PROJET 6

Table of Contents

Activité 1 : Choix du jeu de donnés

Description

L'arbre d'alignement est un objet ponctuel représentant un arbre situé sur le domaine public et géré par le Grand Lyon. Généralement localisé le long des voies de circulation ou sur les espaces publics, il est caractérisé par des informations de gestion (genre, espèce, variété, essence botanique, hauteur, diamètre couronne, localisation, date et année de plantation, …)

Condition d’utilisation

Licence Ouverte

Mode de constitution1

Mise à jour en continu : remontée d'informations travaux des services du Grand Lyon et des partenaires et vérification de la donnée sur le terrain.

Information géographiques

- Type de représentation spatiale : Ponctuel
- Étendue géographique : 5.059776134430927 est, 4.697129744790859 ouest, 45.567725714387095 sud, 45.93833829625869 nord,
- Système de coordonnées : EPSG:4171

Informations tables

- Nombre d’enregistrements : 105 312
- Nombre d'attributs : 27
  - essencefrancais; text; Essence en français
  - circonference_cm; int4; Circonférence du tronc (cm)
  - hauteurtotale_m; int4; Hauteur totale (m)
  - hauteurfut_m; int4; Hauteur du fût (m)
  - diametrecouronne_m; int4; Diamètre de la couronne (m)
  - rayoncouronne_m; numeric; Rayon de la couronne
  - dateplantation; date; Date de plantation
  - genre; text; Genre
  - espece; text; Espèce
  - variete; text; Variété
  - essence; text; Essence
  - architecture; text; Architecture de l'arbre
  - localisation; text; Localisation de l'arbre
  - naturerevetement; text; Revêtement au pied de l'arbre
  - surfacecadre_m2; int4; Surface au pied de l'arbre (m²)
  - mobilierurbain; text; Mobilier urbain
  - anneeplantation; int4; Année de plantation
  - commune; text; Commune
  - codeinsee; int4; Code INSEE Commune
  - nomvoie; text; Nom de la voie
  - codefuv; int4; Code FUV de la voie
  - identifiant; int4; Identifiant de l'arbre
  - numero; int4; Numéro
  - codegenre; int4; Gestion de la symbologie des emplacement libre = 0 souche = 1 le reste =2
  - gid; int4
  - the_geom; geometry SRID=4171 GeomType=POINT

Activité 2 : nettoyage du jeu avec Python

J’ai commencé par sélectionner seulement les attributs utiles à mon projet et ensuite j’ai nettoyé les données qu’il me restait:

- J’ai supprimé toutes les entrées correspondant à des souches ou a des emplacements vides grâce à l’attribut ‘codegenre’
- J’ai rectifié les types des valeurs : tout était en texte donc j’ai mis les nombres en integers ou en float et les dates en un format plus simple à exploiter.
- J’ai supprimés les valeurs mises à 0 pour ne pas fausser les moyennes

Enfin j’ai ajouté un attribut indentifier incrémenté de 1 à chaque entrée et unique qui sert donc de clé primaire.

Pour ma mon nouveau fichier CSV j’ai donc ces attributs :

- identifier (int) : clé primaire
- circumference (int) : Circonférence du tronc (cm)

- height (int) : Hauteur totale (m)
- planting_date (date) : Date de plantation
- genus (text) : Genre
- species (text) : Espèce
- variety (text) : Variété
- planting_area (text) : espace de plantation
- municipality (text) : Commune
- street_name (text) : Nom de la voie
- longitude (float) : Longitude
- latitude (float) : Latitude

Le code python commenté est dans ./modules/CSVprocessing.py.

Activités 3 et 4 : Importer ce jeu dans DBBrowser et requêtes SQL

J'ai donc importé mon fichier CSV (trees_data_clean.csv) dans DBBrowser et crée une base de données (./static/data/trees.sqlite3) dans laquelle la table contenant les infroamtions sur les arbres s'appelle 'trees'.

J'ai ensuite exécuté les commandes SQL suivantes :

- **select count() from trees;** qui m'a renvoyé 99793 le nombre total d'enregistrements, inférieur à celui du premier fichier CSV car nous avons enlevé les souches et les emplacements vides.
- **select count() from trees where municipality='LYON 1ER';** qui m'a renvoyé 861 le nombre d'arbres du premier arrondssements. En faisant varier l'arrondissement on obtient ce classement (il ne montre pas forcément les arrondissements les plus végétalisés car tous les arrondissements n'ont pas la même taille):
  - 7eme : 5502
  - 3eme : 4630
  - 8eme : 3741
  - 2eme : 3121
  - 9eme : 2865
  - 6eme : 2604
  - 5eme : 1341
  - 4eme : 1299
  - 1er : 861
- **select distinct(genus) from trees;** qui m'a renvoyé la liste des genres différents (il y en a 133 donc je ne les affiche pas). Cette liste nous servira par la suite, mais on remarque qu'elle est en latin.

- **select planting_date, genus, street_name from trees where planting_date like '2024%';** qui m'a renvoyé la liste des arbres plantés en 2024 avec leur date, leur genre et le nom de la voie où ils sont :

2024-01-19	Acer	MONTÉE ROY
2024-01-08	Quercus	IMPASSE JEAN ROCHEFORT
2024-01-08	Quercus	IMPASSE JEAN ROCHEFORT

Activité 5 : Utiliser Python pour accéder à une base de données SQLite3

Voici le programme complété afin qu'il interroge la base de données précédemment créée et que le terminal affiche la bonne valeur :

```
# Import du module
import sqlite3

# CONSTANTE
FICHIER_BDD = '/static/data/trees.sqlite3'
MA_REQUETE = 'select count() from trees'

# Connexion à la Base de données SQLite
connexion_BD = sqlite3.connect(FICHIER_BDD)
c = connexion_BD.cursor()

####################
# Programme Principal
####################

#Exécution de la requête
c.execute(MA_REQUETE)

liste_data = c.fetchall()     #Récupération des résultats de la requête sour
forme de liste
print(f"J'ai {liste_data} enregistrements dans ma base de données.")
```

Activité 6 : Exploitation visuelle du jeu de données

La méthode create_map finale :

```
def create_map(data=None, path='./templates/map.html', location=None, zoom=12,
               icon_path="./static/icons/tree-icon.png"):
    """
    Create a map with markers from the given data and save it to the specified
path.

    Args:
        data (list): List of marker data tuples (latitude, longitude,
planting_date).
        path (str): Path to save the map HTML file.
        location (tuple): Center location of the map (latitude, longitude).
        zoom (int): Zoom level of the map.
        icon_path (str): Path to the custom marker icon image.
```

```
    Returns:
        str: Path to the saved map HTML file.
    """
    # Extract the folder path from the given 'path'
    folder_path = "/".join(path.split('/')[:-1])

    # Check if the folder path does not exist
    if not os.path.exists(folder_path):
        # If it doesn't exist, create the folder and all its parent directories
        os.makedirs(folder_path)

    # If 'location' to the mean latitude and longitude from the 'data' list if
it is not provided
    if location is None:
        mean_latitude = mean([marker[0] for marker in data])
        mean_longitude = mean([marker[1] for marker in data])
        location = (mean_latitude, mean_longitude)

    # Create a Folium map object
    m = folium.Map(location=location, zoom_start=zoom, attr='© Contributors
OpenStreetMap')

    # Add markers to the map if data is provided
    if data is not None:
        for marker_data in data[:]:
            # Create marker tooltip showing tree age
            tooltip = "unknown age" if marker_data[2] is None else
f"{datetime.now().year - int(marker_data[2][:4])} years"

            # Add marker to the map
            folium.Marker(
                location=(marker_data[0], marker_data[1]),
                tooltip=tooltip,
                icon=folium.CustomIcon(icon_path, icon_size=(50, 50)),
            ).add_to(m)

    # Save the map as an HTML file
    m.save(path)
    return path
```

Initialement cette méthode était beaucoup plus simple mais j'ai ajouté certaines fonctionnalités :

- Par défaut la map se centre sur la moyenne des positions des arbres à afficher ; au début je l'avait mis à (45.7578, 4.8351) le centre de Lyon mais mon jeu de données s'étendait un peu trop loin et si je dézoomais plus ça rendait moins bien.
- Le dossier 'templates' est crée si il n'existe pas, c'est un détaille mais cela évite les erreurs pour les gens qui téléchargent le code depuis le dépôt git, car je n'ai pas jugé utile d'y ajouter le dossier 'templates' qui est initialement vide.
- J'ai aussi modifié l'apparence du marker pour avoir des petits icons d'arbres sur la carte, ça rend mieux par rapport au thème de mon projet. Pour ajouter un peut d'informations j'ai fait en sorte que la date de plantation s'affiche au survol d'un marker. Je me suis limité à la date

de plantation car c'est peu esthétique si il y a trop d'informations et car le reste des informations n'étaient pas aussi exploitables que la date de plantation.

- Enfin la fonction return le chemin de la carte pour pouvoir l'exploiter par la suite (avec webbrowser on peut l'afficher automatiquement).

Activité 7 : Joindre des tables

Pour cette partie j'en ai profité pour résoudre un problème de ma table trees : les genres des arbres sont en latin. J'aurais pu garder l'attribut essencefrancais présent dans la fichier CSV brut mais il ne permettait pas de sélectionner correctement pas genres.

J'ai donc préféré créer un nouveau fichier CSV (genus_names.csv) qui met en relation les noms des genres des arbres dans différentes langues (Latin, Français, Anglais). Pour cela j'ai donné à ChatGPT la liste des genres présents dans ma base de données (grâce à la commande *select distinct(genus) from trees;*) pour qu'il me rende directement le CSV.
Ensuite grâce à cette commande je pouvais récupérer les informations que je voulais avec le nom du genre en français ou en anglais :

```
c.execute(f"select latitude, longitude, planting_date from trees join genus_names on trees.genus = genus_names.Latin where {language} = '{genus}'")
markers = c.fetchall()
```

Évidemment pour cela la variable language doit correspondre à la langue en laquelle j'ai rentré le genre. La jointure se fait donc sur le nom du genre en latin dans la table trees qui correspond à la clé primaire de la table genus_names.

Activité 8 : Amélioration ergonomique et graphique

J'ai utilisé Tkinter pour faire une interface graphique simple. J'en ai profité pour ajouter le choix de la langue pour la sélection du genre. Pour la sélection du genre j'ai d'ailleurs fait en sorte que les noms soient triées par ordre alphabétique quelque soit la langue.

Pour des questions pratiques la fenêtre de choix reste au premier plan et réaparait après chaque affichage. Aussi si on demande d'afficher un nombre d'arbres supérieur à 1000 (nombre choisit arbitrairement après quelques tests) un disclaimer apparaît pour nous prévenir que le navigateur peut avoir du mal. Je n'ai pas trouvé comment afficher plus d'arbres sans difficultés.

Enfin j'ai décidé d'utilisé une architecture de projet assez simple pour que ce soit clair :

- La présentation du projet et le code principal à la racine
- Un dossier static pour les bases de données et les icons
- Un dossier modules pour mettre le code qui permet de nettoyer le jeu de données et celui qui permet de créer et d'afficher la carte. J'ai d'ailleurs découvert qu'il suffisait d'ajouter un fichier __init__.py dans le dossier des modules pour les importer depuis un autre dossier.

- Un dossier templates est créé lors de l’exécution du programme, il ne sert qu’à contenir map.py mais il pourrait être agrémenté pour une version plus poussée du projet.