11.5.3 Add Filters

The code we helped create will add every object in ou `data.js` file to the table. Bundled into one tidy package, every sighting will be available for Dana (and her readers) to view! There are a lot of objects, though, which will make the table huge! There will be hundreds of rows of sightings in the table, which is entirely too much for one person to reasonably look through and study. Therefore, the next step is to add the ability to filter the data. We'll be using D3.js to help Dana with this part.

Data-Driven Documents (D3 for short) is a JavaScript library that adds interactive functionality, such as when users click a button to filter a table. It works by "listening" for events, such as a button click, then reacts according to the code we've created.

Dana thinks that she would like to filter by date, so she'll add code to create a date filter.

We'll need to add a second function to our code that will focus on filtering the table we just built. We'll use a popular library, D3.js, to equip our website to "listen" for events, such as a user clicking a button.

In our code, we're going to use D3 to handle an action from a user, such as a button click. This means that we'll add an actual button to our HTML page to filter the table. When the button is clicked, D3 will detect the click and react accordingly. Building out user-driven data visualizations is an essential part of the data visualization job; it can be tricky at first, but oh-so-satisfying when it works! Let's get started.

Return to VS Code and our `app.js` file and start a new function. We'll name this one "handleClick" because it will be handling what to do after an input is given, such as filtering the table by date.

Let's go ahead and set up the function.

Since we're adding a date function, we need to create a couple of variables to hold our date data, both filtered and unfiltered.

```
function handleClick() {
    let date = d3.select("#datetime").property("value");
```

So what's going on in this code? D3 looks a little different from what we're used to seeing, but that's because it's closely linked to HTML.

The `.select()` function is a very common one used in D3. It will select the very first element that matches our selector string: "#datetime". The selector string is the item we're telling D3.js to look for.

With `d3.select("#datetime")`, for example, we're telling D3 to look for the #datetime id in the HTML tags. We haven't created our HTML yet, but we know that the date value will be nested within tags that have an id of "datetime."

By chaining `.property("value");` to the `d3.select` function, we're telling D3 not only to look for where our date values are stored on the webpage, but to actually grab that information and hold it in the "date" variable.

Now we need to set a default filter and save it to a new variable. Our default filter will actually be the original table data because we want users to refine their search on their own terms. Let's add the new variable on the next line.

```
let filteredData = tableData;
```

Here's a variable we haven't seen in a while: tableData. This is the original data as imported from our `data.js` file. By setting the filteredData variable to our raw data, we're basically using it as a blank slate. The function we're working on right now will be run each time the filter button is clicked on the website. If no date has been entered as a filter, then all of the data will be returned instead.

The next step is to check for a date filter using an `if` statement.