

## STATS 401 Lab 8 Tutorial: D3.js

### Lab Goal


This week, we will continue to work on the basic D3.js visualization project. After creating different shapes SVG shapes, we will learn how to do:

- Path
- Area Chart
- Advanced Shapes using Groups
- Axes

### Code Instructions and Steps

#### Task 1: Path

If you look at the commonly used SVG shapes we mentioned last week; we only have the path shape left. So let's try to create it then! It's more complex, and you will see it soon. We will only use the parameter "d" and a generator for creating paths.



Shape	Function	Mandatory Attributes	Optional Attributes
rect	.attr()	x, y, width, height	rx, ry, class, transform
circle	.attr()	cx, cy, r	class, transform
ellipse	.attr()	cx, cy, rx,ry	class, transform
line	.attr()	x1, y1, x2, y2	class, transform
text	.text() .attr()	x, y	dx, dy, class, transform, text-anchor
path	.attr()	d	class, transform, pathLength

**You can directly use the main.css, d3.v7.js, index.html files, and the CSS folder you created in Lab7 for this lab.**

Change the title of the html to

```
<title>Lab 8</title>
```

Create a new file, and name it interpolate.js. Go back to the .html file, and remove:

```
<script type="text/javascript", src = "index.js"> </script>
```

Then, add:

```
<script type="text/javascript", src = "interpolate.js">  
</script>
```

You will get a blank new page when you refresh the web browser. In the interpolate.js file, add a new data array that stores 6 points with x and y coordinates in a different way:

```
var dataArray = [{x:,y:},{x:,y:},{x:,y:},{x:,y:},{x:,y:},{x:,y:}];
```

Assign random x and y values but have increasing x values for generating a complex path.

```
var dataArray = [{x:4,y:10},{x:7,y:2},{x:9,y:3},{x:12,y:9},{x:17,y:6},{x:20,y:20}];
```

Create a new SVG variable, use select() and append() methods to create variables, and assign attributes. Then assign attributes.

```
var svg = d3.select('body')
    .append('svg')
    .attr('height','100%')
    .attr('width','100%');
```

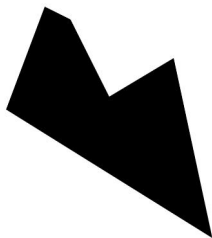
Refresh the page and see what happens? Nothing? Why? Because we didn't assign any dots and line values to this path. Let's use a generator to create lines and connect dots.

```
var line = d3.line()
    .x(function(d,i){return d.x*10})
    .y(function(d,i){return d.y*10})
```

Next, call append for SVG, find the path, and then assign the dots to connect the lines for this shape.

```
svg.append('path').attr('d',line(dataArray))
```

Refresh your page; now, you will see the irregular shape below!



Go to the browser inspector and find the path element. See what's inside! Also, try to change the times value from 10 to 20 for either x/y and see what happens.

Fill and color the shape by adding attributes like the below:

```
svg.append('path').attr('d',line(dataArray))
    .attr('fill','none').attr('stroke','blue');
```

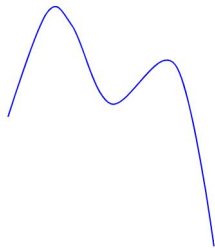
And you will get below shape. Play around with the fill, stroke, and stroke-width attributes' values and see what kinds of shapes you will draw!



Now that we have straight paths, then how do we paint curved lines? In the place where we declare line attributes, add a new attribute:

```
var line = d3.line()  
  .x(function(d,i){return d.x*10})  
  .y(function(d,i){return d.y*10})  
  .curve(d3.curveCardinal);
```

Take a look at the shape now! Also, change it to other values, e.g., d3.curveCardinal, etc., and see what it will look like now!



## Task 2: Area Chart

Create a new file, and name it area.js script.

Create a new html file named task2.html, copy the code below to the file:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-  
scale=1.0">  
  <meta http-equiv="X-UA-Compatible" content="ie=edge">  
  <link rel="stylesheet" href="CSS/main.css">  
  <script type="text/javascript", src = "d3.v7.js"> </script>  
  <title>Lab 8</title>  
</head>  
<body>  
  <nav>  
  </nav>  
  <script type="text/javascript", src = "area.js"> </script>  
</body>  
</html>
```

In the new area.js file, create a new array of 12 values (for the y-axis) and the second array for 12 months (x-axis) to composite this area chart.

```
var dataArray = [14,35,56,67,99,122,150,160,233,400,322,360];  
var dataMonths = [1,2,3,4,5,6,7,8,9,10,11,12];
```

Then, let's create another two variables for the height and width of this chart.

```
var height = 400;  
var width = 1000;
```

Let's create a generator for the area! Still remember how we did it with the path in task1?

```
var area = d3.area()
    .x(function(d,i){return i*20+50;})
    .y0(height)
    .y1(function(d){return height-d;});
```

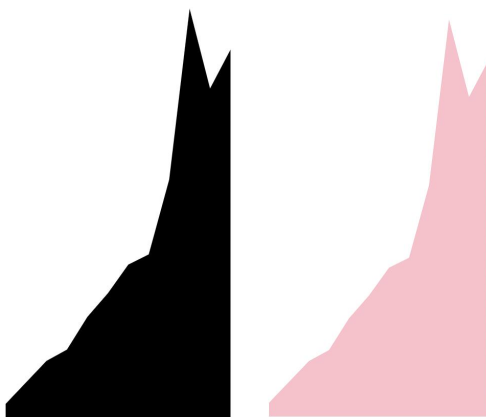
Think about what will be the x, y0, and y1 positions and why we assign (height-d) to y1 and height to y0 position? Next, let's append this area to our page:

```
var svg = d3.select('body')
    .append('svg')
    .attr('height', '100%')
    .attr('width', '100%')
```

And then let's use the generator to create the area!

```
svg.append('path')
    .attr('d', area(dataArray))
```

Now, refresh your page and see what will show up? Do you get below area chart?



Assign a new fill color for this shape by adding “.attr('fill', 'pink')” after the svg.append() line of code. See if your area chart turns pink?

Now, play around with the dataArray variables' values and see how the area chart will change.

### Task 3: Advanced Shapes Using Groups

Create a new file, and name it area2.js script.

Create a new html file named task3.html, copy the code below to the file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet" href="CSS/main.css">
  <script type="text/javascript", src = "d3.v7.js"> </script>
  <title>Lab 8</title>
</head>
```

```

<body>
  <nav>
  </nav>
  <script type="text/javascript", src = "area2.js"> </script>
</body>
</html>

```

Copy the variable from area.js to area2.js:

```

var dataArray = [14,35,56,67,99,122,150,160,233,400,322,360];
var dataMonths = [1,2,3,4,5,6,7,8,9,10,11,12];

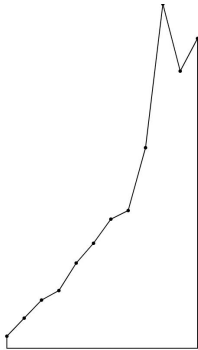
var height = 400;
var width = 1000;

var area = d3.area()
  .x(function(d,i){return i*20+50;})
  .y0(height)
  .y1(function(d){return height-d;});

var svg = d3.select('body')
  .append('svg')
  .attr('height','100%')
  .attr('width','100%')

```

Next, let's add on the data points by showing small dots.



What if we want to have five or six similar area charts? Let's use a loop to do so and continue and create groups of advanced shapes. Create a variable of group named grp, and replace the svg with grp. See below:

```

var grp = svg.append('g')
  .attr('transform','translate(0,0)');

grp.append('path')
  .attr('fill','none')
  .attr('stroke','black')
  .attr('stroke-width',1)
  .attr('d',area(dataArray));

grp.selectAll('circle')
  .data(dataArray)
  .enter().append('circle')
  .attr('cx',function(d,i){return i*20;})
  .attr('cy',function(d){return height-d;})
  .attr('r','2');

```

Refresh your page and see what will happen? Are there many charts now? If you only find one, then go to the page and inspect the grp elements, which contains all dots and path there. Next, let's add a new array named crv to assign the curve styles.

```
var crvTypes = [d3.curveCardinal,d3.curveBasis,d3.curveLinear,d3.curveStep,d3.curveNatural,d3.curveBundle];
var crvTypes =
[d3.curveCardinal,d3.curveBasis,d3.curveLinear,d3.curveStep,d3.
curveNatural,d3.curveBundle];
```

And create a loop to go over the values and put the rest of our prg codes into this loop:

```
for (var t = 0; t<5;t++){

    var area = d3.area()
        .x(function(d,i){return i*20+50;})
        .y0(height) //lowest y coordinate we are using
        .y1(function(d){return height-d;})
        .curve(crvTypes[t]);

    var svg = d3.select('body')
        .append('svg')
        .attr('height','100%')
        .attr('width','100%')

    var grp = svg.append('g')
        .attr('transform','translate(0,0)');

    grp.append('path')
        .attr('fill','none')
        .attr('stroke','black')
        .attr('stroke-width',1)
        .attr('d',area(dataArray));

    grp.selectAll('circle.grpcircle'+t)
        .data(dataArray)
        .enter().append('circle')
        .attr('cx',function(d,i){return i*20+50;})
        .attr('cy',function(d){return height-d;})
        .attr('r','2');
}
```

For the circles, we need to assign a class, so we can repeat to use the circles and generate different instances. Edit below section of codes:

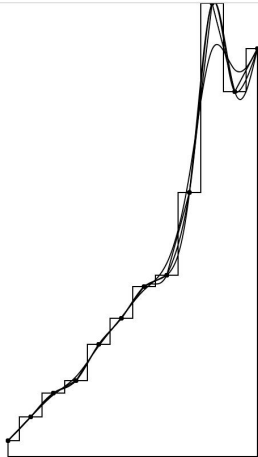
```
grp.selectAll('circle.grpcircle'+t)
    .data(dataArray)
    .enter().append('circle')
    .attr('class',function(d,i){return "grpcircle"+t;})
    .attr('cx',function(d,i){return i*20+50;})
    .attr('cy',function(d){return height-d;})
    .attr('r','2');
```

Refresh the page again and see if there are multiple shapes?

Why not? We need to move the below code block outside the bracket. Refresh again!

```
var svg = d3.select('body')
    .append('svg')
    .attr('height','100%')
    .attr('width','100%')
```

Now you will see this, and it feels that all charts actually overlapped. What's the problem? Look at your codes and try to think about it.



Let's fix this problem now. First of all, we used "i" twice in the loop, and let's replace it with another variable named "t". See the below codes and see the changes. Let's also change the translate position by using the loop index t!

```
for (var t = 0; t<5;t++){

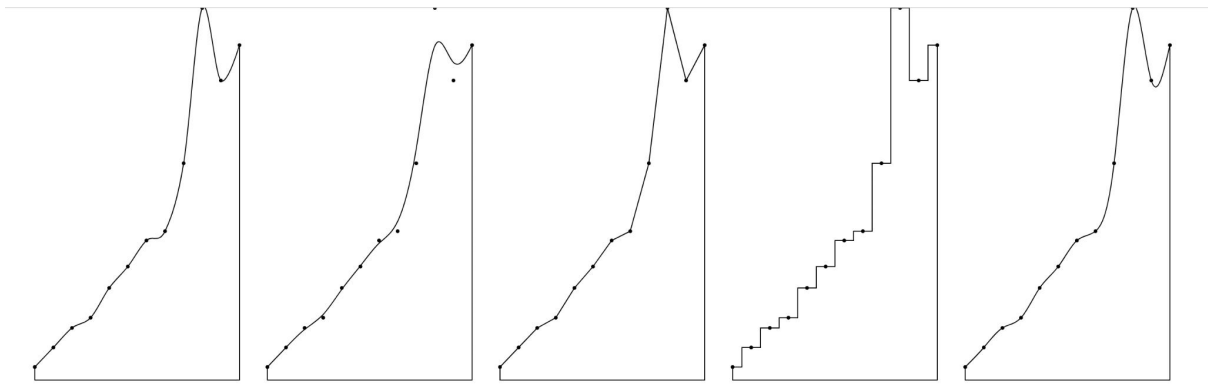
    var area = d3.area()
        .x(function(d,i){return i*20+50;})
        .y0(height) //lowest y coordinate we are using
        .y1(function(d){return height-d;})
        .curve(crvTypes[t]);

    var grp = svg.append('g')
        .attr('transform','translate('+t*250+',0)');

    grp.append('path')
        .attr('fill','none')
        .attr('stroke','black')
        .attr('stroke-width',1)
        .attr('d',area(dataArray));

    grp.selectAll('circle.grpcircle'+t)
        .data(dataArray)
        .enter().append('circle')
        .attr('class',function(d,i){return "grpcircle"+t;})
        .attr('cx',function(d,i){return i*20+50;})
        .attr('cy',function(d){return height-d;})
        .attr('r','2');
}
```

Okay! Now you should be able to get a graph like this!



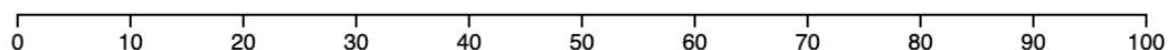
## Task 4: Axes

Create a new file, and name it axes.js script.

Create a new html file named task4.html, copy the code below to the file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet" href="CSS/main.css">
  <script type="text/javascript", src = "d3.v7.js"> </script>
  <title>Lab 8</title>
</head>
<body>
  <nav>
  </nav>
  <script type="text/javascript", src = "axes.js"> </script>
</body>
</html>
```

One of the most useful D3 modules (especially when creating bar, line, and scatter charts) is the axis module which draws axes:



First, we need two things to create a D3 axis:

- an SVG element to contain the axis (usually a g element)
- a D3 scale function A D3 scale function has a domain and range.

The domain specifies the input extent (for example, [0, 100]), and the range defines the output extent (for example, [0, 1000]) of the scale. When a D3 scale function is used to define an axis, the scale domain determines the minimum and maximum tick values, and the range determines the length of the axis. To create an axis:



- make an axis generator function using `d3.axisBottom`, `d3.axisTop`, `d3.axisLeft` or `d3.axisRight` (and pass in your scale function)
- select the container element and pass the axis generator into `.call`

Here's a simple example:

In the `.html` file's body tag, setup up the size of the `svg` component (you can also do it inside the `js` file):

```
<svg width="600" height="100">
  <g transform="translate(20, 50)"></g>
</svg>
```

In the `.js` file, add the below codes:

```
let scale = d3.scaleLinear().domain([0, 100]).range([0, 500]);
let axis = d3.axisBottom(scale);
d3.select('svg g')
  .call(axis);
```

Note: In brief, you pass a function into `.call`. The function's first parameter is a selection on which the function can operate. An axis generator (the `axis` variable in the above example) appends path, line, and text elements (that represent the axis) to the selection. The axis can be positioned by transforming the axis's container. In the above example, the `g` element that contains the axis is translated by (20, 50).

How to set the axis' orientation? `d3.axisBottom`, `d3.axisTop`, `d3.axisLeft` and `d3.axisRight` are used to generate axes that are suitable for the bottom, top, left and right of a chart, respectively:

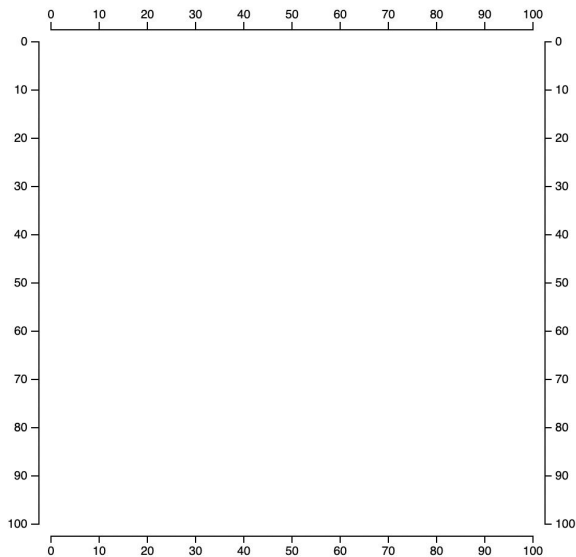
In the `.html` file, add **\*(delete the previously added code)\***:

```
<svg width="500" height="500">
  <g id="left" transform="translate(30, 40)"></g>
  <g id="right" transform="translate(450, 40)"></g>
  <g id="top" transform="translate(40, 30)"></g>
  <g id="bottom" transform="translate(40, 450)"></g>
</svg>
```

In the `.js` file, add **\*(delete the previously added code)\***:

```
let scale = d3.scaleLinear().domain([0, 100]).range([0, 400]);
let axisLeft = d3.axisLeft(scale);
let axisRight = d3.axisRight(scale);
let axisTop = d3.axisTop(scale);
let axisBottom = d3.axisBottom(scale);
d3.select('#left').call(axisLeft);
d3.select('#right').call(axisRight);
d3.select('#top').call(axisTop);
d3.select('#bottom').call(axisBottom);
```

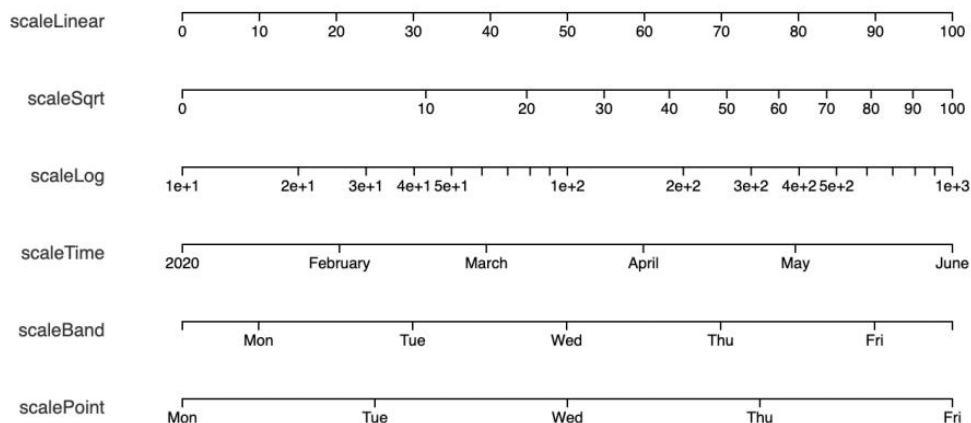
Run your live server and see what the page looks like now!



### \*Scale types

You can pass in any scale function that has numeric output. This includes `scaleLinear`, `scaleSqrt`, `scaleLog`, `scaleTime`, `scaleBand` and `scalePoint`.

Here's an example using each scale type:



### Task 5: Configure the axis' number of ticks (values)/lab/size

You can create a new `.js` and a new `.html` file for this task.

Remember to add the code below to `.html` file:

```
<svg width="600" height="100">
  <g transform="translate(20, 50)"></g>
</svg>
```

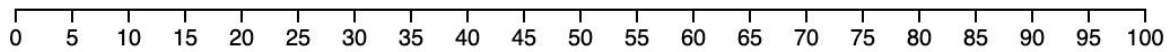
You can configure axes in the following ways:

- specify the number of ticks OR specify the tick values
- specify the format of the tick label (for example, add a percentage sign)
- specify the tick size

### Number of ticks

You can use the `.ticks` method to specify how many ticks the axis has:

```
let scale = d3.scaleLinear().domain([0, 100]).range([0, 500]);
let axis = d3.axisBottom(scale);
axis.ticks(20);
d3.select('svg g')
.call(axis);
```

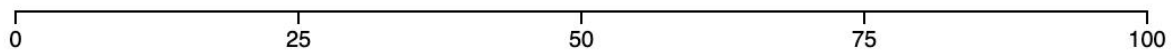


Note: D3 tries to use as many ticks as requested, but in some cases, it'll use more or less for the tick values to be round numbers. The axis uses its scale function's `.ticks` method to generate an array of tick values.

### Tick values

You can specify the axis's tick values by passing an array of tick values into the `.tickValues` method **\*( the previous code )**:

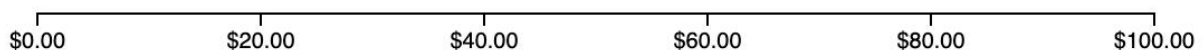
```
let scale = d3.scaleLinear().domain([0, 100]).range([0, 500]);
let axis = d3.axisBottom(scale);
axis.tickValues([0, 25, 50, 75, 100]);
d3.select('svg g')
.call(axis);
```



### Tick label formatting

You can format the tick label in two ways. The first is to use the `.ticks` method and pass in a format string as the second argument:

```
let scale = d3.scaleLinear().domain([0, 100]).range([0, 500]);
let axis = d3.axisBottom(scale);
axis.ticks(4, "$.2f");
d3.select('svg g')
.call(axis);
```



Note: We've already seen that the first argument of `.ticks` is the number of ticks. You can pass in null if you want to use the default number of ticks.

The second approach is to pass a formatting function into the `.tickFormat` method. The function accepts a value and outputs a formatted value.

In this example, we add a % symbol to each tick value:

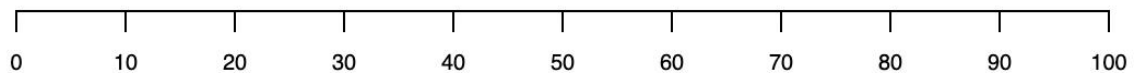
```
let scale = d3.scaleLinear().domain([0, 100]).range([0, 400]);
let axis = d3.axisBottom(scale);
axis.ticks(4)
  .tickFormat(function(d) {
    return d + "%";
  });
d3.select('svg g')
  .call(axis);
```



## Tick size

The length of the ticks can be set using the `.tickSize` method. You can also set the distance between the tick and the tick label using `.tickPadding`:

```
let scale = d3.scaleLinear().domain([0, 100]).range([0, 500]);
let axis = d3.axisBottom(scale)
  .tickPadding(10)
  .tickSize(10);
d3.select('svg g')
  .call(axis);
```



Play around with the `tickSize` and `tickPadding` variables and see what happens?