

Lab 7: Interaction & Transition 1 (Activities)

Wijaya, Mario edited this page 11 days ago · 1 revision

For the lab activity, we will use what we have learned about the **Enter, Update, Exit** pattern to create an interactive scatterplot. Remember the following about the D3 Pattern:

- **ENTER** - incoming elements, entering the scene. Creates a placeholder to append new elements. Should be used to declare attributes and styles that will remain constant over time.
- **UPDATE** - persistent elements, already on the scene. Should be used to declare attributes and styles that will change over time as the scene *updates*. In this case we want to update based on the new data attributes for the x- and y-scales.
- **EXIT** - outgoing elements, exiting the scene. Elements that no longer have data joined to them. These elements can be removed with `.exit().remove()`

ENTER + UPDATE - can be created using `updateSelection.merge(enterSelection)` and can be used to update all attributes and styles that will change based on new or updated data.

Activity 0 - Updating a scatterplot

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 `cars.csv` dataset. The dataset includes 406 rows. Each row corresponds to a model and make of car. The dataset also includes a lot of interesting performance measures for each of these cars:


name	economy (mpg)	cylinders	displacement (cc)	power (hp)	weight (lb)	0-60 mph (s)	year
Ford Escort 2H	29.9	4	98	65	2380	20.7	81
Ford Gran Torino	16	8	302	140	4141	14	74
Honda Civic	33	4	91	53	1795	17.4	76
Subaru DL	30	4	97	67	1985	16.4	77
Subaru DL	33.8	4	97	67	2145	18	80

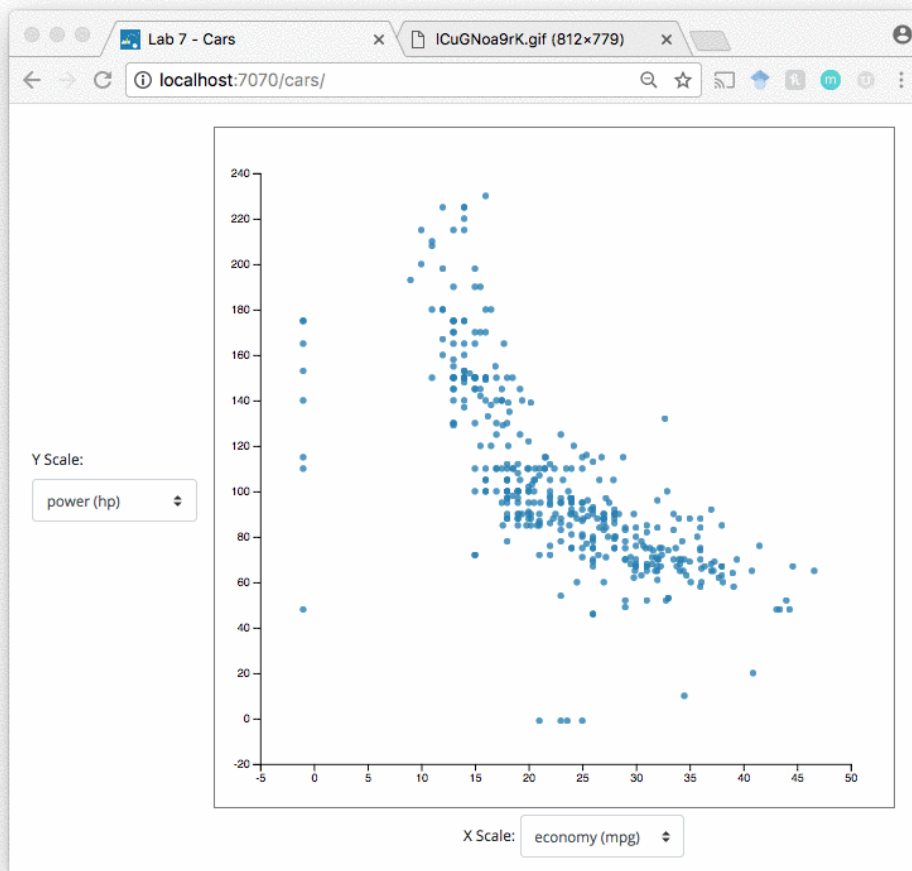
You will be working toward creating the following animated and interactive scatterplot today:

► Pages 18

[Lab 0: HTML & CSS](#)
[Lab 1: Javascript 101](#)
[Lab 2: SVG](#)
[Lab 3: Intro to D3 \(Pre Lab\)](#)
[Lab 3: Intro to D3 \(Activities\)](#)
[Lab 4: D3 Chart Types & Scales \(Pre Lab\)](#)
[Lab 4: D3 Chart Types & Scales \(Activities\)](#)
[Lab 5: D3 Selections & Grouping \(Pre-Lab\)](#)
[Lab 5: D3 Selections & Grouping \(Activities\)](#)
[Lab 6: D3 Enter, Update & Exit \(Pre-Lab\)](#)
[Lab 6: D3 Enter, Update & Exit \(Activities\)](#)
[Lab 7: Interaction & Transition 1 \(Pre Lab\)](#)
[Lab 7: Interaction & Transition 1 \(Activities\)](#)
[Lab 8: Interaction and Transition 2](#)
[Lab 9: D3 Layouts](#)
[Lab 10: D3 Maps](#)

Clone this wiki locally

<https://github.gatech.edu> 



1. How to structure your code for interaction

Like previous weeks we have already added structure to your `cars/main.js` code. This should help speed up the time it takes to complete the lab. There are a number of additions to the code:

- `onXScaleChanged` and `onYScaleChanged` methods - used to handle change events from the `select` input widget for specifying the columns for each x- and y-scale.
- `chartScales` - an object with `x` and `y` properties that keeps the state of which data column is currently used for the x- and y- scales
- Layout parameters for configuring the spacing of your chart. `chartwidth` and `chartHeight` can be used for the `.range()` of your x- and y-scales.
- `chartG` is a group that has been positioned based on the `padding`. Add your scatterplot circles and axes to this element.
- `d3.csv(fileName, processingMethod, dataCallback)` is included
- `updateChart()` the method to be called for new scale values

Take some time to look through the template and read the comments.

2. Create a global data variable

You will need to access the loaded dataset in the `updateChart` method, you will need to create a global variable of the loaded dataset. Add the following declaration within the `function(error, dataset) { ... }` data callback method:

```
cars = dataset;
```

3. Create your x- and y-scales

Create two `scaleLinear` s for `x` and `y`. You will want the output range of this scale to be based on the chart dimensions. **Again you want both `xScale` and `yScale` to be global variables:**

```
xScale = d3.scaleLinear()
  .range([0, chartWidth]);

yScale = d3.scaleLinear()
  .range([chartHeight, 0]);
```

4. Find the min/max for each data attribute column

We want to create an object map (aka dictionary) that maps each data column to the min/max extent pair. We will want to use this map in the `updateChart` method to adjust the input `domain` of the `x`- and `y`-scales. We can do this with the following code:

```
domainMap = {};

dataset.columns.forEach(function(column) {
  domainMap[column] = d3.extent(dataset, function(data_element){
    return data_element[column];
  });
});
```

Note, D3 provides the list of data columns as a property of the loaded csv dataset .

5. Update the scale domain s on updateChart

Now within `updateChart` .

We will use the scale objects that we created as well as the `domainMap` to update the `domain` whenever the user selects a new column for that scale.

```
// Update the scales based on new data attributes
yScale.domain(domainMap[chartScales.y]).nice();
xScale.domain(domainMap[chartScales.x]).nice();
```

Note, we can call `.nice()` on a scale to round off the input domain to nice digits.

6. Update the axes scale domain s on updateChart

Still in `updateChart` .

We will use the scale objects that we just updated to create/update the `x`- and `y`-axes of the chart using the `yAxisG` and `xAxisG` groups that we already created for you:

```
xAxisG.call(d3.axisBottom(xScale));
yAxisG.call(d3.axisLeft(yScale));
```

7. Use D3's Enter & Update to create the circles for the scatterplot

Now we are going to create the dots that make up our scatterplot.

First we are going to create a d3-selection on our class `dot` and create a data-join with the global `cars` array:

```
var dots = chartG.selectAll('.dot')
  .data(cars);
```

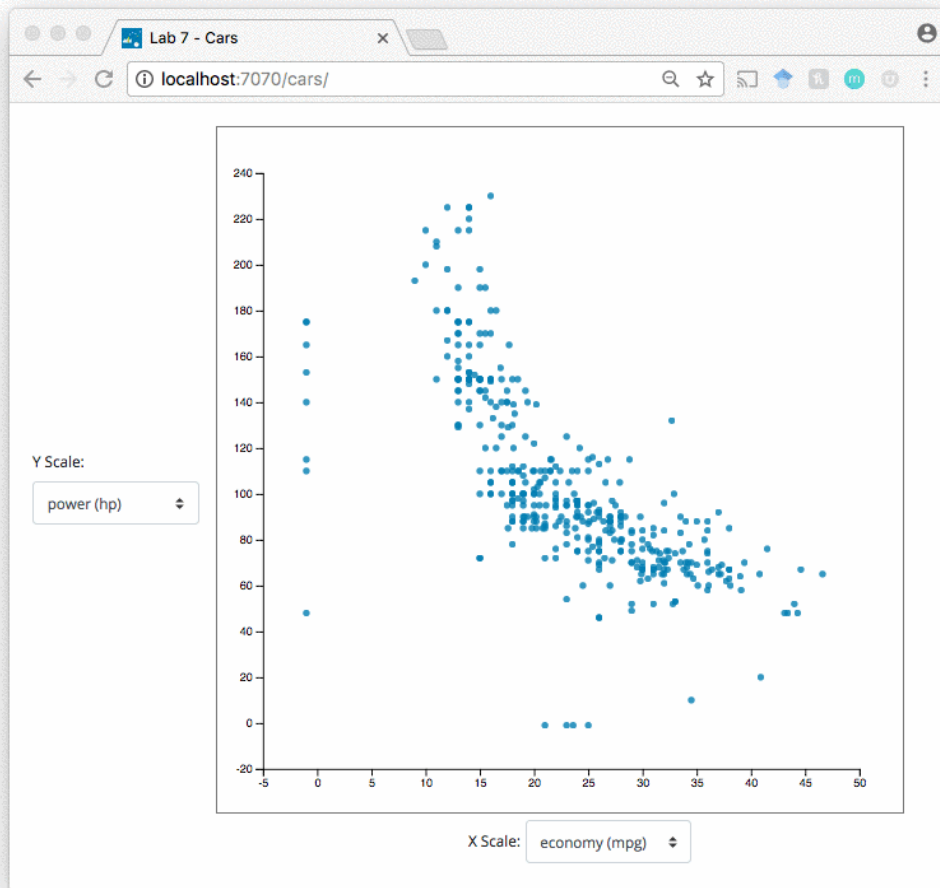
`dots` is our Update selection, but because we don't have any `.dot` elements on the canvas - our Update selection is empty. So let's fix that with the Enter selection:

```
var dotsEnter = dots.enter()
  .append('g')
  .attr('class', 'dot')
  .attr('transform', function(d) {
    var tx = xScale(d[chartScales.x]);
    var ty = yScale(d[chartScales.y]);
    return 'translate(' + [tx, ty] + ')';
  });

dotsEnter.append('circle')
  .attr('r', 3);
```

Here, calling `.enter()` will prepare 406 placeholders for the 406 car array elements that currently do not have a data-joined SVG element. Calling `append` then adds 406 `g.dot` elements and completes the data-binding to elements on the canvas.

The above code will create our scatterplot for the first time `updateChart` is called. However, what happens when we change the x- and y-scale with the `select` pickers?



Nothing, our chart will not update!

So how do we *update* data-bound elements that already exist on the canvas. With `UPDATE` of course!

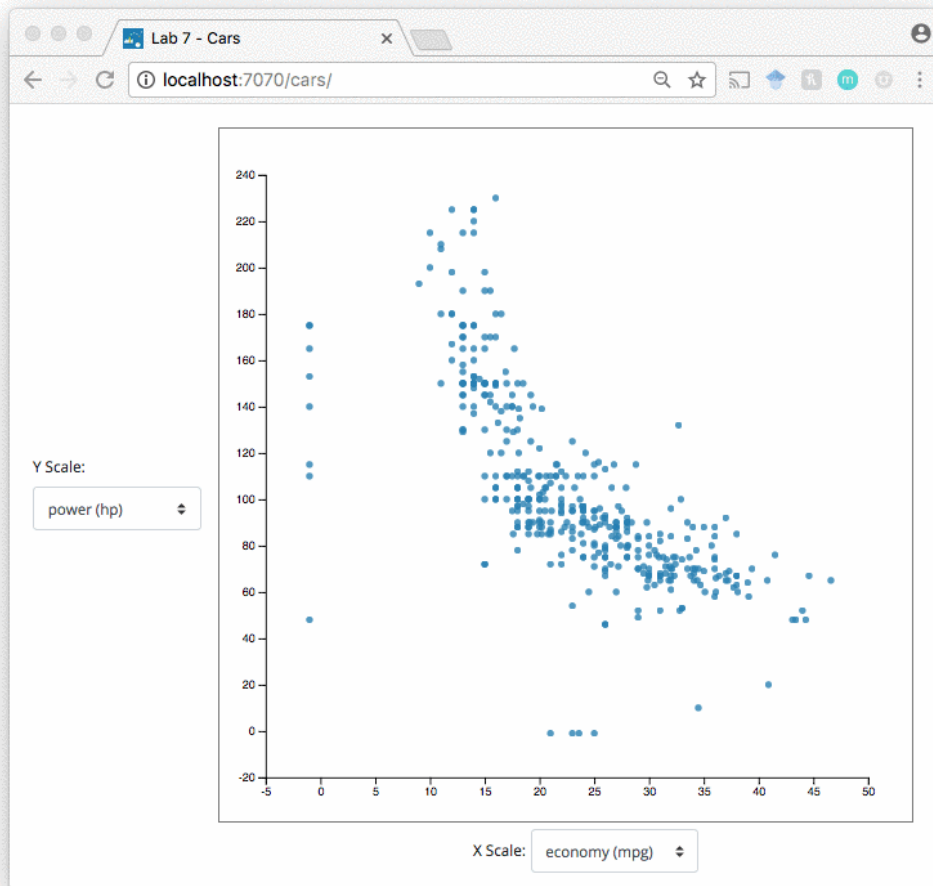
When `updateChart` is called a subsequent time (after all 406 `g.dot` elements have been appended), the `dots` selection now will include all of those elements. Now we can update the transform of them based on the new x- and y-scales:

```
dots.attr('transform', function(d) {
  var tx = xScale(d[chartScales.x]);
  var ty = yScale(d[chartScales.y]);
  return 'translate('+[tx, ty]+)';
});
```

Great, now the chart updates as expected! But wait, why should we call `.attr('transform', function(d) {...})` twice - that seems like wasteful code. Let's use `.merge()` to combine our ENTER + UPDATE selections and only set the transform once:

```
dots.merge(dotsEnter)
  .attr('transform', function(d) {
    var tx = xScale(d[chartScales.x]);
    var ty = yScale(d[chartScales.y]);
    return 'translate('+[tx, ty]+)';
  });
```

And we get the following result:



Where's the EXIT selection though? In this example we are only adding and updating elements on the chart. No need to use an EXIT selection.

In the following activities we will learn how to create a simple hover tooltip and add animation to our chart.

Activity 1 - Adding on hover

In this activity we will add a simple details interaction on hover to our scatterplot. In this activity we will add a text element to each `dot` group in our scatterplot. We will then use CSS rules to show the text on hover.

1. Append the name of the cars

You will first need to append a new text element to your `dotsEnter` selection and declare the text content as the name for each car:

```
dotsEnter.append('text')
    .attr('y', -10)
    .text(function(d) {
        return d.name;
    });
```

This will create a lot of text elements, but we'll now hide them with CSS

2. Create a `:hover` CSS rule

```
.dot text {
    opacity: 0;
}

.dot:hover text {
    opacity: 1;
}
```

Activity 2 - Adding Transitions

In this final activity you will animate your scatterplot. Don't worry there isn't too much more code to add:

1. Transition the dot elements

Similar to what we did in the above example, we want to transition the `transform` attribute so that the scatterplot `dot` s interpolate between start and end points:

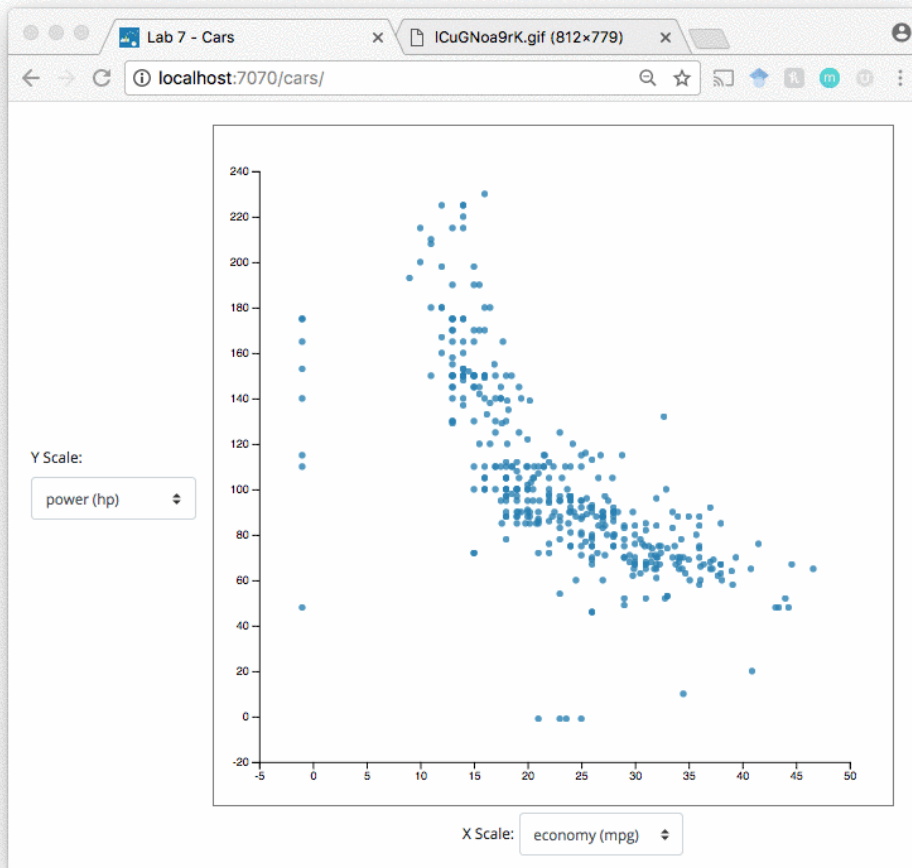
```
dots.merge(dotsEnter)
    .transition()
    .duration(750)
    .attr('transform', function(d) {
        var tx = xScale(d[chartScales.x]);
        var ty = yScale(d[chartScales.y]);
        return 'translate('+[tx, ty]+' )';
    });
```

2. Transition the axes

Next we will add the same `.transition().duration(750)` before we `call()` our axis functions. This will create an animated transition for the updated scale.

```
xAxisG.transition()
    .duration(750)
    .call(d3.axisBottom(xScale));
yAxisG.transition()
    .duration(750)
    .call(d3.axisLeft(yScale));
```

With these steps we get our final result:



Submission

- Please change the text of `<p>` in index.html to your name.
- **Take screenshots of both, the code and the output (browser) of Activity 0, Activity 1, and Activity 2 to receive full credit.**
- Please remember to submit the screenshot before 10:05 AM of lab day.

Congratulations, you have now finished Lab 7 and created transitions and interaction within a chart. Next lab we will cover more in-depth transitions and interactions such as brushing and linking.

This lab was based on the following material:

- Hanspeter Pfister's CS171 Lab Material (Harvard)
- [D3 - Interactive Data Visualization for the Web](#) by Scott Murray
- [Working with Transitions](#) by Mike Bostock
- [Animations and Transitions](#) by Jerome Cukier
- [W3 Schools - HTML Event Attributes](#)

