CS/INFO 3300; INFO 5100
Homework 3
Due 11:59pm Friday, March 5

Goals: Practice using d3 to create SVG elements and set their aesthetic properties. Recognize the effect of data transformations through direct data changes and through scale functions. Practice working with color scales.

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> element. 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.v6.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 will be penalized. Your submission will be graded using a Python web server run in the directory containing your zip file contents – be sure that it works.

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 green end and positive values moving towards the light tan end. Middle values remain teal. Is this an **effective color scale for this task**? Justify your answer in **1-2 sentences**.

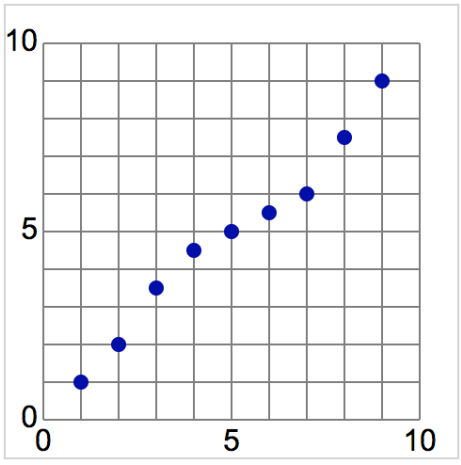
C:

To a majority of 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. In HW2 you reproduced a plot from scratch using SVG. Now **create the same plot again**, but this time use **d3 functions to create it programmatically in a `<script>` tag**. While it should resemble the example image to the right, you don't need to recreate it exactly, so long as your point and line positions are correct.



Create a **240x240 pixel SVG element in HTML**. Use a CSS style to give the canvas a **1px light grey solid border**. The main plot region, excluding labels, should be a square 200x200 pixels in size, running from (20,20) to (220, 220). Reserve the remaining pixels as padding for the labels.

On the last page of this assignment we have included a code version of the dataset. Go ahead and copy it into your `<script>` tag. First create `x` and `y` **scale functions** that map from data coordinates to SVG coordinates, using the same minimum and maximum values as the chart *domain* (0 to 10 for both axes). Remember to account for the "padding" pixels when

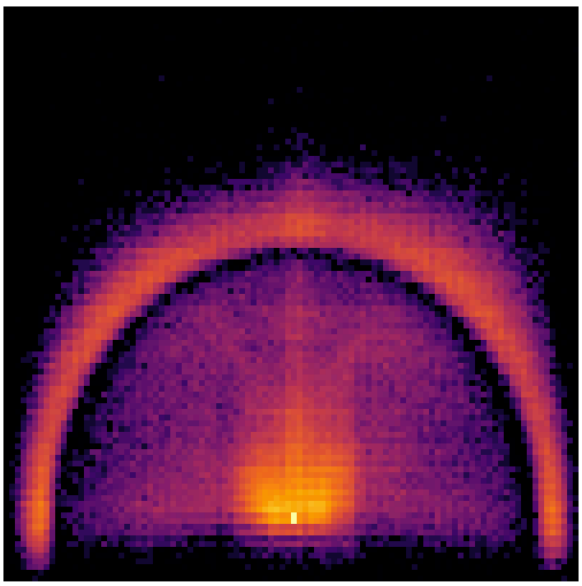
determining the *range* of point positions. You can choose whether to implement the margins by adjusting the range of your scales or adding a `<g>` element and `translate()`.

Next, create the **grid of lines for your chart**. While there is a way to make gridlines using d3 axes, please **manually create gridlines using a `for` loop**. You should create one horizontal line and one vertical line for each number between 0 and 10 (inclusive) in a grey color.

Afterwards, **add the text labels programmatically**. There are a few ways to do this. You could create an array containing the values that need labels (e.g. `[0,5,10]`) and then loop through it with `forEach`. You also could do some clever modulo arithmetic while you are looping to make lines. In either case, make a new `<text>` element for each label with **Arial font**. Use your scales to help place the text and adjust the `dominant-baseline` and `text-anchor` attributes like you did in HW2. You may need to add or subtract a few pixels to position the text nicely.

Now, add **circles** for each point with positions determined by your scales. You don't need to use a data join; it's fine if you just create circles one-by-one in a `forEach` loop. Circles should have a radius of 4px and be colored dark blue.

3. For this problem we have processed a public dataset of NCAA Basketball games from 2019 (Google Cloud & NCAA® March Madness Analytics). Our goal is to recreate a popular heatmap visualization made by Max Woolf. You can see our version of it on 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 of the 2 point and 3 point 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 bottom and an arc created by the "3 point line" that dictates where players can throw the ball to earn more points.

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`, `y`, `width`, and `height` values for making the colored rectangles. 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`.

We have one challenge to tackle first. The dataset comes with x and y positions ranging from 0 to 100, but our SVG is 500 pixels in size. We need to adjust the `x`, `y`, `width`, and `height` values so that they match. At the top of your promise function, use a `forEach()` loop to **alter the data**. For each point in the dataset, **multiply `x`, `y`, `width`, and `height` by 5** so that they range from 0 to 500. You could also do this with scales, but this is far more convenient.

Now 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.interpolateInferno` 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. As the dataset already now has correct `x`, `y`, `width`, and `height` values, configuring the rect elements should be pretty straightforward. **Adjust the fill of the rectangles** using your color scale and the `attempt` value. Please note that at this stage, **your final visualization will not look like the example image**.

3B. There is something odd with the visualization you've created. If you've done it properly, you should see **some intensely yellow blobs 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, causing your color scale to crunch away all of the detail in less popular areas of the court. 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 #3. Only submit the version that uses `log_attempt`.

(see last page for data for problem 2)

Data for scatterplot in #1

	X	Y
p1	1.0	1.0
p2	2.0	2.0
p3	3.0	3.5
p4	4.0	4.5
p5	5.0	5.0
p6	6.0	5.5
p7	7.0	6.0
p8	8.0	7.5
p9	9.0	9.0

For easier import into your code:

```
const data = [{"x":1.0 ,"y":1.0},
              {"x":2.0 ,"y":2.0},
              {"x":3.0 ,"y":3.5},
              {"x":4.0 ,"y":4.5},
              {"x":5.0 ,"y":5.0},
              {"x":6.0 ,"y":5.5},
              {"x":7.0 ,"y":6.0},
              {"x":8.0 ,"y":7.5},
              {"x":9.0 ,"y":9.0}];
```