INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

Biblioteca
Visualización de Datos Avanzada

Importancia del impacto visual, la creatividad y la libertad en la creación de gráficos personalizados

Contenido

1. Introducción

El presente documento expone el caso práctico desarrollado como parte de mi proceso de titulación en la Maestría en Ciencia de Datos del Instituto Tecnológico Autónomo de México (ITAM). Este trabajo amplía y detalla los resultados obtenidos durante mi Estancia de Investigación, llevada a cabo bajo la supervisión del Doctor José Ezequiel Soto Sánchez.

El proyecto consiste en la creación de una biblioteca de visualización de datos enfocada en apoyar el entendimiento y la presentación de datos en contextos educativos y profesionales con el fin de permitir a los usuarios crear gráficos personalizados, trascendiendo las limitaciones de herramientas predefinidas. El origen de esta biblioteca radica en la necesidad de un recurso didáctico para la materia de Visualización de Datos Avanzada, impartida por primera vez en el ITAM durante el semestre enero-mayo 2024. Este curso planteó el reto de enseñar técnicas avanzadas de visualización a estudiantes con antecedentes diversos, incluyendo aquellos sin experiencia previa en lenguajes de programación como JavaScript. Así, uno de los objetivos principales del producto, es servir como un puente entre la teoría y la práctica, facilitando tanto la comprensión de conceptos complejos como la exploración creativa en la presentación de datos.

A diferencia de otras herramientas de visualización existentes, VDA permite a los usuarios diseñar gráficos hechos a la medida de sus necesidades, eliminando la necesidad de ajustar los datos a formatos predefinidos. Este enfoque fomenta la libertad creativa y permite explorar el arte como un componente esencial en la comunicación visual, haciendo que cada gráfico no solo sea funcional, sino también estéticamente impactante y adecuado al contexto.

El desarrollo de la biblioteca requirió un dominio de tecnologías web como HTML5, CSS y JavaScript, así como conocimientos en diseño gráfico, teoría del color y experiencia de usuario. En cada etapa del proyecto, busqué integrar estos elementos para garantizar que VDA sea intuitiva, accesible y escalable, proporcionando un recurso educativo que inspire a futuros científicos de datos a innovar en la forma en que presentan y exploran la información.

Más allá de ser una solución técnica o un recurso académico, esta biblioteca es una invitación a repensar las herramientas de visualización de datos. Al analizar sus capacidades frente a otras opciones del mercado, también se destacan sus limitaciones, permitiendo identificar áreas de mejora y oportunidades de expansión. Considero que el impacto de este proyecto no se limita a su aplicación en el aula, sino que tiene el potencial de convertirse en una plataforma que fomente la creatividad y la personalización en el ámbito de la Ciencia de Datos.

Mediante este proceso, no solo he consolidado parte de los conocimientos adquiridos durante mi maestría, sino que también he respondido a la necesidad de brindar un recurso efectivo para la materia de Visualización de Datos Avanzada y me alegra poder apoyar en la educación de futuros estudiantes.

2. Justificación

La visualización de datos es una herramienta esencial para comunicar información de manera efectiva, al transformar datos complejos en representaciones claras y visualmente atractivas. Si bien existen diversas herramientas en el mercado que facilitan este proceso, muchas de ellas presentan limitaciones importantes para ciertos usuarios y escenarios. A menudo, estas herramientas están diseñadas con gráficos predefinidos que, aunque funcionales, restringen la capacidad de personalización. Esto obliga a los usuarios a ajustar sus datos al formato establecido, en lugar de permitirles crear representaciones totalmente adaptadas a sus necesidades específicas.

Adicionalmente, muchas de estas herramientas requieren conocimientos avanzados en programación o licencias costosas, lo que puede excluir a quienes no poseen estas competencias o recursos. Por otro lado, su enfoque práctico, aunque eficiente, a menudo descuida el aspecto artístico y de personalización en la visualización de datos, ignorando que el diseño visual impacta directamente en la interpretación y comprensión de la información.

3. Objetivos

Accesibilidad:

Una de las principales limitaciones de muchas herramientas de visualización de datos en el mercado es su falta de flexibilidad y accesibilidad para una amplia gama de usuarios. VDA intenta aminorar esta brecha proporcionando una biblioteca de visualización de datos diseñada para ser intuitiva, escalable y accesible incluso para aquellos con poca experiencia en programación.

Escalabilidad

A medida que los proyectos crecen en complejidad, VDA permite integrar funcionalidades avanzadas**,** esto es posible gracias a su enfoque modular, que facilita la adición de nuevos elementos y características sin que el rendimiento o la usabilidad se vean comprometidos.

Flexibilidad y personalización:

La herramienta busca no solo garantizar que los gráficos sean funcionales, sino que también ofrece flexibilidad para personalizar el aspecto visual y funcional de las gráficas, permitiendo a los usuarios diseñar gráficos adaptados a sus datos y necesidades. Esto se logra proporcionando opciones de personalización que permiten crear gráficos que comuniquen eficazmente la información.

Arte y creatividad en la visualización de datos

A diferencia de otras herramientas que vienen con gráficos predefinidos que limitan la creatividad del usuario, VDA permite una mayor personalización de los gráficos según las necesidades específicas de cada proyecto. De forma análoga, VDA promueve la integración del componente artístico como un elemento fundamental en la representación de datos, con el objetivo de captar la atención del lector, estimular la creatividad y subrayar la relevancia del diseño en la comunicación visual.

Así, se busca elevar tanto el estándar estético como el comunicativo de las gráficas, facilitando la creación de visualizaciones únicas que destaquen por su claridad e impacto.

Apoyo a estudiantes y profesionales

Recordemos que la biblioteca nace a raíz de la materia de Visualización de Datos Avanzada, con el propósito de ser un recurso práctico y flexible que acompañe a los estudiantes en su proceso de aprendizaje, facilitando el desarrollo de habilidades en visualización de datos. Sin embargo, ha evolucionado para adaptarse también a las necesidades del entorno profesional. De esta manera, VDA se presenta como un puente entre la teoría y la práctica, favoreciendo el desarrollo de habilidades tanto para futuros como para actuales científicos de datos, permitiendo que la herramienta sea útil tanto en el ámbito académico como profesional.

4. Alcance

El proyecto tiene como objetivo desarrollar una biblioteca de visualización de datos que abarque tanto gráficos esenciales como ejemplos avanzados. Se estima incluir 13 gráficos clásicos o fundamentales, tales como gráficos de barras, líneas, burbujas y mapas, que serán diseñados para cubrir las necesidades más comunes en la representación de datos. Además, se desarrollarán 2 ejemplos avanzados que combinarán elementos artísticos, matemáticos y técnicas de visualización, mostrando cómo la biblioteca puede ser utilizada para crear representaciones visuales únicas, personalizadas y de gran alcance.

La biblioteca será diseñada para ser funcional y accesible, con un enfoque en la facilidad de uso, la adaptabilidad y escalabilidad. Contará con una documentación completa y detallada que permitirá a los usuarios comprender cómo integrarla e implementarla en diferentes proyectos. Esta documentación incluirá una guía de usuario que describirá paso a paso el uso de la biblioteca, desde su instalación hasta la creación de visualizaciones personalizadas mediante casos de uso documentados, además de detallar las necesidades técnicas y de implementación requeridas.

Para garantizar la accesibilidad del proyecto, este se almacenará en un repositorio público de GitHub, donde estarán disponibles tanto el código fuente como la guía de usuario y los casos de uso. Estos ejemplos servirán como modelos para ilustrar las capacidades de la biblioteca en contextos reales, permitiendo a los usuarios adaptar la biblioteca a sus necesidades específicas.

5. Marco teórico

5.1. Análisis de herramientas disponibles

En el campo de la visualización de datos, existe una gran variedad de herramientas diseñadas para generar representaciones gráficas a partir de datos. Estas herramientas se pueden clasificar en dos categorías: herramientas de “propósito general”, como Tableau, y bibliotecas de “programación especializada”, como Plotly. Cada clasificación tiene sus propios objetivos, así como sus propias ventajas y limitaciones, lo que influye en su uso con dependencia de las necesidades de los usuarios.

Las herramientas de “propósito general” están diseñadas para un público amplio, con interfaces gráficas que permiten crear visualizaciones de forma rápida y sencilla sin necesidad de programación, sin embargo, suelen estar limitadas en términos de personalización y comúnmente requieren licencias con costo monetario.

Por otro lado, las bibliotecas de “programación especializada” como Plotly están orientadas a desarrolladores y científicos de datos. Este tipo de herramientas ofrecen un mayor grado de personalización y flexibilidad, no obstante, su uso puede representar un desafío para usuarios sin experiencia en programación, y su curva de aprendizaje suele ser más pronunciada.

Ambos enfoques, aunque efectivos, tienen limitaciones. Mientras que algunas herramientas tienden a ser rígidas en términos de personalización y estilos artísticos, las otras, pueden ser menos accesibles para usuarios sin habilidades técnicas avanzadas. Adicionalmente, en ambos casos, se presenta un problema de escalabilidad ya que no cuentan con un esquema o guía para poder ser modificadas o añadir funcionalidades especificas por el usuario de manera sencilla.

Ante este panorama, VDA pretende combinar la accesibilidad de las herramientas de “propósito general” con la flexibilidad y personalización de las bibliotecas, además de documentar y guiar al usuario para poder escalar la biblioteca personalizándola con nuevas funcionalidades y enfatizando el impacto artístico y visual de los gráficos.

A continuación, se presenta una breve descripción de algunas de las herramientas encontradas en el mercado.

Tableau

Tableau es una herramienta de “propósito general” que permite a los usuarios conectar, analizar y visualizar grandes volúmenes de datos de manera intuitiva. Es especialmente útil para generar dashboards interactivos.

Power BI

Desarrollado por Microsoft, es una herramienta que combina análisis de datos y visualización, también ofrece integración con otras

herramientas del ecosistema Microsoft, como Excel y Azure, facilitando la manipulación de datos y la creación de informes interactivos y visualizaciones gráficas.

Plotly

Es una biblioteca de código abierto para la creación de gráficos interactivos en Python, R y JavaScript. Es bastante flexible y capaz de generar visualizaciones avanzadas, incluyendo gráficos en 3D, mapas y gráficos científicos.

Seaborn

Es una biblioteca basada en Matplotlib que simplifica la creación de gráficos estadísticos en Python. Ofrece una interfaz amigable y un diseño visualmente atractivo, haciendo énfasis en gráficos estadísticos como distribuciones, relaciones y tendencias de datos.

Estas herramientas son potentes y ampliamente utilizadas, pero suelen estar diseñadas para casos específicos y presentan limitaciones. Mientras Tableau y Power BI son ideales para usuarios no programadores, su capacidad de personalización es limitada, especialmente en diseños artísticos. Por otro lado, Plotly y Seaborn ofrecen más flexibilidad a programadores, pero requieren conocimientos técnicos avanzados y no siempre facilitan la creación de gráficos estéticamente impactantes.

A continuación, se presenta una tabla comparativa de las herramientas anteriormente descritas, describiendo sus beneficios y limitaciones.

	Facilidad de uso	Usuario objetivo	Personalización	Escalabilidad	Enfoque	Costo
Tableau	De fácil uso e interfaz intuitiva.	No técnicos.	Ofrece un conjunto limitado de plantillas y opciones de personalización	No permite integrar o modificar su código.	Análisis de información y presentación “básica” de gráficos. Aunque generan gráficos atractivos, su enfoque principal es la funcionalidad.	Tiene un modelo de licencias por usuario.
Power BI	De fácil uso e interfaz intuitiva.	No técnicos.	Ofrece un conjunto limitado de plantillas y opciones de personalización	No permite integrar o modificar su código.	Análisis de información y presentación “básica” de gráficos. Aunque generan gráficos atractivos, su enfoque principal es la funcionalidad.	Incluye versiones gratuitas limitadas y opciones de pago.
Plotly	Es una solución más técnica que requiere de conocimientos de programación.	Orientada a programadores.	Brinda mayor control de estilos y configuraciones técnicas, pero su falta de documentación de código limita su adaptabilidad en casos específicos.	Modificar características suele ser complejo y requiere conocimientos avanzados de su arquitectura interna.	Provee más control técnico que las dos primeras, pero carece de un enfoque de impacto visual.	Tiene una versión de código abierto gratuita, pero su versión empresarial con características avanzadas es de pago.

	Facilidad de uso	Usuario objetivo	Personalización	Escalabilidad	Enfoque	Costo
Seaborn	simplifica la creación de gráficos mediante una interfaz de alto nivel.	Está dirigida a usuarios con conocimientos de programación	Facilita la creación de gráficos atractivos, pero su personalización es limitada y su documentación no siempre es detallada.	Modificar características suele ser complejo y requiere conocimientos avanzados de su arquitectura interna.	Ampliar sus funciones puede ser complicado debido a su arquitectura interna.	Seaborn es de código abierto y gratuito, sin costos de licencia.
VDA	Combina facilidad de uso con la flexibilidad necesaria para personalizar gráficos.	Accesible tanto para estudiantes en proceso de aprendizaje como para profesionales con experiencia en programación.	Permite una personalización completa, desde colores, tipografías, hasta formas y animaciones.	Su documentación y ejemplos documentados, así como su diseño modular la hacen escalable y adaptable.	Diseñada para destacar el aspecto artístico y el impacto visual en las representaciones gráficas, sin comprometer la claridad en la comunicación de los datos. Su principal objetivo es ofrecer un alto nivel de personalización en los gráficos, permitiendo adaptarlos a las necesidades específicas de cada usuario.	Es de código abierto y gratuita.

En conclusión, la biblioteca VDA, a diferencia de las herramientas previamente señaladas, se centra en ofrecer personalización y escalabilidad a nivel de código y funcionalidades. Además, reconociendo que el proyecto surge de una necesidad identificada por los alumnos de la Maestría en Ciencia de Datos, la biblioteca busca ser una solución accesible para quienes no poseen un amplio conocimiento en programación. Su propósito es facilitar la integración gradual de estos usuarios en el campo de la visualización de datos, al mismo tiempo que proporciona personalización, escalabilidad y un enfoque visual impactante, todo ello en una herramienta gratuita, intuitiva y versátil.

5.2. Importancia del impacto visual, el arte y la personalización de gráficos

En el ámbito de la visualización de datos, el impacto visual es fundamental para captar la atención de los usuarios y facilitar la comprensión de información compleja. Un gráfico bien diseñado no solo presenta datos de manera clara, sino que también crea una conexión visual que atrae y retiene la atención del observador. La biblioteca VDA reconoce esta necesidad al priorizar la creación de gráficos con un alto impacto visual, asegurando que cada visualización no solo comunique información, sino que también genere un efecto duradero en la audiencia.

El arte en la visualización de datos va más allá de la estética; es una herramienta esencial para potenciar la comunicación efectiva. La incorporación de elementos artísticos en los gráficos permite resaltar patrones, jerarquías y mensajes clave de forma más accesible e intuitiva. La biblioteca VDA se inspira en esta idea al ofrecer opciones para integrar componentes artísticos en las visualizaciones, creando gráficos que no solo informan, sino que también emocionan y comunican con mayor profundidad.

La personalización es otro aspecto clave en la visualización de datos, ya que las necesidades y contextos de los usuarios varían ampliamente. Las herramientas que ofrecen opciones limitadas a menudo obligan a los usuarios a adaptarse a plantillas predefinidas, lo que puede dificultar la representación precisa de los datos. VDA, por el contrario, pretende que los usuarios tengan el control total sobre sus gráficos, permitiendo modificar colores, formas, estilos y otros elementos clave para reflejar con exactitud sus ideas.

En conjunto, el impacto visual, el arte y la personalización se combinan para transformar gráficos comunes en herramientas poderosas de comunicación. VDA no solo busca cumplir con estos aspectos, sino trasladar esto a un nivel accesible, permitiendo que los usuarios, incluso aquellos sin experiencia en programación, puedan adentrarse paulatinamente a la biblioteca y llegar a crear representaciones visuales que sean claras, impactantes y personalizadas.

6. Diseño e implementación de la biblioteca

6.1 Herramientas y tecnologías utilizadas para la elaboración de la biblioteca

En el desarrollo de la biblioteca VDA se hizo uso de diversas herramientas de programación y presentación de contenido web, a continuación, se menciona cada una de ellas.

JavaScript

Es un lenguaje de programación ampliamente utilizado para el desarrollo web, reconocido por su versatilidad y capacidad para ejecutar código del lado del cliente en navegadores. Es la base para la creación de interfaces interactivas y aplicaciones dinámicas.

De acuerdo con Amazon Web Services (AWS), “JavaScript es un lenguaje de programación que los desarrolladores utilizan para hacer páginas web interactivas. Desde actualizar fuentes de redes sociales a mostrar animaciones y mapas interactivos, las funciones de JavaScript pueden mejorar la experiencia del usuario de un sitio web. Como lenguaje de scripting del lado del servidor, se trata de una de las principales tecnologías de la World Wide Web. Por ejemplo, al navegar por Internet, en cualquier momento en el que vea un carrusel de imágenes, un menú desplegable “click-to-show” (clic para mostrar), o cambien de manera dinámica los elementos de color en una página web, estará viendo los efectos de JavaScript.” [1] (AWS, s.f.).

1. Amazon Web Services (AWS). (s.f.). *¿Qué es JavaScript?* Recuperado el 23 de noviembre d 2024, de https://aws.amazon.com/es/what-is/javascript/

La elección de JavaScript se debe a su papel central en el desarrollo web y su capacidad para interactuar con Canvas, HTML y CSS. Además, permite integrar funcionalidades avanzadas, como la manipulación de datos y animaciones, esenciales para la biblioteca.

D3.js

D3.js (Data-Driven Documents) es una biblioteca de código abierto que permite crear gráficos interactivos y visualizaciones complejas utilizando estándares web como SVG, HTML y CSS.

D3 no es una biblioteca de gráficos en el sentido tradicional, ya que no tiene el concepto de 'gráficos' como tal; cuando se visualizan datos con D3, más bien se hace uso de una variedad de primitivas para generar un gráfico más complejo.

En este caso, D3 fue seleccionada por su potencia y flexibilidad en la creación de gráficos personalizados. Su enfoque en la manipulación directa del DOM y la conexión con datos hace que sea ideal para las necesidades de la biblioteca VDA.

HTML

HTML (HyperText Markup Language) es el lenguaje estándar para estructurar y presentar contenido en la web, funge como el esqueleto de las páginas web, definiendo la disposición de elementos como texto, imágenes y gráficos.

Entre los elementos de HTML, particularmente se hace uso de HTML5 Canvas, el cual permite renderizar gráficos, imágenes y animaciones de manera dinámica en una página web. Este componente proporciona un espacio en blanco sobre el cual es posible dibujar mediante programación, utilizando el contexto de dibujo de JavaScript, además es muy versátil, permitiendo crear desde gráficos simples como líneas y círculos hasta gráficos complejos y animaciones interactivas, lo que lo convierte en una herramienta muy poderosa para visualización de datos y gráficos interactivos.

Con base en los argumentos presentados, HTML es imprescindible para la construcción de la biblioteca, pues representa la base y estructura sobre la cual ésta se construye, haciendo posible la integración de visualizaciones en una página web.

CSS

CSS (Cascading Style Sheets) es el lenguaje de diseño para la personalización y colocación de estilos en páginas web mediante el control de aspectos como colores, tipografías, espaciado y disposición visual.

CSS complementa las capacidades de VDA al permitir estilos personalizados para las visualizaciones, lo que garantiza que los gráficos no solo sean funcionales, sino también estéticamente atractivos. Esta capacidad de personalización es clave para cumplir con el objetivo de fomentar el arte, el diseño y el impacto visual en la representación de datos.

6.2 Estructura del proyecto

El proyecto de la biblioteca VDA adopta una estructura modular diseñada para facilitar su mantenimiento, escalabilidad y comprensión, con lo cual, se pretende ofrecer una descripción de cada método proporcionado en la biblioteca, incluyendo su propósito, entradas, salidas y casos de uso.

Además, los casos de uso de ejemplo han sido diseñados e integrados de manera gradual en cuanto a su complejidad, permitiendo que el usuario adquiera un aprendizaje progresivo. De esta forma, se inicia con ejemplos básicos que ilustran el uso fundamental de la biblioteca y se avanza hacia aplicaciones más elaboradas que combinan múltiples funcionalidades.

A continuación, se describe la estructura de carpetas y archivos principales del proyecto, junto con su funcionalidad.

En la carpeta raíz se encuentra el archivo principal de la biblioteca, vda.js, el cual encapsula todos los métodos y el código necesario para su uso. Al mismo nivel, se ubica la carpeta data, que contiene diversas bases de datos en formato CSV utilizadas para los casos de uso. Además, en el mismo nivel, se organizan varias carpetas, cada una correspondiente a un caso de uso que ejemplifica la aplicación de los distintos métodos de la biblioteca, tal como se observa en la siguiente imagen.

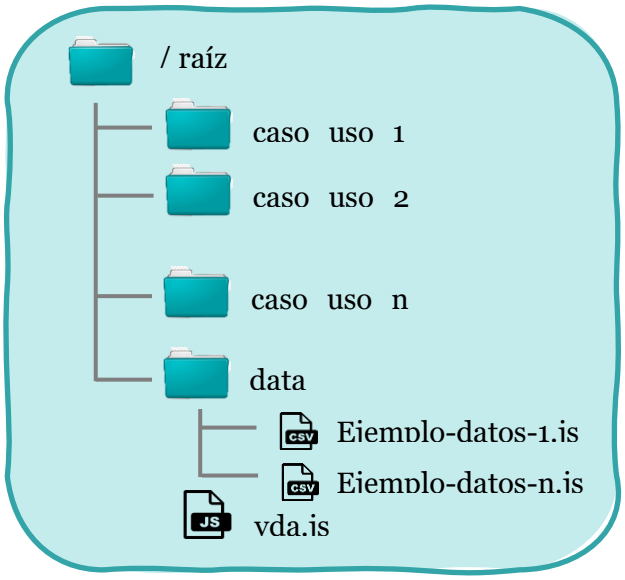


Ilustración 1. Arquitectura de carpetas de la biblioteca VDA

Cada carpeta de caso de uso, puede contener los siguientes archivos:

- Archivo HTML: Define la capa de presentación para el caso de uso específico.
- Archivo CSS: Contiene los estilos aplicados al archivo HTML para mejorar su diseño visual.
- Archivo JS: Gestiona la lógica del caso de uso, incluyendo la lectura de datos, su procesamiento y la generación de visualizaciones.

En la imagen siguiente podemos apreciar la arquitectura previamente descrita.

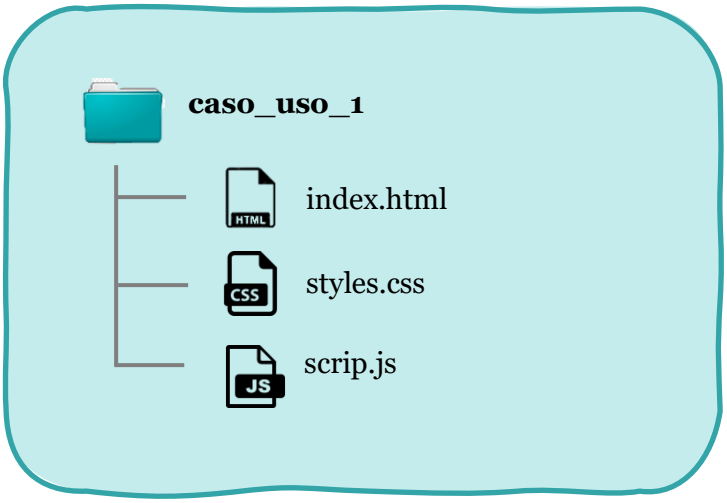


Ilustración 2. Arquitectura de carpetas de un caso de uso

6.3 Guía de usuario. Casos de uso

Se desarrolló una guía de usuario diseñada para facilitar el aprendizaje y uso efectivo de la biblioteca VDA. Esta guía aborda las configuraciones iniciales necesarias para integrar y ejecutar la biblioteca, además incluye una descripción detallada de cada uno de los métodos que la componen.

En la guía se documentan ampliamente los casos de uso diseñados para ilustrar las capacidades de la biblioteca. Cada caso de uso incluye:

- **Descripción de la implementación:** Se explica el caso de uso, y los métodos utilizados.

- **Datos de entrada y salida:** Se especifican los parámetros de entrada y salida para los métodos y los tipos de datos de dichas variables, asegurando que el usuario tenga un entendimiento completo de cada método.

- **HTML y JavaScript:** Se desglosa la estructura y contenido del archivo HTML utilizado para la capa de presentación, así como el código JavaScript que implementa la lógica necesaria para la visualización.

- **Referencias visuales:** Para cada caso de uso, se han incluido capturas de pantalla que muestran el resultado final en el navegador. Estas imágenes permiten al usuario identificar si la implementación ha sido exitosa y si los resultados cumplen con las expectativas planteadas.

- **Código documentado:** Adicionalmente, cada sección del código fuente está debidamente comentada para facilitar su comprensión, garantizando que los usuarios puedan modificar y ampliar la funcionalidad según sus necesidades.

La guía está diseñada pensando en la accesibilidad y claridad, haciendo uso de un lenguaje técnico pero comprensible, para que tanto usuarios con experiencia en programación como principiantes puedan beneficiarse de ella. Al cubrir cada método y funcionalidad de la biblioteca, se asegura que los usuarios tengan las herramientas necesarias para recrear los ejemplos propuestos y realizar sus propias visualizaciones personalizadas.

En el **Anexo. Extracto de la "Guía de usuario"** se presenta un extracto de la guía de usuario, en el cual se detallan las funcionalidades principales de la biblioteca y su aplicación en distintos escenarios. La guía completa, junto con el código necesario para implementar los casos de uso, así como los archivos de datos en formato CSV y GeoJSON utilizados pueden obtenerse directamente del siguiente **repositorio de GitHub** (https://github.com/aguilaral24/VDA).

7. Conclusiones

El desarrollo de este proyecto permitió la creación de una herramienta funcional y viable para la visualización de datos mediante representaciones gráficas, tanto estáticas como interactivas. Se logró cumplir con el objetivo principal de brindar una alternativa flexible y personalizable para la exploración de información, destacando la importancia de otorgar a los usuarios un mayor control sobre la manera en que pueden desarrollar y presentar sus datos.

Además, el proyecto cumple con el propósito de proporcionar una guía para que los usuarios puedan iniciarse en el desarrollo de gráficos más personalizables. A través de su estructura y ejemplos, ofrece un punto de partida accesible para quienes deseen explorar nuevas formas de representar información sin estar limitados por herramientas prediseñadas.

Durante el proceso de implementación surgieron diversas incógnitas y desafíos técnicos, los cuales abordé y resolví manteniendo siempre el enfoque en el objetivo inicial. La evolución del proyecto no solo me permitió encontrar soluciones a estos retos, sino que también abrió nuevas oportunidades de mejora y expansión.

La evolución del proyecto me impulsó en la integración de más ejemplos de los inicialmente propuestos, lo que enriqueció la herramienta y permitió demostrar su aplicabilidad en diferentes escenarios. Se estimaban 13 ejemplos base y se realizaron 14, mientras que en la categoría de ejemplos avanzados, se planearon 2 y finalmente se desarrollaron 4. A lo largo del desarrollo, no solo tomé en cuenta los aspectos técnicos, sino que también logré involucrar la parte creativa, permitiendo que el resultado final fuera mucho más expresivo y dinámico.

Adicionalmente, este proyecto representó una gran oportunidad de aprendizaje, pues me permitió profundizar y ampliar mis conocimientos en ciencia de datos, visualización de datos, matemáticas, metodología y programación. A través de los distintos retos técnicos y conceptuales, adquirí una mayor comprensión sobre la implementación de gráficos interactivos y su impacto en la representación de la información.

Finalmente, el resultado final fue más que satisfactorio, dejando abierta la posibilidad de futuras mejoras y expansiones, consolidando su viabilidad como una solución útil, eficiente y estéticamente atractiva dentro del ámbito de la visualización de datos.

8. Perfeccionamiento y oportunidades de Expansión

Si bien el proyecto ha cumplido con su objetivo, existen diversas oportunidades de mejora y expansión que podrían fortalecer su alcance y funcionalidad. Una posible mejora es la ampliación de la biblioteca, incorporando nuevas gráficos de visualización y mayor compatibilidad con diferentes formatos de datos. También, el empaquetado y la modularización de la biblioteca permitirían una integración más sencilla en otros proyectos, facilitando su uso por parte de desarrolladores con distintos niveles de experiencia, así como el mantenimiento de la biblioteca.

Otra mejora importante sería el fortalecimiento del manejo de errores dentro de la biblioteca, con mensajes más descriptivos y detallados que ayuden a los desarrolladores a identificar y solucionar problemas de manera más eficiente. Implementar validaciones más robustas y mensajes de error específicos permitiría reducir la curva de aprendizaje y mejorar la experiencia de desarrollo, asegurando que cualquier problema sea identificado y comprendido rápidamente.

Adicionalmente, se pretende implementar esta biblioteca como una herramienta de trabajo y guía dentro de la próxima versión de la clase de Visualización de Datos en el ITAM, la cual será impartida por el doctor José Ezequiel Soto Sánchez. Esta implementación permitirá evaluar su aplicabilidad en un entorno académico. Como parte de esta implementación, se planea realizar una encuesta para conocer la percepción de los usuarios sobre la facilidad de uso y posibles mejoras de la biblioteca, lo que brindará información valiosa para su evolución futura.

9. Bibliografía

1. D3.js. (s.f.), What is D3?, recuperado el 23 de noviembre del 2024 de *https://d3js.org/what-is-d3*

2. Mozilla Developer Network (s.f.), Conceptos básicos de HTML, recuperado el 23 de noviembre del 2024 de *https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/HTML_basics*

3. Wikipedia contributors. (s.f.), Canvas element, recuperado el 23 de noviembre de 2024 de *https://en.wikipedia.org/wiki/Canvas_element*

4. GeeksforGeeks (s.f.), CSS Introduction, recuperado el 23 de noviembre de 2024 de *https://www.geeksforgeeks.org/css-introduction/*

5. ClickUp (s.f.). The 15 best data visualization tools in 2023, recuperado el 30 de noviembre de 2024, de *https://clickup.com/blog/data-visualization-tools/*

6. QuestionPro (s. f.). Herramientas de visualización de datos, recuperado el 30 de noviembre de 2024 de *https://www.questionpro.com/blog/es/herramientas-de-visualizacion-de-datos/*

Anexo. Extracto de la “Guía de usuario”

La guía está dividida principalmente en dos secciones. La primera sección abarca la descripción de los métodos disponibles en la biblioteca, detallando su funcionalidad, así como los parámetros de entrada y salida necesarios para su correcto uso. La segunda sección presenta diversos casos de uso, los cuales consisten en ejemplos prácticos que el usuario puede replicar para facilitar su aprendizaje. En estos ejemplos se describe con detalle su implementación, explicando paso a paso cómo integrar uno o varios métodos de la biblioteca para la construcción de un gráfico específico.

Los siguientes ejemplos pertenecen al apartado de métodos.

View Port

Uso

El método initViewport se encarga de inicializar un espacio de trabajo en un elemento canvas. Este método ajusta el tamaño 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)
{

    ...

 }
```

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.
- **csvFilePath** (String): Ruta al archivo de datos.
- **xColumnName** (String): Nombre para la columna X
- **yColumnName** (String): Nombre para la columna Y.
- **infoColumnNames** (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.
- **filledCircles** (Boolean): True para círculos rellenos.
- **pointRadius** (number): Radio de los puntos.
- **canvasPadding** (number): Padding del canvas.
- **axesProperties** (Object, opt): Objeto que contiene las propiedades definidas para los ejes (Ver caso práctico).

Salidas

No retorna valor.

Método { }

```
drawDotPlot (canvas, context,
csvFilePath, xColumnName,
yColumnName, infoColumnNames, color,
filledCircles, pointRadius,
canvasPadding, axesProperties)
{
      ...
 }
```

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.
- **filePath** (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.
- **axesProperties** (Object, opt): Objeto que contiene las propiedades definidas para los ejes (Ver caso práctico).

Salidas

No retorna valor.

Método { }

```
drawBubblePlot (canvas, context, filePath,
xColumnName,yColumnName, sizeColumnName,
minSize, maxSize, infoColumnNames,  color,
canvasPadding, axesProperties)

 {    ...   }
```

Mapa de puntos

Uso

El método drawMapDotPlot 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
- **filePath** (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.
- **filledCircles** (Boolean): True para círculos rellenos.
- **pointRadius** (number): Radio de los puntos.
- **canvasPadding** (number): Padding del canvas.

Salidas

No retorna valor.

Método { }

```
drawMapDotPlot(canvas, context,
mapDataFile, filePath, xColumnName,
yColumnName, infoColumNames, color,
filledCircles, pointRadius,
canvasPadding)

{      ...      }
```

Los siguientes ejemplos pertenecen al apartado de casos de uso.

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 loadCSV 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.

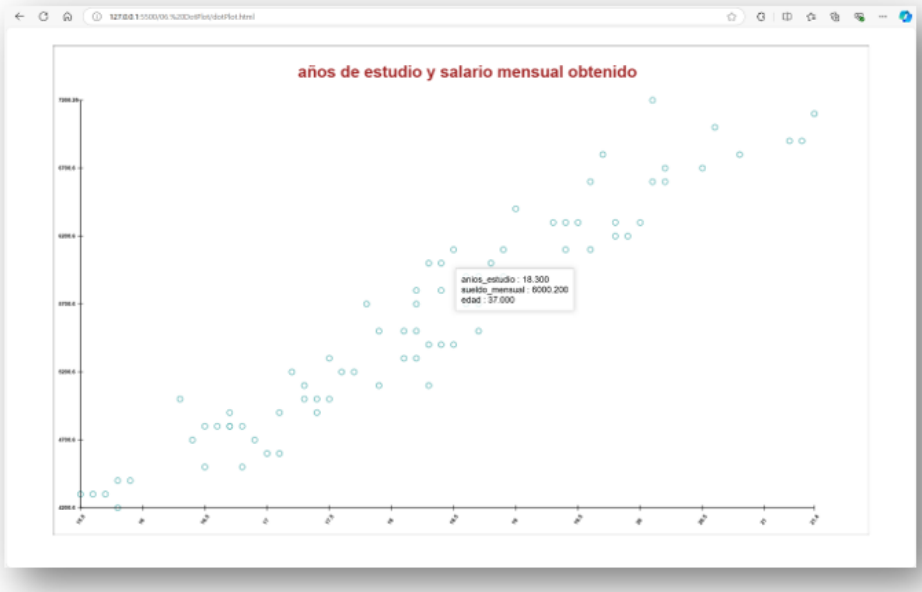
Main.js

```
Import {initViewport,drawDotPlot,drawText,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 filledCircles = false;
  const pointRadius = 5;
  const canvasPadding = 50;

  //Parametros para los ejes
  const axesProperties = {
    xPos: 0,
    yPos: 0,
    xLabelSpace: 20,
    yLabelSpace: -10,
    xLabelTextAngle: 55,
    color: "black",
    xAxeType: 0.5,
    yAxeType: 500
  };

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

  const xPos =  width/2 - 6*canvasPadding ;
  const yPos = height -canvasPadding;
  const titleSize = 30;
  const titleColor = "brown"
  drawText(context,"años de estudio y salario mensual obtenido", xPos, yPos,
titleSize ,titleColor,- 0)
});
```

Nótese que se hace uso del objeto axesPorperties, el cual contiene los valores necesarios para dibujar los ejes X y Y del gráfico.

En la siguiente imagen podemos apreciar el gráfico de puntos 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:

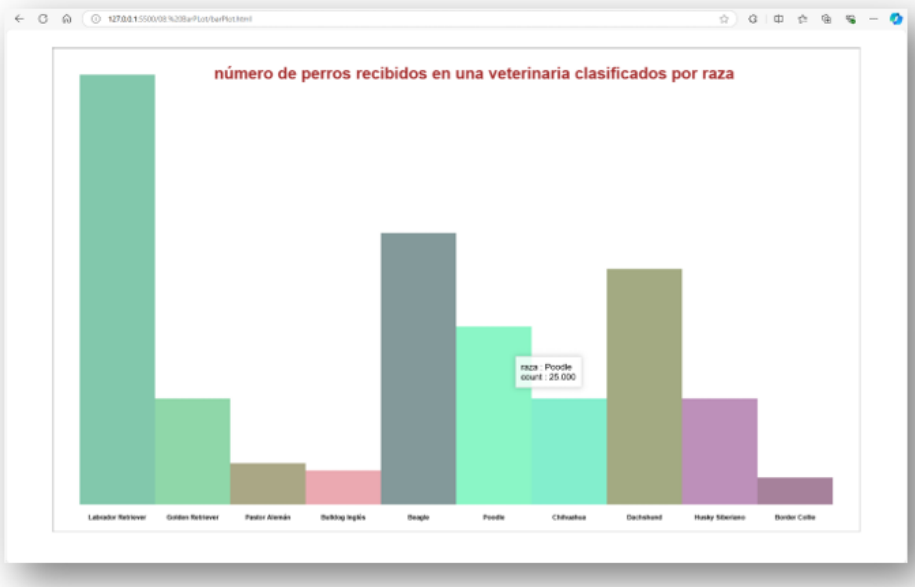
- Se importa los métodos initViewport , drawBarPlot y loadCSV desde el archivo main.js, y opcionalmente, drawXAxisFromArray, drawYAxisFromArray.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Llamamos a drawBarPlot con los parámetros necesarios para el gráfico.

Main.js

```
import { initViewport, drawBarPlot, drawText, 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);

  const xPosTitle =  width/2 - 9*canvasPadding ;
  const yPosTitle = height - canvasPadding;
  const titleSize = 30;
  const titleColor = "brown"

  drawText(context,"número de perros recibidos en una veterinaria
clasificados por raza", xPosTitle, yPosTitle, titleSize
,titleColor,- 0)

});
```

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.

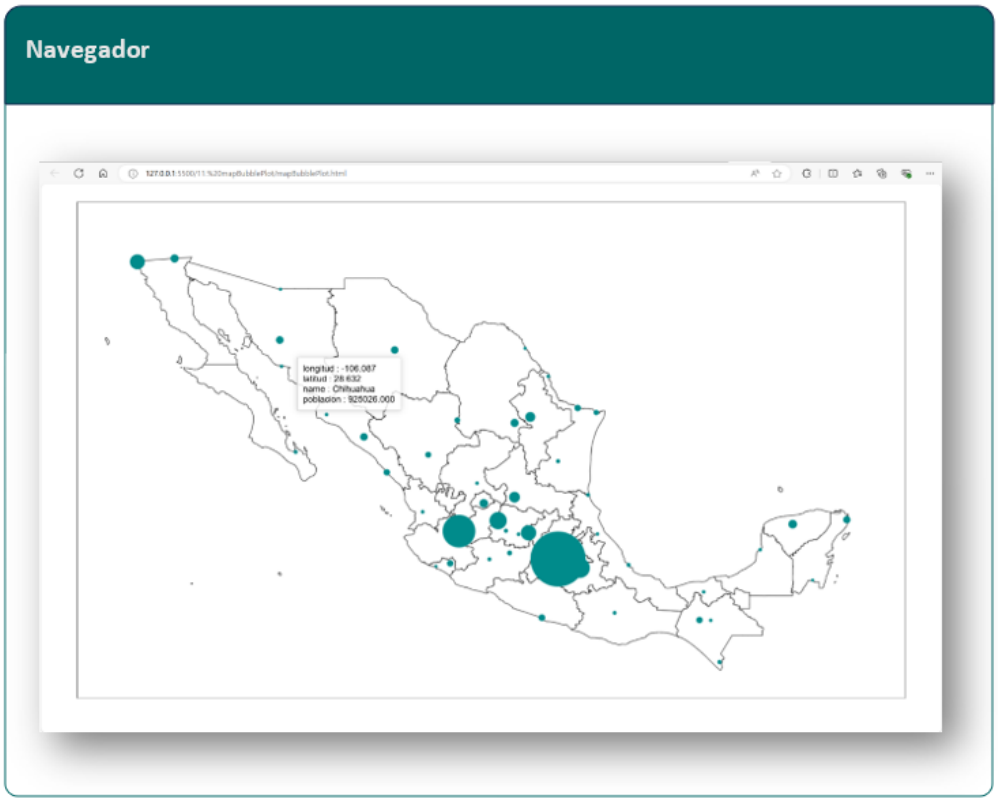
Posteriormente:

- 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, drawText, 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)

  const xPos =  width/2 - 8*canvasPadding ;
  const yPos = height -canvasPadding;
  const titleSize = 30;
  const titleColor = "brown"
  const xPosSubtitle = width/2 - 3.5*canvasPadding ;
  const yPosSubtitle = height - 2*canvasPadding;
  const subtitleSize = 15;

  drawText(context,"Algunas ciudades de méxico y su localización geográfica", xPos,
yPos, titleSize ,titleColor,- 0)
  drawText(context,"(El tamaño de burbuja equivale al número de habitantes)",
xPosSubtitle, yPosSubtitle, subtitleSize ,titleColor,- 0)

});
```

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



Serie de tiempo

En este ejemplo se hace uso de una serie de eventos cronológicos. Se dibuja una colección de puntos con base en el conjunto de fechas dadas en el formato "yyyy-mm-dd" y un valor asociado a cada una de ellas. El valor en las abscisas será definido por la fecha, mientras que el valor en las ordenadas estará definido por el valor asociado a cada fecha; el archivo con estos datos puede obtenerse de la carpeta data time_series_100.csv del repositorio de GitHub.

Mediante el método drawTimeSeries podremos dibujar las líneas que conectan los puntos en la serie de tiempo, haciendo uso de curvas de Bezier. Las curvas de Bezier son un tipo de curva paramétrica ideal para modelar formas suaves, se definen mediante un conjunto de puntos de control y son extremadamente flexibles.

Existen curvas de Bezier lineales, cuadráticas y cúbicas; la librería VDA hace uso de curvas de Bezier cuadráticas para dibujar las líneas entre los puntos.

Como se puede ver en el código del archivo mai.js, utilizaremos el método bezierCurveTo, el cual es parte de la API de dibujo en canvas de HTML5, el cual se define de la siguiente manera:

bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y)

donde:

1. cp1x, cp1y: Coordenadas del primer punto de control
2. cp2x, cp2y: Coordenadas del segundo punto de control
3. x, y: Coordenadas del punto final

Notar que el punto inicial es el último punto que se dibujó.

timeSeriesPlot.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 línea de tiempo</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>
    <div id="eventToolTip" style="position: absolute; display: none;
background: rgba(0, 0, 0, 0.7); color: white; padding: 5px; border-radius:
5px;"></div>
    <script type="module" src="./main.js"></script>
</body>
</html>
```

Adicionalmente, el método permite agregar una serie de bandas de tiempo para enfatizar eventos dentro de la serie de tiempo. Estos eventos son enviados como un objeto de array al método, tal como puede observarse en el código del archivo main.js

Para visualizar los eventos, será necesario incluir la siguiente etiqueta en el archivo timeSeriesPlot.html

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

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

Finalmente, en el archivo main.js:

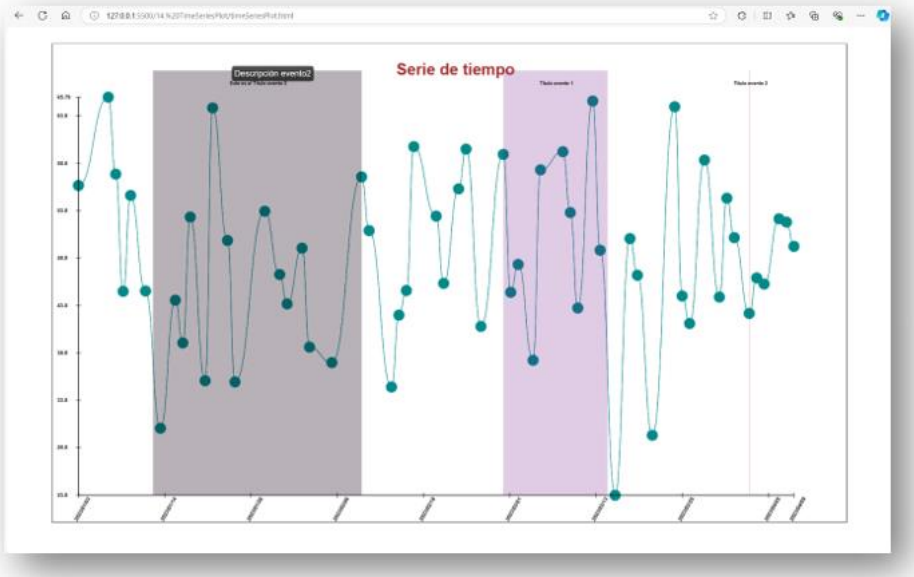
- Se importan los métodos initViewport y drawTimeSeries desde main.js.
- Inicializamos el canvas y su contexto mediante el método initViewport().
- Proporcionamos el archivo CSV que contiene la serie de tiempo.
- Definimos parámetros para la gráfica, ejes en el plano y array de eventos.
- Llamamos a drawTimeSeries para dibujar el gráfico.

Main.js

```
import { initViewport, drawTimeSeries, drawText, 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/time_series_100.csv";
  const data = await loadCSV(filePath);

  //Parametros para la gráfica
  const canvas = document.getElementById("miCanvas");
  const xColumnName = "date";
  const yColumnName = "value";
  let infoColumnNames = [xColumnName, yColumnName];
  const color = "darkcyan";
  const filledCircles = true; const pointRadius = 10;
  const lineWidth = 1;
  const canvasPadding = 50;

  //Parametros para los ejes
  const axesProperties = {
    xPos: 0,
    yPos: 0,
    xLabelSpace: 2,
    yLabelSpace: -10,
    xLabelTextAngle: 60,
    color: "black",
    xAxeType: -1,
    yAxeType: -1
  };

  //Eventos definidos para la serie de tiempo
  const events = [
    { startDate: '2023-01-13', endDate: '2023-02-10', title: 'Título evento 2', desc:
'Descripción 2', color: 'rgba(20, 0, 20, 0.3)' },
    { startDate: '2023-03-01', endDate: '2023-03-15', title: 'Título evento 1', desc:
'Descripción 1', color: 'rgba(105, 0, 123, 0.2)' },
    { startDate: '2023-04-03', endDate: '2023-04-03', title: 'Título evento 2', desc:
'Descripción 2', color: 'rgba(220, 0, 20, 0.3)' }
  ];

  // Lamada a la función para dibujar la serie de tiempo
  drawTimeSeries(canvas, context, filePath, xColumnName,yColumnName, infoColumnNames, color,
filledCircles, pointRadius, lineWidth, canvasPadding, axesProperties,events);

  const xPos =  width/2 - 2*canvasPadding ;
  const yPos = height -canvasPadding;
  const titleSize = 30;
  const titleColor = "brown"
  drawText(context,"Serie de tiempo", xPos, yPos, titleSize ,titleColor,- 0)
});
```

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

Navegador



Mapa de burbujas interactivo (incendios)

Este ejemplo tiene como objetivo crear un **mapa interactivo** que visualice los incendios ocurridos entre 2015 y 2021. El mapa utiliza burbujas para representar los incendios, donde el tamaño de cada burbuja es proporcional al total de hectáreas afectadas por el incendio. Además, el impacto del incendio y el tipo de incendio están representados mediante colores específicos, lo que permite a los usuarios comprender visualmente la magnitud y características de cada evento.

El mapa cuenta con los siguientes controles interactivos:

1. **Control de Año (Deslizador):**
    - Es posible seleccionar un año entre 2015 y 2021 utilizando un control deslizante. Se actualizará el mapa para mostrar los incendios correspondientes al año seleccionado.
2. **Control de Impacto (Checkboxes):**
    - Se ofrecen tres opciones de impacto (mínimo, moderado, severo) mediante casillas de verificación. Es posible seleccionar uno o más tipos de impacto, y el mapa se actualizará para mostrar solo los incendios que coincidan con los tipos seleccionados.

Datos utilizados en el mapa: Los datos que alimentan el mapa provienen de un archivo CSV que contiene las siguientes columnas:

- **Anio**: Año del incendio.
- **Latitud y Longitud**: Ubicación geográfica del incendio (para posicionar la burbuja en el mapa).

- **Tipo_incendio**: Tipo de incendio (de copa, superficial, subterráneo, etc.). Este valor se usa para definir el color del contorno de la burbuja.
- **Tipo_impacto**: Tipo de impacto (mínimo, moderado, severo). Este valor se usa para definir el color de relleno de la burbuja.
- **Total_hectareas**: Tamaño del incendio en hectáreas. Este valor determina el tamaño de la burbuja en el mapa.

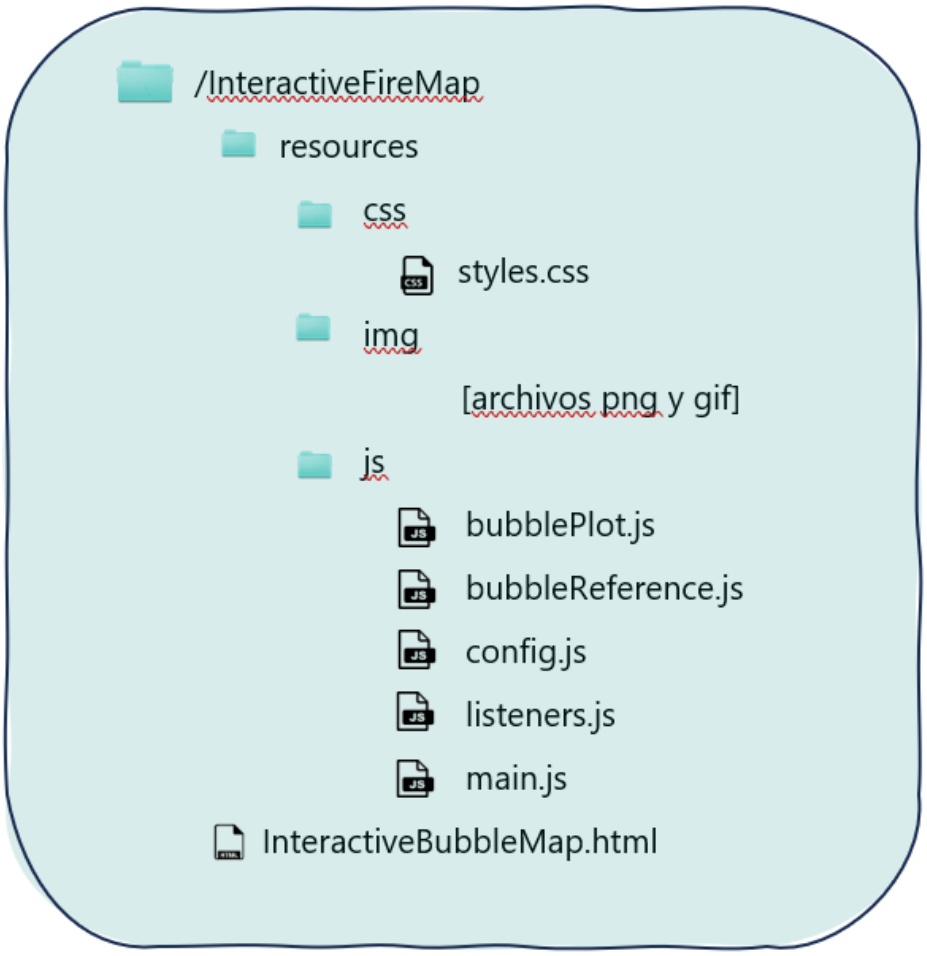
Referencias:

1. **Referencia Dinámica de Tamaño**:
    - Cada burbuja en el mapa tendrá una referencia dinámica que permite comparar el tamaño del incendio con dimensiones conocidas, como el tamaño de Ciudad Universitaria, el Auditorio Nacional, o el Estadio Azteca. Esto ayudará a los usuarios a visualizar la magnitud del incendio en términos más familiares.
2. **Referencia Estática de Colores**:
    - Se incluirá una leyenda estática en el mapa que explique el significado de los colores de las burbujas (relacionados con el impacto) y el color del contorno de las burbujas (relacionados con el tipo de incendio). Esto facilitará la interpretación visual de los datos y mejorará la experiencia del usuario.

Interacción del Mapa: El usuario podrá interactuar con el mapa mediante los controles de impacto y de año. Al realizar una selección en cualquiera de estos controles, el mapa se actualizará dinámicamente para reflejar solo los incendios que coincidan con los filtros seleccionados.

Anteriormente, se mostró un mapa de burbujas, por lo que usaremos dicho ejemplo y código como base para este nuevo ejemplo, añadiendo los controles interactivos para los filtros. El código fue dividido en varios archivos .js con el fin de hacerlo más entendible y modular.

La estructura del proyecto es la siguiente:

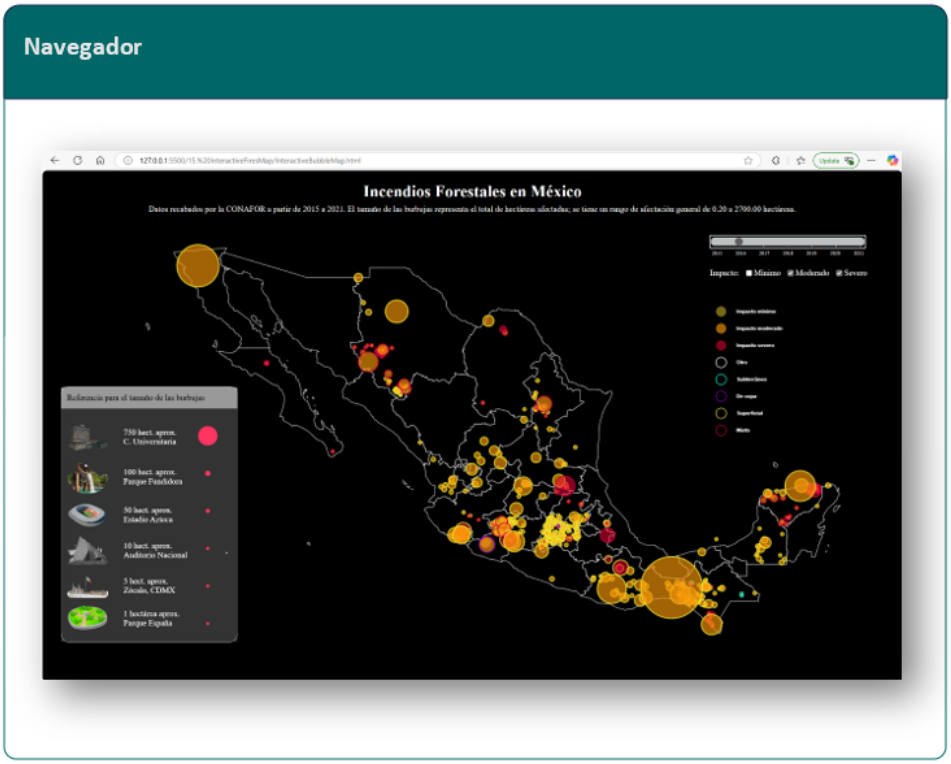


index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <!-- Meta Información -->
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Incendios Forestales en México</title>

    <!-- Estilos -->
    <link rel="stylesheet" href="resources/css/styles.css">

    <!-- Scripts Externos -->
    <script src =
"https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js">
    </script>
    <script src="https://d3js.org/d3.v7.min.js"></script>
</head>
<body>
    <div class="container">
        <!-- Título y Descripción del Mapa -->
        <h1 id="mapTitle">Incendios Forestales en México</h1>
        <p id="mapDesc">
            Datos recabados por la CONAFOR a partir de 2015 a 2021.
            El tamaño de las burbujas representa el total de hectáreas
            afectadas; se tiene un rango de afectación general de
            <span id="minRange">0.005</span> a
            <span id="maxRange">23,809</span> hectáreas.
        </p>

        <!-- Canvases para el Mapa y las Burbujas -->
        <canvas id="mapCanvas"></canvas>
        <canvas id="bubbleCanvas"></canvas>

        <!-- Superposición de Información -->
        <div id="infoOverlay" class="info-overlay"></div>

        <!-- Referencia de Colores para las Burbujas -->
        <canvas id="bubbleColorReferenceCanvas"></canvas>
    </div>
```

Este es un ejemplo extenso que involucra múltiples configuraciones y archivos, por lo que se omite su implementación detallada en este documento. No obstante, el ejemplo completo, junto con su configuración y archivos correspondientes, puede consultarse en la guía de usuario.

A continuación, se presenta el resultado final.



Árbol de adopciones

El árbol de adopciones es un caso específico que ejemplifica cómo la biblioteca VDA permite explorar el factor artístico en la representación de datos, incentivando la creatividad y la libertad visual. Con este ejemplo, se busca inspirar a los usuarios a descubrir nuevas formas de representar información, trascendiendo los enfoques tradicionales y explorando visualizaciones más expresivas y personalizadas.

Esta visualización muestra el número de adopciones registradas entre 2015 y 2023, e incorpora un control interactivo que permite seleccionar el año de visualización y elegir entre una vista acumulada o individual.

La inspiración detrás de este gráfico proviene de la obra Almendro en flor de Vincent Van Gogh. El árbol simboliza las conexiones familiares y el crecimiento a través de la adopción. Su estructura crece de forma exponencial, con ramas que se dividen sucesivamente en dos. Cada flor representa a un niño que se reintegra a una familia mediante la adopción. Para su diseño, se utilizaron pentágonos de Sierpinski, un fractal generado a partir de subdivisiones sucesivas de un pentágono, lo que da como resultado un patrón armónico y dinámico.

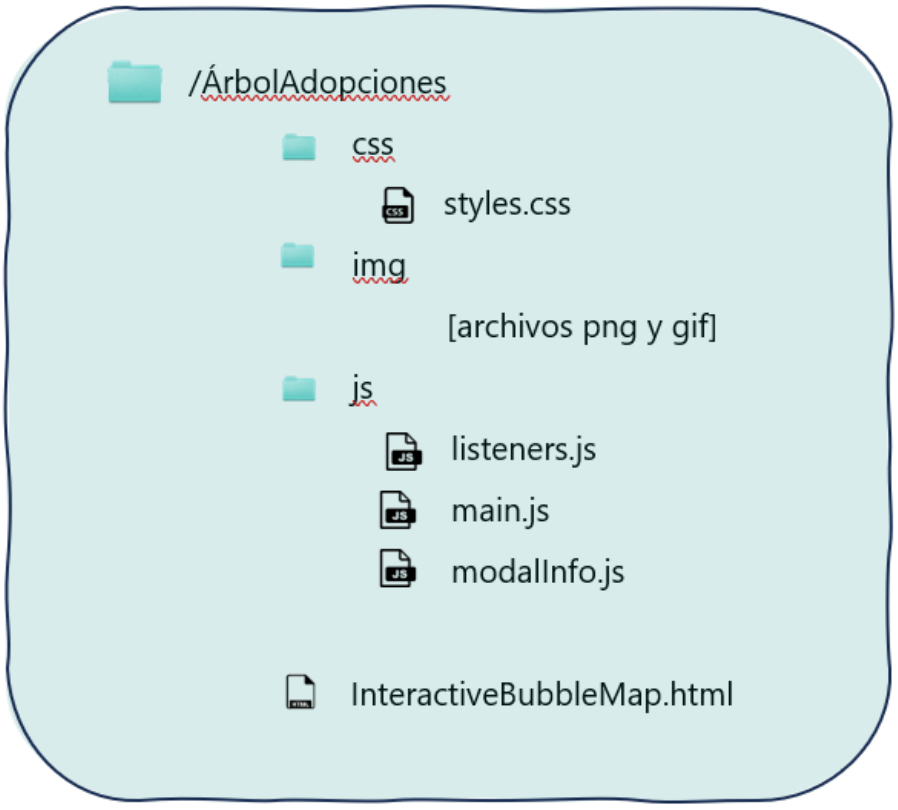
Además, el color juega un papel fundamental para distinguir visualmente a los niños y niñas adoptados, facilitando la interpretación de los datos de forma intuitiva.

La interacción con el árbol se realiza mediante diferentes controles:

1. Un control deslizante en la parte superior izquierda para seleccionar el año de visualización.
2. Un checkbox que permite alternar entre la vista acumulada de adopciones y la del año seleccionado.
3. Dos marcadores en la parte derecha que muestran el conteo total de adopciones.

Este proyecto resalta cómo la combinación de disciplinas puede generar experiencias visuales significativas mientras que el factor artístico mejora la narrativa. Los elementos artísticos como el árbol y las flores generan una conexión emocional con el espectador, de esta manera, el gráfico no solo comunica estadísticas, sino que también resalta las historias humanas detrás de los números.

La estructura del caso de uso es la siguiente:



Se explicará primeramente el archivo html y posteriormente las dependencias a los archivos java script.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Adopciones de niños en México</title>

    <!-- Enlace al archivo CSS para estilos -->
    <link rel="stylesheet" href="css/styles.css">

    <!-- Scripts Externos -->
      <script src =
      "https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.j
      s">  </script>
    <script src="https://d3js.org/d3.v7.min.js"></script>
</head>

<body>
    <!-- Contenedor del modal de descripción -->
    <div class="overlay" id="overlay">
        <!-- Botón para cerrar el modal -->
        <span class="closeModal" id="close">&times;</span>

        <!-- Descripción del propósito del gráfico -->
        <p>
            <span style="font-weight: bold;">Árbol familiar</span>
        </p>
        <p>El gráfico que verás a continuación representa un árbol familiar
      inspirado en la pintura de Vincent Van Gogh "Almendro en flor".</p>
        <p>La información presentada está relacionada con adopciones de niños
      y niñas realizadas en toda la República Mexicana.</p>
        <p>Puedes elegir el año con el control deslizante, además de decidir
      si deseas observar datos específicos de un año o datos acumulados hasta
      cierto año.</p>
```

Este es un ejemplo extenso que involucra múltiples configuraciones y archivos, por lo que se omite su implementación detallada en este documento. No obstante, el ejemplo completo, junto con su configuración y archivos correspondientes, puede consultarse en la guía de usuario.

A continuación, se presenta el resultado final.