In [1]:
```
import geopandas as gpd
import pandas as pd
```

Part 3 - Warsaw population analysis and visualization

This part will utilize geospatial knowledge presented earlier to prepare a visualization of Warsaw population distribution

Task 1

Load Warsaw census data and districts boundaries. At the end find the top 3 most populated districts and plot their boundaries on the map

Data files:

- `../../data/warsaw_population.json`
- `../../data/warsaw_districts.geojson`

Remember to set CRS when loading GeoDataFrames. For most cases, the best choice is `WGS84/EPSG:4326` (same crs, two different names). You can set it using `to_crs()` function, for example `gdf.to_crs(epsg=4326)`

In [2]:

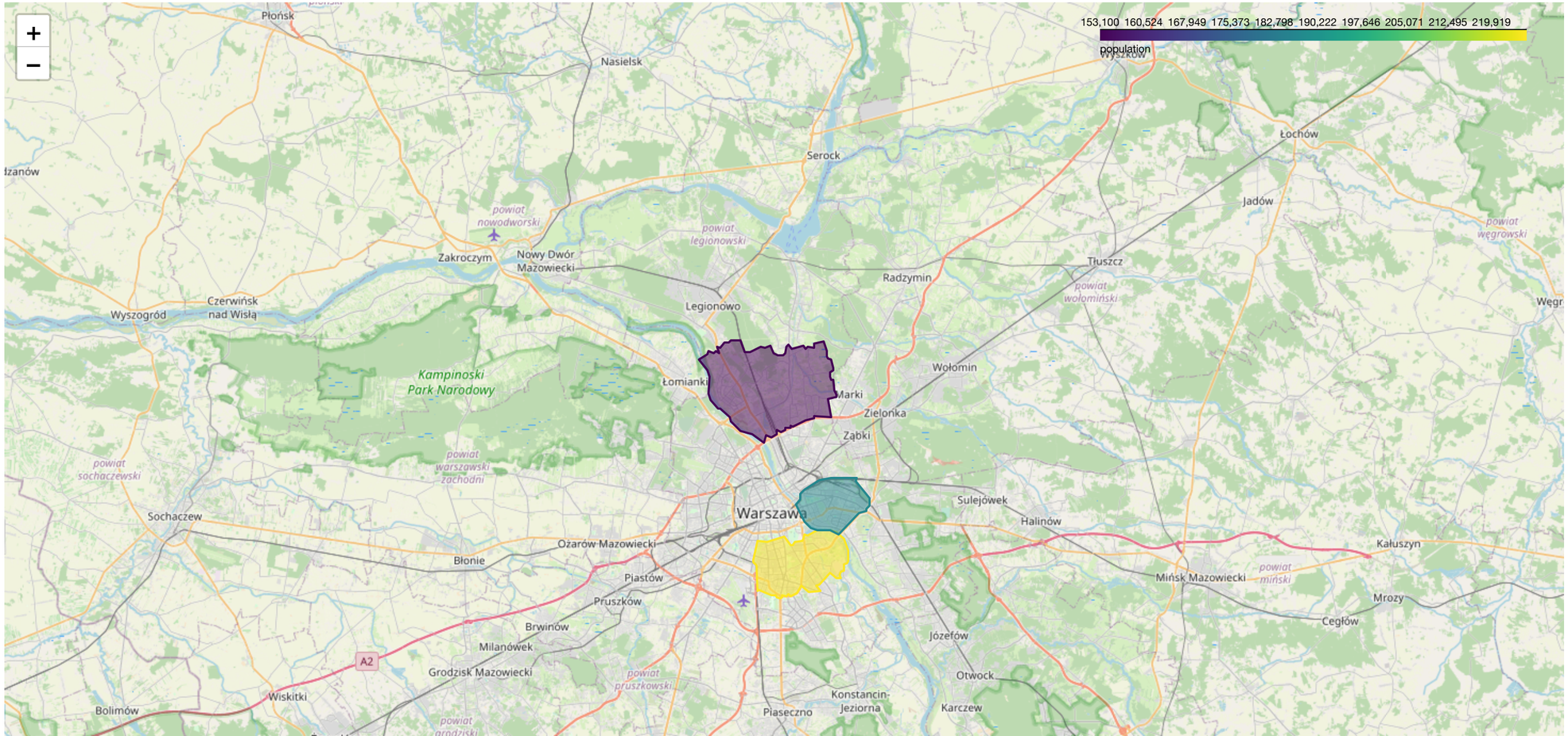
```
warsaw_population = ...
warsaw_districts = ...

# BEGIN SOLUTION
warsaw_population = pd.read_json('../../data/warsaw_population.json')
warsaw_districts = gpd.read_file('../../data/warsaw_districts.geojson').to_crs(epsg=4326)

warsaw_districts = warsaw_districts.merge(warsaw_population, on='district', how='inner')

top_3_districts = warsaw_districts.sort_values(by='population', ascending=False).head(3)
top_3_districts.explore("population")
# END SOLUTION
```

Out[2]:



Task 2

Load all buildings in Warsaw. You can use `OSMOnlineLoader` from the `srai` library.

In [3]:

```python
from srai.loaders.osm_loaders import OSMOnlineLoader

loader = OSMOnlineLoader()

osm_building_types = [
    "residential",
    "apartments",
    "house",
    "semidetached_house",
    "detached",
]

osm_filter = {
    "building": osm_building_types,
    "building:levels": True,
}
```

In [4]:

```
warsaw_polygon = ...  # merge all districts into one polygon
buildings = ... # and load osm data for this polygon

# BEGIN SOLUTION
warsaw_polygon = warsaw_districts.unary_union
buildings = loader.load(warsaw_polygon, osm_filter)
# END SOLUTION

buildings.head()
```

```
Downloading building:levels: True       : 100%|██████████| 6/6 [02:55<00:00, 29.23s/it]
```

Out[4]:

feature_id	geometry	building	building:levels
node/1803205615	POINT (21.04920 52.18866)	None	1
node/2475059251	POINT (20.93873 52.32488)	None	1
node/7620615880	POINT (20.88417 52.27038)	None	1
node/9221345215	POINT (21.03001 52.17558)	None	1
node/9963890819	POINT (20.94447 52.33040)	None	2

OSM loader looks at those two tags (*building* and *building:levels*) independently. We need to clean the result to leave only entries with both of those tags.

For cleaning we should assume that:

- we skip buildings of unknown type
- buildings without levels are assumed to have 1 floor
- levels should be integers

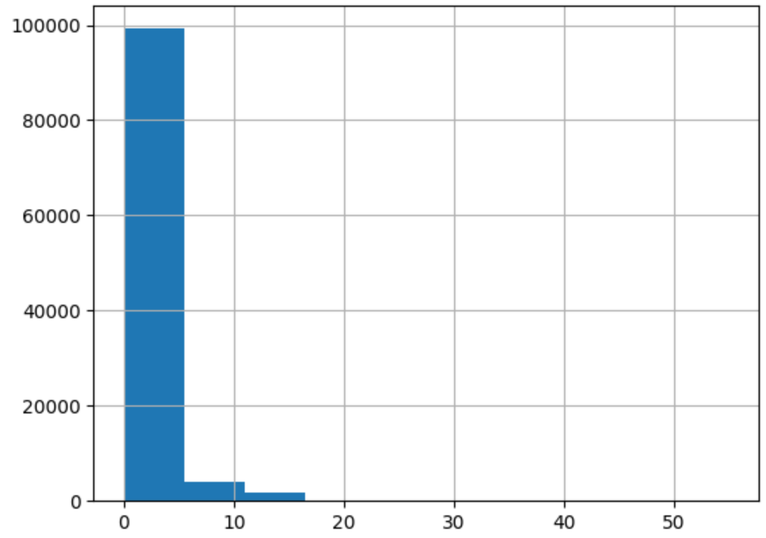
In [5]:
```
# BEGIN SOLUTION
import math

buildings = buildings[buildings["building"].isin(osm_building_types)]
buildings = buildings.fillna(1)

buildings["building:levels"] = buildings["building:levels"].map(lambda x: math.ceil(float(x))).astype(int)
# END SOLUTION

buildings["building:levels"].hist()
```

Out[5]: <Axes: >



Task 3

Approximate the distribution of population across buildings. We will do this in four steps:

- Calculate *inhabited_area* of each building, which we understand as a multiplication of its area by the number of floors. This is based on an assumption that in taller building lives more people
- Simplify each building to the single point on the map instead of a polygon
- Calculate *total_inhabited_area* for each district
- Calculate population of each building from the equation: `population = district_population * inhabited_area / total_inhabited_area`

TIP: Remember that all geometrical calculations (area, circumference, centroid) requires proper CRS setting (`epsg=2180` is best for Poland). After that, bring the results back to `epsg=4326`

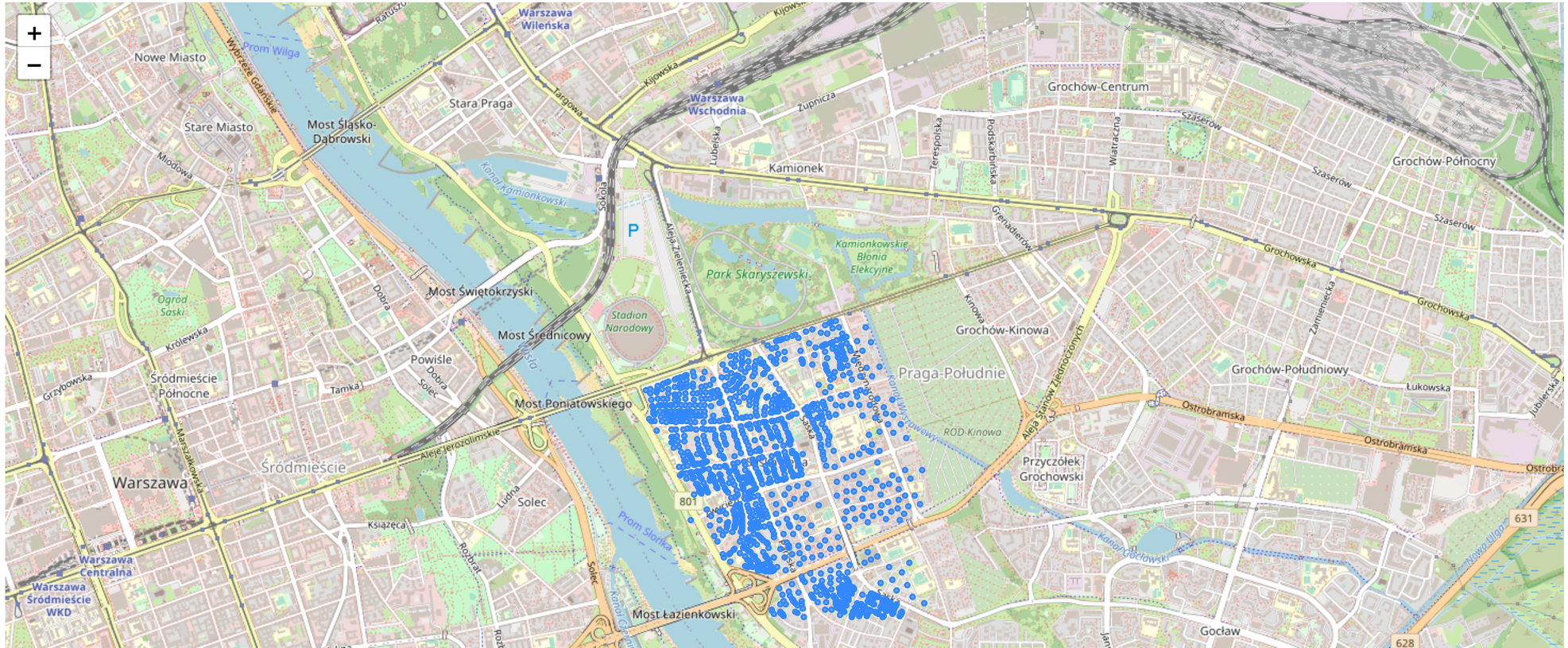
In [6]:
```
# Start with first two tasks - caclulate the inhabited_area and convert buildings to points

# BEGIN SOLUTION
buildings["inhabited_area"] = buildings.to_crs(epsg=2180).area * buildings["building:levels"]
buildings["full_geometry"] = buildings.geometry
buildings["geometry"] = buildings.to_crs(epsg=2180).centroid.to_crs(epsg=4326)
# END SOLUTION

# This is some magic coordinates to zoom on a part of Warsaw
xmin = 21.042753111534097
xmax = 21.069257679735955
ymin = 52.24187245384607
ymax = 52.22533280016626

buildings.cx[xmin:xmax, ymin:ymax].explore()
```

Out[6]:



In [7]:
```
# Next, match buildings to districts, using spatial join operation and calculate total_inhabited_area for each district

buildings_with_districts = ...

# BEGIN SOLUTION
buildings_with_districts = buildings.sjoin(warsaw_districts, predicate='within')
# END SOLUTION

buildings_with_districts.head()
```

Out[7]:

feature_id	geometry	building	building:levels	inhabited_area	full_geometry	index_right	district	population
relation/68308	POINT (21.01596 52.22981)	apartments	5	4165.807370	POLYGON ((21.01617 52.22986, 21.01625 52.22970...	17	Śródmieście	101979
relation/71497	POINT (21.00041 52.23138)	apartments	5	5490.160887	POLYGON ((21.00061 52.23126, 21.00058 52.23126...	17	Śródmieście	101979
relation/71498	POINT (21.00132 52.23155)	apartments	5	5325.160605	POLYGON ((21.00133 52.23138, 21.00132 52.23139...	17	Śródmieście	101979
relation/75094	POINT (21.02276 52.21426)	apartments	8	20812.253944	POLYGON ((21.02225 52.21414, 21.02224 52.21414...	17	Śródmieście	101979
relation/1566190	POINT (21.02198 52.22647)	apartments	5	5514.583870	POLYGON ((21.02221 52.22655, 21.02222 52.22655...	17	Śródmieście	101979

In [8]:
```
# Calculate total_inhabited_area for each district

totals_in_districts = ...

# BEGIN SOLUTION
totals_in_districts = buildings_with_districts.groupby('district')['inhabited_area'].sum().rename("total_inhabited_area")
# END SOLUTION

totals_in_districts.head()
```

Out[8]:
```
district
Bemowo        5.968326e+06
Białołęka     8.228564e+06
Bielany       5.713598e+06
Mokotów       1.270853e+07
Ochota        4.148456e+06
Name: total_inhabited_area, dtype: float64
```

In [9]:
```
# Finally, calculate population in each building

buildings_with_population = ...

# BEGIN SOLUTION
buildings_with_population = buildings_with_districts.merge(totals_in_districts, on="district", how="inner")
buildings_with_population["population_in_building"] = (
    buildings_with_population["population"] * buildings_with_population["inhabited_area"] / buildings_with_population["total_inhab
).round()
# END SOLUTION

buildings_with_population.head()
```

Out[9]:

	geometry	building	building:levels	inhabited_area	full_geometry	index_right	district	population	total_inhabited_area	population_in_building
0	POINT (21.01596 52.22981)	apartments	5	4165.807370	POLYGON ((21.01617 52.22986, 21.01625 52.22970...	17	Śródmieście	101979	6.793394e+06	63.0
1	POINT (21.00041 52.23138)	apartments	5	5490.160887	POLYGON ((21.00061 52.23126, 21.00058 52.23126...	17	Śródmieście	101979	6.793394e+06	82.0
2	POINT (21.00132 52.23155)	apartments	5	5325.160605	POLYGON ((21.00133 52.23138, 21.00132 52.23139...	17	Śródmieście	101979	6.793394e+06	80.0
3	POINT (21.02276 52.21426)	apartments	8	20812.253944	POLYGON ((21.02225 52.21414, 21.02224 52.21414...	17	Śródmieście	101979	6.793394e+06	312.0
4	POINT (21.02198 52.22647)	apartments	5	5514.583870	POLYGON ((21.02221 52.22655, 21.02222 52.22655...	17	Śródmieście	101979	6.793394e+06	83.0

Task 4

Aggregate data to H3 cells. This will allow us to prepare higher quality visualizations. Task to do:

- convert each building (point) to it's corresponding H3 cell ID (cellect appropriate resolution, you can look here for reference)
- calculate sum of population in each cell
- create a geometry column for each H3 cell

In [10]:
```
import h3
```

In [10]:
```
import h3
```

In [11]:
```
# BEGIN SOLUTION
buildings_with_population["h3"] = buildings_with_population.apply(
    lambda row: h3.latlng_to_cell(row.geometry.y, row.geometry.x, 9), axis=1
)
# END SOLUTION

buildings_with_population["h3"].value_counts()
```

Out[11]:
```
h3
891f53cd233ffff    224
891f5352663ffff    196
891f53cd22bffff    190
891f53c98c7ffff    186
891f53c855bffff    178
                  ...
891f5234b93ffff      1
891f53ca6cbffff      1
891f522647bffff      1
891f5234823ffff      1
891f522652bffff      1
Name: count, Length: 3658, dtype: int64
```

In [12]:
```
h3_population = ...

# BEGIN SOLUTION
h3_population = (
    buildings_with_population.groupby("h3")["population_in_building"]
    .sum()
    .rename("population_in_h3")
    .reset_index()
)
# END SOLUTION

h3_population["population_in_h3"].sum()
```

Out[12]:
```
1862875.0
```

In [13]:
```
from srai.h3 import h3_to_geoseries
```

In [13]:
```
from srai.h3 import h3_to_geoseries
```

In [14]:
```
h3_population_gdf = ...

# BEGIN SOLUTION
h3_population_gdf = gpd.GeoDataFrame(h3_population, geometry=h3_to_geoseries(h3_population.h3))
# END SOLUTION

h3_population_gdf.head()
```

Out[14]:

	h3	population_in_h3	geometry
0	891f52240a7ffff	11.0	POLYGON ((20.93781 52.15899, 20.93775 52.15733...
1	891f52240b3ffff	18.0	POLYGON ((20.93021 52.16150, 20.93016 52.15984...
2	891f52240b7ffff	109.0	POLYGON ((20.93534 52.16149, 20.93528 52.15983...
3	891f5224113ffff	1.0	POLYGON ((20.95566 52.15644, 20.95560 52.15478...
4	891f5224117ffff	23.0	POLYGON ((20.96079 52.15642, 20.96073 52.15476...

Task 5

Show data on the map. You can test 2 different approaches to plotting:

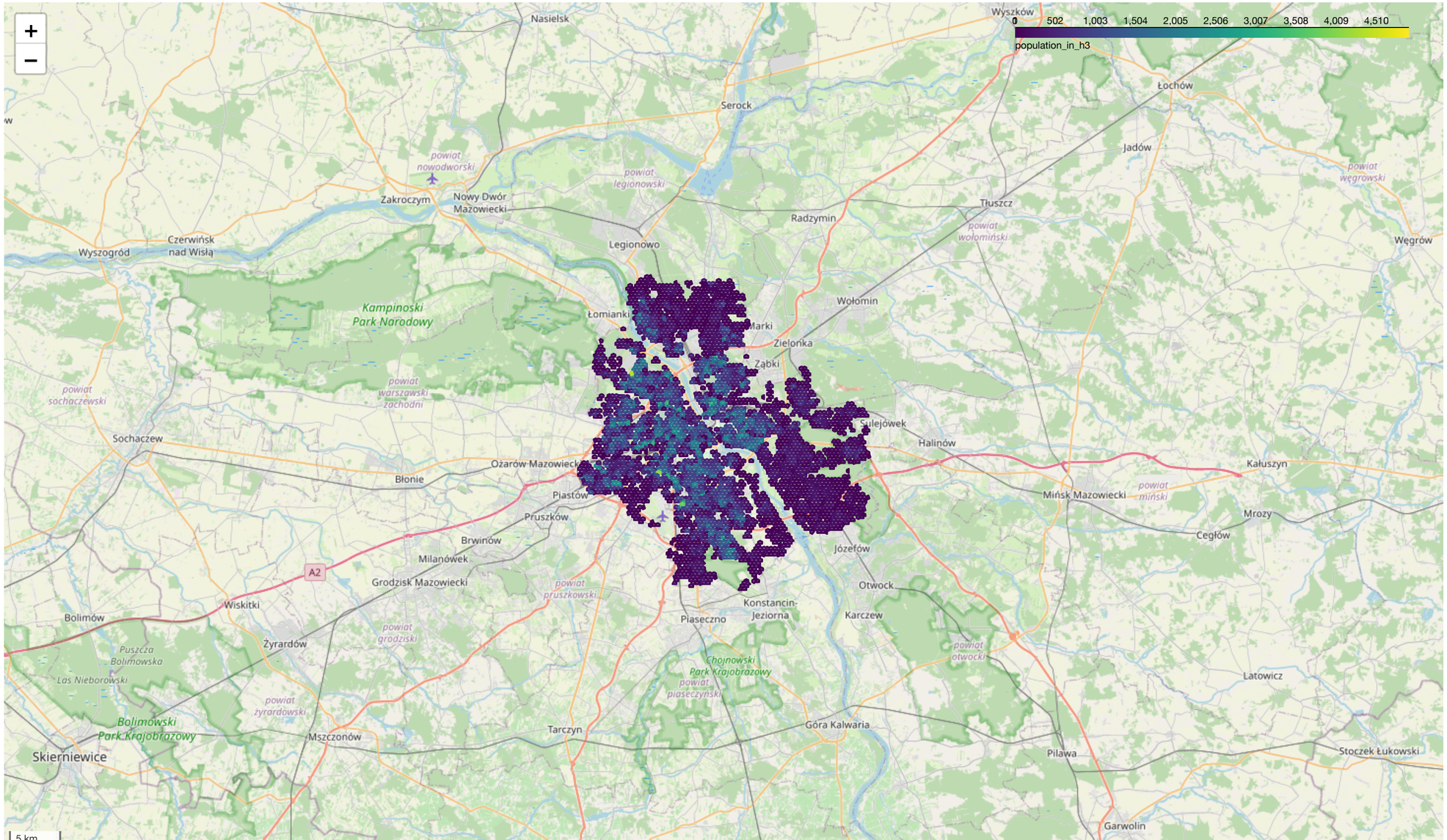
- built-in `.explore()` method from `GeoPandas`
- plotting function from `srai.plotting`

At the end we prepared an advanced, 3D map visualization based on Deck.gl

In [15]:
```
# base folium - explore


# BEGIN SOLUTION
h3_population_gdf.explore("population_in_h3")
# END SOLUTION
```

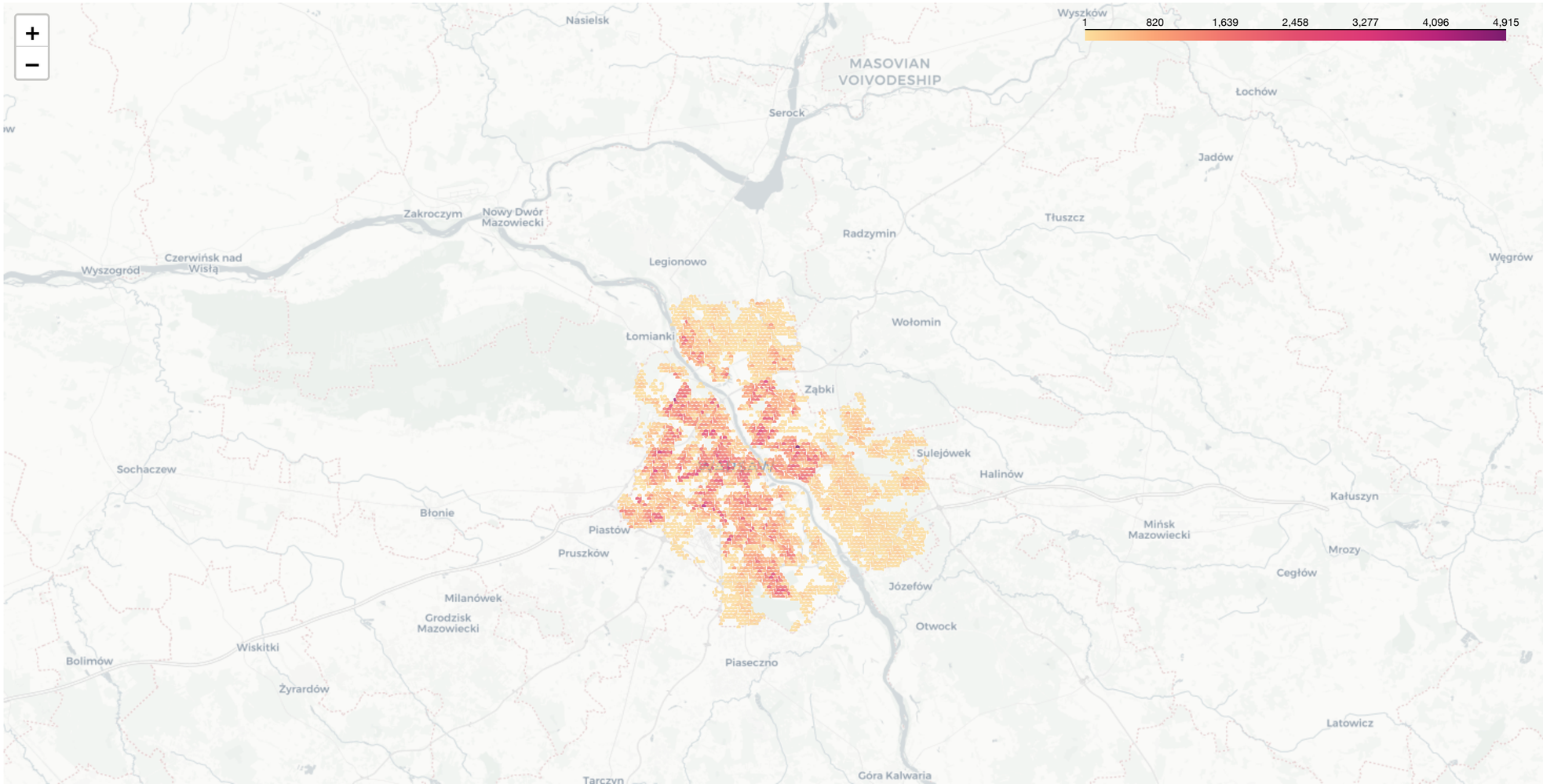
Out[15]:



In [16]:
```python
# srai - plot numeric
# TIP: the library requires the index with the name "region_id" to be set, use "h3" column for that
from srai.plotting import plot_numeric_data

# BEGIN SOLUTION
plot_numeric_data(
    h3_population_gdf.rename(columns={"h3": "region_id"}).set_index("region_id"),
    "population_in_h3"
)
# END SOLUTION
```

Out[16]:



In [17]:

```python
# pydeck 3d
from srai.plotting.folium_wrapper import _generate_linear_colormap
import plotly.express as px
import pydeck as pdk

colormap = _generate_linear_colormap(
    # https://plotly.com/python/builtin-colorscales/
    px.colors.sequential.Aggrnyl_r,
    min_value=h3_population_gdf["population_in_h3"].min(),
    max_value=h3_population_gdf["population_in_h3"].max(),
)

h3_population_gdf["color"] = h3_population_gdf["population_in_h3"].map(
    colormap.rgb_bytes_tuple
)

# Define a layer to display on a map
layer = pdk.Layer(
    "H3HexagonLayer",
    h3_population_gdf,
    pickable=True,
    stroked=True,
    filled=True,
    extruded=True,
    get_hexagon="h3",
    get_fill_color="[color[0], color[1], color[2], 204]",
    elevation_scale=0.5,
    get_elevation="population_in_h3",
    coverage=0.8,
)

# Set the viewport location
view_state = pdk.ViewState(
    latitude=52.2317, longitude=21.0062, zoom=9.5, bearing=0, pitch=30
)


# Render
```