13 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. To install Geopandas, 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, which is a collection of points. A polygon could be used to identify regions, such as a country.

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 13.1. 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 method `bounds` to find the maximum and minimum coordinates of Egypt in a built-in GeoDataFrame.

Table 13.1: Attributes and Methods for GeoSeries

Method	Description
`distance(other)`	returns minimum distance from GeoSeries to `other`
`area`	returns shape area
`contains(other)`	returns `True` if shape is contained in `other`
`intersects(other)`	returns `True` if shape intersects `other`

```
>>> world = geopandas.read_file(geopandas.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 can be accessed through the `.geometry` attribute and is the column that 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
from shapely.geometry import Point
```

```
# 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 = geopandas.GeoDataFrame(df, geometry='Coordinates')
```

Note

Longitude is the angular measurement starting at the Prime Meridian, 0°, and going to 180° to the east and −180° to the west. It is further divided into minutes and seconds. 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.

In order to find the distance between two points in latitude and longitude using the distance function, it is necessary to use the latitude and longitude to convert the points from spherical to cartesian coordinates. Recall that

$$\begin{aligned} x &= \rho \sin(\theta)\cos(\phi) \\ y &= \rho \sin(\theta)\cos(\phi) \\ z &= \rho \cos(\phi). \end{aligned}$$

This means that the code above must be modified such that the geometry column is a zipped list of cartesian coordinates rather than latitudes and longitudes.

Problem 1. Read in the file `airports.csv` as a Pandas DataFrame. Convert the DataFrame into a GeoDataFrame. (Set the geometry column as Point objects).

Find the distance between the following airports:

1. Halifax / CFB Shearwater Heliport (Halifax, Canada) to Murtala Muhammed International Airport (Lagos, Nigeria)
2. Don Mueang International Airport (Bangkok, Thailand) to Beijing Capital International Airport (Beijing, China)

3. Salt Lake City International Airport (Salt Lake City, USA) to Auckland International Airport (Auckland, New Zealand)

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 world GeoDataFrame for gdp_per_capita
world['gdp_per_cap'] = world.gdp_md_est / world.pop_est
```

While many Pandas functionalities are useful with GeoDataFrames, 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 according and `aggfunc` tells how to combine the information in all other columns. The default `aggfunc` is `first`, which returns the first application entry.

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

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

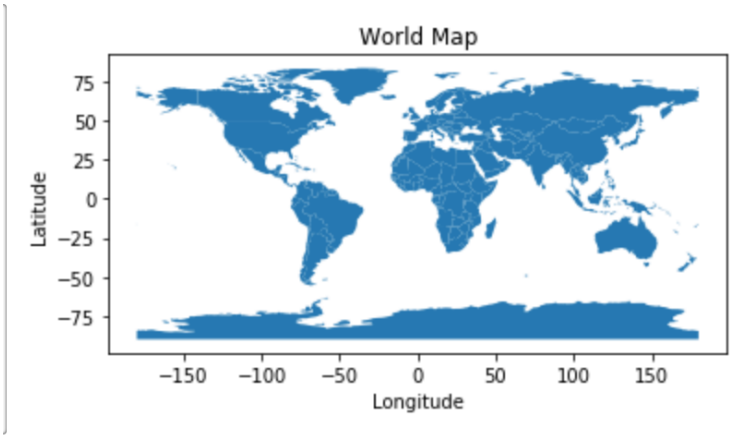
Problem 2. Read in the built-in GeoDataFrame `naturalearth_lowres`. Create a GeoDataFrame that only contains information about the southern hemisphere. Use this data to find the countries with the smallest and largest area in the southern hemisphere. Dissolve this GeoDataFrame to find the continent with the largest and smallest area in the southern hemisphere.

Plotting with GeoPandas

GeoDataFrames can be easily plotted with GeoPlot. GeoPlot plots the information from a GeoDataFrame based on their geometry column and displays the data as geometry objects. To use GeoPlots, import `geoplot`.

```
>>> import geoplot

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



With GeoPlot, multiple GeoDataFrames can be plotted at once. For example, if the information in a GeoDataFrame contains the coordinates of capitals of countries, it can be plotted on top of a GeoDataFrame containing the polygons of country boundaries to show a world map with capitals for each country. This is done by by setting one GeoDataFrame as the base of the GeoPlot.

```
>>> # Set world map as base
>>> base = world.plot(color='white', edgecolor='black')

>>> # Plot airports on world map
>>> airport.plot(ax=base, marker='o', color='green', markersize=5)
```

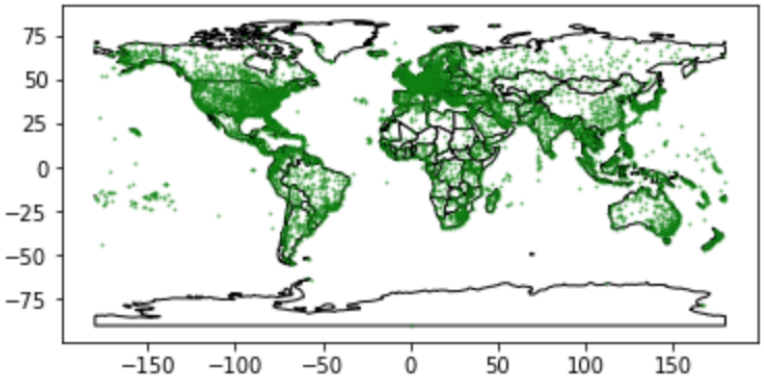


Figure 13.1: Airport-map

When plotting, GeoPlot refers to 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 CRS are WGS84 and WGS85. The crs of WGS84 is ESPG:4326, and it does the standard latitude-longitude projection used by GPS. WGS85, also known as Mercator and EPSG:3395 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 done correctly. GeoDataFrames being layered should also have the same CRS.

```
>>> import geoplot.crs as gcrs

>>> # Check crs of world GeoDataFrame
>>> world.crs
{'init': 'epsg:4326'}

# Change CRS of world to Mercator
>>> world.to_crs({'init': 'epsg:3395'}
>>> world.crs
{'init': 'epsg:3395']
```

GeoDataFrames can also be plotted using GeoPlot by the values in the the other attributes of the GeoSeries. This is done with a Choropleth map. 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.

```
>>> # Plot world based on gdp
>>> world.plot(column='gdp_md_est', cmap='OrRd')
```

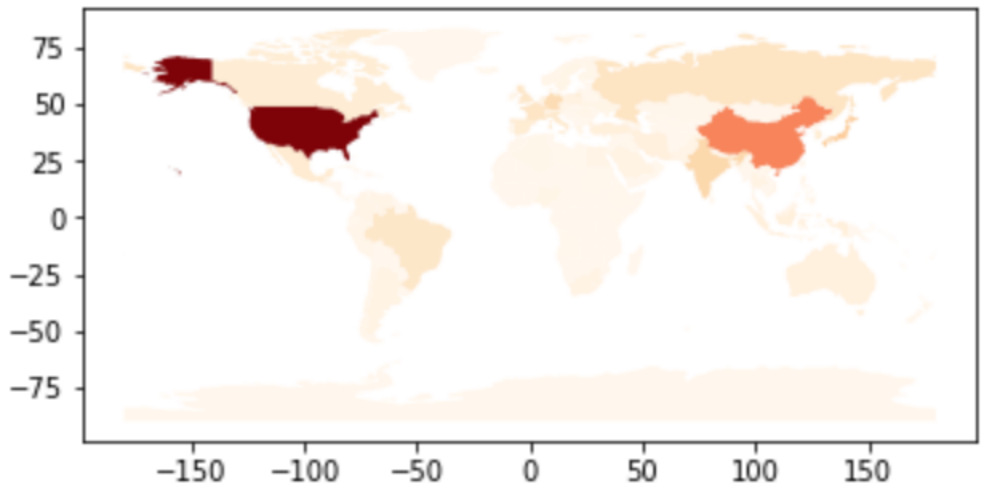


Figure 13.2: World Map Based on GDP

Problem 3. Using the built-in GeoDataFrame `naturalearth_lowres` and GeoPlots, create population density plots for Asia and South America. Use Mercator maps.

Merging GeoDataFrames

As Pandas DataFrames can be merged, GeoDataFrames can be joined on either attributes or spatial joins. An attribute join is similar to that of 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 = geopandas.read_file(geopandas.datasets.get_path('↩
    naturalearth_lowres'))
>>> cities = geopandas.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']]

>>> # Attribute join the GeoDataFrames on their iso_a3 code
>>> # (The iso_a3 code for Afghanistan is AFG)

>>> 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 the two GeoDataFrames desired to be merged and then direction on how to do the merge. It is imperative that two GeoDataFrames being joined spatially have the same CRS. In the example below, we merge with an inner join with the option `intersects`. The inner join tells us that we will use both geometry columns and retain only the left geometry column. Intersects tells the GeoDataFrames to merge on GeoSeries that intersect each other.

```
>>> # Combine countries and cities on their geographic location
>>> # This will combine countries with their capital

countries = geopandas.sjoin(world, cities, how='inner', op='intersects')
```

Problem 4. Merge the airports GeoDataFrame and the world GeoDataFrame on their spatial data. Use this new GeoDataFrame to find the airport in the country with the smallest population estimate.