D3 Tutorial
Data Transformation and Scale Functions

Edit by Jiayi Xu and Han-Wei Shen, The Ohio State University
Populations of cities - Scaling

• Scale populations
  • so that we can display bars within the screen

```javascript
var scaleFactor = 1e-5;
var rects = svg.selectAll("rect")
  .data(cities)
  .enter().append("rect")
  .attr("x", 80)
  .attr("y", function(d, i) {
    return padding + i * (barHeight + padding);
  })
  .attr("width", function(d, i) {
    return d.population * scaleFactor;
  })
  .attr("height", barHeight)
  .style("fill", "steelblue");

var cities = [
  { name: 'London', population: 8674000},
  { name: 'New York', population: 8406000},
  { name: 'Sydney', population: 4293000},
  { name: 'Paris', population: 2244000},
  { name: 'Beijing', population: 11510000}
];
```

• London: 86.74 pixels
• New York: 84.06 pixels
• Sydney: 42.93 pixels
• Paris: 22.44 pixels
• Beijing: 115.1 pixels
Data Transformation Using Scale Functions

• Scale functions of D3
  • Map from an input domain to an output range
    • Usually, map a dimension/attribute of data to a visual variable
  • Take an input
    • usually a number, date or category
  • Return a value
    • e.g., a coordinate, a color, a length or a radius
Data Transformation

• Scale factor = $10^5$

  • Mapping
  • From $[0, 11\,510\,000]$ (domain)
  • To $[0, 115.1]$ (range)

```javascript
var scaleFactor = 1e-5;
var rects = svg.selectAll("rect")
  .data(cities)
  .enter().append("rect")
  .attr("x", 80)
  .attr("y", function(d, i) {
    return padding + i * (barHeight + padding);
  })
  .attr("width", function(d, i) {
    return d.population * scaleFactor;
  })
  .attr("height", barHeight)
  .style("fill", "steelblue");
```

var cities = [
  { name: 'London', population: 8674000 },
  { name: 'New York', population: 8406000 },
  { name: 'Sydney', population: 4293000 },
  { name: 'Paris', population: 2244000 },
  { name: 'Beijing', population: 11510000 }
];

• London: 86.74 pixels
• New York: 84.06 pixels
• Sydney: 42.93 pixels
• Paris: 22.44 pixels
• Beijing: 115.1 pixels
Scale Function - d3.scaleLinear()

• Mapping
  • From [0, 11 510 000] (domain)
  • To [0, 115.1] (range)

• Extend mapping for more generality
  • From [0, 20 000 000] (domain)
  • To [0, 200] (range)

• pop2width – **Population to Width**
  • d3.scaleLinear()
    • Linear mapping

```javascript
var pop2width = d3.scaleLinear()
  .domain([0, 2*1e7])
  .range([0, 200]);

var rects = svg.selectAll("rect")
  .data(cities)
  .enter().append("rect")
  .attr("x", 80)
  .attr("y", function(d, i) {
    return padding + i * (barHeight + padding);
  })
  .attr("width", function(d, i) {
    return pop2width(d.population);
  })
  .attr("height", barHeight)
  .style("fill", "steelblue");
```
Categories of Scale Functions

• Categories
  • Continuous Input -> Continuous Output
  • Continuous Input -> Discrete Output
  • Discrete Input -> Discrete Output

• We’ll now look at these 3 categories one by one
Continuous Input -> Continuous Output

d3.scaleLinear() again

• They use a linear function $y = k \cdot x + b$ to interpolate across the domain ($x$) and range ($y$)

```javascript
var value2width = d3.scaleLinear()
  .domain([0, 10])
  .range([0, 600]);

var data = [0, 1, 2, 3, 4, 5, 6, 7, 7.5, 8, 9, 10];
```
Continuous Input -> Continuous Output
d3.scalePow()

- The pow scale interpolates using a power \( y = m \cdot x^k + b \) function.
  - The exponent \( k \) is set using .exponent():

```
var value2width = d3.scalePow()
  .exponent(2)
  .domain([0, 10])
  .range([0, 600]);

var data = [0, 1, 2, 3, 4, 5, 6, 7, 7.5, 8, 9, 10];
```

[Graph showing interpolation points] 600 px
Continuous Input -> Continuous Output

d3.scaleSqrt()

• The scaleSqrt scale is a special case of the pow scale (where $k = 0.5$)

```javascript
var value2width = d3.scaleSqrt()
  .domain([0, 10])
  .range([0, 600]);
```

```javascript
var data = [0, 1, 2, 3, 4, 5, 6, 7, 7.5, 8, 9, 10];
```
Continuous Input -> Continuous Output

\[ d3\text{.scaleLog()} \]

- Log scales interpolate using a log function \( y = m \cdot \log(x) + b \)
  - useful when the data has an exponential nature to it

```javascript
var value2width = d3.scaleLog()
  .domain([10, 100000])
  .range([0, 600]);

var data = [10, 100, 1000, 10000, 100000];
```
Continuous Input -> Continuous Output

d3.scaleTime() 

• scaleTime is similar to scaleLinear 
  • The domain is expressed as an array of dates 
  • useful when dealing with time series data

```javascript
var time2width = d3.scaleLog()
  .domain([new Date(2016, 0, 1), new Date(2017, 0, 1)])
  .range([0, 600]);

var data = [new Date(2016, 0, 1), new Date(2016, 3, 1), new Date(2016, 6, 1), new Date(2017, 0, 1)];
```
Continuous Input -> Continuous Output

Multiple Segments

- The domain and range of scale functions usually consists of two values, but if we provide 3 or more values the scale function is subdivided into multiple segments.

```javascript
var value2color = d3.scaleLinear()
    .domain([-10, 0, 10])
    .range(['red', '#ddd', 'blue']);
```

#ddd is hexadecimal representation of rgb(221, 221, 221)
Light Grey
Continuous Input -> Continuous Output

d3.scaleSequential(*interpolator*)

• Mapping continuous values to an output range determined by a preset (or custom) *interpolator*
  • Useful to create a continuous colormap

• Usage
  • d3.scaleSequential(*interpolator*);
    • Domain is [0, 1]
  • Or, d3.scaleSequential().domain(*domain*).interpolator(*interpolator*);

```javascript
var value2color = d3.scaleSequential()
  .domain([-10, 10])
  .interpolator(d3.interpolateRainbow);
```

![Color scale with values ranging from -10 to 10 and corresponding colors]

d3.interpolateRainbow
Continuous Input -> Continuous Output
d3.scaleSequential(\textit{interpolator})

- D3 provides a great many interpolators
  - https://github.com/d3/d3-scale-chromatic
- Diverging
  - d3.interpolateBrBG
- Single Hue
  - d3.interpolateBlues
- Multi-Hue
  - d3.interpolateViridis
- Cyclical
  - d3.interpolateRainbow
Continuous Input -> Discrete Output
d3.scaleQuantize()

- **Discrete** output
- scaleQuantize accepts continuous input and outputs a number of discrete quantities defined by the range

```javascript
var value2color = d3.scaleQuantize()
    .domain([0, 100])
    .range(['lightblue', 'orange', 'lightgreen', 'pink']);
```

- \( value < 25 \) is mapped to ‘lightblue’
- \( 25 \leq value < 50 \) is mapped to ‘orange’
- \( 50 \leq value < 75 \) is mapped to ‘lightgreen’
- \( value \geq 75 \) is mapped to ‘pink’
Continuous Input -> Discrete Output

d3.scaleQuantile()

• Quantile scales map a sampled input domain to a discrete range
  • Domain accepts a set of sample values

```javascript
var myData = [0, 1, 2, 3, 4, 30, 35, 40, 60, 62, 65, 70, 80, 90, 100];
var value2color = d3.scaleQuantile()
  .domain(myData)
  .range(['lightblue', 'orange', 'lightgreen']);
```

• the first 5 values are mapped to ‘lightblue’
• the next 5 values to ‘orange’
• the last 5 values to ‘lightgreen’.
Continuous Input -> Discrete Output
d3.scaleThreshold()

• Map arbitrary subsets of the domain to discrete values in the range

```javascript
var value2color = d3.scaleThreshold()
  .domain([0, 50, 100])
  .range(['lightblue', 'orange', 'lightgreen', 'pink']);
```

• $value < 0$ is mapped to ‘lightblue’
• $0 \leq value < 50$ is mapped to ‘orange’
• $50 \leq value < 100$ is mapped to ‘lightgreen’
• $value \geq 100$ is mapped to ‘pink’
Discrete Input -> Discrete Output
d3.scaleOrdinal()

• **Discrete** input and **discrete** output

• scaleOrdinal maps discrete values (specified by an array) to discrete values (also specified by an array)
  • The range array will repeat if it’s shorter than the domain array.

```javascript

var month2color = d3.scaleOrdinal()
  .domain(data)
  .range([['black', 'grey', 'lightgrey']]));
```
Discrete Input -> Discrete Output

d3.scaleOrdinal(colorScheme)

• Use D3 built-in color schemes
  • d3.scaleOrdinal(colorScheme)
  • d3.schemeCategory10: map 0 ~ 9 to nine colors

```javascript
var value2color = d3.scaleOrdinal(d3.schemeCategory10);
```

• D3 also provides a great many ordinal color schemes
  • https://github.com/d3/d3-scale-chromatic
Discrete Input -> Discrete Output
d3.scaleBand()

- Discrete output values are automatically computed by the scale by dividing the continuous range into uniform bands
  - Band scales are typically used for bar charts with an ordinal or categorical dimension

- `paddingOuter` and `paddingInner` are ratios
Discrete Input -> Discrete Output
d3.scaleBand()

- Data
  - Populations of Cities
- d3.scaleBand()
  - Domain is the names of cities

```javascript
var cityNames = cities.map(function(d) { return d.name });
var bandScale = d3.scaleBand()
  .domain(cityNames)
  .range([0, 160])
  .paddingOuter(0.33)
  .paddingInner(0.33);
```
Discrete Input -> Discrete Output

d3.scaleBand()

• Draw bars by `scaleBand`

```javascript
var rects = svg.selectAll("rect")
  .data(cities)
  .enter().append("rect")
  .attr("x", 80)
  .attr("y", function(d) {
    return bandScale(d.name);
  })
  .attr("width", function(d, i) {
    return pop2width(d.population);
  })
  .attr("height", bandScale.bandwidth())
  .style("fill", "steelblue");
```
Discrete Input -> Discrete Output
d3.scalePoint()

- Point scales are a variant of band scales with the bandwidth fixed to zero
  - Point scales are typically used for scatterplots with an ordinal or categorical dimension

- `padding` is a ratio

![Diagram showing point scales and padding ratio]
Discrete Input -> Discrete Output
d3.scalePoint()

• Data
  • Daily sales of fruit Apricots

```javascript
var data = [
  {day: 'Mon', value: 120},
  {day: 'Tue', value: 60},
  {day: 'Wed', value: 100},
  {day: 'Thu', value: 80},
  {day: 'Fri', value: 120}
];
```
Discrete Input -> Discrete Output

d3.scalePoint()

- Create scales
  - d3.scalePoint()
    - Map day of week to x coordinate
  - d3.scaleLinear()
    - Map daily sales to y coordinate

```javascript
var pointScale = d3.scalePoint()
  .domain(['Mon', 'Tue', 'Wed', 'Thu', 'Fri'])
  .range([0, 600])
  .padding(0.1);

var value2height = d3.scaleLinear()
  .domain([0, d3.max(data, function(d) {return d.value;})])
  .range([200, 0]);

var data = [
  {day : 'Mon', value: 120},
  {day : 'Tue', value: 60},
  {day : 'Wed', value: 100},
  {day : 'Thu', value: 80},
  {day : 'Fri', value: 120}
];
```
Discrete Input -> Discrete Output
d3.scalePoint()

• Draw points

```javascript
var circles = d3.select('#wrapper')
    .selectAll('circle')
    .data(data)
    .enter()
    .append('circle')
    .attr('cx', function(d) {
        return pointScale(d.day);
    })
    .attr('cy', function(d) {
        return value2height(d.value);
    })
    .attr('r', 4);

var data = [
    {day: 'Mon', value: 120},
    {day: 'Tue', value: 60},
    {day: 'Wed', value: 100},
    {day: 'Thu', value: 80},
    {day: 'Fri', value: 120}
];
```