

# Information Visualization

## D3.js - 1

Original by 조재민 (Jaemin Jo, <http://github.com/e->)

Modified by Jinwook Bok (bok@hcil.snu.ac.kr)

---

Human-Computer Interaction Laboratory  
Seoul National University

- 
- This image is a dense, multi-colored collage of various data visualization techniques. The elements are arranged in a roughly hexagonal grid pattern. Key components include:
- Maps:** Several geographical maps are visible, including a world map, a map of Europe, and a map of Africa.
  - Bar Charts:** Multiple bar charts in various colors (blue, green, red, orange) are scattered throughout, some showing frequency distributions.
  - Pie Charts:** Several pie charts and donut charts are present, displaying different data segments in various colors.
  - Network Graphs:** There are several network diagrams, including a large, complex graph on the right side and smaller, simpler graphs elsewhere.
  - Abstract Patterns:** The collage features numerous abstract patterns, such as hexagonal grids, circular patterns, and various geometric shapes.
  - Text and Labels:** Some text labels are visible, such as "30 sec", "studies", "science", and "graphy", which appear to be part of the data visualizations.
- The overall aesthetic is modern and data-driven, with a focus on visual complexity and information density.


## D3.js

- 저널 페이퍼로도 발간됨

- <http://vis.stanford.edu/files/2011-D3-InfoVis.pdf>
- Bostock, M., Ogievetsky, V., & Heer, J. (2011). D<sup>3</sup>: Data-Driven Documents. *IEEE Transactions on Visualization & Computer Graphics*, (12), 2301-2309

- 인기 많음

- 논문: 2000여번 인용
- 다양한 시각화 라이브러리의 core
- 스타 많음

 d3 / d3 Watch




4,046

 Star

87,707

 Fork

21,371

 <> Code Issues 4 Pull requests 0 Wiki Security Insights

## Version

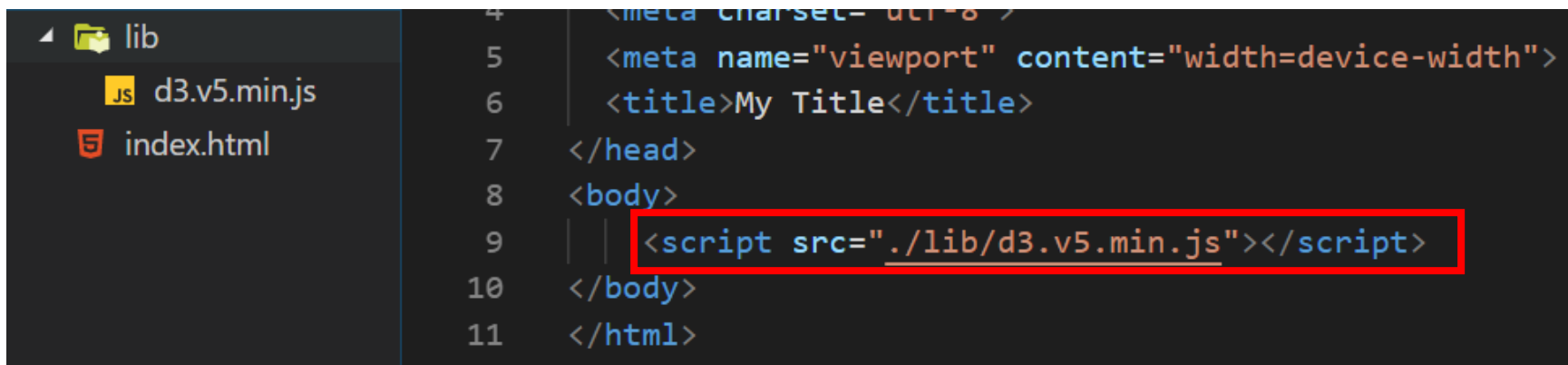
- 현재 최신 D3의 버전 : 5.x
  - [vs. 4.0] 많이 바뀌지 않음 (사용해도 거의 문제 없음)
  - [vs. 3.0] 라이브러리의 모듈화 등등... 많은 변화 (사용 불가)
  - <https://github.com/d3/d3/blob/master/CHANGES.md>
- 배울 때 힘든 점
  - D3 관련 서적, 웹 상의 자료 및 코드가 3 ~ 5 버전에 걸쳐 혼재함  
→ 자료 참고 시 버전에 유의할 것
- 이번 과정 실습에서는 5 버전을 사용

## D3.js의 설치

- Script 태그를 추가
  - 그때 그때 다운로드 하여 사용 - URL 적기

```
<body>  
  <script src="https://d3js.org/d3.v5.min.js"></script>  
</body>
```

- 한 번 받아놓고 로컬에서 라이브러리 파일 사용 - 경로 적기



## Arrow Function Expression

- Introduced in ES 6

```
let my_func = function(a, b) { return a + b; }
```

```
my_func = (a, b) => { return a + b; }
```

```
my_func = (a, b) => a + b;
```

```
my_func = a => a + 1;
```

- Not equivalent to function(){}!

- "lexical this"
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow\\_functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

```
(param1, param2, ..., paramN) => { statements }  
(param1, param2, ..., paramN) => expression  
    // 다음과 동일함: => { return expression; }  
  
// 매개변수가 하나뿐인 경우 괄호는 선택사항:  
(singleParam) => { statements }  
singleParam => { statements }  
  
// 매개변수가 없는 함수는 괄호가 필요:  
() => { statements }
```

## Selection

- D3.js에서 DOM 조작을 위해 요소를 선택하는 것
- rect를 선택하기

```
<body>
  <svg height="1000" width="1000">
    <rect x="0" y="0" width="100" height="50"></rect>
    <rect x="0" y="100" width="200" height="50"></rect>
    <rect x="0" y="200" width="300" height="50"></rect>
  </svg>
  <script src="https://d3js.org/d3.v5.min.js"></script>
  <script>
    d3.select('rect')
      .attr('width', '600');
  </script>
</body>
```

## Selection

- `d3.select(selector)`
  - 문서 전체에서 `selector`에 해당하는 첫 번째 요소를 선택한 뒤, selection을 리턴
- `selection.attr(name[, value])`
  - `selection`이 갖고 있는 요소의 `name` 속성에 `value`를 지정

```
<body>
  <svg height="1000" width="1000">
    <rect x="0" y="0" width="100" height="50"></rect>
    <rect x="0" y="100" width="200" height="50"></rect>
    <rect x="0" y="200" width="300" height="50"></rect>
  </svg>
  <script src="https://d3js.org/d3.v5.min.js"></script>
  <script>

    d3.select('rect')
      .attr('width', '600');

  </script>
</body>
```



## d3.selectAll

- d3.selectAll(*selector*)
  - 문서 전체에서 *selector*에 해당하는 element를 모두 선택한 뒤, *selection*을 리턴

```
d3.selectAll('rect')  
  .attr('width', '600');
```

## 속성, 스타일 지정

- `selection.attr()`, `selection.style()`, ...
  - 이전 시간에 배운 `map`, `forEach` 처럼 *selection* 안의 *element* 배열에 대해 순차적으로 실행됨
  - 이후 *selection*을 리턴 (chaining 가능)
- `d3.selectAll("p").style("font-size", "10px")`
- `d3.selectAll("p").style("font-size", (d, i) => i * 10 + "px")`
- `d3.selectAll("p").style("font-size", "10px").style("color", "red")`

## Empty Selection

- 문서 내에 없는 요소를 선택할 경우

```
<body>
  <svg height="1000" width="1000">
  </svg>
  <script src="https://d3js.org/d3.v5.min.js"></script>
  <script>
    d3.select('rect')
      .attr('width', '600');
  </script>
</body>
```

### Empty Selection

- 존재하지 않는 요소를 select  
→ empty selection 반환
- *empty selection* 의 경우 attr(), style()을 수행해도 아무일도 일어나지 않음

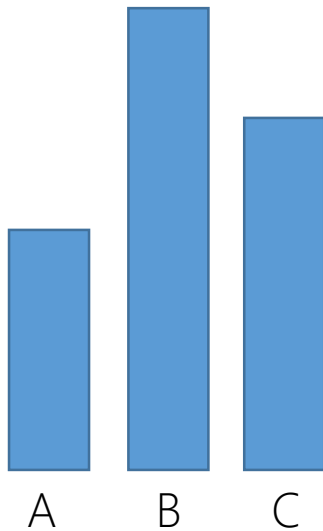
## Data Join

- D3.js helps you bring data to life using HTML, SVG, and CSS.
- D3.js 의 데이터 단위는 배열
  - number, string, object array 등 모든 타입의 배열이 가능
- (예제) 내가 가진 데이터가 숫자 배열일 경우

```
<body>
  <svg height="1000" width="1000">
  </svg>
  <script src="https://d3js.org/d3.v5.min.js"></script>
  <script>
    let my_data = [1, 2, 3];
  </script>
</body>
```

## selection.data 이해하기

```
let oldData = [  
  {name: 'A', value: 10},  
  {name: 'B', value: 20},  
  {name: 'C', value: 15}  
]
```



{name: 'A', value: 10}

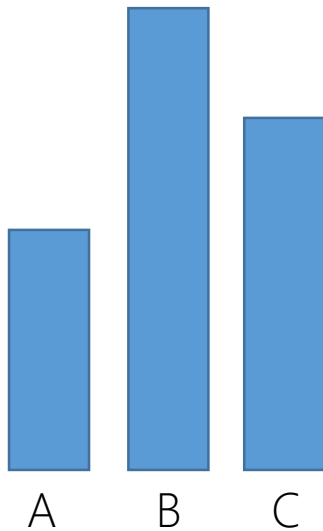
{name: 'B', value: 20}

{name: 'C', value: 15}

각 오브젝트가 바에  
연결됨

## selection.data 이해하기

```
let oldData = [  
  {name: 'A', value: 10},  
  {name: 'B', value: 20},  
  {name: 'C', value: 15}  
]
```



```
{name: 'A', value: 10}  
{name: 'B', value: 20}  
{name: 'C', value: 15}
```

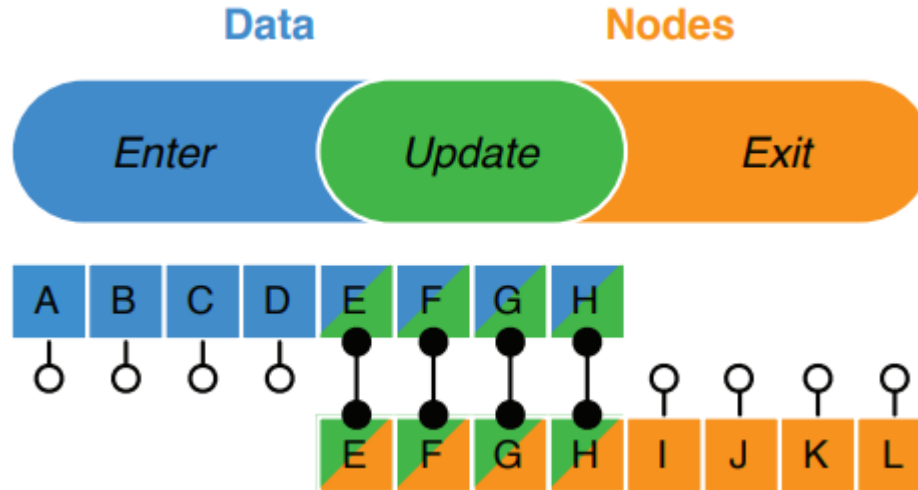
각 오브젝트가 바에  
연결됨

```
let newData = [  
  {name: 'B', value: 10},  
  {name: 'C', value: 15},  
  {name: 'D', value: 5}  
]
```

데이터 변경!

name을 키라고 했을 때,  
A: 없어짐 (exit)  
B, C: 변경됨 (update)  
D: 추가됨 (enter)

## Selection의 종류



Update selection

: 데이터와 성공적으로 조인된 요소들만 가지고 있다.

Enter selection

: 요소와 조인되지 못한 데이터들만 가지고 있다.

Exit selection

: 데이터와 조인되지 못한 요소들만 가지고 있다.



## Data Join

- `selection.data([data[, key]])`
  - `selection` 내의 element들과 주어진 `data` (배열)을 연결시키고 update selection을 반환 함.
  - Update selection에서 `enter()`, `exit()` 함수를 호출하여 `enter`, `exit` selection을 가져올 수 있음.

```
<script>  
  
  let my_data = [1, 2, 3];  
  
  d3.selectAll('rect')  
    .data(my_data);  
  
</script>
```

## Common Scenario

- Enter selection에 있는 데이터들은 시각적 요소를 만들어 줘야 함
  - `selection.enter().append('rect')`
- Update selection에 있는 요소들은 데이터의 값에 따라 시각적 요소의 속성을 변경해 줘야 함
  - 크기, 위치, 색깔, ... `attr`, `style` 메소드를 사용해서
  - 이는 새로 추가된 enter selection의 요소에도 마찬가지로
- Exit selection에 있는 요소들은 DOM 에서 제거

## Common Scenario

```
let newData = [
  {name: 'B', value: 10},
  {name: 'C', value: 15},
  {name: 'D', value: 5}
]

let svg = d3.select('svg')
let bars = svg.selectAll('rect').data(newData, d => d.name)

bars.enter()
  .append('rect')
  .merge(bars)
  .attr('width', 30)
  .attr('height', d => d.value * 10)
  .attr('transform', (d, i) => `translate(' + i * 40 + ', 0)`)

bars.exit().remove();
```



## FAQ

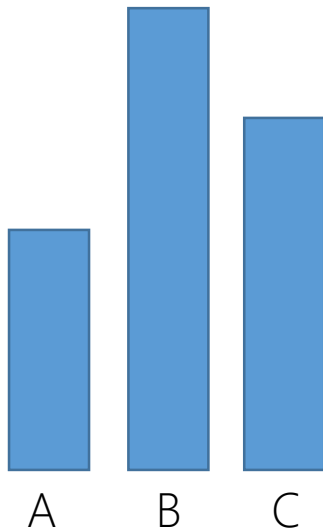
- 처음에는 어떻게 되나요?
  - 처음엔 모든 데이터가 DOM 요소를 가지고 있지 않기 때문에, 모든 데이터가 enter selection에 들어가게 됩니다.
- 왜 굳이 이렇게 하나요? 그냥 다 지웠다가 모두 다시 그리면 안되나요?
  - 성능, 트랜지션, 일반화, incremental processing, ...

```
svg.selectAll('rect').data(newData, d => d.name)
```

- Data 메소드 인자로 왜 함수를 넘기나요?
  - 기본적으로 index를 가지고 join을 하기 때문에 키를 써봤습니다.

## selection.data 이해하기

```
let oldData = [
  {name: 'A', value: 10},
  {name: 'B', value: 20},
  {name: 'C', value: 15}
]
```



```
{name: 'A', value: 10}
{name: 'B', value: 20}
{name: 'C', value: 15}
```

각 오브젝트가 바에  
연결됨

```
let newData = [
  {name: 'B', value: 10},
  {name: 'C', value: 15},
  {name: 'D', value: 5}
]
```

데이터 변경!

```
svg.selectAll('rect').data(newData)
```

키를 지정하지 않았다면,  
세개의 rect가 모두 update  
selection에 있게 됩니다.

selection.append

- *selection.append(type)*

[ Update ]

: *selection*이 갖는

- (1) **각각의 요소들에 대해,**
- (2) *type* 이름의 새로운 요소를
- (3) **마지막 자식으로 추가한다.**

[ Enter ]

: selectAll을 수행한 부모 요소에 현재 enter selection과  
조인 된 새로운 요소가 추가된다.

## 데이터를 반영한 그리기

- selection.style, selection.attr 메소드들의 인자로는
  - 현재 데이터 (d)
  - 현재 요소의 selection 상의 (i)
  - Selection이 가지고 있는 요소들의 배열 (nodes) 가 설정된다.
- selection.attr('width', (d, i, nodes) => d.value \* i)

## 사용할 데이터

- JavaScript 배열 형태의 영화 데이터: <https://pastebin.com/7WNHLVYk>

- Column 정보

- title: 제목 / genre: 장르
- creative\_type: 창작 유형 / source: 원작 / release: 시대
- rating: 등급 / budget: 제작비
- **us\_gross: 미국 수익** / worldwide\_gross: 전세계 수익
- **rotten\_rating: 로튼토마토 평점**
- imdb\_rating: IMDB 평점
- imdb\_votes: IMDB 평가수

```
let data =  
[  
  {  
    "title": "Twin Falls Idaho",  
    "genre": "드라마",  
    "creative_type": "현대소설",  
    "source": "없음",  
    "release": "90년대",  
    "rating": "전체관람가",  
    "budget": "5",  
    "us_gross": "9",  
    "worldwide_gross": "10",  
    "rotten_rating": "77",  
    "imdb_rating": "7.1",  
    "imdb_votes": "2,810"  
  },  
  {  
    "title": "Beverly Hills Cop II",  
    "genre": "액션",  
    "creative_type": "현대소설",
```



## 데이터 파싱(Parsing)

- string → number

```
data.forEach(function(d) {  
    d.us_gross = parseFloat(d.us_gross);  
    d.rotten_rating = parseFloat(d.rotten_rating);  
})
```

- parseFloat

- [https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\\_Objects/parseFloat](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/parseFloat)

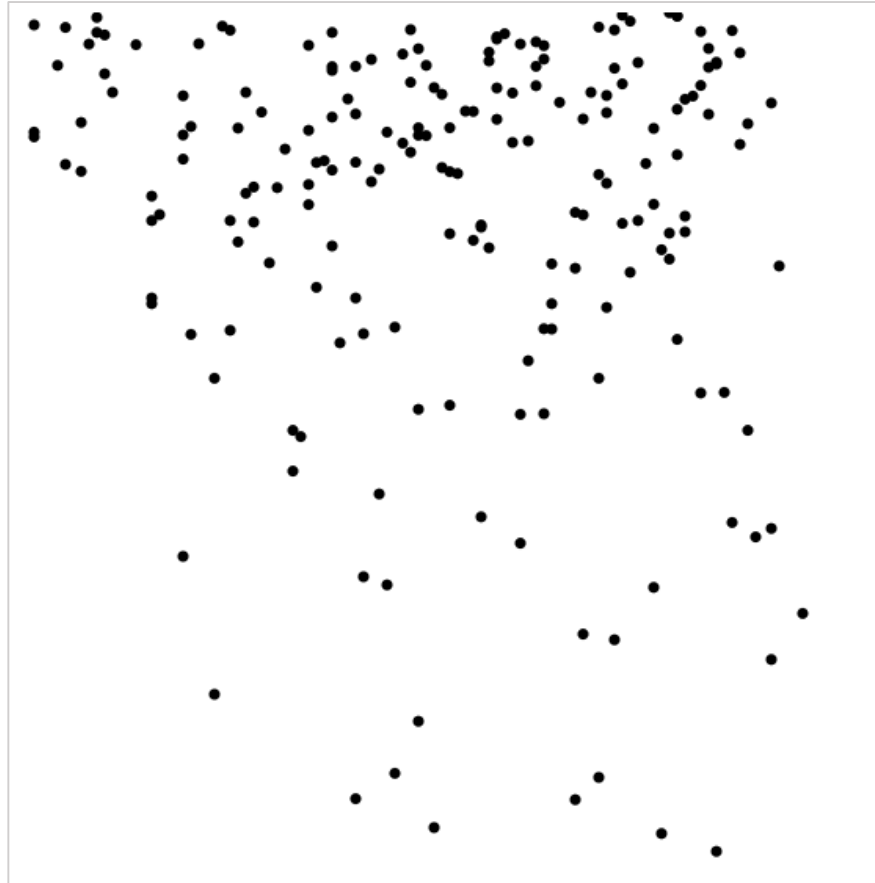
```
{  
    "title": "Beverly Hills Cop II",  
    "genre": "액션",  
    "creative_type": "현대소설",  
    "source": "없음",  
    "release": "80년대",  
    "rating": "전체관람가",  
    "budget": "200",  
    "us_gross": "1536",  
    "worldwide_gross": "2766",  
    "rotten_rating": "46",  
    "imdb_rating": "6.1",  
    "imdb_votes": "29,712"  
},
```

## 점(Point) 찍기

```
let svg = d3.select('svg');  
  
svg  
  .selectAll('circle')  
  .data(data)  
  .enter()  
    .append('circle')  
    .attr('r', 3.5)  
    .attr('cx', d => d.rotten_rating * 5)  
    .attr('cy', d => d.us_gross / 5);
```

- 데이터 배열의 원소 하나하나가 갖고 있는 평점과 수익 값을 대강 살펴본 뒤, 각각 5를 곱하고, 나누어 주는 식으로 점의 좌표를 지정하였다.

## 점(Point) 찍기



### 개선할 수 있는 점

1. 점의 위치를 구할 때 직접 계산해줄 것이 아니라 좀 더 일반적인 방법으로 구할 수 있지 않을까?
2. X, Y 축을 그리고 싶다.
3. 영화 관람 등급에 따라 분포가 어떻게 달라지는지 확인하고 싶다.  
점의 색으로 영화 관람 등급을 표현하자.

### 개선할 수 있는 점

1. 점의 위치를 구할 때 직접 계산해줄 것이 아니라 좀 더 일반적인 방법으로 구할 수 있지 않을까? → `d3.scaleLinear`
2. X, Y 축을 그리고 싶다. → `d3.axisBottom` / `d3.axisLeft`
3. 영화 관람 등급에 따라 분포가 어떻게 달라지는지 확인하고 싶다.  
점의 색으로 영화 관람 등급을 표현하자.  
→ `d3.scaleOrdinal`

## Scale

## • Scale이란?

→ 데이터의 값과 화면 상의 픽셀 값을 연결해주는 함수  
즉, 데이터 상 값을 화면상 위치 (픽셀)에 매핑

us\_gross (억원)

1536  
1300  
1250

정의역 (domain)



Scale

Position (pixel)

260  
190  
180

치역 (range)

## Scale 구하기 &amp; 적용하기

## • X축의 Scale 구하기

X축의 정의역

= data 배열 내의  
rotten\_rating 중 최소값 ~ 최대값

X축의 치역

= X축의 좌표 0 부터  
내가 정한 width까지

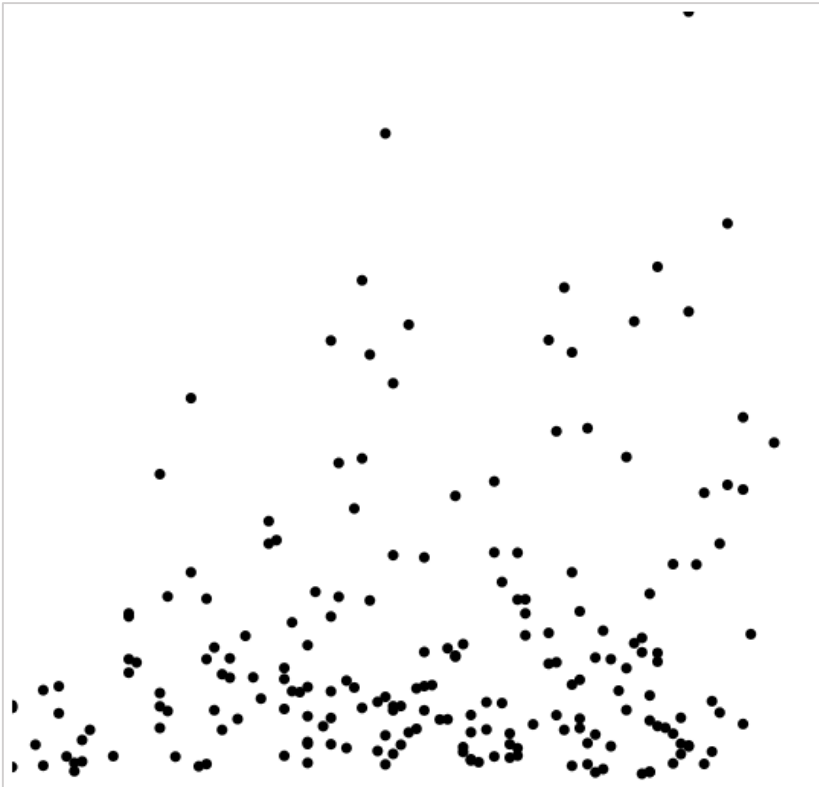
```
let svg = d3.select('svg');  
let width = 500;  
let height = 500;
```

```
let x = d3.scaleLinear()  
    .domain([  
        d3.min(data, d => d.rotten_rating),  
        d3.max(data, d => d.rotten_rating)  
    ])  
    .range([0, width]);
```

```
svg  
    .selectAll('circle')  
    .data(data)  
    .enter()  
    .append('circle')  
    .attr('r', 3.5)  
    .attr('cx', d => x(d.rotten_rating))  
    .attr('cy', d => d.us_gross / 5);
```

## Scale 구하기 &amp; 적용하기

## • 결과



```
let y = d3.scaleLinear()  
    .domain([  
        d3.min(data, d => d.us_gross),  
        d3.max(data, d => d.us_gross)  
    ])  
    .range([height, 0]);  
  
svg  
    .selectAll('circle')  
    .data(data)  
    .enter()  
        .append('circle')  
        .attr('r', 3.5)  
        .attr('cx', d => x(d.rotten_rating))  
        .attr('cy', d => y(d.us_gross));
```

- 여기까지 코드: <https://pastebin.com/5yEmaItV>



## 축(Axis) 그리기

```
let xAxis = d3.axisBottom(x);  
let yAxis = d3.axisLeft(y);  
  
svg  
  .append('g')  
    .attr('transform', 'translate(0, ' + height + ')')  
    .call(xAxis);  
  
svg  
  .append('g')  
    .call(yAxis);
```

- transform 속성을 translate(x, y)으로 지정하면 (x, y) 만큼 평행이동

## 축(Axis) 그리기

- transform 속성에 대해...

```
▶ <g transform="translate(0, 500)" fill="none"
"sans-serif" text-anchor="middle">...</g>
```

- "translate(0, 500)" 이라는 string을 넘긴 것.

```
svg
.append('g')
  .attr('transform', 'translate(0, ' + height + ')')
  .call(xAxis);
```

## 축(Axis) 그리기

- translate는 자주 쓰이므로 함수로 만들어놓고 사용

```
function translate(x, y) {  
  return 'translate(' + x + ', ' + y + ')';  
}
```

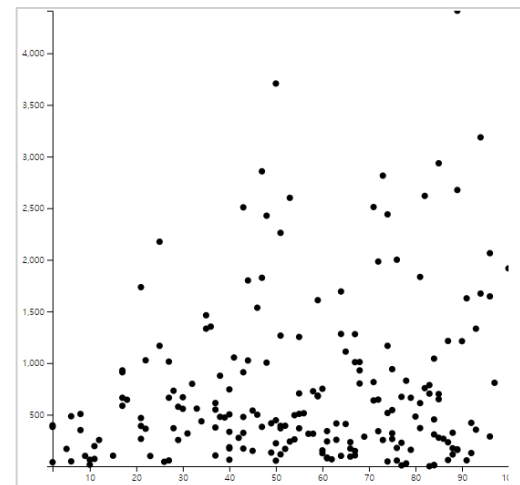
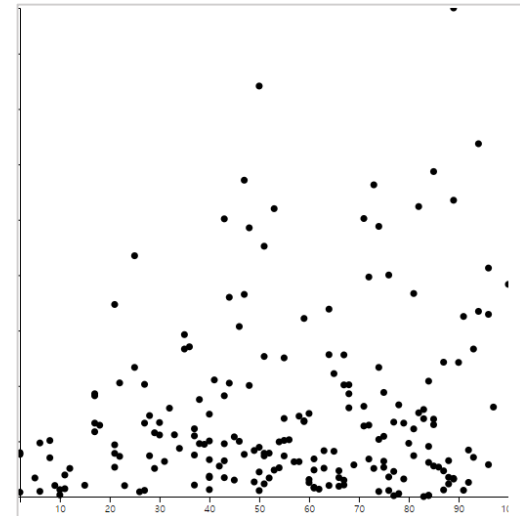
```
svg  
  .append('g')  
    .attr('transform', translate(0, height))  
    .call(xAxis);
```

## Margin 적용하기

- 점들의 치역을 50씩 우측으로
- 축을 우측으로 50 만큼 평행이동

```
let x = d3.scaleLinear()  
  .domain([  
    d3.min(data, d => d.rotten_rating),  
    d3.max(data, d => d.rotten_rating)  
  ])  
  .range([50, width + 50]);
```

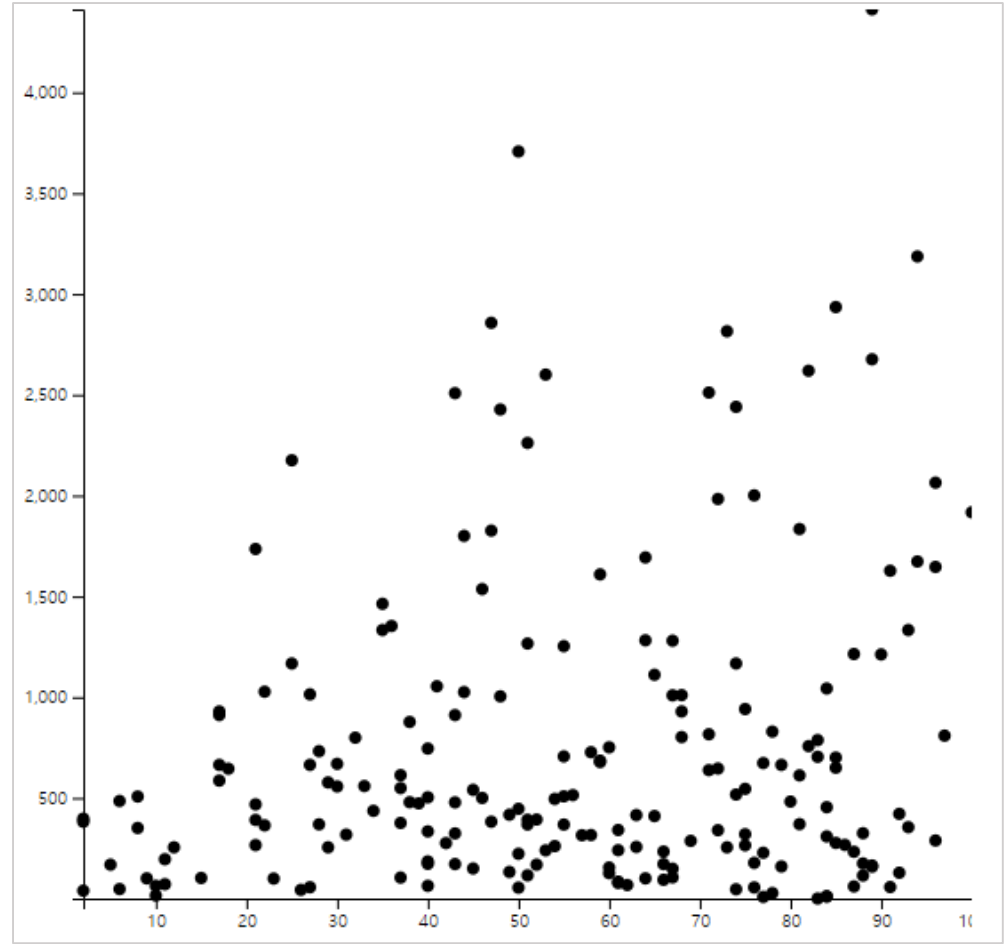
```
svg  
  .append('g')  
  .attr('transform', translate(50, 0))  
  .call(yAxis);
```



## Margin 적용하기

## • 문제점

- 여전히 잘림
- 더욱 일반적인 방법?



## Margin 적용하기

- 새로운 그룹 요소 <g>를 만들고, 거기에 차트를 그리자

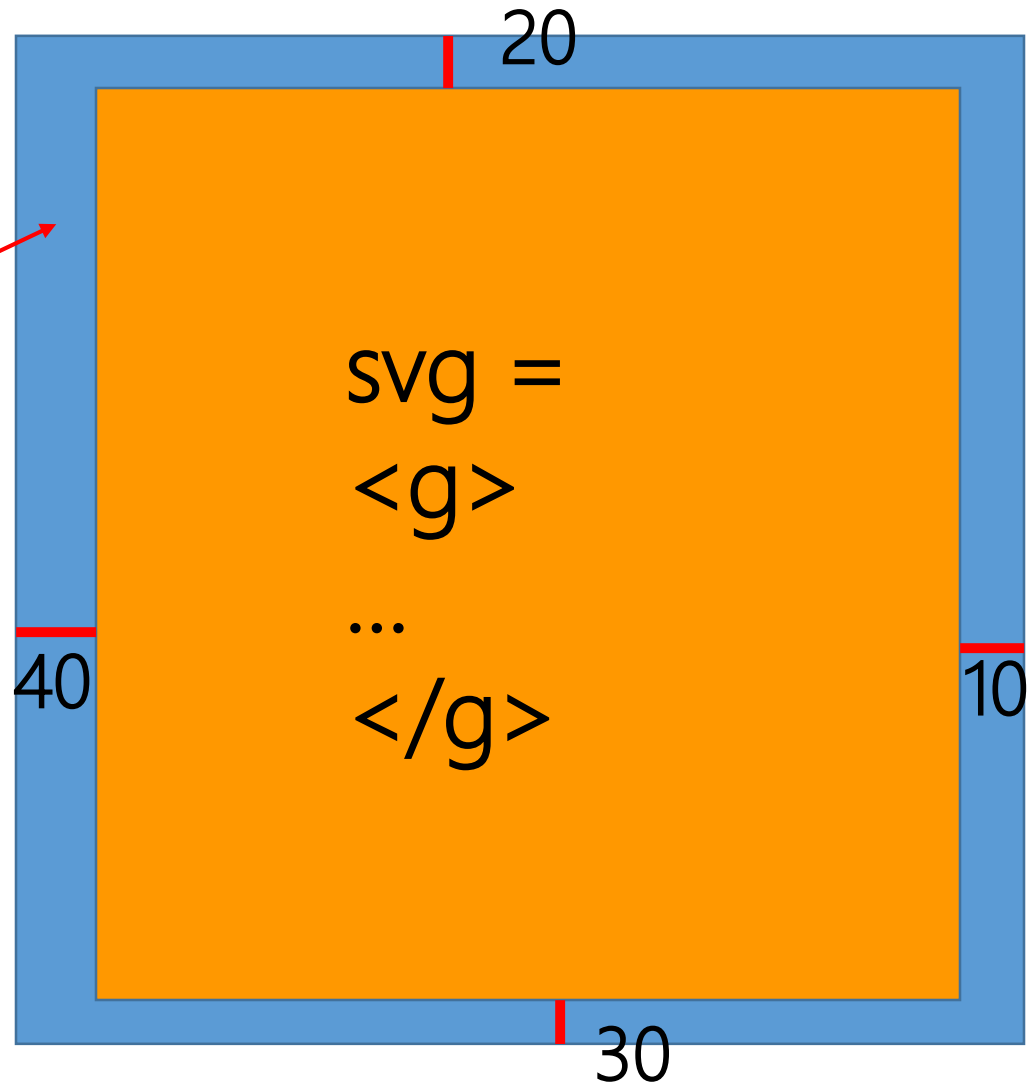
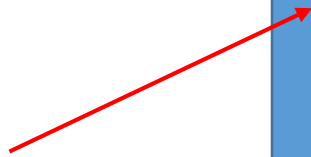
```
<body>  
  <div id="plot"></div>
```

```
let svgWidth = 550;  
let svgHeight = 550;  
let margin = {top: 20, right: 10, bottom: 30, left: 40};  
let width = svgWidth - margin.left - margin.right;  
let height = svgHeight - margin.top - margin.bottom;  
let svg =  
  d3.select('#plot')  
    .append('svg')  
      .attr('width', svgWidth)  
      .attr('height', svgHeight)  
      .append('g')  
        .attr('transform', translate(margin.left, margin.top));
```

## Margin 적용하기

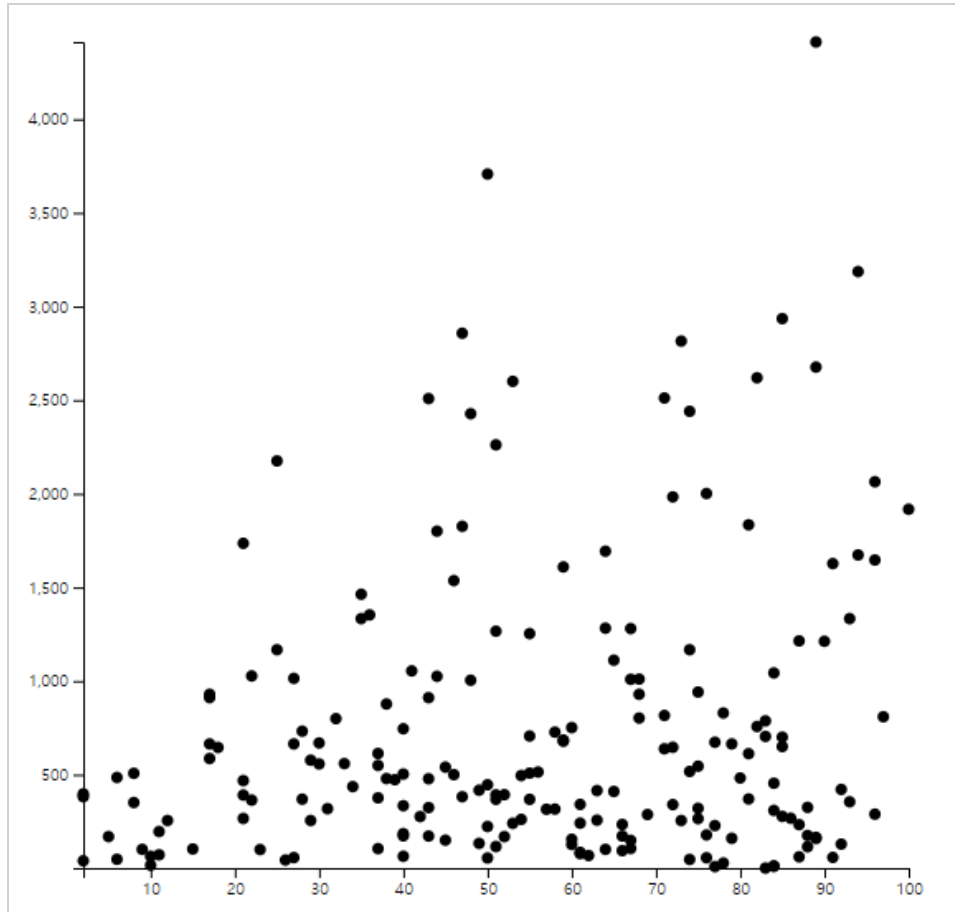
- SVG와 g의 관계

`<svg>`  
`</svg>`



## Margin 적용하기

- 결과 (<https://pastebin.com/PBb2VGHd>)





## Color Scale

- Color Scale?  
→ 데이터의 값과 화면 상의 컬러코드를 연결해주는 함수

등급(rating)

Color (RGB)

전체관람가

#3366cc

7세 이상

#109618

15세 이상

#ff9900

19세 이상

#dc3912



Scale

정의역 (domain)

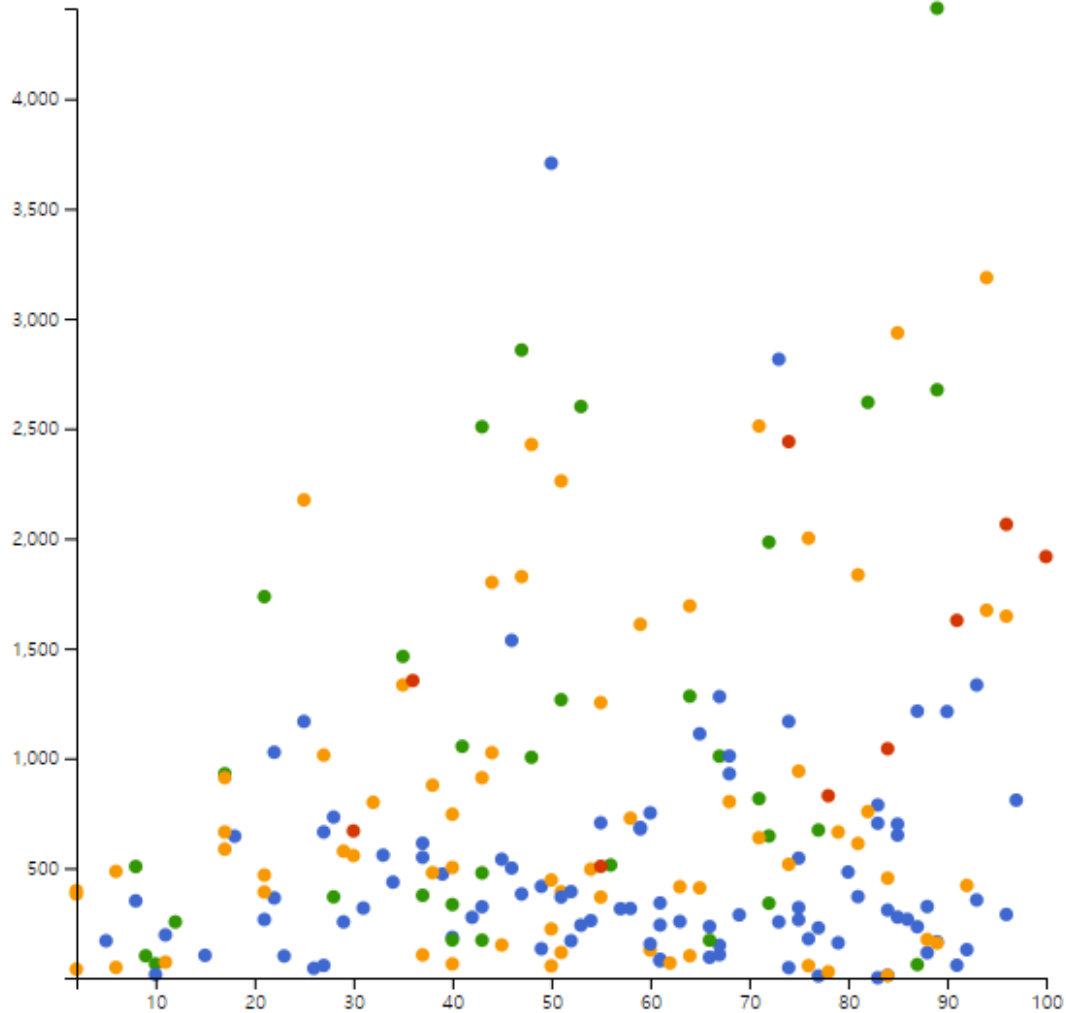
치역 (range)

## Color Scale 구하기 &amp; 적용하기


```
let color =  
d3.scaleOrdinal()  
  .domain(['전체관람가', '7세이상', '15세이상', '19세이상'])  
  .range(['#3366cc', '#109618', '#ff9900', '#dc3912']);  
  
svg  
  .selectAll('circle')  
  .data(data)  
  .enter()  
    .append('circle')  
    .attr('r', 3.5)  
    .attr('cx', d => x(d.rotten_rating))  
    .attr('cy', d => y(d.us_gross))  
    .style('fill', d => color(d.rating));
```

- d3.scaleOrdinal() 을 이용

## 지금까지 그린 산점도



## Event 등록하기

- 뼈대 코드: <https://pastebin.com/ptjcDQ46>
- Event 등록 순서  `d3.selectAll('rect')`
  1. Select 한다.
  2. Selection에 Event를 연결한다

```
d3.selectAll('rect')  
  .on('mouseenter', function() {  
    d3.select(this)  
      .style('fill', 'red');  
  })  
  .on('mouseleave', function() {  
    d3.select(this)  
      .style('fill', 'black');  
  })
```

*selection*  
`.on('이벤트이름', 핸들러)`

## Event의 종류들

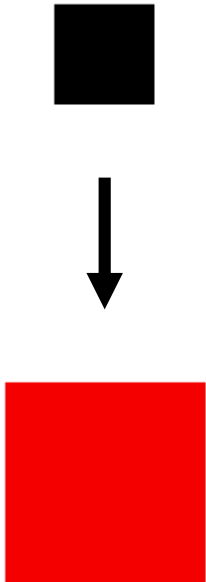
- DOM Event Type
  - [https://developer.mozilla.org/en-US/docs/Web/Events#Standard\\_events](https://developer.mozilla.org/en-US/docs/Web/Events#Standard_events)
- 자주 쓰이는 이벤트

```
selection
  .on('mouseenter', function() { })
  .on('mouseleave', function() { })
  .on('mousemove', function() { })
  .on('click', function() { })
  .on('mouseenter', null)
```

- null 로 지정할 경우 등록된 이벤트가 사라진다.

## Multiple Callbacks

- 한 이벤트에 여러 함수 등록
  - event\_type.name



```
d3.selectAll('rect')
  .on('mouseenter.e1', function() {
    d3.select(this)
      .style('fill', 'red');
  })
  .on('mouseenter.e2', function() {
    d3.select(this)
      .attr('width', 100)
      .attr('height', 100);
  })
  .on('mouseleave.e1', function() {
    d3.select(this)
      .style('fill', 'black');
  })
  .on('mouseleave.e2', function() {
    d3.select(this)
      .attr('width', 50)
      .attr('height', 50);
  });
```

## 주의 사항

- 이벤트 핸들러 내부에서 `this` 키워드는 현재 이벤트가 발생한 요소를 가리킨다.
- 핸들러로 arrow function을 쓰면 `this`의 의미가 다르므로 제대로 동작하지 않는다 (lexical this).

```
selection.on('click', function(d, i) {  
  |   d3.select(this).attr('color', 'red')  
})  
  
selection.on('click', (d, i) => {  
  |   d3.select(this); // wrong  
})  
  
selection.on('click', (d, i, nodes) => {  
  |   d3.select(nodes[i]); // workaround  
})
```

## Transition 사용하기

- Transition을 통해 할 수 있는 것
  - 어떤 속성을
  - 어떤 값(상태)으로
  - 어떻게 바꿀지

간단한 예제 :

<https://bl.ocks.org/d3noob/c3cbb8af554eb848d09ab97306bb5583>

```
var svg = d3.select("body")
  .append("svg")
  .attr("width", 960)
  .attr("height", 500)
  .append("circle")
  .attr("fill", "blue")
  .attr("r", 20)
  .attr('cx', 40)
  .attr('cy', 250)
  .transition()
  .duration(4000)
  .attr('cx', 920);
```



## 실습 – Circle에 간단한 Transition 적용하기

```
<body>
  <svg height='600' width='600'></svg>
  <script src="./lib/d3.v5.min.js"></script>
  <script>

    let svg = d3.select('svg');
    let circle =
      svg
        .append('circle')
        .attr('cx', 50)
        .attr('cy', 50)
        .attr('fill', 'red')
        .attr('r', 10);

  </script>
</body>
```

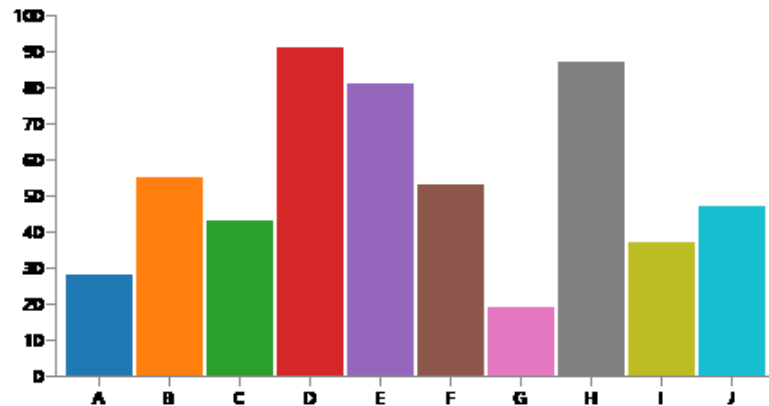
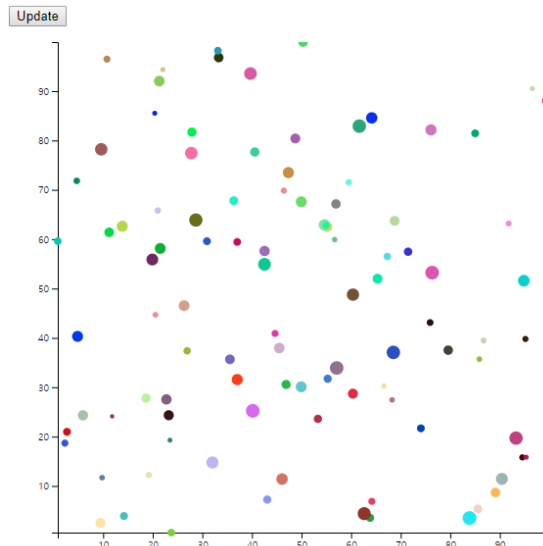
```
circle
  .transition()
  .delay(500)
  .duration(3000)
  .attr('cx', 300)
  .attr('cy', 300)
  .attr('r', 100)
  .attr('fill', 'black');
```

## 오늘 배운 것

- d3 의 조인 (join) 개념
- d3.
  - select, selectAll
- selection.
  - data, enter, exit, attr, style, select, selectAll, on, transition
- 축, 마진, 이벤트, 트랜지션...

## 과제 개요

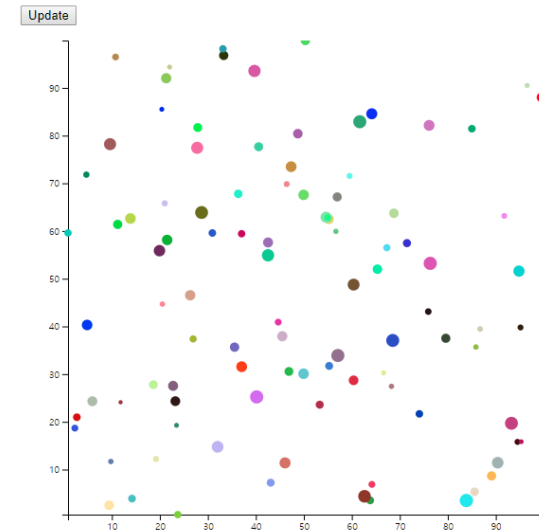
- 바 차트와 산점도를 그려 봅시다.
- 숙제 의도: Javascript를 써보고 d3.js를 사용하기 위한 전반적인 환경을 연습하기



## 산점도 (Scatterplot)

## • 필수 스펙

- 100개의 점
- $cx$ ,  $cy$  (원의 중심 좌표),  $r$  (원의 반지름),  
원의 색깔 rgb코드 (0~255)는  
페이지 상단의 update 버튼을 누를 때 마다  
랜덤으로 변해야 함
- $x$ ,  $y$ 축을 반드시 포함해야 함
- Update 버튼을 누를 때 마다 **각각의 점 및 축**이 모두 *transition* 되면서  
변해야 함
- 점 위에 마우스를 올릴 경우 *visual feedback*을 제공해야 됨
  - 마우스를 뺄 경우 **원래 상태로** 돌아가야 함
  - `on('mouseover', ...)` `on('mouseout', ...)`, `css: hover...`



## 바 차트

## • 필수 스펙

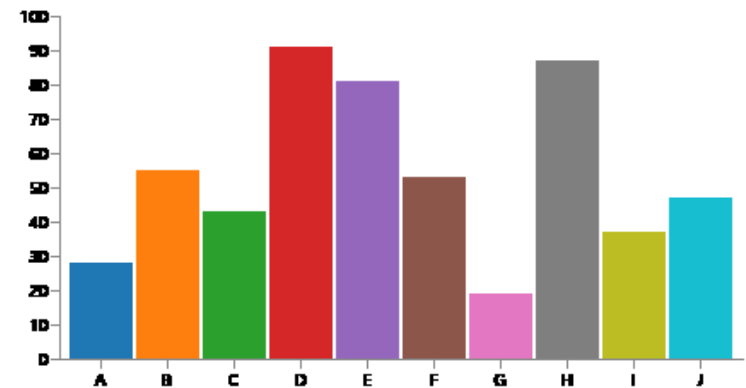
- 10개의 범주(A~ J)에 대해 0 ~ 100 사이의 랜덤 값을 가지는 세로 바 차트
- X축 및 y축 (0 ~ 100)을 반드시 포함할 것
- 막대의 색깔은 서로 다른 10개의 색상을 지정
- Update를 클릭할 경우 Bar의 높이가 변해야 함

## • 나머지 스펙은 자유롭게 정할 것

- 조교가 채점할 수 있는 한에서

## • 힌트

- Math.random
- D3.scaleOrdinal(d3.schemeCategory10)



## 제출 방법

- 두 차트를 한 HTML 페이지에 구현 후 html 페이지를 eTL을 통해 제출
  - 반드시 두 차트를 한 페이지에 구현할 것
  - 제출은 **하나의** html 페이지만 함
  - 독립된 js 파일 포함시키지 말 것 (d3.js도 웹에서 불러 오도록)
  - 압축하지 말것
- 채점은 최신 Chrome 브라우저에서 합니다.
- 스펙 관련 문의는 eTL의 질문답변 게시판을 이용
- **제출 기간: ~10월 16일 (수) 23:59**