TIC

ACTIVIDAD 1

Tipo actividad: Taller de visualización de datos

Tiempo de ejecución: 4 horas

Tiempo de ejecución: 4 horas	
PLANTEAMIENTO DE LA SESIÓN	Materiales
En esta sesión se realizará un taller orientado a la visualización de datos para aumentar la comprensión de las herramientas vistas, las ventajas y el uso en un entorno de exploración de datos.	• Base de datos del titanic. • Acceso a Google Colab: colab.research.google.com

Para este taller se generarán diferentes visualizaciones de datos empleando la base de datos del titanic, previamente estudiada en la unidad 1. Para el desarrollo del taller se emplearán herramientas como matplotlib y seaborn en Python.

En caso de no tener un entorno instalado de Python, se puede emplear herramientas como Google colab (colab.research.google.com) que permite la ejecución de programas de Python con librerías previamente instaladas.

Para iniciar se va a cargar desde Python las herramientas de visualización, para esto se debe usar la palabra import. En Google colab ya se encuentran previamente instaladas muchas de las librerías que se van a emplear en el taller, por lo que no hace falta su instalación. En el bloque de código 1 se muestra el primer bloque, para cargar las librerías. Para ejecutar el bloque se puede hacer en el botón de play en la celda (costado lateral izquierdo) o también se puede en el teclado pulsar alt + enter. Al ejecutarse el código no debe mostrarse nada más (solo la ejecución con una marca verde de aceptación en el costado izquierdo).

```
# carga de paquetes y funciones para generar
gráficos
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

bloque de código 1: importación de librerías

Posteriormente se va a crear un gráfico simple de una función, para esto se crearán datos sintéticos (aleatorios) y luego se generará la suma acumulada para visualizarlos.

```
rango = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rango.randn(500,6), 0)
```

En escencia np.random.RandomState(0) crea un generador de números aleatorios utilizando la función RandomState de NumPy. Establece una semilla (seed) para el generador en 0, lo que garantiza que la secuencia de números aleatorios generada sea la misma cada vez que se ejecute el código, lo que facilita la reproducibilidad de los resultados.
x = np.linspace(0, 10, 500): Esta línea genera un arreglo de 500 valores espaciados uniformemente en el rango de 0 a 10. La función linspace() de NumPy crea una secuencia de números con un paso uniforme entre dos valores dados. En este caso, x representará puntos en el eje x de un gráfico. y = np.cumsum(rango.randn(500,6), 0): Aquí se genera una matriz de datos y de dimensiones 500x6. rango.randn(500,6) genera una matriz de números aleatorios distribuidos de forma normal (gaussiana) con media 0 y desviación estándar 1. np.cumsum() calcula la suma acumulativa a lo largo del eje especificado (0 en este caso), lo que significa que cada fila de la matriz y es la suma acumulativa de las filas anteriores. En otras palabras, cada columna en y representa una serie de puntos que serán graficados, donde los valores acumulativos se suman a lo largo del tiempo o del índice de los datos en x.

```
# Graficar los datos
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```

El código plt.plot(x, y) utiliza la función plot() de la biblioteca Matplotlib para trazar los datos contenidos en los arreglos x e y. Esta función traza una serie de puntos conectados por líneas en un gráfico. Los valores en el arreglo x representan las coordenadas en el eje x, mientras que los valores en el arreglo y representan las coordenadas en el eje y. Por lo tanto, cada columna en y (los datos generados aleatoriamente) correspondería a una serie de puntos trazados frente a los valores en x.
El código plt.legend('ABCDEF', ncol=2, loc='upper left') agrega una leyenda al gráfico. La letra 'ABCDEF' proporciona las etiquetas para cada una de las series trazadas, donde cada letra corresponde a una columna en el arreglo y. ncol=2 especifica el número de columnas en las que se organizan las etiquetas en la leyenda, en este caso, dos columnas. loc='upper left' indica la ubicación de la leyenda en el gráfico, en este caso, en la esquina superior izquierda. La imagen resultante queda en la figura 1

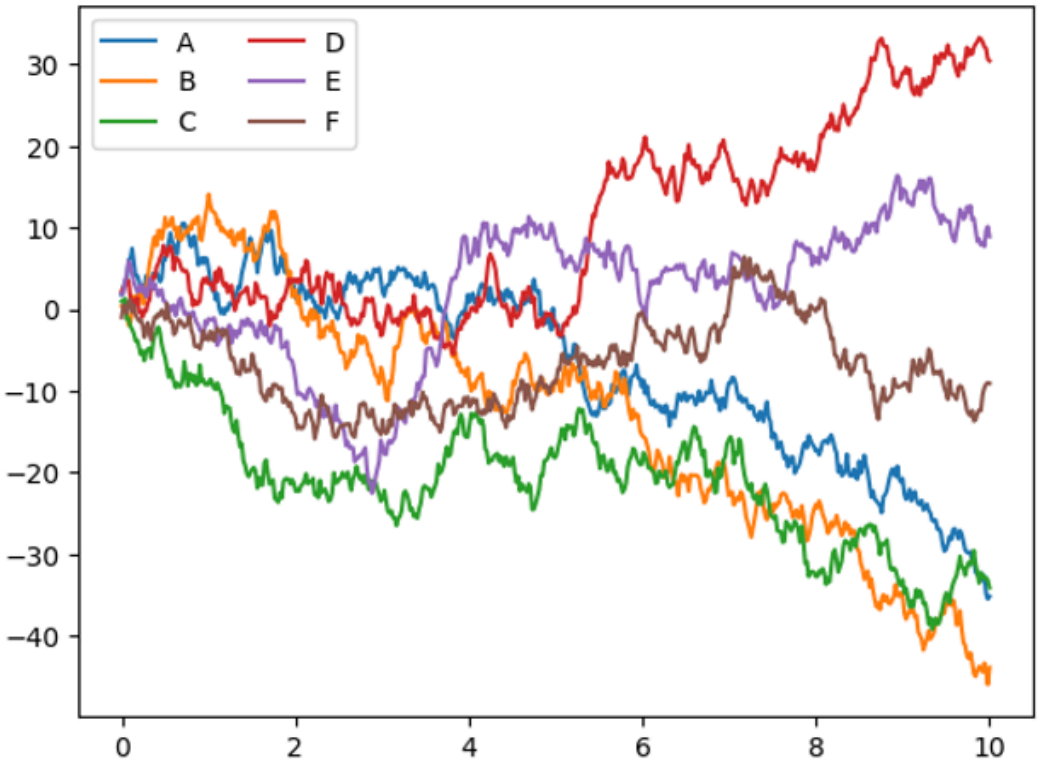


Figura 1: Resultado esperado de la generación del gráfico.

Si bien los datos se aprecian correctamente, la visualización tiene un tono un poco anticuado, para darle una mejor estética se puede emplear la herramienta seaborn, la que sobre escribe algunos estilos de matplotlib mejorando la forma en que se ven las señales dibujadas. Para cambiar el estilo se puede emplear la función set() de seaborn que se invoca como muestra el siguiente código:

```
sns.set() # carga de estilos de seaborn
# luego, se dibuja exactamente igual que en el
# Código anterior
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```

La salida debe ser exactamente igual a la figura 2.

Figura 2: Gráfico mejorado con seaborn

Existen varios tipos de gráficos, en el caso anterior se dibujaron líneas conectadas con la función plot. Sin embargo, se pueden dibujar solamente los puntos, o incluso los puntos sobre las líneas. Para esto se usa la función scatter. En el siguiente ejemplo se crea un espacio lineal desde 0 hasta 10, con 20 puntos de información. Y se crea un rango de variables que tiene los puntos elevados al cuadrado (y_val)
Luego se dibujan los puntos empleando la función scatter

```
generando puntos de una función
val = np.linspace(0, 10, 20)
val = x_val ** 2
t.scatter(x val, y val)
```

El resultado se aprecia en la figura 3:

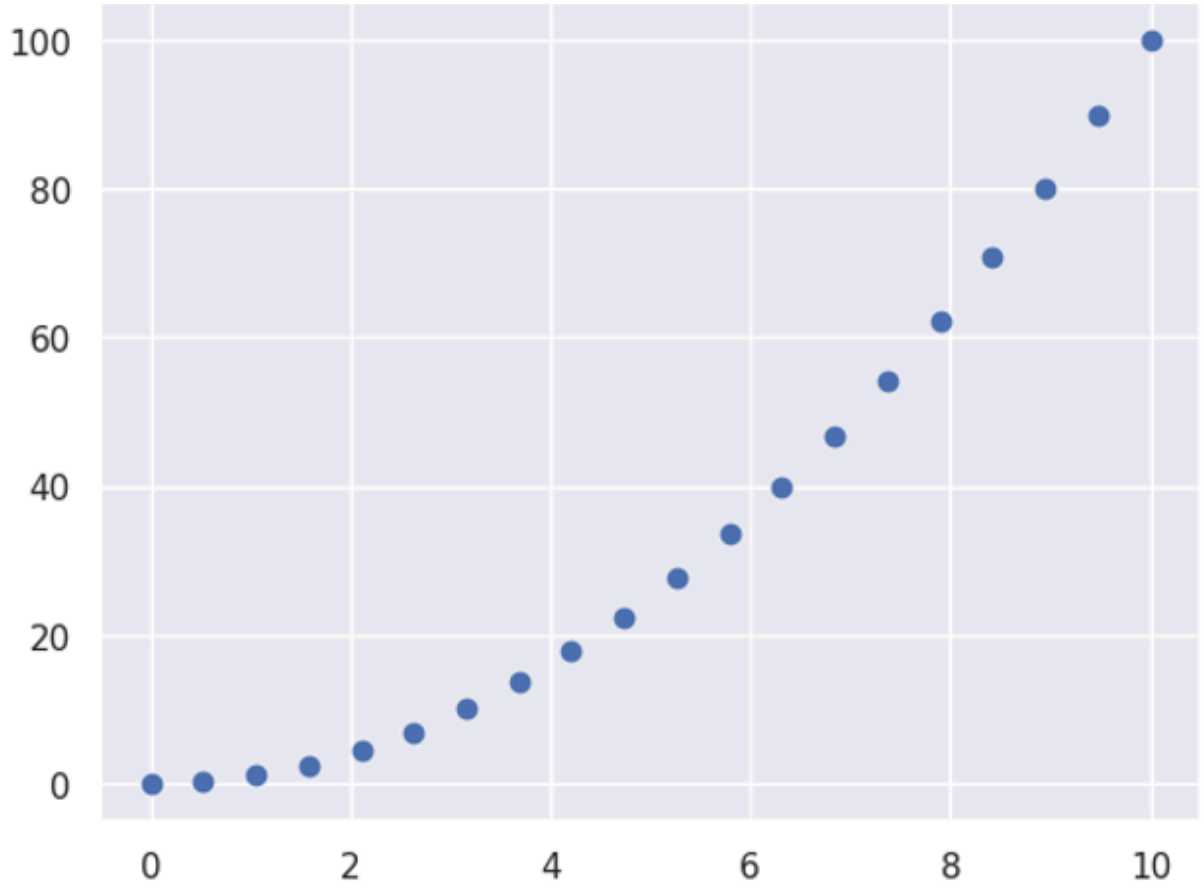


Figura 3: Gráfico de función cuadrática

A pesar de que el estilo se ve moderno, el gráfico necesita diferentes elementos para que se vea mejor. Estos elementos son: título, leyenda, unidades del eje X y unidades del eje Y. Si el gráfico es la posición de un vehículo con respecto al tiempo, entonces, el eje X debería informar de las unidades de medida en el tiempo y el eje Y las de distancia.

```
plt.scatter(x_val, y_val)
plt.title("Desplazamiento")  # título del gráfico
plt.xlabel("tiempo [segundos]")# etiqueta del eje x
plt.ylabel("Distancia[metros]")# etiqueta del eje y
```

El resultado de los cambios se muestra en la figura 4

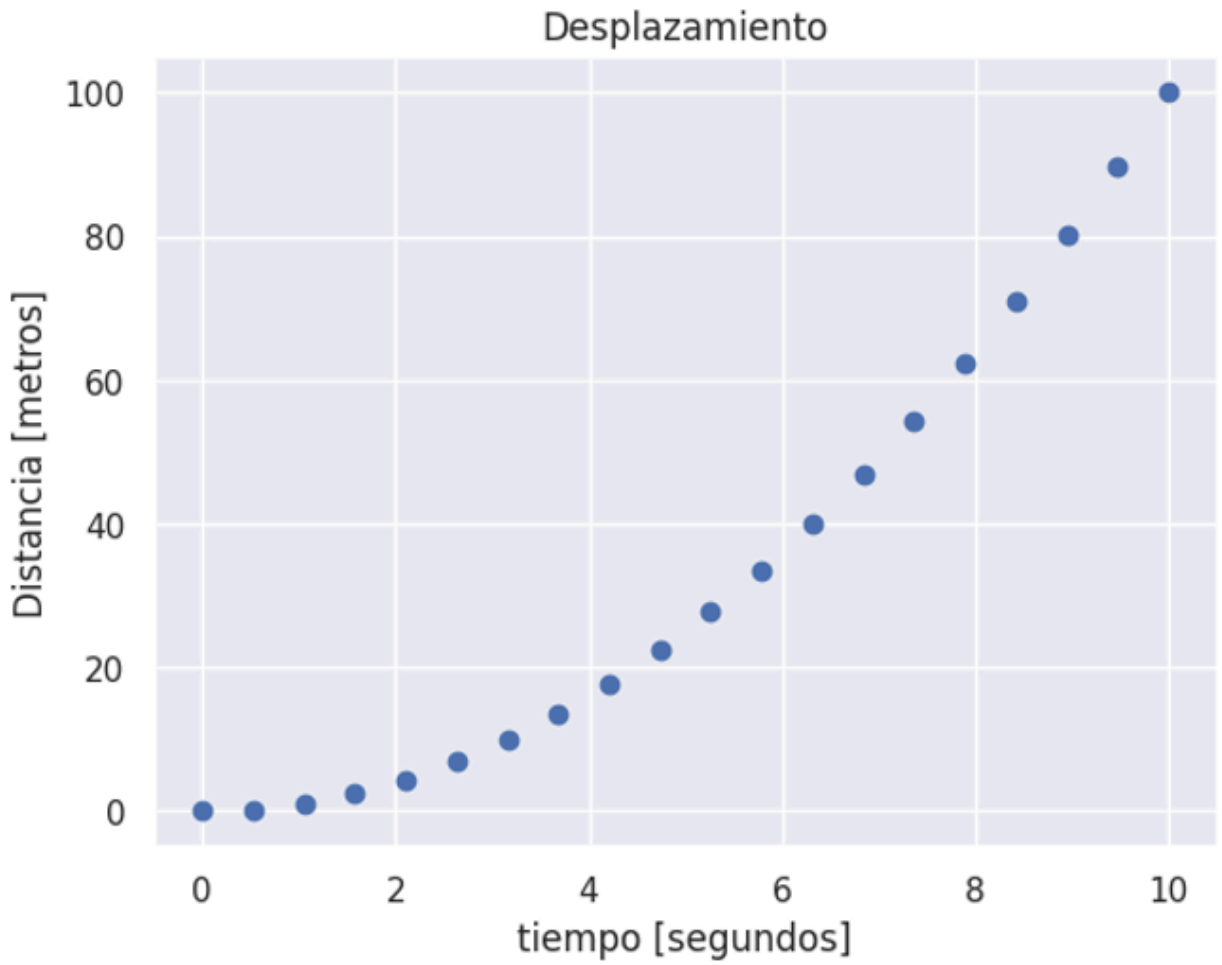
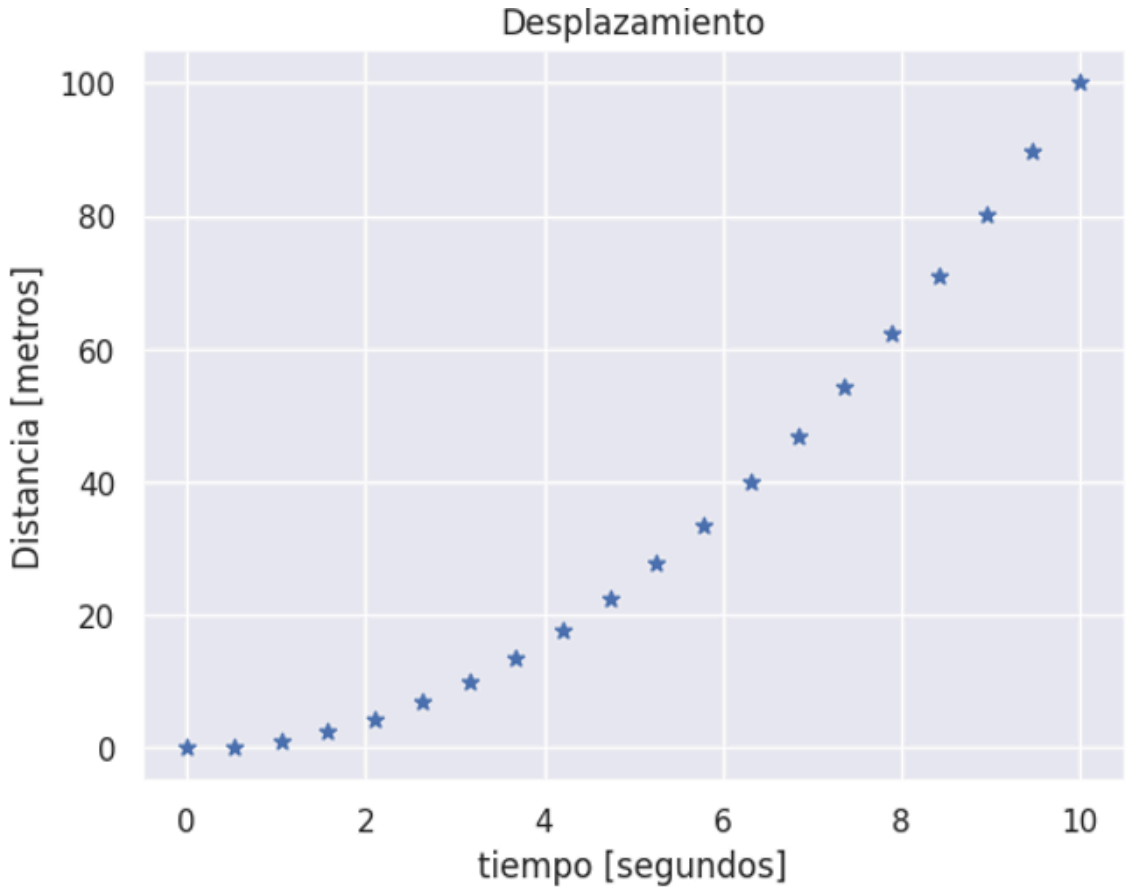


Figura 4: Gráfico con más detalles.

También es posible modificar los rótulos, por ejemplo, cambiarlos por estrellas o por otros símbolos. Para colocar marcas de estrella se debe emplear el símbolo * en el argumento marker, como se ve en el código. El resultado se muestra en la figura 5. Se puede intentar cambiar el siguiente código (en el valor asignado a marker) con las opciones:
[. , 0, v, ^, <, >, 1, 2, 3, 4, 8, s, p, P, h, H, +, x, X, |,]

```
plt.scatter(x_val, y_val, marker='*')
plt.title("Desplazamiento")
plt.xlabel("tiempo [segundos]")
plt.ylabel("Distancia [metros]")
```



Tanto a la función plot como a la función scatter se les puede agregar colores utilizando el parámetro opcional color, por ejemplo, para que los marcadores sean verdes, se puede emplear color='g', de esta forma la figura 6 tendría el resultado

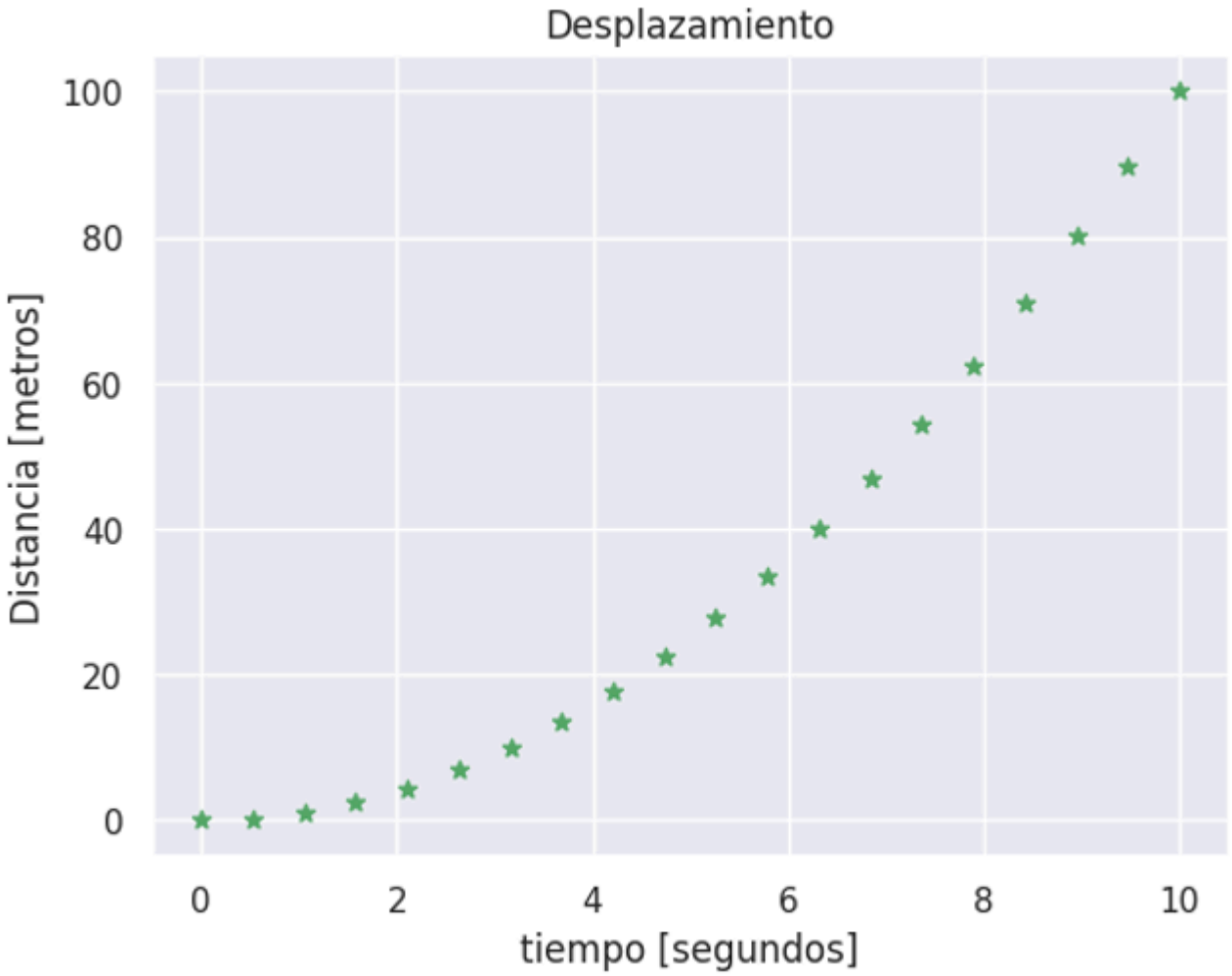


Figura 6: Gráfico con color verde.

También es posible, hacer múltiples cambios. En este caso se utiliza la función plot para dibujar una línea continua de color rojo (utilizando color='r') y la función scatter para dibujar los puntos. En este caso el marcador con una coma sirve para dibujar cuadrados y la letra g en el color indica que deben ser de color rojo. La s=150 sirve para cambiar el tamaño de los puntos a 150. El resultado se aprecia en la figura 7.

```
plt.plot(x_val, y_val, color='r')
plt.scatter(x_val, y_val, marker=',', color='g',
s=150)
plt.title("Desplazamiento")
plt.xlabel("tiempo [segundos]")
plt.ylabel("Distancia [metros]")
```

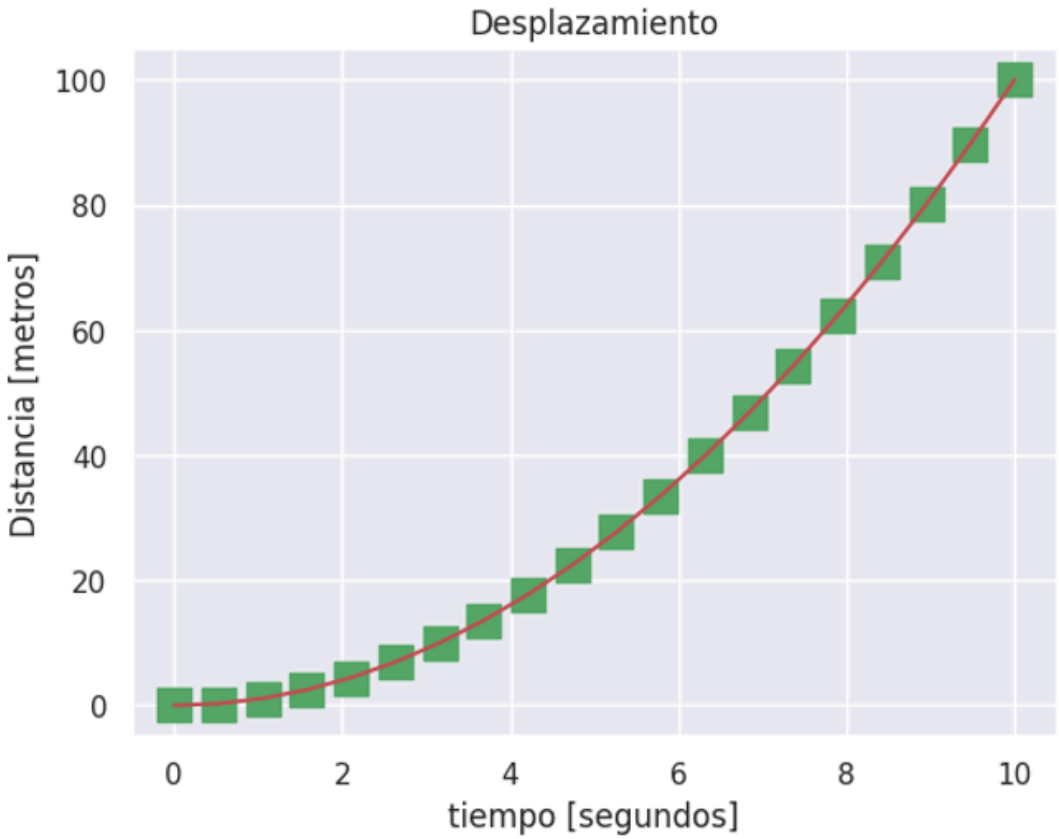


Figura 7: Gráfico con modificaciones

Matplotlib ofrece múltiples opciones para dar estilos a los gráficos, a las líneas y a los marcadores.

Histogramas:

Los histogramas permiten ver la distribución de una variable conforme se repiten sus medidas. Un ejemplo de gráfico de distribución se muestra en el siguiente código.

```
import pandas as pd
data = np.random.multivariate_normal([0, 0], [[5,
2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])
for col in 'xy':
    plt.hist(data[col],  alpha=0.5)
```

El código mostrado utiliza las bibliotecas Pandas y Matplotlib en Python para generar un conjunto de datos bidimensionales aleatorios y luego trazar histogramas para cada una de las dimensiones en el conjunto de datos. Cada una de las líneas hace lo siguiente:

import pandas as pd: Importa la biblioteca Pandas bajo el alias pd. Pandas es una biblioteca de Python que proporciona estructuras de datos y herramientas de análisis de datos.

data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000): Utiliza la función multivariate_normal de la biblioteca NumPy para generar datos aleatorios de una distribución normal multivariada. La función recibe como argumentos la media [0, 0], la matriz de covarianza [[5, 2], [2, 2]] y el tamaño de la muestra size=2000. Estos datos representarán las coordenadas x e y de los puntos.
data = pd.DataFrame(data, columns=['x', 'y']): Crea un DataFrame de Pandas llamado data utilizando los datos generados aleatoriamente. Se especifican los nombres de las columnas como 'x' y 'y'.
for col in 'xy': plt.hist(data[col], alpha=0.5): Utiliza un bucle for para iterar sobre las columnas 'x' y 'y' del DataFrame data. Para cada columna, traza un histograma utilizando la función hist de Matplotlib. El argumento alpha=0.5 especifica la transparencia de los histogramas para que las superposiciones sean visibles. El resultado del gráfico se muestra en la figura 8.

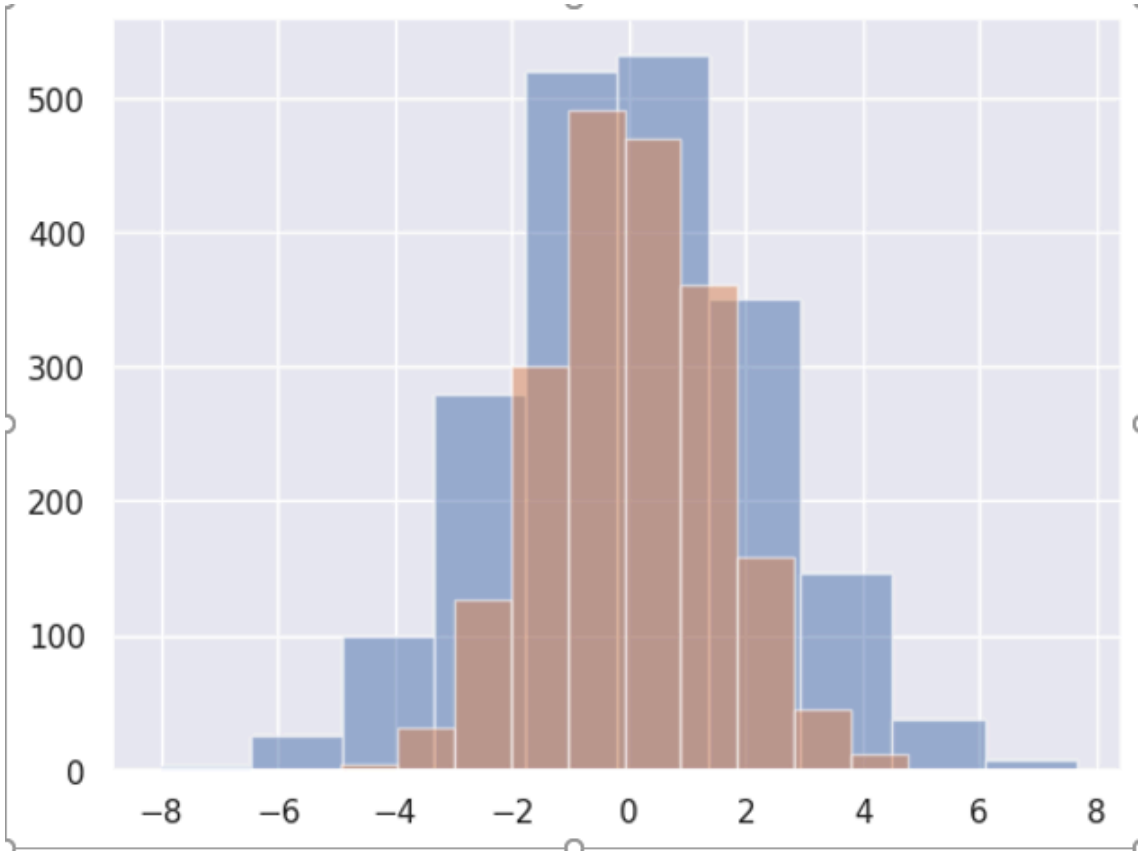


Figura 8: gráfico de distribuciones generadas.

En lugar de un histograma, se pueden dibujar estimaciones suavizadas de la estimación de la distribución de probabilidad con la función kdeplot de seaborn:

```
for col in 'xy':
    sns.kdeplot(data[col], shade=True)
```

El resultado se muestra en la figura 9:

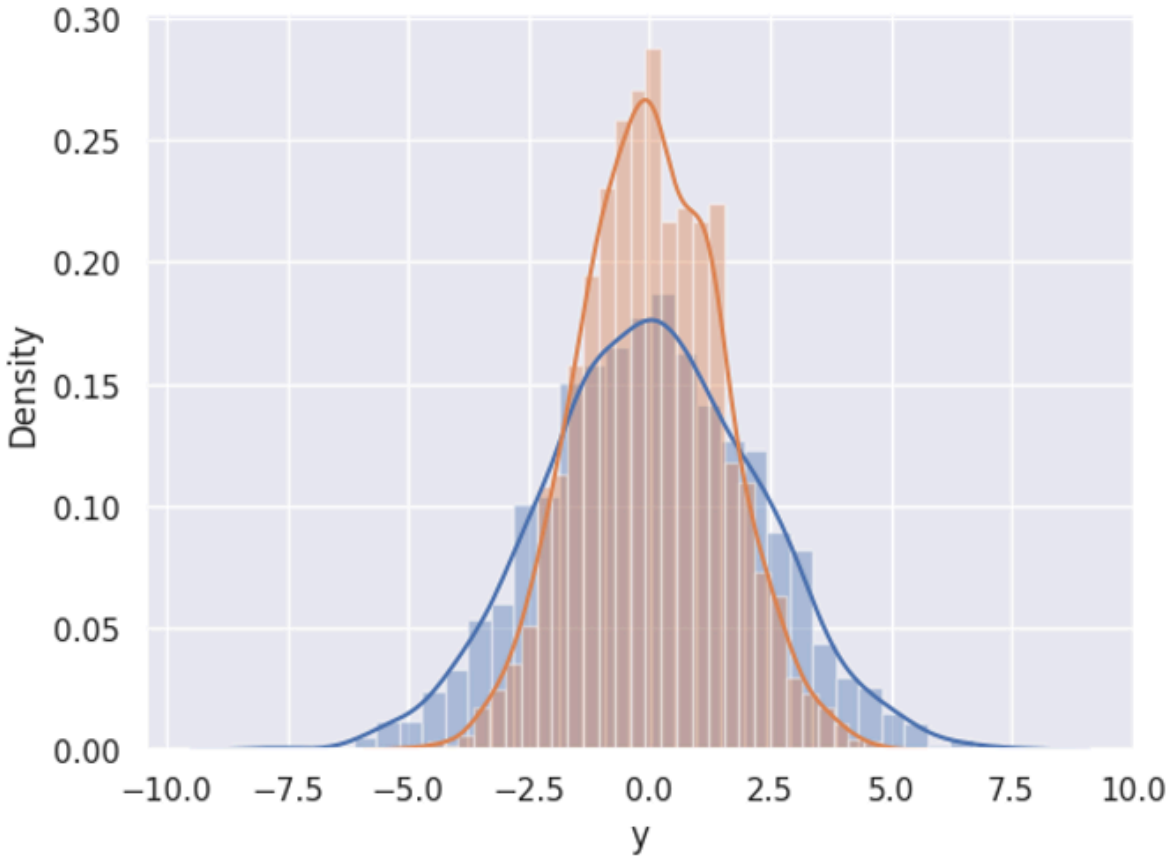


Figura 10: Gráficos superpuestos

Análisis gráfico

Otra de las ventajas de seaborn y matplotlib es que incluyen ciertos conjuntos de datos de exploración. El conjunto de datos iris relacionado con las flores de tipo iris setosa, iris versicolor e iris virgínica, está previamente incluido. Una forma de graficar múltiples variables es con el gráfico pairplot, este análisis permite explorar correlaciones entre muchas dimensiones de los datos y compararlos de forma fácil.

```
iris = sns.load_dataset("iris")
iris.head()
sns.pairplot(iris, hue='species', size=2.5);
```

En la figura 11 se muestra la comparación generada

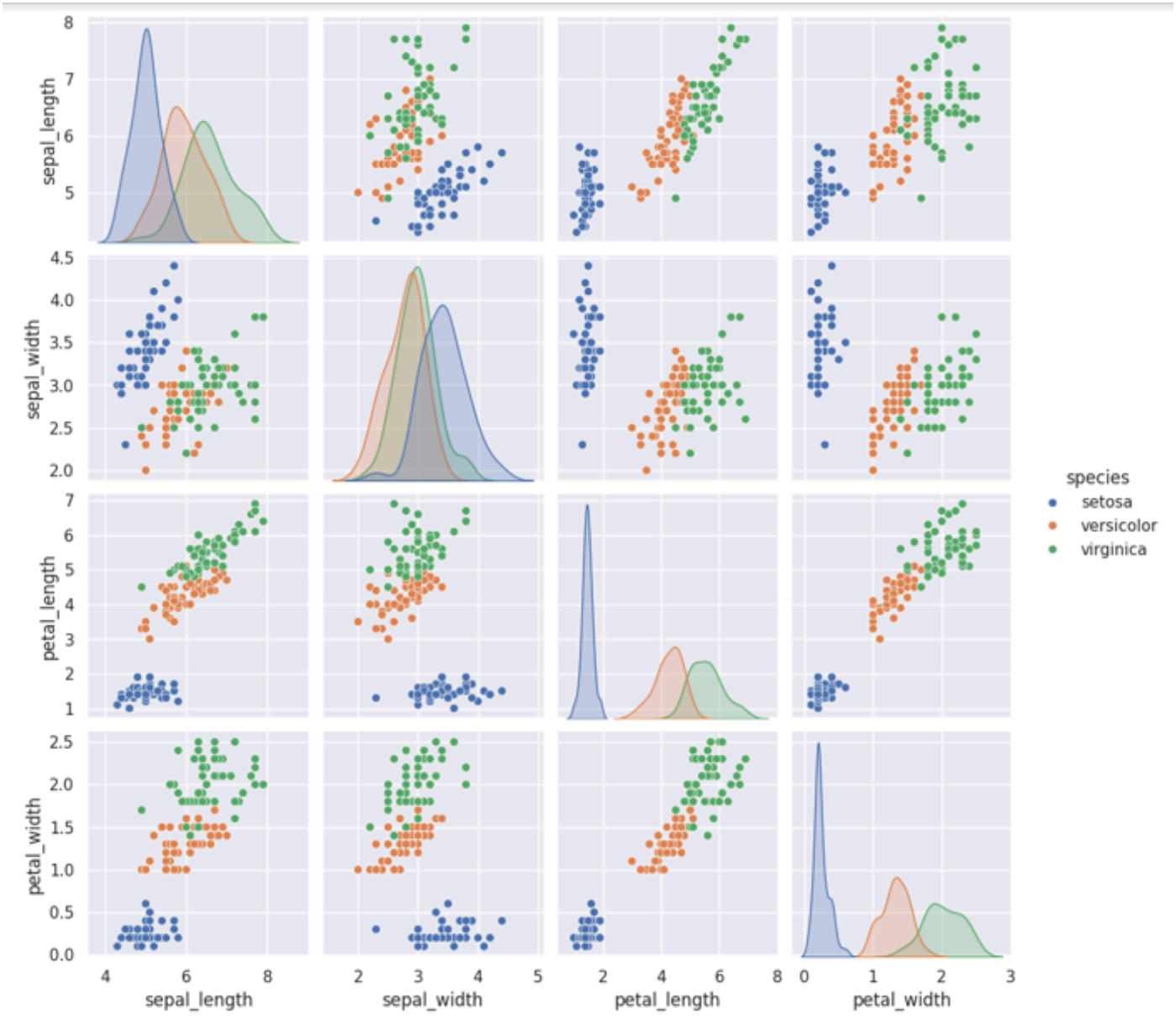


Figura 11: Análisis pairplot sobre el dataset iris

Con los datos de la figura 11 es posible mostrar que el ancho de pétalos permite separar las especies fácilmente, también es posible ver que esta situación se parece al largo del pétalo. Sin embargo, se ve que con el ancho del sépalo no es fácil dividir las flores por tipo.

Se propone como actividad realizar la visualización de datos del dataset titanic siguiendo el cuaderno del link:

https://gist.github.com/mwaskom/8224591