CS/INFO 3300; INFO 5100
Homework 4
Due 11:59pm Wednesday, September 21

Goals: Practice with visual channels. Create a chart for a dataset using d3 built-in scales, axes, and drawing functions. Run through how SVG transforms and chart layout work.

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>**.

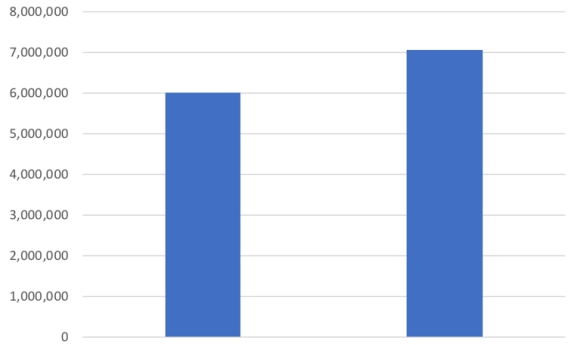
For this homework we will be using d3.js. **In the <head> section of your file, please import d3 using this tag: <script src="https://d3js.org/d3.v7.min.js"></script>**

Create a zip archive containing your **HTML file and all associated data files** (such as chip_dataset.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 (e.g. server started in `~/student_hw`, with your homework at `~/student_hw/your_netid/hw4/index.html`) – be sure that it works. Check the September 12th notes file for more comments on file paths.

1. For each of the following visualizations, identify the **marks** used and **visual channels** employed from the list provided. Be as exhaustive as possible. You do not need to supply the precise data attribute – you only need to write the marks and visual channels used.

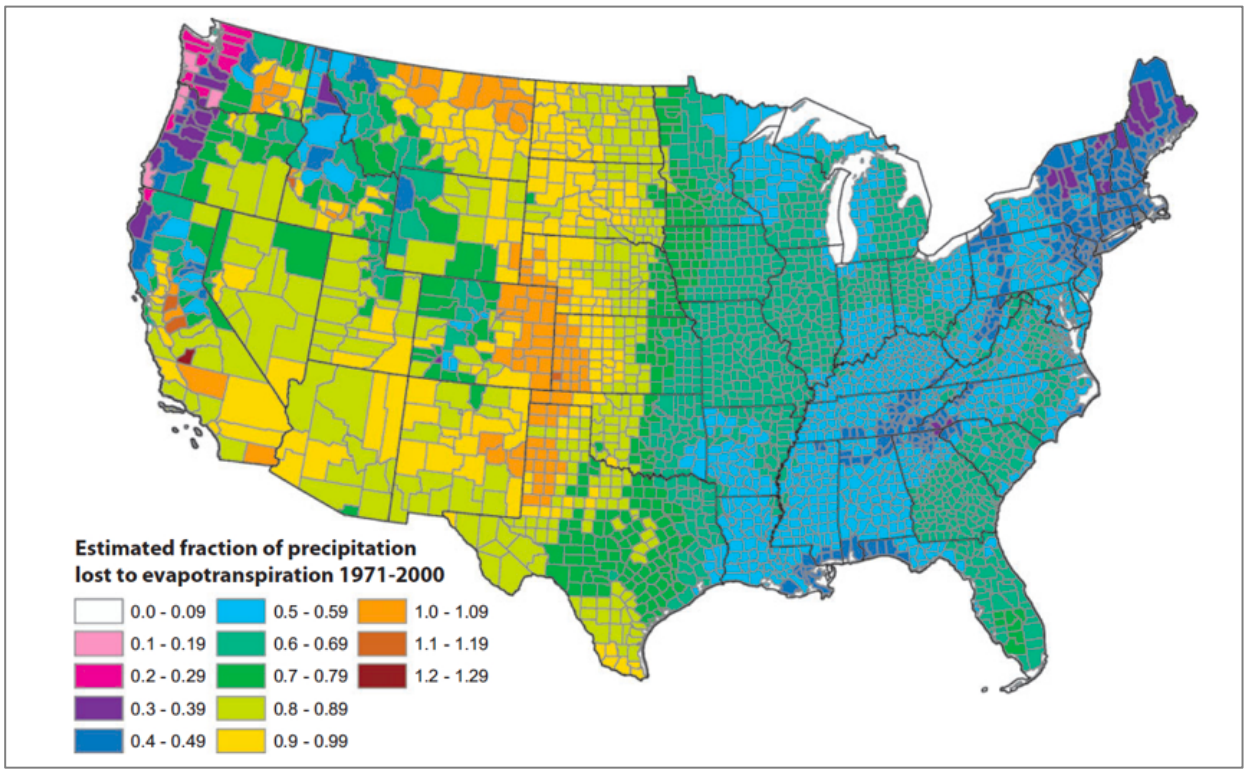
Possible visual channels:
Aligned Position (horizontal or vertical); Unaligned Position (horizontal or vertical); Aligned Length; Unaligned Length; Area; Volume; Color Hue; Color luminosity;

EXAMPLE:

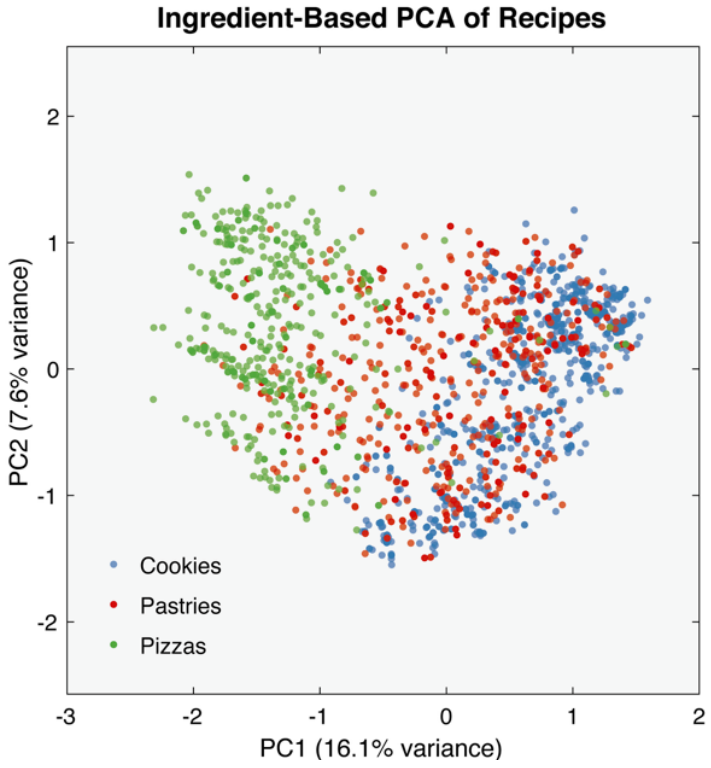


Marks: Blue rectangles
Channels: Varying the vertical aligned length and horizontal aligned position of rectangles

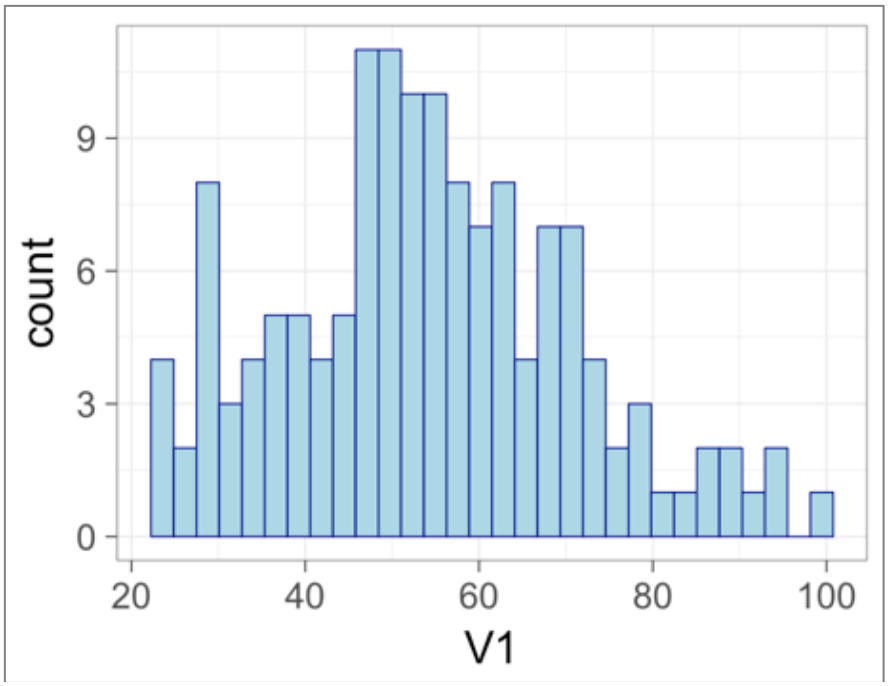
A)


Estimated fraction of precipitation
lost to evapotranspiration 1971-2000
0.0 - 0.09
0.1 - 0.19
0.2 - 0.29
0.3 - 0.39
0.4 - 0.49
0.5 - 0.59
0.6 - 0.69
0.7 - 0.79
0.8 - 0.89
0.9 - 0.99
1.0 - 1.09
1.1 - 1.19
1.2 - 1.29

B)


Ingredient-Based PCA of Recipes
PC2 (7.6% variance)
PC1 (16.1% variance)
Cookies
Pastries
Pizzas
2
1
0
-1
-2
-3
-2
-1
0
1
2

C)



2. In this problem we're going to plot some data about Moore's Law, which predicted that the number of transistors packed into computer chips would double on a yearly basis. We will make a scatterplot of graphics processing units (GPUs) over the past 20 years using a dataset gathered by Sun et al. hoping to see this trend in action.

The file `chip_dataset.json` contains a JSON block that defines an array of objects. Each object represents a particular GPU that was or is commercially available. They have several data attributes, but relevant for us will be `"Release Day"`, or the number of days the chip was released after 3/1/2000 (the earliest date in the file), and `"Transistors(million)"`, the number of transistors on the chip in millions.

A. Load the data file using an asynchronous request with `d3.json`. You can use `.then()` to handle the promise. **Make sure all of your code for this problem is within the `.then()` function**!

B. Create a 500x500px SVG element using d3 functions. Using the `const margins = {top: 10, right: 10, bottom: 50, left: 50};` template shown in class, make a `<g>` tag for your chart area that uses a translate transformation to position your points properly to account for margins. Also compute `chartWidth` and `chartHeight` using your margins so that you can use them to construct some scales.

Now we will create two linear scale functions, one for the x axis which will use `"Release Day"` and one for the y axis which will use `"Transistors(million)"`. First, get the extents of those two variables using `d3.extent()`.

Create a `d3.scaleLinear()` object called `xScale` for the `"Release Day"` variable. Set its domain to be the extent you found earlier. Set its range so that it makes use of the `chartWidth` you figured out earlier.

Create a `d3.scaleLinear()` object called `yScale` for the `"Transistors(million)"` variable. Set its domain to be the extent you found earlier. Set its range so that it makes use of the `chartHeight` you figured out earlier, and reverse the range so `chartHeight` comes first (following the trick shown in class).

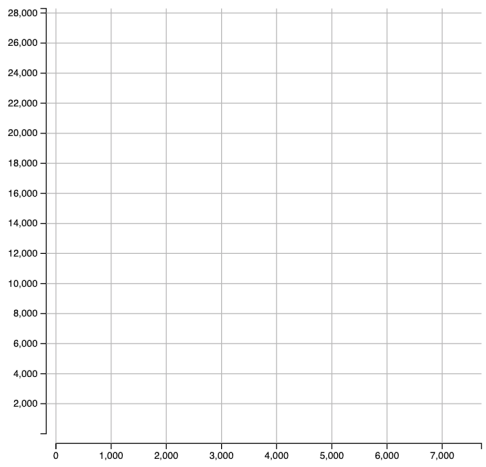
C. Now, create some axes and gridlines following the pattern demonstrated in class. Begin by making sure that you have included the necessary CSS styles in your `<head>`, as shown in the class demonstration of using d3 to draw both axes and gridlines.

Create a `d3.axisLeft()` object using your `yScale`. Append a new `<g>` tag to your SVG, and use a translate `transform` to locate it in a proper position. Use `.call()` to "paste" your `axisLeft` into the `<g>` tag.

Create a second `d3.axisLeft()` object for your gridlines also using the `yScale`. Adjust the `axisLeft` object so that is has the proper `tickFormat()` and `tickSize()`. Append a new `<g>` tag to your SVG, give it the class `"gridlines"`, and use a translate `transform` to locate it in a proper position. Use `.call()` to "paste" your `axisLeft` into this `<g>` tag.

Create a `d3.axisBottom()` object using your `xScale`. Append a new `<g>` tag to your SVG, and use a translate `transform` to locate it in a proper position. Use `.call()` to "paste" your `axisBottom` into the `<g>` tag.

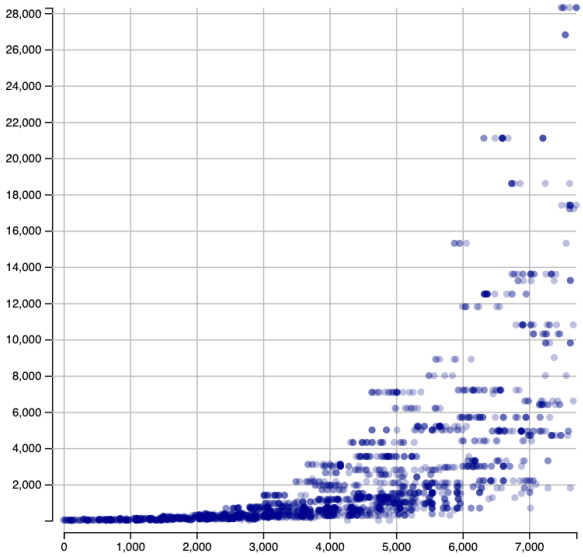
Create a second `d3.axisBottom()` object for your gridlines also using the `yScale`. Adjust the `axisBottom` object so that is has the proper `tickFormat()` and `tickSize()`. Append a new `<g>` tag to your SVG, give it the class `"gridlines"`, and use a translate `transform` to locate it in a proper position. Use `.call()` to "paste" your `axisBottom` into this `<g>` tag.



If you've done everything right so far, you should have an empty chart like the image above.

(next page)

D. Using a `forEach` loop in Javascript, create a **3px radius circle** for each point in the dataset, located at its proper place on the chart. Use your scales to place the points. Fill each point in a **dark blue** color of your choice. Set the `opacity` SVG attribute of each circle to 0.3 to make them translucent and show point density. Your result should look like the following image:



E. Something is a bit off about that image, though. The points from very recently are dramatically higher than those from the start of the dataset's time period due to Moore's Law.

To fix this issue, edit your `yScale` a bit. Change it from `d3.scaleLinear()` to `d3.scaleLog()`. You can use the same domain and range. Afterwards, your chart should look like this: