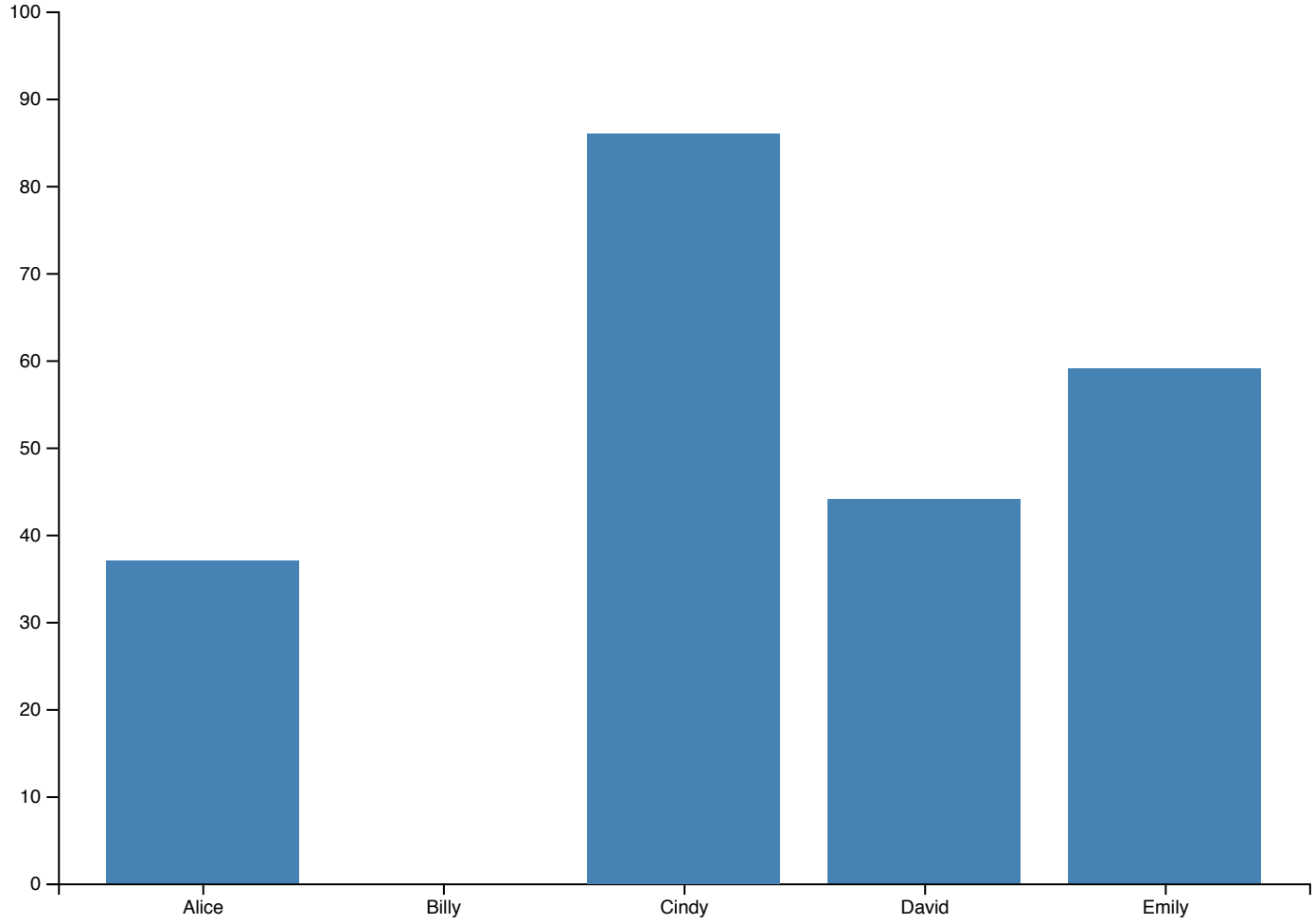
Build your first column chart with D3 v4

Created by Ben Clinkinbeard

Column charts and their horizontal bar chart cousins are staples of every data visualization library ever created, but wouldn't you rather know how to create your own? You can learn how to build a chart like this one in less time than it would take to find, download, and configure some pre-built ball of spaghetti code that may or may not actually meet your needs.



Data visualization in general and D3 in particular can be intimidating if you haven't worked with them before, but they don't have to be! This short document will give you annotated code samples, links to relevant documentation, and access to a working, editable example.

Let's get started!

Data to visualize

Before you can create a data visualization you need data, right? Usually you'll be loading JSON from a server but to minimize moving pieces here we'll just use a plain old JavaScript Array of objects. We have five students and they each have properties for three different subjects.

```
const data = [
  {name: 'Alice', math: 37,   science: 62,   language: 54},
  {name: 'Billy', math: null, science: 34,   language: 85},
  {name: 'Cindy', math: 86,   science: 48,   language: null},
  {name: 'David', math: 44,   science: null, language: 65},
  {name: 'Emily', math: 59,   science: 73,   language: 29}
];
```

Define your dimensions

Next you want to define the size of your chart, including margins that will reserve space for your axes. The `width` and `height` properties take the margins into account so they can be used later and hold the actual usable space for your chart.

```
const margin = { top: 10, right: 10, bottom: 30, left: 30 };
const width = 800 - margin.left - margin.right;
const height = 600 - margin.top - margin.bottom;
```

Setting the stage

With the basics out of the way you can create the root of your chart, an SVG element, using the properties you defined above. The `svg` tag will include the margins in its dimensions since it will be the parent element of the axes too. A base `g` element, or SVG graphics container, is added and moved by the amounts defined in our `left` and `top` margins. This ensures the chart contents are drawn in the proper location and don't interfere with the axes.

```
const svg = d3.select('body')
  .append('svg')
    .attr('width', width + margin.left + margin.right)
    .attr('height', height + margin.top + margin.bottom)
  .append('g')
    .attr('transform', `translate(${margin.left}, ${margin.top})`);
```

Tipping the scales

Now you're ready to define scales, which is how D3 transforms input data like test scores into output values like pixel sizes.

The X scale should be a band scale, which is a special type of scale for bar and column charts. The domain method takes the set of input values, which in this case are the student names. The range method is where you specify the output values that the input domain should be mapped to. For the X scale you want to use the available width of your chart. The padding method will provide a bit of space between the columns on the chart.

```
const xScale = d3.scaleBand()
  .domain(data.map(d => d.name))
  .range([0, width])
  .padding(0.2);
```

Since you want the X axis at the bottom of your chart, create a new `g` element and move it to the bottom using the `height` defined previously. You'll then pass the scale to `d3.axisBottom` since you want the axis labels below the axis itself.

```
svg
  .append('g')
  .attr('transform', `translate(0, ${height})`)
  .call(d3.axisBottom(xScale));
```

The Y scale is a linear scale, which just directly translates input values to output values. In this case the input domain is `[0, 100]` since `0` is the minimum score for a test and `100` is the maximum score. Similar to the X scale, the output range for the Y scale uses the `height` defined previously.

```
const yScale = d3.scaleLinear()
  .domain([0, 100])
  .range([height, 0]);
```

We'll use `d3.axisLeft` to attach the axis this time, since we want the labels to the left of the axis.

```
svg
  .append('g')
  .call(d3.axisLeft(yScale));
```

Ready, set, draw

Now you're ready to render the columns for your chart! We'll wrap this code in a function so it can be called multiple times and accept the subject we want to render.

```
function render (subject = 'math') {
  const update = svg.selectAll('rect')
    .data(data.filter(d => d[subject]), d => d.name);

  update.exit()
    .remove();

  update
    .attr('y', d => yScale(d[subject]))
    .attr('height', d => height - yScale(d[subject]));

  update
    .enter()
    .append('rect')
    .attr('x', d => xScale(d.name))
    .attr('width', d => xScale.bandwidth())
    .attr('y', d => yScale(d[subject]))
    .attr('height', d => height - yScale(d[subject]));
}

render();
```

The first thing this function does is select all the `rect` elements in our SVG element and join it to the data we want to render. We'll filter the base data so we only get people who have a score for the specified subject, and tell D3 to match any existing `rect`s to the data using the name property.

Any rectangles that were drawn previously but don't have data for the current subject will be part of the exit selection, and will be removed.

Existing rectangles we want to keep will just have their `y` and `height` attributes updated.

Data items that don't have a rectangle will be in the enter selection. For those items we'll create a new `rect` and append it to the DOM, then set all the attributes.

Lastly, we call the render function to trigger the initial chart creation.

With all the important bits encapsulated in your `render(subject)` function you can call it from anywhere and your chart will immediately update! Want to update your chart in response to a button click? Toss `<button onclick="render('science')">Science</button>` in your HTML and you're good to go!

What's next? I'm glad you asked!

Try it out for yourself!

Open this JSBin, change some stuff around and see what happens!

Want to learn D3 thoroughly, from the ground up?

Check out my course **Build Interactive JavaScript Charts with D3 v4**.
25 bite-sized video tutorials with sample code! Over 2.5 hours of lessons!

Need custom development or personalized training?

I have 15 years experience and I'm available for hire. **Get in touch!**