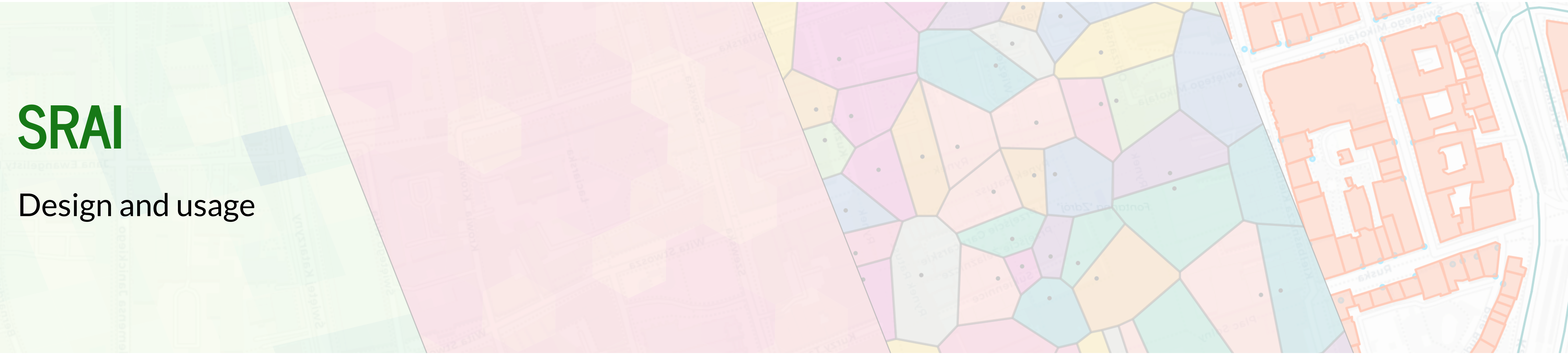
SRAI

Design and usage



What is SRAI?

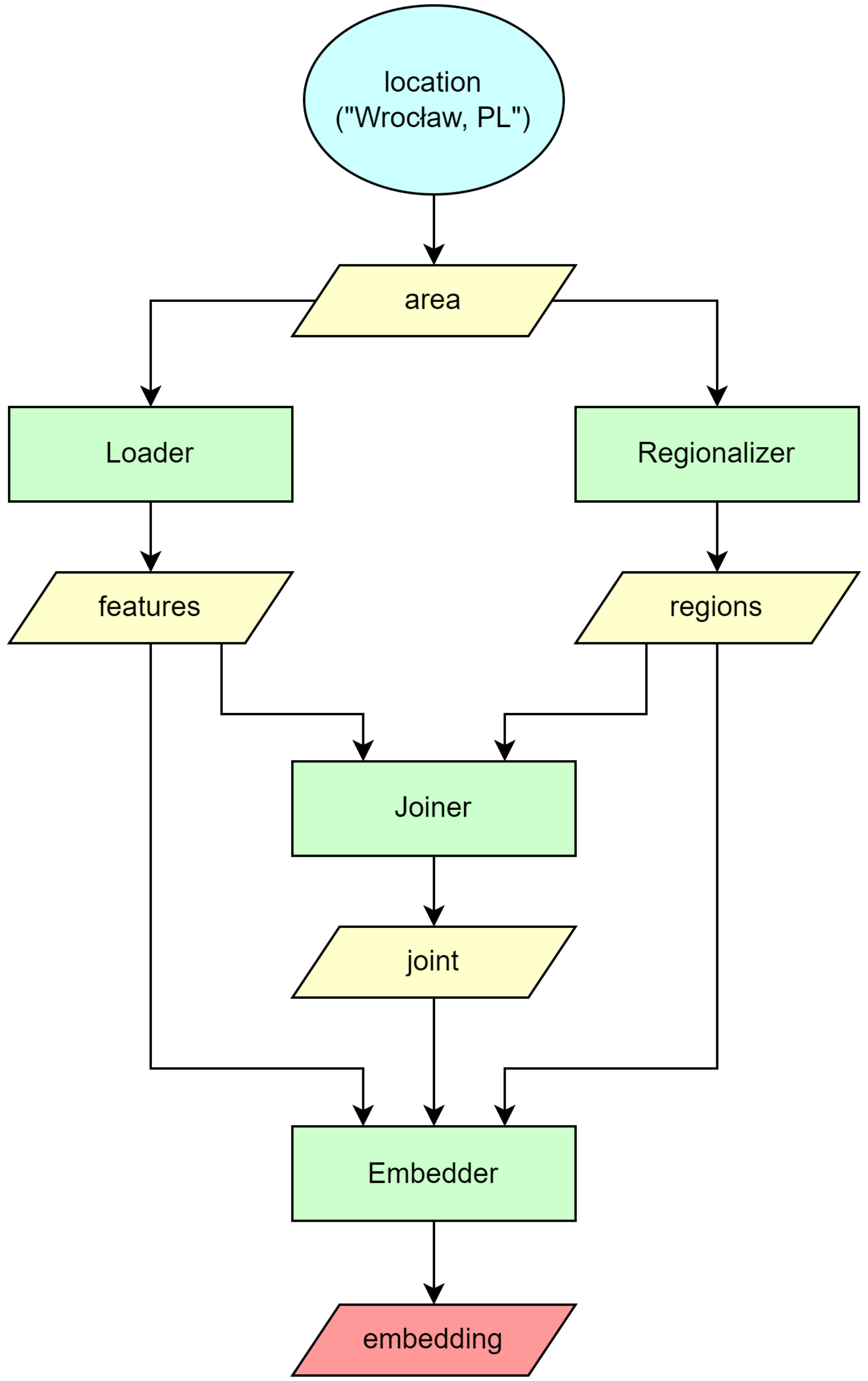
What is SRAI?

A toolbox for geospatial AI

What is SRAI?

A toolbox for geospatial AI

that aims to standardize the domain and make your life easier

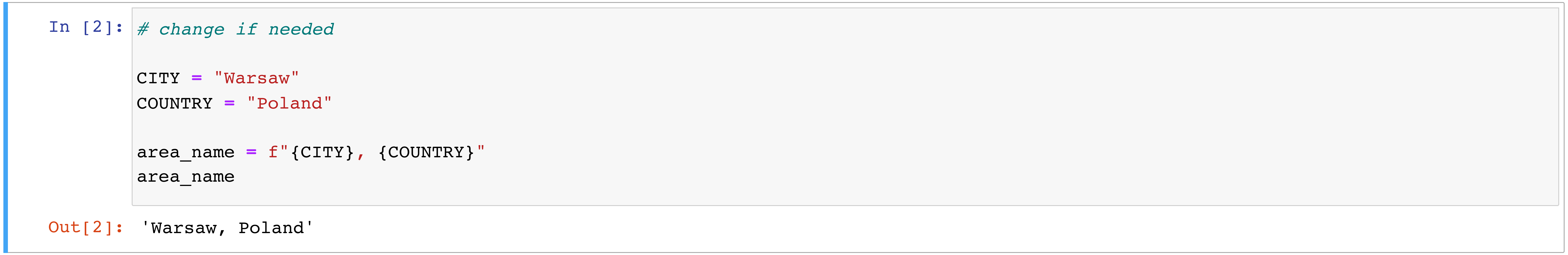

location
("Wrocław, PL")
area
Loader
Regionalizer
features
regions
Joiner
joint
Embedder
embedding

Task 0

Specify the city you want to work on for the rest of the exercise.

Task 0

Specify the city you want to work on for the rest of the exercise.



In [2]:
```
# change if needed

CITY = "Warsaw"
COUNTRY = "Poland"

area_name = f"{CITY}, {COUNTRY}"
area_name
```

Out[2]:
```
'Warsaw, Poland'
```

Task 1

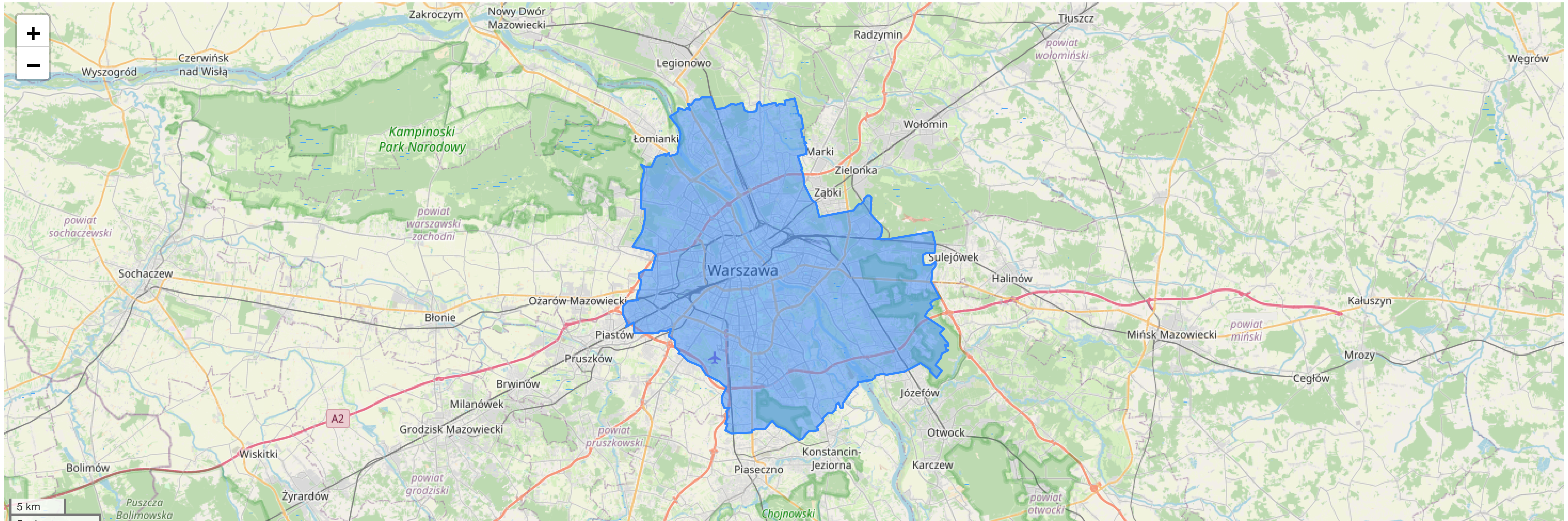
Now download the area's polygon based on the `area_name` specified above. Use `geocode_to_region_gdf` from `srai.regionalizers`.

Task 1

Now download the area's polygon based on the `area_name` specified above. Use `geocode_to_region_gdf` from `srai.regionalizers`.

In [3]:
```
area = None

### BEGIN SOLUTION
from srai.regionalizers import geocode_to_region_gdf

area = geocode_to_region_gdf(area_name)
### END SOLUTION

area.explore(height=500)
```

Out[3]:



Loaders

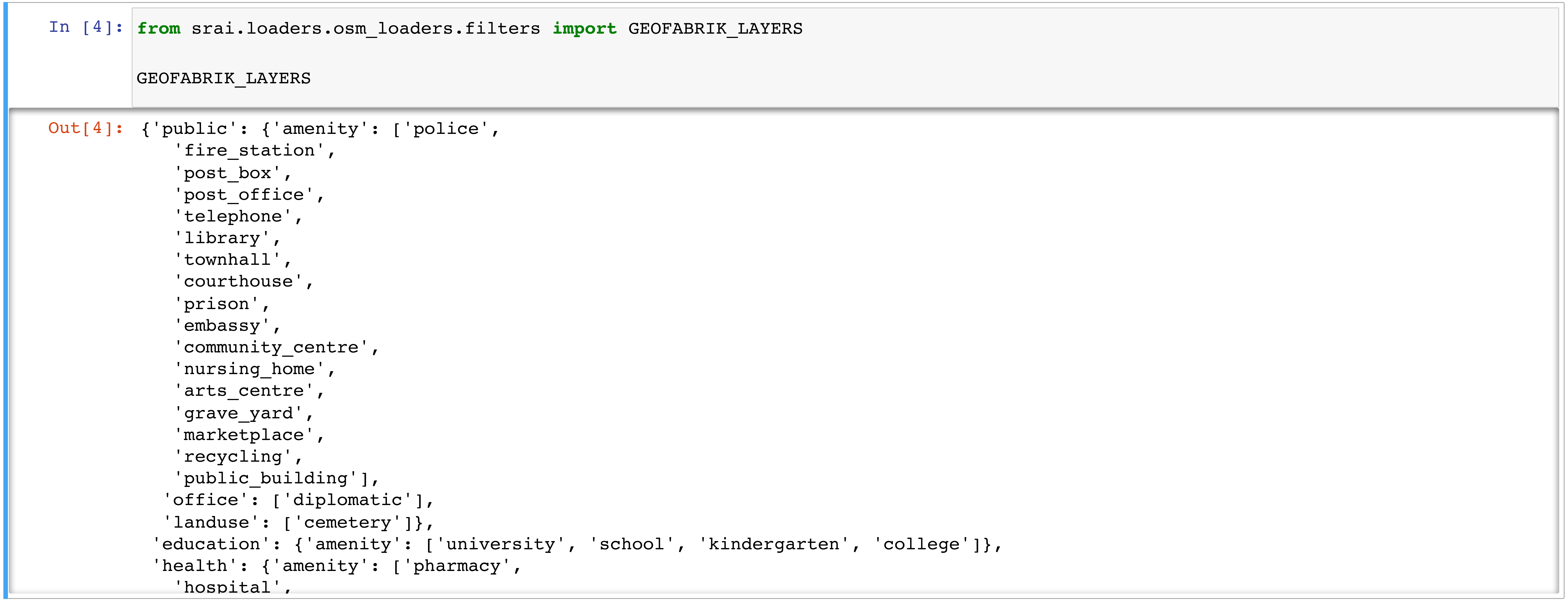
Loaders

- used to load spatial data from different sources
- unify loading into a single interface
- prepare data for the embedding methods

API
Examples

Types of loaders:

- GTFS
- OSM Online
- OSM Pbf
- OSM Way
- OSM Tile



```
In [4]: from srai.loaders.osm_loaders.filters import GEOFABRIK_LAYERS

        GEOFABRIK_LAYERS
```

```
Out[4]: {'public': {'amenity': ['police',
           'fire_station',
           'post_box',
           'post_office',
           'telephone',
           'library',
           'townhall',
           'courthouse',
           'prison',
           'embassy',
           'community_centre',
           'nursing_home',
           'arts_centre',
           'grave_yard',
           'marketplace',
           'recycling',
           'public_building'],
          'office': ['diplomatic'],
          'landuse': ['cemetery']},
         'education': {'amenity': ['university', 'school', 'kindergarten', 'college']},
         'health': {'amenity': ['pharmacy',
           'hospital',
```

Task 2

Let's create an A4 city poster using data about main road infrastructure and water.

Now that we have the city's boundries in `area` we can use them to fetch more data. For that task you can either use OSMOnlineLoader or OSMPbfLoader to `load` the data. Use the provided `tags`.

Additionally, as the loaded data is a bit bigger than the boundaries, clip it to the `area`.

In [5]:

```python
tags = {
    "highway": [
        "primary",
        "primary_link",
        "secondary",
        "secondary_link",
        "tertiary",
        "tertiary_link",
        "trunk",
        "trunk_link",
    ],
    "water": True,
    "waterway": True,
}

features = None

### BEGIN SOLUTION
from srai.loaders import OSMOnlineLoader

features = OSMOnlineLoader().load(area, tags).clip(area)
### END SOLUTION

features.head(3)
```

```
Downloading waterway: True          : 100%|██████████████████████████████████████████| 10/10 [00:02<00:00,  3.44it/s]
```

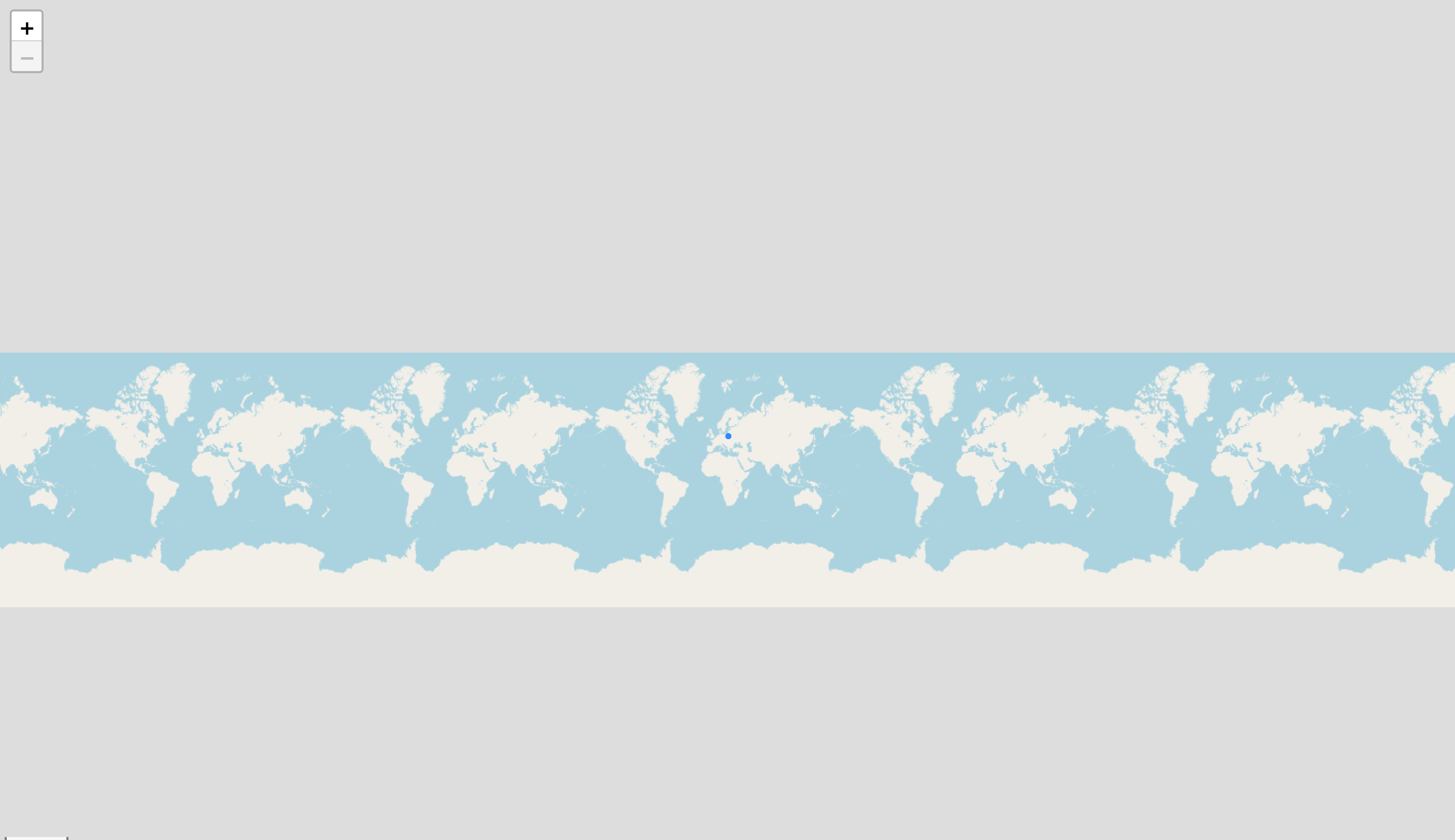
Out[5]:

	geometry	highway	water	waterway
feature_id				
way/913988111	POLYGON ((21.09793 52.10656, 21.09793 52.10656...	None	pond	None
way/624075792	LINESTRING (21.09721 52.10761, 21.09721 52.107...	None	None	stream
way/1051121778	LINESTRING (21.09772 52.11205, 21.09735 52.11244)	None	None	ditch

```
In [6]: features.explore()
```

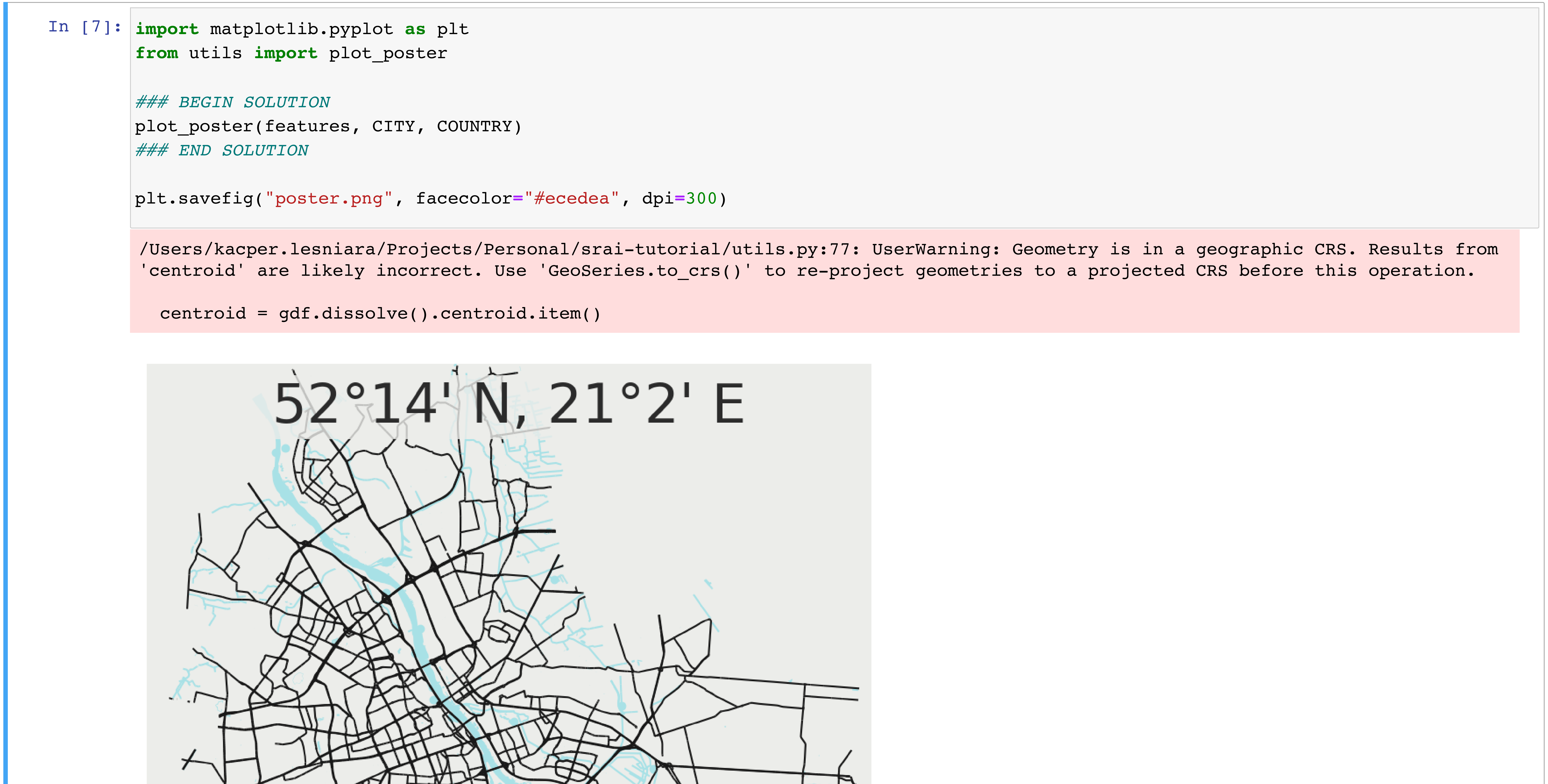
```
/Users/kacper.lesniara/Projects/Personal/srai-tutorial/venv/lib/python3.10/site-packages/folium/features.py:1102: UserWarning: GeoJsonTooltip is not configured to render for GeoJson GeometryCollection geometries. Please consider reworking these features: [{'feature_id': 'way/456262354', 'highway': None, 'water': None, 'waterway': 'river'}] to MultiPolygon for full functionality.
https://tools.ietf.org/html/rfc7946#page-9
  warnings.warn(
```

Out[6]:



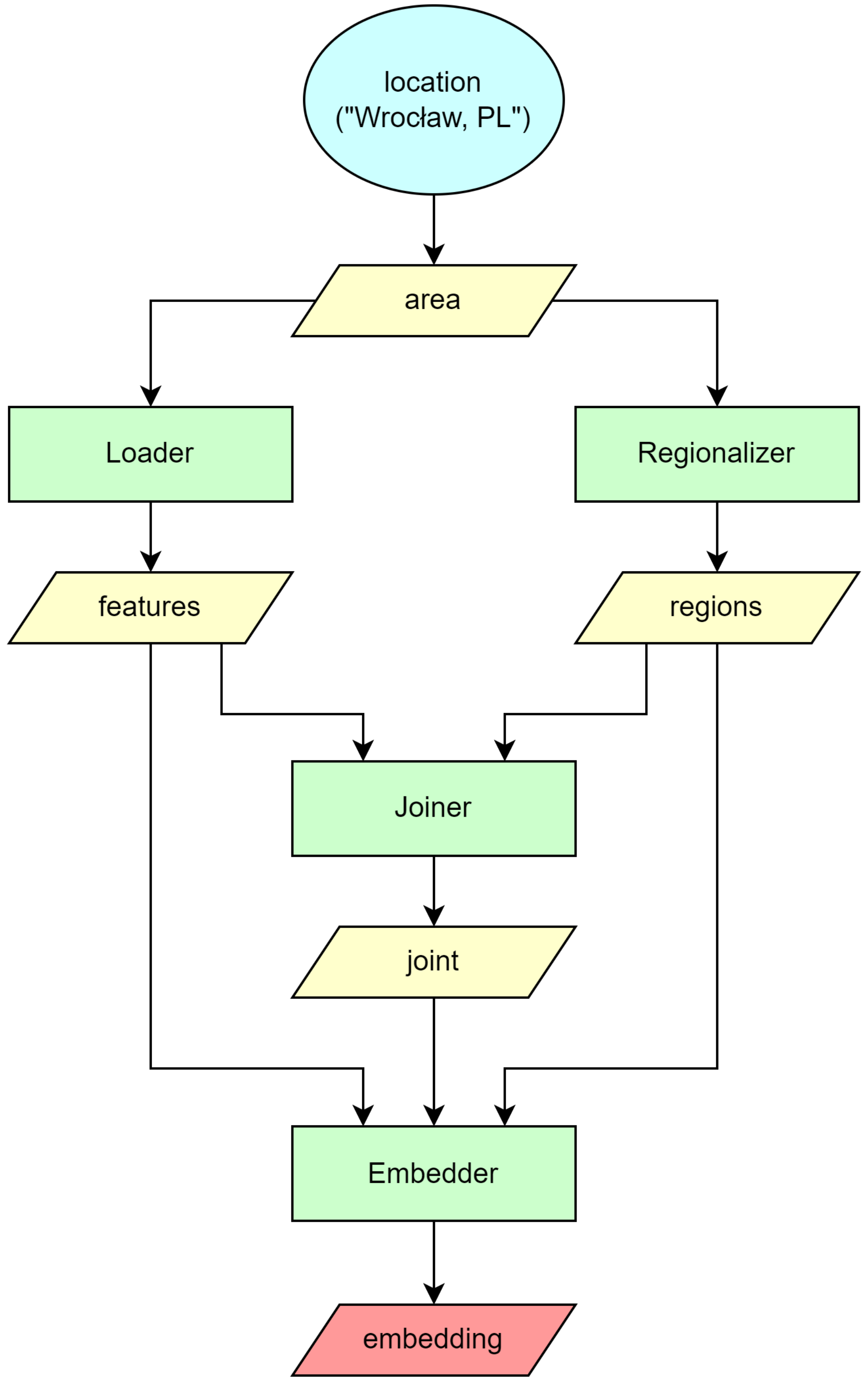
We've downloaded the data for the given boundaries. Now we can plot the actual poster. Use the `plot_poster` function.

We've downloaded the data for the given boundaries. Now we can plot the actual poster. Use the `plot_poster` function.



```
In [7]: import matplotlib.pyplot as plt
        from utils import plot_poster

        ### BEGIN SOLUTION
        plot_poster(features, CITY, COUNTRY)
        ### END SOLUTION

        plt.savefig("poster.png", facecolor="#ecedea", dpi=300)
```

```
/Users/kacper.lesniara/Projects/Personal/srai-tutorial/utils.py:77: UserWarning: Geometry is in a geographic CRS. Results from
'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this operation.

  centroid = gdf.dissolve().centroid.item()
```


location
("Wrocław, PL")
area
Loader
Regionalizer
features
regions
Joiner
joint
Embedder
embedding

Regionalizers

Regionalizers

- unify methods for dividing a given area into smaller regions.
- can be based on spatial indexes.

API
Examples

Types of regionalizers:

- H3
- S2
- Voronoi
- Administative Boundary

Task 3

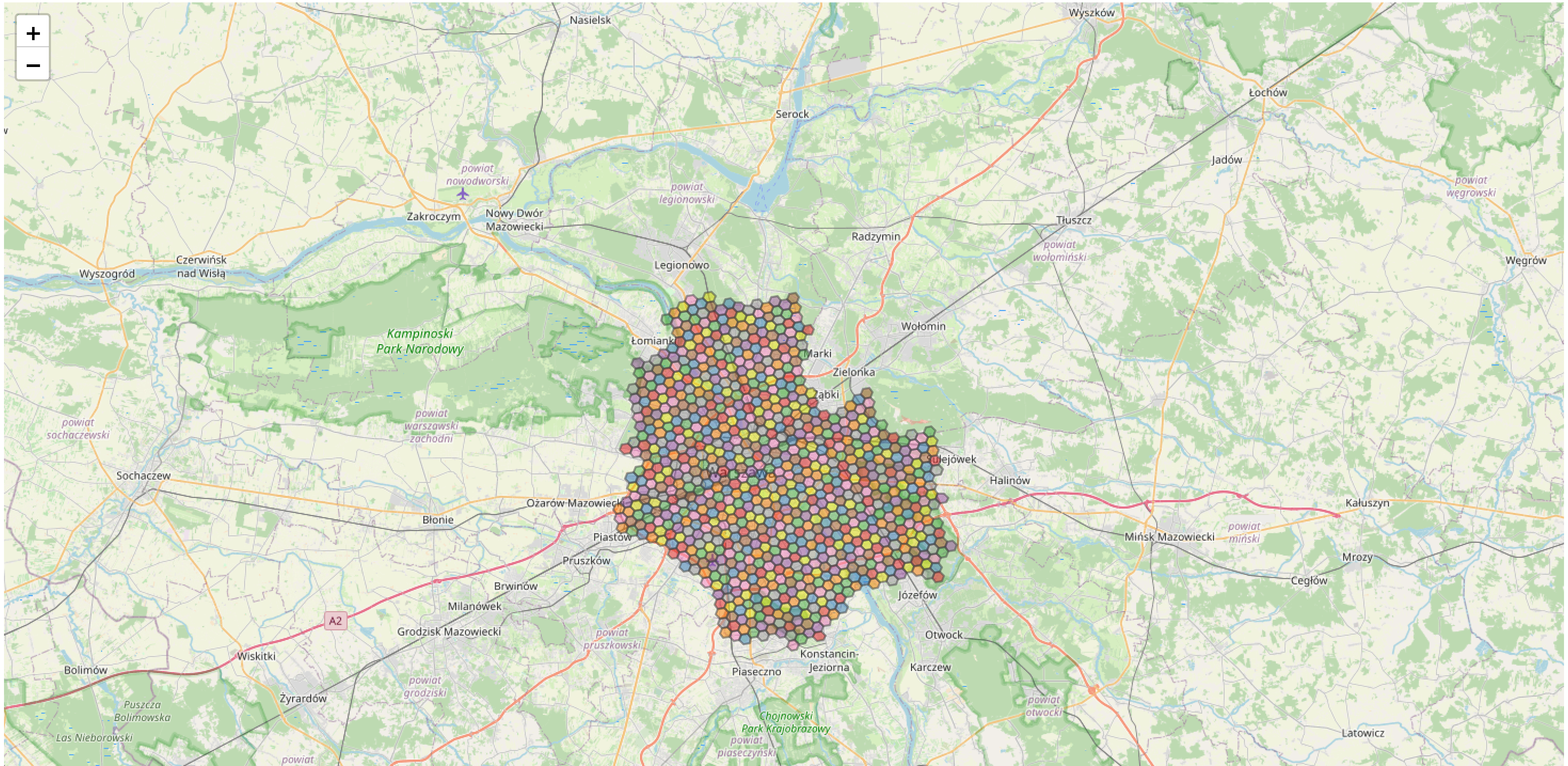
Let's divide our `area` into some regions. Looking above we have a couple of options, but we want you to focus mainly on H3Regionalizer and AdministrativeBoundryRegionalizer. Try using one of them (try both if you have the time) to `transform` our space. Both are available in `srai.regionalizers`. We suggest a `resolution=8` or `admin_level=9`, but feel free to experiment.

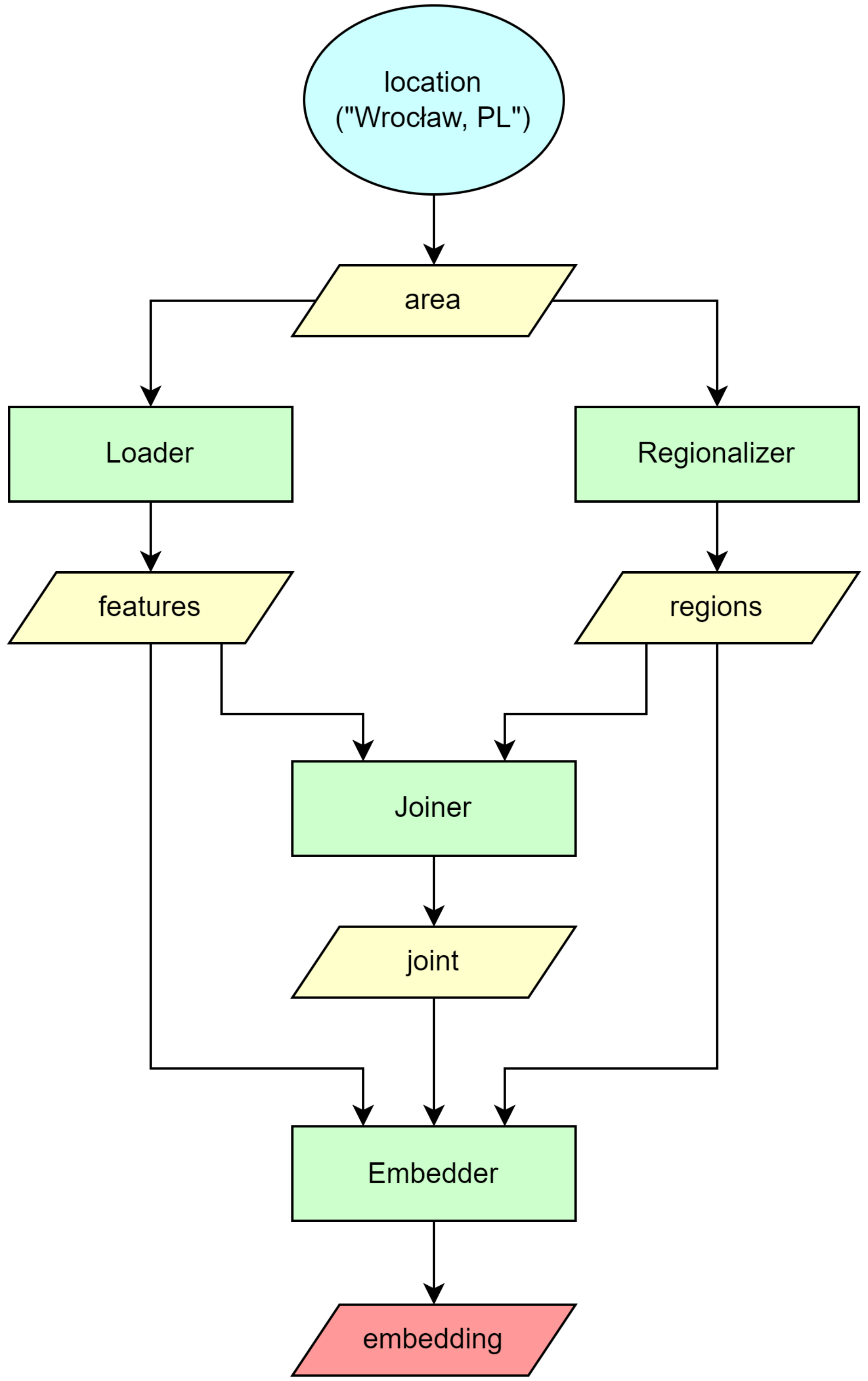
To plot the regions use plot_regions from `srai.plotting`. Use the provided pallete as a `colormap`.

In [8]:

```
from utils import CB_SAFE_PALLETE

regions = None

### BEGIN SOLUTION
from srai.regionalizers import H3Regionalizer
from srai.plotting import plot_regions

regions = H3Regionalizer(8).transform(area)
plot_regions(regions, colormap=CB_SAFE_PALLETE)
### END SOLUTION
```

Out[8]:




location
("Wrocław, PL")
area
Loader
Regionalizer
features
regions
Joiner
joint
Embedder
embedding

Embedders

Embedders

Unify methods for mapping regions into a vector space.

API
Examples

Types of embedders:

- Count
- Contextual Count
- GTFS2Vec
- Hex2Vec
- Highway2Vec
- GeoVex

Task 4

Now that we have `regions` and `features` we can try to combine (intersect) them together. This way we will know which feature lays within which region. Use IntersectionJoiner from `srai.joiners` to get the `joint` DataFrame.

Task 4

Now that we have `regions` and `features` we can try to combine (intersect) them together. This way we will know which feature lays within which region. Use IntersectionJoiner from `srai.joiners` to get the `joint` DataFrame.

In [9]:

```python
joint = None

### BEGIN SOLUTION
from srai.joiners import IntersectionJoiner

joint = IntersectionJoiner().transform(regions, features)
### END SOLUTION

joint.head(3)
```

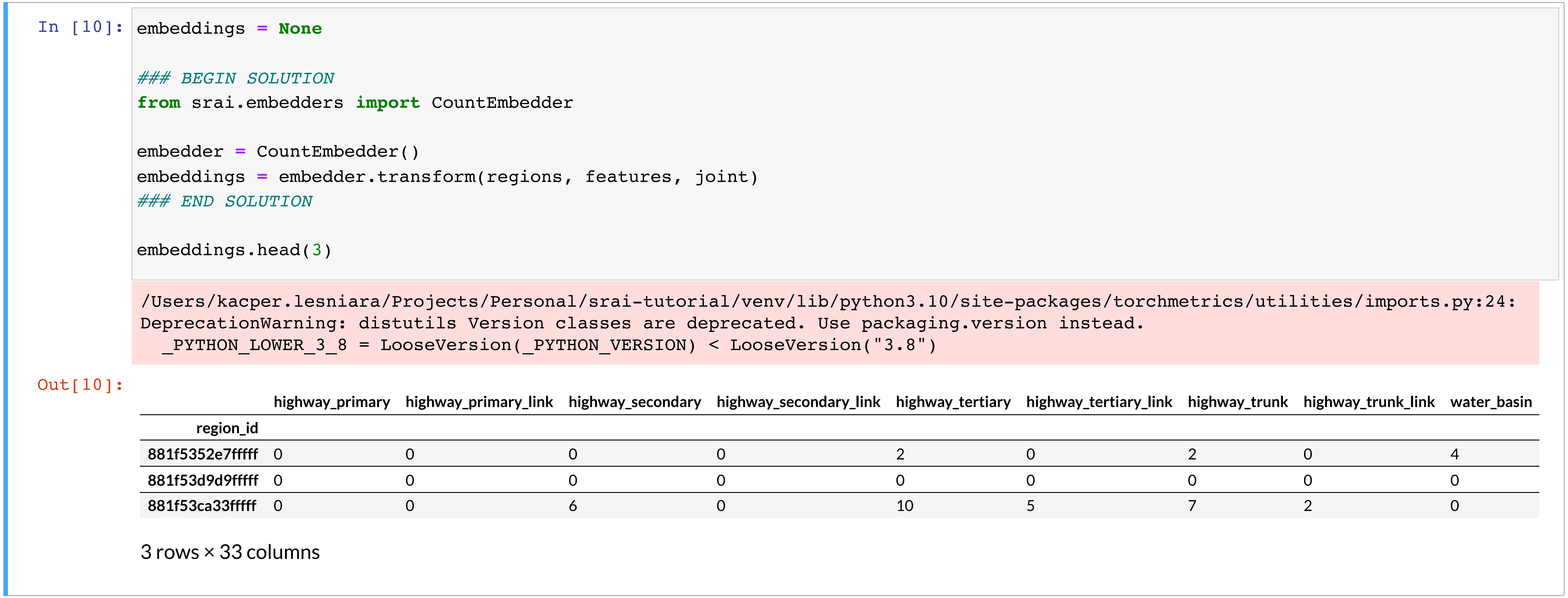
Out[9]:

region_id	feature_id
881f5352e7fffff	way/945954508
881f5352adfffff	way/945954508
881f5352e7fffff	way/1096147590

Task 5

Finally, we can combine results from the previous steps to create a geospatial embedding. There are a couple of methods to choose from, but let's use the simplest `embedder` - CountEmbedder from `srai.embedders`. It simply counts the occurences of features across regions.

With it, transform `regions`, `features` and `joint` into the `embeddings`.



In [10]:

```
embeddings = None

### BEGIN SOLUTION
from srai.embedders import CountEmbedder

embedder = CountEmbedder()
embeddings = embedder.transform(regions, features, joint)
### END SOLUTION

embeddings.head(3)
```

```
/Users/kacper.lesniara/Projects/Personal/srai-tutorial/venv/lib/python3.10/site-packages/torchmetrics/utilities/imports.py:24:
DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  _PYTHON_LOWER_3_8 = LooseVersion(_PYTHON_VERSION) < LooseVersion("3.8")
```

Out[10]:

	highway_primary	highway_primary_link	highway_secondary	highway_secondary_link	highway_tertiary	highway_tertiary_link	highway_trunk	highway_trunk_link	water_basin
region_id									
881f5352e7fffff	0	0	0	0	2	0	2	0	4
881f53d9d9fffff	0	0	0	0	0	0	0	0	0
881f53ca33fffff	0	0	6	0	10	5	7	2	0

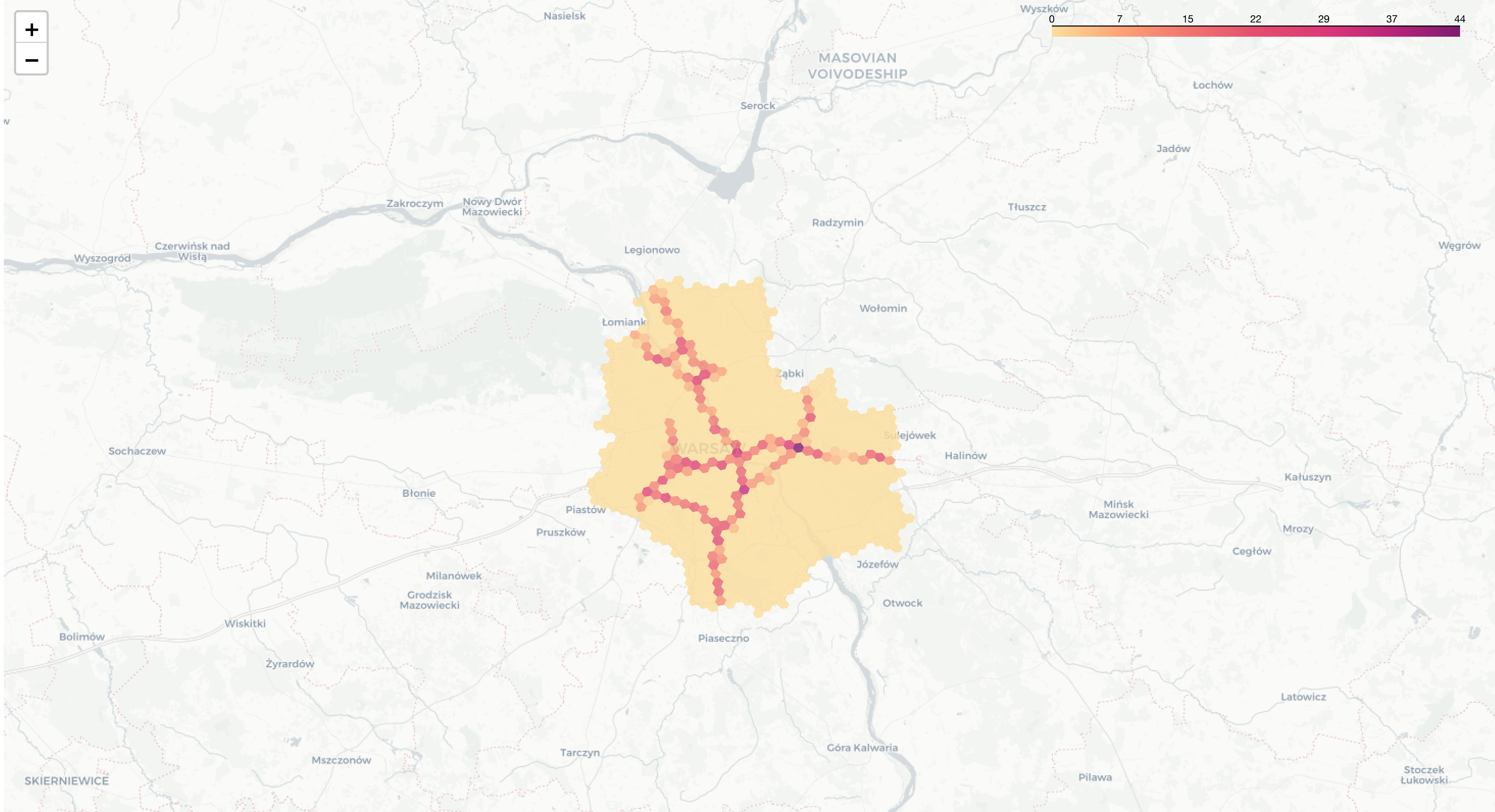
3 rows × 33 columns

It would be nice to see the results. Use plot_numeric_data from `srai.plotting` to visualize the embeddings. As a `data_column` choose one of the columns available in the `embeddings` DataFrame.

In [11]:

```
data_column = None

### BEGIN SOLUTION
from srai.plotting import plot_numeric_data

data_column = "highway_primary"
plot_numeric_data(regions, data_column, embeddings)
### END SOLUTION
```

Out[11]:



Summary

Good job! You managed to use all of the building blocks of `srai` to create an entire pipeline - from only a name of a city, to embeddings of regions in it.

Questions?