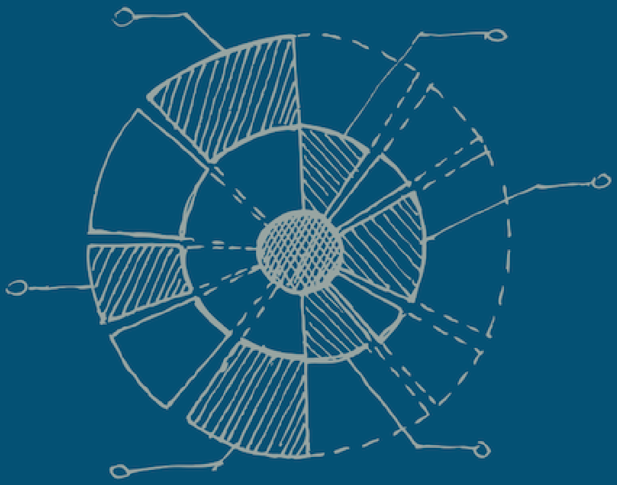
CREATE WITH DATA

DATA DASHBOARDS WITH JAVASCRIPT



PETER COOK

Data Dashboards with Javascript

Peter Cook

Data Dashboards with JavaScript

This version was published on 7th February 2025.

This is a preview version that includes the first and second chapters.

Chapter 1. Introduction

Data Dashboards with Javascript teaches you how to make maps, charts and data dashboards with JavaScript. You'll learn about:

- JavaScript's role in data visualisation
- how to create charts and maps using JavaScript libraries such as Chart.js, D3.js and Leaflet
- how to combine maps and charts to create data dashboards.

You'll learn how to make charts using Chart.js (Figure 1), maps using Leaflet (Figure 2) and data dashboards using HTML, CSS, JavaScript, Chart.js and Leaflet (Figure 3).

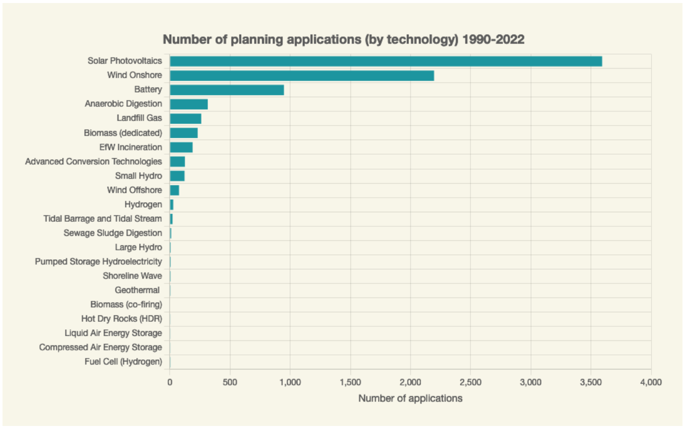
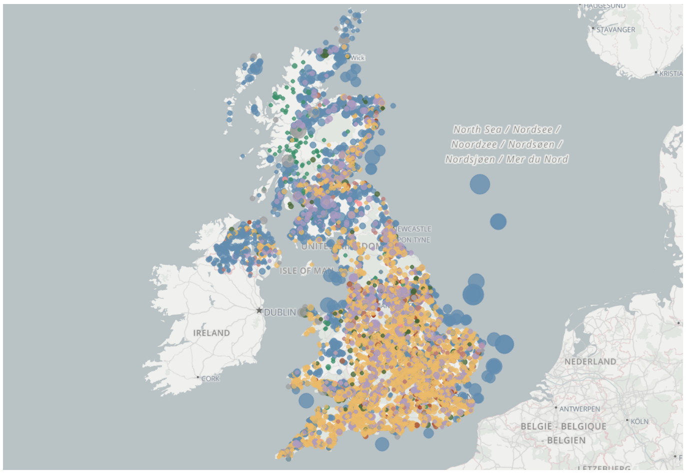


Figure 1. Chart.js bar chart



STAVANGER
North Sea / Nordsee / Noordzee / Nordsøen / Nordsjøen / Mer du Nord
NEWCASTLE UPON TYNE
ISLE OF MAN
DUBLIN
IRELAND
CORK
NEDERLAND
ANTWERPEN
KÖLN
BELGIË - BELGIQUE - BELGIEN

Figure 2. Leaflet map

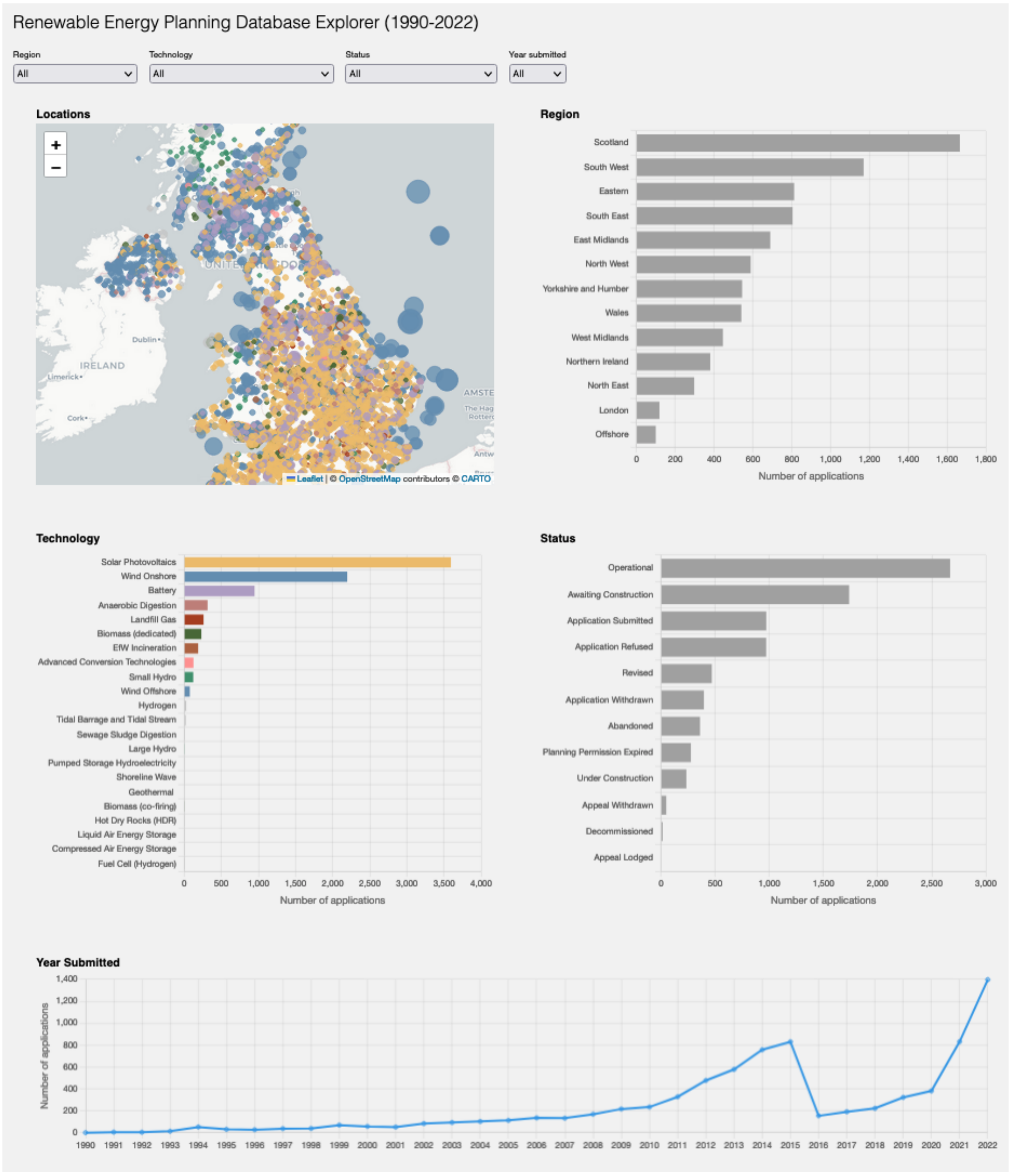


Figure 3. Data dashboard

1.1. Prerequisites

To get the most out of this book you need:

- access to a computer with Node.js installed and a code editor (such as VS Code)
- moderate experience with HTML, CSS and JavaScript
- moderate experience with the command line, Node.js and npm

If you've built (or worked on) a web application you should be fine. Knowledge of JavaScript arrow functions, callback functions and promises is useful for this book. If you need to get up to speed with HTML, CSS, SVG and/or JavaScript I recommend reading Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation (also by myself). This contains all the necessary background knowledge for this book.

1.2. Setting expectations

This book aims to get you up and running with Chart.js, Leaflet and React and shows you how to make a data dashboard.

It covers the fundamentals of making maps and charts as well as how to get started using each of these libraries. It doesn't go into great detail about the libraries but seeks to motivate you to learn more about them and empower you to make your own data visualisations.

1.3. The code

This book covers several example projects. Download a zip of the code from (link hidden in preview version). Unzip the file and move the directory to your usual development directory.

1.4. The data

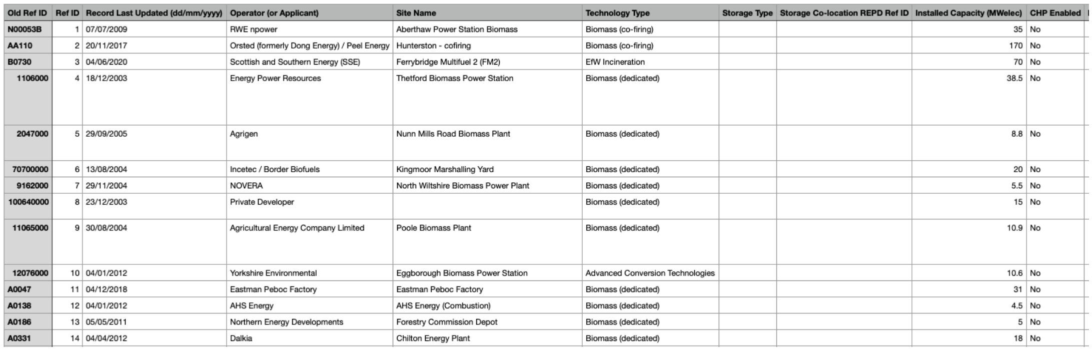
This book uses a single dataset from the UK government's Renewable Energy Planning Database (https://www.gov.uk/government/publications/renewable-energy-planning-database-monthly-extract). It tracks UK renewable energy projects through the planning system from 1990 onwards.

We use a snapshot of the dataset published in January 2023 that contains just over 8000 rows. Each row represents a planning application and has fields including name, technology (the type of technology such as solar or wind), capacity (the output capacity in megawatts), location (latitude and longitude), region, status (e.g. operational, refused, under construction) and year submitted.

We use this data because:

- it's real data that's published by a reputable source
- it has a mix of categorical and numeric data
- it has a geographic element to it (the location of each project)
- it's a good size for educational purposes (it's not tiny nor is it massive)

The original data looks like:



Old Ref ID	Ref ID	Record Last Updated (dd/mm/yyyy)	Operator (or Applicant)	Site Name	Technology Type	Storage Type	Storage Co-location REPD Ref ID	Installed Capacity (MWelec)	CHP Enabled
N00053B	1	07/07/2009	RWE npower	Aberthaw Power Station Biomass	Biomass (co-firing)			35	No
AA110	2	20/11/2017	Orsted (formerly Dong Energy) / Peel Energy	Hunterston - cofiring	Biomass (co-firing)			170	No
B0730	3	04/06/2020	Scottish and Southern Energy (SSE)	Ferrybridge Multifuel 2 (FM2)	EfW Incineration			70	No
1106000	4	18/12/2003	Energy Power Resources	Thetford Biomass Power Station	Biomass (dedicated)			38.5	No
2047000	5	29/09/2005	Agrigen	Nunn Mills Road Biomass Plant	Biomass (dedicated)			8.8	No
70700000	6	13/08/2004	Incetec / Border Biofuels	Kingmoor Marshalling Yard	Biomass (dedicated)			20	No
9162000	7	29/11/2004	NOVERA	North Wiltshire Biomass Power Plant	Biomass (dedicated)			5.5	No
100640000	8	23/12/2003	Private Developer		Biomass (dedicated)			15	No
11065000	9	30/08/2004	Agricultural Energy Company Limited	Poole Biomass Plant	Biomass (dedicated)			10.9	No
12076000	10	04/01/2012	Yorkshire Environmental	Eggborough Biomass Power Station	Advanced Conversion Technologies			10.6	No
A0047	11	04/12/2018	Eastman Peboc Factory	Eastman Peboc Factory	Biomass (dedicated)			31	No
A0138	12	04/01/2012	AHS Energy	AHS Energy (Combustion)	Biomass (dedicated)			4.5	No
A0186	13	05/05/2011	Northern Energy Developments	Forestry Commission Depot	Biomass (dedicated)			5	No
A0331	14	04/04/2012	Dalkia	Chilton Energy Plant	Biomass (dedicated)			18	No

Figure 4. REPD data (some rows and columns omitted)

For the purposes of this book I've transformed the data into a JSON file. A subset of the original data's columns are used and the columns are renamed. Some new properties `yearSubmitted`, `yearGranted` and `daysUntilGranted` have also been added:

Listing 1. First few items in the JSON file

```
[
  {
    "id": "1",
    "name": "Aberthaw Power Station Biomass",
    "technology": "Biomass (co-firing)",
    "capacity": 35,
    "lon": -3.407,
    "lat": 51.387,
    "region": "Wales",
    "status": "Operational",
    "operator": "RWE npower",
    "yearSubmitted": 2004,
    "dateSubmitted": "2004-04-16T00:00:00.000Z",
    "yearGranted": 2004,
    "daysUntilGranted": 140
  },
  {
    "id": "2",
    "name": "Hunterston - cofiring",
    "technology": "Biomass (co-firing)",
    "capacity": 170,
    "lon": -4.889,
    "lat": 55.736,
    "region": "Scotland",
    "status": "Application Withdrawn",
    "operator": "Orsted (formerly Dong Energy) / Peel Energy",
    "yearSubmitted": 2010,
    "dateSubmitted": "2010-06-02T00:00:00.000Z",
    "yearGranted": null,
    "daysUntilGranted": null
  },
  {
    "id": "3",
    "name": "Ferrybridge Multifuel 2 (FM2)",
    "technology": "EfW Incineration",
    "capacity": 70,
    "lon": -1.282,
    "lat": 53.716,
    "region": "Yorkshire and Humber",
    "status": "Operational",
    "operator": "Scottish and Southern Energy (SSE)",
    "yearSubmitted": 2014,
    "dateSubmitted": "2014-08-20T00:00:00.000Z",
    "yearGranted": 2015,
    "daysUntilGranted": 434
  },
  ...
]
```

You can inspect the data for yourself in the code download at `load-data/public/repd.json`.

For more information about converting the REPD CSV file into `repd.json` visit the Data Wrangling section in the Appendix at the end of this book.

Some data items have missing data, this is quite common for real world datasets. Missing data values are represented using `null`.

Viewing JSON files

I use a few methods to view JSON files. A decent text editor such as VS Code is a good way to view text files, even if they are over 100 megabytes. `repd.json` is just under 3 megabytes so can be comfortably viewed in VS Code. Another approach is use the command line tool `less` which is a bit quicker than VS Code for really big files. If it's installed on your system type `less <filename>` in the command line. Another approach I use if I already have a web application running is to add `console.log(data)` to the code. This outputs the data to the browser's developer tools console where you can easily browse through the data.

CSV or JSON?

Two of the most common data file formats are CSV (comma separated value) and JSON (JavaScript object notation). CSV is a tabular format that's often used when exporting data from a spreadsheet. It's basically a text representation of the spreadsheet where each line in the file represents a row in the spreadsheet. Each line contains values separated by a comma character and each value corresponds to a column in the spreadsheet.

JSON derives from JavaScript and is basically a JavaScript object (or array) represented as a string of text. It's a more natural format when working with JavaScript. For example we can use the built-in function `JSON.parse` to construct an object from a JSON string that's been loaded in from a text file. (We need additional libraries to do the same with CSV.)

1.5. Book Structure

The book is organised into the following chapters:

- JavaScript and Data Visualisation
- Make a Chart with Chart.js
- Styling and Customising Chart.js
- Data-driven Maps with Leaflet
- Data Dashboards
- Client/Server Data Dashboards

The **JavaScript and Data Visualisation** chapter gives a brief overview of data visualisation and JavaScript's role in data visualisation. It also covers how to create a web application using the build tool Vite and walks through an example application that loads some data and displays it as a table.

The **Make a Chart with Chart.js** chapter introduces Chart.js and shows how to make a simple bar chart. It introduces the tidy.js JavaScript library which is used for processing and analysing data. The chapter closes by walking through a Chart.js example.

The **Styling and Customising Chart.js** chapter shows how to style Chart.js charts. It also shows how to create multi-series charts such as multi-line charts and stacked bar charts.

The **Data-driven Maps with Leaflet** chapter introduces the mapping library Leaflet and shows how to draw data points on a map.

The **Data Dashboard** chapter shows how to create a statically-served data dashboard containing a map and charts.

The **Client/Server Dashboard** chapter shows to build an API so that the data processing for the data dashboard can be hosted on a separate server.

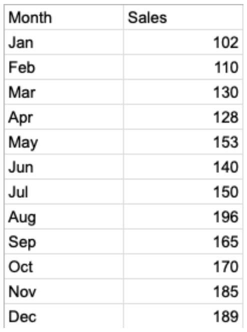
Chapter 2. Data Visualisation and JavaScript

This chapter gives an overview of JavaScript's role in data visualisation. It also shows how to create a web application using the build tool Vite and walks through an example application that loads some data and displays it as a table.

2.1. Data Visualisation

Data visualisation is the process of representing data using graphics (rather than numbers).

For example, suppose you have a table of sales data:



Month	Sales
Jan	102
Feb	110
Mar	130
Apr	128
May	153
Jun	140
Jul	150
Aug	196
Sep	165
Oct	170
Nov	185
Dec	189

Figure 5. A table of sales data

It's hard to spot any trends in the data just by looking at the numbers. A data analyst might run a regression test to see if there's a trend but you can also visualise the data by plotting a line chart:

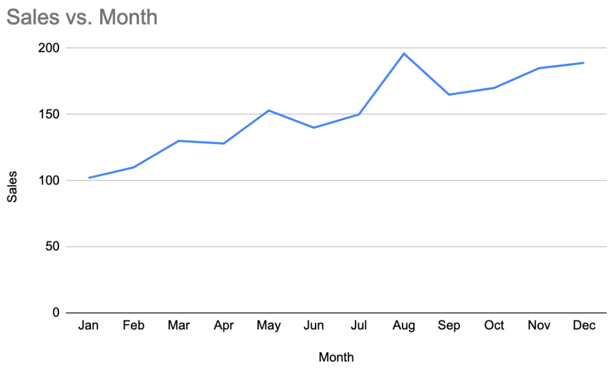


Figure 6. The sales data represented with a line chart

We can immediately see there's an upward trend and that there was a peak in August. This is the power of data visualisation. It reveals patterns that aren't apparent in a table of numbers.

2.1.1. Why visualise data?

There’s plenty of reasons to visualise data such as:

- it’s an effective approach to understanding and communicating data
- it can offer increased visual variety that helps engage the reader
- it can be a visually appealing work in its own right (good for content, SEO etc.)

2.1.2. Best practice

There’s plenty of books about data visualisation best practice. One that springs to mind is Alberto Cairo’s The Functional Art. This has a handy chapter that explains which charts are most effective at conveying data based on academic understanding of human perception. In short it’s best to use position and length to represent data (rather than direction, area and/or shade) because we are able to perceive differences in position and length better than other encodings. This is one of the reasons bar, line and scatter charts are so prevalent.

That’s not to say non-standard charts are ineffective. So long as the important quantities are represented using position and length custom charts can also be effective at conveying data.

2.2. JavaScript

JavaScript is the programming language of the web and is implemented in all major web browsers. It has a similar syntax to C-style programming languages such as C, C++, Java, PHP and is commonly used to add interactivity to web sites and applications.

It’s also used to add and modify content within a web page. For example it can be used to add interactive charts to a web page.

2.2.1. Why visualise data using JavaScript?

There’s several reasons why JavaScript is one of the leading (if not the leading) language for visualising data:

- it’s the only language implemented in major web browsers
- it’s well suited to working with data
- it has plenty of data visualisation libraries including Chart.js, Leaflet and D3.js.

2.2.2. History of JavaScript and Data Visualisation

JavaScript charting libraries have existed almost as long as JavaScript itself (which was introduced in 1995)[1]. As far as I can tell the earliest ones that are still maintained are Google Charts and Flot which were both introduced around 2007 and support standard charts such as bar charts, line charts and pie charts.

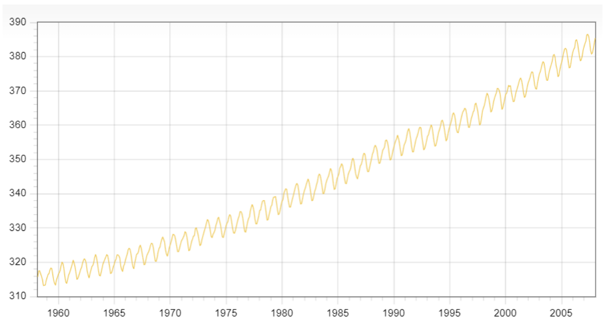


Figure 7. A line chart created with Flot

In 2009 Highcharts and Protovis were launched. Highcharts is still one of the leading JavaScript charting libraries (and is free for non-commercial use) and provides standard charts. Protovis took a different approach. Rather than providing standard charts it provided graphical elements (such as bars, lines and dots) with which you can encode your data. You can combine these elements to create custom charts. Protovis was led by Mike Bostock and Jeff Heer who went on to create D3.js.

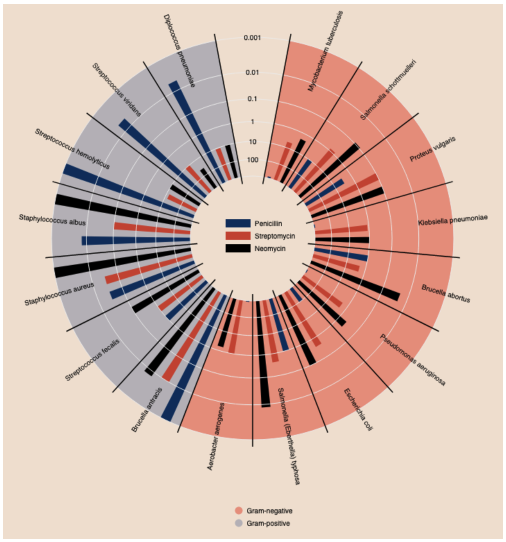


Figure 8. Custom chart created with Protovis

In the mid-2000s Adobe’s Flash was the dominant platform for creating rich interactive applications for the web. Its popularity declined towards the end of the 2000s as web standards such as SVG and Canvas started to be supported by the major browsers.

2011 saw the launch of D3.js which embraced the growing support for the SVG web standard. Like Protovis it provides building blocks for making charts and it became the dominant JavaScript library for creating custom visualisations.

In 2014 Chart.js launched and is currently one of the leading free and open-source standard charting libraries.

2.2.3. JavaScript and Data Visualisation today

The most popular data visualisation projects on Github are currently D3.js (105,000 stars) and Chart.js (60,300 stars). My personal opinion is that D3.js and Chart.js are able to cover most visualisation cases. In general I recommend using Chart.js for standard charts (bar, line, pie and scatter) and D3.js for custom visualisations.

2.3. Creating a Web Application with JavaScript

This section shows how to create a JavaScript web application. It starts by showing the fundamental components of a web application then introduces the tool Vite which helps you join all the pieces together.

Fundamentally a web application consists of HTML, CSS and JavaScript. This is true whether the application is a simple website or a fully featured e-commerce store or social media platform. If you’re not familiar with HTML, CSS or JavaScript I recommend finding a tutorial on the web such as this tutorial at mozilla.org or reading my book Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation.

2.3.1. HTML

HTML (Hypertext Markup Language) describes the static content of the application. Typically this is content that is always visible on the webpage or container elements for JavaScript generated content (such as charts).

Here’s some sample HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>...</head>
<body>
  <h1>My app</h1> ①
  <div id="menu"></div>
  <div class="content">
    <div id="chart"></div> ②
  </div>
</body>
</html>
```

① Some static content

② A container element for a chart that’ll be generated by JavaScript

2.3.2. CSS

CSS (Cascading Style Sheets) describes the layout and style of the app. For example:

```css
body {
  background-color: #eee;
}

h1 {
  color: blue;
}

.content {
  padding: 1rem;
}

...
```

2.3.3. JavaScript

JavaScript code manages interaction and dynamic content. Dynamic content is content that’s generated after the web page has loaded. It augments the static content in the main HTML file.

Here’s a basic example that populates the `<div id="chart">` container element in the HTML file with a Chart.js chart (there’s no need to understand this code at this stage):

```javascript
let chart = new Chart('chart', {
  type: 'bar',
  data: {
    labels: ['A', 'B', 'C'],
    datasets: [
      {
        data: [10, 20, 30]
      }
    ]
  }
});
```

This code results in Chart.js drawing a bar chart inside the `<div id="chart">` element.

If these HTML, CSS or JavaScript snippets look totally alien to you I recommend getting up to speed using Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation.

2.3.4. Joining the pieces

It’s possible to link to each a project’s CSS and JavaScript files using `link` and `script` tags in the HTML file but the usual approach (especially with medium and large projects) is to use a **bundler**. A bundler is a tool that combines multiple JavaScript files into a single bundle.

One of the main advantages is your JavaScript can be modularised meaning that you can split your application into smaller files, each of which has a single purpose. This makes your application code much easier to understand and navigate.

There’s several bundlers available such as Webpack, Rollup, Parcel and Vite. In this book we use Vite because:

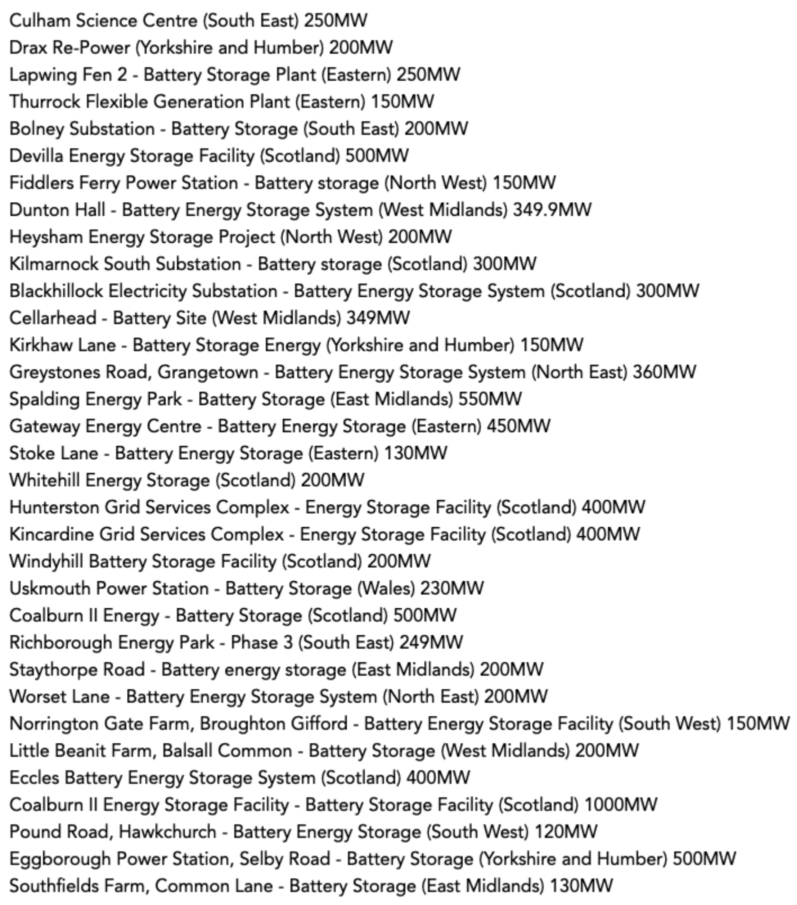
- you can create a project with minimal configuration
- it has a built-in web-server for when you’re developing your application
- it’s fast
- its web-server serves static files (which is useful for loading in CSV and JSON files)

In general I recommend using Vite for any size web application.

All of the projects in this book are created using Vite. Creating a project with Vite is relatively straightforward but there’s no need to do this yourself for the purposes of this book. However if you’d like to know how to create a project with Vite refer to Appendix A.

2.3.5. Example web application

Our first project loads the REPD data, filters it then displays it as a list:



Culham Science Centre (South East) 250MW
Drax Re-Power (Yorkshire and Humber) 200MW
Lapwing Fen 2 - Battery Storage Plant (Eastern) 250MW
Thurrock Flexible Generation Plant (Eastern) 150MW
Bolney Substation - Battery Storage (South East) 200MW
Devilla Energy Storage Facility (Scotland) 500MW
Fiddlers Ferry Power Station - Battery storage (North West) 150MW
Dunton Hall - Battery Energy Storage System (West Midlands) 349.9MW
Heysham Energy Storage Project (North West) 200MW
Kilmarnock South Substation - Battery storage (Scotland) 300MW
Blackhillock Electricity Substation - Battery Energy Storage System (Scotland) 300MW
Cellarhead - Battery Site (West Midlands) 349MW
Kirkhaw Lane - Battery Storage Energy (Yorkshire and Humber) 150MW
Greystones Road, Grangetown - Battery Energy Storage System (North East) 360MW
Spalding Energy Park - Battery Storage (East Midlands) 550MW
Gateway Energy Centre - Battery Energy Storage (Eastern) 450MW
Stoke Lane - Battery Energy Storage (Eastern) 130MW
Whitehill Energy Storage (Scotland) 200MW
Hunterston Grid Services Complex - Energy Storage Facility (Scotland) 400MW
Kincardine Grid Services Complex - Energy Storage Facility (Scotland) 400MW
Windyhill Battery Storage Facility (Scotland) 200MW
Uskmouth Power Station - Battery Storage (Wales) 230MW
Coalburn II Energy - Battery Storage (Scotland) 500MW
Richborough Energy Park - Phase 3 (South East) 249MW
Staythorpe Road - Battery energy storage (East Midlands) 200MW
Worset Lane - Battery Energy Storage System (North East) 200MW
Norrington Gate Farm, Broughton Gifford - Battery Energy Storage Facility (South West) 150MW
Little Beanit Farm, Balsall Common - Battery Storage (West Midlands) 200MW
Eccles Battery Energy Storage System (Scotland) 400MW
Coalburn II Energy Storage Facility - Battery Storage Facility (Scotland) 1000MW
Pound Road, Hawkchurch - Battery Energy Storage (South West) 120MW
Eggborough Power Station, Selby Road - Battery Storage (Yorkshire and Humber) 500MW
Southfields Farm, Common Lane - Battery Storage (East Midlands) 130MW

Make sure you've downloaded and unzipped the code for this book. Refer to Chapter 1 for details.

To run the project, open your command line and navigate to `load-data` of the code download. Type `npm install` to download the project dependencies then `npm run dev` to start the project's web server:

```
cd load-data
npm install
npm run dev
```

All of the projects in this book are started in this manner: navigate to the project directory (e.g. `cd load-data`) and type `npm install` to download the project dependencies (e.g. libraries such as Chart.js). If your computer doesn't recognise `npm` make sure you have Node installed.

You only need to use `npm install` once. Type `npm run dev` to start the project's web server. Vite projects come with a built-in web server that's used during development. Whenever you edit and save a source file Vite automatically updates the application and refreshes the browser page.

When `npm run dev` is run the command line will respond with a message containing a URL to visit (usually `http://localhost:5173/`). Copy the URL and paste in your browser and you should see the list of battery planning applications from the REPD data (see previous figure).

Project overview

The project has the following file structure:

Listing 2. load-data project files

```
load-data/
  index.html
  main.js
  node_modules/
    ...
  package-lock.json
  package.json
  public/
    repd.json
  style.css
```

The main files are `index.html`, `main.js`, `style.css` and `repd.json`. This project has lots in common with the other projects in this book so it's worth getting to know this project well. I recommend viewing this project in your favourite code editor (e.g. VS Code).

Here’s a brief description of the files:

File	Description
`index.html`	The main HTML file. This contains static content and container elements for JavaScript generated content
`main.js`	The main JavaScript file. This loads the REPD data, filters it and constructs a list.
`node_modules`	Directory containing 3rd party modules (such as Chart.js). This is a standard directory that’s created by `npm install`. You don’t usually need to concern yourself with this directory so we won’t pay it any more attention.
`package-lock.json`	File created by `npm install` which logs the exact versions of the installed 3rd party modules. You don’t usually need to concern yourself with this file so we won’t pay it any more attention.
`package.json`	File created by `npm init` that contains the project configuration. For example it lists the installed 3rd party modules (such as Chart.js). You don’t usually need to concern yourself with this file so we won’t pay it any more attention.
`public`	Directory that acts as the server root directory. You can put static files (such as CSV files) here which the application can request.
`style.css`	The main CSS file.

`load-data` is a project that’s derived from Vite’s default counter project (see Appendix A). Let’s examine the files in more detail.

index.html

This is the main HTML file and is the first file that's loaded by the web browser when the site loads. Most of this file is the same as the Vite's default HTML file. Generally speaking the only changes we make to `index.html` in this book are:

- changing the content of `<title>`
- adding, removing or modifying HTML in the `<body>` section.

Listing 3. index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>REPD battery applications</title> ①
  </head>
  <body>
    <div id="output"></div> ②
    <script type="module" src="/main.js"></script> ③
  </body>
</html>
```

① The title that appears in the browser's tab is set using a `<title>` tag.

② The `<div>` element that acts as a container for the list. We populate this element in `main.js`.

③ The main JavaScript file `main.js` is included using a `<script>` tag. This line is generated by Vite and in general we don't need to touch it.

main.js

`main.js` is the only JavaScript file in this project. It loads the data and generates a list. To load the data it uses a JavaScript promise which is a fairly advanced technique. If you’re not familiar with promises don’t worry about them too much as they’re only used to load the data and this code is largely the same throughout the book.

Here’s an outline of `main.js`:

Listing 4. main.js

```
import './style.css'; ①

function makeList(data) { ... }

fetch('./repd.json') ②
  .then(res => res.json()) ③
  .then(data => makeList(data)); ④
```

① Import `style.css`. This is non-standard JavaScript (you can’t import CSS files in standard JavaScript) but is supported by Vite. We’ll see later that this approach is handy for keeping CSS files alongside related JavaScript files.

② When `main.js` runs the first command to execute is `fetch`. This requests the file `./repd.json` and returns a **promise**. A promise is an object that represents an operation (such as a data request) that will complete at some time in the future (typically this is a few hundred milliseconds in the future).

③ Call the promise’s `then` method and pass in a **callback** function that runs when the promise fulfils. (The promise is said to be fulfilled when the data has arrived.) The callback function’s first parameter `res` is an object that represents the response. We call its `json` method which converts the data into a JavaScript array and returns a new promise.

④ Call the second promise’s `then` method and pass in another callback function. This function’s first parameter `data` is the array of data. In the function body call `makeList` and pass the array in.

Don’t worry too much about the detail of this code as JavaScript promises are quite an advanced concept and it can take a while to understand them. Basically this code requests data from a URL, converts the file content into an array and calls a function that renders the data. This is pretty standard code and remains fairly similar throughout this book.

Public directory

Within a Vite application the `./` path refers to the root of the `public` directory. Therefore `./repd.json` refers to the `public/repd.json` file.

JavaScript arrow syntax

We use arrow syntax (`=>`) to define functions. This is a modern JavaScript feature that's an alternative way of defining functions. For example to define a function that adds two numbers use `(a, b) => a + b`. This is equivalent to `function(a, b) { return a + b; }`. It's more concise than using the `function` keyword and can make code easier to read, especially when defining simple callback functions. If the function accepts a single parameter you can omit the brackets. Read Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation for more information.

The array that's passed into `makeList` contains the REPD data:

Listing 5. First few items in the JSON file (data/repd.json)

```
[
  {
    "id": "1",
    "name": "Aberthaw Power Station Biomass",
    "technology": "Biomass (co-firing)",
    "capacity": 35,
    "lon": -3.407,
    "lat": 51.387,
    "region": "Wales",
    "status": "Operational",
    "operator": "RWE npower",
    "yearSubmitted": 2004,
    "dateSubmitted": "2004-04-16T00:00:00.000Z",
    "yearGranted": 2004,
    "daysUntilGranted": 140
  },
  {
    "id": "2",
    "name": "Hunterston - cofiring",
    "technology": "Biomass (co-firing)",
    "capacity": 170,
    "lon": -4.889,
    "lat": 55.736,
    "region": "Scotland",
    "status": "Application Withdrawn",
    "operator": "Orsted (formerly Dong Energy) / Peel Energy",
    "yearSubmitted": 2010,
    "dateSubmitted": "2010-06-02T00:00:00.000Z",
    "yearGranted": null,
    "daysUntilGranted": null
  },
  ...
]
```

JSON is a text format that allows you to exchange data between different systems. It's literally just a string of text. If we read a JSON file into a JavaScript program and wish to operate on it as though it's an array or object, we must first convert it into a JavaScript array or object. We do this using `JSON.parse` or the `json` method of a promise object returned by `fetch`.

Now let's look at the `makeList` function which filters the data (using JavaScript's built in `filter` method), constructs a list and displays it:

Listing 6. main.js

```
function makeList(data) {
  // Filter
  data = data.filter(d => d.technology === "Battery" && d.capacity > 100); ①

  // Build a list from the data
  let list = ''; ②
  data.forEach(d => { ③
    list += '<div>' + d.name + ' (' + d.region + ') ' + d.capacity + 'MW</div>'; ④
  });

  document.querySelector('#output').innerHTML = list; ⑤
}
```

① Use JavaScript's built-in array filter method to filter the data for Battery applications where the capacity is over 100.

② Initialise a variable named `list` and assign an empty string to it.

③ Iterate through each element of `data`.

④ Append a `div` element to `list` containing the name, region and capacity. (The `+=` operator is shorthand for `list = list + '<div>' + …`.)

⑤ Select the `<div id="output">` element defined in `index.html` and set its content to `list`. The browser will parse the string and create the necessary `div` elements to reflect the string `list`.

style.css

`style.css` contains typography settings that were retained from Vite's default `style.css` file. (`style.css` is imported at the top of `main.js`.)

```
body {
  font-family: Inter, Avenir, Helvetica, Arial, sans-serif;
  font-size: 16px;
  line-height: 24px;
  font-weight: 400;
}
```

2.4. Wrapping up

This chapter discussed the role of JavaScript when visualising data. It presented an overview of JavaScript charting libraries and divided them into two categories: those that provide a selection of standard charts and those that provide more flexibility at the cost of a steeper learning curve.

We also looked at a simple project built using Vite. The project loads the REPD data and presents a subset of it as a list. This forms the foundation for the remaining content in this book.

2.5. Ideas for further study

I recommend playing around with the project by exploring, modifying and adding code. Here are some ideas:

- display different fields (for example operator or status)
- change the filter to display a different technology type (or change the capacity threshold)
- try displaying your own data (make sure the dataset isn’t too large, perhaps don’t go over 10Mb).

Visit the Appendix for solutions.

[1] According to ChatGPT one of the earliest JavaScript charting libraries was Netscape Livewire Charts which was developed in the mid-1990s.