In [1]:

```
import geopandas as gpd
import pandas as pd
```

Part 5

Bicycle sharing system stations - analysis and transfer learning

Task 1

Load dataset with *venturilo* bike stations (`data/veturilo_stations.json`) and convert *lat/lon* into a geometry column. Save it to `stations_gdf` variable

In [2]:
```python
data_path = '../../data/veturilo_stations.json'

stations_gdf = ...

### BEGIN SOLUTION
stations_raw = pd.read_json(data_path)
stations_gdf = gpd.GeoDataFrame(
    stations_raw,
    geometry=gpd.GeoSeries.from_xy(stations_raw["lon"], stations_raw["lat"]),
    crs="EPSG:4326",
)
### END SOLUTION

stations_gdf.head()
```

Out[2]:

	name	lat	lon	geometry
0	Nestle House	52.183992	21.009840	POINT (21.00984 52.18399)
1	UKSW	52.296226	20.958327	POINT (20.95833 52.29623)
2	Metro Młociny	52.290974	20.929556	POINT (20.92956 52.29097)
3	Marymoncka - Dewajtis	52.290173	20.950370	POINT (20.95037 52.29017)
4	Metro Wawrzyszew	52.285914	20.940561	POINT (20.94056 52.28591)

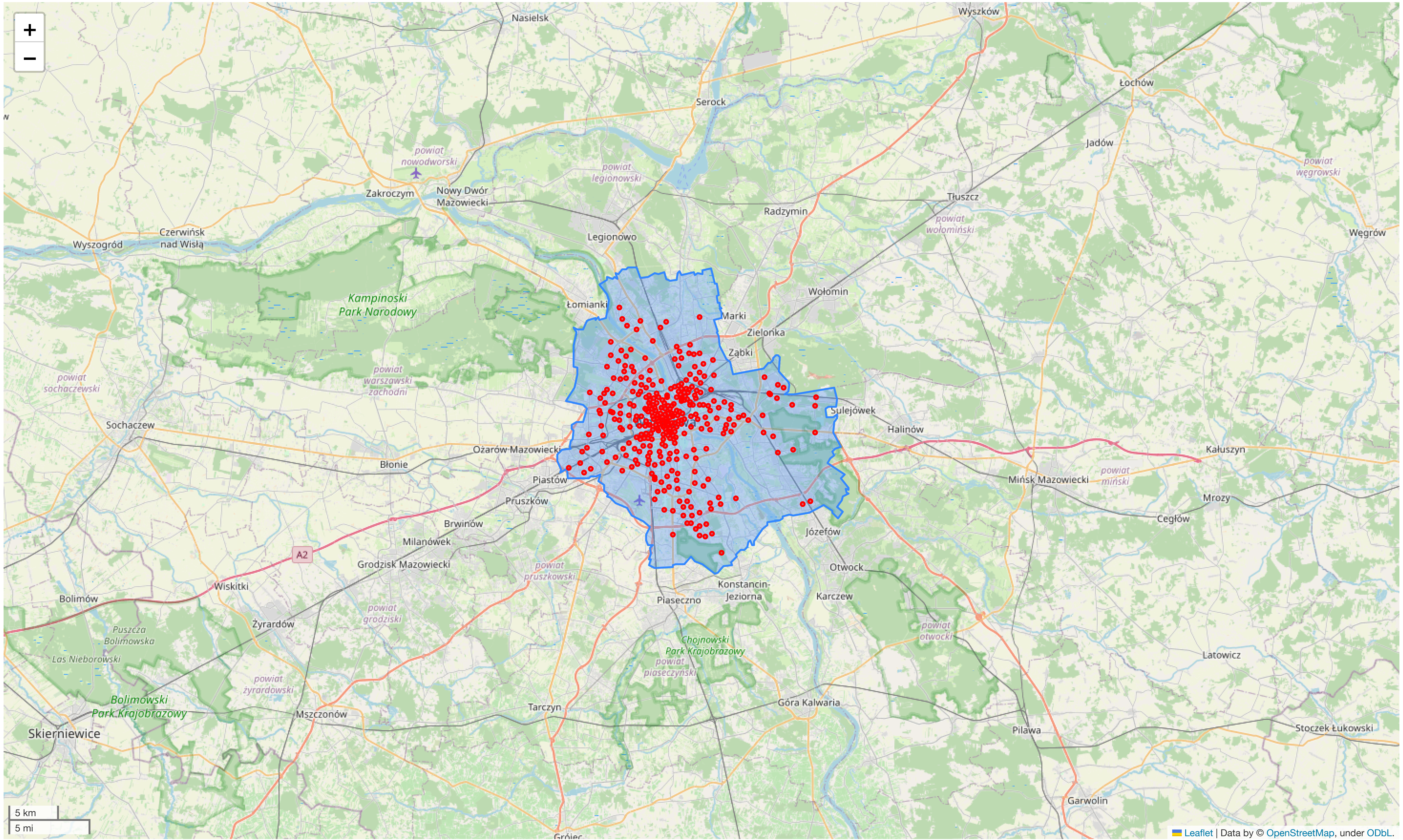
Downloading area of Warsaw in preparation for features download. Visualization of station location on the map

In [12]:

```python
from srai.regionalizers import geocode_to_region_gdf

warsaw_region = geocode_to_region_gdf("Warsaw, PL")
m = warsaw_region.explore(tooltip=False, highlight=False, style_kwds={"fillOpacity": 0.3})
stations_gdf.explore(m=m, color="red")
```

Out[12]:



Task 2

Split the area of Warsaw into regions, for which we will be predicting stations location

In this example we use H3 hierachical index and split the area into hexagons of size 9 (approx 500m in diameter)

In [13]:

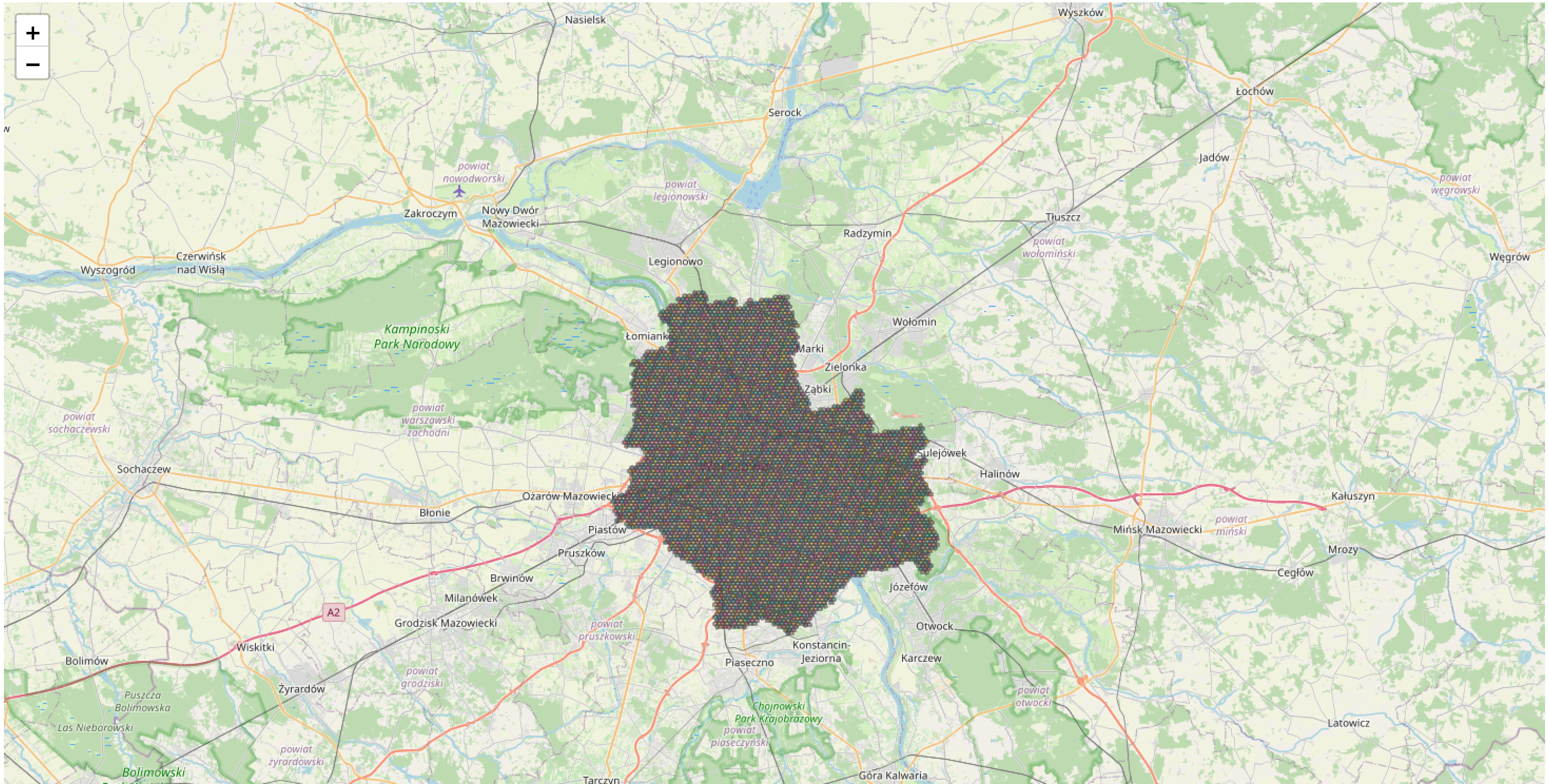
```python
from srai.plotting import plot_regions
from srai.regionalizers import H3Regionalizer

regions_gdf = ...

### BEGIN SOLUTION
regions_gdf = H3Regionalizer(resolution=9).transform(warsaw_region)
### END SOLUTION

plot_regions(regions_gdf)
```

Out[13]:



Task 3

Download the OSM tags which will be used to predict bicycle stations locations. For this case, `OSMPbfLoader` will work the best

We recommend the predefined `GEOFABRIK_LAYERS` filter, since it covers a wide range of different tags. But be honest, remove `{"shopping": "amenity=bicycle_rental"}` tag ;)

In [5]:

```python
from srai.loaders.osm_loaders.filters import GEOFABRIK_LAYERS
from srai.loaders import OSMPbfLoader

features_gdf = ...

### BEGIN SOLUTION
features_gdf = OSMPbfLoader().load(warsaw_region, GEOFABRIK_LAYERS)
features_gdf = features_gdf[features_gdf["shopping"] != "amenity=bicycle_rental"]
### END SOLUTION

features_gdf.head()
```

```
[Warsaw, Masovian Voivodeship, Poland] Counting pbf features: 5249564it [00:07, 666891.69it/s]
[Warsaw, Masovian Voivodeship, Poland] Parsing pbf file #1:  98%|█████| 5142126/5249564 [01:08<00:01, 55259.53it/s]/Users/kacpe
r.lesniara/Projects/Personal/srai-tutorial/venv/lib/python3.10/site-packages/srai/loaders/osm_loaders/pbf_file_handler.py:222:
RuntimeWarning: invalid area (area_id=29859113)
  geometry = self._get_osm_geometry(osm_object, parse_to_wkb_function)
[Warsaw, Masovian Voivodeship, Poland] Parsing pbf file #1: 100%|█████| 5249564/5249564 [01:11<00:00, 72915.50it/s]
Grouping features: 100%|████████████████████████████████████████| 28/28 [00:05<00:00,  4.77it/s]
```

Out[5]:

	geometry	public	education	health	leisure	catering	accommodation	shopping	money	tourism	...	major_roads	minor_roads	highway_links	very_small_roads	paths_ur
feature_id																
node/26063858	POINT (20.99243 52.16657)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
node/26083886	POINT (20.99232 52.17121)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
node/26083913	POINT (21.00105 52.15599)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
node/26083951	POINT (21.00215 52.17502)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
node/26118465	POINT (21.02318 52.15187)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

5 rows × 28 columns

Our features have not been associated with regions yet. We can use an *intersects* predicate and associate them with regions.

In [6]:
```python
from srai.joiners import IntersectionJoiner

joined_features = IntersectionJoiner().transform(regions_gdf, features_gdf)
joined_features
```

Out[6]:

region_id	feature_id
891f53d8a0fffff	relation/6060629
891f53d9d37ffff	relation/6060629
891f53d9d0bffff	relation/6060629
891f53d9dcfffff	relation/6060629
891f53d9dc3ffff	relation/6060629
...	...
891f5352c27ffff	way/1210582036
	way/1206422110
	way/395898580
	way/925919486
	way/1206422112

804534 rows × 0 columns

Task 4

We have already accociated OSM features with regions. To train our model we have to join station locations with regions as well. Write the code which finds regions intersecting with station locations. Use those information to select positive and negative samples for classifier training (regions with and without stations). Remember that we wiil have to train model based on that, so make sure to do any neccessary undersampling to balance our training data

In [7]:
```python
positive_samples = ...
negative_samples = ...

### BEGIN SOLUTION
# First, join bike stations locations with regions, using `IntersectionJoiner`
bikes_joint = IntersectionJoiner().transform(regions_gdf, stations_gdf)

# For future visualizations, we will need to restore geometry column
positive_samples = regions_gdf.join(bikes_joint, how="inner")
positive_samples = positive_samples.reset_index().drop(columns=["feature_id"]).groupby("region_id").agg("first")  # this one is to
positive_samples = positive_samples.reset_index().set_index("region_id")
positive_samples["is_positive"] = True

# Mark remaining regions as negative
negative_samples = regions_gdf.copy()
negative_samples["is_positive"] = False
negative_samples.loc[positive_samples.index, "is_positive"] = True
negative_samples = negative_samples[~negative_samples["is_positive"]]

# Just to keep everything balanced - undersampling
negative_samples = negative_samples.sample(n=3 * len(positive_samples), random_state=42)
### END SOLUTION


train_data = pd.concat([positive_samples, negative_samples])
train_data.explore("is_positive", cmap="cividis", zoom_start=13, tiles="CartoDB positron")
```

```
/Users/kacper.lesniara/Projects/Personal/srai-tutorial/venv/lib/python3.10/site-packages/geopandas/array.py:1486: UserWarning:
CRS not set for some of the concatenation inputs. Setting output's CRS as WGS 84 (the single non-null crs provided).
  warnings.warn(
```

Out[7]:

Let's create embeddings for each region in our city (embeddings for outside of training data will be used for visualizations). Those will serve as our *Xs* for training, and *Ys* will be binary value if station is in the area or not

In [8]:
```
from srai.embedders import ContextualCountEmbedder
from srai.neighbourhoods import H3Neighbourhood

embedder = ContextualCountEmbedder(
    neighbourhood=H3Neighbourhood(),
    neighbourhood_distance=5,
    concatenate_vectors=True,
    expected_output_features=GEOFABRIK_LAYERS,
)
embeddings = embedder.transform(
    regions_gdf=regions_gdf, features_gdf=features_gdf, joint_gdf=joined_features
)
X = embeddings.loc[train_data.index].to_numpy()
Y = train_data["is_positive"].astype(int).to_numpy()
```

Task 5

Select your favourite model and train a classifier for station locations

In [9]:
```
from sklearn.metrics import classification_report

### BEGIN SOLUTION
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=42)


classifier = SVC(probability=True)
classifier.fit(X_train, Y_train)
Y_pred = classifier.predict(X_test)
Y_pred_proba = classifier.predict_proba(X_test)
### END SOLUTION


print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       189
           1       0.72      0.65      0.68        63

    accuracy                           0.85       252
   macro avg       0.80      0.78      0.79       252
weighted avg       0.85      0.85      0.85       252
```

Task 6

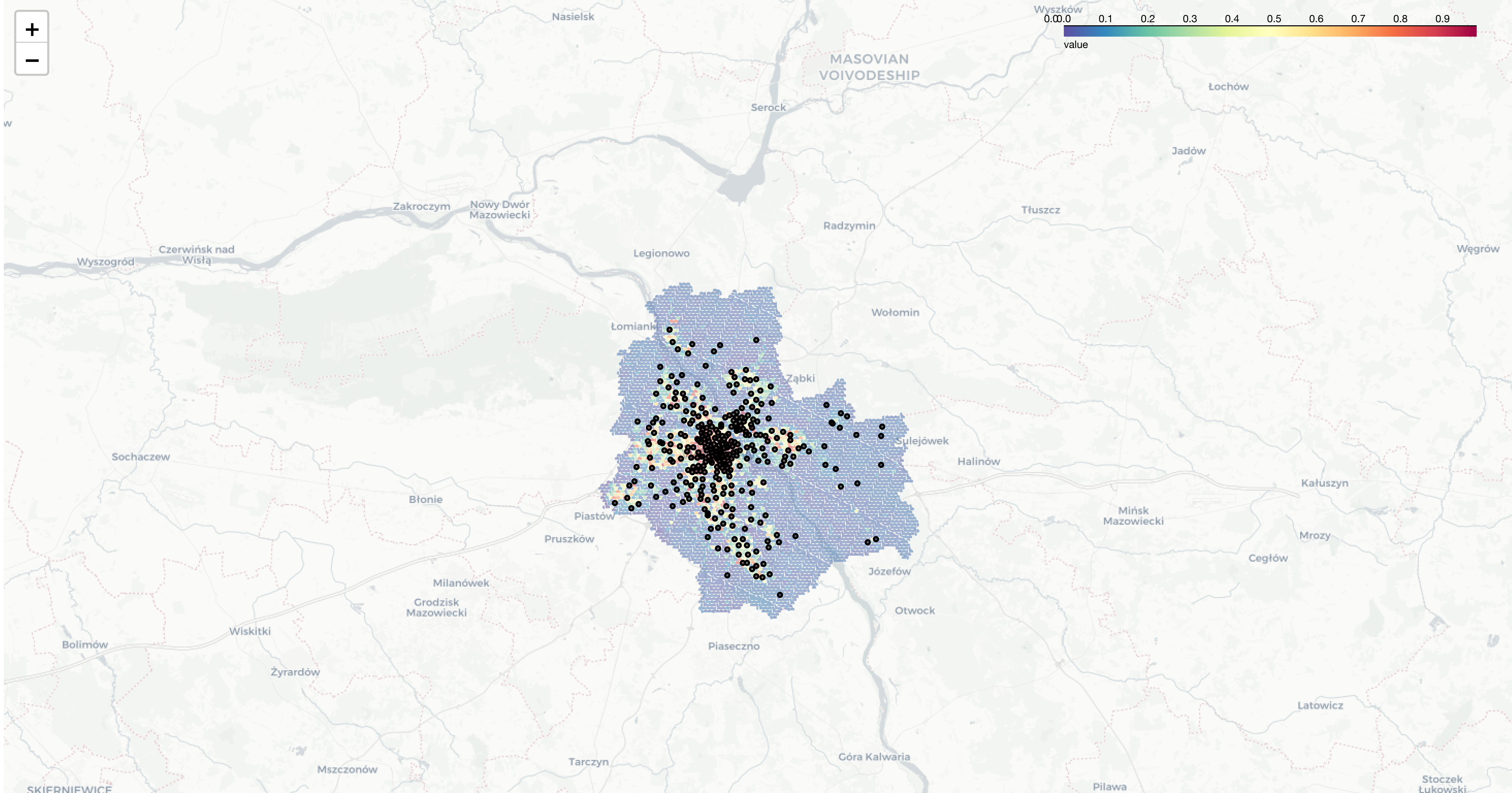
Run predictions for all regions and prepare visualization on the map

In [14]:

```python
from srai.plotting import plot_numeric_data

### BEGIN SOLUTION
station_probas = classifier.predict_proba(embeddings.to_numpy())
regions_gdf["station_proba"] = station_probas[:, 1]
m = plot_numeric_data(regions_gdf, "station_proba", colormap="Spectral_r", opacity=0.5)
stations_gdf.explore(m=m, color='black')
### END SOLUTION
```

Out[14]:



Final task - transfer learning

Now we have a model, which was trained on data from Warsaw. Select some other city, and run predictions on it. Let's see where to put BSS stations there

In [11]:
```python
### BEGIN SOLUTION

# Select area
wroclaw_region = geocode_to_region_gdf('Wrocław, PL')

# Split into regions
wroclaw_regions_gdf = H3Regionalizer(resolution=9).transform(wroclaw_region)

# Load OSM features (the same as for model training). We will also save stations location for visualization later
wroclaw_features_gdf = OSMPbfLoader().load(wroclaw_region,GEOFABRIK_LAYERS)
wroclaw_stations = wroclaw_features_gdf[wroclaw_features_gdf["shopping"] == "amenity=bicycle_rental"]
wroclaw_features_gdf = wroclaw_features_gdf[wroclaw_features_gdf["shopping"] != "amenity=bicycle_rental"]

# Get embeddings for regions
wroclaw_joined_features = IntersectionJoiner().transform(wroclaw_regions_gdf, wroclaw_features_gdf)
wroclaw_embeddings = embedder.transform(
    regions_gdf=wroclaw_regions_gdf,
    features_gdf=wroclaw_features_gdf,
    joint_gdf=wroclaw_joined_features,
)

# Predict and visualize
station_probas_wro = classifier.predict_proba(wroclaw_embeddings.to_numpy())

wroclaw_regions_gdf["station_proba"] = station_probas_wro[:, 1]
m = plot_numeric_data(wroclaw_regions_gdf, "station_proba", colormap="Spectral_r", opacity=0.5)

wroclaw_stations.explore(m=m, color='black')

### END SOLUTION
```

```
[Wrocław, Lower Silesian Voivodeship, Poland] Downloading pbf file #1 (Elements): 100%|█| 4950458/4950458 [00:09<00
52376ab5c09711b6057db9c1f77abbeb59b13c20e4ac1f0b14b427d387aee6ac.osm.pbf: 100%|█| 25.7M/25.7M [00:03<00:00, 6.87MiB
[Wrocław, Lower Silesian Voivodeship, Poland] Counting pbf features: 2891589it [00:04, 646022.52it/s]
[Wrocław, Lower Silesian Voivodeship, Poland] Parsing pbf file #1: 100%|█| 2891589/2891589 [00:39<00:00, 72532.96it
Grouping features: 100%|████████████████████████████████████████████████| 28/28 [00:02<00:00,  9.56it/s]
```

Out[11]: