CS/INFO 3300; INFO 5100
Homework 6
Due 11:59pm Wednesday, October 20

Goals: Practice using d3 to create a line chart

Your work should be in the form of an HTML file called index.html or index.htm with one <p> element per problem. 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 file containing your **HTML file and associated data files** (such as erie_fishing.json or mariokart_records.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 along with many other students' submissions.

(e.g. the grader has a folder, `~/hw6_cms`, where they start their own web server and unzip your assignment. If you just zip up `index.htm` and your data files as we ask, CMS will automatically make a sub-directory for your netID when we download it from the system. This means that your homework would appear at `~/hw6_cms/your_netID/index.htm` on the grader's computer. In their web browser, it would show as `http://127.0.0.1:8000/your_netID/index.htm` in the URL bar.)

1. In this problem you have the choice of two different datasets that involve creating a line chart:

- `erie_fishing.json` – A file containing historical fishing yields for different fish species in Lake Erie (a nearby Great Lake). Fishing in the lake has shifted from sustainable indigenous practices to commercial fishing (which led to species extinction) and finally towards an unsteady balance between conservation and commerce. This file will show how yearly fishing amounts (in 1,000s of pounds) vary for several species between the 19th and 20th centuries.
- `mariokart_records.json` – A file containing historical 'world record' three lap times for the retro video game Mario Kart 64. As speedrunners continually discover unusual shortcuts and tricks, the time it takes to complete three laps of a race track may dramatically decrease. This file may show those decreases as new records are set between 2007 and 2020.

Both datasets have been post-processed to have similar data structures, so **you will follow the same directions regardless of which dataset you choose to process**. Please pick **one dataset only**.

A. After a <p> tag for problem 1, place a **square SVG element 800px in height and 400px in width**.

In a `<script>` tag, begin by writing code that uses an `async function()` and `await` to load the data file. Run `console.log()` on your data to take a look at its structure. You will see several useful keys:

- `timeseries` contains an array of Objects. You will create a line for each of these Objects
- Within `timeseries` you will see that each Object has a `species` or `track` value with which to color the lines and a `values` array which will be given to a line generator to draw each line
- `earliest_date` and `latest_date` contain the earliest and latest date *strings* in the dataset
- `min_value` and `max_value` contain the lowest and highest *values* in the dataset (i.e. fishing yield or world record time)

B. Begin by creating <g> tags for your chart area and annotations using the usual pattern followed in class. Leave margin space on the bottom and left for some axis labels. Now, set up your scales:

- **Create a `scaleTime()` for dates in the dataset**. You already have access to the minimum and maximum dates to use as a domain, and you can easily compute the pixel range for your SVG canvas. However, the dates provided in the dataset are strings, not `JSDate` objects. You will need to **convert** them using `d3.timeParse()`. Examine the dates provided and set up a `d3.timeParse()` to properly parse them. Finally, use the `earliest_date` and `latest_date` keys along with `timeParse` to create your `scaleTime()`.

- **Create a `scaleLinear()` for values in the dataset**. You already have access to `min_value` and `max_value` to use as a domain, and you can compute the range from your SVG setup.

- **Create a `scaleOrdinal()` using a category color scheme of your choice**. You will later use this to color each line by its species or race track. If you choose not to use a built-in d3 schemeCategory, make sure that each line remains distinguishable from others.

C. Now use `d3.axis` functions to **create both axis labels and gridlines**. Please put **time on the x axis and values on the y axis,** with early dates to the left, and low values to the bottom. While you are free to adjust their formatting to be more appealing, it is fine if you leave them set to d3's defaults.

D. We will now set up to draw some lines. Begin by examining one of the Objects inside of the `timeseries` array. Your line generator will receive elements from the `values` key in the Object to draw each line. The `values` array has already been formatted to work nicely with d3's line generation tools. Depending on your dataset, the line generator is going to get points that look like:

```
{ "date": "1999-06-14", "track": "Kalimari Desert", "record_time": 129.46 }
```

OR

```
{ "date": "12-31-1915", "species": "Blue Pike", "total_caught": 23643, }
```

Create a `d3.line()` generator to draw your lines based on these data. Configure it as follows:

- Use your **time scale and your time parser** to determine the `x()` values for the generator. (hint: the line generator should process the `date` string to a JSDate and finally into a pixel location)
- Use your **value scale** to determine the `y()` values for the generator. (hint: use either `record_time` or `total_caught` depending on your dataset and a scale to get a pixel location)

E. **Use a data join to create one <g> tag for each of the elements in the `timeseries` array in your dataset** (hint: `____.data( dataFile.timeseries )`). Give each g tag a class. Use `.style()` and your color scale to set a `'stroke'` color style for the <g> tag based on either the race track or fish species. While in the past we would style the <path> directly, here the <path> will *inherit* its color from the CSS style of the g tag that contains it. (hint: use the `scaleOrdinal` you made earlier and the `d.track` or `d.species` key to set a stroke color style)

F. Now the tricky bit… You need to create a **<path>** element inside *each* of the **<g>** tags. This is similar to how we created circles within the lollipop **<g>** tags during lecture. Here, though, you need to use your line generator to create a **'d'** attribute for the path.

Using `.append()`, create a <path> element within each <g> tag. Use `.attr()` to set `'d'` for the path. When calling **`.attr()`**, you will need to give the line generator the **`values`** key from the **<g>** tag's data so that it can draw the line properly. (hint: **`.attr('d', d => lineGenerator( d.values ) )`**).

Finally, make sure that each path has a **`stroke-width`** of **`2px`** and a **`fill`** of **`none`**. As you already styled the stroke color in the g tag, you do not need to do any additional styling.

G. Lastly, **use a *nested data join* to create `<circle>` elements for each Object in the `values` array**. We have not seen this demo-ed in class. Here is a bit of sample code to get you started:

```
gTags.selectAll('circle')
     .data( d => d.values )  // "look at each G tag's data, d, and use d.values in
                                  this data join to make the circle elements"
     .join('circle')
     .attr( ...
```

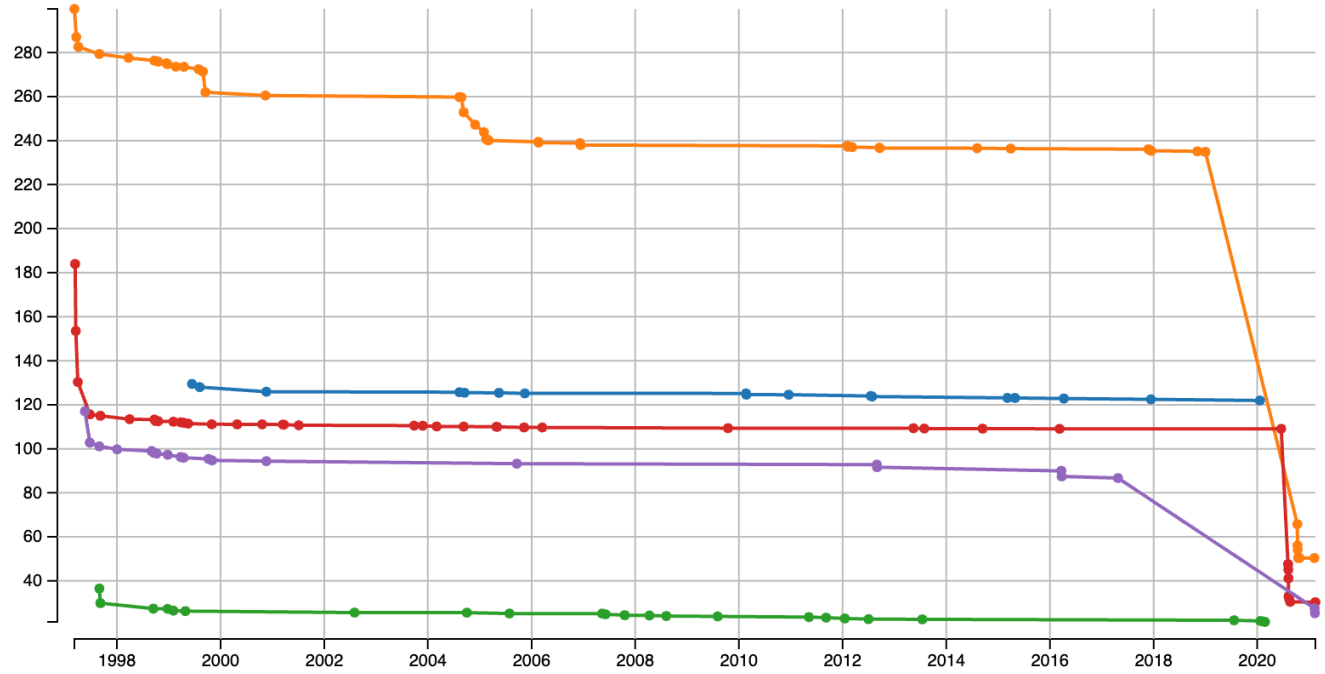
Since each g tag has different data, the nested data join will create circles for each of the elements in each g tag's specific array of **`values`**. As **`d.values`** is what was fed into the line generator, the circles will match up with the points of each line.

Each circle should have a radius of 2px. Give each circle a **`'fill'`** that matches the line using the **`track`** or **`species`** key. It should already receive a stroke color as part of the g tag.

NEXT PAGE

EXAMPLE FINISHED CHARTS:

`mariokart_records.json`



`erie_fishing.json`