SI649/EECS548 Fall 2016 – Lab 4

Oct. 3 & 4, 2016

Lab Logistics:

- **Though some of you may get through this, we do not expect you to finish this entire lab in class.** Work through as much as you can today and then finish the rest at home to turn in for next week (upload through Canvas by midnight, Sunday, Oct. 9th).
- Please work with your lab partner *in class*. You can continue to work with them on the assignment outside of class or find another partner (please try to keep groups to 2-3 people and report everyone you worked with on your assignment). You may not share code with other groups/individuals (though you can talk through solutions). You may either edit the code together with your lab partner in one file, on one computer (just don't forget to share it before you leave class today) or you can work on your own laptops. Whatever is more effective for you…
- Rename the html file with your name (e.g., eadar_lab4.html). Don't forget to modify your name and your partner's name in the HTML file.

What to turn in:

You will upload:

☐ The modified HTML file (e.g., eadar_lab4.html)

Note that to test your code we will put in a new salary table with different numbers. If you hardcode anything it will break!

Step 0) A few basics

☐ **Download the starting file lab4.html**

We are going to work with the upgraded version of the `salaryData` from lab 3. On the top of the file, you will find the dataset. Our employees start with three components to their salaries based on the projects they work on (project A, project B, and project C). Other fields (e.g., total salary can be calculated dynamically based on these pieces).

```
var salaryData =
                    [{name:"A",dept:"IT",salaryA:1, salaryB:3, salaryC:0},
                    {name:"B",dept:"HR",salaryA:1, salaryB:0.5, salaryC:0.5},
                    {name:"C",dept:"HR",salaryA:1, salaryB:1, salaryC:1},

        ……
```

We have included the variables and helper functions to organize the dataset. It's important to understand how to use these variables and helpers. Here are simple descriptions about these functions and variables. You can read detailed documentation about them.

----------Variables ----------

```
// (NEW) a list of projects, initial value is set to
["salaryA","salaryB","salaryC"].
var listOfProjects = ...
```

----------Functions----------

```
// input a data and a list of project
// calculate the sum salary of all projects in the projects,
// (e.g., listOfProjects)
// and store it in the data under a field called "salary"
calculateTotalSalary(data,projects) {...}

// calculate mean salary of each department
// return a dictionary of department means.
findDeptMean(data) {...}

// calculate and add overunder information to the data
addingOverunder(data) {...}

// Input a dataset and a string called fieldName
// Find the min and max value of that field within the dataset
// Store the min max value in a dictionary and return it.
findMinMaxRange(data,fieldName) {...}

// (NEW) Find the min and max salary of all projects in the
// projects array (e.g., listOfProjects)
// Store the min, max value in a dictionary and return it.
findSalaryCompRange(data,projects) {...}
```

Recall last time we have plotted a salary composition chart for each employee. We have coded this part for you (there are various ways to implement this, we picked one). When you load the page, you will see:

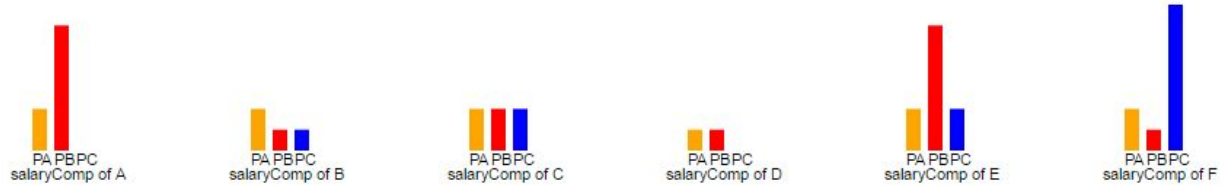


Figure 1: Initial plot of salaryComp Bar charts.

For our specific implementation we loop over the project names. At a high level we have the following steps:

1. Get `salaryCompHeightScale` , a d3 scale function that maps `salaryCompRange` to `salaryCompHeightRange` .
2. Create a `div (id=barCharts)` . Write title, set up svg.
3. Write helper function called `plotEachBar(colorOfBar,xStart,fieldName)` : This function plots bars of a project with the name fieldName. E.g., `if fieldName="salaryA"` , this function will plot bars for salary A. xStart specify the start location of X. Figure 2 explains the layout and variables that is used to set up these charts.
4. We call `plotEachBar(...)` for each project
5. Finally, we add titles for each sub chart.

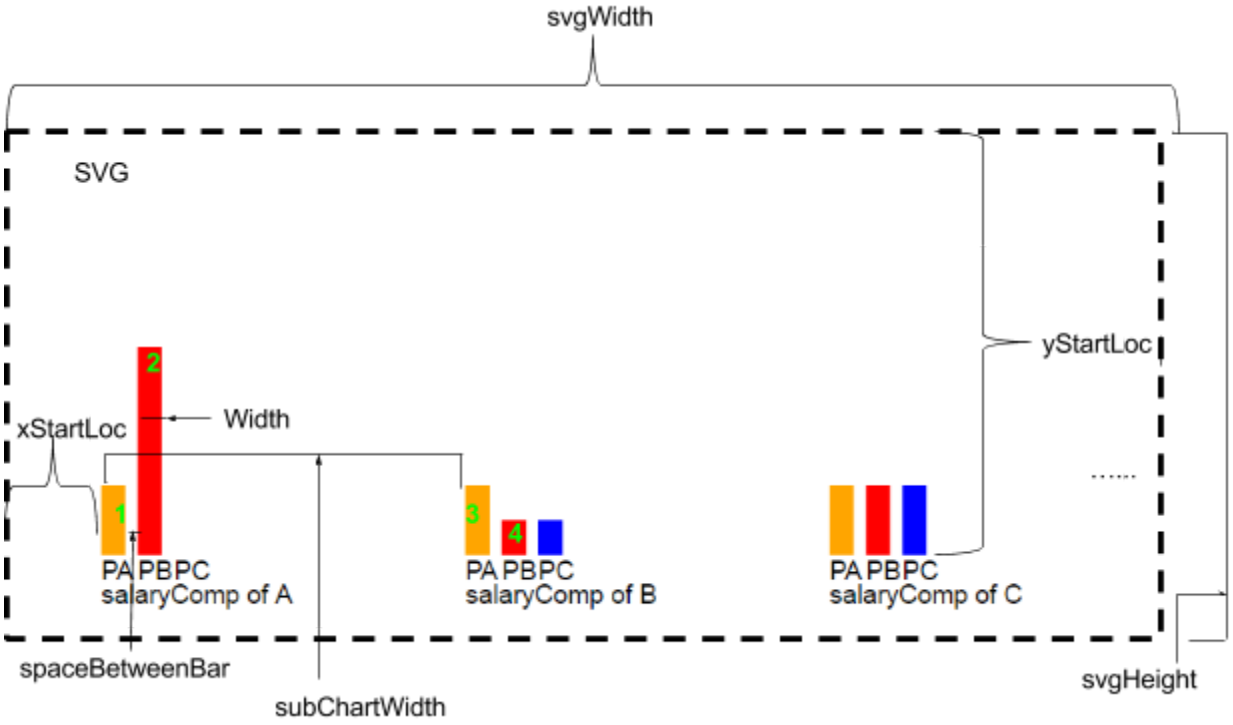


Figure 2 Layout and variables for `salaryComp` Bar Charts. You have used these parameters to determine the x and y location of the bars. You might have hardcoded the `subChartWidth` to be 150 and `yStartLoc` to be 105. For this lab, please use these parameters instead of hard coding these values.

Part 1 Chernoff Faces

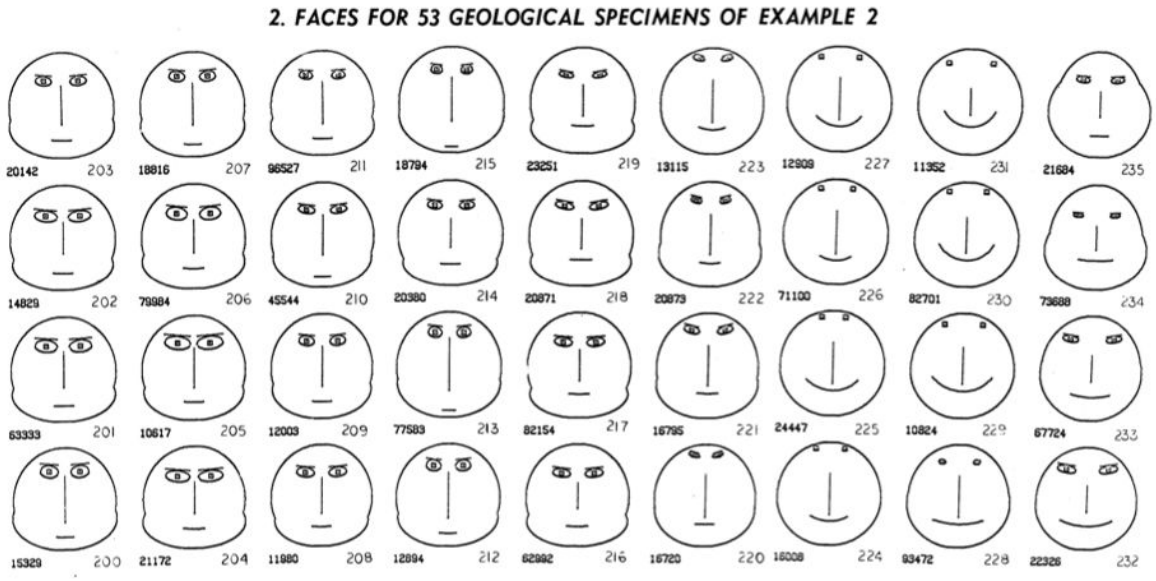


FIG. 2

Each face is encoding a different “entity” (could be a person but it could also be a “car”). Each aspect of the face encodes some piece of data. So the size or shape of the eye could be “miles per gallon” for the car, the size of the node could be “horsepower,” and the shape of the mouth could map to “price.” We’ll talk more about this when we get to multivariate encodings.

Instead of encoding the salary composition information to the height of bars, we are going to create chernoff face charts in this lab. Our chernoff faces have 4 components:

1. Face: a circle that encodes the `salary` information.
2. Right Eye (from the person’s perspective): a circle that encodes the `salaryA` info.
3. Left Eye: a circle that encodes the `salaryB` infoinformation.
4. Mouth: a filled quadratic Bézier curve that encodes `salaryC` information.
5. Colors don’t mean anything and you can feel free to change them. You can also be a little creative with the faces, but the main shapes/mappings should remain.

Figure 3 is what the final result will look like.

Figure 3 :chernoff faces for part 1.

Before we start coding, it’s better that we figure out where and how to draw each component of the chernoff face.

Step 1.1: Calculating attributes of faces, left eyes and right eyes.

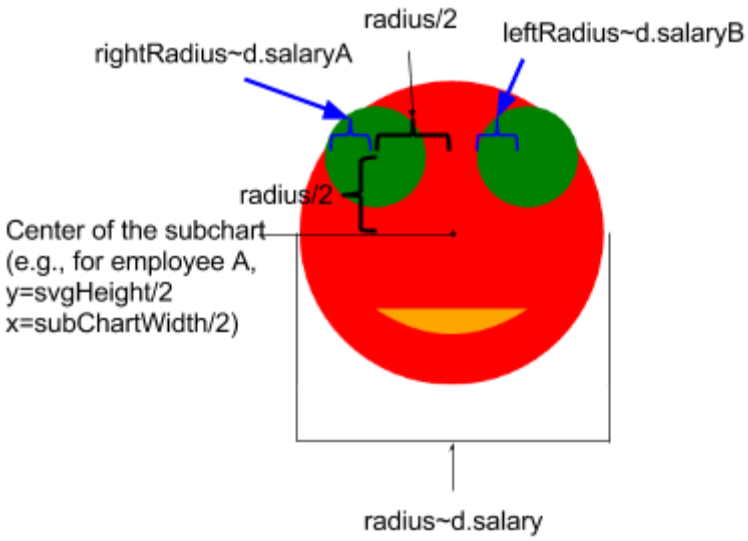


Figure 4 Positioning for one chernoff face (various features are proportional "~" to data (e.g., the right eye's radius is proportional to `salaryA`)

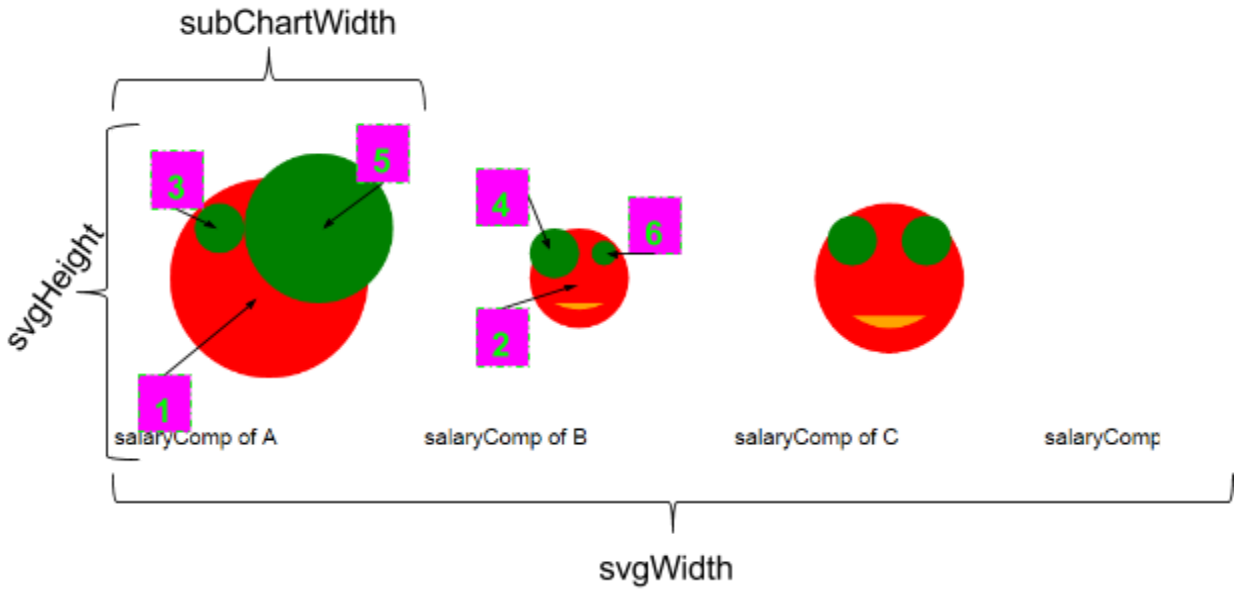


Figure 5: Positioning for multiple chernoff faces

Because salary, `salaryA` and `salaryB` are all visualized as circles, let's figure out the attributes for these first. Figure 4 shows the positioning for one chernoff face. Figure 5 shows

the positioning of multiple chernoff faces. Using this information, answer the following questions. Write your answers in the html file. **Write your answers in terms of these pre-defined parameters rather than hard code them.** We have filled cx for circle 1 and cy for circle 2 for you as examples. HINT: Put parentheses around your equations or store them in variables, otherwise plus signs are performing string concatenations.

1.1.1 For salary-face mapping:

- ☐ **Write the cx, cy and r value for circle 1 in Figure 5.**
- ☐ **Write the cx, cy and r value for circle 2 in Figure 5.**
- ☐ **If function (d,i) loops through the salaryData, write cx, cy and r value in terms of d and i.**

1.1.2 For salaryA-rightEye mapping:

- ☐ **Write the cx, cy and r value for circle 3 in Figure 5.**
- ☐ **Write the cx, cy and r value for circle 4 in Figure 5.**
- ☐ **If function (d,i) loops through the salaryData, write cx, cy and r value in terms of d and i.**

1.1.3 For salaryB-leftEye mapping:

- ☐ **Write the cx, cy and r value for circle 5 in Figure 5.**
- ☐ **Write the cx, cy and r value for circle 6 in Figure 5.**
- ☐ **If function (d,i) loops through the salaryData, write cx, cy and r value in terms of d and i.**

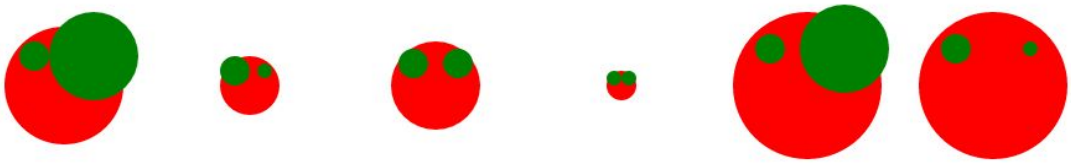
Hint: To answer 1.1.2 and 1.1.3, you need to know the radius of the faces.

Step 1.2 Filling in cx, cy and r equations for faces and eyes

We have set up the svg for you. Using the function you have found to complete faces, `leftEyes` and `rightEyes`.

```
var faces=chernoffFaceHolder.append('circle')
                        .attr("class","salary")
                        .attr('cx',function(d,i){
                            return 0
                        })
                        …….
 var leftEyes=chernoffFaceHolder.append('circle')
                      ……...

 var rightEyes=chernoffFaceHolder.append('circle')
                        …….
```

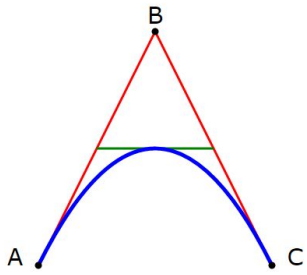
☐ **update the correct cx, cy and r values for the code block above.**
When you finish, you will see the following image (Figure 6):



(Figure 6)

Step 1.3: Calculating attributes of mouths

We want to map salaryC to the mouths of the chernoff faces. To do so, we will use <path></path> element. Path elements start with "M", follow by points that are separated by space. In our case, we are going to use **quadratic Bézier curve.**



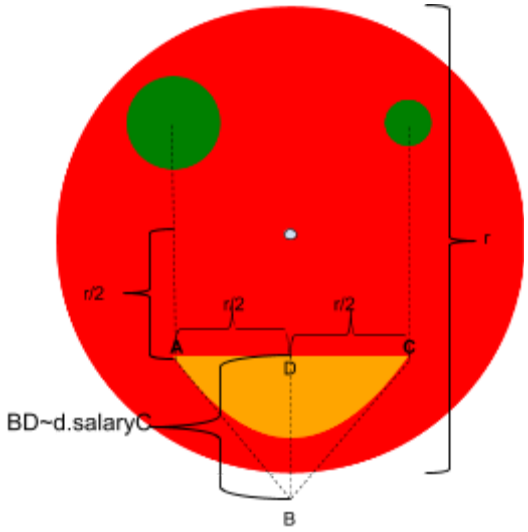
(Figure7, read more about paths in http://www.w3schools.com/graphics/svg_path.asp)

To draw the blue path in Figure 7, you will create an path element that looks like:

```
<path d="M 0 300 Q 150 0 300 300" stroke="blue"></path>
```

- "d" is the attribute that specify the type and location of the path
- "M" indicates the start of the path -- basically point A -- A 's coordinate is (0, 300)
- Q indicates that the path is a quadratic Bézier curve. If using lowercase q, the coordinates are relative to the previous point. -- B's coordinate is (150, 0); C's coordinate is (300,300)

In our chernoff faces, the positioning of the curve is shown in the following image(Figure 8).



(Figure 8)

☐ **Complete drawing mouths by updating the anonymous function for attribute "d" in the following code block:**

```
var mouths=chernoffFaceHolder
                        .append('path')
                        .attr("class","salaryC")
                        .attr('d',function(d,i){
```

.....

☐ **Add titles for chernoff faces by updating the variable `chernoffTitle` in the following code block:**

```
var chernoffTitle=chernoffFaceHolder.append('text')
                                    .text(.........
```

If everything works correctly, you will see Figure 3 on your webpage.

Hint: if you want more information on drawing paths/curves, see:
https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths

Part 2: Adding New Data And Updating the Chart

You just finished plotting chernoff faces with salary, salaryA, salaryB and salaryC. Employees just finished another project which gives them salaryD. In this part, we will update our data and charts with new information.

Step 1: Update data, range and scales

The variable `salaryD=[1,1.4,2,0,2.5,0]` stores the salaryD data of each employee. When we add this new field into our dataset, the salary, overunder will change. Also, the salaryRange, `salaryCompRange,overunderRange` will change. As a consequence, all the

scale functions that utilize these ranges need to be updated as well. We will make this a generic function in case we want to add another project (e.g., “project E”) later.

In the function `updateDataRangeScale(project,projectData)` , we are going to do the following things:

1. Add `project`(e.g., salaryD) data (stored in `projectData`) to each employee’s data under the key “salaryD”
2. Add `project`(e.g., “salaryD”) to the listOfProjects. Now it should contain [“salaryA”,”salaryB”,”salaryC”,”salaryD”]
3. Calculate the new total salary and store in `salaryData`
4. Find mean of each department
5. Add `overunder` information for each employee
6. Calculate `salaryRange`
7. Calculate `salaryCompRange`
8. Calculate `overunderRange`
9. Update `ChernoffFaceRadiusScale`.

☐ **Complete the function updateDataRangeScale(...). Note that most of these steps only require one line of code if you use the helper functions that we have provided.**

In the function `updateDataRangeScale()` , we have updated the `chernoffFaceRadiusScale` as:

```
chernoffFaceRadiusScale.domain([salaryCompRange.min,salaryRange.max])
```

☐ **In the line `document.write("<p>2.1 my answer is "+"yourAnswer"+"</p>")`, explain why we set the domain to be `[salaryCompRange.min,salaryRange.max]` instead of `[salaryRange.min,salaryRange.max]` or `[salaryCompRange.min,salaryCompRange.max]`.**

Step 2 Update faces and eyes with a button.

D3.transition() is a helpful function that let you create animations easily.

☐ **Uncomment the line**

```
d3.select("body").transition().duration(600).style("color", "red")
```

And you will see all the elements that directly stored in the <body> </body> have changed color. We will update the chernoff faces using transitions. The duration is set to be 800 but you can customize this value. We have implemented updateFaces as an example:

```
function updateFaces(){
    chernoffFaceDIV.selectAll("circle."+"salary")
```

```
                    .transition()
                    .duration(duration)
                    .attr('r',function(d,i){
                        return chernoffFaceRadiusScale(d.salary)
                    })
}
```

First, we need to select elements that we want to update. If we use `chernoffFaceDIV.selectAll("circle")` , we will select all the circles including faces, left eyes and right eyes, which is not what we want. Luckily, we have added “salary” as the class attribute/id/label for faces. Therefore, `chernoffFaceDIV.selectAll("circle.salary")` is going to give us all the face circles.

Second, we add transition by calling `.transition()` and `.duration(duration)`

Third, we adjust the attributes that have changed. For the faces, only the radius has changed because the salary information has changed. Therefore, we update ‘r’.

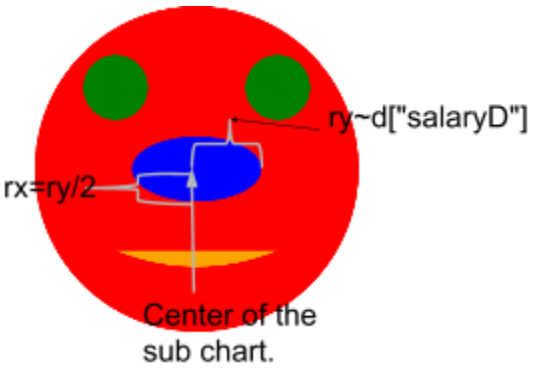
```
function update(){
    updateDataRangeScale('salaryD',salaryD)
    updateFaces()
    updateLeftEyes()
    updateRightEyes()
    updateMouths()
    addNoses()
}


chernoffButton=chernoffFaceDIV
                .append("button")
                .text("update")
                .attr("type","button")
                .attr("value","update")
                .attr("onclick","update()")
                .style("height",50)
                .style("width",100)
```

The code above sets up the button on the right of the svg element. After implementing Step 2.1 correctly, clicking on the button will display the salary transition from old salary data to new salary data.

☐ **Using updateFaces() as a template, implement updateLeftEyes(), updateRightEyes() and updateMouths(). When you finish, click on the update button to see if all elements are adjusted accordingly.**

Step 3 Add salaryD as noses

We are going to map salaryD to noses using the <ellipses> element. The attribute configuration for noses are shown in Figure 9.



(Figure 9)

☐ **Complete the function `addNoses()`. Now the update button should add noses to our charts. If everything works correctly, you will see the following charts (Figure 10)**:

(Figure 10) ‘

Part 3: Adding ToolTips

By just looking at these faces, it quite hard to estimate the salary information. We might be able to see that E has higher salary than employee D, but it is difficult to know whether E has made 4K or 6K. To make it easier, let’s add some tooltips so that when we hover over these faces, more information is displayed.

To demonstrate an example, we have added tooltips for you. Uncomment the code block within the part 3 tag(////////Part3 tooltip/////////////// and ///////////part 3 end/////////////////). Read the code along with the handout.

We add tooltips in three steps. 1) add tooltips as a div element that is not displayed. 2) write functions to change tooltip visibility when mouse events are triggered. 3) add event listeners to the faces (circles, paths, and eclipses).

Step 3.1 Add tooltips as a div element that is hidden(display=none)

We have done this step for you in the following code block: var

```
chernoffTips=chernoffFaceDIV.append("div")
                                .attr("class",'tooltip')
                                .style("display","none")
                                .style('position',"absolute")
                                .style("fill","black")
```

Within this code block, `.style("display","none")` specifies that the tooltip is not display. When the mouse hovers over desired elements, display can be set to “inline” so that the tooltip is displayed.

Step 3.2 Write functions to change tooltips when mouse events are triggered

There are three mouse events that we are interested in. mouseover, mousemove and mouseout. Read here: http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_event_mouseenter_mouseover for more information about these three events. For our specific case,

- mouseover controls what happens when the mouse just enters the selected element. In response, we should display the tooltip. Therefore, in handleMouseOver(), we set display to be inline.
- Mousemove controls what happens when monse is moving inside the selected element. In this case, we should set the tooltip to display correct information. Therefore, in handleMouseMove(d,i), we update the text in the tooltip, as well as the location of the tooltip.
- Mouseout controls what happens when mouse move out of the selected element. In response, we should hide the tooltip. Therefore, we set the display to none again.

Step 3.3 Add event listeners to the faces (circles, paths, and eclipses).

After figuring out what to do when an event happens, we will tie these event listeners to elements of our charts, including circles, paths and eclipses. For instance.

```
chernoffFaceDIV.selectAll("ellipse")
               .on("mouseover", handleMouseOver)
               .on("mouseout",handleMouseOut)
               .on("mousemove", handleMouseMove)
```

ties mouseover, mouseout and mousemove event listeners to all ellipse elements under the chernoffFaceDIV.

☐ **When you update and hover over the noses, the tooltip disappears. Why is that? Can you *place the code block above in appropriate location* so that the tooltip for salaryD will be displayed when hover over?** (HINT: think about what are you selecting and when are you selecting them)

Part 4: Updating The Bar Charts

Now that we have finished updating the Chernoff face charts let's go back and try to fix up the bar charts a bit so that they are more readable.

Step 4.1 Adding bars for salaryD

☐ **Complete the function `addSalaryDbars()`, which adds bars for the salaryD (HINT: if you use the helper function `plotEachBar()`, this can be done with one line of code.)**
☐ **Call `addSalaryDbars()` in `update()` so that when you click on the update button, new bars will be added.**

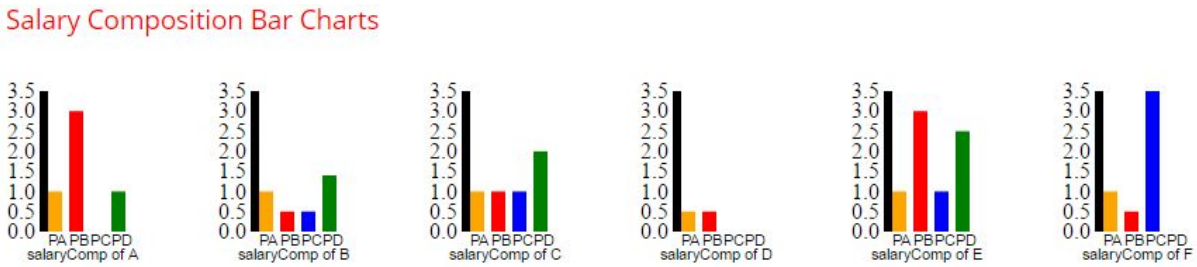
Step 4.2 Adding a y-axis

We will add y-axis for each salaryComp bar chart by completing the scale of the yAxis and the transform attribute in the following code block. We have created a d3 linear scale for you that is called "`yAxisScale`" which has the "reversed" range in comparison to salaryCompHeightScale . You have to modify two lines. In the first, you need to feed in the right "scale" (the function that maps value to coordinate). For the second part, you're going to have to tell d3 where to draw the scale. Modify the translate line to do this.

```
var yAxis = d3.svg.axis()
                  .scale()//your work here
                  .orient("left")

salaryCompG.append('g')
           .call(yAxis)
```

```
.attr('transform',function(d,i){
    return 'translate('+0+','+0+')'//your work here
}));
```

If everything works correctly, you will see the following charts after clicking the update button. Hint: If your y-axis is not correctly aligned, think about which point is actually translated here.



(Figure 11)

If you did this manually you would have to add lines, ticks, text, etc. Thankfully, d3 does a lot of this for you (this relates to the idea of "layouts" that we will use in the next lab).

Hint: take a look here: http://alignedleft.com/tutorials/d3/axes if you're stuck

Step 4.3 Adding tooltips

After updating the bar charts, we will also add tooltips to the bar charts.

☐ **Using the same procedure as shown in Part 3, add tooltips to the bar charts. Make sure your tooltips are displaying for salaryD bars as well.**

Part 5: Chernoff Faces Pros/Cons

You now have created two different representations of the same data and can start thinking about the expressiveness/effectiveness of each. We'll talk a lot more about chernoff faces in a later lecture, but for now:

☐ **Answer this in the space provided in the HTML file: What do you think the pros and cons of using this kind of representation? Are there datasets that are more or less appropriate for this kind of encoding?** You can answer relative to the bar charts if it helps to be concrete.