Visualisation and Sensing, week 5 Workshop: D3 Lines, Scales, and Axes.

In this workshop, we'll look at adding **axes** to our graphs, using time series and scales, and building a line graph.

You should open this individual folder inside VS Code, and run Live Server from inside it. (You'll have errors if you try to use something further "up" the tree).

> ### Javascript reminder: template strings!
>
> Instead of making strings containing dynamic data with concatenation (eg: `"total count: " + count + " flights"`), you can use **template strings**. These are wrapped in *backticks* (`) rather than quote marks. Then, any variable wrapped with `${` and `}` will be interoplated into the string. For instance:
>
> ```
> `total count: ${count} flights`;
> ```
>
> Your editor will probably supply helpful syntax highlighting when you do this.
>
> This is particularly tidy when interpolating more than one variable. The code for this workshop uses them in `transform` attributes.

1. Making a y axis

> Work from the starting point in `d3-1.html`

Let's make a y axis for this bar chart.

Here's the D3 docs for an axis: https://d3js.org/d3-axis.

Read the first two paragraphs of the above docs, and then, use them to make a **y axis** based on `yScale`

- you'll need to make an `axisLeft` using `yScale`
- you'll need to translate it by (margin,margin)
- note how the documentation suggests you use `call`; this is a method on a D3

Selection. If you're interested, read the docs for it here to see what it does.

When you've done that:

- append the method `nice()` to your y **scale**. What changes?

2. Making an x axis; Making a time scale

> Work from the starting point in `d3-2.html`

Making a horizontal x axis

We're first going to add an x axis to the chart.

That means we're going to need an x **scale**, too. For now, we'll use a scale of "day in the dataset", ie, the index of the `flightData` array.

First, make a `scaleLinear` object for the x axis. Your scale should have a domain from 0 to the `monthLength` (a variable I have provided for you). It should have a range from 0 to the width of all the bars in the chart (*including* their spacing...).

You can use this new X scale to set the x position of your bar `<rect>` s.

You can then use this scale to make an x axis - an `axisBottom` - just as you made a y axis.

When you've made the scale and axis, use the translate attribute of the axis group to position it on the screen.

Try deleting some of the flight data in the middle. What changes about the x axis? Why is this disappointing?

Making a time scale

D3 has scales other than `scaleLinear` , and this is where some of its power lies. `scaleTime` has a domain defined with `Date` objects, which allows us to pass `Date` objects into the function, and get a pixel coordinate out.

Replace the `scaleLinear` call with `scaleTime` . Your output `range` will not change. But your input needs to: it should now be a range of Javascript Date objects.

You can make a Date like so: `new Date(year,month,day)` . You'll need to use this to both calculate the `domain` of your data ... and also within your calculations of the rect's x position (which previously just used its index).

> Hint: you might find making a function called `dateForDayData` useful: you can pass a single day-data object into it, and get a Date object out.
>
> Hint: have a look at your x axis when it renders. Is there anything about how Javascript handles dates - and particularly *months* you need to pay attention to?

Now try deleting some of the items in the middle of `flightData`. What happens to the bars and axes? Is this better?

Finally, try altering the `ticks` of the `axisBottom`. Note how the dates change.

3. Making a line graph

Let's make a line graph of this data.

> Work from the starting point in `d3-3.html`

In D3, we can use the SVG element to draw lines

The element uses the `d` attribute to describe what should be drawn.

https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#path_commands

Read the documentation for the d attribute above. You'll see that the syntax is powerful, and... complex.

Fortunately, D3 gives us the `d3.line` function, which makes generating path data for a `d` attribute straightforward.

Start by replacing the `<rect>` elements with a single path.

- First, rename `rectGroup` to `lineGroup`, let's keep things tidy.
- To make our path, replace the `rect` selection with a `path` selection...
- ...and create a `path` as part of the join.
- Delete all the attributes relevant to rects (width, height, fill).
- Add a stroke of eg `#333`.

Before you go any further, have a look at the generated HTML of the page in your browser. Where is the `path` element? What has happened?

D3 data binding makes an element for *each item in the binding*. We don't want that; we want *one* path for all the data. Instead of binding to `flightData`, bind to `[flightData]`: *one* item (an array, containing an array), will lead to *one* `path`.

Now let's populate the path.

Add an attribute `d`. Set its value to `d3.line()`. You now need to chain an `x` and `y` method, to define how `x` and `y` should be calculated. Something like this:

```
.attr("d", d3.line()
  .x((d, i) => SOMETHING)
  .y((d, i) => SOMETHING)
)
```

Pay close attention to the parentheses: x and y are *methods* called on `d3.line`, which is the value passed to `d` *inside* `attr`.

(You could, of course, define your line generator outside this function, and pass it in as a variable)

Finally: you might want to set fill to `none` on your line.

Again, try deleting some inner data from `flightData`. What happens?

If you reach this point and have time left, why not add some blobs/points to make highlight data points on your line. You could do so by creating `circle` elements bound to `flightData`.