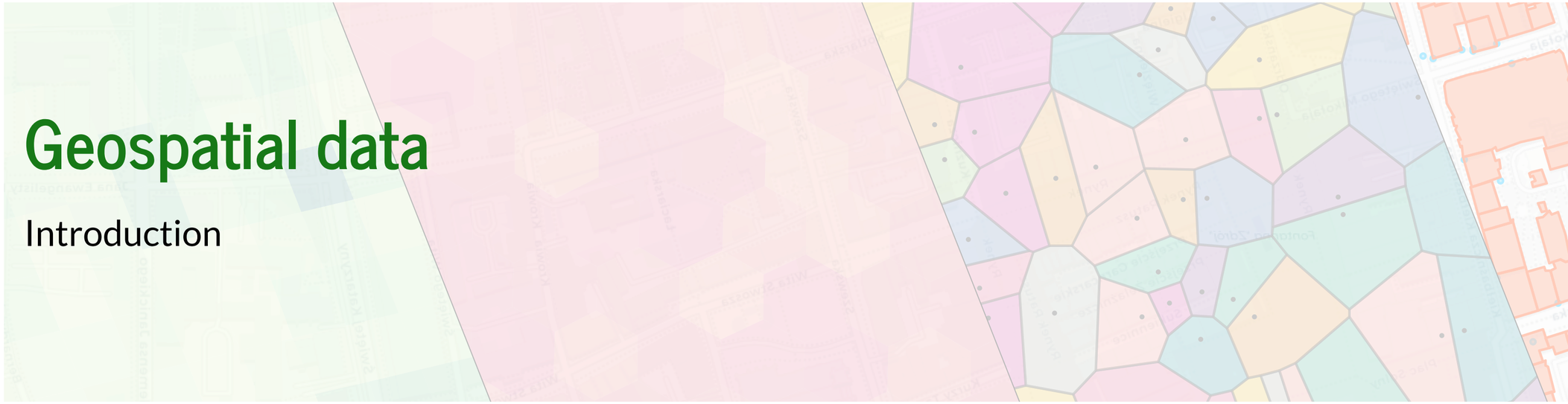
Geospatial data

Introduction



Before we begin

We encorage you to check out the tutorial by Joris Van den Bossche
Introduction to geospatial data analysis with GeoPandas

https://t.ly/agtgJ

What is this part for?

- introduce basic concepts related to geospatial data analysis
- build a common "vocabulary"
- understand geospatial libraries APIs

How is it structured?

- intro to geopandas,
- shapely - geometry library,
- map projections and coordinate reference systems,
- grid systems,
- spatial operations,
- OpenStreetMap.

What is GeoPandas

What is GeoPandas

- open source
- simplifies working with geospatial data
- extends pandas for spatial operations
- geometric operations - shapely
- fiona for file access and matplotlib for plotting

Let's load some data

Let's load some data

GeoPandas implements reading from a number of sources:

- files in formats supported by fiona
- PostGIS databases
- Feather and Parquet files

Let's load some data

GeoPandas implements reading from a number of sources:

- files in formats supported by fiona
- PostGIS databases
- Feather and Parquet files

We'll be use a shapefile zip with countries from Natural Earth

Read the data

Read the data

In [3]:
```
import geopandas as gpd
countries = gpd.read_file("../../data/ne_110m_admin_0_countries.zip")
countries = countries[["ISO_A3", "NAME", "CONTINENT", "POP_EST", "geometry"]]
countries.head(5)
```

Out[3]:

	ISO_A3	NAME	CONTINENT	POP_EST	geometry
0	FJI	Fiji	Oceania	889953.0	MULTIPOLYGON (((180.00000 -16.06713, 180.00000...
1	TZA	Tanzania	Africa	58005463.0	POLYGON ((33.90371 -0.95000, 34.07262 -1.05982...
2	ESH	W. Sahara	Africa	603253.0	POLYGON ((-8.66559 27.65643, -8.66512 27.58948...
3	CAN	Canada	North America	37589262.0	MULTIPOLYGON (((-122.84000 49.00000, -122.9742...
4	USA	United States of America	North America	328239523.0	MULTIPOLYGON (((-122.84000 49.00000, -120.0000...

GeoDataFrames

GeoDataFrames

- an extension of Pandas DataFrames
- consist of:
    - **geometries**: the column where spatial objects are stored
    - **properties**: the rest of the columns, describing the geometries

Let's visualize it

Let's visualize it

We can use:

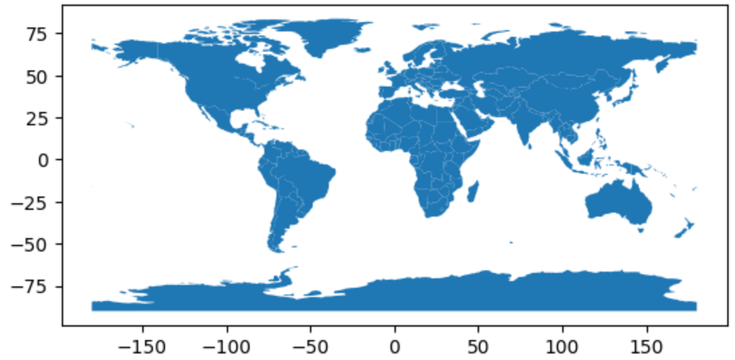
- `.plot()` to plot the geometries on a static map (matplotlib)
- `.explore()` to view them on an interactive map (Folium / Leaflet.js)

.plot()

.plot()

```
In [4]: countries.plot()
```

```
Out[4]: <Axes: >
```

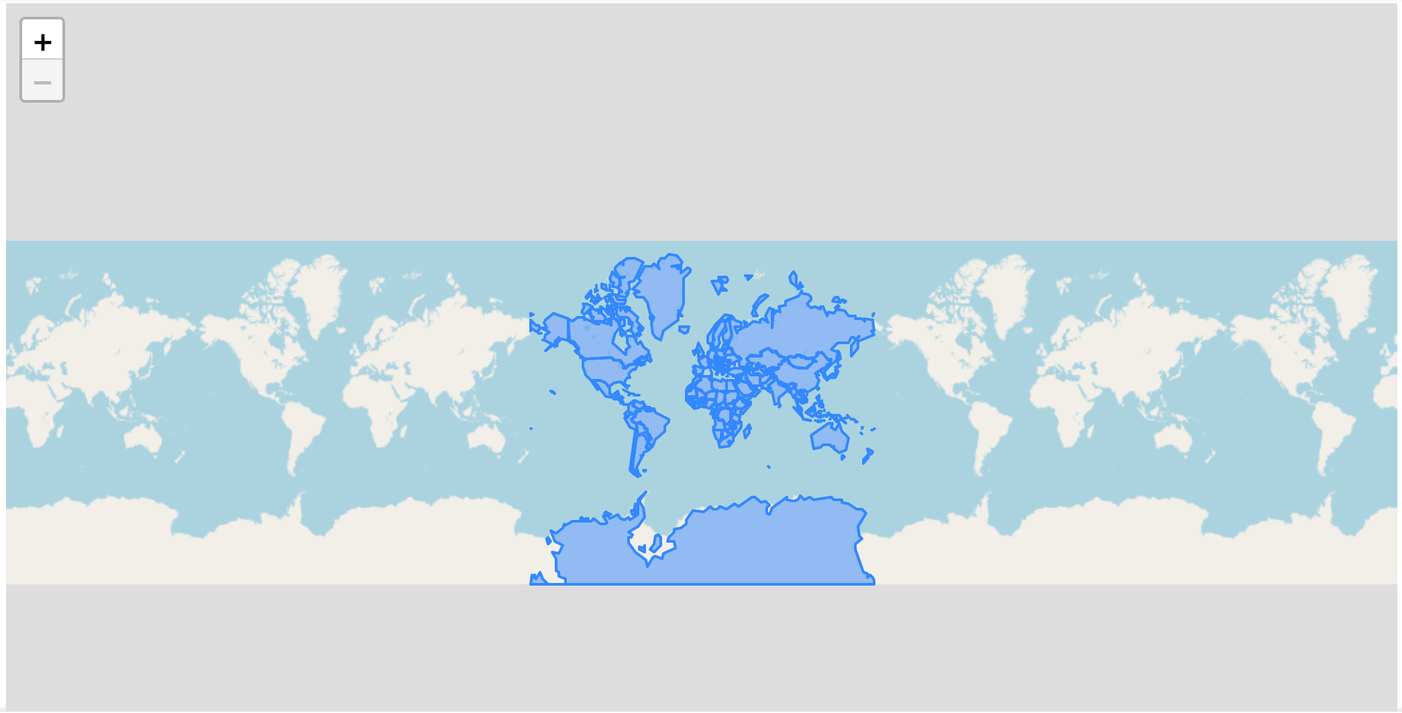


.explore()

.explore()

In [5]:
```
countries.explore()
```

Out[5]:



We are working with a DataFrame

We are working with a DataFrame

```
In [6]: type(countries)

Out[6]: geopandas.geodataframe.GeoDataFrame
```

We are working with a DataFrame

In [6]:
```
type(countries)
```

Out[6]:
```
geopandas.geodataframe.GeoDataFrame
```

In [7]:
```
import pandas as pd
isinstance(countries, pd.DataFrame)
```

Out[7]:
```
True
```

We are working with a DataFrame

In [6]:
```
type(countries)
```

Out[6]:
```
geopandas.geodataframe.GeoDataFrame
```

In [7]:
```
import pandas as pd
isinstance(countries, pd.DataFrame)
```

Out[7]:
```
True
```

In [8]:
```
countries.columns
```

Out[8]:
```
Index(['ISO_A3', 'NAME', 'CONTINENT', 'POP_EST', 'geometry'], dtype='object')
```

Pandas operations

Pandas operations

```
In [9]: countries['POP_EST'].mean()

Out[9]: 43243457.74745763
```

Pandas operations

In [9]:
```
countries['POP_EST'].mean()
```

Out[9]:
```
43243457.74745763
```

In [10]:
```
countries['CONTINENT'].value_counts()
```

Out[10]:
```
CONTINENT
Africa                      51
Asia                        47
Europe                      39
North America               18
South America               13
Oceania                      7
Seven seas (open ocean)      1
Antarctica                   1
Name: count, dtype: int64
```

The geometry column

The geometry column

In [11]:
```
type(countries["POP_EST"]), type(countries.geometry)
```

Out[11]:
```
(pandas.core.series.Series, geopandas.geoseries.GeoSeries)
```

The geometry column

In [11]:
```
type(countries["POP_EST"]), type(countries.geometry)
```

Out[11]:
```
(pandas.core.series.Series, geopandas.geoseries.GeoSeries)
```

In [12]:
```
countries.geometry
```

Out[12]:
```
0      MULTIPOLYGON (((180.00000 -16.06713, 180.00000...
1      POLYGON ((33.90371 -0.95000, 34.07262 -1.05982...
2      POLYGON ((-8.66559 27.65643, -8.66512 27.58948...
3      MULTIPOLYGON (((-122.84000 49.00000, -122.9742...
4      MULTIPOLYGON (((-122.84000 49.00000, -120.0000...
                            ...
172    POLYGON ((18.82982 45.90887, 18.82984 45.90888...
173    POLYGON ((20.07070 42.58863, 19.80161 42.50009...
174    POLYGON ((20.59025 41.85541, 20.52295 42.21787...
175    POLYGON ((-61.68000 10.76000, -61.10500 10.890...
176    POLYGON ((30.83385 3.50917, 29.95350 4.17370, ...
Name: geometry, Length: 177, dtype: geometry
```

Let's go deeper - Shapely objects

- GeoPandas uses Shapely - geometry column
- geometric operations

Let's go deeper - Shapely objects

- GeoPandas uses Shapely - geometry column
- geometric operations

In [13]:
```
pl_gdf = countries[countries["NAME"] == "Poland"]
pl_gdf
```

Out[13]:

	ISO_A3	NAME	CONTINENT	POP_EST	geometry
113	POL	Poland	Europe	37970874.0	POLYGON ((23.48413 53.91250, 23.52754 53.47012...

Let's go deeper - Shapely objects

- GeoPandas uses Shapely - geometry column
- geometric operations

In [13]:
```
pl_gdf = countries[countries["NAME"] == "Poland"]
pl_gdf
```

Out[13]:

	ISO_A3	NAME	CONTINENT	POP_EST	geometry
113	POL	Poland	Europe	37970874.0	POLYGON ((23.48413 53.91250, 23.52754 53.47012...

In [14]:
```
pl_geom = pl_gdf.iloc[0].geometry
pl_geom
```

Out[14]:

Let's go deeper - Shapely objects

- GeoPandas uses Shapely - geometry column
- geometric operations

In [13]:
```
pl_gdf = countries[countries["NAME"] == "Poland"]
pl_gdf
```

Out[13]:

	ISO_A3	NAME	CONTINENT	POP_EST	geometry
113	POL	Poland	Europe	37970874.0	POLYGON ((23.48413 53.91250, 23.52754 53.47012...

In [14]:
```
pl_geom = pl_gdf.iloc[0].geometry
pl_geom
```

Out[14]:

In [15]:
```
type(pl_geom)
```

Out[15]:
```
shapely.geometry.polygon.Polygon
```

Geometry's properties

Geometry's properties

In [16]:
```
## minimum bounding region
pl_geom.bounds
```

Out[16]:
```
(14.074521111719434, 49.02739533140962, 24.029985792748903, 54.85153595643291)
```

Geometry's properties

In [16]:
```
## minimum bounding region
pl_geom.bounds
```

Out[16]:
```
(14.074521111719434, 49.02739533140962, 24.029985792748903, 54.85153595643291)
```

In [17]:
```
pl_geom.area
```

Out[17]:
```
40.759230708989925
```

Geometry's properties

In [16]:
```
## minimum bounding region
pl_geom.bounds
```

Out[16]:
```
(14.074521111719434, 49.02739533140962, 24.029985792748903, 54.85153595643291)
```

In [17]:
```
pl_geom.area
```

Out[17]:
```
40.759230708989925
```

Note: Shapely (and GeoPandas) assumes 2D cartesian plane, so this is only valid when using a proper coordinate reference system.

Creating a geometry manually

Creating a geometry manually

In [18]:
```
from shapely.geometry import LineString
bounds = pl_geom.bounds
line = LineString(
    [(bounds[0], bounds[1]),
    (bounds[2], bounds[3]),]
)
line
```

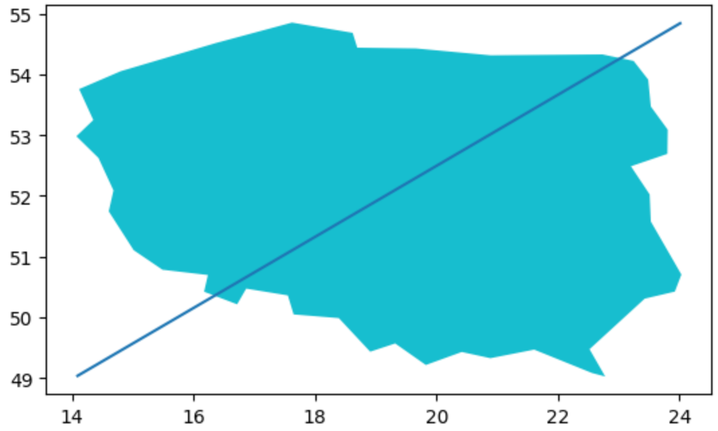
Out[18]:

View it

View it

In [19]:
```
gpd.GeoSeries([line, pl_geom]).plot(cmap='tab10')
```

Out[19]: <Axes: >



Coordinate Reference Systems

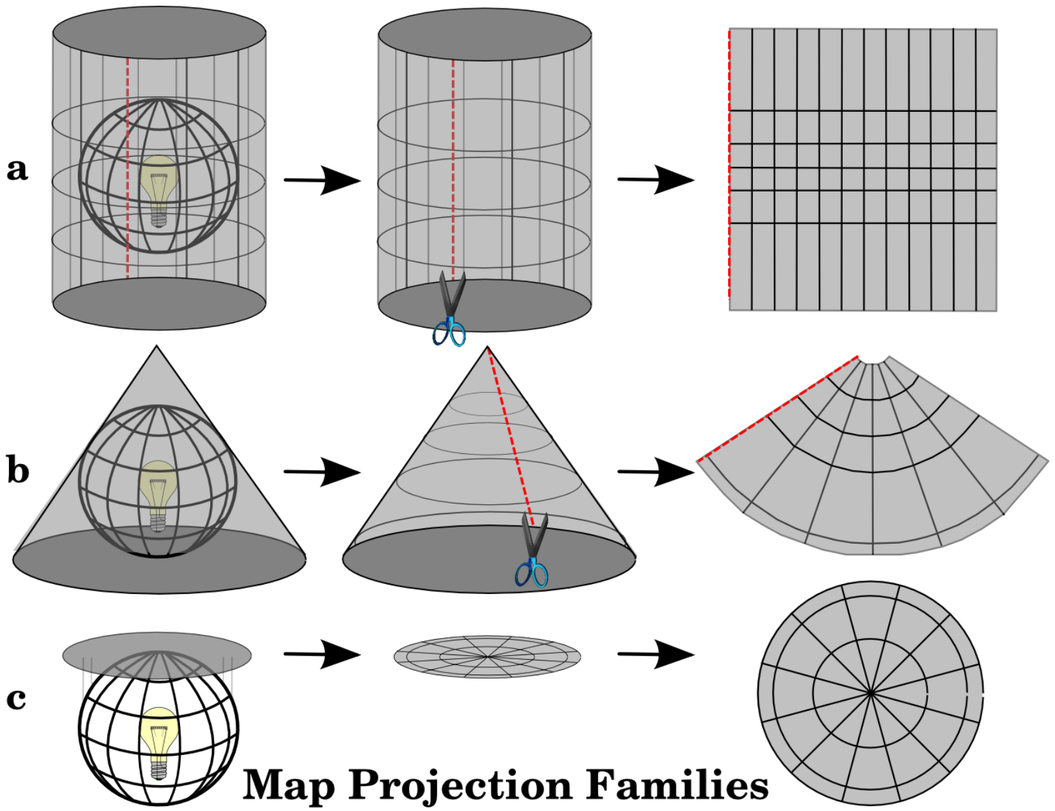
Coordinate Reference Systems

A coordinate reference system (CRS) then defines how the two-dimensional, projected map in your GIS relates to real places on the earth. For a detailed description, see e.g.
https://docs.qgis.org/3.28/en/docs/gentle_gis_introduction/coordinate_reference

The topic **map projection** is very complex and even professionals who have studied geography, geodetics or any other GIS related science, often have problems with the correct definition of map projections and coordinate reference systems. Usually when you work with GIS, you already have projected data to start with. In most cases these data will be projected in a certain CRS, so you don't have to create a new CRS or even re project the data from one CRS to another. That said, it is always useful to have an idea about what map projection and CRS means.

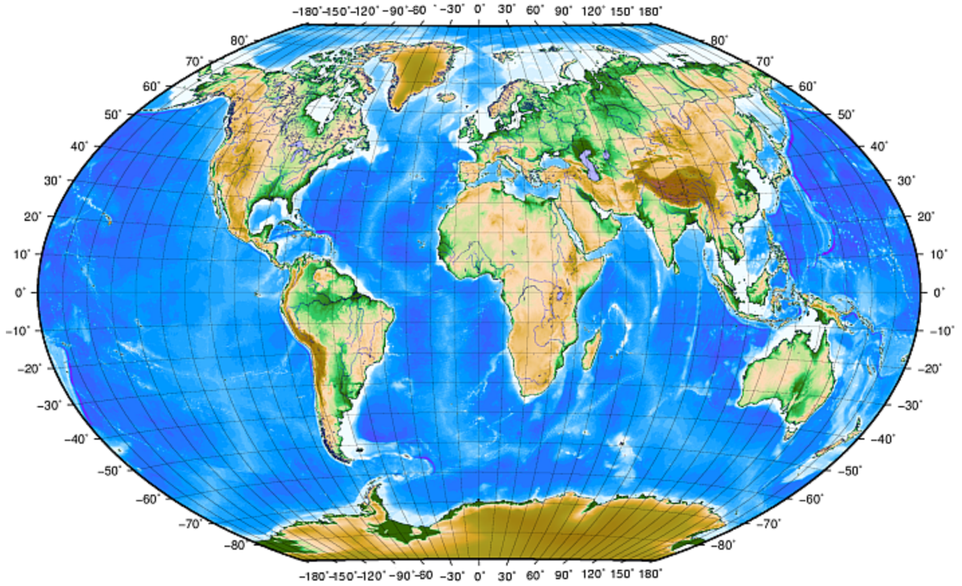
Map projection families

Map projection families



Geographic Coordinate Systems

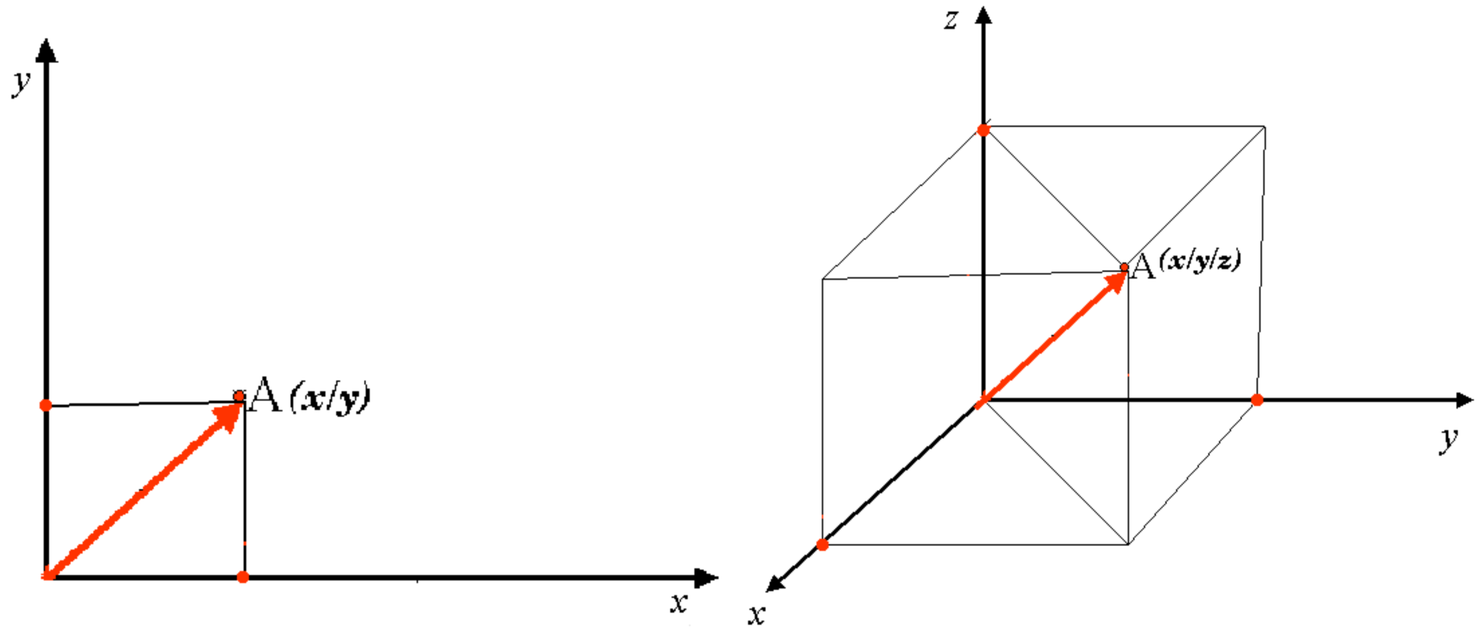
The use of Geographic Coordinate Reference Systems is very common. They use degrees of latitude and longitude and sometimes also a height value to describe a location on the earth's surface. The most popular is called WGS 84.



Note: Throughout the tutorial and the SRAI library, we default to the WGS 84 coordinate system.

Projected coordinate reference systems

Another type of CRS is a projected coordinate reference system. In this type of CRS (x,y) values often represent meters or feet, which makes it easier to work with e.g. when calculating distances or areas.



CRS in Python

CRS in Python

Check crs in GeoPandas

CRS in Python

Check crs in GeoPandas

In [20]:
```
countries.crs
```

Out[20]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

Let's fix Shapely plotting

Let's fix Shapely plotting

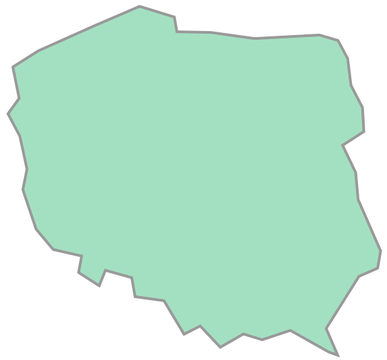
The plotting didn't work in Shapely before because it interpreted degrees as cartesian coordinates. Let's fix that.

Let's fix Shapely plotting

The plotting didn't work in Shapely before because it interpreted degrees as cartesian coordinates. Let's fix that.

In [21]:
```
pl_geom_reprojected = pl_gdf.to_crs(2180).geometry.iloc[0]
pl_geom_reprojected
```

Out[21]:



Calculate area of Poland properly

Calculate area of Poland properly

The boundaries from the shapefile are not perfect but we should be able to get a rough estimate of the area using a proper CRS.

Calculate area of Poland properly

The boundaries from the shapefile are not perfect but we should be able to get a rough estimate of the area using a proper CRS.

In [22]:
```
area_km2 = pl_geom_reprojected.area / 10**6
print(f"Rough estimate of Poland's area: {area_km2:.2f} km2")
```

```
Rough estimate of Poland's area: 310191.98 km2
```

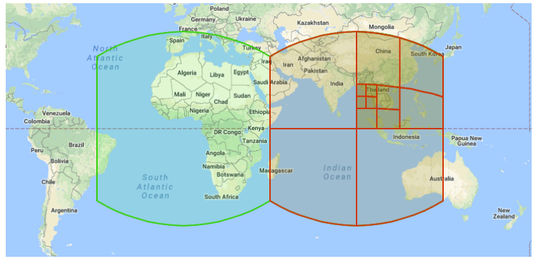
What happens if we use .area in WGS 84?

What happens if we use .area in WGS 84?

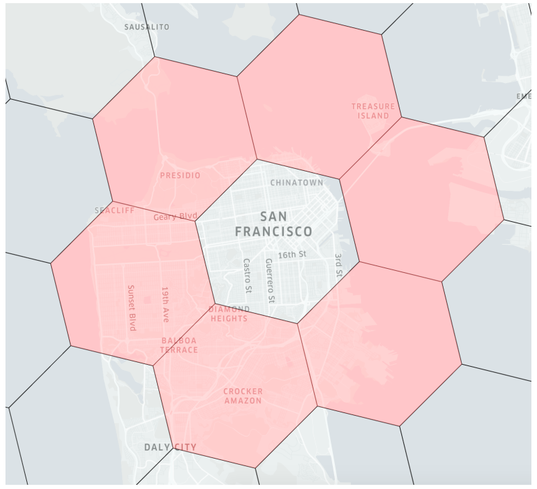
```
In [23]: pl_gdf.area

Out[23]: 113    40.759231
         dtype: float64
```

Grid systems

- A tool to divide and index space
- Examples include H3, S2, Geohash



S2



H3

Geohash

H3 - Hexagonal hierarchical geospatial indexing system

- hexagonal grid
- can be (approximately) subdivided into finer and finer hexagonal grids

Spatial operations

Spatial operations

In [ ]:
```
gpd.GeoSeries([line, pl_geom]).plot(cmap='tab10')
```

Out[24]:
```
<Axes: >
```

Spatial operations

In [ ]:
```
gpd.GeoSeries([line, pl_geom]).plot(cmap='tab10')
```

Out[24]: <Axes: >

In [ ]:
```
line.within(pl_geom)
```

Spatial operations

In [ ]:
```
gpd.GeoSeries([line, pl_geom]).plot(cmap='tab10')
```

Out[24]:
```
<Axes: >
```

In [ ]:
```
line.within(pl_geom)
```

In [ ]:
```
line.intersects(pl_geom)
```

Spatial operations on GeoDataFrames

You can use the same spatial operations as in Shapely, on entire GeoDataFrames.

Spatial operations on GeoDataFrames

You can use the same spatial operations as in Shapely, on entire GeoDataFrames.

In [ ]:
```
pl_de_gdf = countries[countries["NAME"].isin(["Poland", "Germany"])]
pl_de_gdf
```

Spatial operations on GeoDataFrames

You can use the same spatial operations as in Shapely, on entire GeoDataFrames.

In [ ]:
```
pl_de_gdf = countries[countries["NAME"].isin(["Poland", "Germany"])]
pl_de_gdf
```

In [ ]:
```
merged_geom = pl_de_gdf.unary_union
merged_geom
```

Spatial operations on GeoDataFrames

You can use the same spatial operations as in Shapely, on entire GeoDataFrames.

In [ ]:
```
pl_de_gdf = countries[countries["NAME"].isin(["Poland", "Germany"])]
pl_de_gdf
```

In [ ]:
```
merged_geom = pl_de_gdf.unary_union
merged_geom
```

In [ ]:
```
gpd.GeoSeries([line, merged_geom]).plot(cmap='tab10')
```

In [ ]:

```
from srai.regionalizers import geocode_to_region_gdf, H3Regionalizer

regionized = H3Regionalizer(resolution=3).transform(pl_de_gdf)
regionized["intersects"] = regionized.intersects(line)
regionized.explore("intersects")
```

Spatial joins

Spatial joins

In [ ]:
```python
from srai.regionalizers import geocode_to_region_gdf, H3Regionalizer

prague_gdf = geocode_to_region_gdf("Prague, Czech Republic")
regionized = H3Regionalizer(resolution=7).transform(prague_gdf)
regionized.explore()
```

Get bicycle data for Prague

Get bicycle data for Prague

In [ ]:

```python
from srai.loaders import OSMOnlineLoader

loader = OSMOnlineLoader()
prague_bikes = loader.load(prague_gdf, {"amenity": "bicycle_rental"})
prague_bikes.explore(tiles="CartoDB Positron")
```

Perform the join

Perform the join

In [ ]:
```
joint_gdf = regionized.sjoin(prague_bikes)
joint_gdf
```

Count bike stations

Count bike stations

In [ ]:
```
regionized.sjoin(prague_bikes).groupby("region_id").size()
```

OpenStreetMap

OpenStreetMap

OpenStreetMap is a free open data source of map data.
It is a collaborative project to create a free editable map of the world.
It is built using vector data with optional tags to describe the features.
The main page of the project is https://www.openstreetmap.org/
You can find example map features here:
https://wiki.openstreetmap.org/wiki/Map_Features and here:
https://taginfo.openstreetmap.org/

Elements

Elements are the basic components of OpenStreetMap's conceptual data model of the physical world. Elements are of three types:

- nodes (defining points in space),
- ways (defining linear features and area boundaries), and
- relations (which are sometimes used to explain how other elements work together). All of the above can have one or more associated tags (key:value pairs) which describe the meaning of a particular element.

To sum up

To sum up

- GeoPandas
    - Pandas spatial extension
    - very useful tool for working with geospatial data
    - used by SRAI internally
- Shapely
    - used by GeoPandas
    - implements geometries and spatial operations
- Coordinate reference systems
    - how the two-dimensional, projected map relates to real places on the earth
    - "basic" but non-trivial
- Grid systems
- Spatial operations
    - both on Shapely objects and GeoDataFrames
    - relationships such as `within`, `intersects`