Lab 4: D3 Chart Types & Scales (Pre Lab)

Wijaya, Mario edited this page 6 hours ago · 23 revisions

This lab is scheduled for **Mar. 2nd (Friday).**

Learning Objectives

After completing this lab you will be able to:

- Be able to differentiate between *scales* and *axes*
- Understand the concept of *input domains* and *output ranges*
- Know how to create scale functions in D3
- Know how to use scales to create axes in D3

Prerequisites

- Update your *starter code repository* using one of the following methods:
  i. From the GitHub Desktop app click `Sync` on the top right
  ii. Open a command line prompt. Navigate to the repository directory, for example `cd ~\Development\CS4460-Spring2018\Labs` and run command `git pull`.
- You have **read Chapters 7 and 8** in D3 - Interactive Data Visualization for the Web by Scott Murray

Additional Reading

- d3: scales, and color by Jerome Cukier
- Introducing d3-scale by Mike Bostock (creator of D3)
- Let's Make a Bar Chart II by Mike Bostock
- Let's Make a Bar Chart III by Mike Bostock
- D3.js Scale API
- Advanced D3: More on selections and data, scales, axis by A. Lex of U. of Utah
- D3 V4 - What's new? by Irene Ros of Bocoup

> This lab will cover D3 scales and axes. **It is expected that you understand the basics of D3.** That includes d3-selections, how to make a simple data-binding with d3 (i.e. `.data().enter().append()`), and how to declare attributes and styles for DOM elements based on bound data (i.e. `.style('fill', '#333')` and `.attr('x', 80)`). This material can all be found in the Lab 3 - Intro to D3.

Important: Scott Murray's book Interactive Data Visualization for the Web was written for D3 version 3.x. You will notice that some of the version 3.x code looks different from the version 4.x code. Most notably, the namespace for version 4.x follows a `camelCase`, whereas 3.x uses `.dot.notation` - this is really the only major difference.

D3 Scales

Until now, when creating a D3 visualization, the starter code has computed x and y values based on the data for you. Or, in other cases the data could be directly mapped to visual attributes (e.g.

Clone this wiki locally

`https://github.gatech.edu`

percentage to `<div>` width). The values in any dataset are unlikely to correspond exactly to pixel measurements for use in your visualization. Now we will show you how to take any data and create a JavaScript mapping to pixels on the canvas.

→ This is where scales come in. We specify a fixed SVG drawing space in our webpage and scale the data to fit in the dedicated area.

> "Scales transform a number, string or boolean in a certain interval (called the domain) into a number in another interval (called the range)" (Jerome Cukier)

D3 provides built-in methods for many different scales: linear, ordinal, logarithmic, square root, etc. Most of the time you will use linear scale functions, so we will focus on learning this type of scale. You can read more about the types of available D3 scales on the D3 Scale API.

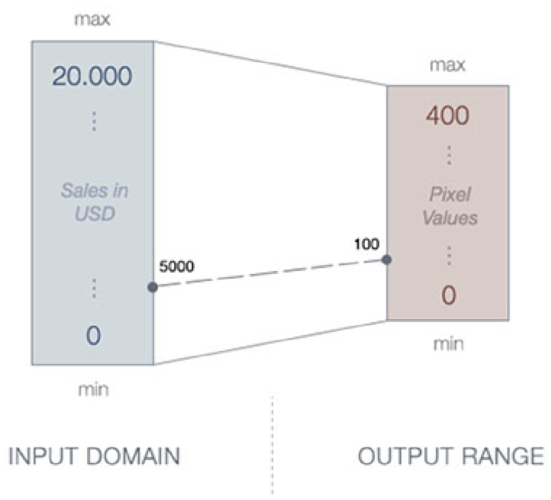
D3 scales are functions with parameters that you define. Once they are created, you call the scale function, pass it a data value, and it nicely returns a scaled output value. You can define and use as many scales as you like.

It might be tempting to think of a *scale* as something that appears visually in the final image—like a set of tick marks, indicating a progression of values. Do not be fooled! Those tick marks are part of an axis, which is a visual representation of a scale. A scale is a mathematical relationship, with no direct visual output. I encourage you to think of scales and axes as two different, yet related, elements.

An Example

Note: Use the code provided in the "example" folder of the Lab 4 code base. Use this as a starting point to understand some of the concepts that are explained in this section. You can try to extend the code to incorporate the missing elements, based on the concepts you will be learning.

We want to visualize the monthly sales of a coffee shop. The input domain (the monthly sales) are numbers between $0 and $20,000. And we want the chart to span an output interval of 400 pixels. We take this input interval, *the domain*, and transform it into the output interval, *the range*.



We could transform the numbers from one domain into the range manually but what if the sales rise above $20,000 and the interval changes? That means a lot of manual work. Thankfully, we can use D3's built-in scaling methods to do this automatically.

Scaling in D3

D3’s scale function generators are accessed with `scale` followed by the type of scale you want (e.g. `Linear` , `Log` , `Sqrt`) to give us `d3.scaleLinear()` or `d3.scaleLog()` - calling either function will generate a new `scale` function.

For example to create the linear scale for our coffee shop data we would use the following:

```
var scale = d3.scaleLinear();
```

Now we want to set the domain and range intervals for our scale. We do that by calling the `domain()` and `range()` methods on our `scale` function. Both methods expect an array of numbers (or strings for categorical scales). Like most D3 functions we can use method chaining here:

```
var scale = d3.scaleLinear()
                          .domain([0,20000])
                          .range([0,400]);
```

Now that we have a scale function, we can map a number from our coffee shop dataset into the pixel output range. To do this we call the `scale()` function on that data number:

```
var barHeight = scale(5000); // returns 100 for $5,000 in sales
```

However, what happens when we don't know the data domain beforehand? When we load a new dataset with `d3.csv` we might not know the domain intervals for all of the column attributes. In this case, D3 provides us with the convenient array functions `d3.min()` , `d3.max()` , or `d3.extent()` :

```
var coffeeData = [
        { month: "May", sales: 6900 },
        { month: "June", sales: 14240 },
        { month: "July", sales: 25000 },
        { month: "August", sales: 17500 }
];

// Returns the maximum value in a given array (= 25000)
var max = d3.max(coffeeData, function(d) {
        return d.sales;
});

// Returns the minimum value in a given array (= 6900)
var min = d3.min(coffeeData, function(d) {
        return d.sales;
});

// Set the scale's domain with min and max
scale.domain([min,max]);

// Returns the min. and max. value in a given array (= [6900,25000])
var extent = d3.extent(coffeeData, function(d) {
        return d.sales;
});

// Set the scale's domain with the extent array
scale.domain(extent);
```

`min()` , `max()` , `extent()` can take either one or two arguments. The first argument is an array of numbers. The second, optional argument is an anonymous function that works as an accessor here. The accessor function is an anonymous function to which `min()` , `max()` , `extent()` hands off each value in the data array, one at a time, as d. The accessor function specifies how to access the value to be used for the comparison.

Incorporating scaled values

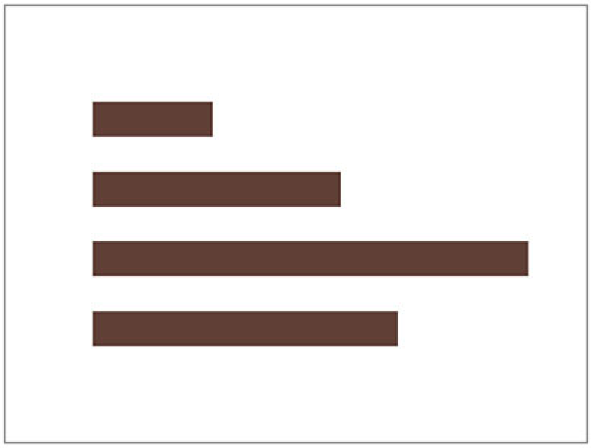
In the next activity we will create scales and then use them to map data attributes to visual properties. Such as the `x` , `y` , and `width` of rectangles in a bar chart. Using scales and coffee shop dataset:

```
var yScale = d3.scaleBand()
            .domain(['May', 'June', 'July', 'August'])
            .rangeRound([40,260])
            .padding(0.5);

var wScale = d3.scaleLinear()
            .domain([0, 25000])
            .range([0,300]);

var svg = d3.select('svg');

svg.selectAll('rect')
    .data(coffeeData)
    .enter()
    .append('rect')
    .attr('y', function(d){
    return yScale(d.month);
    })
    .attr('x', 80)
    .attr('height', yScale.bandwidth())
    .attr('width', function(d){
    return wScale(d.sales);
    });
```



Next try to set the domain of the scales by using the d3 min, max and extent functions that you learned above. You will use the d3-scale functions and the d3-array functions in the following activity to create a bubble chart of Exoplanets captured by the Kepler telescope. This week's lab activities will be focused on re-creating a visualization from National Geographic. **We will be re-creating the following bubble chart:**

Goldilocks Worlds: Just Right for Life? by National Geographic

Also, for those interested, this Exoplanets dataset has been visualized many times in the InfoVis community. Perhaps the most famous is Jer Thorp's breathtaking "Minority Report" style tool. I would recommend watching the showcase video after class:

Exo: A Visualization of Kepler’s Exoplanet Candidates created by Jer Thorp and blprnt.

Activity 0 - Creating and using scales

> Start an http server for this lab's directory. You can accomplish this by opening a command line window, navigating to this lab's directory (e.g. `cd ~/Development/CS4460-`

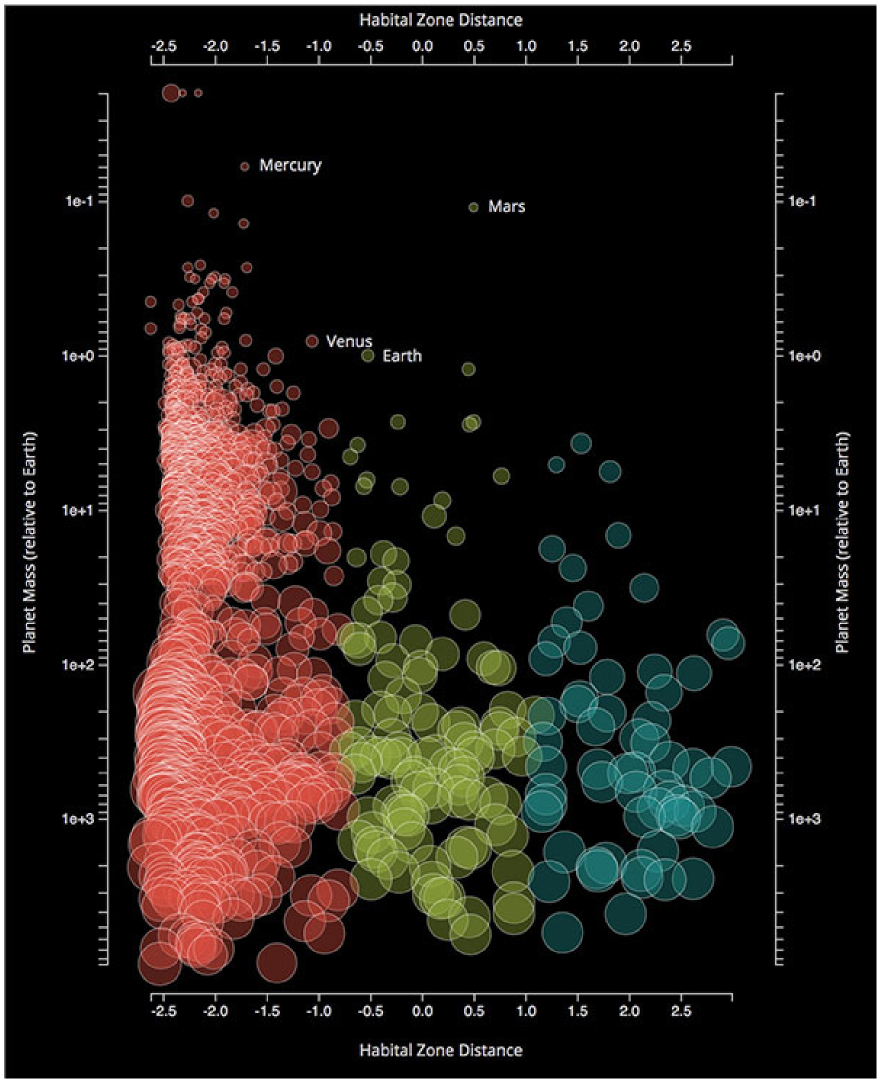
> `Spring2018/Labs/04_lab/`). From there you will start the http server by executing: `python -m SimpleHTTPServer` (for Python 2) or `python -m http.server` (for Python 3). You will need to start an http server for every lab from here on out. Servers are required to serve local files and run JavaScript.

During today's activities you will be working with the `exoplanets.csv` dataset. The dataset includes 1,626 rows. Each row corresponds to an Exoplanet. Here is a snippet of the data table:

name	mass	radius	habital_zone_distance
Kepler-9 d	0.02	2	-2.42
Earth	1	1	-0.52
HD 23596 b	2575.31	11.65	0.1
Mars	0.11	0.53	0.5

- The `mass` and `radius` are ratios to Earth's `mass` and `radius`
- `habital_zone_distance` is a habitability metric for exoplanets

You will be working toward creating the following chart today and in class on friday:



For this activity you will use d3 data-binding to create a circle (`<circle>`) for each Exoplanet. You will then create **4 d3 scale functions** to map the data attributes of the Exoplanets to the visual attribute of their circle on the chart. For this activity and the second activity you will be working with the same HTML/CSS/JavaScript code. All of it can be found in `/04_lab/exoplanets/` .

1. Load the CSV file

Using `d3.csv()` , load the dataset at `./exoplanets.csv` . Make sure to write a callback function to access the loaded data.

We recommend using `console.log()` on the loaded data to double check everything went according to plan.

2. Append a circle for each Exoplanet

Use D3 to bind and append the data to new `<circle>` elements, as you have done before (using D3's `selectAll()` , `data()` , `enter()` , `append()` , etc.).

Circle elements that you appended won't be visible because the default color of the circle is black and the background color of the canvas is black as well. Here are ways to check if you append the circles correctly:

- Inspect your page by right click your page (you need to run local server using python) and select **Inspect element** for Google Chrome browser and look at whether the circles elements are appended to the SVG element
- Change the attribute `fill` of your circles elements to other color (not black), and the circles elements will show up on the screen

If you are lost, please refer back to lab 3 where we covered all of this.

D3 Axes

The current visualization does not really look like a bubble chart yet. It is just a bunch of circles that are nicely arranged. We need x- and y-axes to allow the viewer to actually extract meaningful insights from the visualization.

Drawing axes and legends is critical for data visualization. However, drawing axes by hand is a lot of work. Fortunately, D3 has extensive support for axes. D3 provides methods to create axes, or more precisely, it can display reference lines for D3 scales automatically. These axis components contain lines, labels and ticks. Using the coffee shop example from above we can make an axis for the x-scale:

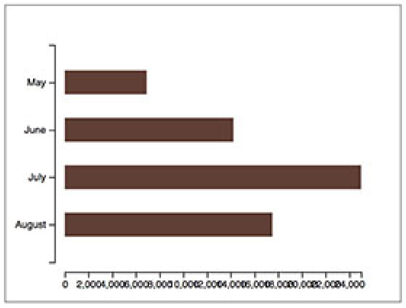
```
// Create a generic D3 axis using the coffee sales scale
var xAxis = d3.axisBottom(scale);

// "Bottom" refers to the orientation of the scale
// "Top", "Right", "Bottom", "Left" are all supported axis orientations
```

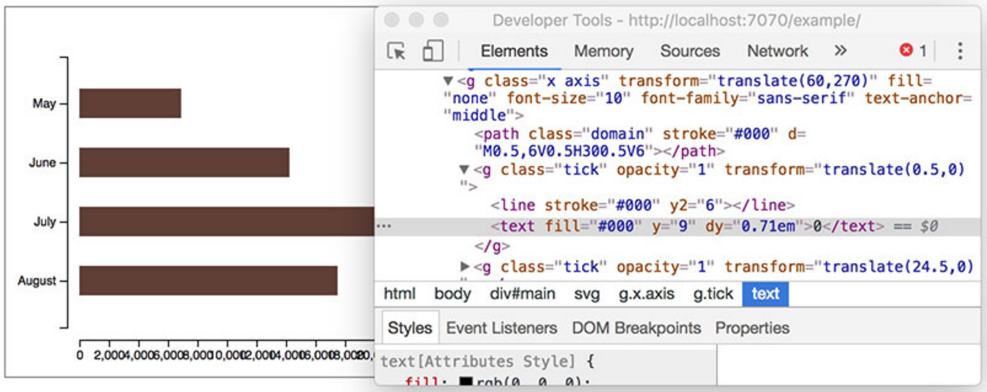
To add the axis to the SVG canvas, we need to create a `<g>` element as a selection and then we have to call the axis function using `call()` . The D3 axis function then automatically creates all the axis elements (i.e. `<line>` , `<text>` and `<path>`) that represent the original `scale` function. All the elements are generated within the `<g>` element:

```
// Append a new <g> element that we will populate the axis with
svg.append('g')
        .attr('class', 'x axis')
        .attr('transform', 'translate(80,250)')
        .call(xAxis);
```

Next try to similarly add a Y-axis to the graph. You can see the result of this code below, notice all of the elements that are automatically created with the d3 axis function:



If we open the web inspector you can see the structure of the automatically generated axis:



```
▼<g class="x axis" transform="translate(60,270)" fill=
"none" font-size="10" font-family="sans-serif" text-anchor=
"middle">
   <path class="domain" stroke="#000" d=
   "M0.5,6V0.5H300.5V6"></path>
 ▼<g class="tick" opacity="1" transform="translate(0.5,0)
 ">
     <line stroke="#000" y2="6"></line>
     <text fill="#000" y="9" dy="0.71em">0</text> == $0
   </g>
 ▶<g class="tick" opacity="1" transform="translate(24.5,0)
```

html body div#main svg g.x.axis g.tick text

Styles Event Listeners DOM Breakpoints Properties

```
text[Attributes Style] {
  fill: ■rgb(0, 0, 0);
```

So what we're doing with the preceding code is to first `append()` a new `<g>` element to the end of the SVG. A `<g>` element is a group element. Group elements are invisible, unlike line, rect, and circle, and they have no visual presence themselves.

Yet they help us in two ways:

1. `<g>` elements can be used to contain (or "group") other elements, which keeps our code nice and tidy.
2. We can apply transformations to `<g>` elements, like we did in the above example to place the axis next to the bars.

So we've created a new `<g>`, and then finally, the function `call()` is called on our new `<g>`. So what is `call()`, and who is it calling?

D3's `call()` function takes the incoming selection, as received from the prior link in the chain, and hands that selection off to any function. In this case, the selection is our new group element. Although the `<g>` isn't strictly necessary, we are using it because the axis function is about to generate lots of crazy lines and numbers, and it's nice to contain all those elements within a single group object. `call()` hands off `<g>` to the `xAxis` function, so our axis is generated within `<g>`.

Cleaning it up

D3 axis functions automatically adjust the spacing and labels for a given scale and range. Depending on the data and the type of the visualization you may want to modify these settings. You can use CSS style sheets to change the appearance of the tick elements:

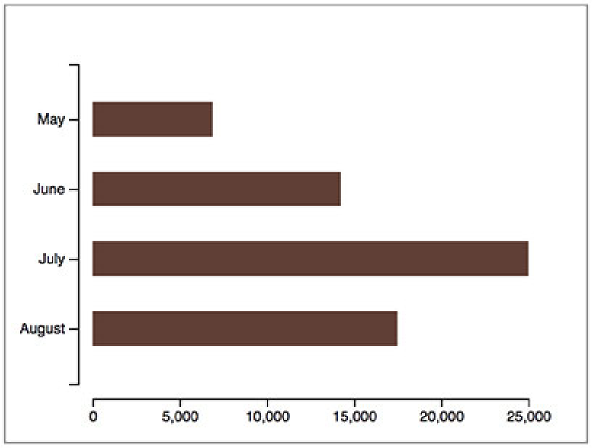
```
.axis .tick text {
        font-size: 10px;
        font-weight: bold;
        fill: #555;
}

.axis path.domain {
        stroke: #555;
}
```

Also, sometimes (like in the example above) D3 creates too many ticks for an axis - making the axis unreadable. In this case we can specify the number of automatically generated ticks with the `.ticks()` function for d3-axis:

```
var xAxis = d3.axisBottom(scale).ticks(6);
```

This will create a more readable axis for our chart:



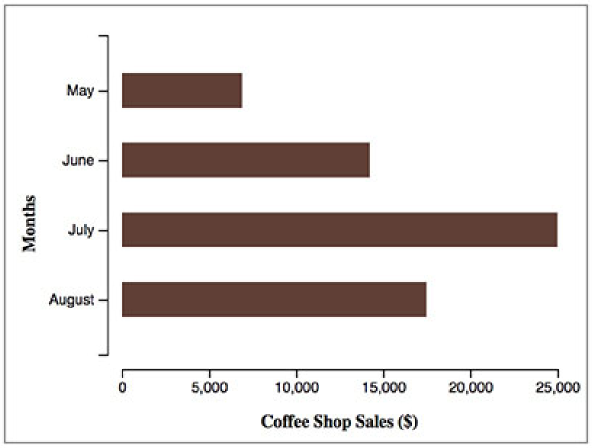
Label your axes!

Unfortunately, in all Mike Bostock's wisdom he has still not added quick and easy functions to add labels to your axes (this is probably because it doesn't require much code). However, you would be amazed at how many D3 charts I see without any titles or labels explaining what the graphic actually means!

In order to add labels to your chart, you can use `d3.append('text')` to append a text element to the SVG canvas. You will then have to transform the `<text>` element to so it is placed alongside your axis. Finally you will need to set the content to something user-friendly with `.text()` . See below for our coffee shop example:

```
svg.append('text')
        .attr('class', 'x label')
        .attr('transform', 'translate(170,290)')
        .text('Coffee Shop Sales ($)');
```

Next try to similarly add a Y-axis label to the graph. Now we can show our chart to someone else without having to explain it to them:


May
June
July
August
Months
0
5,000
10,000
15,000
20,000
25,000
Coffee Shop Sales ($)

Home