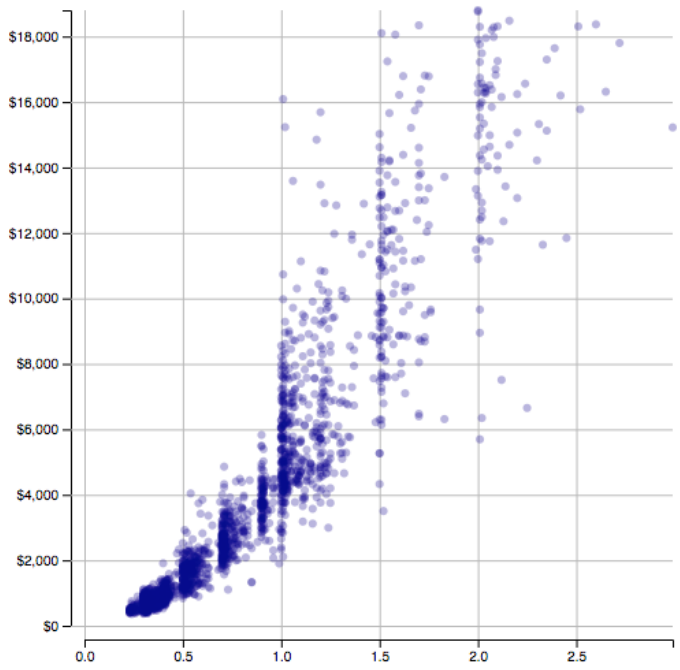
CS/INFO 3300; INFO 5100
Homework 5
Due 11:59pm Friday, March 19
(note: while you have ~1.5 weeks, this assignment is intended to take 1 week to complete)

Goals: Practice using d3 to create SVG elements and set their aesthetic properties. Work with scales some more, and practice data joins. Use events to begin making interactive charts.

Your work should be in the form of an HTML file called index.html with one <p> element per (sub)problem. Wrap any SVG code for each problem in a <svg> element following the <p> element(s). 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 plus associated data files (such as diamonds.json) and upload it to CMS before the deadline.

1. In this problem you will plot some data about diamond sales. The file `diamonds.json` contains a JSON block that defines an array of objects. Each object is a diamond sale. These have been randomly sampled from a much larger dataset. In addition to numeric columns for price and size (carats), the dataset contains a color rating value where 1 is the best value and additional measures of quality.



A. Load the data using an asynchronous `d3.json` request. Implement the rest of this problem in a single promise function and <script> tag. Create a **500x500px SVG element** using d3 functions.

We will use the following block for margins for this chart:

```
margin = { "top": 10, "right": 10, "bottom": 40, "left":60};
```

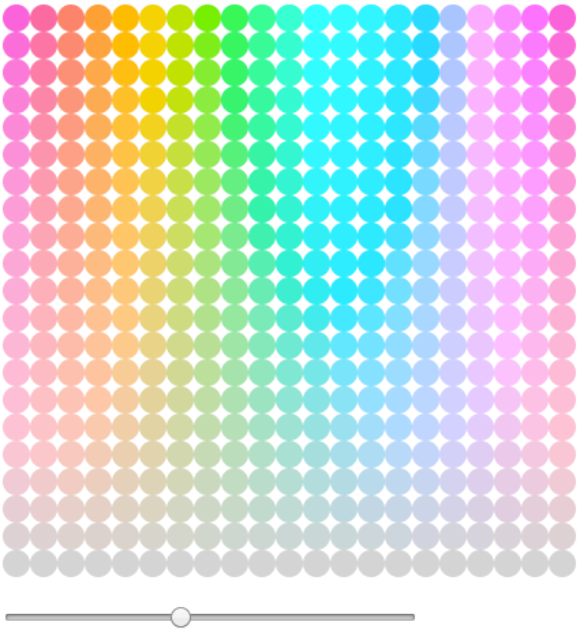
For this chart we will graph `price` on the y-axis and `carat` on the x-axis. Create some **linear scales** that account for the aforementioned margins. **Hardcode the start of the domain for both scales to 0** and end at the maximum value for each respective attribute (we often set 0 as a fixed point because users assume charts start at common reference points). Choose the range attributes to be appropriate for the size of your plot.

B. Create **<g> elements**, use **translate** to move them to an appropriate place, and **populate them with d3.axis labels**. For your **x-axis labels**, **please set `ticks(5)`** so that d3 assigns 5 labels, each on 0.5 carats apart. For your **y-axis labels**, please use `tickFormat` to give them a **leading $ and comma separators** (e.g. $18,000) 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 labels and gridlines**. Don't forget to add the necessary CSS to style your gridlines properly.

C. Finally, use an old or new style **d3 data join to create circles** for your datapoints. Each circle should be positioned properly using your scales. Set each circle's **radius** to `3px`, its **opacity** to `0.3` , and its **fill color** to `"darkblue"`.

D. Finally, in your <p> tag, **write 2-3 sentences about the relationship between price and carats in the diamonds** in the dataset. Is the relationship relatively linear? Non-linear? Are there any discontinuities or unusual elements in the relationship?

2. For this problem, we will be making use of a color space available in d3, HCL. Unlike traditional HSV (hue, saturation, value), HCL (hue angle, chroma, lightness) more closely estimates how humans perceive colors. You will design an interactive tool for exploring the color space, giving you a better idea of why HCL produces more perceptually uniform colors across different hues.



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 "**c**" 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 **"c".** The 0 to 360 value for **"h"** will control the hue angle (hence 0 to 360 degrees around a circle). The "**c**" values should also be evenly spaced in **multiples of 5 (including 0 and 100)**, giving you 21 different "**c**" values for every "**h".** The 0 to 100 value for "**c**" will control the chroma (color intensity). Every combination of h and c 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)

B. Create a function, **showCircles(lightness)** 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. 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 "**c**" 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 HCL color specified by the circle's assigned hue angle and chroma values, with the lightness value 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.hcl() 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.hcl() right into a fill function.**

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 200 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.

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.