**Summer 2016: Metis- Data Visualization Course**

**Class #1: Basics, Intro**

* **Example: First Chart! Axes/circles, no styling**
* **What the class covered:**
  + Margins
  + Axis scale
  + Axes
  + Data join
  + CSS for styling the axis
  + Transform/Translate
* **Learnings**
  + In terminal: write cd + name of folder where files are saved to change directory
  + To make a new folder type in: mkdir
  + Pwd: shows you where you are, LS: lists all the folders available to you
  + Write: Python –m simpleHTTPServer to establish a local server
  + Resource: <http://cli.learncodethehardway.org/book/>
  + Starter code for many d3 charts:
    - <https://github.com/thisismetis/nyc16_dataviz5/blob/master/class3/blueprint.md>

**Class #2: Ancombe’s Quartet**

* **Example: Red anscombe’s data with hover**
* **What the class covered:**
  + Scatter plots
  + Margins, positioning the chart on the page
  + Data joins
  + Bl.ocks
  + Styling
  + Hovering
* **What I learned:** 
  + Create groups to connect together all of the elements of an axis, or all of the circles associated with a specific set of data, enables you to manipulate them in an easier way
  + How to use the window’s console to identify problems, examine the HTML
  + Using d3 is a way to manipulate the DOM, by giving commands that change the HTML structure

**Class #3: Ancombe’s Quartet/Barley Data**

* **Example: no specific coded example**
* **What the class covered:**
  + Designing a chart visually
  + Accessing data from a file & data coercion
  + Ordinal Scales
  + Path objects are used for lines and all types of irregular shapes
  + Timescales (can use an ordinal scale for dates but doesn’t sort well)
  + Ready function
    - D3.tsv(“quartet.tsv”, ready)
    - Function ready(error, data){ All code related to data goes here}
* **What we learned:**
  + If you want the charts to appear side by side, you need to create divs, and style them in “display: inline-block”
  + NYT Trick to make data labels stand out is to write them twice (Once in white, bolded and slightly larger and once in black which is visible. Gives them a little bit of a halo effect
  + Best practices:
    - Use variable names that are easy to understand
    - Use the D3 API for reference
    - **Tidy data:** Make dataeasy to use and manipulate.
      * Each variable is a column, each observation is a row, and each type of observational unit is a table: This layout ensures that values of different variables from the same observation are always paired.
      * It is easier to describe the relationships between variables (columns) than rows
      * It is easier to make comparisons between groups of observations (rows) than between groups of columns
  + **Resources:**  Tidy Data article
    - Tool to show you how to clean your data: <https://github.com/hadley/tidy-data>

**Class #4: Barley Data**

* **Example: Slope chart with Barley Data**
* **What the Class covered:**
  + Pros/Cons of different types of charts
    - Scatter plot: Can see a lot of data visually
    - Bar charts: Easy to understand, sorts can help
    - Slope Charts: Good for comparing 2 years of data, shows overall trends
    - Fish eye distortion: as you hover over it, it scales up
  + The importance of labels and legibility of labels
  + Updating axes
  + Slope
* **What I learned:**
  + **For Each loop** within the data function is a great place to add new variables (aka manipulate current columns to add a new column)
    - Also used to turn things from string into integers:
      * d.x= + d.x: Make this x equivalent into the value of d.x that is an integer
  + **Short hand** for if/then statement: (Turnering statement) If else statement:
    - d.yieldGroup = d.yield >30 ? “large” : “small”).
    - If conditions is true, choose first value, else choose second value
  + **Filtering data**:
    - var sampleDataUF = data.filter(function(d) {return d.site == 'University Farm'; });
  + **Extent Function:** Returns the min/max of an array, useful for setting up the xScale/yScale domains
  + **Using Functions to draw charts**
    - Generalize things into functions so that you can do multiple charts at the same time
    - .sort need to tell it ascending or descending…looks a little different
    - Remember data labels! Explain to people what they are looking at!
  + **Sorting Data**
    - Sample code:
      * var sortedData= d3.nest()  
         .key(function(d){return d.variety})  
         .sortKeys(d3.descending)  
         .entries(sampleDataUF);
    - Another way to do it:
      * **dataset.sort (function(a,b) { return a.value <b.value;})**
      * **dataset.sort (function(a,b) { return a >b;})**
      * **nestedCountryData.sort(function(a, b) { return b.values.firstVal - a.values.firstVal});**
      * **dataset.sort(d3.ascending);**
  + **Nesting Data**
    - To get your data in the right format, use nest/use .key
    - Access data using d.key, d.values and d.values[0] if there are multiple layers
  + **Window.testingData**= data; allows you to access and see it in the console. Work with it in the console (data coersion)
    - $0.\_\_data\_\_: gives you a sense of the data that you are working with (that you have access to). good way to debug within the console
    - Concat(): Returns a new array comprised of the array on which it is called joined with the array(s) and/or value(s) provided as arguments
    - object.\_\_proto\_\_: can be used to set the prototype for an object. Not best practices to use it

**Class #5: Line Paths**

* **Example: Income Data, single line/single area chart**
* What the class covered:
  + Creating line charts: path generators: Need to use paths for anything more complicated than a straight line
  + Timescale
  + Understand your data before you start trying to visualize it
  + Nesting as a concept
* **What I learned:** 
  + **Interpolation:** Way for drawing lines, helps draw line between points. Use smoothing effects to help the line from being pointy.
    - **Different ways to interpolate:** .interpolate("basis").
    - Other options: cardinal, linear, basis, step, check out in spline editor under blocks
  + [Input forms](http://www.w3schools.com/html/html_form_input_types.asp): Used HTML to create buttons, drop down menus
  + **Timescales:**
    - Converts strings and integers into a date format. Then format it into the date format that you like. This helps with drawing lines connecting the dots in the correct order/
    - Steps: parse date, put it into time function, format timescale (%Y= year with 4 digits)
  + **Brush functionality:** Allows you to zoom in on dates, and allow you to find break points in your data
  + When no object exists in the HTML, add a new object using enter().append.

**Class #6: Double nest**

* **Examples: Incomes Data, multiple countries income on one page**
* What the class covered:
  + Nested data, how to access different layers
    - Layer one is used to make multiple SVGs, Layer 2 is used to make multiple lines, using functions
    - Use nesting for data that has subgroups (network charts, tree maps, subgroups, small multiples)
  + Functions to draw multiple lines
  + D3.set: Creates distinct values for an ordinal set of data
  + Dynamically generate class name
    - .attr(“class”, function(d){return “countryLines “ + d.key})
  + Can also create one big svg, and then mini g tags within the svg (assigned to the first level of the nest) rather than many different SVG tags (each assigned to the first level of the nest)
    - Create new scales that are for each of the smaller containers
    - Be explicit about where you are placing the different g tags: like an ordinal scale
* What I learned:
  + **Accessing different level of the nested data**
    - Great for making small multiples
    - First level: d.key, d.values-- using data(entries)
      * Use values or keys, don’t use both (they are a key/value pair)
    - Specific parts within the data: dataset[0].values
    - Next level down:
      * Data[0].values[0]
      * Data[0].values[0].values
      * Data[0].values[0].values.map(function(d){return d.year;})
      * Can name these specific levels of the nest as a variable to make it easier to access
        + Var swissValues = nestedCountryData.filter(function(d){ return d.key = “Switzerland”});
      * nestedCountryData.map(function(d){return d.key; })
        + This shows you all of the keys within the nested data
      * To create a set of data (used for ordinal data):
        + **d3.set(nestedCountryData.map(function(d){return d.key; })).values()** = will return the set (i.e. all for Switzerland get called “switzerland”)
        + Useful for ordinal groups
        + Useful for creating multiple different lines on one chart, each associated with one country but over different years
  + **Filtering within nested data**: nestedcountrydata.filter( return d.key = switzerland;)
  + **Dynamically generate class names for each group**
    - .attr(“class”, function(d){return “countryLines” + d.key})
  + Create a function to draw multiple lines
    - Svg. selectAll(“countryLines”)

.data(function(d){ return d.values})

.enter().append(“path”)

.attr(“class”, “countryLines”)

.attr(“d”, function (d) { return incomeLine(d.values): })

* + **Roll-up Function:** Helps to summarize documents. Eliminates the other values that aren’t rolled up, unless otherwise specified. Sample Code:
    - nestedCountryData = d3.nest()  
         .key(function(d) { return d.country })  
         .sortKeys(d3.descending)  
         .rollup(function(values) {  
           return {  
             totalVal: d3.sum(values, function(d) { return d.val; }),  
             minVal: d3.min(values, function(d) { return d.val; }),  
             maxVal: d3.max(values, function(d) { return d.val; }),  
             firstVal: values[values.length-1].val,  
             allValues: values}})  
         .entries(medianData)
  + xScale.domain and y.Scale.domain : needs to be inside of the function that accesses the data
  + .each(multiple)
  + **X.Rangeband()/2:** Used for a group bar chart, when you want the width of the bar to be half the width of the x axis category so that you can fit 2 bars where there used to be one
  + You need an initial state when using a function to generate multiple different types of charts, so that the chart knows what to draw at the beginning.
  + ReadMe should act as outline of how you want your code to look
  + Best practices for mobile: No hovers, smaller, simpler. Swipes are useful.

**Class #7: Triple nesting (games.tsv)**

* **Example:** Recreating the sports leagues/divisions chart
* What the class covered:
  + IDs are useful for general update patterns
* What I learned:

Class: 6.22.16

* Practice with nested data
* Anscombe Data (switching between groups)
* Practice with maps
* Practice with bar charts

Each div will use nested data, and so will only

$\_\_data\_\_ to see the data attached to it the thing in class

removing var

divisionScale returns a function

divisionScale.domain or .range will give you some sense of things

new scale: division.rangeBand()

Adds gridlines

.axis line {

stroke: #bfbfbf;

stroke-dasharray: 2,2;

group tags help to structure your page

divisionScale.rangeBand(): size the axis to the group.

Groups for each chart, NOT different SVGs, make it like an ordinal way

Each stroke is a different colors

, create a color scale, no range or domain

.style(“stroke”, function (d){return colorScale(d.key);})

style(“stroke-dasharray”, function(d,i){return 2\*i + “,” + 2\*i) or var dashArrayValue

if you have 700 things you cant do colors or shapes …used icons instead: look up noun project

Creating your own color scale:: <http://bl.ocks.org/mbostock/3884955>

Make a scale for the indexes: so that they are different for each thing

D3.set(data.map(function(d){ return d.team})).values

D3.range(returns a number in the range that you provide… Array.

Math.random (returns a random number)

Mand, disc, net interest

Revenue: income taxes etc : This is good for nested data because its

leagueDivs.select(‘.divisionGroups)

append classes as you go throughout

sort the vision groups

enter values! For labels!! D.key[length]

divisionGroups.selectAll(“.teamNames”)

.data(function9d){return d.values})

enter().append(“text”)

.attr(“class”, “teamNames”)

.text(function(d) {return d.key;})

attr(“x”, divisionScale.rangeBand())

.attr(“y”, function (d) {

var last value = d.values[d.values.length – 1]

return percentScale(lastValue.pct)

at the bottom of the nest, you can get all of the data back

You can do this with

**Class #8: General Updating**

* **What the class covered:**
  + General Update Pattern
    - Enter: Add new data points
    - Update: Update existing data points
    - Exit: Remove data points that are not still being used
  + Sliders
  + Responsiveness
  + Buttons
  + Updating axes
* **What I learned:** 
  + Stacked barchart
    - Need y0 and y1
    - Clippath makes sure that nothing goes off the chart in a sloppy way
  + Sliders
  + Responsiveness: Changing size based on screen size
  + Buttons
  + Drill down/drill up
  + Updating axes
  + Pointer events, allow you to do hovers and other things, pointer dash events , none (css style)
  + Finds an area that can fit as much as possible without interfering with another coordinate (Voronoi): Read up on this here: <https://bl.ocks.org/mbostock/8033015>
  + Hover over, add invisible rectangles or circles. It’s a path. Wrap data around voronoi function, like a line generator but more complicated
  + Map returns the numbers

**June 28, 2016**

* Pointer events, allow you to do hovers and other things, pointer dash events , none (css style)
* Finds an area that can fit as much as possible without interfering with another coordinate (Voronoi): Read up on this here: <https://bl.ocks.org/mbostock/8033015>
* Hover over, add invisible rectangles or circles. It’s a path. Wrap data around voronoi function, like a line generator but more complicated
* Map returns the numbers

Next class we'll be going through another fundamental, but difficult, part of D3, which is called the "General Update Pattern."

For that, read through these two blog posts:

* [Thinking with joins](https://bost.ocks.org/mike/join/)
* [Three little circles](https://bost.ocks.org/mike/circles/)
* <https://bost.ocks.org/mike/algorithms/>: visualizing how algorithms work
* d3:: 4.0, new version of d3 coming out!! Read this: <https://github.com/d3/d3/blob/master/CHANGES.md>

can load modules instead of all of the library (API directory).

These all take work to understand, but you can see Mike's explanations get better and better.

* [Three little circles](http://bost.ocks.org/mike/circles/)
* [How selections work](http://bost.ocks.org/mike/selection/)
* [selection.data()](https://github.com/mbostock/d3/wiki/Selections#data)
* [Thinking with joins](http://bost.ocks.org/mike/join/)
* [General Update Pattern](http://bl.ocks.org/mbostock/3808218)
* [General Update Pattern II](http://bl.ocks.org/mbostock/3808221)
* [General Update Pattern III](http://bl.ocks.org/mbostock/3808234)

1. Make a checklist for your
   1. Make new data join
   2. Get rid of old elements
   3. Enter new elements
   4. append elements as needed
   5. update new selection

Core idea of the update pattern

select it and data

exit method for when its not the same number of things connects

more data than elements : add enter

the same amount of data as before: update

if more elements than new

.data (data, function(d) { return d; ) the second part of this is the ID, so d.company (needs to be distinct: different for every single one). So that when you are updating.

SQL, what are you joining on to make sure you connect the data the right way

<https://esri.github.io/cedar/examples/drill-down.html>

if you have over 10,000 html objects on the page, it goes slow. Can use react

do your css : designer does its css page. Directory, separate page

functions are reusable chart functions, that can be called in other files

input type : range min and max with steps with a slider, give it an id

within update chart function::can’t stick it into a

d3.select(“#scaleRange”)

.on(“input”, function(){})

var this.value: slider value

slider:: using the slider!

**Class #9: Maps**

* **What the class covered:**
  + Maps
  + Color Scales
  + Bubble Charts
* **What I learned:**

Maps:

Really help you to find yourself in the data

Geographic variation

Doesn’t work for population, sometimes just captures the size of the population

What do you need to do with a map? (Chloropath)

1. Margin convention
2. SVG
3. Data bind
4. A path to draw a map
5. No need for axis or scale
6. Color scales
7. Legends
8. Data labels

Shape files: lines to show the shape of the data

Let's download some data from the Census. How about a [big shapefile](https://www.census.gov/cgi-bin/geo/shapefiles2010/main) of counties? Download and unzip the 2010 counties file tl\_2010\_us\_county10 to your local project folder.

Can be useful for lots of different maps

Map shaper helps to simplify the image, makes it smaller

Jason davies with different map projections, with different names

Search for shp files , map projections if looking for other things: Mercator

Q allows you to grab multiple datasets at the same time.

queue()

.defer(d3.csv, "guns-history.csv")--- this becomes the 1st variable in the ready function

.defer(d3.json, "us.json")—this becomes the 2nd variable in the ready function

.await(ready);

function ready(error, guns, us) {

if (error) return console.warn(error);

guns : refers to guns-history.csv. US refers to US.json

FIPS is the county number

Path take a set of coordinates…

Look at arcs: shows the FIPS number to help you reference it

Topojson library

Ok, data is referenced

Steps:

* Draw SVG
* Make a scale

Different libraries for different parts of the library: helps for

Used more in the new library

D3.geo.path: helps draw the map.

-helps show the maps, can specify different projections here with a .xxxx; (load it in

Look it up on the api reference to see how it works

Centroid: helps put it in the center of the country

Graticules help show lat and long

Line generator is var path

(topojson.feature(us, us.objects.counties).features);

could show land, states, countries, states etc, then find the features

if you want different maps:

var projection = d3.geo.mercator()

var path = d3. Geo.path():

.projection(projection)

Want the map to show maps and # of guns :

Key: county

Values: # guns

Dictionaries are called map objects

Map objects will not allow duplicates! An array

Create an object

To do this in nesting, creates the data in a different structure

Make yourself a little dictionary

guns.forEach(function(d){

gunsLookup[d.FIPS] = d.count3;

}) if you put it +d.count3, it only changes it within the object NOT within the original dataset

gunsLookup[4011]: and it shows it as it is ….

needs to be a STRING (Put

gunsLookup[40001].state : comes up with Oklahoma

do style down there to

exponential scale or threshold scale can help show gradients in a better way

power type of scale

Paul wrote a thing, which shows the different thresholds. A function that find s it dynamically

Colorbrewer helps to show best scales to use for maps

Every color brewer scale: bl.ocks. If you click on the color in the console it will pop up in the console

The scales can tell the different stories. Would be good to see the states.

Can also draw the states

If you are not going to interact with the states, you might just want the mesh function:  svg.append("path")

     .datum(topojson.mesh(us, us.objects.states, function(a, b) { return a.id !== b.id; }))

     // .enter().append("path")

     // .attr("class","states")

     .attr("d", path)

     .style("fill", "none")

     .style("stroke","#000")

Colors:

<http://colorbrewer2.org/>

<https://bl.ocks.org/mbostock/5577023>

using county data to look at the centroid : figure out the center value here

if there are specific

on top of the data

var radiusScale = d3.scale.linear()

.range([1,10]);

in the data: add radiusScale.domain(d3.extent(guns, function(d){return d.count3;}))

in the end, add in return radiusScale(gunsLookup(d.id)

.text(locationLoopup[d.id]) === This adds the text of the county

**Class #10:**

* What the class covered:
* What I learned:

Class 10 Notes: July 6

* d3 : 4.0 Map projections
* 4.0 color
* Lat and long: can be used across projections
* Can add to data shaper: other database
* Leaflet adds a googlemap layer to it, maybe a lot of stuff you don’t need
* Clone the repo, easier to access
* Longer datasets are smarter (Not wide).

Each map is drought areas, each is a new year.

* Nest by year

If you are having trouble with something, use a function to help to see how it looks like a slider, etc: Slider for scale to update the chart

id for type area

Can filter data OR also access it through the nest

.filter(function(d)

1 map to 60 maps in a tiny bit of extra code

values.d.id[0]

wont show you anything until its done

data.map: allows you to access the data by saying : nestedDroughtdata[2013]

use node to serve up img’s , scrolling window

react would be the fastest way to do this

limit the dataset to only a few

use a function that is updating it. Not redrawing the map function, only the drought info afterwards

class10v3: SVGs get bigger on hover

class10v4: Zoom in and out & Slider!! (Input based on threshold)

with a slider: class10v5

.on(“input”), function(){

var myValue = this.value

d3.selectAll(“counties”)

.style(“fill”, function(d){

return gunsLookup[d.id] > myValue

})

slider triggers the update, set timeout = javascript, this loops

on input: trigger a function

ZOOM:

On click, clicked. Clicked is a variable, k gets scaled

**Class #11:**

* What the class covered:
* What I learned:

July 11, 2016

Practice MAPS!!

Install homebrew, install nodes

Make heroku account, like aws, amazon web services: free!

Network Graphs

Source: 5, target of 2, only need one link

Bubbles = nodes

Links: line between things

D3: 4.0 with graphs: new functionality

Force directed graphs

Simulation figures out the placement on the svg

Arc:

Hive plots help to simplify network graphs: very difficult to do

Tree layouts: collapsible

Difference between these are the layout simulations, helps to figure out the x and y

Facebook of users, all friends with each other, multiple type of relationships

To do this: Checklist

* Data
* Node list/links list
  + 6 nodes, 7 links
  + explicit sources and targets for each value
  + simulation
  + position them
  + circles and lines to draw the elements
  + labels for each node
* array of nodes

graph nodes =

command shift d=

source and target

On tick is the beginning part of the simulation: starting point is a tick and then it goes into a specific direction

Steps:

1. Add in the data, all the nodes and

Would be useful to have the

Layout simulations, cord layouts: use simulations to automatically generate placement in places

Move nodes and links above the function

Error slip up: on a loop just repeats

D3: example file: d3.drag add to the nodes

Link distance

Link strength

Link gravity

Additional category as a year

Variety/site:

6 sites, 2 years, 10 variety

6 sites,

variety/sites: 60,

variety/site/year: 120

parent nodes

source target: source: site, target: variety/site

data.set for distinct set of ordinal

for loops : putting it into arrays

make a dictionary, with all the values that you can look up. With all the values, that could be really efficient to make it

Spend some time:

* Making dictionaries
* Traversing/accessing/understanding nesting data

**Examples for Bl.ocks:**

* ~~Anscombe’s Quartet: Scatter plot~~
* Barley data: Bar chart
* Income Data: Nesting
* Sports data: Double Nesting
* ~~Guns Data: Map~~
* Hovers
* Toggle

Terms:

* Color Scales
* Mouseover
* Mouseenter
* Mouseleave
* Update
* Enter
* Exit
* Transition
  + .ease
  + .delay
  + .during
* D3.scalebands
  + Rangebands
  + o.rangeBand: Can help tell you “ each group is this long” (useful for a bar chart. Range band allows you to have padding
  + Rangepoints: Axes in the middle
* display: inline-block, to have them side by side
* d3.nest
* Window.testingData= data; allows you to access and see it in the console. Work with it in the console
* Entries
* Data.Map
* ---proto---?
* Concat?
* To research
  + .on javascript parameteters

**Reading Notes:**

Bar Chart/Stacked with Labels: <http://bl.ocks.org/same1009/6cd391f67c055d4f0bf2b6d53dea01af>

Slope chart with slider/dynamic axes: <http://bl.ocks.org/pstuffa/5ac32bf4e3810a7ae61866ee5de02d41>

Useful Links:

<http://cli.learncodethehardway.org/book/>

<https://www.dashingd3js.com/svg-group-element-and-d3js>

[SVG](https://developer.mozilla.org/en-US/docs/Web/SVG):

* [g](https://developer.mozilla.org/en-US/docs/Web/SVG/Element/g)
* [text](https://developer.mozilla.org/en-US/docs/Web/SVG/Element/text)
* [circle](https://developer.mozilla.org/en-US/docs/Web/SVG/Element/circle)

[D3](https://github.com/mbostock/d3/wiki):

* [d3.tsv](https://github.com/mbostock/d3/wiki/CSV#tsv)
* [d3.svg.axis()](https://github.com/mbostock/d3/wiki/SVG-Axes)
* [d3.select()](https://github.com/mbostock/d3/wiki/Selections#d3_select)
* [d3.selectAll()](https://github.com/mbostock/d3/wiki/Selections#d3_selectAll)
* [d3.extent](https://github.com/mbostock/d3/wiki/Arrays#d3_extent)
* [selection.data()](https://github.com/mbostock/d3/wiki/Selections#data)
* [ordinal.rangeRoundBands()](https://github.com/mbostock/d3/wiki/Ordinal-Scales#ordinal_rangeRoundBands)

Plain JS

* [Array.map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)
* [Tidy data](http://www.jstatsoft.org/v59/i10/paper) By Hadley Wickham

Also, review [this code](https://bl.ocks.org/mbostock/3886208) and [this code](https://bl.ocks.org/mbostock/3887051) to the point where you generally understand what each block of code does.

data.map is used to return the data through a function (typically used for yScale.domain), as we have seen it.

When you are nesting data the better thing to do is to use “entries” because it returns key/value pairs

**Remaining Questions:**

* 1. If I get the “Cannot read property length of undefined” error message: what does this generally refer to?
  2. Why does order matter? Calling the axis needs to be within the data but earlier on in the data
  3. Color scale? How do you tie it to the data?
  4. Explain this snippet of code in a different way:
     + circleGroup.append("text").attr("class",function(d,i) { return "T" + i }) .text(function(d) { return "(" + d.x + "," + d.y + ")"; }) .attr("dx", 10);
     + var myClass = d3.select(circleGroup).attr("class").split(" ")[1];
  5. what is the benefit of using the ready function? Vs. the way that we have usually been doing it?
* **Adding fonts that are not automatically there?**
* **Axis cutting off slightly??**
* **Adding tick marks? Changing the style of the axis title?**
* **Adding axis labels?**
* **How do you style just one circle?**