Shapely geometries and spatial relationships

WORKING WITH GEOSPATIAL DATA IN PYTHON

Dani Arribas-Bel
Geographic Data Science Lab
(University of Liverpool)

datacamp

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)
```

Scalar geometry values

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

```
POINT (4.33137074969045 50.83526293533032)
```

```
type(brussels)
```

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

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:

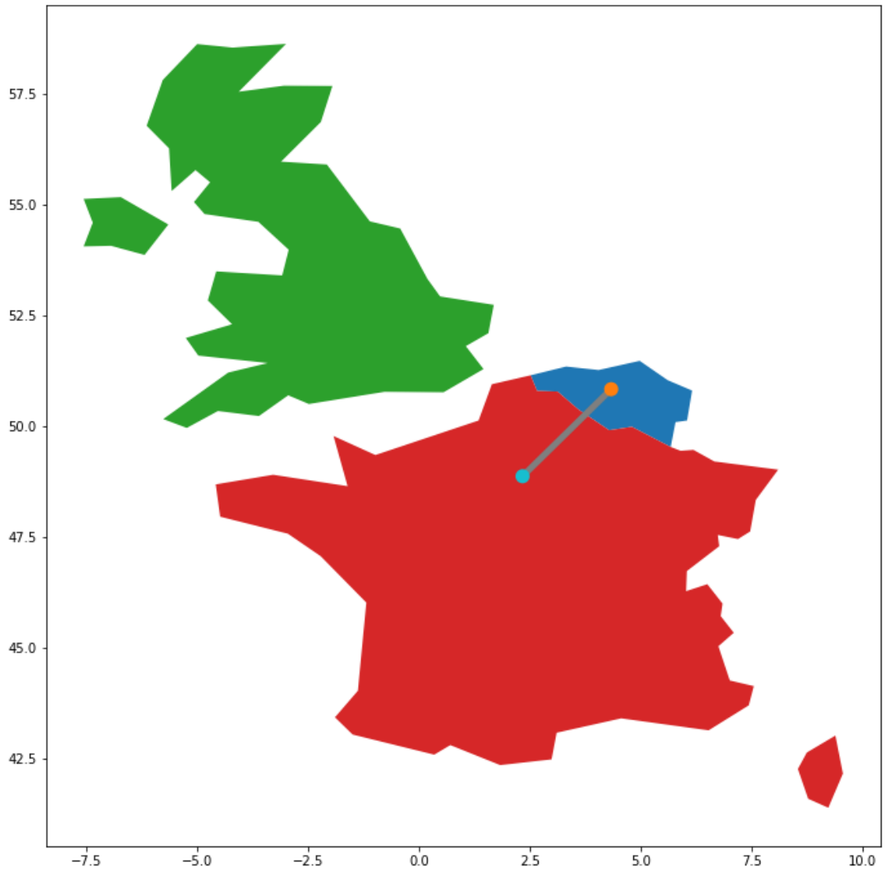
```
brussels.distance(paris)
```

```
2.8049127723186214
```

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

Spatial relationships

```
geopandas.GeoSeries([belgium, france, uk, paris,  brussels, line]).plot()
```



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
```

Let's practice!

WORKING WITH GEOSPATIAL DATA IN PYTHON

Spatial relationships with GeoPandas

WORKING WITH GEOSPATIAL DATA IN PYTHON

Dani Arribas-Bel
Geographic Data Science Lab
(University of Liverpool)

datacamp

Element-wise spatial relationship methods

```
brussels.within(france)
```

```
False
```

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

```
True
```

Element-wise spatial relationship methods

```
brussels.within(france)
```

```
False
```

For full GeoDataFrame?

```
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)
...
```

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
       ...
```

Let's practice!

WORKING WITH GEOSPATIAL DATA IN PYTHON

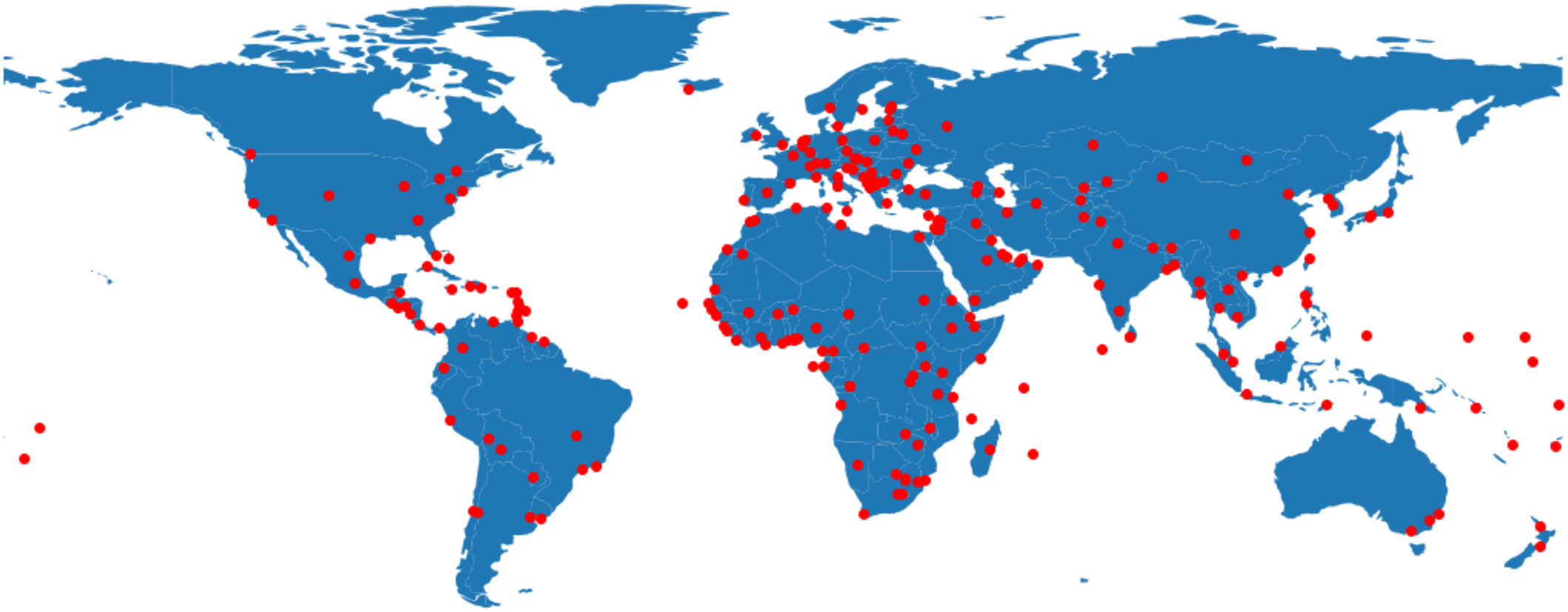
The "spatial join" operation

WORKING WITH GEOSPATIAL DATA IN PYTHON

Dani Arribas-Bel
Geographic Data Science Lab
(University of Liverpool)

datacamp

Spatial relationships I



Spatial relationships II

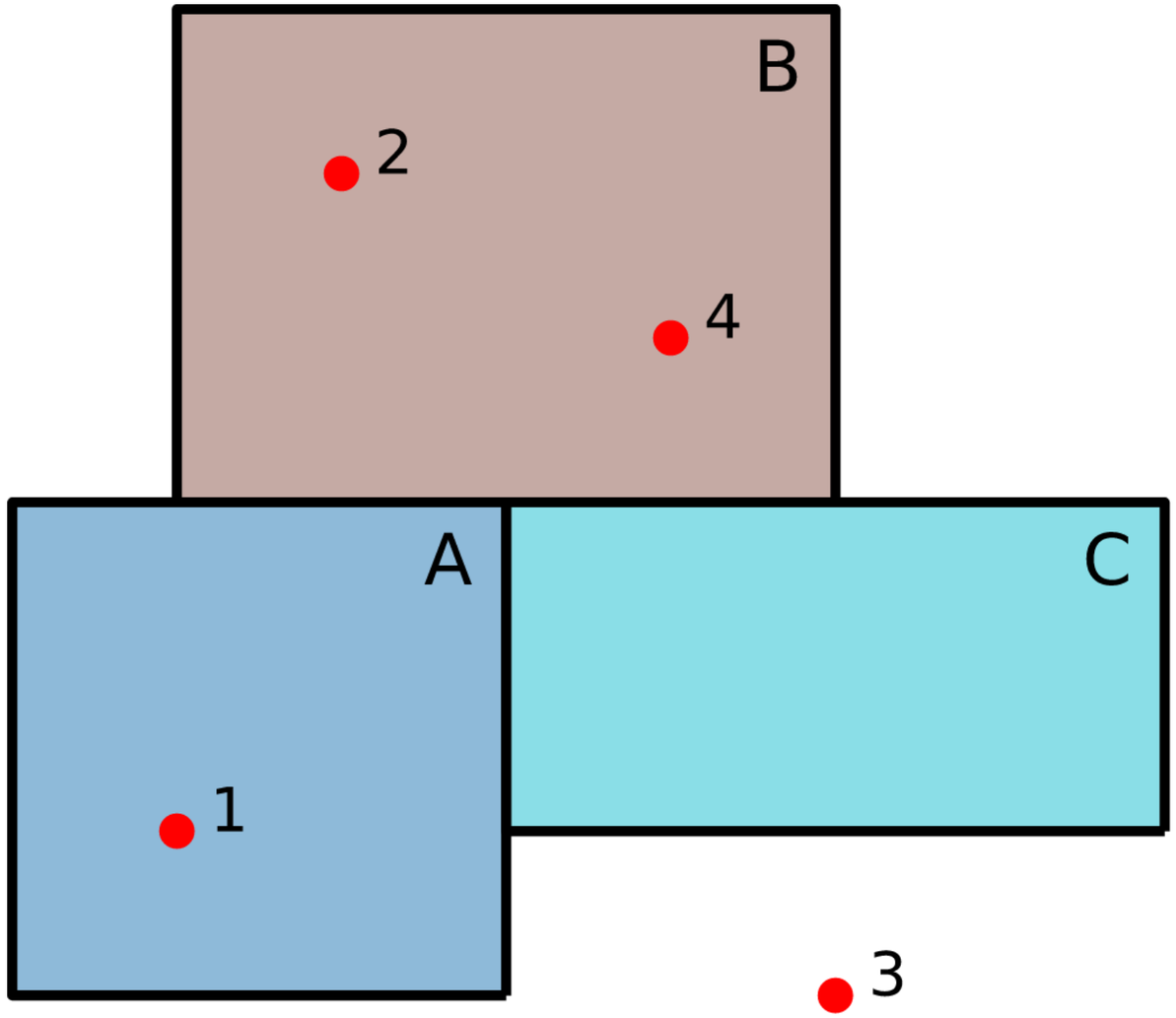
Which cities are located within Brazil?

```
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
```

Let's practice!

WORKING WITH GEOSPATIAL DATA IN PYTHON

Choropleths: Mapping data over space

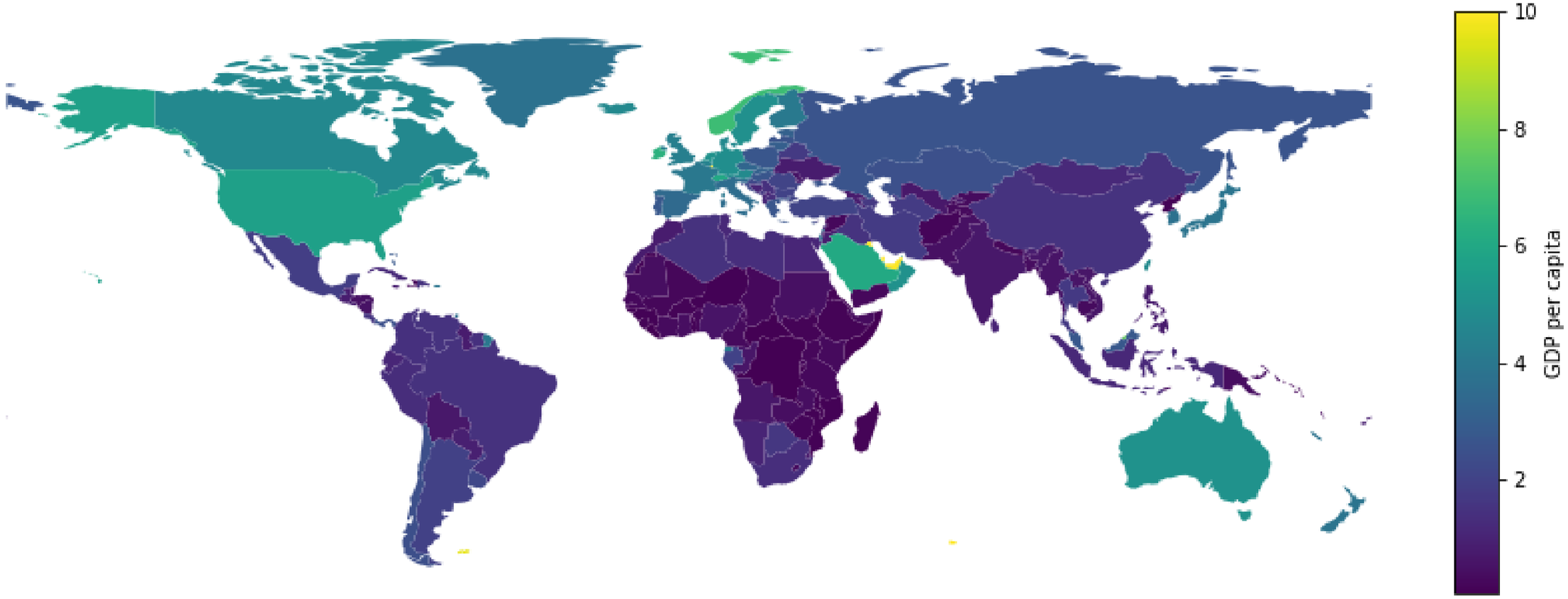
WORKING WITH GEOSPATIAL DATA IN PYTHON

Dani Arribas-Bel
Geographic Data Science Lab
(University of Liverpool)

datacamp

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')
```

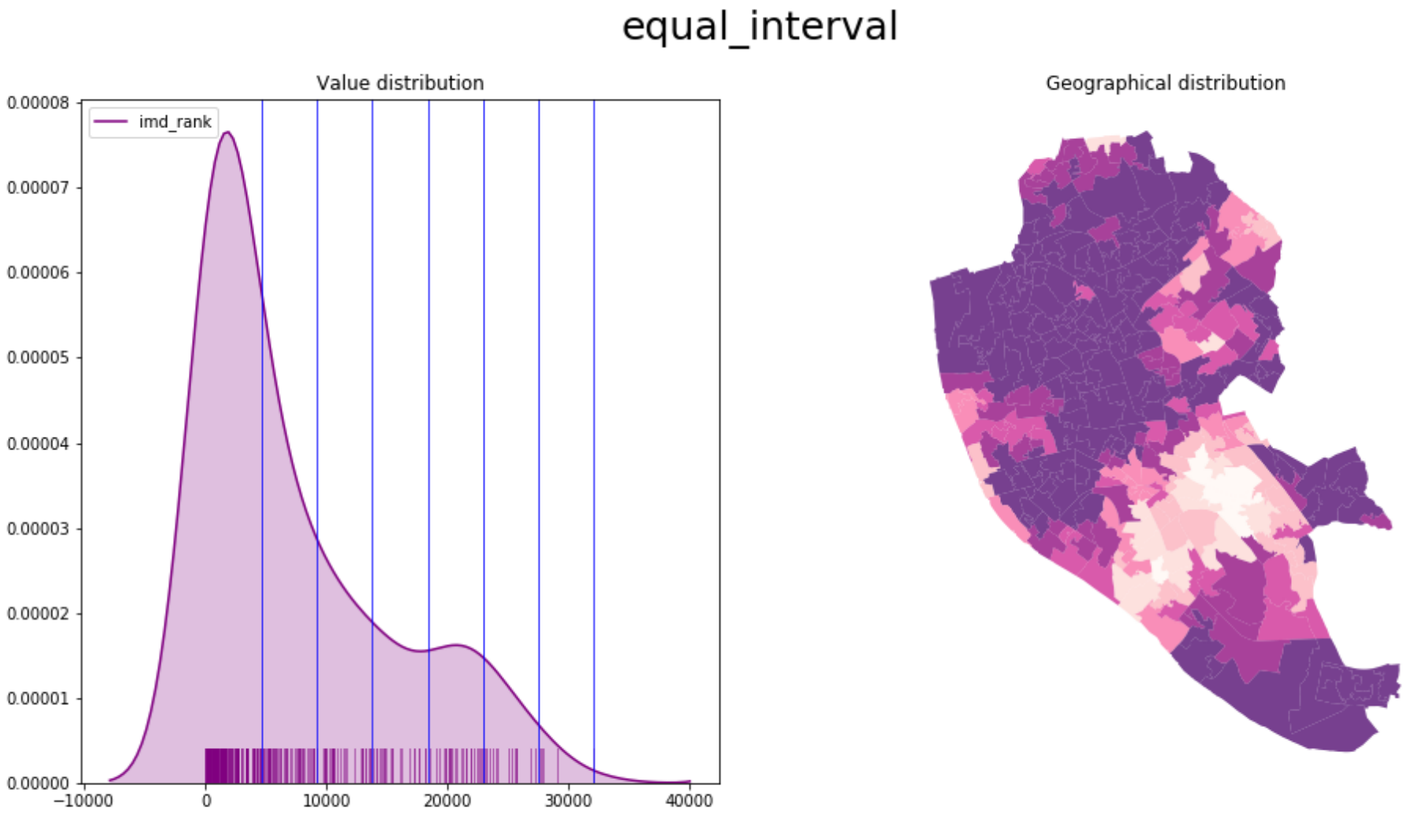
> How do we allocate every value in our `variable` into one of the `k` groups?

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

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

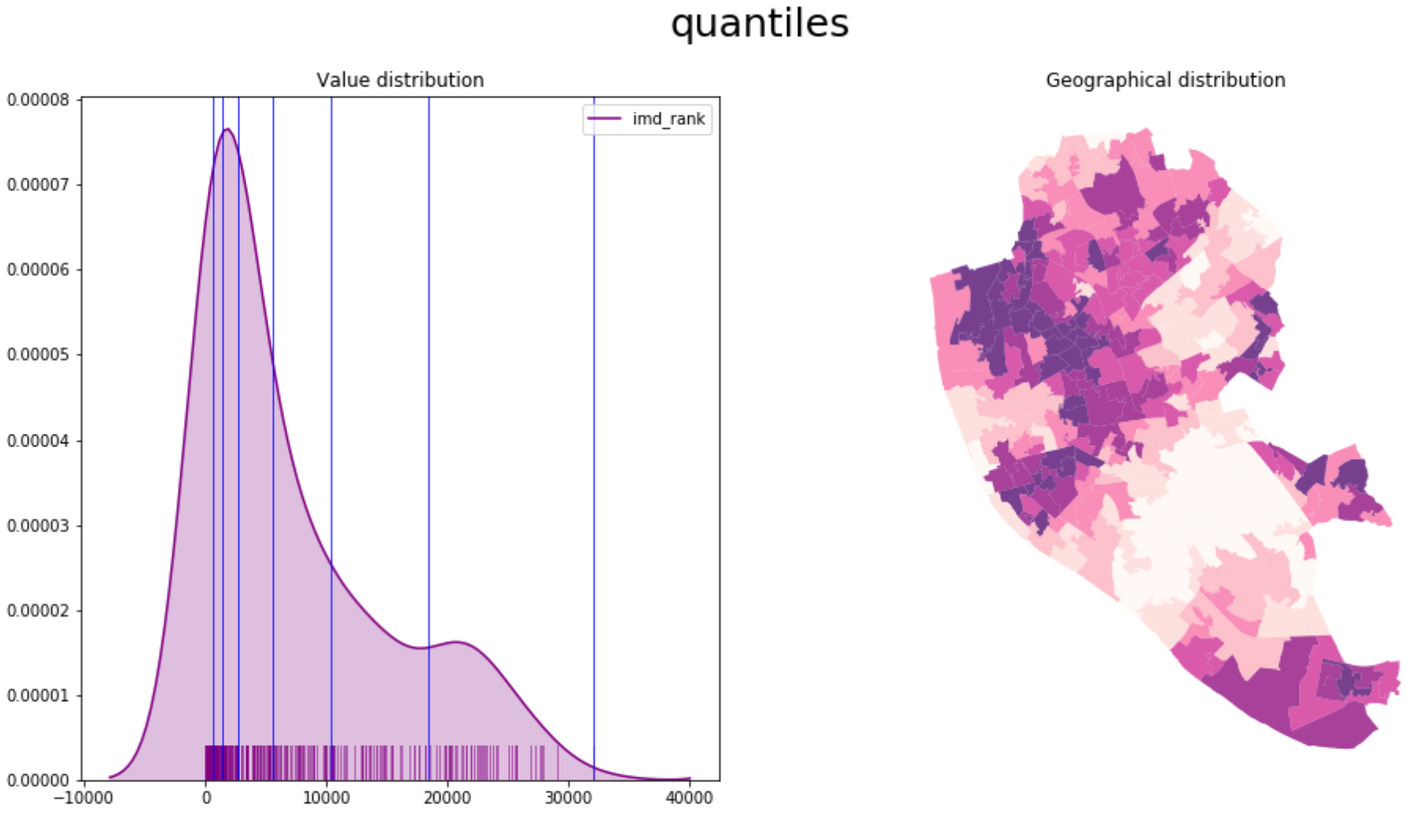
Equal Intervals

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



Quantiles

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



Color

Categories, non-ordered

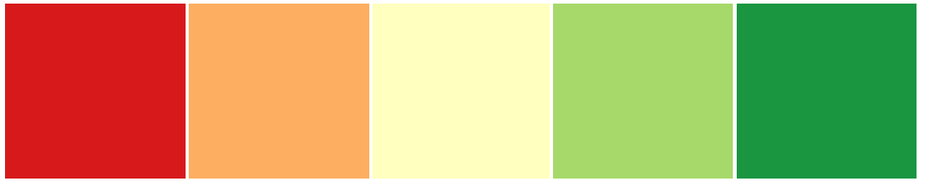
```
locations.plot(column='variable',
               categorical=True, cmap='Purples')
```

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**

Let's practice!

WORKING WITH GEOSPATIAL DATA IN PYTHON