Lecture 11
More Visualisations by D3

DTS204TC Data Visualisation

Outline

- Path
  - Pie chart (p15 – p32)
  - Line chart (p33 – p51)

- Stack bar chart (p52 – p60)

Path

- Path is one of the most powerful elements in SVG
- Path element is defined by attr d

Path

- d
- fill
- stroke
- stroke-width
- transform

Path

- d
- fill
- stroke
- stroke-width
- transform

Path

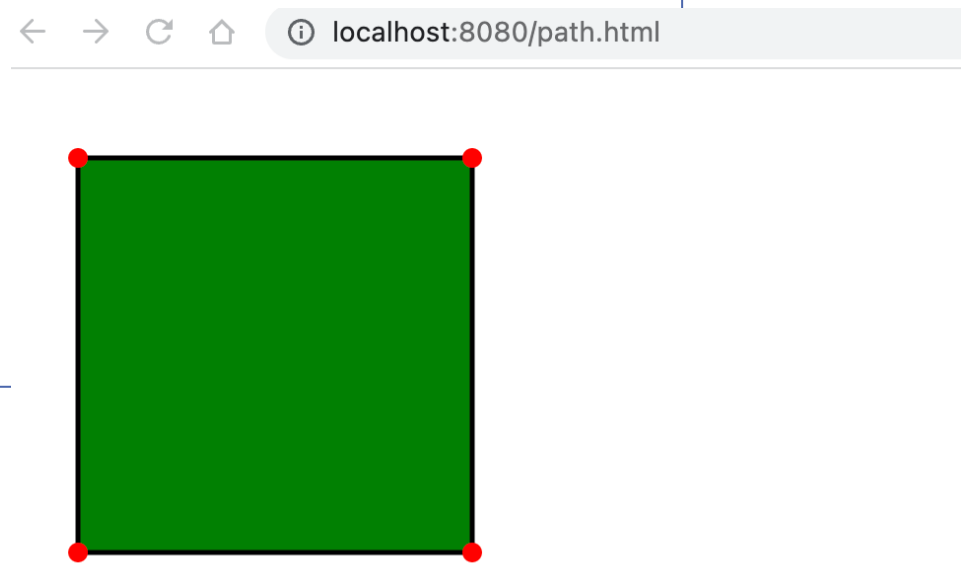
- d attr
  - M = moveto(M X,Y)
  - L = lineto(L X,Y)
  - H = horizontal lineto(H X)
  - V = vertical lineto(V Y)
  - C = curveto(C X1,Y1,X2,Y2,ENDX,ENDY)
  - S = smooth curveto(S X2,Y2,ENDX,ENDY)
  - Q = quadratic Belzier curve(Q X,Y,ENDX,ENDY)
  - T = smooth quadratic Belzier curveto(T ENDX,ENDY)
  - A = elliptical Arc(A RX,RY,XROTATION,FLAG1,FLAG2,X,Y)
  - Z = closepath()
  - These commands allow you to create complex shapes and curves within an SVG image.

Path-line

```
<!-- upper case -->
<svg width="100px" height="100px">
        <path d="M 10 10 H 90 V 90 H 10 Z" fill="green" stroke="black"/>
        <!-- Points -->
        <circle cx="10" cy="10" r="2" fill="red"/>
        <circle cx="90" cy="90" r="2" fill="red"/>
        <circle cx="90" cy="10" r="2" fill="red"/>
        <circle cx="10" cy="90" r="2" fill="red"/>
</svg>
```

Path-line

```
<!-- upper case -->
<svg width="100px" height="100px">
        <path d="M 10 10 H 90 V 90 H 10 Z" fill="green" stroke="black"/>
        <!-- Points -->
        <circle cx="10" cy="10" r="2" fill="red"/>
        <circle cx="90" cy="90" r="2" fill="red"/>
        <circle cx="90" cy="10" r="2" fill="red"/>
        <circle cx="10" cy="90" r="2" fill="red"/>
</svg>
```

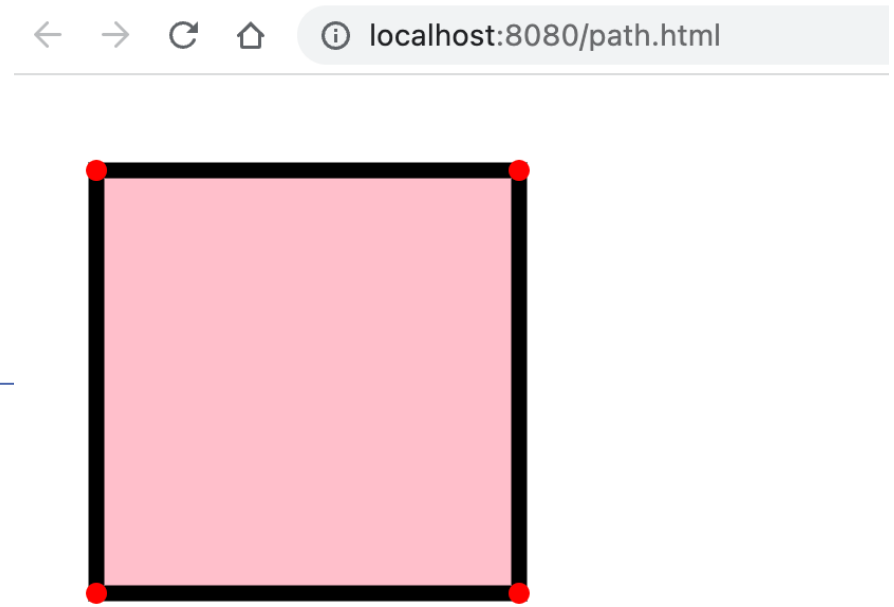


Path-line

```
<!-- lower case-->
<svg width="100px" height="100px">
        <path d="M10 10 h 80 v 80 h -80 Z" fill="pink" stroke="black" stroke-width="3"/>
        <!-- Points -->
        <circle cx="10" cy="10" r="2" fill="red"/>
        <circle cx="90" cy="90" r="2" fill="red"/>
        <circle cx="90" cy="10" r="2" fill="red"/>
        <circle cx="10" cy="90" r="2" fill="red"/>
</svg>
```

Path-line

```
<!-- lower case-->
<svg width="100px" height="100px">
        <path d="M10 10 h 80 v 80 h -80 Z" fill="pink" stroke="black" stroke-width="3"/>
        <!-- Points -->
        <circle cx="10" cy="10" r="2" fill="red"/>
        <circle cx="90" cy="90" r="2" fill="red"/>
        <circle cx="90" cy="10" r="2" fill="red"/>
        <circle cx="10" cy="90" r="2" fill="red"/>
</svg>
```



Path-line

```
<path d="M 10 10 h 80 v 80 h -80 Z" fill="pink" stroke="black" />
<path d="M 10 10 H 90 V 90 H 10 Z" fill="green" stroke="black"/>
```

- H, V and their lowercase counterparts (h, v) are commands used to specify different types of path operations.
  - H x (absolute horizontal line): Draws a horizontal line to the absolute x-coordinate x.
  - h dx (relative horizontal line): Draws a horizontal line by dx units relative to the current x-coordinate.
  - V y (absolute vertical line): Draws a vertical line to the absolute y-coordinate y.
  - v dy (relative vertical line): Draws a vertical line by dy units relative to the current y-coordinate.

Path

- Previous examples: too complex ?
  - <path d="M 10 10 H 90 V 90 H 10 Z" fill="green" stroke="black"/>
  - <path d="M10 10 h 80 v 80 h -80 Z" fill="pink" stroke="black" stroke-width="3"/>

Path

- Previous examples: too complex ?
  - <path d="M 10 10 H 90 V 90 H 10 Z" fill="green" stroke="black"/>
  - <path d="M10 10 h 80 v 80 h -80 Z" fill="pink" stroke="black" stroke-width="3"/>

- Built-in path generator

Path

- Built-in path generator
  - d3.line(...).x(...).y(...): line charts
  - d3.geoPath().projection(): map
  - d3.area()
  - d3.arc(...).innerRadius(...).outerRadius(...): pie chart
  - d3.lineRadial().angle(...).radius(...)
  - https://github.com/d3/d3-shape/tree/v1.3.7#arcs

Pie Chart

- What do we need to draw a pie ?

Pie Chart

- What do we need to draw a pie ?
  - Start angle
  - End angle
  - Inner Radius
  - Out Radius

Pie Chart

Data → Start angle & End angle → Inner Radius & Out Radius → Draw Pie Chart

Pie Chart

- d3.pie() – start angle & end angle
  - The d3.pie() function takes in a dataset and creates handy data for us to generate a pie chart in the SVG.
  - It calculates the start angle and end angle for each wedge of the pie chart. These start and end angles can then be used to create actual paths for the wedges in the SVG.

Pie Chart

```
<script>
        const data = [2, 4, 8, 10];
        //generate pie
        const pie = d3.pie();
        console.log(pie(data))

</script>
```

```
▼Array(4) i                                                                    test.h
  ▶0: {data: 2, index: 3, value: 2, startAngle: 5.759586531581287, endAngle: 6.283185307179586, …}
  ▶1: {data: 4, index: 2, value: 4, startAngle: 4.71238898038469, endAngle: 5.759586531581287, …}
  ▶2: {data: 8, index: 1, value: 8, startAngle: 2.617993877991494, endAngle: 4.71238898038469, …}
  ▶3: {data: 10, index: 0, value: 10, startAngle: 0, endAngle: 2.617993877991494, …}
    length: 4
  ▶[[Prototype]]: Array(0)
```

Pie Chart

- d3.arc() – inner radius & out radius
  - The d3.arc() generates an arc.
    - These are the paths that will create our pie's wedges. Arcs need an inner radius and outer radius. If the inner radius is 0, the result will be a piechart, otherwise the result will be a donut chart. We need to supply these generated arcs to our SVG path elements.

```
//generate arcs
const arc = d3.arc()
.innerRadius(0)
.outerRadius(radius);
```

Pie Chart

- 1.set canvas
  - We first define all our variables like width of the svg and, height of the svg. We calculate the radius as Math.min(width, height) / 2 to ensure that our generated pie will fit into the bounds of the SVG. For this, we choose whichever of the width and height is the minimum value.

```
//set canvas
const svg = d3.select("#mainsvg");
const width = +svg.attr("width");
const height = +svg.attr("height");
const radius = Math.min(width, height) / 2,
g = svg.append("g")
.attr("transform", `translate(${width / 2} ,${height / 2})`);
```

Pie Chart

- 2.set colour
  - we define our color scale as an ordinal scale. When we pass an index of a value in data array to the color scale, it will return the corresponding color value.

```
// set colour
var color = d3.scaleOrdinal(['#4daf4a','#377eb8','#ff7f00','#984ea3','#e41a1c', '#ebe134']);
```

Pie Chart

- 3.generate pie
  - we generate our pie values like startAngle and endAngle as seen in the previous example.

```
//generate pie
const pie = d3.pie().value(function(d){
        return d.value;
});
```

Pie Chart

- 4.generate arc
  - we define our arc with an inner radius of 0 and outer radius as the radius calculated earlier. This will be used to give paths to our pie wedges.

```
//generate arcs
const arc = d3.arc()
.innerRadius(0)
.outerRadius(radius);
```

Pie Chart

- 5.generate group
  - we create group elements for each of our data values. This group element will hold our individual paths or wedges.

```
//generate groups
const arcs = g.selectAll("arc")
.data(pie(data1))
.enter()
.append("g")
.attr("class", "arc")
```

Pie Chart

- 6.draw arc paths
  - we add a path element for each of our wedges. We provide the arc generated earlier and fill it with a color from our color scale.

```
//Draw arc paths
arcs.append("path")
.attr("fill", function(d, i) {
            return color(i);
})
.attr("d", arc);
```

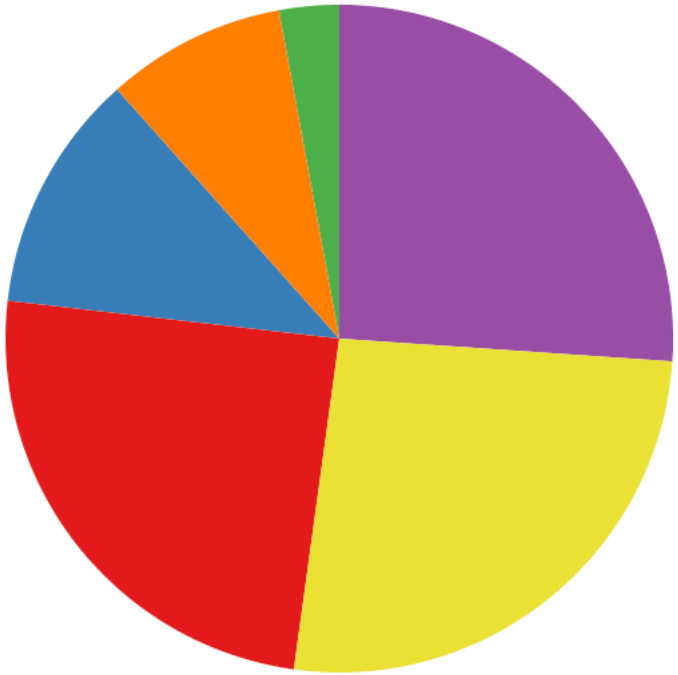
Pie Chart

```
<script>
        const data1 = [
        {name:"a", value:100},{name:"b", value: 400},{name:"c", value: 300},
        {name:"d", value:900},{name:"e", value: 850},{name:"f", value: 900},
        ];
        //set canvas
        const svg = d3.select("#mainsvg");
        const width = +svg.attr("width");
        const height = +svg.attr("height");
        const radius = Math.min(width, height) / 2,
        g = svg.append("g").attr("transform", `translate(${width / 2} ,${height / 2})`);
        // set colour
        var color = d3.scaleOrdinal(['#4daf4a','#377eb8','#ff7f00','#984ea3','#e41a1c',' #ebe134']);
        //generate pie
        const pie = d3.pie().value(function(d){
                return d.value;
        });
        //generate arcs
        const arc = d3.arc()
        .innerRadius(0)
        .outerRadius(radius);
```

Pie Chart

```
        //generate groups
        const arcs = g.selectAll("arc")
        .data(pie(data))
        .enter()
        .append("g")
        .attr("class", "arc")

        //Draw arc paths
        arcs.append("path")
        .attr("fill", function(d, i) {
        return color(i);
        })
        .attr("d", arc);
</script>
```

Pie Chart



Pie Chart

- Labels for pie chart

```
//generate labels
var label = d3.arc()
.outerRadius(radius)
.innerRadius(radius-80);
```

```
// add labels
arcs.append("text")
.attr("transform", function(d) {
return "translate(" + label.centroid(d) + ")";
})
.text(function(d) {
return d.data.name;
})
.attr("font-size","0.5em");
```

Pie Chart

```
// add labels
arcs.append("text")
.attr("transform", function(d) {
return "translate(" + label.centroid(d) + ")";
})
.text(function(d) {
return d.data.name;
})
.attr("font-size","0.5em");
```

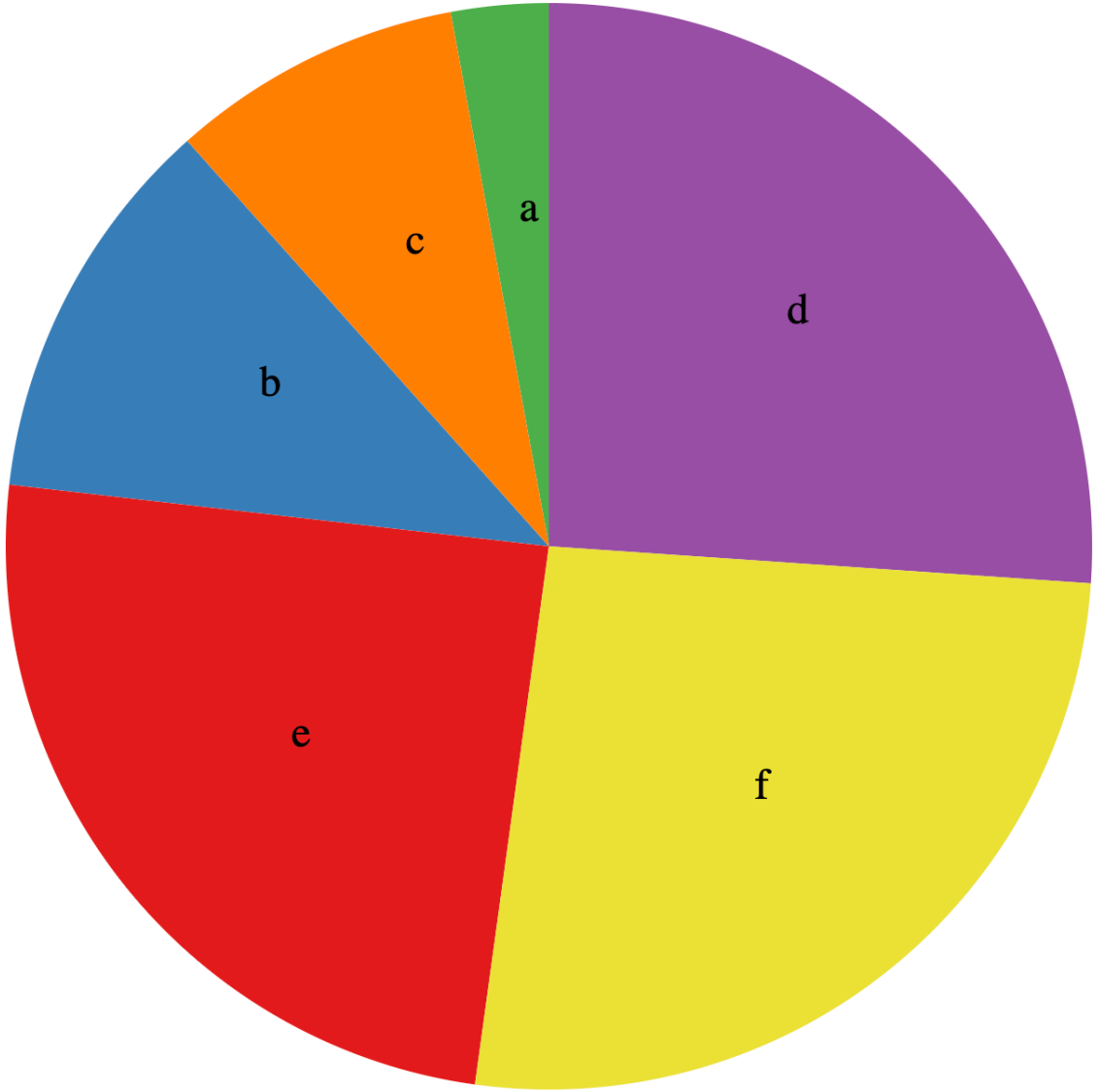
- arcs.append("text"): Appends a `<text>` element to each `<g>` element (`arcs`), which is used to display the labels.
- .attr("transform", function(d) { return "translate(" + label.centroid(d) + ")"; }): Sets the transformation attribute of the `<text>` element to move it to the centroid (center) of the corresponding arc. Here, `label.centroid(d)` returns the centroid coordinates of the arc.
- .text(function(d) { return d.data.name; }): Sets the text content of the `<text>` element to the name of the data element associated with the arc (`d.data.name`).

Pie Chart

- Labels for pie chart

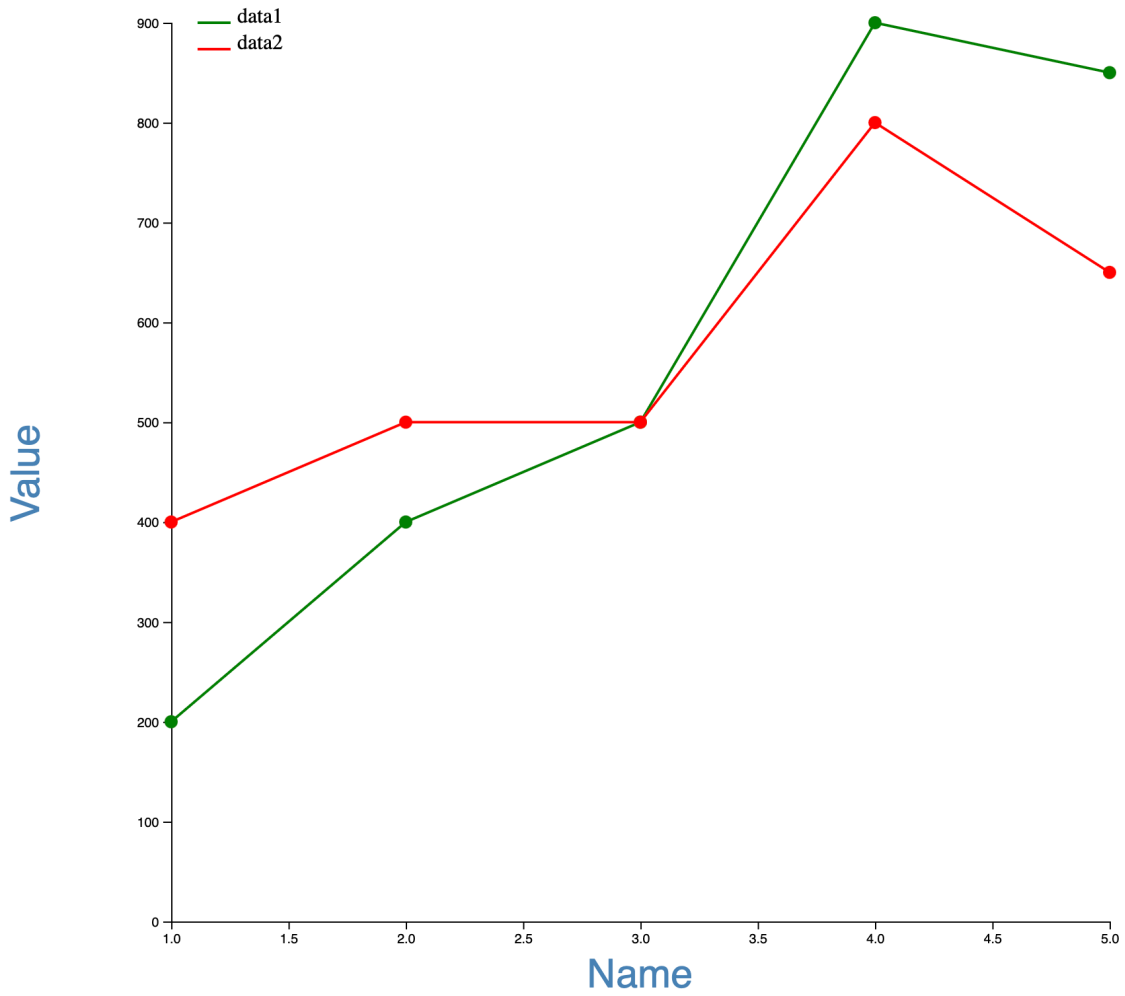
```
//generate labels
var label = d3.arc()
.outerRadius(radius)
.innerRadius(radius-80);
```

```
// add labels
arcs.append("text")
.attr("transform", function(d) {
return "translate(" + label.centroid
})
.text(function(d) {
return d.data.name;
})
.attr("font-size","0.5em");
```



Line chart

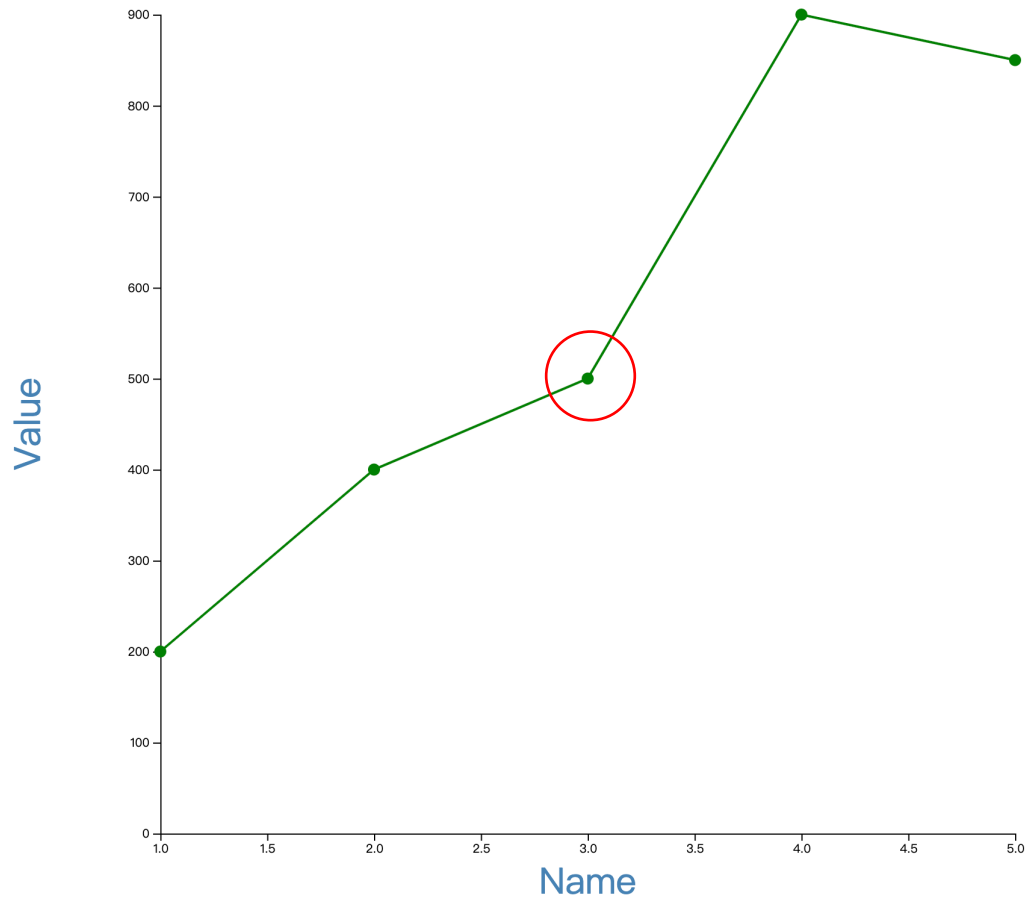
- Similar to bar chart
  - Data → SVG & Margin → Scale → Axis → Line Generate → Draw Line



Line chart

- Line Generate
- d3.line().x().y()

```
// line
const line = d3.line()
.x(d=>xScale(d.name))
.y(d=>yScale(d.value));
```



Line chart

- 1. dataset

```
const data1 = [
{name:1, value:200},{name:2, value: 400},{name:3, value: 500},
{name:4, value:900},{name:5, value: 850},
];
```

Line chart

- 2. margin and svg

```
//set svg and margin
const svg = d3.select("#mainsvg");
const width = +svg.attr("width");
const height = +svg.attr("height");
const margin = {top: 60, right: 30, bottom: 60, left: 150};
const innerWidth = width - margin.left - margin.right;
const innerHeight = height - margin.top - margin.bottom;
```

Line chart

- 3. scale

```
//scale
const xScale = d3.scaleLinear()
.domain(d3.extent(data1, d => d.name))
.range([0,innerWidth/2]);

const yScale = d3.scaleLinear()
.domain([0,d3.max(data1, d => d.value)])
.range([innerHeight,0]);
```

Line chart

- 4. axis

```
//axis
const g = svg.append("g")
.attr("id", "maingroup")
.attr("transform", `translate(${margin.left}, ${margin.top})`);
const x_axis = d3.axisBottom().scale(xScale);
const y_axis = d3.axisLeft().scale(yScale);
g.append ("g").call(x_axis)
.attr("transform",`translate(${0}, ${innerHeight})`)
.append("text")
.text("Name")
.attr("font-size", "3em")
.attr("x", innerWidth / 4)
.attr("y", 50)
.attr("text-anchor", "middle")
.attr("fill", "steelblue");
g.append ("g").call(y_axis)
.append("text")
.text("Value")
.attr("font-size", "3em")
.attr("x", -innerHeight /2 )
.attr("y", -100)
.attr("transform", "rotate(-90)")
.attr("text-anchor", "middle")
.attr("fill", "steelblue");
```

Line chart

- 5. line generator

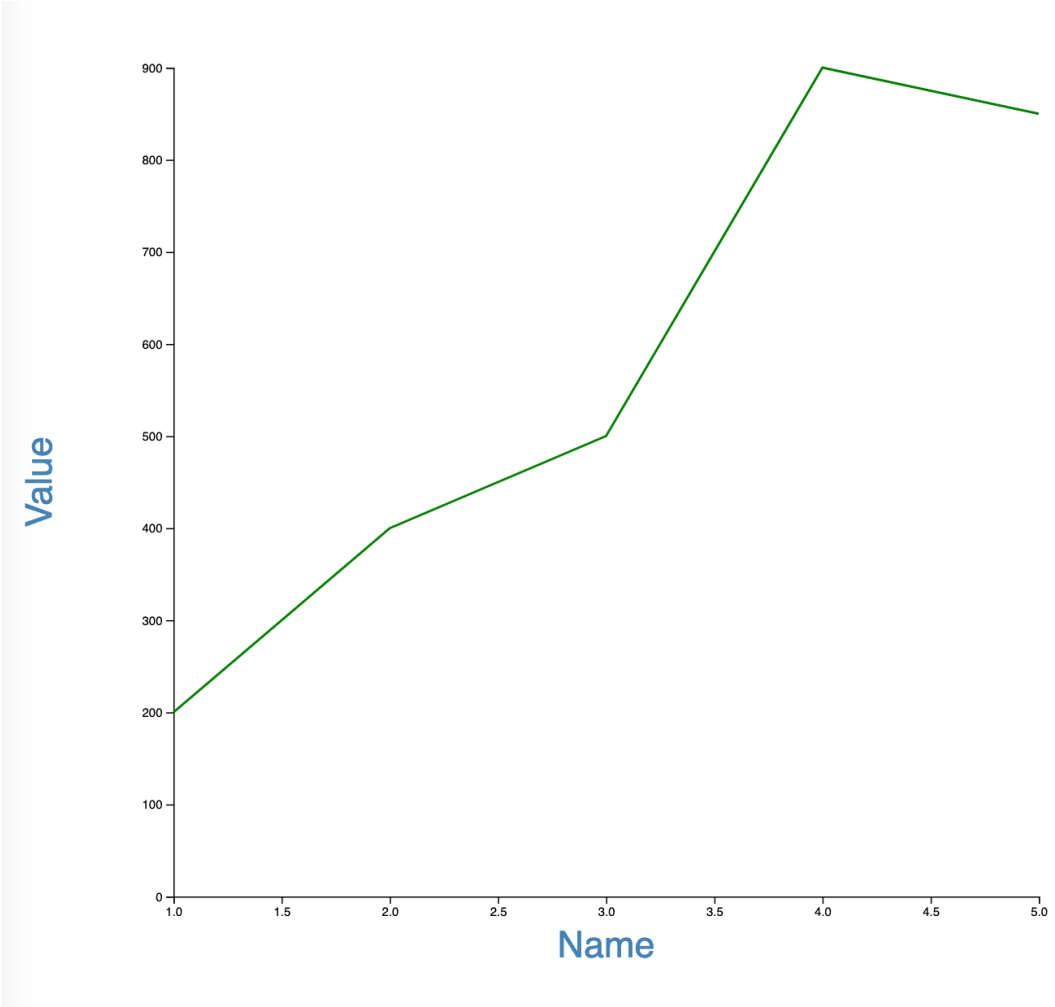
```
// line generator
const line = d3.line()
.x(d=>xScale(d.name))
.y(d=>yScale(d.value));
```

Line chart

- 6. draw line

```
// draw line
g.append('path')
.attr("class","line")
.attr("fill", "none")
.attr('stroke', 'green')
.attr('stroke-width', 2)
.attr("d", line(data1));
```

Line chart


Value
900
800
700
600
500
400
300
200
100
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Name

Line Charts

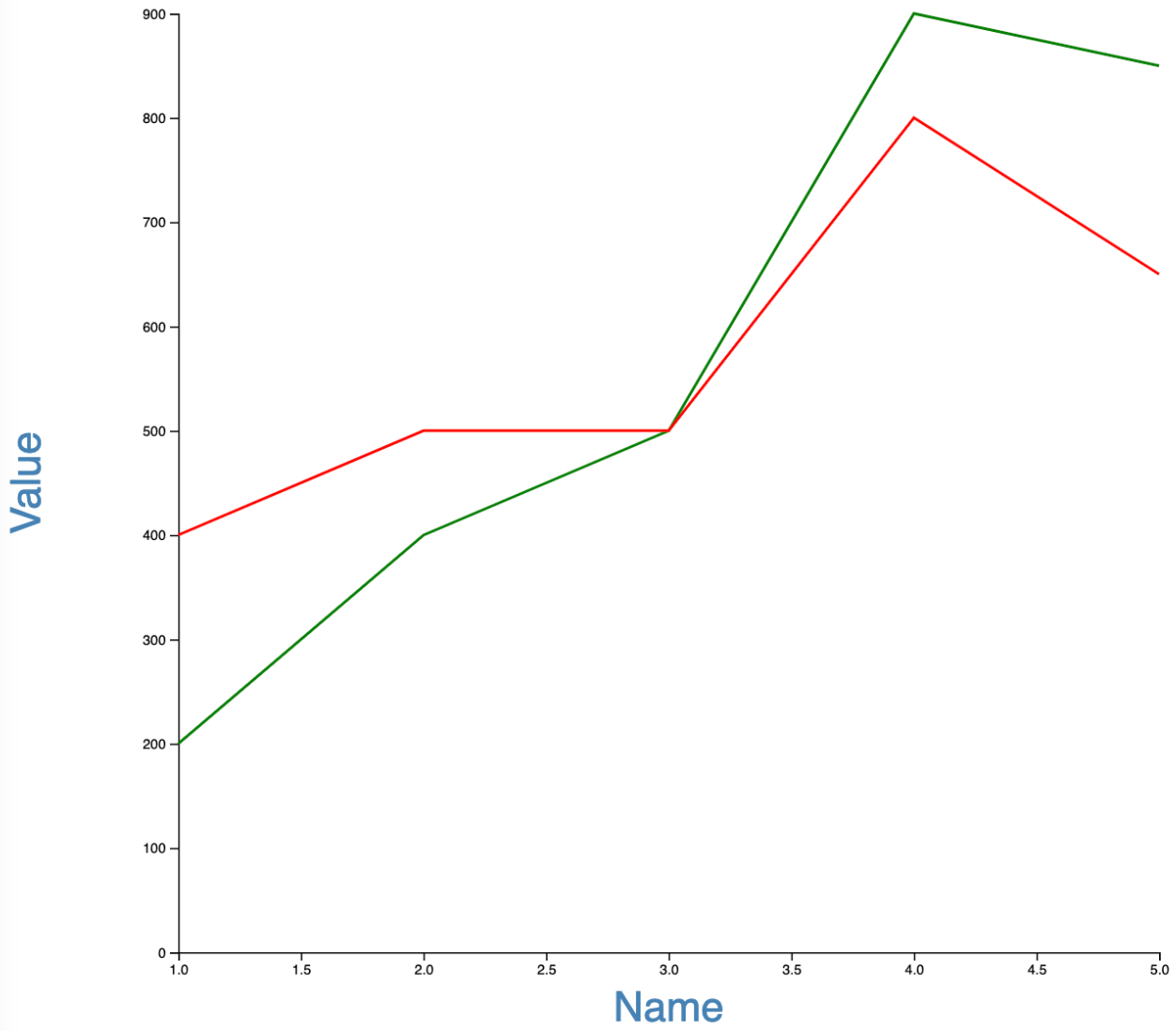
- more than one line

```
// draw line
g.append('path')
.attr("class","line")
.attr("fill", "none")
.attr('stroke', 'green')
.attr('stroke-width', 2)
.attr("d", line(data1));

g.append('path')
.attr("class","line")
.attr("fill", "none")
.attr('stroke', 'red')
.attr('stroke-width', 2)
.attr("d", line(data2));
```

Line Charts

- more than one line

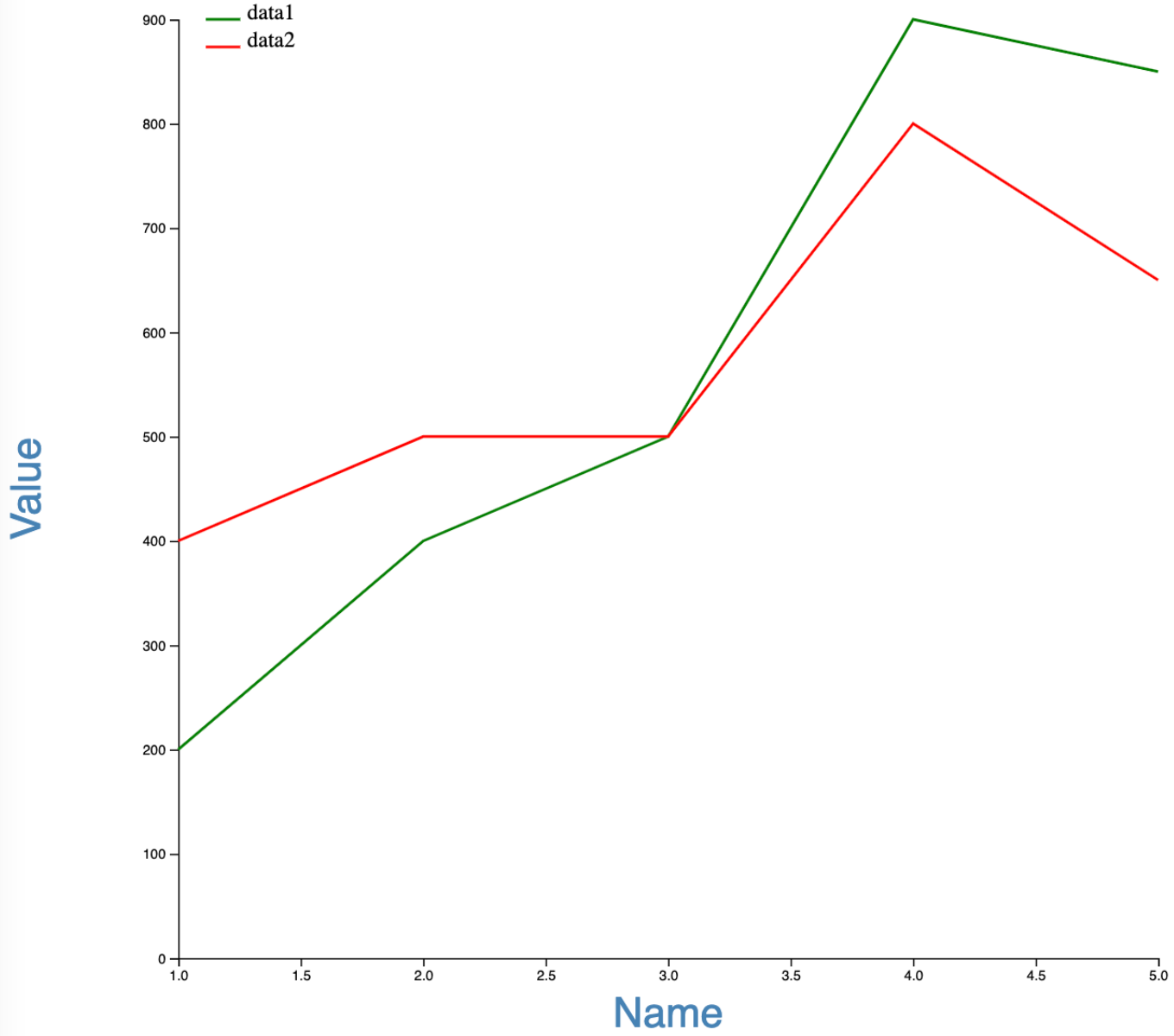
```
// draw line
g.append('path')
.attr("class","line")
.attr("fill", "none")
.attr('stroke', 'green')
.attr('stroke-width', 2)
.attr("d", line(data1));

g.append('path')
.attr("class","line")
.attr("fill", "none")
.attr('stroke', 'red')
.attr('stroke-width', 2)
.attr("d", line(data2));
```



Line Charts

- legend

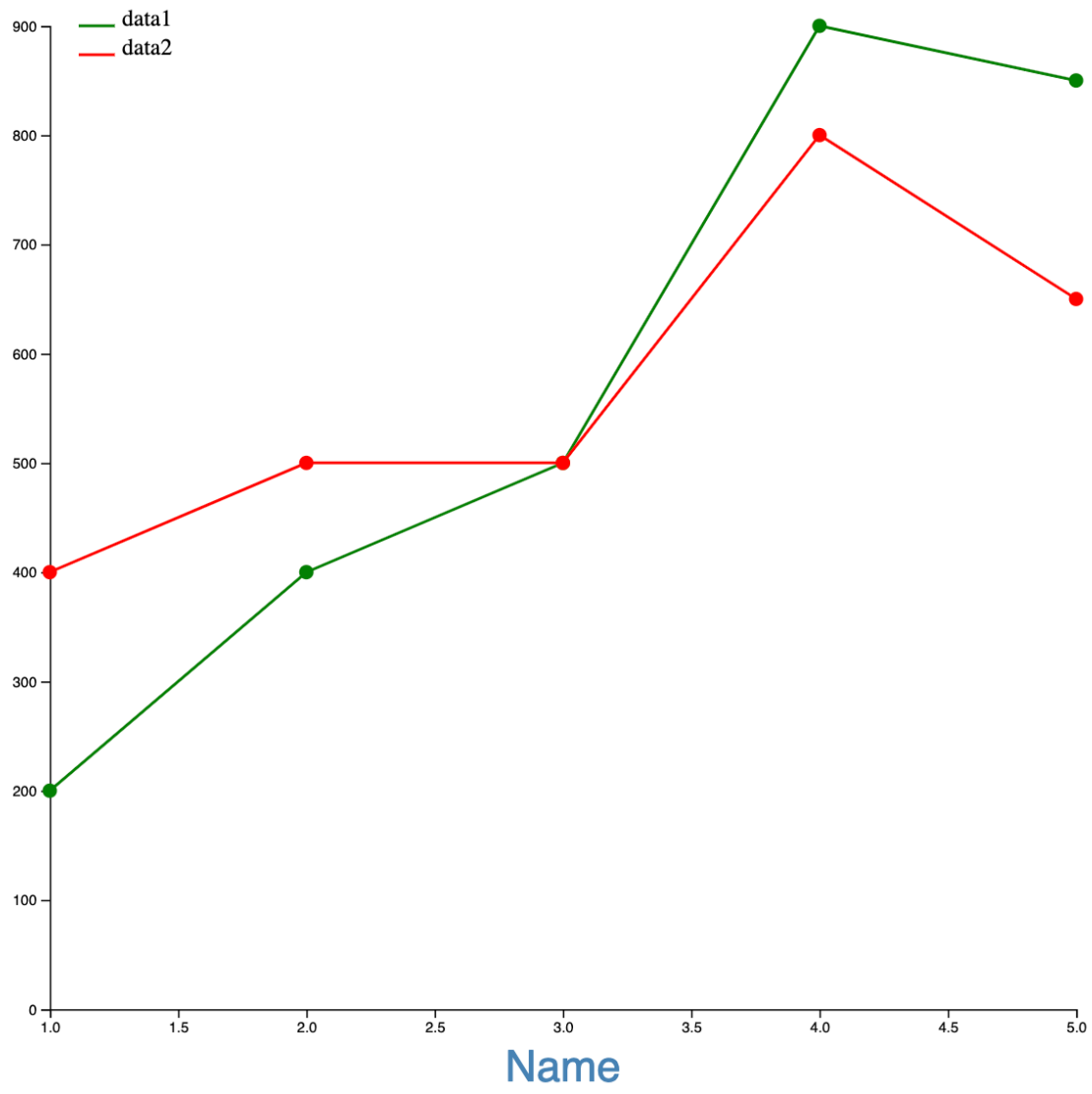
```
//legend
g.append("line")
.attr("x1",20).attr("y1",0)
.attr("x2",45).attr("y2",0)
.attr('stroke', 'green')
.attr('stroke-width', 2)
g.append("line")
.attr("x1",20).attr("y1",20)
.attr("x2",45).attr("y2",20)
.attr('stroke', 'red')
.attr('stroke-width', 2)

g.append("text")
.attr("x",50).attr("y",0)
.text("data1")
g.append("text")
.attr("x",50).attr("y",20)
.text("data2")
```



Line Charts

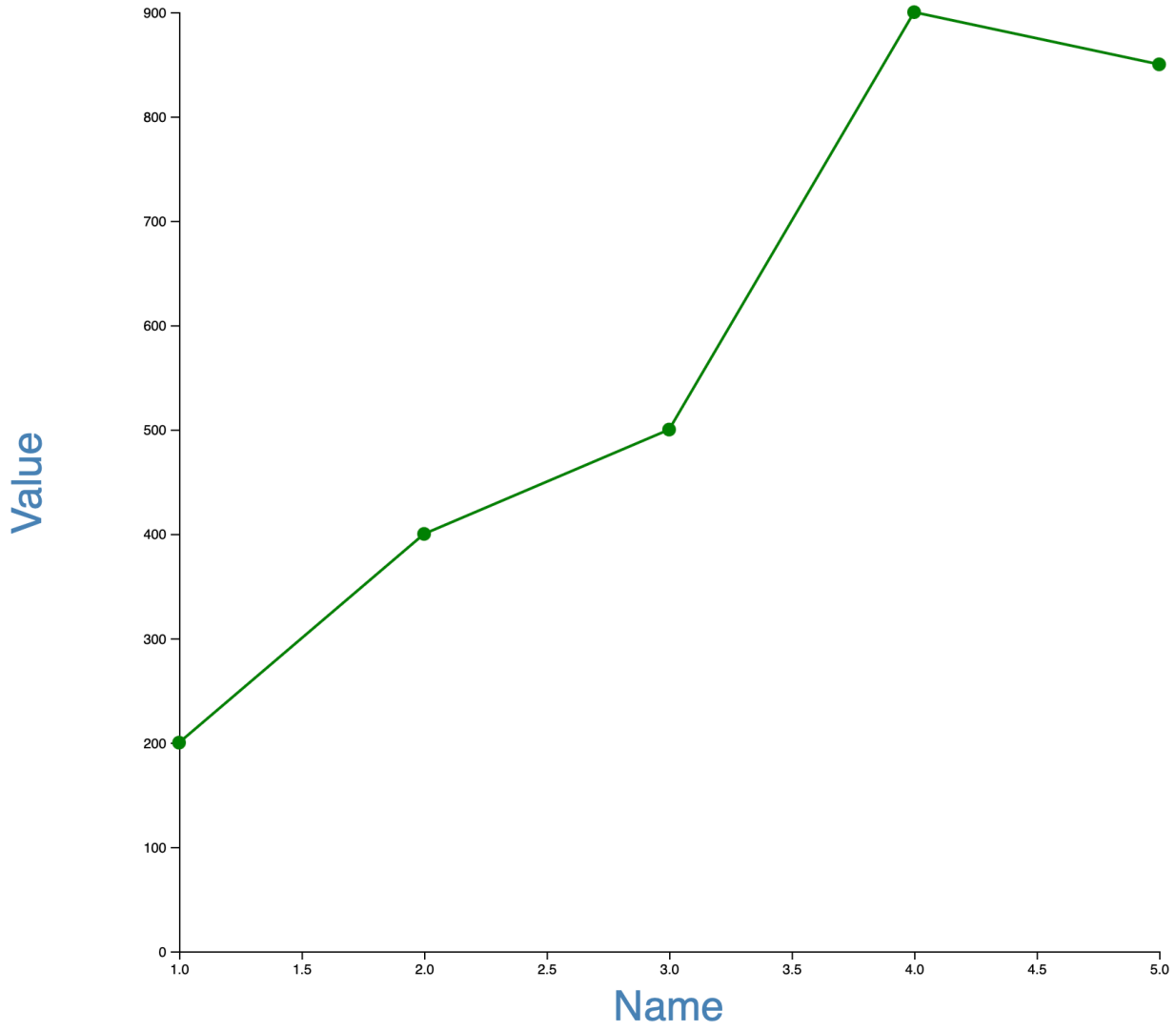
- Points

```
g.selectAll(".point1")
.data(data1)
.enter()
.append("circle")
.attr("class","point")
.attr("cx",d => xScale(d.name))
.attr("cy",d => yScale(d.value))
.attr("r",5)
.attr("fill", "green");

g.selectAll(".point2")
.data(data2)
.enter()
.append("circle")
.attr("class","point")
.attr("cx",d => xScale(d.name))
.attr("cy",d => yScale(d.value))
.attr("r",5)
.attr("fill", "red");
```



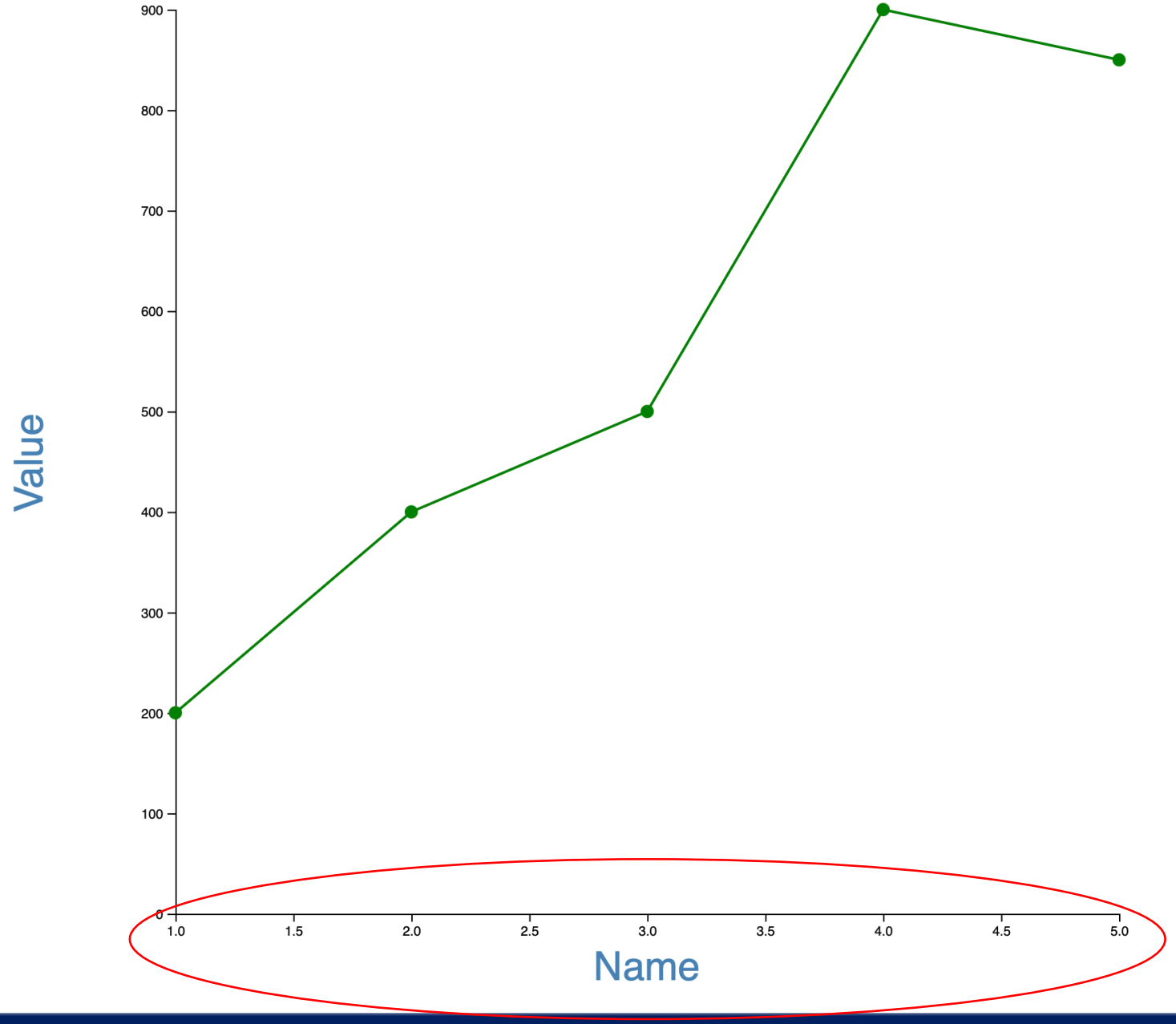
Line Charts

- Problems?



Line Charts

- Problems?



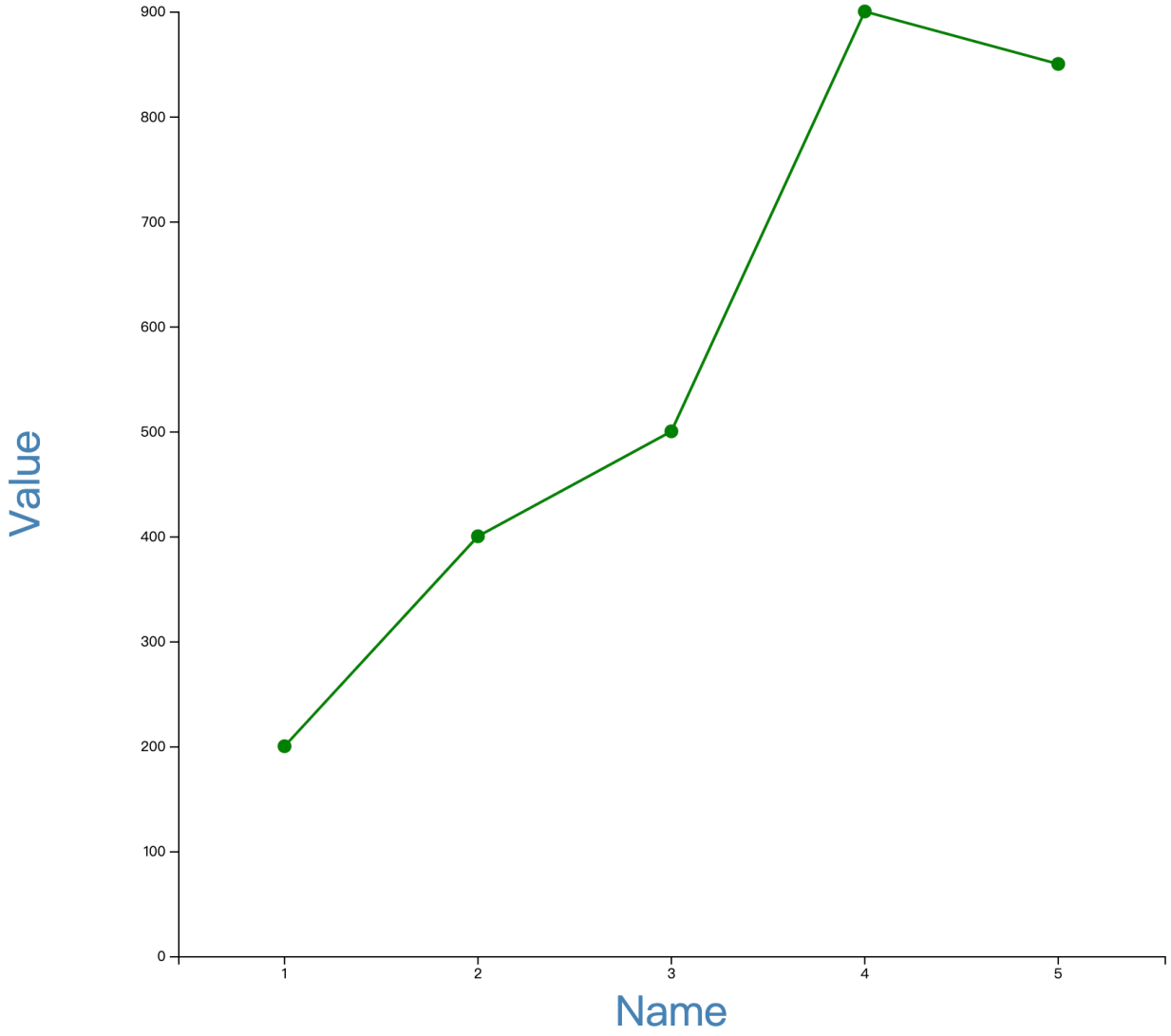
Line Charts

```
//scale
const xScale = d3.scaleBand()
.domain(data1.map(d => d.name))
.range([0,innerWidth/2])
.padding(0.1);
……
// line generator
const line = d3.line()
.x(d=>xScale(d.name)+xScale.bandwidth()/2)
.y(d=>yScale(d.value));
……
//points
```

Line Charts

```
//points
g.selectAll(".point1")
.data(data1)
.enter()
.append("circle")
.attr("class","point")
.attr("cx",d => xScale(d.name)+ xScale.bandwidth()/2)
.attr("cy",d => yScale(d.value))
.attr("r",5)
.attr("fill", "green");
```

Line Charts

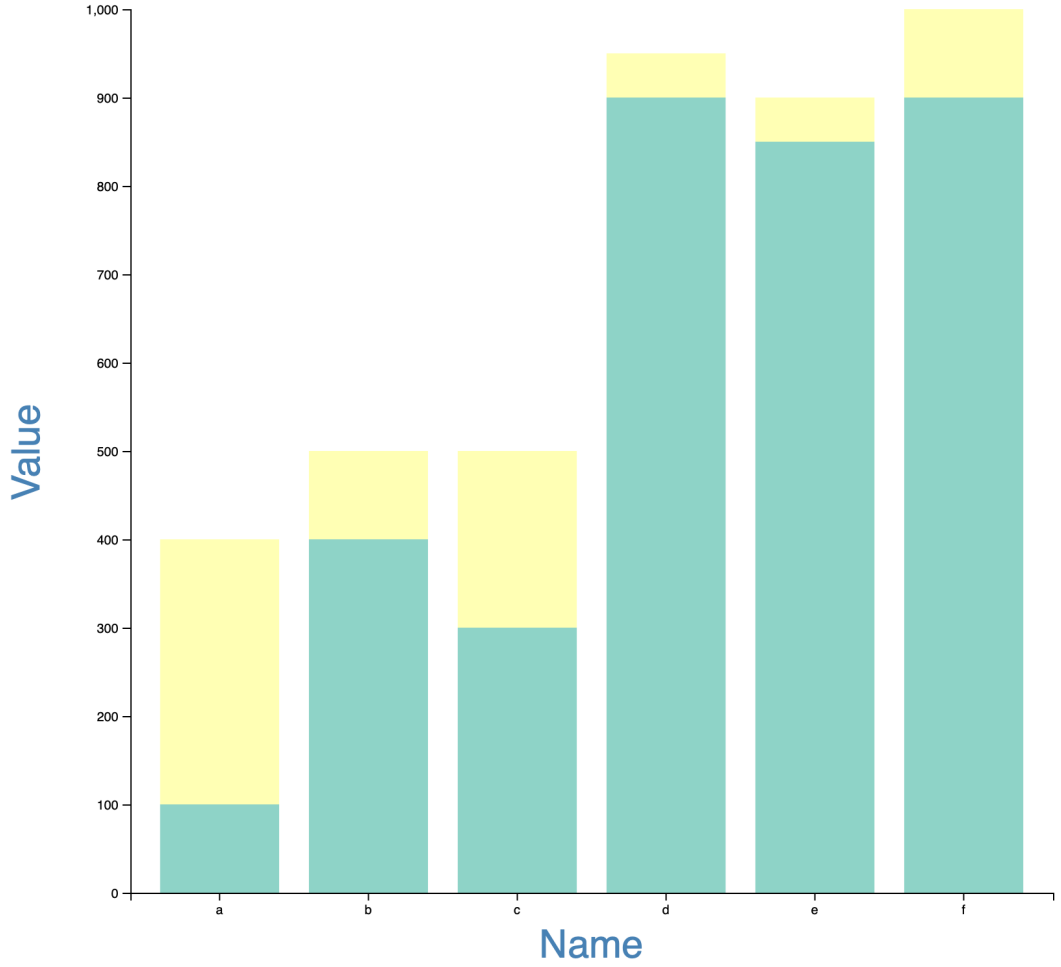


Line Charts

- Interaction?
- Animation?

Stack bar chart

- Similar to bar chart
  - Data → Stack Data →SVG & Margin → Scale → Axis →Draw Bars



Stack bar chart

- Stack data

```
const data = [
        {name:"a", value1:100,value2:300},
        {name:"b", value1: 400,value2:100},
        {name:"c", value1: 300,value2:200},
        {name:"d", value1:900,value2:50},
        {name:"e", value1: 850,value2:50},
        {name:"f", value1: 900,value2:100},
    ];
```

Stack bar chart

- Stack data
  - d3.stack().keys().order()
    - d3.stack(): create stacked visualizations.
    - .keys(): specify the keys (identifiers) of the data series that will be stacked.
    - .order() determine the order in which the data series are stacked.

```
//stack data
const keys = ["value1","value2"];

var stack = d3.stack()
.keys(keys)
.order(d3.stackOrderNone)(data);
console.log(stack);
```

Stack bar chart

- Stack data
  - d3.stack().keys().order()

```
▼ (2) [Array(6), Array(6)] i
  ▶ 0: (6) [Array(2), Array(2), Array(2), Array(2), A…
  ▶ 1: (6) [Array(2), Array(2), Array(2), Array(2), A…
    length: 2
  ▶ [[Prototype]]: Array(0)
```

```
▼ 0: Array(6)
  ▶ 0: (2) [0, 100, data: {…}]
  ▶ 1: (2) [0, 400, data: {…}]
  ▶ 2: (2) [0, 300, data: {…}]
  ▶ 3: (2) [0, 900, data: {…}]
  ▶ 4: (2) [0, 850, data: {…}]
  ▶ 5: (2) [0, 900, data: {…}]
    index: 0
    key: "value1"
    length: 6
  ▶ [[Prototype]]: Array(0)
```

```
▼ 1: Array(6)
  ▶ 0: (2) [100, 400, data: {…}]
  ▶ 1: (2) [400, 500, data: {…}]
  ▶ 2: (2) [300, 500, data: {…}]
  ▶ 3: (2) [900, 950, data: {…}]
  ▶ 4: (2) [850, 900, data: {…}]
  ▶ 5: (2) [900, 1000, data: {…}]
    index: 1
    key: "value2"
    length: 6
```

Stack bar chart

- Scale
  - Problem?

```
//scale
const xScale = d3.scaleBand()
.domain(data.map(d => d.name))
.range([0,innerWidth/2])
.padding(0.2);

const yScale = d3.scaleLinear()
.domain([0,d3.max(data, d => d.value1 + d.value2)])
.range([innerHeight,0]);

const colourScale = d3.scaleOrdinal()
.domain(keys)
.range(d3.schemeSet3);
```

Stack bar chart

- Scale

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

Stack bar chart

- draw bars

```
//draw bars
g.selectAll(".stack")
.data(stack)
.join('g')
.attr("class","stack")
.attr("fill",d => colourScale(d.key))
.selectAll('.stackrect')
.data(d => d)
.join('rect')
.attr('class', 'stackrect')
.attr('y', d => yScale(d[1]))
.attr('x', d => xScale(d.data.name))
.attr('height', d => yScale(d[0]) - yScale(d[1]))
.attr('width', xScale.bandwidth());
```

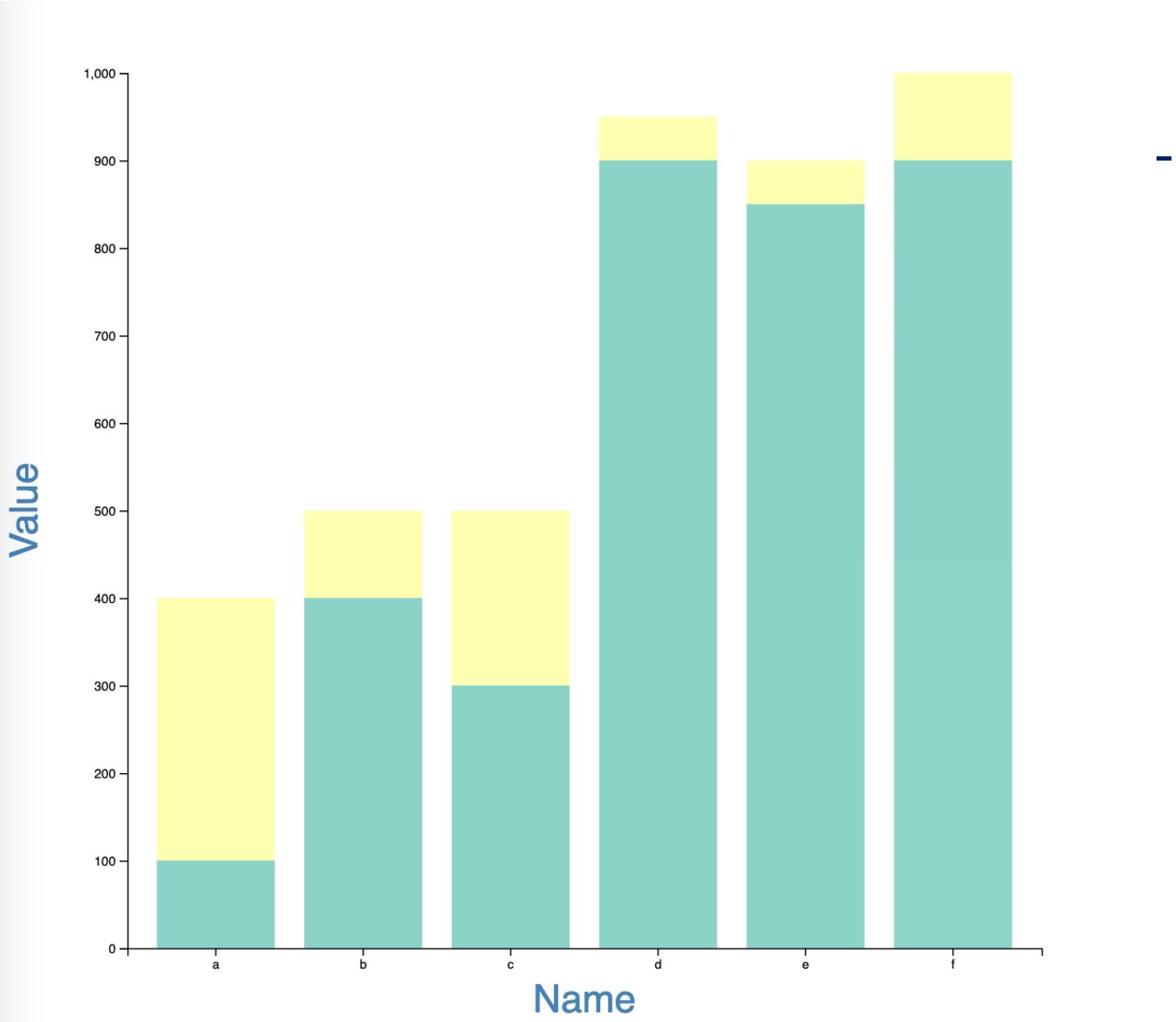
Stack bar chart

- selectAll(".stack"): Selects all elements with the class ".stack".
- .join('g'): Creates a `g` element for each data element.
- .selectAll('.stackrect'): Selects all elements with the class ".stackrect".
- .attr('y', d => yScale(d[1])): Sets the `y` position of the rectangles based on the second value (`d[1]`) of the stacked data.
- .attr('height', d => yScale(d[0]) - yScale(d[1])): Sets the height of the rectangles based on the difference between the first value (`d[0]`) and the second value (`d[1]`) of the stacked data

```
//draw bars
g.selectAll(".stack")
.data(stack)
.join('g')
.attr("class","stack")
.attr("fill",d => colourScale(d.key))
.selectAll('.stackrect')
.data(d => d)
.join('rect')
.attr('class', 'stackrect')
.attr('y', d => yScale(d[1]))
.attr('x', d => xScale(d.data.name))
.attr('height', d => yScale(d[0]) - yScale(d[1]))
.attr('width', xScale.bandwidth());
```

Stack bar chart

- draw bars



Summary

- Path
  - Pie chart
  - Line chart
- Stacked bar chart