Leaflet book

Samuel Gachuhi Ngugi

2023-06-21

Contents

About

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports; for example, a math equation $a^2 + b^2 = c^2$.

Usage

Each **bookdown** chapter is an .Rmd file, and each .Rmd file can contain one (and only one) chapter. A chapter *must* start with a first-level heading: `# A good chapter`, and can contain one (and only one) first-level heading.

Use second-level and higher headings within chapters like: `## A short section` or `### An even shorter section`.

The `index.Rmd` file is required, and is also your first book chapter. It will be the homepage when you render the book.

Render book

You can render the HTML version of this example book without changing anything:

1. Find the **Build** pane in the RStudio IDE, and
2. Click on **Build Book**, then select your output format, or select "All formats" if you'd like to use multiple formats from the same book source files.

Or build the book from the R console:

```
bookdown::render_book()
```

To render this example to PDF as a `bookdown::pdf_book`, you'll need to install XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.org/tinytex/.

Preview book

As you work, you may start a local server to live preview this HTML book. This preview will update as you edit the book when you save individual .Rmd files. You can start the server in a work session by using the RStudio add-in "Preview book", or from the R console:

```
bookdown::serve_book()
```

Chapter 1

Introduction

1.1 What is Leaflet?

Something to do with leaves? Of course not.Leaflet, when bare scrapped to its most basic definition, is simply an open source JavaScript library for interactive maps. It was developed in 2011 by Volodymyr Agafonkin, a Ukrainain with a mathematical background.

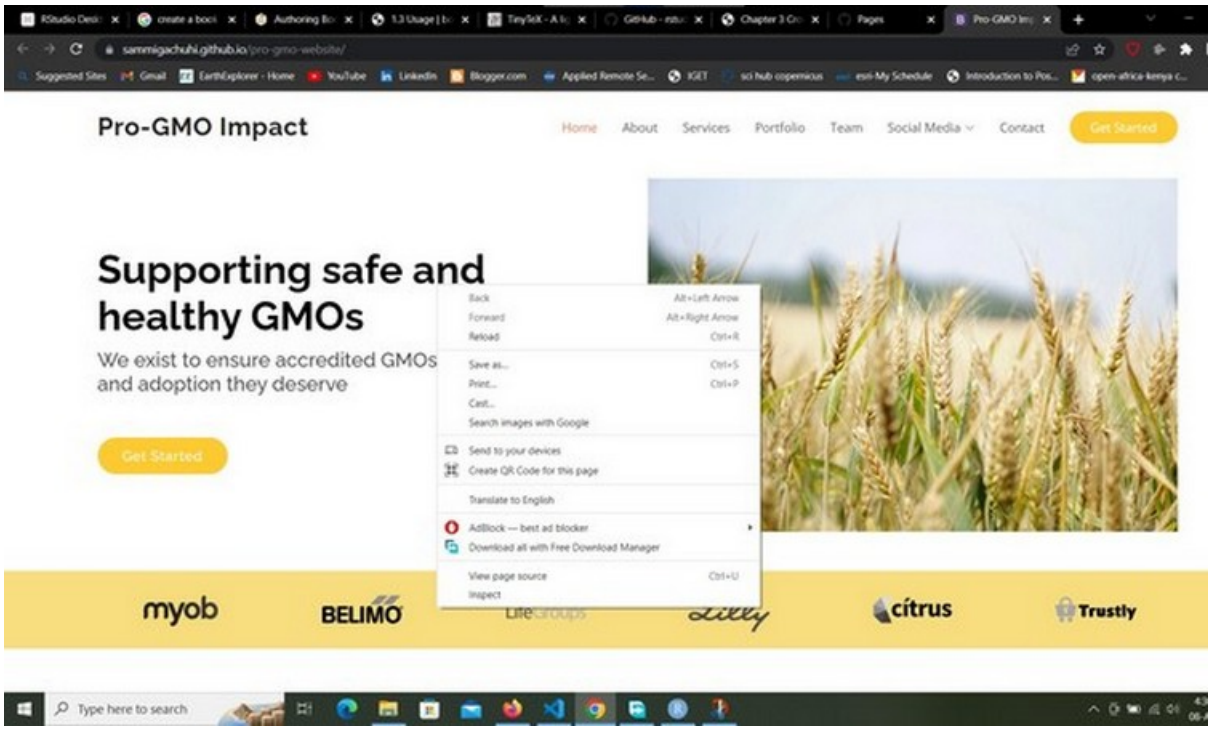
1.2 How does it work?

Leaflet can work if every line of code is inside a `html` document, so long as the code appears under the `<script>` tag. However, for a neat work, especially working with complex maps, it is recommended you separate the `html` file from its other components of `main.js` and `style.css` files.

"HTML we know, but what are `main.js` and `style.css` files?", you may ask.

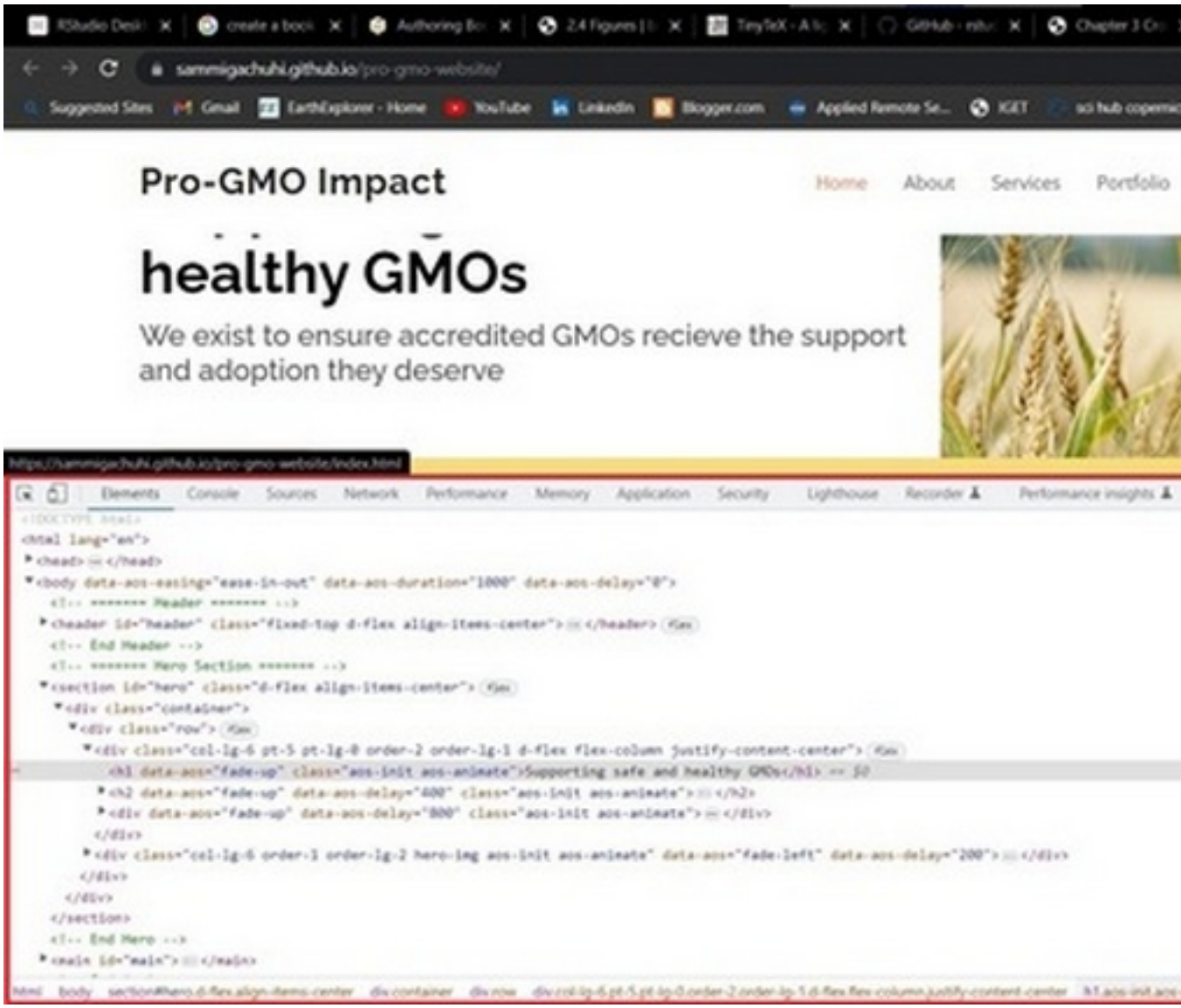
Well, beginning with `html`, which stands for **Hypertext Markup Language**, it is the language that is used in creating webpages. It is actually the standard of making static webmaps. I am yet to come across any webpage that is made up of everything apart from HTML. If you want to have a view of what HTML looks like, just right click any webpage and click *Inspect* in Google Chrome and Firefox. A toolbar will appear at the bottom or side of the webpage, depending on your settings.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/inspect.jpg"))
```



Scroll over to the **Element** tab and you will have something that looks like this:

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/elements.jpg"))
```



The part encircled in red is the `html` that makes up the webpage for the ProGMO website in this case.

So, I am a GIS specialist, I want to learn how to make a html website so as to use leaflet and its functionalities. Whereas this document does not provide an indepth view of all the ins and outs of a html document, html websites are made up of elements known as `tags`. Tags, normally indicated by angle brackets (`<>`) are what introduce any form of content into a webpage, be it a paragraph (`<p>`), an image (`<img>`), video (`<video>`) and even an entire section (`<div>`, `<section>`, `<article>`). With this basic introduction, let's create a basic html

page.

To create a html element along with many other programming files, such as `.js` and `.css` which we shall see later, we use a text editor. A good example of a text editor is VS code or Pycharm. Check their websites on their installation methods for your personal computer. For creating html and working with `.js` documents later, we shall use VS Code unless otherwise stated.

Here is a basic html webpage.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>A basic html webpage</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="style.css">
    </head>
    <body>
        <div id="division-1">
            <p>Hello, World!</p>
        </div>
        <script src="main.js">

        </script>

    </body>
</html>
```

Let's go through the above tags one by one.

1. `<!DOCTYPE html>` - It is an "information" to the browser about what document type to expect.
2. `<html lang="en">` - It is the container for all other HTML elements (except for the `<!DOCTYPE>` tag). The `lang` attribute is used to assist web engines know which language the website uses.
3. `<head>` - It is not displayed on the webpage as other tags, but contains the metadata of the webpage.
4. `<title>` - Can you guess? You had it right. Defines the title of the document. In our case, if you open the webpage assuming you created it in VS Code, the webpage shall be titled *A basic html webpage* at the tab of your web-browser.
5. `<meta charset="utf-8">` - This is one of the metadata hosted by the `<head>` tag. We had mentioned earlier that the `<head>` contains the metadata of the webpage. Now here we would like to add that the `<meta>` tag

found *within* the `<head>` is what *defines* the metadata. You can think of it as **README** text file that comes with any software you download. The `<meta>` tag in our case defines the encoding of our HTML5 document with the attribute `charset="utf-8"`. Don't think about this too much. HTML5 documents have `utf-8` as their encoding. You can try to look up what encoding is but it's not useful for this tutorial!

6. `<link>` - Defines the relationship between a document and an external resource. It has various attributes but `rel` and `href` have been used. The former specifies the relationship between the current document and the linked document/resource. The `rel` here references the `styles.css` file as the style sheet for our html. That is, the styles for our html are found in the `styles.css` file. `href` on the other hand points the html document to the path of the stylesheet –the `styles.css` file.

7. `<body>` - This is the crux of your webpage. If nothing is within the `<body>` tags, your webpage will be as empty as a blank sheet of paper. This tag is the home for all the other contents of the webpage such as headings, paragraphs, images, tables etc.

8. `<div>` - This is a special element that lets you group similar sets of content together on a web page. You can use it as a generic container for associating similar content. In the above html script, we have included an `<id>` attribute that is in other words, a unique identifier for this section of the webpage. `<id>s` are useful if you want to customize the appearance of a certain part of the webpage. `<class>`es behave in a similar way, but the difference between `<id>` and `<class>` is that `<id>` has to be unique, while `<class>`es can be used more than once.

9.`<script>` - It is used to embed executable code or data. In most cases it refers to JavaScript, which enhances interactivity.

If you may have noticed above, most HTML tags end with `</name-of-tag>`. With a few exceptions such as `<img>`, almost all HTML tags end this way.

1.3 JavaScript

JavaScript, shortened to `.js` is the language of the web. It introduces interactivity to HTML files. Without it our HTML files would just remain static. Have you ever clicked a link or a shiny button on a website and some visual or menu popped up? JavaScript was the engine behind all that. Think of `.js` as the life of the party while HTML is just the setting. Without `.js` creating webmaps would not be possible since adding JavaScript code to a html file using `<script>` is what makes the map appear on any website!

1.4 CSS files

CSS stands for *Cascading Style Sheet*. The CSS defines how your HTML is to appear, such as color and size of text, background color of the HTML as well as the structure of your HTML page.

CSS is quite a huge field despite being simple. However, the html elements of a webpage are accompanied by a curly bracket containing the specified properties and values.

- Properties: These are human-readable identifiers that indicate which stylistic features you want to modify. For example, font-size, width, background-color.
- Values: Each property is assigned a value. This value indicates how to style the property.

Using the example of our ProGMO website, this is how we would specify the font and color of the `<body>` element of our webpage. In some cases, the property values in CSS elements can be more than one, as in `font-family` below.

```
body {
  font-family: "Open Sans", sans-serif;
  color: #444444;
}
```

The `body` in the CSS file is known as the selector. Selectors in CSS are what tags are in HTML files. However, selectors can be more specific, such as specifying the exact `<div>` that should be displayed in a particular way. Using our html file example, if there were other `<div>`s apart from the `<div id="division-1">` above, we would specify our first one in a CSS document like so:

```
#division-1 {
  font-family: "Open Sans", sans-serif;
  color: #343a40;
}
```

We would specify other property values for our `<div>`s by their names following a `#` like so. Suppose there was a `<div id="division-2">` somewhere in the HTML we could define some properties specific to it in the manner below:

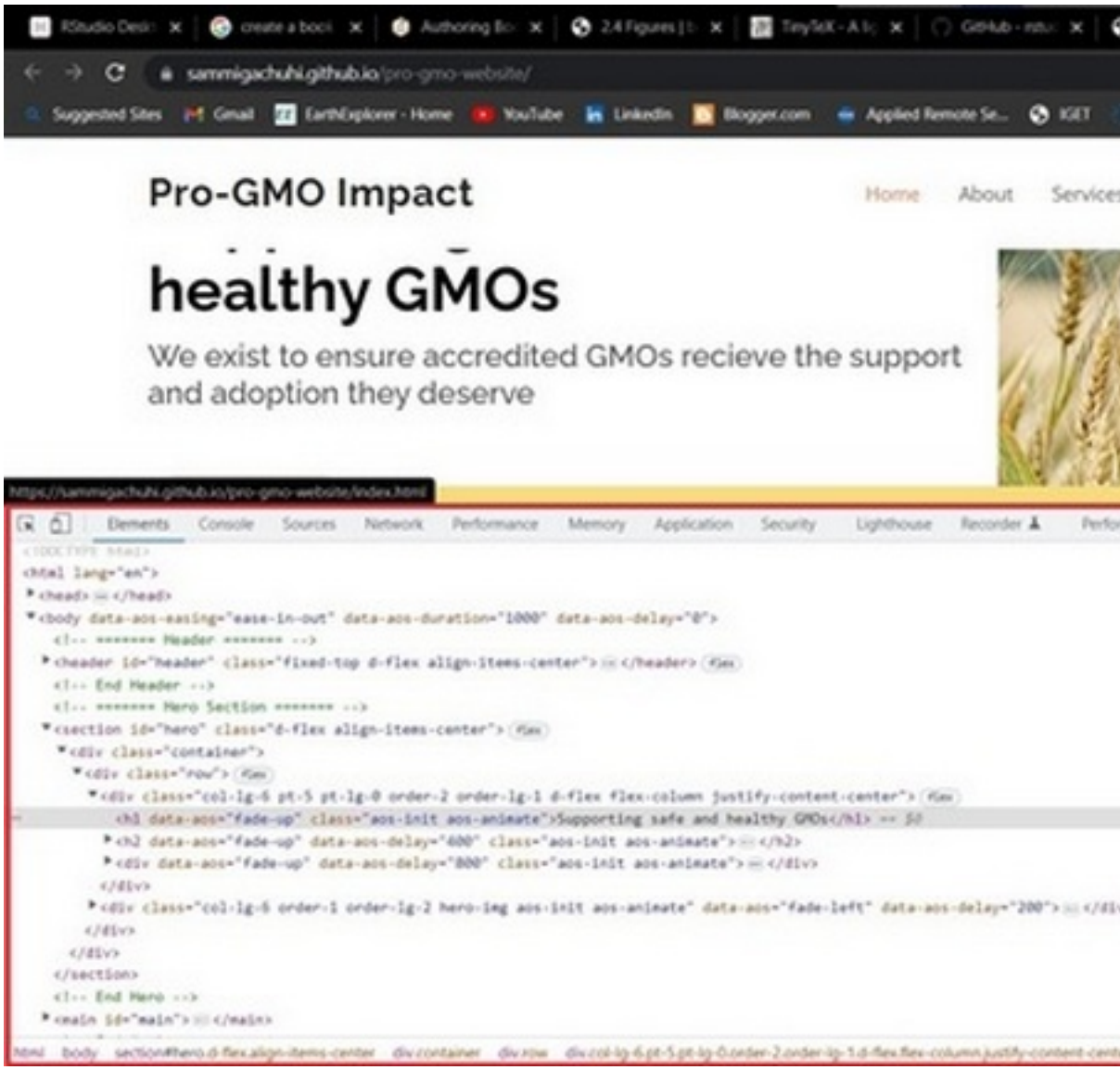
```
#division-2 {
  font-family: helvetica;
  color: #000000;
}
```

For `<classes>` and they can be several, we select each particular class using the convention:

```
.class_name {
property: value
prperty2: value2}
```

You can view the style of a particular HTML element using the styles tab found in the inspect console. It is shown in yellow bounds for a chrome webpage. Firefox should have a similar one.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/elements2.jpg"))
```



The MDN website provides a lot of information on HTML and CSS.

1.5 Summary

This chapter was an introduction to Hyper Text Markup Language (HTML), JavaScript and Cascading Style Sheets (CSS) languages. You learnt the following:

- You can work with leaflet in either a html or JavaScript file. In html, the JavaScript code must appear under the `<script>` tag.
- HTML files are made up of elements called tags. Tags are features that introduce any form of content into a webpage.
- JavaScript is the main language of the web. It is the language responsible for the interactivity in most websites.
- CSS stands for Cascading Style Sheet (CSS). CSS defines how your HTML is to appear, such as color and size of text, background color and even the structure of your HTML page.

Chapter 2

First Leaflet Map

2.1 Setting the superstructure

We had earlier mentioned that Javascript, otherwise shortened to JS is the life of the party when it comes to webpages. In other words, it makes your web pages interactive. It's like the additional component that makes your HTML pages move from static to responsive.

Creating a leaflet map is not like creating any other HTML web page. You have to set up the leaflet essentials in your HTML page first. To begin with, create a new html document called `map.html`. This will be the html document that will act as the structure which will house our webpage to be created using JS. Using VS Code, create `map.html` and paste, or preferably, type the following code.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Leaflet Maps</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="styles.css">
        <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css"
     integrity="sha256-kLaT2GOSpHechhsozzB+flnD+zUyjE2LlfWPgU04xyI="
     crossorigin=""/>
        <script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js"
        integrity="sha256-WBkoXOwTeyKclOHuWtc+i2uENFpDZ9YPdf5Hf+D7ewM="
        crossorigin=""></script>
    </head>
    <body>
        <div id="myMap">
            <script src="main.js">
```

```
            </script>
        </div>

    </body>
</html>
```

You may be wondering why we have two `<link>` tags.

"Won't they confuse the webpage or something?" You may wonder.

Same thing for the two `<script>` tags, one at the head and the other at the `<body>` tag. The answers is 'No'. Once a html script is loaded in our browser, assuming its the `map.html` we've created, the browser reads it from top to bottom. In our html script, the browser will apply the styles defined in `styles.css` to the html elements. To make matters clearer, the following script is what is contained in the `styles.css`:

```
#myMap {
    height: 600px;
}
```

Therefore, the browser will display everything contained in the `<div>` inside the `<body>` tag at a height of 500px. This is because the `<div>` contains the ID `myMap` which has been referenced in the local stylesheet as `#myMap`. Don't fret about what how things outside the `<div>` will be displayed since for our webmap making purposes, hardly will we code anything outside the `<div>` tag.

Now to the two `<script>` tags. One refers to the online JavaScript library. The `src` attribute is in fact linking to a webpage as you can see from the protocol https. The second, housed under the `<div>` tag, references to our local JavaScript file which shall contain all the code to transform our html page to a webmap ninja –lines, polygons and other cool stuff.

2.2 Beautifying the house

Think of the html document as the superstructure, like a huge multistorey building just finished. Though the structure has the best architectural design, it just looks all grey with no life unless we call some interior and exterior designers to add some color. That's what `main.js` file, pointed to by the `<script>` tag in the html document will precisely do.

Open your VS Code, and assuming you had already created `main.js` already, (if not, create one now), insert the following code into the `.js` file.

```
var map = L.map('myMap').setView([-0.0884105,34.7299038], 13);
```

Take a pause.

Breath in, breath out.

You are just about to learn something very important here. In fact, it is the crux of what makes Leaflet work. Your future of understanding leaflet hinges on this little code.

The `L.map()` class we just used is what initializes the leaflet map. Everything within the `<div>` is displayed thanks to this class function. It is referred to as a factory function because it uses the method `map` to return an object.

The `setView` method *sets the view of the map (geographical center and zoom) with the given animation options.* It's properties are Latitude-Longitude, zoom number and other options. If you would like to view the humongous leaflet reference, get it here.

In our case we just inserted the Lat-Long and zoom number.

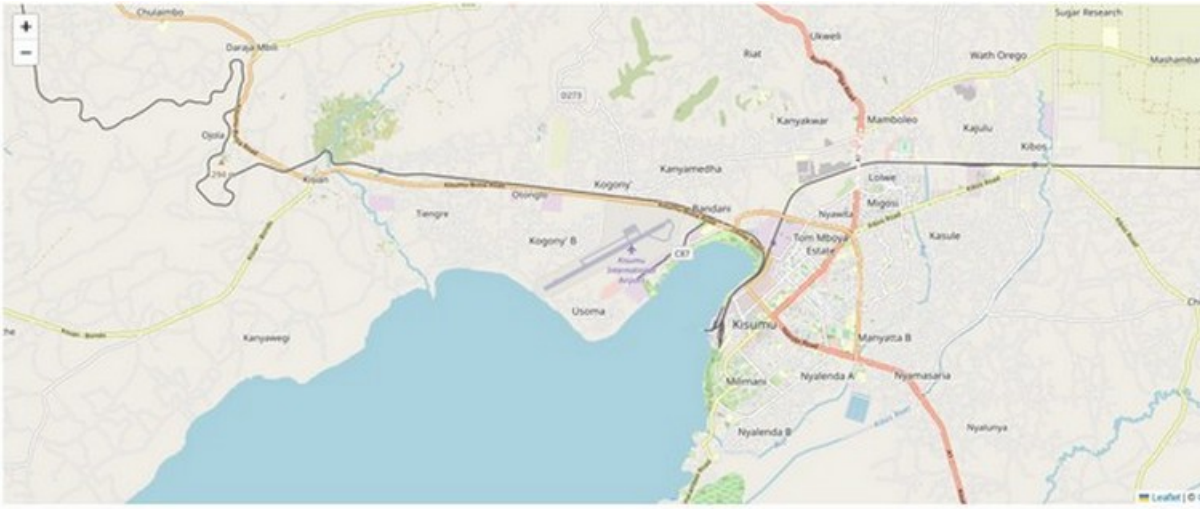
Try loading your `map.html`. All you see is a grey canvas with zoom options. This is because we haven't added a tilelayer yet. A tileLayer is a set of web-accessible tiles that reside on a server. A tile is an individual image or vector file from a server which are collectively joined together to form the webmap. If you've zoomed into a webmap, say Google Maps and noticed boxes appearing as you zoomed in or out, those are *tiles*.

Let's load an example of a common tile layer–the Open StreetMap–into Leaflet.

```
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);
```

Reload your html page again. What do you see?

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/kisumu-leaflet.jpg"))
```



What `L.tileLayer` has done is retrieve the web tiles from the url source provided, and within the dictionary that follows the url, zoom level (`maxZoom`) and map attribution (`attribution`) have been provided. When working with leaflet, the dictionary, indicated by the curly braces `{}` houses most of the additional class options other than the key one(s). In this case we used the additional options of `maxZoom` and `attribution`. Finally, the method `addTo` adds the layer to the given map or layer group. Here, our webtile is added to the `var map` which only contains the `setView` properties.

A very influential person said Kisumu located in Kenya is a town with great potential. How about dispalying it to the whole world to realise it!

2.3 Summary

In this chapter, we created our first leaflet map. Here are a couple of things that you have learnt at the first step of this web mapping journey.

- Browsers read code scripts from top to bottom, much like skimming down a page.
- The styles defined in the CSS style sheet, the `styles.css` in this case, will apply to the HTML elements in the `map.html` file.
- We use the `src` attribute to link to a webpage.
- If HTML is the magnificent building, JavaScript acts like a good interior and exterior designer.

- The `L.map()` class is what initializes the leaflet map.

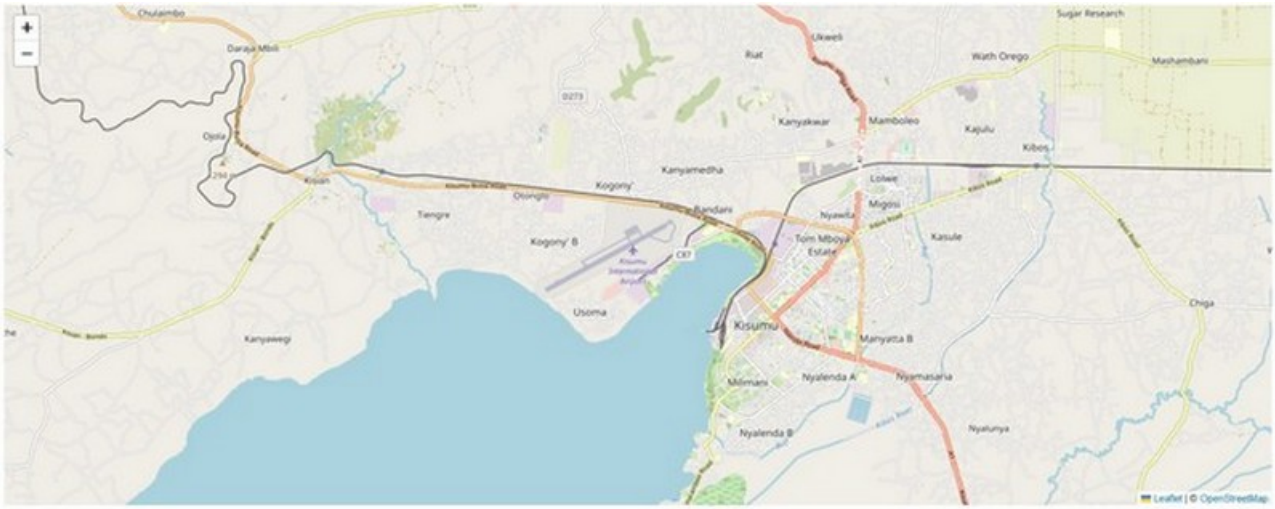
Chapter 3

Add ons

Like in the ultimate finale of a series where the episode begins with "Previously on…", this chapter shall be a continuation of Chapter 2.

So we have a plain looking webmap like the one shown below.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/kisumu-leaflet.jpg"))
```



However, despite being a cool looking webmap, it offers no sort of information to the viewer except that it is an interactive map interface. In order to pass some information, such as showing the location of Kisumu and *inter alia*, markers are one way of displaying content.

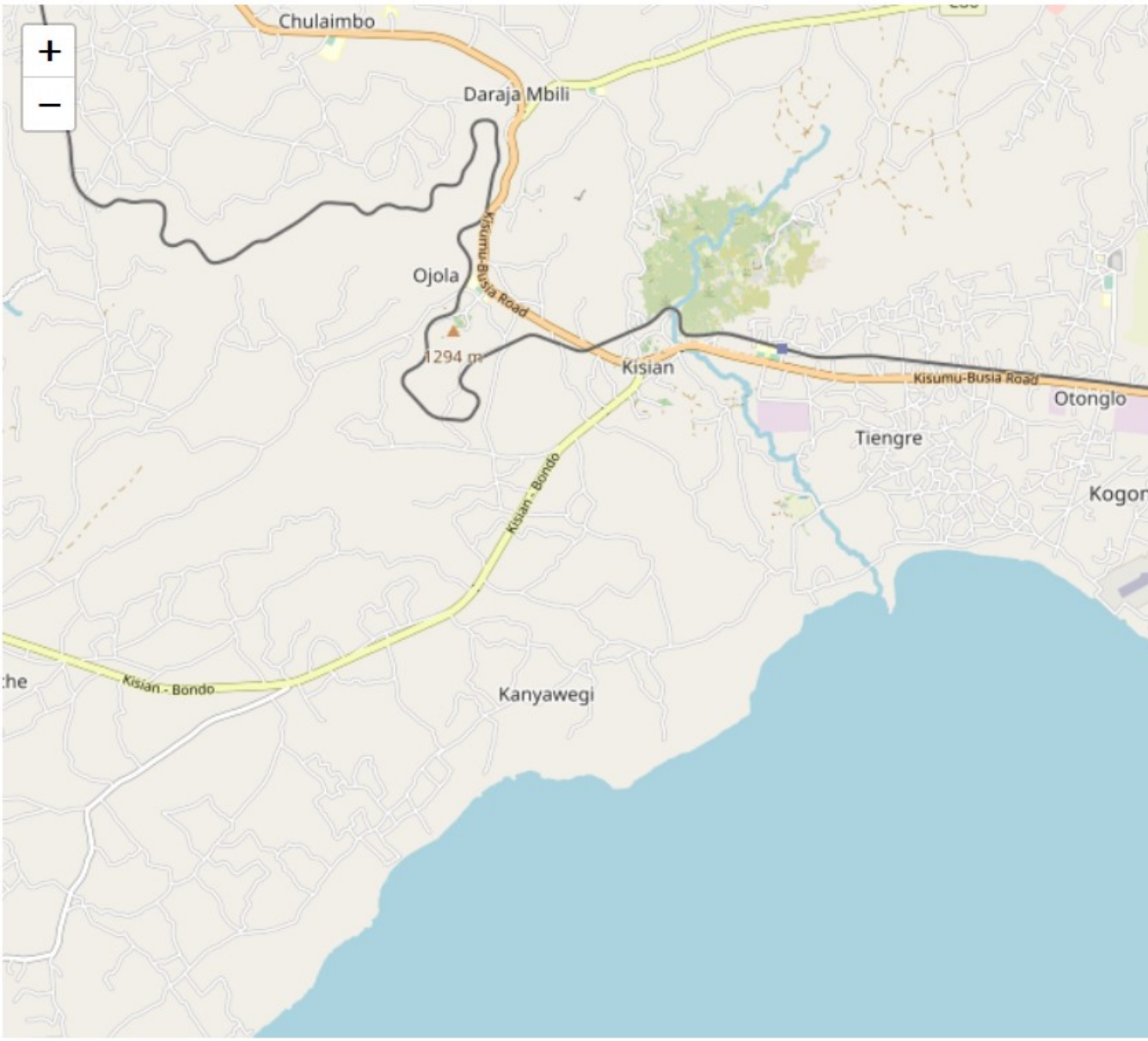
In the last chapter we had last left our `main.js` file looking like in the image above. Ensure yours is also the same, but we encourage you to explore with other leaflet layers available from here.

3.1 A marker

Many people could possibly hardly know where Kisumu, is, so lets indicate its location with a simple pin marker. To be more specific, let's pinpoint Kisumu International Airport.

```
// Location of Kisumu International Airport
var marker = L.marker([-0.0819301, 34.7260167]).addTo(map);
```

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/kisumu-international-airport
```



As a simple exercise, can you try creating a marker for your home location, not forgetting to change the `setView` method you had initially started with in this map creating journey?

Alright, we have a marker. But what's so special about it as a lone pin in the middle of somewhere? Let's try to make this marker have some information, otherwise called attributes in GIS. Let's say the attributes we want to add are

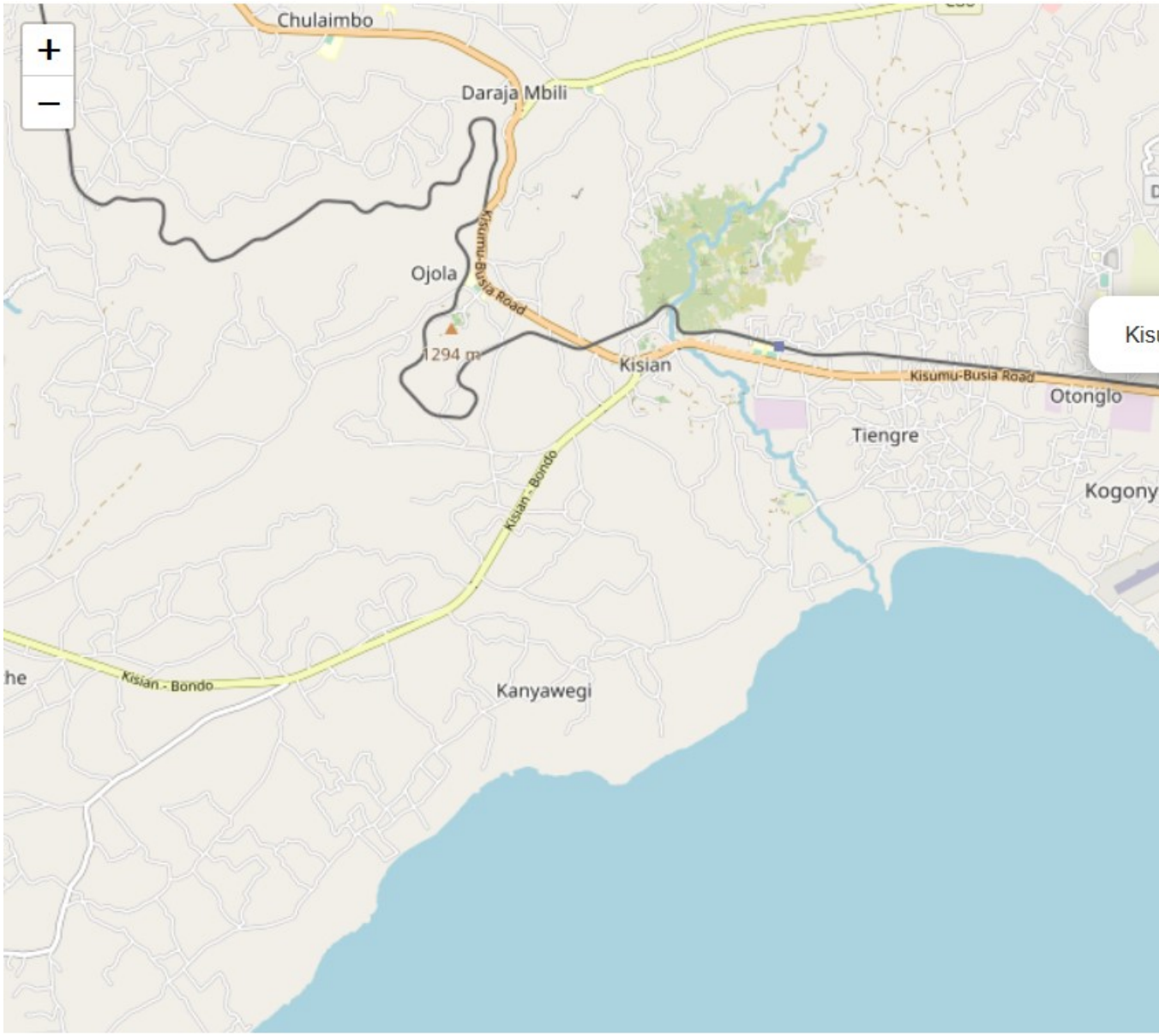
the name of the airport and other auxillary data.

3.2 A marker with a popup

To create popups, leaflet provides the `bindPopup` method. It is especially easy if you already have a marker variable in place, as in our case.

```
// Create popup of Kisumu international Airport
marker.bindPopup("Kisumu International Airport.").openPopup();
```

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/kisumu-airport-popup.jpg"))
```



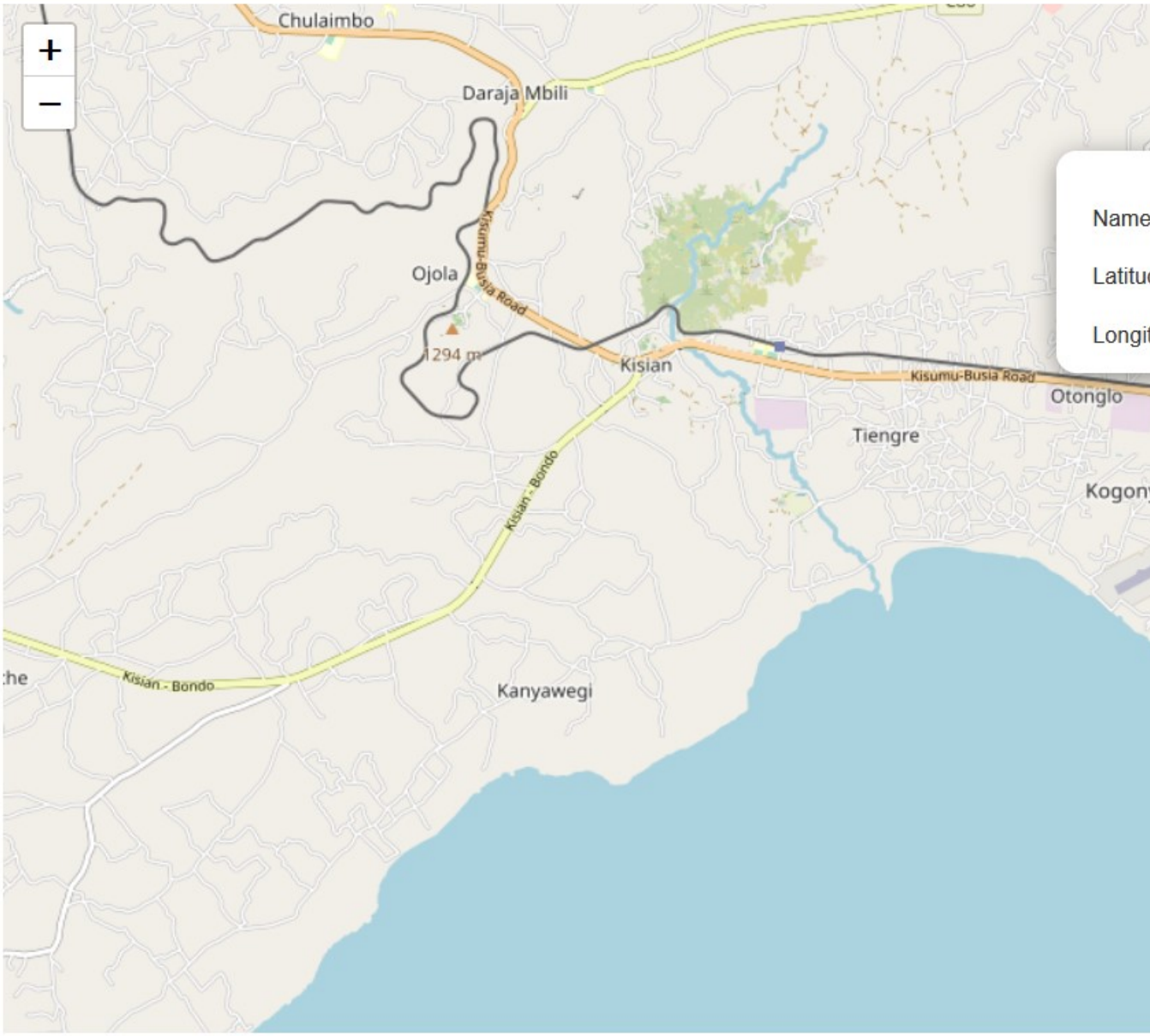
What has just happened is that `bindPopup` binds the popup content– "Kisumu International Airport" to the marker. The following `openPopup` method chained to the method *opens* the popup at that specified latitude longitude. If you remove, or comment `//` out the `popUp` method, you will have to click the marker to see the popup content. Try it out.

Markers can also work with HTML elements, such as when you want to display

additional metadata, say the owner of the place, size of land et cetera. In the below case, we have added the lat-lon coordinates of Kisumu airport location. I would advise not to include lengthy information in an HTML marker element.

```
// With html content
marker.bindPopup("<br>Name: Kisumu International Airport</br><br>Latitude: -0.0819301</
```

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/marker-html.jpg"))
```


Chulaimbo
Daraja Mbili
Ojola
Kisumu-Busia Road
1294 m
Kisian
Kisumu-Busia Road
Otonglo
Tiengre
Kisian - Bondo
Kisian - Bondo
Kanyawegi
Name
Latitu
Longit

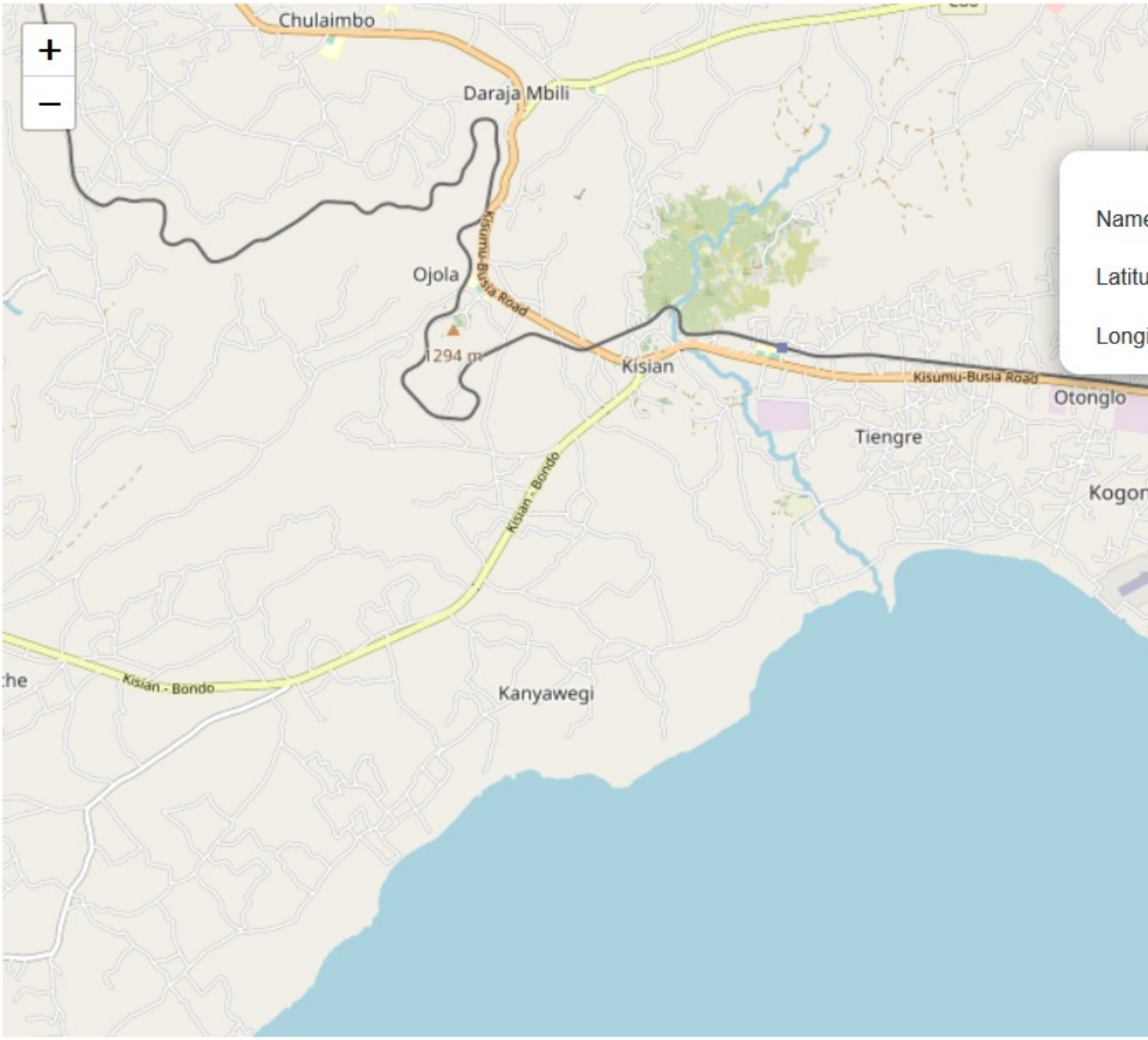
3.3 Different markers and popups

So far you have seen pin markers, but there are also other kinds of markers, such as circles and rectangles. Unlike the pin markers we have been experimenting with, these other markers require additional options, such as radius for circle and lat-long coordinates for rectangles. Let's have a go with each type.

Starting with a circle, let's start by inserting a circle to show the location of Kisumu Museum.

```
// Circle over Kisumu Museum
var circle = L.circle([-0.107637, 34.7435975]).setRadius(2000).addTo(map);
```

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/kisumu-museum-circle.jpg"))
```



The below code will also create a slightly similar circle marker, the only difference is that in the preceding one we didn't insert `{options}` and we set radius using the `setRadius` method. In the second one below, we have been very specific in what we want –our specifications going into the curly brackets {} before eventually adding the circle marker to our map. Brackets in JavaScript denote dictionaries. Dictionaries in JavaScript and even in python are used to denote

key-value pairs.

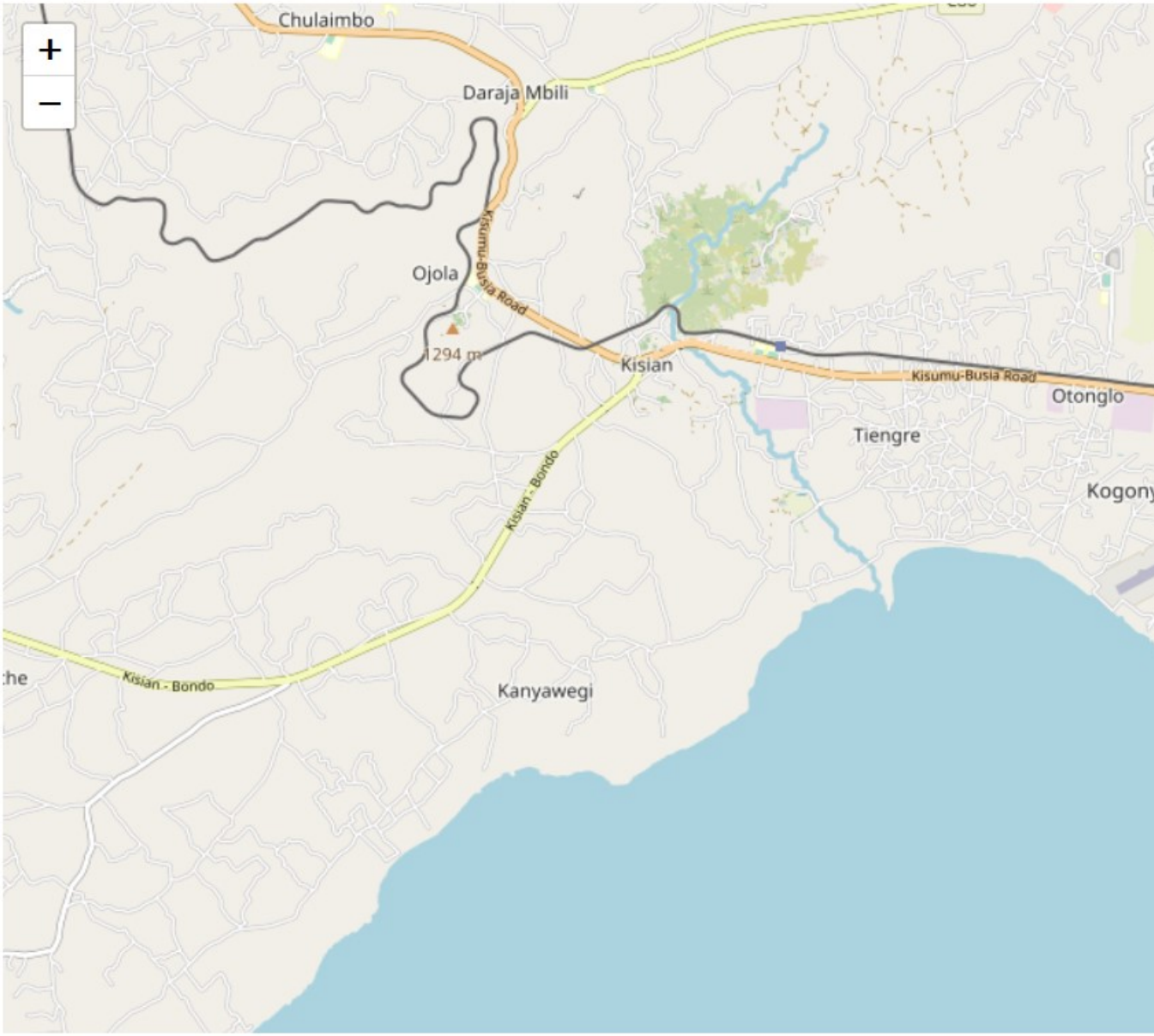
```
var circle = L.circle([-0.107637, 34.7435975], {
    color: 'blue',
    fillColor: 'blue',
    fillOpacity: .5,
    radius: 2000
}).addTo(map);
```

As we had mentioned earlier, other marker elements such as circles and rectangles can have popups attached to them. Ready for something cool? We will attach a popup into our circle. Not just any other ordinary hard coded popup but one which relies on other Leaflet JavaScript methods to generate an output. In our case, we want a pop up that shows the radius of our circle, without us typing it out into the code.

```
// Circle marker pop up for Kisumu Museum
var getRadius = circle.getRadius();
circle.bindPopup("Radius is: " + getRadius.toString() + " metres");
```

In our above code, `getRadius` gets the radius of our circle marker. `bindPopup` as has already been explained before *binds* the popup content to our circle marker. But there is a catch. The variable `getRadius` is used to print out the results, which is 2000 of course. However, `bindPopup` only understands strings so we convert our variable result to a string using `toString()`. We also added other strings to give the popup a wholesome result that is understandable to every Tom, Dick, Harry and Harriet.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/circle-radius.jpg"))
```



Finally, let's try with a rectangle. Actually, leaflet allows us to create polygons, and with a polygon –at least in leaflet, you can also create rectangles. Let's work with the polygon class instead.

Copy the following coordinates.

```
// Draw rectangle around Kisumu Wildlife Impala Park
var impalaParkCoordinates = [
    [-0.1144753, 34.743418],
    [-0.115097, 34.745242],
    [-0.114238, 34.745071],
    [-0.114002, 34.746101],
    [-0.115054, 34.746787],
    [-0.115998, 34.745586],
    [-0.118444, 34.746208],
    [-0.121255, 34.744684]
]
```

Now using the `L.polygon` class and a few optional parameters, let's showcase where the Kisumu Wildlife Impala Park is situated.

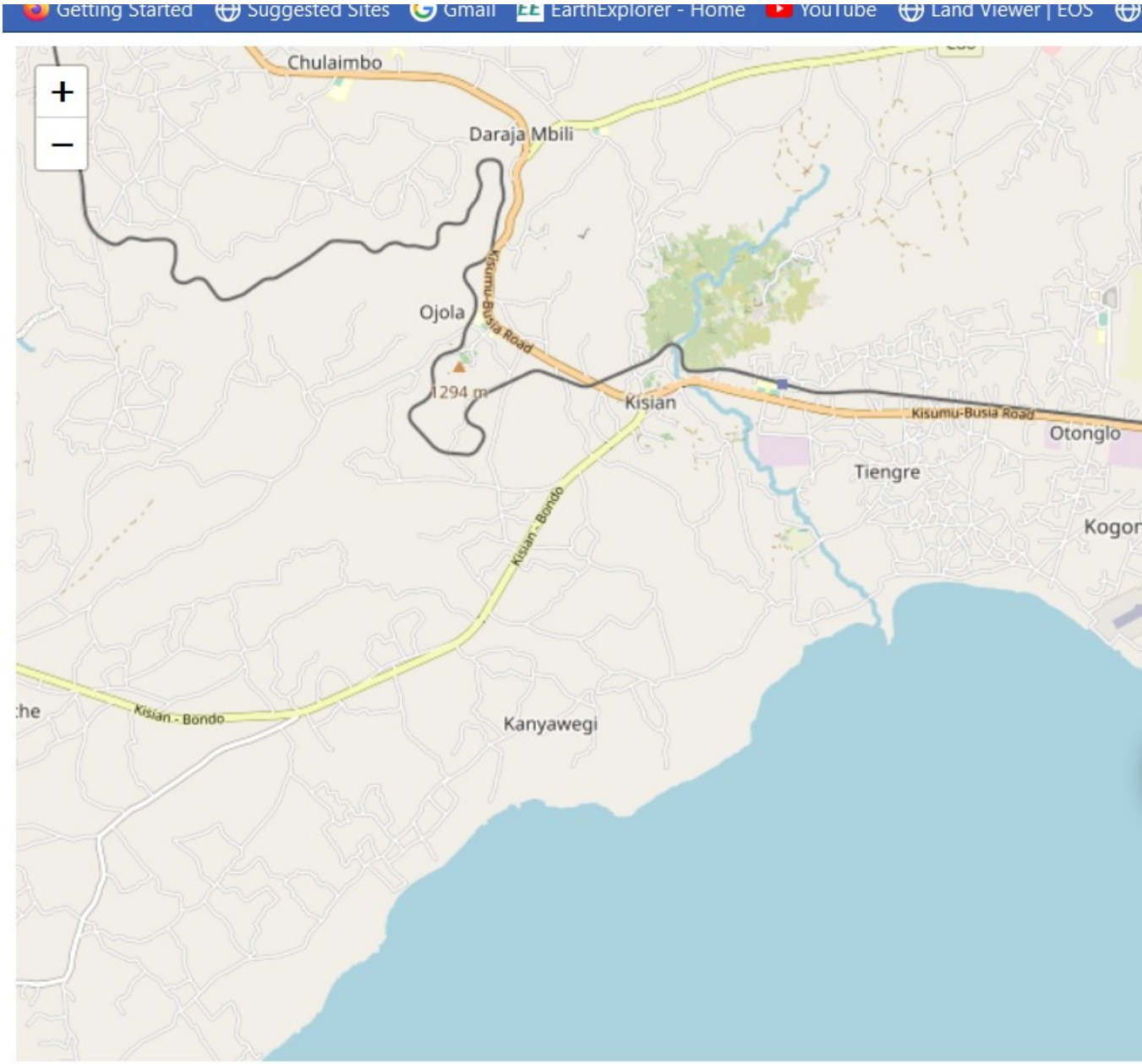
```
// Create a polygon using the above coordinates
var impalaParkPolygon = L.polygon(impalaParkCoordinates, {
    color: 'brown',
    fillOpacity: 0.4
}).addTo(map);
```

You will notice that the Kisumu Museum circle overlaps the location of the Kisumu Impala Park but that is no problem. We will make our popup content appear by default as soon as you load the map. Just like we did for the circle marker, we will make our popup content rely on another variable, in this case `getCenter` which gets the centroid coordinates of our polygon. We were looking for something cooler such as `getArea` in Leaflet, one that automatically prints out the area of a polygon in a popup. Unfortunately, we were unable to find it.

```
// Add popup to the polygon of Kisumu Impala Park
var getCenter = impalaParkPolygon.getCenter();
impalaParkPolygon.bindPopup("Centre is at Lat-Lon: " + getCenter.toString()).openPopup
```

If you find the circle marker radius as an obstraction you can comment out it and its dependancies using `//`.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/polygon-marker.jpg"))
```



You can get the files used in this exercise here.

3.4 Summary

This chapter took you further in enriching the content that can be displayed in a webmap. You have seen that a webmap can offer far more useful information than just mere markers and symbols on a web canvas. Popups are one way of displaying information, and they too can be customized further. Through the practicals in this chapter, you have learnt the following:

- To create popups in leaflet, we use the `bindPopup` method.
- `openPopup` automatically opens the popups once the leaflet map is loaded. They only disappear once you close them.
- Markers can also work with HTML elements.
- Apart from location pins, markers can also be circles and rectangles.
- There exist methods in leaflet that can automatically parse out information in popups without requiring any hardcoding from the programmer. For example, we used `getRadius` to display the circle radius in the popup without necessarily typing it out in the `bindPopup` method.

You comment out JavaScript code with `//`!

Chapter 4

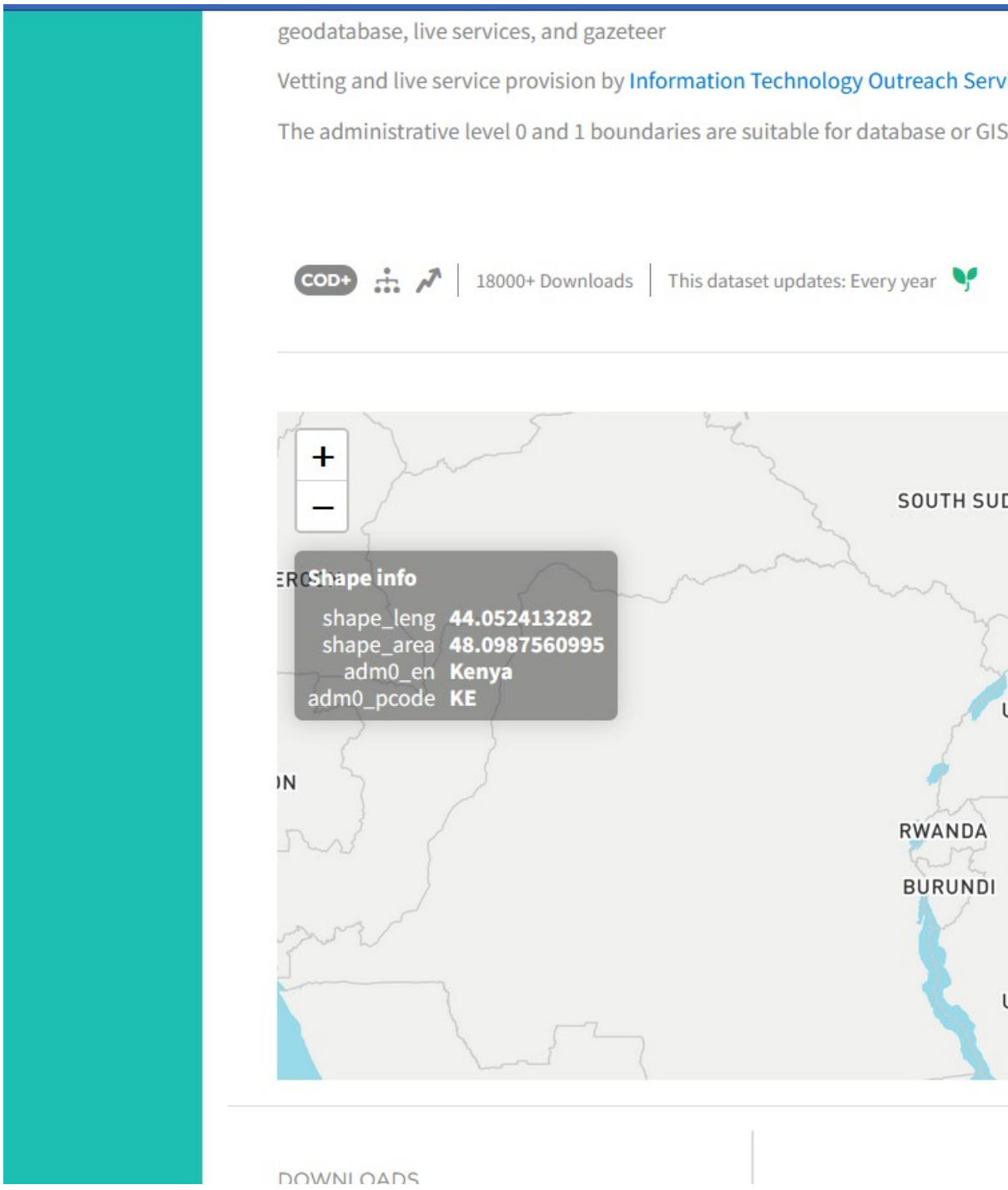
Embedding leaflet map to an external website

4.1 A website with a sense of direction

Now, we have succeeded in making a stand alone leaflet map. However, we want to do something that will quickly upscale you from a novice to a pro. That is, placing the leaflet map into a website.

An example of what we want is shown below, which is a snapshot from the HDX website.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/webmap-in-web.jpg"))
```



For this exercise, we shall embed a leaflet map to a simple HTML webpage. This

webpage doesn't look grand, but it serves the purpose of our exercise. Let's get on to it. Here are the files.

4.2 The HTML webpage

Create a HTML page with the following code.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Pro-GMO Alliance</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="example-styles.css">
        <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css"
        integrity="sha256-kLaT2GOSpHechhsozzB+flnD+zUyjE2LlfWPgU04xyI="
        crossorigin=""/>
        <script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js"
        integrity="sha256-WBkoXOwTeyKclOHuWtc+i2uENFpDZ9YPdf5Hf+D7ewM="
        crossorigin=""></script>
    </head>
    <body>
    <div id="div-for-article">
        <article id="introduction">
            <h2>Introduction</h2>
            <q>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
            doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis
            quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia v
            sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ration
            voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit
            consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labor
            dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exer
            ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis a
            vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae conseq
            vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?</q>
        </article>
    </div>
    <div id="div-for-section">
        <section id="Products">
            <div class="row">
                <h2>Our Products</h2>
                <div class="column">
                  <img src="https://images.unsplash.com/photo-1632125941710-35a9d2fcc7ce?ixlib=rb
                </div>
                <div class="column">
```

```
                    <img src="https://images.unsplash.com/photo-1595615636850-3292eb0a
                  </div>
                <div class="column">
                  <img src="https://images.unsplash.com/photo-1600333859399-228aa03f7d
                </div>
                <div class="column">
                  <img src="https://images.unsplash.com/photo-1630145398476-853543b028
                </div>
              </div>
        </section>
    </div>
    <div>
        <br>
        <h2>Our Branches</h2>
        <br>
    </div>
      <div class="container">
        <div id="map">
          <script src="example-main.js"></script>
        </div>
        <div class="text">
          <h1>Address</h1>
          <p>
            P.O. Box 55044, Nakuru
        </p>
        </div>
      </div>
    </body>
</html>
```

Since this is a geospatial book, we shall not go through every line of the HTML code above. It just a webpage containing some text, some pictures and a webmap. The webmap is the centre of our interest in this chapter. How do we create a Leaflet map, of which we know how to do, and fit it inside an existing webpage?

Before we head there, let's insert the CSS file, which looks like this.

```
/* Three image containers (use 25% for four, and 50% for two, etc) */
.column {
    float: left;
    width: 33.33%;
    padding: 5px;
  }

  /* Clear floats after image containers */
```

```
  .row::after {
    content: "";
    clear: both;
    display: table;
  }


/* Styling the map */
  .container {
    display: flex;
    align-items: center;
    justify-content: center
  }

#map {
    height: 300px;
    width: 90%
}

.text {
    font-size: 15px;
    padding-left: 20px;
}
```

4.3 The leaflet JavaScript code for our website

Back to the leaflet map of our dummy Pro-GMO Alliance webpage. How did we put the leaflet in there? Nothing complicated, just inserting the same leaflet code, with some additional JavaScript and calling it within the HTML file with `<script src="example-main.js"></script>`. The Javascript file used is shown below.

```
var map = L.map('map').setView([-0.302765, 36.146147], 12);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);

var branches = [
    ["Potatoes",-0.328858, 36.008474],
    ["Maize",-0.302765, 36.146147],
    ["Sunflower",-0.224832, 36.159880],
    ["Cotton", -0.214189, 36.135847]
```

```
    ];

for (var i = 0; i < branches.length; i++) {
    marker = new L.marker([branches[i][1], branches[i][2]])
        .bindPopup(branches[i][0])
        .addTo(map);
}
```

The only new thing in the above script is the `for` loop. In JavaScript, the `for` loop is used to iterate over items. In this case, and remembering that indexing in arrays begins from 0, the marker popups will read the latitudes and longitudes which are at index 1 and 2 respectively. The lat-lon are indicated by `([branches[i][1], branches[i][2]])`. The popup strings, which appear as the first elements in the `branches` array, are at index 0. The popup strings are indicated by `branches[i][0]`. At the end of the chain the markers are added to the map with `.addTo`.

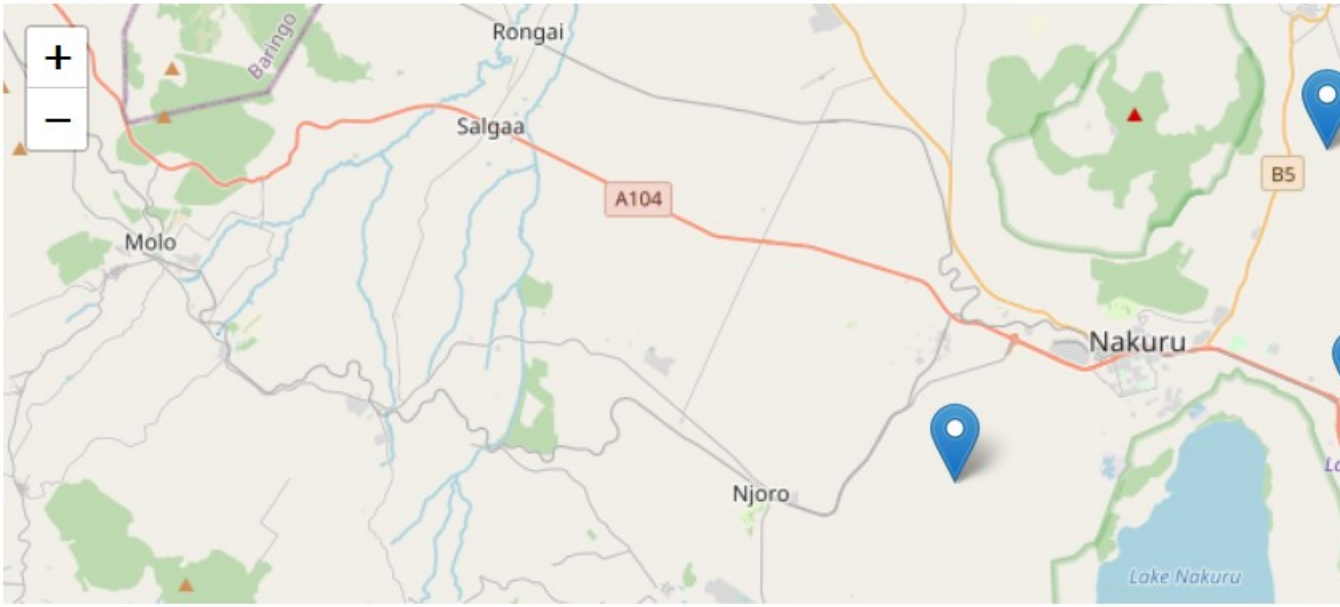
Alright. How about the keyword `new`?

The `new` keyword is a constructor. That is, it creates an empty object. Many instances of the variable `marker` can be created from the instance of `new` object. For more information on the `new` keyword constructor, see this website. Be rest assured that the `new` keyword isn't mandatory to make the webmap to work in our case but its helpful to add another JavaScript trick to your hat.

Below is a snapshot of how the dummy Pro-GMO website looks like with our newly embedded webmap.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/embedded.jpg"))
```

Our Branches



"How are we able to align the webmap to the left and also make other text

stand aside to it?" This is all thanks to the CSS property `display: flex`. `display` is a CSS property that deals with how HTML elements are displayed. This property aligns the element to fill or shrink according to the space available within its assigned portion in the webpage. On the other hand, the CSS property `padding-left` just creates space all around the element, thus creating a neat space between the leaflet map and the address text. Removing this will just make the address text and the leaflet map touch each other edge to edge. The inspiration to use all these CSS properties and values in placing HTML elements side-by-side enamated from this example.

Having done the above, you can consider you are as good a Leaflet mapper to undertake any task!

4.4 Summary

This chapter has introduced you on how you can use CSS to customize the appearance and positioning of your webmap. You have also encountered the use of `for` loop in JavaScript code to retrieve geospatial information, particularly from arrays. Here are brief notes of your take aways from this chapter.

- Leaflet maps can be embedded inside a website as demonstrated in the dummy Pro-GMO webpage.
- One can use `for` loops to iterate over elements from an array and retrieve geospatial information. In this chapter, the `for` loop was used to retrieve both latitude-longitude coordinates and text from the `branches` array variable.
- We can use CSS elements, such as `display` and `padding-left` to position and define how webmap shall be displayed on our webpage.

Chapter 5

Using GeoJSON in Leaflet

5.1 Creating a `.geojson` file

So far, we have created a leaflet map, added some aesthetics such as markers, and even embedded a map into a dummy website. Alright, the website wasn't even close to good, but the methodology should be the same when working with other fully functional and better looking websites. That may be enough to give you confidence to start as a webmapper, but not so fast! There is always, and will be some more things to learn and in this case I would like to introduce another format of storing geospatial information. The use of `.geojson` files.

5.2 What are `.geojson` files?

`.geojson` files, according to the GIS leader ESRI, are an open standard geospatial data interchange format that represents simple geographic features and their nonspatial attributes. GeoJSON is based on the JavaScript Object Notation (JSON) file format which is a lightweight data exchange format that is easily interpretable by both man and machine. In very few instances can any format please both sides of the divide but JSON does, and this site provides examples. Anyway, just like you can tell from the name, it is also based from the JavaScript programming language. If you have worked with JavaScript before, it looks very much like a data format based on dictionaries. In essence, JSON is a large dictionary holding other *dictionaries* of data within it. A GeoJSON is a JSON file that follows a certain structure and has spatial index and geometry specifications in it. See this website for a geojson example.

An example of a GeoJSON file format structure is shown below:

```
{
```

```
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "City": "Nairobi",
        "Population": "4, 300, 000"
      },
      "geometry": {
        "coordinates": [
          36.80617598261199,
          -1.2868825246637812
        ],
        "type": "Point"
      }
    },
    {
      "type": "Feature",
      "properties": {
        "City": "Kisumu",
        "Population": "610, 082"
      },
      "geometry": {
        "coordinates": [
          34.738718987625106,
          -0.10390483386935045
        ],
        "type": "Point"
      }
    },
    ---snip----
```

Below is an example of a json file structure.

```
{

"Influencers" :   [
{
 "name" : "Jaxon",
 "age" : 42,
 "Works At" : "Tech News"
}

{
 "name" : "Miller",
```

```
 "age" : 35
 "Works At" : "IT Day"
}

]
}
```

5.3 Why geojson?

Yours truly could be wrong, but one advantage of `geojson` and `json` is that it's minimal on size (sometimes) and is more portable than shapefiles. Shapefiles are dependent on other data formats that accompany it, such as `.shx`, `.dbf`, `.prj` and others that provide geospatial orientation, metadata, attributes and others. GeoJSON and JSON formats will come as standalone and holding the same amount of data.

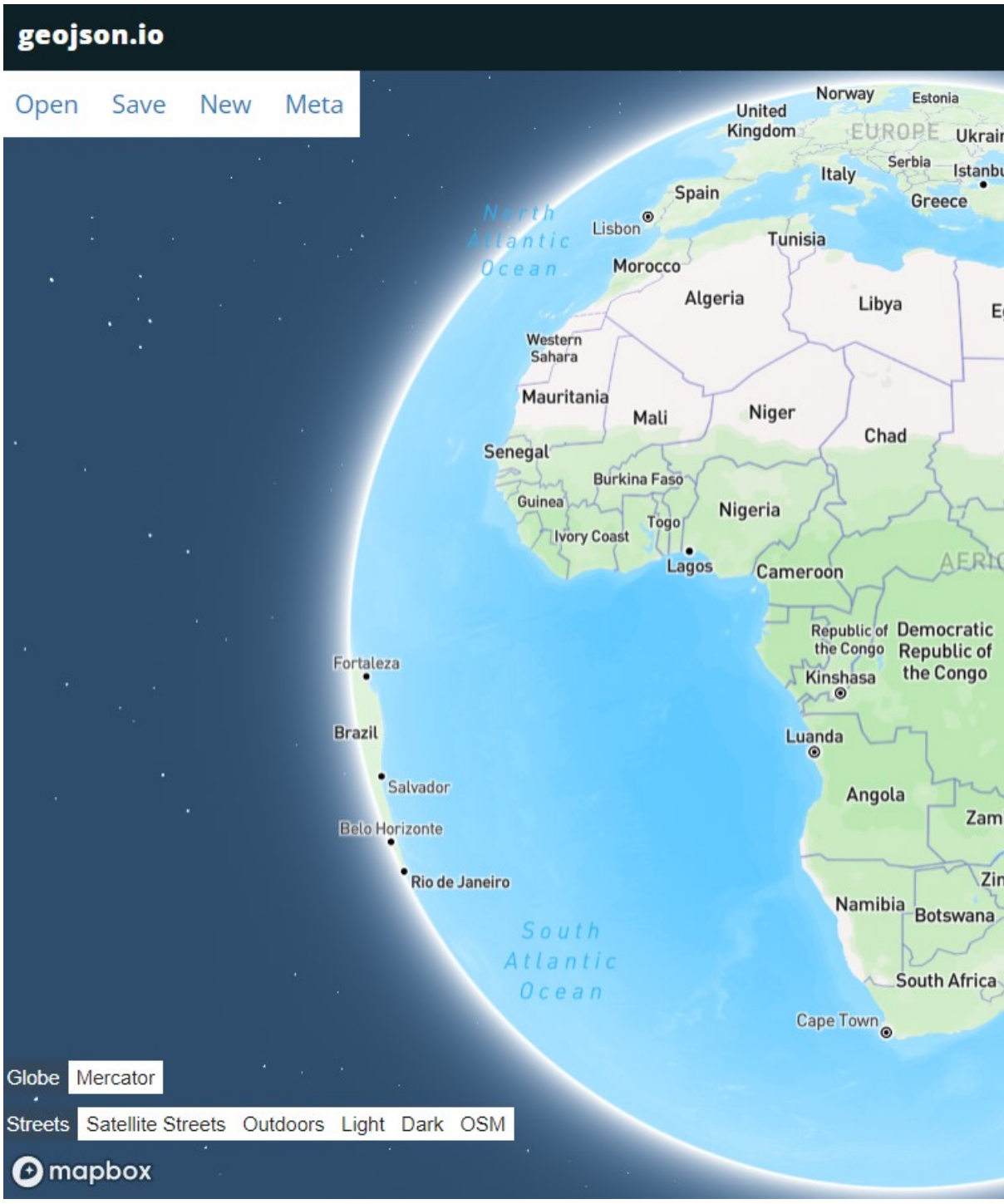
So when do I use shapefiles vis a vis GeoJson? If you want to work with geospatial data in a web interface, `geojson` is the way to go. Period.

5.4 Creating a geojson file

"Creating a geojson file with several dictionaries in it looks very intimidating. No hope of creating one without errors." You may think to yourself. Precisely, but luckily, we have the geojson.io website that does the heavylifting for us. We shall head over to it and create a `geojson` file of some cities and their population.

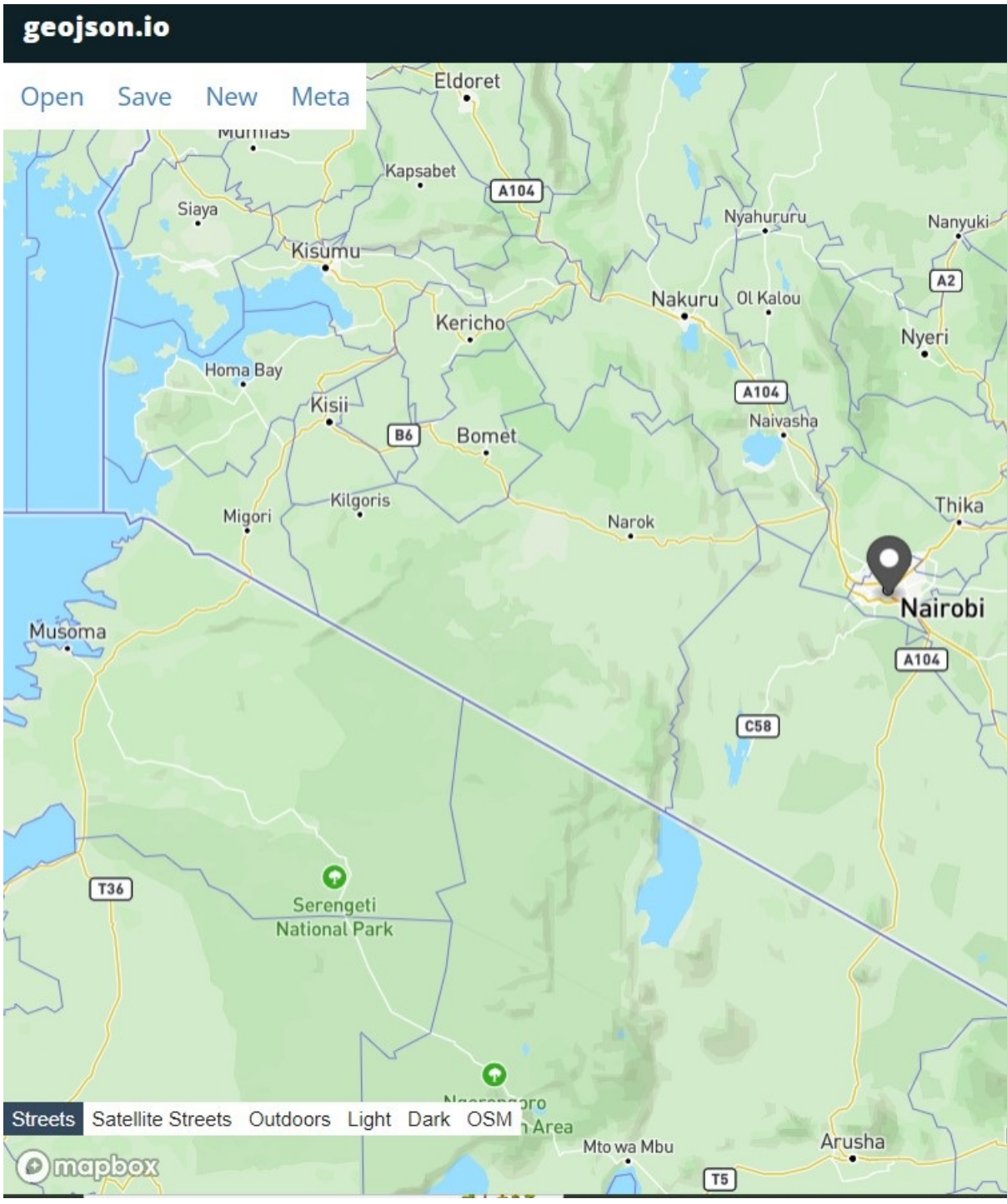
First, of all, the website looks like this, a cool global map powered by Mapbox.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/mapbox-front.jpg"))
```


geojson.io
Open
Save
New
Meta
Norway
Estonia
United Kingdom
EUROPE
Ukrain
Serbia
Istanbu
Italy
Spain
Greece
North Atlantic Ocean
Lisbon
Tunisia
Morocco
Algeria
Libya
Western Sahara
Mauritania
Mali
Niger
Chad
Senegal
Burkina Faso
Guinea
Nigeria
Togo
Ivory Coast
Lagos
Cameroon
Republic of the Congo
Democratic Republic of the Congo
Kinshasa
Fortaleza
Brazil
Luanda
Salvador
Angola
Belo Horizonte
Rio de Janeiro
Namibia
Botswana
South Atlantic Ocean
South Africa
Cape Town
Globe
Mercator
Streets
Satellite Streets
Outdoors
Light
Dark
OSM
mapbox

On your right, under the </>**JSON** tab, the Mapbox folks have already given you a headstart by indicating what feature type and features will go into your `geojson` file. These are important as any website uses these keywords when parsing information from the GeoJSON file. Zoom to Kenya and click a point on top of the Nairobi dot pin, like shown below. Use the highlighted pin in the image below to create a marker over Nairobi.

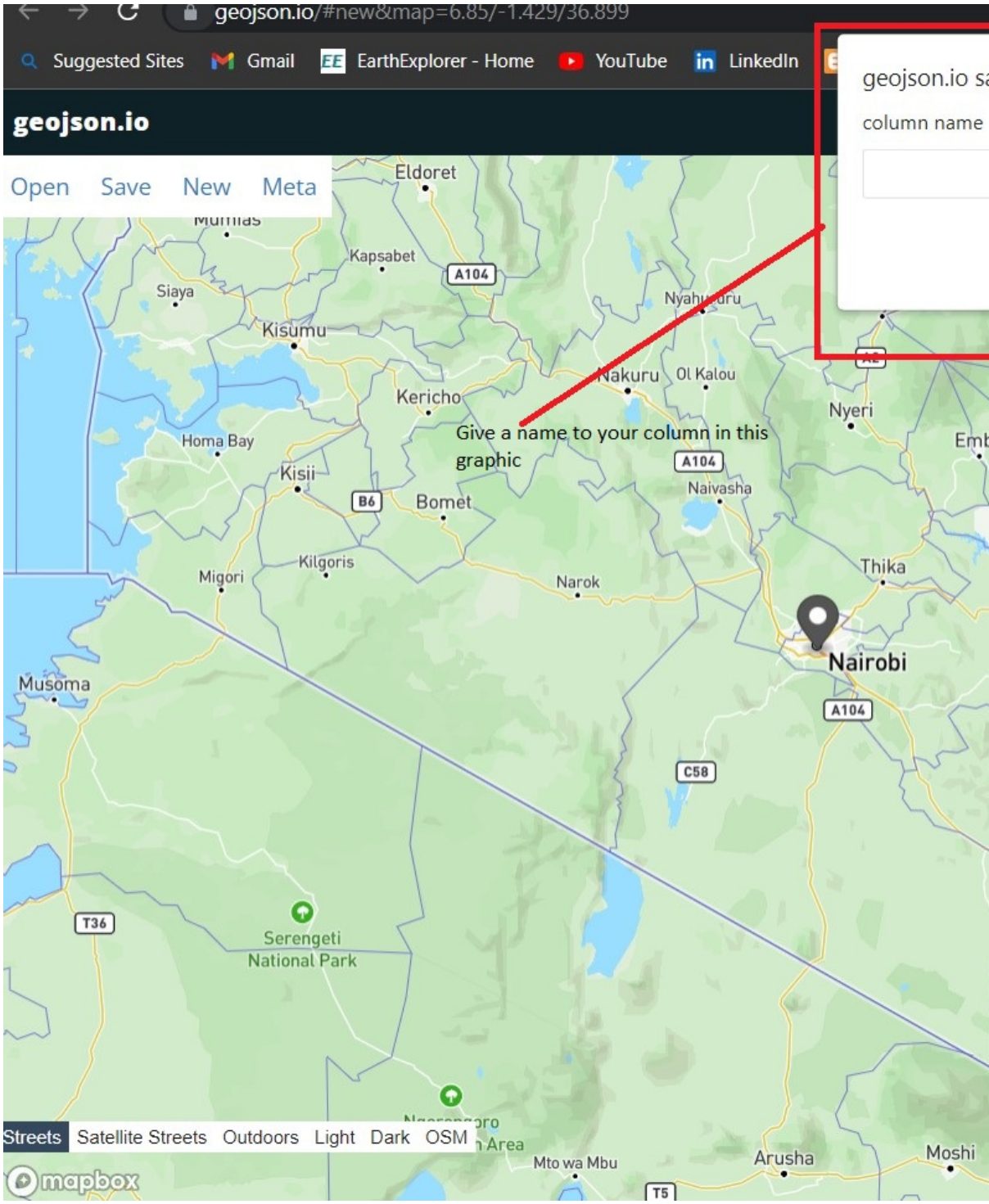
```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/nairobi-pin.jpg"))
```


geojson.io
Open
Save
New
Meta
Eldoret
Kapsabet
A104
Siaya
Kisumu
Nyahururu
Nanyuki
A2
Nakuru
Ol Kalou
Kericho
Nyeri
Homa Bay
Kisii
B6
Bomet
A104
Naivasha
Kilgoris
Migori
Narok
Thika
Nairobi
Musoma
A104
C58
T36
Serengeti
National Park
Streets
Satellite Streets
Outdoors
Light
Dark
OSM
Mto wa Mbu
Arusha
T5
mapbox

By doing so, you will realize that a new dictionary of `type`, `properties` and `geometry` appears within the **features** list. These new keys provide the additional spatial and geometry data in their values in which a website when parsing this information can place them at their appropriate earth locations.

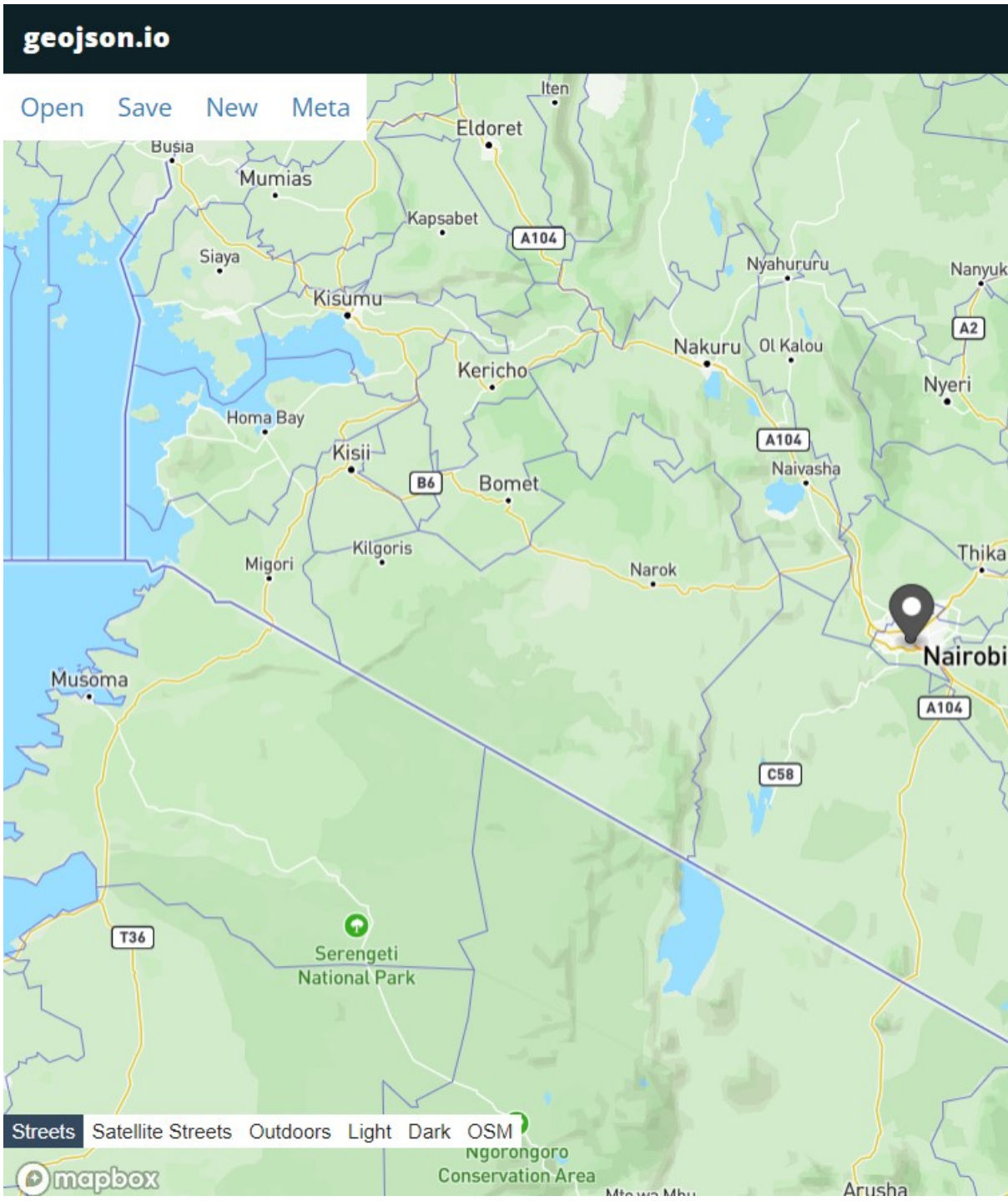
Now move over to the **Table** tab and click new column as shown below.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/table.jpg"))
```


geojson.io/#new&map=6.85/-1.429/36.899
Suggested Sites
Gmail
EarthExplorer - Home
YouTube
LinkedIn
geojson.io
Open
Save
New
Meta
geojson.io sa
column name
Give a name to your column in this graphic
Eldoret
Kapsabet
A104
Siaya
Kisumu
Nyahuru
Nakuru
Ol Kalou
Kericho
Nyeri
Homa Bay
Kisii
B6
Bomet
Naivasha
Migori
Kilgoris
Narok
Thika
Nairobi
Musoma
C58
T36
Serengeti
National Park
Streets
Satellite Streets
Outdoors
Light
Dark
OSM
Mto wa Mbu
Arusha
Moshi
T5
mapbox

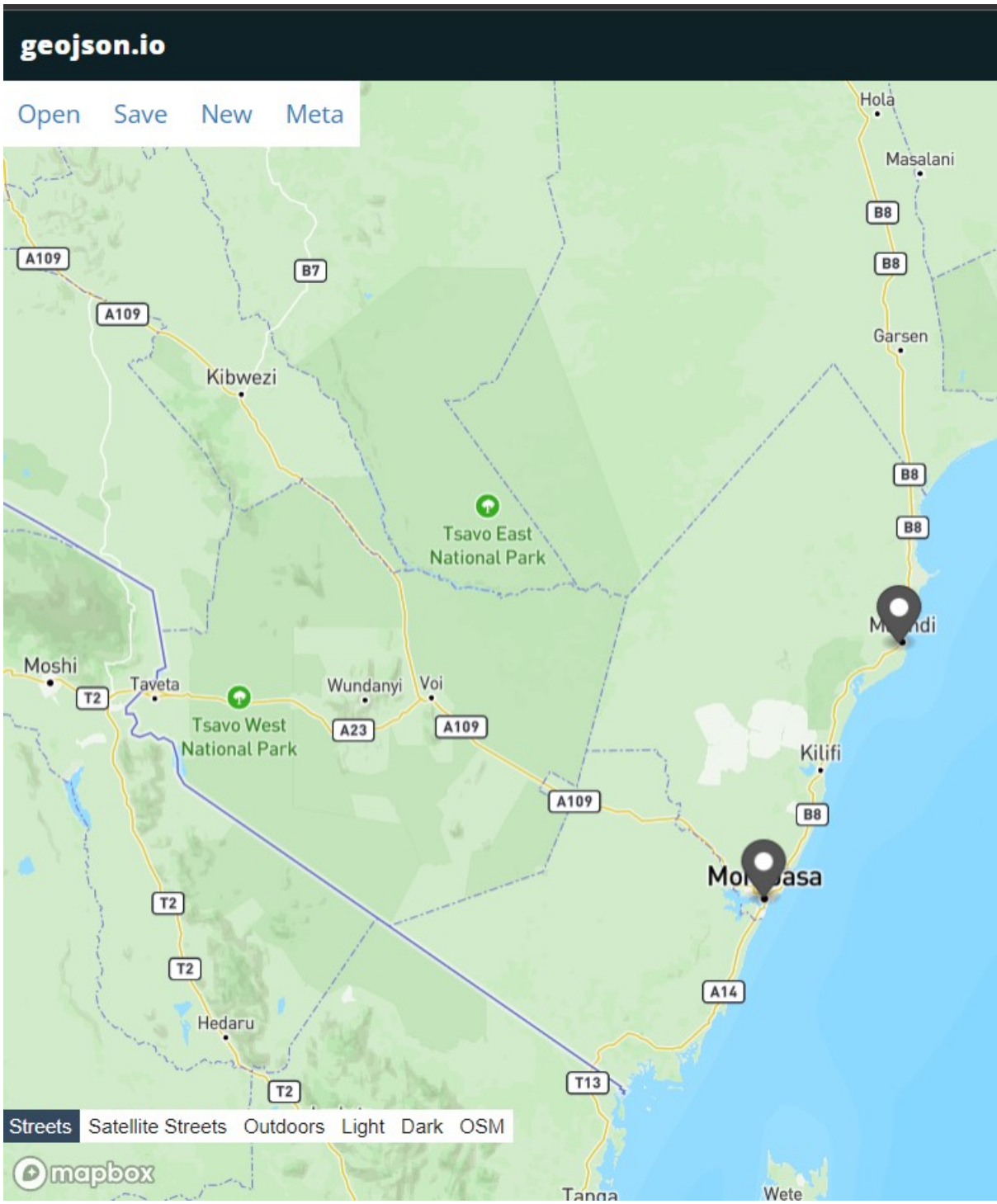
Click on it and in the graphic that appears, give your column the name **City**. Click **Ok** and in the table cell that appears, type **Nairobi**.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/nairobi-named.jpg"))
```


geojson.io
Open
Save
New
Meta
Iten
Eldoret
Busia
Mumias
Kapsabet
A104
Siaya
Kisumu
Nyahururu
Nanyuk
A2
Nakuru
Ol Kalou
Kericho
Nyeri
Homa Bay
Kisii
B6
Bomet
A104
Naivasha
Kilgoris
Migori
Narok
Thika
Nairobi
Musoma
A104
C58
T36
Serengeti
National Park
Streets
Satellite Streets
Outdoors
Light
Dark
OSM
Ngorongoro
Conservation Area
Arusha
mapbox

Now create a new pin over *Kisumu* and a new dictionary will appear below that of Nairobi and a new table row will appear in the **Table** tab. Do the rest for the following cities: Mombasa, Nakuru, Nyeri, Machakos and Malindi. Legally speaking, only the first four are cities by law, the rest are just towns but for the sake of this tutorial, let's corporately refer to them as cities.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/cities-named.jpg"))
```


geojson.io
Open
Save
New
Meta
Hola
Masalani
B8
B8
A109
B7
A109
Garsen
Kibwezi
B8
Tsavo East
National Park
B8
Moshi
Taveta
Wundanyi
Voi
T2
Tsavo West
National Park
A23
A109
Kilifi
A109
B8
T2
T2
A14
Hedaru
T13
T2
Streets
Satellite Streets
Outdoors
Light
Dark
OSM
mapbox
Wete

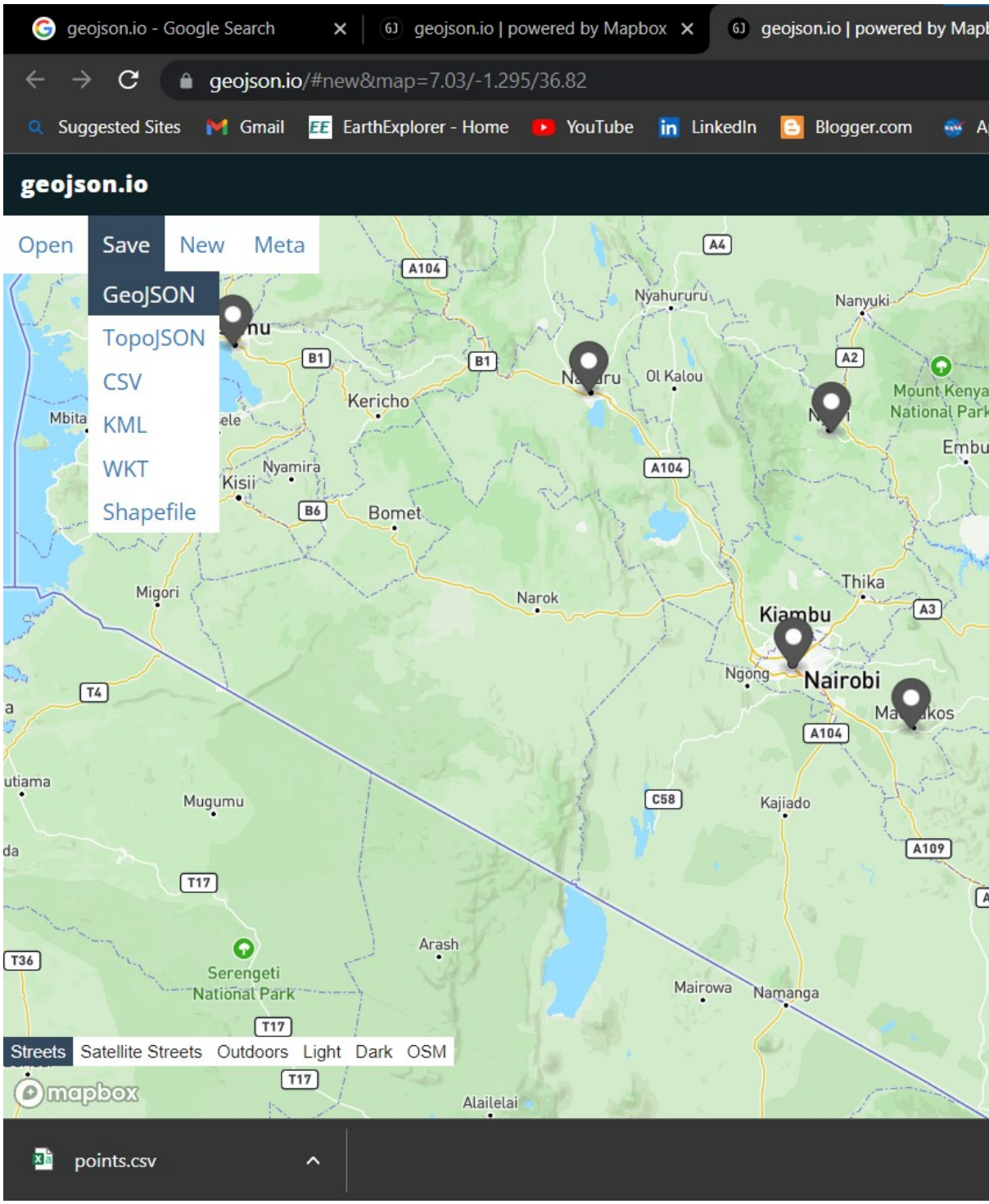
Alright, one process down, one more to go. We will fill these cities with their population statistics. Create a new column with the heading "Population". Fill each of the cities with the following statistics.

```
read.csv("D:/gachuhi/my-leaflet-vs/data/points.csv")[ , 1:2]
```

```
##       City  Population
## 1  Nairobi 4, 300, 000
## 2   Kisumu    610, 082
## 3  Mombasa 1, 440, 000
## 4   Nakuru    422, 000
## 5    Nyeri    759, 164
## 6 Machakos 1, 422, 000
## 7  Malindi    119, 859
```

Once done, head over to the top left of the geojson.io website, and click **Save**. A list of options will appear, click on save as **geojson**. It should appear somewhere in your *Downloads* directory.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-save.jpg"))
```


geojson.io - Google Search
geojson.io | powered by Mapbox
geojson.io/#new&map=7.03/-1.295/36.82
Suggested Sites
Gmail
EarthExplorer - Home
YouTube
LinkedIn
Blogger.com
geojson.io
Open
Save
New
Meta
GeoJSON
TopoJSON
CSV
KML
WKT
Shapefile
Nyahururu
Nanyuki
Ol Kalou
Kericho
Mbita
Nyamira
Kisii
Bomet
Migori
Narok
Thika
Kiambu
Ngong
Nairobi
Mugumu
Kajiado
Arash
Serengeti
National Park
Mairowa
Namanga
Alailelai
Mount Kenya
National Park
Embu
Streets
Satellite Streets
Outdoors
Light
Dark
OSM
mapbox
points.csv

5.5 Saving the Geojson to Github

Now, based on experience, using a local Geojson file (one within your computer directory) is painful in JavaScript. The error I faced had to do with servers or something. To get a way around this and still be able to display `.geojson` data in Leaflet, the data had to be stored on an online server, in this case Github. We would love to show you how to save data on Github, but this would make this chapter long. Therefore, and with sincere apologies, it would be best if you googled it out.

However, on the bright side, we have provided the GeoJSON we will use accessible from this link.

5.6 Loading the GeoJSON into Leaflet

As they always say, there are many ways of killing a rat. There are around three ways in which to load GeoJson data into leaflet, at least from our discovery. We shall start with the easiest and most unreliable to what we consider the best. Let start with the easy one, loading a `.geojson` file from within our Javascript file itself.

5.6.1 The easy way

First of all create a blank JavaScript file called `geojson.js`. Thereafter, go to your `map.html` file which you had created last in Chapter 2 Open it. Change the value in the `src` attribute within the `<script>` tag of the `<body>`–the one inside the `<div>` element with `id="myMap"` to read `"geojson.io"` like below.

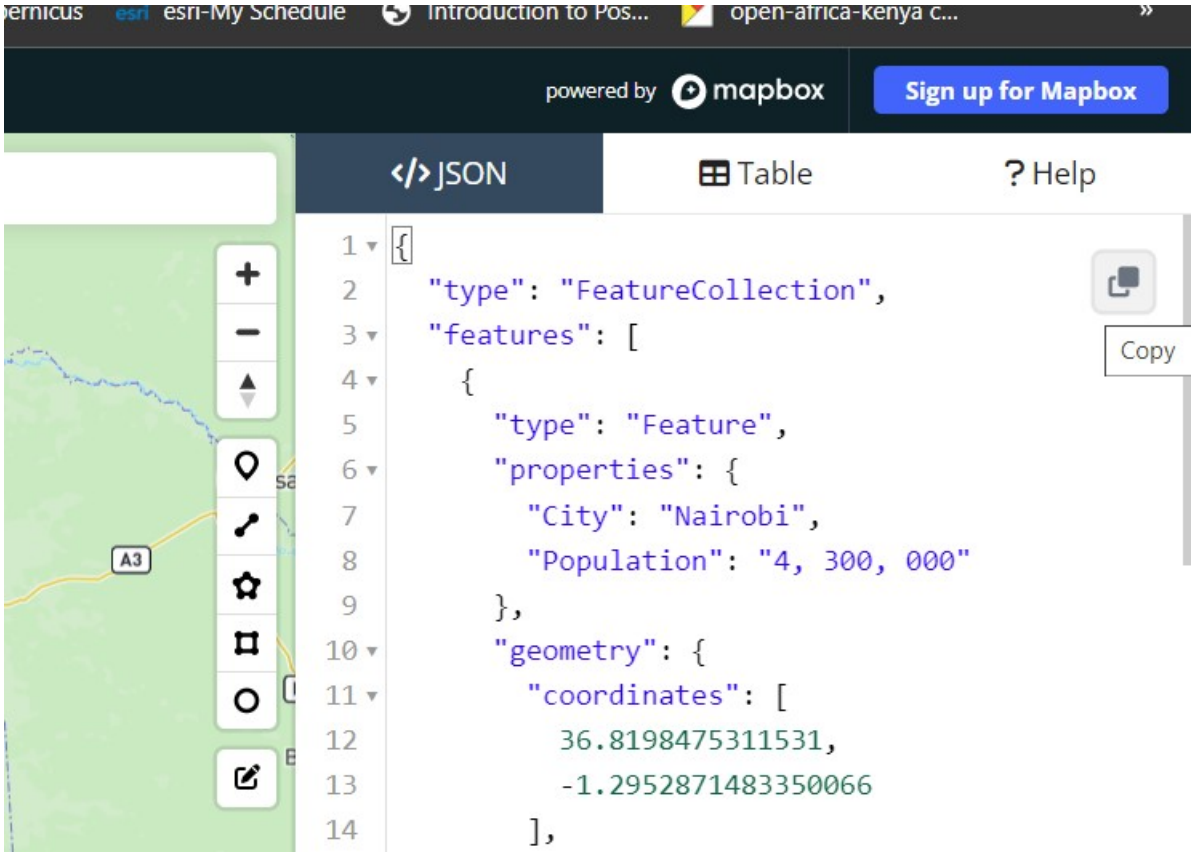
```
<!DOCTYPE html>
<html lang="en">
    <head>
        -- snip ---
    </head>
    <body>
        <div id="myMap">
            <script src="geojson.js">
            </script>
        </div>

    </body>
</html>
```

Alright. Head over to your `geojson.js` and as always, add the leaflet classes `L.map` and `L.tileLayer`. We set the view of our new Leaflet map to that of Nairobi. Your blank `geojson.js` should now be filled with the below code.

```
var map = L.map('myMap').setView([-1.295287148, 36.81984753], 7);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap
}).addTo(map);
```

Okay. Head over to geojson.io website and right under the **</>JSON** you will see a copy icon.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-copy.jpg"))
```



Click it and past it in your `geojson.js` file right above the other `L.map` and `L.tileLayer` classes. Your `geojson.js` should look like below.

```
var cities = {
    "type": "FeatureCollection",
    "features": [
      {
        "type": "Feature",
        "properties": {
          "City": "Nairobi",
          "Population": "4, 300, 000"
        },
        "geometry": {
          "coordinates": [
            36.8198475311531,
            -1.2952871483350066
          ],
          "type": "Point"
        }
      },
      -- snip ---
      {
        "type": "Feature",
        "properties": {
          "City": "Malindi",
          "Population": "119, 859"
        },
        "geometry": {
          "coordinates": [
            40.10521499751357,
            -3.2138767356491655
          ],
          "type": "Point"
        }
      }
    ]
  }


var map = L.map('myMap').setView([-1.295287148, 36.81984753], 7);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);
```

Refresh your `map.html`. It's a map of Kenya alright, but none of our `.geojson` features appear yet. We are about to change that. Leaflet offers the `L.geoJSON` class to add GeoJSON data to a map. The class speaks for itself therefore, let's

use it to add our GeoJSON features. Add the following code below the other leaflet map class layers.

```
L.geoJSON(cities).addTo(map);
```

Refresh your `map.html`. The GeoJSON features should now appear at their exact locations.

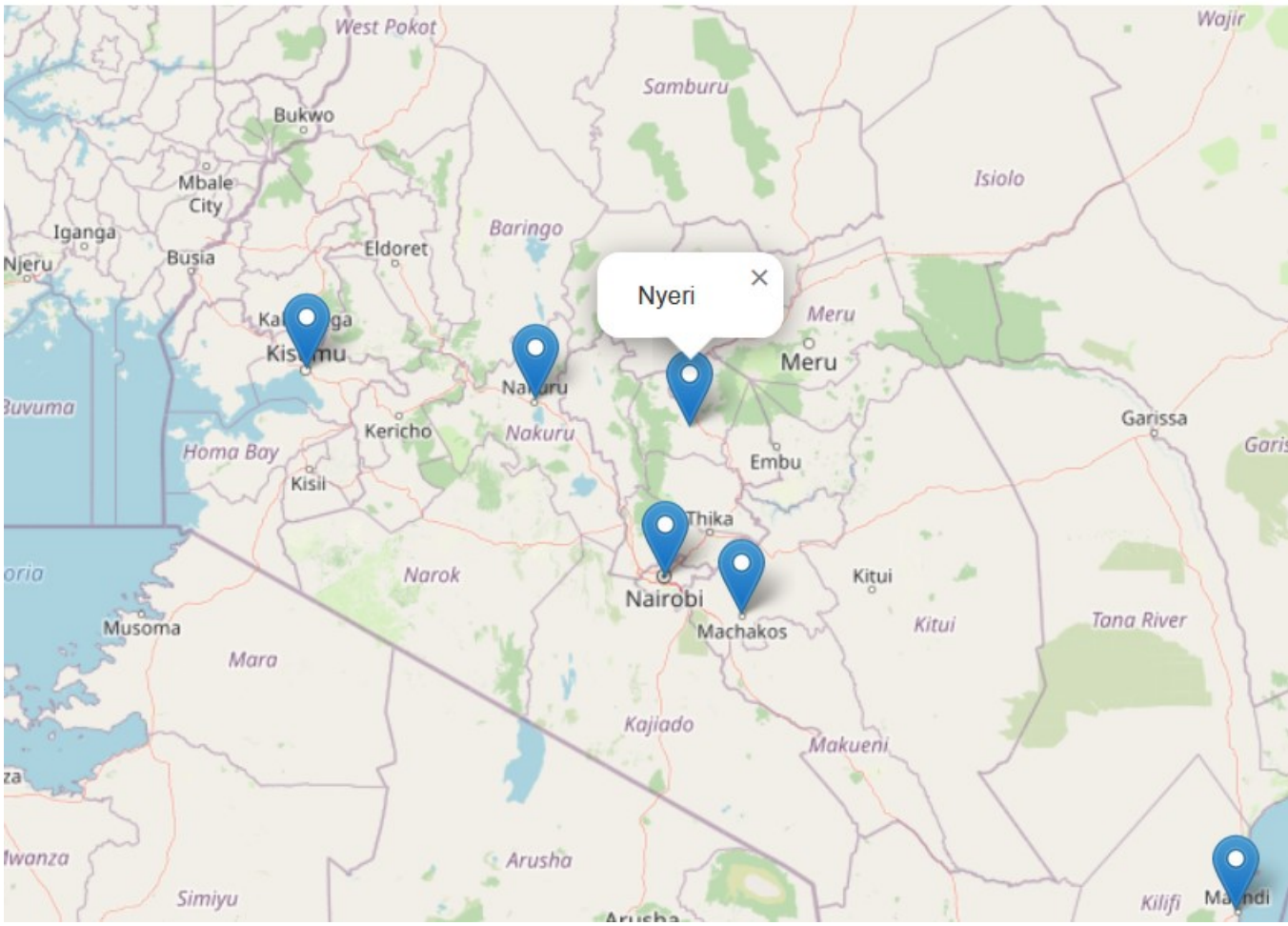
```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-leaflet.jpg"))
```

How about if we made the markers more interactive, say they display popups as we did in Chapter 3? We want the city names to appear when the user clicks on the markers. Easy. Just create a function that does so as in the logic provided

here. We customized it to our case to make sure it references to the `City` key which is part of the dictionary attached to the `properties` key.

```
L.geoJSON(cities).bindPopup(function (layer) {
    return layer.feature.properties.City;
}).addTo(map);
```

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-names.jpg"))
```



There is one issue with this method. If we have a very long GeoJSON data structure, it will clatter our JavaScript file. We only worked with seven cities, but it is very common to work with data holding hundreds and even thousands of dictionaries. That would make your JavaScript file stretch to *ad infitum*.

This brings us to the other two methods, that of using the **Ajax** plugin and using the **Fetch** Application Programming Interface (API). Don't let the words scare you. Take a break, grab a glass of water and come back.

5.6.2 Using the Ajax Plugin

As the term 'plugin' suggests, this is an extension that offers additional functions to the core Leaflet plugin. The Ajax plugin is available from this link. Download it to your directory preferably within the same directory as your `map.html` and `geojson.js`. Alright. Right under the `src` for `leaflet.js` in your `map.html`. Add the following `<script>` tag.

```
<head>
    -- snip --
    <script src="leaflet-ajax-gh-pages\dist\leaflet.ajax.js"></script>

</head>
-- snip --
```

This file will allow you to add `.geojson` files to your Leaflet map. However, there is a catch, both the Ajax and Fetch APIs only work with GeoJson file formats saved on a web server. Based on experience, they will not work with local GeoJson files. As a work around, we saved our GeoJSON file to Github. Don't worry, here is the link to the raw `.geojson` file we had created earlier.

We shall call our GeoJSON file from Github using Ajax as shown in the code below. Please remember to comment out your `var cities` and `L.geoJSON` using \\ because they are irrelevant in this particular case. You actually should.

```
var geojsonLayer = new L.geoJson.ajax("https://raw.githubusercontent.com/sammigachuhi/g
```

Your map should now show the markers of our cities. If you are hawk eyed, you may have noticed that the syntax is a bit different, both from the Ajax Website and that of the `L.geoJSON` from Leaflet. Starting with the latter, we have instead used `L.geoJson` and unlike in the creators website where he used `var geojsonLayer = new L.GeoJSON.AJAX(<your-geojson-file>)` we used the following syntax: `L.geoJson.ajax()` (ajax and geoJson begin with small case). Actually, that's what worked after a lengthy web search.

Just like using `L.geoJSON`, we can also add popups after calling the `.ajax` method.

```
var geojsonLayer = new L.geoJson.ajax("https://raw.githubusercontent.com/sammigachuhi/g
    .bindPopup(function (layer) {
        return layer.feature.properties.City;
    }).addTo(map);
```

Doing so should make the city names appear on click on your leaflet map.

5.6.3 Using Fetch API

Fetch API is a modern interface that allows you to make HTTP requests to servers from web browsers. In other words, it searches for a resource over the web and retrieves it, thus giving it back to you. Think of it as a dog in which you throw a saucer and tell your faithful hound "Sabre, fetch!". The dog runs after the saucer, grips it with its canines before it touches ground and quickly brings it back to you. Same case with Fetch API!

We had mentioned we will show how to retreive our Github stored GeoJSON data and we shall stick to the script. We shall also attempt to explain how this `fetch` works.

First things first. We shall call the `fetch` function and pass it our Github url containing our GeoJSON text. Since `fetch` is an API, it retrieves data from Github.io –the server in this case–and brings it to our laptop the – the client. Enough IT. Let's write it down.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/cities-geojson.geojson")
```

Okay. Next step, once the server beams back the data to us, what do we do with it? According to Digital Ocean, the response is not *actually* the data in the original format but rather *a series of methods that can be used depending on what you want to do with the information.* We have to convert the object to a specific format. In this case, a JSON format. And to do this data conversion, the `json()` method is used. Actually, a function is created to do this. In the code below the function appears in parenthesis after `.then`.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/cities-geojson.geojson")
    .then(function(response) {
        return response.json()
    })
```

The above function takes the `response` argument and convert it to JSON by appending the `.json()` method to it.

After converting our response to JSON, it still needs to be processed further. Processed to what? To a GeoJSON file and subsequently add it to our Leaflet Map.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/cities-geojson.geojson")
    .then(function(response) {
        return response.json()
```

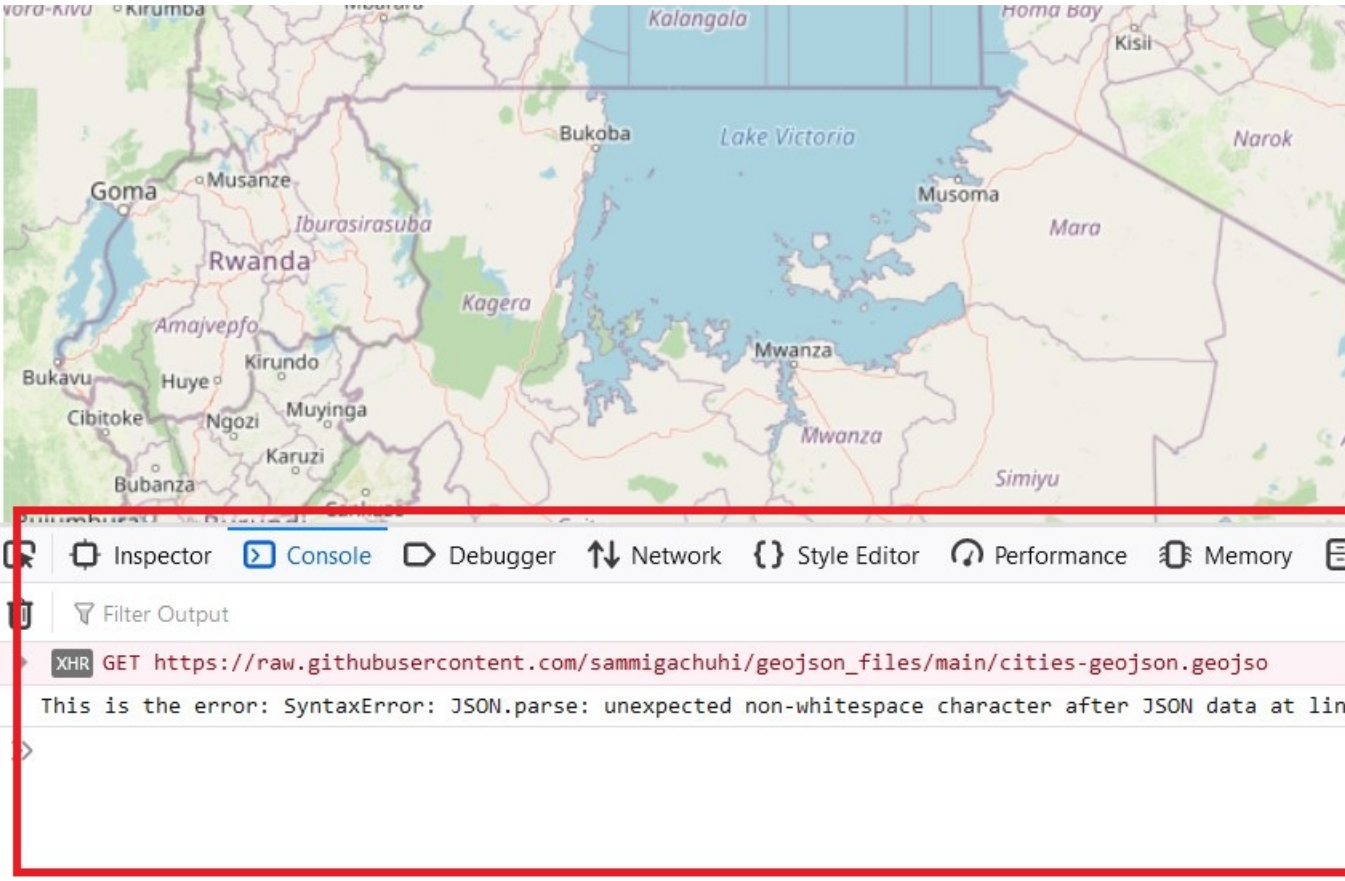
```
    })
    .then(function(data) {
        L.geoJson(data).addTo(map);
    })
```

We shall also add one more function –the `catch()` method. `catch()` is a method that returns an action if our response to the server has been rejected. We shall show the code and demonstrate it.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/cities-geojso
    .then(function(response) {
        return response.json()
    })
    .then(function(data) {
        L.geoJson(data).addTo(map);
    })
    .catch(function(error) {
        console.log(`This is the error: ${error}`)
    })
```

To see the `catch()` in action, omit the last letter in our url so that it reads `cities-geojson.geojso`. You read it right. Just omit the letter 'n' for now for goodness sake. Reload your `map.html`. Right click the webmap page and click on **Inspect** in the small interface that appears. Head over to the console tab and see the error response. It should read like below.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/catch-error.jpg"))
```



Restore the omitted letter and reload your `map.html`. Your leaflet map should have the city markers overlain just like in the case of using Ajax plugin or hard-coding the Geojson data into `var cities`. To stretch your Javascript skills further, we can shorten our code further by retaining the arguments `response`, `data` and `error` and using the arrow function `=>` to pass on the return statements, like so.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/cities-geojson.geojson")
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        L.geoJson(data).addTo(map);
    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

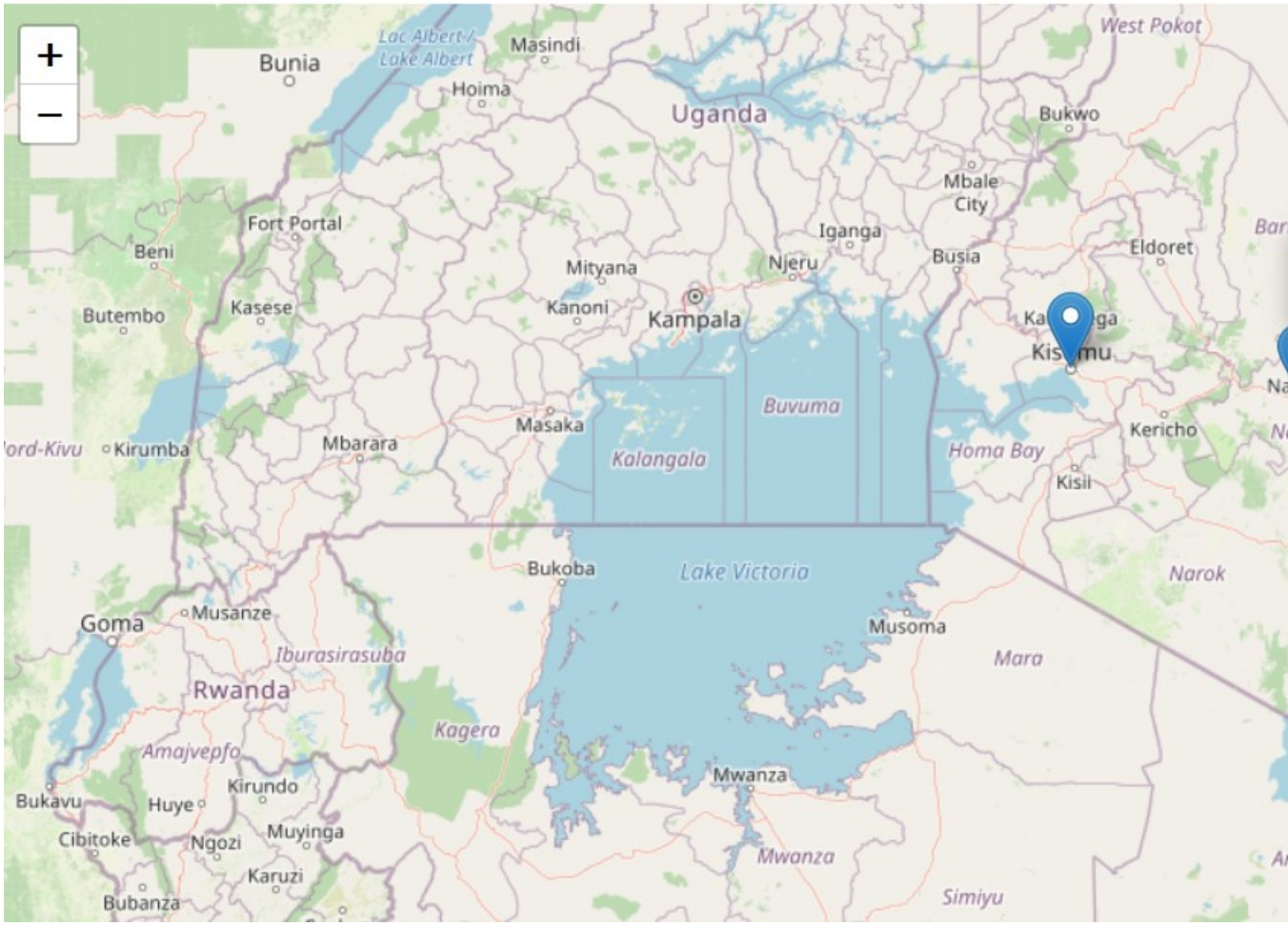
It looks cryptic but don't cry! We have only ommitted the `function()` keyword and instead added `=>` between `function()` and the curly brackets `{<code-to-run>}` which is the function body, aka where the magic happens. Just like when using Ajax, we can also add other functionalities within `L.geoJSON`. In here, and thanks to the use of template literals (``` `` ```), we can even add statements and refer to our GeoJson keys (and in some cases, even variables) using `${}`. Whatever is within the `${}` is executed and passed out as a string to the template literals.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/cities-geojso
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        L.geoJson(data).bindPopup((layer) => {
            return `City: ${layer.feature.properties.City},<br>
            Population: ${layer.feature.properties.Population}`}).addTo(map);
    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

In our case, we added the HTML tag `<br>` to separate the `City` and `Population` keys from our GeoJSON. What we have are neat markers showing both the city name and population figures in two separate lines.

Phew! Enough Javascript for a day!

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/city-population.jpg"))
```



Here are the full files.

Chapter 6

Create your own custom markers

Full codes and files are here.

Hope you didn't trash away the cities we created in the last chapter. In this chapter, we shall focus on creating your own custom markers. We love a clean job, so we will create a new JavaScript file and name it `custom-markers.js`. We understand the previous chapter was quite long but believe you me, although creating custom markers sounds easier, it took us way longer to get the hang around it. Sounds ironic but its the truth. Good news, we received enough punches on the face to teach you how to dodge the pain points.

The very first thing is to create a basemap.

```
var map = L.map('myMap').setView([-1.295287148, 36.81984753], 7);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);
```

We will create a new variable called `cities` that mimics the geojson file saved to github only these time the population values have been tweaked a bit. Paste the below code to your `custom-markers.js`.

```
var cities = {
    "type": "FeatureCollection",
    "features": [
      {
```

```
  "type": "Feature",
  "properties": {
    "City": "Nairobi",
    "Population": 4300000
  },
  "geometry": {
    "coordinates": [
      36.8198475311531,
      -1.2952871483350066
    ],
    "type": "Point"
  }
},
{
  "type": "Feature",
  "properties": {
    "City": "Kisumu",
    "Population": 610082
  },
  "geometry": {
    "coordinates": [
      34.74657469430895,
      -0.10402992528247523
    ],
    "type": "Point"
  }
},
{
  "type": "Feature",
  "properties": {
    "City": "Mombasa",
    "Population": 1440000
  },
  "geometry": {
    "coordinates": [
      39.66358575335434,
      -4.041883912902392
    ],
    "type": "Point"
  }
},
{
  "type": "Feature",
  "properties": {
    "City": "Nakuru",
    "Population": 422000
```

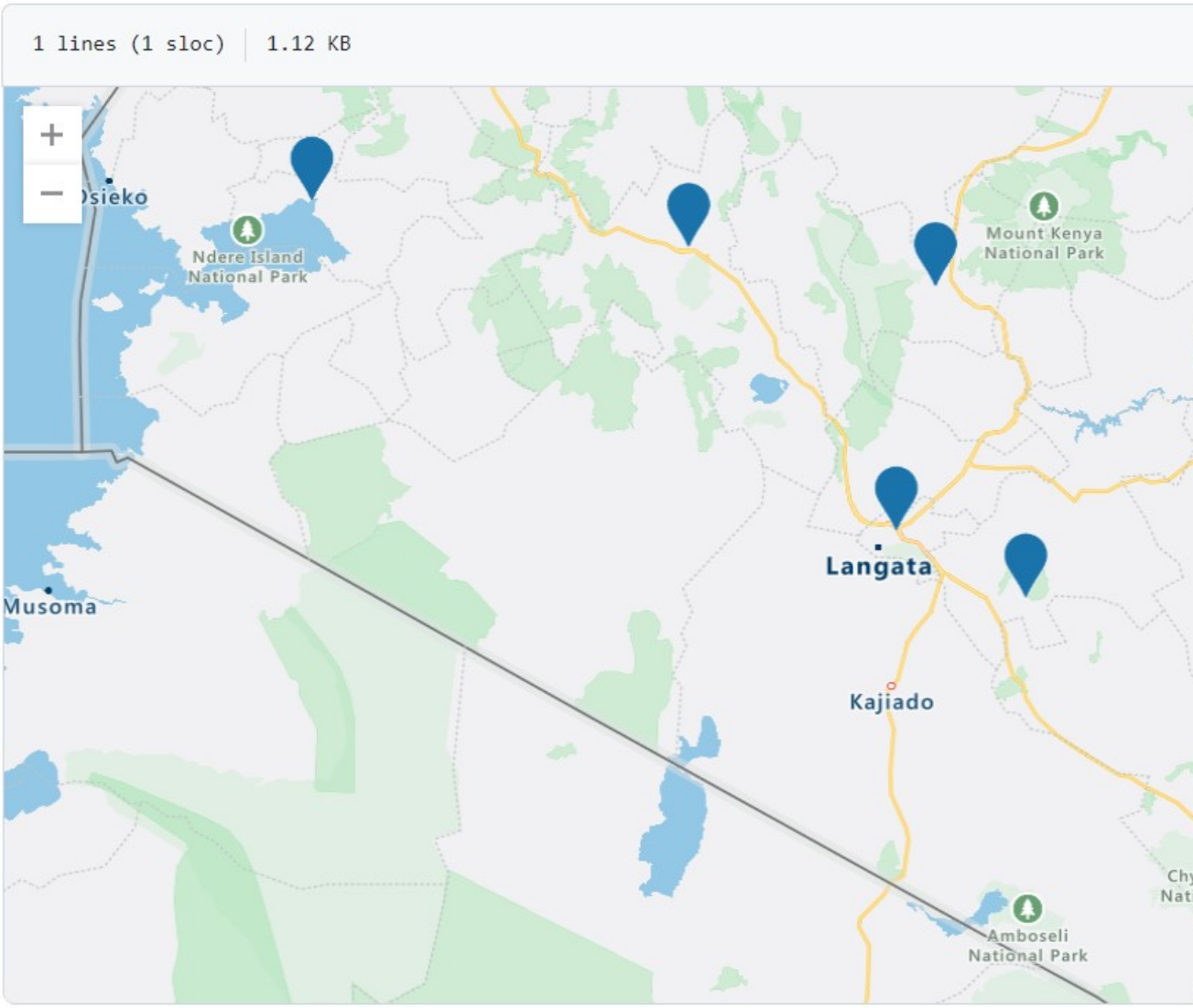
```
  },
  "geometry": {
    "coordinates": [
      36.06412271026528,
      -0.2754534004690896
    ],
    "type": "Point"
  }
},
{
  "type": "Feature",
  "properties": {
    "City": "Nyeri",
    "Population": 759164
  },
  "geometry": {
    "coordinates": [
      36.957036675396154,
      -0.42345404217887506
    ],
    "type": "Point"
  }
},
{
  "type": "Feature",
  "properties": {
    "City": "Machakos",
    "Population": 1422000
  },
  "geometry": {
    "coordinates": [
      37.25780808801821,
      -1.518874011494134
    ],
    "type": "Point"
  }
},
{
  "type": "Feature",
  "properties": {
    "City": "Malindi",
    "Population": 119859
  },
  "geometry": {
    "coordinates": [
      40.10521499751357,
```

```
            -3.2138767356491655
          ],
          "type": "Point"
        }
      }
    ]
  }
```

Can you notice any difference on the `Population` property compared to the code Chapter 5? If you are hawkeyed, you will see the Population values this time round are integers compared to strings in the previous chapter. It sounds superflous to create population values as strings only to convert them to integers now, but please do remember the geojson.io site did that for us, not this author. Here is the raw geojson script customized for this chapter.

Just a small note before going on. When the GeoJSON file has the population values enclosed in strings `""`, they are automatically rendered on a map on the Github server.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-webmap.jpg"))
```



However, when the strings are removed, and the population values remain as integers, they are no longer rendered on a webmap as shown below.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-nowebmap.jpg"))
```

It's just a dictionary of lists and other dictionaries.

6.1 The icons

Alright. Let's create a map of our cities but with custom markers this time round. The below code creates our custom markers.

```
// Yellow Icon
var yellowIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/i
    shadowUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/images/marker-sha
    iconSize: [25, 41],
    iconAnchor: [12, 41],
    popupAnchor: [1, -34],
    shadowSize: [41, 41]
  });

// Orange Icon
var orangeIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/i
    shadowUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/images/marker-sha
    iconSize: [25, 41],
    iconAnchor: [12, 41],
    popupAnchor: [1, -34],
    shadowSize: [41, 41]
  });

// Red Icon
var redIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/i
    shadowUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/images/marker-sha
    iconSize: [25, 41],
    iconAnchor: [12, 41],
    popupAnchor: [1, -34],
    shadowSize: [41, 41]
  });
```

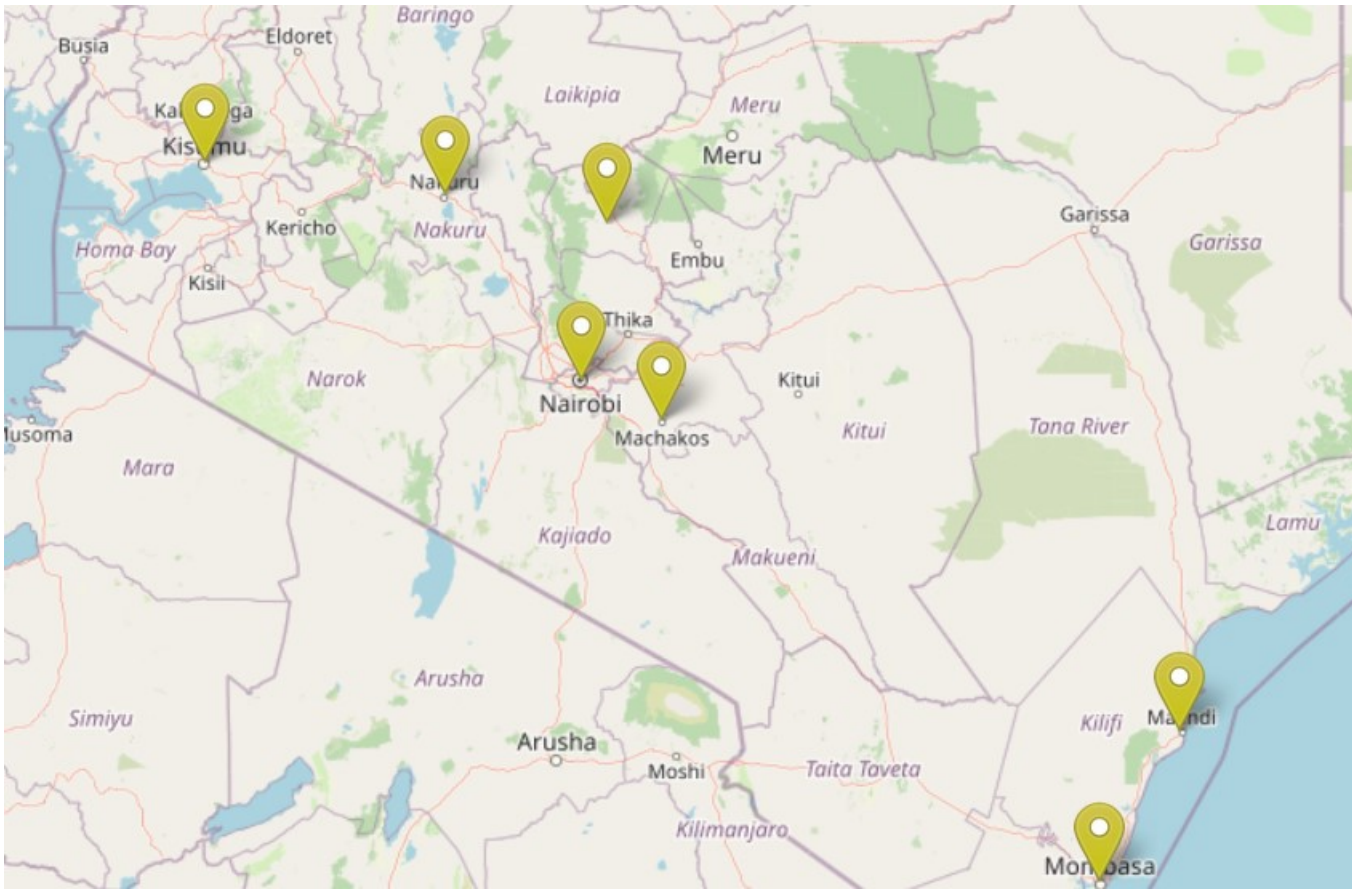
We created three markers in order of importance. One is yellow, the other orange, the other red. You will see the significance (not so much) of these colors later.

Time to create GeoJSON markers out of this.

```
L.geoJSON(cities, {
    pointToLayer(feature, latlng) {
        return L.marker(latlng, {icon: yellowIcon});
    }
}).bindPopup(function (layer) {
    return layer.feature.properties.City;
}).addTo(map);
```

We got this.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-markers.jpg"))
```



All cities were marked yellow, irrespective of their population or jurisdictional significance. But before we raise the alarm on this, what was the purpose of the `pointToLayer()` function? According to the Leaflet guide, the `pointToLayer()` function is a special function for GeoJSON variables that species how they should be drawn. In this case and in any other case, it passes the commands in the body, such as `return L.marker...` for every marker point at its Lat-Lon coordinates.

6.2 Differentiate custom markers on a webmap

Now to our spirited arguments. Not all cities are equal. We would appreciate if the markers would differentiate the cities to a particular variable, say population. By the way, size of cities is determined by population. The below code shall differiante cities by circle markers before we go back to normal markers when we touch the `fetch()` function again.

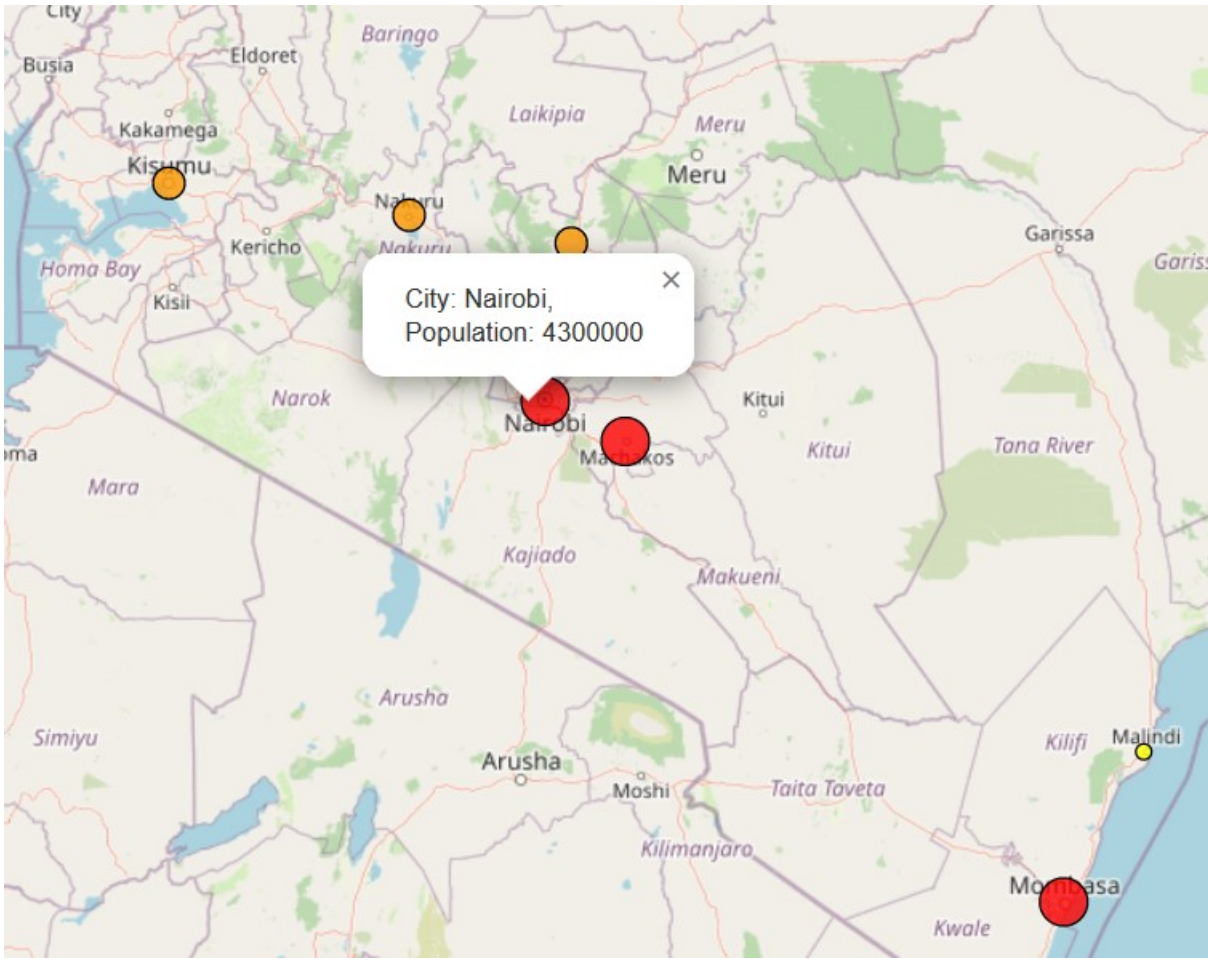
Comment out the earlier code and insert this:

```
L.geoJSON(cities, {
  pointToLayer: function (feature, latlng) {
    if (feature.properties.Population <= 250000) {
      return L.circleMarker(latlng, {
        radius: 4,
        fillColor: '#FFFF00',
        color: '#000',
        weight: 1,
        opacity: 1,
        fillOpacity: 0.8
      });
    } else if (feature.properties.Population <= 800000) {
      return L.circleMarker(latlng, {
        radius: 8,
        fillColor: '#ff9900',
        color: '#000',
        weight: 1,
        opacity: 1,
        fillOpacity: 0.8
      });
    } else {
      return L.circleMarker(latlng, {
        radius: 12,
        fillColor: '#FF0000',
        color: '#000',
        weight: 1,
```

```
        opacity: 1,
        fillOpacity: 0.8
      });
    }

  }
}).bindPopup(function (layer) {
  return `City: ${layer.feature.properties.City},<br>
  Population: ${layer.feature.properties.Population}`;
}).addTo(map);
```

What you get is a map that has different circle markers according to the respective city's population. This time round, the `pointToLayer()` function body worked with **if/else if** statement to differentiate the radius and color of each circle marker. Under the **if/else if** code statement block, different circle marker specifications of radius and fillColor were inserted for each population category. Any city with a population beyond 800000 was fitted into the `else` block.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-diff1.jpg"))
```



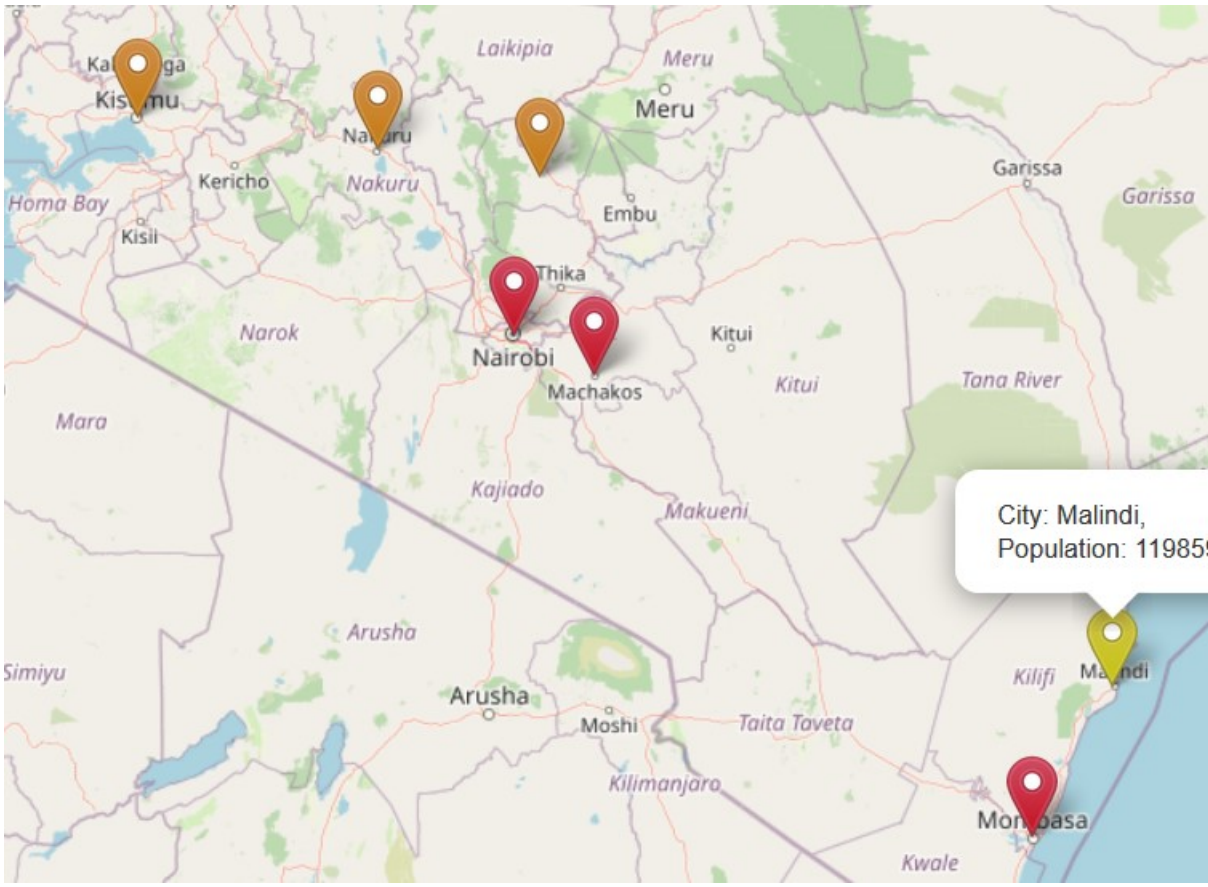
This brings us to why we changed the population values from strings to integers. If we were to work with their former values of strings, any value beyond 1, 000, 000 would receive the settings of `feature.properties.Population <= 250000` that is, color yellow and radius yellow. This is because even though 1 million is by far larger than 250, 000 or 800, 000, because it starts with a 1, it shall be considered even less than 250, 000! This is because when ordering strings in JavaScript, they are by default ordered by the first character. The same case applies to digits 1 to 9. Value 1 will always be inferior to 2, 7, 9 and any other number in between irrespective of whether its a million or a billion.

In the next code sample, we shall create city markers but their populations differentiated with new markers icons we created of yellow, orange and red.

```
L.geoJSON(cities, {
  pointToLayer: function (feature, latlng) {
    if (feature.properties.Population <= 250000) {
```

```
      return L.marker(latlng, {
        icon: yellowIcon
      });
    } else if (feature.properties.Population <= 800000) {
      return L.marker(latlng, {
        icon: orangeIcon
      });
    } else {
      return L.marker(latlng, {
        icon: redIcon
      });
    }

  }
}).bindPopup(function (layer) {
  return `City: ${layer.feature.properties.City},<br>
  Population: ${layer.feature.properties.Population}`;
}).addTo(map);
```

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geojson-diff2.jpg"))
```



In the above map, our preceding code has differentiated large cities with populations above 800, 000 with a red marker, those with populations below 250, 000 with a yellow marker and those between with an orange marker. In all cases, our `bindPopup()` still contains the same settings to show both the city name and population.

The `pointToLayer()` function was passed function that return a specific `L.marker` with Lat-Lon coordinates for each respective marker retrieved of course, only that the `icon` key was assigned a different marker icon variable corresponding to each city's population as the value. That's the trick that assigns a different marker to each city as per its population.

6.3 Using fetch

Remember how `fetch` helped us retrieve data from a server in a previous chapter. Whereas we won't repeat the entire process again (you can breathe a sigh of

relief), the same iterations of differentiating a marker icon can also be inserted in `fetch`, right within the options of `L.geoJson(data, {options})`.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/cities-geojson2.geojson.
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        L.geoJson(data, {
          pointToLayer: function (feature, latlng) {
            if (feature.properties.Population <= 250000) {
              return L.marker(latlng, {
                icon: yellowIcon
              });
            } else if (feature.properties.Population <= 800000) {
              return L.marker(latlng, {
                icon: orangeIcon
              });
            } else {
              return L.marker(latlng, {
                icon: redIcon
              });
            }

          }
        }).bindPopup((layer) => {
            return `City: ${layer.feature.properties.City},<br>
            Population: ${layer.feature.properties.Population}`}).addTo(map);
    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

It works but don't take my word for it. Just paste and see.

6.4 Unique custom markers

This part may not be necessary, but it is just to show you that there are various markers apart from the defaults provided by Leaflet. One can create custom markers outside of leaflet using the [Leaflet.Awesome.Markers plugin]. Just like in the case of Ajax, you will need to install the path to the dependencies in the html document `map.html` using `<script>`. Insert the following `<script>` tags into `map.html`.

```
<script src="Leaflet.awesome-markers-2.0-develop\Leaflet.awesome-markers-2.0-develop\di

<script src="Leaflet.awesome-markers-2.0-develop\Leaflet.awesome-markers-2.0-develop\di
```

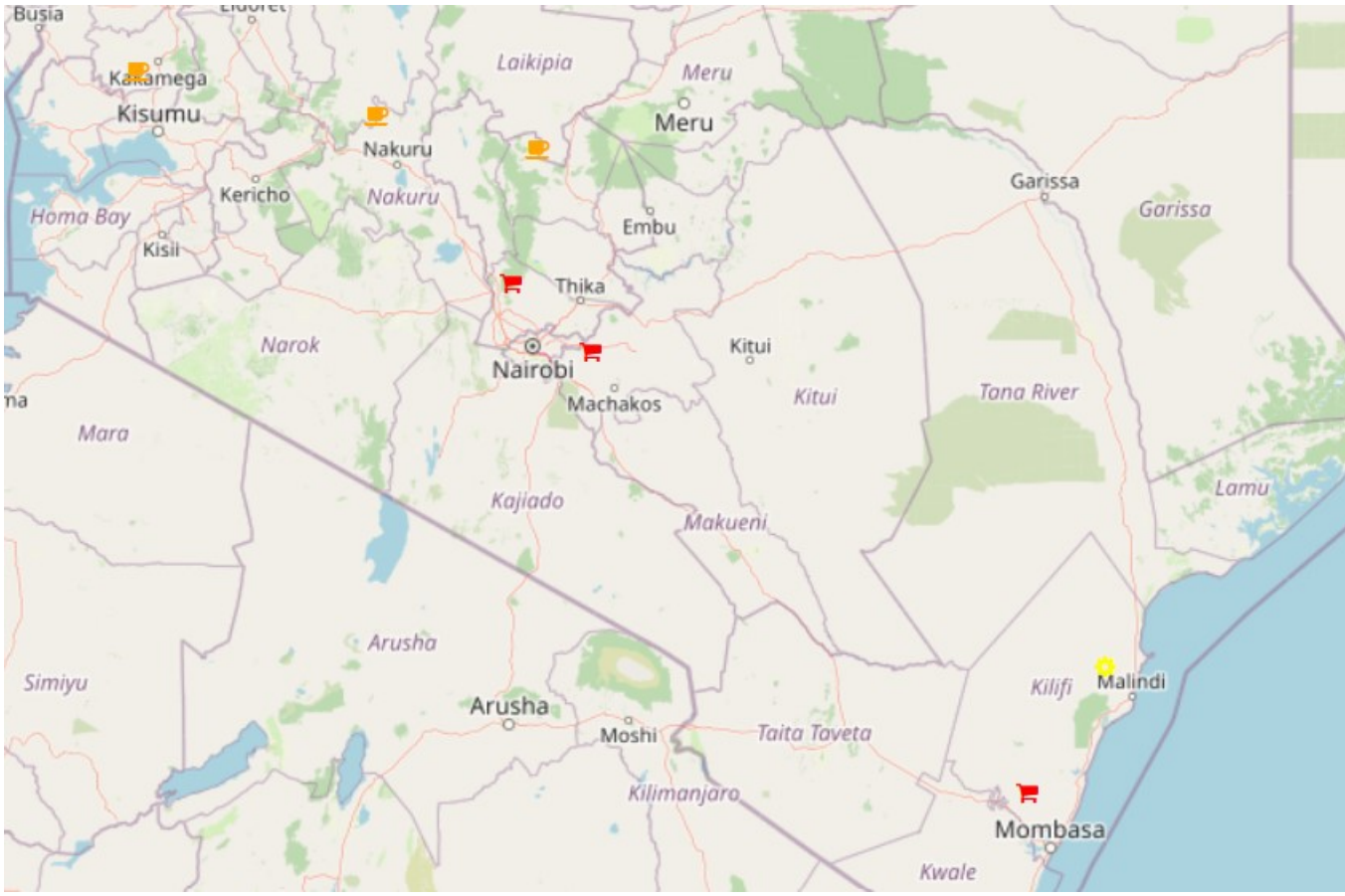
And also this `<link>` tag:

```
<link href="http://netdna.bootstrapcdn.com/font-awesome/4.0.0/css/font-awesome.css" rel
```

To make the best use of time, following the example on their site, we simply replace our `icon` values with the new `L.AwesomeMarkers.icon` for each population category, and also tweaked the colors for each to match those of our previous markers. We assumed that big cities have the best malls, the medium cities, those with above a population of 250, 000 but a peak of 800, 000 to also have good coffee places, and those with populations of less than a quarter million have respectable industries. We assume fair play has been exercised in our assumptions. Here is the code.

```
L.geoJSON(cities, {
    pointToLayer: function (feature, latlng) {
      if (feature.properties.Population <= 250000) {
        return L.marker(latlng, {
          icon: L.AwesomeMarkers.icon({icon: 'cog', prefix: 'fa', markerColor: 'purple
        });
      } else if (feature.properties.Population <= 800000) {
        return L.marker(latlng, {
          icon: L.AwesomeMarkers.icon({icon: 'coffee', prefix: 'fa', markerColor: 'red
        });
      } else {
        return L.marker(latlng, {
          icon: L.AwesomeMarkers.icon({icon: 'shopping-cart', prefix: 'fa', markerColo
        });
      }

    }
  }).bindPopup(function (layer) {
    return `City: ${layer.feature.properties.City},<br>
    Population: ${layer.feature.properties.Population}`;
  }).addTo(map);
```

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/extra-markers.jpg"))
```



The custom markers are also clickable!

6.5 Image overlays

Sometimes an image can act as good a marker as any other displayed so far.
Overlaying images on a map is fairly easy, and if in a rush, here is the code:

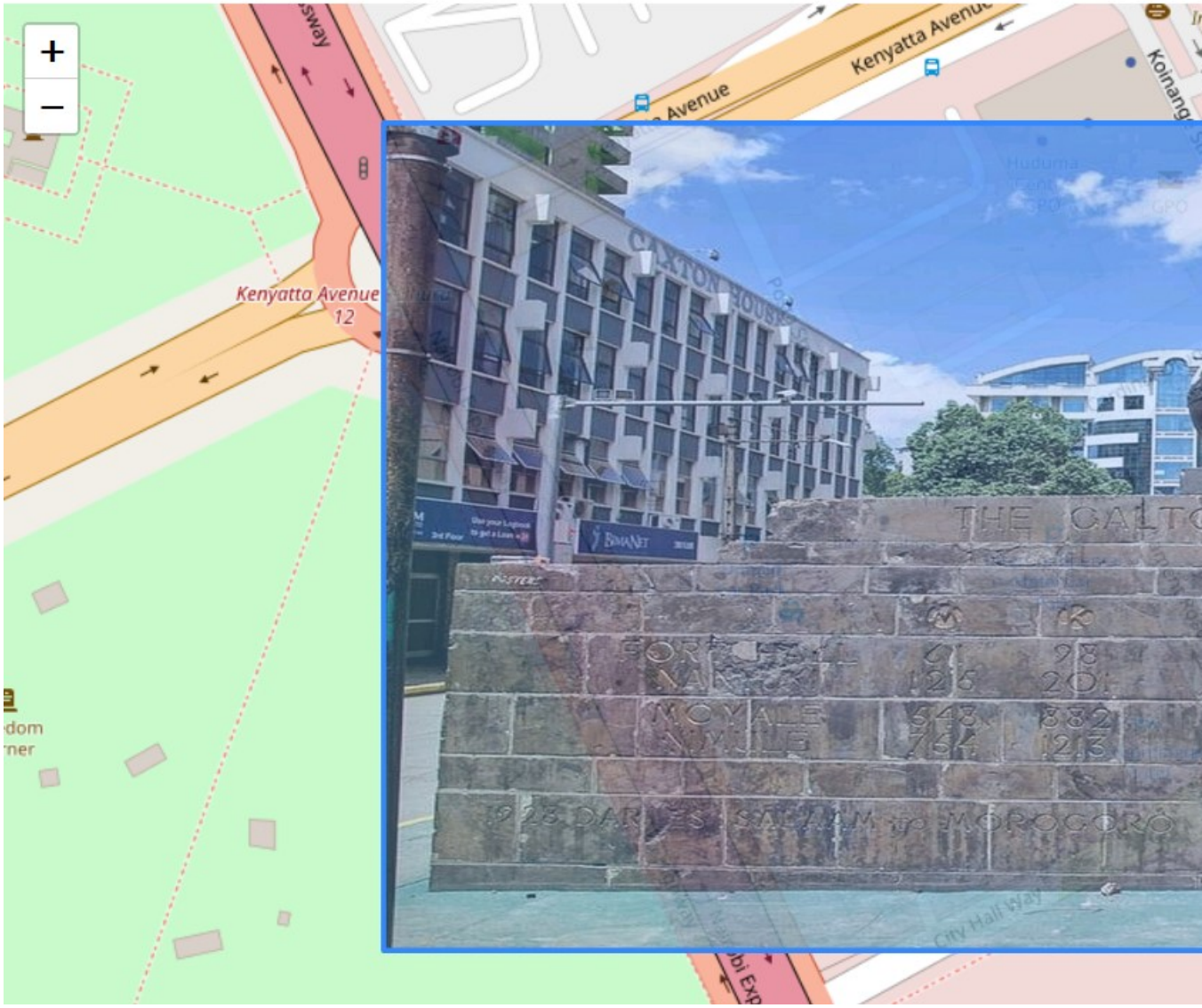
```
// Image overlays
var imageUrl = 'https://pbs.twimg.com/media/DddQBk5WsAAlbdJ?format=jpg&name=large';
var errorOverlayUrl = 'https://pbs.twimg.com/media/DddQBk5WsAAlbdJ?format=jpg&name=large';
var altText = 'The Galton - Fenzi Memorial: Source: Google and Twitter';
var latLngBounds = L.latLngBounds([[-1.2861259,36.8172709], [-1.2886193,36.8230413]]);

var imageOverlay = L.imageOverlay(imageUrl, latLngBounds, {
    opacity: 0.8,
```

```
    errorOverlayUrl: errorOverlayUrl,
    alt: altText,
    interactive: true
}).addTo(map);
```

However, find an image of one location over the wide earth can be tricky and tiring, so we envelope it with a rectangle plus use `map.fitBounds` to zoom to where our image is placed.

```
L.rectangle(latLngBounds).addTo(map);
map.fitBounds(latLngBounds);
```

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/geltan-fenzi.jpg"))
```



Mind you that landmark was set up in 1939 in honour of Lionel Douglas Galton-Fenzi who was the first motorist to drive from Nairobi to Mombasa when all that existed was dirt track laden with wild animals. Yours truly has also heard it contains bearings to various East African cities. Despite being at a point where I normally alight, I've never paid much attention to it nor its bearings. Next time, I will be keen.

Anyway, here is a quick breakdown of the attributes used in `L.imageOverlay`. We understand we have taken a lot of your time on so much a simple chapter, so we will be swift. The `var latLngBounds` uses the `L.latLngBounds` class to

set the lat-lon coordinates. Notice they are two coordinate lists bound with a single `[]`. Your brain will get used to this in time. If you don't enclose the two coordinates with `[]` an error may result. `var ImageUrl` is self explanatory –its the image source and for the rest, here they are:

1. `opacity` - defines the opacity of the image overlay, it equals to 1.0 by default. Decrease this value to make an image overlay transparent and to expose the underlying map layer.

2. `errorOverlayUrl` - is a URL to the overlay image to show in place of the overlay that failed to load.

3. `alt` - sets the HTML alt attribute to provide an alternative text description of the image. Alternative text is essential information for screen reader users. It can also benefit people during poor network connectivity, in the case the image fails to load. Moreover, it can improve the SEO of a website.

4. `interactive` - is false by default. If true, the image overlay will emit mouse events when clicked or hovered.

Chapter 7

Creating an interactive choropleth map

7.1 What is a choropleth map?

We will now move from markers to something larger than life –choropleth maps. What the heck are choropleth maps. Geographers will roll their over this term because they have come across almost throughout their career, but for the sake new readers, a choropleth map is a map whose geographical areas or regions are colored, shaded or patterned in relation to a data variable. If you have seen map that has disaggregated election results to states or provinces, and draped the winning candidate or party by color on the particular state, then that's a choropleth map. They mostly appear in election broadcasts, especially those of a national kind. In this chapter, we will create a choropleth map of Kenyan counties, and make it interactive by leveraging the area and population characteristics of each county.

7.2 Creating a choropleth map: the start

Obviously by now, without going into much details, you can now create a basic leaflet map blindfolded. Anyway, seeing is believing, so lets start what we have done several times over. Create another new JavaScript file (I know its repetitive, but enjoy it nevertheless) called `interactive-choropleth.js`.

In order to cheer you up with the monotonous opening statement in our many JavaScript files, we shall use a new tile layer.

```
var map = L.map('myMap').setView([0.3556, 37.5833], 6.5);
```

```
L.tileLayer('https://{s}.tile-cyclosm.openstreetmap.fr/cyclosm/{z}/{x}/{y}.png', {
    maxZoom: 20,
    attribution: '<a href="https://github.com/cyclosm/cyclosm-cartocss-style/releases"
}).addTo(map); // the CyclOSM tile layer available from Leaflet servers
```

Remember us mentioning that Leaflet has other tile layer servers? Here we have just used another, specifically CyclOSM tile layers.

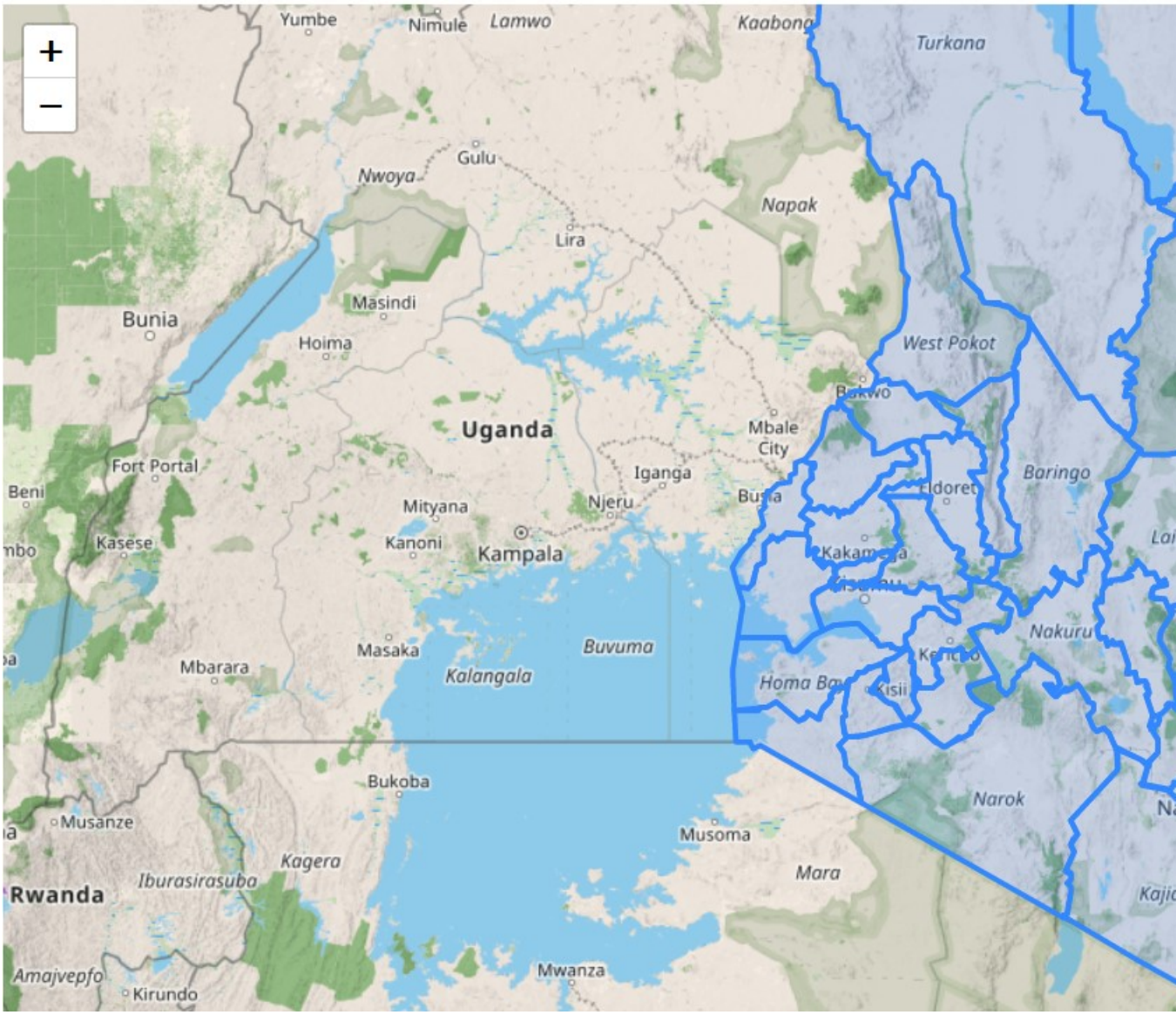
Now to the big part. The geojson files. We would like to mention it was quite a hustle to set up the geojson file in a manner accessible with JavaScript's Fetch API. Only when we converted the geojson to json, we were able to successfully view it using `fetch`. The raw json file for our Kenyan counties which we shall use in creating a choropleth map are available from here.

Let's fetch the counties json file and keep quiet.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/counties_json
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        L.geoJson(data, {style: style}).addTo(map);
    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

So far, you should get a result like below.

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/kenya_json.jpg"))
```



7.3 Coloring the counties

Alright, we have been able to load our json file to a Leaflet map. However, it looks dull and provides no meaningful information to the casual observation. When making maps, aim to provide information at lightning speed. That is, inform the reader at fast glance. The code snippets that follow have been heavily borrowed from Volodymyr's interactive choropleth tutorial.

First, let create a function that sets a color hex code for each population category. We used color brewer for this, as did Volodymyr in his tutorial.

Take a look at this code and we shall explain.

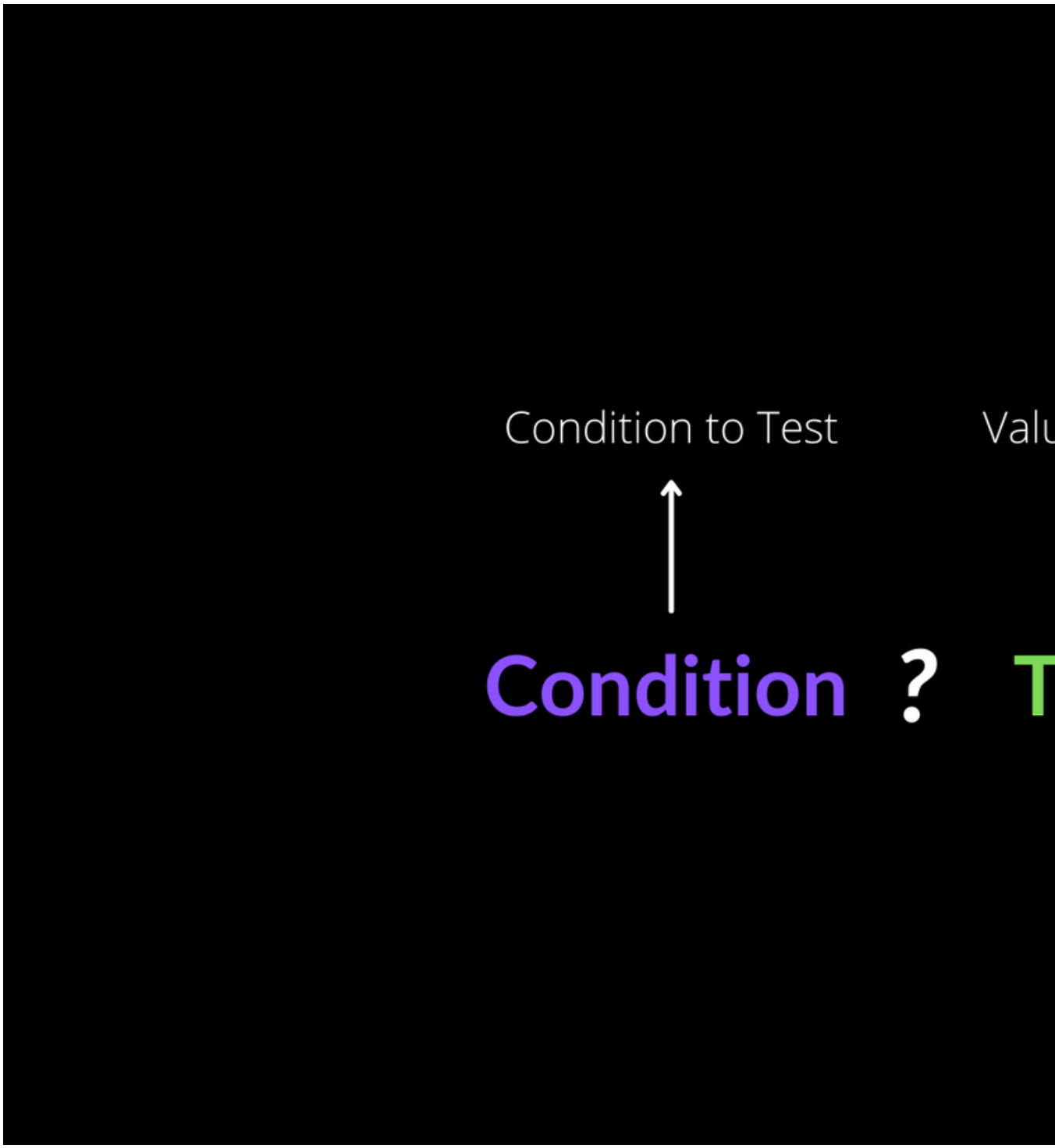
```
//// Adding some color
function getColor(d) {
    return d > 1400 ? '#8c2d04' :
            d > 700 ? '#cc4c02' :
            d > 400 ? '#ec7014' :
            d > 100 ? '#fe9929' :
            d >  50 ?  '#fec44f':
            d > 25 ?   '#fee391':
                       '#ffffd4';
}
```

The above function uses a question mark `?` to act as the if…else statement. In JavaScript terms, it is known as a ternary operator. What's that? How does it work? We can answer the second more arguably. For the former, we can only go as far as the definition.

How the ternary operator works is that any statement to the right of the `?` is returned as true if it agrees with the value to the left of the `?`. Reread that statement again. If the value to the right of the `?` is false, then the value to the right of the colon `:` is returned. Reread that last statement again.

If it still sounds fuzzy, the below image should help (adapted from FreeCodeCamp).

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/ternary.png"))
```


Condition to Test
Valu
Condition ? T

Now that we've created the function of setting colors to our json file on leaflet, we next also have to *create* a function that does to the GeoJSON file itself. Luckily, we have the **style** option from Leaflet which is styles GeoJSON lines and polygons using features from the GeoJSON file itself. We saw it in Chapter and we shall also use it here.

```
var style = ((feature)=> {
    return {
        fillColor: getColor(feature.properties.Pop_Density),
        weight: 2,
        opacity: 1,
        color: 'gray',
        fillOpacity: 0.5
    }
})
```

The above is an arrow function. Unlike regular JavaScript function declarations (function ()), we remove the `function` keyword, enclose everythin in brackets and put an arrow `=>` between the parameter brackets and function body. That's just it. Arrow functions aren't so hard!

Remember we assign the arrow function to a variable called `style` since we will parse it to the `L.geoJson` class. Just to keep things neat.

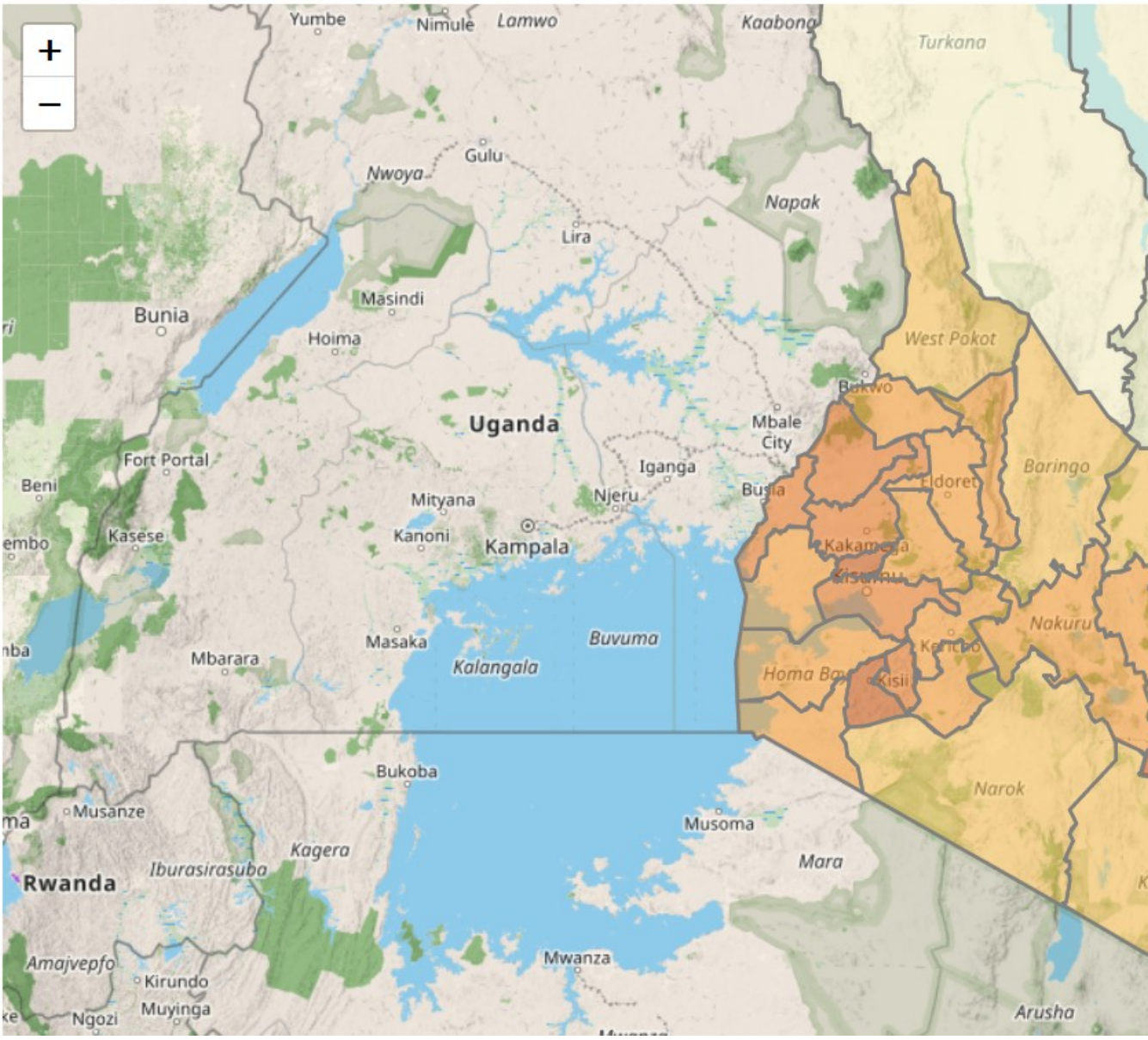
Finally, we add the `style` variable to the **style** option of `L.geoJson` class.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/counties_json
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        L.geoJson(data, {style: style}).addTo(map);
    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

Since the **style** option is a key (and also a function), the value will be the `var style` which we created. This value is in and by itself a function that iterates over every county because of the `getColor(feature.properties.Pop_Density)` contained in it as the `fillColor` value!

Enough JavaScript for one day!

```
knitr::include_graphics(rep("D:/gachuhi/my-leaflet/images/choropleth-map.jpg"))
```



Our choropleth map is beginning to take shape.

7.4 Highlight features

Going on from where we last left, we would like the choropleth map to highlight counties upon mouse hover, and also display their attributes upon click and zoom to the county! Quite a lot to work for. The counties should also be reset to their default characteristics upon mouse out.

Alright. It seems like we have our hands full.

Let's start simple.

Remember the `fetch` API we had used in retrieving our json file. We will tweak it a bit by adding the `var geojson` just before calling the `L.geoJson` class. We shall assing the variable `geojson` to the `L.geoJson` class. We know you are raising eyebrows. We shall explain why.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/counties_json
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        var geojson;
        geojson = L.geoJson(data, {
            style: style,
            --snip--
    })
```

We want upon hovering our mouse over each county, that the county is highlighted in white and appear to slightly 'pop' out above the rest. To do so, we insert the following code:

```
        geojson = L.geoJson(data, {
            style: style,
            onEachFeature: ((feature, layer) => {
                layer.on('mouseover', ((e) => {
                    var layer = e.target;

                    layer.setStyle({
                        weight: 5,
                        color: '#FFFFFF',
                        dashArray: '',
                        fillOpacity: 0.7
                    });

                    layer.bringToFront();
```

```
                })
```

The purpose of `on` method is to add an event listener. Event listeners in JavaScript are functions that run a code when the browser user interacts with the browser in a specific way. Now what `on` method does is that *Upon* each layer, we had an event listener known as `mouseover`. In our case, upon hovering a mouse pointer over our county we want the county hovered over to 'pop' out slightly and with a white border.

Since the change in state of an element in HTML is known as an event (denoted as `e` in our case), the `e.target` property returns the element on which the event is occurring on. Since it's a particular county in our case, we proceed to change its symbology through the parameters in `setStyle` function. Thereafter we use the `bringToFront` method to make the element in which the event has happened on to 'pop' out above the rest.

Remember we had mentioned we also want the counties to be reset to their default status when one hovers out to some other county or outside the map altogether. The following code does the job.

```
layer.on('mouseout', function() {
                    geojson.resetStyle(this);
                })
```

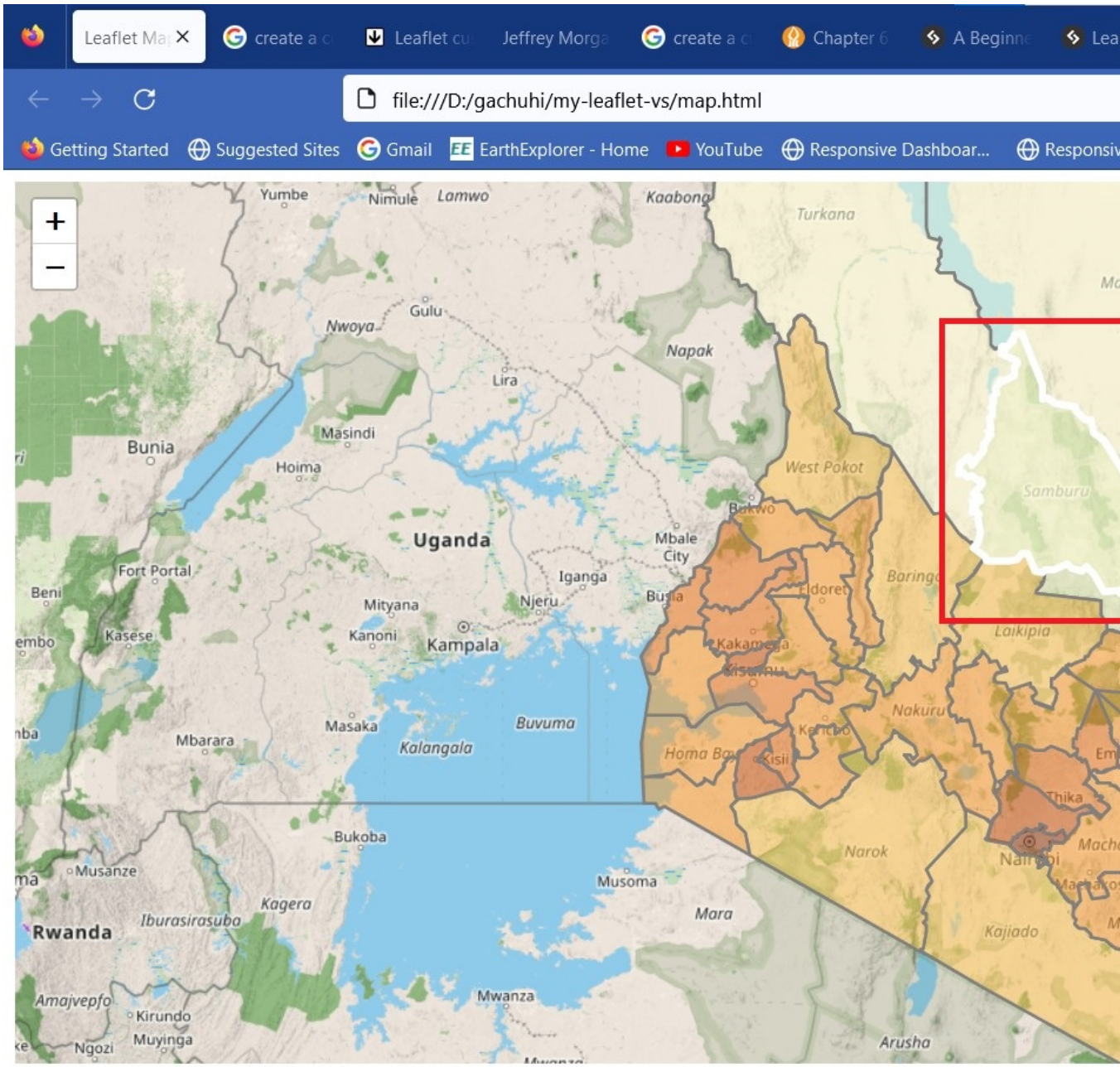
We use the `resetStyle` function to return the layer to their default but there is a twist. However, in this case, we add the argument `this` in parenthesis to refer to the element that was received. In other words, when the mouse 'hovers out' of a county, the element will revert to its original symbology. That particular element is parsed to `resetStyle` through `this` argument in parenthesis.

Before, we end this monologue, we ensure we pass the variable `geojson` to `resetStyle` function or else it won't work. This is how it is explained in the leaflet documentation as well as the *modus operandi* here and here.

Finally, we mentioned we want to zoom to a particular county upon clicking it. The following code fits our map to the bounds of a particular county that was clicked. Note that `fitBounds` is parsed `getBounds` which gets the boundaries of the county clicked upon, as referenced by `e.target`.

```
layer.on('click', ((e) => {
                    map.fitBounds(e.target.getBounds())
                }))
```

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/highlightable-map.jpg'))
```


Leaflet Ma
create a c
Leaflet cu
Jeffrey Morga
create a c
Chapter 6
A Beginne
file:///D:/gachuhi/my-leaflet-vs/map.html
Getting Started
Suggested Sites
Gmail
EarthExplorer - Home
YouTube
Responsive Dashboar...
Yumbe
Nimule
Lamwo
Kaabong
Turkana
Gulu
Nwoya
Napak
Lira
Masindi
Bunia
Hoima
West Pokot
Samburu
Bakwo
Uganda
Mbale
City
Fort Portal
Beni
Iganga
Busia
Eldoret
Mityana
Njeru
Kasese
Kanoni
Kampala
Laikipia
Nakuru
Masaka
Buvuma
Mbarara
Kalangala
Kisii
Thika
Bukoba
Narok
Musanze
Musoma
Kagera
Mara
Rwanda
Iburasirasuba
Kajiado
Amajvepfo
Mwanza
Kirundo
Muyinga
Ngozi
Arusha

Your code within the `fetch` API should look like this.

```
fetch("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/counties_json
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        var geojson;
        geojson = L.geoJson(data, {
            style: style,
            onEachFeature: ((feature, layer) => {
                layer.on('mouseover', ((e) => {  // highlight county on mouse hover
                    var layer = e.target;

                    layer.setStyle({
                        weight: 5,
                        color: '#FFFFFF',
                        dashArray: '',
                        fillOpacity: 0.7
                    });

                    layer.bringToFront();
                }))

                layer.on('mouseout', function () { // return to original symbology upo
                    geojson.resetStyle(this);
                })

                layer.on('click', ((e) => {  // Zoom to county upon clicking it
                    map.fitBounds(e.target.getBounds())
                }))

            })
        }).addTo(map);
    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

Such a monolithic code.

7.5 Creating a custom info

I hope you took a break. If not, take a one good look at the above image for some soothing effects for the sorrow begotten so far.

The following code adds a custom info control to the map. Think of a control as an UI element that allows interactivity with the map. Our custom control info shall provide details of the name, total population and population density.

```
// Add control
var info = L.control();

info.onAdd = function (map) {
    this.div = L.DomUtil.create('div', 'info');
    this.update();
    return this.div;
};

// Method that we will use to update the control based on feature properties passed
info.update = function (props) {
    this.div.innerHTML = '<h4>Kenya Population Density</h4>' + (props ?
        '<b>' + props.ADM1_EN + '</br><br />' + 'Total Population' + '<br>' + props.County_pop +
        props.Pop_Density + ' people / km<sup>2</sup>': 'Hover over state')
};

info.addTo(map);
```

That's both scary and cool at the same time. I want you to have the last. When thrown an incomprehensible code, acknowledge the complexity, and think of it as a cool graphic (or graffiti)!

Okay. Let's go through the above code bit by bit as best as we (hopefully!) can.

`var info = L.control();`

The above creates a variable `info` that holds the base class `L.control()` for all map controls. For example, `L.control.zoom` creates a zoom control in the map.

```
info.onAdd = function (map) {
    this.div = L.DomUtil.create('div', 'info');
    this.update();
    return this.div;
};
```

Returns the DOM element for the control and creates a `<div>` with class `info`. This is done through the help of `L.DomUtil` which, according to the Leaflet

website, provides utility functions to work with the DOM[1]. Actually, this new `<div>` of class `info` is created when you fire up your browser but obviously not in your actual `map.html` file.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/div-info-element.jpg'))
```

Trust me it wasn't there before. What `this.update()` does is return the update function that follows with the properties for each county, and then returns it.

```
// Method that we will use to update the control based on feature properties passed
info.update = function (props) {
    this.div.innerHTML = '<h4>Kenya Population Density</h4>' + (props ?
        '<b>' + props.ADM1_EN + '</br><br />' + 'Total Population' + '<br>' + props.Cou
        props.Pop_Density + ' people / km<sup>2</sup>': 'Hover over state')
};

info.addTo(map);
```

The above function updates the leaflet map with the name, population and population density for each county. This function is passed to the variable `info.update` and thereafter added to the map using the method `addTo`.

But there is a monster called the `this.div.innerHTML`. What this actually does is update the control based on the feature properties of each county hovered over. Actually, the purpose of `innerHtml` is to return the HTML content of an element and since our map is rendered in a HTML page, the features are returned as HTML.

Because the custom control info is a UI element, we need to set up its symbology in our CSS file. Paste the following to your `styles.css` file.

[1]The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web.

You may ask, "I thought I don't have to put CSS styles to Leaflet because it seems to already come already well packaged, style 'n all". I get your point, but remember we created a new `<div>` called `info` that appears when our browser is powered up, right? And since this `<div class="info ...">` must appear when the browser is powered up, CSS styles must be used to define its looks.

```
.info {
    padding: 6px 8px;
    font: 14px/16px Arial, Helvetica, sans-serif;
    background: white;
    background: rgba(255,255,255,0.8);
    box-shadow: 0 0 15px rgba(0,0,0,0.2);
    border-radius: 5px;
}

.info h4 {
    margin: 0 0 5px;
    color: #777;
}
```

7.6 Create a legend

Having created a custom control info, the following code creates the legend.

```
var legend = L.control({position: 'bottomright'});

legend.onAdd = function (map) {
    var div = L.DomUtil.create('div', 'info legend'),
    grades = [0, 25, 50, 100, 400, 700, 1400],
    labels = [];

    // loop through our density intervals and generate a label with a colored square for each int
    for (var i = 0; i < grades.length; i++) {
        div.innerHTML +=
            '<i style="background:' + getColor(grades[i] + 1) + '"></i> ' +
            grades[i] + (grades[i + 1] ? '&ndash;' + grades[i + 1] + '<br>' : '+');
    }
    return div;
}

legend.addTo(map);
```

Obviously the position of our legend is set using the `position` option in `L.control({position: 'bottomright'});`.

Apart from the other elements we discussed in our custom control info, we set the color interval of our legend through the `grades` array. The for loop that follows iterates through the grades array creating a color box for each interval.

Honestly, the for loop looks quite complicated and has been pasted as is from the leaflet choropleth tutorial. What `&ndash` brings back is a hyphen (-). That's how a hyphen is written in HTML. But anyway, getting back to it for we have to sound cool, the for loop essentially is create a range within each interval, such as 0 - 25, 25 - 50 and so on. This takes place after the `?` ternary operator which we discussed earlier. Once it reaches the end of the loop, that is it is out of the range, the `+` is appended to the last value from our `grades` array. This last bit is made possible due to the `:` which returns values that are false in JavaScript.

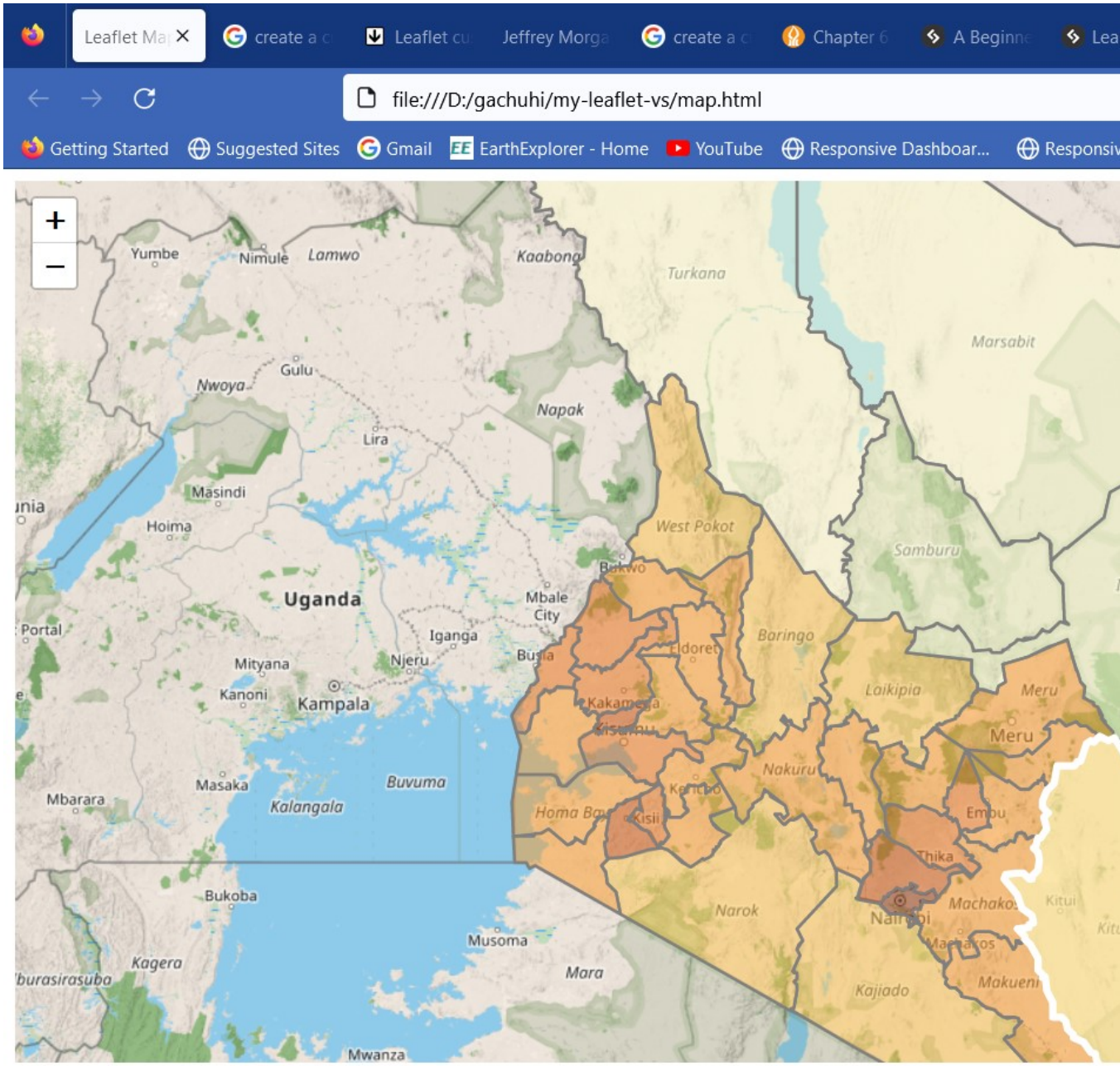
The legend also needs some CSS properties or else it will not appear. The `<div>` class of `info legend` is also created when the browser fires up. You may be wondering why `info legend` and not just `legend`. Well Mr/Mrs/Miss Golden Eye, the class properties of `info` are also inherited by those of legend such as background color and others. We also specify the CSS properites for the color intervals and text through the `.legend i`.

Here's the CSS anyway.

```
.legend {
    line-height: 18px;
    color: #555;
}
.legend i {
    width: 18px;
    height: 18px;
    float: left;
    margin-right: 8px;
    opacity: 0.7;
}
```

The legend is done and is finally added to the map.

```
knitr::include_graphics('D:/gachuhi/my-leaflet/images/choropleth-legend.jpg')
```


Leaflet Ma
create a c
Leaflet cu
Jeffrey Morga
create a c
Chapter 6
A Beginne
file:///D:/gachuhi/my-leaflet-vs/map.html
Getting Started
Suggested Sites
Gmail
EarthExplorer - Home
YouTube
Responsive Dashboar...
Yumbe
Nimule
Lamwo
Kaabong
Turkana
Marsabit
Gulu
Nwoya
Napak
Lira
Masindi
Hoima
West Pokot
Samburu
Uganda
Mbale City
Iganga
Busia
Eldoret
Baringo
Portal
Mityana
Njeru
Kanoni
Kampala
Laikipia
Meru
Nakuru
Masaka
Buvuma
Mbarara
Kalangala
Kisii
Embu
Thika
Bukoba
Narok
Machakos
Musoma
Kagera
Mara
Kajiado
Makueni
Mwanza

The full code files are available from here.

When creating choropleth maps, aim for challenging, impacting rather than merely informing.

Chapter 8

Layer groups and controls

8.1 Purpose of layer groups and controls

Sometimes, one may wish their webmap to consist of several baselayers or overlay maps. This not only add, counter, or enhance the information provided, but they also allow additional interactive features apart from the usual clicking, dragging and zooming. Suppose you want your leaflet to have two basemap layers, with the option of switching to either, and same case to overlays, how would you proceed?

8.2 Set up the basemaps

In order to create controls, we have to set the variables in JavaScript Objects. Thereafter, the `L.controls.layers` class is used to parse the object values to the Leaflet map and show a UI control. To demonstrate this, open a new JavaScript file and name it `groups_controls.js`. Insert the following code which will save our basemaps to the respective variables of `osm` and `cyclOSM`.

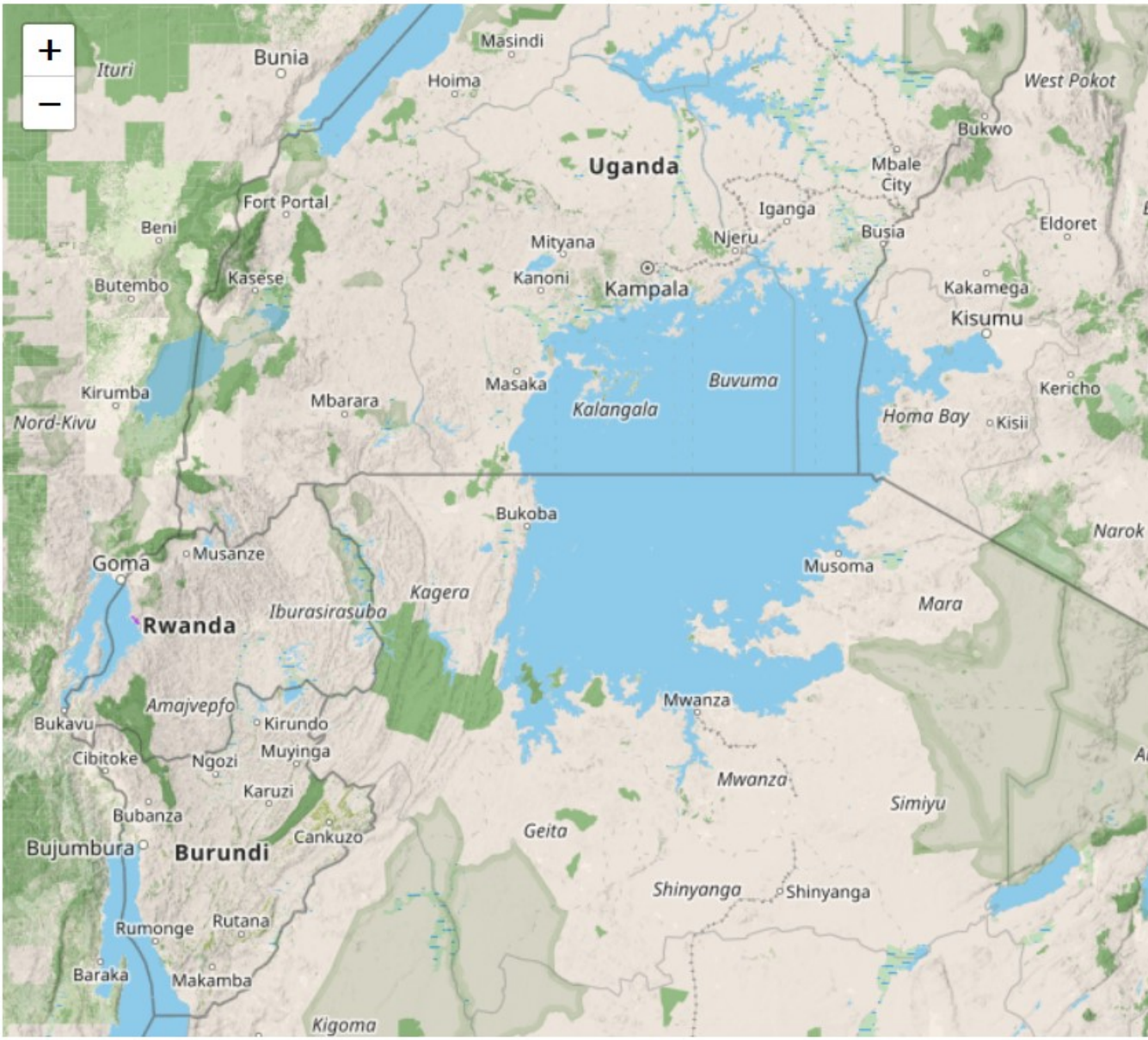
```
var osm = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
});

var cyclOSM = L.tileLayer('https://{s}.tile-cyclosm.openstreetmap.fr/cyclosm/{z}/{x}/{y}.png', {
    maxZoom: 20,
    attribution: '<a href="https://github.com/cyclosm/cyclosm-cartocss-style/releases" title="Cyc
}); // the CyclOSM tile layer available from Leaflet servers
```

We will pass the above two variables of `osm` and `cyclOSM` to the `L.map` class which has an option of `layers` in which one can parse the layers they wish to be displayed on the map.

```
var map = L.map('myMap', {
    layers: [osm, cyclOSM]
}).setView([-1.295287148, 36.81984753], 7);
```

However, that will only add the first basemap variable that appears, that is of `osm` and blocking out that of `cylcOSM`. This is shown below.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/no-control.jpg'))
```


+
−
Ituri
Bunia
Masindi
Hoima
West Pokot
Bukwo
Uganda
Mbale
City
Iganga
Busia
Eldoret
Fort Portal
Beni
Mityana
Njeru
Butembo
Kasese
Kanoni
Kampala
Kakamega
Kisumu
Kirumba
Masaka
Buvuma
Kalangala
Kericho
Nord-Kivu
Mbarara
Homa Bay
Kisii
Bukoba
Narok
Goma
Musanze
Musoma
Kagera
Iburasirasuba
Mara
Rwanda
Amajvepfo
Mwanza
Bukavu
Kirundo
Muyinga
Cibitoke
Ngozi
Mwanza
Karuzi
Simiyu
Bubanza
Geita
Cankuzo
Bujumbura
Burundi
Shinyanga
Shinyanga
Rumonge
Rutana
Baraka
Makamba
Kigoma

8.3 Creating the controls

However, in order to give `cyclOSM` a fair chance, we need to store them in an object say `var basemaps` and parse it to `L.controls.layer` which shall create a checkbutton for each basemap. The below code does just that.

```
// Set object for the basemaps
var basemaps = {
    "OpenStreetMap": osm,
    'cycleOsm': cyclOSM,
}

L.control.layers(basemaps).addTo(map);
```

This is what you get as a result.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/controls.jpg'))
```

8.4 Adding overlay maps

Now we have seen how to add more than one basemap and make all of them appear in the layer control. As was the case for the `basemaps` variable, it can also be replicated for the overlay maps as well.

The first overlay we would like to create is a choropleth map display the percentage of conventional households with access to main sewers as per the 2019 census. For simplicity purposes and to bypass errors we faced, we shall reuse the Ajax plugin for fetching GeoJSON files from online servers. As a reminder, we load Ajax into leaflet by insertint the following `<script>` tags into our `map.html`.

```
<script src="leaflet-ajax-gh-pages\dist\leaflet.ajax.js"></script>
```

```
<script src="leaflet-ajax-gh-pages\dist\leaflet.ajax.min.js"></script>
        <script src="leaflet-ajax-gh-pages\example\leaflet.spin.js"></script>
        <script src="leaflet-ajax-gh-pages\example\spin.js"></script>
```

I initially tried to load the json file for the sanitation overlay map layer using old `fetch` but the errors were too complex to solve. As they say, when life gives you bees, make honey, we decided to safely retreat to the young turk Ajax plugin. The following chunks of code will add the color function and styling for our countrywide sanitation map.

```
//// Adding some color
function getColor(d) {
    return d > 20 ? '#3288bd' :
            d > 10 ? '#99d594' :
            d > 6 ? '#e6f598' :
            d > 4 ? '#fee08b' :
            d >  2 ? '#fc8d59':
                     '#d53e4f';

}

// Function for setting color (using arrow function)
var style = ((feature)=> {
    return {
        fillColor: getColor(feature.properties.Human_waste_disposal),
        weight: 2,
        opacity: 1,
        color: 'gray',
        fillOpacity: 0.9
    }
})
```

Now let's add the overlay map that will display the accessibility to main sewer sanitation services. Spoiler alert: the statistics are grim.

```
// Adding the first overlay - map of household access to main sewer
var countySanitation = new L.geoJson.ajax("https://raw.githubusercontent.com/sammigachu
    style: style
})
.bindPopup(function (layer) {
            return `<b>County Name: </b> ${layer.feature.properties.ADM1_EN} <br>
            <b>Total County Population: </b><br>
            ${layer.feature.properties.County_pop.toString()} <br><br>
            <b>% of Conventional Households with access to main sewer: </b><br><br>
```

```
            ${layer.feature.properties.Human_waste_disposal.toString()}`
        }).addTo(map);
```

Let's add an accompanying legend for the above map. If you did the last Chapter of interactive choropleths, this should not be new.

```
// Create a legend
var legend = L.control({position: 'bottomright'});

legend.onAdd = function (map) {
    var div = L.DomUtil.create('div', 'info legend'),
    grades = [0, 2, 4, 6, 10, 20],
    labels = [];

    // loop through our density intervals and generate a label with a colored square for each int
    for (var i = 0; i < grades.length; i++) {
        div.innerHTML +=
            '<i style="background:' + getColor(grades[i] + 1) + '"></i> ' +
            grades[i] + (grades[i + 1] ? '&ndash;' + grades[i + 1] + '<br>' : '+');
    }
    return div;
}

legend.addTo(map);
```

Just a stick and paste.

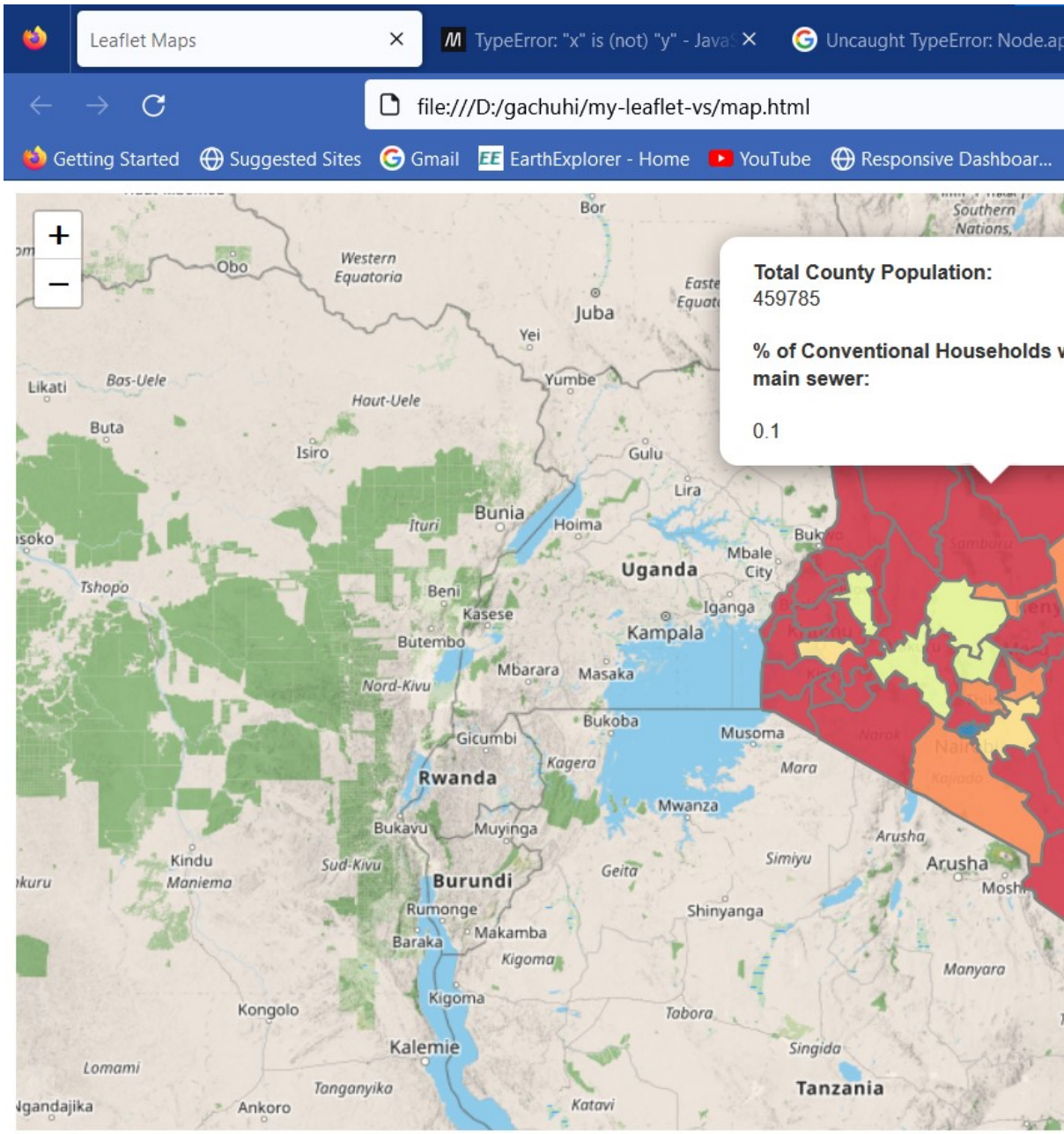
Now set an object to hold on of our two overlay maps. Don't worry, the second will come in no time.

```
var overlays = {
    'countySanitation': countySanitation,
}
```

Finally parse it to the `L.controls.layer` class.

```
// Add layer controls
L.control.layers(basemaps, overlays).addTo(map);
```

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/controls-overlay.jpg'))
```


Leaflet Maps
TypeError: "x" is (not) "y" - Java
Uncaught TypeError: Node.ap
file:///D:/gachuhi/my-leaflet-vs/map.html
Getting Started
Suggested Sites
Gmail
EarthExplorer - Home
YouTube
Responsive Dashboar...
Total County Population:
459785
% of Conventional Households v
main sewer:
0.1
Bor
Southern Nations,
Obo
Western Equatoria
Juba
Yei
Yumbe
Likati
Bas-Uele
Haut-Uele
Buta
Isiro
Gulu
Lira
Bunia
Ituri
Hoima
Mbale City
Uganda
Tshopo
Beni
Kasese
Iganga
Kampala
Butembo
Mbarara
Masaka
Nord-Kivu
Bukoba
Musoma
Gicumbi
Kagera
Mara
Rwanda
Mwanza
Bukavu
Muyinga
Arusha
Kindu
Maniema
Sud-Kivu
Simiyu
Geita
Burundi
Shinyanga
Rumonge
Makamba
Baraka
Kigoma
Manyara
Kongolo
Tabora
Kalemie
Singida
Lomami
Tanganyika
Tanzania
Ankoro
Katavi

If you click anyone of the counties, you will see popups appear.

There is one more overaly we will add to our display to bring our experimentation with layer controls full circle. Remember the geojson of our `cities` variable? Let's call it to action again. Load the custom icon markers first that will differentiate the populations of our cities.

```
// Color icons
// Yellow Icon
var yellowIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/img/marker-i
    shadowUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/images/marker-shadow.png',
    iconSize: [25, 41],
    iconAnchor: [12, 41],
    popupAnchor: [1, -34],
    shadowSize: [41, 41]
  });

// Orange Icon
var orangeIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/img/marker-i
    shadowUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/images/marker-shadow.png',
    iconSize: [25, 41],
    iconAnchor: [12, 41],
    popupAnchor: [1, -34],
    shadowSize: [41, 41]
  });

// Red Icon
var redIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/img/marker-i
    shadowUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/images/marker-shadow.png',
    iconSize: [25, 41],
    iconAnchor: [12, 41],
    popupAnchor: [1, -34],
    shadowSize: [41, 41]
  });
```

Let's load the `cities` geojson.

```
var cities = L.geoJson.ajax("https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/ci
    pointToLayer: function (feature, latlng) {
            if (feature.properties.Population <= 250000) {
              return L.marker(latlng, {
                icon: yellowIcon
              });
```

```
            } else if (feature.properties.Population <= 800000) {
              return L.marker(latlng, {
                icon: orangeIcon
              });
            } else {
              return L.marker(latlng, {
                icon: redIcon
              });
            }

          }
}).bindPopup(function (layer) {
      return `City: ${layer.feature.properties.City},<br>
      Population: ${layer.feature.properties.Population}`;
    }).addTo(map);
```

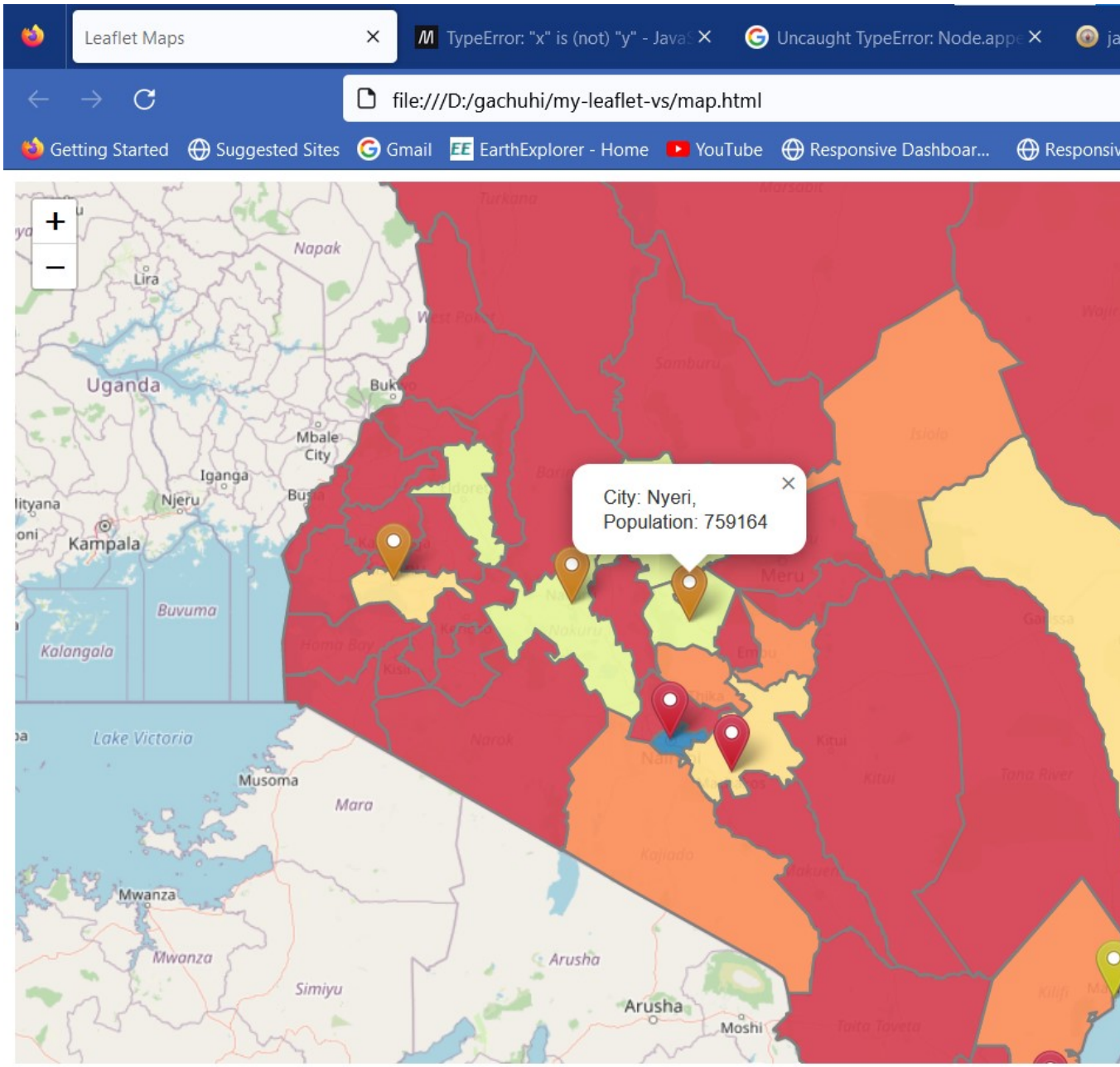
Add the `cities` variable as one of the keys to our `overlays` variable and finally parse the overlays object to `L.control.layers`.

```
// Set object for the overlay maps
var overlays = {
    'countySanitation': countySanitation,
    'cities': cities
}

// Add layer controls
L.control.layers(basemaps, overlays).addTo(map);
```

In the previous chapter, we ended by saying that we strive to make our choropleths map challenge, charge rather than merely informing. This seems a sketchy type, but it shows the discrepancy as well as the effort needed for Kenya to bring a centralized sanitation system to every household. Sanitation matters may always sound insensitive, but the true barometer of any civilization is its sanitation.

Finally, we tried playing around with the options available to the `L.controls.layer` including the simple one as changing position, but everytime a new option was provided in to the parenthesis, as in `L.control.layers(basemap, overlaymap, {options})`, the control would disappear.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/controls-all.jpg'))
```

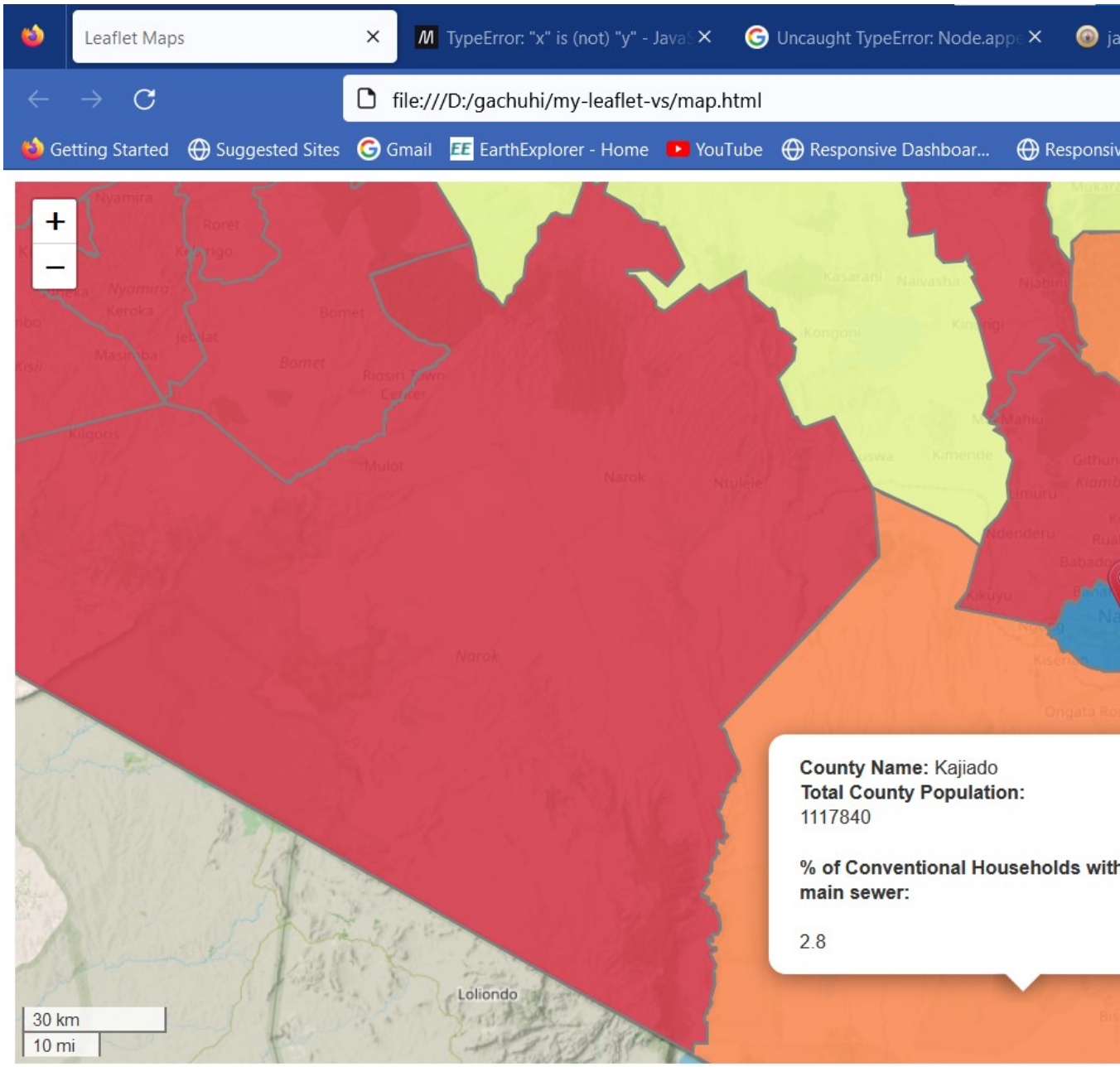

Leaflet Maps
TypeError: "x" is (not) "y" - Java
Uncaught TypeError: Node.app
file:///D:/gachuhi/my-leaflet-vs/map.html
Getting Started
Suggested Sites
Gmail
EarthExplorer - Home
YouTube
Responsive Dashboar...
Lira
Napak
Uganda
Mbale
City
Iganga
Njeru
Kampala
Buvuma
Kalangala
Lake Victoria
Musoma
Mara
Mwanza
Mwonza
Simiyu
Arusha
Moshi
City: Nyeri,
Population: 759164
Meru

8.5 Add a scale bar

Adding a small reactive scale with an option to demonstrate will not hurt!

```
// Add scale
L.control.scale({position:'bottomleft'}).addTo(map);
```

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/controls-scale.jpg'))
```


Leaflet Maps
TypeError: "x" is (not) "y" - Java
Uncaught TypeError: Node.app
file:///D:/gachuhi/my-leaflet-vs/map.html
Getting Started
Suggested Sites
Gmail
EarthExplorer - Home
YouTube
Responsive Dashboar...
Responsiv
+
−
Nyamira
Roret
Keroka
Nyamiro
Bomet
Kasarani
Naivasha
Kongoni
Mulot
Narok
Ntulele
Kilgoris
Loliondo
County Name: Kajiado
Total County Population:
1117840
% of Conventional Households with
main sewer:
2.8
30 km
10 mi

All the files and scripts used in this chapter can be accessed here.

Chapter 9

Heatmaps

9.1 What are heatmaps?

Heatmaps are a type of maps that geographically visualize locations with patterns of higher than average occurrence of particular variables say crime, disease, service centres and et cetera.

9.2 Loading the heatmap plugin

A tool for drawing heatmaps does not come prepackaged in leaflet. We therefore have to use an external plugin by the name of Leaflet.heat. Yours truly downloaded it from the site hyperlinked in the aforementioned sentence. In case you missed it, the link is here. Download and extract the folder in the same directory as your `map.html`.

Once you have extracted, open your `map.html` and add another `<script>` tag within your `<head>` element.

```
<script src="Leaflet.heat-gh-pages\Leaflet.heat-gh-pages\dist\leaflet-heat.js"></script>
```

Thereafter, we will have to think of a way of loading the GeoJson to our `heatmap.js` file, which we shall use as the JavaScript file for our heatmap code. You may thin of using `fetch` or `L.geoJson.ajax` for this purpose but not so fast. Yours truly tried all the above and even though they are proven ways of loading a JavaScript file as we have seen so far it didn't work in creating a heatmap. This is even when we stored the `fetch` and `L.geoJson.ajax` functions into variables to pass them to `L.heatLayer`. `L.heatLayer` is a function that is made possible from the Leaflet.heat plugin we parsed into `map.html` above.

So, what method works in creating a heatmap?

After much internet searching, I came across a code in Stack Overflow. However, it only works after inserting another Ajax plugin. Luckily no download is needed so insert this new `<script>` into your `map.html`.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script
```

9.3 Creating the Leaflet heatmap

Let's call the usual suspects of adding a basemap.

```
var map = L.map('myMap').setView([0.3556, 37.5833], 6.5);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMa
}).addTo(map);
```

Thereafter, we add the following large code chunks.

```
var geoJsonUrl = "https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/sel

    var geojsonLayer = $.ajax({
    url : geoJsonUrl,
    dataType : 'json',
    jsonpCallback: 'getJson',
    success : console.log("Data successfully loaded!"),
    });

geoJson2heat = ((geojson) => {
        return geojson.features.map(function(feature) {
        return [parseFloat(feature.geometry.coordinates[1]),
                parseFloat(feature.geometry.coordinates[0])];
        });
        });


$.when(geojsonLayer).done(function() {
        // var kill = L.geoJSON(geojsonLayer.responseJSON);
        var layer = geoJson2heat(geojsonLayer.responseJSON, 4);
        var heatMap = L.heatLayer(layer, {
                radius: 40,
                blur: 10,
                gradient: {
```

```
                        '0': 'Navy', '0.25': 'Navy',
                        '0.26': 'Green',
                        '0.5': 'Green',
                        '0.51': 'Yellow',
                        '0.75': 'Yellow',
                        '0.76': 'Red',
                        '1': 'Red'
                    },
            maxZoom: 13});
    map.addLayer(heatMap);
    });
```

To give credit where it is due, the code chunks were taken from this Stack Overflow question and modified a bit.

I will do everyone justice by going through the preceding code bit by bit.

```
var geoJsonUrl = "https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/selected_hosp

    var geojsonLayer = $.ajax({
    url : geoJsonUrl,
    dataType : 'json',
    jsonpCallback: 'getJson',
    success : console.log("Data successfully loaded!"),
    });
```

The `var geoJsonUrl` gets the link to our hospitals json file. The `var geojsonLayer` uses the Asynchronous Javascript And Xml (Ajax) method to load data from our Github server. If successfull, we get the message "Data successfully loaded!" in our browser console.

```
geoJson2heat = ((geojson) => {
        return geojson.features.map(function(feature) {
        return [parseFloat(feature.geometry.coordinates[1]),
                parseFloat(feature.geometry.coordinates[0])];
        });
        });
```

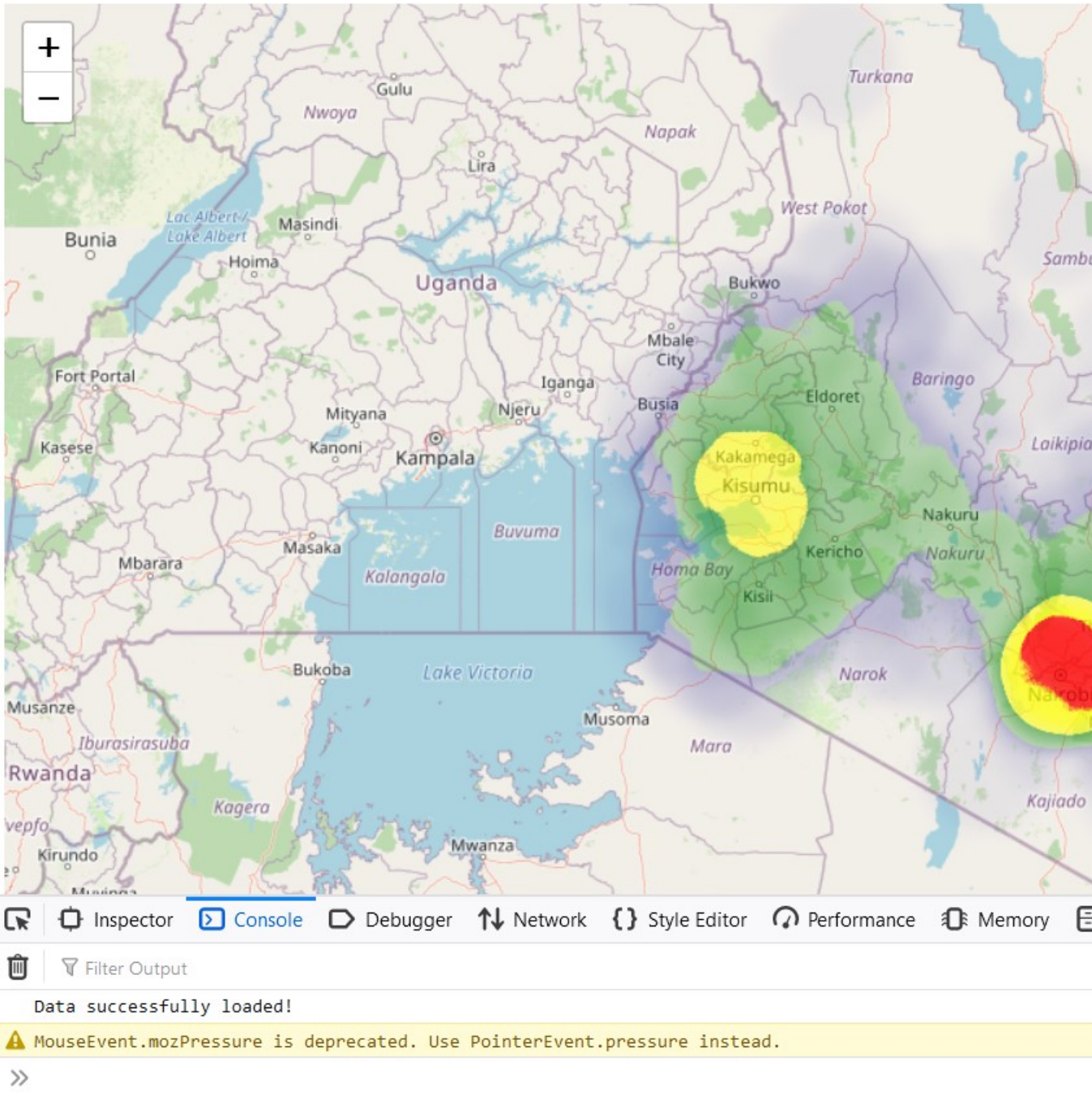
The above code chunk is fairly easy to understand. The function passed to `geoJson2heat` uses the `map` method to iterate over every element, such as cities in the json file and retrieve the longitude and latitude coordinates. Notice the format has been inverted. Rathater than [0]… [1] for Lat-Lon, we use [1]…[0].

The final is a jQuery function that will return the heatmap layer.

```
$.when(geojsonLayer).done(function() {
        // var kill = L.geoJSON(geojsonLayer.responseJSON);
        var layer = geoJson2heat(geojsonLayer.responseJSON, 4);
        var heatMap = L.heatLayer(layer, {
                radius: 40,
                blur: 10,
                gradient: {
                        '0': 'Navy', '0.25': 'Navy',
                        '0.26': 'Green',
                        '0.5': 'Green',
                        '0.51': 'Yellow',
                        '0.75': 'Yellow',
                        '0.76': 'Red',
                        '1': 'Red'
                      },
                maxZoom: 13});
        map.addLayer(heatMap);
        });
```

Honestly the last was a bit of a stretch as yours truly is yet to encounter use of jQuery that much nowadays.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/heatmap.jpg'))
```



The full code files are available from here.

Chapter 10

Cluster to reduce the clutter

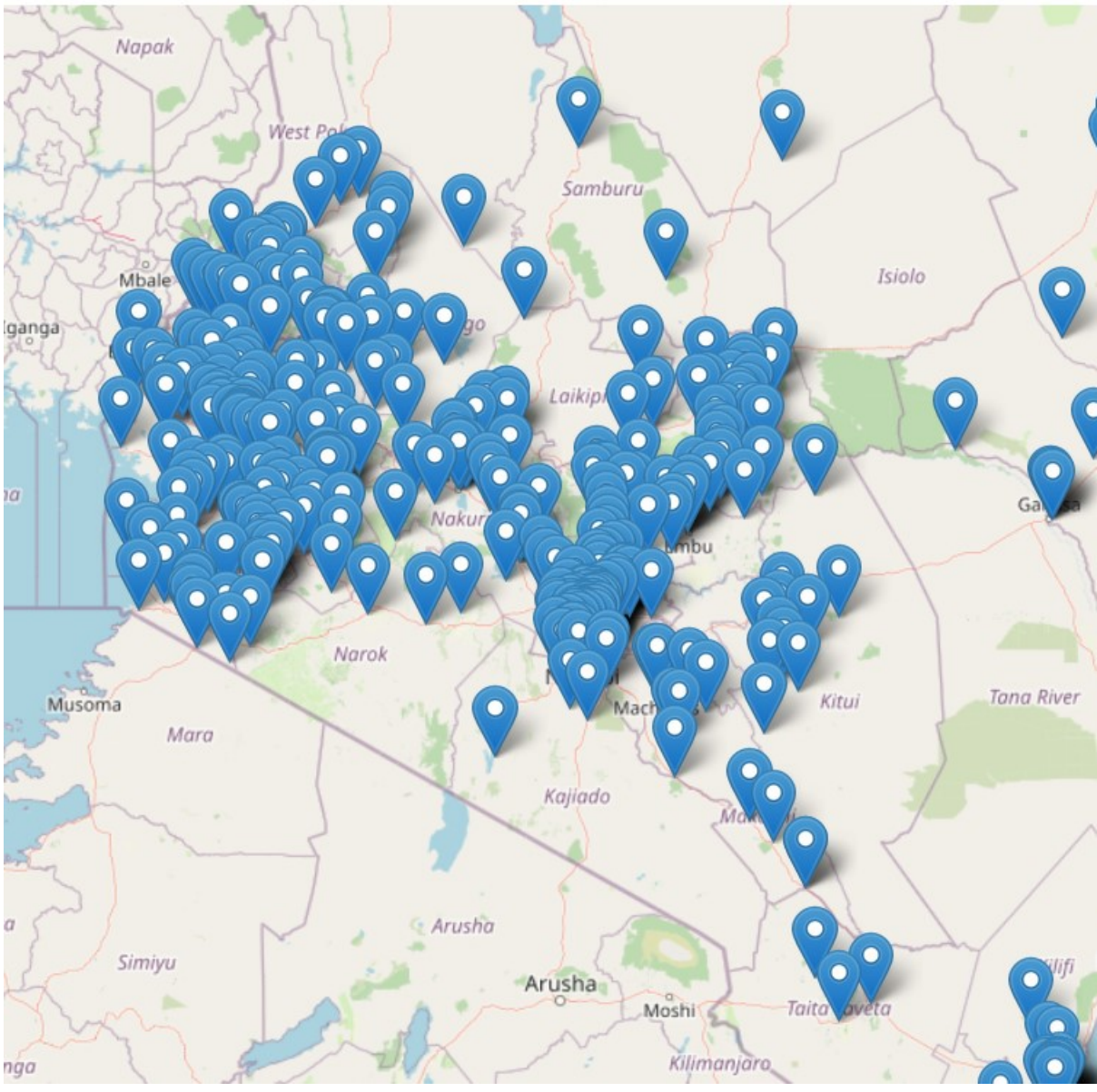
10.1 A map full of clutter

There comes a time when it is convenient to coalesce several points into single multi-cluster point. Consider the following example: you have a geojson file with over 10000 points of houses within a densely populated region, say an island of 5km by 5km, if by any chance such a scenario exists. Will you want to display such a gigantic number of points within such a small area? That would be an overkill! Moreover, it will seem incomprehensible to the viewer. Consider the following example we set up in a new JavaScript file which we fondly called `cluster-markers.js`.

```
var map = L.map('myMap').setView([-1.295287148, 36.81984753], 7);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);


url = 'https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/selected_hospitals.json'

L.geoJson.ajax(url).addTo(map);

var markers = L.markerClusterGroup();
```

The following is the result we got.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/clutter-points.jpg'))
```



Not good at all. The `Leaflet.markercluster` plugin is what transforms a clutter map into one of neatly arranged clustered marker points.

10.2 Preparations

Creating a cluster marker map is fairly easy. You will first have to insert the `Leaflet.markerCluster` plugin into `map.html`. The plugin is available from here. Insert the following `<script>` tag into the `<head>` element of your `map.html`.

```
<script src="Leaflet.markercluster-1.4.1\Leaflet.markercluster-1.4.1\dist\leaflet.markercluster.j
```

You will also have to insert the `Leaflet.markercluster` CSS properties via the `<link>` tag too. Add the following `<link>` tag for `Leaflet.markercluster` after the other `<link>` properties.

```
<link rel="stylesheet" href="Leaflet.markercluster-master\Leaflet.markercluster-master\dist\Marke
```

Don't underestimate them. These CSS properties are necessary to style your cluster points in a nice way -to make it easy on the eye.

Hope you also referenced your `map.html` to the `cluster-markers.js` in the `<script>` file inside the `<body>` element!

If you had done the small exercise at the beginning of the chapter, the following code should be present.

```
var map = L.map('myMap').setView([-1.295287148, 36.81984753], 7);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
}).addTo(map);


url = 'https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/selected_hospitals.json'
```

10.3 Behold, a cluster marker map!

Delete the `L.geoJson.ajax(url).addTo(map);`, we won't need it now. Our real work of creating a cluster marker map begins with the `markerClusterGroup`. Let's proceed!

```
var markers = L.markerClusterGroup();
```

We shall use Ajax again but this time round we shall pass in some functions to customize the appearance and functionalities of our GeoJson markers.

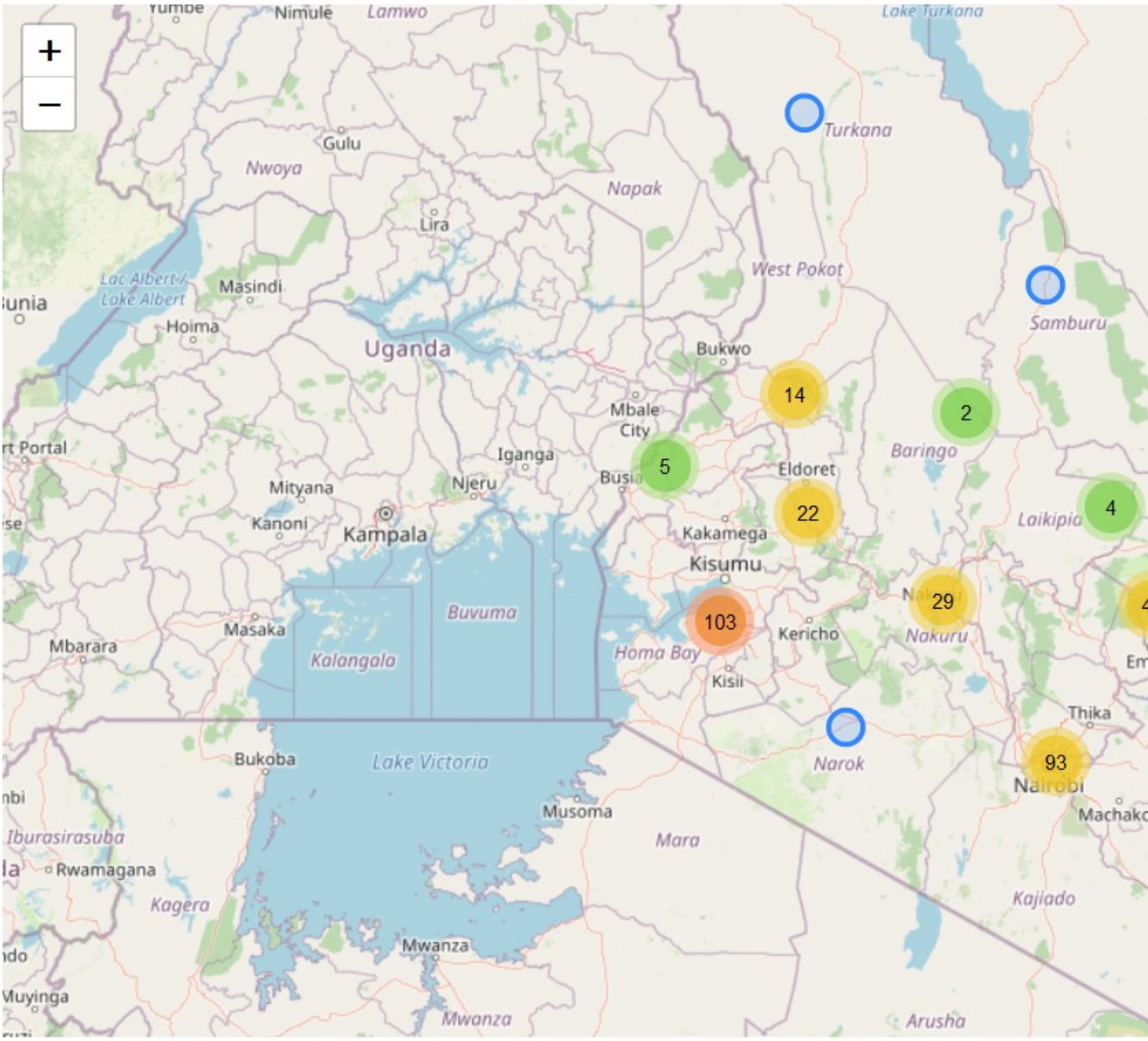
```
L.geoJson.ajax(url, {
    pointToLayer: ((feature, latLng) => {
        return markers.addLayer(L.circleMarker(latLng));
    }),

    onEachFeature: ((feature, layer) => {
        layer.bindPopup(`<b>Facility Name:</b> ${feature.properties.Facility_N} <br>
        <b>Type:</b> ${feature.properties.Type}`)
    })
}).addTo(map);
```

Remember `pointToLayer`? It defines how the GeoJson file will appear. The `pointToLayer` retrieves the Latitude-Longitude coordinates, creates circle markers out of them before finally parsing to the `markers` variable in the return keyword.

How about for `onEachFeature`. You can guess. *O*n *E*ach *F*eature, do this and that. In our case we bind a popup of facility name and type which will appear when a circle marker is clicked on.

Actually, the above code sort of finished the work for us.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/cluster-marker-map.jpg'))
```



Zoom in and out and watch the circle markers *spidefy* to individual points with a popup. I have restrained from tweaking because the defaults are already good enough. See them from this Github Page.

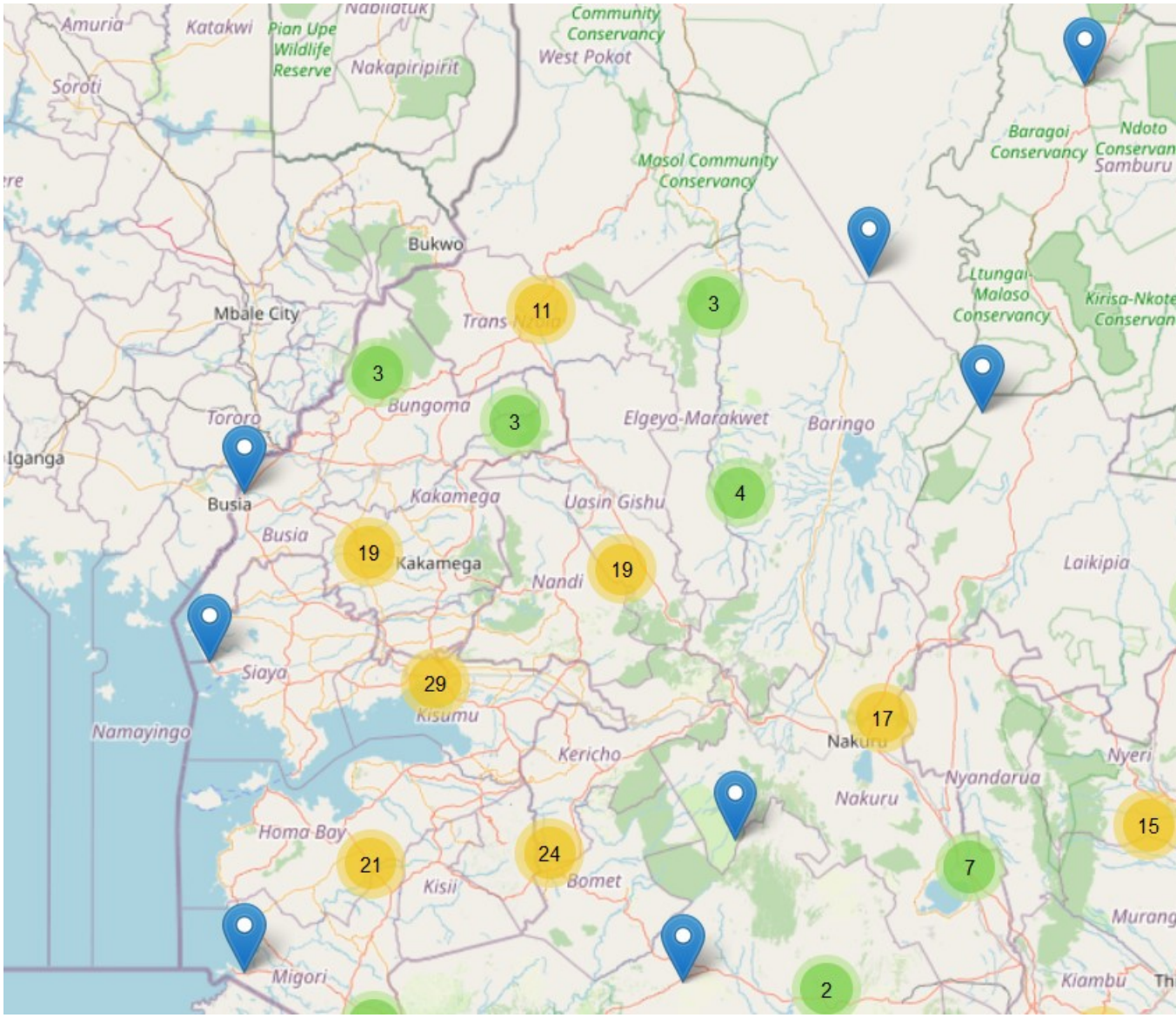
However, as good practice, we need to add the `markers` global variable to the map. We do so by using the following code.

```
map.addLayer(markers)
```

`addLayers`, just like the name suggests, adds the given layer to the map.

The `L.circleMarker` in the `pointToLayer` key of the ajax function can be replaced with `L.marker()`. However, lone hospitals especially to the northeast of Kenya, and when clusters are specified to their individual points will resort to the default marker style of Leaflet. Below is how they would look, and is far from aesthetic.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/cluster-marker-plain.jpg'))
```



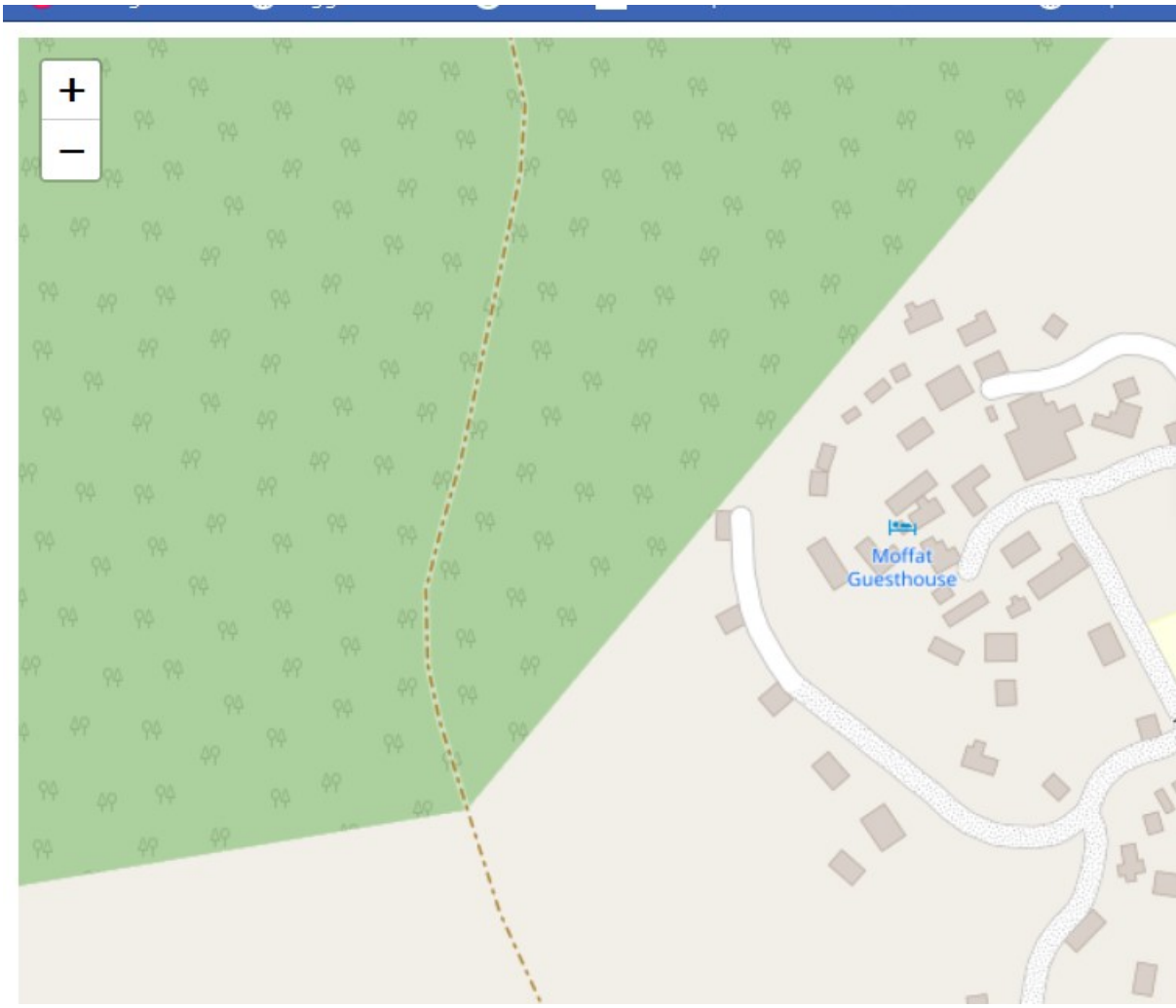
Coming back full circle, circle markers are way better.

Most of the ajax code was inspired by this video.

But wait! Hold your horses, there is a bug. Try to click on any of the spiderfied or lone hospitals and you will notice something that will raise eyebrows. All hospitals display the following popup when clicked:

```
Facility Name: Wajir Tb Manyatta Sub - District Hospital
Type: Sub-District Hospital
```

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/cluster-marker-bug.jpg'))
```



Unless a developer would like to be left with an egg on the face for assigning wrong place names, this should be dealt with expeditiously. Our code is alright, since it works in other scenarios such as here. Yours truly suspects that the Ajax plugin is the issue, since `L.geoJSON` works perfectly in displaying the correct popup for each hospital.

Time to try a different strategy: using `fetch` API. At least `fetch` will use `L.geoJSON()` as it is, without spoilers such as Ajax and each marker will display the correct popup.

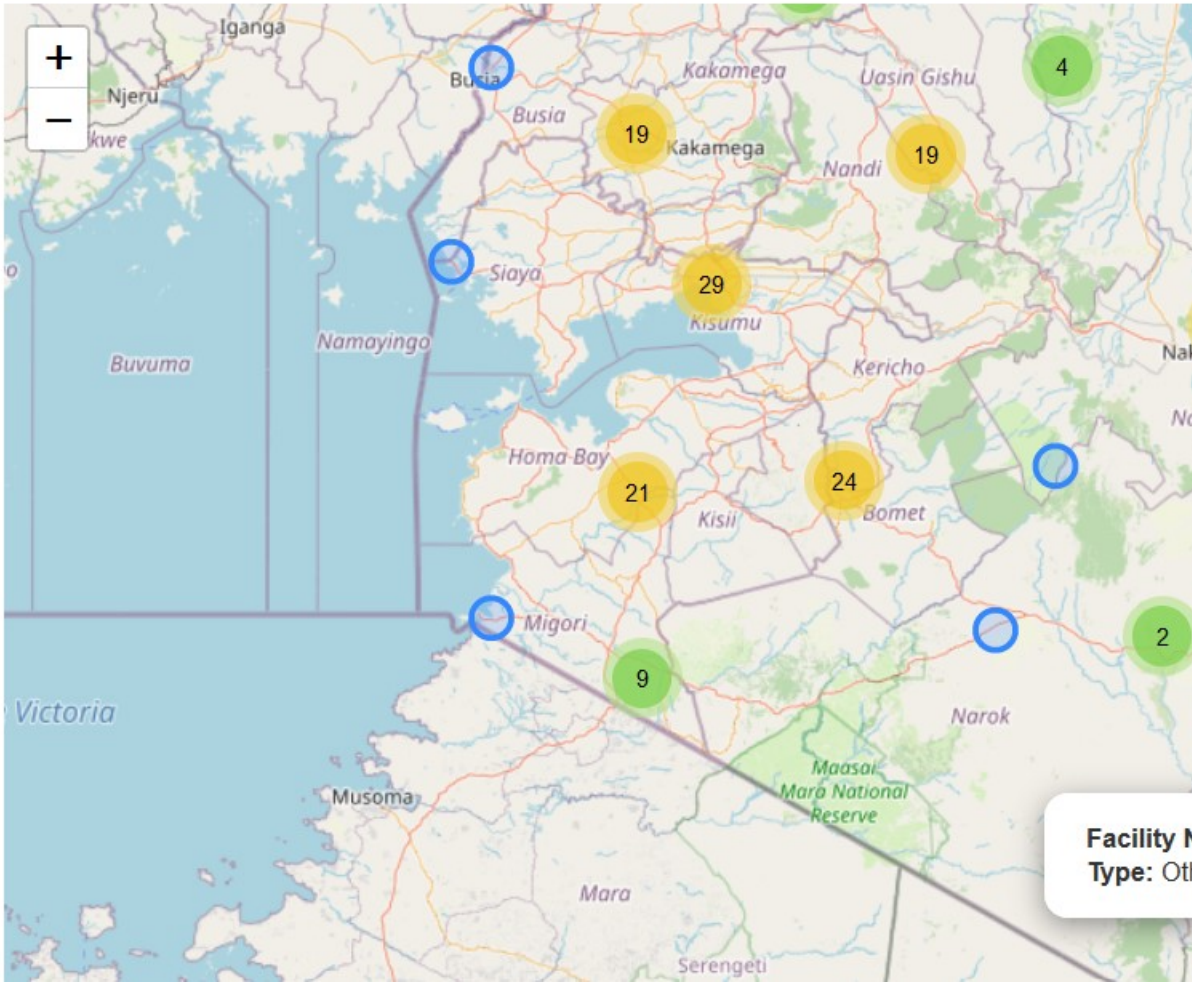
We have worked with `fetch` before so I will not explain it that much here. Feel

free to google a refresher.

Comment out the earlier code beginning from `var markers` to `map.addLayer(markers)` and replace it with the following code chunk.

```
fetch(url)
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        var markers = L.markerClusterGroup();

        var geojsonGroup = L.geoJSON(data, {
            onEachFeature : function(feature, layer){
                layer.bindPopup(`<b>Facility Name:</b> ${feature.properties.Facility_N} <br>
                <b>Type:</b> ${feature.properties.Type}`);
            },
            pointToLayer: function (feature, latlng) {
                return L.circleMarker(latlng);
            }
        });

        markers.addLayer(geojsonGroup);
        map.addLayer(markers);

    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/cluster-marker-map-fixed.jpg'))
```



Now all points have their rightful and respective names.

10.4 Additional features of Cluster marker plugin

The official documentation of the plugin lists many other features that come along with the tool. We can't go through all of them, neither is there the will but let's summise just one important one: the `mouseover` event. The same way that leaflet's markers can have events triggered on them also applies to the Cluster marker plugin. Below is an example in their webpage

```
// We won't use this
```

```
markers.on('click', function (a) {
    console.log('marker ' + a.layer);
});
```

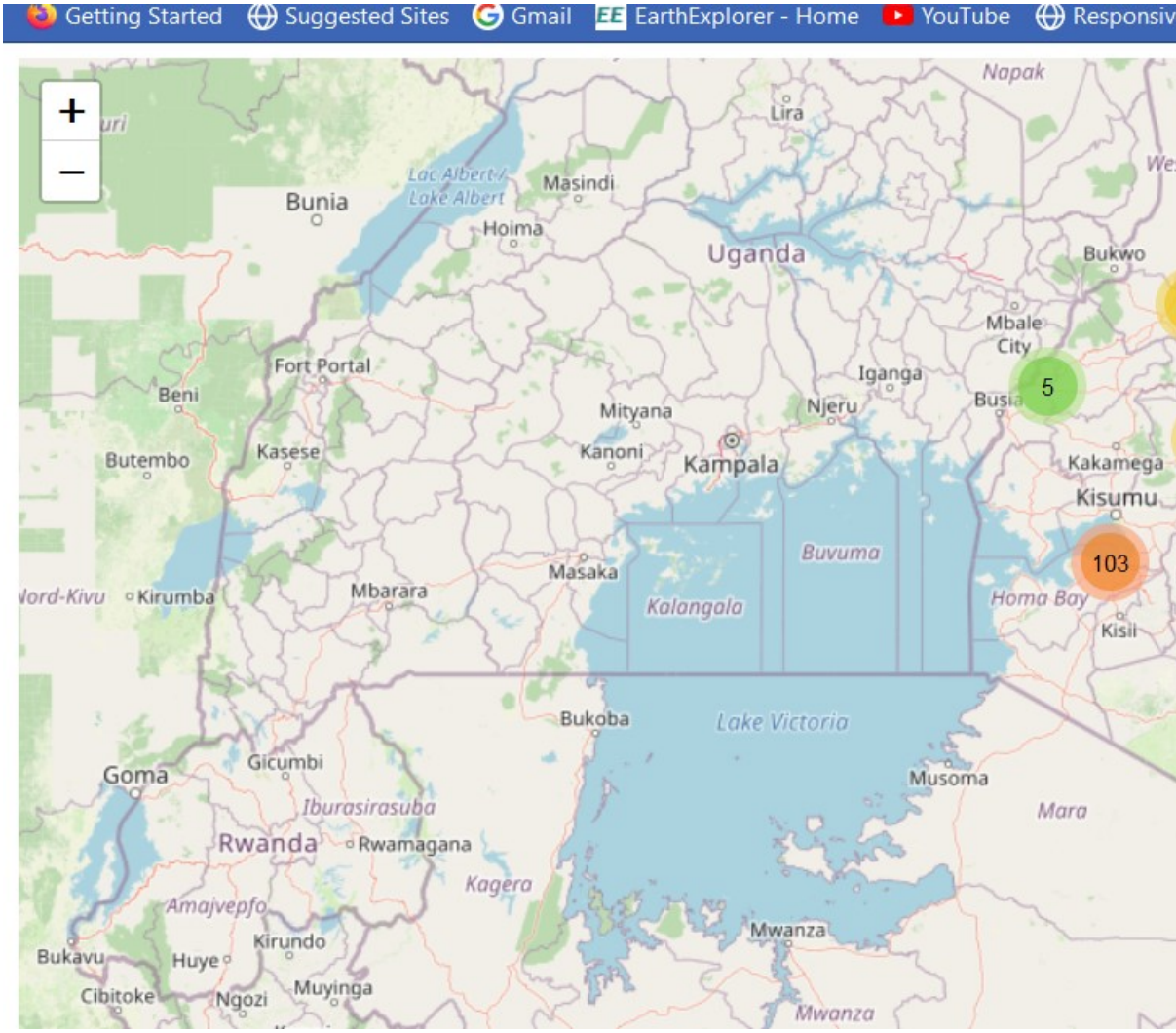
Let's demonstrate adding a hover event to our cluster marker map, thanks to this answer.

```
/////// Added `mouseover` event

fetch(url)
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        var markers = L.markerClusterGroup({chunkedLoading: true}); // Splits the add layers to s

        var geojsonGroup = L.geoJSON(data, {
            onEachFeature : function(feature, layer){
                layer.bindPopup(`<b>Facility Name:</b> ${feature.properties.Facility_N} <br>
                <b>Type:</b> ${feature.properties.Type}`);
            },
            pointToLayer: function (feature, latlng) {
                return L.circleMarker(latlng).on('mouseover', function(){
                    this.bindPopup(`Nearest_Center: ${feature.properties.Nearest_To}`).openPopup(
                });
            }
        });

        markers.addLayer(geojsonGroup);
        map.addLayer(markers);

    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

From the above code, we added the `mouseover` event to each marker point using the `on` method. This was thanks to `this` keyword which refers to the object in the current code chunk –`geojsonGroup`. We also sped the rendering of our Leaflet map by adding `chunkedLoading: true` parameter to `L.markerCluster()` function.

Hovering over each marker point will show an attribute of the centre in closest proximity from the `Nearest_To` properties of the Json Layer.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/cluster-marker-mouseover.jpg
```



The code files used for this chapter are available from here.

Chapter 11

Mobile Friendly Webapps

Short story. Not too long ago I was the proud owner of a famous phone brand on the decline. The phone was like an extension of my personality for I understood its shortcomings, while on the other hand, it perfectly blended, nearing understanding what I wanted and needed, and providing exactly that.

One time, when taking a photo of the iconic Ngong Hills for Wikipedia's Africa Climate photo contest, the phone just died. That was it. A quick visit to the authorized dealer was greeted with the unbelievable and bemusing words of, "We no longer ship the motherboard to the country anymore." Some healing has taken place, but I was totally heartbroken, and occasionaly suffer some nostalgia of the 'good times' I had with my phone.

It's been a long ride with leaflet, you need a breather. Anyway, webapps can be heavy, and they can load slowly on smaller devices such as smartphones. Apps that load slow can put off your web app users, so it is prudent to customize your webapp for your user's phones.

"Well, what the heck?", you may ask, "It will just test their patience for a couple few second..." Alright, I get it. Why the hustle to make the app mobile friendly just to appease someone's impatience. If you think that your user's will only be using your webapp on a laptop, you are wrong. If they won't find it friendly on their smartphone they may not think twice on using it on a laptop either.

Let's get on with making our cluster marker app mobile friendly. We shall also add other functionalities to make the web app heavy, and thus test our mobile-friendly web map making to destruction.

11.1 The basemaps

If you have gone through the Chapter 8 of creating controls the following will look familiar. We will add some basemaps and later on create their control widgets.

```
// Basemaps
var osm = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMa
});

var cyclOSM = L.tileLayer('https://{s}.tile-cyclosm.openstreetmap.fr/cyclosm/{z}/{x}/{
    maxZoom: 20,
    attribution: '<a href="https://github.com/cyclosm/cyclosm-cartocss-style/releases"
}); // the CyclOSM tile layer available from Leaflet servers
```

Let's add our basemaps to Leaflet.

```
// Add the Leaflet basemaps
var map = L.map('myMap', {
    layers: [osm, cyclOSM]
}).setView([-1.295287148, 36.81984753], 7);
```

11.2 Adding the features

Remember our hospital json layer. Let's call it again and transform it to a cluster marker with `fetch`.

```
// Add hospital dataset

url = 'https://raw.githubusercontent.com/sammigachuhi/geojson_files/main/selected_hosp

var cluster = fetch(url)
    .then((response) =>{
        return response.json()
    })
    .then((data) => {
        var markers = L.markerClusterGroup();

        var geojsonGroup = L.geoJSON(data, {
            onEachFeature : function(feature, layer){
                var popupContent =  `<b>Facility Name:</b> ${feature.properties.Facili
```

```
                <b>Type:</b> ${feature.properties.Type}`;
                layer.bindPopup(popupContent)
            },
            pointToLayer: function (feature, latlng) {
                return L.circleMarker(latlng);
            }
        });

        markers.addLayer(geojsonGroup);
        map.addLayer(markers);

    })
    .catch((error) => {
        console.log(`This is the error: ${error}`)
    })
```

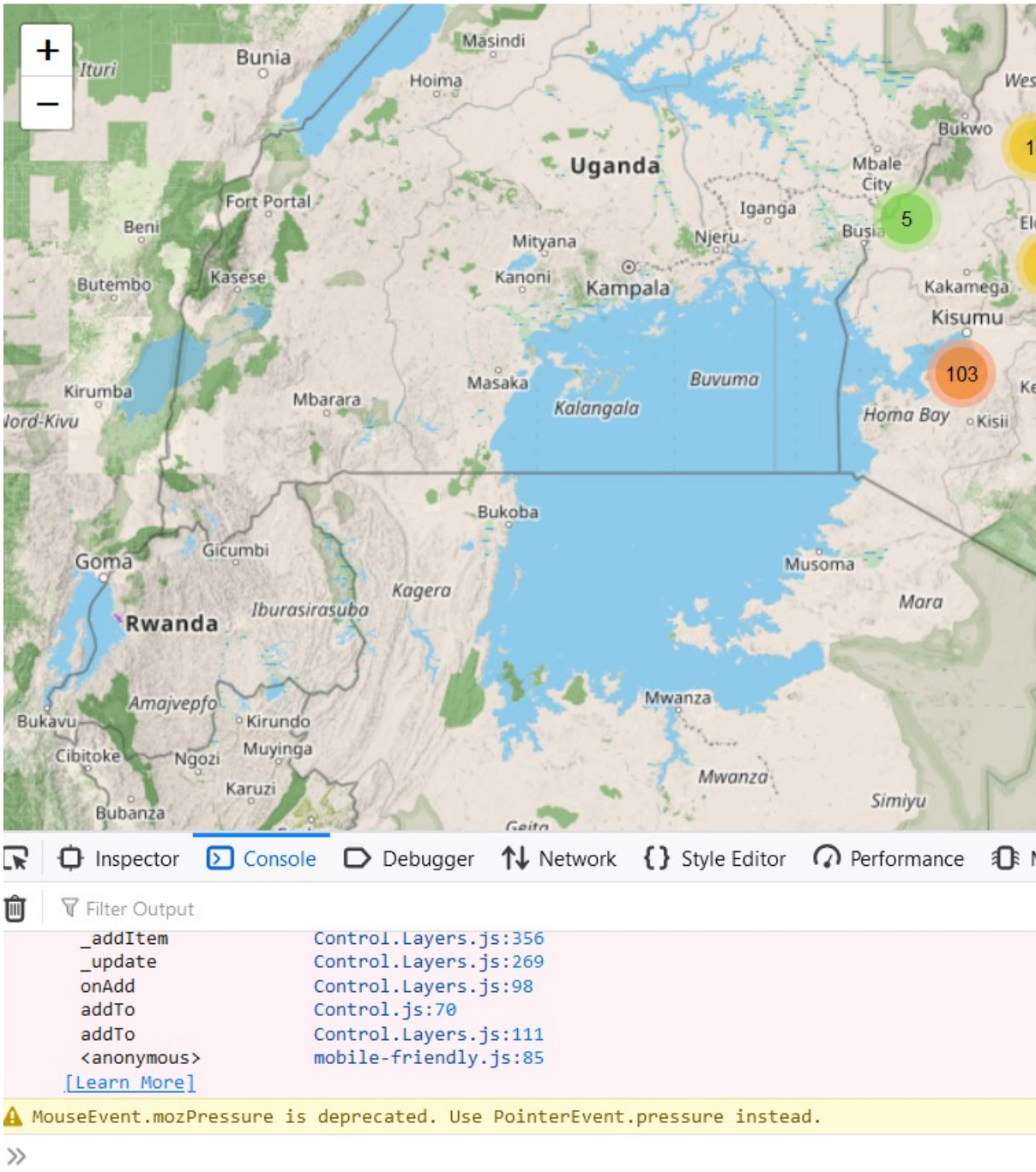
Why assing the `fetch` code to the variable `var cluster`. Well, we were aiming for the stars. We wanted to have a layer control for our `cluster` variable too but unfortunately this plan failed. You will see later on.

Let's put our `basemaps` and `cluster` variables into JavaScript objects to create a layer control for each.

```
// Set object for the basemaps
var basemaps = {
    "OpenStreetMap": osm,
    'cycleOsm': cyclOSM,
}

////Don't add the 'overlays' object. For demonstration purposes only
// Set object for the overlay maps
var overlays = {
    'Hospitals': cluster
}
```

Before you head on any further, let me tell you the result of inserting the `overlays` object into the `L.control.Layers()` class. A liturgy of errors results.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/mobile-friendly-error.jpg'))
```



The webmap works alright, showing popups and all, but there is a list of errors on our console. The one listed as `<anonymous> mobile-friendly.js:85` ac-

tually refers to our `L.control.Layers()` class, and it is without a doubt (no) thanks to the `overlays` argument. It's because `overlays` contains `fetch` which we suspect is behind all these problems. Removing the `overlays` argument and the object altogether eliminates this error.

Incase you had jumped headlong into the pit with me, comment out the `overlays` object and add `L.controLayers()`.

11.3 Zooming to mobile user's location

According to the Leaflet official documentation, Leaflet has a handy shortcut of zooming in to the user's location, if not a geofence of his exact position.

```
// Zoom to your location
map.locate({setView: true, maxZoom: 16});
```

11.4 Add marker to mobile user's geolocation

Even if the location is off by a couple of miles, at least a marker to show the triangulated position will help. At least you will not be *all over* the map! The following code adds a marker to the mobile user's triangulated Latitude-Longitude coordinates, and displays a message showing the radius in which the mobile user is most likely to be found from the marker point.

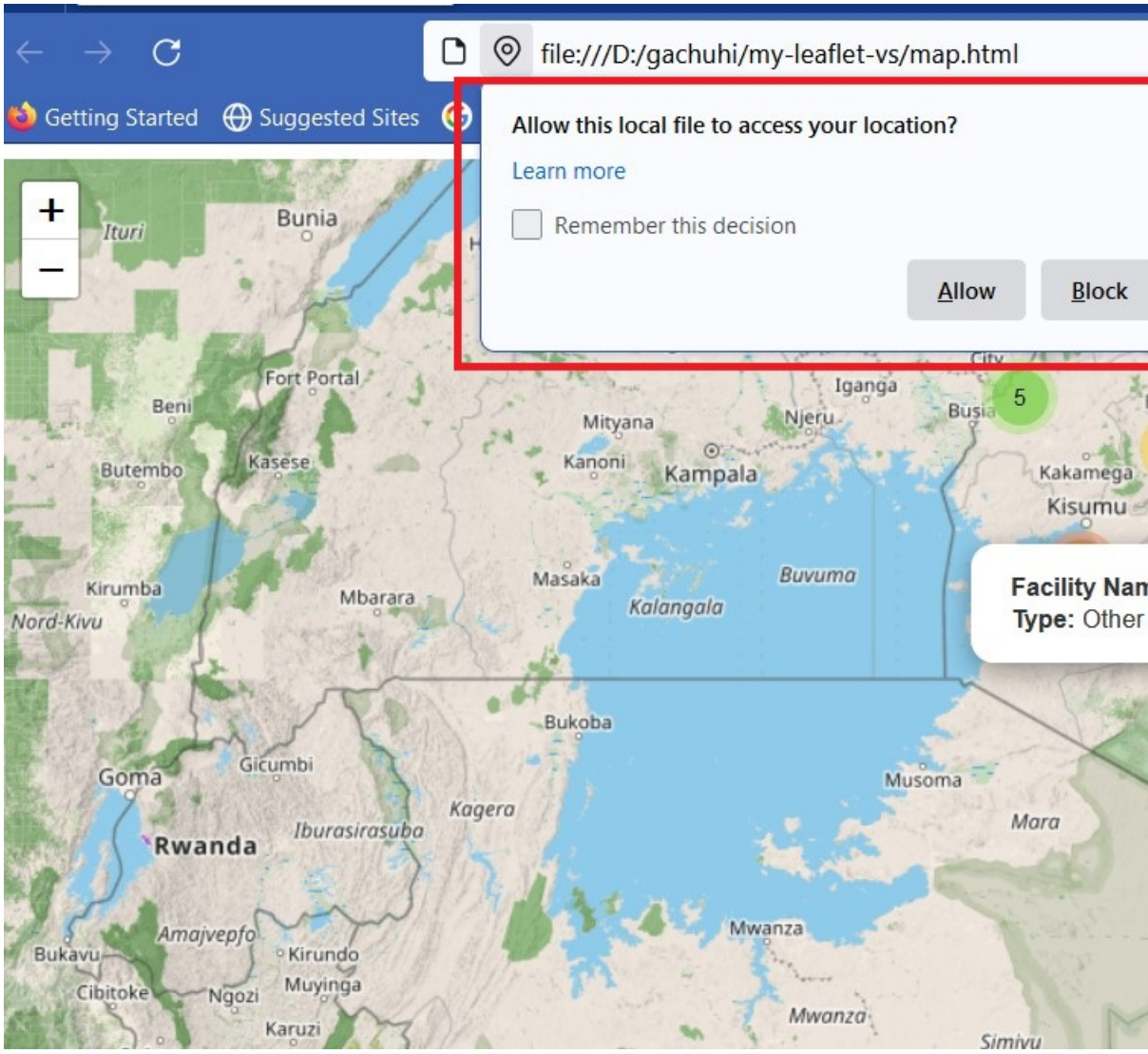
```
// Add marker at your location
function onLocationFound(e) {
    var radius = e.accuracy;

    L.marker(e.latlng).addTo(map)
        .bindPopup("You are within " + Number((radius/1000).toFixed(2)) + " kilometers from this

    L.circle(e.latlng, radius).addTo(map);
}
```

Incase you forgot, the `on` method adds listeners. As a gentle reminder, listeners are codes that run when an event, as simple as hovering or as intensive as double clicking are triggered. In the below code, the listener `'locationfound'` it triggers the `onLocationFound` function in case approximating the mobile user's location went successfully.

```
map.on('locationfound', onLocationFound);
```

The `locationfound` listener is responsible for the message bounded in red when I first load my map. Clicking **Allow** will give the browser the heads up to zoom to my location as it best can.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/location-found.jpg'))
```



What if, getting the mobile user's geolocation is unsuccessful? We will create a function that outputs the error event to our console, as shown below.

```
// Error displayed after finding location failed
```

```
function onLocationError(e) {
    alert(e.message);
}
```

Actually, `message` is an error event that displays the error message of a parameter, in this case the layer. We finally add this `onLocationError` function to the map. It will be triggered if the `'locationerror'` event is true!

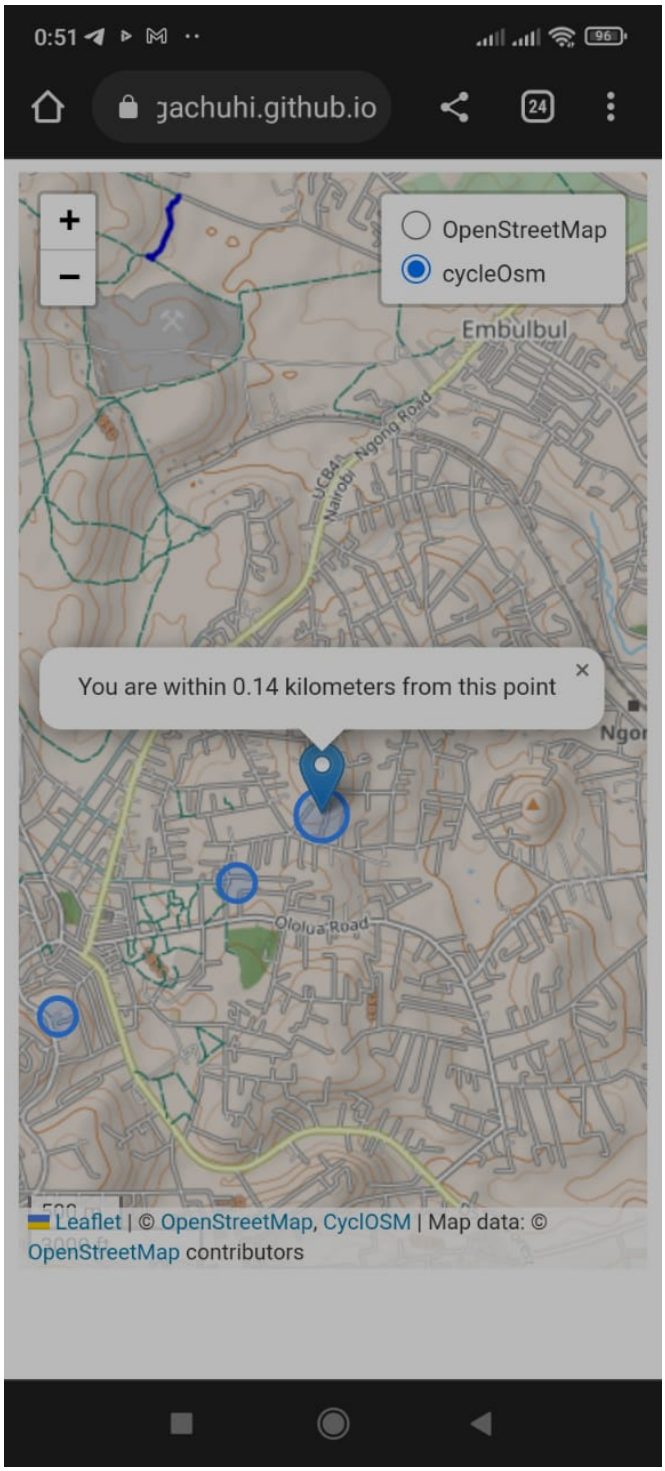
```
map.on('locationerror', onLocationError);
```

11.5 The mobile webmap app in action!

Yours truly has save you the hustle of detailing how this chapter's files have been saved to Github, and converted into a webpage. However, I provide the link to view the web app on your phone. Chances are high it will not fail on you like mine did to me ;). Below is the link. Try it on your phone!

https://sammigachuhi.github.io/hospitals_webapp/

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/mobile-app.jpg'))
```


0:51
gachuhi.github.io
24
OpenStreetMap
cycleOsm
Embulbul
Ngong Road
Nairobi
You are within 0.14 kilometers from this point
Ololua Road
Leaflet | © OpenStreetMap, CyclOSM | Map data: © OpenStreetMap contributors

Remember me mentioning I was aiming for the stars by wanting to add the hospital markers as a layer control? Well, I landed on the skies since you can see that only the basemaps could be added to the layers control widget.

However, this looks like a good hospital locations app! The sky was only the baseline!

The full code script is available from here.

Chapter 12

Web Map Service Layers

12.1 What are Web Map Service (WMS) Layers?

A Web Mapping Service (WMS) consists of geospatial data hosted through the internet with standards set by the Open Geospatial Consortium (OGS).

A WMS enables the exchange of spatial information and viewing over the web in the form of a map or image to your browser. The most common formats of WMS are WMS, WFS, WCS, WPS, WMTS, and WCPS. WMS is the most used. It offers basic panning, zooming and somewhat quick rendering speeds.

12.2 Loading a WMS server

To load a WMS layer into leaflet, we use the `L.tileLayer.wms` class. As simple as that. So let's set up our Leaflet map. Create a new JavaScript file named `wms_layers.js` and insert the following code.

```
let map = L.map('myMap').setView([-1.295287148, 36.81984753], 7);

let tileLayer = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
});
```

We have added an OSM layer because we want to create a layer control widget that also contains WMS layers so that you can switch between both for either spatial awareness or informational purposes (WMS layer). Without further ado, let's add two WMS layers.

```
let wmsLayerTopo = L.tileLayer.wms('https://www.gmrt.org/services/mapserver/wms_merc_ma
    layers: 'topo',
    format: 'image/png'
})

let wmsLayerTopomask = L.tileLayer.wms('https://www.gmrt.org/services/mapserver/wms_mer
    layers: 'topo-mask',
    format: 'image/png'
})
```

12.3 Adding WMS to layer control

Let's insert the above web map layers into a JavaScript object before passing it to a layer control.

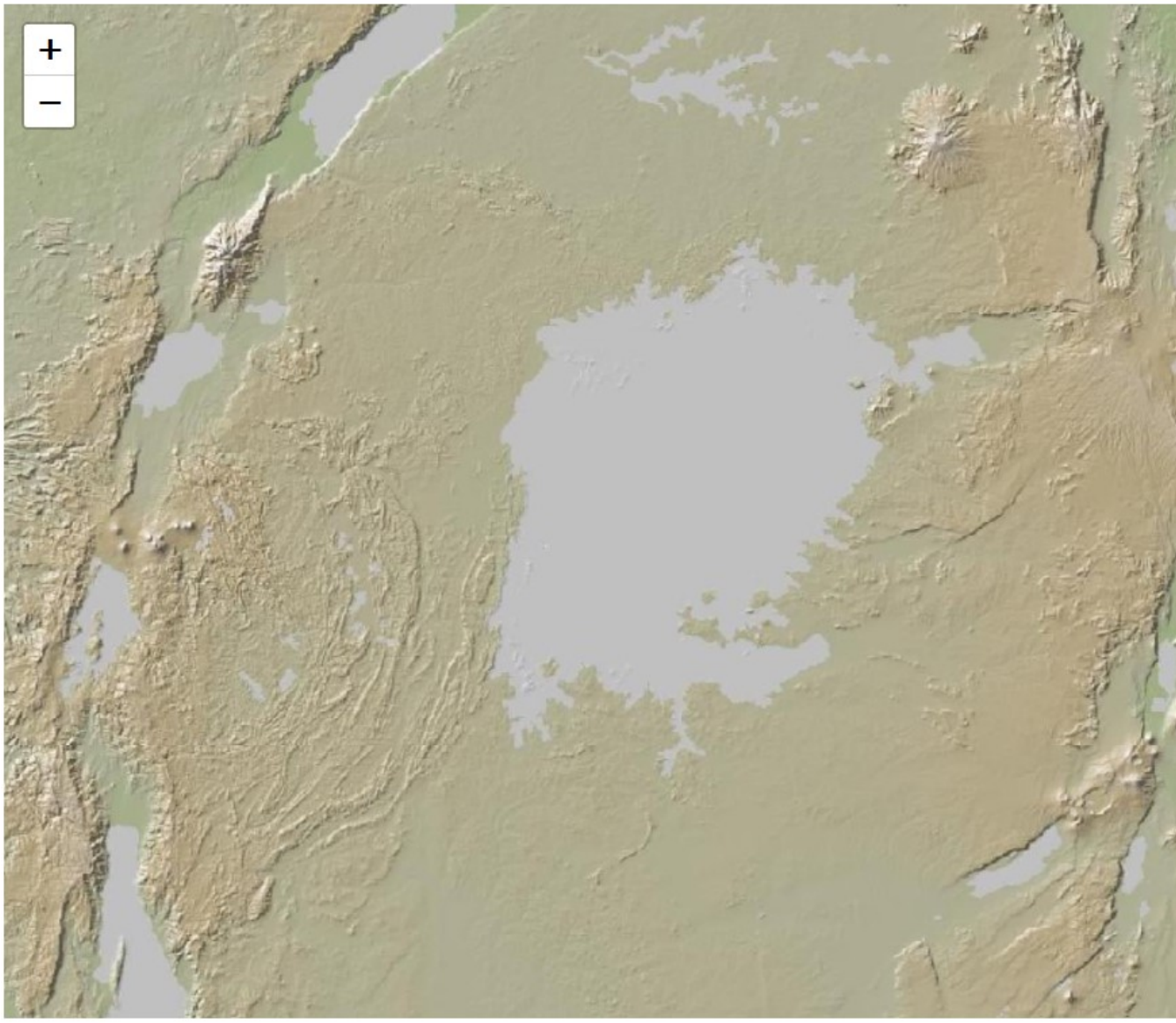
```
var basemaps = {
    OSM: tileLayer,
    Topo: wmsLayerTopo,
    Topo_mask: wmsLayerTopomask,
}
```

Now let's parse the object into a control. If you did Chapter 8 then this should be familiar.

```
L.control.layers(basemaps).addTo(map);

basemaps.Topo.addTo(map);  // To have the Topo as the default map layer
```

The last line `basemaps.Topo.addTo(map)` serves to set up the default layer that will appear when the map is loaded. In this case it is the variable set to `Topo` key in `var basemaps`. If this last line is omitted, the leaflet map will load alright, however, it will be blank canvas unless one of the radioitems is clicked.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/wms_layer.jpg'))
```



And that's it!

We wanted to introduce some African-based WMS layers, of which Digital Earth Africa provides a couple, however, these WMS layers could only be loaded in Qgis. Subsequent attempts to load them in Leaflet even after carefully following the official documents yielded no fruits.

Interested in getting other WMS layers? Go to Spartineo.com.

The files used in these very short chapter are available here.

Chapter 13

Standard Website with Leaflet Project

13.1 Get the HTML Template

Remember me trying to insert a leaflet map into some lousy looking website to do with GMOs back in Chapter 4? Well, there is a aphomism in my local lingo that says only the stay at home thinks that their household cooks well. It is until you challenge yourself with something new that you can prove your salt. In this chapter, we would like to demonstrate how leaflet can be added to a standard HTML website. The purpose of this chapter is to train you how you can insert a leaflet map to your company's, client's or even colleague's website if called upon. They heard you are a web mapping master. So why not?

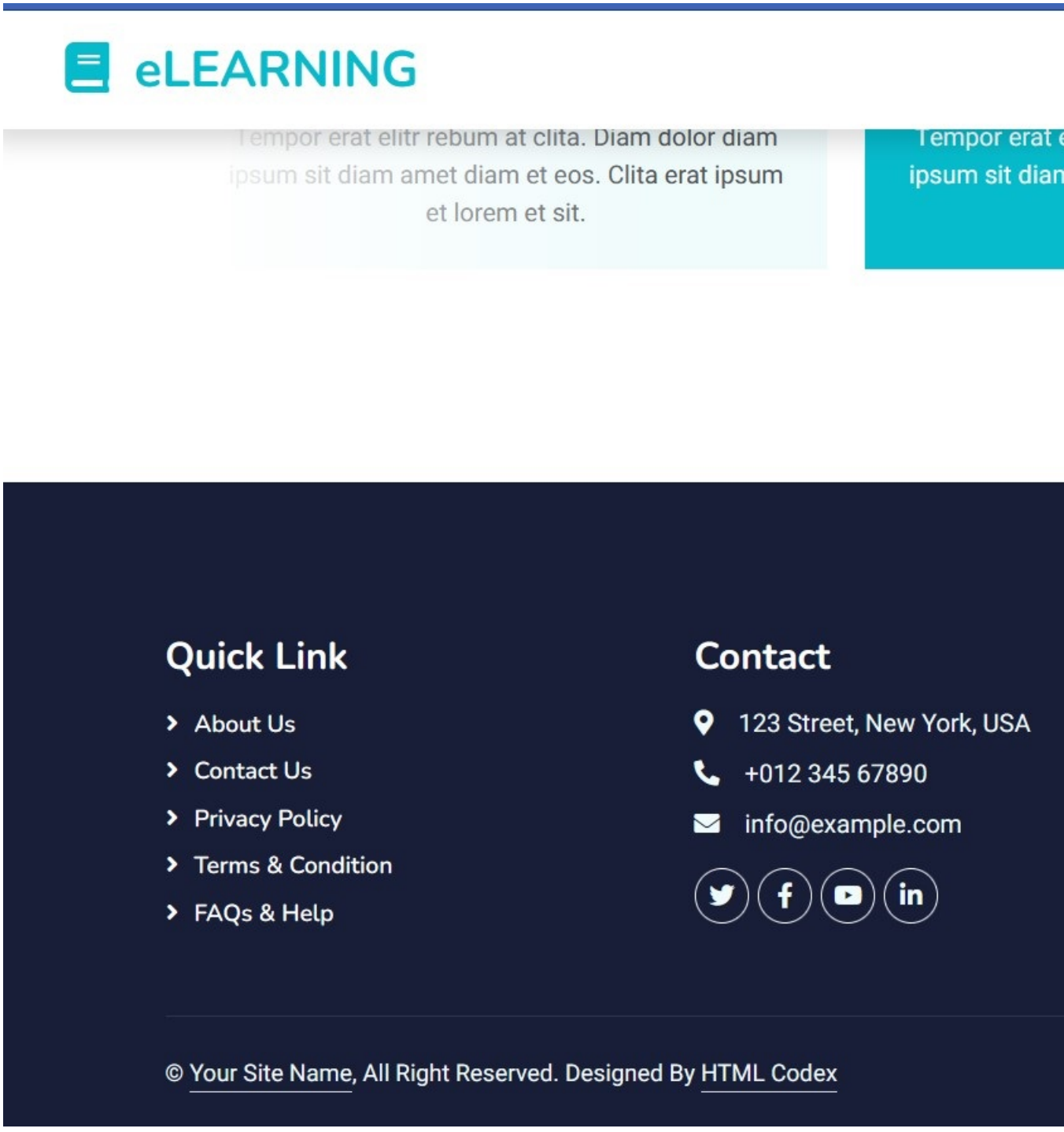
The template used in this exercise was acquired from Free CSS.com and is available here. Extract the files and take a look at the html documents. That is, the `index.html`, `about.html`, `contact.html`, `courses.html`, `team.html` and `testimonial.html`. Pay special attention to the `<head>` element and the `<div>` containing the Gallery section of the website. The Gallery `<div>` looks like so:

```
<div class="col-lg-3 col-md-6">
                    <h4 class="text-white mb-3">Gallery</h4>
                    <div class="row g-2 pt-2">
                        <div class="col-4">
                            <img class="img-fluid bg-light p-1" src="img/course-1.jpg" alt="">
                        </div>
                        <div class="col-4">
                            <img class="img-fluid bg-light p-1" src="img/course-2.jpg" alt="">
                        </div>
                        <div class="col-4">
```

```
                        <img class="img-fluid bg-light p-1" src="img/course-3.jpg"
                    </div>
                    <div class="col-4">
                        <img class="img-fluid bg-light p-1" src="img/course-2.jpg"
                    </div>
                    <div class="col-4">
                        <img class="img-fluid bg-light p-1" src="img/course-3.jpg"
                    </div>
                    <div class="col-4">
                        <img class="img-fluid bg-light p-1" src="img/course-1.jpg"
                    </div>
                </div>
            </div>
```

Here is where the Gallery in the elearning webpage is situated.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/gallery.jpg'))
```


eLEARNING
Tempor erat elitr rebum at clita. Diam dolor diam ipsum sit diam amet diam et eos. Clita erat ipsum et lorem et sit.
Tempor erat
ipsum sit dian
Quick Link
About Us
Contact Us
Privacy Policy
Terms & Condition
FAQs & Help
Contact
123 Street, New York, USA
+012 345 67890
info@example.com
© Your Site Name, All Right Reserved. Designed By HTML Codex

Why do we want to change the gallery, or rather, to insert a leaflet map in its place? Well, I decided to be mean on it. Sorry, that wasn't supposed to come out of my mouth. Candidly speaking, the Gallery section only contained some pretty pictures and personally I felt replacing them with a leaflet map would serve as the best demonstration for this chapter.

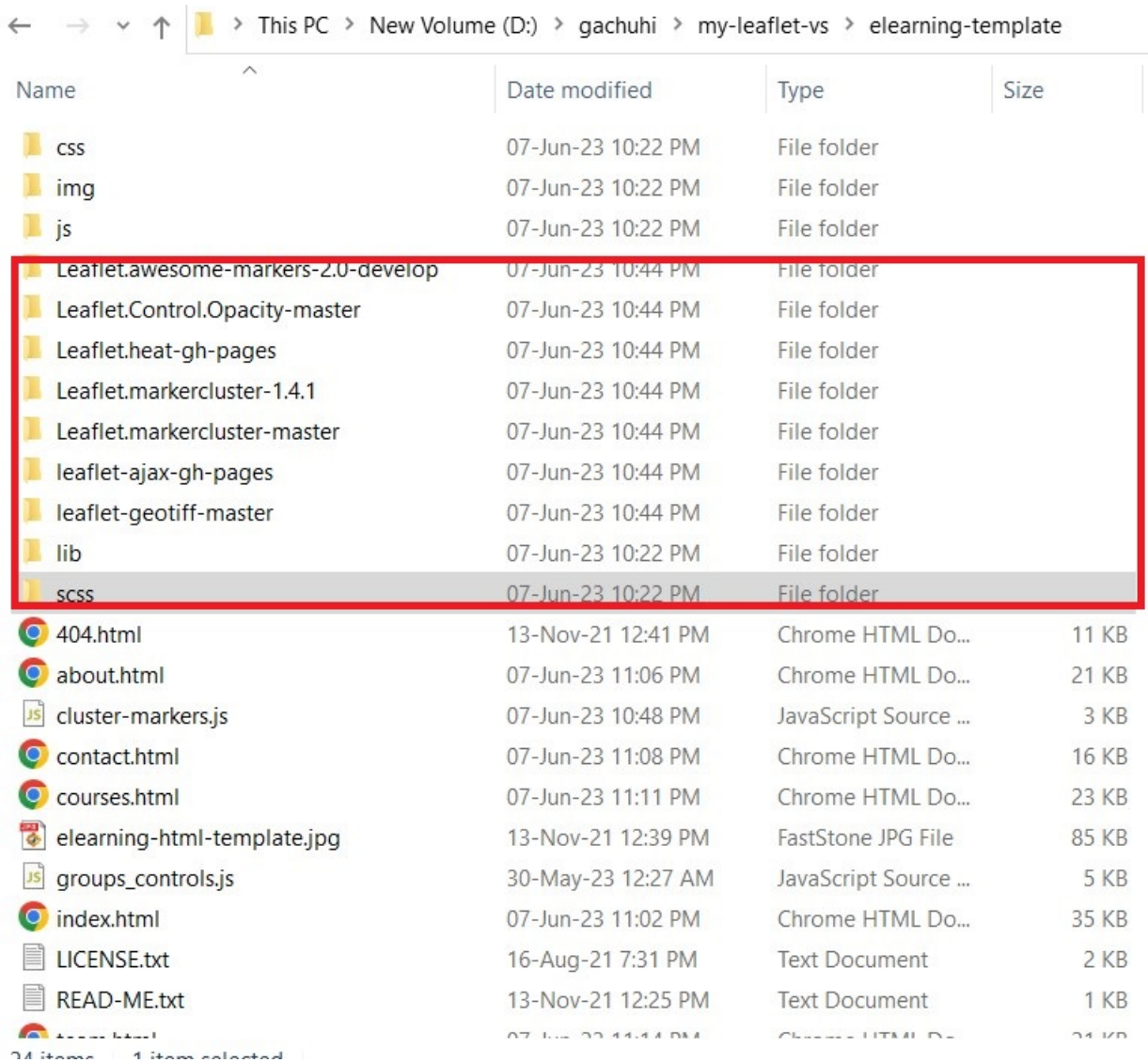
13.2 Insert Leaflet to standard html website

Since we want to insert a leaflet to the eLearning website, we will have to load the requisite Leaflet links and scripts to the websites `<head>` element, just like we used to do to our `map.html`. Thereafter, we shall replace the entire gallery section with just one `<div>` that references one of our many Leaflet JavaScript files.

Ready? Let's go.

Assuming you are working on VSCode, copy all the plugin files for leaflet, such as `Leaflet.markercluster-master` folder and others you had extracted prior to parsing them to your javascript files. Perhaps an image will make the instructions clearer. The below files should be in your eLearning template folder.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/elearning-plugins.jpg'))
```



Why are we ensuring that the leaflet plugin folders are in the same directory of our eLearning template files? YOu will know in a tid bit.

Go to the `index.html` file of your eLearning template. From the `<head>` to `</head>` tags, replace the existing code with the following:

```
<meta charset="utf-8">
    <!-- For leaflet and mobile compatibility -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user
```

```
<title>eLEARNING - eLearning HTML Template</title>
<meta content="width=device-width, initial-scale=1.0" name="viewport">
<meta content="" name="keywords">
<meta content="" name="description">

<!-- Favicon -->
<link href="img/favicon.ico" rel="icon">

<!-- Google Web Fonts -->
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Heebo:wght@400;500;600&family=

<!-- Icon Font Stylesheet -->
<link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.10.0/css/all.min
<link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons

<!-- Libraries Stylesheet -->
<link href="lib/animate/animate.min.css" rel="stylesheet">
<link href="lib/owlcarousel/assets/owl.carousel.min.css" rel="stylesheet">

<!-- Customized Bootstrap Stylesheet -->
<link href="css/bootstrap.min.css" rel="stylesheet">

<!-- Template Stylesheet -->
<link href="css/style.css" rel="stylesheet">

<!-- For leaflet -->
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css"
    integrity="sha256-kLaT2GOSpHechhsozzB+flnD+zUyjE2LlfWPgU04xyI="
    crossorigin=""/>
<link rel="stylesheet" href="Leaflet.Control.Opacity-master\Leaflet.Control.Opacity
<!-- For Leaflet marker clusters -->
<link rel="stylesheet" href="Leaflet.markercluster-master\Leaflet.markercluster-mas

<!-- For leaflet scripts -->
<script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js"
    integrity="sha256-WBkoXOwTeyKclOHuWtc+i2uENFpDZ9YPdf5Hf+D7ewM="
    crossorigin=""></script>
<script src="leaflet-ajax-gh-pages\dist\leaflet.ajax.js"></script>
<script src="leaflet-ajax-gh-pages\dist\leaflet.ajax.min.js"></script>
<script src="leaflet-ajax-gh-pages\example\leaflet.spin.js"></script>
<script src="leaflet-ajax-gh-pages\example\spin.js"></script>
<script src="Leaflet.heat-gh-pages\Leaflet.heat-gh-pages\dist\leaflet-heat.js"></s
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></s
<!-- For leaflet cluster markers -->
```

```
<script src="Leaflet.markercluster-1.4.1\Leaflet.markercluster-1.4.1\dist\leaflet.markerclust
<script src="Leaflet.markercluster-1.4.1\Leaflet.markercluster-1.4.1\dist\leaflet.markerclust
```

We have just retained the existing code and added the leaflet `<link>` and `<script>` tags we have experienced so far – from good old `map.html` of course. Before I forget, here's why the leaflet plugin folders have to be together with the eLearning files. When we copied our `<link>` and `<script>` tags from `map.html` they were referring to certain paths. When you paste them to the `index.html` of your eLearning template, the path's are no longer correct. They have just been placed in a destination/directory! However, if you paste the Leaflet folders you have worked with so far so that they are within the same directory, you don't have to (painfully) change the paths because they will remain *correct*.

Didn't get me? Take the following `<script>` tag.

```
<script src="Leaflet.markercluster-1.4.1\Leaflet.markercluster-1.4.1\dist\leaflet.markercluster.j
```

It still remains correct within the `index.html` of our eLearning template because we have pasted the same `Leaflet.markercluster-1.4.1` folder in the same directory as other eLearning template file. ie. the path still refers to the same place. No adventuring to other folder inside or outside the same eLearning template directory!

Another small favour. Copy and past the `cluster_markers.js` file into the eLearning template directory.

Alright.

A glass of water.

Back to the screen.

Time to add the leaflet map. Where? At the gallery section.

As a reminder, it looks like this:

```
<div class="col-lg-3 col-md-6">
                    <h4 class="text-white mb-3">Gallery</h4>
                    <div class="row g-2 pt-2">
                        <div class="col-4">
                            <img class="img-fluid bg-light p-1" src="img/course-1.jpg" alt="">
                        </div>
                        <div class="col-4">
                            <img class="img-fluid bg-light p-1" src="img/course-2.jpg" alt="">
                        </div>
                        <div class="col-4">
                            <img class="img-fluid bg-light p-1" src="img/course-3.jpg" alt="">
```

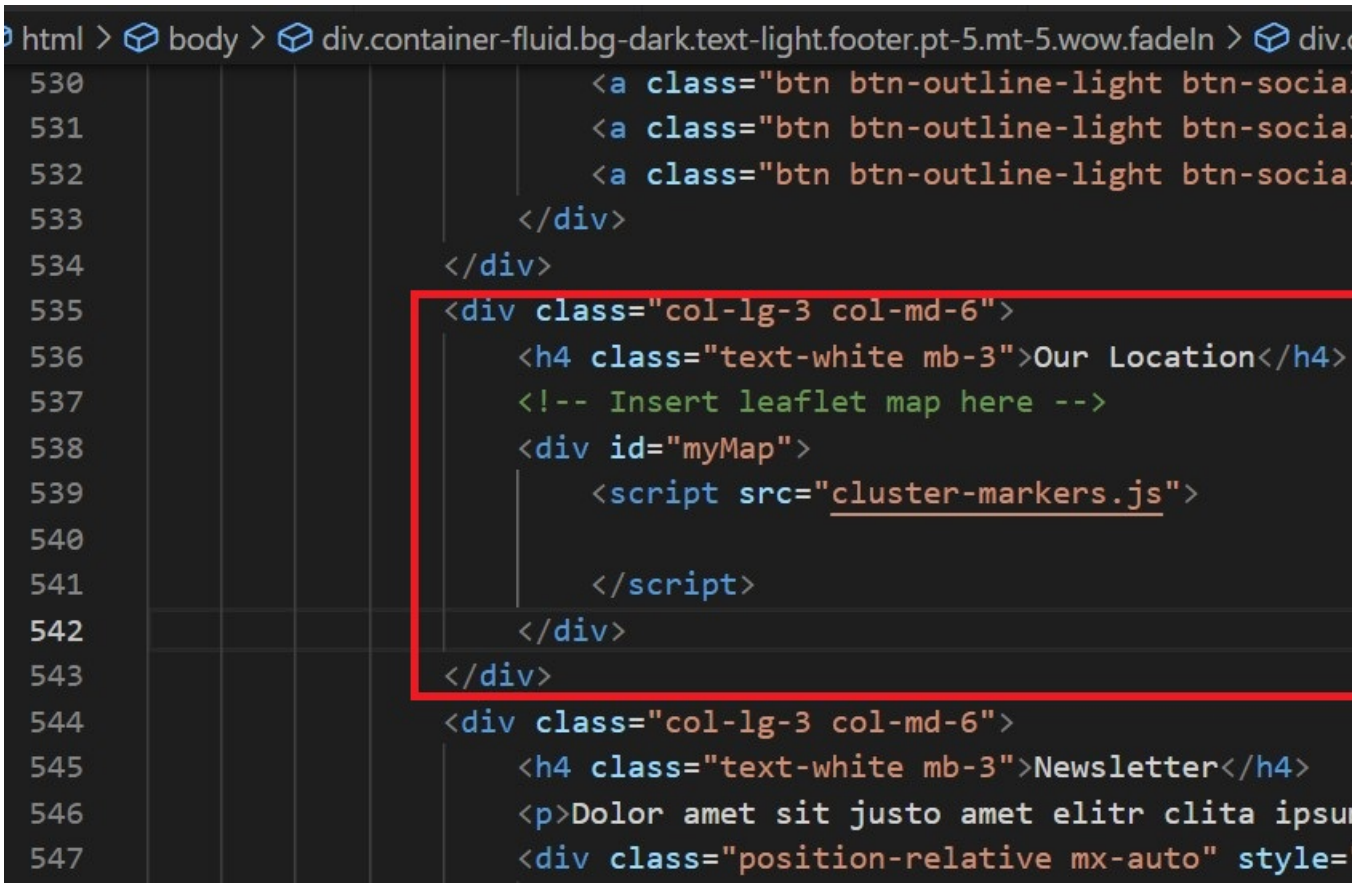
```
                        </div>
                        <div class="col-4">
                            <img class="img-fluid bg-light p-1" src="img/course-2.jpg"
                        </div>
                        <div class="col-4">
                            <img class="img-fluid bg-light p-1" src="img/course-3.jpg"
                        </div>
                        <div class="col-4">
                            <img class="img-fluid bg-light p-1" src="img/course-1.jpg"
                        </div>
                    </div>
                </div>
```

You may have to scroll *really* down to locate it. Right under the `<h4 class="text-white mb-3">Gallery</h4>` tag, replace it with the following code, all the to the `<div>` tag after `<img class="img-fluid bg-light p-1" src="img/course-1.jpg" alt="">`.

```
<div id="myMap">
                        <script src="cluster-markers.js">

                        </script>
                    </div>
```

Perhaps an image will clarify matters a bit. This is how it should look:

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/div_replace.jpg'))
```



Also as a heads up, remember to replace the `<h4>` tags with the statement `Our Location`. It's no longer a gallery!

As if that was not enough, going easy on our fingers and all, we have to set the CSS properties for our leaflet map.

13.3 Editing the CSS

Open the `style.css` file of your eLearning template in your text editor (such as VS code). The path to the CSS file is as follows:

```
elearning-template\css\style.css
```

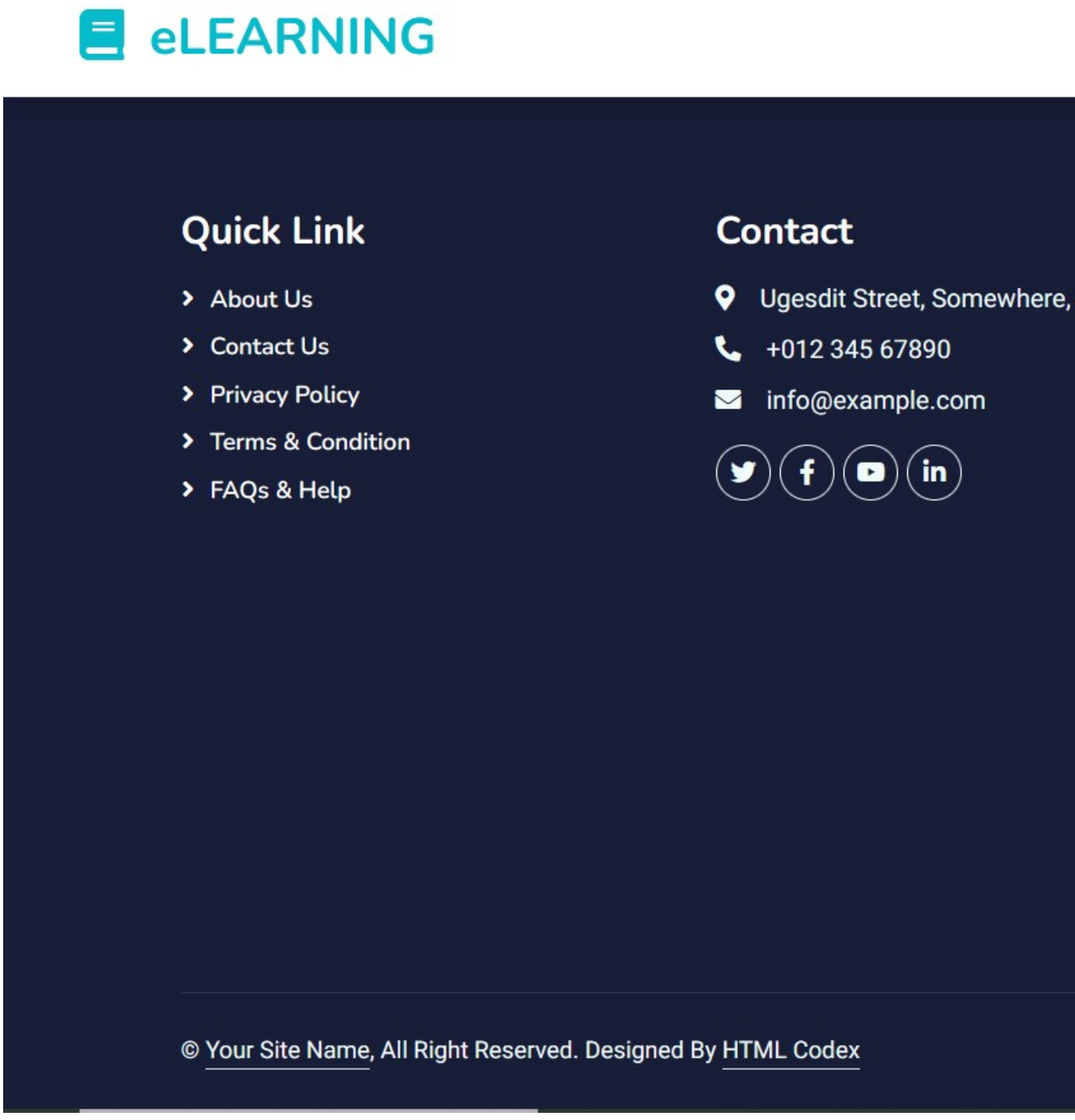
Scroll down to the very bottom of `style.css` and paste the following CSS properites for our Leaflet map.

```
#myMap {
    height: 400px;
    width: auto;
}
```

Remember the `#` and the text that follows references the specifically named `id` attribute in the tags of our html element. In the above case, it references the `<div>` with the ID `#myMap`.

After reloading `index.html`, this is how our leaflet map looks inside the eLearning template.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/location_elearning.jpg'))
```


eLEARNING
Quick Link
About Us
Contact Us
Privacy Policy
Terms & Condition
FAQs & Help
Contact
Ugesdit Street, Somewhere,
+012 345 67890
info@example.com
© Your Site Name, All Right Reserved. Designed By HTML Codex

Good.

13.4 Adding leaflet to every webpage

Now click the **About**, **Courses**, **Pages/Our Team**, **Pages/Testimonial** and **Contact** web pages. Is the leaflet map there? No. It's the gallery. Imagine you will have to replace the `<head>` and Gallery `<div>` elements here as well!

"How and I don't have the time?!!" If you look at the eLearning template directory, you will see the **about**, **courses**, **contact**, **team** and **testimonial** html files. Replace their `<head>` elements and Gallery `<div>`s just as you did for `index.html`. Luckily I am your hero. This was done for you and the files are available here.

13.5 Posting the Html website to the world

This chapter will leave out the details of saving your files to Github. However, here's a clue. In order to automatically render a file in Github online, make sure the root html file is named `index.html`. These are taken as the default root html files by Github.

Yours truly already saved the file to Github. Here's the link to the file live and online.

Chapter 14

Leaflet in ESRI

Alright. If you are a GIS practitioner, you have (most) probably heard about ESRI, one of the world's leading geospatial software and services provider. You (might) have also come across various ESRI basemap servers, such as topographic, streets and imagery layers. They also have plugins that allow leaflet users to access the ArcGIS functionalities. For example, the ESRI plugins for leaflet allow you to access some ESRI basemaps and products, while also allowing you to become an ESRI ArcGIS JavaScript developer. Quite a mouthful of a career name.

To use ESRI Leaflet, you have to create an ArcGIS Developer account and also get an API key. Kindly do so before proceeding.

14.1 ESRI Leaflet plugins

As we had mentioned earlier, you need ESRI leaflet plugins to fire up ESRI power in your leaflet environment. Alright. Remember your `map.html` file? You have probably added a plethora of `<link>`s and `<script>`s to your `map.html` file but bear with us a little longer. Add the following `<script>`s to the `<head>` element of your `map.html`.

```
<!-- Load Esri Leaflet from CDN -->
        <script src="https://unpkg.com/esri-leaflet@3.0.10/dist/esri-leaflet.js"></script>
        <script src="https://unpkg.com/esri-leaflet-vector@4.0.2/dist/esri-leaflet-vector.js"></s
```

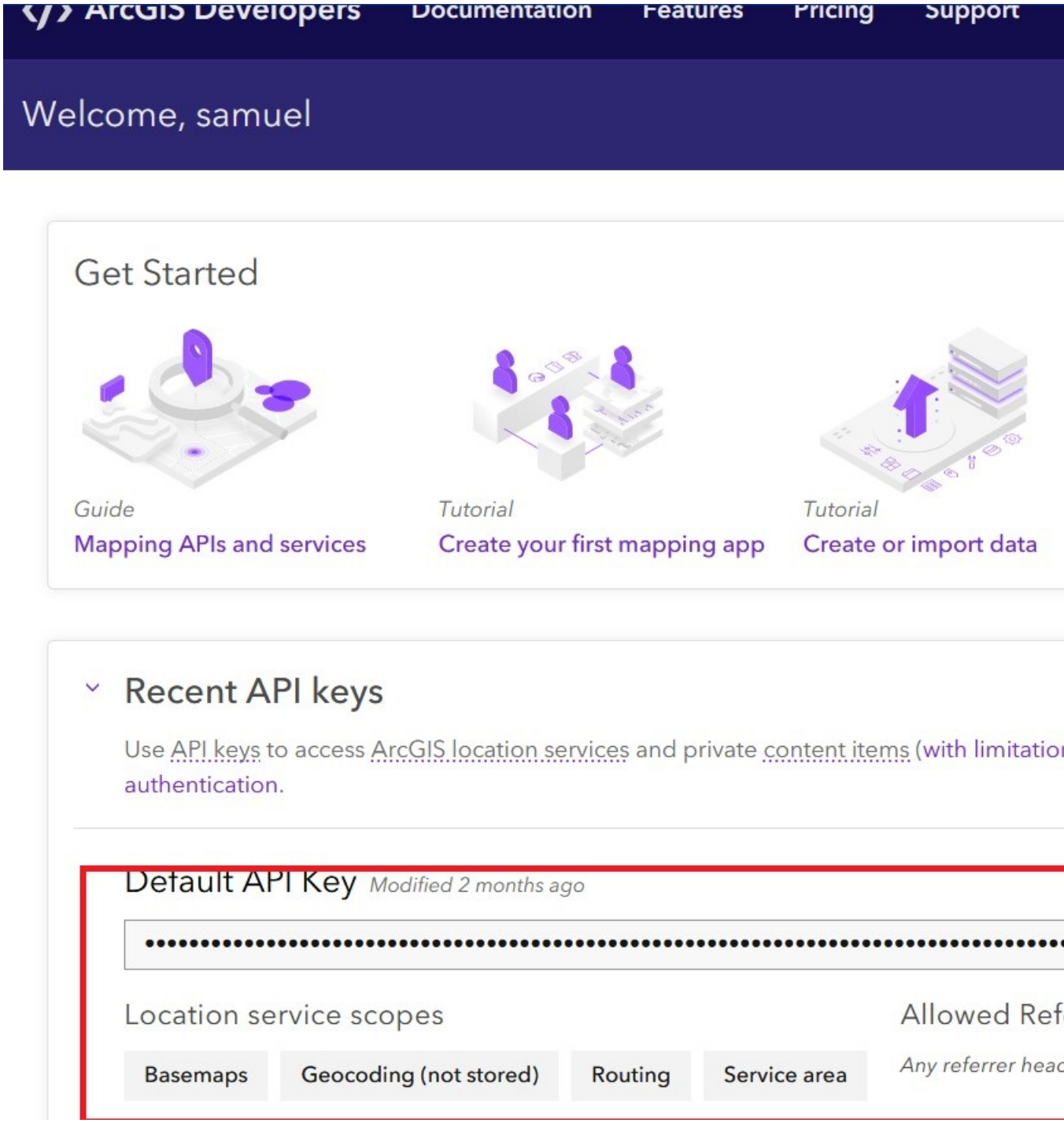
Create a new JavaScript file called `esri_leaflet.js` and get ready to enjoy ESRI services as though you made them!

14.2 Creating an ESRI Leaflet map

On your blank `esri_leaflet.js` file, create a variable to store your Application Programming Interface (API) Key. This key is important to access your ESRI benefits, much like the magic phrase "Open Sesame" opens the doorway to a cave full of treasures as in the legendary story of *Ali Baba and the Forty Thieves*. Of course I am no thief.

Below is an example of how to access your API key from your ArcGIS Developer account.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/esri_leaflet_api.jpg'))
```


ArcGIS Developers
Documentation
Features
Pricing
Support
Welcome, samuel
Get Started
Guide
Mapping APIs and services
Tutorial
Create your first mapping app
Tutorial
Create or import data
Recent API keys
Use API keys to access ArcGIS location services and private content items (with limitation
authentication.
Default API Key
Modified 2 months ago
Location service scopes
Allowed Ref
Basemaps
Geocoding (not stored)
Routing
Service area
Any referrer heac

Copy paste that key to your `esri_leaflet.js` file.

```
const apiKey = "Your key";
```

Of course, don't expect me to showcase mine to the entire world.

Anyways, let's proceed to create a basemap. To create one, just create a variable called `basemapEnum` that stores the ESRI basemap identifier. What is that last thing? It is basically a basestyle map and in ESRI, at least when using Leaflet, it is accessed by parsing the provider name and the desired style, like so: `{Provider}:{Style name}` or `{Provider}:{Style name}:{Component}`.

```
const basemapEnum = "ArcGIS:Streets";
```

Alright. It's about time we fired up our mapping power. Create an ESRI leaflet map instance much like we have always been doing.

```
const map = L.map("myMap", {
    minZoom: 2
}).setView([0.3556, 37.5833], 6.5);
```

We had created a basemap variable earlier, so we will parse it to the `L.esri.Vector.vectorBasemapLayer` class which is responsible for create basemaps. Actually one can also parse the `{Provider}:{Style name}` to the `L.esri.vector.vectorBasemapLayer()` class but our method of using variable names seems far much more cleaner.

```
L.esri.Vector.vectorBasemapLayer(basemapEnum, {
    apiKey: apiKey,
}).addTo(map);
```

Now open your `map.html` file. What do you see?

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/esri_leaflet_basemap.jpg'))
```

Now imagine there are other styles as shown in this webpage. Spoil yourself a bit.

14.3 Geocode search

Of course there are other ESRI Leaflet functionalities. I must confess I wanted to know how to add a raster layer, such as `.tif` to either Leaflet or ESRI leaflet but they all seem to favour tiled layers. The former has a plugin to do so, but was too complicated (still trying unsuccessfully). Nevertheless, there are other cool services that ESRI leaflet offers, such as geocode search[1].

Let's do that.

14.4 Add search bar

Add the following `<link>` and `<script>` to your `<head>` element.

```
<!-- Load Esri Leaflet Geocoder from CDN -->
        <link rel="stylesheet" href="https://unpkg.com/esri-leaflet-geocoder@3.1.4/dis
        <script src="https://unpkg.com/esri-leaflet-geocoder@3.1.4/dist/esri-leaflet-g
```

Time to add a search control widget to the topright of our ESRI leaflet map. Search control widgets are created using the `L.esri.Geocoding.geosearch` class which we shall parse to a variable called `searchControl`.

```
const searchControl = L.esri.Geocoding.geosearch({
        position: "topright",
        placeholder: "Enter an address or place e.g. 1 York St",
        useMapBounds: false,

      }).addTo(map);
```

Just like in leaflet ESRI leaflet constructors have their options. Please refer the options for this specific constructor here.

Another small thing; change your `const basemapEnum` to read `ArcGis:Navigation`.

```
const basemapEnum = "ArcGIS:Navigation";
```

If you refresh your `map.html` file, apart from having a new basemap style, you will see a new search bar at the top right. However, it is non-functional. It takes us nowhere.

[1] Geocoding is the process of converting address or place text into a location. The geocoding service provides address and place geocoding as well as reverse geocoding

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/search_bar.jpg'))
```

14.5 Make the search bar functional

In order to make the search bar do what it says, we will set the value of the `providers` key to a `arcgisOnlineProvider`. The latter is instantiated with the constructor `L.esri.Geocoding.arcgisOnlineProvider`. Okay where did you get it from? Well ESRI has various open source tutorials on its platform.

```
const searchControl = L.esri.Geocoding.geosearch({
    position: "topright",
    placeholder: "Enter an address or place e.g. 1 York St",
    useMapBounds: false,

    // Add provider
    providers: [
        L.esri.Geocoding.arcgisOnlineProvider({
          apikey: apiKey,
          nearby: {
            lat: 0.3556,
            lng: 37.5833
          }
        })
      ]

  }).addTo(map);
```

Now try your search bar again. Some location names appear as you type them in, and if you press **Enter** it takes you to the exact location but there is no marker to pinpoint that specific address. We will sort that latter.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/searchable.jpg'))
```

We will add a layer group to store the geocoding results.

```
const results = L.layerGroup().addTo(map);
```

We will create an event handler to access the `data` from the search results. We shall also add a `clearLayers` call to remove the previous data from the layer group. This is beginning to sound complicated, but I'm just following along the ESRI Leaflet geocoding tutorial.

```
searchControl.on("results", (data) => {
```

```
        results.clearLayers();

      });
```

The following loop adds the coordinates of the searched location to the marker. This loop goes into the `searchControl`.

```
for (let i = data.results.length - 1; i >= 0; i--) {
          const marker = L.marker(data.results[i].latlng);

          results.addLayer(marker);

        }
```

The `lngLatString` variable below will store the rounded coordinates, and our familiar `bindPopup` and `openPopup` will show the coordinates and our address.

```
const lngLatString = `${Math.round(data.results[i].latlng.lng * 100000) / 100000}, ${
            Math.round(data.results[i].latlng.lat * 100000) / 100000
          }`;
          marker.bindPopup(`<b>${lngLatString}</b><p>${data.results[i].properties.Longl

          results.addLayer(marker);

          marker.openPopup();
```

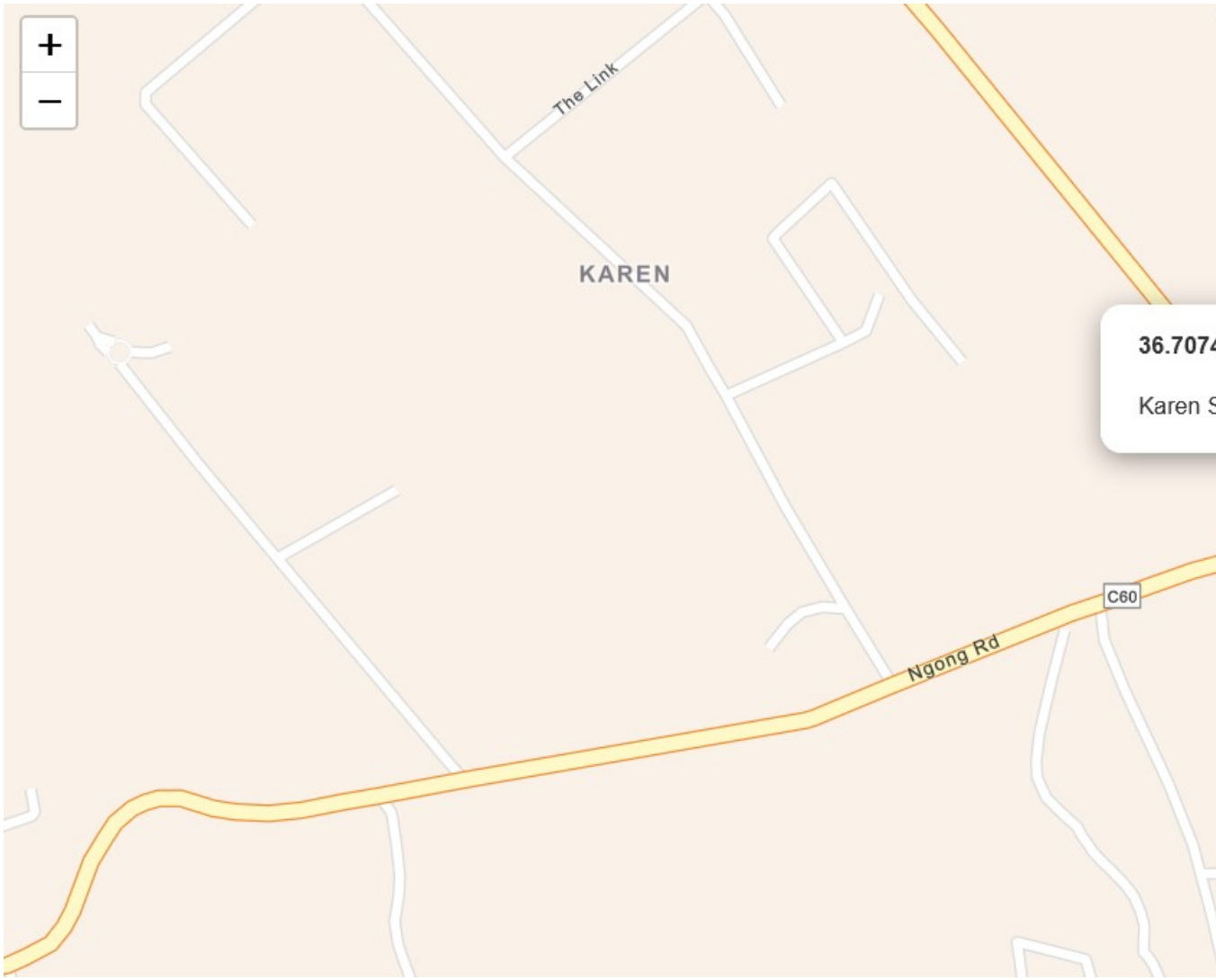
The entire `searchControl` code block should look like below.

```
searchControl.on("results", (data) => {
    results.clearLayers();

    for (let i = data.results.length - 1; i >= 0; i--) {
        const marker = L.marker(data.results[i].latlng);

        const lngLatString = `${Math.round(data.results[i].latlng.lng * 100000) / 1000
            Math.round(data.results[i].latlng.lat * 100000) / 100000
          }`;
          marker.bindPopup(`<b>${lngLatString}</b><p>${data.results[i].properties.Longl

        results.addLayer(marker);

        marker.openPopup();
```

```
    }

});
```

So now when you search for a particular place in the search bar and press **Enter**, a marker and with a pop up showcasing the address and longitude-latitude coordinates of the location should appear.

```
knitr::include_graphics(rep('D:/gachuhi/my-leaflet/images/search_success.jpg'))
```



There are many more tutorials on using ESRI Leaflet in the ArcGIS Developers website.

Here are the files used in this exercise.

Chapter 15

Conclusion

Now here comes the end of our book. If you have reached this far, then congratulations! You now have the power to continue using leaflet to create cool webmaps, even far better than those you've encountered in this book.

I hope this book has been helpful. For anything that you found difficult to understand, or was without sufficient clarity, it had nothing to do with you. *Finis!*