山东大学计算机科学与技术学院

大数据分析实践课程实验报告

学号：202300130003	姓名：肖皓天	班级：数据 23
实验题目：实验 3		
实验学时：32	实验日期：2025/10/18	

实验目标：

Add a new vis function based on the open source spreadsheet

流程描述：

1、HTML 结构与资源引入

```
<!DOCTYPE html>
<html lang="zh-CN">

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>数据可视化：堆叠柱状图 / 堆叠面积图</title>
    <link rel="stylesheet" href="https://unpkg.com/x-data-spreadsheet@1.1.5/dist/xspreadsheet.css" />
    <script src="https://unpkg.com/x-data-spreadsheet@1.1.5/dist/xspreadsheet.js"></script>
    <script src="https://unpkg.com/x-data-spreadsheet@1.1.9/dist/locale/zh-cn.js"></script>
    <script src="https://d3js.org/d3.v6.min.js"></script>
    <style>
        :root {
            --sheetW: 380px;
            --panelH: 460px;
        }

        body {
            font-family: system-ui, -apple-system, "Segoe UI", Roboto, "Helvetica Neue", Arial, sans-serif;
        }

        .row {
            display: flex;
            align-items: flex-start;
            gap: 16px;
        }

        #sheet-wrap {
            flex: 0 0 var(--sheetW);
        }

        #ctrl {
            margin: 6px 0;
        }

        #xspreadsheet {
            width: var(--sheetW);
            height: calc(var(--panelH) - 40px);
            border: 1px solid #eee;
        }

        #my_dataviz {
            flex: 1 1 0;
            height: var(--panelH);
            border: 1px solid #f2f2f2;
        }
    </style>
</head>
```

界面布局采用 Flexbox 模型，将页面划分为左右两大区域：左侧 sheet-wrap 用于容纳 x-spreadsheet 电子表格实例和图表类型切换的单选按钮；右侧 my_dataviz 则作为 D3.js 绘制 SVG 图表的目标容器。

2、表格初始化与数据绑定

```
<script>
    x_spreadsheet.locale("zh-cn");
    const xs = x_spreadsheet("#xspreadsheet", {
        mode: 'edit', showToolbar: true, showGrid: true, showContextmenu: true,
        view: {
            height: () => document.getElementById('xspreadsheet').clientHeight,
            width: () => document.getElementById('xspreadsheet').clientWidth
        },
        row: { len: 15, height: 25 },
        col: { len: 8, width: 100, indexWidth: 60, minWidth: 60 },
    });

    // 初始数据
    xs.cellText(0, 1, "计算机").cellText(0, 2, "法学").reRender();
    xs.cellText(1, 0, "2017").cellText(1, 1, "23").cellText(1, 2, "15").reRender();
    xs.cellText(2, 0, "2018").cellText(2, 1, "36").cellText(2, 2, "26").reRender();
    xs.cellText(3, 0, "2019").cellText(3, 1, "23").cellText(3, 2, "33").reRender();
    xs.cellText(4, 0, "2020").cellText(4, 1, "22").cellText(4, 2, "10").reRender();
```

首先对 x-spreadsheet 进行初始化配置。接着创建表格实例。核心的交互功能通过 xs.on('cell-edited', update) 实现，该行代码监听了所有单元格的编辑事件，一旦用户修改数据，就会自动调用 update() 函数重绘图表，实现了数据与视图的实时同步。最后。向表格中填入关于“计算机”和“法学”两个专业在不同年份人数的初始样本数据。

3、颜色函数与表格数据读取

```
function getColor(i) {
    const palette = ['#5ab1ef', '#ffb980', '#d87a80', '#2ec7c9', '#b6a2de'];
    return palette[i % palette.length];
}

function readSheet() {
    let yTitle = [], xTitle = [], rows = 0, cols = 0, data = [];
    for (let i = 1; i < 200; i++) {
        const c = xs.cell(i, 0); if (!c || !c.text) { rows = i; break; }
        data.push([]); yTitle.push(c.text);
    }
    if (!rows) rows = yTitle.length + 1;
    if (!yTitle.length) return { ok: false, msg: '首列缺少分组名' };
    for (let j = 1; j < 200; j++) {
        const c = xs.cell(0, j); if (!c || !c.text) { cols = j; break; }
        xTitle.push(c.text);
    }
    if (!cols) cols = xTitle.length + 1;
    if (xTitle.length === 0) return { ok: false, msg: '首行缺少系列名' };
    for (let i = 1; i < rows; i++) {
        for (let j = 1; j < cols; j++) {
            const c = xs.cell(i, j);
            if (!c || c.text === undefined || isNaN(+c.text)) return { ok: false, msg: `数据错误 at (${i + 1}, ${j + 1})` };
            data[i - 1][j - 1] = +c.text;
        }
    }
    return { ok: true, yTitle, xTitle, data };
}
```

为了确保图表视觉上的一致性和可读性，我们定义了一个 getColor(i) 函数。它通过一个预设的调色板数组，为不同的数据系列（如“计算机”、“法学”）分配一个固定的颜色，使得图表中的图形颜色与右侧图例一一对应。readSheet() 函数承担了数据转换的关键任务，它负责从电子表格中提取数据并格式化。该函数会智能地遍历表格，从 A 列（第二行起）读取分组名（yTitle，如年份），从第一行（B 列起）读取系列名（xTitle，如专业），并捕获交叉区域的数值。

4. 核心逻辑：数据堆叠处理与图表绘制（update 函数）

```
function update() {
    const type = document.querySelector('input[name="chart-type"]:checked').value;
    const parsed = readSheet();
    const myDiv = document.getElementById('my_dataviz');
    const containerW = myDiv.clientWidth;
    const containerH = myDiv.clientHeight;

    d3.select("#my_dataviz").selectAll('svg').remove();
    if (!parsed.ok) { alert(parsed.msg); return; }

    const { yTitle, xTitle, data: matrix } = parsed;
    const data = yTitle.map((group, i) => {
        const row = { group };
        xTitle.forEach((key, j) => { row[key] = matrix[i][j]; });
        return row;
    });


    const stack = d3.stack().keys(xTitle);
    const stackedData = stack(data);

    const yMax = d3.max(stackedData, d => d3.max(d, item => item[1]));

    const margin = { top: 40, right: 120, bottom: 40, left: 50 };
    const width = Math.max(280, containerW - margin.left - margin.right);
    const height = Math.max(240, containerH - margin.top - margin.bottom);

    const svg = d3.select("#my_dataviz").append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom);

    const g = svg.append("g").attr("transform", `translate(${margin.left},${margin.top})`);

    const x = d3.scaleBand().domain(yTitle).range([0, width]).padding(0.25);
    const y = d3.scaleLinear().domain([0, yMax * 1.15]).nice().range([height, 0]);

    g.append("g").attr("transform", `translate(0,${height})`).call(d3.axisBottom(x));
    g.append("g").call(d3.axisLeft(y));
    g.append("g").call(d3.axisLeft(y).tickSize(-width).tickFormat("")).selectAll("line").attr("stroke", "#eee").attr("stroke
```

update() 函数流程如下：

首先，通过 document.querySelector 获取用户选择的图表类型（堆叠柱状图或堆叠面积图），并调用 readSheet() 获取格式化后的数据。

利用 D3 的 d3.stack() 方法，将原始的行/列数据转换成适用于堆叠图的特殊格式。d3.stack() 会计算出每个数据点在堆叠后的起始值（y0）和结束值（y1）。

与普通图表不同，堆叠图的 Y 轴最大值是各系列在同一分组下的总和的最大值。我们通过 d3.max() 遍历堆叠后的数据，精确计算出这个总和最大值 yMax，并以此为基础创建 Y 轴比例尺 y，确保图表能完整显示所有数据。

stacked-bar：程序会为每个系列(stack-group)绘制一组矩形。矩形的 y 坐标和 height 直接使用 d3.stack() 计算出的 d[0]（y0）和 d[1]（y1）来确定。

stacked-area：使用 d3.area() 面积生成器。我们将面积的下边界 y0 和上边界 y1 分别绑定到堆叠数据 d[0] 和 d[1]。

最后，在图表右侧绘制图例。

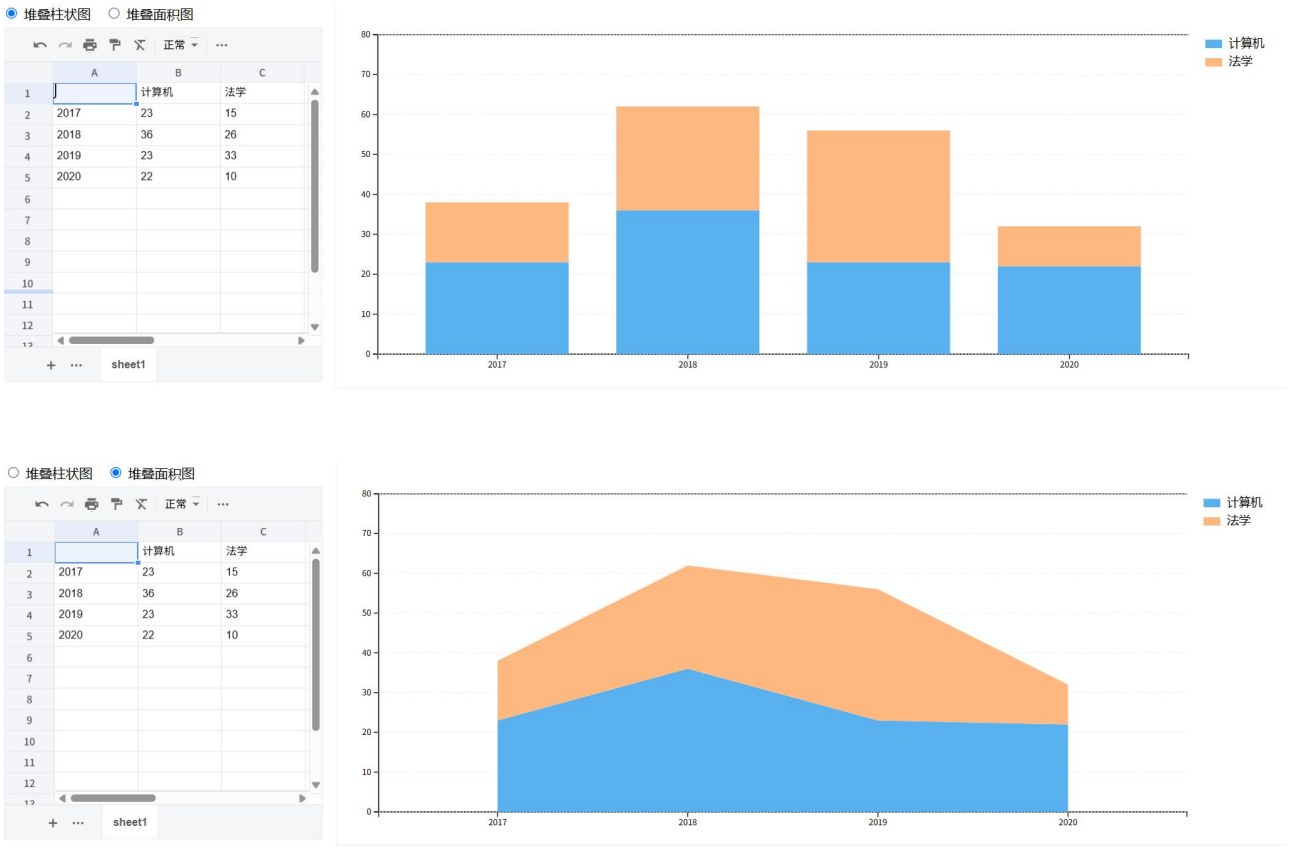
5、事件绑定与首次渲染

```
xs.on('cell-edited', update);
document.querySelectorAll('input[name="chart-type"]').forEach(el => el.addEventListener('change', update));
window.addEventListener('resize', update);


update();
```

代码末尾进行事件绑定。

6.结果：



结论分析与体会：

通过本次实验，我深刻体会到 D3.js 在处理复杂数据结构时的强大能力，特别是 d3.stack() 这类布局生成器，极大地简化了堆叠类图表的实现逻辑。它将复杂的位置计算抽象化，让开发者可以更专注于数据的映射关系。

与基础的柱状图/折线图相比，堆叠图在展示部分与整体的关系上具有显著优势。它不仅能看出各分量的变化趋势，还能直观地反映出总量的变化情况，为数据分析提供了更丰富的维度。