# VDA

## Librería para Visualización de Datos

García Aguilar Luis Alberto

# Contenido

# Introducción

# Introducción y objetivos

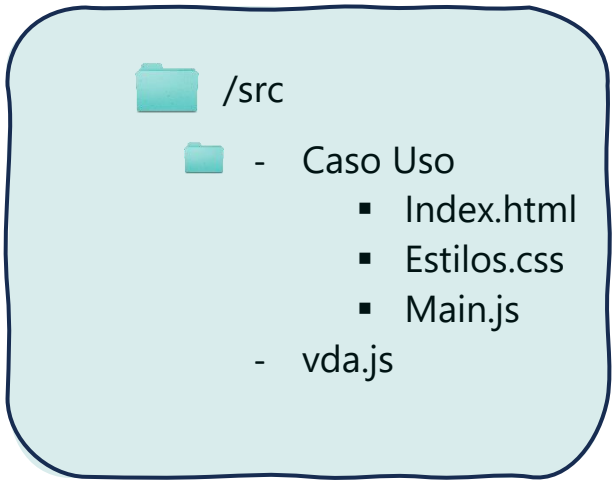
Este documento pretende ser una guía para el uso de nuestra biblioteca de visualización de datos en JavaScript, Vda. Esta guía está diseñada para proporcionar una referencia completa a los métodos disponibles en la biblioteca, facilitando a los desarrolladores la creación de gráficos y visualizaciones interactivas en un entorno web utilizando elementos de canvas.

Se pretende ofrecer una descripción de cada método proporcionado en la biblioteca, incluyendo su propósito, entradas, salidas y casos de uso. La biblioteca tiene como objetivo facilitar al desarrollador el uso de canvas para poder crear una variedad de gráficos y visualizaciones, tales como gráficos de líneas, gráficos de barras, gráficos de burbujas, entre otros.

# Configuración inicial

# Configuración de la biblioteca

Esta guía considera la siguiente estructura para cada caso de uso:




Para hacer uso de la biblioteca y sus métodos es necesario importarla dentro del proyecto:

Por ejemplo, en tu archivo main.js agregar la siguiente línea de código al inicio del archivo:

```
import {initViewport, drawXAxis} from './lib/vda.js';
```

Entre corchetes se indican los métodos a utilizar y entre comillas se indica la ruta del archivo vda.js

También puedes importar la librería completa de la siguiente manera:

```
import * as vda from './vda.js';
```

# Métodos

# View Port

## Uso

El método initViewport se encarga de inicializar un espacio de trabajo en un elemento canvas. Este método ajusta el tamaño Uso del canvas y configura las coordenadas para trabajar con una matriz inversa, facilitando la representación gráfica de datos en el canvas.

## Entradas

- **canvasId** (string): El ID del canvas en el documento HTML.
- **width** (number): Ancho para el canvas.
- **height** (number): Altura para el canvas.

## Salidas

**context** (CanvasRenderingContext2D): Contexto del canvas, configurado para trabajar con el viewport ajustado.

## Método

```
initViewport (canvasId, width,
height) {

    …

    …

    return context;

 }
```

# Eje X

## Uso

El método drawXAxis dibuja una línea de eje con marcas (ticks) distribuidas uniformemente a lo largo del eje. Las marcas se generan para un rango de valores especificado por el usuario (start, end) y se espacian según un paso dado (step).

## Entradas

- **Context** (CanvasRenderingContext2D): Contexto del canvas donde se dibujará.
- **startX** (number): Valor inicial del rango del eje X.
- **endX** (number): Valor de fin del rango del eje X.
- **xPos** (number): Posición X en el canvas donde se comenzará a dibujar el eje X.
- **yPos** (number): Posición Y en el canvas donde se comenzará a dibujar el eje X.
- **step:** Paso entre cada marca (tick) en el eje X.
- **color** (string): Color.
- **labelSpace** (number): Espacio entre etiquetas y el eje X.

## Salidas

No retorna valor. Dibujará directamente el eje X.

## Método { }

```
drawXAxis (context, startX, endX, xPos, yPos, step, color, labelSpace)
{

    ...

 }
```

# Eje Y

## Uso

El método drawYAxis dibuja una línea de eje Y con marcas (ticks) distribuidas uniformemente a lo largo del eje. Las marcas se generan para un rango de valores especificado por el usuario (start, end) y se espacian según un paso dado (step).

## Entradas

- **Context** (CanvasRenderingContext2D): Contexto del canvas donde se dibujará.
- **startX** (number): Valor inicial del rango del eje Y.
- **endX** (number): Valor de fin del rango del eje Y.
- **xPos** (number): Posición X en el canvas donde se comenzará a dibujar el eje Y.
- **yPos** (number): Posición Y en el canvas donde se comenzará a dibujar el eje Y.
- **step:** Paso entre cada marca (tick) en el eje Y.
- **color** (string): Color.
- **labelSpace** (number): Espacio entre etiquetas y el eje Y.

## Salidas

No retorna valor. Dibujará directamente el eje Y.

## Método { }

```
DrawYAxis (context, startY, endY,
xPos, yPos, step, color, labelSpace)
{

    ...

 }
```

# Eje X a partir de un arreglo de datos

## Uso

El método drawXAxis dibuja una línea de eje X en un canvas a partir de valores definidos en un arreglo proporcionado.

## Entradas

- **Context** (CanvasRenderingContext2D): Contexto del canvas.
- **xValues** (Array[Object]): Array de datos 1xn que contiene los valores para las marcas del eje.
- **xPos** (number): Posición X en el canvas donde se comenzará a dibujar el eje.
- **yPos** (number): Posición Y en el canvas donde se comenzará a dibujar el eje.
- **color** (string): Color.
- **labelSpace** (number): Espacio entre etiquetas y el propio eje.
- **canvasPadding** (number): Padding del canvas.

## Salidas

No retorna valor. Dibujará directamente el eje X.

## Método { }

```
drawXAxisFromArray (context,
xValues, xPos, yPos, color,
labelSpace, canvasPadding) {

    ...

}
```

# Eje Y a partir de un arreglo de datos

## Uso

El método drawYAxis dibuja una línea de eje Y en un canvas a partir de valores definidos en un arreglo proprcionado.

## Entradas

- **Context** (CanvasRenderingContext2D): Contexto del canvas.
- **yValues** (Array[Object]): Array de datos 1xn que contiene los valores para las marcas del eje.
- **xPos** (number): Posición X en el canvas donde se comenzará a dibujar el eje.
- **yPos** (number): Posición Y en el canvas donde se comenzará a dibujar el eje.
- **color** (string): Color.
- **labelSpace** (number): Espacio entre etiquetas y el propio eje.
- **canvasPadding** (number): Padding del canvas.

## Salidas

No retorna valor. Dibujará directamente el eje Y.

## Método { }

```
drawYAxisFromArray (context,
yValues, xPos, yPos, color,
labelSpace, canvasPadding) {

    ...

}
```

# Gráfica de puntos

## Uso

El método drawDotPlot dibuja una gráfica de puntos basada en un conjunto de datos proporcionados mediante un archivo csv. Cada punto de datos se representa como un círculo en el gráfico.

## Entradas

- **canvas** (Canvas): Canvas en uso.
- **context** (CanvasRenderingContext2D): Contexto del canvas.
- **csvFilePath** (String): Ruta al archivo de datos.
- **xColumnName** (String): Nombre de la columna de datos a usar para el eje X.
- **yColumnName** (String): Nombre de la columna de datos a usar para el eje Y.
- **infoColumNames** (Array[String]): Arreglo con el nombre de las columnas que serán presentadas como información al pasar el cursor sobre el gráfico.
- **color** (String): Color de los puntos.
- **canvasPadding** (number): Padding del canvas.

## Salidas

No retorna valor.

## Método { }

```
drawDotPlot (canvas, context,
csvFilePath, xColumnName,
yColumnName, infoColumnNames, color,
canvasPadding){

      ...

 }
```

# Gráfica de líneas

## Uso

El método drawLinePlot dibuja una gráfica de línea basada en un conjunto de datos proporcionado mediante un archivo csv. Cada punto de datos se conecta mediante líneas rectas, creando una representación gráfica de la serie de datos.

## Entradas

- **canvas** (Canvas): Canvas.
- **context** (CanvasRenderingContext2D): Contexto del canvas.
- **csvFilePath** (String): Ruta al archivo de datos.
- **xColumnName** (String): Nombre de la columna de datos a usar para el eje X.
- **yColumnName** (String): Nombre de la columna de datos a usar para el eje Y.
- **infoColumNames** (Array[String]): Arreglo con el nombre de las columnas que serán presentadas como información al pasar el cursor sobre el gráfico.
- **color** (String): Color de los puntos.
- **canvasPadding** (number): Padding del canvas.

## Salidas

No retorna valor.

## Método { }

```
drawLinePlot(canvas, context,
csvFilePath, xColumnName,
yColumnName, infoColumnNames,
color, canvasPadding){

      ...   }
```

# Gráfica de barras

## Uso

El método drawBarPlot dibuja una gráfica de barras basada en un conjunto de datos proporcionado en un archivo csv. Cada barra representa una categoría y se dibuja en el canvas con una altura proporcional al conteo de ocurrencias en cada categoría.

## Entradas

- **canvas** (Canvas): Canvas.
- **context**: Contexto del canvas.
- **csvFilePath** (String): Ruta al archivo de datos.
- **xColumnName** (String): Nombre de la columna de categorías.
- **canvasPadding** (number): Padding del canvas.

## Salidas

No retorna valor.

## Método { }

```
drawBarPlot (canvas, context,
csvFilePath, xColumnName,
canvasPadding) {

      ...

      ...

 }
```

# Gráfica de burbujas

## Uso

El método drawBubblePlot dibuja una gráfica de burbujas basada en un conjunto de datos en un archivo csv. Cada burbuja se representa como un círculo en el gráfico, con un radio proporcional a un valor específico indicado.

## Entradas

- **canvas** (Canvas): Canvas.
- **context** (CanvasRenderingContext2D): contexto.
- **csvFilePath** (String): Ruta al archivo de datos.
- **xColumnName** (String): Nombre de la columna de datos a usar para el eje X.
- **yColumnName** (String): Nombre de la columna de datos a usar para el eje Y.
- **sizeColumnName** (String): Nombre de la columna que indicará el tamaño de las burbujas.
- **minSixe** (number): Tamaño mínimo de burbuja.
- **maxSixe** (number): Tamaño máximo de burbuja.
- **infoColumNames** (Array[String]): Arreglo con el nombre de las columnas que se desplegarán como información al pasar el cursor sobre el gráfico.
- **color** (String): Color de los puntos.
- **canvasPadding** (number): Padding del canvas.

## Salidas

No retorna valor.

## Método { }

```
drawBubblePlot (canvas, context, ...)
{

      ...    }
```

# Mapa de puntos

## Uso

El método mapDotPlot dibuja un mapa tomando como base un archivo GeoJSON proporcionado, además grafica puntos específicos sobre él, tomando en cuenta el archivo csv indicado.

## Entradas

- **canvas** (Canvas): Canvas.
- **context** (CanvasRenderingContext2D): Contexto.
- **mapDataFile** (String): Ruta al archivo GeoJSON
- **csvFilePath** (String): Ruta al archivo de datos.
- **xColumnName** (String): Nombre de la columna de datos a usar para la longitud.
- **yColumnName** (String): Nombre de la columna de datos a usar para la latitud.
- **infoColumNames** (Array[String]): Arreglo con el nombre de las columnas que se desplegarán como información al pasar el cursor sobre el gráfico.
- **color** (String): Color de los puntos.
- **canvasPadding** (number): Padding del canvas.

## Salidas

No retorna valor.

## Método { }

```
drawMapDotPlot(canvas, context,
mapDataFile, csvFilePath,
xColumnName, yColumnName,
infoColumNames, color, canvasPadding)
{...
  }
```

# Mapa de calor

## Uso

El método drawHeatMap dibuja un mapa tomando como base un archivo GeoJSON proporcionado. Posteriormente hace uso de un archivo csv proporcionado por el usuario para colorea cada región del mapa en base a una variable determinada.

## Entradas

- **canvas** (Canvas): Canvas.
- **canvasPadding** (number): Padding del canvas.
- **mapDataFile** (String): Ruta al archivo GeoJSON
- **csvFilePath** (String): Ruta al archivo de datos.
- **variableName** (String): Nombre de la columna de datos a usar para el degradado de color.
- **baseColor** (String): Color base para el degradado.
- **stateNameProperty** (String): Nombre de la propiedad que representa el identificador para cada región del mapa.
- **linkNameProperty** (String): Nombre de la columna en el archivo csv que identifica cada región del mapa y cuyos valores deben coincidir con los valores en stateNameProperty.
- **infoColumNames** (Array[String]): Arreglo con el nombre de las columnas que se desplegarán como información al pasar el cursor sobre el gráfico.

## Salidas

No retorna valor.

## Método { }

drawHeatMap (`canvas, canvasPadding, mapDataFile, csvFilePath, ...`) { ... }

# Casos de uso

# View Port

El siguiente ejemplo muestra cómo utilizar el método **initViewport** de la biblioteca vda.js.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DataCanvas Ejemplo initView</title>
</head>
<body>
  <! --La siguiente línea define el canvas "miCanvas"-->
  <canvas id="miCanvas"></canvas>
  <script type="module" src="./main.js"></script>
</body>
</html>
```

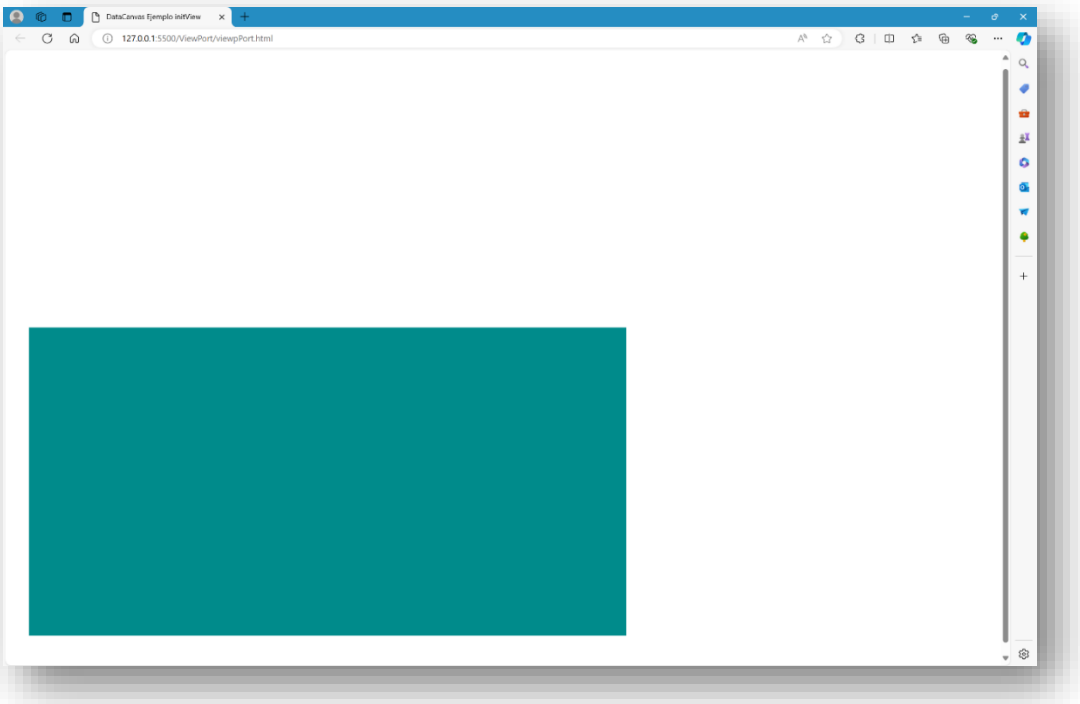
El archivo viewPort.html define un elemento Canvas identificado por un ID con nombre “miCanvas”.

Adicionalmente se tiene el archivo main.js el cual hace uso de la función initViewPort(), de la biblioteca vda.js, la cual recibe el id "miCanvas", y define los parámetros width y height con dimensiones 1000px cada uno. Posteriormente, para verificar el funcionamiento del código, dibujamos un rectángulo comenzando en el punto P(30,50) con ancho 2000 px y altura 500 px.

## Main.js

```
import {initViewport} from '../vda.js';

document.addEventListener('DOMContentLoaded', () => {

  // Llamamos a la función initViewPort de la biblioteca vda.js
  const width = 1000;
  const height = 1000;
  const context = initViewport('miCanvas', width, height);

  // Dibujamos rectángulo para verificar que el viewport funcion
  context.fillStyle = 'darkcyan';
  context.fillRect(30, 50, 2000, 500);
});
```

## Navegador



# Eje X

El siguiente ejemplo muestra cómo utilizar el método **drawXAxis** de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Canvas Ejemplo Eje X</title>
</head>

<body>
  <canvas id="miCanvas" style="border:1px solid #000000;"></canvas>
  <script type="module" src="./main.js"></script>
</body>

</html>
```

Posteriormente:

- Se importa el método initViewport y drawXAxis desde el archivo main.js.
- Cuando el documento está completamente cargado (DOMContentLoaded), inicializamos el canvas y su contexto mediante el método initViewport().

- Llamamos a drawXAxis con los parámetros necesarios para dibujar el eje X en el canvas.

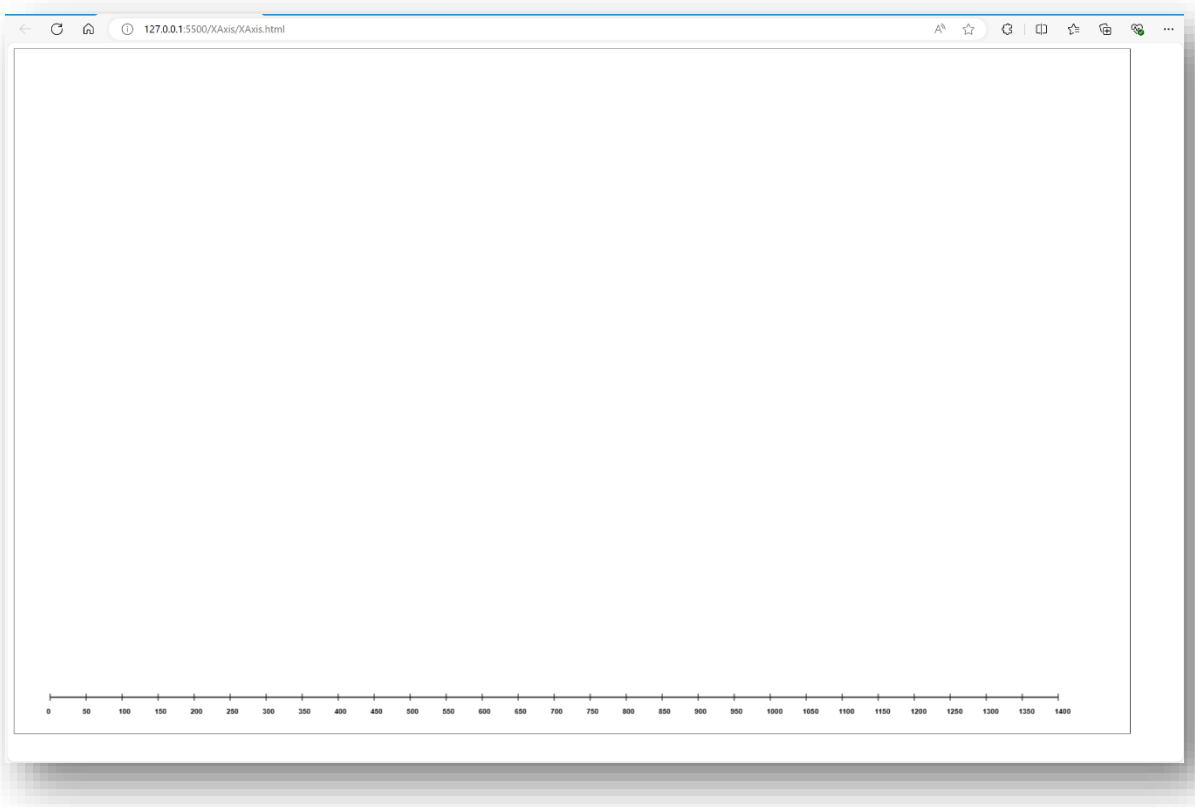
**Main.js**

```
import { initViewport, drawXAxis } from '../vda.js';

document.addEventListener('DOMContentLoaded', () => {
  const canvasId = 'miCanvas';
  const width = 1550;
  const height = 950;
  const context = initViewport(canvasId, width, height);

  // Configuración para el eje X
  const startX = 0;
  const endX = 1400;
  const step = 50;
  const xPosition = 50;
  const yPosition = 50;
  const color = 'black';
  const labelSpace = 20;

  drawXAxis(context, startX, endX, xPosition,yPosition, step, color,
labelSpace);
});
```

En este caso se está dibujando un eje X que inicia en 0 y finaliza en 1400, la posición del origen se encuentra en (50px, 50px), la separación en cada marca es de 50px y el espacio entre las leyendas (números de referencia en el eje) y el eje es de 20px.

En la siguiente imagen puede observarse el resultado, desplegado mediante un navegador web.

**Navegador**



# Eje Y

El siguiente ejemplo muestra cómo utilizar el método drawYAxis de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Canvas Ejemplo Eje Y</title>
</head>

<body>
  <canvas id="miCanvas" style="border:1px solid #000000;"></canvas>
  <script type="module" src="./main.js"></script>
</body>

</html>
```

Posteriormente:

- Se importa el método initViewport y drawYAxis desde el archivo main.js.
- Cuando el documento está completamente cargado (DOMContentLoaded), inicializamos el canvas y su contexto mediante el método initViewport().

- Llamamos a drawYAxis con los parámetros necesarios para dibujar el eje X en el canvas.

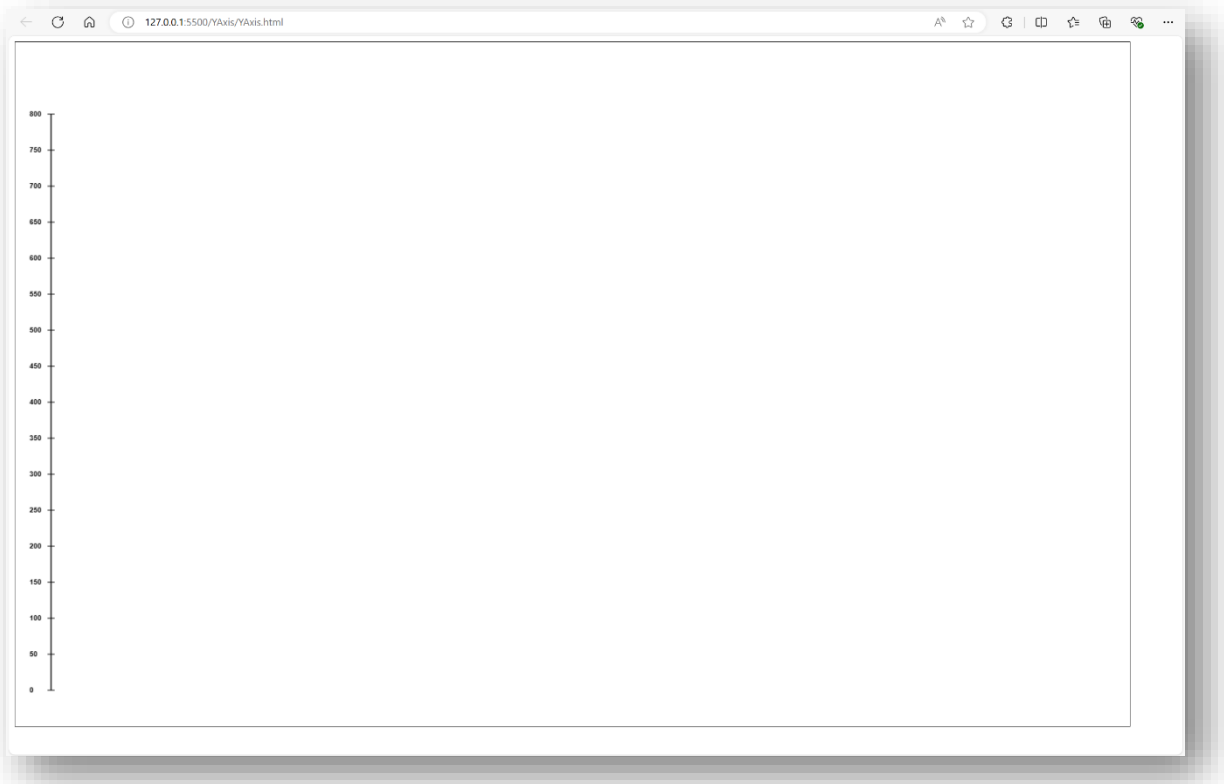
**Main.js**

```
import { initViewport, drawYAxis } from '../vda.js';

document.addEventListener('DOMContentLoaded', () => {
  const canvasId = 'miCanvas';
  const width = 1550;
  const height = 950;
  const context = initViewport(canvasId, width, height);

  // Configuración para el eje X
  const startY = 0;
  const endY = 800;
  const step = 50;
  const xPosition = 50;
  const yPosition = 50;
  const color = 'black';
  const labelSpace = 30;

  drawYAxis(context, startY, endY, xPosition, yPosition, step, color,
labelSpace);
});
```

En este caso se está dibujando un eje Y que inicia en 0 y finaliza en 800, la posición del origen se encuentra en (50px, 50px), la separación en cada marca es de 50px y el espacio entre las leyendas (números de referencia en el eje) y el eje es de 30px.

En la siguiente imagen puede observarse el resultado, desplegado mediante un navegador web.

## Navegador


127.0.0.1:5500/YAxis/YAxis.html
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0

# Eje X a partir de un arreglo de datos

El siguiente ejemplo muestra cómo utilizar el método drawXAxisFromArray de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo de Eje X a partir de un array</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

Posteriormente:

- Se importa el método initViewport, drawXAxisWithCSV y loadCSV desde el archivo main.js.
- Después, inicializamos el canvas y su contexto usando initViewport().
- Cargamos los datos mediante la lectura del archivo csv haciendo uso de la función loadCSV().

- Finalmente, llamamos a drawXAxisFromArray con los parámetros necesarios para dibujar el eje X en el canvas.

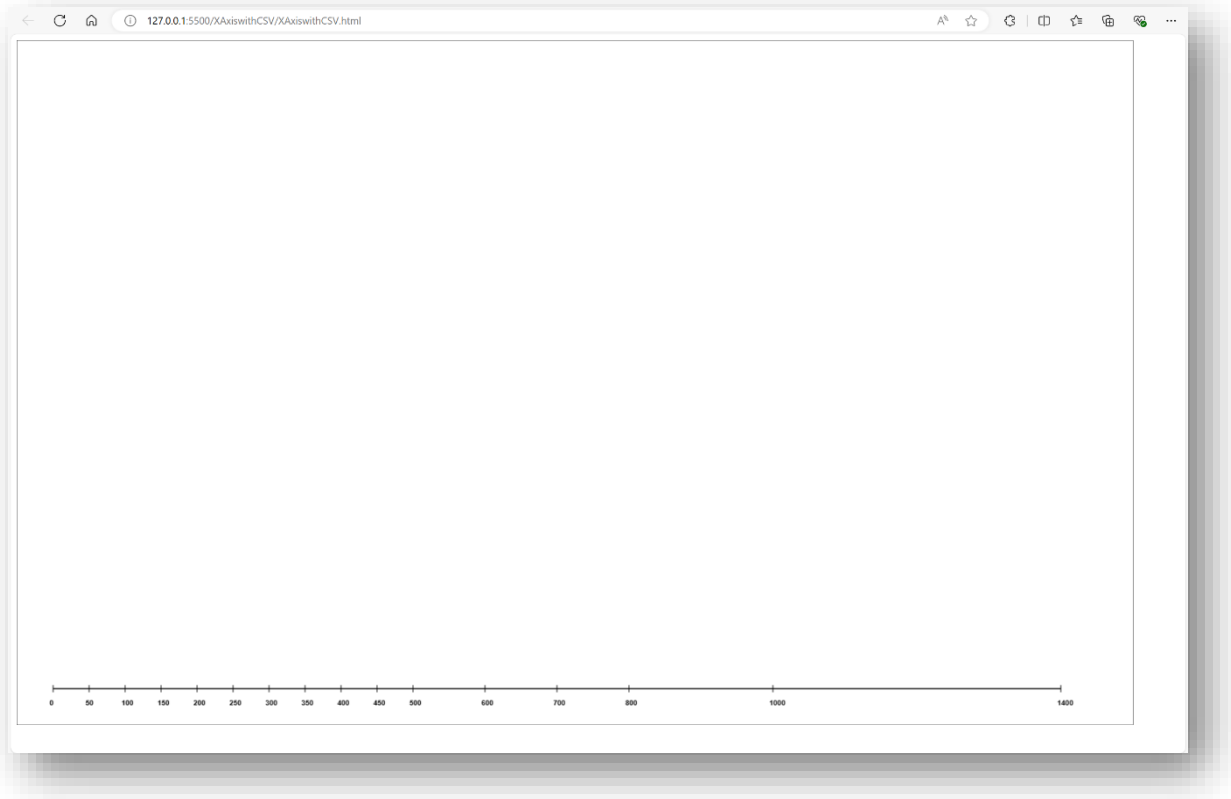
**Main.js**

```
import { initViewport, drawXAxisFromArray, loadCSV } from '../vda.js';

// Función principal para inicializar el canvas y dibujar el eje X
async function init() {

    //Init viewPort
    const canvasId = 'miCanvas';
    const width = 1550;
    const height = 950;
    const context = initViewport(canvasId, width, height);

    //Carga de datos desde CSV
    const data = await loadCSV("/data/xAxisData.csv");
    const xColumnName = "x"
    let xValues = data.map(row => row[xColumnName]);

    //Parametros para el eje X
    const xPos = 0;
    const yPos = 0;
    const color = "black";
    const labelSpace = 20;
    const canvasPadding = 50

    // Llamar a la función para dibujar el eje X
    drawXAxisFromArray(context, xValues , xPos, yPos, color,
labelSpace,canvasPadding);
}

// Llama a la función principal para inicializar todo
init();
```

En este caso, el arreglo de datos fue creado a partir del archivo xAxisData.csv, el cual contiene la una columna de datos con el encabezado “x”. El archivo puede consultarse en la carpeta “data” del paquete “Vda”.

Una vez desplegado, observamos un eje X que inicia en 0 y finaliza en 1400, y cuyos valores son tomados de la columna con nombre “x” del archivo csv cuya ruta es `'../data/xAxisData.csv'`. La posición del origen se encuentra en (50 px, 50px), el color de la línea se indica negro y el espacio entre las leyendas (números de referencia en el eje) y el eje es de 20px.

**Navegador**



# Eje Y a partir de un arreglo de datos

El siguiente ejemplo muestra cómo utilizar el método drawYAxisFromArray de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo de Eje Y a partir de un array</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
</head>
<body>
    <canvas id="miCanvas" style="border:1px solid #000000;"></canvas>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

Posteriormente:

- Se importa el método initViewport, drawYAxisWithCSV y loadCSV desde el archivo main.js.
- Después, inicializamos el canvas y su contexto usando initViewport().
- Cargamos los datos mediante la lectura del archivo csv haciendo uso de la función loadCSV().

- Finalmente, llamamos a drawYAxisFromArray con los parámetros necesarios para dibujar el eje X en el canvas.

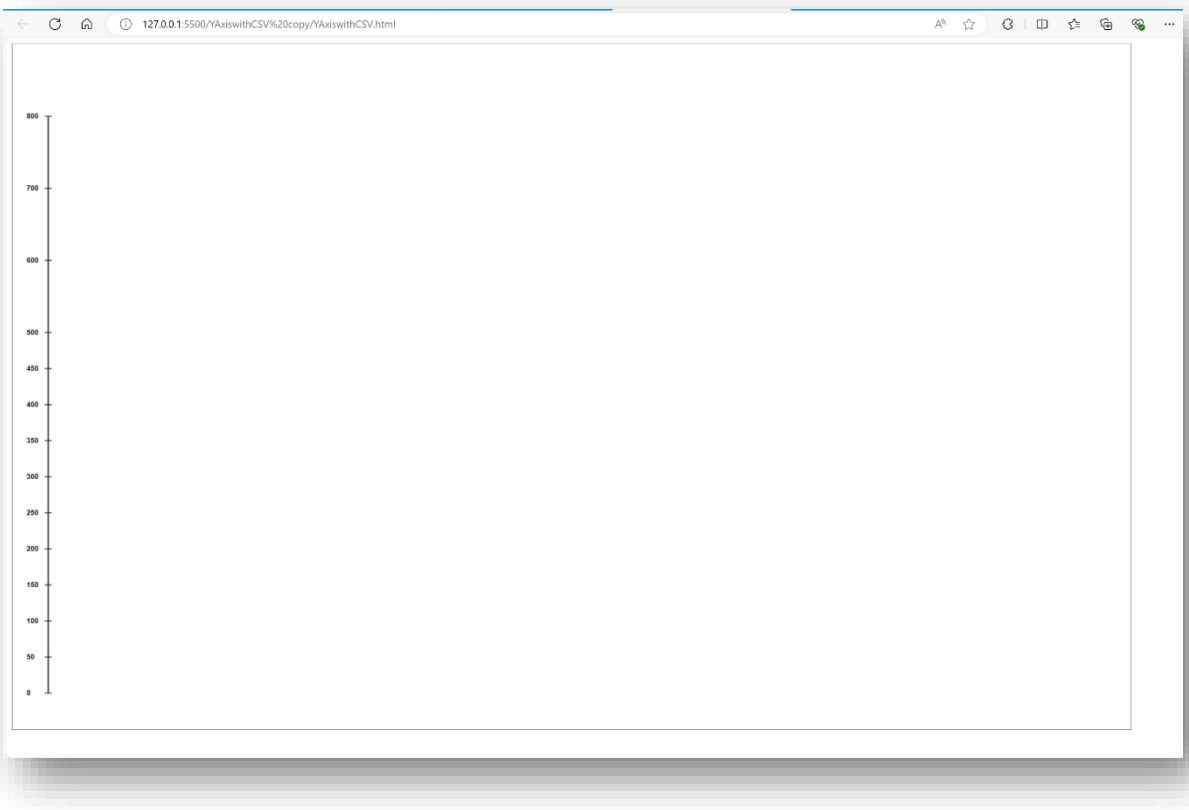
**Main.js**

```
import { initViewport, drawYAxisFromArray, loadCSV } from '../vda.js';

// Función principal para inicializar el canvas y dibujar el eje X
async function init() {

    const canvasId = 'miCanvas';
    const width = 1550;
    const height = 950;
    const context = initViewport(canvasId, width, height);

    //Carga de datos desde CSV
    const data = await loadCSV("/data/yAxisData.csv");
    const yColumnName = "y";
    let yValues = data.map(row => row[yColumnName]);

    //Parametros para dibujar eje Y
    const yPosition = 0;
    const XPosition = 0;
    const color = "black";
    const labelSpace = -10;
    const canvasPadding = 50;

    // Llamar a la función para dibujar el eje Y
    drawYAxisFromArray(context, yValues, XPosition, yPosition, color,
labelSpace,canvasPadding);
}

// Llama a la función principal para inicializar todo
init();
```

En este caso, el arreglo de datos fue creado a partir del archivo yAxisData.csv, el cual contiene la una columna de datos con el encabezado “y”. El archivo puede consultarse en la carpeta “data” del paquete “Vda”.

Una vez desplegado, observamos un eje Y que inicia en 0 y finaliza en 800, y cuyos valores son tomados de la columna con nombre “y” del archivo csv cuya ruta es `'../data/yAxisData.csv'`. La posición del origen se encuentra en (50px, 50px), el color de la línea se indica negro y el espacio entre las leyendas (números de referencia en el eje) y el eje es de 30px.

**Navegador**



# Gráfica de puntos

El siguiente ejemplo muestra cómo utilizar el método drawDotPlot de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo Gráfica de puntos</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <div id="infoOverlay" class="info-overlay"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

En este caso, también nos es posible agregar una etiqueta (de forma opcional)

```
<div id="infoOverlay"></div>
```

la cual nos permitirá agregar una capa de información desplegable cada vez que el mouse sea deslizado sobre el gráfico.

Posteriormente:

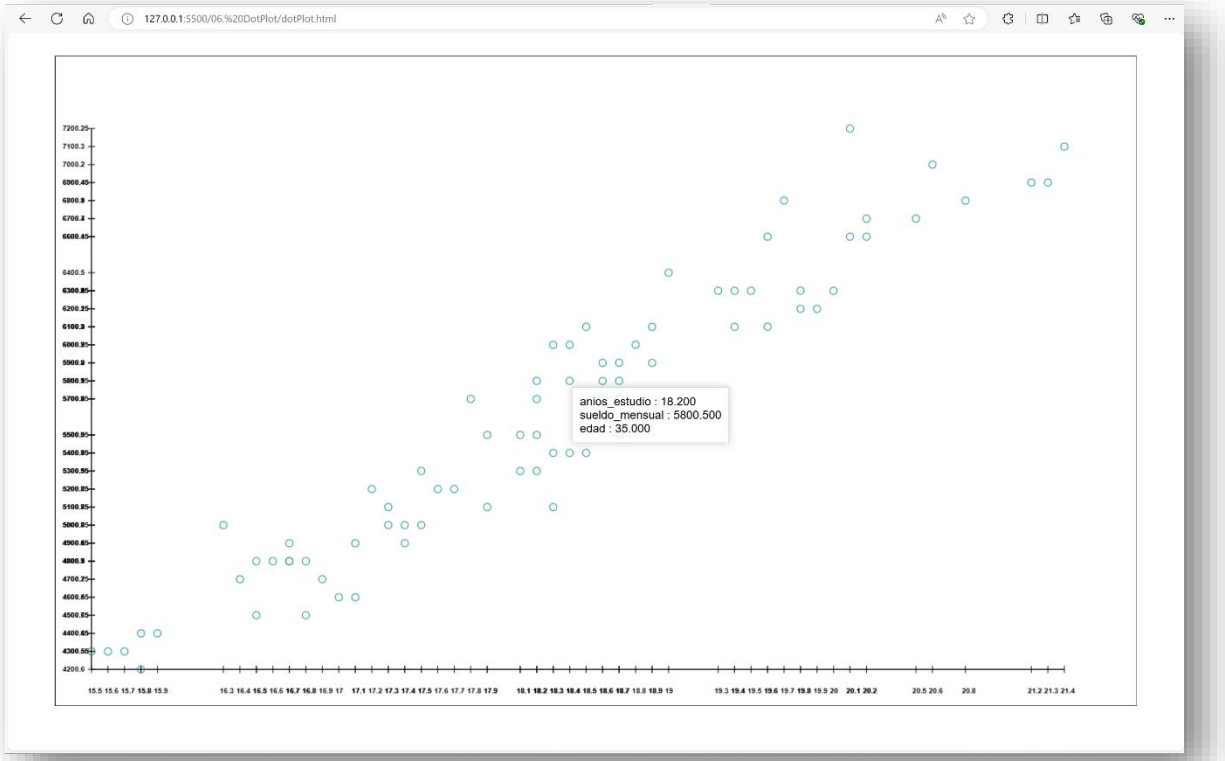
- Se importa los métodos initViewport , drawDotPlot y loasCSV desde el archivo main.js, y opcionalmente, drawXAxisFromArray, drawYAxisFromArray.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Llamamos a drawDotPlot con los parámetros necesarios para el gráfico.

**Main.js**

```
import { initViewport, drawDotPlot, drawXAxisFromArray,
drawYAxisFromArray, loadCSV } from '../vda.js';

document.addEventListener('DOMContentLoaded', async () => {

 //Init ViewPort
  const canvasId='miCanvas'; const width = 1500; const height = 900;
  const context = initViewport(canvasId, width, height);

  //Carga de datos desde CSV
  const filePath  = "/data/salarios.csv";
  const data = await loadCSV(filePath);

  //Parametros para la gráfica
  const canvas = document.getElementById("miCanvas");
  const xColumnName = "anios_estudio"
  const yColumnName = "sueldo_mensual";
  let infoColumnNames = [xColumnName, yColumnName, "edad"]
  const color = "darkcyan"; const canvasPadding = 50;

  drawDotPlot(canvas, context, filePath, xColumnName,yColumnName,
  infoColumnNames, color, canvasPadding);

  //Parametros para el eje X
  let xValues = data.map(row => row[xColumnName]);
  const xPos = 0; const yPos = 0; const labelSpace = 20;
  const color2 = "black";
  drawXAxisFromArray(context, xValues , xPos, yPos, color2,
  labelSpace,canvasPadding);

  //Parametros para dibujar eje Y
  let yValues = data.map(row => row[yColumnName]);
  const yPosition = 0; const XPosition = 0; const labelSpace2 = -10;
  drawYAxisFromArray(context, yValues, XPosition, yPosition,
  color2, labelSpace2,canvasPadding);

});
```

En la siguiente imagen podemos apreciar el gráfico de puntos dibujado en el navegador web.

**Navegador**



# Gráfica de líneas

El siguiente ejemplo muestra cómo utilizar el método drawLinePlot de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo Gráfica de línea</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <div id="infoOverlay" class="info-overlay"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

En este caso, también nos es posible agregar una etiqueta (de forma opcional)

```
<div id="infoOverlay"></div>
```

la cual nos permitirá agregar una capa de información desplegable cada vez que el mouse sea deslizado sobre el gráfico.

Posteriormente:

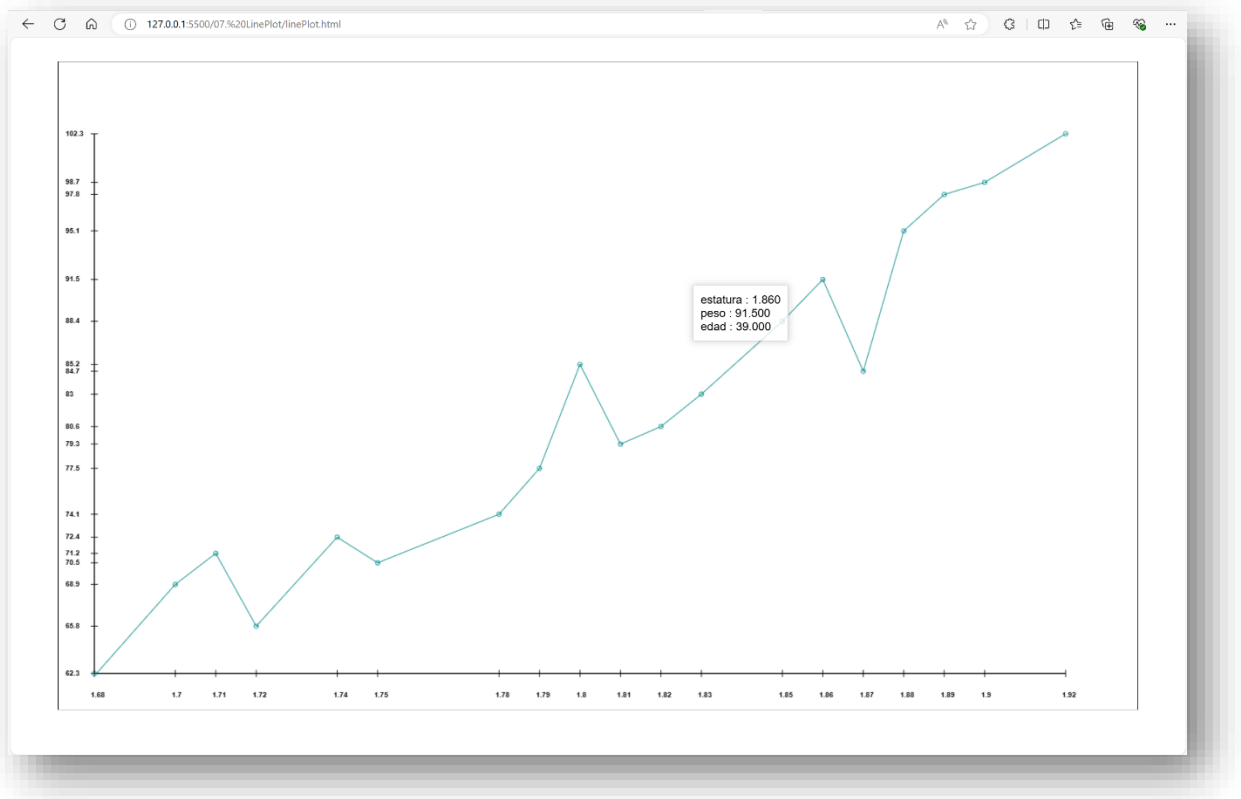
- Se importa los métodos initViewport , drawLinePlot y loasCSV desde el archivo main.js, y opcionalmente, drawXAxisFromArray, drawYAxisFromArray.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Llamamos a drawDotPlot con los parámetros necesarios para el gráfico.

**Main.js**

```
import { initViewport, drawLinePlot, drawXAxisFromArray,
drawYAxisFromArray, loadCSV } from '../vda.js';

document.addEventListener('DOMContentLoaded', async () => {

  //Init ViewPort
  const canvasId = 'miCanvas';
  const width = 1500; const height = 900;
  const context = initViewport(canvasId, width, height);

  //Carga de datos desde CSV
  const filePath  = "/data/pesoEstatura.csv";
  const data = await loadCSV(filePath);

  //Parametros para la gráfica
  const canvas = document.getElementById("miCanvas");
  const xColumnName = "estatura"; const yColumnName = "peso";
  let infoColumnNames = [xColumnName, yColumnName, "edad"]
  const color = "darkcyan";
  const canvasPadding = 50;

  drawLinePlot(canvas, context, filePath, xColumnName,yColumnName,
  infoColumnNames, color, canvasPadding);

  //Parametros para el eje X
  let xValues = data.map(row => row[xColumnName]);
  const xPos = 0; const yPos = 0; const labelSpace = 20;
  const color2 = "black";
  drawXAxisFromArray(context, xValues , xPos, yPos, color2,
  labelSpace,canvasPadding);

  //Parametros para dibujar eje Y
  let yValues = data.map(row => row[yColumnName]);
  const yPosition = 0; const XPosition = 0; const labelSpace2 = -10;
  drawYAxisFromArray(context, yValues, XPosition, yPosition, color2,
labelSpace2,canvasPadding);

});
```

En la siguiente imagen podemos apreciar el gráfico de línea dibujado en el navegador web.

**Navegador**



# Gráfica de barras

El siguiente ejemplo muestra cómo utilizar el método drawBarPlot de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo Gráfica de barras</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <div id="infoOverlay" class="info-overlay"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

En este caso, también nos es posible agregar una etiqueta (de forma opcional)

```
<div id="infoOverlay"></div>
```

la cual nos permitirá agregar una capa de información desplegable cada vez que el mouse sea deslizado sobre el gráfico.

Posteriormente:

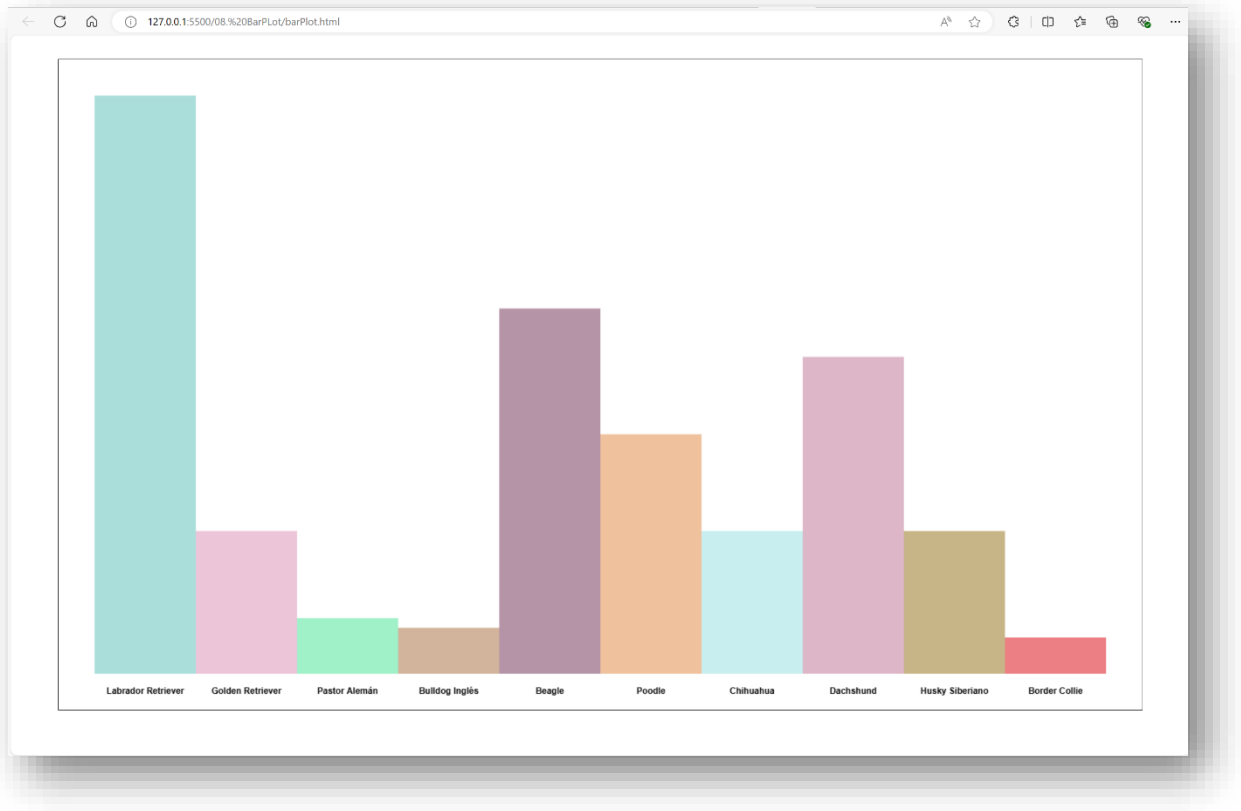
- Se importa los métodos initViewport , drawBarPlot y loasCSV desde el archivo main.js, y opcionalmente, drawXAxisFromArray, drawYAxisFromArray.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Llamamos a drawBarPlot con los parámetros necesarios para el gráfico.

**Main.js**

```
import { initViewport,
         drawBarPlot,
         drawXAxisFromArray,
         drawYAxisFromArray,
         loadCSV } from '../vda.js';

document.addEventListener('DOMContentLoaded', async () => {

  //Init ViewPort
  const canvasId = 'miCanvas';
  const width = 1500;
  const height = 900;
  const context = initViewport(canvasId, width, height);


  //Carga de datos desde CSV
  const filePath  = "/data/perros.csv";
  const data = await loadCSV(filePath);


  //Parametros para la gráfica
  const canvas = document.getElementById("miCanvas");
  const xColumnName = "raza"
  const canvasPadding = 50;


  drawBarPlot(canvas, context, filePath, xColumnName,
  canvasPadding);


});
```

En la siguiente imagen podemos apreciar el gráfico de línea dibujado en el navegador web.

**Navegador**



# Gráfica de burbujas

El siguiente ejemplo muestra cómo utilizar el método drawBubblePlot de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo Gráfica de burbujas</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <div id="infoOverlay" class="info-overlay"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

En este caso, también nos es posible agregar una etiqueta (de forma opcional)

```
<div id="infoOverlay"></div>
```

la cual nos permitirá agregar una capa de información desplegable cada vez que el mouse sea deslizado sobre el gráfico.

Posteriormente:

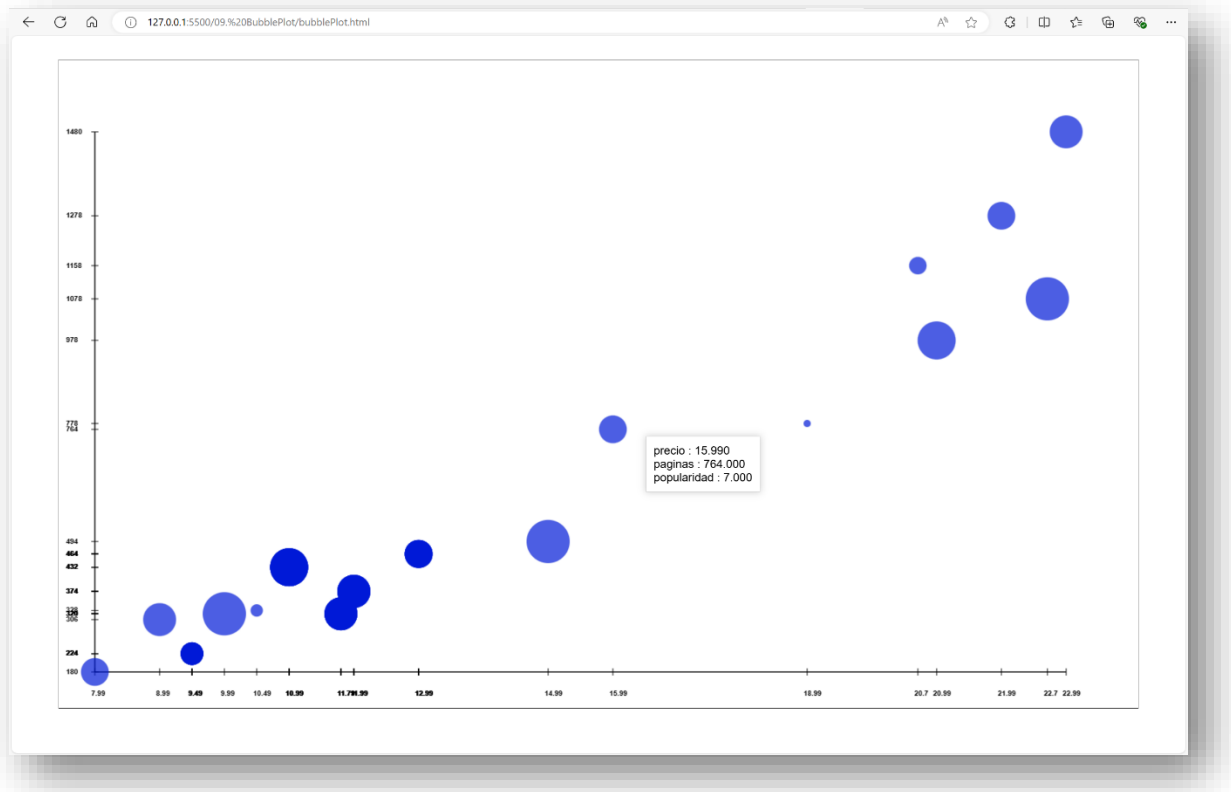
- Se importa los métodos initViewport , drawBubblePlot y loasCSV desde el archivo main.js, y opcionalmente, drawXAxisFromArray, drawYAxisFromArray.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Llamamos a drawBubblePlot con los parámetros necesarios para el gráfico.

**Main.js**

```
import { initViewport, drawBubblePlot, drawXAxisFromArray,
drawYAxisFromArray, loadCSV } from '../vda.js';

document.addEventListener('DOMContentLoaded', async () => {

  //Init ViewPort
  const canvasId ='miCanvas'; const width =1500; const height =900;
  const context = initViewport(canvasId, width, height);

  //Carga de datos desde CSV
  const filePath  = "/data/libros.csv";
  const data = await loadCSV(filePath);

  //Parametros para la gráfica
  const canvas = document.getElementById("miCanvas");
  const xColumnName = "precio"; const yColumnName = "paginas";
  const zColumnName = "popularidad";
  let infoColumnNames = [xColumnName, yColumnName, zColumnName];
  const minSize = 5; const maxSize = 20; const transparence = 0.5;
  const color = 'rgba(0, 25, 215, 0.7)'; const canvasPadding = 50;

  drawBubblePlot(canvas, context, filePath, xColumnName,yColumnName,
  zColumnName, minSize, maxSize, infoColumnNames,  color,
  canvasPadding);

  //Parametros para el eje X
  let xValues = data.map(row => row[xColumnName]);
  const xPos=0;const yPos=0;const labelSpace=20;const color2="black";
  drawXAxisFromArray(context, xValues , xPos, yPos, color2,
  labelSpace,canvasPadding);

  //Parametros para dibujar eje Y
  let yValues = data.map(row => row[yColumnName]);
  const yPosition = 0; const XPosition = 0;const labelSpace2 = -10;
  drawYAxisFromArray(context, yValues, XPosition, yPosition, color2,
  labelSpace2,canvasPadding);
});
```

En la siguiente imagen podemos apreciar el gráfico de línea dibujado en el navegador web.

**Navegador**



# Mapa de puntos

El siguiente ejemplo muestra cómo utilizar el método drawMapDotPlot de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo: Mapa de puntos</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
    <script src="https://d3js.org/d3.v6.min.js"></script>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <div id="infoOverlay" class="info-overlay"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

En este caso, también nos es posible agregar una etiqueta (de forma opcional)

```
<div id="infoOverlay"></div>
```

la cual nos permitirá agregar una capa de información desplegable cada vez que el mouse sea deslizado sobre el gráfico.

Posteriormente:

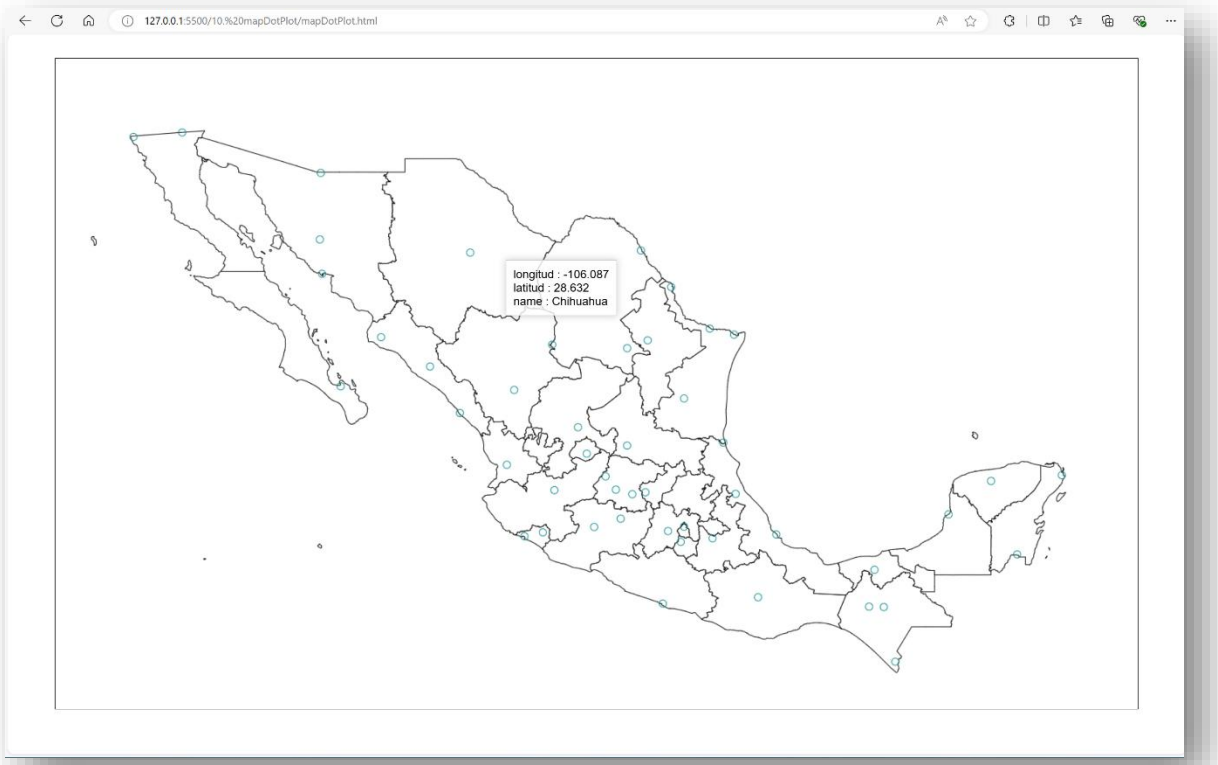
- Se importan los métodos initViewport y drawMapDotPlot desde el archivo main.js.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Llamamos a drawMapDotPlot con los parámetros necesarios para el gráfico.

**Main.js**

```
import { initViewport, drawMapDotPlot } from '../vda.js';

document.addEventListener('DOMContentLoaded', async () => {

  //Init ViewPort
  const canvasId = 'miCanvas';
  const width = 1500;
  const height = 900;
  const context = initViewport(canvasId, width, height);

  const canvas = document.getElementById(canvasId);
  const canvasPadding = 50;
  const mapDataFile = "/data/states.geojson"

  //Carga de datos desde CSV
  const csvFilePath  = "/data/puntos_mapa.csv";

  //Parametros para la gráfica
  const longitudeColumnName = "longitud"
  const latitudeColumnName = "latitud";
  let infoColumnNames = [longitudeColumnName, latitudeColumnName,
  "name"]
  const color = "darkcyan"

  drawMapDotPlot(canvas, context, mapDataFile, csvFilePath,
  longitudeColumnName, latitudeColumnName, infoColumnNames, color,
  canvasPadding)

  //drawDotPlot(canvas, context, filePath, xColumnName,yColumnName,
  infoColumnNames, color, canvasPadding);


});
```

En la siguiente imagen podemos apreciar el gráfico de línea dibujado en el navegador web.

**Navegador**



# Mapa de burbujas

El siguiente ejemplo muestra cómo utilizar el método drawMapBubblePlot de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo: Mapa de burbujas</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
    <script src="https://d3js.org/d3.v6.min.js"></script>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <div id="infoOverlay" class="info-overlay"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

En este caso, también nos es posible agregar una etiqueta (de forma opcional)

```
<div id="infoOverlay"></div>
```

la cual nos permitirá agregar una capa de información desplegable cada vez que el mouse sea deslizado sobre el gráfico.

Posteriormente:

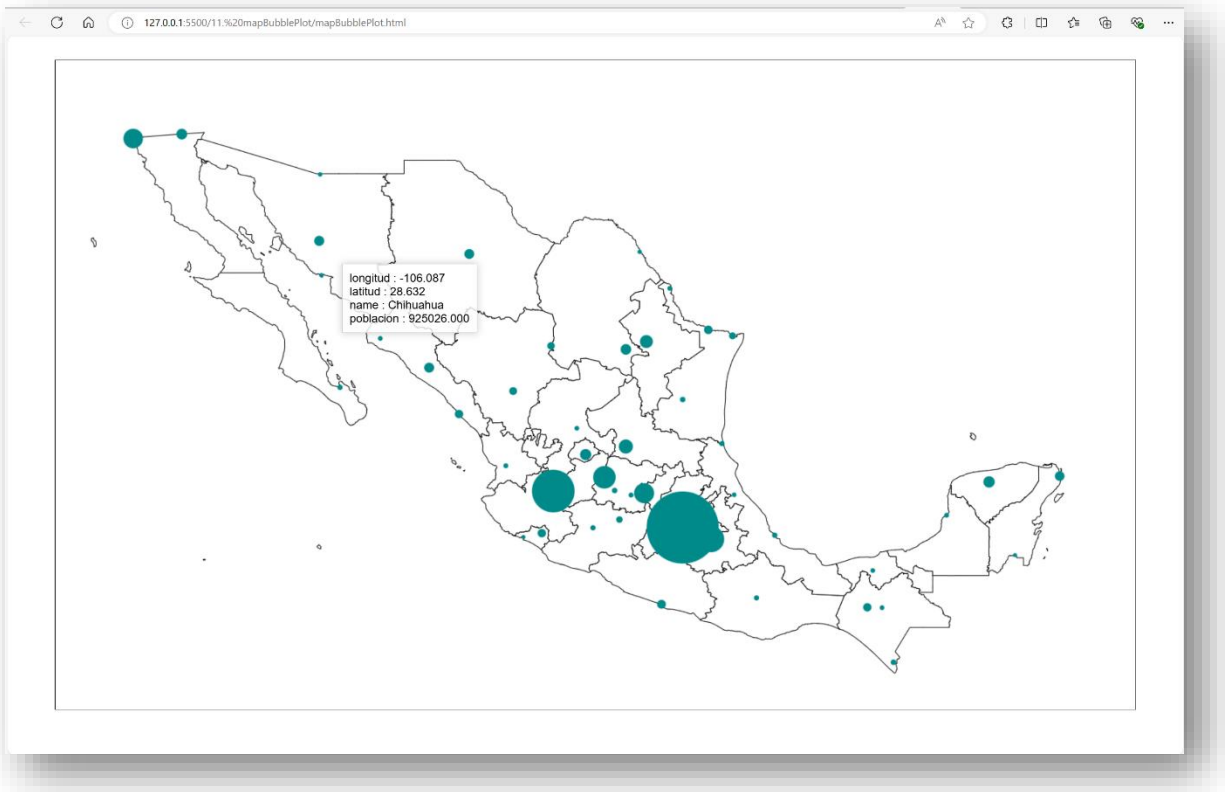
- Se importan los métodos initViewport y drawMapBubblePlot desde el archivo main.js.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Llamamos a drawMapBubblePlot con los parámetros necesarios para el gráfico.

**Main.js**

```
import { initViewport, drawMapBubblePlot } from '../vda.js';

document.addEventListener('DOMContentLoaded', async () => {

  //Init ViewPort
  const canvasId = 'miCanvas';
  const width = 1500;
  const height = 900;
  const context = initViewport(canvasId, width, height);

  const canvas = document.getElementById(canvasId);
  const canvasPadding = 50;
  const mapDataFile = "/data/states.geojson"

  //Carga de datos desde CSV
  const csvFilePath  = "/data/bubbles_mapa.csv";

  //Parametros para la gráfica
  const longitudeColumnName = "longitud"
  const latitudeColumnName = "latitud";
  const sizeColumnName = "poblacion";
  const minSizeBubble = 2;
  const maxSizeBubble = 50;
  let infoColumnNames = [longitudeColumnName, latitudeColumnName,
  "name" , "poblacion"]
  const color = "darkcyan"

  drawMapBubblePlot(canvas, context, mapDataFile, csvFilePath,
  longitudeColumnName, latitudeColumnName,
  sizeColumnName,minSizeBubble, maxSizeBubble, infoColumnNames,
  color, canvasPadding)

  //drawDotPlot(canvas, context, filePath, xColumnName,yColumnName,
  infoColumnNames, color, canvasPadding);


});
```

En la siguiente imagen podemos apreciar el gráfico de línea dibujado en el navegador web.

**Navegador**



# Mapa de color  (República Mexicana)

El siguiente ejemplo muestra cómo utilizar el método drawHeatMap de la biblioteca vda.js.

Definimos un elemento canvas (miCanvas) donde se dibujarán los elementos gráficos.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo: Mapa de color México</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
    <script src="https://d3js.org/d3.v6.min.js"></script>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <div id="infoOverlay" class="info-overlay"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

En este caso, también nos es posible agregar una etiqueta (de forma opcional)

```
<div id="infoOverlay"></div>
```

la cual nos permitirá agregar una capa de información desplegable cada vez que el mouse sea deslizado sobre el gráfico.

Posteriormente:

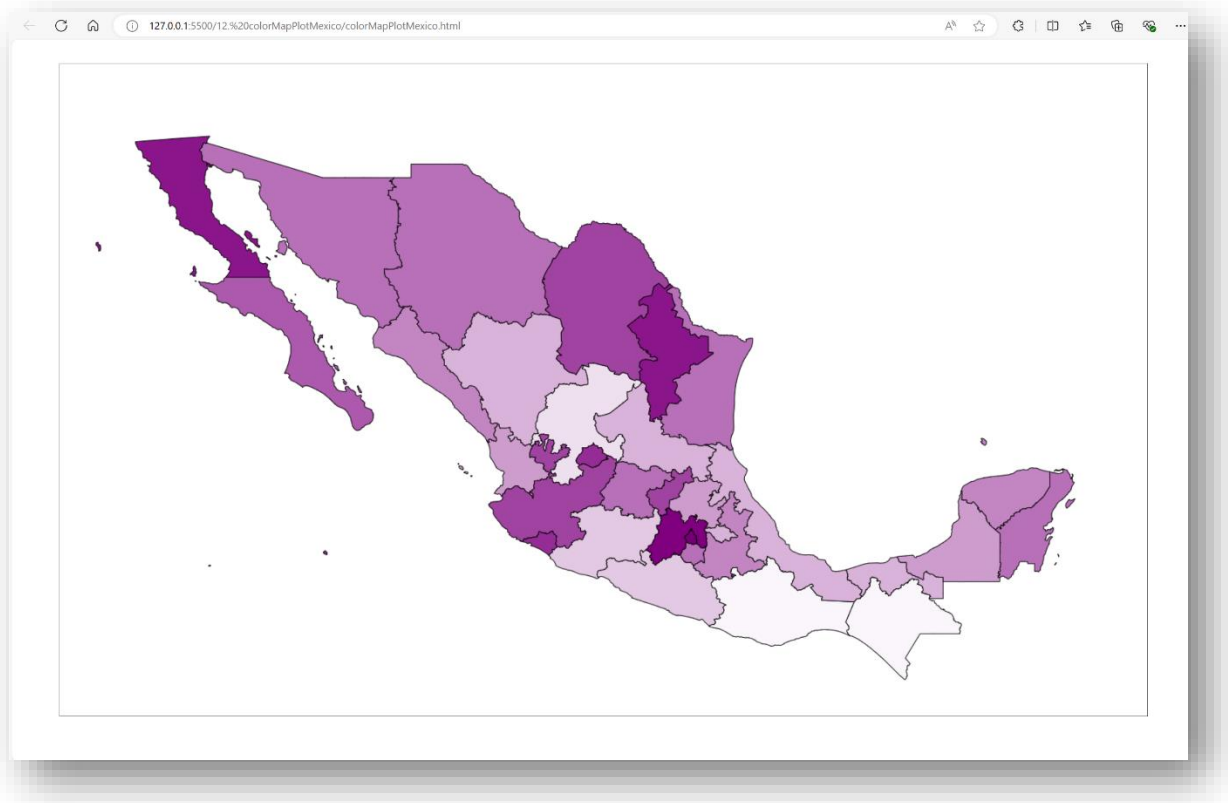
- Se importan los métodos initViewport y drawHeatMap desde el archivo main.js.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Proporcionamos los archivos GeoJson y CSV para la geometría del mapa y los datos a representar respectivamente.
- Definimos los valores `linkNameProperty`, `stateNameProperty`, que son los identificadores de las columnas que comparten ambos sets de datos y que sirven como llaves únicas para enlazar ambos.
- Llamamos a drawHeatMap con los parámetros necesarios para el gráfico.

**Main.js**

```
import { initViewport, drawHeatMap } from '../vda.js';

document.addEventListener('DOMContentLoaded', async () => {

  //Init ViewPort
  const canvasId = 'miCanvas';
  const width = 1500;
  const height = 900;
  const context = initViewport(canvasId, width, height);

  //Obtención de polígonos para dibujar
  const canvasPadding = 50;
  const mapDataFile = "/data/states.geojson"

  // Parámetros para el heatMap
  const canvas = document.getElementById(canvasId);
  const csvFilePath = '/data/colorMap_Mexico.csv';
  const variableName ='urbanizacion';//Nombre de la columna en el CSV
  //const variableName='poblacion'; // Nombre de la columna en el CSV
  //const variableName='temperatura';//Nombre de la columna en el CSV
  const stateNameProperty = "state_name" //nombre de la propiedad en
  el archivo geojson
  const linkNameProperty = "nombre" //nombre de la columna con la que
  se comparará el stateNameProperty
  let infoColumnNames = ["nombre", "temperatura", "poblacion",
  "urbanizacion"];

  drawHeatMap(canvas, canvasPadding, mapDataFile, csvFilePath,
  variableName, "purple", stateNameProperty, linkNameProperty,
  infoColumnNames);

});
```

En la siguiente imagen podemos apreciar el gráfico de línea dibujado en el navegador web.

**Navegador**



# Mapa de color (USA)

Este ejemplo sirve para enfatizar que es posible indicar un archivo GeoJSON propio y no únicamente los dos establecidos en estos casos de uso; esto permite dibujar cualquier geometría siempre y cuando se respete la estructura de este tipo de archivos.

Definimos un elemento canvas (miCanvas) donde se dibujará el mapa de calor.

**viewPort.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo: Mapa de color USA</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"
></script>
    <script src="https://d3js.org/d3.v6.min.js"></script>
    <link rel="stylesheet" href="./styles.css">
</head>
<body>
    <canvas id="miCanvas" width="1200" height="900" style="border:1px solid
#000000;"></canvas>
    <div id="infoOverlay" class="info-overlay"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

En este caso, también nos es posible agregar una etiqueta (de forma opcional)

```
<div id="infoOverlay"></div>
```

la cual nos permitirá agregar una capa de información desplegable cada vez que el mouse sea deslizado sobre el gráfico.

Posteriormente:

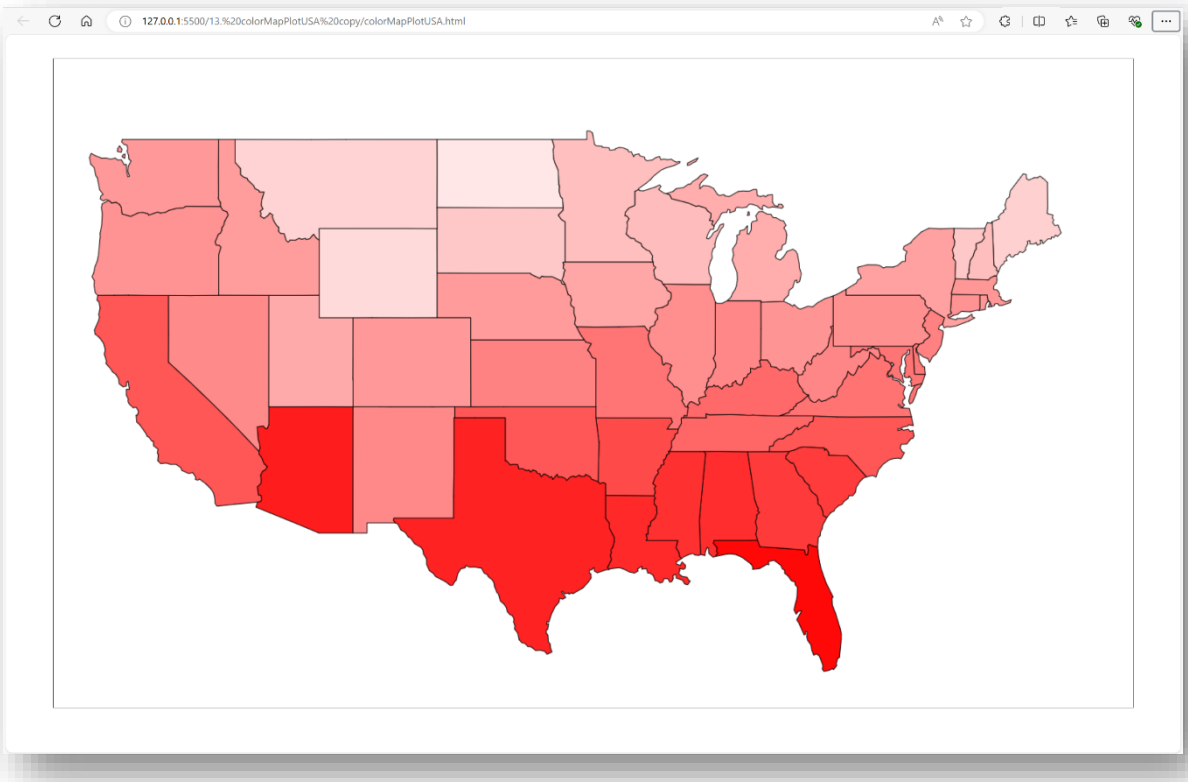
- Se importan los métodos initViewport y drawHeatMap desde el archivo main.js.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Proporcionamos los archivos GeoJson y CSV para la geometría del mapa y los datos a representar respectivamente.
- Definimos los valores `linkNameProperty`, `stateNameProperty`, que son los identificadores de las columnas que comparten ambos sets de datos y que sirven como llaves únicas para enlazar ambos.
- Llamamos a drawHeatMap con los parámetros necesarios para el gráfico.

**Main.js**

```
import { initViewport, drawHeatMap } from '../vda.js';

document.addEventListener('DOMContentLoaded', async () => {

  //Init ViewPort
  const canvasId = 'miCanvas';
  const width = 1500; const height = 900;
  const context = initViewport(canvasId, width, height);

  //Obtención de polígonos para dibujar
  const canvasPadding = 50;
  const mapDataFile = "/data/usaStates_simple.geojson"

  // Parámetros para el heatMap
  const canvas = document.getElementById(canvasId);
  const csvFilePath = '/data/colorMap_USA.csv';
  //const variableName='urbanizacion';//Nombre de columna en el CSV
  //const variableName='poblacion'; // Nombre de la columna en el CSV
  const variableName ='temperatura';// Nombre de la columna en el CSV
  const stateNameProperty = "name" //nombre de la propiedad en el
  archivo geojson
  const linkNameProperty = "nombre" //nombre de la columna con la que
  se comparará el stateNameProperty
  let infoColumnNames = ["nombre", "temperatura", "poblacion",
  "urbanizacion"];

  drawHeatMap(canvas, canvasPadding, mapDataFile, csvFilePath,
  variableName, "red", stateNameProperty, linkNameProperty,
  infoColumnNames);

});
```

En la siguiente imagen podemos apreciar el gráfico de línea dibujado en el navegador web.

**Navegador**




# Referencias

Repositorio GitHub. El código para implementar los casos de uso, así como los archivos de datos csv y geoJson utilizados en esta guía, pueden obtenerse directamente del siguiente repositorio.

# VDA

## Librería para Visualización de Datos

García Aguilar Luis Alberto