EECS E6893 Big Data Analytics
HW3: Data visualization

Tutorial I -- HTML/CSS/JS + D3 Basics

Muyao Zi
mz3056@columbia.edu

Agenda

- How a Web page works
  - ✓ HTML
  - ✓ CSS
  - ✓ JavaScript
- Data visualization: D3.js
- Finance data visualization --- Polygon API

What is a WebPage

A webpage is three layers working together:

- **HTML** (structure/content): what — headings, paragraphs, images, SVGs, containers.
- **CSS** (style/presentation): how it looks — colors, fonts, layout, spacing, responsive rules.
- **JavaScript** (behavior/logic): what it does — fetching data, building charts, reacting to user input.

example file structure for hw part1:

```
part1/                          # Seattle Weather visualizations
  index.html
  styles.css                    # (external CSS)
  app.js                        # (your JS logic)
```

You may submit a single HTML file (with <style> and <script> inside) or separate CSS/JS files. External files are recommended but not required. JavaScript is required for data loading and D3 visualizations; CSS is optional except for the Part I paragraph styling, which may be done in an internal <style> block.

HTML

- HTML is a language for describing web pages.
- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is not a programming language, it is a **markup language**

  A markup language is a set of **markup tags**
- HTML uses **markup tags** to describe web pages

Tutorials for HTML: https://www.w3schools.com/html/html_intro.asp

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>Minimal D3 Page</title>
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link href="styles.css" rel="stylesheet">
 <style>#note{font-size:16px;text-indent:1.5em;color:#333}</style>
 <script src="https://d3js.org/d3.v7.min.js"></script>
 <script defer src="app.js"></script>
</head>
<body>
 <header>
  <h1 class="title">Seattle Weather</h1>
 </header>
 <main>
  <section id="controls">
   <label>Bins <input id="bins" type="range" min="5" max="25" step="5" value="10"></label>
   <label>Variable
    <select id="varSelect">
     <option>precipitation</option>
     <option>temp_max</option>
     <option>temp_min</option>
     <option>wind</option>
    </select>
   </label>
  </section>
```

<!doctype html> tells the browser “modern HTML.” Everything lives inside <html>…</html>

<head> = metadata (title, charset, CSS/JS links, etc)

Character encoding: always set UTF-8 so symbols render correctly

<body> = what actually renders on the page (content, SVGs, etc.)

Elements, tags, attributes

- Element = opening tag, content, closing tag: Hello .
- Void elements (no closing tag): <img>, <meta>, <link>,
, <hr>, <input>.
- Tag: the literal markup tokens like or
- Attributes : a name="value" pair placed inside a start tag to add data, ids, classes, links, ARIA info, etc.
- Global attributes: id, class, style, title, hidden, data-*.

```
<section id="charts">
  <div id="histogram" class="chart"></div>
  <div id="pie" class="chart"></div>
  <div id="line" class="chart"></div>
 </section>

 <section id="summary">
  <h2>Observations</h2>
  <p id="note">Write 2–3 insights here.</p>
 </section>
</main>
<footer>
 <a href="#top" id="backToTop">Back to top</a>
</footer>
</body>
</html>
```

 = tag
id="histogram" = global attribute (unique hook for JS/CSS)
class="chart" = global attribute (style/select as a group)
 = closing tag

Together they form one div element (empty content).

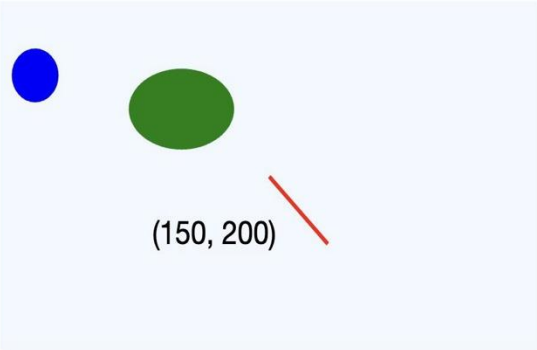
 = tag for a paragraph
id="note" = global attribute
Write 2–3 insights here = content
 closes the element.

What is SVG

- SVG = Scalable Vector Graphics, an XML-based graphics language built into HTML.
- It’s resolution-independent (no blur when zooming) and each shape is a DOM node you can style with CSS and manipulate with JavaScript/D3.
- Coordinate system: origin (0,0) at top-left; +x to the right, +y downward.
- Best for charts, icons, labels, interactive shapes; D3 targets SVG by default.

SVG in HTML

```
<svg width="500" height="300">  <!-- some SVG -->
    <rect x="20" y="20" width="460" height="260" fill="aliceblue"></rect>
    <circle cx="50" cy="75" r="20" fill="blue"></circle>
    <ellipse cx="175" cy="100" rx="45" ry="30" fill="green"></ellipse>
    <text x="150" y="200">(150, 200)</text>
    <line x1="250" y1="150" x2="300" y2="200" stroke="red" stroke-width="3"></line>
</svg>
```




```
<!DOCTYPE html>
<html>
<body>

<svg width="150" height="100" xmlns="http://www.w3.org/2000/svg">
  <rect width="100%" height="100%" fill="green" />
  <circle cx="75" cy="50" r="40" fill="yellow" />
  <text x="75" y="60" font-size="30" text-anchor="middle" fill="red">SVG</text>
</svg>

</body>
</html>
```

More about SVG:https://www.w3schools.com/graphics/svg_intro.asp

Steps to create a HTML file and view in browser

- Step 1: Open a text editor or notepad on your machine. (VSCode)
- Step 2: Enter the lines of code:

myfirstpage.html > html

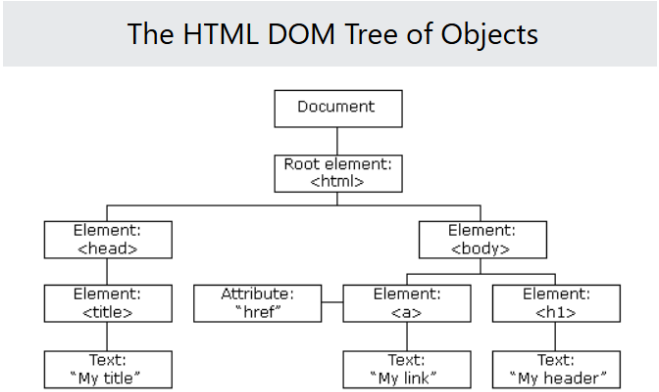
```
1   <!DOCTYPE html>
2   <html>
3
4       <head>
5           <title>Page Title</title>
6       </head>
7
8       <body>
9           <h1>This is a Heading</h1>
10          <p>This is a paragraph.</p>
11      </body>
12  </html>
```

- Step 3: Save the file as myfirstpage.html (go to File-Save As give File name: myfirstpage.html-choose save as type: All Files-click save)
- Step 4: View document in web browser (click on the html file and it will be opened in browser)

What is DOM

What JS/D3 actually changes

- DOM = Document Object Model → the browser's in-memory tree of your HTML page.
- Each HTML tag becomes a node (element node, text node, etc.) in that tree.
- We "hook" nodes with id (unique) and class (reusable) for styling and scripting.
- JavaScript selects nodes, reads/sets attributes/text, and listens to events.
- D3 sits on top of the DOM: it binds data to nodes and updates them when data/UI changes.



About JavaSrcipt

- JavaScript is a programming language
- used by Web browsers to create a dynamic and interactive experience for the user
- Some of the dynamic website enhancements performed by JavaScript are: Loading new content or data onto the page without reloading the page, Rollover effects and dropdown menus etc.
- Some of its most powerful features involve asynchronous interaction with a remote server.

JavaScript adds behavior: fetch data, build charts, handle user interaction.

JavaScript in Web Pages

Three ways to include JS:

- External file (recommended) `<script defer src="app.js"></script>`
- Inline block (OK for tiny demos)

```
<script>
    alert("Hello World!");
</script>
```

- ES Modules (when importing other JS files)

```
<script type="module">
    import { scaleLinear } from 'https://cdn.skypack.dev/d3-scale';
    console.log(scaleLinear);
</script>
```

Loading with D3:

```
<scriptsrc="https://d3js.org/d3.v7.min.js"></script>
<scriptdefersrc="app.js"></script>
```

Your app.js runs after the DOM is ready.

JavaScript Language Basics

- Variables

```
let count = 0;          // reassignable (block scope)
const title = "HW3";    // not reassignable (block scope)
var legacy = 1;         // function scope (avoid)
```

Use const by default, switch to let only when you plan to reassign.

- Types

```
typeof 3        // "number"
typeof "hi"     // "string"
typeof []       // "object"
Array.isArray([]) // true
```

Primitive: number, string, boolean, null, undefined, bigint, symbol
Objects: arrays, plain objects, functions, dates, etc.

- Strings & template literals

```
const name = "D3";
`Hello, ${name}!`    // "Hello, D3!"
"Wind: " + 12.5
```

- Arrays

```
const xs = [3, 1, 4];
xs.push(2);                  // [3,1,4,2]
xs.map(d => d * 2);          // [6,2,8,4]
xs.filter(d => d > 2);       // [3,4]
xs.reduce((a,b)=>a+b, 0);    // 10
for (const x of xs) { /* ... */ }   // iterate values
```

- Objects (records/rows)

```
const row = { date: "2024-01-01", wind: 8.2 };
row.wind           // 8.2
row["date"]        // "2024-01-01"

// Destructuring:
const { date, wind } = row;
```

- Functions

```
function sum(a, b) { return a + b; }
const sum2 = (a, b) => a + b;    // arrow function
```

- Control flow

```
if (x > 5) { ... } else { ... }

for (let i = 0; i < n; i++) { ... }   // classic loop
for (const v of arr) { ... }          // iterate values
```

- Truthy / falsy (common gotcha)

Falsy values: false, 0, "", null, undefined, NaN.

```
if ("") console.log("no");        // not printed
if ("ok") console.log("yes");     // printed
```

JS tutorial:
https://www.w3schools.com/js/js_intro.asp

About CSS

CSS is the language we use to style a Web page.

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

How to use CSS in a page

- External stylesheet (recommended for most styling)

  <link rel="stylesheet" href="styles.css">

- Internal stylesheet (inside <style> in <head>)

  Great for small page-specific rules; required in HW Part I to style your paragraph.

  <style>#observations{font-size:16px;text-indent:1.5em;color:#333}</style>

- Inline style (avoid for maintainability)

  …

Tip: Use external CSS for everything, plus some internal rule to meet the HW requirement.

Example: style.css

```css
/* Base */
:root { --ink:#222; --brand:#2f6fed; --bg:#fafafa; }
html,body{ margin:0; padding:0; font-family:system-ui,Segoe UI,Roboto,Arial,sans-serif; color:var(--ink); background:var(--bg); }
main{ max-width:980px; margin:24px auto; padding:0 16px; }

/* Headings & sections */
h1{ color:var(--brand); line-height:1.2; margin:16px 0 8px; }
h2{ line-height:1.25; margin:20px 0 8px; }
section{ margin:20px 0; }

/* Charts layout (pick one: Grid OR Flex) */
#charts{ display:grid; grid-template-columns:repeat(auto-fit,minmax(320px,1fr)); gap:16px; }
.chart{ background:#fff; border:1px solid #e8e8e8; border-radius:8px; padding:12px; }

/* Controls */
label{ margin-right:16px; font-weight:600; }
input[type="range"]{ vertical-align:middle; }
select{ padding:2px 6px; }

/* SVG & axes */
svg{ width:100%; height:auto; display:block; }
.axis text{ font-size:12px; }
.axis path,.axis line{ stroke:#444; shape-rendering:crispEdges; }

/* Table helper (for Part II) */
table{ border-collapse:collapse; width:100%; background:#fff; }
th,td{ border:1px solid #eee; padding:6px 8px; font-size:14px; text-align:left; }
thead th{ background:#f6f6f6; }
```

How to use CSS in a page

- Hook css style file to HTML

  In HTML head:

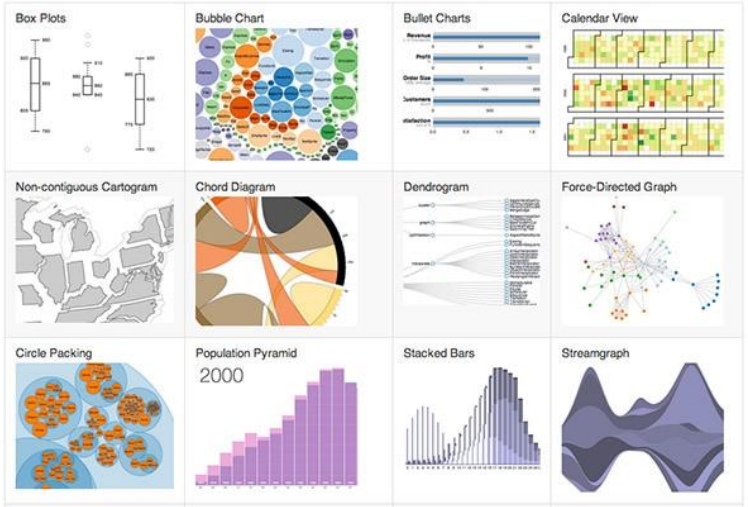
```
<link rel="stylesheet" href="styles.css">
<style>#observations{font-size:16px;text-indent:1.5em;color:#333}</style>
```

CSS tutorial: https://www.w3schools.com/css/css_intro.asp

Introduction to D3.js

Data-Driven Documents

- D3 = Data-Driven Documents.
- D3.js is a JavaScript library for manipulating HTML based on data.
- It lets you bind data to HTML/SVG elements and map numbers to pixels so you can draw charts with code.
- D3 is a toolbox: selections, data joins, scales, axes, shapes, layouts, transitions.



More about D3: https://d3js.org/

Github: https://github.com/d3/d3

Why use D3.js

- Dynamic and Interactive
- Directly binds data to DOM (Document Object Model)
- Extensive flexibility and control over visualization
- Large community and extensive documentation

Note: If your page loads CSV/JSON (e.g., d3.csv(...), fetch(...)), the browser's security model blocks cross-origin reads on file://. Run a local server so your page is served over HTTP.

Two ways to run a server:

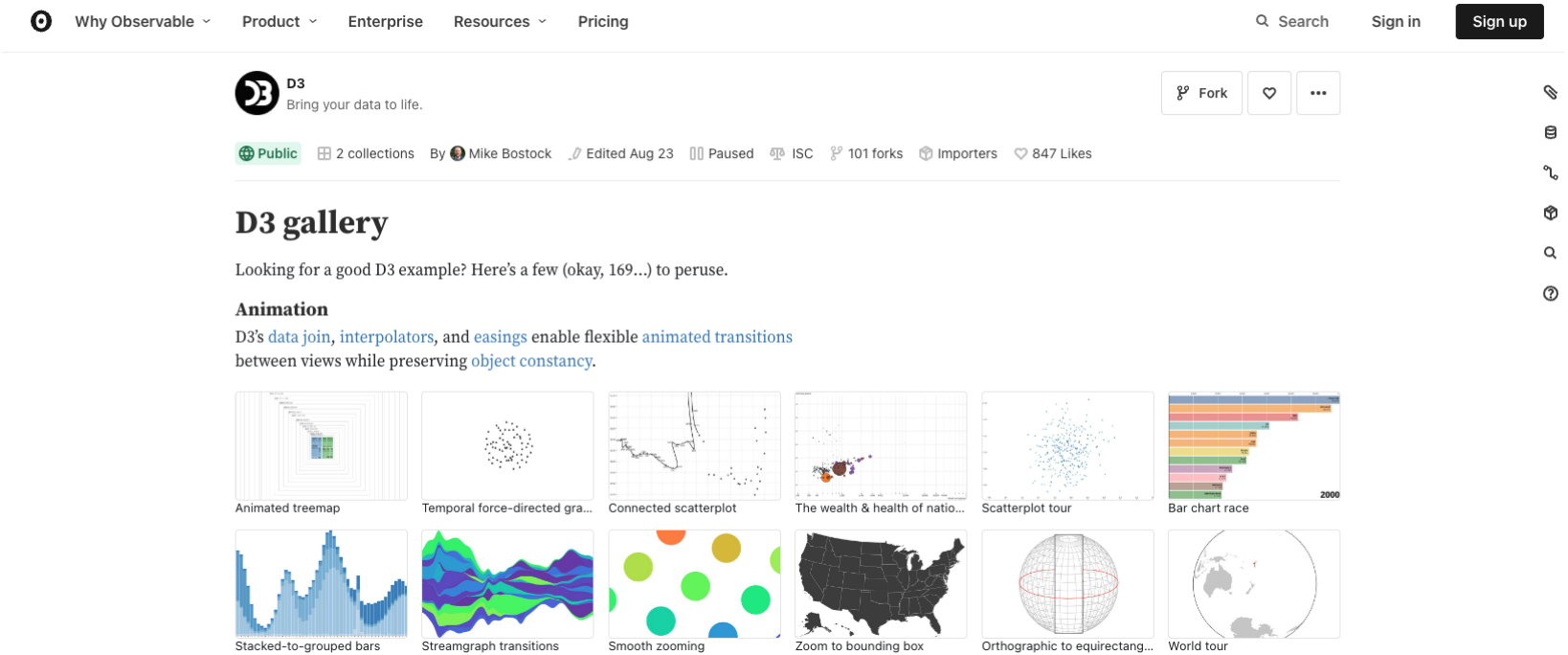
- Option A: Python (built-in)

  ```
  cd path/to/your/project
  python -m http.server 8000
  ```

- then open http://localhost:8000
-
- Option B: VS Code Live Server
- Install the "Live Server" extension, then right-click index.html → "Open with Live Server"

D3 Live Examples

- D3 Examples page: https://observablehq.com/@d3/gallery



Starting D3

- “src” in script tag

d3basic.html > html

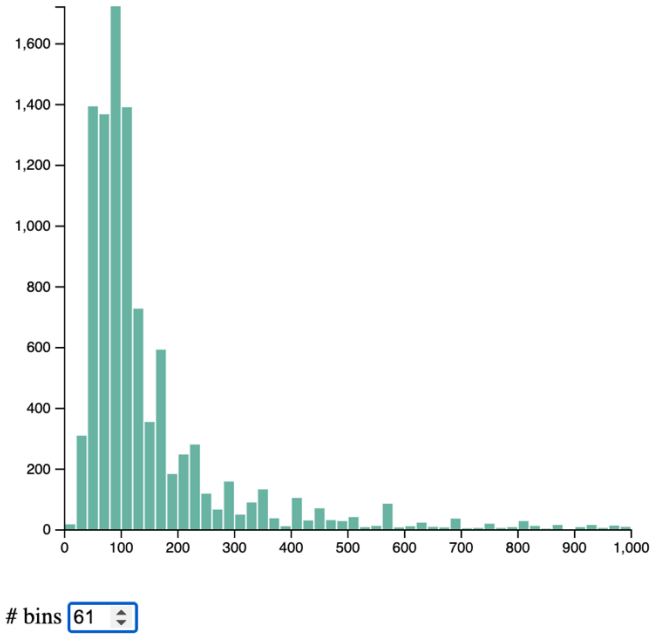
```html
<!DOCTYPE html>
<html>

    <head>
        <script src="https://d3js.org/d3.v4.js"></script>
    </head>

    <body>
        <p>Hello</p>
    </body>
</html>
```

- Load data in D3

```
d3.csv('seattle-weather.csv', d3.autoType).then(data => {
  // numbers & dates parsed for you
  const wind = data.map(d => d.wind).filter(Number.isFinite);
  render(wind, 10);
});
```

D3 examples

(a) Histogram with interactive component



t2 > histogrambins.html > ...

```
<!DOCTYPE html>
<meta charset="utf-8">

<!-- Load d3.js -->
<script src="https://d3js.org/d3.v4.js"></script>

<!-- Create a div where the graph will take place -->
<div id="my_dataviz"></div>

<p>
  <label># bins</label>
  <input type="number" min="1" max="100" step="30" value="20" id="nBin">
</p>


<script>

// set the dimensions and margins of the graph
var margin = {top: 10, right: 30, bottom: 30, left: 40},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#my_dataviz")
  .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 + ")");

// get the data
d3.csv("https://raw.githubusercontent.com/holtzy/data_to_viz/master/Example_dataset/1_OneNum.csv", function(data) {

  // X axis: scale and draw:
  var x = d3.scaleLinear()
      .domain([0, 1000])     // can use this instead of 1000 to have the max of data: d3.max(data, function(d) { return +d.price })
      .range([0, width]);
  svg.append("g")
      .attr("transform", "translate(0," + height + ")")
      .call(d3.axisBottom(x));

  // Y axis: initialization
  var y = d3.scaleLinear()
      .range([height, 0]);
  var yAxis = svg.append("g")

```

```
  // A function that builds the graph for a specific value of bin
  function update(nBin) {
    // set the parameters for the histogram
    var histogram = d3.histogram()
        .value(function(d) { return d.price; })   // I need to give the vector of value
        .domain(x.domain())  // then the domain of the graphic
        .thresholds(x.ticks(nBin)); // then the numbers of bins

    // And apply this function to data to get the bins
    var bins = histogram(data);

    // Y axis: update now that we know the domain
    y.domain([0, d3.max(bins, function(d) { return d.length; })]);   // d3.hist has to be called before the Y axis obviously
    yAxis
        .transition()
        .duration(1000)
        .call(d3.axisLeft(y));
    // Join the rect with the bins data
    var u = svg.selectAll("rect")
        .data(bins)
    // Manage the existing bars and eventually the new ones:
    u
        .enter()
        .append("rect") // Add a new rect for each new elements
        .merge(u) // get the already existing elements as well
        .transition() // and apply changes to all of them
        .duration(1000)
          .attr("x", 1)
          .attr("transform", function(d) { return "translate(" + x(d.x0) + "," + y(d.length) + ")"; })
          .attr("width", function(d) { return x(d.x1) - x(d.x0) -1 ; })
          .attr("height", function(d) { return height - y(d.length); })
          .style("fill", "#69b3a2")
    // If less bar in the new histogram, I delete the ones not in use anymore
    u
        .exit()
        .remove()
    }
    // Initialize with 20 bins
    update(20)

    // Listen to the button -> update if user change it
    d3.select("#nBin").on("input", function() {
      update(+this.value);
    });

});
</script>
```

(b) Table with CSS style

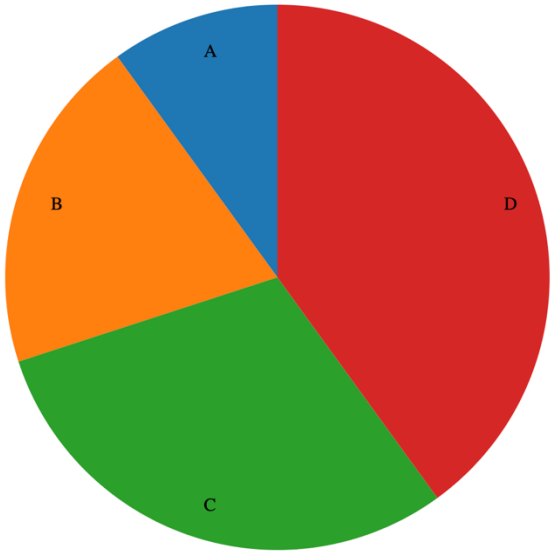
Name	Age	Job
Alice	25	Engineer
Bob	30	Designer
Charlie	28	Teacher

t2 > table.html > html > head > meta

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <script src="https://d3js.org/d3.v5.min.js"></script>
    <style>
        /* Table styles */
        table {
            border-collapse: collapse;
            width: 100%;
        }
        th, td {
            border: 1px solid black;
            padding: 8px 12px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
    </style>
</head>
<body>
    <table></table>
    <script>
        // Sample data
const data = [
    { Name: "Alice", Age: 25, Job: "Engineer" },
    { Name: "Bob", Age: 30, Job: "Designer" },
    { Name: "Charlie", Age: 28, Job: "Teacher" }
];
```

```
// Select the table
const table = d3.select("table");
// Append table headers
const thead = table.append("thead");
thead.append("tr")
    .selectAll("th")
    .data(Object.keys(data[0]))
    .enter().append("th")
    .text(d => d);
// Append table rows and cells
const tbody = table.append("tbody");
tbody.selectAll("tr")
    .data(data)
    .enter().append("tr")
    .selectAll("td")
    .data(d => Object.values(d))
    .enter().append("td")
    .text(d => d);
    </script>
</body>
</html>
```

(c) Pie chart



t2 > pie.html > html > head > script

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <script src="https://d3js.org/d3.v5.min.js"></script>
</head>
<body>
    <svg width="500" height="500"></svg>
    <script>
 // Sample data
const data = [
    { label: "A", value: 10 },
    { label: "B", value: 20 },
    { label: "C", value: 30 },
    { label: "D", value: 40 }
];

// SVG setup
const width = 500;
const height = 500;
const radius = Math.min(width, height) / 2;
const svg = d3.select("svg")
    .append("g")
    .attr("transform", "translate(" + width / 2 + "," + height / 2 + ")");

// Color scale
const color = d3.scaleOrdinal(d3.schemeCategory10);

// Pie layout
const pie = d3.pie().value(d => d.value);
const path = d3.arc().outerRadius(radius - 10).innerRadius(0);
const labelArc = d3.arc().outerRadius(radius - 40).innerRadius(radius - 40);

// Bind data, create pie chart slices
const g = svg.selectAll(".arc")
    .data(pie(data))
    .enter().append("g")
    .attr("class", "arc");

g.append("path")
    .attr("d", path)
    .style("fill", d => color(d.data.label));

g.append("text")
    .attr("transform", d => "translate(" + labelArc.centroid(d) + ")")
    .attr("dy", ".35em")
    .text(d => d.data.label);

    </script>
</body>
</html>
```

Part IV overview

- We will using Polygon API to fetch the stock data we need for Part4.
  You've already seen Polygon API in HW2 tutorial files.

- Problem 1 = live updates (fetch on a timer, append rows).

- Problem 2 = prefetch & render (fetch a small set of dates, save to file, then load that file to build the network).

Part IV Problem1--Live price + difference

For any 3 stocks you choose:

- Fetch the latest trade/price and the previous close.
- Display the difference (current − previous) with green/red up/down.
- Append a new line on every update (don’t overwrite).
- Show the timestamp you received for transparency.

Free API plan tips:

- If same-day ‘latest’ data is blocked or delayed, use the latest available price your plan permits, paired with previous close.
- Always show the timestamp so graders see what you actually got.
- Keep the update interval reasonable to avoid rate limits.

Part IV Problem2--IBM-centered network

Build a correlation network with IBM at the center and edges to AAPL, GOOGL, INTC, MSFT, NVDA.

- Default data window: closing prices on the last trading day of 2022, 2023, 2024 (three dates per ticker).
- Free plan tips: use five quarter-separated dates (e.g., 2022Q4 → 2024Q3). List the exact dates you used.

Example workflow:

- Make a number of API calls, then save responses to a local CSV/JSON.
- In your webpage, load that saved file and render all graphs without more API calls.

Free-plan constraints

If your plan blocks a target date or live endpoint:

- Use the nearest available or quarter-spaced dates as specified.
- Document the dates/times you used.
- Do not hard-code numbers—fetch, save, then load.
- Add a short note in your PDF: ‘Using free plan; endpoints returned delayed/latest-available at timestamp T.’
- Keep your request count low; batch smartly and cache to file.

Homework Submission

Part I (Seattle Weather)

- One working webpage (example: part1/index.html + styles.css + app.js):
- Histogram of wind with 10 bins, then 20 bins (same SVG, not stacked).
- Pie chart of weather categories with % labels.
- Line chart of precipitation vs date.
- Paragraph of observations styled via an internal <style> block.
- Two interactions:

  (a) Slider to change bins (5→25, step 5).

  (b) Dropdown to switch variable (precipitation, temp_max, temp_min, wind) and redraw the histogram with matching labels/title.

Screenshots that show both your code and the rendered plot for each item.

All plots should have titles & x/y labels and readable ticks.

Part II (Auto MPG)

Preprocessing tables (built in JS, client-side):

- Top rows preview.
- Count of cars per model year.
- Total cylinders per model year.
- Count by acceleration.

Three visualizations (using those arrays you build before):

- Pie — distribution by model year (with % labels) next to the table.
- Line — total cylinders per year (sorted x-axis).
- Histogram — acceleration (bins = 10).

Screenshots of each table + its output plot + code.

All preprocessing happens in JavaScript (no Python/Excel).

Each plot uses one SVG (don’t duplicate charts when updating).

Clear titles, axes, legends where relevant.

References

- https://www.w3schools.com/js/js_htmldom.asp
- Vitaly Shmatikov CS 345 Introduction to JavaScript
- Sarbjit Kaur Introduction to HTML
- https://d3js.org/
- https://developer.mozilla.org/en-US/docs/web/SVG
- https://livebook.manning.com/book/d3js-in-action-second-edit ion/chapter-7/70
- https://d3-graph-gallery.com/index.html
- https://www.w3schools.com/graphics/svg_intro.asp
- https://www.w3schools.com/html/html_intro.asp
- https://www.w3schools.com/js/js_intro.asp
- https://www.w3schools.com/css/css_intro.asp
- https://www.w3schools.com/ai/ai_d3js.asp

Suggestion for this week:

- Work on Part I and Part II locally

Next tutorial:

- Graph and Network Analysis using D3.js
- Hosting webpages on Apache Webserver

THANK YOU !!