**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
  + Got 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
* 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