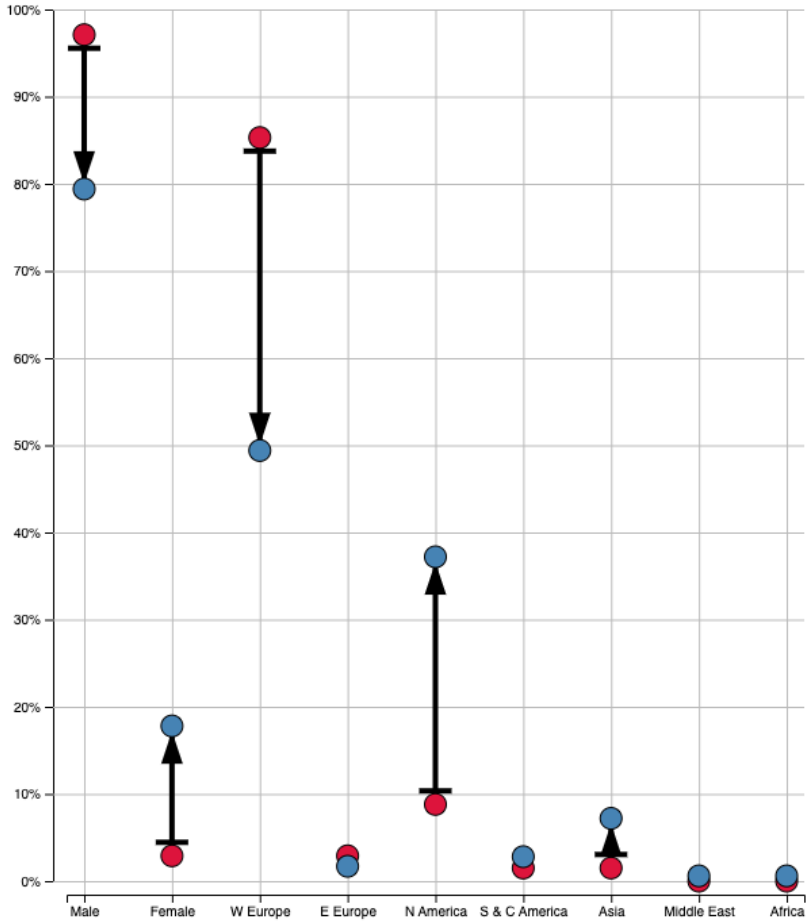
CS/INFO 3300; INFO 5100
Homework 9
Due 11:59pm Wednesday, November 8

Goals: Practice using d3 to create nested data joins and complex styling

1. In this assignment you will create a visualization of artist demographics in two major art history textbooks to show how specific kinds of representation have (or have not) changed over time. This dataset is drawn from Holland Stam's thesis work available here, where he coded images of art depicted in the textbooks. You can read his final report here. We have already processed this raw dataset into a series of simple demographic percentages for the years 1970 and 2020. Using this dataset, you will create a variation on a lollipop chart:

Example:



Red circles depict the percentage of images matching a specific demographic category in the 1970s, and blue circles depict percentages matching the category in the 2020s version of the textbook. Notice how certain kinds of representation improve, but many others do not despite overwhelmingly rich visual cultures in these regions.

A. Following your `<p>` element, please copy the following `<svg>` element into your HTML file. We will be using this element as-is for your visualization, as the `<defs>` section contains special definitions for the line and arrow markers you see in the example image.

```
<svg height="700" width="600" id="lollipop">
    <defs>
      <marker id="start" markerWidth="1" markerHeight="6" refX="-2" refY="3" orient="auto">
        <path d="M 0 0 V 6 H 1 V 0 Z" />
      </marker>
      <marker id="end" markerWidth="6" markerHeight="4" refX="7" refY="2" orient="auto">
        <path d="M 0 0 V 4 L 6 2 Z" />
      </marker>
    </defs>
  </svg>
```

Please also use the following margin block for your charts in your script tag, and remember to space your axes 10 pixels off of the main chart area:

```
let margin = { "top": 10, "right": 10, "bottom": 45, "left": 45};
```

Use `d3.json` and either `await` or `.then()` to import `artists.json`. Run `console.log()` to inspect the dataset. You will find that it contains an array of objects, each corresponding to a demographic `category`. These objects each contain two keys, `1970` and `2020` which have the percentage of painting images matching that category. For example, 97.1% of paintings in the 1970s editions were created by artists with male-coded names. Each one of the objects in the array will eventually become a "lollipop" in the example, comprised of a red `<circle>`, blue `<circle>`, and `<line>` element (all contained within a `<g>` tag). Your end HTML will look something like this:

```
▼<svg height="700" width="600" id="lollipop">
  ▶<defs>…</defs>
  ▶<g class="y axis" transform="translate(35,10)" fill="none" font-size="10" font-family="sans-serif"
   text-anchor="end">…</g>
  ▶<g class="y gridlines" transform="translate(35,10)" fill="none" font-size="10" font-family="sans-ser
   if" text-anchor="end">…</g>
  ▶<g class="y axis" transform="translate(45,665)" fill="none" font-size="10" font-family="sans-serif"
   text-anchor="middle">…</g>
  ▶<g class="y gridlines" transform="translate(45,665)" fill="none" font-size="10" font-family="sans-se
   rif" text-anchor="middle">…</g>
  ▼<g transform="translate(45,10)">
    ▼<g class="lollipop">
       <line x1="12.976190476190482" y1="18.705000000000016" x2="12.976190476190482" y2="132.8699999999
       9998" stroke="black" stroke-width="4" marker-start="url(#start)" marker-end="url(#end)"></line>
       <circle cx="12.976190476190482" cy="18.705000000000016" fill="crimson" stroke="black" r="8"
       stroke-width="1"></circle>
       <circle cx="12.976190476190482" cy="132.86999999999998" fill="steelblue" stroke="black" r="8"
       stroke-width="1"></circle>
     </g>
    ▼<g class="lollipop">
       <line x1="77.85714285714286" y1="626.295" x2="77.85714285714286" y2="530.19" stroke="black"
       stroke-width="4" marker-start="url(#start)" marker-end="url(#end)"></line>
       <circle cx="77.85714285714286" cy="626.295" fill="crimson" stroke="black" r="8" stroke-width=
       "1"></circle>
       <circle cx="77.85714285714286" cy="530.19" fill="steelblue" stroke="black" r="8" stroke-width=
       "1"></circle>
     </g>
    ▶<g class="lollipop">…</g>
    ▶<g class="lollipop">…</g>
    ▶<g class="lollipop">…</g>
    ▶<g class="lollipop">…</g>
```

B. Create two scales for your diagram. First, create a `scaleLinear` that you will use for *percentages.* Set the domain for this scale to run from 0 to 1. Set the range so that it runs along the y-axis of your chart. Next, create a `scalePoint` for the x-axis of your chart. This should contain each of the `categories` contained in the dataset (hint: `.domain(d3.map(data, d => d.key))`), a range between 0 and your chart width, and a `.padding(0.2)` call to add some extra space.

Now use d3 functions to make left axis labels, bottom axis labels, and gridlines for both axes. Offset the gridlines 10 pixels from the chart area (hint: -chartWidth-10) so that it resembles the example. The example also uses a `tickFormat` to add percentage signs, but this is not necessary in your file.

C. Create a new `<g>` tag called `chart` that includes a `transform` to account for the margins provided in part A.

Now, we will use a data join to create a set of `<g>` tags within chart, one for each of the elements in the dataset. Creating `<g>` tags with a data join works the same way as creating `<circle>` elements or any other kind of element. Give each of them a class, `lollipop`, and assign the result of your data join (a selection of all of your lollipop g tags) to a variable called `lollipops`. You can use `console.log` on lollipops to make sure that it has selected a group of all of your new `<g>` tags. At the moment you shouldn't see anything on your canvas besides your axis labels and gridlines.

One cool part of data joins we have not seen yet is that child elements that are *contained within* items made in a data join are also aware of their parent element's data. We will use this property to add two circles and a line to each of these lollipop tags.

D. Add this snippet of code after your data join:

```
lollipops.each( function(d) {
    console.log(this, d);
});
```

The `.each` function works just like a `.forEach` loop, but it runs on d3 selections! Observe the `console.log` output to see how it iterates through each of the `<g>` tags you made in lollipops. Start by selecting `this` with `d3.select` so that you can begin adding new elements into your `<g>` tag

Now, within `.each`, add code that `append`s the following elements *inside of each* `<g>` tag:

- A `circle` element with `cx` set to the correct position for the `category` and the `cy` set to the correct position for the `1970` percent value. Set its radius to `8`, assign a dark red fill color, and give it a 1px black border.
- A second circle element with `cx` set to the correct position for the `category` and the `cy` set to the correct position for the `2020` percent value. Set its radius to `8`, assign a blue fill color, and give it a 1px black border.
- Underneath the two circles, create a `line` element that connects the circles. Make sure that `x1,y1` corresponds to the center of the `1970`s circle and `x2,y2` corresponds to the center of the `2020` circle. Give the line a black color and a `stroke-width` of `4`.

You should now have a chart that looks mostly like the example, but without the arrow markers.

E. We are now going to add some arrow markers to our lines to help the viewer see whether the percentage is increasing or decreasing. The first challenge here is that we don't want to show markers if the space between the circles is too small. This leads to a confusing mess:

Within your `.each` loop, add an `if` statement that measures the overall distance of the `line` element you just made (hint: you can compute the pixel distance between your `y1` and `y2` values). If the length of the line is greater than `30` pixels, then we will add some arrow markers.

To add markers to a line, you first need to use a `<defs>` section to specify what the markers look like. We have already done this for you in the starting code snippet. To *apply* markers to your line elements, we need two add two attributes:

```
.attr("marker-start", "url(#start)")
.attr("marker-end", "url(#end)")
```

These attributes tell the line element to look for a marker with the `id` of `start` and put a copy of it on the line. Likewise, it looks for a marker with the `id` of `end` to put at the end of the line. Because we have drawn a little line marker and arrow marker in `<defs>` copies will appear on the lines.

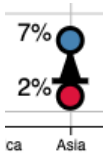
If you have done this correctly, your chart should look like the example image on the first page.

F. Finally, we will add some mouseover interactions. Begin by adding two next `<text>` elements to your chart `<g>` tag and assign them to `label1970` and `label2020`.

Now, using `lollipops.on()`, set up a `mouseover` and a `mouseout` event handler.

Your interaction should have the following behaviors:

- When you mouseover the lollipop, both circles should get a `stroke-width` of 3 (hint: `d3.select(this).selectAll("circle").attr( ... ) )`
- When your mouse leaves the circles, the stroke width should be reset.
- When you mouseover a lollipop, a text label should appear next to both the 1970 and 2020 circles showing the percentage of art images matching the category (i.e., the values used to position the circles).
  - If the lollipop is on the left side of the chart, then the labels should appear to the right of the circles:
  - Else, the lollipop is on the right side of the chart, so the labels should appear on the left side of the circles:



- When you leave a lollipop circle, the text labels should disappear.
- You are welcome to use d3.format() to make the text labels include percent signs and rounding as in the example images, but this is not required.

BONUS. (no extra credit offered; included for completionists only)

While in this homework we used .each() to iterate through each of the <g> tags, this is not actually necessary to do using d3. Canonically, it's actually a bit of a cludge.

The better way to do this would be to make use of the fact that d3 is happy to iterate over selectAll for you. This way, you can complete the assignment without ever using the keywords for, forEach, or each. As a hint of how to get started on this, comment out your .each code. Then, include this snippet and see what happens to your HTML:

```
lollipops.append("circle").attr("cat", d => d.category)
```

Notice how the cat attribute is actually changed using the <g> element data, because child elements also can access parent data. Also notice that the .append actually iterated through all lollipops. You might see where to go from here to do the others.

There is another, technically even better way to do this. This means changing .join("g") in your basic lollipop form to one that uses enter => enter. This will allow you to manually add each of the elements to your lollipops as you create them. Update can even potentially adjust the circle positions and lengths if you think they will change over time.

If you choose to do this bonus, please turn in a version of your HW that uses the plain .each call so that graders do not get confused.