Data-Driven Documents

Maurizio Tesconi

April, 21 - 2015

Anscombe's quartet - datasets

Data Set A	
X	**Y**
10.0	8.04
8.0	6.95
13.0	7.58
9.0	8.81
11.0	8.33
14.0	9.96
6.0	7.24
4.0	4.26
12.0	10.84
7.0	4.82
5.0	5.68

Data Set B	
X	**Y**
10.0	9.14
8.0	8.14
13.0	8.74
9.0	8.77
11.0	9.26
14.0	8.10
6.0	6.13
4.0	3.10
12.0	9.13
7.0	7.26
5.0	4.74

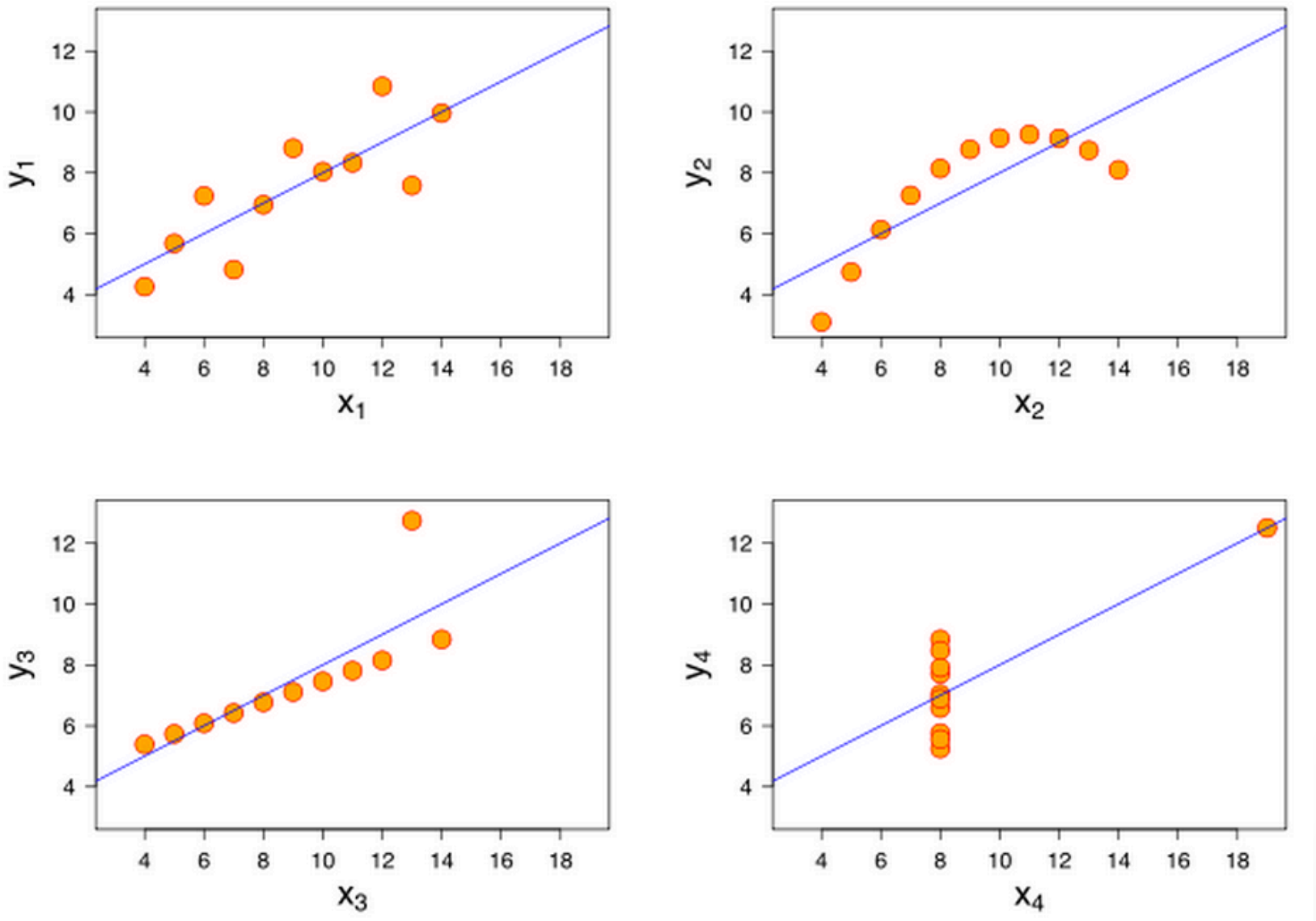
Data Set C	
X	**Y**
10.0	7.46
8.0	6.77
13.0	12.74
9.0	7.11
11.0	7.81
14.0	8.84
6.0	6.08
4.0	5.39
12.0	8.15
7.0	6.42
5.0	5.73

Data Set D	
X	**Y**
8.0	6.58
8.0	5.76
8.0	7.71
8.0	8.84
8.0	8.47
8.0	7.04
8.0	5.25
19.0	12.50
8.0	5.56
8.0	7.91
8.0	6.89

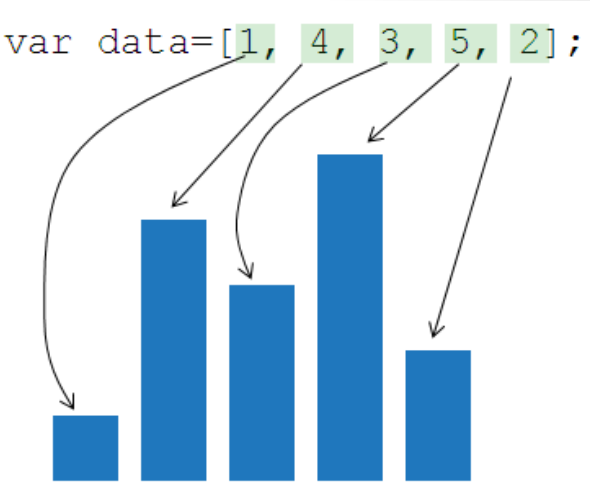
Anscombe's quartet - properties

Property	Value
Mean of x in each case	9 (exact)
Sample variance of x in each case	11 (exact)
Mean of y in each case	7.50 (to 2 decimal places)
Sample variance of y in each case	4.122 or 4.127 (to 3 decimal places)
Correlation between x and y in each case	0.816 (to 3 decimal places)
Linear regression line in each case	$y = 3.00 + 0.500x$ (to 2 and 3 decimal places, respectively)

Anscombe's quartet – graphics



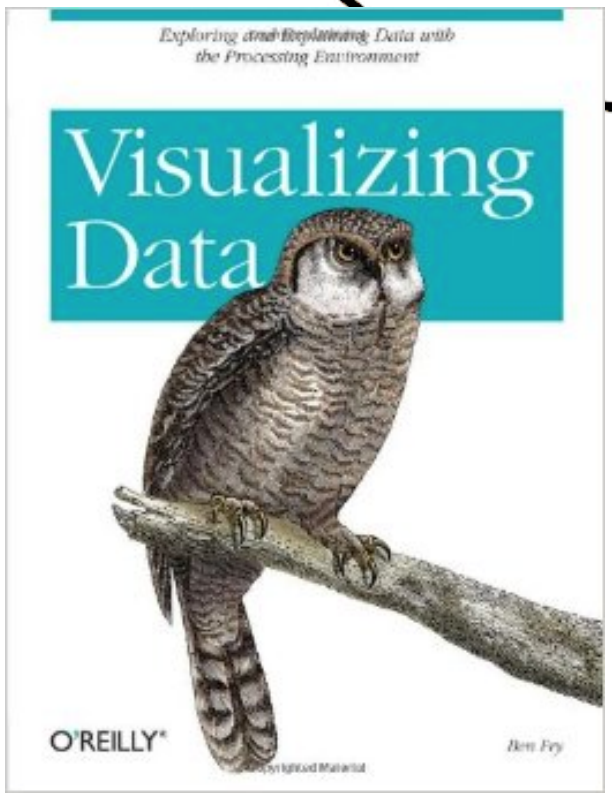
What is D3?



- JavaScript library to make beautiful, interactive, browser-based data visualizations.
- D3 stands for Data Driven Documents
- D3.js is a low level visualization library based on Web standards (HTML, CSS, JS, SVG)
- D3.js is Open Source library written by Mike Bostok
- Mike Bostock Github Profile
- D3 web page
-

7 steps of Data Visualization process

ACQUIRE ▷ PARSE ▷ FILTER ▷ MINE ▷ REPRESENT ▷ REFINE ▷ INTERACT

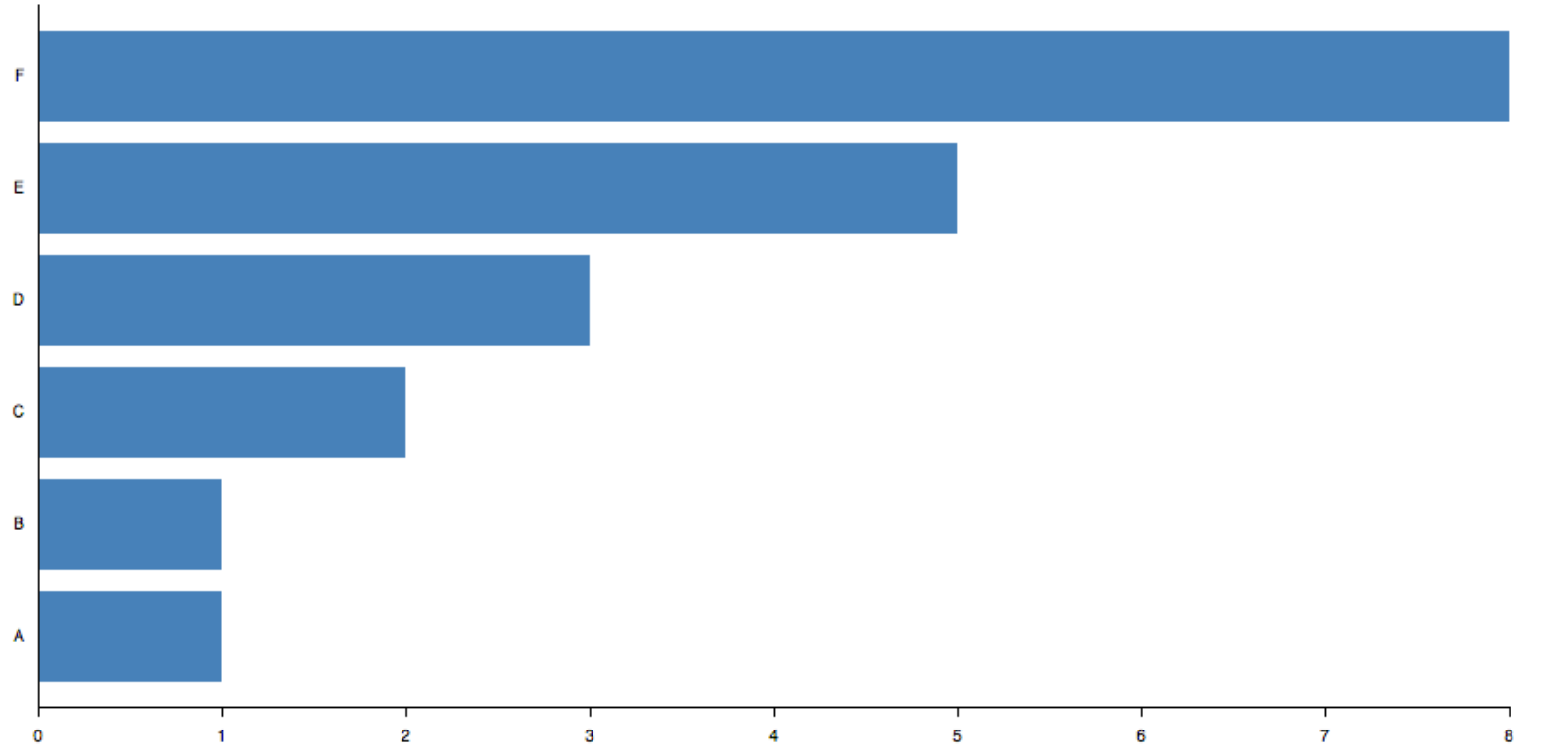


Data-Driven Documents

Data can be numbers.

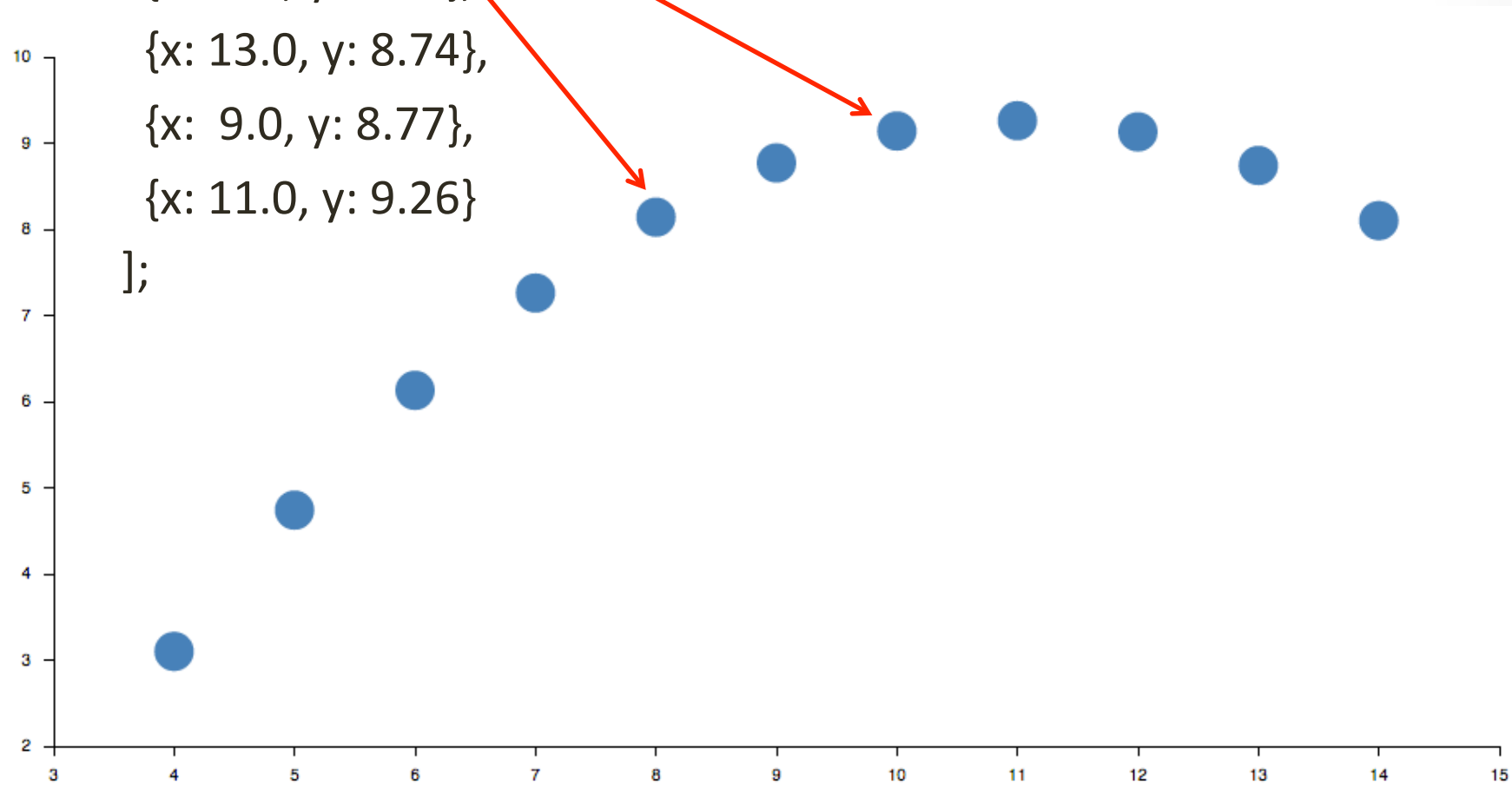
JSBIN

```
var numbers= [1, 1, 2, 3, 5, 8];
```



Data can be objects.

```
var data = [
  {x: 10.0, y: 9.14},
  {x:  8.0, y: 8.14},
  {x: 13.0, y: 8.74},
  {x:  9.0, y: 8.77},
  {x: 11.0, y: 9.26}
];
```



Getting started

```
<!DOCTYPE html>
<html>
        <head>
                <script src="http://d3js.org/d3.v3.min.js"></script>
                <script src="js/main.js" ></script>
        </head>
        <body>
                ...
        </body>
</html>
```

main.js

```
d3.select("body").append("p").text("Hello World!");
```

Chaining methods

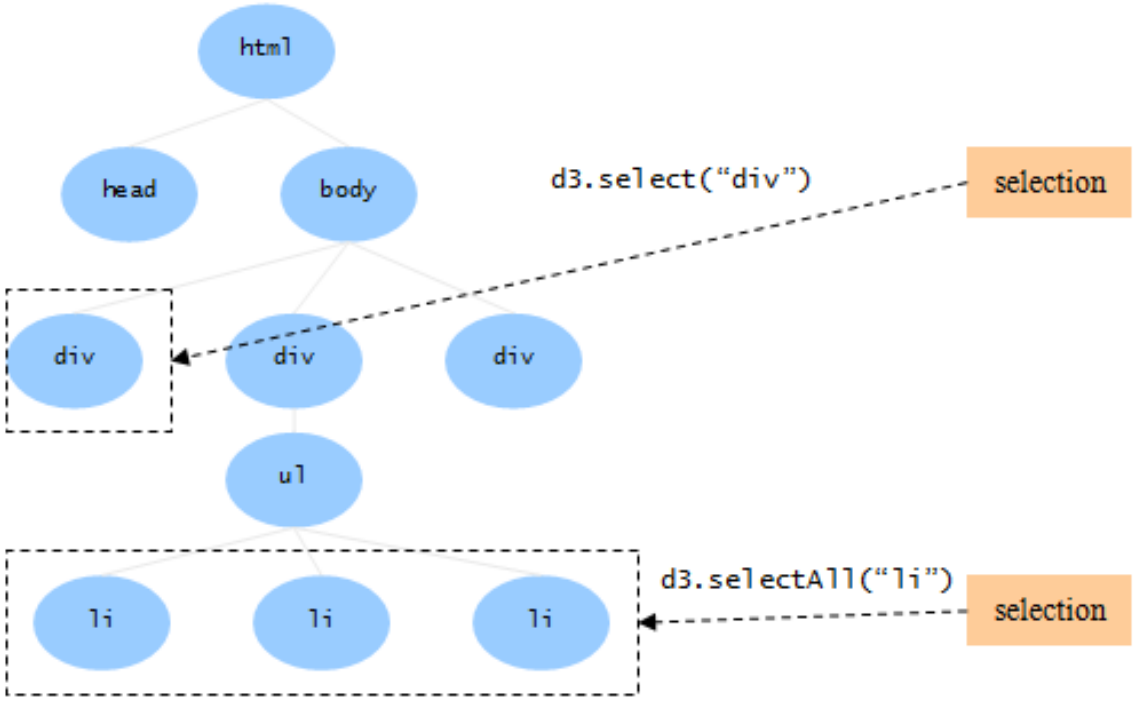
```
d3.select("body").append("p").text("New paragraph!");
```

```
d3.select("body")
    .append("p")
    .text("New paragraph!");
```

```
svg.selectAll("circle")
    .data(data)
  .enter().append("circle")
    .attr("cx", x)
    .attr("cy", y)
    .attr("r", 2.5);
```

Selections

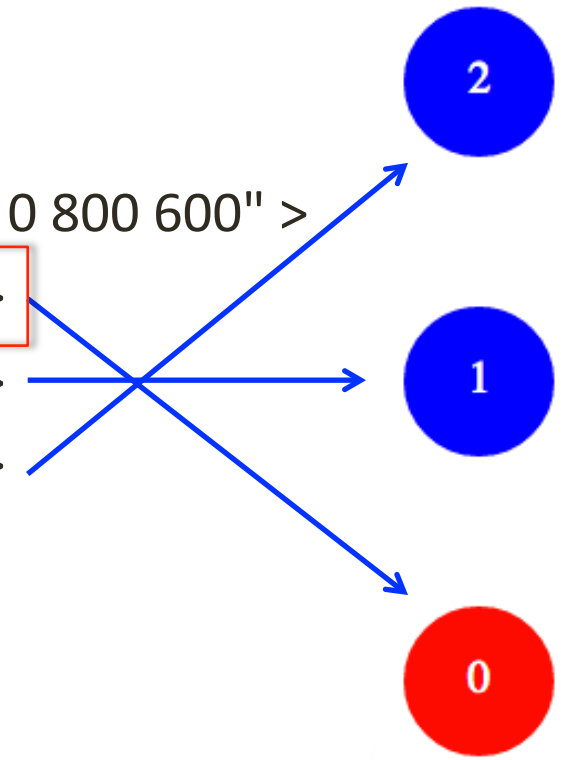
- Javascript document.querySelectorAll("pre, code”)
- Jquery $("pre, code").
- D3 d3.selectAll("pre, code")



d3.select("circle")
.attr("fill", "red");

```
<svg width="100%" height="100%" viewbox="0 0 800 600" >
  <circle cx="400" cy="500" r="50" fill="blue" />
  <circle cx="400" cy="300" r="50" fill="blue" />
  <circle cx="400" cy="100" r="50" fill="blue" />
</svg>
```

N.B.



d3.selectAll("circle")
.attr("fill", "red");

```
<svg width="100%" height="100%" viewbox="0 0 800 600" >
  <circle cx="400" cy="500" r="50" fill="blue" />
  <circle cx="400" cy="300" r="50" fill="blue" />
  <circle cx="400" cy="100" r="50" fill="blue" />
</svg>
```

Another example

```
var svg = d3.select("body").append("svg")
                    .attr("width", 100)
                    .attr("height", 100);
svg.append("rect")
                    .attr("x", 0)
                    .attr("y", 0)
                    .attr("width", 100)
                    .attr("height", 100)
                    .attr("stroke", "black")
                    .attr("fill", "none");
svg.append("circle")
                    .attr("cx", 50)
                    .attr("cy", 50)
                    .attr("r", 25)
                    .style("fill", "orange");;
```

Selections & Data

- Selections are arrays

```
<rect x="0" y="150" width="200" height="40" />
<rect x="0" y="200" width="200" height="40" />
<rect x="0" y="250" width="200" height="40" />
```

- Data are arrays

```
var numbers= [10, 20, 5];
```

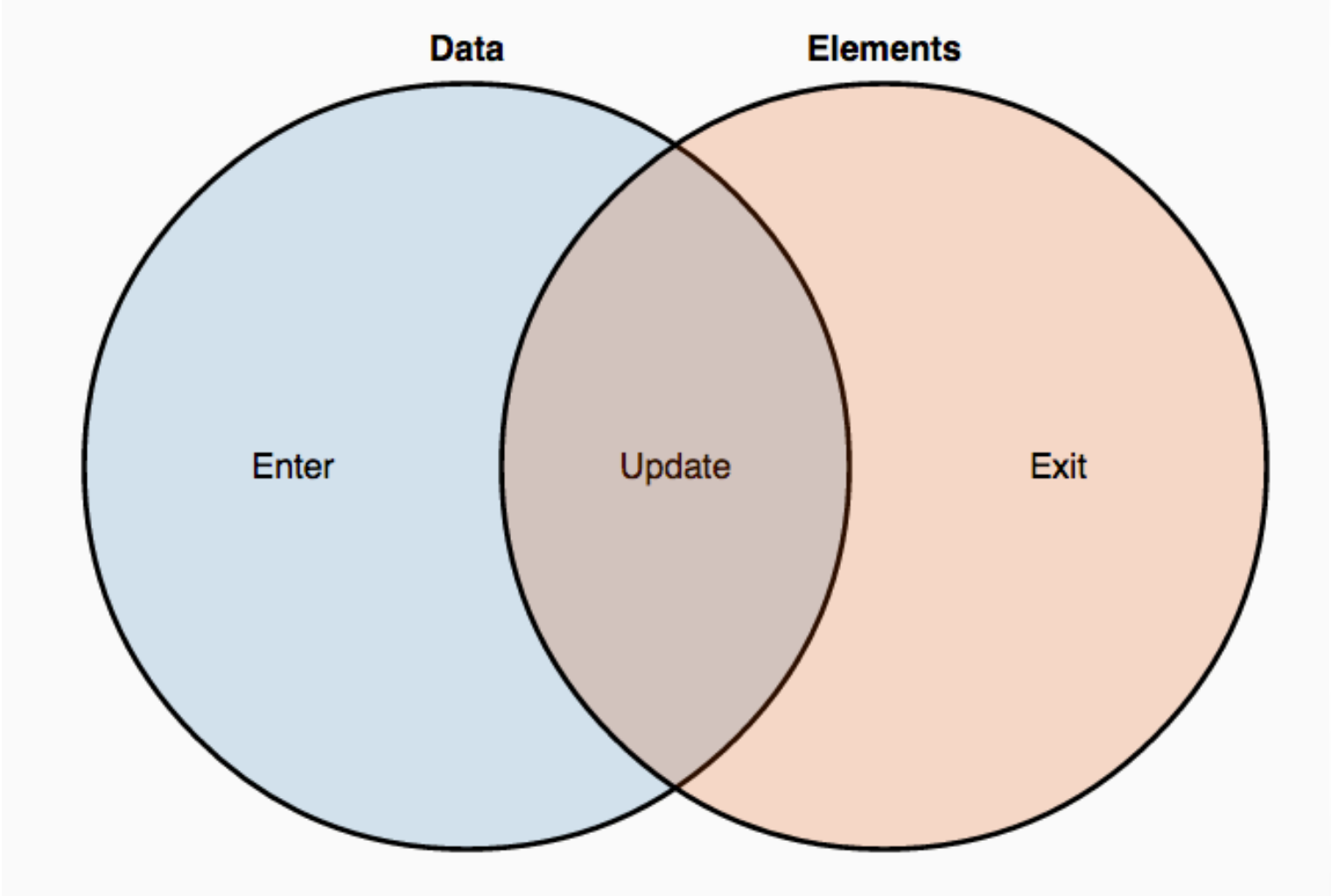
index: 0 1 2

Bind Data to Elements

```
                    0   1   2
var numbers = [10, 20, 5];
d3.selectAll("rect")
    .data(numbers)
    .attr("width", function(d) {return d * 30;});
```

```
0: <rect x="0" y="150" width="200" height="40" />
1: <rect x="0" y="200" width="200" height="40" />
2: <rect x="0" y="250" width="200" height="40" />
```

Thinking with Joins



Update

callBack parameters: function(value, index) {….};


```
var numbers= [10, 20, 5];
svg.selectAll("rect")
  .data(numbers)
  .attr("x", 0)
  .attr("y", function(d, i) { return i * 60 + 150; })
  .attr("width", function(d, i) { return d * 20; })
  .attr("height", 50)
  .style("fill", "steelblue")
```

Enter()

0: <rect x="0" y="150" width="200" height="40" />

1: <rect x="0" y="200" width="200" height="40" />

2: <rect x="0" y="250" width="200" height="40" />

there are not enough rects!

```
                                0     1     2     3     4
var numbers= [10, 20, 5, 15, 7];
```

```
d3.selectAll("rect")
   .data(numbers)
.enter().append("rect")…
```

Enter + Update

```
var selection = svg.selectAll("rect").data(numbers);
//enter
selection.enter().append("rect");
//update
selection
  .attr("x", 0)
  .attr("y", function(d, i) { return i * 60 + 150; })
  .attr("width", function(d, i) { return d * 20; })
  .attr("height", 50)
  .style("fill", "steelblue")
```

Exit()

0: <rect x="0" y="150" width="200" height="40" />
1: <rect x="0" y="200" width="200" height="40" />
2: <rect x="0" y="250" width="200" height="40" />

there are more rects than data

0 1

var numbers = [10, 5];


```
var selection = svg.selectAll("rect").data(numbers);

//exit
selection.exit().remove();
```

Key Functions

```
var data = [
  {name: "Alice", x: 10.0, y: 9.14},
  {name:   "Bob", x:  8.0, y: 8.14},
  {name: "Carol", x: 13.0, y: 8.74},
  {name:  "Dave", x:  9.0, y: 8.77},
  {name: "Edith", x: 11.0, y: 9.26}
];

function key(d) { return d.name; }

var circle = svg.selectAll("circle")
    .data(data, key)
    .attr("cx", function(d) { return d.x; })
    .attr("cy", function(d) { return d.y; })
    .attr("r", 2.5);
```

live DEMO

transition() + duration()

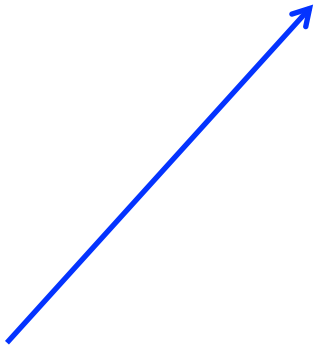
```
svg.selectAll("rect")
  .data(numbers)
  .transition()
  .duration(3000) // 3 seconds
    .attr("x", 0)
   .attr("y", function(d, i) { return i * 60 + 150; })
    .attr("width", function(d, i) { return d * 20; })
    .attr("height", 20)
    .style("fill", "steelblue")
```

Loading Data - CSV

```
var format = d3.time.format("%b %Y");
```

stocks.csv

```
symbol,date,price
S&P 500,Jan 2000,1394.46
S&P 500,Feb 2000,1366.42
S&P 500,Mar 2000,1498.58
S&P 500,Apr 2000,1452.43
...
...
```



```
d3.csv("stocks.csv", function(stocks) {
        stocks.forEach(function(d) {
                d.price = +d.price;
                d.date = format.parse(d.date);
        });
});
```

Loading Data - JSON

```
var format = d3.time.format("%b %Y");
```

stocks.json

```
[{"symbol": "S&P 500", "date": "Jan 2000", "price": 1394.46},
 {"symbol": "S&P 500", "date": "Feb 2000", "price": 1366.42},
 {"symbol": "S&P 500", "date": "Mar 2000", "price": 1498.58},
 ...
 ...
]
```

```
d3.json("stocks.json", function(stocks) {
        stocks.forEach(function(d) {
                d.price = +d.price;
                d.date = format.parse(d.date);
        });
});
```

Javascript arrays methods

```
function isBigEnough(value) {
  return value >= 10;
}
var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtered is [12, 130, 44]
```

```
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
// roots is now [1, 2, 3], numbers is still [1, 4, 9]
```

```
var fruit = ['cherries', 'apples', 'bananas'];
fruit.sort();
// fruit is now ['apples', 'bananas', 'cherries']
```

SVG Path

```
<svg width="100" height="100">
  <path d=" M 10 25
          L 10 75
          L 60 75
          L 10 25”
          stroke="red" stroke-width="2" fill="none" />
</svg>
```

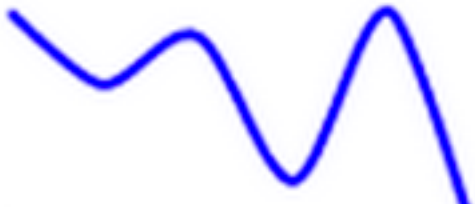
- M 10 25 - Put the pen down at 10 25
- L 10 75 - Draw a line to the point 10 75, from the previous point
- L 60 75 - Draw a line to the point 60 75, from the previous point
- L 10 25 - Draw a line to the point 10 25, from the previous point

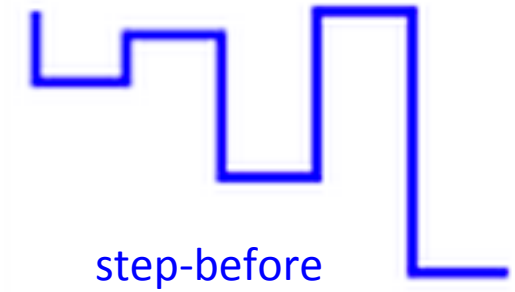
D3.svg.line()

```
//The data for our line
var lineData = [ { "x": 1,   "y": 5},  { "x": 20,  "y": 20}, { "x": 40,  "y": 10},
        { "x": 60,  "y": 40},{ "x": 80,  "y": 5},  { "x": 100, "y": 60}];

//This is the accessor function we talked about above
var lineFunction = d3.svg.line()
                    .x(function(d) { return d.x; })
                    .y(function(d) { return d.y; })
                    .interpolate("linear");

//The line SVG Path we draw
var lineGraph = svg.append("path")
                    .attr("d", lineFunction(lineData))
                    .attr("stroke", "blue")
                    .attr("stroke-width", 2)
                    .attr("fill", "none");
```

linear

cardinal





step-before

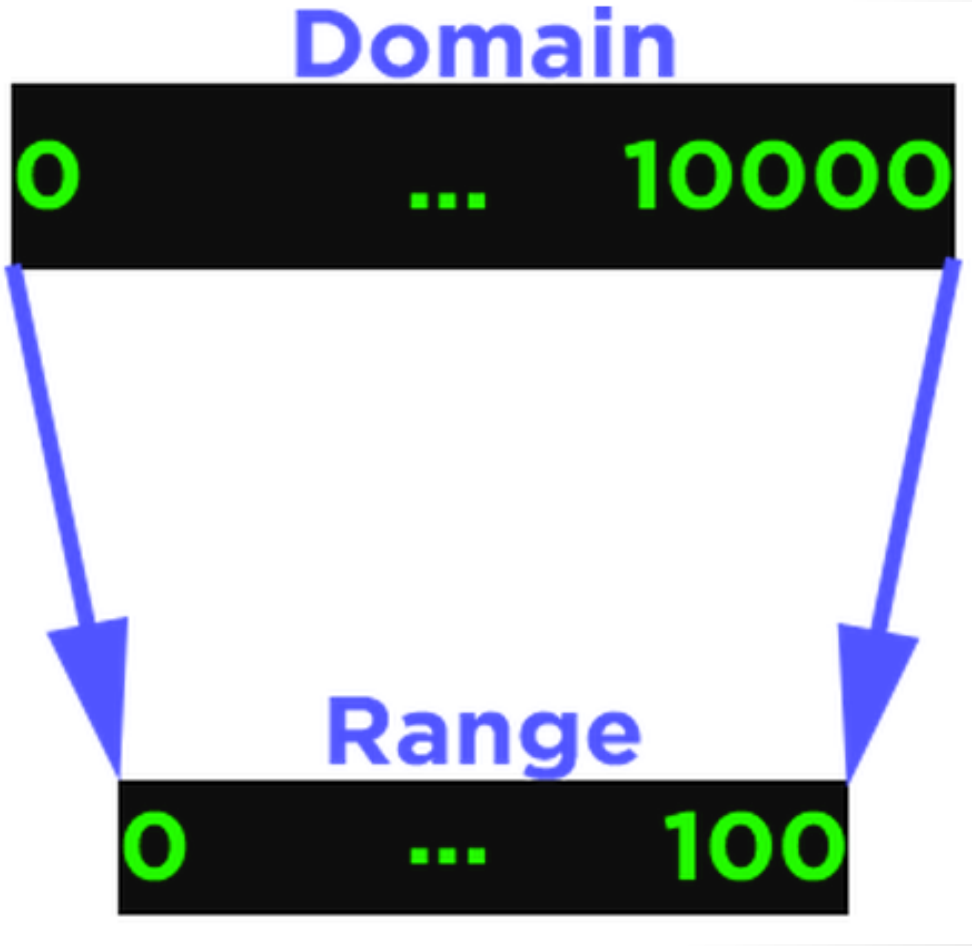
Other Path generators

- d3.svg.line - create a new line generator
- d3.svg.line.radial - create a new radial line generator
- d3.svg.area - create a new area generator
- d3.svg.area.radial - create a new radial area generator
- d3.svg.arc - create a new arc generator
- d3.svg.symbol - create a new symbol generator
- d3.svg.chord - create a new chord generator
- d3.svg.diagonal - create a new diagonal generator
- d3.svg.diagonal.radial - create a new radial diagonal generator

Scales

Scales are functions that map from data-space to visual-space.

- Identity
- Linear
- Power and Logarithmic
- Quantize and Quantile
- Ordinal



Quantitative Scales

- Map a continuous (numeric) domain to a continuous range.

```
var x = d3.scale.linear()
    .domain([12, 24])
    .range([0, 720]);
x(16); // 240
```

or log()

```
var x = d3.scale.sqrt()
    .domain([12, 24])
    .range([0, 720]);
x(16); // 268.9056992603583
```

Domains & Ranges

- Typically, domains are derived from data while ranges are constant.

```
var x = d3.scale.linear()
    .domain([0, d3.max(numbers)])
    .range([0, 720]);
```

```
var x = d3.scale.log()
    .domain(d3.extent(numbers))
    .range([0, 720]);
```

```
function value(d) { return d.value; }

var x = d3.scale.log()
    .domain(d3.extent(objects, value))
    .range([0, 720]);
```

Interpolators

```
var x = d3.scale.linear()
    .domain([12, 24])
    .range(["steelblue", "brown"]);

x(16); // #666586
```

```
var x = d3.scale.linear()
    .domain([12, 24])
    .range(["0px", "720px"]);

x(16); // 240px
```

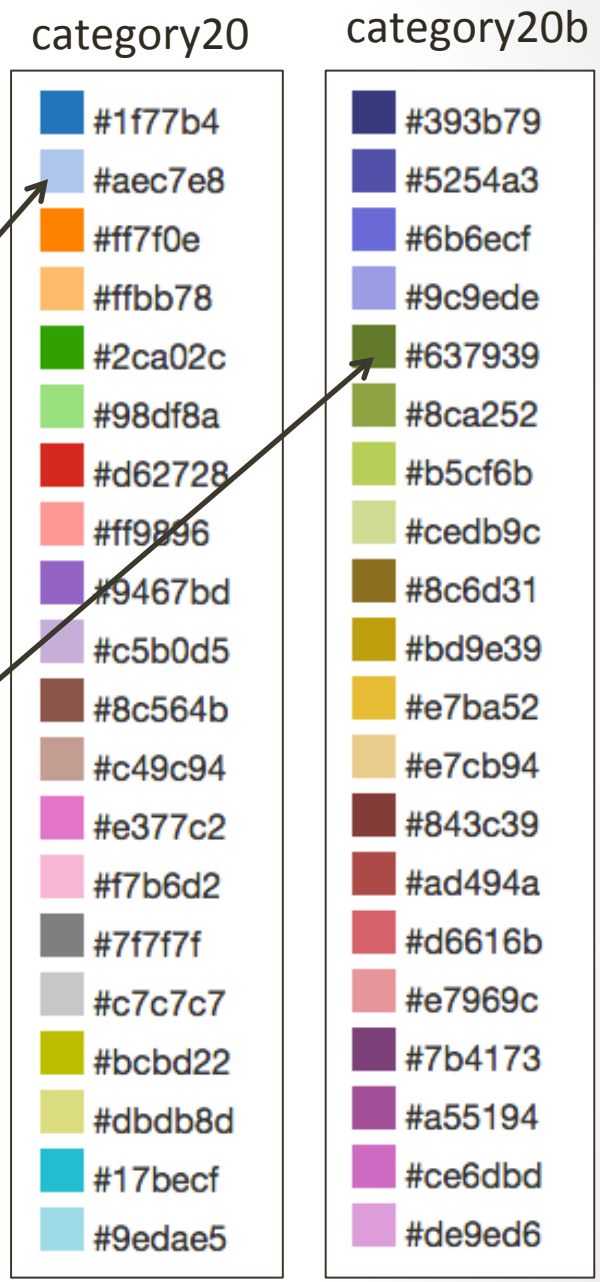
```
var x = d3.scale.linear()
    .domain([12, 24])
    .range(["steelblue", "brown"])
    .interpolate(d3.interpolateHsl);

x(16); // #3cb05f
```

Ordinal Scales

```
var x = d3.scale.ordinal()
    .domain(["A", "B", "C", "D"])
    .range([0, 10, 20, 30]);

x("B"); // 10
```

```
var x = d3.scale.category20()
    .domain(["A", "B", "C", "D"]);

x("B"); // #aec7e8
```

```
var x = d3.scale.category20b()
    .domain(["A", "B", "C", "D"]);

x("E"); // #637939
x.domain(); // A, B, C, D, E
```

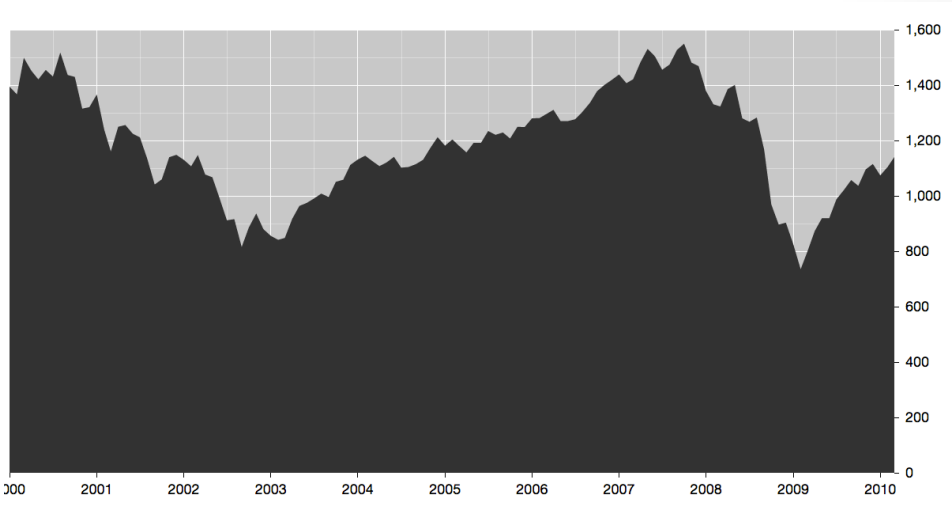


d3.svg.axis()

```
var y = d3.scale.linear()
        .domain([0,100])
        .range([0,100]);

var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left");

svg.append("g")
    .attr("class", "y axis")
    .call(yAxis);
```

CSS

```
.axis path, .axis line {
  fill: none;
  stroke: #000;
  shape-rendering: crispEdges;
}
```



Useful resources

- https://www.dashingd3js.com/

- http://vadim.ogievetsky.com/IntroD3/
- https://github.com/mbostock/d3/wiki/API-Reference

- Tutorial by Mike Bostok
- http://bost.ocks.org/mike/d3/workshop/


KEEP
CALM
AND
ASK
QUESTIONS