Web Map Interactions and Proportional Symbol Mapping

Objectives

- Understand proportional or graduated symbolization in web map format;
- Build your skillset with adding color to thematic layers;
- Adding map interactions;
- Create custom map controls.

In this lecture, you will make a proportional symbol map of US Population Estimates using Leaflet. You will build off your existing skill set with web mapping libraries and programming languages. You will continue working with the GeoJSON data format and create additional interactivity of your choice.

1. Finding and Formatting Data

The first step is to gather spatial data that are suitable for displaying as proportional symbols. Because proportional symbol maps use the visual variable size, you should search for ordinal, or numerical, data. Categorical data will not work. The data should also contain unique timestamps of different data values in different geographic locations. Point or area level data are ideal.

For this lecture, data are supplied in the `uspopchange.csv` file available on ASULearn. This file contains the US Census Bureau Population Estimates for all 50 states in 2019. You will notice that these data contain logical header names, which serve as attribute keys for referencing the information. Since Leaflet intuitively understands geographic coordinate systems, latitude and longitude coordinates for all US state centroids are appended to the table.

id	name	lat	long	estimate
1	Alabama	32.806671	-86.79113	4903185
2	Alaska	61.370716	-152.40442	731545
3	Arizona	33.729759	-111.43122	7278717
4	Arkansas	34.969704	-92.373123	3017804
5	California	36.116203	-119.68156	39512223
6	Colorado	39.059811	-105.3111	5758736
7	Connecticut	41.597782	-72.755371	3565287
8	Delaware	39.318523	-75.507141	973764
9	Florida	27.766279	-81.686783	21477737

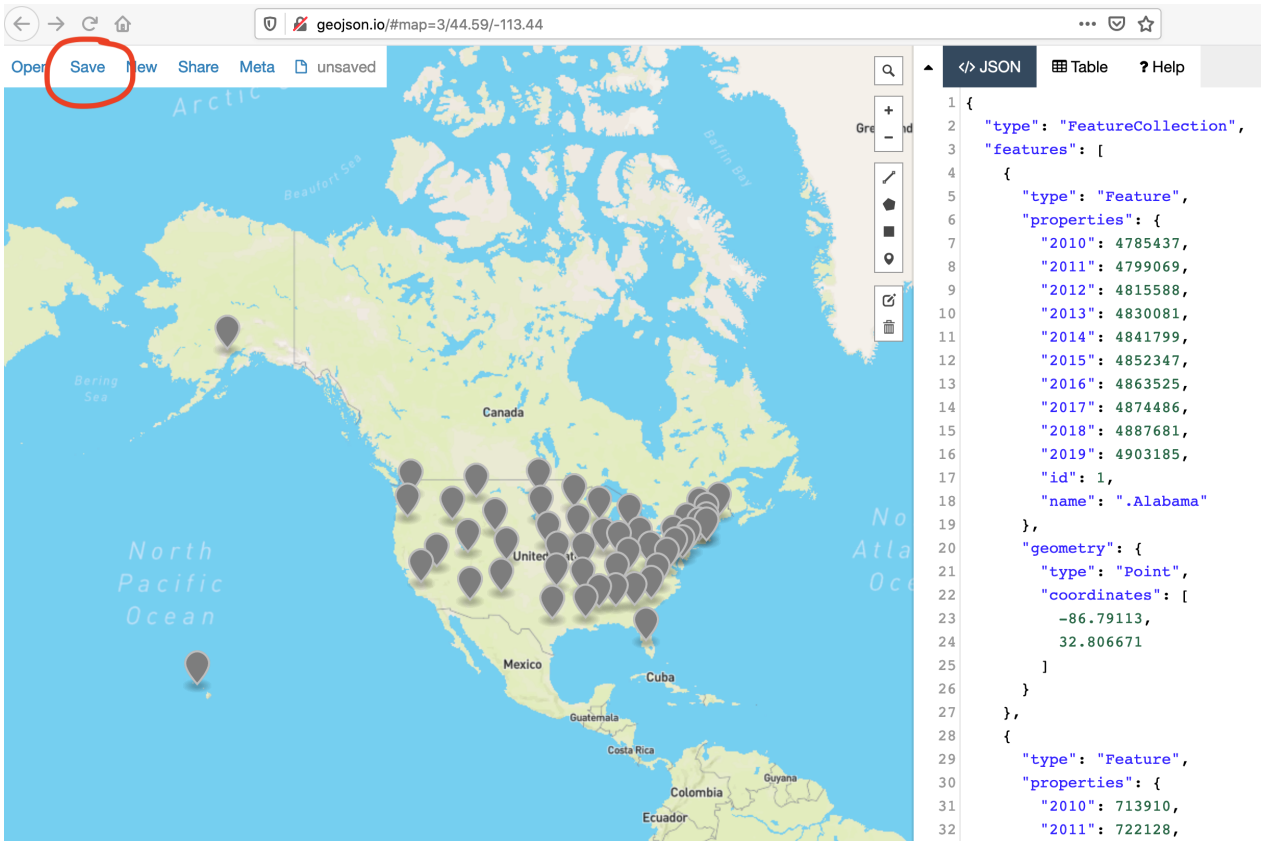
> Take note of the data formatting in the cross section here.

The next step is to convert the data into the GeoJSON format. Remember, JSON stands for JavaScript Object Notation, which is a standard format for data and information to be interpreted by a browser. GeoJSON is its geographic variant.

You can convert the data into GeoJSON format using different options that we've covered in class.

- Using GIS applications such as ArcGIS or QGIS, which allow you to add your .csv file and then convert to a GeoJSON.
- Using free web services, such as GeoJSON.io, which allow you to open your .csv file and then save to GeoJson.

Do the conversion and then save your `data.geojson` file in your assets folder in the directory for exercise.



2. Setup Your Local Directory

Now that you've got a dataset in the proper format, it's time to build your map. That starts with setting up your working directory. Create a folder with the following structure, subfolders, datasets, and text files.

```
[your_repository_name]
    |index.html
    |readme.md
    ├─assets
    |      data.geojson
```

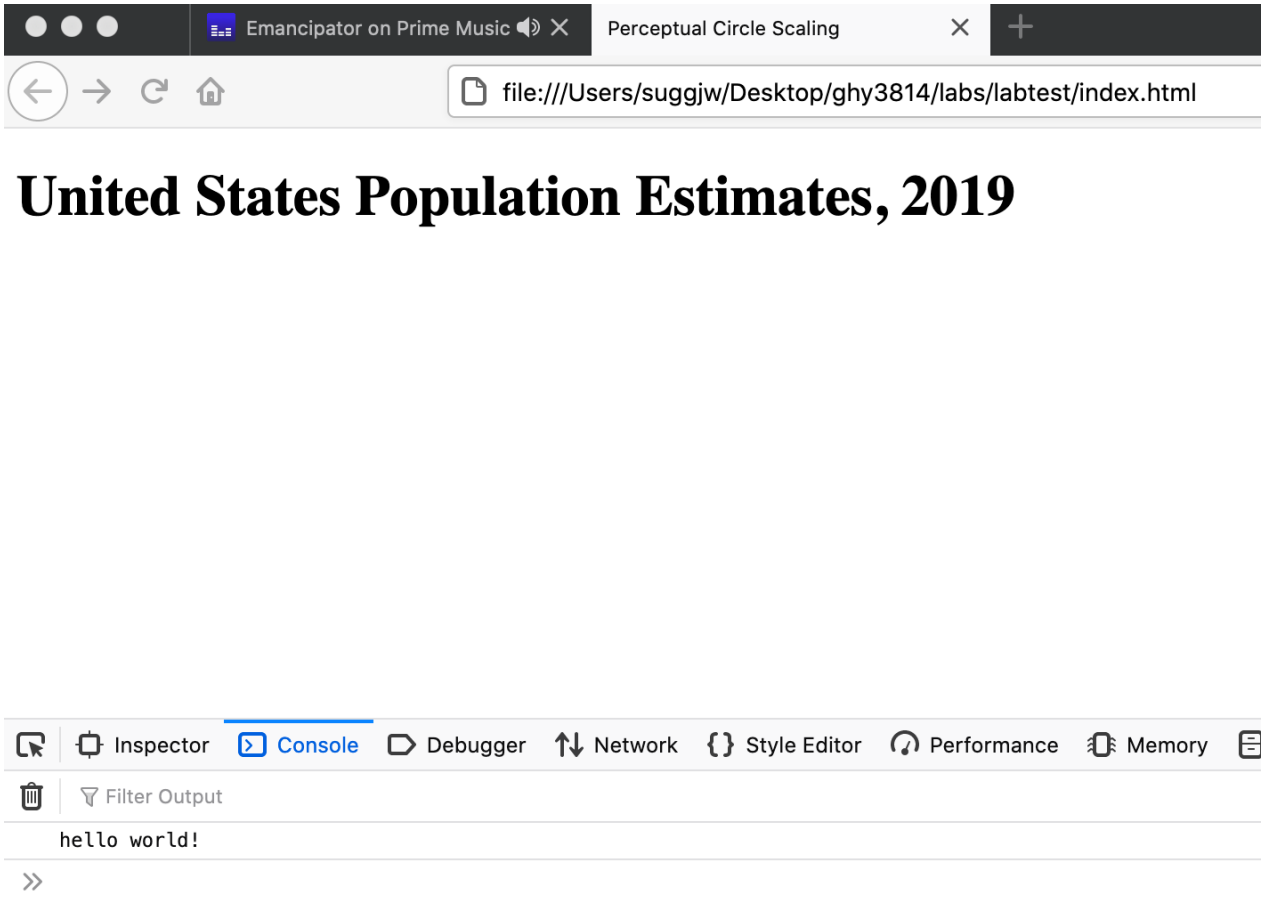
Next, begin working in your `index.html` file and copy and paste the following script to setup a basic web map. This should be more familiar at this point in the semester.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset=utf-8 />
    <title>Perceptual Circle Scaling</title>
    <meta name='viewport' content='initial-scale=1,maximum-scale=1,user-scalable=no' />
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

<style>
</style>

</head>
<body>
    <div id='map'></div>
    <h1>United States Population Estimates, 2019</h1>
```

```
    <script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet-ajax/2.1.0/leaflet.ajax.min.js"></script>


<script>
console.log("hello world!");
</script>

</body>
</html>
```

Notice we've setup a basic HTMl document, called the Leaflet stylesheet, created a header, as well as called jQuery, Leaflet JS library, and AJAX library to help us work with GeoJSON data. We also added a `<div>` element to the `<body>`, giving it the `id` attribute `map` to reference our stylesheets and scripts in their respective places within the directory.

Now, notice `console.log` in the script, which will test whether our setup is working. Once you've saved your document, double click on your *index.html* file to open it in a browser. You should see a blank webpage with our web map title. Open up the web developer tools to view the browser console. You should see our JS command in action.



3. Styling the Page and Adding a Basemap

We're almost ready to load basemap tiles into our webpage from Leaflet. First, we need to add some CSS styling to our page though, so we can make important elements stand out. We will adjust the body sections, main header, and the map object, specifically the fonts, margins, padding, etc. You can experiment with different combinations of these to see the effect.

To do that, add the following code between the style tags in the head of your *index.html* document.

```
<style>
```

```
  body {
    margin:0;
    padding:0;
    font-family:
    sans-serif;
    }

  h1 {
    position: absolute;
    left: 50px;
    top: 10px;
    padding: 8px 2%;
    margin: 0;
    background: rgba(255,121,0,0.8);
    box-shadow: 0 0 15px rgba(0,0,0,0.2);
    border-radius: 3px;
    color: whitesmoke;
    font-size: 1.5em;
    z-index: 800;
    }

  #map {
    position:absolute;
    top:0;
    bottom:0;
    width:960px;
    height: 540px;
   }

</style>
```

Now, ready for the basemap. Here's some review content about the basemap tiles. Leaflet requires a URL reference using the following syntax:

`{s}.[URL listed here, for example]/{z}/{x}/{y}.png`

The {s} indicates possible server instances from which the map can draw tiles. For every tile that gets loaded, the {z} indicates its zoom level, the {x} indicates the horizontal coordinate, and {y} indicates the vertical coordinate. Nearly all public tile sevices use this format.

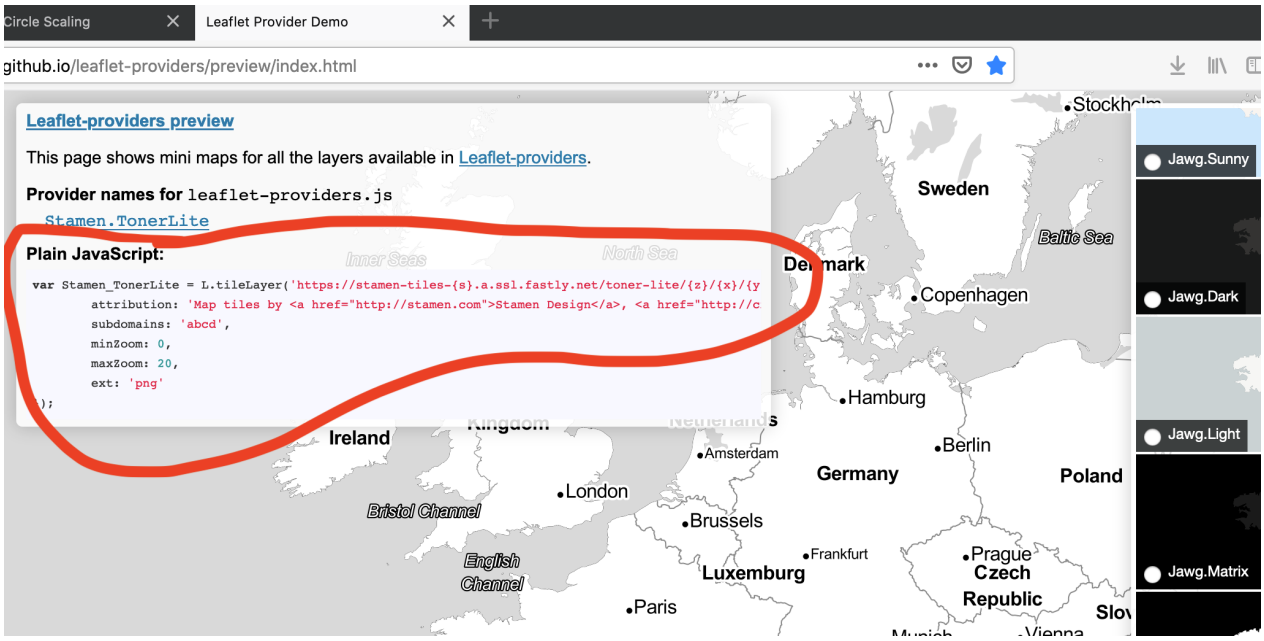
Now move down to the `<body>` section of your *index.html* and change to the following within the `<script>` tags.

```
<script>
console.log("hello world!");

var map = L.map('map', {
    center: [39.5, -95.3],
    zoom: 4
});

var Stamen_TonerLite = L.tileLayer
('https://stamen-tiles-{s}.a.ssl.fastly.net/toner-lite/{z}/{x}/{y}{r}.{ext}', {
    attribution: 'Map tiles by <a href="http://stamen.com">Stamen Design</a>,
    <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash;
    Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
    contributors',
    ext: 'png'
}).addTo(map);
</script>
```

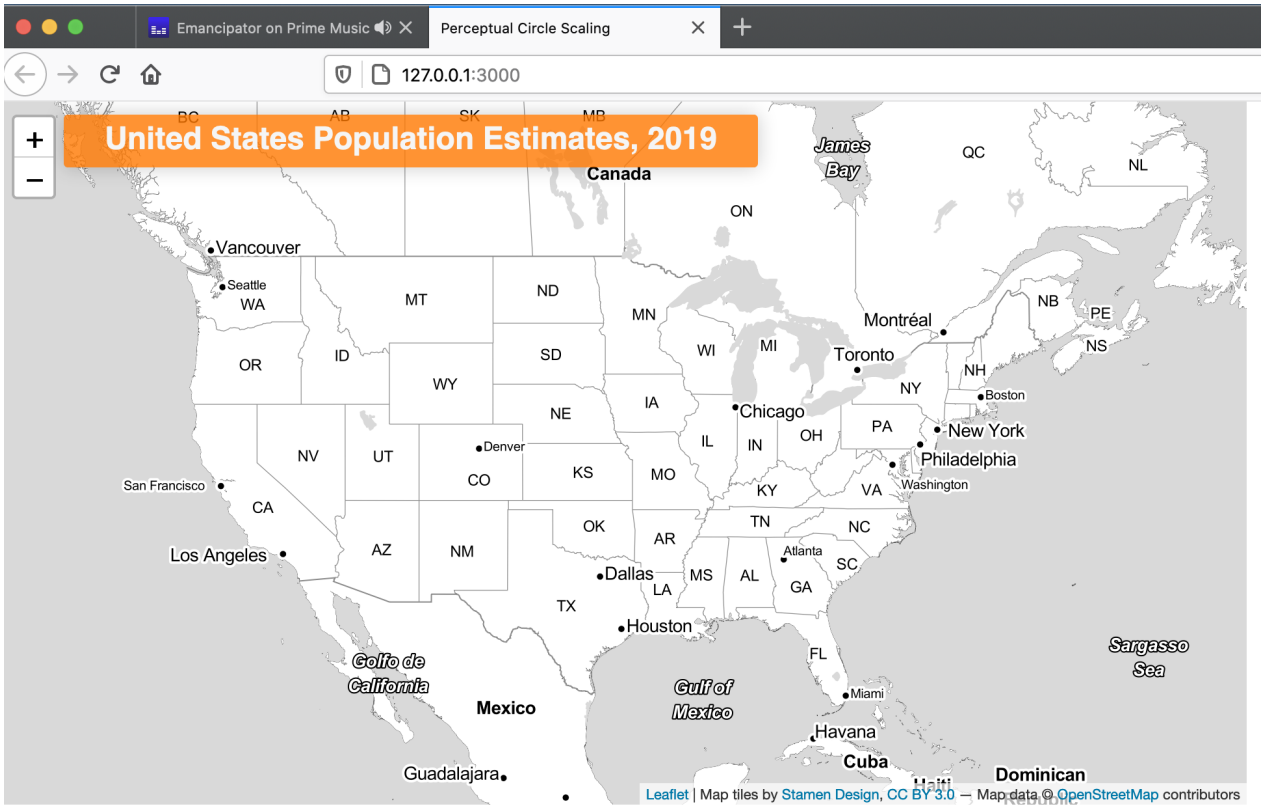
So far we've added the map object using `var map`, centered it to the US centroid location, set the default zoom level, and then added the base map using `var Stamen_TonerLite`.

The base map code comes from Leaflet Provider Demo, where you can scroll through many available basemap tile services. For this exercise, we're using the Stamen Toner Lite basemap because it provides a light colored theme that provides high contrast

with symbology. Notice how we've copied all the available JavaScript and added it to our map document using `L.tileLayer`. Take note here that you will need to copy the script for the basemap directly from the plain JavaScript section listed on the Leaflet Providers Demo or you will be missing some code.



To test it out, launch Atom's Live Server and confirm that your page and basemap are displaying properly.

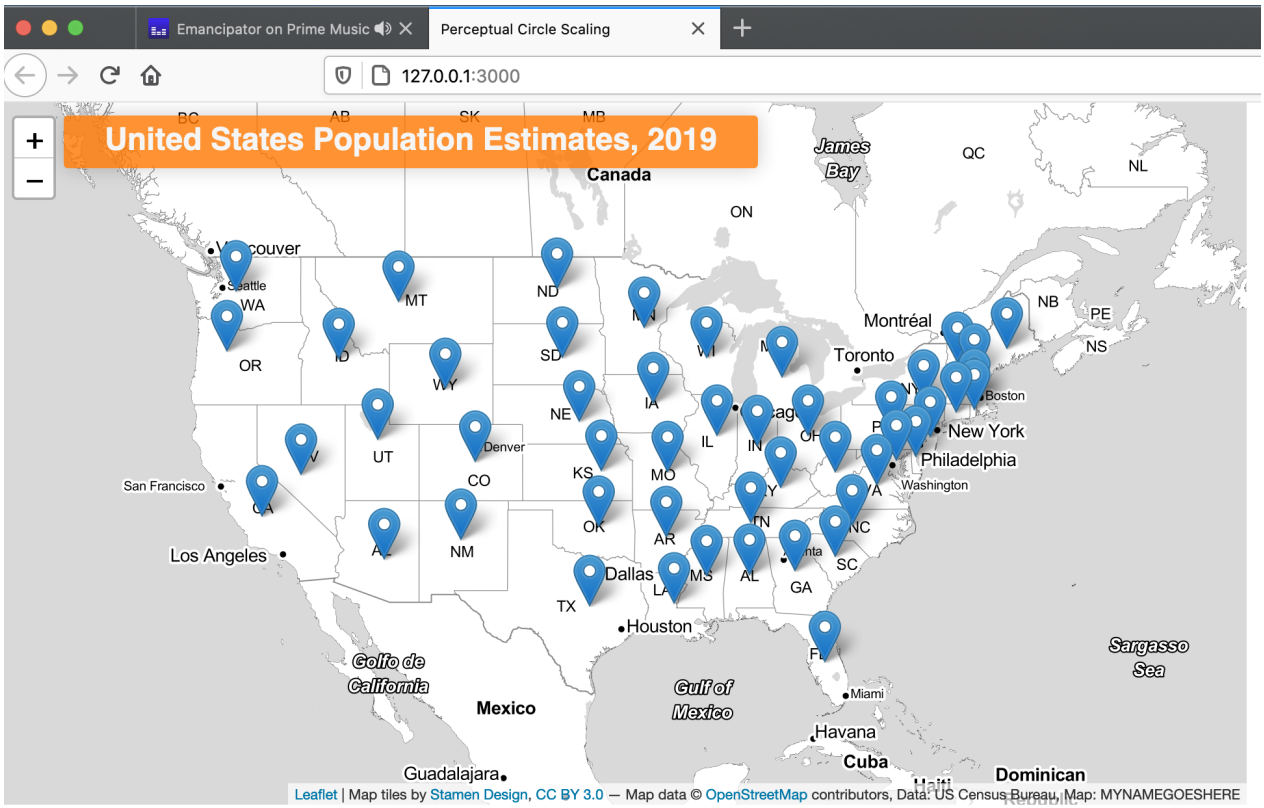


3. Adding Data and Adjusting Symbology

Now it's time to add GeoJSON population data to the map. Continue working within the `<body>` section of your *index.html* and adding new code between the `<script>` tags, just below where the basemap is located.

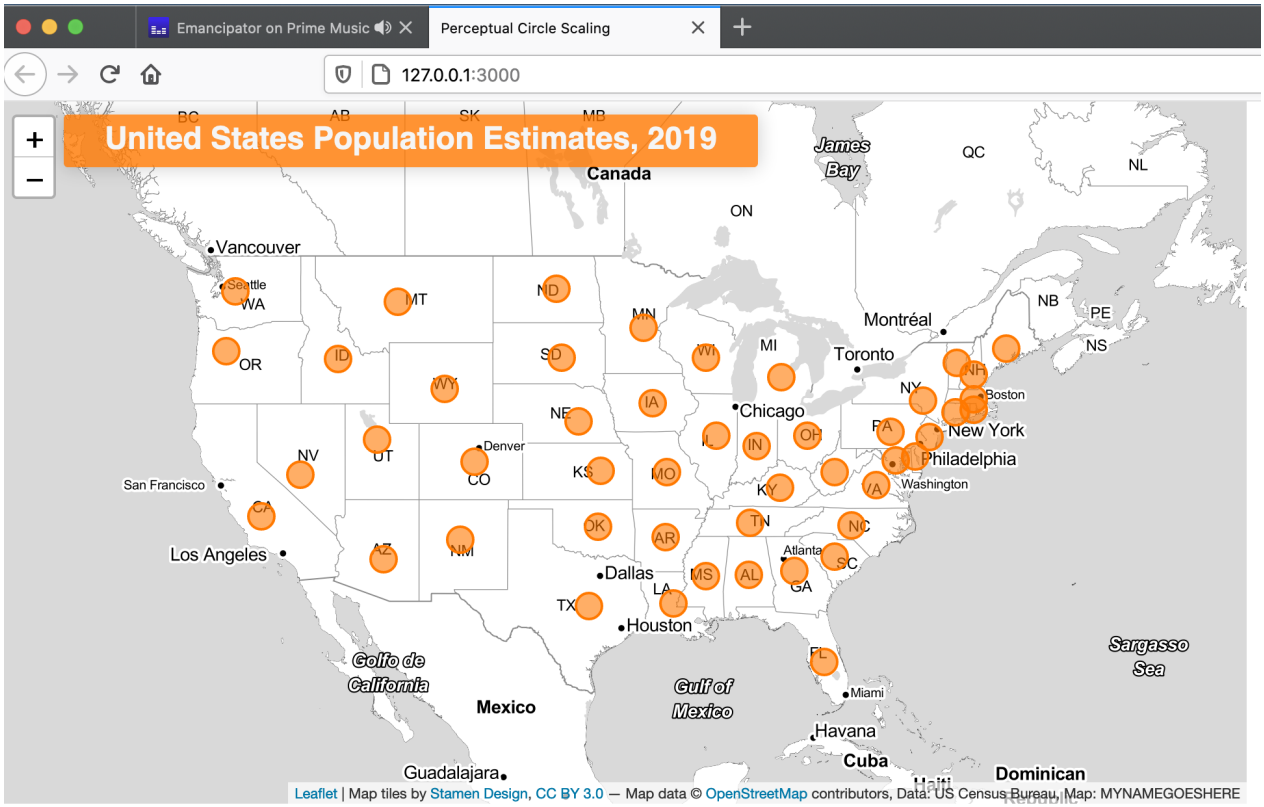
```
states = L.geoJson.ajax("assets/data.geojson", {
attribution: 'Data: US Census Bureau, Map: MYNAMEGOESHERE'});

states.addTo(map);
```

These two lines define the states data which are called using `L.geoJson.ajax`, and then add them to the map. We've also added some additional attribution information to the map. Refresh your map and view the placemarkers.



Adjustments are still needed to create circles for each placemarker. We will use `pointToLayer` to define a feature function on this GeoJSON layer and then adjust some of the style properties while we're at it. Change the code to the following and then experiment with some of the properties.

```
states = L.geoJson.ajax("assets/data.geojson",{
attribution: 'Data: US Census Bureau, Map: MYNAMEGOESHERE',
pointToLayer: function (feature, x){
  return L.circleMarker(x, {
    color: '#ff7900',
    opacity: 1,
    weight: 2,
    fillColor: '#ff7900',
    fillOpacity: .6,
    radius: 10
  }) }});

states.addTo(map);
```



> Nice to see some circles, all radius size 10.

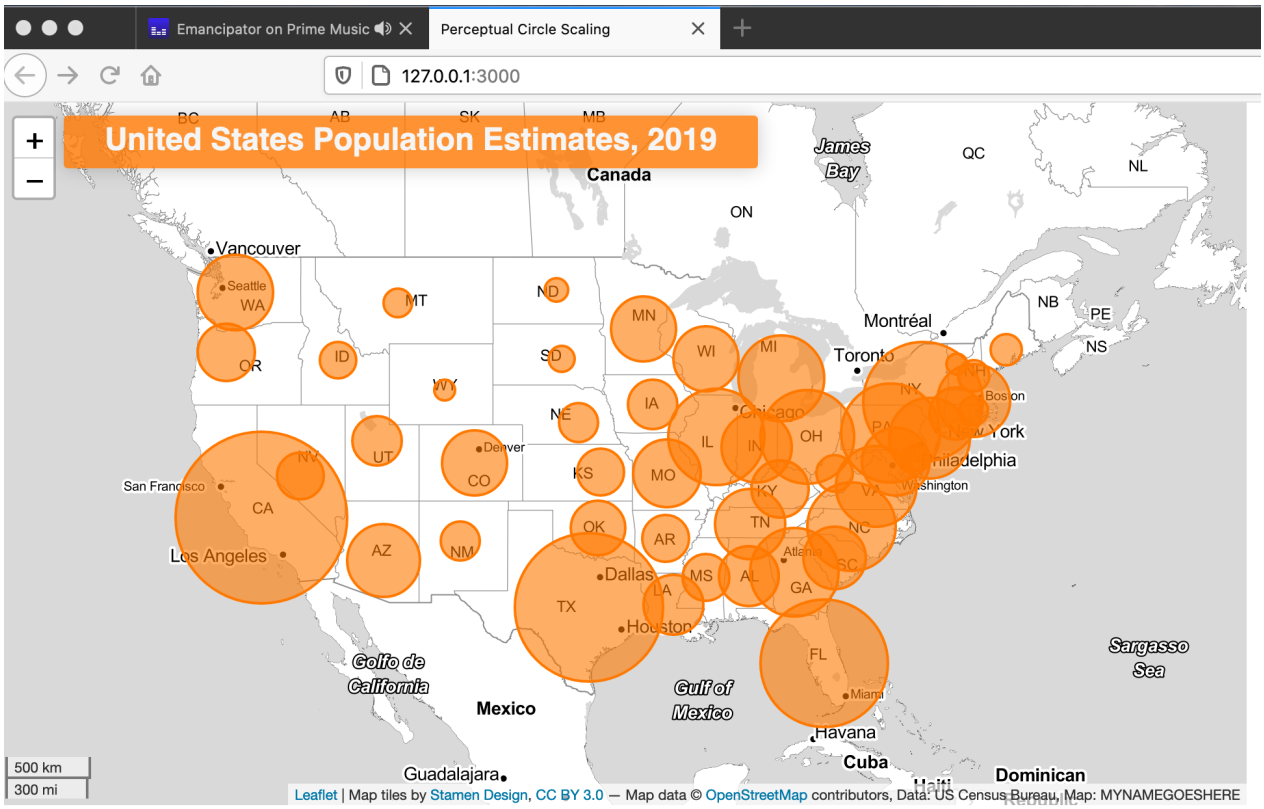
Time to adjust the circle scaling. In a proportional symbol map, the symbols are adjusted based on the visual variable size to indicate differences within the data. The symbols should be directly proportional to the data values. In other words, we will use the data value to calculate the radius of each circle for each of the 50 states. There are many ways to do circle scaling, but we will use a formula based on the perceptual/psychological methods that are discussed by John Krygier.

Change your code based on the snippet below. Notice, the `radius` option has been changed to `calcRadius`, which is a function to help calculate circle sizes. Several new variables are created and then the new function is created. The `minValue` variable must be the minimum data value, which in this case is Wyoming's population estimate. The `minRadius` variable is the smallest desired circle size, and can be adjusted if changes are needed to circle sizing. The `calcRadius` function uses the Math.pow() method to exponentially increase the data value before reducing (i.e. scaling) it in size based on the desired starting radius value. Overall, the scaling method is similar to those used for proportional symbols in Cartography. Finally, `L.control.scale` is used to add a scale bar to the map.

```
states = L.geoJson.ajax("assets/data.geojson",{
attribution: 'Data: US Census Bureau, Map: MYNAMEGOESHERE',
pointToLayer: function (feature, x){
  return L.circleMarker(x, {
    color: '#ff7900',
    opacity: 1,
    weight: 2,
    fillColor: '#ff7900',
    fillOpacity: .6,
    radius: calcRadius(feature.properties.estimate)
  }) }});

  var minValue = 578759;
  var minRadius = 8;

  function calcRadius(val) {
      return 1.00 * Math.pow(val/minValue,.5) * minRadius;
      }
```

```
states.addTo(map);

L.control.scale({position: 'bottomleft'}).addTo(map);
```

Refresh the map and you should see proportionally scaled circles.



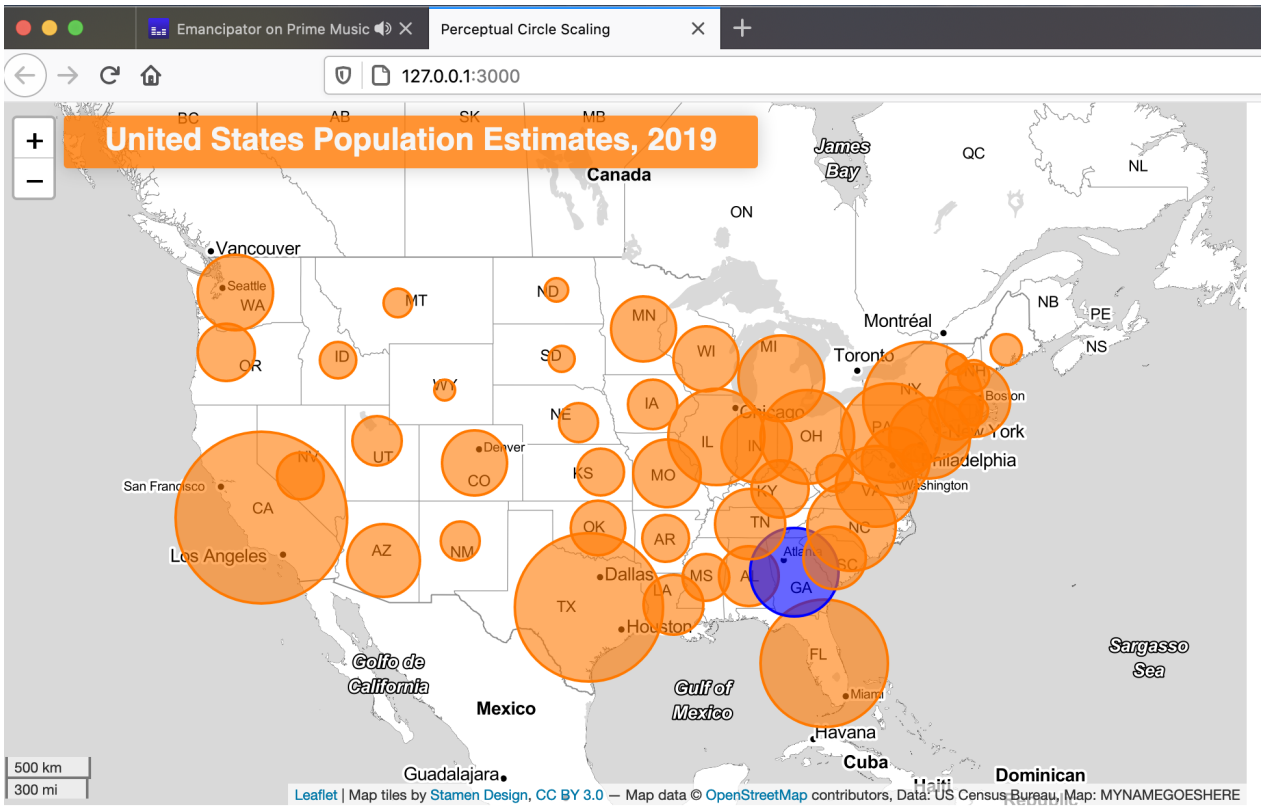
4. Adding Interaction

Finally, let's get started adding some interactivity to this map. Begin by adding some `on.mouseover` and `on.mouseout` event listeners for this feature layer. Specifically, for `on.mouseover` a function (e) is defined to `setStyle` properties on each circle in the layer, including blue stroke and fill color. For `on.mouseout` a function (e) is defined to `setStyle` properties on each circle and return them to their original state. The updated code should look like the following.

```
states = L.geoJson.ajax("assets/data.geojson",{
attribution: 'Data: US Census Bureau, Map: MYNAMEGOESHERE',
pointToLayer: function (feature, x){
  return L.circleMarker(x, {
    color: '#ff7900',
    opacity: 1,
    weight: 2,
    fillColor: '#ff7900',
    fillOpacity: .6,
    radius: calcRadius(feature.properties.estimate)
  }).on({
          mouseover: function(e){
            this.setStyle({color: 'blue',fillColor: 'blue'});
                                },
          mouseout: function(e){
            this.setStyle({color:'#ff7900', fillColor: '#ff7900'});
            } })},});

  var minValue = 578759;
  var minRadius = 8;

  function calcRadius(val) {
      return 1.00 * Math.pow(val/minValue,.5) * minRadius;
```

```
        }

states.addTo(map);
```

Refresh your map and view the result.



5. Final points

As you complete this lecture, please think about what else is left to host it on a GitHub page in a new repository that could be shared with the general public. You should include a short markdown file to describe the map message and its application. What other elements would improve this map?

- A legend with a max and min representative circle size
- A div container for a separate section describing the map message
- A popup or tooltip that shows the state population estimate on mouseover or click event