**Goals of post:**

* Learn map projections in D3
* Learn data binding in D3
* Show someone how to create Nytimes election map

Prerequisite coding knowledge:

* Method chaining for D3
* Basic javascript
* Html, css
* GeoJSON

**WHEN FIRST ENCOUNTER CODE SAY GO TO GITHUB REPO, WITH FILE NAME TO GET INFORMATION**

Introduction:

If you’re a political junkie like me, you probably have wondered how the New York times builds one of its addictive election maps.

In this tutorial, not only will you master how to make one of these maps, but also learn how to do map projections and data binding with D3.

Our map will be of the Texas senate race in 2018 (Beto!). To create this map, we will first learn about data binding in D3. Then we will learn how to display geometric objects in D3. After that, we will be ready to make the map!

All data for these demos are in this github repo. For all tutorials, add:

<script src="https://d3js.org/d3.v5.min.js"></script>

By the end of this tutorial, we will learn how to be able to make a map like this: **BAM.** (the link)

Let’s get to it:

WRITE PREMISE FOR EACH: EG

**Data-Binding in D3**

Data-binding in D3 is simple, but many tutorials overcomplicate it. Here is the concept.

(If your just starting, I recommend reviewing D3 selections.)

Let’s say you have 5 circles, and are looking to place a label for each of them with the array: [ ]

// show 5 circles

If the labels are stored in an array, we would place labels in the circles the following way:

// code

“.data(dataset)” runs a for loop using *dataset*, where the body of the for loop is the code below it.

Now there are two scenarios in data-binding we have yet to cover, when we have more data than objects to bind, or when we have less data than objects to bind.

**.enter( ) - More data than objects**

Example: Let’s say two of our circles goes missing. We now have 3 circles, but five labels. How could we create two extra circles with labels?

Well we could add labels to the three circles as done before, but two extra data elements will be left.

[show example]

.enter( ) takes those elements

As noted by [d3indepth](https://d3indepth.com/enterexit/), this method is commonly used when we have an empty selection, ie no objects selected to begin with.

If you want a selection that reaches all the data elements, not just those extra two, after .enter( ), you merge(d3 object ) which combines the two data sets.

**.exit( ) – More objects than data**

Example: Let’s say two of our labels goes missing. We now have 5 circles, but three labels. How could we remove the two extra data circles?

Similar idea, but when .exit( ), the selected region is the two extra circles. .exit( ).remove( ) will eliminate those circles.

And that’s it! With data-binding out of the way, let’s get to projections.

**Displaying Geometric Objects in D3**

Mesh 100% should be included

Why topojson 100% needs to be included

**Projections**

Every place in the world is mapped using longitude and latitude coordinates. Given a globe, latitude are the coordinates going xxxxxxxxxx, latitude are the coordinates going yyyyyyyyy.

[map with clear lat and lon]

As d3 map gurus, we want to show our maps in two dimensions, not three. Therefore, we need to be able to create a **projection** that enables us to go from three to two dimensions. In essence, projection([longitude, latitude]) 🡪 x and y coordinates

With D3, there are all kinds of projections. Some preserve distance, such as XXXX, and some preserve area, such as YYYY. All of them come with their advantages and disadvantages. Below are two projections compared to each other, XXX, YYY:

[image of two projections compared to each other]

With D3, not only can we call on these projections, but we can also modify them through scaling, rotating, and move left/ right. This [link](https://gist.run/?id=f7ece0ab9a3df06a8cecd2c0e33e54ef) provides a great intuitive feel for the different transformations we can do given a projection. This [link](https://github.com/d3/d3-geo) provides all the different ways to modify a D3 projection.

See below how projection produces a value:

**GeoJSON**

However, what if we want to project not just a point, but any shape? Welcome the GeoJSON.

GeoJSON’s can encode any shape, from points, lines, and polygons. So by learning how to render any GeoJSON, we can render any shape. GeoJSON’s are just like JSON’s, but built for containing geographic information.

There are three different kinds of GeoJSON’s:

* **Geometric objects in space:** for example, Point, LineString, and Polygon. There are seven geometric objects total.
* **Feature**: contains a geometric object and information about that geometric object
* **FeatureCollection:** Collection of features

The keys for a geometric object are the “type”, and “coordinates”. The value for “type” is which one of the seven shapes the geometric object is, and that for “coordinate” is a list of points that make the object (points written in long, lat order). Here is an example:

[example for polygon]

Each Feature contains a “type” (always called “Feature”), “geometry”, and “properties”. The value for “geometry” is a geometric object in GeoJSON, and the value for “properties” is a JSON containing information about the object. Here is an example:

[example code]

Each FeatureCollection contains a “type” (always called FeatureCollection), and “features” (a list of Feature objects.)

Here are how GeoJSON’s work in action. To create the map for the Texas Senate race, we need a map to show the state of texas and all of its countries.

To have a geojson that can represent this map, our geojson object is a FeatureCollection. Within that FeatureCollection, there are 254 features, one for each county. Each country is represented by a “MultiPolygon” geometric object.

Now let’s display these GeoJSON objects!

**Asynchronous**

Slow down there. Before we can display a GeoJSON (next section), we need a way to get them, and upload them.

**Displaying GeoJSON objects – geoPath:**

D3 magic comes into play. D3 has a function called geoPath. Given a projection as input, the geoPath function returns a function that can display any GeoJSON object according to that original projection.

Let’s demonstrate with an example. First, we define the projection:

[projection defined]

Notice we used fitExtent. fitExtent assures our projection will choose x, y coordinates so the map is scaled to its best within the box, and will appear within the box.

With projection, we can render any GeoJSON using geoPath:

[ use code, call path path\_generator]

[Map]

As shown, path\_generator renders a GeoJSON by creating the code for the one attribute in a svg path object, “d”. To learn more about the svg path, go to [W3School](https://www.w3schools.com/graphics/svg_path.asp) or [Mozilla](https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths).

**Last things to note:**

First, path\_generator was able to render the GeoJSON by taking its coordinates, and converting them to a path element.

Second, asynchronous. When user first loads browser, the geojson is so big that it will not have loaded yet. Once it does load, we do not want the page to buffer. This method of loading data is called asynchronous data loading.

In d3, asynchronous loading happens by doing d3.json(json\_name).then(function( d) {

} ))

So by putting the previous code within the then statement, you are gucci.

Also awaken the server. If in python3, go to the directory of your code in the terminal, and type in *python3 –m http.server*. For more info, [link](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server).

Then access by typing in localhost:8000 in your browser.

Also Topojson possibly?

Very awesome!! We just created a dope map in D3.

(Talk where to find such a GeoJSON)

Where can find shapefiles for us states and counties - [link](https://www.census.gov/geo/maps-data/data/tiger.html)

**Include how to get geojson data, also need to awaken terminal**

**Building the NYTimes Election Map Part 3:**

DO ALL THESE BIG STEPS IN THREE STEPS.

SEPARATE JS into different code, so simpler

Start with getting Topojson as well – says samething as Geojson, but will enable us to use a function a little later in this tutorial.

Do a few things.

* Could use canvas, shadow for the text
* Step One: Three Buttons
* Step Two: Legend plus highway in the background
* Part 4: Three buttons: zoom in, out, and back to original
  + Do three clicks then stop
  + After click once, can move around
  + Colors of button change
* Part 5: Drag
* Shapefile of highways, roads overlayed. But those in thin black, county borders in white.
* Dope ass legend, stays constant
* **Extra (make map bigger or smaller based off window size)**

**SUGGEST CODEPEN**

* Can copy and paste parts of the code, and see how it works

work

.data() – bind to group of elements

.datum() – bind to individual element

to know d3 projections, need to know geojsons.

Lingering questions:

* How to create red rectangle inside the text box
* How make the bold appear the same for all paths when the mouse is highlighted over