**CDC\_Data\_Visualization – Mapping Documentation**

var token = 'OWUqA3XHifIffWaYdBATd4TVt';

var url = `https://chronicdata.cdc.gov/resource/hn4x-zwk7.json?$$app\_token=${token}&$limit=2000`;

var colors = ['#FFEDA0', '#FED976', '#FEB24C', '#FD8D3C', '#FC4E2A', '#E31A1C', '#BD0026', '#800026'].reverse(); // lowest <> highest

var promises = [];

* Declared variables (token, url, colors, and promises)
  + For the url, we included the dataset, as well as the app token and a limit of 2,000 records; by default, the limit is 1,000 for this specific API, so increasing the limit (if we have the API key), we can go up to 50,000 per API call.
  + We didn’t know how many records there were per state but we knew that if there was 52 states/territories to be pulled from and we had 1,000 per state, then we’d at least need 52,000 records
  + Retrieved data for each state one-at-a-time and increased the limit for each call—determined 1,652 records per state and pulled all of that data out.
* Console logged features length for statesData to see how many “states” were in the array + console logged statesData to inspect data further
* Const – constant; prevents you from redefining in scope (ever); a const object can be reassigned properties but you can’t reassign entire variable

statesData.features.forEach(function (stateData) {

const state = stateData.properties.name;

* Created loop and:
  + Overwrote statesData
  + Retrieved name of state (stateData) from statesData and assigned it to a constant (state)

const promise = d3.json(`${url}&locationdesc=${state}`).then(function (data) {

* Used d3.json() to convert items into an array of objects and assign to the constant named promise
  + D3.json() takes a JSON file as input and converts it into an array of objects
  + Interpolated – inserting/injecting variables inside a string (use tick marks)
  + Similar to concatenating, which would have been: url + “&locationdesc=” + state

// Determine total # of records and # of records per year

console.log(state, data.reduce(function (result, item) {

result.total = (result.total || 0) + 1;

result[item.yearstart] = (result[item.yearstart] || 0) + 1;

return result;

}, {}));

* Console logged state, and totals of states and year start
* Reduce allows you to reduce the array into a different data type; converting from array type to a new type of data; definition includes accumulator + item in array;
* With the console log, we just wanted to get a total number of records (result total OR start at 0 and increment by 1.

const densities = data.filter(function (item){

return item.data\_value;

}).map(function (item) {

* Assigned densities to a constant that holds filtered and then mapped data to create an array of those numerical values:
  + First, we filtered our items to look for only items with data\_value, then mapped those filtered data back into our item as decimals.
* let average = 0;
* Initialized average by setting it = 0
  + Let – has the same scope properties as const but allows you to re-assign within the same scope

if (densities.length) {

// reduce allows you to map an array to a different data type

average = densities.reduce(function (sum, density) {

return sum + density

}, 0) / densities.length;

* Created an “If” statement to determine if “densities” exist for the state, and if so, calculate the average.

// overwrite density property in stateData (for each state inside map)

stateData.properties.density = average;

* Reassigned the density property in stateData to “average.”

// push our promise into our promises array

promises.push(promise);

* Saved our promise to an array of promises.

Promise.all(promises).then(function () {

const densities = statesData.features.map(function (state) {

return state.properties.density;

* Waited for all promises (API calls) to be resolved using promise.all and when done, we used .then() to extract the densities from all statesData (which is the data\_value in our case).

// find min & max to determine color variation

const min = Math.floor(Math.min.apply(null, densities));

const max = Math.ceil(Math.max.apply(null, densities));

const steps = colors.length;

const step = (max - min) / steps;

console.log('Min', min);

console.log('Max', max);

console.log('Steps', steps, 'x', step);

* Found the minimum and maximum density (data\_value). We rounded down for the minimum and rounded up for the maximum.

// Create the map object

var myMap = L.map("map", {

center: [37.0902, -95.7192],

zoom: 5

});

* Created the map object

Continued on next page….

Remainder of code: styled the map, created functional interactions, and created the legend:

// Add base layer/tile layer to the map (background of map)

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {

attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'

}).addTo(myMap);

// Add color

function getColor(d) {

// from max to min, find the color for the given step

// we decrement (loop in reverse) so that we're finding highest values first

for (let value = max - step, i = colors.length - 1; value >= min; value -= step, i--) {

if (d >= value) {

console.log('Density', d);

console.log('Color', colors[i]);

return colors[i];

}

}

console.log('Density', d);

console.log('Color', colors[0]);

// return the first color by default (for the lowest/min density)

return colors[0];

}

//Style

function style(feature) {

console.log(feature.properties.name);

return {

fillColor: getColor(feature.properties.density),

weight: 2,

opacity: 1,

color: 'white',

dashArray: '3',

fillOpacity: 0.7

};

}

statesData.features.forEach(function (stateData){

console.log(stateData.properties.density)

});

// save geojson

var geojson;

// Add interaction

function highlightFeature(e) {

var layer = e.target;

layer.setStyle({

weight: 5,

color: '#800026',

dashArray: '',

fillOpacity: 0.7

});

layer.bringToFront()

info.update(layer.feature.properties); // update control for highlight

}

// Mouseout

function resetHighlight(e) {

geojson.resetStyle(e.target)

info.update(); // reset control

}

function onEachFeature(feature, layer) {

layer.on({

mouseover: highlightFeature,

mouseout: resetHighlight,

})

// show average obesity percentage (rounded to 2 decimal places)

layer.bindPopup(`<b>Average Obesity Level: ${(feature.properties.density).toFixed(2)}%</b>`);

console.log(feature)

}

// add control - hover box with info on top right

var info = L.control();

info.onAdd = function (myMap) {

this.\_div = L.DomUtil.create('div', 'info'); // create a div with a class "info"

this.update();

return this.\_div};

info.update = function (props) {

this.\_div.innerHTML = '<h4>US Average Obesity Levels</h4>' + (props ?

'<b>' + props.name + '</b><br />' + (props.density).toFixed(2) + '% of adults are obese'

: 'Hover over a state');

};

info.addTo(myMap)

// create legend

let legend = L.control({position: "bottomright"});

legend.onAdd = function() {

// make a div with class = "info legend"

let div = L.DomUtil.create("div", "info-legend");

let limits = [29, 29.71, 30.42, 31.13, 31.84, 32.55, 33.26, 34] // range evenly divided into 8

let labels = []

let legendInfo = "<h3>Adult Population with Obesity</h3>" +

"<div class = \"labels\">" +

"<div class = \"min\">" + limits[0] + "</div>" +

"<div class = \"max\">" + limits[limits.length-1].toLocaleString() +

"</div>"

"</div>";

div.innerHTML = legendInfo

limits.forEach(

function(limit, index)

{

// generate the li with each color and push to the labels array

labels.push("<li style=\"background: " + colors[index]

+"\"></li>");

}

);

div.innerHTML += "<ul" + labels.join("") + "</ul>";

return div;

}

legend.addTo(myMap)

// Add states to map

geojson = L.geoJson(statesData, {style: style, onEachFeature: onEachFeature}).addTo(myMap);

});

**Additional resources for understanding promises:**

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all>

<https://www.w3schools.com/js/js_promise.asp>

<https://www.geeksforgeeks.org/why-we-use-then-method-in-javascript/>

**Choropleth map resource:** <https://leafletjs.com/examples/choropleth/>