Writing Reusable Visualization Software

Michael Freeman
@mf_viz
Part I

But first, *why*...

Facebook's fake news problem, explained

Updated by Timothy B. Lee | tim@vox.com | Nov 16, 2016, 9:10am EST

TWEET SHARE

Photo by Justin Sullivan/Getty Images

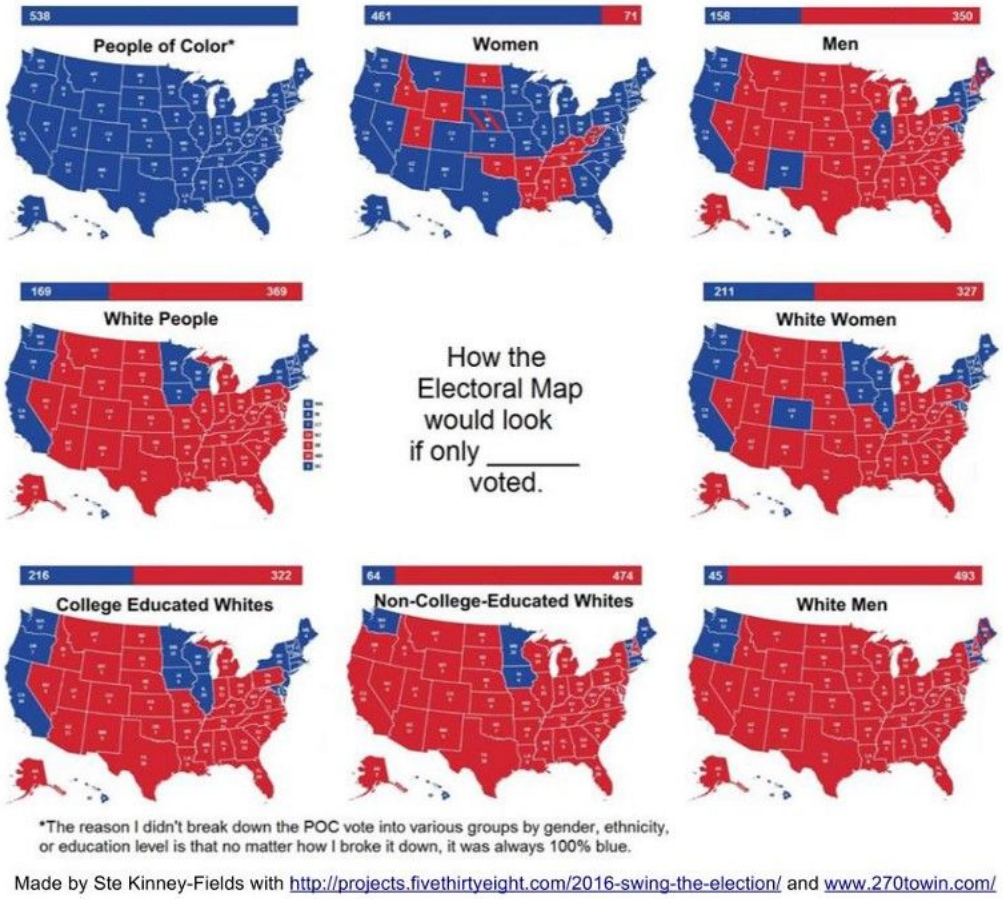
News stories are supposed to help ordinary voters understand the world around them. But in the 2016 election, news stories online too often had the opposite effect. Stories rocketed around the internet that were misleading, sloppily reported, or in some cases totally made up.

Most Read

Everything mattered: lessons from 2016's bizarre presidential election

Gilmore Girls' final words change everything we believe about Rory and Stars Hollow

The tools we build are consequential



We need visualization to understand and explain the world around us (source)

Talk materials

(mfviz.com/strata-2016)

Session 1 Objectives

Operationalize the concept of ***reusable visualization software***

Investigate ***JavaScript particularities*** that we can leverage

See the ***D3.js implementation*** of reusability

Use D3 to manage ***static and dynamic*** elements on the DOM

Build a ***simple reusable example*** using d3.js

Session 2 objectives

Review reusability pattern for D3 charts

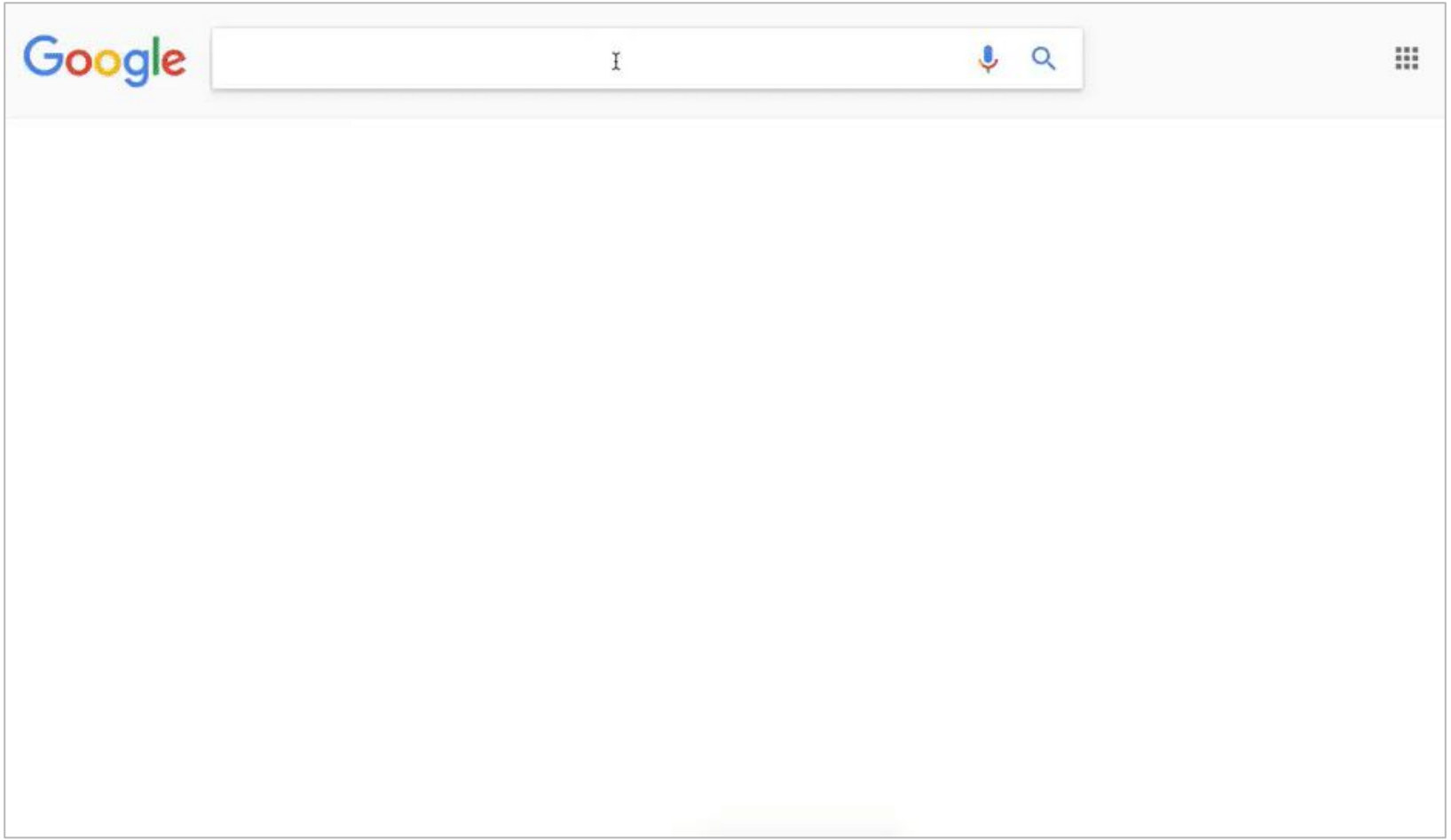
Integrate D3 into a more robust **JavaScript framework** (React)

Leveraging reusable code to make **small multiples**

Discuss challenges of implementing reusable approaches

Reusable Visualization Software

This is how I use D3 after 4 years of experience...



How everyone uses d3.js...

How should I use D3 after 4 years of experience...?

Leverage an API with the same patterns as D3

```
// Construct a new instance of the chart function
var myChart = BarChart() // 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!
```

20 best JavaScript charting libraries

by VAIBHAV SINGHAL — Jun 12, 2015 in DESIGN & DEV

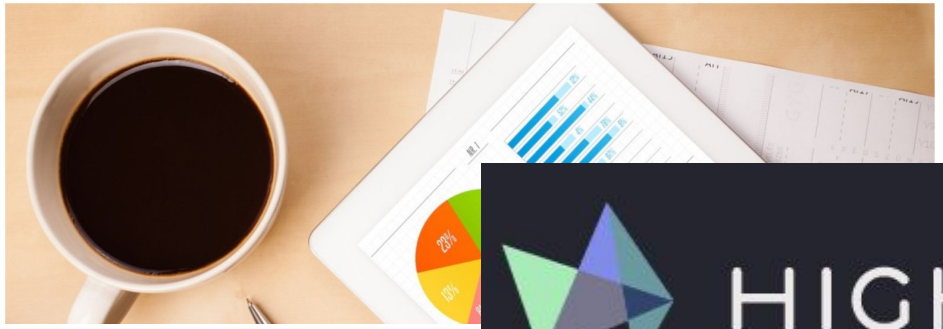


Chart.js

Doesn't this already exist...?

What's missing?

Complex transitions

Advanced chart types

Custom click events

All reasons you started using D3 in the first place...

If you take the time to build a chart in D3.js, **you should be able to reuse it**.

What does reusable software look like?

Easy to **write**, easy to **read**

Easily used for **multiple datasets**

Visual encodings are **configurable**

Programming patterns are **consistent**

Writing Reusable Software Goals

Write code for ***charts***, not ***data***

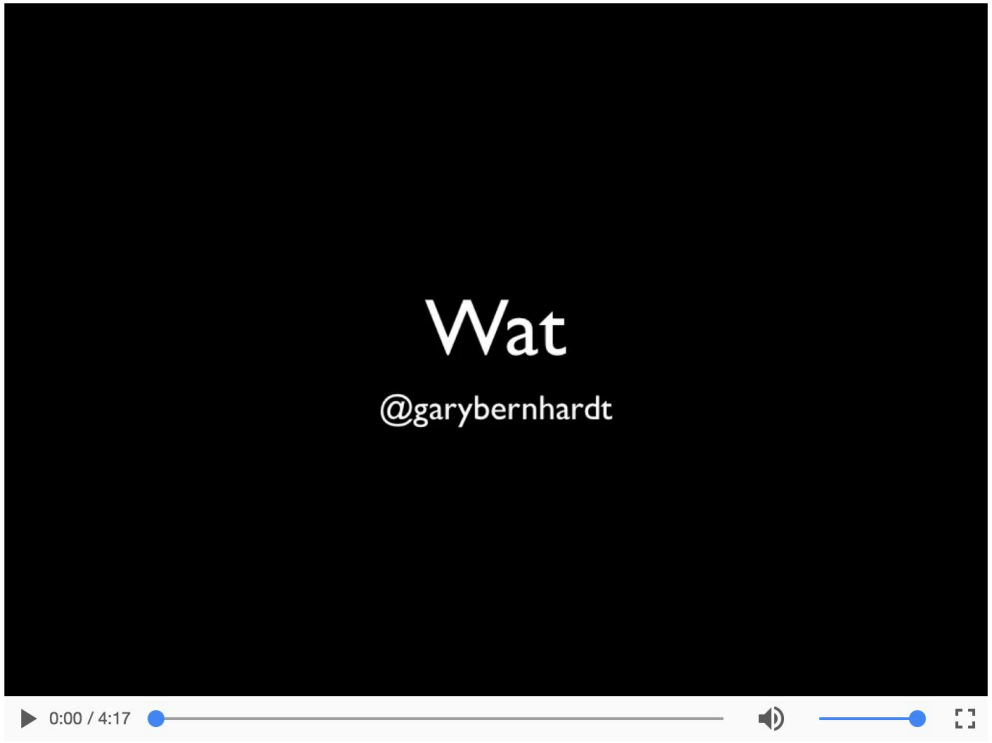
Configure charts to handle data and encoding updates

Write ***visualization software***, not ***project code***

JavaScript Peculiarities

Wat

A lightning talk by Gary Bernhardt from CodeMash 2012



JavaScript has its particularities...

How it feels to learn JavaScript in 2016

No JavaScript frameworks were created during the writing of this article.

Especially now....

What we need to know...

Variable Scope

Getter / Setter methods

Closures

D3 tricks for reusability

Variables are function scoped

```
var someFunction = function(){
    // Define a variable in your function
    var internalVariable = 'only defined in here';


    // Do more things
    return true;
};


// Execute the function
someFunction();


// Try to reference the variable
internalVariable; // Uncaught ReferenceError: internalVariable is not defined
```

Often causes trouble

Variables are function scoped

```
var someFunction = function(){
    // Define a variable in your function
    var internalVariable = 'only defined in here';


    // Do more things
    return true;
};


// Execute the function
someFunction();


// Try to reference the variable
internalVariable; // Uncaught ReferenceError: internalVariable is not defined
```

Can be helpful!

Closures allow us to describe **scoped variables** for **multiple instantiations of an object**.

"A closure is an inner function that has access to the outer (enclosing) function's variables" - source

We leverage closures in many existing patterns

```
// Select an svg from the DOM
var svg = d3.select('#my-svg');


// Set the height property: changes a locally scoped height variable
svg.attr('height', 600);


// Retrieve the height property: returns a locally scoped height variable
svg.attr('height'); // returns 600
```

Writing getter/setter methods

```
// Create a person object
var person = {
    name:"Maria",
    age:22
};


// Write a method that allows you to get or set the `age` attribute
person.ageMethod = function(value) {
    if(!arguments.length) return this.age; // if no value is set, get the age
    this.age = value; // set the age
};


// Get current age
person.ageMethod(); // returns 22
```

Chaining makes these methods easier to use

```
// Select an svg from the DOM
var svg = d3.select('#my-svg');

// Set the height property
svg.attr('height', 600)
     .attr('width', 600)
     .style('background-color', 'green');
```

Enable Method Chaining

```
// Write a method that allows you to get or set the `shoeSize` attribute
person.shoeSizeMethod = function(value) {
    if(!arguments.length) return this.shoeSize;
    this.shoeSize = value;
    return this; // return the object to allow method chaining
};

// Set the age and the shoeSize
person.ageMethod(22) // set the age, return the object
          .shoeSizeMethod(8.5); // set the shoeSize, return the object
```

Demo 1

This demo shows how generalizable getter/setter methods can be used on an object

Manipulate the `input` elements below to change the `course` object

Instructor

credits

Department

Course Object:

```
{
        "title": "Interactive Information Visualization",
        "code": "INFO-474"
}
```

Demo-1

If we write a function that ***returns an object***, that object can ***manipulate variables scoped*** to the function.

```
// Function that returns a new person object
var newPerson = function() {
    // Default values
    var name = 'No Name';
    var age = 100;

    // Define an empty person object to return
    var person = {};

    // Add an `age` property to the person object that will get/set the `age` variable
    person.age = function(value) {
      if(!arguments.length) return age; // returns current value
      age = value; // changes the value of age, only part of this function
      return person; // allows method chaining
    };

    return person; // return the person object when the function is executed
};

// Create a new person object and set the age/name values
var person1 = newPerson() // returns our person object
                .age(22) // sets the age variable within the proper context
                .firstName('Maria'); // sets the name variable within the proper context
```

Demo 2

This demo shows how to leverage **closures** to maintain variable scope.

Manipulate the `input` elements below to change the `people` object(s)

Person-0

Maria

22

Person-1

None

0

Person-2

None

0

People

```
{
        "firstName": "Maria",
        "age": 22
}
```

```
{
        "firstName": "None",
        "age": 0
}
```

```
{
        "firstName": "None",
        "age": 0
}
```

Demo-2

Reusable D3 components

We'll aim to mimic the same reusability structure as D3's core library.

Implementation

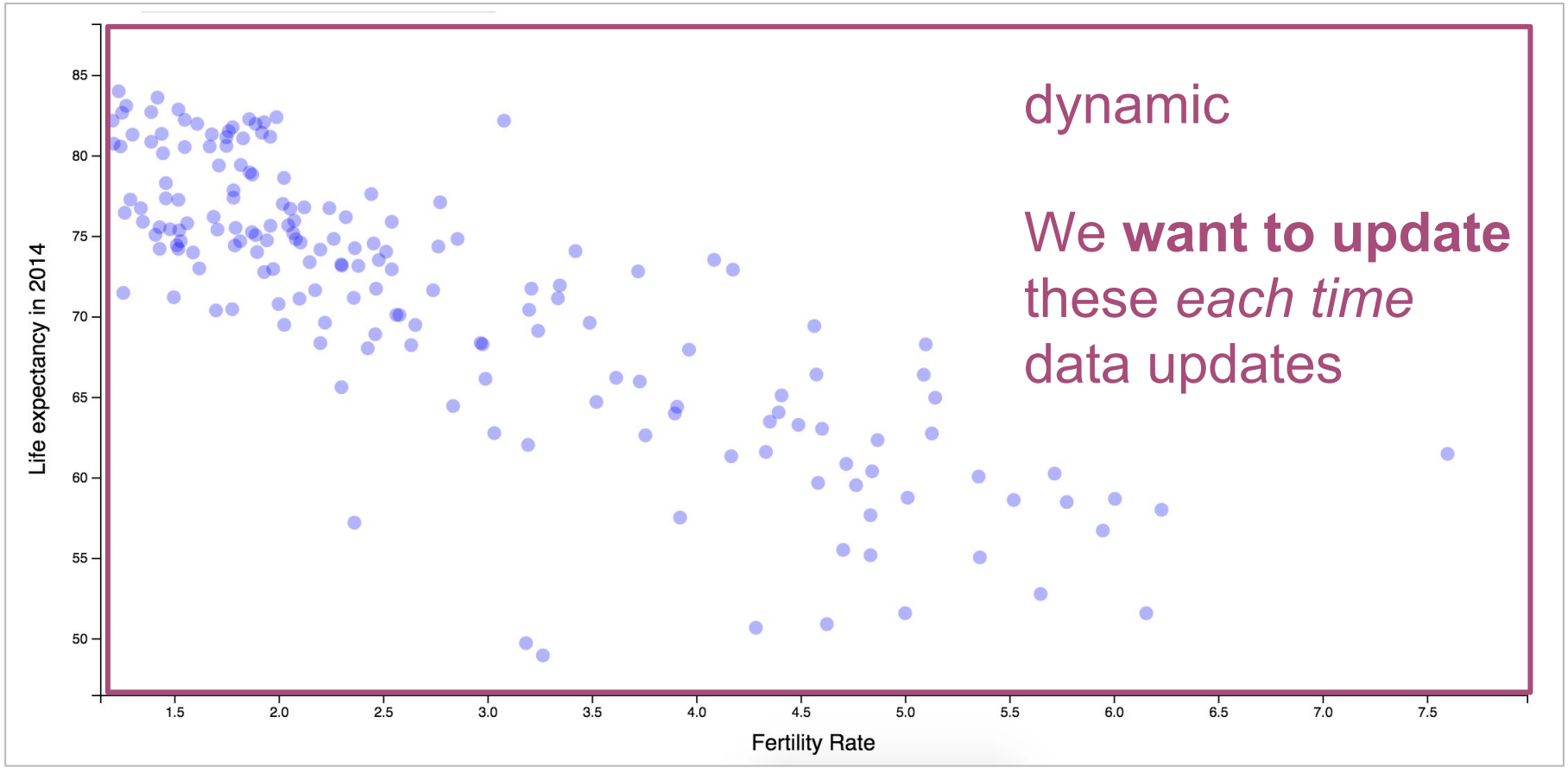
```
// Construct a new axis function using the d3.axisBottom method that returns a function
var axis = d3.axisBottom(); // Function that returns a function using closure pattern

// Set the scale and orientation of the axis function
axis.ticks(5) // Changes a locally scoped variable, returns the function
    .scale(scale); // Changes a locally scoped variable, returns the function

// Create a DOM element in which to render your axis
var axisG = svg.append('g') // Append a 'g' element to your svg in order to render your axis

// Render the axis in the selected axisG
axisG.call(axis); // Call your function in the context of a selected element
```

```
// simplified + commented D3 axis code
function axis() {

    // Default values for the axis
    var ticks = null,
     tickFormat = null;

    // Axis function that gets returned, operates on a selected g element
    function myAxis(context) {
        // Build visual axis components
        path = path.merge(path.enter().insert("path", ".tick")
                   .attr("class", "domain")
                   .attr("stroke", "#000"));

      // put visual elements on the DOM
  }

     myAxis.ticks = function(_) {
         return arguments.length ? (ticks = _, axis) : ticks
     };

    // Return the myAxis function to operate on
    return myAxis;
  };
```

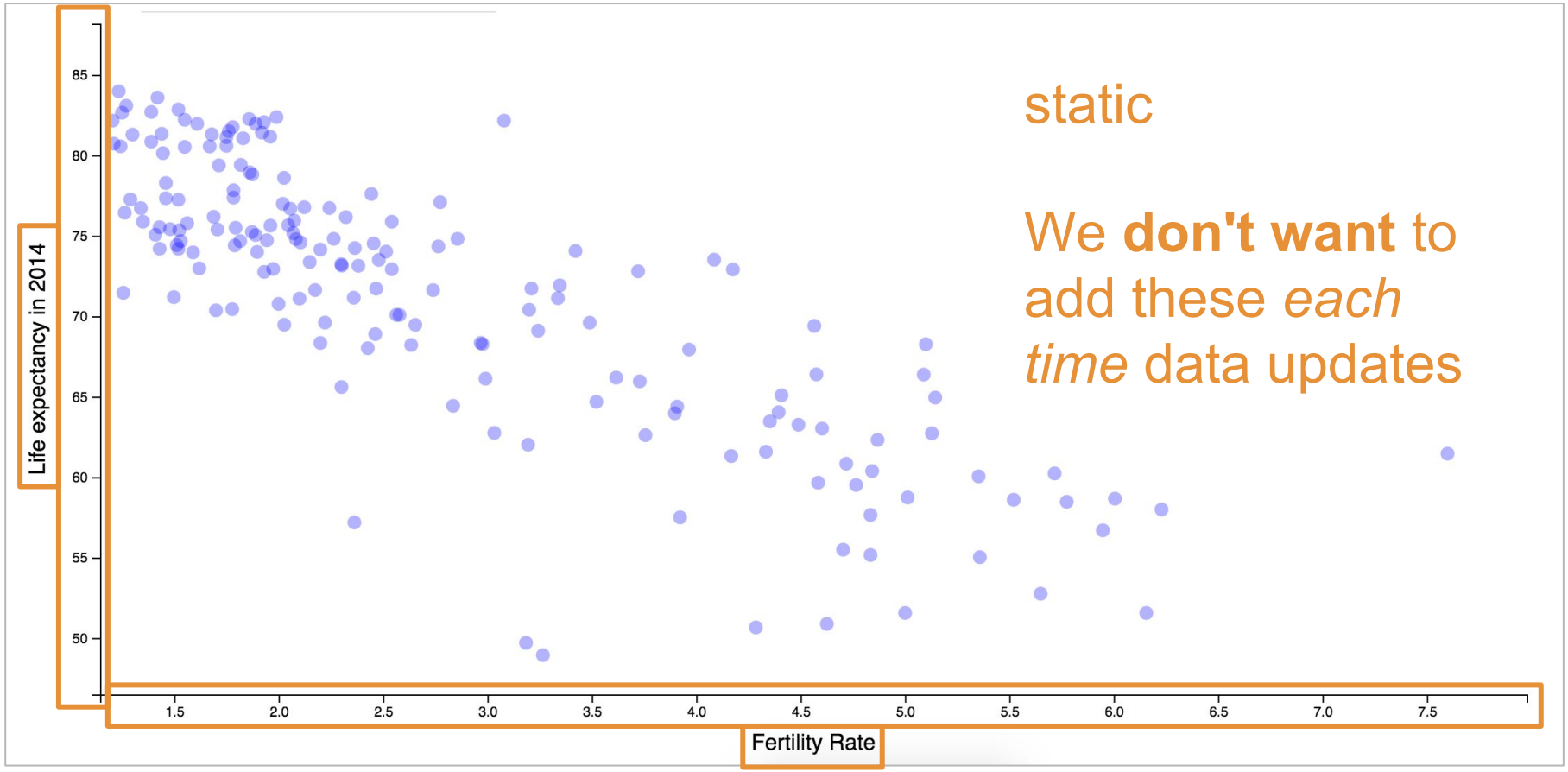
Closure! Function that will manipulate (and access) locally scoped variables.

Getter/setter methods to enable referencing each values of interest.

Static v.s. Dynamic components



Which visual elements are static (added once) v.s. dynamic (may be added / removed)?



Which visual elements are static (added once) v.s. dynamic (may be added / removed)?

How can we determine if an element has already been added to the screen?

```
// Store the data-join in a variable `circles`. Specify the identifier of each element
var circles = svg.selectAll('circle') // select all circles in the svg
                 .data(data, function(d){return d.id}); // bind the data to your selection

// Append a circle element for each observation that was added to the data.
circles.enter() // Determine what elements are new to the dataset
       .append('circle'); // Append a circle for each entering element
       .merge(circles) // Merge update selection to apply changes to entering AND updating
       .attr('r', 5) // set a constant radius of 5
       .attr('cx', function(d) {return d.x}) // specify the x attribute
       .attr('cy', function(d) {return d.y}); // specify the y attribute

// Remove any exiting elements
circles.exit().remove();
```

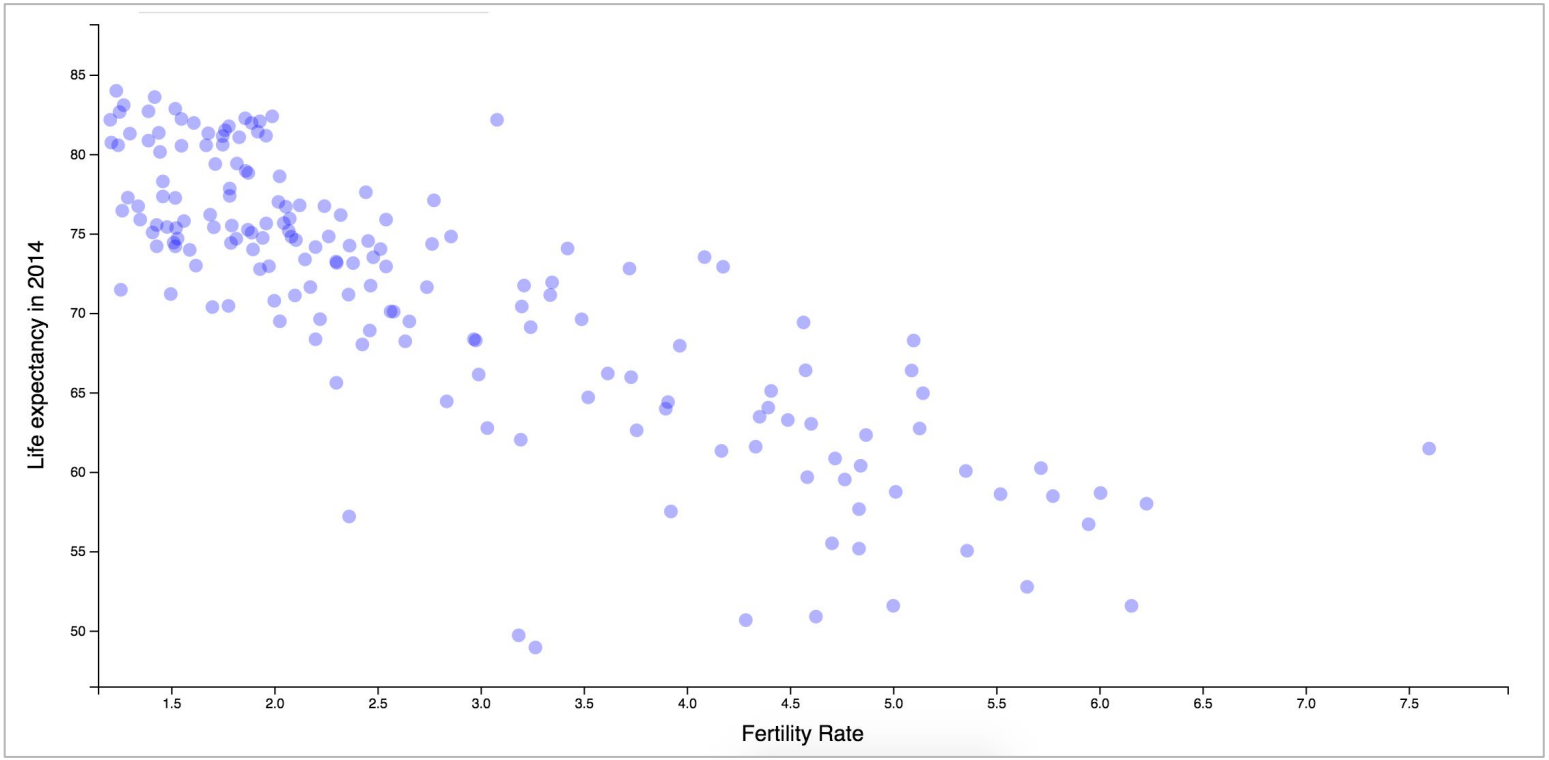
Use the data-join determine which elements are already on the screen

```
var svg = ele.selectAll("svg").data([data]); // bind a single piece of data

// Append static elements (i.e., only added once)
var gEnter = svg.enter() // will return an empty selection if there is an svg
                .append("svg")
                .attr('width', width)
                .attr("height", height)
                .append("g");

// g element for markers
gEnter.append('g') // append a g element to the *entering* g
           .attr('transform', 'translate(' +  margin.left + ',' + margin.top + ')')
           .attr('height', chartHeight)
           .attr('width', chartWidth)
           .attr('class', 'chartG');
```

Do the same thing to conditionally add static elements

Building your API

Making charts reusable means *parameterizing* anything that you want to control.



Width
Height
X title
Y Title
Radius (function?)
Color (function?)
Legend
Scale min/max
Transitions
Scale Type

For a simple chart, what would you want to parameterize?

Basic reusable D3 approach

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

  function myChart() {
    // generate chart here with data join, 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 my;
};
```

Iterate through selections for multiple charts

```
var chart = function() {
  // Set defaults up here

  // Internal function that gets returned
  function my(selection) {
    // For each selected element, perform the function
    selection.each(function(data, i) {
      // generate chart here; `data` is the data and `this` is the element
    });
  }
  return my;
}
```

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!
```

Demo 3

Color: blue

fontSize: 18

# of Paragraphs: 4

Paragraph Number 1

Paragraph Number 2

Paragraph Number 3

Paragraph Number 4

Demo-3

Thanks!

Michael Freeman
@mf_viz
(more to come in the next session...)