Data Visualization (From Theory to Practice)

Seungmin Jin
(dryjins@gmail.com)

Dr Seungmin Jin

진 승 민 / Джин Сын Мин

Researcher

International Laboratory for Intelligent Systems and Structural Analysis
Faculty of Computer Science, HSE University, Moscow, Russia

Laboratory for Human-AI Interaction and Visualization
Faculty of Computer Science, UNIST, Ulsan, South Korea

Mail: sedzhin@hse.ru **LinkedIn:** https://www.linkedin.com/in/dryjins

Research Focus : eXplainable AI, Deep Learning, Visual Analytics

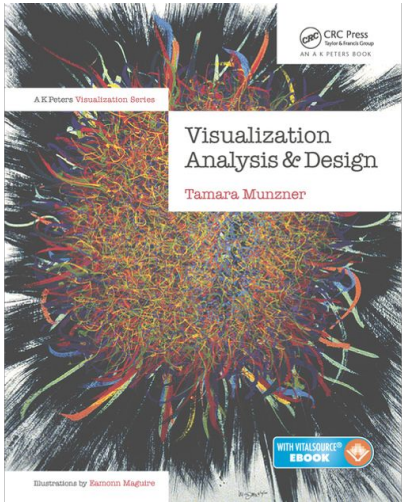
Goals

Course objectives: Understanding visualization development from abstract concepts to implementation

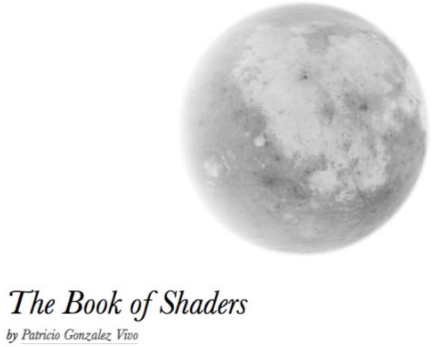
1. The fundamental of data visualization development
2. The core concept of D3
3. The examples and practices of D3 with LLMs

Prerequisite: Programing

Textbook



[Visualization Analysis and Design]



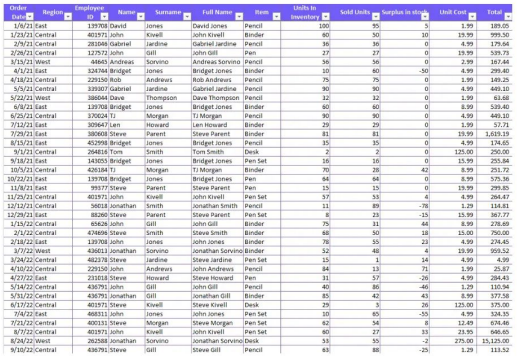
[The Book of Shaders]

[Beautifully Animate Points with WebGL and regl - Peter Beshai]

[Data Visualization with D3, JavaScript, React - Full Course [2021] - YouTube]

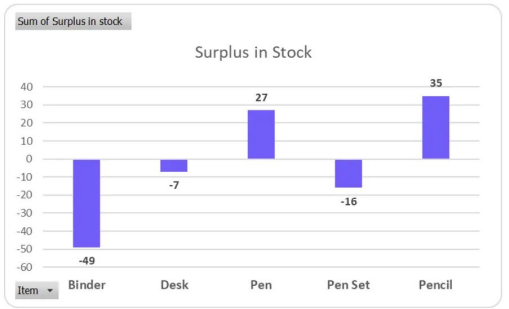
Data Visualization Fundamentals

NASA (1965) "**Man is the lowest-cost, 150-pound, nonlinear, all-purpose computer** system which can be mass-produced by unskilled labour".



Order Date	Region	Employee ID	Name	Surname	Full Name	Item	Units In Inventory	Sold Units	Surplus in stock	Unit Cost	Total
1/6/21	East	139708	David	Jones	David Jones	Pencil	100	95	5	1.99	189.05
1/23/21	Central	401971	John	Kivell	John Kivell	Binder	60	50	10	19.99	999.50
2/9/21	Central	281046	Gabriel	Jardine	Gabriel Jardine	Pencil	36	36	0	4.99	179.64
2/26/21	Central	127572	John	Gill	John Gill	Pen	27	27	0	19.99	539.73
3/13/21	West	44645	Andreas	Sorvino	Andreas Sorvino	Pencil	56	56	0	2.99	167.44
4/1/21	East	324744	Bridget	Jones	Bridget Jones	Binder	10	60	-50	4.99	299.40
4/18/21	Central	229150	Rob	Andrews	Rob Andrews	Pencil	75	75	0	1.99	149.25
5/5/21	Central	339307	Gabriel	Jardine	Gabriel Jardine	Pencil	90	90	0	4.99	449.10
5/22/21	West	386044	Dave	Thompson	Dave Thompson	Pencil	32	32	0	1.99	63.68
6/9/21	East	139708	Bridget	Jones	Bridget Jones	Binder	60	60	0	8.99	539.40
6/25/21	Central	370024	TJ	Morgan	TJ Morgan	Pencil	90	90	0	4.99	449.10
7/12/21	East	309647	Len	Howard	Len Howard	Binder	29	29	0	1.99	57.71
7/29/21	East	380603	Steve	Parent	Steve Parent	Binder	81	81	0	19.99	1,619.19
8/15/21	East	452996	Bridget	Jones	Bridget Jones	Pencil	35	35	0	4.99	174.65
9/1/21	Central	264818	Tom	Smith	Tom Smith	Desk	2	2	0	125.00	250.00
9/18/21	East	143055	Bridget	Jones	Bridget Jones	Pen Set	16	16	0	15.99	255.84
10/5/21	Central	426184	TJ	Morgan	TJ Morgan	Binder	70	28	42	8.99	251.72
10/22/21	East	139708	Bridget	Jones	Bridget Jones	Pen	64	64	0	8.99	575.36
11/8/21	East	99377	Steve	Parent	Steve Parent	Pen	15	15	0	19.99	299.85
11/25/21	Central	401971	John	Kivell	John Kivell	Pen Set	57	53	4	4.99	264.47
12/12/21	Central	98018	Jonathan	Smith	Jonathan Smith	Pencil	11	89	-78	1.29	114.81
12/29/21	East	88260	Steve	Parent	Steve Parent	Pen Set	8	23	-15	15.99	367.77
1/15/22	Central	65626	John	Gill	John Gill	Binder	75	31	44	8.99	278.69
2/1/22	Central	474698	Steve	Smith	Steve Smith	Binder	68	50	18	15.00	750.00
2/18/22	East	139708	John	Jones	John Jones	Binder	78	55	23	4.99	274.45
3/7/22	West	436011	Jonathan	Sorvino	Jonathan Sorvino	Binder	52	48	4	19.99	959.52
3/24/22	Central	482378	Steve	Jardine	Steve Jardine	Pen Set	15	1	14	4.99	4.99
4/10/22	Central	229150	John	Andrews	John Andrews	Pencil	84	13	71	1.99	25.87
4/27/22	East	231018	Steve	Howard	Steve Howard	Pen	31	57	-26	4.99	284.43
5/14/22	Central	436791	John	Gill	John Gill	Pencil	40	86	-46	1.29	110.94
5/31/22	Central	436791	Jonathan	Gill	Jonathan Gill	Binder	85	42	43	8.99	377.58
6/17/22	Central	401971	Steve	Kivell	Steve Kivell	Desk	29	3	26	125.00	375.00
7/4/22	East	468311	John	Jones	John Jones	Pen Set	10	65	-55	4.99	324.35
7/21/22	Central	400131	Steve	Morgan	Steve Morgan	Pen Set	62	54	8	12.49	674.46
8/7/22	Central	401971	John	Kivell	John Kivell	Pen Set	80	27	33	23.95	646.65
8/24/22	West	282588	Jonathan	Sorvino	Jonathan Sorvino	Desk	53	55	-2	275.00	15,125.00
9/10/22	Central	436791	Steve	Gill	Steve Gill	Pencil	63	88	-25	1.29	113.52

Data in table



Data visualized in chart

Cognitive → **Visual Perception**

Workload

Data Visualization Fundamentals

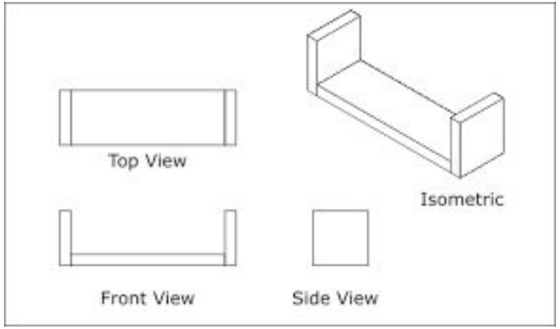
Data ⇨ **Data Manipulation** | **Visual Mapping (Design)** | **Rendering** ⇨ **Visualization**

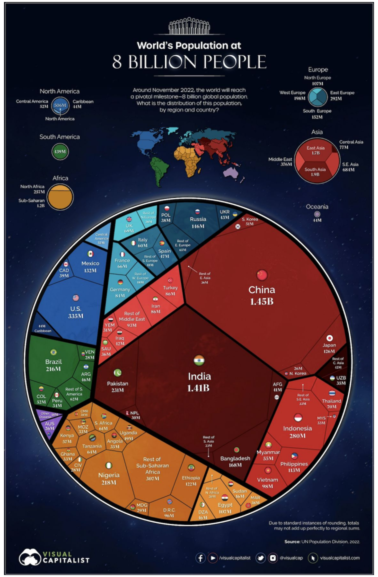
Rank	CCA3	Country/Territory	Capital	Continent	2022 Population	2020 Population	2015 Population	2010 Population	2000 Population	1990 Population	1980 Population	1970 Population	Area (km²)	Density (per km²)	Growth Rate	World Population Percentage
139	JAM	Jamaica	Kingston	North America	2827377	2820436	2794445	2733896	2612205	2392030	2135546	1859091	10991	257.2447	0.9999	0.04
107	LBY	Libya	Tripoli	Africa	6812341	6653942	6192235	6491988	5154790	4236983	2962720	1909177	1759540	3.8717	1.0114	0.09
22	TZA	Tanzania	Dodoma	Africa	65497748	61704518	52542823	45110527	34463704	26206012	19297659	13618192	945087	69.3034	1.0300	0.82
183	PYF	French Polynesia	Papeete	Oceania	306279	301920	291787	283788	250927	211089	163591	117891	4167	73.5011	1.0074	0.00
89	HND	Honduras	Tegucigalpa	North America	10432860	10121763	9294505	8450933	6656725	5053234	3777990	2782753	112492	92.7431	1.0150	0.13
178	GLP	Guadeloupe	Basse-Terre	North America	395752	395642	399089	403072	424067	391951	334234	318310	1628	243.0909	0.9992	0.00
26	MMR	Myanmar	Nay Pyi Taw	Asia	54179306	53423198	51483949	49390988	45538332	40099553	33465781	27284112	676578	80.0784	1.0071	0.68
221	VGB	British Virgin Islands	Road Town	North America	31305	30910	29366	27556	20104	15617	11109	9581	151	207.3179	1.0059	0.00
94	HUN	Hungary	Budapest	Europe	9967308	9750573	9844246	9986825	10202055	10375989	10698679	10315366	93028	107.1431	1.0265	0.12
136	PRI	Puerto Rico	San Juan	North America	3252407	3271564	3497335	3717922	3827108	3543776	3214568	2737619	8870	366.6750	0.9989	0.04

- Filtering
- Transform
- Outliers

- Charts
- Visual Encoding
- Layout

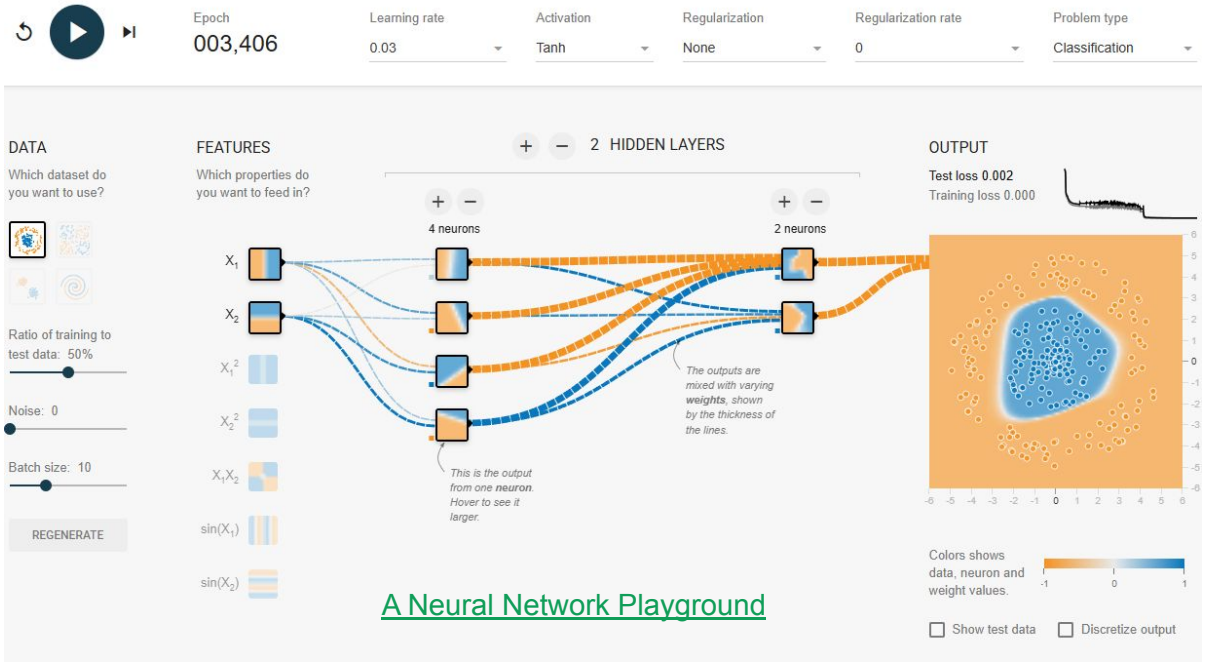
- Views
- Interaction
- Optimization



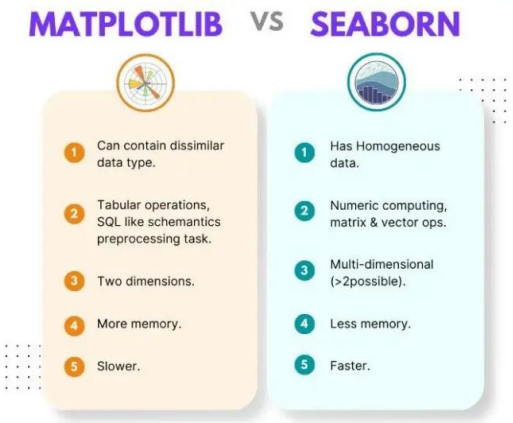


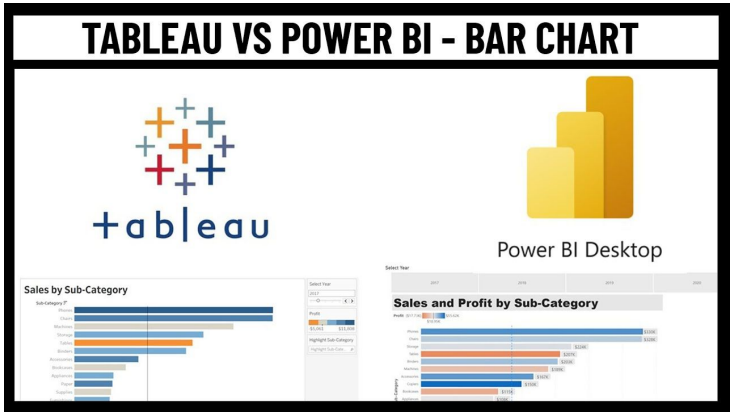
Data Visualization Fundamentals

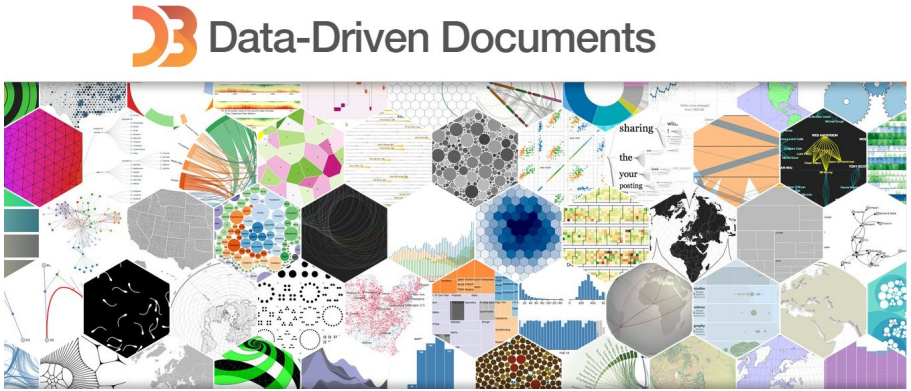
Data → **Data Manipulation** → **Visual Mapping (Design)** → **Rendering** → **Visualization**



Frameworks



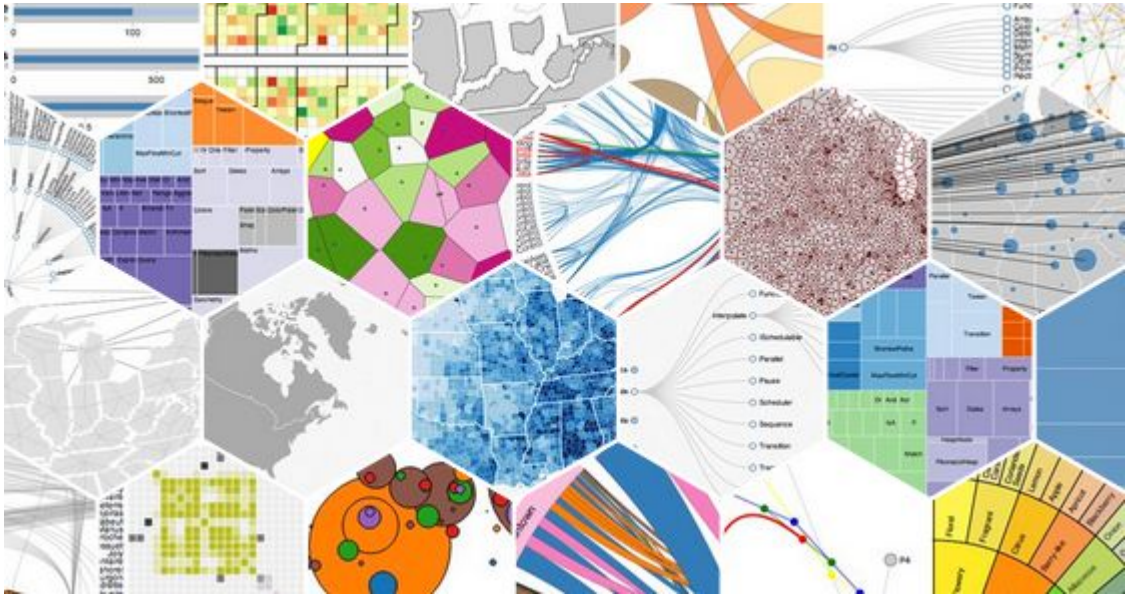


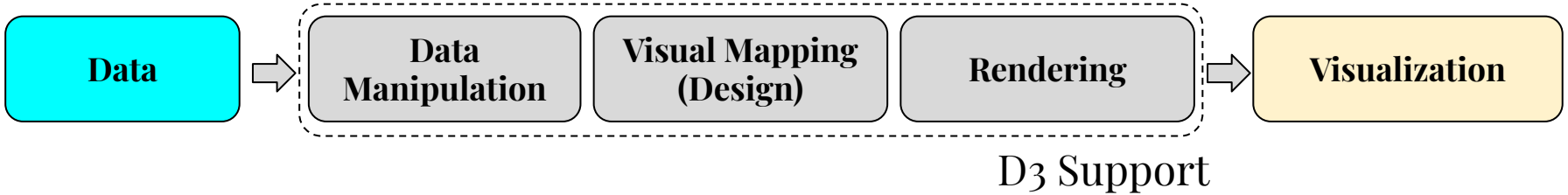


- Opensource
- Industrial Level
- Flexible Customization

D3 Overview

- D3.js is a JavaScript library for manipulating documents based on data.
- Data-Driven Documents (D3)
- Provide unified interface for SVG, DOM and canvas
- **You can access tons of example for visualizations!**
- https://d3js.org/

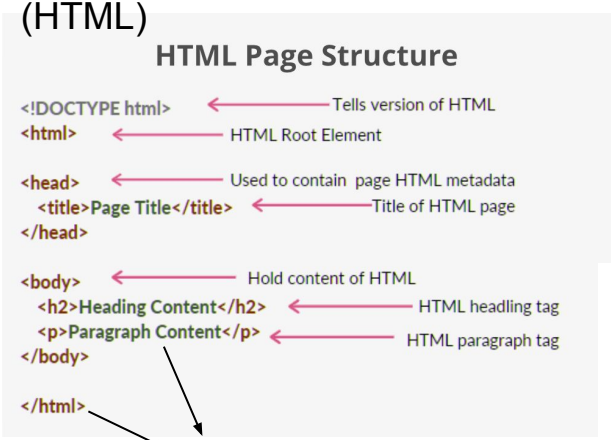




HTML5 and Web Browser

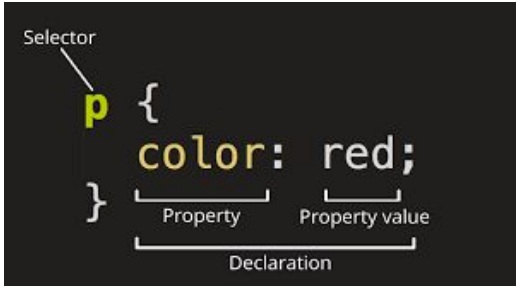
Language for display!

HyperText **Markup** Language (HTML)

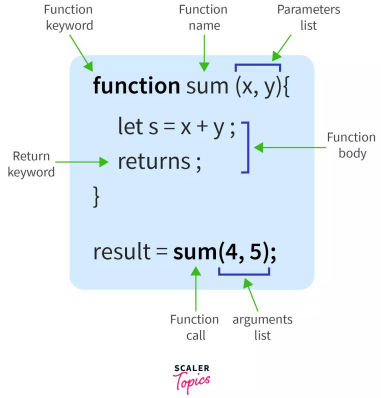


Document Object Model (DOM) Elements

CSS (Cascading **Style** Sheets)



Javascript (**Functions**)



read

rendering

Rendering

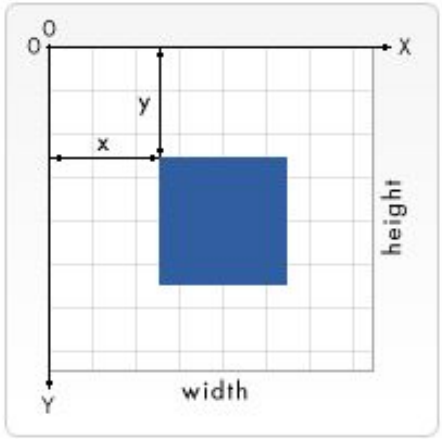
We can visualize anything with HTML and CSS, so HTML provides...

Rendering One Million Datapoints with D3 and WebGL

Scalable Vector Graphics (SVG - DOM)

- SVG defines the graphics in XML format
- Every **element** and every attribute in SVG files can be animated
- SVG requires **incremental space complexity** for each elements (3000 max)

HTML Canvas



CPU → Canvas 2D

GPU → WebGL

- A mere container for graphics.
- You must use JavaScript to **actually draw** the graphics.
- Canvas require **fixed space complexity** based on the height and width.

Scalable Vector Graphics (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 with a specific width and height to your HTML document, for example: <svg width="500" height="500"></svg> .
- The SVG coordinate system places the origin (0/0) in the top-left corner of the svg element.
- SVG has no layering concept or depth property. The order in which elements are coded determines their **depth order.**

Scalable Vector Graphics (SVG)

Rect, Circle, Ellipse, Line and Text

gistfile1.html — Raw

```
<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>
```

https://gist.github.com/dryjins/75de87dfa8c6e28d37683cfdc3cf501a

result

Scalable Vector Graphics (SVG)

Polygon

polygon.html

Raw

```
1  <?xml version="1.0" standalone="no"?>
2  <svg width="12cm" height="4cm" viewBox="0 0 1200 400"
3       xmlns="http://www.w3.org/2000/svg" version="1.1">
4    <desc>Example polygon01 - star and hexagon</desc>
5
6    <!-- Show outline of viewport using 'rect' element -->
7    <rect x="1" y="1" width="1198" height="398"
8          fill="none" stroke="blue" stroke-width="2" />
9
10   <polygon fill="red" stroke="blue" stroke-width="10"
11            points="350,75  379,161 469,161 397,215
12                    423,301 350,250 277,301 303,215
13                    231,161 321,161" />
14   <polygon fill="lime" stroke="blue" stroke-width="10"
15            points="850,75  958,137.5 958,262.5
16                    850,325 742,262.6 742,137.5" />
17 </svg>
```

Define the position to connect polygon, The order matters.

https://gist.github.com/dryjins/d5a1a5bcac96f4f06c3da3358b6fb931

result

Scalable Vector Graphics (SVG)

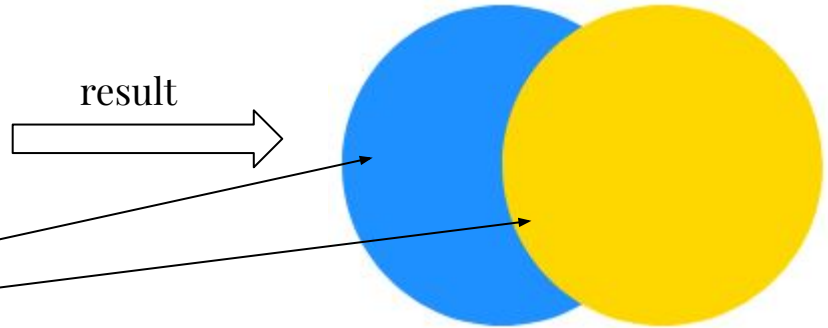
Depth order = Document order

3.3 Rendering Order (SVG specification version 1.1)

Elements in an SVG document fragment have an implicit drawing order, with the **first elements** in the SVG document fragment getting **"painted" first**. Subsequent elements are painted on top of previously painted elements.

index.html

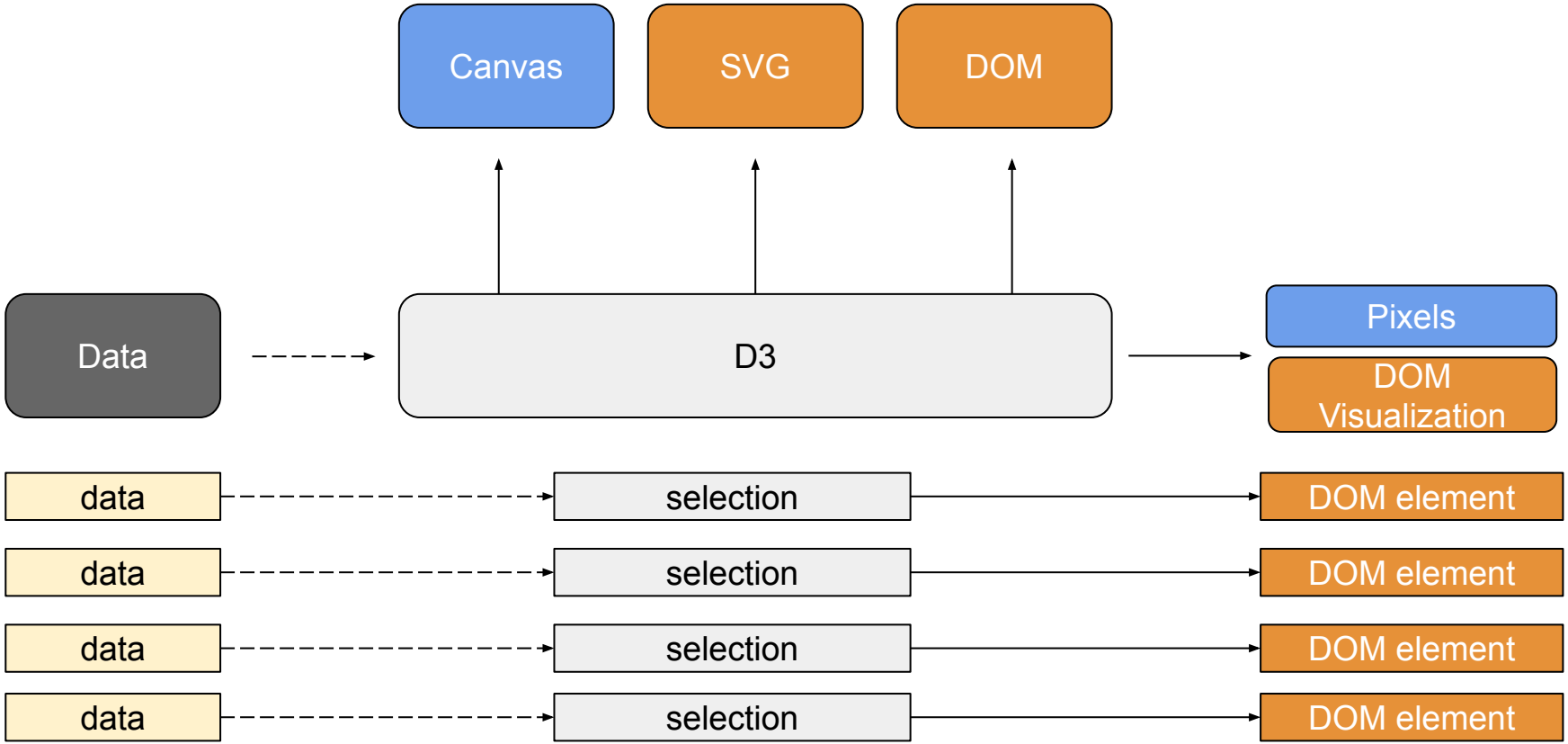
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="src/style.css">
  </head>
  <body>
 <?xml version="1.0" standalone="no"?>
  <svg width="400" height="400" xmlns="http://www.w3.org/2000/svg" version="1.1">
    <circle cx="40" cy="200" r="40" fill="dodgerblue" />
    <circle cx="80" cy="200" r="40" fill="gold" />
  </svg>
  </body>
</html>
```



https://gist.github.com/dryjins/160aeae7253cabe8c16b5b80361de90e

D3 Design

Data Driven Documents



D3 Design

Core Elements

1. **Document Selection**: `d3.select("#visualization")`
2. **(Canvas or SVG) Container Creation**: Establishing visualization canvas
3. **Data Loading & Processing**: Using `d3.csv()` and data transformation
4. **Scales**: Mapping data domains to visual ranges
5. **Axes**: Creating visual reference system
6. **Data Binding**: Associating data with DOM elements
7. **Enter-Update-Exit Pattern**: Managing data-driven element lifecycle
8. **Modern Join Pattern**: Simplified approach for enter/update/exit
9. **Transitions**: Animating changes in the visualization
10. **Interactivity**: Handling user events and dynamic behavior

The structure follows the general "Setup → Data → Scales → Axes → Elements → Interactions" pattern that forms the foundation of most D3.js visualizations.

D3 Basic

Selection : select, append and selectAll

- 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.
- In our example we have used d3.select('body') to select the first DOM element that matches our CSS selector, body. Once an element is selected - and handed off to the next method in the chain - you can apply **operators**. These D3 operators allow you to get and set **properties**, **styles**, and **content** (and will again return the current selection).
- Alternatively, if you need to select more than one element, use selectAll().
- https://gist.github.com/dryjins/a9179e0d793358961c180f910beaf01f

index.html

```
<html>
  <head>
    <script src="https://d3js.org/d3.v7.min.js">
  </head>
  <body>


  </body>
</html>
```

script.js

```
let helloWorld = ['Hello World!']
let data = Array(10).fill(10).map((x,i)=>x+i)

let body = d3.select('body')

body.select('p')
.data(helloWorld)
.enter()
.append('p')
.text(d=>d)

body.append('ul')
.selectAll('li')
.data(data)
.enter()
.append('li')
.text(d=>d)
```

D3 Basic

Selection : select, append and selectAll

- After selecting a specific element or multiple elements, 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.
- At the end, we use the text() operator to insert a string between the opening and closing tags of the current selection ( ).

Dynamic Properties

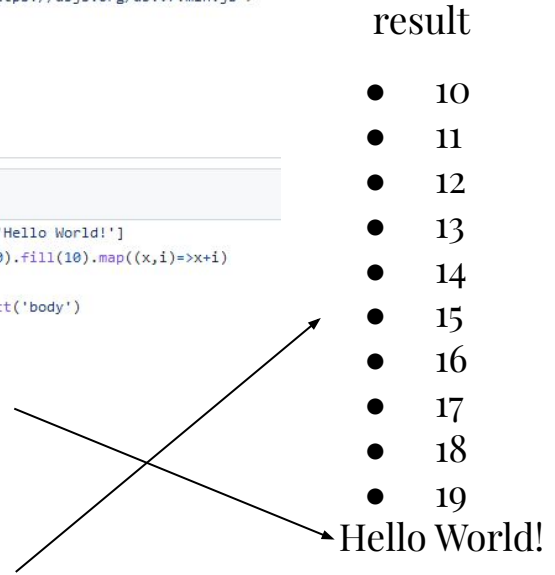
If you want access to the corresponding values from the dataset, you have to change *anonymous functions*. For example, we can change such a function inside the text() operator to change the text.

index.html

```
<html>
  <head>
    <script src="https://d3js.org/d3.v7.min.js">
  </head>
  <body>


  </body>
</html>
```

script.js

```
let helloWorld = ['Hello World!']
let data = Array(10).fill(10).map((x,i)=>x+i)

let body = d3.select('body')

body.select('p')
.data(helloWorld)
.enter()
.append('p')
.text(d=>d)

body.append('ul')
.selectAll('li')
.data(data)
.enter()
.append('li')
.text(d=>d)
```



https://gist.github.com/dryjins/a9179e0d793358961c180f910beaf01f

D3 Basic

HTML attributes and CSS properties

- we can get and set different **properties** and **styles** - not only the textual content. This becomes very important when working with SVG elements.
- We use D3 to set the HTML class, and color for odd and even numbers,
- The font-weight and size defined in style.css file for the odd.
- The font color which depends on the individual array value.
- https://gist.github.com/dryjins/32422667974f39f1ef56f9272e868521

index.html

```
<html>
  <head>
    <script src="https://d3js.org/d3.v7.min.js">
  </head>
  <body>
  </body>
</html>
```

script.js

```
let data = Array(10).fill(10).map((x,i)=>x+i)

let body = d3.select('body')

let ul = body.append('ul')

let li = ul.selectAll('li')
.data(data)
.enter()
.append('li')
.text(d=>d)

// add class for even and odd
li.attr('class',d=> d % 2 == 0 ? 'even' : 'odd')

// change color of even and odd
li.style('color',d=> d % 2 == 0 ? 'red' : 'blue')
```

style.css

```
.odd {
  font-size:20px;
  font-weight:bold;
}
```

result

- 10
- **11**
- 12
- **13**
- 14
- **15**
- 16
- **17**
- 18
- **19**

D3 Basic

HTML attributes and CSS properties for SVG

- We have appended SVG elements to the DOM tree in our second example.
- This means that we had to create the SVG drawing area first. We did this with D3 and saved the selection in the variable svg.
- It is crucial to set the SVG coordinates of visual elements. If we don't set the x and y values, all the rectangles will be drawn at the same position at (0, 0).
- By using the index, of the current element in the selection, we can create a dynamic x property and shift every newly created rectangle 60 pixels to the right.
- https://gist.github.com/dryjins/30e8d4504af95593d19378b8d3eb9333

result

SVG attribute

index.html

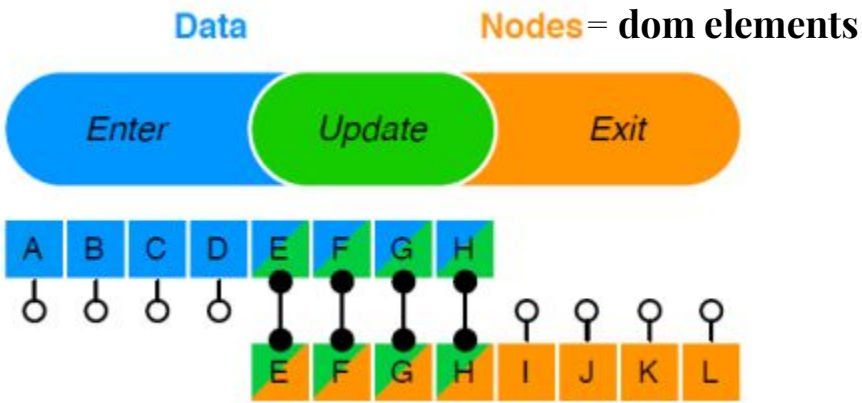
```
<html>
  <head>
    <script src="https://d3js.org/d3.v7.min.js">
  </head>
  <body>



  </body>
</html>
```

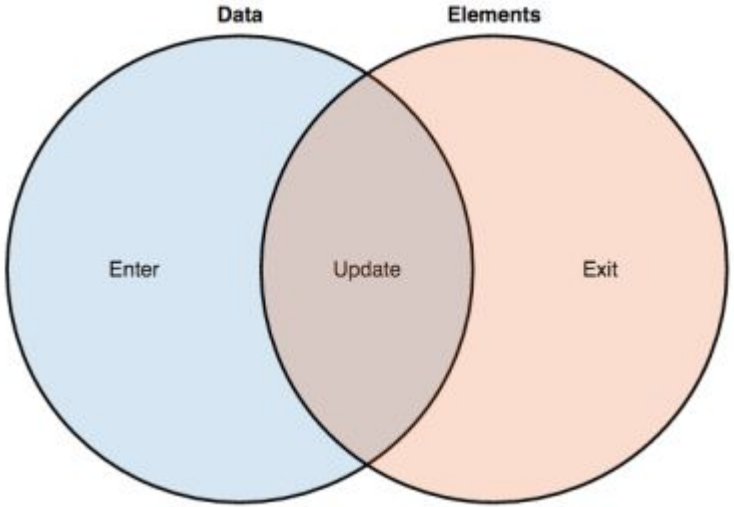
script.js

```
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);
```

D3 Data Binding (join)

Enter, Update, and Exit





- Any changes of DOM trigger redrawing from the web browser which is very expensive for the cost-wise.
- Enter, Update, Exit triggers the browser redawning DOM elements only if necessary.
- D3 provides update and exit at the selection level then apply the changes to the DOM later to redraw the changes at one time.
- These also provide the fancy way to animate the changes of visualization dynamically. We will cover this part later.

D3 Data Binding (join)

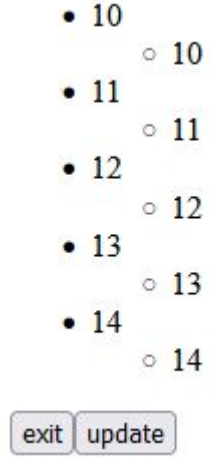
Merge, and Exit

- Merge is combination of update and enter which append new elements.
- From previous our example, lets' slice 5 items from the list so that remains only 10 to 14.
- Select changed <li> and append new sublist <ul> and join itself
- You can test the process by pressing exit and update buttons
- https://gist.github.com/dryjins/180d29ee3a2def213b08dd29f8a5d6a2

script.js

```
let data = Array(10).fill(10).map((x,i)=>x+i)

let body = d3.select('body')

let ul = body.append('ul')

let li = ul.selectAll('li')
.data(data)
.enter()
.append('li')
.text(d=>d)

function exit(){
  let dataExit = data.slice(0,5)
  console.log(data)
  li
    .data(dataExit)
    .exit()
    .remove()
}

body.append('button')
  .text('exit')
  .on('click', exit)

function update(){
  li = li.append('ul')
    .merge(li)
    .append('li')
    .text(d=>d)
}

body.append('button')
  .text('update')
  .on('click', update)

```

result



D3 External Data Loading

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.

CSV (Comma Separated Values)

CSV is a file format which is often used to exchange data. Each line in a CSV file represents a table row and as the name indicates, the values/columns are separated by a comma.

JavaScript Object Notation (JSON)

JSON is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for **transmitting data in web applications.**

In a nutshell: The use of the right file format depends on the data - JSON should be used for hierarchical data and CSV is usually a proper way to store tabular data.

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

=

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

D3 External Data Loading

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.

Why do we need an asynchronous execution?

Reading a file from the disk or an external server usually takes a while. Hence, we are using an asynchronous execution: We don't have to wait and stall, instead we can proceed with further tasks that do not rely on the dataset. After receiving a notification that the data loading process is complete, the callback function then() is executed.

sandwiches.csv

name,	price,size	
Thesis,	7.95,	large
Dissertation	,8.95,	large
Highlander	,6.50,	small
Just Tuna	,6.50,	small
So-La,	7.95,	large
Special,	12.50,	small

script.js

```
1  d3.csv('data/sandwiches.csv')
2    .then(data => {
3          console.log('Data loading complete. Work with dataset.');
4      console.log(data);
5    })
6    .catch(error => {
7      console.error('Error loading the data');
8    });
9
10 console.log('Do something else, without the data');
```

```
Do something else, without the data
Data loading complete. Work with dataset.
▼Array(6) i
  ▶0: {name: "Thesis", price: "7.95", size: "large"}
  ▶1: {name: "Dissertation", price: "8.95", size: "large"}
  ▶2: {name: "Highlander", price: "6.50", size: "small"}
  ▶3: {name: "Just Tuna", price: "6.50", size: "small"}
  ▶4: {name: "So-La", price: "7.95", size: "large"}
  ▶5: {name: "Special", price: "12.50", size: "small"}
  ▶columns: (3) ["name", "price", "size"]
   length: 6
  ▶__proto__: Array(0)
```

Note that the later executed first because of the **asynchronous execution**

Scales and Axes

Scales: Mapping functions

- Convert data domain to visual range
- Handle the mathematical heavy-lifting

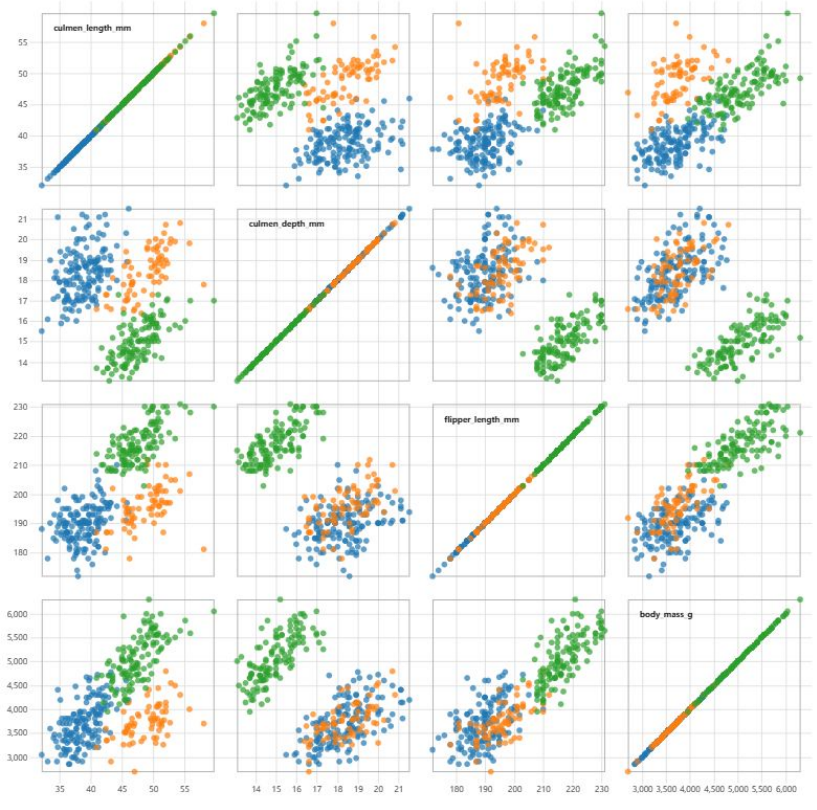
Common scale types:

```javascript
// Linear scale example
const xScale = d3.scaleLinear()
  .domain([0, d3.max(dataset, d => d.value)])
  .range([0, width]);

// Time scale example
const timeScale = d3.scaleTime()
  .domain([new Date(2020, 0, 1), new Date(2020, 11, 31)])
  .range([0, width]);
```

Axes: Visual representation of scales

```javascript
// Creating an axis
const xAxis = d3.axisBottom(xScale);
svg.append("g")
  .attr("transform", `translate(0, ${height})`)
  .call(xAxis);
```

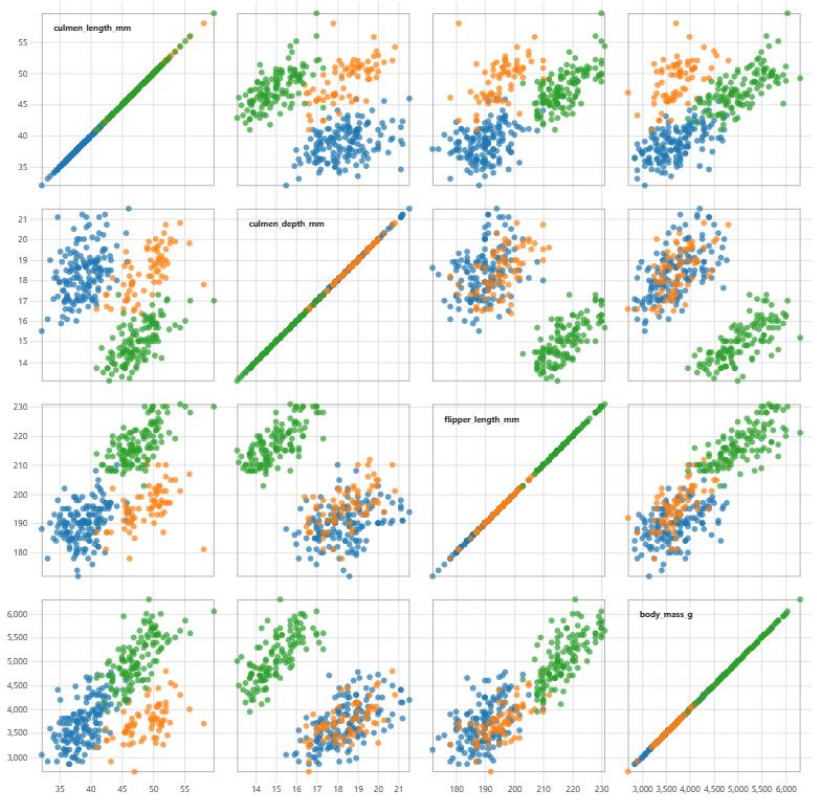
Interaction (Transition)



[Brushable Scatterplot Matrix / D3 | Observable (observablehq.com)]

```
// cell groups
const cell = svg
  .append("g")
  .selectAll("g")
  .data(d3.cross(d3.range(columns.length), d3.range(columns.length)))
  .enter()
  .append("g")
  .attr("transform", ([i, j]) => "translate(" + i * size + "," + j * size + ")");

// draw rectangles for each cell
cell
  .append("rect")
  .attr("fill", "none")
  .attr("stroke", "#aaa")
  .attr("x", padding / 2)
  .attr("y", padding / 2)
  .attr("width", size - padding)
  .attr("height", size - padding);
```

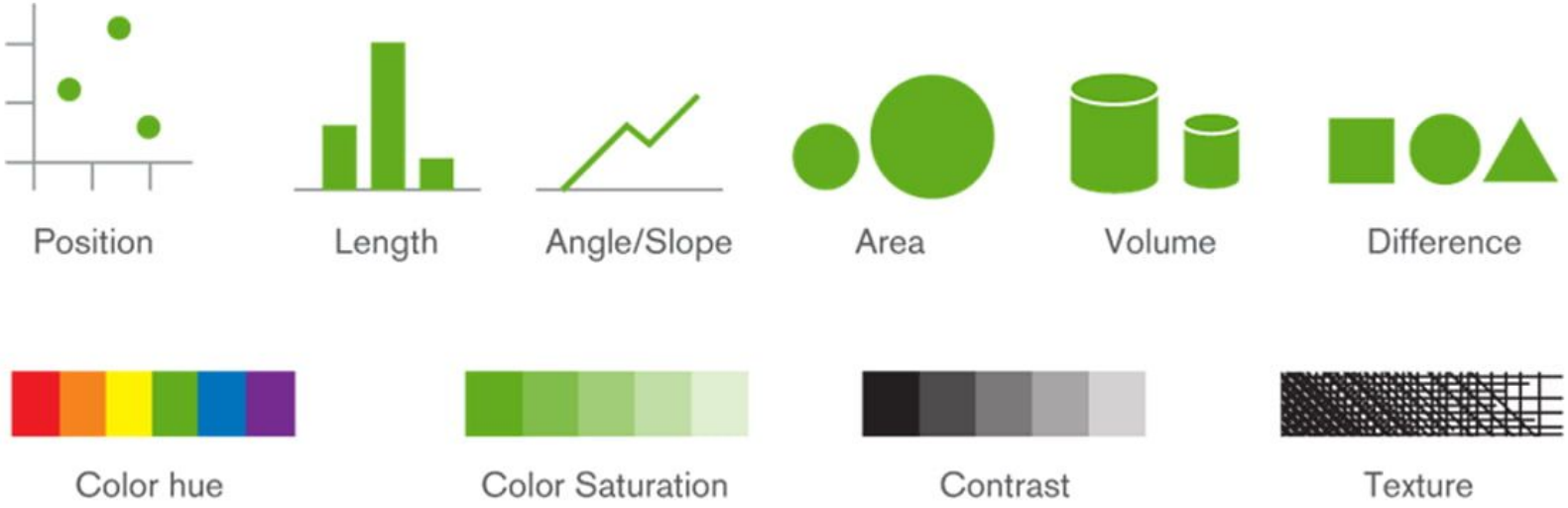
Interaction (Transition)



[Brushable Scatterplot Matrix / D3 | Observable (observablehq.com)]

```
// Highlight the selected circles.
function brushed({selection}, [i, j]) {
  let selected = [];
  if (selection) {
    const [[x0, y0], [x1, y1]] = selection;
    circle.classed("hidden",
      d => x0 > x[i](d[columns[i]])
        || x1 < x[i](d[columns[i]])
        || y0 > y[j](d[columns[j]])
        || y1 < y[j](d[columns[j]]));
    selected = data.filter(
      d => x0 < x[i](d[columns[i]])
        && x1 > x[i](d[columns[i]])
        && y0 < y[j](d[columns[j]])
        && y1 > y[j](d[columns[j]]));
  }
  svg.property("value", selected).dispatch("input");
}
```

D3-shape (Visual Encoding)



D3.stack (d3-shape)

stack · GitHub

- The line const stack = **d3.stack().keys(["value1", "value2", "value3"]); creates a stack generator** function using the d3.stack() method from the d3-shape module in D3.js.
- The d3.stack() function takes an array of data and returns a new array representing the stacked layout of the data. The .keys(["value1", "value2", "value3"]) method call **specifies the keys (or property names) in the data objects that will be used for stacking**.
- In the given example, the resulting stack variable is a stack generator function that is ready to be applied to your data. When the generator function is invoked with the data, it will **calculate the positions for each stacked segment based on the provided keys**.

```
// Sample data
const data = [
  { category: "A", value1: 10, value2: 20, value3: 30 },
  { category: "B", value1: 15, value2: 25, value3: 35 },
  { category: "C", value1: 8, value2: 18, value3: 28 }
];

// Stack generator
const stack = d3.stack().keys(["value1", "value2", "value3"]);

// Apply stack layout to the data
const stackedData = stack(data);

// Create scales for x and y axes
const xScale = d3.scaleBand()
  .domain(data.map(d => d.category))
  .range([0, 300])
  .padding(0.1);

const yScale = d3.scaleLinear()
  .domain([0, d3.max(stackedData, d => d3.max(d, d => d[1]))])
  .range([200, 0]);
```

stackedData

[

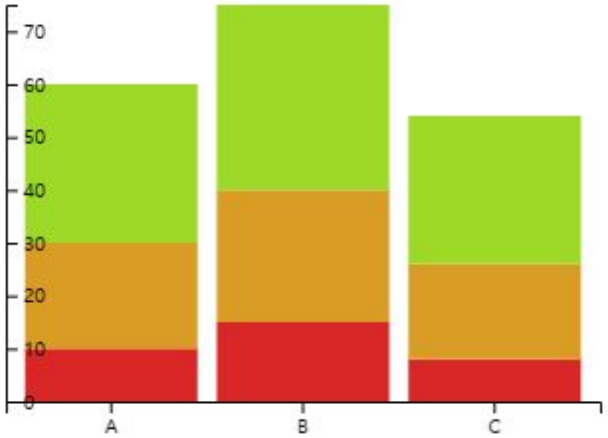
[[0,10], [0,15], [0,8]],

[[10,30], [15,40], [8,26]],

[[30,60], [40,75], [26,54]]

]

D3.stack (d3-shape)



HSL stands for Hue, Saturation, and Lightness

```
[
   [ [0,10],    [0,15],    [0,8] ],
   [ [10,30],   [15,40],   [8,26] ],
   [ [30,60],   [40,75],   [26,54] ]
]
```

```
24  // Create SVG element
25  const svg = d3.select("#chart");
26
27  // Create groups for each stacked bar
28  const groups = svg.selectAll("g")
29    .data(stackedData)
30    .join("g")
31    .attr("fill", (d, i) => `hsl(${i * 40}, 70%, 50%)`);
32
33  // HSL stands for Hue, Saturation, and Lightness. It is a color mx
34
35  // Create stacked bars within each group
36  groups.selectAll("rect")
37    .data(d => d)
38    .join("rect")
39    .attr("x", (d, i) => xScale(data[i].category))
40    .attr("y", d => yScale(d[1]))
41    .attr("height", d => yScale(d[0]) - yScale(d[1]))
42    .attr("width", xScale.bandwidth());
43
```

D3-shape

"D3js provides the translation of code for SVG and canvas."

D3JS

```
d="M150 0 L75 200 L225 200 Z"
```

SVG <path>

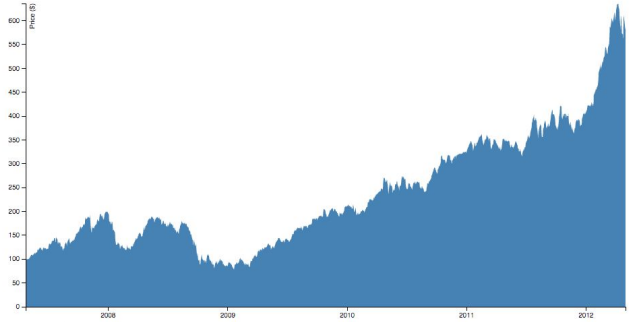
```
context.moveTo(0, 0);
context.lineTo(200, 100);
context.stroke();
```

Canvas

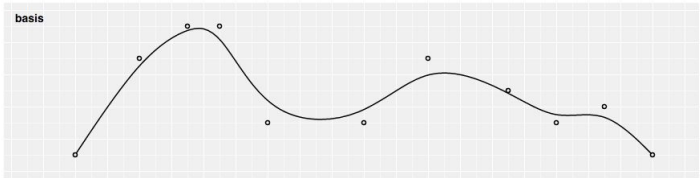
D3-shape

D3-shape

- Arcs
- Pies
- **Lines**
- Areas
- Curves
- Custom Curves
- Links
- **Symbols**
- Custom Symbol Types
- **Stacks**

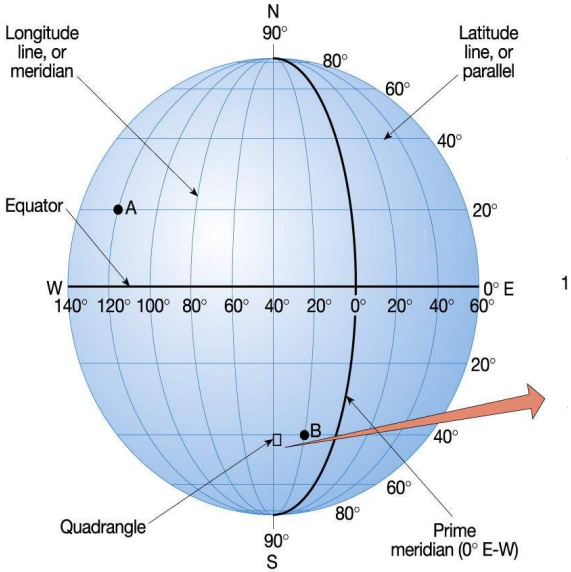


B-spline



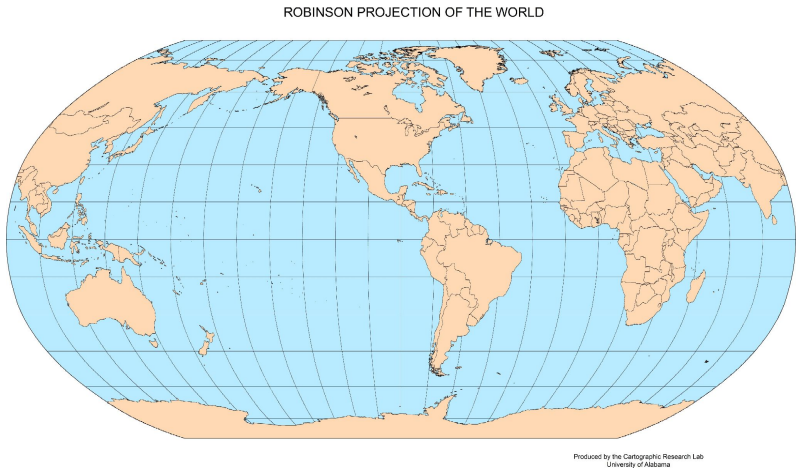
D3-geo: Projection

3D, Latitude, Longitude



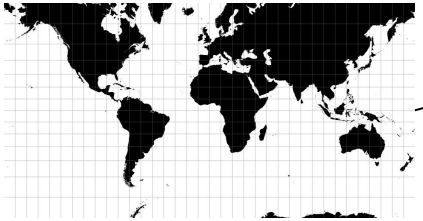
Projection

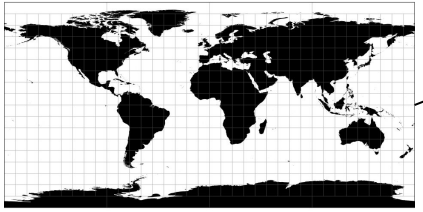
2D, X, Y

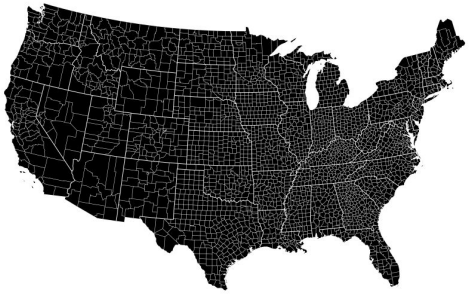


The **transformation** of **spherical coordinates** (latitude and longitude) onto a **two-dimensional plane**, allowing the representation of a three-dimensional Earth on a flat surface.

D3-geo: Projection







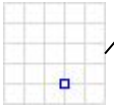
- **Mercator Projection**: The Mercator projection is one of **the most well-known projections** and is often used for world maps. It preserves angles and straight lines but distorts the size of objects as they move away from the equator.
- **Equirectangular Projection**: Also known as the Plate Carrée projection, it simply maps the latitude and longitude values directly onto the Cartesian coordinates, resulting in a rectangular map. This projection distorts shapes and areas, particularly near the poles.
- **Orthographic Projection**: The orthographic projection displays the Earth as if viewed from a distant point in space. It is commonly used for **globe-like visualizations**, where the Earth is represented as a sphere.
- **Albers Projection**: The Albers projection is a conic projection that balances shape and area distortions. It is often used for regional maps, such as country-level or state-level maps.
- **Conic Equidistant Projection**: This projection preserves distances along a set of standard parallels, making it suitable for mapping specific regions or countries.

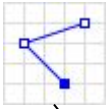
D3-geo: geoJson

Sample of GeoJson

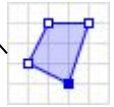
```
{
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [0, 0],
        [10, 0],
        [10, 10],
        [0, 10],
        [0, 0]
      ]
    ]
  },
  "properties": {
    "name": "Example Polygon",
    "color": "blue"
  }
}
```

{ "type": "**Point**", "coordinates": [30.0, 10.0] }





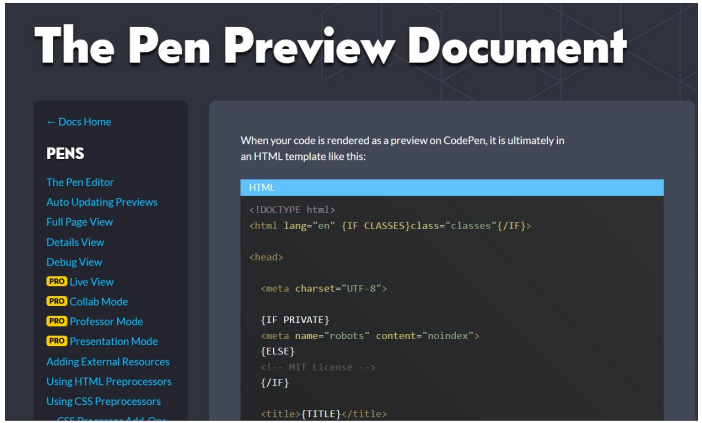
{ "type": "**LineString**", "coordinates": [[30.0, 10.0], [10.0, 30.0], [40.0, 40.0]] }



- "**type**": "Feature": Indicates that the GeoJSON object is a feature.
  - "FeatureCollection" allows array of "Feature"
- "**geometry**": Defines the geometry of the feature.
  - "type": "Polygon": Specifies that the geometry type is a polygon.
  - "coordinates": Contains an array of coordinate values representing the vertices of the polygon. In this case, we have a square defined by a set of five coordinates.
- "**properties**": Contains additional information associated with the feature.
  - "name": "Example Polygon": Specifies a name or label for the polygon.
  - "color": "blue": Represents an additional property like the color of the polygon.

Web Development (environment)

Online Editor



Codepen

https://codepen.io

1. Easy to access
2. Github gist support
3. Computation limit.

Github

Github

https://github.com/

1. Version control
2. Easy to share your codes using Gist

Contents

Leaflet + d3js

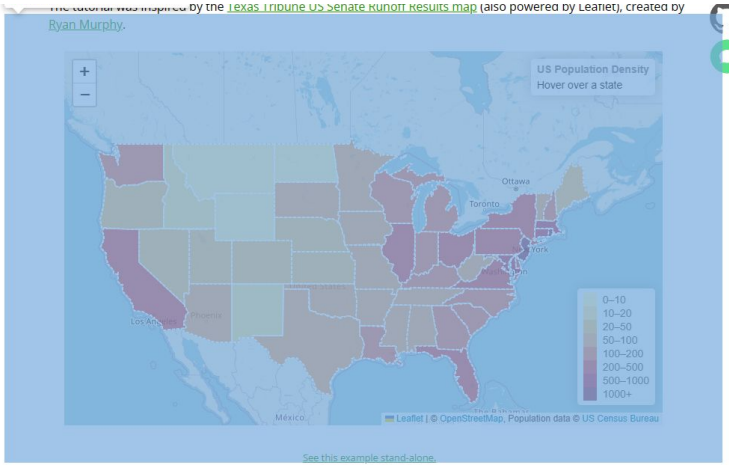
```
var svg =
d3.select(map.getPanes().overlayPane).append( "svg")
,
    g = svg.append( "g").attr("class",
"leaflet-zoom-hide" );
```

Select SVG canvas in Leaflet

Use projection to transform geoJson to screen

```
function projectPoint(x, y) {
  var point = map.latLngToLayerPoint( new
L.LatLng(y, x));
  this.stream.point(point.x, point.y);}
```

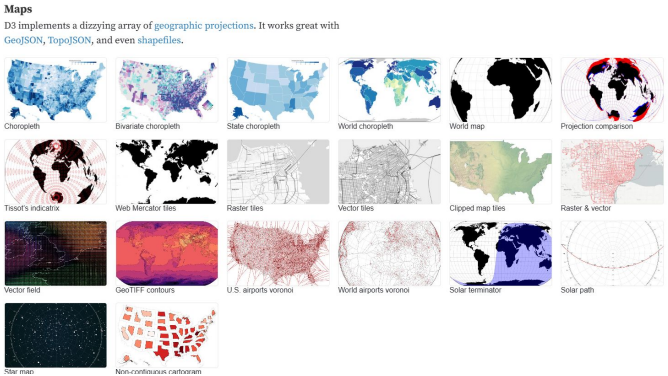
Previous practice!
[d3-geo-leaflet · GitHub]



[D3 + Leaflet (ocks.org)]

Use d3js to create interaction

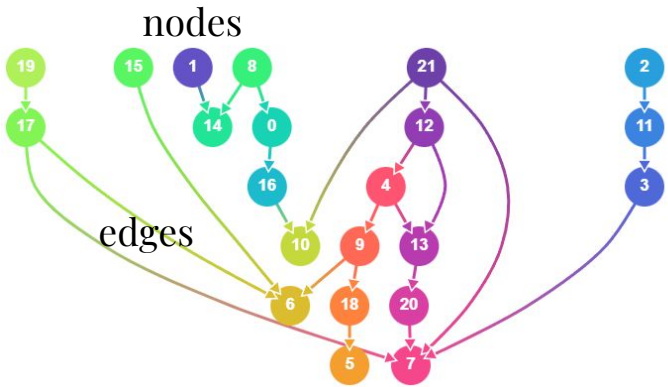
[d3-geo-projection / D3 | Observable (observablehq.com)]



Graph & Trees

- Introduction to Network Graphs
  - Network graphs, also known as force-directed graphs, visualize relationships between nodes as interconnected links.
  - D3.js provides powerful tools for creating interactive and visually appealing network graphs.
- Building a Network Graph with D3.js
  - Data Structure: Represent nodes and links using an appropriate data structure, such as an array of objects.
  - Node and Link Elements: Create SVG elements for nodes and links based on the data structure.
  - Force Simulation: Utilize D3's force simulation to determine node positions and link forces.
  - Data Binding and Visualization: Bind the processed data to DOM elements and apply visual attributes using D3's selection and manipulation functions.
- Customization and Interaction
  - Node Styling: Customize node size, color, shape, and labels to represent different entities or attributes.
  - Link Styling: Configure link width, color, and opacity to indicate the strength or type of connections.
  - Interactivity: Enable mouse events and interactions to highlight nodes, show tooltips, or trigger actions on user interaction.
- Network Analysis and Layout Algorithms
  - Network Metrics: Calculate and analyze metrics such as degree centrality or betweenness centrality to gain insights into the network structure.
  - Layout Algorithms: Apply different layout algorithms like force-directed, hierarchical, or radial layouts to arrange nodes and links.

d3-force/README.md at v3.0.0 · d3/d3-force · GitHub



Directed or Undirected

Graph & Trees

1. **Attractive Force**: This force pulls connected nodes closer together, encouraging them to be near each other. It helps maintain the connections and cohesion within the graph.
2. **Repulsive Force**: This force pushes nodes apart, preventing them from being too close to each other. It helps to avoid node overlap and maintain a visually balanced layout.
3. **Link Force**: This force acts along the links between nodes, influencing their length and tension. It helps maintain the desired link distance and can simulate springs or elastic connections.

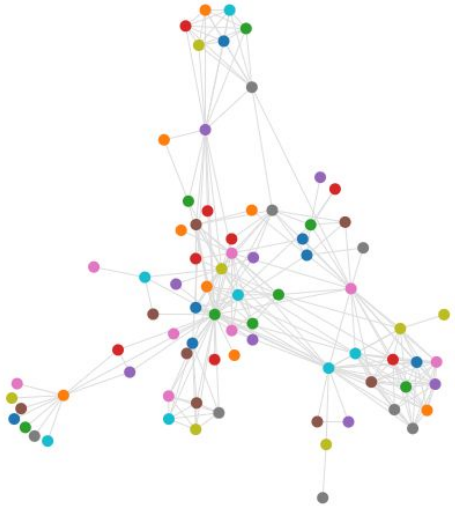
The force simulation algorithm iteratively calculates and updates the forces acting on each node based on these principles.

The forces are then applied to the nodes, causing them to move in response.

This iterative process continues until the forces reach an equilibrium, resulting in a stable layout.

UBC-InfoVis/2021-436V-examples: d3-force-directed-graph - CodeSandbox

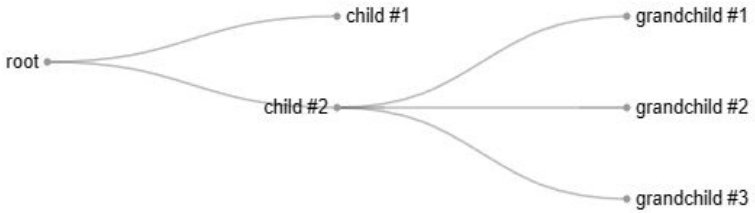
Network of character co-occurence in *Les Misérables*



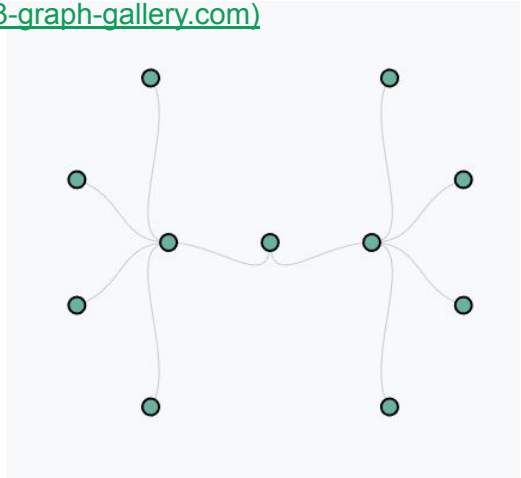
Graph & Trees

- Introduction to D3 Dendrogram
  - Dendrograms are hierarchical tree diagrams that represent relationships between entities.
  - D3.js provides powerful tools for creating interactive and visually appealing dendrograms.
- Key Features of D3 Dendrogram
  - Hierarchical Data Representation: Dendrograms excel at visualizing hierarchical relationships and tree structures.
  - Customizable Visual Elements: D3 Dendrogram allows customization of node shapes, colors, labels, and other visual attributes.
  - Interactive Exploration: Dendrograms created with D3.js offer interactive features like zooming, panning, and node collapsing/expanding.
- Building a D3 Dendrogram
  - **Data Preprocessing**: Prepare the hierarchical data structure in a format compatible with D3 dendrogram requirements.
  - **Layout Configuration**: Define the size, spacing, and orientation of the dendrogram layout.
  - **Data Binding and Visualization**: Bind the processed hierarchical data to DOM elements, create SVG elements for nodes and branches, and apply visual attributes and interactivity.

d3-shape/README.md at v3.2.0 · d3/d3-shape · GitHub



Most basic radial dendrogram in d3.js (d3-graph-gallery.com)



Graph & Trees

Data Preprocessing

```
const rawData = {
  name: "root",
  children: [
    {name: "child #1"},
    {
      name: "child #2",
      children: [
        {name: "grandchild #1"},
        {name: "grandchild #2"},
        {name: "grandchild #3"}
      ]
    }
  ]
};
```

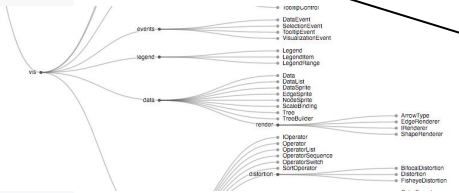
```
const data = d3.hierarchy(rawData);
```

[d3-hierarchy/README.md at v3.1.2 · d3/d3-hierarchy · GitHub]

- *node*.x - the *x*-coordinate of the node
- *node*.y - the *y*-coordinate of the node

Layout Configuration

```
// Draw the edges
const link = chart.selectAll('path')
    .data(treeData.links())
  .join('path')
    .attr('class', 'edge')
    .attr('d', d3.linkHorizontal()
        .x(d => d.y)
        .y(d => d.x));

// Draw the nodes
const node = chart.selectAll('g')
    .data(treeData.descendants())
  .join('g')
    .attr('transform', d => `translate(${d.y},${d.x})`);

node.append('circle')
    .attr('class', 'node-circle')
    .attr('r', 2.5);

node.append('text')
    .attr('dy', '0.31em')
    .attr('x', d => d.children ? -6 : 6)
    .attr('text-anchor', d => d.children ? 'end' : 'start')
    .text(d => d.data.name);
```

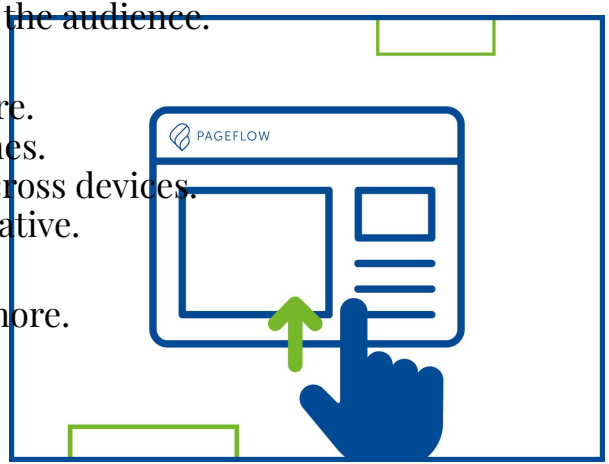


Data Binding and Visualization

[UBC-InfoVis/2021-436V-examples: d3-tidy-tree - CodeSandbox]

ScrollyTelling

- **Scrollytelling** is a technique that combines storytelling and data visualization in a scrolling narrative format.
- Key Features:
  - **Sequential Presentation**: Content is divided into scenes or sections.
  - **Scroll-Triggered Animations**: Each scene is revealed or animated as the user scrolls.
  - **Interactive Elements**: Users can interact with the content and explore data visualizations.
  - **Engaging User Experience**: Combines narrative storytelling with data-driven visuals.
- Benefits of Scrollytelling:
  - **Enhanced Storytelling**: Presents information in a cohesive and engaging narrative.
  - **Data Exploration**: Allows users to interactively explore data visualizations.
  - **Guided Experience**: Provides a guided and immersive experience for the audience.
- Best Practices:
  - **Planning the Narrative**: Designing a clear and coherent story structure.
  - **Visual Transitions**: Smooth and well-timed animations between scenes.
  - **Responsiveness**: Ensuring the scrollytelling experience works well across devices.
  - **User Feedback**: Providing visual cues to guide users through the narrative.
- Use Cases:
  - News articles, storytelling websites, data-driven presentations, and more.



ScrollyTelling

So You Want to Build A Scroller - Jim Vallandingham

Scroller

Scroll detector

Event dispatcher

Slide changes

d3-transition

```
function position() {
var pos = window.pageYOffset - 10;
var sectionIndex =
d3.bisect(sectionPositions, pos);
sectionIndex =
Math.min(sections.size() - 1,
sectionIndex);

if (currentIndex !== sectionIndex) {
dispatch.active(sectionIndex);
currentIndex = sectionIndex;}}
```

Window: pageYOffset property - Web APIs | MDN (mozilla.org)

Waypoints (imakewebthings.com)

Waypoints is the easiest way to trigger a function when you scroll to an element.

```
var waypoint = new Waypoint({
  element: document.getElementById('waypoint'),
  handler: function(direction) {
    console.log('Scrolled to waypoint!')
  }
})
```

Builds are available for multiple DOM libraries.

jQuery	Zepto	No Framework
1.8+	1.1+	IE 9+

UBC-InfoVis/2021-436V-examples: d3-waypoints-scrollytelling - CodeSandbox

Responsive Visualization

Responsive Visualization

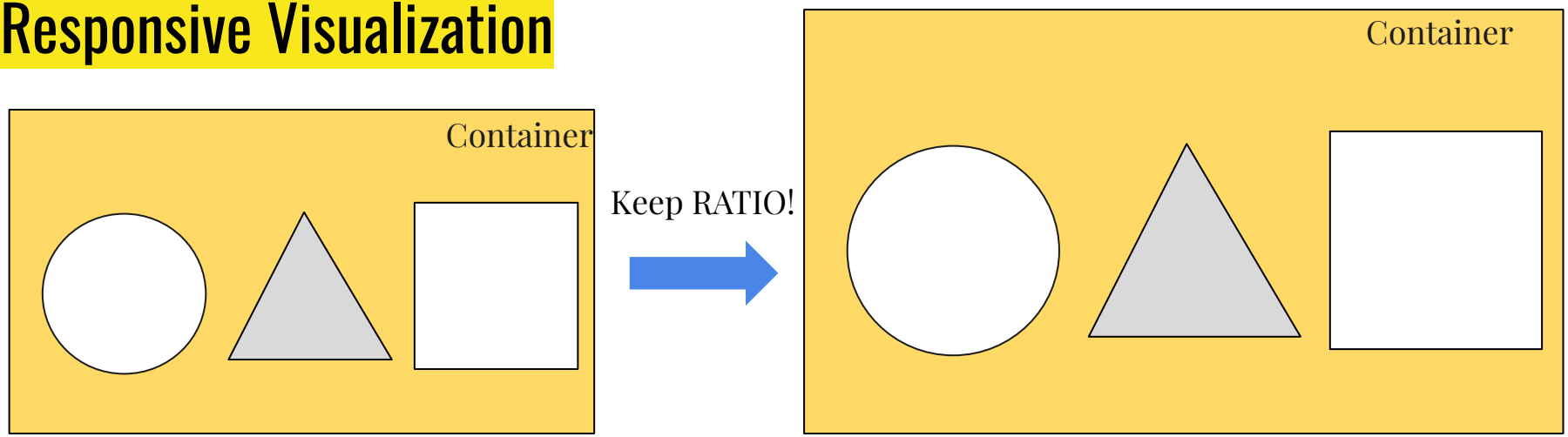
- Responsive visualization is the practice of designing and developing visualizations that adapt and respond to different screen sizes, ensuring an optimal viewing experience across devices.
- It focuses on creating visualizations that are flexible, scalable, and user-friendly, regardless of the device used to access them.

Adapting visualizations to different screen sizes

- With the increasing use of smartphones, tablets, and various screen sizes, it's crucial to ensure that visualizations remain accessible and usable on any device.
- Responsive visualizations allow users to interact with data and insights seamlessly, regardless of the screen real estate available.
- By adapting to different screen sizes, responsive visualizations cater to a wider audience and enhance user engagement.

Better UX!

Responsive Visualization



UBC-InfoVis/2021-436V-examples:
d3-responsive-scatter-plot - CodeSandbox

1.Adaption container changes manually

- Get the width of a div container (i.e., `<div id="scatterplot"></div>`):

```
config.containerWidth = document.getElementById('scatterplot').clientWidth;
```

- Update SVG size

```
svg
        .attr('width', config.containerWidth)
                .attr('height', config.containerHeight);
```

- Update inner chart size based on margin specifications:

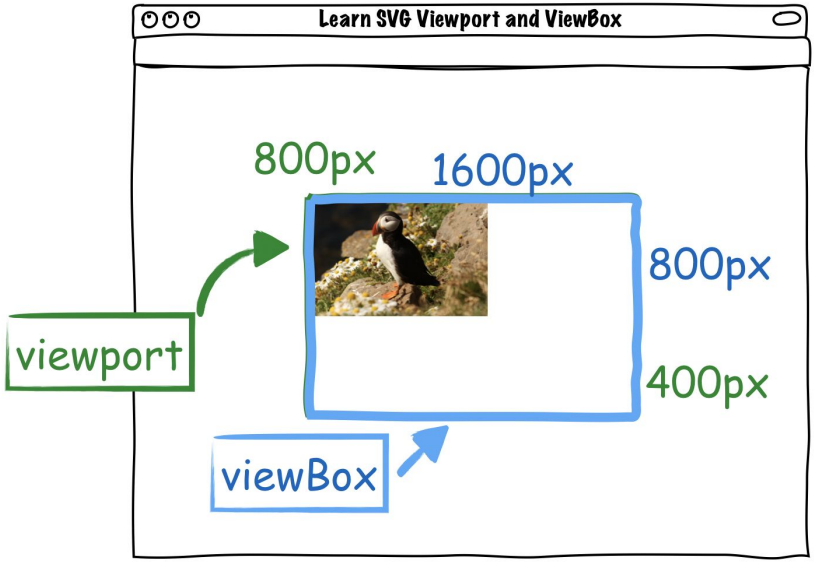
```
config.width = config.containerWidth - config.margin.left - config.margin.right;
config.height = config.containerHeight - config.margin.top - config.margin.bottom;
```

- Update scales, axes, and so on:

```
xAxisG
        .attr('transform', `translate(0,${config.height})`);
```

```
xScale
        .range([0, config.width])
        .domain([minData, maxData]);
```

Responsive Visualization - viewBox



- Scaling Scalable Vector Graphics (SVG) provide a powerful feature called the "viewBox" attribute.
- The viewBox defines the coordinate system and aspect ratio of the SVG content, allowing for responsive scaling.
- When the container size changes, the viewBox ensures that the visualization scales proportionally to fit the available space.

Viewport: The visible area or canvas on which the SVG content is displayed. It is defined by the width and height attributes of the <svg> element.

ViewBox: An attribute of the <svg> element that defines the coordinate system and aspect ratio of the SVG content. It specifies the initial coordinate system and the boundaries of the content that should be visible within the viewport. **The viewBox attribute takes four values: x, y, width, and height.**

viewBox - SVG: Scalable Vector Graphics | MDN (mozilla.org)

Responsive Visualization - viewBox

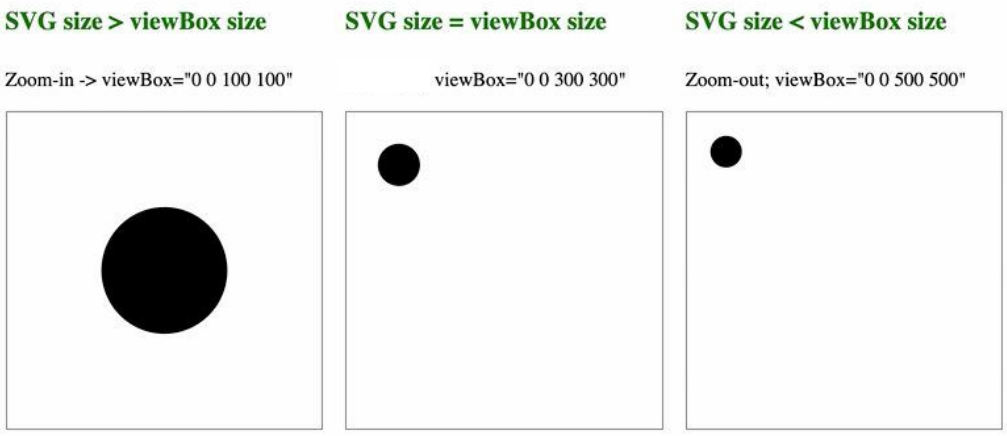
The numbers min-x and min-y represent the top left coordinates of the viewport

Min-x, Min-y

Width, Height

```
<svg width="300" height="300" viewBox="0 0 1500 1500">
```

viewbox · GitHub



This means:

horizontal axis: 1500 (your width unit) = 300px => 1 (your width unit) = 300/1500px = 1/5px
vertical axis: 1500 (your height unit) = 300px => 1 (your height unit) = 300/1500px = 1/5px

- Now all shapes in the svg will scale:

their widths scale to 1/5px (1/5 < 1 => scale down) comparing to the origin.
their heights also scale to 1/5px (1/5 < 1 => scale down) comparing to the origin

Scalable Visualization with Canvas

Introduction to Scalable Visualizations

- Scalable visualizations refer to visual representations of data that can handle large datasets and dynamically adapt to different screen sizes.
- While SVG (Scalable Vector Graphics) is commonly used for visualizations, Canvas provides an alternative approach with unique advantages.

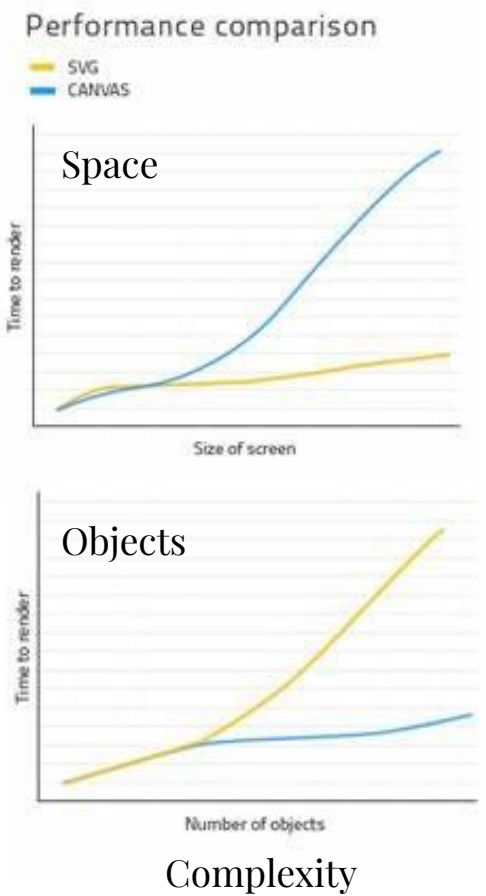
```
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");

ctx.fillStyle = "green";
ctx.fillRect(10, 10, 150, 100);
```

Canvas API - Web APIs | MDN (mozilla.org)

Scalable Visualization with Canvas

Advantages of Canvas for Scalable Visualizations

- Performance and Rendering Efficiency
  - Canvas utilizes immediate mode rendering, allowing direct manipulation of pixel data.
  - It can handle large datasets more efficiently than SVG, making it suitable for complex and dynamic visualizations.
- Flexibility and Customization
  - With Canvas, you have full control over every pixel, enabling custom rendering and advanced graphical effects.
  - It provides the flexibility to create unique visualizations and interactive experiences tailored to specific requirements.
- Responsiveness and Interactivity
  - Canvas can handle real-time updates and interactions more efficiently, making it ideal for interactive visualizations.
  - It enables smooth animations, responsive transitions, and interactive features without sacrificing performance.

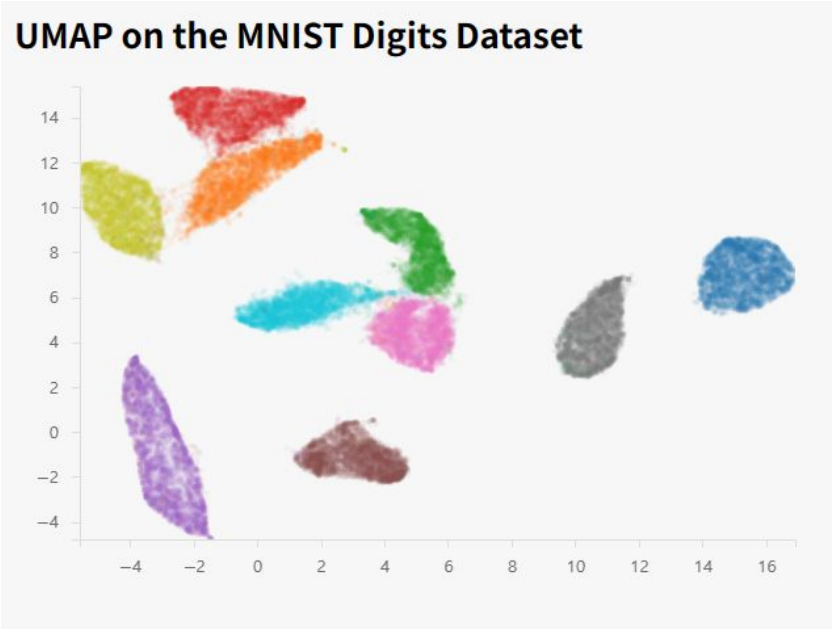


Scalable Visualization with Canvas

Considerations for Using Canvas in Scalable Visualizations

- Complexity and Learning Curve
  - Working with Canvas requires a solid understanding of JavaScript and graphics programming concepts.
  - It may have a steeper learning curve compared to SVG, but the flexibility and performance advantages make it worth considering.
- Browser Compatibility
  - Canvas is widely supported in modern browsers but requires JavaScript to render and interact with the visualizations.
  - Ensure compatibility with target browsers and consider fallback options for older or non-supported browsers.
- Use Cases and Examples
  - Canvas is well-suited for visualizations with large datasets, dynamic updates, complex animations, and custom graphical effects.
  - Showcase examples of scalable visualizations created with Canvas to inspire and demonstrate its capabilities.

UBC-InfoVis/2021-436V-examples: d3-canvas-scatter-plot - CodeSandbox



Scalable Visualization with Canvas

D3 and Canvas in 3 steps (freecodecamp.org)

Bind the elements

To bind data to the elements you first create a base element for all your custom elements you will produce and draw. If you know D3 well, think of it as a replacement to the SVG element:

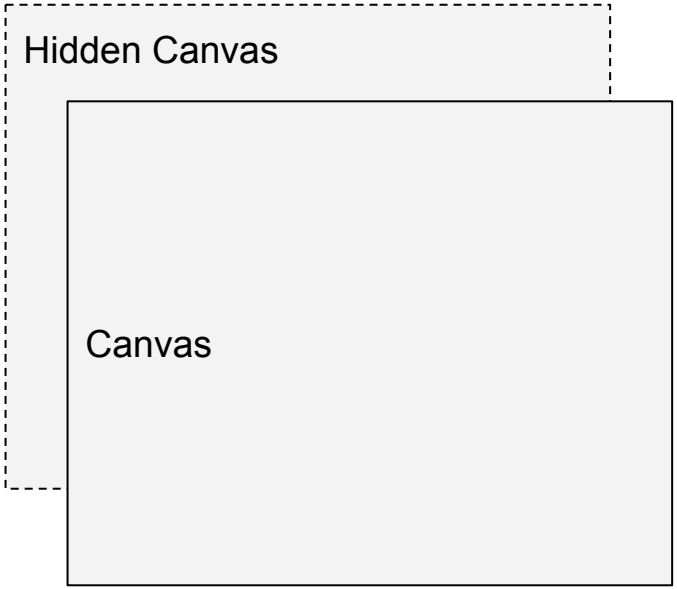
```
var customBase = document.createElement('custom');
```

```
var custom = d3.select(customBase); // This is your SVG replacement a the parent
```

Canvas

Binding

Custom tags



Overlapping

Capture Interaction from Canvas (mousemove)

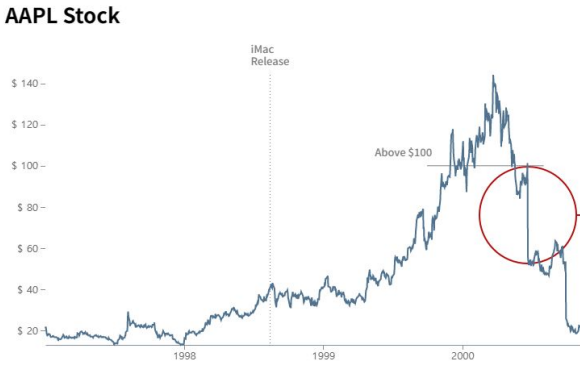
Using mouse point, find the objects from hidden canvas

Update visualization based on the changes of hidden canvas

Annotations

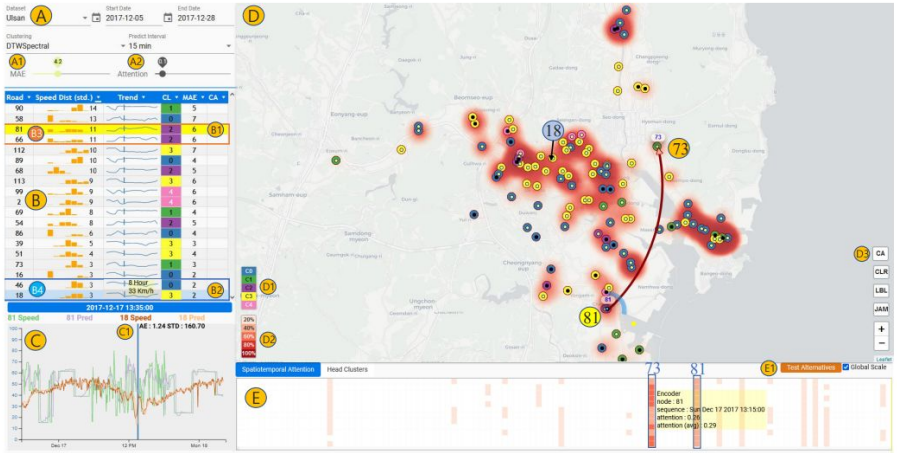
- Types of Annotations
  - Markers: Pointers, arrows, or symbols placed on specific data points or areas of interest.
  - Labels: Text annotations providing additional information or descriptions.
  - Lines and Connectors: Lines or paths connecting data points or groups, emphasizing relationships or comparisons.
  - Regions and Shapes: Shaded areas or shapes indicating specific regions or categories.
- Functions of Annotations
  - Data Explanation: Annotations clarify complex data patterns or outliers, making the visualization more informative.
  - Emphasis and Focus: Annotations draw attention to key insights, trends, or important data points.
  - Storytelling: Annotations help narrate a story, guiding the audience through the visualization and providing context.
- Design Considerations
  - **Placement and Alignment**: Annotations should be strategically placed to avoid clutter and ensure clear visual hierarchy.
  - **Styling and Contrast**: Annotations should be visually distinct from the data points, using color, font size, or other visual cues.
  - **Interactivity**: Interactive annotations allow users to toggle visibility or access additional information when needed.

UBC-InfoVis/2021-436V-examples: d3-annotated-line-chart - CodeSandbox

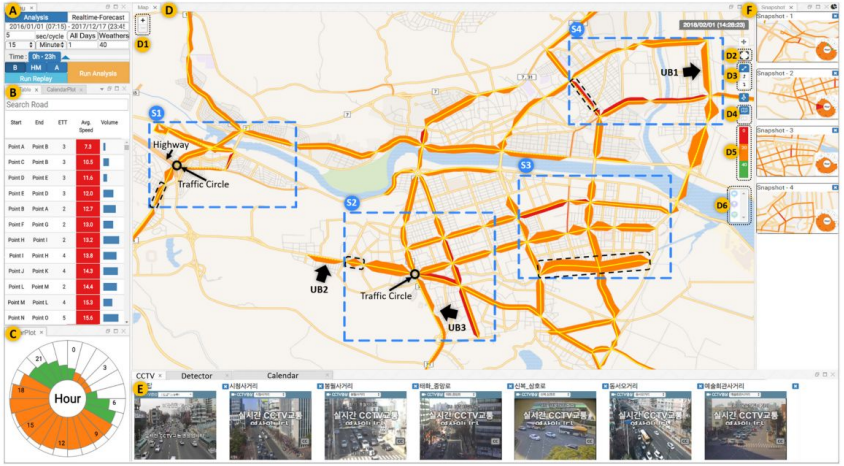


https://d3-annotation.susielu.com

Examples of real application of d3js + leaflet



S. Jin *et al.*, "A Visual Analytics System for Improving Attention-based Traffic Forecasting Models," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 1, pp. 1102-1112, Jan. 2023, doi: 10.1109/TVCG.2022.3209462.



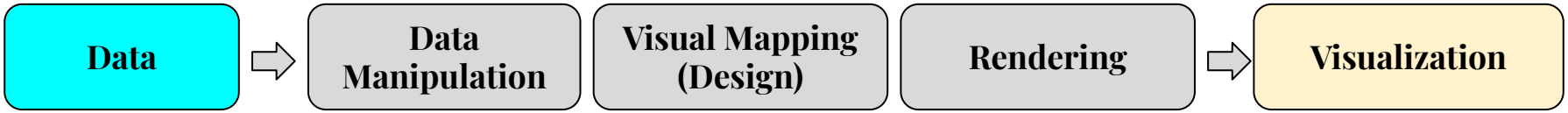
C. Lee *et al.*, "A Visual Analytics System for Exploring, Monitoring, and Forecasting Road Traffic Congestion," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 11, pp. 3133-3146, 1 Nov. 2020, doi: 10.1109/TVCG.2019.2922597.

Structured Prompting for Data Visualization

https://docs.google.com/document/d/19xwdo0A3uLjKKOWDAPjBzA62bTyne1ybK8jM_iA7qoA/edit?usp=sharing

When using AI tools like ChatGPT to assist with D3.js development, structuring your prompts according to the data visualization pipeline yields better results.

The Visualization Pipeline and Prompt Strategy



Stage-Based Prompting Approach

Prompt Structure Example:

I need a D3.js visualization that:
1. Data Manipulation: [describe data processing needs]
2. Visual Mapping: [describe visual encoding requirements]
3. Rendering: [specify rendering approach]

Kaggle Data Visualization Exercise

https://docs.google.com/document/d/12IynQxBxLK7qcemVdaV9DPIl2DpgacJ4tEqZrQE3wJg/edit?usp=sharing

Exercise Overview

Objective: Create an interactive D3.js visualization using a simple Kaggle dataset

Time: Approximately 20 minutes

Difficulty: Beginner-Intermediate

Dataset: "Chocolate Sales Data" (alternative options provided)

Submit the prompt (text format) and code with the compression: dryjins@gmail.com

Reference

- Harvard's visualization course (CS171)
- Codepen
- D3.js
- Starter project