Web Mapping with Python and Leaflet
(/en/lessons/mapping-with-python-leaflet)

Kim Pham

This tutorial teaches users how to create a web map based on tabular data.

Peer-reviewed (https://github.com/programminghistorian/ph-submissions/issues/85)

CC-BY 4.0 (https://creativecommons.org/licenses/by/4.0/deed.en)

Contents

- Lesson Goals
  - Getting Started
    - Initial Setup
    - Getting Data: Download the CSV
  - Geocoding with Python
    - Geocode the placenames in the CSV using Geopy, Pandas
  - Transforming Data with Python
    - Making GeoJSON
  - Using Leaflet to Create a Web Map
    - I now have clean GeoJSON data. Let's make a map!
    - What did I just make?
      - mymap.html walkthrough
      - style.css walkthrough
Lesson Goals

In this lesson, you will learn how to create a web map based on that data. By the end of this lesson, you will be able to:

- Manipulate tabular data programmatically to extract geonames and create location-based data
- Convert tabular data into a meaningful geographic data structure
- Understand and apply the basic concepts of web mapping to design your own web map

Getting Started

Initial Setup

This lesson uses:

- leaflet (http://leafletjs.com/)
- geojson.io (from mapbox) (http://geojson.io/)
- javascript (https://www.javascript.com/) and jquery (https://jquery.com/)

Optional: If you wish to follow along with pre-made scripts you can find them in this directory (https://github.com/programminghistorian/jekyll/tree/gh-pages/assets/mapping-with-python-leaflet).

To set up your working environment:

1. Create a directory for this project where you will keep all of your scripts and files that you will work from
2. If you have a text editor where you can work from the directory of your project, import that directory. You can use editors such as TextWrangler (http://www.barebones.com/products/textwrangler/) for OS X, Notepad++ (https://notepad-plus-plus.org/) for Windows, or Sublime Text (http://www.sublimetext.com/). If you are using a code editor such as Sublime Text, to import the folder you could drag and drop the folder that you want to work from into your editor window. Once you’ve done that, the directory will appear on the left hand sidebar as your root folder. If you click on your folder, you’ll be able to see the contents of your
folder. Importing a folder allows you to easily work with the files in your project. If you need to work with multiple files and directories in directories, this will make it easier to search through these files, switch between them while you’re working and keep you organized.

3. (Optional) It is recommended to use a Python virtual environment (http://docs.python-guide.org/en/latest/dev/virtualenvs/) to store the dependencies and versions required for your specific project.

Getting Data: Download the CSV

We’re going to start with a plain comma-separated values (CSV) data file and create a web map from it.

[The data file can be downloaded here.] (/assets/mapping-with-python-leaflet/census.csv). You can grab this by either opening the link in your browser and saving the page, or you can use the curl command from your command line:

```bash
curl -O https://programminghistorian.org/assets/mapping-with-python-leaflet/census.csv
```

The original source of this data is from the Greater London Authority London Datastore (http://data.london.gov.uk/dataset/historic-census-population).

Geocoding with Python

Geocode the placenames in the CSV using Geopy, Pandas

Now that we have data, we will use this as our source to make a web map. Web maps typically represent locations and features from geographic data formats such as geoJSON and KML. Every location in a geographic data file can be considered to have geometry (such as points, lines, polygons) as well as additional properties. Web maps typically understand locations as a series of coordinates. For example, 43.6426,-79.3871 would represent the exact coordinates of the CN Tower in Toronto (https://en.wikipedia.org/wiki/CN_Tower).

In our data file, we have a list of placenames in our CSV data (the Area Name column), but no coordinates. What we want to do then is to somehow generate coordinates from these locations. This process is called geocoding.

So here is our first problem to solve: how can we geocode placenames? How could we take an entry such as “CN Tower” and add the coordinates 43.6426,-79.3871 to it automatically?

To clarify, we need to figure out how to gather coordinates for a location for each row of a CSV file in order to display these locations on a web map.

There’s a simple way to do this: you can look up a coordinate online in Google Maps and put each coordinate in your spreadsheet manually. But, if you had 5,000 points the task becomes a little bit more daunting. If you’re faced with a repetitive task, it might be worthwhile to approach it...
If you’re familiar with *Programming Historian*, you might have already noticed that there are many lessons available on how to use Python ([/lessons/?topic=python]). Python is a great beginner programming language because it is easy to read and happens to be used a lot in GIS applications to optimize workflows. One of the biggest advantages to Python is the impressive amount of libraries which act like pluggable tools to use for many different tasks. Knowing that this is a good programmatic approach, we’re now going to build a Python script that will automate geocode every address for us.

**Geopy** ([https://github.com/geopy/geopy](https://github.com/geopy/geopy)) is a Python library that gives you access to the various geocoding APIs. Geopy makes it easy for Python developers to locate the coordinates of addresses, cities, countries, and landmarks across the globe using third-party geocoders and other data sources. Geopy includes geocoders built by OpenStreetMap Nominatim, ESRI ArcGIS, Google Geocoding API (V3), Baidu Maps, Bing Maps API, Yahoo! PlaceFinder, Yandex, IGN France, GeoNames, NaviData, OpenMapQuest, What3Words, OpenCage, SmartyStreets, geocoder.us, and GeocodeFarm geocoder services.

**Pandas** ([http://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe](http://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe)) is another python library that we will use. It’s very popular library amongst scientists and mathematicians to manipulate and analyse data.

Finally, **Pip** ([http://pip.readthedocs.org/en/stable/](http://pip.readthedocs.org/en/stable/)) is a very useful package manager to help you install things like Geopy and Pandas! If you’ve already installed Python ([/lessons/introduction-and-installation]) and installed pip ([/lessons/installing-python-modules-pip]), type `pip list` to see if you already have the geopy and pandas packages installed. If you do not have pip installed, you can download [get-pip.py](https://bootstrap.pypa.io/get-pip.py), then from your command line go to the directory where get-pip.py is located and run

```
python get-pip.py
```


To install Geopy and Pandas, open your command line (using this lesson as a guideline if necessary) ([/lessons/intro-to-bash]) and install the Geopy and Pandas libraries:

```
On OS X or Linux, the following commands will install the necessary packages:

```
pip install numpy
pip install python-dateutil
pip install pytz
pip install geopy
pip install pandas
```

Note: We are installing numpy, python-dateutil, and pytz because pandas [requires them](http://pandas.pydata.org/pandas-docs/stable/install.html#dependencies).
For Windows, you may need to install Microsoft Visual C++ Compiler for Python (for 2.7 you can download it from Microsoft (http://www.microsoft.com/en-us/download/details.aspx?id=44266)). Set the environmental variables to recognize python and pip from the command line:

```bash
setx PATH "%PATH%;C:\Python27"
setx PATH "%PATH%;C:\Python27\Scripts"
```

If you keep getting an error when you’re trying to install these libraries, you may need to use `sudo pip install` instead of just `pip install`. You may also need to upgrade your libraries if you’ve installed them earlier and you find that you’re encountering an error when using Python (i.e. an ImportError). In order to do so, the following command works:

```bash
pip install python --upgrade
```

Repeat for the other dependencies listed above.

Now we’re going to start building our script. We’re going to go through the script, line by line, and then run it through the command line at the end. To get started, open your text editor and save your blank document as a python script (name it `geocoder.py`). For the first part of your Python script, you will want to import your libraries and your data:

```python
import geopy
import pandas
from geopy.geocoders import Nominatim, GoogleV3
# versions used: geopy 1.10.0, pandas 0.16.2, python 2.7.8
```

In the code above, we are importing the different Python libraries that we will need to use later on in our script. We import `geopy`, specifically the `geopy.geocoders` that we will call on later which is Nominatim (https://wiki.openstreetmap.org/wiki/Nominatim) and Google Maps V3 API (https://developers.google.com/maps/documentation/geocoding/start), and we import `pandas`.

Then you want to create a function `main()` that reads your input CSV:

```python
def main():
    io = pandas.read_csv('census-historic-population-borough.csv',
                         index_col=None, header=0, sep='","
```

We are first using `pandas`’ pre-existing `read_csv()` function to open the CSV file. We pass the filepath to our data file in the first parameter, ‘census-historic-population-borough.csv’. If it was in a folder called ‘data’, you would put ‘data/census-historic-population-borough.csv’. The second parameter, `index_col=None`, will number the rows to generate the index without using any column. If we use `index_col=0`, it indexes the first column in your data as the row name. The third parameter, `header=0`, indicates that there is a header row, which is the first line of the spreadsheet (Note: Python uses “0” instead of “1” to indicate the first value in an index). The
fourth parameter `sep=','` is where you indicate delimiter symbol that is used to split data into fields. Since are using a comma separated values data format, we need to indicate that we are using a comma to split our data.

There are many other parameters you can use. A full list is available in the pandas documentation on the read_csv() function (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html).

Next, we anticipate that when we geocode the csv we will get points in the format of (latitude, longitude). If we only want the latitude value of the point in a csv column, we will define a function to isolate that value. The same can be done for our longitude value.

```python
import geopy
import pandas

# versions used: geopy 1.10.0, pandas 0.16.2, python 2.7.8

def main():
    io = pandas.read_csv('census-historic-population-borough.csv',
                         index_col=None, header=0, sep=',,')

def get_latitude(x):
    return x.latitude

def get_longitude(x):
    return x.longitude
```

Next, select the geolocator you want to use. Here we’re creating two geolocators: Open Street Map’s Nominatim and Google’s Geocoding API. Here’s a quick comparison:

<table>
<thead>
<tr>
<th>Geolocator</th>
<th>Nominatim()</th>
<th>GoogleV3()</th>
</tr>
</thead>
<tbody>
<tr>
<td>affiliation</td>
<td>OpenStreetMap</td>
<td>Google</td>
</tr>
<tr>
<td></td>
<td>single-threaded</td>
<td></td>
</tr>
<tr>
<td></td>
<td>applications, can</td>
<td></td>
</tr>
<tr>
<td></td>
<td>only run geolocator one process at a time</td>
<td>can upgrade for better performance</td>
</tr>
<tr>
<td>capabilities for app development</td>
<td>can geocode based on user-input</td>
<td>only geocodes static addresses (Google's non-static geocoding service not in geopy)</td>
</tr>
<tr>
<td>request limit</td>
<td>1 request/s or timeout</td>
<td>5 requests/s, 2500/day</td>
</tr>
<tr>
<td>performance test on census data</td>
<td>33.5s</td>
<td>11.6s</td>
</tr>
</tbody>
</table>

You can also choose a different geolocator from the list found in the geopy documentation (http://geopy.readthedocs.org/). Generally, GoogleV3 is a reliable geolocator choice because of their large geographic data coverage and generous quotas. For more information about choosing geolocators, you can follow the discussion in the geopy repository on Github (https://github.com/geopy/geopy/issues/90).

To use a geolocator, import them and assign a variable name (in this case we use the name
```
import geopy
import pandas
from geopy.geocoders import Nominim, GoogleV3

# versions used: geopy 1.10.0, pandas 0.16.2, python 2.7.8

def main():
    io = pandas.read_csv('census-historic-population-borough.csv',
    index_col=None, header=0, sep=",")

    def get_latitude(x):
        return x.latitude

    def get_longitude(x):
        return x.longitude

    geolocator = Nominim()
    # geolocator = GoogleV3()
    # uncomment the geolocator you want to use
```

Finally, using pandas you want to create a column in your spreadsheet called 'latitude'. The script will read the existing 'Area_Name' data column, run the geopy geolocator (http://geopy.readthedocs.io/en/latest/#module-geopy.geocoders) on the column using pandas’ apply function (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.apply.html), and generate a latitude coordinate in that column. The same transformation will occur in the 'longitude' column. Once this is finished it will output a new CSV file with those two columns:
import geopy
import pandas
from geopy.geocoders import Nominatim, GoogleV3
# versions used: geopy 1.10.0, pandas 0.16.2, python 2.7.8

def main():
    io = pandas.read_csv('census-historic-population-borough.csv',
                          index_col=None, header=0, sep='","
    
    def get_latitude(x):
        return x.latitude
    
    def get_longitude(x):
        return x.longitude
    
    geolocator = Nominatim()
    # geolocator = GoogleV3()
    # uncomment the geolocator you want to use
    
    geolocate_column = io['Area_Name'].apply(geolocator.geocode)
    io['latitude'] = geolocate_column.apply(get_latitude)
    io['longitude'] = geolocate_column.apply(get_longitude)
    io.to_csv('geocoding-output.csv')

If we didn't have the .apply(get_latitude) part of the code, we'd get the full point data. For instance, if we were again geocoding the CN Tower and used just .apply(geolocator.geocode), we would get 43.6426,-79.3871 in our column. Adding the additional .apply(get_latitude) would mean that we'd only get 43.6426 in our column.

To finish off your code, it's good practice to make your python modular, that way you can plug it in and out of other applications (should you want to use this script as part of another program). This is what your final python script should look like:
import geopy
import pandas
from geopy.geocoders import Nominatim, GoogleV3

# versions used: geopy 1.10.0, pandas 0.16.2, python 2.7.8

def main():
    io = pandas.read_csv('census-historic-population-borough.csv',
                         index_col=None, header=0, sep=',"
    
    def get_latitude(x):
        return x.latitude

    def get_longitude(x):
        return x.longitude
    
    geolocator = Nominatim()
    # geolocator = GoogleV3()
    # uncomment the geolocator you want to use
    
    geolocate_column = io['Area_Name'].apply(geolocator.geocode)
    io['latitude'] = geolocate_column.apply(get_latitude)
    io['longitude'] = geolocate_column.apply(get_longitude)
    io.to_csv('geocoding-output.csv')

if __name__ == '__main__':
    main()

Do you have the script saved and ready to go? Good. Run the script from your command line by typing:

    python geocoder.py

It takes a few seconds and may take longer depending on the geolocator you use. Once the script finishes running, you should have coordinates for every Area_Name.

Tip 1: If you want to pass the filenames from the command line rather than changing the input file name in the python script everytime, you can import the python ‘sys’ library to pass through arguments. Your code might look like this:
import geopy, sys
import pandas
from geopy.geocoders import Nominatim, GoogleV3
# versions used: geopy 1.10.0, pandas 0.16.2, python 2.7.8

inputfile = str(sys.argv[1])
namecolumn = str(sys.argv[2])

def main():
    io = pandas.read_csv(inputfile, index_col=None, header=0, sep='","

        def get_latitude(x):
            return x.latitude

        def get_longitude(x):
            return x.longitude

    geolocator = Nominatim()
    # geolocator = GoogleV3()
    # uncomment the geolocator you want to use

    geolocate_column = io[namecolumn].apply(geolocator.geocode)
    io['latitude'] = geolocate_column.apply(get_latitude)
    io['longitude'] = geolocate_column.apply(get_longitude)
    io.to_csv('geocoding-output.csv')

if __name__ == '__main__':
    main()

To run your python script your command would look like this:

    python geocoder.py census-historic-population-borough.csv Area_Name

Tip 2: If you run geocoder.py too many times you might get a timeout error. The error will look like this if you use the GoogleV3 geocoder:

    'The given key has gone over the requests limit in the 24'
geopy.exc.GeocoderQuotaExceeded: The given key has gone over the requests limit in the 24 hour period or has submitted too many requests in too short a period of time.

    The error will look like this if you use the Nominatim geocoder:

    geopy.exc.GeocoderTimedOut: Service timed out

    To address the timeout error, you could add the parameter timeout, which specifies the time, in seconds, to wait for the geocoding service to respond before raising a geopy.exc.GeocoderTimedOut exception. So your geolocator declaration will look like this:

    geolocator = Nominatim(timeout=5)
Now that you have a spreadsheet full of coordinate data, we can convert the CSV spreadsheet into a format that web maps like, like GeoJSON. GeoJSON is a web mapping standard of JSON data. There are a couple of ways to make GeoJSON:

The easiest, recommended way is to use a UI tool developed by Mapbox called geojson.io (http://geojson.io). All you have to do is click and drag your csv file into the data window (the right side of the screen, next to the map), and it will automatically format your data into GeoJSON for you. You can select the ‘GeoJSON’ option under ‘Save.’ Save your GeoJSON file as census.geojson.

You can also do it from the command line, using the `csv2geojson` library (https://github.com/mapbox/csv2geojson) that powers geojson.io.

Test this data out by importing it again into geojson.io. You should see points generated in the preview window. That’s your data!

You finally have GeoJSON… but you need to do some cleaning!

If you’ve tested your GeoJSON data, you might notice that not every point is geolocated correctly. We know that every Area_Name is a borough of London, but points appear all over United Kingdom, and some aren’t located even in the country.

To make the results more accurate, save another copy of the census-historic-population-borough.csv file, calling it census_country.csv and include an additional column called ‘Country’
and put ‘United Kingdom’ in every row of your data. For every greater area you add, City and put ‘London’ in every row of your data to provide additional context for your data.

![Country column](//images/mapping-with-python-leaflet/webmap-02-countrycolumn.png)

A new Country column

Make a copy of your geocoder.py python script, calling it geocoder-helpercolumn.py. Remove the following lines:

```python
geolocate_column = io[namecolumn].apply(geolocator.geocode)
io['latitude'] = geolocate_column.apply(get_latitude)
io['longitude'] = geolocate_column.apply(get_longitude)
```

and replacing it with the following that combines the Area_Name and Country or City column to geocode your data:

```python
io['helper'] = io['Area_Name'].map(str) + ' ' + io['Country'].map(str)
geolocate_column = io['helper'].apply(geolocator.geocode)
io['latitude'] = geolocate_column.apply(get_latitude)
io['longitude'] = geolocate_column.apply(get_longitude)
```

Note that we added the .map(str) function. This is a pandas function that is allowing you to concatenate two DataFrame columns into a new, single column (helper) using the syntax format:

```python
df['newcol'] = df['col1'].map(str) + df['col2'].map(str)
```

You will also want to comment out and commenting out `namecolumn=str(sys.argv[2])` since we are now directly specifying the column that we are using in the script.

Your final script should look like this:
import geopy, sys
import pandas
from geopy.geocoders import Nominatim, GoogleV3

inputfile = str(sys.argv[1])
# namecolumn = str(sys.argv[2])

def main():
    io = pandas.read_csv(inputfile, index_col=None, header=0, sep=",",)

    def get_latitude(x):
        return x.latitude
    def get_longitude(x):
        return x.longitude

    # geolocator = Nominatim(timeout=5)
geolocator = GoogleV3(timeout=5)
    # uncomment the geolocator you want to use
    # change the timeout value if you get a timeout error, for instance,
geolocator = Nominatim(timeout=60)
    io['helper'] = io['Area_Name'].map(str) + " " + io['Country'].map(str)
geolocate_column = io['helper'].apply(get_latitude)
io['latitude'] = geolocate_column.apply(get_latitude)
io['longitude'] = geolocate_column.apply(get_longitude)
io.to_csv('geocoding-output-helper.csv')

if __name__ == '__main__':
    main()
other port), use.

    python -m SimpleHTTPServer 8080 or python3 -m http.server 8080 (for Python3)

In your browser go to http://localhost:8080 and you should see the files you've been working with so far.

Now in your text editor open a new document and save it as an html file (mymap.html). If you want to do a quick test, copy and paste the text below, refresh your http://localhost:8080 and open the html file in your browser.
Do you see a map now? Good! If not, you can troubleshoot by inspecting the browser, or by going back and retracing your steps.
Web Mapping with Python and Leaflet | Programming Historian...  
https://programminghistorian.org/en/lessons/map...

You made a web map! Web maps use map tiles, which are pixel based images (rasters) of maps that contain geographical data. This means that each pixel of a map tile has been georeferenced, or assigned a coordinate based on the location that they represent. When you zoom in and out of a web map, you are getting a whole new set of tiles to display at each zoom level. GeoJSON (which you are now familiar with) is a widely used data standard for web mapping. In our example, we are using an open-source Javascript library called Leaflet (http://leafletjs.com/reference.html) to help us build our web map. The benefits of using an open-source library such as Leaflet is the flexibility you get and with developing and customizing your map, without worry of the technology being deprecated or no longer supported that is beyond your control. With frameworks like Leaflet or Google Maps Javascript API, you're not building a map completely from scratch, rather, you're using pre-written functions and controls that helps you customize your own map in code.

Let's go through what each part of the code is doing. But first, it's best practice to maintain your html, css, js in different files so that the web map's content, presentation and behaviour layers are kept separate (though it's not always possible). This adds a bit more structure to your code, making it easier for you and others to understand. It will be easier to focus on certain parts of the code when you're going back and making changes. So here is our code split into three files:

mymap.html

```html
<!DOCTYPE html>
<head>
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.css" />
<script src="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.js"></script>
<script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
<link rel="stylesheet" href="style.css" />
</head>

<body>
  <div id="my-map"></div>
</body>
<script src='leafletmap.js'></script>
</html>
```

style.css

```css
#my-map {
  width: 960px;
  height: 500px;
}
```

leafletmap.js

---

What did I just make? 😎

---

16 of 30 5/1/19, 2:42 PM
window.onload = function() {
  var basemap = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
    });
   
$.getJSON("census.geojson", function(data) {
  var geojson = L.geoJson(data, {
        onEachFeature: function (feature, layer) {
            layer.bindPopup(feature.properties.Area_Name);
        }
    });

  var map = L.map('my-map').
            .fitBounds(geojson.getBounds());

  basemap.addTo(map);
  geojson.addTo(map);
});

Seems a bit easier to understand now, doesn't it? Now let's look at what the html file is doing.

mymap.html walkthrough

mymap.html

<!DOCTYPE html>
<head>
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.css" />
<script src="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.js"></script>
<script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
<link rel="stylesheet" href="style.css" />
</head>

The above code is the first section, or header of your html document. We are linking to the external javascript library and css stylesheets provided by leaflet. We're also linking to our own stylesheets, style.css.

<body>
  <div id="my-map"></div>
</body>

<script src='leafletmap.js'></script>

</html>
Next, we're declaring the body and where you want the map to go on your page. We're also linking to our own javascript file, leafletmap.js.

**style.css walkthrough**

```
#my-map {
    width: 960px;
    height: 500px;
}
```

There's a bit of CSS styling here to specify the size of your map. Some optional styling will happen in your javascript file if you're using the Leaflet library.

**leafletmap.js walkthrough**

```
window.onload = function () {
    var basemap = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
    });

    $.getJSON("census.geojson", function(data) {
        var geojson = L.geoJson(data, {
            onEachFeature: function (feature, layer) {
                layer.bindPopup(feature.properties.Area_Name);
            }
        });

        var map = L.map('my-map').fitBounds(geojson.getBounds());
        basemap.addTo(map);
        geojson.addTo(map);
    });
```

The javascript file is what provides the behaviour, or functionality of our web map. It's what makes our web map come alive! In the code above, we're telling the javascript to load when the browser loads. We're creating our first map layer, which is your basemap. The basemap is the tiles provided by OpenStreetMap that provides places and streetnames found on maps.

Next, we're loading our data as another map layer, census.geojson. This data will have additional properties: each point of data is represented by an icon. It will look and behave like a popup so that when you click on the icon it will load information from your data file (in this case, the Area_Name).
Now we're creating the view for our map. The boundary for our map will be based on the range of our data points in `census.geojson`. You can also manually set your viewport by using the `setView` property (http://leafletjs.com/reference.html#map-set-methods). For example, if you're using `.setView([0.0, -10.0], 2)`, the viewport coordinates `[0.0, -10.0], 2` means that you're setting the centre of the map to be 0.0, -10.0 and at a zoom level of 2.

Finally, the map layers you created will be added to your map. Put it all together and hurrah, you've got your web map! Now let's play around with it. The following five exercises give you tasks to do to learn some of the other elements, with answers provided.

**Exercise 1: Default Viewports**

Let's change the map to use a viewport to 51.505 latitude, -0.09 longitude with a zoom level 9. To do this, we just need to edit one file: `leafletmap.js`.
What we've done there is changed the `.fitBounds` to `.setView`, with the various options mentioned above. Try reloading the file, and you'll see it loads to the correct place.
Exercise 2: Marker Properties

Now let's add the 2001, 1981 and 1991 population property to each marker popup. You can use HTML to style your popup window. To do so, we again need to edit the javascript.

```javascript
window.onload = function () {
    var basemap = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
    });
    $.getJSON("census.geojson", function(data) {
        var geojson = L.geoJson(data, {
            onEachFeature: function (feature, layer) {
                //EDIT HERE
            }
        });
    });

    var map = L.map('my-map')
    .fitBounds(geojson.getBounds());
    .setView([51.505, -0.09], 9);
    basemap.addTo(map);
    geojson.addTo(map);
};
```

What we've done here is edit the onEachFeature function (http://leafletjs.com/reference-1.2.0.html#geojson-oneachfeature), which gets called for each feature (in this case, each marker popup) to add additional information about each marker contained in our census.geojson data. To add attribute information from our census.geojson file, we use the convention feature.properties.ATTRIBUTE_NAME to access the population data. In this case, we are adding feature.properties.Pop_2001, feature.properties.Pop_1981, and feature.properties.Pop_1801, and adding a bit of styling with html for readability.
Exercise 02

Exercise 3: Change Data Source

Change the data source to a different dataset, as an example you can use the `stations.geojson` file.

To do this, we need to edit the javascript file.
Since we're loading a new dataset, we need a new view for our map. First, we've replaced census.geojson with stations.geojson to our $.getJSON request. Next, we add the attribute information found in our stations.geojson file, including the name (feature.properties.name) and line (features.properties.line). Finally, we are using the .fitBounds function so that the viewport automatically centers on our new set of data points.
Exercise 03

Exercise 4: Custom Basemap

Change your data source back to `census.geojson`. Change your basemap layer to a mapbox tileset. You need to get a Mapbox account, create a map or style and get your Mapbox API access token.

Mapbox API

First, you will need to add a reference to the mapbox javascript and css libraries:
Then we edit the javascript file:
In the javascript file, we’ve added our mapbox token in order to access the mapbox API that allows us to access the mapbox basemap that we want to use. Your final result (with your own basemap of choice) should look something like this:
Exercise 04

Exercise 5: Custom Marker Icon

Add a custom leaf icon, as an example you can use `leaf.png` (/assets/mapping-with-python-leaflet/exercises/exercise05/leaf.png). Or use your own!

In this exercise, we only need to edit the javascript file:
Marker icons are defined in leaflet using the `L.icon` object. We specify the image are using to replace our marker by using the property `iconUrl`. Make sure that you specify the proper path to your image. We specified a few additional properties, such as `iconSize` (dimensions of the icon in pixels), `iconAnchor` (coordinates of the icon which will correspond to marker's location), `popupAnchor` (coordinates from which the popup should open relative to the `iconAnchor`). Check Web Mapping with Python and Leaflet | Program... https://programminghistorian.org/en/lessons/map...
out the Icon Leaflet documentation (http://leafletjs.com/reference-1.2.0.html#icon) more information about L.Icon properties.

The final map should look something like this:

![Map](http://images/mapping-with-python-leaflet/webmap-09-exercise05.jpg)

Exercise 05

**Next Steps**

Congratulations! You now have some hands-on experience geocoding using common Python data analysis libraries and working with one of the most popular Javascript web mapping libraries out there.

If you want to explore other web mapping features with Leaflet, there are a number of additional plugins (http://leafletjs.com/plugins.html) to try out. Of particular interest may be ability to create time based visualizations (https://github.com/skeate/Leaflet.timeline) and do heat-mapping (https://github.com/pa7/heatmap.js).

Also, check out the Programming Historian Lesson Using Javascript to Create Maps of Correspondence (https://lessons/using-javascript-to-create-maps) that goes in depth on how to analyze correspondence using geospatial software, and using some of the same tools as this lesson.

**About the author**

Kim Pham is the Digital Projects and Technologies Librarian at the University of Toronto Scarborough Library. Her heart belongs to open-source software & shared, community-driven development practices in libraries.

**Suggested Citation**

Kim Pham, "Web Mapping with Python and Leaflet," The Programming Historian 6 (2017),