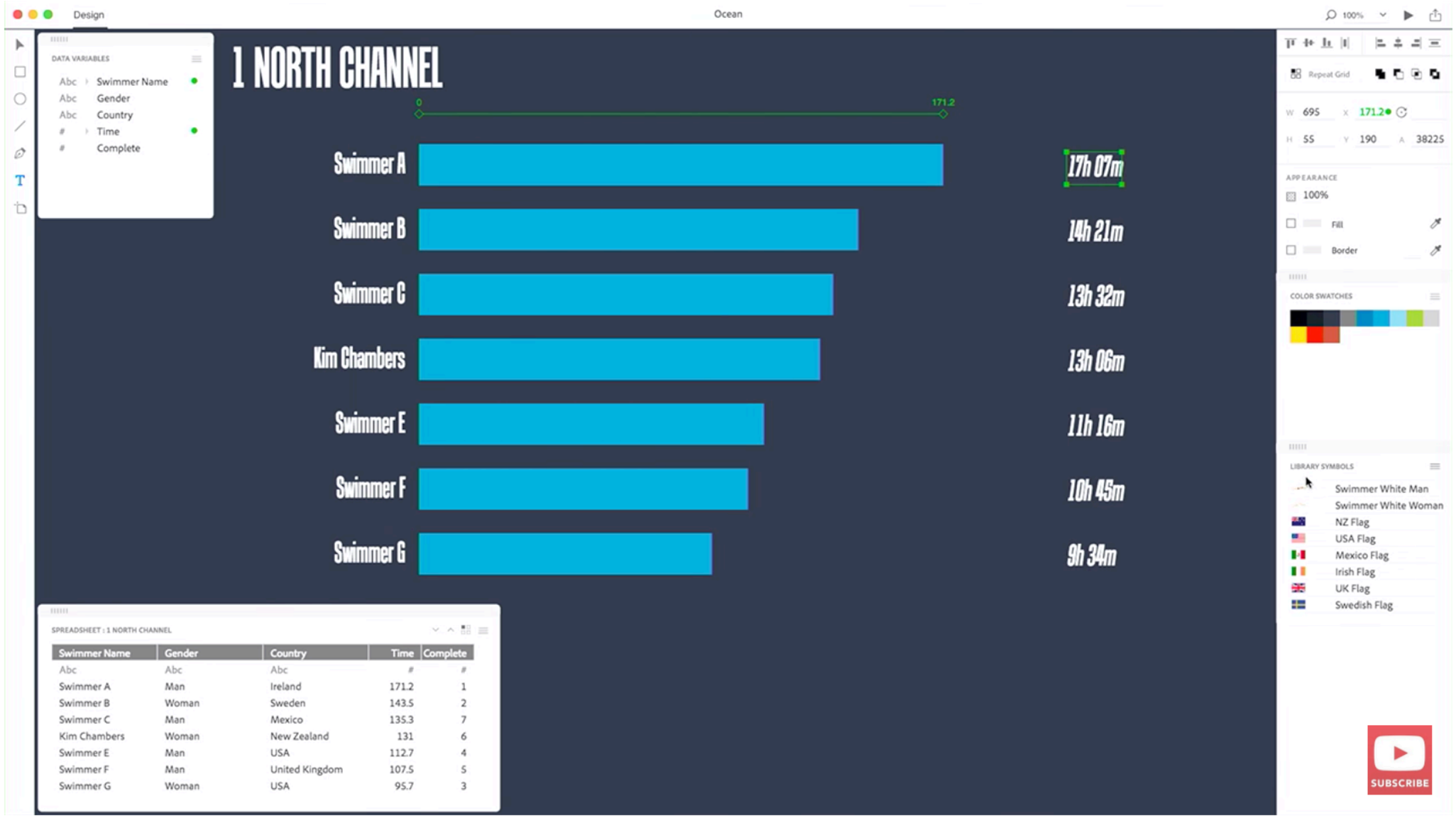
Data Visualization & Design

Week 8

This week in **visualization —**



SPREADSHEET : 1 NORTH CHANNEL

Swimmer Name	Gender	Country	Time	Complete
Abc	Abc	Abc	#	#
Swimmer A	Man	Ireland	171.2	1
Swimmer B	Woman	Sweden	143.5	2
Swimmer C	Man	Mexico	135.3	7
Kim Chambers	Woman	New Zealand	131	6
Swimmer E	Man	USA	112.7	4
Swimmer F	Man	United Kingdom	107.5	5
Swimmer G	Woman	USA	95.7	3

#ProjectLincoln

Interactive visualization — some **concepts**

Filtering

- Analysts rarely analyze the entirety of a dataset at once
- Typically, a presentation is comprised of visualizations that represent selected data dimensions
- Given an overview of dimensions, analysts often want to shift their focus to different subsets (ex. time slices, or particular categories)

Filtering — *Interaction techniques*

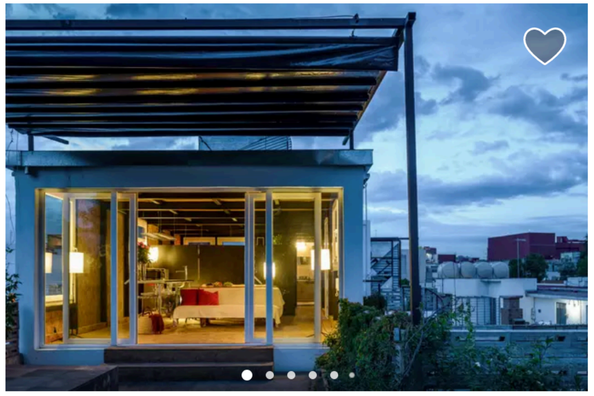
- **User interface elements** (“dynamic query widgets,” dropdowns, radio buttons, checkboxes, etc.)
- Lasso (direct selection)

Anywhere | Anytime | 1 guest

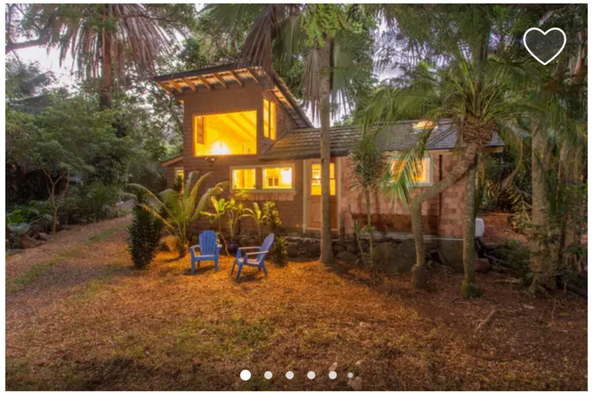
Trips | Messages | Help | Become a host

FOR YOU | HOMES | EXPERIENCES | RESTAURANTS

Room type | Price range | Instant Book | More filters



$84 Apartment 1 of 4 with green terrace in Roma Norte
Entire loft · 2 beds
★★★★★ 265 · Superhost



$175 Adorable Garden Gingerbread House
Entire cabin · 1 bed
★★★★★ 188 · Superhost

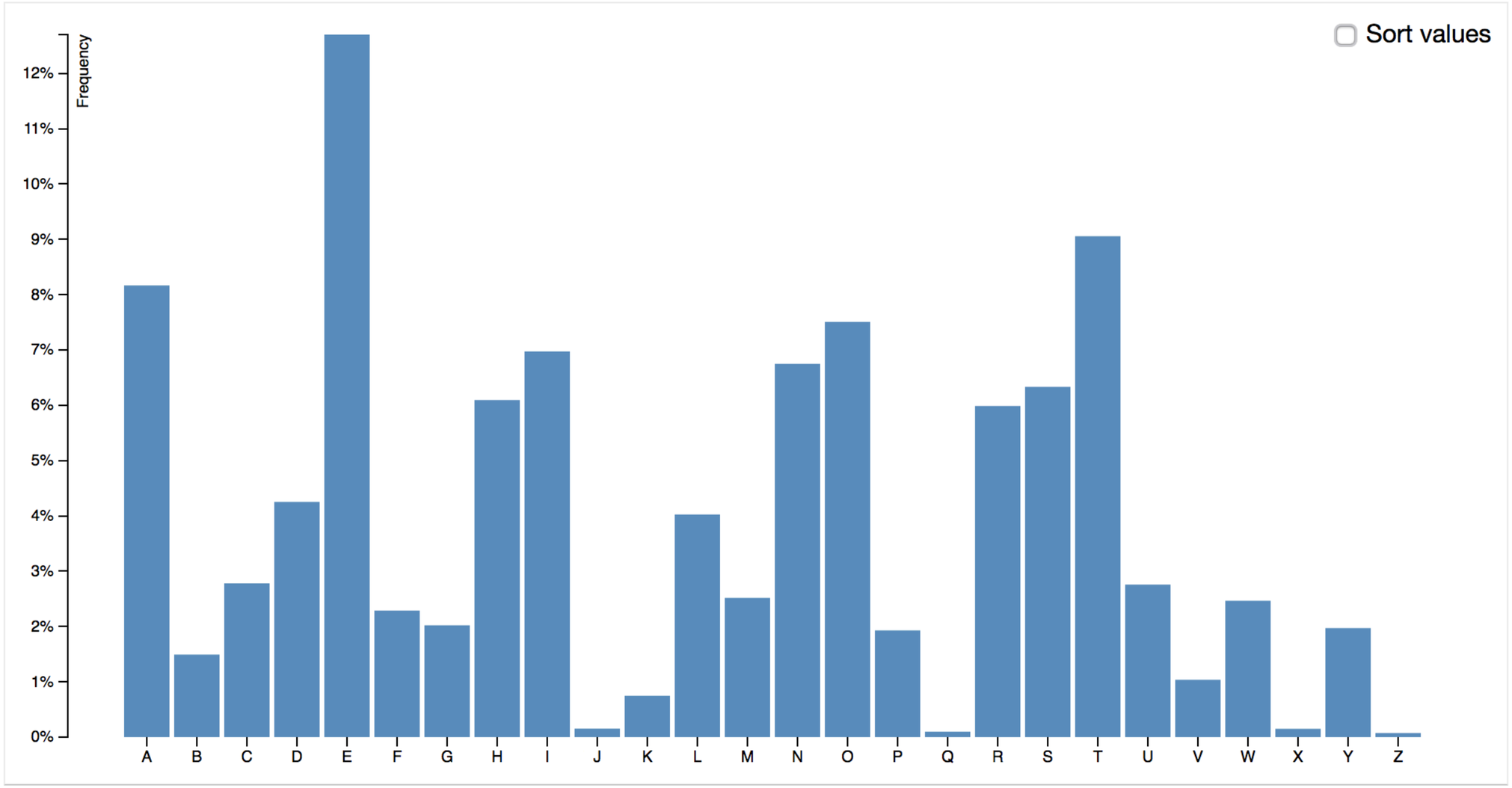
[Example: Airbnb]

Sorting

- Fundamental operation within a visualization
- Enabling the user to **order** values allows for:
    - More easily identifying clusters or trends
    - Reconciling the data within familiar units of analysis (days of the week, financial quarters, etc.)

Sorting — *Interaction techniques*

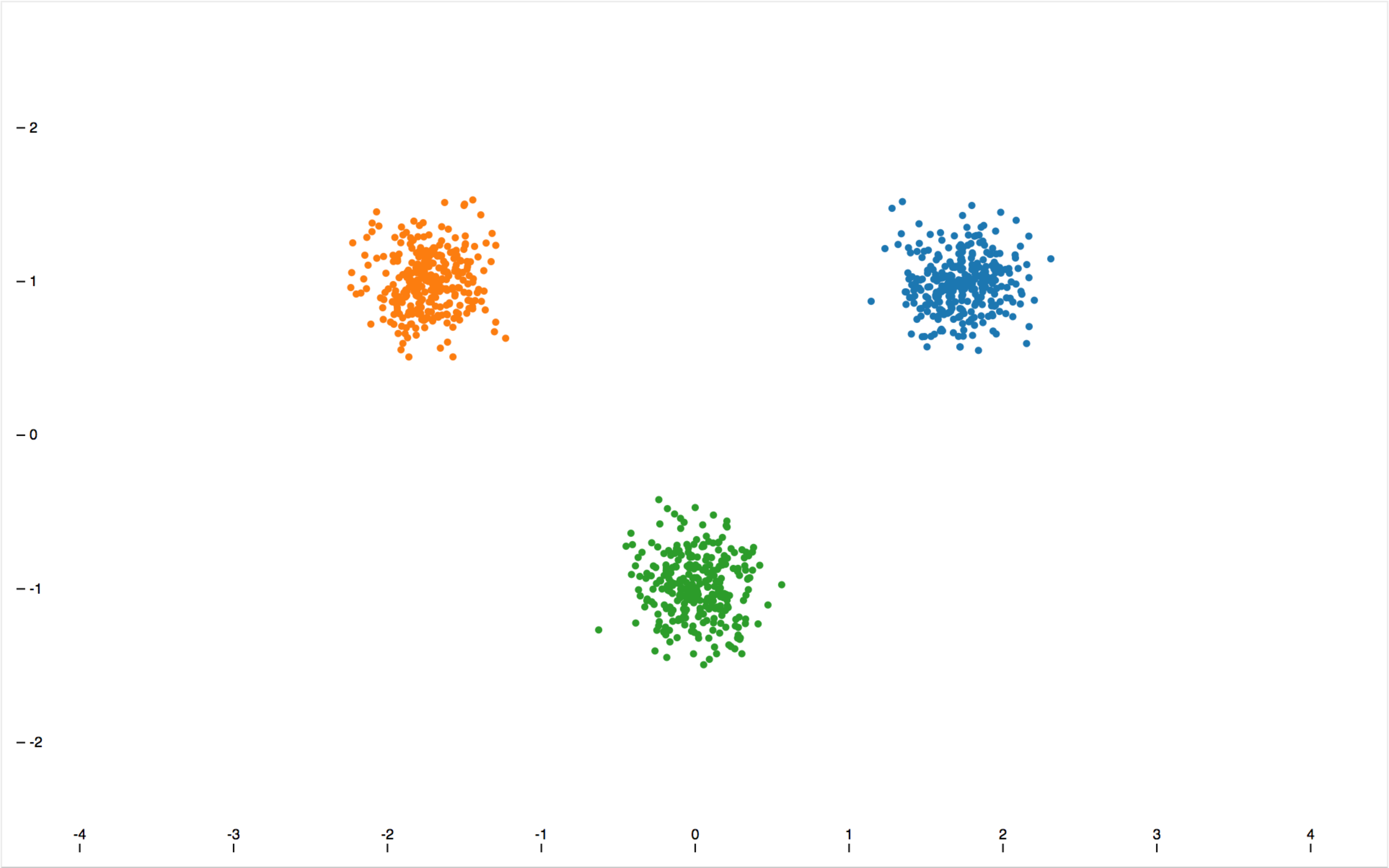
- Choices in a toolbar
- Clicking the header of a table
- User interface elements


Sort values
Frequency
12%
11%
10%
9%
8%
7%
6%
5%
4%
3%
2%
1%
0%
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Example: D3

Brushing and linking

- Address the **connections** between two or more views of the same data
- **Linking —** A change in one view produces a change in the other (i.e. changing filter parameters)
- **Brushing —** Highlighting data to focus a single view or highlight that data in other views



Example: D3

Selection

- Closely related to filtering
- Determines a set of objects to be manipulated or examined further

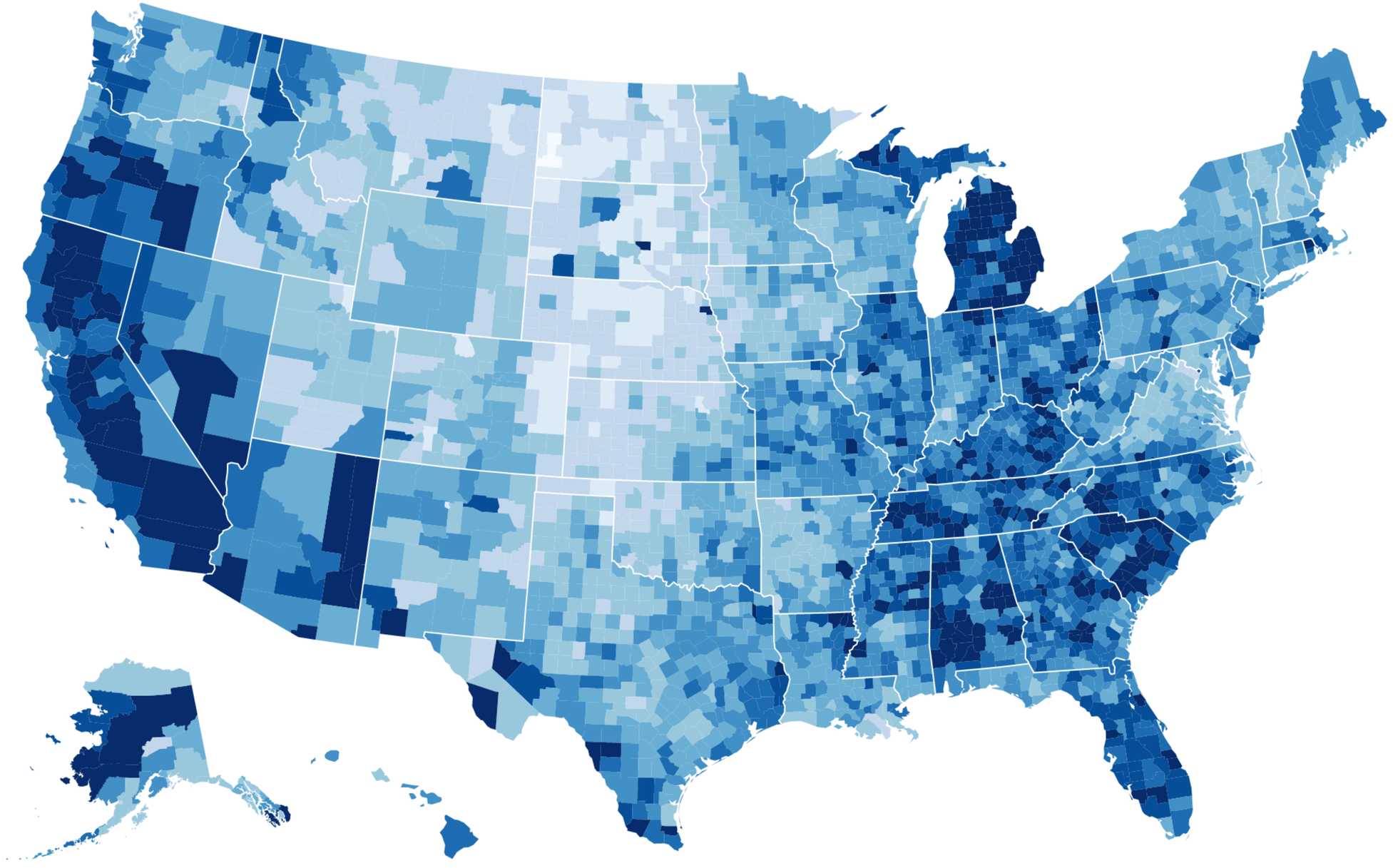
Introduction to **D3.js** —

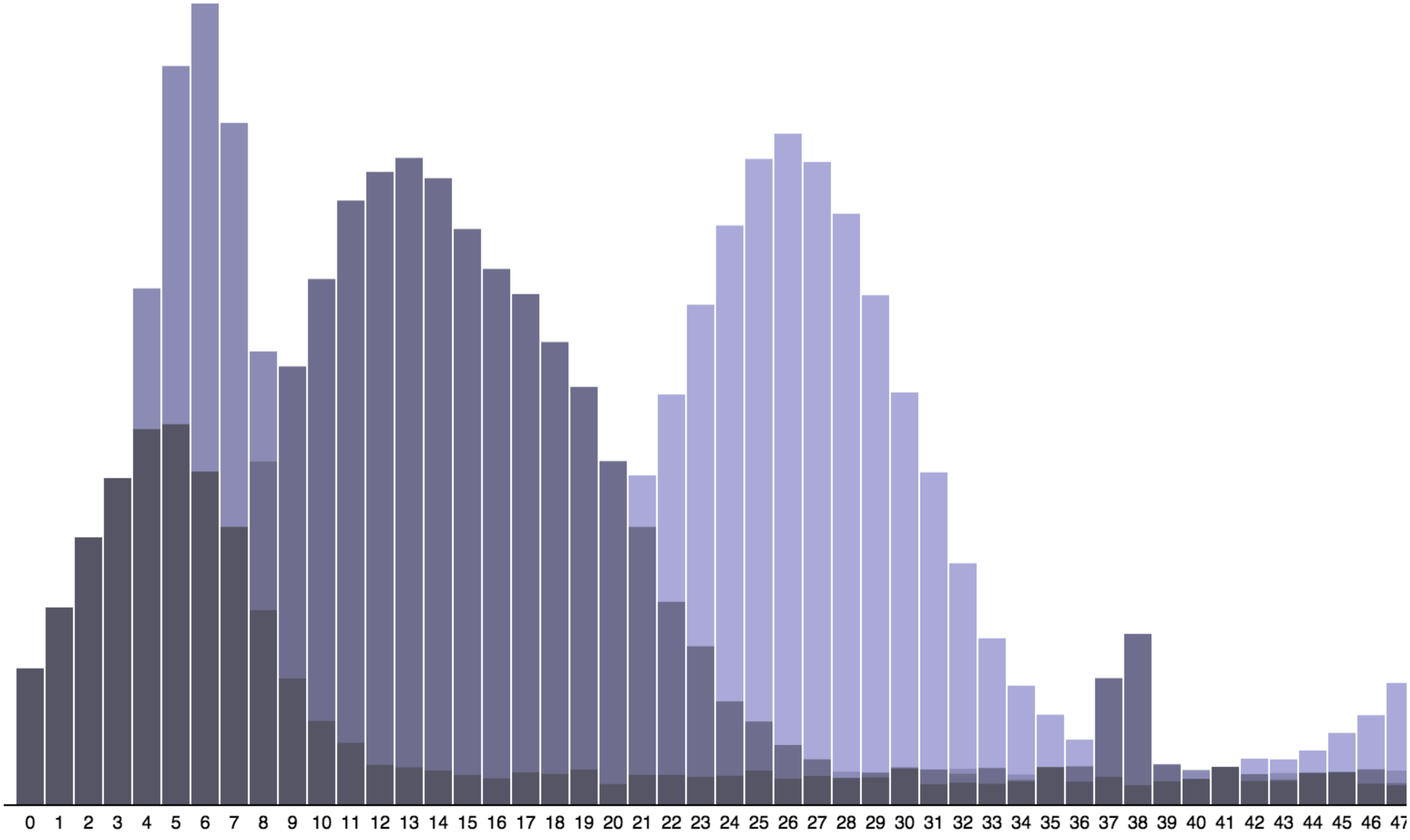
What is D3?

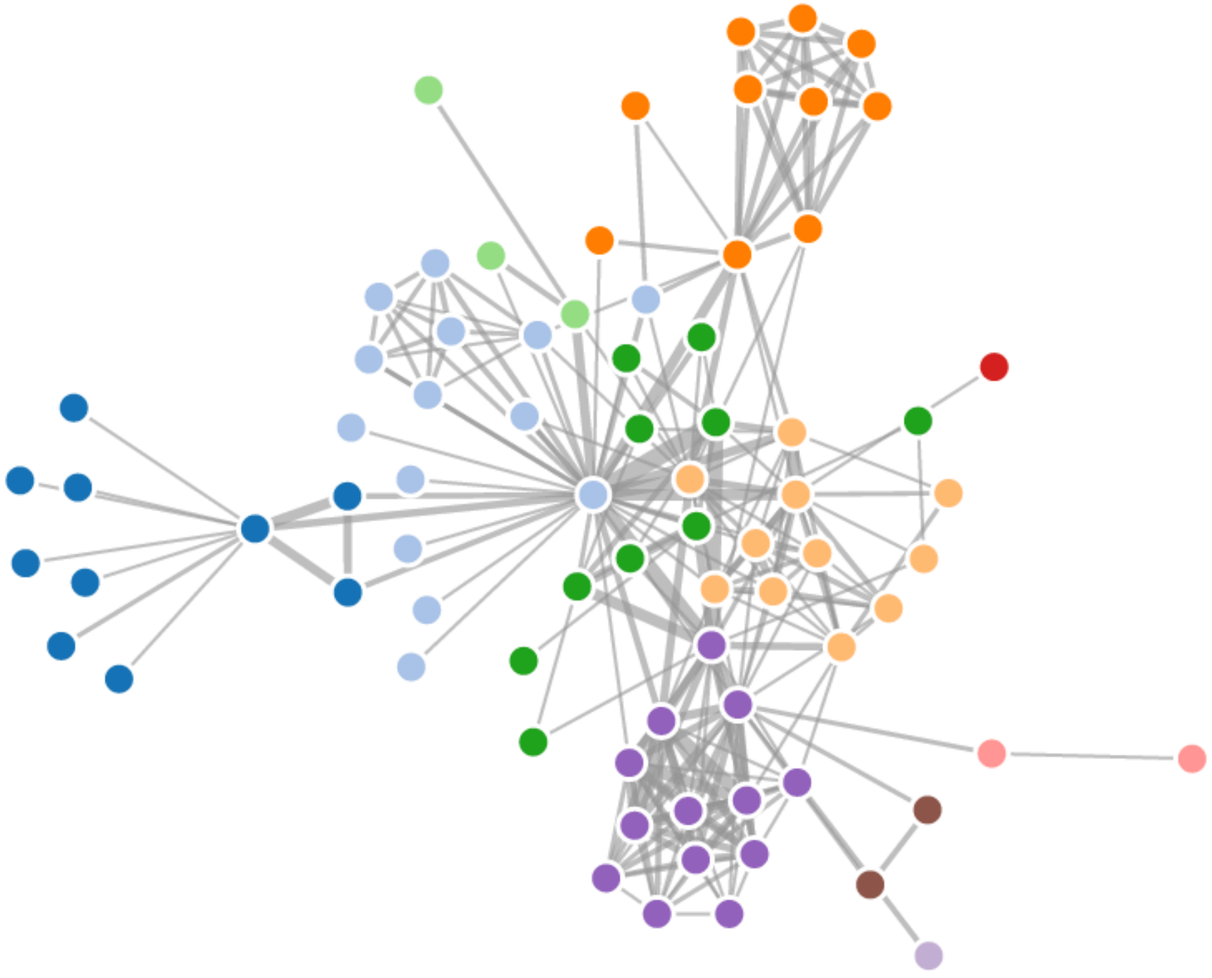
A **general-purpose visualization library** for HTML and SVG.

Efficiently **transform data into elements** in a browser.

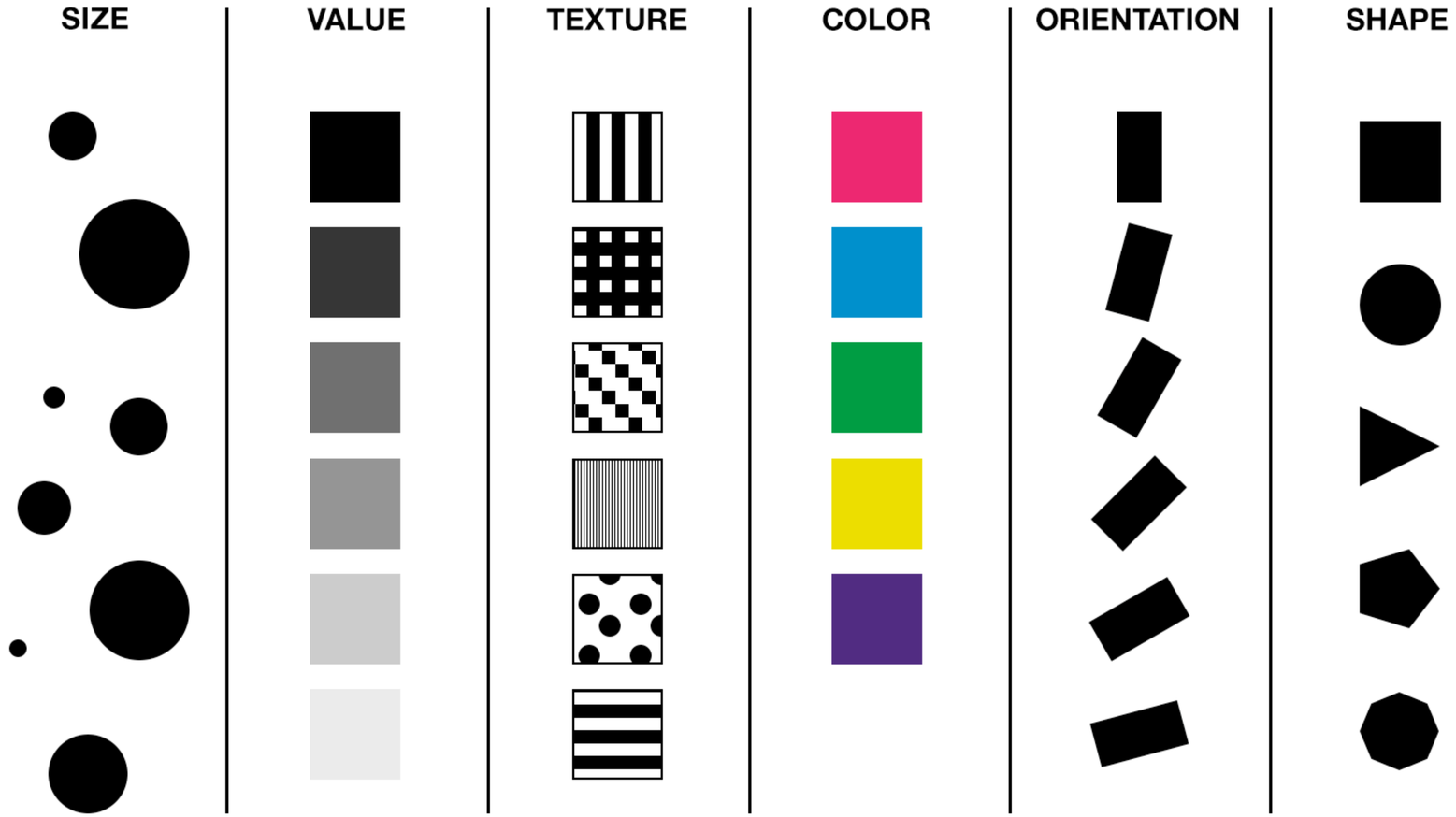
Visualizations are comprised of **basic graphical elements**…




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47



. . . with **attributes**



Jacques Bertin: six retinal properties

SVG is a **DOM for graphics**.

(retina-ready, too)

SVG primitives—

Rectangle

Circle

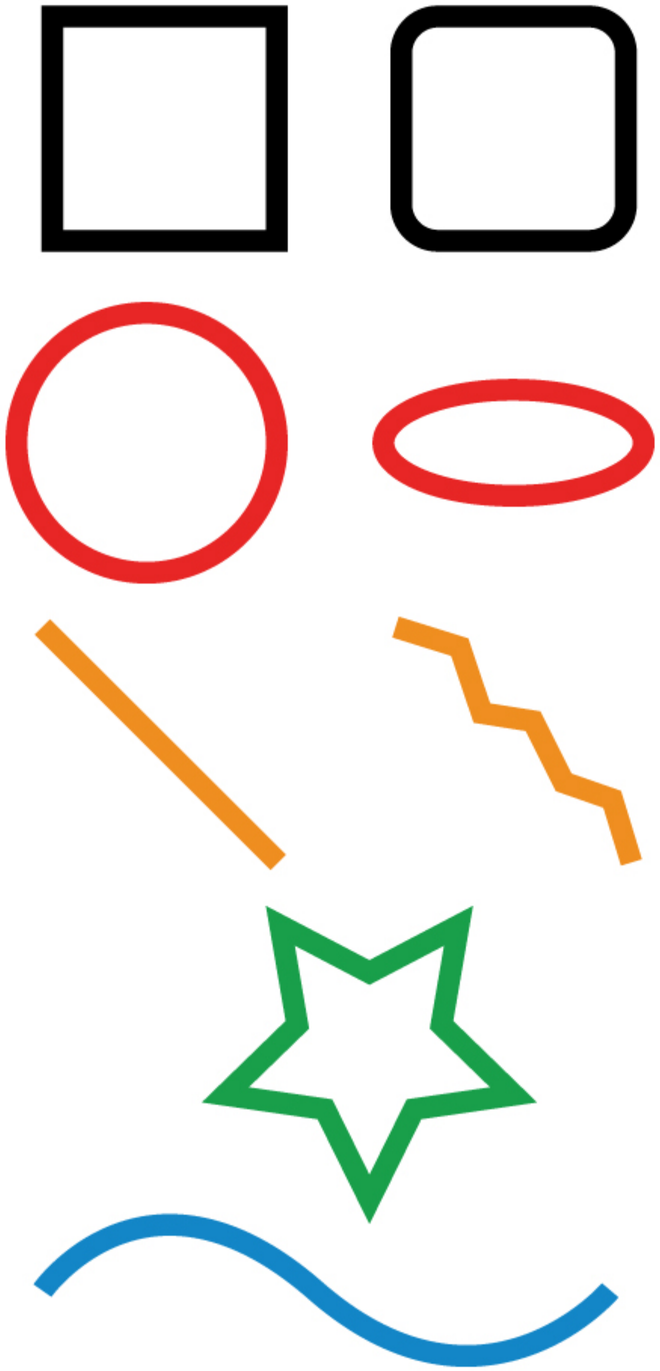
Ellipse

Line

Polyline

Polygon

Path



When is D3 **appropriate**?

- **Web-based** visualizations
- **Highly custom** visualizations (i.e. cannot be replicated by a charting library)
- Visualizations that need to **update frequently**

JS, data types, & JSON.

JavaScript (JS) is a **programming language** that enables you to add interactive features to your website.

HTML skeleton, CSS skin, JS brain.

String

Number

Boolean

Array

Object

STRING

A series of characters enclosed in double- or single-quotes.

```
var x = "hello”
var x = "34"
```

NUMBER

Integer or float.

```
var x = 34
var x = 34.00
```

BOOLEAN

True or false.

```
var x = true
var x = false
```

ARRAY

Square brackets. Items separated by commas.

```
var x = [1, 2, 3]
var x = ["this", "that"]
```

OBJECT

Curly brackets. Properties are written as name/value pairs, separated by commas.

```
var x = {
  "name":"bob",
  "age":35,
  "location":"NY"
}
```

JSON is a syntax for **data storage and exchange**.

JSON syntax is **derived** from JavaScript syntax.

JSON data is written in name/value pairs.

```
"city":"new york"
```

JSON objects are stored inside curly brackets…

```
{"city":"new york"}
```

…which can contain multiple name/value pairs.

```
{"city":"new york", "state":"NY"}
```

JSON arrays are written inside square brackets.

```
"locations":[
   {"city":"new york", ”state":"NY"},
   {"city":"los angeles", ”state":"CA"},
   {"city":"chicago", ”state":"IL"}
]
```

To work with JSON in JS, create an array and assign data to it.

```
var locations = [
   {"city":"new york", ”state":"NY"},
   {"city":"los angeles", ”state":"CA"},
   {"city":"chicago", ”state":"IL"}
];
```

In **Excel**, that data would look like this.

	A	B	C
1	**city**	**state**	
2	new york	NY	
3	los angeles	CA	
4	chicago	IL	
5			
6			
7			
8			
9			

Elements in an array can be accessed using their **position** (index).

```
locations[0]
{"city":"new york", ”state":"NY"}

locations[2]
{"city":"chicago", ”state":"IL”}
```

D3: a closer look.

D3 magic: **The Join**

- Pairs a data object with an element.
- Keeps track of new and old objects.
- Lets you animate differences between new and old.

Pie chart example:

http://bl.ocks.org/dbuezas/9572040

enter, update, exit

Start with a selection.

```
//this is an empty selection
//looks for instantiations of data

var elements = d3.selectAll('div')
```

Join selected elements with data items.

```
var elements = d3.selectAll('div')
    .data([1,2,3]);
```

enter()

For every part of the data that does not correspond to an existing element, add an element.

```
elements.enter().append('div');
```

(update)

For each element in the selection,
update attributes.

```
elements
   .attr('background','red');
```

exit()

Remove elements that no longer correspond to the data.

```
elements.exit().remove();
```

→ data bound to the DOM.

Each element has a **property** that **stores the data**.

The __**data**__ property.

```
46
47  var circles = svg                          //select the ele
48      .selectAll("circle.testNode")          //here we select
49      .data(sampleData);                     //we want the sa
50  circles
51      .enter()
52      .append("circle")                          //append
53      .classed("testNode",true);                 //assign
54  circles
55      .attr("cx",function(d,i){                  //"cx" f
56          return (i+1)*250;                      //for ea
57      })
58      .attr("cy",h/2)                            //"cy" f
59      .attr("r",function(d {debugger; return d; });
60  circles.exit().remove();
61
```

```
51      .enter()
52      .append("circle")
53      .classed("testNode",tr
54  circles
55      .attr("cx",function(d,
56          return (i+1)*250;
57      })
58      .attr("cy",h/2)
59      .attr("r",function(d){
60  circles.exit().remove();
61
```

(anonymous function) vis.js:59

▼ Scope Variables

▼ Local

d: 20

▼ this: circle

__data__: 20

▶ attributes: NamedNode

```
// d3 has a few different
// functions that set stuff

.text()
.property()
.style()
.attr()
```

```
// each takes a function

.attr('foo', function() { })
```

```
// and that function gets data
// from your .data()

.attr('foo', function(d) {
  return d.foo;
})
```

You can chain these, as well as use them to pull an existing value.

Let’s draw some shapes!

Setup

```
//append an SVG element
//to the div with ID “vis”

var svg = d3.select("#vis")
   .append("svg")
```

```
//set the width and height

var svg = d3.select("#vis")
    .append("svg")
    .attr("width",w)
    .attr("height",h);
```

SVG circles have a **center point** position and a **radius**.

`(cx, cy, r)`

```
//full code

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
   .append('circle')
   .attr('cx',500)
   .attr('cy',300)
   .attr('r',function(d){
      return d;
   });
```

```
//create an empty selection
//this looks for instantiations of data

var circles = svg.selectAll('circle')
```

```
//this is data, which
//would be bound to a selection

var circles = svg.selectAll('circle')
   .data([50,100,150])
```

```
//ENTER: for every time we see data,
//but do not see a corresponding element

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
```

```
//append an element

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
   .append('circle')
```

```
//set x-position and y-position

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
   .append('circle')
   .attr('cx',500)
   .attr('cy',300)
```

```
//finally, set circle radius
//based on value from the array

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
   .append('circle')
   .attr('cx',500)
   .attr('cy',300)
   .attr('r',function(d){
      return d;
   });
```

```
//full code

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
   .append('circle')
   .attr('cx',500)
   .attr('cy',300)
   .attr('r',function(d){
      return d;
   });
```

Only one?

```
//add a function to ‘cx’

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
   .append('circle')
   .attr('cx',function(d,i){
      return (i*300);
   })
   .attr('cy',300)
   .attr('r',function(d){
      return d;
   });
```

```
//tweak it

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
   .append('circle')
   .attr('cx',function(d,i){
      return (i*300) + 250;
   })
   .attr('cy',300)
   .attr('r',function(d){
      return d;
   });
```

A good template.


selection
enter()
attributes
interaction
exit()

selection

```
var circles = svg.selectAll(‘circle’)
  .data([50,100,150]);
```

enter()

```
circles.enter().append('circle');
```

attributes

```
circles
   .attr('cx',500)
   .attr('cy',300)
   .attr('r',function(d){
      return d;
   });
```

interaction

```
circles
   .on('mouseover',function(){});
```

exit()

```
circles.exit().remove();
```

```
var circles = svg.selectAll('circle')
   .data([50,100,150]);
```

```
circles.enter().append('circle');
```

```
circles
   .attr('cx',500)
   .attr('cy',500)
   .attr('r',function(d){
      return d;
   });
```

```
circles
   .on('mouseover',function(){})
```

```
circles.exit().remove();
```

Challenges

Challenge—

Give the circles a red stroke.

```
//red stroke code

var circles = svg.selectAll('circle')
   .data([50,100,150])
   .enter()
   .append('circle')
   .attr('cx',function(d,i){
      return (i*300) + 250;
   })
   .attr('cy',300)
   .attr('r',function(d){
      return d;
   })
   .style('stroke’,'red’);
```

Challenge—

Make the data drive the stroke-width of the circles.

```
//stroke-width code

var circles = svg.selectAll('circle')
    .data([50,100,150])
    .enter()
    .append('circle')
    .attr('cx',function(d,i){
       return (i*300) + 250;
    })
    .attr('cy',300)
    .attr('r',function(d){
       return d;
    })
    .style('stroke’,'red')
    .style('stroke-width',function(d,i){
       return (i*2);
    });
```

Challenge—

Make the circles rectangles.

SVG rectangles have an x-position, a y-position, width, and height.

`(x, y, width, height)`

```
//rectangle code

var rects = svg.selectAll('rect')
    .data([50,100,150])
    .enter()
    .append('rect')
    .attr('x',function(d,i){
       return (i*300) + 250;
    })
    .attr(‘y',300)
    .attr('width',function(d){
       return d;
    })
    .attr('height',function(d){
       return d*20;
    });
```

Challenge—

Make the dataset into an array of objects to set the radii and color of the circles.