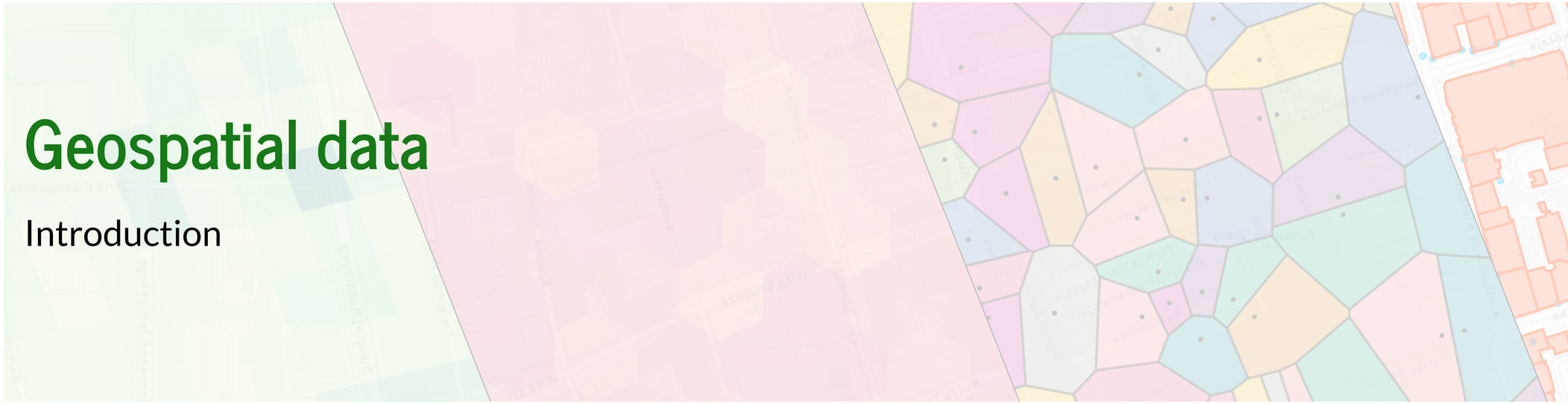
Geospatial data

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 a couple of basic concepts
- build a "vocabulary"
- understand SRAI APIs
- mostly used by SRAI under the hood
- pre/post processing, data preparation, analysis and visualization

SRAI utilizes GeoPandas

SRAI utilizes GeoPandas

- one of its main under-the-hood libraries
- SRAI builds on-top of GeoPandas
- most functions either accept as input, return or otherwise work with GeoDataFrames
- easy to use existing GeoPandas functionalities
    - pre-processing, post-processing, data-preparation, visualization etc.

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

What are GeoDataFrames

What are 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 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 shapefile

Read the shapefile

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...

Visualize the geometries

Visualize the geometries

We can use:

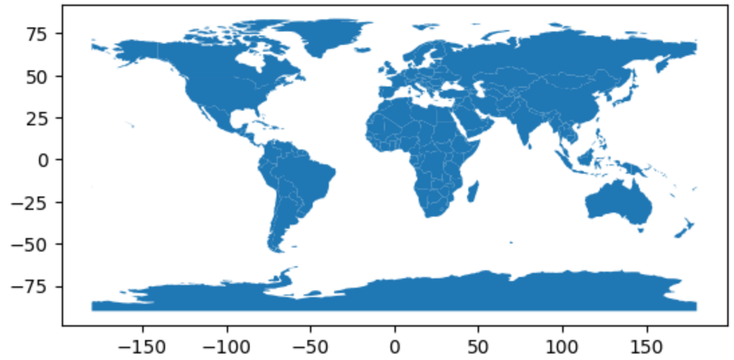
- `.plot()` to plot the geometries on a static map (matplotlib)
- `.explore()` to view them on an interactive map (Folium / Leaflet.js)

.plot()

.plot()

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

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

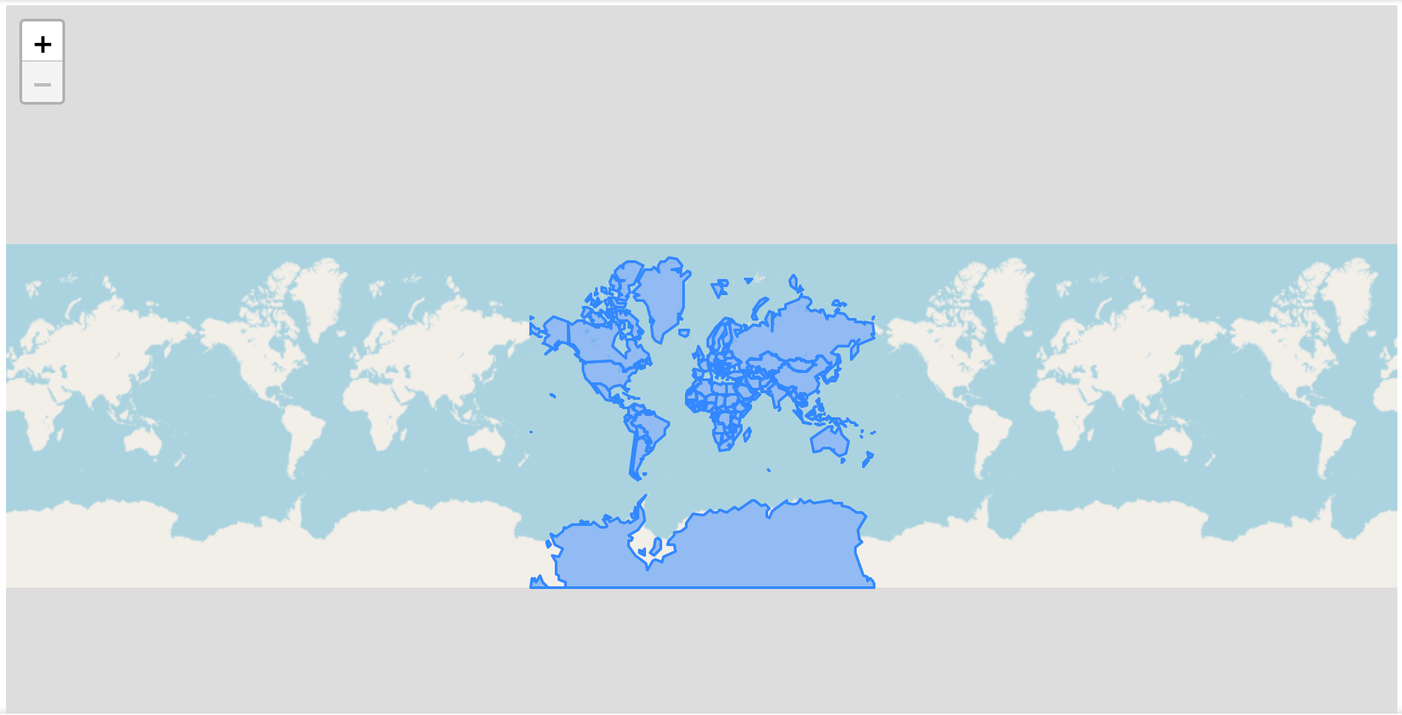


.explore()

.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
```

Calculating the area

Calculating the area

In [13]:
```
countries.geometry.area
```

Out[13]:
```
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
```

Using GeoDataFrames in SRAI

In [14]:
```
poland_gdf = countries[countries["NAME"] == "Poland"]
poland_gdf
```

Out[14]:

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

In [15]:
```
from srai.regionalizers import AdministrativeBoundaryRegionalizer

regionalizer = AdministrativeBoundaryRegionalizer(admin_level=4)
regions_gdf = regionalizer.transform(poland_gdf)
regions_gdf.head(5)
```

```
Loading boundaries: 40it [00:58,  1.46s/it]
```

Out[15]:

region_id	geometry
Region of Pre8ov	MULTIPOLYGON (((20.32066 49.39959, 20.32162 49...
Region of Žilina	MULTIPOLYGON (((18.83549 49.51554, 18.83875 49...
Lesser Poland Voivodeship	POLYGON ((19.95365 50.50579, 19.95498 50.50660...
Subcarpathian Voivodeship	POLYGON ((21.21125 50.35504, 21.21493 50.35558...
Silesian Voivodeship	POLYGON ((18.03507 50.06624, 18.03578 50.06673...

In [16]:
```
regions_gdf.explore()
```

Out[16]:

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 [17]: type(regions_gdf.iloc[2].geometry)
```

```
Out[17]: shapely.geometry.polygon.Polygon
```

Let's go deeper - Shapely objects

- GeoPandas uses Shapely - geometry column
- geometric operations

In [17]:
```
type(regions_gdf.iloc[2].geometry)
```

Out[17]:
```
shapely.geometry.polygon.Polygon
```

In [18]:
```
print(regions_gdf.iloc[2].name)
voivodeship_region = regions_gdf.iloc[2:3]
voivodeship_geom = voivodeship_region.geometry[0]
voivodeship_geom
```

```
Lesser Poland Voivodeship
```

Out[18]:

Geometry's properties

Geometry's properties

In [19]:
```
voivodeship_geom.area
```

Out[19]:
```
1.8662664985355808
```

Geometry's properties

In [19]:
```
voivodeship_geom.area
```

Out[19]:
```
1.8662664985355808
```

In [20]:
```
## minimum bounding region
voivodeship_geom.bounds
```

Out[20]:
```
(19.0831232, 49.21712535256923, 21.4217327, 50.5204879)
```

Let's create a geometry object

Let's create a geometry object

In [21]:
```
from shapely.geometry import LineString
bounds = voivodeship_geom.bounds
line = LineString(
    [(bounds[0], bounds[1]),
    (bounds[2], bounds[3]),]
)
line
```

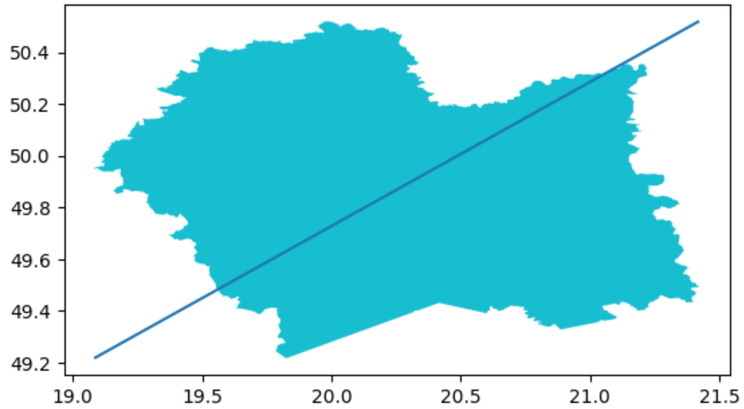
Out[21]:

View it

View it

```
In [22]: gpd.GeoSeries([line, voivodeship_geom]).plot(cmap='tab10')
```

```
Out[22]: <Axes: >
```



Spatial operations

Spatial operations

In [23]:
```
line.within(voivodeship_geom)
```

Out[23]:
```
False
```

Spatial operations

In [23]:
```
line.within(voivodeship_geom)
```

Out[23]:
```
False
```

In [24]:
```
line.intersects(voivodeship_geom)
```

Out[24]:
```
True
```

Spatial operations on GeoDataFrames

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

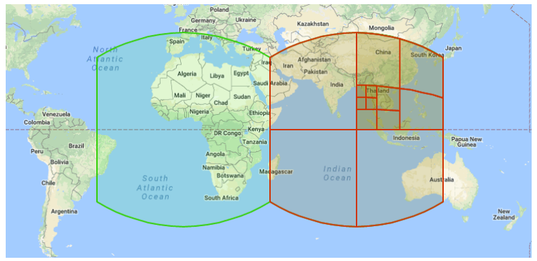
Spatial operations on GeoDataFrames

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

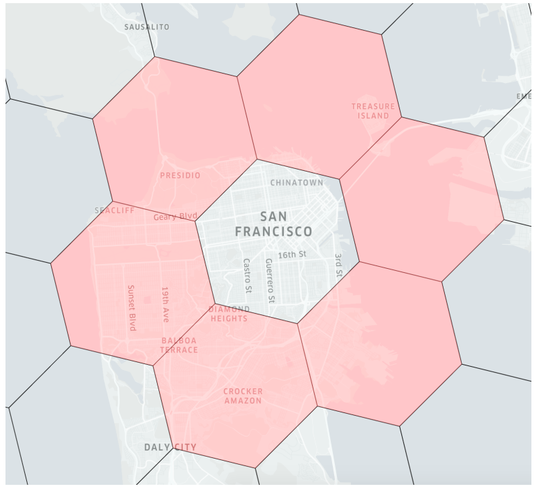
Let's prepare some data to show it.

One last concept - Spatial indexes

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



S2



H3

Geohash

Visit our research group at `kraina.ai`

H3 - Hexagonal hierarchical geospatial indexing system

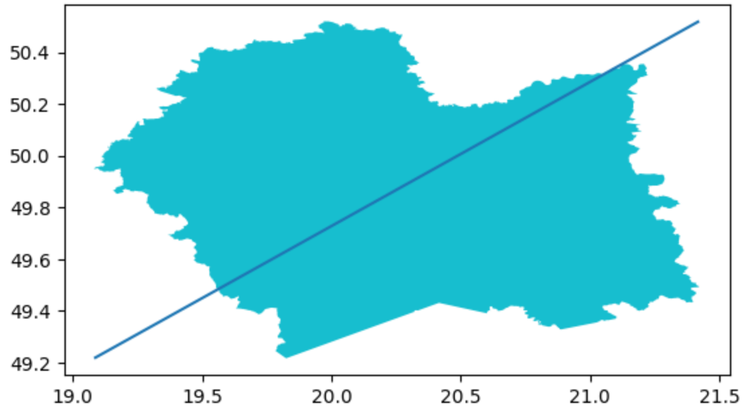
- hexagonal grid
- can be (approximately) subdivided into finer and finer hexagonal grids

Revisitting the previous example

Revisitting the previous example

```
In [25]: gpd.GeoSeries([line, voivodeship_geom]).plot(cmap='tab10')
```

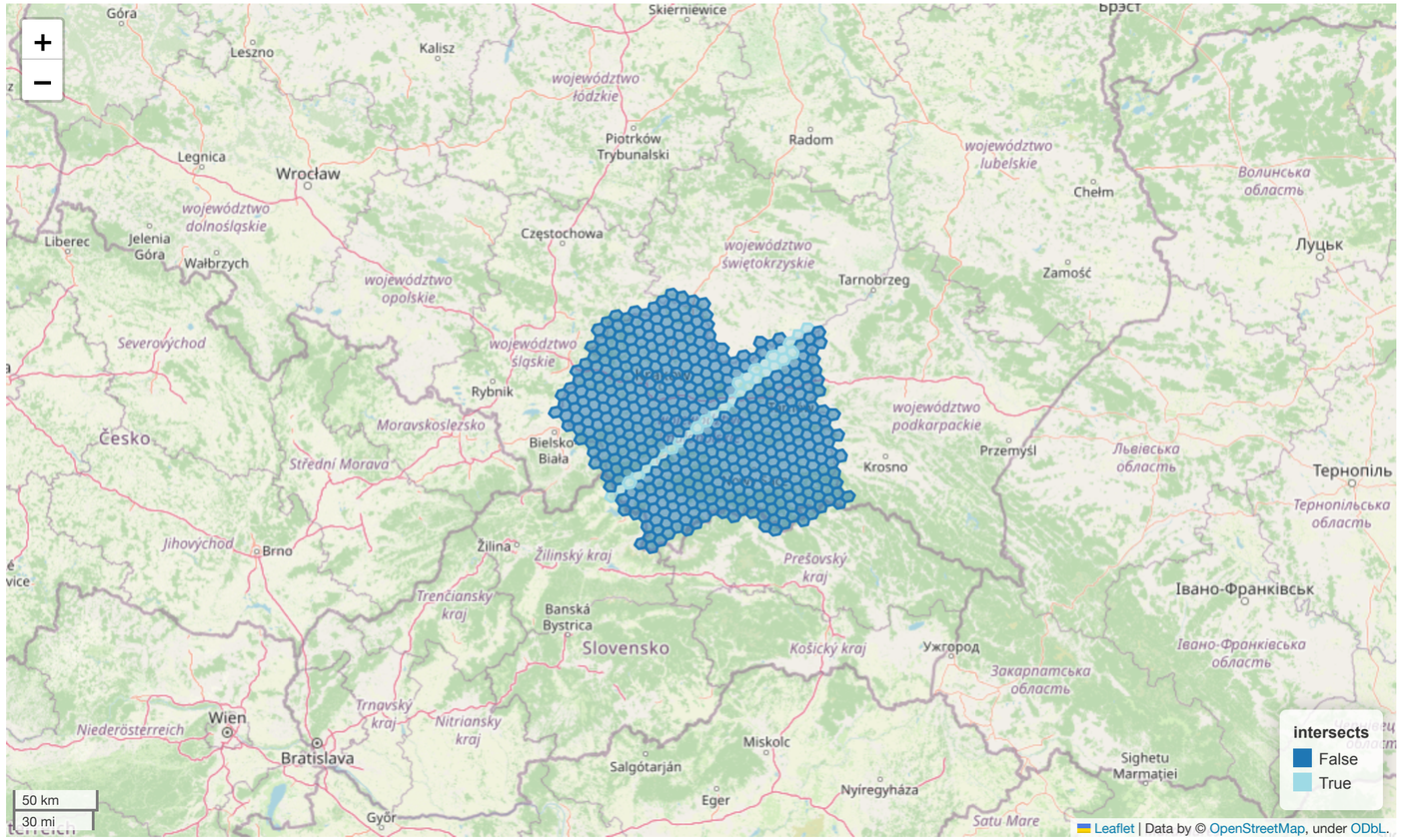
```
Out[25]: <Axes: >
```



In [26]:
```
from srai.regionalizers import geocode_to_region_gdf, H3Regionalizer
from utils import CB_SAFE_PALLETE

regionized = H3Regionalizer(resolution=6).transform(voivodeship_region)
regionized["intersects"] = regionized.intersects(line)
regionized.explore("intersects")
```

Out[26]:



Visit our research group at `kraina.ai`

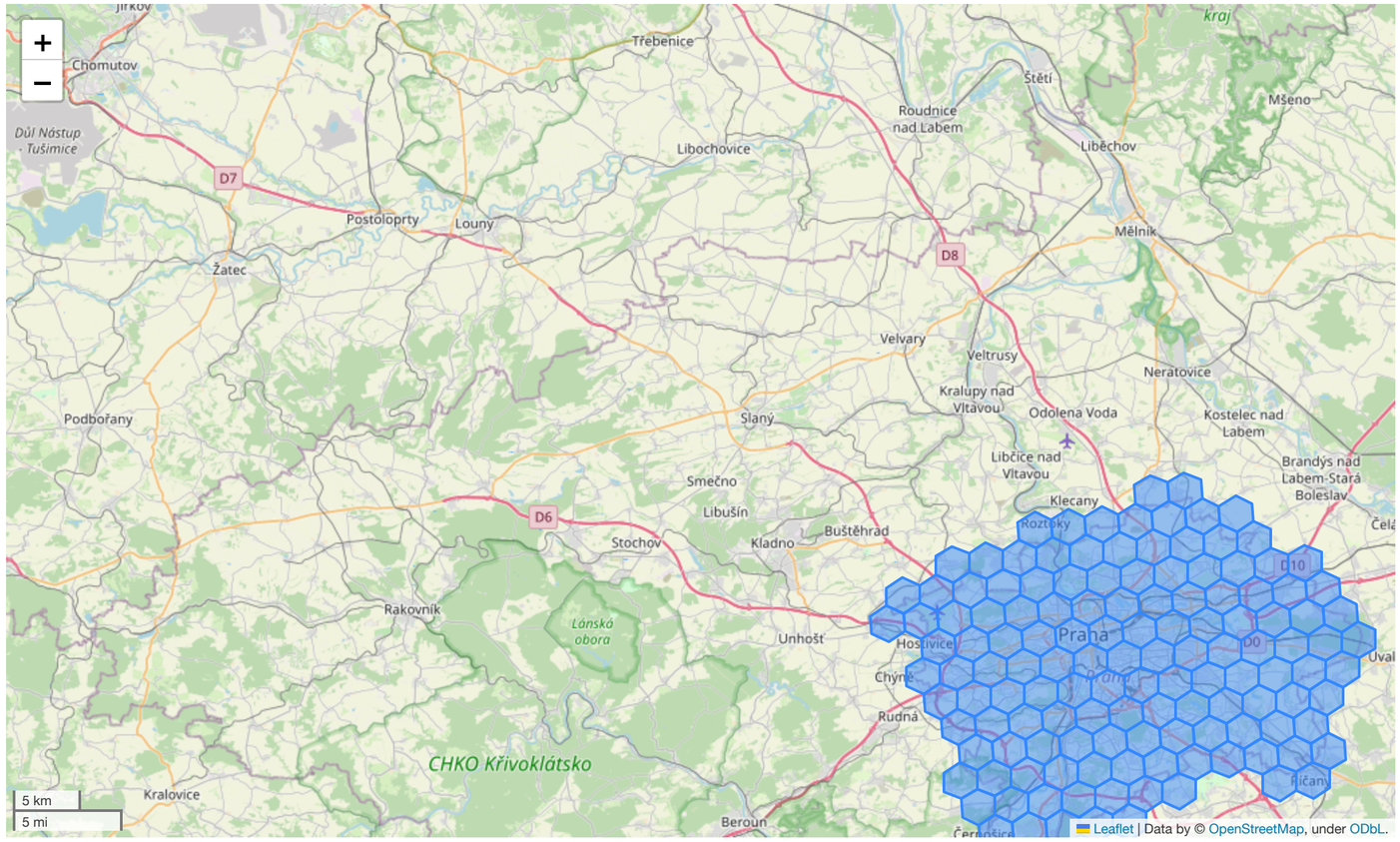
Spatial joins

Spatial joins

In [27]:
```
from srai.regionalizers import geocode_to_region_gdf, H3Regionalizer
from utils import CB_SAFE_PALLETE

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

Out[27]:



Visit our research group at `kraina.ai`

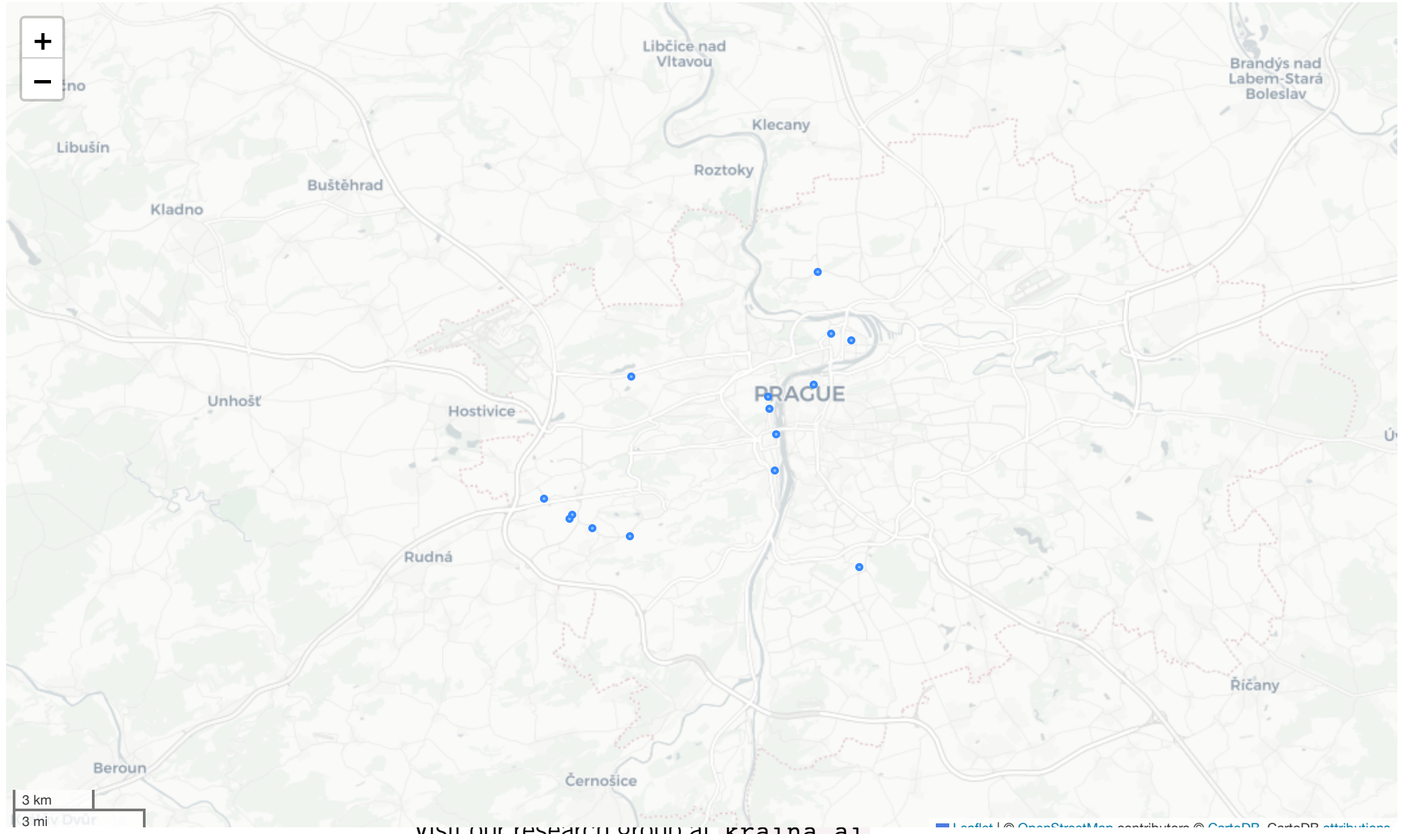
Get bicycle data for Prague

In [31]:
```
from srai.loaders import OSMOnlineLoader

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

```
Downloading amenity: bicycle_rental: 100%|██████████| 1/1 [00:00<00:00,  8.23it/s]
```

Out[31]:



Perform the join

Perform the join

In [29]:
```
joint_gdf = regionized.sjoin(prague_bikes)
joint_gdf
```

Out[29]:

region_id	geometry	index_right	amenity
871e35403ffffff	POLYGON ((14.45921 50.13090, 14.44170 50.13595...	node/2710869649	bicycle_rental
871e354a3ffffff	POLYGON ((14.32607 50.05235, 14.30859 50.05739...	node/7709737634	bicycle_rental
871e354a3ffffff	POLYGON ((14.32607 50.05235, 14.30859 50.05739...	node/7602452911	bicycle_rental
871e354a3ffffff	POLYGON ((14.32607 50.05235, 14.30859 50.05739...	node/7602452912	bicycle_rental
871e3541effffff	POLYGON ((14.39260 50.09165, 14.39328 50.07975...	node/7476047587	bicycle_rental
871e3541effffff	POLYGON ((14.39260 50.09165, 14.39328 50.07975...	node/6742097486	bicycle_rental
871e3541dffffff	POLYGON ((14.46122 50.09522, 14.47805 50.10205...	node/6466614391	bicycle_rental
871e354adffffff	POLYGON ((14.41077 50.07470, 14.39328 50.07975...	node/6423051257	bicycle_rental
871e354adffffff	POLYGON ((14.41077 50.07470, 14.39328 50.07975...	node/7724483398	bicycle_rental
871e3541cffffff	POLYGON ((14.42555 50.11723, 14.40873 50.11039...	node/4323071933	bicycle_rental
871e354aeffffff	POLYGON ((14.32607 50.05235, 14.32677 50.04045...	node/7709737635	bicycle_rental
871e35418ffffff	POLYGON ((14.42758 50.08154, 14.44507 50.07649...	node/3401436639	bicycle_rental
871e35416ffffff	POLYGON ((14.32400 50.08804, 14.34080 50.09489...	node/3393503797	bicycle_rental
871e354f0ffffff	POLYGON ((14.48205 50.03065, 14.48138 50.04256...	node/8043766186	bicycle_rental
871e354a0ffffff	POLYGON ((14.27361 50.06747, 14.27431 50.05557...	node/2289555326	bicycle_rental

Count bike stations

Count bike stations

In [30]:
```
regionized.sjoin(prague_bikes).groupby("region_id").size()
```

Out[30]:
```
region_id
871e35403fffffff    1
871e35416fffffff    1
871e35418fffffff    1
871e3541cfffffff    1
871e3541dfffffff    1
871e3541effffff     2
871e354a0fffffff    1
871e354a3fffffff    3
871e354adfffffff    2
871e354aefffffff    1
871e354f0fffffff    1
dtype: int64
```

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
- Spatial indexes
- Spatial operations
    - both on Shapely objects and GeoDataFrames
    - relationships such as `within`, `intersects`
    - spatial joins