D3.js
数据的堆叠

张松海 张少魁

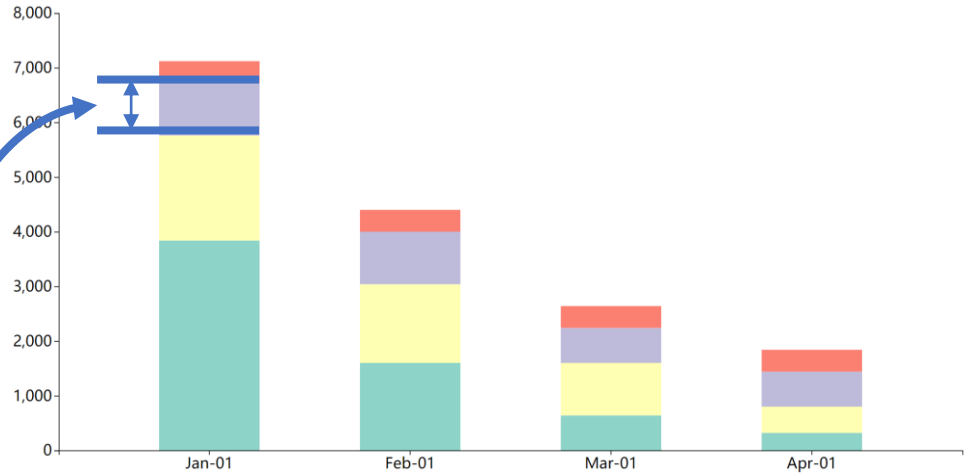
清华大学 可视媒体研究中心

概览

- 使用D3.js的‘stack’接口将数据堆叠：
  - 数据堆叠后的格式。
  - 堆叠的取值与顺序。
- 基于‘stack’实现堆叠柱状图：
  - 嵌套数组的最大值计算。
  - Data-Join嵌套、继承与传递。
- 基于‘stack’实现主题河流：
  - 区域的‘d’属性接口：d3.area()。

使用D3.js的‘stack’接口将数据堆叠

- let stack = d3.stack()：
  - 定义一个堆叠**函数**，用于将输入(CSV)数据堆叠。
  - e.g., let stackedData = stack(data)
  - 返回结果以**属性**为单位，将数据分堆。
  - 如下方数据将包含四堆：apples、bananas、cherries、dates。
  - 每堆包含若干对应属性**堆叠后**的位置，即**起点**和**终点**。




```
const naiveData = [
    { month: new Date(2015, 0, 1), apples: 3840, bananas: 1920, cherries: 960, dates: 400 },
    { month: new Date(2015, 1, 1), apples: 1600, bananas: 1440, cherries: 960, dates: 400 },
    { month: new Date(2015, 2, 1), apples: 640, bananas: 960, cherries: 640, dates: 400 },
    { month: new Date(2015, 3, 1), apples: 320, bananas: 480, cherries: 640, dates: 400 }
];
```

数据堆叠后的格式

- 数据原本的形式：四条数据、每条有四个要堆叠的属性。

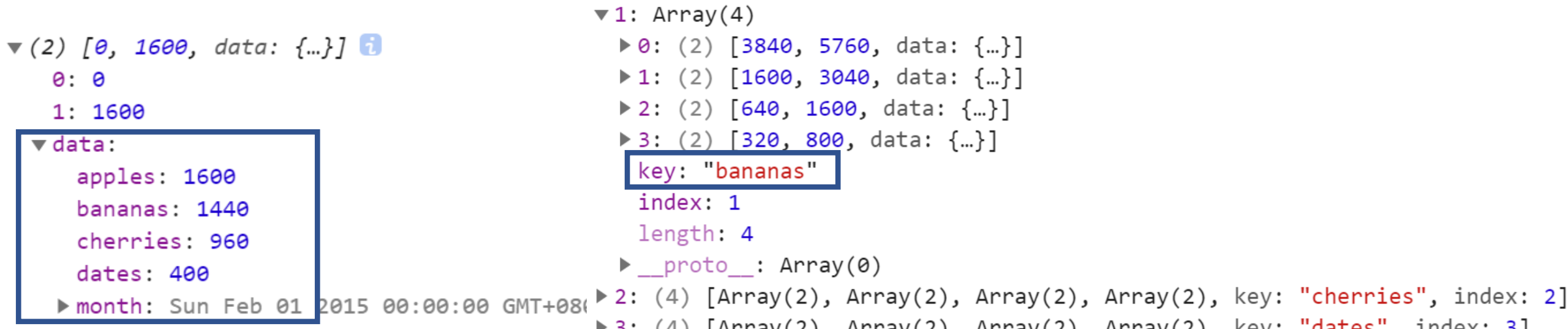
```
const naiveData = [
    { month: new Date(2015, 0, 1), apples: 3840, bananas: 1920, cherries: 960, dates: 400 },
    { month: new Date(2015, 1, 1), apples: 1600, bananas: 1440, cherries: 960, dates: 400 },
    { month: new Date(2015, 2, 1), apples: 640, bananas: 960, cherries: 640, dates: 400 },
    { month: new Date(2015, 3, 1), apples: 320, bananas: 480, cherries: 640, dates: 400 }
];
```

- 返回的形式：

```
[
    [[   0, 3840], [   0, 1600], [   0,  640], [   0,  320]], // apples
    [[3840, 5760], [1600, 3040], [ 640, 1600], [ 320,  800]], // bananas
    [[5760, 6720], [3040, 4000], [1600, 2240], [ 800, 1440]], // cherries
    [[6720, 7120], [4000, 4400], [2240, 2640], [1440, 1840]], // dates
]
```

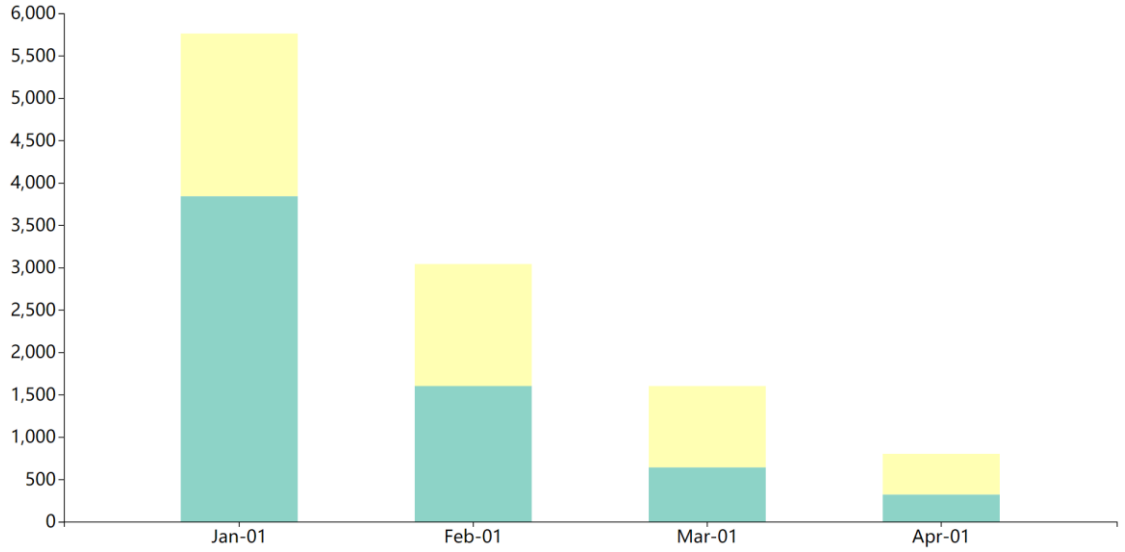
数据堆叠后的格式

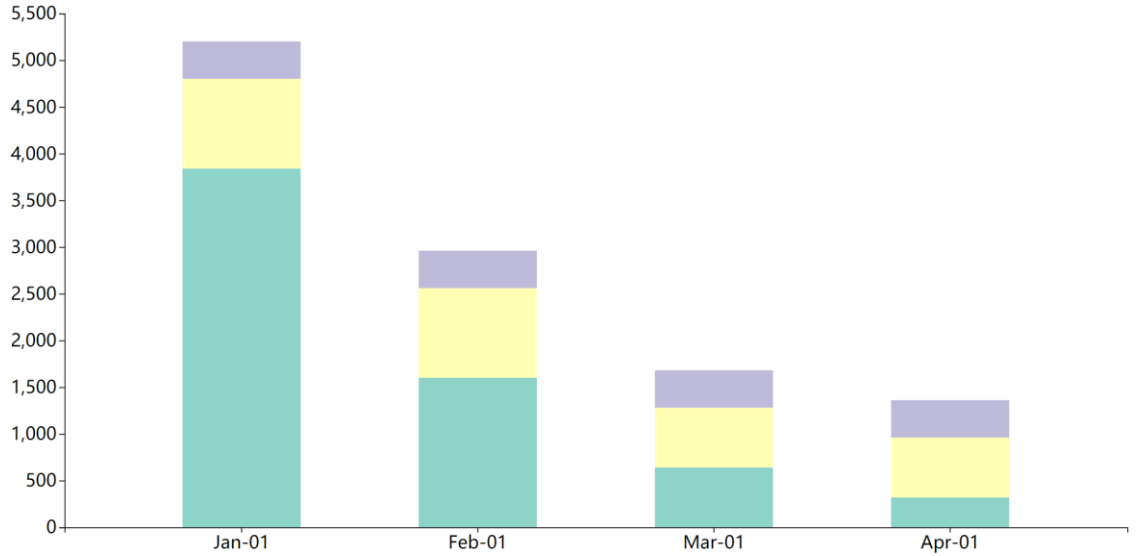
- 堆叠后的数据为 **数组**的**数组**的**数组**：
- **数组**：长度固定为二，包含堆叠后一特定部分的**起点**与**终点**，**包含对原本数据的映射**。
- **数组**：长度为原本CSV数据的条目数，包含‘**key**’即数组属于哪个属性。
- **数组**：堆叠后的数据。



堆叠的取值

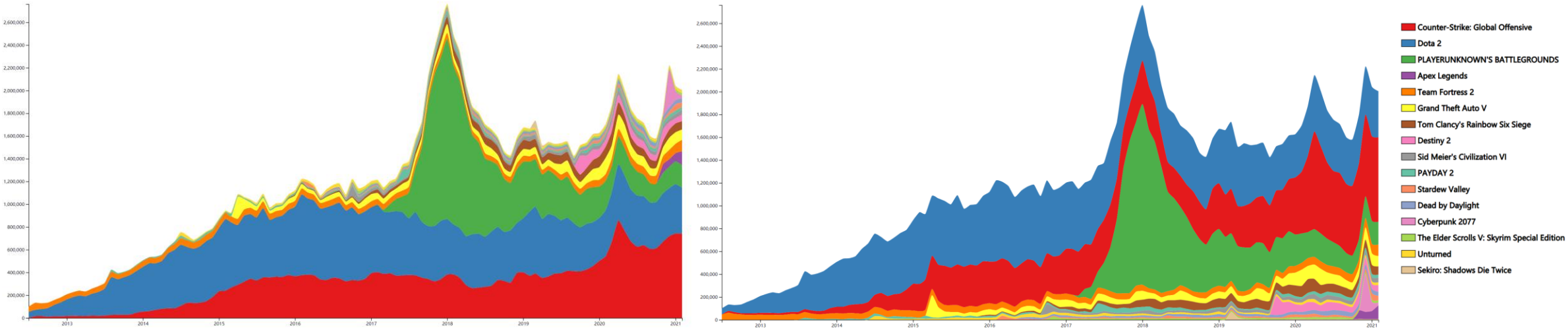
- stack.keys(array)：
  - 设置堆叠参考的值（**Key**）有哪些。
  - e.g., stack.keys([“apples”, “bananas”]); // 只堆叠苹果和香蕉。
  - e.g., stack.keys([“apples”, “cherries”, “dates”]); // 堆叠三者。





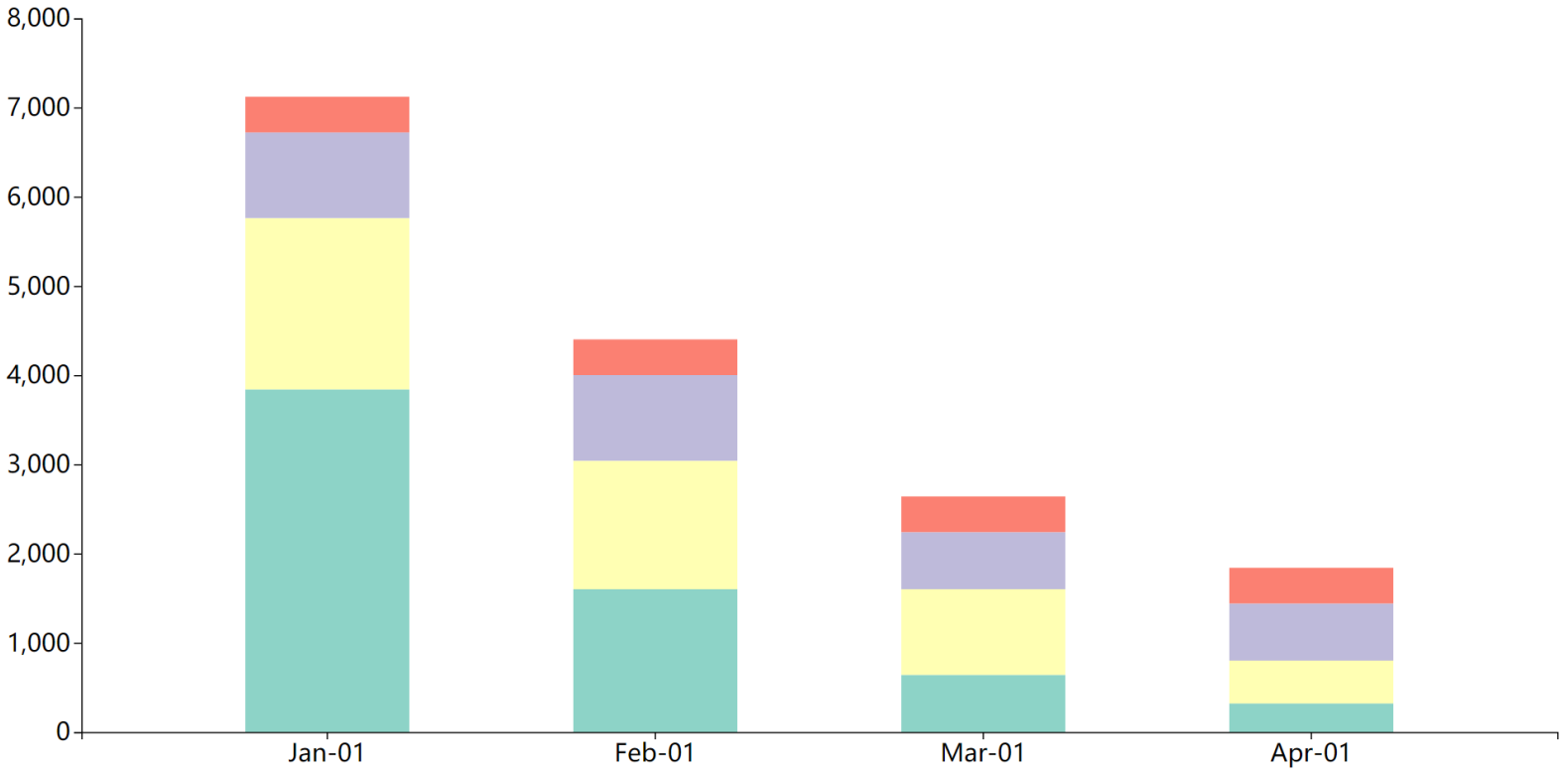
堆叠的顺序

- stack.orders(D3.js的顺序函数):
  - 设置的每个‘堆’属性排布的顺序。
  - e.g., stack.orders(d3.stackOrderNone); // 随**Key**的顺序。
  - e.g., stack.orders(d3.stackOrderAscending); // 按照堆叠值升序。
  - 根据需求查阅文档即可： https://github.com/d3/d3-shape/blob/v2.1.0/README.md#stack-orders



基于‘stack’实现堆叠柱状图

- 数据来源：
  - https://github.com/d3/d3-shape/blob/v2.1.0/README.md#stack



取嵌套数组的最大值

- 因需计算比例尺，故需对堆叠后的数据计算最大值与最小值：
  - 而堆叠后的数据又是**数组**的**数组**的**数组**。
  - 需要取嵌套数组的最大、最小值。
- 取嵌套数组的最大值：
  - d3.max([**[1,2,3]**, **[666,233,999]**, **[110,120,119]**], **a** => d3.max(**a**));
- 编程实例：

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

Data-Join嵌套、传递与继承

- 如果 selection.data(data):
  - data数组会根据图元被分给各个绑定的图元
- 如果 **selection.data(data).attr(…).selectAll(…).data(d => d).join(…)**:
  - 把父节点的数据进一步按数组拆分，分给各个后续的子节点。
- 图元绑定的数据会被‘.append(…)’添加的子节点继承:
  - let gs = svg.selectAll(‘g’).data(data).join(‘g’); // 基于Data-Join添加若干<g>。
  - gs.append(‘rect’); // 为每个<g>添加矩形，每个矩形绑定的数据随<g>。
- 编程实例:

```
g.selectAll('.datagroup').data(naiveStack).join('g')
.attr('class', 'datagroup')
.attr('fill', d => color(d.key))
.selectAll('.datarect').data(d => d).join('rect')
.attr('class', 'datarect')
```

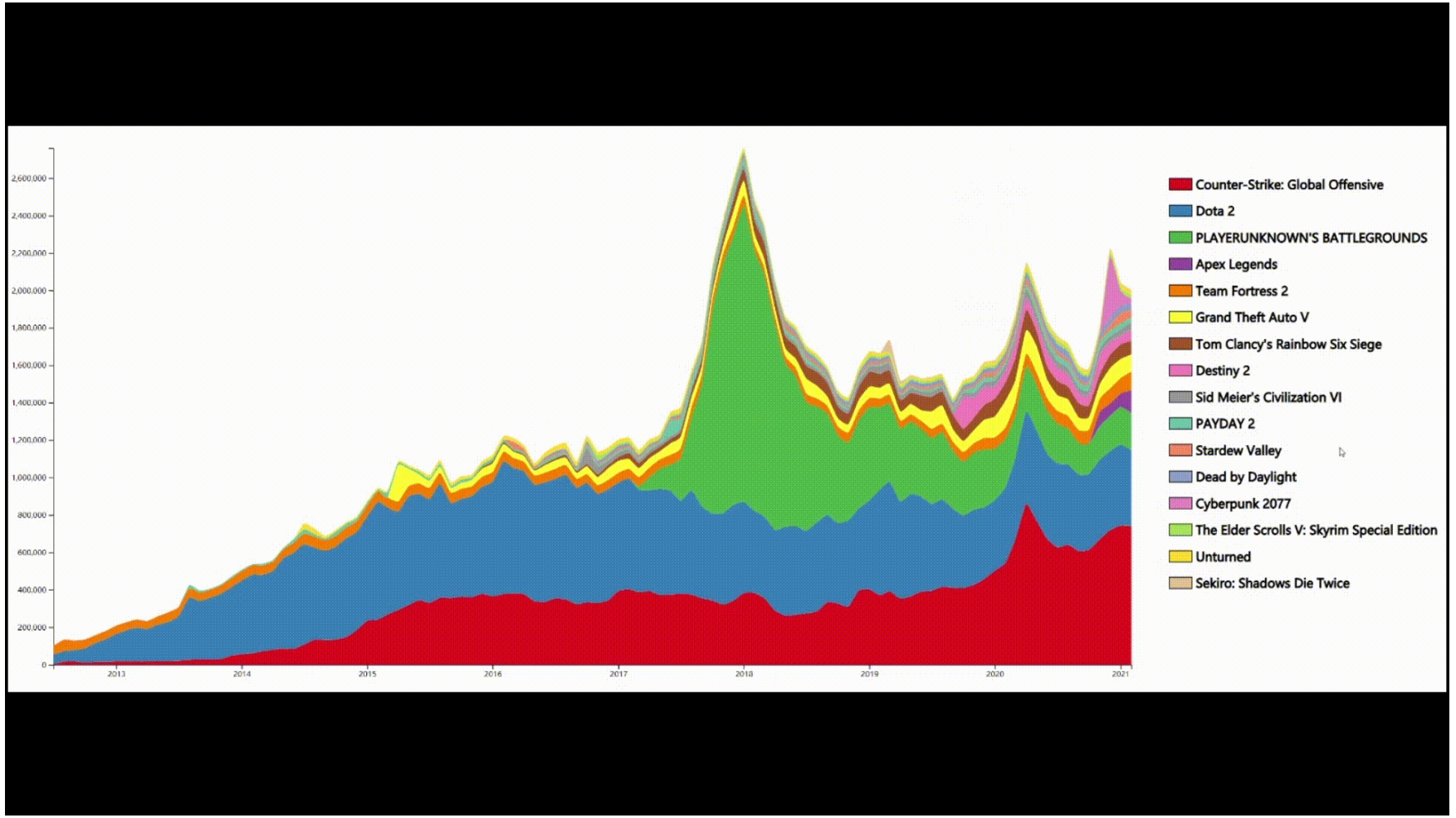
基于‘stack’实现堆叠柱状图

- 编程实例：

```
// start to do data-join;
g.selectAll('.datagroup').data(naiveStack).join('g')
.attr('class', 'datagroup')
.attr('fill', d => color(d.key))
.selectAll('.datarect').data(d => d).join('rect')
.attr('class', 'datarect')
.attr('y', d => yScale(d[1]))
.attr('x', d => xScale(xValue(d.data)))
.attr('height', d => yScale(d[0]) - yScale(d[1]))
.attr('width', xScale.bandwidth());
```

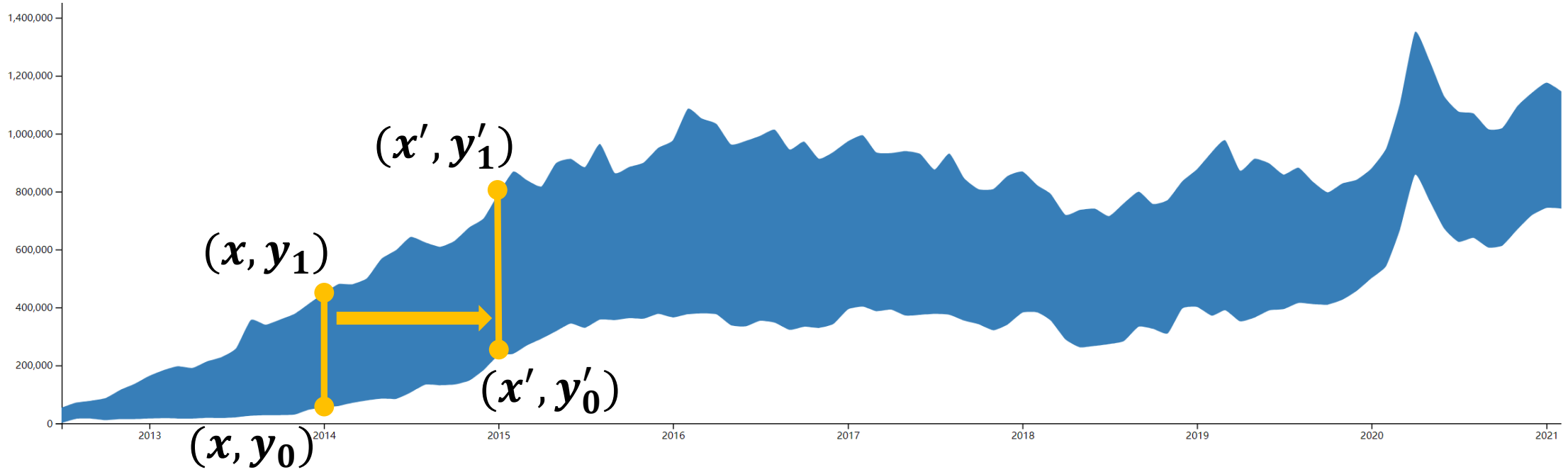
基于‘stack’实现主题河流

- 数据来源：
  - https://www.kaggle.com/michau96/popularity-of-games-on-steam
- Steam平台各主流游戏热度趋势变化可视化。
- 数据与堆叠的数据：
  - 横轴：时间（年、月）。
  - 纵轴：热度（在线人数）即需要堆叠的内容。



基于‘stack’实现主题河流

- 使用D3.js实现主题河流，每条‘河流’都是一个图元，如图中每个颜色都只对应一个<path>。
- 主题河流在D3.js中的本质为，随x的变化，在上y_0、下y_1端点分别插值。
  - 故，x、y_0、y_1 需要编程者以数组的形式给出。



d3.area(…)

- let path = d3.area()
  - 定义一个‘area’生成**函数**，输入为数组，输出为<path>的‘d’属性值。
  - path.**x**(d => xScale(d.data.ym)) // 设置x轴的取值随绑定数据的.data.ym属性。
  - path.**y1**(d => yScale(d[1])) // 设置上界y轴的取值随绑定数据的[1]。
  - path.**y0**(d => yScale(d[0])) //设置上界y轴的取值随绑定数据的[1]。
- path.curve(…):
  - 设置‘area’上边界与下边界的拟合方式，同d3.line().curve()。
  - e.g., path.curve(d3.curveCardinal.tension(0.5));

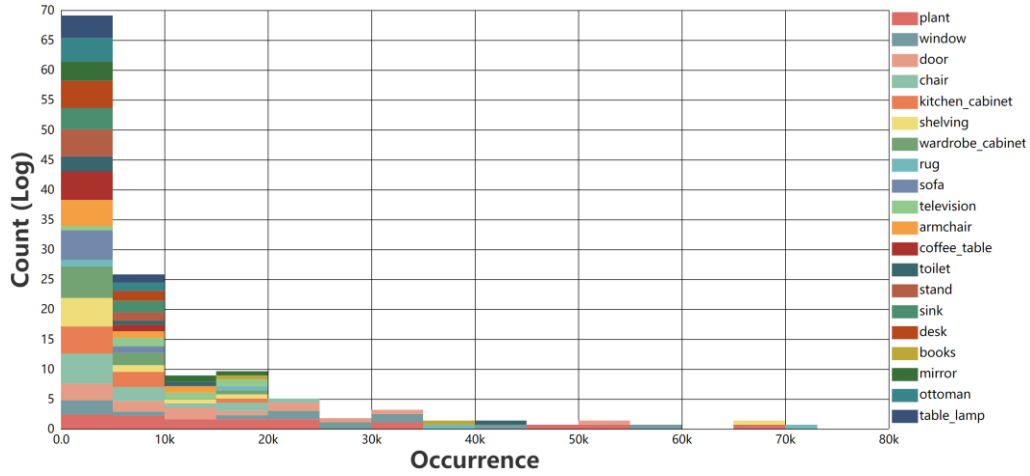
基于‘stack’实现主题河流

- 编程实例：
  - 仍需要对堆叠后的数据取‘嵌套’的最大值与最小值。

```
// define an area generator;
let area = d3.area()
.x(d => xScale(d.data.ym))
.y1(d => yScale(d[1])).y0(d => yScale(d[0]))
.curve(d3.curveCardinal.tension(0.5));

// data-join;
g.selectAll('.riverPath')
.data(stackedData, d => d.key).join('path')
.attr('class', 'riverPath').attr('d', area).attr('fill', d => color(d.key));
```

Tip：堆叠与直方图



- D3.js也含有其他数据预处理的方法
- d3.histogram: 用于将数据按照某一属性分布在不同的区域
  - 常用于绘制**直方图**。
- d3.pie: 用于将数据映射到圆周的各个弧度
  - 常用于绘制**饼图**。
- Stack与Histogram的结合？
  - code: https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/stack_histogram.html
  - Url: http://127.0.0.1:11666/stack_histogram

Tip：比例尺不满足‘结合律’

- 计算矩形高度‘height’时，需基于堆叠数据做减法。
- 比例尺不满足‘结合律’：
  - **let linearScale = d3.scaleLinear().domain([1,10]).range([800, 1000])**
  - linearScale(8-6) // 822.22
  - linearScale(8) - linearScale(6) // 44.44
- 故，尽可能先用比例尺映射数据，再做运算！（**绿色**框）

```
g.selectAll('.datagroup').data(naiveStack).join('g')
.attr('class', 'datagroup')
.attr('fill', d => color(d.key))
.selectAll('.datarect').data(d => d).join('rect')
.attr('class', 'datarect')
.attr('y', d => yScale(d[1]))
.attr('x', d => xScale(xValue(d.data)))
.attr('height', d => yScale(d[0]) - yScale(d[1]))
.attr('width', xScale.bandwidth());
```