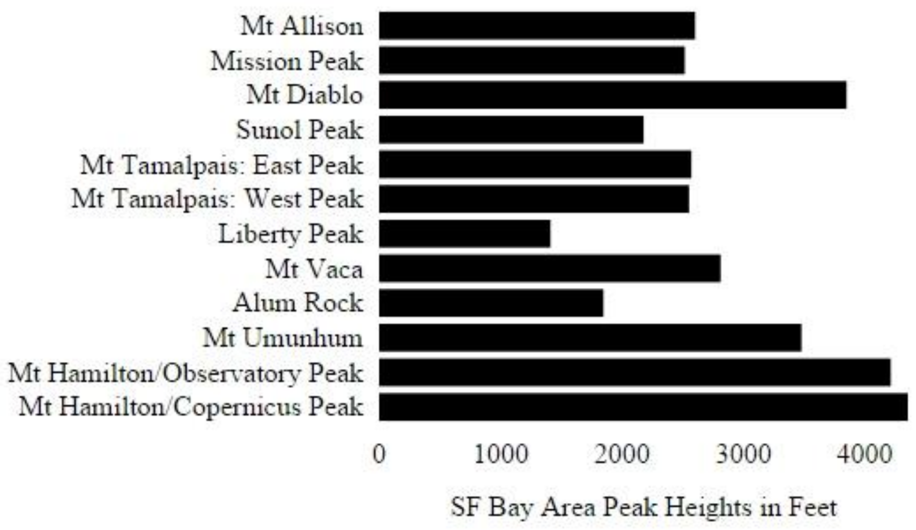
Making a horizontal bar graph using the D3 JavaScript library

By Frank Mock

http://frankmock.com/d3graphs/bargraph.pdf



The purpose of this document is to describe the process in building a bar graph using the D3 JavaScript library. This tutorial assumes you have some HTML and JavaScript experience. I will use the words method and function interchangeably. I will also only offer brief explanations as to how I am coding this and recommend to learn additional information by reading the D3 API documentation. As stated on the D3 website, http://d3js.org/ D3.js is a JavaScript library for manipulating documents based on data.

A Simple Start

I will start simple and focus just on accessing the data and simple DOM manipulation with D3. Below is the basic HTML 5 webpage I will start with named **index.html**. Notice the script tag which includes

d3.js. Alternatively, you could link to the latest version of D3 by specifying the URL below which was obtained from the D3 website.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.6/d3.min.js" charset="utf-8"></script>
```

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <meta charset="utf-8">
                <script type="text/javascript" src="../d3/d3.js"></script>
        </head>
        <body>


        </body>
</html>
```

The data is contained in a CSV file called **sfbaypeaks.csv**. It is a listing of the heights of some of the SF Bay Area peaks. CSV stands for comma separated values. This file format delimits each piece of data with a comma. The first line is a header that contains a descriptive word for each column of data. There are four columns of data in this file: **name**, **GNIS ID**, **county** and **height(feet)**.

The contents of sfbaypeaks.csv

```
name,GNIS ID,county,height(feet)
Mount Allison,218152,Alameda,2602
Mission Peak,228839,Alameda,2517
Mount Diablo,222343,Contra Costa,3849
Sunol Peak,1723545,Alameda,2178
Mount Tamalpais: East Peak,222916,Marin,2571
Mount Tamalpais: West Peak,1660148,Marin,2552
Liberty Peak,1808946,Marin,1411
Mount Vaca,255187,Napa,2812
Alum Rock,218197,Santa Clara,1847
Mount Umunhum,236770,Santa Clara,3478
Mount Hamilton/Observatory Peak,224848,Santa Clara,4213
Mount Hamilton/Copernicus Peak,1658314,Santa Clara,4354
```

In the CSV file I am only interested in the **name** column and **height(feet)** column. To begin, I just want to display these pertinent data, into the DOM by appending paragraph elements containing name and height information from the file. I realize this is not a very amazing start but, if this step is successful, I am assured that my data access code is correct and that I am not having server problems.

Here is how the documents are stored on my server:

- index.html will be in a directory called sfbaypeaks
- sfbaypeaks.csv will be in a directory named csv
- d3.js will be in a directory called d3.

The following script is added to the body of the HTML document.

```
<script type="text/javascript">

        var width = 600;
        var height = 300;

        //Adds a SVG element in to the body of the page
        var svg = d3.select("body").append("svg").attr("width", width).attr("height", height);

        function drawGraph()
        {
                d3.csv("../../csv/sfbaypeaks.csv", convert, function(myData){
                        myData.forEach(function(d){
                                d3.select("body").append("p").text(d["name"] +" "+ d["height(feet)"] +" ft.");
                        });
                });
        }

        //Convert the strings representing the height into numeric values
        function convert(d)
        {
                d["height(feet)"] = parseFloat(d["height(feet)"]);
                return d;
        }

        drawGraph();
</script>
```

The graph will be drawn inside a SVG element. SVG stands for scalable vector graphics. SVG graphics are graphics that can be scaled up or down without loss of image quality. First, I created the variables width and height for the width and height of the SVG element respectively. Next, I use D3 to add a SVG element to the DOM.

```
d3.select("body").append("svg").attr("width", width).attr("height", height);
```

With D3 it is very easy to select elements of the DOM and make changes. The line of code above selects the body tags of the DOM and appends a SVG element to it. Then, using method chaining, the width and height attributes are added to the SVG element. Now, that there is an SVG element to draw graphics to. Next, I make a function called drawGraph that will draw the graphics to the SVG element. But first, inside this function, I will just use D3 to append data from the CSV file to the DOM within paragraph tags. To read in the contents of a CSV file you can use the **d3.csv** method. This method will store the contents of the file in an array. The first argument takes the name of the CSV file. The next argument, which is optional, is a function that works on the data contents of the file. In this case, my function named **convert,** converts the strings that represent the peak heights into numeric values. If you do not do this, d3.csv will write the heights as string values in the resulting array. The third argument is an anonymous function that works on the resulting array that d3.csv creates. In this case, I gave the array the name **myData**. This array will be an array of JavaScript objects which will have fields named after the data descriptors named in the header line of the CSV file. Below is a representation of myData showing only the first two objects.

```
myData = [
            {name: "Mount Allison", GNIS ID: "218152",county: "Alameda", height(feet): 2602},
            {name: "Mission Peak", GNIS ID: "228839",county: "Alameda", height(feet): 2517}
          ]
```

Next, I append paragraph elements to the DOM using a forEach loop, d3 DOM selection and appending as before.

```
myData.forEach(function(d){
                    d3.select("body").append("p").text(d["name"] +" "+ d["height(feet)"] +" ft.");
});
```

Notice the method selection and append chain is used just as before. The anonymous function passed to the forEach loop works on each element of the myData array. The fields of each object are selected from the array using **d["field_name"].**

With D3, there is another way to loop through the data and append selected fields of each object to DOM elements. After making the DOM selection you can use the d3 **data** method to attach the data array to the d3.selection and then the **enter** method to work on new elements to the DOM. In this case, the new DOM elements are the paragraph elements we will append to the body of the HTML document. I changed the drawGraph function to reflect this different approach to looping through the data.

```
function  drawGraph()
{
          d3.csv("../../csv/sfbaypeaks.csv", convert, function(myData){
                    d3.select("body")
                       .data(myData) //attach myData to the current d3 selection
                       .enter() //for each new element that enters the DOM
                       .append("p") //add a paragraph tag
                       .text(function(d){return d["name"] +" "+ d["height(feet)"] +" feet.";});
          });
}
```

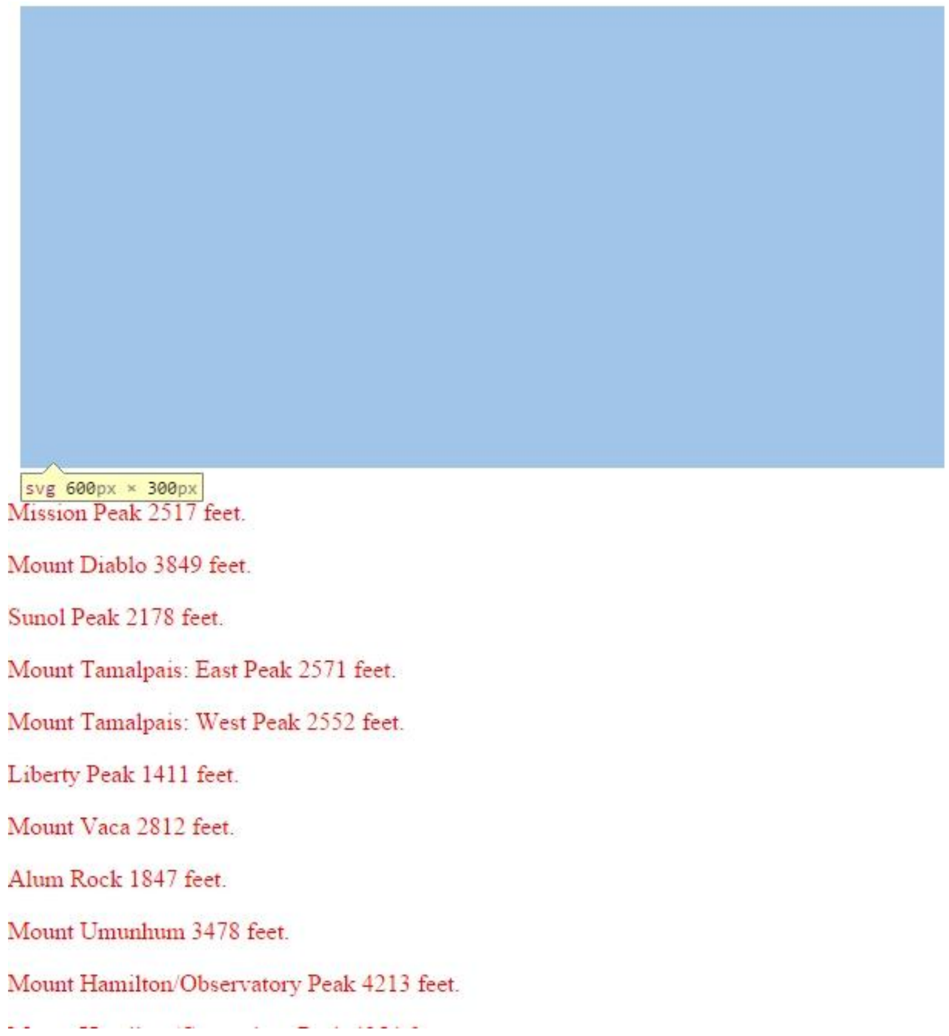
By the way, attributes can be added to the paragraph tags that contain the data by using the D3 **attr** method. To change the color of the text for example, add the following to the chain after the text method:

```
.attr("style", "color:blue");
```

Here's the full code so far:

```
<!DOCTYPE html>
<html lang="en">
	<head>
		<meta charset="utf-8">
		<title>Drawing A Bar Graph Using D3</title>
		<script type="text/javascript" src="../../d3/d3.js"></script>
	</head>
	<body>
		<script type= "text/javascript">
		var width = 600;
		var height = 300;

		//Adds a SVG element in to the body of the page
		var svg = d3.select("body").append("svg").attr("width", width).attr("height", height);

		function drawGraph()
		{
			d3.csv("../../csv/sfbaypeaks.csv", convert, function(myData){
				d3.select("body")
				  .data(myData)
				  .enter()
				  .append("p")
				  .text(function(d){return d["name"] +" "+ d["height(feet)"] +" feet.";})
				  .attr("style", "color:blue");
			});
		}

		//Convert the strings representing the height into numeric values
		function convert(d)
		{
			d["height(feet)"] = parseFloat(d["height(feet)"]);
			return d;
		}

		drawGraph();
		</script>
	</body>
</html>
```

And the figure below shows the output with the text colored red (not blue as shown in the code). I highlighted the SVG element so you can see the space it occupies on the web page. This is exactly what I wanted, an empty SVG element and paragraphs displaying text below it. At this point I know I am accessing the data correctly and that my JavaScript code is drawing an SVG element. Next, I will draw SVG rectangles to the SVG element using the data array myData.



```
Mission Peak 2517 feet.

Mount Diablo 3849 feet.

Sunol Peak 2178 feet.

Mount Tamalpais: East Peak 2571 feet.

Mount Tamalpais: West Peak 2552 feet.

Liberty Peak 1411 feet.

Mount Vaca 2812 feet.

Alum Rock 1847 feet.

Mount Umunhum 3478 feet.

Mount Hamilton/Observatory Peak 4213 feet.
```

Figure 1 The paragraph elements and the svg element (the svg does not have color, but is blue here so the area it covers can be seen)

A Better Day Tomorrow
"NO MATTER WHAT!"
by FrankMock
Henry, we need a bar graph on our company website ASAP!
AWARD
No Problem boss! I'll use JavaScript and the D3 library and get it done in no time.
I don't care what you use as long as the graph ALWAYS shows a positive trend in profits NO MATTER WHAT!

Drawing the Bar Graph

Now it’s time to get down to some serious business and build the bar graph. This is where things get fun with D3 and you start to appreciate the built-in functions D3 has to offer. I changed the structure of my code slightly by placing the call to d3.csv at the bottom of the code. Instead of passing an anonymous function as the last argument, I pass drawGraph. Notice that I don’t pass an array of data to drawGraph explicity for it to work on. The csv function does this for me automatically; that is to say it hands the function passed as the third argument the final version of the data array. Let me explain this process again in case I did not make myself clear. First csv imports the data from the CSV file, next it passes it to the second argument which is a function to process the data and finally, it hands the last argument, which is also a function, the nice and shinny data array ready to be used to draw SVG shapes. I give this array a name of myData in the parenthesis of the drawGraph function.

```
<script type= "text/javascript" >
var width = 600;
var height = 300;
var barPadding = 4;
var x = 0;
var y = 0;

//Adds a SVG element in to the body of the page
var svg = d3.select("body").append("svg").attr("width", width).attr("height", height);

function drawGraph(myData)
{
        var maxHeight = d3.max(myData, function(d){return d["height(feet)"];});
        svg.selectAll("rect").data(myData)
                                .enter()
                                .append("rect")
                                .attr("y", function(d, i){return i * (height/myData.length);})
                                .attr("x", 0)
                                .attr("height", height/myData.length - barPadding)
                                .attr("width", function(d){return (d["height(feet)"]/maxHeight) * width});
}

//Convert the strings representing the height into numeric values
function convert(d)
{
        d["height(feet)"] = parseFloat(d["height(feet)"]);
        return d;
}

d3.csv("../../csv/sfbaypeaks.csv", convert, drawGraph);
</script>
```

In the drawGraph method I use method chaining just as before. This time I am attaching them to the global SVG object named svg. Another difference is that rectangles are being appended to the svg element. The **d3.selectAll** method is selecting all the rectangles that will be appended to the svg element; data is being made available to each rectangle by chaining the **data(myData)**; then **enter()** is attached to the chain. I like to think of this method as the beginning of a forEach loop. Whatever is chained after this method will be repeated for all new items that enter the DOM. In this case, for all new rectangles that are appended to the svg element. Rectangle x, y, width and height attributes are added to the rectangle just as attributes were attached to the paragraph elements of the previous section; by chaining the attr function.

To ensure all the rectangles fit inside the svg element, the svg element’s height is divided by the number of elements in the myData array. This will yield the height if each rectangle. A padding of 4 pixels is subtracted to separate each rectangle from the next. Each rectangle's y attribute uses the second argument of an anonymous function named **i**. Think of this as a counter which starts at zero and increments for each item in myData after the first. This counter is multiplied by the quotient of the height divided by the number of elements in the myData array which evenly spaces each rectangles y element. Since I am drawing a horizontal bar graph, the width of each rectangle will vary with the value of each peaks height in the data array. To scale the bars to the size of the svg element, I based the width of each rectangle on the ratio of the peak height to the maximum peak height times the width of the svg element. By the way, to get the maximum peak height from the dataset, I used the **d3.max** function and assigned the return value to a variable **maxHeight**.

```
var maxHeight = d3.max(myData, function(d){return d["height(feet)"];});
```

The d3.max function takes an array as the first argument and a function as the second argument to access the data items of interest. There is also a **d3.min** function that returns the minimum value from a given array.

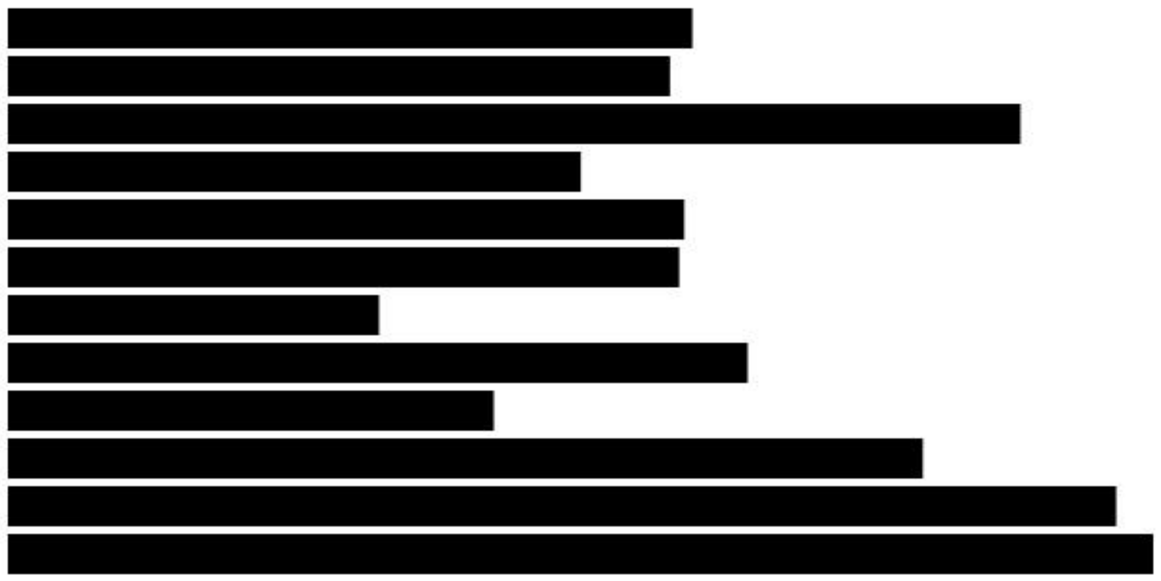


Figure 2. The width of each rectangle is proportional to the peak height written in the CSV file.

Above is the bar chart so far. Handsome, don't you think? Okay, maybe not. It could use a few extra features to help it tell a better story such as text and a set of axis. As is, the graph is not very useful.

Using D3 Scales

Before I add an axis and labels to the graph, I want to introduce another way to scale the rectangles to the SVG element. By using the **d3.scale** function the domain of the dataset can be scaled to the dimensional pixel values of the SVG element that contains the rectangles. The size of the SVG element is the range. Basically, we will let D3 do the math in normalizing the data and scaling it to a set range. Another advantage to using d3.scale is that the scale can be used when adding an axis to the graph. The graph will need two scales, one for the width and another for the height of the rectangles. In addition to scales, I will break up the code that draws the rectangles into sections of task related code. Since I am breaking the chain that constructs the rectangles, binds data and assigns attributes, I need to catch the initial rectangle selection and data binding action in a variable. I named this variable rects for obvious reasons. The four sections of code and the function they serve are:

- **Selection and Bind Data** – Selects the items to be drawn and binds data to them to be used in their creation.
- **Enter Section** – code whose resulting output does not change with data set changes
- **Update Section** – code whose resulting output varies with data set changes
- **Exit** – clean up code

The first section starts with the **selectAll** method which selects all future rectangles to be appended. This is followed by the **data** method that binds the data to these rectangles. The Enter section contains code that stays constant as the data changes. We will always append rectangles, the height of our graph will remain the same as the data changes, and the x position is always zero. The Update section contains code whose resulting output will vary as the dataset varies. The Exit section will select previous rectangles in the DOM that are not currently attached to data. It will then remove them from the DOM. This will happen for instance, if the dataset items are reduced from the previous set of items, so there should be less rectangles drawn to the screen.

```
function drawGraph(myData)
{
        var maxHeight = d3.max(myData, function(d){return d["height(feet)"];});

        //Bind the data to the rectangles to be drawn
        var rects = svg.selectAll("rect").data(myData);

        //Enter Section - items that do not vary with data changes
        rects.enter()
             .append("rect")
             .attr("height", height/myData.length - barPadding)
              .attr("x", 0);

        //Update Section - items that vary as the data changes
        rects.attr("y", function(d, i){return i * (height/myData.length);})
                        .attr("width", function(d){return  (d["height(feet)"]/maxHeight) * width});

        //Remove rectangles if necessary
        rects.exit().remove();
 }
```

Next, I created two scale objects that data can be passed to and whose return value can be used in the drawing process of the rectangles. These scale objects are actually functions since they return a value for any value passed to them. I assigned variables xScale and yScale to the scale functions to scale the width and height of each rectangle respectively. I will also use the yScale to set the y value of each rectangle. With D3, there are several different types of scales so I recommend reading up on them at the D3 website, but for this tutorial I will use the **linear** scale for the width and an **ordinal** scale for the rectangle y value and the height. A linear scale is used for a numeric succession of numbers where the difference between numbers is based on a linear function. An ordinal scale is used for an ordinal set of values that have a distinct order to them, like the letters of the alphabet or the days of the week. I will create the variables that are the scale functions outside the drawGraph function. To set the range of xScale, I use the **range** function. This function takes one parameter, which is a two element array corresponding to the beginning and ending values of the range. I set the range of xScale to go from zero to the width of the SVG element. To set the range of yScale, I use the function rangeBands which is designed to work with bar graphs. Like, range the rangeBands function takes a two element array to represent the range of the scale. This function also takes a second parameter which is the space between rectangles, or the padding.

```
var rectPadding = 0.2;
var xScale = d3.scale.linear().range( [0, width]);
var yScale = d3.scale.ordinal().rangeBands([0, height], rectPadding);
```

The domain of each scale needs to be set inside the drawGraph function since it requires access to the data which the csv function passes to drawGraph behind the scenes. Similar to the range functions, the **domain** function takes a two element array where the first value is the low end of the domain and the second value is the high end. I use the JavaScript **map** function that returns an array from a given array. In this case, map returns the array of names from the myData object array. Since this scale is an ordinal scale, it will use the names of the peaks in the order they appear in the array as the domain.

```
xScale.domain([0, d3.max(myData, function (d){ return d["height(feet)"]; })]);
yScale.domain( myData.map(function(d){return d["name"]}));
```

With the scales setup, they can be used to set the attributes of the rectangles for the graph. The second parameter to the y and width attr functions are now anonymous functions that include calls to the scale functions. The height attribute is set using yScale and the D3 **rangeBands** function.

```
<script type= "text/javascript">
        var width = 600;
        var height = 300;
        var rectPadding = 0.2;

        //Adds a SVG element in to the body of the page
        var svg = d3.select("body").append("svg").attr("width", width).attr("height", height);

        var xScale = d3.scale.linear().range( [0, width]);
        var yScale = d3.scale.ordinal().rangeBands([0, height], rectPadding);

        function drawGraph(myData)
        {
                xScale.domain([0, d3.max(myData, function (d){ return d["height(feet)"]; })]);
                yScale.domain(myData.map(function(d){return d["name"]}));

                //Bind the data to the rectangles to be drawn
                var rects = svg.selectAll("rect").data(myData);

                //Enter Section - items that do not vary with data changes
                rects.enter()
                   .append("rect")
                   .attr("height", yScale.rangeBand());

                //Update Section - items that vary as the data changes
                rects.attr("x", 0)
                        .attr("y", function(d){    return yScale(d["name"]);})
                        .attr("width", function(d){return xScale(d["height(feet)"]);});

                //Remove rectangles if necessary
                rects.exit().remove();
        }

        //Convert the strings representing the height into numeric values
        function convert(d)
        {
                d["height(feet)"] = parseFloat(d["height(feet)"]);
                return d;
        }

        d3.csv("../../csv/sfbaypeaks.csv", convert, drawGraph);
</script>
```

When setting the height attribute of each rectangle, yScale calls the ordinal rangeBands method which will base the y position of each rectangle depending on the number of elements in the input domain. In other words, D3 does all the algebraic work to calculate how tall each bar needs to be to fit in the vertical pixel space of the SVG element. Remember, when defining yScale, the bar padding was passed as a parameter, so this value is also used in deciding how tall each rectangle needs to be for the given space, but D3 does the work in figuring it out. As previously mentioned, another reason to use scales is that they are used to set the axis of a graph. That’s what I will do next.

The G Element

Now it’s time to dress up the bar graph to be more useful and have it reflect information to give the graph meaning. As is, the bar graph is using all the space inside the SVG element. To add a set of axis to the bar graph, we need to shrink the space the bar graph uses to make room for axis and text. The bar graph will be drawn in an inner sub-space of the SVG element and the text and axis will be on the outside of this inner space, but inside the bounds of the SVG element. By placing all the rectangles that make up the bar graph into a grouping element, the grouping element can be sized to fit inside the SVG element. The D3 group element reminds me of an HTML div element, since it serves as a container for context related code, but does not take up actual space.

In addition to the SVG width and height variables, we need an inner-width and inner-height for the inner grouping element. By appending a ‘g’ element to the SVG element a group element is added to the SVG. We can space the grouping by defining margin values using a margin object and translating the grouping by specific margin values.

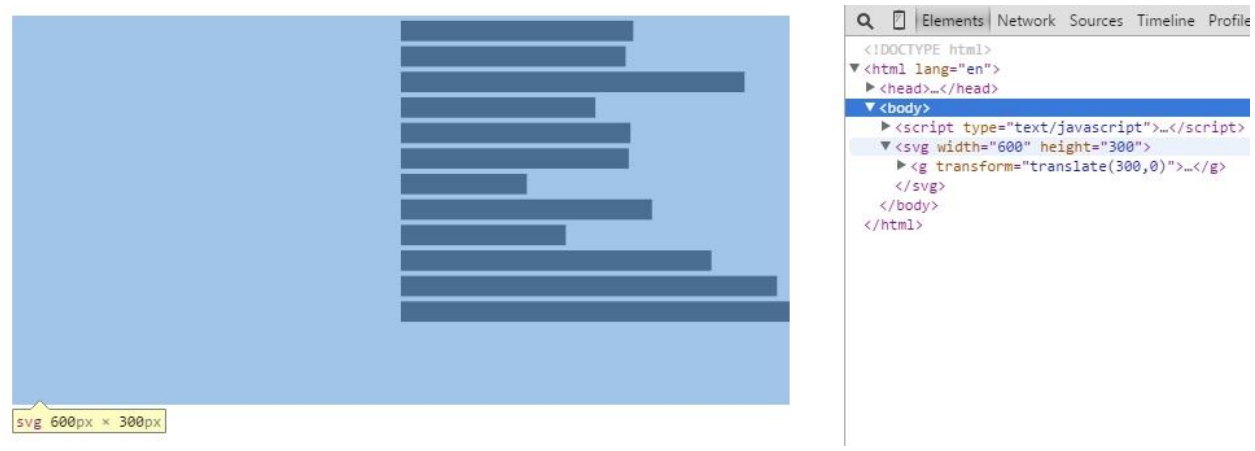
```
var margin = { left: 130, top: 0, right: 0, bottom: 60 };
var innerWidth  = width  - margin.left - margin.right;
var innerHeight = height - margin.top  - margin.bottom;
var g = svg.append("g")
            .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

When defining the scale functions, the inner-width and inner-height need to be specified since that is the area we would like the bar graph drawn inside.

```
var xScale = d3.scale.linear().range([0, innerWidth]);

var yScale = d3.scale.ordinal().rangeBands([0, innerHeight], barPadding);
```

Within the drawGraph function, the rectangles are selected and appended to the g element not the SVG element.

```
var bars = g.selectAll("rect").data(myData);
```



```
<!DOCTYPE html>
▼<html lang="en">
  ▶<head>…</head>
  ▼<body>
    ▶<script type="text/javascript">…</script>
    ▼<svg width="600" height="300">
      ▶<g transform="translate(300,0)">…</g>
      </svg>
    </body>
</html>
```

Figure 3. The rectangles are now drawn inside the g element that is contained within the SVG element. Now there is room for labels and an axis (The SVG element is highlighted in the image).

With the above mentioned changes, the rectangles are now drawn inside the grouping element that is contained inside the SVG element. As you can see there is plenty of room for the peak names to the left of the rectangles and an axis with labels indicating how many feet high they are.

Below is the complete JavaScript code for the bar graph thus far. This code is contained inside the HTML body tags of index.html.

```
<script type="text/javascript">
                var width = 600;
                var height = 300;
                var rectPadding = 0.2;
                var margin = { left: 300, top: 0, right: 0, bottom: 60 };
                var innerWidth  = width  - margin.left - margin.right;
                var innerHeight = height - margin.top  - margin.bottom;

                //Adds a SVG element in to the body of the page
                var svg = d3.select("body").append("svg").attr("width", width).attr("height", height);

                var g = svg.append("g")
                                .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

                var xScale = d3.scale.linear().range(    [0, innerWidth]);
                var yScale = d3.scale.ordinal().rangeBands([0, innerHeight], rectPadding);

                function drawGraph(myData)
                {
                        xScale.domain([0, d3.max(myData, function (d){ return d["height(feet)"]; })]);
                        yScale.domain(          myData.map(function(d){return d["name"]}));

                        //Bind the data to the rectangles to be drawn
                        var rects = g.selectAll("rect").data(myData);

                        //Enter Section - items that do not vary with data changes
                        rects.enter().append("rect")
                                .attr("height", yScale.rangeBand());

                        //Update Section - items that vary as the data changes
                        rects.attr("x", 0)
                                .attr("y", function(d){   return yScale(d["name"]);})
                                .attr("width", function(d){return xScale(d["height(feet)"]);});

                        //Remove rectangles if necessary
                        rects.exit().remove();
                }

                //Convert the strings representing the height into numeric values
                function convert(d)
                {
                        d["height(feet)"] = parseFloat(d["height(feet)"]);
                        return d;
                }
                d3.csv("../../csv/sfbaypeaks.csv", convert, drawGraph);
</script>
```

Adding an Axis and Text Labels

When adding labels and axis to a graph, it is a good idea to place them in a group element so that you have placement control via a transform and translate attribute. I created an X axis and Y axis grouping variable for the axis across the bottom of the graph that will represent the peak height and for the name of the vertical list of peak names to the left of each rectangle respectively. I've added class attributes to them so that they may be styled using CSS.

```
 var xAxisGroup = g.append("g")
                          .attr("class", "axis")
                          .attr("transform", "translate(0," + innerHeight + ")");

 var yAxisGroup = g.append("g")
                          .attr("class", "y axis");
```

To add axis to the bar graph, D3 provides a very useful **axis** function that does all the hard work in drawing an axis for a graph. The axis function requires a user created scale function so it knows how to draw the axis. This is accomplished by passing a scale function as an argument to the D3 **scale** method. In this example, I pass the xScale and yScale function variables I created in the last section. The D3 **orient** method tells D3 how to orient the text for the scale. The D3 **tick** method takes a positive integer as an argument which is the number of ticks you would like for the axis. D3 will attempt to use this value, but may use another number of ticks if it is deemed visually pleasing. The D3 **tickFormat** method allows you to override the default tick formatting. As an argument it takes the D3 **format** method that takes a string type specifier as an argument. I passed the string "f" which tells the format method to format the numbers using fixed point notation. I recommend reading about the other type formatters that can be used in the D3 API documentation. As an example, you can use the string "x" to have the numbers displayed in hexadecimal form. By passing a value of zero as an argument to the D3 **outerTickSize** method, the axis lines are removed to make the graph visually pleasing.

```
var xAxis = d3.svg.axis().scale(xScale)
                              .orient("bottom")
                              .ticks(5)
                              .tickFormat(d3.format("f"))
                              .outerTickSize(0);

var yAxis = d3.svg.axis().scale(yScale)
                               .orient("left")
                               .outerTickSize(0);
```

To get the axis drawn into the grouping element, the D3 **call** method must be used which takes a scale as an argument. Do this inside the drawGraph function.

```
xAxisGroup.call(xAxis);
yAxisGroup.call(yAxis);
```

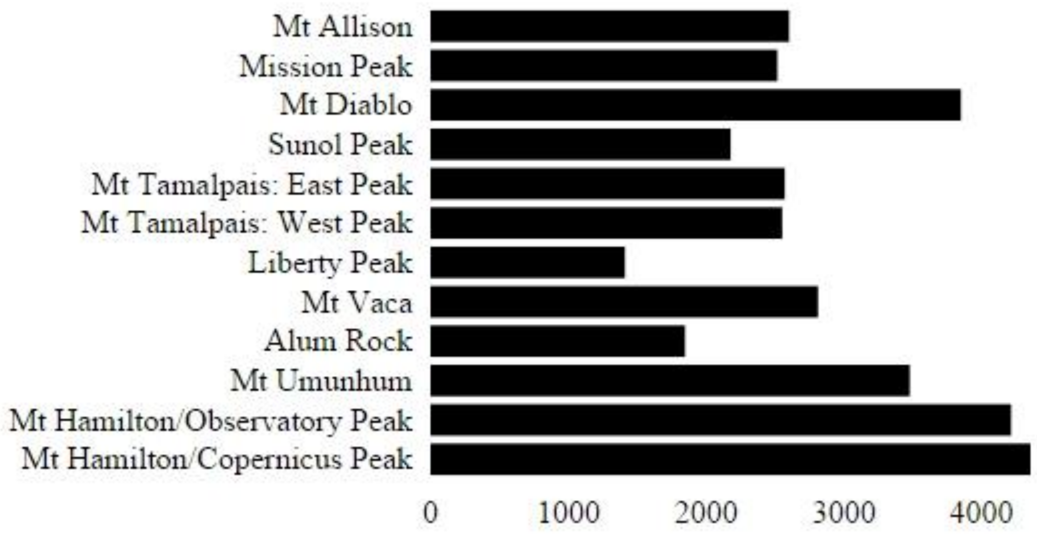


Figure 4.Almost completed graph. Just need to add a text element that specifies the units along the X axis.

Now the bar graph is almost complete. Each rectangle is clearly labeled with the name of the peak and along the X axis there is a scale which represents the name of each peak in feet. The only thing left is to add text at the bottom stating what unit the numbers represent. Next, we will add a label stating that the numbers are in feet, not meters or miles.

The complete JavaScript code with most of the comments removed so that it may fit on a single page.

```
<script type="text/javascript">

            var width = 600;
            var height = 300;
            var rectPadding = 0.2;
            var margin = { left: 300, top: 0, right: 0, bottom: 60 };
            var innerWidth  = width  - margin.left - margin.right;
            var innerHeight = height - margin.top  - margin.bottom;

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

            var g = svg.append("g")
                            .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

            var xAxisGroup = g.append("g").attr("class", "axis")
                                                            .attr("transform", "translate(0," + innerHeight + ")");

            var yAxisGroup = g.append("g").attr("class", "y axis");

            var xScale = d3.scale.linear().range( [0, innerWidth]);
            var yScale = d3.scale.ordinal().rangeBands([0, innerHeight], rectPadding);

            var xAxis = d3.svg.axis().scale(xScale).orient("bottom")
                                                          .ticks(5)
                                                          .tickFormat(d3.format("f"))
                                                          .outerTickSize(0);

             var yAxis = d3.svg.axis().scale(yScale).orient("left")
                                                           .outerTickSize(0);

function drawGraph(myData)
{
            xScale.domain([0, d3.max(myData, function (d){ return d["height(feet)"]; })]);
            yScale.domain( myData.map(function(d){return d["name"]}));

            xAxisGroup.call(xAxis);
            yAxisGroup.call(yAxis);

            var rects = g.selectAll("rect").data(myData);

            rects.enter().append("rect").attr("height", yScale.rangeBand());

            rects.attr("x", 0)
                    .attr("y", function(d){    return yScale(d["name"]);})
                    .attr("width", function(d){return xScale(d["height(feet)"]);});

            rects.exit().remove();
}

//Convert the strings representing the height into numeric values
function convert(d)
{
            d["height(feet)"] = parseFloat(d["height(feet)"]);
            return d;
}

d3.csv("../../csv/sfbaypeaks.csv", convert, drawGraph);

</script>
```

A units label can easily be added to the X axis by appending a text element to the xAxisGroup. Do this outside the drawGraph function.

```
var xAxisLabel = xAxisGroup.append("text")
                                   .style("text-anchor", "middle")
                                        .attr("x", innerWidth / 2)
                                        .attr("y", xAxisLabelOffset)
                                        .attr("class", "label")
                                        .text("Feet");
```

And below is the finished bar graph. It's quite handsome, if I do say so myself.

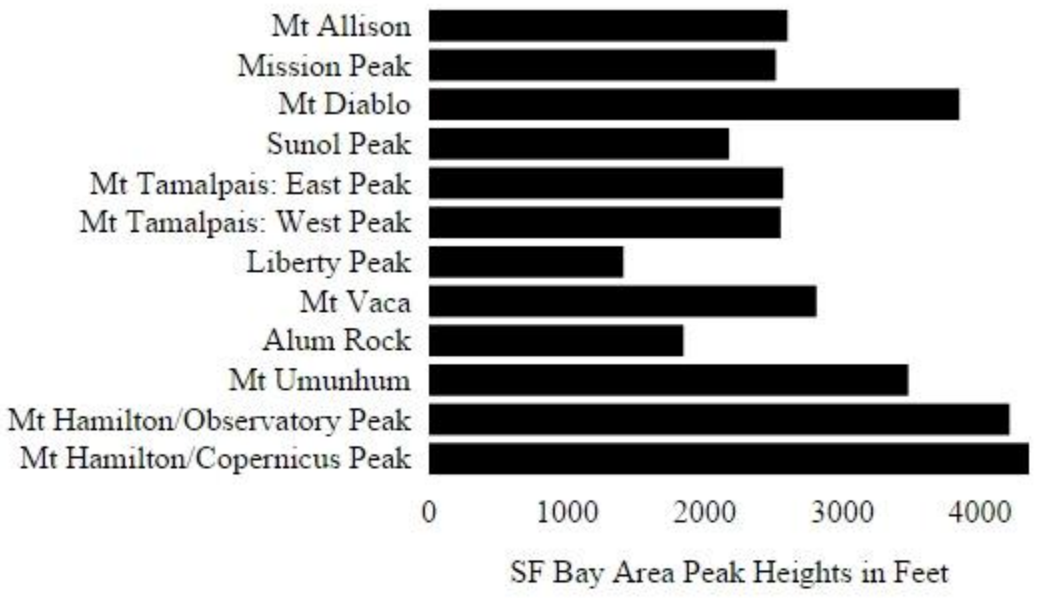


Figure 5. The completed bar graph showing the height of the peaks in feet around the SF Bay Area.

This tutorial only touches the surface of what D3 is capable of. I'm sure you will agree that D3 makes it pretty easy to add bar graphs to HTML pages. By the way, to convert the above bar graph into a vertical bar graph, just simply swap the x and y variables and the width and height variables. It's as simple as that. The Horizontal bar graph works better though for graphs with many bars which are labeled with long text strings.