Lab 2: GIS in Python

v. 23/09/2024 – EPF – A. Gademer

Abstract:

The objective of this Lab is to show you how to manipulate geographical data (both vector and raster) in Python in order to:

1. EXPLORE a dataset, discover link and implications
2. PRODUCE a data visualization in form of an (interactive) map

⚠️ **Important remark**: This capsule will use a set of icon and graphical chart to indicate several levels of information.

- 👉: Something that you should **do** to complete the Lab
- 💾 : You need to **validate** something (call the teacher, save something for deposit)
- 💡 : A tips or **useful** remark to help you in your lab
- 🧠: An explanation on a new knowledge, that you should probably know/**retain**
- ⚠️ : Something important that you **need** to read
- ⭐: Something to try if you want to **explore** further on the subject
- IT SHOULD NOT BE NEEDED TO BE WRITTEN BUT EVERYTHING ON WHITE BACKGROUND SHOULD BE READ ALSO! (Only ⭐ is optional).

Preamble: Setting up the workspace

To realize the practical exercises, you need **a working Python installation** and the **capacity to install new libraries** (with pip per example) and to **read/run Jupyter Notebooks**.

💡 : A common way to do that is to use VSCode but you may use any IDE you prefer.

⚠️ : Note that you will need to produce a **python script** (Not a notebook) as a report of the Lab

Specifications

You will produce a python script that:

- Ask the use to enter a (French) postal address
- Get the coordinates of this address through IGN's geocodage service
- Create a 1km buffer around these coordinates
- Download all the [food store] in this buffer from OpenStreeMap Overpass API
- Organize all the info into a pretty interactive map
- Save it as a html file

⚠️ : If you want, you can change the "Food store" use case for another one of your choices:
- Health (Hospital, Pharmacy, Doctor?)
- Tourism (Cinema, Theatre, Museum, Gallery, ...)

Geocodage

Documentation: https://geoservices.ign.fr/documentation/services/services-geoplateforme/geocodage
Example of valid URL:
https://data.geopf.fr/geocodage/search?limit=1&q=21+boulevard+Berthelot+34000

💡 : The quote_plus function is probably an ally to transform a String into a valid URL.
See: https://docs.python.org/3/library/urllib.parse.html#urllib.parse.quote_plus

Overpass API

This API has it's own specific syntax. Not an easy one for beginners... but it would be cool to be able to request data that are updated in real time, no?

Note that it exist a website with a cool interface (and even an assistant) to produce Overpass queries! Turbo Overpass: https://overpass-turbo.eu/

The following code should work but you will have to adapt the overpass query to your use case (at the minimum, to take in account more amenity types.

```
# OpenStreetMap Restaurant & co

import requests
import pandas as pd

bbox=[3.85896472, 43.59122094,  3.88372524, 43.60921512] # To change

overpass_url = "http://overpass-api.de/api/interpreter"
overpass_query = """
[out:json]
[bbox:{},{},{},{}];
(
  node["amenity"="restaurant"];
  way["amenity"="restaurant"];
  node["amenity"="fast_food"];
  way["amenity"="fast_food"];
); /* the ( ); merge all the output together */

out center; /* cente give the center coordinates for way */
""".format(bbox[1],bbox[0],bbox[3],bbox[2])

display(overpass_query)
```

```
response = requests.get(overpass_url,
                        params={'data': overpass_query})
data = response.json()
df_restauration = pd.DataFrame(data['elements']) # The actual data is in
the elements value.
tags=pd.json_normalize(df_restauration.tags) # The tags column is another
JSON inside the JSON! We normalize it then concatenate to the previous
one.
df_restauration = pd.concat([df_restauration, tags], axis=1)

# Magic lines to merge nodes and ways.
# Ways are polygons so we take the coordinates of their "center" as
longitude/latitude
# Then we remove the center and (list of) nodes columns
df_restauration.lon = df_restauration.apply(lambda x: x.center['lon'] if
x.type=='way' else x.lon ,axis=1)
df_restauration.lat = df_restauration.apply(lambda x: x.center['lat'] if
x.type=='way' else x.lat ,axis=1)
df_restauration.drop(['center','nodes'], axis=1,inplace=True)

# Transforming pandas into geopandas!
gdf_restauration =
geopandas.GeoDataFrame(df_restauration,geometry=geopandas.points_from_xy(d
f_restauration.lon, df_restauration.lat), crs="EPSG:4326")

display(gdf_restauration)
```

IGN's WFS service (BD TOPO)

Documentation: https://geoservices.ign.fr/services-geoplateforme-diffusion
Example of valid URL:
https://data.geopf.fr/wfs/ows?SERVICE=WFS&VERSION=2.0.0&request=GetFeature&OUTPUTFORMAT=application/json&typename=BDTOPO_V3:troncon_de_route&bbox=43.59122094,3.85896472,43.60921512,3.88372524,urn:ogc:def:crs:EPSG::4326

Make a pretty map with Folium

(See Capsule)

💡 : How can I show pretty marker instead of dots?

Based of fontawesome : https://fontawesome.com/search

```
# Considering that you have an amenity field in your geodataframe
icon_dict={'restaurant':'utensils','fast_food':'burger',None:'circle'}

# Magic line
gdf_restauration.explore(m=map, name="Restauration",
marker_type='marker',marker_kwds={'icon':folium.DivIcon()},style_kwds={'st
yle_function':lambda x: {
    "html": f"""<span class="fa fa-{icon_dict[x["properties"]['amenity']]
if x["properties"]['amenity'] in icon_dict else 'circle'}"
style="color:blue; font-size:14px"></span>"""
}})
```

marker_type → to have visual marker and not dots
marker_kwds → to specify that the visual marker will be icons
style_kwds → dark magic. We use CSS class from fontawesome to get the images we want. classe name are like “fa-ustensils”, etc. but we use a dictionary to make is different depending on the amenity property.

💡 : How could I refine my selection? What are the information useful for my use case?

It could be cool to be able to filter AFTER the map is created. (But it is probably a big challenge).

A smaller one would be to separate different amenities into different layers so we can show/hide them at will.

You will be evaluated on:

- The pertinence of your map (for you use case)
- The legibility of your map
- The esthetics of your map
- The presence of all mandatory elements
- The quality of your code
- The originality of your map (in a smaller proportion)