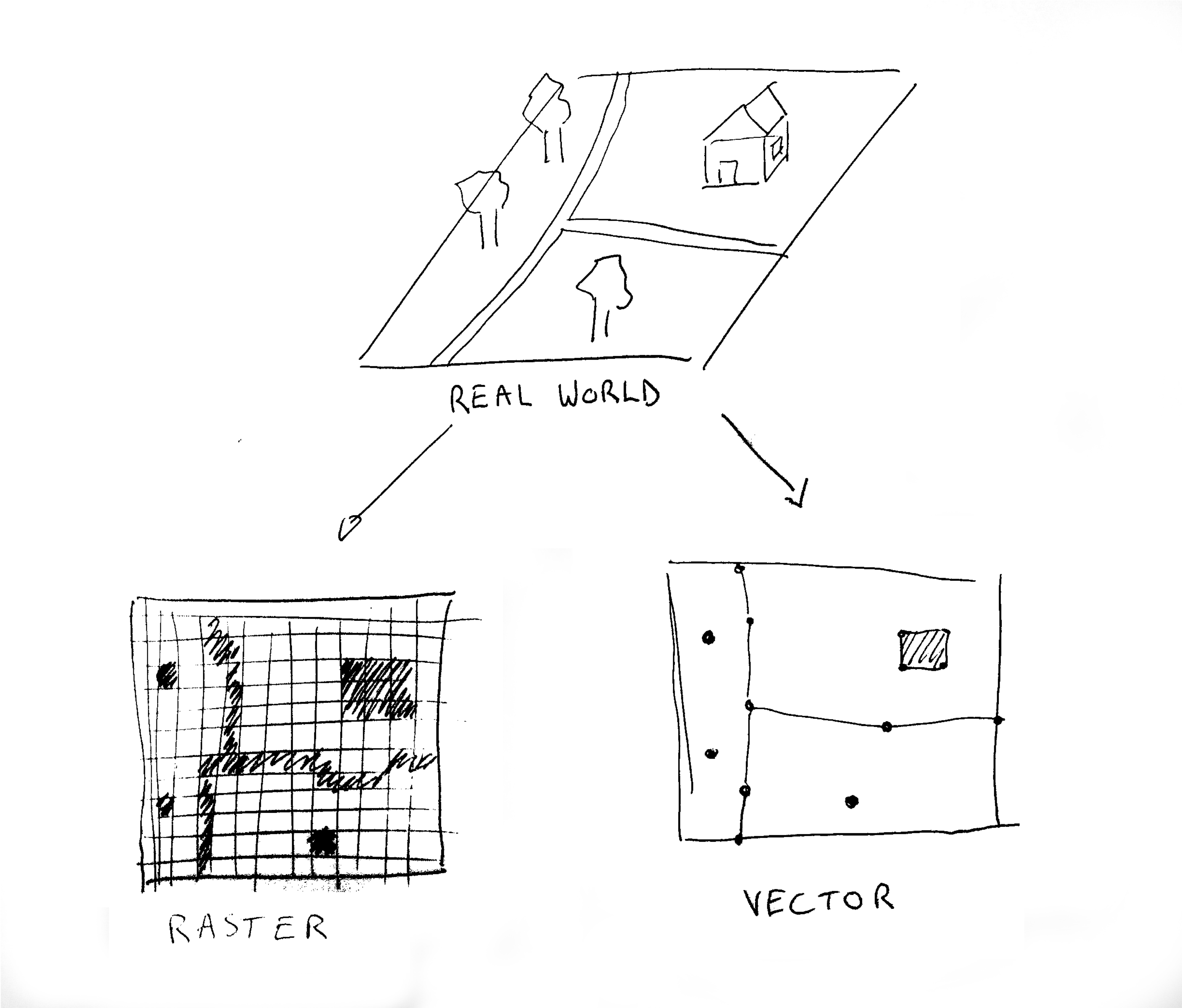
Geospatial Data

WORKING WITH GEOSPATIAL DATA IN PYTHON

Instructors

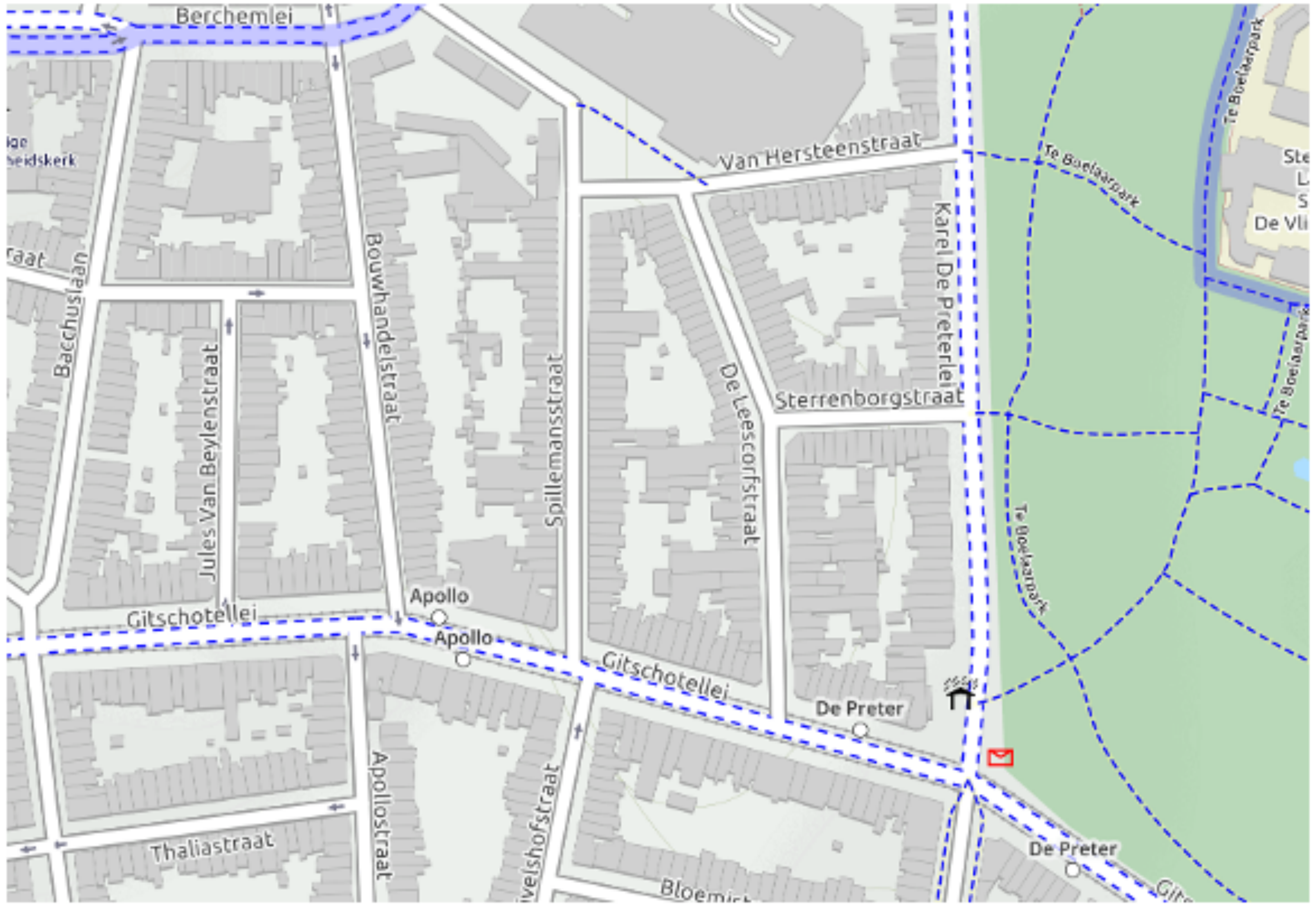
Joris Van den Bossche & Dani Arribas-Bel

How we record the real world



Raster versus vector data

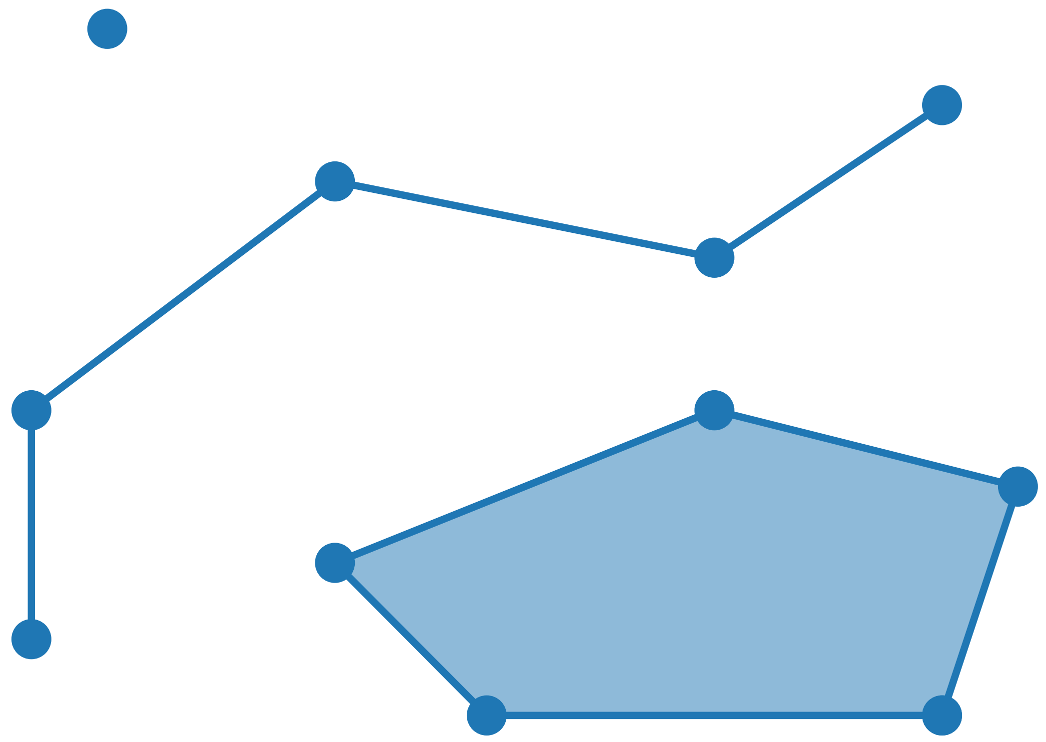
Raster



Vector

Vector features

"Discrete" representations that turn the world into:

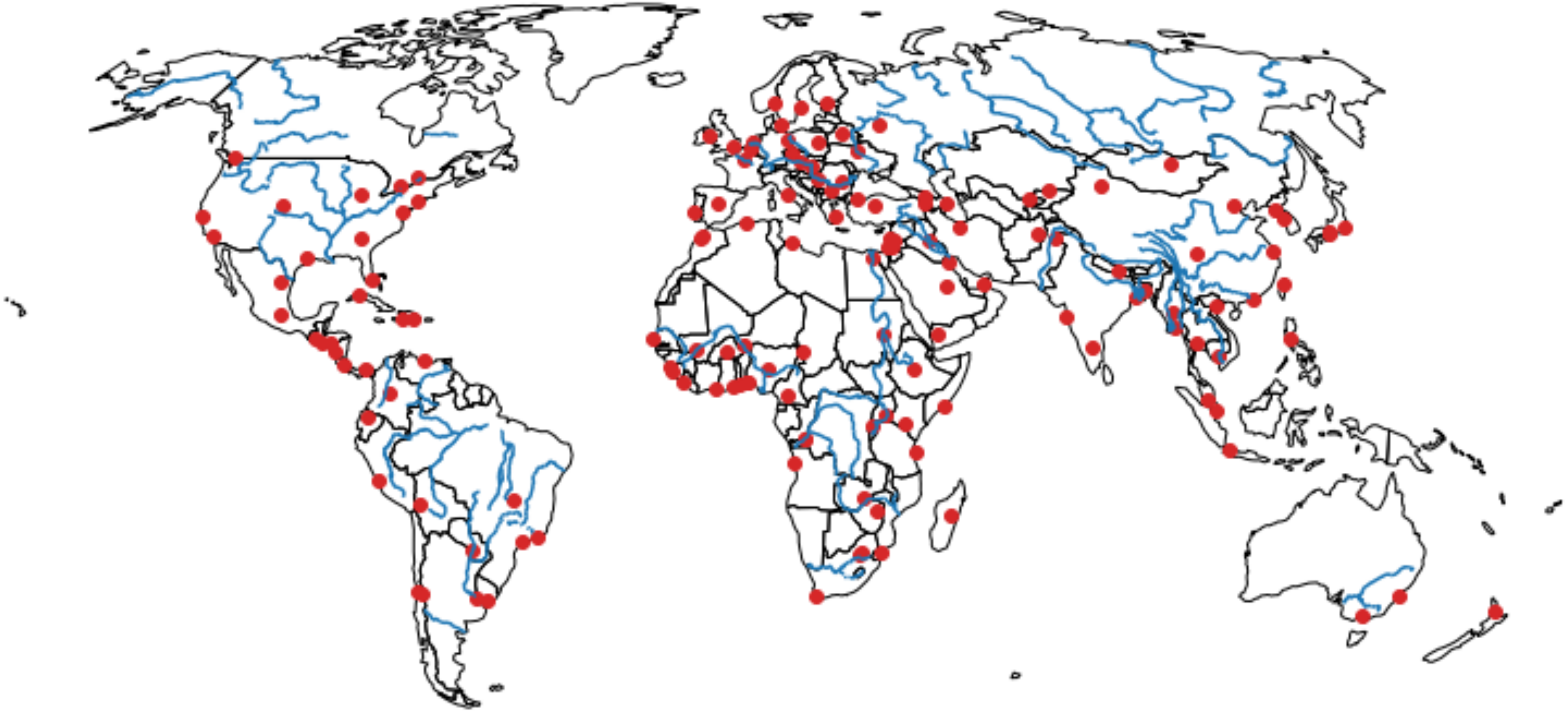


Point(2, 10)

LineString([(1, 2), (1, 5), ...])

Polygon([(13, 1), (14, 4), ...])

Feature consisting of multiple geometries: eg `MultiPolygon`



Vector attribute data

Vector features can have information associated that describe them: **attributes**

Tabular vector data:

	name	capital	population	geometry
0	Afghanistan	Kabul	34124811.0	POLYGON ((61.21081709172574 35.65007233330923,...
1	Angola	Luanda	29310273.0	(POLYGON ((23.90415368011818 -11.7222815894063...
2	Albania	Tirana	3047987.0	POLYGON ((21.0200403174764 40.84272695572588, ...
...	...	...	...	...
174	South Africa	Cape Town	54841552.0	POLYGON ((19.89576785653443 -24.76779021576059...
175	Zambia	Lusaka	15972000.0	POLYGON ((23.21504845550606 -17.52311614346598...
176	Zimbabwe	Harare	13805084.0	POLYGON ((29.43218834810904 -22.09131275806759...

Spatial specific data formats

```
restaurants = pd.read_csv("datasets/paris_restaurants.csv")
restaurants.head()
```

```
                                    type         x          y
0                    Restaurant européen  259641.6  6251867.4
1          Restaurant traditionnel français  259572.3  6252030.2
2          Restaurant traditionnel français  259657.2  6252143.8
3  Restaurant indien, pakistanais et Moyen Orient  259684.4  6252203.6
4          Restaurant traditionnel français  259597.9  6252230.0
```

In the rest of the course:

- spatial file formats (Shapefiles, GeoJSON, GeoPackage, ...)
- GeoPandas: pandas dataframes with support for spatial data

Importing geospatial data with GeoPandas

```
import geopandas
```

```
countries = geopandas.read_file("countries.geojson")
```

```
countries.head()
```

```
         name      continent       gdp                            geometry
0  Afghanistan          Asia   64080.0  POLYGON ((61.21 35.65, 62.23 35...
1       Angola        Africa  189000.0  MULTIPOLYGON (((23.90 -11.72, 2...
2      Albania        Europe   33900.0  POLYGON ((21.02 40.84, 21.00 40...
4    Argentina South America  879400.0  MULTIPOLYGON (((-66.96 -54.90, ...
5      Armenia          Asia   26300.0  POLYGON ((43.58 41.09, 44.97 41...
```

The GeoDataFrame

```
countries.head()
```

```
          name      continent       gdp                                      geometry
0  Afghanistan           Asia   64080.0  POLYGON ((61.21 35.65, 62.23 35...
1       Angola         Africa  189000.0  MULTIPOLYGON (((23.90 -11.72, 2...
2      Albania         Europe   33900.0  POLYGON ((21.02 40.84, 21.00 40...
...
```

A GeoDataFrame represents a tabular, geospatial vector dataset:

- a **'geometry' column**: that holds the geometry information
- other columns: **attributes** describe each of the geometries

Spatial aware DataFrame

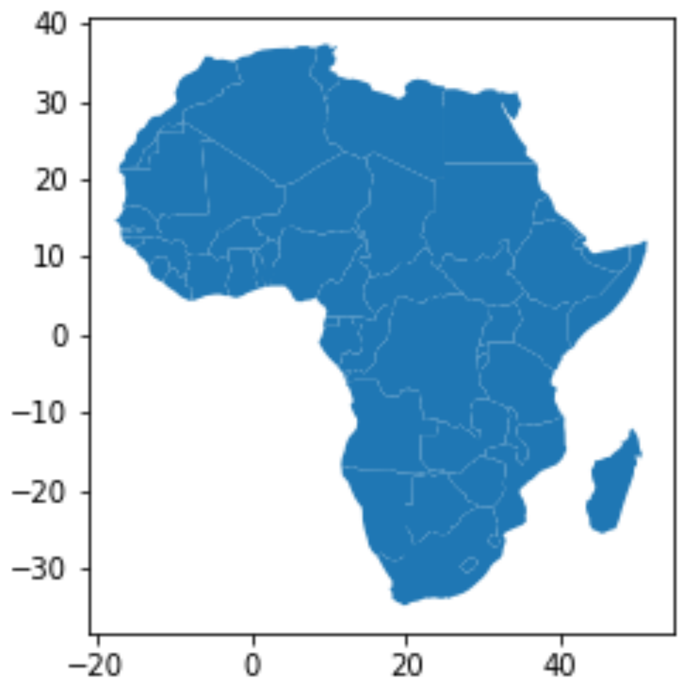
```
countries.geometry.area
```

```
0        63.593500
1       103.599439
2         3.185163
            ...
174     112.718524
175      62.789498
176      32.280371
Length: 177, dtype: float64
```

Filtering data

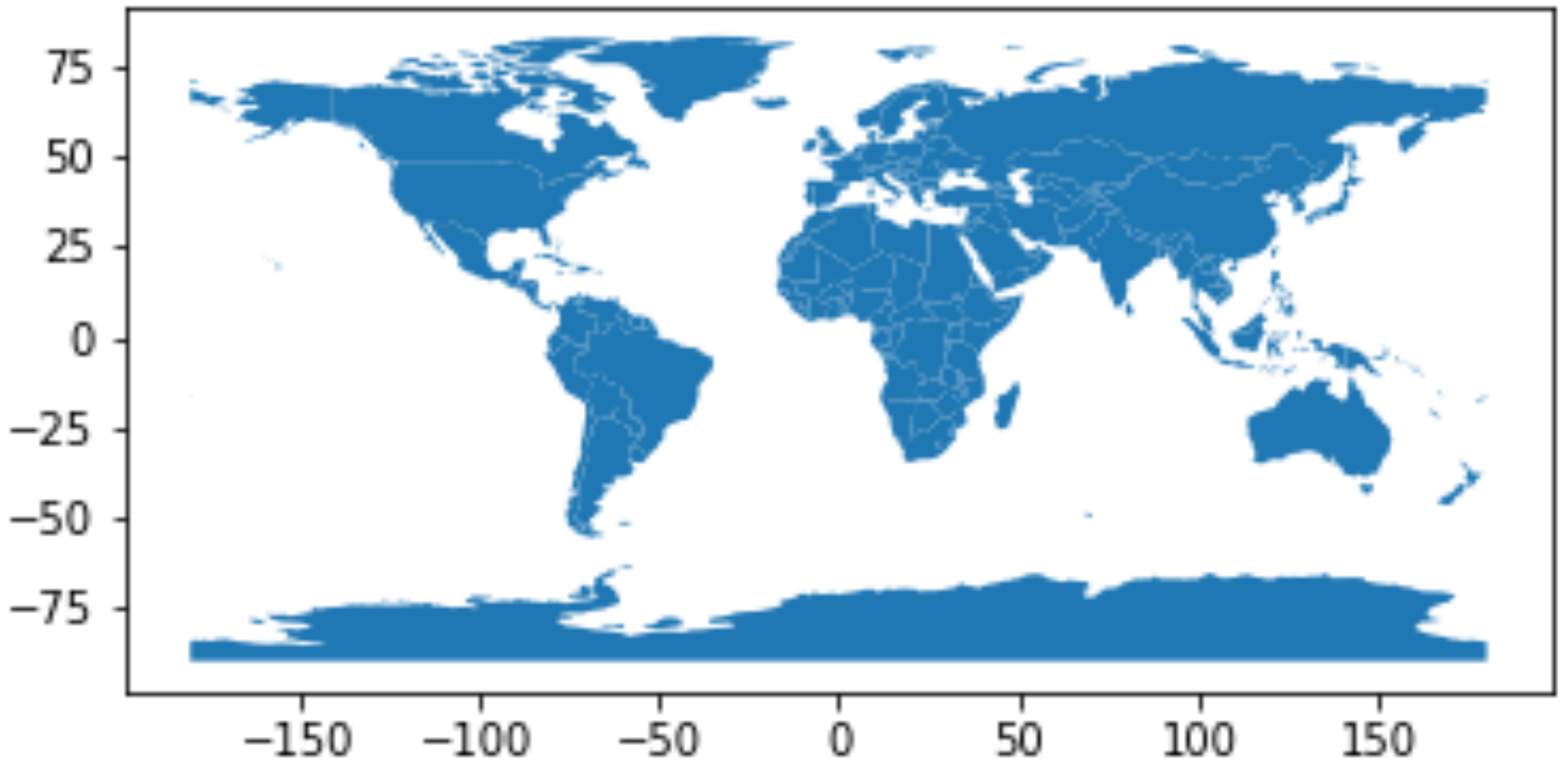
```
countries_africa = countries[countries['continent'] == 'Africa']
```

```
countries_africa.plot()
```



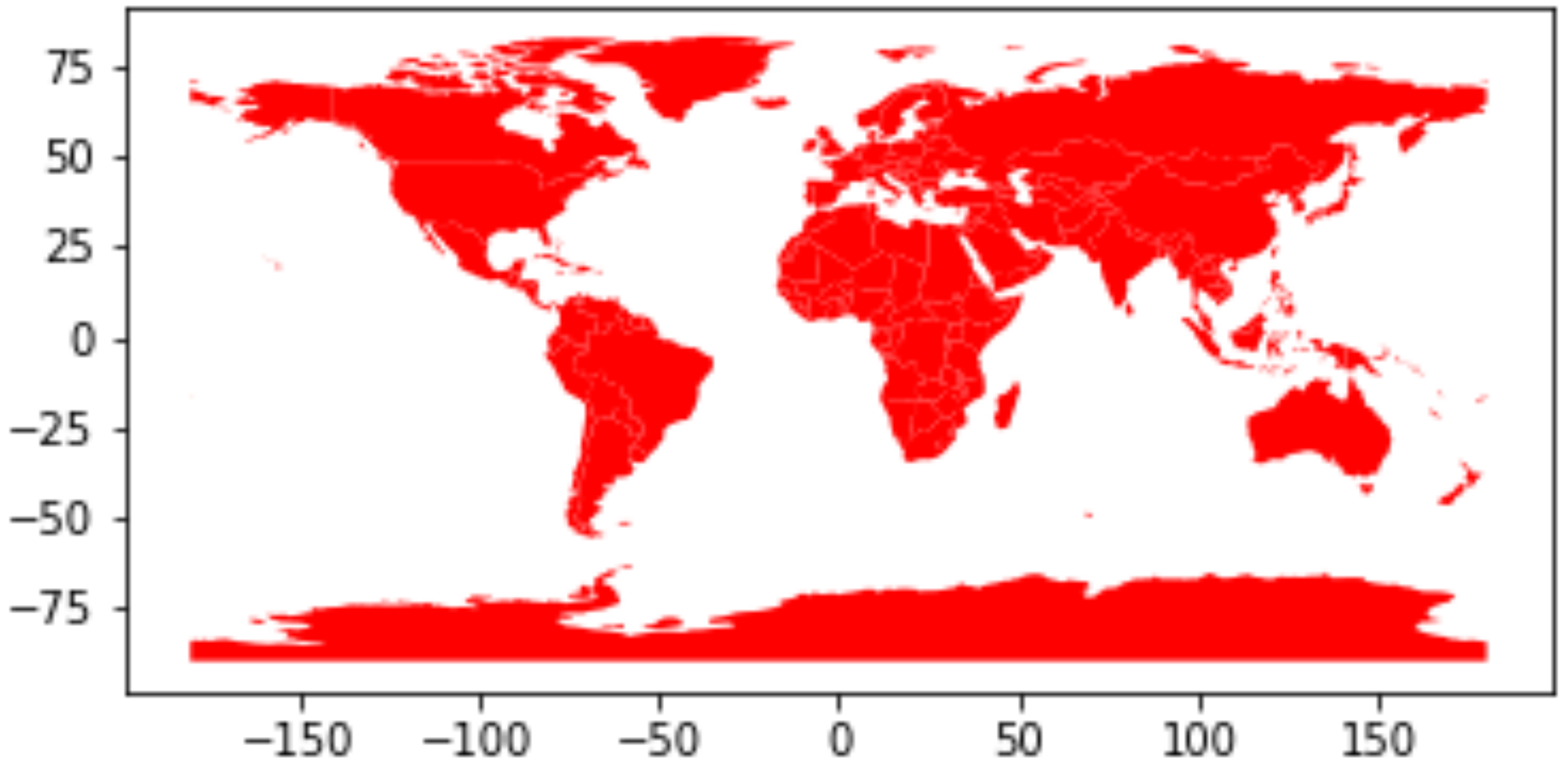
Visualizing spatial data

```
countries.plot()
```



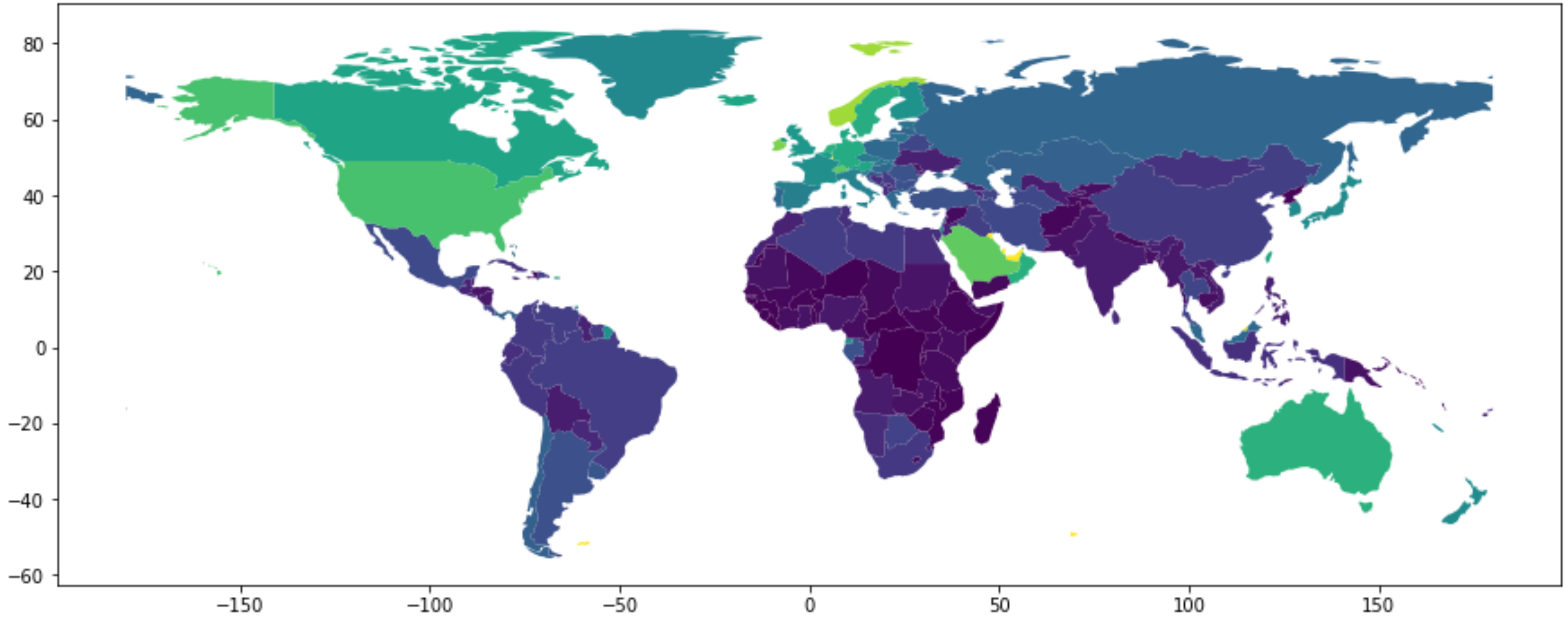
Adjusting the color: uniform color

```
countries.plot(color="red")
```



Adjusting the color: based on attribute values

```
countries.plot(column='gdp_per_cap')
```



Multi-layered plot

```
fig, ax = plt.subplots(figsize=(12, 6))
countries.plot(ax=ax)
cities.plot(ax=ax, color='red', markersize=10)
ax.set_axis_off()
```

Scalar geometry values

```
cities = geopandas.read_file("ne_110m_populated_places.shp")
cities.head()
```

```
            name                                   geometry
0  Vatican City   POINT (12.45338654497177 41.90328217996012)
1    San Marino     POINT (12.44177015780014 43.936095834768)
2         Vaduz   POINT (9.516669472907267 47.13372377429357)
3       Lobamba  POINT (31.19999710971274 -26.46666746135247)
4    Luxembourg   POINT (6.130002806227083 49.61166037912108)
```

```
brussels = cities.loc[170, 'geometry']
print(brussels)
```

```
POINT (4.33137074969045 50.83526293533032)
```

The Shapely python package

```
type(brussels)
```

```
shapely.geometry.point.Point
```

Shapely

- Python Package for the manipulation and analysis of geometric objects
- Provides the `Point` , `LineString` and `Polygon` objects
- GeoSeries (GeoDataFrame 'geometry' column) consists of shapely objects

Geometry objects

Accessing from a GeoDataFrame:

```
brussels = cities.loc[170, 'geometry']
paris = cities.loc[235, 'geometry']
belgium = countries.loc[countries['name'] == 'Belgium', 'geometry'].squeeze()
france = countries.loc[countries['name'] == 'France', 'geometry'].squeeze()
uk = countries.loc[countries['name'] == 'United Kingdom', 'geometry'].squeeze()
```

Creating manually:

```
from shapely.geometry import Point
p = Point(1, 2)
print(p)
```

```
POINT (1 2)
```

Spatial methods

The **area** of a geometry:

```
belgium.area
```

```
3.8299974609075753
```

The **distance** between 2 geometries:

```
brussels.distance(paris)
```

```
2.8049127723186214
```

And many more! (e.g. `centroid` , `simplify` , ...)

Spatial relationships

```
belgium.contains(brussels)
```

```
True
```

```
france.contains(brussels)
```

```
False
```

```
brussels.within(belgium)
```

```
True
```

```
belgium.touches(france)
```

```
True
```

```
line.intersects(france)
```

```
True
```

```
line.intersects(uk)
```

```
False
```

Element-wise spatial relationship methods

The `within()` operation for each geometry in `cities` :

```
cities.within(france)
```

```
0       False
1       False
2       False
        ...
240     False
241     False
242     False
Length: 243, dtype: bool
```

```
cities['geometry'][0].within(france)
```

```
False
```

```
cities['geometry'][1].within(france)
```

```
False
```

```
cities['geometry'][2].within(france)
```

```
False
```

...

Filtering by spatial relation

Filter `cities` depending on the `within()` operation:

```
cities[cities.within(france)]
```

```
        name                                   geometry
10    Monaco  POINT (7.406913173465057 43.73964568785249)
13   Andorra    POINT (1.51648596050552 42.5000014435459)
235    Paris   POINT (2.33138946713035 48.86863878981461)
```

Filtering by spatial relation

Which countries does the Amazon flow through?

```
rivers = geopandas.read_file("ne_50m_rivers_lake_centerlines.shp")
rivers.head()
```

```
             type     name                                   geometry
0  Lake Centerline     Kama  LINESTRING (51.94 55.70, 51.88 55.69...
1            River     Kama  LINESTRING (53.69 58.21, 53.68 58.27...
2  Lake Centerline     Abay  LINESTRING (37.11 11.85, 37.15 11.89...
...
```

```
amazon = rivers[rivers['name'] == 'Amazonas'].geometry.squeeze()
mask = countries.intersects(amazon)
```

Filtering by spatial relation

```
countries[mask]
```

```
          name      continent                                    geometry
22      Brazil  South America  POLYGON ((-57.63 -30.22, -56.29 -28....
35    Colombia  South America  POLYGON ((-66.88 1.25, -67.07 1.13, ...
124       Peru  South America  POLYGON ((-69.53 -10.95, -68.67 -12....
```

- `within`
- `contains`
- `intersects`

More at **https://shapely.readthedocs.io/en/latest/**

Shapely objects

```
paris.within(france)
```

```
True
```

```
france.intersects(amazon)
```

```
False
```

GeoPandas

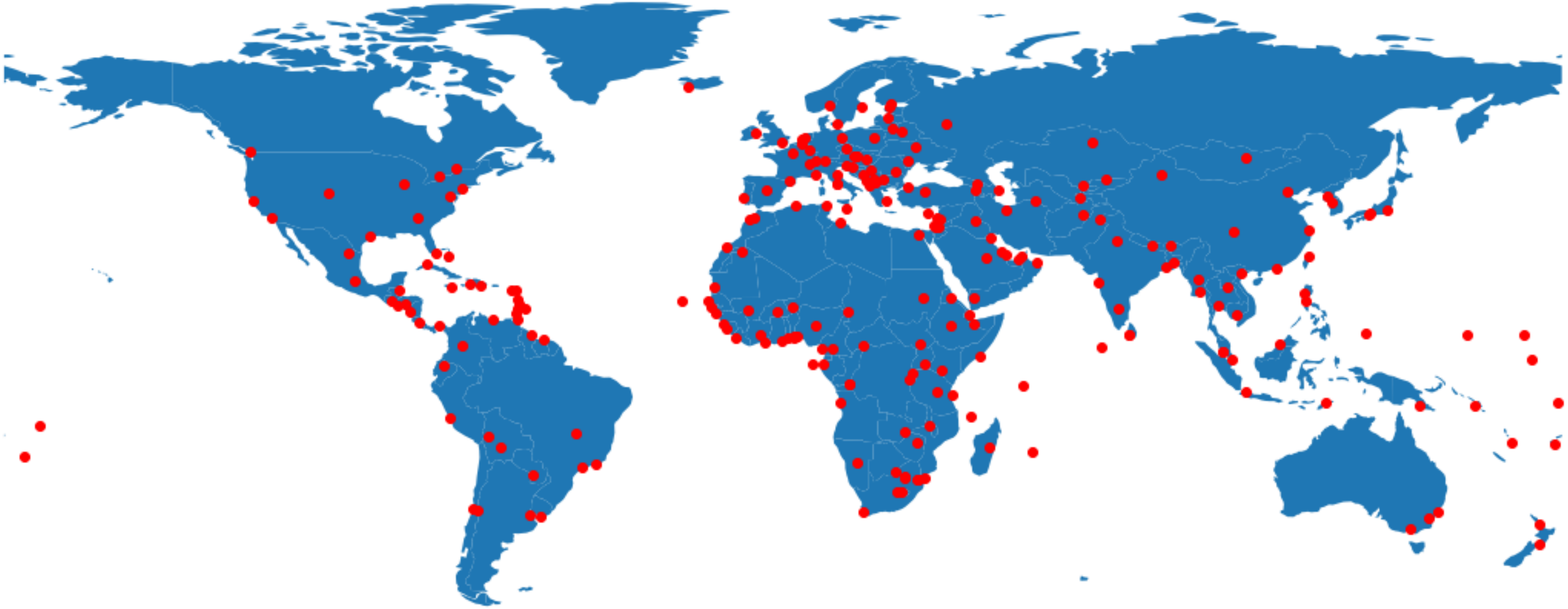
```
cities.within(france)
```

```
0       False
1       False
2       False
        ...
```

```
countries.intersects(amazon)
```

```
0       False
1       False
2       False
        ...
```

Spatial relationships I



Spatial relationships II

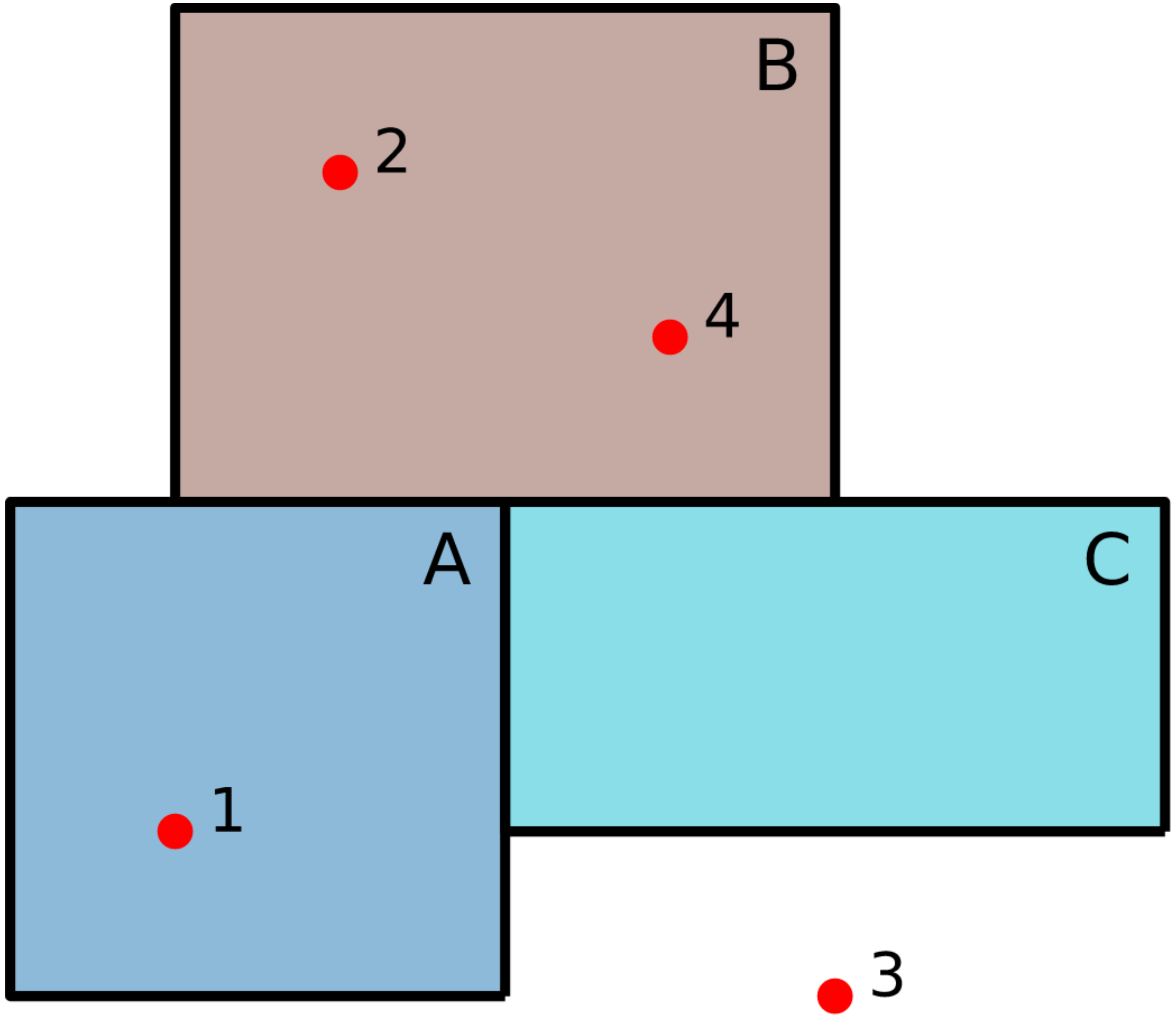
Which cities are located within Brazil?

```
brazil = countries.loc[22, 'geometry']
cities[cities.within(brazil)]
```

```
                name                                     geometry
169         Brasília  POINT (-47.91799814700306 -15.78139437287899)
238   Rio de Janeiro  POINT (-43.22696665284366 -22.92307731561596)
239        São Paulo  POINT (-46.62696583905523 -23.55673372837896)
```

But what if we want to know for each city in which country it is located?

The Spatial Join



points	geometry
1	POINT (2 2)
2	POINT (3 6)
3	POINT (6 1)
4	POINT (5 5)

←

polygon
A
B
nan
B

SPATIAL JOIN = *transferring attributes from one layer to another based on their spatial relationship*

The spatial join with GeoPandas

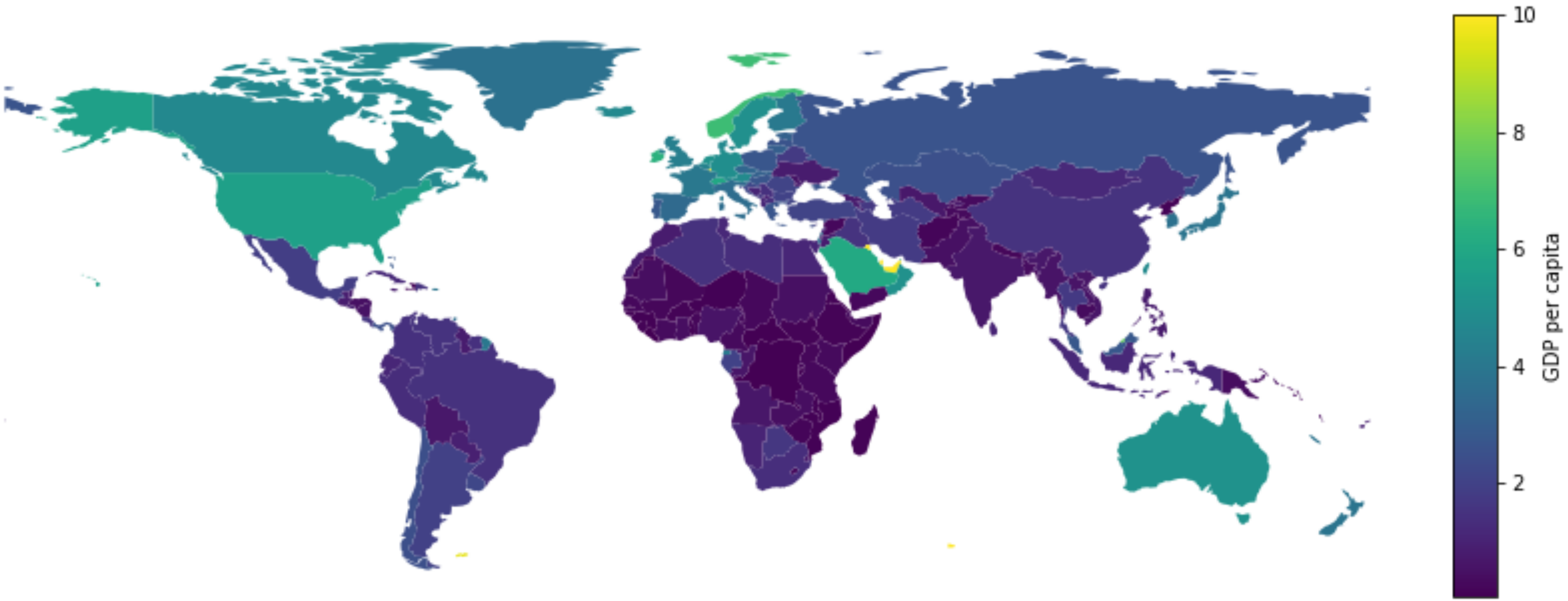
```
joined = geopandas.sjoin(cities,
                         countries[['name', 'geometry']],
                         op="within")
```

```
joined.head()
```

```
       name_left                                   geometry name_right
0    Vatican City  POINT (12.45338654497177 41.90328217996012)      Italy
1      San Marino    POINT (12.44177015780014 43.936095834768)      Italy
226          Rome    POINT (12.481312562874 41.89790148509894)      Italy
2           Vaduz  POINT (9.516669472907267 47.13372377429357)    Austria
212        Vienna  POINT (16.36469309674374 48.20196113681686)    Austria
```

Choropleths

```
countries.plot(column='gdp_per_cap', legend=True)
```



Choropleths

Specifying a column:

```
locations.plot(column='variable')
```

Choropleth with classification scheme:

```
locations.plot(column='variable', scheme='quantiles', k=7, cmap='viridis')
```

Key choices:

- Number of classes (`k`)
- Classification algorithm (`scheme`)
- Color palette (`cmap`)

Number of classes ("k")

```
locations.plot(column='variable', scheme='Quantiles', k=7, cmap='viridis')
```

Choropleths necessarily imply **information loss** (but that's OK)

Tension between:

- Maintaining **detail** and granularity from original values (higher `k`)
- **Abstracting** information so it is easier to process and interpret (lower `k`)

Rule of thumb: 3 to 12 classes or "bins"

Classiffication algorithms ("scheme")

```
locations.plot(column='variable', scheme='quantiles', k=7, cmap='viridis')
```

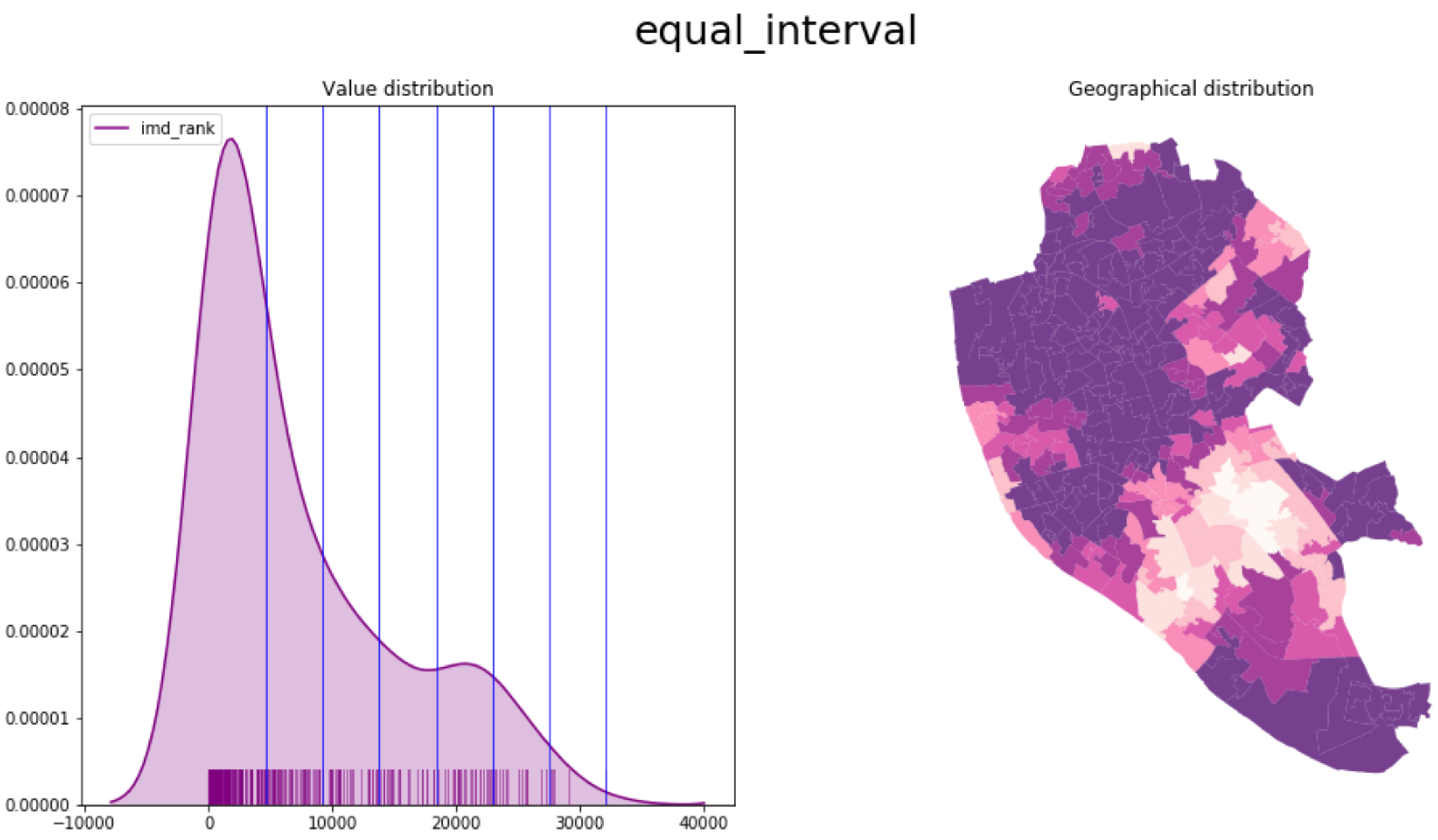
> How do we allocate every value in our `variable` into one of the `k` groups?

Two (common) **approaches** for **continuous** variables:

- Equal Intervals (`'equal_interval'`)
- Quantiles (`'quantiles'`)

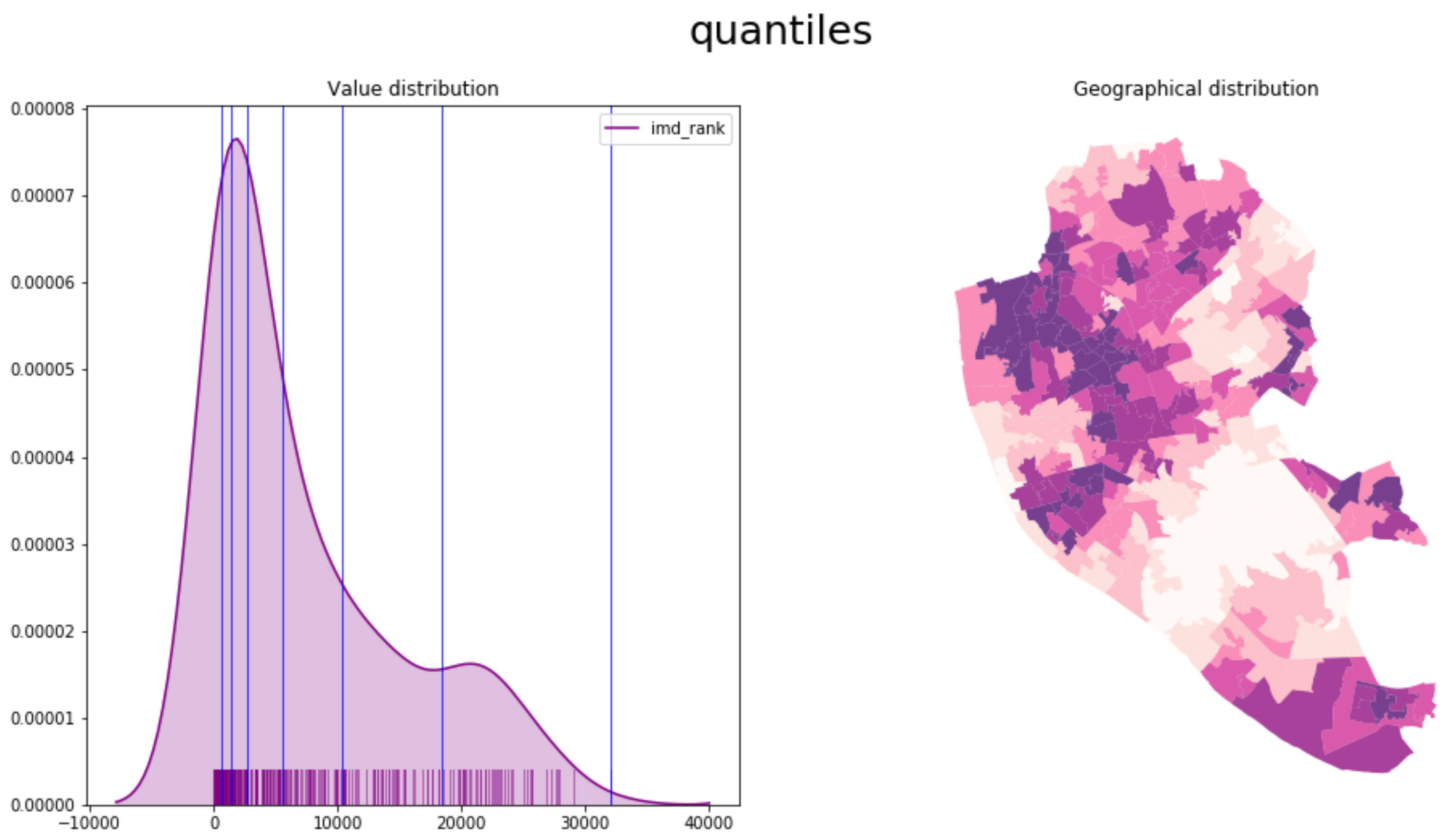
Equal Intervals

```
locations.plot(column='variable', scheme='equal_interval', k=7, cmap='Purples')
```



Quantiles

```python
locations.plot(column='variable', scheme='quantiles', k=7, cmap='Purples')
```



Color

Categories, non-ordered

```
locations.plot(column='variable',
               categorical=True, cmap='Purples')
```


Graduated, **sequential**

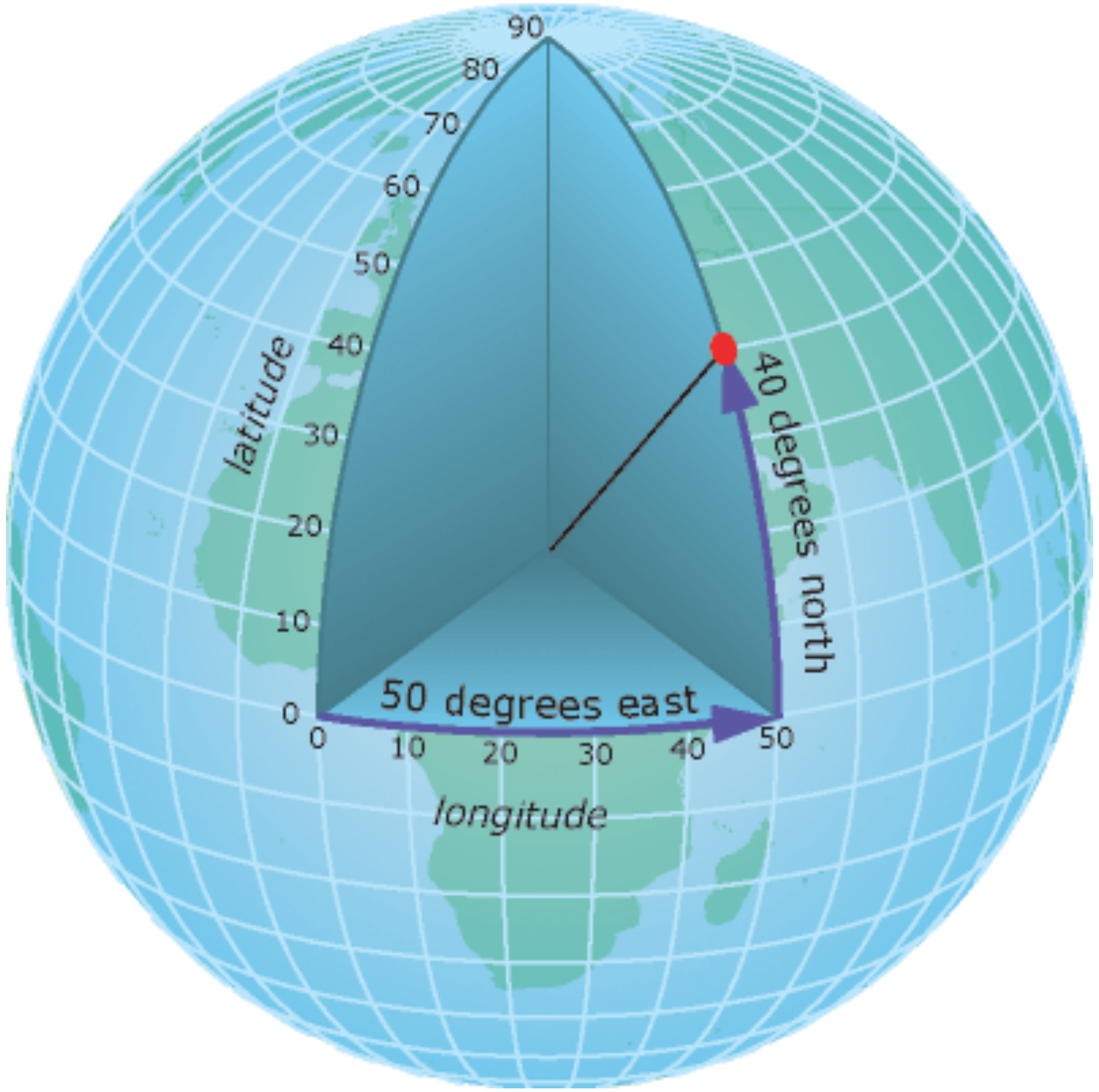
```
locations.plot(column='variable',
               k=5, cmap='RdPu')
```

Graduated, **divergent**

```
locations.plot(column='variable',
               k=5, cmap='RdYlGn')
```

IMPORTANT: Align with your **purpose**

Geographic coordinates



Degrees of latitude and longitude.

E.g. 48°51′N, 2°17′E

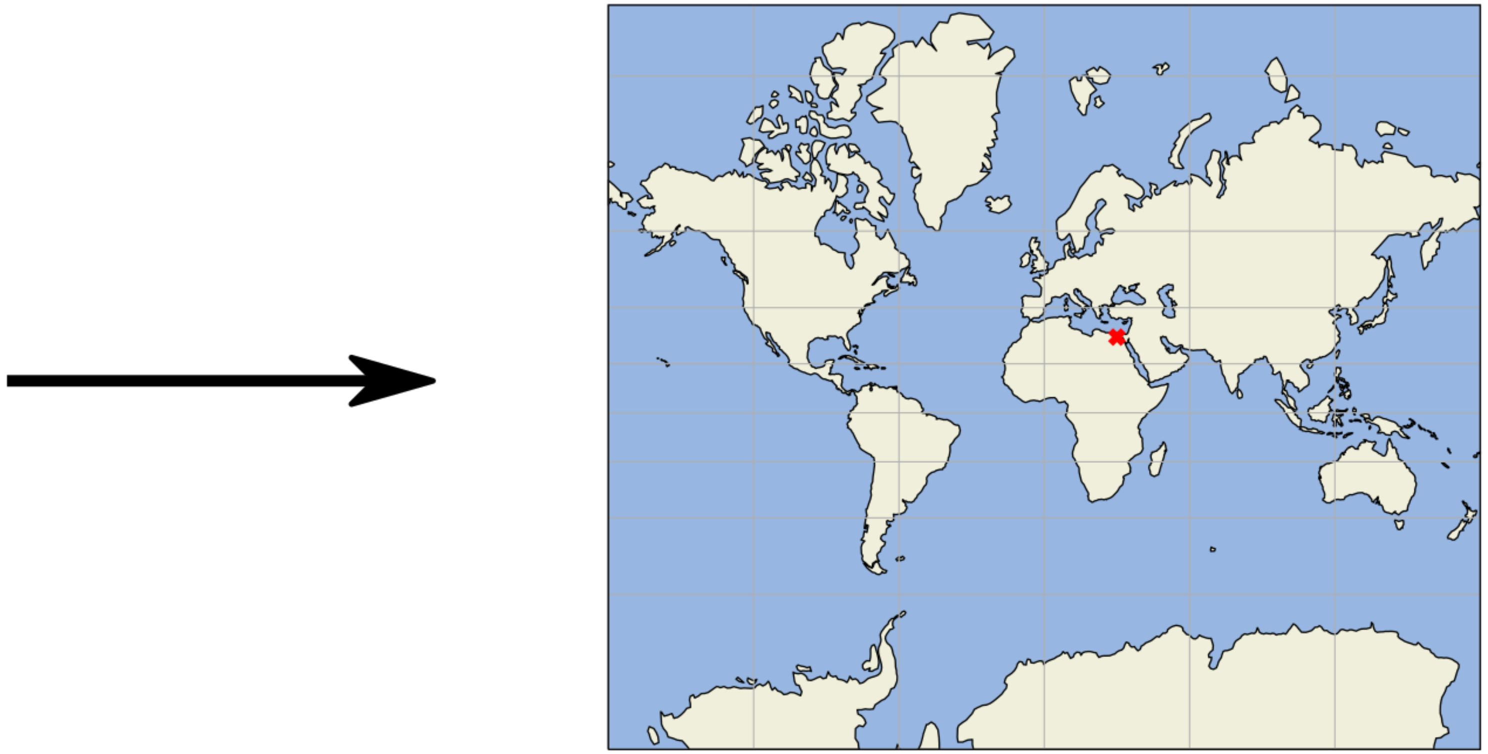
Used in GPS, web mapping applications...

Attention!
in Python we use (lon, lat) and not (lat, long)

- Longitude: [-180, 180]
- Latitude: [-90, 90]

Maps are 2D

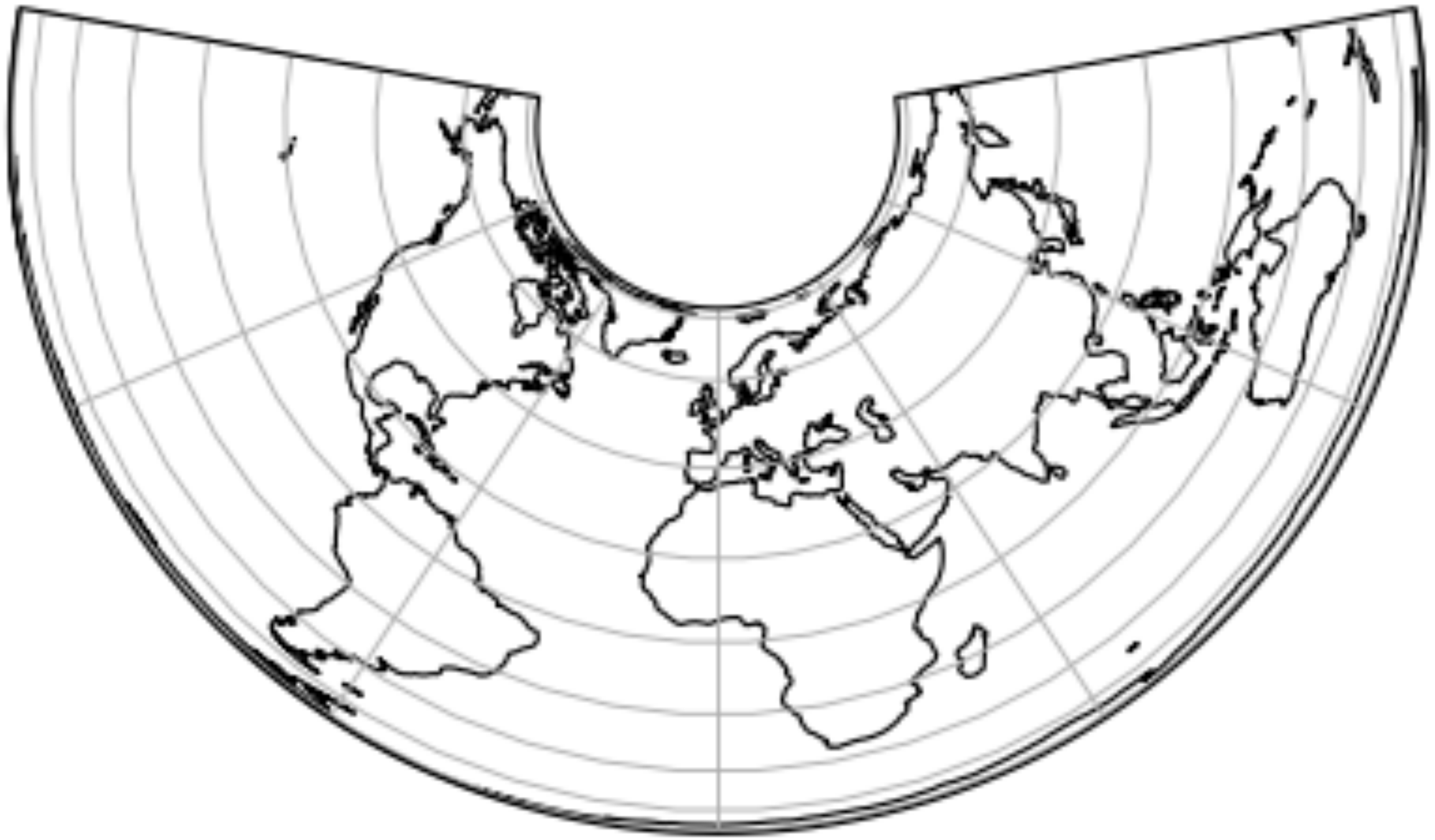
Projected coordinates



`(x, y)` coordinates are usually in meters or feet

Projected coordinates - Examples

Albers Equal Area projection



Projected coordinates - Examples

Mercator projection

Specifying a CRS

`proj4` string

Example: `+proj=longlat +datum=WGS84 +no_defs`

Dict representation:

```
{'proj': 'longlat', 'datum': 'WGS84', 'no_defs': True}
```

EPSG code

Example:

`EPSG:4326` = WGS84 geographic CRS (longitude, latitude)

CRS in GeoPandas

The `.crs` attribute of a GeoDataFrame/GeoSeries:

```python
import geopandas
gdf = geopandas.read_file("countries.shp")
print(gdf.crs)
```

```
{'init': 'epsg:4326'}
```

Setting a CRS manually

```
gdf_noCRS = geopandas.read_file("countries_noCRS.shp")
print(gdf_noCRS.crs)
```

```
{}
```

Add CRS information to `crs` :

```
# Option 1
gdf.crs = {'init': 'epsg:4326'}

# Option 2
gdf.crs = {'proj': 'longlat', 'datum': 'WGS84', 'no_defs': True}
```

Transforming to another CRS

```python
import geopandas
gdf = geopandas.read_file("countries_web_mercator.shp")
print(gdf.crs)
```

```
{'init': 'epsg:3857', 'no_defs': True}
```

The `to_crs()` method:

```python
# Option 1
gdf2 = gdf.to_crs({'proj': 'longlat', 'datum': 'WGS84', 'no_defs': True})
# Option 2
gdf2 = gdf.to_crs(epsg=4326)
```

Why converting the CRS?

1) Sources with a different CRS

```
df1 = geopandas.read_file(...)
df2 = geopandas.read_file(...)


df2 = df2.to_crs(df1.crs)
```

Why converting the CRS?

1) Sources with a different CRS

2) Mapping (distortion of shape and distances)

Why converting the CRS?

1) Sources with a different CRS

2) Mapping (distortion of shape and distances)

3) Distance / area based calculations

How to choose which CRS to use?

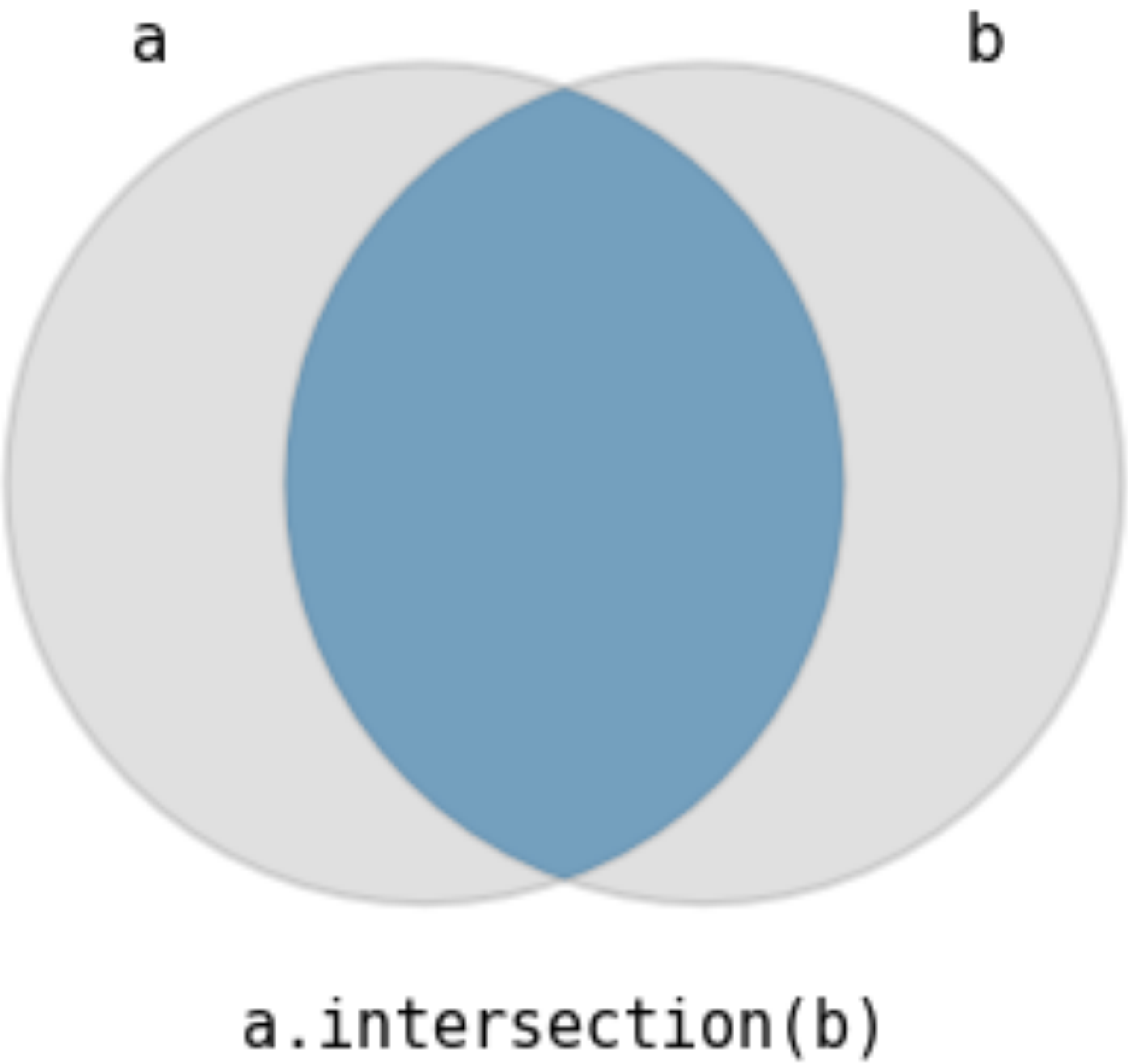
Tips:

- Use projection specific to the area of your data
- Most countries have a standard CRS

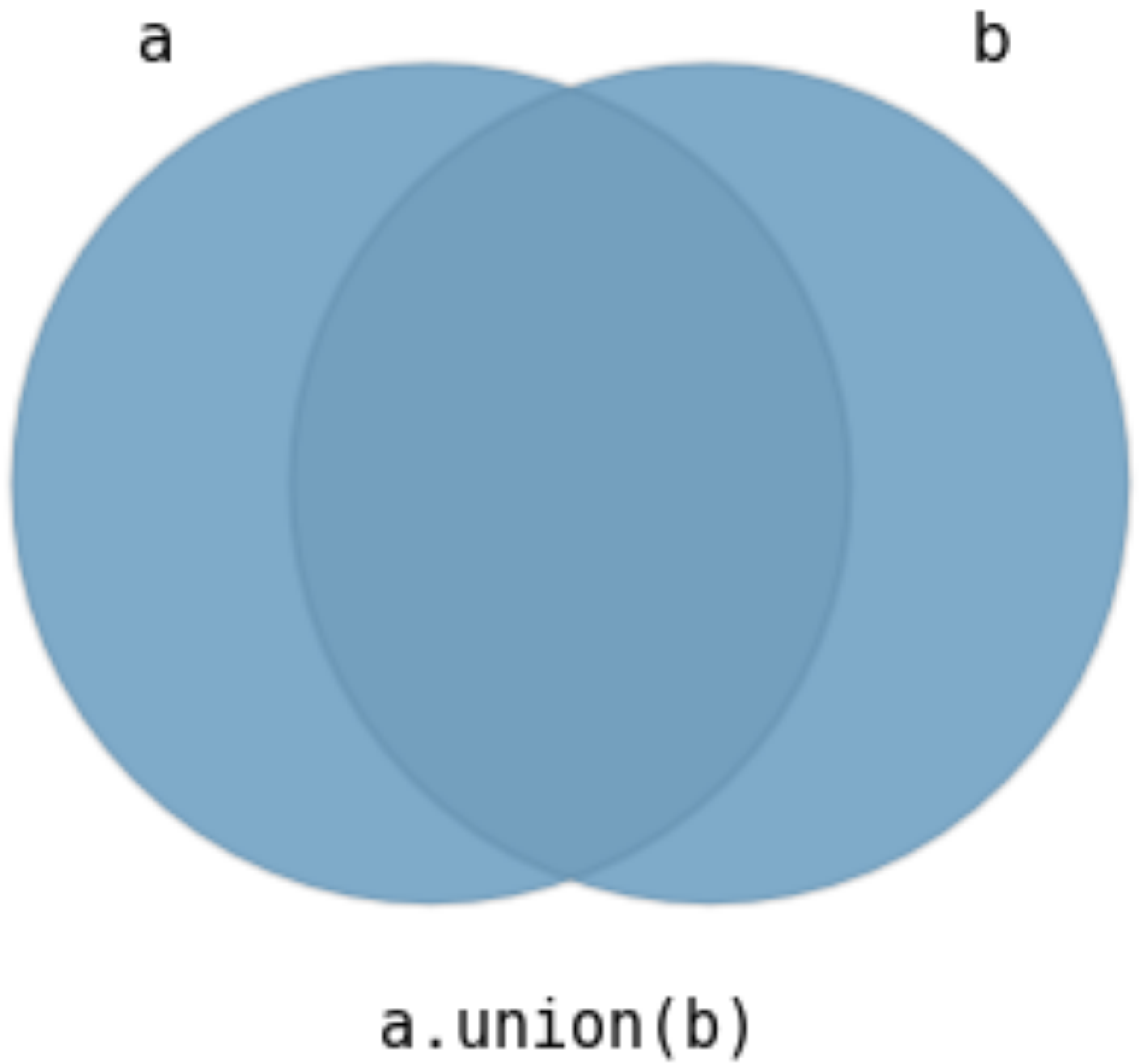
Useful sites:

- http://spatialreference.org/
- https://epsg.io/

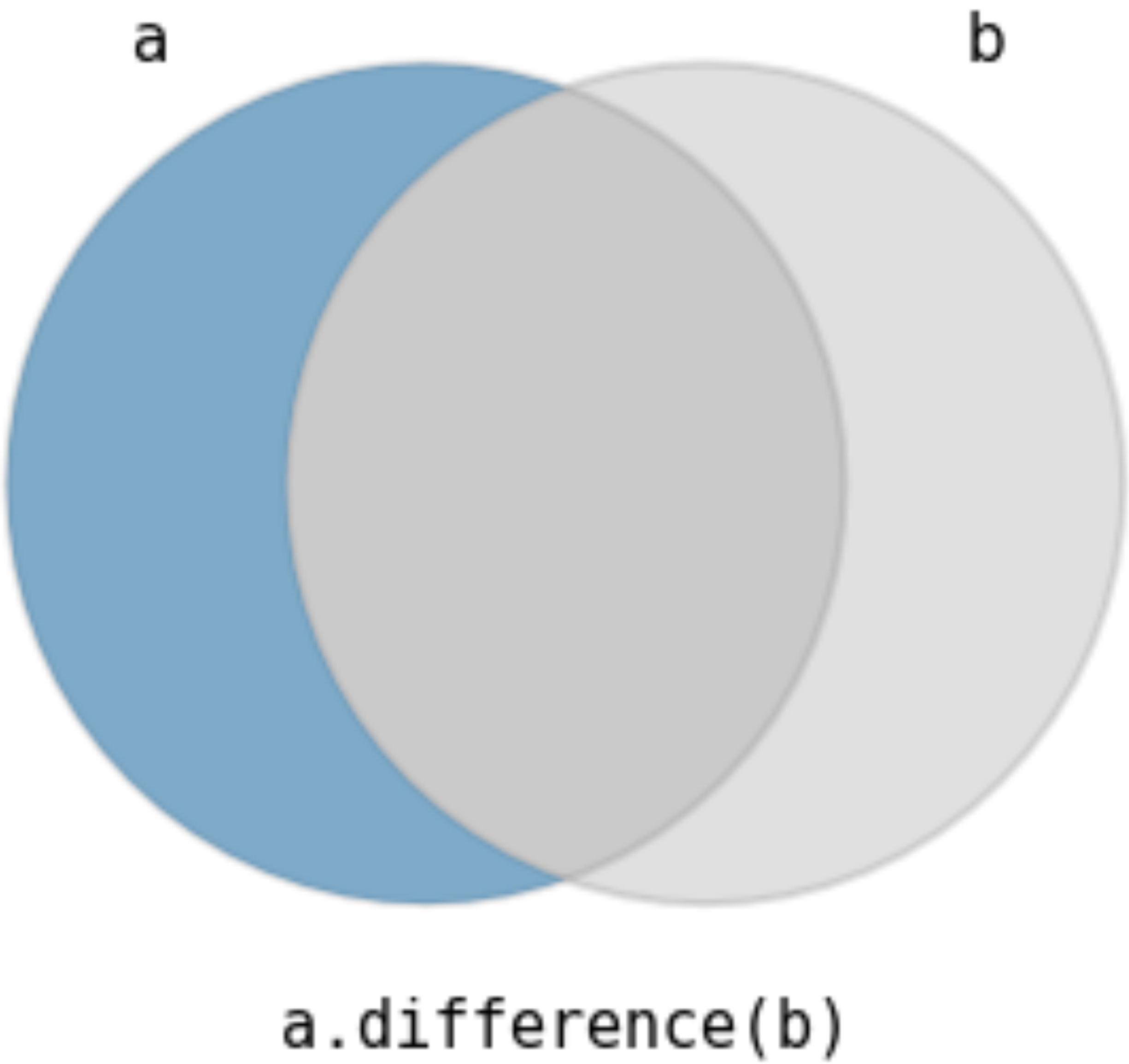
Spatial operations: intersection



Spatial operations: union



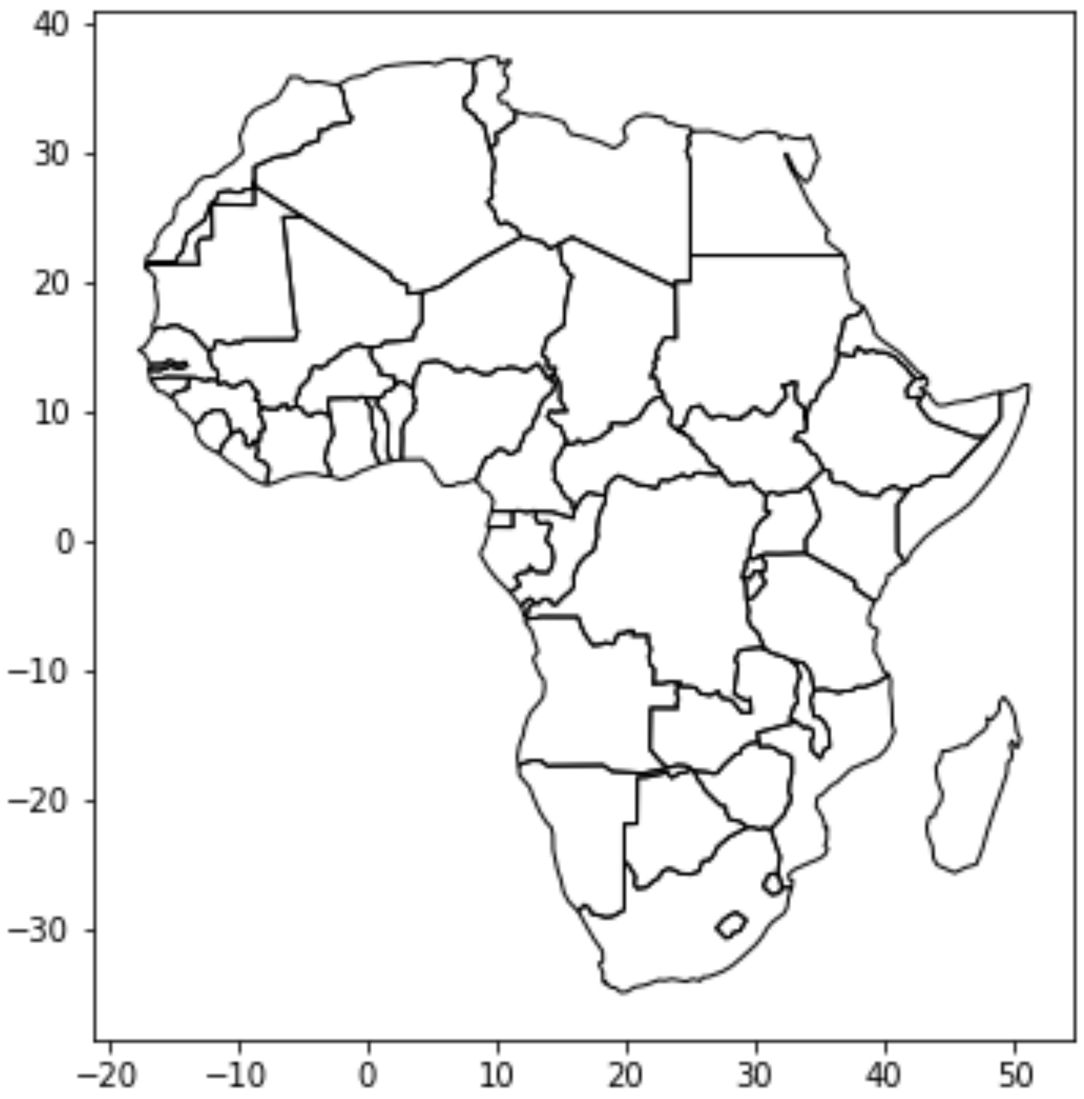
Spatial operations: difference



Spatial operations with GeoPandas

```
africa.head()
```

```
                name                   geometry
0             Angola  (POLYGON ((23.90...
1            Burundi  POLYGON ((29.339...
2              Benin  POLYGON ((2.6917...
3       Burkina Faso  POLYGON ((2.1544...
4           Botswana  POLYGON ((29.432...
```



Spatial operations with GeoPandas

```
print(box)
```

```
POLYGON ((60 10, 60 -10, -20 -10, -20 10)
```