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：202300130051	姓名： 汤冉	班级：数据班
实验题目：实验三		
实验学时： 2	实验日期：2025.10.16	

实验目的：
在开源电子表格基础上添加新的可视化功能。

硬件环境：
计算机

软件环境：
vscode,

实验步骤与内容：

1、HTML 结构与资源引入

```
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>数据可视化：柱状图 或 折线图（二选一）</title>
    <link rel="stylesheet" href="https://unpkg.com/x-data-spreadsheet@1.1.5/dist
    <script src="https://unpkg.com/x-data-spreadsheet@1.1.5/dist/xspreadsheet.js
    <script src="https://unpkg.com/x-data-spreadsheet@1.1.9/dist/locale/zh-cn.js
    <script src="https://d3js.org/d3.v6.min.js"></script>
    <style>
        #xspreadsheet {
            width: 400px;
            height: 500px;
            margin: 0;
            display: inline-block;
            vertical-align: top;
        }
        #ctrl {
            margin: 6px 0 0 6px;
        }
        #my_dataviz {
            width: 1000px;
            height: 900px;
            margin: 0;
            display: inline-block;
            vertical-align: top;
        }
        .line {
            fill: none;
            stroke-width: 2.5px;
        }
    </style>
</head>
```

定义两块区域（左表格 + 右图表）和基本样式。xspreadsheet：放电子表格和“柱状图/折线图”的单选按钮。my_dataviz：放 D3 画出来的 SVG 图。line 仅用于折线的线宽。

2、表格初始化

```
x_spreadsheet.locale("zh-cn");
var xs = x_spreadsheet("#xspreadsheet", {
    mode: 'edit',
    showToolbar: true, showGrid: true, showContextmenu: true,
    view: { height: () => document.documentElement.clientHeight, width: () => 400 },
    row: { len: 15, height: 25 },
    col: { len: 8, width: 100, indexWidth: 60, minWidth: 60 },
    style: {
        bgcolor: '#ffffff', align: 'left', valign: 'middle', textwrap: false,
        strike: false, underline: false, color: '#0a0a0a',
        font: { name: 'Helvetica', size: 10, bold: false, italic: false },
    },
});

// 初值
xs.on('cell-edited', update);
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();
```

初始化电子表格（x-spreadsheet）：页面用 x-data-spreadsheet 创建一个可编辑表格并写入演示数据。关键做法是先设中文界面 x_spreadsheet.locale，再通过 x_spreadsheet 生成实例，row/col/style/view 控制行列数量、单元格尺寸和字体。用 xs.on('cell-edited', update) 监听单元格变更，任何编辑都会触发 update() 重绘图表。示例数据用 cellText.reRender() 写入，并约定表格结构：第一列（A2 开始）是年份等分组名，第一行（B1 开始）是系列名（如学科），交叉区域为数值。

3、颜色函数与读取表格数据

```
var yTitle = [], xTitle = [], rows = 0, cols = 0, data = [];

for (var i = 1; i < 200; i++) {
    if (xs.cell(i, 0) === null || xs.cell(i, 0).text === undefined || xs.cell(i, 0).text
    data.push([]); yTitle.push(xs.cell(i, 0).text);
}
if (!rows) rows = yTitle.length + 1;
if (yTitle.length === 0) return { ok: false, msg: '首列缺少分组名（从第2行开始）' };

for (var j = 1; j < 200; j++) {
    if (xs.cell(0, j) === null || xs.cell(0, j).text === undefined || xs.cell(0, j).text
    xTitle.push(xs.cell(0, j).text);
}
if (!cols) cols = xTitle.length + 1;

// 自动补系列名（如果首行没有）
if (xTitle.length === 0) {
    var cnt = 0;
    for (var j = 1; j < 200; j++) {
        var raw = xs.cell(1, j)?.text;
        if (raw === undefined || raw === "" || isNaN(+raw)) break;
        cnt++;
    }
    if (cnt === 0) return { ok: false, msg: '无法自动识别系列列，请在第2行第2列起输入数字' }
    for (var k = 0; k < cnt; k++) xTitle.push('系列' + (k + 1));
    cols = cnt + 1;
}

for (var i = 1; i < rows; i++) {
    for (var j = 1; j < cols; j++) {
        var cell = xs.cell(i, j);
```

颜色函数：为每个系列分配稳定的配色，使柱子、折线与图例颜色一致。函数 getColor(idx) 从预设调色板数组按下标取色，idx 不同得到的颜色不同但在全图保持一致。readSheet 函数的

作用是把表格中的单元格内容整理成绘图所需的数据结构：它先从 A2 往下读取年份作为分组名得到 yTitle，再从 B1 往右读取系列名得到 xTitle，若首行为空则根据第 2 行起的数字列数自动补上“系列 1、系列 2…”；接着双循环读取数值并转成数字，遇到空格或非数字会返回错误提示。最终返回 { ok:true, yTitle, xTitle, data }，出错时返回 { ok:false, msg:'...' }。

4、互斥选择 + 绘图（update）

update 函数的作用是根据单选按钮的选择来决定绘制柱状图还是折线图，并完成整个绘制流程。它首先读取当前选择的图表类型（只能是 'bar' 或 'line'），然后调用 readSheet() 获取年份 yTitle、系列名 xTitle 和二维数值矩阵 matrix，再把它们转成行对象数组 data，同时计算所有值的最大值用来设置 y 轴上限。接下来建立比例尺：x 控制年份在横轴的位置，xSub 控制每个年份内部不同系列的子位置（柱状图用），y 负责数值到纵轴的映射，并绘制坐标轴和背景网格。随后根据类型分支绘制：如果是柱状图则为每个年份画多个矩形柱并加数值标签；如果是折线图则为每个系列生成一条折线，并在折线上加圆点和数据标签。最后在右侧绘制图例，包括系列的颜色块和名称，保证和图中柱子或折线的颜色一致。

```
function update() {
    // 只会有一个被选中
    const type = document.querySelector('input[name="chart-type"]:checked').value;

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

    var yTitle = parsed.yTitle, xTitle = parsed.xTitle, matrix = parsed.data;
    var max = 0, data = [];
    for (var i = 0; i < yTitle.length; i++) {
        var jsd = { group: yTitle[i] };
        for (var j = 0; j < xTitle.length; j++) {
            var v = matrix[i][j]; if (v > max) max = v;
            jsd[xTitle[j]] = v;
        }
        data.push(jsd);
    }

    const margin = { top: 40, right: 160, bottom: 40, left: 50 },
        width = 600 - margin.left - margin.right,
        height = 420 - margin.top - margin.bottom;

    d3.select("#my_dataviz").selectAll('svg').remove();

    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(0.22);
    const y = d3.scaleLinear().domain([0, max * 1.12]).nice().range([height, 0]);
    const xSub = d3.scaleBand().domain(xTitle).range([0, x.bandwidth()]).padding(0.18);
```

```
if (type === 'bar') {
    // — 只绘柱状图 — //
    const groupG = g.selectAll(".bar").data(data).join("g")
        .attr("class", "bar").attr("transform", d => `translate(${x(d.group)},0)`);

    groupG.selectAll("rect")
        .data(d => xTitle.map(key => ({ key, value: d[key] })))
        .join("rect")
        .attr("x", d => xSub(d.key))
        .attr("y", d => y(d.value))
        .attr("width", xSub.bandwidth())
        .attr("height", d => height - y(d.value))
        .attr("fill", (d, i) => getColor(i));

    groupG.selectAll("text")
        .data(d => xTitle.map(key => ({ key, value: d[key] })))
        .join("text")
        .attr("x", d => xSub(d.key) + xSub.bandwidth() * 0.5)
        .attr("y", d => y(d.value) - 8)
        .attr("text-anchor", "middle")
        .text(d => d.value);
} else if (type === 'line') {
```

```
    // — 只绘折线图 — //
    const line = d3.line()
        .x(d => x(d.group) + x.bandwidth() / 2)
        .y(d => y(d.value));

    xTitle.forEach((seriesName, si) => {
        const seriesData = yTitle.map(group => ({ group, value: data.find(r => r.group =

        g.append("path")
            .datum(seriesData)
            .attr("class", "line")
            .attr("stroke", getColor(si))
            .attr("d", line);

        g.selectAll(".dot-" + si)
            .data(seriesData).join("circle")
            .attr("cx", d => x(d.group) + x.bandwidth() / 2)
            .attr("cy", d => y(d.value))
            .attr("r", 3.5)
            .attr("fill", getColor(si));

        g.selectAll(".label-" + si)
            .data(seriesData).join("text")
            .attr("x", d => x(d.group) + x.bandwidth() / 2)
            .attr("y", d => y(d.value) - 10)
            .attr("text-anchor", "middle")
            .attr("fill", "#333")
            .text(d => d.value);
```

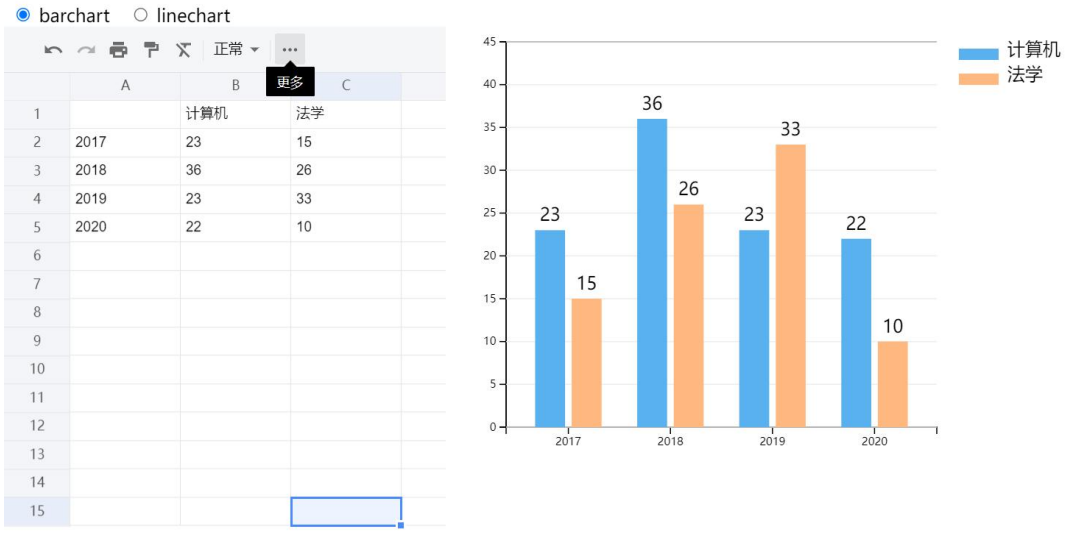
6、事件绑定与首次渲染

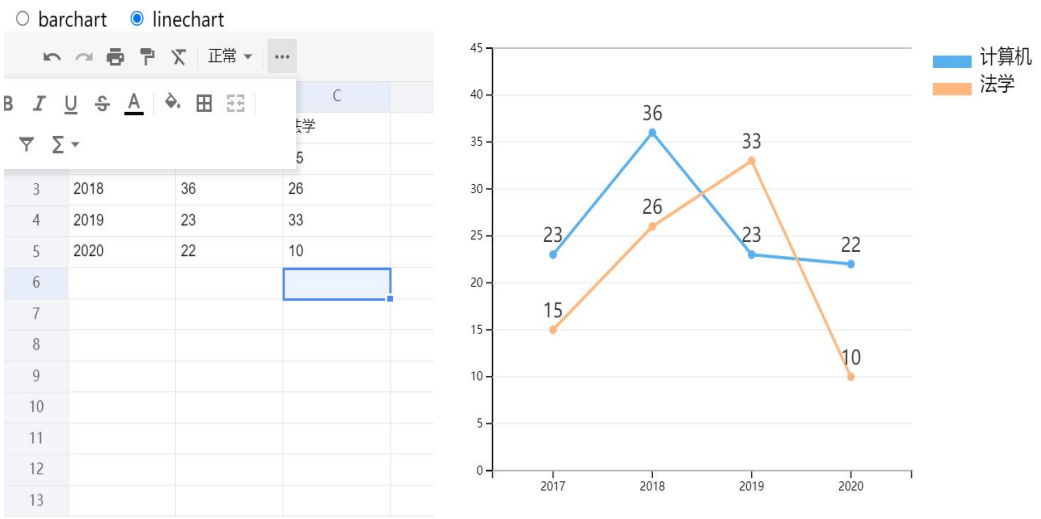
```
// 单选切换事件
document.querySelectorAll('input[name="chart-type"]').forEach(el => {
    el.addEventListener('change', update);
});

// 首次渲染
update();
</script>
```

这部分代码的作用是保证在切换单选按钮或编辑表格时能自动更新图表，同时页面初次加载时先绘制一次。实现方法是通过 document.querySelectorAll('input[name="chart-type"]') 为

所有单选按钮绑定 change 事件监听器，只要用户切换选项就调用 update() 重新渲染；并在代码结尾直接调用一次 update()，让页面打开时立即生成初始图表。

最终效果展示：





结论分析与体会：

通过本实验，我体会到前端可视化的核心并不是单纯画图，而是数据组织 + 图形映射。只有把数据规范化，图形才能正确呈现。另一方面，x-data-spreadsheet 与 D3 的结合展示了前端开发的灵活性：一个负责数据管理，一个负责可视化，配合得当就能快速搭建完整的应用。实践过程中也发现，一些细节（比如 y 轴范围预留空间、颜色统一、事件绑定）对用户体验影响很大，需要在实现中不断调试和优化。

注：实验报告的命名规则：学号_姓名_实验 n_班级