

27-11-2025

Actividad 1

Visualización Interactivas con datos
JSON y D3.js

Leonard José Cuenca Roa
UNIR – UNIVERSIDAD RIOJA DE ESPAÑA

Contenido

Descripción Breve del Desarrollo	2
Desarrollo Frontend	2
Estructura Front	2
.....	2
Resultados de pantallas	3
Desarrollo Backend	4
Estructura Backend	5
.....	5
Interpretación de Datos	11
Insights Detectados	11
Análisis de Resultados (La Narrativa)	12
Conclusiones	13

Descripción Breve del Desarrollo

La actividad actual tiene como objetivo demostrar las habilidades adquiridas en el desarrollo de gráficas, utilizando dos de las librerías principales del mercado: **Google Charts** y **D3.js**.

Para este proyecto, se generó una arquitectura modular basada en buenas prácticas de desarrollo, buscando fomentar una mayor flexibilidad y escalabilidad.

A continuación, se describirán los puntos más resaltantes del desarrollo. Sin caer en la retórica de explicar cada línea de código, se desglosará para mayor comprensión qué se realizó en el **Frontend** y en el **Backend**, detallando tanto el "qué" (la funcionalidad) como el "cómo" (la implementación).

Desarrollo Frontend

Para el desarrollo del Frontend y para su evaluación se mostrará una captura de pantalla que permita visualizar el resultado final:

Estructura Front

```
/mi-dashboard
├── node_modules/      <-- Paquetes Funcionales para el ambiente Node.js
├── package.json        <-- Lista de Paquetes para la actividad
├── package-lock.json
├── server.js           <-- Servidor Express (Backend)
└── public              <-- Repositorio Principal (Presentación)
    ├── index.html       <-- Plantilla html que permite la visualización
    ├── app.js            <-- Script Interactivo (Orquestador Cliente)
    └── js
        ├── google-chart.js <-- Lógica para graficar google chart (Grafico 1)
        ├── d3-chart.js      <-- Lógica para graficar la funcionalidad de grafica deburbuja (Grafico 2)
        └── util.js          <-- Lógica modular y flexible permitiendo tener calculos, validaciones, llamados API, para mantener la modularidad y flexibilidad. (Utilidades)
    └── css
        ├── global.css      <-- Archivo de limpieza csss
        └── dash_venta.css   <-- Archivo para generar el estilo deseado visual para la actividad
```

En la imagen actual destaco la organización de los archivos CSS para el desarrollo visual del dashboard. Me apoyo en dos elementos clave:

global.css: Este elemento me permite reiniciar las etiquetas HTML con el propósito de tener un control total sobre los estilos nativos, ya que los diferentes tipos de navegadores aplican estilos predeterminados. [\[consultar código fuente aquí\]](#)

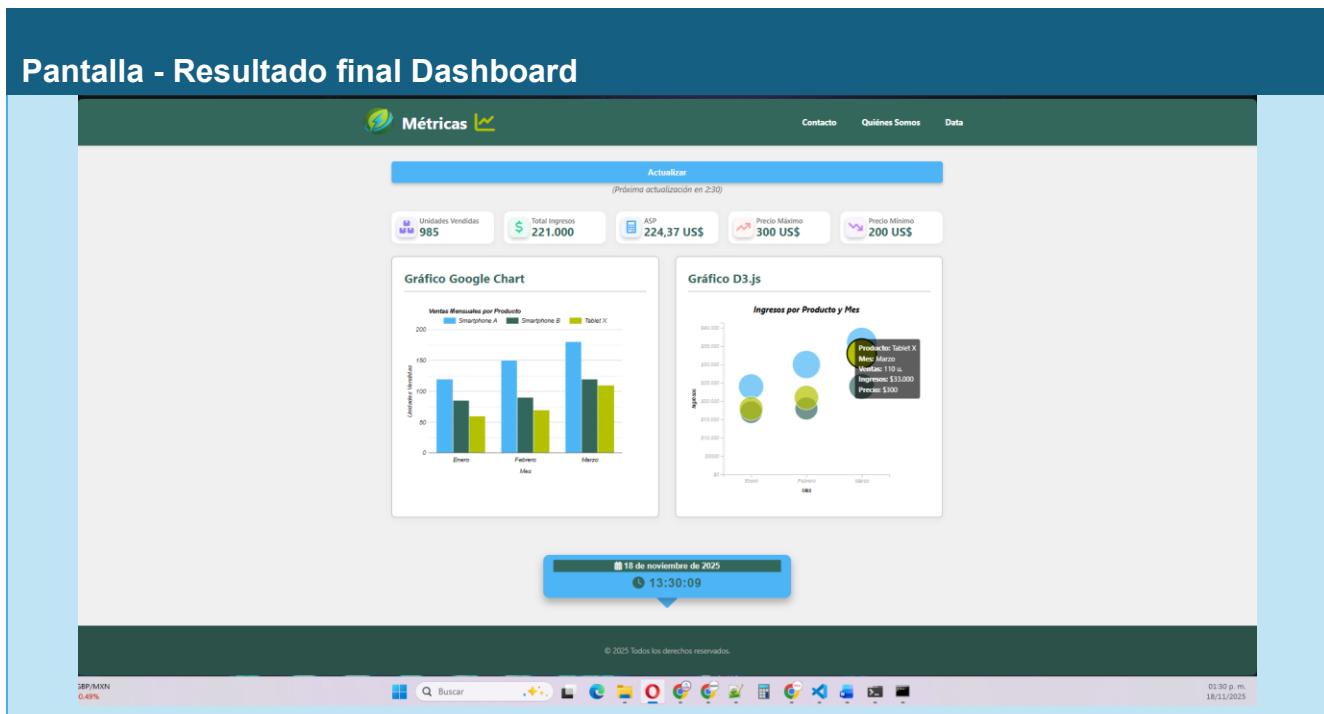
dash_ventas.css: Por medio de este archivo logro generar las reglas de cada estilo para la composición visual del dashboard. Aquí defino clases para estructurar los bloques HTML (**header, body y footer**), con el fin de mantener un código organizado y homogeneizado.

Esto garantiza flexibilidad, escalabilidad y facilidad de mantenimiento, cumpliendo con las buenas prácticas de desarrollo y los requisitos de la actividad. [\[consultar código fuente aquí\]](#)

El index.html (La Presentación)

Sin dejar de lado el index.html que es la joya de la corona, este es el otro orquestador, donde se enlazan los diferentes métodos y llamadas a librerías para generar toda la magia visual, funcionando bajo un ambiente **Cliente/Servidor**. [consultar código fuente aquí], en la sección de **Backend** se detallará mas en detalle.

Resultados de pantallas



Como se puede apreciar en la captura de pantalla, el desarrollo cumple con los requisitos descritos en la actividad, creando un espacio visual central que contiene dos gráficos clave:

- **Primer Gráfico (Barras):** Representa las ventas del primer trimestre del año, segmentando las ventas de los tres principales productos por mes. Esto permite una comparativa rápida y clara del rendimiento de cada producto a lo largo del primer trimestre.
- **Segundo Gráfico (Burbujas):** Se implementó para ofrecer interactividad. Al pasar el cursor (efecto hover) sobre el área de cada burbuja, se logra visualizar el detalle específico de la información como el **producto, mes, ventas, ingresos, precio**.

Funcionalidades Adicionales Implementadas:

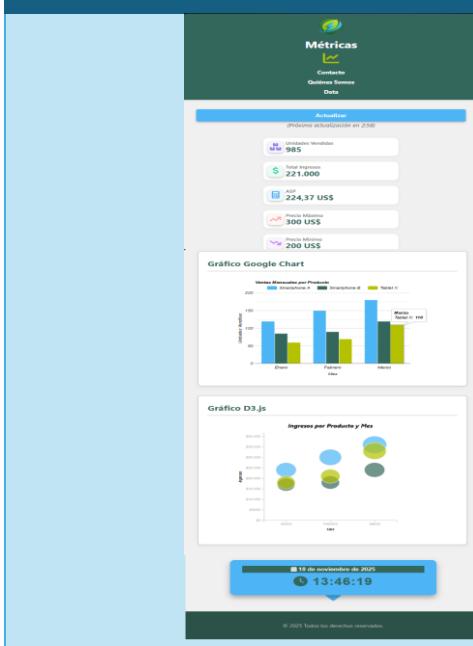
Se agregó un botón que permite la actualización manual de los gráficos en caso de que exista un nuevo valor en el archivo JSON.

Como valor agregado extra, se generó un contador de tiempo regresivo de 5 minutos. Al llegar a cero, el sistema realiza un **refresh automático** de las gráficas para mostrar los datos más recientes.

Se agregó también un resumen de métricas clave, una forma práctica de resaltar cantidades importantes. Para este caso, se resaltaron las siguientes métricas:

- Total de Unidades Vendidas
- Total de Ingresos
- ASP (Precio Promedio de Venta)
- Precio Máximo
- Precio Mínimo

Pantalla - Resultado Responsivo



Como se puede apreciar en esta imagen, se realiza la validación de que el dashboard cumple con su potencial de adaptabilidad o mejor conocido diseño responsivo, ofreciendo la oportunidad de que el dashboard pueda ser consumido en dispositivos móviles como celulares o tabletas, sin perder la funcionalidad principal y conservar una buena experiencia de usuario.

Desarrollo Backend

Para el desarrollo del **Backend** empleé la tecnología **Node.js**. Esta plataforma me permitió utilizar **JavaScript** de forma **Full Stack**, abarcando tanto el ambiente del servidor como el ambiente del cliente. Específicamente, utilicé el entorno del servidor para la validación del JSON y para el despliegue de un **API Endpoint** que será consumido desde el cliente. En el cliente, se desarrolló la interfaz de usuario y toda la lógica de presentación con **HTML, CSS y JavaScript**.

Estructura Backend

```
/mi-dashboard
  ├── node_modules/      <-- Paquetes Funcionales para el ambiente Node.js
  ├── package.json        <-- Lista de Paquetes para la actividad
  ├── package-lock.json
  ├── server.js          <-- Servidor Express (Backend)
  └── public              <-- Repositorio Principal (Presentación)
    ├── index.html        <-- Plantilla html que permite la visualización
    ├── app.js             <-- Script Interactivo (Orquestador Cliente)
    ├── /js
    │   ├── google-chart.js <-- Lógica para graficar google Chart (Grafico 1)
    │   ├── d3-chart.js     <-- Lógica para graficar la funcionalidad de grafica deburbuja (Grafico 2)
    │   └── util.js         <-- Lógica modular y flexible permitiendo tener calculos, validaciones, llamados API, para mantener la modularidad y flexibilidad. (Utilidades)
    └── /css
        ├── global.css      <-- Archivo de limpieza css
        └── dash_venta.css   <-- Archivo para generar el estilo deseado visual para la actividad
```

El `server.js` (Backend)

El archivo `server.js`, utilizando el **paquete Express**, es fundamental para configurar y crear el servidor que permite desplegar y operar todo el entorno de la aplicación. El archivo JSON con los datos se encuentra alojado en una sección del repositorio. Simplemente tomamos ese JSON, lo validamos y luego generamos un API *Endpoint* para que los datos puedan ser consumidos por el *frontend*. [\[consultar código fuente aquí\]](#)

Fragmiento Código

```
import express from "express";
import path from "path";
import fs from "fs/promises";
import dotenv from "dotenv";

// Cargo las variables de entorno del archivo .env
dotenv.config();

// __dirname no está disponible directamente con ES Modules, lo creamos
import { fileURLToPath } from "url";
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const app = express();
const PORT = 3000;
// Obtener el nombre del archivo del .env, con un fallback por si no se define
const salesDataFilename = process.env.SALES_DATA_FILENAME || 'ventas.json';

// 1. Servir los archivos estáticos (HTML, CSS, JS, imágenes, ¡y el JSON!) de la carpeta 'public'
app.use(express.static(path.join(__dirname, "public")));

// 2. Genero la API para leer el JSON desde el archivo
app.get("/api/data", async (req, res) => {
```

```

console.log("Petición recibida en /api/data");
const filePath = path.join(__dirname, "public", "datos", salesDataFilename);
console.log(`Intentando leer datos de: ${filePath}`);

try {
    // Leo el archivo de forma asíncrona
    const data = await fs.readFile(filePath, "utf8");

    // Convierto el contenido JSON y enviarlo como respuesta
    res.json(JSON.parse(data));
    console.log("Datos de ventas.json servidos exitosamente.");
} catch (error) {
    console.error("Error al leer ventas.json:", error);
    res.status(500).json({
        error: "No se pudieron obtener los datos de ventas. X",
    });
}

// 3. Iniciar el servidor
app.listen(PORT, () => {
    console.log(`Servidor corriendo en http://localhost:${PORT}`);
});

```

El app.js (Orquestador Cliente)

El archivo app.js actúa como el orquestador principal (**main**) del lado del cliente. Desde aquí puedo importar las librerías necesarias, así como mis propios módulos (**models**). Esto me permite cumplir con las bases de desarrollo para crear un **dashboard** que sea flexible, incremental y escalable, asegurando el uso de buenas prácticas y evitando el código espagueti.

[\[consultar código fuente aquí\]](#)

Fragmento Código

```

/** Método Principal.
 * Descripción: Configura eventos e inicia la actualización periódica.
 */
async function main() {
    // 1. Configuro el evento para el botón de actualizar gráficos
    const updateButton = document.getElementById("updateButton");
    if (updateButton) {
        updateButton.addEventListener("click", async () => {
            if (timerElement) timerElement.textContent = "(Actualizando...)";
            // 1. Ejecutar actualización
            await actualizarDatosYGraficos();
            // 2. Actualizar vista (mostrar "en 5:00")
            segundosRestantes = TIEMPO_ACTUALIZACION_SEG;
        });
    }
}

```

```

        actualizarContadorVisual();
    });
}

// 2. Ejecutamos la actualización por PRIMERA VEZ al cargar la página.
if (timerElement.textContent = "(Cargando datos...)";
await actualizarDatosYGraficos();

// 3. Mostramos el contador inicial (ej. "en 5:00")
actualizarContadorVisual();

// 4. Configuro el intervalo para que se ejecute CADA SEGUNDO.
setInterval(gestionarTickActualizacion, UN_SEGUNDO_MS); // Cuenta Regresiva 5 minutos
setInterval(() => dibujaReloj(), UN_SEGUNDO_MS); // Pinto el reloj cada segundo
}

// 5. Ejecutar la función 'main' cuando el DOM esté completamente cargado
document.addEventListener("DOMContentLoaded", main);

```

Los Módulos de Gráficas (google-chart.js y d3-chart.js)

Generé módulos separados que se almacenan en el repositorio JS. Uno, llamado **google-chart.js**, contiene la lógica para mostrar la gráfica de barras. El otro archivo, **d3-chart.js**, contiene la lógica para mostrar las gráficas de burbuja (utilizando la librería D3.js). [\[consultar código fuente aquí\]](#)

Fragmento Código

```

/**
 * Módulo para Google Charts.
 * Objetivo: Visualización básica con Google Chart Library.
 * Implementa un gráfico de columnas que muestre la evolución de las
 * ventas mensuales por producto, con cada producto representado por una columna distinta.
 */

export function dibujarGraficosBarrasProductos(chartdataArray, DivArea) {
    // 3. Cargar la librería de Google Charts y dibujar el gráfico.
    google.charts.load("current", { packages: ["corechart"] });

    //Vamos a dibujar el gráfico una vez que la librería esté cargada
    google.charts.setOnLoadCallback(() => {
        const chartData = google.visualization.arrayToDataTable(chartdataArray);

        const options = {
            title: "Ventas Mensuales por Producto", // Nuevo título más descriptivo
            is3D:true,
            hAxis: {
                // Eje Horizontal (Meses)
                title: "Mes",

```

```

        titleTextStyle: { color: "#333" },
    },
    vAxis: {
        // Eje Vertical (Ventas)
        title: "Unidades Vendidas",
        minValue: 0,
        titleTextStyle: { color: "#333" },
    },
    bar: { groupWidth: "75%" }, // Ajustar el ancho de los grupos de barras
    isStacked: false, // ¡IMPORTANTE! Para columnas agrupadas (no apiladas)
    legend: { position: "top", alignment: "center", maxLines: 3 }, // Leyenda arriba y centrada para productos

    // Usamos tus colores, asegurándonos de tener uno para cada producto
    colors: ["#4cb5f5", "#34675C", "#B3C100"],

    chartArea: { width: "80%", height: "70%" }, // Ajustar el área para dar espacio a la leyenda
    animation: {
        duration: 1000,
        easing: "out",
        startup: true,
    },
    backgroundColor: { fill: "transparent" },
};

const chart = new google.visualization.ColumnChart(
    document.getElementById(DivArea)
);
chart.draw(chartData, options);
});
}

```

El Módulo de Utilidades (util.js)

También generé el módulo util.js. Este archivo contiene los métodos que me permiten consumir el **API Endpoint** para obtener los datos. Posteriormente, estos datos se envían como parámetros a las funciones que generan las gráficas, que en este caso son los módulos para el gráfico de barras y el gráfico de burbujas. Además, incluye métodos de validación y cálculos de totales.

[\[consultar código fuente aquí\]](#)

El index.html (La Presentación)

Sin dejar de lado el index.html, la joya de la corona. Este es el otro orquestador, donde se enlazan los diferentes métodos y llamadas a librerías para generar toda la magia visual, funcionando bajo un ambiente **Cliente/Servidor**. [\[consultar código fuente aquí\]](#)

```

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

```

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard Principal</title>

    <link rel="stylesheet" href="css/global.css">
    <link rel="stylesheet" href="css/dash_venta.css">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
    <script src="https://example.com/fontawesome/v7.0.1/js/solid.js"></script>
    <script src="https://example.com/fontawesome/v7.0.1/js/brands.js"></script>
    <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>

    <script src="https://d3js.org/d3.v7.min.js"></script>

    <script type="module" src="app.js"></script>
</head>
<body>

    <header class="main-header">
        <div class="header-content">
            <div class="header-left">

                <div class="logo-container">
                    <span>
                        
                    </span>
                </div>

                <h1>Métricas</h1>
                <i class="fas fa-chart-line header-icon"></i>
            </div>
        </div>
    </header>

    <nav class="main-nav">
        <ul>
            <li><a href="mailto:cuenca623@gmail.com">Contacto</a></li>
            <li><a href="https://github.com/LeoSan?tab=repositories">Quiénes Somos</a></li>
            <li><a href="https://github.com/LeoSan/MaestriaAnalysisDatosBigData_UNIR_2024/blob/main/03_SEMESTRE/01_HerramientasVisualizacion/Actividad_01/dash_ventas/public/datos/ventas.json">Data</a></li>
        </ul>
    </nav>
</div>
</header>

<main>
    <div class="controls">
        <button id="updateButton">Actualizar</button>
        <span id="countdown-timer" style="font-style: italic; color: #555;"></span>

```

```
</div>

<div class="info-card-container">

    <div class="info-card total">
        <div class="icon-wrapper">
            <i class="fa-solid fa-boxes-stacked card-icon"></i>
        </div>
        <div class="card-content">
            <p class="card-label">Unidades Vendidas</p>
            <p id="total-unidades" class="card-value">0</p>
        </div>
    </div>

    <div class="info-card ingresos">
        <div class="icon-wrapper">
            <i class="fa-solid fa-dollar-sign card-icon"></i>
        </div>
        <div class="card-content">
            <p class="card-label">Total Ingresos</p>
            <p id="total-ingresos" class="card-value">$0</p>
        </div>
    </div>

    <div class="info-card asp">
        <div class="icon-wrapper">
            <i class="fa-solid fa-calculator card-icon"></i>
        </div>
        <div class="card-content">
            <p class="card-label">ASP</p>
            <p id="asp-value" class="card-value">$0</p>
        </div>
    </div>

    <div class="info-card maximo">
        <div class="icon-wrapper">
            <i class="fa-solid fa-arrow-trend-up card-icon"></i>
        </div>
        <div class="card-content">
            <p class="card-label">Precio Máximo</p>
            <p id="precio-maximo" class="card-value">$0</p>
        </div>
    </div>

    <div class="info-card minimo">
        <div class="icon-wrapper">
            <i class="fa-solid fa-arrow-trend-down card-icon"></i>
        </div>
        <div class="card-content">
```

```

        <p class="card-label">Precio Mínimo</p>
        <p id="precio-minimo" class="card-value">$0</p>
    </div>
</div>
</div>
<div class="chart-container-wrapper">
    <div class="chart-box">
        <h2>Gráfico Google Chart</h2>
        <h4 id="msj-data-chart"></h4>
        <div id="gchart_div"></div>
    </div>

    <div class="chart-box">
        <h2>Gráfico D3.js</h2>
        <h4 id="msj-data-d3"></h4>
        <div id="d3_chart_div"></div>
    </div>
</div>
</main>

<section class="clock-section">
    <div class="clock-display">
        <div id="date-display" class="date-text"></div>
        <div id="time-display" class="time-text"></div>
    </div>
</section>

<footer class="main-footer">
    <p>&copy; <span id="currentYear"></span> Todos los derechos reservados.</p>
</footer>

<script type="module">
    document.getElementById('currentYear').textContent = new Date().getFullYear();
</script>

</body>
</html>

```

Interpretación de Datos

Para este análisis, se generó un dashboard que mide el rendimiento de ventas del primer trimestre **enero, febrero, marzo**. Se desea resaltar, que se realizó un planteamiento hipotético todo con el fin educativo de dar una narrativa a los datos y poder aplicar el conocimiento técnico que es el desarrollo y uso de gráficas **Google-chart y DN3**.

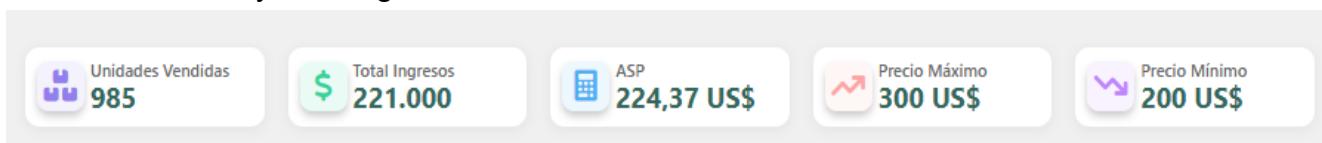
Insights Detectados

Se estableció dos objetivos (KPIs) supuestos para este trimestre:

- **Objetivo de Ingresos:** Alcanzar una meta de ingresos totales de \$125.000.
- **Objetivo de Eficiencia:** Mantener un Ingreso Promedio por Venta (ASP) superior a \$220.

Análisis de Resultados (La Narrativa)

Los resultados del primer trimestre son muy positivos y superan ambos objetivos supuestos. **Se alcanzó un total de ingresos de \$128.000**, superando nuestra meta de **\$125.000**. Esto se logró a través de un volumen total de **575 unidades vendidas**. Más importante aún, nuestro **ASP (Ingreso Promedio por Venta)** se situó en **\$222,61**. Esto no solo supera nuestro objetivo de **\$220**, sino que también nos indica que estamos logrando una mezcla de productos saludable y constantes que la mayoría de nuestros clientes tienen la confianza de adquirir por su excelente calidad, vendiendo eficientemente nuestros artículos de alta y media gama.



Las gráficas de Google Chart y D3.js nos muestran cómo logramos esos números, el producto estrella **Smartphone A**. El Gráfico **Google Chart (Ventas Mensuales)** muestra claramente que el **Smartphone A** es nuestro producto líder en volumen.

El gráfico de **burbujas (Ingresos)** de D3.js confirma que también es nuestro mayor generador de ingresos. Se observó un crecimiento de enero a febrero, impulsado principalmente por un aumento en las ventas del **Smartphone A** y una ligera subida en la **Tablet X**.



La Importancia de esta mezcla de productos es que se tiene un precio máximo y un mínimo, aunque la **Tablet X** es nuestro producto más caro, su contribución es vital para mantener nuestro **ASP** por encima del objetivo. El **Smartphone B** es el precio mínimo de \$200 actúa como una base de volumen estable.

Conclusiones

En conclusión, el análisis realizado con el dashboard, basado en datos hipotéticos del primer trimestre de ventas y que tienen fines de prueba, demuestra un resultado exitoso. No solo se cumplió la meta de ingresos brutos establecida, sino que se logró con eficiencia al **superar el objetivo** de Precio Promedio de Venta (ASP). El **Smartphone A** se consolida como el motor del crecimiento, mientras que la **Tablet X** es clave para mantener una rentabilidad promedio por su alto valor a la venta.

Adicionalmente, deseo proponer llevar a cabo estudios complementarios claves, con el fin detectar que está pasando con la estrategia de venta de la **Smartphone B** realizar posibles encuestas de satisfacción del cliente, análisis de sentimiento del productos, validar las campañas de ofertas y una validación exhaustiva de las publicaciones en redes sociales. El objetivo es identificar con precisión las áreas de mejora para estar mejor preparados en el **próximo trimestre** y conseguir **un aumento en las ventas** de aquellos productos con bajo desempeño. Todo este trabajo se abordará con un enfoque riguroso, listo para aplicar el conocimiento adquirido y generar una ventaja competitiva.

Sin duda, saber emplear estos conocimientos técnicos para construir un ambiente **CLIENTE/SERVIDOR**, utilizando **HTML, CSS y JAVASCRIPT**, es fundamental para diseñar dashboard visuales e interactivos. Esta base técnica no solo facilita la interacción con los datos, sino que también es un elemento esencial para lograr una mayor precisión al **crear la narrativa y la historia de los datos mejor conocido del English Data Storytelling**. Como bien se enfatiza en esta unidad, una imagen bien definida y presentada tiene más valor que mil palabras. Si bien poseer un profundo conocimiento técnico es fundamental, este esfuerzo resulta insuficiente si no logramos construir una narrativa de datos sólida y efectiva. Por lo tanto, se puede concluir que la combinación de sólidas habilidades técnicas con una perspicacia aguda e intuición es clave para lograr una exposición de datos impactante y accionable.