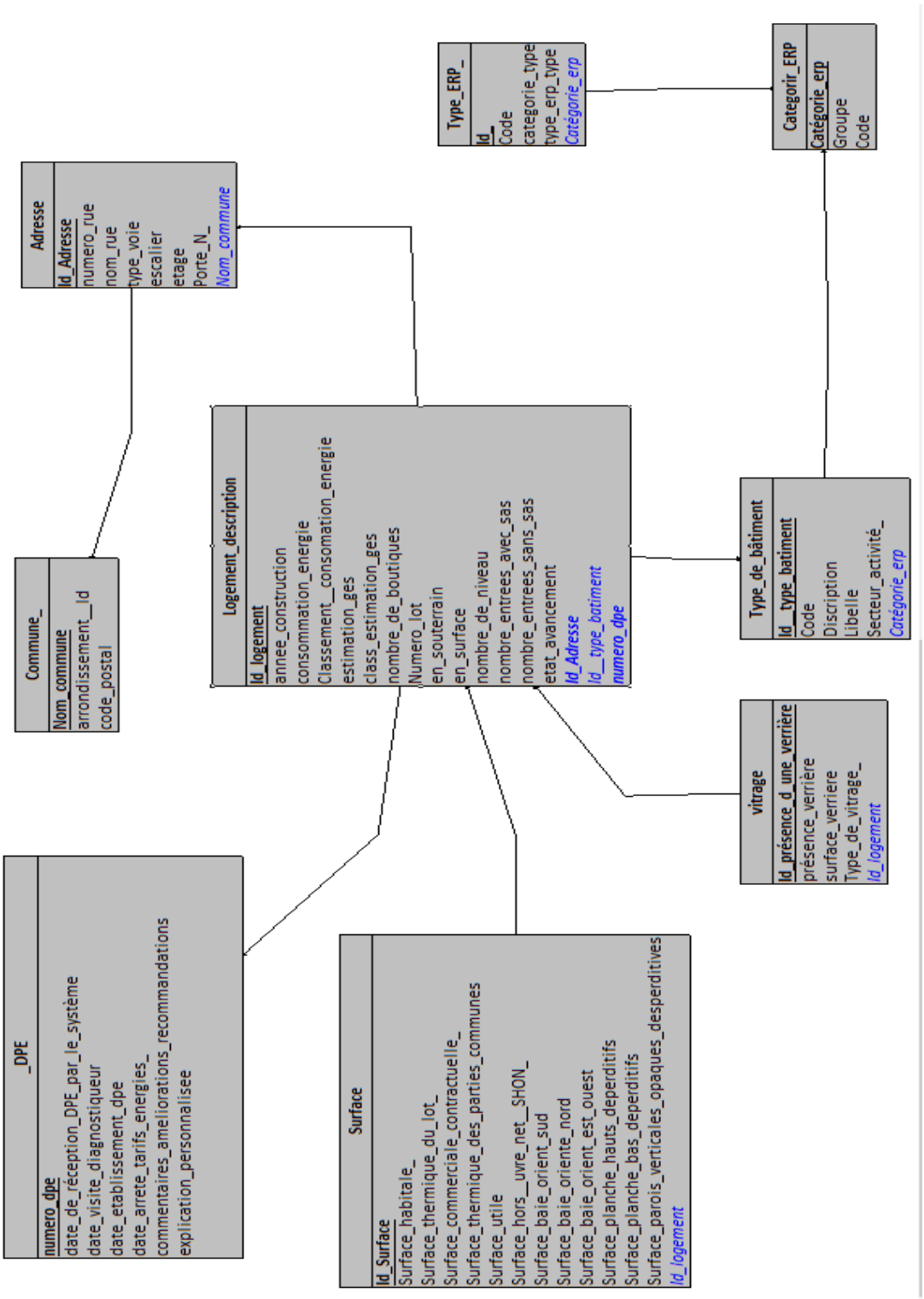
- ✓ Nous avons fait une fonction avec python, pour créer la base de données. Il prend en entre le nom de base de données à créer et les paramètres pour connecter au SGBDR (PostgreSQL). Tous d'abord il connecte au postgreSQL, ensuit il vérifie si la base de données existe déjà, s'il existe il rendre un message que la base de données existe, sinon il va la créer.
- ✓ Puis pour créer les tables, nous avons fait une fonction python qui prend en entré la connexion à la base de données, il contient les requetés SQL nécessaire pour créer les tables

Suite à la création des tables nous avons utilisé pgAdmin l'outil d'administration graphique pour PostgreSQL pour Visualiser et interroger les tables crées.

Et pour visualiser le diagramme d'entité, nous avons utilisé Dbever fig(9) le diagramme d'entité va nous permettre de visualiser les tables créer, les noms de ses colonnes et les type de données dans chaque colonne, ainsi les relations entre elle issue de la clé étrangère



Fig(6) : Modèle conceptuelle de données MCD



Fig(7) : modèle logique de données MLD

```python
def create_database(db_name,utilisateur, mot_passe, host='localhost', port=5432):
    """Creation base de donne postgresql
    """
    try:
        conn = psycopg2.connect(user=utilisateur, password=mot_passe, host=host, port=5432)
        conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
        cursor = conn.cursor()
        cursor.execute("SELECT datname FROM pg_database")
        nom_db_exist = cursor.fetchall()
        if (f"{db_name}",) in nom_db_exist:
            print("La base de données existe")
        else:
            cursor.execute(f"CREATE DATABASE {db_name}")
            print("La base de données a été crée avec succès")

    except psycopg2.Error as e:
        print("Erreur lors de la connection et la creation de la base de données")
        print(e)
        return None
    cursor.close()
    conn.close()
    return True
```

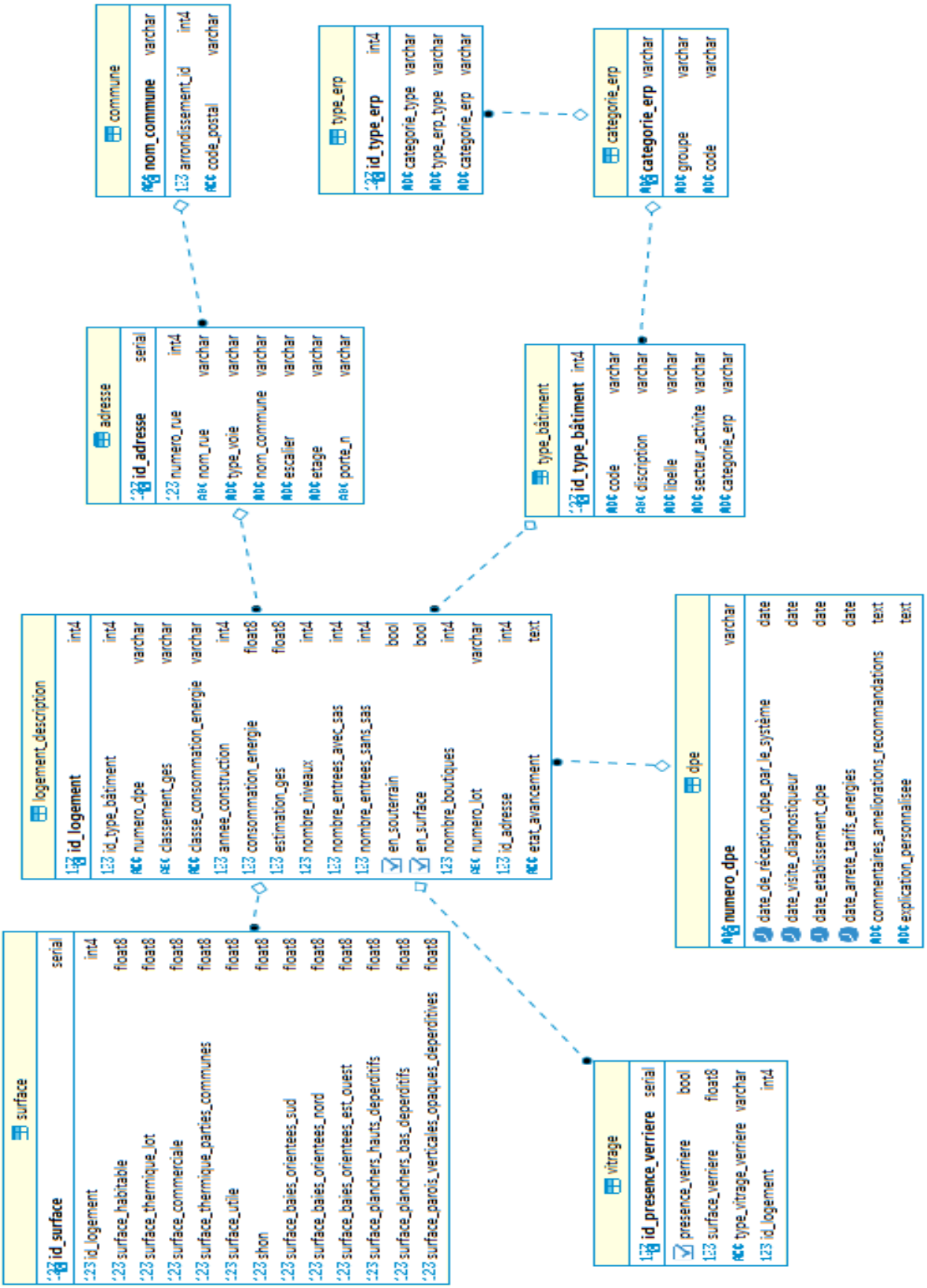
```python
def create_table(conn):

    """Creation des tables
    """
    try:
        cursor = conn.cursor()


        # creation de la table classe_estimation_ges:
        cursor.execute("""CREATE TABLE IF NOT EXISTS classe_estimation_ges(
                       classement_ges VARCHAR PRIMARY KEY,
                       ges VARCHAR)
                       """)
        # creation de la table classe_consomation_energie:
        cursor.execute("""CREATE TABLE IF NOT EXISTS classe_consomation_energie(
                      classe_consommation_energie VARCHAR PRIMARY KEY,
                      consommation VARCHAR)
                      """)
        # creation de la table Categorie ERP:
        cursor.execute("""CREATE TABLE IF NOT EXISTS categorie_erp(
                       categorie_erp VARCHAR PRIMARY KEY,
                       groupe VARCHAR,
                       code VARCHAR)
                       """)
        # creation de la table type ERP:
        cursor.execute("""CREATE TABLE IF NOT EXISTS type_erp(
                      id_type_erp INTEGER PRIMARY KEY,
                      categorie_type VARCHAR,
                      type_erp_type VARCHAR,
                      categorie_erp VARCHAR,
                      FOREIGN KEY (categorie_erp) REFERENCES categorie_erp(categorie_erp))
                      """)

        # creation de la table commune:
        cursor.execute("""CREATE TABLE IF NOT EXISTS commune(
                      nom_commune VARCHAR PRIMARY KEY,
                      arrondissement_Id INTEGER,
                      code_insee_commune INTEGER,
                      code_postal VARCHAR
                      )
```

Fig(8) : create_database : scripts python pour créer la base de données, create_table : Une partie du script python de pour créer les tables SQL



Fig(9) : Diagramme d'entité

2.5.2. Base de données NoSQL :

Nous avons utilisé MongoDB pour créer cette base de données, comme nous avons mentionné précédemment, la création de cette base de données ne nécessite pas de conception. Sur MongoDB la base de données se crée lorsque l'insertion de la première collection, donc nous avons automatiser cette tâche avec un scripte python fig(10) :

```
def insert_collectionMongoDB(db_name,collection_name,nom_departement):
    """ En mongodB, la base de donnees se créer lors de la première sauvegarde
        de la valeur dans la collection définie
        nom_departement: pour récuperer des donnees geojson d'un departement apartir d'api
    """
    try:
        myclient = pymongo.MongoClient("mongodb://localhost:27017/")                # connecter mongodb
        bd_nosql=myclient[db_name]                                                  # connecter au bas de donnees
        collection_list = bd_nosql.list_collection_names()                          # verifie l'existance de la collection avant de créer
        if collection_name in collection_list:
            print("La collection existe")
        else:
            geojson_donnees='https://public.opendatasoft.com/explore/dataset/correspondance-code-insee-code-postal'+\
                            f'/download/?format=geojson&refine.nom_dept={nom_departement}&timezone=Europe/Berlin&lang=fr'
            reponse=requests.get(geojson_donnees)
            obj = reponse.json()
            mycol = bd_nosql[collection_name]
            mycol.insert_one(obj)
            print('Les données est insere avec succée')
    except:
        print('Erreur lors de insertion des données')
    return True
```

Fig(10) : script python pour créer la base de donnée NoSQL

Cette fonction prend en entrée le nom de base de données et de collection a créer, le nom de département à fin d'interroger l'api et récupérer les données selon le nom de département, dont il va tout d'abord : connecter au MongoDb, vérifier si la collection existe déjà, si oui il va rendre un message que la collection existe, si non il va interroger l'api, récupérer les données et créer la collection en insérant les données.

2.6. Alimenter la base de données :

Nous avons passés par les étapes suivant pour alimenter la base de données :

2.6.1. Nettoyage des données :

Nous avons commencé par crée avec python la fonction lire_données, fig(11) cette fonction va lire le fichier 44.csv enregistre dans le dossier data_brute, en forme de pandas data frame.
Donc, il prend en entre le code_departement, définir le schéma d'accès et lire le fichier en forme de pandas data frame

```python
def lire_donnees(code_departement):
    """lire les données csv en forme de pandas dataframe
    """
    one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))
    data_path = os.path.join(one_directory_up_path, 'data','data_brute')
    donnees_csv=pandas.read_csv(os.path.join(data_path,f'{code_departement}.csv'),sep=",",low_memory=False)
    return donnees_csv
```

Fig(11) : script python pour lire des données csv en forme de pandas data frame

Ensuite, nous avons créés plusieurs fonctions afin de réaliser les tâches de nettoyage suivants : ***en détail dans l'annexe***

1. def nettoyage_error(donnees_df): il prend en entre les données lu dans l'étapes précédant, à l'aide de `replace` il va remplacer @ par A dans le colonne numero_dpo, et 0 par No ERP dans le colonne categorie_erp, il retourne les données nettoyer.
2. def nettoyage_drop_duplicates(donnees_df): il prend en entre les données issu de l'étape président, il utilise `drop_duplicates` afin de supprimer les doublon selon le numero_dpo, il retourne les données nettoyer.
3. def nettoyage_suprim_0value(donnees_df): il prend en entre les données issu de l'étape président, il utilise `drop` afin de supprimer les lignes lors que la consomation_energie est zéro, ou `dropna` lors que la consomation d'energie n'est pas fournie .
4. def nettoyage_suprim_col(donnees_df,colonne_supprimer): Pour supprimer les colonnes non utilisé dans ce projet . Il prend en entre les données issues de l'étape président et une liste de colonnes à supprimer, il utilise `drop` pour faire cette tâche, il rendre les données nettoyer.
5. def nettoyage_to_numeric(donnees_df,donnees_numeric): Transforme les données de certain colonne en numérique, Il prend en entre les données issu de l'étape président, une liste de colonnes a transformer en numérique, il utilise `pandas.to_numeric` pour réaliser cette tâche. Il rendre les données nettoyer.
6. def nettoyage_to_boolean(donnees_df, donnees_boolean):
   Transformer les données des colonnes définies dans la liste donnees_boolean to true, false. Dans le fichier csv importe, les données boolean sont
   Présentés en 0,1 mais le boolean en PostgreSQL est true, fals
   Donc, à l'aide `replace`([0,1],["f","t"]), nous avons réalisé cette changement.
7. def nettoyage_suprim_commune(donnees_df,df_commune): cette fonction va supprimer les données des communes n'appartient pas au département en étude, pour notre projet c'est le département LOIRE-ATLANTIQUE. Cette fonction prend en entre les données issues de l'étape précèdent donnees_df, les nom de commune d'un département df_commune, il va croiser les deux jeu

de données, il va supprimer les données de communes n'appartient pas au département à l'aide de `isin`, avant de croiser les deux jeu de données, nous avons transformer les nom de communes dans donnees_df en majuscule `str.upper()` afin d'assure que les nom de communes sont écrire de même façon dans les deux jeu de données

8. def nettoyage_adress(donnees_df): l'adresse est définie par trois colonnes, le numéro de rue, le type de vois et le nome de rue. Dans le fichier csv importé les donné es de ces trois colonnes sont concaténer dans une de ces colonnes pour les plus par des lignes, Pour résoudre ce problème nous avons créer cette fonction, qui va tout d'abord :
   Concaténer les données de ces trois colonnes en une seul, que nous avons l'appeler Full_adress. Pour concaténer les données de ces trois colonnes, les données doivent être du même type, donc nous avons tout transformer en string `astype(str),` ensuite pour identifier le numéro de rue à partir des données concaténer dans le colonne full_adress nous avons utilisé l'expression régulier pour définir le motif `(r'-?\d+\.?\d*')` , puis `findall` va Trouve toutes les sous-chaînes qui correspondent à la RE et les renvoie sous la forme d'une liste, qui va être enregistre dans le colonnes 'numero_rue', mais RE va retourner les sous-chaînes dans forme de liste, c'est-à-dire dans chaque ligne de la colonne 'numero_rue' on va trouver les données sous le forme [3] par exemple, ce qui va compliquer les analyse, pour détourner cette problème, nous avons utilisé `apply(lambda x : ', '.join(x)`

```
donnees_df['numero_rue']=donnees_df['Full_adress'].str.findall(r'-?\d+\.?\d*').apply(lambda x : ', '.join(x))
```

   La même manipulation est faite pour définir le type de voie, donc nous avons définir le motif qui comporte tous le type do vois possible :
   pattern=r'RUE|CHEMIN|ROUTE|AVENUE|CHAUSSEE|ALLEE|PASSAGE|BD|IMPASSE|RESIDENCE|BOULEVARD|BIS RUE|TER RUE|LOTISSEMENT|QUAI|SQUARE|LOT|AV'

```
donnees_df['type_voie']=donnees_df['Full_adress'].str.findall(pattern,flags=re.IGNORECASE).apply(lambda x : ', '.join(x))
```

   Finalement pour récupérer le nom de rue, nous avons utilisé `str.replace(pattern,'')` replace va chercher selon le motif(type de vois) définie les chaine de caractère correspondance et la supprimer (la remplacer avec rien ' '), pareil pour supprimer le numéro de rue `.str.replace(r'-?\d+\.?\d*','')` et en fin les article définie `str.replace(r'DES|LES|DE|LE|,','')`

2.6.2. Insertion des données :

Nous avons créé la fonction insertion_données pour réaliser cette tâche fig(12), elle prend en entrée :
Conn : la connexion à la base de données dpe_batiment crée sous PostgreSQL,

Conn est une fonction que nous avons créé avec python, elle prend en entrée le nom de base de données et les paramètres de connexions (mot de passe, user, host, port), elle retourne la connection.
Sql_insertion_table : la requête SQL pour insère les données dans une table définir, fig exemple de requête d'insertion
Donnees_df : les données nettoyer à insère. Ils sont en forme de pandas dataframe, récupérer en utilisant la fonction lire_donnees à partir du dossier data_nettoyer

```
def insertion_donnees(conn, sql_insertion_table, donnees_df):
    """insertion des donnees
    """
    try:
        cursor = conn.cursor()
        for index,row in donnees_df.iterrows():
            cursor.execute(sql_insertion_table, row)
        conn.commit()
    except psycopg2.Error as e:
        print("Erreur lors de l'insertion des données")
        print(e)
        return
    cursor.close()
    conn.close()
    print("Les données ont été insérées avec succès")
```

Fig(12) : script python pour insérer des données en forme de pandas df dans une base de données SQL

2.7. Automatiser le processus de sauvegarde de la base de données (back-up) :

Afin de préserver la base de données et de pouvoir les récupérer en cas de panne d'équipement ou autre incident entraînant une perte. Nous allons archiver la base de données (faire un back up).

Nous avons créé la fonction backup_db , fig(13) afin d'automatiser le processus de sauvegarde Cette fonction prend en entrée le nom de la base de données à archivé, il détermine le chemin de sauvegarde, pour ce projet, il sera dans le dossier backup_bd. Dans le dossier backup_bd, la fonction va créer un nouveau dossier nommé selon la date au moment de sauvegarde. Il va déterminer le chemin ou se trouve la base de donnée à archivé avec `os.chdir.` Enfin, le back_up va être crées avec `pg_dump`

```python
def backup_db(bd_nom):
    one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))
    BACKUP_PATH = os.path.join(one_directory_up_path, 'data','backup_bd')

    # Getting current DateTime to create the separate backup folder like "20180817-123433"
    DATETIME = time.strftime('%Y%m%d-%H%M%S')
    TODAYBACKUPPATH =os.path.join(BACKUP_PATH,DATETIME)
    # Checking if backup folder already exists or not. If not exists will create it
    try:
        os.stat(TODAYBACKUPPATH)
    except:
        x=os.mkdir(TODAYBACKUPPATH)
        print("le dossier today backup est crée")

    os.chdir("C:\\Program Files\\PostgreSQL\\13\\bin")

    dumpcmd=f"pg_dump -h localhost -U postgres -f {os.path.join(BACKUP_PATH,DATETIME,'backup.sql')} {bd_nom} "
    os.system(dumpcmd)
    return(True)
```

Fig(13) : scripte python pour faire un back up de la base de données

2.8. Créer le répertoire pour les métadonnées :

Les métadonnées peuvent répondre à de nombreux objectifs, Nous pouvons citer comme exemple :

- Faciliter la recherche d'information en décrivant le contenu de la ressource
- Favoriser l'interopérabilité par le partage et l'échange de données
- Améliorer la gestion et l'archivage
- Gérer et de protéger les droits de propriété intellectuelle
- Permettent d'authentifier une ressource

Nous avons créé un répertoire pour les métadonnées. Donc, nous avons récupère les métadonnées des fichiers :

- ✓ 44.csv (déjà importé à partir de l'api)
- ✓ donnee_nettoyee_44.csv : créer à partir de 44.csv, suite au processus de nettoyage
- ✓ Correspondance-code-insee-code-postal.geojson
- ✓ metadonnees de la base de données dpe_logement

Cette procédure est automatisée ave script python, donc nous avons créé deux fonctions, la premier pour récupérer les metadonnées des fichier csv, json. Le deuxième est pour récupérer les métadonnées de la base de données :

def get_file_metadata(nom_dossier, filename, metadata_filename,metadata):

def metadata_bd(conn,info_schema):

Les deux fonctions détaillent dans l'annexe.

III. Analyse et visualisation :

1. Exploitation de la base de données :

Afin d'éviter de répéter de taper des requêtes très longue *(des jointures, des calcules…)* lorsque de l'interrogation de la base de données pour répondre à notre problématique, Nous avons créé une vue, qui regroupe des colonnes de différentes tables, qui vont être utilise plusieurs fois pendant l'analyse. Cette vue comporte 5 colonnes, le colonne SHON

(Surface hors œuvre nette) est créé en faisant des calcule sur la colonne surface_habitable :

```
# creation une vue:
creation_vue_logement = """CREATE VIEW logement_info(
                        classe_consommation_energie,
                        consommation_energie,
                        annee_construction,
                        nom_commune,
                        SHON)
                        AS
                        SELECT logement_description.classe_consommation_energie,
                        consommation_energie,
                        annee_construction,
                        adresse.nom_commune,
                        ROUND((surface_habitable/0.80)::numeric, 2) AS SHON
                        FROM logement_description
                        LEFT JOIN adresse ON logement_description.id_logement=adresse.id_logement
                        LEFT JOIN surface ON logement_description.id_logement=surface.id_logement
                        ORDER BY classe_consommation_energie
                        """
```

A partir de cette vue, nous avons définir trois requêtes afin de répondre aux questions suivantes :
Les nombres de logement à rénover dans les différentes communes (classe G,F):

```
requet_sql="""SELECT nom_commune,
           COUNT(classe_consommation_energie) AS nombre_logement
           FROM logement_info
           WHERE classe_consommation_energie='G' OR classe_consommation_energie='F'
           GROUP BY nom_commune
           ORDER BY nombre_logement DESC;
           """
```

Type de rénovation (global/par élément) :

```
requet_sql="""SELECT nom_commune,
                    COUNT(nom_commune) AS numbre_logement
                    FROM logement_info
                    WHERE (classe_consommation_energie='G' OR classe_consommation_energie='F')
                    AND SHON>1000
                    AND annee_construction>1948
                    GROUP BY nom_commune;
                    """
```

```
requet_sql="""SELECT nom_commune,
                    COUNT(nom_commune) AS numbre_logement
                    FROM logement_info
                    WHERE (classe_consommation_energie='G' OR classe_consommation_energie='F')
                    AND SHON<1000
                    GROUP BY nom_commune;
                    """
```

Nous avons aussi formule une quatrième requête, afin d'avoir une idée générale sur le nombre de logements à rénover par rapport au nombres totale de logement dans l'échantillon en étude :

Nombres de logements par classe de consommation :

```
# visualisation nombres de logements par classe de consommation
requet_sql="""SELECT classe_consommation_energie,
            COUNT(classe_consommation_energie) AS nombre_logement
            FROM logement_info
            GROUP BY classe_consommation_energie
            ORDER BY classe_consommation_energie;
            """
```

Apres la définition de ces différents requetés, nous avons utilisé la fonction read_sql_query de la module de python pandas `pandas.read_sql_query(conn, requet_sql)`, cette fonction va enregistre les résulte de ses requetés dans une pandas data frame, pour ensuite faire la visualisation.

2. Analyse et optimisation manuelle des requêtes

Analyse de coûts de requêtes (EXPLAIN ANALYSE) :

Le moteur SQL d'une BD peut afficher le plan qu'il prévoit d'exécuter, pour une requête donnée, c'est à dire la liste des opérations qui vont être exécutées, ainsi qu'une estimation du coût de ces opérations.

Donc, à partir de Query Editor de `pgAdmin`, nous allons appliquer `EXPLAIN ANALYSE` sur la requête, qui va être utilisé pour créer la vue, afin d'estimer le temps d'exécution, puis d'essayer de changer la façon d'écrire de cette requête pour déminuer ce temps de l'exécution.

Le résulte de cette requête va être enregistrer dans une vue afin d'éviter la recalcule de SHON et refaire les jointures lorsque la réponse aux questions de ce projets, ce qui est aussi plus rapide au terme de temps d'exécution de requête.

Query Editor Query History

```
1  EXPLAIN ANALYSE SELECT classe_consommation_energie,
2                         consommation_energie,
3                         annee_construction,
4                         nom_commune,
5                         ROUND((surface_habitable/0.80)::numeric, 2) AS SHON
6                         FROM logement_description
7                         LEFT JOIN adresse ON logement_description.id_logement=adresse.id_logement
8                         LEFT JOIN surface ON logement_description.id_logement=surface.id_logement
9                         ORDER BY classe_consommation_energie
```

Selon la plane d'exécution fig(14), le liste des opérations à exécuter sont :

Tous d'abord PostgreSQL commence un parcours séquentiel de toute la table surface `Parallel Seq Scan on surface`, donc :

`cost`: Le coût estimé de démarrage (coût pour récupérer le premier enregistrement) est de 0.00 et le coût estimé total (coût pour récupérer tous les enregistrement) est de 4003.94.

`rows`=80794 Le nombre estimé de lignes rapportées, pour une taille de `width`=12 octets chacune

`loop=2` indique que l'opération de Seq Scan a été exécuté deux fois

La résulte de parcourir la table surface va être enregistre dans la table de Hashage (mémoire de Hash) au fur et à mesure de leur lecture `Parallel Hash`, Comme nous pouvons le voir par les coûts, cela ne prend pas du tout de temps.

Le même chose va être répéter pour la table adresse

Puis PostgreSQL va lire 114062 enregistrements de la table logement_discription et les compares aux enregistrements de la table d'adresse, qui sont déjà entré dans HASH table (mémoire de HASH), donc la condition de compare est dans `HASH Cond` (logement_discription.id_logement=adresse.id_logement)

La même étape est faite pour la table logement_discription et surface

En fin les résulte sont trié à l'aide `Sort key` class_consomation_energie, alors Sort Method donne des informations sur l'algorithme qui est utilisé pour tirer, ainsi si le trié est fait en mémoire ou en disk, qui affecte grandement le temps d'exécution `Exécution Time` : **3634.608 ms** le coût total de l'exécution inclut le temps lié à l'environnement du moteur d'exécution (start and stop)

Afin d'optimiser le temps d'exécution de cette requête, nous avons enlevé ORDER BY et réanalyser le coût de requêtes. Donc, nous remarquons que le temps d'exécution est diminué **2325 .744 ms** fig(15)

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Gather Merge (cost=35820.94..48938.07 rows=114062 width=56) (actual time=2750.216..3603.740 rows=193906 loops=1)
2	Workers Planned: 1
3	Workers Launched: 1
4	-> Sort (cost=34820.93..35106.08 rows=114062 width=56) (actual time=2649.755..2815.347 rows=96953 loops=2)
5	Sort Key: logement_description.classe_consommation_energie
6	Sort Method: external merge Disk: 4664kB
7	Worker 0: Sort Method: external merge Disk: 4392kB
8	-> Parallel Hash Left Join (cost=10838.27..21341.03 rows=114062 width=56) (actual time=1120.629..2207.651 rows=96953 loops=2)
9	Hash Cond: (logement_description.id_logement = surface.id_logement)
10	-> Parallel Hash Left Join (cost=5429.40..12218.70 rows=114062 width=28) (actual time=481.950..737.274 rows=96953 loops=2)
11	Hash Cond: (logement_description.id_logement = adresse.id_logement)
12	-> Parallel Seq Scan on logement_description (cost=0.00..3795.62 rows=114062 width=18) (actual time=0.037..114.327 rows=96953 loops=2)
13	-> Parallel Hash (cost=3446.62..3446.62 rows=114062 width=14) (actual time=209.914..209.915 rows=96953 loops=2)
14	Buckets: 131072 Batches: 4 Memory Usage: 3488kB
15	-> Parallel Seq Scan on adresse (cost=0.00..3446.62 rows=114062 width=14) (actual time=0.306..94.195 rows=96953 loops=2)
16	-> Parallel Hash (cost=4003.94..4003.94 rows=80794 width=12) (actual time=221.368..221.369 rows=96953 loops=2)
17	Buckets: 131072 Batches: 4 Memory Usage: 3360kB
18	-> Parallel Seq Scan on surface (cost=0.00..4003.94 rows=80794 width=12) (actual time=0.036..112.002 rows=96953 loops=2)
19	Planning Time: 0.673 ms
20	Execution Time: 3634.608 ms

Chaque Opération est identifiée par ->

Fig(14) : plan execution de requite SQL

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Hash Left Join (cost=15770.77..32165.35 rows=193906 width=56) (actual time=604.740..2299.129 rows=193906 loops=1)
2	Hash Cond: (logement_description.id_logement = adresse.id_logement)
3	-> Hash Right Join (cost=8154.89..16829.96 rows=193906 width=26) (actual time=342.605..842.058 rows=193906 loops=1)
4	Hash Cond: (surface.id_logement = logement_description.id_logement)
5	-> Seq Scan on surface (cost=0.00..5135.06 rows=193906 width=12) (actual time=0.030..94.279 rows=193906 loops=1)
6	-> Hash (cost=4594.06..4594.06 rows=193906 width=18) (actual time=340.890..340.891 rows=193906 loops=1)
7	Buckets: 65536 Batches: 4 Memory Usage: 3160kB
8	-> Seq Scan on logement_description (cost=0.00..4594.06 rows=193906 width=18) (actual time=0.045..161.779 rows=193906 loops=1)
9	-> Hash (cost=4245.06..4245.06 rows=193906 width=14) (actual time=261.497..261.498 rows=193906 loops=1)
10	Buckets: 131072 Batches: 4 Memory Usage: 3307kB
11	-> Seq Scan on adresse (cost=0.00..4245.06 rows=193906 width=14) (actual time=0.033..101.959 rows=193906 loops=1)
12	Planning Time: 0.918 ms
13	Execution Time: 2325.744 ms

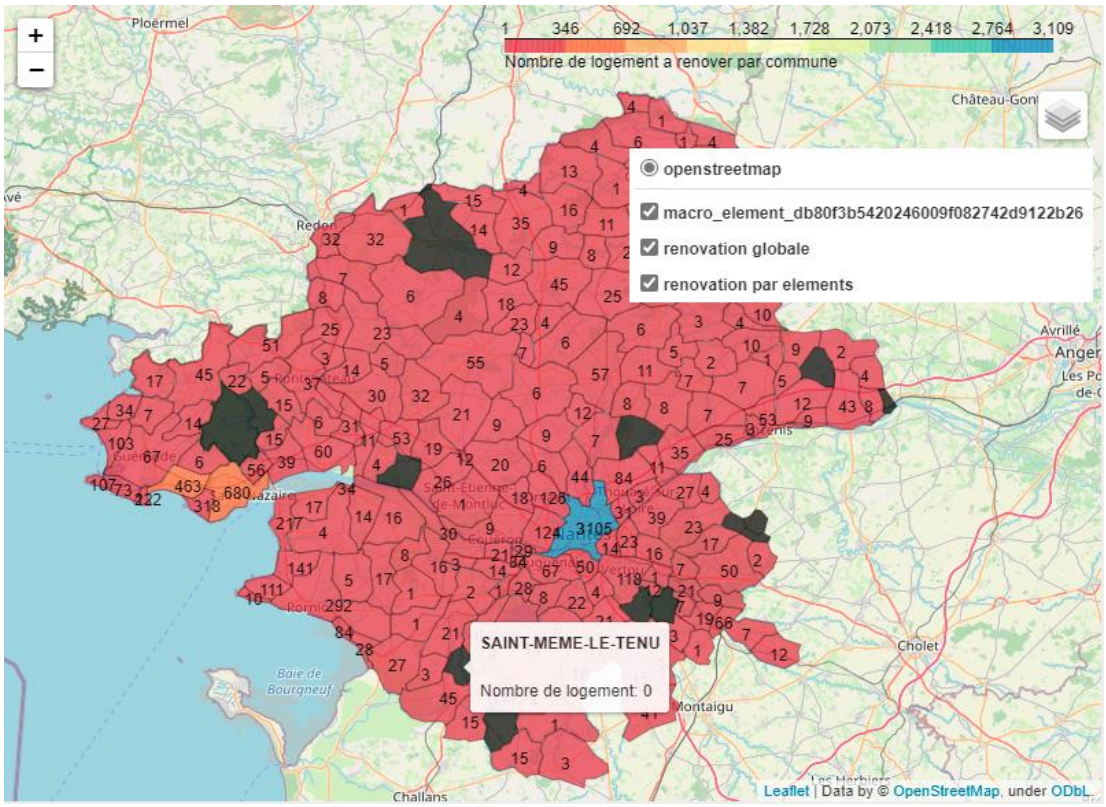
Fig(15): plan execution de rêquete SQL optimizer

3. Exposer et représenter les données visuellement :

Nous avons créé un tableau de bord pour visualiser les résulte d'analyse, avec la bibliothèque python Dash. Pour rappel, la problématique énoncée dans l'introduction de ce rapport, est de la réduction l'émission de GES des logements par rénovation, dans les différentes communes du département LOIR-ATLANTIQUE. Donc, nous avons identifié les nombres de logement à rénover dans chaque commune (classe de consommation d'énergie F, G), ainsi le type de rénovation (par élément ou rénovation globale). Puis, nous avons visualise ces résulte sur une carte en utilisant Folium (bibliothèque python). fig(16)

Dans Folium, les cartes sont interactives, ce qui signifie que nous pouvons facilement zoomer et dézoomer.

Sur cette carte, pour chaque commune, nous avons visualiser les nombres totaux de logement à rénover, il y a aussi la possibilité de filtre le résultat selon le type de rénovation : (rénovation globale, rénovation par élément)



Fig(16) : visualisation des nombre de logement à rénover

Afficher cette carte nécessite les actions suivantes fig(17) :

1. Instancier la classe `Map` en passant au constructeur le paramètre `location` qui définit le centre de la carte *(La cartographie utilisée par défaut est OpenStreetMap)* et `zoom_start` le niveau de zoom appliqué.
2. Après la création de la carte, il faut utiliser la classe `Choropleth`, pour la colorier selon le nombre de logement à rénover par commune : dont les paramètres importants sont :

   `geo_data` : les données géographiques (contour de communes) au format geojson.

   `data` : les données numériques au format Pandas représente le nombre de logement à rénover par communes. *(Doit contenir un identifiant commun avec les* informations contenues dans le GeoJSON afin de faire le jointure)

   `Columens` : Colonnes utilisées pour générer la carte choroplèthique. (nom_commune, nombre_logement)

   `key_on` : l'item dans le GeoJSON avec lequel nous ferons la jointure (nom_comm)

   La carte choroplèthique est ajouté à la carte avec la méthode `add_to`

```python
#creation la map:

m = folium.Map(location=[latitude_centrale,longitude_centrale], zoom_start=9)

folium.features.Choropleth(geo_data=url,
    data=nb_logement_a_renover_commune,
    columns=['nom_commune','nombre_logement'],
    key_on='feature.properties.nom_comm',
    fill_color='Spectral',
    fill_opacity=0.7,
    line_opacity=0.3,
    bins=9,
    weight=1,
    legend_name='Nombre de logement a renover par commune',
    dashArray='5, 3',
    highlight=True
    ).add_to(m)
```

Fig(17) : script python pour créer la carte avec foluim

Ensuit, pour filtrer les résultats selon le type de la rénovation, il faut créer deux layer, (rénovation globale, rénovation par élément), par exemple pour crée le layer rénovation globale, il faut Instancier la classe `FeatureGroupe` en passant au constructeur le paramètre `name` qui définit le nom de layer à créer, puis pour l'ajouter à la carte il faut utiliser la méthode `add_child`, Ensuit, nous avons ajouté à cette layer le nombre de logement à rénover (type globale), en utilisant layer1.adde_child et `Marker`
`Marker` sera simplement défini par ses coordonnées (latitude et longitude), dont le centre des chaque communes en étude et avec `icon` nous avons définir les informations à visualiser sur cette layer, dont le nombre de logement à rénover type globale.

```python
# ajouter layer type de renovation:

layer1=folium.FeatureGroup(name="renovation globale")
m.add_child(layer1)
for  index,row in df_type_renovation_globale.iterrows():    # ajouter tooltip nom_commune, nb logement
     layer1.add_child(folium.Marker(location=[row["longitude"],row["latitude"]],
                    icon=folium.DivIcon(
                    html=f"""<div style="font-family:Gill Sans Extrabold, sans-serif;
                                         color:black";
                                         font-weight: bold;
                                         font-size="10";>{row["numbre_logement"]}</div>""")))


layer2=folium.FeatureGroup(name="renovation par elements")
m.add_child(layer2)
for  index,row in df_type_renovation_par_element.iterrows():    # ajouter tooltip nom_commune, nb logement
     layer2.add_child(folium.Marker(location=[row["longitude"],row["latitude"]],
                     icon=folium.DivIcon(
                     html=f"""<div style="font-family:Gill Sans Extrabold, sans-serif;
                                          color:black";
                                          font-weight: bold;
                                          font-size="10";>{row["numbre_logement"]}</div>""")))

folium.LayerControl().add_to(m)
m.save("map.html")
```

Fig(18): script python pour filtre les résulte sur la carte selon le type de logement

Enfin, pour ajouter le `tooltip` fig(19) ,(nom de commune, nombre de logement à rénover), nous avons utilisé `Marker` et dans tooltip nous avons défini les infos à visualiser.

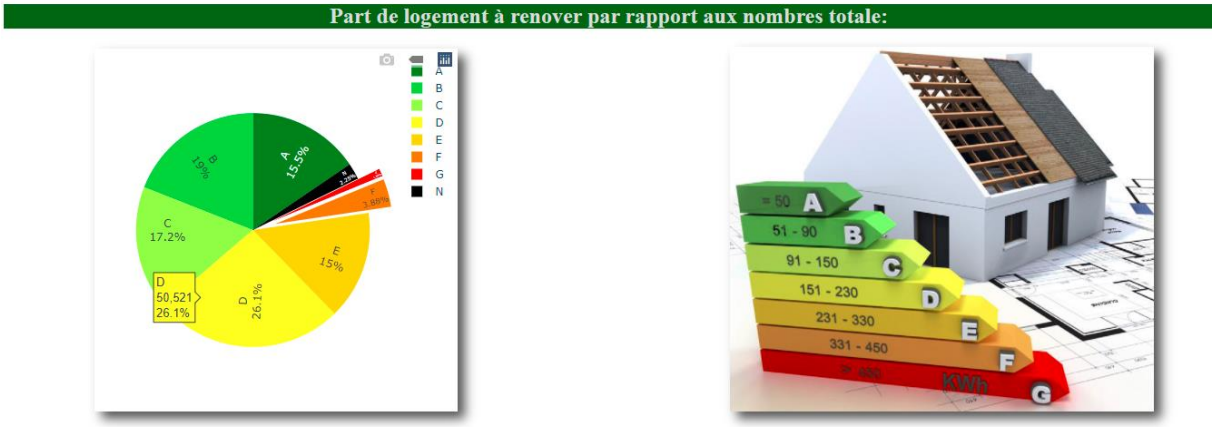
```
# ajouter tooltip nom_commune, nb logement
for  index,row in nb_logement_vis.iterrows():
     folium.Marker(location=[row["longitude"],row["latitude"]],icon=folium.DivIcon(),
     tooltip=f"""<b>{row["nom_commune"]}</b><br><br>Nombre de logement: {row["nombre_logement"]}"""
     ).add_to(m)
```

Fig(19) : ajouter une tooltip sur la carte, nombre de logement à rénover et nome de commune

Nous avons montre aussi sur le tableau de bord, à l'aide de pie chart fig(20) le nombre de logement selon les différentes class de consommation en LOIRE-ATLANTIQUE. Ce graphique va nous permettre d'estimer la situation actuelle dans ce département. Est-ce que c'est un département plutôt vert ? quelle est la part de logement à rénover par rapport au nombre totaux de logement dans ce département.

Selon ce graphique, la part de class F=3.88%, et de G=1.2%

Donc, la situation dans le département LOIRE-ATLANTIQUE conforme aux trajectoires de réduction d'émission GES défini.



Fig(20) : Répartition des nombres de logement selon les différente classe de consomation d'energie.

```
import plotly.graph_objects as go

color_discrete_map=("green","#0CD220","#ADFF2F","yellow","#FFD700","#F2820D","red","black")

fig = go.Figure(
    data=[go.Pie(
        labels=nb_logement_class['classe_consommation_energie'],
        values=nb_logement_class['nombre_logement'],
        textinfo='percent+label',
        marker_colors=color_discrete_map,
        insidetextorientation='radial',
        textposition='inside',
        #  make sure that Plotly won't reorder your data while plotting
        sort=False)])

fig.update_traces(hoverinfo='label+value', textinfo='label+percent', textfont_size=12)
fig.update_layout(
    autosize=False,
    width=400,
    height=400,
    margin=dict(
        l=30,
        r=30,
        b=10,
        t=10),
    paper_bgcolor="white",
    title_font_family="Times New Roman")
```

Fig(21) : script python pour créer le pie charte

Pour produire ce graphique fig(21), Nous avons importé le module `graph_objects` de la bibliothèque python `plotly`, donc nous avons utilisé la classe `Figure` et la class `Pie`. Nous devons donner minimalement en argument à la classe Pie les éléments suivants :

`Labels :` Définit les étiquettes de secteur, la colonne class_consomation_energie de pandas data frame nb_logement_class

`Values` : Définir les valeurs associées aux secteurs, la colonne nombre_logement pandas data frame nb_logement_class

`Marker_colors` : séquence de colore utiliser pour colorier les différents secteurs de pie chart, donc nous avons défini la tuple color_discrete_map, les nombres d'élément de cette tuple doit corresponde aux nombres d'élément de la colonne nombre_logement utilisé pour définir les secteurs.

Les sources utilisées dans la recherche de solutions :

[1] https://www.data.gouv.fr/fr/datasets/diagnostics-de-performance-energetique-pour-les-logements-par-habitation/

[2] https://public.opendatasoft.com/explore/dataset/correspondance-code-insee-code-postal/export/?refine.nom_dept=LOIRE-ATLANTIQUE

[3] Présentation - Le site "www.RT-bâtiment.fr" devient le site "RT-RE-bâtiment" (rt-batiment.fr)

[4] **http://www.carbone4.com/article-batiment-snbc/**

[5] https://ineumann.developpez.com/tutoriels/merise/initiation-merise/

[6] os.path — manipulation courante des chemins — Documentation Python 3.9.5

[7] PostgreSQL: Documentation: 9.5: information_schema_catalog_name

[8] 5 different ways to backup your PostgreSQL database using Python | by Simon-Pierre Gingras | poka-techblog | Medium

[9] Reading a Postgres EXPLAIN ANALYZE Query Plan (thoughtbot.com)

[10] Géolocalisation — Documentation Cours Python (esiee.fr)

[11] https://stackoverflow.com/

[12] plugins — Folium 0.12.1 documentation (python-visualization.github.io)

[13] https://plotly.com/python/choropleth-maps/

[14] Folium — Folium 0.12.1 documentation (python-visualization.github.io)

[15] Develop Data Visualization Interfaces in Python With Dash – Real Python
SQL vs noSQL : Quelles différences ? - GETC

CONCLUSION

L'objectif de ce travail était de répondre au problématique de la réduction d'émission des gaz à effet de serre dans le secteur du bâtiment, appliquée à un travail technique lié à la data, dans le but de concevoir et développer une base de données, afin de mettre en évidence des analyses et résultats liés à ce sujet.

La réponse à ce problématique passe par la rénovation du logement existant, Donc, nous avons :

- ✓ Identifier les logements à rénover, en appuyant sur le DPE (diagnostic de performance énergétique).
- ✓ Identifier le type de rénovation en appuyant sur le RT (réglementation thermique).

L'analyse était applique sur les données du département LOIRE-ATLANTIQUE
Nous avons utilisé la langage python pour réaliser ce Project. Nous avons affiché les résulte d'analyse sur un tableau de bord.

Donc, sur une carte nous avons affiché les nombres de logement à rénover par commune avec la possibilité de filtrer les résultats selon le type de rénovation.

Nous avons affiché aussi, dans un tableau, selon un nom de commune, des détaille sur les logements à rénover (adresse, surface, état d'avancement …).

L'objectif fixé est bien atteint, afin de répondre mieux à ce problématique, il sera bien d'élargir ce travail, en important des données détaille sur les huit points de rénovation par éléments définir par RT.

ANNEXE

1. Utils.py :

```
import pandas
import geopandas
import os
import psycopg2
from psycopg2 import sql
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
import requests
import numpy as np
import pymongo
from pymongo import MongoClient,GEOSPHERE
import shapely.geometry
from statistics import *
import folium
import plotly.express as px
import plotly.graph_objects as go


def create_database(db_name,utilisateur, mot_passe, host='localhost', port=5432):
    """Creation base de donne postgresql
    """
    try:
        conn = psycopg2.connect(user=utilisateur, password=mot_passe, host=host, port=5432)
        conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
        cursor = conn.cursor()
        cursor.execute("SELECT datname FROM pg_database")
        nom_db_exist = cursor.fetchall()
        if (f"{db_name}",) in nom_db_exist:
            print("La base de données existe")
        else:
            cursor.execute(f"CREATE DATABASE {db_name}")
            print("La base de données a été crée avec succès")

    except psycopg2.Error as e:
        print("Erreur lors de la connection et la creation de la base de données")
        print(e)
        return None
    cursor.close()
    conn.close()
    return True

def ouvrir_connection(nom_bdd, utilisateur, mot_passe, host='localhost', port=5432):
    """Se connecter à une source de données PostgreSQL
    """
    try:
        conn = psycopg2.connect(dbname=nom_bdd, user=utilisateur, password=mot_passe, host=host, port=5432)
    except psycopg2.Error as e:
        print("Erreur lors de la connection à la base de données")
        print(e)
        return None
    conn.set_session(autocommit=True)
    return conn


def insert_collectionMongoDB(db_name,collection_name,nom_departement):
    """ En mongodB, la base de donnees se créer lors de la première sauvegarde
        de la valeur dans la collection définie
        nom_departement: pour récuperer des donnees geojson d'un departement apartir d'api
    """
    try:
        myclient = pymongo.MongoClient("mongodb://localhost:27017/")                # connecter mongodb
        bd_nosql=myclient[db_name]                                                  # connecter au bas de donnees
        collection_list = bd_nosql.list_collection_names()                          # verifie l'existance de la collection avant de crée
        if collection_name in collection_list:
            print("La collection existe")
        else:
            geojson_donnees='https://public.opendatasoft.com/explore/dataset/correspondance-code-insee-code-postal'+\
                            f'/download/?format=geojson&refine.nom_dept={nom_departement}&timezone=Europe/Berlin&lang=fr'
```

```python
def import_donnees(code_departement):
    """ récupere les données a partire de API koumoul.com et l'enregistre en local dans le fichier data_brute
    """
    try:
        one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))
        data_path = os.path.join(one_directory_up_path, 'data','data_brute')
        csv_file = open(os.path.join(data_path,f'{code_departement}.csv'), 'wb')
        url=f'https://koumoul.com/s/data-fair/api/v1/datasets/dpe-{code_departement}/data-files/dpe-{code_departement}.csv'
        req = requests.get(url)
        url_content = req.content
        csv_file.write(url_content)
        csv_file.close()
    except import_donnees.Error as e:
        print("Erreur lors de l'importation des données")
        print(e)
        return
    print("Les données importées et sauvegardés  avec succès")


#########################################################################################################


def lire_donnees_en_df(nom_fichier,nom_dossier):
    """lire les données csv en forme de pandas dataframe
    """
    one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))
    data_path = os.path.join(one_directory_up_path, 'data',f'{nom_dossier}')
    donnees_csv=pandas.read_csv(os.path.join(data_path,f'{nom_fichier}.csv'),sep=",",low_memory=False)
    return donnees_csv


def lire_geo_collection_MongoDb(nom_db,nom_collection):
    """cette fonction va lire une geo collection apartir de MongoDB
       il va supprimer l'objetId creer par MongoDB lors que l'insertion
       de la collection. il va retourner une dictionnaire
    """
    client = pymongo.MongoClient("mongodb://localhost:27017/")
    db = client[f'{nom_db}']
    col = db[f'{nom_collection}']
    geodonnees= col.find_one()
    geodonnees.pop('_id', None)
    return(geodonnees)



#########################################################
#Nettoyage des donnees:
# les fonction à appeler successivement pour nettoyer les donnees:

def nettoyage_error(donnees_df):
    """remplacer la @ par la lettre A dans le colonne numero_dpe
       remplase zero value en categorie_erp par NoERP
    """
    try:
        donnees_df['numero_dpe'] = donnees_df['numero_dpe'].str.replace('@','A')
        donnees_df=donnees_df.fillna(0)
        donnees_df['tr012_categorie_erp_categorie'] = donnees_df['tr012_categorie_erp_categorie'].replace(0,'NoERP')

    except:
         print("Erreur lors le corrige des données")

    print("Les error sont corriges avec succès")
    return donnees_df
```

```python
def nettoyage_drop_duplicates(donnees_df):
    """supprimer les duplicates selon le numero de dpe
    """
    try:
        donnees_df=donnees_df.drop_duplicates(subset =["numero_dpe"],
                                              keep = 'first')
    except:
         print("Erreur lors que la suppression des duplicates")

    print("Les duplicates sont supprimer avec succès")
    return donnees_df


def nettoyage_suprim_0value(donnees_df):
    """ supprimer les lignes lors que la consomation d'energie est
        zéro ou non fournie
    """
    try:
         supprimer_consommation_energie0=donnees_df.drop(donnees_df[donnees_df['consommation_energie']==0.00].index)

         donnees_df=supprimer_consommation_energie0.dropna(subset = ["consommation_energie","estimation_ges"])
    except:
         print("Erreur lors la suppression des zeros values")

    print("Les zeros values sont supprimer avec succès")
    return donnees_df


def nettoyage_suprim_col(donnees_df,colonne_supprimer):
    """supprimer les colonnes qui n'sont pas utiliser
       pour l'analyse
    """
    try:
        donnees_df=donnees_df.drop(columns=colonne_supprimer,errors='ignore').fillna(0)
    except:
         print("Erreur lors la suppression des colonnes")

    print("Les colonnes sont supprimes avec succès")
    return donnees_df


def nettoyage_suprim_col(donnees_df,colonne_supprimer):
    """supprimer les colonnes qui n'sont pas utiliser
       pour l'analyse
    """
    try:
        donnees_df=donnees_df.drop(columns=colonne_supprimer,errors='ignore').fillna(0)
    except:
         print("Erreur lors la suppression des colonnes")

    print("Les colonnes sont supprimes avec succès")
    return donnees_df


def nettoyage_to_numeric(donnees_df,donnees_numeric):
    """transformer les données des colonnes définie dans le list
       donnée_numeric to numeric
    """
    try:
        donnees_df[donnees_numeric]=donnees_df[donnees_numeric].apply(pandas.to_numeric,errors='coerce')    #invalid parsing will be set as NaN, retourn type float
        donnees_df[donnees_numeric]=donnees_df[donnees_numeric].fillna(0)                                   # remplacer Nan par 0 afin de transformer en int apres
        donnees_df[donnees_numeric]=donnees_df[donnees_numeric].applymap(np.int64)                          # int ne s'applique pas lors que il y a des NaA
    except:
         print("Erreur lors la transformation en numeric")

    print("Les données sont transforme en numeric avec succès")
    return donnees_df
```

```
def nettoyage_to_boolean(donnees_df,donnees_boolean):
    """transformer les données des colonnes définie dans le list
       donnees_boolean to true, false
       dans le fichier csv importe les données boolean est
       presenté en 0,1 mais le boolean en postgresql est true, fals
    """
    try:
        donnees_df[donnees_boolean]=donnees_df[donnees_boolean].replace([0,1],["f","t"])
    except:
         print("Erreur lors la transformation en boolean")

    print("Les données sont transforme en boolean avec succès")
    return donnees_df

def nettoyage_suprim_commune(donnees_df,df_commune):
    """supprimer les communes n'appartien pas au
       departement LOIRE-ATLANTIQUE
    """
    try:
        donnees_df['commune']=donnees_df['commune'].str.upper()
        donnees_df=donnees_df[donnees_df['commune'].isin(df_commune['nom_commune'])]
    except:
         print("Erreur lors la supprsion des communes")

    print("Les commune n'appartien pas au departement sont supprime avec succès")
    return donnees_df


def nettoyage_adress(donnees_df):
    """ les donnees des adress sont concatene pour les plus part des lignes
        dans la colonne nom_rue, cette fonction va separer ces dpnnees en
        numero_rue, nom_rue, type_voie
    """
    try:
        donnees_df[["numero_rue","nom_rue","type_voie"]]=donnees_df[["numero_rue","nom_rue","type_voie"]].astype(str)
        donnees_df['Full_adress']=np.where(donnees_df['type_voie']!="0",donnees_df['type_voie']+" "+donnees_df['nom_rue'],donnees_df['nom_rue'])
        donnees_df['Full_adress']=np.where(donnees_df['numero_rue']!="0",donnees_df['numero_rue']+" "+donnees_df['Full_adress'],donnees_df['Full_adress'])
        donnees_df['Full_adress']=donnees_df['Full_adress'].str.upper()
        # extraie les nombre de la colonne full adress a l'aide de findall et les mettre dans le numero du rue
        donnees_df['numero_rue']=donnees_df['Full_adress'].str.findall(r'-?\d+\.?\d*').apply(lambda x : ', '.join(x))
        # definir un pattern les diffferent type de voie:
        pattern=r'RUE|CHEMIN|ROUTE|AVENUE|CHAUSSEE|ALLEE|PASSAGE|BD|IMPASSE|RESIDENCE|BOULEVARD|BIS RUE|TER RUE|LOTISSEMENT|QUAI|SQUARE|LOT|AV'
        donnees_df['type_voie']=donnees_df['Full_adress'].str.findall(pattern,flags=re.IGNORECASE).apply(lambda x : ', '.join(x))    # apply lambda pour transformer
        # a l'aide de replace et re on va supprimer de la nom de rue le numero, le type et les articl le les...
        donnees_df["nom_rue"]=donnees_df['Full_adress'].str.replace(pattern,'')
        donnees_df["nom_rue"]=donnees_df["nom_rue"].str.replace(r'-?\d+\.?\d*','')
        donnees_df["nom_rue"]=donnees_df["nom_rue"].str.replace(r'DES|LES|DE|LE|,','')
    except:
          print("Erreur lors la separation  des donnees de la colonne adresse")

    print("La separation des donnees de la colonne adress est fait avec succès")
    return donnees_df


def insertion_donnees(conn, sql_insertion_table, donnees_df):
    """insertion des donnees
    """
    try:
        cursor = conn.cursor()
        for index,row in donnees_df.iterrows():
            cursor.execute(sql_insertion_table, row)
```

```
def commune(geodonnees):
    """geodonnees a recuprer a partitr de MongoDB en utilisant la fonction
       lire_geo_collection_MongoDb
    """
    x=len(geodonnees['features'])
    comm_df=pandas.DataFrame()
    for i in range(0,x-1):
        commune_donnees=[
            geodonnees['features'][i]['properties']['nom_comm'],
            geodonnees['features'][i]['properties']['insee_com'],
            geodonnees['features'][i]['properties']['postal_code'],
            geodonnees['features'][i]['properties']['code_arr'],
            geodonnees['features'][i]['properties']['geo_point_2d'][0],
            geodonnees['features'][i]['properties']['geo_point_2d'][1]]
        comm_df=comm_df.append([commune_donnees],ignore_index=True)
    comm_df.columns=["nom_commune","code_insee","code_postal","code_arrondissement","longitude","latitude"]
    return(comm_df)


# Fonction permettant d'exécuter un requête SQL sur une BDD définie par sa connexion conn
def executer_requete(requete_sql, conn):
    try:
        cursor = conn.cursor()
        cursor.execute(requete_sql)
        conn.commit()
    except psycopg2.Error as e:
        print("Erreur lors de l'execution de la requête")
        print(e)
        return
    cursor.close()



def sauvgarder_donnees_en_csv(donnees_df,nom_fichier,nom_dossier):
    """sauvgarder les donnees netoyer en local
    """
    try:
        one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))

        data_path = os.path.join(one_directory_up_path, 'data',f'{nom_dossier}')

        donnees_df.to_csv (os.path.join(data_path,f'{nom_fichier}.csv'), index = False, header=True)          # enregistre les données nettoyer en format csv
    except:
          print("Erreur lors le sauvgardage des données")
          return
    print("Les données sont sauvgardes avec succès")


def get_file_metadata(nom_dossier, filename, metadata_filename,metadata):
    """
    filename: nome de fichier pour lequele on va récuperer le metadonnées,il doit contenir une extension, i.e. "PID manual.pdf"
    nom_dossier: nom de dossier ou se trove le fichier pour lequele on va récuperer le metadonnées
    metadata_filename: nom de fichier metadonnée a retourné
    cette fonction returne une dictionnaire contien les metadonnées
    """
    try:
        one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))
        data_path = os.path.join(one_directory_up_path, 'data',f'{nom_dossier}')
        sh = win32com.client.gencache.EnsureDispatch('Shell.Application', 0)
        ns = sh.NameSpace(data_path)
        # Enumeration is necessary because ns.GetDetailsOf only accepts an integer as 2nd argument
        file_metadata = dict()
        item = ns.ParseName(str(filename))
        for ind, attribute in enumerate(metadata):
            attr_value = ns.GetDetailsOf(item, ind)
            if attr_value:
                file_metadata[attribute] = attr_value
        data_path_save = os.path.join(one_directory_up_path, 'data','metadonnees')
        with open(os.path.join(data_path_save,f'{metadata_filename}.json'),'w') as fp:
           json.dump(file_metadata, fp)
    except:
        print("error lors que la récuperation des metadonnées")
    print("les metadonnées sont récuperer avec succées")
    return True
```

2. creation_bd_tables.py :

```
from utils import *



# creation base de donnee DPE_logement

utilisateur = "postgres"
mot_passe = os.environ.get('pg_psw')
nom_bdd="dpe_logement"
conn=ouvrir_connection(nom_bdd, utilisateur, mot_passe, host='localhost', port=5432)

create_database(db_name,utilisateur, mot_passe, host='localhost', port=5432)



# Creation tables:


def create_table(conn):

    """Creation des tables
    """
    try:
        cursor = conn.cursor()

        # creation de la table Categorie ERP:
        cursor.execute("""CREATE TABLE IF NOT EXISTS categorie_erp(
                       categorie_erp VARCHAR PRIMARY KEY,
                       groupe VARCHAR,
                       code VARCHAR)
                       """)
        # creation de la table type ERP:
        cursor.execute("""CREATE TABLE IF NOT EXISTS type_erp(
                      id_type_erp INTEGER PRIMARY KEY,
                      categorie_type VARCHAR,
                      type_erp_type VARCHAR,
                      categorie_erp VARCHAR,
                      FOREIGN KEY (categorie_erp) REFERENCES categorie_erp(categorie_erp))
                      """)

      # creation de la table commune:
      cursor.execute("""CREATE TABLE IF NOT EXISTS commune(
                    nom_commune VARCHAR PRIMARY KEY,
                    arrondissement_Id INTEGER,
                    code_postal VARCHAR
                    )
                    """)
      # creation de la table adresse:
      cursor.execute("""CREATE TABLE IF NOT EXISTS adresse(
                    id_adresse SERIAL PRIMARY KEY,
                    numero_rue VARCHAR,
                    nom_rue VARCHAR,
                    type_voie VARCHAR,
                    nom_commune VARCHAR,
                    escalier VARCHAR,
                    etage VARCHAR,
                    Porte_N VARCHAR,
                    FOREIGN KEY (nom_commune) REFERENCES commune(nom_commune))
                    """)
      # creation de la table Information sur diagnostique DPE:
      cursor.execute("""CREATE TABLE IF NOT EXISTS dpe(
                    numero_dpe VARCHAR PRIMARY KEY,
                    date_de_réception_dpe_par_le_système DATE,
                    date_visite_diagnostiqueur DATE,
                    date_etablissement_dpe DATE,
                    date_arrete_tarifs_energies DATE,
                    commentaires_ameliorations_recommandations TEXT,
                    explication_personnalisee TEXT
                    )
                    """)
```

```
    # creation de la table type_bâtiment:
    cursor.execute("""CREATE TABLE IF NOT EXISTS type_bâtiment(
                id_type_bâtiment INTEGER PRIMARY KEY,
                code VARCHAR,
                discription VARCHAR,
                libelle VARCHAR,
                secteur_activite VARCHAR,
                categorie_erp VARCHAR,
                FOREIGN KEY (categorie_erp) REFERENCES categorie_erp(categorie_erp))
                """)

    # creation de la table commune:
    cursor.execute("""CREATE TABLE IF NOT EXISTS commune(
                nom_commune VARCHAR PRIMARY KEY,
                arrondissement_Id INTEGER,
                code_postal VARCHAR
                )
                """)
    # creation de la table adresse:
    cursor.execute("""CREATE TABLE IF NOT EXISTS adresse(
                id_adresse SERIAL PRIMARY KEY,
                numero_rue VARCHAR,
                nom_rue VARCHAR,
                type_voie VARCHAR,
                nom_commune VARCHAR,
                escalier VARCHAR,
                etage VARCHAR,
                Porte_N VARCHAR,
                FOREIGN KEY (nom_commune) REFERENCES commune(nom_commune))
                """)
    # creation de la table Information sur diagnostique DPE:
    cursor.execute("""CREATE TABLE IF NOT EXISTS dpe(
                numero_dpe VARCHAR PRIMARY KEY,
                date_de_réception_dpe_par_le_système DATE,
                date_visite_diagnostiqueur DATE,
                date_etablissement_dpe DATE,
                date_arrete_tarifs_energies DATE,
                commentaires_ameliorations_recommandations TEXT,
                explication_personnalisee TEXT
                )
                """)
     # creation de la table type_bâtiment:
    cursor.execute("""CREATE TABLE IF NOT EXISTS type_bâtiment(
                id_type_bâtiment INTEGER PRIMARY KEY,
                code VARCHAR,
                discription VARCHAR,
                libelle VARCHAR,
                secteur_activite VARCHAR,
                categorie_erp VARCHAR,
                FOREIGN KEY (categorie_erp) REFERENCES categorie_erp(categorie_erp))
                """)
```

```
        # creation de la table Logement description:
        cursor.execute("""CREATE TABLE IF NOT EXISTS logement_description(
                    id_logement INTEGER PRIMARY KEY,
                    id_type_bâtiment INTEGER,
                    numero_dpe VARCHAR,
                    classement_ges VARCHAR,
                    classe_consommation_energie VARCHAR,
                    annee_construction INTEGER,
                    consommation_energie FLOAT,
                    estimation_ges FLOAT,
                    nombre_niveaux INTEGER,
                    nombre_entrees_avec_sas INTEGER,
                    nombre_entrees_sans_sas INTEGER,
                    en_souterrain BOOLEAN,
                    en_surface BOOLEAN,
                    nombre_boutiques INTEGER,
                    numero_lot VARCHAR,
                    id_adresse INTEGER,
                    etat_avancement TEXT,
                    FOREIGN KEY (Id_type_bâtiment) REFERENCES type_bâtiment(Id_type_bâtiment),
                    FOREIGN KEY (id_adresse) REFERENCES adresse(id_adresse),
                    FOREIGN KEY (numero_dpe) REFERENCES dpe(numero_dpe)
                    )
                    """)
        # creation de la table vitrage:
        cursor.execute("""CREATE TABLE IF NOT EXISTS vitrage(
                    id_presence_verriere SERIAL PRIMARY KEY,
                    presence_verriere BOOLEAN,
                    surface_verriere FLOAT,
                    type_vitrage_verriere VARCHAR,
                    id_logement INTEGER,
                    FOREIGN KEY (Id_logement) REFERENCES logement_description(Id_logement)
                    )
                    """)
    # creation de la table Surface:
    cursor.execute("""CREATE TABLE IF NOT EXISTS surface(
                id_Surface SERIAL PRIMARY KEY,
                id_logement INTEGER,
                surface_habitable FLOAT,
                surface_thermique_lot FLOAT,
                Surface_commerciale FLOAT,
                surface_thermique_parties_communes FLOAT,
                surface_utile FLOAT,
                shon FLOAT,
                surface_baies_orientees_sud FLOAT,
                surface_baies_orientees_nord FLOAT,
                surface_baies_orientees_est_ouest FLOAT,
                surface_planchers_hauts_deperditifs FLOAT,
                surface_planchers_bas_deperditifs FLOAT,
                surface_parois_verticales_opaques_deperditives FLOAT,
                FOREIGN KEY (Id_logement) REFERENCES logement_description(Id_logement)
                )
                """)


    conn.commit()

except psycopg2.Error as e:
    print("Erreur lors de la création de la table")
    print(e)
    return
cursor.close()
conn.close()
print("Les tables ont été créer avec succès")
return True
```

3. importer_nettoyer_inserer.py :

```
from utils import *


"""
Dans ce fichier, nous allons:
1- importer des donnees en forma csv a partie d'api et les suavgarder en local dans le dossier data_brut
2- Nettoyer les donnees et les sauvgarder en local dans le dossier data_nettoyer
3- inserer les données dans la base de données dpe_logement
4- Créer un repertoir de metadonnees pour les fichiers CSV et la base de donnees
"""



#################
# 1- importer des donnees:
utilisateur = "postgres"
mot_passe = os.environ.get('pg_psw')
nom_bdd="dpe_logement"
conn=ouvrir_connection(nom_bdd, utilisateur, mot_passe, host='localhost', port=5432)


code_departement="44"
nom_departement="LOIRE-ATLANTIQUE"
#import_donnees(code_departement)              # importer les données a partir de API et sauvgarder en locale


################################################
# 2- Nettoyage des donnees:

nom_fichier="44"
nom_dossier="data_brute"

donnees_df=lire_donnees_en_df(nom_fichier,nom_dossier)   # Lire des données en pandas df



colonne_supprimer=["usr_diagnostiqueur_id","usr_logiciel_id","tr001_modele_dpe_id","nom_methode_dpe",
                   "version_methode_dpe","nom_methode_etude_thermique","version_methode_etude_thermique",
                   "tv016_departement_id","adresse_proprietaire","adresse_proprietaire_installations_communes",
                   "nombre_circulations_verticales","est_efface","latitude","longitude","geo_score","geo_type",
                   "geo_adresse","geo_id","geo_id","geo_l4","geo_l5","tr001_modele_dpe_type_id",
                   "tr001_modele_dpe_code","tr001_modele_dpe_modele","tr001_modele_dpe_type_libelle",
                   "tr001_modele_dpe_description","tr001_modele_dpe_fichier_vierge","tr001_modele_dpe_est_efface",
                    "tr001_modele_dpe_type","tr001_modele_dpe_type_ordre","tr013_type_erp_est_efface",
                   "tv016_departement_id","tv016_departement_code","tv017_zone_hiver_id","tv017_zone_hiver_code",
                   "tv017_zone_hiver_t_ext_moyen","tv017_zone_hiver_peta_cw","tv017_zone_hiver_dh14","tv017_zone_hiver_prs1",
                   "tv018_zone_ete_id","tv018_zone_ete_code","tv018_zone_ete_sclim_inf_150","tv018_zone_ete_sclim_sup_150",
                   "tv018_zone_ete_rclim_autres_etages","tv018_zone_ete_rclim_dernier_etage","tv016_departement_departement",
                   "tv016_departement_altmin","tv016_departement_altmax","tv016_departement_nref","tv016_departement_dhref",
                   "tv016_departement_pref","tv016_departement_c2","tv016_departement_c3","tv016_departement_c4",
                    "tv016_departement_t_ext_basse","tv016_departement_e","tv016_departement_fecs_solaire_m_i",
                    "tv016_departement_fecs_recente_i_c","tv016_departement_fecs_ancienne_m_i","tv016_departement_fecs_recente_m_i",
                   "tv016_departement_fch","tv016_departement_fecs_ancienne_i_c","tv016_departement_fecs_recente_i_c"]

donnees_boolean=["dpe_vierge","en_souterrain","en_surface","presence_verriere"]
donnees_numeric=["arrondissement","numero_rue","batiment","code_postal","code_insee_commune","code_insee_commune_actualise",
                 "tr013_type_erp_code"]


donnees_df=nettoyage_error(donnees_df)

donnees_df=nettoyage_drop_duplicates(donnees_df)

donnees_df=nettoyage_suprim_0value(donnees_df)

donnees_df=nettoyage_suprim_col(donnees_df,colonne_supprimer)

donnees_df=nettoyage_to_numeric(donnees_df,donnees_numeric)

donnees_df=nettoyage_to_boolean(donnees_df,donnees_boolean)

df_commune=commune(nom_departement)

donnees_df=nettoyage_suprim_commune(donnees_df,df_commune)

donnees_df=nettoyage_adress(donnees_df)
```

```
nom_fichier='data_nettoyee_44'
nom_dossier='data_nettoyee'

sauvgarder_donnees_en_csv(donnees_df,nom_fichier,nom_dossier)

#########################################################################

# 3- inserer les données:

conn=ouvrir_connection(nom_bdd, utilisateur, mot_passe, host='localhost', port=5432)

nom_fichier='data_nettoyee_44'
nom_dossier='data_nettoyee'

# lire les données nettoyer a partir du dossier data_nettoyer en pandas data frame
df_donnee_nettoyer=lire_donnees_en_df(nom_fichier,nom_dossier)


"""
donnee demo: exrtai des données afin de faire le test
donnee_demo= df_donnee_nettoyer.iloc[100:120,:]
sauvgarder_donnees_en_csv(donnee_demo,'donnees_demo','data_demo')
"""

df_donnee_nettoyer=df_donnee_nettoyer.drop(df_donnee_nettoyer.index[100:120],0)


# insertion des données dans la table categorie erp:
# a partir de dictionnaire des donnéées presente sur le sit ADEME ou nous avons importer les données
# nous avons crééer le df categorie erp:
data_erp = {'tr012_categorie_erp_categorie':['1ère Catégorie','2ème Catégorie' ,'3ème Catégorie','4ème Catégorie','5ème Catégorie','NoERP'],
           'tr012_categorie_erp_groupe':['1er Groupe', '1er Groupe','1er Groupe','1er Groupe','2ème Groupe','0'],
           'tr012_categorie_erp_code':['TR012_001','TR012_002','TR012_003','TR012_004','TR012_005','0']
       }

categorie_erp= pandas.DataFrame(data_erp, columns = ['tr012_categorie_erp_categorie','tr012_categorie_erp_groupe','tr012_categorie_erp_code'])



sql_insertion_categorie_erp="""INSERT INTO categorie_erp
                           (categorie_erp,
                           groupe,code)
                           VALUES (%(tr012_categorie_erp_categorie)s,
                           %(tr012_categorie_erp_groupe)s,
                           %(tr012_categorie_erp_code)s);
                           """
#insertion_donnees(conn,sql_insertion_categorie_erp,categorie_erp)


# insertion les donnees dans la table type ERP:
# cette table est statique, contient cinq type erp, pour faciliter l'insertion
# j'ai extrait ces donnes en utilisant pandas df et dropduplicate:

def type_erp(df_donnee_nettoyer):
    type_erp=df_donnee_nettoyer[["tr013_type_erp_categorie_id","tr013_type_erp_code","tr013_type_erp_categorie","tr013_type_erp_type"]]
    type_erp=type_erp.drop_duplicates(["tr013_type_erp_categorie_id"])
    type_erp=type_erp.sort_values(by=['tr013_type_erp_categorie_id'])
    type_erp=type_erp.assign(categorie_erp='5ème Catégorie')
    return(type_erp)



sql_insertion_type_erp="""INSERT INTO type_erp
                      (id_type_erp,
                      categorie_type,
                      type_erp_type,
                      categorie_erp)
                      VALUES (
                      %(tr013_type_erp_categorie_id)s,
                      %(tr013_type_erp_categorie)s,
                      %(tr013_type_erp_type)s,
                      %(categorie_erp)s);
                      """

#type_erp=type_erp(df_donnee_nettoyer)
#insertion_donnees(conn,sql_insertion_type_erp,type_erp)
```

```
# insertion les donnees dans la table type_bâtiment:

sql_insertion_type_bâtiment="""INSERT INTO type_bâtiment
                            (id_type_bâtiment,
                            code,
                            discription,
                            libelle,
                            secteur_activite,
                            categorie_erp)
                            VALUES (
                            %(tr002_type_batiment_id)s,
                            %(tr002_type_batiment_code)s,
                            %(tr002_type_batiment_description)s,
                            %(tr002_type_batiment_libelle)s,
                            %(secteur_activite)s,
                            %(tr012_categorie_erp_categorie)s);
                            """

# type de batiment est une table statique contient quelque lignes,
# le plus facil est d'extrait ses ligne avec pandas et dropduplicate:

def type_batiment(df_donnee_nettoyer):
    type_batiment=df_donnee_nettoyer[["tr002_type_batiment_id","tr013_type_erp_code","tr002_type_batiment_code",
                                      "tr002_type_batiment_description","tr002_type_batiment_libelle",
                                      "secteur_activite","tr002_type_batiment_ordre","tr002_type_batiment_simulateur",
                                      "tr012_categorie_erp_categorie"]]
    type_batiment=type_batiment.drop_duplicates(["tr002_type_batiment_id"])
    type_batiment=type_batiment.sort_values(by=['tr002_type_batiment_id'])
    return(type_batiment)


#type_batiment=type_batiment(df_donnee_nettoyer)
#insertion_donnees(conn,sql_insertion_type_bâtiment,type_batiment)



# insertion les donnees dans la table info_diagnostique_dpe:

sql_insertion_info_diagnostique_dpe="""INSERT INTO dpe
                                    (numero_dpe,
                                    date_de_réception_dpe_par_le_système,
                                    date_visite_diagnostiqueur,
                                    date_etablissement_dpe,
                                    date_arrete_tarifs_energies,
                                    commentaires_ameliorations_recommandations,
                                    explication_personnalisee)
                                    VALUES (
                                    %(numero_dpe)s,
                                    %(date_reception_dpe)s,
                                    %(date_visite_diagnostiqueur)s,
                                    %(date_etablissement_dpe)s,
                                    %(date_arrete_tarifs_energies)s,
                                    %(commentaires_ameliorations_recommandations)s,
                                    %(explication_personnalisee)s);
                                    """

#insertion_donnees(conn,sql_insertion_info_diagnostique_dpe,df_donnee_nettoyer)



 # insertion les donnees dans la table commune:

 sql_insertion_commune="""INSERT INTO commune
                       (nom_commune,
                       arrondissement_Id,
                       code_postal
                       )
                       VALUES (
                       %(nom_commune)s,
                       %(code_arrondissement)s,
                       %(code_postal)s
                       );
                       """
 nom_departement='LOIRE-ATLANTIQUE'
 #commune_df=commune(nom_departement)

 #insertion_donnees(conn,sql_insertion_commune,commune_df)
```

```
# insertion les donnees dans la adresse:

sql_insertion_adresse="""INSERT INTO adresse
                        (numero_rue,
                        nom_rue,
                        type_voie,
                        nom_commune,
                        escalier,
                        etage,
                        Porte_N)
                        VALUES (
                        %(numero_rue)s,
                        %(nom_rue)s,
                        %(type_voie)s,
                        %(commune)s,
                        %(escalier)s,
                        %(etage)s,
                        %(porte)s);
                        """

#insertion_donnees(conn,sql_insertion_adresse,df_donnee_nettoyer)

 insertion les donnees dans la table logement:

ql_insertion_Logement_description="""INSERT INTO logement_description
                                    (Id_logement,
                                    id_type_bâtiment,
                                    numero_dpe,
                                    classement_ges,
                                    classe_consommation_energie,
                                    annee_construction,
                                    consommation_energie,
                                    estimation_ges,
                                    nombre_niveaux,
                                    nombre_entrees_avec_sas,
                                    nombre_entrees_sans_sas,
                                    en_souterrain,
                                    en_surface,
                                    nombre_boutiques,
                                    numero_lot,
                                    id_adresse,
                                    etat_avancement)
                                    VALUES (
                                    %(id)s,
                                    %(tr002_type_batiment_id)s,
                                    %(numero_dpe)s,
                                    %(classe_estimation_ges)s,
                                    %(classe_consommation_energie)s,
                                    %(annee_construction)s,
                                    %(consommation_energie)s,
                                    %(estimation_ges)s,
                                    %(nombre_niveaux)s,
                                    %(nombre_entrees_avec_sas)s,
                                    %(nombre_entrees_sans_sas)s,
                                    %(en_souterrain)s,
                                    %(en_surface)s,
                                    %(nombre_boutiques)s,
                                    %(numero_lot)s,
                                    %(id_adresse)s,
                                    %(etat_avancement)s);
                                    """
```

```
requet_sql=""" SELECT id_adresse FROM adresse
          """

id_adresse=pandas.read_sql_query(requet_sql, conn)     # recuprer l'id adresse de la table adresse
id_adresse.index = df_donnee_nettoyer.index            # definir le même index au deux df avant de fusione

df_donnee_nettoyer['id_adresse']=id_adresse            # ajouter id_adresse a df_donnee_nettoyer


#insertion_donnees(conn,sql_insertion_Logement_description,df_donnee_nettoyer)

# insertion les donnees dans la table Surface:

sql_insertion_Surface="""INSERT INTO surface
                      (id_logement,
                      surface_habitable,
                      surface_thermique_lot,
                      surface_commerciale,
                      surface_thermique_parties_communes,
                      surface_utile,
                      shon,
                      surface_baies_orientees_sud,
                      surface_baies_orientees_nord,
                      surface_baies_orientees_est_ouest,
                      surface_planchers_hauts_deperditifs,
                      surface_planchers_bas_deperditifs,
                      surface_parois_verticales_opaques_deperditives)
                      VALUES (
                      %(id)s,
                      %(surface_habitable)s,
                      %(surface_thermique_lot)s,
                      %(surface_commerciale_contractuelle)s,
                      %(surface_thermique_parties_communes)s,
                      %(surface_utile)s,%(shon)s,
                      %(surface_baies_orientees_sud)s,
                      %(surface_baies_orientees_nord)s,
                      %(surface_baies_orientees_est_ouest)s,
                      %(surface_planchers_hauts_deperditifs)s,
                      %(surface_planchers_bas_deperditifs)s,
                      %(surface_parois_verticales_opaques_deperditives)s);
                      """


 # insertion les donnees dans la table presence_verrière:

 sql_insertion_presence_verriere="""INSERT INTO vitrage
                                  (presence_verriere,
                                  surface_verriere,
                                  type_vitrage_verriere,
                                  Id_logement)
                                  VALUES (
                                  %(presence_verriere)s,
                                  %(surface_verriere)s,
                                  %(type_vitrage_verriere)s,
                                  %(id)s);
                                  """

 #insertion_donnees(conn,sql_insertion_presence_verriere,df_donnee_nettoyer)

 ##########################################################################################################

 # 4- Créer un repertoir de metadonnees:

 metadata = ['Name', 'Size', 'Item type', 'Date modified', 'Date created']

 # 1- metadonnées du fichier csv données brute:
 get_file_metadata('data_brute','44.csv','metadata_donnee_brute',metadata)

 # 2- metadonnées du fichier csv données nettoyer:
 get_file_metadata('data_nettoyee','data_nettoyee_44.csv','metadata_donnee_nettoyer',metadata)

 # 3- metadonnées du fichier geojson, données geographique:
 get_file_metadata('data_brute','correspondance-code-insee-code-postal.geojson','metadata_donnee_geographique',metadata)

 # 4- metadonnée base de données:
 def metadata_bd(conn,info_schema):
     curs = conn.cursor()
     list_info_schema=[]
     curs.execute(f"SELECT * FROM information_schema.{info_schema};")
     for i in curs.fetchall():
         list_info_schema.append(i)

     return(list_info_schema)

 schema_infos=["information_schema_catalog_name","administrable_role_authorizations","applicable_roles",
               "attributes","character_sets","collations","collation_character_set_applicability",
               "foreign_data_wrapper_options","foreign_data_wrappers","foreign_server_options",
               "sql_features","sql_implementation_info","sql_parts","sql_sizing",
               "foreign_servers","foreign_table_options",
               "foreign_tables","user_mapping_options","user_mappings","parameters","referential_constraints",
               "routines","sequences","table_constraints","tables","triggers","user_defined_types","views",
               "transforms","column_privileges","role_column_grants","role_routine_grants","role_table_grants",
               "role_udt_grants","role_usage_grants","routine_privileges","table_privileges","udt_privileges",
               "usage_privileges","data_type_privileges","enabled_roles","check_constraint_routine_usage","column_domain_usage",
               "column_udt_usage","constraint_column_usage","constraint_table_usage","domain_udt_usage","key_column_usage","view_column_usage",
               "view_routine_usage","view_table_usage","columns","triggered_update_columns","column_options","domain_constraints",
               "domains","element_types","schemata"]
 metadonnees_bd={}
 for x in schema_infos:
     metadonnees_bd[x] =metadata_bd(conn,x)

 one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))

 data_path = os.path.join(one_directory_up_path, 'data','metadonnees')

 with open(os.path.join(data_path,'metadonnees_bd.json'),'w') as fp:
     json.dump(metadonnees_bd, fp)
```

3.backup.py :

```
import os
import time
import datetime
from utils import *



def backup_db(bd_nom):
    one_directory_up_path =os.path.abspath(os.path.join(os.path.dirname( __file__ ),os.pardir))
    BACKUP_PATH = os.path.join(one_directory_up_path, 'data','backup_bd')

    # Getting current DateTime to create the separate backup folder like "20180817-123433"
    DATETIME = time.strftime('%Y%m%d-%H%M%S')
    TODAYBACKUPPATH =os.path.join(BACKUP_PATH,DATETIME)
    # Checking if backup folder already exists or not. If not exists will create it
    try:
        os.stat(TODAYBACKUPPATH)
    except:
        x=os.mkdir(TODAYBACKUPPATH)
        print("le dossier today backup est crée")

    os.chdir("C:\\Program Files\\PostgreSQL\\13\\bin")

    dumpcmd=f"pg_dump -h localhost -U postgres -f {os.path.join(BACKUP_PATH,DATETIME,'backup.sql')} {bd_nom} "
    os.system(dumpcmd)
    return(True)

#test:
backup_db("dpe_batiment")
```

4. analyse_visualisation.py :

```
from utils import *
import dash
import dash_core_components as dcc
import dash_html_components as html
import warnings
warnings.filterwarnings('ignore')
import dash_table
from dash.dependencies import Input, Output



app = dash.Dash()

"""Le code est organise soue trois parties:
1. recuprer les données a analyser avec des requete SQL NOSQL et faire les analyse.
2. Creer les graphique: apartir des données analyse et avec pie chart et foluim
3. visualiser les graphique sur le tableau de board avec app.layout

"""

#1. recuprer les données a analyser avec des requete SQL NOSQL et faire les analyse.

# connecter au bd:
utilisateur = "postgres"
mot_passe = os.environ.get('pg_psw')
nom_bdd="dpe_logement"
conn=ouvrir_connection(nom_bdd, utilisateur, mot_passe, host='localhost', port=5432)


# récuperer les données geographique a partir de MongoDb:
geodonnees=lire_geo_collection_MongoDb("geo_communes","loire_atlantique")


# creation une vue:
creation_vue_logement = """CREATE VIEW logement_info(
                        classe_consommation_energie,
                        consommation_energie,
                        annee_construction,
                        nom_commune,
                        numero_rue,
                        type_voie,
                        nom_rue,
                        SHON)
                        AS
                        SELECT classe_consommation_energie,
                        consommation_energie,
                        annee_construction,
                        nom_commune,
                        numero_rue,
                        type_voie,
                        nom_rue,
                        ROUND((surface_habitable/0.80)::numeric, 2) AS SHON
                        FROM logement_description
                        LEFT JOIN adresse ON logement_description.id_adresse=adresse.id_adresse
                        LEFT JOIN surface ON logement_description.id_logement=surface.id_logement
                        ORDER BY classe_consommation_energie
                        """

#executer_requete(creation_vue_logement, conn)



# requete sql afin de visualiser les nombres de logements par classe de consommation sur piechart
requet_sql="""SELECT classe_consommation_energie,
            COUNT(classe_consommation_energie) AS nombre_logement
            FROM logement_info
            GROUP BY classe_consommation_energie
            ORDER BY classe_consommation_energie;
            """
nb_logement_class = pandas.read_sql_query(requet_sql, conn)
```

```
# recuperation les données geo des communes:
#recuprer les nome de commune a partir de MongoDb
commun_df=commune(geodonnees)
commun_df=commun_df[["nom_commune","longitude","latitude"]]
commun_dict=commun_df[["nom_commune"]].to_dict()


# requet sql afin de visualiser les nombres de logement(classe G F) dans les differente communes sur la carte:

requet_sql="""SELECT nom_commune,
           COUNT(classe_consommation_energie) AS nombre_logement
           FROM logement_info
           WHERE classe_consommation_energie='G' OR classe_consommation_energie='F'
           GROUP BY nom_commune
           ORDER BY nombre_logement DESC;
           """
nb_logement_a_renover_commune=pandas.read_sql_query(requet_sql,conn)
#ajouter les données geo a nb_logement_a_renover_commune dataframe:
geo_communes_renov=commun_df[commun_df["nom_commune"].isin(nb_logement_a_renover_commune["nom_commune"])]
nb_logement_a_renover_commune=pandas.merge(nb_logement_a_renover_commune,geo_communes_renov)



# requet sql afin de visualiser le type de renovation (global/par element) sur la carte:
## requete renovation globale:

requet_sql="""SELECT nom_commune,
                    COUNT(nom_commune) AS numbre_logement
                    FROM logement_info
                    WHERE (classe_consommation_energie='G' OR classe_consommation_energie='F')
                    AND SHON>1000
                    AND annee_construction>1948
                    GROUP BY nom_commune;
                    """
df_type_renovation_globale=pandas.read_sql_query(requet_sql,conn).assign(type_renovation="gobale")
#ajouter les données geo a nb_logement_a_renover_commune dataframe:
geo_communes_renov_g=commun_df[commun_df["nom_commune"].isin(df_type_renovation_globale["nom_commune"])]
df_type_renovation_globale=pandas.merge(df_type_renovation_globale,geo_communes_renov_g)


## requete renovation par element:
requet_sql="""SELECT nom_commune,
                    COUNT(nom_commune) AS numbre_logement
                    FROM logement_info
                    WHERE (classe_consommation_energie='G' OR classe_consommation_energie='F')
                    AND SHON<1000
                    GROUP BY nom_commune;
                    """
df_type_renovation_par_element=pandas.read_sql_query(requet_sql,conn).assign(type_renovation="par_element")
#ajouter les données geo a nb_logement_a_renover_commune dataframe:
geo_communes_renov_e=commun_df[commun_df["nom_commune"].isin(df_type_renovation_par_element["nom_commune"])]
df_type_renovation_par_element=pandas.merge(df_type_renovation_par_element,geo_communes_renov_e)



# les noms de communes zero renovation:
commun_zero_renovation=commun_df[~commun_df["nom_commune"].isin(nb_logement_a_renover_commune["nom_commune"])]   # ~ pour renverser l'effet de isin
commun_zero_renovation["nombre_logement"]=0
nb_logement_vis=pandas.concat([nb_logement_a_renover_commune,commun_zero_renovation])


# requet sql pourcréer la table visualiser sur la dash board:
requet_sql="""SELECT  etat_avancement,
                    dpe.date_etablissement_dpe,
                    consommation_energie,
                    annee_construction,
                    surface_habitable AS SURFACE,
                    libelle AS TYPE_DE_LOGEMENT,
                    adresse.nom_commune AS commune,
                    code_postal AS CODE_POSTAL,
                    numero_rue AS NUMERO_RUE,
                    type_voie AS TYPE_VOIE,
                    nom_rue
                    FROM logement_description
                    LEFT JOIN adresse ON logement_description.id_adresse=adresse.id_adresse
                    LEFT JOIN commune ON adresse.nom_commune=commune.nom_commune
                    LEFT JOIN surface ON logement_description.id_logement=surface.id_logement
                    LEFT JOIN dpe ON logement_description.numero_dpe=dpe.numero_dpe
                    LEFT JOIN type_bâtiment ON logement_description.id_type_bâtiment=type_bâtiment.id_type_bâtiment
                    WHERE classe_consommation_energie='G' OR classe_consommation_energie='F';
            """
deatille_logement = pandas.read_sql_query(requet_sql, conn)


################################################################################
```

```
#2. Creer les graphique: apartir des données analyse et avec pie chart et foluim


## Visualiser le nombre de logement par classe de consomation sur pie chart:

color_discrete_map=("green","#0CD220","#ADFF2F","yellow","#FFD700","#F2820D","red","black")

fig = go.Figure(
    data=[go.Pie(
        labels=nb_logement_class['classe_consommation_energie'],
        values=nb_logement_class['nombre_logement'],
        textinfo='percent+label',
        marker_colors=color_discrete_map,
        insidetextorientation='radial',
        textposition='inside',
        #  make sure that Plotly won't reorder your data while plotting
        sort=False,pull=[0, 0, 0, 0,0,0.2,0.2])])

fig.update_traces(hoverinfo='label+value+percent',
                  textinfo='label+percent',
                  textfont_size=12)
fig.update_layout(
    autosize=False,
    width=400,
    height=400,
    margin=dict(
        l=20,
        r=20,
        b=10,
        t=10),
    paper_bgcolor="white",
    title_font_family="Times New Roman")

fig.update_layout(title_font_color="red")


##visulalisation des données sur la carte:

geo_donnée=geopandas.GeoDataFrame.from_features(geodonnees)      # recuprer les données geo a partir de MongoDB et les lires en geopandas

### deffinir le centre de map:
map_centre=pandas.DataFrame()
for index, row in geo_donnée.iterrows():
    x=row.geometry.centroid.x
    y=row.geometry.centroid.y
    map_centre=map_centre.append([[x,y]])

longitude_centrale=median(map_centre[0])
latitude_centrale=median(map_centre[1])

### creation la map:

m = folium.Map(location=[latitude_centrale,longitude_centrale], zoom_start=9)

folium.features.Choropleth(geo_data=geodonnees,
    data=nb_logement_a_renover_commune,
    columns=['nom_commune','nombre_logement'],
    key_on='feature.properties.nom_comm',
    fill_color='Spectral',
    fill_opacity=0.7,
    line_opacity=0.3,
    bins=9,
    weight=1,
    legend_name='Nombre de logement a renover par commune',
    dashArray='5, 3',
    highlight=True
    ).add_to(m)

### ajouter tooltip nom_commune, nb logement
for  index,row in nb_logement_vis.iterrows():
     folium.Marker(location=[row["longitude"],row["latitude"]],icon=folium.DivIcon(),
     tooltip=f"""<b>{row["nom_commune"]}</b><br><br>Nombre de logement: {row["nombre_logement"]}"""
     ).add_to(m)
```

```
### ajouter layer type de renovation:

layer1=folium.FeatureGroup(name="renovation globale")
m.add_child(layer1)
for  index,row in df_type_renovation_globale.iterrows():    # ajouter tooltip nom_commune, nb logement
     layer1.add_child(folium.Marker(location=[row["longitude"],row["latitude"]],
                    icon=folium.DivIcon(
                    html=f"""<div style="font-family:Gill Sans Extrabold, sans-serif;
                                color:black";
                                font-weight: bold;font-size="10";>{row["numbre_logement"]}</div>""")))


layer2=folium.FeatureGroup(name="renovation par elements")
m.add_child(layer2)
for  index,row in df_type_renovation_par_element.iterrows():    # ajouter tooltip nom_commune, nb logement
     layer2.add_child(folium.Marker(location=[row["longitude"],row["latitude"]],
                     icon=folium.DivIcon(
                     html=f"""<div style="font-family:Gill Sans Extrabold, sans-serif;
                                       color:black";
                                       font-weight: bold;
                                       font-size="10";>{row["numbre_logement"]}</div>""")))

folium.LayerControl().add_to(m)
m.save("map.html")


deatille_logement2=deatille_logement[deatille_logement['commune'] == 'NANTES']


 #3. visualiser les graphique sur le tableau de board avec app.layout

 app.layout = html.Div(children=[
                      html.H2("Identifier les logement à rénover et les choix de rénovation en LOIRE-ATLANTIQUE",
                             style={'color':'black',
                                    'text-shadow': '2px 2px 8px #FF7F50',
                                    'text-align': 'center',
                                    'font-family': 'cursive'}),
                      html.H2("Part de logement à renover par rapport aux nombres totale:",
                             style={'color':'#DCDCDC',
                                    'text-align': 'center',
                                    'background-color': '#006400',}),
                      html.Div(dcc.Graph(id='fig',figure=fig),
                              style={'display':'inline-block',
                              'box-shadow': '8px 8px 12px #555',
                              'margin-left':100}),
                      html.Img(src=app.get_asset_url('dpe.jpg'),
                              style={'height':'35%',
                                     'width':'35%',
                                     'display':'inline-block',
                                     'margin-left':300,
                                     'margin-Top':0,
                                     'box-shadow': '8px 8px 12px #555'}),
                      html.Br(),
                      html.Br(),
                      html.H2("Visualisation des nombres de logement à renover et les choix de rénovation par communes en LOIRE-ATLANTIQUE",
                             style={'color':'#DCDCDC',
                                    'background-color': '#006400',
                                    'text-align': 'center'}),
                      html.Iframe(id='map',
                                  srcDoc=open('map.html','r').read(),
                                  width='800',height='600',
                                  ),
                      html.Br(),
                      html.Br(),

                      html.H2("Détailes sur les logements à renover:",
                             style={'color':'#DCDCDC',
                                    'background-color': '#006400',
                                    'text-align': 'center'}),
```

```
                        dcc.Dropdown(
                           id='commune_filtre',
                           options=[{'label': nom_commune,
                                     'value':nom_commune} for nom_commune in deatille_logement['commune'].unique()],
                           value='SAINT-NAZAIRE',
                           searchable=True,
                           style={'box-shadow': '0 0 10px #666',
                                  'width': '400px',
                                  'margin': '0'}),
                        html.Br(),
                        dash_table.DataTable(id='table_commune',
                            columns=[{"name": i, "id": i} for i in deatille_logement2.columns],
                            data=deatille_logement2.to_dict('records'),
                            style_cell={'textAlign': 'left',},
                            style_data_conditional=[
                               {
                                  'if': {'row_index': 'odd'},
                                  'backgroundColor': '#FBF2B7'
                               }
                            ],
                           style_header={
                              'backgroundColor': '#FF7F50',
                              'fontWeight': 'bold'
                           },
                           page_size=10,

                           style_table={'height': '300px','overflowY': 'auto'}
                        )
                        ],
                       )

@app.callback(
    Output("table_commune", "data"),
    Input("commune_filtre", "value"))

def display_table(commune):
    deatille_logement2=deatille_logement[deatille_logement['commune'] == commune]
    data=deatille_logement2.to_dict('records')

    return data




if __name__ == '__main__':
    app.run_server(debug=False)
```