Geospatial Analysis Home Page (https://www.kaggle.com/learn/geospatial-analysis)

Introduction

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, and
- select all points within some radius of a feature.

Code

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

In [2]:

```
releases = gpd.read_file("../input/geospatial-learn-course-data/toxic_release_pennsylvania/toxic_release_pennsylvania/toxic_release_pennsylvania.shp")
releases.head()
```

Out[2]:

	YEAR	CITY	COUNTY	ST	LATITUDE	LONGITUDE	CHEMICAL	U
0	2016	PHILADELPHIA	PHILADELPHIA	PA	40.005901	-75.072103	FORMIC ACID	P
1	2016	PHILADELPHIA	PHILADELPHIA	PA	39.920120	-75.146410	ETHYLENE GLYCOL	P
2	2016	PHILADELPHIA	PHILADELPHIA	PA	40.023880	-75.220450	CERTAIN GLYCOL ETHERS	P
3	2016	PHILADELPHIA	PHILADELPHIA	PA	39.913540	-75.198890	LEAD COMPOUNDS	P
4	2016	PHILADELPHIA	PHILADELPHIA	PA	39.913540	-75.198890	BENZENE	P

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

In [3]:

```
stations = gpd.read_file("../input/geospatial-learn-course-data/PhillyHealth_Air_Monitoring_Stations/PhillyHealth_Air_Monitoring_Stations/PhillyHealth_Air_Monitoring_Stations.shp")
stations.head()
```

Out[3]:

	SITE_NAME	ADDRESS	BLACK_CARB	ULTRAFINE_	CO	SO2	OZONE	NO2	NOY_NC
0	LAB	1501 East Lycoming Avenue	N	N	Y	N	Y	Y	Y
1	ROX	Eva and Dearnley Streets	N	N	N	N	N	N	N
2	NEA	Grant Avenue and Ashton Street	N	N	N	N	Y	N	N
3	CHS	500 South Broad Street	N	N	N	N	N	N	N
4	NEW	2861 Lewis Street	N	N	Y	Y	Y	N	Y

5 rows × 24 columns

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.

In [4]:

```
print(stations.crs)
print(releases.crs)
```

```
{'init': 'epsg:2272'}
{'init': 'epsg:2272'}
```

We also check the CRS to see which units it uses (meters, feet, or something else). In this case, EPSG 2272 has units of feet. (*If you like, you can check this here (https://epsg.io/2272).*)

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.

In [5]:

```
# 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
```

Out[5]:

```
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.

In [6]:

```
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.

In [7]:

```
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                     39.9128
LONGITUDE                   -75.1854
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*5280 feet) around a different air monitoring station.

In [8]:

```
two_mile_buffer = stations.geometry.buffer(2*5280)
two_mile_buffer.head()
```

Out[8]:

```
0    POLYGON ((2721944.640797138 257149.3104284704,...
1    POLYGON ((2682494.289907977 271248.9000113755,...
2    POLYGON ((2744886.638220146 280980.2466903776,...
3    POLYGON ((2703638.579968393 233247.1013432145,...
4    POLYGON ((2726959.772827223 251134.9763285518,...
dtype: object
```

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.

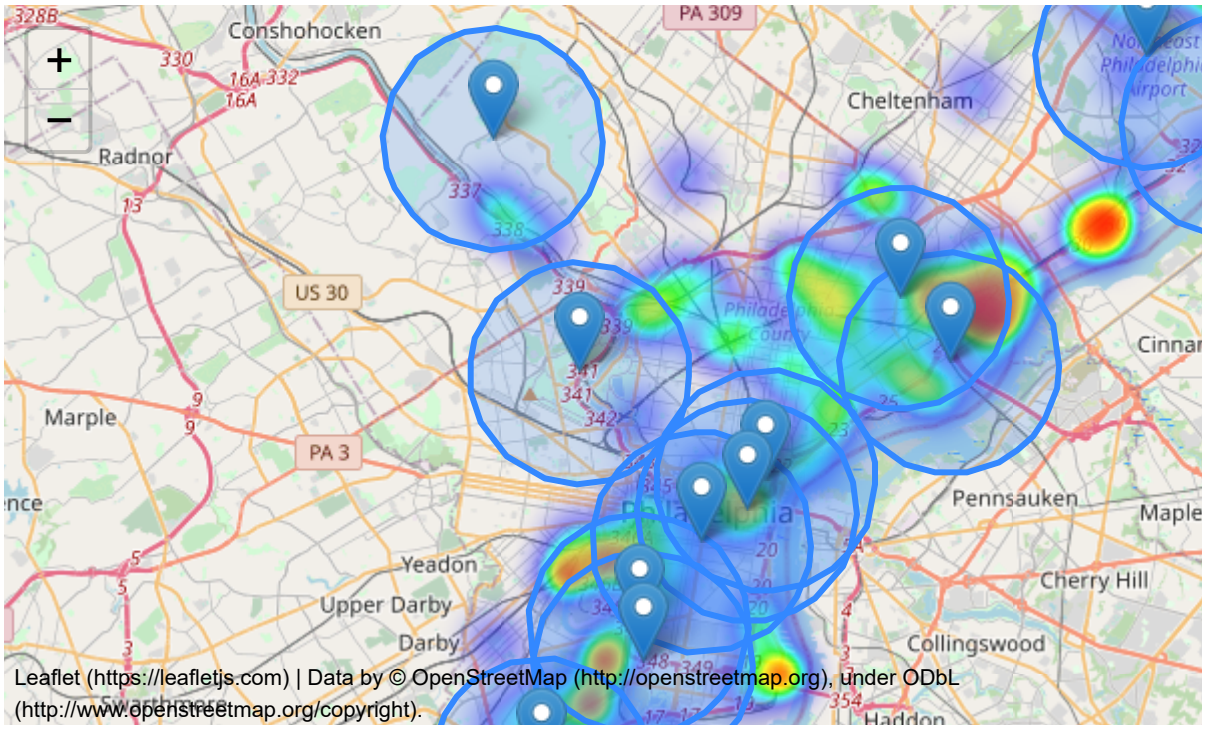
In [9]:

```
# 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
```

Out[9]:



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.

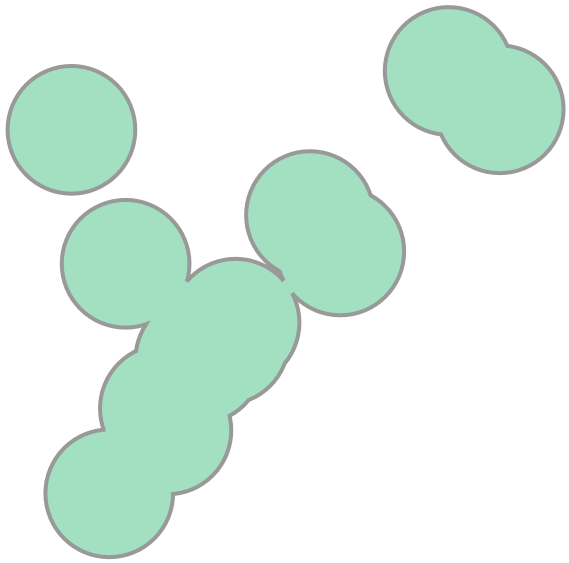
In [10]:

```
# 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'>
```

Out[10]:



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 3781 feet to the closest monitoring station.

In [11]:

```
# The closest station is less than two miles away
my_union.contains(releases.iloc[360].geometry)
```

Out[11]:

```
True
```

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

In [12]:

```
# The closest station is more than two miles away
my_union.contains(releases.iloc[358].geometry)
```

Out[12]:

```
False
```

Your turn

In the **final exercise (https://www.kaggle.com/kernels/fork/5832147)**, you'll investigate hospital coverage in New York City.

Geospatial Analysis Home Page (https://www.kaggle.com/learn/geospatial-analysis)

Have questions or comments? Visit the Learn Discussion forum (https://www.kaggle.com/learn-forum/161464) to chat with other Learners.