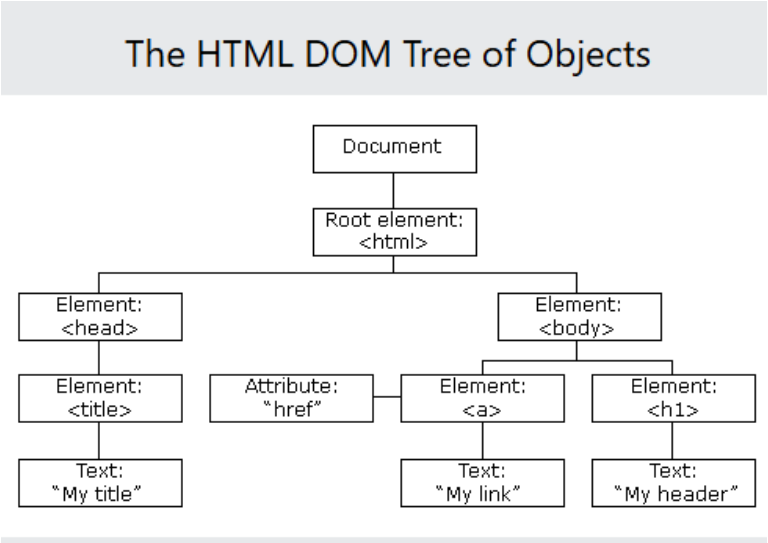
ITEC 5207 Assignment HW04: D3 + SQL

In this assignment you will work with **D3.js** or **Data-Driven Document.**

D3 is a **JavaScript library** that allows developers and designers to create interactive, dynamic, and data-linked visualization**s** in the web browser. At its core, D3 connects **data** with **visual elements** (HTML, SVG, or Canvas). It doesn't *draw charts for you* like Chart.js or Tableau — instead, it gives you **the low-level tools to build any chart from scratch**.

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page. The **HTML DOM** model is constructed as a tree of **Objects**:



Source: [What is HTML DOM]

D3 lets you bind data to the DOM and then transform the document based on that data.

D3 is not…

- Not a "chart library" (like Chart.js or Plotly).
- Not only for static visuals — it supports animation, transitions, filtering, zooming.
- Not limited to one visual style — you can make custom, expressive visualizations.

What Makes D3 Powerful

- Direct access to the web stack: uses HTML, SVG, CSS, and JavaScript — no plugin.
- Declarative transformations: you describe how visuals should respond to data changes.
- Integrates naturally with data tools: you can fetch JSON, CSV, or query results from SQL / APIs directly.
- Scales and Layouts: D3 helps you convert data values into visual properties (size, position, color).

- Community and examples: massive ecosystem of tutorials and reusable snippets (Observable, bl.ocks.org).

Conceptual Flow

1. **Select** elements on a web page (e.g., all circles in an SVG).
2. **Bind data** to those elements (e.g., an array of numbers).
3. **Create / update / remove** elements to match your data.
4. **Use scales & axes** to map data values to screen positions, colors, or sizes.
5. **Add interactivity** (hover, click, filter) to make the visualization dynamic.

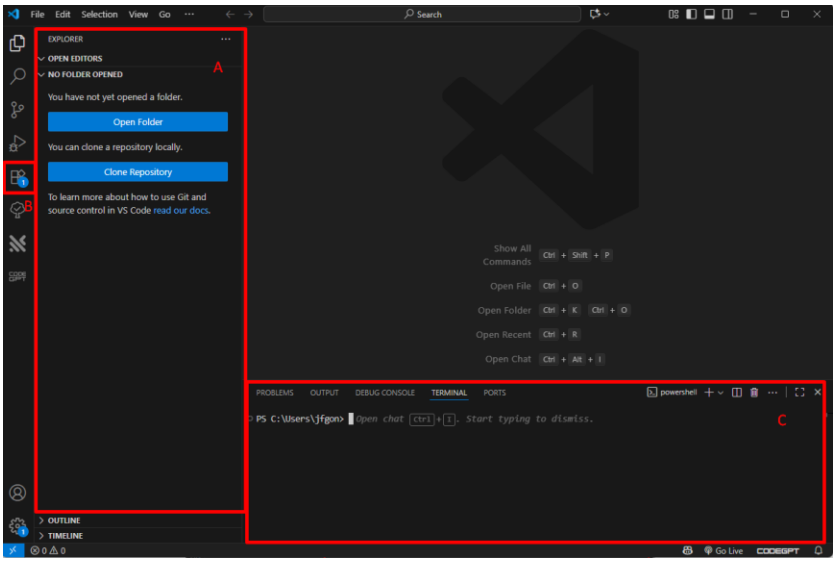
This pattern is called **data join** — and it's what makes D3 "data-driven."

Get Started

Goal: Set up your local environment to create interactive data visualizations using D3.js

Step 1 — Install Visual Studio Code (VS Code)

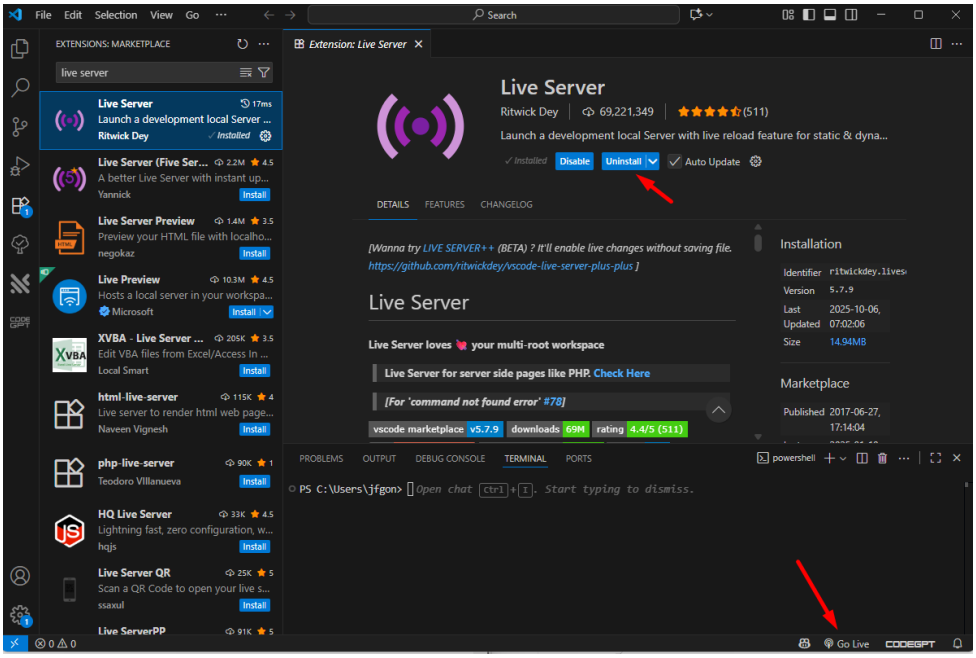
1. Go to https://code.visualstudio.com/
2. Download and install the version for your operating system (Windows, macOS, or Linux).
3. Open VS Code and familiarize yourself with:
    a. The **Explorer** (left sidebar): shows your project files.
    b. The **Terminal** (View → Terminal): to run commands.
    c. The **Extensions tab**: for adding functionality.



Step 2 — Install the "Live Server" Extension

Why: HTML files opened directly from disk (file://) cannot load local data (like CSVs). Live Server creates a local web server that enables D3 to fetch data correctly.

1. In VS Code, go to the **Extensions** tab (left sidebar).
2. Search for **"Live Server"** by *Ritwick Dey*.
3. Click **Install**.
4. Once installed, open any index.html file → click **"Go Live"** at the bottom-right corner.
5. This launches a local server at a URL like http://127.0.0.1:5500/.



Step 3 — Create Your Project Folder
Inside a folder (e.g. d3-tutorial), create the following structure:

```
d3-tutorial/
│
├── index.html
├── script.js
└── data.csv
```

Step 4 — Prepare Your HTML File

This is the entry point of your visualization. It creates a simple web page, it adds the d3 library, it creates a SVG tag – which will includes our visualization – , and it adds the script.js that we wil modify to create our visualization

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My First D3 Visualization</title>
  <script src="https://d3js.org/d3.v7.min.js"></script>
</head>
<body>
  <h1>Hello D3!</h1>
  <svg width="600" height="400"></svg>
  <script src="script.js"></script>
</body>
</html>
```

Step 5 — Add a Simple Dataset

data.csv
name,value
A,10
B,25
C,40

Step 6 — Write Your First D3 Script

```
// Select the SVG element
const svg = d3.select("svg");

// Load data from CSV file
d3.csv("data.csv").then(data => {

  // Convert 'value' from text to number
  data.forEach(d => {
    d.value = +d.value;
  });

  console.log("Data loaded:", data);

  // Create a circle for each row of data
  svg.selectAll("circle")
     .data(data)
     .enter()
     .append("circle")
     .attr("cx", (d, i) => 80 + i * 100) // x-position
     .attr("cy", 100)                    // y-position
     .attr("r", d => d.value / 3)        // radius scaled by 'value'
     .attr("fill", "steelblue");

  // Optional: add labels
  svg.selectAll("text")
     .data(data)
     .enter()
     .append("text")
     .attr("x", (d, i) => 80 + i * 100)
     .attr("y", 180)
     .attr("text-anchor", "middle")
     .text(d => d.name);
});
```

Step 7 — Run Your Project

1. Open index.html in VS Code.
2. Click **"Go Live"** to launch the local server.
3. Your browser will open at http://127.0.0.1:5500/.
4. Open the browser console (F12 → Console tab) to see Data loaded: messages.

Congratulations — you now have a running D3 environment!

Step 8 — Verify data connection

Change the data.csv by editing or adding new values and verify, by refreshing the website, that your output updates coherently.

This single example encapsulates D3's entire philosophy:

Data → Elements → Attributes → Visual Mapping

Assignment

Submit: You will submit a ZIP file containing:

- Your code.
- A PDF file answering the questions numbered in the assignment.

You will be working with real data creating interactive visualization. Use the attached CSV that contains information related to internet penetration information worldwide. Replace the data.csv in your previous exercise with the provided one. Keep the structure.

Binding data

We will start by creating a simple line chart to compare the number of internet users across the continents.

D3 allow you to manipulate graphical elements in the DOM by binding data to them. We will start by defining basic elements in the script.js

```
1
2
3   // Select the SVG and set margins
4   const svg = d3.select("svg"); // Reference to the existing SVG element in the index.html
5   const margin = {top: 40, right: 80, bottom: 40, left: 80}; // SVG Margins
6   const width = +svg.attr("width") - margin.left - margin.right; // Available width for the chart
7   const height = +svg.attr("height") - margin.top - margin.bottom; // Available height for the chart
8
9
10  // Create a group for the chart. This will contain all the chart elements
11  const chart = svg.append("g")
12    .attr("transform", `translate(${margin.left},${margin.top})`);
13
```

Then we load data and transform it from its original format

```
// Create a group for the chart. This will contain all the chart elements
const chart = svg.append("g")
  .attr("transform", `translate(${margin.left},${margin.top})`);

// Load CSV data
d3.csv("data.csv").then(data => {

  // Parse numbers
  data.forEach(d => {
    d.Year = +d.Year;
    d.NumberofInternetUsers = +d.NumberofInternetUsers;
  });

```

Data contains information about all countries in the world. We are going to focus only on the continents' information, making the chart more readable. We filter the information. We also create groups and start creating the scales. A scale in D3 helps you to map the abstract data you have (domain) into coordinates in your chart (range) based on the available space. Thus, categorical data is mapped into positions in the X axis, for instance, and quantitative data is mapped into positions in the chart coherently.

```
  // Filter only continents (we can check manually or hardcode them)
  const continents = ["Asia", "Europe", "Africa", "North America", "South America", "Oceania"];
  data = data.filter(d => continents.includes(d.Entity));

  // Nest data by continent
  const nested = d3.groups(data, d => d.Entity);

  // Create scales. In this case years are numeric and the scale is linear
  // If years were strings, we would use scalePoint or scaleBand
  // For the y-axis, we use a linear scale
  const x = d3.scaleLinear()
    .domain(d3.extent(data, d => d.Year))
    .range([0, width]);

  const y = d3.scaleLinear()
    .domain([0, d3.max(data, d => d.NumberofInternetUsers)]).nice()
    .range([height, 0]);

```

We create a linear function that will also map given a set of point X and Y, positions in the chart to create the paths. We also define a categorical scale, in this case to assign a unique color (range) to each continent (domain).

```
  // Define line generator. This function will create the "d" attribute for path elements
  // based on the data points. For x we use the year, for y the number of internet users
  // The line generator will be called for each continent's data
  const line = d3.line()
    .x(d => x(d.Year))
    .y(d => y(d.NumberofInternetUsers));

  // Color scale for each continent. We use a categorical color scheme for categorical data
  // d3.schemeTableau10 is a good choice for up to 10 categories
  const color = d3.scaleOrdinal()
    .domain(continents)
    .range(d3.schemeTableau10);

```

We place now the axes in the chart.

```
// Add axes to the chart
chart.append("g")
  .attr("transform", `translate(0,${height})`) // Move to the bottom
  .call(d3.axisBottom(x).tickFormat(d3.format("d"))); // Format ticks as integers

// Y axis on the left
chart.append("g")
  .call(d3.axisLeft(y));

```

Finally, we create specific elements and bind them to the data. We are going to create the lines based on the data we have. Each data point will give information related to the continent, year and number of users. This information is used to create the lines and give the color to the line.

```
// Draw one line per continent. This is where data is bound to the path elements
chart.selectAll(".line")
  .data(nested) // Specifically bind nested data, where each element is [continent, data[]]
  .enter() // Create a new path for each continent
  .append("path")
  .attr("class", "line") // In this case we are adding lines
  .attr("d", d => line(d[1])) // d is the current group, use the line generator on the continent's data (d[1])
  .attr("fill", "none") // No fill for lines
  .attr("stroke-width", 2) // Line width
  .attr("stroke", d => color(d[0])); // Color by continent name (d[0])

```

To make it clear, we are also creating a label for each continent at the end of the line.

```
// Add labels at the end of each line. We are adding a new set of elements also bound to the nested data
chart.selectAll(".label")
  .data(nested) // Bind nested data again
  .enter()
  .append("text") // In this case text elements for labels
  .attr("class", "label")
  .datum(d => ({ // Use datum to bind a single object instead of an array
    name: d[0],
    value: d[1][d[1].length - 1]  // last point, we need this to know the x position -- last year
  }))
  .attr("x", d => x(d.value.Year) + 5)  // Position slightly to the right of the last data point
  .attr("y", d => y(d.value.NumberofInternetUsers)) // Align with the last data point
  .text(d => d.name)
  .attr("fill", d => color(d.name))
  .style("font-size", "10px");
});

```

Refresh your website and look at the results.

1. Analyze the chart.
    a. What marks and channels are used?
    b. Wha tasks can be done with the viz?
    c. Is it a good viz or not?

Details on demand

We are going to allow the users to get information on demand of specific points in the chart.

First, we need to modify the style and add elements in the index.html. Final result should look like this

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Internet Users by Continent</title>
  <script src="https://d3js.org/d3.v7.min.js"></script>
  <style>
    body { font-family: sans-serif; }
    .line { fill: none; stroke-width: 2px; }
    .label { font-size: 10px; }
    .dot { stroke: white; stroke-width: 0.5px; }
    .tooltip {
      position: absolute;
      background: rgba(0,0,0,0.7);
      color: white;
      padding: 4px 8px;
      border-radius: 4px;
      font-size: 12px;
      pointer-events: none;
    }
  </style>
</head>
<body>
  <h2>Number of Internet Users by Continent (1990-2020)</h2>
  <svg width="800" height="500"></svg>
  <div class="tooltip" style="opacity:0;"></div>
  <script src="script.js"></script>
</body>
</html>

```

Now, after creating the lines but before creating the labels, we are going to create dots for each point in the chart. We are going to add interactivity by allowing a tooltip to show up describing the information of that specific point when hovering over it.

```
// We are going to creeate dots for each point in each continent
// Combine all points into one array
const allPoints = nested.flatMap(d => d[1]);
//Reference to the tooltip div
const tooltip = d3.select(".tooltip");
chart.selectAll(".dot")
  .data(allPoints) // Bind all points
  .enter()
  .append("circle") // Create a circle for each point
  .attr("class", "dot")
  .attr("cx", d => x(d.Year)) // x position based on year
  .attr("cy", d => y(d.NumberofInternetUsers)) // y position based on number of internet users
  .attr("r", 3) // Radius of the circle
  .attr("fill", d => color(d.Entity)) // Color by continent
   // Tooltip interaction
  .on("mouseover", (event, d) => { // event is the mouse event, d is the data point.
      tooltip.transition().duration(100).style("opacity", 1); // Make tooltip visible
      tooltip.html(`
        <strong>${d.Entity}</strong><br>
        Year: ${d.Year}<br>
        Users: ${d3.format(",")(d.NumberofInternetUsers)}
      `)
      .style("left", (event.pageX + 10) + "px") // Position tooltip
      .style("top", (event.pageY - 28) + "px");
    })
  .on("mousemove", (event) => { // Update position on mouse move
      tooltip.style("left", (event.pageX + 10) + "px")
             .style("top", (event.pageY - 28) + "px");
    })
  .on("mouseout", () => { // Hide tooltip
      tooltip.transition().duration(200).style("opacity", 0);
    });

```

Creating a map!

We are going to use the data to link it with a map. There is a style.css file in this assigment, put it in the same folder.

We are going to create a separate script for the map, to keep the code organized rename your script.js file to linechart.js. Now, create a new file and name it map.js. Thus, you should have in your folder the following files

d3-tutorial/
│
├── index.html
├── style.css
├── data.csv
├── linechart.js
└── map.js

We are going to create the SVG for the new viz and link the style file. Modify the index to look like:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Internet Users: Map & Line Chart</title>
  <script src="https://d3js.org/d3.v7.min.js"></script>
  <link rel="stylesheet" href="style.css">

</head>
<body>
  <h2>Global Internet Users: Map & Trend by Continent</h2>

  <div class="container">
    <div class="viz">
      <h3 style="text-align:center">Map of Internet Users (Most Recent Year)</h3>
      <svg id="map" width="600" height="400"></svg>
    </div>

    <div class="viz">
      <h3 style="text-align:center"> Internet Users by Continent (1990–2020)</h3>
      <svg id="linechart" width="600" height="400"></svg>
    </div>
  </div>

  <div class="tooltip" id="tooltip" style="opacity:0;"></div>

  <!-- Separate scripts -->
  <script src="map.js"></script>
  <script src="linechart.js"></script>
</body>
</html>

```

Now, let's move to create the map. Open you map.js file. We first start by referencing the SVG and margins. We are using a function to crete a geospatial projection and a path element that will draw the map

```javascript
const svgMap = d3.select("#map");
const widthMap = +svgMap.attr("width");
const heightMap = +svgMap.attr("height");
const tooltip = d3.select(".tooltip");

// Define map projection
const projection = d3.geoNaturalEarth1()
  .scale(150)
  .translate([widthMap / 2, heightMap / 1.8]);

const path = d3.geoPath(projection);
```

We are going to use an online resource to get geometric information about the different countries so we can draw them. We are going to load it and the data into variables. We then clear the information with the last year available

```
// Load both GeoJSON (countries) and CSV (data)
// D3 needs to load geometric and tabular data separately
// We use Promise.all to wait for both to load before proceeding
Promise.all([
  d3.json("https://raw.githubusercontent.com/datasets/geo-countries/master/data/countries.geojson"),
  d3.csv("data.csv")
]).then(([geoData, csvData]) => {

  // Clean and summarize data: get latest year per country
  const latestByCountry = d3.rollup(
    csvData.filter(d => d.Code),  // ignore regions/continents without country code
    v => v.reduce((a, b) => a.Year > b.Year ? a : b), // latest record
    d => d.Code
  );

```

Geospatial data uses a code, which is also included in the data to link the data information with the geospatial information. We create the map of code + value to display – number of internet users. We now create the scale to represent the number of users with colors.

```
  // Create a map from country code → number of users
  const usersByCode = new Map(
    Array.from(latestByCountry, ([code, d]) => [code, +d.NumberofInternetUsers])
  );

  // Compute colorMap scale
  const colorMap = d3.scaleSequentialSqrt()
    .domain(d3.extent(usersByCode.values()))
    .interpolator(d3.interpolateBlues);

```

Finally, we draw the chart

```
  // Draw countries
  svgMap.selectAll("path")
    .data(geoData.features) // Bind GeoJSON information
    .enter()
    .append("path")
    .attr("d", path)
    .attr("class", "country")
    .attr("fill", d => { // We use the country code to get the number of users from our data
      const val = usersByCode.get(d.properties['ISO3166-1-Alpha-3']); // ISO 3166-1 alpha-3 code represent the code of the country
      return val ? colorMap(val) : "■#eee";
    })
    // Tooltip -- same behavior as in line chart
    .on("mouseover", (event, d) => {
      const val = usersByCode.get(d.properties['ISO3166-1-Alpha-3']);
      tooltip.transition().duration(100).style("opacity", 1);
      tooltip.html(`
        <strong>${d.properties['name']}</strong><br>
        Internet users: ${val ? d3.format(",")(val) : "No data"}
      `)
      .style("left", (event.pageX + 10) + "px")
      .style("top", (event.pageY - 28) + "px");
    })
    .on("mousemove", (event) => {
      tooltip.style("left", (event.pageX + 10) + "px")
             .style("top", (event.pageY - 28) + "px");
    })
    .on("mouseout", () => {
      tooltip.transition().duration(200).style("opacity", 0);
    });

});
```

2. Add the result in your PDF
3. Explain your experience working with D3, how was it, what is the most challenging part?
4. Critique the visualizations.