CS/INFO 3300; INFO 5100
Homework 5
Due 11:59pm Wednesday, September 27

Goals: Practice with color scales. Create charts using loops and data joins.

Your work should be in the form of an HTML file called index.html with one `<p>` element per problem. Wrap any SVG code for each problem in a `<svg>` element following the `<p>`.

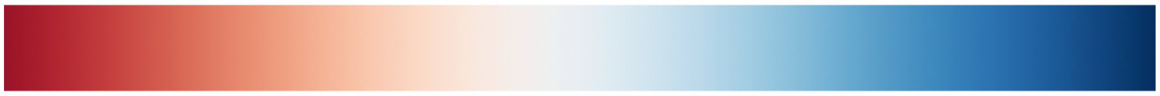
For this homework we will be using d3.js. In the <head> section of your file, please import d3 using this tag: <script src="https://d3js.org/d3.v7.min.js"></script>

Create a zip archive containing your **HTML file and all associated data files** (such as NCAA_shots.csv) 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 (e.g. server started in `~/student_hw`, with your homework at `~/student_hw/your_netid/hw5/index.html`) – be sure that it works. Check the September 11th notes file for more comments on file paths.

(next page)

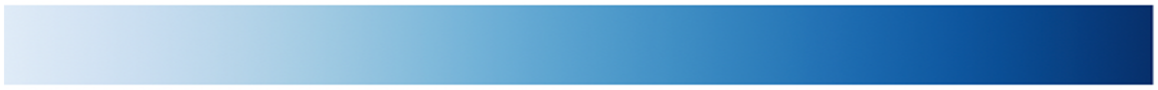
1. Instead of a `<p>` element, for this question please create a `<ul>` element. For each of the following scales, create a `<li>` sub-element and answer the following questions (5pts each): *(If you have a color vision deficiency and cannot perceive hues in a color scale in order to answer a subitem, please instead describe what you see.)*

A:



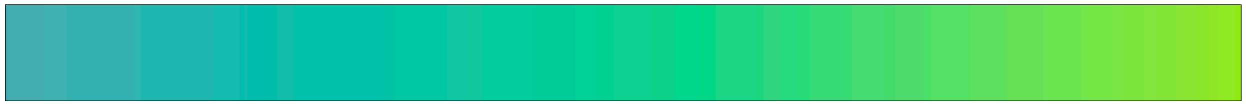
Is this a **sequential** or a **divergent** scale?
Is this an **effective** sequential/divergent color scale? Justify your answer in **1-2 sentences**.

B:



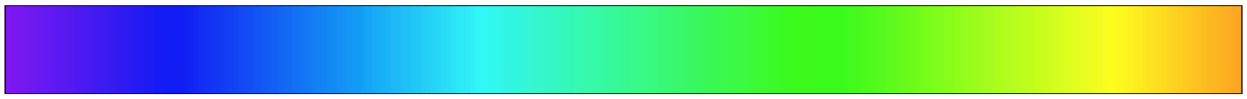
Assume that this scale is applied to a **numeric data attribute ranging from -1 to 1 representing sentiment (e.g. dislike to neutral to like),** with negative values moving towards the dark blue end and positive values moving towards the white end. Middle values remain sky blue. Is this an **effective color scale for this task**? Justify your answer in **1-2 sentences**.

C:



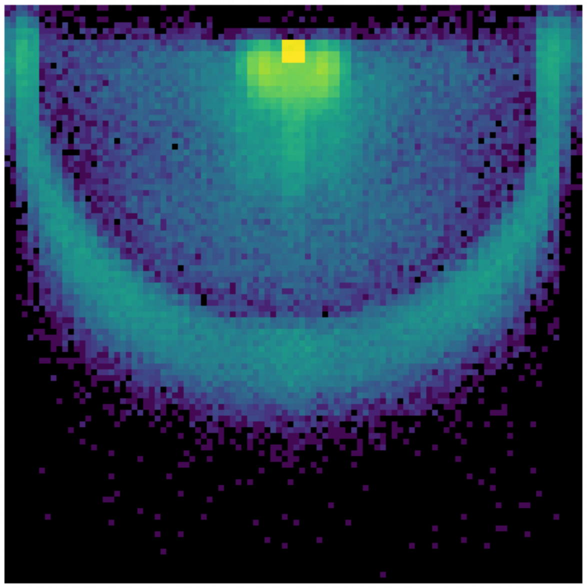
To most individuals, this color scale will appear to vary in both hue and luminosity (grey-ish blue on the left, lime green on the right). However, the hue channel of this scale is not visible for individuals with certain color vision deficiencies. This poses usability issues. Use an online color blindness image testing tool to identify and list **which kind(s) of anomalous trichromatic and/or dichromatic color vision deficiencies (e.g. deuteranomaly)** would cause a loss of perceivable hue variation (file included in ZIP).
[If you have color vision deficiencies that make the scale's hue hard to interpret, you have two choices: You can either a) self-disclose your color vision deficiency and describe what the image looks like to you, or b) ask a trusted friend to describe what they see to you.]

D:



A data scientist is designing a choropleth map for a new **continuous, numeric county-by-county "average life expectancy" data attribute** they developed. **Would you recommend that they use this rainbow scale to color the counties in their map?** Justify your answer in 1-2 sentences.

2. For this problem we have processed a public dataset of NCAA Basketball games from the 2021-22 and 2022-23 seasons (aggregated from data scraped by Luke Benz). Our goal is to recreate a popular heatmap visualization made by Max Woolf. The dataset contains 159,410 successful and unsuccessful shots made by players whose position was logged during the game. You can see our final version of it to the right. Areas of high activity in the game are colored yellow and areas of low activity are black. Marks are individual `<rect>` squares and channels are aligned position and color hue+luminosity.

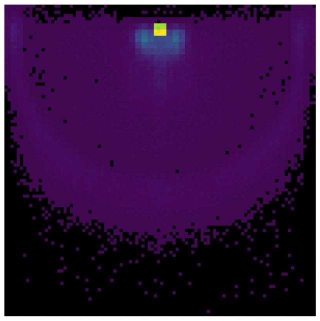


To make this visualization we have added up all the shot attempts made by players at different locations in the court. A shot attempt refers to when a player throws the ball attempting to get it into the opposing team's basket to score points (where and when they can throw is influenced by rules and team strategy). You can see the basketball hoop in the middle top and an arc created by the "3 point line" that dictates where players can throw the ball to earn more points.

A. In your HTML, create **a 500x500px SVG element**. Use CSS styles to give it a **black background**. Now load the included data file `NCAA_shots.csv` by using an **asynchronous request** (i.e. `d3.csv().then()`). Implement the rest of this problem in the promise function. Use `console.log()` to check out the data you're using for this problem. You will notice that each element contains x and y values for making the colored rectangles at various positions in the visualization. The other keys contain different kinds of count data about what shots happened in a specific area of the court. For this assignment we'll start by examining the total number of all successful and missed shots at a location: `attempt`.

First create a **new sequential color scale** for the heatmap. Use `d3.extent()` to **figure out the extent** of `attempt` in the dataset. Then, make a sequential color scale using that as your domain. Use the `d3.interpolateViridis` color scale in your sequential scale (hint: docs).

Finally, use a `for` or `forEach` loop to **create new `<rect>` elements for each row** of data in your dataset. Each data object has an `attempt` key which you can **run through your color scale to find a fill color** for your rectangle. Rects also need `x`, `y`, `width`, and `height`. While our SVG is 500x500px in size, the x and y positions in the dataset range between 0 and 100. Use some arithmetic to **make 5x5 rectangles to completely fill the canvas** with color. Please note that at this stage, **your final visualization will not look like the top example image**. It will be mostly dark purple with one bright yellow rectangle:



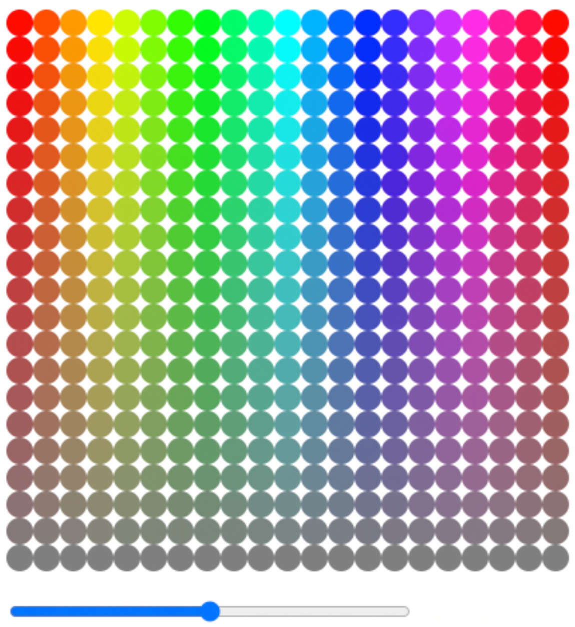
B. There is something odd with the visualization you've created. If you've done it properly, you should see an intensely yellow blob and not much else. This is because the data have an *exponential distribution*. Close to the basket there are many, many more attempts than far away. This causes the color scale to assign yellow to a large value while all other values are so small that they receive virtually the same color at the bottom of the scale. One common approach for resolving this issue is to use a logarithmic scale instead of a linear scale (which scaleSequential uses). While there are ways to do this with d3 scales, they are needlessly complex. Instead, we have provided for you another data attribute: `log_attempt`.

Adjust your code so that you use `log_attempt` instead of `attempt` for your rectangle fill color. You should only need to change your `d3.extent()` call and `"fill"` setter.

Compare that result with your previous visualization. In 2-3 sentences in your <p> tag, please describe one advantage and one disadvantage of the logarithmic color scale as compared to the original, linear scale.

You do not need to submit both versions of #2. Only submit the version that uses `log_attempt`.

(next page)



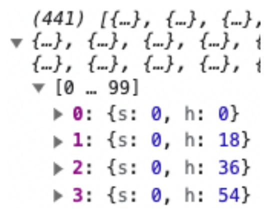
3. For this problem, we will be making use of a color space available in d3, HSL. This is another name for the traditional HSV (hue, saturation, value) color space, which has issues but does produce visually appealing color charts. You will design an interactive tool for exploring the color space, giving you a sense for its benefits and problems in displaying colors perceptually.

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 that creates a single array containing Objects. Each object should have a **"h"** property (i.e. key/value pair) ranging in value from **0 to 360** and a "**s**" property **ranging from 0 to 100**. The **"h"** values should be evenly spaced **in multiples of 18 (including 0 and 360)**, giving you 21 different **"h"** values for every **"s".** The 0 to 360 value for **"h"** will control the hue angle (hence 0 to 360 degrees around a circle). The "**s**" values should also be evenly spaced in **multiples of 5 (including 0 and 100)**, giving you 21 different "**s**" values for every "**h**". The 0 to 100 value for "**s**" will control the saturation (color intensity). Every combination of h and s ought to be represented in the array, which will give you a total of **441 objects in your final array**. You'll later use a slider to handle the "**l**" part of the HCL color space.

(hint: use a nested `for` loop structure to create objects with H and C values for your array)

If you create your array properly, a `console.log()` output of it should look something like this:



```
  (441) [{…}, {…}, {…},
▼ {…}, {…}, {…}, {…}, {
  {…}, {…}, {…}, {…}, {…}, {
  ▼ [0 … 99]
    ▶ 0: {s: 0, h: 0}
    ▶ 1: {s: 0, h: 18}
    ▶ 2: {s: 0, h: 36}
    ▶ 3: {s: 0, h: 54}
```

B. Create a function, **showCircles(lightness)** that uses a d3 "**data join**" (i.e. selectAll(), data(), join(), attr(), and style() functions) to create or modify one circle for each object in the list. Feel free to use either the old style or new style of data join. Set the **radius of each circle to 10px** and **do not give each circle a stroke**. Your goal will be to spread these circles evenly across the canvas in a grid, as seen in the image above.

Please **space circles 20px apart in a grid** as seen in the image above. Vary "**h**" on the x-axis (data values of 0 on the left, 360 on the right) and vary "**s**" on the y-axis (data values of 0 at the bottom, 100 at the top). We suggest that you use scales to make your circle placement easier. Both scales will have identical ranges – but be careful to accommodate the fact that you are assigning cx and cy for the circles and **use your range to pad accordingly**. Do not let circles get cut off by the sides of the canvas! If you set your scales properly, the circles will just be touching each other and use the entire canvas.

Set the fill of each circle to an HSL color specified by the circle's assigned hue angle and saturation (with the lightness value between 0 and 100 supplied as a parameter to 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.hsl() 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.hsl() right into a fill function. **(Hint: d3.hsl expects s and l to be between 0 and 1; you may need to do division)**

Now test your function by running `showCircles(80);` and seeing if a grid shows up. It should resemble the example image.

C. Finally, add a **slider input** (docs) so the user can choose a **lightness value**. This slider should range in value from **0 to 100 with a step size of 1**. Use d3 to attach an event listener functions 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 (the example image uses 50 as its default value).

Hint: 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 repeatedly makes new circles ad infinitum. This would be bad.