CS/INFO 3300; INFO 5100
Homework 6
Due 11:59pm **Friday** April 17

Goals: Practice using d3 to create paths based on TopoJSON files. Explore map projections and evaluate their trade-offs. Get more experience building choropleth maps.

Your work should be in the form of an HTML file called index.html with one <p> element per (sub)problem. Wrap any SVG code for each problem in a <svg> element following the <p> element(s). For this homework we will be using d3.js and topoJSON. In the <head> section of your file, 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>
```

Create a zip archive containing your HTML file plus ALL associated data files and upload it to CMS before the deadline. You will be penalized for missing data files or non-functional code.

1. There are many different ways to project Earth's sphere into 2-dimensional space. Each one provides benefits and imposes costs for map-makers. In this problem you will construct 3 different projections and identify their trade-offs.

A. We have included with this homework a topoJSON dataset containing outlines of world countries (as of 2016) called world_110m.json. Using .then() or await, load these data into a variable called "world". Create two more variables: "countries" which contains the topojson.feature for the world.objects.countries GeometryCollection, and "countriesMesh" which contains the topojson.mesh for the world.objects.countries GeometryCollection. We will use these two variables for the rest of this problem.

B. In the HEAD portion of your HTML, create a <style> block. Create CSS styles for .country with a lightgrey fill and none stroke, .outline with a black stroke, 1px stroke-width, and none fill, and .graticule with a none fill and lightgrey stroke.

C. In the HTML part of your submission, create an SVG element 600px in width and 400px in height. Give it the ID "mercator". Using d3 in the <script> section of your submission, create a d3.geoMercator projection fit to the size of this canvas. Construct a d3.geoPath() path generator that uses this projection. Before drawing countries, add a "graticule", or set of latitude and longitude lines, to the canvas by adapting the following sample code:

```
map.append("path") // Add a single path to your SVG canvas
  .attr( "class", "graticule" )
    // with the class "graticule" to use our CSS style
  .attr( "d", pathGen( d3.geoGraticule10() ) );
    // and call your path generator on the output of d3's
    //  built-in d3.geoGraticule10 function to set "d"
```

After adding a path for the graticule, **use a data join to construct filled paths for each country** (hint: map.selectAll("path").data(countries.features).join()), giving them the class "**country**" so that they use the CSS styles you added earlier. Make sure that you use your path generator to set the "d" element. Finally, **use a .datum() call and your path generator to create a single "path" containing the countriesMesh** you created earlier with the class "**outline**" (hint: this subproblem will follow the same general pattern used in class notes)

D. In a <p> tag, write about 3 sentences identifying a) an **advantage** of this Mercator projection, b) a potential **draw-back** of this projection or places where it might be misleading, and c) a specific **example use case** for which it would be well suited.

E. In the HTML part of your submission, create an SVG element **600px in width** and **400px in height**. Give it the ID "**equalEarth**". Using d3 in the <script> section of your submission, create a **d3.geoEqualEarth** projection fit to the size of this canvas. Construct a d3.geoPath() path generator that uses this projection. **Reusing your code from part C**, **add a graticule with class "graticule", country paths with class "country", and a path for the country mesh with class "outline" to the SVG canvas.** Make sure that your reused code makes use of this new projection and the correct SVG. In a <p> tag, write about 3 sentences identifying a) an **advantage** of this Equal Earth projection, b) a **draw-back** of this projection or places where it might be misleading, and c) a specific **example use case** for which it would be well suited.

F. In the HTML part of your submission, create an SVG element **600px in width** and **400px in height**. Give it the ID "**azimuthalEqualArea**". Using d3 in the <script> section of your submission, create a **d3. geoAzimuthalEqualArea** projection fit to the size of this canvas. Construct a d3.geoPath() path generator that uses this projection. **Reusing your code from part C, add a graticule with class "graticule", country paths with class "country", and a path for the country mesh with class "outline" to the SVG canvas.** Make sure that your reused code makes use of this new projection and the correct SVG. In a <p> tag, write about 3 sentences identifying a) an **advantage** of this Azimuthal projection, b) a **draw-back** of this projection or places where it might be misleading, and c) a specific **example use case** for which it would be well suited.

(next page)

2. In this problem we will make a choropleth map of NY 2016 presidential election results. We have provided a topoJSON data file that contains the shapes of electoral districts in NY and integrated data on votes for the two leading candidates into the dataset.

A. In the HTML portion of your submission, create an SVG canvas in **800px in width** and **700px in height**. In the <script> portion of your submission, use **.then()** or **await** to load the **new_york_districts.json** dataset into a variable called "**nyd**". Create a variable "**districts**" that contains the **topojson.feature** for the **nyd.objects.districts** GeometryCollection. Do not make a topojson.mesh for this problem. Create a **d3.geoMercator** projection fit to the size of this canvas. Construct a **d3.geoPath() path generator** that uses this projection.

B. Construct a **d3.scaleLinear scale** for your electoral choropleth colors. Set the **domain to [30,50,70]** so that we can assign a color to the midpoint. **For the range, provide a list of three colors.** One end should be a **red color** and the other a **blue color**. As this is a **divergent scale**, set the **middle color to be a light grey**. Add **.clamp(true)** at the end of the function chain on your scale after domain and range to make sure numbers above and below your domain don't create crazy colors. Also add **.interpolate(d3.interpolateHcl)** to the chain to make use of a better color interpolator for blending between your chosen colors.

C. Using a **data join**, create paths for each of the **districts.features** in the dataset. Give each district a **1px white stroke**.

D. Each district in the dataset has a dictionary attached, "**properties**", which contains two values we might use, "**percent_trump**" and "**percent_clinton**". Choose one of these values and use it, combined with your color scale, to **fill the districts with either a red or a blue color based on voting trends**. If you choose to use the "percent_trump" variable, then districts with higher values ought to be more red (by US cultural conventions). If you make use of the "percent_clinton" variable, then higher values should be more blue.
(hint: d => scale(d.properties.percent_clinton))

E. Finally, **add a white circle of radius 10 to mark the location of Ithaca, NY**. Ithaca is located at **latitude 42.443333** and **longitude -76.5**. (hint: you can use your **projection** to determine the x and y pixel locations for the circle, see **d3.geo documentation** and be careful of your latitude/longitude parameter order).