CS50 for MBAs

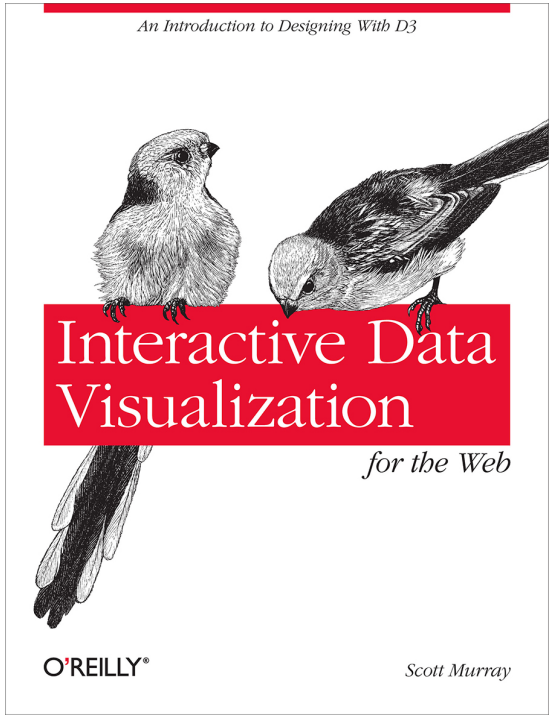
Maria Zlatkova

zlatkova@college.harvard.edu

Data Visualization with D3

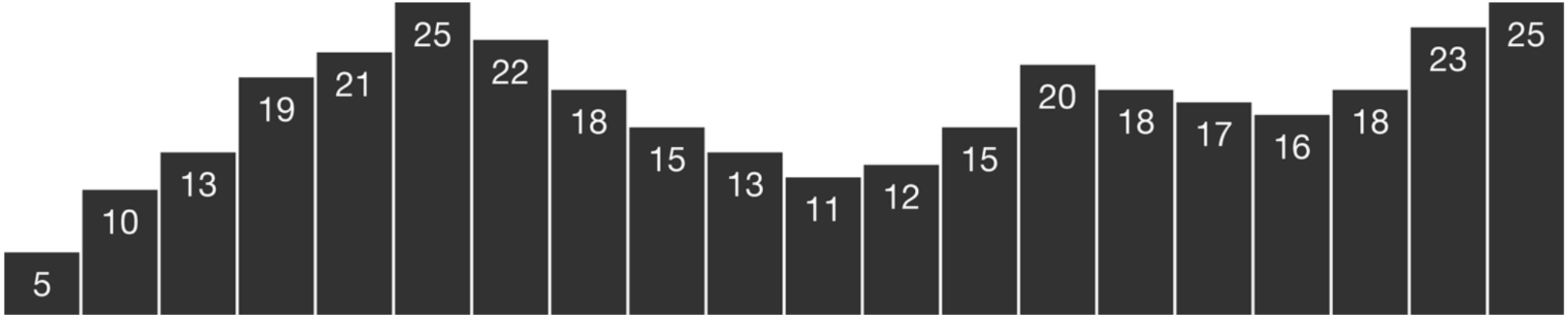
Data Visualization with D3

- Presentation slides use examples from Interactive Data Visualization for the Web by Scott Murray
- Additional examples use source code adapted from Harvard's CS 171



Why Data Visualization?

- Mapping information to visuals by creating rules that interpret data and express its values as visual properties



Why code?

- The alternative to computation is slow and tedious, yet we want to work with large datasets of thousands or millions of values; what would have taken years of effort by hand can be mapped in a moment
- We can easily experiment with *alternate mappings*, tweaking our rules and seeing their output re-rendered immediately. This loop of write/render/evaluate is critical to the iterative process of refining a design
- Sets of mapping rules function as *design systems*

Why interactive?

- Static visualizations can offer only precomposed “views” of data
- The number of dimensions of data are limited
- Dynamic, interactive visualizations can empower people to explore the data for themselves
- They make the data accessible to different audiences, from those who are merely browsing or exploring the dataset to those who approach the visualization with a specific question in search of an answer
- They encourage engagement with the data

D3

- D3 (D^3 or d3.js) —is a JavaScript library for creating data visualizations
- *Data-Driven Documents*
  - The *data* is provided, the *documents* are web-based, and D3 does the *driving*, in connecting the data to the documents
- *d3js.org*

What does D3 do?

- *Allows us to generate and manipulate web documents by:*
  - *Loading* data into the browser's memory
  - *Binding* data to elements within the document, creating new elements as needed
  - *Transforming* those elements by interpreting each element's bound datum and setting its visual properties accordingly
  - *Transitioning* elements between states in response to user input
- Learning D3 = learning the syntax tell it how you want it to load and bind data, and transform and transition elements

What does D3 do?

- The *transformation* step is most important; it's where the *mapping* happens
- D3 provides a structure for applying these transformations, but developers define the mapping rules

Alternatives

- D3 might not be perfect for every project – you might just need a quick chart and don’t have time to code it from scratch or you might need to support older browsers & can’t rely on newer technologies like SVG

Alternatives

Easy Charts

- DataWrapper
- Flot
- Image Charts API
- gRaphaël
- Highcharts JS
- JavaScript InfoVis Toolkit
- jqPlot
- jQuery Sparklines
- Peity
- Timeline.js
- YUI

Graph Visualizations

- *Arbor.js*
- *Sigma.js*

Geomapping

- *Kartograph*
- *Leaflet*
- *Modest Maps*
- *Polymaps*

Three-Dimensional

- *PhiloGL*
- Three.js

Some cool D3 Visualizations

- Where We Came From and Where We Went, State by State
- At the National Conventions, the Words They Used
- The America's Cup Finale: Oracle's Path to Victory
- Simpson's Paradox
- An Interactive Visualization of NYC Street Trees
- Four Ways to Slice Obama's 2013 Budget Proposal

JSON (JavaScript Object Notation)

- JSON is a specific syntax for storing and exchanging data
- Popular data-interchange format for APIs (application program interfaces). It is essentially a JS object, with the only difference being that the property names are surrounded by double quotation marks

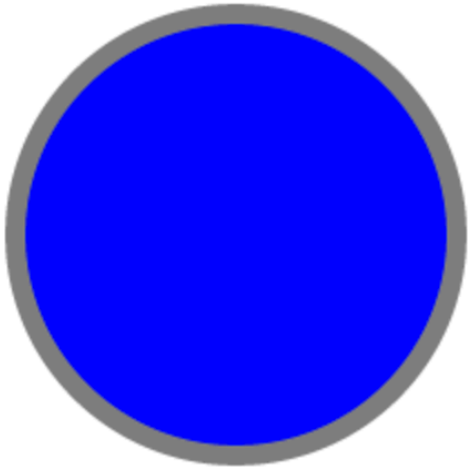
```
// JSON object
var course = {
      "id": "CS50",
      "name": "CS50 for MBAs",
      "students": 100,
      "active": true
}
```

```
var course = {
      id: "CS50",
      name: "CS50 for MBAs",
      students: 100,
      active: true
}
course.id;
course.students;

var course = {
      id: ”CS50",
      TFs: "Maria", "Will", "Christine”, "Daniel", "Christian", "Alex"]
};

var courses = [
              {id: "CS50",
                name: " Introduction to Computer Science " },
               { id: "CS51",
                name: "Introduction to Computer Science II" }
      ];
courses[1].id;
```

SVGs

- Scalable Vector Graphics (SVG) – more reliable, visually consistent, and faster
- Illustrator can generate SVG files, but generating them with code is more scalable

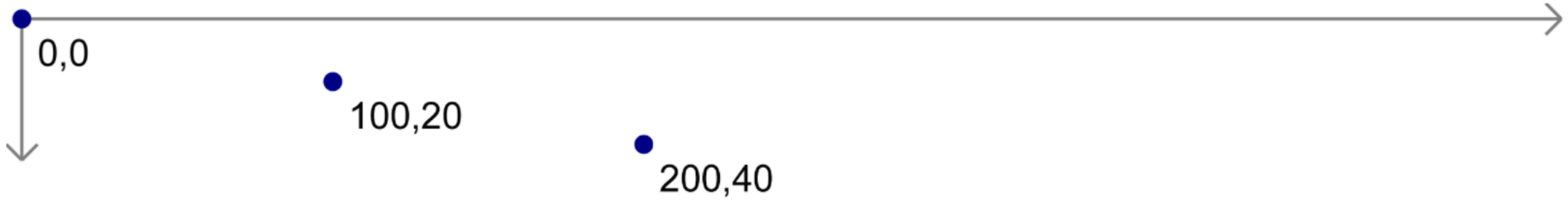
```
<svg width="50" height="50">
       <circle cx="25" cy="25" r="22" fill="blue" stroke="gray" stroke-width="2"/>
</svg>
```



SVGs

```
<svg width="500" height="50">
</svg>
rect, circle, ellipse, line, text, path
```

- *SVG coordinates system*

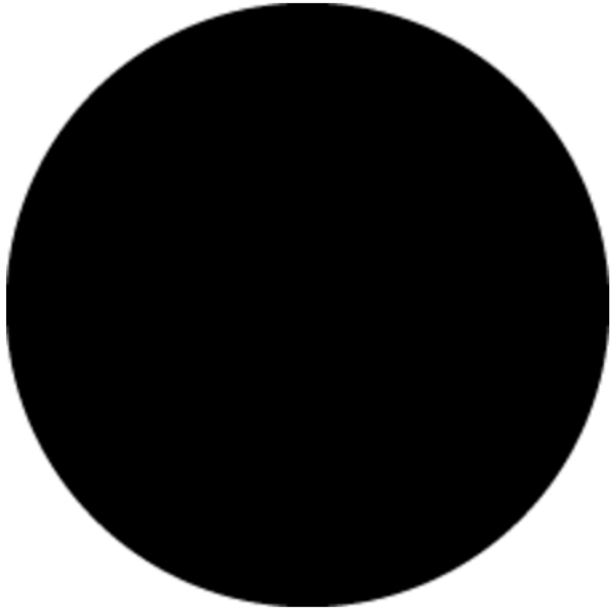


SVGs

```
<rect x="0" y="0" width="500" height="50"/>
```

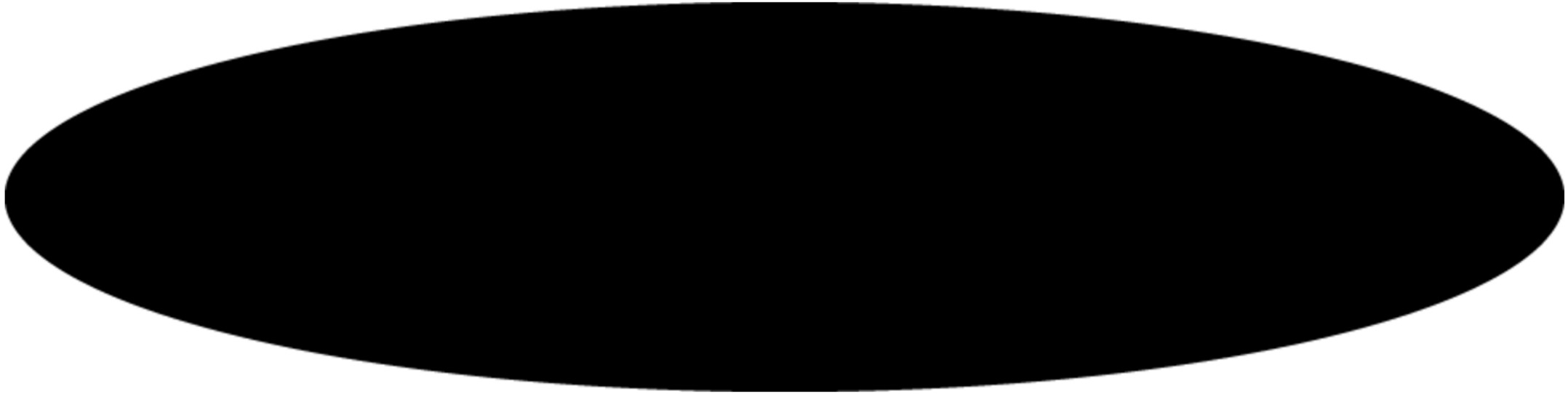
SVGs

```
<circle cx="250" cy="25" r="25"/>
```



SVGs

```
<ellipse cx="250" cy="25" rx="100" ry="25"/>
```



SVGs

```
<line x1="0" y1="0" x2="500" y2="50" stroke="black"/>
```

SVGs

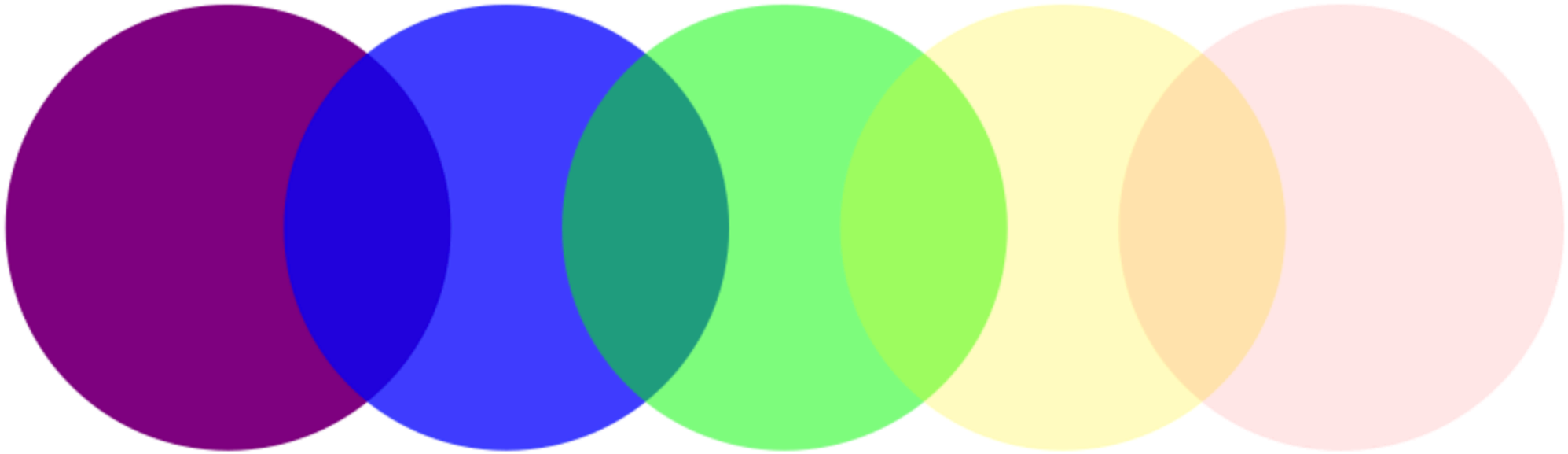
```
<circle cx="25" cy="25" r="22" fill="yellow"
stroke="orange" stroke-width="5"/>
```

SVGs

```
<rect x="0" y="0" width="30" height="30" fill="purple"/>
<rect x="20" y="5" width="30" height="30" fill="blue"/>
<rect x="40" y="10" width="30" height="30" fill="green"/>
<rect x="60" y="15" width="30" height="30" fill="yellow"/>
<rect x="80" y="20" width="30" height="30" fill="red"/>
```

SVGs

```
<circle cx="25" cy="25" r="20" fill="rgba(128, 0, 128, 1.0)"/>
<circle cx="50" cy="25" r="20" fill="rgba(0, 0, 255, 0.75)"/>
<circle cx="75" cy="25" r="20" fill="rgba(0, 255, 0, 0.5)"/>
<circle cx="100" cy="25" r="20" fill="rgba(255, 255, 0, 0.25)"/>
<circle cx="125" cy="25" r="20" fill="rgba(255, 0, 0, 0.1)"/>
```



Referencing D3

```
project-folder/
    d3/
        d3.v3.js
        d3.v3.min.js (optional)
    index.html
```

Referencing D3

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3 Page Template</title>
        <script type="text/javascript" src="d3/d3.v3.js"></script>
    </head>
    <body>
        <script type="text/javascript">
            // Your beautiful D3 code will go here
        </script>
    </body>
</html>
```

Setting up a web server

```
python -m SimpleHTTPServer 8888 &.
```

http://localhost:8888/

Chaining Methods

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

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

Chaining methods

- D3
  - References the D3 object, so we can access its methods. Our D3 adventure begins here.
- .select("body")
  - CSS selector as input, returns a reference to the first element in the DOM that matches
  - selectAll()
- .append("p")
  - creates whatever new DOM element you specify and appends it to the end (but *just inside*) of whatever selection it's acting on
  - hands off a reference to the new element it just created.
- .text("New paragraph!")
  - takes a string and inserts it between the opening and closing tags of the current selection
- ;
  - Chain over

Going chainless

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

Binding data

- We *bind* our data input values to elements in the DOM
- Like “attaching” or associating data to specific elements, so that later you can reference those values to apply mapping rules
- `selection.data()` method to bind data to DOM elements
  - First, we need the data and a selection of DOM elements

Binding data

```
// could also use JSON or CSV data
var dataset = [ 5, 10, 15, 20, 25 ];

d3.select("body")
     .selectAll("p")
     .data(dataset)
     .enter()
     .append("p")
     .text("New paragraph!");
```

New paragraph!

New paragraph!

New paragraph!

New paragraph!

New paragraph!

Binding data

- d3.select("body")
  - Finds the body in the DOM and hands off a reference to the next step in the chain.
- .selectAll("p")
  - Selects all paragraphs in the DOM. Because none exist yet, this returns an empty selection
- .data(dataset)
  - Counts and parses our data values
  - There are five values in our array called dataset, so everything past this point is executed five times
- .enter()
  - Creates new, data-bound elements; looks at the current DOM selection, then at the data being handed to it
  - If more data values than corresponding DOM elements, *creates a new placeholder element*, then hands off a reference to this new placeholder to the next step in the chain
- .append("p")
  - Takes the empty placeholder selection created by enter() and appends a p element into the DOM.
  - hands off a reference to the element it just created to the next step in the chain.
- .text("New paragraph!") - Takes the reference to the newly created p and inserts a text value.

Binding data

- When binding data, order matters!
- Methods often return references to selections

Binding data

```
d3.select("body")
    .selectAll("p")
    .data(dataset)
    .enter()
    .append("p")
    .text("New paragraph!");
```

New paragraph!

New paragraph!

New paragraph!

New paragraph!

New paragraph!

Binding data

- Typically use anonymous functions
- Data likes being held – we need to know where it comes from
- We can’t just do:

```
.text(d)

// instead need to do:
.text(function(d) {
    return d;
});
```

Binding data

```
d3.select("body")
     .selectAll("p")
     .data(dataset)
     .enter()
     .append("p")
     .text("New paragraph!");

     // change last line to
     .text(function(d) {
          return d;
     });
```

5

10

15

20

25

Binding data

```
d3.select("body")
     .selectAll("p")
     .data(dataset)
     .enter()
     .append("p")
     .text("New paragraph!");

     // change last line to
     .text(function(d) {
     return "I can count up to " + d;
});
```

I can count up to 5

I can count up to 10

I can count up to 15

I can count up to 20

I can count up to 25

Binding data

```
.style("color", "red");
```

I can count up to 5

I can count up to 10

I can count up to 15

I can count up to 20

I can count up to 25

```
.style("color", function(d) {
     if (d > 15) {
          //Threshold of 15
          return "red";
     } else {
          return "black";
     }
});
```

I can count up to 5

I can count up to 10

I can count up to 15

I can count up to 20

I can count up to 25

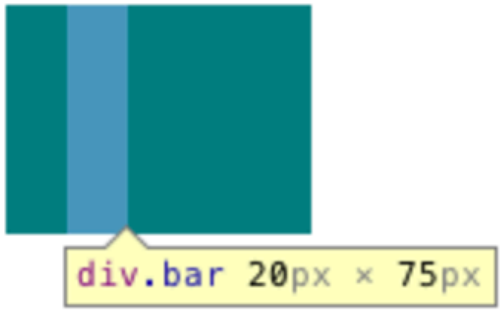
Drawing with Data

```
var dataset = [ 5, 10, 15, 20, 25 ];
d3.select("body")
     .selectAll("div")
     .data(dataset)
     .enter()
     .append("div")
     .attr("class", "bar");
```

Drawing with Data

```
var dataset = [ 5, 10, 15, 20, 25 ];
d3.select("body")
     .selectAll("div")
     .data(dataset)
     .enter()
     .append("div")
     .attr("class", "bar");
```

Drawing with Data

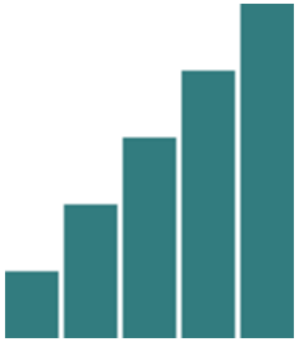


drawing_divs_1.html 〉 DOM Tree 〉 html 〉 body 〉 div.bar

```
<!DOCTYPE html>
▼<html lang="en">
  ▶<head>…</head>
  ▼<body>
    ▶<script type="text/javascript">…</script>
     <div class="bar"></div>
     <div class="bar"></div>
     <div class="bar"></div>
     <div class="bar"></div>
     <div class="bar"></div>
   </body>
 </html>
```

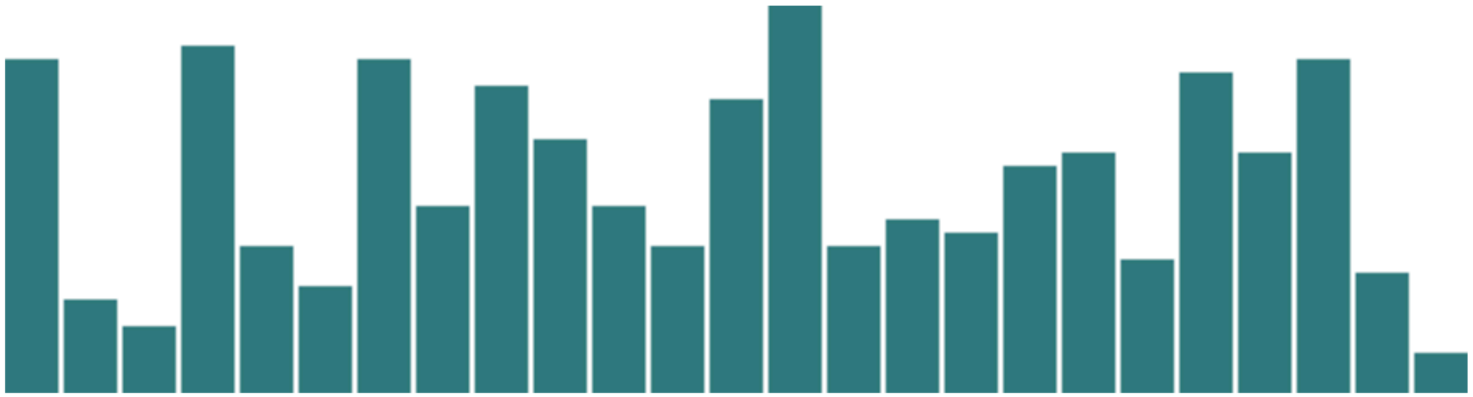
Drawing with Data

```
.style("height", function(d) {
     return d + "px";
});

.style("height", function(d) {
     var barHeight = d * 5;
     //Scale up by factor of 5
     return barHeight + "px";
});

margin-right: 2px;
```



data() Flexibility

```
var dataset = [ 25, 7, 5, 26, 11, 8, 25, 14, 23, 19,
14, 11, 22, 29, 11, 13, 12, 17, 18, 10, 24, 18, 25,
9, 3 ];
```



Data-driven shapes with SVGs

```
var svg = d3.select("body").append("svg");

// width and height
var w = 500; var h = 50;

var svg = d3.select("body")
             .append("svg")
             .attr("width", w)
             .attr("height", h);
var circles = svg.selectAll("circle")
                  .data(dataset)
                  .enter()
                  .append("circle");
```

Data-driven shapes with SVGs

```
circles.attr("cx", function(d, i) {
        return (i * 50) + 25;
    })
    .attr("cy", h/2)
    .attr("r", function(d) {
        return d;
    });
```

Data-driven shapes with SVGs

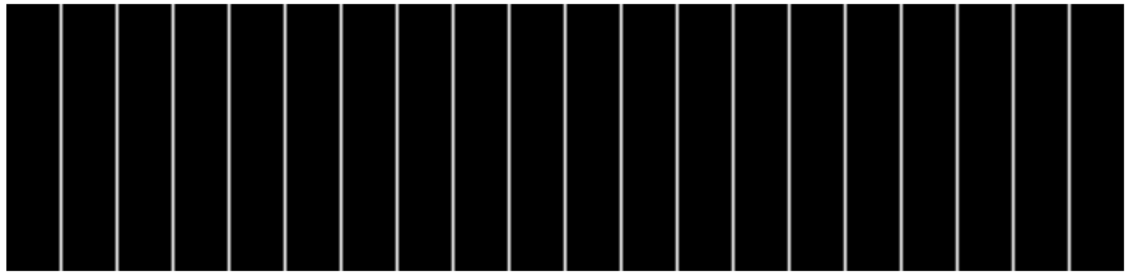
```
.attr("fill", "yellow")
.attr("stroke", "orange")
.attr("stroke-width", function(d) {
    return d/2;
});
```

Data-driven shapes with SVGs

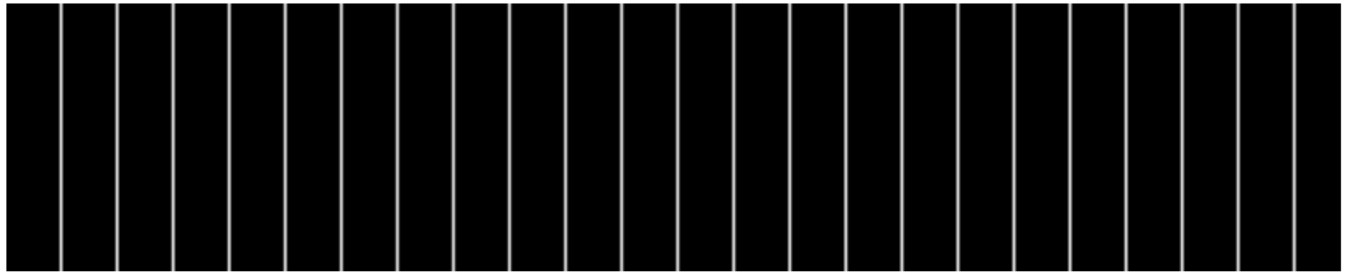
```
var dataset = [ 5, 10, 13, 19, 21, 25, 22, 18, 15, 13, 11, 12, 15, 20, 18, 17, 16,
18, 23, 25 ];

//Create SVG element
var svg = d3.select("body") .append("svg") .attr("width", w) .attr("height", h);

svg.selectAll("rect")
       .data(dataset)
       .enter()
       .append("rect")
       .attr("x", 0)
       .attr("y", 0)
       .attr("width", 20)
       .attr("height", 100);
```

Data-driven shapes with SVGs

```
.attr("x", function(d, i) {
    return i * 21; //Bar width of 20 + 1 for padding
})
```



Fixed width may cut off some values

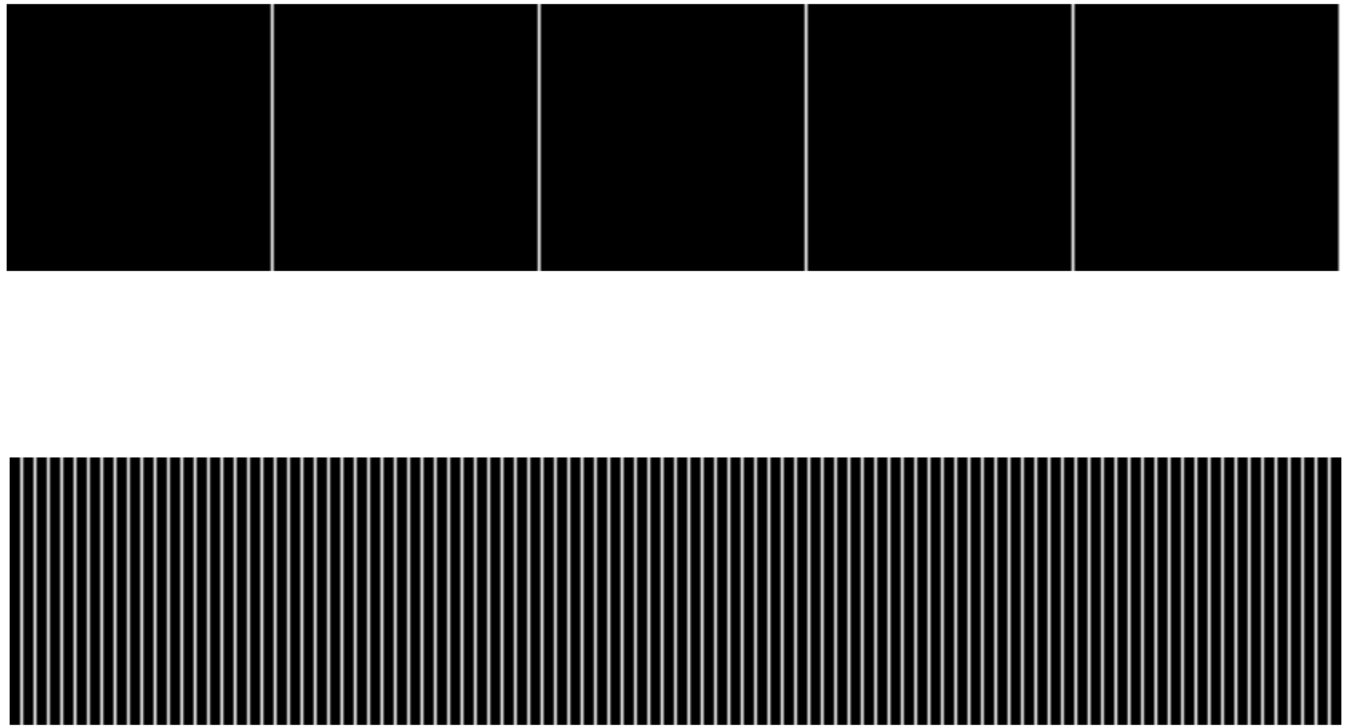


Data-driven shapes with SVGs

```
.attr("x", function(d, i) {
    return i * (w / dataset.length);
})
```

```
.attr("width", w / dataset.length - barPadding)
```

Data-driven shapes with SVGs



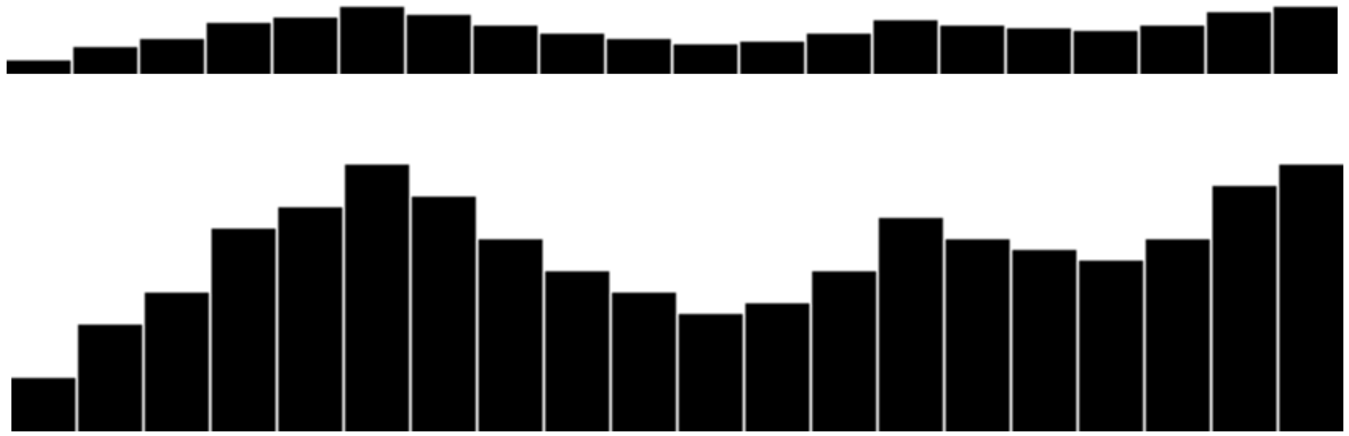
Data-driven shapes with SVGs

```
.attr("height", function(d) {
    return d;
});
```

```
.attr("height", function(d) {
    return d * 4;
});
```

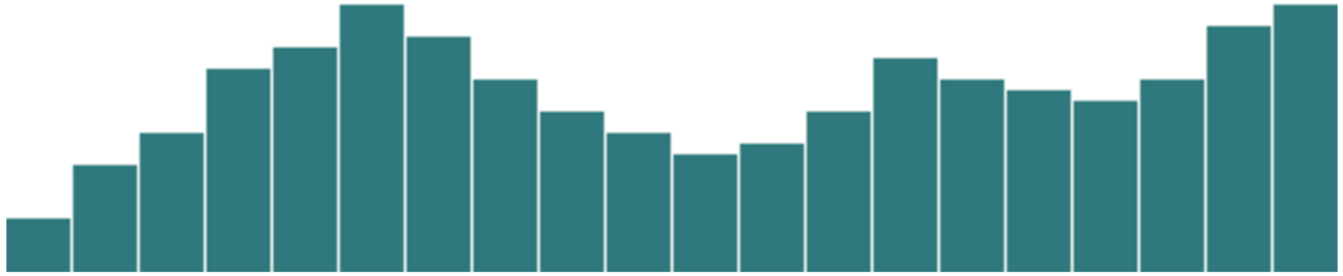
Data-driven shapes with SVGs

```
.attr("y", function(d) {
    return h - d; //Height minus data value
})

.attr("height", function(d) {
    return d; //Just the data value
});
```



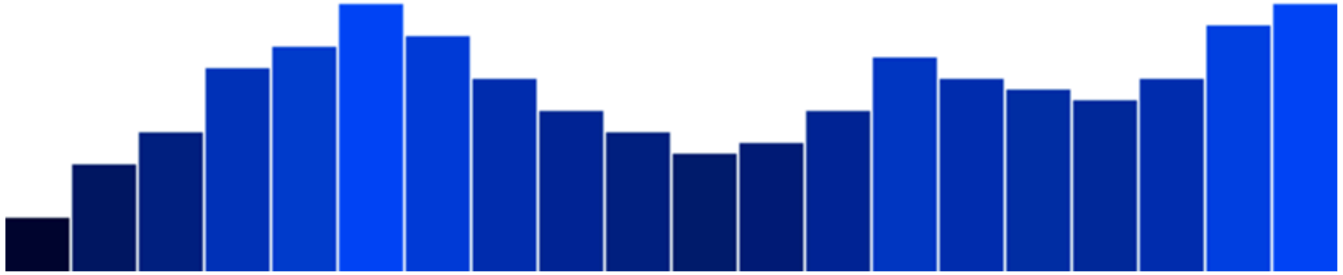
More attributes: Color

```
.attr("fill", "teal");
```

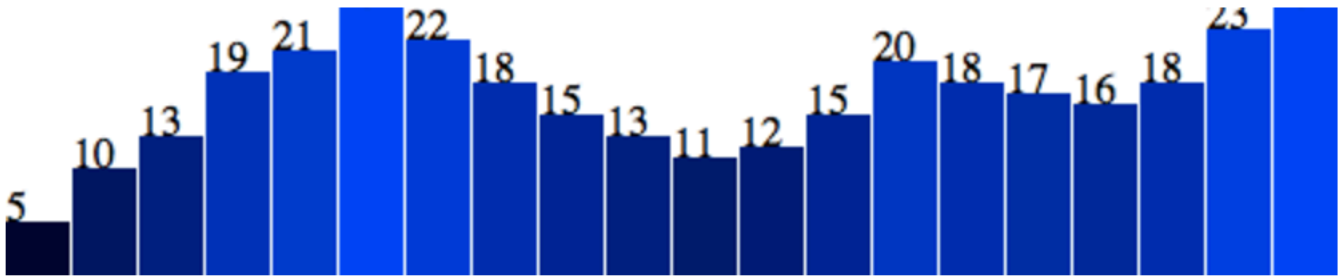


Data-driven shapes with SVGs

```
.attr("fill", function(d) {
    return "rgb(0, 0, " + (d * 10) + ")";
});
```



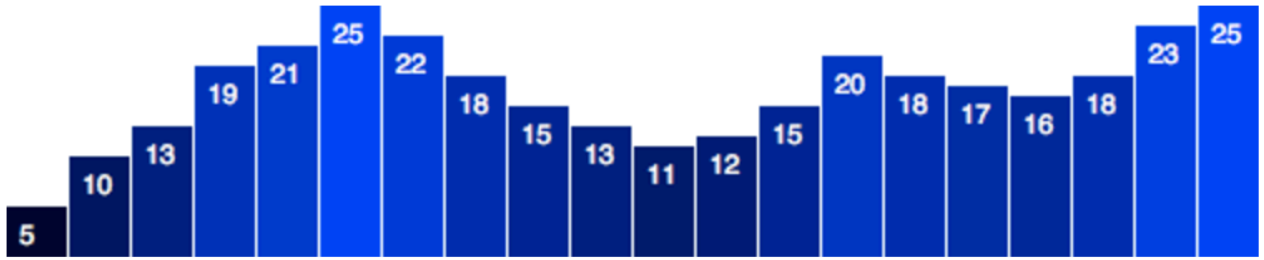
Data-driven shapes with SVGs




```
.text(function(d) {
    return d;
})
.attr("x", function(d, i) {
    return i * (w / dataset.length);
})
.attr("y", function(d) {
    return h - (d * 4);
});
```

Data-driven shapes with SVGs

```
.attr("x", function(d, i) {
    return i * (w / dataset.length) + 5; // +5
})
.attr("y", function(d) {
    return h - (d * 4) + 15; // +15
});
.attr("font-family", "sans-serif")
.attr("font-size", "11px")
.attr("fill", "white");
```



Making a Scatterplot

```
var dataset = [
    [5, 20], [480, 90], [250, 50], [100, 33], [330, 95], [410, 12], [475, 44], [25, 67], [85, 21], [220, 88]
];
```

Making a Scatterplot

```
//Create SVG element
var svg = d3.select("body")
      .append("svg")
      .attr("width", w)
      .attr("height", h);
```

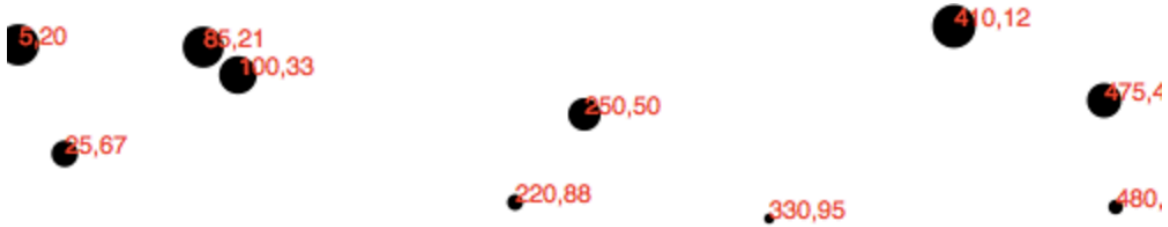
```
svg.selectAll("circle")
      .data(dataset)
      .enter()
      .append("circle”)
      .attr("cx", function(d) {
            return d[0];
      })
      .attr("cy", function(d) {
            return d[1];
      })
      .attr("r", 5);
```

Making a Scatterplot

```
.attr("r", function(d) {
    return Math.sqrt(h - d[1]);
});
```

Making a Scatterplot

```
svg.selectAll("text") // <-- Note "text", not "circle" or "rect"
      .data(dataset)
      .enter()
      .append("text") // <-- Same here!
      .text(function(d) {
            return d[0] + "," + d[1];
      })
      .attr("x", function(d) { return d[0]; })
      .attr("y", function(d) { return d[1]; })
       .attr("font-family", "sans-serif")
      .attr("font-size", "11px")
      .attr("fill", "red");
```



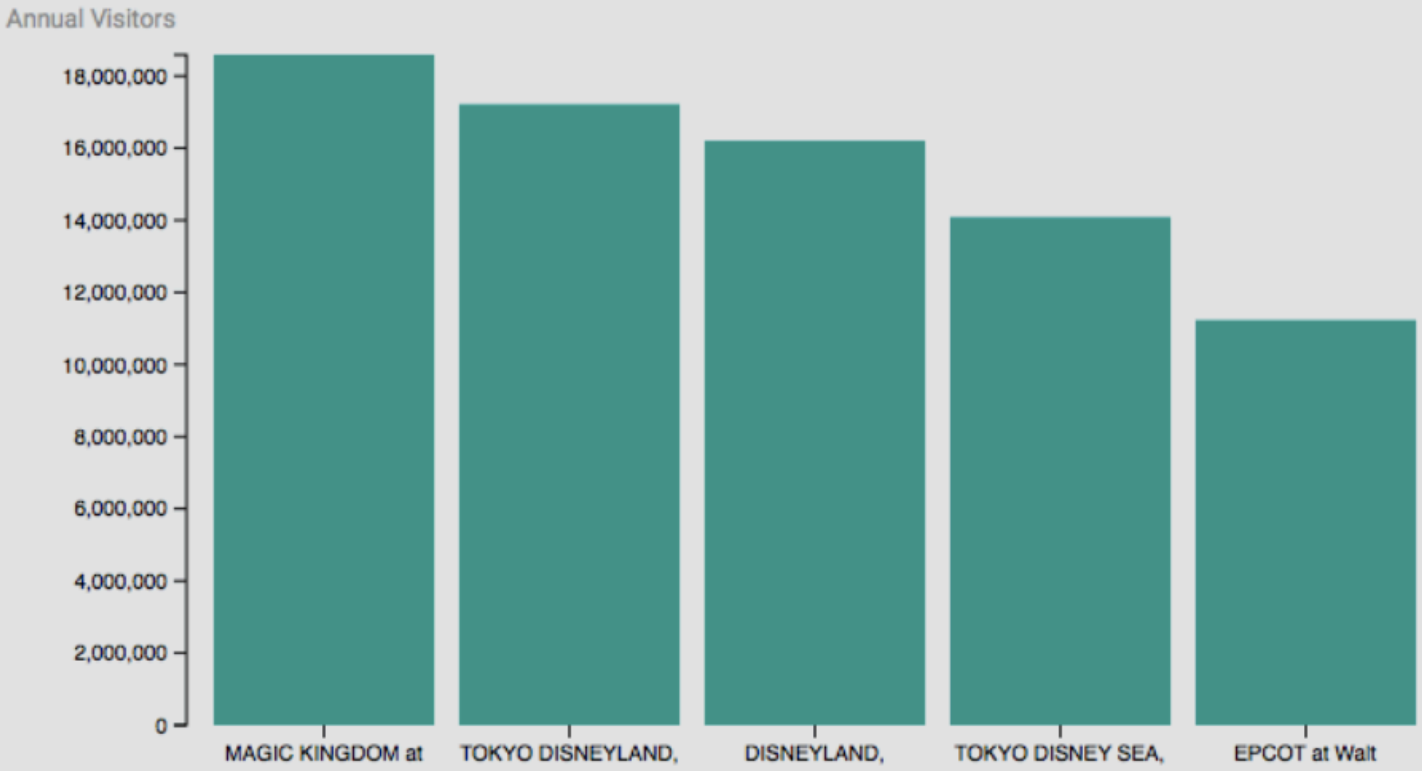
Let’s visualize some data

- 3 **attractions** in an amusement park that we want to store. Each amusement ride has several attributes
  - ID
  - Name
  - Price in USD
  - List with opening days (some attractions are open only on specific days, like weekends)
  - Limited access for children (yes / no)

Theme Parks, Water Parks and Museums

Global Attractions Attendance

Choose Category: All Attractions



Source: TEA/AECOM 2013 Theme Index and Museum Index

Thank you! ☺

- To learn more about D3:
  - [d3js.org](d3js.org)