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 рандомные точки на планете Земля до тех пор, пока не попадете на территорию `Афганистана`

1. Вы можете использовать функции принадлжености 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 [325]:
```
import numpy as np
import pandas as pd
import geopandas as gpd

import folium
import folium.plugins

import pyproj
import requests
import json

from datetime import datetime, timedelta
from haversine import haversine, Unit
import shapely.geometry
from shapely.geometry import Point, Polygon, LineString, MultiPolygon
from OSMPythonTools.overpass import overpassQueryBuilder, Overpass
from OSMPythonTools.nominatim import Nominatim

from rtree import index

import tqdm.notebook
```

In [2]:
```
url = "https://raw.githubusercontent.com/python-visualization/folium/master/examples/data/world-countries.json"

response = requests.get(url)
all_countries_data = json.loads(response.content.decode())
```

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

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

In [48]:
```
# Бин. поиск по осям нашего полигона
lat_1, lat_2, lon_1, lon_2 = -90, 90, -180, 180
point = Point(0, 0)
points = []
while (not Point(point.y, point.x).within(afg_polygon)):
    points.append(point)
    lat_traverse = abs(lat_2 - lat_1) // 4
    lon_traverse = abs(lon_1 - lon_2) // 4

    if (dist_to_afg(point.x,  point.y - lon_traverse) <=
        dist_to_afg(point.x, lon_traverse + point.y)):
        lon_2 = point.y
    else:
        lon_1 = point.y

    if (dist_to_afg(lat_traverse + point.x, point.y) <=
        dist_to_afg(point.x - lat_traverse, point.y)):
        lat_1 = point.x
    else:
        lat_2 = point.x
    point = Point(lat_1 + abs(lat_2 - lat_1)//2, lon_1 + abs(lon_2 - lon_1)//2)
points.append(point)
```

In [49]:
```
# Preparing for visualization with Timestamped Geo Json
date = datetime.now()
features = [
    {
        'type': 'Feature',
        'geometry': {
            'type': 'LineString',
            'coordinates': [[point.y, point.x]],
        },
        'properties': {
            'times': [(date + timedelta(minutes = i*10)).strftime("%Y-%m-%dT%H:%M:%S")],
        }
    }
    for i, point in enumerate(points)]
```

In [50]:
```
# Create map
search_map = folium.Map(location = afg_center,
                zoom_start = 1)
folium.GeoJson(afghanistan_data).add_to(map_)
folium.LayerControl().add_to(map_);

folium.plugins.TimestampedGeoJson({
    'type': 'FeatureCollection',
    'features': features,
}, period='PT1M', add_last_point=True).add_to(search_map);
```

In [51]:
```
search_map
```

Out[51]:
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, наодящихся 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 [52]:
```
POINT_ = points[-1]
overpass = Overpass()
nominatim = Nominatim()
```

In [85]:
```
# union запросы
def query_residentional_buildings(country_id: int):
    query_line = f"area({country_id}) -> .t;"
    query_body =  """
    (node["building"="residential"](area.t);
    node["building"~"apartments|bungalow|cabin|detached|dormitory|farm|ger|hotel|house|houseboat|residential|semidet
    node["landuse"="residential"](area.t););
    out body;
    """
    return query_line + query_body

def query_capital(country_id: int):
    query_line = f"area({country_id}) -> .t;"
    query_body =  """
    (nwr["capital"="yes"](area.t);
    nwr["capital"="country"](area.t);
    nwr["capital"="2"](area.t););
    out body;
    """
    return query_line + query_body

def query_bus_station(country_id: int):
    query_line = f"area({country_id}) -> .t;"
    query_body =  """
    (nw["amenity"="bus_station"](area.t);
     nw["highway"="bus_stop"](area.t););
    out body;
    """
    return query_line + query_body

def query_apotek(country_id: int):
    query_line = f"area({country_id}) -> .t;"
    query_body =  """
    (nwr["amenity"="pharmacy"](area.t);
    nwr["shop"="chemist"](area.t););
    out body;
    """
    return query_line + query_body
```

In [55]:
```
# берем все страны оканчивающиеся на 0
response = overpass.query('relation[admin_level=2][type=boundary][boundary=administrative];out;', timeout = 120)
countries_data = {}
for country in response.elements():
    if country.id() % 10 == 0:
        countries_data[country.id() + 3600000000] = country.tags()['name:en']
```

In [56]:
```
# считаем количество жилый строений
countries_buildings = []
for id_ in tqdm.notebook.tqdm(countries_data):
    buildings = overpass.query(query_residentional_buildings(id_), timeout=120).countElements()
    countries_buildings.append((id_, countries_data[id_], buildings))
```

100% 25/25 [09:28<00:00, 22.76s/it]

In [57]:
```
country = max(countries_buildings, key = lambda x: x[2])
print(country, 'Страна с наибольшим количеством жилых строений (согласно osm)')
```

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

In [71]:
```
# ищем наши банкоматы
query = overpassQueryBuilder(area=country[0], elementType=['node'], selector='amenity=atm')
atms = overpass.query(query)

idx = index.Index()
for atm in atms.elements():
    idx.insert(atm.id(), atm.geometry().coordinates)
nearest_atms = idx.nearest((POINT_.x, POINT_.y), 5, objects = True)
```

In [72]:
```
result = sum([haversine((p.bbox[0], p.bbox[1]), (POINT_.x, POINT_.y), unit = Unit.KILOMETERS) for p in nearest_atms]
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 [74]:
```
# собираем названия столиц
capitals = []
for id_ in countries_data:
    if countries_data[id_] == "Tokelau":
        print("Tokelau has no official capital!")
        continue
    res = overpass.query(query_capital(id_), timeout=120)

    for el in res.elements():
        if "admin_level" in el.tags() and el.tags()["admin_level"] == '2':
            capitals.append((countries_data[id_], el.tags()["name:en"]))
```

```
Tokelau has no official capital!
```

In [75]:
```
# собираем аптеки в столицах
capital_apoteks = []
for capital in capitals:
    areaId = nominatim.query(capital[1]).areaId()
    if (areaId != None):
#         баг с бразилией (помог дискорд)
        if capital[0] == "Brazil":
            areaId = 3602758138
        capital_apoteks.append((capital, areaId, overpass.query(query_apotek(areaId), timeout=120).countElements()))
```

In [76]:
```
max_apoteks = max(capital_apoteks, key=lambda x: x[2])
print(max_apoteks, 'Страна с наибольшим количеством аптек в столице (согласно osm)')
# лол
```

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

In [81]:
```
# собираем все школы в узбекистане
query = overpassQueryBuilder(area=max_apoteks[1], elementType=['node'], selector='amenity=school',  includeGeometry=
response = overpass.query(query)

idx = index.Index()
# находим как в предыдущем пункте
for el in response.elements():
    idx.insert(el.id(), el.geometry().coordinates)
nearest_schools = idx.nearest((POINT_.x, POINT_.y), 5, objects = True)
```

In [82]:
```
result = sum([haversine((p.bbox[0], p.bbox[1]), (POINT_.x, POINT_.y), unit = Unit.KILOMETERS) for p in nearest_schoo
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, наодящихся в стране с самым большим отношением числа железнодорожных станций к автобусным остановкам в южной части

In [83]:
```
# берем южный полигон страны, по ее пограничным точкам
def get_south_polygon(country):
    resp = overpass.query(f'relation[admin_level=2][type=boundary][boundary=administrative]["name:en"="{country}"];o
    if resp.elements()[0].geometry().type == 'Polygon':
        bound_points = np.array(resp.elements()[0].geometry().coordinates[0])
        bound_points.reshape(bound_points.shape[-2], 2)
    elif resp.elements()[0].geometry().type == 'MultiPolygon':
        bound_points = np.array([point for polygon in resp.elements()[0].geometry().coordinates for point in polygon

    return Polygon([[bound_points[:, 0].max(),  bound_points[:, 1].max()],
                    [bound_points[:, 0].max(),  bound_points[:, 1].min()],
                    [bound_points[:, 0].min(), bound_points[:, 1].min()]])
```

In [145]:
```
def get_within_point(el):
    if el.geometry().type == 'Point':
        point = Point(el.geometry().coordinates[0], el.geometry().coordinates[1])
    elif el.geometry().type == 'Polygon':
        centroid = Polygon(el.geometry().coordinates[0]).centroid
        point = Point(centroid.x, centroid.y)
    return point

def count_points_within_polygon(response, polygon):
    obj_counter = 0
    for el in response.elements():
        if el.type() == 'node':
            point = get_within_point(el)
        else:
            for node in el.nodes():
                point = get_within_point(node)
        if (point.within(polygon)):
            obj_counter += 1
    return obj_counter
```

In [ ]:
```
# собираем страны автобусные остановки и жд вокзалы
countre_rel = []
for id_ in tqdm.notebook.tqdm(countries_data):
    south_part = get_south_polygon(countries_data[id_])
    country = countries_data[id_]
    if countries_data[id_] == "Brazil":
        id_ = 3602758138
    denominator = 1+count_points_within_polygon(overpass.query(overpassQueryBuilder(area=id_, elementType=['node', '
    try:
        denominator += count_points_within_polygon(overpass.query(overpassQueryBuilder(area=id_, elementType=['node'
    except:
        pass
    nominator = count_points_within_polygon(overpass.query(overpassQueryBuilder(area=id_, elementType=['node', 'way'
    countre_rel.append((id_, country, nominator/denominator))
```

In [149]:
```
max_rel = max(countre_rel, key=lambda x: x[2])
print(max_rel, 'Страна с наибольшим отношением жд станций к автобусным остановкам')
# 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 [152]:
```
# Собираем кинотеатры в Австралии
query = overpassQueryBuilder(area=max_rel[0], elementType=['node'], selector='amenity=cinema',  includeGeometry=True
response = overpass.query(query)

idx = index.Index()
for el in response.elements():
    idx.insert(el.id(), el.geometry().coordinates)
nearest_schools = idx.nearest((POINT_.x, POINT_.y), 5, objects = True)
result = sum([haversine((p.bbox[0], p.bbox[1]), (POINT_.x, POINT_.y), unit = Unit.KILOMETERS) for p in nearest_schoo
```

In [154]:
```
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 неположенном 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)

In [392]:
```
def CreateBufferPolygon(point_in, resolution=10, radius=10):

    point_buffer_proj = point_in.buffer(radius, resolution=resolution)
    poly_wgs = []
    for point in point_buffer_proj.exterior.coords:
        poly_wgs.append(point)
    return poly_wgs
```

In [373]:
```
def style_route_function(color):
    return lambda feature: dict(color=color,
                                weight=5,
                                opacity=1)
def school_slyle_function(color):
    return lambda feature: dict(color=color,
                                scale=1,
                                opacity=1)
```

In [396]:
```
ny_json = json.load(open("new_york.json", 'r'))
```

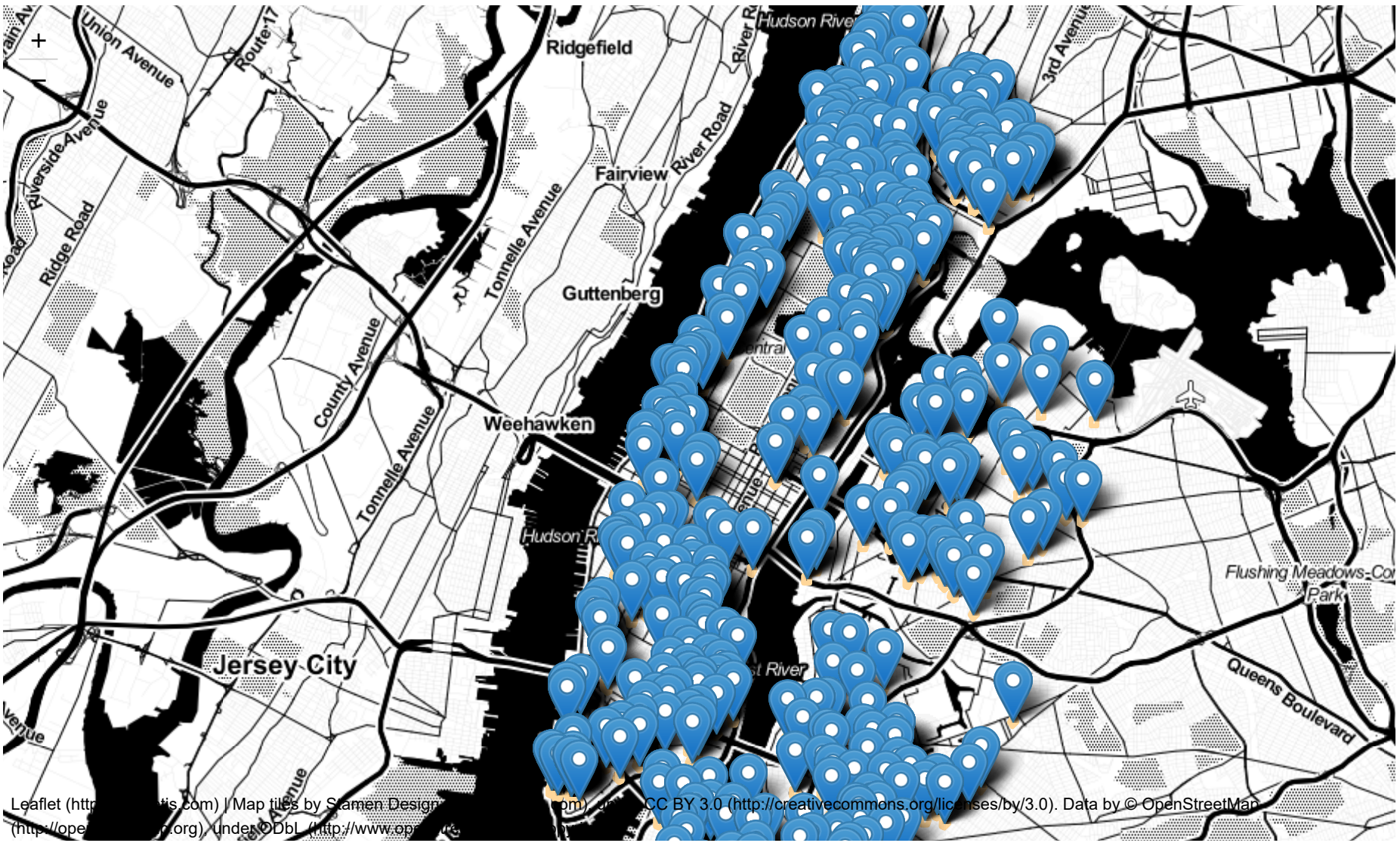
In [397]:
```
map_params = {'tiles':'Stamen Toner',
              'location':([40.764779, -73.972621]),
              'zoom_start': 12}
map_ny = folium.Map(**map_params)

polygonOfInterestInNY = Polygon(CreateBufferPolygon(Point([40.764779, -73.972621]), radius=0.08))
```

In [398]:
```
sites_poly = []
for site_data in ny_json["data"]:
    site_coords = [float(x) for x in list(filter(lambda x: "POINT" in str(x), site_data))[0].rsplit('(')[1].rsplit('
    site_point = Point(reversed(site_coords))
    if site_point.within(polygonOfInterestInNY):
        folium.features.Marker(list(reversed(site_coords)),
                               popup='School<br>{0}'.format(site_coords)).add_to(map_ny)
        # Create buffer polygons around construction sites with 10 m radius and low resolution
        site_poly_coords = CreateBufferPolygon(Point(site_coords),
                                               resolution=2, # low resolution to keep polygons lean
                                               radius=0.0009)
        sites_poly.append(site_poly_coords)
        site_poly_coords = [(y,x) for x,y in site_poly_coords] # Reverse coords for folium/Leaflet
        folium.vector_layers.Polygon(locations=site_poly_coords,
                                     color='#ffd699',
                                     fill_color='#ffd699',
                                     fill_opacity=0.2,
                                     weight=3).add_to(map_ny)
```

Нарисуем школы в полигоне, который может относится к маршруту

In [399]:
```
map1
```

Out[399]:



In [400]:
```
from openrouteservice import client
api_key = "5b3ce3597851110001cf6248060d47e0e2284c51bd23966179c84b7c"
clnt = client.Client(key=api_key)
```

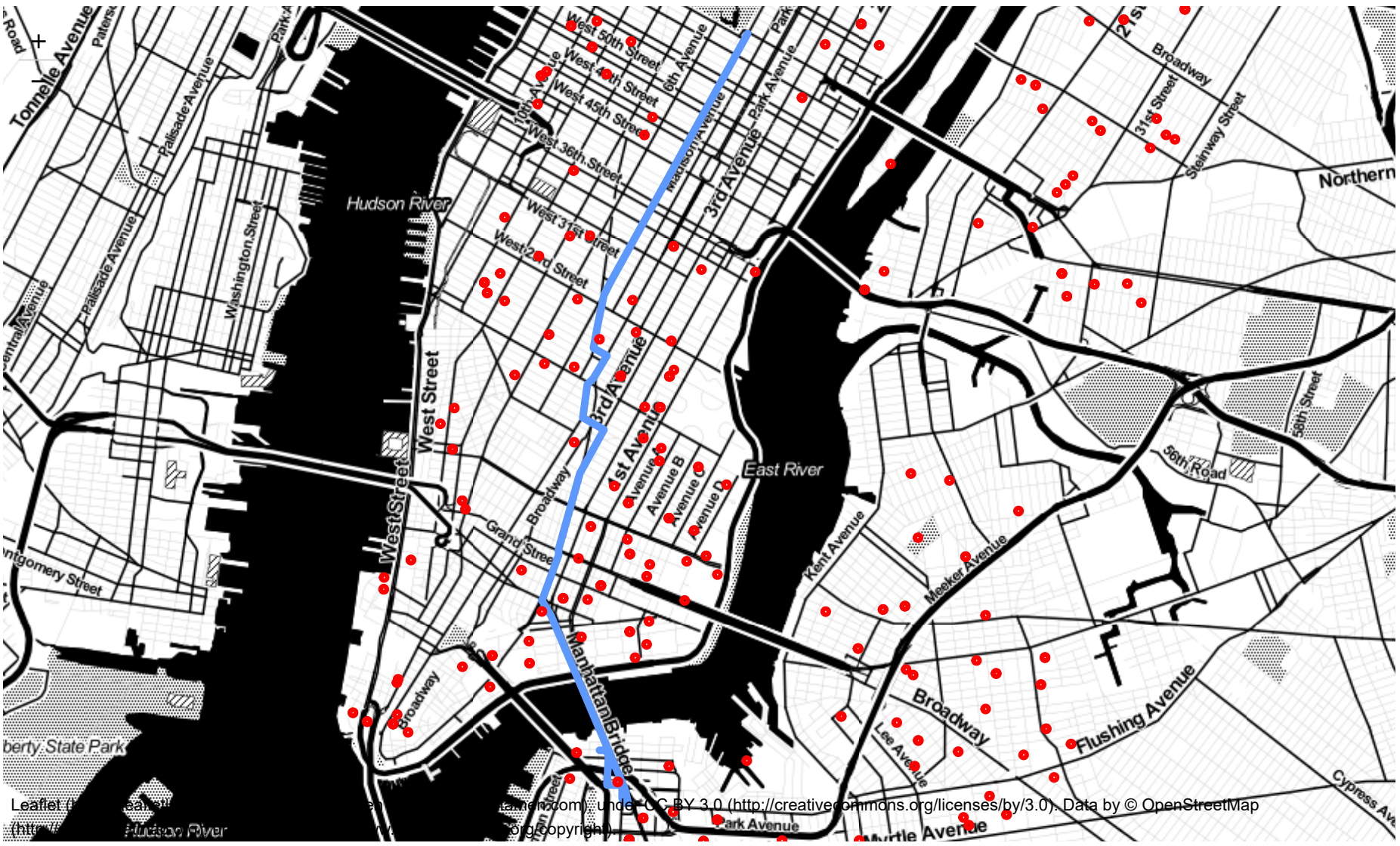
In [401]:
```
request_params = {'coordinates': [[-73.972621, 40.764779],
                                  [-73.989536, 40.703221]],
                  'format_out': 'geojson',
                  'profile': 'driving-car',
                  'preference': 'shortest',
                  'instructions': 'false',}
route_normal = clnt.directions(**request_params)
```

In [407]:
```
map_params_route = {'tiles':'Stamen Toner',
                'location': ([40.728779, -73.972621]),
                'zoom_start': 12.5}
map_route = folium.Map(**map_params_route)

folium.features.GeoJson(data=route_normal,
                        name='Route without construction sites',
                        style_function=style_route_function("#5D98FE"),
                        overlay=True).add_to(map_route)

route_buffer = LineString(route_normal['features'][0]['geometry']['coordinates']).buffer(0.05)

sites_buffer_poly = []
for site_poly in sites_poly:
    poly = Polygon(site_poly)
    if route_buffer.intersects(poly):
        folium.vector_layers.Circle(list(reversed(poly.centroid.coords[0])), color='red', radius=35).add_to(map_rout
        sites_buffer_poly.append(poly)
```

Нарисуем маршрут проходящий "наивно" - не учитывая школы

In [408]:
```
1 map_route
```

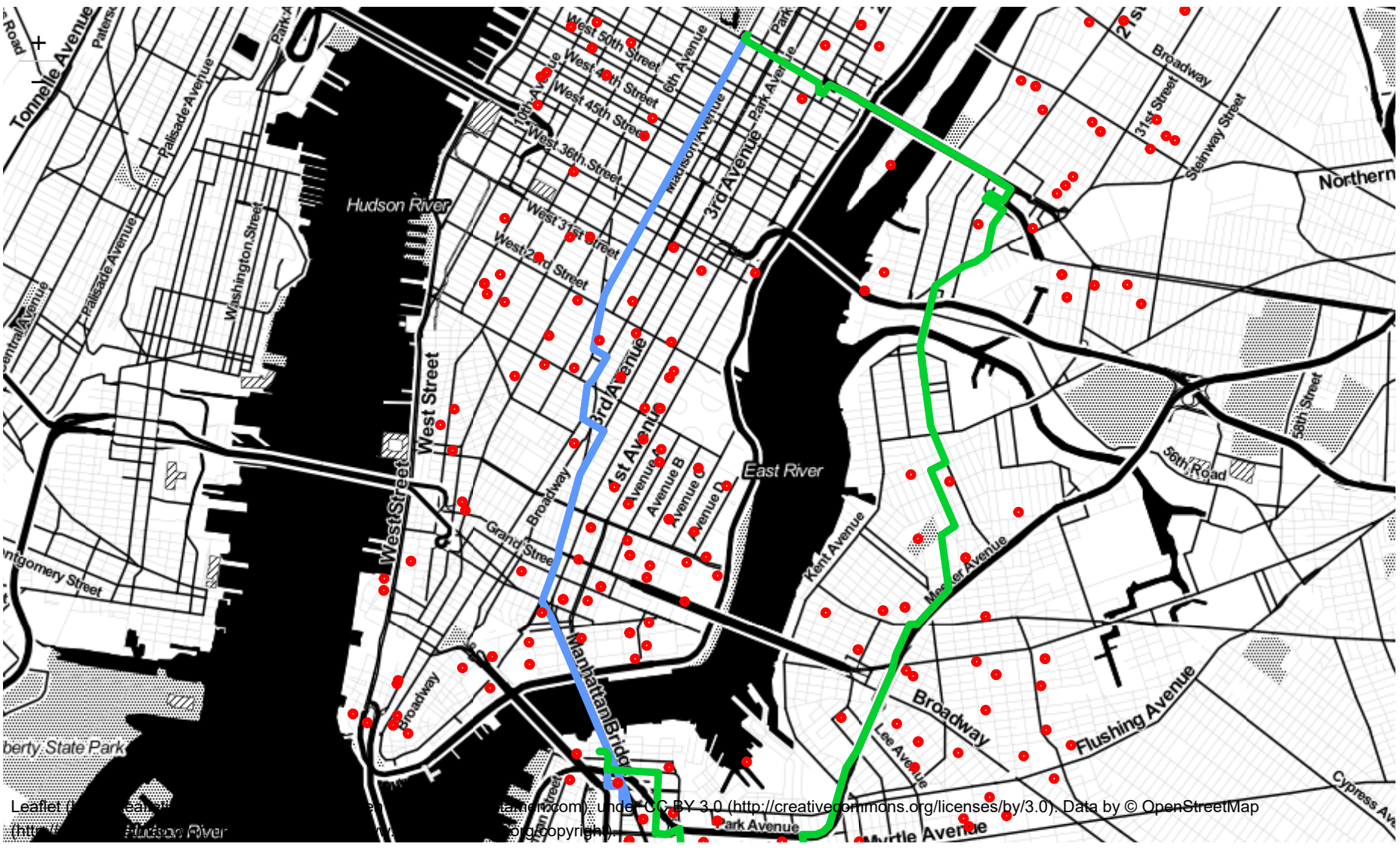
Out[408]:



Нарисуем маршрут обходящий все наши школы (зеленый)

In [412]:
```
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_function('#00D12E'),
7                         overlay=True).add_to(map_route)
8 map_route.add_child(folium.map.LayerControl())
```

Out[412]:



In [ ]:
```
1
```