Introduction to geospatial vector data in Python

In [ ]:
```
# %matplotlib inline

# import pandas as pd
import geopandas
```

Importing geospatial data

Geospatial data is often available from specific GIS file formats or data stores, like ESRI shapefiles, GeoJSON files, geopackage files, PostGIS (PostgreSQL) database, ...

We can use the GeoPandas library to read many of those GIS file formats (relying on the `fiona` library under the hood, which is an interface to GDAL/OGR), using the `geopandas.read_file` function.

For example, let's start by reading a shapefile with all the taghsimat of the world (adapted from http://www.naturalearthdata.com/downloads/110m-cultural-vectors/110m-admin-0-taghsimat/, zip file is available in the `/data` directory), and inspect the data:

In [9]:
```
# taghsimat = geopandas.read_file("data/ne_110m_admin_0_taghsimat.zip")
# or if the archive is unpacked:
taghsimat = geopandas.read_file("data/taghsimat98_3.shp")
```

In [ ]:
```
taghsimat.head()
```

In [ ]:
```
taghsimat
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
taghsimat.plot()
```

In [ ]:
```
taghsimat.explore()
```

What do we observe:

- Using `.head()` we can see the first rows of the dataset, just like we can do with Pandas.
- There is a `geometry` column and the different taghsimat are represented as polygons

- We can use the `.plot()` (matplotlib) or `explore()` (Folium / Leaflet.js) method to quickly get a *basic* visualization of the data

What's a GeoDataFrame?

We used the GeoPandas library to read in the geospatial data, and this returned us a `GeoDataFrame`:

In [ ]:
```
type(taghsimat)
```

A GeoDataFrame contains a tabular, geospatial dataset:

- It has a **'geometry' column** that holds the geometry information (or features in GeoJSON).
- The other columns are the **attributes** (or properties in GeoJSON) that describe each of the geometries

Such a `GeoDataFrame` is just like a pandas `DataFrame`, but with some additional functionality for working with geospatial data:

- A `.geometry` attribute that always returns the column with the geometry information (returning a GeoSeries). The column name itself does not necessarily need to be 'geometry', but it will always be accessible as the `.geometry` attribute.
- It has some extra methods for working with spatial data (area, distance, buffer, intersection, ...), which we will learn in later notebooks

In [ ]:
```
taghsimat.geometry
```

In [ ]:
```
type(taghsimat.geometry)
```

In [ ]:
```
taghsimat.geometry.area
```

It's still a DataFrame, so we have all the Pandas functionality available to use on the geospatial dataset, and to do data manipulations with the attributes and geometry information together.

For example, we can calculate average population number over all taghsimat (by accessing the 'pop_est' column, and calling the `mean` method on it):

In [ ]:
```
taghsimat['AR1'].mean()
```

Or, we can use boolean filtering to select a subset of the dataframe based on a condition:

In [27]:
```
tehran = taghsimat[taghsimat['Region_Eng'] == 'Tehran']
```

In [ ]:
```
tehran.plot();
```

In [ ]:
```
tehran.explore()
```

The rest of the tutorial is going to assume you already know some pandas basics, but we will try to give hints for that part for those that are not familiar.
A few resources in case you want to learn more about pandas:

- Pandas docs: https://pandas.pydata.org/pandas-docs/stable/10min.html
- Other tutorials: chapter from pandas in https://jakevdp.github.io/PythonDataScienceHandbook/, https://github.com/jorisvandenbossche/pandas-tutorial, https://github.com/TomAugspurger/pandas-head-to-tail, ...

REMEMBER:

- A `GeoDataFrame` allows to perform typical tabular data analysis together with spatial operations
- A `GeoDataFrame` (or *Feature Collection*) consists of:
  - **Geometries** or **features**: the spatial objects
  - **Attributes** or **properties**: columns with information about each spatial object

Geometries: Points, Linestrings and Polygons

Spatial **vector** data can consist of different types, and the 3 fundamental types are:

- **Point** data: represents a single point in space.
- **Line** data ("LineString"): represents a sequence of points that form a line.
- **Polygon** data: represents a filled area.

And each of them can also be combined in multi-part geometries (See https://shapely.readthedocs.io/en/stable/manual.html#geometric-objects for extensive overview).

For the example we have seen up to now, the individual geometry objects are Polygons:

In [ ]:
```
print(taghsimat.geometry[2])
```

The `shapely` library

The individual geometry objects are provided by the `shapely` library

In [ ]:
```
type(taghsimat.geometry[0])
```

To construct one ourselves:

In [34]:
```
from shapely.geometry import Point, Polygon, LineString
```

In [35]:
```
p = Point(0, 0)
```

In [ ]:
```
print(p)
```

In [39]:
```
polygon = Polygon([(1, 1), (2,2), (2, 1)])
```

In [ ]:
```
polygon.area
```

In [ ]:
```
polygon.distance(p)
```

REMEMBER:

Single geometries are represented by `shapely` objects:

- If you access a single geometry of a GeoDataFrame, you get a shapely geometry object
- Those objects have similar functionality as geopandas objects (GeoDataFrame/GeoSeries). For example:
  - `single_shapely_object.distance(other_point)` -> distance between two points
  - `geodataframe.distance(other_point)` -> distance for each point in the geodataframe to the other point

Let's practice!

Throughout the exercises in this course, we will work with several datasets about the city of Paris.

Here, we start with the following datasets:

- The administrative districts of Paris (https://opendata.paris.fr/explore/dataset/quartier_paris/): `paris_districts_utm.geojson`
- Real-time (at the moment I downloaded them ..) information about the public bicycle sharing system in Paris (vélib, https://opendata.paris.fr/explore/dataset/stations-velib-

disponibilites-en-temps-reel/information/):
`data/paris_bike_stations_mercator.gpkg`

Both datasets are provided as spatial datasets using a GIS file format.

Let's explore further those datasets, now using the spatial aspect as well.

EXERCISE 1:

We will start with exploring the bicycle station dataset (available as a GeoPackage file: `data/paris_bike_stations_mercator.gpkg`)

- Read the stations datasets into a GeoDataFrame called `stations` .
- Check the type of the returned object
- Check the first rows of the dataframes. What kind of geometries does this datasets contain?
- How many features are there in the dataset?

▶ Hints

EXERCISE 2:

- Make a quick plot of the `stations` dataset.
- Make the plot a bit larger by setting the figure size to (12, 6) (hint: the `plot` method accepts a `figsize` keyword).

EXERCISE 4:

- Make a histogram showing the distribution of the number of bike stands in the stations.

▶ Hints

EXERCISE 5:

Let's now visualize where the available bikes are actually stationed:

- Make a plot of the `stations` dataset (also with a (12, 6) figsize).
- Use the `'available_bikes'` columns to determine the color of the points. For this, use the `column=` keyword.
- Use the `legend=True` keyword to show a color bar.

EXERCISE 6:

Next, we will explore the dataset on the administrative districts of Paris (available as a GeoJSON file: "data/paris_districts_utm.geojson")

- Read the dataset into a GeoDataFrame called `districts`.
- Check the first rows of the dataframe. What kind of geometries does this dataset contain?
- How many features are there in the dataset? (hint: use the `.shape` attribute)
- Make a quick plot of the `districts` dataset (set the figure size to (12, 6)).

EXERCISE 7:

What are the largest districts (biggest area)?

- Calculate the area of each district.
- Add this area as a new column to the `districts` dataframe.
- Sort the dataframe by this area column for largest to smallest values (descending).

▶ Hints

In [ ]:

In [88]:
```
# tehran = taghsimat[taghsimat['Region_Eng'] == 'Tehran']
zanjan = taghsimat[taghsimat['Region_Eng'] == 'Zanjan']
```

In [ ]:
```
zanjan
```

In [94]:

In [ ]:

In [43]:

In [64]:
```
buf= tehran.to_crs('3857').buffer(5000)
```

In [ ]:
```
map = buf.explore(color='red', label='Buffer Areas')

# Add the second GeoDataFrame to the same map
tehran.explore(m=map, color='blue', label='Tehran Areas')

# Display the map
map
```

In [ ]:

In [ ]:

In [66]:
```
geojson= {
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "coordinates": [
          [
            [
              56.372642909090075,
              37.992756079036184
            ],
            [
              47.187028840249496,
              39.268702414842096
            ],
            [
              47.580171232158534,
              35.32433304672443
            ],
            [
              55.21952161076629,
              34.27179112848765
            ],
            [
              56.372642909090075,
              37.992756079036184
            ]
          ]
        ],
        "type": "Polygon"
      }
    }
  ]
}
```

In [79]:
```
geojson_polygon = geopandas.GeoDataFrame.from_features(geojson['features'])
iran_polygon= geojson_polygon.set_crs('4326')
```

In [ ]:
```
iran_polygon.explore()
```

In [ ]:
```

```

In [81]:
```
clipped_gdf = taghsimat.clip(iran_polygon)
```

In [ ]:
```
clipped_gdf.explore()
```

In [ ]:
```

```