Module III: Leaflet Interactions and API

We would like to try our hand at creating an interaction that occurs outside of a map that we make. In this example we will try to create a side bar with click-able links that coincide with locations on a map. Once clicked we would like to have the map “react” by showing the correct pop-up at the correct location corresponding to the link that was clicked. This will also require us to use some d3 and a few other libraries. Lets start by setting up our index.html file, crack it open and lets add the following to the head:

<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />

This sets up a “full-screen” map. Now we can set up links to our stylsheet and to Leaflets (notice we have downloaded our Leaflet library this time so we are accessing it within our directory):

<link type="text/css" rel="stylesheet" href="libraries/leaflet-0.7.3/leaflet.css"/>

<link type="text/css" rel="stylesheet" href="css/stylesheet.css"/>

Now go the body of the index.html code and add a new div for the sidebar we want to create:

<div id="list"></div>

Great! To get things moving we are going to need to add links to all of our libraries and scripts, for this example we will have several that need to be added after the body of index.html:

<script type="text/javascript" src="libraries/leaflet-0.7.3/leaflet.js"></script>

<script type="text/javascript" src="libraries/jQuery/jquery-1.11.2.min.js"></script>

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

<script type="text/javascript" src="libraries/underscore/underscore-min.js"></script>

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

Ok! Now lets crack open our javascript file and get to work! First thing is first, set our map view:

var map = L.map('map').setView([40.65,-73.93], 12);

Set a tile layer to be CartoDB tiles:

var CartoDBTiles = L.tileLayer('http://{s}.basemaps.cartocdn.com/light\_all/{z}/{x}/{y}.png',{

attribution: 'Map Data &copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a> Contributors, Map Tiles &copy; <a href="http://cartodb.com/attributions">CartoDB</a>'

});

Add these tiles to our map:

map.addLayer(CartoDBTiles);

Set data layer as global variable so we can use it in the layer control below:

var acsGeoJSON;

Use jQuery, get geoJSON to grab geoJson layer, parse it, then plot it on the map using the plotDataset function:

$.getJSON( "data/acs\_data\_joined.geojson", function( data ) {

var dataset = data;

Draw the dataset on the map:

plotDataset(dataset);

Create the sidebar with links to fire polygons on the map:

createListForClick(dataset);

});

Function to plot the dataset passed to it:

function plotDataset(dataset) {

acsGeoJSON = L.geoJson(dataset, {

style: acsStyle,

onEachFeature: acsOnEachFeature

}).addTo(map);

Create layer controls:

createLayerControls();

}

Function that sets the style of the geojson layer:

var acsStyle = function (feature, latlng) {

var calc = calculatePercentage(feature);

var style = {

weight: 1,

opacity: .25,

color: 'grey',

fillOpacity: fillOpacity(calc[2]),

fillColor: fillColorPercentage(calc[2])

};

return style;

}

function calculatePercentage(feature) {

var output = [];

var numerator = parseFloat(feature.properties.ACS\_13\_5YR\_B07201\_HD01\_VD14);

var denominator = parseFloat(feature.properties.ACS\_13\_5YR\_B07201\_HD01\_VD01);

var percentage = ((numerator/denominator) \* 100).toFixed(0);

output.push(numerator);

output.push(denominator);

output.push(percentage);

return output;

}

Function that fills polygons with color based on the data:

function fillColorPercentage(d) {

return d > 9 ? '#006d2c' :

d > 7 ? '#31a354' :

d > 5 ? '#74c476' :

d > 3 ? '#a1d99b' :

d > 1 ? '#c7e9c0' :

'#edf8e9';

}

Function that sets the fillOpacity of layers -- if % is 0 then make polygons transparent:

function fillOpacity(d) {

return d == 0 ? 0.0 :

0.75;

}

Empty L.popup so we can fire it outside of the map:

var popup = new L.Popup();

Set up a counter so we can assign an ID to each layer:

var count = 0;

On each feature function that loops through the dataset, binds popups, and creates a count:

var acsOnEachFeature = function(feature,layer){

var calc = calculatePercentage(feature);

Let's bind some feature properties to a pop up with an .on("click", ...) command. We do this so we can fire it both on and off the map:

layer.on("click", function (e) {

var bounds = layer.getBounds();

var popupContent = "<strong>Total Population:</strong> " + calc[1] + "<br /><strong>Population Moved to US in Last Year:</strong> " + calc[0] + "<br /><strong>Percentage Moved to US in Last Year:</strong> " + calc[2] + "%";

popup.setLatLng(bounds.getCenter());

popup.setContent(popupContent);

map.openPopup(popup);

});

We'll now add an ID to each layer so we can fire the popup outside of the map:

layer.\_leaflet\_id = 'acsLayerID' + count;

count++;

}

function createLayerControls(){

Add in layer controls:

var baseMaps = {

"CartoDB Basemap": CartoDBTiles,

};

var overlayMaps = {

"Percentage Moved to US in Last Year": acsGeoJSON,

};

Add control:

L.control.layers(baseMaps, overlayMaps).addTo(map);

}

Add in a legend to make sense of it all, and create a container for the legend and set the location:

var legend = L.control({position: 'bottomright'});

Using a function, create a div element for the legend and return that div:

legend.onAdd = function (map) {

A method in Leaflet for creating new divs and setting classes:

var div = L.DomUtil.create('div', 'legend'),

amounts = [0, 1, 3, 5, 7, 9];

div.innerHTML += '<p>Percentage Population<br />That Moved to US in<br />the Last Year</p>';

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

div.innerHTML +=

'<i style="background:' + fillColorPercentage(amounts[i] + 1) + '"></i> ' +

amounts[i] + (amounts[i + 1] ? '% &ndash;' + amounts[i + 1] + '%<br />' : '% +<br />');

}

return div;

};

Add the legend to the map:

legend.addTo(map);

Function to create a list in the right hand column with links that will launch the pop-ups on the map:

function createListForClick(dataset) {

Use d3 to select the div and then iterate over the dataset appending a list element with a link for clicking and firing, first we'll create an unordered list ul elelemnt inside the <div id='list'></div>. The result will be <div id='list'><ul></ul></div>

var ULs = d3.select("#list")

.append("ul");

Now that we have a selection and something appended to the selection, let's create all of the list elements (li) with the dataset we have:

ULs.selectAll("li")

.data(dataset.features)

.enter()

.append("li")

.html(function(d) {

return '<a href="#">' + d.properties.ACS\_13\_5YR\_B07201\_GEOdisplay\_label + '</a>';

})

.on('click', function(d, i) {

console.log(d.properties.ACS\_13\_5YR\_B07201\_HD02\_VD01);

console.log(i);

var leafletId = 'acsLayerID' + i;

map.\_layers[leafletId].fire('click');

});

}

Lets add data from the API now, set a global variable to use in the D3 scale below.Use jQuery geoJSON to grab data from API:

$.getJSON( "https://data.cityofnewyork.us/resource/erm2-nwe9.json?$$app\_token=rQIMJbYqnCnhVM9XNPHE9tj0g&borough=BROOKLYN&complaint\_type=Noise&status=Open", function( data ) {

var dataset = data;

// draw the dataset on the map

plotAPIData(dataset);

});

Create a leaflet layer group to add your API dots to so we can add these to the map:

var apiLayerGroup = L.layerGroup();

Since these data are not geoJson, we have to build our dots from the data by hand:

function plotAPIData(dataset) {

Set up D3 ordinal scle for coloring the dots just once:

var ordinalScale = setUpD3Scale(dataset);

Loop through each object in the dataset and create a circle marker for each one using a jQuery for each loop:

$.each(dataset, function( index, value ) {

Check to see if lat or lon is undefined or null:

if ((typeof value.latitude !== "undefined" || typeof value.longitude !== "undefined") || (value.latitude && value.longitude)) {

Create a leaflet lat lon object to use in L.circleMarker:

var latlng = L.latLng(value.latitude, value.longitude);

var apiMarker = L.circleMarker(latlng, {

stroke: false,

fillColor: ordinalScale(value.descriptor),

fillOpacity: 1,

radius: 5

});

Bind a simple popup so we know what the noise complaint is:

apiMarker.bindPopup(value.descriptor);

Add dots to the layer group:

apiLayerGroup.addLayer(apiMarker);

}

});

apiLayerGroup.addTo(map);

}

function setUpD3Scale(dataset) {

Create unique list of descriptors, first we need to create an array of descriptors:

var descriptors = [];

Loop through descriptors and add to descriptor array:

$.each(dataset, function( index, value ) {

descriptors.push(value.descriptor);

});

Use underscore to create a unique array:

var descriptorsUnique = \_.uniq(descriptors);

Create a D3 ordinal scale based on that unique array as a domain:

var ordinalScale = d3.scale.category20()

.domain(descriptorsUnique);

return ordinalScale;

}

Wow! That was a lot to code! Now what? Well, you guessed it, we need to crack open our CSS stylesheet and set up our styles:

body {

padding: 0;

margin: 0;

}

html, body, #map {

height: 100%;

}

#map {

width: 65%;

display: inline-block;

}

#list {

width: 32%;

height: 100%;

float: right;

margin: 20px 0px 0px 20px;

overflow: auto;

display: inline-block;

}

The code above set the style and space for our map (full screen remember!) as well as our sidebar or “list”. And if you remember Module II you will know that we now need to add the CSS for our legend:

.legend {

line-height: 18px;

padding: 6px 8px;

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: 18px;

float: left;

margin-right: 8px;

opacity: 0.7;

}

.circle {

background-color: #909090;

}

Awesome!! You are finally ready to check your handy work, open your index.html in a browser (it may take a few seconds to load!!)