Introduction to Data Visualization with D3

Computational Science Center

Barnard College

1.0 Background

Welcome to D3! D3, or Data Driven Documents, is a JavaScript library for creating dynamic data visualizations. D3 is JavaScript and the code executes in the browser, making it cross compatible and easy to get started with.

Why use D3 when there are simpler libraries?

- Customization and flexibility
  ..* Seasons Map
  ..* Trade
  ..* Sound Interaction
- Reusable code
- Pure JavaScript means it will always work, regardless of web framework, etc. Drop it into your HTML, and it's ready to go.
- While there is a learning curve, you don't need to be an expert in JavaScript, HTML, or CSS to use D3!

Typically, we work with D3 in either pure JavaScript files or inside HTML - today we'll create an HTML file and work within that.

1.1 Resources

- d3js.org
- *Interactive Data Visualization for the Web: An Introduction to Designing with D3* by Scott Murray
- Marko Krkeljas | mkrkelja@barnard.edu | Milstein 506
- Anna Carlson | acarlson@barnard.edu | Milstein 511

2.0 Introduction to SVG

- D3 leverages a technology called SVG (Scalable Vector Graphics)
- SVGs are used to draw shapes

- Instead of describing pixels, we give the browser a set of instructions: ..* Draw a blue circle with radius 10cm in the center of a canvas
- We tell the browser which shape, how large, and where to place it

2.1 SVG Shapes

- Circle
- Ellipse
- Line
- Paths
- Texts
- Rectangles

2.2 Setup HTML

We'll write our javascript directly in the HTML file, but if your visualizations get more complex, you'll write pure javascript scripts and reference them in your HTML scripts.

```
<html>
    <head>
        <title>D3 Workshop</title>
    </head>

    <body>

    </body>

</html>
```

2.3 Creating a Rectangle

- Define width and height attributes of an SVG element
- Draw a blue rectangle
- The rectangle takes width, height, x, and y arguments
- In an SVG canvas, the default coordinate (0,0) is the top left.

```
<html>
    <head>
        <title>D3 Workshop</title>
    </head>
    <body>
        <svg width = "200" height = "150">
            <rect x = "100" y = "100" width = "25" height = "25" fill = "blue"></re
        </svg>
    </body>
</html>
```

2.4 Add a circle to the canvas

- Circle attributes are relative to the center of the circle
- Instead of "x" & "y", use "cx" & "cy"
- Don't forget radius! Use "r"

```
<html>
    <head>
        <title>D3 Workshop</title>
    </head>
    <body>
        <svg width = "200" height = "150">
            <rect x = "100" y = "100" width = "25" height = "25" fill = "blue"></re
            <circle cx = "15" cy = "30" r="15" fill="red"></circle>
        </svg>
    </body>
</html>
```

2.5 Add a line to the canvas

- Add a line
- Attributes include: x1, x2, y1,y2, stroke, width

```
<html>
    <head>
        <title>D3 Workshop</title>
    </head>
    <body>
        <svg width = "200" height = "150">
            <rect x = "100" y = "100" width = "25" height = "25" fill = "blue"></re
            <circle cx = "15" cy = "30" r="15" fill="red"></circle>
            <line x1 = "0" y1 = "0" x2="70" y2="100" stroke="green" stroke-width ="
        </svg>
    </body>
</html>
```

2.6 Add a path to the canvas

- Very useful type of SVG
- Used for controlling curves with the "d" attribute
- "d" is a series of "path" commands
- More on path commands:
  ..* [https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d]

```
<html>
    <head>
        <title>D3 Workshop</title>
    </head>
    <body>
        <svg width = "200" height = "150">
            <rect x = "100" y = "100" width = "25" height = "25" fill = "blue"></re
            <circle cx = "15" cy = "30" r="15" fill="red"></circle>
            <line x1 = "0" y1 = "0" x2="70" y2="100" stroke="green" stroke-width ="
            <path fill="none" stroke="red"
                d="M 10,30
                    A 20,20 0,0,1 50,30
                    A 20,20 0,0,1 90,30
                    Q 90,60 50,90
                    Q 10,60 10,30 z"></path>
        </svg>
    </body>
</html>
```

3.0 Import D3

- To do this, just add a script tag to our head with the url of D3's latest version, v5

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>

    </body>
</html>
```

3.1 Create SVG "canvas" with D3 as in 2.3

- Instead of hard-coding an SVG canvas with tags, create canvas with D3 function
- All we are doing is creating a variable called svg which we 'append' an 'svg' HTML element to

- To summarize: svg = <'svg'><' svg'> just as in the previous examples
- We'll place all our D3 code inside of a script tag in our body, with the script type as javascript. In more complex cases, you might have a separate javascript file that you reference here

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            var svg = d3.select("body").append("svg")
        </script>

    </body>
</html>
```

3.2 Method Chaining - add height and width attributes

- Method chaining refers "chaining" together a sequence of functions (methods)
- A method is a fancy name for a function
- Typical syntax is a period "." followed by some function "f" which may or may not accept some parameter/arguments
  ..* Assume a class "Human" with 2 "methods" (functions): stepForward(n), stepBackward(n)
  ..* Both functions accept a parameter "n" which tells "human" how many steps to take
  ..* Method Chaining Example:

```
frank = Human()
frank.stepForward(2).stepBackwards(2).stepForward(10).......
```

..* In JavaScript we usually write it like so (although you don't have to):

```
frank = Human()
 .stepFoward(2)
 .stepBackward(2)
 .stepForward(10)....
```

- We will use the .attr method to append to the variable svg height and width attributes. The function below creates an svg canvas with width and height set to 200 and 150 pixels respectively, just like our earlier examples:

```
<svg> width="200" height="150"> </svg>
```

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            var svg = d3.select("body").append("svg")
                .attr('width', 200)
                .attr('height', 150);
        </script>
    </body>
</html>
```

3.3 Add a blue rectangle

- Add a blue rectangle as in 2.3, but this time with D3
- Note that formatting and spacing has no effect on code execution
- We add the attributes of the rectangle via method chaining

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            var svg = d3.select("body").append("svg")
                .attr('width', 200)
                .attr('height', 150);
             var rectangle = svg.append("rect") // adding the rect here
                .attr("x", 0)
                .attr("y", 0)
                .attr("width", 25)
                .attr("height", 25)
                .attr("fill", "blue");
        </script>
    </body>
</html>
```

3.4 Alternative format

- In the last example, we created a new variable for the rectangle SVG element called rectangle
- Alternatively, you could have chained the original svg variable by appending directly onto it

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            var svg = d3.select("body").append("svg")
            .attr('width', 200)
            .attr('height', 150)
            .append("rect") // adding the rect here
            .attr("x", 0)
            .attr("y", 0)
            .attr("width", 25)
            .attr("height", 25)
            .attr("fill", "blue");
        </script>
    </body>
</html>
```

3.5 Data Array & Indexing

In this example, will hard-code an array called my_array and pick out the values based on their index position:

- The index refers to the "positon" of an item inside an array
- In computer science, the first position is almost always 0 (not 1!)
- Pass the position inside square brackets to index an array:

```
var my_data = [12, 20, 35]

my_data[0] // 12
my_data[1] // 20
my_data[2] // 35
```

3.5.1 Looping through an array

Here we loop through array:

- We declare a random variable i which will be work as our index
- We set i to 0, the first position
- Then we declare: while i is less than my_data.length (3), do the stuff inside the parenthesis
- The .length method returns the "length" of the array (3)
- Lastly, i++ is very common shorthand syntax for count up by one or more formally: i = i + 1

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            var my_data = [12, 20, 35];
            var i;
            for (i = 0; i< my_data.length; i++) {
                alert(my_data[i])
            }
        </script>
    </body>
</html>
```

3.6 Data Array & D3

- This time, we will create a rectangle for each value in the data array

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            var my_data = [12, 20, 35];

            var svg = d3.select("body").append("svg")
                .attr('width', 200)
                .attr('height', 150);
            var rectangles = svg.selectAll("rect")
                .data(my_data);
            rectangles.enter()
                .append("rect")
                .attr("x", 0)
                .attr("y", 0)
                .attr("width", 25)
                .attr("height", 25)
                .attr("fill", "blue")
        </script>
    </body>
</html>
```

3.6.1 Data Array & D3… What?

- Why is there only one rectangle?
- What is .selectAll()?
- What is .data(data)?
- What does .enter() do?

We just bound our data to elements rects elements. To understand data binding, lets turn to Scott Murray:

What is binding, and why would I want to do it to my data? Data visualization is a process of mapping data to visuals. Data in, visual properties out. Maybe bigger numbers make taller bars, or special categories trigger brighter colors. The mapping rules are up to you.

With D3, we bind our data input values to elements in the DOM. Binding is like “attaching” or associating data to specific elements, so that later you can reference those values to apply mapping rules. Without the binding step, we have a bunch of data-less, un-mappable DOM elements. No one wants that."

Let's start with our rectangles variable, where we introduced two new methods: .selectAll() and .data()

```
var rectangles = svg.selectAll("rect")
                            .data(my_data)
```

- What we have done is associated all the items in our array my_data to rect elements
- In effect, we have said: I want to select a rectangle for each item in the array.
- But technically, these elements don't exist yet - as Scott Murray writes:
  ..* "this gets at one of the most common points of confusion with D3: How can we select elements that don’t yet exist? The answer lies with enter()…"

Before we get to enter(), a quick recap:

d3.select("body").append("svg") - Appends an SVG canvas to our HTML page

.selectAll("rect") — Selects all rects - which don't exist yet - it's an "empty selection". Quoting again from Murray, "think of this empty selection as representing the rects that will *soon exist*.

.data(my_data) — Quoting again, "Counts and parses our data values. There are five values in our data set, so everything past this point is executed five times, once for each value."

.enter() — "To create new, data-bound elements, you must use enter(). This method looks at the DOM, and then at the data being handed to it. If there are more data values than corresponding DOM elements, then enter() creates a new placeholder element on which you may work your magic. It then hands off a reference to this new placeholder to the next step in the chain."

- When we call our enter method:

```
rectangles.enter()
              .append("rect")
              .attr("x", 0)
              .attr("y", 0)
              .attr("width", 25)
              .attr("height", 25)
              .attr("fill", "blue")
```

- D3 checks the elements that exist ("rects") and sees that none exist
- D3 then checks the data coming in (that which is "bound" to the variable "rectangles") and sees 3 new elements down the pipeline
- It then creates "placeholder" elements, which we then add attributes, and finally binds it

3.6.2 Why is there still only one blue rectangle?

- If everything is working correctly, shouldn't there be three rectangles, one for each value in my_data?
- Look at the x & y attributes:

```
.attr("x", 0)
.attr("y", 0)
```

- They're stacked on top of each other, with each rectangle is assigned the position (0,0)
- How do you change that?

3.7 Anonymous Functions aka "Lambdas"

- An anonymous function, also called a "lambda", is simply a function with no name
- Lambdas are usually created on the fly, for one time use
- If you are going to reuse a function, it's better to assign it a name, so you can call it later

in your code:
Suppose you're writing a program that sums three values and raises to some power n, and you need to do this often. In this case, you should name the function, and call it whenever you need it:
Example - Not Proper JavaScript!

```
function sumPower(value1, value2, value3, n) {
    var sum_power = (value1 + value2 + value3)^n
    return sum_power;
}
.
..
...
sumPower(1,5,5,3)
.....
sumPower(3,9,2,5)
......
```

But if you only need to do it once, you could write a "lamda" on the fly:

```
function(1,5,5,3){
    return (1 + 5 + 5)^3
}
```

Let's rewrite these two function in proper JavaScript:

```
// Named Function

function sumPower(value1, value2, value3, n){
    var sum_power = Math.pow(value1 + value2 + value3, n)
    return alert(sum_power)
}

// Lambda

function(){
    return Math.pow(3 + 6 + 2, 4) // 14641
}()
```

3.7.1 D3 and Lambdas

- To fix the stacking issue, we're going update the "x" position of each rectangle element
- We'll repl;aced the "0" in .attr("x", 0) with an anyonymous function:

```
    return (i)
}
```

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            var my_data = [12, 20, 35];

            var svg = d3.select("body").append("svg")
                .attr('width', 200)
                .attr('height', 150);
            var rectangles = svg.selectAll("rect")
                .data(my_data);
            rectangles.enter()
                .append("rect")
                .attr("x", function(d,i) {
                    return i
                })
                .attr("y", 0)
                .attr("width", 25)
                .attr("height", 25)
                .attr("fill", "blue")
        </script>
    </body>
</html>
```

3.7.2 Still one rectangle…

- We're still seeing one rectangle, but it's a little wider
- And what are (d,i)?
- The d refers to the data point in our array which we bound rectangle elements, and i refers to the index
  How does D3 know what those variables are if we never defined them?

It knows because we "bound" the data, and then created "references" with the enter method. Behind the scenes, D3 is performing something similar to the loop from 3.5.1 and since it needs to create three rectangles - one for each item in our array - it executes the anonymous function for each item.

This is why the rectangle is a bit wider this time, since the x position has changed slightly, from 0,0,0 to 0,1,2

If we were to manually code the rectangles, they would look something like this:

```
// Rectangle 1
.append("rect")
.attr("x", 0})
.attr("y", 0)
.attr("width", 25)
.attr("height", 25)
.attr("fill", "blue")

// Rectangle 2
.append("rect")
.attr("x", 1})
.attr("y", 0)
.attr("width", 25)
.attr("height", 25)
.attr("fill", "blue")

// Rectangle 3
.append("rect")
.attr("x", 2})
.attr("y", 0)
.attr("width", 25)
.attr("height", 25)
.attr("fill", "blue")
```

The problem we still have is that the rectangles overlap since the space between them isn't far enough - we're changing the x position by 1 unit, but the rectangles are 25 units wide.

Multiply the index position by 30, and you won't have any overlap:

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            var my_data = [12, 20, 35];

            var svg = d3.select("body").append("svg")
                .attr('width', 200)
                .attr('height', 150);
            var rectangles = svg.selectAll("rect")
                .data(my_data);
            rectangles.enter()
                .append("rect")
                .attr("x", function(d,i) {
                    return i*30
                })
```

```
                   .attr("y", 0)
                   .attr("width", 25)
                   .attr("height", 25)
                   .attr("fill", "blue")
           </script>
       </body>
   </html>
```

The last example illustrated a simple, but extremely powerful concept in D3 - and computer science in general - the idea that we can pass functions as parameters to other functions.

By passing functions instead of values, we expand on our capabilities to with data. By binding our data to certain "elements" or "objects" and then using functions to transform the data, you begin to understand the idea behind D3 (data driven documents)

4.0 Binding json data

Now that we have bound our rectangles to the my_data array and displayed them, we'll go through an example of binding rectangles to actual data in json format. JSON stands for JavaScript Object Notation, and is a common data structure for JavaScript.

On a computer, open the data.json file in a text editor to see the structure of our dataset.

4.1 JSON & D3

To visualize information from a JSON file in D3, you'll need to use the D3 function d3.json(). Starting with the most recent version of D3, v5, you'll need to perform any D3 functions inside of this d3.json() function. Read more about it here (link).

Here, we put most of our code from before inside of d3.json(), using data.json as our input for d3.json. You will see 4 rectangles instead of 3, because we now have 4 observations.

We also make the svg canvas larger (500 x 500), to accomodate for the bar chart we are about to build!

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            d3.json("data.json").then(function(data) {
                var svg = d3.select("body").append("svg")
                .attr('width', 500)
                .attr('height', 500);

                var rectangles = svg.selectAll("rect")
                .data(data);
            rectangles.enter()
                .append("rect")
                .attr("x", function(d,i) {
                    return i*30
                })
                .attr("y", 0)
                .attr("width", 25)
                .attr("height", 25)
                .attr("fill", "blue")
            })
        </script>
    </body>
</html>
```

4.2 Visualizing JSON variables

Now that we have rectangles representing our data points, we simply refer to attributes of those observations if we wish to visualize them as shapes or objects. In this example, we have two variables of interest: old and new. 'Old' represents the number of buildings in that borough built before 1939. 'New' represents the number of buildings built after 2014.

If we are interested in the 'new' buildings for a given borough, instead of using 'd', we can use 'd.new'.

This is why it is important to work with well-cleaned data in d3, as it will save you time and confusion later (for example, if your varaiable names had spaces or were otherwise messy).

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            d3.json("data.json").then(function(data) {
                var svg = d3.select("body").append("svg")
                .attr('width', 500)
                .attr('height', 500);

                var rectangles = svg.selectAll("rect")
                .data(data);
            rectangles.enter()
                .append("rect")
                .attr("x", function(d,i) {
                    return i*30
                })
                .attr("y", 0)
                .attr("width", 25)
                .attr("height", function(d,i) {
                    return d.Number;
                })
                .attr("fill", "blue")
            })
        </script>
    </body>
</html>
```

Here, we've changed the height attribute to return d.Number, to visualize our "Number" variable. However, the rectangles are a little short, so feel free to play around with scaling the height, first by multiplying d.Number by 10, and then by 100.

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            d3.json("data.json").then(function(data) {
                var svg = d3.select("body").append("svg")
                .attr('width', 500)
                .attr('height', 500);

                var rectangles = svg.selectAll("rect")
                .data(data);
            rectangles.enter()
                .append("rect")
                .attr("x", function(d,i) {
                    return i*30
                })
                .attr("y", 0)
                .attr("width", 25)
                .attr("height", function(d,i) {
                    return d.Number*100;
                })
                .attr("fill", "blue")
            })
        </script>
    </body>
</html>
```

4.3 Making it look like a real bar chart (kind of)

As of right now, our bar chart is showing our data upside down. In order to make it right side up, we'll need to set a y value that is dependent on the height of the canvas and the value. Our new y value will be 500 - d.Number*100, and our height will stay the same.

```
<html>
    <head>
        <title>D3 Workshop</title>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type = "text/javascript">
            d3.json("data.json").then(function(data) {
                var svg = d3.select("body").append("svg")
                .attr('width', 500)
                .attr('height', 500);

                var rectangles = svg.selectAll("rect")
                .data(data);
            rectangles.enter()
                .append("rect")
                .attr("x", function(d,i) {
                    return i*30
                })
                .attr("y", function(d,i) {
                    return 500 - d.Number*100;
                })
                .attr("width", 25)
                .attr("height", function(d,i) {
                    return d.Number*100;
                })
                .attr("fill", "blue")
            })
        </script>
    </body>
</html>
```

And that is the beginning of a bar chart in d3, using "real" data!