Lab 2: Dynamic and Interactive

Data Visualisation and Analytics F20DV/F21DV

The labs enable you to demonstrate a variety of skills, such as, problem solving, communication skills, time-management and so on. The labs are structured to encourage a mix of guided and active learning, innovative development, critical thinking and knowledge-based expertise of the subject.

Each lab requires you to discuss/explain and justify your implementation details (also extra check to verify that you understand concepts/reasons/limitations). The lab also offers you the chance to discuss or ask any questions while getting guided feedback on your progress/lab work.

1. Each lab must be demonstrated during lab session (multiple sessions/opportunities each week)
2. Report and code must be submitted on CANVAS (evidence) for each lab (after the demonstration to a lab helper/staff)
3. After demonstrating your work (you have until the end of the week to submit a report and code - single zip as evidence/check on CANVAS)

Note – If you fail to demonstrated/discussed/explained your work then you get 0% (even if a report is submitted). The report is a 'check' while the demonstration/discussion formulates the mark. If circumstances prevent you from demonstrating or attending the lab session, please inform the lecturer. Important, there are multiple sessions that give you an opportunity to demonstrate/complete the task early (don't leave it until the last minute).

Each lab/code must be to the highest standard (e.g., commented code, naming convention, structured, organized, tested, ...).

This lab is designed to focus on dynamic and interactive visualization concepts. Only a small subset of the operations are described in this lab but should give you an idea of the potential capabilities of the library/tools. The lab includes a number of exercises that you can work through to help you get started. You're encouraged to work through all the exercises and modify programs to perform similar tasks (experiment). You should work on this lab in your own time, not just the scheduled assessed lab sessions. During the lab sessions, once you have completed the exercises on this sheet, show it a lab helper who will discuss and mark your work. **This is worth 25% of the marks for this course**. Check the timetable for the scheduled sessions for this lab.

Resources:

- F20/21DV Material (https://f21dv.github.io/)
- D3 Homepage/Documentation (https://d3js.org/)

Overview, by the end of this **lab 2**, the students should have

- Interactive visualization
- Animations and transition effects
- Dynamically update/change data and visualizations
- Generating interactive graphs/graphics

FAQ

1. How should I manage my work/exercises?

As you work through the exercises in each lab you should keep a record of your progress (e.g., separate files, comments in code, name/date at the top of each file, …). You should manage your files/folders and logs.

2. What should I include in my report? What format?

Your report should be formatted to a professional standard which documents your progress/exercise results (e.g., lab report structure that detailing each exercise/results and reason/reflection). At the top of the report on the first page, you should include your name, which lab, the date and who you demonstrated your work to during the lab session

3. Do the exercises have to be completed in the order they're given?

Each exercise increases in complexity and builds upon previous tasks (hence, you need to work through them in order)

4. The answer isn't in the lab worksheet?

In addition to the lab notes, you'll also have to refer to the lectures, recommended resources/documentation/support material (i.e., not just copy and repeating each lab exercise – you also have to demonstrate active learning, understanding, problem solving and original thinking). Some exercises may require you to demonstrate basic knowledge and problem solving while other exercises more critical thinking/analysis of the task

5. Do I only get one chance to demonstrate my lab work?

There are multiple lab sessions allocated to each lab (multiple weeks), so you have several chances to complete and demonstrate your work (i.e., not a single deadline). You can demonstrate and complete the lab exercise early (don't leave it until the very end). Recommended that you start early and this way if there are any issues or problems you'll have time to address these and resolve them for your final demonstration/submission.

6. Which version of D3 do I need to use?

Your lab exercises should run using v7+ of d3.js

7. When will I get feedback/mark for my lab coursework?

After the deadline has passed, your preliminary mark will be released with your submitted report on CANVAS (i.e., you need to have demonstrated/discussed your work during the lab session).

8. How many marks are each lab worth?

25% each.

9. How many lab courseworks are there?

There are 4 lab courseworks.

Assessment Breakdown

Task

The assessed lab courseworks consist of understanding, designing, developing, implementing and testing visualization scenarios. The requirements for the courseworks are:

1. **You should attend and contribute** to the labs.
2. You need to conceptualise, develop, and test a **prototype of your visualizations**.
3. You need to plan, implement and debug **prototype of your visualizations**.
4. You need to incorporate these tested visualizations into designs and demonstrate/discuss/**presented them**.

Each assessed labs is worth 25% of your overall course mark. The concepts necessary to complete the assessment are available and covered during the lectures/support material/recommended texts.

This lab coursework will be marked as following:

Sections	Description	Marks
Exercises/Visualization Prototypes	In this section your ability to complete/develop/extend exercises/prototype will be assessed.	18
Communication, Analysis, Presentation Visualization	Marks will be awarded for demonstrating creative thinking, feedback and concept used. Show understanding of the material and the appropriateness and effectiveness of associated visualisation and analysis techniques	5
Management and documentation	A couple of mark will be awarded for management, documentation and organisation of the tasks/code/exercises	2
	Total Marks F2xDV Assessment	**25**

Marking Details

Below are the marking guidelines for the lab courseworks.
There are allocated sessions for demonstrating your exercises in the timetable (lab sessions).
The demonstration sessions are run over multiple weeks, you should organise and prepare for your demonstration (i.e., organised and able to show your work/exercises/answers in addition to being able to answer questions, e.g., results, concepts used, modifications).
Note: You will need to both demonstrate your work during one of the assigned lab sessions; after this, you also need to submit a report and your code (single file zipped together) on CANVAS (evidence).
Late submissions will be subject to the normal penalties as defined in the late coursework policy. If you have any questions or queries about the assessment, please do not hesitate to contact B.kenwright@.ac.uk (Edinburgh) or r.soobhany@hw.ac.uk (Dubai)
Feedback and your marks: you will receive feedback for this coursework during the demonstration/discussion.

Late Submissions

The University recognises that, on occasion, students may be unable to submit coursework on the submission date or be unable to present their work on the submission date. In these cases, the University's Submission of Coursework Policy outlines are:
• No individual extensions are permitted under any circumstances.
• Standard 30% deduction from the mark awarded (maximum of five working days).
• In the case where a student submits coursework up to five working days late, and the student has valid mitigating circumstances, the mitigating circumstances policy will apply, and appropriate mitigation will be applied.
• Any coursework submitted after five calendar days of the set submission date shall be automatically awarded a no grade with no formative feedback provided.
Please contact your Personal Tutor or Counsellor if you are unable to meet the deadlines or need information for Mitigating Circumstances or Temporal Suspensions of Studies.

Part 1. CSS Effects/Animations

The D3 library works seamlessly with the CSS standard; which enables you to easily take advantage of CSS interactive/animation features.

For instance, the CSS keyframes example below shows a simple ‘svg’ box which pulses (animate smaller and larger) when the mouse cursor moves over the item.

```
<style>
.pulse {
  border: 1px solid blue;
  fill: lightblue;
  stroke: purple;
  -webkit-transform: scale(1);
  -webkit-transform-origin: 50% 50%;
  transform: scale(1);
  transform-origin: 50% 50%;
}

.pulse:hover
{
  -webkit-animation-name: pulsar;
  -webkit-animation-duration: 0.2s;
  -webkit-animation-iteration-count: infinite;
  -webkit-animation-direction: alternate;
  animation-name: pulsar;
  animation-duration: 0.2s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
  -webkit-transform-origin: 50% 50%;
  transform-origin: 50% 50%;
}

@keyframes pulsar {
  from {
      fill: red;
  }
  to {
      fill: red;
      transform: scale(1.3,1.3);
      transform-origin: 50% 50%;
  }
}
</style>

<svg class='pulse' width='30' height='30'></svg>
```

See here for details/further examples on CSS keyframes control styles [LINK].

Exercise 1: Using the keyframes animation concept from the example above, write a simple D3 program that draws a ‘line-graph’ For each of the points on the graph, draw a small ‘svg’ circle. Set an animated keyframe style on each graph point, so when the mouse cursor moves over the point, it ‘pulses’.

There are a range of transition effects within CSS which can easily be used by your visualizations (e.g., D3 style(..) method). For instance, you could also use the CSS styles for 'tooltips' or extra visual information.

The simple example below shows the principle for adding some display text when the mouse cursor moves over some text.

```
<style>
div {
  position: relative;
  background-color: yellow;
  padding: 20px;
  display: none;
}

span:hover + div {
  position: absolute;
  display: inline-block;
}
</style>

<span>Hover over me!</span>
<div>I will show on hover</div>
```

Exercise 2: Create a webpage using D3 (adds items dynamically), then set the styles for the items so they use CSS to display extra information when the mouse cursor moves over them.

CSS transitions are an easy way to 'smooth' changes (e.g., instead of items popping or jumping around they can be interpolated using CSS styles).

See online documentation for details/examples of CSS transitions [LINK].

Part 2. Events

D3 supports built-in events and custom events. You can bind an event listener to any DOM element using "d3.selection.on()" method. You use events to capture actions like click, mouseover and mouseout.

Following example demonstrates handling of mouseover and mouseout events:

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<script>
   d3.select('body')
     .append('div')
     .style('width', '100px')
     .style('height', '20px')
     .style('background-color', 'green');

   d3.selectAll("div")
     .on("mouseover", function(event){
        d3.select(this)
          .style("background-color", "orange");
        // Get current event info
        // Note: d3.event  (event) passed as the first argument to all listeners
        console.log(event);

        // Get x & y co-ordinates
        // Note: d3.mouse was removed in d3v6, you should use d3.pointer(event)
        console.log(d3.pointer(event));
    })
    .on("mouseout", function(){
        d3.select(this)
          .style("background-color", "steelblue")
    });
</script>
```

Exercise 3: Modify the above example so in addition to the color changing, other styles change (e.g., size and border styles).

Exercise 4: Based on the example above, instead of a 'div' element, use an 'svg' container and add a 'circle' svg. When the mouse moves over the svg circle, change the radius (e.g., larger when the mouse moves over and back to the default size when the mouse moves out)

Exercise 5: The "d3.pointer" method lets you get the mouse position. When the mouse moves over the 'svg' container, add a 'text' svg element at the location of the mouse position. As the mouse cursor moves around the svg container, have the text move to the cursor position (i.e., text follows the mouse cursor).

Part 3. D3 Transitions

Transitions let you animate change. Transitions are made on DOM selections using “<selection>.transition()” method.

The main transition methods are:

selection.transition()	this schedules a transition for the selected elements
transition.duration()	duration specifies the animation duration in milliseconds for each element
transition.ease()	ease specifies the easing function, example: linear, elastic, bounce
transition.delay()	delay specifies the delay in animation in milliseconds for each element

The following example demonstrates changing the background color of a div from blue to red with animation.

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<script>
d3.select('body')
  .append("div")
  .style('width',  '100px')
  .style('height', '100px')
  .style('background-color', 'blue')
  .transition()
  .duration(1000)
  .style("background-color", "red");
</script>
```

Note: try modifying the example so the width/height of the div doesn’t include the units ‘px’ – what happens?

Exercise 6: Chain an extra transition onto the example above so that after the ‘div’ element animates to ‘red’ it then continues to transition to ‘green’ over 2 seconds.

Exercise 7: In addition to transitioning the color, also change the ‘size’ of the div element so it transitions smaller then transitions larger (i.e., width and height style values).

Exercise 8: Combining the ‘transition’ method with the mouse over ‘event’. So that the ‘div’ transitions color only when the mouse moves over and back to the original color when moves away from the element.

You use the “transition.ease” method to control the type of transition (e.g., linear or non-linear).
See the following [link] for a visual comparison of the different transition easing types.

The following example adds the ‘bounce’ easing type, so the ‘div’ shrinks but appears to ‘bounce’ during the transition.

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<script>
d3.select('body')
  .append("div")
  .style('width',  '100px')
  .style('height', '100px')
  .style('background-color', 'blue')
  .style('transform', 'scale(1.0)')
  .transition()
  .ease( d3.easeBounce )
  .duration(1000)
  .style("background-color", "red")
  .style('transform', 'scale(0.5)')
</script>
```

Exercise 9: Add two additional ‘div’ elements and perform the same ‘transitions’ but with different ‘easing’ methods (see the elements transition at the same time but with different easing motions).

Exercise 10: Add the easing effect to the svg circle example, so when a mouse moves over a ‘circle’ svg element, it grows by a small amount, when the mouse moves away (out of focus) the svg circle returns to the original size (use the ease ‘bounce’).

Exercise 11: Add a ‘text’ svg element, when the mouse moves over the text, the size of the text changes color and increases in size (when the mouse moves out/away the text goes back to the original color/size).

The “transition.delay” lets you add a ‘delay’ to the transition effect (i.e., when you want the transition to start).

The example below will animate two bars on their height with a linear ease. At the start, the first bar increases its height; however, after a small duration (delay), the second bar will start to animate in same way.

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<script>
  var svg = d3.select("body")
    .append("svg")
    .attr("width", 500)
    .attr("height", 500);


  var bar1 = svg.append("rect")
    .attr("fill", "blue")
    .attr("x", 100)
    .attr("y", 20)
    .attr("height", 20)
    .attr("width", 10)

  var bar2 = svg.append("rect")
    .attr("fill", "blue")
    .attr("x", 120)
    .attr("y", 20)
    .attr("height", 20)
    .attr("width", 10)

  update();

function update() {
  bar1.transition()
    .ease(d3.easeLinear)
```

```
      .duration(2000)
      .attr("height",100)

   bar2.transition()
      .ease(d3.easeLinear)
      .duration(2000)
      .delay(2000)
      .attr("height",100)
}
</script>
```

Exercise 12: Add a third bar to the example above which starts to animate after 4 seconds.

Exercise 13: Chain an additional ‘transition’ effect to the above example, so that after completing the transition, the bars ‘transition’ back to their original height (grow then shrink).

Exercise 14: Modify the transition effect so that the color also changes for the example above (e.g., blue to red)

Part 4. Animated Chart

Taking previous sections and combining them to create a complete interactive chart example.

Given the following csv data:

```
year,value
2011,45
2012,47
2013,52
2014,70
2015,75
2016,78
```

If you type in the following example, it should create a bar chart, however, instead of 'popping' in instantly – the bar graph animates in.

```
<style>
.bar {
   fill: steelblue;
}

.highlight {
   fill: orange;
}
</style>

<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<svg width="600" height="500"></svg>

<script>
var svg = d3.select("svg");
var margin = 200;
var width  = svg.attr("width") - margin;
var height = svg.attr("height") - margin;

svg.append("text")
  .attr("transform", "translate(100,0)")
  .attr("x", 50)
  .attr("y", 50)
  .attr("font-size", "24px")
  .text("Stock Price")

var x = d3.scaleBand().range([0, width]).padding(0.4);
var y = d3.scaleLinear().range([height, 0]);

var g = svg.append("g")
        .attr("transform", "translate(" + 100 + "," + 100 + ")");

d3.csv( csvfile.csv ).then(function(data) {

      x.domain( data.map(function(d) { return d.year; }) );
```

```
        y.domain([0, d3.max(data, function(d) { return d.value; })]);

        g.append("g")
         .attr("transform", "translate(0," + height + ")")
         .call(d3.axisBottom(x))
         .append("text")
         .attr("y", height - 250)
         .attr("x", width - 100)
         .attr("text-anchor", "end")
         .attr("stroke", "black")
         .text("Year");

        g.append("g")
         .call(d3.axisLeft(y).tickFormat(function(d){
             return "$" + d;
         }).ticks(10))
         .append("text")
         .attr("transform", "rotate(-90)")
         .attr("y", 6)
         .attr("dy", "-5.1em")
         .attr("text-anchor", "end")
         .attr("stroke", "black")
         .text("Stock Price");

        g.selectAll(".bar")
         .data(data)
         .enter().append("rect")
         .attr("class", "bar")
     //    .on(..... ) – call mouse events here…
         .attr("x", function(d) { return x(d.year); })
         .attr("y", function(d) { return y(d.value); })
         .attr("width", x.bandwidth())
         .transition()
         .ease(d3.easeLinear)
         .duration(400)
         .delay(function (d, i) {
             return i * 50;
         })
         .attr("height", function(d) { return height - y(d.value); });
});
```

Exercise 15: Take the above example, and add in the following mouse events below:

```
/*
e.g.,        .on("mouseover", onMouseOver) //Add listener for the mouseover event
             .on("mouseout", onMouseOut)   //Add listener for the mouseout event
*/

//mouseover event handler function
function onMouseOver(d, i) {

        d3.select(this).attr('class', 'highlight');
        d3.select(this)
```

```
          .transition()     // adds animation
          .duration(400)
          .attr('width', x.bandwidth() + 5)
          .attr("y", function(d) { return y(d.value) - 10; })
          .attr("height", function(d) { return height - y(d.value) + 10; });

        g.append("text")
         .attr('class', 'val')
         .attr('x', function() {
            return x(d.year);
         })
         .attr('y', function() {
            return y(d.value) - 15;
         })
         .text( function(d) { return '$' + i.value; } ); // Value of the text
}

//mouseout event handler function
function onMouseOut(d, i) {
        // use the text label class to remove label on mouseout
        d3.select(this).attr('class', 'bar');
        d3.select(this)
          .transition()     // adds animation
          .duration(400)
          .attr('width', x.bandwidth())
          .attr("y", function(d) { return y(i.value); })
          .attr("height", function(d) { return height - y(i.value); });

        d3.selectAll('.val')
          .remove()
}
```

Exercise 16: Modify the example above so the popup text that is displayed when the mouse cursor moves over each bar is positioned 'above' the bar instead of the top left.

Exercise 17: Modify the bar chart example so each bar has a color that represents the value (e.g., red maximum and blue the minimum range)

Part 5. Changing Data and Transitioning

This section is about managing changing data and the visual transition.

The example below creates a simple ‘bar chart’, however, you can select different data to be visualized (e.g., data set 1 or 2). Through ‘transitions’, you’re able to create a visual animation that smoothly interpolates between the data sets.

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<!-- Add buttons -->
<button onclick="update(data1)">Variable 1</button>
<button onclick="update(data2)">Variable 2</button>

<script>
// create 2 data_set
const data1 = [
  {group: "A", value: 5},
  {group: "B", value: 20},
  {group: "C", value: 9}
];

const data2 = [
  {group: "A", value: 10},
  {group: "B", value: 2},
  {group: "C", value: 22}
];

// set the dimensions and margins of the graph
const margin = {top: 30, right: 30, bottom: 70, left: 60};
const width  = 460 - margin.left - margin.right;
const height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select('body')
          .append('div')
  .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");

// X axis
var x = d3.scaleBand()
  .range([ 0, width ])
  .domain(data1.map(function(d) { return d.group; }))
  .padding(0.2);
svg.append("g")
  .attr("transform", "translate(0," + height + ")")
  .call(d3.axisBottom(x))

// Add Y axis
var y = d3.scaleLinear()
```

```
  .domain([0, 20])
  .range([ height, 0]);
svg.append("g")
  .attr("class", "myYaxis")
  .call(d3.axisLeft(y));

// A function that create / update the plot for a given variable:
function update(data) {

  var u = svg.selectAll("rect")
                  .data(data)

  u .enter()
     .append("rect")
     .merge(u)
     .transition()
     .duration(1000)
      .attr("x", function(d) { return x(d.group); })
      .attr("y", function(d) { return y(d.value); })
      .attr("width", x.bandwidth())
      .attr("height", function(d) { return height - y(d.value); })
      .attr("fill", "#69b3a2")
}

// Initialize the plot with the first dataset
update(data1)

</script>
```

Exercise 18: Modify the example so that it has a 3rd data set (e.g., add extra button and an extra test data set to the top of the file)

Exercise 19: Modify the example so that each data set is displayed in a different color (rectangle bars are drawn in a different color)

Exercise 20: Add an additional 'event' so that when the 'mouse' moves over each bar it displays a text svg above the bar with the 'value' of the data.

Exercise 21: Add an axis to the top and to the right of the bar chart (also displays an axis along the top and along the right of the chart)

Exercise 22: Update the example so that the data sets can be different sizes (e.g., [A,B,C] but the second data set could have [A,B,C,D]). You'll need to update the 'axis' details (range) and extra details to 'remove' (fade away unused elements using 'exit').

Exercise 23: Extend the concept of the 'bar chart', however, instead draw a 'line graph' with two data sets that 'transition' between them when you select a button (as you did for the bar graph example above).

Part 6. Pie Chart

The “Interpolate()” function in D3 is used to interpolate the two given values. These values can be simple numbers, colors or arrays of data.

For instance, you can ‘interpolate’ two arrays of data, as shown below:

```
<script src="//d3js.org/d3.v7.min.js"></script>

<script>
let intr = d3.interpolate( [20, 40, 4], [1, 12, 10])

console.log("Type of returned function is: ", typeof (intr) );

console.log( intr(0.2) )
</script>
```

Exercise 24: What is the output for the example above (and why)?

The following example, interpolates two color values:

```
let cc = d3.interpolate('red', 'green')
console.log( cc(0.5)  );
```

Exercise 25: What is the interpolated color value for the above code (and why)?

Exercise 26: How would you interpolate a ‘date’ using D3? (show an example in code)

The following example shows a ‘pie chart’ that uses the ‘interpolate’ to spin in the pie chart segments:

```
<script src="//d3js.org/d3.v7.min.js"></script>

<script>
var dataset = {
  apples: [5345, 2879, 1997, 2437, 4045],
};

var width = 460,
   height = 300,
   radius = Math.min(width, height) / 2;

var color = d3.scaleOrdinal().range(d3.schemeSet3);

var pie = d3.pie()
   .sort(null);

var arc = d3.arc()
   .innerRadius(radius - 100)
   .outerRadius(radius - 50);

var svg = d3.select("body").append("svg")
   .attr("width", width)
```

```
    .attr("height", height)
    .append("g")
    .attr("transform", "translate(" + width / 2 + "," + height / 2 + ")");

var path = svg.selectAll("path")
    .data(pie(dataset.apples))
  .enter().append("path")
    .attr("fill", function(d, i) { return color(i); })
    .attr("d", arc)
    .transition()
        .duration(1000)
        .attrTween("d", function (d) {
          var i = d3.interpolate(d.endAngle, d.startAngle);
          return function (t) {
            d.startAngle = i(t);
            return arc(d);
          }
        });
</script>
```

Exercise 27: Add an additional data set and modify to the code so the 'pie chart' transitions (animated) between the currently displayed data set and your new test data set (e.g., click button the pie chart segments animate to new data values). Note, all the segments should not 'pop' out/in but should transition smoothly.

Part 7. D3 Force layout

D3's force layout uses a physics based simulator for positioning visual elements (dynamic).

The D3 force layout uses the 'd3.forceSimulation()' method, which manages the forces and the update. After the forces/simulation has run the visual update is called via an Event callback ('tick') to update what's displayed on screen.

The example below shows a simple D3 force example, which shows a random set of spheres all 'pulled' by forces towards the centre (in addition to forces/collisions that stops them overlapping).

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>
<script>
var width = 400, height = 400;

// setup svg
d3.select('body').append('svg').attr('width',width).attr('height',height);

// generate some random data
var numNodes = 100;
var nodes = d3.range(numNodes).map(function(d) {
  return {radius: Math.random() * 25}
})

var simulation = d3.forceSimulation(nodes)
      .force('charge', d3.forceManyBody().strength(5))
      .force('center', d3.forceCenter(width / 2, height / 2))
      .force('collision', d3.forceCollide().radius(function(d) {
          return d.radius
      }))
      .on('tick', ticked);

function ticked() {
  var u = d3.select('svg')
    .selectAll('circle')
    .data(nodes)
    .join('circle')
    .attr('fill', 'blue')
    .attr('r', function(d) {
      return d.radius
    })
    .attr('cx', function(d) {
      return d.x
    })
    .attr('cy', function(d) {
      return d.y
    })
}
console.log('ready..');
</script>
```

Refer to the D3 documentation for details on the D3 force layout methods/syntax [LINK].

Exercise 28: For the example above, modify the code so that each sphere is displayed as a different color.

Exercise 29: For the example above, instead of the data getting generated randomly, modify the code so the data is defined either an inline array or from an external source (e.g., csv/json).

Exercise 30: Update the example, so each of the spheres displays extra information when the mouse cursor moves over them (e.g., popup text)

Exercise 31: Modify the example, so the spheres change color when the mouse moves over them

Exercise 32: Using the D3 documentation as a reference, extend the example to include additional ‘forces’ to make the visualization more interesting and interactive.