Geopandas Hands-on: Introduction to Geospatial Machine Learning

Tutorial on how to deal with geospatial machine learning popular library, Geopandas

Juan Nathaniel Apr 21, 2021 · 4 min read ★

Part 1: Introduction to geospatial concepts (*this post*)
Part 2: Geospatial visualization and geometry creation (*follow here*)
Part 3: Geospatial operations (*follow here*)
Part 4: Building geospatial machine learning pipeline (*follow here*)

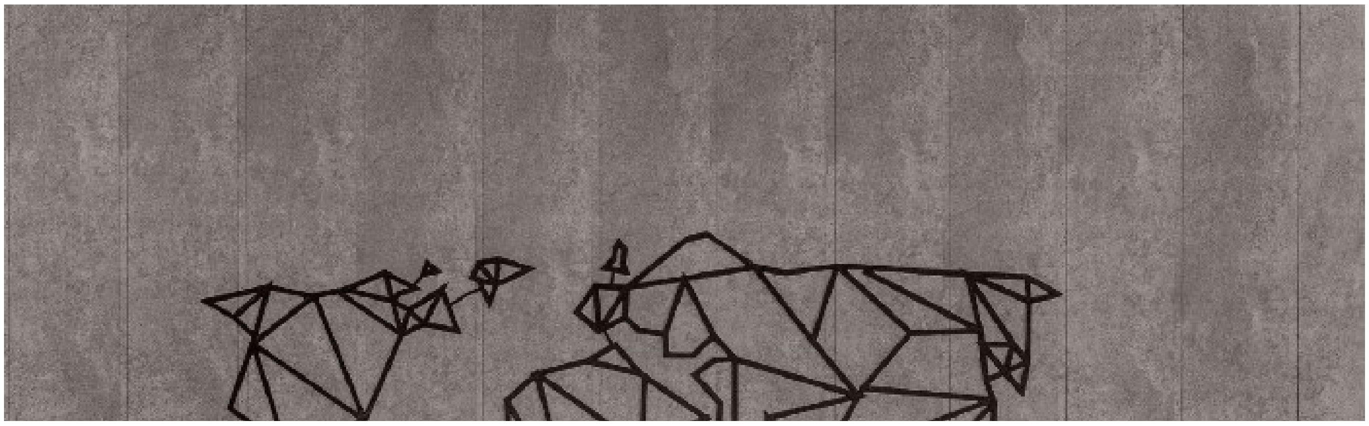

Photo by Marjan Blan | @marjanblan on Unsplash

In this post we are going to cover the preliminary ground of basic geospatial datatypes, attributes, and how to use geopandas to achieve these.

Table of Content:

What is Geopandas

Geopandas is open-sourced library and enables the use and manipulation of geospatial data in Python. It extends the common datatype used in pandas to allow for the many and unique geometric operations: GeoSeries and GeoDataFrame. Geopandas is also built on top of shapely for its geometric operation; its underlying datatype allows Geopandas to run blazingly fast and is appropriate for many machine learning pipelines that require large geospatial datasets.

Installation

Now that we are acquainted slightly with Geopandas, lets begin the installation process.

There are many options, but I would recommend installating using *conda* as it creates a new environment for your project without affecting other libraries in your system. To install conda, follow this link: installing conda.

1. Create a new conda environment and setup some configs

```
conda create -n geo_env
conda activate geo_env
conda config --env --add channels conda-forge
conda config --env --set channel_priority strict
```

2. Install with conda

```
conda install python=3 geopandas
```

Geospatial concepts

A. Geospatial common datatypes

There are some common geospatial datatypes that you need to be familiar with: Shapefile (.shp) and GeoJSON (.geojson).

Shapefile is a vector data format that is developed and maintained mostly by a company called ESRI. It stores many important geospatial information including the topology, shape geometry, etc.

Shapefile data format

GeoJSON, similar to JSON, stores *geometry* information (coordinates, projection, etc) in addition to your typical attributes relevant to the object (index, name, etc).

Example of GeoJSON format with the added “geometry” key inside the JSON object

Once you load either of these dataformat using Geopandas, the library will create a DataFrame with the additional *geometry* column.

GeoDataFrame (Source: geopandas.org)

This is how you import the default geodata built-in within the Geopandas library that we are going to use in this and subsequent posts.

```
import geopandas

path_to_data = geopandas.datasets.get_path("nybb")
gdf = geopandas.read_file(path_to_data)

gdf
```

Introduction to basic geometric attributes

Now that we have some ideas of geospatial data and how to import our very first one using Geopandas, lets perform some basic methods to further cement our understanding.

First, lets make the boroughs name as index to make our explorations easier.

```
gdf = gdf.set_index("BoroName")
```

AREA

From the geometry column, we can measure the areas (if they are of type POLYGON or MULTIPOLYGON: since we can't measure the area of lines or points)

```
gdf["area"] = gdf.area

gdf["area"]
>>BoroName
>>Staten Island    1.623822e+09
>>Queens           3.045214e+09
>>Brooklyn         1.937478e+09
>>Manhattan        6.364712e+08
>>Bronx            1.186926e+09
>>Name: area, dtype: float64
```

POLYGON BOUNDARY

Since our geometry is of type polygon or multipolygon, we can extract out the line coordinates of the objects. This can be useful when, say, we want to measure the perimeter of the polygon objects, etc.

Polygon Boundary Lines (Source: Wikipedia)

```
gdf['boundary'] = gdf.boundary

>>gdf['boundary']
>>BoroName
>>Staten Island    MULTILINESTRING ((970217.022 145643.332, 97022...
>>Queens           MULTILINESTRING ((1029606.077 156073.814, 1029...
>>Brooklyn         MULTILINESTRING ((1021176.479 151374.797, 1021...
>>Manhattan        MULTILINESTRING ((981219.056 188655.316, 98094...
>>Bronx            MULTILINESTRING ((1012821.806 229228.265, 1012...
>>Name: boundary, dtype: geometry
```

CENTROID

If you want to find the centroid point of the given polygons, you can call the gdf attribute as follows.

Polygon Centroids (source: Wikipedia)

```
gdf['centroid'] = gdf.centroid

>>gdf['centroid']
>>BoroName
>>Staten Island     POINT (941639.450 150931.991)
>>Queens           POINT (1034578.078 197116.604)
>>Brooklyn          POINT (998769.115 174169.761)
>>Manhattan         POINT (993336.965 222451.437)
```

```
>>Bronx            POINT (1021174.790 249937.980)
>>Name: centroid, dtype: geometry
```

DISTANCE

Now that wealready know the positions of the centroids and wanted to find out where the distance between Queens and everywhere else, this can be done easily using the distance() method:

```
queens_point = gdf['centroid'].iloc[1]
gdf['distance'] = gdf['centroid'].distance(queens_point)
```

You can then perform many spatial aggregates function to find out the mean, max, or min distances.

```
gdf['distance'].mean() #To find the mean distance
gdf['distance'].max() #To find the maximum distance
gdf['distance'].min() #To find the minimum distance
```

Conclusion

That's it! Hope you learn something new today. In the next post, we will dig deeper into how these geospatial data can be visualized and created from scratch. Stay tuned!

Do subscribe to my Email newsletter: https://tinyurl.com/2npw2fnz where I regularly summarize AI research papers in plain English and beautiful visualization.

Geospatial Data Science Artificial Intelligence Machine Learning Visualization