Webteknologi

Modul 7

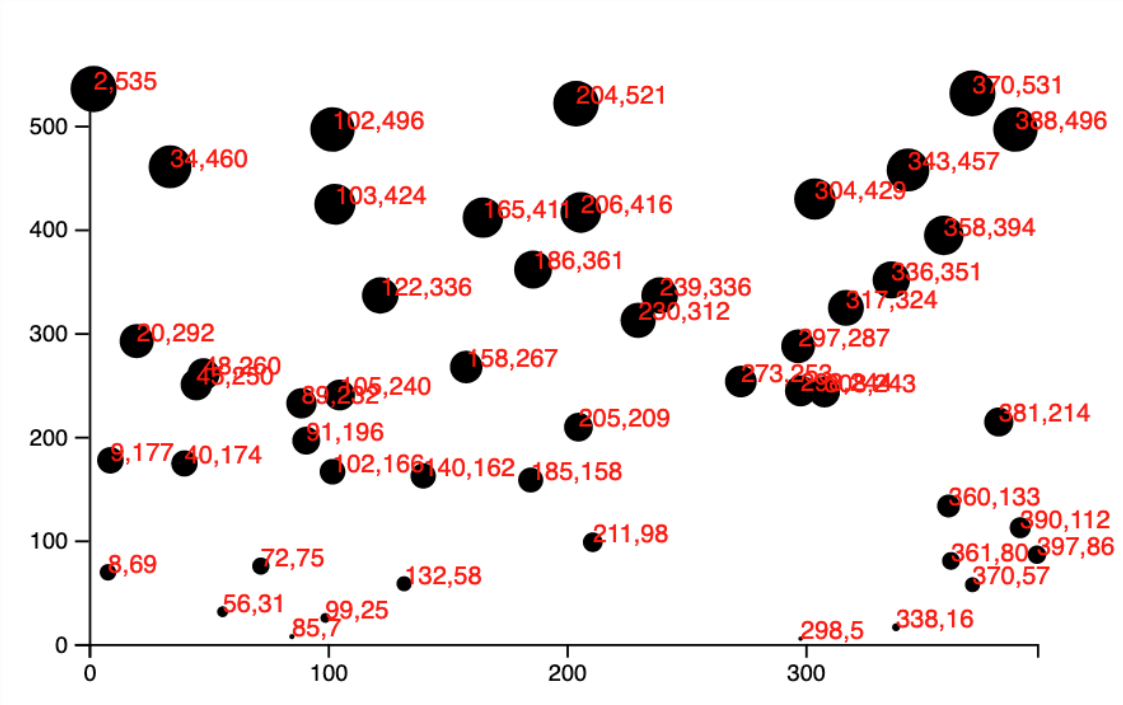
d3.js

Dagens plan

- 08.30 Opfølgning på sidst
- 09.00 Scatter Plots og 2d-arrays
- 09.20 Øvelser i Scatter Plots
- 10.00 Pause
- 10.30 Skaleringsfunktioner
- 11.00 Øvelser i datavisualisering
- 12.00 Slut

Scatter Plot

Indledning

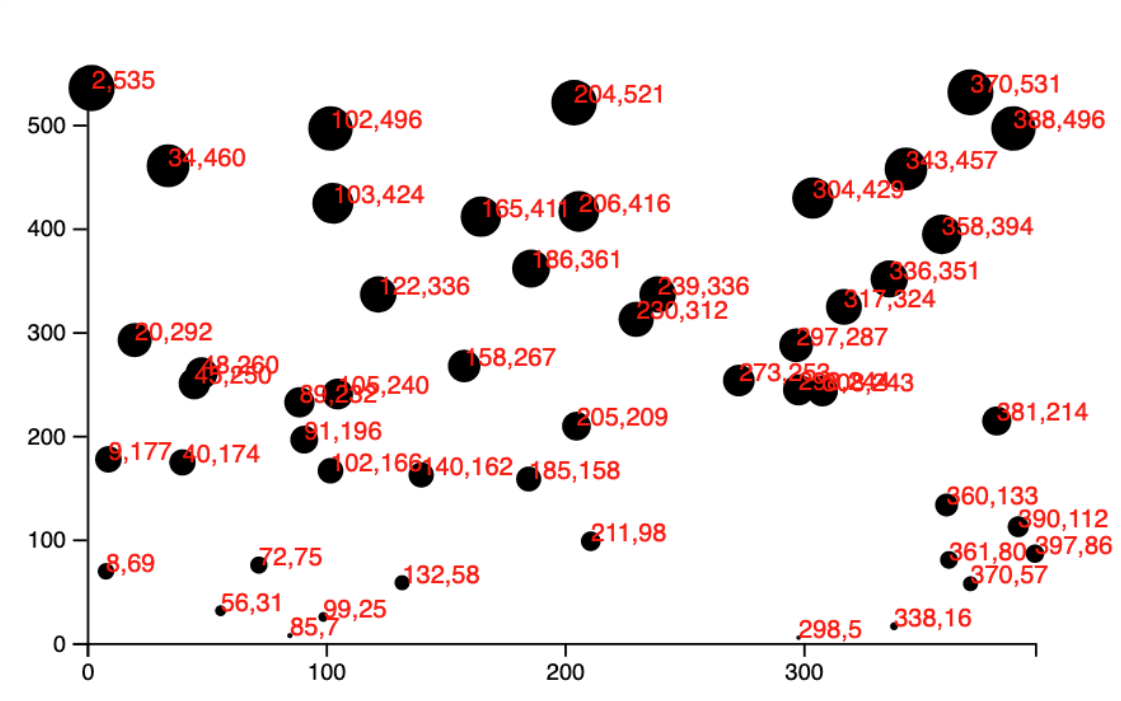


Et scatter plot er et diagram hvor alle målepunkter har (x,y)-koordinater, og så plotter man punkterne ind på et to-dimensionelt koordinatsystem (med x- og y-akser).

Scatter Plot

Indledning

Krav til data



Det stiller krav til vores data. Før havde vi bare et array af tal. Nu skulle vi have et array af koordinater.

Scatter Plot

Rent praksis kan vi lave et array af koordinater ved at have et **array af arrays**.

```
const dataset = [
  [5, 20],
  [480, 90],
  [250, 50],
  [100, 33],
  [330, 95],
  [410, 12],
  [475, 44]
];
```

Når man har et array af arrays, betyder det at hver value i vores array består af endnu et array:

```
let value = dataset[0];
console.log(value); // Udskriver: Array [ 5, 20 ]

// Udskriv første værdi i første array
console.log(dataset[0][0]); // Udskriver: 5
```

Når man henter værdier ud fra de "indre arrays", bruger man altså to sæt firkantparenteser: **[][]**. Den første at for at vælge det indre array, og den sidste, for at vælge hvilken værdi i det indre array, der skal bruges.

```
// Første værdi i første array
console.log(dataset[0][0]);
// Første værdi i andet array
console.log(dataset[1][0]);
// Anden værdi i fjerde array
console.log(dataset[3][1]);
// Anden værdi i femte array
console.log(dataset[4][1]);
```

Scatter Plot

For at lave et scatter plot med d3.js, kan man groft sagt gøre lige som med **barchart**, men med følgende væsentlige ændringer:

1. Erstat **rect** med **circle**.
2. Vælg en radius til circle (fx 5 pixels)
3. Lad cx være lig med første værdi
4. Lad cy være lig med anden værdi
5. Slet de gamle attributter: (x, y, width, height)

Scatter Plot


```
svg.selectAll("circle")  // <-- Før var det 'rect'
  .data(dataset)
  .enter()
  .append("circle")     // <-- Før var det 'rect'
  .attr("cx", function(d) {
    // 'd' er et koordinat på formen [x, y]
    return d[0]; // Første værdi i indre array (x)
  })
  .attr("cy", function(d) {
    return d[1]; // Anden værdi i indre array (y)
  })
  .attr("r", 5); // Radius er en konstant på '5'
```

Scatter Plot

Tryk på knappen (virker ikke i pdf)

Run

Scatter Plot

Findes på Canvas.

Scales i d3.js

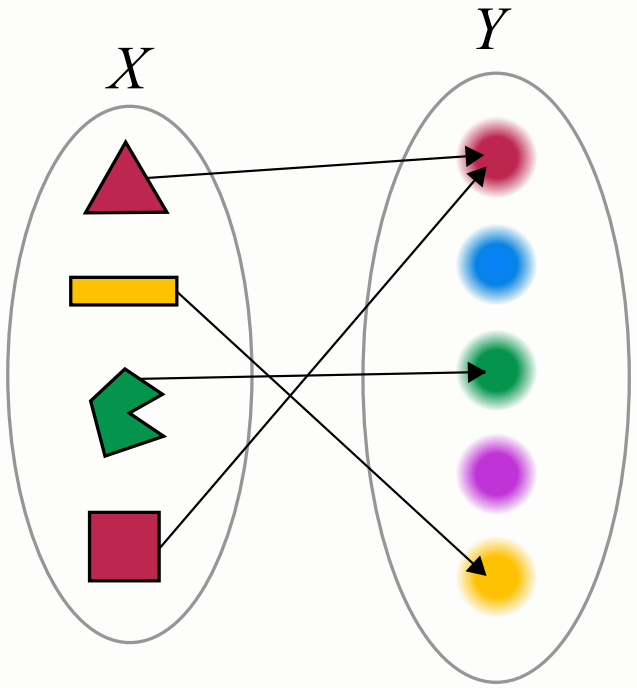
Indledning

Skalaer er funktioner der **mapper** fra **inputdomænet** til **outputområdet**.

- **mapping**: Det at tage en værdi fra et sæt og vælge en tilsvarende værdi i et andet sæt. Gøres med en funktion. En matematisk konstruktion.
- **inputdomænet**: Alle mulige målepunkter vi kan få som input. Som eksempel har folk typisk en højde imellem 50 og 220 cm - sådan cirka :)
- **outputområdet**: Alle mulige værdier som vi kan mappe til. Hvis vores SVG fx har en **height** på 500 pixels, så kan vores søjler i en barchart ikke være højere end det. Så er **outputområdet** groft sagt fra 0 til 500 pixels.

Scales i d3.js

Inputdomænet X og outputområdet Y. Pilene er mappings.



Du fået lavet nogle målepunkter der ligger imellem 0 og 1000.

Men du har et SVG-billede der kun er 300 pixels højt, og du gerne vil lave en barchart.

Så er løsningen at lave en mapping:

[0, 1000] → [0, 300].

Dvs. at værdierne fra "0 til 1000" bliver lavet om til værdier der ligger fra "0 til 300".

Man kan både mappe til noget større eller noget mindre.

En mapping function kunne konkret se sådan ud (hvis man bruger JavaScript):

```
function mapping(inputValue) {
  return inputValue / 1000 * 300 ;
}
```

Som eksempel så ville

- en inputværdi på 1000 blive til 300.
- en inputværdi på 500 blive til 150.
- en inputværdi på 50 blive til 15

Der er her tale om en **linæer** mapping. Funktionen kunne også kaldes en **normalisering**.

Scales i d3.js

Med d3.js behøver man ikke selv lave linæere skaleringsfunktioner eller normalisering-funktioner.

De er faktisk indbygget. Se her:

```
// Opret en lineær skala (map function)
const scale = d3.scaleLinear()
  .domain([0, 1000])
  .range([0, 300]);

// Prøv den af!
console.log(scale(1000)); // Udskriver 300
console.log(scale(500)); // Udskriver 150
console.log(scale(50)); // Udskriver 15
```

Anvende scales

Indledning

Formålet med skala-funktioner er nemt at kunne oversætte tal fra et **input-domæne** til et **output-område**.

Dette er fx nyttigt når man vil have diagrammer til automatisk til at skalere så de passer inden for rammerne af et **SVG-tag**.

I praksis løser man det ved at lave skala-funktioner der **automatisk tilpasser sig størrelsen på værdierne i input-data** og **størrelsen på output-området**. Se de næste slides.

d3.min() og d3.max() er simple funktioner som finder minimum og maximum i et datasæt.

```
const data = [5, 10, 2, 7, 12, 8];
console.log(d3.max(data)); // Udskriver: 12
console.log(d3.min(data)); // Udskriver: 2
```

Super smart når man skal finde ud af hvad hvad der er den største værdi i ens input-data.

Anvende scales

Hvis data er koordinater i stedet for tal, kan man bruge en callback function som beslutter, hvad det er man sammenligner på, for at finde max:

```
// Her er et sæt koordinater
const data = [[2, 5], [5, 15], [1, 10]];

// Vi skal finde max i 'data'
console.log(d3.max(data,
  // Funktionen vælger altid y-værdien fra et koordinat.
  function(d) { return d[1]; })
); // Udskriver: 15
// Som er y-værdien i koordinat 'data[1]'
```

Anvende scales

Sådan laver man skalerings-funktioner for x-aksen og y-aksen:

```
// Lav en skala for x-aksen
const xScale = d3.scaleLinear()
  .domain([0, d3.max(data, function(d) { return d[0]; })])
  .range([0, w]);

// Lav en skala for y-aksen
const yScale = d3.scaleLinear()
  .domain([0, d3.max(data, function(d) { return d[1]; })])
  .range([0, h]);
```

Anvende scales

Sådan laver man skalerings-funktioner for x-aksen og y-aksen:

```
// Lav en skala for x-aksen
const xScale = d3.scaleLinear()
  // Sæt domænet til at gå fra 0 til max x-værdi i 'data'
  .domain([0, d3.max(data, function(d) { return d[0]; })])
  // Sæt range til at gå fra 0 til 'w' (width på SVG)
  .range([0, w]);

// Lav en skala for y-aksen
const yScale = d3.scaleLinear()
  // Sæt domænet til at gå fra 0 til max y-værdi i 'data'
  .domain([0, d3.max(data, function(d) { return d[1]; })])
  // Sæt range til at gå fra 0 til 'h' (height på SVG)
  .range([0, h]);
```

Anvende scales

Lad os rette det gamle kode så vi nu bruger skalaer i visualiseringen. Før gjorde vi sådan her:

```
.attr("cx", function(d) {
    return d[0]; // Den originale x-værdi returneres
})
```

Men med skaleringsfunktionen for 'x' kan vi gøre sådan her:

```
.attr("cx", function(d) {
    return xScale(d[0]); // Den skalerede x-værdi
})
```

Og samme princip for y-aksen.

Anvende scales

Med skaleringsfunktioner er det ikke længere nødvendigt at tweake 'width' og 'height' på SVG-tagget så det passer til vores data. I stedet kan data skaleres automatisk og løbende med skalerings-funktioner.

Husk sidste gang hvor I skulle ændre et barchart så der var værdier over 100, der ikke blev "klippet over". Det her er den generelle løsning på det problem.

Anvende scales

Kapitlet kommer ind på andre interessante skalaer man kan bruge, fx logaritmisk, kvadratisk, tidsskala, og meget mere.

Derudover er der en række andre muligheder som også er værd at kigge nærmere på. Fx findes der metoder der kan:

- Gør tal pænere (**nice()**)
- Runde tal af til nærmeste heltal.
- **Clamp** (fjerne alle værdier uden for outputområdet)

Og mange flere.

Øvelser

Nu er du klar til at lave dagens resterende