6 Geopandas

Lab Objective: *Geopandas is a package designed to organize and manipulate geographic data, It combines the data manipulation tools from Pandas and the geometric capabilities of the Shapely package. In this lab, we explore the basic data structures of GeoSeries and GeoDataFrames and their functionalities.*

Installation

Geopandas is a new package designed to combine the functionalities of Pandas and Shapely, a package used for geometric manipulation. Using Geopandas with geographic data is very useful as it allows the user to not only compare numerical data, but geometric attributes. Since Geopandas is currently under development, the installation procedure requires that all dependencies are up to date. While possible to install Geopandas through **pip** using

```
>>> pip install geopandas
```

it is not recommended. You can view the warnings here `https://geopandas.org/install.html`.

To install Geopandas through Conda, the recommended day, run the following code.

```
>>> conda install geopandas
>>> conda install -c conda-forge gdal
```

A particular package needed for Geopandas is Fiona. Geopandas will not run without the correct version of this package. To check the current version of Fiona that is installed, run the following code. If the version is not at least 1.7.13, update Fiona.

```
# Check version of Fiona
>>> conda list fiona

# Update Fiona if necessary
>>> pip install fiona --upgrade
```

GeoSeries

A GeoSeries is a Pandas Series where each entry is a set of geometric objects. There are three classes of geometric objects inherited from the Shapely package:

1. Points / Multi-Points
2. Lines / Multi-Lines
3. Polygons / Multi-Polygons

A point is used to identify objects like coordinates, where there is one small instance of the object. A line could be used to describe a road. A polygon could be used to identify regions, such as a country. Multipoints, multilines, and multipolygons contain lists of points, lines, and polygons, respectively.

Since each object in the GeoSeries is also a Shapely object, the GeoSeries inherits many methods and attributes of Shapely objects. Some of the key attributes and methods are listed in Table 6.1. These attributes and methods can be used to calculate distances, find the sizes of countries, and determine whether coordinates are within country's boundaries. The example below uses the attribute `bounds` to find the maximum and minimum coordinates of Egypt in a built-in GeoDataFrame.

Method/Attribute	Description
`distance(other)`	returns minimum distance from GeoSeries to `other`
`contains(other)`	returns `True` if shape contains `other`
`intersects(other)`	returns `True` if shape intersects `other`
`area`	returns shape area
`convex_hull`	returns convex shape around all points in the object

Table 6.1: Attributes and Methods for GeoSeries

```
>>> import geopandas as gpd
>>> world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
# Get GeoSeries for Egypt
>>> egypt = world[world['name']=='Egypt']

# Find bounds of Egypt
>>> egypt.bounds
         minx    miny         maxx      maxy
47   24.70007    22.0     36.86623  31.58568
```

Creating GeoDataFrames

The main structure used in GeoPandas is a GeoDataFrame, which is similar to a Pandas DataFrame. A GeoDataFrame has one special column called `geometry`. This GeoSeries column is used when a spatial method, like `distance()`, is used on the GeoDataFrame.

To make a GeoDataFrame, first create a Pandas DataFrame. At least one of the columns in the DataFrame should contain geometric information. Convert a column containing geometric information to a GeoSeries using the `apply` method. At this point, the Pandas DataFrame can be cast as a GeoDataFrame. When creating a GeoDataFrame, if more than one column has geometric data, assign which column will be the `geometry` using the `set_geometry()` method.

```
>>> import pandas as pd
>>> import geopandas as gpd
>>> from shapely.geometry import Point, Polygon

# Create a Pandas DataFrame
>>> df = pd.DataFrame({'City': ['Seoul', 'Lima', 'Johannesburg'],
...                    'Country': ['South Korea', 'Peru', 'South Africa'],
...                    'Latitude': [37.57, -12.05, -26.20],
...                    'Longitude': [126.98, -77.04, 28.04]})

# Create geometry column
>>> df['Coordinates'] = list(zip(df.Longitude, df.Latitude))

# Make geometry column Shapely objects
>>> df['Coordinates'] = df['Coordinates'].apply(Point)

# Cast as GeoDataFrame
>>> gdf = gpd.GeoDataFrame(df, geometry='Coordinates')

# Display the GeoDataFrame
>>> gdf
           City       Country  Latitude  Longitude                  Coordinates
0         Seoul   South Korea     37.57     126.98   POINT (126.98000 37.57000)
1          Lima          Peru    -12.05     -77.04  POINT (-77.04000 -12.05000)
2  Johannesburg  South Africa    -26.20      28.04   POINT (28.04000 -26.20000)

# Create a polygon with all three cities as points
>>> city_polygon = Polygon(list(zip(df.Longitude, df.Latitude)))
```

> **Note**
>
> Longitude is the angular measurement starting at the Prime Meridian, 0°, and going to 180° to the east and −180° to the west. Latitude is the angle between the equatorial plane and the normal line at a given point; a point along the Equator has latitude 0, the North Pole has latitude +90° or 90°N, and the South Pole has latitude −90° or 90°S.

Plotting GeoDataFrames

Information from a GeoDataFrame is plotted based on the geometry column. Data points are displayed as geometry objects. The following example plots the shapes in the `world` GeoDataFrame.

```
# Plot world GeoDataFrame
>>> world.plot()
```

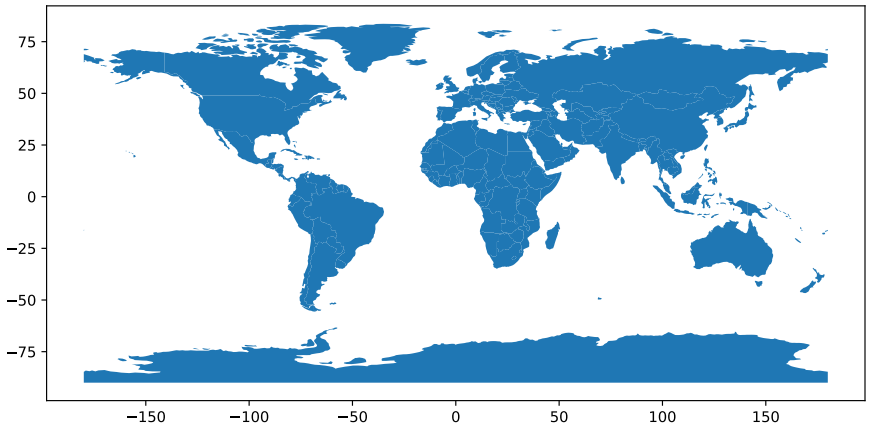

Figure 6.1: World map

Multiple GeoDataFrames can be plotted at once. This can be done by by setting one GeoDataFrame as the base of the plot and ensuring that each layer uses the same axes. In the following example, a GeoDataFrame containing the coordinates of world airports is plotted on top of a GeoDataFrame containing the polygons of country boundaries, resulting in a world map of airport locations.

```
# Set outline of world countries as base
>>> fig,ax = plt.subplots(figsize=(10,7), ncols=1, nrows=1)
>>> base = world.boundary.plot(edgecolor='black', ax=ax, linewidth=1)

# Plot airports on world map
>>> airports.plot(ax=base, marker='o', color='green', markersize=1)
>>> ax.set_xlabel('Longitude')
>>> ax.set_ylabel('Latitude')
>>> ax.set_title('World Airports')
```

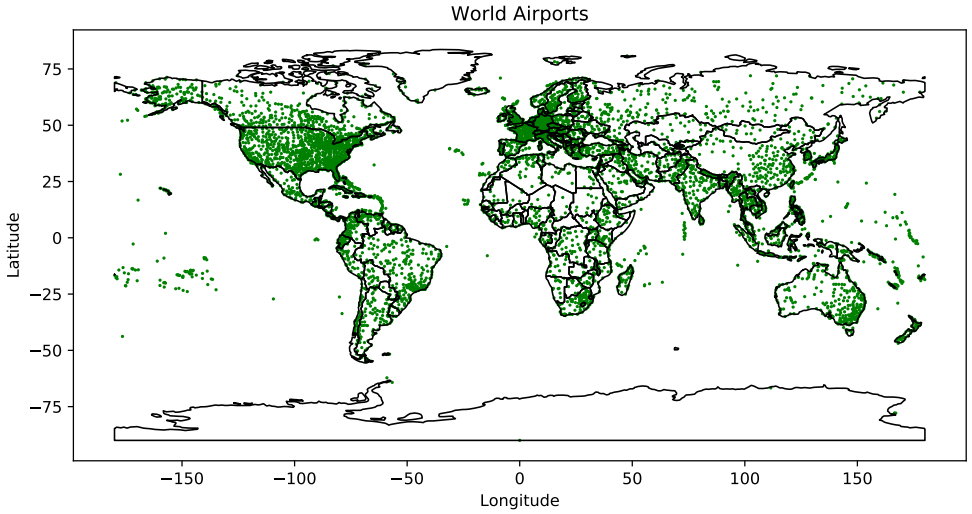

Figure 6.2: Airport map

Problem 1. Read in the file `airports.csv` as a Pandas DataFrame. Create three convex hulls around the three sets of airports listed below. This can be done by passing in lists of the airports' coordinates to a `shapely.geometry.Polygon` object.

Create a new GeoDataFrame with these three Polygons as entries. Plot this GeoDataFrame on top of an outlined world map.

- Maio Airport, Scatsta Airport, Stokmarknes Skagen Airport, Bekily Airport, K. D. Matanzima Airport, RAF Ascension Island
- Oiapoque Airport, Maio Airport, Zhezkazgan Airport, Walton Airport, RAF Ascension Island, Usiminas Airport, Piloto Osvaldo Marques Dias Airport
- Zhezkazgan Airport, Khanty Mansiysk Airport, Novy Urengoy Airport, Kalay Airport, Biju Patnaik Airport, Walton Airport

Working with GeoDataFrames

As previously mentioned, GeoDataFrames contain many of the functionalities of Pandas DataFrames. For example, to create a new column, define a new column name in the GeoDataFrame with the needed information for each GeoSeries.

```
# Create column in the world GeoDataFrame for gdp_per_capita
>>> world['gdp_per_cap'] = world.gdp_md_est / world.pop_est
```

GeoDataFrames can utilize many Pandas functionalities, and they can also be parsed by geometric manipulations. For example, a useful way to index GeoDataFrames is with the `cx` indexer. This splits the GeoDataFrame by the coordinates of each geometric object. It is used by calling the method `cx` on a GeoDataFrame, followed by a slicing argument, where the first element refers to the longitude and the second refers to latitude.

```
# Create a GeoDataFrame containing the northern hemisphere
>>> north = world.cx[:, 0:]

# Create a GeoDataFrame containing the southeastern hemisphere
>>> south_east = world.cx[0:, :0]
```

GeoSeries in a GeoDataFrame can also be dissolved, or merged, together into one GeoSeries based on their geometry data. For example, all countries on one continent could be merged to create a GeoSeries containing the information of that continent. The method designed for this is called `dissolve`. It receives two parameters, `by` and `aggfunc`. `by` indicates which column to dissolve along, and `aggfunc` tells how to combine the information in all other columns. The default `aggfunc` is `first`, which returns the first application entry. In the following example, we use `sum` as the `aggfunc` so that each continent is the combination of its countries.

```
>>> world = world[['continent', 'geometry', 'gdp_per_cap']]

# Dissolve world GeoDataFrame by continent
>>> continent = world.dissolve(by = 'continent', aggfunc='sum')
```

Projections

When plotting, GeoPandas uses the CRS (coordinate reference system) of a GeoDataFrame. This reference system informs how coordinates should be spaced on a plot. GeoPandas accepts many different CRSs, and references to them can be found at `www.spatialreference.org`. Two of the most commonly used CRSs are EPSG:4326 and EPSG:3395. EPSG:4326 uses the standard latitude-longitude projection used by GPS. EPSG:3395, also known as Mercator, is the standard navigational projection.

When creating a new GeoDataFrame, it is important to set the `crs` attribute of the GeoDataFrame. This allows the plot to be shown correctly. GeoDataFrames being layered need to have the same CRS. To change the CRS, use the method `to_crs()`.

```
# Check CRS of world GeoDataFrame
>>> print(world.crs)
epsg:4326

# Change CRS of world to Mercator
# inplace=True ensures that we modify world instead of returning a copy
>>> world.to_crs(3395, inplace=True)
>>> print(world.crs)
epsg:3395
```

GeoDataFrames can also be plotted using the values in the the other attributes of the GeoSeries. The map plots the color of each geometry object according to the value of the column selected. This is done by passing in the parameter `column` into the `plot()` method.

```
>>> fig, ax = plt.subplots(1, figsize=(10,4))
# Plot world based on gdp
```

```
>>> world.plot(column='gdp_md_est', cmap='OrRd', legend=True,  ax=ax)
>>> ax.set_title('World Map based on GDP')
>>> ax.set_xlabel('Longitude')
>>> ax.set_ylabel('Latitude')
>>> plt.show()
```

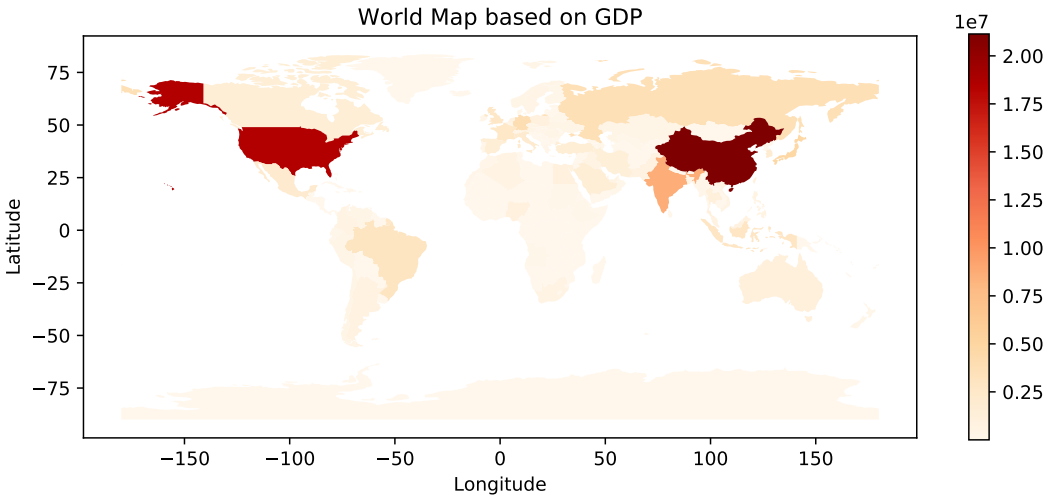

Figure 6.3: World Map Based on GDP

Problem 2. Use the command `geopandas.read_file('county_data.gpkg')` to create a GeoDataFrame of information about US counties.[a] Each county's shape is stored in the `geometry` column. Use this to plot all US counties two times, first using the default CRS and then using EPSG:5071.

Next, create a new GeoDataFrame that merges all counties within a single state. Drop regions with the following STATEFP codes: 02, 15, 60, 66, 69, 72, 78. Plot this GeoDataFrame to see an outline of all 48 contiguous states. Ensure a CRS of EPSG:5071.

[a] Source: `http://www2.census.gov/geo/tiger/GENZ2016/shp/cb_2016_us_county_5m.zip`

Merging GeoDataFrames

Just as multiple Pandas DataFrames can be merged, multiple GeoDataFrames can be merged with attribute joins or spatial joins. An attribute join is similar to a merge in Pandas. It combines two GeoDataFrames on a column (not the geometry column) and then combines the rest of the data into one GeoDataFrame.

```
>>> world = gpd.read_file(geopandas.datasets.get_path('naturalearth_lowres'))
>>> cities = gpd.read_file(geopandas.datasets.get_path('naturalearth_cities'))

# Create subsets of the world and cities GeoDataFrames
```

```
>>> world = world[['continent', 'name', 'iso_a3']]
>>> cities = cities[['name', 'iso_a3']]

# Merge the GeoDataFrames on their iso_a3 code
>>> countries = world.merge(cities, on='iso_a3')
```

A spatial join merges two GeoDataFrames based on their geometry data. The function used for this is `sjoin`. `sjoin` accepts two GeoDataFrames and then direction on how to merge. It is imperative that two GeoDataFrames have the same CRS. In the example below, we merge using an `inner` join with the option `intersects`. The `inner` join means that we will only use keys in the intersection of both geometry columns, and we will retain only the left geometry column. `intersects` tells the GeoDataFrames to merge on GeoSeries that intersect each other. Other options include `contains` and `within`.

```
# Combine countries and cities on their geographic location
>>> countries = gpd.sjoin(world, cities, how='inner', op='intersects')
```

Problem 3. Load in the file `nytimes.csv`[a] as a DataFrame. This file includes county-level data for the cumulative cases and deaths of Covid-19 in the US, starting with the first case in Snohomish County, Washington, on January 21, 2020. Begin by converting the `date` column into a `DatetimeIndex`.

Next, use county FIPS codes to merge your GeoDataFrame from Problem 2 with the DataFrame you just created. A FIPS code is a 5-digit unique identifier for geographic locations. Ignore rows in the Covid-19 DataFrame with unknown FIPS codes as well as all data from Hawaii and Alaska.

Note that the `fips` column of the Covid-19 DataFrame stores entries as floats, but the county GeoDataFrame stores FIPS codes as strings, with the first two digits in the `STATEFP` column and the last three in the `COUNTYFP` column.

Once you have completed the merge, plot the cases from March 21, 2020 on top of your state outline map from Problem 2, using the CRS of EPSG:5071. Finally, print out the name of the county with the most cases on March 21, 2020 along with its case count.

[a] Source: `https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv`

Logarithmic Plotting Techniques

The color scheme of a graph can also help to communicate information clearly. A good list of available colormaps can be found at `https://matplotlib.org/3.2.1/gallery/color/colormap_reference.html`. Note also that you can reverse any colormap by adding `_r` to the end. The following example demonstrates some plotting features, using country GDP as in Figure 6.3.

```
>>> fig, ax = plt.subplots(figsize=(15,7), ncols=1, nrows=1)
>>> world.plot(column='gdp_md_est', cmap='plasma_r',
...         ax=ax, legend=True, edgecolor='gray')

# Add title and remove axis tick marks
```

```
>>> ax.set_title('GDP on Linear Scale')
>>> ax.set_yticks([])
>>> ax.set_xticks([])
>>> plt.show()
```

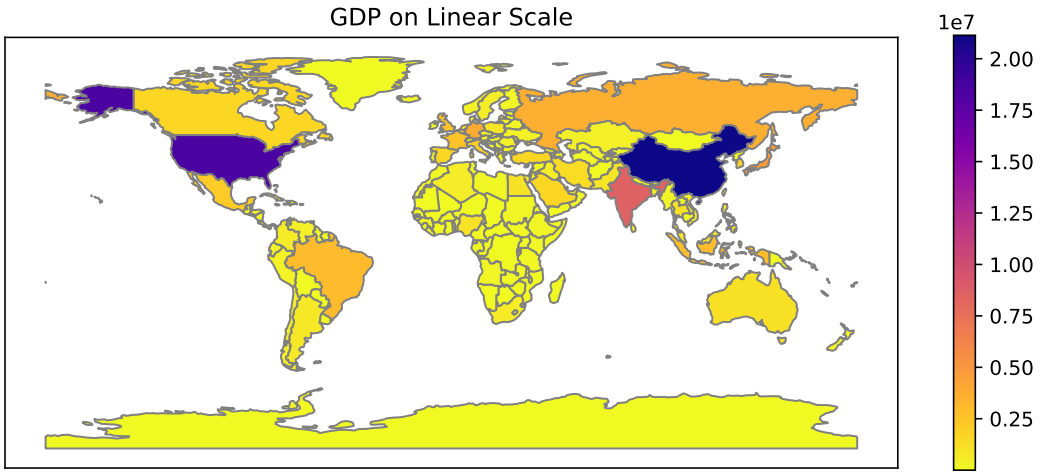

Figure 6.4: World map showing country GDP

Sometimes data can be much more informative when plotted on a logarithmic scale. See how the world map changes when we add a `norm` argument in the code below. Depending on the purpose of the graph, Figure 6.5 may be more informative than Figure 6.4.

```
>>> from matplotlib.colors import LogNorm
>>> from matplotlib.cm import ScalarMappable
>>> fig, ax = plt.subplots(figsize=(15,6), ncols=1, nrows=1)

# Set the norm using data bounds
>>> data = world.gdp_md_est
>>> norm = LogNorm(vmin=min(data), vmax=max(data))

# Plot the graph using the norm
>>> world.plot(column='gdp_md_est', cmap='plasma_r', ax=ax,
...         edgecolor='gray', norm=norm)

# Create a custom colorbar
>>> cbar = fig.colorbar(ScalarMappable(norm=norm, cmap='plasma_r'),
...         ax=ax, orientation='horizontal', pad=0, label='GDP')

>>> ax.set_title('Country Area on a Log Scale')
>>> ax.set_yticks([])
>>> ax.set_xticks([])
>>> plt.show()
```

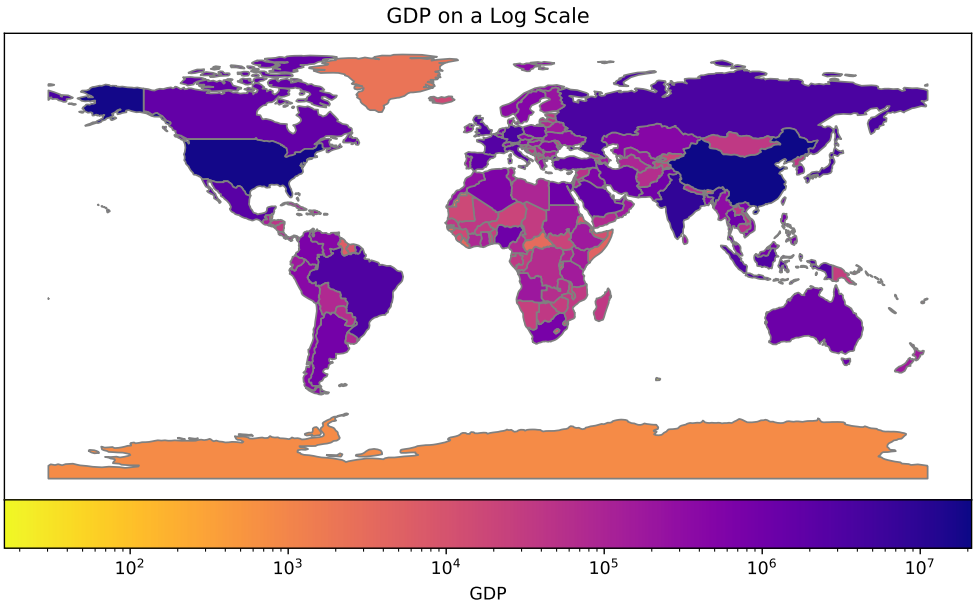

Figure 6.5: World map showing country GDP using a log scale

Problem 4. As in Problem 3, plot your state outline map from Problem 2 on top of a map of the Covid-19 cases from March 21, 2020. This time, however, use a log scale. Use EPSG:5071 for the CRS. Pick a good colormap (the counties with the most cases should generally be darkest) and be sure to display a colorbar.

Problem 5. In this problem, you will create an animation of the spread of Covid-19 through US counties from January 21, 2020 to June 21, 2020. Use a log scale and a good colormap, and be sure that you're using the same norm and colorbar for the whole animation. Use EPSG:5071 for the projection.

As a reminder, below is a summary of what you will need in order to animate this map. You may also find it helpful to refer to the animation section included with the Volume 4 lab manual.

1. Set up your figure and norm. Be sure to use the highest case count for your `vmax` so that the scale remains uniform.
2. Write your `update` function. This should plot the cases from a given day.
3. Set up your colorbar. Do this outside the `update` function to avoid adding a new colorbar each day.

4. Create the animation. Check to make sure everything displays properly before you save it.
5. Save the animation.
6. Display the animation.