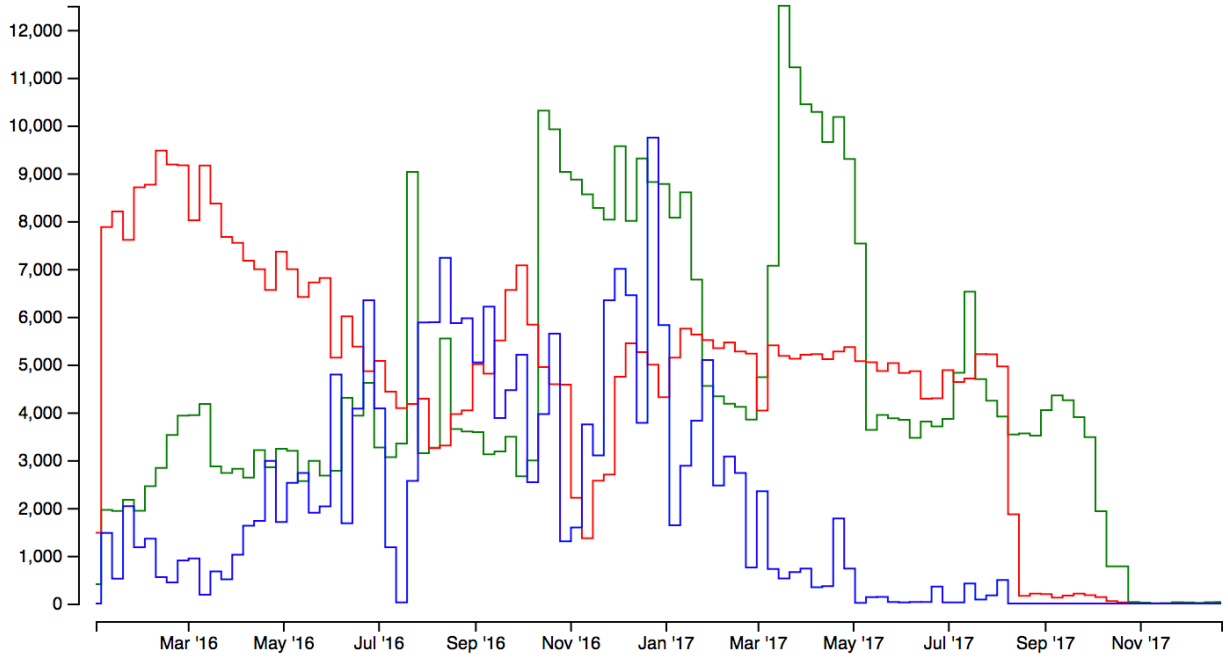
CS/INFO 3300; INFO 5100
Homework 5
Due 11:59pm Monday March 9

Goals: Work with d3.shape generators and time scales

Your work should be in the form of an HTML file called index.html with one <p> element per problem. Wrap any SVG code for each problem in a <svg> element following the <p> element. For this homework we will be using d3.js. **In the <head> section of your file, please import d3 using this tag: <script src="https://d3js.org/d3.v5.min.js"></script>**

Create a zip archive containing your **HTML file and all associated data files** (such as tweets_weekly.json) and upload it to CMS before the deadline. Submissions that do not include data files will be penalized. Your submission will be graded using a Python web server run in the directory containing your zip file contents – be sure that it works.



1. (50pt)
In HW3 you created a series of plots showing a dataset of tweets made by the Internet Research Agency as a part of Russian state-sponsored election interference on Twitter (problems 3A and 3B). FiveThirtyEight wrote an article about the dataset in association with researchers at Clemson and published it on GitHub. In this problem, we'll be using line generators to make more performant versions of the HW3 visualizations.

Unlike in HW3 where you worked with *daily* totals, I have post-processed the data into *weekly* totals for this assignment into a JSON file titled `tweets_weekly.json`. It contains a list of

elements, **each one corresponding to a specific week**. Each element has a specific `Date` and a `Total` number of IRA tweets identified by the researchers. The `date` field is formatted as "`day-month-year`". Additional count data come from tags provided by the researchers. Unlike in the previous assignment, we will not be using the `Number` field.

Your goal in this problem is to **create a line chart showing three different kinds of tweets** (the `NonEnglish`, `NewsFeed`, and `HashtagGamer` data attributes). We will put **time on the X axis** and **activity on the Y axis**. Instead of creating rectangles, you will be making lines. First, **load the data file** using an asynchronous request with `d3.json`. Implement the rest of this problem **inside of the promise function**. Create a **700px wide and 400px tall SVG element** using d3 functions or in your HTML code. Reserve **50 pixels at the bottom and the left side** of the SVG for axis labels. You can add padding to the top and right as you feel necessary.

Begin by **constructing two scales**. For your X axis, construct a `scaleTime()` entity with **appropriate minimum and maximum values** (*hint: you'll need to use* `d3.timeParse` *to create JS date objects for this to work properly*). For the Y axis, we'll be charting the `NonEnglish`, `NewsFeed`, and `HashtagGamer` data attributes as lines. Create a `scaleLinear()` entity that is properly scaled for the **minimum and maximum value of those attributes together** (*hint: the minimum and maximum values could be in any of those three attributes – you have to find out which*). Make sure that you handle the Y coordinates of your SVG canvas appropriately.

Create corresponding **X and Y axis labels for your scales**. They should appear on the **bottom** and **left** sides of the chart. For your Y axis, you can use d3's default values and formatting. For your dates on your X axis, **make the following adjustments** (hint: use this reference page):

- Create a tick for **every 2 months** using the `.ticks()` command
- Using `.tickformat()`, make your labels appear as
  "<abbreviated month name> <year without century>" – e.g. "Mar 17", "May 17", "Jul 17"

You **do not** need to make gridlines.

Now, **create three different lines on your chart** in the form of <path> elements using three different `d3.line()` line generators:

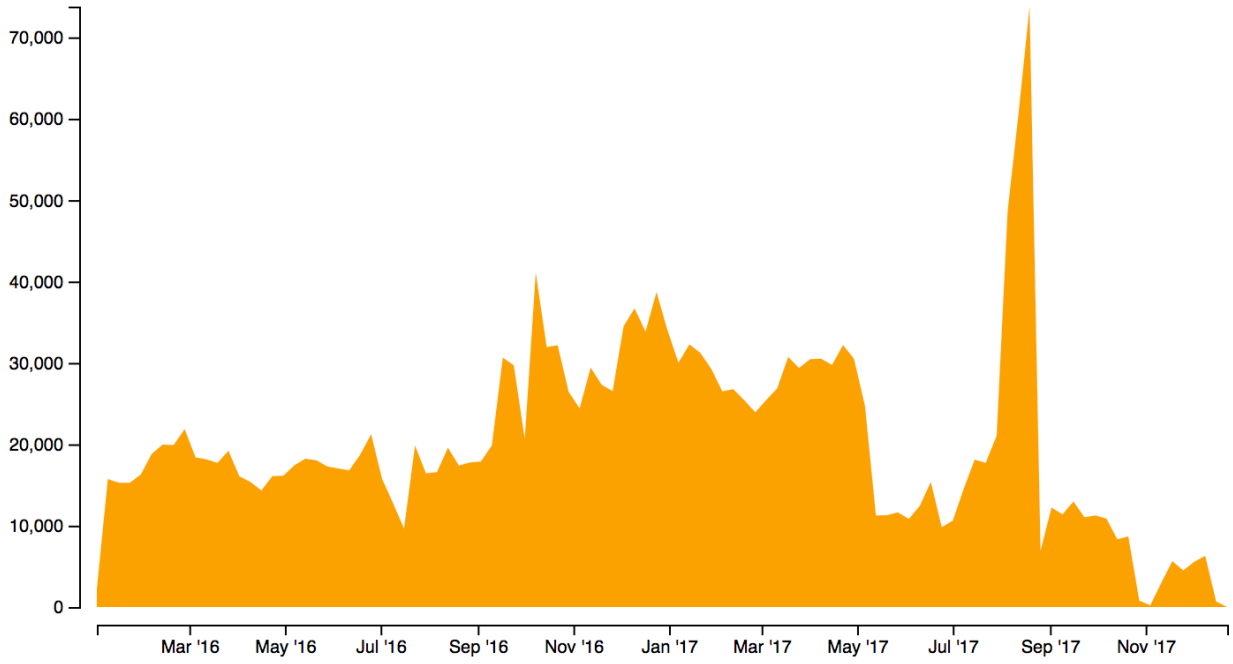
Your first line generator should use your **JS dates for X coordinates** and the `NonEnglish` **tweet values for Y coordinates**. **Apply your scales** to convert from data values into pixel locations. In order to make your line have a stair-step appearance, make use of the `.curve()` function. **Add `.curve(d3.curveStep)` to your line generator's function chain**. **Use your line generator** to fill in the `d` attribute of a new `<path>` element (*hint: be sure to call* `.datum()` *on your dataset*). This line should have a **solid 1px green stroke**.

Your second line generator should use your **JS dates for X coordinates** and the `NewsFeed` **tweet values for Y coordinates**. **Apply your scales** to convert from data values into pixel locations. **Add `.curve(d3.curveStep)` to your line generator's function chain to give a stair-**

step appearance to your line. **Use your line generator** to fill in the `d` attribute of a new `<path>` element. This line should have a **solid 1px red stroke**.

Your third line generator should use your **JS dates for X coordinates** and the **`HashtagGamer` tweet values for Y coordinates**. **Apply your scales** to convert from data values into pixel locations. **Add `.curve(d3.curveStep)` to your line generator's function chain to give a stair-step appearance to your line**. **Use your line generator** to fill in the `d` attribute of a new `<path>` element. This line should have a **solid 1px blue stroke**.

Remember to make **new line generators** and **new `<path>`s for each** of the three attributes.



2. (50pt)
In this problem you will make use of a new d3.shape generator we have not covered in class in order to create a chart similar to the `Total` chart from HW3.

You have already loaded `tweets_weekly.json` using `d3.json()` in problem 1, and gone to the trouble of creating JS date objects. Feel free to **re-use them** in this problem in one `<script>` tag. Everything can be nested within one d3.json promise.

As in problem 1, using either JS or HTML, create an SVG canvas that is **700 pixels wide** and **400 pixels tall** with **50px reserved on the left and bottom for axis labels** (you can add padding to the top and right as you feel necessary). **Make sure to use different variable names from your first chart**. Create a new **`scaleTime()` for your chart X axis**. For your Y axis,

we will **only be showing the `Total` attribute** from the dataset. Create a `scaleLinear()` entity with appropriate minimum and maximum values.

As in problem 1, **create axis labels on the left and bottom sides of the canvas**. You are welcome to **leave their `.ticks()` and `.tickFormat()` at default** or make use of your settings from problem 1. **You do not need to create gridlines**.

For this problem, instead of creating a line chart, **you will create an area chart**. **Refer to the** documentation for `d3.area()`. Creating areas using d3 is very similar to creating lines. You must **first create an area generator**, and then `.call()` that generator on a **<path>** that has been given some data using `.datum()`. Like the line generator, **you will need to tell `d3.area()` how to find its X and Y values**. Unlike line generators, **the area chart has two different y values** (accessed through `.y0()` and `.y1()`), corresponding to the top and bottom parts of the area. It has **one X value**, accessed through `.x()` just as in line generators.

Create an area generator for the `Total` attribute over time. Configure its **X coordinates for dates**. **Configure `y0` so that it is always the minimum value of Total so that it "sticks" to the bottom of the chart** (*hint: you don't need to use* `d => d.Total` *here because it's always the minimum value, but you do need to use your scale to figure out the right pixel location for that data value*). **Configure `y1` to vary in response to `Total`, just as you would do in a line generator**. **Don't use `.curve()`.** Once you've set `.x()`, `.y0()`, and `.y1()`, you're ready to go.

Create a new <path> entity within your SVG canvas. Give it an **orange fill**. Finally, use your area generator to fill in the d attribute for the path (*hint: this works just like line generators do*). If your filled area looks patchy or upside-down, verify that you are setting `y0` correctly.

(note, because the data are aggregated on a weekly basis in the HW5 dataset, the spike in 2016 that was visible in HW3 is hidden here)

Bonus (not for extra credit):

In problem #1, you created 3 separate line generators and 3 separate .datum() calls. You can actually complete the problem with only 1 line generator and 1 call to .data()! This requires two things: reconstructing a new dataset and building a .data join instead of a .datum join.

In our current dataset, every element in the array contains a date and values for all of the lines. If you want to do a data join to make all 3 lines at once, then you need to have a different structure. At the top level, you need one element for each of the lines, so that your data join can create a <path> for each one. Within each of those "super" elements, you need the list of points that go in the line so that the line generator knows what to do.

For example:

```
[{'Date':'01-2020', 'MiamiTotal':33, 'IthacaTotal':44},
 {'Date':'02-2020', 'MiamiTotal':20, 'IthacaTotal':52},
 {'Date':'02-2020', 'MiamiTotal':10, 'IthacaTotal':62}]
```

should become…

```
[{'City':'Miami', 'Values':[{'Date':'01-2020', 'Total':33},
                            {'Date':'02-2020', 'Total':20},
                            {'Date':'03-2020', 'Total':10}] },
 {'City':'Ithaca', 'Values':[{'Date':'01-2020', 'Total':44},
                             {'Date':'02-2020', 'Total':52},
                             {'Date':'03-2020', 'Total':62}] } ]
```

…so that your data join can make a <path> for city=Miami and city=Ithaca.

The second part will be to adjust how you define the D attribute for the <path>. While in our class example things worked fine because our data were stored in a flat list, here we've got data stored within 'Values' for a given element in the list. You've got to call the line generator slightly differently as a result.

Once you get D figured out, it's a simple matter of executing the .join() and using a scaleOrdinal to color by the 'City' attribute.