**Generating Maps with Python**[**¶**](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-5-1-Generating-Maps-in-Python-py-v2.0.ipynb#Generating-Maps-with-Python)

Estimated time needed: **30** minutes

**Objectives**

After completing this lab you will be able to:

* Visualize geospatial data with Folium

**Introduction**

In this lab, we will learn how to create maps for different objectives. To do that, we will part ways with Matplotlib and work with another Python visualization library, namely **Folium**. What is nice about **Folium** is that it was developed for the sole purpose of visualizing geospatial data. While other libraries are available to visualize geospatial data, such as **plotly**, they might have a cap on how many API calls you can make within a defined time frame. **Folium**, on the other hand, is completely free.

**Table of Contents**

1. [Exploring Datasets with *pandas*](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-5-1-Generating-Maps-in-Python-py-v2.0.ipynb#0)
2. [Downloading and Prepping Data](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-5-1-Generating-Maps-in-Python-py-v2.0.ipynb#2)
3. [Introduction to Folium](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-5-1-Generating-Maps-in-Python-py-v2.0.ipynb#4)
4. [Map with Markers](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-5-1-Generating-Maps-in-Python-py-v2.0.ipynb#6)
5. [Choropleth Maps](https://jupyterlab-0-labs-prod-jupyterlab-us-east-1.labs.cognitiveclass.ai/user/akulasamson/lab/tree/labs/DV0101EN/DV0101EN-3-5-1-Generating-Maps-in-Python-py-v2.0.ipynb#8)

**Exploring Datasets with *pandas* and Matplotlib**

Toolkits: This lab heavily relies on [*pandas*](http://pandas.pydata.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) and [*Numpy*](http://www.numpy.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) for data wrangling, analysis, and visualization. The primary plotting library we will explore in this lab is [**Folium**](https://github.com/python-visualization/folium/).

Datasets:

1. San Francisco Police Department Incidents for the year 2016 - [Police Department Incidents](https://data.sfgov.org/Public-Safety/Police-Department-Incidents-Previous-Year-2016-/ritf-b9ki?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) from San Francisco public data portal. Incidents derived from San Francisco Police Department (SFPD) Crime Incident Reporting system. Updated daily, showing data for the entire year of 2016. Address and location has been anonymized by moving to mid-block or to an intersection.
2. Immigration to Canada from 1980 to 2013 - [International migration flows to and from selected countries - The 2015 revision](http://www.un.org/en/development/desa/population/migration/data/empirical2/migrationflows.shtml?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork20297740-2021-01-01) from United Nation's website. The dataset contains annual data on the flows of international migrants as recorded by the countries of destination. The data presents both inflows and outflows according to the place of birth, citizenship or place of previous / next residence both for foreigners and nationals. For this lesson, we will focus on the Canadian Immigration data

**Downloading and Prepping Data**

Import Primary Modules:

[1]:



**import** numpy **as** np *# useful for many scientific computing in Python*

**import** pandas **as** pd *# primary data structure library*

**Introduction to Folium**

Folium is a powerful Python library that helps you create several types of Leaflet maps. The fact that the Folium results are interactive makes this library very useful for dashboard building.

From the official Folium documentation page:

Folium builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the Leaflet.js library. Manipulate your data in Python, then visualize it in on a Leaflet map via Folium.

Folium makes it easy to visualize data that's been manipulated in Python on an interactive Leaflet map. It enables both the binding of data to a map for choropleth visualizations as well as passing Vincent/Vega visualizations as markers on the map.

The library has a number of built-in tilesets from OpenStreetMap, Mapbox, and Stamen, and supports custom tilesets with Mapbox or Cloudmade API keys. Folium supports both GeoJSON and TopoJSON overlays, as well as the binding of data to those overlays to create choropleth maps with color-brewer color schemes.

**Let's install Folium**

**Folium** is not available by default. So, we first need to install it before we are able to import it.

[ ]:



**!**conda install **-**c conda**-**forge folium**=**0.5.0 **--**yes

**import** folium

​

print('Folium installed and imported!')

Generating the world map is straightforward in **Folium**. You simply create a **Folium** *Map* object, and then you display it. What is attractive about **Folium** maps is that they are interactive, so you can zoom into any region of interest despite the initial zoom level.

[ ]:



*# define the world map*

world\_map **=** folium.Map()

​

*# display world map*

world\_map

Go ahead. Try zooming in and out of the rendered map above.

You can customize this default definition of the world map by specifying the centre of your map, and the initial zoom level.

All locations on a map are defined by their respective *Latitude* and *Longitude* values. So you can create a map and pass in a center of *Latitude* and *Longitude* values of **[0, 0]**.

For a defined center, you can also define the initial zoom level into that location when the map is rendered. **The higher the zoom level the more the map is zoomed into the center**.

Let's create a map centered around Canada and play with the zoom level to see how it affects the rendered map.

[ ]:



*# define the world map centered around Canada with a low zoom level*

world\_map **=** folium.Map(location**=**[56.130, **-**106.35], zoom\_start**=**4)

​

*# display world map*

world\_map

Let's create the map again with a higher zoom level.

[ ]:



*# define the world map centered around Canada with a higher zoom level*

world\_map **=** folium.Map(location**=**[56.130, **-**106.35], zoom\_start**=**8)

​

*# display world map*

world\_map

As you can see, the higher the zoom level the more the map is zoomed into the given center.

**Question**: Create a map of Mexico with a zoom level of 4.

[ ]:



*### type your answer here*

​

​

​

​

Click here for a sample python solution

*#The correct answer is:*

*#define Mexico's geolocation coordinates*

mexico\_latitude **=** 23.6345

mexico\_longitude **=** **-**102.5528

*# define the world map centered around Canada with a higher zoom level*

mexico\_map **=** folium.Map(location**=**[mexico\_latitude, mexico\_longitude], zoom\_start**=**4)

*# display world map*

mexico\_map

Another cool feature of **Folium** is that you can generate different map styles.

**A. Stamen Toner Maps**

These are high-contrast B+W (black and white) maps. They are perfect for data mashups and exploring river meanders and coastal zones.

Let's create a Stamen Toner map of canada with a zoom level of 4.

[ ]:



*# create a Stamen Toner map of the world centered around Canada*

world\_map **=** folium.Map(location**=**[56.130, **-**106.35], zoom\_start**=**4, tiles**=**'Stamen Toner')

​

*# display map*

world\_map

Feel free to zoom in and out to see how this style compares to the default one.

**B. Stamen Terrain Maps**

These are maps that feature hill shading and natural vegetation colors. They showcase advanced labeling and linework generalization of dual-carriageway roads.

Let's create a Stamen Terrain map of Canada with zoom level 4.

[ ]:



*# create a Stamen Toner map of the world centered around Canada*

world\_map **=** folium.Map(location**=**[56.130, **-**106.35], zoom\_start**=**4, tiles**=**'Stamen Terrain')

​

*# display map*

world\_map

Feel free to zoom in and out to see how this style compares to Stamen Toner, and the default style.

Zoom in and notice how the borders start showing as you zoom in, and the displayed country names are in English.

**Question**: Create a map of Mexico to visualize its hill shading and natural vegetation. Use a zoom level of 6.

[ ]:



*### type your answer here*

​

​

​

​

Click here for a sample python solution

*#The correct answer is:*

*#define Mexico's geolocation coordinates*

mexico\_latitude **=** 23.6345

mexico\_longitude **=** **-**102.5528

*# define the world map centered around Canada with a higher zoom level*

mexico\_map **=** folium.Map(location**=**[mexico\_latitude, mexico\_longitude], zoom\_start**=**6, tiles**=**'Stamen Terrain')

*# display world map*

mexico\_map

**Maps with Markers**

Let's download and import the data on police department incidents using *pandas* read\_csv() method.

Download the dataset and read it into a *pandas* dataframe:

[ ]:



df\_incidents **=** pd.read\_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Police\_Department\_Incidents\_-\_Previous\_Year\_\_2016\_.csv')

​

print('Dataset downloaded and read into a pandas dataframe!')

Let's take a look at the first five items in our dataset.

[ ]:



df\_incidents.head()

So each row consists of 13 features:

1. **IncidntNum**: Incident Number
2. **Category**: Category of crime or incident
3. **Descript**: Description of the crime or incident
4. **DayOfWeek**: The day of week on which the incident occurred
5. **Date**: The Date on which the incident occurred
6. **Time**: The time of day on which the incident occurred
7. **PdDistrict**: The police department district
8. **Resolution**: The resolution of the crime in terms whether the perpetrator was arrested or not
9. **Address**: The closest address to where the incident took place
10. **X**: The longitude value of the crime location
11. **Y**: The latitude value of the crime location
12. **Location**: A tuple of the latitude and the longitude values
13. **PdId**: The police department ID

Let's find out how many entries there are in our dataset.

[ ]:



df\_incidents.shape

So the dataframe consists of 150,500 crimes, which took place in the year 2016. In order to reduce computational cost, let's just work with the first 100 incidents in this dataset.

[ ]:



*# get the first 100 crimes in the df\_incidents dataframe*

limit **=** 100

df\_incidents **=** df\_incidents.iloc[0:limit, :]

Let's confirm that our dataframe now consists only of 100 crimes.

[ ]:



df\_incidents.shape

Now that we reduced the data a little, let's visualize where these crimes took place in the city of San Francisco. We will use the default style, and we will initialize the zoom level to 12.

[ ]:



*# San Francisco latitude and longitude values*

latitude **=** 37.77

longitude **=** **-**122.42

[ ]:



*# create map and display it*

sanfran\_map **=** folium.Map(location**=**[latitude, longitude], zoom\_start**=**12)

​

*# display the map of San Francisco*

sanfran\_map

Now let's superimpose the locations of the crimes onto the map. The way to do that in **Folium** is to create a *feature group* with its own features and style and then add it to the sanfran\_map.

[ ]:



*# instantiate a feature group for the incidents in the dataframe*

incidents **=** folium.map.FeatureGroup()

​

*# loop through the 100 crimes and add each to the incidents feature group*

**for** lat, lng, **in** zip(df\_incidents.Y, df\_incidents.X):

incidents.add\_child(

folium.features.CircleMarker(

[lat, lng],

radius**=**5, *# define how big you want the circle markers to be*

color**=**'yellow',

fill**=True**,

fill\_color**=**'blue',

fill\_opacity**=**0.6

)

)

​

*# add incidents to map*

sanfran\_map.add\_child(incidents)

You can also add some pop-up text that would get displayed when you hover over a marker. Let's make each marker display the category of the crime when hovered over.

[ ]:



*# instantiate a feature group for the incidents in the dataframe*

incidents **=** folium.map.FeatureGroup()

​

*# loop through the 100 crimes and add each to the incidents feature group*

**for** lat, lng, **in** zip(df\_incidents.Y, df\_incidents.X):

incidents.add\_child(

folium.features.CircleMarker(

[lat, lng],

radius**=**5, *# define how big you want the circle markers to be*

color**=**'yellow',

fill**=True**,

fill\_color**=**'blue',

fill\_opacity**=**0.6

)

)

​

*# add pop-up text to each marker on the map*

latitudes **=** list(df\_incidents.Y)

longitudes **=** list(df\_incidents.X)

labels **=** list(df\_incidents.Category)

​

**for** lat, lng, label **in** zip(latitudes, longitudes, labels):

folium.Marker([lat, lng], popup**=**label).add\_to(sanfran\_map)

*# add incidents to map*

sanfran\_map.add\_child(incidents)

Isn't this really cool? Now you are able to know what crime category occurred at each marker.

If you find the map to be so congested will all these markers, there are two remedies to this problem. The simpler solution is to remove these location markers and just add the text to the circle markers themselves as follows:

[ ]:



*# create map and display it*

sanfran\_map **=** folium.Map(location**=**[latitude, longitude], zoom\_start**=**12)

​

*# loop through the 100 crimes and add each to the map*

**for** lat, lng, label **in** zip(df\_incidents.Y, df\_incidents.X, df\_incidents.Category):

folium.features.CircleMarker(

[lat, lng],

radius**=**5, *# define how big you want the circle markers to be*

color**=**'yellow',

fill**=True**,

popup**=**label,

fill\_color**=**'blue',

fill\_opacity**=**0.6

).add\_to(sanfran\_map)

​

*# show map*

sanfran\_map

The other proper remedy is to group the markers into different clusters. Each cluster is then represented by the number of crimes in each neighborhood. These clusters can be thought of as pockets of San Francisco which you can then analyze separately.

To implement this, we start off by instantiating a *MarkerCluster* object and adding all the data points in the dataframe to this object.

[ ]:



**from** folium **import** plugins

​

*# let's start again with a clean copy of the map of San Francisco*

sanfran\_map **=** folium.Map(location **=** [latitude, longitude], zoom\_start **=** 12)

​

*# instantiate a mark cluster object for the incidents in the dataframe*

incidents **=** plugins.MarkerCluster().add\_to(sanfran\_map)

​

*# loop through the dataframe and add each data point to the mark cluster*

**for** lat, lng, label, **in** zip(df\_incidents.Y, df\_incidents.X, df\_incidents.Category):

folium.Marker(

location**=**[lat, lng],

icon**=None**,

popup**=**label,

).add\_to(incidents)

​

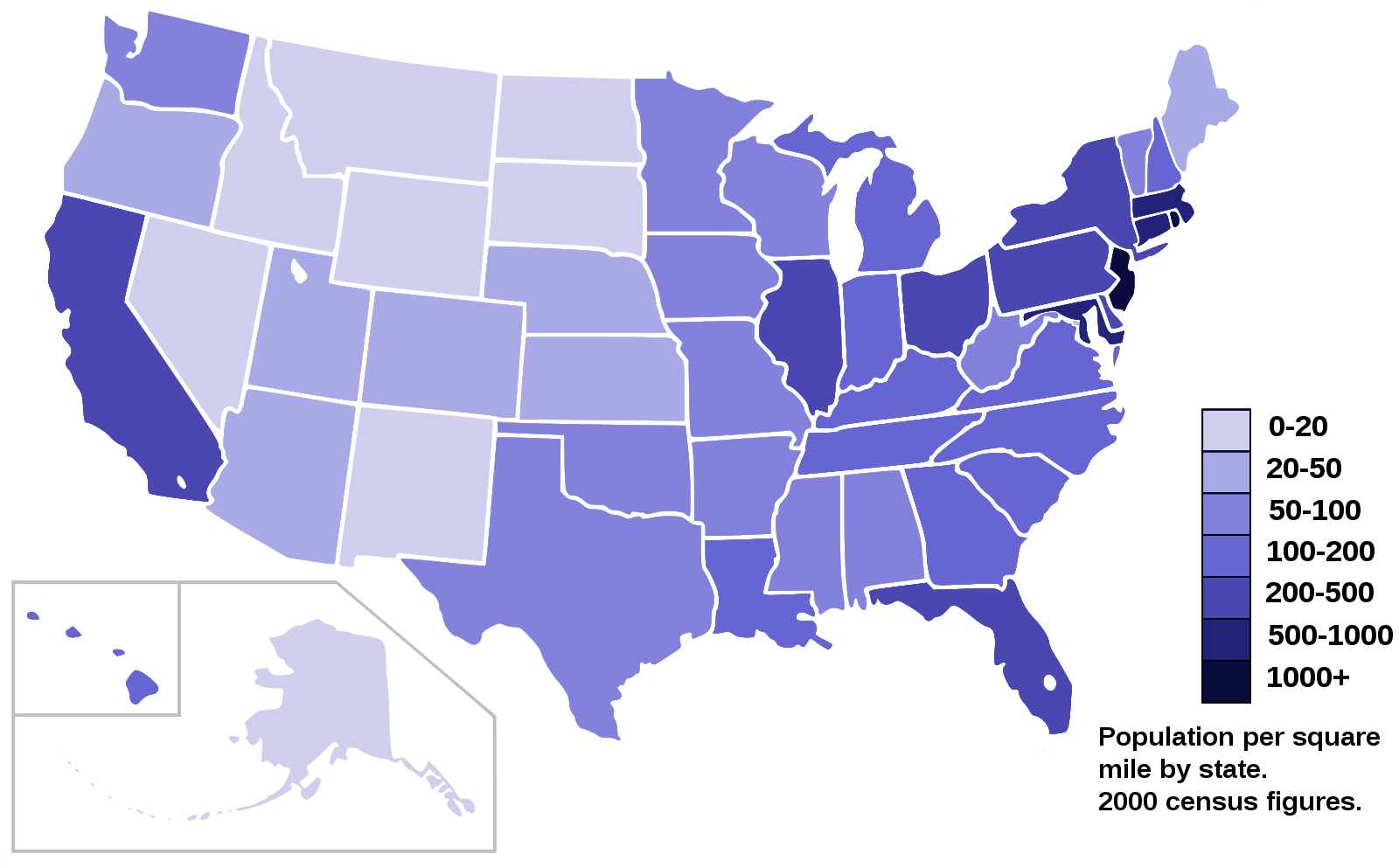
*# display map*

sanfran\_map

Notice how when you zoom out all the way, all markers are grouped into one cluster, *the global cluster*, of 100 markers or crimes, which is the total number of crimes in our dataframe. Once you start zooming in, the *global cluster* will start breaking up into smaller clusters. Zooming in all the way will result in individual markers.

**Choropleth Maps**

A Choropleth map is a thematic map in which areas are shaded or patterned in proportion to the measurement of the statistical variable being displayed on the map, such as population density or per-capita income. The choropleth map provides an easy way to visualize how a measurement varies across a geographic area, or it shows the level of variability within a region. Below is a Choropleth map of the US depicting the population by square mile per state.



Now, let's create our own Choropleth map of the world depicting immigration from various countries to Canada.

Let's first download and import our primary Canadian immigration dataset using *pandas* read\_excel() method. Normally, before we can do that, we would need to download a module which *pandas* requires reading in Excel files. This module was **openpyxl** (formerly **xlrd**). For your convenience, we have pre-installed this module, so you would not have to worry about that. Otherwise, you would need to run the following line of code to install the **openpyxl** module:

! pip3 install openpyxl

Download the dataset and read it into a *pandas* dataframe:

[ ]:



df\_can **=** pd.read\_excel(

'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Canada.xlsx',

sheet\_name**=**'Canada by Citizenship',

skiprows**=**range(20),

skipfooter**=**2)

​

print('Data downloaded and read into a dataframe!')

Let's take a look at the first five items in our dataset.

[ ]:



df\_can.head()

Let's find out how many entries there are in our dataset.

[ ]:



*# print the dimensions of the dataframe*

print(df\_can.shape)

Clean up data. We will make some modifications to the original dataset to make it easier to create our visualizations. Refer to *Introduction to Matplotlib and Line Plots* and *Area Plots, Histograms, and Bar Plots* notebooks for a detailed description of this preprocessing.

[ ]:



*# clean up the dataset to remove unnecessary columns (eg. REG)*

df\_can.drop(['AREA','REG','DEV','Type','Coverage'], axis**=**1, inplace**=True**)

​

*# let's rename the columns so that they make sense*

df\_can.rename(columns**=**{'OdName':'Country', 'AreaName':'Continent','RegName':'Region'}, inplace**=True**)

​

*# for sake of consistency, let's also make all column labels of type string*

df\_can.columns **=** list(map(str, df\_can.columns))

​

*# add total column*

df\_can['Total'] **=** df\_can.sum(axis**=**1)

​

*# years that we will be using in this lesson - useful for plotting later on*

years **=** list(map(str, range(1980, 2014)))

print ('data dimensions:', df\_can.shape)

Let's take a look at the first five items of our cleaned dataframe.

[ ]:



df\_can.head()

In order to create a Choropleth map, we need a GeoJSON file that defines the areas/boundaries of the state, county, or country that we are interested in. In our case, since we are endeavoring to create a world map, we want a GeoJSON that defines the boundaries of all world countries. For your convenience, we will be providing you with this file, so let's go ahead and download it. Let's name it **world\_countries.json**.

[ ]:



*# download countries geojson file*

**!** wget **--**quiet https:**//**cf**-**courses**-**data.s3.us.cloud**-**object**-**storage.appdomain.cloud**/**IBMDeveloperSkillsNetwork**-**DV0101EN**-**SkillsNetwork**/**Data**%**20Files**/**world\_countries.json

print('GeoJSON file downloaded!')

Now that we have the GeoJSON file, let's create a world map, centered around **[0, 0]** *latitude* and *longitude* values, with an initisal zoom level of 2.

[ ]:



world\_geo **=** r'world\_countries.json' *# geojson file*

​

*# create a plain world map*

world\_map **=** folium.Map(location**=**[0, 0], zoom\_start**=**2)

And now to create a Choropleth map, we will use the *choropleth* method with the following main parameters:

1. geo\_data, which is the GeoJSON file.
2. data, which is the dataframe containing the data.
3. columns, which represents the columns in the dataframe that will be used to create the Choropleth map.
4. key\_on, which is the key or variable in the GeoJSON file that contains the name of the variable of interest. To determine that, you will need to open the GeoJSON file using any text editor and note the name of the key or variable that contains the name of the countries, since the countries are our variable of interest. In this case, **name** is the key in the GeoJSON file that contains the name of the countries. Note that this key is case\_sensitive, so you need to pass exactly as it exists in the GeoJSON file.

[ ]:



*# generate choropleth map using the total immigration of each country to Canada from 1980 to 2013*

world\_map.choropleth(

geo\_data**=**world\_geo,

data**=**df\_can,

columns**=**['Country', 'Total'],

key\_on**=**'feature.properties.name',

fill\_color**=**'YlOrRd',

fill\_opacity**=**0.7,

line\_opacity**=**0.2,

legend\_name**=**'Immigration to Canada'

)

​

*# display map*

world\_map

As per our Choropleth map legend, the darker the color of a country and the closer the color to red, the higher the number of immigrants from that country. Accordingly, the highest immigration over the course of 33 years (from 1980 to 2013) was from China, India, and the Philippines, followed by Poland, Pakistan, and interestingly, the US.

Notice how the legend is displaying a negative boundary or threshold. Let's fix that by defining our own thresholds and starting with 0 instead of -6,918!

[ ]:



world\_geo **=** r'world\_countries.json'

​

*# create a numpy array of length 6 and has linear spacing from the minimum total immigration to the maximum total immigration*

threshold\_scale **=** np.linspace(df\_can['Total'].min(),

df\_can['Total'].max(),

6, dtype**=**int)

threshold\_scale **=** threshold\_scale.tolist() *# change the numpy array to a list*

threshold\_scale[**-**1] **=** threshold\_scale[**-**1] **+** 1 *# make sure that the last value of the list is greater than the maximum immigration*

​

*# let Folium determine the scale.*

world\_map **=** folium.Map(location**=**[0, 0], zoom\_start**=**2)

world\_map.choropleth(

geo\_data**=**world\_geo,

data**=**df\_can,

columns**=**['Country', 'Total'],

key\_on**=**'feature.properties.name',

threshold\_scale**=**threshold\_scale,

fill\_color**=**'YlOrRd',

fill\_opacity**=**0.7,

line\_opacity**=**0.2,

legend\_name**=**'Immigration to Canada',

reset**=True**

)

world\_map

Much better now! Feel free to play around with the data and perhaps create Choropleth maps for individuals years, or perhaps decades, and see how they compare with the entire period from 1980 to 2013.

**Thank you for completing this lab!**