CS/INFO 3300; INFO 5100
Homework 5
Due 11:59pm Wednesday, October 6

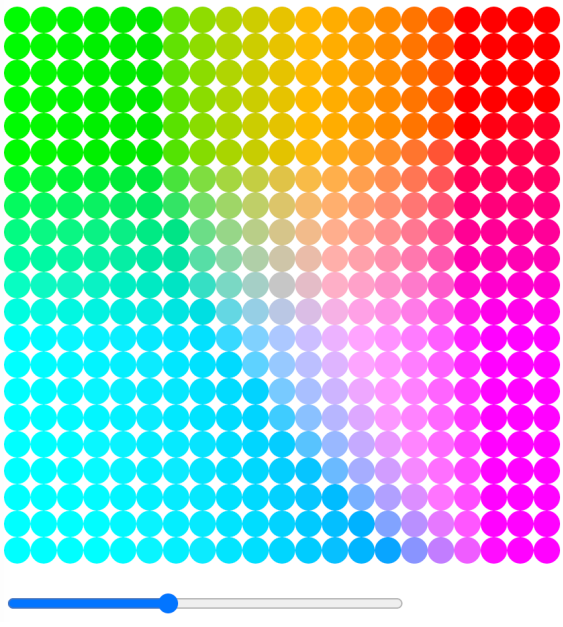
Goals: Practice using d3 to create some simple charts. Get more experience importing data and working with data joins.

Your work should be in the form of an HTML file called index.html or index.htm with one <p> element per problem. If you must add any SVG canvases programmatically, we suggest that you add a element to the HTML to hold each SVG to make positioning a bit easier. For this homework we will be using d3.js. **In the <head> section of your file, please import d3 using this tag: <script src="https://d3js.org/d3.v7.min.js"></script>**

Create a zip archive containing your **HTML file and all associated data files** (such as wines.json) and upload it to CMS before the deadline. Submissions that do not include data files may be penalized. Your submission will be graded using a Python web server run in a parent directory containing your zip file contents along with many other students' submissions.

(e.g. the grader has a folder, `~/hw5_cms`, where they start their own web server and unzip your assignment. If you just zip up `index.htm` and your data files as we ask, CMS will automatically make a sub-directory for your netID when we download it from the system. This means that your homework would appear at `~/hw5_cms/your_netID/index.htm` on the grader's computer. In their web browser, it would show as `http://127.0.0.1:8000/your_netID/index.htm` in the URL bar.)

1. For this problem, we will be making use of a color space available in d3, LAB. Unlike traditional HSV (hue, saturation, value), LAB (luminosity, a, b) more closely estimates how humans perceive colors. You will design an interactive visualization of the color space, illustrating what the two chroma dimensions, A and B, are about.



A. After a <p> tag for problem 2, place a **square SVG element 420px in height and width**.

In a `<script>` tag, first write code which creates a single array containing Objects (which you will use as dictionaries). Each object should have a **"a"** property (i.e. key/value pair) ranging in value from **-160 to 160** and a **"b"** variable also **ranging from -160 to 160**, evenly spaced in **multiples of 16 (including -160 and 160)**. This means that you should have 21 different **"a"** values for every **"b"**, and visa versa. Every combination of A and B ought to be represented in the array, giving you a total of **441 objects** in your array. You'll later use a slider to handle the luminosity in LAB.

(hint: use a nested `for` loop structure to create objects with **"a"** and **"b** values for your array)

B. Your goal is to spread 10px radius circles evenly across the canvas in a grid, as seen in the image above. To begin, create two scales, `aScale` and `bScale`, for `"a"` and `"b"` values. You will use these to compute `cx` and `cy` for each circle in the grid. `"a"` values with go from -160 on the left edge to 160 on the right edge. `"b"` values will go from -160 on the bottom to 160 on the top. Set the domain of each scale to match the domain of values you created in step A. Set the range of values to span the entire canvas (hint: remember that circle `cx` and `cy` are for the *center* and not the corner – your scales probably should not start at pixel (0,0) if you want the circles to just touch the edge of the canvas).

C. Create a function, `showCircles(luminosity)` that uses a new or old style d3 "**data join**" (i.e. selectAll(), data(), enter(), attr(), and style() functions) to create or modify one circle for each object in the list. Make sure that you account for both creating new circles as well as updating ones that may already exist. Set the **radius of each circle to 10** and and **do not give each circle a stroke**. Use your scales to create a grid of circles. Done properly, the circle centers will be 20px apart and fill the entire canvas. Due to floating point error, you may see very small decimal values if you inspect the HTML.

Set the fill of each circle to a LAB color specified by the circle's assigned `"a"` and `"b"` values, with the `"l"` value supplied as the `luminosity` parameter of your `showCircles` function. Make sure you use the features provided by your data join, and not a regular `forEach` loop. You may want to use d3.lab() to create the color rather than trying to do the color conversion manually. **Check out the d3-color documentation to make sure you are constructing the color correctly. Done right, you can insert the output of d3.lab() right into a fill function.**

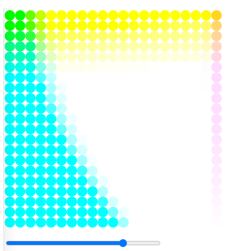
Now test your function by running `showCircles(80);` in the console. It should resemble the example.

D. Finally, add a **slider input** (docs) so the user can choose a **lightness value**. You can hardcode this slider right into the HTML of your page. This slider should range in value from **0 to 200 with a step size of 1**. Use d3 to attach an event listener to the "**input**" event for the slider to call **showCircles** with the **current lightness value of the slider**. The data join within showCircles should then do the hard work of updating the visuals to match. Be sure to choose a reasonable default value for the slider.

Hint 1: `d3.select('yourSlider').on('input', function() { console.log(this.value) });`

Hint 2: If nothing shows up until you move the slider, think about *when* `showCircles()` is being called. You may need to modify your code to *initialize* the visualization prior to use of the slider.

Hint 3: Double check your HTML after moving the slider a little bit. You must **verify that your data join is modifying the existing circles** every time you move the slider. If you have constructed your data join incorrectly, then it will repeatedly make new circles ad infinitum. This is bad.



Hint 4: Due to the way that LAB colors work, it is expected that some circles will max out at white for high luminosity values, making the grid appear patchy when the slider is close to 200.

NEXT PAGE

2. You've now seen a few scatterplots in class as well as developed your own in HW4. In this problem you will once again be developing a scatterplot, however this time many of the specific **design features of the plot will be left to you**. Inside the homework ZIP file you will find **wines.json**, a subsample of a larger dataset of wine reviews sampled from the web (source). We have sampled wine scores from four US wine regions: New York, Washington, Oregon, and California. Please use this dataset to complete this homework. For each of the following sub-problems, use a **<p>** tag as directed to briefly **explain your procedure or design rationale for that step** (e.g. How did you decide on axis scale labels? What made you choose a linear/log scale? What compromises did you make?).

Your goal in this problem is to create a plot that can help to answer two questions:

- Is there a relationship between the `price` of wines and their `point` scores (i.e. quality)?
- Are some wine regions better or more expensive than others?

A. This data file isn't exactly perfect. In fact, we've gone ahead and **added some fake points with confusing, missing, or bad data values**. Worse, we've not bothered fixing any types or standardizing values. Begin by loading the data file using `d3.json`. Feel free to use `d3.autotype` if you'd like, but it may not convert everything cleanly. You may need to loop through and manually convert strings into numbers -- after cleaning up some pesky characters so you don't get NaNs.

Use **filter** and a **forEach** loop to **hide or correct any important data quality issues**. Describe what data issues you **found** and **how you fixed them** in your **<p>** tag. Please write an **exact count** of the number of bad data items you removed. Additionally, after filtering, please print the length of your data array to the console to prove that you've got the right number of remaining points.

B. Create an SVG object that is **800 pixels in width and 500 pixels in height**. Give it an ID so that you can use selectors to find it later. You are going to be graphing **`price` on the x-axis**, showing **`points` on the y-axis**, and **coloring them by state**. In this step, please **find extents and construct scales for your chart**. Create **<g> elements**, use **translate** to move them to an appropriate place, and **populate them with `d3.axis` labels**. Use a **second set of `d3.axis` objects** to create **corresponding x- and y-axis gridlines** in a light color. You must use `d3.axis` for your gridlines.

Feel free to choose whatever margin/padding values, domain, range, log/linear scales, colors, and axis formatting styles you like. Use the **<p>** tag to **explain the choices you made** in designing the axes/scales. We will reward scales/axes that are legible, show the distribution of data clearly (or as clearly as possible), use color effectively, and avoid visual clutter. **A good rationale in the <p> for your design will outweigh problems in these areas.**

C. Create a **<g> element** and **translate** it so that it can act as your **main chart region**.
Begin by **creating a new function, `jitter()`, which returns a random number between -4 and 4** (hint: check out what you did in HW4). Now, populate the chart with **<circle> elements corresponding to valid data points**. Move, scale, and color them as necessary using the structures you built in step B. When you are positioning the circles, **add a random number from your jitter function to both the x and y position**. This will reduce the amount of overlap when circles have the same point and price. You may still want to reduce the opacity of circles if there is heavy overlap. In addition to **writing**

about your design rationale for the choices you made in creating circle elements in your <p> tag, please also write 2-3 sentences identifying 1 advantage and 1 disadvantage of this jittering approach to managing overlap.

D. Using `d3.on("mouseover", ...)` and `d3.on("mouseout", ...)`, provide users a way to move their mouse onto points and see the "title" of the wine they are hovering over. At minimum, points should grow in size and a floating <text> label should appear nearby when the mouse enters the inside of a circle. Your label can be either in a fixed position in the canvas or a variable position near where your mouseover is located. A corner is fine, so long as the label and a circle do not experience a race between `mouseover` and `mouseout` due to overlap (this will appear as a flashing label). Do not use HTML elements; your event actions must happen entirely within SVG canvas elements. When the mouse leaves the circle, it must return to its normal appearance. Feel free to add more complexity to make the highlighting of points more obvious or the text more legible, but this is not explicitly required. Write 2-3 sentences in your <p> tag describing the possible benefits to users from this approach and identifying places in the chart where it may be ineffective or confusing (or, if you fixed them, what you did to improve the user experience).

E. Finally, create a simple legend for the chart with mouseover filters. First, add a element underneath your SVG for your legend labels. For each of the different states in the dataset, add a tag for that state. Color the text with the corresponding color in your chart. (hint: if you use a scaleOrdinal, you can access scaleOrdinal.domain() after you fill a bunch of points – you might consider a *loop* that adds elements and colors them

Using `d3.on("mouseover", ...)`, make it so that whenever you mouse over a label, the corresponding circles on the charts appear with high opacity and the other circles have reduced opacity (hint: data joins give you access to the data embedded every circle – think about how to configure a `.style(...)` call to use data *conditionally*). Additionally, add a final element labeled "Clear" in black that returns all of the circles to their original opacity when you mouse over it. Note: you do not need to do anything special with your mouseover labels when a filter is applied – it is okay if you can see the names of points that are filtered. You can talk about design elements in your <p> tag, but you are not required to add anything for this sub-problem.