Chapitre VI Données Géospatiales

M1 IA Data Science

Ilyas Bambrik

2024

Table des matières

Introduction 3

I - **Module geopandas** 4

II - **Méthodes GeoDataframe** 8

Références 14

Introduction

Lorsque les données inclues dans notre *Dataframe* représentent une entité géographique (tel que des coordonnées, une zone géographique, un trajet), ce genre de données peut être visualisé sur une carte. Ce chapitre introduit le module *geopandas*[20*] qui permet de manipuler et visualiser les données géospatiales d'une manière personnalisée.

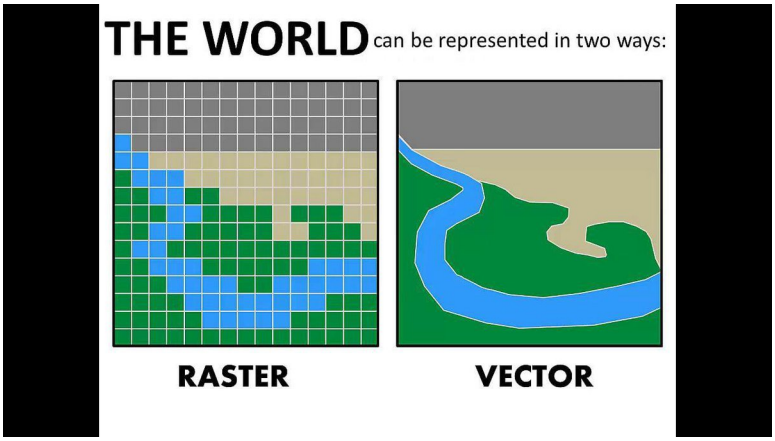
Module geopandas

I

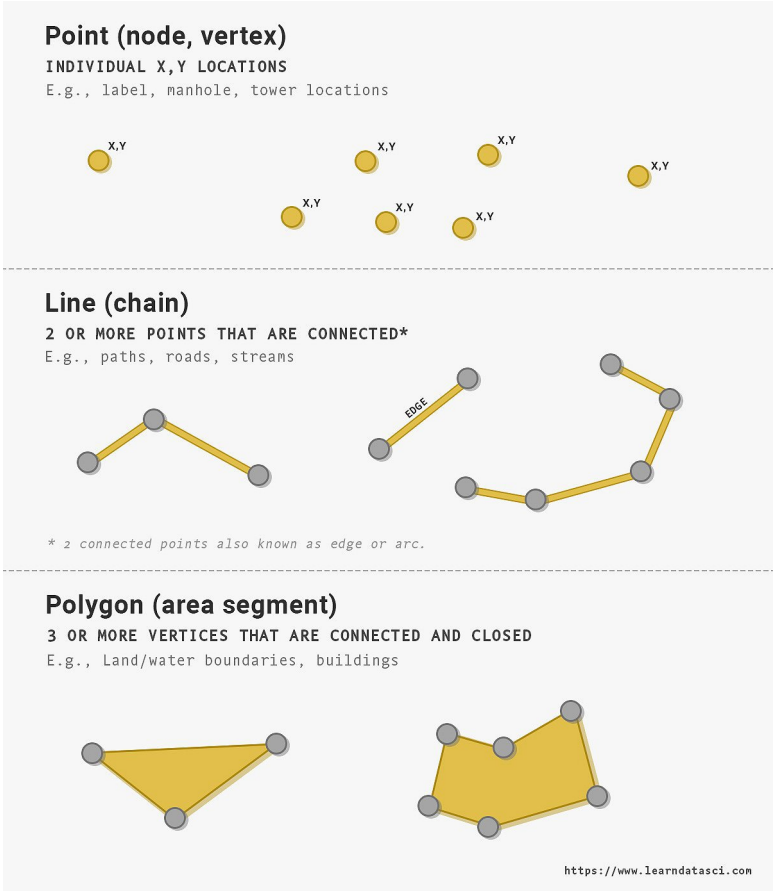
Deux types de représentations existent pour les données géospatiales:

a) Matricielle (Raster): une grille de points représentant l'information géospatiale. Cette représentation est utilisée pour les images satellite.

b) Vectorielle : une liste de points discrets représentant l'information géospatiale. Cette représentation est utilisée dans Google Map par exemple.



Dans ce cours, nous allons manipuler la représentation vectorielle où une donnée peut être : a) Un point , b) Une ligne (liste de deux points ou plus), c) Un polygone (ligne fermée), d) multi-polygone. Ainsi, selon cette logique, la valeur d'une colonne peux être une liste de points/polygones représentant une entité géospatiale.



Des formats de fichier spécifiques ont été développés pour ce genre de DataSet (*ShapeFile*, *GeoJson*, *GeoPackage*). En outre, le module pandas a été étendu pour ce genre de Dataframe dans un module nommé *geopandas*.

Comme *pandas*, *geopandas* propose une méthode *read_file* permettant de lire un DataSet. Le code suivant crée un objet *GeoDataFrame* à partir du fichier *.shp* (*ShapeFile*) :

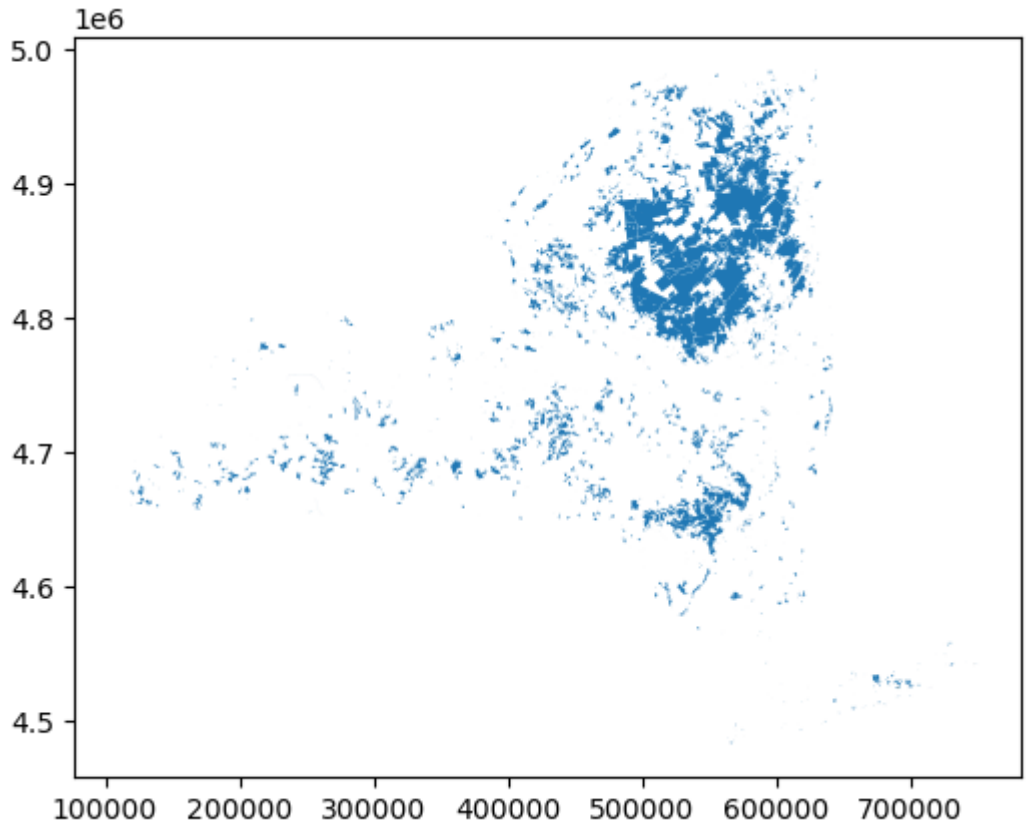
```
1 import geopandas as gpd
2 gdf=gpd.read_file("./gpd/DEC_lands.shp")
3 gdf.head()
```

	URL	SOURCE	UPDATE_	OFFICE	ACRES	LANDS_UID	GREENCERT	SHAPE_AREA	SHAPE_LEN	geometry
0	http://www.dec.ny.gov/	DELAWARE RPP	5/12	STAMFORD	738.620192	103	N	2.990365e+06	7927.662385	POLYGON ((486093.245 4635308.586, 486787.235 4...
1	http://www.dec.ny.gov/	DELAWARE RPP	5/12	STAMFORD	282.553140	1218	N	1.143940e+06	4776.375600	POLYGON ((491931.514 4637416.256, 491305.424 4...
2	http://www.dec.ny.gov/	DELAWARE RPP	5/12	STAMFORD	234.291262	1780	N	9.485476e+05	5783.070364	POLYGON ((486000.287 4635834.453, 485007.550 4...
3	http://www.dec.ny.gov/	GREENE RPP	5/12	STAMFORD	450.106464	2060	N	1.822293e+06	7021.644833	POLYGON ((541716.775 4675243.268, 541217.579 4...
4	http://www.dec.ny.gov/lands/22593.html	DECRP, ESSEX RPP	12/96	RAY BROOK	69.702387	1517	N	2.821959e+05	2663.909932	POLYGON ((583896.043 4909643.187, 583891.200 4...

Un *GeoDataframe* étend l'objet *Dataframe* avec de nouvelles méthodes et fonctionnalités. Particulièrement, la colonne *geometry* est un colonne particulière qui comporte les données géométriques à tracer.

La méthode *plot*() de l'objet *GeoDataframe* permet simplement de tracer les objets géospatiales présentes dans la colonne *geometry* .

```
gdf.plot()
```



Un attribut important de l'objet *GeoDataframe* est le *crs* (Coordinate Reference System) qui représente le système de coordonnées géographiques. Pour identifier des emplacements exacts sur la surface de la Terre, nous utilisons un système de coordonnées géographiques. Par exemple, essayez de rechercher 35.91801, 0.06266 sur Google Maps. Ces deux chiffres indiquent un endroit exact : Salamandre, Mostaganem, en Algérie. Les deux nombres sont des coordonnées définies par le CRS. Même si la Terre est une sphère tridimensionnelle, nous utilisons un système de coordonnées bidimensionnel de longitude (lignes verticales allant du nord au sud) et de latitude (lignes horizontales allant de l'est en ouest) pour identifier une position dans la surface de la Terre.

Dans ce GeoDataframe, le système de coordonnées géographiques utilisé est *EPSG:26918*. Mais celui-ci peut être changé en réaffectant l'attribut *.crs* de l'objet *GeoDataframe*.

```
gdf.crs
```

```
<Projected CRS: EPSG:26918>
Name: NAD83 / UTM zone 18N
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: North America - between 78°W and 72°W - onshore and offshore. Canada -
Nunavut; Ontario; Quebec. United States (USA) - Connecticut; Delaware; Maryland;
Massachusetts; New Hampshire; New Jersey; New York; North Carolina; Pennsylvania;
Virginia; Vermont.
```

```
- bounds: (-78.0, 28.28, -72.0, 84.0)
Coordinate Operation:
- name: UTM zone 18N
- method: Transverse Mercator
Datum: North American Datum 1983
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

Un changement de CRS avec la méthode *to_crs()* introduit automatiquement une translation des coordonnées dans la colonne *geometry* vers le nouveau système de coordonnées. Par exemple, initialement les valeurs de la colonne *geometry* sont les suivantes :

```
gdf.geometry.head()
```

```
0    POLYGON ((486093.245 4635308.586, 486787.235 4...
1    POLYGON ((491931.514 4637416.256, 491305.424 4...
2    POLYGON ((486000.287 4635834.453, 485007.550 4...
3    POLYGON ((541716.775 4675243.268, 541217.579 4...
4    POLYGON ((583896.043 4909643.187, 583891.200 4...
Name: geometry, dtype: geometry
```

Après la translation vers le CRS *EPSG:2263*, les coordonnées dans la colonne *geometry* change ainsi que les informations affichées par l'attribut *crs* (le CRS EPSG:26918 initiale utilise l'unité *mètre*, après la translation vers EPSG:2263 l'unité utilisée est le *pieds [foot]*)

```
gdf.to_crs("EPSG:2263",inplace=True)
display(gdf.crs)
display(gdf.geometry.head())
```

```
<Projected CRS: EPSG:2263>
Name: NAD83 / New York Long Island (ftUS)
Axis Info [cartesian]:
- X[east]: Easting (US survey foot)
- Y[north]: Northing (US survey foot)
Area of Use:
- name: United States (USA) - New York - counties of Bronx; Kings; Nassau; New
York; Queens; Richmond; Suffolk.
- bounds: (-74.26, 40.47, -71.8, 41.3)
Coordinate Operation:
- name: SPCS83 New York Long Island zone (US Survey feet)
- method: Lambert Conic Conformal (2SP)
Datum: North American Datum 1983
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
0    POLYGON ((666192.710 622596.170, 668463.483 62...
1    POLYGON ((685435.256 629296.292, 683388.604 63...
2    POLYGON ((665907.220 624325.765, 662658.515 62...
3    POLYGON ((850286.928 751592.839, 848658.643 75...
4    POLYGON ((997977.275 1519931.685, 997961.628 1...
Name: geometry, dtype: geometry
```

Syntaxe : Installation

Pour installer *geopandas*, il suffit d'exécuter la commende suivante. La version 0.14.4 est particulièrement plus utile pour le cours et TP de ce module :

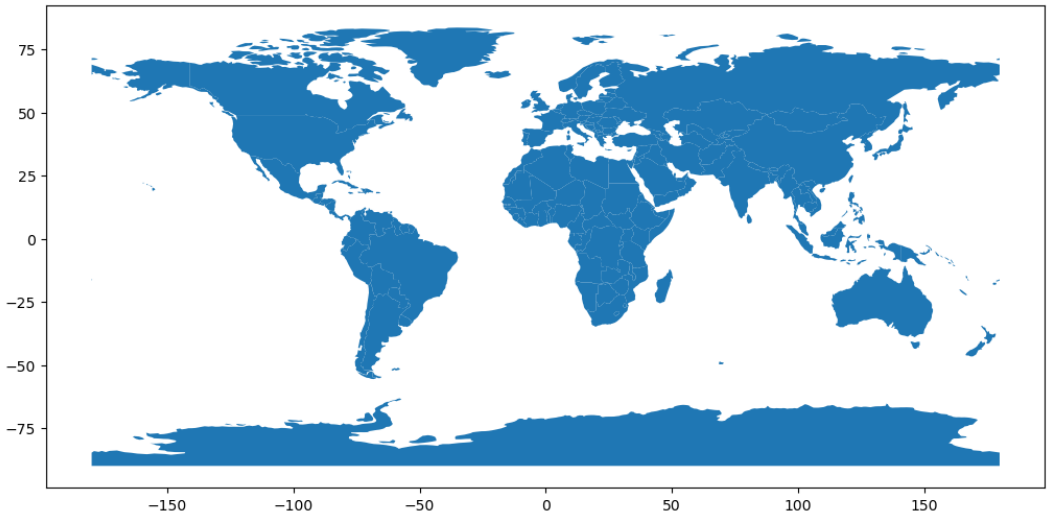
```
pip install geopandas==0.14.4
```

Méthodes GeoDataframe

II

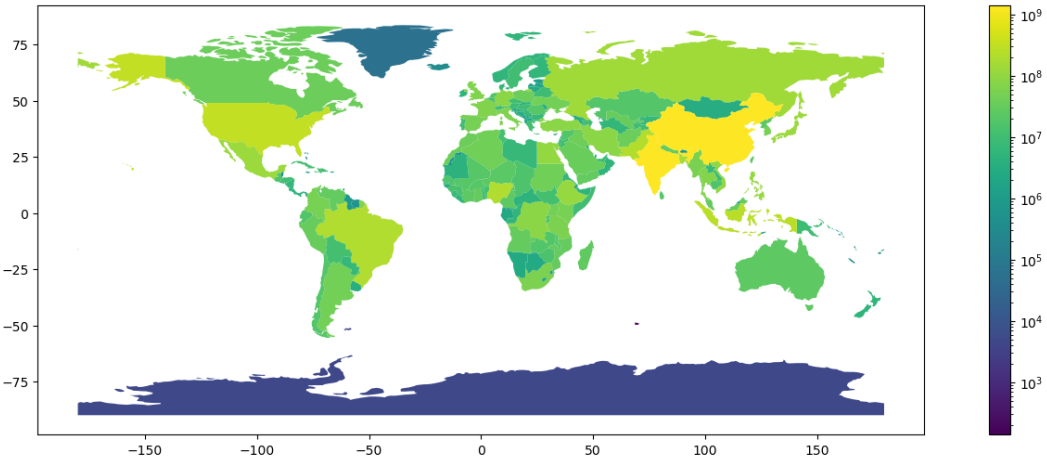
Prenant le *GeoDataframe* disponible avec *geopandas* qui représente la densité de la population dans chaque pays :

```
world_gdf = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
world_gdf.plot(figsize=(18,6))
```



Le code suivant trace un Heatmap des pays (représentés par des polygones dans la colonne *geometry*) selon la colonne *pop_est* (estimation de la population) du GeoDataframe world_gdf. L'argument *norm* permet de définir les couleurs utilisées selon un échelle. *matplotlib.colors.LogNorm* permet de normaliser les valeurs à visualiser dans la plage 0-1 sur l'échelle logarithmique.

```
import matplotlib
world_gdf = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
world_gdf.plot("pop_est",
               legend=True,
               figsize=(18,6),
               norm=matplotlib.colors.LogNorm(vmin=world_gdf.pop_est.min(),vmax=
world_gdf.pop_est.max()))
```



L'attribut *boundary* permet d'obtenir le contour/bordure des polygones contenus dans l'objet *GeoSeries*:

```
world_gdf.boundary
```

```
0      MULTILINESTRING ((180.00000 -16.06713, 180.000...
1      LINESTRING (33.90371 -0.95000, 34.07262 -1.059...
2      LINESTRING (-8.66559 27.65643, -8.66512 27.589...
3      MULTILINESTRING ((-122.84000 49.00000, -122.97...
4      MULTILINESTRING ((-122.84000 49.00000, -120.00...
                            ...
172    LINESTRING (18.82982 45.90887, 18.82984 45.908...
173    LINESTRING (20.07070 42.58863, 19.80161 42.500...
174    LINESTRING (20.59025 41.85541, 20.52295 42.217...
175    LINESTRING (-61.68000 10.76000, -61.10500 10.8...
176    LINESTRING (30.83385 3.50917, 29.95350 4.17370...
Length: 177, dtype: geometry
```

L'attribut *centroid* renvoie le centre de chaque objet polygone dans le *GeoSeries* :

```
world_gdf.centroid
```

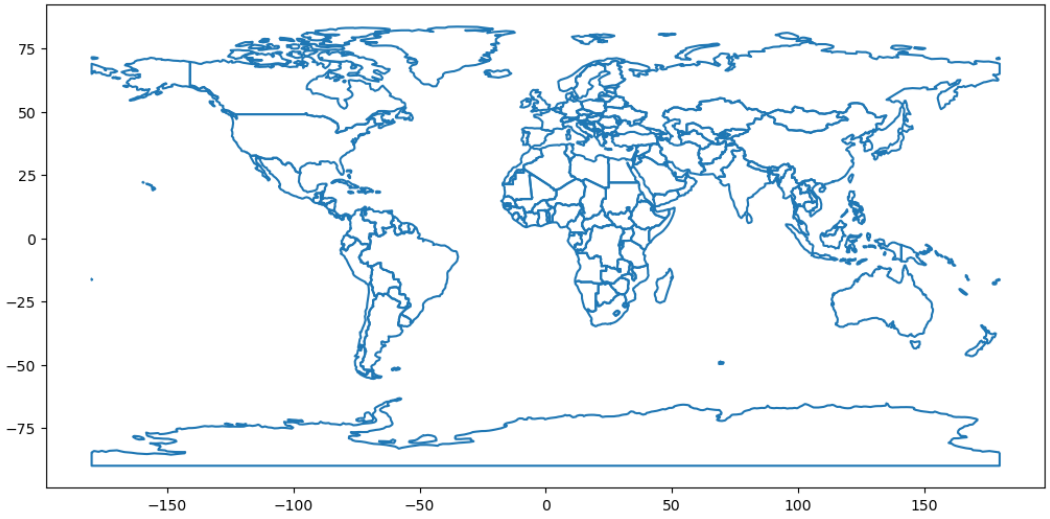
```
0      POINT (163.85316 -17.31631)
1        POINT (34.75299 -6.25773)
2       POINT (-12.13783 24.29117)
3       POINT (-98.14238 61.46908)
4      POINT (-112.59944 45.70563)
                  ...
172      POINT (20.81965 44.23304)
173      POINT (19.28618 42.78904)
174      POINT (20.89536 42.57937)
175     POINT (-61.33037 10.42824)
176       POINT (30.19862 7.29289)
Length: 177, dtype: geometry
```

Pour définir une colonne comme étant la colonne *geometry* active de *GeoDataframe*, la méthode *set_geometry*() est utilisée (l'argument *inplace* peut être utilisé comme dans *set_index*()).

Le code suivant crée une nouvelle colonne pour le contour des pays, rend celle-ci comme la colonne *geometry* active et puis affiche le résultat.

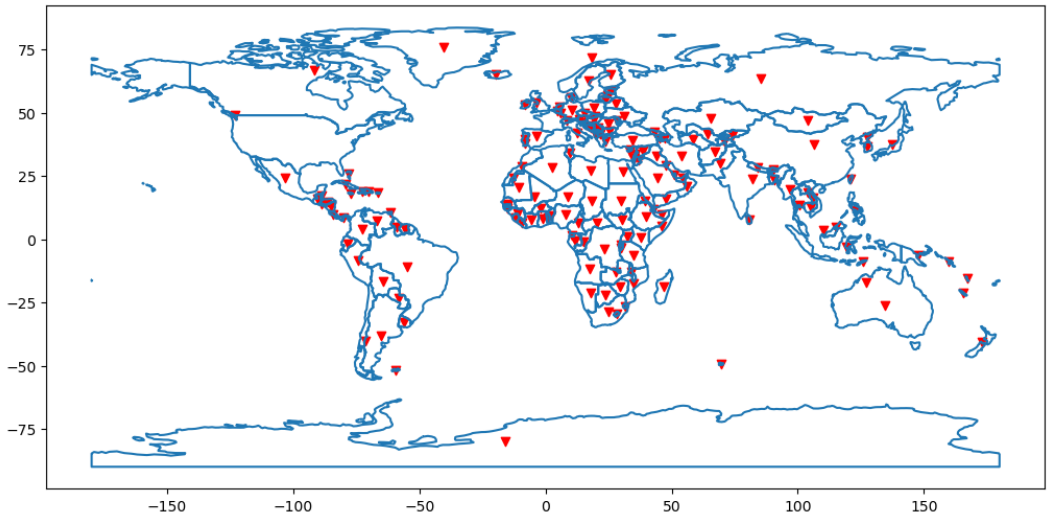
```
df=world_gdf.copy()
df["bordure"]=df.boundary
df=df.set_geometry("bordure")
```

```
4 ax=df.plot(figsize=(18,6))
```



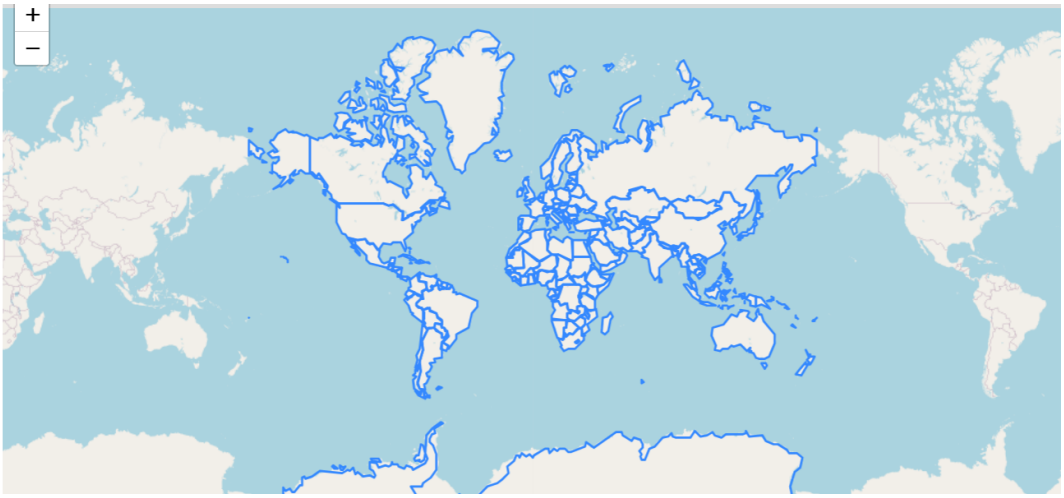
Le code suivant affiche les bordures et puis les centres de chaque pays dans le même plan avec une syntaxe identique à seaborn/matplotlib.

```
1 df=world_gdf.copy()
2 df["bordure"]=df.boundary
3 df=df.set_geometry("bordure")
4 ax=df.plot(figsize=(18,6))
5 df["centre"]=df.centroid
6 df.set_geometry("centre").plot(marker="v",ax=ax,color="red")
```



L'objet *GeoDataframe* propose aussi la méthode *explore* qui affiche une carte interactive. Cependant, celle-ci nécessite le téléchargement de modules supplémentaire (*folium* et *mapclassify*) :

```
1 df=world_gdf.copy()
2 df["bordure"]=df.boundary
3 df=df.set_geometry("bordure")
4 df.explore(figsize=(18,6))
```



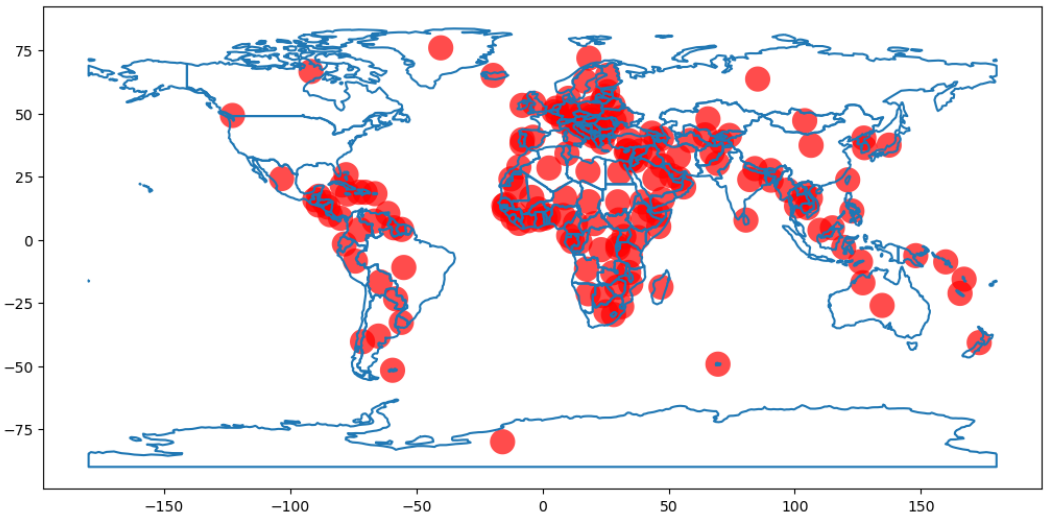
Pour avoir tous les points dans un périmètre d'un point référence (idem pour les points d'une ligne ou polygone), la méthode *buffer* d'une colonne de type *geometry* est utilisée. Celle-ci permet de définir l'épaisseur du contour d'une entité géométrique. Le code suivant renvoie pour chaque centre du pays, tous les points dans un périmètre de 5 unités (l'unité dépend du *crs*).

```
df["centre"]=df.centroid.buffer(5)
df["centre"].head()
```

```
0    POLYGON ((132.01664 -17.06903, 131.99256 -17.5...
1    POLYGON ((39.77684 -6.30563, 39.75277 -6.79572...
2    POLYGON ((-7.55205 24.08211, -7.57613 23.59202...
3    POLYGON ((-86.83746 66.66476, -86.86154 66.174...
4    POLYGON ((-117.96405 49.22056, -117.98813 48.7...
Name: centre, dtype: geometry
```

De même, le code suivant affiche les centres avec un *buffer* de 5 unités. Dans la méthode plot, le paramètre *alpha* est utilisé pour définir la transparence de la couleur.

```
df=world_gdf.copy()
df["bordure"]=df.boundary
df=df.set_geometry("bordure")
ax=df.plot(figsize=(18,6))
df["centre"]=df.centroid.buffer(5)
df.set_geometry("centre").plot(marker="v",ax=ax,color="red",alpha=0.7)
```



Une autre méthode très utile est la méthode *distance(point(s))* qui revoie la liste des distances entre une série de points/polygones et un point de référence ou des entités dans une autre série géométrique de la même taille. Le code suivant calcule la distance entre le centre de l'Algérie et le reste des pays :

```
df.geometry.distance((df.loc[df.name=="Algeria"].geometry.centroid).iloc[0])
```

```
0      180.619172
1       40.418830
2       11.276055
3       58.536841
4       71.521570
          ...
172     22.616484
173     21.345209
174     22.596153
175     65.815748
176     28.736746
Length: 177, dtype: float64
```

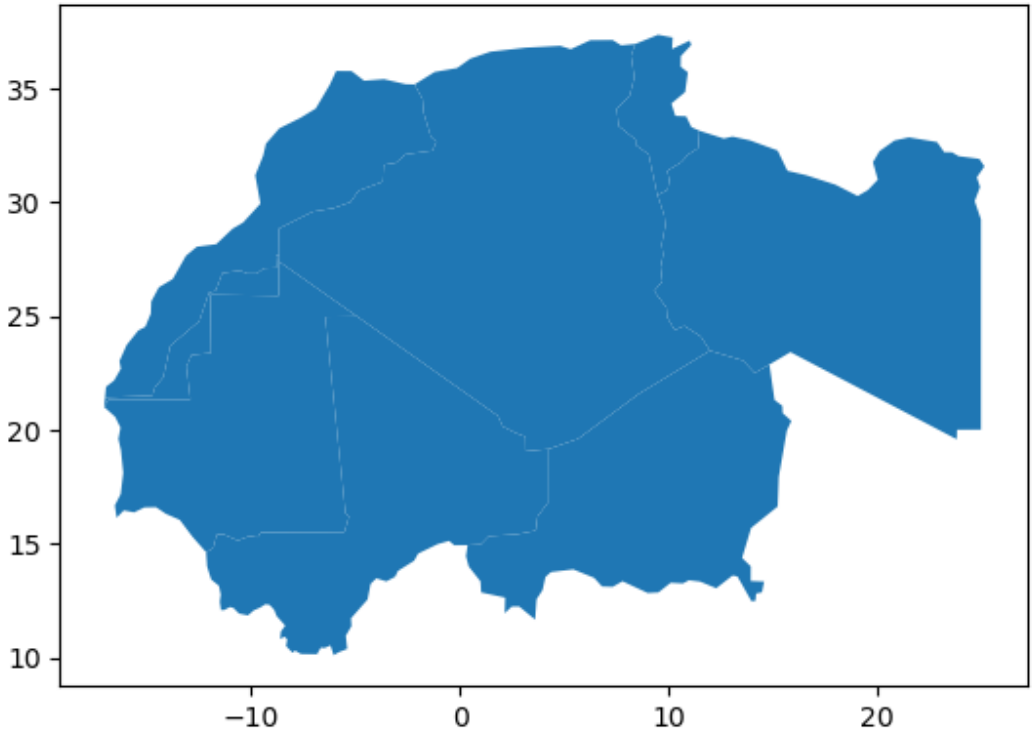
Similairement, l'attribut *area* renvoie la surface de chaque polygone de la géosérie. La distance ainsi que la surface calculées dépendent du crs :

```
df.geometry.area
```

```
0         1.639511
1        76.301964
2         8.603984
3      1712.995228
4      1122.281921
          ...
172       8.604719
173       1.479321
174       1.231641
175       0.639000
176      51.196106
Length: 177, dtype: float64
```

La méthode *intersects()* permet de tester l'intersection entre objets géospatiales. Supposant que nous souhaitons trouver les pays qui ont des frontières proches de l'Algérie. Ici, nous utilisons la méthode *buffer*() pour amplifier l'épaisseur des frontières de l'Algérie avant de tester l'intersection :

```
frontiers_alg=df.loc[df.name=="Algeria"].geometry.buffer(1).iloc[0]
df["bordure_alg"]=df.geometry.intersects(frontiers_alg)
df=df[df.bordure_alg]
df.plot()
```



Similairement au *join* vue dans le chapitre II, *sjoin* (spacial join) permet de combiner deux geodataframe s'il y a intersection spéciale entre les valeurs des deux colonnes *geometry*.

Fondamental : intersects, distance et sjoin

Lorsque les méthodes *intersects*, *distance* et *sjoin* sont utilisées, il est primordiale d'avoirs un crs unifié entre les objets géospatiales inclus dans le calcule (sinon le résultat n'aura aucun sens).

Références

[20] Geopandas 2023, https://geopandas.org/en/stable/