scrubbing-data-visualization

January 27, 2022

1 Scrubbing data and visualizing

1.1 Objectives

1.1.1 Objectives

1. Retrieve data and place in a pandas dataframe
2. Determine modifications needed in the data
3. Match the data to visualization package

1.1.2 Example Questions

1. How can we retrieve data from a webpage?
2. How can we parse hhtml?
3. What is a MultiIndex in Pandas?
4. What is a Choropleth map?

1.2 Highlevel topics

- Data retrieval
- Data storage
- Data manipulation
- Visualization

1.3 What to hand in

- As far as you can get!
- What did you learn about scrubbing data today?
  - 1. Tools like `pandas` and `beautifulsoup` make extracting data from webpages quick and easy!
  - 2. We can use `folium` to make nice plots of geographical data!
  - 3. It's important to check the attributes and values in your data files to make sure the functions you use to plot give you the correct figures!

1.4 Synopsis

You're a structural engineer working on a team that analyses the bridge infrastructure in the US. To make a convincing argument, you are constructing a map of the current bridge conditions across the US.

Your Task Your goal is to plot the bridge conditions at the state level.

1.5 Datasets

In this session two datasets will be used: - Bridge Condition by Highway System 2019 - https://www.fhwa.dot.gov/bridge/nbi/no10/condition19.cfm - Bridge Condition by County 2019 - https://www.fhwa.dot.gov/bridge/nbi/no10/county19.cfm - In addition you will use the state/county level geo files: - `us-states.json`: https://github.com/python-visualization/folium/tree/master/examples/data - http://eric.clst.org/tech/usgeojson/

1.6 Getting Started

We will introduce four new packages in this lesson:

- `requests` is a package the makes URL requests *easy*. Give it a URL and it retrieves the page.
- `bs4` or BeautifulSoup parses an html file and places it in a convenient structure
- `json` is a package for reading structured JSON files
- `folium` is one of many packages that can be used to plot information on a geographical map

[1]:
```python
import folium          # visualizing maps
import os
import pandas as pd    # data frames
import bs4             # parse html
import requests        #
import json            # structured data
from IPython.display import HTML, display
```

[2]:
```python
m = folium.Map(
    location=[40.114942, -88.226492],
    #tiles='Stamen Toner',
    tiles="Stamen Terrain",
    zoom_start=13
)


m
```

[2]:
```
<folium.folium.Map at 0x7f181faac8b0>
```

1.7 First grab the webpage

Here we'll do two things:

1. retrieve the raw html of the webpage; and
2. parse the html to make a structured `soup`

[3]:
```python
url = 'https://www.fhwa.dot.gov/bridge/nbi/no10/condition21.cfm'
r = requests.get(url)                    # grab the html source
html = r.text                            # as text
soup = bs4.BeautifulSoup(html, 'lxml') # make a parseable "tree" of html
```

1.8 ________________________________

We can do any number of things with `soup` at this point. We can scrub for emails, find links, extract figures, etc. In this case we wish to find all of the tables in the html. `<table>` and `<table class="something">` are both examples of tags that we wish to find — bs4 makes this easy – try it with `find_all` (https://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigating-using-tag-names)

```
table = ...
```

1.9 Try it! ↓

[4]:
```python
table = soup.find_all('table')[0]
display(HTML(str(table)))
```

```
<IPython.core.display.HTML object>
```

1.10 ________________________________

If you have `table`, you can `find_all` on the resulting markup.

```
rows = table.find_all()
```

Use this to find all rows (marked with `tr`) in the table.

1.11 Try it! ↓

[5]:
```python
rows = table.find_all('tr')
```

Let's print it:

[6]:
```python
print(rows[0])
print(rows[1])
print(rows[2])
```

```
<tr>
<th rowspan="2" scope="col">State</th>
<th colspan="4" scope="colgroup">Bridge Counts</th>
<th colspan="4" scope="colgroup">Bridge Area (Square Meters)</th>
</tr>
<tr>
<th scope="col">All</th>
<th scope="col">Good</th>
<th scope="col">Fair</th>
<th scope="col">Poor</th>
<th scope="col">All</th>
<th scope="col">Good</th>
<th scope="col">Fair</th>
<th scope="col">Poor</th>
</tr>
<tr>
```

```
<th class="left" scope="row">ALABAMA</th>
<td class="txtright">16,164</td> <td class="txtright">6,550</td> <td
class="txtright">9,028</td> <td class="txtright">586</td> <td
class="txtright">9,979,973</td> <td class="txtright">3,655,411</td> <td
class="txtright">6,173,319</td> <td class="txtright">151,243</td>
</tr>
```

1.11.1 One approach

One approach is to zip through the rows, then parse each of the columns. We may do this like the following:

[7]:
```python
for row in rows:

    # find all 'th' headers
    state_name = row.find('th', {"class": "left"})
    if state_name is not None:

        # get the state name
        state_name = state_name.text
        print(state_name)

        # get the next four data rows
        count = row.findAll('td')[:4]
        count = [int(c.text.replace(',','')) for c in count]
        print(count)
```

```
ALABAMA
[16164, 6550, 9028, 586]
ALASKA
[1632, 716, 782, 134]
ARIZONA
[8467, 5275, 3075, 117]
ARKANSAS
[12941, 6234, 6028, 679]
CALIFORNIA
[25737, 12224, 12020, 1493]
COLORADO
[8869, 3063, 5337, 469]
CONNECTICUT
[4361, 1249, 2881, 231]
DELAWARE
[875, 291, 567, 17]
DISTRICT OF COLUMBIA
[246, 74, 165, 7]
FLORIDA
[12680, 8052, 4169, 459]
GEORGIA
```

[14987, 11054, 3614, 319]
HAWAII
[1162, 265, 810, 87]
IDAHO
[4561, 1322, 3001, 238]
ILLINOIS
[26846, 12848, 11593, 2405]
INDIANA
[19337, 7866, 10389, 1082]
IOWA
[23870, 9354, 10012, 4504]
KANSAS
[24925, 13335, 10313, 1277]
KENTUCKY
[14410, 4089, 9331, 990]
LOUISIANA
[12782, 5931, 5220, 1631]
MAINE
[2485, 728, 1443, 314]
MARYLAND
[5446, 1789, 3404, 253]
MASSACHUSETTS
[5245, 1321, 3468, 456]
MICHIGAN
[11284, 4091, 5953, 1240]
MINNESOTA
[13496, 7857, 5021, 618]
MISSISSIPPI
[16788, 9921, 5693, 1174]
MISSOURI
[24590, 9654, 12718, 2218]
MONTANA
[5266, 1600, 3301, 365]
NEBRASKA
[15348, 7966, 6102, 1280]
NEVADA
[2067, 1070, 968, 29]
NEW HAMPSHIRE
[2527, 1344, 989, 194]
NEW JERSEY
[6798, 1809, 4507, 482]
NEW MEXICO
[4025, 1466, 2351, 208]
NEW YORK
[17555, 6355, 9528, 1672]
NORTH CAROLINA
[18877, 7840, 9712, 1325]
NORTH DAKOTA

```
[4285, 2046, 1758, 481]
OHIO
[27151, 16493, 9324, 1334]
OKLAHOMA
[23220, 9898, 11026, 2296]
OREGON
[8235, 2800, 5053, 382]
PENNSYLVANIA
[23166, 7705, 12263, 3198]
RHODE ISLAND
[779, 168, 475, 136]
SOUTH CAROLINA
[9395, 4142, 4754, 499]
SOUTH DAKOTA
[5886, 1943, 2925, 1018]
TENNESSEE
[20331, 8689, 10801, 841]
TEXAS
[55175, 27807, 26579, 789]
UTAH
[3056, 1005, 1988, 63]
VERMONT
[2836, 1494, 1274, 68]
VIRGINIA
[13997, 4644, 8823, 530]
WASHINGTON
[8358, 4331, 3626, 401]
WEST VIRGINIA
[7314, 1719, 4105, 1490]
WISCONSIN
[14307, 7289, 6031, 987]
WYOMING
[3114, 920, 1964, 230]
GUAM
[10, 2, 6, 2]
PUERTO RICO
[2334, 426, 1626, 282]
U.S. VIRGIN ISLANDS
[24, 4, 14, 6]
TOTALS
[619622, 278128, 297908, 43586]
```

1.11.2 Another (easier) approach

1.12 ______________________

The previous approach is often necessary. Dirty data, incomplete html, different formats, etc often force us to parse the html by hand. However in the case of a table, Pandas can be used directly:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_html.html

How can we use pandas to read the soup table? ## Try it! ↓

[8]:
```
df = pd.read_html(url)[0]
#pd.read_html(str(table))[0] also works
```

Print the dataframe:

[9]:
```
df
```

[9]:
```
                  State Bridge Counts                               \
                  State           All   Good   Fair   Poor
0               ALABAMA         16164   6550   9028    586
1                ALASKA          1632    716    782    134
2               ARIZONA          8467   5275   3075    117
3              ARKANSAS         12941   6234   6028    679
4            CALIFORNIA         25737  12224  12020   1493
5              COLORADO          8869   3063   5337    469
6           CONNECTICUT          4361   1249   2881    231
7              DELAWARE           875    291    567     17
8    DISTRICT OF COLUMBIA         246     74    165      7
9               FLORIDA         12680   8052   4169    459
10              GEORGIA         14987  11054   3614    319
11               HAWAII          1162    265    810     87
12                IDAHO          4561   1322   3001    238
13             ILLINOIS         26846  12848  11593   2405
14              INDIANA         19337   7866  10389   1082
15                 IOWA         23870   9354  10012   4504
16               KANSAS         24925  13335  10313   1277
17             KENTUCKY         14410   4089   9331    990
18            LOUISIANA         12782   5931   5220   1631
19                MAINE          2485    728   1443    314
20             MARYLAND          5446   1789   3404    253
21        MASSACHUSETTS          5245   1321   3468    456
22             MICHIGAN         11284   4091   5953   1240
23            MINNESOTA         13496   7857   5021    618
24          MISSISSIPPI         16788   9921   5693   1174
25             MISSOURI         24590   9654  12718   2218
26              MONTANA          5266   1600   3301    365
27             NEBRASKA         15348   7966   6102   1280
28               NEVADA          2067   1070    968     29
29        NEW HAMPSHIRE          2527   1344    989    194
30           NEW JERSEY          6798   1809   4507    482
31           NEW MEXICO          4025   1466   2351    208
32             NEW YORK         17555   6355   9528   1672
33       NORTH CAROLINA         18877   7840   9712   1325
34         NORTH DAKOTA          4285   2046   1758    481
```

35	OHIO	27151	16493	9324	1334
36	OKLAHOMA	23220	9898	11026	2296
37	OREGON	8235	2800	5053	382
38	PENNSYLVANIA	23166	7705	12263	3198
39	RHODE ISLAND	779	168	475	136
40	SOUTH CAROLINA	9395	4142	4754	499
41	SOUTH DAKOTA	5886	1943	2925	1018
42	TENNESSEE	20331	8689	10801	841
43	TEXAS	55175	27807	26579	789
44	UTAH	3056	1005	1988	63
45	VERMONT	2836	1494	1274	68
46	VIRGINIA	13997	4644	8823	530
47	WASHINGTON	8358	4331	3626	401
48	WEST VIRGINIA	7314	1719	4105	1490
49	WISCONSIN	14307	7289	6031	987
50	WYOMING	3114	920	1964	230
51	GUAM	10	2	6	2
52	PUERTO RICO	2334	426	1626	282
53	U.S. VIRGIN ISLANDS	24	4	14	6
54	TOTALS	619622	278128	297908	43586

Bridge Area (Square Meters)

	All	Good	Fair	Poor
0	9979973	3655411	6173319	151243
1	757896	269361	436844	51692
2	6069459	3327139	2665058	77262
3	6837378	3147183	3359349	330845
4	30124854	13956035	14379766	1789054
5	5155162	1975318	2927082	252762
6	3448066	567456	2623888	256722
7	1026306	213835	780631	31840
8	572196	102245	434978	34974
9	18229935	11431006	6324518	474411
10	10453854	7969336	2370016	114502
11	1375035	268221	1072343	34472
12	1787321	441088	1285749	60483
13	13560234	4435753	7466619	1657862
14	8346936	3780629	4286858	279449
15	8981008	4181314	3932278	867415
16	8992619	5756802	2953688	282129
17	6639366	2099123	4222010	318232
18	16707128	6824971	8461863	1420294
19	1274099	418649	742246	113204
20	5478185	1472953	3836649	168582
21	4148482	863976	2816797	467709
22	6486938	1887323	4097377	502238
23	7135710	3068163	3775528	292019

24	9884290	5905056	3613316	365918
25	10795966	3818790	6073873	903304
26	2070501	505213	1419020	146268
27	4382788	2511952	1659280	211556
28	1911947	935622	953490	22835
29	1157396	658752	421946	76698
30	7501410	1704029	5289213	508168
31	2094826	724619	1272684	97523
32	13379835	3843461	8134311	1402063
33	10672362	4723647	5327677	621039
34	1326782	659090	599559	68134
35	14189311	8583424	5129168	476719
36	9015245	4269938	4337441	407866
37	5096809	1023505	3910045	163260
38	13315383	4055576	8287128	972679
39	766405	134125	483071	149209
40	7129031	3181128	3631485	316418
41	1859316	513911	1164755	180650
42	10395496	3860487	6050784	484225
43	53243079	27208992	25437286	596801
44	1973115	591663	1363559	17893
45	944777	479434	430291	35052
46	10264213	3306325	6597191	360697
47	7166874	2786785	3919129	460960
48	3828355	571461	2690651	566242
49	7098826	3541381	3296350	261095
50	1329562	307142	908151	114269
51	769	133	464	171
52	2213107	440168	1576926	196013
53	4969	447	4081	441
54	398580883	172959545	205407779	20213559

Now lets just take the state name and the bridge counts to make the dataframe a bit more manageable:

[10]:
```
dfcounts = df[['State','Bridge Counts']].copy()
dfcounts
```

[10]:

	State	Bridge Counts			
	State	All	Good	Fair	Poor
0	ALABAMA	16164	6550	9028	586
1	ALASKA	1632	716	782	134
2	ARIZONA	8467	5275	3075	117
3	ARKANSAS	12941	6234	6028	679
4	CALIFORNIA	25737	12224	12020	1493
5	COLORADO	8869	3063	5337	469
6	CONNECTICUT	4361	1249	2881	231
7	DELAWARE	875	291	567	17

8	DISTRICT OF COLUMBIA	246	74	165	7
9	FLORIDA	12680	8052	4169	459
10	GEORGIA	14987	11054	3614	319
11	HAWAII	1162	265	810	87
12	IDAHO	4561	1322	3001	238
13	ILLINOIS	26846	12848	11593	2405
14	INDIANA	19337	7866	10389	1082
15	IOWA	23870	9354	10012	4504
16	KANSAS	24925	13335	10313	1277
17	KENTUCKY	14410	4089	9331	990
18	LOUISIANA	12782	5931	5220	1631
19	MAINE	2485	728	1443	314
20	MARYLAND	5446	1789	3404	253
21	MASSACHUSETTS	5245	1321	3468	456
22	MICHIGAN	11284	4091	5953	1240
23	MINNESOTA	13496	7857	5021	618
24	MISSISSIPPI	16788	9921	5693	1174
25	MISSOURI	24590	9654	12718	2218
26	MONTANA	5266	1600	3301	365
27	NEBRASKA	15348	7966	6102	1280
28	NEVADA	2067	1070	968	29
29	NEW HAMPSHIRE	2527	1344	989	194
30	NEW JERSEY	6798	1809	4507	482
31	NEW MEXICO	4025	1466	2351	208
32	NEW YORK	17555	6355	9528	1672
33	NORTH CAROLINA	18877	7840	9712	1325
34	NORTH DAKOTA	4285	2046	1758	481
35	OHIO	27151	16493	9324	1334
36	OKLAHOMA	23220	9898	11026	2296
37	OREGON	8235	2800	5053	382
38	PENNSYLVANIA	23166	7705	12263	3198
39	RHODE ISLAND	779	168	475	136
40	SOUTH CAROLINA	9395	4142	4754	499
41	SOUTH DAKOTA	5886	1943	2925	1018
42	TENNESSEE	20331	8689	10801	841
43	TEXAS	55175	27807	26579	789
44	UTAH	3056	1005	1988	63
45	VERMONT	2836	1494	1274	68
46	VIRGINIA	13997	4644	8823	530
47	WASHINGTON	8358	4331	3626	401
48	WEST VIRGINIA	7314	1719	4105	1490
49	WISCONSIN	14307	7289	6031	987
50	WYOMING	3114	920	1964	230
51	GUAM	10	2	6	2
52	PUERTO RICO	2334	426	1626	282
53	U.S. VIRGIN ISLANDS	24	4	14	6
54	TOTALS	619622	278128	297908	43586

1.12.1 A MultiIndex?!

If we take a look a the column names we run into another type in Pandas: a MultiIndex.

[11]:
```
dfcounts.columns
```

[11]:
```
MultiIndex([(        'State', 'State'),
            ('Bridge Counts',   'All'),
            ('Bridge Counts',  'Good'),
            ('Bridge Counts',  'Fair'),
            ('Bridge Counts',  'Poor')],
           )
```

One easy thing to do in this case (mainly to make referencing a specific column easier) is to reduce to the column header to a simple `Index`. Here we'll just use the second level of the MultiIndex:

[12]:
```
dfcounts.columns = dfcounts.columns.get_level_values(1)
print(dfcounts.columns)
dfcounts
```

```
Index(['State', 'All', 'Good', 'Fair', 'Poor'], dtype='object')
```

[12]:

	State	All	Good	Fair	Poor
0	ALABAMA	16164	6550	9028	586
1	ALASKA	1632	716	782	134
2	ARIZONA	8467	5275	3075	117
3	ARKANSAS	12941	6234	6028	679
4	CALIFORNIA	25737	12224	12020	1493
5	COLORADO	8869	3063	5337	469
6	CONNECTICUT	4361	1249	2881	231
7	DELAWARE	875	291	567	17
8	DISTRICT OF COLUMBIA	246	74	165	7
9	FLORIDA	12680	8052	4169	459
10	GEORGIA	14987	11054	3614	319
11	HAWAII	1162	265	810	87
12	IDAHO	4561	1322	3001	238
13	ILLINOIS	26846	12848	11593	2405
14	INDIANA	19337	7866	10389	1082
15	IOWA	23870	9354	10012	4504
16	KANSAS	24925	13335	10313	1277
17	KENTUCKY	14410	4089	9331	990
18	LOUISIANA	12782	5931	5220	1631
19	MAINE	2485	728	1443	314
20	MARYLAND	5446	1789	3404	253
21	MASSACHUSETTS	5245	1321	3468	456
22	MICHIGAN	11284	4091	5953	1240
23	MINNESOTA	13496	7857	5021	618
24	MISSISSIPPI	16788	9921	5693	1174
25	MISSOURI	24590	9654	12718	2218

```
26              MONTANA    5266    1600    3301    365
27             NEBRASKA   15348    7966    6102   1280
28               NEVADA    2067    1070     968     29
29        NEW HAMPSHIRE    2527    1344     989    194
30           NEW JERSEY    6798    1809    4507    482
31           NEW MEXICO    4025    1466    2351    208
32             NEW YORK   17555    6355    9528   1672
33       NORTH CAROLINA   18877    7840    9712   1325
34         NORTH DAKOTA    4285    2046    1758    481
35                 OHIO   27151   16493    9324   1334
36             OKLAHOMA   23220    9898   11026   2296
37               OREGON    8235    2800    5053    382
38         PENNSYLVANIA   23166    7705   12263   3198
39         RHODE ISLAND     779     168     475    136
40       SOUTH CAROLINA    9395    4142    4754    499
41         SOUTH DAKOTA    5886    1943    2925   1018
42            TENNESSEE   20331    8689   10801    841
43                TEXAS   55175   27807   26579    789
44                 UTAH    3056    1005    1988     63
45              VERMONT    2836    1494    1274     68
46             VIRGINIA   13997    4644    8823    530
47           WASHINGTON    8358    4331    3626    401
48        WEST VIRGINIA    7314    1719    4105   1490
49            WISCONSIN   14307    7289    6031    987
50              WYOMING    3114     920    1964    230
51                 GUAM      10       2       6      2
52          PUERTO RICO    2334     426    1626    282
53  U.S. VIRGIN ISLANDS      24       4      14      6
54               TOTALS  619622  278128  297908  43586
```

1.12.2 Look ahead

Looking ahead to our mapping, we'll be using a GEO file, and each state name will be in the form of `Illinois` or `New Mexico`, etc. However, in the data frame of bridge data, notice that each state name is in all caps. To fix this, we'll use the `.title()` command and return to lesson1:

[13]:
```
'NEW MEXICO'.title()
```

[13]:
```
'New Mexico'
```

[14]:
```
'U.S. VIRGIN ISLANDS'.title()
```

[14]:
```
'U.S. Virgin Islands'
```

1.13 ________________

Let's use this to change the string in the `State` column:

1.14 Try it! ↓

[15]:
```python
def f(x):
    try:
        x = x.title()
    except:
        x = ''
        raise ValueError('Not a string!')
    return x
    #pass # <-- have it return something
dfcounts['State'] = dfcounts['State'].apply(f)
```

Now look at our modified data:

[16]:
```python
dfcounts
```

[16]:

	State	All	Good	Fair	Poor
0	Alabama	16164	6550	9028	586
1	Alaska	1632	716	782	134
2	Arizona	8467	5275	3075	117
3	Arkansas	12941	6234	6028	679
4	California	25737	12224	12020	1493
5	Colorado	8869	3063	5337	469
6	Connecticut	4361	1249	2881	231
7	Delaware	875	291	567	17
8	District Of Columbia	246	74	165	7
9	Florida	12680	8052	4169	459
10	Georgia	14987	11054	3614	319
11	Hawaii	1162	265	810	87
12	Idaho	4561	1322	3001	238
13	Illinois	26846	12848	11593	2405
14	Indiana	19337	7866	10389	1082
15	Iowa	23870	9354	10012	4504
16	Kansas	24925	13335	10313	1277
17	Kentucky	14410	4089	9331	990
18	Louisiana	12782	5931	5220	1631
19	Maine	2485	728	1443	314
20	Maryland	5446	1789	3404	253
21	Massachusetts	5245	1321	3468	456
22	Michigan	11284	4091	5953	1240
23	Minnesota	13496	7857	5021	618
24	Mississippi	16788	9921	5693	1174
25	Missouri	24590	9654	12718	2218
26	Montana	5266	1600	3301	365
27	Nebraska	15348	7966	6102	1280
28	Nevada	2067	1070	968	29
29	New Hampshire	2527	1344	989	194
30	New Jersey	6798	1809	4507	482

```
31            New Mexico    4025    1466    2351    208
32              New York   17555    6355    9528   1672
33        North Carolina   18877    7840    9712   1325
34          North Dakota    4285    2046    1758    481
35                  Ohio   27151   16493    9324   1334
36              Oklahoma   23220    9898   11026   2296
37                Oregon    8235    2800    5053    382
38          Pennsylvania   23166    7705   12263   3198
39          Rhode Island     779     168     475    136
40        South Carolina    9395    4142    4754    499
41          South Dakota    5886    1943    2925   1018
42             Tennessee   20331    8689   10801    841
43                 Texas   55175   27807   26579    789
44                  Utah    3056    1005    1988     63
45               Vermont    2836    1494    1274     68
46              Virginia   13997    4644    8823    530
47            Washington    8358    4331    3626    401
48         West Virginia    7314    1719    4105   1490
49             Wisconsin   14307    7289    6031    987
50               Wyoming    3114     920    1964    230
51                  Guam      10       2       6      2
52           Puerto Rico    2334     426    1626    282
53   U.S. Virgin Islands      24       4      14      6
54                Totals  619622  278128  297908  43586
```

2 Your Turn: Displaying the Data on a Map

Your job is to visualize the information about the bridges on a map (choropleth visualization). To get you started, this section will go through some examples first of some basic maps with other data. Go through these examples to get an understanding for the map visualization then complete the 2 tasks at the bottom.

2.1 Example 1: Our first map

Let's make a map instance with `folium`. Here we set the lat/long coordinates to the Beckman Quad. The tiles parameter is used to determine they style of the map.

(You can use `folium.Map?` in this notebook to experiment with different map types – neat!).

[17]:
```
folium.Map?
```

[18]:
```
m = folium.Map(
    location=[40.114942, -88.226492],
    tiles='Stamen Toner',
    zoom_start=13
)

m
```

[18]: <folium.folium.Map at 0x7f181edc3eb0>

The above example illustrated a map for some specific lat/long coordinates. In practice, we'll want to display some information on the map. We'll start in the next example with a simple visualization of data on a map.

2.2 Example 2: Starting Map with State Data (3 states)

Let's start with a basic visualization of data for 3 states: Iowa, Illinois, and Colorado. Let's assume we have the following dataframe with some data for a few states.

[19]:
```
df = pd.DataFrame(
    [
        ['Iowa', 'Illinois', 'Colorado'],
        [50, 1, 100]],
    index=['State', 'Some Value']
).T

print(df)
```

```
      State Some Value
0      Iowa         50
1  Illinois          1
2  Colorado        100
```

2.2.1 State JSON Info

To plot on map with Folium, a JSON file is used to describe the polygons that will represent each state. For example, Illinois is composed of a list of coordinates – check it out:

[20]:
```
url = 'https://raw.githubusercontent.com/python-visualization/folium/master/
 ↪examples/data/us-states.json'
r = requests.get(url) # grab the source from the url
state_geo = r.json()  # convert to `json`
state_geo['features'][11]
```

[20]:

```
{'type': 'Feature',
 'id': 'ID',
 'properties': {'name': 'Idaho'},
 'geometry': {'type': 'Polygon',
  'coordinates': [[[-116.04751, 49.000239],
    [-116.04751, 47.976051],
    [-115.724371, 47.696727],
    [-115.718894, 47.42288],
    [-115.527201, 47.302388],
    [-115.324554, 47.258572],
    [-115.302646, 47.187372],
    [-114.930214, 46.919002],
    [-114.886399, 46.809463],
```

```
[-114.623506, 46.705401],
[-114.612552, 46.639678],
[-114.322274, 46.645155],
[-114.464674, 46.272723],
[-114.492059, 46.037214],
[-114.387997, 45.88386],
[-114.568736, 45.774321],
[-114.497536, 45.670259],
[-114.546828, 45.560721],
[-114.333228, 45.456659],
[-114.086765, 45.593582],
[-113.98818, 45.703121],
[-113.807441, 45.604536],
[-113.834826, 45.522382],
[-113.736241, 45.330689],
[-113.571933, 45.128042],
[-113.45144, 45.056842],
[-113.456917, 44.865149],
[-113.341901, 44.782995],
[-113.133778, 44.772041],
[-113.002331, 44.448902],
[-112.887315, 44.394132],
[-112.783254, 44.48724],
[-112.471068, 44.481763],
[-112.241036, 44.569394],
[-112.104113, 44.520102],
[-111.868605, 44.563917],
[-111.819312, 44.509148],
[-111.616665, 44.547487],
[-111.386634, 44.75561],
[-111.227803, 44.580348],
[-111.047063, 44.476286],
[-111.047063, 42.000709],
[-112.164359, 41.995232],
[-114.04295, 41.995232],
[-117.027882, 42.000709],
[-117.027882, 43.830007],
[-116.896436, 44.158624],
[-116.97859, 44.240778],
[-117.170283, 44.257209],
[-117.241483, 44.394132],
[-117.038836, 44.750133],
[-116.934774, 44.782995],
[-116.830713, 44.930872],
[-116.847143, 45.02398],
[-116.732128, 45.144473],
[-116.671881, 45.319735],
```

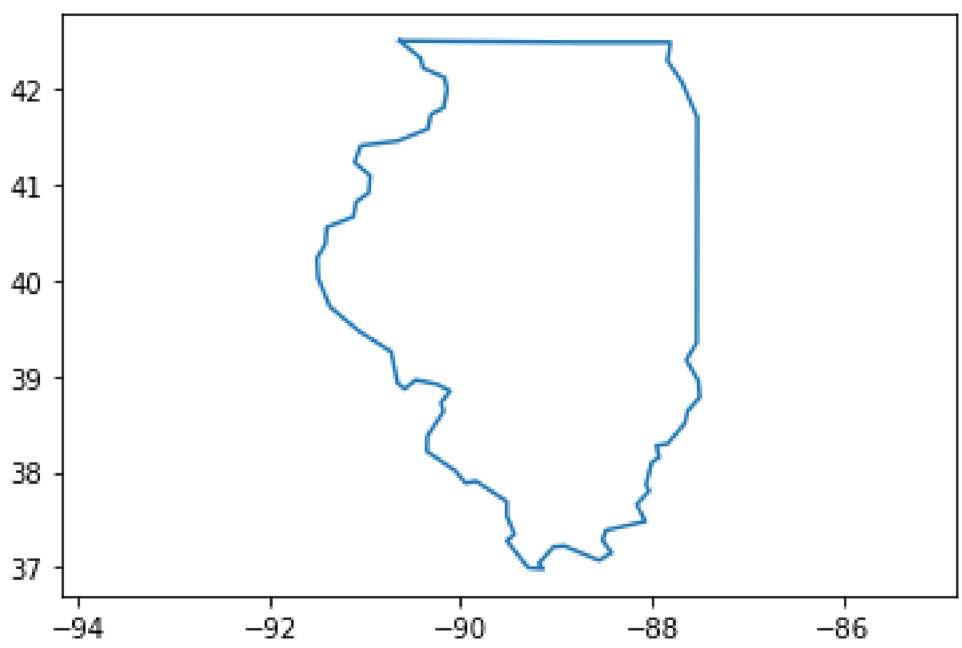
```
[-116.463758, 45.61549],
[-116.545912, 45.752413],
[-116.78142, 45.823614],
[-116.918344, 45.993399],
[-116.92382, 46.168661],
[-117.055267, 46.343923],
[-117.038836, 46.426077],
[-117.044313, 47.762451],
[-117.033359, 49.000239],
[-116.04751, 49.000239]]]}}
```

2.3 Just plot it

We could just plot this polygon data on our own:

[21]:
```
import matplotlib.pyplot as plt
import numpy as np
coords = state_geo['features'][12]['geometry']['coordinates'][0]
coords = np.array(coords)
plt.plot(coords[:,0], coords[:,1])
plt.axis('equal')
```

[21]:
```
(-91.705796, -87.29576399999999, 36.70752035, 42.786376649999994)
```



We printed just Illinois above, but `state_geo` contains the polygons for each of the states.

2.3.1 Creating Map with State Data

To create a Choropleth, we first initialize a map with folium like we did in the basic example above. Then, we call `folium.Choropleth()`.

What is a Choropleth? https://en.wikipedia.org/wiki/Choropleth_map

The call to set the `Choropleth` map has 4 important entries:

- `geo_data=state_geo`, here we set the geo data.
- `data=df`, here we set the data *source* (the stuff we'll plot on each state)
- `columns=['State', 'Some Value']`, where to find the state name or state id and the numbers to visualize (in `data`)
- `key_on='feature.properties.name'`, how the entries are represented in the JSON. For the state geo_data json file, this will be either `'feature.id'` (abbreviation) or `'feature.properties.name'` (name), depending on whether our dataframe uses state names or abbreviations.

The other parameters control specifics of looks of the visualization (opacity, coloring).

[22]:
```
m = folium.Map(location=[44, -102], zoom_start=3)

folium.Choropleth(
    geo_data=state_geo,
    name='choropleth',
    data=df,
    columns=['State', 'Some Value'],
    key_on='feature.properties.name',
    fill_color='Reds',
    fill_opacity=0.6,
    line_opacity=0.2,
).add_to(m)

folium.LayerControl().add_to(m)

m
```

[22]:
```
<folium.folium.Map at 0x7f181d2b8490>
```

2.4 Example 3: Full Example with Unemployment Data

2.5 ___________________________________

For this next part we'll take straight out of the Folium examples.

As before, we use the same json state geo data. This example reads in unemployment data into a pandas dataframe. The call to `folium.Choropleth()` is roughly the same as the above, except this dataframe, `state_data`, uses abbreviations not full names, so `key_on=` a different value. Additionally, the coloring is a little different and the colorbar is labeled.

2.6 Try it! ↓

[23]:
```
state_data.plot()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_7619/4149434812.py in <module>
----> 1 state_data.plot()

NameError: name 'state_data' is not defined
```

[24]:
```
url = 'https://raw.githubusercontent.com/python-visualization/folium/master/
 ↪examples/data/us-states.json'
state_geo = requests.get(url).json()

url = 'https://raw.githubusercontent.com/python-visualization/folium/master/
 ↪examples/data/US_Unemployment_Oct2012.csv'
state_data = pd.read_csv(url)

m = folium.Map(location=[48, -102], zoom_start=5)

folium.Choropleth(
    geo_data=state_geo,
    name='choropleth',
    # vvvv fill this in
    data=state_data,
    columns=['State', 'Unemployment'],
    key_on='feature.id',
    #######
    fill_color='YlGn',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Unemployment Rate (%)'
).add_to(m)

folium.LayerControl().add_to(m)

m
```

[24]: <folium.folium.Map at 0x7f181ee35f10>

3 Your Tasks:

Now that you've seen some examples, it's time for you to visualize the bridge data that is the topic of this lesson.

Your task is to create 2 different choropleth visualizations based on the bridge data (each is described below).

3.1 Task #1:

3.2 ______________________________

Create a choropleth visualization that illustrates the number of bridges in each state.

Hint: Think about what the visualization should look like based on the data. If your visualized results are not as expected, it may help to look at the dataframe to identify what is happening. Notice any rows that aren't simply state data? ## Try it! ↓

[25]:
```
dfcounts.drop(54, inplace=True)
```

[26]:
```
dfcounts
```

[26]:

	State	All	Good	Fair	Poor
0	Alabama	16164	6550	9028	586
1	Alaska	1632	716	782	134
2	Arizona	8467	5275	3075	117
3	Arkansas	12941	6234	6028	679
4	California	25737	12224	12020	1493
5	Colorado	8869	3063	5337	469
6	Connecticut	4361	1249	2881	231
7	Delaware	875	291	567	17
8	District Of Columbia	246	74	165	7
9	Florida	12680	8052	4169	459
10	Georgia	14987	11054	3614	319
11	Hawaii	1162	265	810	87
12	Idaho	4561	1322	3001	238
13	Illinois	26846	12848	11593	2405
14	Indiana	19337	7866	10389	1082
15	Iowa	23870	9354	10012	4504
16	Kansas	24925	13335	10313	1277
17	Kentucky	14410	4089	9331	990
18	Louisiana	12782	5931	5220	1631
19	Maine	2485	728	1443	314
20	Maryland	5446	1789	3404	253
21	Massachusetts	5245	1321	3468	456
22	Michigan	11284	4091	5953	1240
23	Minnesota	13496	7857	5021	618
24	Mississippi	16788	9921	5693	1174
25	Missouri	24590	9654	12718	2218
26	Montana	5266	1600	3301	365
27	Nebraska	15348	7966	6102	1280
28	Nevada	2067	1070	968	29
29	New Hampshire	2527	1344	989	194
30	New Jersey	6798	1809	4507	482
31	New Mexico	4025	1466	2351	208

```
32                New York  17555   6355   9528  1672
33          North Carolina  18877   7840   9712  1325
34            North Dakota   4285   2046   1758   481
35                    Ohio  27151  16493   9324  1334
36                Oklahoma  23220   9898  11026  2296
37                  Oregon   8235   2800   5053   382
38            Pennsylvania  23166   7705  12263  3198
39            Rhode Island    779    168    475   136
40          South Carolina   9395   4142   4754   499
41            South Dakota   5886   1943   2925  1018
42               Tennessee  20331   8689  10801   841
43                   Texas  55175  27807  26579   789
44                    Utah   3056   1005   1988    63
45                 Vermont   2836   1494   1274    68
46                Virginia  13997   4644   8823   530
47              Washington   8358   4331   3626   401
48           West Virginia   7314   1719   4105  1490
49               Wisconsin  14307   7289   6031   987
50                 Wyoming   3114    920   1964   230
51                    Guam     10      2      6     2
52             Puerto Rico   2334    426   1626   282
53     U.S. Virgin Islands     24      4     14     6
```

[27]:

```python
url = 'https://raw.githubusercontent.com/python-visualization/folium/master/
 ↪examples/data/us-states.json'
state_geo = requests.get(url).json()

m = folium.Map(location=[48, -102], zoom_start=3)

folium.Choropleth(
    geo_data=state_geo,
    name='choropleth',
    # fill this in vvvv
    data=dfcounts,
    columns=['State', 'All'],
    key_on='feature.properties.name',
    fill_color='YlGn',
    fill_opacity=0.7,
    line_opacity=0.2,
).add_to(m)

folium.LayerControl().add_to(m)

m
```

[27]: <folium.folium.Map at 0x7f181db418b0>

3.3 Task #2

3.4 ________________________________

Create a different choropleth visualization that illustrates the percentage of bridges in each state rated as poor.

Hint: this value is not currently in the dataframe of bridge data – you'll need to compute it first

3.5 Try it! ↓

[28]:
```
dfcounts['Percent in Poor Condition'] = dfcounts['Poor']/dfcounts['All'] * 100
dfcounts['Percent in Fair Condition'] = dfcounts['Fair']/dfcounts['All'] * 100
dfcounts['Percent in Good Condition'] = dfcounts['Good']/dfcounts['All'] * 100
```

[29]:
```
m = folium.Map(location=[48, -102], zoom_start=3)

folium.Choropleth(
    geo_data=state_geo,
    name='choropleth',
    data=dfcounts,
    columns=['State', 'Percent in Poor Condition'],
    key_on='feature.properties.name',
    fill_color='YlGn',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Percent of Bridges Ranked as Poor (%)'
).add_to(m)

folium.LayerControl().add_to(m)
m
```

[29]:
```
<folium.folium.Map at 0x7f181dad9400>
```

[30]:
```
m = folium.Map(location=[48, -102], zoom_start=3)

folium.Choropleth(
    geo_data=state_geo,
    name='choropleth',
    data=dfcounts,
    columns=['State', 'Percent in Fair Condition'],
    key_on='feature.properties.name',
    fill_color='YlGn',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Percent of Bridges Ranked as Poor (%)'
).add_to(m)

folium.LayerControl().add_to(m)
```

```
m
```

[30]: <folium.folium.Map at 0x7f181db29fd0>

[31]:
```
m = folium.Map(location=[48, -102], zoom_start=3)

folium.Choropleth(
    geo_data=state_geo,
    name='choropleth',
    data=dfcounts,
    columns=['State', 'Percent in Good Condition'],
    key_on='feature.properties.name',
    fill_color='YlGn',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Percent of Bridges Ranked as Poor (%)'
).add_to(m)

folium.LayerControl().add_to(m)
m
```

[31]: <folium.folium.Map at 0x7f181d32e490>

[ ]: