DDA 2003/MDS 6112 Assignment 5

Start: April 3 at 9:00 AM

End: April 10 at 11:59 PM

Correspondence: Haotian Ma (haotianma@link.cuhk.edu.cn)

1 Description

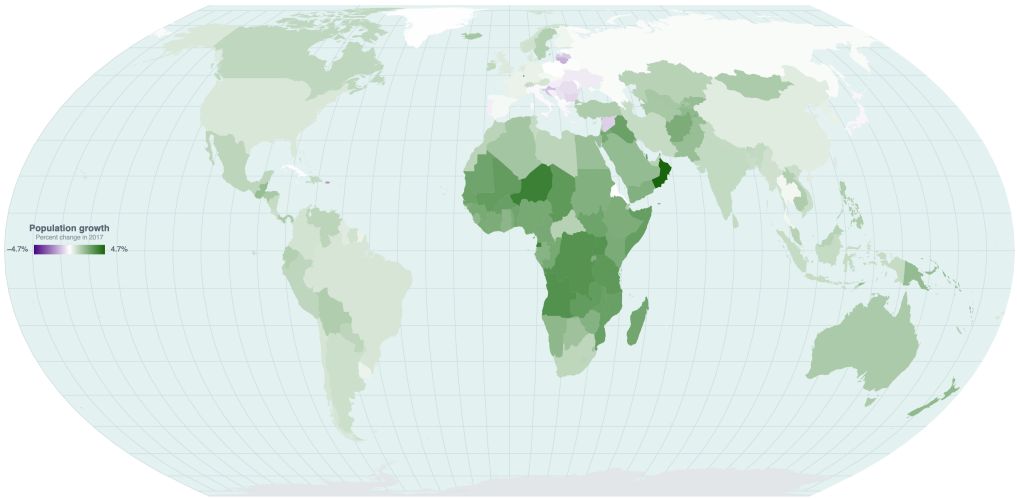
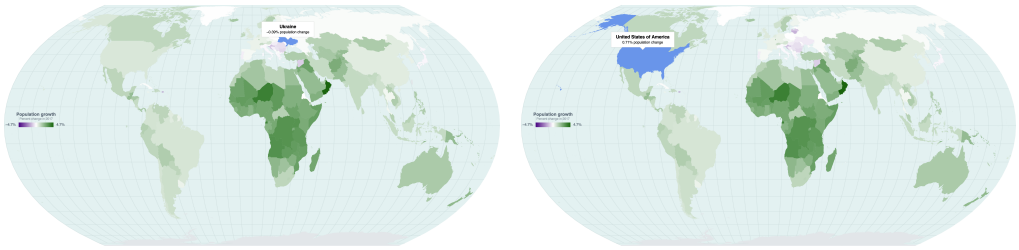


Figure 1: The visual result of assignment 5.

This assignment aims to help students get familiar with map visualization using D3.

In this assignment, you are asked to produce the following visualization results.

- Given a geospatial in a .json file, visualize the data using a choropleth map, as shown in Figure 1.
- Create mouse hover event to display population change for each country in the map, as shown in Figure 2.



(a) (b)

Figure 2: Interaction on the map.

2 Requirement

- Process data.
- Projection.
- Create color scale.
- Visualize data.
- Set up interactions.

Online Resources The following online resources will be helpful in finishing this assignment.

- Map Visualization in D3
- Map Visualization in D3 Tutorial
- D3-geo
- D3-geo-projection

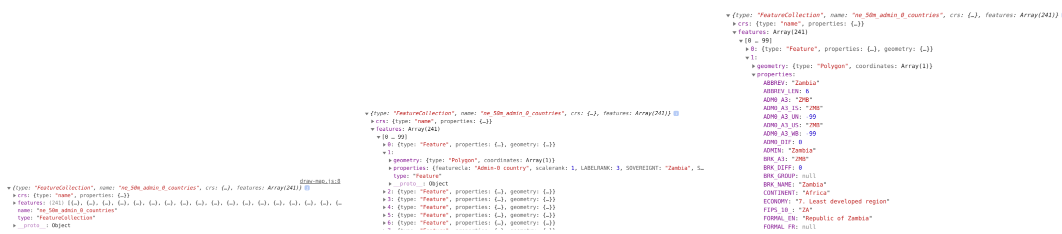
3 Instruction

3.1 Data Format

GeoJSON is a format used to represent geographic structures (a geometry, a feature, or a collection of features). A GeoJSON object can contain features of the following types: Point, MultiPoint, LineString, MultiLineString,

Polygon, MultiPolygon, GeometryCollection, Feature, or FeatureCollection. If you're curious, check out the spec38 for more information. There are many sources of GeoJSON files. In this assignment, we use Natural Earth39, which is a large collection of public domain map data of various features, locations, and granularities.

Once the .json data is read successfully, we can see the following output, shown in Figure 3.



(a) (b) (c)

Figure 3: Data structure in GeoJSON.

Next, we read a .csv data set to show population growth. The output of this .csv file is shown in Figure 4.

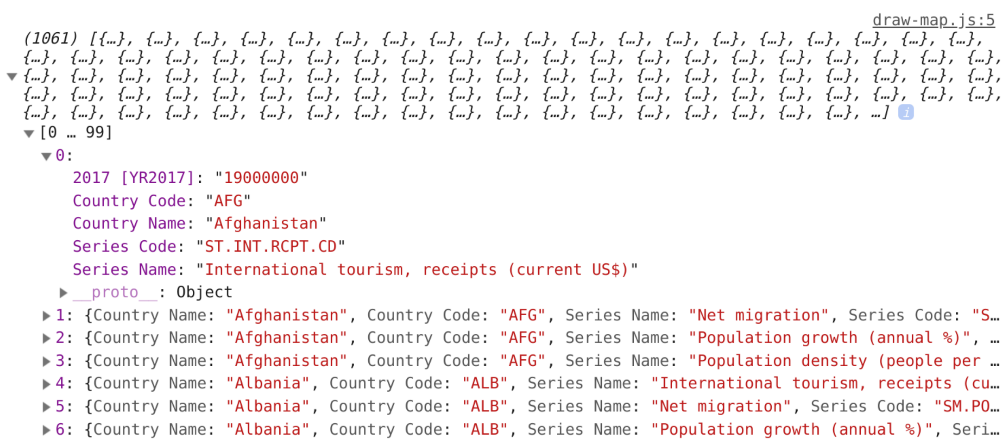


Figure 4: Data structure in CSV.

3.2 Process Data

An easy way to look up the population growth number is using a country name. We could find a matching item in the dataset array, but that could be expensive. Instead, we can create an object with country ids as keys and the population growth amount as values. To start, we initialize an empty object (i.e., metricDataByCountry). Then we will go over each item in the dataset array. If the item's "Series Name" doesn't match the metric, we won't do anything. If it does match, we'll add a new value to the metricDataByCountry object: the key is the item's "Country Code" and the value is the item's "2017 [YR2017]" number. Note that these values are stored as Strings — we'll convert the value to a number by prepending +, and default to 0 if the value doesn't exist.

3.3 Projection

Despite what flat-earthers might say, the Earth is a sphere. When representing it on a 2D screen, we'll need to create some rules — these rules are the projection. Imagine peeling an orange and turning the skin into a flat sheet — even with slicing it in various places, it's impossible to do perfectly. Projections will use a combination of distortion (stretching parts of the map) and slicing to approximate the Earth's actual shape. There are many projections out there. D3-geo has 15 projections built in, but many more in a D3 module that's not included in the core build: D3-geo-projection. Figure 5 shows a collection of different projections.

You are asked to choose one projection method to visualize the geospatial data using the D3-geo or D3-geo-projection library. It is not required to choose the same projection as shown in Figure 1.

Each of the projections are implemented in either d3-geo or d3-geo-projection. We can use these projection functions to convert from [longitude, latitude] coordinates to [x, y] pixel coordinates. Essentially, a projection function is a scale in the geographic world. Each projection function has its own default size (i.e., range). But we want our projection to be the same width as our bounds. To update our projection's width, we can use its .fitWidth() method, which takes two parameters: (1) the width in pixels and (2). a GeoJSON object. When we call this method, our projection will update its size so that the GeoJSON object (2) we pass will be the specified width (1). Similarly, to get the height, we can use d3.geoPath() is similar to

the line generator (d3.line()). When we pass it our projection, it will create a generator function that will help us create our geographical shapes. An output of d3.geoPath() is shown in Figure 6. Next, we also need to find out how tall the path would be. d3.geoPath() has a .bounds() method that will return an array of [x, y] coordinates describing a bound box of a specified GeoJSON object.

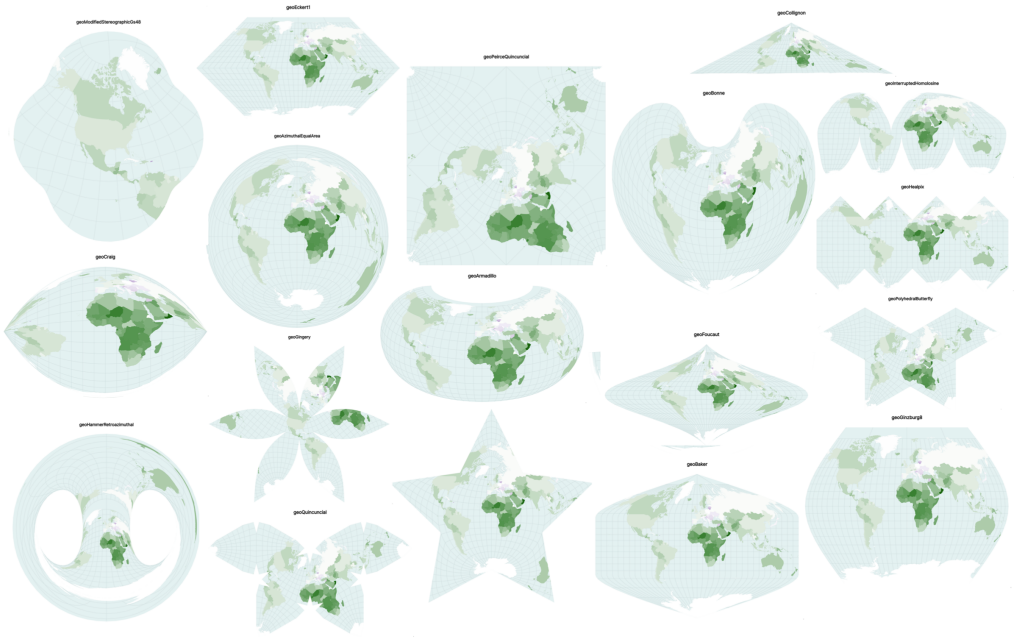


Figure 5: Collection of projections.

3.4 Color Scale

Next, we need to create our scales. Typically, we would create x and y scales, but those are covered by our projection. We'll only need a scale to help us turn our metric values (population growth amounts) into color values. We can grab all of the population growth values by using Object.values(), which returns an object's values in an array. Then we can extract the smallest and largest values by using d3.extent().

While we can create scales that have one minimum and one maximum value for both their domain and range. But we would like to create piecewise

```
M137.6850612805441,2.842170943040401e-14L337.85,2.842170943040401e-                draw-map.js:38
14L538.0149387194559,2.842170943040401e-
14L573.2969871814703,17.046318574021143L601.7388721244954,35.11582806465225L627.4033152905058,57.
060601880579455L648.1636120831404,81.74126598217812L656.4778194202438,94.83731333047876L663.34592
97549905,108.31468205596332L668.7284340918884,122.09019037238977L672.5945365846244,136.0854555836
9653L674.922602578265,150.22546379143057L675.7,164.4373301346346L674.922602578265,178.64919647783
864L672.5945365846244,192.78920468557268L668.7284340918884,206.78446989687944L663.3459297549905,2
20.5599782133059L656.4778194202438,234.03734693879045L648.1636120831404,247.1333942870911L627.403
3152905058,271.8140583886898L601.7388721244954,293.75883220461697L573.2969871814703,311.828341695
24807L538.0149387194559,328.874660269269l6L337.85,328.874660269269l6L137.6850612805441,328.874660
26926916L102.40301281852976,311.828341695248O7L73.96112787550464,293.75883220461697L48.2966847094
94286,271.8140583886898L27.536387916859724,247.1333942870911L19.222180579756184,234.0373469387904
5L12.35407024500961,220.5599782133059L6.9715659081117,206.78446989687944L3.1054634153755956,192.7
8920468557268L0.7773974217350315,178.64919647783864L5.684341886080802e-
14,164.4373301346346L0.7773974217350315,150.22546379143057L3.1054634153755956,136.08545558369653L
6.9715659081117,122.09019037238977L12.35407024500961,108.31468205596332L19.222180579756184,94.837
31333047876L27.536387916859724,81.74126598217812L48.296684709494286,57.060601880579455L73.9611278
7550464,35.11582806465225L102.40301281852976,17.046318574021143L137.6850612805441,2.842170943040
401e-14Z
```


Figure 6: Data structure of d3.geoPath().

scales, which are basically multiple scales in one. If we use a domain and range with more than two values, d3 will create a scale with more than two "anchors". In this case, we want to create a scale with a "middle" population growth of 0% that converts to a middle color of white. Population growth amounts above 0 should be converted with a color scale from white (0) to green (5), and population growth amounts below 0 should be converted with a color scale from red (-2) to white (0). Creating a piecewise scale like this is easier than it seems: just add a middle value to the domain and the range. A possible choice is that.

- create a scale which scales evenly on both sides (eg. .domain([-5, 0, 5])). This approach will let the viewer see if countries are shrinking at the same pace as countries that are growing, since a light red (-2.5%) will correlate to a light green (2.5%).

- create a scale which scales unevenly from white to red and white to green (eg. .domain([-2, 0, 5])). This approach will maximize the color scale, making distinctions between countries with different growth rates clear. However, it can be confusing when a light red color (-0.1%) doesn't match the scale of a light green color (2.5%).

An example of a color scale is shown in Figure 7.

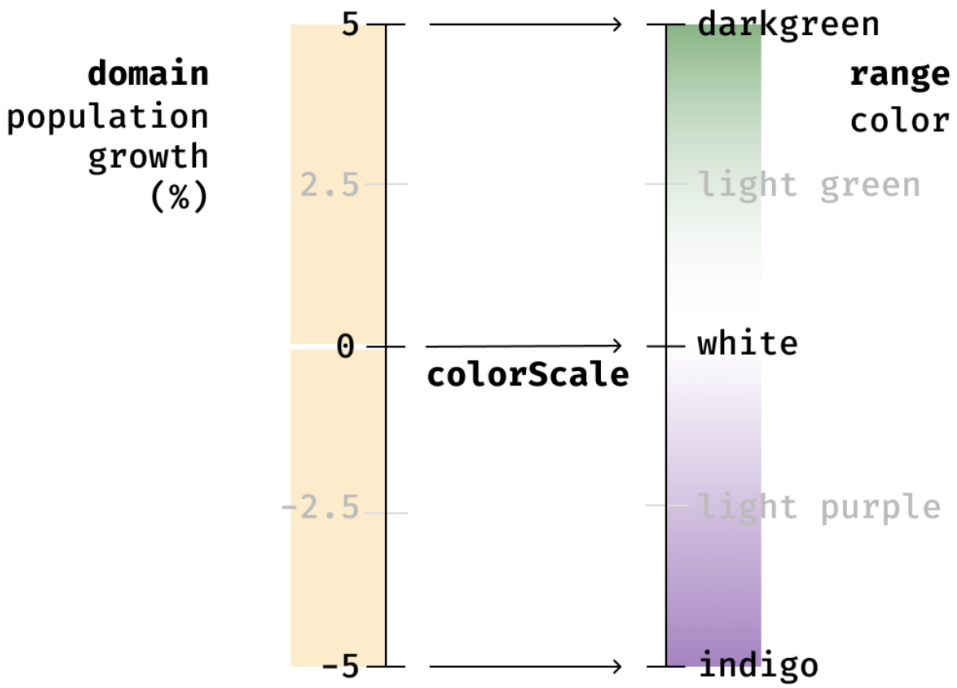


Figure 7: Color scale.

3.5 Visualize Data

Next, we will draw geospatial data. To start, we'll draw our Earth outline. It makes sense to draw this first, since SVG elements layer based on their order in the DOM, and we'll want all other elements to cover this outline. Remember that our pathGenerator is aware of the projection we're using and will act as a scale, converting a GeoJSON object into a ¡path¿ d string. Next, let's display a graticule on our map. A graticule is a grid of latitudinal and longitudinal lines — essentially tick marks for a map. These are helpful for aligning map elements that are far apart, and also for conveying how the projection is distorting the globe. We can easily create a GeoJSON graticule — d3.geoGraticule10() will generate a classic graticule with lines every 10 degrees. The pathGenerator() knows how to handle any GeoJSON type — we can use it to draw the graticules. Finally, let's draw our countries. We'll use the data join method. First, we'll select all of the elements with a class of .country. Remember: even though these elements don't yet exist, this is priming our d3 selection object to bind data to similar elements.

Next, we'll bind our data to our selection object. Because d3 will create one element per item in the dataset we pass to .data(), we'll want to use the list of features instead of the whole object. Next, we'll use .enter().append("path") to tell d3 to create one new ¡path¿ for each item in our list of countries. We'll give each of these ¡path¿s a class of "country" and then pass it straight to our pathGenerator() to get a d string, generating the shape of each country. The last is we want to color our countries based on their population growth amount. We'll need to set their fill by looking up the country's id in the metricDataByCountry object we created at the top of our file, then passing that id to our colorScale(). Since some countries might be missing from our dataset, we'll want to indicate that on our map by coloring them white.

3.6 Mark location

One of the fun parts of building in the browser is that we can interact with the user. Let's take advantage of this by showing user's location on the map. Modern browsers have a global navigator object, which provides information about the device accessing the website. navigator.geolocation has a method for grabbing the browser's location: .getCurrentPosition(). We can pass .getCurrentPosition() a callback function that has the user's location. The browser will ask to access our location the first time we load the page. The callback will only run if we click Allow. After clicking allow, we can see the user's current location on the map, as shown in Figure 8.

3.7 Set Up Interactions

Lastly, we need to give the user more information when they hover over a country. First, we want to initialize our mouse events using the .on() method for our countries' elements. Next, we want to grab the tooltip element — this is already created in our index.html file. Having a static tooltip reference outside of our mouse event callbacks keeps us from having to look it up every time a country is hovered. Next, let's initialize our onMouseEnter() and onMouseLeave() callback functions and have them toggle our tooltip's visibility. Next, let's populate the text of our tooltip. Remember that the first parameter of our mouse event (that we've named datum) contains the data bound to the hovered element. We can use this to grab the country name and population growth value. We can check the index.html file to find

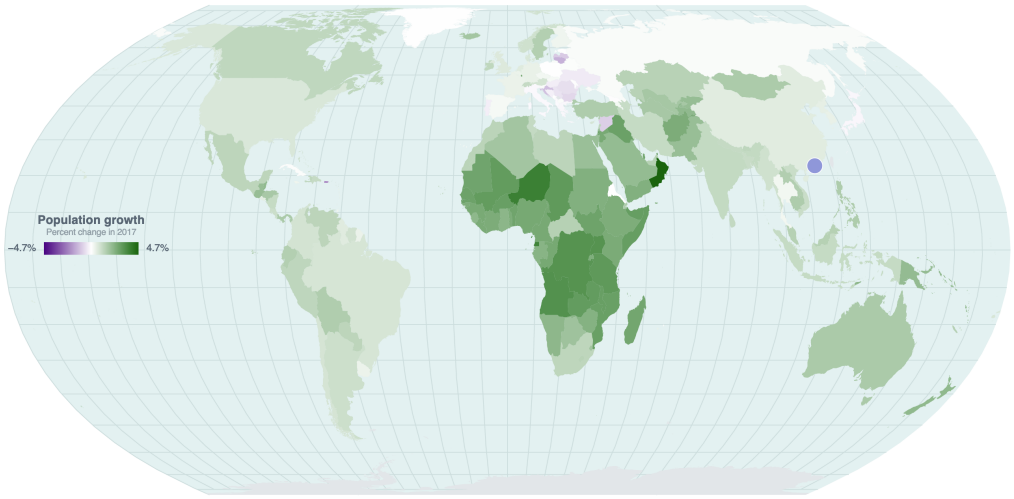


Figure 8: Mark location.

the ids of the elements which will contain this text. In order to know where to position the tooltip, the pathGenerator also has a .centroid() method that will give us the center of the passed GeoJSON object. Let's see what we get when we ask for the center of our hovered country's bound data. Remember that we want to shift the information with our top and left margins, since our tooltip element lives outside of our bounds. We'll need to use calc() to accommodate the tooltip's own width and height.

4 Evaluation

In total, there are 100 points in this assignment. A detailed evaluation is provided here.

1. Process data. (10 pts)
2. Projection. (20 pts)
3. Create color scale. (10 pts)
4. Visualize data. (40 pts)
5. Set up interactions. (15 pts)

6. Submission (5pts). Please compress your code and a readme file (optional) into a zip file and submit the zip file to Black Board. The readme file can include descriptions that help the grader run the interface successfully.

Note that a penalty of 10 pts will be given to those students who submit the assignment one day after the deadline. A penalty of 20 pts will be given to those students who submit the assignment two days after the deadline. Submissions three days after the deadline will not be graded.