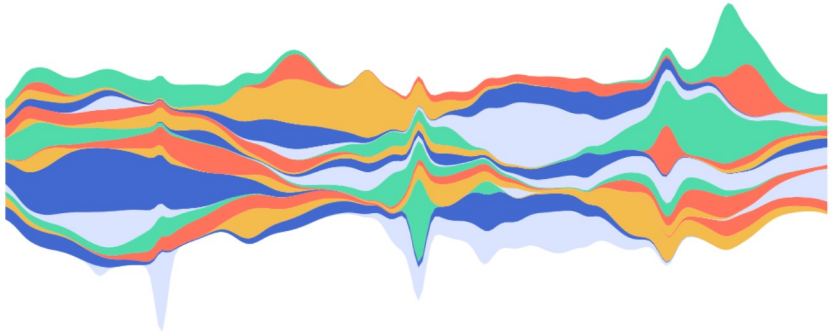
D3 by Observable

Introduction to D3

Session 3

Observable notebook

YouTube video



These slides are provided with the course "Introduction to D3" by Paul Buffa – Head of Product Education at Observable

Agenda

1. SVG Path
2. D3 Path generators
3. Bar charts
4. Stack layouts

SVG Paths

SVG Paths

SVG Paths are one of the more difficult SVG elements to master.

SVG Paths

SVG Paths are one of the more difficult SVG elements to master.

- They can be used to generate any custom shape

SVG Paths

SVG Paths are one of the more difficult SVG elements to master.

- They can be used to generate any custom shape
- They have their own mini-language

SVG Paths

All of these shapes are a single SVG Path.

Source: [Hello, Flubber]

SVG Paths

Paths start with the M command, which is short for “MoveTo”

```
<svg width=300 height=200>
  <path
    d='M'
    style="stroke: #000">
  </path>
</svg>
```

SVG Paths

Then we give an X, Y coordinate. Given it's a single point, there is still nothing we see.

```
<svg width=300 height=200>
  <path
    d='M 0, 100'
    style="stroke: #000">
  </path>
</svg>
```

SVG Paths

The L command draws a straight line to our next X, Y coordinate, and is short for “LineTo”


```
<svg width=300 height=200>
  <path
    d='M 0, 100 L 100, 100'
    style="stroke: #000">
  </path>
</svg>
```

Activity 1

D3 Path Generators

D3 Path Generators

It is difficult to write the Path mini-language by hand, and would be very tedious to write this for a dataset. D3 has multiple functions for making path generation much easier.

d3.line()

D3.line() is a path generator for making the line for something like a line chart.

```
const line = d3.line()
    .x(d => x(d.xValue))
    .y(d => y(d.yValue));
```

d3.line()

You can supply optional parameters for curve factories for making smoothed lines or stepwise lines.

```
const line = d3.line()
    .curve(d3.curveBasis)
    .x(d => x(d.xValue))
    .y(d => y(d.yValue));
```

d3.line()

When using a path generator, it's important to remember that this is for a single path.

```
g.append("path")
  .datum(myData)
  .attr("d", d => line(d))
  .style("stroke", "#000")
  .style("fill", "none");
```

d3.line()

You can also opt to note use a function for calling the line, as it will by default use the data bound.

```
g.append("path")
  .datum(myData)
  .attr("d", line)
  .style("stroke", "#000")
  .style("fill", "none");
```

Activity 2 & 3

Bar Charts

Bar Charts

While bar charts are considered a simple chart form, they require a little more consideration when implementing with D3, due to the inverted coordinate plane.

Activity 4 & 5

Stack Layouts

Stack Layouts

Given the complexity making a bar chart, to make a stacked bar chart using this method would be very difficult. Fortunately there is d3.stack() for the win!

Stack Layouts

We need to tell the d3.stack() function what we're stacking. These are called the keys.

```
stackLayout = f()
```

```
stackLayout = d3.stack()
  .keys()
```

Stack Layouts

When we enter a dataset into our stack function, we are returned an array of arrays, i.e. hierarchical data.

```
stackLayout = ▼Array(9) [
  0: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
  1: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
  2: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
  3: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
  4: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
  5: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
  6: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
  7: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
  8: ▶Array(52) [Array(2), Array(2), Array(2), Array(2), Array(2),
]
```

Stack Layouts

Given we're stacking values, we'll need to figure out how to find the max y-scale value correctly.

```
d3.scaleLinear()
  .domain([0, d3.max(stackLayout, d => d3.max(d, d => d[1]))])
  .range([100, 0])
```

Stack Layouts

And since our stacked data is hierarchical, that means we'll need our elements to be hierarchical.

```
const bars = g.selectAll(".group")
  .data(stackLayout)
  .join("g")
  .attr("class", "group")
  .selectAll("rect")
  .data(d => d)
  .join("rect")
```

Activity 6

Session 3 is a wrap!

Observable