Geographic Visualization and Data Analysis in Python

A Jupyter Notebook Example

Introduction

This presentation serves as a walk-through and detailed explanation of the provided Example Jupyter Notebook covering geographic visualization and data analysis in Python.

This Notebook loads pre-generated datasets provided by the libraries used to generate the visualizations and analyze the data and performs those tasks.

Here, we will step through the example code and explain each block’s function.

Part 1 - Choropleth Maps

```python
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
```

These three lines import the required libraries to construct the choropleth map. The *pandas* and *geopandas* libraries handle data representation and constructing the graph. The *geopandas* library is dependent on the *pandas* library as the names imply. The *matplotlib* library is required to display the plot in the Jupyter Notebook environment.

Choropleth Maps Cont.

```
world = gpd.read_file(
    gpd.datasets.get_path(
        'naturalearth_lowres'))
world = world[
    (world.pop_est>0) &
    (world.name!="Antarctica")]
world['gdp_per_cap'] =
    world.gdp_md_est /
    world.pop_est
```

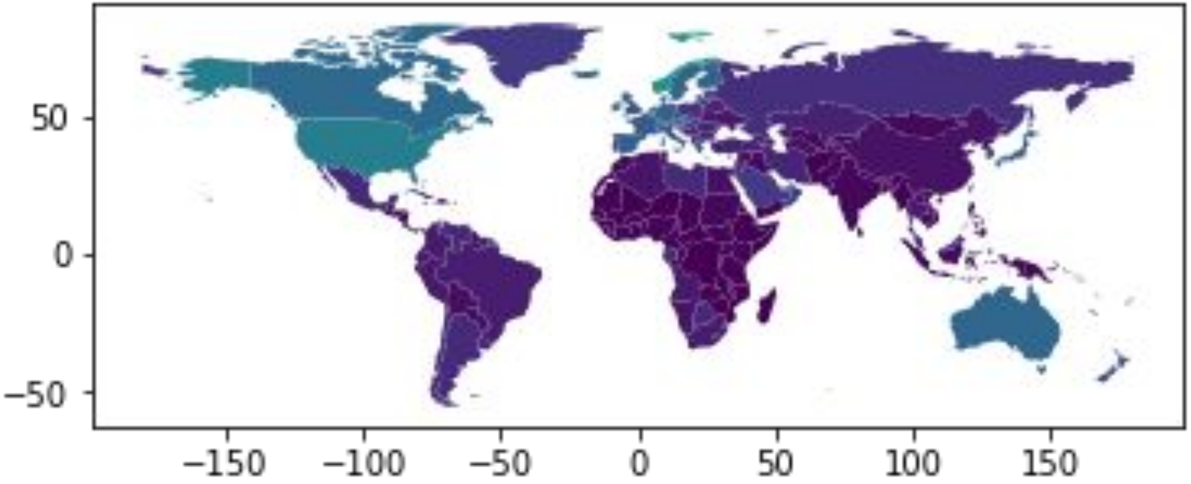
In these lines, a dataset of countries, their boundaries, their populations, and their GPDs is loaded from the *geopandas* library. The second line filters out areas with zero estimated population and Antartica to avoid an error in the calculation step, which is third. This step computes the quantity to be plotted, the GDP per Capita.

Choropleth Maps Cont.

```
world.plot(
    column='gdp_per_cap');
plt.show()
```

The *geopandas* library is set up to plot choropleth maps quickly and easily, so after computing the quantity of interest, we need only call the *plot* function. This plots the choropleth map of interest, using the specified column to determine the values plotted. Using *matplotlib*'s *show* function instead of *geopandas*' causes the graph to be output to the notebook environment.

Resultant Plot



Part 2 - Proportional Symbol Maps

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
```

As in the choropleth map, the only required libraries for the proportional symbol map are *pandas*, *geopandas*, and *matplotlib*.

Proportional Symbol Maps Cont.

```
world = gpd.read_file(
        gpd.datasets.get_path(
                'naturalearth_lowres'))
world = world[
        (world.pop_est>0) &
        (world.name!="Antarctica")]
```

These two lines repeat the load of the world map as performed for the choropleth map. This instance of the data set will serve as the base on which the proportional symbols will be displayed. This is only necessary since we are using geographic information and without the map of the world as a base, the data would be difficult to understand.

Proportional Symbol Maps Cont.

```
data = gpd.read_file(
        gpd.datasets.get_path(
                'naturalearth_lowres'))
data = data[
        (data.pop_est > 0) &
        (data.name != 'Antarctica')]
data['centroid_column'] = data.centroid
data['gdp_per_cap'] =
        data.gdp_md_est / data.pop_est
```

In these lines, we re-input the dataset to keep it separate from the base layer data. The last two lines repeat the GDP per Capita computation from the choropleth map and compute the centroid of each country. Computing the centroid will allow for the plotting of the proportional symbol for each country in the center of that country's outline on the map, as we will see in the next steps.

Proportional Symbol Maps Cont.

```
centroids = list(data['centroid_column'])
df = pd.DataFrame({
        'y':[centroids[i].y for i in
                range(len(centroids))],
        'x':[centroids[i].x for i in
                range(len(centroids))],
        'data':list(data['gdp_per_cap'])})
```

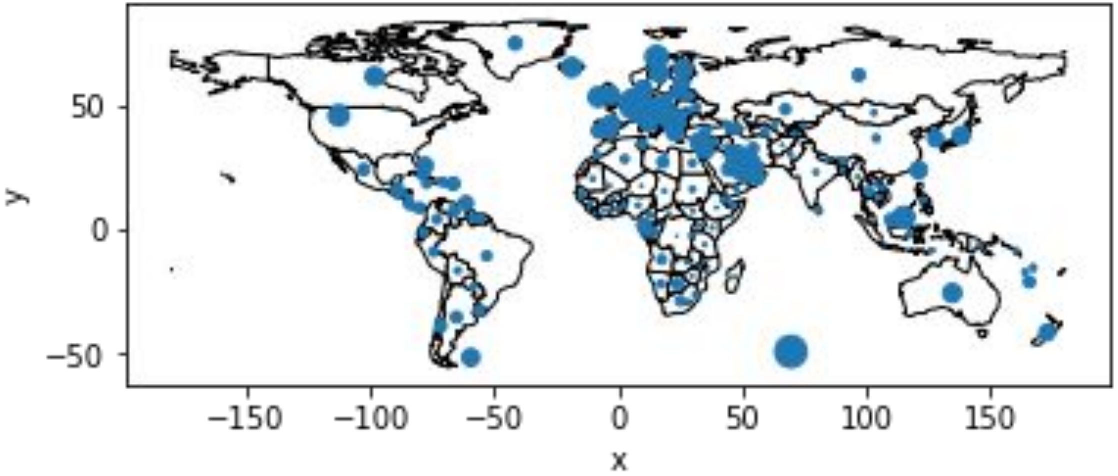
These two lines convert the data stored in the *data* variable into a *pandas* DataFrame suitable for plotting with *pandas*’ built-in functions. The centroids computed in the previous function are stored in a lat-long data structure that will correspond to y and x on the final graph. Finally, the computed GDP per Capita will determine the size of the marker.

Proportional Symbol Maps Cont.

```
base = world.plot(
    color='white',
    edgecolor='black')
df.plot(
    kind='scatter',
    x='x', y='y',
    s=df['data']*1000,
    ax=base)
plt.show()
```

These three lines plot the world map base, plot the symbols, and show the result in the notebook, respectively. Using a *pandas* Scatter plot on top of a *geopandas* outline base, we can represent each country’s GDP per Capita as a point located at that country’s centroid. The *s* parameter in the Scatter plot determines the scale for each point. In this case, we multiply by 1000 to make the points that represent poorer countries visible.

Resultant Plot



Part 3 - Computing Moran’s I

```
import pysal
import numpy as np
```

These two lines import the libraries required for easy computation of Moran’s I. The Python Spatial Analysis (*pysal*) library implements the analysis techniques we will use in an easy-to-access way, as we will see. It also contains sample data that we will use in this example.

Computing Moran’s I Cont.

```
f = pysal.open(
      pysal.examples.get_path(
            "stl_hom.txt"))
y = np.array(f.by_col['HR8893'])
w = pysal.open(
      pysal.examples.get_path(
            'stl.gal')).read()
```

These three lines import a data set of homicide rates for various parts of St Louis, Missouri. The first line imports the raw data set, while the second line selects a specific column of the data to analyze. The third line imports the continuity matrix, which was computed using Rook continuity.

Computing Moran’s I Cont.

```
mi = pysal.Moran(y, w,
     two_tailed=False)
print (mi.I)
```

These two lines compute and report Moran’s I for the data loaded in the previous step. The *Moran* data structure contains more information than just the final I computation that may be useful for spatial analysis. Complete information about the structure and other features of *pysal* is available in the *pysal* documentation.