SI 649 Fall 2018, 9/26/2018

Lab 4 Handout

DUE: Tuesday 10/2 at 11:59 PM

TLDR

What you'll need: Most of the files are in Canvas under week 4/lab – 1) Online copy of this document [http://bit.ly/2OOQmhS -> lab4_handout.pdf], 2) Folder for d3 in-class http://bit.ly/2OOQmhS -> inclass.zip, (3) Folder for d3 individual task http://bit.ly/2OOQmhS -> demographics.zip

What to turn in: Two files: (1) a zipped folder with in-class exercise code (3) a zipped folder with individual task code.

When to turn in: Tuesday 10/2 at 11:59 PM, Fine print: All technical labs are due Tuesday (see the schedule). You may turn **one** lab in late by the time the next class starts (Wednesday at 2:30). Labs build off each other so it's important that you keep up. Do your best with these. We know some of you are building up coding skills at the same time as learning the APIs. We do give partial credit so turn in what you have.

Learning Objectives:

- How to load a .csv file using d3
- How to deploy a simple server to serve HTML content
- How to group data in d3
- Scale functions in D3
- How to create a simple bar chart in d3
- How to create small multiple bar charts

NOTE: In this lab we are going to use the Metafilter dataset. Metafilter is a community weblog, and the dataset consists of **total number of questions (n)** posted by **topic (category)** during **2004 – 2014 (year)**. The dataset is made available to you in your inclass code folder for this week **(inclass > data> data.csv).**

year	category	n
2004	clothing, beaut	141
2004	computers & ir	2489
2004	education	151
2004	food & drink	275
2004	grab bag	285
2004	health & fitnes	379

Figure 1: Snapshot of Metafilter dataset

Loading a CSV file

🟥 Q1: Load the csv file and assign it to the "data" variable. The data variable has already been declared for you at the top of the file.

draw.js

```
function loadData() {
    d3.csv("data/data.csv", function (d) {
        data = d;
        data.forEach(function (item) {
           item.n = parseInt(item.n);
         });
    });
}
```

Running a Python Server

At this point, if you try to open index.html in your browser you will get an error. For security reasons, browsers don't allow loading external files such as our data.csv. There are a few different ways to fix this, one of which is to set up a local server. [You can read more about Cross Origin Resources here: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS]

- Have python 2.7/3 installed on your computer
- If you are on a Windows machine:
    - Open file explorer and navigate to the "inclass" folder
    - From the file option on your ribbon-bar, select "Open Windows PowerShell" (you may use the command prompt instead)
    - In the PowerShell window, type:
      python -m SimpleHTTPServer 8000
      OR
      python3 -m http.server
- On the mac, open terminal and navigate to the "inclass" folder. Then execute on of the above commands depending on your python version.
- In Google Chrome, type: localhost:8000 in your browser address bar and hit enter.

Grouping Data

🟥 Q2: Let us group the data by "category" column.

In JSON terms we need a nested array. The top-level JSON object has a **key** attribute which will be set of one of the categories in the dataset, and a **values** attribute which is an array of all data that belongs to that category.

draw.js

```
function groupDataByCategory() {
    var groupedData = d3.nest()
        .key(function (d) { return d.category })
        .entries(data);
    return groupedData;
}
```

☐ Open the developer console and type "data" and hit enter. This will show you the data attribute which is an array of 220 items

☐ Next, call the **groupDataByCategory**() function using the console and look at the output. You should see a nested array of 20 items. Expand one of the items, and look at its values attribute.

Aggregate grouped data

■ Q3: Let us write a function the aggregates all n by year.

draw.js

```
function groupDataByYear() {
    var groupedData = d3.nest()
        .key(function (d) { return d.year })
        .rollup(function (v) { return d3.sum(v, function (d) {
return d.n; }) })
        .entries(data);
    return groupedData;
}
```

☐ Call the **groupDataByYear** () function using the console and look at the output. You should see a nested array of 11 items, one for each year. Expand one of the items, and look at its values attribute. It should show you a single number which is the sum of all rows for that year.

☐ D3 supports a number of aggregation functions. Look up the documentation and write a function that returns the mean number (instead of sum) per year. https://github.com/d3/d3-array#statistics

An introduction to D3 Scale Functions

In the previous lab, we wrote code to **manually set x and y positions of individual rectangles** (and circles). While this is ok for simple (and a small number of) shapes, calculating x and y positions for different chart elements (marks, axes, labels, tooltips, etc) can be hard.

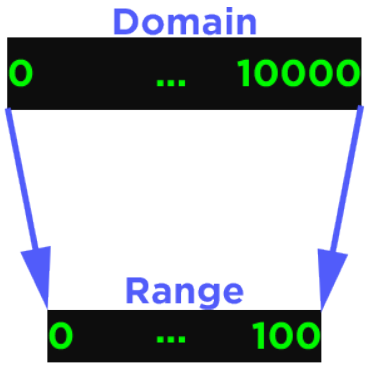
D3 offers several useful functions that do the calculations for us. These are broadly called **scale functions.**

Domains and Ranges

- When making a chart, we are **mapping data to pixels**.
- We only have a limited and fixed number of pixels available to draw the chart.
- This is typically defined by the **width and height attributes of our SVG** element.
- For example, when drawing a scatter plot, the point with the **smallest x and y value** will be drawn at the **bottom-left corner of the SVG**, and the point with **largest x and y value** will be drawn at **the top-right corner of the SVG**. All other elements will be placed somewhere in between.
- Similarly, on a bar chart, we want n number of bars to be evenly distributed along the width of the SVG.

To support such mappings (min and max x-y values, count, etc.), D3 scales have two attributes called **DOMAIN** and **RANGE**.

- *A scale's **input domain** is the range of possible input data values.*
- *A scale's **output range** is the range of possible output values, commonly used as display values in pixel units.*



D3 offers the following types of scale types:

- Scales with continuous input and continuous outputs (e.g., scaleLinear, scaleTime)
- Scales with continuous input and discrete outputs (e.g., scaleQuantile)
- Scales with discrete input and discrete outputs (e.g., scaleOrdnial, scaleBand)

https://d3indepth.com/scales/

In this lab, we will look at **BAND** and **LINEAR** scales that correspond to the x and y axis of the (vertical) bar chart respectively.

■ **Q4: But first, let's define the dimensions of the SVG so we can map it to the range attribute of the scale function**

draw.js

```
function visualizeBarChart(dataitems) {
    // set the dimensions and margins of the graph
  var margin = { top: 20, right: 20, bottom: 30, left: 60 },
        width = 940 - margin.left - margin.right,
        height = 500 - margin.top - margin.bottom;
}
```

■ **Q5:** Defining the Band Scale for x-axis of the bar chart

When creating bar charts `scaleBand` helps to determine the geometry of the bars, taking into account padding between each bar. The **domain** is specified as an array of values (one value for each band) and the **range** as the minimum and maximum extents of the bands (e.g. the total width of the bar chart).

draw.js > visualizeBarChart()

```
    var x = d3.scaleBand()
        .domain(dataitems.map(function (d) { return d.key; }))
        .range([0, width])
        .padding(0.1);
```

■ **Q6:** Defining the Linear Scale for y-axis of the bar chart

Linear scales are probably the most commonly used scale type as they are the most suitable scale for transforming data values into positions and lengths. If there's one scale type to learn about this is the one.

They use a linear function (`y = m * x + b`) to interpolate across the domain and range.

draw.js > visualizeBarChart()

```
var y = d3.scaleLinear()
        .domain([0, d3.max(dataitems, function (d) { return d.value; })])
        .range([height, 0]);
```

Drawing a bar chart

■ Q7: Let us first draw an SVG container for our bar chart.

draw.js > visualizeBarChart()

```
var svg = d3.select("#chart1").append("svg")
      .attr("width", width + margin.left + margin.right)
      .attr("height", height + margin.top + margin.bottom)
      .append("g")
      .attr("transform",
      "translate(" + margin.left + "," + margin.top + ")");
```

■ Q8: Next let us draw the bars by appending rectangles

draw.js > visualizeBarChart()

```
svg.selectAll(".bar")
    .data(dataitems)
    .enter().append("rect")
    .attr("class", "bar")
    .attr("fill", "#E0D22E")
    .attr("x", function (d) { return x(d.key); })
    .attr("width", x.bandwidth())
    .attr("y", function (d) { return y(d.value); })
    .attr("height", function (d) { return height - y(d.value); });
```

■ Q9: Lastly, let us draw the axes.

x- axis:

draw.js > visualizeBarChart()

```
// add the x Axis
svg.append("g")
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(x));
```

y-axis:

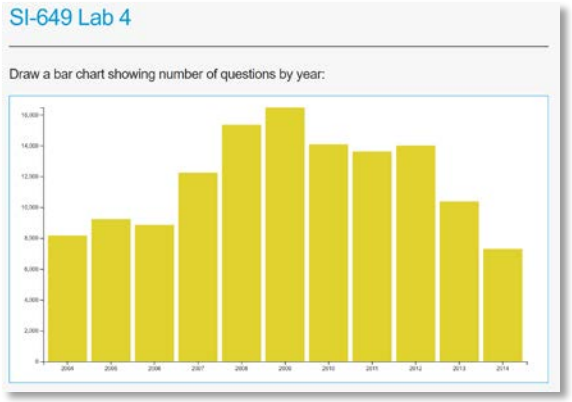
draw.js > visualizeBarChart()

```
// add the y Axis
svg.append("g")
    .call(d3.axisLeft(y));
```

■ Q10 Calling the visualizeBarChart() function

draw.js > loadData()

```
function loadData() {
    //code for Q1 goes here
    d3.csv("data/data.csv", function (d) {
        data = d;
        visualizeBarChart(groupDataByYear());
```



☐ Q11 Take a few minutes to look at the rendered SVG elements using the dev tools. Specifically, **look at the x and y attribute values of each bar rectangle**. Expand each of the scales and **see what element is used to draw the line, how are axes labels** drawn

Drawing a small multiple bar chart

LOGIC: For a small multiple bar chart we must

1. Define width, and height of each svg, and also x and y scales
2. Iterate through each of the categories from Q2
3. Execute questions Q7 – Q9 for each category

■ Q12 defining margin and dimensions for each small SVG

draw.js > visualizeSmallMultipleBarChart()

```
var margin = { top: 0, right: 10, bottom: 40, left: 40 },
        width = 280 - margin.left - margin.right,
        height = 150 - margin.top - margin.bottom;
```

■ Q13 Define x scale

draw.js > visualizeSmallMultipleBarChart()

```
    var x = d3.scaleBand()
        .domain(dataitems[0].values.map(function (d) { return d.year; }))
        .range([0, width])
        .padding(0.1);
```

Since all items have the same set of year, we are just taking the first item and looping through its years to define the key

■ Q14 Define y scale

draw.js > visualizeSmallMultipleBarChart()

```
       var y = d3.scaleLinear()
        .domain([0, d3.max(data, function (d) { return d.n; })])
        .range([height, 0]);
```

Note the use of data here instead of dataitems. Since we are not doing any aggregate operations on the data, we want to define the y-scale using the maximum n value. (We can do the same using the nested dataitems object, but it will be a bit more complex as we need to iterate max of max).

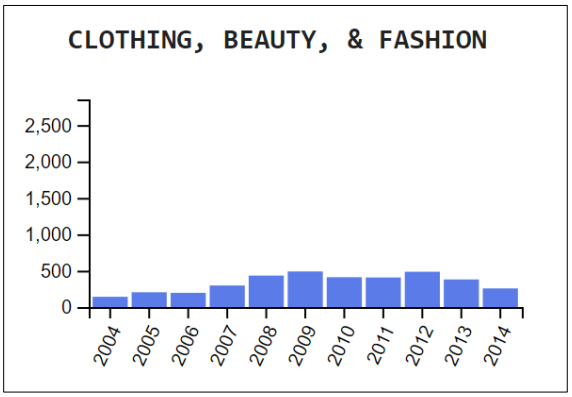
■ Q15 Looping through each category (nested item)

draw.js > visualizeSmallMultipleBarChart()

```
dataitems.forEach(function (v, i) {

});
```

For **each category** we are going to draw:



```
▼<div id="holderclothing, beauty, & fashion" class="chartholder">
    <h6>clothing, beauty, & fashion</h6>
  ▼<svg class="categoryBar" width="280" height="150"> == $0
    ▼<g transform="translate(40,0)">
        <rect class="bar" fill="#5B7BE9" x="2.072072072072075" width=
        "18.64864864864865" y="104.5617110799439" height="5.438288920056095
        <rect class="bar" fill="#5B7BE9" x="22.792792792792795" width=
        "18.64864864864865" y="102.17040673211781" height="7.82959326788218
```

NOTE:

- The entire chart is contained within a div tag
- In other words, the div tag has a header for the category (<h6>) and an SVG

■ Q16 Let us first draw the div wrapper

draw.js > visualizeSmallMultipleBarChart() > Inside the forEach loop we created in Q15

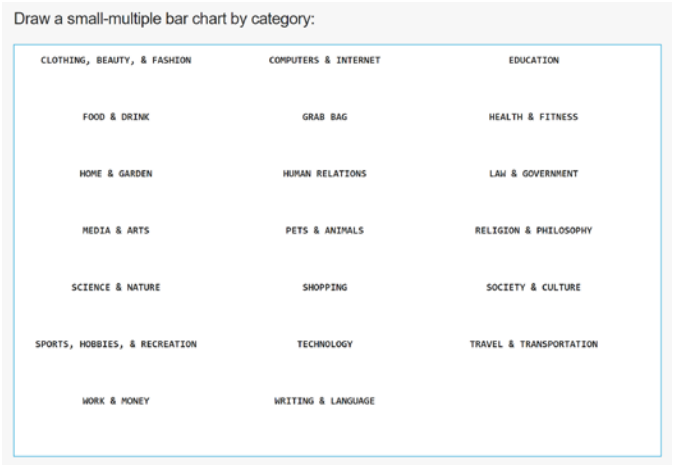
```
var div = d3.select("#chart2").append("div")
            .attr("id", "holder" + v.key)
            .attr("class", "chartholder");
```

■ Q17 Next, let us add the header with the category name

draw.js > visualizeSmallMultipleBarChart() > Inside the forEach loop we created in Q15

```
    div.append("h6").html(v.key);
```

At this point, you should be seeing something like this:



■ Q18 Let us draw the SVG

draw.js > visualizeSmallMultipleBarChart() > Inside the forEach loop we created in Q15

```
        var svg = div.append("svg")
            .attr("class", "categoryBar")
            .attr("width", width + margin.left + margin.right)
            .attr("height", height + margin.top + margin.bottom)
            .append("g")
            .attr("transform", "translate(" + margin.left + "," +
margin.top + ")");
```

■ Q19 Let us draw the bars

draw.js > visualizeSmallMultipleBarChart() > Inside the forEach loop we created in Q15

```
svg.selectAll(".bar")
            .data(v.values)
            .enter().append("rect")
            .attr("class", "bar")
            .attr("fill", function (d) { return category_colors[v.key] })
            .attr("x", function (d) { return x(d.year); })
            .attr("width", x.bandwidth())
            .attr("y", function (d) { return y(d.n); })
            .attr("height", function (d) { return height - y(d.n); });
```

■ Q20 Let us draw the y axis

draw.js > visualizeSmallMultipleBarChart() > Inside the forEach loop we created in Q15

```
      // add the y Axis
        svg.append("g")
            .call(d3.axisLeft(y).ticks(5));
```

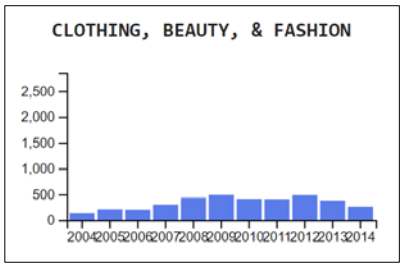
Note that here, we are limiting the number of ticks to 5

■ Q21 Let us draw the x axis

draw.js > visualizeSmallMultipleBarChart() > Inside the forEach loop we created in Q15

```
        svg.append("g")
           .attr("transform", "translate(0," + height + ")")
           .call(d3.axisBottom(x))
           .selectAll("text")
           .style("text-anchor", "end")
           .attr("dx", "-.8em")
           .attr("dy", ".15em")
           .attr("transform", "rotate(-65)");
```

We are selecting all the axes text and rotating it by 65 degrees so that they don't overlap



The final output should look like this:

Section: Individual Task

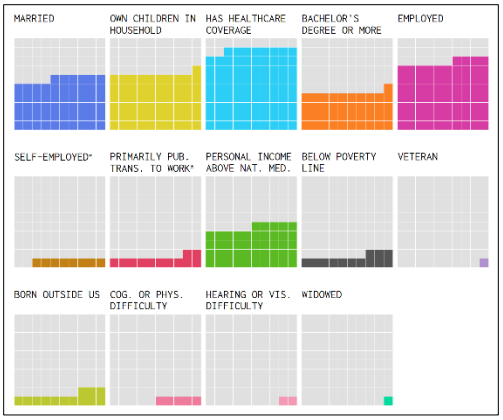
Part A: The table below lists many of the functions we covered in this lab. For each function, look up the documentation and write a description of what the function does.

Summary of functions covered in this lab

Function Name	Description	URL
parseInt()		
d3.nest()		
d3.keys()		

d3.entries()		
d3.rollup()		
d3.scaleBand()		
domain()		
range()		
padding()		
d3.scaleLinear()		
.call()		
d3.axisLeft()		
d3.axisBottom()		
.ticks()		
.style()		

Part B: Recreate the static version of https://flowingdata.com/2018/01/23/the-demographics-of-others/



- The individual task assignment **from last week** (part A) is d3 code to create a grid of 10x10.
- The dataset and skeleton code is available to you on canvas. Lab4 -> demographics.zip
- **Based on what we did in class today and last week's code for grid 10x10, create the following small multiple chart by categories**
- HINT: You will update the category_colors accordingly, and conditionally set the fill of each rectangle based on its index.
- Additional hints are provided to you in the draw.js file

Extra Credits:

In the individual task index.html we have provided the buttons like the ones found in the original article. See if you can wire the button click events to update the charts. You DO NOT need to have the animated transitions, just redraw. Additional hints provided in the draw.js file.