DV0101EN-3-5-1-Generating-Maps-in-Python-py-v2.0

December 20, 2018

Generating Maps with Python

0.1 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.

0.2 Table of Contents

1. Section **??**
2. Section **??**
3. Section **??**
4. Section **??**
5. Section **??**

1 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

2 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
```

3 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!')
```

```
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda

  added / updated specs:
    - folium=0.5.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
```

```
    vincent-0.4.4              |            py_1          28 KB  conda-forge
    branca-0.3.1               |            py_0          25 KB  conda-forge
    altair-2.3.0               |       py36_1001         533 KB  conda-forge
    pandas-0.23.4              |  py36hf8a1672_0        27.8 MB  conda-forge
    folium-0.5.0               |            py_0          45 KB  conda-forge
    ------------------------------------------------------------
                                           Total:        28.4 MB

The following NEW packages will be INSTALLED:

    altair:  2.3.0-py36_1001      conda-forge
    branca:  0.3.1-py_0           conda-forge
    folium:  0.5.0-py_0           conda-forge
    vincent: 0.4.4-py_1           conda-forge

The following packages will be UPDATED:

    pandas:  0.23.4-py37h04863e7_0            --> 0.23.4-py36hf8a1672_0 conda-forge


Downloading and Extracting Packages
vincent-0.4.4        | 28 KB     | ##################################### | 100%
branca-0.3.1         | 25 KB     | ##################################### | 100%
altair-2.3.0         | 533 KB    | ##################################### | 100%
pandas-0.23.4        | 27.8 MB   | ##################################### | 100%
folium-0.5.0         | 45 KB     | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Folium installed and imported!
```

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

In [3]:
```
# define the world map
world_map = folium.Map()

# display world map
world_map
```

Out[3]:
```
<folium.folium.Map at 0x7f86991bc550>
```

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 intial 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 intial 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 [4]: # 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[4]: <folium.folium.Map at 0x7f86975b18d0>
```

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

```
In [5]: # 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[5]: <folium.folium.Map at 0x7f8697537c18>
```

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 [6]: ### type your answer here

        mexico_map = folium.Map(location = [19.0000, -102.3667], zoom_start=4, tiles='Stamen Ter


        # display world map
        mexico_map

Out[6]: <folium.folium.Map at 0x7f86975499b0>
```

Double-click **here** for the solution.

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

3.0.1 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 [7]: # 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[7]: <folium.folium.Map at 0x7f8697557f28>
```

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

3.0.2 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 [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 Terrain')

        # display map
        world_map
```

```
Out[8]: <folium.folium.Map at 0x7f869756ecc0>
```

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

3.0.3 C. Mapbox Bright Maps

These are maps that quite similar to the default style, except that the borders are not visible with a low zoom level. Furthermore, unlike the default style where country names are displayed in each country's native language, *Mapbox Bright* style displays all country names in English.

Let's create a world map with this style.

```
In [9]: # create a world map with a Mapbox Bright style.
        world_map = folium.Map(tiles='Mapbox Bright')

        # display the map
        world_map
```

```
Out[9]: <folium.folium.Map at 0x7f86974ff9e8>
```

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]: ### type your answer here
         mexico_map = folium.Map(location = [19.0000, -102.3667], zoom_start=6, tiles = 'Stamen


         # display world map
         mexico_map
```

```
Out[10]: <folium.folium.Map at 0x7f86974f8978>
```

Double-click **here** for the solution.

4 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://ibm.box.com/shared/static/nmcltjmocdi8sd5tk93uembzd

         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  \
         0   120058272   WEAPON LAWS                 POSS OF PROHIBITED WEAPON
         1   120058272   WEAPON LAWS  FIREARM, LOADED, IN VEHICLE, POSSESSION OR USE
         2   141059263      WARRANTS                            WARRANT ARREST
         3   160013662  NON-CRIMINAL                             LOST PROPERTY
         4   160002740  NON-CRIMINAL                             LOST PROPERTY

           DayOfWeek                    Date   Time  PdDistrict      Resolution  \
         0    Friday  01/29/2016 12:00:00 AM  11:00    SOUTHERN  ARREST, BOOKED
         1    Friday  01/29/2016 12:00:00 AM  11:00    SOUTHERN  ARREST, BOOKED
         2    Monday  04/25/2016 12:00:00 AM  14:59     BAYVIEW  ARREST, BOOKED
         3   Tuesday  01/05/2016 12:00:00 AM  23:50  TENDERLOIN            NONE
         4    Friday  01/01/2016 12:00:00 AM  00:30     MISSION            NONE

                         Address           X          Y  \
         0  800 Block of BRYANT ST -122.403405  37.775421
         1  800 Block of BRYANT ST -122.403405  37.775421
         2   KEITH ST / SHAFTER AV -122.388856  37.729981
         3  JONES ST / OFARRELL ST -122.412971  37.785788
         4    16TH ST / MISSION ST -122.419672  37.765050

                                         Location            PdId
         0   (37.775420706711, -122.403404791479)  12005827212120
         1   (37.775420706711, -122.403404791479)  12005827212168
         2  (37.7299809672996, -122.388856204292)  14105926363010
         3  (37.7857883766888, -122.412970537591)  16001366271000
         4  (37.7650501214668, -122.419671780296)  16000274071000
```

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 bit, 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]: <folium.folium.Map at 0x7f8694ba9320>
```

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]: <folium.folium.Map at 0x7f8694ba9320>

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]: <folium.folium.Map at 0x7f8694ba9320>

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]: <folium.folium.Map at 0x7f8696fa6630>
```

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]: <folium.folium.Map at 0x7f8695301f60>
```

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.

5 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 to read in excel files. This module is **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 **xlrd** module:

```
!conda install -c anaconda xlrd --yes
```

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

```
In [22]: df_can = pd.read_excel('https://ibm.box.com/shared/static/lw190pt9zpy5bd1ptyg2aw15awomz
                                 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 [23]: df_can.head()
```

```
Out[23]:          Type    Coverage          OdName  AREA AreaName   REG  \
         0  Immigrants  Foreigners     Afghanistan   935     Asia  5501
         1  Immigrants  Foreigners         Albania   908   Europe   925
         2  Immigrants  Foreigners         Algeria   903   Africa   912
         3  Immigrants  Foreigners  American Samoa   909  Oceania   957
         4  Immigrants  Foreigners         Andorra   908   Europe   925

                   RegName  DEV             DevName  1980  ...   2004  2005  2006  \
         0   Southern Asia  902  Developing regions    16  ...   2978  3436  3009
         1 Southern Europe  901   Developed regions     1  ...   1450  1223   856
         2 Northern Africa  902  Developing regions    80  ...   3616  3626  4807
         3       Polynesia  902  Developing regions     0  ...      0     0     1
```

```
4  Southern Europe  901   Developed regions     0  ...      0     0     1

   2007  2008  2009  2010  2011  2012  2013
0  2652  2111  1746  1758  2203  2635  2004
1   702   560   716   561   539   620   603
2  3623  4005  5393  4752  4325  3774  4331
3     0     0     0     0     0     0     0
4     1     0     0     0     0     1     1

[5 rows x 43 columns]
```

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

```
In [24]: # print the dimensions of the dataframe
         print(df_can.shape)
```

```
(195, 43)
```

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 [25]: # 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 [26]: df_can.head()
```

```
Out[26]:          Country Continent           Region             DevName  1980  1981  \
         0    Afghanistan      Asia    Southern Asia  Developing regions    16    39
         1        Albania    Europe  Southern Europe   Developed regions     1     0
```

```
2         Algeria    Africa  Northern Africa  Developing regions    80    67
3  American Samoa   Oceania        Polynesia  Developing regions     0     1
4         Andorra    Europe  Southern Europe   Developed regions     0     0

   1982  1983  1984  1985  ...   2005  2006  2007  2008  2009  2010  2011  \
0    39    47    71   340  ...   3436  3009  2652  2111  1746  1758  2203
1     0     0     0     0  ...   1223   856   702   560   716   561   539
2    71    69    63    44  ...   3626  4807  3623  4005  5393  4752  4325
3     0     0     0     0  ...      0     1     0     0     0     0     0
4     0     0     0     0  ...      0     1     1     0     0     0     0

   2012  2013  Total
0  2635  2004  58639
1   620   603  15699
2  3774  4331  69439
3     0     0      6
4     1     1     15

[5 rows x 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 [27]:
```
# download countries geojson file
!wget --quiet https://ibm.box.com/shared/static/cto2qv7nx6yq19logfcissyy4euo8lho.json -

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 intial zoom level of 2, and using *Mapbox Bright* style.

In [28]:
```
world_geo = r'world_countries.json' # geojson file

# create a plain world map
world_map = folium.Map(location=[0, 0], zoom_start=2, tiles='Mapbox Bright')

# generate choropleth map using the total immigration of each country to Canada from 19
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[28]: <folium.folium.Map at 0x7f86947e00b8>

Wow! Very interesting 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!

In [29]:
```
world_geo = r'world_countries.json'

# create a numpy array of length 6 and has linear spacing from the minium total immigra
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 li

# let Folium determine the scale.
world_map = folium.Map(location=[0, 0], zoom_start=2, tiles='Mapbox Bright')
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[29]: <folium.folium.Map at 0x7f86936a57b8>

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.

5.0.4 Thank you for completing this lab!

This notebook was created by Alex Aklson. I hope you found this lab interesting and educational. Feel free to contact me if you have any questions!

This notebook is part of a course on **Coursera** called *Data Visualization with Python*. If you accessed this notebook outside the course, you can take this course online by clicking here.