Visualizations with D3.js

Learn how to create visualizations with D3.js

Jump to Section

☐ Before we start...

```
We want to make this a great learning experience. We would encourage you
to type all the code yourself and follow the steps. Also please make sure
you understand why and what you are doing.
We have provided checkboxes for you to keep track of your progress.
Work smart and play smart! Don't hesitate to ask the coaches for help.
```

Introduction to HTML and JS

This section is meant for beginners in html and javascript. If you already know the basics, feel free to go to D3JS section.

Create rectangles with HTML code

☐ Create an empty folder for your project and add a html file. Open the file in your favourite text editor and add the following basic template to it.

```
<html>
  <head>
  </head>

  <body>
        Hello world!
  </body>
</html>
```

- [ ] See it in action.

```
Open this file by path in a browser and make sure you can see "Hello
World". Use your browser's developer tools to inspect the element and see
the HTML code. In Chrome, you can right click the text and choose 'Inspect
Element'. Edit the html and type in your name instead of "world".
```

- [ ] A bar chart consists of rectangles grouped together. Lets start with drawing a simple rectangle using plain SVG. You will need to begin with creating an SVG viewport first inside the body tags. Then add a rectangle inside.

```
Notes: Scalable Vector Graphics (SVG) is an XML-based vector image format for two
-dimensional graphics.
The viewport is the visible area of the SVG image. An SVG image can be as wide an
d high as you want, but only a certain part of the image is visible.
```

```
<svg width="400" height="300">
  <rect width="300" height="100" />
</svg>
```

- [ ] Examine the html using your browser's developer tools.

```
When you examine the html, you will see the above code. Try increasing and
decreasing the viewport to be smaller than the rectangle to see how it
affects the display.
```

- [ ] Observation

```
Using SVG directly can be limited and static. Lets draw the same thing
with D3.js.
```

Replace the HTML code with JavaScript code

- [ ] Reference the D3.js library

To get started with D3.js, you need to add the library to your page first. Downlo ad the D3.js source code from https://github.com/mbostock/d3/releases [version 3. 5.5]. Extract the d3.min.js file and place it in the same folder as your html fil e. Then reference the js file from the head section of your HTML code.

```
<head>
   <script type="text/javascript" src="d3.min.js"></script>
</head>
```

- [ ] Replace the HMTL code with JavaScript code

Next use the D3.js library to create the same code in JavaScript that we previous ly created with HTML code.
(You can replace everything within the body tags with this new code.)

Note: Learn about the script tag at https://developer.mozilla.org/en-US/docs/Web/ HTML/Element/script

```
<script>
  var svg = d3.select("body")
       .append("svg");

  var rect = svg
       .append("rect");
</script>
```

- [ ] Look at it in your browser

Refresh your browser page, and examine the html. You should see empty <rect> tags, but you do not see the rectangle displayed yet.

- [ ] Add height and width to the svg and rect elements

Notes: Alternatively, this could be done with CSS to reduce the JS code. You stil l will not see the rectangle displayed.

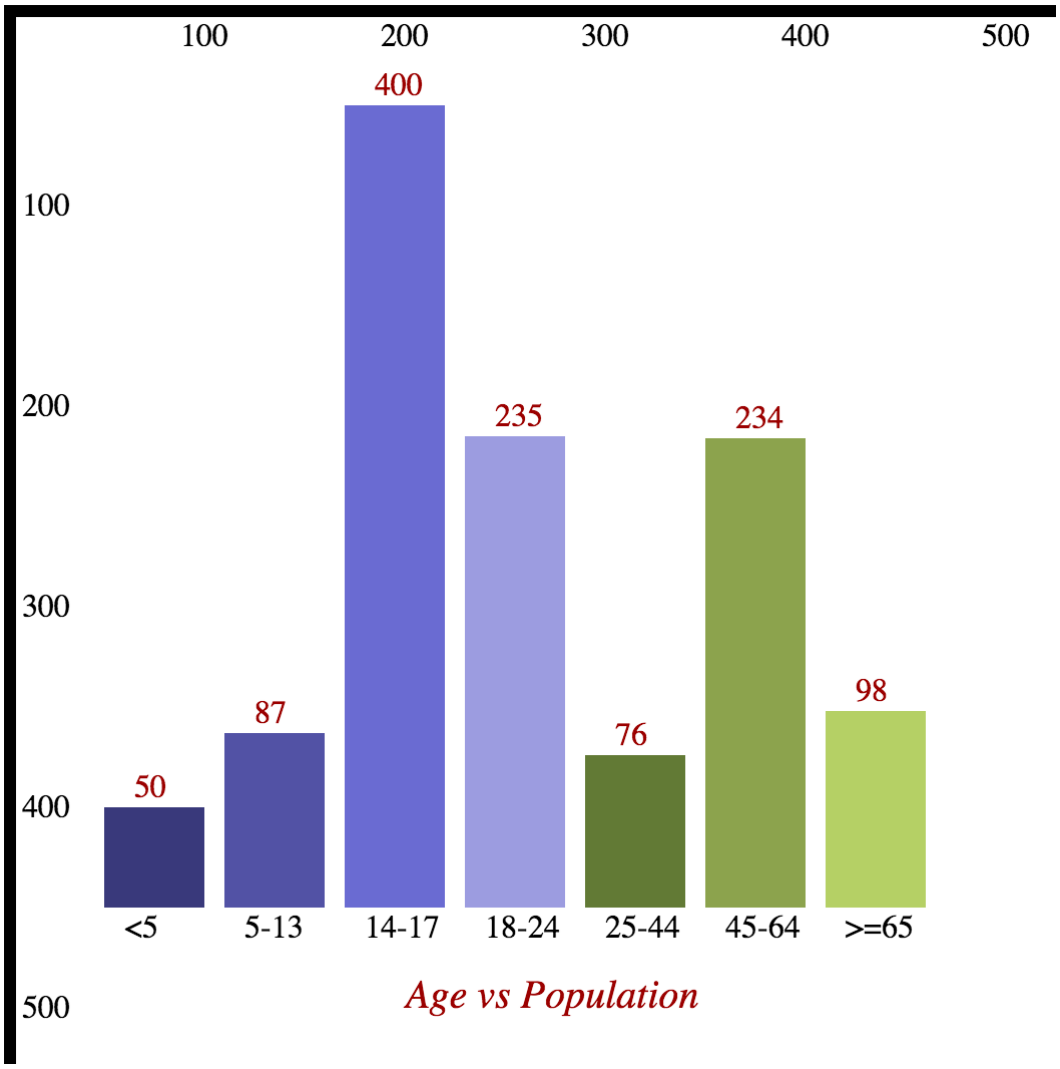
```
svg
  .attr("height", 300)
  .attr("width", 300);
rect
  .attr("height", 300)
  .attr("width", 300);
```

Population Bar Chart with D3.js

Lets use an array of data to setup a simple bar chart with D3.js

- [ ] Preview the chart

Here is screenshot of what your bar chart will look like at the end of this secti on:



Set up the page outline with sample data

- [ ] Simple data set

```
The above section covers some basics of creating rectangles. Now lets add
more rectangles based on a simple data set to draw a bar chart. Remember
how you had to manually add more <rect> tags in html? We will use D3.js to
add them dynamically to represent the data.
```

- [ ] Start with the following code in your html file:

```
Notice we are referencing a JavaScript file that doesn't exist yet.
Ask your coach why it has to be after the chart span, and what the alternative is
.
```

```
<html>
  <head>
    <script type="text/javascript" src="d3.min.js"></script>
  </head>
  <body>
    <span class="chart"></span>
    <script type="text/javascript" src="population_bar_chart.js"></script>
  </body>
</html>
```

- [ ] Create a separate JavaScript file with the following code. Name it population_bar_chart.js.

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

- [ ] Add the following sample data set at the top of the JavaScript file. This represents the population by age in a small town.

```
Note: Normally, your data set will not be embedded within your JavaScript code, b
ut this is a simple example.
```

```
var data = [
   {age: "<5", population: 50},
   {age: "5-13", population: 87},
   {age: "14-17", population: 400},
   {age: "18-24", population: 235},
   {age: "25-44", population: 76},
   {age: "45-64", population: 234},
   {age: ">=65", population: 98}
];
```

Add the bars for the chart

- [ ] Grouping

```
The "g" element represents a group in SVG. Grouping allows us to perform
operations on a group of elements instead of repeating the operation
multiple times. Read more here: https://developer.mozilla.org/en-
US/docs/Web/SVG/Element/g
```

- [ ] Add the following code, refresh the browser and examine the html in your browser.

```
You should see a g tag inside the 'chart' span, ready to add all your rectangles
inside.
```

```
var barGroup = svg.append("g");
```

- [ ] Bind the data to the group.

```
You won't see anything yet, but this will let the elements know about the data se
t. D3 calls this a 'data join'.
Note: You can read more details here: https://github.com/mbostock/d3/wiki/Selecti
ons#data
```

```
var barData = barGroup
    .selectAll("g")
    .data(data).enter()
```

- [ ] Add the rectangles for each piece of data:

```
Make sure you refresh the page after this and examine the html. You should see th
e same number of rect tags as the length of data.
In our case there are 7 age groups, hence 7 rectangles.
```

```
var bars = barData.append("g");
var rect = bars.append("rect");
```

- [ ] But why cant I SEE the rectangles displayed on the page?

```
... because they are missing the height and width. We'll add the width and
height next.
```

- [ ] Set the dimensions of the rectangles.

Each bar will have the same width, but the height will differ based on the population value.
Since we have already bound the data set, we can use that data in anonymous functions to dynamically set values. In this case, we'll set the height.

Note: Research more about JavaScript anonymous functions, or ask your coach. Understand that the d represents the data of that individual rectangle.
More: The author of D3.js explains it here: http://bost.ocks.org/mike/bar/#automatic

```
var barWidth = 50;
rect.attr("width", barWidth);

rect.attr("height", function (d) {
   return d.population;
});
```

- [ ] Still doesn't look right!

Now you should have all your rectangles. But you won't see them yet. If you examine the html, you will all the rect tags with the correct heights and widths, but when you hover over the each tag, you will notice they all overlap each other. That is because we have not defined where these rectangles should be placed, they are all drawn from the same start position in SVG [0,0] which is the top left corner of the viewport.

- [ ] Set the x position of each rectangle by using another anonymous function.

Note: Think about what d and i represent in the function's parameters, and why we are adding 10. Don't just copy/paste!

```
rect.attr("x", function (d, i) {
  return i * (barWidth + 10);
});
```

- [ ] We see the bars! But we only see five bars, when there should be seven.

When you examine every rect dom, you will notice the bars are there, but they are outside the viewport (visible area). Hover over the svg element and it will show you a default height and width of 300 x 150. Increase the size so all the bars are visible.

```
var viewPortHeight = 1000;
var viewPortWidth = 1000;
svg
       .attr("height", viewPortHeight)
       .attr("width", viewPortWidth);
```

Add x and y axis for reference

- [ ] Axes

```
You should see all the bars that represent the population data. There is
more you can do to make this more readable. We'll also re-orient the chart
so the bars align at the bottom.

To understand how this works, lets add some helper x and y axes. This is
just to understand the coordinates, we will remove the axes later.

Understanding the axes: The width of the SVG viewport is 1000 units. We
want to map the numbers of 0 to 1000 (.domain([0, 1000])) onto the width
of the view port - which is 1000 units. So we define the range of the
scale as .range([0, 1000]). Then add the xAxis to the main element with
its own group.
```

- [ ] Set up the x axis

```
First setup a linear scale to represent the range of numbers on the x axis. Then
create the x axis with that scale and append it to the svg element.

Notes:
There are other types of scales that can be used. This one is just a simple linea
r scale of continuous numbers.  Read more at https://github.com/mbostock/d3/wiki/
Scales.
The domain is the scale's input values, and the range is the scale's output (disp
lay) values. In this simple case, they are the same. Read more at https://github.
com/mbostock/d3/wiki/Quantitative-Scales#linear_domain.
```

```
var x = d3.scale.linear().domain([0,viewPortWidth]).range([0, viewPortWidth]);
var xAxis = d3.svg.axis().scale(x);
var xAxisGroup = svg.append("g").call(xAxis);
```

- [ ] Do this part on your own: Add the y axis

```
Add a y axis. This will be similar to the x axis, but the orientation must
run top to bottom on the left side of the graph.
Hint: https://github.com/mbostock/d3/wiki/SVG-Axes#orient
```

Re-orient the bars to align at the bottom

- [ ] Now we can go back to re-orienting the graph so that the bars align at the bottom.

```
One of the easiest ways to align the bars at the bottom is to change the y
position of each bar, essentially pushing them down on the page. We'll
calculate each position based on the height of the tallest bar.
```

- [ ] Get the height of the tallest bar

```
In our case the tallest bar will be: age 14-17 with a population of 400. We can u
se javascript anonymous functions again to go through the population of each data
 element. D3.js provides a max function to pick the max of these.
```

```
var maxHeight = d3.max(data, function (d) {
   return d.population;
});
```

- [ ] Now that we have the tallest bar, we can position the rest relative it.

```
rect.attr("y", function (d) {
  return maxHeight - d.population;
});
```

- [ ] Look at the graph

```
Check out the graph in the browser now. The bars should be aligned at the
bottom.
```

Add labels to each bar

You bar chart is ready, but it's still hard to read what it's representing.

- [ ] Add some text for each bar to tell us what it's represents.

```
First, let's make some space for the text. We can use SVG transforms to
move the entire graph. Remember how we added a group tag at the beginning?
We can now apply a transform on that tag to move the entire collection of
bars.
```

- [ ] Introducing: Transformations

```
This transformation moves the element 50px to the right and 50px down. Use whatev
er amount of space you want.
Read about other transform options here: http://www.w3.org/TR/SVG/coords.html#Tra
nsformAttribute
```

```
barGroup.attr("transform", "translate(50, 50)");
```

- [ ] Display the population value for each bar. Use the “text” function to add it to all the bars.

```
bars
       .append("text")
       .text(function (d) {
           return d.population;
       })
```

- [ ] Text position

```
Notice all the text is placed in one point because we did not specify the
x and y coordinates of it. The text coordinates should be similar to the
bar coordinates, but slightly above the top of the bar.
```

- [ ] Specify the x position of the text

Add the the following code to the above chain of functions. (This is the exact same function we used to position the horizontal spacing of the bars.)
You can add a little bit of extra spacing to center the text over the bar if you want.

```
.attr("x", function (d, i) {
   return i * (barWidth + 10)
})
```

- [ ] Do this on your own: Position the height of each text element

```
Specify the y coordinate of the text so that the numbers rest on top of each bar.
Hint: The coordinates should be almost the same as the y-coordinates of the bars.
```

- [ ] Do this part on your own: Add text to identify the age group of each bar

```
Add the age group at the bottom of each bar and give it your favourite color.
Hint: The y coordinates will be different than the coordinates of the population text.
```

- [ ] Add a title to the chart:

```
barGroup
       .append("text")
       .attr("x", 150).attr("y", 450)
       .attr("class", "heading")
       .text("Age vs Population");
```

- [ ] Color the text

Use this code to color the text red. Feel free to use your favourite color. Use color names or html hex codes.
(Alternatively, this could be done with CSS to reduce the JavaScript code.)
Hex code reference: http://www.w3schools.com/html/html_colornames.asp

```
.style("fill", "red")
```

- [ ] Classes and CSS

The reason we add the class attributes, is to help us style some of these element s using CSS. The heading is an example.
Add the following code under the style tags to style your heading or in a separat e CSS file. Feel free to style it anyway you want.
You can also move out some of the other static JavaScript styles into the CSS.

```
.chart .heading {
   font-size: 20px;
   font-style: italic;
}
```

Color the bars

- [ ] Make each bar a different color

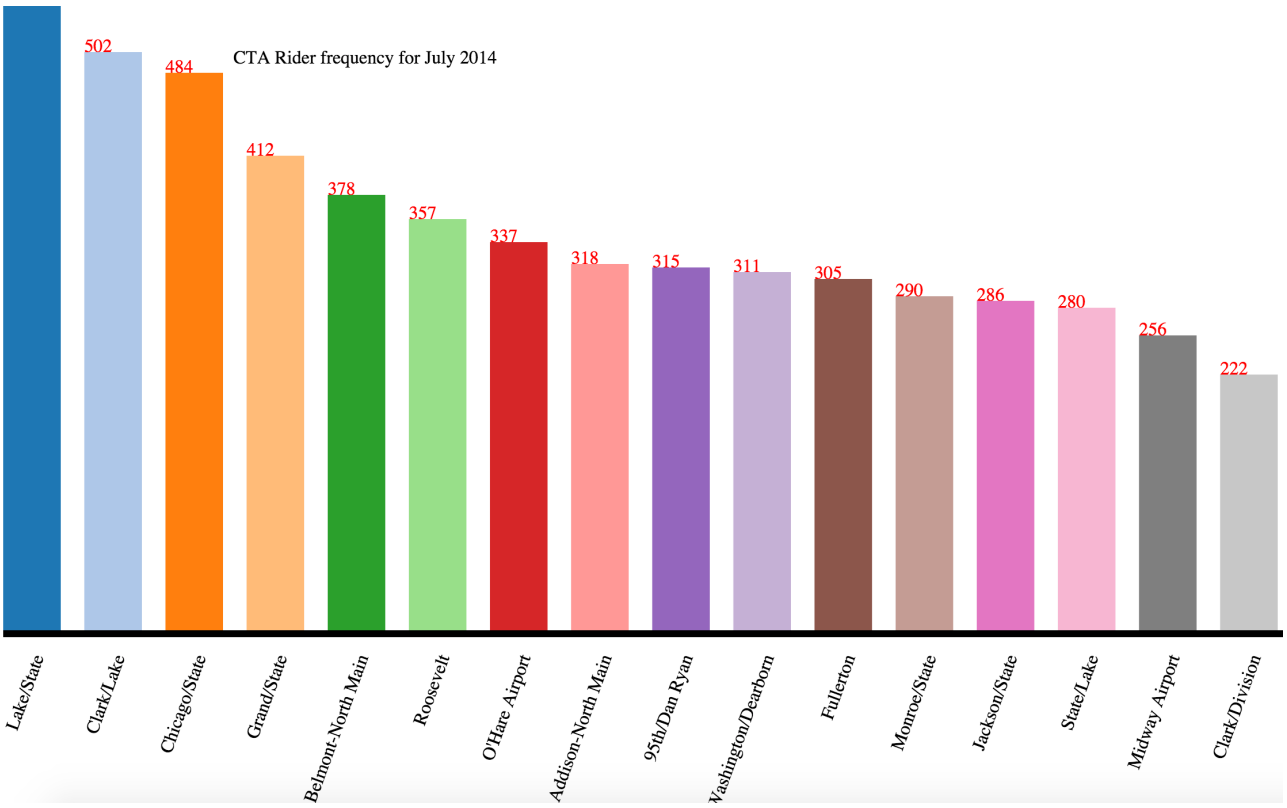
D3's ordinal scale provides some color groups you can use: https://github.com/mbo stock/d3/wiki/Ordinal-Scales#category10

```
var colors = d3.scale.category20();
bars.select("rect").attr("fill", function (d, i) {
   return colors(i)
});
```

D3.js Extension

In this section we learn how to make our graph more flexible to adapt to data from an external source

- [ ] Here is a screenshot of what you are building in this section.



☐ Prerequisites

```
We are going to load data from a csv file and build the bar chart to adapt
to the data. In order to support reading from a file, we are going to need
a server running to feed the file to the browser.

Starting a server: If you are on an OS which has python installed, open a
terminal, go to the directory with the your html files and run the server
on port 8000:

python -m SimpleHTTPServer 8000

Also you can download the data file we are going to use in csv format from
here:
http://bit.ly/1JIZUXg

If you are on a Windows machine, you will need to download python since it
does not come pre-installed (you can download python here:
https://www.python.org/downloads/windows/). If you have downloaded python3
(for example version 3.4.3) then you can run the http server with the
following command:

c:\Python34\python.exe -m http.server
```

☐ Lets start with a skeleton html that is similar to what was done in the previous step. Create a new file with a name as bar_extension.html and add the following html:

```
<html>
<head>
   <script type="text/javascript" src="../d3.min.js"></script>
</head>
<body>
<span class="chart"></span>
<script type="text/javascript">

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

   var group = svg
           .append("g");

</script>
</body>
</html>
```

About the .csv file

Before we start inserting a the bars that correspond to our data, let us first prepare our data from the “cta-rides-july-2014.csv” file. If you open the file, you will see a lot of columns and data under it. Lets say, we want to build something to visualize what the busiest CTA stations in Chicago are. We have filtered the csv file to contain only data for July 2014. This is data collected and published by the city of chicago, you can find more here: https://data.cityofchicago.org/.

Make sure you refresh your browser and examine the changes with every piece you build!

Read the file

Lets read the file, clean up the data and extract only the points we need. D3Js offers us a very easy way of dealing with data files. D3Js helps you read directly from many file formats, some being .csv, .tsv, json, html, etc.
You can read from CSV using the function "d3.csv" [https://github.com/mbostock/d3/wiki/CSV].
Method signature: d3.csv(url[[, accessor], callback]). Place the following code to begin reading your file.

```
d3.csv("cta-rides-july-2014.csv");
```

☐ The above line should have loaded the entire csv file into the browser, but we dont see any difference yet because we are not doing anything with the data. This is where the accessor and callback come into picture.

```
Lets add an accessor, that just returns the data as it is when the file is read.
```

```
function (d) {
                return d;
            }
```

☐ Now to see what data is being read, we can use our callback. The callback is a function thats called after the entire file has been read successfully. In our callback, we use javascript to print out the first element of the read data.

```
function (data) {
   console.log(data[0]);
}
```

☐ Examine the console in your browser, do you see the first row on from the csv file, details about the 'Howard' CTA station, being printed? Great ! Looks like you are all set to filter on the data you need. Lets change the accessor to do that.

```
We will only need the station names and monthly total for our bar chart. Check yo
ur browser after you filter as below, you should see on name and total displayed.
```

```
function (d) {
   return {
       stationName: d.stationame,
       monthlyTotal: d.monthtotal
   };
}
```

☐ We are now all set to build the graph. All this happens in the callback function. Lets add the same code to building bars as previous tutorial.

```
Refresh your browser after you enter the below code. Examine the html, you will n
otice we have different <rect> tags with different heights and width of 50px but
all jammed together on one bar.
```

```
function (data) {
                var viewPortHeight = 1000;
                var viewPortWidth = 1000;
                var barWidth = 50;

                svg
                        .attr("height", viewPortHeight)
                        .attr("width", viewPortWidth);


                var bars = group
                        .selectAll("g")
                        .data(data).enter()
                        .append("g");

                var rect = bars
                        .append("rect");

                rect
                        .attr("width", barWidth)
                        .attr("height", function (d) {
                            return d.monthlyTotal;
                        });
            }
```

D3Js Scales

Now, instead of resizing our svg viewport to data, d3js offers us scaling options to resize the graph inside a given viewport.

- [ ] Lets start with the x-axis. The x-axis consists of mapping the datapoints [CTA station names in our case] into the given viewportwidth of 1000.

```
We need to use the "ordinal" scale in d3 for non-quatitative datapoints such as n
ames. Read more: https://github.com/mbostock/d3/wiki/Ordinal-Scales
```

```
var x = d3.scale.ordinal().rangePoints([0, viewPortWidth]);
```

- [ ] The domain on this scale would be what we need to map, thats our station names.

Thus we will be distributing our station names across a width of 1000 on the x-axis. The "data.map" here is a javascript function that creates an array from the data you pass it.

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

- [ ] Next, we use this scale that we just created to map the station names on the x-axis

```
rect
       .attr("x", function (d) {
           return x(d.stationName);
       });
```

- [ ] Refresh your browser... Voila !

Wait a minute….. you will just see a giant block and no individual bars. Hover over the html elements, and you will notice we have not given the bars (each of width 50) enough space, hence they have been crushed inside the 1000 width. Lets fix the viewport width.

- [ ] We will have to consider the bar width: 50, the space between bars: 10 and total number of bars to calculate what the total width SHOULD be.

Replace your original viewPortWidth setting of 100 to this calculated value.

```
var spaceBetweenBars = 20;
var viewPortWidth = data.length * (barWidth + spaceBetweenBars);
```

- [ ] Great! Now we see the bars, but they are really long, because the monthly total goes into 6 digit numbers. We can easily reduce the all the monthly totals, but adjusting the monthly total in the accessor part of reading from the file.

Fix your accessor function after you read the data file. It should look like this ..

```
function (d) {
   var monthTotal = Math.round(d.monthtotal / 1000);
   return {
       stationName: d.stationame,
       monthlyTotal: monthTotal
   };
}
```

- [ ] Refresh your browser. Now you see the bars but they are upside down. We will fix that soon. Before that, Add the y-axis by yourself.

```
You can get more help from here: https://github.com/mbostock/d3/wiki/Quantitative
-Scales
```

```
The Y-axis will be similar, except the scale this time is "linear", not
"ordinal" since we will be mapping a number (the month totals). Hint:
Your y-axis will will be in the range between the tallest bar and 0. Also
the domain, the possible values will be between 0 and the tallest point.
```

- [ ] Calculate the max height or tallest bar with d3.max function as follows

```
d3.max(data, function(d) {
                    return d.monthlyTotal;
                })
```

Adding text

- [ ] Adding text is similar to what you did in the basic tutorial. Only difference is here, since we already have a scale defined for x and y coordinates, we can use that scale to map points on the graph for the text.

```
Refresh your screen, you will see the red text aligned to the different bars vert
ically, but not horizontally.
```

```
bars
       .append("text")
       .attr("y", function (d) {
           return y(d.monthlyTotal);
       })
       .style("fill", "red")
       .text(function (d) {
           return d.monthlyTotal;
       });
```

- [ ] Do this yourself: Distribute the text along the x-axis or horizontally. Above, to find out what y coordinate the monthlyTotal needs to appear, we have found out the points on the "y" scale where the monthlyTotals exists. Thus you see: y(d.monthlyTotal);

```
Add the x attr for the text. Hint: remember, we need to find points at
which station names are mapped on the x scale.
```

Adding y and x axes.

Now lets look into adding an axis at the bottom with ticks that have the stationNames on them.

- [ ] D3Js axes

```
D3js provides an easy way of drawing an axis using the scales that we
created earlier to draw the bars.
[https://github.com/mbostock/d3/wiki/SVG-Axes]
```

- [ ] Lets start by using the axis function and adding an axis to use the already existing "x" scale. We then give the axis its own group and call the xAxis which begins to draw the horizontal line.

```
var xAxis = d3.svg.axis().scale(x);
var xAxisGroup = group
       .append("g")
       .attr("class", "x axis")
.call(xAxis)
```

- [ ] Do this yourself: position the axis

```
Lets use the translate attribute we learnt about in the basic tutorial to move the bar to the bottom. Exercise: Add an attribute to the xAxisGroup, to push it down by maxHeight. You can use transform-translate combo. Remember we don't need to move it to the left of right. you can use: translate(0," + maxHeight + ")
```

Sorting the bars from most used CTA station to least.

- [ ] In order to do this, all we need to do is sort the data coming in, BEFORE using it to draw a graph.

```
Again d3js provides a "sort" function to make this very simple. We can use the sort function to and provide it two things: 1) what order it the data needs to be sorted? ascending or descending? 2) On what data point should it be sorted? In our case we need to sort by monthly totals.
```

- [ ] Write your sort function, as the first thing that happens in the callback for the csv reader:

```
data = data.sort(function (a, b) {
   return d3['descending'](a.monthlyTotal, b.monthlyTotal);
});
```

Prettify: Lets fix he names overlapping each other at the bottom.

- [ ] We can use "transform" again, but this time to rotate the text of the station names. This way we can make sure the names dont overlap each other.

```
We can do that easily, by first selecting all the "text" attributes in the xAxisG roup. The applying a rotate on that text. We also add some styling to make the te xt more visible.
```

```
.selectAll("text")
.attr("transform", "rotate(-70)")
.style("text-anchor", "end")
```

- [ ] A few more things you can do yourself!

```
Exercise: Notice how the axis line now cuts the text into half after
rotation. Add x and y attributes to the xAxisGroup chain and move to a
desired position. [Dont forget to try coordinates on a negative space as
well !]
Exercise: Add colors to every "rect"
Exercise: Add text “CTA Rider frequency for July 2014” to the "group" and
place it in the appropriate position
Exercise: Lastly, add a transform to the "group" on the entire chart to
move the chart down by 50px.
```

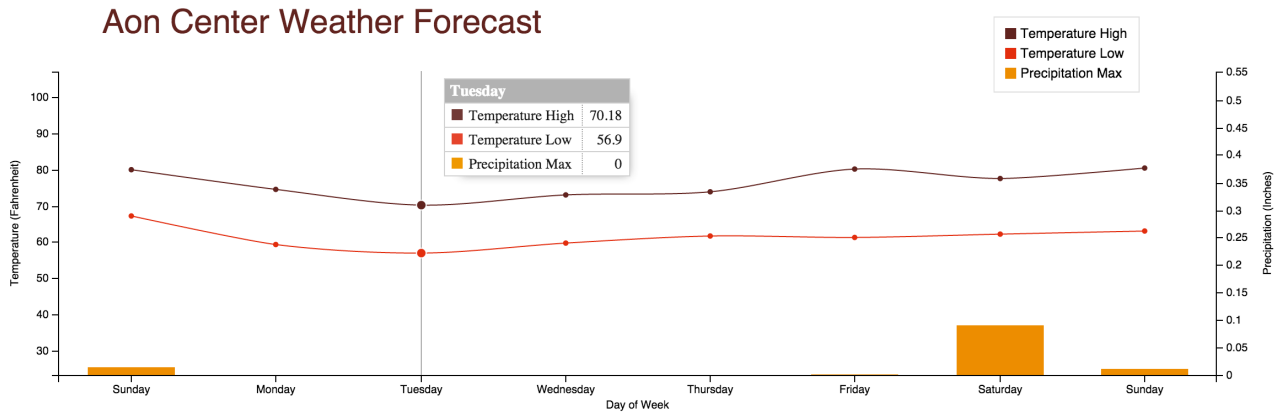
Weather Forecast Chart with C3.js

- [ ] Wow! That took a lot of time to get a basic chart done. D3.js has a very rich feature set that will help anyone build out minute details of a graph. Now that we have explored the basic principles, we'll look at an easier way to build standard graphs.

```
There are many other libraries that are built on top of D3.js to abstract
the basics and make development easier and more fun.
Examples are C3.js, NVD3, Dimple, etc.
In this example we'll try out C3.js using some data about the weather at
the AON Center this week.
```

- [ ] Preview the chart

```
Here is screenshot of what your chart will look like at the end of this section,
depending on how you customize it:
```



Create a basic graph with C3.js

- [ ] Download the C3.js source code.

```
Copy the c3.min.js and c3.min.css files from the source package into your
working directory.
Download from https://github.com/masayuki0812/c3/archive/0.4.10.zip
```

- [ ] Create an HTML with the following code.

Notice that it includes the source for D3 and C3, a div with an id of 'weather-chart', and a new JavaScript file that we are about to create.

```
<html>
  <head>
    <script type="text/javascript" src="c3.min.js"></script>
    <script type="text/javascript" src="d3.min.js"></script>
    <link rel="stylesheet" type="text/css" href="c3.min.css" />
  </head>
  <body>
    <div id="weather-chart"></div>
    <script type="text/javascript" src="aon_daily_weather.js"></script>
  </body>
</html>
```

- [ ] Add the data file

```
Download this json file and put it in your working directory:
https://raw.githubusercontent.com/sarah12345/d3-c3-
charts/master/c3_weather/aon_daily_weather.json
Take a look at it to understand the structure of the data.
This data was retrieved from forecast.io. There are many sites that host
free data that you can use to experiment.
```

Create the JavaScript file

```
Add this code to a new JavaScript file named aon_daily_weather.js.

This code uses the C3.js library to generate a default graph.
It sets a few properties already:
  bindto: This specifies the id of the html element that will hold the graph
Three data properties:
  url: This is the name of the data file we just added (it's possible to put the
data inline here instead)
  mimeType: This specifies the type of data that's in the data file
  keys: This specified the keys from the data that should be used as values in th
e graph
```

```
var chart = c3.generate({
  bindto: '#weather-chart',
  data: {
    url: 'aon_daily_weather.json',
    mimeType: 'json',
    keys: {value: ['temperatureMax', 'temperatureMin']}
  }
});
```

Run a simple web server

```
You'll need to run a simple web server since the code is reading the data from a
separate file.
For Mac, python is already available. Just run the command below from the command
 line.
```

```
python -m SimpleHTTPServer
```

Checkout your browser

```
You've got a chart already! That was so easy, it even has nice styles and
hover functionality. Now we'll customize it.
```

Customize the x axis

- [ ] Set the x axis labels

Inside the data keys properties, add an entry that sets the x values to the 'time ' field from the data file.
The full keys should look like the code below.
When you view it in the browser, the data will still be in its raw format (millis econds since epoch)

```
    keys: {
      x: 'time',
      value: ['temperatureMax', 'temperatureMin'],
    }
```

- [ ] Format the x axis labels

Specify 'timeseries' as the x-axis type to identify that the x values are dates.
Use formatting from the D3 library to display the date as the name of the weekday .
Format specifiers are documented here: https://github.com/mbostock/d3/wiki/Time-F ormatting

```
  axis: {
    x: {
       type: 'timeseries',
       tick: {
         format: '%A'
       }
    }
```

Falling back to D3.js for the chart title

- [ ] If visualizations easier with other libraries, why ever use D3.js?

```
Not all functionality is available in the abstracted libraries. D3 gives
you more control.
For example, C3 doesn't let you set a title for a graph unless it's a
donut chart.
(Why, I don't know...)
```

- [ ] Set the chart title using D3.js

```
This code should look familiar from the first part of the tutorial. Add it to the
 JavaScript file.
Feel free to add a class and style it with CSS.
```

```
    d3.select("svg").append("text")
      .attr("x", 100 )
      .attr("y", 50)
      .text("Aon Center Weather Forecast");
```

- [ ] Using callbacks

```
The title probably didn't get rendered. That's because the D3 code may be trying
to execute before the C3 code is done executing.
We can control the order of execution by using callbacks provided by C3. Move the
 D3 code into a function assigned to the oninit property of the C3 code.
```

```
  oninit: function () {
    (d3 code goes here)
  }
```

- [ ] Move the graph down to make space for the title

```
The title is rendering right on top of the graph. Add some top padding to the gra
ph to make space.
```

```
  padding: {
    top: 80
  }
```

- [ ] Increase the graph height

The graph looks a little smashed now, so increase the height.

```
size: {
  height: 400
}
```

Add another value to the chart: Precipitation

- [ ] Add precipitation to the chart

Update the data > keys > value attribute to include the precipIntensityMax field from the data file.
When you refresh the page, another line will appear on the chart.

```
value: ['temperatureMax', 'temperatureMin','precipIntensityMax']
```

- [ ] Configure separate axes for the different data ranges

The precipitation data has a much smaller range than the temperature data, so it makes sense to create a separate y axis on the right side of the graph to measure it more granularly.
Update the axis section of the code to include configuration for a y axis (left side) and a y2 axis (right side).
While you're at it, add a label for the x axis.

```
axis: {
    x: {
      type: 'timeseries',
      tick: {
        format: '%A'
      },
      label: {
        text: 'Day of Week',
        position: 'outer-center'
      }
    },
    y: {
      max: 100,
      min: 30,
      label: {
        text: 'Temperature (Fahrenheit)',
          position: 'outer-middle'
        }
    },
    y2: {
      show: true,
      max: 0.5,
      label: {
        text: 'Precipitation (Inches)',
          position: 'outer-middle'
        }
    }
  },
```

- [ ] Assign the different y axes to the correct data points

Add axes assignments inside the data configuration. Assign temperature to the y axis and precipitation to the y2 axis.

```
    axes: {
      temperatureMax: 'y',
      precipIntensityMax: 'y2'
    },
```

- [ ] Update the chart types

Since precipitation is very different than temperature, it makes sense to display it differently.
We'll make temperature data show up as lines and precipitation show up as bars.
Add this code in the data section of the configuration, to specify the chart type for each data point.

```
    types: {
      temperatureMin: "spline",
      temperatureMax: "spline",
      precipIntensityMax: "bar"
    }
```

Customize the Legend

- [ ] Position the legend over the graph

Add this code to the chart configuration to position the legend in the top right corner of the graph.
Adjust the position however you like.

```
  legend: {
    position: 'inset',
    inset: {
      anchor: 'top-right',
      x: 80,
      y: -50,
    }
  }
```

- [ ] Make the data labels more readable

Add this code in the data section of the configuration. It will affect the data labels in the legend and the hover.

```
    names: {
      temperatureMin: "Temperature Low",
      temperatureMax: "Temperature High",
      precipIntensityMax: "Precipitation Max"
    }
```

- [ ] Update the colors

Specify the color for each data point. This will update the color in the legend and on the graph.
Add this code in the data section of the configuration. Use whatever colors you want by name or hex value.

```
    colors: {
      temperatureMin: 'red',
      temperatureMax: 'orange',
      precipIntensityMax: 'turquoise',
    }
```

On Your Own

- [ ] Have fun

Now that you know the basics of D3.js and C3.js, explore an area you are most interested in.

Some ideas:
* Use a different data set. There is a lot of data available for free on the internet, though you might need to massage it.
* Make a different type of graph. Try a donut chart or scatter chart.
* Use some of the C3.js callbacks. Make it change to a different type of graph when you hover over it.
* Style one of your graphs to look unique.

Resources:
* The City of Chicago provides interesting data sets here: https://data.cityofchicago.org/
* D3.js ideas: https://github.com/mbostock/d3/wiki/Gallery
* D3.js documentation: https://github.com/mbostock/d3/wiki/API-Reference
* C3.js ideas: http://c3js.org/examples.html
* C3.js documentation: http://c3js.org/reference.html
* D3.js plugins: https://github.com/mbostock/d3/wiki/Plugins

Chicago Datasets:
The following pages will give you datasets to play with, and show you examples of how you might graph them. To get the raw data, click on the 'export' button, and choose json or csv.
* Crimes 2001 to present: https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2
* Beach Water Quailty: https://data.cityofchicago.org/Parks-Recreation/Beach-Water-Quality-Automated-Sensors-2015-Water-T/3ipq-j3gd
* Potholes patched last 7 days: https://data.cityofchicago.org/Service-Requests/Potholes-Patched-Last-Seven-Days/xpdx-8ivx
* List of 'L' stops: https://data.cityofchicago.org/Transportation/CTA-System-Information-List-of-L-Stops/8pix-ypme
* Divy bicycle stations: https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations/bbyy-e7gq