13.5.5

Map GeoJSON LineStrings

Since you did so well on fetching and mapping the GeoJSON point type objects, Basil would like you to map GeoJSON LineString type objects.

Sadhana likes the work you have been pushing to the branches on the work repository. These are becoming a big hit with the other members of your team. So, Sadhana would like you to repeat the process for mapping GeoJSON LineStrings.

Create a new branch called "Mapping_GeoJSON_Linestrings," with the following folder structure. Copy the folders and files from your Mapping_GeoJSON_Points branch and add them to the Mapping_GeoJSON_Linestrings folder:

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

Next, download the `torontoRoutes.json` file and put it on the Mapping_Earthquakes repository.

Download torontoRoutes.json (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_13/torontoRoutes.json)

Let's take a look at the `torontoRoutes.json` file, which contains a majority of the nonstop airline routes from Toronto, Canada.

```
{
    type: "FeatureCollection",
  - features: [
    - {
          type: "Feature",
        - properties: {
              airline: "9W",
              airline_id: "3000",
              src: "YYZ",
              src_id: "193",
              dst: "BRU",
              dst_id: "302",
              codeshare: "",
              stops: "0",
              equipment: "332 333"
          },
        - geometry: {
              type: "LineString",
            - coordinates: [
                - [
                      -79.63059997559999,
                      43.6772003174
                  ],
                - [
                      4.48443984985,
                      50.901401519800004
                  ]
              ]
          }
      },
```


Looking at the first object in the `torontoRoutes.json` file, we can see that there is a properties and geometry object. In the geometry object, we can see the type is `LineString` and the `coordinates` are an array of point coordinates, like when we mapped the airline route from LAX to SEA earlier in this module.

Now, map the nonstop routes from Toronto on two map styles: light and dark. First, we'll need to create both map styles.

1. For the dark map: Use the code from the Mapping_GeoJSON_Points `logic.js` file to get the API for the dark map.

Retake

2. For the light map:
    - In the streets variable in the `logic.js` file (from the Mapping_GeoJSON_Points), replace `streets-v11` with `light-v10` for the `streets` map.
    - Change the variable for the `streets` tile layer to `light`.

Retake

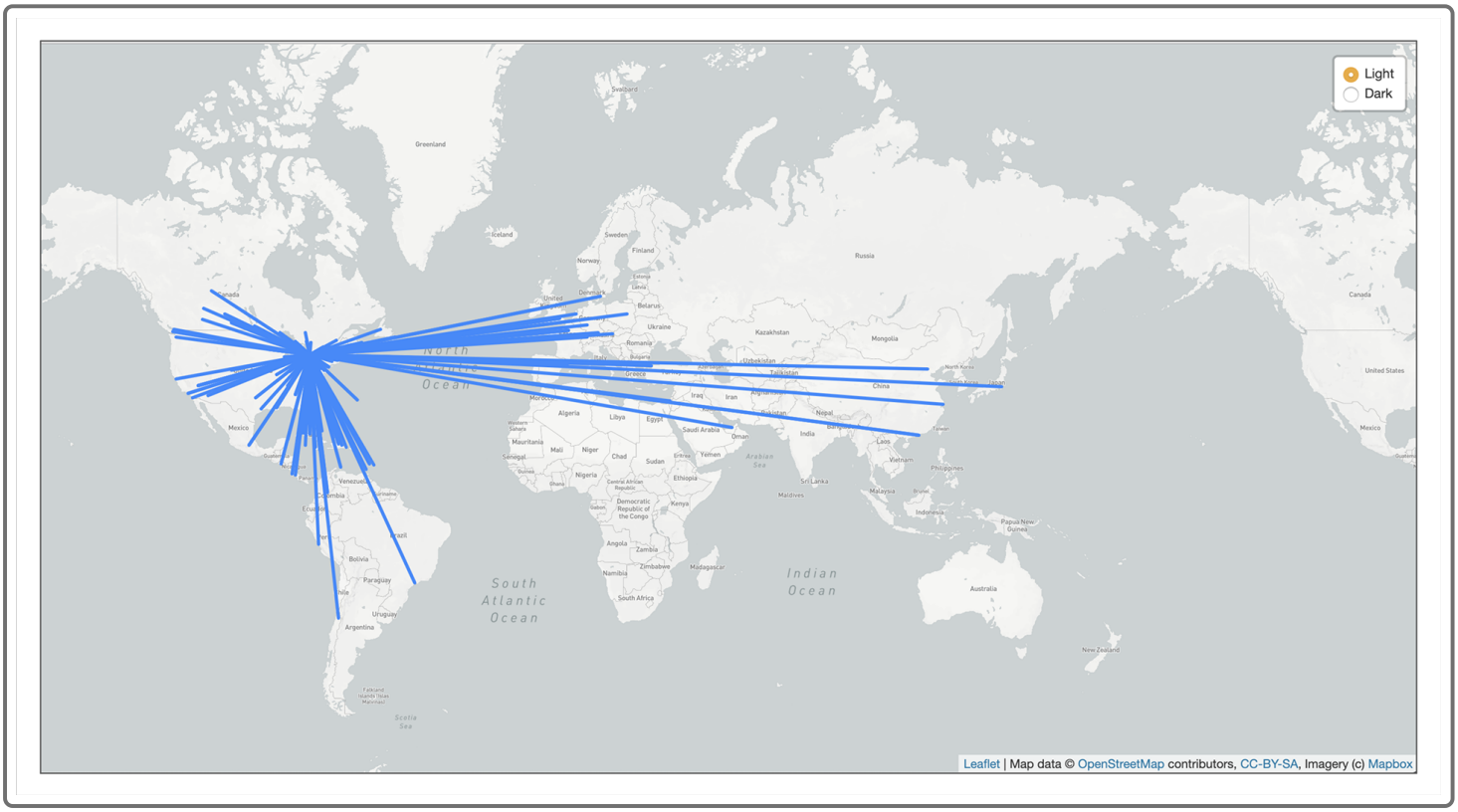
3. Change the center of the map to Toronto with a zoom level of 2. Toronto's coordinates are listed first in the array: `[44.0, -80.0]`.
4. Create a `torontoData` variable in the `logic.js` file and assign it to the URL for the `torontoRoutes.json` file on GitHub.

```
// Accessing the Toronto airline routes GeoJSON URL.
let torontoData = "https://raw.githubusercontent.com/<GitHub_name>/Mapping_Earthquakes/main/torontoRoutes.js
```

5. Edit the `d3.json()` method to look like the following:

```
// Grabbing our GeoJSON data.
d3.json(torontoData).then(function(data) {
    console.log(data);
  // Creating a GeoJSON layer with the retrieved data.
  L.geoJSON(data).addTo(map);
});
```

When you open `index.html` in your browser, your map should look like the following:



Great job on mapping the nonstop routes from Toronto!

SKILL DRILL

Edit the `L.geoJSON()` layer so that it displays the following:

1. The default map layer as night navigation with day navigation as another option.
2. The airline routes are in light yellow with a `weight` of 2.
3. Each airline route has a popup marker that shows the airline code and destination.

   Your map should look like the following:

The code in your `logic.js` file specifies a dark map showing nonstop flight routes, styled in light yellow with a `weight` of 2. Your file should look like the following:

```
// Grabbing our GeoJSON data.
d3.json(torontoData).then(function(data) {
  console.log(data);
// Creating a GeoJSON layer with the retrieved data.
L.geoJson(data, {
  color: "#ffffa1",
  weight: 2,
  onEachFeature: function(feature, layer) {
    layer.bindPopup("<h3> Airline: " + feature.properties.airline + "</h3> <hr><h3> Destination: "
    + feature.properties.dst + "</h3>");
  }
})
.addTo(map);
});
```

To make this code easier to read, now we'll create an object with the style parameters for the lines and assign it to a variable, `myStyle`. Add the following code before `d3.json()`.

```
// Create a style for the lines.
let myStyle = {
    color: "#ffffa1",
    weight: 2
}
```

Then, in the `L.geoJSON()` layer, the `style` key will be assigned to the `myStyle` object as shown:

```
// Grabbing our GeoJSON data.
d3.json(torontoData).then(function(data) {
  console.log(data);
// Creating a GeoJSON layer with the retrieved data.
L.geoJson(data, {
  style: myStyle,
  onEachFeature: function(feature, layer) {
    layer.bindPopup("<h3> Airline: " + feature.properties.airline + "</h3> <hr><h3> Destination: "
    + feature.properties.dst + "</h3>");
  }
})
.addTo(map);
});
```

Next, Sadhana will show you how to map polygon type geometry from a JSON file.

ADD/COMMIT/PUSH

Add, commit, and push your changes to your Mapping_GeoJSON_LineStrings branch. Don't delete the branch so that others can use it to learn how to map GeoJSON LineStrings.