2-3, Lesson 2 - Drawing a Coordinated Visualization

2.1 Responsively Framing a Data Visualization

For the D3 lab assignment, you are required to create a **coordinated data visualization**, or a graphic to visually communicate information about the data that may not be as easily visible on the choropleth map. The coordinated visualization must be **linked** to the map, such that it shows a different view of the same data and reacts to user *retrieve* interactions with the map.

In this tutorial, we will create a simple bar chart as our coordinated visualization. However, you should not feel limited to the bar chart as your only option. If you are feeling adventurous and want to try implementing a different type of visualization, revisit the D3 Examples Gallery (https://github.com/mbostock/d3/wiki/Gallery) for inspiration. Any type of coordinated visualization will count equally toward your grade on the assignment. If you do decide to stick with a bar chart, make sure you customize its look and feel. Do *not*simply use the default styles shown in this tutorial. You may copy-paste example code, but copycatting the visual appearance of the map or chart will result in a deduction of points from your assignment grade.

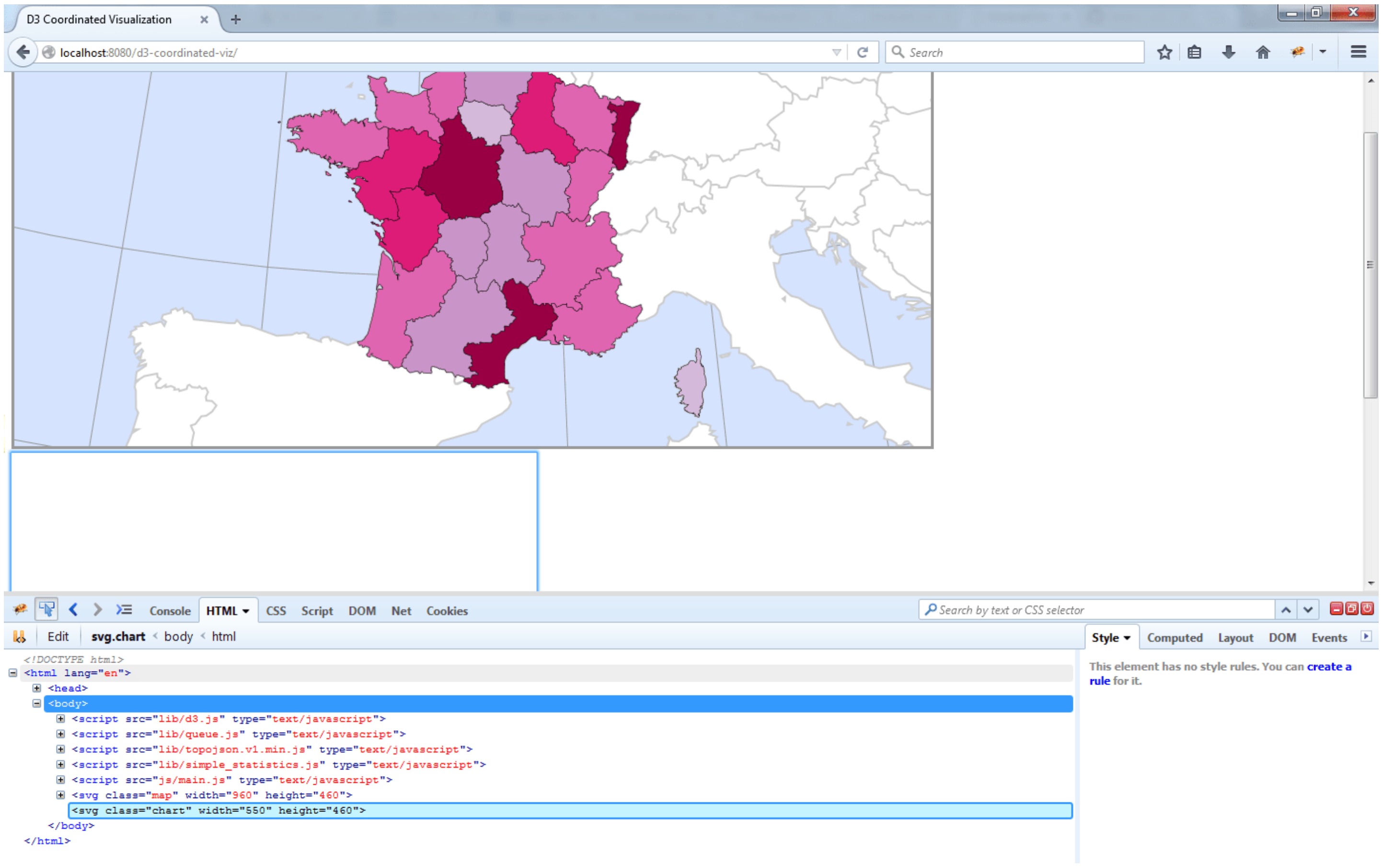
The first step in creating our linked visualization is to build the chart container in *main.js*. We can do this in a new function called from within the `callback()` function (Example 2.1):

Example 2.1: Creating the bar chart container in *main.js*

JavaScript

```
        //Example 1.4 line 4...add enumeration units to the map
         setEnumerationUnits(franceRegions, map, path, colorScale);

         //add coordinated visualization to the map
         setChart(csvData, colorScale);
    };
}; //end of setMap()

//...

//function to create coordinated bar chart
function setChart(csvData, colorScale){
    //chart frame dimensions
    var chartWidth = 550,
        chartHeight = 460;

    //create a second svg element to hold the bar chart
    var chart = d3.select("body")
        .append("svg")
        .attr("width", chartWidth)
        .attr("height", chartHeight)
        .attr("class", "chart");
};
```

In Example 2.1, we anticipate that we will eventually need the `csvData` and the `colorScale` to draw and color the bars, so we pass those variables as parameters to our new `setChart()` function (lines 5, 12). Within the `setChart()` function, we set a width and height for the chart (lines 14-15) and build its `<svg>` container using a `chart` block (lines 18-22). If we use the Inspector, we can see our chart container on the browser page (Figure 2.1):


D3 Coordinated Visualization
localhost:8080/d3-coordinated-viz/
Search
Console
HTML
CSS
Script
DOM
Net
Cookies
Search by text or CSS selector
Edit
svg.chart
body
html
Style
Computed
Layout
DOM
Events

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <body>
    <script src="lib/d3.js" type="text/javascript">
    <script src="lib/queue.js" type="text/javascript">
    <script src="lib/topojson.v1.min.js" type="text/javascript">
    <script src="lib/simple_statistics.js" type="text/javascript">
    <script src="js/main.js" type="text/javascript">
    <svg class="map" width="960" height="460">
    <svg class="chart" width="550" height="460">
  </body>
</html>
```

This element has no style rules. You can create a rule for it.

Figure 2.1: The bar chart container viewed with the Inspector

Obviously, it is poor UI design to have our chart appear immediately below our map on the page. Much of the utility of a coordinated visualization is in the ability of the users to see the entirety of both the map and visualization at the same time so as to compare the two. Thus, our map will have to become smaller so that the chart can fit next to it. While we could simply adjust the map `width` variable with a guess as to how wide the map should be, it is better to use some principles of [responsive web design (http://www.alistapart.com/articles/responsive-web-design/)](http://www.alistapart.com/articles/responsive-web-design/) that you learned in Geography 572. We can make the widths of the chart and map responsive to each other by setting each to a fraction of the browser window's `innerWidth` property, which reflects the internal width of the browser frame (Example 2.2):

Example 2.2: Setting responsive map and chart widths in *main.js*

JavaScript

```
//Example 1.3 line 2...set up choropleth map
function setMap(){
    //map frame dimensions
    var width = window.innerWidth * 0.5,
        height = 460;

//...

//Example 2.1 line 11...function to create coordinated bar chart
function setChart(csvData, colorScale){
    //chart frame dimensions
    var chartWidth = window.innerWidth * 0.425,
        chartHeight = 460;
```

In Example 1.3, the map frame width is set to 50% of the `window.innerWidth` property (line 4) and the chart frame width is set to 42.5% (line 12). The 7.5% gap between the two frames will leave space for a margin on either side of the page and ensure a break point (the window width at which the chart falls below the map) that is in between common device display sizes. To make it easier to see our chart frame and fine-tune the appearance of the two frames, we can add some styles in *style.css* (Example 2.3):

Example 2.3: Adding a map frame margin and chart frame styles in *style.css*

CSS

```
.map {
    border: medium solid #999;
    margin: 10px 0 0 20px;
}

.chart {
    background-color: rgba(128,128,128,.2);
    border: medium solid #999;
    float: right;
    margin: 10px 20px 0 0;
}
```

In Example 2.3, we add a 10-pixel top margin and 20-pixel left margin to the map frame (line 3) and a 10-pixel top margin and 20-pixel right margin to the chart frame (line 10). We also add a chart background color and border and make it adhere to the right side of the page, rather than abut the map frame (lines 7-9). Here are the results in the browser (Figure 2.2):

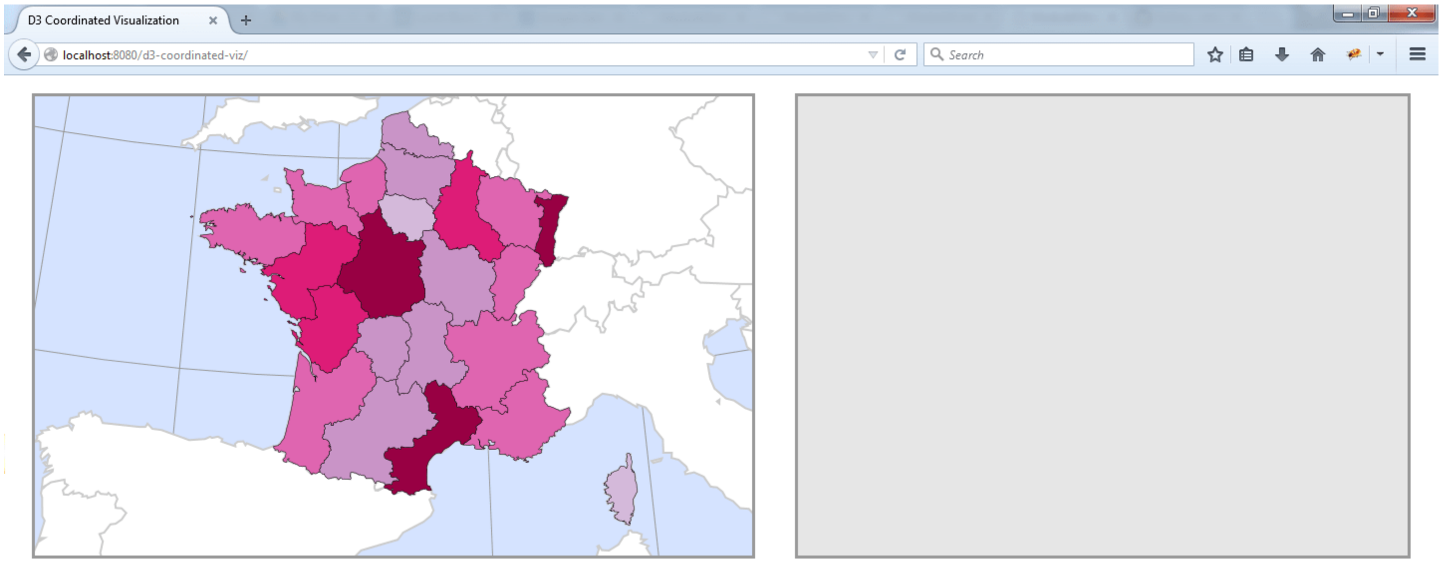

D3 Coordinated Visualization
localhost:8080/d3-coordinated-viz/
Search

Figure 2.2: Even, responsive map and chart frames

If you try to resize your browser window, you will find that the frames are only "responsive" if the page is reloaded. Eventually, we can use some event listeners to enable dynamic frame adjustment any time the window is resized. Next Module will cover how to do this. For now, we will move on to drawing our bars in the bar chart.

> **Practice!** Add an SVG container for your data visualization and adjust your map container size so that both fit neatly on the web page for a wide range of browser window sizes.

2.2 Adding Bars

To make our bars, we need to build a new `.selectAll()` block that appends a rectangle to the chart container for each feature in the dataset, positions it, and sizes it according to its attribute value. The `<rect>` (https://developer.mozilla.org/en-US/docs/Web/SVG/Element/rect) element is used to create rectangles in SVG graphics. To draw the bars, we will use four attributes of `<rect>`: `width`, `height`, `x` (the horizontal coordinate of the left side of the rectangle), and `y` (the vertical coordinate of the rectangle bottom). We will start by looking at `width` and `x` (Example 2.4):

Example 2.4: Creating bars in *main.js*

JavaScript

```
    //Example 2.1 line 17...create a second svg element to hold the bar chart
    var chart = d3.select("body")
        .append("svg")
        .attr("width", chartWidth)
        .attr("height", chartHeight)
        .attr("class", "chart");

    //set bars for each province
    var bars = chart.selectAll(".bars")
        .data(csvData)
        .enter()
        .append("rect")
        .attr("class", function(d){
            return "bars " + d.adm1_code;
        })
        .attr("width", chartWidth / csvData.length - 1)
        .attr("x", function(d, i){
            return i * (chartWidth / csvData.length);
        })
        .attr("height", 460)
        .attr("y", 0);
```

In Example 2.4, to make each bar just wide enough so that they fill the container horizontally but have gaps in between, we set the `width` attribute of each bar to *1/n - 1* pixels, where *n* is the number of bars, represented by the `length` of the `csvData` features array (line 16). To spread the bars evenly across the container, we set the `x` attribute of each bar to *i* * `chartWidth` */n*, where *i* is the index of the datum; this will have the effect of moving each bar to the right of the previous one (lines 17-19). Temporarily, we set an arbitrary bar `height`—the height of the chart container—and an arbitrary `y` attribute of 0, just so the bars are visible (lines 20-21). We will deal more with the vertical attributes momentarily, but for now, let's take a look at our evenly-spaced bars (Figure 2.3):

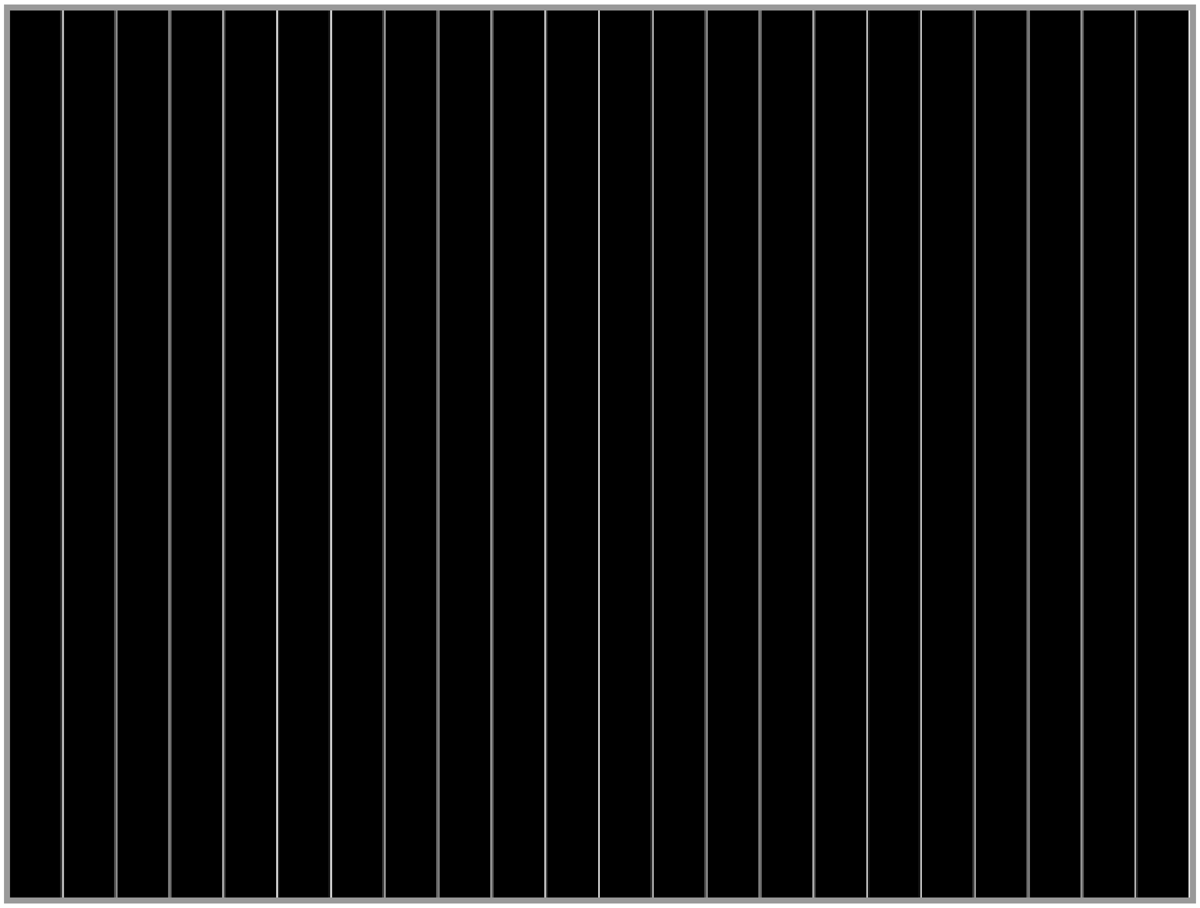


Figure 2.3: Evenly-spaced bars in the bar chart frame

Now let's take a look at bar `height` and `y` coordinate. We want each bar's height to be sized proportionally to its attribute value. Recall from Previous Module , Lesson 3 that we can use a linear scale to produce a range of output values between 0 and the chart height. We can then use that scale to assign both vertical attributes (Example 2.5):

Example 2.5: Setting the bar heights with a linear scale in *main.js*

JavaScript

```
    //create a scale to size bars proportionally to frame
     var yScale = d3.scaleLinear()
         .range([0, chartHeight])
         .domain([0, 105]);

     //Example 2.4 line 8...set bars for each province
     var bars = chart.selectAll(".bars")
         .data(csvData)
         .enter()
         .append("rect")
         .attr("class", function(d){
             return "bars " + d.adm1_code;
         })
         .attr("width", chartWidth / csvData.length - 1)
         .attr("x", function(d, i){
             return i * (chartWidth / csvData.length);
         })
         .attr("height", function(d){
             return yScale(parseFloat(d[expressed]));
         })
         .attr("y", function(d){
             return chartHeight - yScale(parseFloat(d[expressed]));
         });
```

In Example 2.5, we create a linear `yScale`, assigning a range from 0 to the height of the chart and a domain that will encompass all of our sample data attribute values (lines 2-4). We then apply the `yScale` to each attribute value to set the bar `height` (lines 18-20). We subtract the scale output from the chart height to set the `y` attribute to ensure that the bars "grow" up from the bottom rather than "fall" down from the top of the chart (lines 21-23).

We can also use our bar chart to show users where our class breaks are in the dataset by applying our `choropleth` function to create the `<rect> fill`, passing it the datum and `colorScale` (Example 2.6):

Example 2.6: Applying the color scale at the end of the `bars` block in *main.js*

JavaScript

```
         //Example 2.5 line 23...end of bars block
         .style("fill", function(d){
             return choropleth(d, colorScale);
         });
```

We can now see our attribute values represented by the bar height and classes shown by bar color (Figure 2.4):

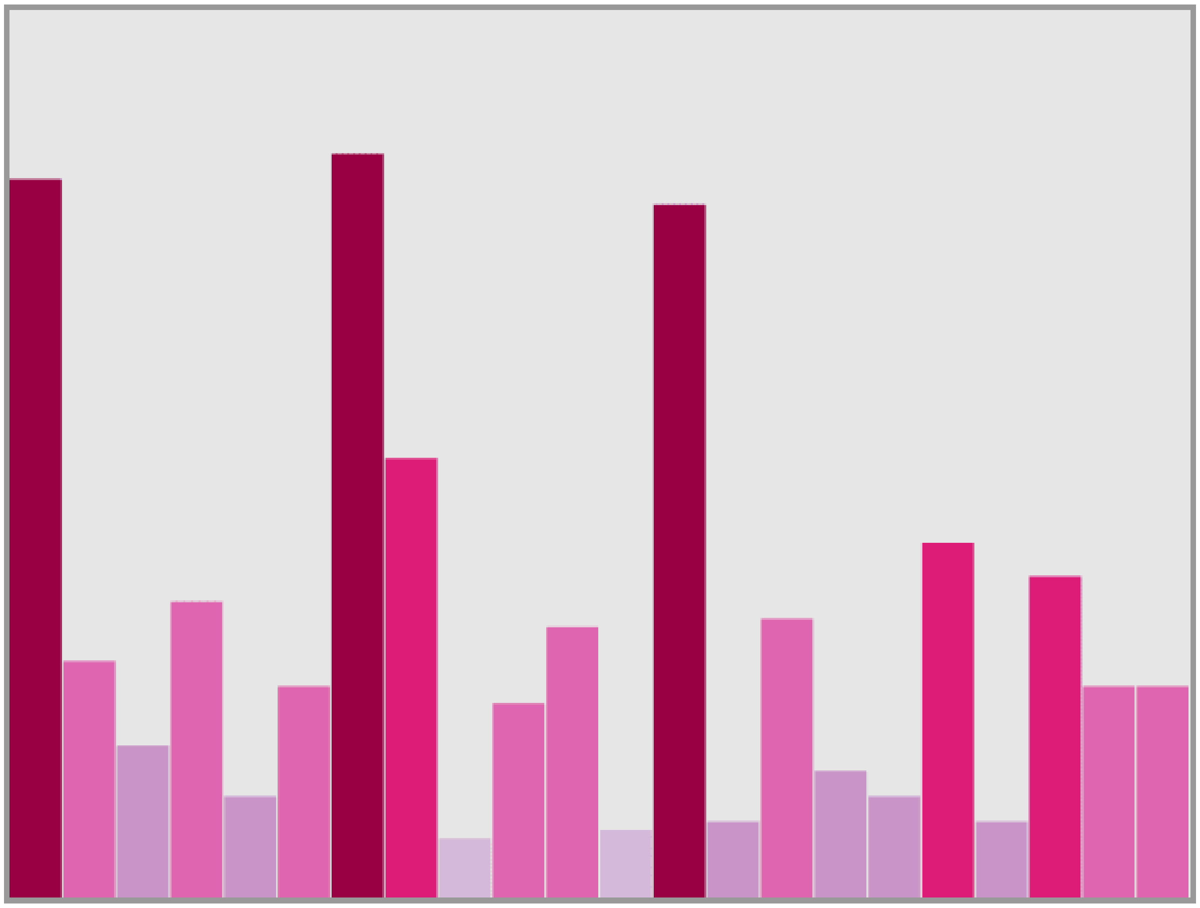


Figure 2.4: Bar chart with vertical scale and choropleth classification applied

We're making good progress, but the chart is still a little messy. We can neaten it and provide a better visual representation of the data by sorting the bars in either ascending or descending size order. This can be accomplished using D3's `.sort()` (https://github.com/d3/d3-selection/blob/master/README.md#selection_sort) method to sort the data values before applying any of our `<rect>` attributes (Example 2.7):

Example 2.7: Sorting attribute values to reorder the bars in *main.js*

JavaScript

```
     //Example 2.5 line 6...set bars for each province
    var bars = chart.selectAll(".bars")
        .data(csvData)
        .enter()
        .append("rect")
        .sort(function(a, b){
            return a[expressed]-b[expressed]
        })
        .attr("class", function(d){
            return "bars " + d.adm1_code;
        })
        //...
```

D3's `.sort()` method, like the array sort method (http://www.w3schools.com/jsref/jsref_sort.asp) native to JavaScript, uses a comparator function to compare each value in the data array to the next value in the array and rearrange the array elements if the returned value is positive (lines 6-8). Subtracting the second value from the first in the comparator function (line 7) orders the bars from smallest to largest, making the chart more readable. Note that if you want to order the bars from largest to smallest, you can simply reverse the two values in the comparator function.

We now have a nicely arranged bar chart (Figure 2.5):

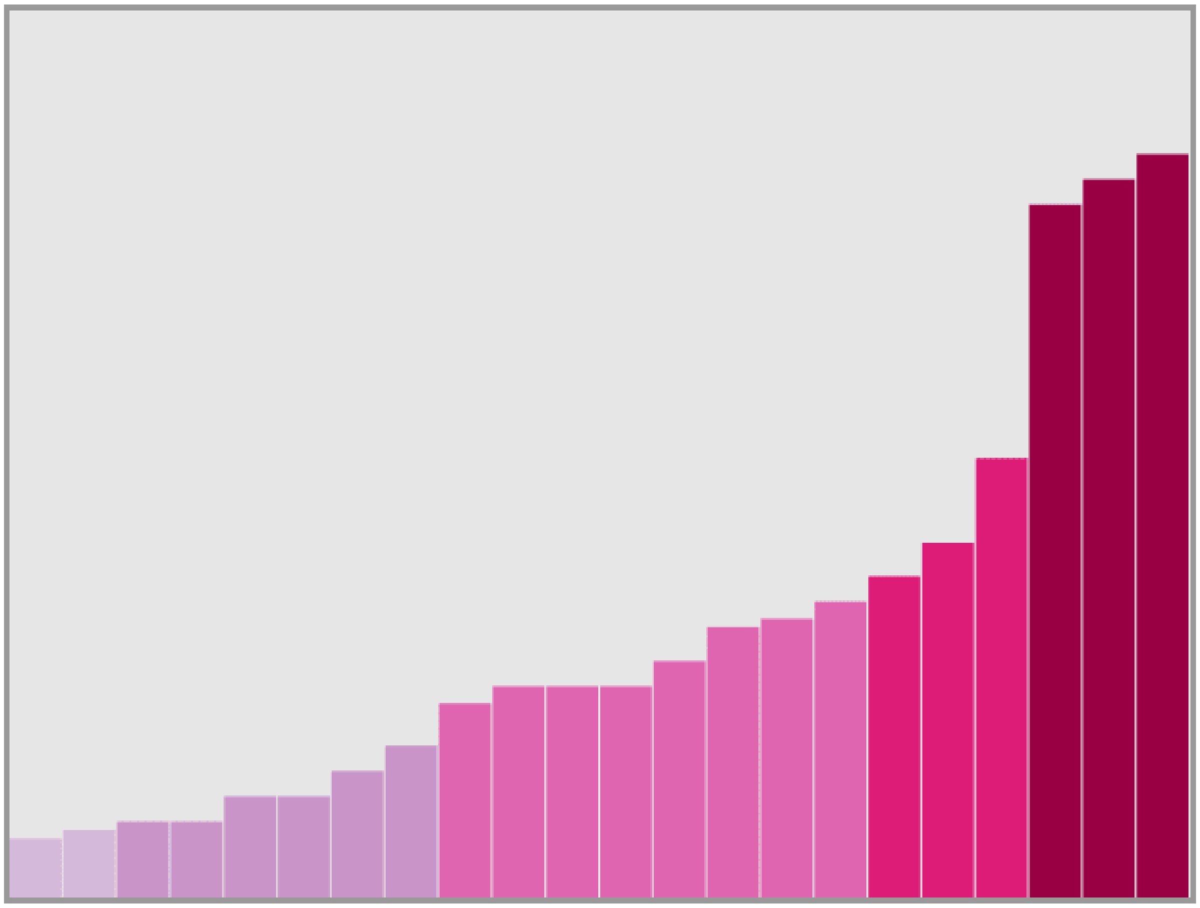


Figure 2.5: A neatly arranged and classed bar chart

2.3 Chart Annotation

As it stands, the bar chart gives the user a better sense of the shape of our attribute dataset for the expressed attribute. However, it would be difficult to tell anything about the attribute *values* without more information added. Some of this information will be given to the user via the *retrieve* operator, which we will add in the next Module. But just a glance at the chart should give the user a basic overview of the data. Thus, we need to **annotate** the chart, or add the important contextual information that will vastly improve the utility of the visualization.

One approach we can take is to add the attribute values as numerical text to the bars themselves. Recall from previous Module that text can only be added within `<text>` elements in an SVG graphic. We can add our bar values by creating a new `.selectAll()` selection similar to our `bars` block, but appending `<text>` elements instead of `<rect>` elements (Example 2.8):

Example 2.8: Adding text to the bars in *main.js*

JavaScript

```
    //annotate bars with attribute value text
    var numbers = chart.selectAll(".numbers")
        .data(csvData)
        .enter()
        .append("text")
        .sort(function(a, b){
            return a[expressed]-b[expressed]
        })
        .attr("class", function(d){
            return "numbers " + d.adm1_code;
        })
        .attr("text-anchor", "middle")
        .attr("x", function(d, i){
            var fraction = chartWidth / csvData.length;
            return i * fraction + (fraction - 1) / 2;
        })
        .attr("y", function(d){
            return chartHeight - yScale(parseFloat(d[expressed])) + 15;
        })
        .text(function(d){
            return d[expressed];
        });
```

In Example 2.8, we construct our `numbers` block following the same pattern as our `bars` block but append `<text>` elements (line 5) and alter their attributes. The `text-anchor` attribute center-justifies the text (line 12). The `x` attribute adds half of the bar's width to the formula for the horizontal coordinate used in the `bars` block so that each number is centered in the bar (lines 13-16). The `y` attribute accesses the `yScale` using the same formula as in the `bars` block, but adds 15 pixels to lower the text so it appears inside of, rather than on top of, each bar (lines 17-19). Finally, the `.text()` operator places the expressed attribute value in each `<text>` element.

A minor stylistic addition is to change the default black text to white in *style.css* to make it fit better with the chart's color scheme (Example 2.9):

Example 2.9: Styling attribute value annotation in *style.css*

CSS

```
.numbers {
    fill: white;
    font-family: sans-serif;
}
```

This creates tidy numbers in the bars showing the attribute values represented by each bar (Figure 2.6):

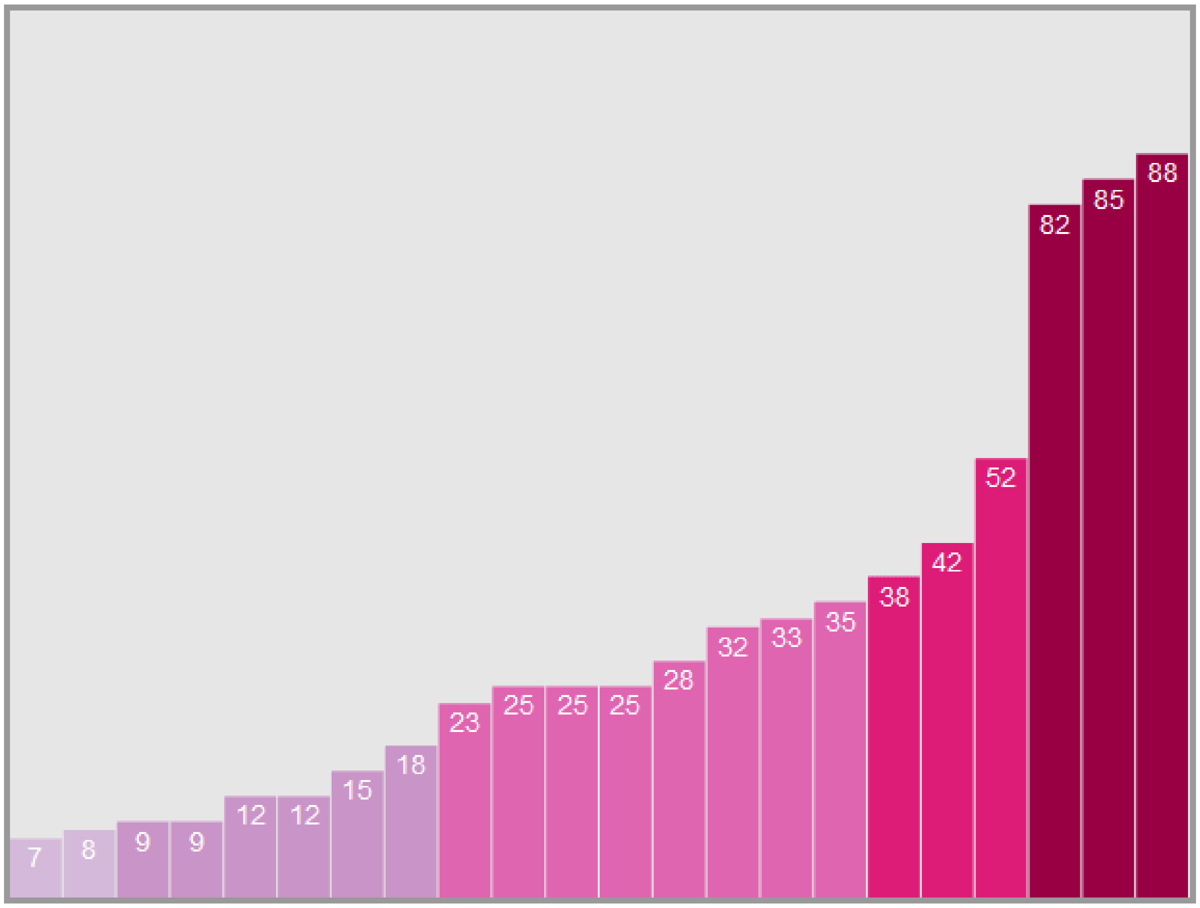


Figure 2.6: Bar chart with numerical attribute value annotation

While we are on the subject of text, we may as well give our chart a title that reflects the current attribute. The title can be added with a simple block appending a single `<text>` element to the chart and positioning it where we want it (Example 2.8):

Example 2.10: Adding a dynamic chart title in *main.js*

JavaScript

```
    //below Example 2.8...create a text element for the chart title
    var chartTitle = chart.append("text")
        .attr("x", 20)
        .attr("y", 40)
        .attr("class", "chartTitle")
        .text("Number of Variable " + expressed[3] + " in each region");
```

In Example 2.10, we append a `<text>` element to the chart container and position it 20 pixels to the right and 40 pixels below the top-left corner of the container (lines 2-4). For the title itself, we create a string that includes the fourth character of the currently expressed attribute name, which in this case indicates the attribute (line 6). Note that you will need to change the formatting of this title string to make sense given the attribute names in your dataset.

Of course, our title should be big and bold, which means overriding the default styles for SVG text with our own styles in *style.css* (Example 2.9):

Example 2.11: Chart title styles in *style.css*

CSS

```css
.chartTitle {
    font-family: sans-serif;
    font-size: 1.5em;
    font-weight: bold;
}
```

We can now see our chart title (Figure 2.7):

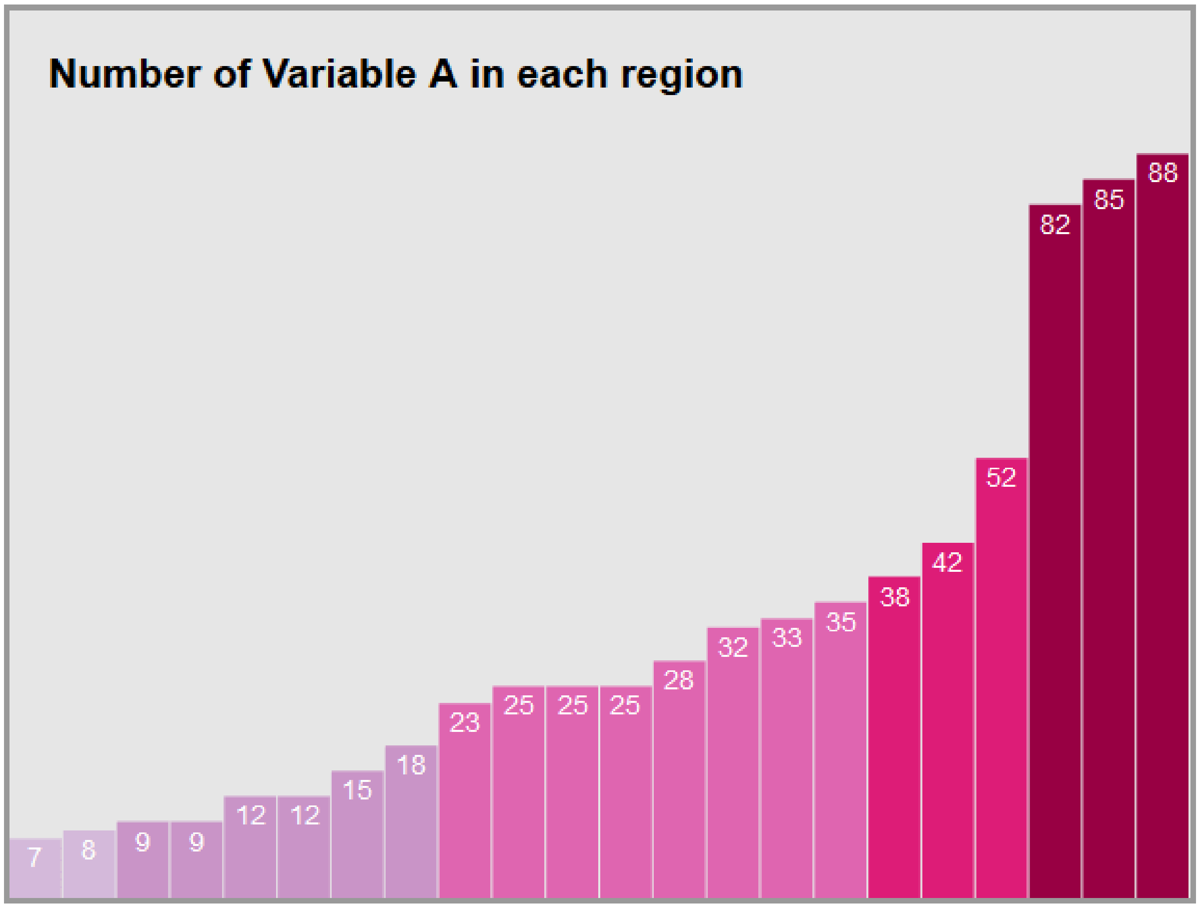


Figure 2.7: Bar chart with dynamic title

3.4 Chart Axis

An alternative annotation that would also work well for the bar chart is a vertical axis. To avoid the problem of "**chartjunk** (https://en.wikipedia.org/wiki/Chartjunk)," it's probably a good idea to include either an axis *or* numbers, not both, so the examples below remove the numbers from the bars. If your graphic will include one or more axes, review previous Module for a complete tutorial on creating axes in D3.

If we want to add a vertical axis to our bar chart, we face a dilemma. Our bars currently expand horizontally to the edges of the `<svg>` container, but the axis numbers and tics must be inside the container to be visible, and so will overlap the bars without significant adjustment to the rest of the chart. We should also reverse the order of the bars so that the tallest bars are closest to the axis, making them easier to measure visually. Figure 2.8 shows our adjusted chart:

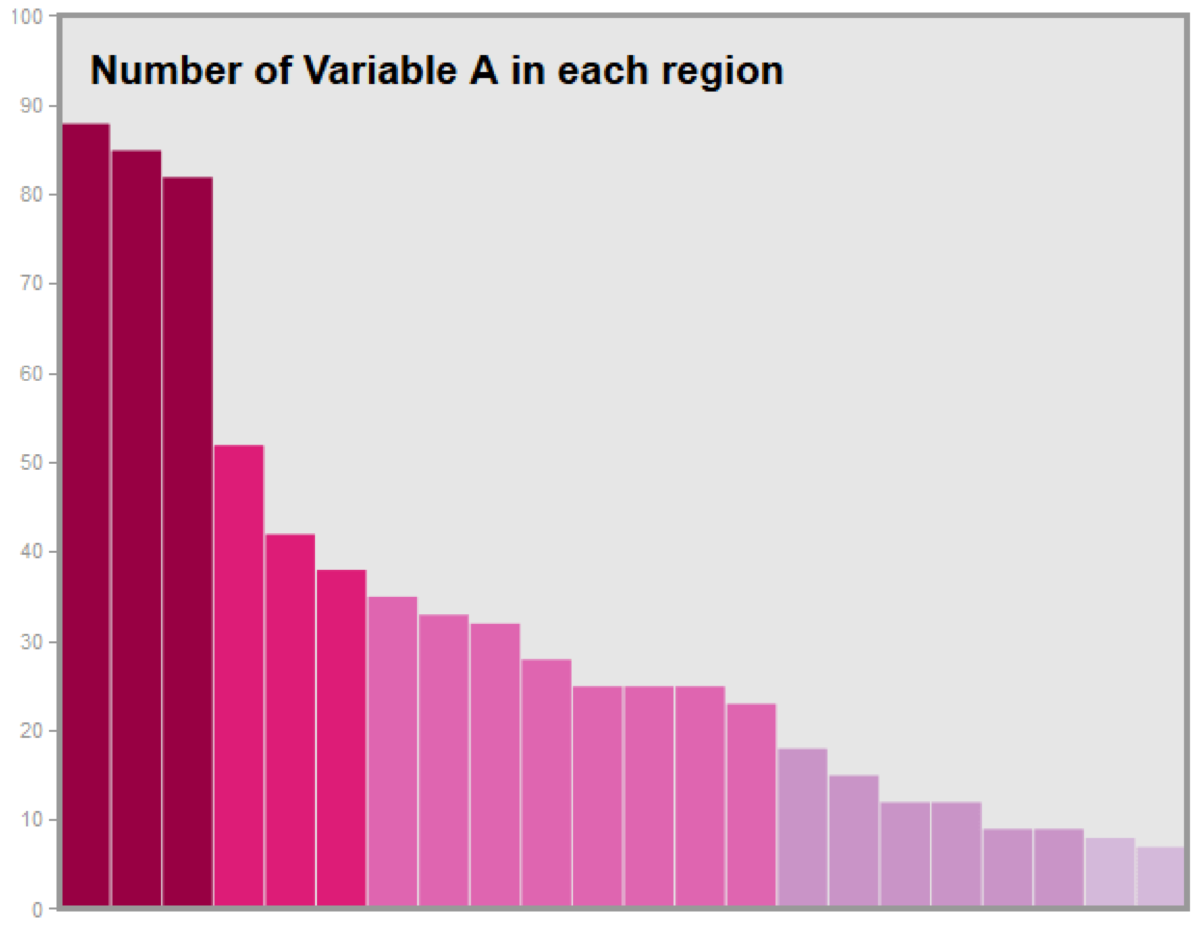


Figure 2.8: Bar chart with an axis

Rather than step through each of the necessary adjustments to the script and styles, we provide our full code for the chart with an axis in Examples 2.12 and 2.13. We trust that at this point, you are able to compare these examples to our previous example code, pick out the differences between the two versions, and analyze what these adjustments accomplish. You may wish to construct both versions, then compare them using the Inspector to see the differences (Figure 2.9).

Example 2.12: Building a bar chart with an axis in *main.js*

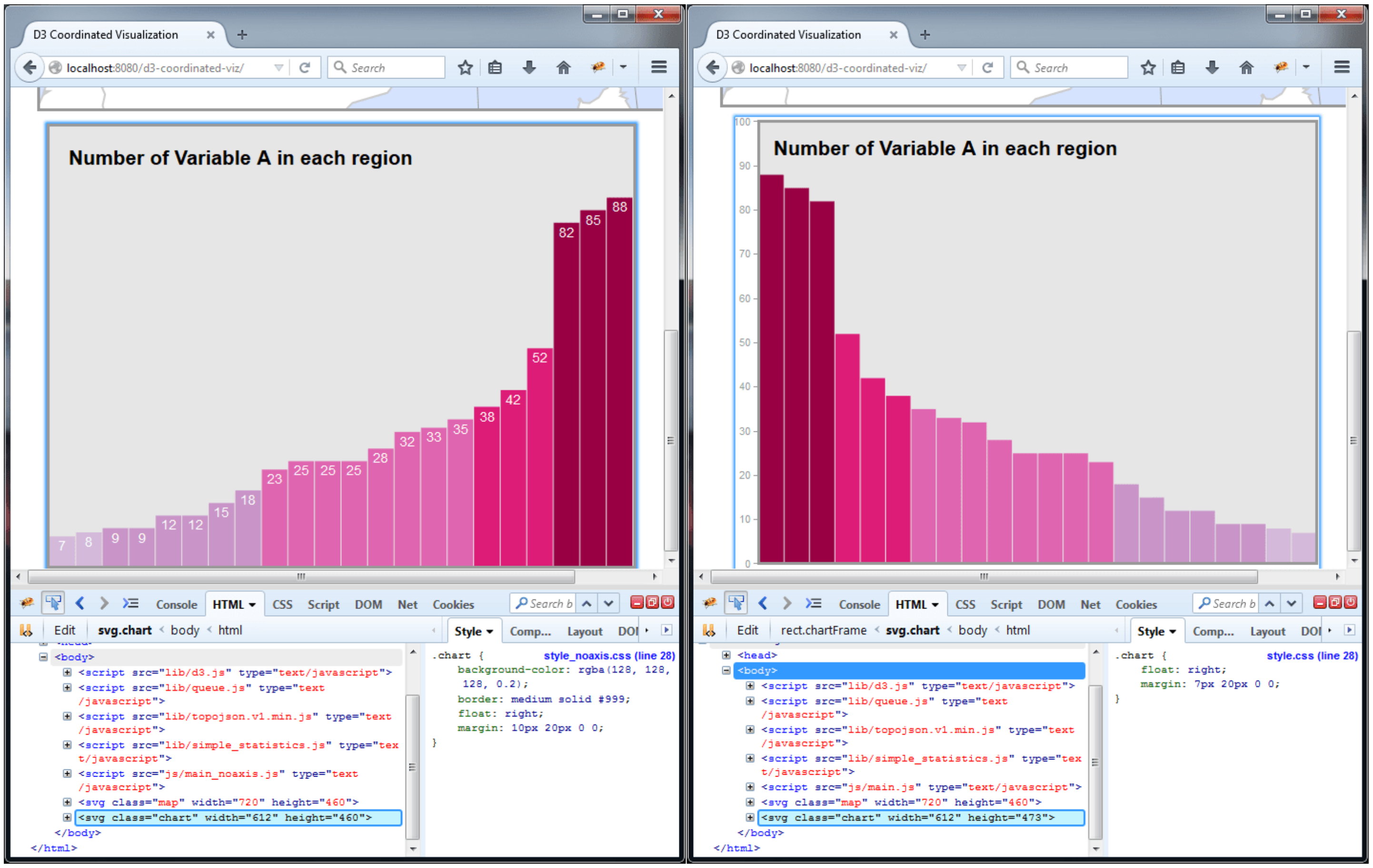
JavaScript

```
//function to create coordinated bar chart
function setChart(csvData, colorScale){
    //chart frame dimensions
    var chartWidth = window.innerWidth * 0.425,
        chartHeight = 473,
        leftPadding = 25,
        rightPadding = 2,
        topBottomPadding = 5,
        chartInnerWidth = chartWidth - leftPadding - rightPadding,
        chartInnerHeight = chartHeight - topBottomPadding * 2,
        translate = "translate(" + leftPadding + "," + topBottomPadding + ")";

    //create a second svg element to hold the bar chart
    var chart = d3.select("body")
        .append("svg")
        .attr("width", chartWidth)
        .attr("height", chartHeight)
        .attr("class", "chart");

    //create a rectangle for chart background fill
    var chartBackground = chart.append("rect")
        .attr("class", "chartBackground")
        .attr("width", chartInnerWidth)
        .attr("height", chartInnerHeight)
        .attr("transform", translate);

    //create a scale to size bars proportionally to frame and for axis
    var yScale = d3.scaleLinear()
        .range([463, 0])
        .domain([0, 100]);

    //set bars for each province
    var bars = chart.selectAll(".bar")
        .data(csvData)
        .enter()
        .append("rect")
        .sort(function(a, b){
            return b[expressed]-a[expressed]
        })
        .attr("class", function(d){
            return "bar " + d.adm1_code;
        })
        .attr("width", chartInnerWidth / csvData.length - 1)
        .attr("x", function(d, i){
            return i * (chartInnerWidth / csvData.length) + leftPadding;
        })
        .attr("height", function(d, i){
            return 463 - yScale(parseFloat(d[expressed]));
        })
        .attr("y", function(d, i){
            return yScale(parseFloat(d[expressed])) + topBottomPadding;
        })
        .style("fill", function(d){
            return choropleth(d, colorScale);
        });

    //create a text element for the chart title
    var chartTitle = chart.append("text")
        .attr("x", 40)
        .attr("y", 40)
        .attr("class", "chartTitle")
        .text("Number of Variable " + expressed[3] + " in each region");
    //create vertical axis generator
    var yAxis = d3.axisLeft()
        .scale(yScale)
        .orient("left");

    //place axis
    var axis = chart.append("g")
        .attr("class", "axis")
        .attr("transform", translate)
        .call(yAxis);

    //create frame for chart border
    var chartFrame = chart.append("rect")
        .attr("class", "chartFrame")
        .attr("width", chartInnerWidth)
        .attr("height", chartInnerHeight)
        .attr("transform", translate);
};
```

Example 2.13: Styles for bar chart with axis in *style.css*

CSS

```
.chart {
    float: right;
    margin: 7px 20px 0 0;
}

.chartTitle {
    font-family: sans-serif;
    font-size: 1.5em;
    font-weight: bold;
}

.chartBackground {
    fill: rgba(128,128,128,.2);
}

.chartFrame {
    fill: none;
    stroke: #999;
    stroke-width: 3px;
    shape-rendering: crispEdges;
}

.axis path,
.axis line {
    fill: none;
    stroke: #999;
    stroke-width: 1px;
    shape-rendering: crispEdges;
}

.axis text {
    font-family: sans-serif;
    font-size: 0.8em;
    fill: #999;
}
```



```
<body>
    <script src="lib/d3.js" type="text/javascript">
    <script src="lib/queue.js" type="text
    /javascript">
    <script src="lib/topojson.v1.min.js" type="text
    /javascript">
    <script src="lib/simple_statistics.js" type="tex
    t/javascript">
    <script src="js/main_noaxis.js" type="text
    /javascript">
    <svg class="map" width="720" height="460">
    <svg class="chart" width="612" height="460">
</body>
</html>
```

```
.chart {                          style_noaxis.css (line 28)
    background-color: rgba(128, 128,
    128, 0.2);
    border: medium solid #999;
    float: right;
    margin: 10px 20px 0 0;
}
```

```
<head>
<body>
    <script src="lib/d3.js" type="text/javascript">
    <script src="lib/queue.js" type="text
    /javascript">
    <script src="lib/topojson.v1.min.js" type="text
    /javascript">
    <script src="lib/simple_statistics.js" type="tex
    t/javascript">
    <script src="js/main.js" type="text/javascript">
    <svg class="map" width="720" height="460">
    <svg class="chart" width="612" height="473">
</body>
</html>
```

```
.chart {                          style.css (line 28)
    float: right;
    margin: 7px 20px 0 0;
}
```

Figure 2.9: Comparing the two chart versions using the Inspector

---

Practice! Create a bar chart or alternative data visualization that clearly expresses the attribute values shown on the choropleth map and is classed using your choropleth classification scheme.

Self-Check:

1. Which of the following visual elements can be used to annotate a coordinated visualization?

a. Text

b. Color

c. One or more axes

d. a and b

e. a, b, and c

2. True/False: Points will be deducted from your grade if you copycat the visual appearance of the choropleth map and bar chart shown in this module.

Module Reminder

1. Join your CSV attribute data with your GeoJSON features.

2. Encapsulate your *main.js* script in an anonymous wrapper function and neatly organize tasks within the script into separate named functions.

3. Dynamically style your enumeration units as a choropleth map using a classification scheme that makes sense given your dataset.

4. Create a container for your coordinated visualization that is neatly arranged with the map on the web page so that both are visible and tidy at a number of different browser window sizes.

5. Create a coordinated visualization that supports your choropleth map by providing a sensible alternative view of the data.

6. Annotate your coordinated visualization with a title and value labels or one or more axes.

7. With all of the above tasks completed, commit changes to your *d3-coordinated-viz* web directory and sync with GitHub.