«Talento Tech»

Data Analytics

con Python

Clase 14

Buenos Aires Ciudad

Clase 14 | Visualización Avanzada con Seaborn

Temario:

- Crear visualizaciones estadísticas usando Seaborn.
- Personalizaciones
- Gráficas orientadas a objetos.

Objetivos de la Clase:

- Explorar cómo Seaborn complementa a Matplotlib, destacando sus diferencias clave y ventajas en la visualización de datos.
- Adquirir habilidades para construir y personalizar diferentes tipos de gráficos utilizando Seaborn.
- Familiarizarse con la interacción de Seaborn y estructuras de datos pandas.
- Conocer cómo crear y organizar múltiples gráficos en subplots utilizando tanto Matplotlib como Seaborn, mejorando la comparación y presentación de datos.

Data-Viz con Seaborn

Previamente habíamos aprendido que la visualización de datos permite interpretar y comunicar la información de manera efectiva en el análisis de datos. Seaborn es una **biblioteca de visualización basada en Matplotlib que proporciona una interfaz más fácil y atractiva para crear gráficos estadísticos complejos**. En esta clase, exploraremos la relación entre Seaborn y Matplotlib, sus diferencias y varios tipos de gráficos que se pueden crear y personalizar.

Relación y Diferencias con Matplotlib

Seaborn se construye sobre Matplotlib y complementa sus capacidades añadiendo una **gama de funciones y estilos predefinidos que simplifican la creación de gráficos estadísticos**. Aunque Matplotlib es poderoso y flexible, a menudo requiere más código para obtener visualizaciones atractivas, mientras que Seaborn simplifica este proceso mediante funciones que automatizan aspectos como el manejo de paletas de colores y la representación de estadísticas.

Las principales diferencias son:

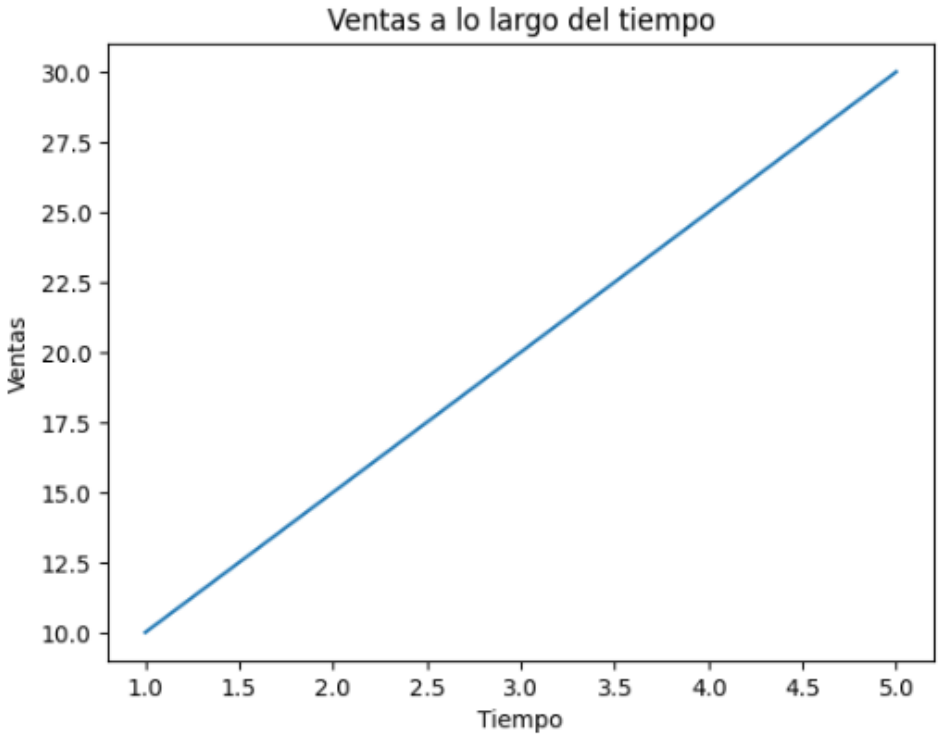
1. **Estilo y Atractivo Visual**: Seaborn incluye estilos de gráfico más atractivos predeterminados, lo que permite a los usuarios crear visualizaciones agradables de manera más sencilla.
2. **Soporte para Datos Estadísticos**: Seaborn maneja automáticamente los datos estadísticos y permite crear gráficos que pueden mostrar tendencias y distribuciones de manera más intuitiva.
3. **Interacción con DataFrames**: Seaborn trabaja mejor con estructuras de datos como DataFrames de pandas, lo que facilita la manipulación de datos y la creación de gráficos.

Tipos de Data-Viz Básicos y sus Personalizaciones

1. Gráficos de Líneas

Los gráficos de líneas en Seaborn son **ideales para mostrar la evolución de una variable durante un período de tiempo**.

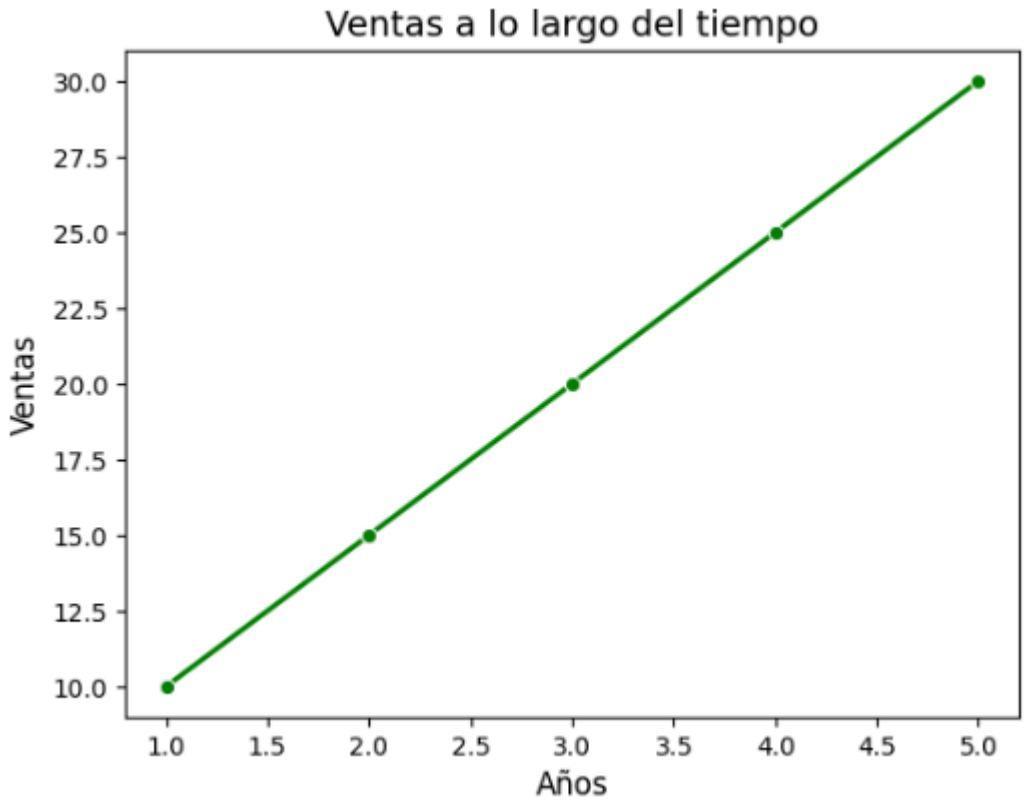
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Datos
data = pd.DataFrame({'Tiempo': [1, 2, 3, 4, 5], 'Ventas':
[10, 15, 20, 25, 30]})

# Gráfico de líneas
sns.lineplot(x='Tiempo', y='Ventas', data=data)
plt.title('Ventas a lo largo del tiempo')
plt.show()
```



Personalización del Gráfico de Líneas

Se pueden ajustar colores, estilos de línea y más:

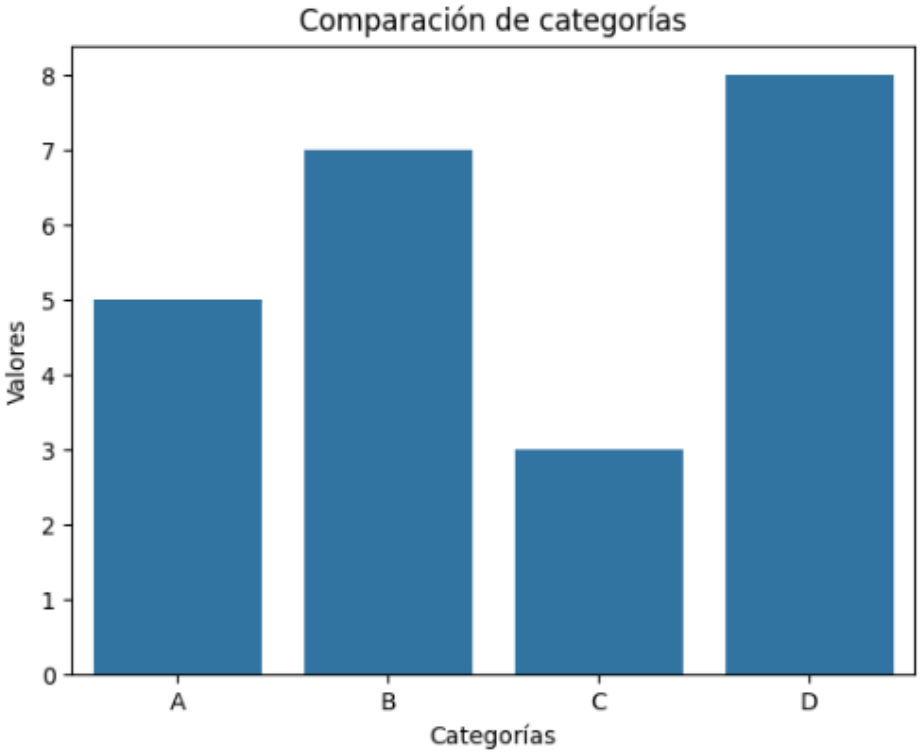
```
sns.lineplot(x='Tiempo', y='Ventas', data=data, marker='o', color='green',
linewidth=2)
plt.title('Ventas a lo largo del tiempo', fontsize=14)
plt.xlabel('Años', fontsize=12)
plt.ylabel('Ventas', fontsize=12)
plt.show()
```



2. Gráficos de Barras

Los gráficos de barras son excelentes para **comparar diferentes grupos de datos**.

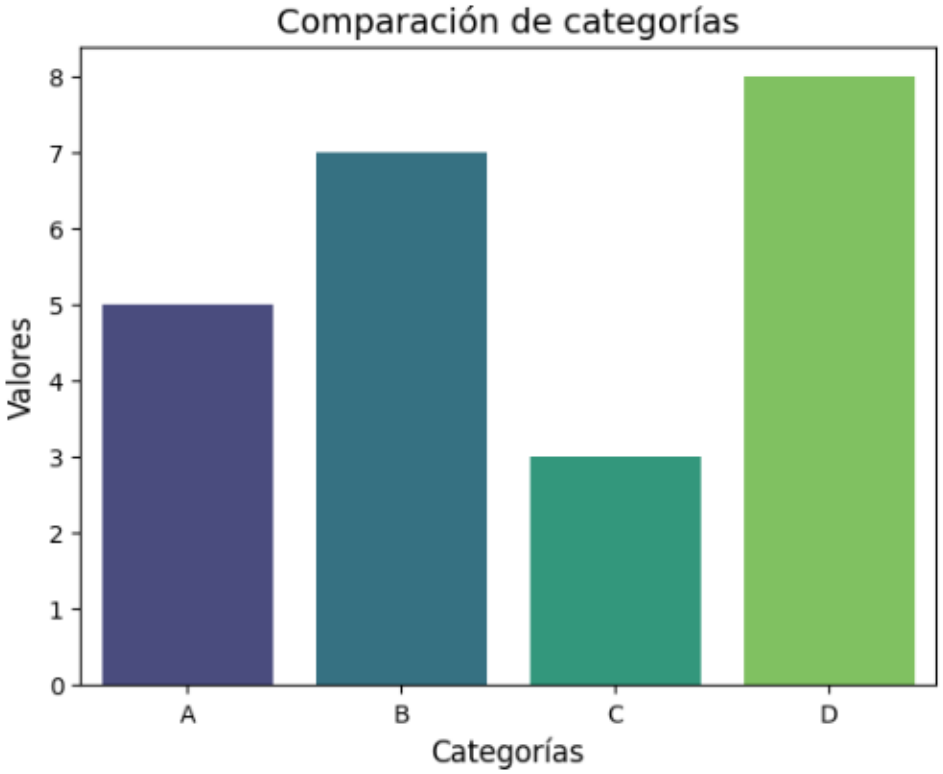
```
# Datos
categorias = ['A', 'B', 'C', 'D']
valores = [5, 7, 3, 8]
data_barras = pd.DataFrame({'Categorías': categorias,
'Valores': valores})

# Gráfico de barras
sns.barplot(x='Categorías', y='Valores', data=data_barras)
plt.title('Comparación de categorías')
plt.ylabel('Valores')
plt.show()
```



Personalización del Gráfico de Barras

Podemos agregar colores y ajustar el ancho de las barras:

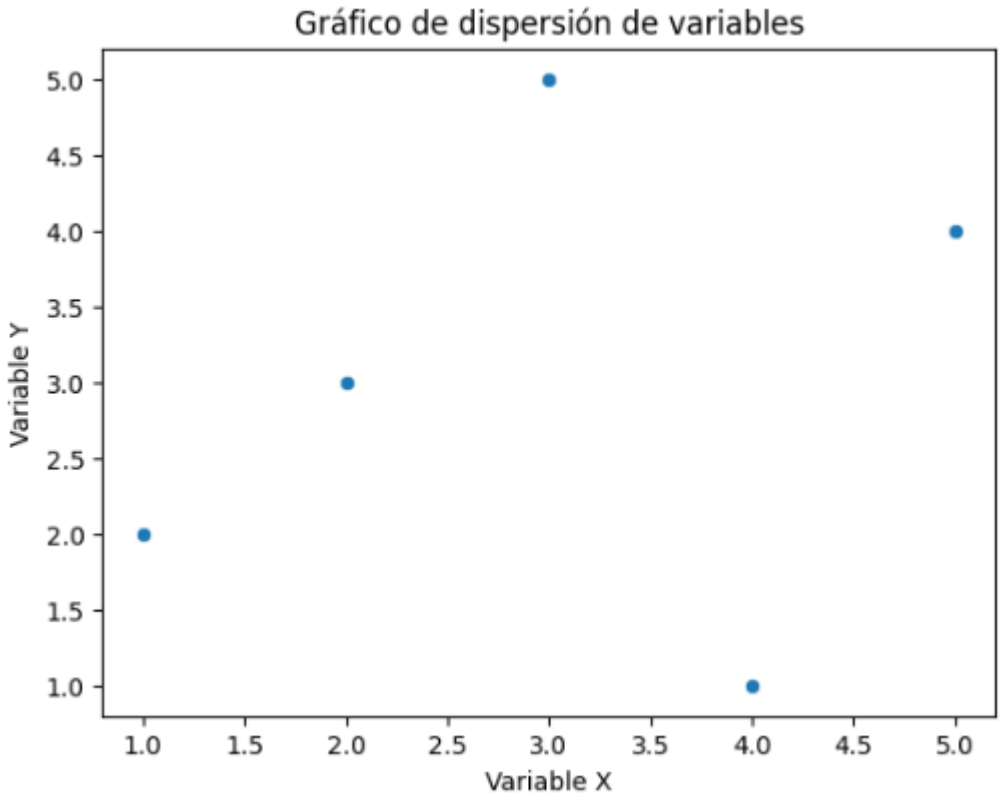
```
sns.barplot(x='Categorías',  y='Valores',  data=data_barras,
palette='viridis')
plt.title('Comparación de categorías', fontsize=14)
plt.xlabel('Categorías', fontsize=12)
plt.ylabel('Valores', fontsize=12)
plt.show()
```



3. Gráficos de Dispersión

Los gráficos de dispersión permiten **visualizar la relación entre dos variables**.

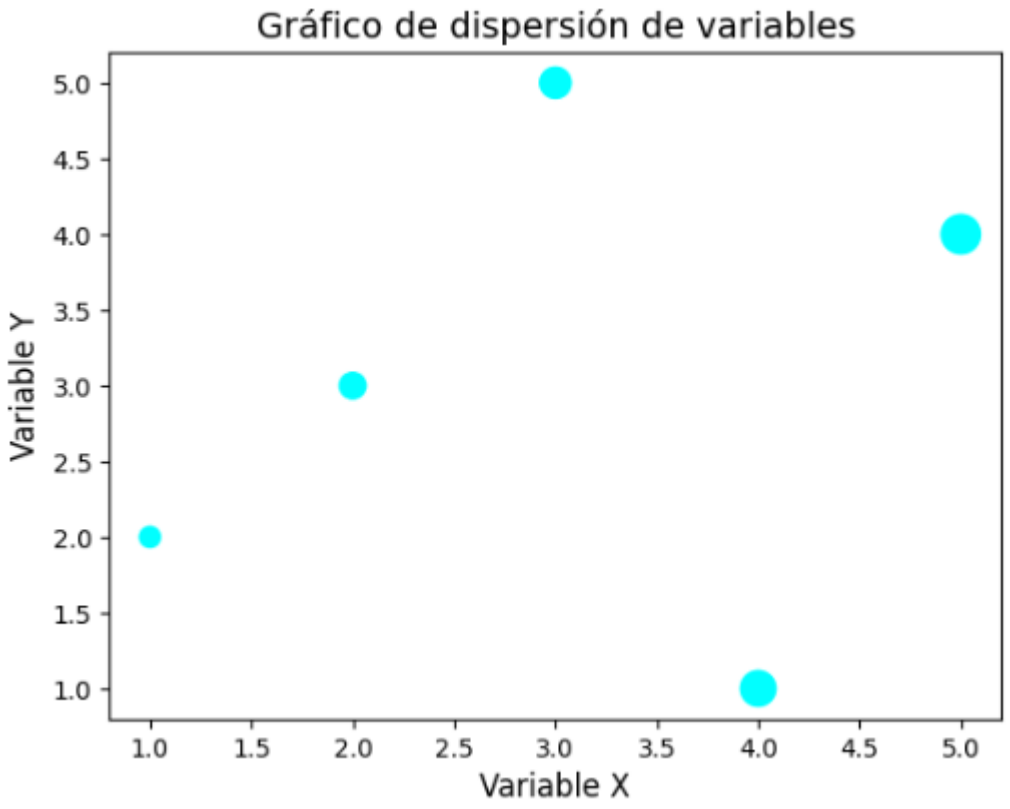
```
# Datos
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 1, 4]
data_dispersion = pd.DataFrame({'Variable X': x, 'Variable Y': y})

# Gráfico de dispersión
sns.scatterplot(x='Variable X', y='Variable Y', data=data_dispersion)
plt.title('Gráfico de dispersión de variables')
plt.show()
```



Personalización del Gráfico de Dispersión

Podemos ajustar colores y tamaños de los puntos:

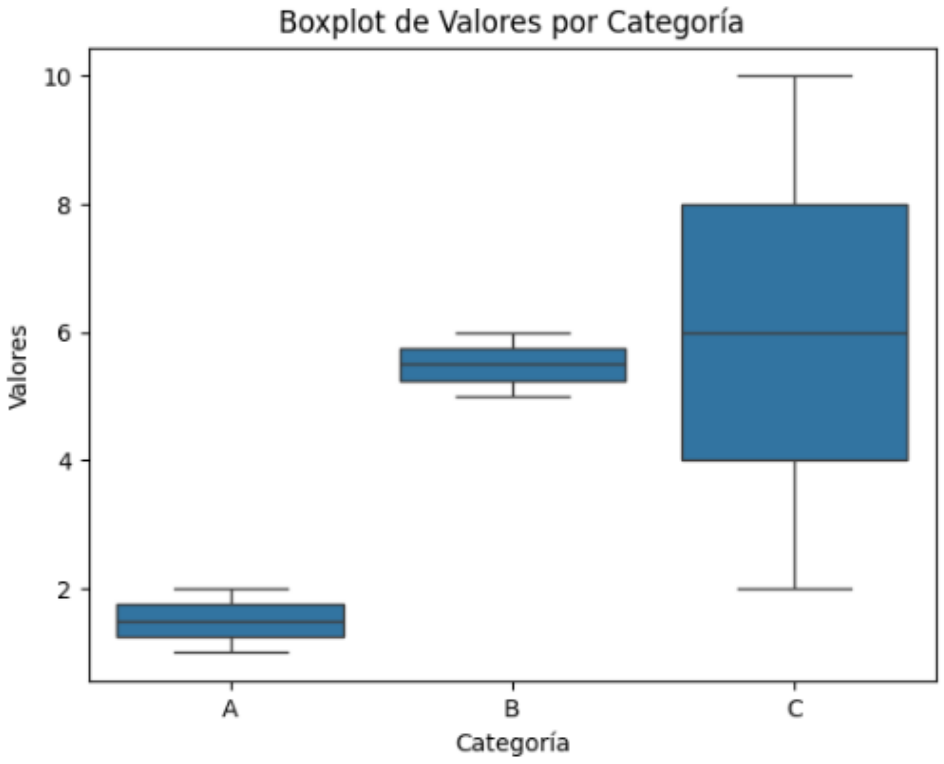
```
sns.scatterplot(x='Variable X', y='Variable Y',
data=data_dispersion, color='cyan', s=[100, 150, 200, 250,
300])
plt.title('Gráfico de dispersión de variables', fontsize=14)
plt.xlabel('Variable X', fontsize=12)
plt.ylabel('Variable Y', fontsize=12)
plt.show()
```



4. Boxplot (Gráfico de Caja)

Los boxplots son útiles para mostrar la **distribución de datos y resaltar valores atípicos**.

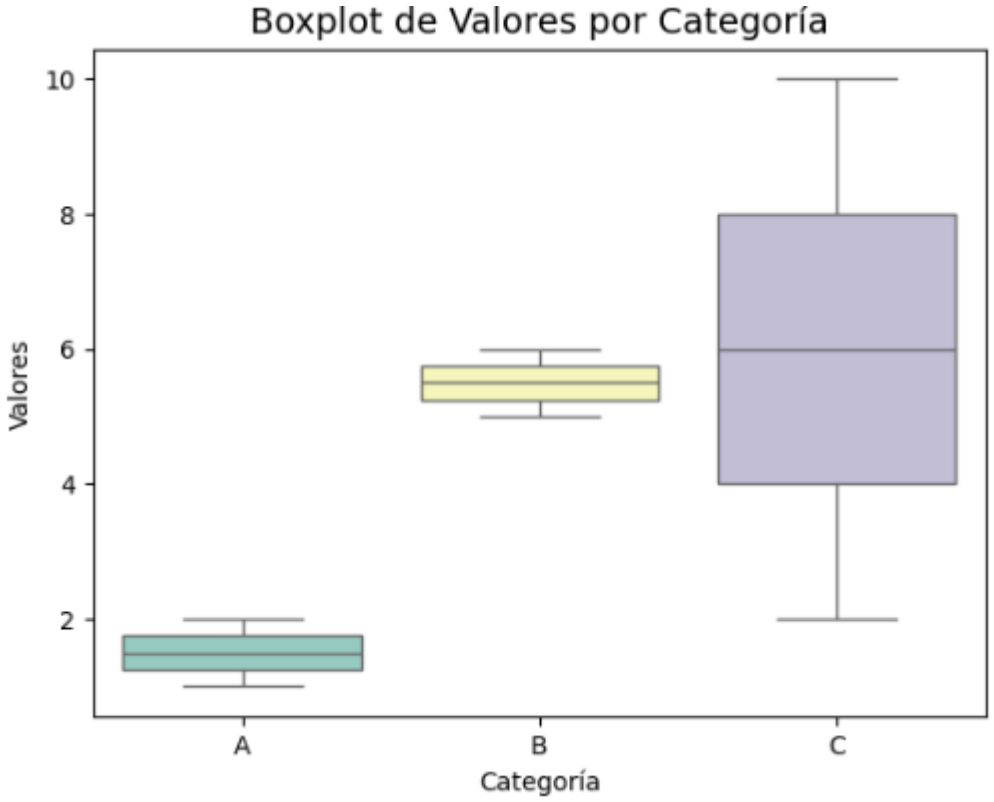
```
# Datos
data_box = pd.DataFrame({
    'Categoría': ['A', 'A', 'B', 'B', 'C', 'C'],
    'Valores': [1, 2, 5, 6, 2, 10]
})

# Gráfico de caja
sns.boxplot(x='Categoría',                y='Valores',
data=data_box)
plt.title('Boxplot de Valores por Categoría')
plt.show()
```



Personalización del Boxplot

Podemos ajustar la paleta de colores o agregar puntos para los valores atípicos:

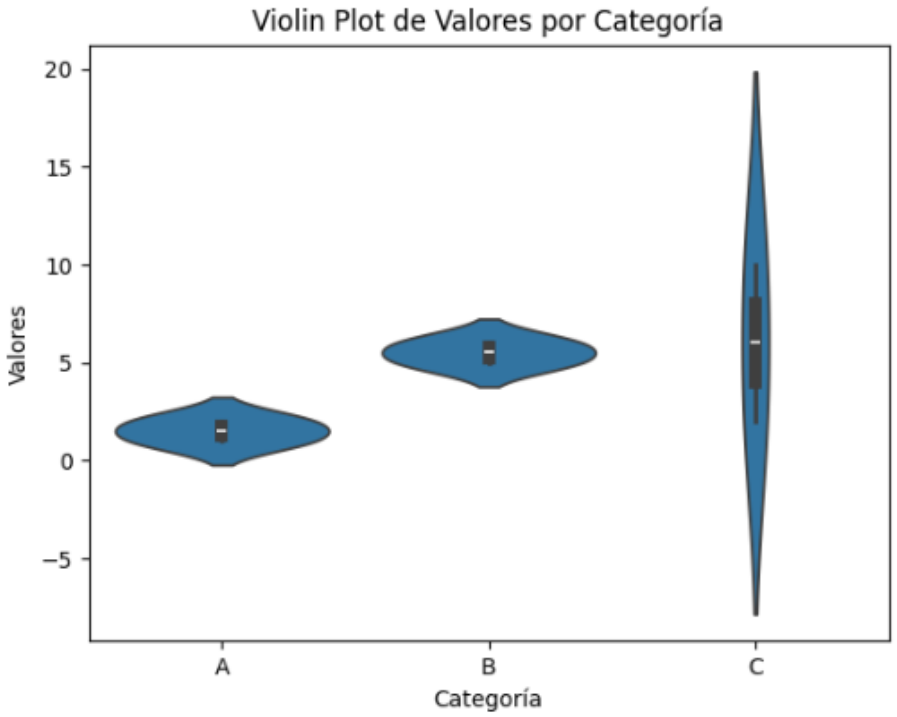
```
sns.boxplot(x='Categoría',     y='Valores',     data=data_box,
palette='Set3')
plt.title('Boxplot de Valores por Categoría', fontsize=14)
plt.show()
```



5. Violin Plot

El Violin Plot **combina un boxplot con la densidad de probabilidad de la variable**.

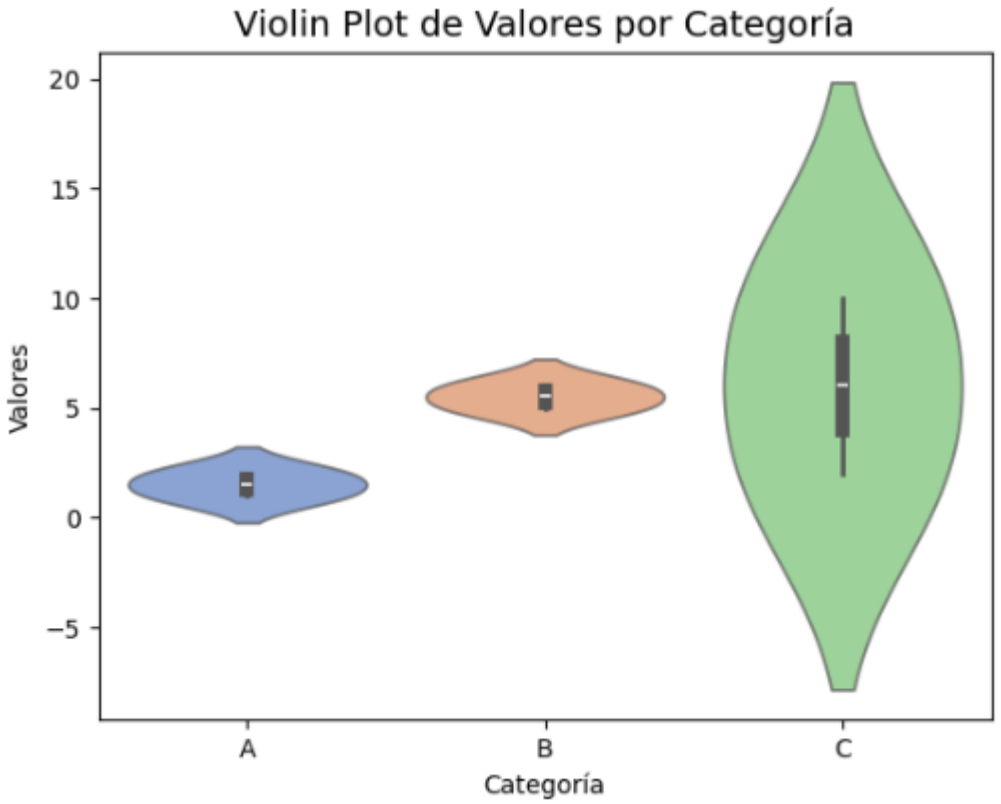
```
# Gráfico de violín
sns.violinplot(x='Categoría', y='Valores', data=data_box)
plt.title('Violin Plot de Valores por Categoría')
plt.show()
```



Personalización del Violin Plot

Podemos ajustar los colores y la transparencia:

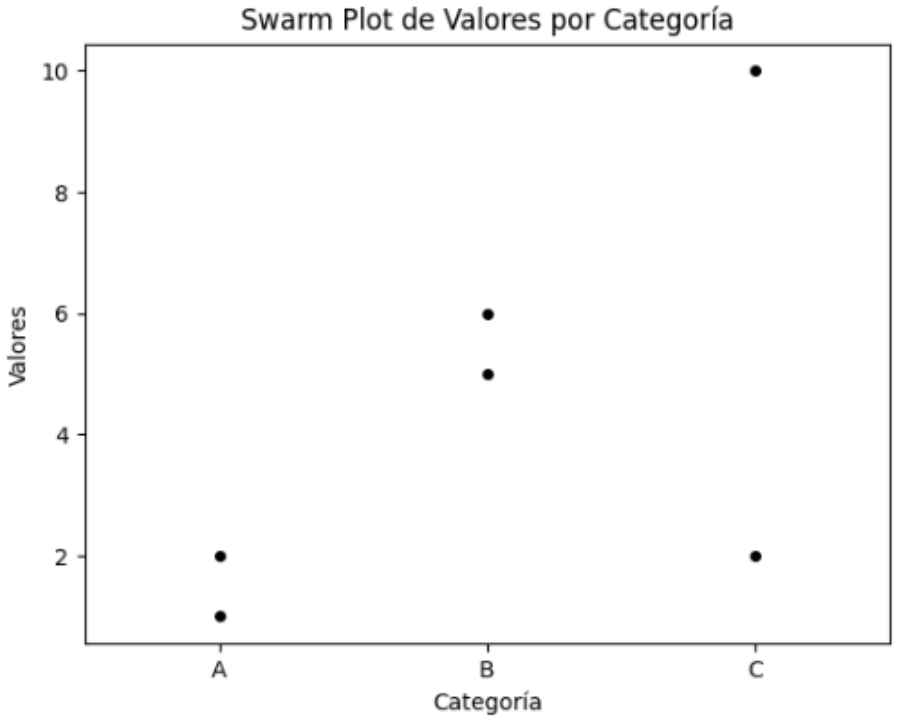
```
sns.violinplot(x='Categoría', y='Valores', data=data_box, palette='muted',
alpha=0.7)
plt.title('Violin Plot de Valores por Categoría', fontsize=14)
plt.show()
```



6. Swarm Plot

Los swarm plots son útiles para **visualizar la distribución de datos sobre una categoría y evitar la sobreposición de puntos**.

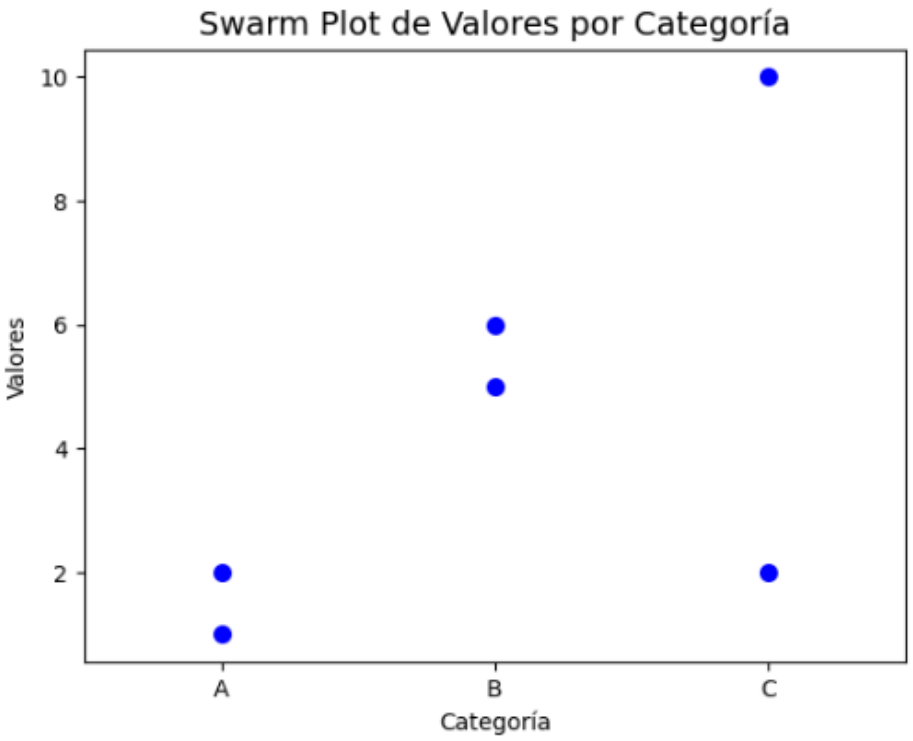
```
# Gráfico de swarm
sns.swarmplot(x='Categoría',    y='Valores',    data=data_box,
color='black')
plt.title('Swarm Plot de Valores por Categoría')
plt.show()
```



Personalización del Swarm Plot

Se pueden cambiar el color y ajustar el tamaño de los puntos:

```
sns.swarmplot(x='Categoría',    y='Valores',    data=data_box,
color='blue', size=8)
plt.title('Swarm Plot de Valores por Categoría', fontsize=14)
plt.show()
```



Subplots en Matplotlib y Seaborn

Los subplots son una herramienta poderosa en la visualización de datos, que **permite crear múltiples gráficos en una sola figura**. Esta funcionalidad es particularmente útil cuando se desea comparar visualmente diferentes conjuntos de datos o mostrar diversas representaciones gráficas de la misma información.

¿Qué son los Subplots?

En Matplotlib, un subplot es una **división de la figura principal en una cuadrícula de subgráficos**. Cada subgráfico puede contener un gráfico diferente, lo que permite al usuario organizar y presentar datos de manera efectiva. Los subplots se pueden crear ajustando la disposición de filas y columnas para acomodar los diferentes gráficos.

Beneficios de Usar Subplots

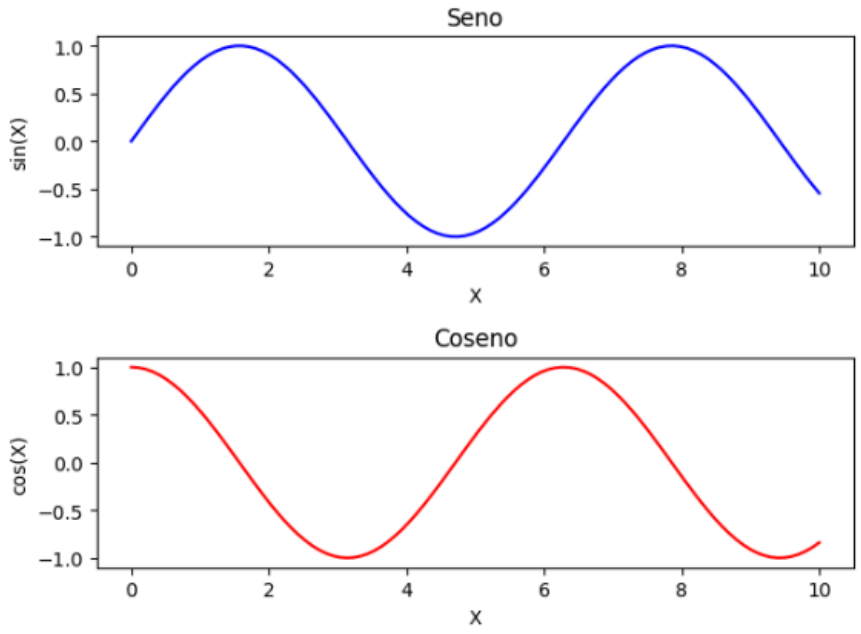
1. **Comparación Visual**: Permiten comparar varios gráficos de manera fácil y directa.
2. **Organización**: Ayudan a mantener la información organizada en una sola vista.
3. **Ahorro de Espacio**: Reducen la necesidad de crear varias figuras, haciendo la presentación más eficiente.
4. **Contexto**: Pueden proporcionar contexto al mostrar diferentes aspectos de un mismo conjunto de datos.

Ejemplos de Subplots en Matplotlib

A continuación, se presentan ejemplos de cómo crear subplots utilizando Matplotlib.

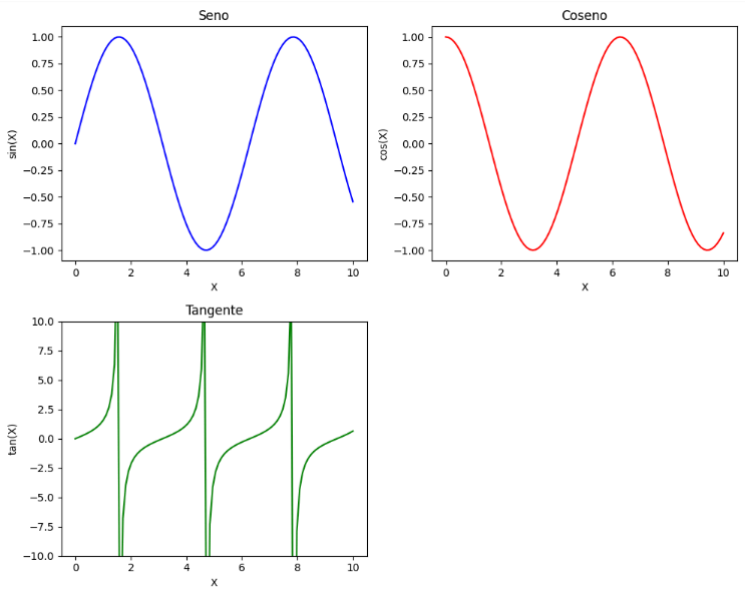
Ejemplo 1: Subplots Básicos

```
import matplotlib.pyplot as plt
import numpy as np

# Crear datos
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Crear una figura y varios subplots
fig, axs = plt.subplots(2, 1)  # 2 filas y 1 columna

# Primer subplot: Seno
axs[0].plot(x, y1, color='blue')
axs[0].set_title('Seno')
axs[0].set_xlabel('X')
axs[0].set_ylabel('sin(X)')

# Segundo subplot: Coseno
axs[1].plot(x, y2, color='red')
axs[1].set_title('Coseno')
axs[1].set_xlabel('X')
axs[1].set_ylabel('cos(X)')

# Ajustar el layout
plt.tight_layout()
plt.show()
```



Ejemplo 2: Subplots en una Cuadrícula

```
# Crear datos
y3 = np.tan(x)

# Crear una figura y varios subplots en una cuadrícula de 2x2
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

# Gráfico de Seno
axs[0, 0].plot(x, y1, color='blue')
axs[0, 0].set_title('Seno')
axs[0, 0].set_xlabel('X')
axs[0, 0].set_ylabel('sin(X)')

# Gráfico de Coseno
axs[0, 1].plot(x, y2, color='red')
axs[0, 1].set_title('Coseno')
axs[0, 1].set_xlabel('X')
axs[0, 1].set_ylabel('cos(X)')

# Gráfico de Tangente
axs[1, 0].plot(x, y3, color='green')
axs[1, 0].set_ylim(-10, 10)  # Limitar el eje y
axs[1, 0].set_title('Tangente')
axs[1, 0].set_xlabel('X')
axs[1, 0].set_ylabel('tan(X)')

# Gráfico vacío
axs[1, 1].axis('off')  # Ocultar este subplot

# Ajustar el layout
plt.tight_layout()
plt.show()
```

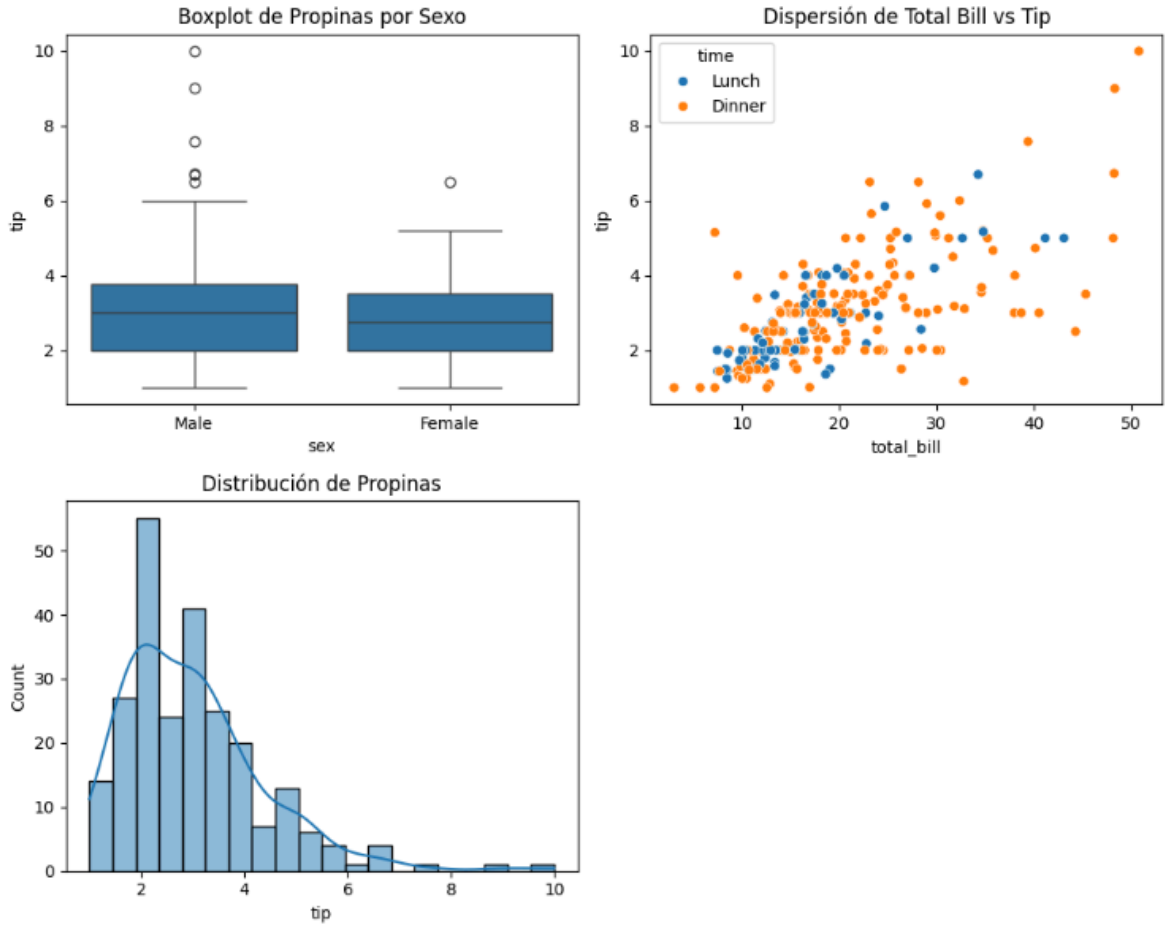


Ejemplos de Subplots en Seaborn

Aunque Seaborn se basa en Matplotlib, se pueden crear subplots de manera similar utilizando la función plt.subplots().

Ejemplo 3: Subplots con Seaborn

```
import seaborn as sns
import pandas as pd

# Cargar el conjunto de datos
tips = sns.load_dataset('tips')

# Crear una figura y varios subplots
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

# Gráfico de cajas de la propina por sexo
sns.boxplot(x='sex', y='tip', data=tips, ax=axs[0, 0])
axs[0, 0].set_title('Boxplot de Propinas por Sexo')

# Gráfico de dispersión de total_bill vs tip
sns.scatterplot(x='total_bill', y='tip', data=tips, ax=axs[0,
1], hue='time')
axs[0, 1].set_title('Dispersión de Total Bill vs Tip')

# Histograma de propinas
sns.histplot(tips['tip'], bins=20, ax=axs[1, 0], kde=True)
axs[1, 0].set_title('Distribución de Propinas')

# Gráfico vacío
axs[1, 1].axis('off')  # Ocultar este subplot

# Ajustar el layout
plt.tight_layout()
plt.show()
```



Reflexión Final

Seaborn ofrece una forma efectiva y estética de visualizar datos, facilitando la creación de gráficos estadísticos complejos con menos código que Matplotlib. Al incorporar diferentes tipos de gráficos como boxplots, violin plots y swarm plots, Seaborn enriquece la presentación de datos y permite a los analistas comunicar sus hallazgos de manera más convincente y atractiva. **Los subplots son una herramienta que permite la visualización de múltiples gráficos en una sola figura**. Ya sea en Matplotlib o Seaborn, el uso de subplots mejora la capacidad de comparar, organizar y presentar información visualmente.

Materiales y recursos adicionales

- seaborn.pydata.org

Próximos Pasos

- Introducción a la visualización interactiva con Plotly.
- Comparación con Matplotlib y Seaborn.
- Creación de gráficos dinámicos y dashboards.

Ejercicios Prácticos

Actividad 1: Comparación de Rendimiento de Clientes con Boxplots

Contexto

Durante esta semana de tu pasantía en SynthData, Matías, el Data Analyst, te ha solicitado que realices un análisis más profundo sobre el comportamiento de los clientes. Se te pide que visualices la distribución de las compras de los clientes y que identifiques posibles valores atípicos que podrían influir en el rendimiento general.

Objetivos

- Aprender a crear boxplots utilizando Seaborn para analizar la distribución de los datos.
- Identificar clientes con conductas de compra inusuales.
- Fortalecer la comprensión de la visualización estadística como herramienta en la toma de decisiones empresariales.

Ejercicio práctico

Utilizarás un dataset que contiene información sobre las compras realizadas por diferentes clientes. Las columnas son: **cliente**, **importe_compra**, y **mes**.

1. Cargá el dataset y visualizá los primeros registros.
2. Creá un boxplot que muestre la distribución de las compras por mes. Asegurate de personalizar el gráfico con títulos y etiquetas adecuadas.

Dataset

```
 Dataset
data = pd.DataFrame({
    'cliente': ['Cliente 1', 'Cliente 2', 'Cliente 3', 'Cliente 4', 'Cliente 5'],
    'importe_compra': [200, 450, 120, 700, 30],
    'mes': ['Enero', 'Enero', 'Febrero', 'Febrero', 'Febrero']
})
```

¿Por qué importa esto en SynthData?

Comprender la distribución de las compras por parte de los clientes permite detectar patrones y anomalías que podrían impactar en las estrategias comerciales. Los boxplots ayudan a visualizar la variabilidad de los datos y descubrir valores atípicos que merecen un análisis más profundo. Esto ayuda a SynthData a mejorar la segmentación y personalización de sus ofertas.

Actividad 2: Análisis de Productos con Violin Plots y Gráficos de Barras

Contexto

Como parte de tu pasantía en SynthData, Luis, el Analista de BI, te ha encargado evaluar cómo se distribuyen los precios de varios productos en diferentes categorías, utilizando técnicas de visualización para hacer esta información más accesible. Para realizar un análisis completo, deberás crear gráficos integrados utilizando subplots, incluyendo violin plots y gráficos de barras.

Objetivos

- Aprender a crear violin plots y gráficos de barras utilizando Seaborn.
- Comprender cómo las diferentes categorías de productos afectan la distribución de precios.
- Desarrollar habilidades en la creación de subplots para una comparación visual efectiva de datos.

Ejercicio práctico

Utilizarás un dataset que contiene información sobre los precios de diferentes productos categorizados. Las columnas son: **categoría**, **producto**, y **precio**.

1. Cargá el dataset y visualiza los primeros registros.

2. Creá un subplot que contenga un violin plot mostrando la distribución de precios por categoría, junto con un gráfico de barras que represente el precio promedio por categoría.

Dataset

```
# Dataset
data = pd.DataFrame({
    'categoría': ['Electrónica', 'Electrónica', 'Ropa', 'Ropa', 'Alimentos',
'Alimentos'],
    'producto': ['Televisor', 'Radio', 'Camisa', 'Pantalón', 'Pan', 'Leche'],
    'precio': [300, 150, 20, 40, 2, 1.5]
})
```

¿Por qué importa esto en SynthData?

La capacidad de visualizar la distribución de precios y entender cómo se agrupan los productos en diferentes categorías es crucial para la estrategia de precios de SynthData. Utilizar violin plots y gráficos de barras juntos permite al equipo identificar patrones y tendencias, lo que facilita la toma de decisiones informadas sobre la optimización del catálogo y el ajuste de precios en distintas categorías.

⛔ **Estos ejercicios son una simulación de cómo se podría resolver el problema en este contexto específico. Las soluciones encontradas no aplican de ninguna manera a todos los casos.**
Recuerda que las soluciones dependen de los sets de datos, el contexto y los requerimientos específicos de los stakeholders y las organizaciones.


Buenos Aires
aprende
Agencia de Habilidades para el Futuro
Buenos
Aires
Ciudad