# Using the D3.js framework for data visualization.

As background, review <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4264639/>

First download the current frame at <https://d3js.org> My standalone examples are using v5.15

Now create a directory for your work. I have mine at my user root inside my gitcheckouts folder: ~/gitcheckouts/BIOS824d3

For linux and MacOS, open a terminal window and move to your working directory. For me, that is:

cd gitcheckouts/BIOS824d3/

Now initialize a git repository, and on the MacOS tell git to ignore .DS\_Store files:

#initialize git repository

git init

#add ignore .DS\_Store files to .gitignore

echo "" >> .gitignore

echo "# Ignore .DS\_Store files" >> .gitignore

echo ".DS\_Store" >> .gitignore

Now create a ‘js’ directory and move the d3 files into it

Once that is done, create an html document shell. I called mine ‘homework5d3.html’

Put the following code into the html file:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>D3 Homework and tutorial</title>

</head>

<body>

<div class="greetings">

<h1>Greetings!</h1>

<p>Hello World!</p>

</div>

<script>

// d3 code goes here..

d3.select("div.greetings").append("h2").text("Hello World! ").append("span").text("from D3.js");

</script>

</body>

</html>

Go ahead and add the files to git and make your initial commit:

git add .

git commit -m "Initial d3 project commit"

I also added some tutorial files to my directory. I see this on commit:

6 files changed, 18529 insertions(+)

create mode 100644 .gitignore

create mode 100644 D3handsOn.pptx

create mode 100644 d3js\_tutorial.pdf

create mode 100644 homework5d3.html

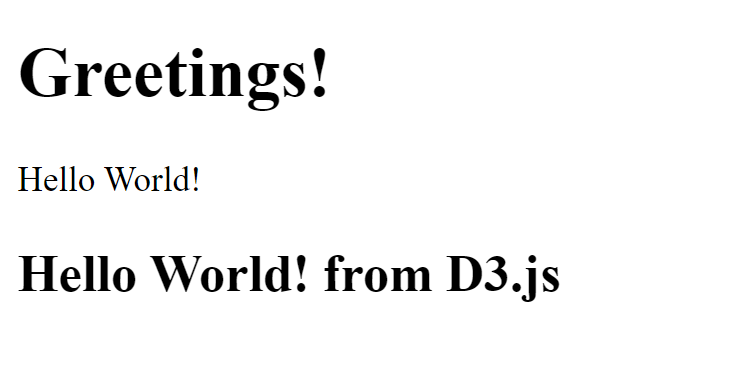
create mode 100644 js/d3.js

create mode 100644 js/d3.min.js

ccis:BIOS824d3 $

Now we have am initial file, directory, and git commit.

Q1. Load the file in a browser. Take a screenshot of what you see and attach it below.

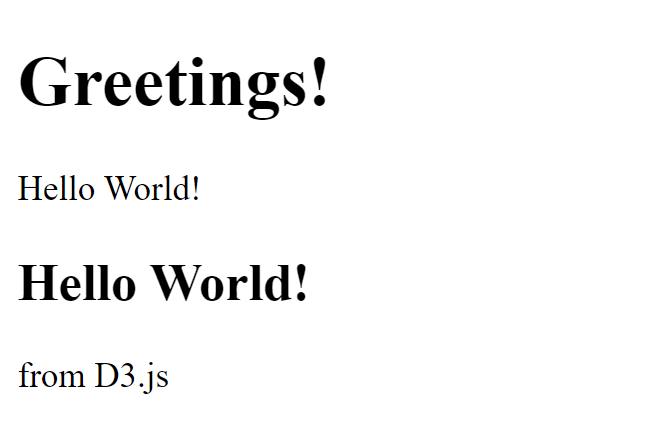


The D3 framework allows you to interact directly with the browser DOM. We appended an <h2> element and a span element inside the div element with the class ‘greetings’. This ability to directly manipulate the DOM is very powerful and is the basis for everything we will do with D3.

Q2. Change the D3 code so that the span that is appended goes after the h2 element we added, not inside it. Put your working D3 code below:

d3.select("div.greetings").append("h2").text("Hello World! ");

d3.select("div.greetings").append("span").text("from D3.js");



Now lets start working with an SVG image. Note that the d3.select selector returns the first matching element, whereas d3.selectAll returns an array of all matched elements.

Simple barchart:

Lets use this example from Mike Bostock at <https://bost.ocks.org/mike/bar/> . Put this in the start of the script section in the body of your html file. You might want to create a new file to manipulate.

var data = [4, 8, 15, 16, 23, 42];

This creates a very simple javascript array that is available to any javascript function after this statement. We should also add a html element to manipulate. We could choose to use D3 to do this for us, but I find it cleaner and clearer to define the html elements in the html file rather than manipulate everything in javascript.

<div class="chart"></div>

Now add this D3 code to the end of the body script section:

d3.select(".chart")

.selectAll("div")

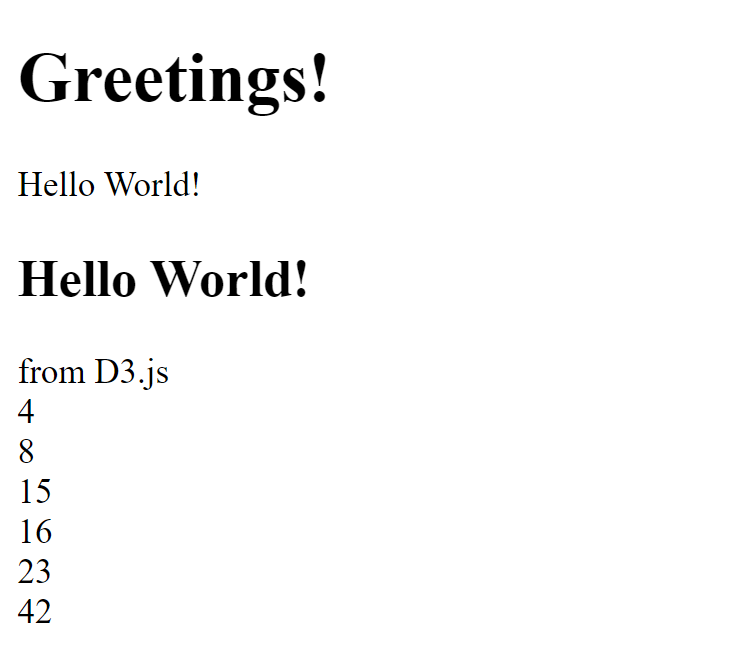
.data(data)

.enter().append("div")

.style("width", function(d) { return d \* 10 + "px"; })

.text(function(d) { return d; });

Q3. Reload the html file in your browser. What do you see?



That might not look like much. Lets go ahead and add a style statement in the <head> section – I put css and style statements before javascript, but that is a matter of preference.

<style>

.chart div {

font: 10px sans-serif;

background-color: steelblue;

text-align: right;

padding: 3px;

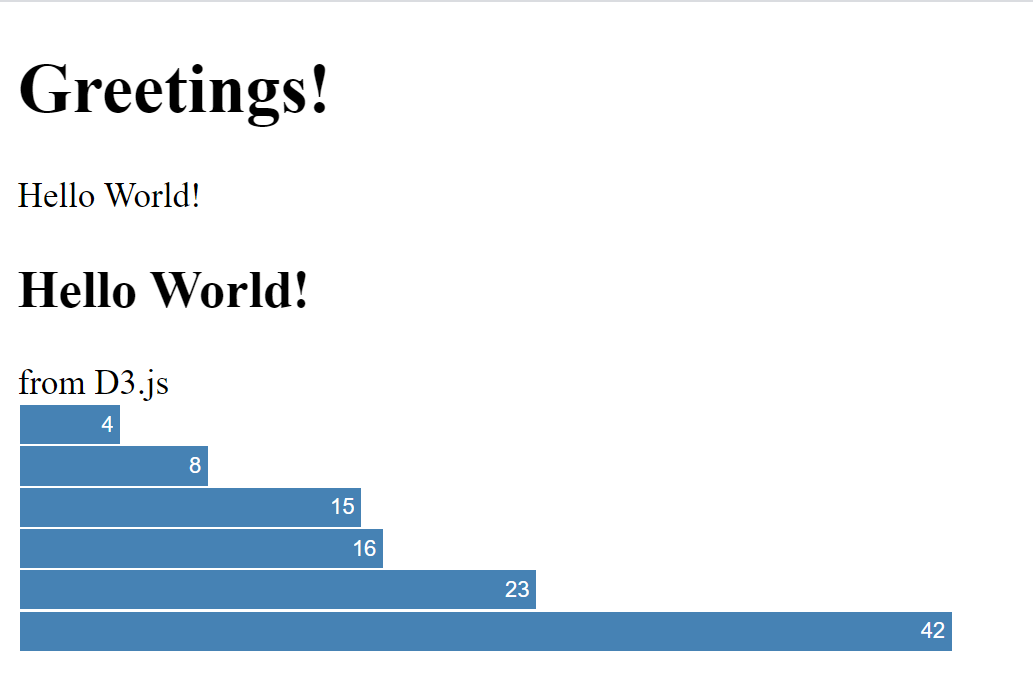
margin: 1px;

color: white;

}

</style>

Q4. Now reload. What do you see?



We now have made our first d3 plot!

However, we have done some coding shortcuts and bad things along the way. Our data is being scaled and manipulated to pixels in the code, and that makes this method very fragile.

Q5. Add a data point of 200. Reload. What happened?



As with ggplot, there are powerful ways to scale plots in D3. We will use one of them. Add this above the plotting code. It defines the scaling function:

var x = d3.scaleLinear()

.domain([0, d3.max(data)])

.range([0, 420]);

Now replace the plotting code with this:

d3.select(".chart")

.selectAll("div")

.data(data)

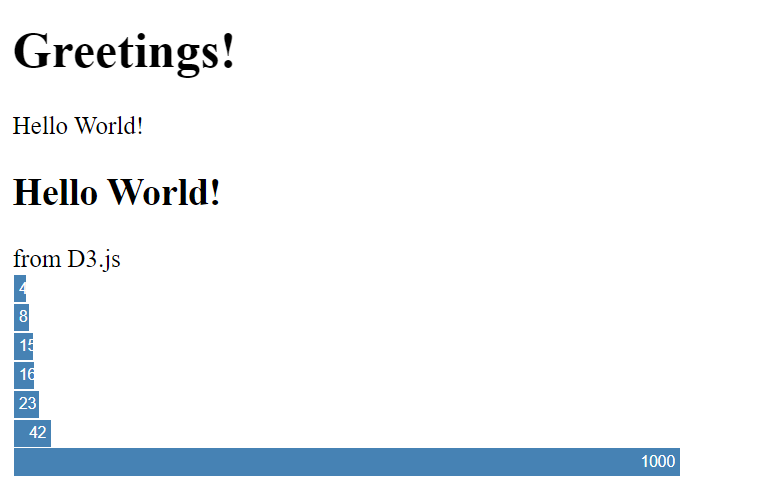
.enter().append("div")

.style("width", function(d) { return x(d) + "px"; })

.text(function(d) { return d; });

There are two very important pieces – domain defines the input range, that is 0 to the maximum value found in the data array, and range defines the size of the resulting plot. If you explore the DOM with your browser’s debugger, you can see that the largest element is now 420px (pixels).

Q6. Change the range to 1000 and reload. Did it work as expected?



Now we will redo the plot. Rather than working with html div objects and a stylesheet, lets manipulate an SVG canvas. The techniques are the same but a little harder to understand. A more detailed tutorial is at <https://bost.ocks.org/mike/bar/2/>. I am going to skip the code to manually manipulate the SVG object and go straight to using D3.

Add to the style element in your html file:

.chart rect {

fill: steelblue;

}

.chart text {

fill: white;

font: 10px sans-serif;

text-anchor: end;

}

Add to your body the new SVG element:

<svg class="chart"></svg>

Lets now define the width and barheight as a javascript variable. I put it immediately after the data variable.

var width = 420,

barHeight = 20;

Now rewrite the scaling code to use the above width variable and replace the existing javascript function:

var x = d3.scaleLinear()

.domain([0, d3.max(data)])

.range([0, width]);

Q7. Plot and generate a pdf and include in your homework.



And now add this to the end of the D3 javascript section:

var chart = d3.select("svg.chart")

.attr("width", width)

.attr("height", barHeight \* data.length);

var bar = chart.selectAll("g")

.data(data)

.enter().append("g")

.attr("transform", function(d, i) { return "translate(0," + i \* barHeight + ")"; });

bar.append("rect")

.attr("width", x)

.attr("height", barHeight - 1);

bar.append("text")

.attr("x", function(d) { return x(d) - 3; })

.attr("y", barHeight / 2)

.attr("dy", ".35em")

.text(function(d) { return d; });

var width = 420,

barHeight = 20;

var x = d3.scale.linear()

.domain([0, d3.max(data)])

.range([0, width]);

var chart = d3.select(".chart")

.attr("width", width)

.attr("height", barHeight \* data.length);

var bar = chart.selectAll("g")

.data(data)

.enter().append("g")

.attr("transform", function(d, i) { return "translate(0," + i \* barHeight + ")"; });

bar.append("rect")

.attr("width", x)

.attr("height", barHeight - 1);

bar.append("text")

.attr("x", function(d) { return x(d) - 3; })

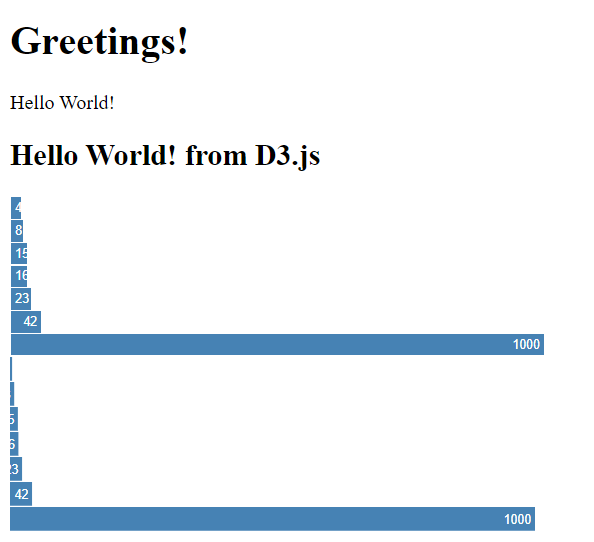
.attr("y", barHeight / 2)

.attr("dy", ".35em")

.text(function(d) { return d; });

There is a lot more here!

Q8. Reload your page. Generate a pdf/screenshot and include in your homework.



A very powerful technique for visualization is a force layout. To see what that looks like in D3, see:

<http://www.coppelia.io/2014/07/an-a-to-z-of-extra-features-for-the-d3-force-layout/>

<https://tomroth.com.au/d3/>

Next we will work with D3 inside of R next. These are handy resources:

<https://rstudio.github.io/r2d3/articles/introduction.html>

<https://rstudio.github.io/r2d3/articles/dependencies.html>

<https://rstudio.github.io/r2d3/articles/learning_d3.html>

<https://rstudio.github.io/r2d3/articles/publishing.html>

Note that you can do all the visualizations in a single notebook and hand in one pdf and one notebook with them all, even though I say include a pdf of the result and R notebook file after every question.

Q9. Create a new R notebook as described here: <https://blog.rstudio.com/2018/10/05/r2d3-r-interface-to-d3-visualizations/> . You will need to install the R2D3 package. Find out in the environment what version of D3 is included in the R2D3 package. Do the simple bar chart demo.

Include a pdf of the result, and the R notebook file as well in your homework submission.

Q10. Embed a D3 Chord layout in the R Notebook. Use the example here: <https://rstudio.github.io/r2d3/articles/gallery/chord/>

Include a pdf of the result, and the R code as well in your homework submission.

Q11. Embed a D3 force layout in the R Notebook. Use the NetworkD3 R package: <http://christophergandrud.github.io/networkD3/>

Include a pdf of the result, and the R code as well in your homework submission.

Q12. Embed a D3 Heatmap in an R notebook file. See <https://blog.rstudio.com/2015/06/24/d3heatmap/> for instructions.

Include a pdf of the result, and the R code as well in your homework submission.

Q13. Embed a D3 Bubblechart in an R notebook file. See <https://blog.rstudio.com/2015/06/24/d3heatmap/> for instructions. You can use the following for your flare.csv but you are welcome to find your own data too.

id,value

flare,

flare.analytics,

flare.analytics.cluster,

flare.analytics.cluster.AgglomerativeCluster,1

flare.analytics.cluster.CommunityStructure,2

flare.analytics.cluster.HierarchicalCluster,1

flare.analytics.cluster.MergeEdge,1

flare.analytics.graph,

flare.analytics.graph.BetweennessCentrality,1

flare.analytics.graph.LinkDistance,1

flare.analytics.graph.MaxFlowMinCut,1

flare.analytics.graph.ShortestPaths,1

flare.analytics.graph.SpanningTree,1

flare.analytics.optimization,

flare.analytics.optimization.AspectRatioBanker,1

Include a pdf of the result, and the R code as well in your homework submission.

Q14. Pick one of the plot types above, and rather than hard code the data or load the data from a file, pass the data in from an R variable. You will need to format the data appropriately for the plot type you choose, and then find out how to pass the data into R2D3.

Include a pdf of the result, and the R code as well in your homework submission.

Q15. Choose a visualization from <https://rstudio.github.io/r2d3/articles/gallery.html> that we have not yet done. Build it in your R notebook file.

Include a pdf of the result, and the R code as well in your homework submission.

Now that we have gotten a little bit of familiarity with D3 and even working inside R Studio, I would like to switch to chord diagrams. I will walk through a couple of D3 examples and then a cytoscape.js version.

A few great examples are at <http://dataviscourse.net/tutorials/lectures/lecture-d3-layouts/> <http://bl.ocks.org/dukevis/4374072a30331316a1d3> the Chord Layout section of <http://dataviscourse.net/tutorials/lectures/lecture-d3-layouts/> and Nadieh Bremer’s two great examples using cell phone plan switching data: <http://bl.ocks.org/nbremer/6540350> and <http://bl.ocks.org/nbremer/94db779237655907b907> The last link has a really great example of how to build an interactive presentation through D3.

Now lets switch to use the Cytoscape.js framework. It is more streamlined and does force-directed and chord graphs with less overhead. Use the tutorial <https://blog.js.cytoscape.org/2016/05/24/getting-started/> to make your first graph in cytoscape. Use the example in the tutorial to add 15 more nodes and edges with a for loop. Don’t forget to ask cytoscape to recalculate the layout. That code can look like this:

cy.layout({

name: 'circle'

}).run();

Q16. Include a pdf of the result, and the html file as well in your homework submission.

Now lets create a chord diagram directly using Cytoscape.js See the example code at <http://bl.ocks.org/dukevis/4374072a30331316a1d3> Go ahead and reproduce this graph on your own, but use a local d3.js file rather than loading remotely. Note that the data is all hard coded in javascript.

Q17. Include a pdf of the result, and the html file as well in your homework submission.

Now replace the hardcoded data with either loading from a file or loading from a URL.

Q18. Include a pdf of the result, and the code file as well in your homework submission.

We haven’t fully explored what D3.js and Cytoscape.js are capable of, but we have started!