Tutorial 6: D3.js Part 2

WANG Yong
College of Computing and Data Science
Nanyang Technological University

6.Axes & Legends

Code: 7-axes.html

Code: 8-legends.html

Creating axes

Axes can be generated based on the scales in your visualization. Axes allow you to specify the scale to use, the orientation, and properties of the tick marks on the axis.

axisLeft() is a function. To create our axis, we create or select the element where we want to place it, and then use **call()** to apply the function to it.

The axis labels will also be automatically drawn.

See the D3 Axes page for more information.

Scale:

```
y = d3.scaleLinear()
    .domain(d3.extent(sampleData,function(d){ return d.age; }))
    .range([465,10]);
```

Specify axis:

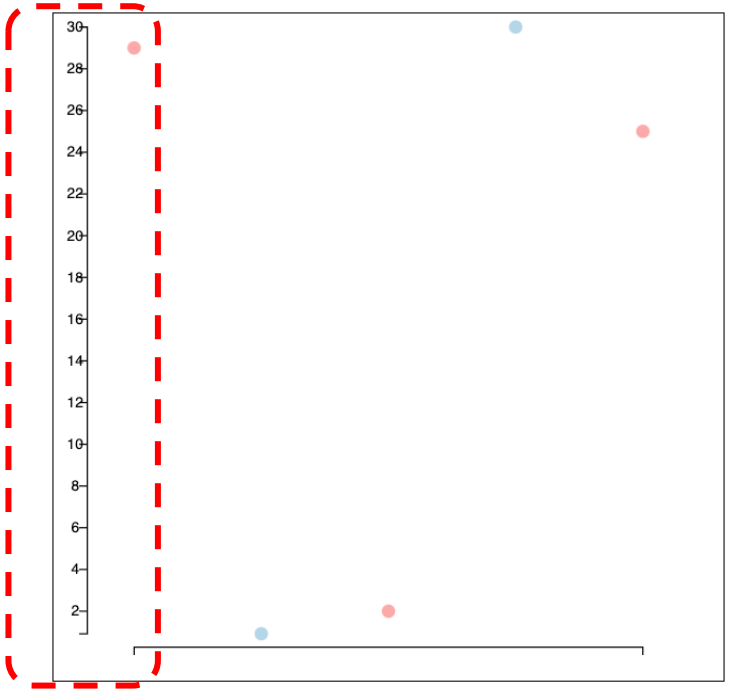
```
yAxis = d3.axisLeft()
    .scale(y);
```

Draw axis:

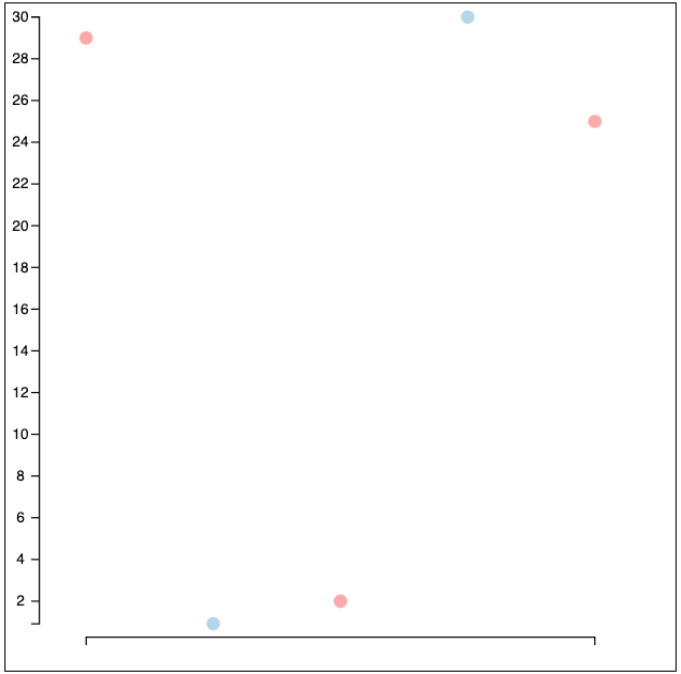
```
canvas.append("g")
  .attr("class","axis")
  .attr("transform","translate(25,0)")
  .call(yAxis);
```

Adjusting axis labels

Axis labels can be adjusted.




```
yAxisGroup.selectAll("text")
    .attr("transform", "translate(-5,0)");
```



Legends

Legends can be constructed just like the other elements of your visualization: by creating a new set of marks and using scales to style the attributes.

In addition to the rect for the legend mark, we can append text to create the legend labels.

```
legend.append("rect")
    .attr("x", 475)
    .attr("y", 9)
    .attr("width", 18)
    .attr("height", 18)
    .style("fill", c);

legend.append("text")
    .attr("x", 465)
    .attr("y", 18)
    .attr("dy", ".35em")
    .style("text-anchor", "end")
    .text(function(d) { return d.charAt(0).toUpperCase()+d.slice(1); });
```

7.Events & Transitions

Code: 9-events.html

Code: 10-transitions.html

Reacting to events

Event listeners can be added to marks to react to events on the underlying selection. Every time an event fires, the listener fires and the function is evaluated.

An anonymous function can be used as the callback for the event listener. The input to the function **d**, represents the underlying data of the mark. The scope, **this**, corresponds to the DOM element.

```
.on("mouseover", function(event, d){
  d3.select(this)
    .attr("stroke-width", "5px")
    .attr("r", r(d.fare));
})
.on("mouseout", function(event, d){
  d3.select(this)
    .attr("stroke-width","0px")
    .attr("r",5);
});
```

Old versions of d3 (<= v5):

```
.on("mouseover", function(d){
  d3.select(this).attr("stroke-width", "5px")
    .attr("r", r(d.fare));
})
.on("mouseout", function(){
  d3.select(this).attr("stroke-width","0px")
    .attr("r",5);
})
```

8.Loading Files

Code: 11-csv.html

Loading data from external files

Data can be loaded from many types of external files using commands such as **d3.csv**, **d3.json**, **d3.tsv**.

The D3 functions additionally support callback functions for dealing with the resulting data or error cases.

```
d3.csv("titanic passenger list.csv", (row, i) => {
  return {
    name: row.name,
    survived: (row.survived==1) ? "Yes": "No",
    sex: row.sex,
    age: +row.age,
    fare: +row.fare,
  };
}).then(rows => {
  rows.sort(function(a,b) { return (a.name).localeCompare(b.name); });
  allData = rows;
  makeChart(rows.slice(index,index+10));
}).catch(error => {
  console.log(error);
});
```

Loading data from external files

What to do per row:
(Including creating aliases or specifying data type.

Callback function

What to do with all returned rows (including sorting, filtering, etc.

Error handling

```
d3.csv("titanic passenger list.csv", (row, i) => {
  return {
    name: row.name,
    survived: (row.survived==1) ? "Yes": "No",
    sex: row.sex,
    age: +row.age,
    fare: +row.fare,
  };
}).then(rows => {
  rows.sort(function(a,b) { return (a.name).localeCompare(b.name); });
  allData = rows;
  makeChart(rows.slice(index,index+10));
}).catch(error => {
  console.log(error);
});
```

9.Enter/Update/Exit

Code: 12-exit.html

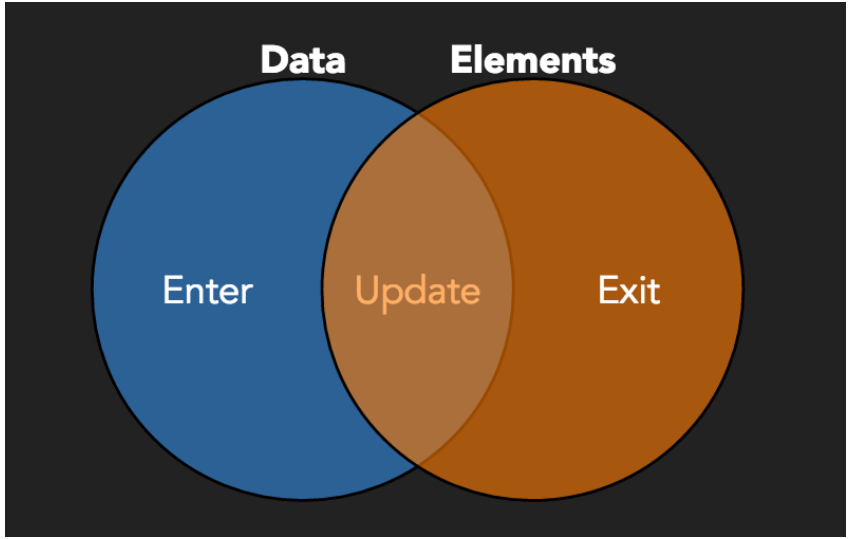
Rebinding

Three things can happen when we call **data()**:

Update: We want to change the elements we already have.

Enter: We have new data.

Exit: We have data that is no longer bound.



Rebinding

Good practice to have an update function.

1. **Bind** or rebind data
2. Perform **update** operations
3. Perform operations on **enter** set
4. Perform operations on **update+enter** sets
5. Perform **exit** operations

```
/***************
 * Update Loop *
 ***************/

function update(data){

  //BIND DATA
  var scatter = d3.select(".chart").selectAll("circle")
      .data(data);

  // x position
  x = d3.scaleLinear()
      .domain([0,data.length-1])
      .range([35,490]);

  //UPDATE
  scatter.attr("stroke-width","5px");

  //ENTER
  var enter = scatter.enter().append("circle")
      .attr("fill-opacity",0.85)
      .attr("r",5)
      .attr("cx",function(d,i){ return x(i); })
      .attr("cy",function(d){ return y(d.age); })
      .attr("stroke",function(d){ return s(d.survived); })
    .on("mouseover",function(d){
      d3.select(this).transition()
        .attr("stroke-width","5px")
        .attr("r",r(d.fare));
    })
    .on("mouseout",function(){
      d3.select(this).transition()
        .delay(1000)
        .attr("stroke-width","0px")
        .attr("r",5);
    });

  // Add a title to the point (on mouseover)
  enter.append("svg:title")
      .text(function(d){ return d.name;});

  //ENTER + UPDATE
  enter.merge(scatter).transition().duration(1000)
      .attr("cx",function(d,i){ return x(i); })
      .attr("cy",function(d){ return y(d.age); })
      .attr("fill",function(d){ return c(d.sex); })
      .attr("stroke",function(d){ return s(d.survived); })
      .attr("stroke-width","0px");

  //EXIT
  scatter.exit().transition().duration(1000)
      .attr("cx",0)
      .attr("fill-opacity",0)
      .remove();
}
```

1. Update

Things I want to happen to all of our data, whenever the function is called. Potentially overwritten by later steps.

```
//UPDATE
// x position
x = d3.scaleLinear()
    .domain([0,data.length-1])
    .range([35,490]);

scatter.attr("stroke-width","5px");
```

2. Enter

Things I want to happen to all new data

Can use **append()** to make new elements for new data.

```
//ENTER
var enter = scatter.enter().append("circle")
    .attr("fill-opacity",0.85)
    .attr("r",5)
    .attr("cx",function(d,i){ return x(i); })
    .attr("cy",function(d){ return y(d.age); })
    .attr("stroke",function(d){ return s(d.survived); })
  .on("mouseover",function(d){
    d3.select(this).transition()
      .attr("stroke-width","5px")
      .attr("r",r(d.fare));
  })
  .on("mouseout",function(){
    d3.select(this).transition()
      .delay(1000)
      .attr("stroke-width","0px")
      .attr("r",5);
  });

// Add a title to the point (on mouseover)
enter.append("svg:title")
    .text(function(d){ return d.name;});
```

3. Enter+Update

Things I want to set initially. Can use transitions to have attributes fade in after creation.

```
//ENTER + UPDATE
enter.merge(scatter).transition().duration(1000)
    .attr("cx",function(d,i){ return x(i); })
    .attr("cy",function(d){ return y(d.age); })
    .attr("fill",function(d){ return c(d.sex); })
    .attr("stroke",function(d){ return s(d.survived); })
    .attr("stroke-width","0px");
```

4. Exit

Things I want to happen to old data

Can use transitions to make old data fade away

Can use **remove()** to keep only elements that are bound to our current data.

```
//EXIT
scatter.exit().transition().duration(1000)
    .attr("cx",0)
    .attr("fill-opacity",0)
    .remove();
```

Key binding

With only one argument, binding will only keep track of the amount of data we have.

If we always have the same *amount* of data, then nothing will "exit."

Can use a argument to specify unique identifiers for data, to define whether data should enter or exit.

Here, our key is the index (row number) of the data in our original csv. Passenger name is not unique, and so would not make a good key.

```
d3.csv("titanic passenger list.csv", (row, i) => {
  return {
    name: row.name,
    survived: (row.survived==1) ? "Yes": "No",
    sex: row.sex,
    age: +row.age,
    fare: +row.fare,
  };
}).then(rows => {
  rows.sort(function(a,b) { return (a.name).localeCompare(b.name); });
  allData = rows;
  makeChart(rows.slice(index,index+10));
}).catch(error => {
  console.log(error);
});
```

```
var key = function(d){ return d.key; };
```

```
//BIND DATA
var scatter = d3.select(".chart").selectAll("circle")
    .data(data,key);
```

Questions?

Optional Exercises After Class

- Introduction to D3 (Part 2): https://observablehq.com/@jiayouwyhit/introduction-to-d3-part-2

- You can fork the above exercises, and then finish the three TO-DO tasks

Acknowledgement

This tutorial is adapted from data visualization course taught by Dr. Jeffrey Heer at the University of Washington.