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

Exploring Datasets with *pandas* and Matplotlib

Toolkits: This lab heavily relies on *pandas* and *Numpy* for data wrangling, analysis, and visualization. The primary plotting library we will explore in this lab is **Folium**.

Datasets:

1. San Francisco Police Department Incidents for the year 2016 - Police Department Incidents 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 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:

In [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.

In [2]:
```
!conda install -c conda-forge folium=0.5.0 --yes
import folium

print('Folium installed and imported!')
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda/envs/python

  added / updated specs:
    - folium=0.5.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    altair-4.1.0               |             py_1         614 KB  conda-forge
    attrs-21.2.0               |     pyhd8ed1ab_0          44 KB  conda-forge
    branca-0.4.2               |     pyhd8ed1ab_0          26 KB  conda-forge
    entrypoints-0.3            |  pyhd8ed1ab_1003           8 KB  conda-forge
    folium-0.5.0               |             py_0          45 KB  conda-forge
    jsonschema-3.2.0           |     pyhd8ed1ab_3          45 KB  conda-forge
    pyrsistent-0.17.3          |   py36h8f6f2f9_2          89 KB  conda-forge
    vincent-0.4.4              |             py_1          28 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         900 KB

The following NEW packages will be INSTALLED:

  altair             conda-forge/noarch::altair-4.1.0-py_1
  attrs              conda-forge/noarch::attrs-21.2.0-pyhd8ed1ab_0
  branca             conda-forge/noarch::branca-0.4.2-pyhd8ed1ab_0
  entrypoints        conda-forge/noarch::entrypoints-0.3-pyhd8ed1ab_1003
  folium             conda-forge/noarch::folium-0.5.0-py_0
  jsonschema         conda-forge/noarch::jsonschema-3.2.0-pyhd8ed1ab_3
  pyrsistent         conda-forge/linux-64::pyrsistent-0.17.3-py36h8f6f2f9_2
```

```
  vincent            conda-forge/noarch::vincent-0.4.4-py_1



Downloading and Extracting Packages
pyrsistent-0.17.3    | 89 KB     | ##################################### | 100%
folium-0.5.0         | 45 KB     | ##################################### | 100%
branca-0.4.2         | 26 KB     | ##################################### | 100%
altair-4.1.0         | 614 KB    | ##################################### | 100%
attrs-21.2.0         | 44 KB     | ##################################### | 100%
jsonschema-3.2.0     | 45 KB     | ##################################### | 100%
entrypoints-0.3      | 8 KB      | ##################################### | 100%
vincent-0.4.4        | 28 KB     | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
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.

In [4]:

```python
# define the world map
world_map = folium.Map()

# display world map
world_map
```

Out[4]: Make this Notebook Trusted to load map: File -> Trust Notebook

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.

In [5]:

```
# 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
```

Out[5]: Make this Notebook Trusted to load map: File -> Trust Notebook

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

In [6]:

```
# 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
```

Out[6]: Make this Notebook Trusted to load map: File -> Trust Notebook

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.

In [7]:
```
mexico_latitude = 23.6345
mexico_longitude = -102.5528

world_map = folium.Map(location=[mexico_latitude, mexico_longitude], zoom_start=4)

world_map
```

Out[7]: Make this Notebook Trusted to load map: File -> Trust Notebook

Click here for a sample python solution ```python #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.

In [8]:

```
# 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
```

Out[8]: Make this Notebook Trusted to load map: File -> Trust Notebook

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.

In [9]:

```
# 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 Terra

# display map
world_map
```

Out[9]: Make this Notebook Trusted to load map: File -> Trust Notebook

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.

In [10]:

```
mexico_latitude = 23.6345
mexico_longitude = -102.5528

world_map = folium.Map(location=[mexico_latitude, mexico_longitude], zoom_start=6, t

world_map
```

Out[10]: Make this Notebook Trusted to load map: File -> Trust Notebook

Click here for a sample python solution ```python #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:

In [11]:
```
df_incidents = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdo

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

```
Dataset downloaded and read into a pandas dataframe!
```

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

In [12]:
```
df_incidents.head()
```

Out[12]:

	IncidntNum	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution
0	120058272	WEAPON LAWS	POSS OF PROHIBITED WEAPON	Friday	01/29/2016 12:00:00 AM	11:00	SOUTHERN	ARREST, BOOKED
1	120058272	WEAPON LAWS	FIREARM, LOADED, IN VEHICLE, POSSESSION OR USE	Friday	01/29/2016 12:00:00 AM	11:00	SOUTHERN	ARREST, BOOKED
2	141059263	WARRANTS	WARRANT ARREST	Monday	04/25/2016 12:00:00 AM	14:59	BAYVIEW	ARREST, BOOKED
3	160013662	NON-CRIMINAL	LOST PROPERTY	Tuesday	01/05/2016 12:00:00 AM	23:50	TENDERLOIN	NONE
4	160002740	NON-CRIMINAL	LOST PROPERTY	Friday	01/01/2016 12:00:00 AM	00:30	MISSION	NONE

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.

In [13]:
```
df_incidents.shape
```

Out[13]:
```
(150500, 13)
```

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.

In [14]:
```
# 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.

In [15]:
```
df_incidents.shape
```

Out[15]:
```
(100, 13)
```

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.

In [16]:
```
# San Francisco latitude and longitude values
latitude = 37.77
longitude = -122.42
```

In [17]:
```
# create map and display it
sanfran_map = folium.Map(location=[latitude, longitude], zoom_start=12)
```

```
# display the map of San Francisco
sanfran_map
```

Out[17]: Make this Notebook Trusted to load map: File -> Trust Notebook

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`.

In [18]:

```
# 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)
```

Out[18]: Make this Notebook Trusted to load map: File -> Trust Notebook

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.

In [19]:
```
# 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)
```

Out[19]: Make this Notebook Trusted to load map: File -> Trust Notebook

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:

In [20]:

```
# 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
```

Out[20]: Make this Notebook Trusted to load map: File -> Trust Notebook

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.

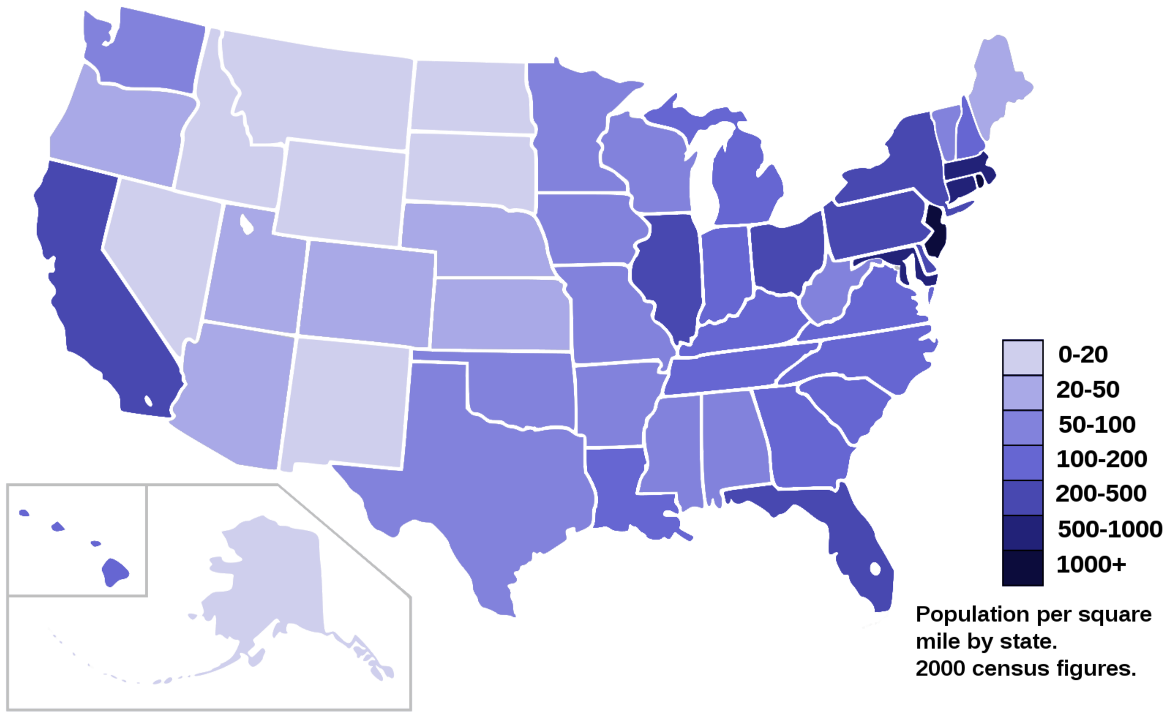
In [21]:
```
from folium import plugins

# 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
```

Out[21]: Make this Notebook Trusted to load map: File -> Trust Notebook

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:

In [22]:
```
df_can = pd.read_excel(
    'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
    sheet_name='Canada by Citizenship',
    skiprows=range(20),
    skipfooter=2)

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

```
Data downloaded and read into a dataframe!
```

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

In [24]:

```
df_can.head()
```

Out[24]:

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0

5 rows × 43 columns

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

In [28]:

```
# print the dimensions of the dataframe
print(df_can.shape)
```

```
(195, 39)
```

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.

In [26]:

```
# 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'

# 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)
```

```
data dimensions: (195, 39)
```

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

In [30]:
```
df_can.head()
```

Out[30]:

	Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	...	2005
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	...	3436
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	...	1223
2	Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	...	3626
3	American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	...	0
4	Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	...	0

5 rows × 39 columns

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**.

In [31]:
```
# download countries geojson file
! wget --quiet https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IB

print('GeoJSON file downloaded!')
```

```
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.

In [34]:
```
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.

In [35]:

```
# generate choropleth map using the total immigration of each country to Canada from
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
```

Out[35]: Make this Notebook Trusted to load map: File -> Trust Notebook

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!

In [36]:

```
world_geo = r'world_countries.json'

# create a numpy array of length 6 and has linear spacing from the minimum total imm
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

# 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
```

Out[36]: Make this Notebook Trusted to load map: File -> Trust Notebook

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!

Author

Alex Aklson.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-05-29	2.4	Weiqing Wang	Fixed typos and code smells.
2020-01-20	2.3	Lakshmi Holla	Updated TOC markdown
2020-11-03	2.2	Lakshmi Holla	Made changes in URL
2020-10-06	2.1	Lakshmi Holla	Removed Map Box Bright Style
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab