Machine Learning Example

Predictive model and transfer learning

Transfer learning with bicycle rental stations

In this part we will see:

- How to use a pre-trained hex2vec model with `srai`
- How to train classification model based on `srai` embeddings
- How to use `srai` to gather training data

In [3]:
```python
# srai components used in this lesson
from srai.loaders import OSMOnlineLoader, OSMPbfLoader
from srai.regionalizers import geocode_to_region_gdf
from srai.joiners import IntersectionJoiner
from srai.embedders import Hex2VecEmbedder
from srai.regionalizers import H3Regionalizer

# classification model using scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# plotting utilities
from srai.plotting import plot_numeric_data
import plotly.express as px

from utils import CB_SAFE_PALLETE
import pandas as pd
```

Experiment description

In [4]:
```
SOURCE_CITY = "Wrocław, Poland"
TARGET_CITY = "Basel, Switzerland"
H3_RESOLUTION = 10

bike_stations_osm_tag = {"amenity": "bicycle_rental"}
```

In [5]:
```
source_area = geocode_to_region_gdf(SOURCE_CITY)
target_area = geocode_to_region_gdf(TARGET_CITY)
```

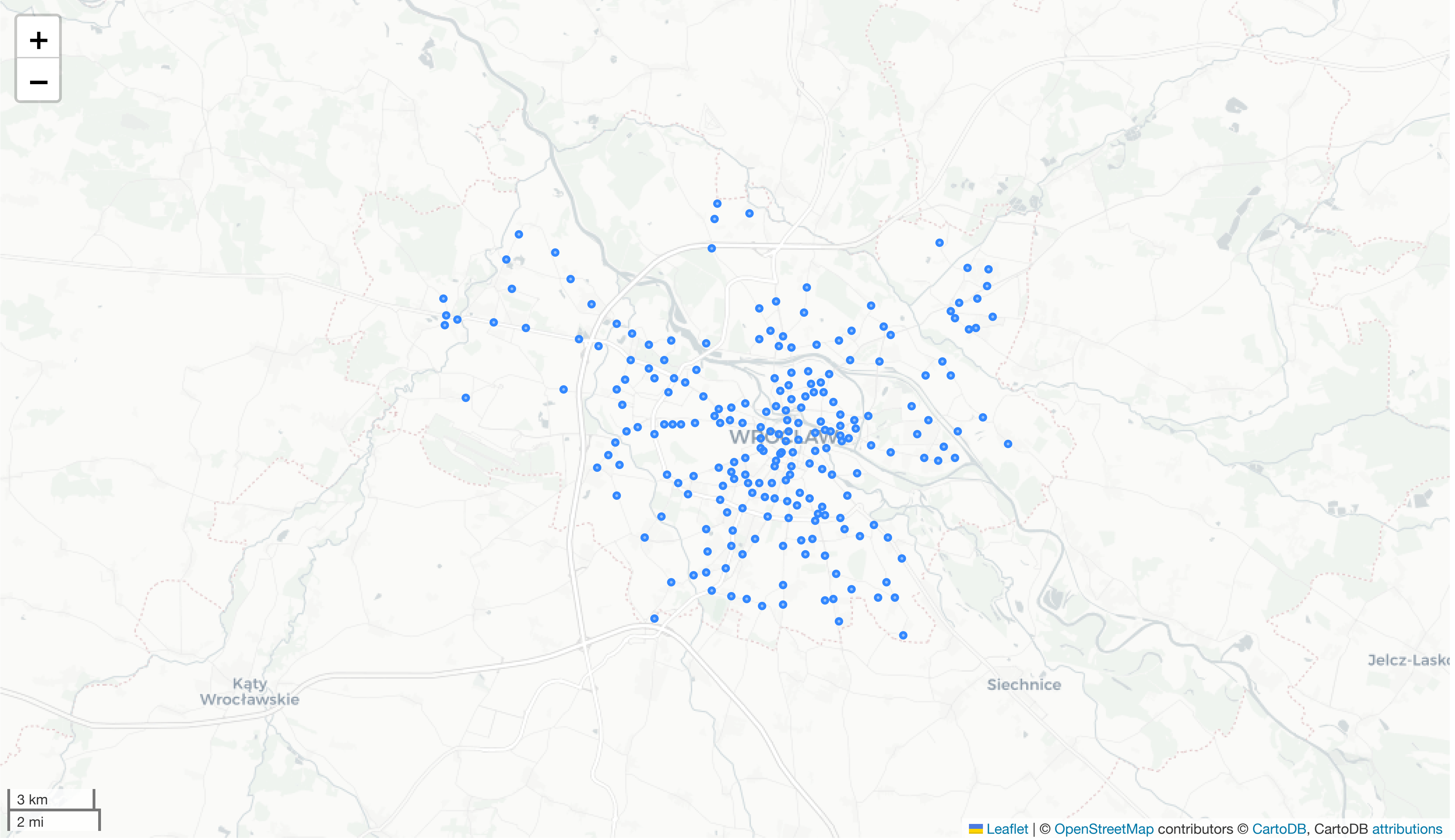
Downloading bike rental stations

In [6]:

```
loader = OSMOnlineLoader()
stations = loader.load(source_area, bike_stations_osm_tag)
stations.explore(height=600, tiles="CartoDB positron")
```

```
Downloading amenity: bicycle_rental: 100%|██████████████████████████████| 1/1 [00:00<00:00,  8.42it/s]
```

Out[6]:



Load pre-trained embedding model

We use pre-trained hex2vec model. This one was trained by us on all polish cities with 50k+ inhabitants. Models are available for download, link in our repo. For this tutorial, model for resoulution 10 is already downloaded and placed in `models` directory.

In [7]:
```
embedder = Hex2VecEmbedder.load(f"models/hex2vec_{H3_RESOLUTION}_poland_50k")
embedder.expected_output_features
```

Out[7]:
```
0            aeroway_aerodrome
1                aeroway_apron
2                 aeroway_gate
3               aeroway_hangar
4              aeroway_helipad
                   ...
720     waterway_tidal_channel
721     waterway_turning_point
722       waterway_water_point
723         waterway_waterfall
724               waterway_weir
Length: 725, dtype: object
```

We need to translate those features to OSM tags (+ remove bicycle rental stations)

In [8]:
```
embedder_osm_tags = {}

for element in embedder.expected_output_features:
    if element == 'amenity_bicycle_rental':
        continue
    key, value = element.split('_', 1)
    if key not in embedder_osm_tags:
        embedder_osm_tags[key] = [value]
    else:
        embedder_osm_tags[key].append(value)
```

Load features from OSM and prepare regions

We need to load features to calculate embeddings for our cities. We will use OSMPbfLoader this time, since it is faster than OSMOnlineLoader when we have a lot of tags to download.

In [9]:
```
# load features
train_features = OSMPbfLoader().load(source_area, embedder_osm_tags)
# split into regions
train_regions = H3Regionalizer(resolution=H3_RESOLUTION).transform(source_area)
# join regions and features
train_joint = IntersectionJoiner().transform(train_regions, train_features)
# calculate embeddings
train_embeddings = embedder.transform(train_regions, train_features, train_joint)
train_embeddings.head(5)
```

```
[Wrocław, Lower Silesian Voivodeship, Poland] Counting pbf features: 2878850it [00:04, 693507.46it/s]
[Wrocław, Lower Silesian Voivodeship, Poland] Parsing pbf file #1: 100%|██████████| 2878850/2878850 [00:26<00:00, 110372.35it/s]
```

Out[9]:

region_id	0	1	2	3	4	5	6	7	8	9
8a1e2042e3b7fff	0.171020	-0.274647	-0.405161	-0.492315	-0.602791	-0.531773	-0.014250	-0.129288	-0.140136	-0.159902
8a1e20429327fff	-0.348374	-0.200974	-0.246354	-0.091986	0.130055	0.263121	0.038625	-0.454178	-0.161510	0.286223
8a1e2040318ffff	0.353291	0.226918	-0.138446	-0.139644	0.004650	0.184755	0.138181	-0.690888	0.603068	-0.125255
8a1e20456357fff	0.017708	-0.332491	0.278416	-0.179365	0.118547	-0.118545	-0.061888	0.028283	-0.284859	0.135998
8a1e2041982ffff	0.019867	-0.035198	0.306941	0.201169	0.235859	-0.156453	-0.160706	-0.074845	0.163790	-0.487590

5 rows × 64 columns

Assign bike rental stations to regions to create training data for machine learning

In [10]:
```
bikes_joint = IntersectionJoiner().transform(train_regions, stations)
```

Select regions with stations as positive samples

In [11]:
```
positive_samples = train_regions.join(bikes_joint, how="inner")
positive_samples = positive_samples.reset_index().drop(columns=["feature_id"]).set_index("reg
positive_samples["is_positive"] = True
len(positive_samples)
```

Out[11]: 223

Select regions with stations as positive samples

In [11]:
```
positive_samples = train_regions.join(bikes_joint, how="inner")
positive_samples = positive_samples.reset_index().drop(columns=["feature_id"]).set_index("reg
positive_samples["is_positive"] = True
len(positive_samples)
```

Out[11]: 223

Now remaining regions are negative samples

In [12]:
```
negative_samples = train_regions.copy()
negative_samples["is_positive"] = False
negative_samples.loc[positive_samples.index, "is_positive"] = True
negative_samples = negative_samples[~negative_samples["is_positive"]]
len(negative_samples)
```

Out[12]: 21114

This is very imbalanced! Let's undersample to make it possible to train model

In [13]:
```
negative_undersampled = negative_samples.sample(n=3 * len(positive_samples), random_state=42)
negative_undersampled
```

Out[13]:

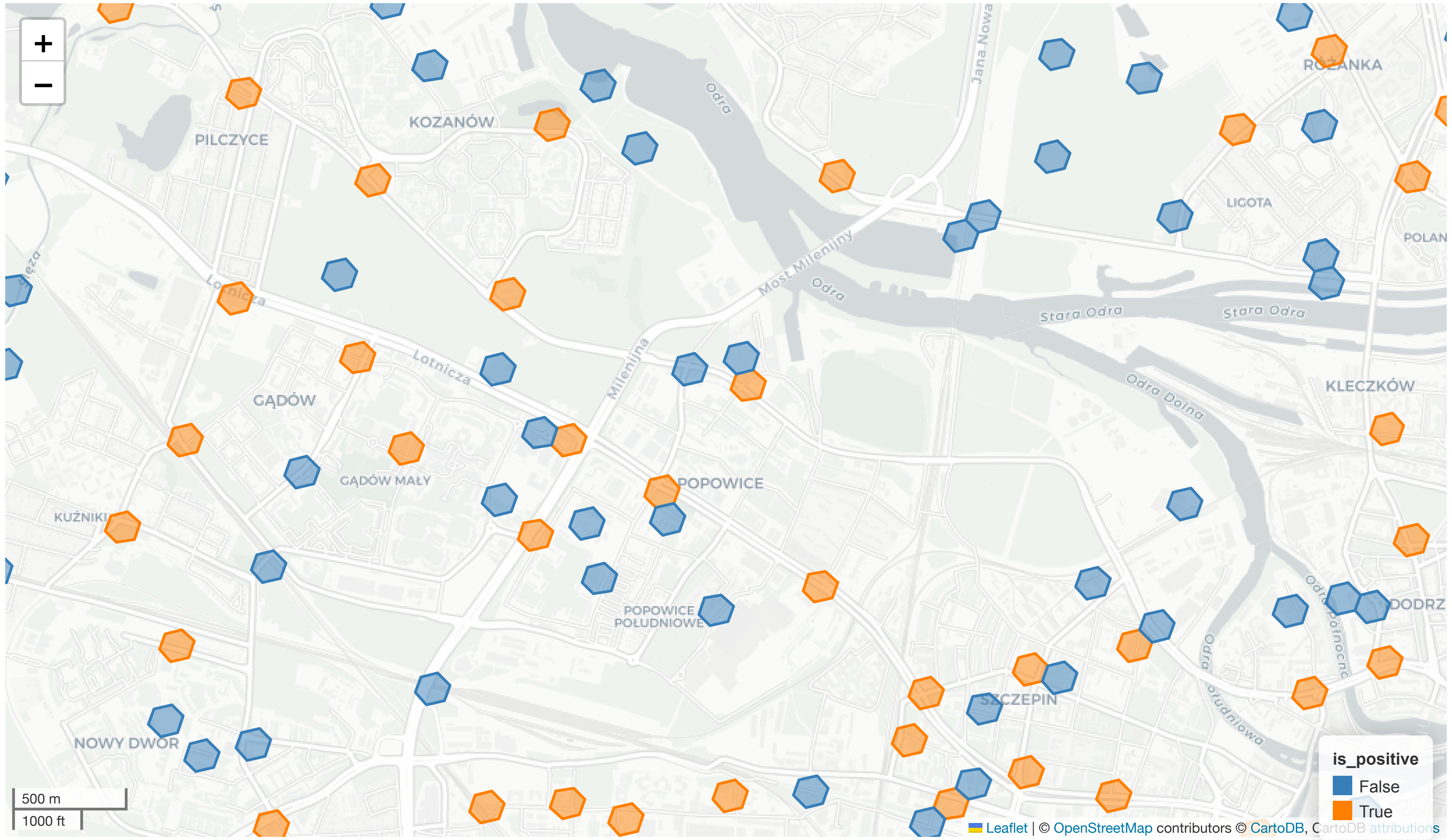
region_id	geometry	is_positive
8a1e20409cb7fff	POLYGON ((17.04880 51.10133, 17.04915 51.10194...	False
8a1e2040e307fff	POLYGON ((17.00582 51.10029, 17.00509 51.10075...	False
8a1e2040441ffff	POLYGON ((16.91346 51.12706, 16.91312 51.12645...	False
8a1e2051b18ffff	POLYGON ((16.85053 51.16584, 16.84980 51.16630...	False
8a1e2043536ffff	POLYGON ((16.88638 51.15074, 16.88604 51.15013...	False
...	...	...
8a1e20430657fff	POLYGON ((16.85946 51.12215, 16.86053 51.12230...	False
8a1e2051aa9ffff	POLYGON ((16.82468 51.16719, 16.82433 51.16658...	False
8a1e2040c42ffff	POLYGON ((17.00231 51.11329, 17.00303 51.11283...	False
8a1e2042574ffff	POLYGON ((16.94808 51.18146, 16.94736 51.18192...	False
8a1e20470107fff	POLYGON ((17.09567 51.14057, 17.09640 51.14011...	False

669 rows × 2 columns

We can see training data on the map

In [14]:
```
train_data = pd.concat([positive_samples, negative_undersampled])
train_data.explore("is_positive", cmap=CB_SAFE_PALLETE, zoom_start=14, height=600, tiles="Car
```

Out[14]:



Train classifier

In [15]:
```
X = train_embeddings.loc[train_data.index].to_numpy()
y = train_data["is_positive"].astype(int).to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, str
```

In [16]:
```
classifier = SVC(probability=True)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
y_pred_proba = classifier.predict_proba(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87       134
           1       0.62      0.53      0.57        45

    accuracy                           0.80       179
   macro avg       0.73      0.71      0.72       179
weighted avg       0.79      0.80      0.79       179
```

Transfer knowledge to Basel

Let's repeat embedding for target city

In [17]:
```
target_regions = H3Regionalizer(resolution=10).transform(target_area)
target_features = OSMPbfLoader().load(target_area, embedder_osm_tags)
target_joint = IntersectionJoiner().transform(target_regions, target_features)
target_embeddings = embedder.transform(target_regions, target_features, target_joint)
```

```
[Basel, Basel-City, Switzerland] Counting pbf features: 636246it [00:01, 625397.32it/s]
[Basel, Basel-City, Switzerland] Parsing pbf file #1: 100%|██████████| 636246/6
36246 [00:06<00:00, 103097.30it/s]
```

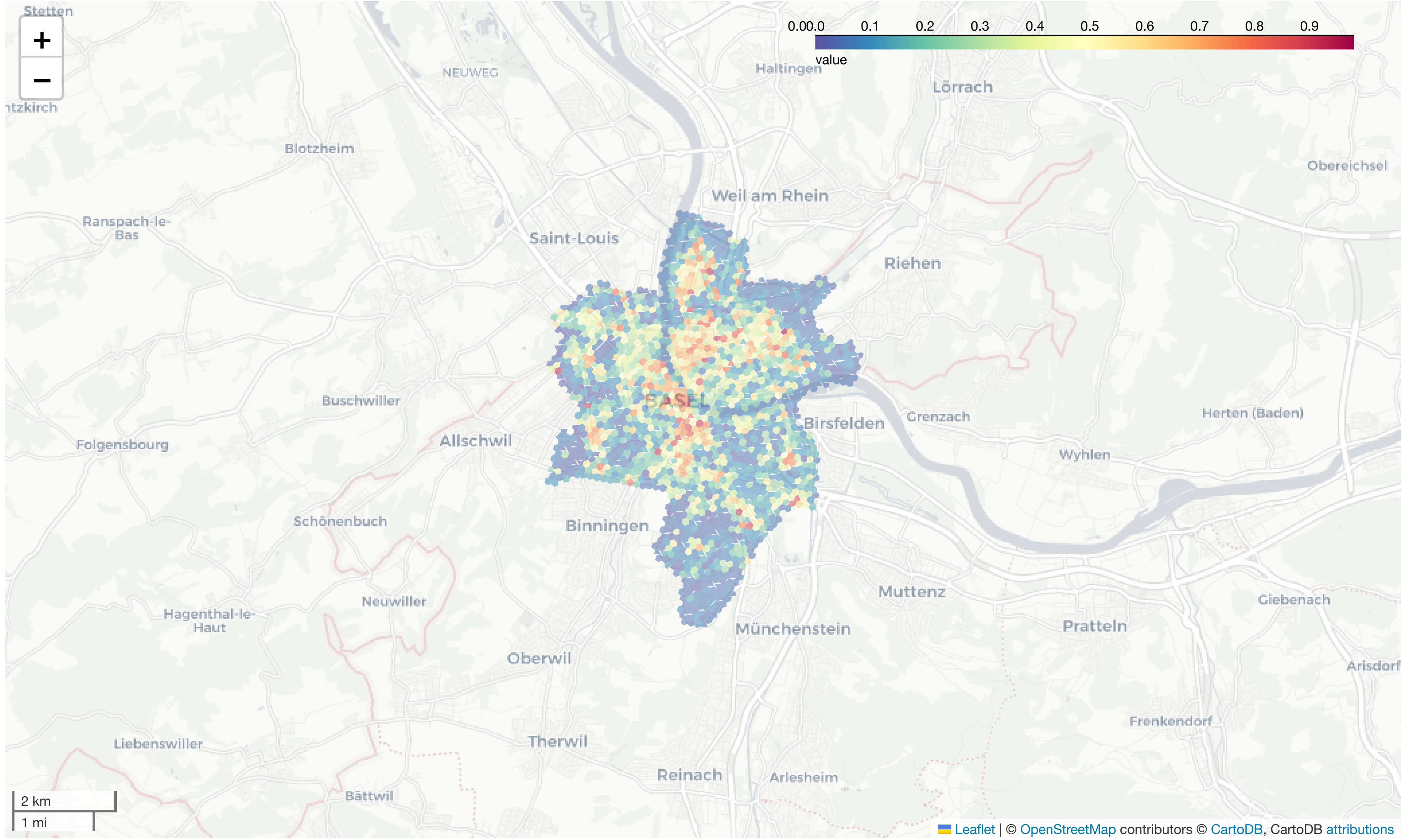
And now find regions with high score for station location

In [18]:

```
station_probas = classifier.predict_proba(target_embeddings.to_numpy())

target_regions["station_proba"] = station_probas[:, 1]
plot_numeric_data(target_regions, "station_proba", colormap="Spectral_r", opacity=0.5)
```

Out[18]:



Way better results

Kamil's past project took this task more seriously. He used larger selection of cities and obtained great results. See them here:

https://t.ly/cgQPA