CSC4130
Introduction to Human-Computer Interaction

Lecture 13
User Interface Technology: D3

Outline

- Introduction to D3
- Making a chart
- Data joins and basic interactivity
- Multiple views and advanced interactivity
- Advanced concepts

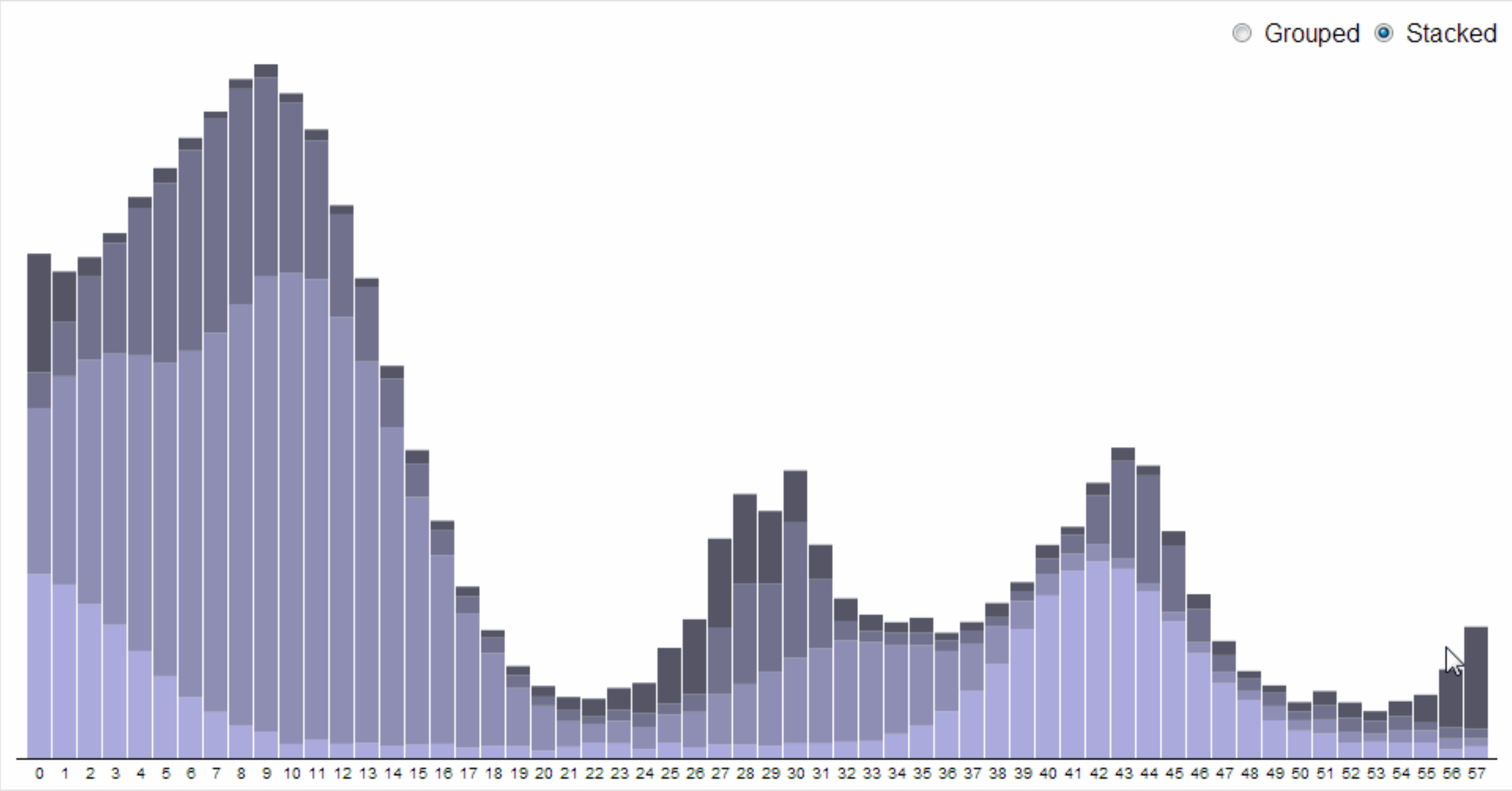
Outline

- Introduction to D3
- Making a chart
- Data joins and basic interactivity
- Multiple views and advanced interactivity
- Advanced concepts

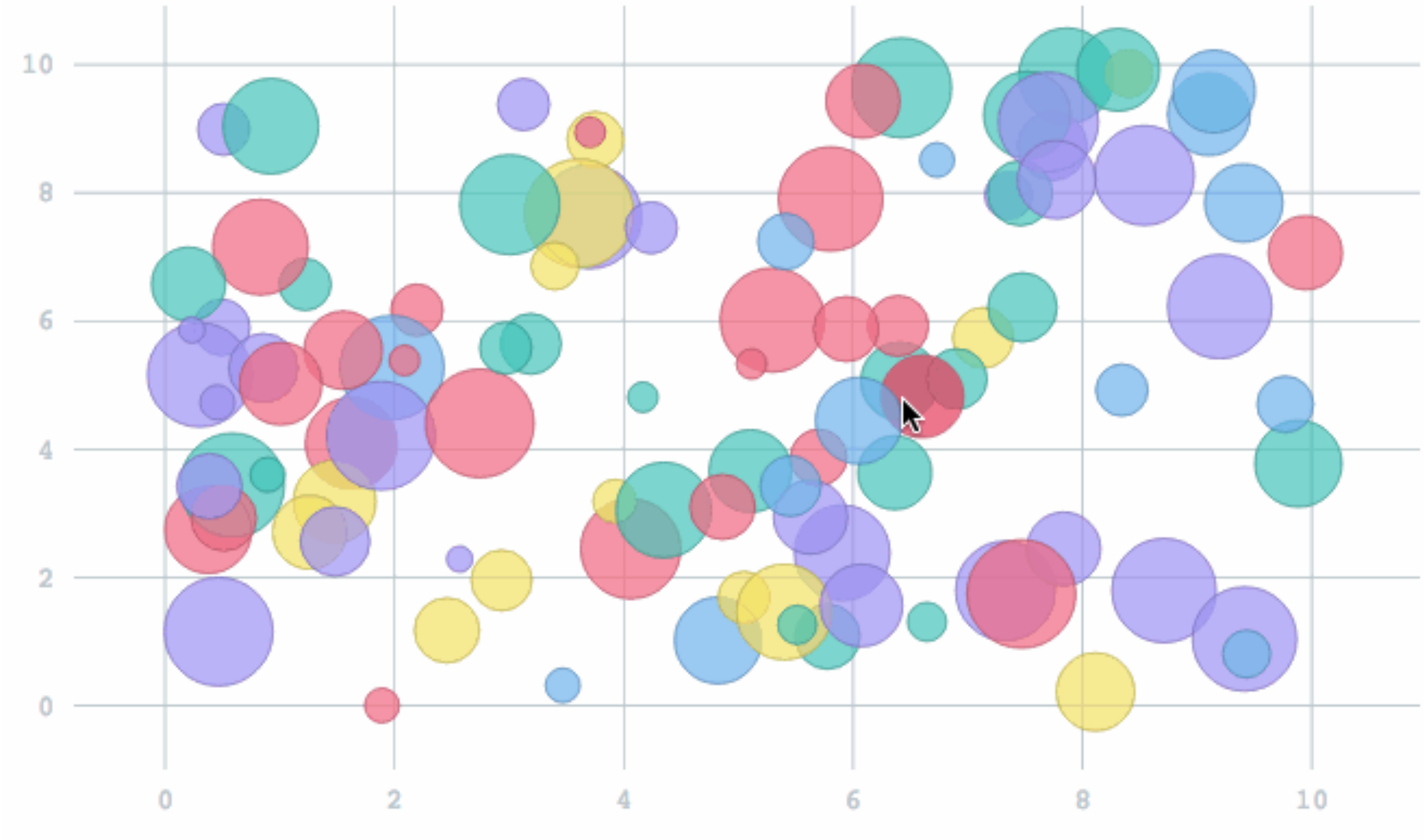
Introduction to D3

- D3 allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document.

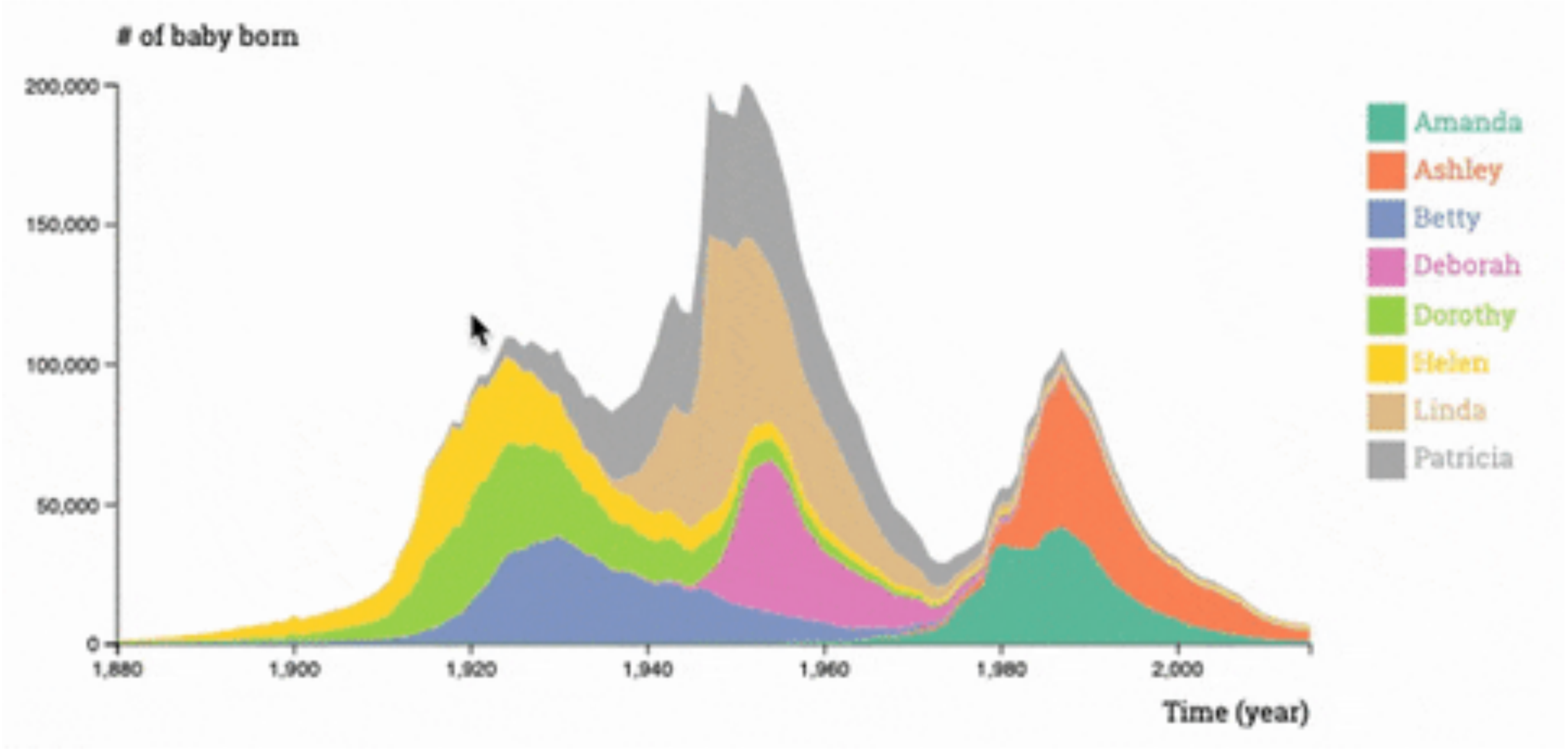
Stacked-to-Grouped Bars



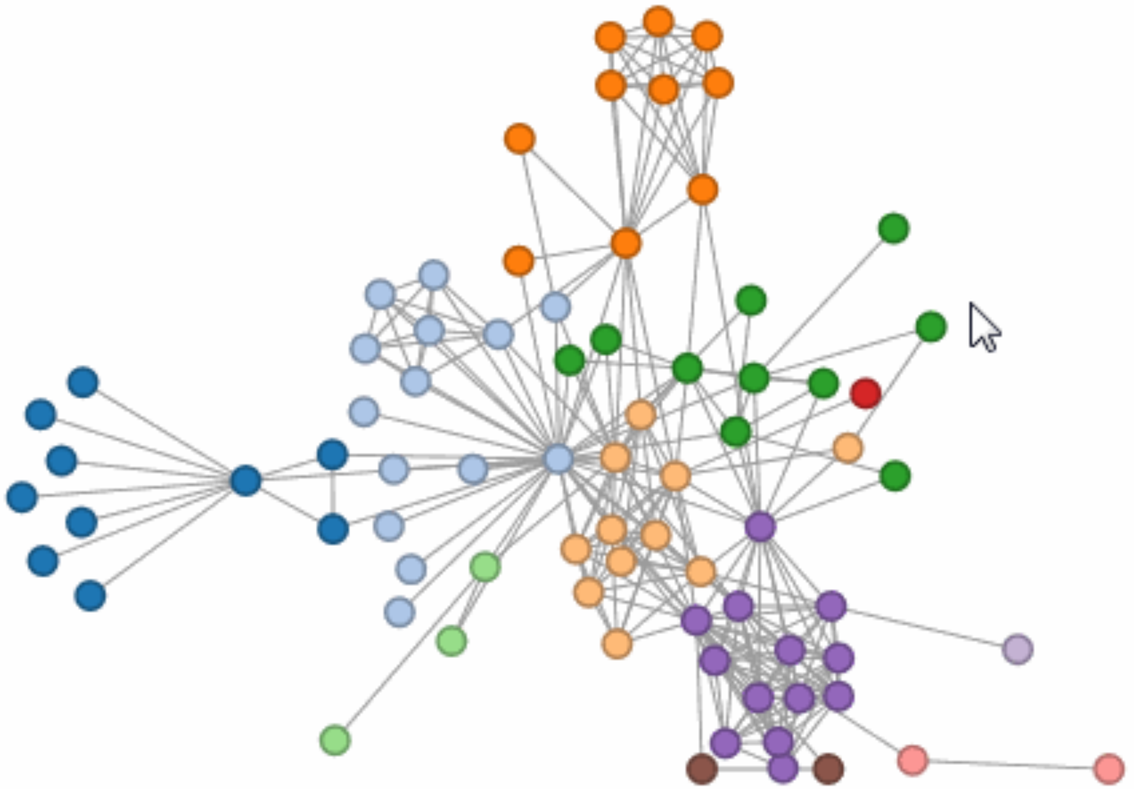
Introduction to D3



Introduction to D3



Introduction to D3



- D3 integration

```
project/
  index.html
  css/
    style.css
  js/
    d3.min.js
    main.js
```

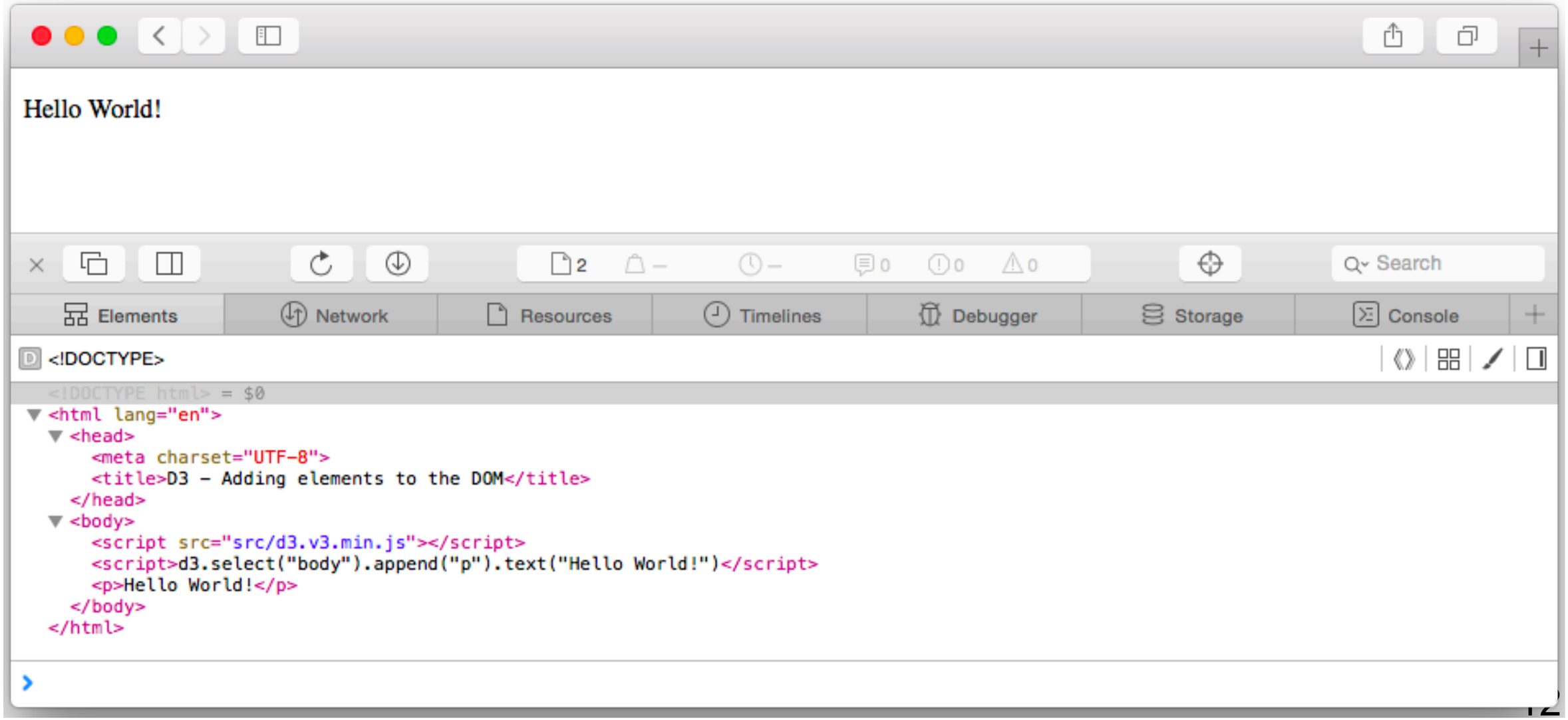
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>D3 Project</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>

  <script src="js/d3.min.js"></script>
  <script src="js/main.js"></script>
</body>
</html>
```

- Overview of SVG
  - SVG is defined using markup code similar to HTML
  - SVG elements don't lose any quality when they are resized
  - SVG elements can be included directly within any HTML document or dynamically inserted into the DOM with JS
  - Before you can draw SVG elements, you have to add an <svg> element
  - The SVG coordinate system places the origin (0/0) in the top-left corner of the svg element

Introduction to D3


```
<svg width="400" height="50">

<!-- Rectangle (x and y specify the coordinates of the upper-left corner -->
<rect x="0" y="0" width="50" height="50" fill="blue" />

<!-- Circle: cx and cy specify the coordinates of the center and r the radius -->
<circle cx="85" cy="25" r="25" fill="green" />

<!-- Ellipse: rx and ry specify separate radius values -->
<ellipse cx="145" cy="25" rx="15" ry="25" fill="purple" />

<!-- Line: x1,y1 and x2,y2 specify the coordinates of the ends of the line -->
<line x1="185" y1="5" x2="230" y2="40" stroke="gray" stroke-width="5" />

<!-- Text: x specifies the position of the left edge and y specifies the vertical position of the baseline -->
<text x="260" y="25" fill="red">SVG Text</text>

</svg>
```

- Add a DOM Element with D3



- D3 select
  - The select() method uses CSS selectors as input to grab page elements. It will return a reference to the first element in the DOM that matches the selector
  - Alternatively, if you need to select more than one element, use selectAll()

- D3 append
  - After selecting a specific element, we can apply an operator, such as .append(‘p’)
  - The append() operator adds a new element as the last child of the current selection. We specified "p" as the input argument, so an empty paragraph has been added to the end of the *HTML body*. The new paragraph is automatically selected for further operations

- Binding data to visual elements

```
const provinces = ['AB', 'BC', 'MB', 'NB', 'NL', 'NT', 'NS', 'NU', 'ON', 'PE', 'QC', 'SK', 'YT'];

const p = d3.select('body').selectAll('p')
  .data(provinces)
  .enter()
  .append('p')
  .text('Array Element');
```

Reference to the target container

Selection representing the elements (paragraphs) we want to create

Loads the dataset (array of strings).

Creates new data-bound elements/placeholder

Takes the empty placeholder selection and appends a paragraph to the DOM for each element

Adds a string to each newly created paragraph

- Dynamic properties
  - Access to the corresponding values from the dataset, you have to use *anonymous functions*

```
// Our preferred option: ES6 arrow function syntax
.text(d => d);

// Alternative: Traditional function syntax
.text( function(d) { return d; } );
```

- HTML attributes and CSS properties

```
const provinces = ['AB', 'BC', 'MB', 'NB', 'NL', 'NT', 'NS', 'NU', 'ON', 'PE', 'QC', 'SK', 'YT'];

// Append paragraphs and highlight one element
let p = d3.select('body').selectAll('p')
    .data(provinces)
    .enter()
  .append('p')
    .text(d => d)
    .attr('class', 'custom-paragraph')
    .style('font-weight', 'bold')
    .style('color', d => {
      if(d == 'BC')
        return 'blue';
      else
        return 'red';
    });
```

- HTML attributes and CSS properties

```
const numericData = [1, 2, 4, 8, 16];

// Add <svg> element (drawing space)
const svg = d3.select('body').append('svg')
    .attr('width', 300)
    .attr('height', 50);

// Add rectangle
svg.selectAll('rect')
    .data(numericData)
    .enter()
  .append('rect')
    .attr('fill', 'red')
    .attr('width', 50)
    .attr('height', 50)
    .attr('y', 0)
    .attr('x', (d, index) => index * 60);
```

Introduction to D3

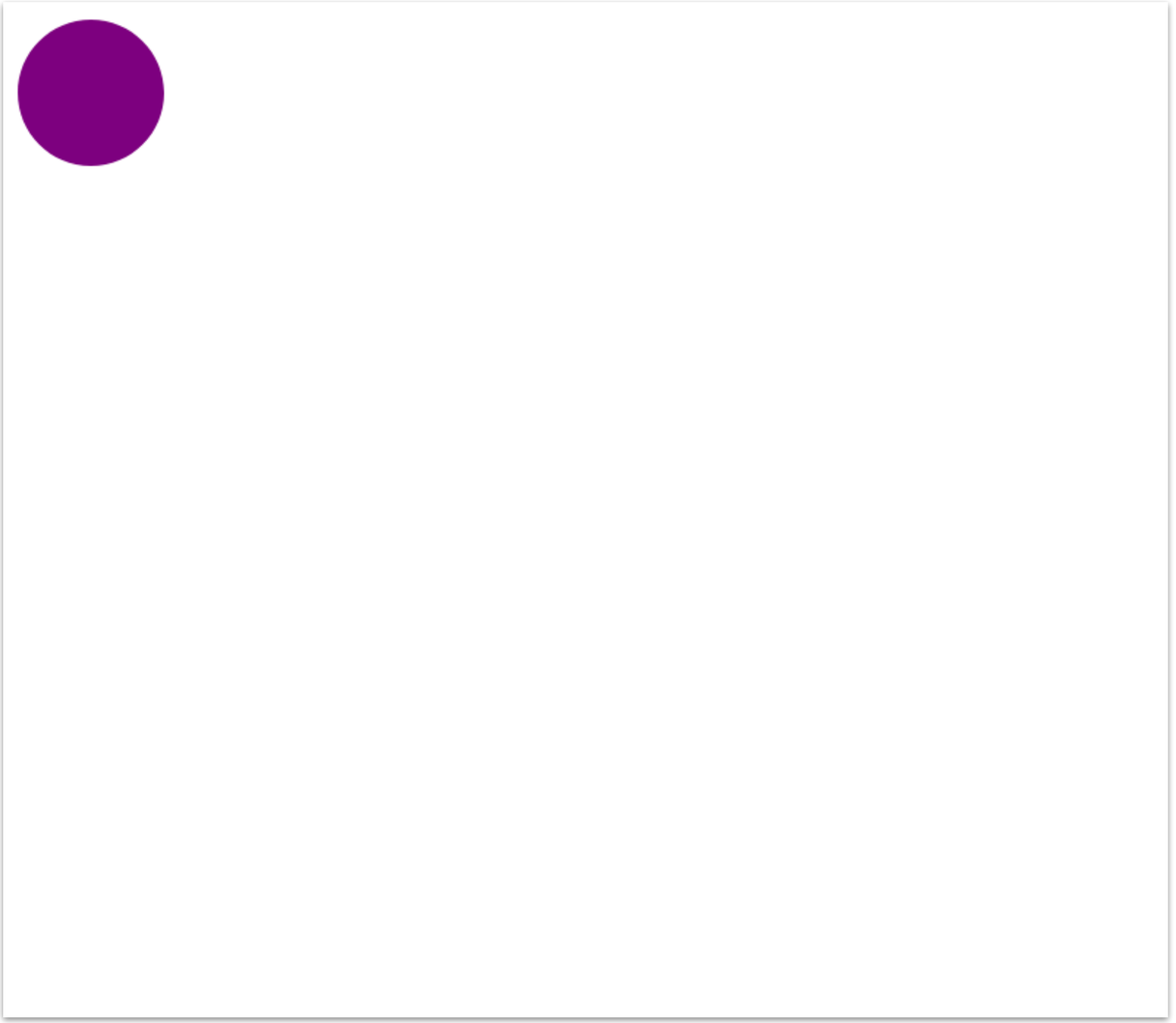
- Create a new D3 project
- Append a new SVG element to your HTML document with D3 (width: 500px, height: 500px)
- Draw circles with D3
- Define dynamic properties
  - Set the x/y coordinates and make sure that the circles don't overlap each other
  - Radius: *large sandwiches* should be twice as big as small ones
  - Colours: use two different circle colours. One colour (fill) for cheap products < 7.00 USD and one for more expensive products
  - Add a border to every circle (SVG property: stroke)

```
const sandwiches = [
    { name: "Thesis", price: 7.95, size: "large" },
    { name: "Dissertation", price: 8.95, size: "large" },
    { name: "Highlander", price: 6.50, size: "small" },
    { name: "Just Tuna", price: 6.50, size: "small" },
    { name: "So-La", price: 7.95, size: "large" },
    { name: "Special", price: 12.50, size: "small" }];
```

- Loading external data
  - Instead of typing the data in a local variable, which is only convenient for very small datasets, we can load data *asynchronously* from external files. The D3 built-in methods make it easy to load JSON, CSV, and other files
  - d3.csv()
  - d3.json()

```
d3.csv('data/sandwiches.csv')
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error('Error loading the data');
  });
```

D3 SVG exercise



D3 SVG exercise


```
d3.select("body")
  .append("svg")
  .attr("width", 50)
  .attr("height", 50)
  .append("circle")
  .attr("cx", 25)
  .attr("cy", 25)
  .attr("r", 25)
  .style("fill", "purple");
```

Outline

- Introduction to D3
- Making a chart
- Data joins and basic interactivity
- Multiple views and advanced interactivity
- Advanced concepts

Making a chart

- SVG groups
  - The group element (<g> </g>) helps us to organize other elements and to apply *transformations*. In this way, we can create hierarchical structures

```
// Create group element
const group = svg.append('g');

// Append circle to the group
const circle = group.append('circle')
        .attr('r', 4)
        .attr('fill', 'blue');
```

Making a chart

- SVG groups
  - Invisible but we can apply transformations, for example *translate()* or *rotate()*, to the group and it will affect the rendering of all child elements

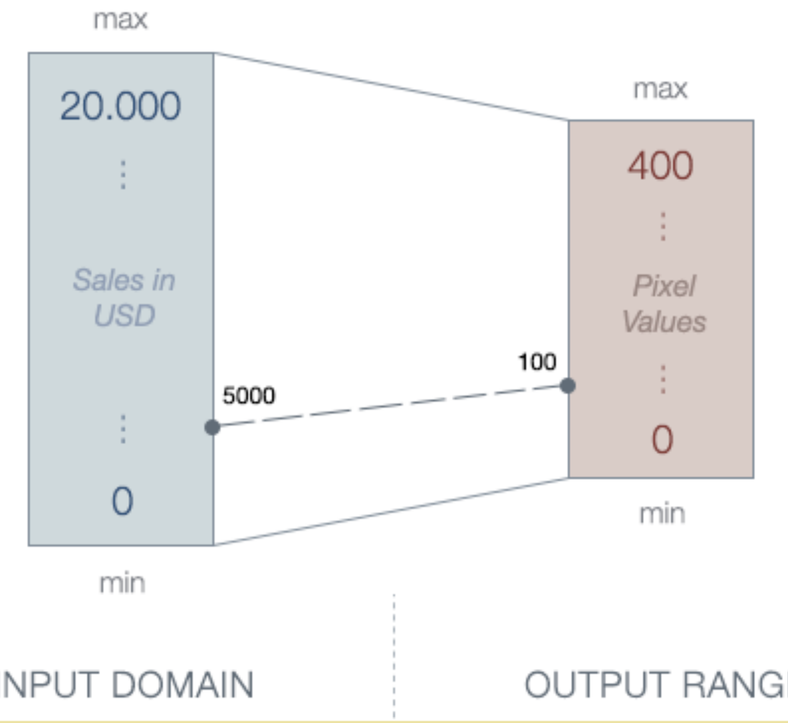
```
const group = svg.append("g")
    .attr("transform", "translate(70, 50)");
```

Making a chart

- Scales
  - Scales are functions that map from an input domain to an output range
  - D3 provides built-in methods for many different scales: linear, ordinal, logarithmic, square root, and so on. Most of the time you will use *linear scale functions*, so we will focus on learning this type of scale
  - You can read more about D3 scales here: https://github.com/d3/d3-scale/blob/master/README.md

Making a chart

- Scales
- Visualize the monthly sales of an ice cream store. The input data are numbers between 0 and 20,000 USD and the maximum height of the chart is 400px. We take an input inverval (called **domain**) and transform it into a new output interval (called **range**).



Making a chart

- Scales

```
// Creating a linear scale function
const iceCreamScale = d3.scaleLinear()
        .domain([0, 20000])
        .range([0, 400]);

// Call the function and pass an input value
iceCreamScale(5000);      // Returns: 100
```

Making a chart

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

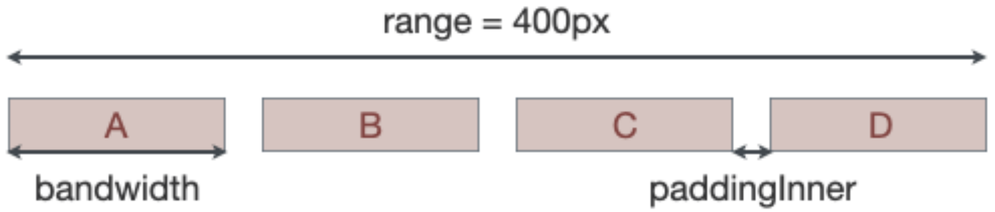
- Scales

```
const quarterlyReport = [
        { month: 'May', sales: 6900 },
        { month: 'June', sales: 14240 },
        { month: 'July', sales: 25000 },
        { month: 'August', sales: 17500 }
];

// Returns the maximum value in a given array (= 25000)
const max = d3.max(quarterlyReport, d => d.sales);

// Returns the minimum value in a given array (= 6900)
const min = d3.min(quarterlyReport, d => d.sales);

// Returns the min. and max. value in a given array (= [6900,25000])
const extent = d3.extent(quarterlyReport, d => d.sales);
```

Making a chart

- Scales
  - Time scales: time scales are a variant of linear scales that have a temporal domain. They require JS date objects
  - Ordinal scales: create ordinal scales with a discrete domain

```
// Create an ordinal scale function
const xScale = d3.scaleBand()
        .domain(['Vanilla', 'Cookies', 'Chocolate', 'Pistachio'])
        .range([0, 400])                    // D3 fits n (=4) bands within a 400px space
        .paddingInner(0.05);  // Adds spacing between bands

// By definition all bands have the same width
// and you can get it with `xScale.bandwidth()`

// We can use JS .map() instead of manually specifying all possible values
const months = quarterlyReport.map(d => d.month);
months // Returns: ['May', 'June', 'July', 'August']
```



Making a chart

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

- Scales
  - Colour scales: built-in color palettes that work like ordinal scales and can also be accessed like other scales

```
// Construct a new ordinal scale with a range of ten categorical colours
var colorPalette = d3.scaleOrdinal(d3.schemeCategory10);

// We can log the color range and see 10 distinct hex colours
console.log(colorPalette.range());
// ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b", "#e377c2", "#7f7f7f", "#bcbd22", "#17becf"]

// Specify domain (optional)
colorPalette.domain(['Vanilla', 'Cookies', 'Chocolate', 'Pistachio']);

// Use color palette
colorPalette("Chocolate") // Returns: #2ca02c
```

Making a chart

- Refine axis
  - Automatically adjust the spacing and labels for a given scale and range

```
var xAxis = d3.axisBottom()
        .scale(xScale)
        . ... // Add options here
```

- Number of ticks: .ticks(5)
- Tick format, e.g. as percentage: .tickFormat(d3.format(".0%"))
- Predefined tick values: .tickValues([0, 10, 20, 30, 40])
- Remove tick marks at the beginning and end of an axis: .tickSizeOuter(0)

Making a chart

- Axes
  - Provide four methods to create axes with different orientations and label placements (d3.axisTop, d3.axisBottom, d3.axisLeft, and d3.axisRight) which can display reference lines for D3 scales automatically
  - Use call() to add into svg

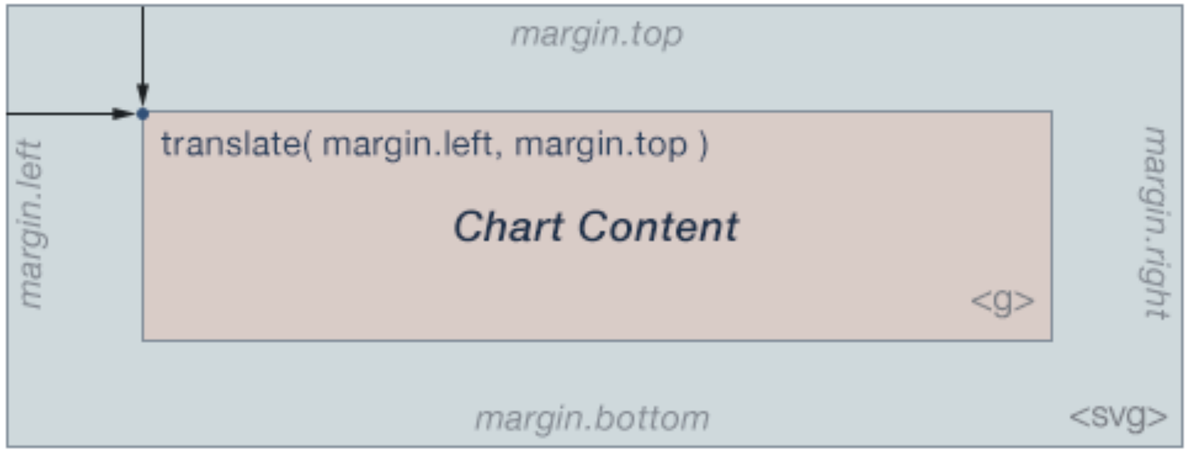
```
// Create a horizontal axis with labels placed below the axis
const xAxis = d3.axisBottom();

// Pass in the scale function
xAxis.scale(xScale);
```

```
svg.append('g')
    .attr('class', 'axis x-axis')
    .call(xAxis);
```

Making a chart

- Draw a bar chart

```
const margin = {top: 5, right: 5, bottom: 20, left: 50};

        // Width and height as the inner dimensions of the chart area
        const width = 500 - margin.left - margin.right,
        height = 140 - margin.top - margin.bottom;
// Define 'svg' as a child-element (g) from the drawing area and include spaces
        const svg = d3.select('#chart').append('svg')
                .attr('width', width + margin.left + margin.right)
                .attr('height', height + margin.top + margin.bottom)
                .append('g')
                .attr('transform', `translate(${margin.left}, ${margin.top})`);
```



Making a chart

- Draw a bar chart

```
// Initialize linear and ordinal scales (input domain and output range)
const xScale = d3.scaleLinear()
    .domain([0, d3.max(data, d => d.sales)])
    .range([0, width]);

const yScale = d3.scaleBand()
    .domain(data.map(d => d.month))
    .range([0, height])
    .paddingInner(0.15);

// Initialize axes
const xAxis = d3.axisBottom(xScale)
    .ticks(6)
    .tickSizeOuter(0);

const yAxis = d3.axisLeft(yScale)
    .tickSizeOuter(0);
```

Making a chart

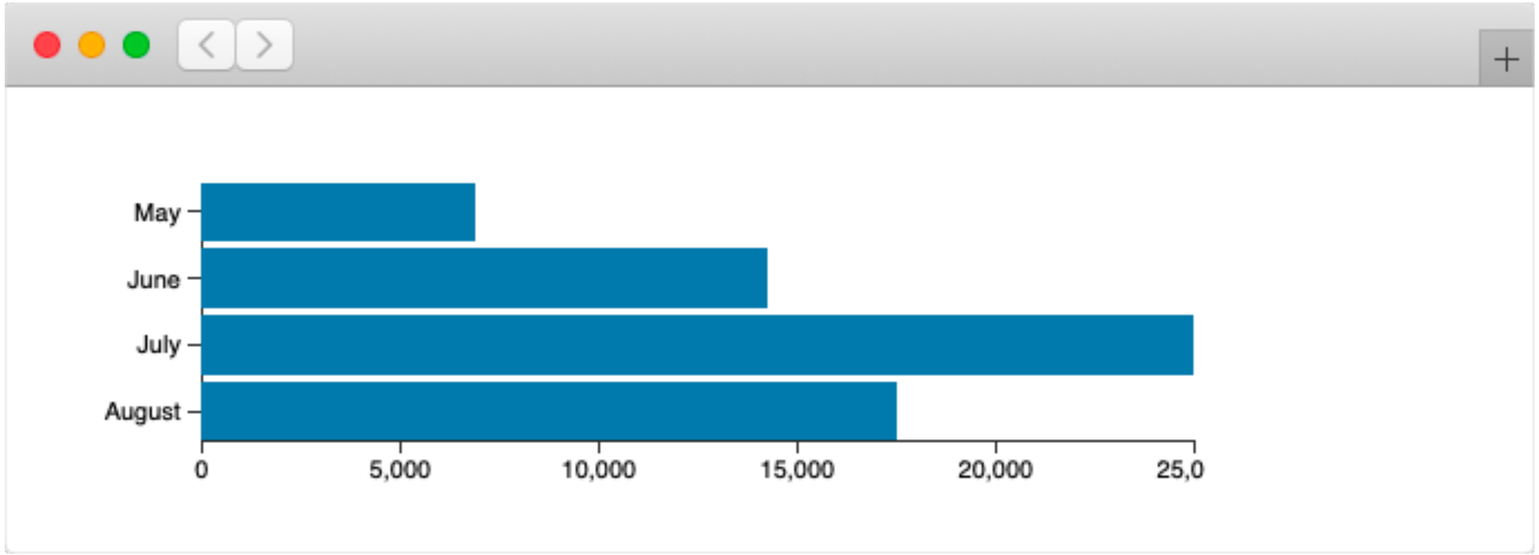
- Draw a chart

```
// Draw the axis (move xAxis to the bottom with 'translate')
  const xAxisGroup = svg.append('g')
    .attr('class', 'axis x-axis')
    .attr('transform', `translate(0, ${height})`)
    .call(xAxis);

  const yAxisGroup = svg.append('g')
    .attr('class', 'axis y-axis')
    .call(yAxis);
```

Making a chart

- Draw a bar chart

```
// Add rectangles
   svg.selectAll('rect')
      .data(data)
      .enter()
     .append('rect')
      .attr('class', 'bar')
      .attr('width', d => xScale(d.sales))
      .attr('height', yScale.bandwidth())
      .attr('y', d => yScale(d.month))
      .attr('x', 0);
```

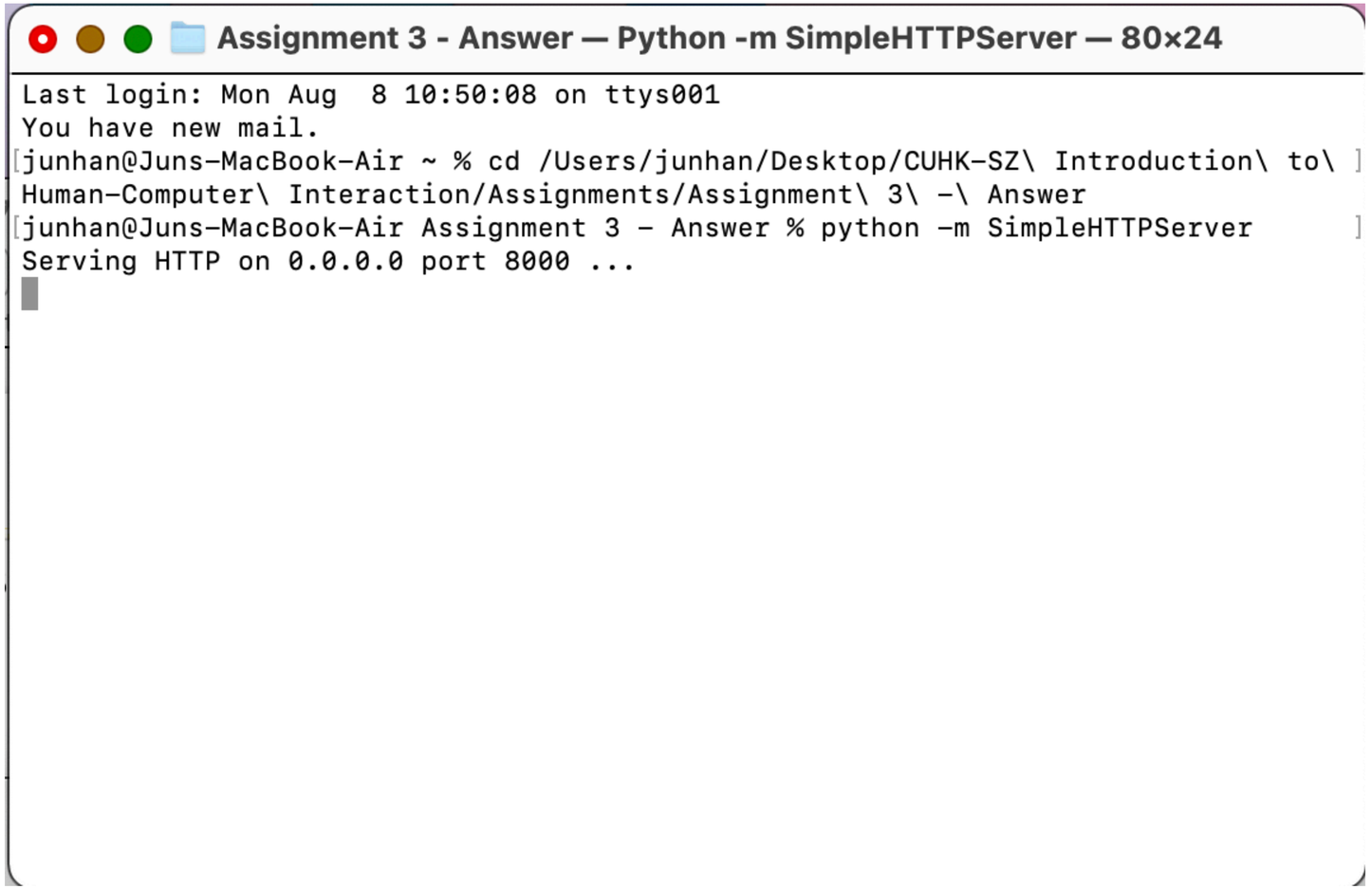


How to access local files

- Build a local server
- Change security permission in browser

How to access local files (using python)

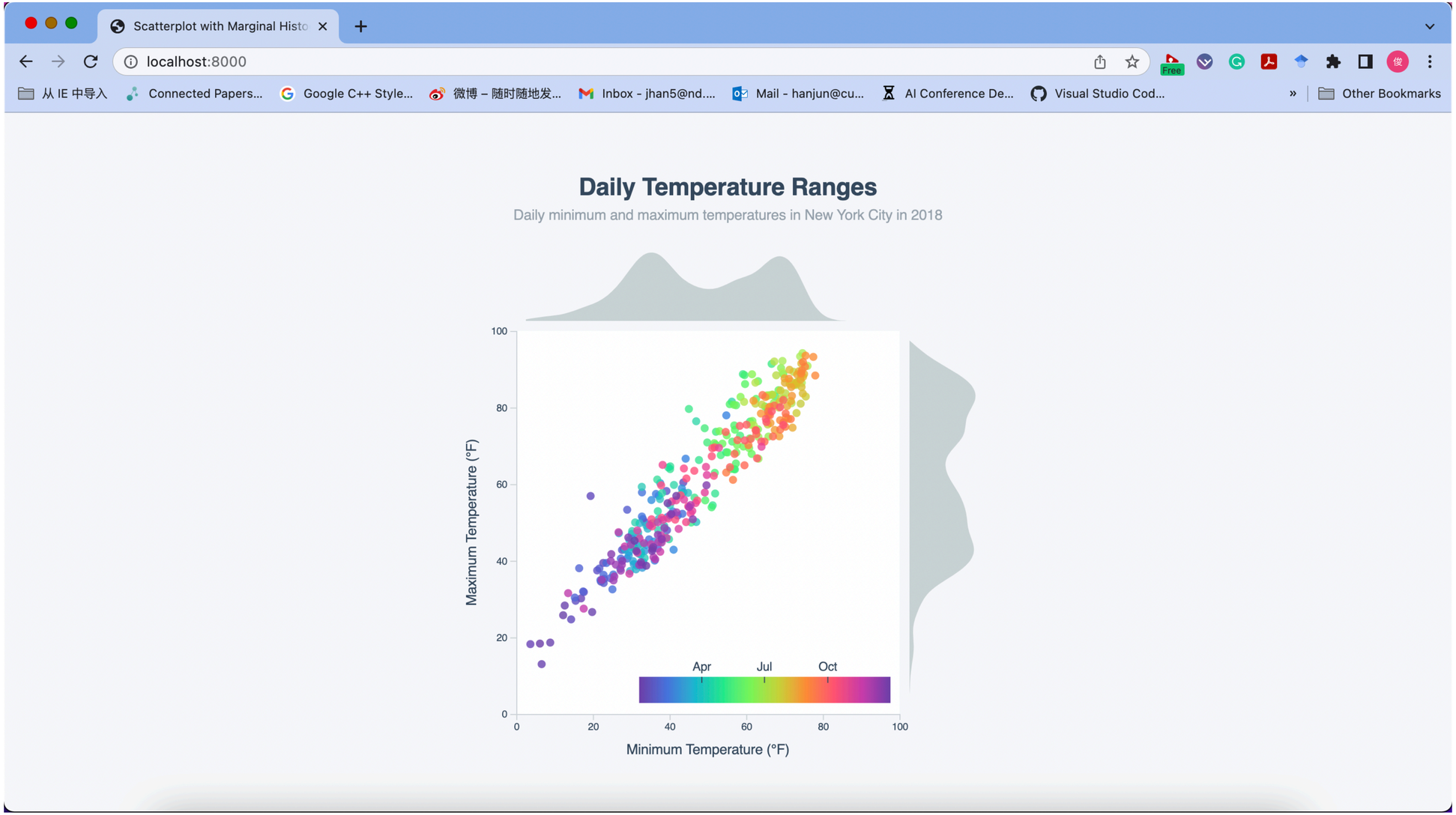
- Go to the folder including html file(s) and type python -m SimpleHTTPServer in terminal



How to access local files (using python)

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

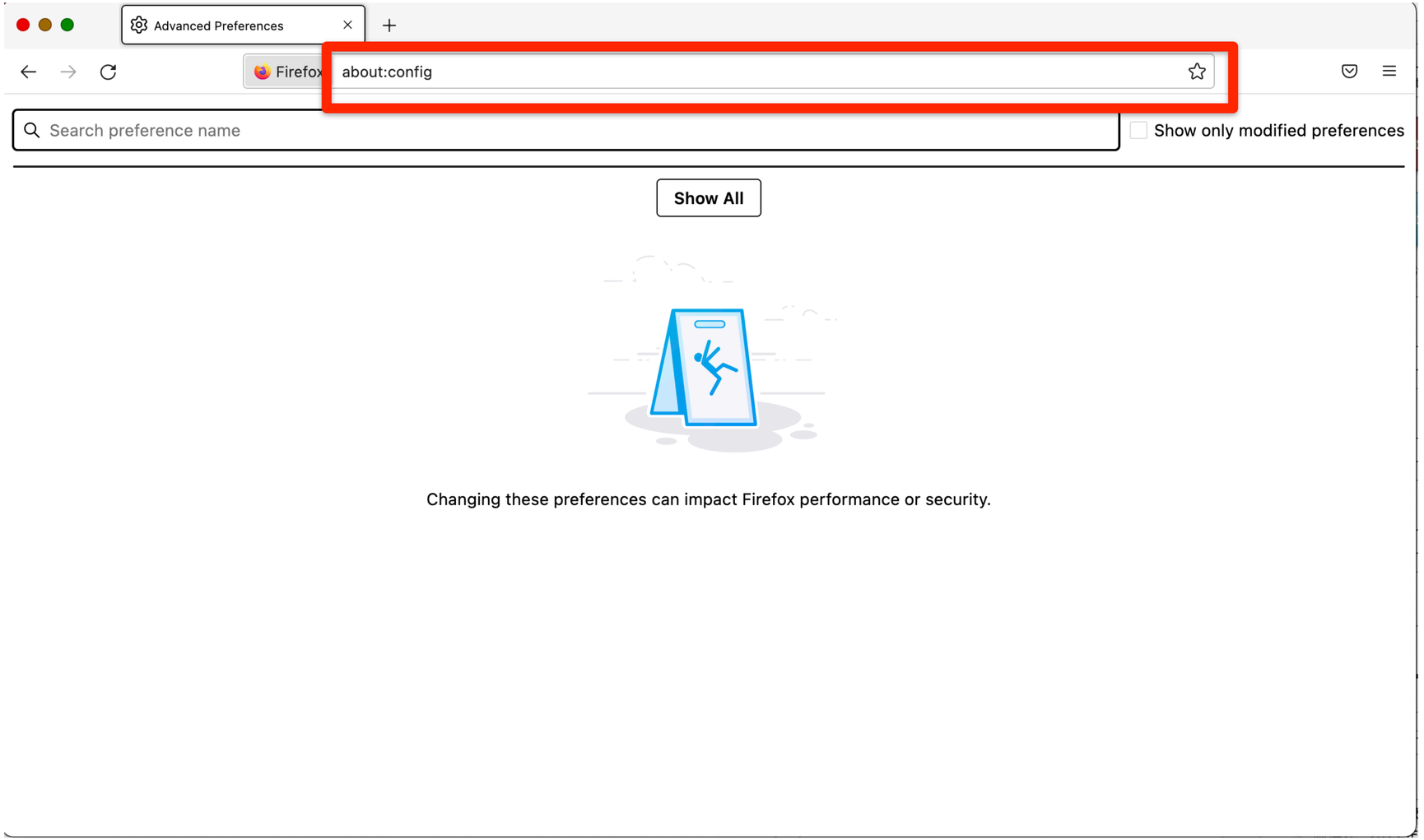
- Open a browser and go to http://localhost:8000



How to access local files (FireFox)

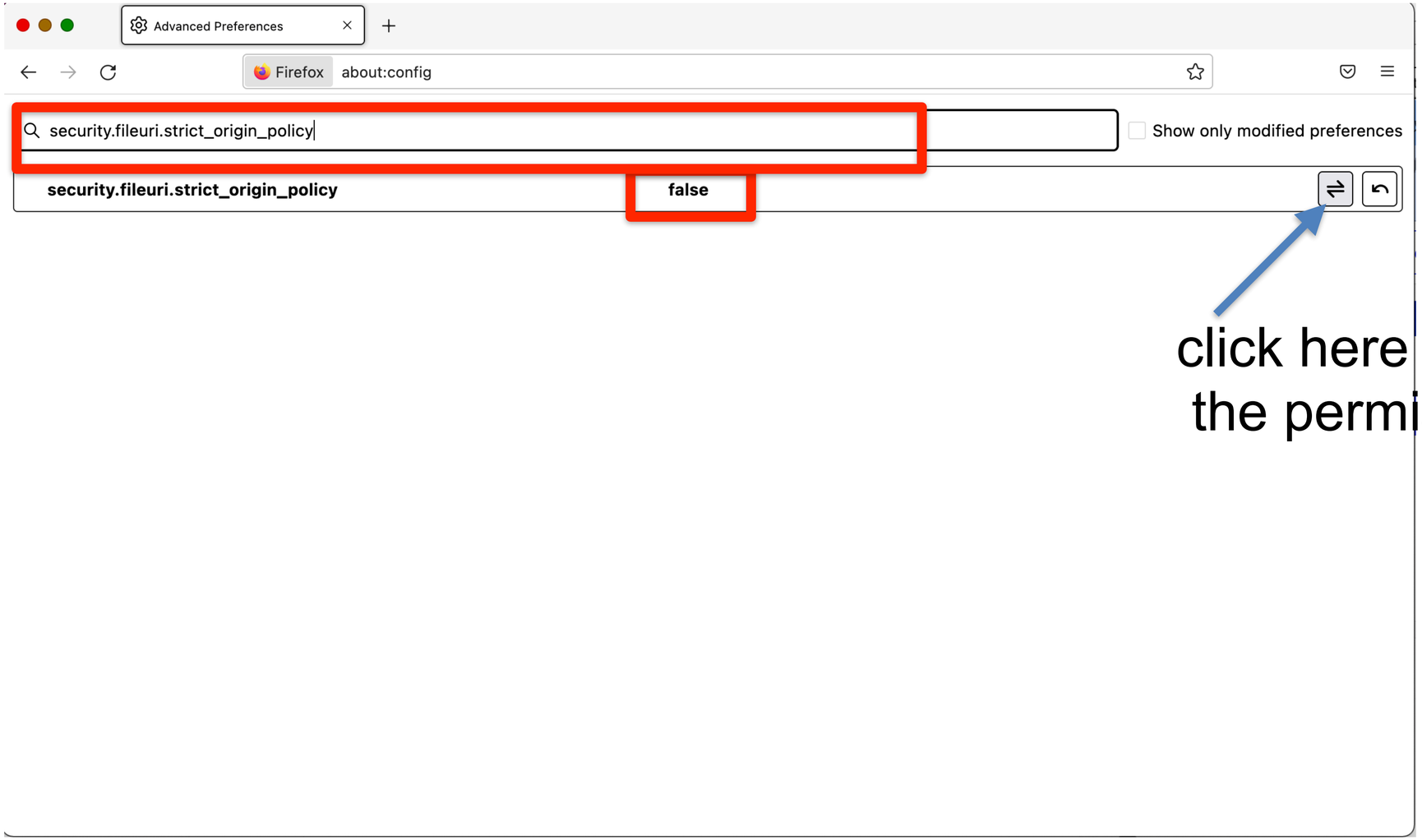
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

- Type about:config

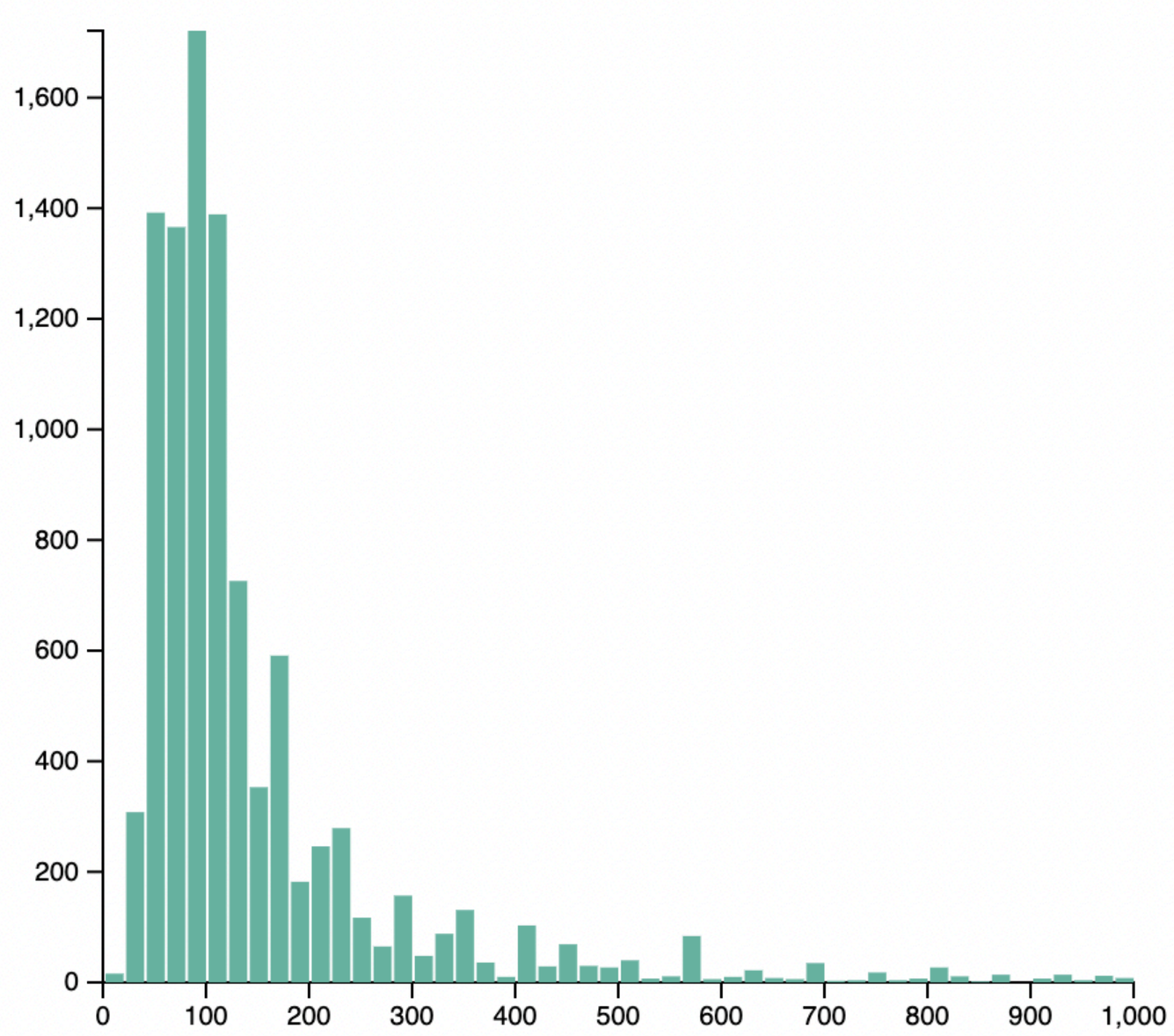


How to access local files (FireFox)

- Search security.fileuri.strict_origin_policy and change true to false



Bar chart

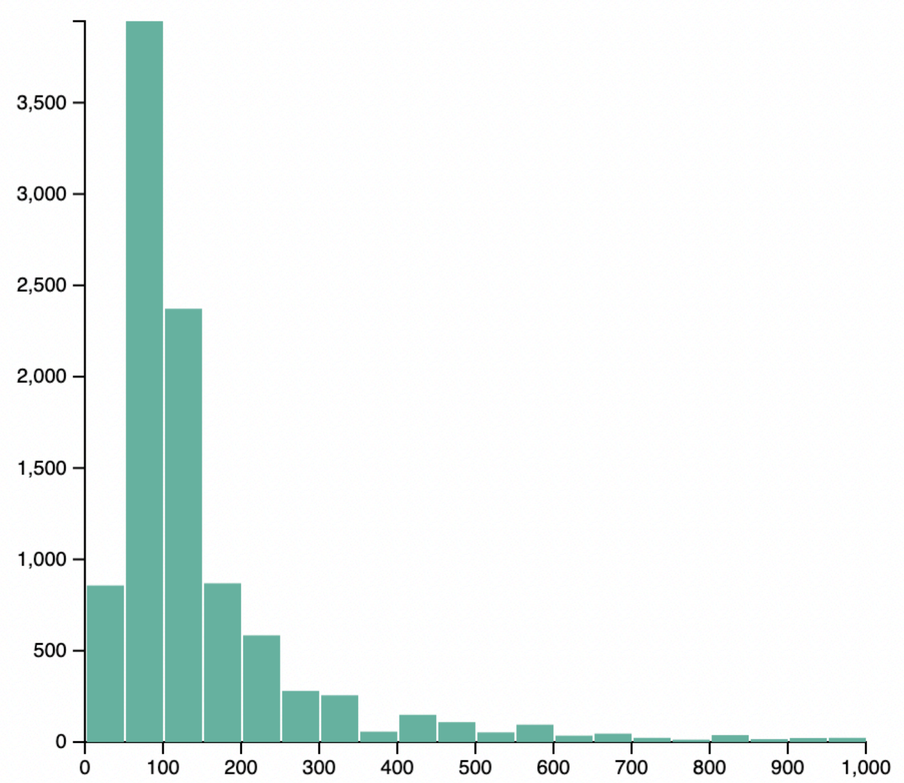

1,600
1,400
1,200
1,000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900
1,000

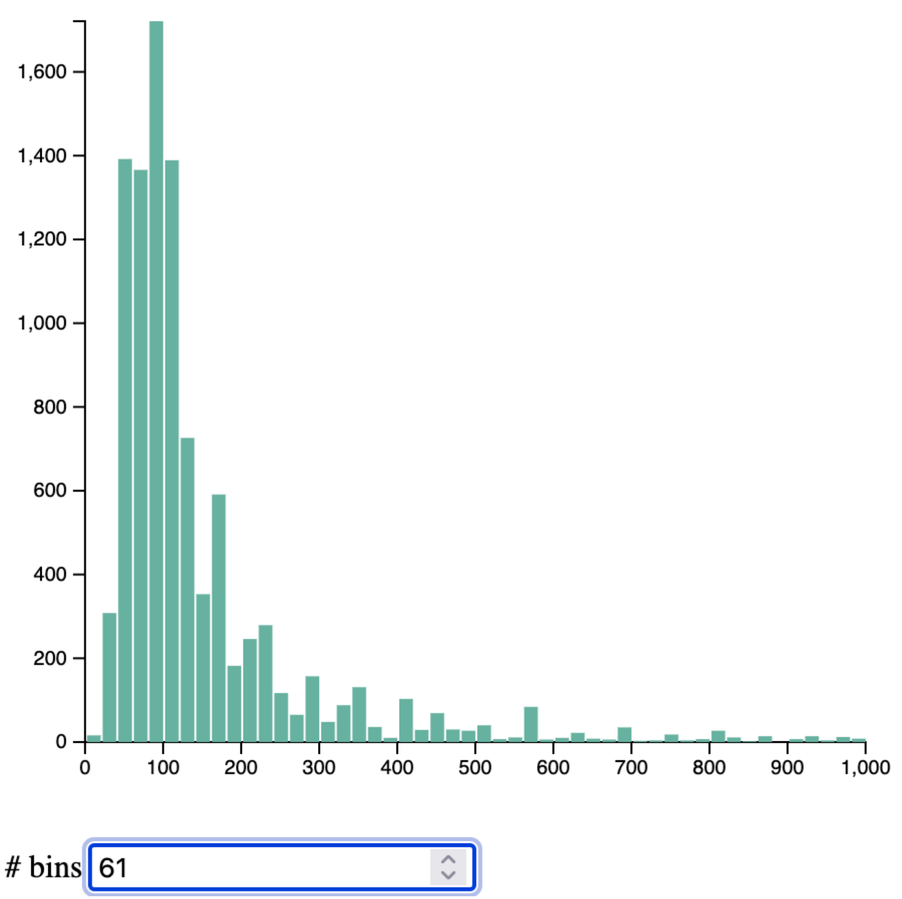
D3 exercise bar chart

- In html, create a div that will be modified by D3 later
- In Javascript, set a svg area and specify the chart size and its margin
- Read data from a .csv file
- This numeric variable is provided to the d3.histogram() function that will compute the binning and returns the coordinates of each bar
- These bars can thus be drawn using a classic .append(“rect”) approach

Bar chart interaction exercise

- Based on the previous example, add a controller that users can choose the number of bins when drawing a bar chart





Making a chart

- Making line and area charts

```
<svg width="500" height="200">
  <path style="fill: none; stroke: blue" d="M0 10 L100 75 L300 90 L350 20"></path>
</svg>
```

Making a chart

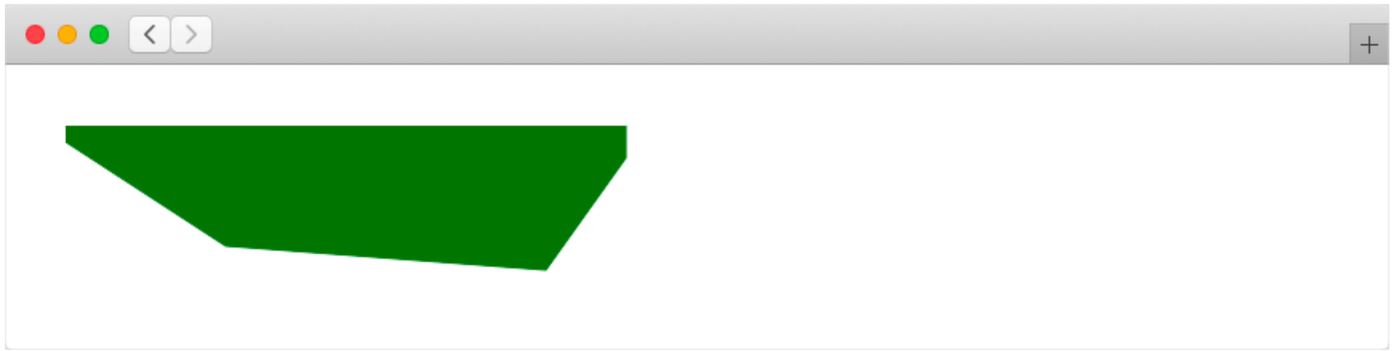
- Making line and area charts

```
const data = [{x: 0, y: 10}, {x: 100, y: 75}, {x: 300, y: 90}, {x: 350, y: 20}]

// Prepare a helper function
const line = d3.line()
    .x(d => d.x)
    .y(d => d.y);

// Add the <path> to the <svg> container using the helper function
d3.select('svg').append('path')
    .attr('d', line(data))
    .attr('stroke', 'red')
    .attr('fill', 'none');
```

Making a chart

- Making line and area charts

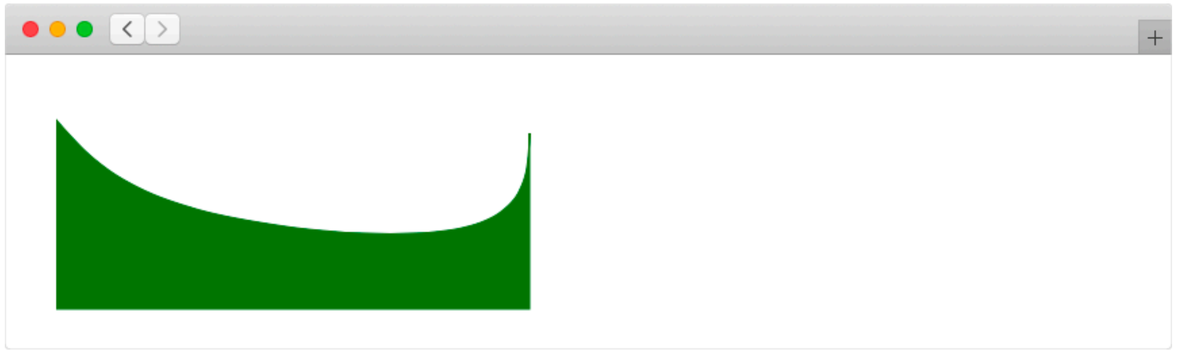
```
const data = [{x: 0, y: 10}, {x: 100, y: 75}, {x: 300, y: 90}, {x: 350, y: 20}]

// Prepare a helper function
const area = d3.area()
  .x(d => d.x)      // Same x-position
  .y1(d => d.y)     // Top line y-position
  .y0(0)            // Bottom line y-position

// Add the area path using this helper function
d3.select('svg').append('path')
  .attr('d', area(data))
  .attr('stroke', 'green')
  .attr('fill', 'green');
```



Making a chart

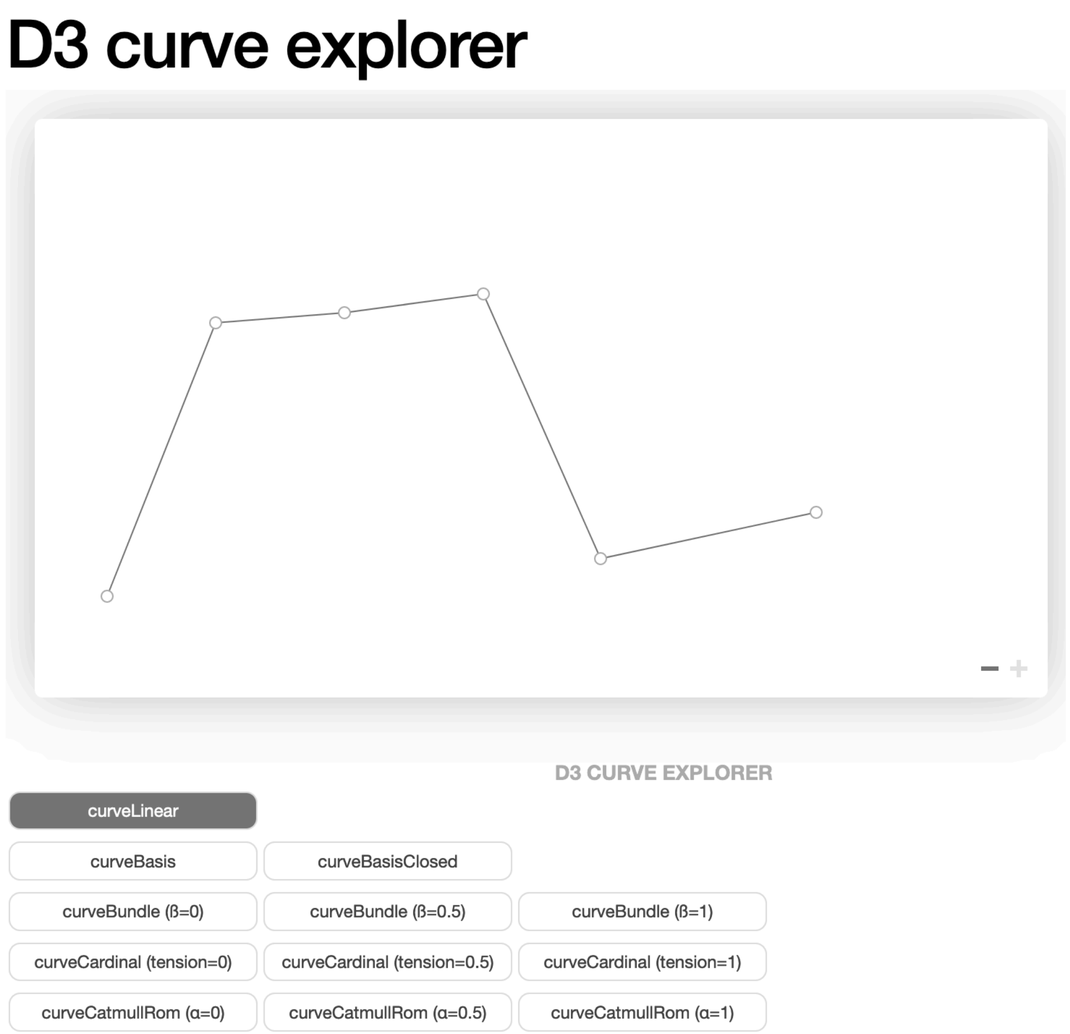
- Making line and area charts

```
const data = [{x: 0, y: 10}, {x: 100, y: 75}, {x: 300, y: 90}, {x: 350, y: 20}]

// Prepare a helper function
const area = d3.area()
  .x(d => d.x)      // Same x-position
  .y1(d => d.y)     // Top line y-position
  .y0(150)          // Bottom line y-position
  .curve(d3.curveNatural)

// Add the area path using this helper function
d3.select('svg').append('path')
  .attr('d', area(data))
  .attr('stroke', 'green')
  .attr('fill', 'green');
```



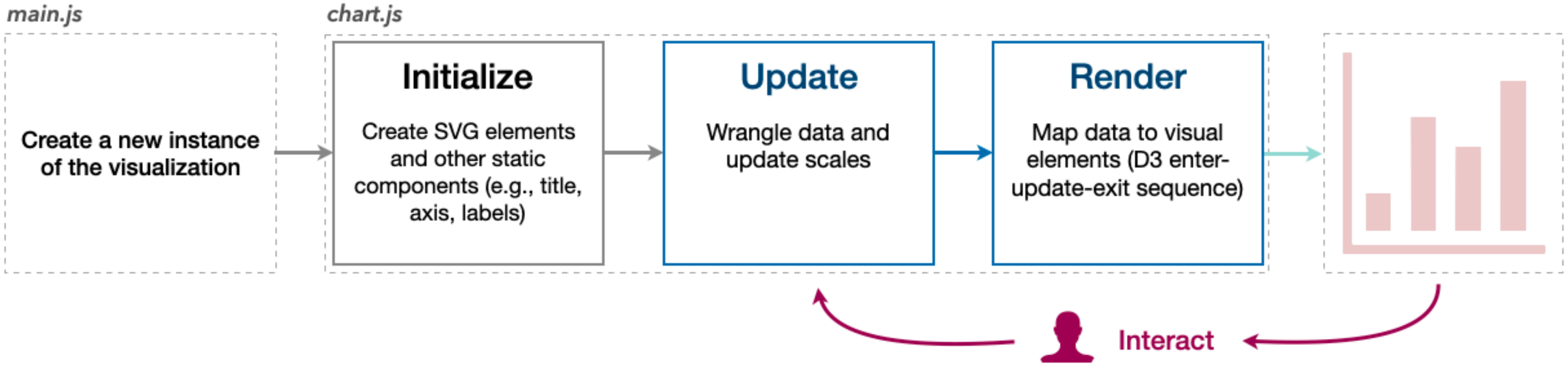
Making a chart

- Compare different curve interpolation types interactively



Making a chart

- Reuse D3 component



Making a chart

- Introduce classes to better support object-oriented programming that you should use

```
// ES6 Class
class BarChart {
  constructor(_config, _data) {
    this.config = {
      parentElement: _config.parentElement,
      containerWidth: _config.containerWidth || 500,
      containerHeight: _config.containerHeight || 140,
      margin: { top: 10, bottom: 30, right: 10, left: 30 }
    }
    this.data = _data;
    // Call a class function
    this.initVis();
  }

  initVis() {
        ...
  }
  ...
}
```

```
// Create an instance (for example in main.js)
let barchart = new BarChart({
        'parentElement': '#bar-chart-container',
        'containerHeight': 400
});
```

```
initVis() {
    let vis = this;

    vis.svg = d3.select(vis.config.parentElement).append('svg');
    vis.chart = vis.svg.append('g')
        .attr('transform', `translate(${vis.config.margin.left},${vis.config.margin.top})`);
    ...
}
```

Making a chart

Application
main.js

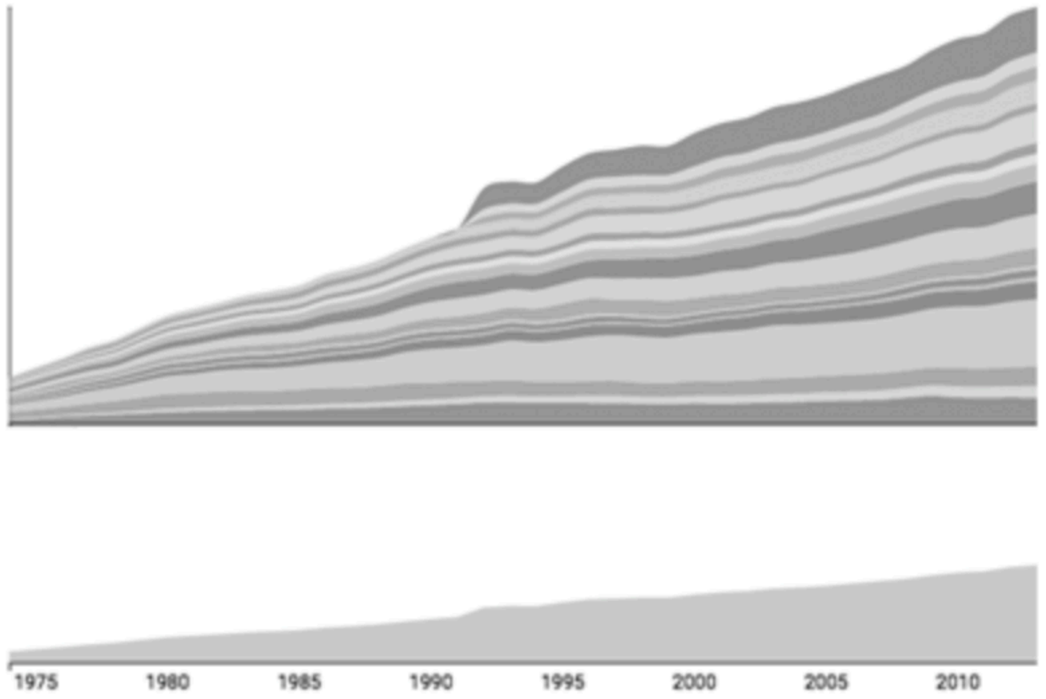
- Load data
- Initialize visualizations
- Listen to global events and update visualizations

stackedAreaChart.js

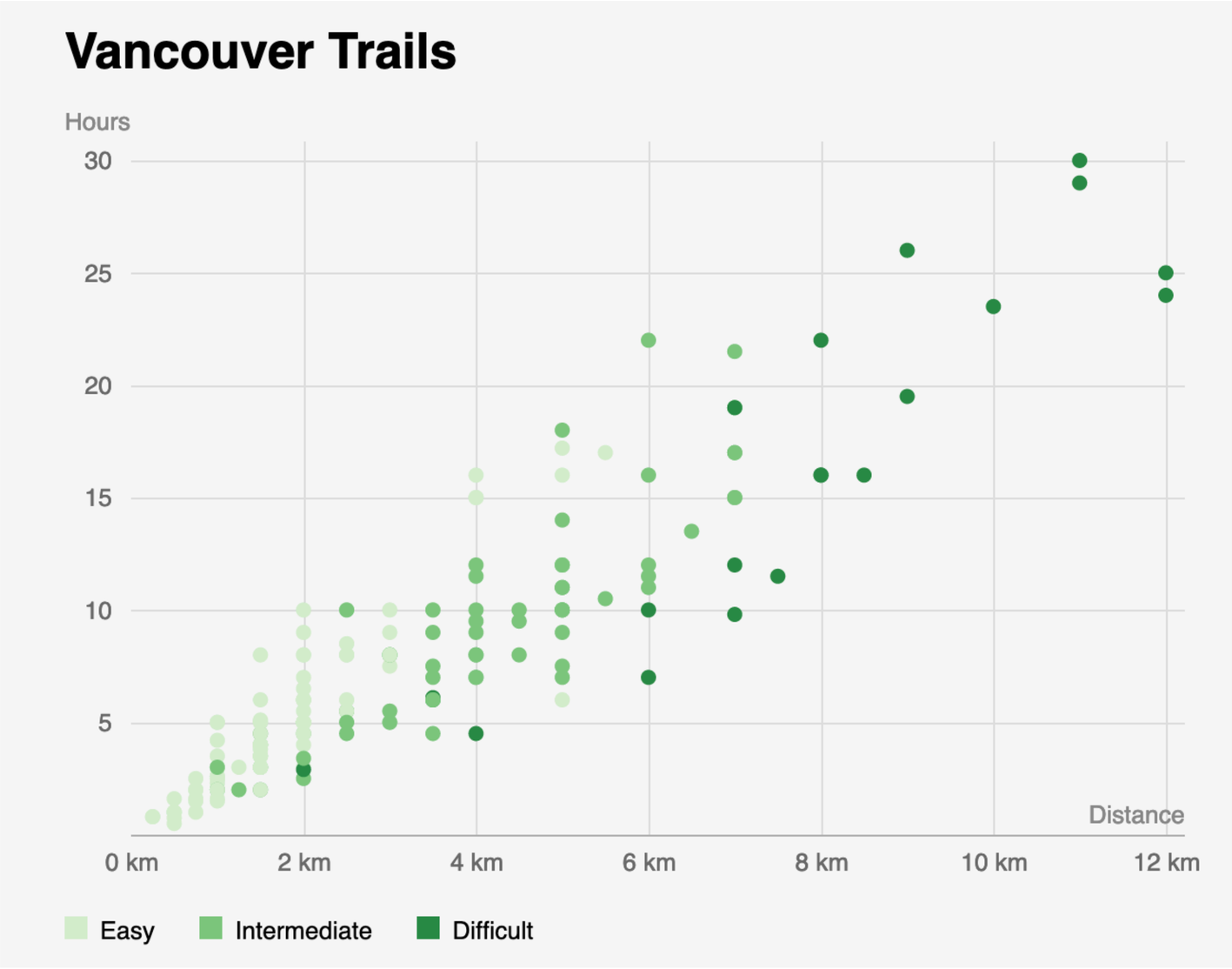
- `constructor`
- `initVis()`
- `updateVis()`
- `renderVis()`

timeline.js

- `constructor`
- `initVis()`
- `updateVis()`
- `renderVis()`

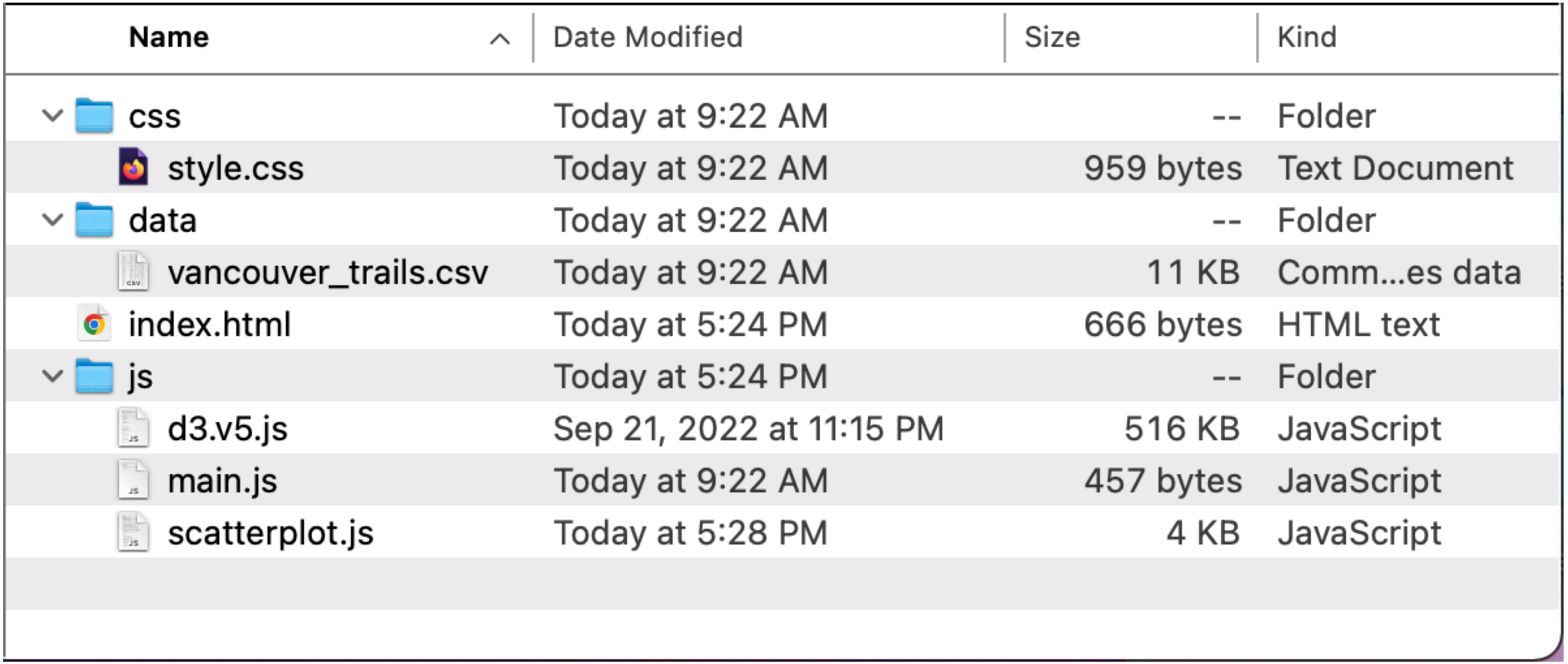


Scatter chart



Scatter chart

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen



Name	Date Modified	Size	Kind
css	Today at 9:22 AM	--	Folder
style.css	Today at 9:22 AM	959 bytes	Text Document
data	Today at 9:22 AM	--	Folder
vancouver_trails.csv	Today at 9:22 AM	11 KB	Comm...es data
index.html	Today at 5:24 PM	666 bytes	HTML text
js	Today at 5:24 PM	--	Folder
d3.v5.js	Sep 21, 2022 at 11:15 PM	516 KB	JavaScript
main.js	Today at 9:22 AM	457 bytes	JavaScript
scatterplot.js	Today at 5:28 PM	4 KB	JavaScript

Scatter chart

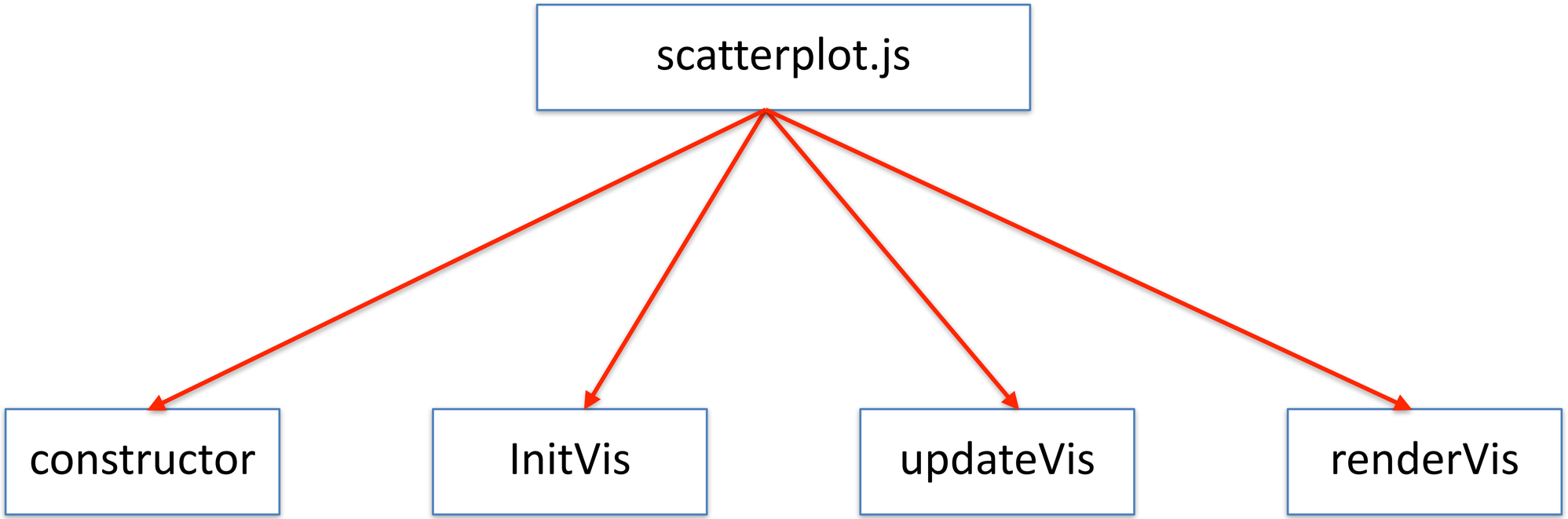

```
<!DOCTYPE html>
<html>
<head>
    <title>D3 Static Scatter Plot</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <h1>Vancouver Trails</h1>

    <!-- Empty SVG drawing area -->
    <svg id="scatterplot"></svg>

    <ul class="legend">
        <li><span class="legend-e easy"></span> Easy</li>
        <li><span class="legend-e intermediate"></span> Intermediate</li>
        <li><span class="legend-e difficult"></span> Difficult</li>
    </ul>

    <!-- D3 bundle -->
    <script src="js/d3.v5.js"></script>

    <!-- Our JS code -->
    <script src="js/scatterplot.js"></script>
    <script src="js/main.js"></script>
</body>
</html>
```

Scatter chart

main.js

```
/**
 * Load data from CSV file asynchronously and render scatter plot
 */
d3.csv('data/vancouver_trails.csv')
  .then(data => {
    // Convert strings to numbers
    data.forEach(d => {
      d.time = +d.time;
      d.distance = +d.distance;
    });

    // Initialize chart
    const scatterplot = new Scatterplot({ parentElement: '#scatterplot'}, data);

    // Show chart
    scatterplot.updateVis();
  })
  .catch(error => console.error(error));
```

Scatter chart



Configure attributes

Initialize necessary variables

Update chart when interaction occur

Visualize data

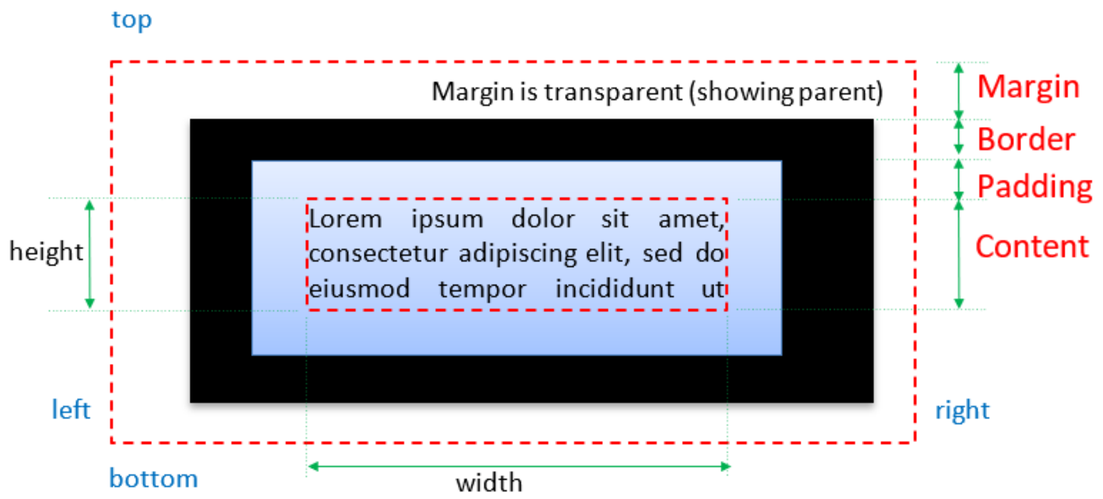
Scatter chart

scatterplot.js

constructor

Configure attributes

```
constructor(_config, _data) {
  // Configuration object with defaults
  // Important: depending on your vis and the type of interactivity you need
  // you might want to use getter and setter methods for individual attributes
  this.config = {
    parentElement: _config.parentElement,
    containerWidth: _config.containerWidth || 600,
    containerHeight: _config.containerHeight || 400,
    margin: _config.margin || {top: 25, right: 20, bottom: 20, left: 35}
  }
  this.data = _data;
  this.initVis();
}
```



CSS Box Model

Scatter chart

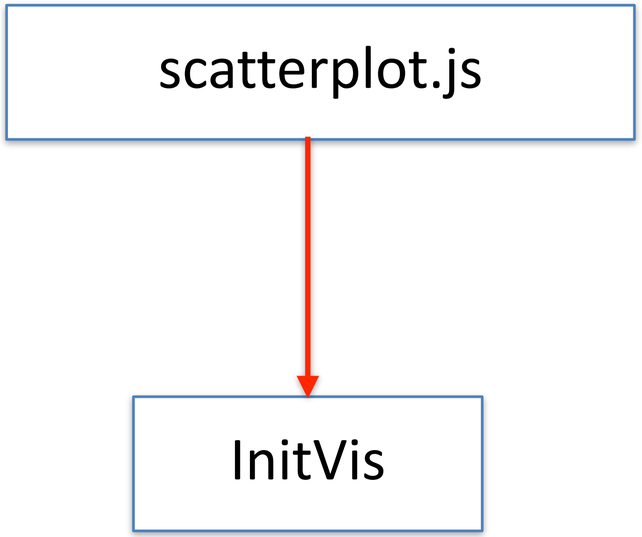
scatterplot.js

InitVis

Initialize necessary variables

```
initVis() {
  // We recommend avoiding simply using the this keyword within complex class code
  // involving SVG elements because the scope of this will change and it will cause
  // undesirable side-effects. Instead, we recommend creating another variable at
  // the start of each function to store the this-accessor
  let vis = this;

  // Calculate inner chart size. Margin specifies the space around the actual chart.
  // You need to adjust the margin config depending on the types of axis tick labels
  // and the position of axis titles (margin convetion: https://bl.ocks.org/mbostock/3019563)
  vis.width = vis.config.containerWidth - vis.config.margin.left - vis.config.margin.right;
  vis.height = vis.config.containerHeight - vis.config.margin.top - vis.config.margin.bottom;

  // Initialize scales
  vis.colorScale = d3.scaleOrdinal()
      .range(['#d3eecd', '#7bc77e', '#2a8d46']) // light green to dark green
      .domain(['Easy','Intermediate','Difficult']);

  // Create Scales

  // To DO

  // Initialize axes

  // To DO

  // Define size of SVG drawing area
  vis.svg = d3.select(vis.config.parentElement)
      .attr('width', vis.config.containerWidth)
      .attr('height', vis.config.containerHeight);

  // Append group element that will contain our actual chart
  // and position it according to the given margin config
  vis.chart = vis.svg.append('g')
      .attr('transform', `translate(${vis.config.margin.left},${vis.config.margin.top})`);

  // Append empty x-axis group and move it to the bottom of the chart
  vis.xAxisG = vis.chart.append('g')
      .attr('class', 'axis x-axis')
      .attr('transform', `translate(0,${vis.height})`);

  // Append y-axis group
  vis.yAxisG = vis.chart.append('g')
      .attr('class', 'axis y-axis');

  // Append both axis titles
  vis.chart.append('text')
      .attr('class', 'axis-title')
      .attr('y', vis.height - 15)
      .attr('x', vis.width + 10)
      .attr('dy', '.71em')
      .style('text-anchor', 'end')
      .text('Distance');

  vis.svg.append('text')
      .attr('class', 'axis-title')
      .attr('x', 0)
      .attr('y', 0)
      .attr('dy', '.71em')
      .text('Hours');
}
```

Scatter chart



Initialize necessary variables

```
initVis() {
  // We recommend avoiding simply using the this keyword within complex class code
  // involving SVG elements because the scope of this will change and it will cause
  // undesirable side-effects. Instead, we recommend creating another variable at
  // the start of each function to store the this-accessor
  let vis = this;

  // Calculate inner chart size. Margin specifies the space around the actual chart.
  // You need to adjust the margin config depending on the types of axis tick labels
  // and the position of axis titles (margin convetion: https://bl.ocks.org/mbostock/3019563)
  vis.width = vis.config.containerWidth - vis.config.margin.left - vis.config.margin.right;
  vis.height = vis.config.containerHeight - vis.config.margin.top - vis.config.margin.bottom;

  // Initialize scales
  vis.colorScale = d3.scaleOrdinal()
      .range(['#d3eecd', '#7bc77e', '#2a8d46']) // light green to dark green
      .domain(['Easy','Intermediate','Difficult']);

  // Create Scales

  // To DO
```

Two scales: x, and y

d3.scaleLinear

Set domain and range

Scatter chart

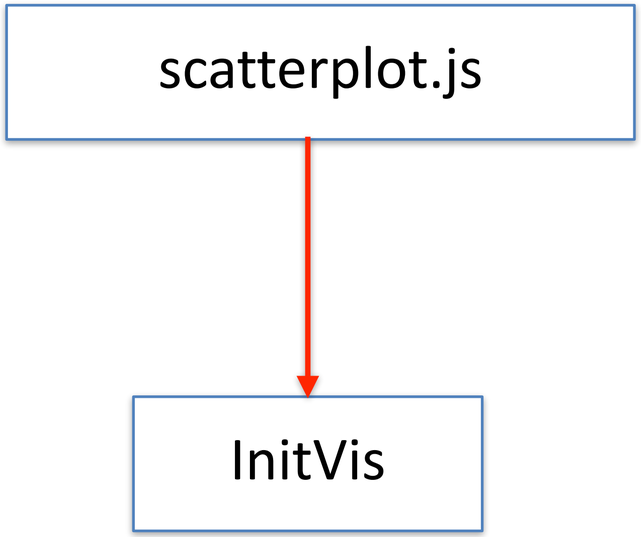
scatterplot.js

InitVis

Initialize necessary variables

```
*/
initVis() {
  // We recommend avoiding simply using the this keyword within complex class code
  // involving SVG elements because the scope of this will change and it will cause
  // undesirable side-effects. Instead, we recommend creating another variable at
  // the start of each function to store the this-accessor
  let vis = this;

  // Calculate inner chart size. Margin specifies the space around the actual chart.
  // You need to adjust the margin config depending on the types of axis tick labels
  // and the position of axis titles (margin convetion: https://bl.ocks.org/mbostock/3019563)
  vis.width = vis.config.containerWidth - vis.config.margin.left - vis.config.margin.right;
  vis.height = vis.config.containerHeight - vis.config.margin.top - vis.config.margin.bottom;

  // Initialize scales
  vis.colorScale = d3.scaleOrdinal()
      .range(['#d3eecd', '#7bc77e', '#2a8d46']) // light green to dark green
      .domain(['Easy','Intermediate','Difficult']);

  // Create Scales

  // To DO

  // Initialize axes

  // To DO
```

Scatter chart



Initialize necessary variables

```
vis.xScale = d3.scaleLinear()
    .range([0, vis.width]);

vis.yScale = d3.scaleLinear()
    .range([vis.height, 0]);

// Initialize axes
vis.xAxis = d3.axisBottom(vis.xScale)
    .ticks(6)
    .tickSize(-vis.height - 10)
    .tickPadding(10)
    .tickFormat(d => d + ' km');

vis.yAxis = d3.axisLeft(vis.yScale)
    .ticks(6)
    .tickSize(-vis.width - 10)
    .tickPadding(10);
```

Scatter chart

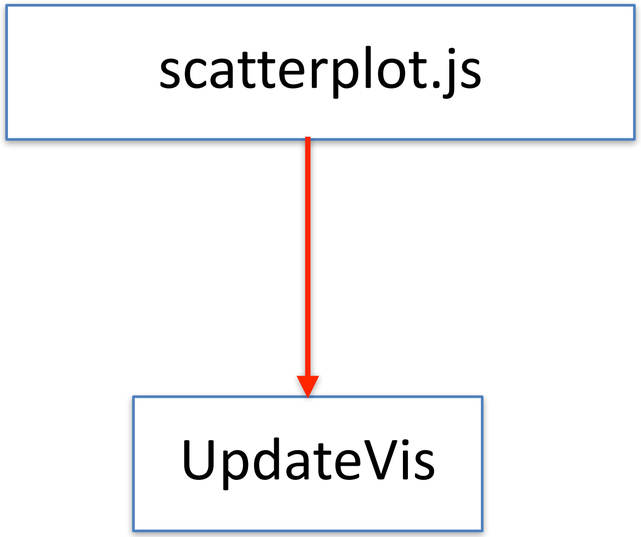
scatterplot.js

UpdateVis

Update chart
when interaction occur

```
 * This function contains all the code to prepare the data before we render it.
 * In some cases, you may not need this function but when you create more complex visualizations
 * you will probably want to organize your code in multiple functions.
 */
updateVis() {
  let vis = this;

  // Specificy accessor functions
  vis.colorValue = d => d.difficulty;
  vis.xValue = d => d.time;
  vis.yValue = d => d.distance;

  // Set the scale input domains

  // To DO

  vis.renderVis();
}
```

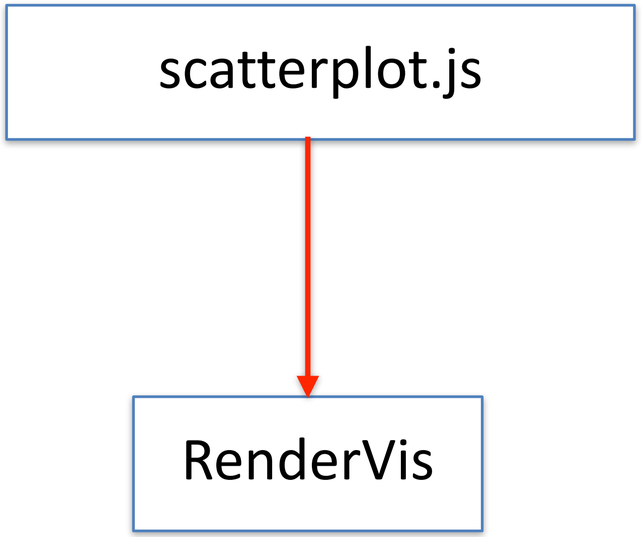
Scatter chart



Update chart
when interaction occur

```
// Set the scale input domains
vis.xScale.domain([0, d3.max(vis.data, vis.xValue)]);
vis.yScale.domain([0, d3.max(vis.data, vis.yValue)]);
```

Scatter chart

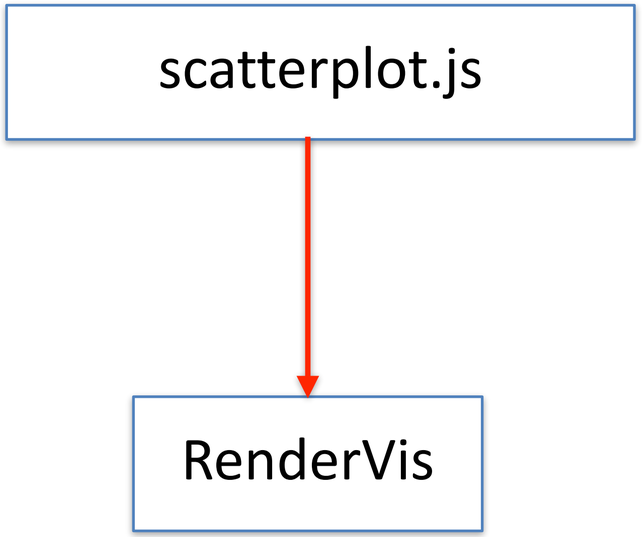


Visualize data

```
/**
 * This function contains the D3 code for binding data to visual elements.
 * We call this function every time the data or configurations change.
 */
renderVis() {
  let vis = this;

  // Add circles
  // To DO

  // Update the axes/gridlines
  // We use the second .call() to remove the axis and just show gridlines

  // To DO
}
```

```
svg.selectAll('rect')
    .data(numericData)
    .enter()
  .append('rect')
    .attr('fill', 'red')
    .attr('width', 50)
    .attr('height', 50)
    .attr('y', 0)
    .attr('x', (d, index) => index * 60);
```

Scatter chart



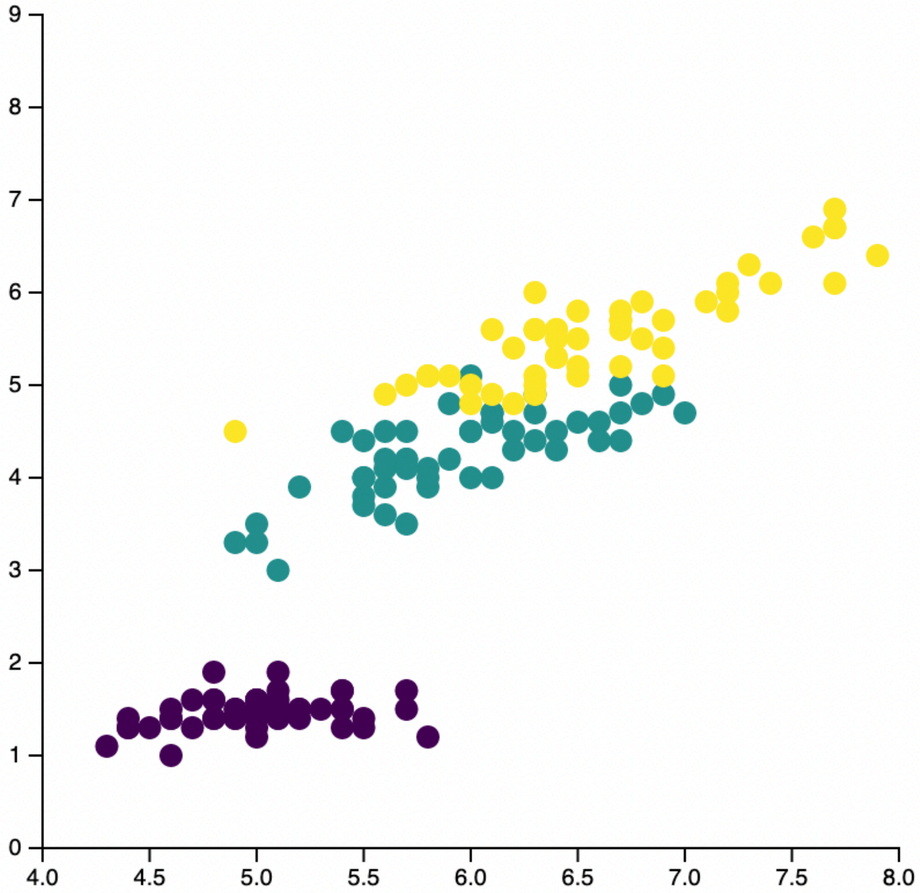
Visualize data

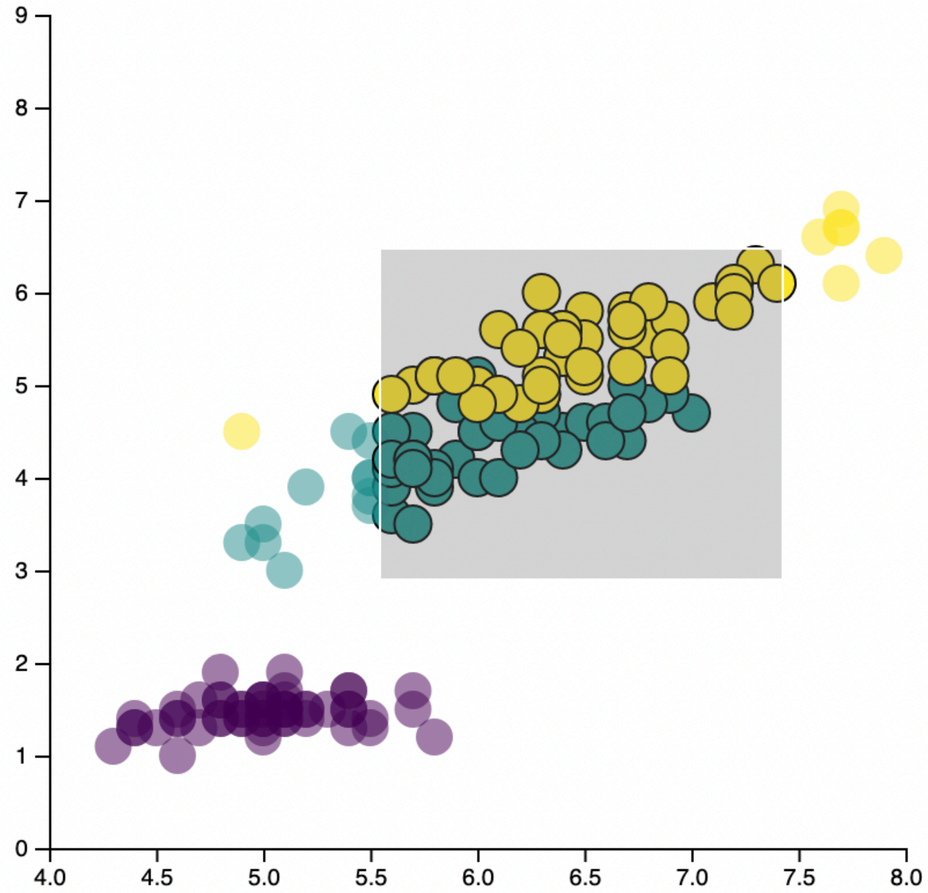
```
// Add circles
vis.chart.selectAll('circle')
    .data(vis.data)
    .enter()
  .append('circle')
    .attr('class', 'point')
    .attr('r', 4)
    .attr('cy', d => vis.yScale(vis.yValue(d)))
    .attr('cx', d => vis.xScale(vis.xValue(d)))
    .attr('fill', d => vis.colorScale(vis.colorValue(d)));

// Update the axes/gridlines
// We use the second .call() to remove the axis and just show gridlines
vis.xAxisG
    .call(vis.xAxis)
    .call(g => g.select('.domain').remove());

vis.yAxisG
    .call(vis.yAxis)
    .call(g => g.select('.domain').remove())
```

Scatter chart interaction exercise

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

- Based on the previous example, add a mouse brush function that users can brush dots in the scatter and highlight the selected dots





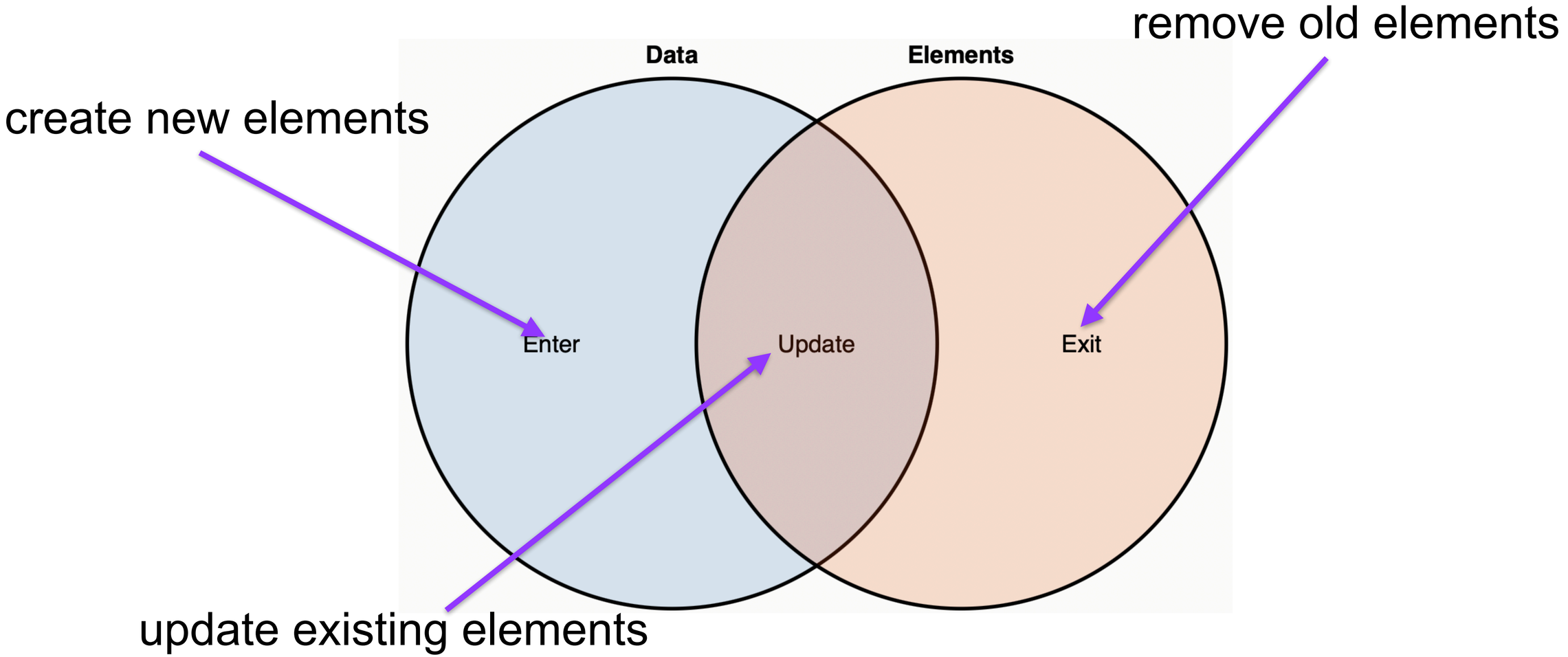
Outline

- Introduction to D3
- Making a chart
- Data joins and basic interactivity
- Multiple views and advanced interactivity
- Advanced concepts

Data joins and basic interactivity

- Instead of removing and redrawing visualizations each time new data arrives, we want to update only affected components to improve loading times and create smooth transitions
- Enter, update, and exit
  - Enter:What happens to new data values without existing, associated DOM elements?
  - Update: What happens to existing elements which have changed?
  - Exit: What happens to existing DOM elements which are not associated with data anymore?

Data joins and basic interactivity



Data joins and basic interactivity

- Join() shortcut for the enter-update-exit pattern
  - Alternatively use the join() method which is simpler and more convenient for many cases. Instead of specifying enter, update, and exit operations separately, join() handles all three stages automatically. New elements will be added, existing elements will be updated, and obsolete elements will be removed

```
function updateChart(data) {
  svg.selectAll('circle')
      .data(data)
    .join('circle')
      .attr('fill', '#707086')
      .attr('r', d => d)
      .attr('cx', (d,index) => (index * 80) + 50)
      .attr('cy', 80);
}
```

Data joins and basic interactivity

- Handle user input
  - Bind an event listener to any DOM element using d3.select().on()

Data joins and basic interactivity

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

```
<!-- ... -->
<body>
    <!-- HTML form -->
    <div>
        <label for="radius-slider">Radius: <span id="radius-value">60</span></label>
        <input type="range" min="1" value="60" max="80" id="radius-slider">
    </div>

    <!-- Empty SVG drawing area -->
    <svg id="chart" width="200" height="200"></svg>

    <script src="js/d3.v6.min.js"></script>
    <script src="js/main.js"></script>
</body>
</html>
```

```
const svg = d3.select('svg');

// Show circle with initial radius of 60px
const circle = svg.append('circle')
    .attr('cx', 100)
    .attr('cy', 100)
    .attr('fill', 'none')
    .attr('stroke', 'green')
    .attr('r', 60);

function updateCircle(radius) {
  circle.attr('r', radius);
}
```

```
d3.select('#radius-slider').on('input', function() {
  // Update visualization
  updateCircle(parseInt(this.value));

  // Update label
  d3.select('#radius-value').text(this.value);
});
```

Data joins and basic interactivity

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

- Interaction process
  - Add global event listeners (e.g., checkboxes, sliders, …)
  - Update configurations or data
  - Update the visualization accordingly
  - Add chart-specific events

```
svg.selectAll('circle')
      .data(data)
    .join('circle')
    .attr('fill', 'green')
    .attr('r', 4)
    .attr('cx', d => vis.xScale(d.x))
    .attr('cy', d => vis.yScale(d.y))
    .on('mouseover', d => console.log('debug, show tooltip, etc.'))
```

- Animated transitions
  - Apply transition() method that creates smooth, animated transitions between states
  - Default time span is 250 milliseconds
  - Use duration() to specify the value
  - Use delay() to delay a transition

```
d3.selectAll('circle')
   .transition()
   .duration(3000)
   .attr('fill', 'blue');
```

- Pros of animated transitions
  - Transitions show what is happening between states and add a sense of continuity to your visualization
  - Animations can draw the user's attention to specific elements or aspects
  - Animations can provide the user with interactive feedback

- Cons of animated transitions
  - Too many transitions will confuse the user (e.g., overused PowerPoint effects)
  - If the transition is not continuous, animations look strange and can even be deceiving based on the interpolation used
  - Animation across many states is the least effective use case for data analysis tasks. In this case, use a static comparison of several charts/images (e.g., small multiples) instead of creating video-like animations

Data joins and basic interactivity

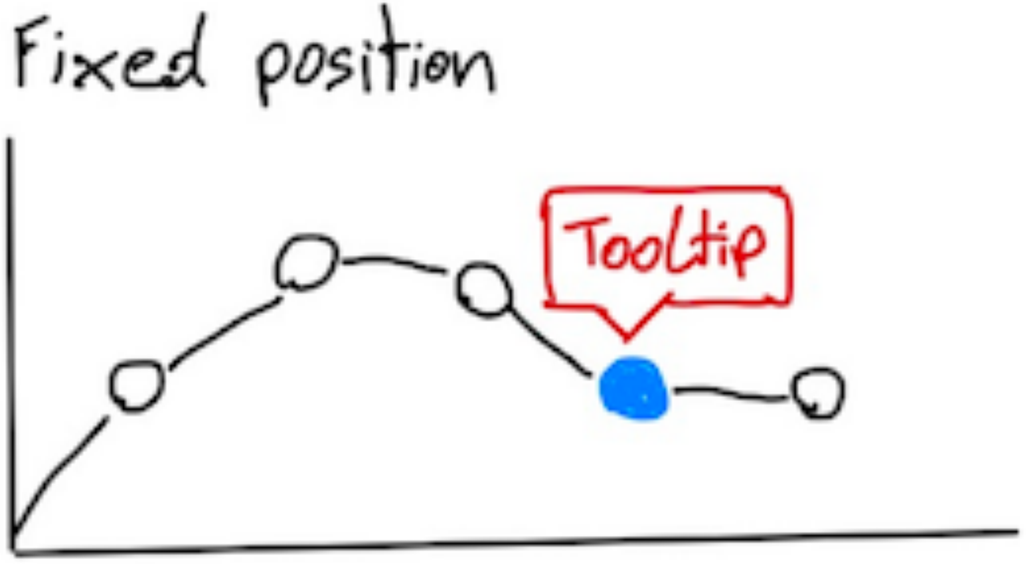
- Tooltips
  - Reveal more details and information to the users

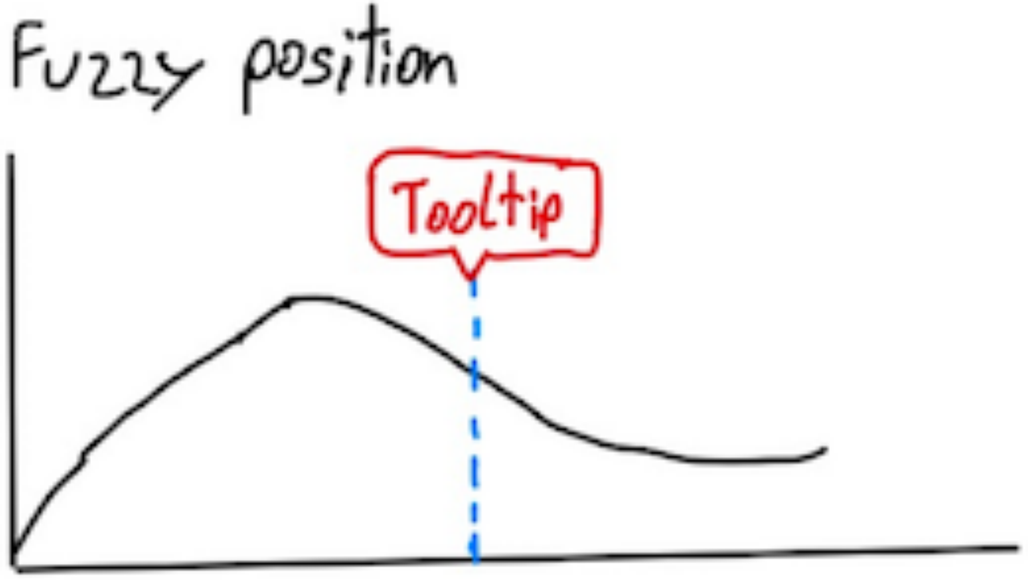
```
<div id="tooltip"></div>
```

```
#tooltip {
  position: absolute;
  display: none;
  /* ... other tooltip styles ... */
}
```

```
myMarks
    .on('mouseover', (event,d) => {
      d3.select('#tooltip')
        .style('display', 'block')
        // Format number with million and thousand separator
        .html(`<div class="tooltip-label">Population</div>${d3.format(',')(d.population)}`);
    })
    .on('mousemove', (event) => {
      d3.select('#tooltip')
        .style('left', (event.pageX + vis.config.tooltipPadding) + 'px')
        .style('top', (event.pageY + vis.config.tooltipPadding) + 'px')
    })
    .on('mouseleave', () => {
      d3.select('#tooltip').style('display', 'none');
    });
```

- Tooltips for path elements





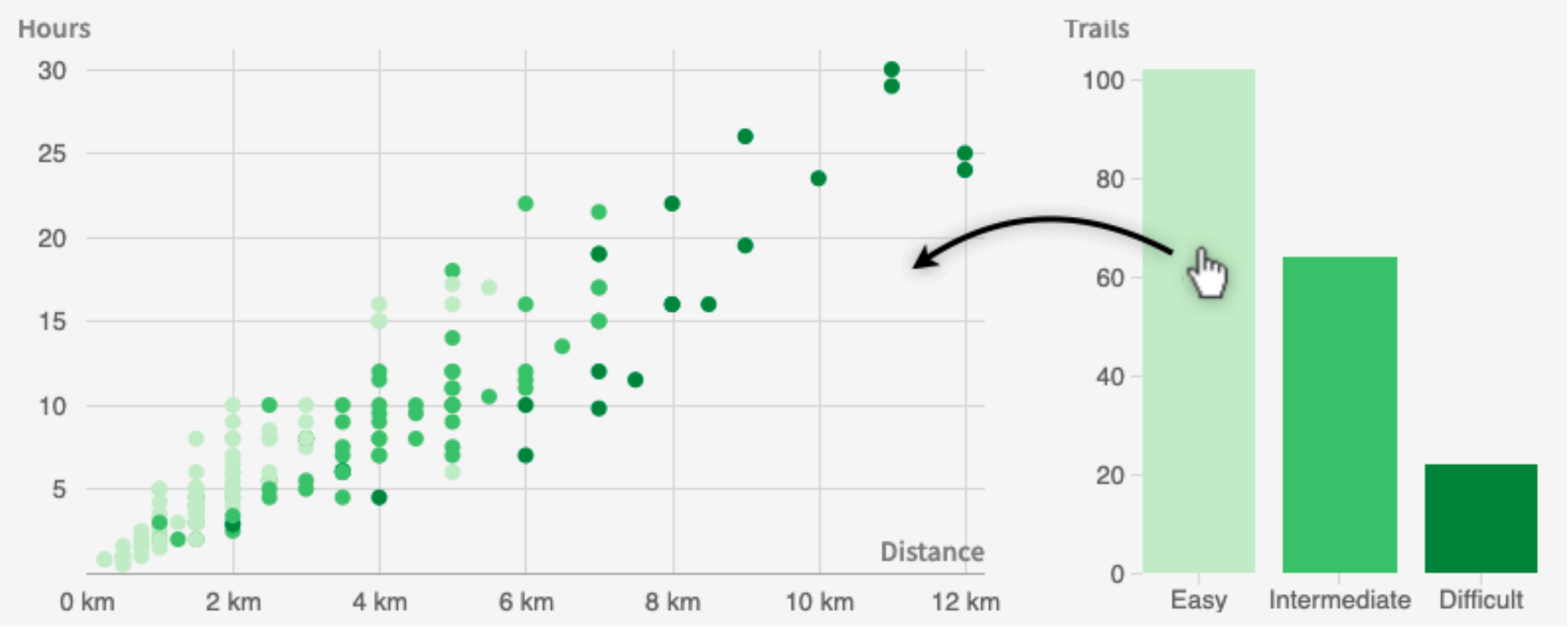
Outline

- Introduction to D3
- Making a chart
- Data joins and basic interactivity
- Multiple views and advanced interactivity
- Advanced concepts

Multiple views and advanced interactivity

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

- Linked interactions



Multiple views and advanced interactivity

- Multi-view event handler
  - d3.dispatch()
  - d3.dispatch('filteredCategories', 'selectedPoints', 'reset')

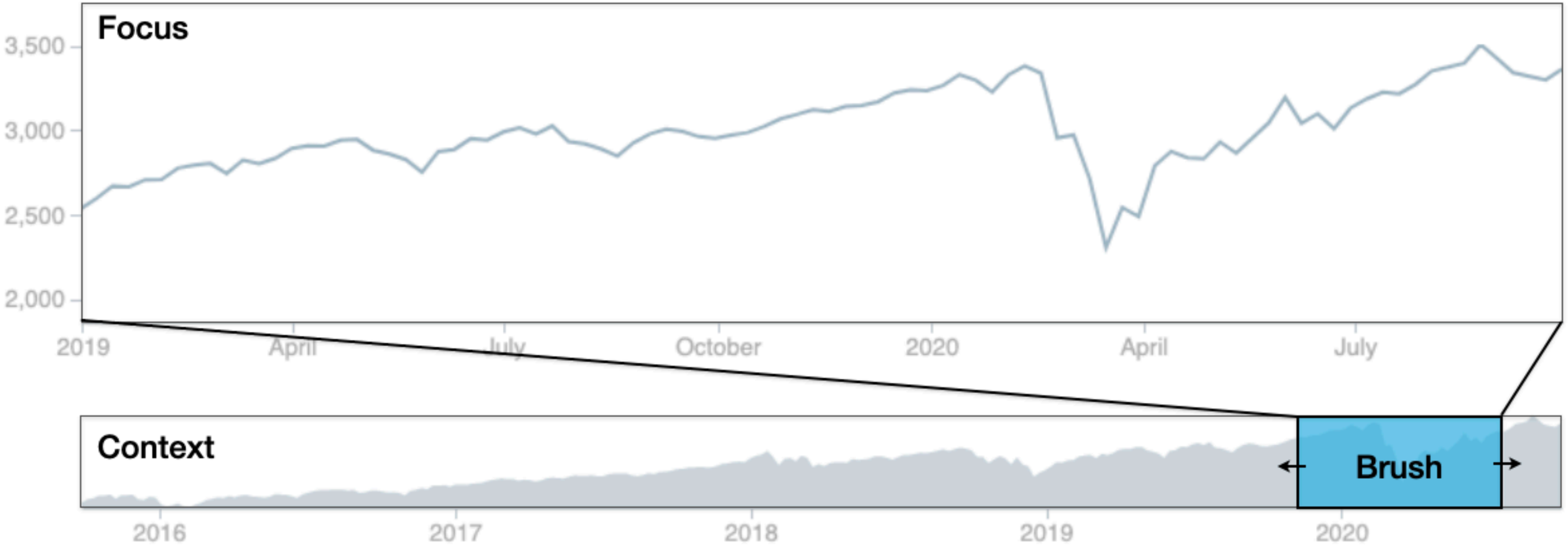
```
dispatcher.on('filterCategories', selectedCategories => {
    if (selectedCategories.length == 0) {
        scatterplot.data = data;
    } else {
        scatterplot.data = data.filter(d => selectedCategories.includes(d.difficulty));
    }
    scatterplot.updateVis();
});
```

Multiple views and advanced interactivity

- Brushing and linking
  - Brushing: a technique to interactively select a region or a set of data points in a visualization
  - linking: change are automatically dispatched to linked visualizations
- Focus + Context
  - The *context view* provides a global perspective at reduced detail and allows users to brush
  - The *focus view* shows the selected data points, for example, a specific time period, in greater detail

Multiple views and advanced interactivity

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen



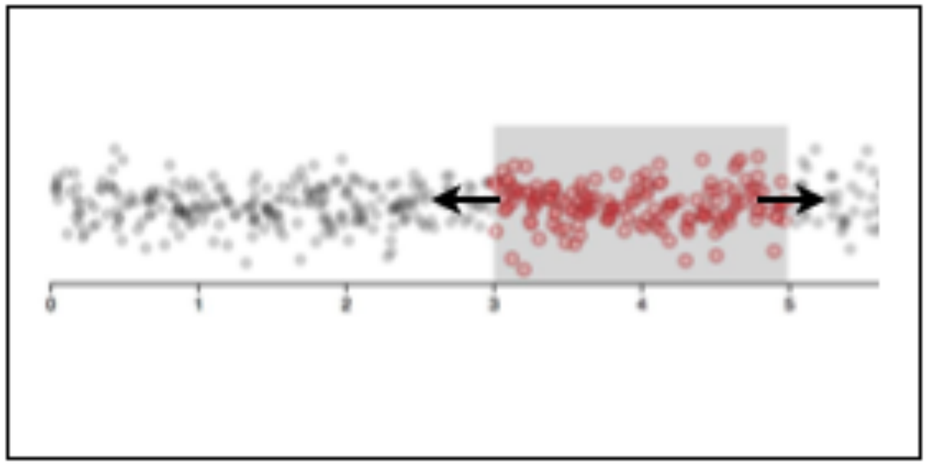
Multiple views and advanced interactivity

- Brushing
  - d3.brushX
  - d3.brushY
  - d3.brush

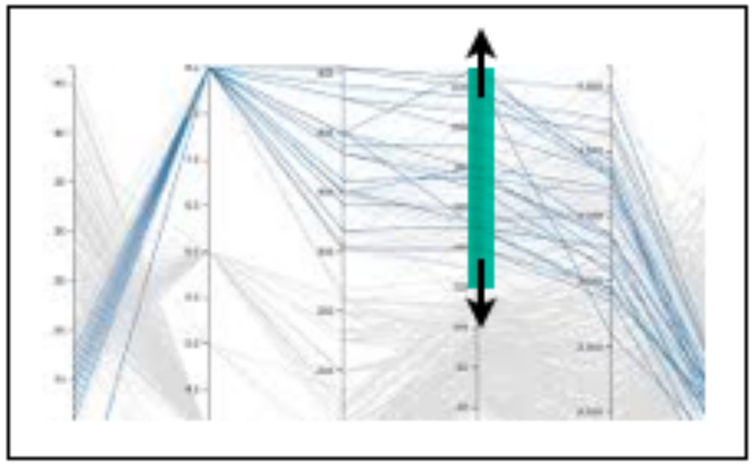
```
const brush = d3.brushX()
    .extent([[0, 0], [width, height]])
    .on('brush', brushed)
    .on('end', brushended);
```

```
const brushG = svg.append('g')
    .attr('class', 'brush x-brush')
    .call(brush);
```

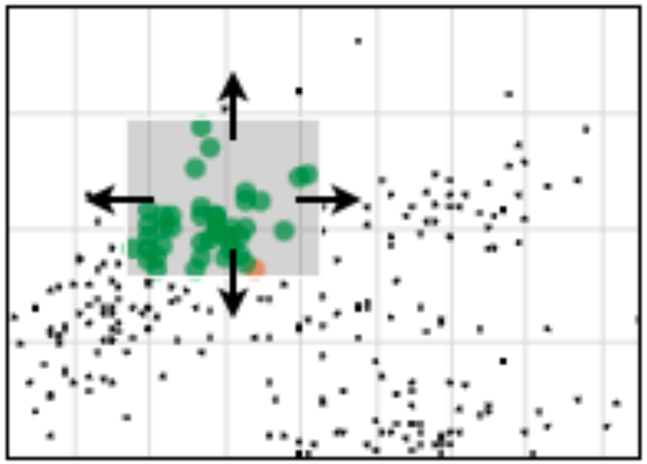
d3.**brushX**



d3.**brushY**



d3.**brush**

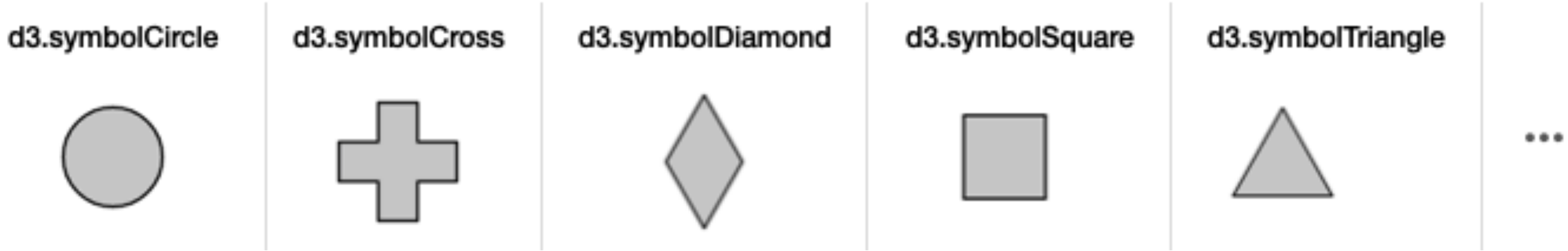


Multiple views and advanced interactivity

- D3 shape generator
  - Line
  - Area
  - Symbol
  - Stacks
  - etc.

Multiple views and advanced interactivity

- Symbol
  - d3.symbol




```
const symbolScale = d3.scaleOrdinal()
    .range([
      d3.symbol().type(d3.symbolCircle)(),
      d3.symbol().type(d3.symbolSquare)(),
      d3.symbol().type(d3.symbolDiamond)()
    ])
    .domain(['Easy', 'Intermediate', 'Difficult']);
```

Multiple views and advanced interactivity

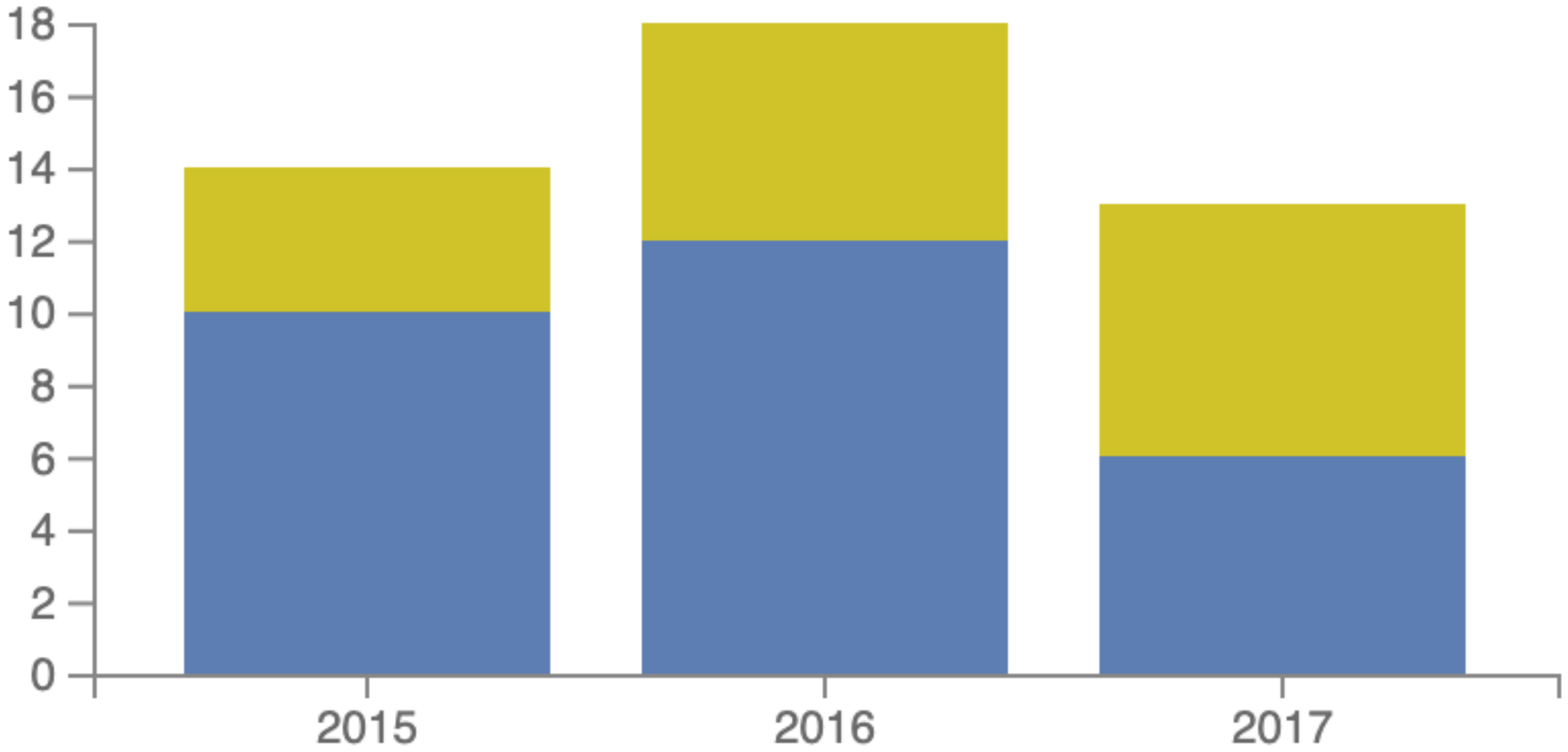
- Stacks
  - d3.stack

```
const data = [
        { 'year': 2015, 'milk': 10, 'water': 4 },
        { 'year': 2016, 'milk': 12, 'water': 6 },
        { 'year': 2017, 'milk': 6, 'water': 7 }
];

const stack = d3.stack().keys(['milk', 'water']);

const stackedData = stack(data);
```

```
const rectangles = svg.selectAll('category')
                .data(stackedData)
        .join('g')
                .attr('class', d => `category cat-${d.key}`)
        .selectAll('rect')
                .data(d => d)
        .join('rect')
                .attr('x', d => xScale(d.data.year))
                .attr('y', d => yScale(d[1]))
                .attr('height', d => yScale(d[0]) - yScale(d[1]))
                .attr('width', xScale.bandwidth());
```

Multiple views and advanced interactivity

- Stacks
  - d3.stack



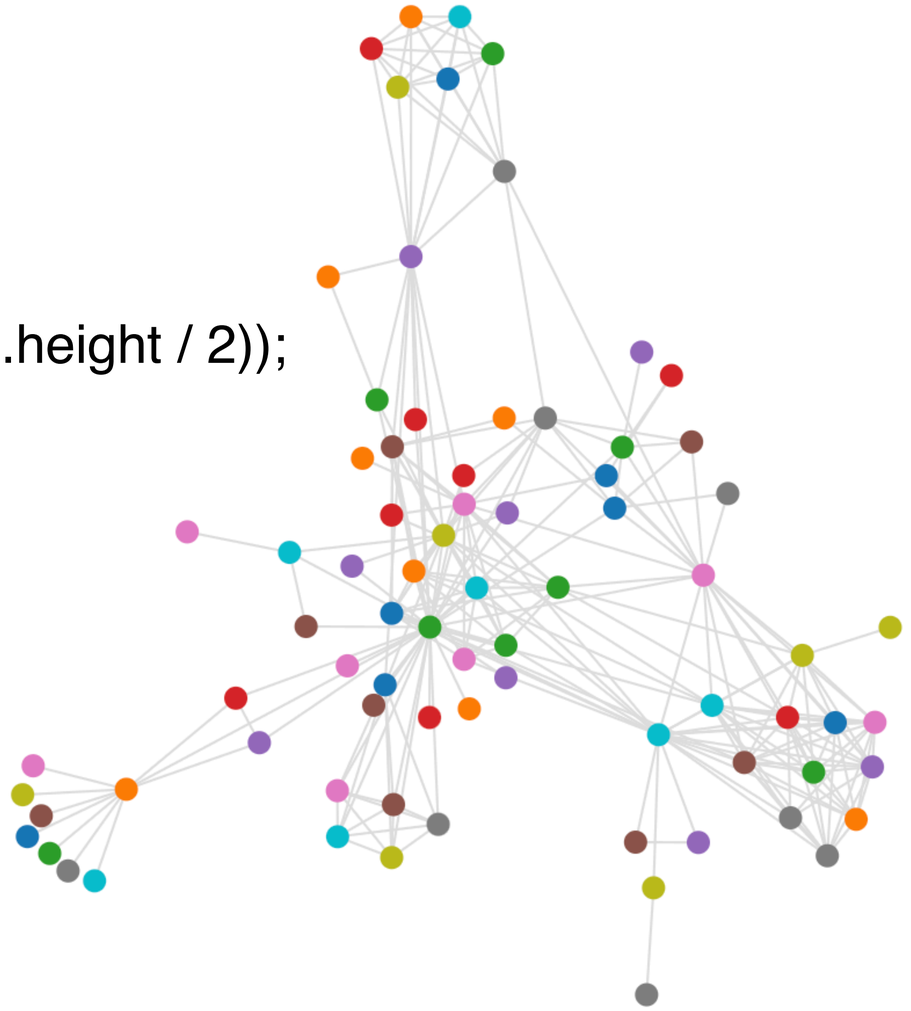
Outline

- Introduction to D3
- Making a chart
- Data joins and basic interactivity
- Multiple views and advanced interactivity
- Advanced concepts

Advanced concepts

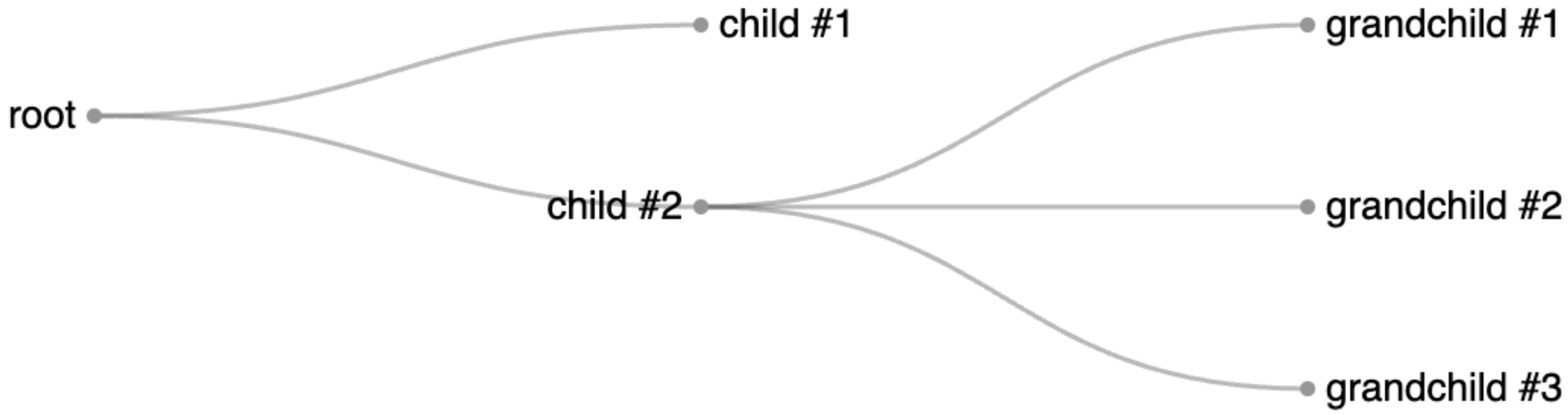
- Graph and trees
  - Graph: forced-directed layout

```
const simulation = d3.forceSimulation()
        .force('link', d3.forceLink().id(d => d.id))
        .force('charge', d3.forceManyBody())
        .force('center', d3.forceCenter(config.width / 2, config.height / 2));
```



- Graph and trees
  - Trees: tiny-tree layout

```
const data = d3.hierarchy(rawData)
const treeData = d3.tree().size([height, width])(data);
```

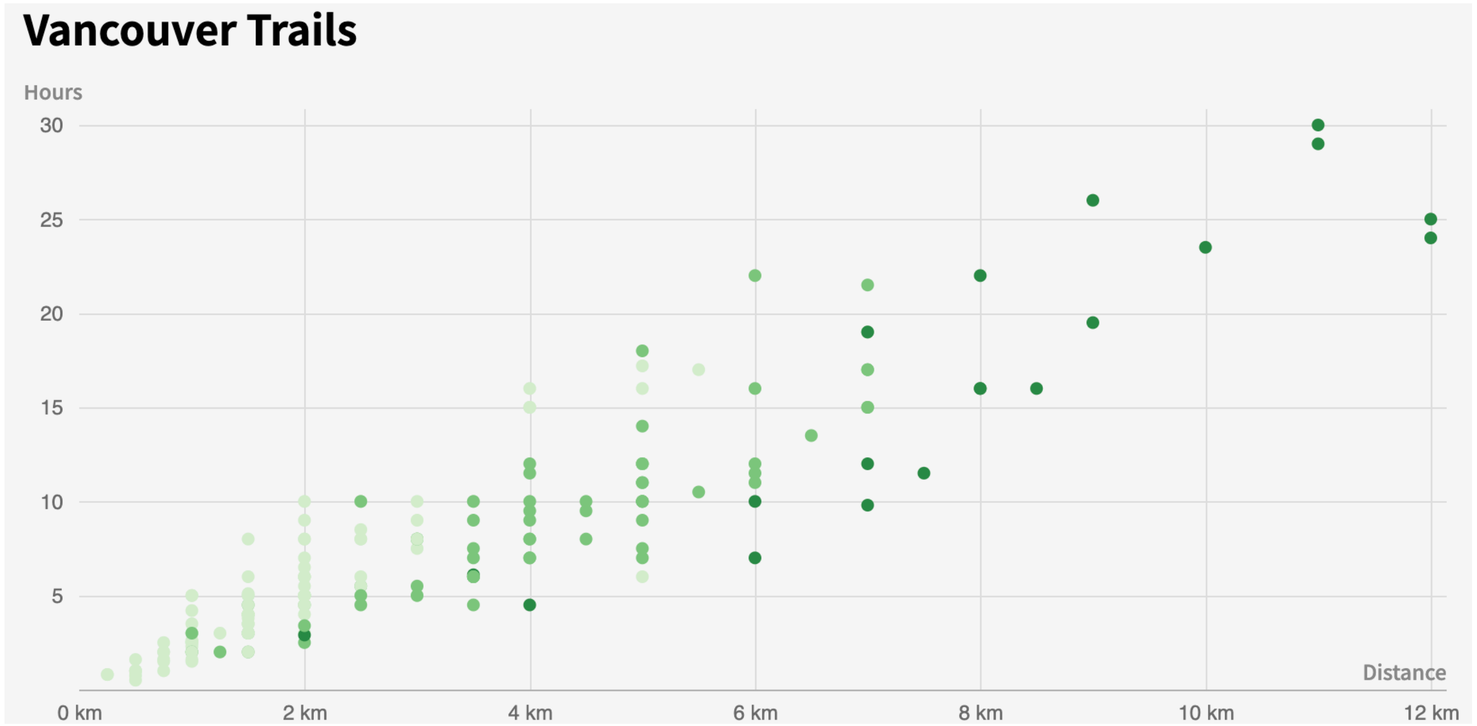


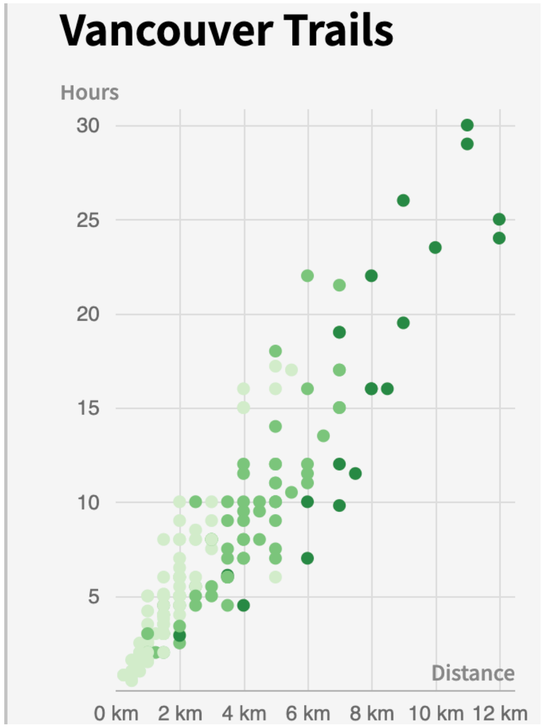
- Scroll-based interaction
  - A popular technique to reveal or alter content based on the user's scroll behavior
  - Recommend Waypoints to listen for scroll events
  - https://www.theguardian.com/us-news/ng-interactive/2015/oct/19/homan-square-chicago-police-detainees
  - https://pudding.cool/2017/03/hamilton/index.html
  - https://vallandingham.me/scroller.html

Advanced concepts

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

- Responsive visualization
  - Dynamically change the size of visualization forms based on wide variety of screen sizes





- Annotations
  - Make a visualization more accessible to your target audience
  - Improve clarity, for example, by explaining data anomalies or highlighting interesting patterns in a visualization
  - https://d3-annotation.susielu.com/

Thank Dr. Michael Oppermann for many of the slides!