CREATE WITH DATA

D3 START TO FINISH

2nd Edition

Learn how to make a custom data visualisation using D3.js

PETER COOK

D3 Start to Finish

Peter Cook

D3 Start to Finish

2nd Edition

This version was published on 21st February 2025.

Also by Peter Cook:

Visualising Data with JavaScript

Data Dashboards with JavaScript

Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation

Visit leanpub.com/u/createwithdata for more information.

This is a preview version that includes the first chapter and a chapter on D3 selections.

Chapter 1. Introduction

Welcome! My aim is to teach you about D3.js and show you how a real-world interactive data visualisation called Energy Explorer is built.

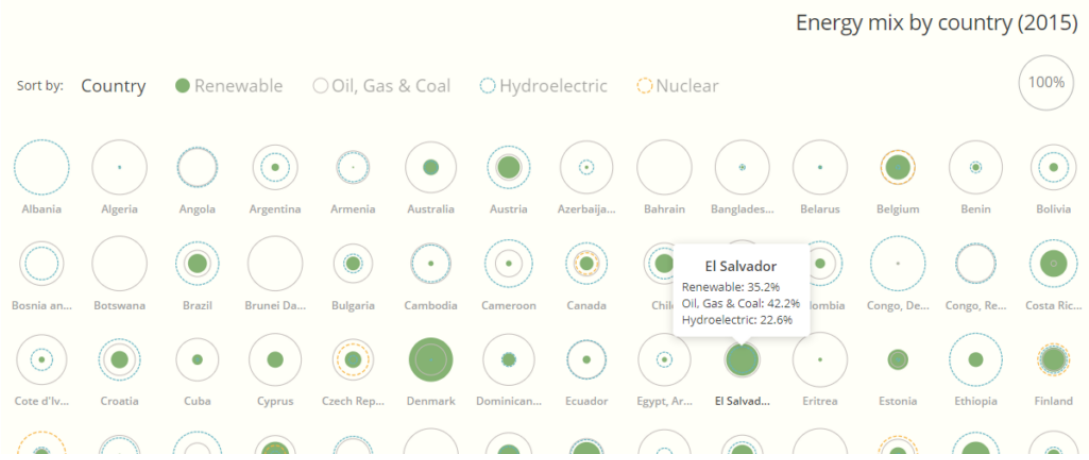

Figure 1. Energy Explorer

As you read this book you'll learn a particular aspect of D3 and see it in the context of Energy Explorer.

The book is divided into several chapters. Some contain theoretical information and others cover the build of Energy Explorer. Each chapter on the build has the name 'Build' in its title, for example 'Build: Draw the Data'.

1.1. Prerequisites

To get the most out of this book I recommend that you're familiar with:

- HTML
- SVG
- CSS
- JavaScript
- Node.js
- the command line (Terminal on MacOS or Command Prompt on Windows).

If you need to get up to speed on the above languages I suggest reading my Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation book.

You should also be comfortable opening and editing code in a code editor such as VS Code.

You also need Node.js version 18+ or 20+ installed on your computer. This is necessary for the Vite build tool.

See Appendix A Tools and Set-up for help on Node and command line interfaces.

1.2. What you'll learn

This book teaches you enough D3 to build a **custom data visualisation**.

Broadly speaking you'll learn about:

- **D3 selections and joins** (these are used to update HTML and SVG elements in a data-driven fashion)
- **D3 modules** (such as scale functions and transitions)
- **Third party libraries** such as Lodash that are useful when building data visualisations
- General **web development patterns** that are helpful when building interactive data visualisations.

1.2.1. D3 selections and joins

D3 has a powerful mechanism for adding, removing and updating HTML/SVG elements in a data-driven fashion known as selections and joins. You'll learn how to use **selections** to select and update HTML/SVG elements.

You'll also learn how to update the elements in a data-driven fashion using **joins**. For example, given an array of data, you'll learn how to keep a group of SVG circles synchronised with the data.

1.2.2. D3 modules

D3 has a large number of modules that help you build data visualisations. We cover the following modules in this book:

Name	Description
d3-selection	Add, remove and modify HTML and SVG elements in a data-driven fashion.
d3-transition	Animate the position, size and colour of HTML and SVG elements.
d3-scale	Tansform data values (e.g. percentages) into visual values (such as position, size or colours).
d3-fetch	Load CSV and JSON files into your visualisation.

1.2.3. Third party libraries

Besides D3 we use another couple of JavaScript libraries in this book: the Tippy.js tooltip library and Lodash.

Tippy.js lets you add a tooltip to a web page and Lodash provides functions for processing data (such as sorting and filtering).

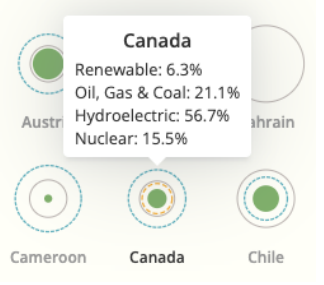

Figure 2. Tippy.js tooltip (appears when an item is hovered)

1.2.4. Web development patterns

We also cover web development techniques that are useful when building interactive data visualisations:

Name	Description
Modularisation	Split your code up into separate modules rather than having all your code in a single file.
State management	Implement a state management pattern so that user interaction can be handled in a clean and easy to understand manner.
Data manipulation	Use the Lodash library to process (sort, filter etc.) data.

1.3. Energy Explorer

In this book we show how an interactive data visualisation called **Energy Explorer** is built. This shows the energy mix of 141 countries. Visit live version.

Each country is represented by 4 circles and each circle shows the amount of **renewable**, **fossil fuel**, **hydroelectric** and **nuclear** energy produced by a country.

Figure 3. Energy Explorer

The chart is interactive. When a country is hovered a tooltip appears showing detailed information.

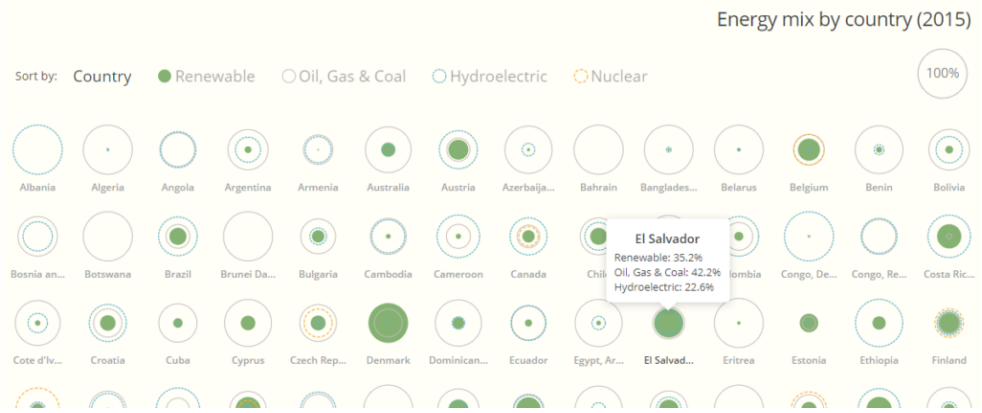

Figure 4. Information tooltip when a country is hovered

The countries can be sorted by name or one of the energy types. When the circles sort they animate into new positions. The following image shows the view when Renewable has been selected. The countries are sorted by the percentage of renewable energy in their energy mix:

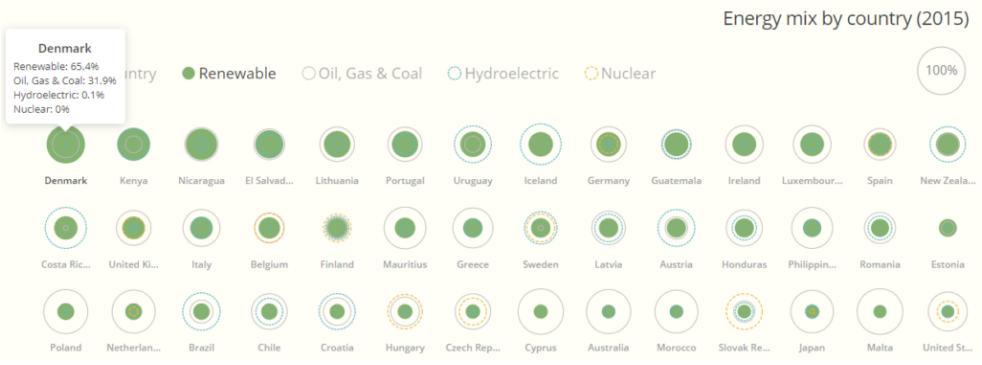

Figure 5. Countries sorted by renewable energy

The visualisation is styled to look clean, modern and fresh. Colours, fonts, line widths, spacing etc. have been carefully selected to make the chart attractive and easy to digest.

Energy Explorer is built using **HTML**, **SVG**, **CSS**, **JavaScript** and **D3**. HTML, SVG, CSS and JavaScript are standard technologies used to build websites and web applications. Practically all websites and web applications are built from these four technologies.

1.4. Code download

To download the code for this book see the [getting-started] chapter.

1.5. Deployment

If you wish to deploy Energy Explorer visit Appendix B [appendix-b].

1.6. Bonus material

The standard Energy Explorer is designed to be as easy to understand as possible. For this reason, it doesn't adapt to device width and it doesn't have a dark theme option. However I've included two appendices that show how Energy Explorer can be made responsive and how a dark theme can be added. There's also an appendix showing how the original data from the World Bank is transformed into CSV files that are loaded by Energy Explorer.

1.7. Where to get help

If you need help on anything in this book please tweet me at @createwithdata or email me at help@createwithdata.com.

1.8. Stay in touch

I love to stay in touch with my readers. One of the best ways to do this is via my mailing list. I send occasional messages containing useful information related to implementing data visualisations (e.g. using JavaScript or other tools). There'll also be discount codes for my other books. You can sign up on my website www.createwithdata.com/newsletter.

1.9. Translators

If you're interested in translating D3 Start to Finish to another language contact me at info@createwithdata.com and I'll be happy to discuss further.

1.10. Acknowledgements

Thank you to David Specht for spotting a mistake relating to the parameters passed into D3 event handlers.

Chapter 2. D3 Selections

D3 selections are basically **arrays of HTML or SVG elements**. They let you **modify the style and attributes** of the elements you've selected.

For example you can use D3 to select some SVG circles and change their colour to red. Selections are also used when **joining an array** to HTML or SVG elements (see next chapter).

2.1. Creating a selection

D3 has two functions for making selections: `select` and `selectAll`. Both of these are defined in D3's `d3-selection` module.

Name	Description
`select`	Select a single element
`selectAll`	Select multiple elements

We import `select` and `selectAll` from the `d3-selection` module using:

```
import { select, selectAll } from 'd3-selection';
```

For brevity I don't always include the `import` statement in code samples.

2.1.1. select

The syntax of `select` is:

```
select(selector)
```

where `selector` is a string containing a CSS selector string (for example `'#chart'`).

> If you're not familiar with CSS selectors take a look at the CSS section in Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation.

`select` selects the **first** element on the page that matches the selector string. It returns an object that has a number of methods (such as `attr` and `style`) that modify the selected HTML/SVG element.

For example, if you have some SVG circles on the page:

```
<circle></circle>
<circle></circle>
<circle></circle>
```

the following statement makes a selection containing the first circle:

```
select('circle');
```

You can assign a selection to a variable. For example:

```
let s = select('circle');
```

Remember we must import `select` before using it:

```
import { select } from 'd3-selection';
let s = select('circle');
```

2.1.2. selectAll

`selectAll` is similar to `select` but it selects **all** matching elements on the page. For example, if you have some SVG circles on the page:

```
<circle></circle>
<circle></circle>
<circle></circle>
```

the following statement makes a selection containing all three circles:

```
selectAll('circle');
```

`selectAll` returns an object that has a number of methods (such as `attr` and `style`) that modify all the selected HTML/SVG elements. The next section covers these methods in detail.

2.2. Updating a selection's elements

Once you've used `select` or `selectAll` to create a selection of HTML/SVG elements you can **modify the elements** using a number of methods. Four of the most commonly used methods are `style`, `attr`, `classed`, `text` and `html`:

Method name	Description
`style`	Adds CSS rules to the selection's elements
`attr`	Adds attributes to the selection's elements
`classed`	Adds a class attribute to the selection's elements
`text`	Sets the text content of the selection's elements
`html`	Sets the content of the selection's elements to an HTML string

2.2.1. style

The `style` method **sets a CSS style property** on each element in a selection. The syntax is:

```
style(property, value)
```

`property` is a string representing the CSS property (such as `'color'`, `'fill'` or `'font-size'`) and `value` is the value you're setting the property to (such as `'red'`, `'#ddd'` or `'12px'`).

Suppose you have a selection of three circle elements:

```
let s = selectAll('circle');
```

You can update the `fill` style of each circle using:

```
s.style('fill', 'red');
```

This will change the fill colour of each circle to `red`.

An alternative format that omits the variable declaration is:

```
selectAll('circle')
  .style('fill', 'red');
```

Each method call is on a separate line and indented. This is the usual way to format D3 code.

Under the hood, D3 adds an attribute named `style` to each element:

```
<circle r="10" style="fill: red;"></circle>
<circle r="20" style="fill: red;"></circle>
<circle r="30" style="fill: red;"></circle>
```

If you need to add more than one style rule, you can **chain** `style` calls:

```
selectAll('circle')
  .style('fill', 'red')
  .style('stroke', '#333');
```

Method chaining is a technique where you can make multiple method calls by writing them one after the other. So instead of writing:

```
let s = selectAll('circle');
s.style('fill', 'red');
s.style('stroke', '#333');
```

you write:

```
selectAll('circle')
  .style('fill', 'red')
  .style('stroke', '#333');
```

Here's a CodePen example that demonstrates `style`:

https://codepen.io/createwithdata/pen/abvYowg.

> The CodePen examples import the entire D3 library so they use `d3.selectAll` instead of `selectAll`.

As an exercise, try adding another `style` call to change the `stroke` of the circles to `#333`.

2.2.2. attr

`.attr` **sets an attribute** on each element of a selection. The syntax is:

```
attr(name, value)
```

`name` is a string representing the attribute name (for example `'id'`, `'cx'` or `'width'`) and `value` is the value you're setting the attribute to (for example `'main-menu'`, `10` or `100`).

Suppose you have three `<circle>` elements. You can update the `r` attribute of each circle using:

```
selectAll('circle')
  .attr('r', 50);
```

This will change the radius of each circle to 50:

```
<circle r="50"></circle>
<circle r="50"></circle>
<circle r="50"></circle>
```

As with `style` (and all other selection methods) you can chain method calls:

```
selectAll('circle')
  .attr('cx', 200)
  .attr('cy', 100)
  .attr('r', 50);
```

Here's a CodePen example that uses `attr` to set all the circle radii to 30:

https://codepen.io/createwithdata/pen/GRZWPjM

As an exercise, try using `attr` to change each circle's vertical position to 30. Hint: use the `cy` attribute.

2.2.3. classed

You can **add and remove class attributes** to or from a selection's elements using `classed`. The syntax is:

```
classed(className, value)
```

`className` is the name of the class you're adding or removing and `value` indicates whether you're adding or removing the class attribute. If `value` is `true` the class is added to the element. If `value` is `false` the class is removed from the element.

For example to add a class attribute named `highlighted` to each element in a selection use:

```
selectAll('circle')
  .classed('highlighted', true);
```

To remove a class attribute named `highlighted` use:

```
selectAll('circle');
  .classed('highlighted', false);
```

You can add more than one class attribute by separating the class names with a space:

```
selectAll('circle');
  .classed('item highlighted', true);
```

Under the hood `classed` adds a class attribute to a selection's element(s):

```
<circle class="item highlighted"></circle>
```

Here's a CodePen example that uses `classed` to add a class attribute named `highlighted` to each circle: https://codepen.io/createwithdata/pen/WNwpLog

In this CodePen example there's a CSS rule that sets the fill colour for elements with class `highlighted`.

2.2.4. text

The `text` method lets you **set the text content** of each element in a selection. It's generally used on selections of HTML elements (such as `div`, `p` and `span`) and SVG `text` elements. The syntax is:

```
text(value)
```

where `value` is the string you're setting the content to.

Suppose you have two `<div>` elements on your page. You can set the text content of each element using:

```
selectAll('div')
  .text('Some content');
```

This results in the following change to the HTML:

```
<div>Some content</div>
<div>Some content</div>
```

Here's an example in CodePen: https://codepen.io/createwithdata/pen/ExKWGWj

2.2.5. html

The `html` method lets you **set the content** of each element in a selection to a snippet of HTML. The syntax is:

```
html(value)
```

where `value` is a string of HTML.

Suppose you have two `<div>` elements on your page. You can set the content of each element using:

```
selectAll('div')
  .html('<p>Paragraph 1</p><p>Paragraph 2</p>');
```

This results in the following change to the HTML:

```
<div>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</div>
<div>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</div>
```

Here's an example in CodePen: https://codepen.io/createwithdata/pen/dyxpoKQ

Each menu item in Energy Explorer consists of a circle and a label. To save us having to write a nested D3 join we use the `html` method to generate a snippet of HTML consisting of an SVG circle and a text label.

2.3. Multiple updates

The `style`, `attr`, `classed` and `text` methods may be chained together which allows you to set multiple style, attribute and classes in a single statement. For example:

```
selectAll('circle')
  .attr('cx', 100)
  .attr('cy', 100)
  .attr('r', 50)
  .style('fill', 'red')
  .classed('item', true);
```

2.3.1. .select and .selectAll

The `style`, `.attr`, `classed` and `text` methods can be used on selections created using `select` and `selectAll`. All the above examples use `selectAll`.

If `select` is used, only the first matching element will be selected. Try changing `selectAll` to `select` in the CodePen examples and you should see that only the first element on the page is modified.

2.4. Chained selections

You can make a selection of elements **within another selection** by **chaining** calls to `select` and `selectAll`. This concept is best explained by an example. Suppose you have the HTML:

```
<div class="first">
  <p></p>
  <p></p>
</div>
<div class="second">
  <p></p>
  <p></p>
</div>
```

You can select the `p` elements inside the first `div` by selecting the `div` and chaining a call to `selectAll`:

```
select('div.first')
  .selectAll('p');
```

Similarly you can select the `p` elements from within the second `div` using:

```
select('div.second')
  .selectAll('p');
```

Here's a Codepen containing these examples:

https://codepen.io/createwithdata/pen/xxZYvBZ

You might be wondering whether you can achieve something similar using a nested CSS selector. For example, you might think:

```
selectAll('div.second p');
```

is equivalent to:

```
select('div.second')
  .selectAll('p');
```

They’re similar but there is a difference: when selections are chained **the selection keeps a record of the parent element**. In the previous code snippet the selection keeps a record of the parent element (`div.second`) of the selected `p` elements.

This will come in useful when joining data. For now, don’t worry too much about this, but just remember that selections can be chained.

Appendix A: Tools and Set-up

This Appendix helps you get set up so you can run Energy Explorer on your computer.

You need:

- a code editor
- a web browser
- a command line interface
- Node.js

A.1. Code editor

A code editor is a desktop application that lets you edit HTML, CSS and JavaScript files. I recommend using Visual Studio Code which is free and multi-platform.

A.2. Web browser

A modern web browser such as Chrome, Firefox, Safari or Edge is recommended for web development. Create With Data books assume that Google Chrome is being used. The majority of material will apply regardless of your browser choice but there may be differences if the developer tools are used, as these differ from browser to browser.

A.3. Command line interface

A command line interface (CLI) lets you interact with your operating system via text commands. When reading this book you'll use your CLI to navigate to a particular build step and run Energy Explorer.

If you're on a Mac you can use the Terminal application and on Windows the Command Prompt.

For this book you don't need to know many commands at all. The most useful will be `cd` which lets you navigate to a particular directory. Other commands are presented as you read this book.

For an introduction to the CLI I recommend this tutorial.

A.4. Node.js

JavaScript was originally designed for web browsers but in 2009 Node.js was launched which lets you run JavaScript code outside of a browser. This means you can write a JavaScript program and run it on your computer, much like you'd run a Python or R program. For example you can use Node.js to process data or to make a web server.

Nowadays Node.js can be used to support you when developing a web application. Even if you're developing a purely front-end application (such as Energy Explorer) Node.js can help you develop and deploy your application. In this book we use Vite which runs on Node.js. It provides a local web server, supports the use of modules and builds an optimised package that you can deploy to the web.

> The previous edition of this book required the set up of a local webserver. This edition uses Vite which provides its own webserver.

A.4.1. Installing Node.js

First check whether you already have Node.js installed. Open a CLI (for example Terminal on Mac or the Command Prompt on Windows) and type `node`. If you get a 'Welcome to Node.js' message you already have Node.js installed. If not you need to install Node.js.

This book uses Vite which requires Node.js versions 18 or 20. If you have a different version installed you probably need to use nvm (see later). If you're using a work machine **please** check with someone who understands your setup!

There's a few alternatives for installing Node.js on your computer and the approach you choose depends on your specific situation. Also bear in mind if you're using a computer provided by your workplace there might be security measures in place which limit what you can install.

The standard approach to install Node.js is to visit the Node.js website where you can download the Node.js installer. If you're using a Mac and have Homebrew installed you can use `brew install node`. Lastly there's a tool called nvm that lets you install multiple versions of Node.js. You can find instructions on how to use it at https://github.com/nvm. Personally I use nvm and find it reliable.