From: Piruz Alemi
Subject: Report on Leaflet – JavaScript
Date: Feb 27th, 2020

Run Index.html

To get results of the following report:

Alemi- Leaflet.js

Objectives

- Understand the benefits that visualizing data with maps can provide.
- Learn the basics of creating maps and plotting data with the Leaflet.js library.
- Gain an understanding of the GeoJSON format.
- Understand the concept of layers and layer controls and how we can use them to add interactivity to our maps.
- Gain a firm grasp of mapping with GeoJSON.
- Learn about and practice using Leaflet plugins and third-party libraries.
- Learn how different maps can effectively visualize different datasets.
- Gain a Leaflet mastery by completing an in-class project.
- Understand how different types of maps are better for visualizing different datasets.

Helpful Links I utilized:

- Leaflet.js Tutorial

  Leaflet supports all of the GeoJSON types above, but Features and FeatureCollections work best as they allow you to describe features with a set of properties. We can even use these properties to style our Leaflet vectors. Here's an example of a simple GeoJSON feature:

```
var geojsonFeature = {
    "type": "Feature",
    "properties": {
        "name": "Coors Field",
        "amenity": "Baseball Stadium",
        "popupContent": "This is where the Rockies play!"
    },
    "geometry": {
        "type": "Point",
        "coordinates": [-104.99404, 39.75621]
    }
};
```

  https://leafletjs.com/examples/geojson/
- https://leaflet-extras.github.io/leaflet-providers/preview/

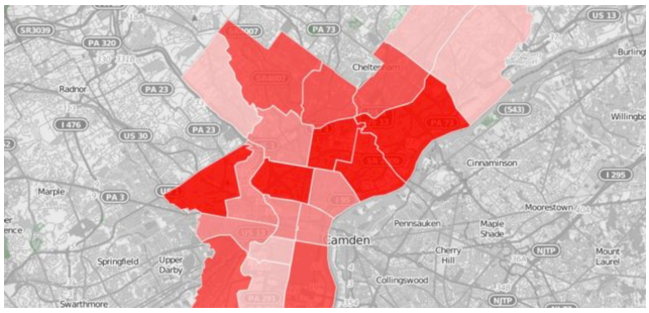
- https://github.com/timwis/Leaflet-choropleth

While Leaflet provides a choropleth tutorial, that approach requires us to specify exact breakpoints and colors. This plugin uses chroma.js . We just tell it which property in the GeoJSON to use and some idea of the color scale we want.

Usage

```
L.choropleth(geojsonData, {
        valueProperty: 'incidents', // which property in the features to use
        scale: ['white', 'red'], // chroma.js scale - include as many as you like
        steps: 5, // number of breaks or steps in range
        mode: 'q', // q for quantile, e for equidistant, k for k-means
        style: {
                color: '#fff', // border color
                weight: 2,
                fillOpacity: 0.8
        },
        onEachFeature: function(feature, layer) {
                layer.bindPopup(feature.properties.value)
        }
}).addTo(map)
```

Choropleth plugin for Leaflet (color scale based on value) - Demo



Some additional helpful links:

- Leaflet Usage Example
- Citi Bike Station Information API EndPoint
- Leaflet Popup Doc
- Citi Bike Station Status API EndPoint
- Leaflet Layer Groups Doc
- Leaflet Extra Markers
- Leaflet Legend Doc
- https://github.com/Leaflet/Leaflet.heat
- https://github.com/Leaflet/Leaflet.heat
- https://leaflet-extras.github.io/leaflet-providers/preview/
- Interactive examples of a bunch of different basemaps you can use with Leaflet

Welcome to the United States Geological Survey, or USGS for short! The USGS is responsible for providing scientific data about natural hazards, the health of our ecosystems and environment; and the impacts of climate and land-use change. Their scientists develop new methods and tools to supply timely, relevant, and useful information about the Earth and its processes. As a new hire, I was helping them out with an exciting new project!

The USGS is interested in building a new set of tools that will allow them visualize their earthquake data. They collect a massive amount of data from all over the world each day, but they lack a meaningful way of displaying it. Their hope is that being able to visualize their data will allow them to better educate the public and other government organizations (and hopefully secure more funding) on issues facing our planet.

Interesting Solutions:

- Had to provide my own API key to bypass following error:

```
Failed to load resource: the server responded with a status of 401 (Unauthorized)    2928.png:1
```

1. Set my path Options: https://leafletjs.com/reference-1.6.0.html#path
2. Tested the layers: https://leafletjs.com/reference-1.6.0.html#layer
3. Locations: https://leafletjs.com/reference-1.0.3.html#toc
4. Used the 'Vector Layers' section of the Leaflet documentation for reference.
5. Refered to the Leaflet docs for Path Options when I was stuck creating vector layers.
6. Refered to the Leaflet Layers Control Docs
7. I was successful. I was able to toggle between Street Map and Dark Map base layers, as well as turn earthquakes layers on and off.

See Leaflet Documentation on GeoJSON:

8. http://leafletjs.com/reference.html#geojson
9. http://leafletjs.com/examples/geojson/
10. https://leafletjs.com/reference-1.0.3.html#geojson
11.

Before I Began

1. I created a new repository for this project called `leaflet-challenge`. Did not add this homework to an existing repository.
2. Cloned the new repository to my computer.
3. Inside my local git repository, created a directory for the Leaflet challenge. Used the folder names to correspond to the challenges.
4. I utilized both **html** and **Javascript** that i added to all the necessary files. These were the main files I ran for analysis.
5. Pushed the above changes to GitHub.

My Task

Level 1: Basic Visualization

My first task was to visualize an earthquake data set.

1. **To Get my data set**

   The USGS provides earthquake data in a number of different formats, updated every 5 minutes. I Visited the USGS GeoJSON Feed page and picked a data set to visualize. When I clicked on a data set, for example 'All Earthquakes from the Past 30 Days', I was given a JSON representation of that data. After prettifying the json data, I used the URL of this JSON to pull in the data for our visualization.

2. **Imported & Visualized the Data**

   Created a map using Leaflet that plots all of the earthquakes from my data set based on their longitude and latitude.

   - My data markers reflected the magnitude of the earthquake in their **size and color**. Earthquakes with higher magnitudes appear larger and darker in color.
   - Included **popups** that provided additional information about the earthquake when a marker is clicked.
   - Created a **legend** that provided a context for my map data.

In Level 2: More Data (Optional - Advance)

I successfully completed the Advance part. The USGS wanted to plot a second data set on my map to illustrate the relationship between tectonic plates and seismic activity. I needed to pull in a second data set and visualize it along side my original set of data. Data on tectonic plates was found at https://github.com/fraxen/tectonicplates.

In this step I did plot:

- Plotted a second data set on our map.
- Added a number of base maps to choose from as well as separate out our two different data sets into overlays that can be turned on and off independently.
- Added layer controls to our map.

Assessment

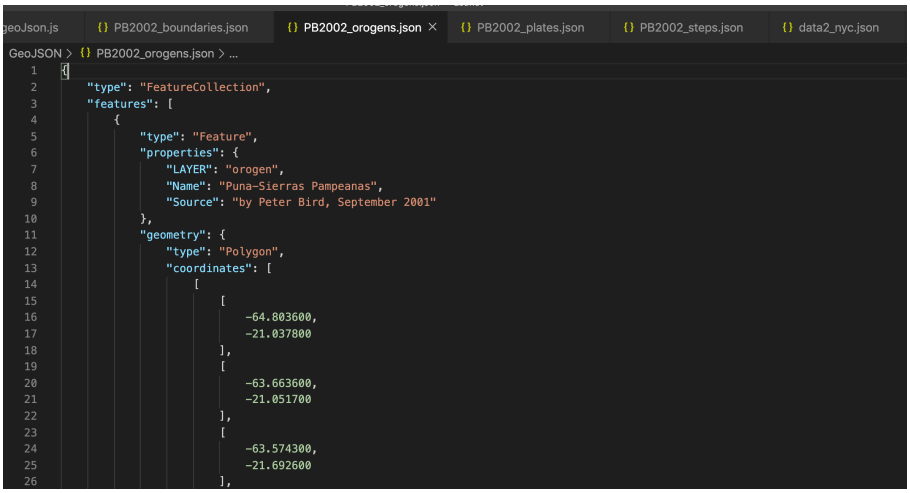
My final product was assessed on the following metrics:

- Completion of assigned tasks
- Visual appearance
- Professionalism

Appendix 1

DATA USED:

I used a geoJson data as in PB2002_orogens.json below:
4 different data sets exist on the following github:
For further details see: https://github.com/fraxen/tectonicplates



The second map, consisted of earthquakes in the following format:

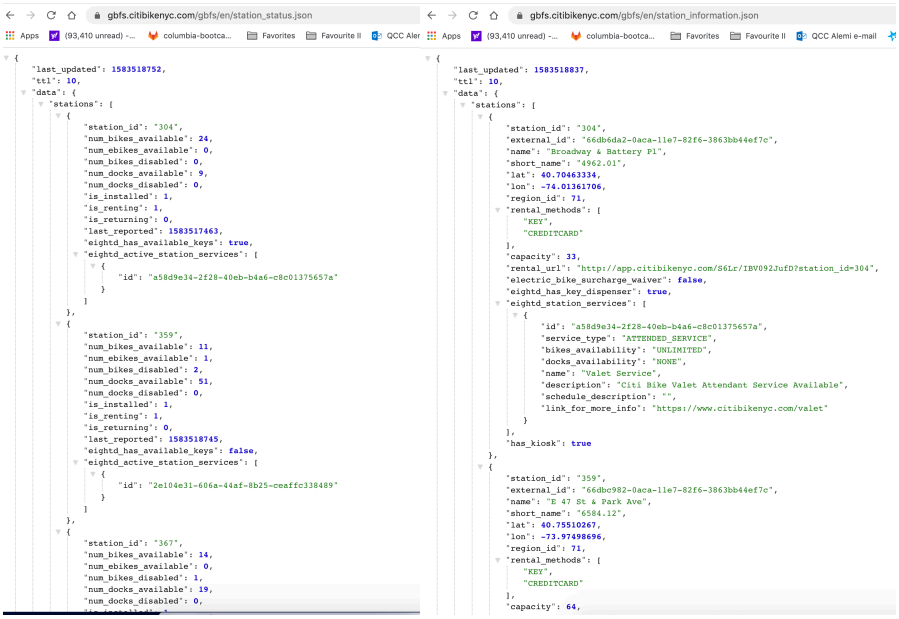

```
{
  "type": "FeatureCollection",
  "metadata": {
    "generated": 1583517966000,
    "url": "https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime=2014-01-01&endtime=2014-01-02&maxlongitude=-69.52148437&minlongitude=-123.83789062&maxlatitude=48.74894534&minlatitude=25.16517337",
    "title": "USGS Earthquakes",
    "status": 200,
    "api": "1.8.1",
    "count": 213
  },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "mag": 1.29,
        "place": "10km SSW of Idyllwild, CA",
        "time": 1388620296020,
        "updated": 1457728844428,
        "tz": -480,
        "url": "https://earthquake.usgs.gov/earthquakes/eventpage/ci11408890",
        "detail": "https://earthquake.usgs.gov/fdsnws/event/1/query?eventid=ci11408890&format=geojson",
        "felt": null,
        "cdi": null,
        "mmi": null,
        "alert": null,
        "status": "reviewed",
        "tsunami": 0,
        "sig": 26,
        "net": "ci",
        "code": "11408890",
        "ids": ",ci11408890,",
        "sources": ",ci,",
        "types": ",cap,focal-mechanism,general-link,geoserve,nearby-cities,origin,phase-data,scitech-link,",
        "nst": 39,
        "dmin": 0.06729,
        "rms": 0.09,
        "gap": 51,
        "magType": "ml",
        "type": "earthquake",
        "title": "M 1.3 - 10km SSW of Idyllwild, CA"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          -116.7776667,
          33.6633333,
          11.008
```


From the bike data where we used the original program and adapted it to earthquake data, the data consisted of two parts:

1. Bike. Info and Bike Stations_Status
2. Information – which carries the coordinates of Lat & Longitude

On this data set a loop was set on Data.Stations to get each stations's properties. First the data was read as in:

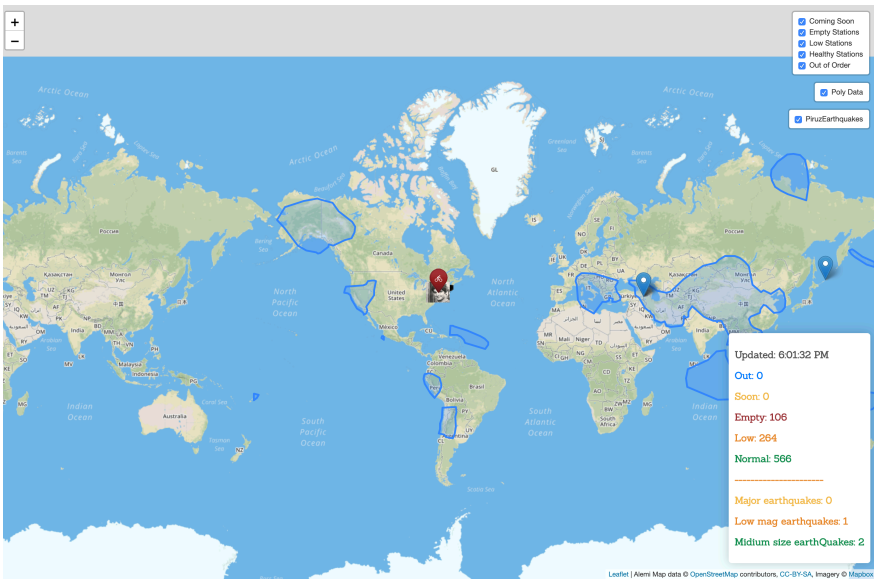
```
157
158   // Perform an API call to the Citi Bike Station Information endpoint
159   d3.json("https://gbfs.citibikenyc.com/gbfs/en/station_information.json", function(infoRes) {
160
161     // When the first API call is complete, perform another call to the Citi Bike Station Status endpoint
162     d3.json("https://gbfs.citibikenyc.com/gbfs/en/station_status.json", function(statusRes) {
163       var updatedAt = infoRes.last_updated;
164       var stationStatus = statusRes.data.stations;
165       var stationInfo = infoRes.data.stations;
166
167       console.log("updatedAt:", updatedAt)
```



Next there was a loop as in:

```
93
94      // ------------------------------------------------------------------------
95      // Initialize a stationStatusCode, which will be used as a key to access the
96      // appropriate layers, icons, and station count for layer group
97      // ------------------------------------------------------------------------
98      var stationStatusCode;
99
00      // Loop through the stations (they're the same size and have partially matching data)
01      for (var i = 0; i < stationInfo.length; i++) {
02
03        // Create a new station object with properties of both station objects
04        var station = Object.assign({}, stationInfo[i], stationStatus[i]);
05        // If a station is listed but not installed, it's coming soon
06        if (!station.is_installed) {
07          stationStatusCode = "COMING_SOON";
08        }
09        // If a station has no bikes available, it's empty
10        else if (!station.num_bikes_available) {
11          stationStatusCode = "EMPTY";
12        }
13        // If a station is installed but isn't renting, it's out of order
14        else if (station.is_installed && !station.is_renting) {
15          stationStatusCode = "OUT_OF_ORDER";
16        }
```

The final result was:


Coming Soon
Empty Stations
Low Stations
Healthy Stations
Out of Order
Poly Data
PiruzEarthquakes
Updated: 6:01:32 PM
Out: 0
Soon: 0
Empty: 106
Low: 264
Normal: 566
Major earthquakes: 0
Low mag earthquakes: 1
Midium size earthQuakes: 2
Leaflet | Alemi Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox