Writing Reusable Visualization Software

Michael Freeman
@mf_viz
Part II

Session 2 objectives

Review reusability pattern for D3 charts

Integrate D3 into a more robust **JavaScript framework** (React)

Leveraging reusable code to make **small multiples**

Discuss challenges of implementing reusable approaches

Reusability Pattern

Basic reusable D3 approach

```
function chart() {
  var width = 720, // default width
      height = 80; // default height

  function myChart() {
    // generate chart here, using `width` and `height`
  }

  myChart.width = function(value) {
    if (!arguments.length) return width;
    width = value;
    return myChart;
  };

  myChart.height = function(value) {
   // same pattern as for width
  };
  return myChart;
};
```

Using your function

```
// Construct a new instance of the chart function
var myChart = chart() // Function that returns a function
                .width(500) // set parameters
                .height(500); // set parameters

// Bind your dataset (datum) to a div element and call the chart function
var chartWrapper = d3.select('#my-div')
                .datum(dataSet) // bind dataset, datum is used b/c only 1 chart element is created
                .call(myChart); // call the chart function!
```

React

React

JavaScript library for building user interfaces

Build **reusable components** to use throughout your application (`<Todo />`)

Use a **one-directional data-flow** to pass information (**properties**) into components

Easily keep track of the **state** of your application

Lifecycle methods trigger events at different points (**mounts**, when **props change**)

Commonly written in **JSX / ES6**

Build sophisticated web applications

```
// Input Component
var InputComponent = React.createClass({
    getInitialState:function() {
        return {text:''};
    },
    update:function(event) {
        var value = event.target.value;
        this.setState({text:value});
    },
    render:function() {
        return (
            <div>
                <input onChange={this.update} />
                <br/>
                <text>The user has typed: {this.state.text}</text>
            </div>
        );
    }
});
```

Demo-4

D3 + React

Which software
should be responsible
for what processes?

D3 + React

Manage data, keep track of settings (i.e., what data to visualize): **React**

Compute visual layouts: **D3**

Bind data to DOM: **D3**

Highly contested, may depend on type of app, team, data size, etc.

D3 + React Example

Demo 5

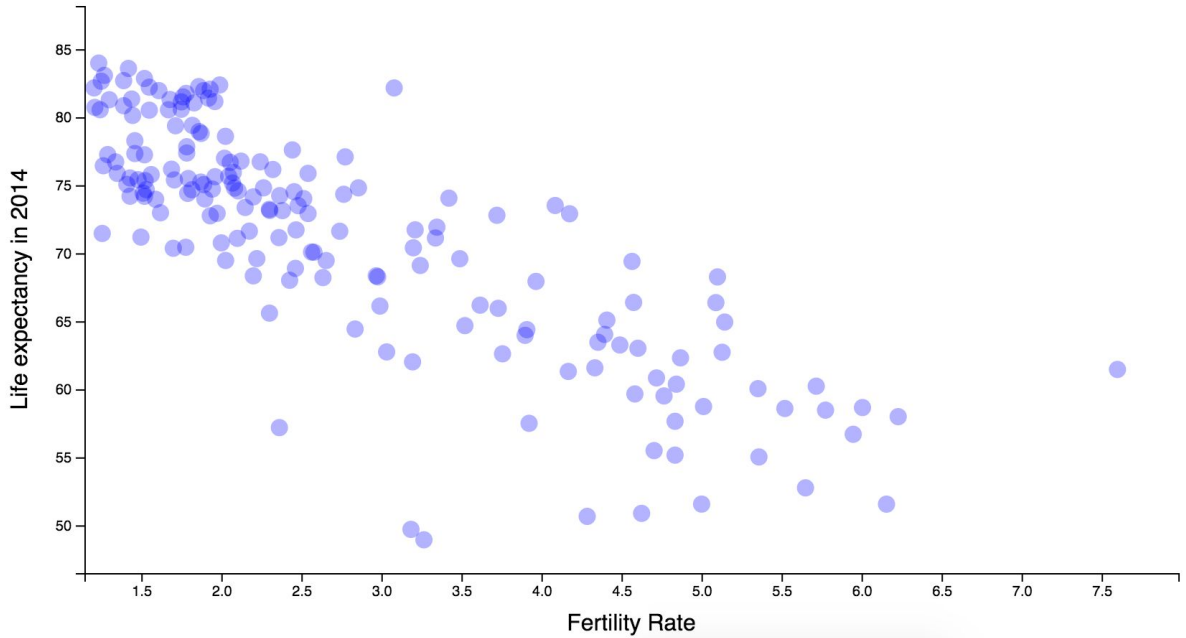
X Axis Variable

Fertility

Y Axis Variable

Life Expectancy

Find a country



Demo-5

JavaScript Files

App.js

- load data (using d3.csv)
- render a <ScatterPlotComponent />

ScatterPlotComponent.js

- Create a DOM node in which to create a D3 chart
- Call the ScatterPlot function when data/parameters update

ScatterPlot.js

- Same scatterplot function written in a reusable D3 approach
- Creates / updates chart on the DOM

```
var App = React.createClass({
    getInitialState() {
        return {
            data:[],
            xVar:'gdp',
            yVar:'life_expectancy',
            idVar:'country',
            search:''
        }
    },
    componentWillMount() {
        // Get data
        d3.csv('data/prepped_data.csv', function(data){
            this.setState({data:data})
        }.bind(this))
    },
```

React loads, manages data: **App.js**

```
render() {
        // Prep data
        let chartData = this.state.data.map((d) => {
            return {
                x:d[this.state.xVar],
                y:d[this.state.yVar],
                id:d[this.state.idVar]
            }
        });
      return(
      <ScatterPlotComponent
            titles={titles}
            search={this.state.search}
            data={chartData}
            width={window.innerWidth/2}
            height={window.innerHeight * .9} />

      )
```

React transforms data into a consistent format: **App.js**

```
// Scatterplot component
var ScatterPlotComponent = React.createClass({
    componentDidMount(){
        // Define scatterplot function
        this.scatter = ScatterPlot();
        this.update();
    },
```

On mount, create an initial scatter function, then update: **ScatterPlotComponent.js**

```
// render
      render() {
            // Return links and show anything inside the <App> component (children)
            return (

             <div width={this.props.width}
                  height={this.props.height}
                  ref={(node) => { this.root = node;}} />
            );
      }
```

On render, expose DOM node for d3 to use: **ScatterPlotComponent.js**

```
// Update on new props
    componentWillReceiveProps (props){
        this.props = props;
        this.update();
    },
```

Update on receiveProps: **ScatterPlotComponent.js**

```
// Create chart
    update() {
        // Update parameters
        this.scatter
            .width(window.innerWidth * .9)
            .height(window.innerHeight - 130)
            .fill('blue')
            .xTitle(this.props.xTitle)
            .yTitle(this.props.yTitle)
            .radius((d) => d.selected == true ? 6 : 1);

        // Call d3 update
        d3.select(this.root)
            .datum(this.props.data)
            .call(this.scatter);
    },
```

Update function manipulating the DOM: **ScatterPlotComponent.js**

Demo 5

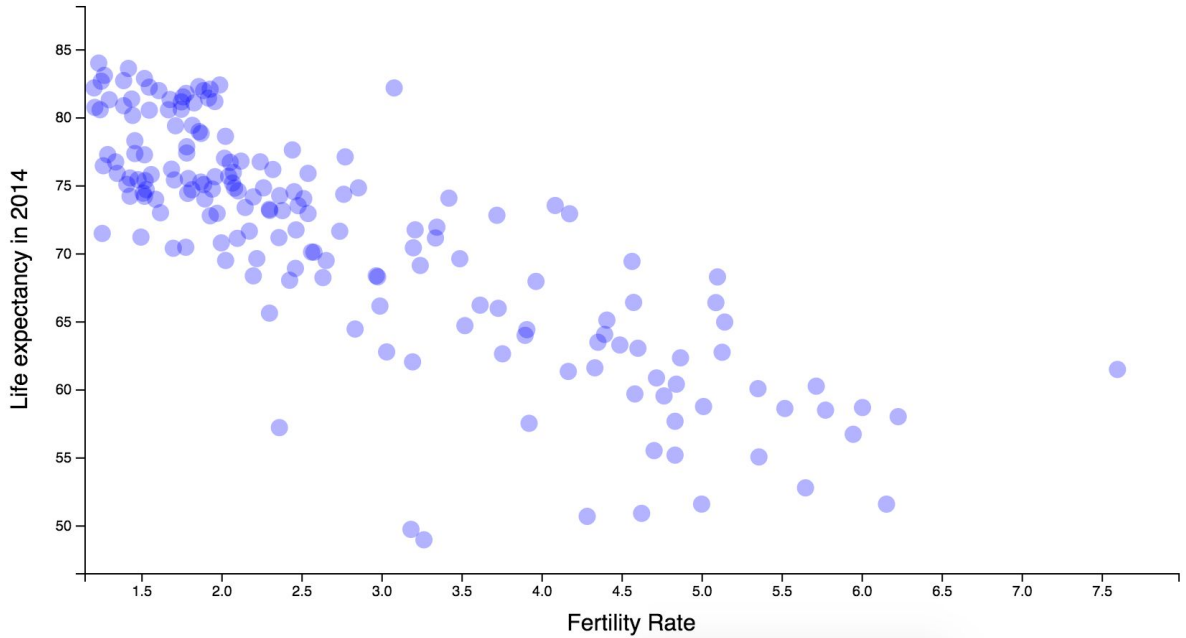
X Axis Variable

Fertility

Y Axis Variable

Life Expectancy

Find a country



Demo-5

Small Multiples

Create + manage multiple instances of the same chart type by using the **data-join.**

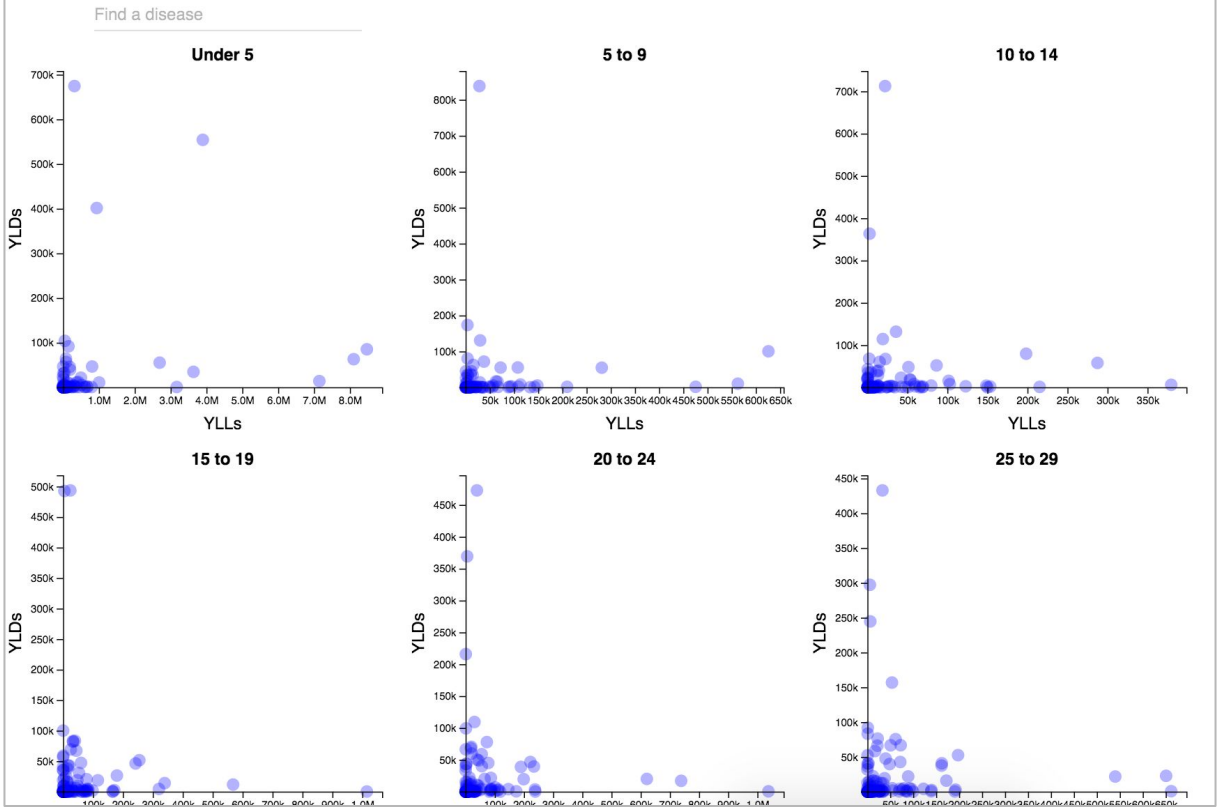
```
// Use d3 data-join at the chart level
var charts = d3.select(this.root)
                .selectAll('.chart')
                .data(this.props.data)// Array of datasets

charts.enter()
       .append('div')
       .attr('class', 'chart')
       .merge(charts)// entering AND updating elements
       .call(this.scatter);

charts.exit().remove()
```

Small multiples

Disease Burden in India



Demo-6

Challenges

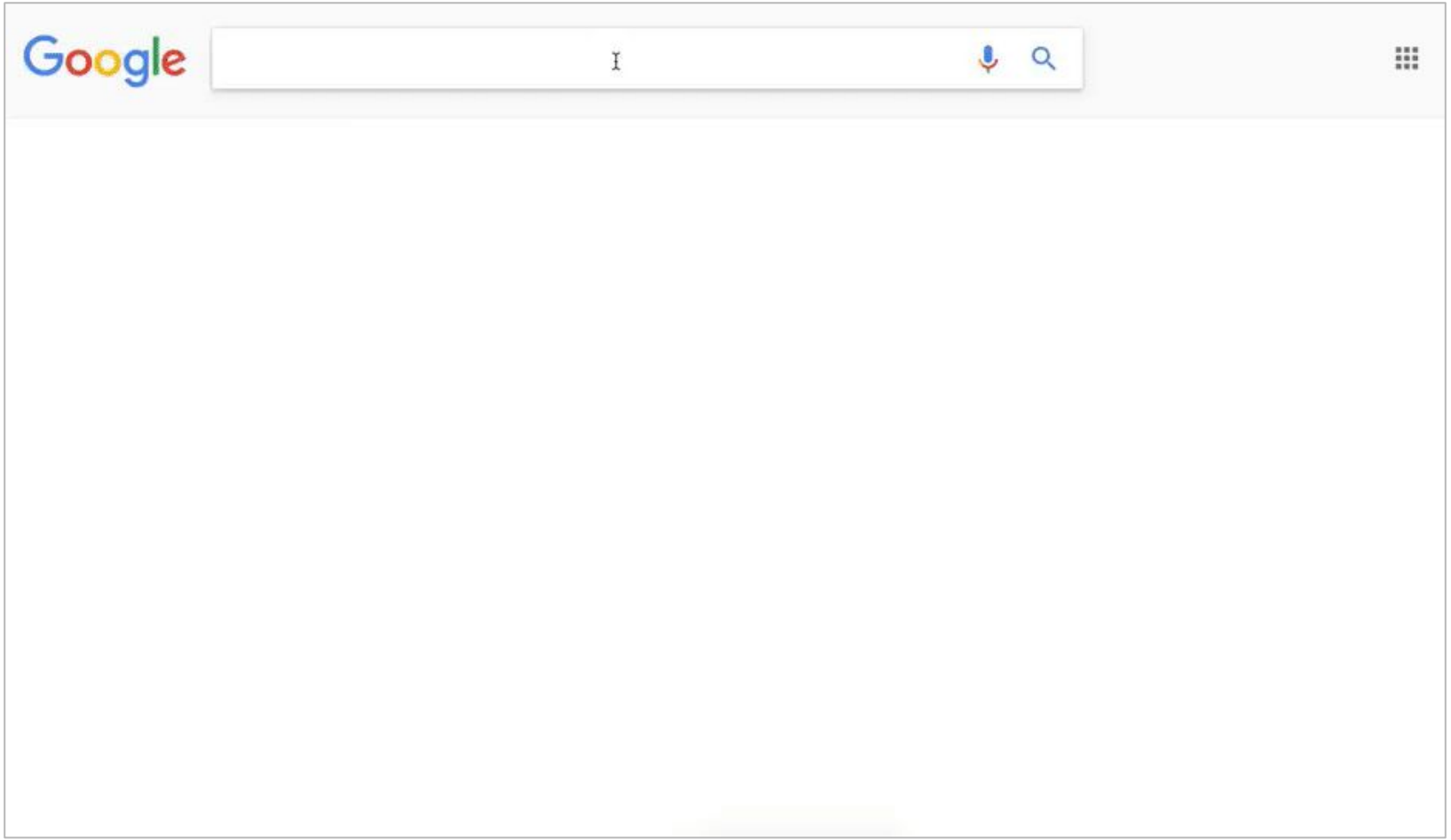
Why doesn't everyone do this?!?!?

Reusability Challenges

Time (or money, or both)

Building an API is a complex task

For smaller projects, it may be ***harder to write*** and ***harder to read***.



But it beats doing this year after year....

Thanks!

Michael Freeman
@mf_viz