INFO 3300
Take-home Exam
Due before 11:59pm on December 13th

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: Hmm, I wonder. </p>
  <p>B: Hmm, I also wonder. </p>
</div>
```

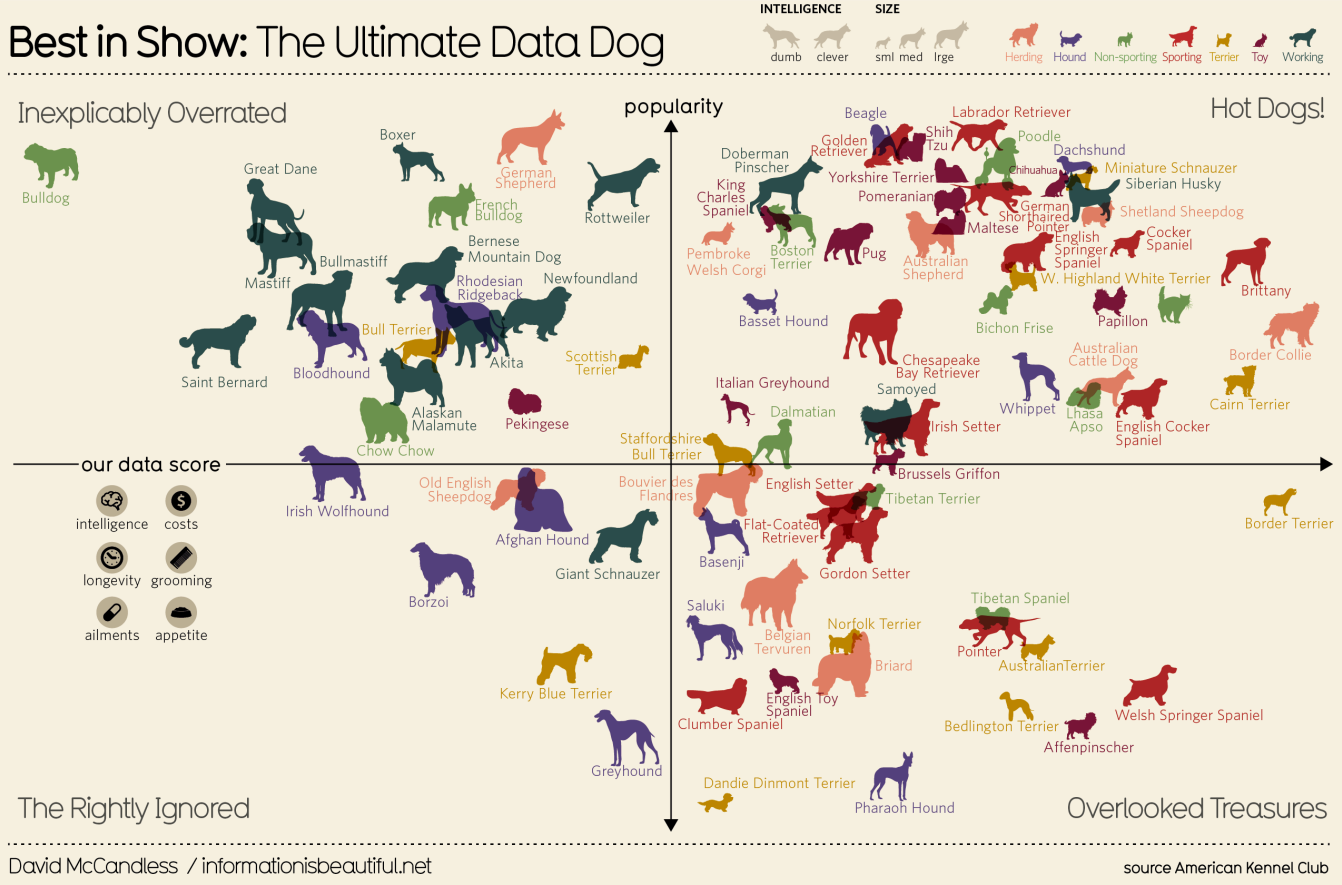
All programming work will be separated into individual JS files named after the problem number. For example, the code used to answer problem #7 is contained in "`7.js`". Some of the .js files already have code included in them. Please do not modify the code unless directed.

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, penguin_flippers.csv,
 olympic_ages.json, 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**:

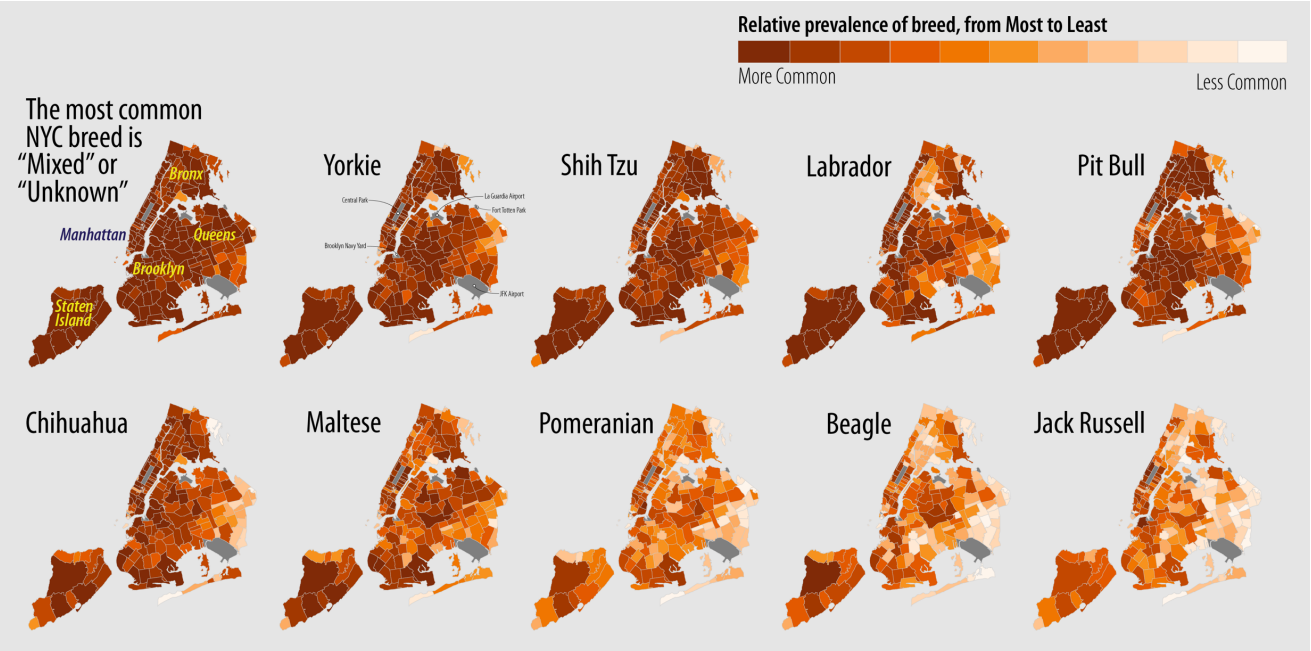
- Popularity (y-axis)
- AKC Dog group (upper right corner; Herding, Hound, Non-sporting, etc.)

1.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.)

1.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.

#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**:

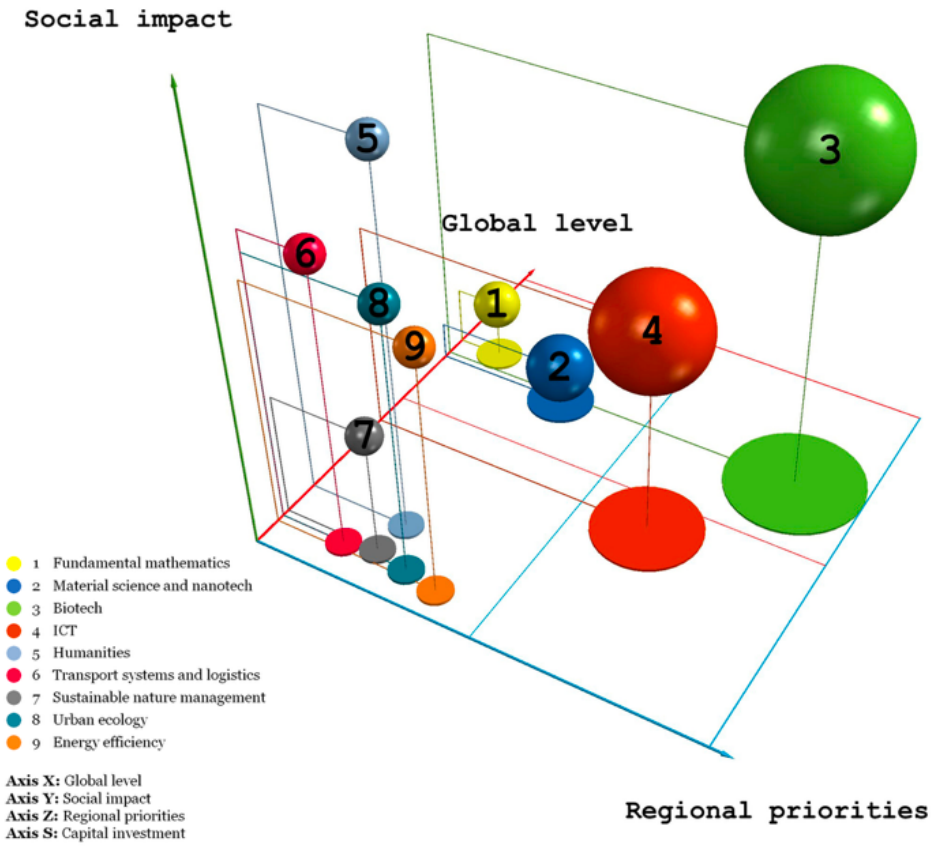
- Breed (Yorkie, Shih Tzu, Labrador, etc.)
- Relative prevalence of breed (Dark orange to warm white.)

2.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):

- Breed (Yorkie, Shih Tzu, Labrador, etc.)
- Relative prevalence of breed (Dark orange to warm white.)

2.D: In one sentence, make a judgment on whether the warm color scale used for "Relative prevalence of breed" is **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.)
- Social Impact (score for social impact of a domain, as reflected by higher/lower 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.)
- Social Impact (score for social impact of a domain, as reflected by higher/lower 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. Hierarchies are a special case of network data.

b. In a dynamic querying interface, users might use sliders to set filter criteria quickly and reversibly.

c. Instant mouseover tooltips on a choropleth map are an example of a force-directed layout.

d. On average, humans can track between 10 and 20 moving targets that change attributes.

e. In an adjacency matrix, node order does not affect the kinds of relationships that users observe.

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

a. Depth (3D)

b. Angle / tilt

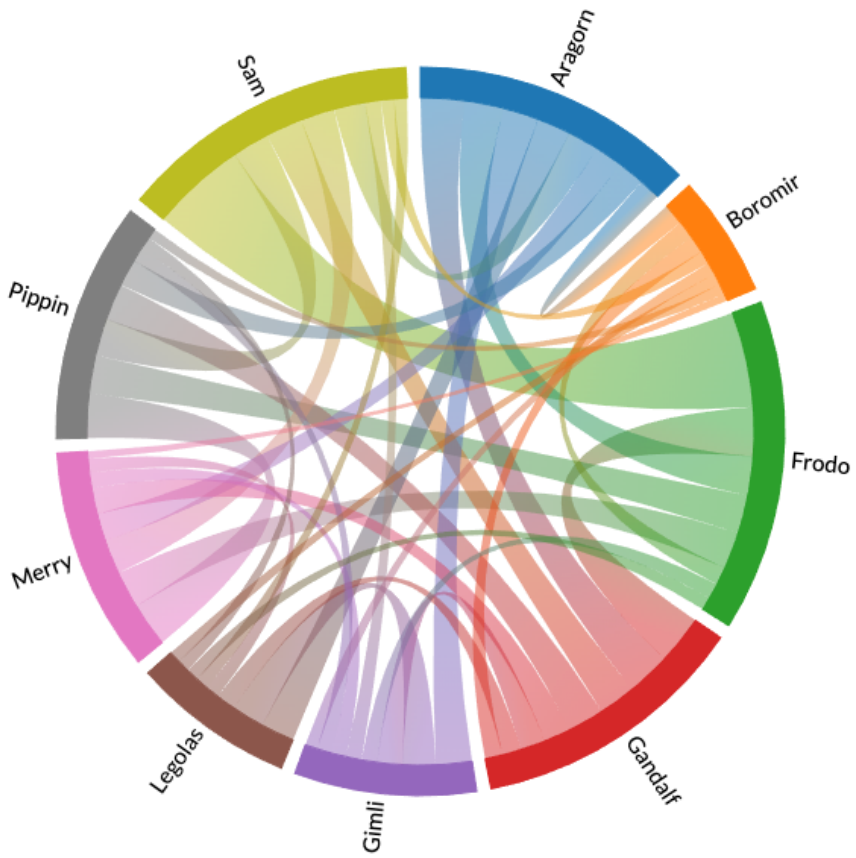
c. Unaligned position

d. Color luminance

e. Aligned length

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:



(How often members of the Fellowship in Lord of the Rings appear in the same chapter as one another.)

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 graph 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 in #8 and #9)

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

- Uses d3 to **select the `#scatter` SVG canvas** (which is 700px wide and 400px tall). Please **reserve 32 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**:
  - One with the class `"chart"` which will contain your scatterplot points
  - 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

- 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**.

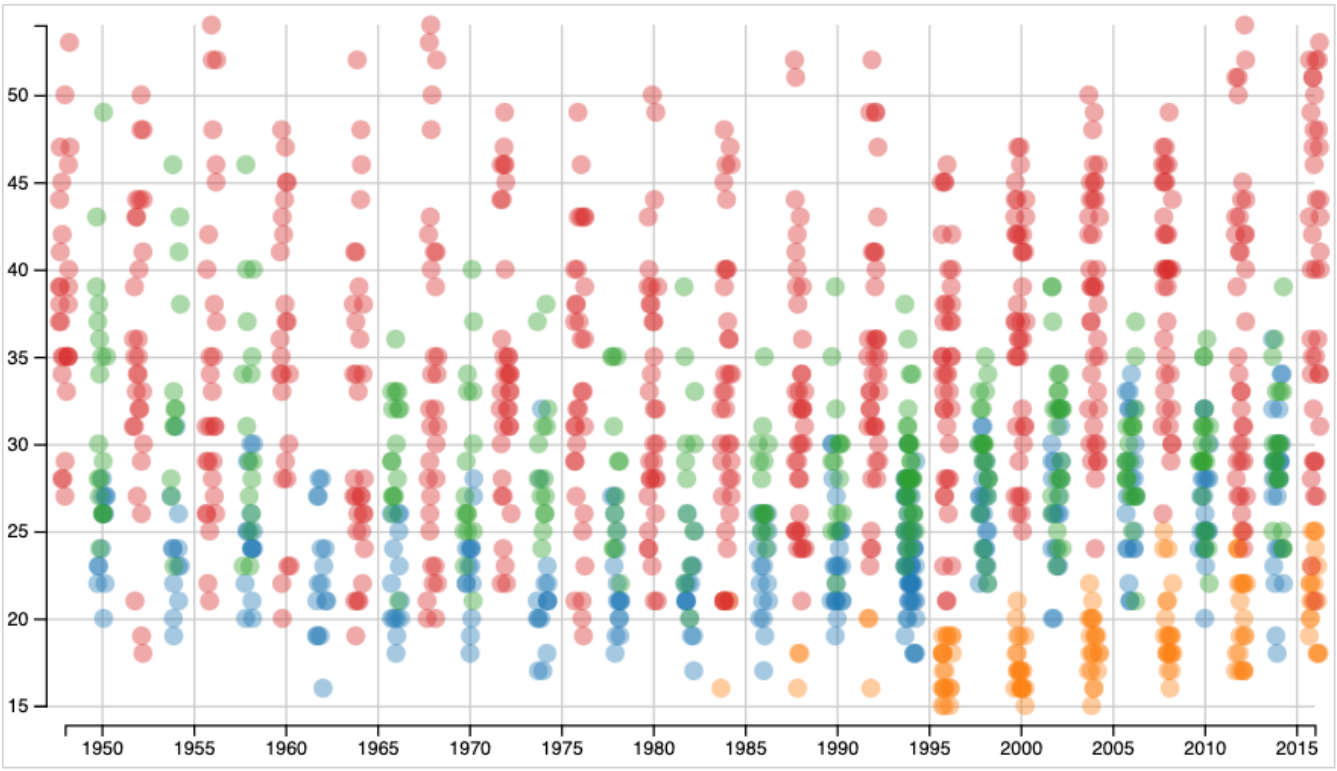
- **Populates the `".gridlines"` <g> elements** with `d3.axisLeft()` and `d3.axisBottom()` **gridlines**. (hint: we have already added the necessary CSS at the top of `index.html`)

- 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`).
  - 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 script, `8.js`, containing code for simulating the trajectory of a projectile. **Your job is to add Javascript code to the `updatePlot()` function to show all of the lines in the `trajectories` array and add code to the `submitButton` on click function to add a new trajectory and update the plot.** We have already written the code to generate the trajectory data, and there are four calls to `addTrajectory` and a call to `updatePlot` at the bottom of the file to get things started. Assuming you do your job correctly, your result should look like the example on the following page (before adding any new trajectories using the button)

Please refer to `8.js` for this problem:

- Check out the code and comments that are already included in the file.

- Begin by creating a **d3 line generator** which can populate a **`<path>` `d`** attribute for each element in the `trajectories` array. Refer to the sample output in the comments to help construct your generator.

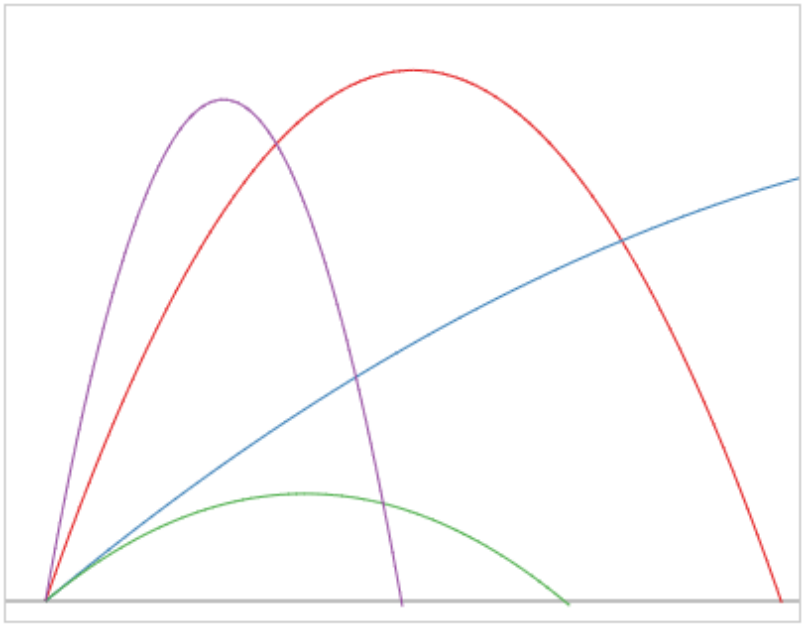
- In the `updatePlot()` function, please add Javascript code that:
  - Calls `console.log()` on `trajectories` (already done)
  - Uses a **d3 data join to append a new `<path>` element** to the `lineChart` object *for each of the elements* in `trajectories`. Be sure to use your line generator appropriately to figure out `d`. (hint: don't use `.datum()` here, since we want to make multiple `<path>`s)
  - Gives each line a 1px wide stroke using the `color` property on each element in `trajectories`.
  - Does not make use of `<canvas>` elements or functions

- In the `submitButton.on("click", function() { } )` call, please update the function so that it:
  - Gets the text the user entered into the velocity and angle text boxes
  - Converts the user's text into numbers (do not worry about handling bad input)
  - Calls `addTrajectory(velocity, angle)` to add a trajectory to the trajectories array
  - Calls `updatePlot()` to update the chart visuals so that the user can see the new trajectory.

(next page)

Example output for #8:

(showing the default trajectories added at the bottom of 8.js)



#9. In the zip file for this assignment we have included a data file, `penguin_flippers.csv`, which you will visualize for this problem. It contains count information for the length of Gentoo, Adelie, and Chinstrap penguin flippers from an ecological study. **We have already included an empty script file**, `9.js`, **where you will place your Javascript code for solving this problem**. 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.csv` and a **promise or await call** to load `"penguin_flippers.csv"` (no `..` or `/` please)

- Properly **converts the strings in** `penguin_flippers.csv` into numbers so they can be used, **handling any issues with data cleanliness gracefully**
  (hint: there will be data quality issues in both `count` and `flipper_length` !)

- Uses d3 to **select the** `#bars` **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 linear scales** for working with the data. While **range can be hard-coded**, **domain should be set to the proper data extent** using a call to `d3.extent()`. The two scales should be:
  - A linear scale for `"flipper_length"` values that has a range of `[10,490]`
  - A linear scale for `"count"` values that has a range of `[100, 0]`

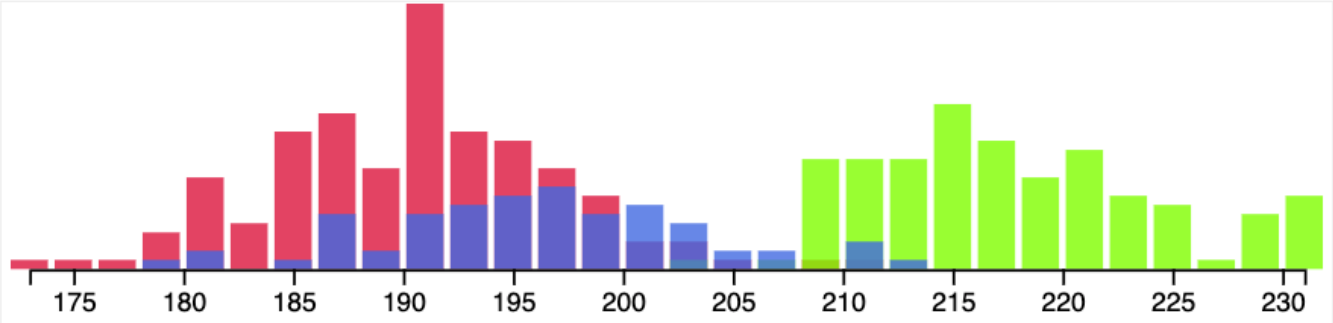
- Creates one **scaleOrdinal** color scale with a **predefined domain** and **predefined range** (do not use any default d3 color schemes, `d3.extent()`, or `Object.values()`). The color scale should:
  - Define the domain to specifically be `species` values `["Adelie","Gentoo","Chinstrap"]`
  - Define the range to be the following color array `["crimson","chartreuse","royalblue"]`

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

- Draws a **bar chart with bars positioned on the X axis by** `flipper_length` **and with a height corresponding to** `count` (maximum values 100px tall, minimum values 0px tall). Your code should:
  - Use a d3 `.data().join()` to create `<rect>` elements **for each point of data**
  - Draw your `<rect>` with a **width of 14 pixels centered at the correct X location** for the point
  - Fill the `<rect>` with the correct color for `species` using your color scale with no stroke and set the opacity of each rectangle to be 0.8.
    (Expect the bars at the ends of the chart to overhang a little bit)

(next page)

Example output for #9: (note, the CSV has been ordered so that blue Chinstraps sit on top of red and green)



#10. In the zip file for this assignment we have included a data file, `europe.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 500x500 `<svg>` element where the map will go.

In this problem you will be making a **choropleth map of Gini coefficients for different European countries**. I have already inserted Gini coefficient data into the dataset, `europe.topojson`. You will find it as **a property for each of the Features in the topoJSON FeatureCollection**. Make sure to include the dataset within your ZIP file so your code works properly.

Within `10.js`, please 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 #9)
- 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 **geopath generator**
- Builds a `d3.scaleSequential` using the `d3.interpolatePlasma` built-in **color scale**
- Sets the **domain of the scale** to the proper extent of Gini coefficient values in the dataset
  (hint 1: `d3.extent` works on the `.features` list of a topojson feature)
  (hint 2: you can find the Gini coefficient for country feature `"d"` at `"d.properties.gini"`)
- Adds `<path>` elements to the SVG using a **data join on the feature collection** of countries and the geopath generator created earlier
- **Does not use a `forEach` or `for` loop – only data joins**
- Fills in the `<path>`s based on their Gini coefficients using the **color scale** and **applies no stroke**

(next page)

Example output for #10: