

Анализ пространственных данных. Домашнее задание №2

Мягкий дедлайн: 4 ноября 2020 г. 23:59

Жесткий дедлайн (со штрафом в 50% от количества набранных вами за ДЗ баллов): 5 ноября 2020 г. 08:59

Визуализация "чего-либо" без выполненного основного задания оценивается в 0 баллов

ФИО: Панфилов Александр Николаевич

Группа: MADE-DS-32

Задание №1. Горячая точка (алгоритм - 10 баллов, визуализация - 10 баллов).

Генерируйте случайные точки на планете Земля до тех пор, пока не попадете на территорию Афганистана

1. Вы можете использовать функции принадлежности точки полигону и расстояния от точки до полигона (в метрах)
2. Предложите не наивный алгоритм поиска (генерировать **напрямую** точку из полигона границ Афганистана **запрещено**)

Визуализируйте пошагово предложенный алгоритм при помощи Folium

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import geopandas as gpd
4
5 import folium
6 import folium.plugins
7
8 import pyproj
9 import requests
10 import json
11
12 from datetime import datetime, timedelta
13 from haversine import haversine, Unit
14 import shapely.geometry
15 from shapely.geometry import Point, Polygon, LineString, MultiPolygon
16 from OSMPythonTools.overpass import overpassQueryBuilder, Overpass
17 from OSMPythonTools.nominatim import Nominatim
18
19 from rtree import index
20
21 import tqdm.notebook
```

```
In [2]: 1 url = "https://raw.githubusercontent.com/python-visualization/folium/master/examples/data/world-countries.json"
2
3 response = requests.get(url)
4 all_countries_data = json.loads(response.content.decode())
```

```
In [3]: 1 for item in all_countries_data['features']:
2     if item['properties']['name'] == 'Afghanistan':
3         afghanistan_data = item
4         break
5 afg_polygon = Polygon(afghanistan_data['geometry']['coordinates'][0])
6 afg_center = [afg_polygon.centroid.y, afg_polygon.centroid.x]
```

```
In [4]: 1 def dist_to_afg(x, y, afg_center=afg_center, unit=Unit.KILOMETERS):
2     return haversine((x, y), afg_center, unit = unit)
```

```
In [5]: 1 # Бун. поиск по осям нашего полигона
2 lat_1, lat_2, lon_1, lon_2 = -90, 90, -180, 180
3 point = Point(0, 0)
4 points = []
5 while (not Point(point.y, point.x).within(afg_polygon)):
6     points.append(point)
7     lat_traverse = abs(lat_2 - lat_1) // 4
8     lon_traverse = abs(lon_1 - lon_2) // 4
9
10    if (dist_to_afg(point.x, point.y - lon_traverse) <=
11        dist_to_afg(point.x, lon_traverse + point.y)):
12        lon_2 = point.y
13    else:
14        lon_1 = point.y
15
16    if (dist_to_afg(lat_traverse + point.x, point.y) <=
17        dist_to_afg(point.x - lat_traverse, point.y)):
18        lat_1 = point.x
19    else:
20        lat_2 = point.x
21    point = Point(lat_1 + abs(lat_2 - lat_1)//2, lon_1 + abs(lon_2 - lon_1)//2)
22    points.append(point)
```

```
In [6]: 1 # Preparing for visualization with Timestamped Geo Json
2 date = datetime.now()
3 features = [
4     {
5         'type': 'Feature',
6         'geometry': {
7             'type': 'LineString',
8             'coordinates': [[point.y, point.x]],
9         },
10        'properties': {
11            'times': [(date + timedelta(minutes = i*10)).strftime("%Y-%m-%dT%H:%M:%S")],
12        }
13    }
14    for i, point in enumerate(points)]
```

```
In [8]: 1 # Create map
2 search_map = folium.Map(location = afg_center,
3                          zoom_start = 1)
4 folium.GeoJson(afghanistan_data).add_to(search_map)
5 folium.LayerControl().add_to(search_map);
6
7 folium.plugins.TimestampedGeoJson({
8     'type': 'FeatureCollection',
9     'features': features,
10 }, period='PT1M', add_last_point=True).add_to(search_map);
```

```
In [9]: 1 search_map
```

Out[9]: Make this Notebook Trusted to load map: File -> Trust Notebook



Задание №2. Качество жизни (20 баллов).

1. Расстояние от точки до 5 ближайших * банкоматов, находящихся в стране с наибольшим количеством объектов жилой недвижимости
2. Расстояние от точки до 5 ближайших школ, находящихся в стране с наибольшим количеством аптек в столице
3. Расстояние от точки до 5 ближайших кинотеатров, находящихся в стране с самым большим отношением числа железнодорожных станций к автобусным остановкам в южной части **

Для измерения показателя качества жизни в точке, найденной в предыдущем задании, вам необходимо рассчитать следующую сумму расстояний (в метрах):

* При поиске N ближайших объектов обязательно использовать R-tree

** Южной частью страны является территория, находящаяся к югу от множества точек, равноудаленных от самой северной и самой южной точек страны

Пункт 1

1. Расстояние от точки до 5 ближайших банкоматов, находящихся в стране с наибольшим количеством объектов жилой недвижимости

```
In [10]: 1 POINT_ = points[-1]
          2 overpass = Overpass()
          3 nominatim = Nominatim()
```

```
In [11]: 1 # union запросы
          2 def query_residential_buildings(country_id: int):
          3     query_line = f"area({country_id}) -> .t;"
          4     query_body = """
          5     (node["building"="residential"](area.t);
          6     node["building"~"apartments|bungalow|cabin|detached|dormitory|farm|ger|hotel|house|houseboat|residential|semidet
          7     node["landuse"="residential"](area.t)););
          8     out body;
          9     """
          10     return query_line + query_body
          11
          12 def query_capital(country_id: int):
          13     query_line = f"area({country_id}) -> .t;"
          14     query_body = """
          15     (nwr["capital"="yes"](area.t);
          16     nwr["capital"="country"](area.t);
          17     nwr["capital"="2"](area.t)););
          18     out body;
          19     """
          20     return query_line + query_body
          21
          22 def query_bus_station(country_id: int):
          23     query_line = f"area({country_id}) -> .t;"
          24     query_body = """
          25     (nw["amenity"="bus_station"](area.t);
          26     nw["highway"="bus_stop"](area.t)););
          27     out body;
          28     """
          29     return query_line + query_body
          30
          31 def query_apotek(country_id: int):
          32     query_line = f"area({country_id}) -> .t;"
          33     query_body = """
          34     (nwr["amenity"="pharmacy"](area.t);
          35     nwr["shop"="chemist"](area.t)););
          36     out body;
          37     """
          38     return query_line + query_body
```

```
In [12]: 1 # берем все страны оканчивающиеся на 0
          2 response = overpass.query('relation[admin_level=2][type=boundary][boundary=administrative];out;', timeout = 120)
          3 countries_data = {}
          4 for country in response.elements():
          5     if country.id() % 10 == 0:
          6         countries_data[country.id() + 3600000000] = country.tags()['name:en']
```

```
In [13]: 1 # считаем количество жилых строений
2 countries_buildings = []
3 for id_ in tqdm.notebook.tqdm(countries_data):
4     buildings = overpass.query(query_residential_buildings(id_), timeout=120).countElements()
5     countries_buildings.append((id_, countries_data[id_], buildings))
```

100% 25/25 [00:08<00:00, 3.01it/s]

```
In [14]: 1 country = max(countries_buildings, key = lambda x: x[2])
2 print(country, 'Страна с наибольшим количеством жилых строений (согласно osm)')
```

(3600195290, 'Malawi', 9749) Страна с наибольшим количеством жилых строений (согласно osm)

```
In [15]: 1 # ищем наши банкоматы
2 query = overpassQueryBuilder(area=country[0], elementType=['node'], selector='amenity=atm')
3 atms = overpass.query(query)
4
5 idx = index.Index()
6 for atm in atms.elements():
7     idx.insert(atm.id(), atm.geometry().coordinates)
8 nearest_atms = idx.nearest((POINT_.x, POINT_.y), 5, objects = True)
```

```
In [16]: 1 result = sum([haversine((p.bbox[0], p.bbox[1]), (POINT_.x, POINT_.y), unit = Unit.KILOMETERS) for p in nearest_atms])
2 print("Сумма расстояний от точки, до ближайших 5 банкоматов в найденной стране (км)", result)
```

Сумма расстояний от точки, до ближайших 5 банкоматов в найденной стране (км) 35088.20386300688

Пункт 2

2. Расстояние от точки до 5 ближайших школ, находящихся в стране с наибольшим количеством аптек в столице

```
In [17]: 1 # собираем названия столиц
2 capitals = []
3 for id_ in countries_data:
4     if countries_data[id_] == "Tokelau":
5         print("Tokelau has no official capital!")
6         continue
7     res = overpass.query(query_capital(id_), timeout=120)
8
9     for el in res.elements():
10        if "admin_level" in el.tags() and el.tags()["admin_level"] == '2':
11            capitals.append((countries_data[id_], el.tags()["name:en"]))
```

Tokelau has no official capital!

```
In [18]: 1 # собираем аптеки в столицах
2 capital_apoteks = []
3 for capital in capitals:
4     areaId = nominatim.query(capital[1]).areaId()
5     if (areaId != None):
6         # баг с бразилией (помог дискорд)
7         if capital[0] == "Brazil":
8             areaId = 3602758138
9     capital_apoteks.append((capital, areaId, overpass.query(query_apotek(areaId), timeout=120).countElements()))
```

```
In [19]: 1 max_apoteks = max(capital_apoteks, key=lambda x: x[2])
2 print(max_apoteks, 'Страна с наибольшим количеством аптек в столице (согласно osm)')
3 # лол
```

(('Uzbekistan', 'Tashkent'), 3602216724, 857) Страна с наибольшим количеством аптек в столице (согласно osm)

```
In [20]: 1 # собираем все школы в узбекистане
2 query = overpassQueryBuilder(area=max_apoteks[1], elementType=['node'], selector='amenity=school', includeGeometry=True)
3 response = overpass.query(query)
4
5 idx = index.Index()
6 # находим как в предыдущем пункте
7 for el in response.elements():
8     idx.insert(el.id(), el.geometry().coordinates)
9 nearest_schools = idx.nearest((POINT_.x, POINT_.y), 5, objects = True)
```

```
In [21]: 1 result = sum([haversine((p.bbox[0], p.bbox[1]), (POINT_.x, POINT_.y), unit = Unit.KILOMETERS) for p in nearest_schools])
2 print("Сумма расстояний до 5 ближайших школ в Узбекистане (км)", result)
```

Сумма расстояний до 5 ближайших школ в Узбекистане (км) 21659.239046425682

Пункт 3

3. Расстояние от точки до 5 ближайших кинотеатров, находящихся в стране с самым большим отношением числа железнодорожных станций к автобусным остановкам в южной части

```
In [22]: 1 # берем южный полигон страны, по ее пограничным точкам
2 def get_south_polygon(country):
3     resp = overpass.query(f'relation[admin_level=2][type=boundary][boundary=administrative]["name:en"="{country}"];o
4     if resp.elements()[0].geometry().type == 'Polygon':
5         bound_points = np.array(resp.elements()[0].geometry().coordinates[0])
6         bound_points.reshape(bound_points.shape[-2], 2)
7     elif resp.elements()[0].geometry().type == 'MultiPolygon':
8         bound_points = np.array([point for polygon in resp.elements()[0].geometry().coordinates for point in polygon
9
10     return Polygon([[bound_points[:, 0].max(), bound_points[:, 1].max()],
11                     [bound_points[:, 0].max(), bound_points[:, 1].min()],
12                     [bound_points[:, 0].min(), bound_points[:, 1].min()]])
```

```
In [23]: 1 def get_within_point(el):
2     if el.geometry().type == 'Point':
3         point = Point(el.geometry().coordinates[0], el.geometry().coordinates[1])
4     elif el.geometry().type == 'Polygon':
5         centroid = Polygon(el.geometry().coordinates[0]).centroid
6         point = Point(centroid.x, centroid.y)
7     return point
8
9 def count_points_within_polygon(response, polygon):
10     obj_counter = 0
11     for el in response.elements():
12         if el.type() == 'node':
13             point = get_within_point(el)
14         else:
15             for node in el.nodes():
16                 point = get_within_point(node)
17             if (point.within(polygon)):
18                 obj_counter += 1
19     return obj_counter
```

```
In [24]: 1 # собираем страны автобусные остановки и жд вокзалы
2 countre_rel = []
3 for id_ in tqdm.notebook.tqdm(countries_data):
4     south_part = get_south_polygon(countries_data[id_])
5     country = countries_data[id_]
6     if countries_data[id_] == "Brazil":
7         id_ = 3602758138
8     denominator = 1+count_points_within_polygon(overpass.query(overpassQueryBuilder(area=id_, elementType=['node', '
9     try:
10         denominator += count_points_within_polygon(overpass.query(overpassQueryBuilder(area=id_, elementType=['node'
11     except:
12         pass
13     nominator = count_points_within_polygon(overpass.query(overpassQueryBuilder(area=id_, elementType=['node', 'way'
14     countre_rel.append((id_, country, nominator/denominator))
```

100%

25/25 [03:34<00:00, 8.58s/it]

Exception: [overpass] runtime error: Query timed out in "query" at line 1 after 26 seconds.

NoneType: None

[overpass] error in result (cache/overpass-8fa74a16cb841f152e01f5c756d951ffa2458f3f): [timeout:25][out:json];area(3600080500)->.searchArea;(node[highway=bus_stop](area.searchArea);way[highway=bus_stop](area.searchArea)); out geom;

NoneType: None

Exception: [overpass] runtime error: Query timed out in "query" at line 1 after 26 seconds.

NoneType: None

[overpass] error in result (cache/overpass-c30f9e8c4a04681a789dc07454ce2a72f547e01d): [timeout:25][out:json];area(3602978650)->.searchArea;(node[highway=bus_stop](area.searchArea);way[highway=bus_stop](area.searchArea)); out geom;

NoneType: None

```
In [25]: 1 max_rel = max(countre_rel, key=lambda x: x[2])
2 print(max_rel, 'Страна с наибольшим отношением жд станций к автобусным остановкам')
3 # с австралией что то не так - очень сомнительно, что такое отношение может быть > 1
```

(3600080500, 'Australia', 1.3454545454545455) Страна с наибольшим отношением жд станций к автобусным остановкам

```
In [26]: 1 # Собираем кинотеатры в Австралии
2 query = overpassQueryBuilder(area=max_rel[0], elementType=['node'], selector='amenity=cinema', includeGeometry=True)
3 response = overpass.query(query)
4
5 idx = index.Index()
6 for el in response.elements():
7     idx.insert(el.id(), el.geometry().coordinates)
8 nearest_schools = idx.nearest((POINT_.x, POINT_.y), 5, objects = True)
9 result = sum([haversine((p.bbox[0], p.bbox[1]), (POINT_.x, POINT_.y), unit = Unit.KILOMETERS) for p in nearest_schoo
```

```
In [27]: 1 print("Сумма расстояний до 5 кинотеатров в Австралии (км)", result)
```

Сумма расстояний до 5 кинотеатров в Австралии (км) 35011.90030466444

Задание №3. Поездка по Нью-Йорку (маршрут - 20 баллов, визуализация - 10 баллов).

Добраться **на автомобиле** от входа в Central Park **Нью-Йорка** (со стороны 5th Avenue) до пересечения Water Street и Washington Street в Бруклине (откуда получаются лучшие фото Манхэттенского моста) довольно непросто - разумеется, из-за вечных пробок. Однако еще сложнее это сделать, проезжая мимо школ, где дети то и дело переходят дорогу в неполюженном месте.

Вам необходимо построить описанный выше маршрут, избегая на своем пути школы. Визуализируйте данный маршрут (также добавив школы и недоступные для проезда участки дорог) при помощи Folium

Данные о расположении школ Нью-Йорка можно найти [здесь \(https://catalog.data.gov/dataset/2019-2020-school-point-locations\)](https://catalog.data.gov/dataset/2019-2020-school-point-locations)

```
In [28]: 1 def create_polygon(point_in, resolution=10, radius=10):
2     point_buffer_proj = point_in.buffer(radius, resolution=resolution)
3     poly_wgs = []
4     for point in point_buffer_proj.exterior.coords:
5         poly_wgs.append(point)
6     return poly_wgs
7
8 def style_route_function(color):
9     return lambda feature: dict(color=color,
10                                weight=5,
11                                opacity=1)
12
13 def school_slyle_function(color):
14     return lambda feature: dict(color=color,
15                                scale=1,
16                                opacity=1)
```

```
In [29]: 1 ny_json = json.load(open("new_york.json", 'r'))
```

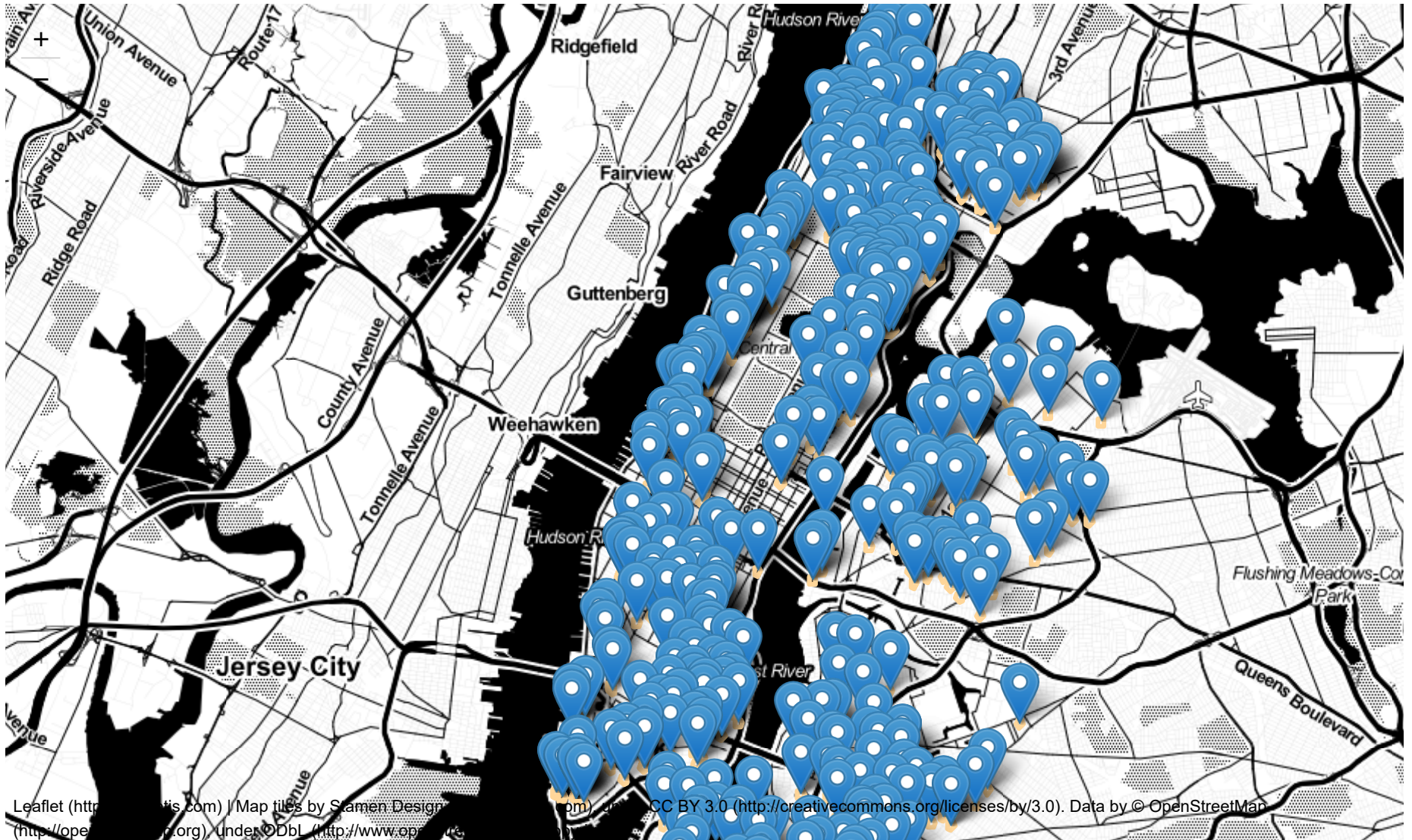
```
In [30]: 1 map_params = {'tiles': 'Stamen Toner',
2                 'location': ([40.764779, -73.972621]),
3                 'zoom_start': 12}
4 map_ny = folium.Map(**map_params)
5
6 polygonOfInterestInNY = Polygon(create_polygon(Point([40.764779, -73.972621]), radius=0.08))
```

```
In [31]: 1 sites_poly = []
2 for site_data in ny_json["data"]:
3     site_coords = [float(x) for x in list(filter(lambda x: "POINT" in str(x), site_data))[0].rsplit('(')[1].rsplit('
4     site_point = Point(reversed(site_coords))
5     if site_point.within(polygonOfInterestInNY):
6         folium.features.Marker(list(reversed(site_coords)),
7                                popup='School<br>{0}'.format(site_coords)).add_to(map_ny)
8     # Create buffer polygons around construction sites with 10 m radius and low resolution
9     site_poly_coords = create_polygon(Point(site_coords),
10                                     resolution=2, # Low resolution to keep polygons lean
11                                     radius=0.0009)
12     sites_poly.append(site_poly_coords)
13     site_poly_coords = [(y,x) for x,y in site_poly_coords] # Reverse coords for folium/Leaflet
14     folium.vector_layers.Polygon(locations=site_poly_coords,
15                                color='#ffd699',
16                                fill_color='#ffd699',
17                                fill_opacity=0.2,
18                                weight=3).add_to(map_ny)
```

Нарисуем школы в полигоне, который может относиться к маршруту

In [33]: 1 map_ny

Out[33]:



```
In [34]: 1 from openrouteservice import client
2         api_key = "5b3ce3597851110001cf6248060d47e0e2284c51bd23966179c84b7c"
3         clnt = client.Client(key=api_key)
```

```
In [35]: 1 request_params = {'coordinates': [[-73.972621, 40.764779],
2                                     [-73.989536, 40.703221]],
3                     'format_out': 'geojson',
4                     'profile': 'driving-car',
5                     'preference': 'shortest',
6                     'instructions': 'false',}
7         route_normal = clnt.directions(**request_params)
```

```
In [36]: 1 map_params_route = {'tiles': 'Stamen Toner',
2                       'location': ([40.728779, -73.972621]),
3                       'zoom_start': 12.5}
4         map_route = folium.Map(**map_params_route)
5
6         folium.features.GeoJson(data=route_normal,
7                                name='Route without construction sites',
8                                style_function=style_route_function("#5D98FE"),
9                                overlay=True).add_to(map_route)
10
11         route_buffer = LineString(route_normal['features'][0]['geometry']['coordinates']).buffer(0.05)
12
13         sites_buffer_poly = []
14         for site_poly in sites_poly:
15             poly = Polygon(site_poly)
16             if route_buffer.intersects(poly):
17                 folium.vector_layers.Circle(list(reversed(poly.centroid.coords[0])), color='red', radius=35).add_to(map_route)
18                 sites_buffer_poly.append(poly)
```

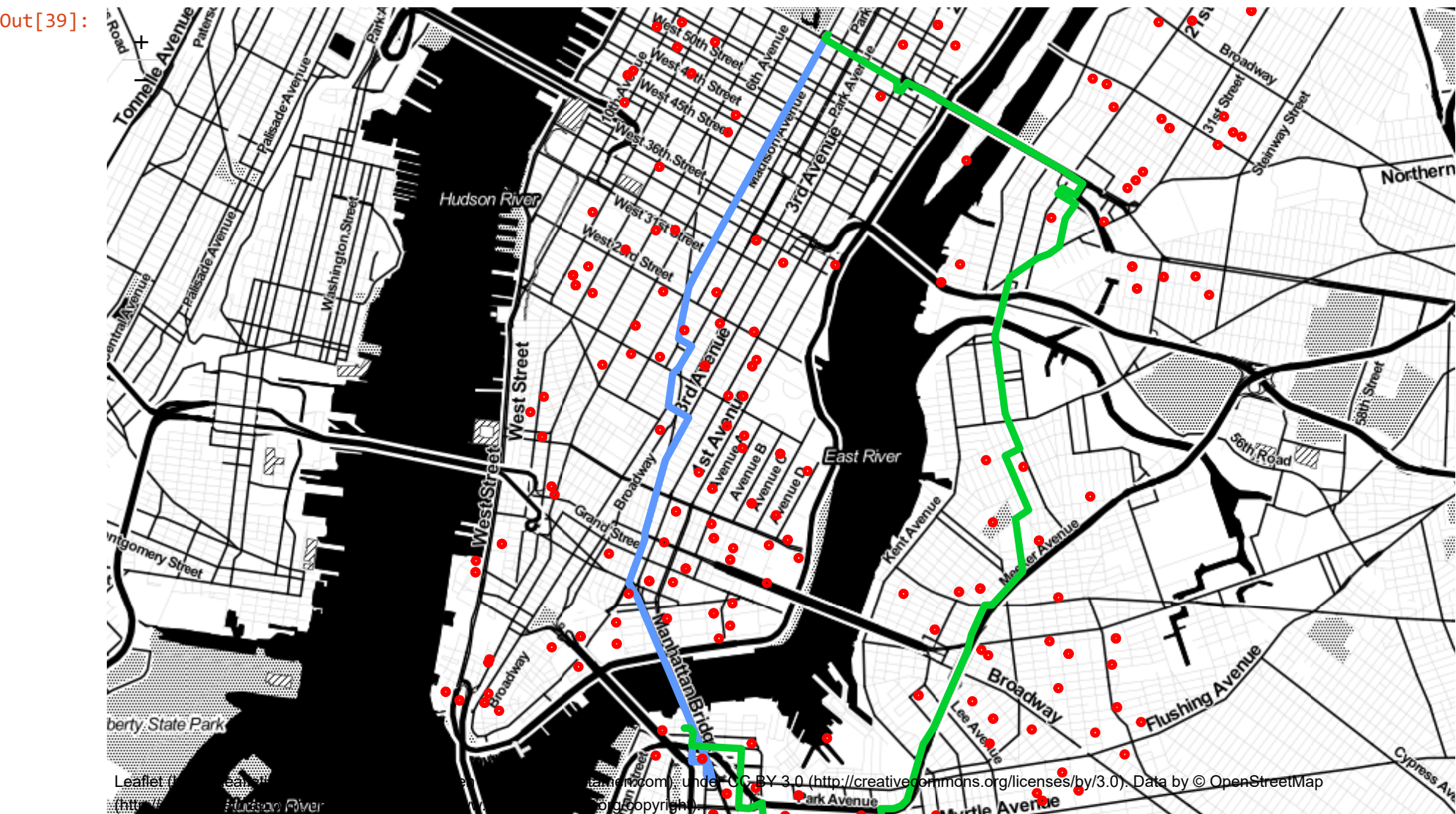
Нарисуем маршрут проходящий "наивно" - не учитывая школы

In [37]: 1 map_route



Нарисуем маршрут обходящий все наши школы (зеленый)

In [39]: 1 request_params['options'] = {'avoid_polygons': shapely.geometry.mapping(MultiPolygon(sites_buffer_poly))}
2 route_detour = clnt.directions(**request_params)
3
4 folium.features.GeoJson(data=route_detour,
5 name='Route with construction sites',
6 style_function=style_route_function('#00D12E'),
7 overlay=True).add_to(map_route)
8 map_route.add_child(folium.map.LayerControl())



In []: 1

