



HAL
open science

Atelier SAGEO 2021 : Traitements spatiaux avec le plugin Python t4gpd dans le contexte d'un Jupyter Notebook

Thomas Leduc

► To cite this version:

Thomas Leduc. Atelier SAGEO 2021 : Traitements spatiaux avec le plugin Python t4gpd dans le contexte d'un Jupyter Notebook. École thématique. La Rochelle, France. 2021. hal-03262455

HAL Id: hal-03262455

<https://hal.science/hal-03262455>

Submitted on 16 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Atelier SAGEO 2021

Traitements spatiaux avec le plugin Python t4gpd dans le contexte
d'un Jupyter Notebook

Thomas Leduc

thomas.leduc@crenau.archi.fr

UMR 1563 AAU / CRENAU



5 mai 2021

Plan de la présentation

- 1 Introduction à Pandas
- 2 Introduction à Shapely
- 3 Introduction à GeoPandas
- 4 *t4gpd*

Pandas

- Bibliothèque Python de manipulation et analyse de données
- Les principales structures de données Pandas sont :
 - ▶ les [Series](#) pour les données 1D
 - ▶ les [DataFrame](#) pour les données 2D (ou plus)
- Ces structures de données sont [indexées](#) pour plus d'efficacité, couplées à des utilitaires de lecture/écriture de fichiers [csv](#), tableurs [xls\(x\)](#), fichiers [HDF5](#), bases de données [SQL](#), etc.
- Pandas permet la concaténation et la fusion de gros volumes de données (*), l'analyse de séries temporelles (*), facilite la gestion des données manquantes (*), les tableaux croisés dynamiques (*), etc.
- Ce qui suit s'inspire largement de [10 minutes to pandas](#)

Pandas - à propos de Series

- « *Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index.* »
- Series acts similarly to a `numpy.ndarray` or a fixed-size `dict`.

```

1 from numpy import mean; from pandas import Series
2
3 s1 = Series(data=[ 'a1', 'b2', 'c3', 'a4' ])
4 print(s1[ s1.str.startswith('a') ])
5
6 s2 = Series(data=range(0, 40, 10))
7 print(s2[ s2 > mean(s2) ])
8
9 print(s2.is_monotonic_increasing)
10
11 s3 = Series({c:f'a{i}' for i,c in enumerate(['a','b','c'])})
12 print(s3)
13
14 print(s3.iloc[1], s3.loc['b'])
15
16 s4 = Series([1, 2, 3, 4, None, 2])
17 print(s4.hasnans, s4.is_unique)

```

```

In [1]: %run scripts/1_1_pandas.py
0    a1
3    a4
dtype: object

2    20
3    30
dtype: int64

True

a    a0
b    a1
c    a2
dtype: object

a1 a1

True False

In [2]:

```

Pandas - à propos de DataFrame

- « *DataFrame* is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. »

```

1 from pandas import DataFrame
2
3 df1 = DataFrame(data=[[ 'DE', 'Allemagne', 'Berlin' ],
4                       [ 'FR', 'France', 'Paris' ], [ 'IT', 'Italie', 'Rome' ]],
5                 columns=[ 'code', 'pays', 'capitale' ]
6                 )
7 print(df1)
8
9 df1.set_index([ 'code' ], drop=True, inplace=True)
10 print(df1)
11
12 df1.reset_index(inplace=True)
13 print(df1)
14
15 df2 = df1[~df1.code.isin([ 'FR' ])]
16 print(df2)
17
18 df2.reset_index(drop=True, inplace=True)
19 print(df2)

```

```

In [1]: %run scripts/s_2_pandas.py
code    pays capitale
0  DE  Allemagne  Berlin
1  FR   France   Paris
2  IT   Italie   Rome

code    pays capitale
DE  Allemagne  Berlin
FR   France   Paris
IT   Italie   Rome

code    pays capitale
0  DE  Allemagne  Berlin
1  FR   France   Paris
2  IT   Italie   Rome

code    pays capitale
0  DE  Allemagne  Berlin
2  IT   Italie   Rome

code    pays capitale
0  DE  Allemagne  Berlin
1  IT   Italie   Rome

```

Pandas - concaténation et fusion (jointure attributaire)

- Consulter notamment `pandas.concat(...)` et `pandas.merge(...)`

```

1 from pandas import concat, DataFrame, merge
2
3 df3 = DataFrame(data=[[ 'Allemagne', 'Europe'],
4   [ 'Italie', 'Europe']],
5   columns=[ 'pays', 'continent' ]
6   )
7 print(df3)
8
9 df4 = DataFrame(data=[[ 'Inde', 'Asie']],
10  columns=[ 'pays', 'continent' ]
11  )
12 print(df4)
13
14 df5 = concat([df3, df4])
15 print(df5)
16
17 df5.reset_index(drop=True, inplace=True)
18 print(df5)
19
20 df6 = merge(df5, df1[[ 'pays', 'capitale']],
21  how='left', on='pays')
22 print(df6)

```

```

      pays continent
0  Allemagne  Europe
1    Italie  Europe

      pays continent
0    Inde  Asie

      pays continent
0  Allemagne  Europe
1    Italie  Europe
0    Inde  Asie

      pays continent
0  Allemagne  Europe
1    Italie  Europe
2    Inde  Asie

      pays continent capitale
0  Allemagne  Europe  Berlin
1    Italie  Europe  Rome
2    Inde  Asie  NaN

```

Pandas - mécanisme de vue

- Attention au mécanisme de **vue**¹

```

1 from pandas import DataFrame
2
3 df1 = DataFrame(data=[[ 'a', 13], [ 'd', 7], [ 'z', 4],
4   [ 'a', 9], [ 'z', 2]], columns=[ 'colA', 'colB'])
5 print(df1)
6
7 df2 = df1[(df1.colB >= 7) & (df1.colB <= 9)]
8 print(df2) # df2 est une vue de df1
9
10 # df2.colB = 10 * df2.colB
11 # A value is trying to be set on a copy of a slice
12 # from a DataFrame.
13 # Try using .loc[row_indexer, col_indexer] = value
14 # instead
15
16 df3 = df1.loc[df1[(df1.colB >= 7) & (df1.colB <= 9)].index]
17 df3.colB = 10 * df3.colB
18 print(df3) # df3 est une copie profonde de df1
19
20 df4 = df2.copy(deep=True)
21 df4.colB = 10 * df4.colB
22 print(df4) # df4 est une copie profonde de df2

```

```

In [1]: %run scripts/S_4_pandas.py
colA colB
0 a 13
1 d 7
2 z 4
3 a 9
4 z 2

colA colB
1 d 7
3 a 9

colA colB
1 d 70
3 a 90

colA colB
1 d 70
3 a 90

```

1. Une modification de df2 ne sera pas reportée sur df1.

Plan de la présentation

- 1 Introduction à Pandas
- 2 Introduction à Shapely
- 3 Introduction à GeoPandas
- 4 *t4gpd*

Shapely

- Extrait de <https://shapely.readthedocs.io>
 - ▶ Shapely is a Python package for set-theoretic analysis and manipulation of planar features using (via Python's ctypes module) functions from the well known and widely deployed **GEOS** library. GEOS, a port of the **Java Topology Suite (JTS)**, is the geometry engine of the **PostGIS** spatial extension for the PostgreSQL RDBMS. The designs of JTS and GEOS are largely guided by the **Open Geospatial Consortium's** Simple Features Access Specification 1 (*) and Shapely adheres mainly to the same set of standard classes and operations. Shapely is thereby deeply rooted in the conventions of the geographic information systems (GIS) world, but aspires to be equally useful to programmers working on non-conventional problems.

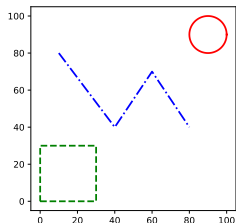
Shapely

- Déclaration de géométries — représentation discrète du monde

```

1 import matplotlib.pyplot as plt
2 from shapely.geometry import LineString, Point, Polygon
3
4 red = Point(90, 90).buffer(10)
5 blue = LineString([ (10, 80), (40, 40), (60, 70), (80, 40) ])
6 green = Polygon([ (0, 0), (30, 0), (30, 30), (0, 30) ])
7
8 plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
9 plt.plot(*red.exterior.xy, 'r-', linewidth=2)
10 plt.plot(*blue.xy, 'b-.', linewidth=2)
11 plt.plot(*green.exterior.xy, 'g--', linewidth=2)
12 plt.savefig('../img/6_1_shapely.pdf')

```



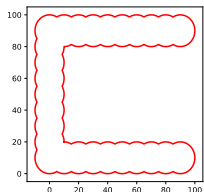
Shapely

- Union (efficace) de géométries

```

1 import matplotlib.pyplot as plt
2 from shapely.geometry import MultiPoint
3 from shapely.ops import unary_union
4
5 geoms = list()
6 for i in range(10, 100, 10):
7     geoms.append(MultiPoint([(0, i), (i, 10), (i, 90)]).buffer(10))
8
9 red = unary_union(geoms)
10
11 plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
12 plt.plot(*red.exterior.xy, 'r-', linewidth=2)
13 plt.savefig('../img/6_2_shapely.pdf')

```



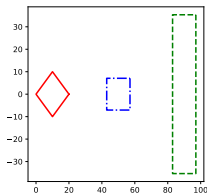
Shapely

- Transformations affines de géométries

```

1 import matplotlib.pyplot as plt
2 from shapely.affinity import rotate, translate, scale
3 from shapely.geometry import Point
4 from shapely.ops import unary_union
5
6 red = Point(10, 0).buffer(10, 1)
7 blue = translate(rotate(red, 45, origin='center'), xoff=40.0, yoff=0.0)
8 green = scale(translate(blue, xoff=40.0, yoff=0.0), xfact=1, yfact=5, origin='center')
9
10 plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
11 plt.plot(*red.exterior.xy, 'r-', linewidth=2)
12 plt.plot(*blue.exterior.xy, 'b-', linewidth=2)
13 plt.plot(*green.exterior.xy, 'g--', linewidth=2)
14 plt.savefig('../img/6_3_shapely.pdf')

```



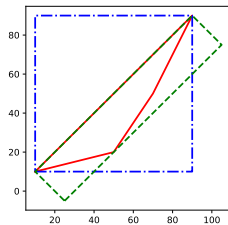
Shapely

- Création de géométries englobantes

```

1 import matplotlib.pyplot as plt
2 from shapely.geometry import MultiPoint
3
4 mpts = MultiPoint( [ (90, 90), (70, 50), (50, 20), (10, 10), (30, 30) ] )
5 red = mpts.convex_hull
6 blue = mpts.envelope
7 green = mpts.minimum_rotated_rectangle
8
9 plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
10 plt.plot(*red.exterior.xy, 'r-', linewidth=2)
11 plt.plot(*blue.exterior.xy, 'b-', linewidth=2)
12 plt.plot(*green.exterior.xy, 'g--', linewidth=2)
13 plt.savefig('../img/6_4_shapely.pdf')

```



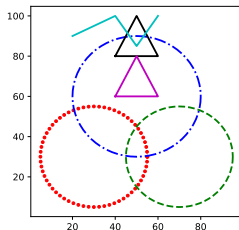
Shapely

● Prédicats topologiques

```

1 from shapely.geometry import LineString, Point, Polygon
2
3 red = Point((30, 30)).buffer(25)
4 green = Point((70, 30)).buffer(25)
5 blue = Point((50, 60)).buffer(30)
6 black = Polygon([(40, 80), (60, 80), (50, 100)])
7 magenta = Polygon([(40, 60), (60, 60), (50, 80)])
8 cyan = LineString([(20, 90), (40, 100), (50, 85), (60, 100)])
9
10 print(magenta.within(blue)) #~ True
11 print(blue.contains(magenta)) #~ True
12 print(red.difference(blue).intersects(green)) #~ True
13 print(magenta.touches(black)) #~ True
14 print(black.overlaps(blue)) #~ True
15 print(cyan.crosses(black)) #~ True
16 print(black.union(magenta).disjoint(green)) #~ True

```



Shapely

- A propos du **Dimensionally Extended 9-Intersection Model (DE-9IM)**

$$DE9IM(a, b) = \begin{bmatrix} \dim(I(a) \cap I(b)) & \dim(I(a) \cap B(b)) & \dim(I(a) \cap E(b)) \\ \dim(B(a) \cap I(b)) & \dim(B(a) \cap B(b)) & \dim(B(a) \cap E(b)) \\ \dim(E(a) \cap I(b)) & \dim(E(a) \cap B(b)) & \dim(E(a) \cap E(b)) \end{bmatrix}$$

- ▶ où \dim est la dimension de l'intersection (\cap) entre l'intérieur (I), la frontière (B) ou le complémentaire (E) des géométries a et b passées en paramètre.
- ▶ cette dimension prend la valeur F si l'intersection est l'ensemble vide (\emptyset), 0 si l'intersection est un ensemble de points, 1 si l'intersection est un ensemble de lignes et 2 si l'intersection est un ensemble de polygones.
- ▶ cette matrice peut être sérialisée, ligne après ligne, sous forme d'un *DE-9IM string code*. Ainsi, à partir de l'exemple précédent, l'instruction Shapely : `black.relate` (magenta) renvoie la chaîne de caractères : `FF2F01212`.

Shapely

● Opérations géométriques

```

1 green = Point((5,5)).buffer(5)
2 blue = Point((10, 5)).buffer(5)
3 _intersection = green.intersection(blue)
4 _union = green.union(blue)
5 _symmetric_difference = green.symmetric_difference(blue)
6 _difference = green.difference(blue)

```

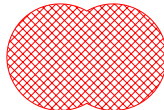
Données en entrée



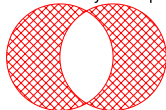
Intersection



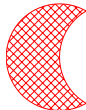
Union



Difference symétrique



Difference



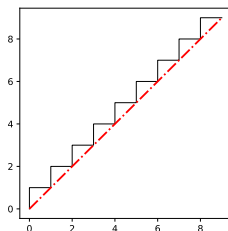
Shapely

● Simplification de géométrie

```

1 import matplotlib.pyplot as plt
2 from shapely.geometry import LineString
3
4 black = LineString([ (i//2, i-i//2) for i in range(19) ])
5 red = black.simplify(1.0, preserve_topology=True)
6
7 plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
8 plt.plot(*black.xy, 'k-', linewidth=1)
9 plt.plot(*red.xy, 'r-.', linewidth=2)
10 plt.savefig('../img/6_7_shapely.pdf')

```

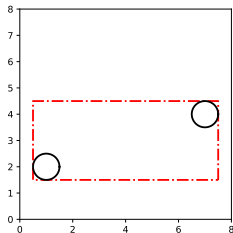


Shapely

- Boîte englobante (aussi appelée *bounding box*)

```

1 import matplotlib.pyplot as plt
2 from random import randint
3 from shapely.geometry import box, MultiPolygon, Point
4
5 pts = [Point(1,2).buffer(0.5), Point(7,4).buffer(0.5)]
6 pts = MultiPolygon(pts)
7
8 print(pts.bounds)
9 #~ (minx, miny, maxx, maxy) = (0.5, 1.5, 7.5, 4.5)
10 envelop = box(*pts.bounds)
11
12 plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
13 plt.xlim(0, 8)
14 plt.ylim(0, 8)
15 plt.plot(*envelop.exterior.xy, 'r--', linewidth=2)
16 for pt in pts:
17     plt.plot(*pt.exterior.xy, 'k-', linewidth=2)
18 plt.savefig('../img/6_8_shapely.pdf')
```



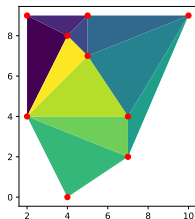
Shapely

- Triangulation via `shapely.ops.triangulate (...)`

```

1 from shapely.geometry import MultiPoint
2 from shapely.ops import triangulate
3
4 mpts = MultiPoint([(4,0), (7,2), (2,4), (7,4), \
5 (5,7), (4,8), (2,9), (5,9), (10,9)])
6 tin = triangulate(mpts) #~ retourne une liste de "Polygon"

```

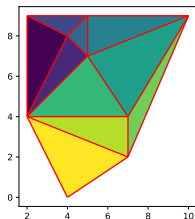


- Polygonisation via `shapely.ops.polygonize (...)`

```

1 from shapely.geometry import MultiLineString
2 from shapely.ops import polygonize
3
4 mls = MultiLineString([(2,9), (2,4)], [(2,4), (4,8)], \
5 [(4,8), (2,9)], [(4,8), (5,9)], [(5,9), (2,9)], [(4,8), (5,7)], \
6 [(5,7), (5,9)], [(5,7), (10,9)], [(10,9), (5,9)], [(5,7), (7,4)], \
7 [(7,4), (10,9)], [(7,4), (7,2)], [(7,2), (10,9)], [(4,0), (7,2)], \
8 [(7,2), (2,4)], [(2,4), (4,0)], [(7,4), (2,4)], [(5,7), (2,4)] ])
9 polygons = polygonize(mls) #~ retourne une liste de "Polygon"

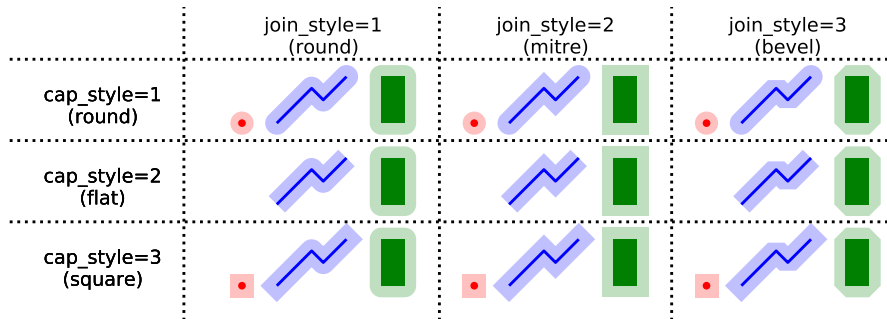
```



Shapely

- Dilatation, érosion via `buffer(...)`

► `Point((0, 0)).buffer(distance, resolution=16, cap_style=1, join_style=1, mitre_limit=5.0)`



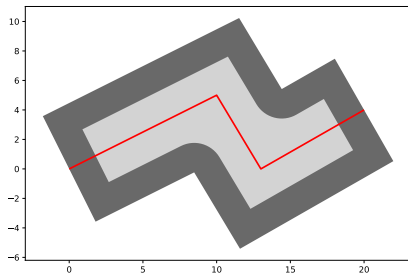
Shapely

- Dilatation, érosion via `buffer(...)`

```

1 from shapely.geometry import LineString
2 from shapely.geometry import CAP_STYLE, JOIN_STYLE
3
4 red = LineString([(0,0), (10, 5), (13, 0), (20, 4)])
5 #~ DILATATION
6 dimgrey = red.buffer(4, cap_style=CAP_STYLE.flat, join_style=JOIN_STYLE.mitre)
7 #~ EROSION
8 lightgrey = dimgrey.buffer(-2, join_style=JOIN_STYLE.round)

```



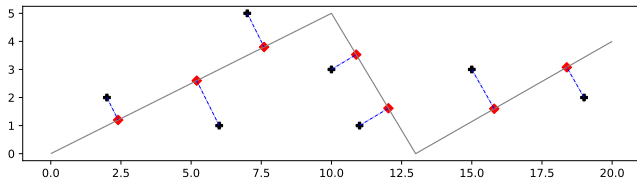
Shapely

- `linemerge(...)`, `project(...)` et `interpolate(...)`

```

1 from shapely.geometry import LineString, MultiPoint
2 from shapely.ops import linemerge
3
4 black = MultiPoint([(2, 2), (7, 5), (6, 1), (10, 3), (11, 1), (15, 3), (19, 2)])
5
6 lines = [LineString([ (0, 0), (10, 5) ]), LineString([ (10, 5), (13, 0) ]),
7           LineString([ (13, 0), (20, 4) ])]
8 grey = linemerge(lines) #- LINESTRING (0 0, 10 5, 13 0, 20 4)
9
10 red = []
11 for _point in black.geoms:
12     #- project(...) retourne une abscisse curviligne
13     #- interpolate(...) retourne un Point shapely
14     red.append( grey.interpolate(grey.project(_point)) )
15 red = MultiPoint(red)

```



Shapely

- Import-export **Well-Known Text (WKT)**

- ▶ Export au format WKT :

```
from shapely.geometry import LineString  
LineString([(0,0), (10,0), (10,10)]).wkt
```

- ▶ Import du format WKT :

```
from shapely import wkt  
wkt.loads('LINESTRING (0 0, 10 0, 10 10)')
```

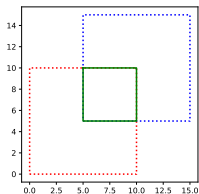

Shapely

- Attention, Shapely est une bibliothèque résolument 2D. La composante en Z des géométries est "décorative".

```

1 import matplotlib.pyplot as plt
2 from shapely.affinity import translate
3 from shapely.geometry import Polygon
4
5 red = Polygon([ (0,0,0), (10,0,0), (10,10,0),(0,10,0) ])
6 blue = translate(red, xoff=5.0, yoff=5.0, zoff=5.0)
7 green = red.intersection(blue)
8 print(red.intersects(blue)) #~ True
9 print(green.wkt)
10 #~ POLYGON Z ((5 10 2.5, 10 10 0, 10 5 2.5, 5 5 5, 5 10 2.5))
11
12 plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
13 plt.plot(*red.exterior.xy, 'r:', linewidth=2)
14 plt.plot(*blue.exterior.xy, 'b:', linewidth=2)
15 plt.plot(*green.exterior.xy, 'g-', linewidth=2)
16 plt.savefig('../img/6_d_shapely.pdf')

```



Plan de la présentation

- 1 Introduction à Pandas
- 2 Introduction à Shapely
- 3 Introduction à GeoPandas**
- 4 *t4gpd*

GeoPandas

- Extraits de <https://geopandas.org>
 - ▶ GeoPandas is an open source project to make working with geospatial data in python easier. GeoPandas extends the datatypes used by [pandas](#) to allow spatial operations on geometric types. Geometric operations are performed by [shapely](#). Geopandas further depends on [fiona](#) for file access and [descartes](#) and [matplotlib](#) for plotting.
 - ▶ The goal of GeoPandas is to make working with geospatial data in python easier. It combines the capabilities of pandas and shapely, providing geospatial operations in pandas and a high-level interface to multiple geometries to shapely. GeoPandas enables you to easily do operations in python that would otherwise require a spatial database such as PostGIS.

GeoPandas

- `geopandas.GeoDataFrame` est une sous-classe de `pandas.DataFrame` (de même que `geopandas.GeoSeries` est une sous-classe de `pandas.Series`)
- Chaque `GeoSeries` embarque son propre `crs`
- Un `GeoDataFrame` est une combinaison d'une ou plusieurs `Series` (données attributaires) et d'une ou plusieurs `GeoSeries` (données spatiales), accompagnée d'un mécanisme d'indexation

```

      colA  colB      geometry
0      12   34  POINT (12.00000 34.00000)
1      56   89  POINT (56.00000 89.00000)
2      23   45  POINT (23.00000 45.00000)
3      78   90  POINT (78.00000 90.00000)
  
```

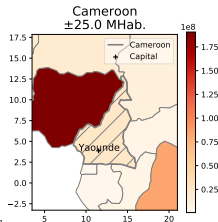
GeoPandas

Mise en carte GeoPandas + Matplotlib

```

1 import geopandas as gpd, matplotlib.pyplot as plt
2
3 countries = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
4 cities = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
5
6 africa = countries[ countries.continent == "Africa" ]
7 cameroon = africa.query('name == "Cameroon"')
8 yaounde = gpd.sjoin(cities, cameroon)
9 nHab = cameroon.pop_est.squeeze() / 1e6
10
11 _, ax = plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
12 ax.set_title('Cameroon\n±25.0 MHab.' % nHab, fontsize=16)
13
14 africa.plot(ax=ax, column='pop_est', cmap='OrRd',
15            edgecolor='gray', legend=True)
16 cameroon.boundary.plot(ax=ax, edgecolor='gray', hatch='/',
17                       linewidth=2, label='Cameroon')
18 yaounde.plot(ax=ax, color='black', marker='+', label='Capital')
19 yaounde.apply(lambda x: ax.annotate(
20             s=x.name_left, xy=x.geometry.coords[0],
21             color='black', size=12, ha='center'), axis=1);
22
23 minx, miny, maxx, maxy = cameroon.buffer(5.0).total_bounds
24 plt.axis([minx, maxx, miny, maxy])
25 plt.legend(loc='upper right', framealpha=0.5)
26 plt.savefig('../img/7_1_geopandas.pdf')

```



GeoPandas

- Un **GeoDataFrame** GeoPandas est un **DataFrame** pandas (presque) comme les autres...

```

1 import geopandas as gpd, numpy as np
2
3 countries = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
4
5 continents = countries.groupby(by='continent').\
6     pop_est.aggregate([np.size, np.sum]).\
7     rename(columns={'size': 'NPays', 'sum': 'NHab'}).\
8     sort_values('NHab', ascending=False)
9
10 print(continents)

```

```

Fichier  Édition  Affichage  Rechercher  Terminal  Aide
In [1]: %run 7_2_geopandas.py
          NPays          NHab
continent
Asia          47  4389144868
Africa        51  1219176238
Europe        39  746398461
North America 18  573042112
South America 13  418540749
Oceania        7   36782844
Antarctica     1    4050
Seven seas (open ocean) 1    140

In [2]: 

```

- avec un champ geometry qui embarque l'information spatiale
- avec une méthode plot et d'autres méthodes spécifiques

GeoPandas

- *Table-Oriented Programming* : manipulation de données (cf. [Comparison with SQL](#))

▶ `type(countries . pop_est) #~ pandas.core.series . Series`

▶ `type(countries . geometry) #~ geopandas.geoseries.GeoSeries`

▶ `SELECT continent, pop_est FROM countries WHERE countries LIKE 'Cameroon'`

▶ `countries [countries . name == 'Cameroon'] [['name', 'pop_est']]`

▶ `SELECT continent, pop_est FROM countries WHERE countries LIKE 'Cameroon' OR countries LIKE 'France'`

▶ `countries [countries . name . isin (['Cameroon', 'France'])] [['name', 'pop_est']]`

▶ `countries [~ countries . continent . isin (['Africa', 'Asia', 'Europe', \ 'North America', 'South America'])] [['name', 'continent']]`

▶ `sum(countries . pop_est)`

GeoPandas

- *Table-Oriented Programming* : manipulation de données

- ▶ `cities [cities .geometry.y == max(cities.geometry.y)]`

- ▶ `cities [abs(cities .geometry.y) == min(abs(cities.geometry.y))]`

- ▶ `cities [cities .name == 'Paris'].geometry.y.squeeze()`

- ▶ `countries [(countries .continent == 'Europe') & (countries.bounds.maxx < 5)].name`

- ▶ `north = cities [cities .geometry.y >= 0]`

- `south = cities [cities .geometry.y < 0]`

- `lesCapitales = south.append(north)`

- ▶ `countries .continent .unique()`

GeoPandas

- *Table-Oriented Programming* : manipulation de données

- ▶ Jointure attributaire : `pandas.merge(...)`

```

1 import geopandas as gpd, pandas as pd
2 from numpy import round
3
4 countries = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
5
6 continents = countries.groupby(by='continent').sum().\
7     rename(columns={'pop_est': 'NHab'})
8
9 result = pd.merge(countries, continents, how='left',\
10     left_on='continent', right_index=True, validate='m:1')
11 result['ratio'] = round(100 * result.pop_est / result.NHab, 1)
12 result.sort_values('ratio', ascending=False, inplace=True)
13
14 print(result[['continent', 'name', 'ratio']].head(7))

```

```

Fichier Édition Affichage Rechercher Terminal Aide
In [1]: %run 7_3_geopandas.py
continent name ratio
23 Seven seas (open ocean) Fr. S. Antarctic Lands 100.0
159 Antarctica Antarctica 100.0
137 Oceania Australia 63.2
4 North America United States of America 57.0
29 South America Brazil 49.5
139 Asia China 31.4
98 Asia India 29.2

In [2]: 

```

GeoPandas

- *Table-Oriented Programming* : manipulation de données

- ▶ Jointure spatiale : `geopandas.sjoin (...)`

```

1 import geopandas as gpd
2
3 countries = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
4 cities = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
5
6 #~ op: binary predicate, one of {'intersects', 'contains', 'within'}
7 result = gpd.sjoin(cities, countries, how='left', op='intersects',
8                   lsuffix='city', rsuffix='country')
9 result.sort_values('pop_est', ascending=True, inplace=True)
10 result.rename(columns={'pop_est': 'NHab_country'}, inplace=True)
11
12 print(result[['name_city', 'continent', 'name_country', 'NHab_country']].head(7))

```

```

Fichier  Édition  Affichage  Rechercher  Terminal  Aide
In [1]: %run 7_4_geopandas.py
202 202
      name_city      continent name_country  NHab_country
132  Nicosia         Asia        N. Cyprus    265100.0
46   Reykjavik      Europe       Iceland    339747.0
105  Belmopan       North America Belize     360346.0
103  Bandar Seri   Asia         Brunei     443593.0
49   Paramaribo    South America Suriname   591919.0
3    Luxembourg    Europe       Luxembourg 594130.0
19   Podgorica     Europe       Montenegro 642550.0
In [2]: 

```

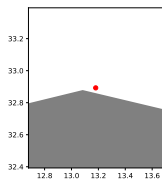
GeoPandas

- *Table-Oriented Programming* : manipulation de données
 - ▶ Jointure spatiale : `geopandas.sjoin (...)`

```

1 import geopandas as gpd, matplotlib.pyplot as plt
2
3 countries = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
4 cities = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
5
6 result = gpd.sjoin(cities, countries, how='left', op='intersects',
7                   lsuffix='city', rsuffix='country')
8 print('Nombre de capitales sans pays de rattachement : %d' %
9       len(result[result.name_country.isnull()]))
10 #~ Nombre de capitales sans pays de rattachement : 30
11
12 countries = countries[countries.name == 'Libya']
13 cities = cities[cities.name == 'Tripoli']
14
15 _, ax = plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
16 ax = countries.plot(ax=ax, color='grey')
17 cities.plot(ax=ax, color='red', marker='o')
18 minx, miny, maxx, maxy = cities.buffer(0.5).total_bounds
19 plt.axis([minx, maxx, miny, maxy])
20 plt.savefig('../img/7_5_geopandas.pdf')

```



GeoPandas

- *Table-Oriented Programming* : manipulation de données

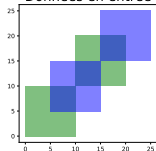
- ▶ Overlay : `geopandas.overlay(...)`

```

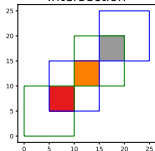
1 green1 = Polygon([ (0,0), (10,0), (10,10), (0,10) ])
2 green2 = Polygon([ (10,10), (20,10), (20,20), (10,20) ])
3 blue1 = Polygon([ (5,5), (15,5), (15,15), (5,15) ])
4 blue2 = Polygon([ (15,15), (25,15), (25,25), (15,25), (15,15) ])
5 green = gpd.GeoDataFrame( [{"geometry": green1}, {"geometry": green2} ] )
6 blue = gpd.GeoDataFrame( [{"geometry": blue1}, {"geometry": blue2} ] )
7
8 _intersection = gpd.overlay(green, blue, how='intersection')
9 _union = gpd.overlay(green, blue, how='union')
10 _symmetric_difference = gpd.overlay(green, blue, how='symmetric_difference')
11 _difference = gpd.overlay(green, blue, how='difference')

```

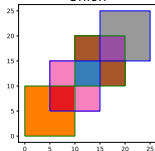
Donnees en entree



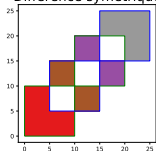
Intersection



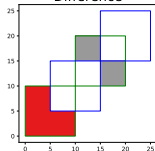
Union



Difference symetrique



Difference



GeoPandas

- *Table-Oriented Programming* : manipulation de données

- ▶ Overlay : `geopandas.overlay (...)`

```

1 from geopandas import GeoDataFrame, overlay
2 from shapely.geometry import Polygon
3
4 green1 = Polygon([(0,0), (10,0), (10,10), (0,10)])
5 green2 = Polygon([(10,10), (20,10), (20,20), (10,20)])
6 blue1 = Polygon([(5,5), (15,5), (15,15), (5,15)])
7 blue2 = Polygon([(15,15), (25,15), (25,25), (15,25), (15,15)])
8 green = GeoDataFrame([{'geometry': green1, 'id': 'g1'},
9                       {'geometry': green2, 'id': 'g2'}])
10 blue = GeoDataFrame([{'geometry': blue1, 'id': 'b1'},
11                      {'geometry': blue2, 'id': 'b2'}])
12
13 print( overlay(green, blue, how='intersection') )

```

```

Fichier  Édition  Affichage  Rechercher  Terminal  Aide
In [1]: %run 7_7_geopandas.py
   id_1 id_2 geometry
0  g1  b1  POLYGON ((5.00000 10.00000, 10.00000 10.00000,...
1  g2  b1  POLYGON ((10.00000 10.00000, 10.00000 15.00000...
2  g2  b2  POLYGON ((15.00000 20.00000, 20.00000 20.00000...

In [2]: 

```

GeoPandas

- Manipulation de données : `geopandas.clip` (...)

```

1 import geopandas as gpd, matplotlib.pyplot as plt
2 from shapely.geometry import box
3 from os import path
4 if path.exists('c:/Users/tleduc'):
5     HOMEDIR = 'c:/Users/tleduc'
6 elif path.exists('/home/tleduc'):
7     HOMEDIR = '/home/tleduc'
8 bdtopo = HOMEDIR + '/data/bdtopo/\
9 BDTOPO_3-0_TOUSTHEMES_SHP_LAMB93_D044_2020-06-15/BDTOPO\
10 /1_DONNEES_LIVRAISON_2020-06-00047\
11 /BDT_3-0_SHP_LAMB93_D044-ED2020-06-15/BATI'
12
13 roi = 355019.3, 6689077.3, 355311.4, 6689528.0
14 red = gpd.read_file(bdtopo + '/BATIMENT.shp', bbox=roi)
15
16 roi = box(*roi)
17 grey = gpd.clip(red, roi)
18
19 _, basemap = plt.subplots(figsize=(0.5*8.26, 0.5*8.26))
20 plt.axis('off')
21 red.plot(ax=basemap, color='red')
22 grey.plot(ax=basemap, color='grey')
23 plt.savefig('../img/7_8_geopandas.pdf', bbox_inches='tight')

```



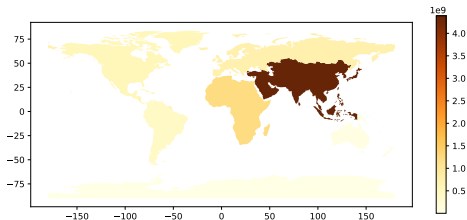
GeoPandas

- Manipulation de données : agrégation de géométries via `dissolve (...)`

```

1 import geopandas as gpd, matplotlib.pyplot as plt
2 from shapely.ops import unary_union
3
4 countries = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
5
6 continents = countries[['continent', 'pop_est', 'geometry']].\
7     dissolve(by='continent', aggfunc='sum')
8
9 _, basemap = plt.subplots(figsize=(1.2*8.26, 0.5*8.26))
10 #~ continents.reset_index(inplace=True)
11 #~ continents.plot(ax=basemap, column='continent', cmap='tab10')
12 continents.plot(ax=basemap, column='pop_est', cmap='YlOrBr', legend=True)
13 plt.savefig('../img/7_9_geopandas.pdf', bbox_inches='tight')

```



- l'argument `cmap` de la méthode `plot` permet de spécifier la palette de couleurs choisie.

GeoPandas

- *Table-Oriented Programming* : modification de données

- ▶ `countries.drop(columns=['iso_a3', 'gdp_md_est'], inplace=True)`
`cities.drop(cities [(abs(cities.geometry.x) > 5)].index, inplace=True)`
- ▶ `countries.rename(columns={'pop_est': 'NHab.', 'name': 'NomPays'}, inplace=True)`
- ▶ `from shapely.geometry import Point`
`cities.append({'name': 'Nantes', 'geometry': Point((47.2173, -1.5534))}, ignore_index = True)`
- ▶ `cities ['hemisphere'] = cities.geometry.apply(lambda geom: 'Nord' if (geom.y > 0) else 'Sud')`
- ▶ `result.NHab_country.dtypes` #~ de type float64
`result.NHab_country.astype(int)` #~ Cannot convert non-finite values (NA or inf) to integer
`result.NHab_country.fillna(-9999, inplace=True)` #~ remplacement des valeurs NaN par -9999
`result.NHab_country = result.NHab_country.astype(int)`

GeoPandas

- Création de `GeoDataFrame`, ajout d'un champ et modification de la référence au champ géométrique

```

1 import geopandas as gpd, matplotlib.pyplot as plt
2 from shapely.geometry import Point
3
4 gdf = gpd.GeoDataFrame([ { 'geometry': Point((0, i)), 'gid': i }
5   for i in range(0,50,10)], crs='epsg:2154')
6 gdf['disc'] = gdf.buffer(2, resolution=1)
7 gdf.set_geometry('disc', inplace=True)
8 gdf.plot(column='gid')
9 plt.savefig('../img/7_a_geopandas.pdf', bbox_inches='tight')

```



- Modification du système de coordonnées

```

▶ import geopandas as gpd
   cities = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
   cities = cities[cities.name == 'Paris']
   cities.crs #~ EPSG:4326 – WGS 84
   cities = cities.to_crs('epsg:2154')
   cities.crs #~ EPSG:2154 – RGF93 / Lambert-93

```

GeoPandas

● Entrées-sorties (*)

- ▶ Chargement de SHP (l'usage d'un filtre – cf. arguments **bbox** ou **mask** – peut substantiellement accélérer le chargement)

- `import geopandas as gpd`

```
myGrid = gpd.read_file('data/ville_numerique/insee/\nFilosofi2015_carreaux_200m_metropole/Filosofi2015_carreaux_200m_metropole.shp')
```

- ▶ Ecriture de SHP (mais aussi GeoJSON, GPKG)

- `myGrid.to_file('data/grid200.shp', driver='ESRI Shapefile')`

- ▶ Ecriture de CSV (avec encodage WKT des géométries)

- `myGrid.to_csv('data/grid200.csv')`

GeoPandas

- GeoPandas en quelques liens

- ▶ <https://geopandas.org/>

- ▶ <https://geopandas.readthedocs.io/en/latest/>

- ▶ <https://automating-gis-processes.github.io/>

Plan de la présentation

- 1 Introduction à Pandas
- 2 Introduction à Shapely
- 3 Introduction à GeoPandas
- 4 *t4gpd*

- Cinq registres de forme (Lévy, 2005) :
 - ① l'approche de la forme urbaine comme **forme du paysage urbain** - l'espace saisi visuellement dans sa tridimensionnalité et dans sa matérialité plastique,
 - ② l'approche de la forme urbaine comme **forme sociale** - l'espace étudié dans son occupation par les divers groupes sociaux ou la distribution des activités et fonctions dans la ville,
 - ③ l'approche de la forme urbaine comme **forme bioclimatique** - l'espace étudié dans sa dimension environnementale, microclimatique (héliothermique, écologique, etc.),
 - ④ l'approche de la forme urbaine comme **forme des tissus** - étude des interrelations parcellaire/viaire/libre/bâti, formes de mobilité/formes de ville,
 - ⑤ l'approche de la forme urbaine comme **forme des tracés** - forme du plan (organique/planifié, orthogonal/radioconcentrique).

t4gpd

- L'outil t4gpd permet d'analyser les formes d'espace construit dans différents registres, de **l'analyse bioclimatique** (orientation héliothermique, vue du ciel, etc.), à **l'analyse de tracés** (orientations, distances sur un graphe, etc.), en passant par des **analyses à connotations paysagères** (visibilités, études d'alignements d'arbres, etc.) ou des **analyses de composantes des tissus urbains** (identification de rues canyons, typologie d'intersections, etc.).
- Développé au sein de AAU-CRENAU, il bénéficie autant d'un ensemble de travaux conduits depuis plusieurs décennies à l'école nationale supérieure d'architecture de Nantes, que des développements récents autour de bibliothèques telles que GeoPandas ou Shapely.

t4gpd

- *t4gpd* s'inscrit dans une « filiation ». Il hérite d'un ensemble de connaissances développées dans :
 - ▶ le plugin SketchUp *t4su*
 - ▶ les scripts *t4qg* utilisables en console Python de QGIS 2 et 3
 - ▶ mais aussi *Solene*, GearScape, OrbisGIS, etc.
- C'est un plugin Python 3 qui se télécharge sur [SourceSup](#) et se déploie, dans le contexte d'un environnement [Miniconda3](#) (notamment), via un `: pip install t4gpd-0.2.0.tar.gz`
- Pour plus de détails concernant son installation, rendez-vous sur <https://github.com/crenau/t4gpd>.

t4gpd

- <https://github.com/crenau/t4gpd>
- <https://t4gpd-docs.readthedocs.io>

