Data Visualization: Advanced

Seungmin Jin
(dryjins@gmail.com)

Goals

Course objectives: Understanding Visual Analytics Development

1. Review the exercise
2. Understand canvas api with virtual dom interaction
3. Understand GeoJson for Map Viz

Submission Guide

1. Create Github repo
   a. GitHub Basics Made Easy: A Fast Beginner's Tutorial!
2. Deploy the work as the github-page.
   a. How to Use GitHub Pages in 2025! (Beginner's Guide)
3. Share the link of code and demo

Examples

ohdoyoel/unist_cse468: UNIST CSE468 Information Visualization Exercise Code

Chocolate Sales Visualization

DataTable

HTML Tables

Chocolate Sales Data Table

Search across all columns...

Sales Person ↓	Country ↑↓	Product ↑↓	Date ↑↓	Amount ↑↓	Boxes Shipped ↑↓
All	All	All	Filter...	Filter...	Filter...
Wilone O'Kielt	Australia	Manuka Honey Choco	2022. 5. 11.	$4284.00	94
Wilone O'Kielt	Australia	Caramel Stuffed Bars	2022. 4. 14.	$2030.00	11
Wilone O'Kielt	Australia	Organic Choco Syrup	2022. 7. 7.	$1743.00	111
Wilone O'Kielt	Australia	Drinking Coco	2022. 4. 6.	$623.00	283
Wilone O'Kielt	USA	After Nines	2022. 4. 25.	$392.00	30
Wilone O'Kielt	New Zealand	85% Dark Bars	2022. 8. 2.	$1827.00	117
Wilone O'Kielt	Australia	After Nines	2022. 5. 27.	$3325.00	26
Wilone O'Kielt	UK	Drinking Coco	2022. 5. 18.	$3388.00	55
Wilone O'Kielt	New Zealand	Mint Chip Choco	2022. 8. 19.	$11662.00	242
Wilone O'Kielt	Australia	Fruit & Nut Bars	2022. 6. 15.	$392.00	102

Showing 1 to 10 of 1094 entries

Previous Next

Code: datavis/data-table at main · dryjins/datavis

Demo: Chocolate Sales Data Table

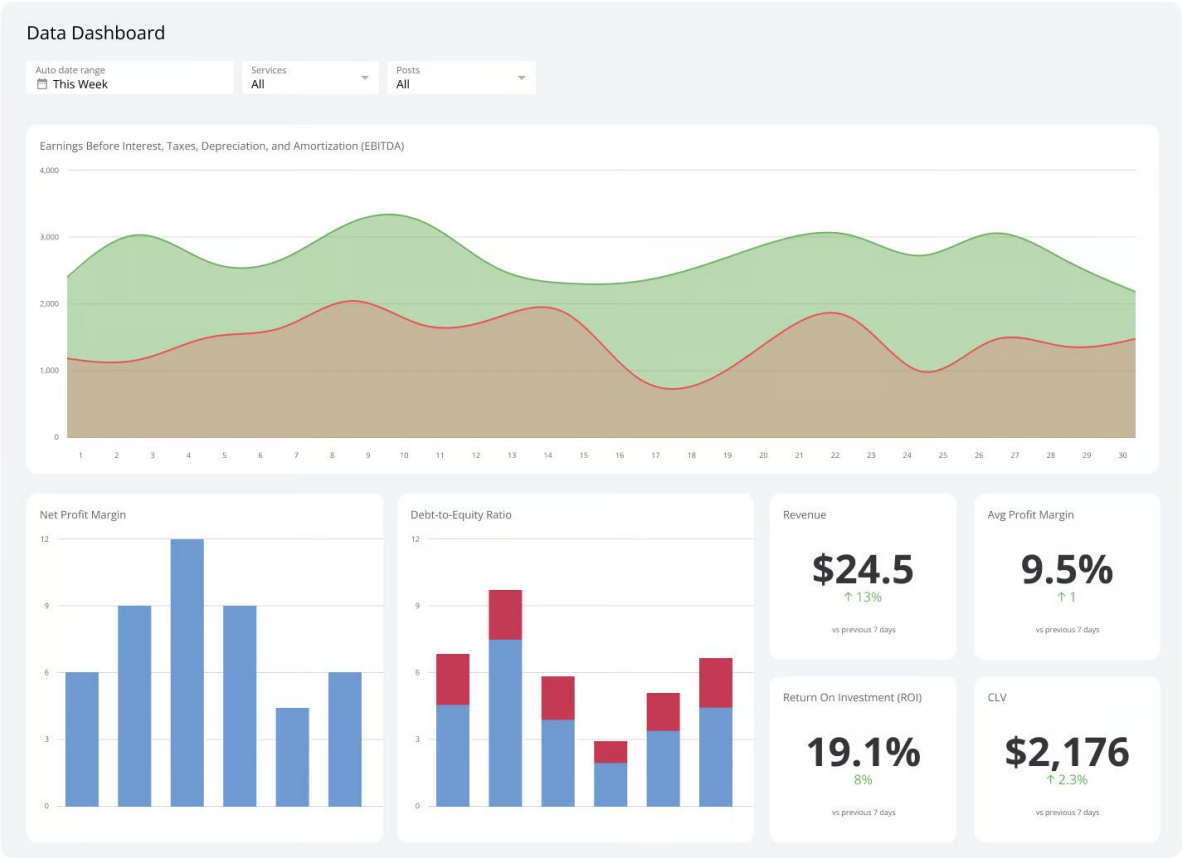
Layout - Dashboard (Grid)

Key Characteristics:

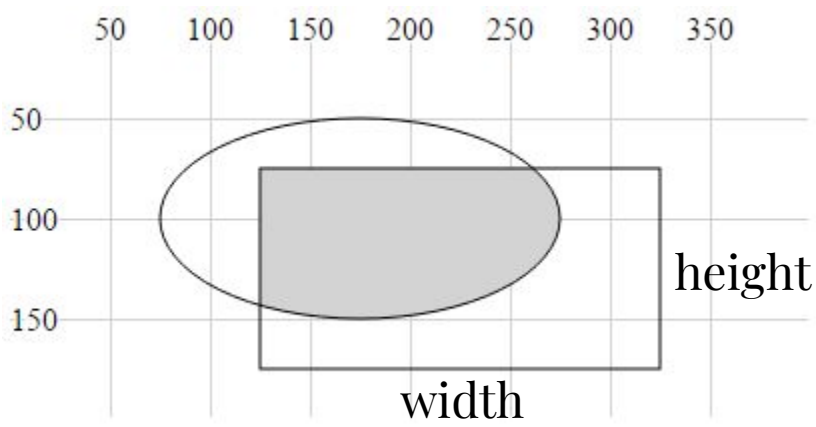
- Utilizes CSS Grid and Flexbox for organizing visualization elements
- Supports responsive design to adapt to various screen sizes
- Maintains consistent spacing and alignment between components

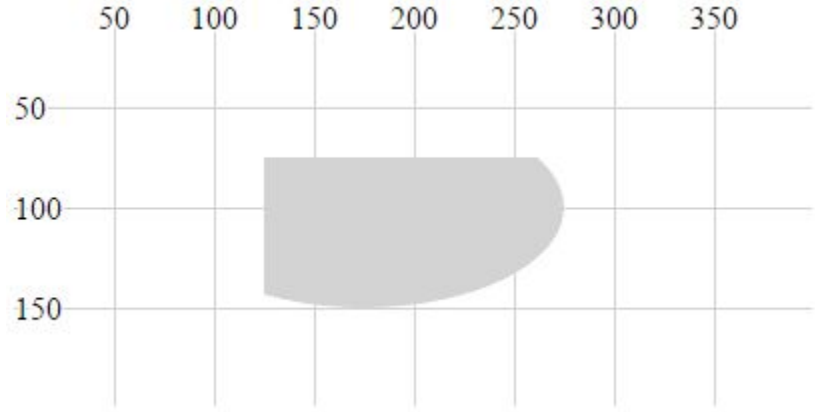
Benefits:

- Clear visual structure with balanced presentation
- Predictable interface that users can quickly understand
- Efficient use of screen real estate with defined proportions



Clipping






```javascript
svg.append("defs").append("clipPath")
    .attr("id", "clip")
    .append("rect")
    .attr("width", width)
    .attr("height", height);
```

```javascript
.attr("clip-path", "url(#clip)")
```

Example: [Chocolate-Sales Dashboard]

Ex 2

Upgrade your vis to dashboard

https://docs.google.com/document/d/1bW5BopFKGp4Nw2tAGDdcI9QvGc8xIW9sZitZkXKmHNw/edit?usp=sharing

Repo list

https://docs.google.com/spreadsheets/d/1C491T4Du438Q5GAzYKty_L2mL3h34DZ_lWrmTwWYZ6U/edit?usp=sharing

Issue of SVG

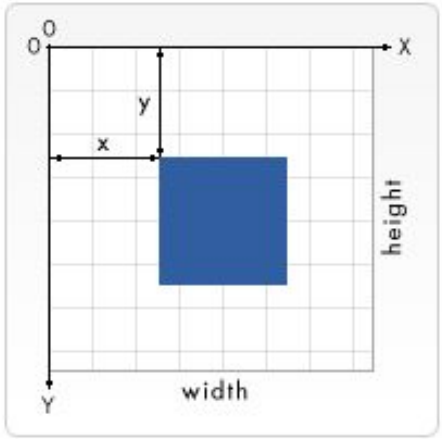
We can visualize anything with HTML and CSS, so HTML provides...

Rendering One Million Datapoints with D3 and WebGL

Scalable Vector Graphics (SVG - DOM)

- SVG defines the graphics in XML format
- Every **element** and every attribute in SVG files can be animated
- SVG requires **incremental space complexity** for each elements (3000 max)

HTML Canvas

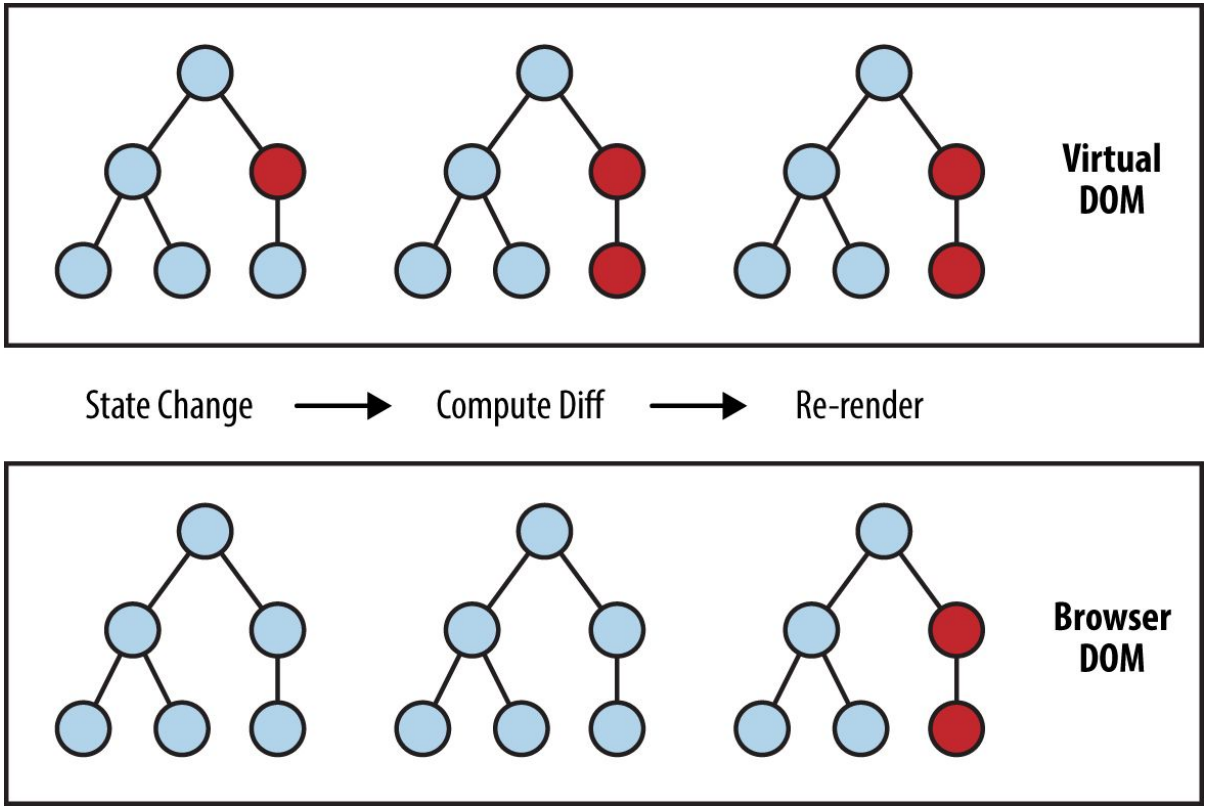


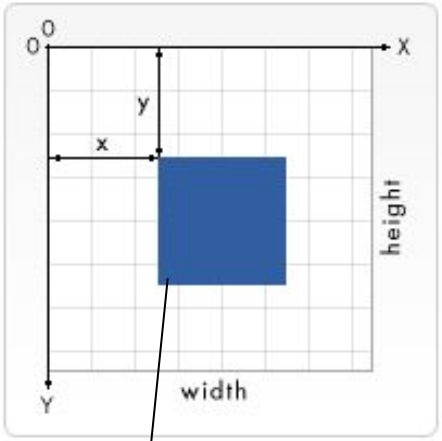
CPU → Canvas 2D

GPU → WebGL

- A mere container for graphics.
- You must use JavaScript to **actually draw** the graphics.
- Canvas require **fixed space complexity** based on the height and width.

Virtual DOM

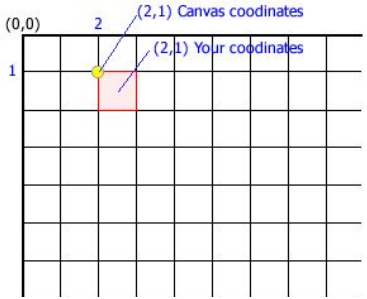




How to interact?
: Creating references in the hidden dimension

Virtual DOM- Hitscan

Pixel



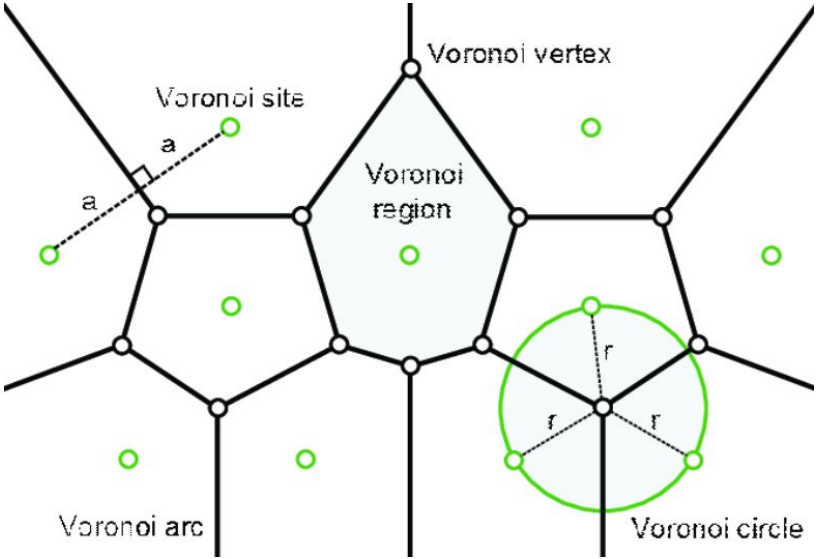
Hitscan

First Name	Last Name	Bats	Throws	G	AB	R	RBI
				Games as Batter		H	2B
David	Aardsma	R	R	254 68	3	0 0	0 0
Hank	Aaron	R	R	3298 3298	12364	2174 3771	2297 624
Tommie	Aaron	R	R	437 437	944	102 216	94 42
Don	Aase	R	R	448 81	5	0 0	0 0
Andy	Abad	L	L	15 15	21	1 2	0 0
Fernando	Abad	L	L	22 22	1	0 0	0 0

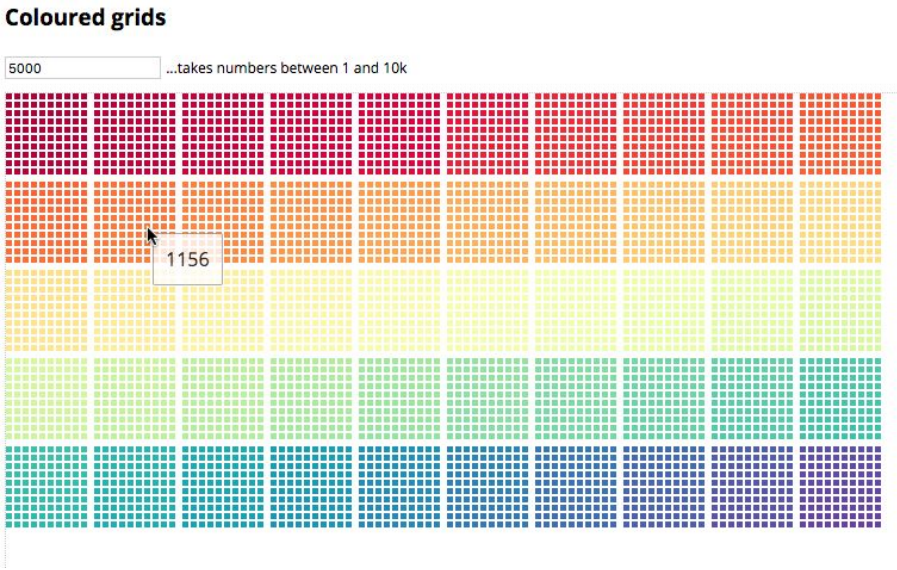
Data

Virtual DOM- Example

A Voronoi diagram
The boundaries between regions form where distances to two or more seeds are equal.



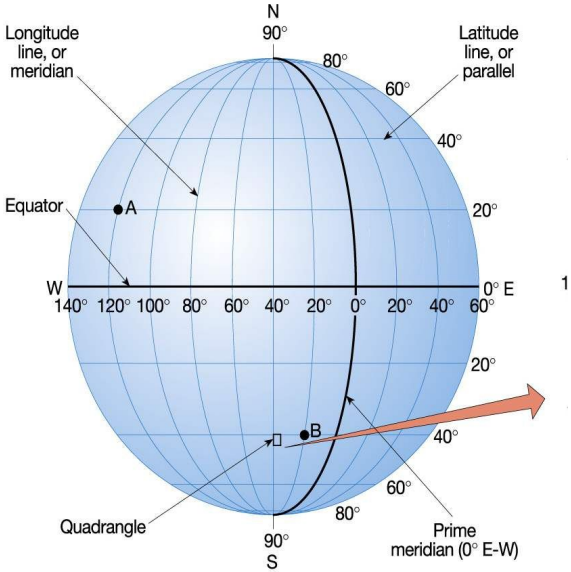
Voronoi Diagram:[Interaction in d3 Canvas based Scatter Plots / Sumant Pattanaik | Observable]



Pixel Viz: [D3 and Canvas in 3 steps]

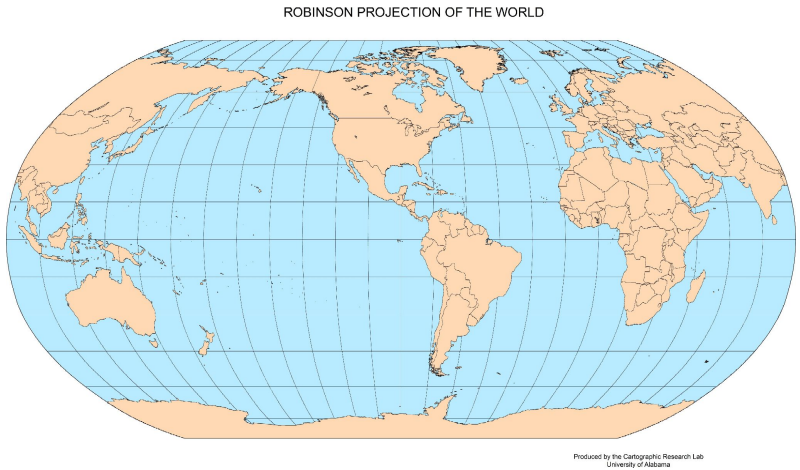
D3-geo: Projection

3D, Latitude, Longitude



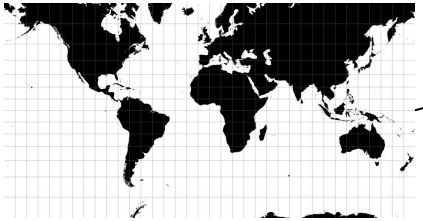
Projection

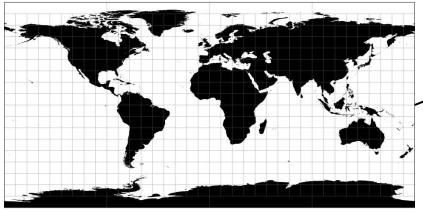
2D, X, Y

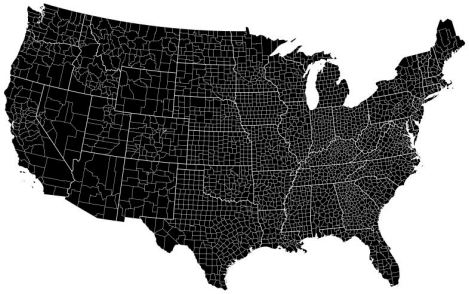


The **transformation** of **spherical coordinates** (latitude and longitude) onto a **two-dimensional plane**, allowing the representation of a three-dimensional Earth on a flat surface.

D3-geo: Projection







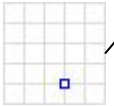
- **Mercator Projection**: The Mercator projection is one of **the most well-known projections** and is often used for world maps. It preserves angles and straight lines but distorts the size of objects as they move away from the equator.
- **Equirectangular Projection**: Also known as the Plate Carrée projection, it simply maps the latitude and longitude values directly onto the Cartesian coordinates, resulting in a rectangular map. This projection distorts shapes and areas, particularly near the poles.
- **Orthographic Projection**: The orthographic projection displays the Earth as if viewed from a distant point in space. It is commonly used for **globe-like visualizations**, where the Earth is represented as a sphere.
- **Albers Projection**: The Albers projection is a conic projection that balances shape and area distortions. It is often used for regional maps, such as country-level or state-level maps.
- **Conic Equidistant Projection**: This projection preserves distances along a set of standard parallels, making it suitable for mapping specific regions or countries.

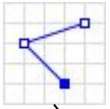
D3-geo: geoJson

Sample of GeoJson

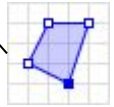
```
{
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [0, 0],
        [10, 0],
        [10, 10],
        [0, 10],
        [0, 0]
      ]
    ]
  },
  "properties": {
    "name": "Example Polygon",
    "color": "blue"
  }
}
```

{ "type": "**Point**", "coordinates": [30.0, 10.0] }





{ "type": "**LineString**", "coordinates": [[30.0, 10.0], [10.0, 30.0], [40.0, 40.0]] }



- "**type**": "Feature": Indicates that the GeoJSON object is a feature.
  - "FeatureCollection" allows array of "Feature"
- "**geometry**": Defines the geometry of the feature.
  - "type": "Polygon": Specifies that the geometry type is a polygon.
  - "coordinates": Contains an array of coordinate values representing the vertices of the polygon. In this case, we have a square defined by a set of five coordinates.
- "**properties**": Contains additional information associated with the feature.
  - "name": "Example Polygon": Specifies a name or label for the polygon.
  - "color": "blue": Represents an additional property like the color of the polygon.

D3-geo

D3-geo is a module in D3.js for working with geographic data and creating map visualizations. It provides a set of powerful tools and projections for handling geospatial data.

1. Projection
   a. d3.geoProjection(): Creates a new projection function. projection([coordinates]): Converts geographic coordinates to screen coordinates.
2. Path Generation
   a. d3.geoPath(): Creates a new path generator for rendering GeoJSON. path([feature]): Generates an SVG path string for a given GeoJSON feature.
3. GeoJSON
   a. d3.geoJSON(): Converts GeoJSON data into a GeoJSON feature collection. d3.geoPath().projection(projection)(geojson): Generates SVG paths for a GeoJSON feature collection.
4. Map Projections
   a. Various projection methods like d3.geoMercator(), d3.geoAlbers(), d3.geoNaturalEarth1(), etc. Projection methods define how the spherical Earth is projected onto a flat surface.
5. Interactivity
   a. Mouse events like mouseover, mouseout, click, etc., can be attached to map elements. Use d3.select() to select and manipulate map elements based on interactions.

D3-geo

1. d3.geoPath(): This function creates a new path generator that converts GeoJSON geometries into SVG path strings. It is specifically designed to work with geographic data.
2. .projection(projection): The .projection() method is used to set the projection for the path generator. The projection variable refers to a projection function that defines how geographic coordinates should be transformed into screen coordinates.
    a. In the example, projection is defined using d3.geoMercator() function, which sets up a Mercator projection. You can use different projection methods like d3.geoAlbers(), d3.geoNaturalEarth1(), or create custom projections depending on your requirements.

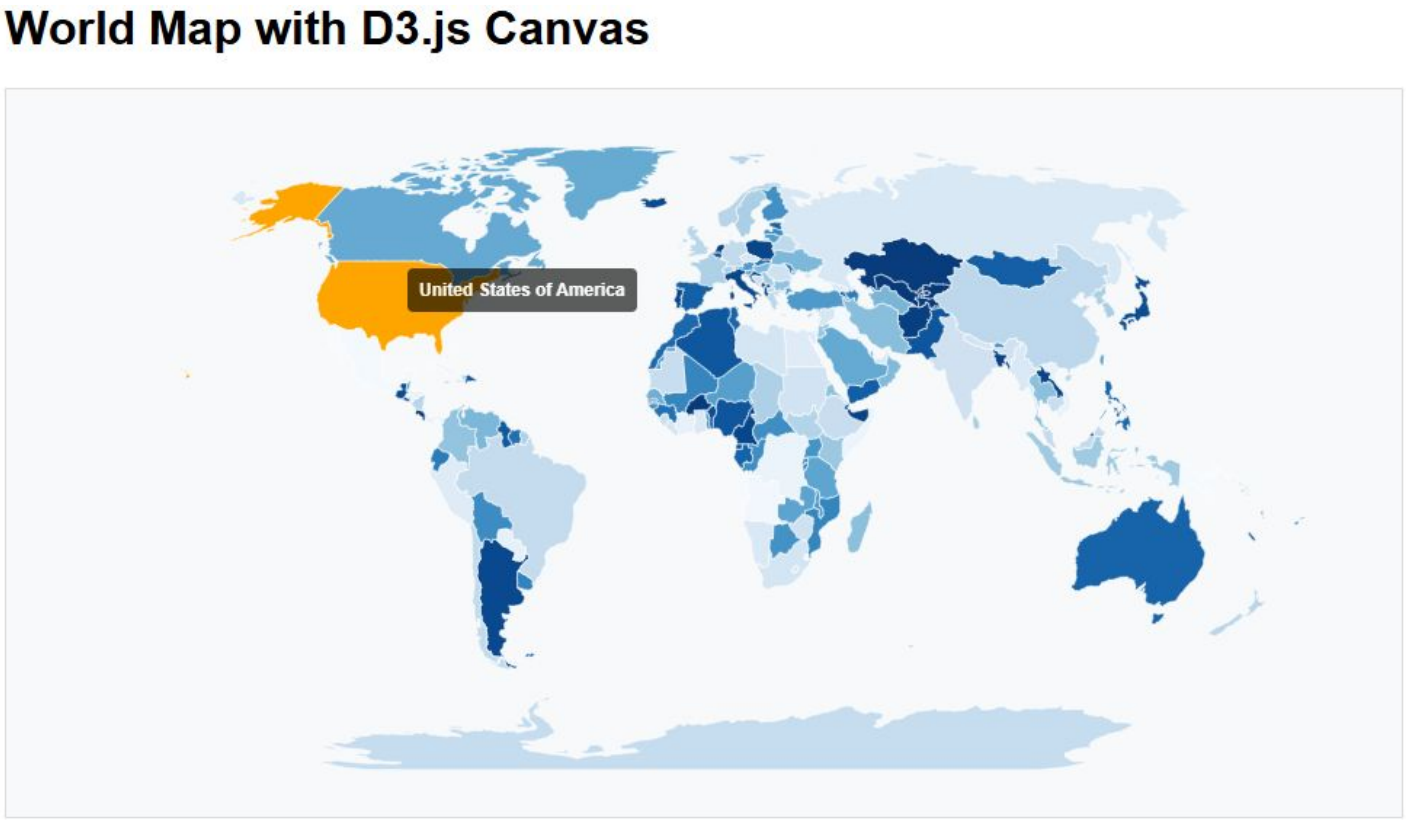
d3-geo-test (codepen.io)

```
// Create the path generator
const path = d3.geoPath().projection(projection);

// Sample GeoJSON data (replace this with your actual data)
const geojson = {
  type: "FeatureCollection",
  features: [
    {
      type: "Feature",
      properties: {},
      geometry: {
        type: "Polygon",
        coordinates: [
          [
            [-10, -10],
            [10, -10],
            [0, 10],
            [-10, -10]
          ]
        ]
      }
    }
  ]
};

// Generate SVG paths for the features
svg.selectAll("path")
  .data(geojson.features)
  .enter()
  .append("path")
  .attr("d", path);
```

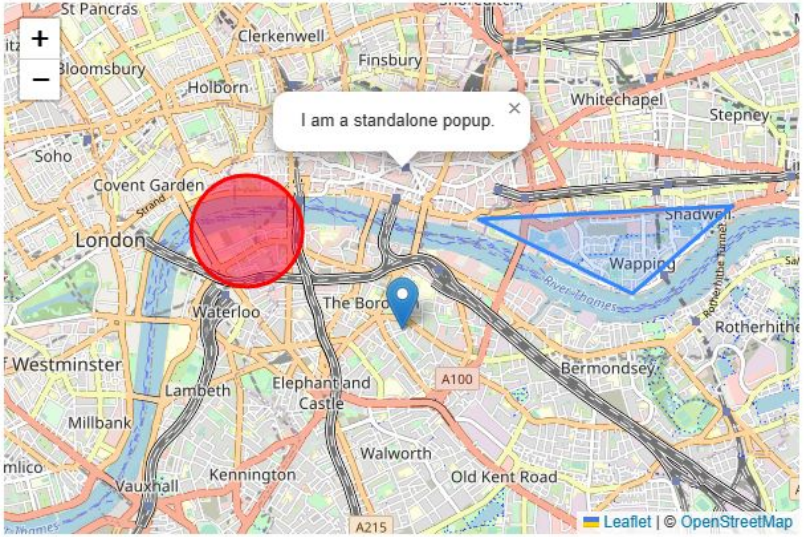
Using Canvas for Map



D3.js Canvas Map

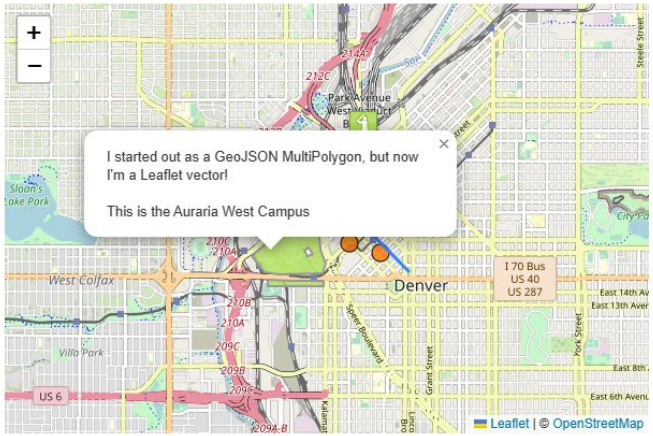
dryjins/datavis

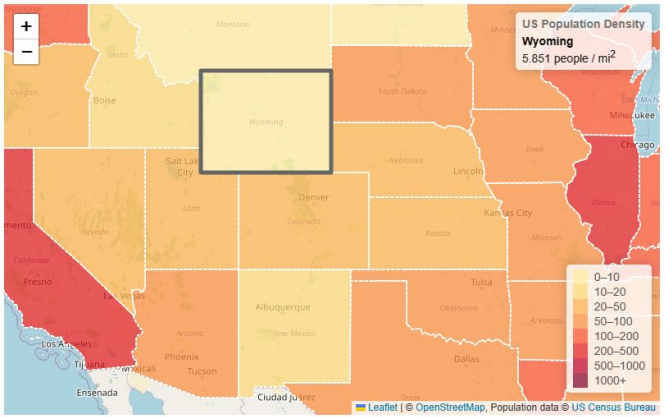
Leaflet



1. Getting Started
   a. Leaflet is easy to get started with. Include the Leaflet library in your HTML file using a CDN or by downloading the library.
   b. Create a element in your HTML where the map will be displayed, and give it an id or class.
2. Basic Map Initialization
   a. In your JavaScript code, create a new Leaflet map instance by calling the L.map() function and passing the ID or class of the element.
   b. Set the initial map center and zoom level using the setView() method.
   c. Add a tile layer to provide the base map using L.tileLayer() and specify the tile source.
3. Markers and Popups
   a. Markers are used to add points of interest or locations to the map. Create markers using L.marker() and specify the coordinates.
   b. Customize markers with icons, colors, and popups to display additional information or interactivity.

Leaflet





1. Layers and Layer Controls
   a. Leaflet supports various types of layers, including tile layers, GeoJSON layers, image overlays, and more.
   b. Use layer controls (L.control.layers) to allow users to toggle different layers on and off.
2. Interactivity and Events
   a. Leaflet provides event handling capabilities for interacting with map elements.
   b. Attach event listeners to map objects, such as markers or polygons, to respond to user interactions like clicks or hovers.
3. Plugins and Extensions
   a. Leaflet has a rich ecosystem of plugins and extensions that enhance its functionality.
   b. Explore available plugins for adding features like clustering, heatmaps, geocoding, routing, and more.

Leaflet + d3js

- Integration allows combining geospatial data with advanced visualizations. Leaflet handles mapping, zooming, and basic interactivity.
- D3 enables data-driven visualizations and SVG rendering.
- Use cases: overlaying charts on maps, linking map interactions with visualizations.
- Workflow: load data with Leaflet, process with D3, overlay visualizations.
- Benefits: comprehensive and interactive map visualizations.
- Consider performance for large datasets and complex visualizations.

Data-Driven Documents

Why??

Leaflet

an open-source JavaScript library
for mobile-friendly interactive maps

Overview Tutorials Docs Download Plugins Blog

May 18, 2023 — Leaflet 1.9.4 has been released!

Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. Weighing just about 42 KB of JS, it has all the mapping features most developers ever need.

Leaflet is designed with *simplicity*, *performance* and *usability* in mind. It works efficiently across all major desktop and mobile platforms, can be extended with lots of plugins, has a beautiful, easy to use and well-documented API and a simple, readable source code that is a joy to contribute to.