D3.js

Data Driven Documents

DOM Manipulation Library

- Provides methods for easily manipulating the Document Object Model (web page elements)
- SVGs are the most common scenario for D3.js
- Create graphs, charts, maps, etc.
- Update them based upon user input - interactive is important
- This project will plot number of births per month per year

Let's Get Started

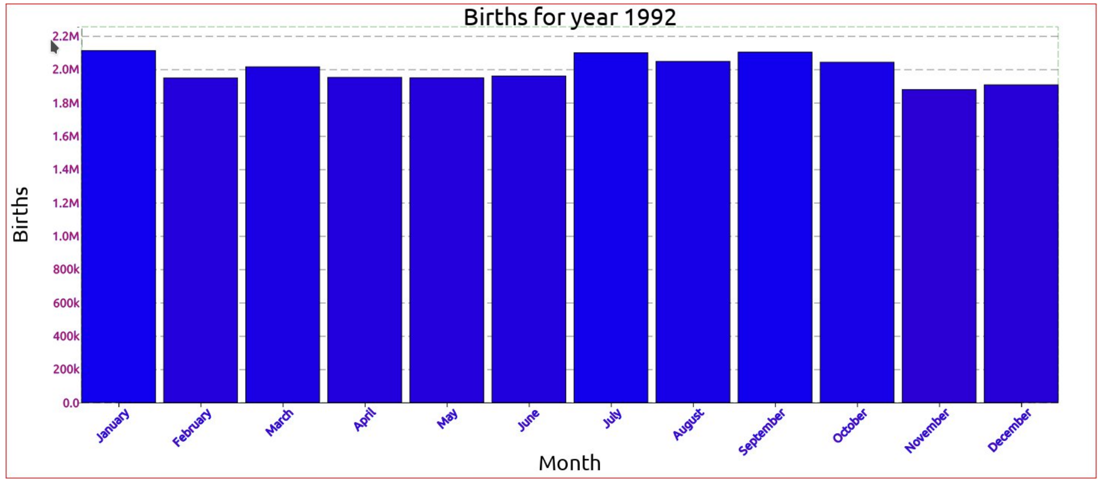
Turning JSON into visual output

- Our data source is a simple JSON file - an array of month objects with 3 key / value pairs each:

```
[
        {
        “year”: 1967,
        “month”: “January”,
        “births”: 31502
        },
        {
        “year”: 1967,
        “month”: “February”,
        “births”: 26703
        },
...
```

-]

- Final output:



Page 1

http://bclug.ca:8008/d3/kwlug/bar-chart/page1.html

Create an SVG

There is a `<div id=”graph”>` in HTML for our SVG, and we create one like this:

```
d3.select(“#graph”)
     .append(“svg”)
          .attr(“width”, ...)
          .attr(“height”, ...)
          .attr(“x”, ...)
          .attr(“y”, ...)
          .style(“outline”, “solid red 1px”)
```

That’s all that is required to create an SVG.

Page 2

http://bclug.ca:8008/d3/kwlug/bar-chart/page2.html

Data Source

- Our data source is a simple JSON file - an array of month objects with 3 key / value pairs each:

```
[
        {
        “year”: 1967,
        “month”: “January”,
        “births”: 31502
        },
        {
        “year”: 1967,
        “month”: “February”,
        “births”: 26703
        },
]
```

Fetch Data

- To fetch some data (several formats supported), queue request(s) for asynchronous retrieval into new variable birthDataJSON:

```
URL = “http://bclug.ca:8008/d3/kwlug/bar-chart/birthData-
JSON.js”;


d3.queue()
      .defer(d3.json, URL)
      .await(function(error, birthDataJSON) {
            if (error) throw error;
            birthData = birthDataJSON; // assign to global var
...
```

Update DOM with data

```
...
// Add data to our input selector:
d3.select(“#inputYear”)
      .property(“min”, d3.min(birthData, d => (d.year))
      .property(“max”, d3.max(birthData, d => (d.year))
      .property(“value”, minYear)
      ;
// Update input selector’s label
d3.select(“label”)
      .text(`${minYear} <— Year Range —> ${maxYear}`)
      ;
})
```

- Updating DOM elements is easy with D3.js

Page 3

http://bclug.ca:8008/d3/kwlug/bar-chart/page3.html

Creating Scales

- Data visualization requires scales
- A scale's **domain** is the scope of input data to be plotted
- A scale's **range** is the output location in the SVG to plot the data
- Many scales to choose from, we'll use `scaleLinear`, with continuous (not discrete) input and continuous output:

```
xScale = d3.scaleLinear()
       // domain is months in a year:
       .domain([ 1,12 ])

       // Spread bars across width of SVG starting at padding offset:
       .range([ padding.left, padding.left + width ])
```

Axes

- Axes keep charts honest
- **Axes take scales as parameter**s
- 4 types of axes; axisLeft has “ticks” & labels to **left** of the line
- Axes comprise a “path” (line) with ticks (also paths) and tick labels

Create and append an axis

```
yAxis = d3.axisLeft(yScale);

d3.select(“svg”)
// Add a “group” to hold the axis elements:
     .append(“g”)
          // Assign an ID
          .attr(“id”, “yaxis”);
```

Axis formatting

- There are many options to format an axis
- The "ticks" are the little marks indicating precise location
- Tick marks can have sizes:
  - `tickSizeOuter` for ends of axis line beyond labels
  - `tickSizeInner` for normal positional indicators
- This bar chart will have Y-axis tick marks that stretch the width of the chart and the lines will be dashes, not solid
- Also, CSS can be applied to most SVG elements

Axis formatting: X Axis

For our X axis, our data is in the form of month numbers 1 to 12 as set by:

```
xScale

       .domain( [1,12] )
```

And, it should display the months' names, which can be looked up in our months array, converting array element number (not index) to name:

```
d3.select("#xAxis")

       .call(xAxis)

       // Tick labels: look up month names by number:
       .text( (data,idx) => (months[idx].name ))

       // Rotate month names slightly to avoid collisions:

       .attr("transform", "rotate(-45)")
```

Axis formatting: Y Axis

```
yAxis

      // Make ticks width of SVG in opposite direction of labels
      .tickSizeInner( -1 * width)

      .tickSizeOuter(0);

d3.select(“#yaxis”)

      // Apply our axis and formatted labels:

      .call(yAxis)

      .selectAll(“text”)

            // Format each data (d) as 2.0M vs 2000000:

            .text( d => (d3.format(“.2s”)(d) ))
```

Page 4

http://bclug.ca:8008/d3/kwlug/bar-chart/page4.html

Binding data to graph elements

- D3.js will join an array of data to a selection of DOM elements
- Bar charts are made with rectangles, called “rect” in SVG-speak

- Here we select all existing bars and join the JSON data filtered to the year chosen by the user (i.e. 1992):

```
let barData = birthData.filter(d => d.year === 1992);
let bars = d3.selectAll(“rect”)
    .data(barData)
    ;
```

Binding data: General Update Pattern

- When there are more data elements than DOM elements, new DOM elements will be added by the `.enter()` selection

- Initially, we’ll have 12 elements of birth data and zero bars, so we’ll use `.enter()` to append some bars (rectangles)

- D3.js uses a “General Update Pattern” comprising **enter**, **update**, and **exit** (add, update, delete) selection actions.

Appending new bars: `.enter()`

- First, we’ll use .enter() to add DOM elements.

```
bars
     .enter()
     // Append 12 rectangles for the 12 months
     .append(“rect”)
     ...
```

Defining bars with attributes

- Use the `.attr()` function to give bars some attributes:

```
bars.enter().append(“rect”)
       .attr("width", barWidth)
       .attr("x", (data, index) => (xScale(
              // Convert month name in JSON data to number:
              months.find( m => (m.name === data.month)).num)
              // Adjust the x pos left by ½ barWidth to centre-align at ticks:
              - barWidth / 2))
       // Starts mid-way, ends at bottom (at x axis):
       .attr("y", (d) => (yScale(d.births) + padding.top))
       .attr("height", (d) => (height - yScale(d.births)))
```

Page 5

http://bclug.ca:8008/d3/kwlug/bar-chart/page5.html

Updating data

- The app gets the year from the input selector, which has a “change” listener which invokes updateGraph, which reads value of that input slider.
- Default selection action is update (if no .enter() or .exit() specified)

```
// Now selectAll returns array of 12 items:
bars = d3.selectAll(“rect”);
```

Updating data: shortcut

- Save time with `.merge()` - it joins `.enter()` and `update`
- Only **ONE** line of code has been added to page 5’s JS code:

```
.merge(bars)
```

- And we have updating bars in our chart because `.enter()` and update share all the code for applying attributes to the rectangles
- However, there’s a **bug**: year 2015 has only May’s data; January’s bar moves to May’s position. Tool tip changes mid-bar.

Page 6

http://bclug.ca:8008/d3/kwlug/bar-chart/page6.html

Data “constancy”: key functions

- Default data binding order is first-come first-serve
- Binding data to specific DOM elements is possible
- Just add a “key function” to the .data(), which returns an array of unique values which D3.js will bind to specific items:

```
.data(barData, function(d) {
    return d.month;
    })
```

- May 2015 now has correct data in correct location

Page 7

http://bclug.ca:8008/d3/kwlug/bar-chart/page7.html

Removing data: `.exit()`

- Where there are more DOM elements than data array elements, items need to be removed from DOM
- We need to remove elements that no longer have data bound to them
- `.exit().remove()` does that:

```
bars
   .exit()
      .remove()
```

Page 8

http://bclug.ca:8008/d3/kwlug/bar-chart/page8.html

Revisiting Scales: adding colour

- Our black bars need enhancement; it’s easy to add colour based on our data - the month number:

```
let colourScaleX = d3.scaleLinear();
colourScaleX
      .domain([ 1,12 ])
      // D3 will interpolate between colours!
      .range([ “red”, “blue” ])
```

Then, in our .merge() where we enter & update our bars, apply colour to fill attribute:

```
      .attr(“fill”, (d,index) => (colourScaleX(index+1) ))
```

Page 9

http://bclug.ca:8008/d3/kwlug/bar-chart/page9.html

Transitions

- To make our bars transition between states, there's a `.transition()` function
- Transitions have durations, delays, and easing functions
- Many choices of "easing" between states, we'll use the default `.ease(d3.easeCubicInOut)`
- Delay function staggers the transitions
- Transition the `.exit().remove()` to a width of zero:

```
    .transition().duration(1000).delay(...)
    .attr("width", 0)
```

Page 10

http://bclug.ca:8008/d3/kwlug/bar-chart/page10.html

More transitions

- Our bars are now nicely transitioning between states
- Notice the delay function to stagger transitions: take the data object and its index in the array, and return `index*50` ms:

```
.transition()
.duration(1000)
.delay( (data,index) => (index * 50))
```

- Transition the axes, labels, title,...
- Changed the fill to colourScaleY: colour based on Y value