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

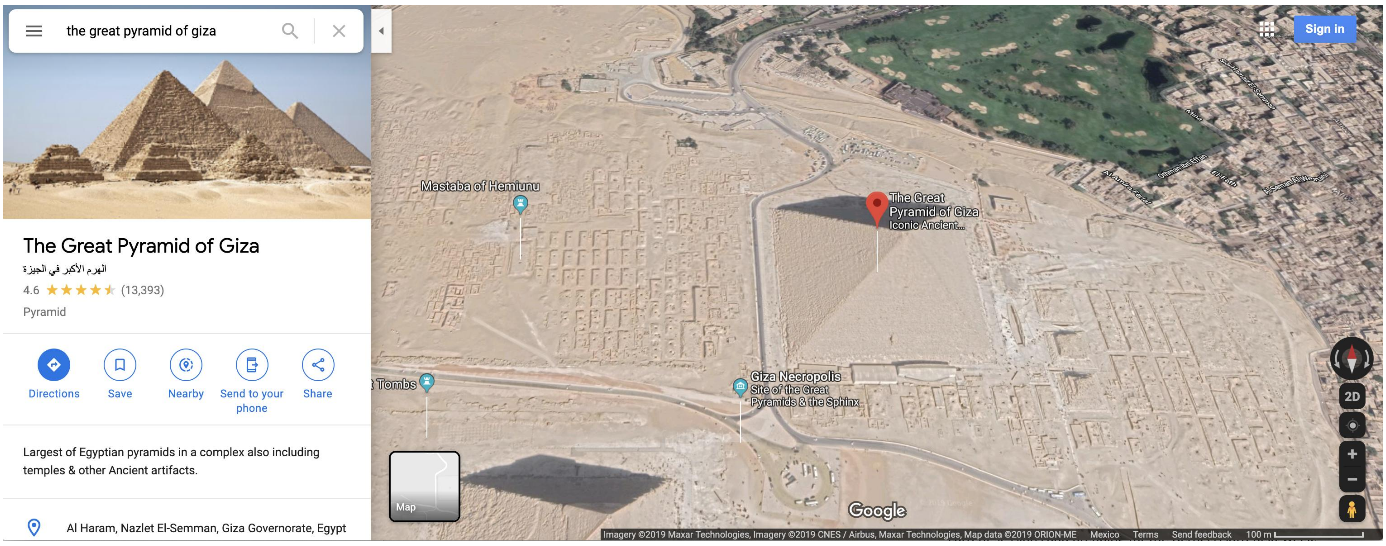
Introduction

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

Code

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 (https://www.google.com/maps), Bing Maps (https://www.bing.com/maps), or Baidu Maps (https://map.baidu.com/), for instance, then you have used a geocoder!



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

In [2]:

```
from geopandas.tools import geocode
```

To use the geocoder, we need only provide:

- the name or address as a Python string, and
- the name of the provider; to avoid having to provide an API key, we'll use the OpenStreetMap Nominatim geocoder (https://nominatim.openstreetmap.org/).

If the geocoding is successful, it returns a GeoDataFrame with two columns:

- the "geometry" column contains the (latitude, longitude) location, and
- the "address" column contains the full address.

In [3]:

```
result = geocode("The Great Pyramid of Giza", provider="nominatim")
result
```

Out[3]:

	geometry	address
0	POINT (31.13424 29.97913)	هرم خوفو, Cause way, كوم الأخضر, الجيزة, محافظ...

The entry in the "geometry" column is a `Point` object, and we can get the latitude and longitude from the `y` and `x` attributes, respectively.

In [4]:

```
point = result.geometry.iloc[0]
print("Latitude:", point.y)
print("Longitude:", point.x)
```

```
Latitude: 29.9791264
Longitude: 31.13423837510151
```

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.

In [5]:

```
universities = pd.read_csv("../input/geospatial-learn-course-data/top_universiti
es.csv")
universities.head()
```

Out[5]:

	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.)

In [6]:

```
def my_geocoder(row):
    try:
        point = geocode(row, provider='nominatim').geometry.iloc[0]
        return pd.Series({'Latitude': point.y, 'Longitude': point.x, 'geometry':
point})
    except:
        return None

universities[['Latitude', 'Longitude', 'geometry']] = universities.apply(lambda
x: my_geocoder(x['Name']), axis=1)

print("{}% of 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=universities.geometry)
universities.crs = {'init': 'epsg:4326'}
universities.head()
```

```
90.0% of addresses were geocoded!
```

Out[6]:

	Name	Latitude	Longitude	geometry
0	University of Oxford	51.758708	-1.255668	POINT (-1.25567 51.75871)
1	University of Cambridge	52.199852	0.119739	POINT (0.11974 52.19985)
2	Imperial College London	51.498871	-0.175608	POINT (-0.17561 51.49887)
3	ETH Zurich	47.377327	8.547509	POINT (8.54751 47.37733)
4	UCL	51.524444	-0.133512	POINT (-0.13351 51.52444)

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!

In [7]:

```
# 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
```

Out[7]:

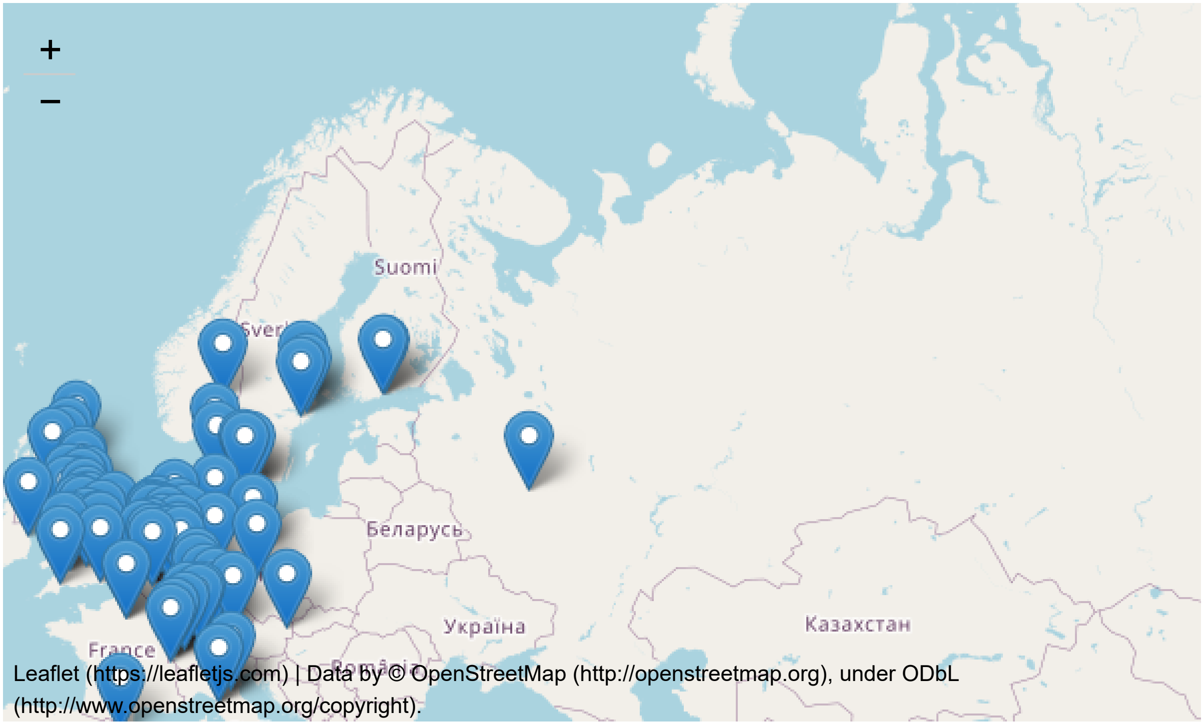


Table joins

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

Attribute join

You already know (https://www.kaggle.com/residentmario/renaming-and-combining) 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.

Code

In [9]:

```
europe_boundaries.head()
```

Out[9]:

	name	geometry
0	Russia	MULTIPOLYGON (((178.725 71.099, 180.000 71.516...
1	Norway	MULTIPOLYGON (((15.143 79.674, 15.523 80.016, ...
2	France	MULTIPOLYGON (((-51.658 4.156, -52.249 3.241, ...
3	Sweden	POLYGON ((11.027 58.856, 11.468 59.432, 12.300...
4	Belarus	POLYGON ((28.177 56.169, 29.230 55.918, 29.372...

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

In [10]:

```
europe_stats.head()
```

Out[10]:

	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`.

In [11]:

```
# Use an attribute join to merge data about countries in Europe
europe = europe_boundaries.merge(europe_stats, on="name")
europe.head()
```

Out[11]:

	name	geometry	pop_est	gdp_md_est
0	Russia	MULTIPOLYGON (((178.725 71.099, 180.000 71.516...	142257519	3745000.0
1	Norway	MULTIPOLYGON (((15.143 79.674, 15.523 80.016, ...	5320045	364700.0
2	France	MULTIPOLYGON (((-51.658 4.156, -52.249 3.241, ...	67106161	2699000.0
3	Sweden	POLYGON ((11.027 58.856, 11.468 59.432, 12.300...	9960487	498100.0
4	Belarus	POLYGON ((28.177 56.169, 29.230 55.918, 29.372...	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()`.

In [12]:

```python
# 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 count
ries).".format(
    len(european_universities), len(european_universities.name.unique())))

european_universities.head()
```

```
We located 90 universities.
Only 85 of the universities were located in Europe (in 15 different
countries).
```

Out[12]:

	Name	Latitude	Longitude	geometry	index_right	name	pop_est	gc
0	University of Oxford	51.758708	-1.255668	POINT (-1.25567 51.75871)	28	United Kingdom	64769452	27
1	University of Cambridge	52.199852	0.119739	POINT (0.11974 52.19985)	28	United Kingdom	64769452	27
2	Imperial College London	51.498871	-0.175608	POINT (-0.17561 51.49887)	28	United Kingdom	64769452	27
4	UCL	51.524444	-0.133512	POINT (-0.13351 51.52444)	28	United Kingdom	64769452	27
5	London School of Economics and Political Science	51.514429	-0.116588	POINT (-0.11659 51.51443)	28	United Kingdom	64769452	27

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'`). We won't go into the details in this micro-course, but you can learn more in the documentation (http://geopandas.org/reference/geopandas.sjoin.html).

Your turn

Use geocoding and table joins (https://www.kaggle.com/kernels/fork/5832170) to identify suitable locations for the next Starbucks Reserve Roastery.

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.