Geospatial Analysis: 4th lesson – Manipulating Geospatial Data

In this tutorial, you'll learn about two common manipulations for geospatial data: **geocoding** and **table joins**.

import pandas as pd

import geopandas as gpd

import numpy as np

import folium

from folium import Marker

import warnings

warnings.filterwarnings('ignore')

/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

Geocoding:

Geocoding is the process of converting the name of a place or an address to a location on a map. If you have ever looked up a geographic location based on a landmark description with **Google Maps**, **Bing Maps**, or **Baidu Maps**, for instance, then you have used a geocoder!



We'll use geopy to do all of our geocoding.

from geopy.geocoders import Nominatim

In the code cell above, Nominatim refers to the geocoding software that will be used to generate locations. We begin by instantiating the geocoder. Then, we need only apply the name or address as a Python string (in this case, we supply "Pyramid of Khufu", also known as the Great Pyramid of Giza).

If the geocoding is successful, it returns a geopy.location.Location object with two important attributes:

* the "point" attribute contains the (latitude, longitude) location, and
* the "address" attribute contains the full address

geolocator = Nominatim(user\_agent="kaggle\_learn")

location = geolocator.geocode("Pyramid of Khufu")

print(location.point)

print(location.address)

29 58m 44.976s N, 31 8m 3.17625s E

هرم خوفو, شارع ابو الهول السياحي, نزلة البطران, الجيزة, 12125, مصر

The value for the "point" attribute is a geopy.point.Point object, and we can get the latitude and longitude from the latitude and longitude attributes, respectively.

point = location.point

print("Latitude:", point.latitude)

print("Longitude:", point.longitude)

Latitude: 29.97916

Longitude: 31.134215625236113

It's often the case that we'll need to geocode many different addresses. For instance, say we want to obtain the locations of 100 top universities in Europe.

universities = pd.read\_csv("../input/geospatial-learn-course-data/top\_universities.csv")

universities.head()

Name

0 University of Oxford

1 University of Cambridge

2 Imperial College London

3 ETH Zurich

4 UCL

Then we can use a lambda function to apply the geocoder to every row in the DataFrame (we use a try/except statement to account for the case that the geocoding is unsuccessful).

def my\_geocoder(row):

try:

point = geolocator.geocode(row).point

return pd.Series({'Latitude': point.latitude, 'Longitude': point.longitude})

except:

return None

universities[['Latitude', 'Longitude']] = universities.apply(lambda x: my\_geocoder(x['Name']), axis=1)

print("**{}% o**f addresses were geocoded!".format(

(1 - sum(np.isnan(universities["Latitude"])) / len(universities)) \* 100))

# Drop universities that were not successfully geocoded

universities = universities.loc[~np.isnan(universities["Latitude"])]

universities = gpd.GeoDataFrame(

universities, geometry=gpd.points\_from\_xy(universities.Longitude, universities.Latitude))

universities.crs = {'init': 'epsg:4326'}

universities.head()

91.0% of addresses were geocoded!

Name Latitude Longitude geometry

0 University of Oxford 51.758879 -1.259603 POINT (-1.25960 51.75888)

1 University of Cambridge 52.200623 0.110474 POINT (0.11047 52.20062)

2 Imperial College London 51.498959 -0.175641 POINT (-0.17564 51.49896)

3 ETH Zurich 47.562772 7.580947 POINT (7.58095 47.56277)

4 UCL 51.521785 -0.135151 POINT (-0.13515 51.52179)

Next, we visualize all of the locations that were returned by the geocoder. Notice that a few of the locations are certainly inaccurate, as they're not in Europe!

*# Create a map*

m = folium.Map(location=[54, 15], tiles='openstreetmap', zoom\_start=2)

*# Add points to the map*

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

Marker([row['Latitude'], row['Longitude']], popup=row['Name']).add\_to(m)

*# Display the map*

m

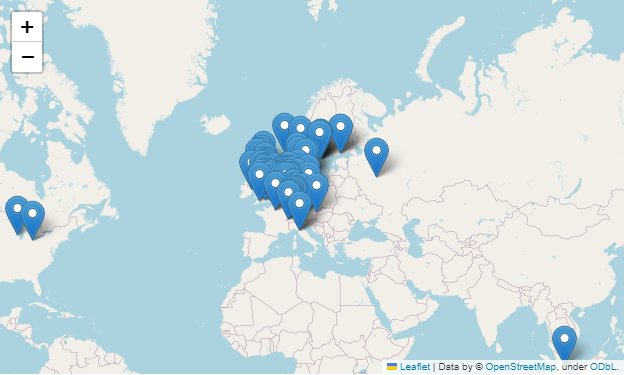


Table joins:

Now, we'll switch topics and think about how to combine data from different sources.

* Attribute join

You already know how to use pd.DataFrame.join() to combine information from multiple DataFrames with a shared index. We refer to this way of joining data (by simpling matching values in the index) as an attribute join. When performing an attribute join with a GeoDataFrame, it's best to use the gpd.GeoDataFrame.merge(). To illustrate this, we'll work with a GeoDataFrame europe\_boundaries containing the boundaries for every country in Europe. The first five rows of this GeoDataFrame are printed below.

world = gpd.read\_file(gpd.datasets.get\_path('naturalearth\_lowres'))

europe = world.loc[world.continent == 'Europe'].reset\_index(drop=True)

europe\_stats = europe[["name", "pop\_est", "gdp\_md\_est"]]

europe\_boundaries = europe[["name", "geometry"]]

europe\_boundaries.head()

name geometry

0 Russia MULTIPOLYGON (((178.72530 71.09880, 180.00000 ...

1 Norway MULTIPOLYGON (((15.14282 79.67431, 15.52255 80...

2 France MULTIPOLYGON (((-51.65780 4.15623, -52.24934 3...

3 Sweden POLYGON ((11.02737 58.85615, 11.46827 59.43239...

4 Belarus POLYGON ((28.17671 56.16913, 29.22951 55.91834...

We'll join it with a DataFrame europe\_stats containing the estimated population and gross domestic product (GDP) for each country.

europe\_stats.head()

name pop\_est gdp\_md\_est

0 Russia 142257519 3745000.0

1 Norway 5320045 364700.0

2 France 67106161 2699000.0

3 Sweden 9960487 498100.0

4 Belarus 9549747 165400.0

We do the attribute join in the code cell below. The on argument is set to the column name that is used to match rows in europe\_boundaries to rows in europe\_stats.

*# Use an attribute join to merge data about countries in Europe*

europe = europe\_boundaries.merge(europe\_stats, on="name")

europe.head()

name geometry pop\_est gdp\_md\_est

0 Russia MULTIPOLYGON (((178.72530 71.09880, 180.00000 ... 142257519 3745000.0

1 Norway MULTIPOLYGON (((15.14282 79.67431, 15.52255 80... 5320045 364700.0

2 France MULTIPOLYGON (((-51.65780 4.15623, -52.24934 3... 67106161 2699000.0

3 Sweden POLYGON ((11.02737 58.85615, 11.46827 59.43239... 9960487 498100.0

4 Belarus POLYGON ((28.17671 56.16913, 29.22951 55.91834... 9549747 165400.0

* Spatial join

Another type of join is a spatial join. With a spatial join, we combine GeoDataFrames based on the spatial relationship between the objects in the "geometry" columns. For instance, we already have a GeoDataFrame universities containing geocoded addresses of European universities. Then we can use a spatial join to match each university to its corresponding country. We do this with gpd.sjoin().

*# Use spatial join to match universities to countries in Europe*

european\_universities = gpd.sjoin(universities, europe)

*# Investigate the result*

print("We located **{}** universities.".format(len(universities)))

print("Only **{}** of the universities were located in Europe (in **{}** different countries).".format(

len(european\_universities), len(european\_universities.name.unique())))

european\_universities.head()

We located 91 universities.

Only 87 of the universities were located in Europe (in 14 different countries).

Name Latitude Longitude geometry index\_right name pop\_est gdp\_md\_est

0 University of Oxford 51.758879 -1.259603 POINT (-1.25960 51.75888) 28 United Kingdom 64769452 2788000.0

1 University of Cambridge 52.200623 0.110474 POINT (0.11047 52.20062) 28 United Kingdom 64769452 2788000.0

2 Imperial College London 51.498959 -0.175641 POINT (-0.17564 51.49896) 28 United Kingdom 64769452 2788000.0

4 UCL 51.521785 -0.135151 POINT (-0.13515 51.52179) 28 United Kingdom 64769452 2788000.0

5 London School of Economics and Political Science 51.514211 -0.116808 POINT (-0.11681 51.51421) 28 United Kingdom 64769452 2788000.0

The spatial join above looks at the "geometry" columns in both GeoDataFrames. If a Point object from the universities GeoDataFrame intersects a Polygon object from the europe DataFrame, the corresponding rows are combined and added as a single row of the european\_universities DataFrame. Otherwise, countries without a matching university (and universities without a matching country) are omitted from the results.

The gpd.sjoin() method is customizable for different types of joins, through the how and op arguments. For instance, you can do the equivalent of a SQL left (or right) join by setting how='left' (or how='right').