D3 for R Users

Joyce Robbins

2019-11-05

Contents

Note to readers

This guide adapts Scott Murray's *Interactive Data Visualization for the Web, 2nd edition*–a required text for GR5702–for the needs of this course. Be sure to get the second edition, which is a comprehensive update to D3 version 4. The first edition uses D3 version 3, which is not compatible. (To add to the complication, the current version of D3 is v5. However, since differences between v4 and v5 are minimal, unless otherwise indicated in this guide, the code in *IDVW2* will work with either.)

We rely on the text heavily but also deviate from it in several ways. *IDVW2* is written for graphics designers not data science students so the pain points are somewhat different.

In terms of content, we will use certain ES6 conventions not covered in *IDVW2* that make coding easier (and more like R!). We use different examples, though you are strongly encouraged to study Murray's code examples in addition to reading the text. Particularly through the first half, we don't follow the text in order, so always refer to this guide first which will direct you to the pages of the text that you should read.

This is very much a work-in-progress so please submit issues on GitHub to provide feedback and edit or add text by submitting pull requests. (Click the icon at the top of each page to get started. More detailed instructions are available on edav.info.)

Chapter 1

Jump in the deep end

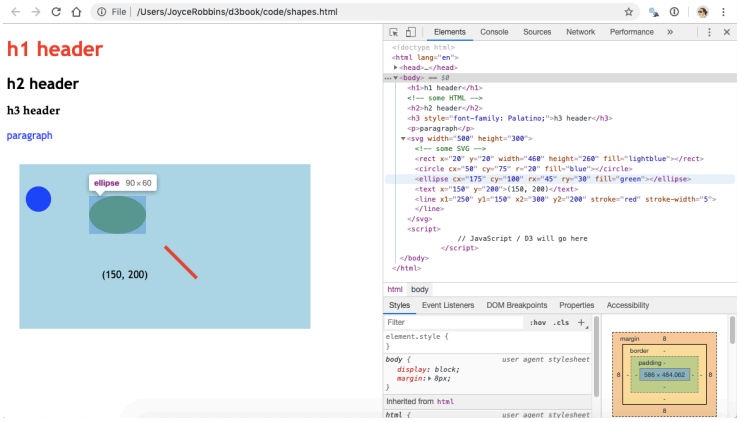
Let's skip the explanations and start coding in D3 right now. Why? So you can see the benefits and know what you're working toward when you get stuck in the weeds. Then we'll go back and start learning step by step.

1.1 Get ready

1. If you don't have it already, download the Chrome browser.
2. Download a copy of `shapes.html`: open the following page and then click *File, Save Page As…*: `shapes.html`. (Or download a zip of the whole repo. Clicking here will start the download. Or fork and clone the repo).
3. If Chrome is your default browser, open `shapes.html` by double clicking it. Otherwise, open it with *File, Open File…* in Chrome.

1.2 Elements tab

1. In Chrome, click *View, Developer, Developer Tools,* then the Elements tab.
2. Hover the mouse over various elements in the `<body> ... </body>` section. Observe the highlighted sections in the rendered web page on the left of the screen. Click on the mini black triangles to the left of the `<body>` and `<svg>` tags if needed to open these sections of the DOM tree. Your screen should look like this:



3. Now try the reverse: right click on elements on the web page, choose "Inspect" and see what is highlighted in the Elements pane. Get comfortable with the connection between the code on the right and the rendered elements on the left.

1.3 Console tab

1. Switch to the Console tab, next to the Elements tab. Let's practice running some code. Note that the code is unrelated to the `shapes.html` web page that we have open.

   We will spend a lot of time in the Console since it's interactive – think R console. Eventually we will switch to including JavaScript/D3 in `.html` or `.js` files and use the Console only for testing things out or debugging.

2. Type the following lines of code at the prompt (>), press enter after each line–that is, after the semicolon (;)–and see what happens:

```
3 + 4;

"3" + "4";

 x = [1, 2, 3];

x[1];

x + 1;

y = {a: 3, b: 4};

y["b"];
```

1.4 Modify elements

1. Now we'll start using D3 to manipulate elements on the page. Try the following, by entering one line at a time in the Console as before:

```
d3.select("circle").attr("cx", "200");

d3.select("circle").attr("cx", "500");

d3.select("circle").attr("cx", "100");

d3.select("circle").attr("r", "30");

d3.select("circle").attr("r", "130");

d3.select("circle").attr("r", "3");

d3.select("circle").attr("fill", "red");

d3.select("circle").attr("fill", "aliceblue");

d3.select("circle").attr("fill", "lightseagreen");
```

   Note that "select" and "attr" are separate operations chained together with "." – think pipe (%>%) operator.

2. Refresh the page. What happened?

3. Go to Elements. Look at the value of the `y1` attribute of the SVG `<line>` element. Go back to the Console and enter the following:

```
d3.select("line").attr("y1", "10");
```

4. Switch back to Elements and observe. What happened?

5. Stay in Elements and refresh the page. What happened to `y1`?

6. Return to the Console to make style changes to the HTML elements:

```
d3.select("h1").style("color", "purple");

d3.select("h2").style("font-size", "50px");

d3.select("h2").style("font-family", "Impact");
```

1.5 Transitions

1. Try these:

```
d3.select("circle").transition().duration(2000).attr("cx", "400");

d3.select("ellipse").transition().duration(2000).attr("transform", "translate (400, 400)");

d3.select("line").transition().duration(2000).attr("x1", "400");

d3.select("line").transition().duration(2000).attr("y1", "250");

d3.select("p").transition().duration(2000).style("font-size", "72px");
```

2. Experiment with more transitions.

1.6 Interactivity

1. Set up a function to turn the fill color to yellow:

```
function goyellow() {d3.select(this).attr("fill", "yellow")};
```

2. Add an event listener to the circle that will be trigger a call to `goyellow()` on a mouseover:

```
d3.select("circle").on("mouseover", goyellow);
```

3. Test it out.
4. Add the same event listener to the ellipse. Test it out.
5. Create a function `goblue()` that changes the fill color to blue.
6. Add event listeners to the circle and ellipse that will trigger a call to `goblue()` on a *mouseout*. Test out your code.
7. Try out a click event. (Note the use of an anonymous function.)

```
d3.select("line").on("click", function()
  {d3.select(this).attr("stroke-width", "10");});
```

8. Try another click event. What's happening?

```
d3.select("svg").on("click", function()
  {d3.select("text").text(`(${d3.mouse(this)})`)});
```

Ok, now that we have a taste for what D3 can do, let's break it down, which we'll do in the next chapter.

Chapter 2

Web tech

Read: Chapter 3 "Technology Fundamentals" (pp. 17-62)

There is a lot of material in this chapter. It is worth making the effort to learn it now and start D3 with a solid foundation of elementary HTML/CSS/SVG/JavaScript.

Here we examine `shapes.html` from Chapter 1 to see how the various technologies are combined into a single document.

2.1 HTML

Note that `shapes.html` has an HyperText Markup Language or `.html` extension; HTML in fact provides the structure for the document. It has a `<head>` and `<body>` section.

In the `<head>` section we use `<script>` tags to link to the D3 library:

```
<script src="https://d3js.org/d3.v5.min.js"></script>`
```

2.2 CSS

CSS (Cascading Style Sheets) is used for styling web pages, and more importantly for our purposes, selecting elements on a page or in a graphic. We will generally work with internal style sheets since it's simpler when starting out to have everything in one document. External style sheets, however, are generally the preferred method for web design.

2.2.1 Internal style sheet

`shapes.html` has an *internal style sheet*: CSS style information appears in the `<head>` section marked off with `<style>` tags:

```
<style type="text/css">
    h1 {color:red;}     /* CSS styling */
    p {color:blue;}
</style>
```

Here we specify that all HTML `<h1>` headers should be red and all HTML paragraphs `<p>` should be blue. This is an example of an *internal style sheet*. Later we will consider alternatives: *external style sheets* and *inline styling*.

Styling for coder designed classes is also specified in this section. For example, we could style a "formal" class as such:

```
<style type="text/css">
    .formal {color: red;
        font-size: 30px;
        font-family: Lucida Calligraphy;
        }
</style>
```

Note that classes are defined by the "." before the name.

2.2.2 External style sheets

External style sheets are `.css` files that contain styling information and are linked to with a `<link>` tag in the `<head>` section of an HTML document:

```
<head>
    <link rel="stylesheet" href="style.css">
</head>
```

External style sheets are the preferred way of styling as they can easily be modified without changing the web page; in fact, the motivation for CSS came from a desire in the early days of the internet to separate styling from content.

Developers have the option now of choosing premade themes, which are shared through external style sheets. They can be quite complex. The `.css` file for the Minty theme from Bootswatch, for example, contains over 10,000 lines.

CSS Zen Garden demonstrates the power of external style sheets: the same HTML document takes on very different looks depending on the stylesheet to which it is linked.

2.2.3 Inline styling

With inline styling, styling is added to each tag individually:

```
<span style="color: white; background-color: fuchsia; font-family: impact;
      font-size: 24px; border-style: solid; border-color: limegreen;
      border-width: 3px">
      Styled inline
</span>
```

This is how early web pages were styled. To take a step back in time, use developer tools to view the source code for the main page of www.dolekemp96.org, an old web site that has been maintained for historical purposes. As you can see, it's a tedious way of writing content, which internal and external style sheets eliminate.

Although you will not be adding inline styling manually, you will notice that when we select elements and change the styling with D3, the modifications are made inline. In other words, we do not make changes to the elements directly, not via a style sheet.

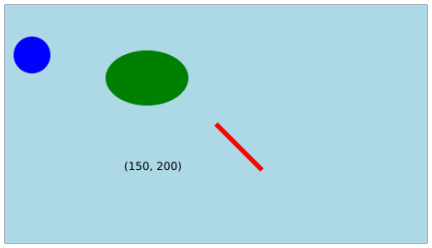
2.3 SVG

SVG (Scalable Vector Graphics) is a human readable graphics format that facilitates manipulation of individual elements. You may be familiar with `.svg` files. Here we have SVG graphics within `<svg>` tags in the `<body>` section of the HTML document:

```
<svg width="500" height="300">  <!-- some SVG -->
    <rect x="20" y="20" width="460" height="260" fill="lightblue"></rect>
    <circle cx="50" cy="75" r="20" fill="blue"></circle>
    <ellipse cx="175" cy="100" rx="45" ry="30" fill="green"></ellipse>
    <text x="150" y="200">(150, 200)</text>
    <line x1="250" y1="150" x2="300" y2="200" stroke="red" stroke-width="5"></line>
</svg>
```

Rendered:

‘



There are very few SVG tags that you'll need to know, and once we get going with D3, you will not have to code any SVG manually. It is worth doing a little to become familiar with the format and in particular to get used to the new location of the origin.

2.4 JavaScript

JavaScript is the most common language for making web pages interactive. Code is executed when pages are opened or refreshed. So far we have run JavaScript in the Console, but have not included it in the web page itself. When we do so, it will be between `<script>` tags in the `<body>` section of the HTML document, or in a separate `.js` file.

2.5 D3

D3 (Data Driven Documents) is a JavaScript library well suited to interactive graphics. As such, it is also included between `<script>` tags in the `<body>` section. For D3 to work, you must link to the D3 library in the `<head>` section of the document.

There seems to be a misconception that D3 is a high level language. It is not. You will be working on the pixel level to create graphics, including drawing your own axes and doing other things that you're not used to doing if you've been working in R or Python.

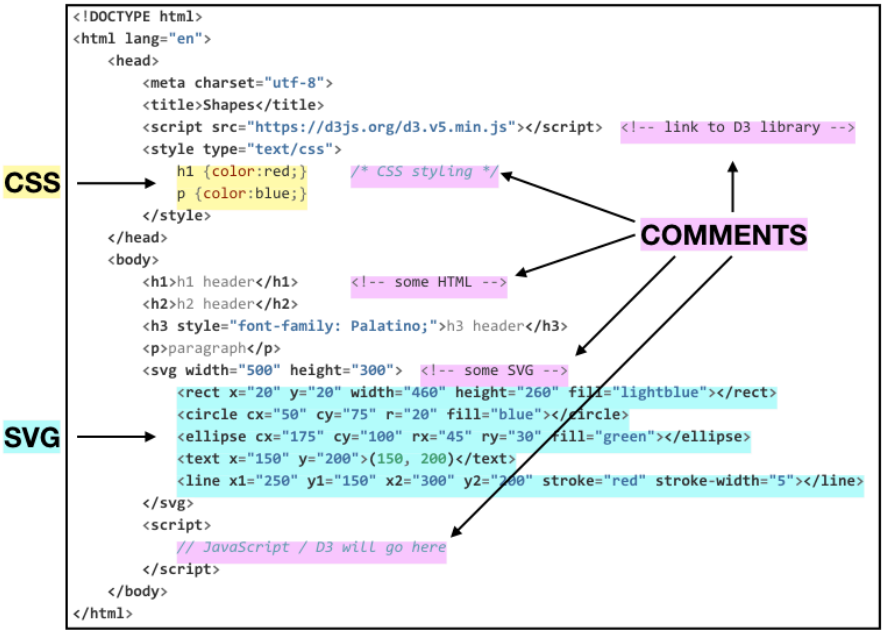
> *On the bright side, after D3, you will gain a new appreciation for base R graphics. You will write code such as* `plot(iris$Sepal.Length, iris$Sepal.Width, pch = 16, col = iris$Species, las = 1, xlab = "Sepal.Length", ylab = "Sepal.Width")` *and think: wow, there are axes! Amazing!*

It is legitimate to ask why you need to know D3 as a data scientist. Many if not most of you will not be coding in JavaScript from the ground up in your future careers. However, it's a great way to learn how interactive graphics work under

the hood, and will give you a solid foundation which you can draw on to tweak visualizations that you build with high level tools such as Plotly.

2.6 HTML tree

While `shapes.html` appears as a single consistent document, it is actually comprised of multiple languages. HTML, CSS, and SVG are already there, and we will be adding JavaScript / D3 soon.



Of note:

- An HTML document is composed of lines or sections set off with tags. In particular `<style> ... </style>`, `<svg> ... </svg>`, and `<script> ... </script>` indicate the inclusion of CSS, SVG, and JavaScript/D3 respectively.
- For D3 to work, you must link to a D3 library. Here we link to an online version, but you can also download a copy from https://d3js.org and reference the local copy with:

```
<script src="d3.js"></script>
```

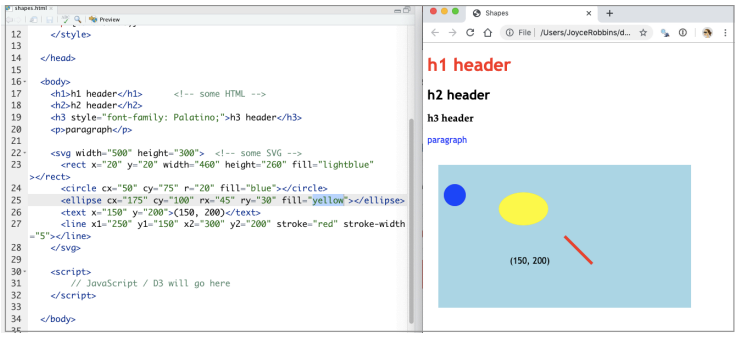
- Comment syntax varies with language:

- `<!-- single or multiline HTML or SVG comment -->`
- `/* single or multiline CSS comment */`
- `// single line JavaScript comment`
- `/* JavaScript multiline comment */`

2.7 Exercise

Download a fresh copy of `shapes.html`. (Reminder: open the following page and then click *File, Save Page As…*: `shapes.html`). Open the file in a text editor of your choice on one half of your screen. If you don't want to think about it, just use RStudio since it's already installed and provides syntax highlighting for `.html`. On the other half of your screen open the same file in Chrome. Developer Tools should not be open; we will not be using the Console. As you make changes to the `.html` file, save the file and then refresh the browser to see the effects. Keyboard shortcuts to save and refresh are helpful here.

Your screen should look like this:



1. Add an additional circle to the svg.
2. Add styling to the internal style sheet to style circles.
3. Add two additional paragraphs use the `<p>` tag.
4. Add an ID attribute to one of the circles.
5. Add a class attribute to two of the `<p>` tags.
6. Use the internal style sheet to style paragraphs of the class you created in 5.
7. Adjust additional elements as desired.

Chapter 3

D3 in the Console

3.1 Selections

3.1.1 Select by tag

The ability to select elements on a page is key to being able to manipulate them. `d3.select()` will select the first match; `d3.selectAll()` will select all matches.

```
d3.select("svg").select("circle");
```

selects the first circle in the order in which circles appear in the `<svg>` grouping. If there were more than one circle we could select them all with:

```
d3.select("svg").selectAll("circle");
```

We can select HTML elements by tag in the same way:

```
d3.select("body").select("h1");
d3.select("body").selectAll("h1");
```

3.1.2 Select by class

Classes are selected by adding a "." before the class name:

```
d3.select("svg").selectAll("circle.apple")
```

This provides one method of selecting a certain collection of elements of the same type.

3.1.3 Select by ID

IDs differ from classes in that they are unique identifies. IDs are selected by adding a "#" before the ID:

```
d3.select("svg").select("circle#henry");
```

3.1.4 Store selections

It is often helpful to store selections for later use. Here we store the svg selection in `mysvg`:

```
var mysvg = d3.select("svg");
```

> *The JavaScript community is moving toward using* `let` *and* `const` *instead of* `var`*; we, however, will stick with* `var` *to be consistent with* IDVW*. Of course you're welcome to use* `const` *and* `let` *instead, and if so, may find these articles helpful: Let It Be - How to declare JavaScript variables and ES2015 const is not about immutability.*

Store circle selection in a variable:

```
var svg = d3.select("svg");

var circ = svg.selectAll("circle");
```

3.2 Modify existing elements

Try out the code in this section with a downloaded copy of five_green_circles.html opened in Chrome and the Console visible.

3.2.1 Modifying attributes

link to get or set attribute API

```
d3.select("circle").attr("r");          // see radius

d3.select("circle").attr("r", "10");    // set radius to 10
```

3.2.2 Modifying styles

link to get or set style API

```
d3.select("h1").style("color");

d3.select("h1").style("color", "blue");
```

> *It is often difficult to remember whether to use* `.attr()` *or* `.style()` *In general, properties such as position on the SVG, class, and ID are* attributes*, while decorative properties such as color, font, font size, etc. are* styles*. However, in some cases, you can use either. For example, the following both make the circle blue:*

```
d3.select("circle").attr("fill", "blue");

d3.select("circle").style("fill", "blue");
```

The first will add a `fill="blue"` *attribute to the* `<circle>` *tag, while the latter will add* `style="fill: blue;"`*. All is well and good until you find yourself with both in the same tag, in which case the* `style` *property will take precedence. The bottom line: don't mix the two options because it can cause problems.*

> *To further complicate matters,* `.style()` *is just shorthand for* `.attr("style", "...")` *so the following are in fact equivalent:*

```
d3.select("circle").style("fill", "blue");

d3.select("circle").attr("style", "fill: blue;");
```

In other words, style is an attribute!

3.2.3 Modifying text

This section is interactive. Hover over code as directed to observe effects.

HTML text

```
<p id="typo" class="fancy">Manhatten</p>
```

Manhatten

Hover to execute this code (and fix the typo):

```
d3.select("#typo").text("Manhattan");
```

SVG text

```
<svg width="500" height="100">
  <rect width="500" height="100" fill="#326EA4"></rect>
  <text id="svgtypo" x="50" y="70" fill="white" font-weight="bold" font-size="40px">
     Web scrapping is fun.</text>
</svg>
```

Web scrapping is fun.

Hover to execute this code (and fix the typo):

```
d3.select("#svgtypo").text("Web scraping is fun.");
```

> *The SVG `<text>` tag can be tricky. It differs from HTML text tags (`<p>`, `<h1>`, `<h2>`, etc.) in that it has `x` and `y` attributes that allow you to position text on an SVG canvas. Unlike HTML, the fill attribute controls the color of the text. Compare:*

```
d3.select("p").style("color", "red");   // HTML

d3.select("text").attr("fill", "red");  // SVG
```

3.2.4 Moving SVG text

```
<svg width="600" height="100">
  <rect width="600" height="100" fill="#326EA4"></rect>
  <text id="moveleft" x="200" y="70" fill="white" font-weight="bold" font-size="40px">
      I want to move left.</text>
</svg>
```

I want to move left.

Hover to execute this code:

```
d3.select("#moveleft").attr("x", "20").text("Thanks, now I'm happy!");
```

3.3 Adding elements

3.3.1 HTML

Continue trying out code with `five_green_circles.html` open in Chrome.

The following adds a `<p>` tag but doesn't change how the page looks, since there's no text associated with it.

```
d3.select("body").append("p");
```

To add text, use `.text()`:

```
d3.select("body").append("p").text("This is a complete sentence.");
```

> *To debug adding an element, go to the Elements tab to see what was added and where. If an element is in the wrong place in the HTML tree, it will not be visible.*

3.3.2 SVG

Likewise, here we add a `<circle>` to the `<svg>`, but we can't see it since it has no attributes.

```
d3.select("svg").append("circle");
```

Adding attributes will create visible circles:

```
d3.select("svg").append("rect").attr("x", "0").attr("y", "0")
    .attr("width", "500").attr("height", "400").attr("fill", "lightblue");

d3.select("svg").append("circle").attr("cx", "200")
    .attr("cy", "100").attr("r", "25").attr("fill", "orange");

d3.select("svg").append("circle").attr("cx", "300")
    .attr("cy", "150").attr("r", "25").attr("fill", "red");
```

We can use a saved selection to assist in creating a new element:

```
mysvg = d3.select("svg");

mysvg.append("circle").attr("cx", "250").attr("cy", "250").attr("r", "50")
  .attr("fill", "red");
```

3.4 Removing elements

These methods will remove matching elements in order, starting with the first find in the document.

3.4.1 HTML

```
d3.select("p").remove();
```

3.4.2 SVG

```
d3.select("svg").select("circle").remove();

d3.select("svg").selectAll("circle").remove();
```

3.5 Exercise

Download and open a fresh copy of five_green_circles.html in Chrome with Developer Tools open. Do the following in the Console with D3:

1. Select the circle with ID "henry" and make it blue.
2. Select all circles of "apple" class make them red.
3. Select the first circle and add an orange border ("stroke"), and stroke width ("stroke-width") of 5.
4. Select all circles of "apple" class and move them to the middle of the svg.

3.6 Binding data... *finally!*

To follow along with the code in this section, download and open a fresh copy of six_blue_circles.html.

Bind data:

```
d3.select("svg").selectAll("circle").data([90, 230, 140, 75, 180, 25]);
```

Check data binding:

```
d3.select("svg").selectAll("circle").data();
```

Set x-coordinate of each circle to data value using arrow function:

```
d3.select("svg").selectAll("circle").attr("cx", d => d);
```

Set x-coordinate of each circle to data value with a JavaScript function:

```
d3.select("svg").selectAll("circle").attr("cx", function(d) {return d;});
```

We'll bind a new set of data to the circles, this time storying the dataset in a variable:

```
var dataset = [50, 80, 110, 140, 170, 200];
```

We'll also store a selection of all circles before binding the data:

```
circ = d3.select("svg").selectAll("circle");
```

And now, the data bind:

```
circ.data(dataset);
```

Nothing appears to have happened; the circles remain the same and there is no evidence of any changes looking at the circles in the DOM (see Elements tab).

We can check that the data are indeed bound with:

```
circ.data();  // now we see data
```

Modify elements w/ stored selections, bound data:

```
circ.attr("cx", function(d) {return d;});

circ.attr("cx", function(d) {return d/2;});

circ.attr("cx", function(d) {return d/4;}).attr("r", "10");
```

Same as above, using arrow functions:

```
circ.attr("cx", d => d);

circ.attr("cx", d => d/2);

circ.attr("r", d => d/4).attr("r", "10");
```

Note that if we bind a new set of data to the DOM elements, the original set will be overwritten:

```
var newdata = [145, 29, 53, 196, 200, 12];

circ.data(newdata);

circ.transition()
    .duration(2000)
    .attr("cx", d => 2*d);
```

3.7 Exercise

With `six_blue_circles.html` open in Chrome, practice binding data to the circles and modifying the circles based on the data as in the examples above.

Chapter 4

More chapters coming soon

via GIPHY