Information Visualization

D3.js - 2

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

Modified by Dantae An (dtan@hcil.snu.ac.kr)

Human-Computer Interaction Laboratory
Seoul National University

hci lab
HUMAN × COMPUTER INTERACTION

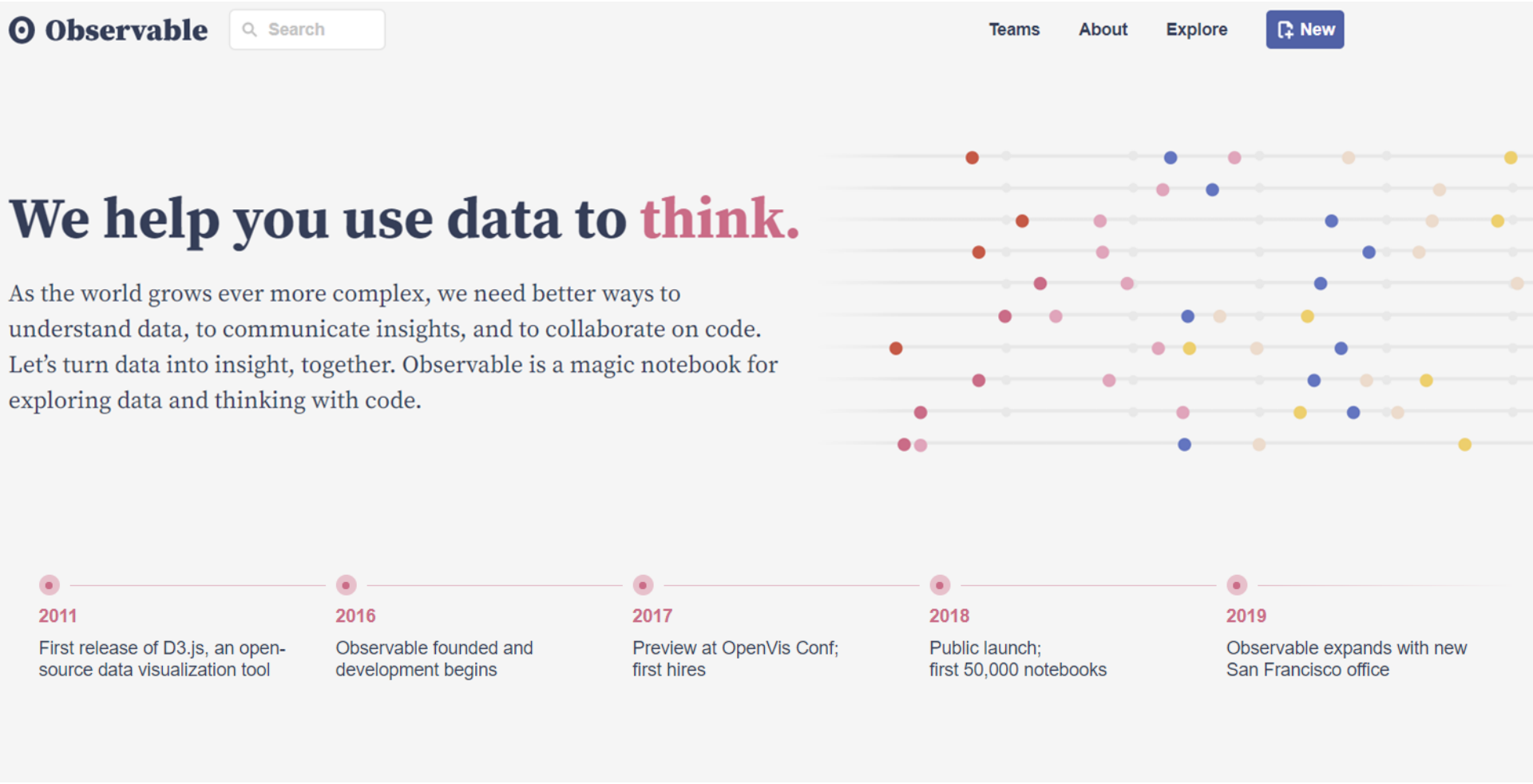
D3.js

Observable

D3 학습의 어려운 점

- 복잡한 시각화를 만들려면 그만큼 복잡한 데이터가 필요
- D3의 모듈 별로 원하는 standard data structure을 파악하고,
  그에 맞는 sample data를 준비하고,
  실제로 시각화를 건드려 보기까지 진입장벽이 높음
- Documentation이 그다지 친절하지 않음
- 여러 버전의 코드가 혼재함

Observable


Observable
Search
Teams
About
Explore
New
We help you use data to think.
As the world grows ever more complex, we need better ways to understand data, to communicate insights, and to collaborate on code. Let's turn data into insight, together. Observable is a magic notebook for exploring data and thinking with code.
2011
First release of D3.js, an open-source data visualization tool
2016
Observable founded and development begins
2017
Preview at OpenVis Conf; first hires
2018
Public launch; first 50,000 notebooks
2019
Observable expands with new San Francisco office

Observable

- Observable
  - https://observablehq.com/
  - JavaScript 기반 노트북
  - 모든 코드가 웹 브라우저에서 실행됨
  - 시각화를 탐색, 실험, 공유하기 편리함
- Observable/@d3
  - Observable 노트북에서 D3 라이브러리를 가져다 쓸 수 있음
  - 풍부한 예제와 설명 제공
  - 최신 버전(6)의 코드 제공
  - https://observablehq.com/@d3
  - https://observablehq.com/@d3/gallery

Observable

- Observable 노트북의 특징
  - https://observablehq.com/@observablehq/observable-for-jupyter-users
  - JS 뿐만 아니라 html, markdown 도 가능
  - 셀은 단 하나의 output을 가지며, 셀 위에 표시된다.
  - 코드를 숨길 수도 있다.
  - 셀 당 하나씩 notebook-level global variable을 선언할 수 있다.
    - 전역변수는 다른 셀에서 참조할 수 있다.
    - 전역변수의 값이 바뀌면 그것을 참조하는 모든 셀이 업데이트 된다.
    - 전역변수 간의 dependency를 분석해서 필요한 셀을 모두 재실행해주므로 **cell의 순서는 중요하지 않다.**
    - 전역변수는 다른 노트북에서 import해올 수 있다.

Observable

Observable Search Explore Demo Fork Sign in

D3 d3js.org
Bring your data to life.
By Fil

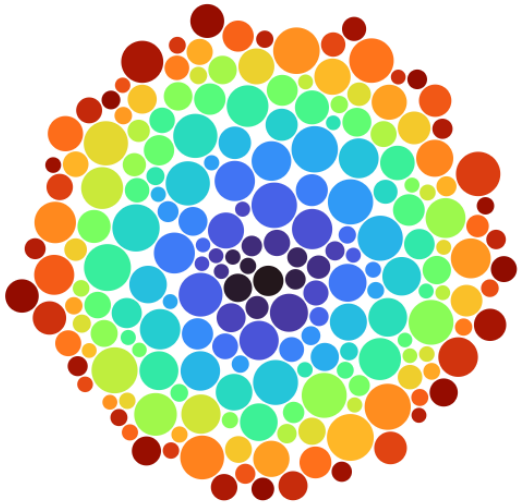
21

Published May 6 ISC 1 Fork Listed in d3-force

Force layout phyllotaxis arrangement

An easily overlooked feature of d3-force is that "If either x or y is NaN, the position is initialized in a phyllotaxis arrangement"; start your nodes without {x,y} and the initial layout is set in a deterministic (and quite pleasing) way.

Replay



```
chart = {
  // the default phyllotaxis arrangement is centered on <0,0> with a distance between nodes of ~10 pixels
  // we will scale & translate it to fit the canvas
  const scale = 1.7;
```

```
  // differ application of the forces
  setTimeout(() => {
    simulation.restart();
    node.transition().attr("r", d => d.r);
  }, 2000);

  // once the arrangement is initialized, scale and translate it
  if (rescale) {
    for (const node of nodes) {
      node.x = node.x * scale + center[0];
      node.y = node.y * scale + center[1];
    }
  }

  function tick() {
    node.attr("cx", d => d.x).attr("cy", d => d.y);
  }

  // show the initial arrangement
  tick();

  return svg.node();
}
```

```
height = 600
```

```
height = 600
```

```
n = 200
```

```
n = 200
```

```
nodes = ▶ Array(200) [Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object
```

```
nodes = replay, Array.from({length: n}, (_, i) => ({
  r: 2 * (4 + 9 * Math.random() ** 2),
  color: color(i)
}))
```

```
color = f(n)
```

```
color = d3.scaleSequential(d3.interpolateTurbo).domain([0, n])
```

```
d3 = ▶ Object {format: f(t), formatPrefix: f(t, n), timeFormat: f(t), timeParse: f(t), utcFormat: f(t), utcParse: f(t), Adde
```

```
d3 = require("d3@6")
```

Load Files

d3-fetch

- 파일을 불러오는 D3 내장 함수들
  - `d3.csv` , `d3.tsv` , `d3.json` , `d3.text` , ...
- 함수에 parsing 기능이 포함되어 있음

```
a,b,c
1,2,3
4,5,6
```

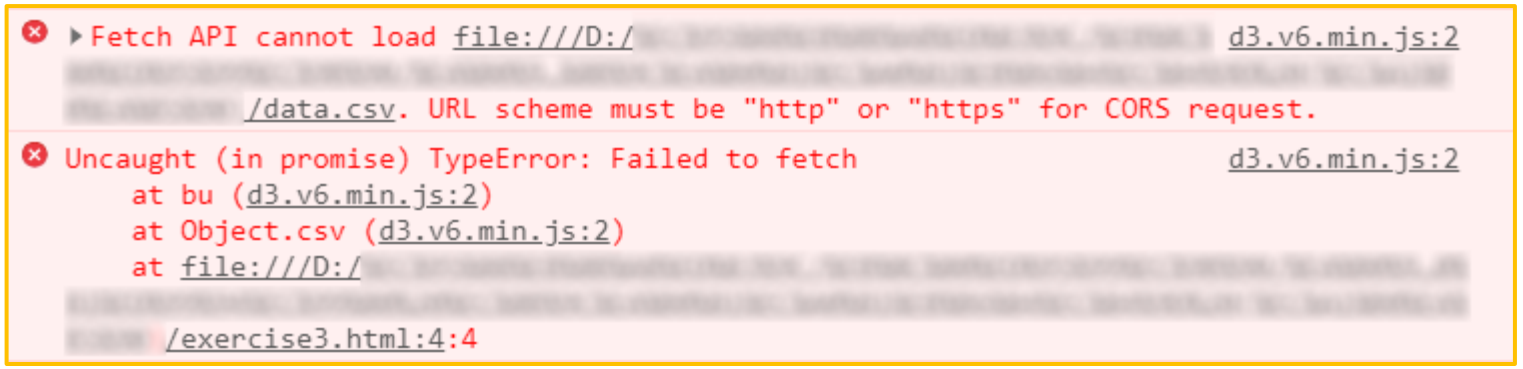
```
[
  {"a":"1", "b":"2", "c":"3"},
  {"a":"4", "b":"5", "c":"6"}
]
```

- 사용 방법
  - `d3.csv(filename).then(function(data) { /* 코드 작성 */ });`
  - *filename*에 원하는 파일의 디렉토리를 적고, callback으로 *data*를 받음
  - 예)

```
d3.csv('data.csv').then(function(data) {
  // data에 data.csv의 데이터가 json 형식으로 들어가 있음
  console.log(data);
});
```

유의사항 1 – 로컬 데이터

- Chrome에서는 코드가 local 데이터에 접근하는 것을 막음



- 해결방법
  1. 서버 이용
  2. .js 또는 .html 파일에 원하는 데이터를 통째로 넣음 (추천하지 않음)

```
let data = [
  ...
]
```

서버 열기

- 방법 1 – 1: npm live-server 이용
  1. 터미널 또는 명령 프롬프트(cmd) 실행
  2. 실행하려는 .html 파일(exercise3.html)이 있는 프로젝트 폴더로 이동

     ```
     cd "D:/infovis/exercise3"
     ```

  3. 서버 열기

     ```
     npx live-server --open="./exercise3.html"
     ```

     - 서버가 열려 있는 동안 브라우저에서 다음 주소로 접속 가능

       `http://localhost:8080/exercise3.html`

     - 서버를 돌리는 중에 **코드를 변경**하면 브라우저에서도 **자동으로 새로고침**된다.
     - 서버를 종료하려면 터미널에서 Ctrl + c 키 입력
     - 이 기능은 개발 용도로만 사용하고,
       외부에서 접속 가능한 웹 서버에는 사용하면 안 된다.
     - https://www.npmjs.com/package/live-server

서버 열기

- 방법 1 – 2: python3 http.server 이용
  1. 터미널 또는 명령 프롬프트(cmd) 실행
  2. 실행하려는 .html 파일(exercise3.html)이 있는 프로젝트 폴더로 이동

     ```
     cd "D:/infovis/exercise3"
     ```

  3. 서버 열기

     ```
     python -m http.server
     ```

     - 서버가 열려 있는 동안 브라우저에서 다음 주소로 접속 가능

       `http://localhost:8000/exercise3.html`

     - 서버를 돌리는 중에 **코드를 변경**하면 브라우저에서 **수동으로 새로고침**해야 한다.
     - 서버를 종료하려면 터미널에서 Ctrl + c 키 입력
     - 이 기능은 개발 용도로만 사용하고,
       외부에서 접속 가능한 웹 서버에는 사용하면 안 된다.
     - https://docs.python.org/ko/3/library/http.server.html

유의사항 2 – 데이터 Type

- csv, tsv 등과 같은 경우 데이터는 string이 기본값이다.
  - 처리하지 않으면 의도치 않은 결과가 나올 수도...

```
// data.csv
// a,b,c
// 1,2,3
// 4,5,6

d3.csv('data.csv').then(function(data) {
  let sum_of_a = data.map(k => k['a'])
                     .reduce((acc, cur) => acc + cur, 0);
  console.log(sum_of_a);
});
// 0 + 1 + 4 = 5가 나오면 좋겠지만,
// 실제로는 "0" + "1" + "4" = "014"가 나옴
```

유의사항 2 – 데이터 Type

- csv, tsv 등과 같은 경우 데이터는 string이 기본값이다.
  - 처리하지 않으면 의도치 않은 결과가 나올 수도...

```
// data.csv
// a,b,c
// 1,2,3
// 4,5,6

d3.csv('data.csv').then(function(data) {
  let sum_of_a = data.map(k => k['a'])
                     .reduce((acc, cur) => acc + cur, 0);
  console.log(sum_of_a);
});
// 0 + 1 + 4 = 5가 나오면 좋겠지만,
// 실제로는 "0" + "1" + "4" = "014"가 나옴
```

`array.reduce((acc, cur) => acc + cur, 0)`

배열 *array*의 각 값을 cur가 순회하면서,
이 값들을 acc에 누적(+)하여 최종 acc 값을 반환한다.
acc의 초기값은 0이다. (공식 문서 참조)

유의사항 3 – 버전

- 버전 4 이하와 버전 5 이상의 작동 방식이 다르다.
- https://github.com/d3/d3/blob/master/CHANGES.md#changes-in-d3-50

For example, to load a CSV file in v4, you might say:

```
d3.csv("file.csv", function(error, data) {
  if (error) throw error;
  console.log(data);
});
```

In v5, using promises:

```
d3.csv("file.csv").then(function(data) {
  console.log(data);
});
```

Note that you don't need to rethrow the error—the promise will reject automatically, and you can *promise*.catch if desired. Using await, the code is even simpler:

```
const data = await d3.csv("file.csv");
console.log(data);
```

D3.js

Selection

Selector

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

- Selector에는 DOM object 뿐만 아니라 W3C selector string도 가능함

```
d3.selectAll('rect')     // rect인 DOM들을 모두 선택
d3.selectAll('#rect')    // id가 'rect'인 DOM들을 모두 선택
d3.selectAll('.rect')    // class가 'rect'인 DOM들을 모두 선택
```

Selection Chaining

- `select()` , `selectAll()` 도 chaining이 가능함
  - *selection* 내에서 *selector*에 해당되는 element들이 포함된 *selection*을 반환함

```
d3.select('#table')       // 전체에서 id가 'table'인 DOM을 하나 선택
  .selectAll('.selected') // 그 안에서 class가 'selected'인 DOM 선택
```

- 비효율적인 selection 비용을 줄일 수 있음
  - 전체 html을 찾는 대신, 보다 좁은 영역에서 찾음

selection.merge()

- **`selection.merge(other)`**
  - 현재 *selection*과 *other*(selection)을 합친 새로운 selection을 반환함
- enter와 update selection을 한꺼번에 묶어서 처리 가능

```
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);

/* enter & update selection 처리 */
bars.enter()
    .append('rect')
    .merge(bars)
    .attr('width', 30)
    .attr('height', d => d.value * 10)
    .attr('transform', (d, i) => `translate(${i * 40}, 0)`);

/* exit selection 처리 */
bars.exit().remove();
```

enter selection

enter selection + **bars** (update selection)

selection.join()

- `selection.join(name)`
  - `selection.data()` 에 의해 bind된 데이터에 알맞게 추가, 업데이트, 제거
  - 통상적으로 하는 enter, update, exit을 전부 한번에 수행
  - String(태그 유형)**을 인자**로 넘길 경우
    - 그 태그를 enter하고 append한다.
    - exit에 대해서는 remove한다.
    - 처리 후에 enter과 update의 merge selection을 반환한다.

```
function randomLetters() {
  return d3.shuffle("abcdefghijklmnopqrstuvwxyz".split(""))
    .slice(0, Math.floor(6 + Math.random() * 20))
    .sort();
}

const svg = d3.select("svg")
    .attr("width", 500)
    .attr("height", 33)
    .attr("viewBox", `0 -20 500 33`);
```

```
// string을 join()의 인자로 넘길 때

svg.selectAll("text")
  .data(randomLetters())
  .join("text")
    .attr("x", (d, i) => i * 16)
    .text(d => d);
```

b e f l n p r t w z

selection.join()

- `selection.join(enter[, update][, exit])`
  - `selection.data()` 에 의해 bind된 데이터에 알맞게 추가, 업데이트, 제거
  - 통상적으로 하는 enter, update, exit을 전부 한번에 수행
  - **함수**(*enter, update, exit*)**를 인자**로 넘길 경우
    - 첫 번째 인자는 enter selection, 두 번째 인자는 update selection,
      세 번째 인자는 exit selection을 처리하는 함수이어야 한다.
      - 함수를 꼭 3개 다 넘길 필요는 없다.
    - 각각의 selection을 처리하는 함수를 자유롭게 작성할 수 있다.
      - 각 함수는 하나의 인자 selection을 받아서 **selection을 반환**해야 한다.
      - 함수 안에서 transition을 사용해야 할 경우, 반환 값이 selection이 되도록 하기 위해서 `call()` 을 호출하고 이 안에서 transition을 사용해야 한다.
        (다음 슬라이드의 코드 및 슬라이드 25 참조)
    - 처리 후에 enter과 update의 merge selection을 반환한다.
  - https://observablehq.com/@d3/selection-join

selection.join()

```
function randomLetters() {
  return d3.shuffle(
      "abcdefghijklmnopqrstuvwxyz".split(""))
    .slice(0,
      Math.floor(6 + Math.random() * 20))
    .sort();
}

const svg = d3.select("svg")
    .attr("width", 500)
    .attr("height", 33)
    .attr("viewBox", `0 -20 500 33`);
```

```
function sleep(ms) {
  return new Promise(resolve =>
    setTimeout(resolve, ms));
}

async function showText() {
  while (true) {
    // 여러 transition 동기화
    const t = svg.transition()
        .duration(750);
```

```
    // (showText() 함수 이어서 구현)
    // 함수를 join()의 인자로 넘길 때
    svg.selectAll("text")
      .data(randomLetters(), d => d)
      .join(
        enter => enter.append("text")
            .attr("fill", "green")
            .attr("x", (d, i) => i * 16)
            .attr("y", -30)
            .text(d => d)
          .call(enter => enter.transition(t)
            .attr("y", 0)),
        update => update
            .attr("fill", "black")
            .attr("y", 0)
          .call(update => update.transition(t)
            .attr("x", (d, i) => i * 16)),
        exit => exit
            .attr("fill", "brown")
          .call(exit => exit.transition(t)
            .attr("y", 30)
            .remove())
      );
    await sleep(2000);
  }
}
showText();
```

join() 안의 함수에서 transition()을 호출할 때에는 반드시 call()을 사용해야 함에 유의!

selection.classed()

- `selection.classed(names[, value])`
  - selection들의 class를 효과적으로 관리하기 위한 함수
    - e.g. class가 foo인 DOM들에 새로운 class인 bar를 추가하고 싶음
  - *value*가 truthy일 경우 *names* class를 *selection*에 모두 추가함
  - *value*가 false일 경우 *names* class를 *selection*에서 모두 제거함
  - *value*가 함수일 경우 각 *selection*에 대하여 함수의 반환 값(Boolean)에 따라...
  - *value*가 없을 경우 null이 아닌 첫 번째 *selection*에 *names* class가 할당되어 있는지 확인 후 true/false를 반환함 (*selection*이 하나일 때 사용)

```
// 위 코드와 아래의 코드가 동일한 기능
d3.selectAll('.foo')
  .attr('class', function(k) {
    return d3.select(this).attr('class') + ' bar';
  });

d3.selectAll('.foo').classed('bar', true);
```

```
selection.classed("foo", function() { return Math.random() > 0.5; });
```

selection.each()

- `selection.each(function)`
  - 각 *selection*에 대하여 *function*을 iterate함

```
d3.selectAll('.foo').each(function(d, i, nodes) {
  // d      현재 요소에 bind(join)된 데이터
  // i      현재 요소의 selection 상의 인덱스
  // nodes  selection이 가지고 있는 요소들의 배열

  // 인자로 함수를 넘길 때의 parameter들은
  // 위와 같이 모든 d3 함수에서 동일함!
});
```

- parent DOM의 데이터와 child DOM의 데이터를 모두 접근해야 할 때 유용함

```
parent.each(function(p, j) {
  d3.select(this)
    .selectAll('.child')
      .text(function(d, i) { return "child " + d.name + " of " + p.name; });
});
```

selection.call()

- `selection.call(function[, arguments])`
  - *selection*과 *arguments*를 인자로 하는 함수를 1회 실행하고 *selection*을 반환함
  - 아래의 두 코드는 같은 기능

```
d3.selectAll("div").call(name, "John", "Snow");
```

```
name(d3.selectAll("div"), "John", "Snow");
```

- D3에서는 axis, brush(슬라이드 39 참조) 등을 그릴 때 활용함

```
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);
```

selection.property() / *selection*.text() / *selection*.html()

- `selection.property(name[, value])`
  - `selection.attr()` 이나 `selection.style()` 로 접근이 불가능한 property(예: checkbox의 checked)들을 접근할 때 활용함
- `selection.text([value])`
  - *value*가 정의되지 않은 경우, *selection*의 inner text 내용을 반환함
  - *value*가 정의될 경우, DOM의 inner text 내용을 *value*로 설정함
  - value가 함수일 경우, 각 *selection*에 대하여 inner text 내용을 반환 값으로 설정 (함수의 인자는 `d, i, nodes`)
- `selection.html([value])`
  - `selection.text()` 에서 inner text 대신 inner html

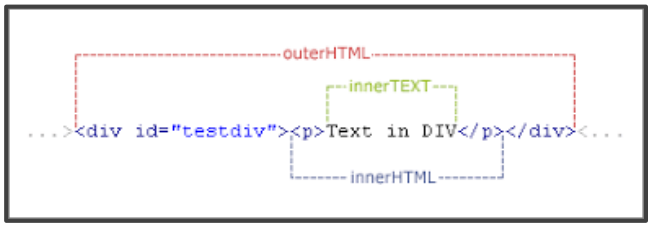


Image from https://blog.csdn.net/tjcyjd/article/details/6680819

Data Binding

Table 그리기

- 그동안 배운 내용들을 활용하여 D3로 table을 만들어보자.

```
let myData = [{a:1, b:2, c:3},
              {a:4, b:5, c:6}];
```

```
<table id="table">
  <thead>
    <tr>
      <td>a</td>
      <td>b</td>
      <td>c</td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>1</td>
      <td>2</td>
      <td>3</td>
    </tr>
    <tr>
      <td>4</td>
      <td>5</td>
      <td>6</td>
    </tr>
  </tbody>
</table>
```

a b c

1 2 3

4 5 6

`<table>` : 표
`<thead>` : 표의 헤더 그룹화
`<tbody>` : 표의 본문 그룹화
`<tr>` : 표의 행(row) 요소
`<td>` : 표의 열(column) 요소

a	b	c
1	2	3
4	5	6

Table 그리기

- 그동안 배운 내용들을 활용하여 D3로 table을 만들어보자.

```
let myData = [{a:1, b:2, c:3}, {a:4, b:5, c:6}];
let keys = Object.keys(myData[0]);                    // [a, b, c]

let table = d3.select('body').append('table').attr('id', 'table');
let thead = table.append('thead').append('tr')
  .selectAll('td').data(keys).join('td')
  .text(k => k);

let tbody = table.append('tbody');
let trs = tbody.selectAll('tr').data(myData).join('tr');
```

- 각 `<tr>` 에 `<td>` 는 어떻게 붙이지?

a	b	c
{a:1, b:2, c:3}		
{a:4, b:5, c:6}		

a b c

Binding 여러 번 하기

- 우리는 `<tr>` 에 각각의 데이터 오브젝트를 bind하고 싶을 뿐만 아니라, 각 `<td>` 에 a, b, c에 해당하는 값을 bind하고 싶음
- D3에서는 **미리 binding된 데이터를 활용**하여 binding이 가능함
  - `selection.data()` 의 첫 번째 인자로 **함수**를 넘기면 됨

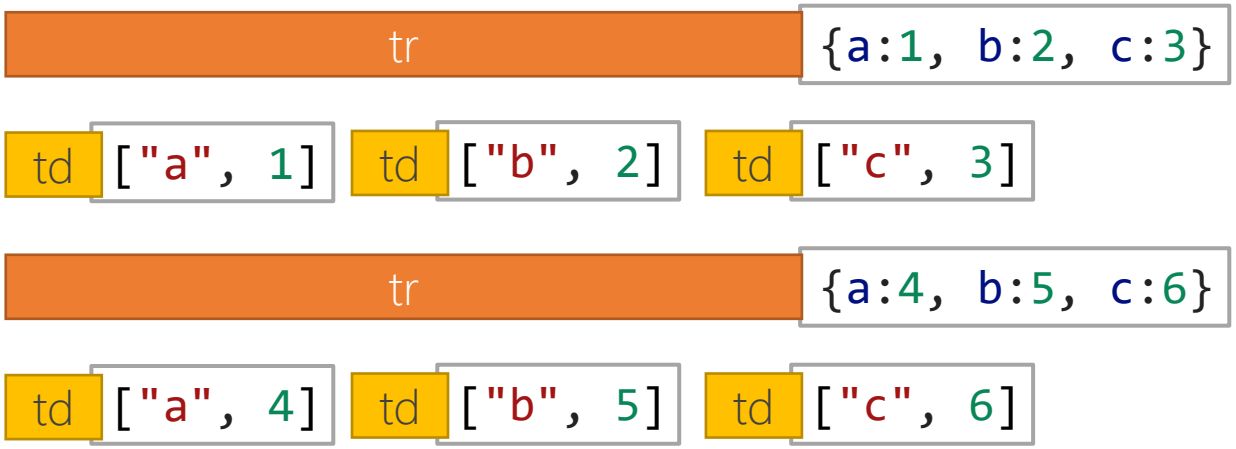
```
let tbody = table.append('tbody');
let trs = tbody.selectAll('tr').data(myData).join('tr');

trs.selectAll('td')
  .data(k => Object.entries(k))
  .join('td').text(d => d[1]);
```

k로 들어가는 값은
각 <tr>에 binding되어 있던 데이터인
{a:1, b:2, c:3} 와
{a:4, b:5, c:6} 이다.

Object.entries({a:1, b:2, c:3})
를 수행하면 다음을 반환한다.
[["a", 1], ["b", 2], ["c", 3]]

Binding 여러 번 하기

```
let tbody = table.append('tbody');
let trs = tbody.selectAll('tr').data(myData).join('tr');

trs.selectAll('td')
   .data(k => Object.entries(k))
   .join('td').text(d => d[1]);
```



1 2 3

4 5 6

Binding 여러 번 하기

- 아래의 코드들은 모두 같은 기능을 함 (맨 위가 앞 슬라이드의 코드)

```
trs.selectAll('td')
  .data(k => Object.entries(k))
  .join('td').text(d => d[1]);
```

```
trs.each(function(k) {
  d3.select(this)
    .selectAll('td')
    .data(Object.entries(k))
    .join('td').text(d => d[1])
});
```

```
trs.selectAll('td')
  .each((k, i, nodes) => {
    d3.select(nodes[i]).data(Object.entries(k))
      .join('td').text(d => d[1])
  });
```

"lexical this"

보다 제대로 코딩하기

```
let myData = [{a:1, b:2, c:3}, {a:4, b:5, c:6}];
let keys = Object.keys(myData[0]);                    // [a, b, c]

let table = d3.select('body').append('table').attr('id', 'table');
let thead = table.append('thead').append('tr')
  .selectAll('td').data(keys).join('td')
  .text(k => k);

let tbody = table.append('tbody');
let trs = tbody.selectAll('tr').data(myData).join('tr');

trs.selectAll('td')
//.data(k => Object.entries(k))
  .data(k => keys.map(eachKey => [eachKey, k[eachKey]]), tuple => tuple[0])
  .join('td').text(d => d[1]);
```

- 주석 친 코드를 바로 아래의 노란색 코드로 바꾼 이유는 무엇일까?

보다 제대로 코딩하기

```
let myData = [{a:1, b:2, c:3}, {a:4, b:5, c:6}];
let keys = Object.keys(myData[0]);                    // [a, b, c]

let table = d3.select('body').append('table').attr('id', 'table');
let thead = table.append('thead').append('tr')
  .selectAll('td').data(keys).join('td')
  .text(k => k);

let tbody = table.append('tbody');
let trs = tbody.selectAll('tr').data(myData).join('tr');

trs.selectAll('td')
  .data(k => keys.map(eachKey => [eachKey, k[eachKey]]), tuple => tuple[0])
  .join('td').text(d => d[1]);
```

1. 표 헤더(제목)와의 *key* **순서**를 맞춰줘야 함
   - `keys.map()` 을 통하여 object의 [*key*, *value*]를 `keys` 의 순서대로 가져옴
2. 인덱스가 아닌, object의 ***key* 값으로** 각 `<td>` 에 join을 시켜줘야 함
   - [*key*, *value*] 값의 *key*를 join의 key로 지정 (두 번째 실습 PPT 슬라이드 31 참조)

Table 대신 SVG 사용!

- 표 시각화를 그릴 때에는 `<table>`을 사용하면 안 된다.
  - `<td>`, `<tr>`에는 다음 속성들이 적용되지 않는다.
    - `selection.attr('transform', ...)`
    - `selection.attr('font-weight', ...)`
    - `selection.style('fill', ...)`
    - 기타 등등...

- 표는 SVG의 `<g>`, `<text>`를 이용하여 그리자.

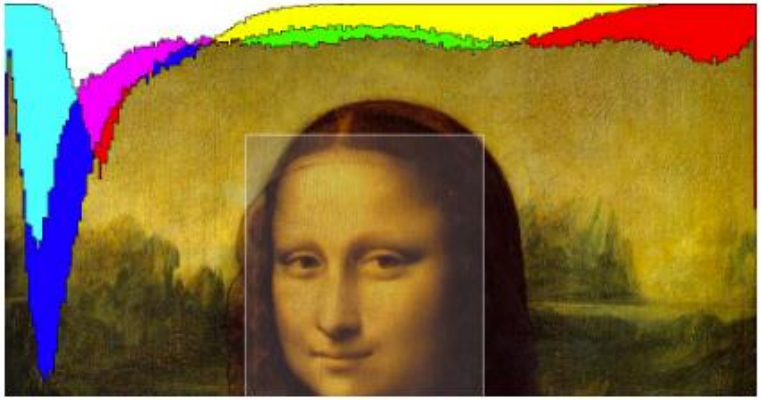
Brushing

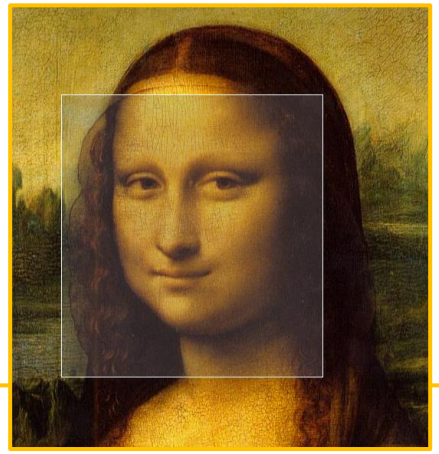
Brushing이란?

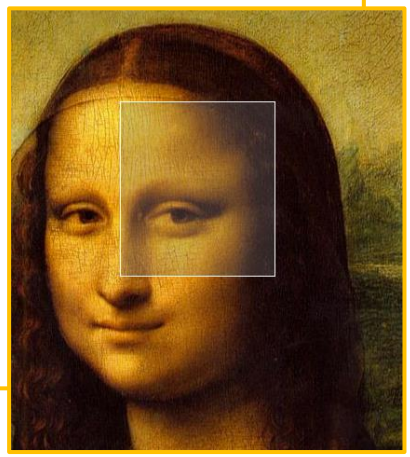
- 마우스를 활용하여 Interactive하게 시각화의 영역을 선택하도록 해주는 기능

d3-brush

Brushing is the interactive specification a one- or two-dimensional selected region using a pointing gesture, such as by clicking and dragging the mouse. Brushing is often used to select discrete elements, such as dots in a scatterplot or files on a desktop. It can also be used to zoom-in to a region of interest, or to select continuous regions for cross-filtering data or live histograms:

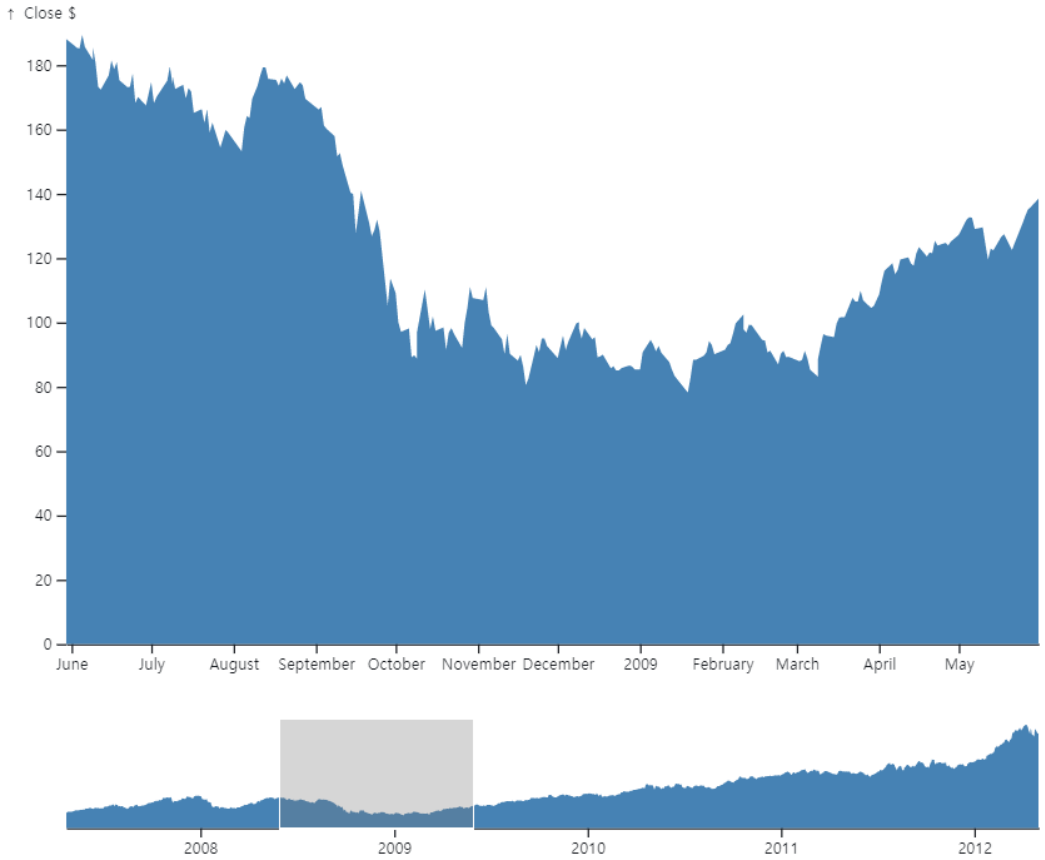






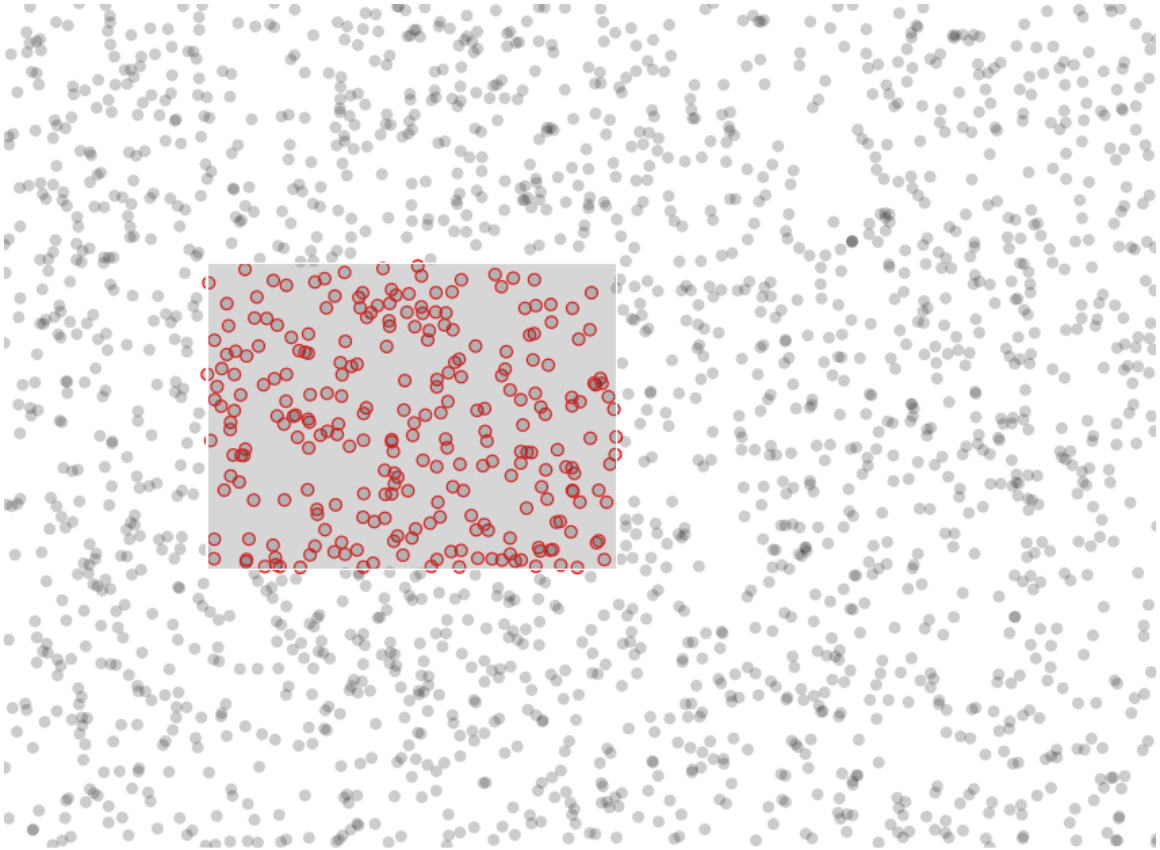
Brushing으로 할 수 있는 것들

- Zooming, Panning
  - https://observablehq.com/@d3/focus-context



Brushing으로 할 수 있는 것들

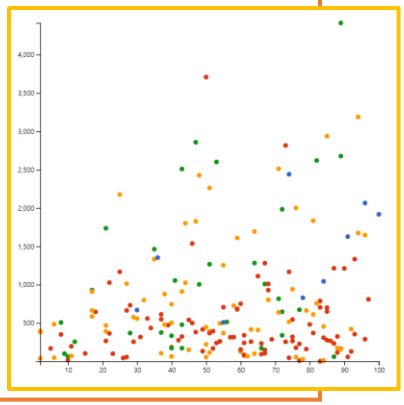
- Filtering
  - https://observablehq.com/@d3/brush-filter



Brush 만들어 보기

- 지난 실습 때 만든 산점도에 brush 기능을 넣어보자.

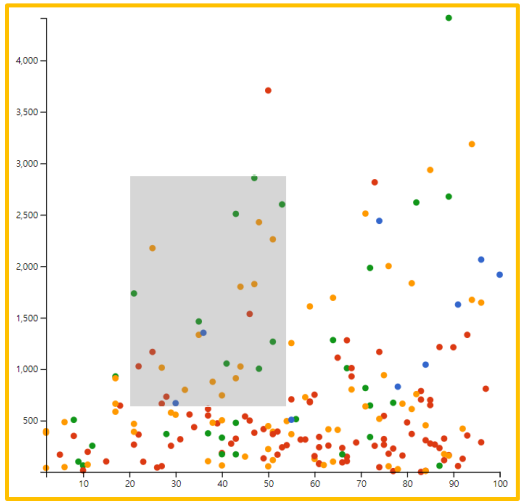
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Scatterplot</title>
</head>
<body>
  <div id="plot"></div>
  <script src="https://d3js.org/d3.v6.min.js"></script>
  <script>
  let data = [];  // TODO: https://pastebin.com/7WNHLVYk 에서 가져오기

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

  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));

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

  let y = d3.scaleLinear()
            .domain([
              d3.min(data, d => d.us_gross),
              d3.max(data, d => d.us_gross)
            ])
            .range([height, 0]);

  let color = d3.scaleOrdinal()
                .domain(['전체관람가', '7세이상', '15세이상', '19세이상'])
                .range(['#3366cc', '#109618', '#ff9900', '#dc3912']);

  let circle = svg
                .selectAll('circle')
                .data(data)
                .join('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));

  let xAxis = d3.axisBottom(x);
  let yAxis = d3.axisLeft(y);

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

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

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

  // TODO: brush 코드 작성

  </script>
</body>
</html>
```

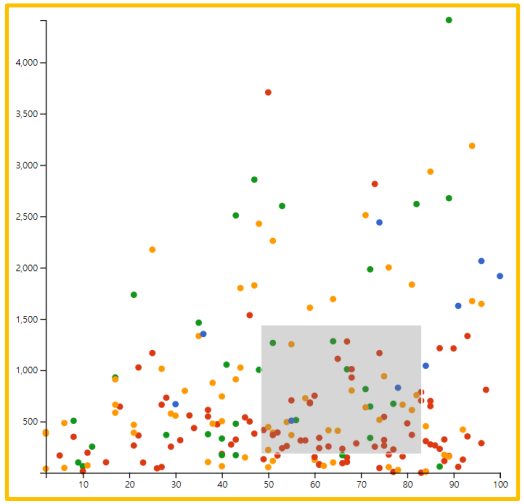


Brush 만들어 보기

- SVG 영역에 brush가 들어갈 영역을 추가하고 brush를 call한다.

```
let brush = d3.brush()
                .extent([[0, 0], [width, height]]);

svg
  .append('g')
  .call(brush);
```





마우스를 drag & drop하여 원하는 영역을 선택할 수 있다.

d3.brush()

- `d3.brush()`
  - 주로 SVG의 `<g>` selection에서 call을 통하여 그린다.
  - `<g>` 의 영역에 2차원 brush를 그린다.
    - 1차원으로 만들려면 `d3.brushX()` , `d3.brushY()`

- *`brush.extent([extent])`*
  - *`extent`* 인자를 넘길 경우, *brush*의 최대 영역을 설정한다.
    - *`extent = [[x0, y0], [x1, y1]]`* 에서
      `[x0, y0]` : 좌측 상단 좌표, *`[x1, y1]`* : 우측 하단 좌표
  - 별도의 인자를 넘기지 않을 경우,
    설정한 extent의 좌표(좌측 상단, 우측 하단)를 반환한다.
  - 여기서의 좌표는 *brush*가 call된 `<g>` 를 기준으로 한다.

Brushing을 활용한 Filtering – *brush*.on()

- 선택된 영역의 circle들만 테두리를 부여해보자.

- `brush.on(typenames[, listener])`
  - *brush*에 대한 callback 함수
  - *typenames*로 `"start brush end"` 중 하나 이상을 받는다.
    - `"start"` : 드래그를 시작하는 순간 *listener* 호출
    - `"brush"` : 드래그를 하는 동안 *listener* 호출
    - `"end"` : 드래그를 끝내는 순간 *listener* 호출
  - *listener*는 `selection.on()` 의 이벤트 핸들러와 동일하게 사용한다.
    - 첫 번째 인자로 `event` (일어난 이벤트 정보),
      두 번째 인자로 `d` (현재 요소에 bind된 데이터) 를 받는 함수이다. (버전 6 기준)
    - 두 번째 실습 PPT 슬라이드 66, 67 참조

Brushing을 활용한 Filtering

```
let brush = d3.brush()
              .extent([[0, 0], [width, height]])
              .on("start brush end", brushed);
svg
  .append('g')
  .call(brush);

function brushed({selection}) {
  if (selection === null) {
    circle.classed("selected", false);
  }
  else {
    let [[x0, y0], [x1, y1]] = selection;
    circle.classed("selected", d => {
      let xCoord = x(d.rotten_rating);
      let yCoord = y(d.us_gross);
      return x0 <= xCoord && xCoord <= x1
          && y0 <= yCoord && yCoord <= y1;
    });
  }
}
```

callback 함수를 brush에 추가

event로부터 현재 brush의 영역 좌표를 얻음

brush 영역에 속하는 circle에게만 "selected" 클래스 부여

"selected" 클래스에 검은색 테두리 적용

```
<head>
  <meta charset="utf-8">
  <title>Scatterplot</title>
  <style>
    .selected {
      stroke: black;
    }
  </style>
</head>
```

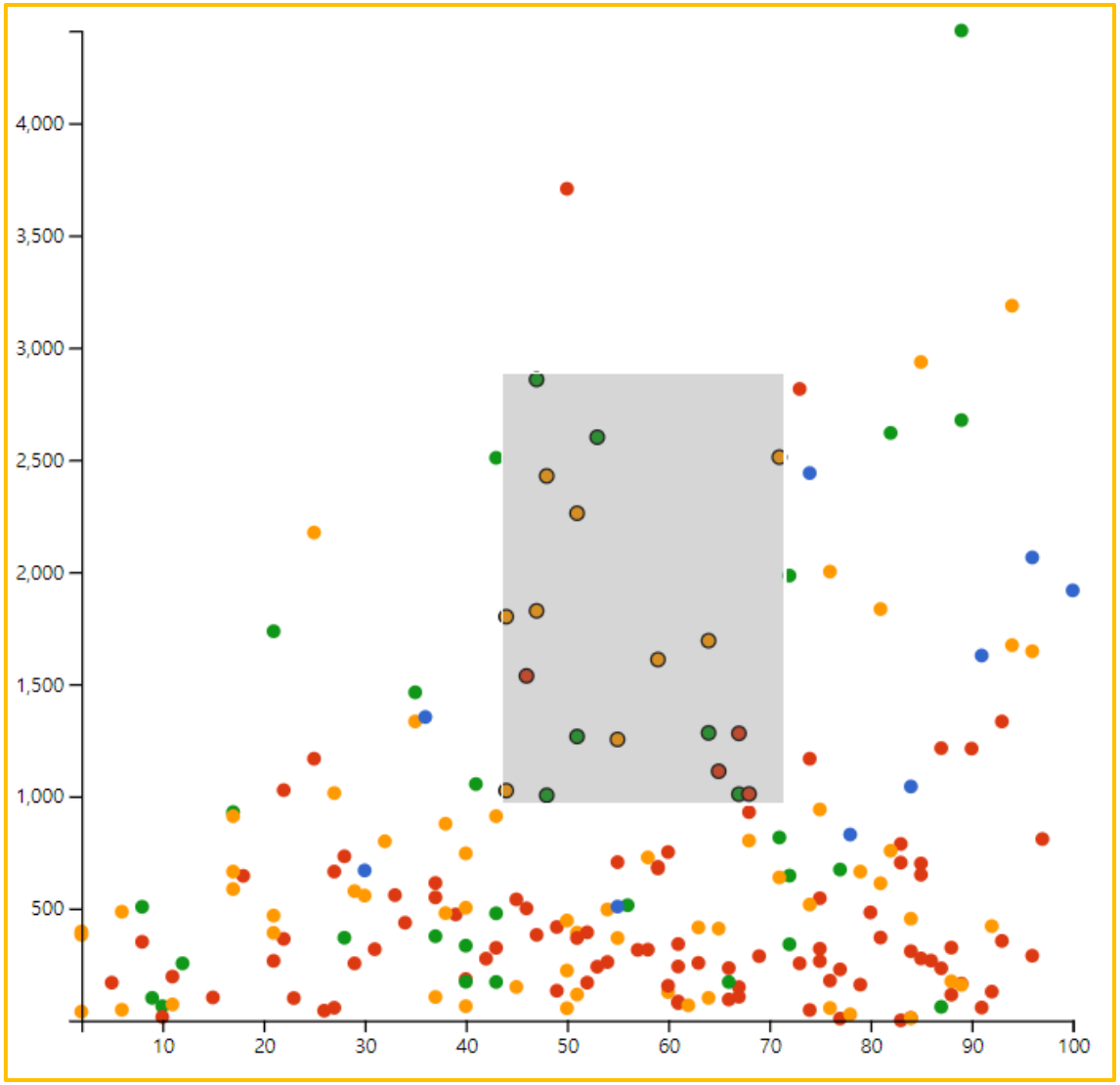
Brushing을 활용한 Filtering

```
let brush = d3.brush()
              .extent([[0, 0], [width, height]])
              .on("start brush end", brushed);
svg
  .append('g')
  .call(brush);

function brushed({selection}) {
  if (selection === null) {
    circle.classed("selected", false);
  }
  else {
    let [[x0, y0], [x1, y1]] = selection;
    circle.classed("selected", d => {
      let xCoord = x(d.rotten_rating);
      let yCoord = y(d.us_gross);
      return x0 <= xCoord && xCoord <= x1
          && y0 <= yCoord && yCoord <= y1;
    });
  }
}
```

Destructuring:
*listener*의 첫 번째 인자인 `event` 객체에서 `"selection"` 키의 값을 `selection`에 할당한다.
그리고 이를 다시 분해하여 `x0, y0, x1, y1`에 할당한다.
(공식 문서 참조)

```
<head>
  <meta charset="utf-8">
  <title>Scatterplot</title>
  <style>
    .selected {
      stroke: black;
    }
  </style>
</head>
```

Brushing을 활용한 Filtering



유의사항

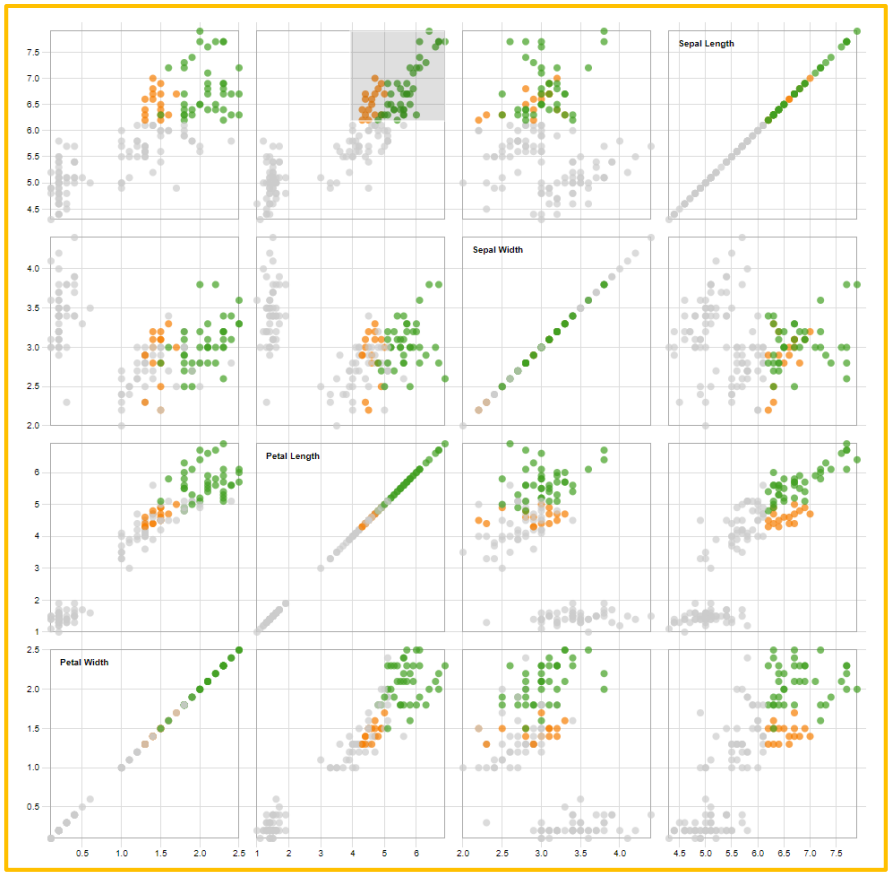
- Brush를 시각화보다 앞에 그리기 때문에,
  시각화 요소에 click이나 hover interaction을 적용해도 작동하지 않음
  - Brush 관련 event가 z좌표 상에서 앞에 있기 때문

- 너무 많은 연산을 다룰 경우 callback 중 `"brush"` callback은 매우 느려짐
  - 이 경우 `"end"` 에만 연산하거나, 연산을 줄여야 함

Linking

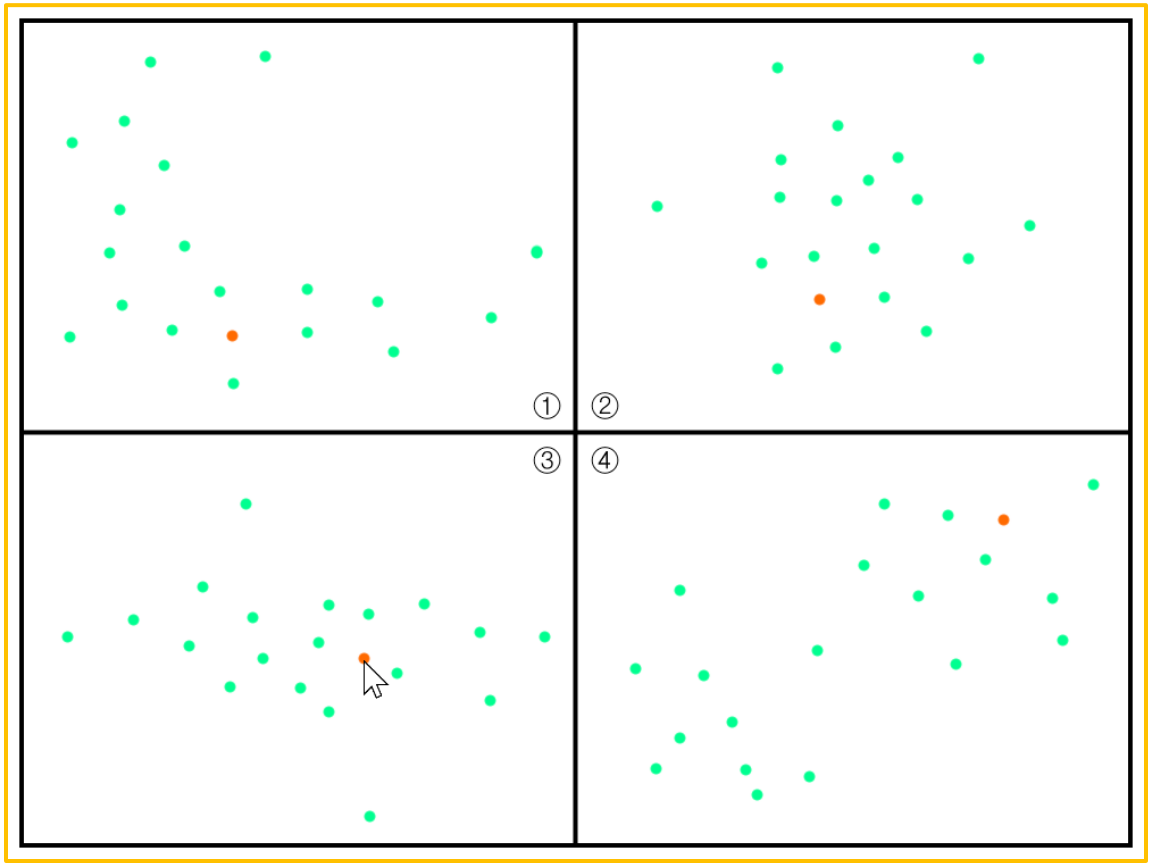
Linking이란?

- 여러 시각화를 한 화면에 그렸을 때, 한 시각화에서의 interaction이 다른 시각화에서도 적용되도록 하는 것



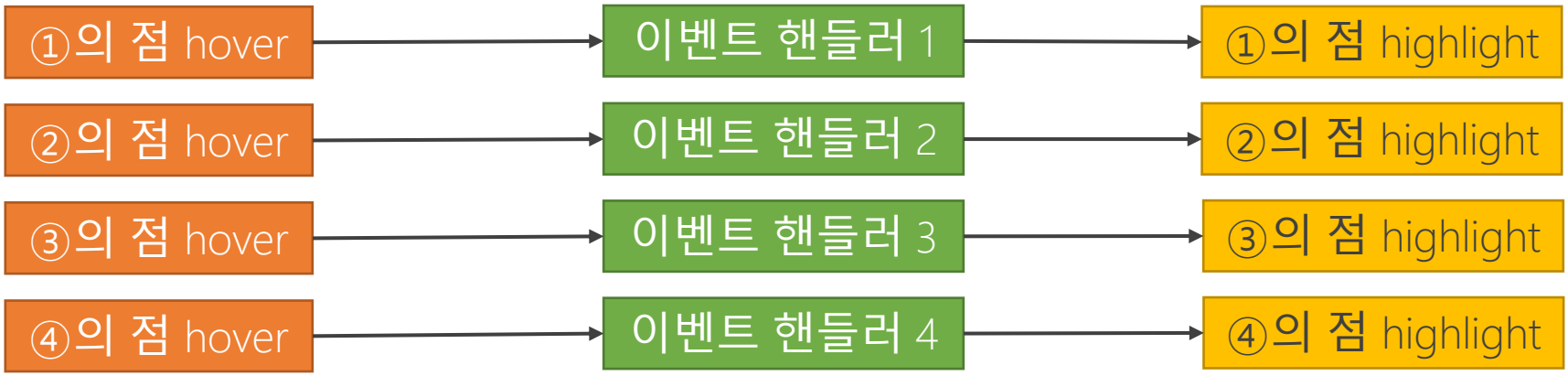
Linking 구현하기

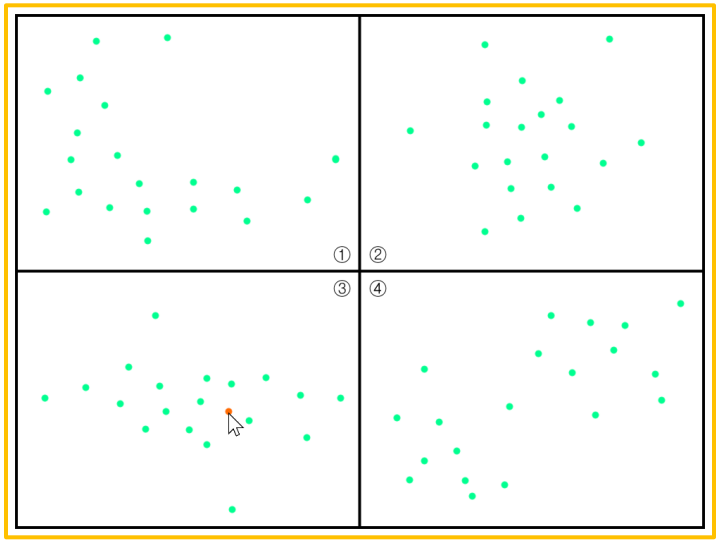
- 하나의 view에서 어떤 한 점에 hover하면
  네 개의 view에서 같은 데이터에 bind된 점들이 highlight되게 해보자.



Linking 구현하기

- Linking을 구현하지 않은 경우

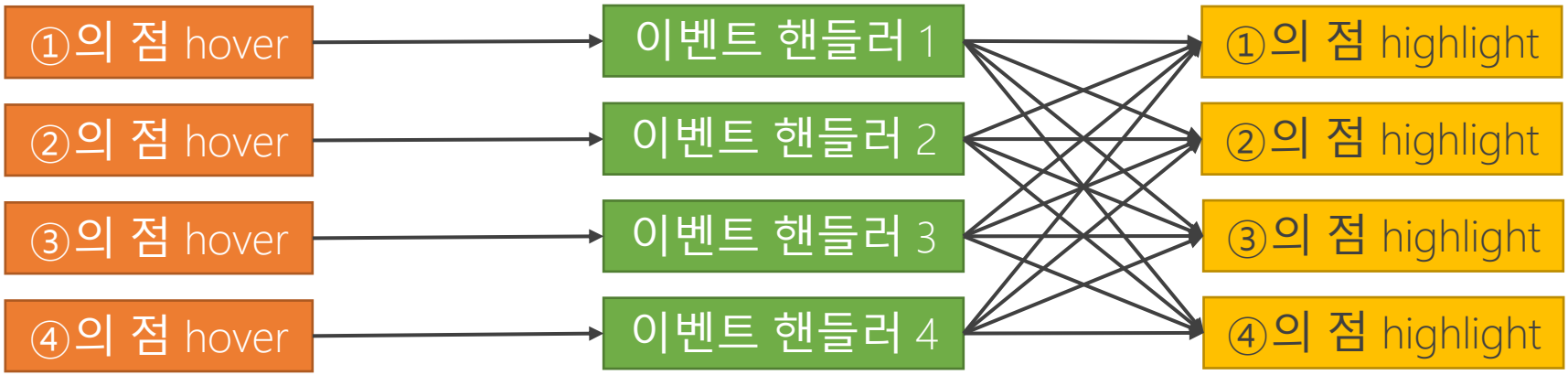


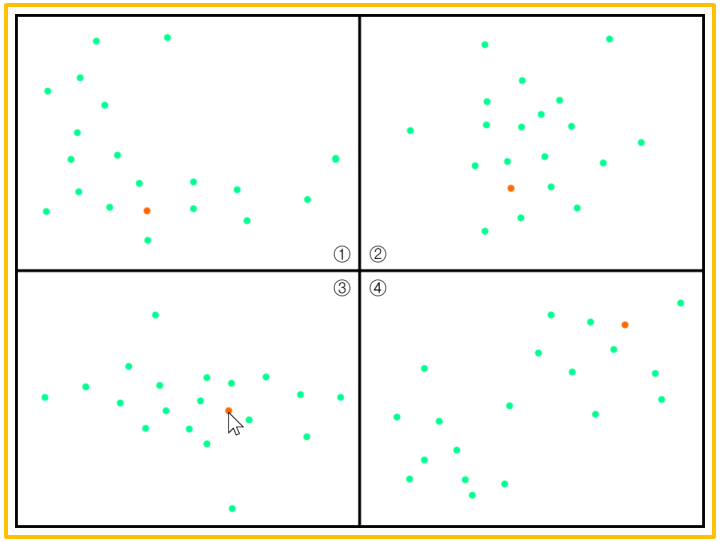


다른 view의 점들은 highlight되지 않는다.

Linking 구현하기

- Linking을 구현하면… → 불필요한 코드 반복

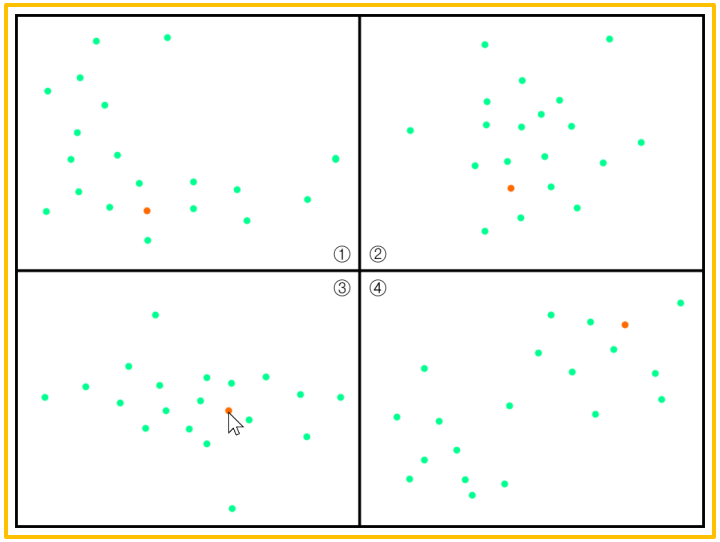




같은 데이터를 가진 다른 view의 점들이 동시에 highlight된다.

Linking 구현하기

- Highlight하는 부분을 일괄 처리하는 구조이므로...



같은 데이터를 가진 다른 view의 점들이 동시에 highlight된다.

다양한 Linking 방법들

- 후보군인 DOM들을 모두 선택하여
  그 중에서 linking의 대상이 되는 DOM들을 찾는다.
  - 모든 데이터에 일괄적으로 적용되는 filtering 등에 효과적이다.
- Class로 link하고 싶은 DOM들을 묶는다.
  - 여러 DOM에 공통된 visual feedback을 부여할 때 유용하다.

```
// 마우스를 올린 하나의 circle과 같은 id를 갖는 모든 DOM들을 선택하고,
// 이들에게 'hovered' 클래스를 부여
circle.on('mouseover', function(event, d) {
  d3.selectAll('.' + d.id).classed('hovered', true);
});
```

Other Utilities

d3-format / d3-time-format

- format: 숫자를 다양한 형태로 표시하거나 parsing할 때 유용하다.
  - 용례 참조: https://observablehq.com/@d3/d3-format

```
d3.format(".0%")(0.123);  // rounded percentage, "12%"
d3.format("($.2f")(-3.5); // localized fixed-point currency, "(£3.50)"
d3.format("+20")(42);     // space-filled and signed, "                 +42"
d3.format(".^20")(42);    // dot-filled and centered, ".........42........."
d3.format(".2s")(42e6);   // SI-prefix with two significant digits, "42M"
d3.format("#x")(48879);   // prefixed lowercase hexadecimal, "0xbeef"
d3.format(",.2r")(4223);  // grouped thousands with two significant digits, "4,200"
```

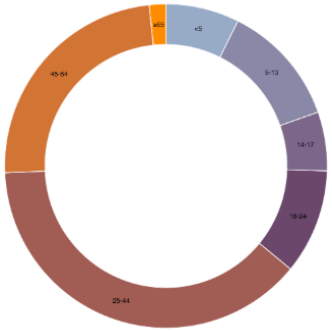
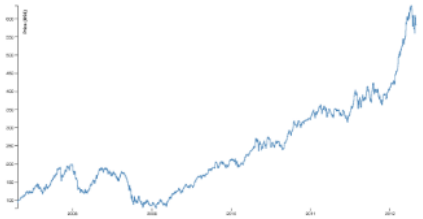
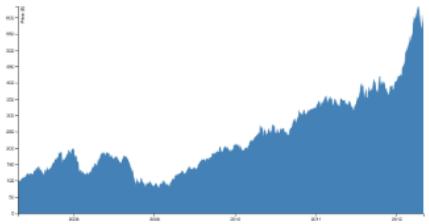
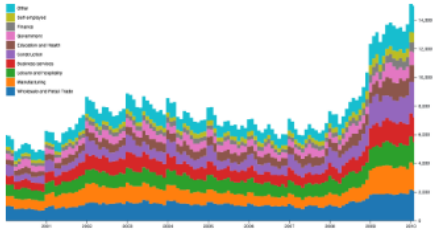
  - 버전 6에서 마이너스(−)를 표시하는 방법이 달라졌다.

- time-format: 시간과 관련된 parsing을 도와준다.

```
let formatTime = d3.timeFormat("%B %d, %Y");
formatTime(new Date); // "June 30, 2015"

let parseTime = d3.timeParse("%B %d, %Y");
parseTime("June 30, 2015"); // Tue Jun 30 2015 00:00:00 GMT-0700 (PDT)
```

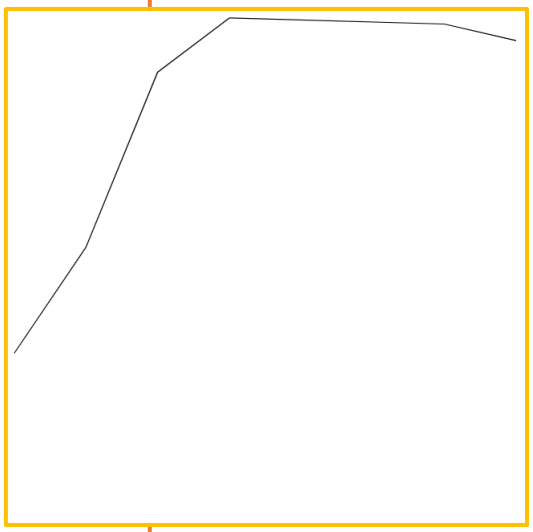
d3-shape

- API 참조: https://github.com/d3/d3-shape
- 용례 참조: https://observablehq.com/collection/@d3/d3-shape
- SVG에서 다양한 도형을 그리는 것을 도와준다.
  - SVG에서는 정의된 기본 도형(rect, circle...) 외의 다른 도형을 D3 없이 그리려면 복잡한 문법으로 좌표를 찍어야 한다.
  - `d3.arc`, `d3.pie`, `d3.line`, `d3.area`, curve 등이 있다.
- 대부분 path의 형태로 뽑아준다.
  - 생성된 좌표들을 path의 `'d'` attribute에 적용하자.

d3-shape – d3.line()

```
let data = [
  {date: new Date(2007, 3, 24), value: 93.24},
  {date: new Date(2007, 3, 25), value: 95.35},
  {date: new Date(2007, 3, 26), value: 98.84},
  {date: new Date(2007, 3, 27), value: 99.92},
  {date: new Date(2007, 3, 30), value: 99.80},
  {date: new Date(2007, 4,  1), value: 99.47},
];

let x = d3.scaleTime()
          .domain([new Date(2007, 3, 24), new Date(2007, 4, 1)])
          .range([0, 500]);
let y = d3.scaleLinear()
          .domain([90, 100])
          .range([500, 0]);

let line = d3.line()
    .x(function(d) { return x(d.date); })
    .y(function(d) { return y(d.value); });

d3.select('svg').attr('width', 500).attr('height', 500)
  .append('path')
    .attr('d', line(data))
    .attr('fill', 'none')
    .attr('stroke', 'black');
```



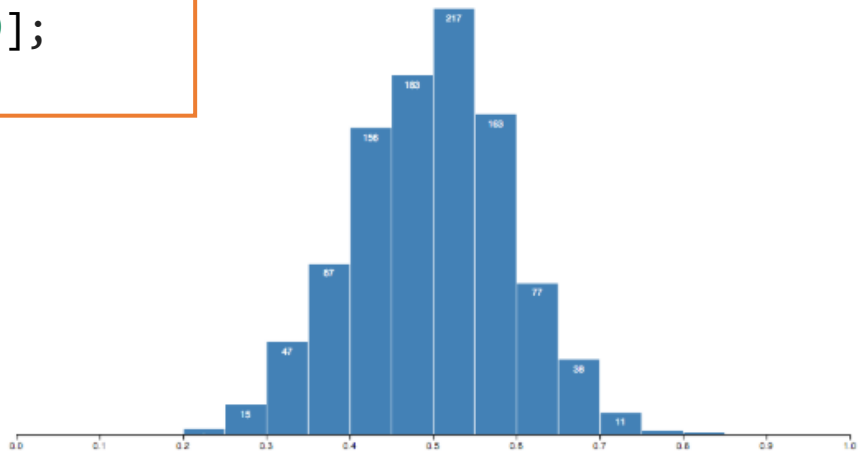
d3-bin

- API 참조: https://github.com/d3/d3-array/blob/master/README.md#bin
- 용례 참조: https://observablehq.com/@d3/d3-bin

- Array 데이터에서 히스토그램을 쉽게 그릴 수 있도록 변환해준다.
  - 버전 5까지는 d3-histogram이었지만 버전 6부터는 d3-bin이다.

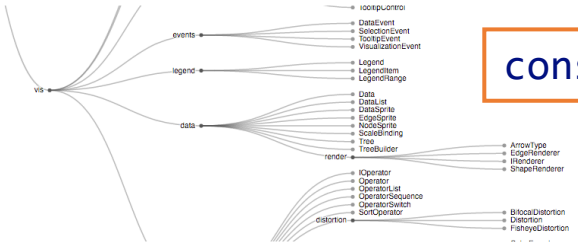
```
let array = [0, 1, 1, 2, 3, 3, 3, 5, 5, 5, 5, 5,
             6, 6, 7, 7, 7, 7, 8, 8, 8, 9];
console.log(d3.bin()(array));
```

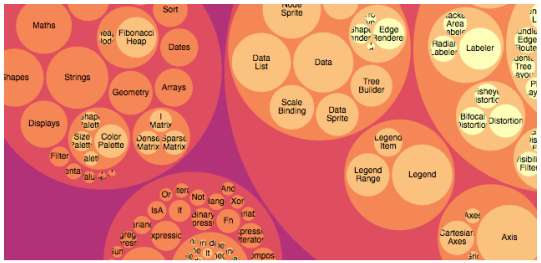
```
▼ (5) [Array(3), Array(4), Array(5), Array(6), Array(4)]
  ▶ 0: (3) [0, 1, 1, x0: 0, x1: 2]
  ▶ 1: (4) [2, 3, 3, 3, x0: 2, x1: 4]
  ▶ 2: (5) [5, 5, 5, 5, 5, x0: 4, x1: 6]
  ▶ 3: (6) [6, 6, 7, 7, 7, 7, x0: 6, x1: 8]
  ▶ 4: (4) [8, 8, 8, 9, x0: 8, x1: 10]
    length: 5
  ▶ __proto__: Array(0)
```

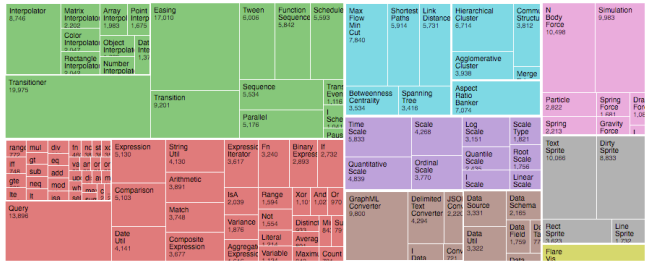


d3-hierarchy

- API 참조: https://github.com/d3/d3-hierarchy
- 용례 참조: https://observablehq.com/@d3/d3-hierarchy
- 한글 설명: https://observablehq.com/@jonghunpark/d3-hierarchy
- 계층이 있는 데이터를 tree, graph 등을
  그리기 쉬운 형태로 변환해준다.







```
console.log(d3.hierarchy(hierarchical));
```

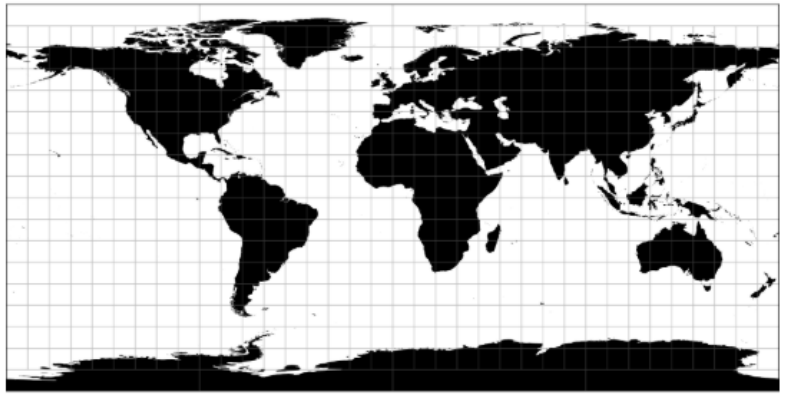
```
let hierarchical = {
  "name": "Eve",
  "children": [
    {
      "name": "Cain"
    },
    {
      "name": "Seth",
      "children": [
        {
          "name": "Enos"
        },
        {
          "name": "Noam"
        }
      ]
    },
    {
      "name": "Abel"
    },
    {
      "name": "Awan",
      "children": [
        {
          "name": "Enoch"
        }
      ]
    },
    {
      "name": "Azura"
    }
  ]
};
```

d3-geo

- API 참조: https://github.com/d3/d3-geo
- 용례 참조: https://observablehq.com/collection/@d3/d3-geo
  - 세계 지도: https://observablehq.com/@d3/world-map

- 지도를 그리고 지도 좌표에 projection하도록 도와준다.





오늘 배운 것

- Observable
- Load local files: `d3.csv` `d3.tsv` `d3.json` `d3.text` Open server
- Selection: Selection chaining `selection.merge` `selection.join` `selection.classed` `selection.each` `selection.call` `selection.property` `selection.text` `selection.html`
- Data binding: `selection.data` Table vs. SVG
- Brushing: `d3.brush` `brush.extent` `brush.on`
- Linking
- Other utilities: `d3.format` `d3.timeFormat` `d3.timeParse` `d3.bin` `d3.arc` `d3.pie` `d3.line` `d3.area` `d3.hierarchy`

Homework 2

과제 개요

- 간단한 visual interface를 만들어보자.
- 과제 의도
  - 실제 데이터를 활용한 D3의 기능 이해
  - 그동안 배운 D3의 다양한 기능 활용

- 데이터: 한국 프로야구 역대 순위 데이터
  - 1982년 ~ 2019년(year)의 각 구단(team)의
    등수(rank), 경기 수(games), 승(W), 무(D), 패(L), 승률(win_rate), 승차(diff_game)
    - 1999년, 2000년 데이터는 존재하지 않음
    - 1991년, 2011년에 같은 등수를 가진 팀 존재
  - JSON 파일로 제공

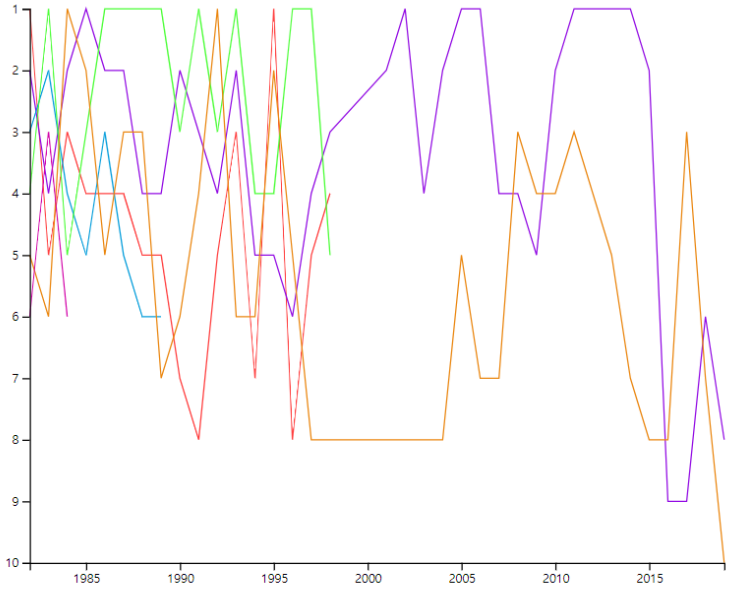
과제 개요

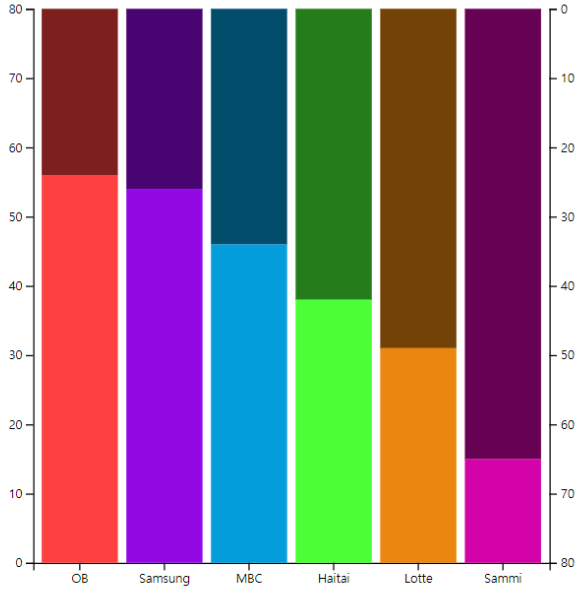
2020-54321 안단태

KBO Rankings

1982 Rank

rank	team	W	D	L	win_rate	diff_game
1	OB	56	0	24	0.700	0
2	Samsung	54	0	26	0.675	2
3	MBC	46	0	34	0.575	10
4	Haitai	38	0	42	0.475	18
5	Lotte	31	0	49	0.388	25
6	Sammi	15	0	65	0.188	41





필수 스펙 – 구조

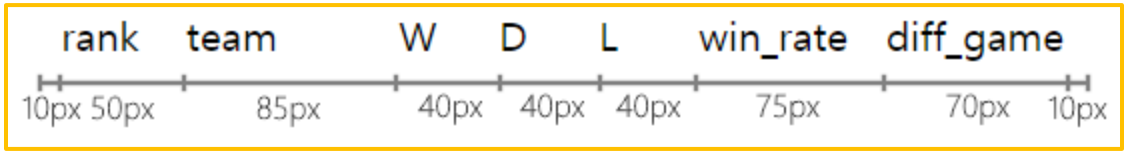
- 페이지 맨 위에 학번과 이름을 표시한다.
- 학번과 이름 바로 아래에 "KBO Rankings"를 표시한다. (따옴표 없이)
- "KBO Rankings" 바로 아래에 `<select>` 를 두 개 배치한다.
  - 첫 번째 `<select>` 에서 데이터에 존재하는 모든 연도를 선택할 수 있다.
  - 두 번째 `<select>` 에서 "Rank", "WR", "DG"를 선택할 수 있다. (따옴표 없이)
- `<select>` 바로 아래에 세 개의 `<svg>` 시각화를 수평으로 배치한다.
  - 맨 왼쪽에는 SVG width가 420px인 **표**(table)가 온다.
  - 중간에는 SVG width가 600px인 **라인 차트**(line chart)가 온다.
  - 맨 오른쪽에는 SVG width가 480px인 **누적 바 차트**(stacked bar chart)가 온다.
  - SVG height는 모두 500px이다.
- 시각화 요소가 SVG 경계에서 잘리지 않도록 한다.

필수 스펙 – 일반

- 세 개의 시각화는 함께 동작해야 한다.
  - 첫 번째 `<select>` 에서 연도를 바꿀 경우
    표, 라인 차트, 누적 바 차트 모두 동시에 변한다.
  - 두 번째 `<select>` 에서 값을 바꿀 경우 라인 차트에서만 변화가 생긴다.
- 팀마다 다른 색을 부여해야 한다.
  - 모든 팀의 색이 다를 필요는 없지만, 함께 등장하는 팀의 색은 달라야 한다.
  - 시각화에 변화가 생겨도 각 팀의 색은 같게 유지되어야 한다.
  - **힌트**: 팀에 따라 고유색을 지정하거나 D3 color scheme 중 cyclical을 이용한다.
- 현재 제공된 데이터에서만 동작하게 만들면 안 된다.
  - 미래에 2020년 이후의 프로야구 순위 데이터가 추가되어도 동작해야 한다.
- D3 버전은 6을 사용해야 한다.

필수 스펙 – 표

- 첫 번째 `<select>` 에서 선택한 연도(year)의 순위 표
  - 반드시 `<svg>` 로 구현해야 한다. (`<table>` 사용 금지)
  - 표 헤더에는 반드시 다음과 같은 **순서**와 **간격**으로 attributes를 표시한다.



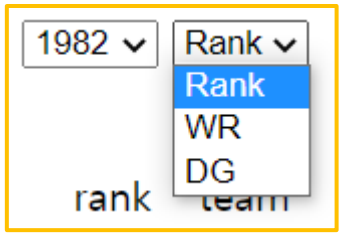
  - 표 본문에, 선택된 연도에 경기한 팀들의 정보를 표시한다.
    - 각 행(row)은 팀 이름(team)을 기준으로 data binding이 이루어져야 한다.
    - 순위(rank)가 높은 팀일수록 항상 상단에 위치해야 한다.
    - 순위가 같은 팀이 여럿 있는 경우 이들 간의 순서는 상관없다.
    - 각 열(column)에 해당 팀의 해당 attribute 값을 텍스트로 표시한다.
  - 텍스트가 서로 겹치면 안 된다.
    - **힌트**: 간격 스펙을 잘 따르고 순위가 같은 팀을 잘 처리하면 문제가 없을 것이다.
  - 텍스트의 글자색은 해당 팀에 부여한 색으로 칠한다.

필수 스펙 – 표

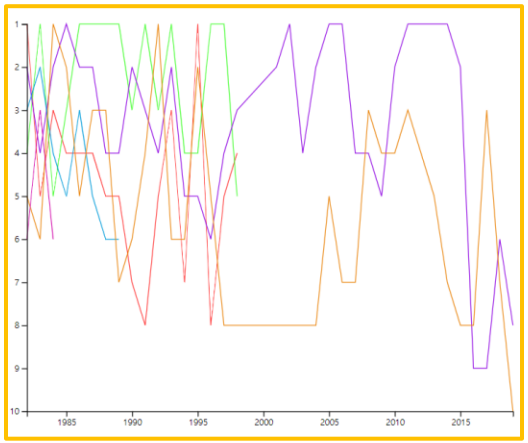
- 첫 번째 `<select>` 를 통해 원하는 연도(year)로 이동할 수 있어야 한다.
  - 선택한 연도에 없는 팀의 행은 즉시 제거한다.
  - 선택한 연도에 남은 팀의 행은 1.2초 간 transition하며 새로운 순위로 이동한다. 텍스트 내용은 transition하지 않고 즉시 바뀐다.
  - 선택한 연도에 새로 등장하는 팀의 행은 남은 팀들의 transition이 끝난 후에, 1.2초 간 fade in transition하며 해당 순위 위치에 등장한다.

필수 스펙 – 라인 차트

- 첫 번째 `<select>` 에서 **선택한 연도에 속해 있는 팀들**에 대해
  두 번째 `<select>` 에서 선택한 값의 연도 별 변화를 나타낸 라인 차트
  - x좌표: 연도(year) → 과거일수록 왼쪽에 위치
  - y좌표: 두 번째 `<select>` 에서 선택한 값에 따라...
    - "Rank"를 선택한 경우: 순위(rank) → 1등이 가장 상단에 위치
    - "WR"을 선택한 경우: 승률(win_rate) → 높을수록 상단에 위치
    - "DG"를 선택한 경우: 승차(diff_game) → **낮을수록** 상단에 위치

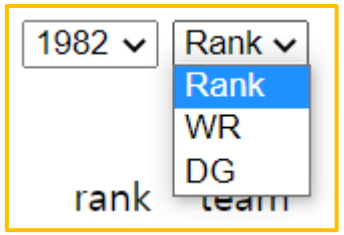


  - y축 값의 범위는 데이터 전체(모든 연도) 중 rank, win_rate, diff_game의 최솟값 ~ 최댓값 범위로 한다.
  - x축을 차트 하단에, y축을 차트 왼쪽에 그린다.



필수 스펙 – 라인 차트

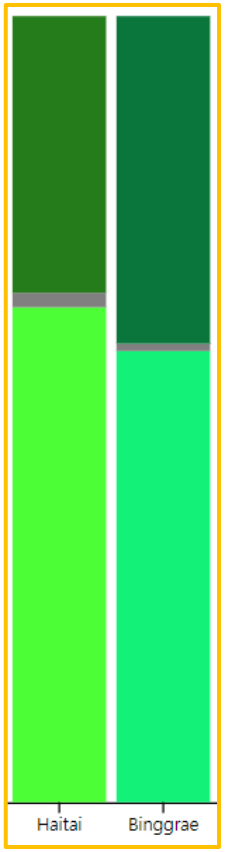
- 각 line은 팀 이름(team)을 기준으로 data binding이 이루어져야 한다.
- 각 line의 색은 해당 팀에 부여한 색으로 칠한다.
- 첫 번째 `<select>` 를 통해 원하는 연도(year)로 이동할 수 있어야 한다.
  - 선택한 연도에 없는 팀의 line은 즉시 제거한다.
  - 선택한 연도에 남은 팀의 line은 1.2초 간 transition하며 새로운 값을 나타내도록 바뀐다.
  - 선택한 연도에 새로 등장하는 팀의 line은 남은 팀들의 transition이 끝난 후에, 1.2초 간 fade in transition하며 등장한다.



- 두 번째 `<select>` 의 값을 바꾼 경우,
  모든 line의 모양이 1.2초 간 transition하며 새로운 값을 나타내도록 바뀐다.
- y축은 transition 없이 즉시 바뀌어도 상관없다.

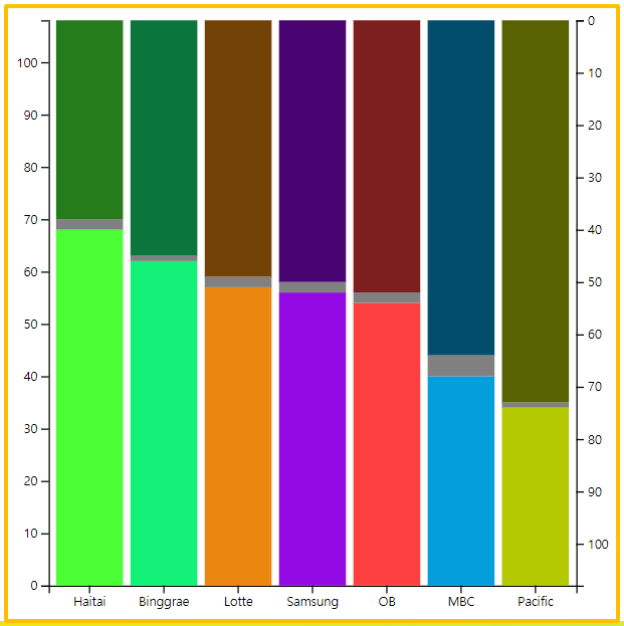
필수 스펙 – 누적 바 차트

- 첫 번째 `<select>` 에서 선택한 연도(year)의 팀들의 전적을 나타낸 누적 바 차트
  - x좌표: 팀 이름(team) → 순위(rank)가 높을수록 왼쪽에 위치
    - 가로로 이웃한 두 막대 사이에는 약간의 공백이 있어야 한다.
  - y좌표 및 높이: 세 개의 `<rect>` 에 다음을 적용한다.
    - 가장 **위**에 위치한 `<rect>` 의 세로 길이는 패(L)를 나타내고, 색은 `d3.rgb(해당_팀에_부여한_색).darker(2)` 로 칠한다.
    - 중간에 위치한 `<rect>` 의 세로 길이는 무(D)를 나타내고, 색은 `"#808080"` 의 색으로 칠한다.
    - 가장 **아래**에 위치한 `<rect>` 의 세로 길이는 승(W)을 나타내고, 색은 해당 팀에 부여한 색으로 칠한다.
    - 한 팀의 전적을 나타내는 세 `<rect>` 는 **서로 겹치지 않아야 한다**.
    - 한 팀의 전적을 나타내는 세 `<rect>` 를 하나의 `<g>` 로 묶어서 관리한다. 이 `<g>` 는 팀 이름(team)을 기준으로 data binding이 이루어져야 한다.



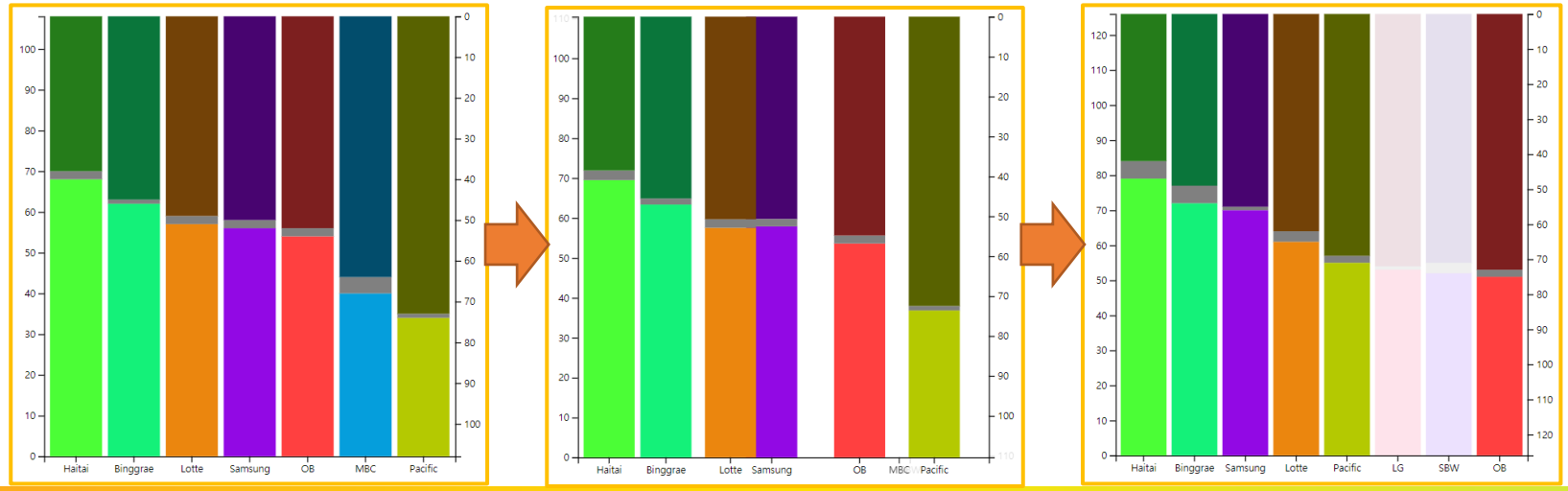
필수 스펙 – 누적 바 차트

- 첫 번째 `<select>` 에서 선택한 연도(year)의 팀들의 전적을 나타낸 누적 바 차트
  - x축을 차트 하단에 그린다.
  - y축을 두 개 그린다.
    - 둘 다 0 이상, 선택한 연도의 경기 수(games) 이하의 범위를 갖는다.
    - 하나는 차트 왼쪽에, 다른 하나는 차트 오른쪽에 그린다.
      - **힌트**: `d3.axisRight(scale)` 를 이용한다.
    - 왼쪽 y축은 작은 값일수록 하단에 위치해야 한다.
    - 오른쪽 y축은 큰 값일수록 하단에 위치해야 한다.
    - 축이 SVG 경계에서 잘리지 않도록 한다.



필수 스펙 – 누적 바 차트

- 첫 번째 `<select>` 를 통해 원하는 연도(year)로 이동할 수 있어야 한다.
  - 선택한 연도에 없는 팀의 막대는 즉시 제거한다.
  - 선택한 연도에 남은 팀의 막대는 1.2초 간 transition하며
    새로운 순위에 따라 x좌표가 바뀌고 새로운 승, 무, 패에 따라 높이가 바뀐다.
  - 선택한 연도에 새로 등장하는 팀의 막대는 남은 팀들의 transition이 끝난 후에,
    1.2초 간 fade in transition하며 해당 순위 위치에 등장한다.
- 연도가 바뀔 때 x축과 두 y축도 1.2초 간 transition하며 범위가 바뀐다.



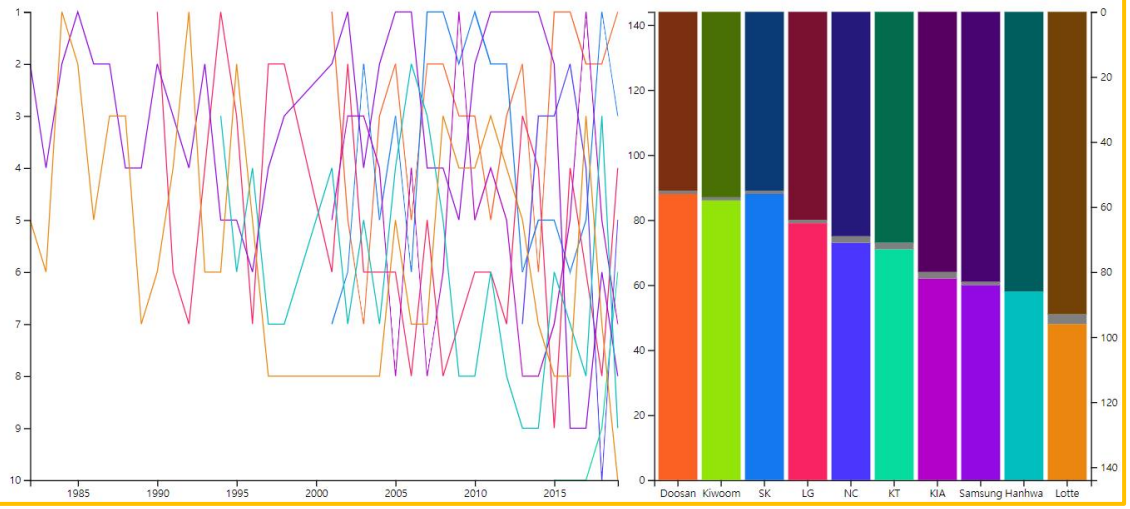
필수 스펙 – Linking

- 표의 본문 텍스트, 라인 차트의 line, 누적 바 차트의 막대를 hover할 경우
  - 해당되는 팀의 표 행의 텍스트를 굵게 한다.
    - 힌트: `selection.attr('font-weight', 900)`
  - 해당되는 팀의 line을 굵게 한다.
    - 힌트: `selection.attr('stroke-width', "5")`
  - 해당되는 팀의 막대 테두리를 진하게 표시한다.
    - 힌트: `selection.attr('stroke', "black").attr('stroke-width', "3")`
  - 이는 **같은 팀**의 데이터에 대해서는 **모두** link되어 이루어져야 한다.
- 위의 요소들을 hover하지 않을 경우 처음 상태로 돌아가야 한다.
  - 해당되는 팀의 표 행의 텍스트를 원래대로 얇게 한다.
  - 해당되는 팀의 line을 원래대로 얇게 한다.
  - 해당되는 팀의 막대 테두리를 표시하지 않는다.

필수 스펙 – Linking

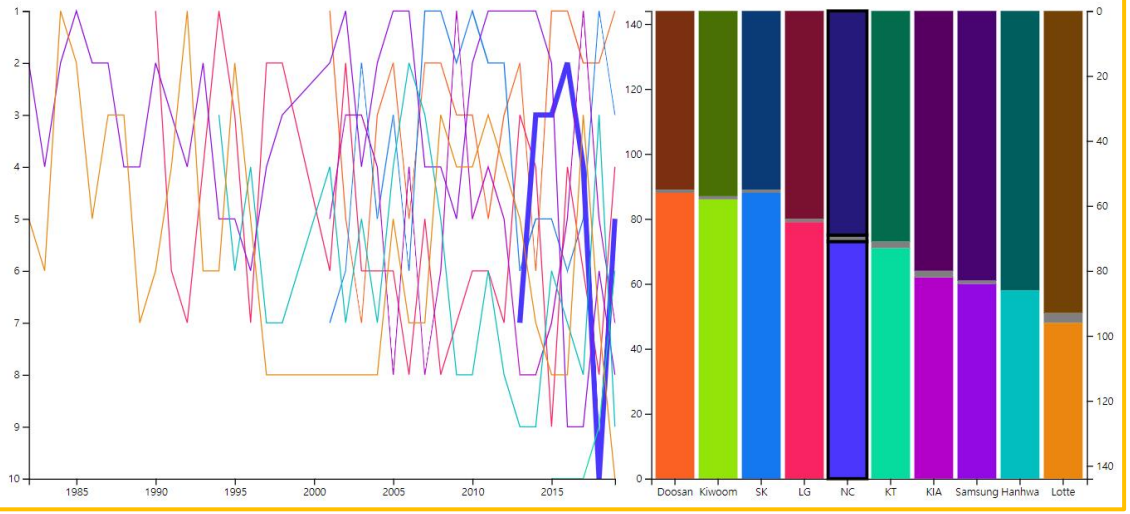
2019 Rank

rank	team	W	D	L	win_rate	diff_game
1	Doosan	88	1	55	0.615	0
2	Kiwoom	86	1	57	0.601	2
3	SK	88	1	55	0.615	0
4	LG	79	1	64	0.552	9
5	NC	73	2	69	0.514	14.5
6	KT	71	2	71	0.500	16.5
7	KIA	62	2	80	0.437	25.5
8	Samsung	60	1	83	0.420	28
9	Hanhwa	58	0	86	0.403	30.5
10	Lotte	48	3	93	0.340	39



2019 Rank

rank	team	W	D	L	win_rate	diff_game
1	Doosan	88	1	55	0.615	0
2	Kiwoom	86	1	57	0.601	2
3	SK	88	1	55	0.615	0
4	LG	79	1	64	0.552	9
5	**NC**	**73**	**2**	**69**	**0.514**	**14.5**
6	KT	71	2	71	0.500	16.5
7	KIA	62	2	80	0.437	25.5
8	Samsung	60	1	83	0.420	28
9	Hanhwa	58	0	86	0.403	30.5
10	Lotte	48	3	93	0.340	39



힌트

- 처음 한 번만 그리면 되는 부분과
  반복적으로 업데이트해야 하는 부분의 함수를 구분하면 편하다.

```
d3.json('data.json').then(function(data) {

  function init() {
    // ...
  }

  function update() {
    // ...
  }

  init();

  d3.selectAll('select').on('change', function(event, k) {
    // ...
    update();
  });
});
```

힌트

- 일반적인 업데이트 패턴
  - https://observablehq.com/@d3/general-update-pattern
    - 실습 2에서 배운 패턴
    - D3에서는 이 패턴보다 아래의 패턴을 따를 것을 권장
    - **주의**: `selection.transition()` 을 호출한 후에는 selection이 아닌 transition이 반환되므로 이 뒤에 `.data()` 같은 함수를 chaining할 수 없다.
  - https://observablehq.com/@d3/selection-join
    - 오늘(실습 3에서) 배운 패턴 (슬라이드 22 참조)
    - 더 기억하기 쉬우면서 여전히 강력하다고 하는데...
    - **주의**: `selection.transition()` 을 그냥 호출하면 안 되고 `selection.call()` 을 통해서 호출해야 한다.

힌트

- 시각화에 사용할 데이터를 편집할 때 다양한 함수들을 활용하자.
  - `array.map()` `d3.max()` `d3.min()` `d3.extent()` 등등…

- 데이터의 모든 entry는 string이다.
  - string을 쉽게 number로 바꾸려면? `+string`

```
d3.scaleLinear()
  .domain(d3.extent(data, k => k.W));   // k.W가 string으로 들어감 -> 오류 발생

d3.scaleLinear()
  .domain(d3.extent(data, k => +k.W));  // k.W가 number로 들어감
```

- 기존에 구현되어 있는 line chart, bar chart 등을 참고하면 좋다.
  - Observable 이용

제출 방법

- 하나의 HTML 페이지에 구현하여, 하나의 .html 파일만 eTL을 통해 제출
  - 파일 이름은 **본인이름**_2.html 으로 할 것
    - 예: 안단태_2.html
  - .js, .json 파일(data.json)을 비롯한 다른 모든 파일은 제출 시 포함시키지 말 것
  - D3.js도 웹에서 불러오도록 할 것
  - 압축하지 말 것
  - 데이터 이름은 반드시 data.json 으로 할 것

```
d3.json('data.json').then(function(data) { /* 코드 작성 */ });
```

- 채점은 최신 Chrome 브라우저에서 진행
- 스펙 관련 문의는 eTL의 질의응답 게시판을 이용
- 제출 기한: ~ 10월 28일 (수) 23:59