

Gdzie pojechać w Polsce
Lego foodtruckiem,
żeby zarobić dużo siana?

Krzysztof Stawarz

Lena Stec

Informatyka Społeczna WH AGH

Modelowanie w Data Mining

Kraków, styczeń 2024



Konceptualizacja

Trzeba jakoś ogarnąć gdzie jechać

- potrzebne nam będą dane
 - skąd je wziąć?
 - jakie dane mogą mieć znaczenie?
- trzeba napisać coś w pythonie
 - jak przedstawić dane geograficzne?
 - robić w linii prostej/po drogach?
 - robić prosto i optymalnie czy skomplikowanie i do dupy?

PREZKA U MIREGO



WYKŁAD U PIETRONIA



KIEDY UŚWIADAMIASZ
SOBIE ŻE INFORMATYKA SPOŁECZNA
NIE UMIE ODPALIĆ JUPYTERA



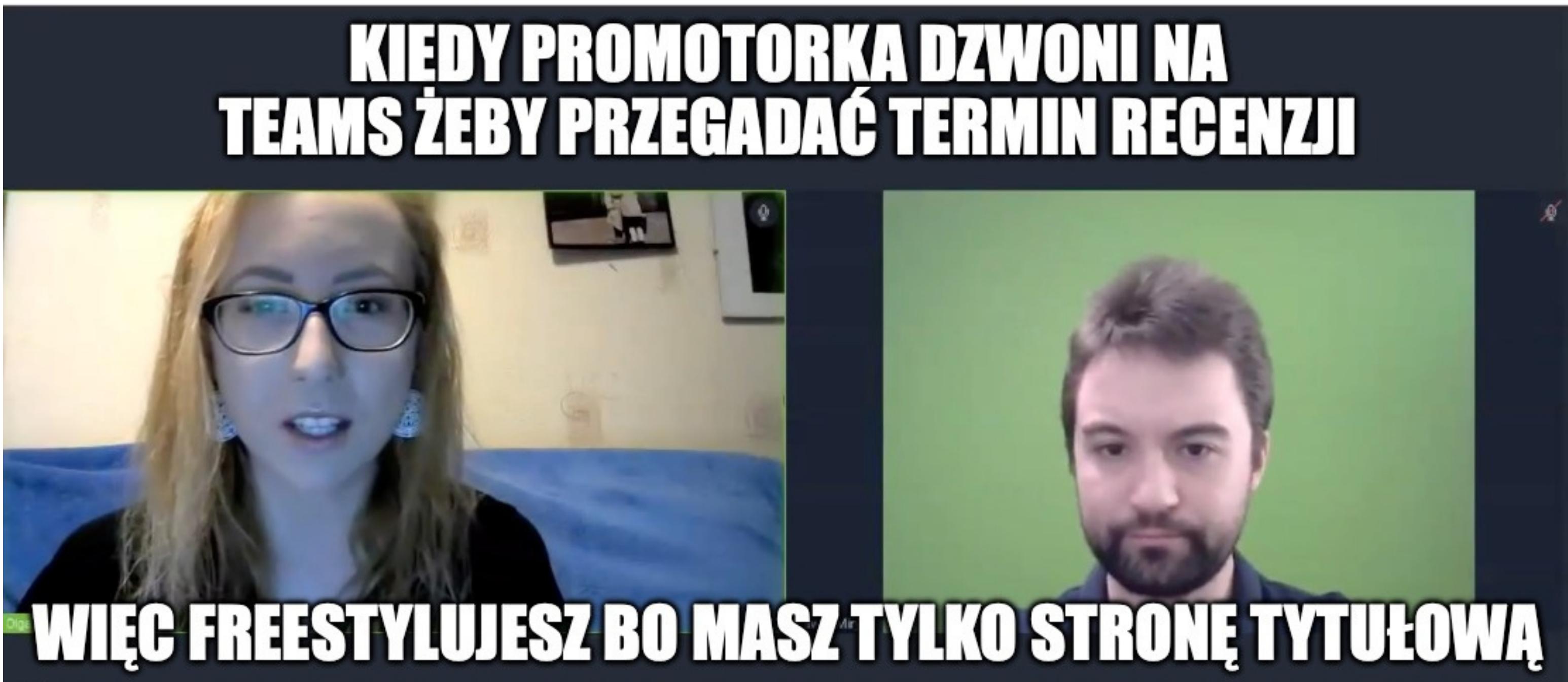
Dane

- Miasta Polski
 - populacja
 - współrzędne geograficzne
 - czy ma lego shop (albo ile idk)
 - ~~jakies eventy lego-oriented?~~

Tyle będzie git

Skąd ukradliśmy dane?

Oglądając jakiegoś gracza Geoguesser'a na yt natknęliśmy się na OP stronkę - [Overpass-turbo](#). Pisze się skrypt w ich języku i można querować mega dokładne dane o jakiś miejscach (np. najbliższa fontanna przy szkole specjalnej w północnej macedonii)





Run

Share

Export

Wizard

Save

Load

Settings

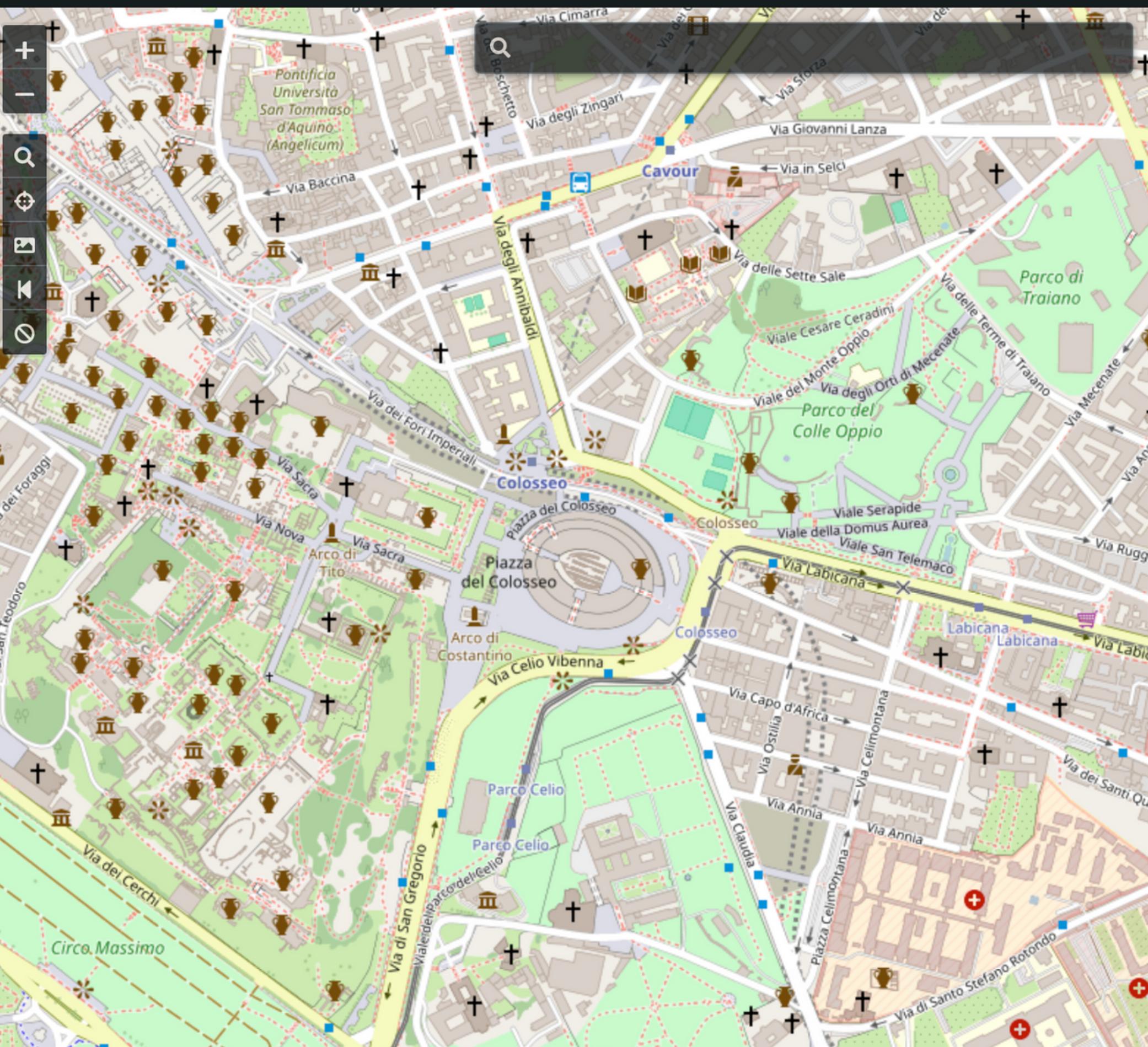
Help

overpass turbo

Map

Data

```
1 /*  
2 This is an example Overpass query.  
3 Try it out by pressing the Run button above!  
4 You can find more examples with the Load tool.  
5 */  
6 node  
7   [amenity=drinking_water]  
8   ({{bbox}});  
9 out;
```





Run

Share

Export

Wizard

Save

Load

Settings

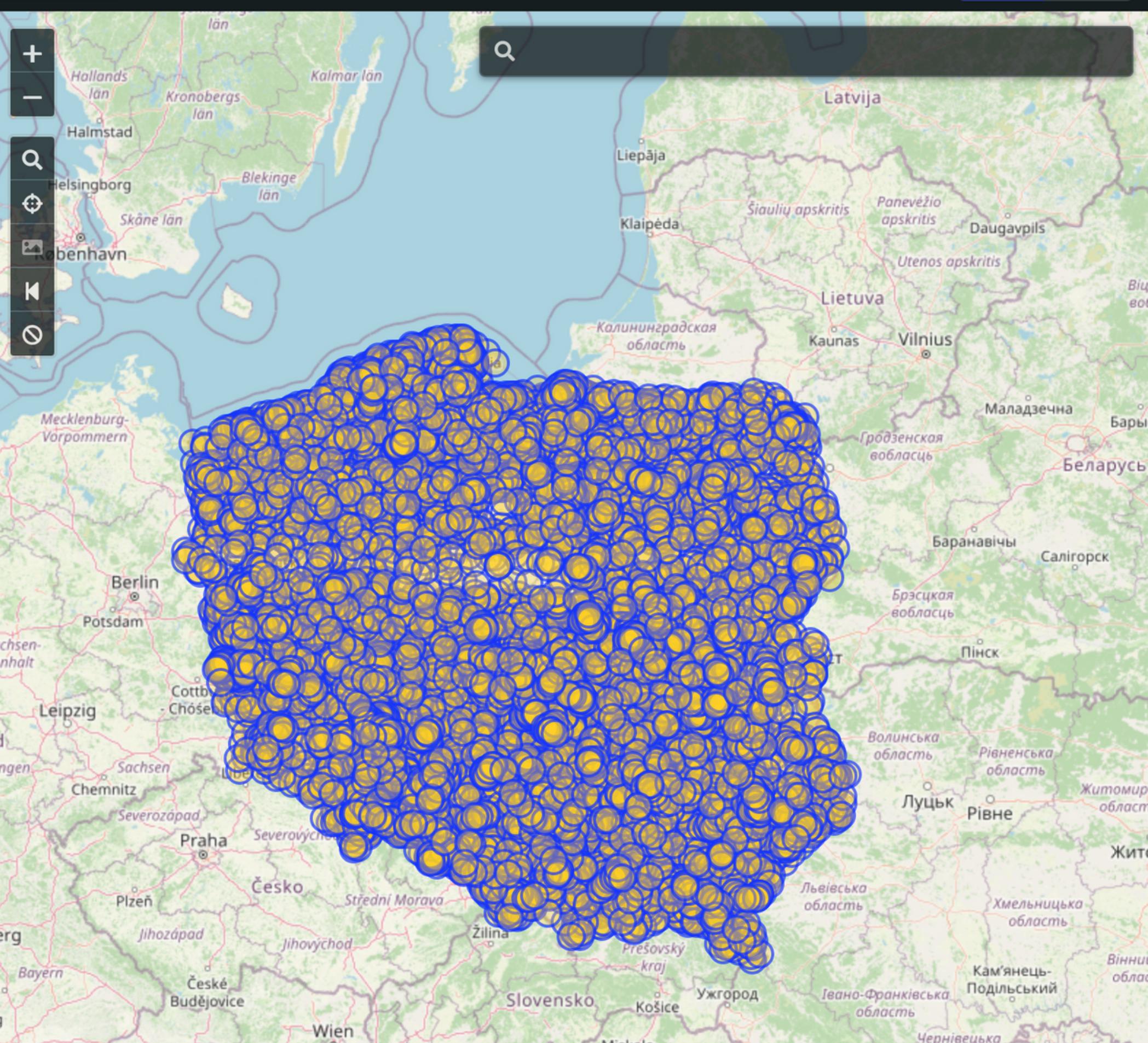
Help

overpass turbo

Map

Data

```
1 [out:json][timeout:250];
2 area["IS03166-1"="PL"]->.searchArea;
3 (
4   node["place"="city"]["population"](.searchArea);
5   node["place"="town"]["population"](.searchArea);
6   node["place"="village"]["population"](.searchArea);
7 );
8 out body;
9 >;
10 out skel qt;
```





```
1 import overpy
2 import csv
3
4 api = overpy.Overpass()
5
6 query = """
7 [out:json];
8 (
9 area["ISO3166-1"]="PL"]->.searchArea;
10 node["place"]=="city"]["population"](area.searchArea);
11 node["place"]=="town"]["population"](area.searchArea);
12 node["place"]=="village"]["population"](area.searchArea);
13 );
14 out body;
15 """
16
17 result = api.query(query)
18
19
20 with open('data/population.csv', 'w', newline='', encoding='utf-8') as f:
21     writer = csv.writer(f)
22     writer.writerow(["ID", "name", "population", "latitude", "longitude"])
23
24     for i, node in enumerate(result.nodes):
25         name = node.tags.get('name', 'n/a')
26         population = node.tags.get('population', 'n/a')
27         lat, lon = node.lat, node.lon
28         writer.writerow([i, name, population, lat, lon])
29
30
```

ID	Name	Population	Latitude	Longitude
0	Skarszewy	7082	54.07	18.45
1	Nowa Karczma	792	54.13	18.2
2	Lubań	1000	54.12	18.16
3	Będomin	150	54.12	18.12
4	Małych Klincz	308	54.13	18.06
5	Nowy Klincz	586	54.12	18.03
6	Kościerzyna	23701	54.12	17.98
7	Kościarska Hu	308	54.14	18.02
8	Kaliska Koście	322	54.16	18.04
9	Kłobuczyno	490	54.19	18.1
10	Grabowska Hu	74	54.18	18.17
11	Egiertowo	600	54.24	18.2
12	Kartuzy	14186	54.33	18.2
13	Hopowo	346	54.26	18.24
14	Borcz	510	54.27	18.29
15	Małkowo	390	54.37	18.33
16	Miszewo	290	54.39	18.36
17	Rębiechowo	463	54.39	18.43
18	Starogard Gdańsk	48621	53.97	18.53
19	Gdańsk	486345	54.35	18.65
20	Gdynia	244104	54.52	18.54
21	Chojnice	39484	53.7	17.56
22	Krosno Odrzańskie	11983	52.05	15.1
23	Słubice	16908	52.36	14.57
24	Kamierowo	221	54.09	18.46

```
population = pd.read_csv("data/population.csv", index_col=0)
population.head()
```

ID		name	population	latitude	longitude
0		Skarszewy	7082	54.071752	18.445995
1		Nowa Karczma	792	54.133217	18.203053
2		Lubań	1000	54.118653	18.159010
3		Będomin	150	54.123342	18.123258
4		Małych Klincz	308	54.125140	18.056115

```
population.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 11524 entries, 0 to 11523
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          -----          ----  
 0   name        11524 non-null   object 
 1   population   11524 non-null   object 
 2   latitude     11524 non-null   float64
 3   longitude    11524 non-null   float64
dtypes: float64(2), object(2)
memory usage: 450.2+ KB
```

```
for i in range(population.shape[0]):
    row = population.iloc[i, :]
    try:
        int(row["population"])
    except:
        print("-"*20)
        print(i)
        print(row["population"])
    -----
```

3210
1616 (2011 r.)

```
-----
```

4102
ok. 250

```
-----
```

9315
12 266

```
-----
```

10411
blisko ponad 150 osób

```
-----
```

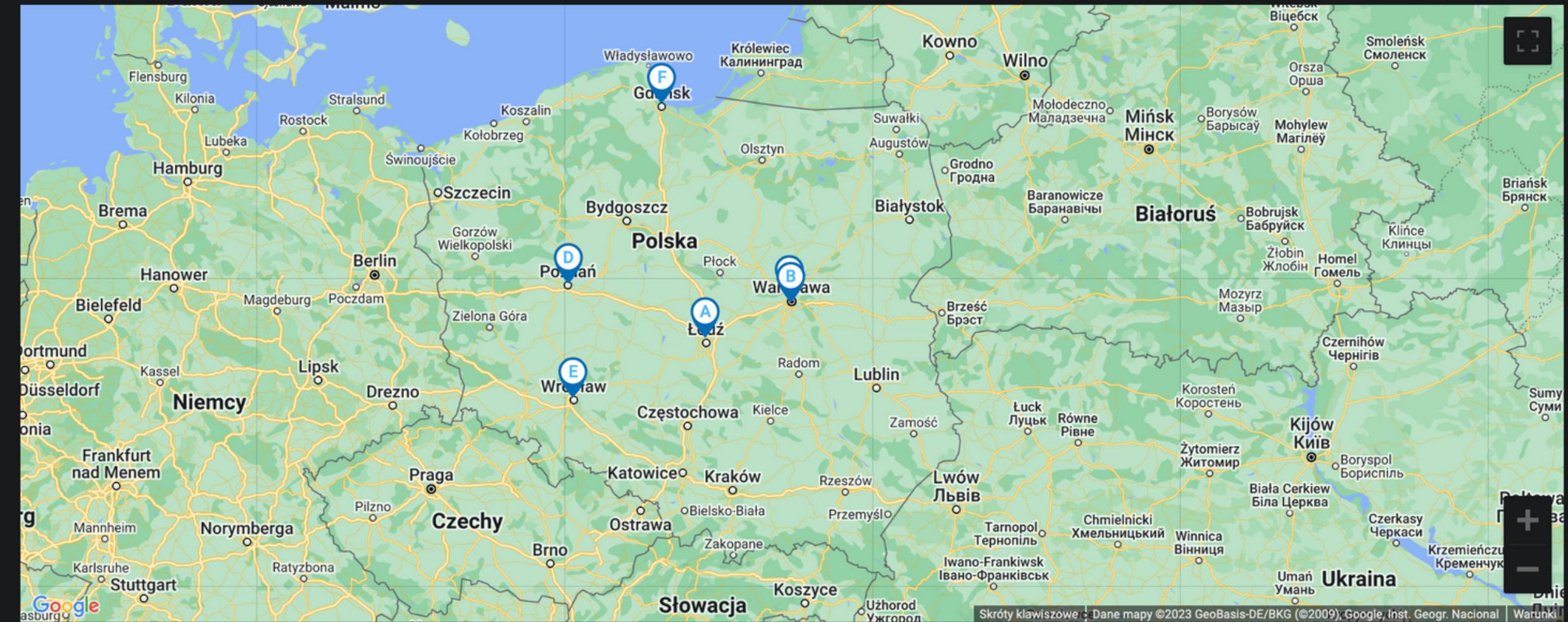
population.iloc[3210, 1] = 1616
population.iloc[4102, 1] = 250
population.iloc[9315, 1] = 12_266
population.iloc[10411, 1] = 150

	name	population	latitude	longitude
ID				
9844	Warszawa	1792718	52.231956	21.006725
10787	Kraków	762508	50.061947	19.936857
41	Łódź	711332	51.768734	19.456991
9767	Wrocław	634487	51.108978	17.032669
28	Poznań	548028	52.408268	16.933519
19	Gdańsk	486345	54.348289	18.654024
26	Szczecin	402465	53.430183	14.550962
37	Lublin	340466	51.250561	22.570103
75	Bydgoszcz	330038	53.121964	18.000254
8804	Białystok	296401	53.132397	23.159168
8203	Katowice	280190	50.259899	19.021585
20	Gdynia	244104	54.516499	18.540274
8265	Częstochowa	232318	50.812046	19.113213
6128	Radom	218466	51.402256	21.154154
8775	Rzeszów	197863	50.037453	22.004717
8844	Sosnowiec	197586	50.278084	19.134295
10668	Toruń	196935	53.010273	18.604809
11123	Kielce	187374	50.871990	20.631048
32	Gliwice	185450	50.294113	18.665731
8367	Zabrze	178357	50.308617	18.786375
8764	Olsztyn	174675	53.776684	20.476507
8881	Bielsko-Biała	173699	49.822117	19.044893
8841	Bytom	173439	50.347038	18.923185
27	Zielona Góra	140403	51.938377	15.505041
8882	Rybnik	140173	50.095581	18.541994
8511	Ruda Śląska	137030	50.285801	18.874794
10687	Tychy	128812	50.114395	18.996593
2080	Elbląg	126460	54.155872	19.404459
9286	Gorzów Wielkopolski	124344	52.730991	15.240046
10642	Dąbrowa Górnica	123994	50.330868	19.207891

Liczba wyników wyszukiwania terminu „polska”: 6

W Polsce mamy
6 sklepów Lego:

- Łódź
- Wrocław
- Gdańsk
- Poznań
- 2x Warszawa



A LEGO® Store Łódź

Drewnowska 58
Łódź
91- 002

Phone: +48 42 239 90 15

[Zobacz dane sklepu >](#)

B LEGO® Store Mokotów

Wołoska 12 (Unit no. A0135A)
Warszawa
02-675

Phone: +48 224624048

[Zobacz dane sklepu >](#)

C LEGO® Store Warsaw

Westfield Arkadia, John Paul II Avenue 82
Warsaw
00-175

Phone: +48 2241 73310

[Zobacz dane sklepu >](#)

D LEGO® Store Poznań

Posnania Mall, Posnania, ul. Pleszewska 1,
unit: B 382
Poznań
61-136
Phone: +48616618883

E LEGO® Store Wroclavia

Wroclavia shopping centre, Wroclavia,
Sucha 1
Wrocław
Phone: 0048 71 75717 31

[Zobacz dane sklepu >](#)

F LEGO® Store Gdańsk

Targ Sienny 7
Gdańsk
80-806
Phone: (+48) 587731600

[Zobacz dane sklepu >](#)

```
df["how_many_legoshops"] = df["name"].apply(lambda x: 2 if x=="Warszawa" else 1 if x in ["Łódź", "Wrocław", "Poznań", "Gdańsk"] else 0)
```

✓ 0.0s

```
df = df.sort_values("population", ascending=False)  
df.head()
```

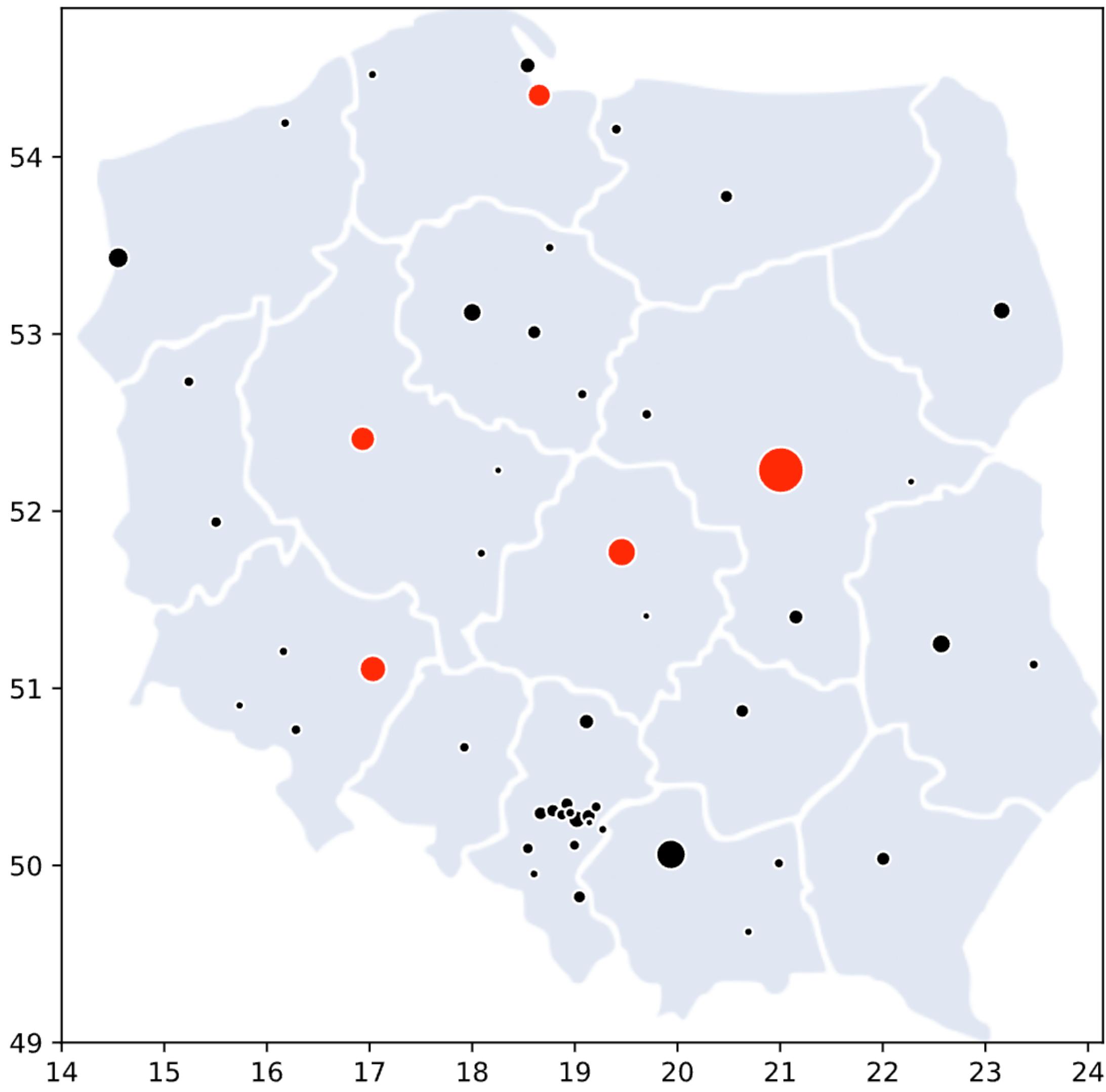
✓ 0.0s

ID	name	population	latitude	longitude	how_many_legoshops
9844	Warszawa	1792718	52.231956	21.006725	2
10787	Kraków	762508	50.061947	19.936857	0
41	Łódź	711332	51.768734	19.456991	1
9767	Wrocław	634487	51.108978	17.032669	1
28	Poznań	548028	52.408268	16.933519	1

```
MAX_N, MAX_S, MAX_W, MAX_E = 54.8399, 49.0025, 14.1226, 24.1455
```

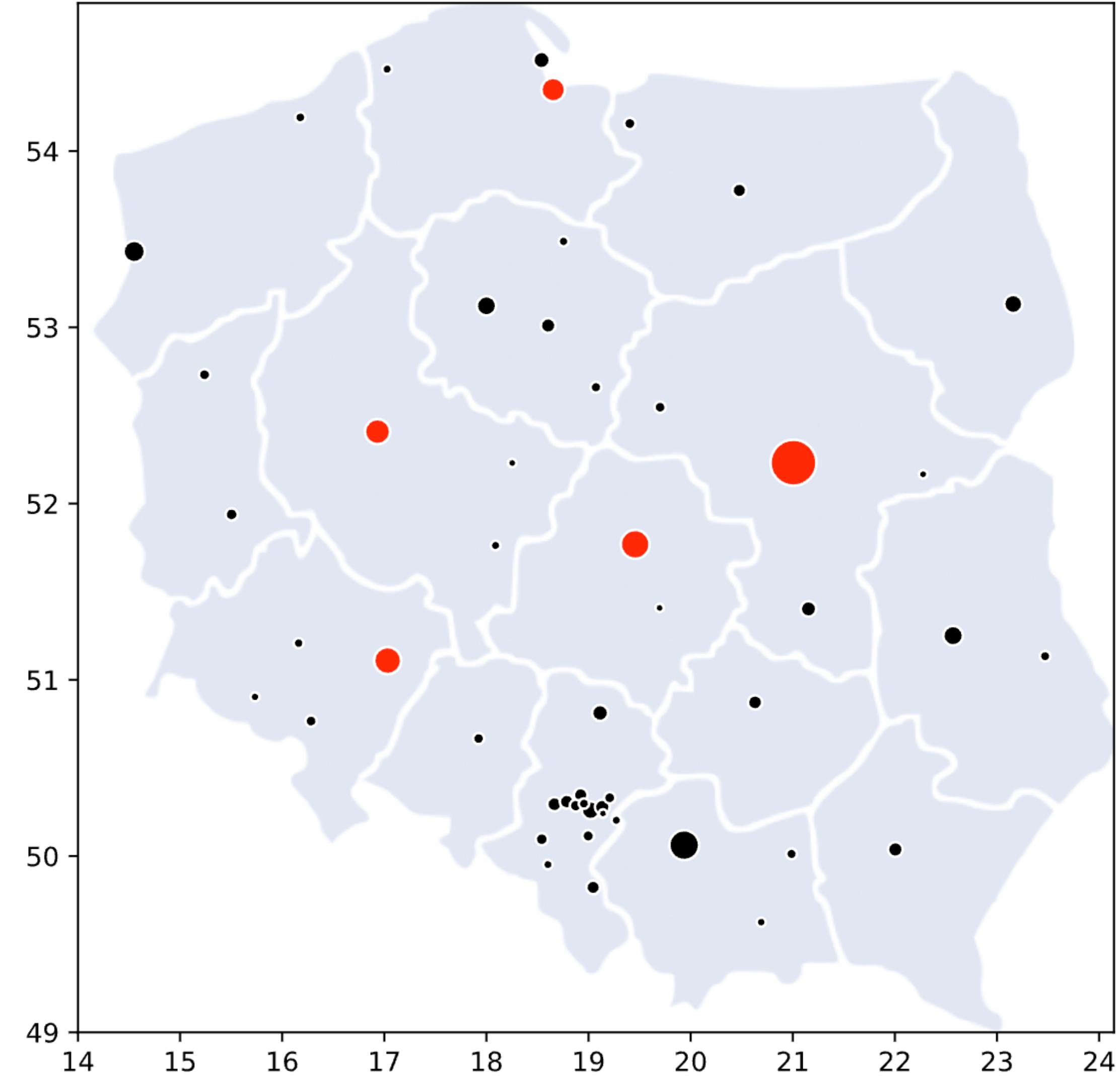
```
# scaling points based on city population
norm_pop = (df['population']-df['population'].min())/(df['population'].max()-df['population'].min())
min_size, max_size = 10, 300
scaled_sizes = min_size + (max_size - min_size) * norm_pop

# choosing points color depending if there is lego shop
point_colors = list(df["how_many_legoshops"].apply(lambda x: "red" if x>=1 else "black"))
step = 1. # both axis tick step
fig, ax = plt.subplots(1, 1, figsize=(7, 7))
img = plt.imread("assets/Poland.png")
plt.imshow(img, extent=[MAX_W, MAX_E, MAX_S, MAX_N], aspect="auto", alpha=.3)
plt.scatter(data=df, x="longitude", y="latitude", c=point_colors, s=scaled_sizes,
            edgecolors='white', linewidth=1)
plt.xticks(range(14, 25))
plt.yticks(range(49, 55))
# plt.grid(True, alpha=.15, c="grey", linestyle="dashed")
with open('map.pkl', 'wb') as f:
    pickle.dump(plt.gcf(), f)
plt.show()
```



No i git, co teraz?

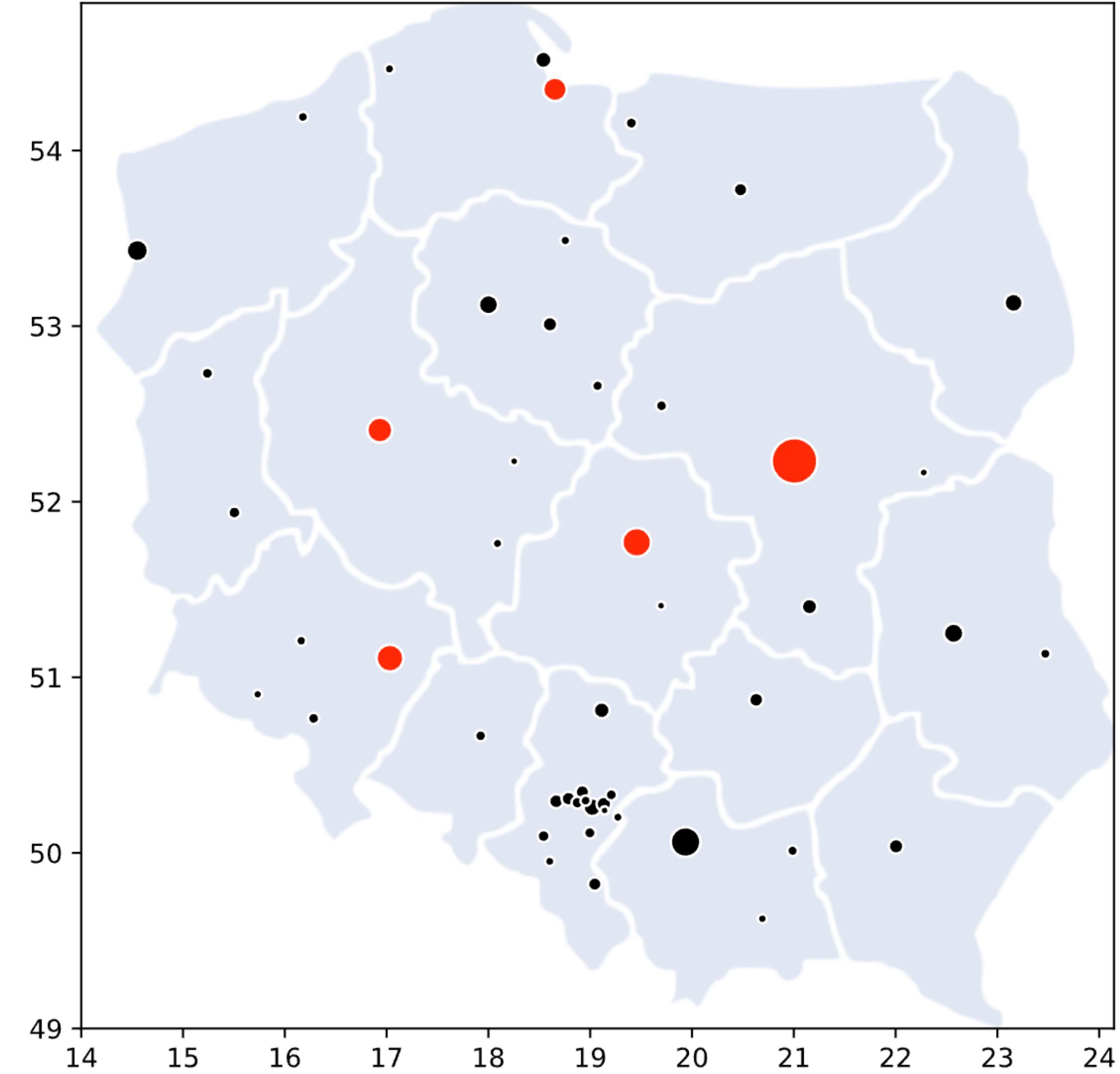
Tu zaczynają się schody bo
trzeba coś mądrego
zaprogramować, żeby udzielić
odpowiedzi na pytanie z tytułu.



No i git, co teraz?

Tu zaczynają się schody bo
trzeba coś mądryego
zaprogramować, żeby udzielić
odpowiedzi na pytanie z tytułu.

... ale czy na pewno?

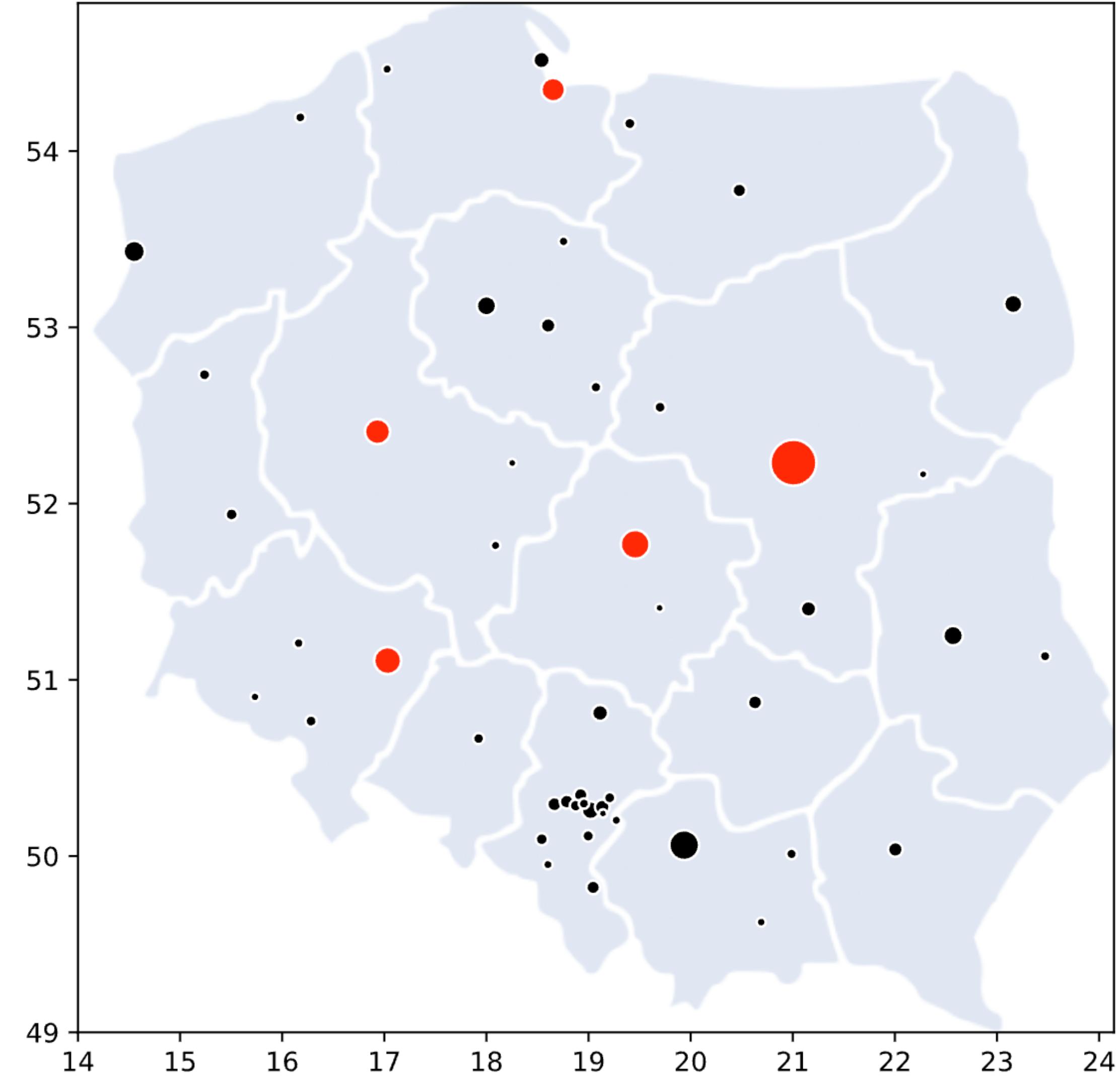


Metoda 1.

Chłopski rozum

Metoda 1. Chłopski rozum

Polega na popatrzeniu na mapę i
użyciu mózgu.

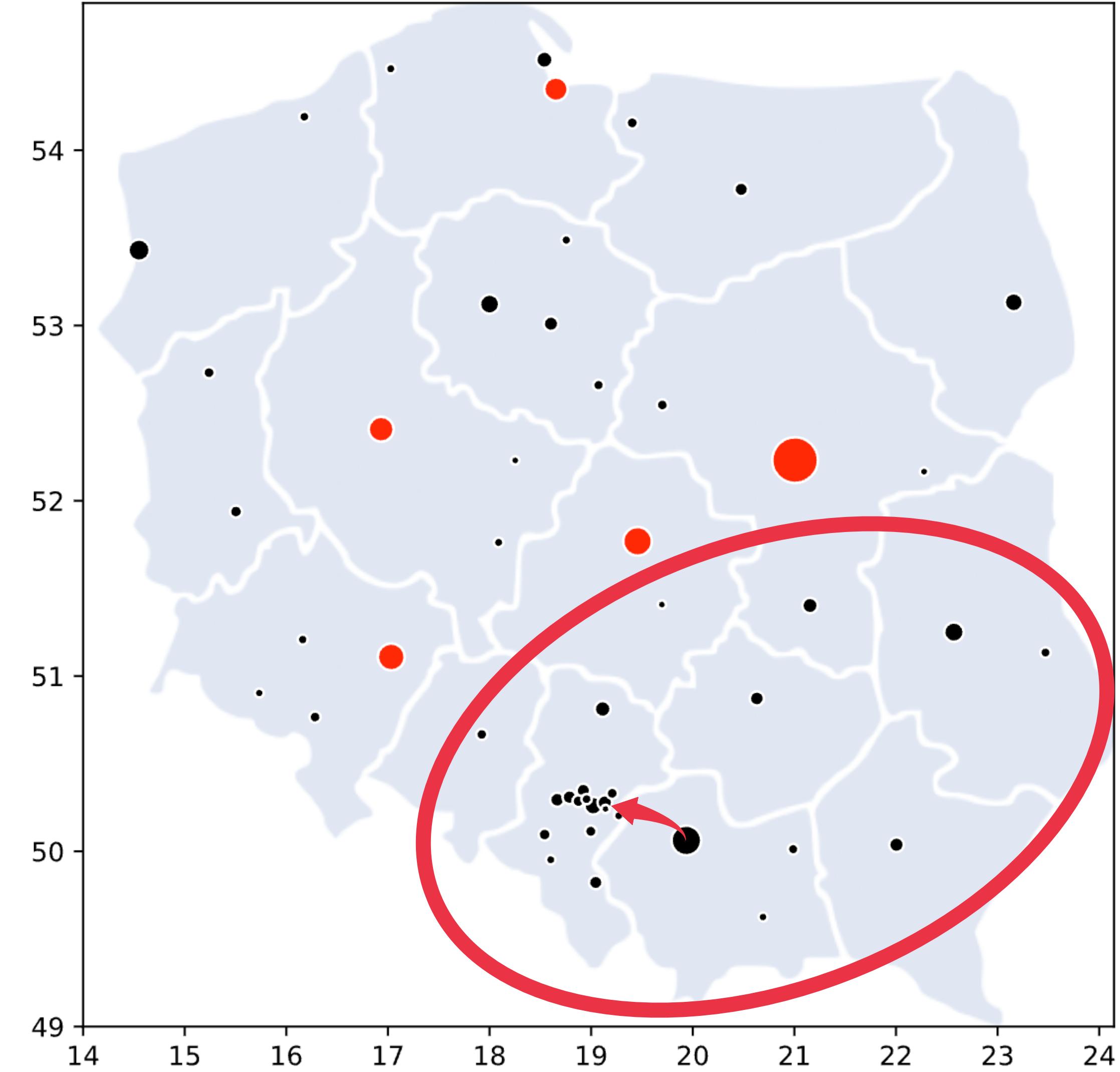


Metoda 1. Chłopski rozum

Polega na popatrzeniu na mapę i
użyciu mózgu.

No pewnie gdzieś tu powinniśmy
jechać bo jest dużo zaludnionych
miast i relatywnie daleko do
Lego Shopów.

Ja bym jechał na Śląsk.



Dziękujemy za uwagę!

Krzysztof Stawarz

Lena Stec

Informatyka Społeczna WH AGH

Modelowanie w Data Mining

Kraków, styczeń 2024

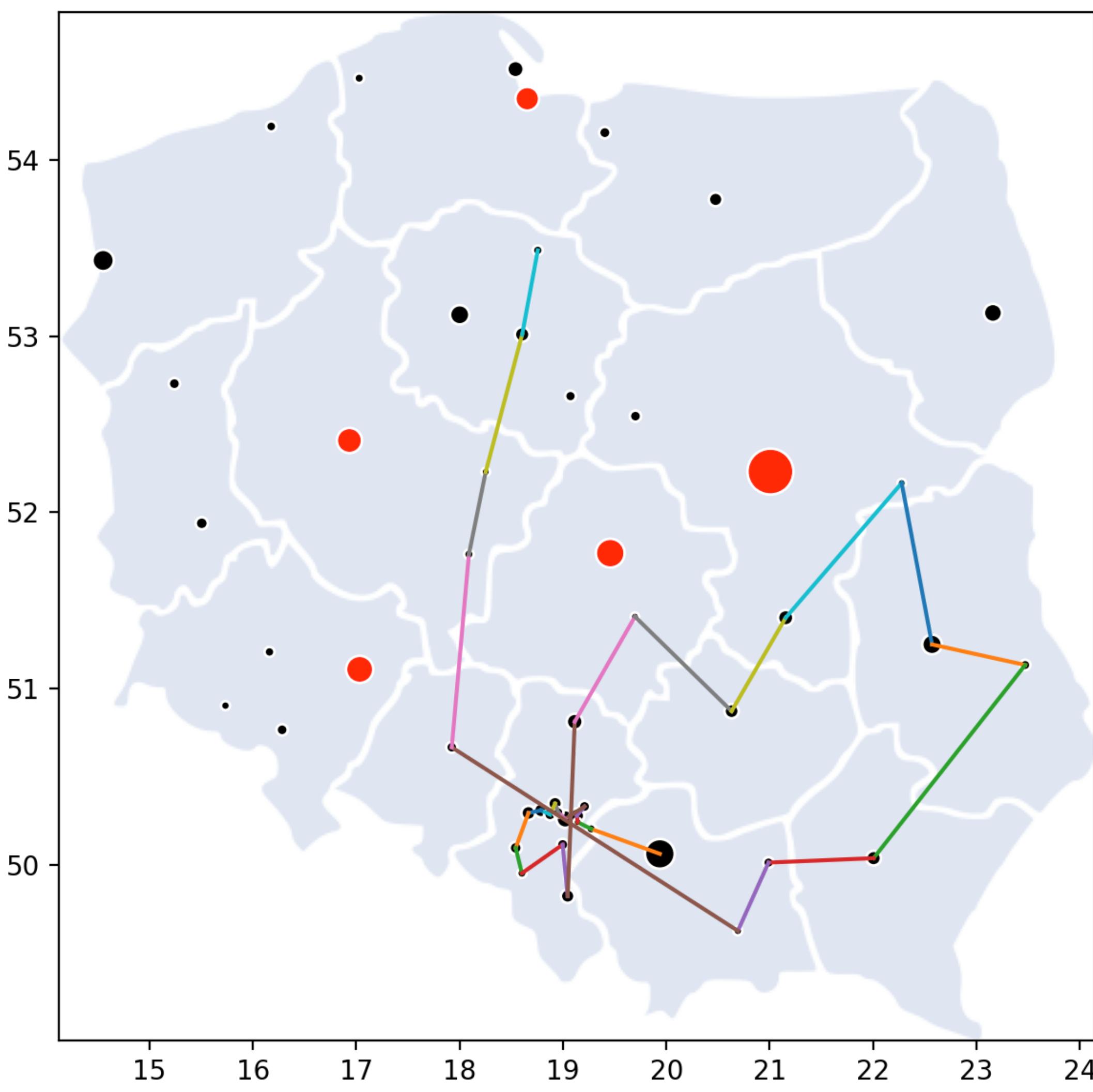


Żarcik

Metoda 2. Nearest Neighbour



```
1 import pandas as pd
2 import numpy as np
3 import pickle
4 import matplotlib.pyplot as plt
5
6 with open('map.pkl', 'rb') as f:
7     fig = pickle.load(f)
8 df = pd.read_csv("data/final_data.csv")
9
10 df = df[df["how_many_legoshops"]==0]
11 df.set_index('name', inplace=True)
12
13 def find_closest_city(city):
14     distances = np.sqrt(((df[['latitude', 'longitude']]
15                         .astype(float)
16                         - city[['latitude', 'longitude']]
17                         .astype(float))**2)
18                         .sum(axis=1))
19
20     return distances.idxmin()
21
22 route = ["Kraków"]
23 starting_city = df.loc["Kraków"]
24
25 for _ in range(30):
26     closest_city_name = find_closest_city(starting_city)
27     closest_city = df.loc[closest_city_name]
28
29     plt.plot([starting_city['longitude'], closest_city['longitude']],
30             [starting_city['latitude'], closest_city['latitude']])
31
32     route.append(closest_city_name)
33     starting_city = closest_city
34
35 print(route)
36 plt.show()
```



Travelling salesman problem

Problem komiwojażera XD

Problem komiwojażera [\[edytuj\]](#)

Artykuł [Dyskusja](#) Czytaj Edytuj Edytuj kod źródłowy Wyświetl historię Narzędzia ▾

Problem komiwojażera (ang. *travelling salesman problem, TSP*) – zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym^{[1][2]}.

Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie^{[1][2]}.

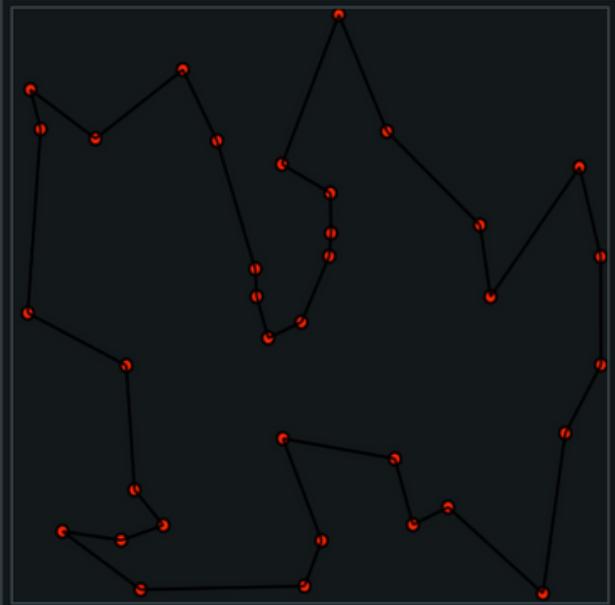
Symetryczny problem komiwojażera (STSP) polega na tym, że dla dowolnych miast A i B odległość z A do B jest taka sama jak z B do A. W **asymetrycznym problemie komiwojażera (ATSP)** odległości te mogą być różne.

Główna trudnością problemu jest duża liczba danych do analizy. W przypadku symetrycznego problemu komiwojażera dla n miast liczba kombinacji wynosi $\frac{(n - 1)!}{2}$ ^[3], tak więc dla 20 miast uzyskujemy wynik $\frac{19!}{2} \approx 6 \times 10^{16}$

Rozwinięciem problemu komiwojażera jest [problem marszrutyzacji](#).

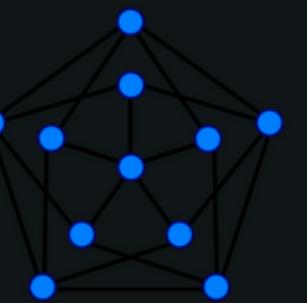
Historia [\[edytuj | edytuj kod\]](#)

Początek badań nad problemem komiwojażera nie jest jasny. Wspomina o nim podręcznik z 1832^[a], który zawiera przykładową trasę po Niemczech i Szwajcarii, lecz nie zawiera żadnych matematycznych uzasadnień.



Rozwiązywanie przykładowego problemu komiwojażera: najkrótszą ścieżką przechodzącą przez wszystkie czerwone punkty jest czarna pętla.

Niniejszy artykuł jest częścią cyklu [teoria grafów](#).

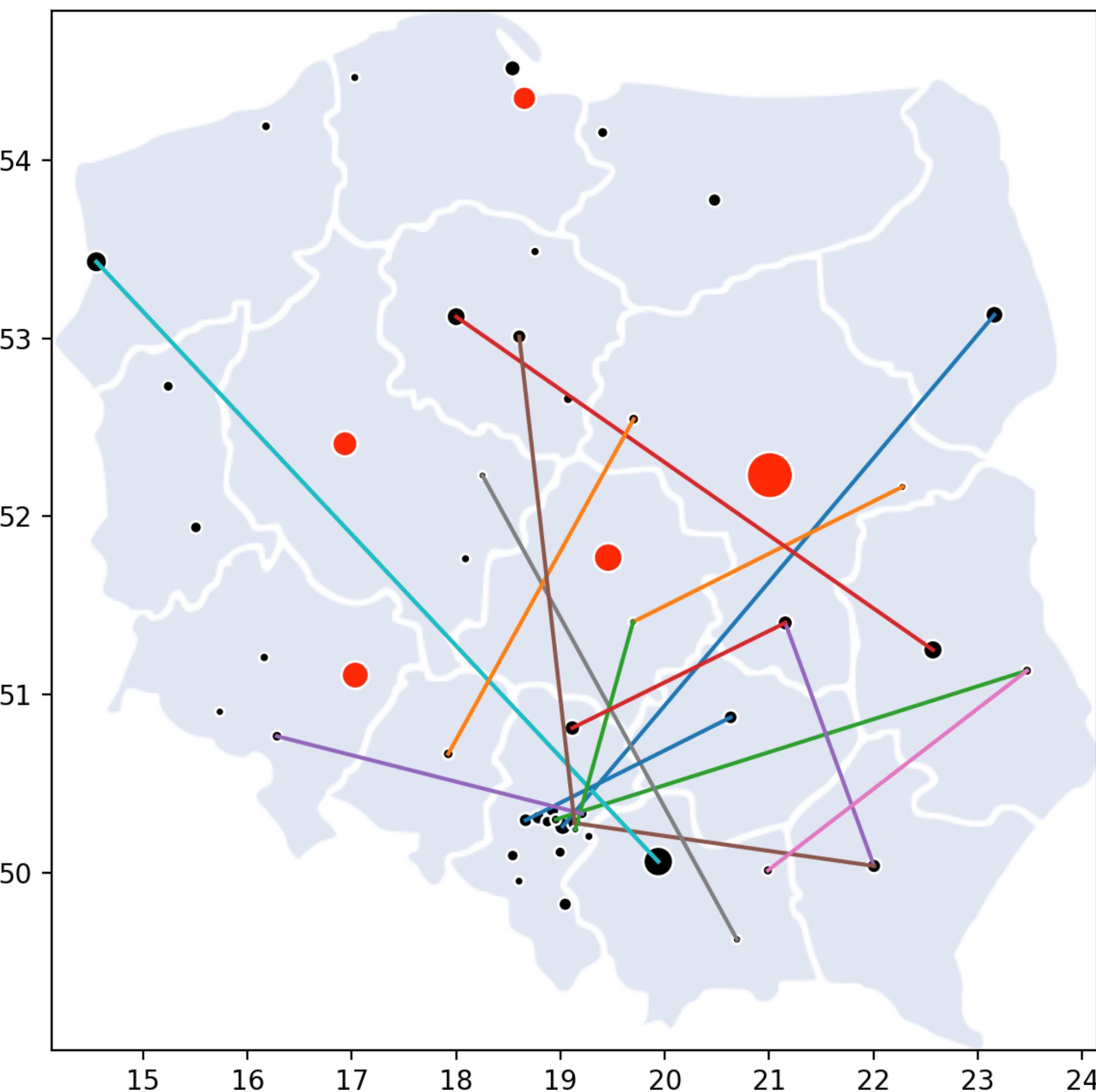


Metoda 3.

Simulated annealing



```
1 import pandas as pd
2 import numpy as np
3 from scipy.spatial import distance_matrix
4 from random import randint
5 import copy
6 import matplotlib.pyplot as plt
7 import pickle
8
9 df = pd.read_csv('data/final_data.csv')
10 df = df[df['how_many_legoshops'] == 0].reset_index(drop=True)
11
12 coordinates = df[['latitude', 'longitude']].to_numpy()
13 distance_matrix_result = distance_matrix(coordinates, coordinates)
14 distance = pd.DataFrame(distance_matrix_result)
15
16 def cost(route):
17     return sum([distance.iloc[route[i-1]][route[i]] for i in range(1, len(route))])
18
19 def simulated_annealing(route, T=100, alpha=0.999, stopping_T=0.0001, stopping_iter=1000):
20     iter = 1
21     while T >= stopping_T and iter < stopping_iter:
22         candidate = list(route)
23         l = randint(2, len(route) - 1)
24         i = randint(0, len(route) - 1)
25         candidate[i:(i+l)] = reversed(candidate[i:(i+l)])
26         F = cost(route)
27         F_new = cost(candidate)
28         if np.random.rand() < np.exp((F - F_new) / T):
29             route = copy.deepcopy(candidate)
30         T *= alpha
31         iter += 1
32     return route
33
34 start_city = df[df['name'] == 'Kraków'].index[0]
35 remaining_cities = list(set(df.index) - {start_city})
36 route = [start_city] + remaining_cities + [start_city]
37
38 optimal_route = simulated_annealing(route)
39 optimal_cost = cost(optimal_route)
```



Metoda 4.

Reszta co nie

wyszła

(pokazac w VSC)

Co bym zrobił
jakbym miał czas i
siły?

Pewnie Reinforcement
Learning (Q-Learning)

Dziękujemy
za uwagę!

(na serio)

Krzysztof Stawarz

Lena Stec

Informatyka Społeczna WH AGH

Modelowanie w Data Mining

Kraków, styczeń 2024

