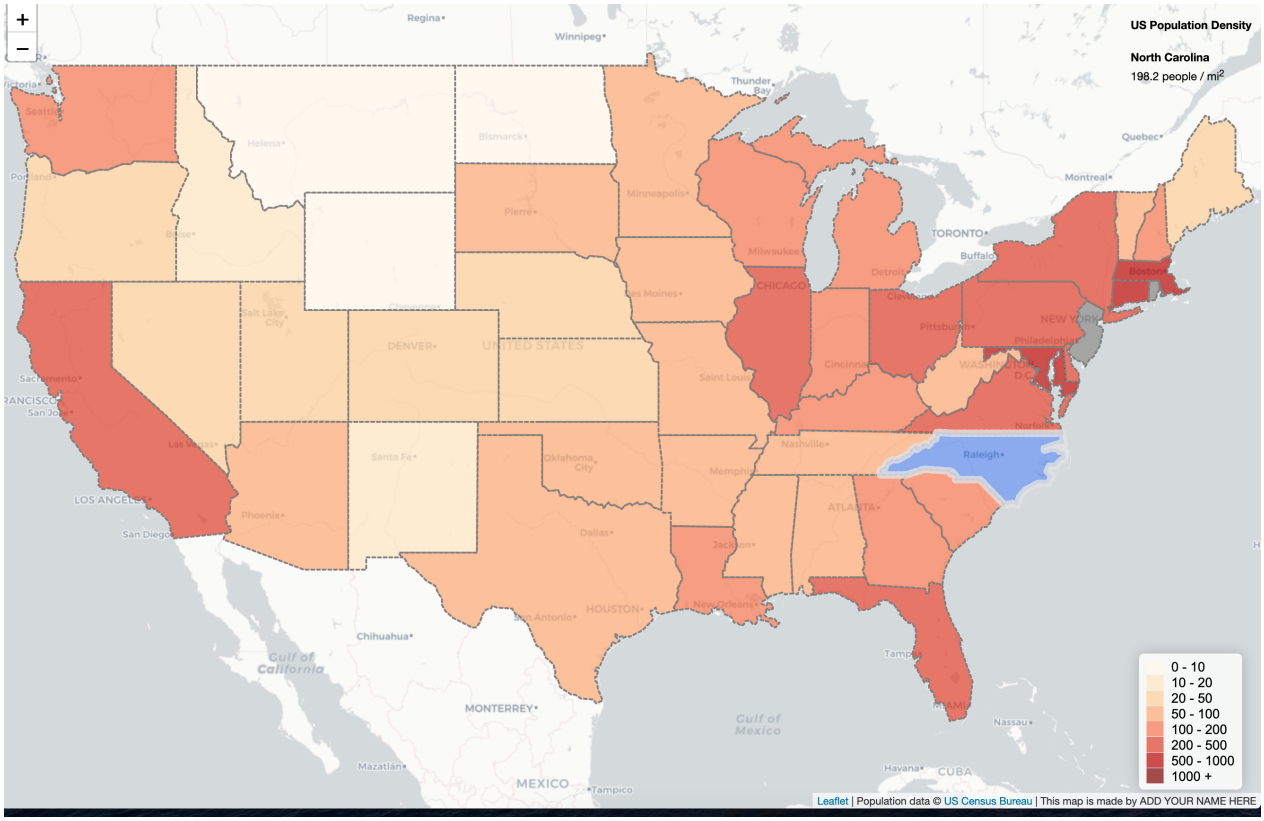
Web Map Interactions and Choropleth Symbolization

Objectives

- Understand Choropleth symbology in web map format;
- Learn to color thematic layers;
- Add map interactions;
- Create custom map controls.

This lecture creates a shaded interactive choropleth map of US States Population Density with the help of GeoJSON and some custom controls. It will assist you with the coding for the next lab.

> **Note:** This lecture is based on the leaflet tutorial here. It's a similar web map, however the code is changed for additional functionality in the web map. Specifically, this map uses Jquery and Chroma.js.



> **Note:** Remember from Cartography and GIS, a choropleth map is a thematic map in which areas are shaded in proportion to the measurement of the statistical variable being displayed on the map, such as population density or per-capita income. The choropleth map provides an easy way to visualize how a measurement varies across a geographic area or it shows the level of variability within a region.

1. Data source

We'll be creating a visualization of population density per US state. As the amount of data (state shapes and the density value for each state) is not very big, the most convenient and simple way to store and then display it is GeoJSON.

The states data are located in the assets folder for this lecture on ASULearn. Below is an example of a feature in our GeoJSON data `us-states.geojson`:

```
{
  "type": "Feature",
  "properties": {
    "name": "North Carolina",
    "density": 198.2
  },
  "geometry": ...
    ...
}
```

The GeoJSON with state shapes is from Mike Bostock of D3, joined with population density values from the US Census Bureau 2010.

2. Empty html page and prerequisite libraries

Start by creating an empty html page and loading prerequisite libraries. Other than the `leaflet` library, we will use `chroma.js` to shade states on this map, `jQuery` to manipulate html elements, and `leaflet.ajax` to load geojson data. We will add these three libraries in the `head` element as shown in the snippet below. For future work, remember that it is important to load these libraries when building the code for a new web map.

A `template.html` file is provided for you in the materials for this lecture. You may use this file to load the code below and begin programming. Remember to rename it as `index.html` in your project folder for this work.

```
<head>
    <title>Adding Web Map Interactions</title>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.0/dist/leaflet.css"/>
    <style>
        ...
    </style>
    <script src="https://unpkg.com/leaflet@1.7.0/dist/leaflet.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet-ajax/2.1.0/leaflet.ajax.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/chroma-js/1.3.4/chroma.min.js"></script>
</head>
<body>
    <script>
        //Our JavaScript will go here.
    </script>
</body>
</html>
```

2. Creating the map object, adding the base layer and states file

The first step is to add our states data on a map using a CartoDB style for a grayscale tiled basemap. To begin, create a div element that will hold the map.

```
<div id='map'></div>
```

Rembember, this snippet belongs in the `<body>` section of your code before the `<script>` tag.

Next, create the javascript object of the map, and anchor it to the div element. Remember, this snippet belongs at the top of the `<script>` section within the `<body>` of the document.

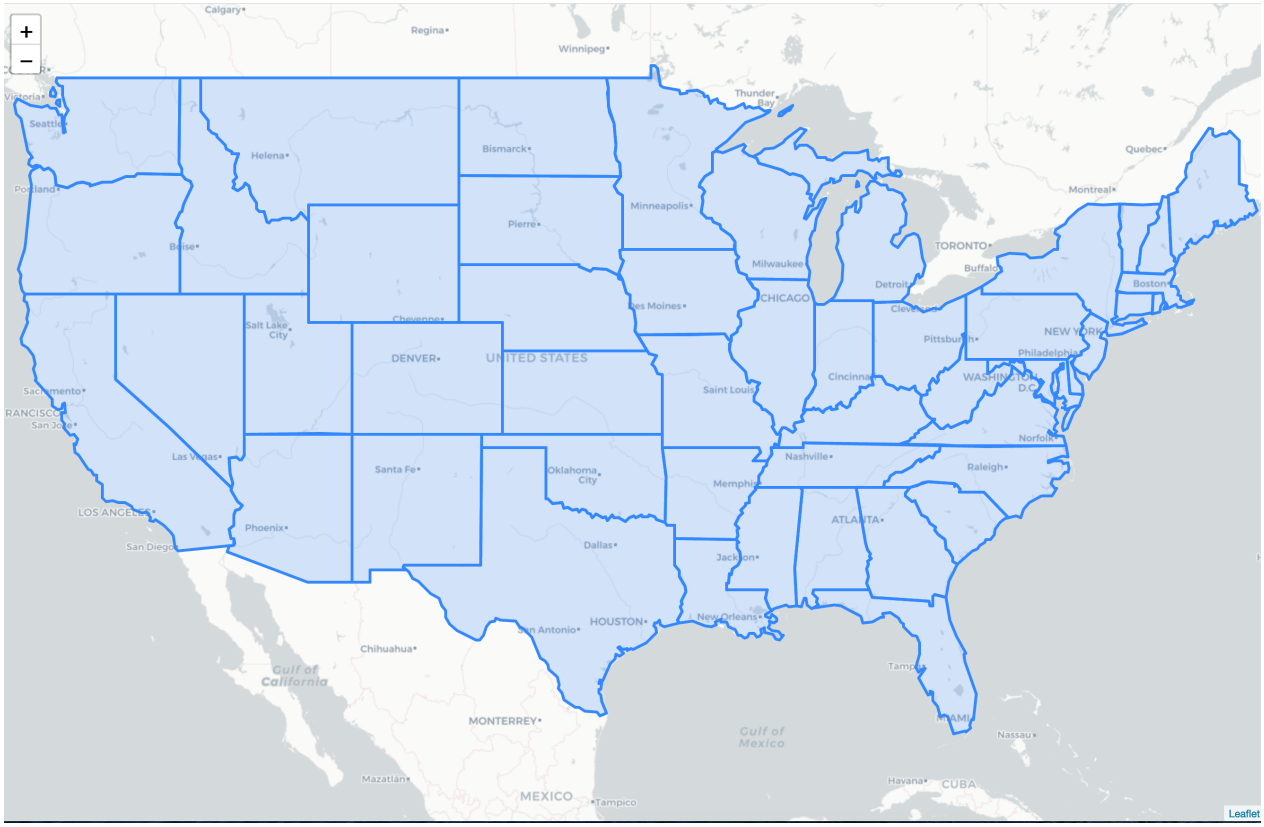
```
// 1. create the map object and the base layer.
var map = L.map('map').setView([37.8, -96], 5); //latlong coordinates centered over US
L.tileLayer('http://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}.png').addTo(map);
```

Add the style class for the map, which will expand the map to full screen.

```
//this snippet belongs within the <style> tags in the <head> of our document
html { height:100%;}
body {
    height:100%;
    padding: 0;
    margin: 0;
}
#map {
    width: 100%;
    height: 100%;
}
```

Finally, add the GeoJSON states data to the map. Create a step 3 within the `<body>`, `<script>` section of your document. Specifically, the step for adding the states file will be step 3.4 because we will add other intermediate steps as we add functionality and style to the map below.

```
// 3. add states file to the map

//3.4 Add the states GeoJSON layer to the map
  geojson = L.geoJson.ajax("assets/us-states.geojson").addTo(map);
```



3. Adding colors to states

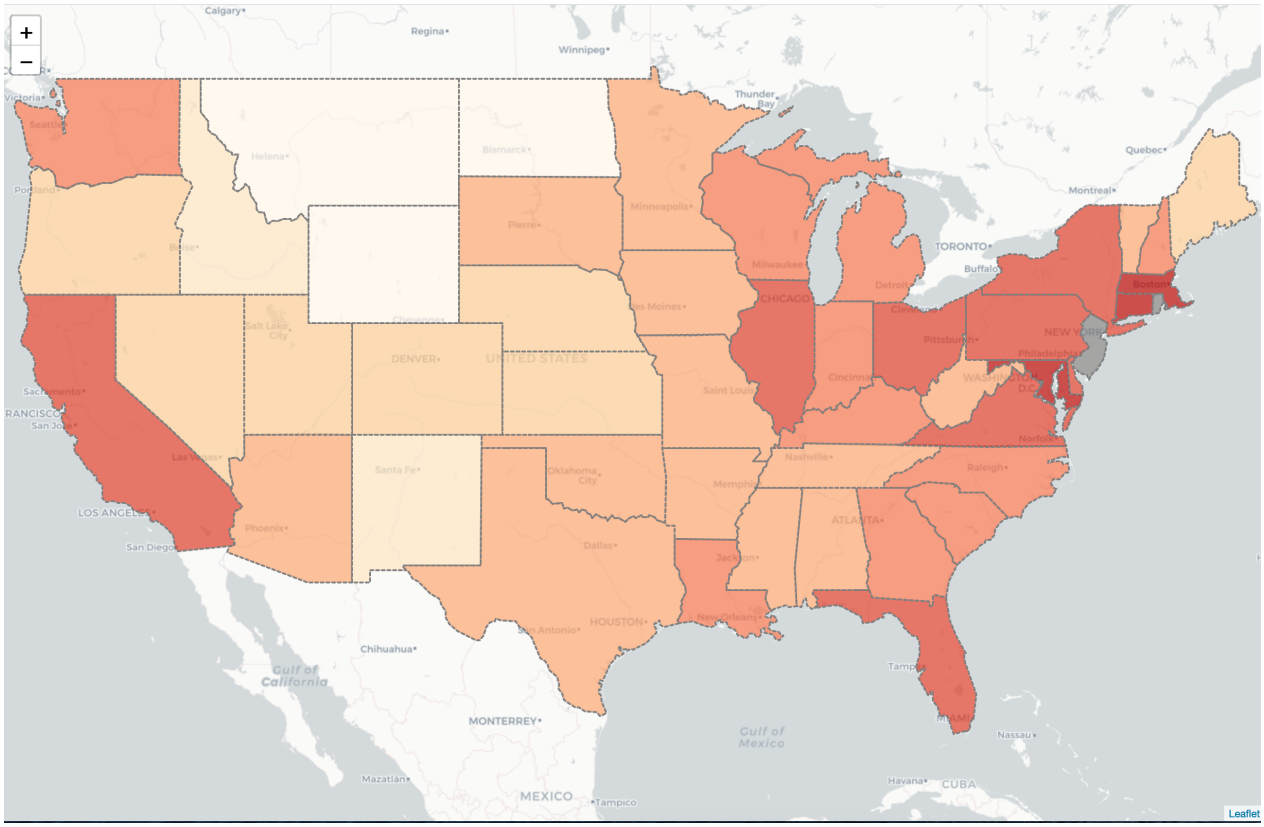
States should be shaded according to their population density. There is a tutorial on color at W3Schools that provides some helpful guidance for working with color on the web. Choosing appropriate colors for a map can be especially tricky, but remember from previous exercises that we should use ColorBrewer to maximize our chances of choosing a proper color palette for our data. Here's the next section of code, which should be added as section 2 in your JavaScript.

```
// 2. Create the chorepleth map with interactive functions.
// determine the number of classes and their respective break values.
// the class breaks are listed below, there will be eight classes
//starting with a class 1-10, and ending with a class 1000+
var grades = [0, 10, 20, 50, 100, 200, 500, 1000];

// now determine the color ramp.
// the number of colors is determined by the number of classes.
// try different interpolation method lch, lab, hsl
// for example
// var colors = chroma.scale(['yellow', 'navy']).mode('hsl').colors(grades.length);
// or
var colors = chroma.scale('OrRd').colors(grades.length);
// you can change the forward slashes to see the difference in these two lines
// notice how i've subbed in the color palette name 'OrRd' from color brewer
//this provides a sequential color palette from orange to red.

// get the color based on the class of the input value
function getColor(d) {
    for (var i = 0; i < grades.length - 1; i++) {
        if ( d > grades[i] && d < grades[i+1] ) return colors[i];
    }
    if (d > grades[grades.length - 1]) return colors[grades.length];
}
```

Next we define a styling function for our GeoJSON layer so that its fillColor depends on the feature.properties.density property, also adjusting the appearance a bit and adding a nice touch with dashed stroke for the state boundaries.

```
// determine the style class based on the input feature
// note, we are still working in section 2.
function style(feature) {
    return {
        weight: 2,
        opacity: 1,
        color: 'grey',
        dashArray: '3',
        fillOpacity: 0.7,
        fillColor: getColor(feature.properties.density)
    };
}

//the line below already exists in your code under step 3.4
geojson = L.geoJson.ajax("assets/us-states.geojson").addTo(map);

//however you must edit it like the following in order to add the
//style properties we just created
geojson = L.geoJson.ajax("assets/us-states.geojson", {
  style: style
}).addTo(map);
```

You should see a shaded choropleth map with a sequential color palette now.



4. Adding Interaction

We can emphasize certain information on the map by adding interactions. Let's make states show some additional information when the mouse is hovered over them. First we'll define an event listener for the layer mouseover event:

```
// note we're adding a new section in the appropriate location to do this
// 3.2.1 highlight a feature when the mouse hovers on it.
function highlightFeature(e) {
    // e indicates the current event, we could call it anything
    var layer = e.target;
    //the target captures the object that the event associates with
    layer.setStyle({
        weight: 8,
        opacity: 0.8,
        color: '#e3e3e3',
        fillColor: '#1c5ee3',
        fillOpacity: 0.5
    });
    // then bring the layer to the front.
    layer.bringToFront();
    // select the update class, and update the contet with the input value.
    // remember whenver we call feature.properties, we're looking in the attributes
    // for a particular layer - state name and state density, for example
    info.update(layer.feature.properties); // this line will be called later
}
```

This only shows us what happens when the mouse hovers over the layer, using `e.target`. A thick grey border is our highlight effect, which brings our feature to the front so that the border doesn't clash with nearby states.

We still have to define what happens on mouseout for the event to work properly:

```
// continuing in a new subsequent section
// 3.2.3 reset the hightlighted feature when the mouse is out of its region.
```

```
function resetHighlight(e) {
    geojson.resetStyle(e.target);
      info.update(); // this line will be called later
}
```

The useful `geojson.resetStyle` method will reset the layer style to its default state (defined by our style function). For this to work, make sure our GeoJSON layer is accessible through the geojson variable by defining it before our listeners and assigning the layer to it later. In other words, we need to go back in our code and create a new section 3.1 where we can call the following:

```
// 3.1 declare an empty GeoJSON object
var geojson = null;
// then we will continue to work on our event listeners
```

We also need a click listener that zooms to the state level.

```
// 3.2.2 zoom to the highlighted feature when the mouse clicks it.
function zoomToFeature(e) {
    map.fitBounds(e.target.getBounds());
}
```

Now use the onEachFeature option to add the listeners on our state layers.

```
// 3.3 add these event the layer obejct.
function onEachFeature(feature, layer) {
    layer.on({
        mouseover: highlightFeature,
        click: zoomToFeature,
        mouseout: resetHighlight
    });
}

// 3.4 assign the geojson data path, style option and onEachFeature option.
// And then add the geojson layer to the map.
geojson = L.geoJson.ajax("assets/us-states.geojson", {
    style: style,
    onEachFeature: onEachFeature
}).addTo(map);
```

This makes the states highlight nicely on hover and gives us the ability to add other interactions inside our listeners.

5. Custom Info Control

We could use the usual popups on click to show information about different states, but let's learn a different way — instead showing it on state hover inside a custom control.

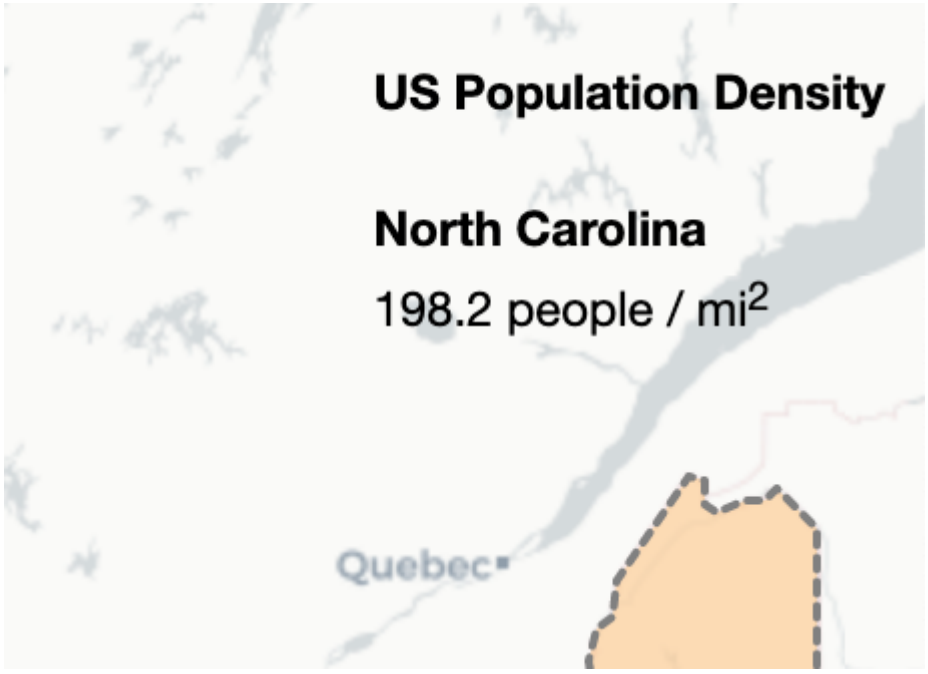
Here's the code for our control, which you can add to section 3.1 in our `<script>` tag of the body. Note, that this code is pasted into the script before section 3.2.1.

```
var info = L.control();

info.onAdd = function (map) {
    this._div = L.DomUtil.create('div', 'info'); // create a div with a class "info"
    this.update();
    return this._div;
};
```

```
// method that we will use to update the control based on feature properties passed
info.update = function (props) {
    this._div.innerHTML = '<h4>US Population Density</h4>' +  (props ?
        '<b>' + props.name + '</b><br />' + props.density + ' people / mi<sup>2</sup>'
        : 'Hover over a state');
};

info.addTo(map);
```

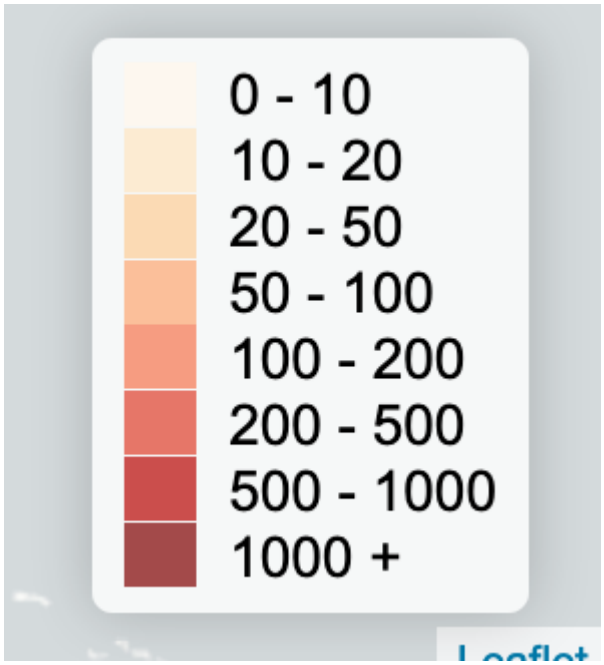
The control also needs some CSS styles to look nice:

```
//remember, these styles will go within our <style> tags in the <head> of our document
// beneath the line where we added style information to 'full screen' the map
.info {
    z-index: 1000;
    position: absolute;
    right: 20px;
    top: 20px;
    padding: 6px 8px;
    font: 14px Arial, Helvetica, sans-serif;
    text-align: right;
    background: white;
    background: rgba(255, 255, 255, 0.8);
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
    border-radius: 5px;
}
.info h1 {
    font-size: 16px;
    margin: 0 0 5px;
    color: #777777;
}
```



6. Custom Legend Control

Creating a control with a legend is easier, since it is static and doesn't change on hover. Here's what the legend will look like.



And, here's our code that belongs at the top of the `<body>`, beneath where we added the other `<div>` tags.

```
<div class='legend'></div>
```

And the javascript code:

```
// 4. create the legend
// note that line breaks have been added and may need to be removed
var labels = [];
for (var i = 0; i < grades.length - 1; i++) {
    labels.push('<i style="background:' + colors[i] +
     '"></i> ' + grades[i] + ' - ' + grades[i + 1]);
}
labels.push('<i style="background:' + colors[grades.length - 1] +
'"></i> ' + grades[grades.length - 1] + ' +');
$(".legend").html(labels.join('<br>'));
```

The legend will not render on the map until we've defined the CSS styles for the control (we also reuse the info class defined earlier). Add the following lines to the bottom section of the `<style>` tags.

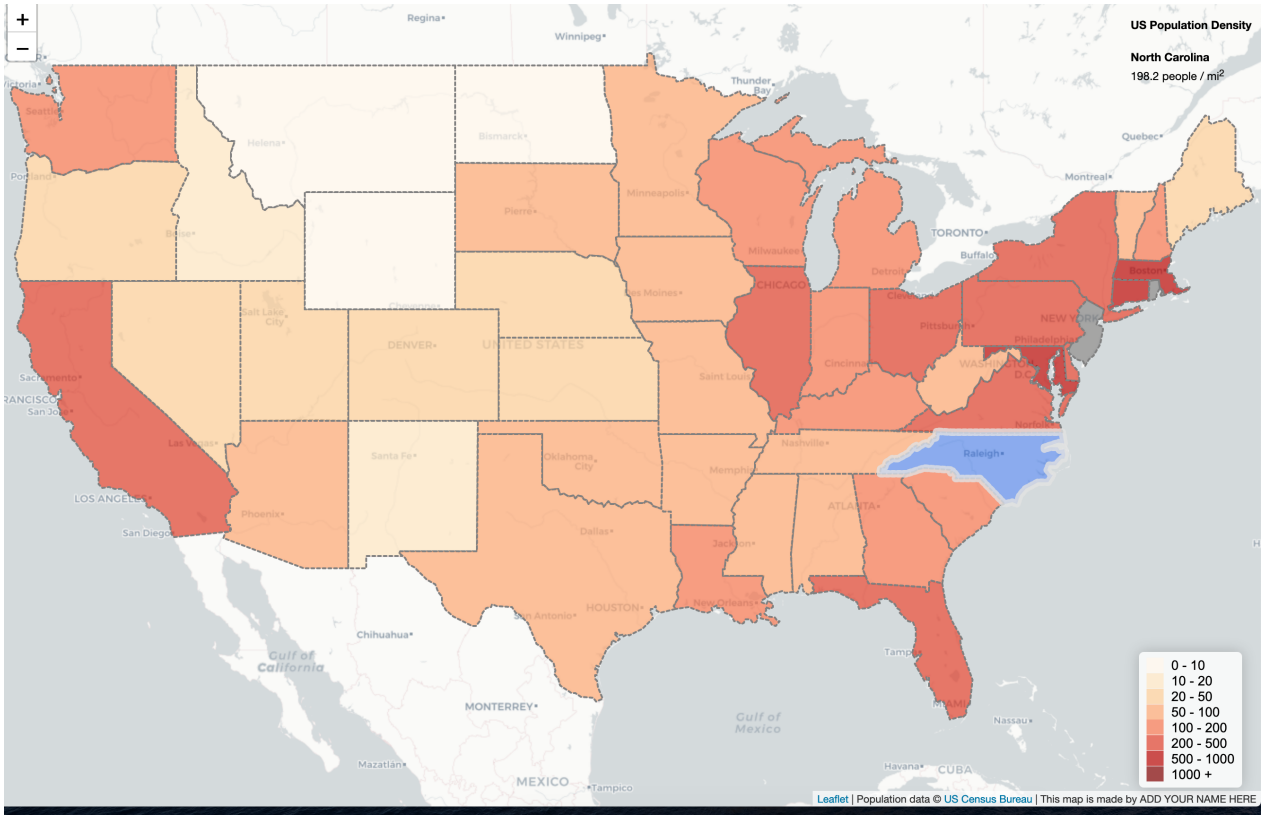
```
/*legend panel*/
.legend {
    z-index: 1000;
    position: absolute;
    right: 20px;
    bottom: 20px;
    padding: 6px 8px;
    font: 14px Arial, Helvetica, sans-serif;
    background: white;
    background: rgba(255, 255, 255, 0.8);
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
    border-radius: 5px;
}
.legend i {
    width: 18px;
    height: 16px;
    float: left;
    margin-right: 8px;
```

```
    opacity: 0.7;
}
```

7. Credits

At last, add data sources and author name to the bottom of the main `<script>`.

```
// 5. create the credits
//note that line breaks have been added and may need to be removed

map.attributionControl.addAttribution('Population data &copy;
<a href="http://census.gov/">US Census Bureau</a> |
This map is made by ADD YOUR NAME HERE</a>');
```



Reference

[1] Choropleth map https://en.wikipedia.org/wiki/Choropleth_map

[2] Color for styling thematic layers http://colorbrewer2.org/

[3] http://www.w3schools.com/colors/default.asp