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：202300130178	姓名：刘爽	班级：23 数据
实验题目：电子表格 I		
实验学时：4	实验日期：2025.10.17	

实验目标：

Add a new vis function based on the open source spreadsheet

实验环境：

Window，vscode，利用 x-spreadsheet 进行表格操作，利用 d3 进行可视化

作品描述：

1. 实验背景：

在数据可视化与交互分析领域，传统工具常存在“数据编辑”与“可视化展示”脱节的问题：电子表格虽便于数据录入，但可视化功能单一且需手动更新图表；

专业可视化工具虽图表丰富，却难以快速修改原始数据。随着前端技术发展，开源表格库（x-data-spreadsheet）与可视化库（D3.js）的结合，为实现“数据编辑 - 实时可视化”联动提供了轻量化解决方案。

2. 实验目标：

（1）搭建一个**集表格编辑、双图表（柱状图 + 折线图）**展示于一体的交互系统；

（2）实现 **“表格数据修改 - 图表实时更新”** 的联动效果，支持图表**显示 / 隐藏**控制；

（3）提供**数据保存、加载、重置**功能，保障数据安全性与可复用性；

（4）优化初始化逻辑，修复潜在的表格与图表加载不同步问题。

3. 代码以及说明：

（1） 核心数据处理：**getTableData () 函数，该函数是表格与图表的 “数据桥梁”，负责从 x-spreadsheet 表格中提取、校验并格式化数据，为可视化提供标准输入**

① 数据校验：过滤非数字数据，避免图表渲染错误；

② 格式统一：补全空数据为 0，确保 “年份 - 学科” 数据矩阵对齐；

③ 信息提取：分离标题与数值，符合 D3.js 图表对“轴标签 - 数据”的输入要求。

```
function getTableData() {
    let data = [];
    let yTitle = [];
    let xTitle = [];
    let rows = 0;
    let cols = 0;

    // 获取所有数据
    const sheetData = xs.getData();
    if (!sheetData || !sheetData[0] || !sheetData[0].rows) return null;

    const rowsData = sheetData[0].rows;

    // 获取列标题（第一行，从第二列开始）
    if (rowsData["0"] && rowsData["0"].cells) {
        const firstRow = rowsData["0"].cells;
        for (let i = 1; i < 20; i++) {
            if (firstRow[i] && firstRow[i].text) {
                xTitle.push(firstRow[i].text);
            } else {
                break;
            }
        }
    }

    // 获取行标题和数据（从第二行开始）
    for (let i = 1; i < 20; i++) {
        if (rowsData[i] && rowsData[i].cells && rowsData[i].cells["0"] && rowsData[i].cells["0"].text) {
            yTitle.push(rowsData[i].cells["0"].text);
            data.push([]);
```

```
            for (let j = 1; j < 20; j++) {
                if (rowsData[i].cells[j] && rowsData[i].cells[j].text && !isNaN(+rowsData[i].cells[j].text)) {
                    data[i - 1].push(+rowsData[i].cells[j].text);
                } else if (rowsData[i].cells[j] && rowsData[i].cells[j].text) {
                    alert("表格中存在无效数据，请输入数字！");
                    return null;
                } else {
                    break;
                }
            }
        } else {
            break;
        }
    }

    // 确保所有行的数据长度一致
    const maxCols = Math.max(...data.map(row => row.length));
    data = data.map(row => {
        while (row.length < maxCols) {
            row.push(0);
        }
        return row;
    });

    return {
        data,
        yTitle,
        xTitle: xTitle.slice(0, maxCols),
        maxValue: data.length > 0 ? Math.max(...data.flat()) : 0
    };
}
```

（2） 柱状图实现：**drawBarChart（）函数，基于 D3.js 实现“年份 - 学科”对比柱状图，核心是通过“分组柱状图”展示同一年份下不同学科的数值差异**

① 分组柱状图：通过 xScale（年份）+ xSubgroup（学科）实现 “年份下嵌套学科” 的柱子布局，清晰对比同一时间维度的类别差异；
② 交互优化：柱子顶部显示数值、图例悬浮提示，避免用户 “估读” 数据；
③ 响应式适配：通过固定内宽高 + margin 偏移，确保图表在不同容器中显示完整。

```
function drawBarChart(dataObj) {
    const { data, yTitle, xTitle, maxValue } = dataObj;
    const container = "#bar-chart";
    d3.select(container).selectAll("svg").remove();

    if (data.length === 0 || xTitle.length === 0) return;

    const margin = { top: 40, right: 120, bottom: 60, left: 60 };
    const width = 700 - margin.left - margin.right;
    const height = 400 - margin.top - margin.bottom;

    const svg = d3.select(container)
        .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
        .append("g")
        .attr("transform", `translate(${margin.left}, ${margin.top})`);

    const x = d3.scaleBand().domain(yTitle).range([0, width]).padding(0.2);
    svg.append("g")
        .attr("transform", `translate(0, ${height})`)
        .call(d3.axisBottom(x).tickSizeOuter(0))
        .selectAll("text")
        .style("text-anchor", "end")
        .attr("dx", "-.8em")
        .attr("dy", ".15em")
        .attr("transform", "rotate(-45)");
```

```
svg.append("text")
    .attr("class", "axis-label")
    .attr("transform", `translate(${width / 2}, ${height + margin.bottom - 10})`)
    .style("text-anchor", "middle")
    .text("年份");

const y = d3.scaleLinear().domain([0, maxValue || 1]).range([height, 0]).nice();
svg.append("g").call(d3.axisLeft(y));

// Y轴标签
svg.append("text")                any
    .attr("class", "axis-label")
    .attr("transform", "rotate(-90)")
    .attr("y", 0 - margin.left)
    .attr("x", 0 - (height / 2))
    .attr("dy", "1em")
    .style("text-anchor", "middle")
    .text("数值");

const xSubgroup = d3.scaleBand().domain(xTitle).range([0, x.bandwidth()]).padding(0.05);

const chartData = yTitle.map((group, i) => {
    const item = { group };
    xTitle.forEach((key, j) => item[key] = data[i] ? data[i][j] || 0 : 0);
    return item;
});

svg.append("g")
    .selectAll("g")
    .data(chartData)
```

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

    svg.append("g")
        .selectAll("g")
        .data(chartData)
        .join("g")
        .attr("transform", d => `translate(${x(d.group)}, 0)`)
        .selectAll("text")
        .data(d => xTitle.map(key => ({ key, value: d[key] })))
        .join("text")
        .attr("x", d => xSubgroup(d.key) + xSubgroup.bandwidth()/2)
        .attr("y", d => y(d.value) - 5)
        .text(d => d.value)
        .attr("text-anchor", "middle")
        .style("font-size", "12px")
        .style("font-weight", "bold");

    // 图例
    const legend = svg.selectAll(".bar-legend")
        .data(xTitle)
        .join("g")
        .attr("class", "bar-legend")
        .attr("transform", (d, i) => `translate(${width + 20}, ${i * 25})`);
```

```
    // 图例
    const legend = svg.selectAll(".bar-legend")
        .data(xTitle)
        .join("g")
        .attr("class", "bar-legend")
        .attr("transform", (d, i) => `translate(${width + 20}, ${i * 25})`);

    legend.append("rect")
        .attr("width", 18)
        .attr("height", 18)
        .attr("fill", (d, i) => getColor(i));

    legend.append("text")
        .attr("x", 25)
        .attr("y", 12)
        .text(d => d)
        .style("font-size", "14px")
        .attr("class", "legend");
}
```

（3） 折线图实现：drawLineChart () 函数，基于 D3.js 实现“学科 - 年份”趋势折线图，核心是通过“多线条”展示同一学科在不同年份的数值变化

① 采用 d3.scalePoint()，让年份刻度点居中，更直观；
② 通过 d3.curveMonotoneX() 实现平滑曲线，避免折线生硬断裂；
③ 图例用 “线条” 替代柱状图的 “色块”，与图表类型强关联。

```
function drawLineChart(dataObj) {
    const { data, yTitle, xTitle, maxValue } = dataObj;
    const container = "#line-chart";
    d3.select(container).selectAll("svg").remove();

    if (data.length === 0 || xTitle.length === 0) return;

    const margin = { top: 40, right: 120, bottom: 60, left: 60 };
    const width = 700 - margin.left - margin.right;
    const height = 400 - margin.top - margin.bottom;

    const svg = d3.select(container)
        .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
        .append("g")
        .attr("transform", `translate(${margin.left}, ${margin.top})`);

    const x = d3.scalePoint().domain(yTitle).range([0, width]).padding(0.1);
    svg.append("g")
        .attr("transform", `translate(0, ${height})`)
        .call(d3.axisBottom(x).tickSizeOuter(0))
        .selectAll("text")
        .style("text-anchor", "end")
        .attr("dx", "-.8em")
        .attr("dy", ".15em")
        .attr("transform", "rotate(-45)");

    // X轴标签
    svg.append("text")
        .attr("class", "axis-label")
```

```
        .style("text-anchor", "middle")
        .text("年份");

    const y = d3.scaleLinear().domain([0, maxValue || 1]).range([height, 0]).nice();
    svg.append("g").call(d3.axisLeft(y));

    // Y轴标签
    svg.append("text")
        .attr("class", "axis-label")
        .attr("transform", "rotate(-90)")
        .attr("y", 0 - margin.left)
        .attr("x", 0 - (height / 2))
        .attr("dy", "1em")
        .style("text-anchor", "middle")
        .text("数值");

    const lineGenerator = d3.line()
        .x((d, i) => x(yTitle[i]))
        .y(d => y(d))
        .curve(d3.curveMonotoneX);

    xTitle.forEach((subject, idx) => {
        const subjectData = data.map(row => row[idx] || 0);

        svg.append("path")
            .datum(subjectData)
            .attr("fill", "none")
            .attr("stroke", getColor(idx))
            .attr("stroke-width", 2.5)
            .attr("d", lineGenerator);
```

```
        .attr("stroke-width", 1);

    svg.selectAll(`.line-text-${idx}`)
        .data(subjectData)
        .join("text")
        .attr("class", `line-text-${idx}`)
        .attr("x", (d, i) => x(yTitle[i]))
        .attr("y", d => y(d) - 10)
        .text(d => d)
        .attr("text-anchor", "middle")
        .style("font-size", "12px")
        .style("font-weight", "bold");
});

// 图例
const legend = svg.selectAll(".line-legend")
    .data(xTitle)
    .join("g")
    .attr("class", "line-legend")
    .attr("transform", (d, i) => `translate(${width + 20}, ${i * 25})`);

legend.append("line")
    .attr("x1", 0)
    .attr("y1", 9)
    .attr("x2", 18)
    .attr("y2", 9)
    .attr("stroke", (d, i) => getColor(i))
    .attr("stroke-width", 2.5);

legend.append("text")
```

```
    legend.append("text")
        .attr("x", 25)
        .attr("y", 12)
        .text(d => d)
        .style("font-size", "14px")
        .attr("class", "legend");
}
```

（4） 数据管理功能代码解析（保存 / 加载 / 重置）：

① 数据管理功能基于浏览器 LocalStorage 实现，无需后端数据库，即可实现数据持久化存储；

```
// 保存数据到本地存储
function saveData() {
    try {
        const data = xs.getData();
        localStorage.setItem('spreadsheetData', JSON.stringify(data));
        showStatus('数据保存成功！', 'success');
    } catch (error) {
        console.error('保存数据时出错:', error);
        showStatus('保存数据时出错: ' + error.message, 'error');
    }
}
```

依赖 xs.getData() 方法获取表格完整数据（包含 Sheet 名称、行 / 列结构、单元格内容）；

使用 JSON.stringify() 处理数据，解决 LocalStorage 对复杂对象的存储限制；

加入 try-catch 异常处理，避免因浏览器存储权限、空间不足等问题导致功能崩溃。

② **加载数据：loadData () 函数**

```
// 从本地存储加载数据
function loadData() {
    try {
        const savedData = localStorage.getItem('spreadsheetData');
        if (savedData) {
            const data = JSON.parse(savedData);
            xs.loadData(data);
            showStatus('数据加载成功！', 'success');
            // 加载数据后更新图表
            setTimeout(updateAllViz, 100);
        } else {
            showStatus('没有找到保存的数据', 'error');
        }
    } catch (error) {
        console.error('加载数据时出错:', error);
        showStatus('加载数据时出错: ' + error.message, 'error');
    }
}
```

通过 localStorage.getItem() 读取数据，先判断是否存在存储内容，避免空值；

延迟调用 updateAllViz()，解决表格数据加载与图表渲染的“时序问题”（确保图表用新数据渲染）；

兼容旧数据：若存储数据格式与当前表格不匹配（如字段缺失），xs.loadData() 会自动忽略异常字段，保证功能可用性。

③ 重置数据：resetData () 函数

```
// 重置数据到初始状态
function resetData() {
    if (confirm('确定要重置数据吗？这将清除所有当前数据并恢复初始示例数据。')) {
        xs.loadData(initialData);
        updateAllViz();
        showStatus('数据已重置到初始状态', 'success');
    }
}
```

加入 confirm() 交互确认，降低误操作风险；
直接复用初始数据 initialData，确保每次重置后数据一致性；

④ 状态提示：showStatus () 函数（支撑数据管理功能）

```
function showStatus(message, type) {
    const statusDiv = document.getElementById('statusMessage');
    statusDiv.textContent = message;
    statusDiv.className = 'status-message';
    statusDiv.classList.add(type === 'success' ? 'status-success' : 'status-error');

    // 3秒后自动隐藏消息
    setTimeout(() => {
        statusDiv.textContent = '';
        statusDiv.className = 'status-message';
    }, 3000);
}

// 页面加载时初始化
document.addEventListener('DOMContentLoaded', function() {
    // 先检查是否有保存的数据
    const savedData = localStorage.getItem('spreadsheetData');

    if (savedData) {
        // 询问用户是否加载保存的数据
        if (confirm('检测到有保存的数据，是否加载？')) {
            // 先初始化表格
            initializeSpreadsheetWithData();
            // 然后加载保存的数据
            setTimeout(loadData, 100);
        } else {
```

```
        } else {
            // 用户选择不加载保存的数据，直接使用初始数据
            initializeSpreadsheetWithData();
        }
    } else {
        // 没有保存的数据，直接使用初始数据
        initializeSpreadsheetWithData();
    }

    // 初始渲染图表（稍后执行，确保表格数据已加载）
    setTimeout(updateAllViz, 300);

    // 绑定复选框事件
    d3.selectAll(".checkbox").on("change", updateAllViz);
;

 按钮事件监听
cument.getElementById('saveData').addEventListener('click', saveData);
cument.getElementById('loadData').addEventListener('click', loadData);
cument.getElementById('resetData').addEventListener('click', resetData);
```

为保存 / 加载 / 重置操作提供即时反馈，让用户明确操作结果；
自动清除机制避免提示信息堆积，保持页面整洁；
颜色区分成功 / 错误状态，绿色 = 安全，红色 = 警告。

4. 实验实现效果：

（1） 顶部标题 “表格与双可视化联动实验”，中间是可编辑表格，下方左侧是 “显示柱状图 / 折线图” 复选框（默认均勾选），右侧是 “保存 / 加载 / 重置” 按钮，最下方是两个并列图表（柱状图 + 折线图）与使用说明；

（2） 图表初始状态：

① 柱状图：按年份分组，每个年份下有 4 个不同颜色的柱子（对应 4 个学科），柱子顶部显示具体数值，右侧图例标注学科名称；

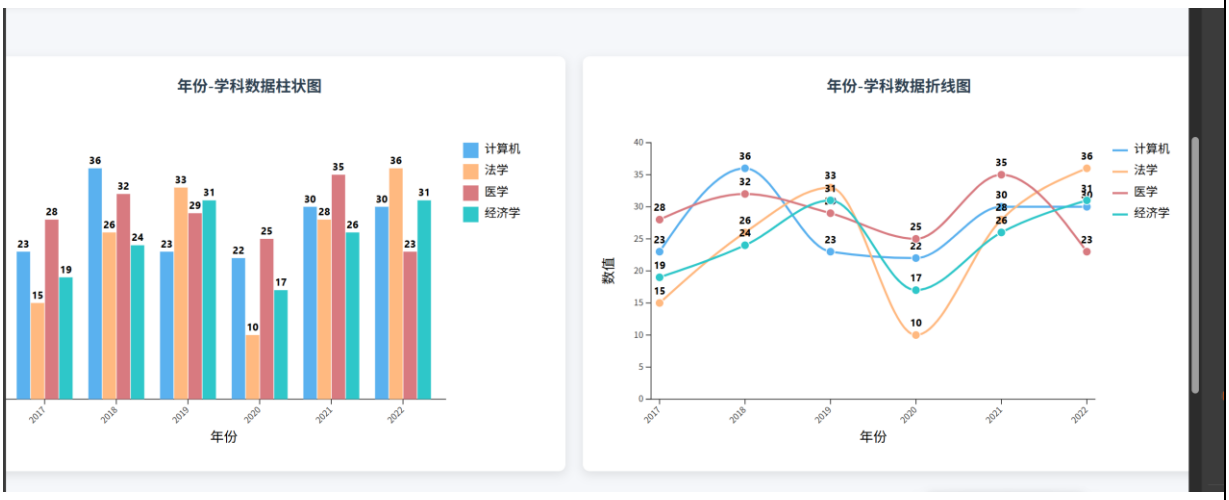
② 折线图：4 条不同颜色的平滑曲线（与柱状图学科颜色一一对应），每个数据点用白色边框的圆点标记，右侧图例用线条区分学科。

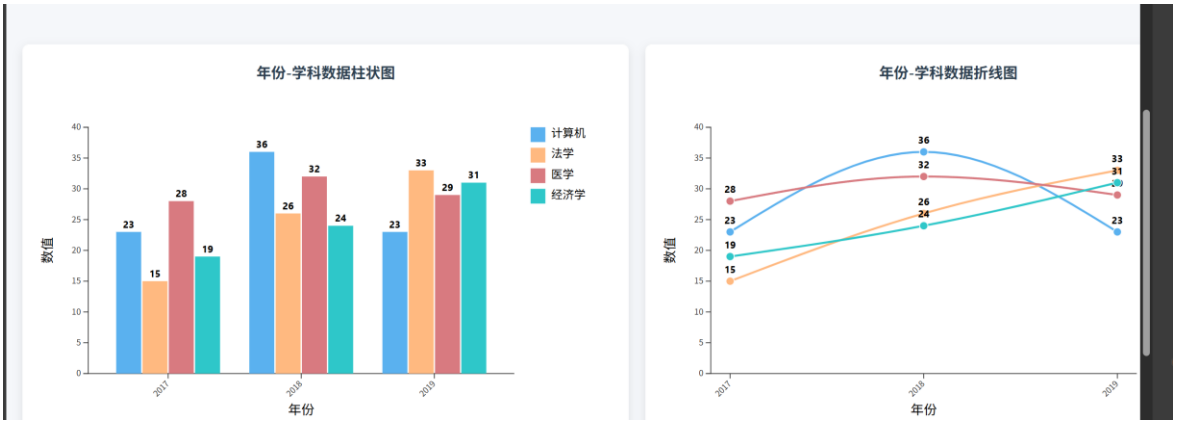
（3） 表格编辑联动效果

① 实时更新：双击表格单元格修改数值，松开鼠标后，柱状图对应的柱子高度立即增加，折线图对应的点与线条也同步上移；

② 数据校验：若输入非数字内容（如 “abc”），会弹出 “表格中存在无效数据，请输入数字！” 的提示，图表不更新，避免错误数据影响可视化效果；

③ 格式适配：若删除某行 / 列数据，图表会自动移除对应年份的柱子 / 线条，保持数据与可视化的一致性

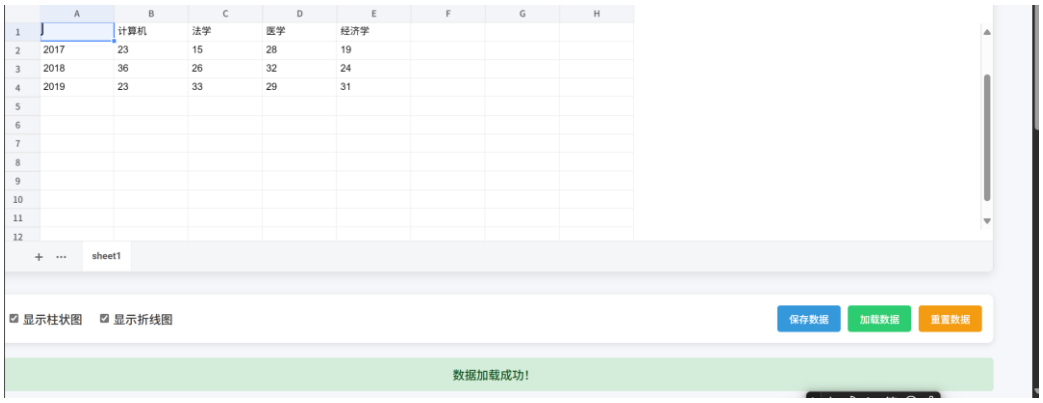




（4） 数据管理功能效果

① 保存数据：点击 “保存数据” 按钮，页面中间弹出绿色提示 “数据保存成功！”，3 秒后消失；此时在浏览器 “开发者工具 - Application-LocalStorage” 中，可看到 “spreadsheetData” 键对应的 JSON 数据；

② 加载数据：修改表格数据后，点击 “加载数据” 按钮，页面弹出绿色提示 “数据加载成功！”，表格与图表立即恢复到上次保存的状态；若未保存过数据，会弹出红色提示 “没有找到保存的数据”；



③ 重置数据：点击 “重置数据” 按钮，弹出确认框，点击 “确定” 后，页面弹出绿色提示 “数据已重置到初始状态”，表格与图表同步恢复为 2017–2021 年的初始数据。