Leaflet book

Samuel Gachuhi Ngugi

2023-04-25

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? Not really.Leaflet, when barescrapped 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.

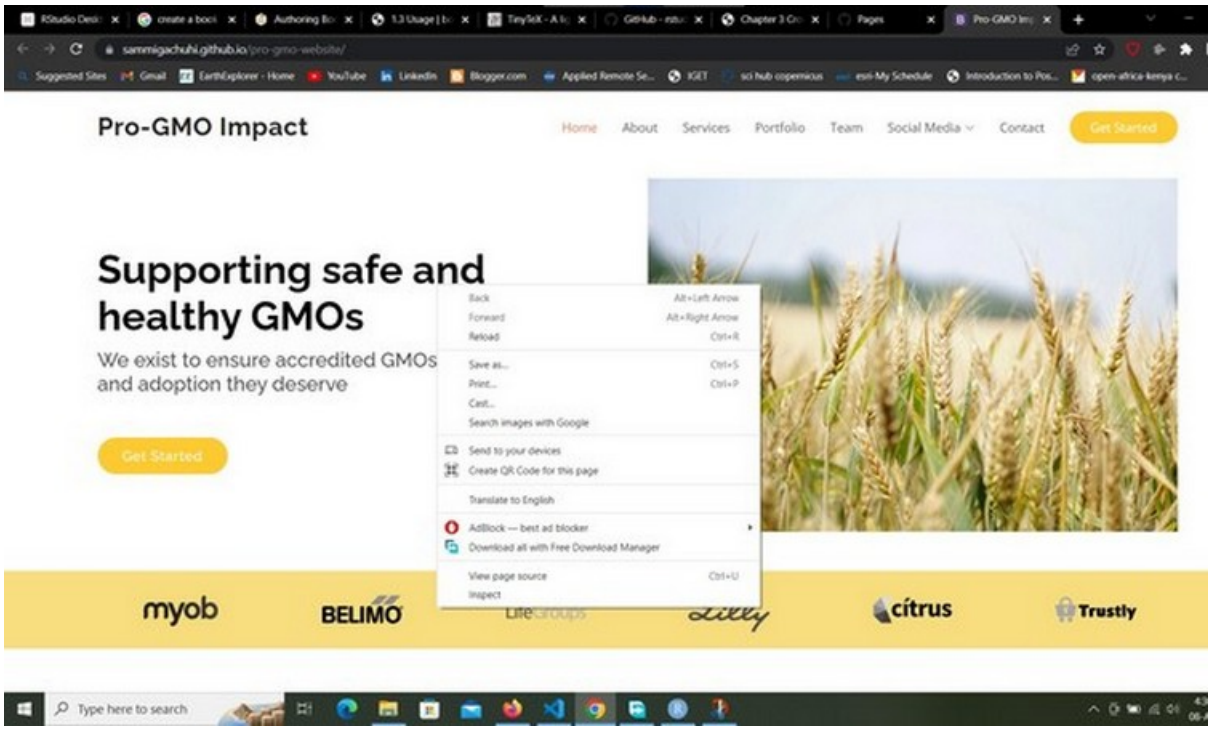
1.2 How does it work?

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?

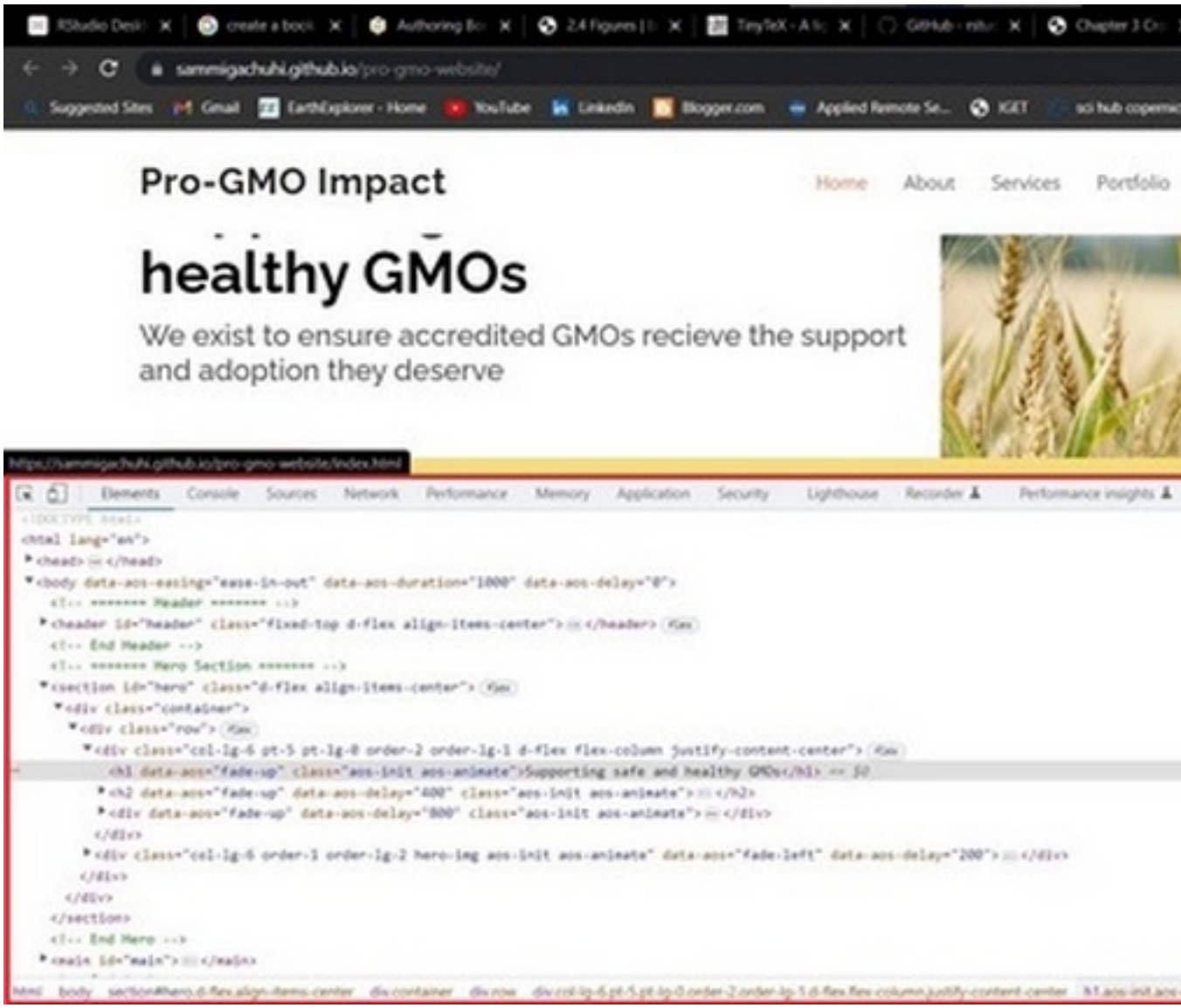
Well, beginning with `html`, which stands for **Hypertext Markup Language**, it is the language that is used in creating webpages. By talking of language, it is actually the standard. I am yet to come across any webpage that is made up of everything apart from HTML. If you want to have a view of what 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/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/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. Whereas this document does not provide an indepth view 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 and Pycharm. Check their websites on their installation methods for your personal computer. For this book, we shall be using VS code.

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

   tag. The

   tag defines, rather than contains, as in the case of

the metadata of the html webpage. In our case, we have used the attribute `charset="utf-8"` to specify the encoding for HTML5 documents which is `utf-8`.

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 enhances interactivity to HTML files which without it remain just static. 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 them to a html file using `<script>` is what brings in the interactive web features to an otherwise blank html.

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 `<body>` element of our webpage.

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

The body is known as the selector. 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 one above, we would specify our first one in a CSS document like so:

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

The values of that particular `<div>` could be changed to whatever you like, so long as they correspond to the right property. If it were a class, the particular class, assuming they were several, would be selected with 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/elements2.jpg"))
```