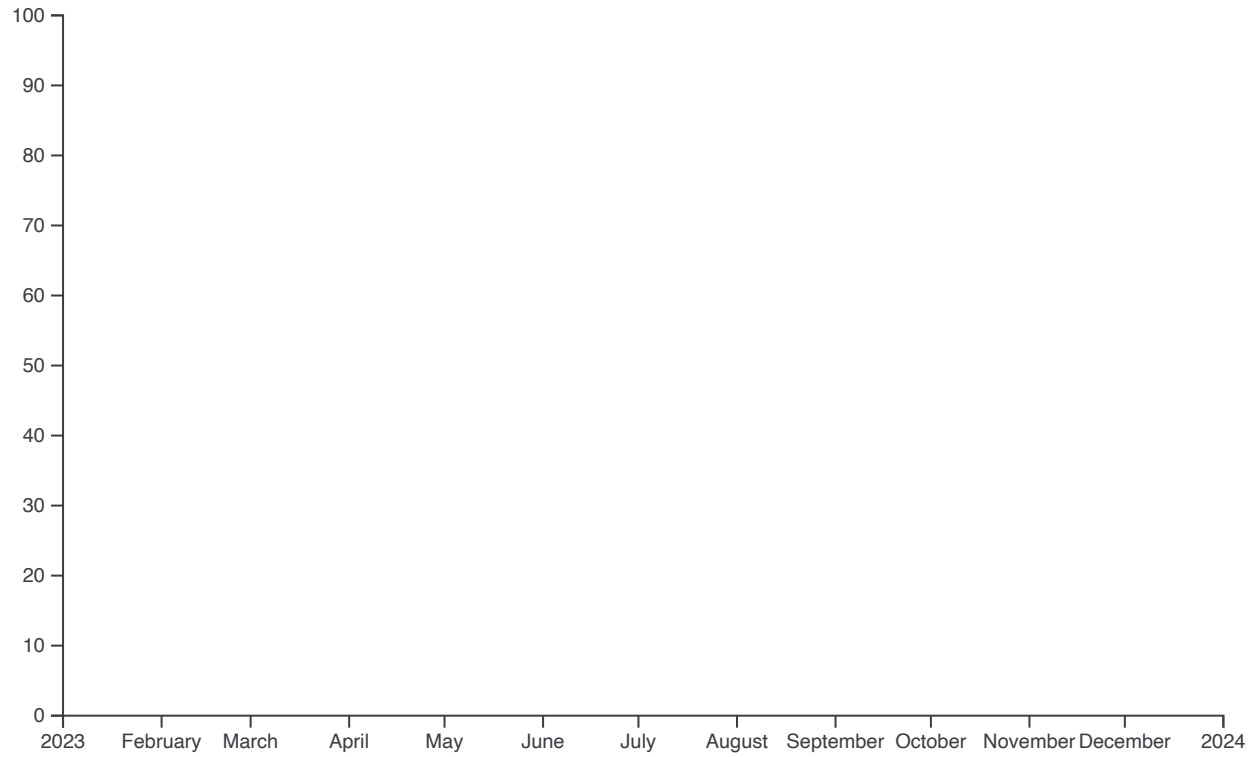
Getting started

D3 works in any JavaScript environment.

Try D3 online

The fastest way to get started (and get help) with D3 is on Observable! D3 is available by default in notebooks as part of Observable's standard library. To create something with D3, return the generated DOM element from a cell. Here is a blank chart to get you started:



js

```
{
  // Declare the chart dimensions and margins.
  const width = 640;
  const height = 400;
  const marginTop = 20;
  const marginRight = 20;
```

```
  const marginBottom = 30;
  const marginLeft = 40;

  // Declare the x (horizontal position) scale.
  const x = d3.scaleUtc()
      .domain([new Date("2023-01-01"), new Date("2024-01-01")])
      .range([marginLeft, width - marginRight]);

  // Declare the y (vertical position) scale.
  const y = d3.scaleLinear()
      .domain([0, 100])
      .range([height - marginBottom, marginTop]);

  // Create the SVG container.
  const svg = d3.create("svg")
      .attr("width", width)
      .attr("height", height);

  // Add the x-axis.
  svg.append("g")
      .attr("transform", `translate(0,${height - marginBottom})`)
      .call(d3.axisBottom(x));

  // Add the y-axis.
  svg.append("g")
      .attr("transform", `translate(${marginLeft},0)`)
      .call(d3.axisLeft(y));

  // Return the SVG element.
  return svg.node();
}
```

As a more complete example, try one of these starter templates:

- Area chart
- Bar chart
- Donut chart
- Histogram
- Line chart

See the D3 gallery for more forkable examples.

Observable includes a few D3 snippets when you click **+** to add a cell (type "d3" when the cell menu is open to filter), as well as convenient sample datasets to try out D3 features. Or upload a CSV or JSON file to start playing with your data. You can also fork any of the hundreds of notebooks we've published for a head start.

Observable is free for public use. Sign up for a Pro account to connect to private databases, collaborate on private notebooks, and more.

D3 in vanilla HTML

In vanilla HTML, you can load D3 from a CDN such as jsDelivr or you can download it locally. We recommend using the CDN-hosted ES module bundle. But for those who need it, we also provide a UMD bundle that exports the `d3` global when loaded as a plain script.

ESM + CDN | UMD + CDN | UMD + local

html

```html
<!DOCTYPE html>
<div id="container"></div>
<script type="module">

import * as d3 from "https://cdn.jsdelivr.net/npm/d3@7/+esm";

// Declare the chart dimensions and margins.
const width = 640;
const height = 400;
const marginTop = 20;
const marginRight = 20;
const marginBottom = 30;
const marginLeft = 40;

// Declare the x (horizontal position) scale.
const x = d3.scaleUtc()
    .domain([new Date("2023-01-01"), new Date("2024-01-01")])
    .range([marginLeft, width - marginRight]);

// Declare the y (vertical position) scale.
const y = d3.scaleLinear()
    .domain([0, 100])
    .range([height - marginBottom, marginTop]);
```

```
// Create the SVG container.
const svg = d3.create("svg")
    .attr("width", width)
    .attr("height", height);

// Add the x-axis.
svg.append("g")
    .attr("transform", `translate(0,${height - marginBottom})`)
    .call(d3.axisBottom(x));

// Add the y-axis.
svg.append("g")
    .attr("transform", `translate(${marginLeft},0)`)
    .call(d3.axisLeft(y));

// Append the SVG element.
container.append(svg.node());

</script>
```

You can also import and destructure individual D3 modules like so:

html

```
<script type="module">

import {forceSimulation, forceCollide, forceX} from "https://cdn.jsdelivr.net

const nodes = [{}, {}];
const simulation = forceSimulation(nodes)
    .force("x", forceX())
    .force("collide", forceCollide(5))
    .on("tick", () => console.log(nodes[0].x));

</script>
```

If you'd prefer to run D3 locally (or offline), you can download the UMD bundles of D3 here:

- d3.v7.js
- d3.v7.min.js

Then, create an `index.html` file as shown above in the **UMD + local** tab. Use the non-minified bundle for debugging, and the minified bundle for faster performance in production.

Installing from npm

If you're developing a web application using Node, you can install D3 via yarn, npm, pnpm, or your preferred package manager.

yarn npm pnpm

bash

```
npm install d3
```

You can then load D3 into your app as:

js

```
import * as d3 from "d3";
```

You can instead import specific symbols if you prefer:

js

```
import {select, selectAll} from "d3";
```

Alternatively you can install and import from D3 submodules:

js

```
import {mean, median} from "d3-array";
```

TypeScript declarations are available via DefinitelyTyped.

D3 in React

Most D3 modules (including d3-scale, d3-array, d3-interpolate, and d3-format) don't interact with the DOM, so there is no difference when using them in React. You can use them in JSX for purely declarative visualization, such as the line plot below.

LinePlot.jsx

```
import * as d3 from "d3";

export default function LinePlot({
  data,
  width = 640,
  height = 400,
  marginTop = 20,
  marginRight = 20,
  marginBottom = 20,
  marginLeft = 20
}) {
  const x = d3.scaleLinear([0, data.length - 1], [marginLeft, width - marginR
  const y = d3.scaleLinear(d3.extent(data), [height - marginBottom, marginTop
  const line = d3.line((d, i) => x(i), y);
  return (
    <svg width={width} height={height}>
      <path fill="none" stroke="currentColor" strokeWidth="1.5" d={line(data)
      <g fill="white" stroke="currentColor" strokeWidth="1.5">
        {data.map((d, i) => (<circle key={i} cx={x(i)} cy={y(d)} r="2.5" />))
      </g>
    </svg>
  );
}
```

Sandbox ↗

D3 modules that operate on selections (including d3-selection, d3-transition, and d3-axis) do manipulate the DOM, which competes with React's virtual DOM. In those cases, you can attach a ref to an element and pass it to D3 in a useEffect hook.

LinePlot.jsx

jsx

```
import * as d3 from "d3";
import {useRef, useEffect} from "react";

export default function LinePlot({
  data,
  width = 640,
  height = 400,
  marginTop = 20,
  marginRight = 20,
```

```
    marginBottom = 30,
    marginLeft = 40
  }) {
    const gx = useRef();
    const gy = useRef();
    const x = d3.scaleLinear([0, data.length - 1], [marginLeft, width - marginR
    const y = d3.scaleLinear(d3.extent(data), [height - marginBottom, marginTop
    const line = d3.line((d, i) => x(i), y);
    useEffect(() => void d3.select(gx.current).call(d3.axisBottom(x)), [gx, x])
    useEffect(() => void d3.select(gy.current).call(d3.axisLeft(y)), [gy, y]);
    return (
      <svg width={width} height={height}>
        <g ref={gx} transform={`translate(0,${height - marginBottom})`} />
        <g ref={gy} transform={`translate(${marginLeft},0)`} />
        <path fill="none" stroke="currentColor" strokeWidth="1.5" d={line(data)
        <g fill="white" stroke="currentColor" strokeWidth="1.5">
          {data.map((d, i) => (<circle key={i} cx={x(i)} cy={y(d)} r="2.5" />))
        </g>
      </svg>
    );
  }
```

Sandbox ↗

For more guidance using D3 in React, see Amelia Wattenberger's post.

D3 in Svelte

As with React, you can use Svelte exclusively for rendering if you like, and only use D3 modules that don't manipulate the DOM. Here is a line plot of an array of numbers that uses d3-shape and d3-scale.

LinePlot.svelte

```svelte
<script>
  import * as d3 from 'd3';

  export let data;
  export let width = 640;
  export let height = 400;
```

```
  export let marginTop = 20;
  export let marginRight = 20;
  export let marginBottom = 20;
  export let marginLeft = 20;

  $: x = d3.scaleLinear([0, data.length - 1], [marginLeft, width - marginRigh
  $: y = d3.scaleLinear(d3.extent(data), [height - marginBottom, marginTop]);
  $: line = d3.line((d, i) => x(i), y);
</script>
<svg width={width} height={height}>
  <path fill="none" stroke="currentColor" stroke-width="1.5" d={line(data)} /
  <g fill="white" stroke="currentColor" stroke-width="1.5">
    {#each data as d, i}
      <circle key={i} cx={x(i)} cy={y(d)} r="2.5" />
    {/each}
  </g>
</svg>
```

Svelte's reactive statements (`$:`) pair nicely with D3 data joins for efficient updates. Below, we use them to render dynamic axes as the data changes.

LinePlot.svelte

svelte

```
<script>
  import * as d3 from 'd3';

  export let data;
  export let width = 640;
  export let height = 400;
  export let marginTop = 20;
  export let marginRight = 20;
  export let marginBottom = 30;
  export let marginLeft = 40;

  let gx;
  let gy;

  $: x = d3.scaleLinear([0, data.length - 1], [marginLeft, width - marginRigh
  $: y = d3.scaleLinear(d3.extent(data), [height - marginBottom, marginTop]);
  $: line = d3.line((d, i) => x(i), y);
```

```
    $: d3.select(gy).call(d3.axisLeft(y));
    $: d3.select(gx).call(d3.axisBottom(x));
  </script>
  <svg width={width} height={height}>
    <g bind:this={gx} transform="translate(0,{height - marginBottom})" />
    <g bind:this={gy} transform="translate({marginLeft},0)" />
    <path fill="none" stroke="currentColor" stroke-width="1.5" d={line(data)} /
    <g fill="white" stroke="currentColor" stroke-width="1.5">
      {#each data as d, i}
        <circle key={i} cx={x(i)} cy={y(d)} r="2.5" />
      {/each}
    </g>
  </svg>
```

Next?

Previous page
What is D3?

Next page
API index