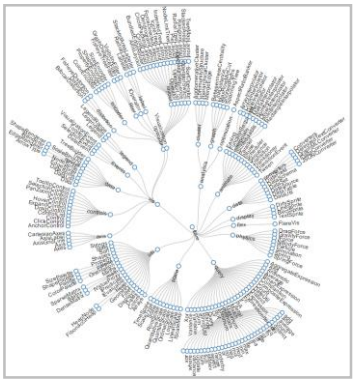
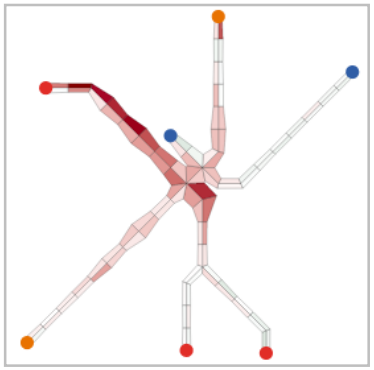
Introduction to D3.js

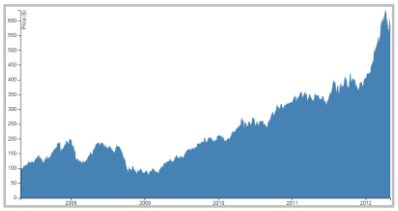
Tan-Chi Ho

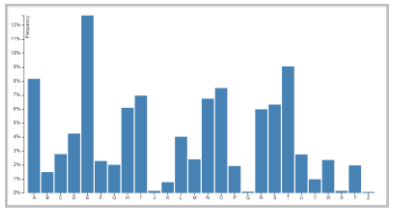
D3.js Introduction

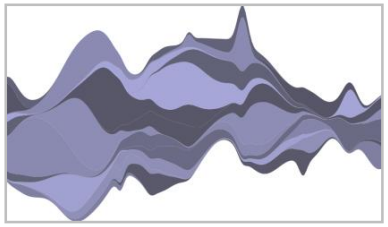
- A JavaScript library for manipulating documents based on data.
- 𝔻3 = **D**ata-**D**riven **D**ocuments

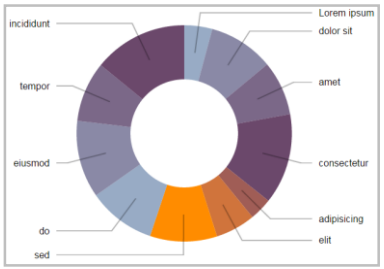


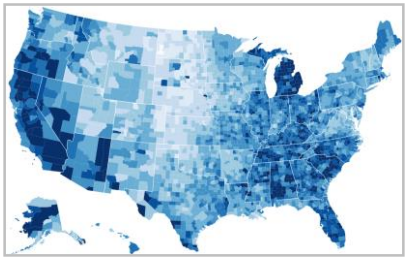


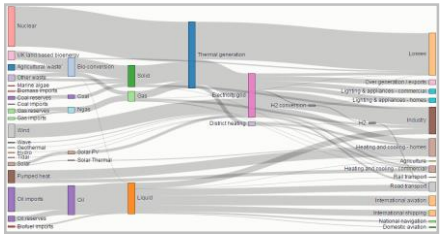


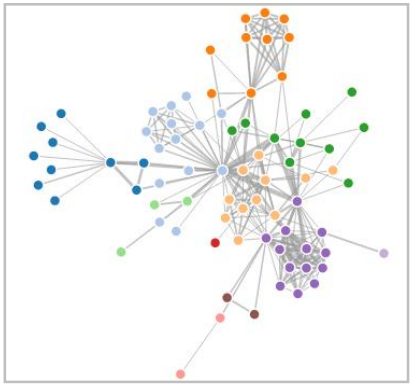








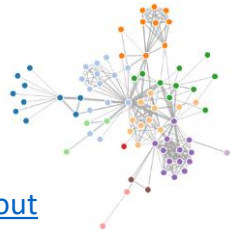




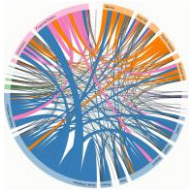
D3.js Introduction

- More Examples

Relations (graph、relationship)

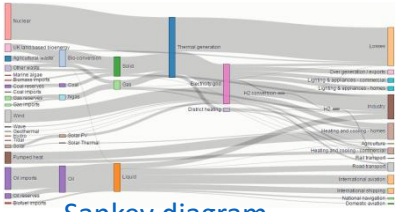


Graph layout

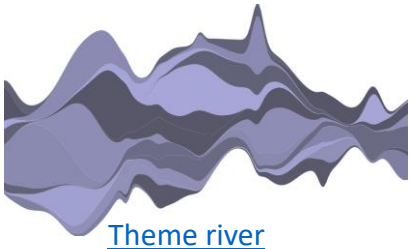


Chord diagram

Temporal Stream / Flow



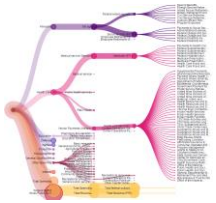
Sankey diagram



Theme river

Hierarchies (tree、hierarchy)

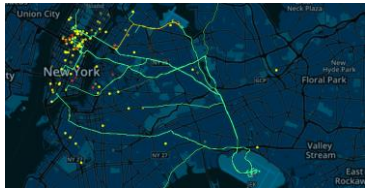
Radial Reingold-Tilford tree



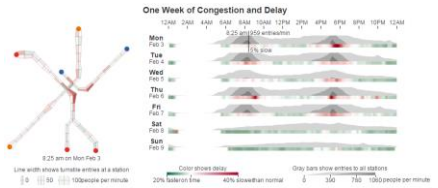
Tree expansion

Sunburst chart

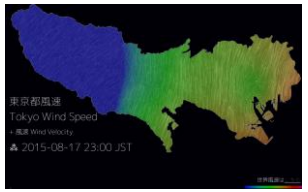
Animation



NYC flight-taxi Vis



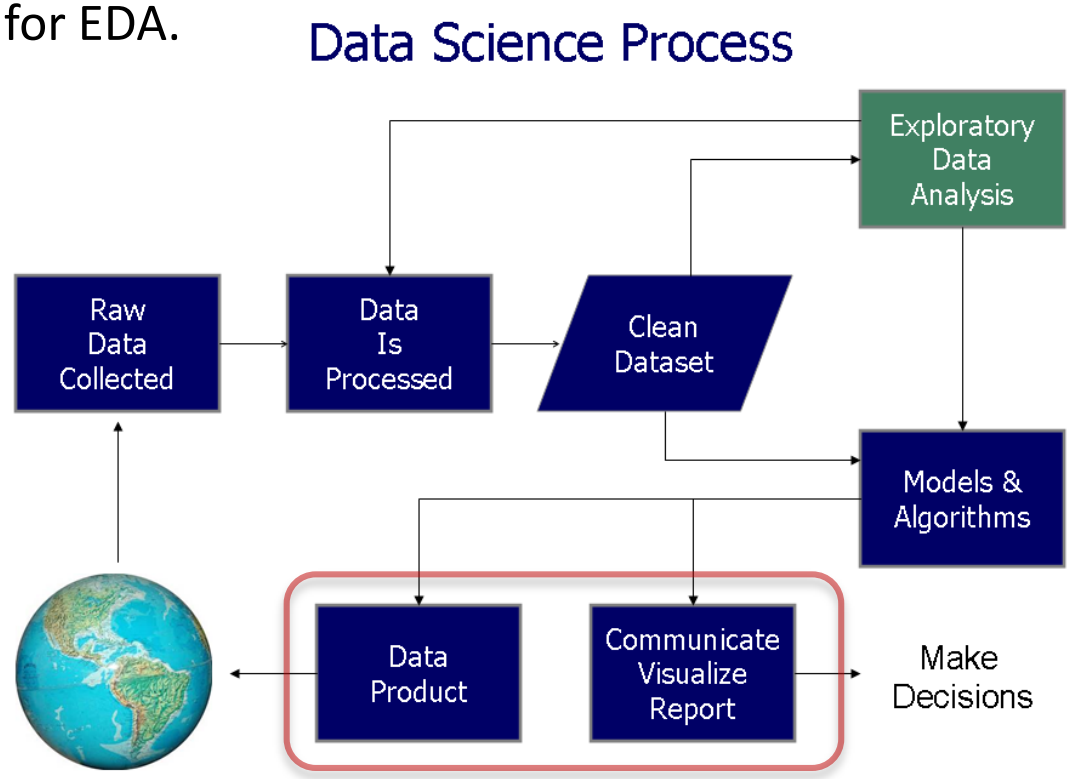
MBTA Vis



東京都風速

D3.js Introduction

- Position of D3.js in Data Science Process Pipeline
  - For documentation / presentation
    - Not suite for EDA.

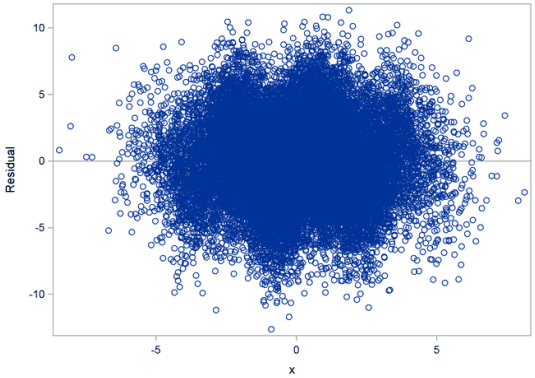


-- Wikipedia --

D3.js Introduction

- Pros.
  - Highly customized graphical charts
    - Through HTML, SVG, CSS
    - Various layouts
    - Animation / transition
  - Web based
  - Unstructured data support (json)
    - Graph (edges, nodes), Hierarchy (nested), sequential data, …
- Cons.
  - Graphics programming required
  - Poor performance
    - Limit to browser capability
      - Ex. A scatter plot with 100K samples

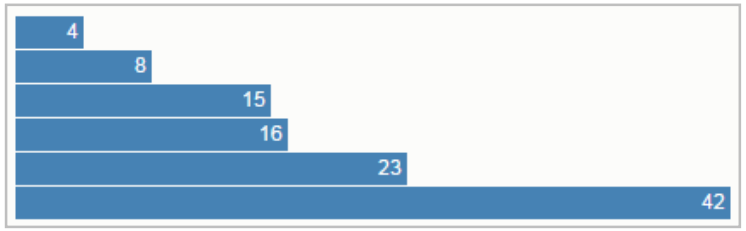
Layouts in D3.js



D3.js Introduction

- Make a simple bar chart

HTML + JavaScript + CSS



HTML

```
<svg class="chart"></svg>
```

JavaScript + D3 4.13.0

```
var data = [4, 8, 15, 16, 23, 42];
var width = 420,
    barHeight = 20;

var x = d3.scaleLinear()
    .domain([0, d3.max(data)])
    .range([0, width]);

var chart = d3.select(".chart")
    .attr("width", width)
    .attr("height", barHeight * data.length);

var bar = chart.selectAll("g").data(data)
  .enter().append("g")
    .attr("transform", function(d, i) { return "translate(0," + i * barHeight + ")"; });

bar.append("rect")
    .attr("width", x)
    .attr("height", barHeight - 1);

bar.append("text")
    .attr("x", function(d) { return x(d) - 3; })
    .attr("y", barHeight / 2)
    .attr("dy", ".35em")
    .text(function(d) { return d; });
```

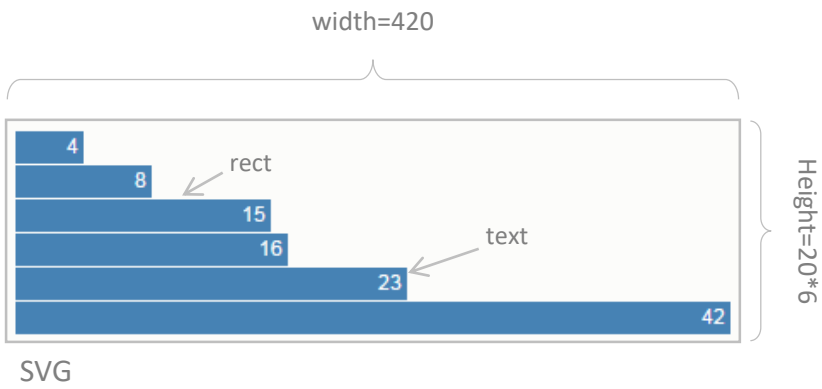
CSS

```
.chart rect {
  fill: steelblue;
}
.chart text {
  fill: white;
  font: 10px sans-serif;
  text-anchor: end;
}
```

D3.js Introduction

- Make a simple bar chart

HTML + JavaScript + CSS



HTML

```
<svg class="chart"></svg>
```

SVG

JavaScript + D3 4.13.0

```
var data = [4, 8, 15, 16, 23, 42];
var width = 420,
    barHeight = 20;

var x = d3.scaleLinear()
    .domain([0, d3.max(data)])
    .range([0, width]);

var chart = d3.select(".chart")
    .attr("width", width)
    .attr("height", barHeight * data.length);
```

SVG

transformation

```
var bar = chart.selectAll("g").data(data)
  .enter().append("g")
    .attr("transform", function(d, i) { return "translate(0," + i * barHeight + ")"; });

bar.append("rect")
    .attr("width", x)
    .attr("height", barHeight - 1);
```

rect

```
bar.append("text")
    .attr("x", function(d) { return x(d) - 3; })
    .attr("y", barHeight / 2)
    .attr("dy", ".35em")
    .text(function(d) { return d; });
```

text

CSS

```
.chart rect {
  fill: steelblue;
}
.chart text {
  fill: white;
  font: 10px sans-serif;
  text-anchor: end;
}
```

Data

- Example of Data in JavaScript
  - Array of values

```
var data = [4, 8, 15, 23, 42];
```

  - Object

```
var person = {
    firstName:”John”,
    lastName:”Doe”,
    age:50,
    eyeColor:”blue”
};
```

  - Array of objects

```
var aqi = {
    {datetime:”2019-10-01”, town:”竹東”, aqi:20},
    {datetime:”2019-10-02”, town:”竹北”, aqi:23},
    {datetime:”2019-10-03”, town:”竹南”, aqi:37}
};
```

- API for Data Loading
  - d3.csv(url [, row, callback]);
    - Read CSV file as array of JS objects.
  - d3.json(url[, row, callback]);
    - Read JSON file as JS object.
  - …

```
d3.csv("/data/鄉鎮人口統計.csv", function(data) {
    for (var i = 0; i < data.length; i++) {
        console.log(data[i].date);
        console.log(data[i]["縣市"]);
        console.log(data[i]["人口總"]);
    }
});
```

Notice that the data loading APIs are async. functions. Always put the data process (and the corresponding visualization) routines in the callback function.

Selections

- Select specific DOM elements
  - d3.select("svg")
    - Select (first) SVG element.
  - d3.selectAll("td")
    - Select all "TD" elements (table data).
  - d3.selectAll(".my_class")
    - Select all DOM elements with class name my_class.
  - d3.selectAll("#my_id")
    - Select all DOM elements with ID=my_id.

- Cascaded selection
  - d3.select("svg").selectAll("rect")
    - Select all rect (rectangles) in SVG.

Selections

- Update DOM properties with selection.

DOM attribute :
- Set class name “my_var”.
- Set rect X position to 0px.

CSS style :
- Set fill color to red.

```
d3.selectAll(“rect”)
    .attr(“class”, “my_bar”)
    .attr(“x”, “0px”)
    .style(“fill”, “red”);
```

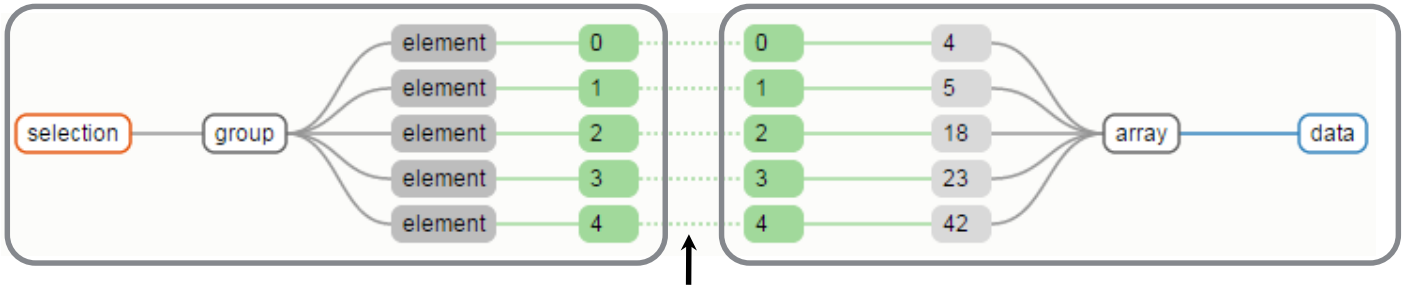
- Append DOM
  - d3.select(“svg”).append(“rect”)
    - Append a rect (rectangle) on “SVG”.

Selections

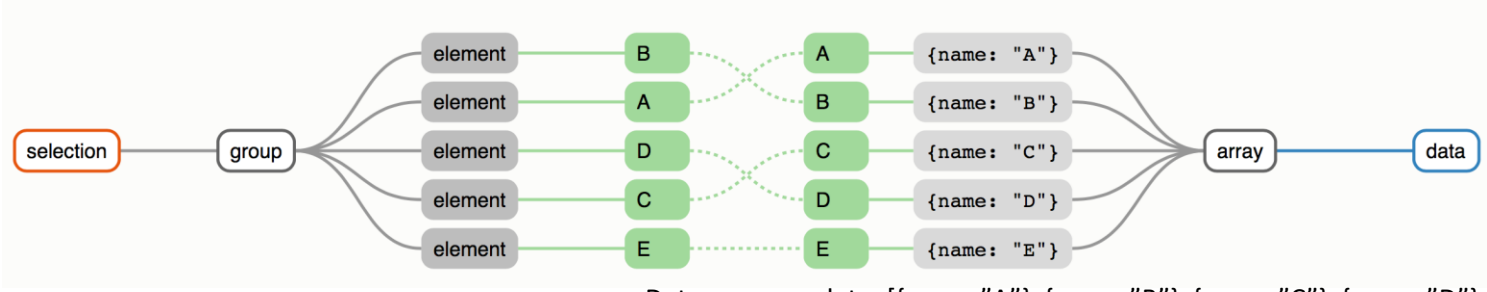
- Bind data with DOM elements

DOM elements (rect.width, td.text, …)

Data : var my_data=[4, 5, 18, 23, 42]



d3.selectAll("element").data(my_data)



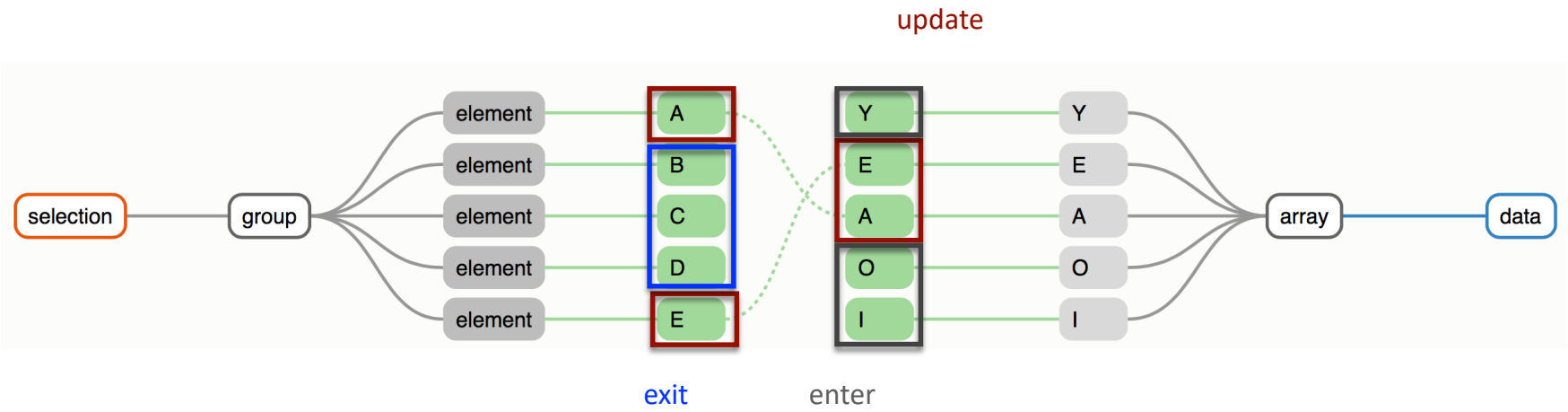
Data : var my_data=[{name:"A"}, {name:"B"}, {name:"C"}, {name:"D"}, {name:"E"}]

Name binding :

d3.selectAll("element").data(my_data, function(d){return d.name;})

Selections

- Mapping between data and DOM elements
  - Update : DOMs that match data
    - Matched elements
  - Enter : Missing DOMs in data
    - Elements to be created (appended)
  - Exit : DOMs that have no data matched
    - Elements to be removed

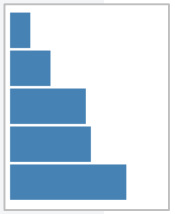


Selections

- Example of data mapping

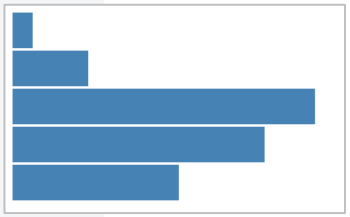
```
var my_data=[
        {name:"A", value:4}, {name:"B", value:8}, {name:"C", value:15},
        {name:"D", value:16}, {name:"E", value:23}, {name:"F", value:42}
    ];
var allRects = d3.select("svg").selectAll("rect").data(my_data, function(d){return d.name});

allRects.enter().append("rect")
    .attr("x", 0)
    .attr("y", function(d,i){return i*30})
    .attr("width", function(d){return d.value*4;})
    .attr("height", 28);

// ================================================================================

var my_data2=[
        {name:"A", value:4}, {name:"C", value:15}, {name:"E", value:60},
        {name:"I", value:50}, {name:"J", value:33}, {name:"K", value:23}
    ];

var allRects = d3.select("svg").selectAll("rect").data(my_data2, function(d){return d.name});

allRects
    .attr("y", function(d,i){return i*30})
    .attr("width", function(d){return d.value*4;})
    .attr("height", 28);

allRects.enter().append("rect")
    .attr("x", 0)
    .attr("y", function(d,i){return i*30})
    .attr("width", function(d){return d.value*4;})
    .attr("height", 28);

allRects.exit().remove();
```

Create new rects



Update old rects

Create missing rects

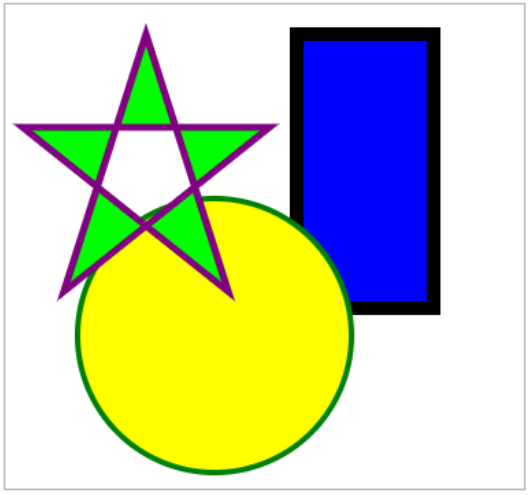


Remove rects

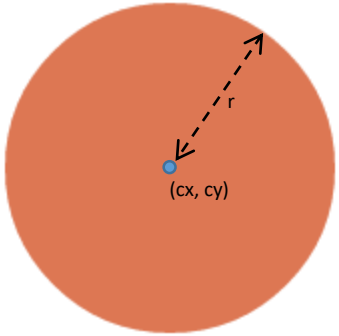
Introduction to SVG

- **S**calable **V**ector **G**raphics
  - A vector image format of 2D graphics
    - XML based
    - W3C standard, supported by all modern browsers

```
<svg class="chart" width="500" height="500">
  <rect x="210" y="10" width="100" height="200"
      style="fill:rgb(0,0,255);stroke-width:10;stroke:rgb(0,0,0)" />
  <circle cx="150" cy="230" r="100"
      stroke="green" stroke-width="4" fill="yellow" />
  <polygon points="100,10 40,198 190,78 10,78 160,198"
      style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />
</svg>
```

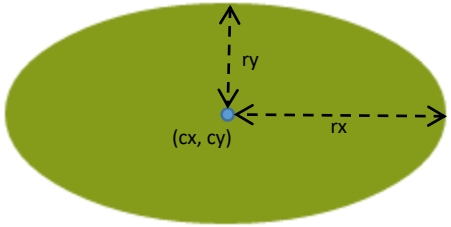


Introduction to SVG - Shape



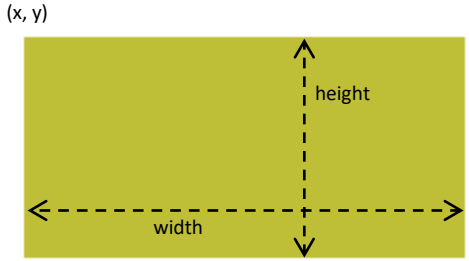
Circle

```
<circle cx="75" cy="75" r="75"
   fill="#ED6E46" />
```



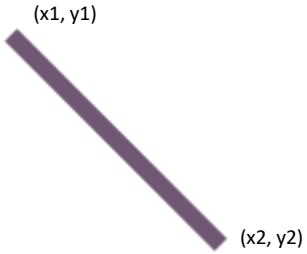
Ellipse

```
<ellipse cx="100" cy="100" rx="100" ry="50“
   fill="#7AA20D" />
```



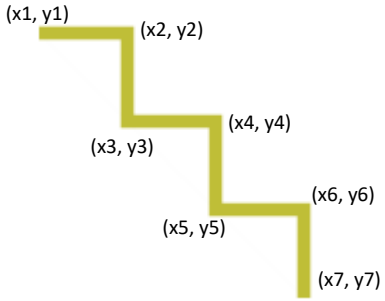
Rectangle

```
<rect x=“10" y=“10" width="200" height="100“
   fill="#BBC42A" />
```



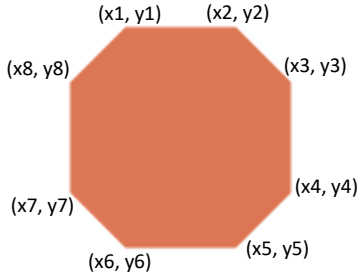
Line

```
<line x1="5" y1="5" x2="100" y2="100“
   stroke="#765373" stroke-width="8"/>
```



Polyline

```
<polyline points="0,40 40,40 40,80 80,80 80,120 120,120 120,160“
   fill="white" stroke="#BBC42A" stroke-width="6" />
```



Polygon

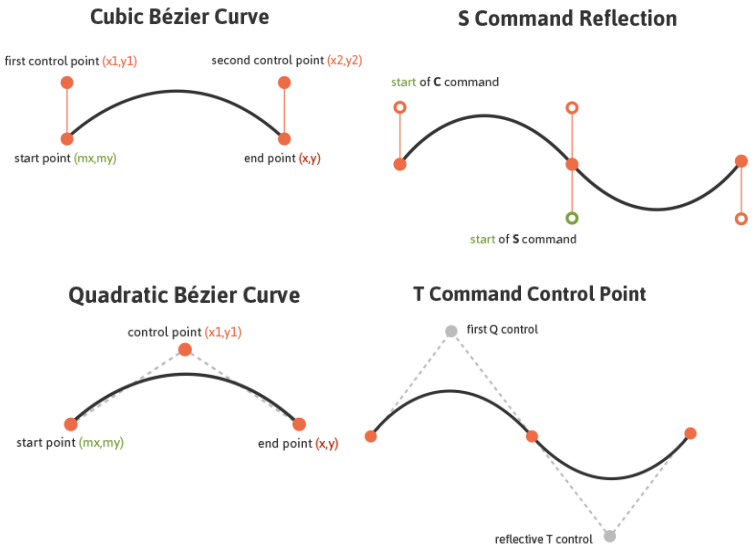
```
<polygon points="50,5 100,5 125,30 125,80 100,105 50,105 25,80 25,30“
   fill="#ED6E46" />
```

Introduction to SVG - Shape

- Path
  - A generalized approach for defining the outline of a shape
    - Straight lines
    - Curves (cubic or quadratic)
  - General form

    ```
    <path d="<path data specifics>" />
    ```

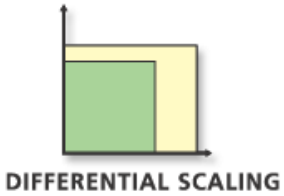
  - Example

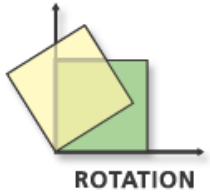


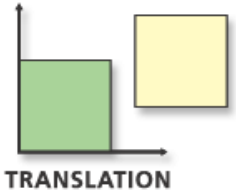

```
<path fill="#7AA20D" stroke="#7AA20D" stroke-width="9" stroke-linejoin="round"
d="M248.761,92c0,9.801-7.93,17.731-17.71,17.731c-0.319,0-0.617,0-0.935-0.021c-10.035,37.291-51.174,65.206-
100.414,65.206 c-49.261,0-90.443-27.979-100.435-65.334c-0.765,0.106-1.531,0.149-2.317,0.149c-9.78,0-17.71-7.93-
17.71-17.731 c0-9.78,7.93-17.71,17.71-
17.71c0.787,0,1.552,0.042,2.317,0.149C39.238,37.084,80.419,9.083,129.702,9.083
c49.24,0,90.379,27.937,100.414,65.228h0.021c0.298-0.021,0.617-0.021,0.914-
0.021C240.831,74.29,248.761,82.22,248.761,92z" />
```

Introduction to SVG - Transformation

- Type of transformations
  - Translate
    - Move the shape along x and y axis
      - transform="translate(<tx>,<ty>)"
  - Rotate
    - Rotate the shape with rotation angle (in degree) along rotation center (cx,cy) ((0,0) by default).
      - transform="rotate(<rotation angle> [<cx>,<cy>])"
  - Scale
    - Resize the shape
      - transform="scale(<sx> [<sy>])"
  - Skew
    - Skew along X-axis
      - transform="skewX(20)“
    - Skew along Y-axis
      - transform="skewY(20)"



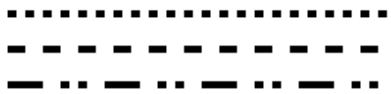




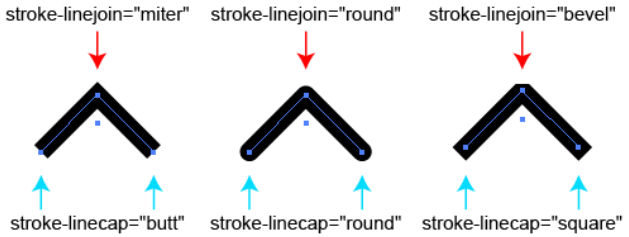
- Multiple transformations
  - transform="translate(10,30) rotate(30) scale(0.5)“
    - Be aware of the order of transformations
- Transformation can be applied to any shape, or group object <g>
  - Group of shapes

Introduction to SVG - Style

- Fills and Strokes
  - fill : specify color of the interior of shape
    - fill-opacity
      - opacity (0~1) of the fill color
  - stroke : specify color of the border of shape
    - stroke-opacity
      - opacity (0~1) of the fill color
    - stroke-width
      - width of stroke
    - stroke-dasharray
      - type of dashlines
    - stroke-linecap / stroke-linejoin
      - type of endings and corners

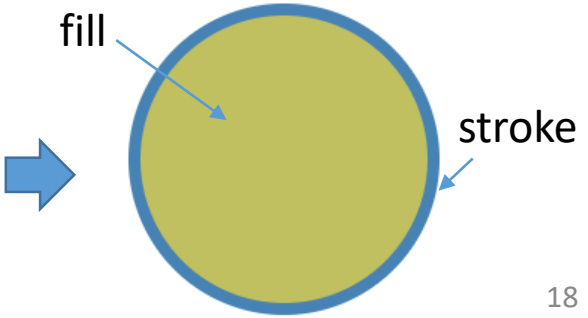


Type of dashlines



- How to specify color
  - Red color can be specified as
    - “red” (HTML color name)
    - #0000ff
    - rgb(0,0,255)

```
<circle cx="150" cy="230" r="100"
   stroke="SteelBlue" stroke-width="8" fill="#C0C060" />
```



Practice Platform

- How to practice?
  - Local HTML file
    - Unable to access local data file (d3.csv(), d3.json()).
  - Web server
    - Apache, IIS, …
    - Python simple http server
      - python –m http.server
  - Online editor
    - Jsfiddle : https://jsfiddle.net/
    - Codepen : https://codepen.io/

```
<html>
<script src="https://d3js.org/d3.v4.min.js"></script>
<body>
</body>
</html>

<script type="text/javascript">
… <-- put JS code here. -->
</script>
```

Be aware to add HTML bodies and include the d3.js library.

Let's Make a Scatter Plot

- Create a SVG canvas

Create SVG under the HTML <body> element.

Give a class name

```
var svg = d3.select("body")
  .append("svg")
      .attr("class","my_canvas")
      .attr("width","600px")
      .attr("height","400px");
```

Specify SVG width, height = (600px, 400px)

```
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▶<script>…</script>
      <svg class="my_canvas" width="600px" height="400px"></svg>
  </body>
</html>
```

DOM elements (in debugger)

- Read data

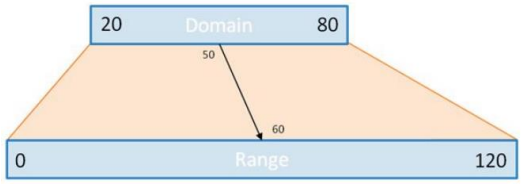
```
d3.csv("https://raw.githubusercontent.com/holtzy/data_to_viz/master/Example_dataset/2_TwoNum.csv",
  function(data) {
    // put the data dependent routines here.
  })
```

Put all data dependent routines in the callback function.

```
▼data: Array(1460)
  ▶[0 … 99]
  ▼[100 … 199]
    ▶100: {GrLivArea: "1610", SalePrice: "205000"}
    ▶101: {GrLivArea: "1732", SalePrice: "178000"}
    ▶102: {GrLivArea: "1535", SalePrice: "118964"}
    ▶103: {GrLivArea: "1226", SalePrice: "198900"}
    ▶104: {GrLivArea: "1818", SalePrice: "169500"}
    ▶105: {GrLivArea: "1992", SalePrice: "250000"}
    ▶106: {GrLivArea: "1047", SalePrice: "100000"}
    ▶107: {GrLivArea: "789", SalePrice: "115000"}
    ▶108: {GrLivArea: "1517", SalePrice: "115000"}
    ▶109: {GrLivArea: "1844", SalePrice: "190000"}
    ▶110: {GrLivArea: "1855", SalePrice: "136900"}
    ▶111: {GrLivArea: "1430", SalePrice: "180000"}
    ▶112: {GrLivArea: "2696", SalePrice: "383970"}
    ▶113: {GrLivArea: "2259", SalePrice: "217000"}
    ▶114: {GrLivArea: "2320", SalePrice: "259500"}
    ▶115: {GrLivArea: "1458", SalePrice: "176000"}
    ▶116: {GrLivArea: "1092", SalePrice: "139000"}
    ▶117: {GrLivArea: "1125", SalePrice: "155000"}
    ▶118: {GrLivArea: "3222", SalePrice: "320000"}
    ▶119: {GrLivArea: "1456", SalePrice: "163990"}
    ▶120: {GrLivArea: "988", SalePrice: "180000"}
    ▶121: {GrLivArea: "1123", SalePrice: "100000"}
    ▶122: {GrLivArea: "1080", SalePrice: "136000"}
    ▶123: {GrLivArea: "1199", SalePrice: "153900"}
    ▶124: {GrLivArea: "1586", SalePrice: "181000"}
    ▶125: {GrLivArea: "754", SalePrice: "84500"}
```

Data view (in debugger)

Let's Make a Scatter Plot



- Set scales
  - Functions to map from a domain (data) to a range (vis)

```
// convert from string to int
data.forEach(function(d){
  d.GrLivArea = parseInt(d.GrLivArea);
  d.SalePrice = parseInt(d.SalePrice);
});

// get the min/max value of data
var GrLivArea_Range = d3.extent(data, function(d){return d.GrLivArea;});
var SalePrice_Range = d3.extent(data, function(d){return d.SalePrice;});

// map "GrLivArea" to X-axis
var x_scale = d3.scaleLinear()
  .domain([0, GrLivArea_Range[1]])
  .range([ 0, 600]);
                SVG width
// map "SalePrice" to Y-axis
var y_scale = d3.scaleLinear()
  .domain([0, SalePrice_Range[1]])
  .range([ 0, 400]);
                SVG height
```

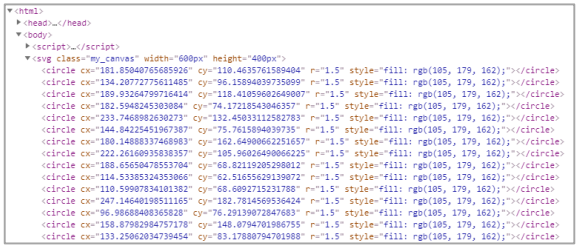
Data manipulation

Map GrLiveArea to X-axis

Map SalePrice to Y-axis

Let's Make a Scatter Plot

- Plot the dots



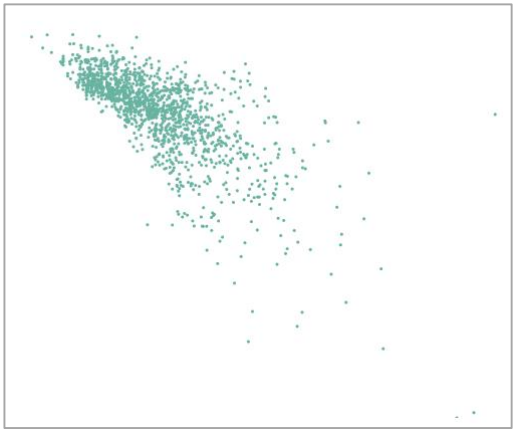
DOM elements (in debugger)

Get SVG element

Bind ".dot" with data

```
d3.select("svg").selectAll(".dot").data(data)
  .enter().append("circle")
    .attr("class", "dot")
    .attr("cx", function (d) { return x_scale(d.GrLivArea); } )
    .attr("cy", function (d) { return y_scale(d.SalePrice); } )
    .attr("r", 1.5)
    .style("fill", "#69b3a2");
```

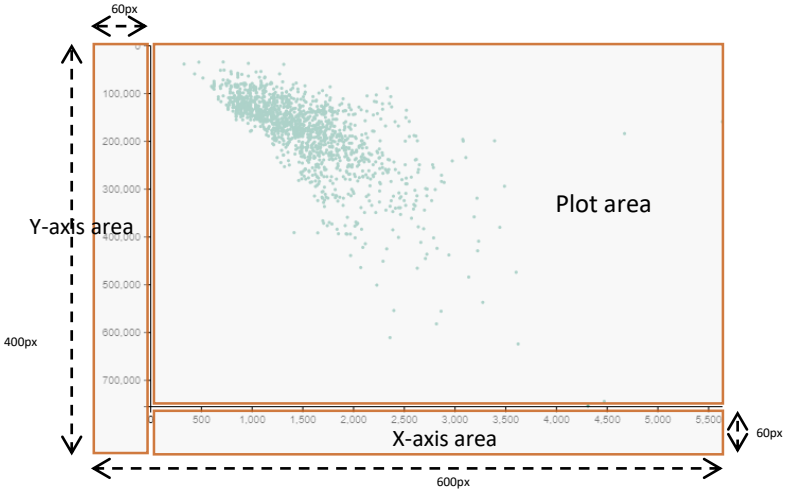
Append new circles, set class name as "dot"



Plot result

Let's Make a Scatter Plot

- Add axes
  1. Create 3 transformation groups
  2. Set scale ranges to new plot area (540px, 340px).
  3. Append axes with new scales.



rescale

3 transformation groups

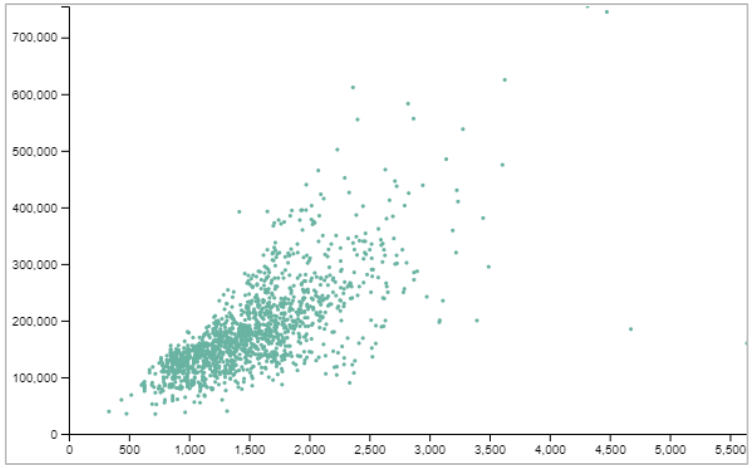
```
// map "GrLiveArea" to X-axis
var x_scale = d3.scaleLinear()
                .domain([0, GrLivArea_Range[1]])
                .range([0, 600-60]);

// map "salePrice" to Y-axis
var y_scale = d3.scaleLinear()
                .domain([0, SalePrice_Range[1]])
                .range([0, 400-60]);

// add plot canvas
d3.select("svg").append("g")
  .attr("class", "new_canvas")
  .attr("transform", "translate(60, 0)")
                                        Translate to (60, 0)
// Add X axis
d3.select("svg").append("g")
  .attr("transform", "translate(60, 340)")
  .call(d3.axisBottom(x_scale));
                                        Translate to (60, 340)
                                        Append X-axis
// Add Y axis
d3.select("svg").append("g")
  .attr("transform", "translate(60, 0)")
  .call(d3.axisLeft(y_scale));
                                        Translate to (60, 0)
                                        Append Y-axis
// Add dots
d3.select(".new_canvas").selectAll(".dot").data(data)
  .enter().append("circle")
    .attr("cx", function (d) { return x_scale(d.GrLivArea); } )
    .attr("cy", function (d) { return y_scale(d.SalePrice); } )
    .attr("r", 1.5)
    .style("fill", "#69b3a2");
```

Append dots under transformation group

Let's Make a Scatter Plot

- Correct Y_scale
  - Reverse the range.

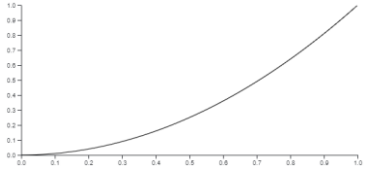
```
// map "salePrice" to Y-axis
var y_scale = d3.scaleLinear()
                .domain([0, SalePrice_Range[1]])
                .range([400-60, 0]);
```



Let's Make a Scatter Plot

- More about Scales
  - Continuous scale
    - d3. scaleLinear(), d3.scaleSqrt(), d3.scalePow(), …
    - Type of mapping
      - Value to value (size, position, …)
      - Value to color

        d3.scaleSqrt().range(["blue", "red"])



d3.scalePow()

    - Diverging scale
      - Data that have different scale in positive and negative.

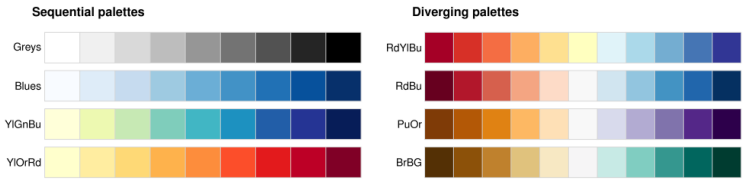
        d3.scaleDiverging().domain([extent[0], 0, extent[1]])

  - Discrete (ordinal) scale
    - For categorical (ordinal) values

      D3.scaleOrdinal()

      .domain(["A","B","C"])

      .range(["red","blue","green"])
    - One-to-one mapping between domain and range



Linear (sequential) vs Diverging color patterns

Reference : https://github.com/d3/d3-scale

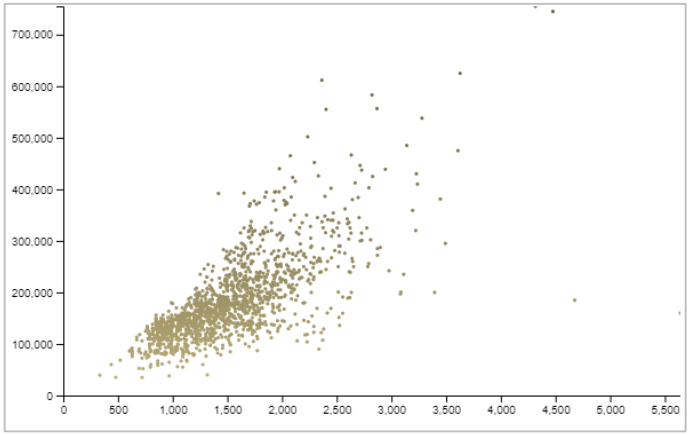
Let's Make a Scatter Plot

- Add color mapping to dots

Define color scale function

```
var color_scale = d3.scaleSqrt()
                .domain([0, SalePrice_Range[1]])
                .range(["#D4C685", "#696047"])

// Add dots
d3.select(".new_canvas").selectAll(".dot").data(data)
  .enter().append("circle")
    .attr("cx", function (d) { return x_scale(d.GrLivArea); } )
    .attr("cy", function (d) { return y_scale(d.SalePrice); } )
    .attr("r", 1.5)
    .style("fill", function(d) { return color_scale(d.SalePrice);});
```

Fill with color scale function



Interact and Animation

DOM Events

- Set of notify code of interesting things that have taken place.
  - Associated with DOM elements.
  - Callback functions are provided for event handling.

- Popular events
  - Mouse events
    - click / dblcick
    - mouseover / mouseout
    - wheel
  - Keyboard events
    - Keydown, keypress, keyup, …

Interact with Chart

- How to interact with DOM elements
  1. Set event handlers on source elements
  2. Update target elements (in callback functions)

- Assign event handlers with D3js

  selection.on(eventname [, listener [, options]])

On mouseover :
Update the selected dot color to "steelblue" and enlarge the dot.

On mouseout :
Reset the selected dot color and radius.

```
// Add dots
d3.select(".new_canvas").selectAll(".dot").data(data)
  .enter().append("circle")
    .attr("class", "dot")
    .attr("cx", function (d) { return x_scale(d.GrLivArea); } )
    .attr("cy", function (d) { return y_scale(d.SalePrice); } )
    .attr("r", 1.5)
    .style("fill", function(d) { return color_scale(d.SalePrice);})
    .on('mouseover', function(d){
        d3.select(this)
        .transition().duration(300)
            .style("fill", "steelblue")
                .attr("r", 5);
    })
    .on('mouseout', function(d){
        d3.select(this)
        .transition().duration(1000)
            .style("fill", function(d) { return color_scale(d.SalePrice);})
            .attr("r", 1.5);
    })
```

Interact with Chart

- Zooming
  - Zooming = mouse events + transformation
    - Mouse wheel -> scaling
    - Mouse move -> translation

Define zoom function

Bind zoom function to element

```
// define zoom function
var zooming = d3.zoom()
    .scaleExtent([0.5, 10])
    .extent([[0, 0], [540, 340]])
    .translateExtent([[-30, -30], [540+30, 340+30]])
    .on("zoom", zoom_callback);

// bind zoom function to element (dot group)
d3.select(".new_canvas").call(zooming);

// zoom update function
function zoom_callback() {
    // compute the transformation
    var newXScale = d3.event.transform.rescaleX(x_scale);
    var newYScale = d3.event.transform.rescaleY(y_scale);

    // update axes with these new boundaries
    d3.selectAll(".x-axis").call(d3.axisBottom(newXScale));
    d3.selectAll(".y-axis").call(d3.axisLeft(newYScale));

    // update circle position
    d3.select(".new_canvas").selectAll(".dot")
      .attr("cx", function (d) { return newXScale(d.GrLivArea); } )
      .attr("cy", function (d) { return newYScale(d.SalePrice); } )
  }
```

Interact with Chart

- Zooming
  - Zooming = mouse events + transformation
    - Mouse wheel -> scaling
    - Mouse move -> translation

Define zoom function

Bind zoom function to element

```
// define zoom function
var zooming = d3.zoom()
    .scaleExtent([0.5, 10])
    .extent([[0, 0], [540, 340]])
    .translateExtent([[-30, -30], [540+30, 340+30]])
    .on("zoom", zoom_callback);

// bind zoom function to element (dot group)
d3.select(".new_canvas").call(zooming);

// zoom update function
function zoom_callback() {
    // compute the transformation
    var newXScale = d3.event.transform.rescaleX(x_scale);
    var newYScale = d3.event.transform.rescaleY(y_scale);

    // update axes with these new boundaries
    d3.selectAll(".x-axis").call(d3.axisBottom(newXScale));
    d3.selectAll(".y-axis").call(d3.axisLeft(newYScale));

    // update circle position
    d3.select(".new_canvas").selectAll(".dot")
      .attr("cx", function (d) { return newXScale(d.GrLivArea); } )
      .attr("cy", function (d) { return newYScale(d.SalePrice); } )
  }
```

Callback function for zooming behavior

Interact with Chart

- Zooming
  - Zooming = mouse events + transformation
    - Mouse wheel -> scaling
    - Mouse move -> translation

Define zoom function

Bind zoom function to element

```
// define zoom function
var zooming = d3.zoom()
    .scaleExtent([0.5, 10])
    .extent([[0, 0], [540, 340]])
    .translateExtent([[-30, -30], [540+30, 340+30]])
    .on("zoom", zoom_callback);

// bind zoom function to element (dot group)
d3.select(".new_canvas").call(zooming);

// zoom update function
function zoom_callback() {
    // compute the transformation
    var newXScale = d3.event.transform.rescaleX(x_scale);
    var newYScale = d3.event.transform.rescaleY(y_scale);

    // update axes with these new boundaries
    d3.selectAll(".x-axis").call(d3.axisBottom(newXScale));
    d3.selectAll(".y-axis").call(d3.axisLeft(newYScale));

    // update circle position
    d3.select(".new_canvas").selectAll(".dot")
      .attr("cx", function (d) { return newXScale(d.GrLivArea); } )
      .attr("cy", function (d) { return newYScale(d.SalePrice); } )
  }
```

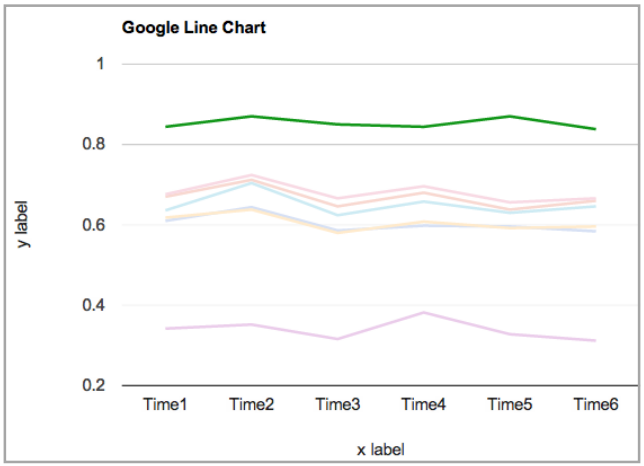
Callback function for zooming behavior

Transformation update

Apply transformation to all elements

Interact with Chart

- Cross chart interaction
  - Task :
    - highlight line when mouse over label in color legend.
  - Process
    - Set on.mouseover event on rect (color legend)
      - Update opacity of lines, hide all lines that is not corresponded.
    - Set on.mouseout event on rect
      - Reset all line opacities.
  - Tips
    - Set corresponding ID for each line and rect pair



SVG1 – line chart

SVG2 - color legend

Animation

- Animation = Transition of shape/type along time.
  - Transition
    - duration : time of animation (milliseconds)
    - delay : delay tine (in milliseconds) before animation
    - ease : methods to control the behavior of motion.

```
        .on('mouseover', function(d){
            d3.select(this)
              .transition().duration(300)
                  .style("fill", "steelblue")
                      .attr("r", 5);
        })
        .on('mouseout', function(d){
            d3.select(this)
              .transition() .delay(100).duration(1000)
                  .style("fill", function(d) { return color_scale(d.SalePrice);})
                  .attr("r", 1.5);
        })
```

Animation

- Animation through transition
  - Initialization
    - Place the elements at initial positions (and styles)
  - Animation process
    - Transition along time.
    - Cascaded transition.

Initialization

Amination 1
Move dots to target positions.

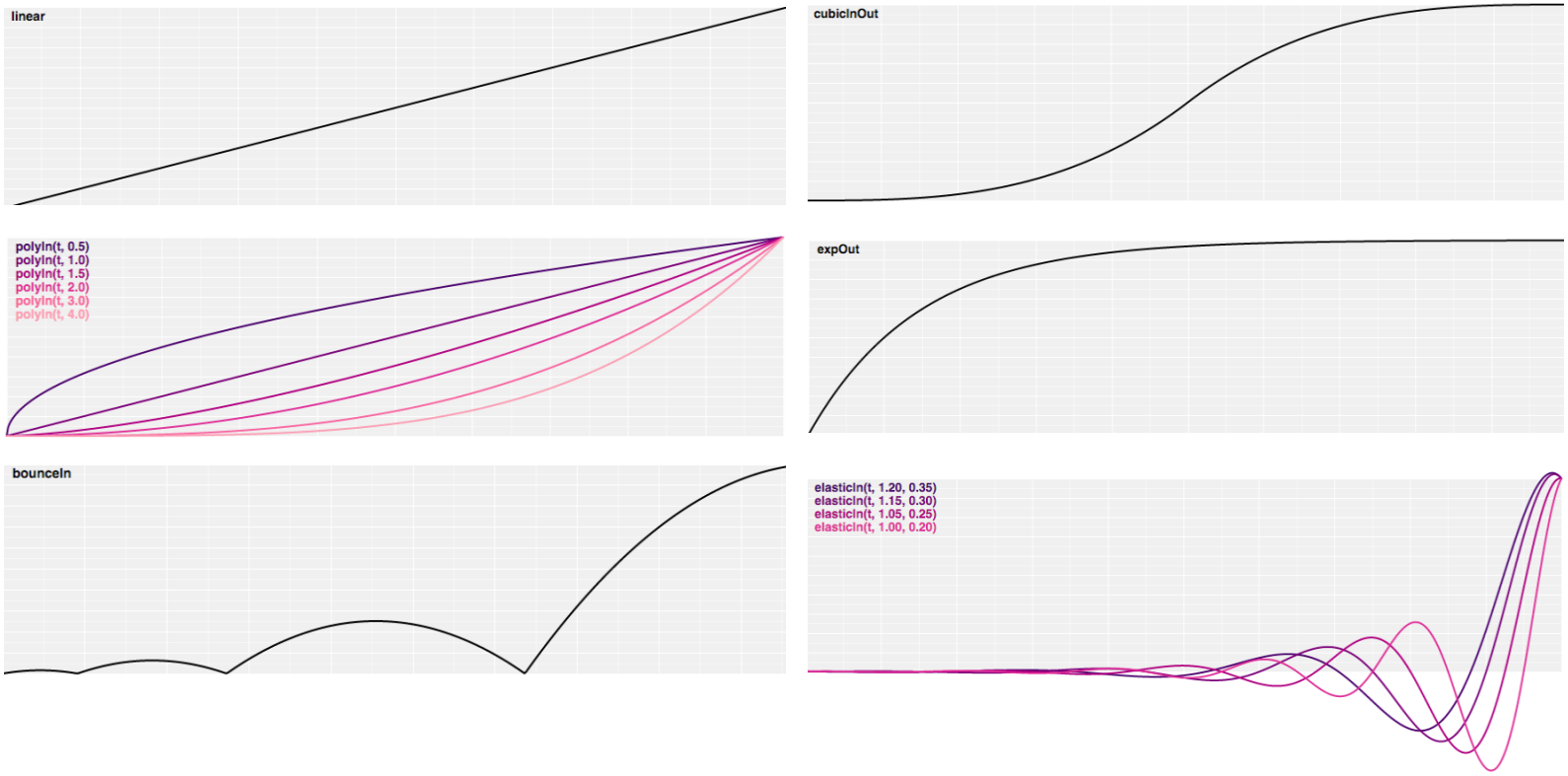
Amination 2
Resize dots

```
// Initialize dots
d3.select(".new_canvas").selectAll(".dot").data(data)
  .enter().append("circle")
    .attr("class", "dot")
    .attr("cx", function (d) { return x_scale(d.GrLivArea); } )
    .attr("cy", 340)
    .attr("opacity", 0)
    .attr("r", 1.5)
    .style("fill", function(d) { return color_scale(d.SalePrice);})

// apply animation to dots
d3.selectAll(".dot")
    .transition()
        .duration(200+Math.random()*1000)
        .delay(function(d){ return x_scale(d.GrLivArea); })
        .ease(d3.easeBackOut)
            .attr("cy", function (d) { return y_scale(d.SalePrice); } )
            .attr("r", 4.0)
            .attr("opacity", 0.4)
            .transition()
                .duration(200+Math.random()*1000)
                .ease(d3.easePolyOut)
                    .attr("r", 1.5)
                    .attr("opacity", 1)
```

Ease function to control the motion behavior.

Animation

- Example of ease functions



Reference : https://github.com/d3/d3-ease

Animation

- More on animation
  - Animation loop
    - Web animation events
      - on(“end”) event
        - https://bl.ocks.org/d3noob/bf44061b1d443f455b3f857f82721372

```
var timeCircle = svg.append("circle")
    .attr("fill", "steelblue")
    .attr("r", 20);
repeat();

function repeat() {
    timeCircle
        .attr('cx', 40)
        .attr('cy', 250)
        .transition()
            .duration(2000)
            .attr('cx', 920)
            .transition()
                .duration(2000)
                .attr('cx', 40)
                .on("end", repeat);
};
```

Infinite loop

  - Transition interpolation behavior
    - tween / attrTween / styleTween
      - Interpolate of data value / attr / style during transition.

```
.styleTween("opacity", function(d, i, a)
{
    return function(t) {
        return 0*(1-t)+1*t;
    }
})
```

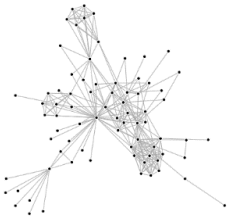
t = 1 -> 0

Layouts

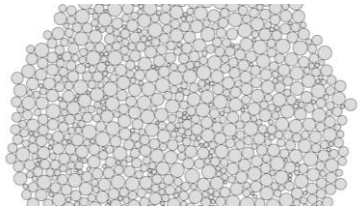
Layout in D3.js

- Methods to compute the position, size, or orientation of elements according to relations in data.
  - Through optimization process.

- Type of data relation
  - Chord layout
  - Force directed layout
  - Hierarchy layout
    - Tree
    - Treemap
    - Circle packing

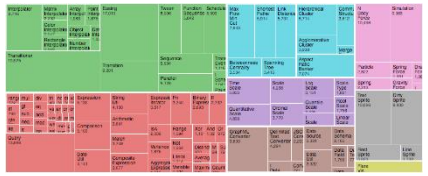


Forced-directed (graph)

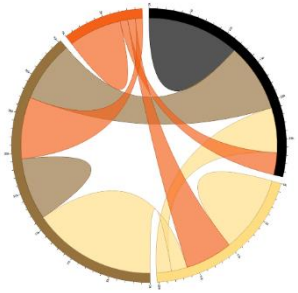


Forced-directed

Hierarchy (tree)



Hierarchy (treemap)



Chord diagram

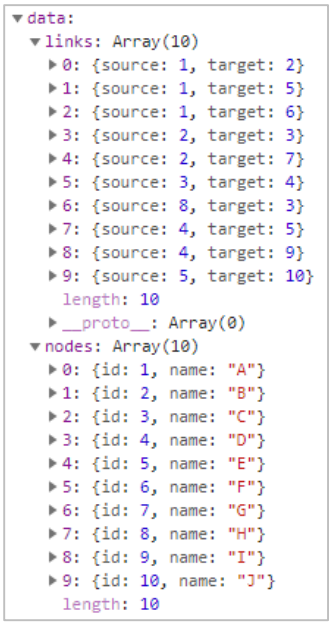
Hierarchy (circle packing)

Layout in D3.js

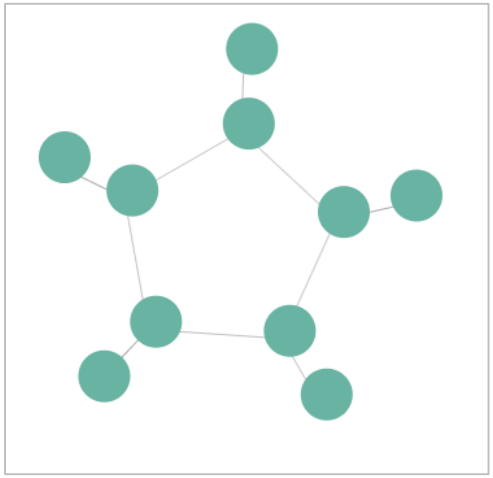
- How to choose layout?
  - According to the relation (structure) of data
    - Data is hierarchical (nested)
      - Hierarchy laouts
    - Data is (sparse) relationship
      - Force directed layouts
      - Chord diagram
      - Correlation matrix
    - Structured data
      - Standard statistical charts

Example of Graph Vis.

```
var width = 600;
var height = 400;

var svg = d3.select("body").select("svg")
  .attr("width", width)
  .attr("height", height);

d3.json("https://raw.githubusercontent.com/holtzy/D3-graph-gallery/master/DATA/data_network.json", function( data) {
  // Initialize the links
  var link = svg
    .selectAll("line")
    .data(data.links)
    .enter()
    .append("line")
      .style("stroke", "#aaa")

  // Initialize the nodes
  var node = svg
    .selectAll("circle")
    .data(data.nodes)
    .enter()
    .append("circle")
      .attr("r", 20)
      .style("fill", "#69b3a2")

  // Let's list the force we wanna apply on the network
  var simulation = d3.forceSimulation(data.nodes)              // Force algorithm is applied to data.nodes
      .force("link", d3.forceLink()                               // This force provides links between nodes
            .id(function(d) { return d.id; })                     // This provide  the id of a node
            .links(data.links)                                    // and this the list of links
      )
      .force("charge", d3.forceManyBody().strength(-400)) // Adds repulsion between nodes. (the repulsion strength = -400)
      .force("center", d3.forceCenter(width / 2, height / 2)) // This force attracts nodes to the center of the svg area
      .on("end", ticked);

  // This function is run at each iteration of the force algorithm, updating the nodes position.
  function ticked() {
    link
        .attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });

    node
         .attr("cx", function (d) { return d.x+6; })
         .attr("cy", function(d) { return d.y-6; });
  }
});
```



Data structure



Vis result

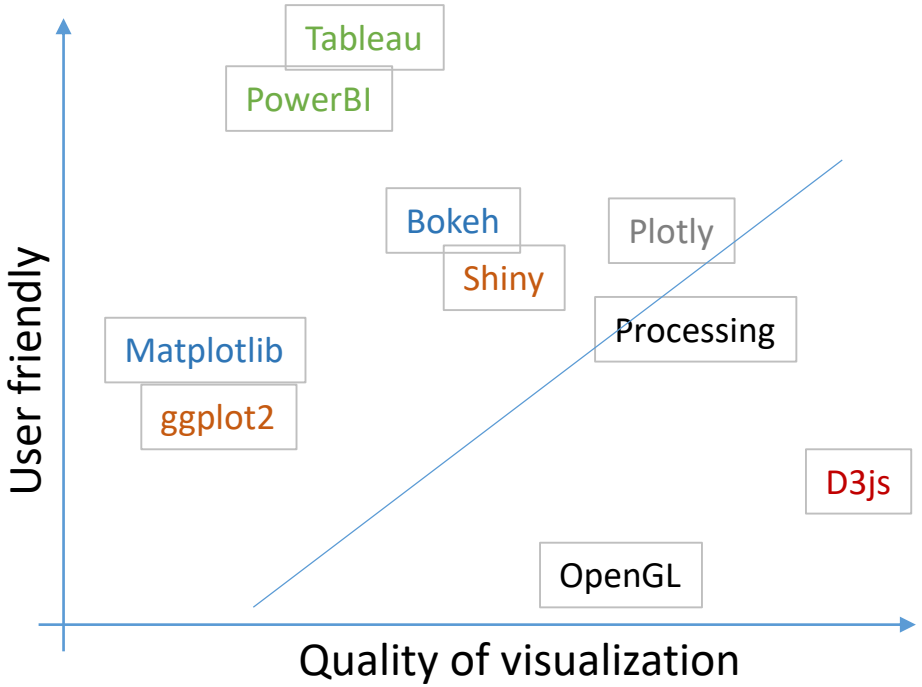
More on D3js

- How to learn D3js quickly
  - D3js Gallery : https://github.com/d3/d3/wiki/Gallery
    - Start from a chart
    - Modify to fit your data
    - Fine tune it

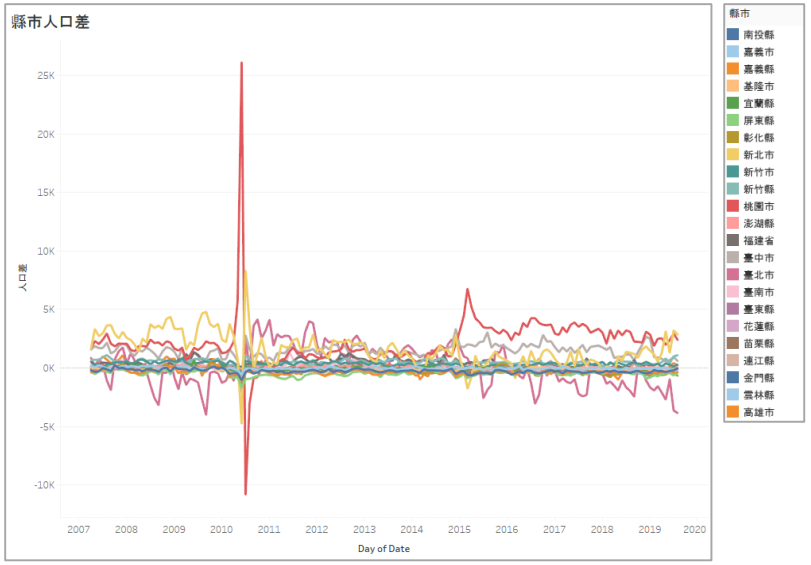
- Performance
  - Poor performance when lots of DOM elements (SVG issue).
    - Replace SVG with HTML5 Canvas or WebGL.

Data Visualization Resources

- Tool / Library of Data Visualization
  - Commercial Tools
    - Tableau
    - PowerBI
  - OpenSource
    - Javascript
      - D3js
      - Plotly
    - Python
      - Bokeh
      - Matplotlib
      - Plotly
    - R
      - ggplot2
      - Shily
      - Plotly
    - Processing
    - OpenGL



Assignment



- 利用D3js呈現各歷年縣市人口差資料
  - Data source : 縣市人口統計.csv
  - Requirement
    - 利用**折線圖**呈現
      - 每一**縣市**為一條線
      - 利用**顏色**區隔縣市
      - X-axis表示時間
      - Y-axis表示人口差
  - Bonus
    - 加上Color legend描述縣市與顏色對應 (*)
    - Interaction
      - Zooming on 折線圖 (*)
      - 加一個文字框(tooltip)，當滑鼠移到折線上時顯示詳細資料(縣市、時間、數值) (**)
      - 當滑鼠移到color legend，highlight出對應的縣市資料(折線) (**)
    - Animation (*~**)
    - 動態更新圖表大小來對齊視窗寬度 (Responsive Web Design) (***)