13.4.2 Map Multiple Points

Now that you have added a single marker to a map and changed some of the features of the marker, Sadhana wants you to iterate through an array of objects and map them. Other employees really like the branches you created, so Sadhana would like you to create a new branch for adding multiple points to a map.

Before we plot multiple markers and points, Sadhana wants you to create a new branch for mapping multiple points.

Create a new branch called "Mapping_Multiple_Points" with the following folder structure:

- Mapping_Multiple_Points
  - `index.html`
  - static
    - css
      - `style.css`
    - js
      - `config.js`
      - `logic.js`

Copy the necessary folders and files from your Mapping_Single_Points branch and add them to the Mapping_Multiple_Points folder.

Add Multiple Markers

When we added a single marker to our simple map, we assigned our `marker` variable to the Leaflet class `marker()` function. This function will only add one latitude and longitude to the map. To add more markers to the map, the latitudes and longitudes are usually nested in an array. To add a marker for each location, we have to iterate through the array and add each latitude and longitude to the map.

First, in the `logic.js` file, replace the `marker` variable (which we used to map one location) with the `cities` variable that references the five most populous cities array in the following code block. Then save the file:

```
// An array containing each city's location, state, and population.
let cities = [{
  location: [40.7128, -74.0059],
  city: "New York City",
  state: "NY",
  population: 8398748
},
{
  location: [41.8781, -87.6298],
  city: "Chicago",
  state: "IL",
  population: 2705994
},
{
  location: [29.7604, -95.3698],
  city: "Houston",
  state: "TX",
  population: 2325502
},
{
  location: [34.0522, -118.2437],
  city: "Los Angeles",
  state: "CA",
  population: 3990456
},
{
  location: [33.4484, -112.0740],
  city: "Phoenix",
  state: "AZ",
  population: 1660272
}
];
```

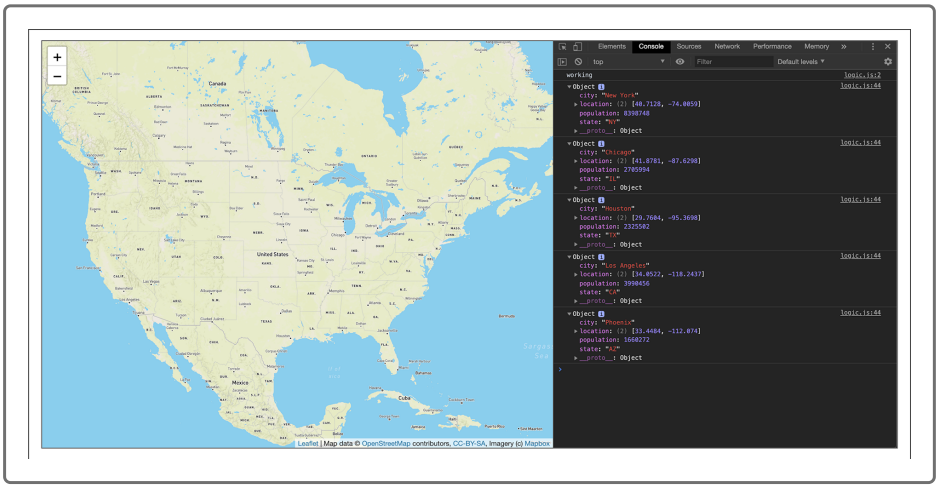
Next, we need to iterate through each `city` object and add each city location to the `marker()` function, which will, in turn, be added to the map.

Below the cities array, add the following code to iterate through the array. Inside the brackets, use the `console.log()` function to print each object in the array to the console:

```
// Loop through the cities array and create one marker for each city.
cities.forEach(function(city) {
 console.log(city)
});
```

Save the `logic.js` file and open the `index.html` file in your browser.

If we look at the console tab, we'll see that each object, or city, of the `cities` array is printed to the console.



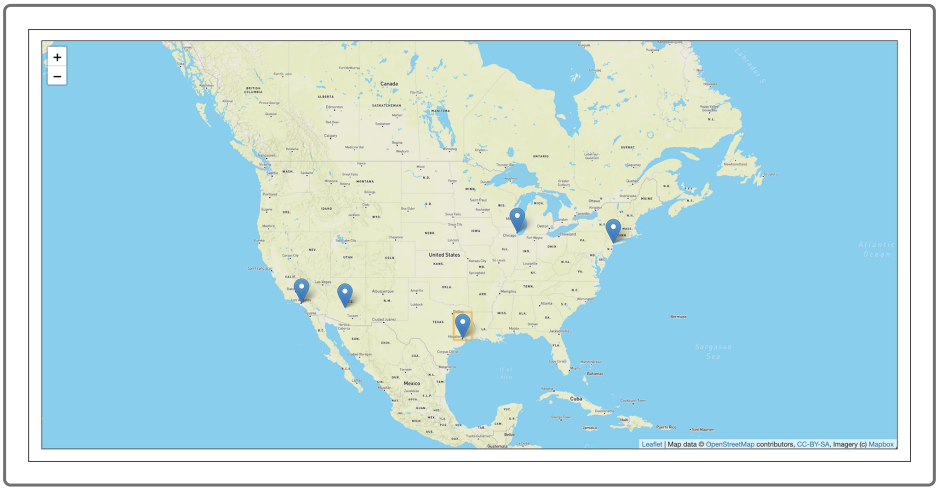
Now, add each city's location to the map by adding the location to the `marker()` function.

In the `forEach()` function, assign the `city` variable to each object of the `cities.js` file. Then, get the coordinates of each city by adding `city.location` in the `L.marker()` function. We can then add each location to the map with the `addTo()` function and pass the `map` object as the argument.

Add the following code to your `logic.js` file and save it:

```
// Loop through the cities array and create one marker for each city.
cities.forEach(function(city) {
    console.log(city)
    L.marker(city.location).addTo(map);
});
```

When you open the `index.html` file in your browser, the map will show a marker on each city in the `cities` array.



When handling large datasets, it's a best practice to have the data in an external file and refer to that file and dataset in the `logic.js` file.

Even though our `cities` array is not that large, let's create a new file in the "js" folder called `cities.js`. Cut the `cities` array data from the `logic.js` file, place it in the `cities.js` file, and save the file.

Next, in the `logic.js` file, where the `cities` array was located, add a variable and assign it to the `cities` array. Add the following code to the `logic.js` file:

```
// Get data from cities.js
let cityData = cities;
```

Now the `cities` array is assigned to the `cityData` variable, which means we'll need to replace `cities` with `cityData` in our `forEach()` function. Edit the `forEach()` function so that it looks like the following and save the `logic.js` file:

```
// Loop through the cities array and create one marker for each city.
cityData.forEach(function(city) {
    console.log(city)
    L.marker(city.location).addTo(map);
});
```

Now open the `index.html` in your browser to confirm these changes worked.

Uh-oh! Something went wrong, as shown in the following image:

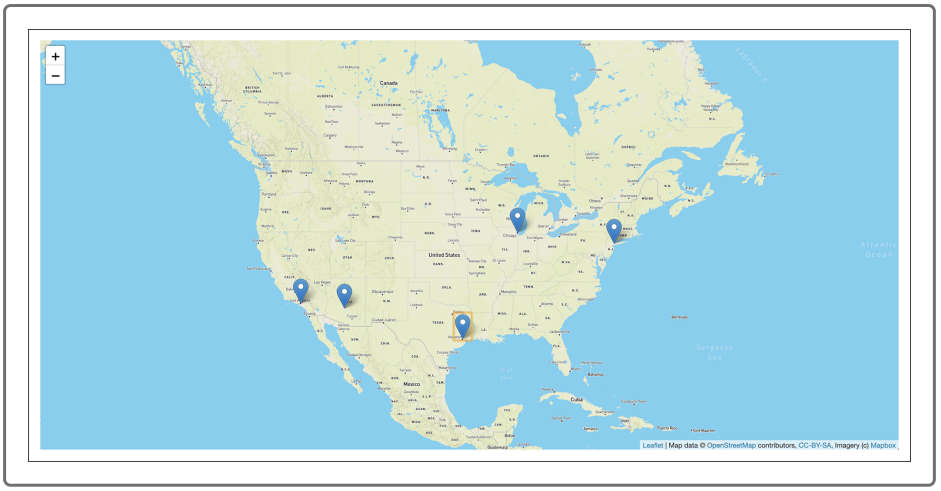
After you inspect the page using the DevTools, the console might have an error message that says `Uncaught ReferenceError: cities is not defined`. This means the `cities` array data can't be found.

To correct this error, in the body of the `index.html` file and before the path to the `logic.js` script, add a `<script>` file with the path to the JavaScript `cities.js` file, like this:

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

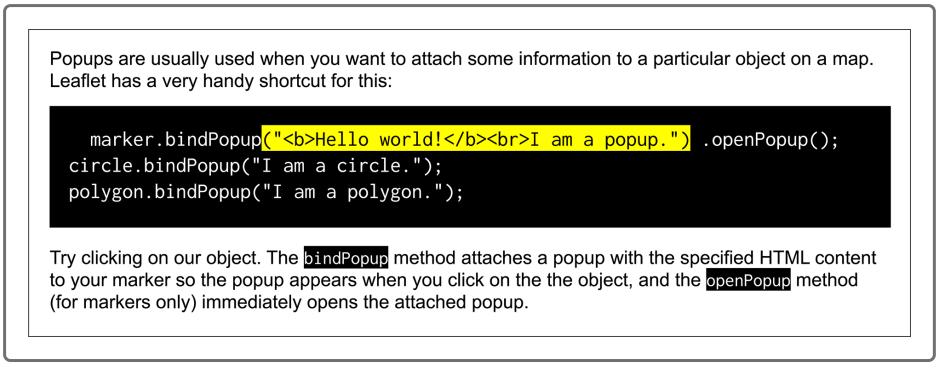
After adding the `<script>` file, the body of our `index.html` file should look like the following:

Now, when we open up the `index.html` in our browser, the map should look like it did before we created the `cities.js` file and edited the `logic.js` and `index.html` files.



Bind a Popup to the Marker

To add data from each object in the cities array, we'll use Leaflet's `bindPopup()` method on the `marker()` function. According to the guidance in the **Quick Start Guide (https://leafletjs.com/examples/quick-start/)** 's "Working with popups" section, we only need to add HTML code inside the parentheses of the `bindPopup()` method:



In the `logic.js` file, edit the `forEach` function and add the `bindPopup()` method. Inside the parentheses of the `bindPopup()` method, we'll retrieve the name of the city, state, and population.

Edit the `forEach` function to look like the following, save the `logic.js` file, and open the `index.html` file in your browser:

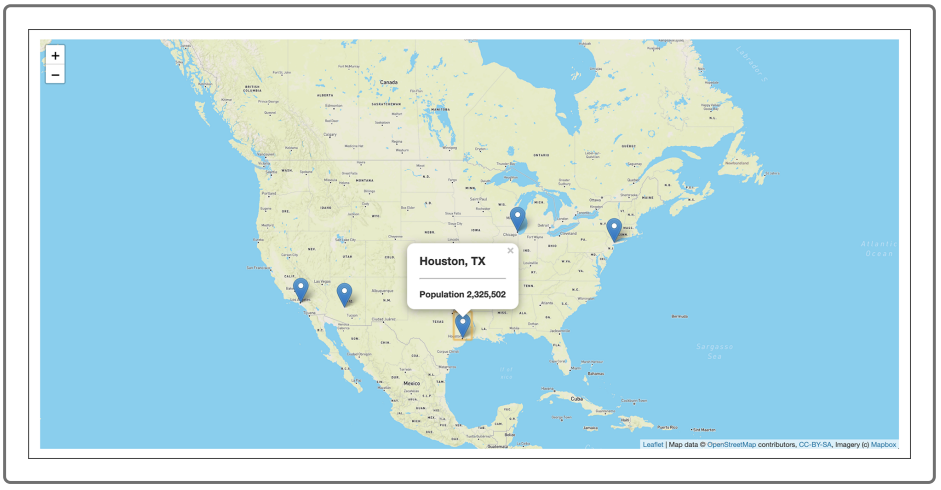
```
// Loop through the cities array and create one marker for each city.
cityData.forEach(function(city) {
    console.log(city)
    L.marker(city.location)
    .bindPopup("<h2>" + city.city + ", " + city.state + "</h2> <hr> <h3>Popu
  .addTo(map);
});
```

Now, when we click on each marker, it will show the name, state, and population of the city.

Let's format the population with a thousands separator by using the `toLocaleString()` method on the `city.population` in the `bindPopup()` method, like this:

```
// Loop through the cities array and create one marker for each city.
cityData.forEach(function(city) {
  console.log(city)
  L.marker(city.location)
  .bindPopup("<h2>" + city.city + ", " + city.state + "</h2> <hr> <h3>Population " + city.population.toLocaleString() + "</h3>")
  .addTo(map);
});
```

Now our popup markers have the population formatted with a thousands separator.



Next, change the marker for each city to a circle that has a radius equivalent to the city's population.

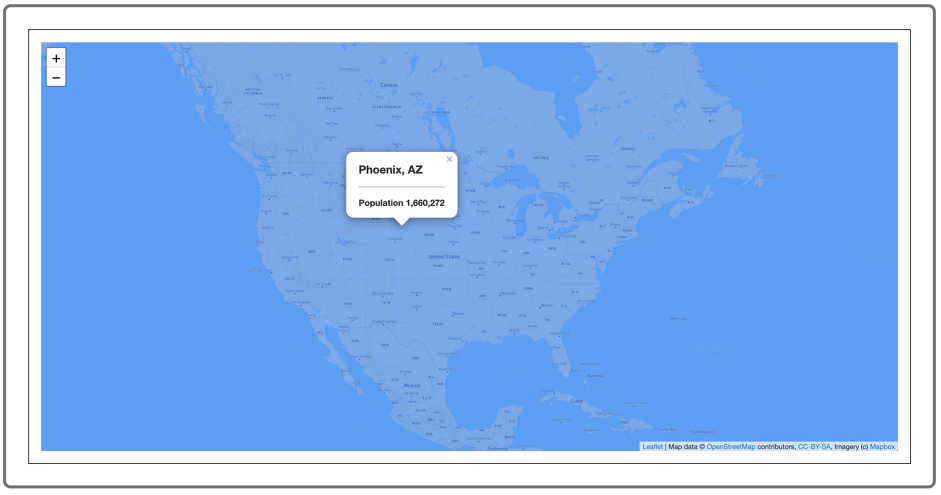
In the `logic.js` file, we'll replace the `marker()` function with the `circleMarker()` function in the `forEach()` function. Then we'll assign the "radius" key to the population by using `city.population`.

The `forEach()` function in our `logic.js` file should look like the following:

```
// Loop through the cities array and create one marker for each city.
cityData.forEach(function(city) {
  console.log(city)
  L.circleMarker(city.location, {
    radius: city.population
  })
  .bindPopup("<h2>" + city.city + ", " + city.state + "</h2> <hr> <h3>Population " + city.population.toLocaleString() + "</h3>")
  .addTo(map);
});
```

After you save the `logic.js` file and open the `index.html` file in your browser, your map will look like the following:

Well, that doesn't look like the map from before! If we click on that map, "Phoenix, AZ" and its population appear in a popup.



We know that the data is being loaded onto the map, but what is the problem?

The problem with the map is that the radii are too large and don't fit on the map. To fix this, we'll have to decrease each city's radius so the circle markers fit on the map. In the `logic.js` file, divide the `city.population` value by "100000" to look like this:

```
radius: city.population/100000
```

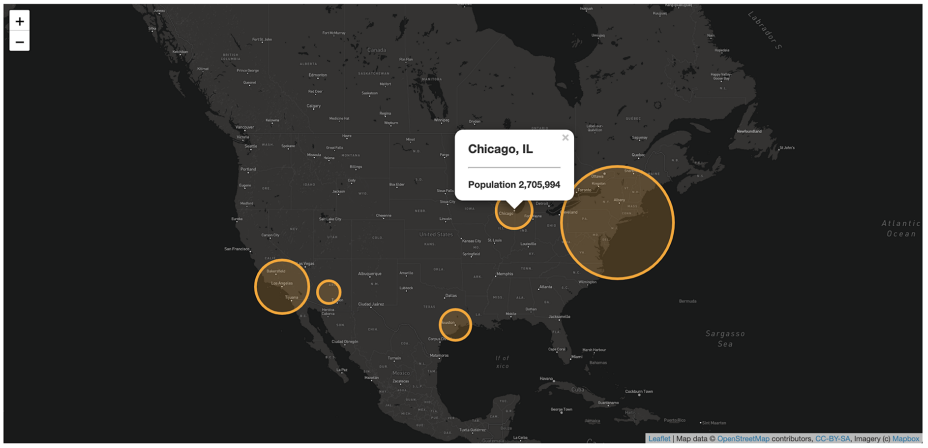
Now when we open the map in our browser, the radius for each city looks proportional to the population.

Congratulations on creating varying size circle markers with popup information!

SKILL DRILL

Edit the `logic.js` file to create an orange circle popup marker for each city, with a lineweight of 4, a radius where the population number is decreased by 200,000, that's on a dark map. When you click on the circle, the popup should display the city, state, and the population formatted with a thousands separator.

Your map should look similar to the following:



Next, Sadhana will show you how to plot lines on a map.

ADD/COMMIT/PUSH

Add, commit, and push your changes to the Mapping_Mulitple_Points branch. Don't delete the branch so that others can use it to learn how to map multiple points with popup markers.

NOTE

> For more information, see the **Leaflet documentation on the bindPopup() method (https://leafletjs.com/reference-1.6.0.html#popup)**.