Geospatial Analysis: 5th lesson – Proximity Analysis

In this tutorial, you'll explore several techniques for proximity analysis. In particular, you'll learn how to do such things as:

* Measure the distance between points on a map
* Select all points within some radius of a feature

import folium

from folium import Marker, GeoJson

from folium.plugins import HeatMap

import pandas as pd

import geopandas as gpd

/opt/conda/lib/python3.7/site-packages/geopandas/\_compat.py:115: UserWarning: The Shapely GEOS version (3.9.1-CAPI-1.14.2) is incompatible with the GEOS version PyGEOS was compiled with (3.10.4-CAPI-1.16.2). Conversions between both will be slow.

shapely\_geos\_version, geos\_capi\_version\_string

You'll work with a dataset from the US Environmental Protection Agency (EPA) that tracks releases of toxic chemicals in Philadelphia, Pennsylvania, USA.

releases = gpd.read\_file("../input/geospatial-learn-course-data/toxic\_release\_pennsylvania/toxic\_release\_pennsylvania/toxic\_release\_pennsylvania.shp")

releases.head()

YEAR CITY COUNTY ST LATITUDE LONGITUDE CHEMICAL UNIT\_OF\_ME TOTAL\_RELE geometry

0 2016 PHILADELPHIA PHILADELPHIA PA 40.005901 -75.072103 FORMIC ACID Pounds 0.160 POINT (2718560.227 256380.179)

1 2016 PHILADELPHIA PHILADELPHIA PA 39.920120 -75.146410 ETHYLENE GLYCOL Pounds 13353.480 POINT (2698674.606 224522.905)

2 2016 PHILADELPHIA PHILADELPHIA PA 40.023880 -75.220450 CERTAIN GLYCOL ETHERS Pounds 104.135 POINT (2676833.394 261701.856)

3 2016 PHILADELPHIA PHILADELPHIA PA 39.913540 -75.198890 LEAD COMPOUNDS Pounds 1730.280 POINT (2684030.004 221697.388)

4 2016 PHILADELPHIA PHILADELPHIA PA 39.913540 -75.198890 BENZENE Pounds 39863.290 POINT (2684030.004 221697.388)

You'll also work with a dataset that contains readings from air quality monitoring stations in the same city.

stations = gpd.read\_file("../input/geospatial-learn-course-data/PhillyHealth\_Air\_Monitoring\_Stations/PhillyHealth\_Air\_Monitoring\_Stations/PhillyHealth\_Air\_Monitoring\_Stations.shp")

stations.head()

SITE\_NAME ADDRESS BLACK\_CARB ULTRAFINE\_ CO SO2 OZONE NO2 NOY\_NO PM10 ... PAMS\_VOC TSP\_11101 TSP\_METALS TSP\_LEAD TOXICS\_TO1 MET COMMUNITY\_ LATITUDE LONGITUDE geometry

0 LAB 1501 East Lycoming Avenue N N Y N Y Y Y N ... Y N Y N y N N 40.008606 -75.097624 POINT (2711384.641 257149.310)

1 ROX Eva and Dearnley Streets N N N N N N N N ... N N Y N Y N N 40.050461 -75.236966 POINT (2671934.290 271248.900)

2 NEA Grant Avenue and Ashton Street N N N N Y N N N ... N N N N N Y N 40.072073 -75.013128 POINT (2734326.638 280980.247)

3 CHS 500 South Broad Street N N N N N N N N ... N N Y N Y N N 39.944510 -75.165442 POINT (2693078.580 233247.101)

4 NEW 2861 Lewis Street N N Y Y Y N Y Y ... N Y N Y N Y N 39.991688 -75.080378 POINT (2716399.773 251134.976)

Measuring distance:

To measure distances between points from two different GeoDataFrames, we first have to make sure that they use the same coordinate reference system (CRS). Thankfully, this is the case here, where both use EPSG 2272.

print(stations.crs)

print(releases.crs)

PROJCS["NAD83\_Pennsylvania\_South\_ftUS",GEOGCS["GCS\_North\_American\_1983",DATUM["D\_North\_American\_1983",SPHEROID["GRS\_1980",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.0174532925199433]],PROJECTION["Lambert\_Conformal\_Conic\_2SP"],PARAMETER["latitude\_of\_origin",39.3333333333333],PARAMETER["central\_meridian",-77.75],PARAMETER["standard\_parallel\_1",40.9666666666667],PARAMETER["standard\_parallel\_2",39.9333333333333],PARAMETER["false\_easting",1968500],PARAMETER["false\_northing",0],UNIT["Foot\_US",0.304800609601219],AXIS["Easting",EAST],AXIS["Northing",NORTH]]

PROJCS["NAD83\_Pennsylvania\_South\_ftUS",GEOGCS["GCS\_North\_American\_1983",DATUM["D\_North\_American\_1983",SPHEROID["GRS\_1980",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.0174532925199433]],PROJECTION["Lambert\_Conformal\_Conic\_2SP"],PARAMETER["latitude\_of\_origin",39.3333333333333],PARAMETER["central\_meridian",-77.75],PARAMETER["standard\_parallel\_1",40.9666666666667],PARAMETER["standard\_parallel\_2",39.9333333333333],PARAMETER["false\_easting",1968500],PARAMETER["false\_northing",0],UNIT["Foot\_US",0.304800609601219],AXIS["Easting",EAST],AXIS["Northing",NORTH]]

We also check the CRS to see which units it uses (meters, feet, or something else). In this case, EPSG 2,272 has units of feet. It's relatively straightforward to compute distances in GeoPandas. The code cell below calculates the distance (in feet) between a relatively recent release incident in recent\_release and every station in the stations GeoDataFrame.

*# Select one release incident in particular*

recent\_release = releases.iloc[360]

*# Measure distance from release to each station*

distances = stations.geometry.distance(recent\_release.geometry)

distances

0 44778.509761

1 51006.456589

2 77744.509207

3 14672.170878

4 43753.554393

5 4711.658655

6 23197.430858

7 12072.823097

8 79081.825506

9 3780.623591

10 27577.474903

11 19818.381002

dtype: float64

Using the calculated distances, we can obtain statistics like the mean distance to each station.

print('Mean distance to monitoring stations: **{}** feet'.format(distances.mean()))

Mean distance to monitoring stations: 33516.28487007786 feet

Or, we can get the closest monitoring station.

print('Closest monitoring station (**{}** feet):'.format(distances.min()))

print(stations.iloc[distances.idxmin()][["ADDRESS", "LATITUDE", "LONGITUDE"]])

Closest monitoring station (3780.623590556444 feet):

ADDRESS 3100 Penrose Ferry Road

LATITUDE 39.91279

LONGITUDE -75.185448

Name: 9, dtype: object

Creating a buffer:

If we want to understand all points on a map that are some radius away from a point, the simplest way is to create a buffer. The code cell below creates a GeoSeries two\_mile\_buffer containing 12 different Polygon objects. Each polygon is a buffer of 2 miles (or, 2 \* 5,280 feet) around a different air monitoring station.

two\_mile\_buffer = stations.geometry.buffer(2\*5280)

two\_mile\_buffer.head()

0 POLYGON ((2721944.641 257149.310, 2721893.792 ...

1 POLYGON ((2682494.290 271248.900, 2682443.441 ...

2 POLYGON ((2744886.638 280980.247, 2744835.789 ...

3 POLYGON ((2703638.580 233247.101, 2703587.731 ...

4 POLYGON ((2726959.773 251134.976, 2726908.924 ...

dtype: geometry

We use folium.GeoJson() to plot each polygon on a map. Note that since folium requires coordinates in latitude and longitude, we have to convert the CRS to EPSG 4326 before plotting.

*# Create map with release incidents and monitoring stations*

m = folium.Map(location=[39.9526,-75.1652], zoom\_start=11)

HeatMap(data=releases[['LATITUDE', 'LONGITUDE']], radius=15).add\_to(m)

for idx, row **in** stations.iterrows():

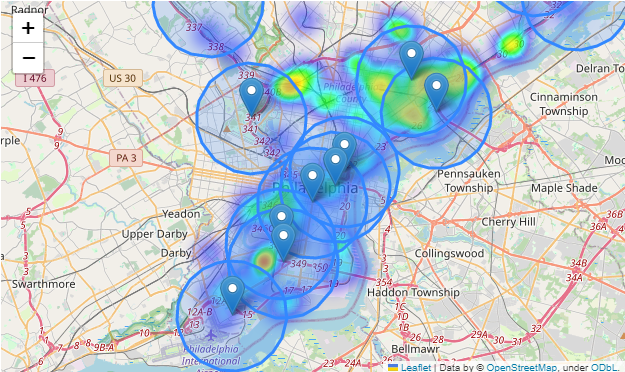
Marker([row['LATITUDE'], row['LONGITUDE']]).add\_to(m)

*# Plot each polygon on the map*

GeoJson(two\_mile\_buffer.to\_crs(epsg=4326)).add\_to(m)

*# Show the map*

m



Now, to test if a toxic release occurred within 2 miles of any monitoring station, we could run 12 different tests for each polygon (to check individually if it contains the point). But a more efficient way is to first collapse all of the polygons into a **MultiPolygon** object. We do this with the unary\_union attribute.

*# Turn group of polygons into single multipolygon*

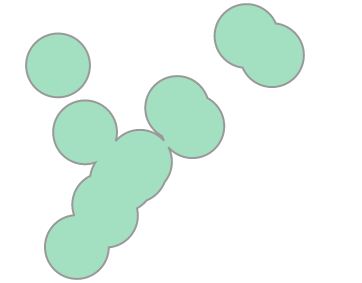
my\_union = two\_mile\_buffer.geometry.unary\_union

print('Type:', type(my\_union))

*# Show the MultiPolygon object*

my\_union

Type: <class 'shapely.geometry.multipolygon.MultiPolygon'>



We use the contains() method to check if the multipolygon contains a point. We'll use the release incident from earlier in the tutorial, which we know is roughly 3,781 feet to the closest monitoring station.

*# The closest station is less than two miles away*

my\_union.contains(releases.iloc[360].geometry)

True

But not all releases occured within two miles of an air monitoring station!

*# The closest station is more than two miles away*

my\_union.contains(releases.iloc[358].geometry)

False