

Rahul Rajendran  
CSCE-679:600  
Assignment #1  
UIN: 132009695

GitHub: <https://github.com/rahul-rajendran3/csce679-HKG-temp-visualization>

You may refer to the hosted project: <https://hkg-temp-visualization.vercel.app/>

### **AI Interaction log:**

Before using Gemini, I wanted to get a basic understanding of d3.js. I referred to official documentation and online tutorials for heatmap creation. By doing so, I was able to gain a better understanding of the various chromatic scales available in d3, such as the one used for this heatmap (<https://d3js.org/d3-scale-chromatic/interpolate#schemeSpectral>).

After getting an initial base version by myself, I used Gemini to incrementally develop features for the final version. Some of these tasks include data aggregation (rollup), color scale bucketing, and mini line chart implementation. Notably, the code provided by Gemini led to an off by one error in the date calculation. This issue then caused the calculation of the max/min temperature of the month to be incorrect. I found the issue was caused by the use of the “default” local date, instead of UTC. I was able to identify this error, allowing Gemini to then provide a corrected version. This shows the capability of LLMs to interpret user instructions and convert natural language into code (HTML/CSS/JS). I believe this has fundamentally changed the way we code.

When Gemini did generate code, I made sure to understand the output and chose not to copy/paste directly.

Interestingly, Gemini would try and “predict” what I would add next (without me explicitly asking) Some of these features include the legend, zooming feature, and tooltip.

The full interaction log is below.

Rahul Rajendran  
CSCE-679:600  
Assignment #1  
UIN: 132009695

I need to create a heatmap for Hong Kong temperature data from 2008-2017, grouped by Month. For a given month/year, the cell should display the maximum temperature. This is what I have so far:  
...



It looks like you've got a solid foundation for your D3.js heatmap! However, there is a core logic issue in how you're handling the daily data.

Right now, your code is trying to draw a rectangle for **every single day** in your CSV (roughly 3,650 days). Since you only have 120 cells (12 months × 10 years), D3 is stacking hundreds of rectangles on top of each other in the same spot. To fix this, you need to **group and aggregate** the data by month and year before binding it to your SVG elements.

#### The Fix: Grouping and Aggregating

You can use `d3.groups` (or `d3.rollup`) to find the maximum temperature for each month/year combination.

Here is the updated script logic:

JavaScript



```
d3.csv("temperature_daily.csv").then(function (data) {  
  
  // 1. Parse and Filter  
  data = data.map(entry => ({  
    date: new Date(entry.date),  
    max_temp: +entry.max_temperature  
  })).filter(entry => entry.date.getFullYear() >= 2008 && entry.date.getFullYear()  
  
  // 2. Aggregate: Group by Year, then by Month, and find the MAX  
  const aggregatedData = d3.rollup(data,  
    v => d3.max(v, d => d.max_temp), // The "Value" (max temp)  
    d => d.date.getFullYear(),        // Key 1 (Year)  
    d => d.date.getMonth()           // Key 2 (Month index 0-11)  
  );  
  
  // Flatten the rollup back into a simple array for D3 to use  
  const finalData = [];  
  aggregatedData.forEach((monthsMap, year) => {
```

```
    aggregatedData.forEach((monthsMap, year) => {  
      monthsMap.forEach((temp, month) => {  
        finalData.push({ year: year, month: month, temp: temp });  
      });  
    });  
  
    const years = Array.from(new Set(finalData.map(d => d.year))).sort();  
    const monthsNames = d3.range(12).map(i => new Date(2008, i).toLocaleString('c'  
  
    // ... Update Scales ...  
    const x = d3.scaleBand().range([0, width]).domain(years).padding(0.05);  
    const y = d3.scaleBand().range([0, height]).domain(monthsNames).padding(0.05);  
  
    // ... Rendering the Squares ...  
    svg.selectAll()  
      .data(finalData)  
      .join("rect")  
      .attr("x", d => x(d.year))  
      .attr("y", d => y(monthsNames[d.month])) // Match the name in the domain  
      .attr("width", x.bandwidth())  
      .attr("height", y.bandwidth())  
      .style("fill", d => myColor(d.temp))  
      .attr("rx", 4)  
      .attr("ry", 4)  
      .style("stroke-width", 1)  
      .style("stroke", "white")  
  });
```

### Key Changes Made:

- **d3.rollup**: This is the magic function. It collapses multiple daily entries into a single "Max" value for that specific month and year.
- **Flattening**: **rollup** returns a nested Map. Converting it to the **finalData** array makes it much easier to bind to your **rect** elements.
- **Padding**: Added **.padding(0.05)** to the **scaleBand**. Heatmaps are much easier to read when there is a tiny gap (stroke or padding) between the "tiles."
- **Coordinate Matching**: Ensure your **y** attribute uses the same string format as your **y.domain**. Since your domain is month names, we use **monthsNames[d.month]**.

#### Pro-Tip: The Color Scale

You are using `d3.interpolateSpectral`. By default, this interpolator puts **red** (hot) at 0 and **blue** (cold) at 1. Since high temperatures are "hot," you might find it more intuitive to reverse it so 40°C is red:

JavaScript



```
const myColor = d3.scaleSequential()
  .interpolator(t => d3.interpolateSpectral(1 - t)) // Reverses the scale
  .domain([0, 40]);
```

Would you like me to help you add a tooltip so you can see the exact temperature when hovering over a cell?

I think there is an off by one error because JS is using default timezone instead of HKG to determine the dates. Can you help me fix this? Only show what I need to change



You're exactly right. If your CSV contains a date like `2015-01-01`, `new Date("2015-01-01")` interprets it as UTC midnight. Depending on your local timezone (e.g., if you are in the US), it might shift back to `2014-12-31 19:00:00`, which ruins your grouping.

To fix this, swap your date parsing and grouping logic to use **UTC methods**.

#### 1. Update the Data Parsing

Replace your `data.map` block with this to ensure the date is parsed as a "wall-clock" UTC time:

JavaScript



```
data = data.map(entry => {
  const d = new Date(entry.date);
  return {
    // This treats the string as UTC to prevent local timezone shifts
```

Rahul Rajendran  
CSCE-679:600  
Assignment #1  
UIN: 132009695

```
    year: d.getUTCFullYear(),  
    month: d.getUTCMonth(),  
    max_temp: +entry.max_temperature  
  };  
  }).filter(d => d.year >= 2008 && d.year <= 2017);
```

## 2. Update the Grouping

Since we already extracted the `year` and `month` as numbers, simplify your `d3.rollup` (or `d3.groups`):

JavaScript



```
const aggregatedData = d3.rollup(data,  
  v => d3.max(v, d => d.max_temp),  
  d => d.year,  
  d => d.month  
);
```

## 3. Adjust the "Add Squares" Logic

Since `d.month` is now an integer (0-11) and `d.year` is a number, update the attributes:

JavaScript



```
svg.selectAll()  
  .data(finalData)  
  .join("rect")  
  .attr("x", d => x(d.year))  
  .attr("y", d => y(months[d.month])) // months[0] = "January"  
  // ... rest of your code ...
```

### Why this fixes it

In JavaScript, `new Date()` is notorious for "correcting" the time to your computer's local clock. By switching to `getUTC...` methods, you are telling the browser to ignore your local timezone and treat the data exactly as it's written in the CSV.



To create a tooltip that matches your example, you need to define a hidden `div` in your HTML/CSS and then add event listeners ( `mouseover` , `mousemove` , `mouseleave` ) to your D3 rectangles.

### 1. Add the CSS

Place this in your `<style>` tag or at the top of your script to ensure the tooltip is positioned correctly and looks like the example.

CSS

```
.tooltip {  
  position: absolute;  
  text-align: center;  
  padding: 8px;  
  font: 12px sans-serif;  
  background: white;  
  border: 1px solid #ccc;  
  border-radius: 4px;  
  pointer-events: none; /* Crucial: stops the tooltip from flickering */  
  box-shadow: 2px 2px 6px rgba(0,0,0,0.1);  
  opacity: 0;  
}
```

### 2. Create the Tooltip Element

Add this line before your `d3.csv` call:

JavaScript

```
const tooltip = d3.select("body")  
  .append("div")  
  .attr("class", "tooltip");
```

### 3. Update the Rectangle Code

Update your `.join("rect")` block with the following event listeners. Note how we format the date to `YYYY-MM` to match your example.

JavaScript

JavaScript



```
svg.selectAll()
  .data(finalData)
  .join("rect")
  .attr("x", d => x(d.year))
  .attr("y", d => y(months[d.month]))
  .attr("width", x.bandwidth())
  .attr("height", y.bandwidth())
  .style("fill", d => myColor(d.temp))
  .style("stroke", "white")
  .on("mouseover", function(event, d) {
    tooltip.style("opacity", 1);
    d3.select(this).style("stroke", "black").style("opacity", 1);
  })
  .on("mousemove", function(event, d) {
    // Format month to 2 digits (e.g., 04)
    const monthFormat = (d.month + 1).toString().padStart(2, '0');

    tooltip
      .html(`Date: ${d.year}-${monthFormat}, max: ${d.temp}`)
      .style("left", (event.pageX + 10) + "px")
      .style("top", (event.pageY - 10) + "px");
  })
  .on("mouseleave", function(event, d) {
    tooltip.style("opacity", 0);
    d3.select(this).style("stroke", "white").style("opacity", 0.8);
  });
```

### Key Logic Points:

- `event.pageX/Y` : This pulls the mouse coordinates to position the tooltip box right next to your cursor.
- `.padStart(2, '0')` : This ensures that April shows as `04` instead of `4`, maintaining the `YYYY-MM` format from your image.
- **Stroke Toggle**: I added a "black" stroke on mouseover; this provides visual feedback so the user knows exactly which cell they are highlighting.

Would you like me to show you how to add a "Download as PNG" button so you can easily export this chart?

Now, I want to add a line chart inside each cell for the monthly temperature (both high and low).

What do I need to add/change?

DO NOT MODIFY THE EXISTING CODE.



To add line charts inside each cell, you need to transition from drawing simple `rect` elements to creating a "container" (a `<g>` element) for each cell. This container will hold both the background color and an SVG line path.

Since you have daily data already loaded, you need to group that daily data so each cell has access to the full array of temperatures for that specific month.

### 1. The Necessary CSS

Add this to ensure the lines are visible against the heatmap colors:

CSS



```
.cell-line {  
  fill: none;  
  stroke: #333; /* Dark grey line for contrast */  
  stroke-width: 1.5px;  
}  
.cell-line-min {  
  fill: none;  
  stroke: #666;  
  stroke-width: 1px;  
  stroke-dasharray: 2,2; /* Dashed line for min temp */  
}
```

### 2. Update the Data Aggregation

You need to change your `rollup` to keep the **array of daily values** instead of just the max/min, so the line generator has points to plot.



JavaScript



```
const aggregatedData = d3.rollup(
  data,
  v => ({
    max_temperature: d3.max(v, d => d.max_temperature),
    min_temperature: d3.min(v, d => d.min_temperature),
    daily_values: v.sort((a, b) => a.date - b.date) // Keep sorted daily data
  }),
  d => d.date.getUTCFullYear(),
  d => d.date.getUTCMonth()
);
```

### 3. Replace the "Add Squares" Section

Instead of `svg.selectAll().data(finalData).join("rect")`, use this logic to create groups (`g`) and draw lines inside them:

JavaScript



```
// 1. Create a group for every cell
const cells = svg.selectAll(".cell-group")
  .data(finalData)
  .join("g")
  .attr("class", "cell-group")
  .attr("transform", d => `translate(${x(d.year)}, ${y(months[d.month])})`);

// 2. Add the background rectangle (the heatmap color)
cells.append("rect")
  .attr("width", x.bandwidth())
  .attr("height", y.bandwidth())
  .style("fill", d => color(d.max_temperature))
  .style("opacity", 0.8)
  .on("mouseover", mouseover)
  .on("mousemove", (event) => { // Update tooltip position on move
    tooltip.style("left", (event.pageX + 10) + "px")
      .style("top", (event.pageY - 10) + "px");
  })
  .on("mouseleave", () => tooltip.style("opacity", 0));

// 3. Define internal scales for the mini-charts
// This maps the 1st-31st of the month to the width of the cell
const xLine = d3.scaleLinear().domain([0, 31]).range([2, x.bandwidth() - 2]);
// This maps 0-40 degrees to the height of the cell
const yLine = d3.scaleLinear().domain([0, 45]).range([y.bandwidth() - 2, 2]);
```

```
// 4. Line generators
const lineMax = d3.line()
  .x((d, i) => xLine(i))
  .y(d => yLine(d.max_temperature));

const lineMin = d3.line()
  .x((d, i) => xLine(i))
  .y(d => yLine(d.min_temperature));

// 5. Add the Max Temp Lines
cells.append("path")
  .attr("class", "cell-line")
  .attr("d", d => lineMax(d.daily_values));

// 6. Add the Min Temp Lines
cells.append("path")
  .attr("class", "cell-line-min")
  .attr("d", d => lineMin(d.daily_values));
```

#### What changed?

- **The Container Strategy:** We now use `join("g")`. This allows us to stack a `rect` (for color) and `path` (for the line) in the exact same coordinate space.
- **Mini-Scales:** `xLine` and `yLine` are "local" scales. They ensure the line chart fits perfectly inside the `bandwidth` of a single heatmap cell.
- **Data Passing:** By nesting `daily_values` inside `finalData`, the line generator knows exactly which points to draw for each specific year/month block.

Would you like me to show you how to add a "Zoom" behavior so you can click a cell to enlarge that specific month's line chart?