Module IV: D3

In this module we want to explore the D3 javascript library a little more. A great way to do this is to use it to help join a file containing the U.S. state outlines to some agricultural production data, and simultaneously plot some city population data. Before we begin you should crack open your data files to make sure you understand what we are looking at! Notice that the Agricultural data has no geographic information, which is why we will need to merge it with the geoJSON file, but the city population data does have latitude and longitude data. To start off lets open our index.html file and get it set up, add this in the body of your html to give our map a place to live:

<div id="myMap"></div>

Next we will need to add the links to our scripts after the body in your html:

<script type="text/javascript" src="d3/d3.v3.js"></script>

<script type="text/javascript" src="js/script.js"></script>

Ok now lets crack open our javascript file and get to work. Start by creating two new variables, one for width and one for height that we will use to help project our data:

var w = 500;

var h = 300;

We can use the previous variables we made to generate our projection for the map:

var projection = d3.geo.albersUsa()

.translate([w/2, h/2])

.scale([500]);

Now we need to define a path generator to apply our projection:

var path = d3.geo.path()

.projection(projection);

We create a function to define a quantized scale to sort data values into buckets of color, note that the colors are taken from colorbrewer.js, included in the D3 download:

var color = d3.scale.quantize()

.range(["rgb(237,248,233)","rgb(186,228,179)","rgb(116,196,118)","rgb(49,163,84)","rgb(0,109,44)"]);

Time to work on the map itself, first we create an SVG element:

var svg = d3.select("#myMap")

.append("svg")

.attr("width", w)

.attr("height", h);

Now lets build a function that loads in the agriculture data and sets the input domain for the color scale:

d3.csv("data/us-ag-productivity-2004.csv", function(data) {

color.domain([

d3.min(data, function(d) { return d.value; }),

d3.max(data, function(d) { return d.value; })

]);

Great! Next we need to create a function that will load in GeoJSON data then merge it with the agricultural data:

d3.json("data/us-states.json", function(json) {

To merge the data we loop through once for each ag. data value:

for (var i = 0; i < data.length; i++) {

Grab the state name:

var dataState = data[i].state;

Grab the data value, and convert from string to float:

var dataValue = parseFloat(data[i].value);

Now find the corresponding state inside the GeoJSON:

for (var j = 0; j < json.features.length; j++) {

var jsonState = json.features[j].properties.name;

if (dataState == jsonState) {

Copy the data value into the JSON:

json.features[j].properties.value = dataValue;

Of course once we get a match and the data we need, we want to stop looking through the JSON file!

break;

}

}

}

And now we will bind the data and create a separate path per GeoJSON feature (remember it holds our projection to everything is plotted correctly!):

var paths = svg.selectAll("path")

.data(json.features)

.enter()

.append("path")

.attr("d", path)

Of course with the paths created we will still need to figure out what color will fill our states:

.style("fill", function(d) {

Start by getting data value and returning the corresponding color using our color scale we created earlier. Note that we use an “if-else” statement to handle any cases in which there is no data:

var value = d.properties.value;

if (value) {

return color(value);

} else {

return "#ccc";

}

});

On to the cities data, lets load it in:

d3.csv("data/us-cities.csv", function(data) {

And then create a circle that has a radius scaled by the population. We start by creating our circles:

svg.selectAll("circle")

.data(data)

.enter()

.append("circle")

Then get the (correctly projected) Latitude and Longitude:

.attr("cx", function(d) {

return projection([d.lon, d.lat])[0];

})

.attr("cy", function(d) {

return projection([d.lon, d.lat])[1];

})

And calculate the radius using the population for the given city, finishing up by giving the style that will be used for the cicles (fill,stroke, and opacity):

.attr("r", function(d) {

return Math.sqrt(parseInt(d.population) \* 0.00004);

})

.style("fill", "grey")

.style("stroke", "white")

.style("opacity", 0.75);

});

Of course we need to close the rest of the functions up!

});

});

With that all done its time to check your handy work! Open your new map in your browser!