Introduction to D3.js

By Rui Li
01/12/2023

Slide Material Source Credits

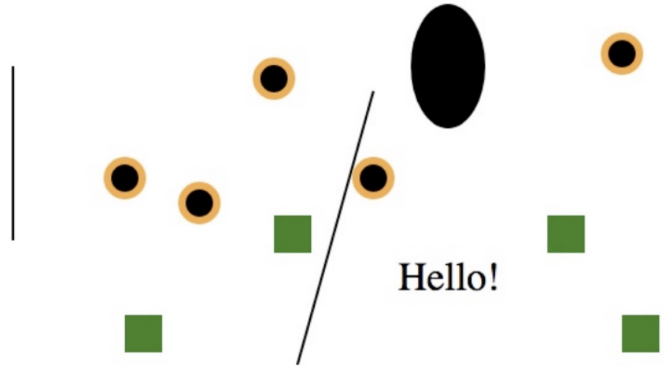
- https://d3js.org/
- https://www.d3indepth.com/
- https://d3-graph-gallery.com/
- https://observablehq.com/@d3/gallery
- Prof. Han-Wei Shen, Jiayi Xu, and Wenbin He

Outline

- D3 basics
  - scale
  - colormap
  - axis
- D3 shapes & layouts
  - SVG shapes
  - line

Review

- HTML
- CSS
- SVG
- JavaScript
  - Manipulate the HTML DOM



What is D3?

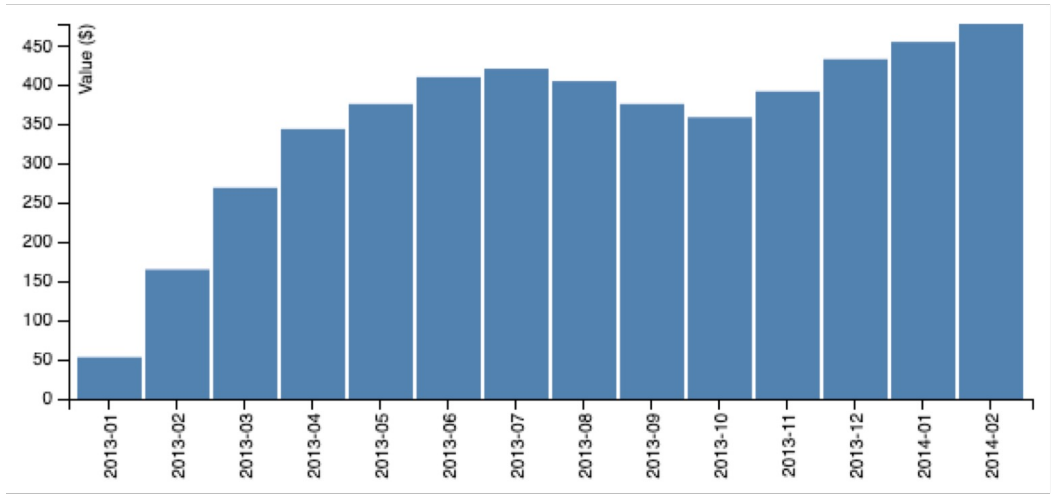
- It is an open-source JavaScript library developed by Mike Bostock to create custom interactive data visualizations in the web browser using SVG, HTML and CSS.
- Official website: d3js.org

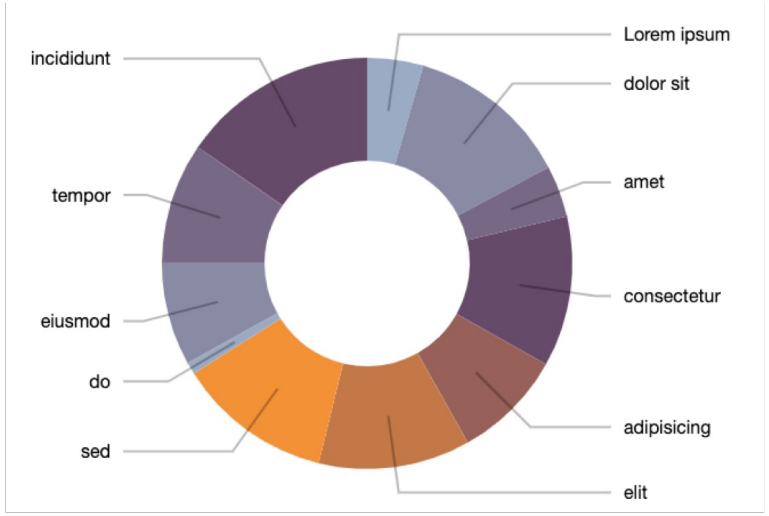
What is D3?

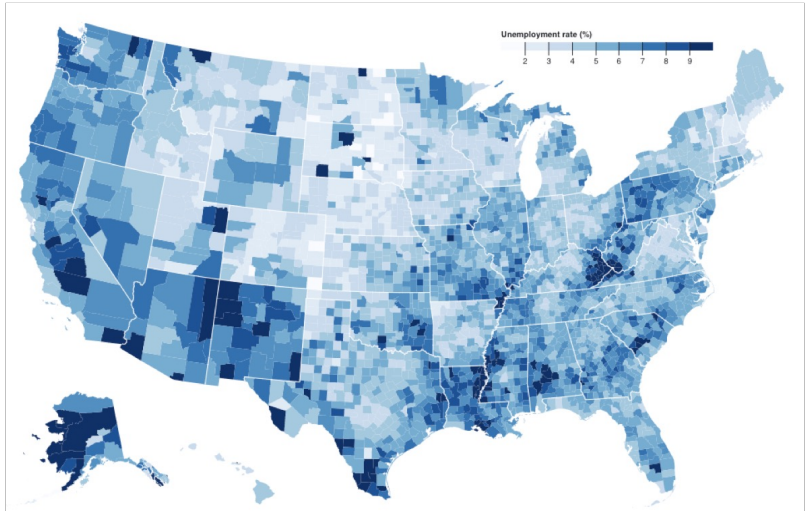
Features

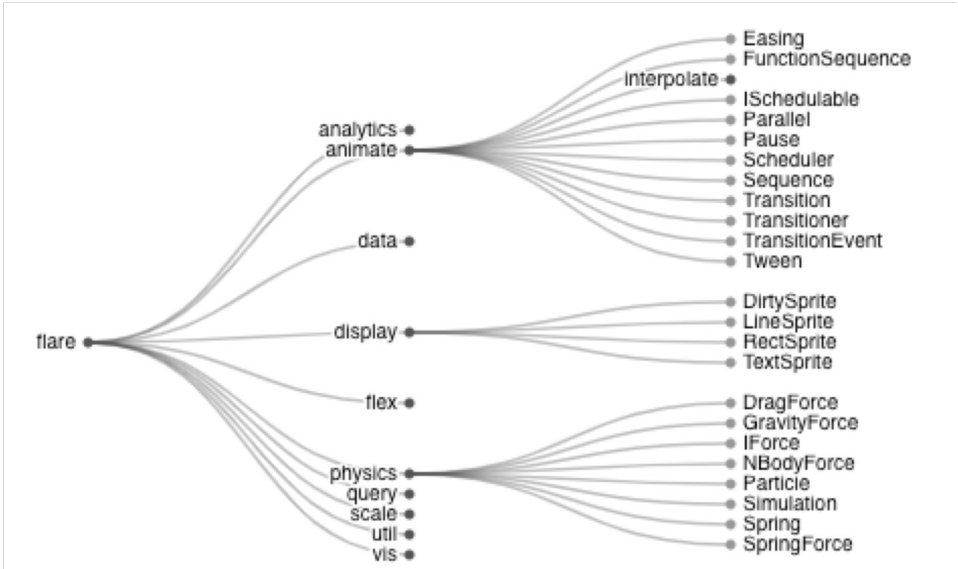
- **Data-driven**: read data from a number of formats (csv, json, xml...).
- **DOM Manipulation:** D3 allows you to manipulate the Document Object Model (DOM) based on your data.
- **Data Driven Elements:** It empowers your data to dynamically generate elements and apply styles to the elements, be it a table, a graph or any other HTML element and/or group of elements.
- **Interaction and Animation**

Examples of data visualizations generated by D3









Why/When choose D3?

- Web environment
- Focuses on data visualization (mostly 2D visualization)
- Support for animation and interaction
- Community support 

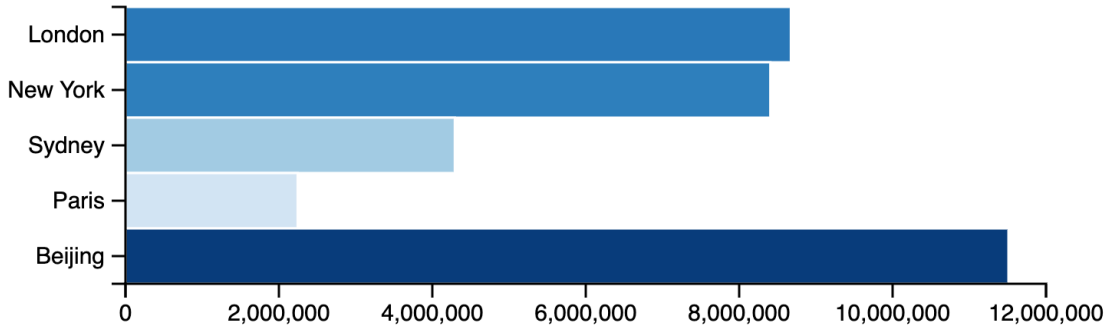
Note: for spatial data / scientific visualization / large scale data:

d3.js might not be a good choice.

Goal

Create a bar chart to show the populations of cities

	A	B
1	name	population
2	London	8674000
3	New York	8406000
4	Sydney	4293000
5	Paris	2244000
6	Beijing	11510000

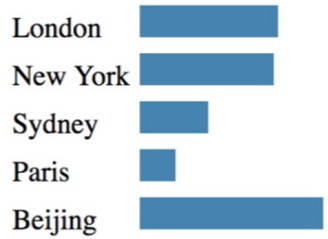


Workflow

Create a bar chart to show the populations of cities

- environment setup
- load the CSV file
- bind the data to visual elements (DOMs)
- …

	A	B
1	name	population
2	London	8674000
3	New York	8406000
4	Sydney	4293000
5	Paris	2244000
6	Beijing	11510000



Environment Setup

- D3.js library
- Web server
- Editor
- Web browser

Environment Setup

- D3.js library
  - latest version: v7.8.1 (used for this lecture)
  - https://d3js.org/
- Web server
- Editor
- Web browser

Environment Setup

- D3.js library
- Web server
  - for loading external data
  - live server plugin provided by VScode
  - simple http, node.js, etc. also works
  - observable
- Editor
- Web browser

Environment Setup

- D3.js library
- Web server
- Editor: VS code
- Web browser: chrome

Demo

D3 - Data Loading

https://github.com/d3/d3-dsv/tree/v3.0.1

- d3.csv
- d3.json
- …

- **note**: the data loading function in d3.js changes in V4 and V5
- **suggestion**: use await to read the data

Demo

D3 - Map data to visual elements

D3 can map data to HTML/SVG elements.

- We can construct the **DOMs** from **data**.

PatientID	Gender	Age	Age_Group
3822546	MALE	46.0	45 to 64
3822046	MALE	0.0	0 to 4
3822163	FEMALE	29.0	25 to 34
3822263	FEMALE	22.0	18 to 24
3822241	MALE	36.0	35 to 44
3822163	FEMALE	35.0	35 to 44
3822453	MALE	12.0	5 to 14
3822077	MALE	47.0	45 to 64
3822767	MALE	23.0	18 to 24
3822763	MALE	29.0	25 to 34
3822453	MALE	17.0	15 to 17

data table

visual channels and marks

(html / svg elements)

D3 - Selection (Manipulation of DOMs)

- D3 selection allow us to select DOM elements and manipulate them. (changing style, modifying their attributes, etc.)
- Pattern

```
d3.select(element).function_name(key, value)
```

Name	Behaviour	Example
.style	Update the style	d3.selectAll('circle').style('fill', 'red')
.attr	Update an attribute	d3.selectAll('rect').attr('width', 10)
.classed	Add/remove a class attribute	d3.select('.item').classed('selected', true)
.property	Update an element's property	d3.selectAll('.checkbox').property('checked', false)
.text	Update the text content	d3.select('div.title').text('My new book')
.html	Change the html content	d3.select('.legend').html(' 0 - 10 ')

https://www.d3indepth.com/selections/

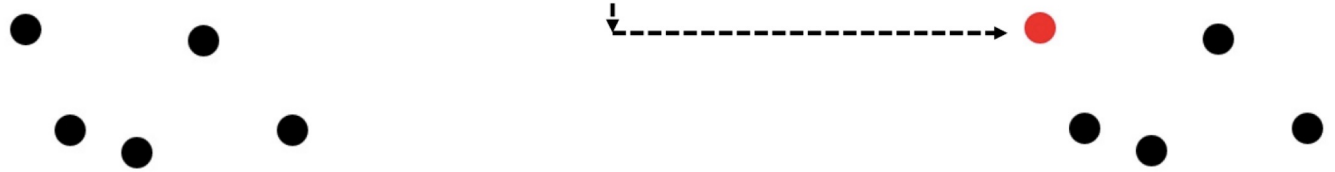
D3 - Selection (Manipulation of DOMs)

D3 has two functions to make selections d3.select and d3.selectAll.

d3.select() selects the first matching element.

```
<svg>
    <g id="group_1">
        <circle cx="30" cy="30" r="7" />
        <circle cx="50" cy="75" r="7" />
        <circle cx="80" cy="85" r="7" />
    </g>

    <g id="group_2">
        <circle cx="150" cy="75" r="7" />
        <circle cx="110" cy="35" r="7" />
    </g>
</svg>
```

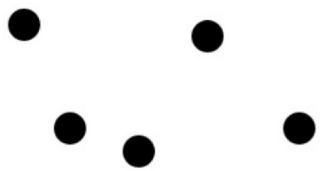
```
d3.select("circle").style("fill", "red");
```



D3 - Selection (Manipulation of DOMs)

d3.selectAll() selects all matching elements. Each function takes a single argument which specifies the selector string.

```
<svg>
    <g id="group_1">
        <circle cx="30" cy="30" r="7" />
        <circle cx="50" cy="75" r="7" />
        <circle cx="80" cy="85" r="7" />
    </g>

    <g id="group_2">
        <circle cx="150" cy="75" r="7" />
        <circle cx="110" cy="35" r="7" />
    </g>
</svg>
```



```
d3.selectAll("circle").style("fill", "red");
```

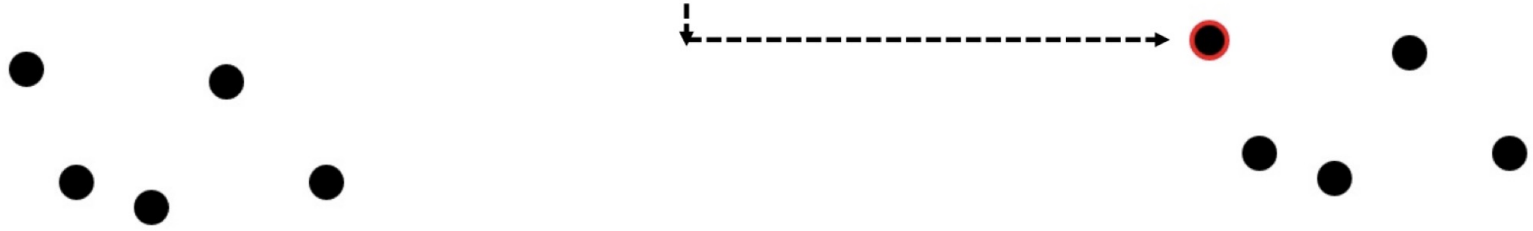
D3 - Selection (Manipulation of DOMs)

selection.**style**(StyleName, value):

- Set the CSS style property to the specified value on the selected elements.

```
<svg>
    <g id="group_1">
        <circle cx="30" cy="30" r="7" />
        <circle cx="50" cy="75" r="7" />
        <circle cx="80" cy="85" r="7" />
    </g>

    <g id="group_2">
        <circle cx="150" cy="75" r="7" />
        <circle cx="110" cy="35" r="7" />
    </g>
</svg>
```

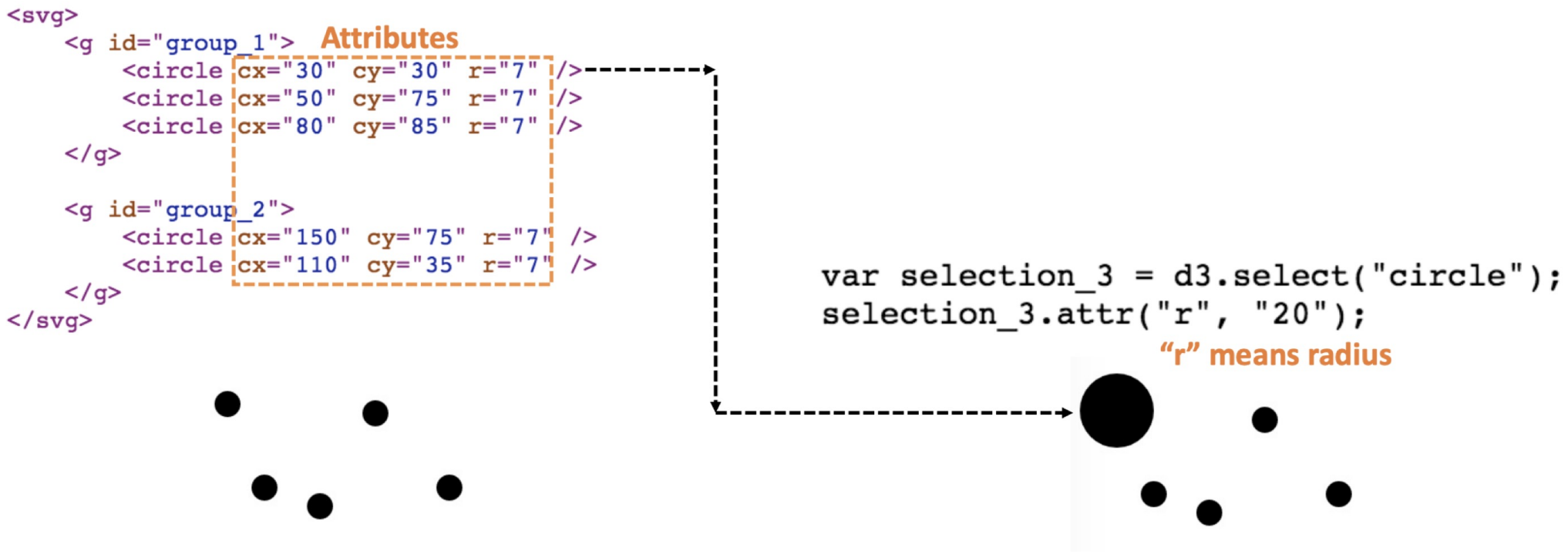
```
var selection_3 = d3.select("circle");
selection_3.style("stroke", "red");
selection_3.style("stroke-width", "2px");
```



D3 - Selection (Manipulation of DOMs)

selection.**attr**(AttrName, value):

- Set the attribute to the specified value on the selected elements.

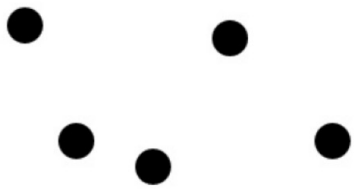


D3 - Selection (Manipulation of DOMs)

selection.**remove**():

- Remove the selected elements from the document.

```
<svg>
    <g id="group_1">
        <circle cx="30" cy="30" r="7" />
        <circle cx="50" cy="75" r="7" />
        <circle cx="80" cy="85" r="7" />
    </g>

    <g id="group_2">
        <circle cx="150" cy="75" r="7" />
        <circle cx="110" cy="35" r="7" />
    </g>
</svg>
```



```
var selection_5 = d3.select("#group_2");
selection_5.remove();
```

D3 - Selection (Manipulation of DOMs)

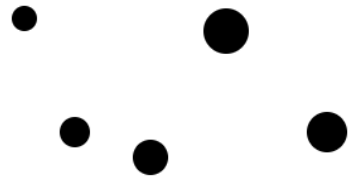
Selection with **functions**

- we can pass a function to .style, .attr, .text, .html, etc.
- pattern

```
d3.selectAll(element)
  .style('fill',function(d, i){
     return value;
  });
```

```
<svg>
 <g id="group_1">
   <circle cx="30" cy="30" r="5"></circle>
   <circle cx="50" cy="75" r="6"></circle>
   <circle cx="80" cy="85" r="7"></circle>
 </g>
 <g id="group_2">
   <circle cx="150" cy="75" r="8"></circle>
   <circle cx="110" cy="35" r="9"></circle>
 </g>
</svg>
```

```
d3.selectAll("circle").attr("r", function(d,i){
    return i+5;
});
```



D3 - Data binding

- Data-driven programming
- Pattern

```
d3.select(container)
  .selectAll(element-type)
  .data(array)
  .join(element-type);
```

- data(): specify the array of data to be joined
- join(): creates a correspondence between an array of data and a selection of HTML or SVG elements.

D3 – Data binding

- bind data with HTML elements

```
let myData = [10, 20, 30, 40, 50];

d3.select('body')
    .selectAll('p')
    .data(myData)
    .join('p')
    .text(function(d,i){
        return 'index:' + i + ' value:' + d;
    })
```

```
<p>index:0 value:10</p>
<p>index:1 value:20</p>
<p>index:2 value:30</p>
<p>index:3 value:40</p>
<p>index:4 value:50</p>
```

index:0 value:10

index:1 value:20

index:2 value:30

index:3 value:40

index:4 value:50

D3 - Data binding

- bind data with SVG shapes

```
let myData = [10, 20, 30, 40, 50];

//bind array data to svg circles
d3.select('.chart')
    .selectAll('circle')
    .data(myData)
    .join('circle')
    .attr('cx', function (d, i) {
        return i * 100 + 100;
    })
    .attr('cy', 50)
    .attr('r', function (d) {
        return 0.5 * d;
    })
    .style('fill', 'steelblue');
```

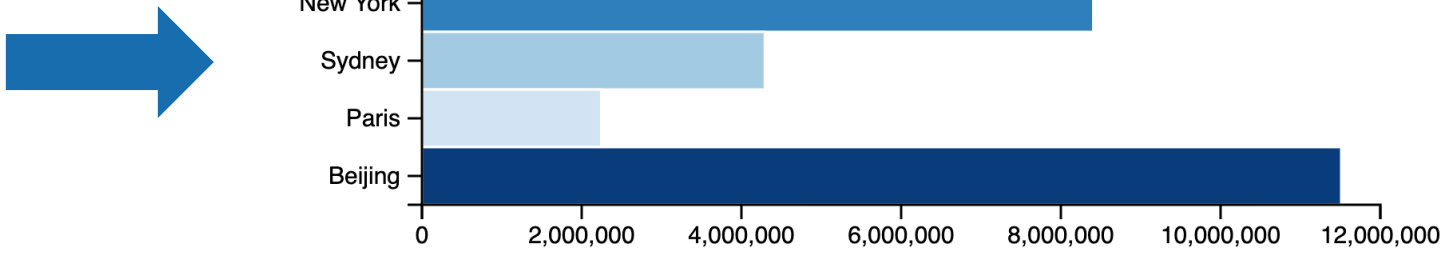
```
<svg>
 <g class="chart">
    <circle cx="100" cy="50" r="5" style="fill: steelblue;"></circle>
    <circle cx="200" cy="50" r="10" style="fill: steelblue;"></circle>
    <circle cx="300" cy="50" r="15" style="fill: steelblue;"></circle>
    <circle cx="400" cy="50" r="20" style="fill: steelblue;"></circle>
    <circle cx="500" cy="50" r="25" style="fill: steelblue;"></circle>
  </g>
</svg>
```

D3 - Data binding

Goals:

- bind name with SVG texts
- bind population with SVG rectangles
  - visual mark: line (SVG rects)
  - visual channel: length/position (width and height of SVG rects)

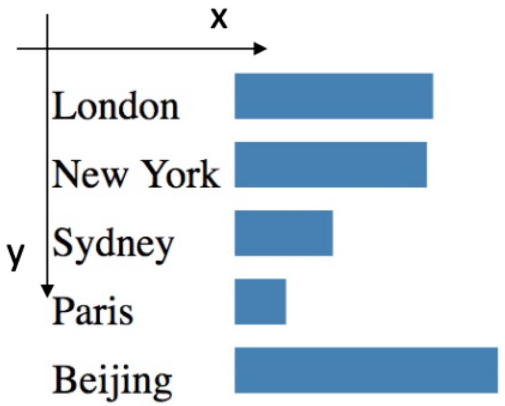
	A	B
1	name	population
2	London	8674000
3	New York	8406000
4	Sydney	4293000
5	Paris	2244000
6	Beijing	11510000



Demo

Questions

```
var bar = svg
    .selectAll("rect")
    .data(data)
    .join("rect")
    .attr("x", 80)
    .attr("y", function (d, i) {
        return i * barHeight;
    })
    .attr("width", function (d, i) {
        return d.population / 20000;
    })
    .attr("height", barHeight)
    .style("fill", "steelblue")
    .style("stroke", "white");
```



Can we have a better solution to map the data value to the length?

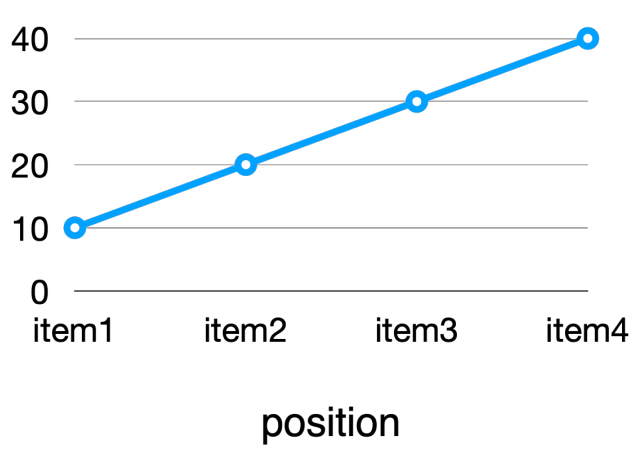
D3 Scale

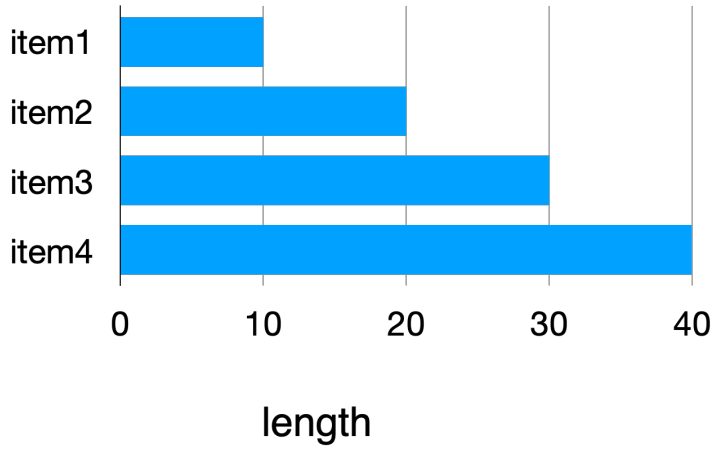
Scale functions:

- Take an input (usually a number, date or category)
- Return a value (e.g., coordinate, color, length, or a radius)

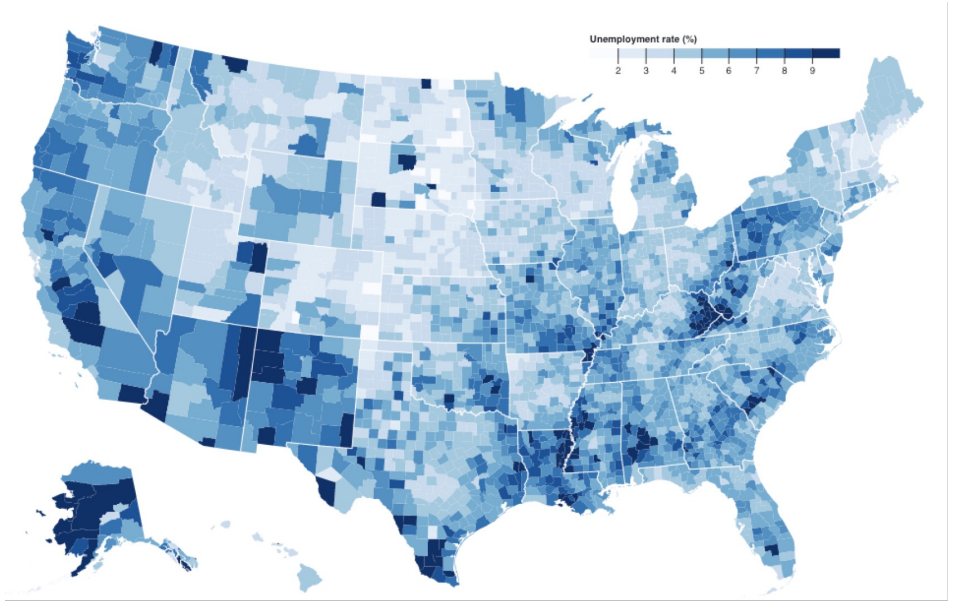
Map data value to visual channels (length, color, size, position).

	Att1	Att2
item 1	10	0.1
item 2	20	0.2
item 3	30	0.3
item 4	40	0.4

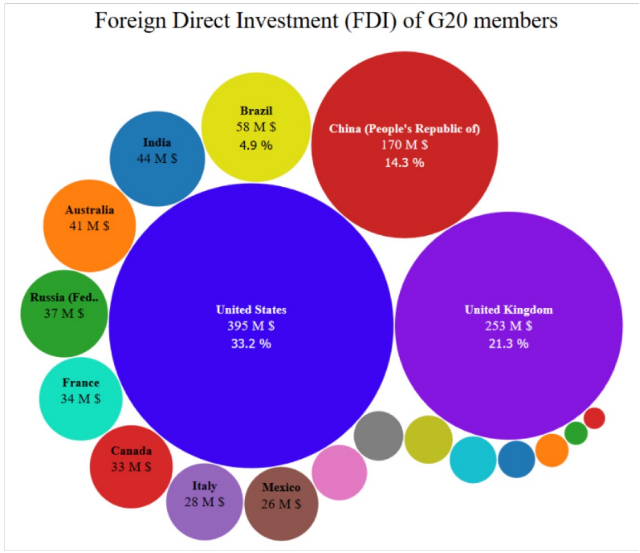




D3 Scale



color



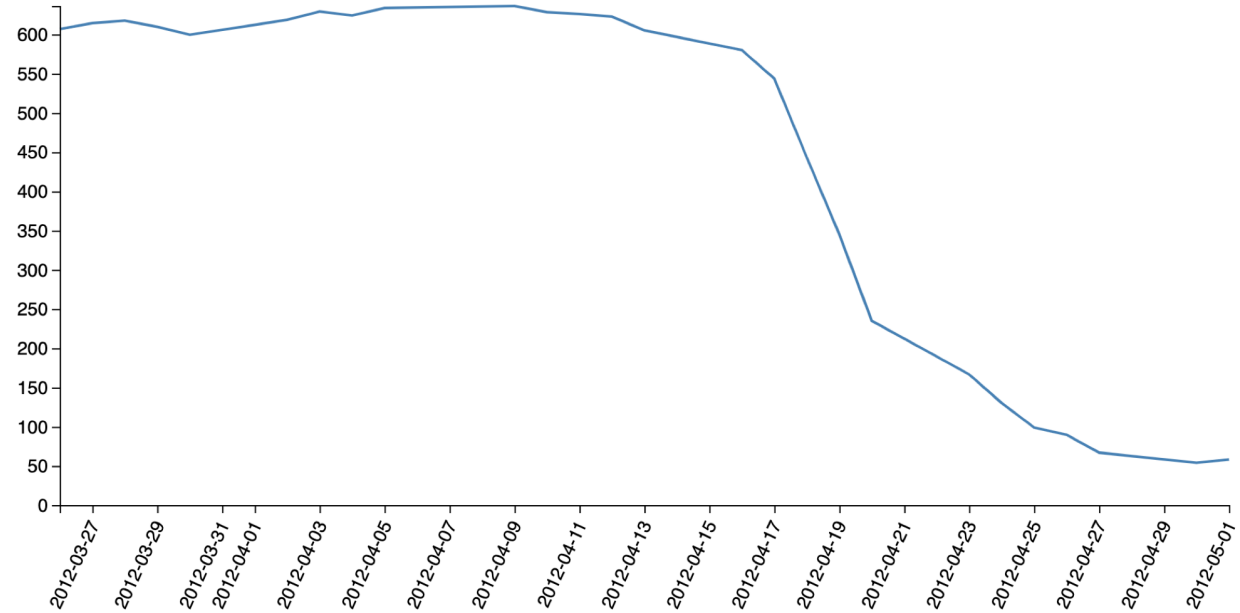
area

D3 Scale

- scales with continuous input and continuous output
- scales with continuous input and discrete output
- scales with discrete input and discrete output

D3 Scale

- scales with continuous input and continuous output
  - scaleLinear()
  - scaleTime()
  - scalePow()
  - scaleSqrt()
  - scaleLog()



D3 Scale

- scaleLinear()
  - domain: continuous interval (e.g., [0,100])
  - range: continuous interval (e.g., [0,600])
  - function: use a linear function y=mx+c to interpolate across the domain and range

```
var data = [0, 1, 2, 3, 4, 5, 6, 7, 7.5, 8, 9, 10];
var linearScale = d3.scaleLinear()
    .domain([0, 10])
    .range([0, 600]);
```

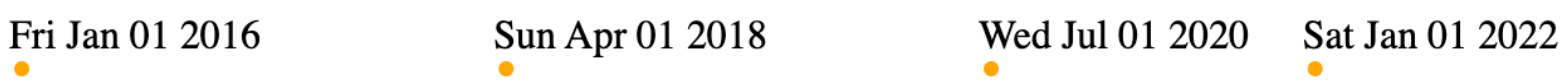
- linearScale(0) => 0
- linearScale(5) => 300
- linearScale(10) => 600

D3 Scale

- scaleTime()
  - domain: continuous interval of dates (e.g., [new Date(2016, 0, 1), new Date(2023, 0, 1)])
  - range: continuous interval (e.g., [0,600])
  - function: use a linear function to interpolate across the domain and range

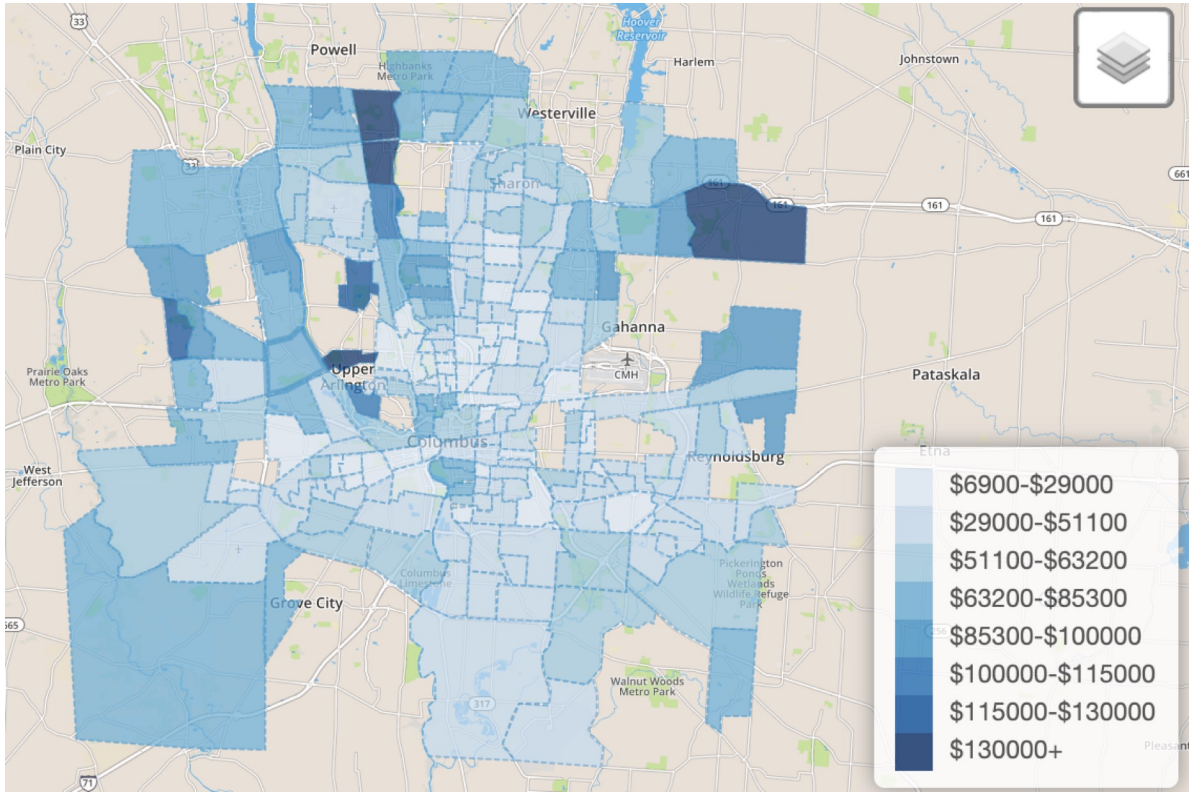
```
var data = [new Date(2016, 0, 1), new Date(2018, 3, 1),
    new Date(2020, 6, 1), new Date(2022, 0, 1)];
var timeScale = d3.scaleTime()
    .domain([new Date(2016, 0, 1), new Date(2023, 0, 1)])
    .range([0, 600]);
```

- linearScale(0) => 0
- linearScale(5) => 300
- linearScale(10) => 600



D3 Scale

- scales with continuous input and discrete output
  - scaleQuantize()
  - scaleQuantile()
  - scaleThreshold()
  - scaleBand()



D3 Scale

- scaleQuantize()
  - domain: continuous interval (e.g., [0,100])
  - range: an array of discrete values (e.g., ['lightblue', 'orange', 'lightgreen', 'pink'])
  - function: segments the domain into k uniform segments (k is the # elements in range)

```
var quantizeScale = d3.scaleQuantize()
    .domain([0, 100])
    .range(['lightblue', 'orange', 'lightgreen', 'pink']);
```

- $0 \le u < 25$: lightblue
- $25 \le u < 50$: orange
- $50 \le u < 75$: lightgreen
- $75 \le u < 100$: pink

D3 Scale

- scaleQuantile()
  - domain: an array of **sorted** continuous numbers (e.g., [0,1,2,3,14,30])
  - range: an array of discrete values (e.g., ['lightblue', 'orange'])
  - function: given n elements in the domain set and k elements in the range set, it segments the domain into k intervals such that the ith n/k elements in the domain set are mapped to the ith element in the range set

```
var myData = [0, 1, 2, 3, 14, 30];
var quantileScale = d3.scaleQuantile()
    .domain(myData)
    .range(['lightblue', 'orange']);
```

- n = 6, k = 2, n/k = 3
- the first 6/2 elements 0,1,2: lightblue
- the second 6/2 elements, 3,14,30: orange

D3 Scale

- scaleThreshold()
  - domain: an array of **sorted** continuous threshold numbers (e.g., [0,50,100])
  - range: an array of discrete values (e.g., ['lightblue', 'orange', 'lightgreen', 'pink'])
  - function: sets the boundaries for the domain's k segments

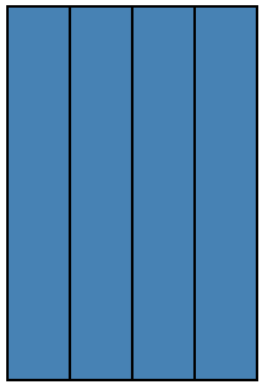
```
var myData = d3.range(-10, 110, 2);
var thresholdScale = d3.scaleThreshold()
    .domain([10, 50, 100])
    .range(['#feebe2', '#fbb4b9', '#f768a1', '#ae017e']);
```

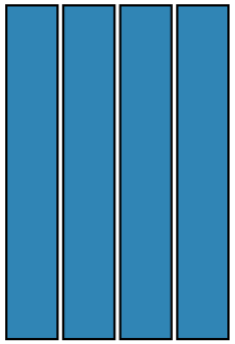
- $u < 10$: #feebe2
- $10 \le u < 50$: #fbb4b9
- $50 \le u < 100$: #f768a1
- $u > 100$: #ae017e

D3 Scale

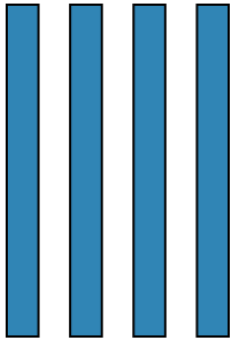
- scaleBand()
  - domain: an array of discrete values (e.g., ['Jan', 'Feb', 'Mar']
  - range: an interval of discrete values (e.g., [0,200])
  - function: split the range into n bands (n: # elements in the domain)
  - paddingInner: the amount of padding between each band.
  - paddingOuter: the amount of padding before the first band and after the last band
  - interactive demo

```
var myData = ["one", "two", "three", "four"];
var bandScale = d3.scaleBand()
    .domain(myData)
    .range([0, 100])
```





paddingInner = 0.1



paddingInner = 0.5

D3 Scale

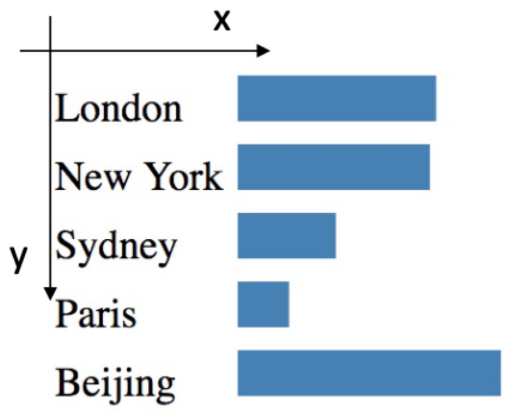
- scales with discrete input and discrete output
- scaleOrdinal()
  - domain: an array of discrete values (e.g., ['Jan', 'Feb', 'Mar']
  - range: an array of discrete values (e.g., ['lightblue', 'orange', 'lightgreen', 'pink'])
  - function: maps discrete values in the domain to values in the range (1 to 1)

```
var data = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];

var ordinalScale = d3.scaleOrdinal()
    .domain(data)
    .range(['blue', 'orange', 'green', 'red']);
```

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

Draw the bars

	A	B
1	name	population
2	London	8674000
3	New York	8406000
4	Sydney	4293000
5	Paris	2244000
6	Beijing	11510000



x => map the population value

- from [0, 12000000] continuous
- to [0,200] continuous
- scaleLinear()

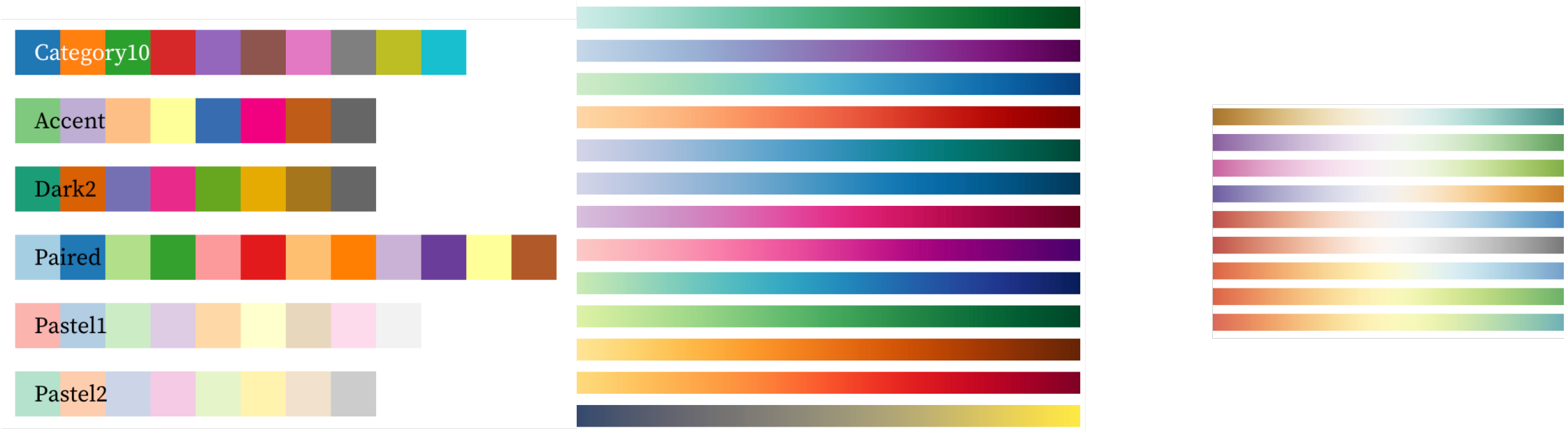
y => map the name

- from [London, ..., Beijing] discrete
- to [0,150] continuous
- scaleBand()

Demo

D3 Color mapping

- d3 color schemes: https://github.com/d3/d3-scale-chromatic/blob/main/README.md
- color brewer 2.0: http://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3



Continuous color schemes

- Pattern

```
//continuous
let colorScale = d3.scaleSequential()
        .domain(data) //interval
        .interpolator(d3.interpolate + colorname)
```

```
var myData = d3.range(0, 100, 2);
var sequentialColorScale = d3.scaleSequential()
    .domain([0, 100])
    .interpolator(d3.interpolateRdYlBu);
```

Discrete color schemes

- Pattern

```
//categorical
let colorScale = d3.scaleOrdinal()
        .domain(data)  //array
        .range(d3.scheme + colorname)
```

```
var myData = d3.range(0, 100, 10);
var colorScale = d3.scaleOrdinal()
    .domain(myData)
    .range(d3.schemeCategory10);
```

D3 Color mapping

- Accessing color values
  - `d3.scheme<colorname>[k]`
  - for continuous color schemes, k should be less than 10
  - you can use continuous color schemes in scaleOrdinal through this approach

```
d3.schemeBlues[9]
▼ (9) ['#f7fbff', '#deebf7', '#c6dbef', '#9ecae1', '#6baed6', '#4292c6', '#2171b5', '#08519c', '#08306b'] i
    0: "#f7fbff"
    1: "#deebf7"
    2: "#c6dbef"
    3: "#9ecae1"
    4: "#6baed6"
    5: "#4292c6"
    6: "#2171b5"
    7: "#08519c"
    8: "#08306b"
    length: 9
  ▶ [[Prototype]]: Array(0)
```

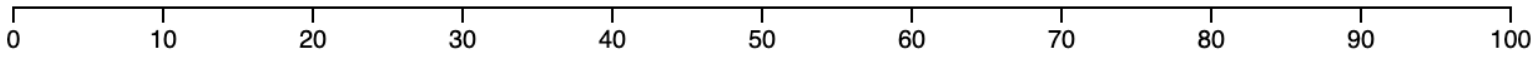
```
d3.schemeSet3
▼ (12) ['#8dd3c7', '#ffffb3', '#bebada', '#fb8072', '#80b1d3', '#fdb462', '#b3de69', '#fccde5', '#d9d9d9', '#bc80bd', '#ccebc5', '#ffed6f'] i
    0: "#8dd3c7"
    1: "#ffffb3"
    2: "#bebada"
    3: "#fb8072"
    4: "#80b1d3"
    5: "#fdb462"
    6: "#b3de69"
    7: "#fccde5"
    8: "#d9d9d9"
    9: "#bc80bd"
    10: "#ccebc5"
    11: "#ffed6f"
```

Demo

D3 Axis

- Create an axis:
  - an SVG element as a container
  - a D3 scale function
  - create an axis generator function
  - call the axis generator function

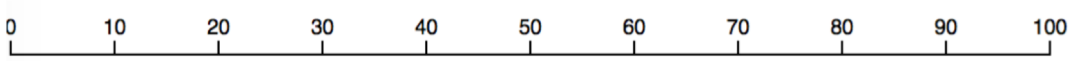
```
var linearScale = d3.scaleLinear()
    .domain([0, 100])
    .range([0, 600]);

let axis = d3.axisBottom(linearScale);

d3.select('.chart')
    .call(axis);
```



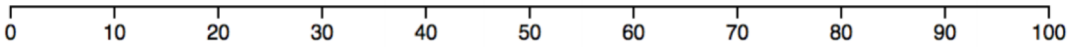
Demo

D3 Axis - Orientation

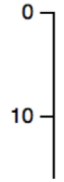
- d3.axisTop(*scale*)



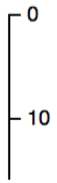
- d3.axisBottom(*scale*)



- d3.axisLeft(*scale*)

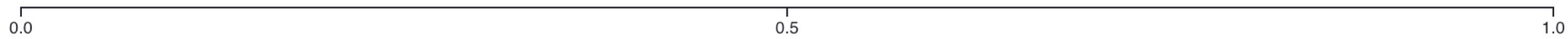


- d3.axisRight(*scale*)



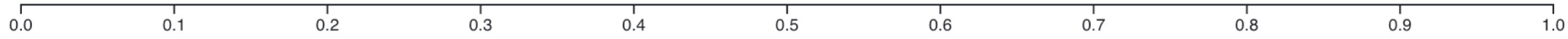

```
var linearScale = d3.scaleLinear()
    .domain([0, 100])
    .range([0, 400]);

let axisLeft = d3.axisLeft(linearScale);
let axisRight = d3.axisRight(linearScale);
let axisTop = d3.axisTop(linearScale);
let axisBottom = d3.axisBottom(linearScale);

d3.select('#left').call(axisLeft);
d3.select('#right').call(axisRight);
d3.select('#top').call(axisTop);
d3.select('#bottom').call(axisBottom);
```

D3 Axis – Number of ticks



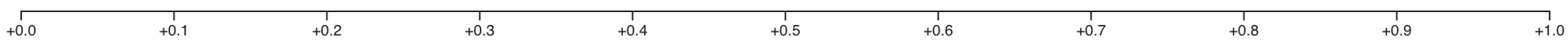

```
axis(d3.scaleLinear())
  .ticks(2)
  .render()
```


```
axis(d3.scaleLinear())
  .ticks(5)
  .render()
```

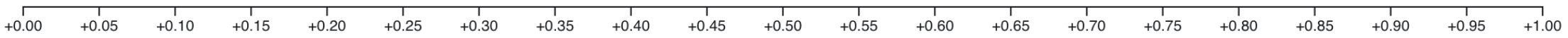



```
axis(d3.scaleLinear())
  .ticks(10)
  .render()
```

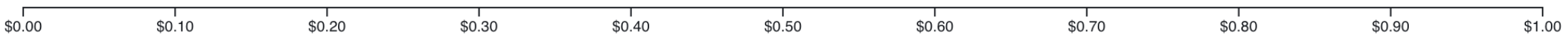
D3 Axis – Tick format




```
axis(d3.scaleLinear())
  .ticks(10, "+f") // Implicit precision of one.
  .render()
```




```
axis(d3.scaleLinear())
  .ticks(15, "+f") // Implicit precision of two.
  .render()
```




```
axis(d3.scaleLinear())
  .ticks(10, "$.2f") // Explicit precision of two.
  .render()
```

Summary – Bar chart example

- Prepare the data
  - load CSV file
- Mapping
  - create the scale function
- Rendering
  - create SVG element
  - join data with SVG rect and text
  - render the axis