Lab 5: D3 Selections and Grouping Activities

Aggarwal, Mallika edited this page 3 days ago · 3 revisions

D3 Selections And Grouping (Activities)

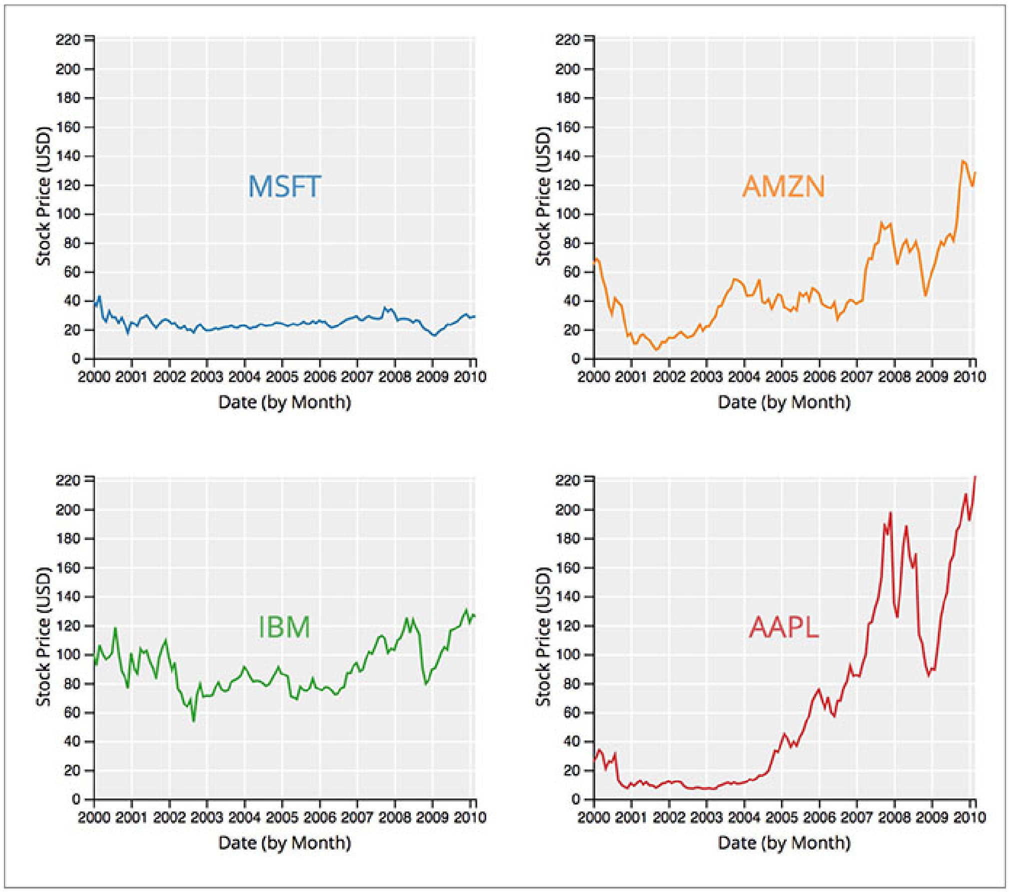
Activity 0 - Nesting a dataset

> Reminder: Start an http server for this lab's directory. From command line call `python -m SimpleHTTPServer 8080` (for Python 2) or `python -m http.server 8080` (for Python 3).

During today's activities you will be working with the `stock_prices.csv` dataset. The dataset includes 492 rows. Each row corresponds to the closing stock price at the end of each month for a company. Here is a snippet of the data table:

company	date	price
MSFT	Jan 2000	39.81
MSFT	Feb 2000	36.35
AMZN	Jun 2006	38.68
AAPL	Jul 2009	163.39

You will be working toward creating the following trellis line chart today:



Clone this wiki locally

`https://github.gatech.edu`

For this activity you will use the D3 you have learned so far - e.g. d3 data-binding, axes, scales, and appending new elements. You are also going to use new concepts to create grouped SVG elements with nested data. First though, you will need to learn how to nest your dataset and to structure your code to layout trellis subplots in a grid.

1. How to structure your d3 code

We have already added structure to your starter code in `stock_prices.js`. The variable declarations at the top help to compute pixel-space values for laying out a 2x2 trellis plot grid.

Take a second now to follow along in the comments of the code so that you understand the purpose of the variables.

You can then see them in action as we add 4 rectangles that will be the backgrounds for your trellis plots.

2. Parse the dates

With your dataset loaded, you will need to convert the `string` date attributes into actual JavaScript `Date` objects. `Date` objects refer to a specific date and time, they can be used by D3 to create a `d3.timeScale()`. So in order to parse the dates we need to loop through the entire dataset, and parse each date string with the `parseDate` function that we have already added for you, and then declare the `Date` object:

```
dataset.forEach(function(row) {
    row.date = parseDate(row.date);
});
```

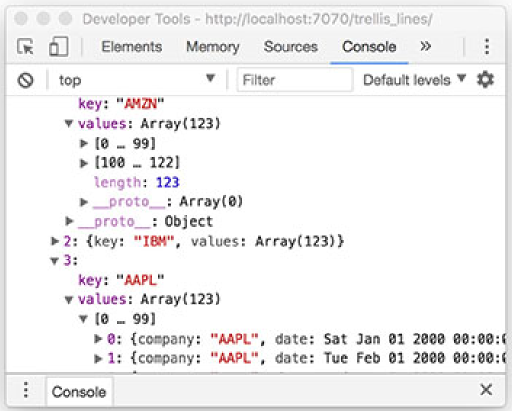
3. Nest the loaded dataset

Now we need to nest the dataset to prepare it for visualizing. Our trellis plot varies on the `company` data attribute, so we want our data to follow suit. Let's call a `d3.nest()` function where we return the `company` property for each data object.

Make sure to use `entries(dataset)` to close out the nest function, not `object()`. Also, we will not be using the `rollup` function in this example because we do not need to show any aggregate values in our trellis display.

(You can refer to the instructions from prelab on how to use `d3.nest()`)

`console.log()` your nesting result, and you should get this:



```
      key: "AMZN"
    ▼ values: Array(123)
      ▶ [0 … 99]
      ▶ [100 … 122]
        length: 123
      ▶ __proto__: Array(0)
    ▶ __proto__: Object
  ▶ 2: {key: "IBM", values: Array(123)}
  ▼ 3:
      key: "AAPL"
    ▼ values: Array(123)
      ▼ [0 … 99]
        ▶ 0: {company: "AAPL", date: Sat Jan 01 2000 00:00:
        ▶ 1: {company: "AAPL", date: Tue Feb 01 2000 00:00:
```

We will now use nesting visual elements with data to create our trellis line chart of stock data.

Activity 1 - Grouping a trellis line chart

Creating a trellis display can look daunting when you first start. There are a lot of elements on the canvas.

It is important when learning something new that you work incrementally. Start with creating one element, check that it appears in the canvas (remember DOM inspection is the best way to debug). If the element wasn't correctly added you only have that one block of code to debug. If you add a bunch of code at once, its much harder to find what went wrong.

So let's get started on creating the trellis plot, and lets do it incrementally:

1. Append trellis groupings

First we want to append a group for each trellis subplot. We have already created a nested dataset earlier in the lab. Our `nested` variable is a list of 4 objects for each company, each object has a `values` list of the monthly stock prices for that company. We will use that data to create 4 `.trellis` groups:

```
var trellisG = svg.selectAll('.trellis')
    .data(nested)
    .enter()
    .append('g')
    .attr('class', 'trellis')
    .attr('transform', function(d, i){
       //use similar code as has been used for the background rectangles
     });
```

Now we want to position each of these groups in front of the rectangle backgrounds. To do this, we will use the same `.attr('transform', function(d,i) {...` callback used for the background rectangles.

This code uses the indices of the 4 appended groups to position them. The math `(i % 2)` and `Math.floor(i/2)` takes the indices `[0,1,2,3]` and converts them to position indices `[[0,0], [1,0], [0,1], [1,1]]`. We can then multiply those indices by a constant factor `width + padding` to get the pixel space position.

Check to make sure your groups were translated correctly using DOM inspection.

2. Create scales for our line charts

Next we are going to append a `path` element for each trellis subplot. But before we do that - we need to create scales to convert the stock prices and dates into trellis pixel space. We have already computed the data `domain` s for both the `price` and `date` attributes.

```
var xScale = d3.scaleTime()
    .domain(dateDomain)
    .range([0, trellisWidth]);

var yScale = d3.scaleLinear()
    .domain(priceDomain)
    .range([trellisHeight, 0]);
```

Notice, that the `range` for the `yScale` has been flipped. This is because we want our line charts to grow upward. Also notice, that we only need to use 0 and the `trellisWidth` or `trellisHeight` for the range. We don't want to hard-code any padding in our scale, because the 4 groups that we just added will take care of positioning in the overall canvas space.

Next we will need to define a line interpolator for the line chart. `d3.line()` is a function that generates a `d` path attribute for x and y data callbacks:

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

We will use this in the following section to define the x and y points of the lines.

3. Create the line chart

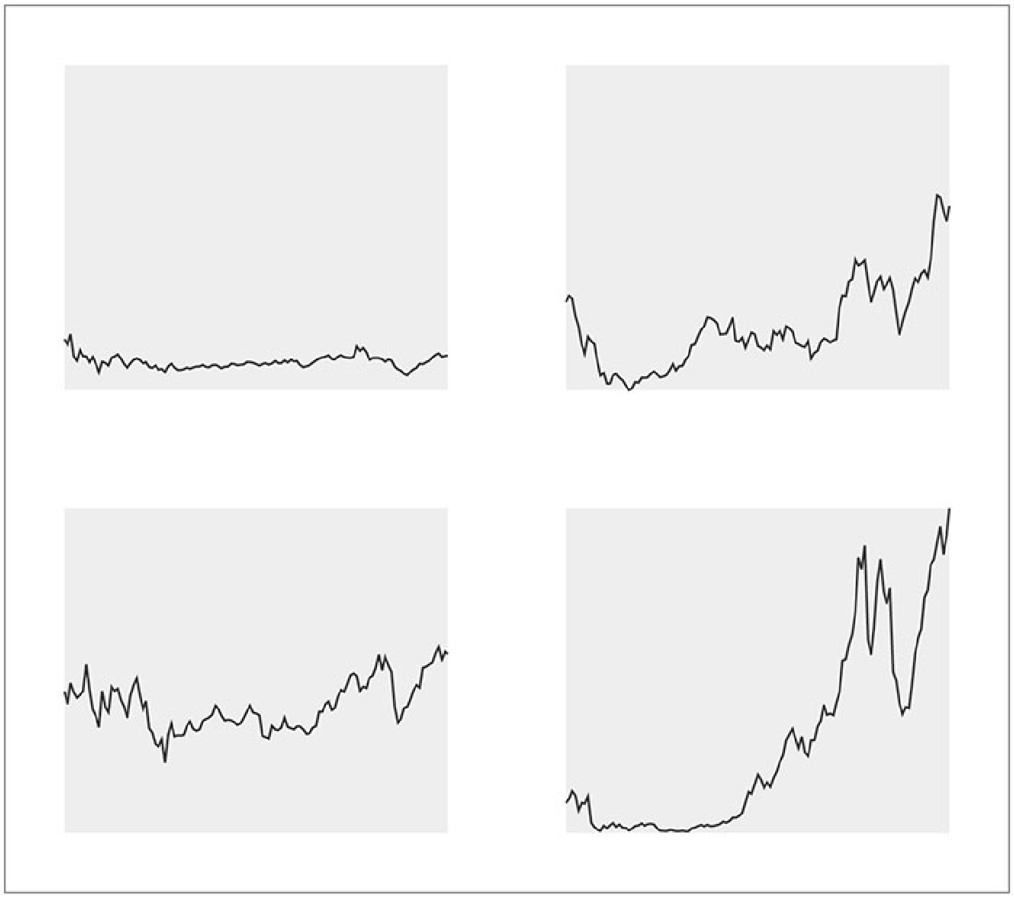
Finally we can add our lines! To do this we will need to nest the `values` array for each company into a new `path` element. The `path` element will also be appended to its respective parent group element.

```
trellisG.selectAll('.line-plot')
    .data(function(d){
        return [d.values];
    })
    .enter()
    .append('path')
    .attr('class', 'line-plot')
    .attr('d', lineInterpolate)
    .style('stroke', '#333');
```

Wait, the data function looks even stranger this time! We are returning a new array that contains the array of `values` ? This is because we only want to append one `path` element. So we want to return an array with one array element in it.

We are then going to use all of the `values` to populate the position of our `path` . `.attr('d', lineInterpolate)` handles all of this for us - it specifies the x and y locations for the path based on the data bound to it, and the rules we defined when we declared `lineInterpolate` .

Here is the result:



4. Create axes for each subplot

Next we are going to continue to use the `trellisG` selection to add new elements to the trellis subplot. Any elements that we append to this selection will be added to all 4 of our subplots.

So if we define new axes, we can add them to each of the 4 subplots:

```
var xAxis = d3.axisBottom(xScale);
trellisG.append('g')
    .attr('class', 'x axis')
    .attr('transform', 'translate(0,'+trellisHeight+')')
    .call(xAxis);

var yAxis = d3.axisLeft(yScale);
trellisG.append('g')
    .attr('class', 'y axis')
    .attr('transform', 'translate(0,0)')
    .call(yAxis);
```

Notice the `translate` s for both of the axis groups we just appended. The `translate` positions only need to be relative to the trellis pixel space! Because of relative positioning inside of a group we only need to position the axes at the left and bottom of each of the trellis subplots.

5. Add color

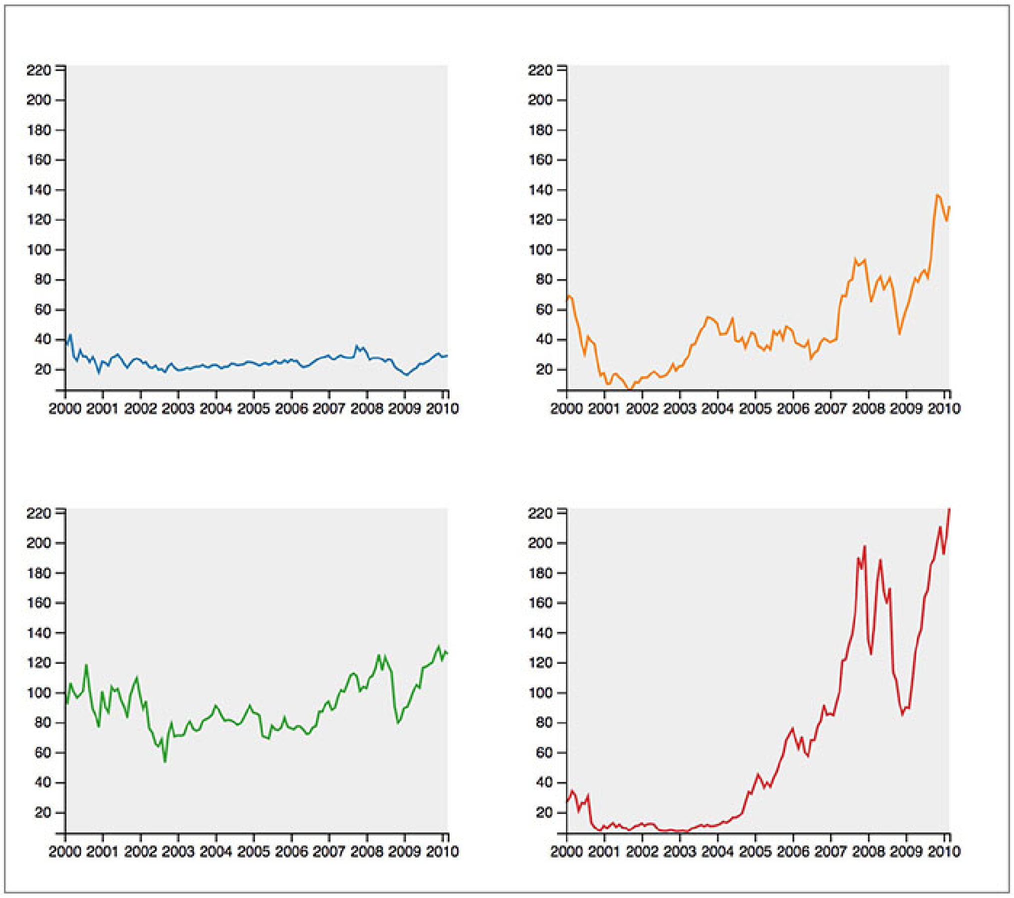
Finally, lets add some color to our chart. We are going to use the `d3.scaleOrdinal` and the `d3.schemeCategory10` to create an automated categorical color mapping for us. D3 has a handful of pre-defined color mappings we can use. All we need to do is define an input `domain` . Which we do by using the array `map` function to generate a list of the `string` company names, the result looks like this `['MSFT', 'AAPL', 'IBM', 'AMZN']`

```
var companyNames = nested.map(function(d){
        return d.key;
    });
var colorScale = d3.scaleOrdinal(d3.schemeCategory10)
    .domain(companyNames);
```

We can now use our color scale on our line charts to color each one. We do this by removing the `.style('stroke', '#333')` line from the block where we appended our lines. And add the following:

```
.style('stroke', function(d) {
    return colorScale(d[0].company);
});
```

From this we get the following result:



Challenge 1 - Finishing touches

You can work on this challenge if you have finished both the activities above. For this, we will add grid lines to the chart, a text label for each trellis subplot, and label the axes. Again, we will do this incrementally:

1. Append company labels

We first want to append a text element for each trellis plot. Remember that we have a `trellisG` selection that we can use to append an element to each subplot. So lets use that to add a company label:

```
 trellisG.append('text')
    .attr('class', 'company-label')
    .attr('transform', 'translate('+[trellisWidth / 2, trellisHeight / 2]+')')
    .attr('fill', function(d){
        return colorScale(d.key);
    })
    .text(function(d){
        return d.key;
    });
```

Note that this appended `text` element still has a reference to the company data object. We use it in the `.text(function(d) { ...` command to set the content.

2. Label the axes

Next we are going to label the axes for each chart. We can do this similar to how we added the company label.

```
trellisG.append('text')
    .attr('class', 'x axis-label')
    .attr('transform', 'translate('+[trellisWidth / 2, trellisHeight + 34]+')')
    .text('Date (by Month)');

trellisG.append('text')
    .attr('class', 'y axis-label')
    .attr('transform', 'translate('+[-30, trellisHeight / 2]+') rotate(270)')
    .text('Stock Price (USD)');
```

Remember that when we append to `trellisG` that the coordinate space is relative to each of the trellis subplots. So we only need to position the text relative to that space.

3. Add grids

Adding grids in D3, is a bit of a hack. Even Mike Bostock uses this method though, so who are we to protest. To add an axis we are going to create modified axes that have the following properties:

- Long ticks to extend across the chart
- No text on each tick
- No domain path

To achieve all of this we only need to set the `tickSize` and `tickFormat` to special values:

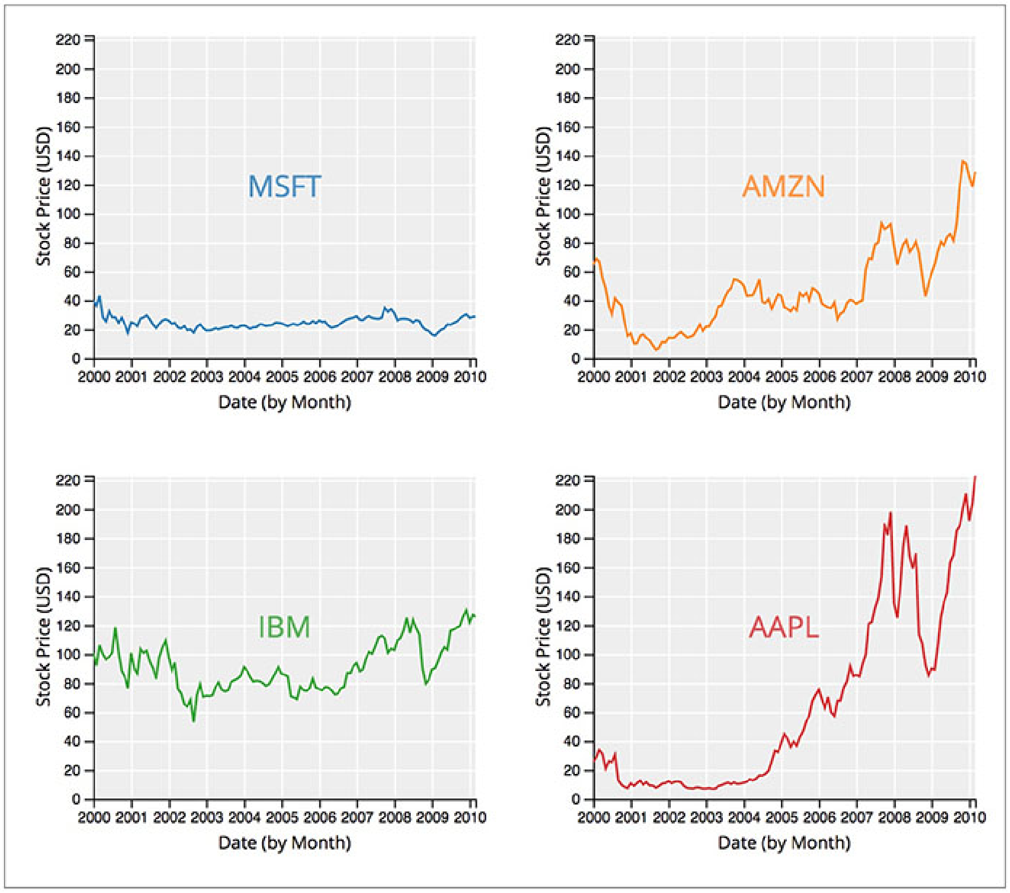
```
var xGrid =d3.axisTop(xScale)
    .tickSize(-trellisHeight, 0, 0)
    .tickFormat('');

trellisG.append('g')
    .attr('class', 'x grid')
    .call(xGrid);

var yGrid = d3.axisLeft(yScale)
    .tickSize(-trellisWidth, 0, 0)
    .tickFormat('')

trellisG.append('g')
    .attr('class', 'y grid')
    .call(yGrid);
```

And with that final addition we have created our trellis line chart:



Submission

- Please change the text of `<p>` in index.html to your name.
- **Take a screenshot of both, the code and the output (browser) to receive full credit.**
- Please remember to submit the screenshot before 10:05 AM of lab day.

Congratulations, you have now finished Lab 5 and created trellis plots in D3. However, before winding up - think about some of the design principles that were discussed in class in the context of trellis charts and small multiples. There are a few things "wrong" with the design above - what could you improve?

This lab was based on the following material:

- Nested Selections by Mike Bostock
- Manipulating data like a boss with d3 by Jerome Cukier
- Advanced D3: More on selections and data, scales, axis by A. Lex of U. of Utah

Home