

Data Visualization & Design

Week 6

Announcements

Next week, October 16th →
ONLINE MEETING

This week in **visualization**...

Territories Languages Treaties

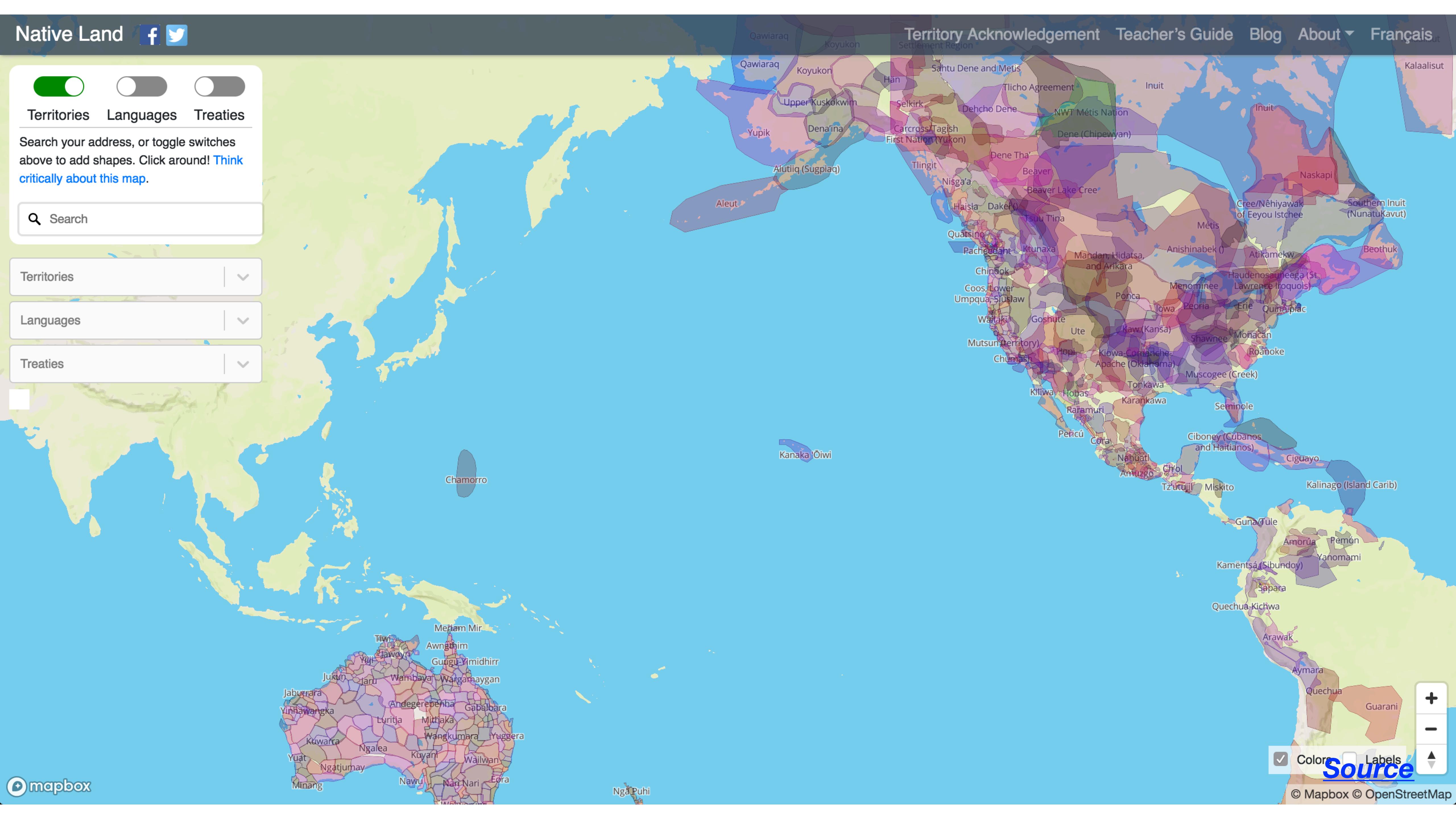
Search your address, or toggle switches above to add shapes. Click around! [Think critically about this map.](#)

 Search

Territories

Languages

Treaties



1. Concepts in **Interactive Visualization**
2. Introduction to **D3.js**
3. D3.js **Studio**

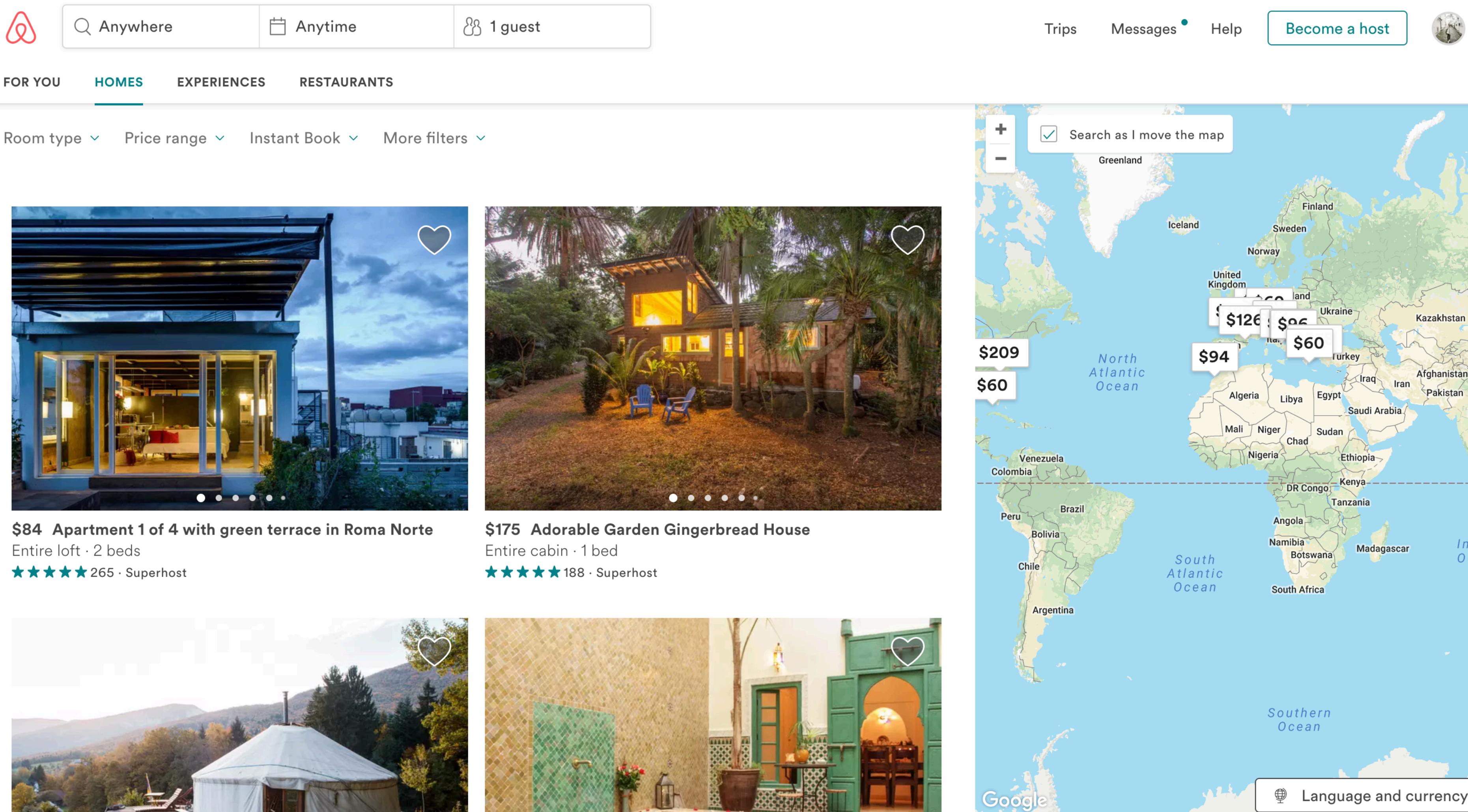
1. Concepts in **Interactive Visualization**
2. Introduction to **D3.js**
3. D3.js **Studio**

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)



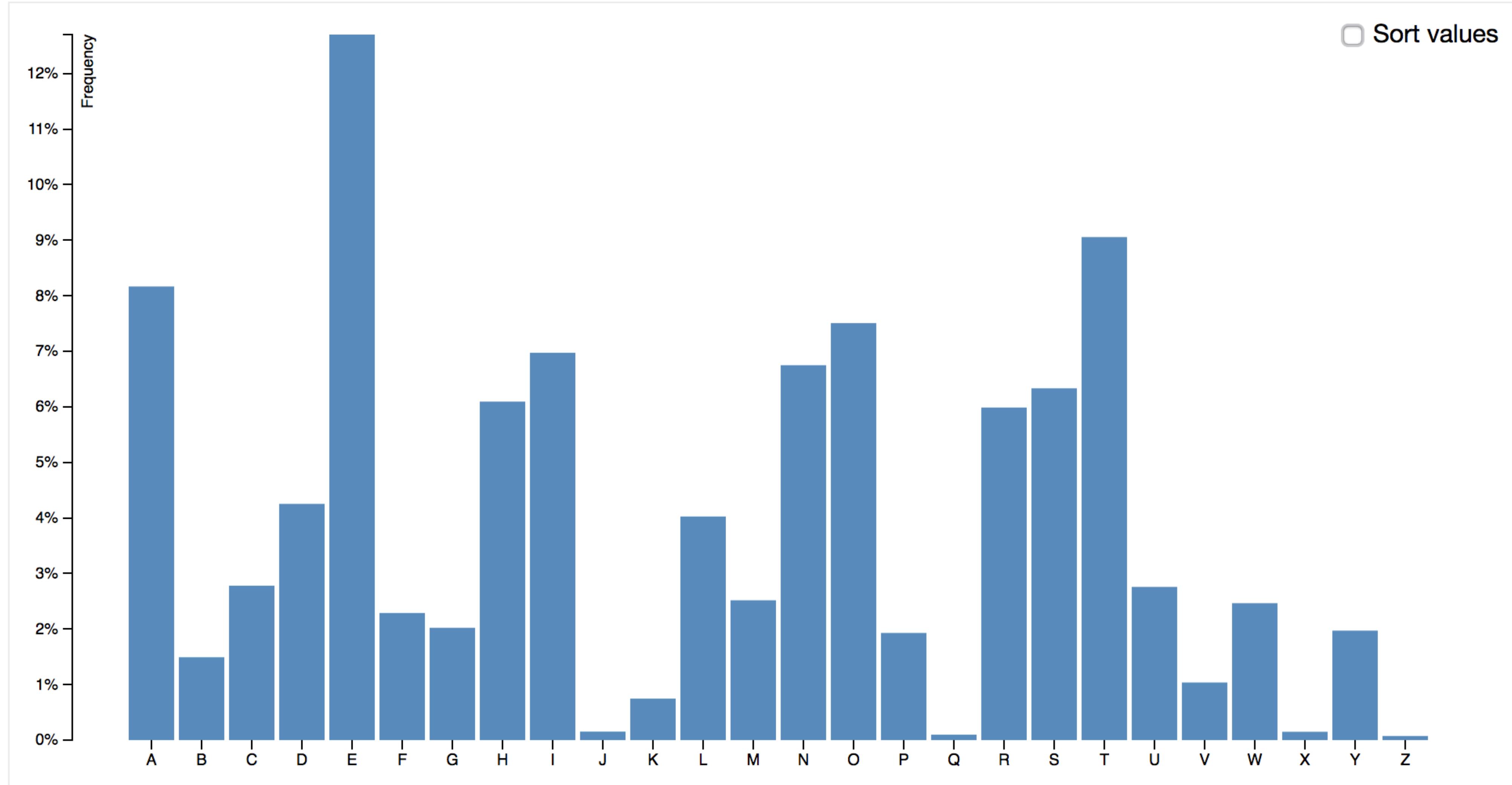
Example: Airbnb

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*

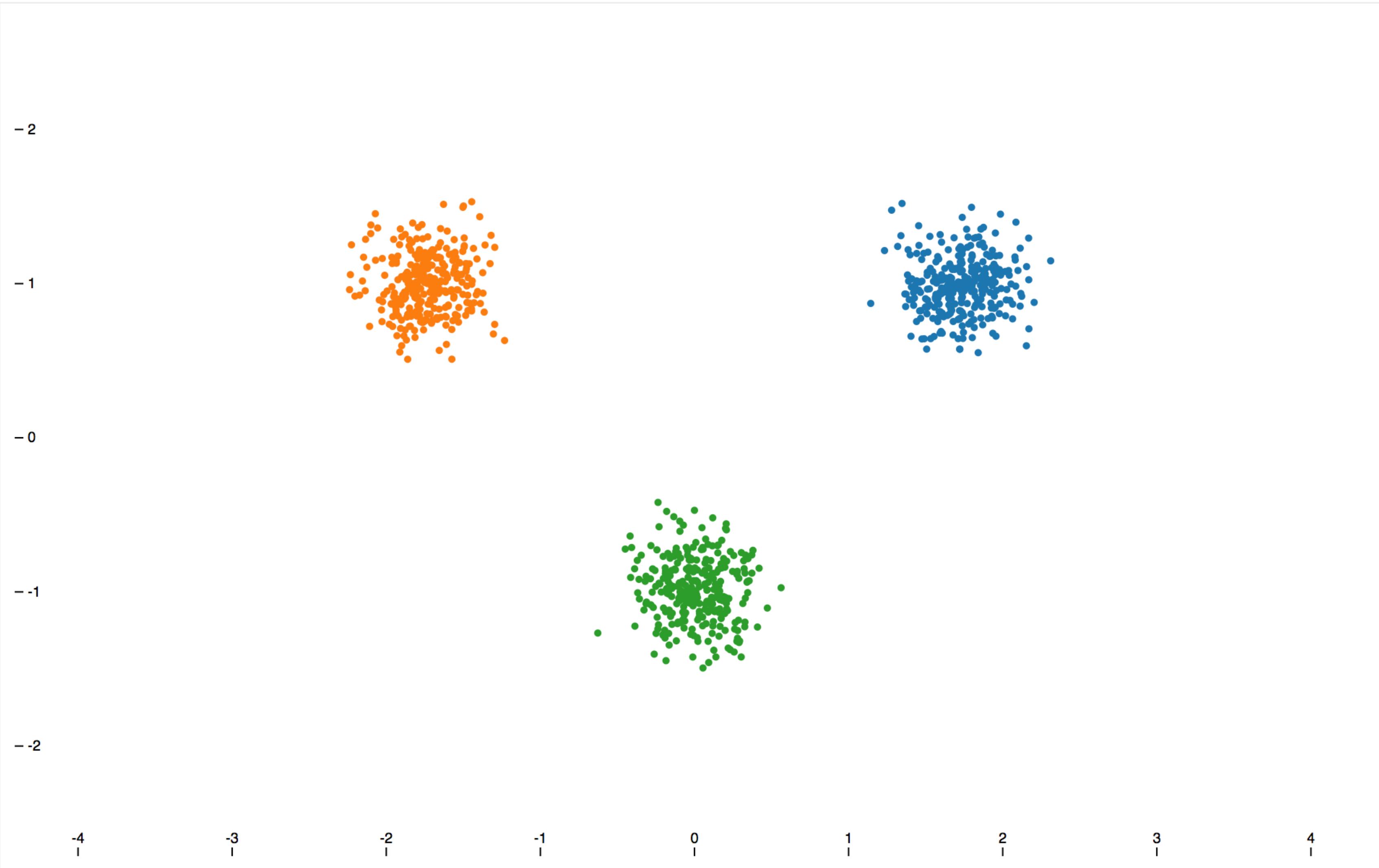
- Choices in a toolbar
- Clicking the header of a table
- User interface elements



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

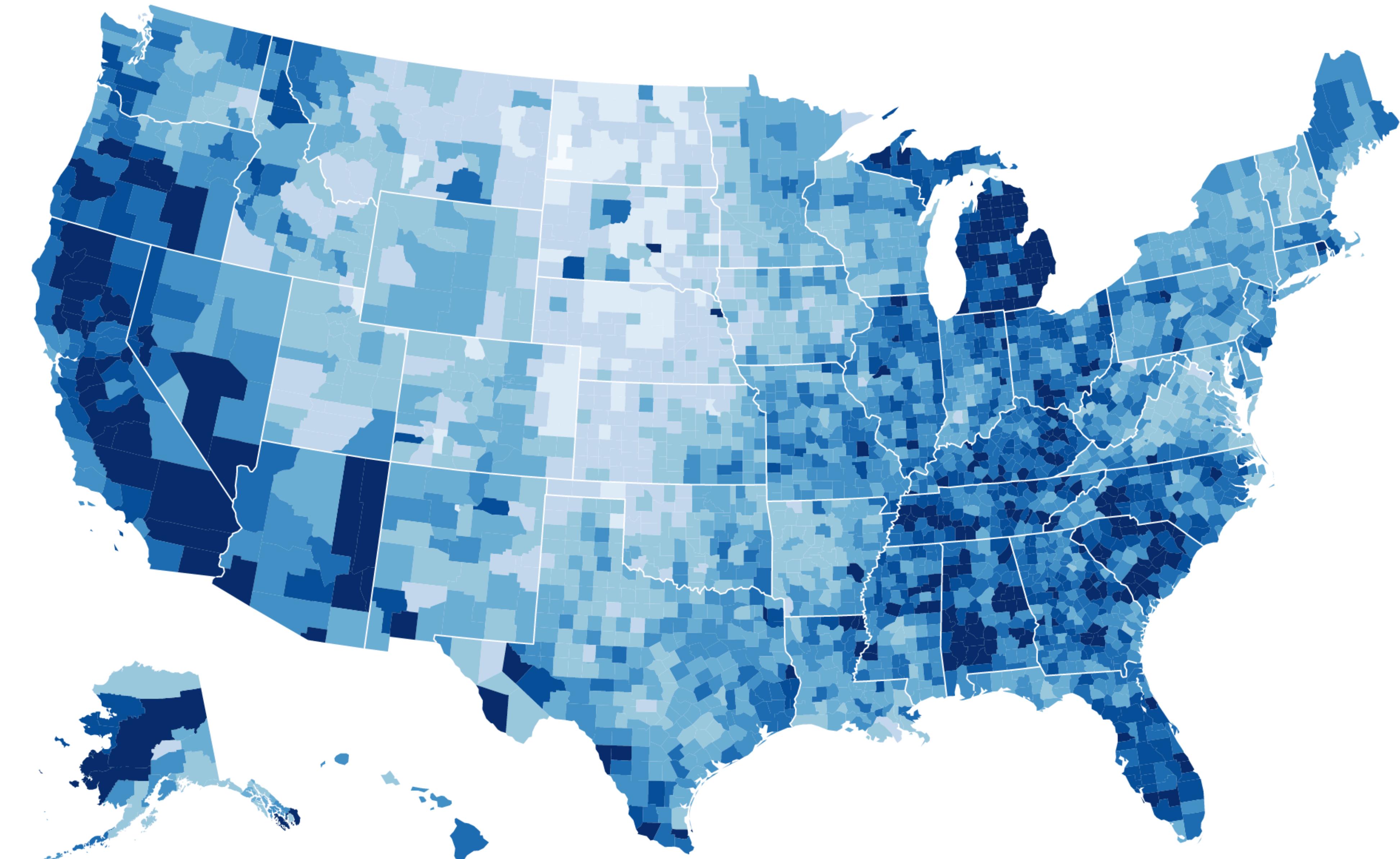
1. Concepts in **Interactive Visualization**
2. Introduction to **D3.js**
3. D3.js **Studio**

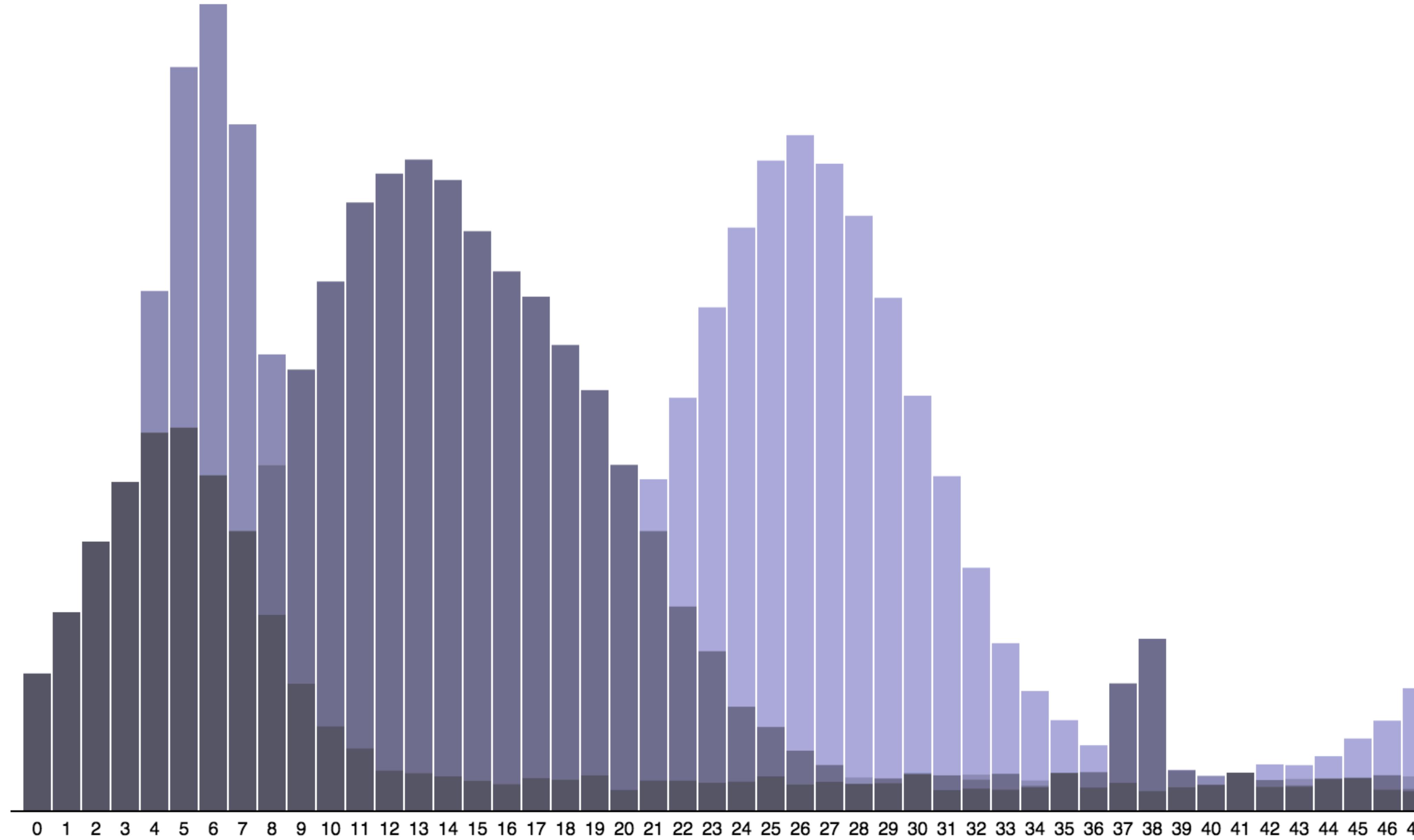
What is D3?

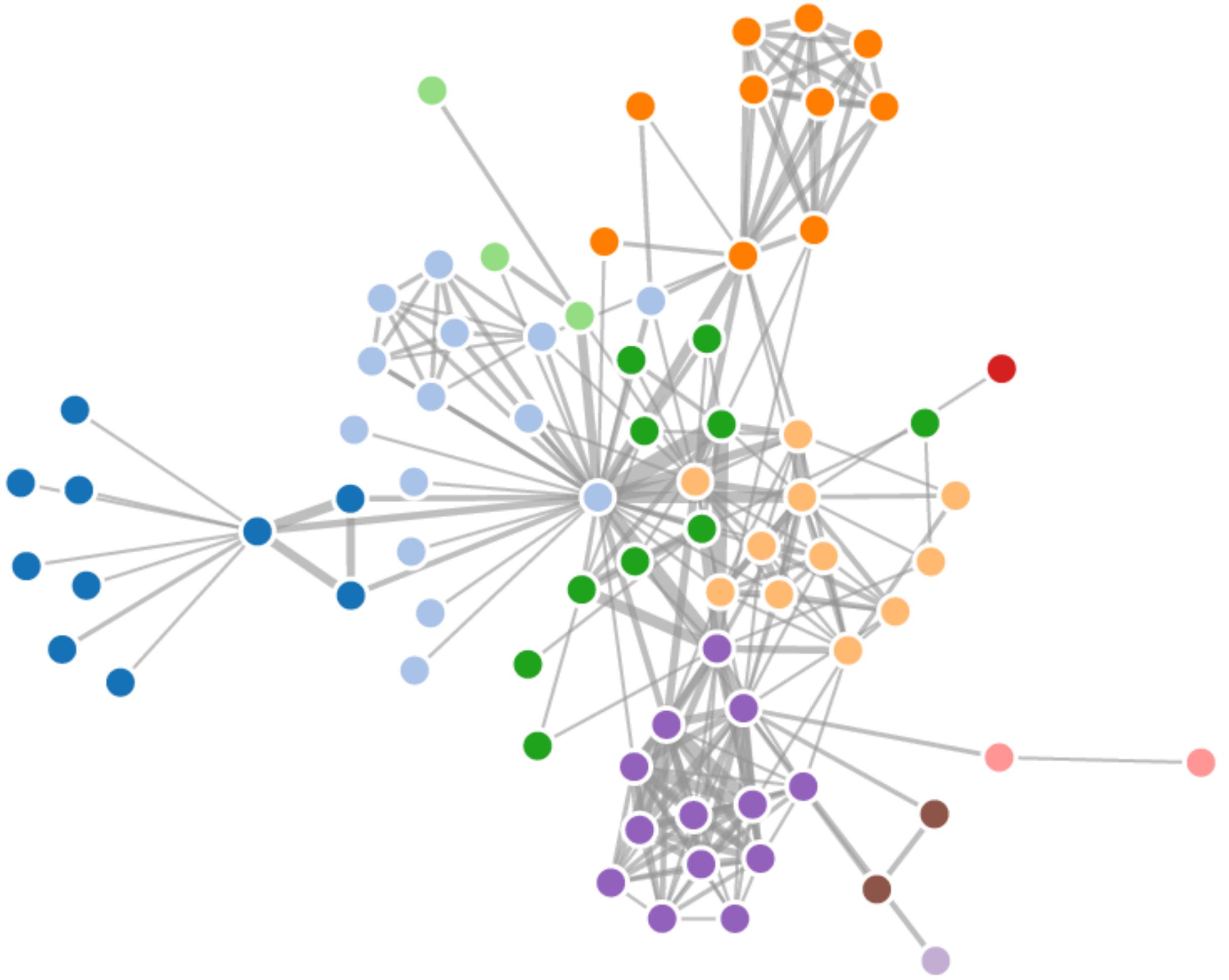
A **general-purpose visualization library** for
HTML and SVG.

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

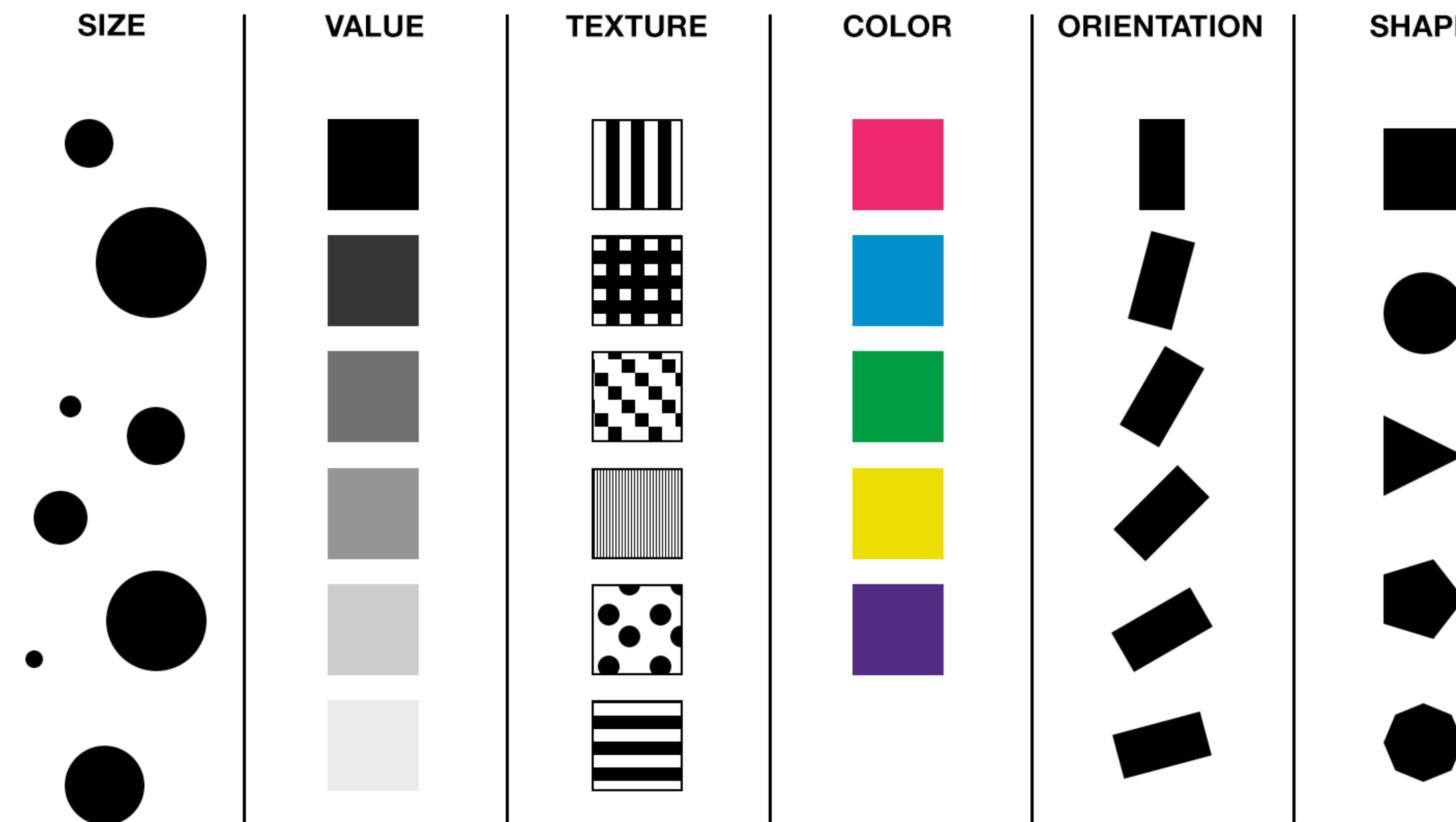
Visualizations are comprised of **basic graphical elements...**







...with **attributes**.



Jacques Bertin: six retinal properties

SVG is a **DOM for graphics**.

(Retina-ready, too.)

SVG primitives—

Rectangle

Circle

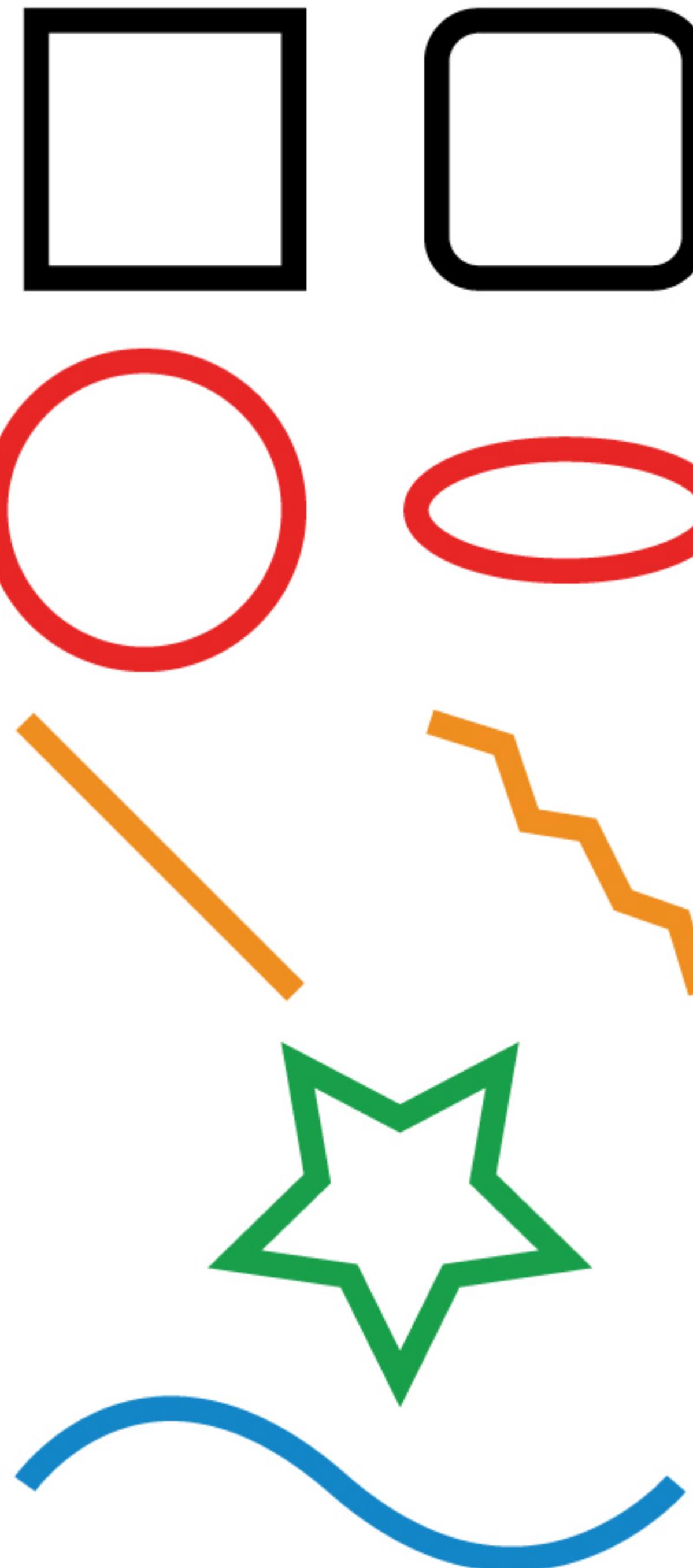
Ellipse

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**

JavaScript, 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

What makes D3.js **unique**?

- 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')
```

merge

Combine the enter and update groups.

.merge(elements)

(update)

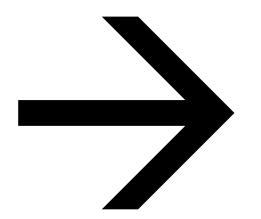
For each element in the selection,
update attributes.

```
.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" :
56     return (i+1)*250; //for each
57   })
58   .attr("cy",h/2) // "cy" :
59   .attr("r",function(d {debugger; return d; }));
60 circles.exit().remove();
61
```

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

vis.js:59

(anonymous function)

▼ Scope Variables

▼ Local

d: 20

▼ this: circle

 __data__: 20

► attributes: NamedNodeMap

 ... - ... - ... - ... - ... - ...

```
// 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;  
})
```

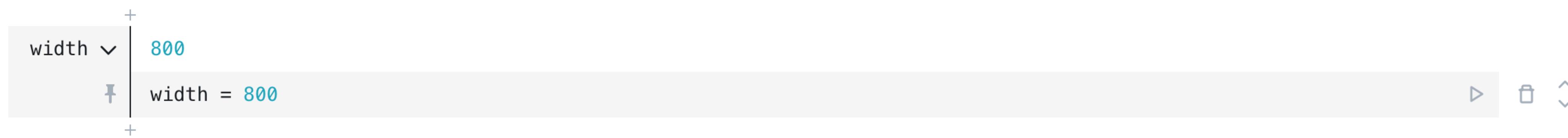
You can **chain** these, as well as use them to **pull an existing value**.

1. Concepts in **Interactive Visualization**
2. Introduction to **D3.js**
3. D3.js **Studio**

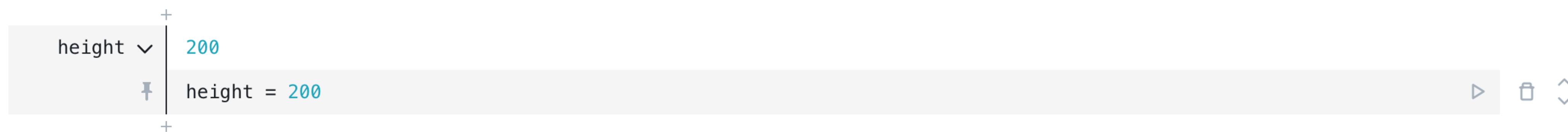
Let's draw some shapes!

[https://beta.observablehq.com/
@emilyfuhrman/d3-js-template](https://beta.observablehq.com/@emilyfuhrman/d3-js-template)

This row defines the **width** of your drawing space.



This row defines the **height** of the drawing space.



This row **includes the D3.js library**.

```
+  
v | undefined  
+ | d3 = require_("https://d3js.org/d3.v5.min.js")|  
+
```

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

(**cx**, **cy**, **r**)

//full code

```
var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
    .merge(my_circles)
    .attr("cx",500)
    .attr("cy",100)
    .attr("r",function(d){
        return d;
    });
my_circles.exit().remove();
```

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

```
var my_circles = svg.selectAll("circle")
```

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

```
var my_circles = svg.selectAll("circle")  
    .data([10,20,30]);
```

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

```
var my_circles = svg.selectAll("circle")
  .data([10,20,30]);
my_circles.enter()
```

//append an element

```
var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
```

//merge enter and update selections

```
var my_circles = svg.selectAll("circle")
  .data([10,20,30]);
my_circles.enter()
  .append("circle")
  .merge(my_circles)
```

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

```
var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
    .merge(my_circles)
    .attr("cx",500)
    .attr("cy",100)
```

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

```
var my_circles = svg.selectAll("circle")  
    .data([10,20,30]);  
  
my_circles.enter()  
    .append("circle")  
    .merge(my_circles)  
    .attr("cx",500)  
    .attr("cy",100)  
    .attr("r",function(d){  
        return d;  
   });
```

```
//and clean everything up
```

```
var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
    .merge(my_circles)
    .attr("cx",500)
    .attr("cy",100)
    .attr("r",function(d){
        return d;
    });
my_circles.exit().remove();
```

//full code

```
var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
    .merge(my_circles)
    .attr("cx",500)
    .attr("cy",100)
    .attr("r",function(d){
        return d;
    });
my_circles.exit().remove();
```

Only one?

```
//add a function to 'cx'

var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
    .merge(my_circles)
    .attr("cx",function(d,i){
        return i*300;
    })
    .attr("cy",100)
    .attr("r",function(d){
        return d;
    });
my_circles.exit().remove();
```

```
//tweak it
```

```
var my_circles = svg.selectAll("circle")
    .data([10,20,30]);
my_circles.enter()
    .append("circle")
    .merge(my_circles)
    .attr("cx",function(d,i){
        return i*300 + 25;
    })
    .attr("cy",100)
    .attr("r",function(d){
        return d;
    });
my_circles.exit().remove();
```

Challenges

Challenge—
Give the circles a red stroke.

Challenge—

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

Challenge—

Make the circles rectangles.

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

(x, y, width, height)

Challenge—

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

-