# Data Visualization Practical

In this practical you will utilise the D3 (Data Driven Documents) JavaScript library to generate a bar graph of power outages across NSW over the past several years. The example given here is quite complex due to the use of grouping elements that allow bars (svg rectangles) and text elements to be co-located on the visualization – an approach that is encouraged because it reduces the likelihood of related elements being misplaced or handled incorrectly when generating the graph. The graph itself is also placed into a group so that margins can be created to make space for axes.

Contents

[Task 1. Download Notepad++ text editor 1](#_Toc496041984)

[Task 2. Create a HTML Page 1](#_Toc496041985)

[Task 3. Create a link to the D3 JavaScript Library 1](#_Toc496041986)

[Task 4. Incorporating some JavaScript 1](#_Toc496041987)

[Task 5. Calculating important values 2](#_Toc496041988)

[Task 6. Adding the SVG element 2](#_Toc496041989)

[Task 7. Selecting SVG Groups 3](#_Toc496041990)

[Task 8. Adding New Groups 3](#_Toc496041991)

[Task 9. Adding New Rectangles 4](#_Toc496041992)

[Task 10. Creating a Y-axis Linear Scale 5](#_Toc496041993)

[Task 11. Creating a y-Axis 6](#_Toc496041994)

[Task 12. Providing space for the Axes 6](#_Toc496041995)

[Task 13. Adding Text to the bars 8](#_Toc496041996)

[Your Tasks 9](#_Toc496041997)

[Task 14. Using an example JSON data source 9](#_Toc496041998)

[Task 15. Creating an X-scale and X-axis 9](#_Toc496041999)

[Complete Solution Tasks 1-13 11](#_Toc496042000)

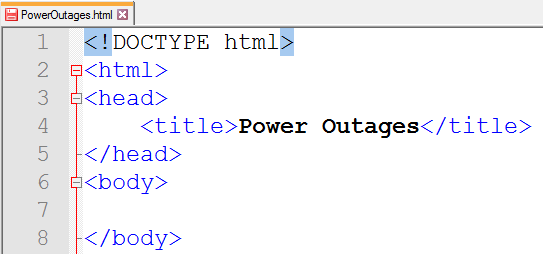
[Complete Solution Including Tasks 14, 15 13](#_Toc496042001)

1. Download Notepad++ text editor

* Notepad++ is a general text editor with syntax highlighting for a number of languages including HTML and JavaScript. You can use any editor, but this one may help you to find errors.
* <https://notepad-plus-plus.org/download/v7.5.1.html>

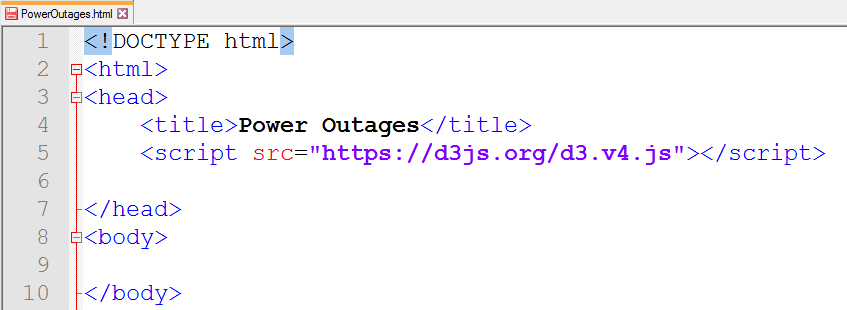
1. Create a HTML Page

* Open Notepad++ and create a new html page complete with the HTML5 doctype, head, title and body elements.
* Save the page as PowerOutages.html so that the notepad editor starts highlighting HTML and JavaScript syntax.



1. Create a link to the D3 JavaScript Library

* In the head element of your page, create a link to the D3 Version 4 JavaScript library hosted at D3.org (<https://d3js.org/>). You can use the link provided below which is taken from the D3.org home page:
* <script src="https://d3js.org/d3.v4.js"></script>



1. Incorporating some JavaScript

* Next create a script element in the body of your HTML document. **Everything we change from now on will be within these script tags.**

<script>

var dataset = [871, 2022, 1538, 1663, 1166];

</script>

* Declare a new variable called “dataset” and assign it the following array of values

[871, 2022, 1538, 1663, 1166 ];

1. Calculating important values

* A number of important boundary values need to be calculated. These include:
  1. Min and max data values
  2. SVG dimensions
  3. Bar dimensions
* After the dataset array, create JavaScript variables named min and max, then use the d3 min() and max() functions to calculate their values. Note that you could also use the extent function. We now have two variables containing the minimum value and the maximum value in our dataset. We could of course just have put in 871 and 2022 but that would have meant every time we change the dataset we have to manually calculate the values.
* Create JavaScript variables for height and width, set their values to 500. These will be used to set the dimensions of the SVG element. Similarly we now have variables to store these values for use later.
* Create two more JavaScript variables called barMargin and barWidth. Set the value of barMargin to 10, and the value of barWidth to the maximum possible size that will support all of the dataset values in the available SVG width.

// important values

var max = d3.max(dataset, function (d, i) { return d; });

var min = d3.min(dataset, function (d, i) { return d; });

var dataRange = d3.extent(dataset, function (d, i) { return d; });

// set SVG dimensions

var height = 500;

var width = 500;

// set bar dimensions

var barMargin = 10;

var barWidth = width / dataset.length;

1. Adding the SVG element

* Below our above variables create a JavaScript variable called **svg** and use d3 to select the body element, append a figure element to the body and then an svg element to the figure element
* Set the height and width of the SVG to their corresponding JavaScript variables. Feel free to add a border to the SVG element if you want to see the graphing space
* This code looks complicated but basically……does exactly that. Each time you append, the code selected the new appended element and applies the next function/style to the new element:

var svg = d3.select('body')

.append('figure')

.append('svg')

.attr('width', width)

.attr('height', height)

.style('border', '1px solid black');

1. Selecting SVG Groups

Remember that when using SVG if you want each of your bars to have associated text values the text is best co-located with the bars by using an SVG group to position both elements. So what we are doing grouping the bar and the text associated with that bar.

* Create a new JavaScript variable called **barGroups** and then assign this all the “g” elements in the svg selection. Remember that the “g” element in SVG means a group of things.
* Load the dataset in to the bars using .data().
* This is one of the most important steps because at this point the data visualization is actually created. Up until now we have some data and an un-related SVG element that may or may not contain any existing groups. Calling this function will allow D3 to map each data value to a group in the SVG element on the page and work out if it needs to:
  1. Append more groups (which it does in this case because we have none)
  2. Update existing groups
  3. Remove surplus groups from a previous visualization

You can read more about this process which is called “Binding Data” on [Scott Murrays D3 tutorial page](http://alignedleft.com/tutorials/d3/binding-data)

var barGroups = svg

.selectAll('g')

.data(dataset);

1. Adding New Groups

* Next create a new statement using barGroups to enter() in new svg group elements and append() them to the page. Give these new bar groups the name **newBarGroups**. This step is also critical because the enter() function allows D3 to select all the data values from the data array that don’t currently have groups (our bar groups) associated with them. If we wanted to select any surplus bars from a previous visualization then we would use the exist() function instead. And just in case you are wondering, if you wanted to update existing bars – then you simply leave out the enter() or exit() function call and you have the existing bar groups from a previous visualization.
* The next step is rather difficult because SVG group elements do not have x and y attributes. Instead we must use the transform attribute to translate(x,y) the position of the group elements. It’s more difficult to work with now, but it will save us having to set the x and y attributes of all the bar elements later on!
  1. The x position will need to be the current index \* barWidth
  2. The y position will need to be height – (current data value/max)\*height

var newBarGroups = barGroups.enter()

.append('g')

.attr('transform', function (d, i) {

return 'translate(' // translate(x,y)

+ (i \* barWidth) // (x

+ ',' // ,

+ (height - (d / max) \* height) // y)

+ ')';

}); // end transform

Open your page in a web browser, right click on the page and select “**Inspect Element**” to view the generated HTML by the above code:



Note how the highest value is now represented by y=”0” as this element will be the full height of the SVG element. If additional space is needed in the graph then this would need to be considered.

1. Adding New Rectangles

Since we are using bar groups to contain the text and the rectangles in this bar graph, the groups must be selected in order to append a rectangle to each group that corresponds with a data value.

* Using the **newBarGroups**, append() new rectangles
* Set the x position to 0 (note this is optional because the bar groups are positioning the bars)
* Set the height of the bars based on the height - current data value / max
* Set the width of the bars based on barWidth – margin
* Set the fill of the bars to black

newBarGroups

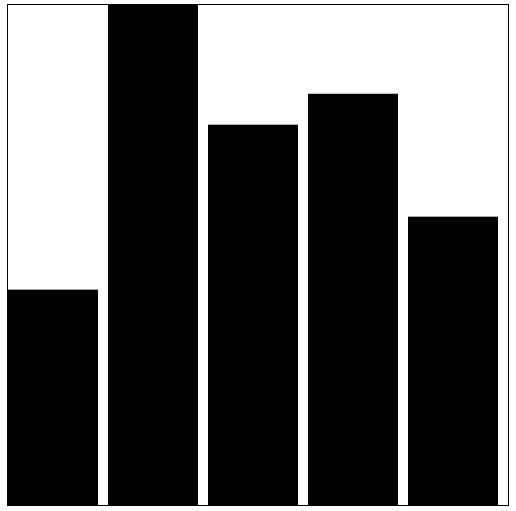
.append('rect')

.attr('x', 0) // optional line that can be left out since 0 is the default

.attr('height', function (d, i) { return d / max \* height; })

.attr('width', barWidth - barMargin)  
 .attr('fill', 'black');

Next view your file in the web browser and see if you get the following graph



1. Creating a Y-axis Linear Scale

So far all your y-coordinate calculations have been manual. D3 includes several functions that can be used to automatically scale the bar sizes and simplify the calculations that currently use the SVG height and max data values.

* Underneath the svg element declaration, create a new yScale variable
* Set it equal to a d3.scaleLinear() function
* Set the data range (domain) to be between 0 and the max value
* Set the output range (range) to be between height and 0 (it is reversed because we want the graph to grow from the bottom up and not top down).

// Task 6

// Adding the SVG element

var svg = d3.select('body')

.append('figure')

.append('svg')

… etc

// Task 10

// Create a YScale

var yScale = d3.scaleLinear()

.domain([0, max])

.range([height,0]);

* The next step is to use the linear scale to simplify some of the y and height calculations.  
  Replace the y-calculation of the bars group to use the new yScale function

var newBarGroups = barGroups.enter()

.append('g')

.attr('transform', function (d, i) {

return 'translate('

+ (i \* barWidth)

+ ','

+ (height - (d / max) \* height)

+ ')';

}

);

var newBarGroups = barGroups.enter()

.append('g')

.attr('transform', function (d, i) {

return 'translate('

+ (i \* barWidth)

+ ','

**+ (yScale(d))**

+ ')';

}

);

* Next replace the height calculation of the rectangles with the yScale. This time it will have to be inverted because the range of the yScale was entered upside down.

newBarGroups

.append('rect')

.attr('x', 0)

.attr('height', function (d, i) {  
 return d / max \* height;\*

})

.attr('width', barWidth - barMargin)  
 .attr('fill', 'black');

newBarGroups

.append('rect')

.attr('x', 0)

.attr('height', function (d, i) {  
 **return height - yScale(d);**

})

.attr('width', barWidth - barMargin)  
 .attr('fill', 'black');

1. Creating a y-Axis

The next step is to create a y-Axis with labels so that we can see what the bars values represent.

* At the end of all the JavaScript statements after the bars have been appended to the SVG element, add the following code which will automatically create groups and paths that make up the y-Axis.
* You can try the different axis arrangements to see how they position the ticks differenty:
  + axistLeft(), axisBottom(), axisTop()

newBarGroups

.append('rect')

.attr('x', 0)

.attr('height', function (d, i) { return height - yScale(d); })

.attr('width', barWidth - barMargin)

.attr('fill', 'black');

**// create the axis and add them to the page**

**var yAxis = d3.axisLeft(yScale); // declare the axis generator**

**svg.append('g')**

**.classed('axis y', true)**

**.call(yAxis);**

* Ideally we want the axis on the Left – but if you try this the numbers will disappear. This is because the graph needs to allocate space for any axes. This can be done by creating a margin property and utilising it in all the graph calculations.
* Set the yAxis back to axisLeft()

1. Providing space for the Axes

* To accommodate the various axes on the graph we need to reserve some space around the visualization and this space is refer to as a margin – the distance between the graph body and the SVG perimeter. The size of the margins will need to be calculated with a bit of trial and error because the size will depend on the space required by the axes and the size of the font on the various axis scales.
* Under the height and width variables create a new JavaScript object called chartMargins with the following properties
  + left: 40; right: 25; top: 25; bottom:40
  + The numbers can be changed later as necessary.

// set SVG dimensions

var height = 500;

var width = 500;

var chartMargin = new Object();

chartMargin.left = 40;

chartMargin.right = 25;

chartMargin.top = 25;

chartMargin.bottom = 40;

* Next, update the barWidth calculation to account for the left and right margins the graph may require

// set bar dimensions

var barMargin = 10;

var barWidth = (width - chartMargin.left - chartMargin.right) / dataset.length;

* Next the range of values needs to be updated to reflect the reduction in vertical space available on the SVG element. Rather than having the full height available, the maximum size of a bar will be the height minus the chartMargin top and bottom values! This can be achieved by putting a calculation inside the range yScale range function or you can re-calculate the height before it is used in the range function. The latter is shown below as it doesn’t interfere with the SVG height which has previously been set:

// Task 6

// Adding the SVG element

var svg = d3.select('body')

.append('figure')

.append('svg')

.attr('width', width)

.attr('height', height)

. . . etc

// Task 12

// reduce the available SVG height by taking into account top and bottom margins

**height = height - chartMargin.bottom - chartMargin.top;**

* In addition to reducing the height of the range, the “zero” position or top most point from which the SVG graph gets drawn must also be moved. This means rather than drawing all the SVG elements from y=0 we need to move it down from the top of the SVG element by chartMargin.top:

// Task 12

// reduce the available SVG height by taking into account top and bottom margins

**height = height - chartMargin.bottom - chartMargin.top;**

// Task 10

// Creating a Y-axis Linear Scale

var yScale = d3.scaleLinear()

.domain([0, max])

**.range([height, chartMargin.top]); // Task 12 Modification**

* Next, we need to make a group that can be positioned using the new margin information. The graph and axes can then be placed into this group instead of the SVG directly.
* Locate the section of code that creates the SVG element.
  + Create a new variable called chart group and make it equal a new group element within the current SVG element
  + Give it the class name ‘chartGroup’
* Transform the group by setting the x,y translate properties to chartMargin.left and chartMargin.top

var chartGroup = svg.append('g')

.classed('chartGroup', true)

.attr('transform', 'translate(' + chartMargin.left + ',' + chartMargin.top+ ')');

Next, for each of the bar group elements and axes append them to the new chart group element instead of directly to the SVG element:

var barGroups = svg

.selectAll('g')

.data(dataset);

var barGroups = **chartGroup**

.selectAll('g')

.data(dataset);

var yAxis = d3.axisLeft(yScale);

svg.append('g')

.classed('axis y', true)

.call(yAxis);

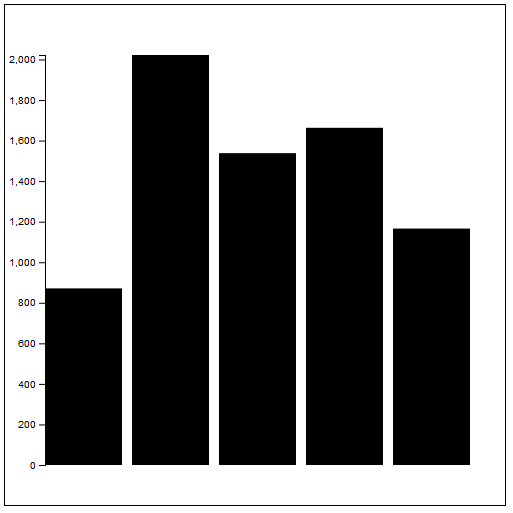
var yAxis = d3.axisLeft(yScale);

**chartGroup.append('g')**

**.classed('axis y', true)**

**.call(yAxis);**

If you run the graph it should now have the y-Axis clearly visible on the left hand side. Note here that the left margin of 25 wasn’t quite enough so feel free to increase the chartMargin.left value.



1. Adding Text to the bars

As you know this one will be a little more difficult. The bars are part of a group and the text element must be added to the bar group and positioned accordingly.

* If you want your text positioned above each bar, then you will need to ensure the graph is not full height to allow for enough pixel space to position the text. This could be achieved by limiting the range property on the yScale
* If you want your text positioned inside the bar (ie over the bar) then there is no issue so long as you append the text elements after the rectangles.

For this example, we will append the text over the bars for simplicity but feel free to try both approaches.

* After the JavaScript where rectangles are created and just before adding the yAxis start a new selection based on the “newBarGroups” that appends a text element
  + Set the x attribute to 50
  + Set the y attribute to 20
  + Set the fill style/attribute to white (so the text shows up on the black bar background)
  + Set the text() to the current data value for each text element
  + Set the font-size style to 1.1em or slightly larger

**newBarGroups**

**.append('text')**

**.attr('x', 25)**

**.attr('y', 20)**

**.attr('fill', 'white')**

**.text(function (d, i) { return d; })**

**.style('font-size', '1.1em');**

var yAxis = d3.axisLeft(yScale);

chartGroup

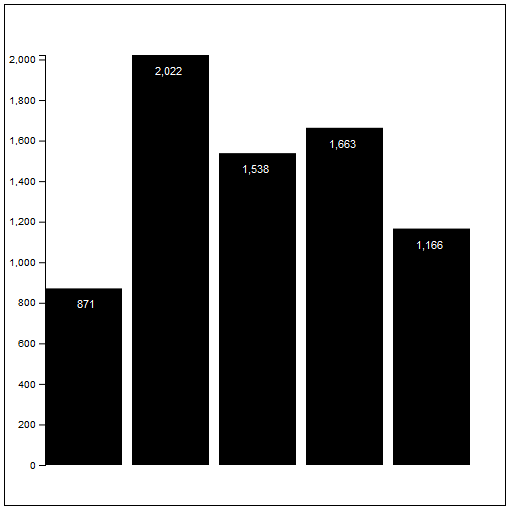
.append('g')

.classed('axis y', true)

.call(yAxis);

If everything is working well, then your graph should now have detailed information on each bar. You can also take this opportunity to format the text if necessary by using functions like toLocaleString()

.text(function (d, i) { return d.toLocaleString(); })



# Your Tasks

1. Using an example JSON data source

The data provided to create the above graph is only part of the total data. These counts represent the number of power outages across NSW for each year starting 2012.

The real dataset should be declared as follows:

var dataset = [

{ year: 2012, events: 871 },

{ year: 2013, events: 2022 },

{ year: 2014, events: 1538 },

{ year: 2015, events: 1663 },

{ year: 2016, events: 1166 }

];

This means that each D3 JavaScript function that accesses the data in the array now represents an object and not a basic integer value.

1. Update your graph code to get the event information from the array data (ie change everything that uses return d to read return d.events);
2. Fix up the yAxis and all functions to reflect these changes.
3. Creating an X-scale and X-axis
4. Research how to apply an scaleBand() and xAxis so that the year or each bar can be added along the bottom of the graph – you will need to use a new group and translate it’s position!
5. Make sure you reserve space at the bottom of the graph by making use of the chartMargin.bottom/top values

**On the next page is the complete code for the practical. However, please avoid the temptation of just looking at this if you get stuck. Try to figure it out by using other references first.**

# Complete Solution Tasks 1-13

<script>

// Task 4:

var dataset = [871, 2022, 1538, 1663, 1166];

// Task 5

// important values

var max = d3.max(dataset, function (d, i) { return d; });

var min = d3.min(dataset, function (d, i) { return d; });

var dataRange = d3.extent(dataset, function (d, i) { return d; });

// set SVG dimensions

var height = 500;

var width = 500;

// Task 12

// Providing space for Axes

var chartMargin = new Object();

chartMargin.left = 40;

chartMargin.right = 25;

chartMargin.top = 25;

chartMargin.bottom = 40;

// Task 5

// Calculating important values

// set bar dimensions

var barMargin = 10;

// Task 12 modification

var barWidth = (width - chartMargin.left - chartMargin.right) / dataset.length;

// Task 6

// Adding the SVG element

var svg = d3.select('body')

.append('figure')

.append('svg')

.attr('width', width)

.attr('height', height)

.style('border', '1px solid black');

// Task 12

// reduce the available SVG height by taking into account top and bottom margins

height = height - chartMargin.bottom - chartMargin.top;

// Task 10

// Creating a Y-axis Linear Scale

var yScale = d3.scaleLinear()

.domain([0, max])

.range([height, chartMargin.top]); // Task 12 Modification

// Task 12

// Creating a chartGroup for the margins

var chartGroup = svg.append('g')

.classed('chartGroup', true)

.attr('transform', 'translate(' + chartMargin.left + ',' + chartMargin.top + ')');

// Task 7

// Selecting SVG Groups

var barGroups = chartGroup // Task 12 modification

.selectAll('g')

.data(dataset);

// Task 8 Adding New Groups

var newBarGroups = barGroups.enter()

.append('g')

.attr('transform', function (d, i) {

return 'translate('

+ (i \* barWidth)

+ ','

+ (yScale(d)) // Task 10 modification

// + (height - (d / max) \* height)

+ ')';

}

);

// Task 9

// Adding New Rectangles

newBarGroups

.append('rect')

.attr('x', 0)

//.attr('height', function (d, i) { return d / max \* height; })

.attr('height', function (d, i) { return height - yScale(d); }) // Task 10 modification

.attr('width', barWidth - barMargin)

.attr('fill', 'black');

// Task 13

// Adding Text to the bars

newBarGroups

.append('text')

.attr('x', 50)

.attr('y', 20)

.attr('fill', 'white')

.text(function (d, i) { return d.toLocaleString(); })

.style('font-size', '1.1em');

// Task 11

// Creating a y-Axis

var yAxis = d3.axisLeft(yScale); // declare the axis generator

chartGroup.call(yAxis); // Task 12 modification

# Complete Solution Including Tasks 14, 15

**// Task 14**

**// new JSON like dataset**

**var dataset = [**

**{ year: 2012, events: 871 },**

**{ year: 2013, events: 2022 },**

**{ year: 2014, events: 1538 },**

**{ year: 2015, events: 1663 },**

**{ year: 2016, events: 1166 }**

**];**

// Task 5

// important values

**var max = d3.max(dataset, function (d, i) { return d.events; }); // Task 14 modification**

**var min = d3.min(dataset, function (d, i) { return d.events; }); // Task 14 modification**

**var dataRange = d3.extent(dataset, function (d, i) { return d.events; }); // Task 14 modification**

// set SVG dimensions

var height = 500;

var width = 500;

// Task 12

// Providing space for Axes

var chartMargin = new Object();

chartMargin.left = 40;

chartMargin.right = 25;

chartMargin.top = 25;

chartMargin.bottom = 40;

**// Task 5**

**// These calculations are replaced by the xScale scaleBand() function**

**// Calculating important values**

**// set bar dimensions**

**//var barMargin = 10;**

**// Task 12 modification**

**//var barWidth = (width - chartMargin.left - chartMargin.right) / dataset.length;**

// Task 6

// Adding the SVG element

var svg = d3.select('body')

.append('figure')

.append('svg')

.attr('width', width)

.attr('height', height)

.style('border', '1px solid black');

// Task 12

// reduce the available SVG height by taking into account top and bottom margins

height = height - chartMargin.bottom - chartMargin.top;

// Task 10

// Creating a Y-axis Linear Scale

var yScale = d3.scaleLinear()

.domain([0, max])

.range([height, chartMargin.top]); // Task 12 Modification

**// Task 15**

**// Creating an X-axis ScaleBand (bar width calculator)**

**// as with the height you have the option of re-calculating the width to include the margins**

**// or do the calculation in the range function**

**width = width - chartMargin.left - chartMargin.right;**

**// Task 15 x-Scale:**

**var xScale = d3.scaleBand()**

**.domain([2012, 2013, 2014, 2015, 2016]) // or use map function (see below)**

**.range([0, width]) // or do chartMargin calculations in here (see below)**

**.paddingInner(0.15); // margin between bars**

**// alternative example:**

**//xScale = d3.scaleBand()**

**//.domain(dataset.map(function (d) { return d.year; })) // get years using map function**

**//.range([0, width - chartMargin.left - chartMargin.right])**

**//.paddingInner(0.15); // margin between bars**

// Task 12

// Creating a chartGroup for the margins

var chartGroup = svg.append('g')

.classed('chartGroup', true)

.attr('transform', 'translate(' + chartMargin.left + ',' + chartMargin.top + ')');

// Task 7

// Selecting SVG Groups

var barGroups = chartGroup // Task 12 modification

.selectAll('g')

.data(dataset);

// Task 8 Adding New Groups

var newBarGroups = barGroups.enter()

.append('g')

.attr('transform', function (d, i) {

return 'translate('

**+ (xScale(d.year)) // Task 15 modification**

+ ','

**+ (yScale(d.events)) // Task 10 AND Task 14 modification**

+ ')';

});

// Task 9

// Adding New Rectangles

newBarGroups

.append('rect')

.attr('x', 0)

**.attr('height', function (d, i) { return height - yScale(d.events); }) // Task 10,14 modification**

**.attr('width', xScale.bandwidth()) // Task 15 modification**

.attr('fill', 'black');

// Task 13

// Adding Text to the bars

newBarGroups

.append('text')

.attr('x', 50)

.attr('y', 20)

.attr('fill', 'white')

**.text(function (d, i) { return d.events.toLocaleString(); }) // Task 14 modification**

.style('font-size', '1.1em');

// Task 11

// Creating a y-Axis

var yAxis = d3.axisLeft(yScale); // declare the axis generator

chartGroup.call(yAxis); // Task 12 modification

// Task 15

// Creating an x-Axis

// xScale change

var xAxis = d3.axisBottom(xScale); // declare the axis generator

chartGroup.append('g')

.attr('transform', 'translate(0,' + height + ')')

.classed('axis x', true)

.call(xAxis);