STATS 401 Lab 9 Tutorial: D3.js Part Three

In this lab, we will learn to (1) visualize data with the chord diagram and (2) create a line chart with data from a .csv file in D3.

Note:

It is very likely that you will need Mozilla Firefox  *for the second task today if you don't know how to solve the "cross-origin request" problem with your default browser. Therefore, you may want to start downloading Firefox in advance if you don't have one.*

Task 1. Visualize data with the chord diagram

Chord diagrams visualize links (or flows) between a group of nodes, where each flow has a numeric value (figure 1). For example, they can show migration flows between countries. The data needs to be in the form of an *n* by *n* matrix, where *n* is the number of items. The first row of the matrix represents flows from the 1st item to the 1st, 2nd, 3rd items, etc.

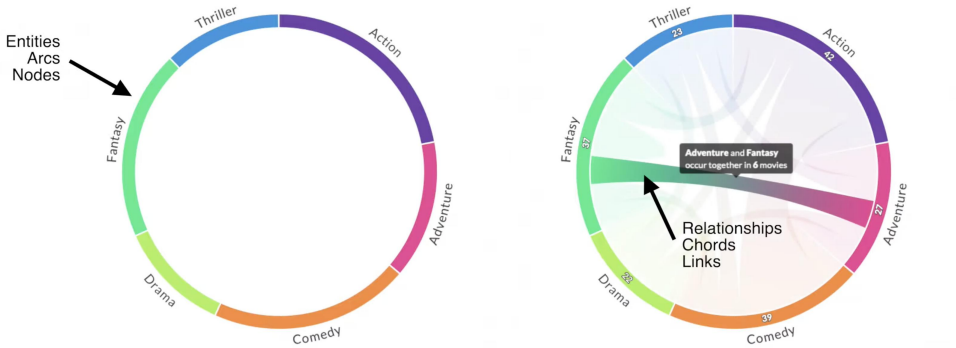

Figure 1. *Chord diagram introduction*

In this task, we will make a chord diagram with four entities in d3 with the mock data matrix below:

```
[[    0,  5871, 8916,  2868],
 [ 1951,     0, 2060,  6171],
 [ 8010, 16145,    0,  3045],
 [ 1013,   990,  940,  1540]];
```

Step 0. Set up

Open the project folder for Lab 7 and create a new file named `chord.js`. Go to `index.html` and changed the script source to `chord.js`. Open the `.html` file in your browser. Don't forget to change the webpage title to "Lab 9".

Step 1. Input data and assign a color for each group

Create a matrix variable containing the relationship information between the four entities and assign colors to each entity.

```
var matrix = [
  [    0,  5871, 8916,  2868],
  [ 1951,     0, 2060,  6171],
  [ 8010, 16145,    0,  3045],
  [ 1013,   990,  940,  1540]
];

var colors = [ "#f2808f", "#f2e380", "#c8f280", "#808ff2"];

```

Step 2. Create the svg variable

New an svg variable and use the translate function to locate it at point (400, 380) in the coordinate plane.

```
var svg = d3.select("body")
            .append("svg")
            .attr("width", 800)
            .attr("height", 800)
            .append("g")
            .attr("transform", "translate(400,380)")
```

Step 3. Calculate data information for drawing the arcs and ribbons

Give the data matrix to `d3.chord()` and use this function to calculate all the information we need to configure the chord. There are several parameters to play around with during this process:

`.padAngle()` sets the angle between adjacent groups in radians
`.sortGroups()` specifies the order of the groups
`.sortSubgroups()` sorts within each group
`.sortChords()` determines the z-order of the chords

```
var res = d3.chord()
            .padAngle(0.05)
            .sortSubgroups(d3.descending)
            (matrix)

```

Step 4. Draw the entities/arcs

Draw the entities with the information generated in step 3 with the code fragment below. You can try to change the sequence, color, and size of the entities.

```
21
22  svg.datum(res)
23     .append("g")
24     .selectAll("g")
25     .data(function(d) { return d.groups; })
26     .enter()
27     .append("g")
28     .append("path")
29     .style("fill", function(d,i){ return colors[i] })
30     .style("stroke", "none")
31     .attr("d", d3.arc()
32     .innerRadius(300)
33     .outerRadius(310));
34
```

The expected outcome is shown in figure 2.

Figure 2. *The entities/arcs of the chord diagram*

Step 5. Draw the links

The last and the most important step is drawing the relationship between each entity with the function `d3.ribbon()`.

```
35  svg.datum(res)
36     .append("g")
37     .selectAll("path")
38     .data(function(d) { return d; })
39     .enter()
40     .append("path")
41     .attr("d", d3.ribbon().radius(300))
42     .style("fill", function(d){ return(colors[d.source.index]) })
43     .style('opacity',0.6)
44     .style("stroke", "none");
45
```

You should get something similar to the diagram in figure 3.

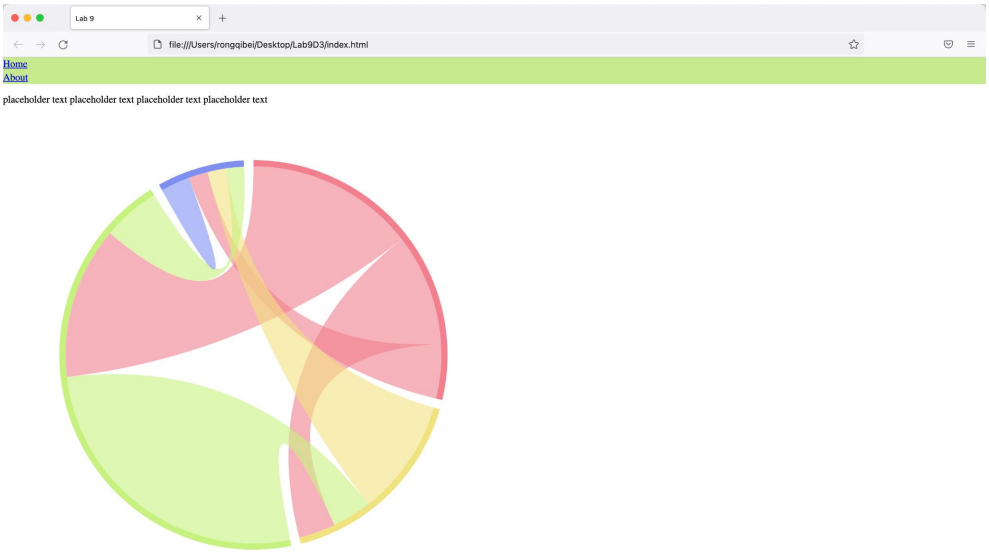

Figure 3. *The expected outcome of task 1*

Comparing this generated diagram and our data source, do you understand how the chord diagram works?

If yes, let's try to think about what the chord diagram should be like with the below data source in your mind (something that requires reverse thinking).

```
matrix = [
  [  50,  100, 100,  100],
  [ 100,    0,   0,  100],
  [ 100,    0,   0,    0],
  [ 100,   50,   0,    0]
];
```

You can try to check the result with your code. Did you get it right?

Task 2: Make a line chart with data stored in the .csv file

In previous labs, we assigned the cleaned data to variables directly in the `.js` file. However, in real-life practices, it is more likely that our data is stored in other formats, such as `.csv`. In this task, we are going to read date and price data from a raw dataset and create a line chart to visualize the linear relationship between date and price.

Step 0. Set up.

Since we are going to read your local `.csv` files, we will encounter the cross-origin request problem. You can find different ways to deal with this problem for different browsers (e.g., Chrome, Safari) online. I use Firefox for this task here since I believe the solution for Firefox is the easiest and fastest one. For Chrome, you can refer to the following link: https://stackoverflow.com/questions/3102819/disable-same-origin-policy-in-chrome

***Also, there is one easier way to solve the issue. You can just go to the Command Line in your system (windows, Linux or MacOs) and then go to the directory of this Lab. Then just directly type in "python3 -m http.server 4000" and restart the browser. You can find out that the issue is fixed.**

To handle this problem, let's first go to the setting page by going in about:config and accepting the risk (figure 4).

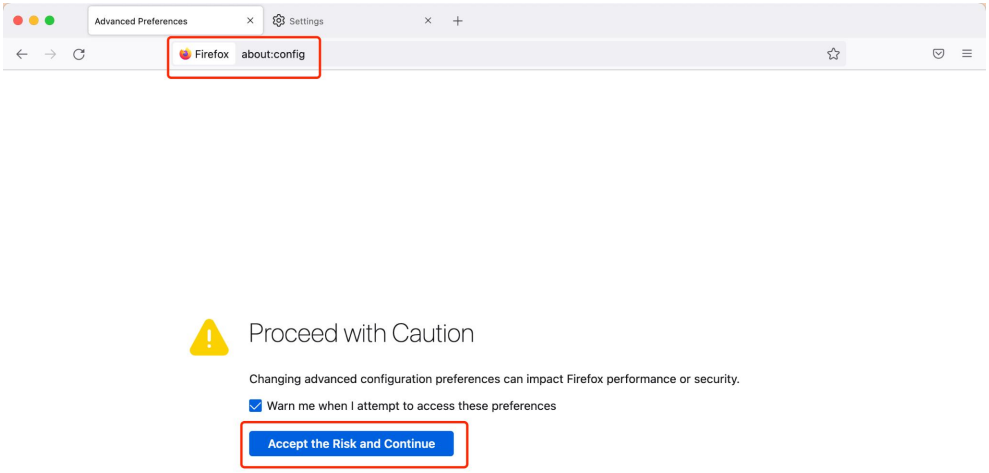

Figure 4. *Go to about:config and accept the risk to continue*

Then, search for security.fileuri.strict_origin_policy and turn the setting here to "false" (figure 5).

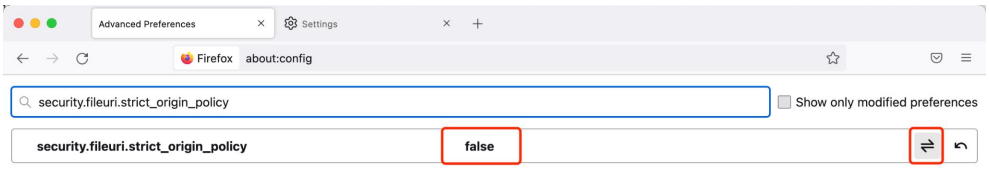

Figure 5. *Search for a security setting and turn it to false*

Lastly, quit your browser and start it again.

After finishing this setting, download the price_data.csv file from Sakai and upload it to the project folder. Also, create a new `.js` file called `price.js` and change the script source to `price.js` in the `.html` file. You can inspect and revise the data from the Atom/VSCode editor as shown in figure 6.

This data source file contains two data columns and 100 rows. The first column provides date data in the format of mm/dd/yyyy. The other column shows the corresponding price in string. This is a mock dataset generated by Mockaroo.

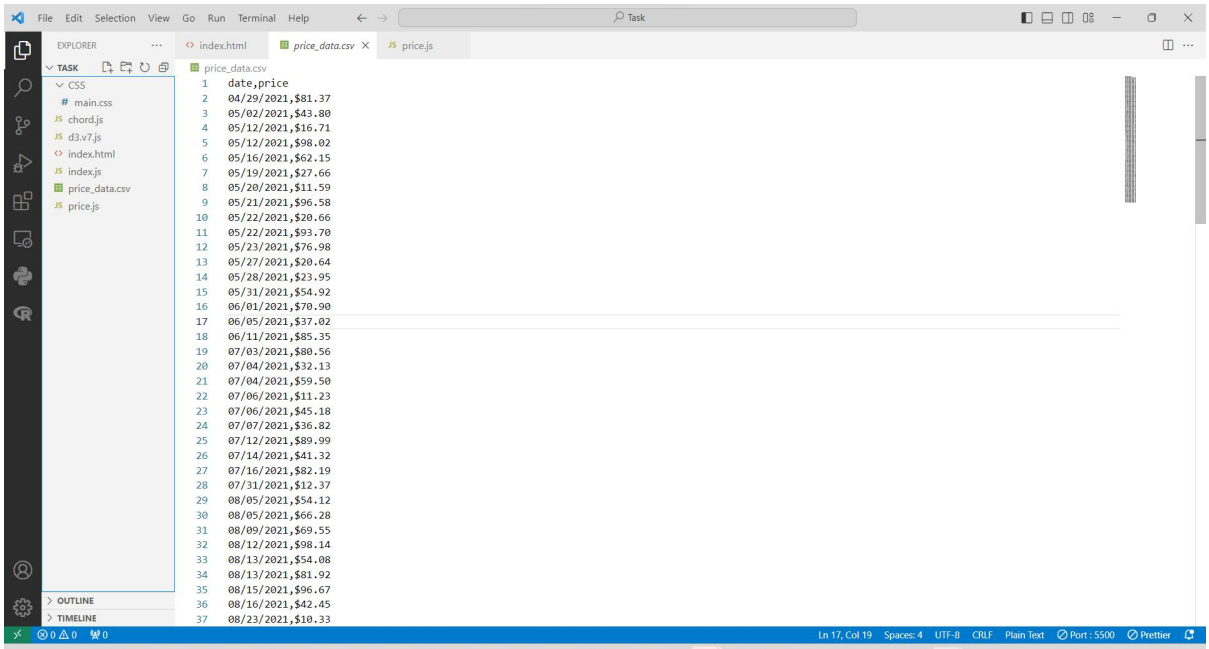

Figure 6. *import the .csv file in VS Code*

Step 1. Inspect the raw data in your browser

In your `price.js` file, write the below code to load data and inspect the loading result from your browser's console.

```
d3.csv("price_data.csv", function(d) {
  console.log("Raw data:", d);
}).catch(function(error) {
    console.error("Error loading CSV:", error);
  });

```

Here, the `error` in the first row allows you to see an error message if the `d3.csv` function cannot understand what is getting from the file (e.g., something that is broken).

Since we have already solved the cross-origin request problem, you should successfully see a data array in the console (figure 7). The array contains 100 items, each with a date value and a price value. You can click on each item, expand it, and check more detailed information.

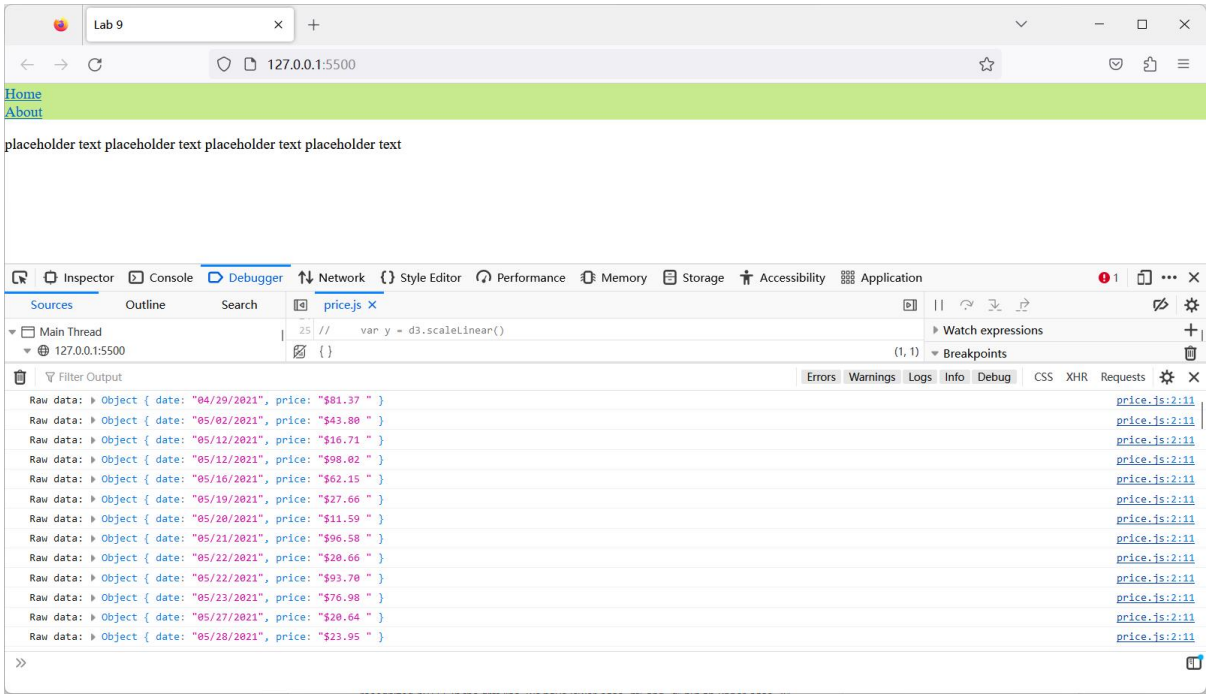

Figure 7. *Inspecting raw dataset in console*

Step 2. Clean the raw data into the format that can be utilized by us

First, let's use the `timeParse()` function to clean the date value to the date/time data type recognized by D3. In the first line, we have lower-case "m" and "d" but an upper-case "Y" because the month and day only take up two digits each but the year takes up four digits.

Second, let's remove the dollar sign in front of the price value with the replace`()` and `slice()` function.

```
var parseDate = d3.timeParse("%m/%d/%Y");

d3.csv("price_data.csv", function(d) {
  console.log("Raw data:", d);
  return {
    date: parseDate(d.date),
    price: Number(d.price.replace(/\$/g, '').trim())
  };
});
```

Use the console to check your cleaned data. Did you get a similar outcome to figure 8?

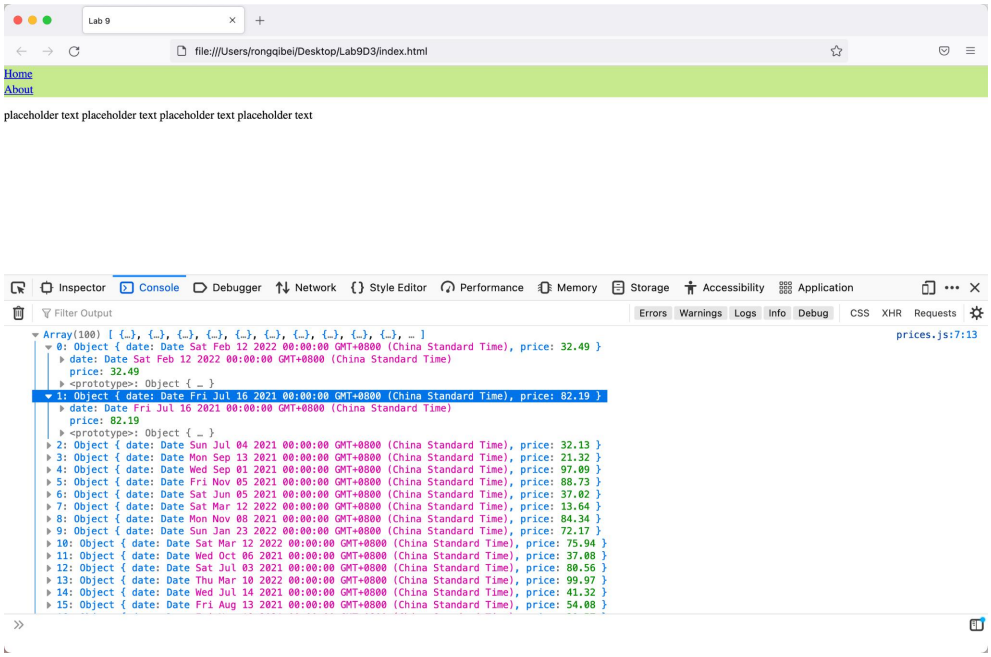

Figure 8. *The expected data cleaning outcome*

Step 3. Create variables for the x-axis and y-axis

First, let's mark out the endpoints on the axes: the minimum date value (the starting date), the maximum date value (the ending date), and the maximum price.

```
10  .then(function(data) {
11    var height = 600;
12    var width = 1200;
13    var maxDate = d3.max(data, function(d) { return d.date; });
14    var minDate = d3.min(data, function(d) { return d.date; });
15    var maxPrice = d3.max(data, function(d) { return d.price; });
16    console.log(maxDate);
17    console.log(minDate);
18    console.log(maxPrice);
19
20  });
21
```

To confirm that we successfully obtained these values, refresh your browser and check the console. Did you get a similar outcome to the results shown in figure 9?

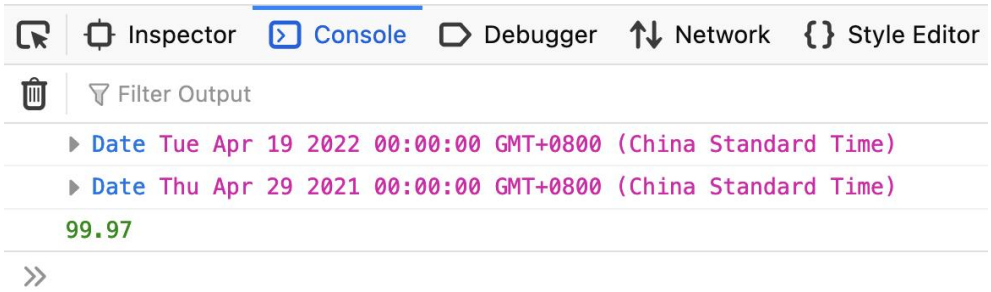

Figure 9. *Endpoints for drawing axises*

Use the code below to create the x-axis and y-axis. Note that we set the starting point of the price to zero. We put the y-axis on the left side and the x-axis at the bottom.

```
var y = d3.scaleLinear()
          .domain([0, maxPrice])
          .range([height, 0]);

var x = d3.scaleTime()
          .domain([minDate, maxDate])
          .range([0, width]);

var yAxis = d3.axisLeft(y);
var xAxis = d3.axisBottom(x);

```

Step 4. Draw the line

Now we finally come to the stage of drawing lines. Follow the code below and refresh your browser, what did you get?

```
var svg = d3.select('body').append('svg')
            .attr('height', '100%')
            .attr('width', '100%');

var chartGroup = svg.append('g')
                    .attr('transform', 'translate(50, 50)');

var line = d3.line()
             .x(function(d) { return x(d.date); })
             .y(function(d) { return y(d.price); });

chartGroup.append('path')
          .attr('d', line(data));

});
```

Unfortunately, we get nothing but a messy graph (figure 10). Two mistakes are needed to be corrected to turn this messy graph into a line chart.

Figure 10. *The first attempt at drawing the line chart*

You may have identified the first problem - the areas surrounded by those lines are filled with black. To correct this problem, please go to the stylesheet `.css` file and set the filling option to be none.

```
8   path{
9     stroke: #b08dea;
10    stroke-width: 2px;
11    fill: none;
12  }
```

Now, refresh the browser again. We perceive this graph as more of a composition of lines after removing the filing (figure 11). However, this is still far from a line chart.

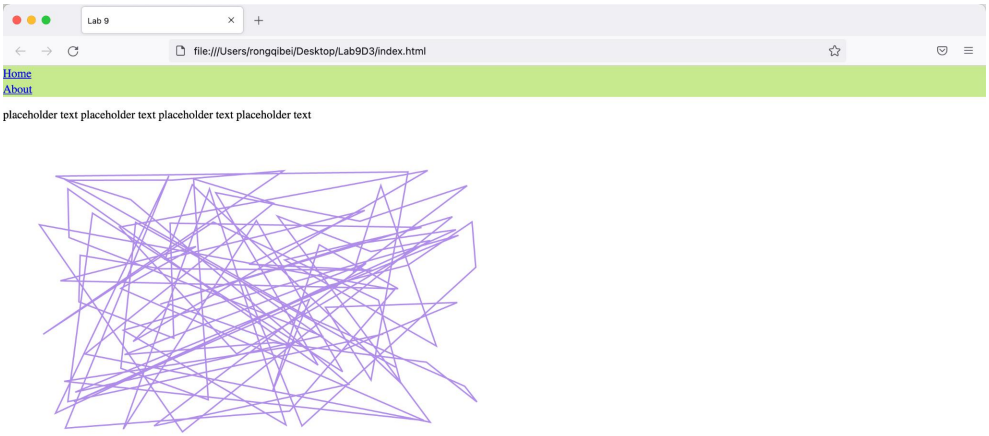

Figure 11. *The second attempt at drawing the line chart*

You may have also discovered the second problem: the data in the `.csv` file is not sorted! The line went back and forth because it needs to connect data points presented in a random sequence in the unsorted file.

Please sort the raw data based on **"date"** in Excel ascendingly (from past to now) and substitute the unsorted file with the sorted one. Refresh the browser again, you should finally get a chart (figure 12) visualizing a linear relationship.

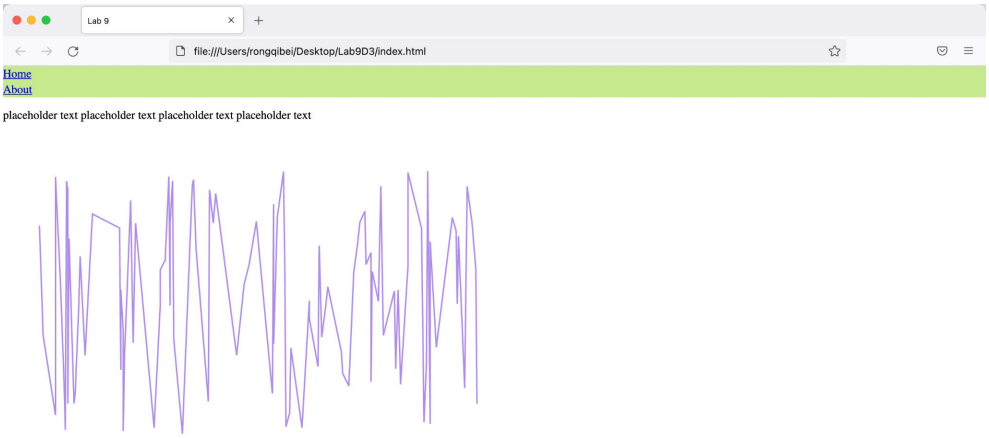

Figure 12. *The linear relationship between date and price*

Step 5. Add axes to the graph

Add the x-axis and y-axis to the line chart by adding the below lines of code.

```
42      chartGroup.append('path')
43              .attr('d', line(data));
44
45      chartGroup.append('g')
46              .attr('class', 'x axis')
47              .call(xAxis);
48
49      chartGroup.append('g')
50              .attr('class', 'y axis')
51              .call(yAxis);})
```

However, the x-axis seems to be higher than expected (figure 13).

Figure 13. *Line chart with an extremely high x-axis*

To solve this problem, please apply the translate function by adding the below lines of code.

```
42      chartGroup.append('path')
43              .attr('d', line(data));
44
45      chartGroup.append('g')
46              .attr('class', 'x axis')
47              .attr('transform', 'translate(0, ' + height + ')')
48              .call(xAxis);
49
50      chartGroup.append('g')
51              .attr('class', 'y axis')
52              .call(yAxis);})
```

Then, we finally get a line chart with proper axes (figure 14). I also slightly adjusted the ratio of the graph to make it aesthetically better (figure 15).

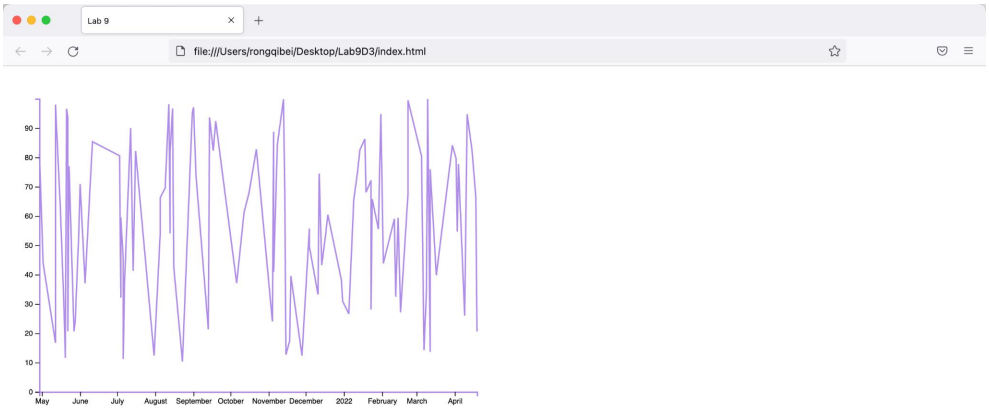

Figure 14. *The expected outcome of task 2 (without ration adjustment)*

Figure 15. *The expected outcome of task 2*