CS/INFO 3300; INFO 5100
Take-home Exam
Due 11:30am Monday, May 13

Goals: Demonstrate skills learned while taking the course.

There is no time limit for this take-home examination. You are welcome to use as a reference any class notes or slides published on the course repository, but please be careful to not directly copy course code. You are not permitted to work with others on this exam. Your work must be your own.

Unlike in previous homeworks, this take-home exam will be subdivided into multiple files.
Create an HTML file called `index.html`. In the `<head>` section of `index.html`, please import d3 and topoJSON using these tags:

```
<script src="https://d3js.org/d3.v5.min.js"></script>
<script src="https://d3js.org/topojson.v2.min.js"></script>
```

For multiple choice and short answer questions, your work will be placed directly into `index.html`. Create an `<h5>` element containing the problem number, followed by a `<p>` element(s) containing the answer(s). For example, an answer for a fictional problem might look like:

```
<h5>#10</h5>
<p>A: Hmm, I wonder. </p>
<p>B: Hmm, I also wonder. </p>

<h5>#11</h5>…
```

All programming work will be separated into individual JS files named after the problem number. For example, the code used to answer problem #5 will be contained in "`5.js`".
You will import your script into `index.html` to demonstrate that your code functions properly. For programming problems, create an `<h5>` tag containing the problem number, followed by any HTML tags necessary to complete the problem. Afterwards, import your code into `index.html` using `<script>` tags (this will also allow your .js file to use d3.js functions because `index.html` imported them in the `<head>` section). For example:

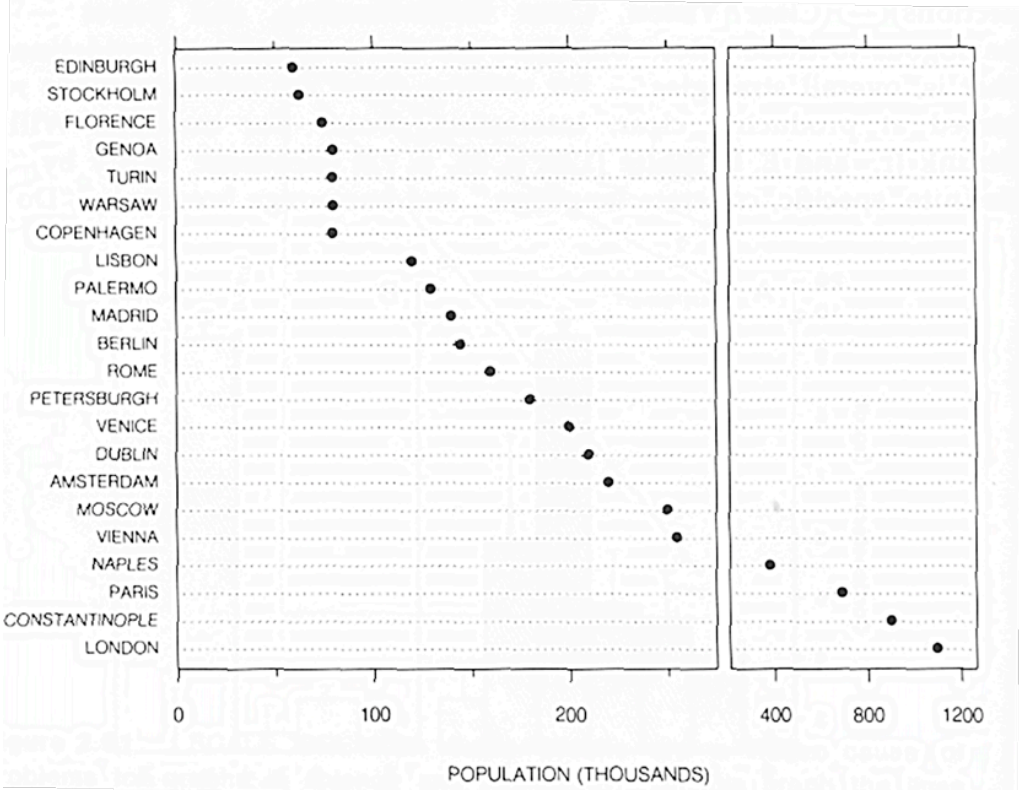
```
<h5>#14</h5>
<svg id=”scatterplot”></svg>
<script type="text/javascript" src="14.js"></script>

<h5>#15</h5>…
```

Create a zip archive containing `index.html`, your JS files, and ALL associated data files. Upload it to CMS before the deadline. Code that does not function will not receive partial credit, so please be sure to debug carefully. As all `.js` files are being imported into `index.html`, take care to avoid re-using variable names and function names. Make sure any paths you use to files such as `5.js` are relative and do not contain a"`..`" or "`/`". Your final zip submission should include the following files:

```
index.html , 5.js , 6.js, 7.js, diamond_counts.csv, europe.topojson
```

These files must be stored in the root of the zip file. No subdirectories should be included.

#1: When answering this problem in `index.html`, you are welcome to use multiple `<p>` elements, a `<ul>` element, or a `<table>` element if you so choose. Be sure to identify the specific subproblem.



1.A: Identify the marks used in this visualization.

1.B: For each of the following data attributes in the visualization, identify whether the data are nominal, ordinal, or quantitative:

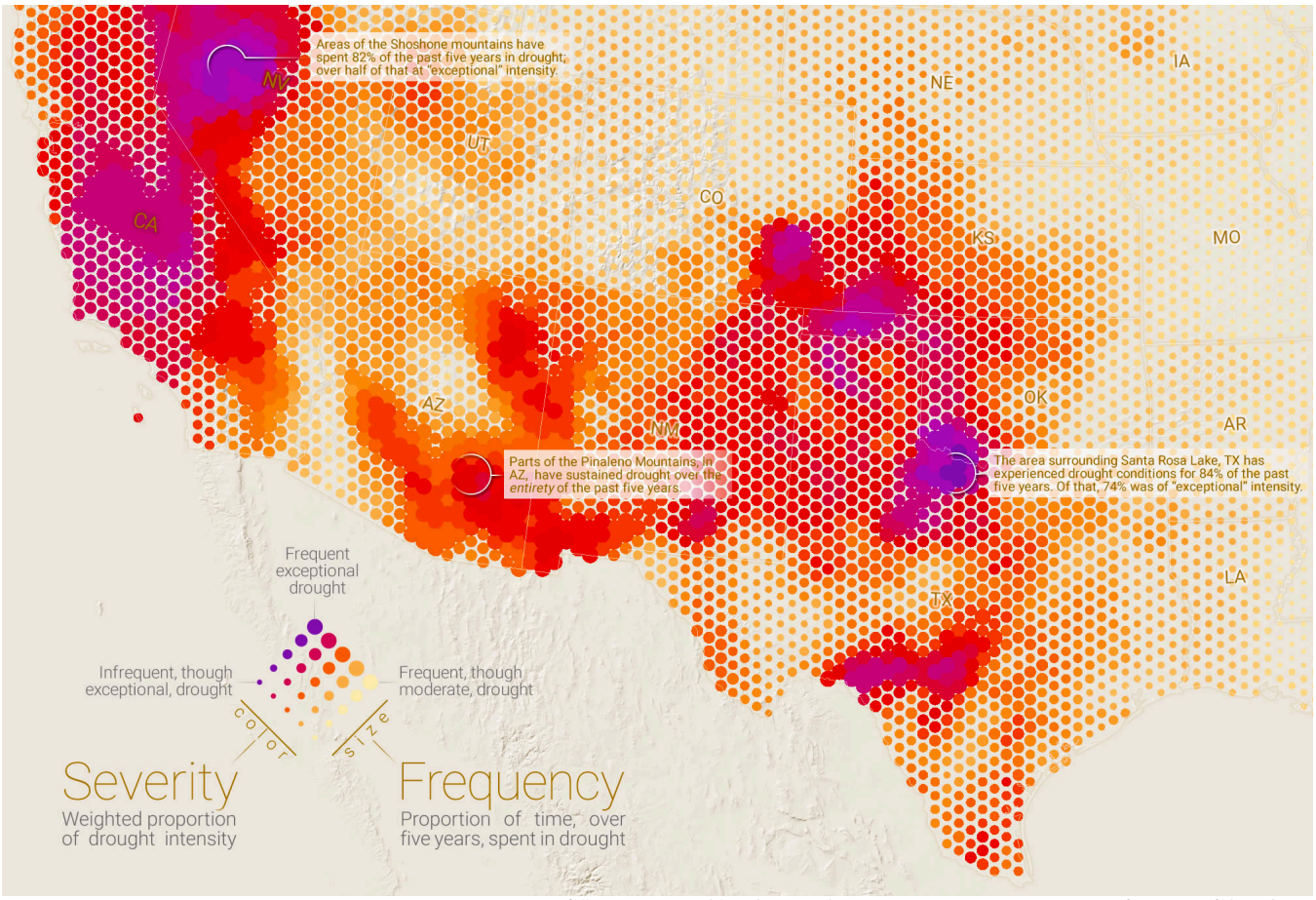
- Population (Thousands)
- City (Edinburgh, Stockholm, Florence, etc.)

1.C: For each of the following data attributes in the visualization, identify the visual channel employed (in the case of position or length, be sure to note whether it is aligned or unaligned):

- Population (Thousands)
- City (Edinburgh, Stockholm, Florence, etc.)

1.D: This visualization includes a scale break on Population (Thousands). In one sentence, state whether you think this is an effective or ineffective break and provide one reason for your choice.

#1 cont'd: When answering this problem in `index.html`, you are welcome to use multiple `<p>` elements, a `<ul>` element, or a `<table>` element if you so choose. Be sure to identify subproblems.



(full version accessible at https://adventuresinmapping.com/2016/07/12/five-years-of-drought/)

1.E: For each of the following data attributes in the visualization, identify whether the data are nominal, ordinal, or quantitative:

- Severity (weighted proportion of drought intensity)
- Frequency (proportion of time over five years spent in drought)

1.F: For each of the following data attributes in the visualization, identify the visual channel employed (if multiple visual channels are used redundantly, include all applicable visual channels):

- Severity (weighted proportion of drought intensity)
- Frequency (proportion of time over five years spent in drought)

1.G: This visualization includes a color scale representing drought severity. In one sentence, state whether you think this an effective or ineffective color scale, and provide one reason for your choice.

#2: For each of the following items, identify whether they involve a direct manipulation interaction:

a: Immediately updating the visualization as a user types a search query

b: Hovering over a circle on a scatterplot to show more details in a tooltip

c: Brushing on a histogram to filter data based on a range

d: Dragging directly on a map to move/pan it around

e: Dragging a slider tool to explore time series data

#3: Please number the following visual channels in order of accuracy, as generally agreed by the visualization community. 1 is most accurate, 5 is least:

a: Volume

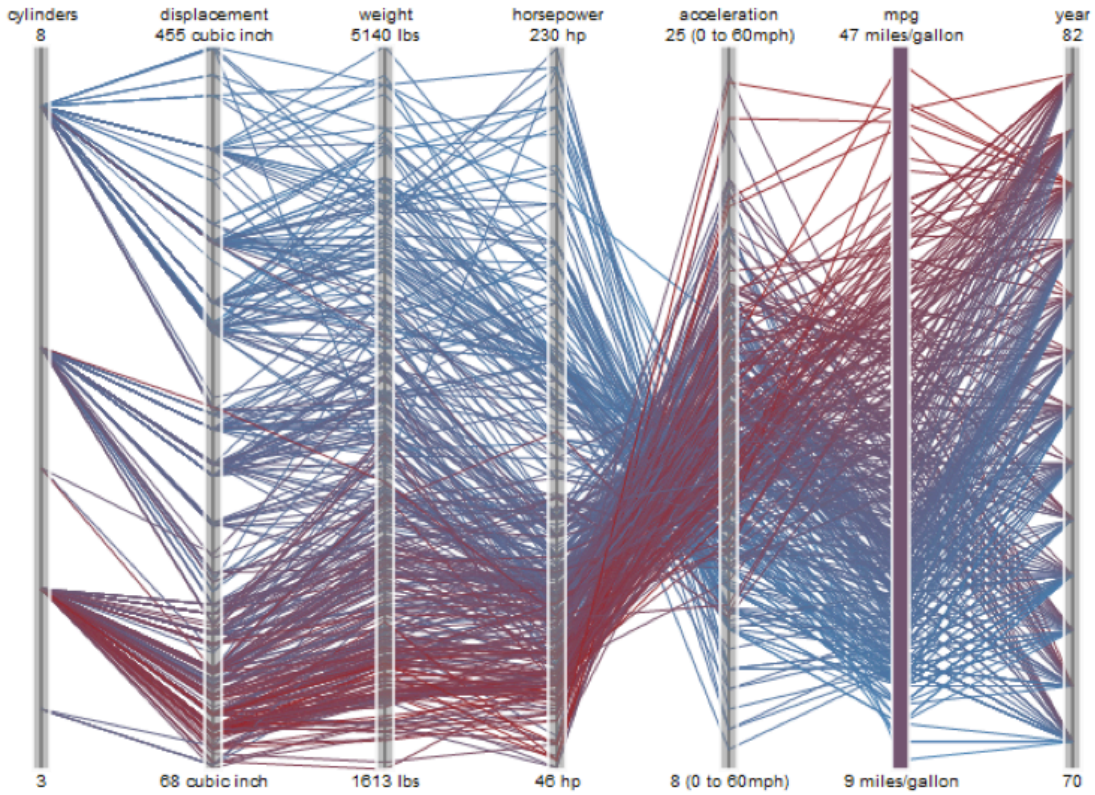
b: Angle / tilt

c: Aligned position

d: Color saturation

e: Length (aligned or unaligned)

#4: Please answer subproblems using the following example visualization:



4.A: What is this type of visualization called?

4.B: What do the horizontal red and blue lines on the visualization represent?

4.C: In one sentence, describe 1 advantage of this visual metaphor for multivariate data. In another sentence, describe 1 potential drawback of this visual metaphor.

#5. For the following problem, please create a file, `5.js`, containing your code. In your `index.html` file, create an `<svg>` element that is 500px in width and 120px in height with the ID set to `"area"`. Import `5.js` in your HTML file so it runs. This problem makes use of `diamond_counts.csv`, so make sure to include it within your ZIP file so your code works properly.

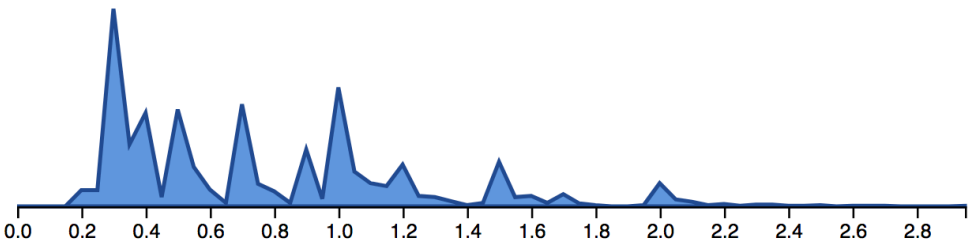
In this problem you will be making an area histogram chart of the diamond price data we used earlier this term. I have already binned the data into a CSV file, `diamond_counts.csv`. Data points in the file contain two attributes, `"carat"` for the starting value of a bin and a numeric `"count"`.

For the purposes of this problem, we are not worrying about interpolation or properly representing bin width. We will set the value of the area chart to `"carat"` and `"count"` and do nothing fancy.

Within `5.js`, please create Javascript code that:

- Uses `d3.csv` and a promise or await call to load `"diamond_counts.csv"` (no `..` or `/`, please)
- Properly converts the strings in `diamond_counts.csv` into numbers so they can be used, handling any issues with data cleanliness gracefully
- Uses d3 to select your `#area` SVG canvas
- Programmatically creates two `<g>` elements:
    - One with the class `"chart"`
    - Another with the class `"axis"` that is translated downwards 100px using `"transform"`
- Creates two scales for working with the data. While range can be hard-coded, domain ought to be set to the proper extent of the data. The two scales should be:
    - A linear scale for `"carat"` values that has a range of `[10,490]`
    - A linear scale for `"count"` values that has a range of `[100, 0]`
- Populates the `".axis"` `<g>` element with `d3.axisBottom()` labels
- Constructs a `d3.area()` path generator, with:
    - `.x()` set to properly use the `"carat"` data and scale
    - `.y0(100)` so the bottom of the area is stuck to the axis at the bottom
    - `.y1()` set to properly use the `"count"` data and scale
- Add a `<path>` object to the `".chart"` `<g>` element using the path generator you created
    - The `<path>` element should be filled with a blue color and stroked with a 2px dark blue

Example output:



#6. For the following problem, please create a file, `6.js`, containing your code. In your `index.html` file, create a `<canvas>` element that is 500px in width and 40px in height with the ID set to `"colorscale"`. Import `6.js` in your HTML file so it runs.

In this problem you will create your own color scale and then present it as a smooth gradient using a `<canvas>` element and manual, pixel-by-pixel drawing. Do not use `.fillRect()`.

Within `6.js`, write Javascript code that:

- Uses `d3.scaleLinear()` to create a color scale. Your color scale should have these properties:
    - Varies color hue while also varying either saturation or luminosity (for redundancy)
    - Is friendly to individuals who are red-green color blind
    - Scale domain should be set to `[0, 500]`
    - Scale range contains at least two color strings based on your scale design
    - Makes use of `d3.interpolateHcl` for perceptually uniform color interpolation
    - Does not closely resemble any pre-made `d3.scaleSequential` color themes (e.g. Viridis)
- Uses plain Javascript calls to select your `#colorscale <canvas>`
- Fetches an image object from the canvas using `getImageData()` and edits its pixels
- Uses two `for` loops to go through the entire canvas pixel-by-pixel along x and y
- Sets the color of a pixel by applying the scale to the x value of the pixel (and ignoring y)
    (hint: cast your color to `d3.rgb()` to access .r, .g, .b)
- "Paints" the modified image object back onto the canvas

Example (only for canvas drawing – no claims here as to whether this is a good scale or not):

#7. For the following problem, please create a file, `7.js`, containing your code. In your `index.html` file, create an `<svg>` element that is 500x500px with the ID set to `"map"`. After your `<svg>`, copy this element into your code: `<p>PISA Score: <span id="hint"></span></p>` to use as a placeholder for hover-for-details interactions. Finally, import `7.js` in your HTML file so it runs.

In this problem you will be making a choropleth map of the PISA academic test data we used earlier this term for regression analyses. I have already inserted PISA test data into a topoJSON dataset, `europe.topojson`. Make sure to include it within your ZIP file so your code works properly.

Within `7.js`, write Javascript code that:

- Uses `d3.json` and a promise or await call to load `"europe.topojson"` (no `..` or `/` in path, and be sure the names you choose do not clash with the file load in #6)
- Uses d3 to select your `#map <svg>` canvas
- Calls `topojson.feature()` on `<dataset>.objects.europe` to extract a featureCollection
- Creates a Mercator projection fit to the size of the SVG canvas, and makes a path generator
- Builds a `d3.scaleSequential` using the `d3.interpolateViridis` built-in color scale
- Sets the domain of the scale to the proper extent of PISA values in the dataset
  (hint: you can find the PISA score for a country `"d"` at `"d.properties.pisa"`)
- Adds `<path>` elements to the SVG using a data call on the feature collection of countries
- Does not use a `forEach` or `for` loop – only data joins
- Fills in the country `<path>`s based on their PISA scores using the color scale and applies no stroke
- Uses `.on("mouseover")` to apply a black stroke to a country when the user hovers over it and adjusts the contents of the `#hint <span>` element to show the PISA score of the country
- Uses `.on("mouseout")` to gracefully reset the stroke and contents of the `<span>` element

Example to the right: