DDA 2003/MDS 6112 Assignment 4

Start: March 16 at 9:00 AM

End: April 2 at 11:59 PM

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

1 Description

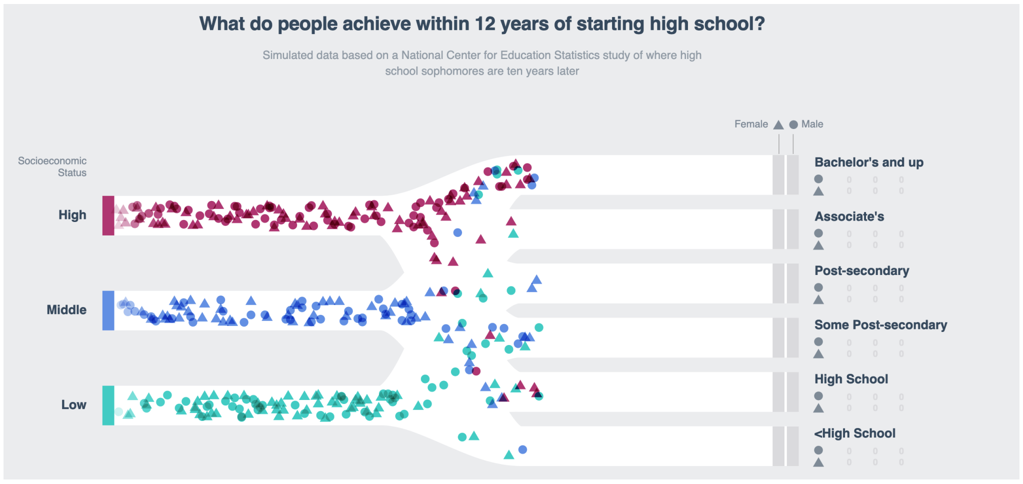
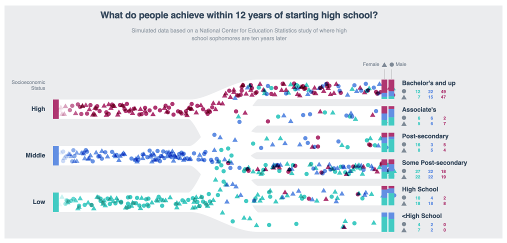


Figure 1: The visual result of assignment 4.

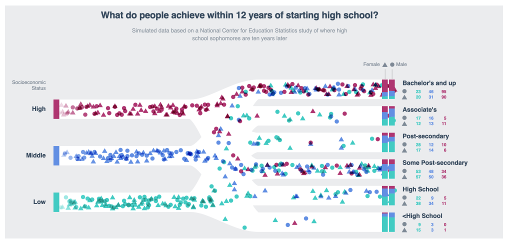
This assignment aims to help students get familiar with animated high-dimensional data visualization using d3.sankey.

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

- Given a high-dimensional data set in a .json file, visualize the data using a Sankey diagram, as shown in Figure 1.
- Visualize two categories into different shapes.
- Using stacked bar charts to visualize and show the statics, as shown in Figure 1 at the right part.



(a)



(b)

Figure 2: Animation of high-dimensional data visualization.

- Create an animation. Two frames are shown in Figure 2

For the complete animation, please refer to the provided video.

2 Requirement

- Access data.
- Stack probabilities.
- Create person.
- Visualize paths.
- Visualize persons.
- Add color and filter.
- Visualize bars.
- Update numbers.

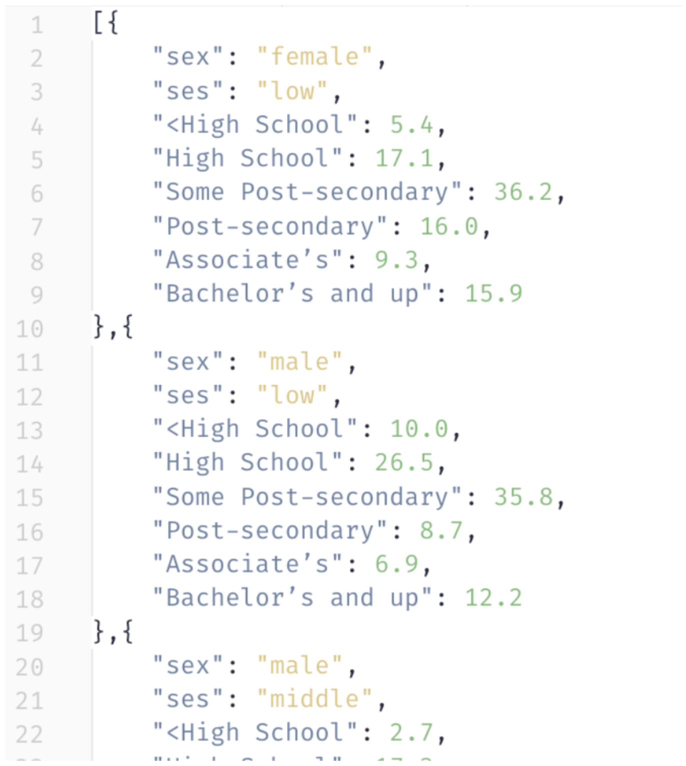
Online Resources The following online resources will be helpful in finishing this assignment.

- SVG in D3
- Scale in D3
- Selection in D3
- Sankey in D3
- Timer in D3

3 Instruction

3.1 Access variables

Figure 3 shows the data we used in this assignment. Each object represents a possible starting point, specified by a sex and a socioeconomic status (ses). The object also has the percentage of people in that starting group that attained (at most) various degrees. Here, you need to define the data accessor, data category, and data range for education and socioeconomic status. The sex variable has been defined in the .js file. You are asked to define the remaining two variables.



```json
[{
    "sex": "female",
    "ses": "low",
    "<High School": 5.4,
    "High School": 17.1,
    "Some Post-secondary": 36.2,
    "Post-secondary": 16.0,
    "Associate’s": 9.3,
    "Bachelor’s and up": 15.9
},{
    "sex": "male",
    "ses": "low",
    "<High School": 10.0,
    "High School": 26.5,
    "Some Post-secondary": 35.8,
    "Post-secondary": 8.7,
    "Associate’s": 6.9,
    "Bachelor’s and up": 12.2
},{
    "sex": "male",
    "ses": "middle",
    "<High School": 2.7,
```

Figure 3: The high-dimensional data.

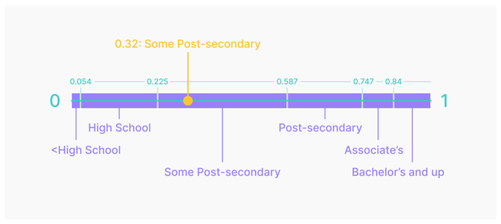
3.2 Stack probabilities

Since we need to create several people per second, we want our generatePerson() function to be as simple as possible. Given their sex and socioeconomic status, we know the probability of a person falling into one of our education "buckets". Since these probabilities sum up to 100%, we can stack these probabilities and use a random number to assign a person to a bucket. Here we take the probabilities for females who grew up in a low-income household as an example.

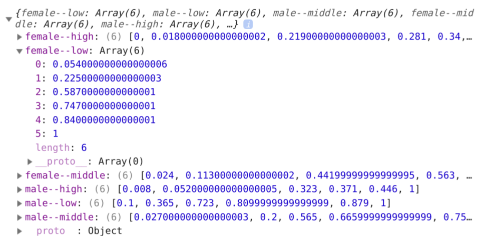
We can take these probabilities and stack them on top of each other so that each level instead represents the probability that a person achieves that level or lower. The highest level (Bachelor's and up), will get the number 1 because there is a 100% chance that a person achieved that level or lower.

When we have these stacked probabilities, we can choose a random number between 0 and 1, which we will locate on the number line. For example, if we choose the number 0.32, this person will be placed in the Some Post-secondary education bucket, sketched in Figure 4(a).

In this part, you need to generate an empty stackedProbabilities object to populate. You will loop over each of the starting points in the data set, generating the status key and instantiating a stackedProbability number. Next, you loop over each of the education buckets (in order), adding the current probability to the stacked probability, then returning the current sum. Note that you need to add an additional check – if we are looking at the last education bucket, we will return 1 instead of the running sum. This will help account for rounding errors, where the sum is 0.99 and does not completely add up to 1. The output of stackedProbabilities is shown in Figure 4(b).



(a) probability diagram



```
▼{female--low: Array(6), male--low: Array(6), male--middle: Array(6), female--mid
 dle: Array(6), male--high: Array(6), …} i
  ▶female--high: (6) [0, 0.018000000000000002, 0.21900000000000003, 0.281, 0.34,…
  ▼female--low: Array(6)
     0: 0.054000000000000006
     1: 0.22500000000000003
     2: 0.5870000000000001
     3: 0.7470000000000001
     4: 0.8400000000000001
     5: 1
     length: 6
   ▶__proto__: Array(0)
  ▶female--middle: (6) [0.024, 0.11300000000000002, 0.44199999999999995, 0.563, …
  ▶male--high: (6) [0.008, 0.052000000000000005, 0.323, 0.371, 0.446, 1]
  ▶male--low: (6) [0.1, 0.365, 0.723, 0.8099999999999999, 0.879, 1]
  ▶male--middle: (6) [0.027000000000000003, 0.2, 0.565, 0.6659999999999999, 0.75…
  ▶ proto  : Object
```

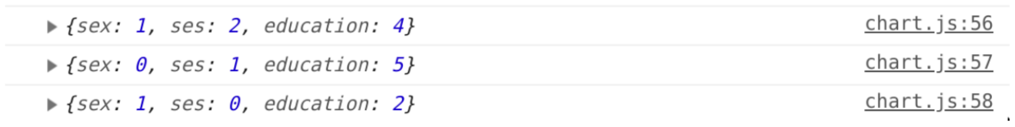
(b) probability object structure

Figure 4: Illustration of stacking probabilities.

3.3 Create person

In this part, you are asked to put the stacked probabilities to use and create a generatePerson() function. Here we want this function to return an object with a sex, ses, and education. If we peek at the bottom of the chart.js file, we wil see a few utility functions that will help us with some dirty work. For example, there is a getRandomValue() function that takes an array of values and returns a random value. The procedure is as follows:

- Generate a statusKey and grab the matching stacked probabilities.
- Generate a random number using Math.random() and use d3.bisect() to find the index where that number will "fit" in the probabilities array.



```
▶ {sex: 1, ses: 2, education: 4}                                    chart.js:56
▶ {sex: 0, ses: 1, education: 5}                                    chart.js:57
▶ {sex: 1, ses: 0, education: 2}                                    chart.js:58
```

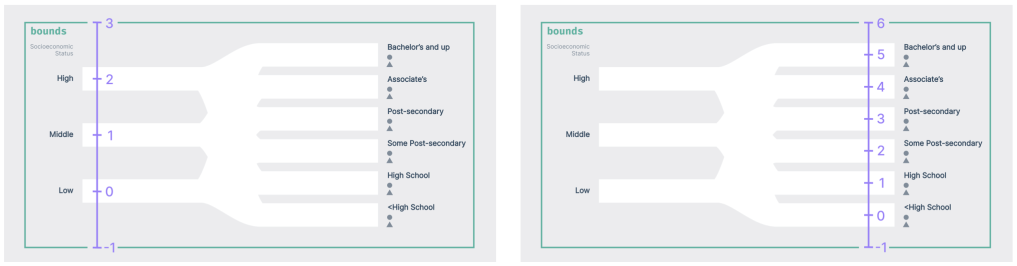
Figure 5: Output of generatePerson() function

Figure 5 shows an output of generatePerson function.

3.4 Draw path

First of all, you need to create some scales in the Create scales step. Create an x scale that converts a person's progress (from left to right) into an x-position and represents this progress with a number from 0 (not started yet) to 1 (has reached the right side). Since we do not want the markers moving beyond the left or right side of the paths, so you can use .clamp() this scale. This way, the smallest number our scale will return is 0 and the largest number is 1, even if we give it a progress of 10.

Next, you will create a scale that will convert from a socioeconomic id to a y-position. We want these paths to be evenly spaced between the bounds, but to still fit inside. You can achieve this by padding the scale's domain, making it span $[-1, 3]$ instead of $[0, 2]$. This way, the real socioeconomic status ids will fit inside the bounds.



(a) y scale left (b) y-scale right

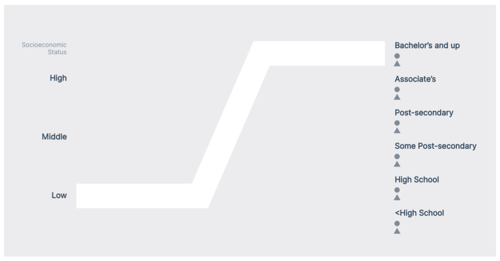
Figure 6: Y scale diagram.

You will also want the scale's domain to be backwards: $[3, -1]$ instead of $[-1, 3]$ because we want the highest y position (closer to the bottom) to correspond to the lowest id. An example is shown in Figure 6.

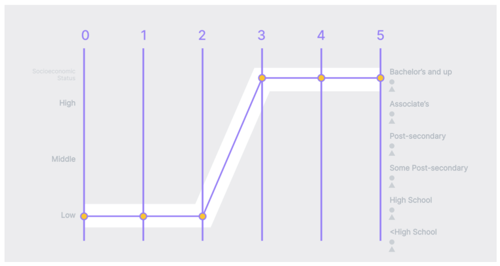
Next, you will use the scales to draw some paths. The goal is to draw a path between each of the starting points and each of the ending points, shown in Figure 7 (a). Since these shapes are not linear, we will draw them using path by using d3.line() to create a string attribute generator. The line generator will take an array of six identical arrays. The first item in each of these arrays (0) is the socioeconomic status id (starting point) and the second item (5) is the education id (ending point). The link generator will return the starting y position for the first 3 arrays, and the ending position for the last 3 arrays. The reason we want to repeat this array 6 times is to devote one-fifth of the horizontal space to the y-position transition. If we only had four identical arrays, we would be devoting half of the horizontal space to the transition, which would make the final chart way more chaotic. The markers would spend one third of their time moving up and down. An example is shown in Figure 7(b).

You will need to create that six-item array for each permutation of starting and ending ids. Map over each of the starting ids and also each of the ending ids, creating that six-item array for each loop. Pass the result to LinkOptions (using d3.merge()), which will flatten these into one array.

Finally, the output of linkOptions is shown in Figure 8. Note that you need to add an interpolator function to line generator to smooth the paths. (i.e., .curve(d3.curveMonotonX)).



(a) Path example



(b) Path parts diagram

Figure 7: Example path.

3.5 Visualize persons

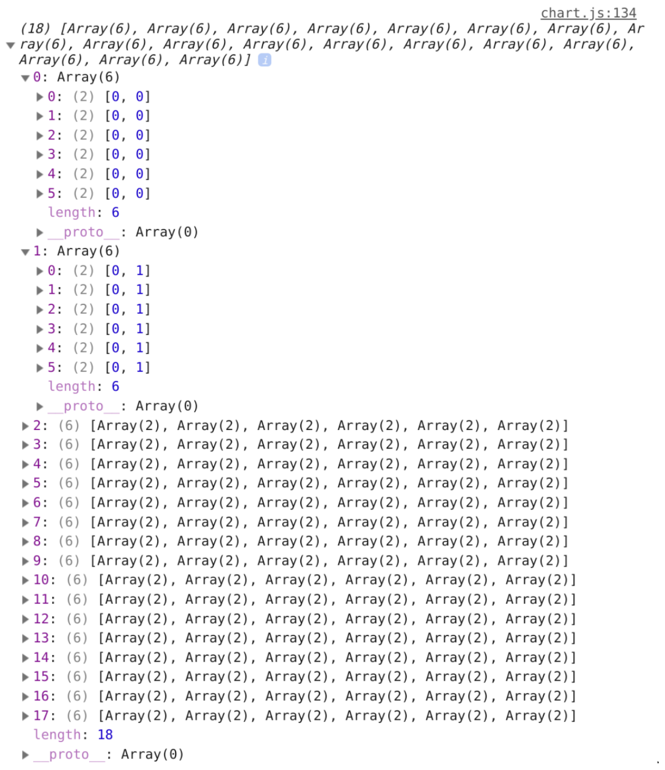
In this part, you need to create a function called updateMarkers() that will draw people. You will use d3.timer() to update the position of people. d3.timer()'s first parameter is a callback function that it will call until the timer is stopped (which we can do the timer's .stop() method). The callback function will have access to one parameter: how many milliseconds have elapsed since the timer started. Then you need to add a new person to the people array every time updateMarkers() runs.

Then, starting with females, you can draw a triangle for every person in the array. You can isolate these people by .filter() the people array with a sex of 0. Similarly, you can draw male markers, then update all of the markers' positions at once. Note that the generatePerson() function needs to record when that person was created beside the regular information.

3.6 Add color and filter

Once the simulation has been running for a while, we will have a lot of elements on the screen at once, which can get expensive. So you need clean up markers that have finished their journey by adding another filter rule when you create the female and male markers. To achieve that, you need to remove any markers that have completed their journey.

Once the markers make it to the right side, their sex is still clear (based on their shape), but it's not clear what socioeconomic status they started in. So you need to add a color scheme that helps distinguish the markers. You need to create a color scale at the end of Create scales step. You can use a



```
                                                        chart.js:134
(18) [Array(6), Array(6), Array(6), Array(6), Array(6), Array(6), Array(6), Ar
▼ray(6), Array(6), Array(6), Array(6), Array(6), Array(6), Array(6), Array(6),
Array(6), Array(6), Array(6)] i
 ▼0: Array(6)
   ▶0: (2) [0, 0]
   ▶1: (2) [0, 0]
   ▶2: (2) [0, 0]
   ▶3: (2) [0, 0]
   ▶4: (2) [0, 0]
   ▶5: (2) [0, 0]
    length: 6
   ▶__proto__: Array(0)
 ▼1: Array(6)
   ▶0: (2) [0, 1]
   ▶1: (2) [0, 1]
   ▶2: (2) [0, 1]
   ▶3: (2) [0, 1]
   ▶4: (2) [0, 1]
   ▶5: (2) [0, 1]
    length: 6
   ▶__proto__: Array(0)
 ▶2: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶3: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶4: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶5: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶6: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶7: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶8: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶9: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶10: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶11: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶12: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶13: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶14: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶15: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶16: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
 ▶17: (6) [Array(2), Array(2), Array(2), Array(2), Array(2), Array(2)]
  length: 18
 ▶__proto__: Array(0)
```

Figure 8: Output of linkOptions.

linear scale that interpolates between two colors – by setting the domain to an array of the sesIds, then you will get three unique, equally-spaced colors. One interpolator that can be used is d3.interpolateHcl.

You may notice how the markers overlap the start bars, once the markers reach the right side? This is because the existing people are being recycled and already have an opacity of 0.

To address this, you need to give each person a unique id when we create them. Each time we call generatePerson(), we will increment the current-PersonId variable that we'll use as an id. This way, each person's id will be distinct.

When you create the female and male markers in the updateMarkers() function, you can tell D3 how to distinguish people from one another. D3 selection objects' .data() method takes a second parameter: a key accessor.

This key accessor defaults to the element's index – this explains why the markers were being recycled: items in the filtered array are removed once they reach the end of their journey, and new elements end up in the same position in the filtered list. Instead, you can set the key accessor functions to return a person's id, guaranteeing they are not recycled.

3.7 Visualize bars

In this part, you are asked to draw the ending bars. At the end of updateMarkers() function, you need to create an array of people who have finished their journey and fit inside of that education bucket. We will want to draw one bar per path: each permutation of sex, socioeconomic status, and educational attainment. To achieve this, you need to create a flattened array, with one object per permutation that contains the starting and ending positions, the count, the percentage above, and the total count in the bar. Now that you have an array for each of the ending bars, you can create one rect per bar. Now once the markers finish their journey, you will see their colors populate the bars on the right. After the simulation has been running for a while, the stacked bars should start to level out, approximating the percentages in the data set.

3.8 Update numbers

Show the exact counts underneath the labels for each of our path endings, next to the female and male markers.

4 Evaluation

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

1. Access education attributes and socioeconomic status. (5 pts)
2. Stack probabilities. (10 pts)
3. Create person. (5 pts)
4. Visualize path. (15 pts)

5. Visualize persons. (20 pts)
6. Add color and filter. (20 pts)
7. Visualize bars. (10 pts)
8. Update numbers (10 pts)
9. Submission (5pts). Please compress your code and a readme file (optional) into a zip file and submit the zip file to Black Board. The readme file can include descriptions that help the grader run the interface successfully.

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