Contents

D3 + Plot in JupyterLab Notebooks

Provided is a tutorial on running D3.js [BOH11] and the newly developed Plot from ObservableHQ [Obs20] .

Please note, this was designed to work in a **JupyterLab** environment.

This project started on 5/18/2021, so it is still in development. Feel free to provide suggestions.

[[BOH11]] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[[Obs20]] HQ Observable. Observable. 2020.

Notebook Introduction

Getting Started

From the Top!

Why am I here?

You are here to learn about D3.js, which is a data visualization library, and how to use D3.js in Jupyter. The skills learned in here can easily be translated outside of Jupyter, but this is the simplist way to get started with D3.js.

If you are interested in how to do this for observablehq.com, here is a link to that discussion: https://observablehq.com/@dudaspm/learning-d3-part-1?collection=@dudaspm/d3-in-observablehq

Who should be here?

This is meant to be an introduction to D3 (and a little bit of introduction to Jupyter). I am focused on helping people who have little to no programming experence. I may use a technical term here and there, but my goal is to create analogies whenever I can and include some technical terms to make searching for more details accessible.

Who am I?

I am teacher and D3 user for about 10 years, back when we used to call it Protovis. Even before that I was using SVG, XSLT, XPath, and XML to make interactive graphs (note, don't worry about looking these up, this type of development is REALLY old and outdated).

I am very passionate about helping people increase their visual literacy, or simply, using data to tell better visual stories.

Let's start

Our easel

File:Tripod easel.jpg. (2019, January 24). Wikimedia Commons, the free media repository. Retrieved 14:27, June 1, 2020 from https://commons.wikimedia.org/w/index.php?title=File:Tripod_easel.jpg&oldid=336282573.

When you want to create artwork, you need something to draw on. For this anology, I will be calling this the easel.

Our paint

File:Oil painting palette.jpg. (2017, September 27). Wikimedia Commons, the free media repository. Retrieved 14:30, June 1, 2020 from https://commons.wikimedia.org/w/index.php?title=File:Oil_painting_palette.jpg&oldid=260097528.

Now before you get our easel ready, we need to pick a type of paint or art medium (water colors, markers, finger paints). Well in this analogy we are choosing D3 as our art medium. So, let's start by declaring, in our notebook, that we will be using D3.

I DECLARE I WANT TO USE D3!

No, no...close... but we need to do this in a way that tells the notebook you want to use the D3.js.

(Required First Step) Enabling D3 in JupyterLab/Google Colab

```
from IPython.display import  HTML

def load_d3_in_cell_output():
  display(HTML("<script src='https://d3js.org/d3.v6.min.js'></script>"))
get_ipython().events.register('pre_run_cell', load_d3_in_cell_output)
```

The example below should output a blue circle. If so, everything is working properly! If not, please make sure to run the "Required First Step."

Both Colab and JupyterLab will be able to run the following example

```
%%html
<div id="example1"></div>

<script type="text/javascript">
    var width = 300
    var height = 100

    var svg = d3.select("div#example1").append("svg")
        .attr("width", width)
        .attr("height", height)

    var circle = svg.append("circle")
        .attr("cx", 150)
        .attr("cy", 50)
        .attr("r", 20)
        .style("fill", "blue")
        .style("stroke", "black")

</script>
```

Our canvas

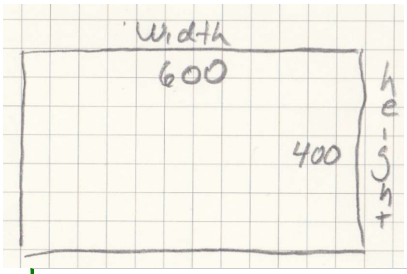
File:Splined Canvas.jpg. (2015, October 11). Wikimedia Commons, the free media repository. Retrieved 14:38, June 1, 2020 from https://commons.wikimedia.org/w/index.php?title=File:Splined_Canvas.jpg&oldid=175318221.

The next analogy we need to consider is the canvas for our art work. I need to be a bit careful here, if you have a bit of visualization background, especially online. You may have heard of a programming langauge called Canvas. This is not that Canvas. For the sake of simplicity, I will be using the lower case "canvas" to reference the supporting medium for our D3.js art.

Let's start talking about what is needed to create the canvas. We basically need to know two things:

- Height of the canvas
- Width of the canvas

In the image below, I am assuming the width of canvas will be 600 (600 pixels) and height will be 400 (400 pixels).




```
%%html
<div id="example2"></div>

<script type="text/javascript">
    var width = 600
    var height = 400

    var svg = d3.select("div#example2").append("svg")
        .attr("width", width)
        .attr("height", height)

</script>
```

Let's walk through this...

Variable types, as the name suggests, are the types of variables you can have in Javascript. For now, we will use one type: var or var(iable). This means the variable will available throughout our code.

Variables we will need:

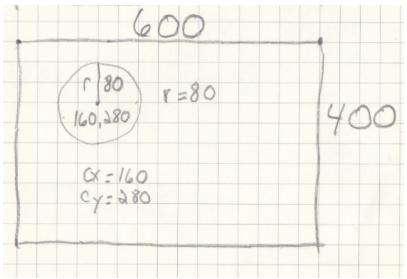
- Well the first two are pretty straight forward, we need our canvas to have a width and height. So...
  - var width = 600
    - or in human words... our variable called "width" will be 600.
  - var height = 400
    - or in human words... our variable called "height" will be 400.
- and this one is bit more difficult, but we need to our canvas to be these values
  - var svg = d3.select("div#example2").append("svg").attr("width", width).attr("height", height)
    - we're going to talk more about d3.select and d3.selectAll later, but for now you can assume that we are selecting the given cell (our easel) that we are in and adding our canvas of width (600) and height (400) to it`

TA DA! You did! You made your first "canvas" in the art of D3.js!

Our shapes

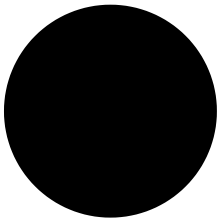
There are number of shapes we can add to our canvas, but for now we will focus on one of those shapes. Here is a listed of the available shape types.

Our first shape we will add is a circle. A circle requires 3 things: its x position (cx), its y position (cy), and a radius (r). Here is an example mock-up for a circle at position (160, 280) and radius 80.



To do this, we take our canvas and *append*, well, a circle. Considering a circle needs a cx, cy, and r, we will add those as attr(ibutes). Which gets us...

```
%%html
<div id="example3"></div>

<script type="text/javascript">
    var width = 600
    var height = 400

    var svg = d3.select("div#example3").append("svg")
        .attr("width", width)
        .attr("height", height)

    svg.append("circle")
        .attr("cx", 160)
        .attr("cy", 280)
        .attr("r", 80)
</script>
```



I want to point out something that may look confusing. Some may notice that circle appears to be lower than expected. That is because in HTML (the structure of webpages), the 0,0 point is in the top left (figure 2 below) and not the bottom left (figure 1 below). Just something to be mindful of moving forward.

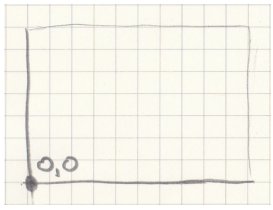


Figure 1 -

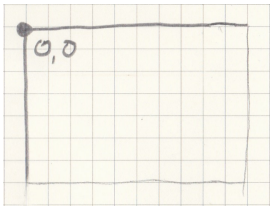
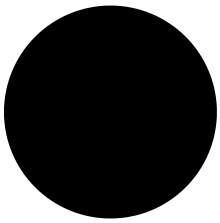


Figure 2 -

How most assume an X,Y coordinate would look. How it is done in HTML and thus, D3.js

To create the actual graph from above, we need to take advantage of the height variable to do so.

```
%%html
<div id="example4"></div>

<script type="text/javascript">
    var width = 600
    var height = 400

    var svg = d3.select("div#example4").append("svg")
        .attr("width", width)
        .attr("height", height)

    svg.append("circle")
        .attr("cx", 160)
        .attr("cy", height - 280)
        .attr("r", 80)

</script>
```



There we go!

Your turn: Create a new easel and canvas. On the canvas, add 3 circles to your canvas at different places and of different sizes.

```
%%html
<div id="example5"></div>

<script type="text/javascript">
    var width = 600
    var height = 400

    var svg = d3.select("div#example5").append("svg")
        .attr("width", width)
        .attr("height", height)

    # YOUR CODE HERE #

</script>
```

Your turn: Create a new easel and canvas. On the canvas, add a rectangle to your canvas at different places and of different sizes.

To help, here is an SVG Rectangle

```
<svg width="600" height="400">
  <rect x="10" y="10" width="30" height="30" />
</svg>
```

```
%%html
<div id="example6"></div>

<script type="text/javascript">
    var width = 600
    var height = 400

    var svg = d3.select("div#example6").append("svg")
        .attr("width", width)
        .attr("height", height)

    # YOUR CODE HERE #

</script>
```

Troubleshooting D3.js Issues

The Basics - Intro

Now with data!

Now that we know how to make a basic graph, we should focus on what makes D3 so special: its ability to connect data to visualizations. D3 actually stands for Data Driven Documents, meaning that data should inform the visual artifact or the document.

Introduction to Arrays

For this exercise, we will be using our HTML with JavaScript within the document. To do this, we will be using the script brackets.

```
<script>

</script>
```

Anything we put between these two brackets will be used as JavaScript. In the next cell, we are creating our first array. We need three things for this, its declariation, variable name, and type.

var - is our declaration

groceryList - is the name of the variable (or array)

[] - tells JavaScript to create an array or list.

```
%%html
<script>
    var groceryList = []
</script>
```

Note, what we did was create a variable or a named piece of code. Something that can vary (hence its name, variable). Variables must be one continous string of letters/numbers. So, instead of **grocery list**, I called it **groceryList**. Why the capitalization? because it makes it easier to separate out the words in the name. As in, **grocerylist** vs. **groceryList**. We have a fun name for this: camel case. 

File:CamelCase new.svg. (2020, April 15). Wikimedia Commons, the free media repository. Retrieved 15:25, June 3, 2020 from https://commons.wikimedia.org/w/index.php?title=File:CamelCase_new.svg&oldid=411544943.

Now to add to our array, we need to **push** some data into it. What, thats not how you usually put things in other things? By pushing it? OK, the naming is a bit weird, but we need to push data into our groceryList.

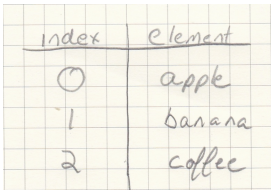
```
%%html
<p id="printout1"></p>
<script>
var groceryList = []
groceryList.push("apples")
groceryList.push("bananas")
groceryList.push("coffee")
document.getElementById("printout1").innerHTML = groceryList
</script>
```

1

Referencing parts of our array.

With our lists, we can access the data we put into them by using a numeric reference. Like saying, this is first on the list, second on the list, and so on. Unfortunately, this is where it gets a little weird. Where we expect to see that ordering would be 1, 2, 3, etc. it actually starts at 0. So, it's 0, 1, 2, etc...

So, to access let's say "banana". We would use position 1 or index 1 to access this element. Here is our array written out:



```
%%html
<p id="printout2"></p>
<script>
var groceryList = ["apples","bananas","coffee"]
document.getElementById("printout2").innerHTML = groceryList[1]
</script>
```

3

I want to access “apple”.

```
%%html
<p id="printout3"></p>
<script>
var groceryList = ["apples","bananas","coffee"]
document.getElementById("printout3").innerHTML = groceryList[0]
</script>
```

18

Another way to create an array is just to list out the items or elements in array.

Here is an example:

```
%%html
<p id="printout4"></p>
<script>
var listOfThings = ["item1", 42, 3.14, ["pb","j"]]
document.getElementById("printout4").innerHTML = listOfThings[0]
</script>
```

3,18

```
%%html
<p id="printout5"></p>
<script>
var listOfThings = ["item1", 42, 3.14, ["pb","j"]]
document.getElementById("printout5").innerHTML = listOfThings[2]
</script>
```

3.14

```
%%html
<p id="printout6"></p>
<script>
var listOfThings = ["item1", 42, 3.14, ["pb","j"]]
document.getElementById("printout6").innerHTML = listOfThings[3]
</script>
```

pb,j

Arrays to Artwork

For our array to artwork, we will be working with some data. For this I will be using some integer (or whole numbers).

```
drawingCircles = [3, 5, 5, 6, 15, 18]
```

OR

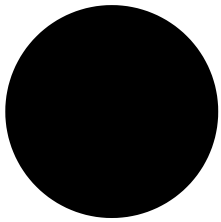
index	element
0	3
1	5
2	5
3	6
4	15
5	18

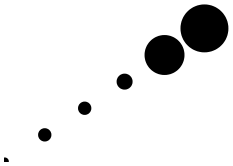
Let's copy and paste our code from our previous notebook for creating a circle

```
from IPython.display import  HTML

def load_d3_in_cell_output():
  display(HTML("<script src='https://d3js.org/d3.v6.min.js'></script>"))
get_ipython().events.register('pre_run_cell', load_d3_in_cell_output)
```

In Jupyter Lab

```
%%html
<div id="gohere1"></div>

<script type="text/javascript">
    var width = 600
    var height = 400

    var svg = d3.select("div#gohere1").append("svg")
        .attr("width", width)
        .attr("height", height)

    svg.append("circle")
        .attr("cx", 160)
        .attr("cy", 280)
        .attr("r", 80)
</script>
```





Now we need to add our data to our artwork, which will allow us to create 6 circles from our 6 data points.

There are 3 things we need to add to our code to connect the data to the shapes.

1. **.selectAll("circle")** - This is probably one of the most confusing parts of D3.js. I have spent many hours trying to figure how best to describe what is going on here. The basic concept is this; because we want to create circles based on our data, logically, we need to know how many circles do we originally have on our canvas and then compare this to our data. The best analogy I can come up with is imagine cooking with a friend and

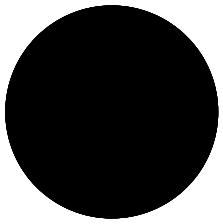
having that friend ask you to put 6 cloves of garlic into the bowl. A pretty standard question you might ask is "how many did you add already?" Because you don't want to add more than what is neccessary. This is what is going on here, you want to add 6 circles, so you need to ask, "how many do I have so far?"

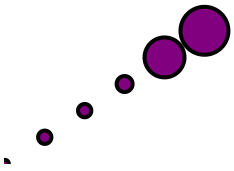
2. **.data(drawingCircles)** - This is where we add the data. The function, or the way we bring in the data, is called **.data()** and our data is called **drawingCircles**. **REALLY IMPORTANT** that the data you bring in *has to be* an array. Hence why we spent all that time talking and learning about arrays 😃
3. **.join("circle")** - This last step tells d3.js what you want to add to our canvas. In this case, we want to add a circle. In our next notebook, we will discuss the other types we can use in D3.js.

To simplify these steps, let's create a 3 piece set of actions:

1. Observe (the original scene) - **.selectAll()**
2. Collect (the data we need to update) - **.data()**
3. Update (the shape with this data) - **.join()**

```
%%html
<div id="gohere2"></div>

<script type="text/javascript">

    var drawingCircles = [3, 5, 5, 6, 15, 18]
    var width = 600
    var height = 400

    var svg = d3.select("div#gohere2").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Observe
    svg.selectAll("circle")
    // Collect
        .data(drawingCircles)
    // Update
        .join("circle")
        .attr("cx", 160)
        .attr("cy", 280)
        .attr("r", 80)

</script>
```





We did it! We made 6 circles! You might be wondering why you only see 1 circle here. Well, logically, if you draw 6 circles all on top of each other of the same radius. This is what you would see.

Let's try using the data to make the circles unique.

To do this, we need to change the way we use the **.attr()**. Specifically, use these in conjunction with our data to make unique circles based on our data. Here is an example:

- .attr("r", 80) becomes .attr("r", (d,i) => d) this is called arrow function expression or using an arrow to indicate a function or action taking place. You may notice that we are "shooting" two things – the letter *d* and *i*. The *d* references the d(ata) and the *i* the i(ndex). So, if we want to change the radius based on the data, we would use the d value and if you wanted to use the index, you would use the i.

Let's change the radius based on the d(ata) and the cx value based on the i(ndex).

Also, let's show this using two different methods.

1. using a tradition JavaScript function
2. using an arrow function

Traditional JavaScript function

```
%%html
<div id="traditional"></div>

<script type="text/javascript">

    var drawingCircles = [3, 5, 5, 6, 15, 18]
    var width = 600
    var height = 200

    var svg = d3.select("div#traditional").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Observe
    svg.selectAll("circle")
          // Collect
        .data(drawingCircles)
          // Update
        .join("circle")
        .attr("cx",
          function(d,i) {
            return i*20
          }
        )
        .attr("cy", 100)
        .attr("r",
          function(d,i) {
            return d
          }
        )

</script>
```

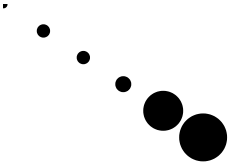
Arrow JavaScript function

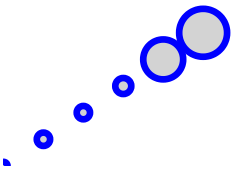
```
%%html
<div id="arrow"></div>

<script type="text/javascript">

    var drawingCircles = [3, 5, 5, 6, 15, 18]
    var width = 600
    var height = 400

    var svg = d3.select("div#arrow").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Observe
    svg.selectAll("circle")
          // Collect
        .data(drawingCircles)
         // Update
        .join("circle")
        .attr("cx", (d,i) => i*20)
        .attr("cy", 280)
        .attr("r", (d,i) => d)

</script>
```

As you can see, I used the i(ndex) and multiplied it by 20 to help create more spacing. This shows you the power of D3.js, the data is dictating the drawing (maybe they should use this for D3 acronym instead 😆).

Knowing this, let's making a drawing that looks like a plot of points with a positive slope (meaning the points are heading up as you move left to right.

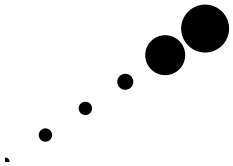
```
%%html
<div id="gohere3"></div>

<script type="text/javascript">

    var drawingCircles = [3, 5, 5, 6, 15, 18]
    var width = 600
    var height = 400

    var svg = d3.select("div#gohere3").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Observe
    svg.selectAll("circle")
    // Collect
        .data(drawingCircles)
    // Update
        .join("circle")
        .attr("cx", (d,i) => i*30)
        .attr("cy", (d,i) => i*20)
        .attr("r", (d,i) => d)

</script>
```





So, in this example, the points are heading down the hill. Remember from before, the (0,0) position is in the top-left, not the bottom-left. With this in mind, we can use the **height** to draw these.

```
%%html
<div id="gohere4"></div>

<script type="text/javascript">

    var drawingCircles = [3, 5, 5, 6, 15, 18]
    var width = 600
    var height = 400

    var svg = d3.select("div#gohere4").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Observe
    svg.selectAll("circle")
    // Collect
        .data(drawingCircles)
    // Update
        .join("circle")
        .attr("cx", (d,i) => i*30)
        .attr("cy", (d,i) => height - (i*20))
        .attr("r", (d,i) => d)

</script>
```



Your turn: Create a new array with random numbers in it (how many and the order does not matter). Then create circles with that array.

Styles, Scales, and Shapes

Styles and Shapes

Styling - The styling is handled through something call CSS (Cascading Style Sheets), but we can specifically handle this within our D3.js code. Here is a list of typical style changes.

- fill - the color inside the shape
- stroke - the border of the shape
- opacity - the transparency of the shape
- Note: there are more combinations, but these are the basics and covers a good amount of styling.

Shapes - At this point, we have been using circles primarily for out designs. There are few other shapes we need to cover and what is needed to draw these.

- Circles ("circle") - cx, cy, r
- Rectangles ("rect") - x, y, width, height
- Line ("line") - x1, y1, x2, y2
- Text ("text") - x, y
- Paths ("path") - these are by far the most complicate shapes, and will require further discussion throughout these notebooks
- Note: There are ellipse, polylines, and polygons, but these are RARELY used in D3.js .

First things, first... Let's bring back our last project for Part 2.

```
from IPython.display import  HTML

def load_d3_in_cell_output():
  display(HTML("<script src='https://d3js.org/d3.v6.min.js'></script>"))
get_ipython().events.register('pre_run_cell', load_d3_in_cell_output)
```

```
var dataset = [3, 5, 5, 6, 15, 18]
```

```
%%html
<div id="gohere1"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var dataset = [3, 5, 5, 6, 15, 18]

    var svg = d3.select("div#gohere1").append("svg")
        .attr("width", width)
        .attr("height", height)

  svg.selectAll("circle")
    .data(dataset)
    .join("circle")
    .attr("cx", (d,i)=> i*30)
    .attr("cy", (d,i)=> height - (i*20))
    .attr("r", (d,i)=> d)
</script>
```

In the above example, we used .attr(ibute) for the attributes of the given shape. We can use another function called .style() to add the CSS styling directly to the shape.

For our first example, we make these *purple circles with a black border that is 3 px in width*

```
%%html
<div id="gohere2"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var dataset = [3, 5, 5, 6, 15, 18]

    var svg = d3.select("div#gohere2").append("svg")
        .attr("width", width)
        .attr("height", height)

  svg.selectAll("circle")
    .data(dataset)
    .join("circle")
    .attr("cx", (d,i)=> i*30)
    .attr("cy", (d,i)=> height - (i*20))
    .attr("r", (d,i)=> d)
    .style("fill", "purple")
    .style("stroke", "black")
    .style("stroke-width", 3) // reminder, this means 3 pixels

</script>
```

Your Turn - Create the 6 circles with the stroke: blue, stroke-width: 5px, and fill: lightgrey

```
%%html
<div id="gohere3"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var dataset = [3, 5, 5, 6, 15, 18]

    var svg = d3.select("div#gohere3").append("svg")
        .attr("width", width)
        .attr("height", height)

  svg.selectAll("circle")
    .data(dataset)
    .join("circle")
    .attr("cx", (d,i)=> i*30)
    .attr("cy", (d,i)=> height - (i*20))
    .attr("r", (d,i)=> d)
    .style("fill", "lightgrey")
    .style("stroke", "blue")
    .style("stroke-width", 5) // reminder, this means 3 pixels
</script>
```

Again, the cool thing about D3.js is that we can use the data to style as well. Here is the same plot, but the stroke-width will be the value of our data set.

```
%%html
<div id="gohere4"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere4").append("svg")
        .attr("width", width)
        .attr("height", height)

    svg.selectAll("circle")
        .data(dataset)
        .join("circle")
        .attr("cx", (d,i)=> i*30)
        .attr("cy", (d,i)=> height - (i*20))
        .attr("r", (d,i)=> d)
        .style("fill", "purple")
        .style("stroke", "black")
        .style("stroke-width", (d,i) => i)
</script>
```

Color Scales

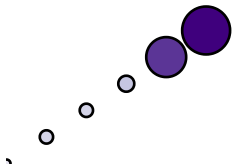
For the most part, this is not very helpful. One way to make this more useful is to use color based on our data, or in other words, the darker the color, the larger the value.

To implement this, I would HIGHLY suggest visit color-scales. As mentioned in the article, the color scales were created using Cynthia A. Brewer's ColorBrewer. ColorBrewer was designed to help designers find color-blind safe and print and copier safe color palette. ColorBrewer is my (and several other programming languages) "go-to" color palette.

For this, I will be choosing the Sequential (Single Hue) - Purple color. For these color palettes, they are expecting a value between 0 to 1, where 0 is the far-left of the color palette, the far-right is 1.

For now, to get our values between 0 and 1. I will take the largest value in our array (dataset, which is 18) and divide all of our values by this. Meaning, 3 will become 3/18, 5 will be 5/18 and so on.

```
%%html
<div id="gohere5"></div>

<script type="text/javascript">

    var width = 300
    var height = 300
    var dataset = [3, 5, 5, 6, 15, 18]
    var palette =  d3.interpolatePurples

    var svg = d3.select("div#gohere5").append("svg")
        .attr("width", width)
        .attr("height", height)

    svg.selectAll("circle")
        .data(dataset)
        .join("circle")
        .attr("cx", (d,i)=> i*30)
        .attr("cy", (d,i)=> height - (i*20))
        .attr("r", (d,i)=> d)
        .style("fill", (d,i) => palette(d/18))
        .style("stroke", "black")
        .style("stroke-width", 2)

</script>
```



Cool! We have some color! Though, we want to setup a way for our data to fit the 0 to 1 range withOUT needing us to manaully adding the largest value. There is a way to do this using scaling. Scaling allows use to create a range of values based on our data set. There are multiple types of scaling.For continuous data, or data with numeric values (Linear, Power, Log, Identity, Time, Radial). There are types of scaling, but for now we will focus on these.

Scale Linear (d3.scaleLinear)

Let's start by creating a scaling function called between0and1. To do this we will need two things:

- the **.domain()** which is the lowest and highest number that will be given to the function. This is usually the smallest number in our array or 3, and largest number, 18.
- the **.range()** is the range of values we want to map to, as in we want all numbers to be within 0 to 1.

Our smallest number (3), will be mapped 0 and our largest number (18) mapped to 1.

(No Audio) Figure showing how are list [5,3,16,5,6,18] maps to the domain and range [0,1]

```
%%html
<p id="printout1"></p>
<script>
var dataset = [5, 3, 16, 5, 6, 18]
var between0and1 = d3.scaleLinear().range([0,1]).domain([3,18])
document.getElementById("printout1").innerHTML = between0and1(18)
</script>
```

Again, though, we are still manually putting these values into our boundries. That's why there are built in functions to help AUTOMAGICALLY find the lowest and highest values in an array.

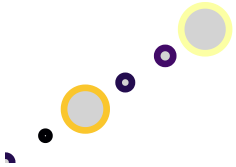
- **d3.max()** - finds the max value in the array
- **d3.min()** - finds the min value in the array
- **d3.extent()** - finds both the min and max values in an array

```
%%html
<p id="printout2"></p>
<script>
var dataset = [5, 3, 16, 5, 6, 18]
document.getElementById("printout2").innerHTML = d3.min(dataset)
</script>
```

```
%%html
<p id="printout3"></p>
<script>
var dataset = [5, 3, 16, 5, 6, 18]
document.getElementById("printout3").innerHTML = d3.max(dataset)
</script>
```

```
%%html
<p id="printout4"></p>
<script>
var dataset = [5, 3, 16, 5, 6, 18]
document.getElementById("printout4").innerHTML = d3.extent(dataset)
</script>
```

```
%%html
<div id="gohere6"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolateInferno
    var color = d3.scaleLinear().range([0,1]).domain(d3.extent(dataset))

    var svg = d3.select("div#gohere6").append("svg")
        .attr("width", width)
        .attr("height", height)

    svg.selectAll("circle")
        .data(dataset)
        .join("circle")
        .attr("cx", (d,i)=> i*30)
        .attr("cy", (d,i)=> height - (i*20))
        .attr("r", (d,i)=> d)
        .style("fill", "lightgrey" )
        .style("stroke", (d,i) => palette(color(d)) )
        .style("stroke-width", 5)
</script>
```



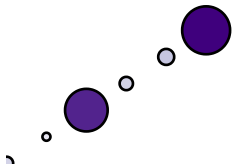
So, when to use d3.extent or d3.min/d3.max? This is a good example of this case. Right now,

```
const color = d3.scaleLinear().range([0,1]).domain(d3.extent(dataset))
```

assumes that our lowest number, 3, is mapped to 0. Though, in some cases, we want 0 to be mapped to 0. Meaning we should use

```
const color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])
```

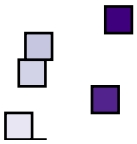
```
%%html
<div id="gohere7"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolatePurples
    var color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])

    var svg = d3.select("div#gohere7").append("svg")
        .attr("width", width)
        .attr("height", height)

    svg.selectAll("circle")
        .data(dataset)
        .join("circle")
        .attr("cx", (d,i)=> i*30)
        .attr("cy", (d,i)=> height - (i*20))
        .attr("r", (d,i)=> d)
        .style("fill", (d,i) => palette(color(d)) )
        .style("stroke", "black")
        .style("stroke-width", 2)

</script>
```



Shapes

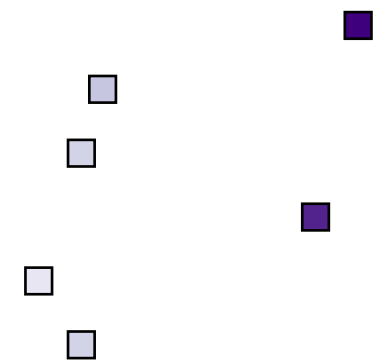
Rectangles

```
%%html
<div id="gohere8"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolatePurples
    var color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])

    var svg = d3.select("div#gohere8").append("svg")
        .attr("width", width)
        .attr("height", height)

    svg.selectAll("rect")
    .data(dataset)
    .join("rect")
    .attr("x", (d,i)=> d*5)
    .attr("y", (d,i)=> height - (i*20))
    .attr("width", 20)
    .attr("height", 20)
    .style("fill", (d,i) => palette(color(d)) )
    .style("stroke", "black")
    .style("stroke-width", 2)
    .style("stroke-width", 2)
</script>
```



For rectangles, the x,y is the origin of the rectangle (again in the top-lefthand corner). Right now, we are missing a rectangle, well, not missing it, it just off the canvas. For data point 3, index 0, the x position is 15, and the y position is the height. Meaning, we need to correct this. Also, we are not using the space very well. This was true with our circles, but let's see if we can fix this issue here as well. The best way to do this is to create margins. For now, let's just set a margin of 30. 30 on the top, bottom, left, and right. We will do this using the scaleLinear function for both the x and y axis.

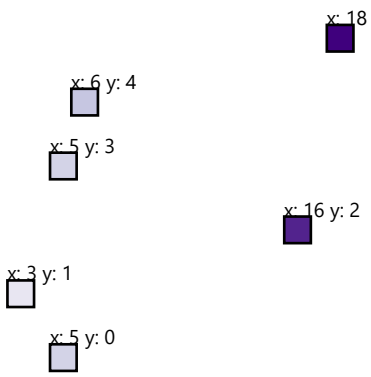
```
%%html
<div id="gohere9"></div>

<script type="text/javascript">

    var width = 300
    var height = 300
    var margin = 30 // Add my margin
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolatePurples
    var color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])

    var svg = d3.select("div#gohere9").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Use the margin to create an x domain and range
    var x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset))

    // Use the margin to create an y domain and range
    var y = d3.scaleLinear().range([height-margin,margin]).domain([0,dataset.length-1])

    svg.selectAll("rect")
        .data(dataset)
        .join("rect")
        .attr("x", (d,i)=> x(d))
        .attr("y", (d,i)=> y(i))
        .attr("width", 20)
        .attr("height", 20)
        .style("fill", (d,i) => palette(color(d)) )
        .style("stroke", "black")
        .style("stroke-width", 2)

</script>
```



Text

Next, we add some text next to our boxes. For the most part, we will be using the same code as our rectangles. Let's take a look.

```
%%html
<div id="gohere10"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var margin = 30 // Add my margin
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolatePurples
    var color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])

    var svg = d3.select("div#gohere10").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Use the margin to create an x domain and range
    var x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset))

    // Use the margin to create an y domain and range
    var y = d3.scaleLinear().range([height-margin,margin]).domain([0,dataset.length-1])

    svg.selectAll("rect")
        .data(dataset)
        .join("rect")
        .attr("x", (d,i)=> x(d))
        .attr("y", (d,i)=> y(i))
        .attr("width", 20)
        .attr("height", 20)
        .style("fill", (d,i) => palette(color(d)) )
        .style("stroke", "black")
        .style("stroke-width", 2)

    // adding in the text
    svg.selectAll("text")
        .data(dataset)
        .join("text")
        .attr("x", (d,i)=> x(d))
        .attr("y", (d,i)=> y(i))
        .text((d,i) => "x: "+d+" y: "+i)
</script>
```



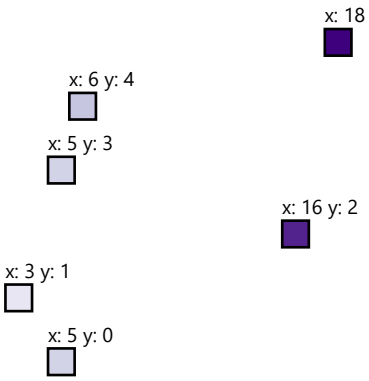
Adding in the text, the only additional piece we need to add is what the .text() will be. In this case, I am again using the data to add specific data related text to the screen. If we take a particular look at this function, we can see how we used both text and data together.

```
.text((d,i) => "x: "+d+" y: "+i)
```

Using the "x: ", the plus sign (+), and d will combine or concatenate the two to make one string

The last thing that needs to be adjusted is the fact that both the rectangle and the text occupy the same x,y coordinate, which means they overlap a bit. Also, the text for our last rectangle is off the canvas. So, let's adjust both of these.

```
%%html
<div id="gohere11"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var margin = 30 // Add my margin
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolatePurples
    var color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])

    var svg = d3.select("div#gohere11").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Use the margin to create an x domain and range
    var x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset))

    // Use the margin to create an y domain and range
    var y = d3.scaleLinear().range([height-margin,margin]).domain([0,dataset.length-1])

    svg.selectAll("rect")
        .data(dataset)
        .join("rect")
        .attr("x", (d,i)=> x(d))
        .attr("y", (d,i)=> y(i))
        .attr("width", 20)
        .attr("height", 20)
        .style("fill", (d,i) => palette(color(d)) )
        .style("stroke", "black")
        .style("stroke-width", 2)

    // adding in the text
    svg.selectAll("text")
        .data(dataset)
        .join("text")
        .attr("x", (d,i)=> x(d))
        // moving our text up a bit (subtracting 5 pixels)
        .attr("y", (d,i)=> y(i)-5)
        .text((d,i) => "x: "+d+" y: "+i)
</script>
```



Lines

Next, we have lines. Lines are similar to circles, rectangles, and text. You need a starting x,y, and similar to the rectangle you need secondary dimension (width/height), whereas lines need an ending x,y position. I think there can be a misconception about lines, based on “line graphs.” Line graphs (as seen below) look like a single line but fluctuations here and there when a better way to think about lines independent of one another. Paths (talked about next) are where we will be able to think about one, continuous line.

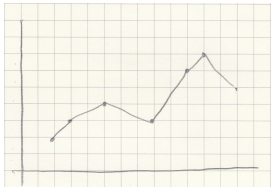


Figure 1 - How you might assume the line shape would work.

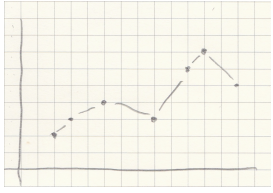
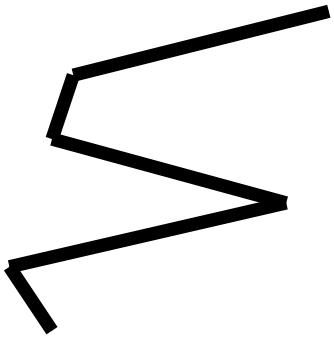


Figure 2 - How the line shape actually works.

```
%%html
<div id="gohere12"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var margin = 30 // Add my margin
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolatePurples
    var color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])

    var svg = d3.select("div#gohere12").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Use the margin to create an x domain and range
    var x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset))

    // Use the margin to create an y domain and range
    var y = d3.scaleLinear().range([height-margin,margin]).domain([0,dataset.length-1])

    // update this to select the lines, then join (or add) lines.
    svg.selectAll("line")
        .data(dataset)
        .join("line")
        // lines require x1,y1 (where the line starts) and x2,y2 (where the lines end)
        .attr("x1", (d,i)=> x(d))
        .attr("y1", (d,i)=> y(i))
        .attr("x2", (d,i)=> x(d)+10)
        .attr("y2", (d,i)=> y(i)+10)
        // to change the color of the line we need to update the stroke, not the fill.
        .style("stroke", (d,i) => palette(color(d)) )
        .style("stroke-width", 10)

</script>
```

To create a line chart from these lines would require several changes. Here is an example of how to do it, but there is a MUCH simpler way using paths, which we will talk about in a second.

```
%%html
<div id="gohere13"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var margin = 30 // Add my margin
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolatePurples
    var color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])

    var svg = d3.select("div#gohere13").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Use the margin to create an x domain and range
    var x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset))

    // Use the margin to create an y domain and range
    var y = d3.scaleLinear().range([height-margin,margin]).domain([0,dataset.length-1])

    // update this to select the lines, then join (or add) lines.
    svg.selectAll("line")
        .data(dataset.slice(0, -1))
        .join("line")
        .attr("x1", (d,i)=> x(d))
        .attr("y1", (d,i)=> y(i))
        // we then need to end our line with the next data point.
        // that's why we use (i+1) and why we cutoff the last position in our array
        .attr("x2", (d,i)=> x(dataset[i+1]))
        .attr("y2", (d,i)=> y(i+1))
        .style("stroke", "black" ) // changing this back, as not all the data points (and
colors) will be represented
        .style("stroke-width", 10)
</script>
```

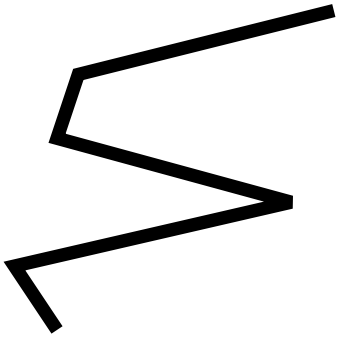


As you can see, this is not the most elegant way to handle creating a line.

Paths

Finally, we have paths. Up to this point, we have been focusing on the idea “for each data point, we create a shape or object.” Paths require multiple data points to create the shape/object. We can use a function to create our line. We can either do this in a new (separate) cell or in with the code itself. Let’s build it in our design.

```
%%html
<div id="gohere14"></div>

<script type="text/javascript">
    var width = 300
    var height = 300
    var margin = 30 // Add my margin
    var dataset = [5, 3, 16, 5, 6, 18]
    var palette =  d3.interpolatePurples
    var color = d3.scaleLinear().range([0,1]).domain([0,d3.max(dataset)])

    var svg = d3.select("div#gohere14").append("svg")
        .attr("width", width)
        .attr("height", height)

    // Use the margin to create an x domain and range
    var x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset))

    // Use the margin to create an y domain and range
    var y = d3.scaleLinear().range([height-margin,margin]).domain([0,dataset.length-1])

    // this is our function to create the line, where the x is based on the d(ata) and the y on
the i(ndex)
    var line = d3.line()
        .x((d,i)=> x(d))
        .y((d,i)=> y(i))

    svg.selectAll("path")
        .data([dataset])
        .join("path")
        .attr("d", line)
        .style("stroke", "black" )
        .style("stroke-width", 10)
        .style("fill", "none") // SPECIAL NOTE HERE: we set the fill to "none" because a path
will have a fill color

</script>
```



Accessing Data

In this notebook, we discuss how to access datasets within D3.js. There are two ways to accomplish this in our Jupyter Lab environment.

d3-fetch

The most straightforward way is to use d3-fetch.

JavaScript

```
d3.csv("/path/to/file.csv").then((data) => { })
```

Python (Pandas) + d3-fetch

The second way include d3-fetch, but processing the data first in python, saving the file, and then using d3.fetch. Typically, I will use a library called Pandas to do this, as it similar features to d3.js.

Python

```
import pandas as pd
data=pd.read_csv("/path/to/file.csv")
data.to_csv('newFile.csv', index = False, header=True)
```

JavaScript

```
d3.csv("/path/to/newFile.csv").then((data) => { })
```

Pros/Cons of Both

There is little difference between the two, obviously you are adding another step with Python, but you get the added benefit of being able to use all of Python's tools before working with the file. Not to say D3.js does not have these similar capabilities. If anything, I (my preference) think D3.js are a bit better and easier to use, but this not the case for everyone. Let's showcase both.

Start D3.jsfrom IPython.display import HTML

```
from IPython.display import  HTML

def load_d3_in_cell_output():
  display(HTML("<script src='https://d3js.org/d3.v6.min.js'></script>"))
get_ipython().events.register('pre_run_cell', load_d3_in_cell_output)
```

Acknowledgement for Data

Cite as: Menne, Matthew J., Imke Durre, Bryant Korzeniewski, Shelley McNeal, Kristy Thomas, Xungang Yin, Steven Anthony, Ron Ray, Russell S. Vose, Byron E.Gleason, and Tamara G. Houston (2012): Global Historical Climatology Network - Daily (GHCN-Daily), Version 3. CITY:US420020. NOAA National Climatic Data Center. doi:10.7289/V5D21VHZ 02/22/2021.

Publications citing this dataset should also cite the following article: Matthew J. Menne, Imke Durre, Russell S. Vose, Byron E. Gleason, and Tamara G. Houston, 2012: An Overview of the Global Historical Climatology Network-Daily Database. J. Atmos. Oceanic Technol., 29, 897-910. doi:10.1175/JTECH-D-11-00103.1.

Use liability: NOAA and NCEI cannot provide any warranty as to the accuracy, reliability, or completeness of furnished data. Users assume responsibility to determine the usability of these data. The user is responsible for the results of any application of this data for other than its intended purpose.

Links: https://data.noaa.gov/onestop/

https://www.ncdc.noaa.gov/cdo-web/search

d3-fetch Implementation

The data we will be using for this exercise is acknowledged above, but the basic idea is this is a weather data for the State College, PA area (near Penn State University).

The data preview is:

- DATE - date in YYYY-MM-DD format
- DAY - the day (_D format)
- MONTH - the month (_M format)
- YEAR - the year (YYYY format)
- PRCP - the amount of precipitation in inches
- SNOW - the amount of snow in inches
- TMAX - the max temperature in F°
- TMIN - the min temperature in F°

Here's a print out of the first row of data:

```
%%html
<p id="print1"></p>
<script>

d3.csv("https://gist.githubusercontent.com/dudaspm/13174849c09aba7a0716d5fa230ebe95/raw/a4866834185bad7e4a1e1b8f90da76b168eb1361/StateCollege2010-2020_min.csv")
    .then((data) =>
        document.getElementById("print1").innerHTML = JSON.stringify(data[0])
    )
</script>
```

{"DATE":"2000-01-01","DAY":"1","MONTH":"1","YEAR":"2000","PRCP":"0","SNOW":"0","TMAX":"44","TMIN":"23"}

One thing you can add to this (🕸🕸AND I STRONGLY RECOMMEND DOING THIS🕸🕸) is adding

```
.catch((error) => console.log(error) )
```

Why? because Jupyter does a not so great job of showcasing errors. As much we try and teach ourselves to be perfect coders 😵. We all make mistakes and too be honest, as much as I love D3.js. One small thing can cause problems. If you have not looked at the troubleshooting section of the notebook, please check it out. It will save you hours of your life in small errors.

```
%%html
<p id="print2"></p>
<script>

d3.csv("https://gist.githubusercontent.com/dudaspm/13174849c09aba7a0716d5fa230ebe95/raw/a4866834185bad7e4a1e1b8f90da76b168eb1361/StateCollege2010-2020_min.csv")
    .then((data) =>
        document.getElementById("print2").innerHTML = JSON.stringify(data[0])
    )
    .catch((error) => console.log(error) )
</script>
```

{"DATE":"2000-01-01","DAY":"1","MONTH":"1","YEAR":"2000","PRCP":"0","SNOW":"0","TMAX":"44","TMIN":"23"}

Python + Pandas + d3-fetch Implementation

Now this implementation adds an extra step but again, if you want to do more in Python and make the data available to D3.js afterwards. This is where you want to be!

Let's do the same steps as above, but this time we will add Pandas.

> **Note**
>
> For those using this notebook in Google Colabs, Pandas is already installed. But, for those running this locally or on another system. You may need to install pandas first. You can actually do this in the notebook. Create a new cell (after this one) and run the following:
>
> *!pip install pandas*
>
> Here, I'll help with this. The next cell will have this, but it will commented out. Just remove the #.

```
#!pip install pandas
```

We need to break this up into three steps. This will allow Jupyter Lab to keep this variable within the notebook and use it internally (instead of writing to file).

Also, and most importantly, this works in Google Colab.

Using Python + Pandas

Suprisingly, this is much harder than it needs to be. Specifically because I am trying to design this for Jupyter Lab and (the culprit 😈) Google Colab. Google Colab is a fantastic implementation of Jupyter Lab but it has to be locked down a bit more because well, people could do some really bad things if they didn't. So, after months (and I mean months) of searching and trying things, this is my best implementation.

Side note, if by chance you figure out a better solution, please let me know! It would make my day 😁

Starting Pandas and Getting the File

```
import pandas as pd
data=pd.read_csv("https://gist.githubusercontent.com/dudaspm/13174849c09aba7a0716d5fa230ebe95/raw/a4866834185bad7e4a1e1b8f90da76b168eb1361/StateCollege2010-2020_min.csv")
```

```
%%html
<p id="print3"></p>
<script>
var receiver = new BroadcastChannel('channel');
receiver.onmessage = (msg) => {
    var data = d3.csvParse(msg.data.replaceAll(";;;","\n"))
    document.getElementById("print3").innerHTML = JSON.stringify(data[0])
};
</script>
```

```
from IPython.display import display, HTML
d = data.head().to_csv(index=False,line_terminator='\n').replace('\n',';;;')[:-3]
js = """
<script>
var sender = new BroadcastChannel('channel');
var message = '{}'
senderChannel.postMessage(message);
</script>""".format(d)


display(HTML(js))
```

Scaling, Legends, and Color

```
from IPython.display import HTML, Javascript, display

def configure_d3():
    display(Javascript("""
    require.config({
      paths: {
        d3: "https://d3js.org/d3.v6.min"
      }
    })"""))


configure_d3()
```

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations != ""))
            const dateConverter = d3.timeParse("%Y-%m-%d")
            data = data.map(d=> ({date:dateConverter(d.date),vaccinated:+d.people_vaccinated,location:d.location}))
            console.log(data)
        })
        .catch(function(error){
        //
        })

})
</script>
```

Acknowledgement

Max Roser, Hannah Ritchie, Esteban Ortiz-Ospina and Joe Hasell (2020) - "Coronavirus Pandemic (COVID-19)". Published online at OurWorldInData.org. Retrieved from: 'https://ourworldindata.org/coronavirus' [Online Resource]

Original Link: https://github.com/owid/covid-19-data (Accessed 3/11/2021) Source Link: https://github.com/owid/covid-19-data/tree/master/public/data/vaccinations

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12), 2301-2309.

All Data

```
%%html
<div id="gohere1"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 400
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dateConverter = d3.timeParse("%Y-%m-%d")
            data = data.map(d=>
({date:dateConverter(d.date),vaccinated:+d.people_vaccinated,location:d.location}))

            const xScale = d3.scaleTime().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => d.date))
            const yScale = d3.scaleLinear().range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.vaccinated))

            const line = d3.line()
                .x((d,i)=> xScale(d.date))
                .y((d,i)=> yScale(d.vaccinated))


            const svg = d3.select("div#gohere1").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale).tickFormat(d3.timeFormat("%m-%d"))

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Date")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(d.date))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d.location+"\nVaccinated: "+d.vaccinated)



        })
        .catch(function(error){

        })

})
</script>
```

Sorting: https://observablehq.com/@d3/d3-ascending

```
%%html
<div id="gohere2"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 400
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d") // creates the date object
            const dataFormat = d3.timeFormat("%Y-%m-%d") // creates a string from the date
object
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))
            console.log(data.sort((a, b) => b.date - a.date)[0])
            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )
            console.log(data)

            // Sorting by vaccinated and using a linear scale
            data = data.sort((a, b) => b.vaccinated - a.vaccinated)
            const xScale = d3.scaleLinear().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => i))
            const yScale = d3.scaleLinear().range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.vaccinated))


            const svg = d3.select("div#gohere2").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Index ordered by vaccinated")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(i))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d.location+" Vaccinated: "+d.vaccinated)



        })
        .catch(function(error){

        })

})
</script>
```

Log

```
%%html
<div id="gohere3"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 500
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d")
            const dataFormat = d3.timeFormat("%Y-%m-%d")
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))
            console.log(data.sort((a, b) => b.date - a.date)[0])
            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )
            console.log(data)

            // Sorting by vaccinated and using a linear scale
            data = data.sort((a, b) => b.vaccinated - a.vaccinated)
            data = data.filter(d=> d.vaccinated != 0 )
            const xScale = d3.scaleLinear().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => i))
            const yScale = d3.scaleLog().range([height-margin , margin]).domain(d3.extent(data,
(d,i) => d.vaccinated))


            const svg = d3.select("div#gohere3").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Index ordered by vaccinated")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(i))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d.location+" Vaccinated: "+d.vaccinated)



        })
        .catch(function(error){

        })

})
</script>
```

Power Scales -

mx^k + b, where k is the exponent value

https://github.com/d3/d3-scale#scalePow

```
%%html
<div id="gohere4"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 500
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d")
            const dataFormat = d3.timeFormat("%Y-%m-%d")
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))
            console.log(data.sort((a, b) => b.date - a.date)[0])
            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )
            console.log(data)

            // Sorting by vaccinated and using a linear scale
            data = data.sort((a, b) => b.vaccinated - a.vaccinated)

            const xScale = d3.scaleLinear().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => i))
            const yScale = d3.scalePow().exponent(.5).range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.vaccinated))


            const svg = d3.select("div#gohere4").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Index ordered by vaccinated")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(i))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d.location+" Vaccinated: "+d.vaccinated)



        })
        .catch(function(error){

        })

})
</script>
```

k=.25

```
%%html
<div id="gohere5"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 500
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d")
            const dataFormat = d3.timeFormat("%Y-%m-%d")
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))
            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )

            // Sorting by vaccinated and using a linear scale
            data = data.sort((a, b) => b.vaccinated - a.vaccinated)

            const xScale = d3.scaleLinear().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => i))
            const yScale = d3.scalePow().exponent(.25).range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.vaccinated))


            const svg = d3.select("div#gohere5").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Index ordered by vaccinated")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(i))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d.location+" Vaccinated: "+d.vaccinated)



        })
        .catch(function(error){

        })

})
</script>
```

k=.75

```
%%html
<div id="gohere6"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 500
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d")
            const dataFormat = d3.timeFormat("%Y-%m-%d")
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))
            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )

            // Sorting by vaccinated and using a linear scale
            data = data.sort((a, b) => b.vaccinated - a.vaccinated)

            const xScale = d3.scaleLinear().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => i))
            const yScale = d3.scalePow().exponent(1).range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.vaccinated))


            const svg = d3.select("div#gohere6").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Index ordered by vaccinated")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(i))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d.location+" Vaccinated: "+d.vaccinated)



        })
        .catch(function(error){

        })

})
</script>
```

Discrete Bins

There is a really nice tutorial about this located here: https://observablehq.com/@d3/quantile-quantize-and-threshold-scales?collection=@d3/d3-scale

```
%%html
<div id="gohere6b"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 500
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d")
            const dataFormat = d3.timeFormat("%Y-%m-%d")
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))

            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )

            // Sorting by vaccinated and using a linear scale
            data = data.sort((a, b) => b.vaccinated - a.vaccinated)

            const xScale = d3.scaleLinear().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => i))

            const cut1 = 300000
            const cut2 = 3000000
            const yScale = d3.scaleThreshold().range([height-margin, (height-margin)/2,
margin]).domain([cut1, cut2])

            const svg = d3.select("div#gohere6b").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Index ordered by vaccinated")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(i))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d.location+" Vaccinated: "+d.vaccinated)



        })
        .catch(function(error){

        })

})
</script>
```

Quantize

```
%%html
<div id="gohere7"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 500
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d")
            const dataFormat = d3.timeFormat("%Y-%m-%d")
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))

            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )

            // Sorting by vaccinated and using a linear scale
            data = data.sort((a, b) => b.vaccinated - a.vaccinated)

            const xScale = d3.scaleLinear().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => i))

            const q1 = d3.quantile(data.map(d=> d.vaccinated).sort(d3.ascending),.33)
            const q2 = d3.quantile(data.map(d=> d.vaccinated).sort(d3.ascending),.66)
            console.log(q1,q2,d3.max(data,d=>d.vaccinated))
            const yScale = d3.scaleQuantize().range([height-margin, (height-margin)/2,
margin]).domain(d3.extent(data,d=>d.vaccinated))

            const svg = d3.select("div#gohere7").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Index ordered by vaccinated")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(i))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d.location+" Vaccinated: "+d.vaccinated)



        })
        .catch(function(error){

        })

})
</script>
```

Quantile

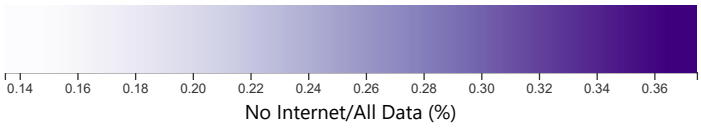
```
%%html
<div id="gohere8"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 700
            const height = 500
            const margin = 60
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d")
            const dataFormat = d3.timeFormat("%Y-%m-%d")
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))

            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )

            // Sorting by vaccinated and using a linear scale
            data = data.sort((a, b) => b.vaccinated - a.vaccinated)

            const xScale = d3.scaleLinear().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => i))

            const yScale = d3.scaleQuantile().range([height-margin, (height-margin)/2,
margin]).domain(data.map(d=>d.vaccinated))

            const svg = d3.select("div#gohere8").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Index ordered by vaccinated")

            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d/100000 )

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("People Vaccinated (x 100000)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(i))
                .attr("cy", (d,i)=> yScale(d.vaccinated))
                .attr("r", 5)
                .append("title")
                .text(d=> "Location: "+d[0]
                      .location+" Vaccinated: "+d.vaccinated)



        })
        .catch(function(error){
            console.log(error)
        })

})
</script>
```

Legends

D3.js Colors - https://github.com/d3/d3-scale-chromatic

```
%%html
<div id="legend1"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            const width = 800
            const height = 300
            const margin = 40
            data = data.filter(d=> (d.location != "United States") && (d.total_vaccinations !=
""))
            const dataParse = d3.timeParse("%Y-%m-%d") // creates the date object
            const dataFormat = d3.timeFormat("%Y-%m-%d") // creates a string from the date
object
            data = data.map(d=>
({date:dataParse(d.date),vaccinated:+d.people_vaccinated,location:d.location}))

            const lastDate = data.sort((a, b) => b.date - a.date)[0].date
            data = data.filter(d=> dataFormat(d.date) == dataFormat(lastDate) )
            data = data.filter(d=> d.vaccinated != 0 )

            data = data.sort((a, b) => b.vaccinated - a.vaccinated)
            const palette = d3.interpolateTurbo
            const scaling = d3.scaleLog().range([margin,width-margin]).domain(d3.extent(data,
(d,i) => d.vaccinated))
            const x = d3.scaleLinear().range([width-margin,margin]).domain(d3.extent(data,
(d,i) => i))
            const coloringCircles = d3.scaleLog().range([0,1]).domain(d3.extent(data, (d,i) =>
d.vaccinated))


            const svg = d3.select("div#legend1").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(scaling).ticks(15, "~s")

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Vaccinated (Log Scale)")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> x(i))
                .attr("cy", (d,i)=> height/2)
                .attr("r", 5)
                .style("fill",d=>palette(coloringCircles(d.vaccinated)))
                .style("stroke","black")
                .style("stroke-width",1)
                .append("title")
                .text(d=> "Location: "+d.location+" Vaccinated: "+d.vaccinated)

            const num = 20
            const values = d3.range(1,num)

            const coloring = d3.scaleLinear().range([0,1]).domain(d3.extent(values))
            var defs = svg.append("defs")
            var linearGradient = defs.append("linearGradient")
                .attr("id", "linear-gradient1")

            linearGradient.selectAll("stop").data(values).join("stop")
                .attr("offset", d=> d/num)
                .attr("stop-color", d=>palette(coloring(d)) )
            svg.append("rect")
                .attr("x", margin)
                .attr("y", (height-margin)-50)
                .attr("width", (width-margin)-(margin))
                .attr("height", 50)
                .style("fill", "url(#linear-gradient1)")


        })
        .catch(function(error){
            console.log(error)
        })
```

```
})
</script>
```



Transitions

```
from IPython.display import HTML, Javascript, display

def configure_d3():
    display(Javascript("""
    require.config({
      paths: {
        d3: "https://d3js.org/d3.v6.min"
      }
    })"""))


configure_d3()
```

Starting Graph

```
%%html
<div id="gohere1"></div>

<script type="text/javascript">
require(['d3'], function (d3) {
    const width = 300
    const height = 100
    margin = 40
    const dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere1").append("svg")
        .attr("width", width)
        .attr("height", height)

    const x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset,(d,i)=>
i))

    svg.selectAll("circle")
    .data(dataset)
    .join("circle")
    .attr("cx", (d,i)=> x(i))
    .attr("cy", height/2)
    .attr("r", (d,i)=> d)
});
</script>
```

Transition

```
%%html
<div id="gohere2"></div>

<script type="text/javascript">
require(['d3'], function (d3) {
    const width = 300
    const height = 100
    margin = 40
    const dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere2").append("svg")
        .attr("width", width)
        .attr("height", height)

    const x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset,(d,i)=>
i))

    svg.selectAll("circle")
    .data(dataset)
    .join("circle")
    .attr("cx", (d,i)=> x(i))
    .attr("cy", height/2)
    .attr("r", 0)
    .transition()
    .attr("r", (d,i)=> d)
});
</script>
```

Duration

Amount of time the transition takes in ms (1000 ms = 1 second)

```
%%html
<div id="gohere3"></div>

<script type="text/javascript">
require(['d3'], function (d3) {
    const width = 300
    const height = 100
    margin = 40
    const dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere3").append("svg")
        .attr("width", width)
        .attr("height", height)

    const x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset,(d,i)=>
i))

    svg.selectAll("circle")
    .data(dataset)
    .join("circle")
    .attr("cx", (d,i)=> x(i))
    .attr("cy", height/2)
    .attr("r", 0)
    .transition()
    .duration(10000)
    .attr("r", (d,i)=> d)
});
</script>
```

Delay

Setting a specific delay on each element based on an amount of time.

```
%%html
<div id="gohere4"></div>

<script type="text/javascript">
require(['d3'], function (d3) {
    const width = 300
    const height = 100
    margin = 40
    const dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere4").append("svg")
        .attr("width", width)
        .attr("height", height)

    const x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset,(d,i)=>
i))

    svg.selectAll("circle")
    .data(dataset)
    .join("circle")
    .attr("cx", (d,i)=> x(i))
    .attr("cy", height/2)
    .attr("r", 0)
    .transition()
    .duration(1000)
    .delay((d,i)=> i*100)
    .attr("r", (d,i)=> d)
});
</script>
```

Easing

The pattern the transition use. https://observablehq.com/@d3/easing-animations

```
%%html
<div id="gohere5"></div>

<script type="text/javascript">
require(['d3'], function (d3) {
    const width = 300
    const height = 100
    margin = 40
    const dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere5").append("svg")
        .attr("width", width)
        .attr("height", height)

    const x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset,(d,i)=>
i))

    svg.selectAll("circle")
        .data(dataset)
        .join("circle")
        .attr("cx", (d,i)=> x(i))
        .attr("cy", height/2)
        .attr("r", 2)
        .transition()
        .duration(1000)
        .attr("cy", height/3)
        .transition()
        .duration(1000)
        .attr("r", (d,i)=> d)
});
</script>
```

Using inputs for transitions.

There are a number of different types of inputs, but we will be covering the following:

- Buttons
- Radio buttons
- Checkboxes
- Range

Buttons

```
%%html
<input type="button" id="button1" onclick="console.log('Hello, Everyone!')" value="Click the
button">
```

Click the button

Radio buttons

```
%%html
<input type="radio" id="vanilla" name="iceCream" value="vanilla">
<label for="vanilla">Vanilla</label><br>
<input type="radio" id="chocolate" name="iceCream" value="chocolate">
<label for="chocolate">Chocolate</label><br>
<input type="radio" id="other" name="iceCream" value="other" checked>
<label for="other">Other</label>
```

- ○ Vanilla
- ○ Chocolate
- ○ Other

```
%%html
<input type="radio" id="vanilla" name="iceCream" value="vanilla" checked>
<label for="vanilla">Vanilla</label><br>
<input type="radio" id="chocolate" name="iceCream" value="chocolate">
<label for="chocolate">Chocolate</label><br>
<input type="radio" id="other" name="iceCream" value="other">
<label for="other">Other</label>
```

- ◉ Vanilla
- ○ Chocolate
- ○ Other

Checkbox

```
%%html
<input type="checkbox" id="toppings1" name="toppings1" value="sprinkles" checked>
<label for="toppings1">Sprinkles</label><br>
<input type="checkbox" id="toppings2" name="toppings2" value="whippedCream" checked>
<label for="toppings2">Whipped Cream</label><br>
<input type="checkbox" id="toppings3" name="toppings3" value="cherry" checked>
<label for="toppings3">Cherry</label>
```

- ☑ Sprinkles
- ☑ Whipped Cream
- ☑ Cherry

Sliders

```
%%html
<input type="range" id="orders" name="orders" min="0" max="10" value="5" step="5"
style="width:200px">
<label for="orders">Number of Orders</label>
```

Number of Orders

Adding a button

```
%%html
<div id="gohere6"></div>
<input type="button" id="button2" value="Run the Transition!">
<script type="text/javascript">
require(['d3'], function (d3) {
    const width = 300
    const height = 100
    margin = 40
    const dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere6").append("svg")
        .attr("width", width)
        .attr("height", height)

    const x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset,(d,i)=>
i))

    svg.selectAll("circle")
        .data(dataset)
        .join("circle")
        .attr("cx", (d,i)=> x(i))
        .attr("cy", height/2)
        .attr("r", 1)

    // d3.select can select any element on our page
    d3.select("input#button2")
        .on("click",function() {
            console.log("button pushed!")
        })


});
</script>
```

Run the Transition!

```
%%html
<div id="gohere7"></div>
<input type="button" id="button3" value="Run the Transition!">
<script type="text/javascript">
require(['d3'], function (d3) {
    const width = 300
    const height = 100
    margin = 40
    const dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere7").append("svg")
        .attr("width", width)
        .attr("height", height)

    const x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset,(d,i)=>
i))

    svg.selectAll("circle")
        .data(dataset)
        .join("circle")
        .attr("cx", (d,i)=> x(i))
        .attr("cy", height/2)
        .attr("r", 1)

    // d3.select can select any element on our page
    d3.select("input#button3")
        .on("click",function() {
            // svg.select selects things in the given graph
            svg.selectAll("circle")
                .transition()
                .duration(1000)
                .delay((d,i)=> i*100)
                .attr("r", (d,i)=> d)
        })

});
</script>
```

Run the Transition!

```
%%html
<div id="gohere8"></div>
<input type="button" id="button4" value="Run the Transition!">
<script type="text/javascript">
require(['d3'], function (d3) {
    const width = 300
    const height = 100
    margin = 40
    const dataset = [3, 5, 5, 6, 15, 18]

    const svg = d3.select("div#gohere8").append("svg")
        .attr("width", width)
        .attr("height", height)

    const x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(dataset,(d,i)=>
i))

    svg.selectAll("circle")
        .data(dataset)
        .join("circle")
        .attr("cx", (d,i)=> x(i))
        .attr("cy", height/2)
        .attr("r", 1)

    var selected = 1;

    // d3.select can select any element on our page
    d3.select("input#button4")
        .on("click",function() {
            // svg.select selects things in the given graph
            if (selected) {
                svg.selectAll("circle")
                    .transition()
                    .duration(1000)
                    .delay((d,i)=> i*100)
                    .attr("r", (d,i)=> d)
                selected = 0;
            }
            else {
                svg.selectAll("circle")
                    .transition()
                    .duration(1000)
                    .delay((d,i)=> i*100)
                    .attr("r", (d,i)=> 1)
                selected = 1;
            }

        })

});
</script>
```

Run the Transition!

Specific Designs - Intro

Connected Scatterplot

Our Data Set

```
import pandas as pd
url="https://gist.githubusercontent.com/dudaspm/e518430a731ac11f52de9217311c674d/raw/4c2f2bd663
9582a420ef321493188deebc4a575e/StateCollege2000-2020.csv"
data = []
data=pd.read_csv(url)
data = data.fillna(0) # replace all NAs with 0s
data.to_csv('weather.csv', index = False, header=True)
data.head()
```

	DATE	DAY	MONTH	YEAR	PRCP	SNOW	TMAX	TMIN	WT_FOG	WT_THUNDER
0	1/1/2000	1	1	2000	0.00	0.0	44.0	23	0.0	0.0
1	1/2/2000	2	1	2000	0.00	0.0	52.0	23	0.0	0.0
2	1/3/2000	3	1	2000	0.01	0.0	60.0	35	0.0	0.0
3	1/4/2000	4	1	2000	0.12	0.0	62.0	54	0.0	0.0
4	1/5/2000	5	1	2000	0.04	0.0	60.0	30	0.0	0.0

Acknowledgement

Cite as: Menne, Matthew J., Imke Durre, Bryant Korzeniewski, Shelley McNeal, Kristy Thomas, Xungang Yin, Steven Anthony, Ron Ray, Russell S. Vose, Byron E.Gleason, and Tamara G. Houston (2012): Global Historical Climatology Network - Daily (GHCN-Daily), Version 3. CITY:US420020. NOAA National Climatic Data Center. doi:10.7289/V5D21VHZ 02/22/2021.

Publications citing this dataset should also cite the following article: Matthew J. Menne, Imke Durre, Russell S. Vose, Byron E. Gleason, and Tamara G. Houston, 2012: An Overview of the Global Historical Climatology Network-Daily Database. J. Atmos. Oceanic Technol., 29, 897-910. doi:10.1175/JTECH-D-11-00103.1.

Use liability: NOAA and NCEI cannot provide any warranty as to the accuracy, reliability, or completeness of furnished data. Users assume responsibility to determine the usability of these data. The user is responsible for the results of any application of this data for other than its intended purpose.

Links: https://data.noaa.gov/onestop/

https://www.ncdc.noaa.gov/cdo-web/search

The first step in the process is to take a piece of paper and draw a picture of what you believe the graph will look like. While doing this, I would suggest answering the following questions:

1. Is the data continuous or discrete?

- For *continuous* we assume the data has a range of values
- For *discrete* we assume the data can be binned
    - this question can be a challenging first conversation to have, but the basic rule is: can you use a ruler? If so, continuous. If not, discrete.

1. Are you interested in *comparing* your data points, or are you looking for a *trend* in your data?
2. Which variables are our independent variables or our dependent variables?

- For *independent* this is what we control
- For *dependent* with this control, this is what we are measuring.

For us, I would say both our continuous, we are interested in the trend of the data, and date is independent and interest in the dependent. So, let's draw this out.

Start D3.js

```
from IPython.display import HTML, Javascript, display

def configure_d3():
    display(Javascript("""
    require.config({
      paths: {
        d3: "https://d3js.org/d3.v6.min"
      }
    })"""))


configure_d3()
```

Getting the Data into D3.js

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('https://raw.githubusercontent.com/owid/covid-19-
data/master/public/data/vaccinations/us_state_vaccinations.csv')
        .then(function(data) {
            console.log(data[0])
        })
        .catch(function(error){

        })

})
</script>
```

Filter the Data

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            console.log(data)
        })
        .catch(function(error){

        })

})
</script>
```

Mapping the Data

Now let's convert our data from its original format to an updated format using a function called d3.map(). d3.map() is a really nice function that can take your original data and convert it to other formats. Here is a simple example where we have an array of numbers, but we want to re-map them, so the values are multiplied by pi.

```
%%javascript
var justSomeData = [1,2,3,4]
const output = justSomeData.map((d,i)=> d * Math.PI)
element.text(output)
```

```
%%javascript
var justSomeData = [1,2,3,4]
const output = justSomeData.map((d,i)=> d + "\uD83D\uDC08")
element.text(output[0])
```

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":d.DATE,"TMAX":d.TMAX}))

            console.log(data)

        })
        .catch(function(error){

        })

})
</script>
```

Adjusting the data

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":d.DATE,"TMAX":+d.TMAX}))

            console.log(data)

        })
        .catch(function(error){

        })

})
</script>
```

Ok, the first step we need a function to convert our date in a way to make it usable in our chart. We can use a built-in function in D3.js called timeParse. timeParse needs to know how the date/time is formatted in our text and then use a specific format (called a specifier string).

If we look at our dates we have (for example) “01/2019” or “month/full year”.

Here is a list of all specifier strings, and a smaller list for formats that I typically use:

- **%d** - zero-padded day of the month as a decimal number [01,31]
- **%H** - hour (24-hour clock) as a decimal number [00,23]
- **%m** - month as a decimal number [01,12]
- **%M** - minute as a decimal number [00,59]
- **%y** - year without century as a decimal number [00,99]
- **%Y** - year with century as a decimal number, such as 1999

Meaning, we need a specifier string of “**%m/%Y**” = “**month/full year**”

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"TMAX":+d.TMAX}))
            console.log(data)

        })
        .catch(function(error){

        })

})
</script>
```

Layout

For the artwork itself, we will need a few basic things: a width, height, and margins.

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const width = 600
            const height = 300
            const margin = 60
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"TMAX":+d.TMAX}))

        })
        .catch(function(error){

        })

})
</script>
```

Next, we will need to have a function to scale our date. Again, we are assuming our “date” will be our independent data (the x-axis) and the “interest” our dependent data (y-axis). D3.js has a specific way to scale our time called d3.scaleTime, and for the “interest,” we will use the d3.scaleLinear as we did before.

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const width = 600
            const height = 300
            const margin = 60
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"TMAX":+d.TMAX}))

            const xScale = d3.scaleTime().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => d.DATE))
            const yScale = d3.scaleLinear().range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.TMAX))


        })
        .catch(function(error){

        })

})
</script>
```

Finally (for now), we will make our function to create the line itself. This will be a function that creates the path from the data. Remember, *DATE is the x* and *TMAX is the y*.`

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const width = 600
            const height = 300
            const margin = 60
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"TMAX":+d.TMAX}))

            const xScale = d3.scaleTime().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => d.DATE))
            const yScale = d3.scaleLinear().range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.TMAX))

            const line = d3.line()
                .x((d,i)=> xScale(d.DATE))
                .y((d,i)=> yScale(d.TMAX))

        })
        .catch(function(error){

        })

})
</script>
```

Add our line

```
%%html
<div id="gohere1"></div>

<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const width = 600
            const height = 300
            const margin = 60
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"TMAX":+d.TMAX}))

            const xScale = d3.scaleTime().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => d.DATE))
            const yScale = d3.scaleLinear().range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.TMAX))

            const line = d3.line()
                .x((d,i)=> xScale(d.DATE))
                .y((d,i)=> yScale(d.TMAX))

            const svg = d3.select("div#gohere1").append("svg")
                .attr("width", width)
                .attr("height", height)

            svg.selectAll("path")
                .data([data])
                .join("path")
                .attr("d", function(d,i) { return line(d) })
                .style("stroke", "green" )
                .style("stroke-width", 3)
                .style("fill", "none")

        })
        .catch(function(error){

        })

})
</script>
```

Adding our Circles

```
%%html
<div id="gohere2"></div>

<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const width = 600
            const height = 300
            const margin = 60
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            data = data.filter(d=> (d.MONTH=="3") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"TMAX":+d.TMAX}))

            const xScale = d3.scaleTime().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => d.DATE))
            const yScale = d3.scaleLinear().range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.TMAX))

            const line = d3.line()
                .x((d,i)=> xScale(d.DATE))
                .y((d,i)=> yScale(d.TMAX))


            const svg = d3.select("div#gohere2").append("svg")
                .attr("width", width)
                .attr("height", height)

             svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(d.DATE))
                .attr("cy", (d,i)=> yScale(d.TMAX))
                .attr("r", 5)
                .style("fill", "none")
                .style("stroke", "black" )
                .style("stroke-width", 3)

            svg.selectAll("path")
                .data([data])
                .join("path")
                .attr("d", function(d,i) { return line(d) })
                .style("stroke", "green" )
                .style("stroke-width", 3)
                .style("fill", "none")



        })
        .catch(function(error){

        })

})
</script>
```

That's it! That is all the code we need to make the artwork. Now, we probably want to add an axis for our x and y values. Let's see how that will look within our code.

The x-axis will need to have the axis label at the bottom of the axis. This is pretty straight-forward by using the axisBottom() function to make the axis text on the bottom. The scale of our axis (the smallest, largest, and the labels in-between) will be based on our x values or scaleTime() values. So, set this as our scale.

```
%%html
<div id="gohere3"></div>

<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const width = 600
            const height = 300
            const margin = 60
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            data = data.filter(d=> (d.MONTH=="5") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"TMIN":+d.TMIN}))

            const xScale = d3.scaleTime().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => d.DATE))
            const yScale = d3.scaleLinear().range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.TMIN))

            const line = d3.line()
                .x((d,i)=> xScale(d.DATE))
                .y((d,i)=> yScale(d.TMIN))


            const svg = d3.select("div#gohere3").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale).tickFormat(d3.timeFormat("%d"))

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.selectAll("path.temp")
                .data([data])
                .join("path")
                .attr("d", function(d,i) { return line(d) })
                .attr("class", "temp")
                .style("stroke", "green" )
                .style("stroke-width", 3)
                .style("fill", "none")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(d.DATE))
                .attr("cy", (d,i)=> yScale(d.TMIN))
                .attr("r", 5)

        })
        .catch(function(error){

        })

})
</script>
```

The y-axis will need to have the axis label to the left. Again, straight-forward, axisLeft(), and the scaling is based on our y values or scaleLinear() values.

```
%%html
<div id="gohere4"></div>

<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const width = 600
            const height = 300
            const margin = 60
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            data = data.filter(d=> (d.MONTH=="5") && (d.YEAR=="2020"))
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"TMAX":+d.TMIN}))

            const xScale = d3.scaleTime().range([margin , width -
margin]).domain(d3.extent(data, (d,i) => d.DATE))
            const yScale = d3.scaleLinear().range([height-margin ,
margin]).domain(d3.extent(data, (d,i) => d.TMAX))

            const line = d3.line()
                .x((d,i)=> xScale(d.DATE))
                .y((d,i)=> yScale(d.TMAX))


            const svg = d3.select("div#gohere4").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(xScale).tickFormat(d3.timeFormat("%d"))

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)


            const yAxis = d3.axisLeft().scale(yScale).tickFormat((d,i) => d + "°")

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.selectAll("path.temp")
                .data([data])
                .join("path")
                .attr("d", function(d,i) { return line(d) })
                .attr("class", "temp")
                .style("stroke", "green" )
                .style("stroke-width", 3)
                .style("fill", "none")

            svg.selectAll("circle")
                .data(data)
                .join("circle")
                .attr("cx", (d,i)=> xScale(d.DATE))
                .attr("cy", (d,i)=> yScale(d.TMAX))
                .attr("r", 5)

        })
        .catch(function(error){

        })

})
</script>
```

Bar Charts

Data

```
import pandas as pd
url="https://gist.githubusercontent.com/dudaspm/e518430a731ac11f52de9217311c674d/raw/4c2f2bd6639582a420ef321493188deebc4a575e/StateCollege2000-2020.csv"
data = []
data=pd.read_csv(url)
data = data.fillna(0) # replace all NAs with 0s
data.to_csv('weather.csv', index = False, header=True)
data.head()
```

	DATE	DAY	MONTH	YEAR	PRCP	SNOW	TMAX	TMIN	WT_FOG	WT_THUNDER
0	1/1/2000	1	1	2000	0.00	0.0	44.0	23	0.0	0.0
1	1/2/2000	2	1	2000	0.00	0.0	52.0	23	0.0	0.0
2	1/3/2000	3	1	2000	0.01	0.0	60.0	35	0.0	0.0
3	1/4/2000	4	1	2000	0.12	0.0	62.0	54	0.0	0.0
4	1/5/2000	5	1	2000	0.04	0.0	60.0	30	0.0	0.0

Acknowledgement

Cite as: Menne, Matthew J., Imke Durre, Bryant Korzeniewski, Shelley McNeal, Kristy Thomas, Xungang Yin, Steven Anthony, Ron Ray, Russell S. Vose, Byron E.Gleason, and Tamara G. Houston (2012): Global Historical Climatology Network - Daily (GHCN-Daily), Version 3. CITY:US420020. NOAA National Climatic Data Center. doi:10.7289/V5D21VHZ 02/22/2021.

Publications citing this dataset should also cite the following article: Matthew J. Menne, Imke Durre, Russell S. Vose, Byron E. Gleason, and Tamara G. Houston, 2012: An Overview of the Global Historical Climatology Network-Daily Database. J. Atmos. Oceanic Technol., 29, 897-910. doi:10.1175/JTECH-D-11-00103.1.

Links: https://data.noaa.gov/onestop/

https://www.ncdc.noaa.gov/cdo-web/search

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12), 2301-2309.

```
from IPython.display import HTML, Javascript, display

def configure_d3():
    display(Javascript("""
    require.config({
      paths: {
        d3: "https://d3js.org/d3.v6.min"
      }
    })"""))


configure_d3()
```

Group

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            const daysOfTheWeek = d3.timeFormat("%a")
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"PRCP":+d.PRCP}))
            console.log(d3.group(data, d => daysOfTheWeek(d.DATE)))
        })
        .catch(function(error){

        })

})
</script>
```

Rollup

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            const daysOfTheWeek = d3.timeFormat("%a")
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"PRCP":+d.PRCP}))
            console.log(d3.rollup(data, v => d3.mean(v, d => d.PRCP), k =>
daysOfTheWeek(k.DATE)))
        })
        .catch(function(error){

        })

})
</script>
```

ScaleBand

```
%%html

<script type="text/javascript">
require(['d3'], function (d3) {

    const someData = [0,4,14,20,30,31,42,50,59,62]
    tryingScaleBands = d3.scaleBand().range([0,100]).domain(d3.extent(someData))
    d3.select("div#graph1").text(someData.map(d=>tryingScaleBands(d)))
})
</script>
<div id="graph1"></div>
```

```
%%html

<script type="text/javascript">
require(['d3'], function (d3) {

    const someData = [0,4,14,20,30,31,42,50,59,62]
    tryingScaleBands = d3.scaleBand().range([0,100]).domain(someData.map(d=>d))
    d3.select("div#graph2").text("list length: "+(someData.length)+"   scaleBand output:
"+someData.map(d=>tryingScaleBands(d)))
})
</script>
<div id="graph2"></div>
```

This makes a bit more sense. As you can see, this is evenly spacing all of our data based on the maximum range (100) minus the minimal range value (0), then dividing this by the size (the number of values) in the list. Or...

$$ \frac{(\text{maximum range value} - \text{minimum range value})}{\text{total number in the array}} = \frac{(100-0)}{10} $$

scaleBand() has a couple of neat features that can help with bar chart design. Two in particular are called scaleBand().bandwidth and scaleBand().padding.

- scaleBand().bandwidth - will give you the distance between points in scaleBand(). Meaning, it will be perfect for our bar charts, because we will be using rectangles for our bars.
- scaleBand().padding - increases the padding between each bar.

Here is a graph showing the use of scaleBand() and using scaleBand().bandwidth to create the width of the rectangles.

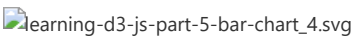
The figure below maps out 10 values and padding for 0 to 1. 0 indicating no padding and 1 meaning 100% padding (or no rectangle at all). Notice how values and the axis get evenly spaced based on the padding as well.

Creating the rectangles themselves is another significant change from the line chart. With rectangles, you need 4 components:

```
svg.append("g").selectAll("rect")
  .data(data)
  .join("rect")
the x position (as it relates to the date)
  .attr("x", (d,i)=>x(d.day))
the y position (as it relates to the interest), NOTE it is NOT the x-axis
  .attr("y",(d,i)=>y(d.avg))
the width, which we talked about in regards to using the bandwidth
  .attr("width",x.bandwidth)
this one is a bit, well, weird. I will explain below.
  .attr("height", d => y(0) - y(d.avg))
  .style("stroke-width", 2)
  .style("stroke","black")
  .style("fill", "steelblue")
```

OK, let's talk about height. The weirdness stems from our 0,0 being in the top-left corner of the screen. Meaning, when we create a rectangle and add a ''height,'' it goes down and not up.

Here is an example of a rectangle that starts in the middle of the box. When we add height, it goes down. You may think, well, can I use a negative height? The answer is no. What does this mean? Continue below.



This how we get the following. We first need to recall that the y is
.attr("y",(d,i)=>y(d.avg))
This is NOT the x-axis, but the position of the actual value at that given avg.

Next, we take the maximum height value from our y scaleLinear().

```
y = d3.scaleLinear().range([height-margin.bottom , margin.top]).domain([0,d3.max(backToList, (d,i) => d.avg)])
```

The largest value is (height-margin.bottom) or in another words, the smallest index in our y scaleLinear() (y(0)) Last part we need to remember that we are starting at y(d.interest) and we trying to get back to the x-axis. Meaning, we need to subtract out y(d.avg)

We can write out the height two different ways:

```
.attr("height", d => y(0) - y(d.avg))
```

OR

```
.attr("height", d => (height-margin.bottom) - y(d.avg))
```

My choice? .attr("height", d => (height-margin.bottom) - y(d.avg))
because this will be true no matter what the minimal value is for y. It is constant.

Map - JavaScript | MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map. Accessed 9 Apr. 2021.

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            const daysOfTheWeek = d3.timeFormat("%a")
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"PRCP":+d.PRCP}))
            const nestedData = d3.rollup(data, v => d3.mean(v, d => d.PRCP), d =>
daysOfTheWeek(d.DATE))
            console.log(nestedData)
            var backToList = []
            for (let [key, value] of nestedData) {
                console.log(key + ' = ' + value)
                backToList.push({"day":key,"avg":value})
            }
            console.log(backToList)


        })
        .catch(function(error){

        })

})
</script>
```

Graph

```
%%html
<div id="graph3"></div>
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.csv('weather.csv')
        .then(function(data) {
            const dateConverter = d3.timeParse("%_m/%_d/%Y")
            const daysOfTheWeek = d3.timeFormat("%a")
            data = data.map(d=> ({"DATE":dateConverter(d.DATE),"PRCP":+d.PRCP}))
            const nestedData = d3.rollup(data, v => d3.mean(v, d => d.PRCP), k =>
daysOfTheWeek(k.DATE))
            var backToList = []
            for (let [key, value] of nestedData) {
                backToList.push({"day":key,"avg":value})
            }
            const width = 600
            const height = 300
            const margin = 60
            const svg = d3.select("div#graph3").append("svg")
                .attr("width", width)
                .attr("height", height)

            const x = d3.scaleBand().range([margin , width -
margin]).domain(backToList.map(d=>d.day)).padding(0)
            const y = d3.scaleLinear().range([height-margin ,
margin]).domain([0,d3.max(backToList, (d,i) => d.avg)])

            const xAxis = d3.axisBottom().scale(x)
            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("Days of the Week")

            const yAxis = d3.axisLeft().scale(y)
            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + margin + ",0)")
                .call(yAxis)

            svg.append("text")
                .attr("transform", "rotate(-90,15,"+(height/2)+")")
                .attr("x", 15)
                .attr("y", height/2)
                .style("text-anchor", "middle")
                .text("Average Rainfall (inches)")

            svg.append("g").selectAll("rect")
                .data(backToList)
                .join("rect")
                .attr("x", (d,i)=>x(d.day))
                .attr("y",(d,i)=>y(d.avg))
                .attr("width",x.bandwidth)
                .attr("height", d => (height-margin) - y(d.avg))
                .style("stroke-width", 2)
                .style("stroke","black")
                .style("fill", "steelblue")
                .append("title")
                .text(d=>d.avg)

        })
        .catch(function(error){

        })

})
</script>
```

Network Graph

For this exercise we will be creating a random graph using NetworkX.

Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

https://networkx.org/

https://networkx.org/documentation/latest/auto_examples/index.html

We will be using the Barabási–Albert algorithm to create the network

A. L. Barabási and R. Albert "Emergence of scaling in random networks", Science 286, pp 509-512, 1999.
https://networkx.org/documentation/stable/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html?highlight=barabasi#networkx.generators.random_graphs.barabasi_albert_graph

D3.js Force-Directed Graph Notebook: https://observablehq.com/@d3/force-directed-graph

```
%%capture
!pip3 install networkx
```

```
import networkx as nx
import json

graphsize = 100
nodes = {}
links = {}
Graph = nx.barabasi_albert_graph(graphsize, 1)
j = "" # JSON object
j = j + "{"
j = j + """\t"nodes": ["""

for n in nx.nodes(Graph):
    nodes[n] = {}
    nodes[n]['name'] = n
for n in nodes:
    j = j + str(json.dumps(nodes[n])) + ",\n"
j = j[:-2]
j = j + "\t],\n"
j = j + """\t"links":[\n"""
for link in nx.edges(Graph):
    links[str(link)] = {}
    links[str(link)]['source'] = link[0]
    links[str(link)]['target'] = link[1]
for l in links:
    j = j + str(json.dumps(links[l])) + ",\n"
j = j[:-2]
j = j + "\t]\n"
j = j + "}"
#print (j)

f = open("network.json", "w")
f.write(j)
f.close()
```

Start D3.js

```
from IPython.display import HTML, Javascript, display

def configure_d3():
    display(Javascript("""
    require.config({
      paths: {
        d3: "https://d3js.org/d3.v6.min"
      }
    })"""))


configure_d3()
```

Getting the Data into D3.js

```
%%html
<script type="text/javascript">
require(['d3'], function (d3) {

    d3.json('network.json')
        .then(function(data) {
            nodes = data.nodes.map(d=> d)
            console.log(nodes)
            links = data.links.map(d=> ({source:nodes[d.source], target:nodes[d.target]}))
            console.log(links)
        })
        .catch(function(error){

        })

})
</script>
```

Creating the Graph

```
%%html
<div id="gohere1"></div>

<script type="text/javascript">
require(['d3'], function (d3) {

    d3.json('network.json')
        .then(function(data) {
            nodes = data.nodes.map(d=> d)
            links = data.links.map(d=> ({source:nodes[d.source], target:nodes[d.target]}))
            const width = 600
            const height = 600

            const svg = d3.select("div#gohere1").append("svg")
                .attr("width", width)
                .attr("height", height)

            const link = svg
                .selectAll("line.link")
                .data(links)
                .join("line")
                .attr("class", "link")
                .style("stroke", "#999")
                .style("stroke-width", ".6px");

            // append the nodes with specified data and style properties
            const node = svg
                .selectAll("circle.node")
                .data(nodes)
                .join("circle")
                .attr("class", "node")
                .attr("r", 5)
                .style("stroke", "#fff")
                .style("stroke-width", "1.5px")
                .style("fill", (d,i)=> (i%2) ? "steelblue" : "purple")

            // attach titles to nodes, so when the mouse hovers over the nodes it projects the
name
            node.append("title")
                .text(function(d) { return d.name; });
        })
        .catch(function(error){

        })

})
</script>
```

Simulation Time

```
%%html
<div id="gohere2"></div>

<script type="text/javascript">
require(['d3'], function (d3) {

    d3.json('network.json')
        .then(function(data) {
            nodes = data.nodes.map(d=> d)
            links = data.links.map(d=> ({source:nodes[d.source], target:nodes[d.target]}))
            const width = 600
            const height = 600

            const svg = d3.select("div#gohere2").append("svg")
                .attr("width", width)
                .attr("height", height)

            const link = svg
                .selectAll("line.link")
                .data(links)
                .join("line")
                .attr("class", "link")
                .style("stroke", "#999")
                .style("stroke-width", ".6px");

            // append the nodes with specified data and style properties
            const node = svg
                .selectAll("circle.node")
                .data(nodes)
                .join("circle")
                .attr("class", "node")
                .attr("r", 5)
                .style("stroke", "#fff")
                .style("stroke-width", "1.5px")
                .style("fill", (d,i)=> (i%2) ? "steelblue" : "purple")

            // attach titles to nodes, so when the mouse hovers over the nodes it projects the
name
            node.append("title")
                .text(function(d) { return d.name; });

            // this actually adheres the data to the force base layout and starts the layout
            const simulation = d3.forceSimulation(nodes)
                .force("link",
                       d3.forceLink(links).id(d => d.name)
                       .distance(10)
                       .strength(1)
                      )
                .force("charge", d3.forceManyBody())
                .force("center", d3.forceCenter(width / 2, height / 2))
                .on("tick", ticked);

            // this is the main mechanism of the force based diagram
            // which moves the nodes and links from their starting and finishing positions
            // once it hits equilibrium, it will stop moving the positions
            function ticked() {
                svg.selectAll("line.link")
                    .attr("x1", d => d.source.x )
                    .attr("y1", d => d.source.y )
                    .attr("x2", d => d.target.x )
                    .attr("y2", d => d.target.y )

                svg.selectAll("circle.node")
                    .attr("cx", d => d.x)
                    .attr("cy", d => d.y)
            };
        })
        .catch(function(error){

        })

})
</script>
```

Making it move

```
%%html
<div id="gohere3"></div>

<script type="text/javascript">
require(['d3'], function (d3) {

    d3.json('network.json')
        .then(function(data) {
            nodes = data.nodes.map(d=> d)
            links = data.links.map(d=> ({source:nodes[d.source], target:nodes[d.target]}))
            const width = 600
            const height = 600

            const svg = d3.select("div#gohere3").append("svg")
                .attr("width", width)
                .attr("height", height)

            const link = svg
                .selectAll("line.link")
                .data(links)
                .join("line")
                .attr("class", "link")
                .style("stroke", "#999")
                .style("stroke-width", ".6px");

            // append the nodes with specified data and style properties
            const node = svg
                .selectAll("circle.node")
                .data(nodes)
                .join("circle")
                .attr("class", "node")
                .attr("r", 5)
                .style("stroke", "#fff")
                .style("stroke-width", "1.5px")
                .style("fill", (d,i)=> (i%2) ? "steelblue" : "purple")
                .call(d3.drag()
                      .on("start", dragstarted)
                      .on("drag", dragged)
                      .on("end", dragended))

            // attach titles to nodes, so when the mouse hovers over the nodes it projects the
name
            node.append("title")
                .text(function(d) { return d.name; });

            // this actually adheres the data to the force base layout and starts the layout
            const simulation = d3.forceSimulation(nodes)
                .force("link",
                       d3.forceLink(links).id(d => d.name)
                       .distance(d => 0)
                       .strength(1)
                      )
                .force("charge", d3.forceManyBody())
                .force("center", d3.forceCenter(width / 2, height / 2))
                .on("tick", ticked);

            // this is the main mechanism of the force based diagram
            // which moves the nodes and links from their starting and finishing positions
            // once it hits equilibrium, it will stop moving the positions
            function ticked() {
                svg.selectAll("line.link").attr("x1", function(d) { return d.source.x; })
                    .attr("y1", function(d) { return d.source.y; })
                    .attr("x2", function(d) { return d.target.x; })
                    .attr("y2", function(d) { return d.target.y; });
                svg.selectAll("circle.node").attr("cx", function(d) { return d.x; })
                    .attr("cy", function(d) { return d.y; });
            };


            function dragstarted(event) {
                if (!event.active) simulation.alphaTarget(0.3).restart();
                event.subject.fx = event.subject.x;
                event.subject.fy = event.subject.y;
            }

            function dragged(event) {
                event.subject.fx = event.x;
                event.subject.fy = event.y;
            }

            function dragended(event) {
                if (!event.active) simulation.alphaTarget(0);
                event.subject.fx = event.x;
                event.subject.fy = event.y;
            }

        })
```

```
        .catch(function(error){

        })

})
</script>
```

Making a Map

Let's talk about making a map in D3. D3 is GREAT for making maps and it pretty straightforward to do so.

Let's create a [choropleth map](https://en.wikipedia.org/wiki/Choropleth_map#:~:text=A choropleth map (from Greek,each area, such as population) for PA.

> A really good resource for this: https://observablehq.com/@floledermann/drawing-maps-from-geodata-in-d3

```
from IPython.display import  HTML

def load_d3_in_cell_output():
  display(HTML("<script src='https://d3js.org/d3.v6.min.js'></script>"))
get_ipython().events.register('pre_run_cell', load_d3_in_cell_output)
```

Data (Shapefiles)

Acknowledgement

Bureau, US Census. "Cartographic Boundary Files - Shapefile." The United States Census Bureau, https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html. Accessed 25 May 2021.
[]

```
import requests
# URL to https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html
url = 'https://www2.census.gov/geo/tiger/GENZ2018/shp/cb_2018_us_county_20m.zip'
# get the zip file
r = requests.get(url)
print (r.content)
# create a new file called 2018_us_county_500k.zip
f = open('cb_2018_us_county_20m.zip', "wb")
# write the zip from www2.census.gov to this new file
f.write(r.content)
# close the new file
f.close()
```

```
b''
```

The way we need to get this zip file to ogre.adc4gis.com is to use a POST. Thanks to Rob Story's Github file for providing much of the details here []

```
import requests
import json
url = r'http://ogre.adc4gis.com/convert'
f = open('cb_2018_us_county_20m.zip', "rb")
shp_data = {'upload': f}
print('Calling Ogre to perform shapefile to geoJSON conversion...')
try:
    r = requests.post(url, files=shp_data)
except:
    print("There was an error with the HTTP request")
# close the new file
f.close()
with open("cb_2018_us_county_20m.json", 'w') as f:
    f.write(json.dumps(r.json(),separators=(',', ':')))
print("File is ready, called: cb_2018_us_county_20m.json")
```

```
Calling Ogre to perform shapefile to geoJSON conversion...
File is ready, called: cb_2018_us_county_20m.json
```

Let’s look at our data

This is a basic “read-in” function for D3.js, specifically, for JSON.

For this function, d3.js uses Promises to make sure we get our file first, then tell the program what we are going to do with this file (using *.then*) and if it does go as planned, an error (using *.catch*). This is important to understand because your data is not available **outside** of the promise.

For the purposes of viewing the structure of a typical geo(json) file. I made a mini-version. Here is the structure of these files.

Preview of the JSON file


```
{
 "features": [
  {
   "geometry": {
    "coordinates": [
     [
      [
       -78.901998,
       34.835268
      ]
     ]
    ],
    "type": "Polygon"
   },
   "properties": {
    "AFFGEOID": "0500000US37017",
    "ALAND": 2265887723,
    "AWATER": 33010866,
    "COUNTYFP": "017",
    "COUNTYNS": "01026336",
    "GEOID": "37017",
    "LSAD": "06",
    "NAME": "Bladen",
    "STATEFP": "37"
   },
   "type": "Feature"
  }
 ],
 "type": "FeatureCollection"
}
```


As you can see, the structure of this document contains a FeatureCollection and each *Feature* is a reference to a county. Let's take a look at the first county.

The first feature (or location)


```
{
 "geometry": {
  "coordinates": [
   [
    [
     -78.901998,
     34.835268
    ]
   ]
  ],
  "type": "Polygon"
 },
 "properties": {
  "AFFGEOID": "0500000US37017",
  "ALAND": 2265887723,
  "AWATER": 33010866,
  "COUNTYFP": "017",
  "COUNTYNS": "01026336",
  "GEOID": "37017",
  "LSAD": "06",
  "NAME": "Bladen",
  "STATEFP": "37"
 },
 "type": "Feature"
}
```


For each county we have two sets of data. The *geometry* and the *properties*. The *geometry* reflects all the latitudes and longitudes to create a single county. Think of this as "Connect the Dots" and all the dots connected form the shape (or polygon) of each county. Below, we focus on the properties.

The properties of the first feature (or location)


```
{
 "AFFGEOID": "0500000US37017",
 "ALAND": 2265887723,
 "AWATER": 33010866,
 "COUNTYFP": "017",
 "COUNTYNS": "01026336",
 "GEOID": "37017",
 "LSAD": "06",
 "NAME": "Bladen",
 "STATEFP": "37"
}
```

The *properties* are the meta-data of each county. As an example, the *Name* is Bladen. There are a couple we need to point out for later but are also helpful for understanding shapefiles in general. This being the *FIPS* or the Federal Information Processing Standards. Look at the *STATEFP* for the county, this is 37, meaning North Carolina. Here is a list of the State FIPS. Now the *COUNTYFP*, is the FIPS of the that specific county within North Carolina. Finally, you might see this information combined, like the *GEOID*. The *GEOID* is 37017 or Bladen, North Carolina.

> **Common Question**
>
> Why do we need a FIPS number if the name is available?

> **Answer!**
>
> Sometimes people use different names for different locations. Example: New York and New York State. Both are correct, but would look different based on the data. Instead, we can use FIPS 36. Also, there might be inconsistencies based on what is capitalized and what is not. North Dakota is not the same as north dakota.

Creating the Projection for the US

Now that we have our data, we need to take the latitudes and longitudes and re-map them to our plot. For our purposes we need to define three things:

- the *width* of our plot
- the *height* of our plot
- the *margins* of our plot

the *re-mapping* is actually done for us using

```
d3.geoMercator().fitExtent([[margin, margin], [width - margin, height - margin]], us)
```

```
.fitExtent([[top-left corner],[bottom-right corner]], the geojson file)
```

takes two inputs:

- our dimensions for the graph [[top-left corner],[bottom-right corner]]
- and the geojson data itself

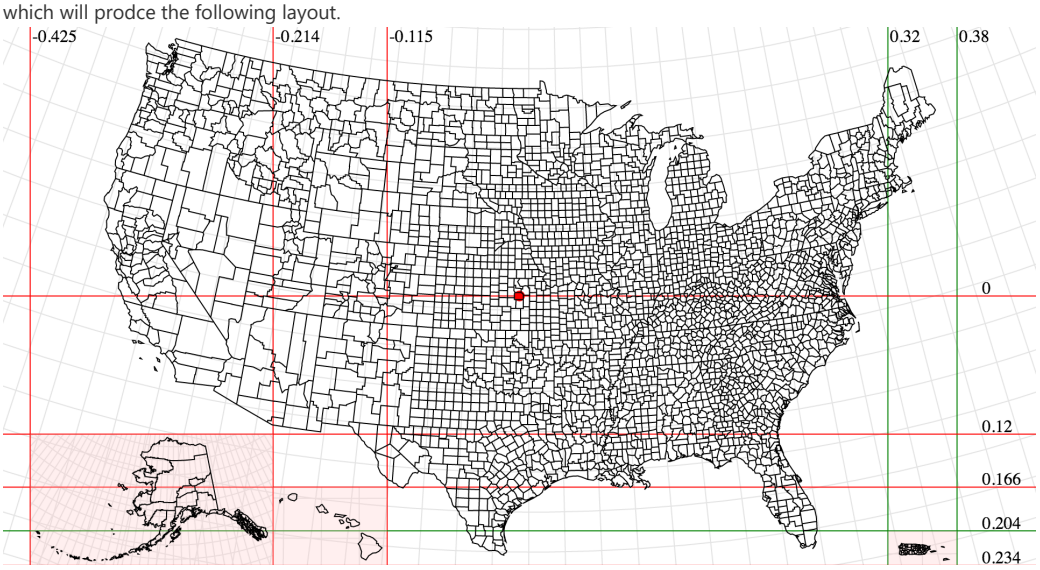
This will re-map the data.

Lastly, we need to define our *projection*. There are a ton of different projection. See https://github.com/d3/d3-geo.

We will be using the Composite Project called

```
d3.geoAlbersUsa()
```

which will prodce the following layout.




```
%%html
<div id="map1"></div>
<script type="text/javascript">


d3.json("https://raw.githubusercontent.com/dudaspm/d3plotbook/main/cb_2018_us_county_20m.json")
        .then(function(us) {
            var width = 600
            var height = 400
            var margin = 0
            // Create the Mercator Projection
            projectionUS = d3.geoAlbersUsa().fitExtent([[margin, margin], [width - margin,
height - margin]], us)
            // Create a function to generate our paths (counties)
            pathGeneratorUS = d3.geoPath().projection(projectionUS)

            const svg = d3.select("div#map1").append("svg")
                .attr("width", width)
                .attr("height", height)

            // construct the element
            svg.selectAll("path")
                .data(us.features)
                .join("path")
                .attr('d', pathGeneratorUS)
                .attr('fill', 'none')
                .attr('stroke', '#999999')
                .attr('stroke-width', '2')

        })
        .catch(function(error){
            console.log(error)
        })

</script>
```

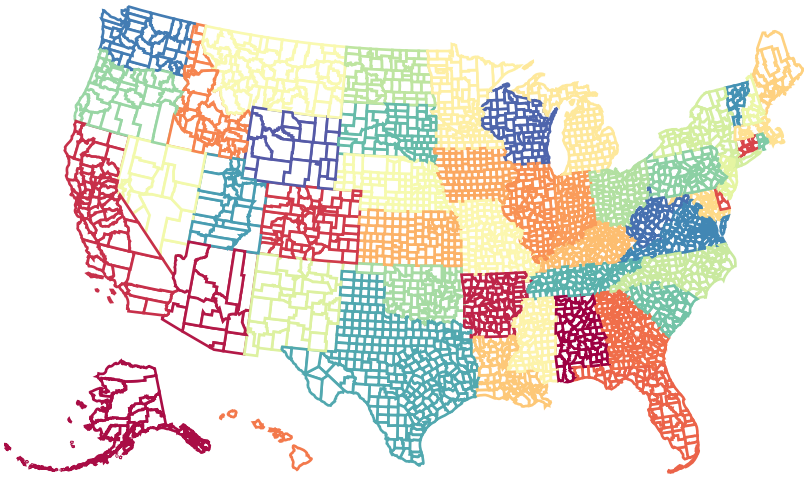
Let's add some color to this and color each state. We need to create at least 50 colors. So, for this, we need to see how many states we have and create a linearScale for the colors.

```
%%html
<div id="count"></div>
<script type="text/javascript">


d3.json("https://raw.githubusercontent.com/dudaspm/d3plotbook/main/cb_2018_us_county_20m.json")
        .then(function(us) {
            var states = []
            us["features"].forEach(d=> states.push(+d.properties.STATEFP))
            states =  [...new Set(states)];
            d3.select("div#count").text("Number of states in our dataset: "+states.length)

        })
        .catch(function(error){
            console.log(error)
        })

</script>
```

Number of states in our dataset: 52

Looks like we have 52. So, let's use this data to color our map.

```
%%html
<div id="map2"></div>
<script type="text/javascript">


d3.json("https://raw.githubusercontent.com/dudaspm/d3plotbook/main/cb_2018_us_county_20m.json")
        .then(function(us) {
            var width = 600
            var height = 400
            var margin = 0
            // Create the Mercator Projection
            projectionUS = d3.geoAlbersUsa().fitExtent([[margin, margin], [width - margin,
height - margin]], us)
            // Create a function to generate our paths (counties)
            pathGeneratorUS = d3.geoPath().projection(projectionUS)

            var states = []
            us["features"].forEach(d=> states.push(+d.properties.STATEFP))
            states.sort((a, b) => a - b)
            states =  [...new Set(states)];

            color = d3.scaleLinear().range([0,1]).domain([0,states.length-1])
            palette = d3.interpolateSpectral

            const svg = d3.select("div#map2").append("svg")
                .attr("width", width)
                .attr("height", height)

            // construct the element
            svg.selectAll("path")
                .data(us.features)
                .join("path")
                .attr('d', pathGeneratorUS)
                .attr('fill', "white")
                .attr('stroke', d=> palette(color(states.indexOf(+d.properties.STATEFP))))
                .attr('stroke-width', '2')
                .append("title")
                .text(d=>"The StateFP is: "+d.properties.STATEFP)

        })
        .catch(function(error){
            console.log(error)
        })

</script>
```

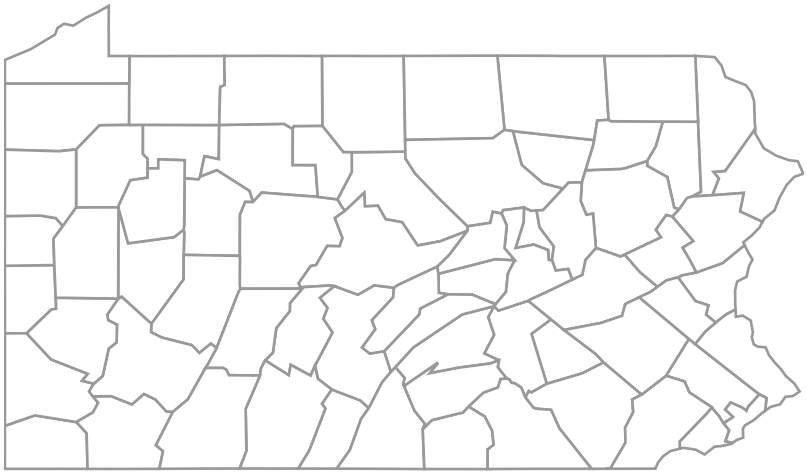


This is a county map for the entire US. Let's focus on just Pennsylvania. To do this, we need to filter to PA's data.

If we take a look at the data. The data contains a stateFP for the FIPS for each state.

For PA it is 42.

```
%%html
<div id="map3"></div>
<script type="text/javascript">


d3.json("https://raw.githubusercontent.com/dudaspm/d3plotbook/main/cb_2018_us_county_20m.json")
        .then(function(us) {
            var width = 600
            var height = 400
            const margin = 0
            // filter data to only PA
            pa = ({type:"FeatureCollection", features:us.features.filter(d=>
d.properties.STATEFP==42)})

            // Create the Mercator Projection
            projectionPA = d3.geoMercator().fitExtent([[margin, margin], [width - margin,
height - margin]], pa)
            // Create a function to generate our paths (counties)
            pathGeneratorPA = d3.geoPath().projection(projectionPA)

            const svg = d3.select("div#map3").append("svg")
                .attr("width", width)
                .attr("height", height)

            // construct the element
            svg.selectAll("path")
                .data(pa.features)
                .join("path")
                .attr('d', pathGeneratorPA)
                .attr('fill', 'white')
                .attr('stroke', '#999999')
                .attr('stroke-width', '2')
                .append("title")
                .text(d=> JSON.stringify(d.properties,null,'\t'))


        })
        .catch(function(error){
            console.log(error)
        })

</script>
```



Let's get some data to add to our map.

Example is from [Los Angeles Times Data and Graphics Department]

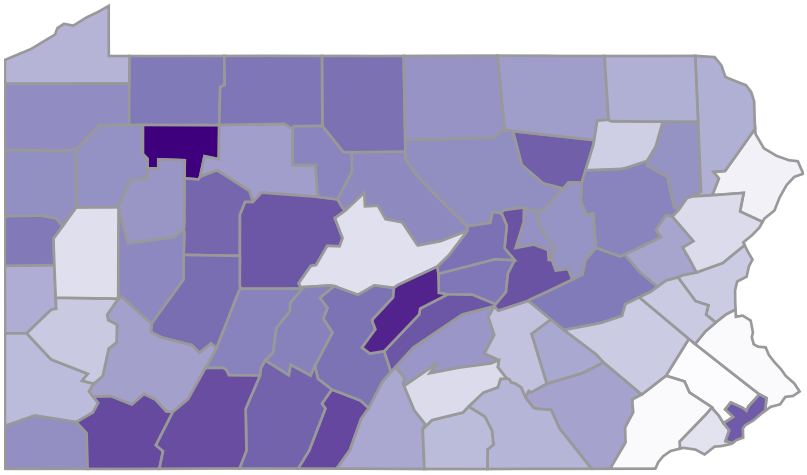
An example of using data from the Data Desk's open-source [census-data-downloader]

```
%%html
<div id="output4"></div>
<script type="text/javascript">

    d3.csv("https://raw.githubusercontent.com/datadesk/census-data-
downloader/613e69f6413d917a6db60186e5ddf253e722dcfd/data/processed/acs5_2017_internet_counties.
csv")
        .then(function(census) {
            d3.select("div#output4").text(JSON.stringify(census[0],null,'\t'))

        })
        .catch(function(error){
            console.log(error)
        })


</script>
```

{ "geoid": "72047", "name": "Corozal Municipio, Puerto Rico", "universe": "11026.0",
"total_with_computer": "6202.0", "dialup_internet": "174.0", "broadband_internet": "4554.0",
"computer_no_internet": "1474.0", "total_no_computer": "4824.0", "county": "047", "state": "72",
"total_with_internet": "4728.0", "total_no_internet": "6298.0" }

So, based on our data we have a state and a county code (like above). Let's use this data to see the percentage of internet access per county in PA.

```
%%html
<div id="map4"></div>
<div id="legend1"></div>
<script type="text/javascript">


d3.json("https://raw.githubusercontent.com/dudaspm/d3plotbook/main/cb_2018_us_county_20m.json")
        .then(function(us) {
        d3.csv("https://raw.githubusercontent.com/datadesk/census-data-
downloader/613e69f6413d917a6db60186e5ddf253e722dcfd/data/processed/acs5_2017_internet_counties.
csv")
            .then(function(census) {
                var width = 600
                var height = 400
                const margin = 0
                // filter data to only PA
                pa = ({type:"FeatureCollection", features:us.features.filter(d=>
d.properties.STATEFP==42)})

                // Adding our census data
                census = census.filter(d=> d.state==42)
                censusObject = {}
                census.forEach(d=> censusObject[d.county] = (+d.total_no_internet/+d.universe))

                // create our color
                palette = d3.interpolatePurples
                color = d3.scaleLinear().range([0,1]).domain(d3.extent(census,d=>
(+d.total_no_internet/+d.universe)))

                // Create the Mercator Projection
                projectionPA = d3.geoMercator().fitExtent([[margin, margin], [width - margin,
height - margin]], pa)
                // Create a function to generate our paths (counties)
                pathGeneratorPA = d3.geoPath().projection(projectionPA)

                const svg = d3.select("div#map4").append("svg")
                    .attr("width", width)
                    .attr("height", height)

                // construct the element
                svg.selectAll("path").append('path')
                    .data(pa.features)
                    .join("path")
                    .attr('d', pathGeneratorPA)
                    .attr('fill', d=> palette(color(censusObject[d.properties.COUNTYFP])))
                    .attr('stroke', '#999999')
                    .attr('stroke-width', '2')
                    .append("title")
                    .text(d=> "Location: "+d.properties.NAME+"\nData:
"+censusObject[d.properties.COUNTYFP])

            })
            .catch(function(error){
                console.log(error)
            })



        })
        .catch(function(error){
            console.log(error)
        })

</script>
```



```
%%html

<script type="text/javascript">

    d3.csv("https://raw.githubusercontent.com/datadesk/census-data-
downloader/613e69f6413d917a6db60186e5ddf253e722dcfd/data/processed/acs5_2017_internet_counties.
csv")
        .then(function(census) {
            var width = 600
            var height = 100
            var margin = 40
            census = census.filter(d=> d.state==42)
            palette = d3.interpolatePurples
            x = d3.scaleLinear().range([margin,width-margin]).domain(d3.extent(census,d=>
(+d.total_no_internet/+d.universe)))

            const svg = d3.select("div#legend1").append("svg")
                .attr("width", width)
                .attr("height", height)


            const xAxis = d3.axisBottom().scale(x)

            svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (height-margin) + ")")
                .call(xAxis)

            svg.append("text")
                .attr("x", width/2)
                .attr("y", height-5)
                .style("text-anchor", "middle")
                .text("No Internet/All Data (%)")


            const num = 20
            const values = d3.range(1,num)

            const coloring = d3.scaleLinear().range([0,1]).domain(d3.extent(values))
            var defs = svg.append("defs")
            var linearGradient = defs.append("linearGradient")
                .attr("id", "linear-gradient1")

            linearGradient.selectAll("stop").data(values).join("stop")
                .attr("offset", d=> d/num)
                .attr("stop-color", d=>palette(coloring(d)) )
            svg.append("rect")
                .attr("x", margin)
                .attr("y", (height-margin)-50)
                .attr("width", (width-margin)-(margin))
                .attr("height", 50)
                .style("fill", "url(#linear-gradient1)")


        })
        .catch(function(error){
            console.log(error)
        })

</script>
```

By Patrick M. Dudas