Cisco Networking Academy® Mind Wide Open™

Lab - Advanced Data Visualization

Objectives

- **Part 1: Prepare the data**
- **Part 2: Visualize the data**

Scenario/Background

In this lab, you will learn how to combine an `SQLite` database, `JSON` files, and `pandas` `DataFrame` s. You will interface with a wrapper for the library `folium` , that enables you to plot data on a geographical map. You will produce a map of the United Kingdom divided in areas, each colored in a shade that is proportional to the internet speed, a very powerful way of understanding the data. With such a map, it will be very clear how internet speed vaires across England. You will learn how to find and fix problems in the data.

Required Resources

- 1 PC with Internet access
- Raspberry Pi version 2 or higher
- Python libraries: `pandas` , `numpy` , `sqlite3` , `matplotlib` , `folium_utils`
- Datafiles: LA_Poligons.json

Part 1: Prepare the data

To use the `folium` library, we will modify the data to get it in the right format. At the moment, the data is in an SQLite database and we want to have it in a `pandas` `DataFrame` . A few more cahnges also need to be made.

Step 1: Get the data and store it in a `pandas` `DataFrame` .

First, we need to get the data. We are going to connect to the InternetSpeed database.

a) Import the libraries.

- `pandas`
- `numpy`
- `sqlite3`
- `pyplot` (and use the style fivethirtyeight, or another one if you prefer)

- `folium_utils`

In [30]:
```python
# Code Cell 1
import pandas as pd, numpy, sqlite3, matplotlib.pyplot, folium

%matplotlib inline
```

b) Connect to the database.

Now we need to go back to the database we created and populated in the second lab of Chapter 2:

```
./Data/InternetSpeed.db
```

A copy of the database is stored in your Chapter 5 folder. Connect to the database.

In [31]:
```python
# Code Cell 2
# Create the connection to the database
conn = sqlite3.connect('./Data/InternetSpeed.db')
# Create a cursor
cur = conn.cursor()
```

The database contains the table `average_speed` with the average ping time, upload speed, and download speed for the different Local Authorities in England. The first step in this lab is to read the data in the table into a `pandas` `DataFrame` with the function `.read_sql()` .

This function needs a string containing the query to be executed and the variable that contains the connection to the database (see [http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_sql.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_sql.html)](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_sql.html) for details). Assuming the query is stored in the variable `query` , the call

```
pd.read_sql(query)
```

will return a `pandas` `DataFrame` .

c) Store the data in a `DataFrame` .

1. Write the query to select all the data in the table `average_speed` .
2. Use `pandas` to put the results of the query into a `DataFrame` .

In [32]:
```python
# Code Cell 3
# Create a query to select all the data in the table average_speed
query = "SELECT * FROM average_speed;"

# Read the the data from the table into a DataFrame.
#params: list, tuple/dic
df = pd.read_sql(query,conn)
```

Step 2: Clean the data.

Using the `pandas` functions learned so far, explore the produced `Dataframe` and remove anv unwanted columns.

a) Explore the data.

Print the first few rows of the `DataFrame` df.

In [33]:
```
# Code Cell 4
df.head()
```

Out[33]:

	index	Area	Average_p	Average_u	Average_d
0	0	E07000188	8.451897	21.114114	51.967713
1	1	E07000101	8.317833	20.733028	51.709226
2	2	E09000030	8.983481	22.469438	54.412001
3	3	E09000031	8.956481	22.353593	55.138017
4	4	E09000032	9.144285	22.859003	55.962499

The column `index` is not needed for this exercise.

b) Clean the data.

Drop the column `index` from the `DataFrame`. Remember that the change is effective if the parameter `inplace` is set to True or if you assign the modified `DataFrame` to a new `DataFrame`.

In [34]:
```
# Code Cell 5
df.drop(["index"], inplace=True, axis=1)

print(df.shape)
df.head()
```

```
(326, 4)
```

Out[34]:

	Area	Average_p	Average_u	Average_d
0	E07000188	8.451897	21.114114	51.967713
1	E07000101	8.317833	20.733028	51.709226
2	E09000030	8.983481	22.469438	54.412001
3	E09000031	8.956481	22.353593	55.138017
4	E09000032	9.144285	22.859003	55.962499

Step 3: Learn about `JSON` files.

The map will be created by drawing the borders of each Local Authority and coloring them according to a colormap. The color coded map will indicate the relative Internet speed of an area with respect to other areas.

Each area is represented by a polygon, which will be drawn on our map. The edges of each polygon represent the borders of each Local Authority. A `JSON` file is provided, containing the coordinates of the borders of each Local Authority.

`JSON` is a popular alternative to XML, and it is a format used to store and exchange data (to learn more about `JSON` see [http://www.w3schools.com/js/js_json_intro.asp (http://www.w3schools.com/js/js_json_intro.asp)](http://www.w3schools.com/js/js_json_intro.asp)).

a) Open and observe the file.

```
LA_poligons.json
```

Similar to the functionalities provided for reading `csv` files and `SQL` tables, it is possible to read the content of a `JSON` file in a `pandas` `DataFrame` with the method `read_json()` . This method takes an input string containing the path to the file we intend to read (see this documentation for more details: [http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_json.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_json.html)](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_json.html)).

b) Read `LA_poligons.json` into a `pandas` `DataFrame` , and then visualize the first few rows.

FIX: save the right path to json data in to url_la_json variable. Later you will use this path

In [35]:

```python
# FIXED

# Code Cell 6
# save LA_poligons_fixed.json path to variable url_la_json
url_la_json = "LA_poligons_fixed.json"

# Read the JSON file into a DataFrame
la_json = pd.read_json(r'LA_poligons_fixed.json')

# Visualize the first rows of la_json DataFrame
la_json.head()
```

Out[35]:

	type	features
0	FeatureCollection	{'type': 'Feature', 'geometry': {'type': 'Mult...
1	FeatureCollection	{'type': 'Feature', 'geometry': {'type': 'Mult...
2	FeatureCollection	{'type': 'Feature', 'geometry': {'type': 'Mult...
3	FeatureCollection	{'type': 'Feature', 'geometry': {'type': 'Mult...
4	FeatureCollection	{'type': 'Feature', 'geometry': {'type': 'Poly...

~~The information contained in this `Dataframe` will be used later in the visualization.~~

Step 4: Average the ping speed.

For each map that will be produced, only one of the three Internet speed indicators can be visualized (average ping, upload speed, or download speed). In this lab, the focus will be on the average ping speed, but the student is encouraged to repeat the exercise to visualize upload and download speeds.

a) Prepare the data.

Create a new `pandas` `DataFrame` that contains only two columns: `Area` and `Average_p` , selecting them from the `DataFrame` that contains the `InternetSpeed` database data. Now rename the column `Area` to `LA_code` , using the method `rename()` . You will need to pass to the method the field `columns` with this code:

```
columns={'Area':'LA_code'}
```

1. Create the new `DataFrame` .
2. Rename the columns.
3. Display the first few lines of the new `DataFrame` .

In [41]:

```
# Code Cell 7
dfp = df.loc[:,["Area","Average_p"]]

dfp=dfp.rename(columns={'Area':'LA_code'})

dfp.head()
```

Out[41]:

	LA_code	Average_p
0	E07000188	8.451897
1	E07000101	8.317833
2	E09000030	8.983481
3	E09000031	8.956481
4	E09000032	9.144285

By default, `folium` uses only 6 different colors to define a colormap. Rather than defining a custom colormap, the data of the Internet speed indicator of interest will be divided into 6 bins, and each bin will be associated to a different color. The bins must be carefully defined, in order to maximize the differences between the different areas. To choose the bins, it's a good idea to have a look at the range of the data.

So far, the columns of a `pandas` `DataFrame` have been accessed in the following way:

```
df['Name_of_the_column']
```

An alternative way to do so is:

```
df.Name_of_the_column
```

In [42]:
```
# Code Cell 8
print(dfp.Average_p.min())
print(dfp.Average_p.max())
```

```
0.0
11.999447541819524
```

The ping time is roughly contained in the interval 0-12. If the bins are chosen to be [0, 10, 20, 30, 40, 50], all the data would fall in the first bin and all the areas would be represented with the first color. A good starting point is to take the range in which the data lives and divide it in six different parts.

To create the bins, use the `numpy` method `arange()` . This method takes as input the minimum value of the range, the maximum value of the range, and the step. If 6 bins in the range 1 to 12 are required, the function call would look like:

```
np.arange(1, 12, (12-1)/6)
```

b) Select good bins for visualization.

Create a range of six values that goes from the minimum to the maximum value of the average ping speed and cast it to a list (`list(p_bins)`).

Note: Make sure you have all the data values in side your bin range

In [54]:
```
# Code Cell 9
p_bins = numpy.arange(dfp.Average_p.min(), dfp.Average_p.max()+0.1, (dfp.A

p_bins = list(p_bins)

print(p_bins)
```

```
[0.0, 1.9999079236365873, 3.9998158472731746, 5.999723770909762, 7.9996316945
46349, 9.999539618182936, 11.999447541819524]
```

Part 2: Visualize the data

In the folder that contains the code, there is a Python script that contains a few functions that will help produce a data visualization on a map. The main library it uses is `folium` . The library documentation can be found here: http://python-visualization.github.io/folium/ (http://python-visualization.github.io/folium/) ~~https://folium.readthedocs.io/en/latest/ (https://folium.readthedocs.io/en/latest/)~~).

You can import the functions in a Python script just like any other library.

Step 1: Plot the data on a map.

You use Folium's Chrolopleth function an not folium_utils.

a) Learn about the `folium_utils` code.

Open the Python source file `folium_utils.py` and read through it. This script contains a few functions that make use of the `folium` library.

Can you identify the main function that we are going to use later in the notebook?

This code wraps what a user should write as multiple functions into one with a simpler interface.

To produce the first map, use the function

```
folium_top_x_preds_mapper()
```

contained in `folium_utils` .

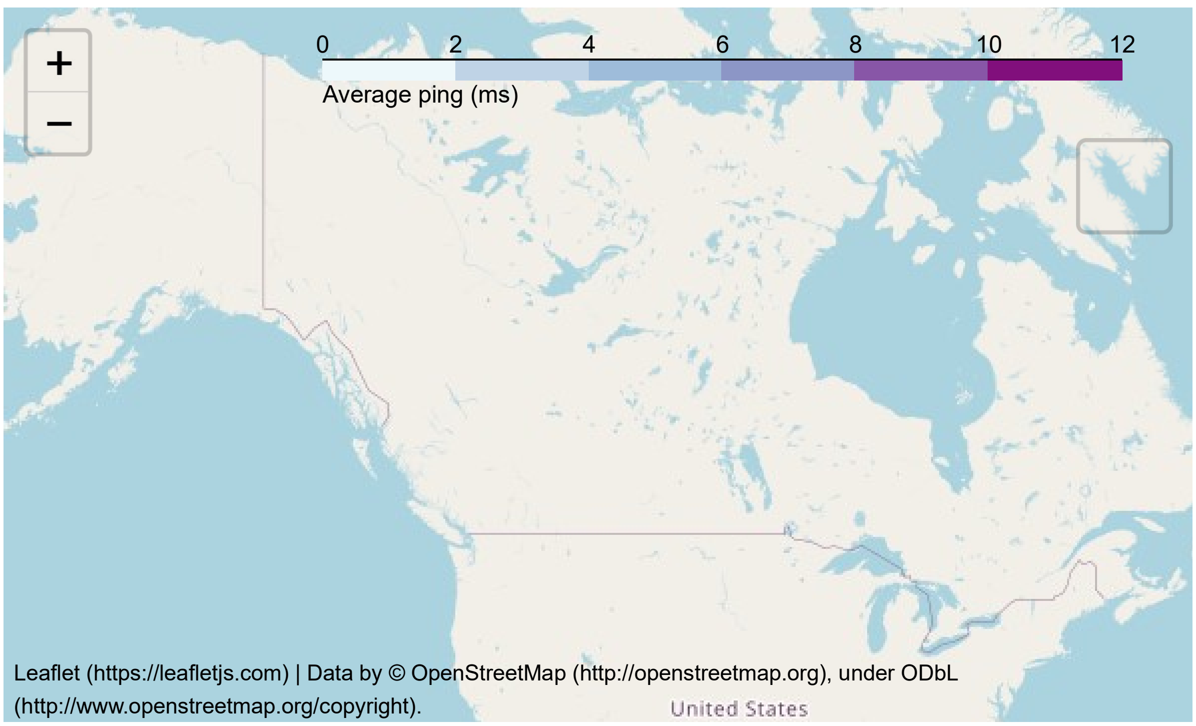
b) Call the function `folium_top_x_preds_mapper()` with the following parameters:

- The `Dataframe` that contains the data to be visualized (`dfp`)
- The `Dataframe` that contains the coordinates of the polygons (`la_json`)
- The name of the column on which to merge the two `DataFrames` (LA_code)
- The name of the column that contains the data to plot (Average_p)
- The list containing the limits of the bins (`p_bins`)

In [55]:

```
# FIXED

# Code Cell 10
# Create the map
mymap = folium.Map(location=[54, -1], zoom_start=6)

folium.Choropleth(
    geo_data=url_la_json, #geometry data can use path to GeoJSON file
    name="choropleth", #layer name -not in real use here
    data=dfp, # data table for the values
    columns=["LA_code", "Average_p"], # key (to bind with geometry), value
    key_on="properties.LA_code", # key on geometry
    fill_color="BuPu", # color
    fill_opacity=1,
    line_opacity=0.2,
    nan_fill_opacity=0.0,
    legend_name="Average ping (ms)",
    bins=p_bins,
    reset=True,
).add_to(mymap)

folium.LayerControl().add_to(mymap)

mymap
```

Out[55]:



~~What happened? The number of Local Authorities contained in the DataFrame dfp is bigger than the one contained in the la_json. In this case, folium would fail to visualize the map, resulting in an error. In folium_utils, a check was added to prevent this from happening, and the error message Length mismatch is visualized. But why is this happening?~~

Notting special happened, even if there is some michmatches in the data.

Let's examine the data:

c) Plot the column Average_p.

`pandas` provides the basic functionalities to generate plots starting from `DataFrame` s. The plot method on `Series` and `DataFrame` is just a simple wrapper around plt.plot(), but it saves a few lines of code.

You can call the method `plot()` directly on the column you want to plot (`dfp.Average_p.plot()`).

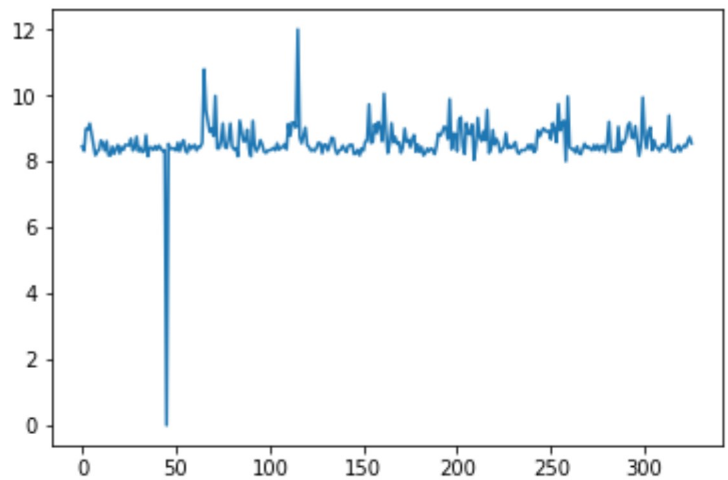
(See http://pandas.pydata.org/pandas-docs/version/0.18.1/visualization.html (http://pandas.pydata.org/pandas-docs/version/0.18.1/visualization.html) for more details about

In [57]:

```
# Code Cell 11
# Plot the column Average_p
dfp.Average_p.plot()
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x9e384550>
```



The visualization shows that there is a Local Authority for which the average internet speed is zero. Because it is clearly an anomaly, it is necessary to have a closer look.

d) Select the row in which the average speed is zero from the `DataFrame` .

In [71]:

```
# Code Cell 12
dfp.loc[dfp["Average_p"] == 0]
```

Out[71]:

	LA_code	Average_p
45	DateTime	0.0

DateTime is not the name of a Local Authority: there was a mistake in collecting or storing the data.

Step 2: Fix the data and plot the data on a map again.

a) Remove the wrong row from your `DataFrame`

In [82]:

```
# Code Cell 13
# hint: you can use the index (45) to remove the row
#dfp.drop([45], inplace=True)

# Plot here the cleaned data
dfp.head(50)
```

Out[82]:

	LA_code	Average_p
0	E07000188	8.451897
1	E07000101	8.317833
2	E09000030	8.983481
3	E09000031	8.956481
4	E09000032	9.144285
5	E09000033	8.826062
6	E07000123	8.534960
7	E07000047	8.185066
8	E07000212	8.302110
9	E07000010	8.342006
10	E07000011	8.631665
11	E07000012	8.520717
12	E07000164	8.325232
13	E07000165	8.610279
14	E07000166	8.188387
15	E07000167	8.173249
16	E07000221	8.448097
17	E07000163	8.227297
18	E07000098	8.377899
19	E07000099	8.494437
20	E07000223	8.241653
21	E07000094	8.432513
22	E07000095	8.344410
23	E07000096	8.502912
24	E07000097	8.495186
25	E07000090	8.482718
26	E07000091	8.684208
27	E07000092	8.328357
28	E07000093	8.417244
29	E06000032	8.747576

	LA_code	Average_p
30	E07000044	8.306130
31	E07000238	8.419876
32	E07000156	8.279978
33	E07000119	8.307542
34	E07000154	8.793760
35	E07000191	8.154217
36	E07000190	8.418205
37	E07000193	8.421844
38	E07000192	8.344268
39	E07000195	8.464318
40	E07000194	8.355531
41	E07000197	8.479471
42	E07000196	8.383260
43	E07000199	8.318568
44	E07000198	8.341121
46	E07000222	8.529148
47	E07000061	8.379333
48	E07000118	8.393854
49	E07000063	8.378082

Now that the data looks better, we can plot the map again by calling the function in the folium wrapper. We need however, stop for a second and pay attention.

Attention! (don't use chrome browser)

If using Chrome, there is currently a limit on the size of map that can be correctly visualized inside a notebook. To this end, we need to limit the size of the `dfp` `DataFrame` to the first 50 rows. The discrepancy in the size of the `dfp` and `la_json` `DataFrame` 's in this case will not cause an error, because `dfp` cannot be bigger than `la_json` , but it can be smaller. This is operation is not needed if using Firefox.

In [ ]:
```
# resize the dfp DataFrame. This operation is not needed if using Firefox
#dfp = dfp.iloc[0:51]
```

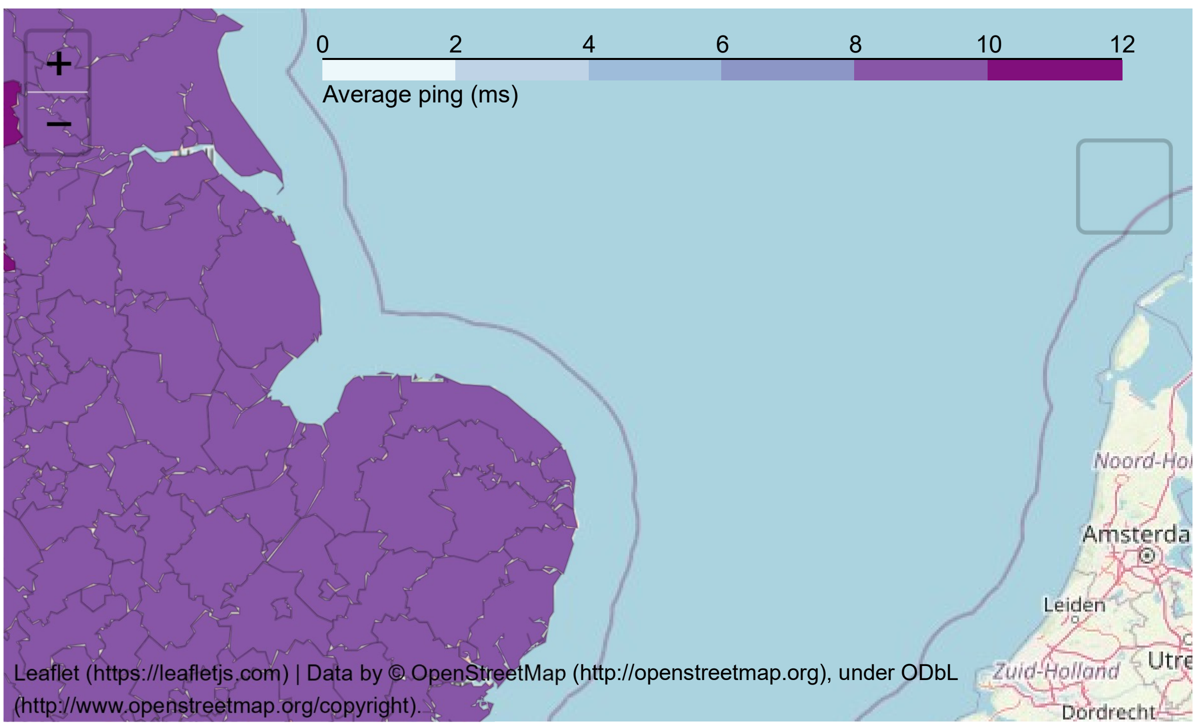
b) Draw the map again with like in code cell 10

~~b) Call the function folium_top_x_preds_mapper() again , with the same parameters as before.~~

In [83]:

```
# Code Cell 14
mymap = folium.Map(location=[54, -1], zoom_start=6)

folium.Choropleth(
    geo_data=url_la_json, #geometry data can use path to GeoJSON file
    name="choropleth", #layer name -not in real use here
    data=dfp, # data table for the values
    columns=["LA_code", "Average_p"], # key (to bind with geometry), value
    key_on="properties.LA_code", # key on geometry
    fill_color="BuPu", # color
    fill_opacity=1,
    line_opacity=0.2,
    nan_fill_opacity=0.0,
    legend_name="Average ping (ms)",
    bins=p_bins,
    reset=True,
).add_to(mymap)

folium.LayerControl().add_to(mymap)

mymap
```

Out[83]:



If you have resized the `dfp` `DataFrame`, you should see a map with only one color for the 6 to 8 bin. If you haven't resized it, you should also see the 8 to 10 bin. What happened there? 23 The value 0 was removed, because it was found to be an anomaly, but the range of the bins for the colormap was not changed. 4 What is the new minimum value for `Average_p`?

In [84]:

```
# Code Cell 15
print(dfp.Average_p.min())
```

```
8.00433373382758
```

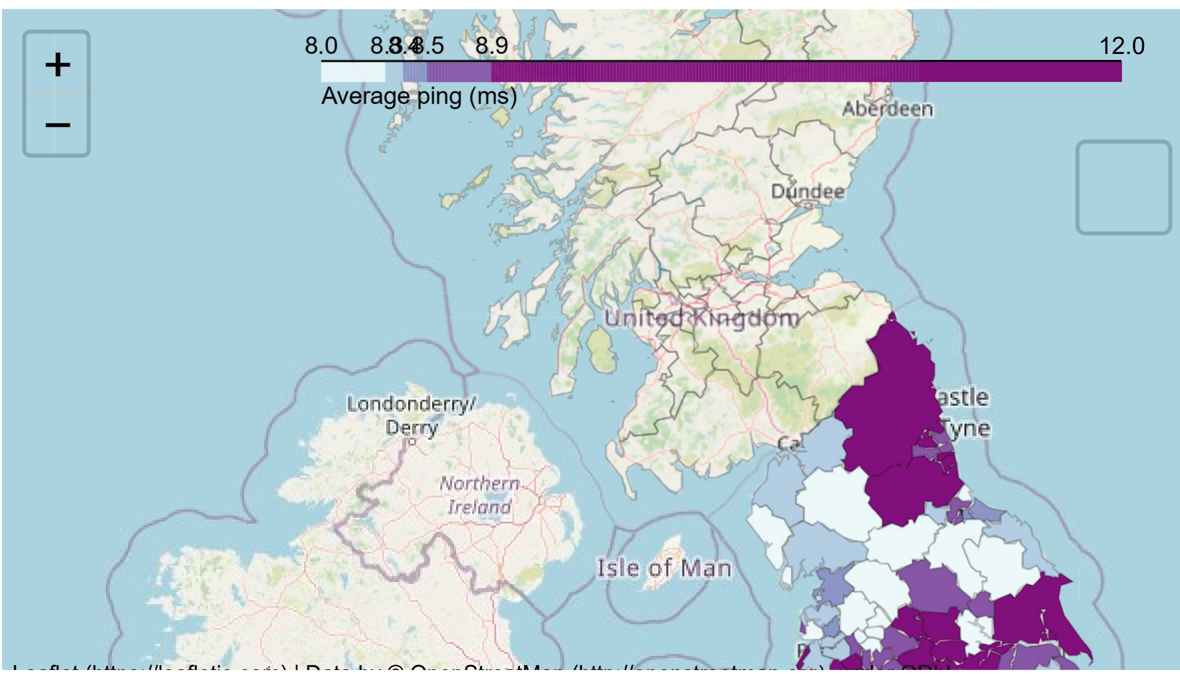
It is necessary to repeat the creation of bins with the cleaned dataset. After this, draw the map again ~~call the function folium_top_x_preds_mapper()~~ to correctly visualize the average ping speed across the UK.

c) Recreate the bin range, recreate the map, and visualize it.

In [91]:

```
# FIXED

# Code Cell 16
#p_bins = numpy.arange(dfp.Average_p.min(), dfp.Average_p.max()+1, (dfp.Av

p_bins = numpy.quantile(dfp["Average_p"], [0,0.2,0.4,0.6,0.8,1])
#p_bins = np.percentile(dfp["Average_p"], [0,20,40,60,80,100])

p_bins = list(p_bins)

#draw the map again as previously. You can try different methods to create

mymap = folium.Map(location=[54, -1], zoom_start=6)

folium.Choropleth(
    geo_data=url_la_json, #geometry data can use path to GeoJSON file
    name="choropleth", #layer name -not in real use here
    data=dfp, # data table for the values
    columns=["LA_code", "Average_p"], # key (to bind with geometry), value
    key_on="properties.LA_code", # key on geometry
    fill_color="BuPu", # color
    fill_opacity=1,
    line_opacity=0.2,
    nan_fill_opacity=0.0,
    legend_name="Average ping (ms)",
    bins=p_bins,
    reset=True,
).add_to(mymap)

folium.LayerControl().add_to(mymap)

mymap
```

Out[91]:



Much better!

The visualization of the data with a simple line plot lets us spot an error very easily. Sometimes a deeper investigation is needed.

For additional practice, repeat the exercise, but create the visualizations for the columns