INFO/CS 3300
Take-home Exam
Due before 12:00pm (noon) on December 14th

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 videos published on the repository or Canvas, but 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**.

In this zip file we have provided you with .html and .js files to use when completing this examination. **Please put all of your written answers in index.html and update the .js files with your own code as directed by the individual problems**.

For multiple choice and short answer questions, your work will be **placed directly into `index.html`**. Please put your answer in the area marked underneath the **<h5>** element for the problem number. For example, an answer for a fictional problem might look like:

```
<div class="answer">
  <h5>#12</h5>
  <p>A: I'm Spartacus! </p>
  <p>B: No, I'm Spartacus! </p>
</div>
```

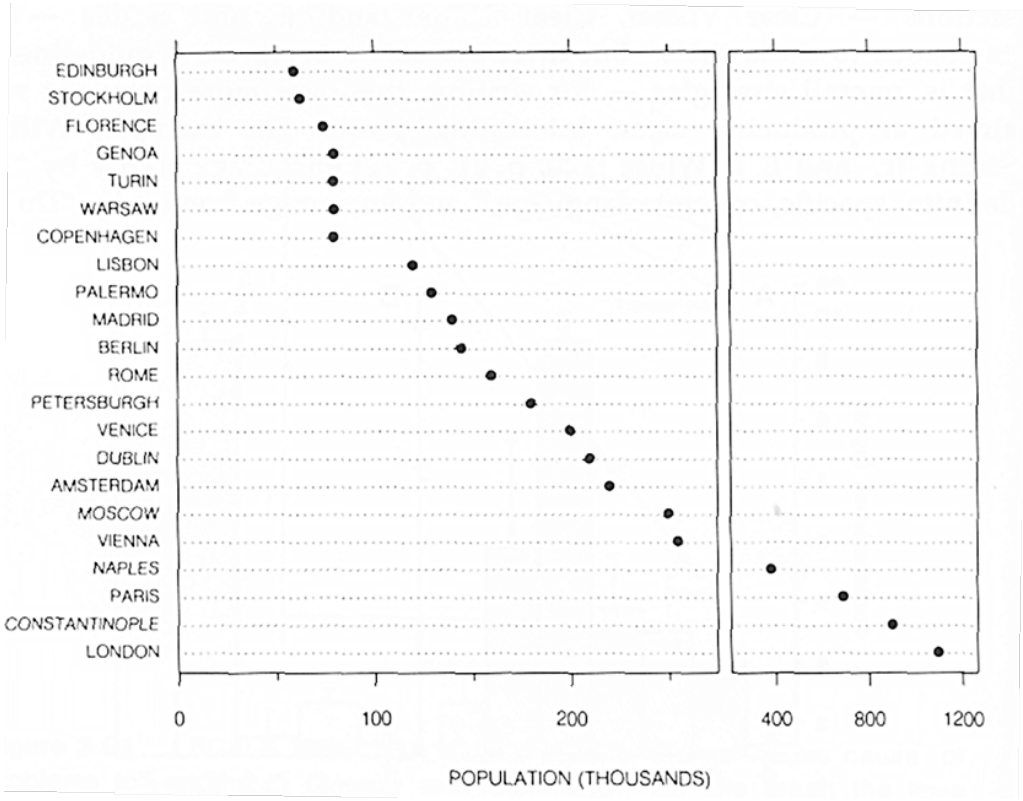
All programming work will be separated into individual JS files named after the problem number. For example, the code used to answer problem #7 is contained in "`7.js`". Please be aware that some browsers have trouble refreshing code contained in .js files. If you notice this occurring, try a hard refresh by holding the shift key and pressing the refresh button in the browser.

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 may not receive 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 data files are relative and do not contain a"`..`" or "`/`". Your final zip submission should include the following files:

```
index.html, 7.js, 8.js, 9.js, 10.js, bakeoff_scores.csv, class_network.json,
 olympic_ages.json, us_artists.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**:

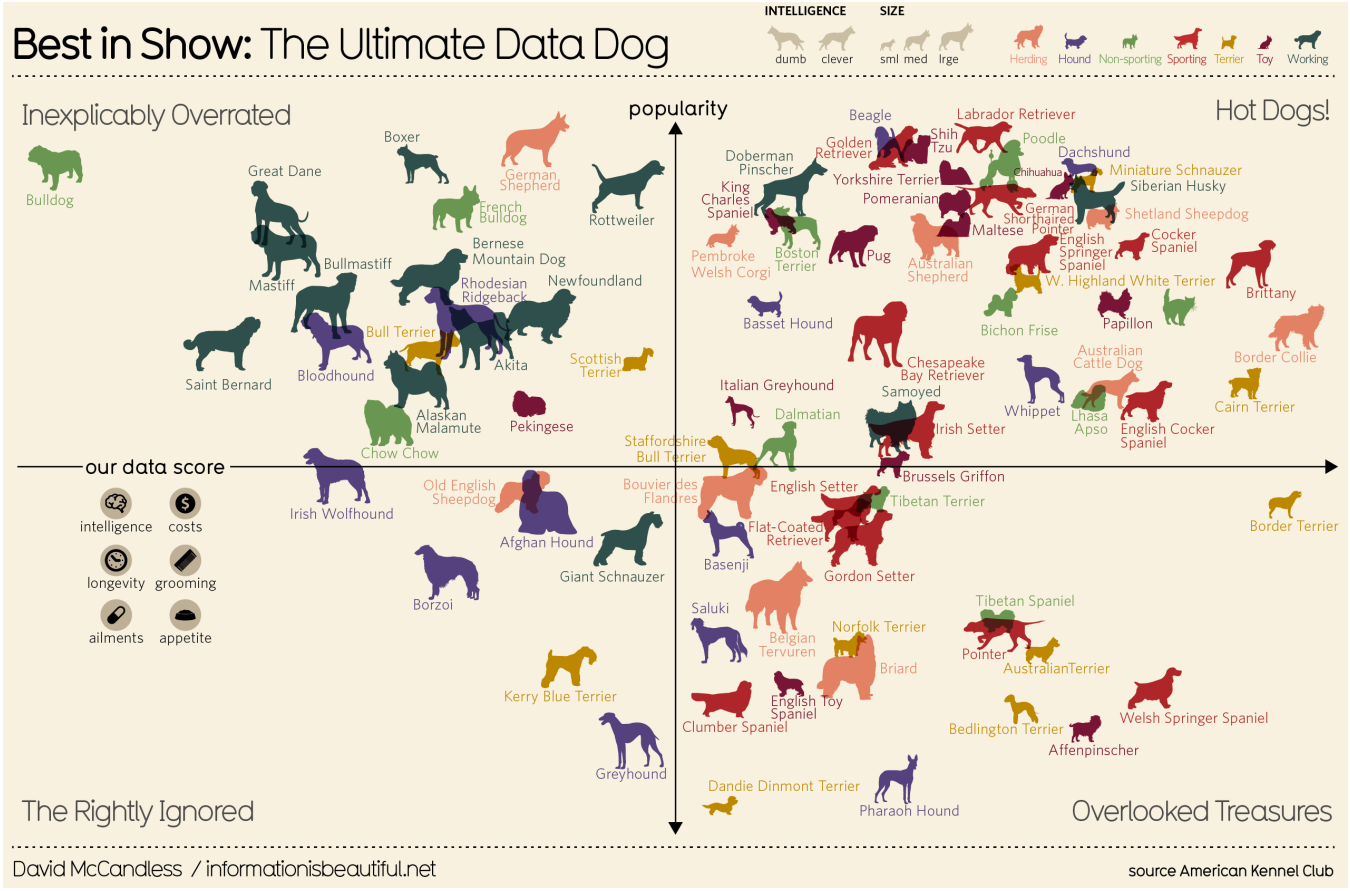
- Population (Thousands)
- City (Edinburgh, Stockholm, Florence, etc.)

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.

#2: 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.**



2.A: Identify the **marks** used in this visualization.

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

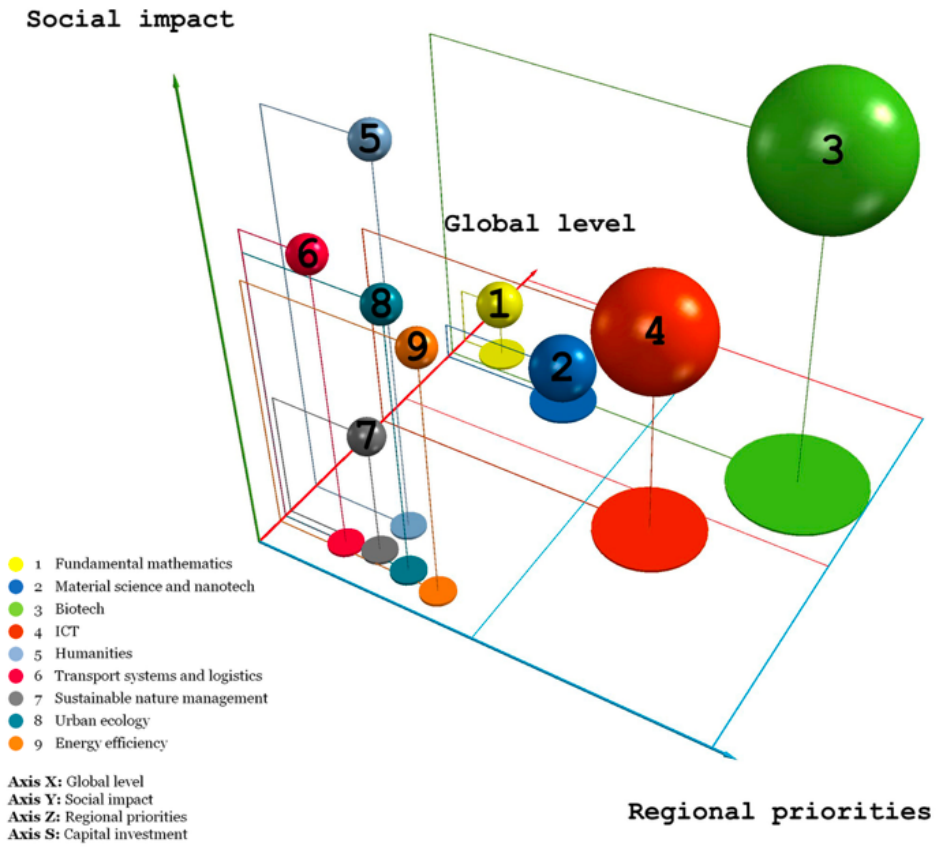
- Popularity (y-axis)
- AKC Dog group (upper right corner; Herding, Hound, Non-sporting, etc.)

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

- Popularity (y-axis)
- "Our Data Score" (x-axis)
- AKC Dog group (upper right corner; Herding, Hound, Non-sporting, etc.)

2.D: Unlike a traditional scatterplot that may use circles to indicate breeds of dogs, this visualization includes a variety of outlines of dog breeds. In one sentence, describe whether you think that this design choice was **effective or ineffective** and provide one **reason** for your decision.

#3: 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.**



3.A: For each of the following data attributes in the visualization, identify whether the data are **nominal, ordinal, or quantitative**:

- Domain area (Biotech, ICT, Humanities, etc.)
- Regional priorities (score for prioritization of a domain, as reflected by left/right location)
- Capital Investment (dollars invested in a domain, as reflected by bubble radius)

3.B: 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):

- Domain area (Biotech, ICT, Humanities, etc.)
- Regional priorities (score for prioritization of a domain, as reflected by left/right location)
- Capital Investment (dollars invested in a domain, as reflected by bubble radius)

3.C: There are many reasons why one could argue that this is an ineffective visualization. However, let's be positive! In one to two sentences identify **one positive design aspect of this visualization** and **explain** your reasoning.

#4: For each of the following statements, **identify whether they are `True` or `False`** according to course materials. You are welcome to use multiple **<p>** elements, a **<ul>** element, or a **<table>** element if you so choose. Be sure to identify subproblems

a. An SVG canvas uses raster data (i.e., pixels) rather than DOM elements.

b. In a dynamic querying interface, users might type their answers and then press enter to see results.

c. Force-directed layouts for networks create the same final layout regardless of starting state.

d. On average, humans can track between 4 and 6 moving targets on a screen as long as they don't change much while moving.

e. In a chord diagram, the order of nodes does not affect the way that edges are presented and understood by users.

f. Pre-attentive processing is incredibly rapid, identifying low-level, simple features in visual stimuli.

#5: Please **number the following visual channels in order of accuracy**, as generally agreed by the visualization community.

a. Volume

b. Aligned length

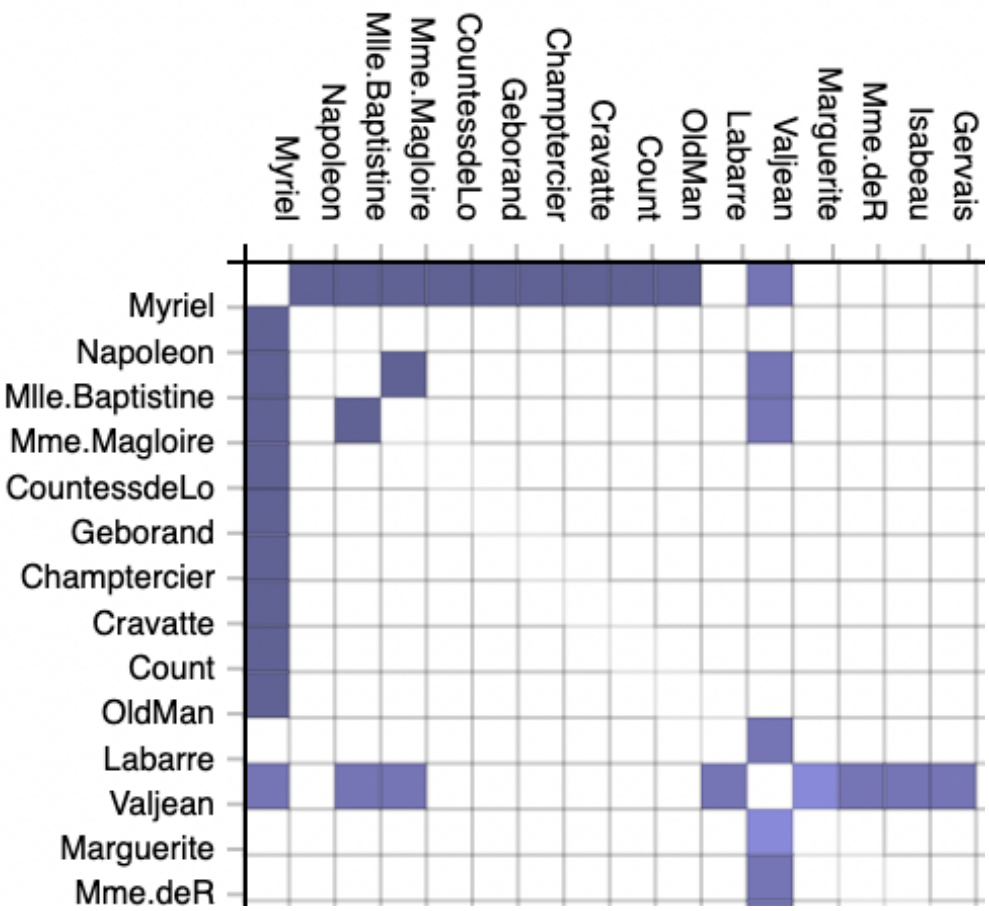
c. Angle / tilt

d. Unaligned position

e. Color saturation

Enter your answer using an **<ol>** element to **number the channels from 1, corresponding to the most accurate channel, to 5, corresponding to the least accurate channel**.

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



(A network visualization showing character co-occurrence in the book Les Misérables.
Filled cells indicate a connection between two characters denoted by rows and columns)

6.A: What is this **type of visualization** called?

6.B: What visual **marks** correspond to the a) **nodes** and the b) **edges** in the original network data?

6.C: In one sentence, describe **1 advantage** of this visual metaphor for network data. In another sentence, describe **1 potential drawback** of this visual metaphor.

#7. In the zip file for this assignment we have included a data file, `olympic_ages.json`, which you will visualize for this problem. It contains data on medal winners at both the Summer and Winter Olympics. We have selected four sports for which to **compare the ages of athletes over time in a scatterplot** (please note that dates for Winter Olympics prior to 1992 have been adjusted to make the chart clearer). **We have already included a script file**, `7.js`, **where you will place your Javascript code for solving this problem**. Make sure to include the data file within your ZIP file.

Please add code to `7.js` **which:**

- Uses `d3.json` and a **promise or await call** to load `"olympic_ages.json"` (no `..` or `/` in path, and be sure the names you choose do not clash with the file loads elsewhere)

- Pre-processes the `date` attribute in the dataset into **JS time objects** using `d3.timeParse()` (hint: the attribute just contains the year [`%Y`] of the competition)

- Uses d3 to **select the `#scatter` SVG canvas** (which is 700px wide and 400px tall). Please **reserve 40 pixels at the bottom and left sides as a margin** for axis labels and **10 pixels at the top and right sides for margin** (use whatever method you prefer to reserve these spaces)

- Programmatically **creates five <g> elements**:
  - Two with the class `"axis"` for the X and Y axes, translated properly based on your margins
  - Two with the class `"gridlines"` for the X and Y gridlines, translated properly based on margins
  - One with the class `"chart"` which will contain your scatterplot points

- Creates **three scales**. While **range can be hard-coded (including margins)**, **domain should be set to the data extent** using a call to `d3.extent()` as necessary. The three scales should be:
  - A `scaleTime()` for the `"date"` key **X-axis**, with a proper range for the chart area width
  - A `scaleLinear()` for the `"age"` key **Y-axis**, with a proper range for the chart area height
  - A `scaleOrdinal()` for the `"sport"` colors, using `d3.schemeCategory10` with no domain/range

- **Populates the `".axis"` <g> elements** with `d3.axisLeft()` and `d3.axisBottom()` **labels**.

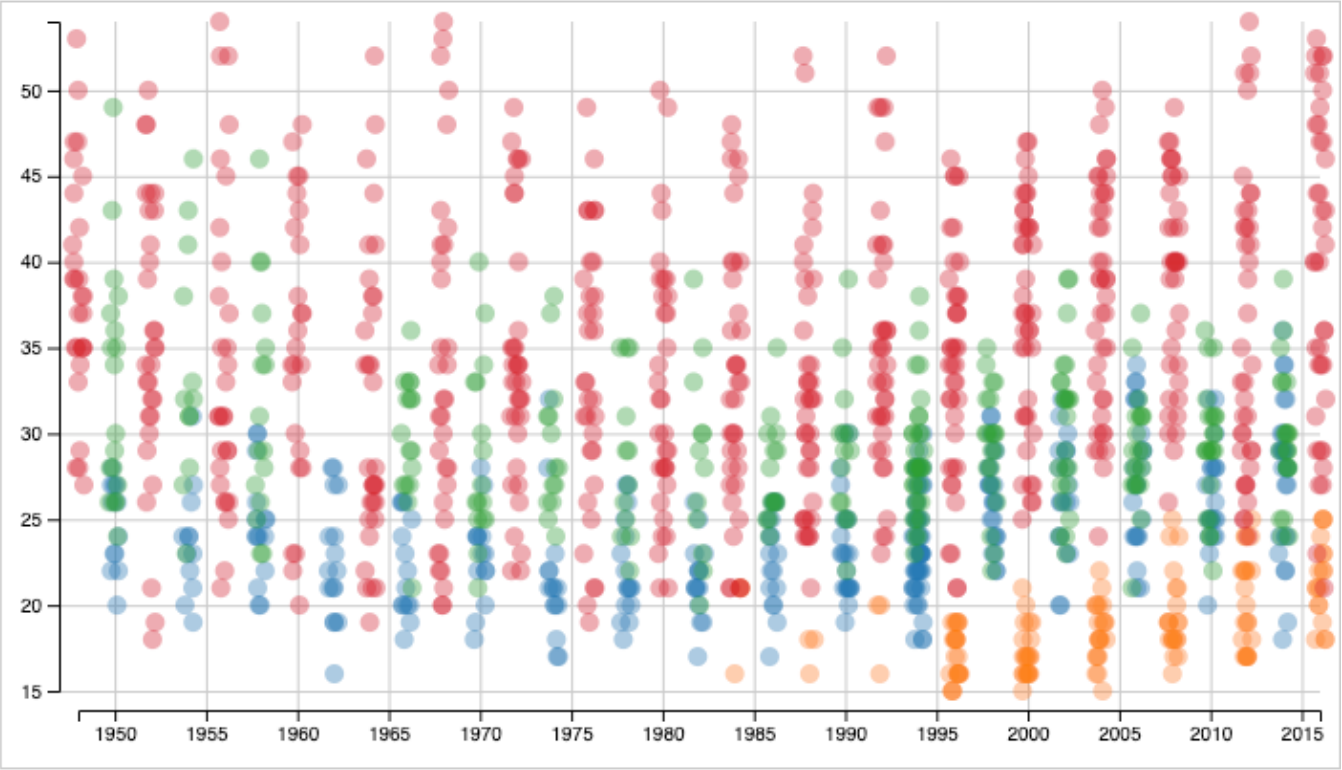
- **Populates the `".gridlines"` <g> elements** with `d3.axisLeft()` and `d3.axisBottom()` **gridlines**. (hint: we have already added the necessary CSS at the top of `index.html`)

- Uses a **d3 data join to append new `<circle>` elements** to the `chart` <g> tag.
  - **Position** the circles using your scales, and **set their `fill`** using your ordinal scale.
  - Please **include jitter on your X axis** by **adding a random number between -3 and 3** pixels after you have run your scale (i.e. `xScale(jsDate) + randomNumber`). (hint: d3.randomUniform)
  - Circles should have a **radius of 5**, and an **opacity of 0.4**.

(next page)

Example output for #7:

(note that yours may not look identical due to the random jitter on the x-axis)



#8: In the zip file for this assignment we have included a data file, `bakeoff_scores.json`, which you will visualize for this problem. It contains data on competitors in the TV reality series, *The Great British Baking Show*. We have computed 'scores' ranging from 0 to 100 for each performer in the show and then averaged them by age group. This chart will explore the question of **whether the show penalizes older performers**, and whether **older performers might do better on the "technical" challenges** which evaluate their baking knowledge.

You will create a chart similar to a *lollipop chart* in order to visualize these data. **We have already included a script file**, `8.js`, **where you will place your Javascript code for solving this problem**. Make sure to include the data file within your ZIP file.

Please add code to `8.js` **which:**

- Uses `d3.json` and a **promise or await call** to load **`"bakeoff_scores.json"`** (no `..` or `/` in path, and be sure the names you choose do not clash with the file loads elsewhere)

- Uses d3 to **select the `#bakeoff` SVG canvas** (which is 700px wide and 400px tall). Please **reserve 40 pixels at the bottom and left sides as a margin** for axis labels and **10 pixels at the top and right sides for margin** (use whatever method you prefer to reserve these spaces)

- Programmatically **creates five `<g>` elements**:
  - Two with the class **`"axis"`** for the X and Y axes, translated properly based on your margins
  - Two with the class **`"gridlines"`** for the X and Y gridlines, translated properly based on margins
  - One with the class **`"chart"`** which will contain your data marks

- Creates **two scales**. **Range can be hard-coded (including margins):**
  - A `scaleLinear()` for the age **X-axis**, using `[17, 71]` as its domain and a proper range array for the chart area width
  - A `scaleLinear()` for the score **Y-axis**, using `[0, 100]` as its domain and a proper range array for the chart area height

- **Populates the `".axis"` `<g>` elements** with `d3.axisLeft()` and `d3.axisBottom()` **labels** and the **the `".gridlines"` `<g>` elements** with `d3.axisLeft()` and `d3.axisBottom()` **gridlines**.
  (hint: we have already added the necessary CSS at the top of `index.html`)

- Uses a **d3 data join to append new `<g>` elements** inside of the `chart` `<g>` tag, one for each data point so that each tag will contain one rectangular "lollipop".
  (hint: assign the result of your join to a variable to make the rest of the problem easier)

- Uses `.append()` to add a **horizontal blue `<line>`** element **inside each of the `<g>` tags** you created.
  - **Position** the line as follows:
    - `x1` is the result of calling your age scale on `d.start_age` (i.e. `d => ageScale(d.start_age)`
    - `x2` is the result of calling your age scale on `d.end_age`
    - `y1` and `y2` are the result of calling your score scale on `d.overall_score`
  - Give the line a **blue stroke color** and a **stroke width of 10**.

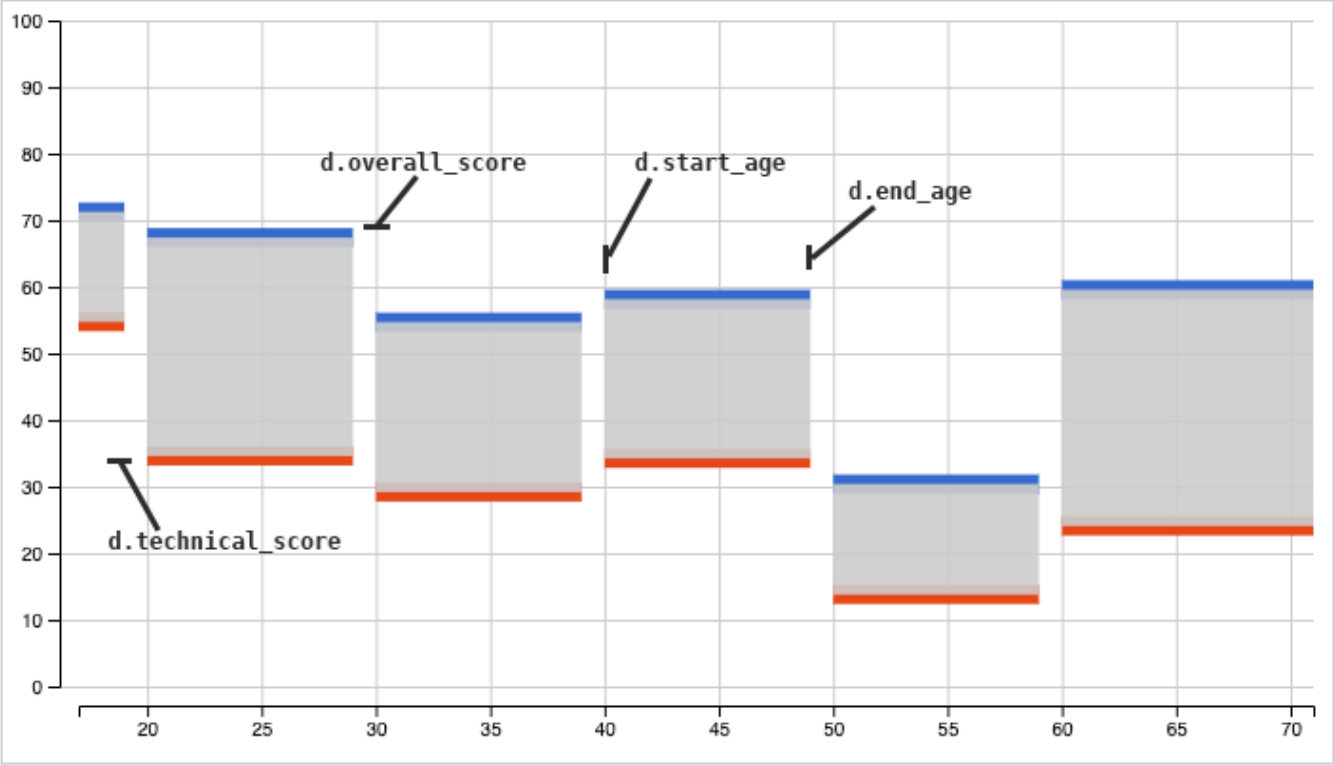
- Uses `.append()` to add a **horizontal red** `<line>` element **inside each of the** `<g>` **tags** you created.
    - **Position** the line as follows:
        - `x1` is the result of calling your age scale on `d.start_age`
        - `x2` is the result of calling your age scale on `d.end_age`
        - `y1` and `y2` are the result of calling your score scale on `d.technique_score`
    - Give the line a **red stroke color** and a **stroke width of 10**.

- Uses `.append()` to add a **vertical grey** `<line>` element **inside each of the** `<g>` **tags** you created.
    - **Position** the line as follows:
        - `x1` and `x2` are the result of calling your age scale on `(d.start_age+d.end_age)/2`
            (i.e. `d => ageScale( (d.start_age+d.end_age)/2)`)
        - `y1` is the result of calling your score scale on `d.overall_score`
        - `y2` is the result of calling your score scale on `d.technique_score`
    - Give the line a **grey stroke color**,
      an **opacity of 0.9**,
      and a **stroke width of** `d => ageScale(d.end_age) - ageScale(d.start_age)`.

(next page)

Example output for #8:

(note that some text annotations have been added to help you see where the values are located, these do not need to be included in your submission)



HTML hint for #8:

(your HTML should *resemble this* in structure if you follow the directions)

```
▼<svg width="700" height="400" id="bakeoff" style="border:1px solid #ccc">
  ▶<g class="y gridlines" transform="translate(30,10)" fill="none" font-size="10" font-family="sans-serif" text-anchor="end">…
   </g>
  ▶<g class="y axis" transform="translate(30,10)" fill="none" font-size="10" font-family="sans-serif" text-anchor="end">…</g>
  ▶<g class="x gridlines" transform="translate(40,370)" fill="none" font-size="10" font-family="sans-serif" text-anchor="middl
   e">…</g>
  ▶<g class="x axis" transform="translate(40,370)" fill="none" font-size="10" font-family="sans-serif" text-anchor="middle">…
   </g>
  ▼<g class="chart" transform="translate(40,10)">
    ▼<g class="lollipop">
       <line class="technique" r="4" x1="0" x2="24" y1="158" y2="158" stroke="#e64715" stroke-width="10"></line>
       <line class="overall" r="4" x1="0" x2="24" y1="100" y2="100" stroke="#376acf" stroke-width="10"></line>
       <line class="stick" r="4" x1="12" x2="12" y1="158" y2="100" stroke="#ccc" stroke-width="24" opacity="0.9"></line>
     </g>
    ▶<g class="lollipop">…</g>
    ▶<g class="lollipop">…</g>
    ▶<g class="lollipop">…</g>
    ▶<g class="lollipop">…</g>
    ▶<g class="lollipop">…</g>
   </g>
 </svg>
```

#9. In the zip file for this assignment, we have included a data file, `class_network.json`, which you will visualize for this problem. It contains an attempt at showing the course dependencies in INFO and CS courses at Cornell. You will use a **force-directed network diagram** to visualize these relationships. **We have already included an empty script file**, `9.js`, **where you will place your Javascript code for solving this problem**. We have also included some extra **<path>** elements in the SVG to help make your chart more appealing. Make sure to include the data file within your ZIP file so your code works properly.

Please add code to `9.js` **which:**

- Uses `d3.json` and a **promise or await call** to load `"class_network.json"` (no `..` or `/` please)
  (You will find the nodes and links as keys in this dataset)

- Creates a `d3 forceSimulation` **associated with the nodes** and containing the following **forces**:
  - A **forceLink** to connect nodes together. Note that **you will need to use** `.id()`, as the links in the dataset make use of the `course` **key in the nodes** (hint: `d => d.course`)
  - A **forceManyBody** to handle repulsion between nodes. Set its **strength to -120**.
  - A **forceCollide** with a **collision radius of 6**
  - A **forceRadial** to create rings of classes with different levels (i.e. 1-level, 2-level, etc.). As we haven't seen this in class, you are free to paste in this force directly:

```
d3.forceRadial().radius( d => 60 * d.level ).x(300).y(300).strength(3)
```

- Uses d3 to **select the** `#network` **SVG canvas**

- Creates a `d3.scaleOrdinal` with the `d3.schemeCategory10` color scheme

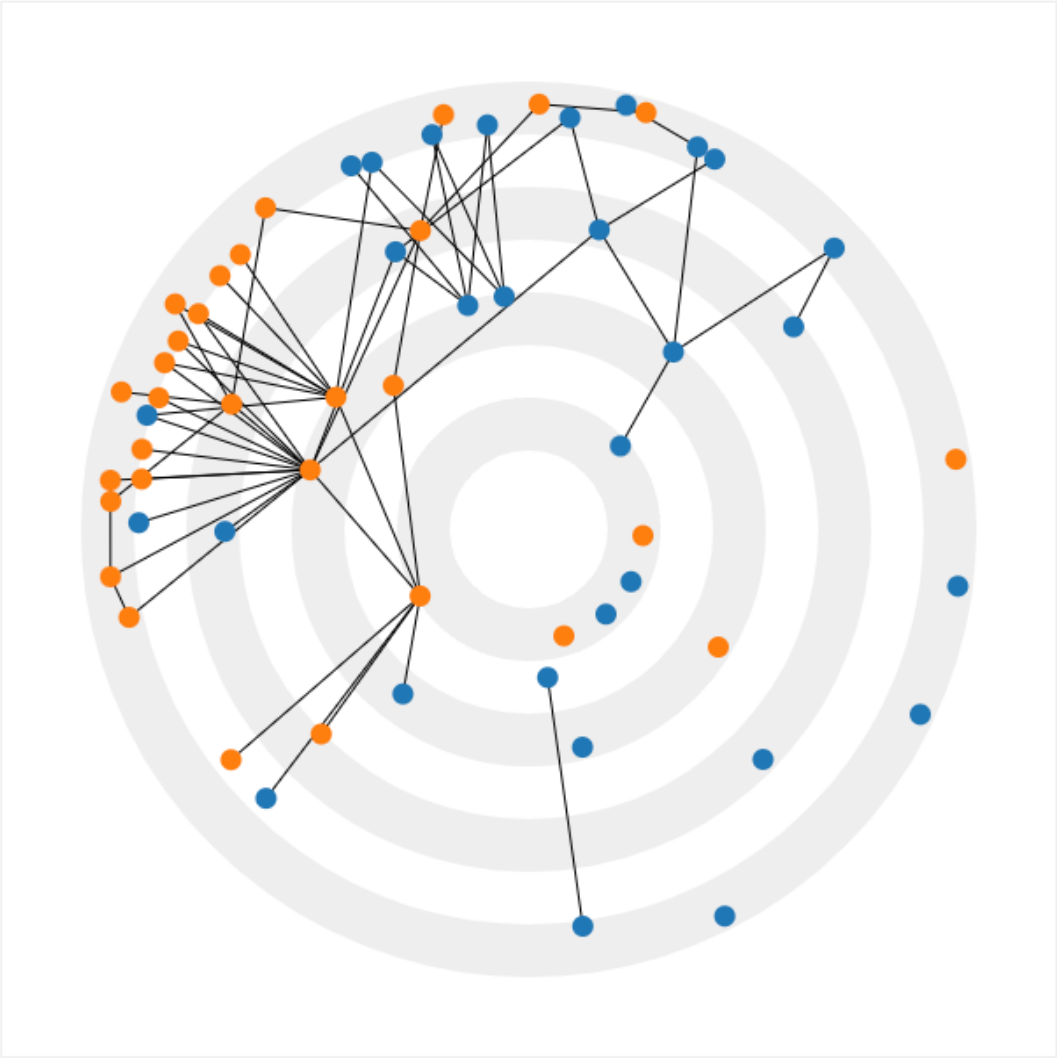
- Assigns a **function called** `tick` **to the** `.on("tick")` **event** of the simulation

- Uses **data joins to draw the network diagram** inside of the `tick` function:
  - The first data join should create **<line>** elements from source to target for each link in black.
  - The second data join should create **<circle>** elements for each node with **radius 6** and with a **fill color set using your scaleOrdinal and the** `d.prefix` key provided for each node.

(next page)

Example output for #9:

(note that node positions may not be identical to the example due to the force simulation)



#10. In the zip file for this assignment we have included a data file, `us_artists.topojson`, which you will visualize for this problem. **We have already included an empty script, `10.js`, where you will place your Javascript code for solving this problem**. In the HTML template we have included a 900x600 `<svg>` element where the map will go.

In this problem you will be making a **choropleth map of the population of artists in different US states according to census bureau data**. I have already inserted the values into the geographic dataset. You will find what you need included as **a property called `percent_artists` for each of the Features in the topoJSON FeatureCollection**. Make sure to include the dataset within your ZIP file.

Within `10.js`, please write Javascript code that:

- Uses `d3.json` and a **promise or await call to load `"us_artists.topojson"`** (no `..` or `/` in path, and be sure the names you choose do not clash with the file load in #9)

- Uses d3 to **select your `#map` `<svg>` canvas**

- Calls `topojson.feature()` on `<dataset>.objects.states` to extract a featureCollection

- Creates a **geoAlbersUsa projection** fit to the size of the SVG canvas, and a **geopath generator**

- Builds a `d3.scaleSequential` using the `d3.interpolateViridis` built-in **color scale**

- Sets the **domain of the scale** to the proper extent of `percent_artist` values in the dataset
  (hint 1: `d3.extent` works on the `.features` list of a topojson feature)
  (hint 2: you can find the value for state feature `"d"` at `"d.properties.percent_artists"`)

- Adds `<path>` elements to the SVG using a **data join on the feature collection** of states and the geopath generator created earlier

- Fills in the `<path>`s based on their `percent_artists` values using the **color scale** with **no stroke**

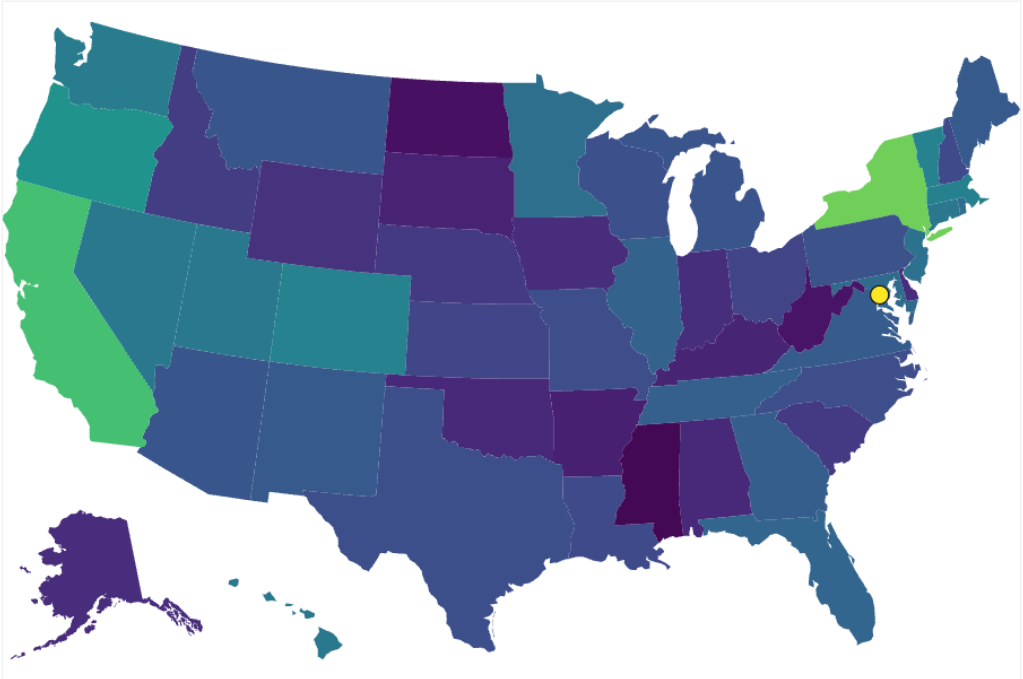
- Builds a `d3.scaleSequential` using the `d3.interpolateViridis` built-in **color scale**

- Adds a **circle where the District of Columbia (Washington D.C.) is located** showing its high value.
  - Use your **geopath generator to find the pixel location** of the city.
    It is located at **longitude** `-77.025955` and **latitude** `38.942142`
  - Make an **circle that is 8 pixels in radius**, set the **fill color** by running the value `0.224` through your color scale, and give it a **1px black stroke** outline

(You do not need to add a mesh for this visualization)

(next page)

Example output for #10:



(END OF EXAM)