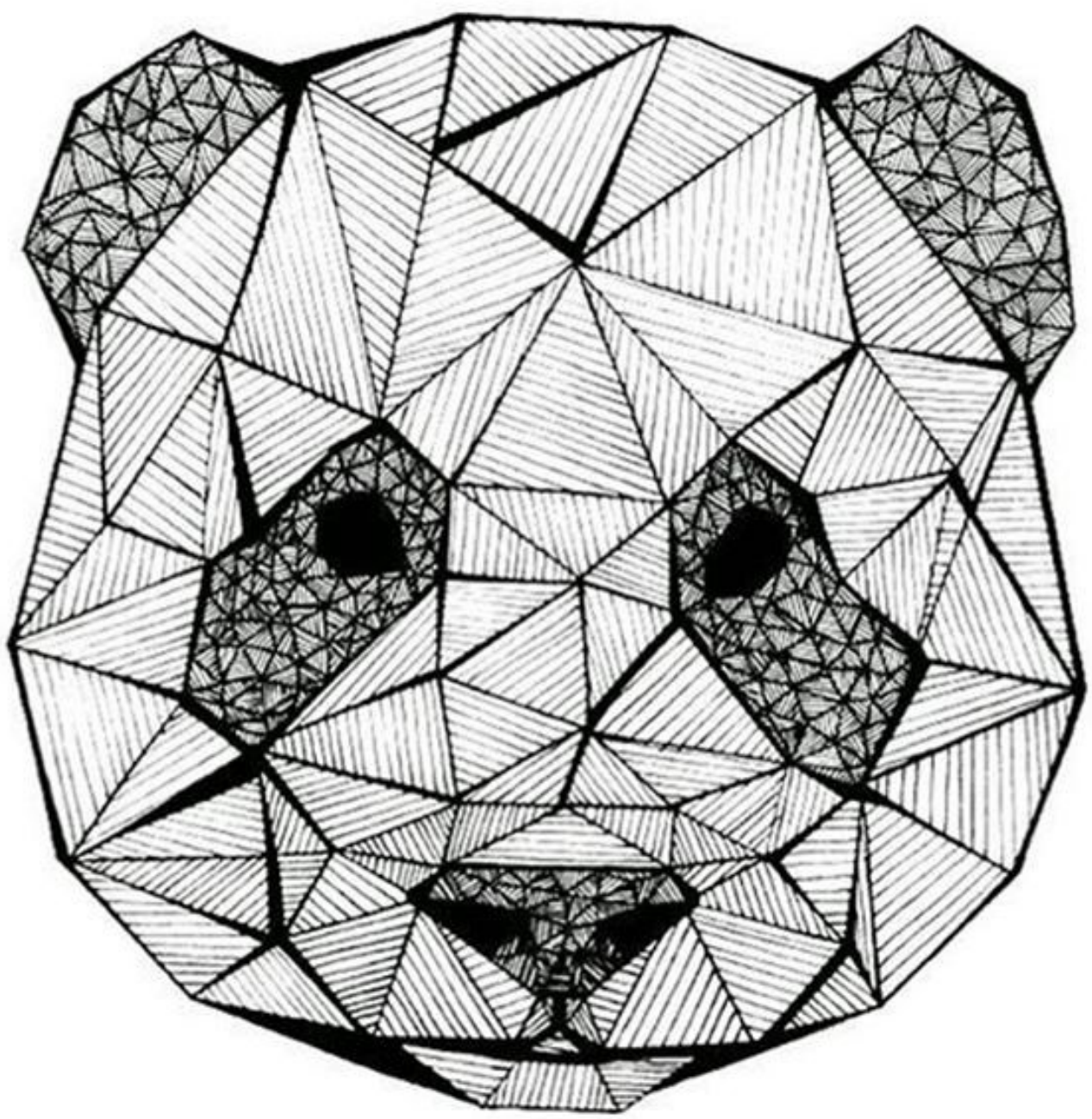
# GeoPandas 101: Plot any data with a latitude and longitude on a map

Ryan Stewart

Nov 1, 2018 · 4 min read



I was inspired to write this tutorial after a few of my classmates asked me how to use GeoPandas(commonly imported under the alias gpd). I originally learned from an article on Medium but the link seems to have stopped working. Alas, here we are. While I cannot possibly share everything about the wonderful world of GeoPandas in a blog post, my intention is to give you a starting point. I strongly encourage you to look at the official documentation, even if just to see all the cool things GeoPandas is capable of. Hopefully you find this tutorial helpful and exciting! All of the relevant data and notebooks can be found on my GitHub page here.

In my opinion, GeoPandas is one of the most satisfying Python packages to use because it produces a tangible, visible output that is directly linked to the real world. Moreover, a lot of the objects we would collect data on (e.g., demographic data, sales metrics, sensor data) have at least one physical element that can help us tie data to a specific location and describe something about the object.

. . .

In this example, we'll use data from the city of Chicago to plot the locations of mosquitoes with West Nile Virus. The first step is to download a shape-file(.shp file) of that area if you know the general area in which your geo-spatial data exists. If you don't know where to find a shape-file, Google it! There are often a couple of other files that come with the shape-file; make sure to keep all of those files in the same folder together, or you won't be able to read in your file.

The next step is to import your libraries. We will need the following, all of which can be installed quickly from the command line with a quick "pip install <package_name>":

```
import pandas as pd
import matplotlib.pyplot as plt
import descartes
import geopandas as gpd
from shapely.geometry import Point, Polygon

%matplotlib inline
```

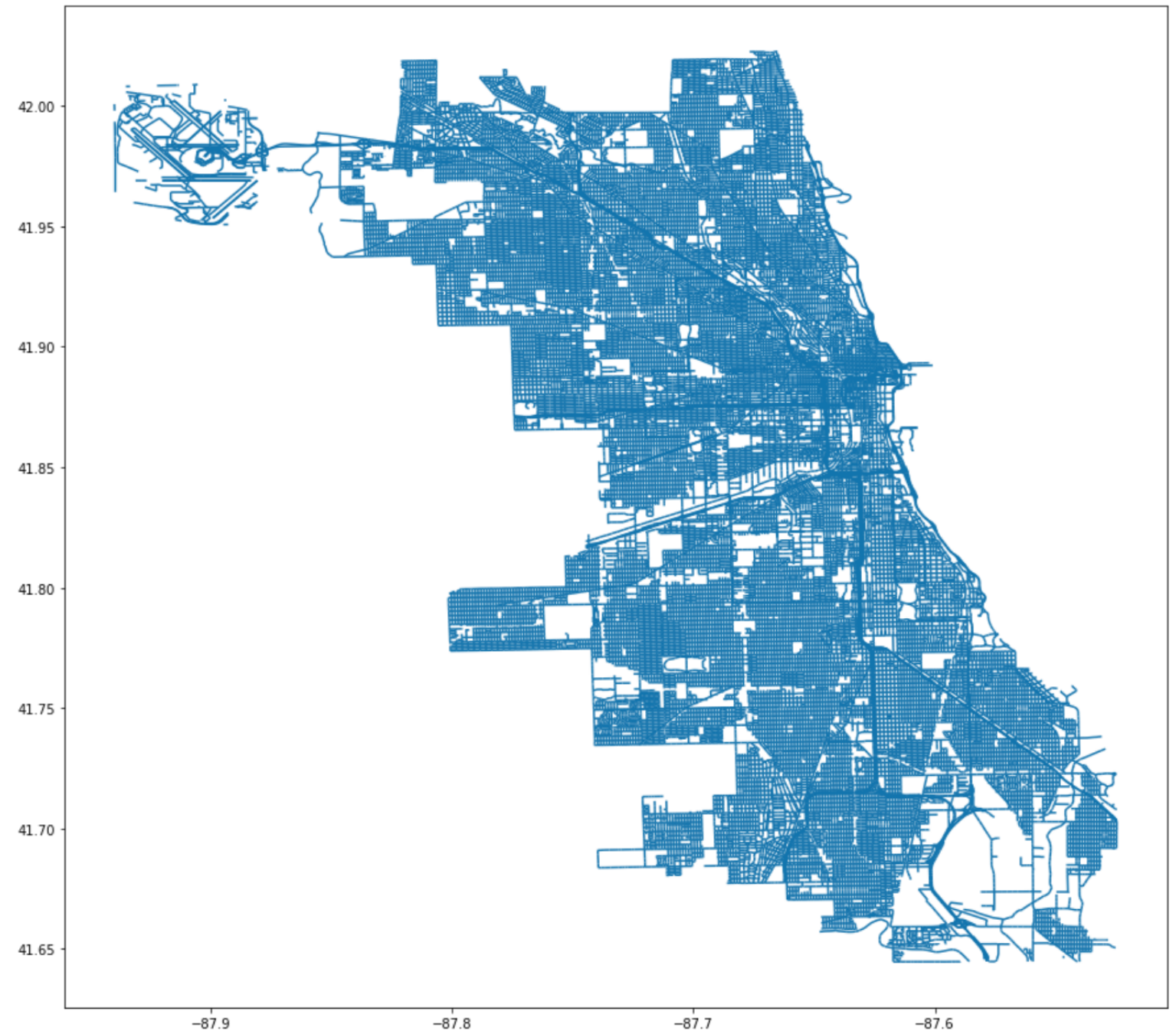
Now that we have a shape-file and the requisite packages, let's plot our map! Simply

read in your shape-file with GeoPandas and use matplotlib to plot it like so:

```
street_map = gpd.read_file('/home/ryan/Downloads/geo_export_1525534b-b713-44d0-abb6-b80b5d009726.shp')
```

```
fig,ax = plt.subplots(figsize = (15,15))
street_map.plot(ax = ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c9a58f2b0>
```




Notice that our map shows the street center-lines; you can find all different kinds of shape-files with varying levels of detail, so choose the type that best fits your specific needs.

The next step is to get the data in the right format. The way we do this is by turning our regular Pandas DataFrame into a geo-DataFrame, which will require us to specify as parameters the original DataFrame, our coordinate reference system (CRS), and the geometry of our new DataFrame. In order to format our geometry appropriately, we will need to convert the longitude and latitude into *Points* (we imported Point from shapely above), so first let's read in the training data-set and

specify the *EPSG:4326* CRS like so:

```
df = pd.read_csv('./kaggle_data/train.csv')
crs = {'init': 'epsg:4326'}
df.head()
```

| Date | Address | Species | Block | Street | Trap | AddressNumberAndStreet | Latitude | Longitude | AddressAccuracy | NumMosquitos | WnvPresent |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2007-05-29 | 4100 North Oak Park Avenue, Chicago, IL 60634,... | CULEX PIPIENS/RESTUANS | 41 | N OAK PARK AVE | T002 | 4100 N OAK PARK AVE, Chicago, IL | 41.954690 | -87.800991 | 9 | 1 | 0 |
| 2007-05-29 | 4100 North Oak Park Avenue, Chicago, IL 60634,... | CULEX RESTUANS | 41 | N OAK PARK AVE | T002 | 4100 N OAK PARK AVE, Chicago, IL | 41.954690 | -87.800991 | 9 | 1 | 0 |
| 2007-05-29 | 6200 North Mandell Avenue, Chicago, IL 60646, USA | CULEX RESTUANS | 62 | N MANDELL AVE | T007 | 6200 N MANDELL AVE, Chicago, IL | 41.994991 | -87.769279 | 9 | 1 | 0 |
| 2007-05-29 | 7900 West Foster Avenue, Chicago, IL | CULEX PIPIENS/RESTUANS | 79 | W FOSTER AVE | T015 | 7900 W FOSTER AVE, Chicago, IL | 41.974089 | -87.824812 | 8 | 1 | 0 |

Looking at the first few rows, we can see Latitude and Longitude are among the features that describe our data-points.

Now that we have latitude and longitude information we can create *Points*. A *Point* is essentially a single object that describes the longitude and latitude of a data-point. Using list comprehension will allow us to do this in one line, but make sure to always specify the “Longitude” column before the “Latitude” column:

```
geometry = [Point(xy) for xy in zip( df["Longitude"], df["Latitude"])]
geometry[:3]
```

```
[<shapely.geometry.point.Point at 0x7efcb51a1860>,
 <shapely.geometry.point.Point at 0x7efcb51a1748>,
 <shapely.geometry.point.Point at 0x7efcb51a1710>]
```

One point for each row in our DataFrame. Check the length of the geometry list if you want to make sure.

We should end up with a list of *Points* that we can use to create our GeoDataFrame:

```
geo_df = gpd.GeoDataFrame(df, #specify our data
                          crs = crs,#specify our coordinate reference system
                          geometry = geometry) #specify the geometry list we created
geo_df.head()
```

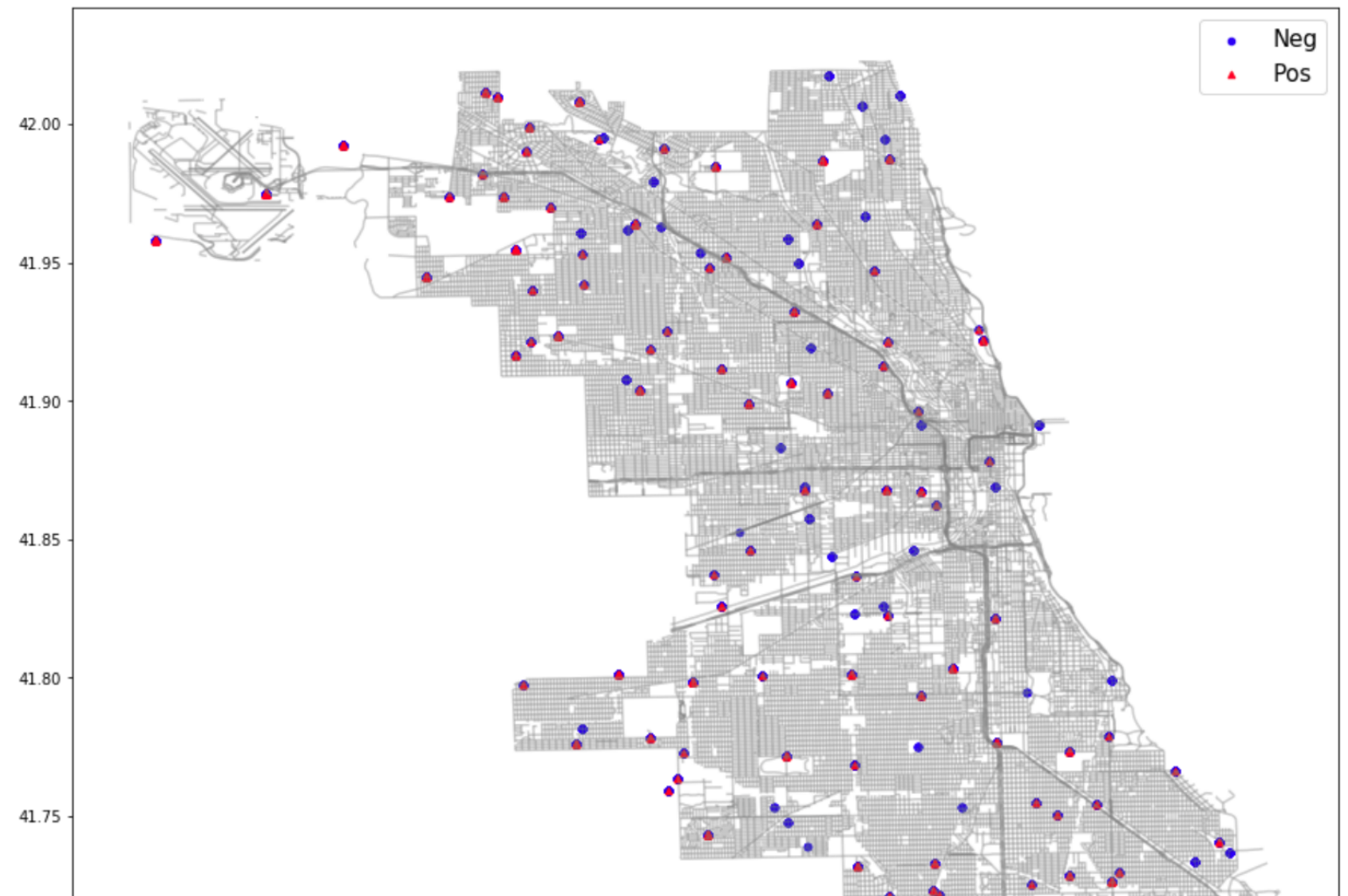
| | Species | Block | Street | Trap | AddressNumberAndStreet | Latitude | Longitude | AddressAccuracy | NumMosquitos | WnvPresent | geometry |
|---|---|---|---|---|---|---|---|---|---|---|---|

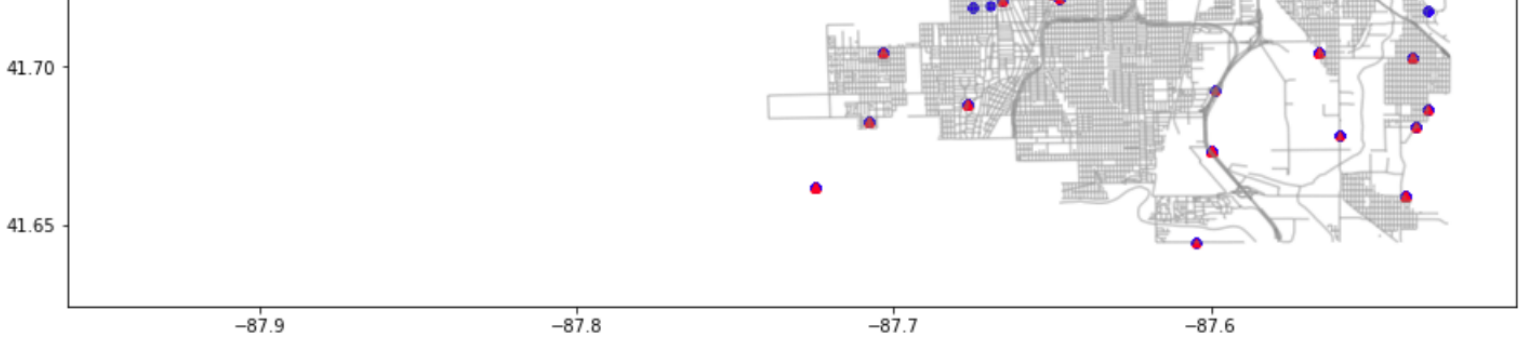
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| CULEX PIENS/RESTUANS | 41 | N OAK PARK AVE | T002 | 4100 N OAK PARK AVE, Chicago, IL | 41.954690 | -87.800991 | 9 | 1 | 0 | POINT (-87.800991 41.95469) |
| CULEX RESTUANS | 41 | N OAK PARK AVE | T002 | 4100 N OAK PARK AVE, Chicago, IL | 41.954690 | -87.800991 | 9 | 1 | 0 | POINT (-87.800991 41.95469) |
| CULEX RESTUANS | 62 | N MANDELL AVE | T007 | 6200 N MANDELL AVE, Chicago, IL | 41.994991 | -87.769279 | 9 | 1 | 0 | POINT (-87.769279 41.994991) |
| CULEX PIENS/RESTUANS | 79 | W FOSTER AVE | T015 | 7900 W FOSTER AVE, Chicago, IL | 41.974089 | -87.824812 | 8 | 1 | 0 | POINT (-87.82481199999999 41.974089) |

At this point, you may drop the “Latitude” and “Longitude” columns if you wish, but GeoPandas will automatically reference the “geometry” column when you plot your data. To do so, we simply layer our data onto the map we plotted above. I’m going to change some display options so that our map is easier to see, use masking to plot WNV+ and WNV- mosquitoes with different markers, and add labels and a legend:

```
fig,ax = plt.subplots(figsize = (15,15))
street_map.plot(ax = ax, alpha = 0.4, color="grey")
geo_df[geo_df['WnvPresent'] == 0].plot(ax = ax, markersize = 20, color = "blue", marker = "o", label = "Neg")
geo_df[geo_df['WnvPresent'] == 1].plot(ax = ax, markersize = 20, color = "red", marker = "^", label = "Pos")
plt.legend(prop={'size': 15})
```

```
<matplotlib.legend.Legend at 0x7efcaa5c47f0>
```






You've done it! There are tons of options and awesome packages to play around with and use in conjunction with GeoPandas to make a map pop. If you want to dig deeper, I'd suggest you start with Bokeh. While it's not as easy as plotting with GeoPandas alone, it can add some impressive interactivity features to your geo-maps and is a lot of fun to use!

Hopefully by now, you feel comfortable making simple plots with GeoPandas. They're great for exploratory analysis, and for conveying spatial information quickly and intuitively. Feel free to reach out if you have questions, or send me some examples of your GeoPandas creations!