graficas

October 24, 2024

1 Gráficas con matplotlib

[2]:
```
# Importar la libreria matplotlib.pyplot con el nombre de plt (esto es un␣
 ↪estandar en la comunidad):

import matplotlib.pyplot as plt
```

```
Matplotlib is building the font cache; this may take a moment.
```

[5]:
```
# Para ver las graficas directamente en este notebook se debe hacer con este␣
 ↪comando:
%matplotlib inline
```

[6]:
```
import warnings; warnings.simplefilter('ignore')
```

[7]:
```
# Veamos un ejemplo muy simple usando dos arreglos numpy.
import numpy as np
x = np.linspace(0,5, 11)
y = x ** 2


# La función np.linspace() de NumPy para generar un array de números␣
 ↪equidistantes dentro de un intervalo especificado.

# 0: Este es el valor inicial del intervalo. En este caso, se establece en 0.
# 5: Este es el valor final del intervalo. En este caso, se establece en 5.
# 11: Este es el número de elementos que se generarán en el array. En este␣
 ↪caso, se generarán 11 elementos.

# Entonces, np.linspace(0, 5, 11) creará un array NumPy de 11 números␣
 ↪equidistantes que comienzan en 0 y terminan en 5 (incluyendo ambos extremos).
#Específicamente, el resultado será un array con los siguientes valores:
# [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
# Observa que el espaciado entre los elementos es de 0.5, ya que el intervalo␣
 ↪total (5 - 0 = 5) se divide en 10 partes iguales para obtener 11 elementos.
```

```
#Esta línea es útil cuando se necesita crear un conjunto de valores␣
 ↪equidistantes dentro de un rango específico,
# ya sea para graficar, evaluar funciones, o cualquier otro propósito que␣
 ↪requiera un muestreo uniforme del intervalo.
```

[8]:
```
x
```

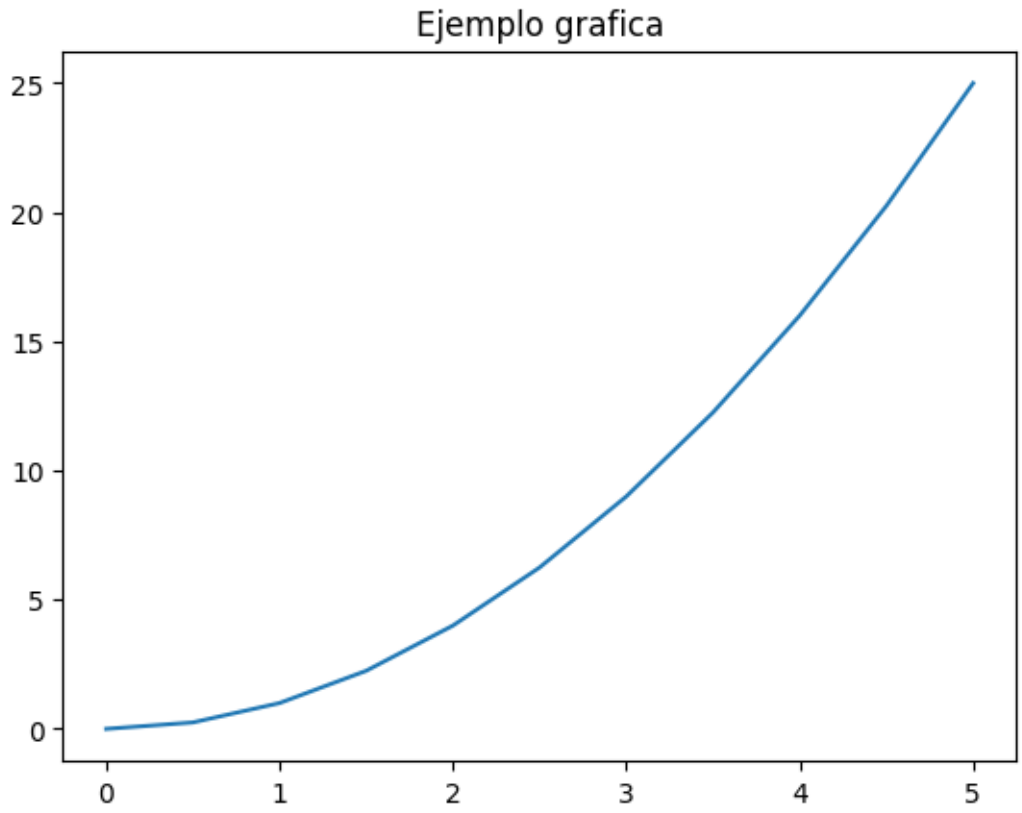
[8]:
```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

[9]:
```
y
```

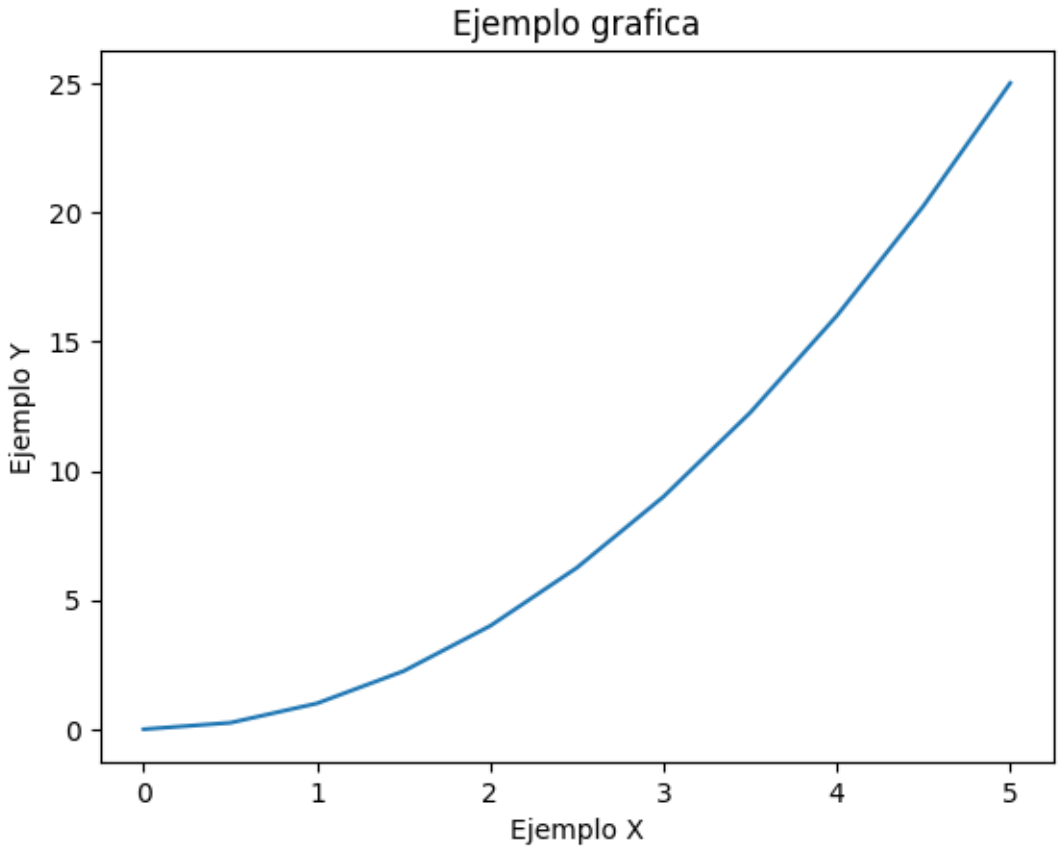
[9]:
```
array([ 0.  ,  0.25,  1.  ,  2.25,  4.  ,  6.25,  9.  , 12.25, 16.  ,
       20.25, 25.  ])
```

[10]:
```
# Podemos crear un diagrama de líneas muy simple usando lo siguiente:
plt.plot(x, y)
# definir el titulo de la grafica
plt.title('Ejemplo grafica')
```

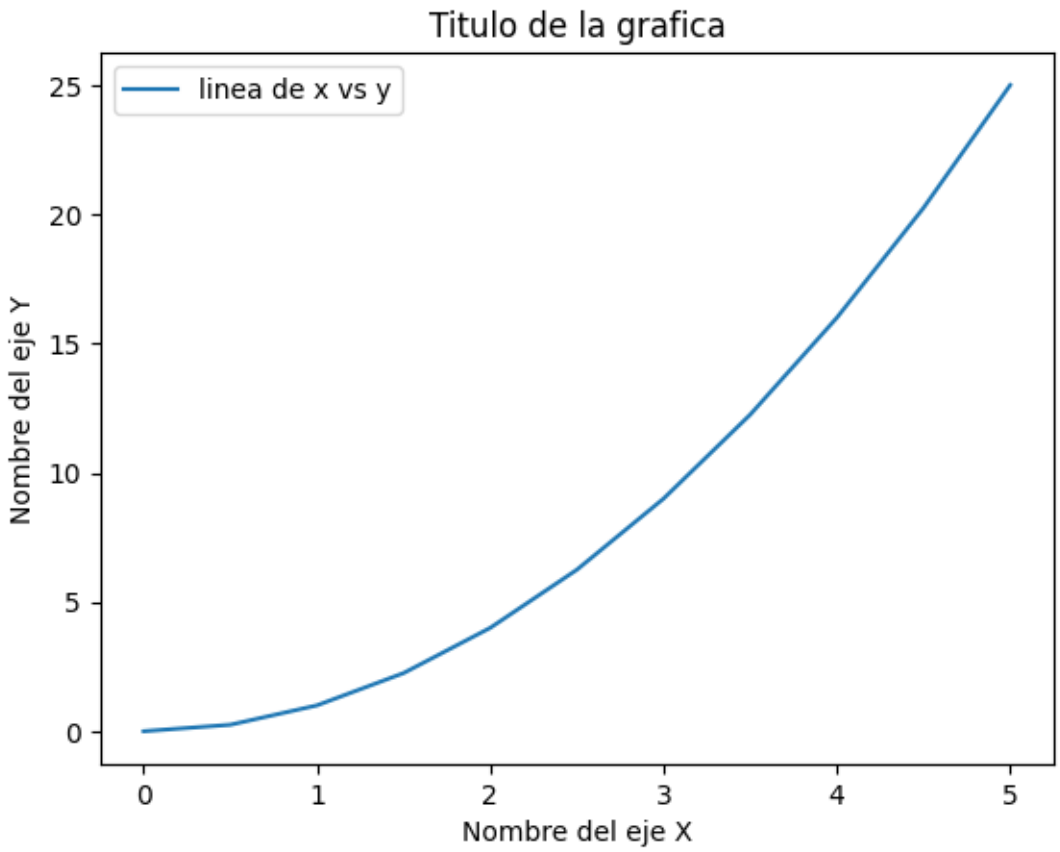
[10]:
```
Text(0.5, 1.0, 'Ejemplo grafica')
```



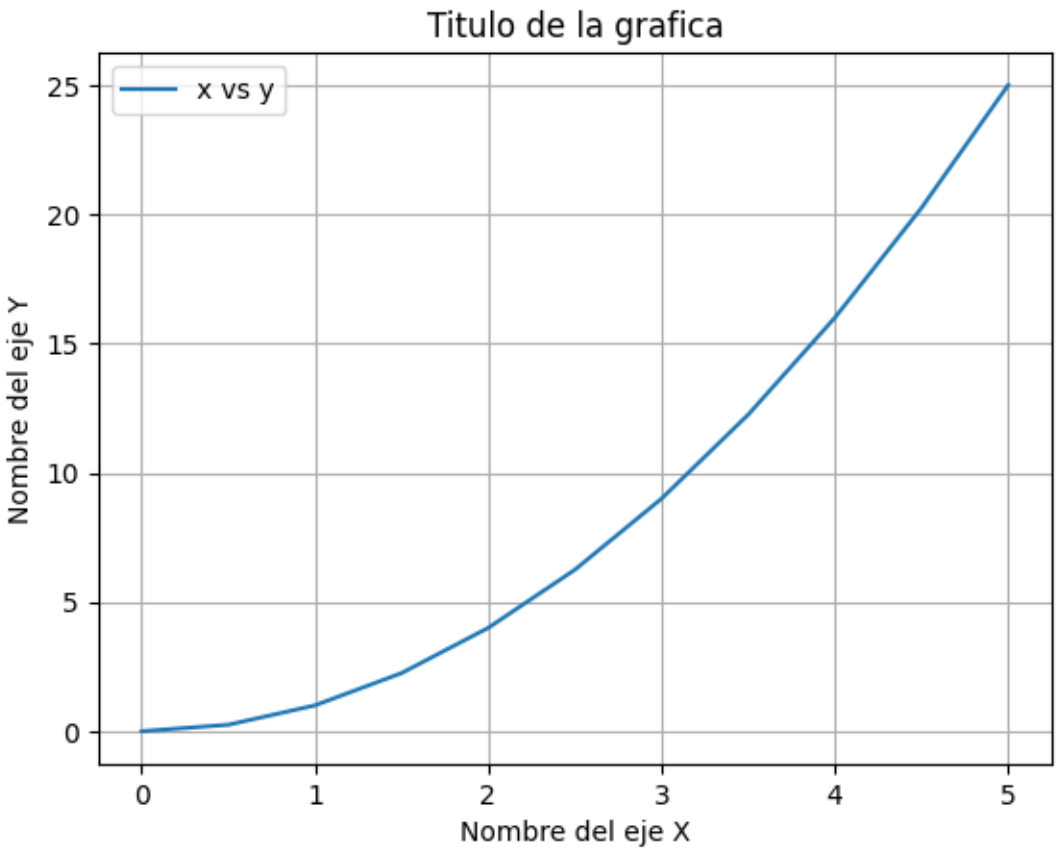
[11]:
```
#Nombres de los ejes
plt.plot(x, y) # se grafica una linea de color azul
plt.xlabel('Ejemplo X') # definir el nombre del eje X
plt.ylabel('Ejemplo Y') # definir el nombre del eje Y
plt.title('Ejemplo grafica'); # definir el titulo de la grafica
```



[12]:
```
#Legend

plt.plot(x, y, label="linea de x vs y") # se grafica una linea de color azul
# se pone en el atributo 'label' el texto deseado

plt.xlabel('Nombre del eje X') # definir el nombre del eje X
plt.ylabel('Nombre del eje Y') # definir el nombre del eje Y
plt.title('Titulo de la grafica') # definir el titulo de la grafica
plt.legend(); # agregar el legend al plot
```



[13]:
```
# cuadricula (grid)
plt.plot(x, y, label="x vs y") # se grafica una linea de color azul
# se pone en el atributo 'label' el texto deseado

plt.xlabel('Nombre del eje X') # definir el nombre del eje X
plt.ylabel('Nombre del eje Y') # definir el nombre del eje Y
plt.title('Titulo de la grafica') # definir el titulo de la grafica
plt.legend() # agregar el legend al plot

plt.grid(True) # poner grid en la grafica
```



2 Tamaño de la Figura y DPI

Matplotlib permite especificar la relación de aspecto, el DPI y el tamaño de la figura cuando se crea el objeto Figure. Puede usar los argumentos de las palabras clave figsize ydpi. No es necesario poner las dos.

- figsize es una tupla del ancho y alto de la figura en pulgadas
- dpi es el punto por pulgada (pixel por pulgada).

[15]:
```
# se cambia el tamaño de la figura y el numero de puntos por pulgada
plt.figure(figsize=(8,4), dpi=150)

plt.plot(x, y) # se grafica una linea de color azul

plt.xlabel('Nombre del eje X') # definir el nombre del eje X
plt.ylabel('Nombre del eje Y') # definir el nombre del eje Y
plt.title('Titulo de la grafica') # definir el titulo de la grafica
```

[15]:
```
Text(0.5, 1.0, 'Titulo de la grafica')
```

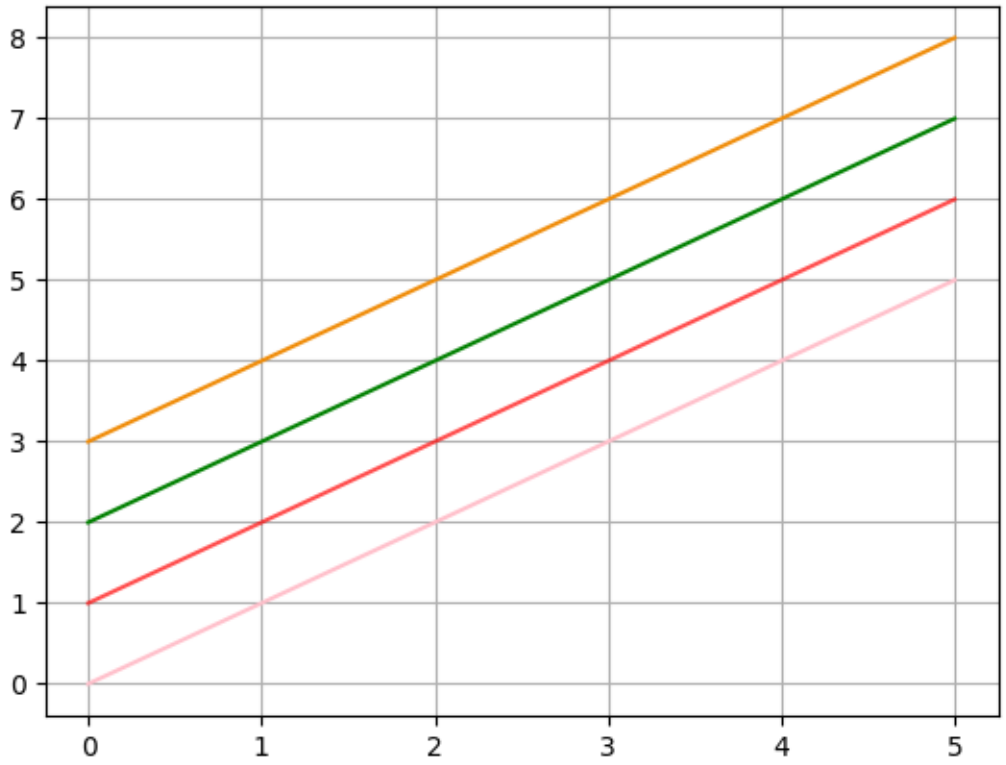
3 Parametros de las lineas: colores, ancho y tipos

Matplotlib brinda muchas opciones para personalizar colores, anchos de línea y tipos de línea.

3.1 Existe la sintaxis básica que se puede consultar en:

https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

[18]:
```
# Colores Basicos
# Con matplotlib, podemos definir los colores de las líneas y otros elementos␣
 ↪gráficos de varias maneras.
# Podemos definir colores por sus nombres o códigos hexadecimales RGB y,␣
 ↪opcionalmente,
# proporcionar un valor alpha utilizando los argumentos de palabras clave color␣
 ↪y alpha.
# Alpha indica opacidad.

plt.plot(x, x, color="pink")
plt.plot(x, x+1, color="red", alpha=0.7) # Medio transparente
plt.plot(x, x+2, color="green")        # RGB hex code
plt.plot(x, x+3, color="#F08C08");       # RGB hex code
plt.grid(True) # poner grid en la grafica
```

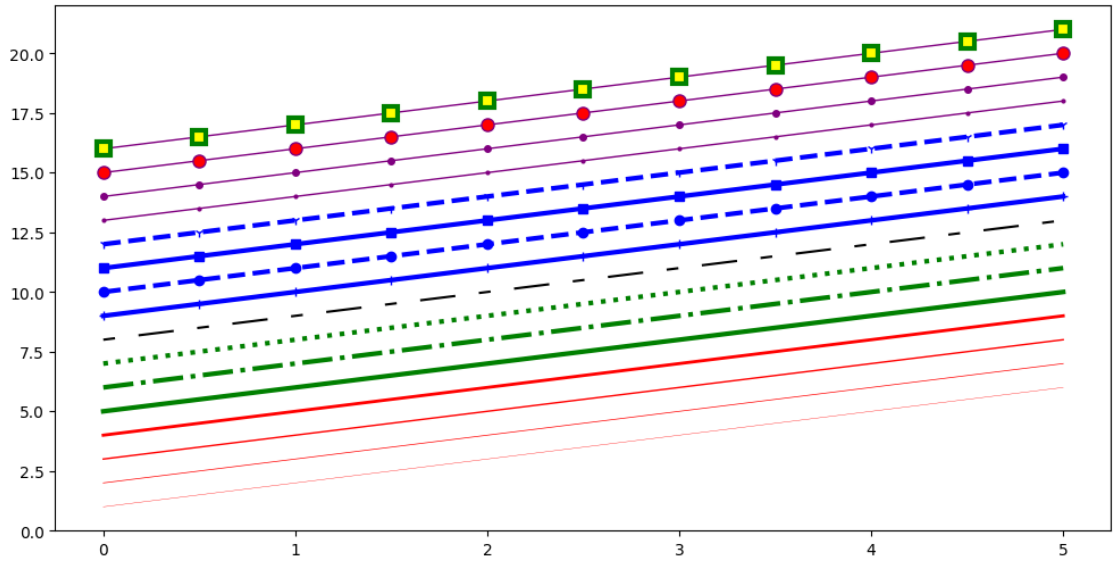


4 Estilos de Lineas y marcadores

- Para cambiar el ancho de línea, podemos usar el argumento de la palabra clave linewidth olw.
- El estilo de línea se puede seleccionar usando los argumentos de palabras clave linestyle ols:

[19]:
```
plt.subplots(figsize=(12,6))

plt.plot(x, x+1, color="red", linewidth=0.25)
plt.plot(x, x+2, color="red", linewidth=0.50)
plt.plot(x, x+3, color="red", linewidth=1.00)
plt.plot(x, x+4, color="red", linewidth=2.00)

# posibles opciones linestype '-', '--', '-.', ':', 'steps'
plt.plot(x, x+5, color="green", lw=3, linestyle='-')
plt.plot(x, x+6, color="green", lw=3, ls='-.')
plt.plot(x, x+7, color="green", lw=3, ls=':')

# lineas parametrizadas
line, = plt.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # formato: longitud de linea, longitud de␣
 ↪espacio, ...
```

```
# posibles simbolos del marcas: marker = '+', 'o', '*', 's', ',', '.',bb '1',␣
 ↪'2', '3', '4', ...
plt.plot(x, x+ 9, color="blue", lw=3, ls='-', marker='+')
plt.plot(x, x+10, color="blue", lw=3, ls='--', marker='o')
plt.plot(x, x+11, color="blue", lw=3, ls='-', marker='s')
plt.plot(x, x+12, color="blue", lw=3, ls='--', marker='1')

# tamaño y color de la marca
plt.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
plt.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
plt.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8,␣
 ↪markerfacecolor="red")
plt.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=3, markeredgecolor="green");
```

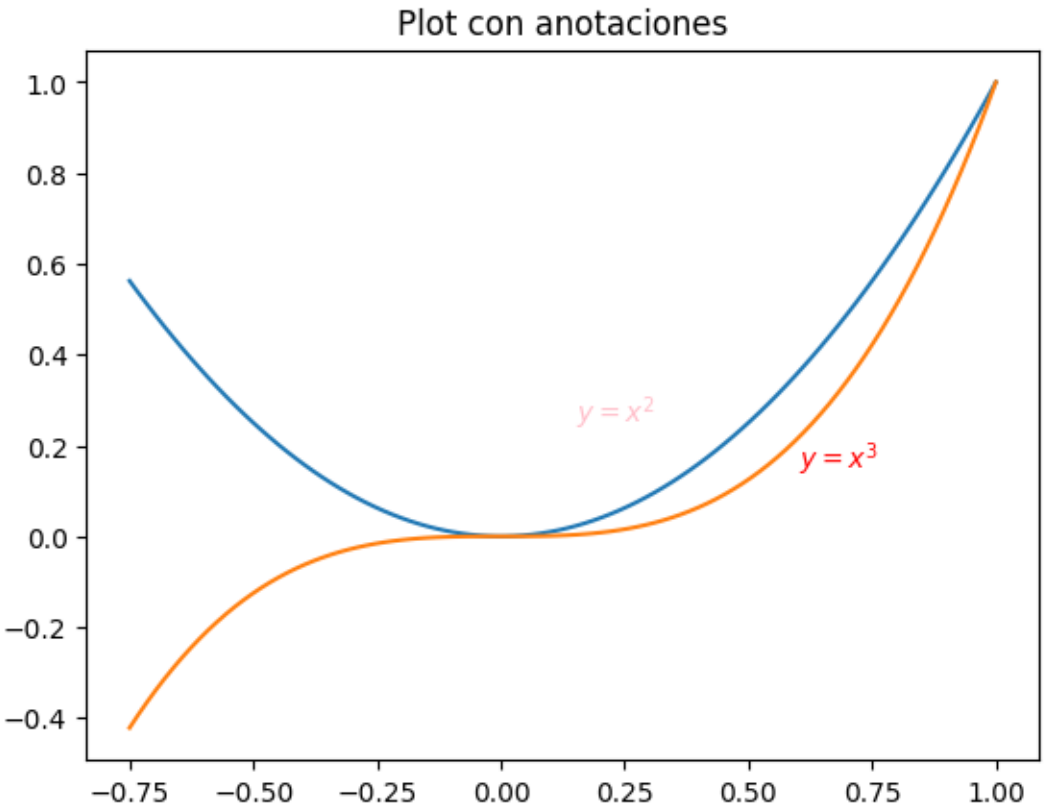


5 Anotaciones de texto

Anotar texto en figuras matplotlib se puede hacer usando la función text. Es compatible con el formato LaTeX al igual que los textos y títulos de la etiqueta del eje:

[20]:
```
# Datos para graficar
xx = np.linspace(-0.75, 1., 100)

plt.plot(xx, xx**2, xx, xx**3)
plt.title("Plot con anotaciones")
```

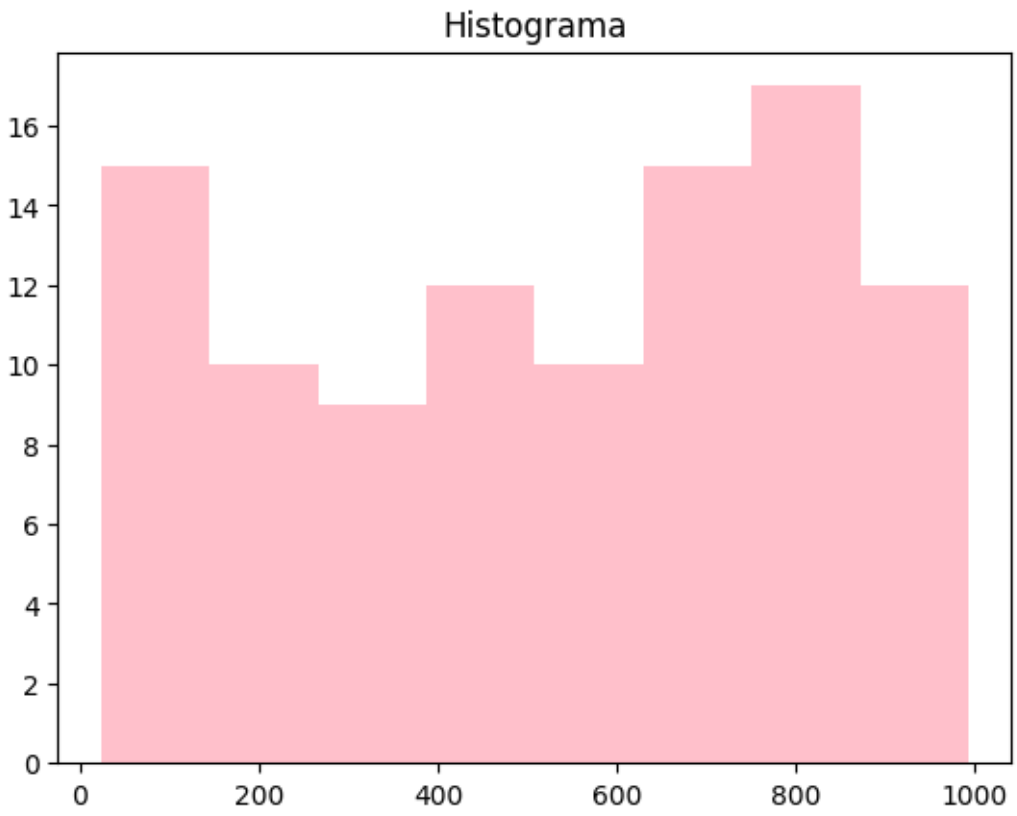
```
# Anotacion 1
plt.text(0.15, 0.25, r"$y=x^2$", fontsize=10, color="pink")
#Anotacion 2
plt.text(0.60, 0.15, r"$y=x^3$", fontsize=10, color="red");
```



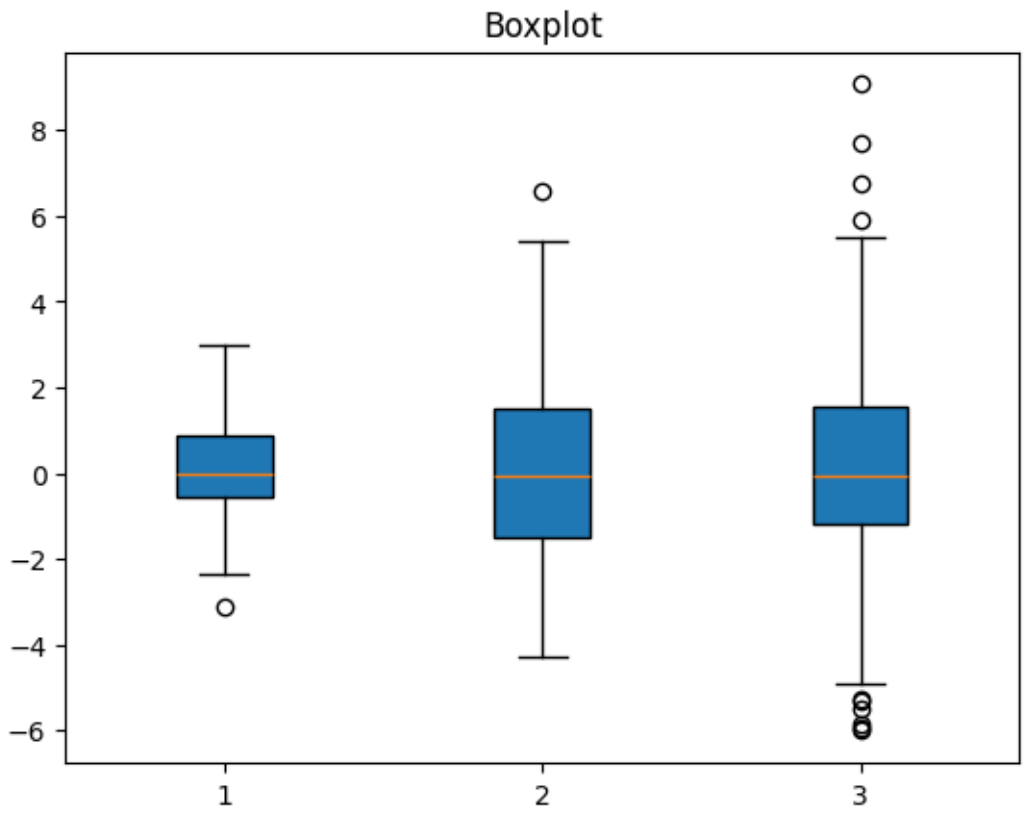
6 Tipos Especiales de Plots

- Hay muchas Graficas especializadas que podemos crear, como barras, histogramas, diagramas de dispersión y mucho más.

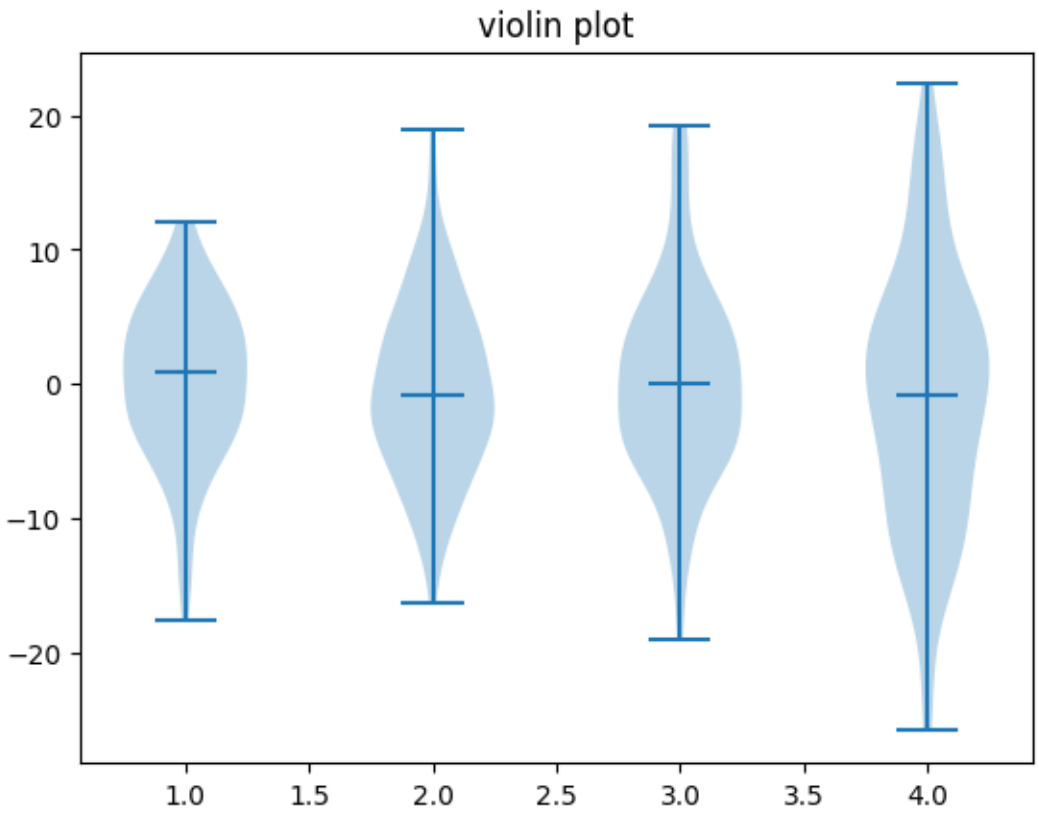
[22]:
```
# Histograma

# crear datos aleatorios
from random import sample
data = sample(range(1, 1000), 100)

plt.hist(data,bins = 8, color="pink") # bins el numero de divisiones del␣
 ↪histograma
plt.title("Histograma");
```



[25]:
```
# Boxplot

#crear datos aleatorios
data = [np.random.normal(0, std, 100) for std in range(1, 4)]
#Esta línea crea una lista data con tres arrays NumPy de 100 elementos cada␣
 ↪uno, donde cada array contiene números aleatorios
# normalmente distribuidos con una media de 0 y desviaciones estándar de 1, 2 y␣
 ↪3, respectivamente.

# boxplot rectangular
plt.boxplot(data,vert=True,patch_artist=True);
# Esta línea utiliza la función boxplot de la librería Matplotlib para crear un␣
 ↪gráfico de caja y bigotes (box plot)
# de los datos en la variable data. Los parámetros especifican que el gráfico␣
 ↪se dibuje verticalmente y que los cuadros se rellenen con un color sólido.

plt.title("Boxplot");
```

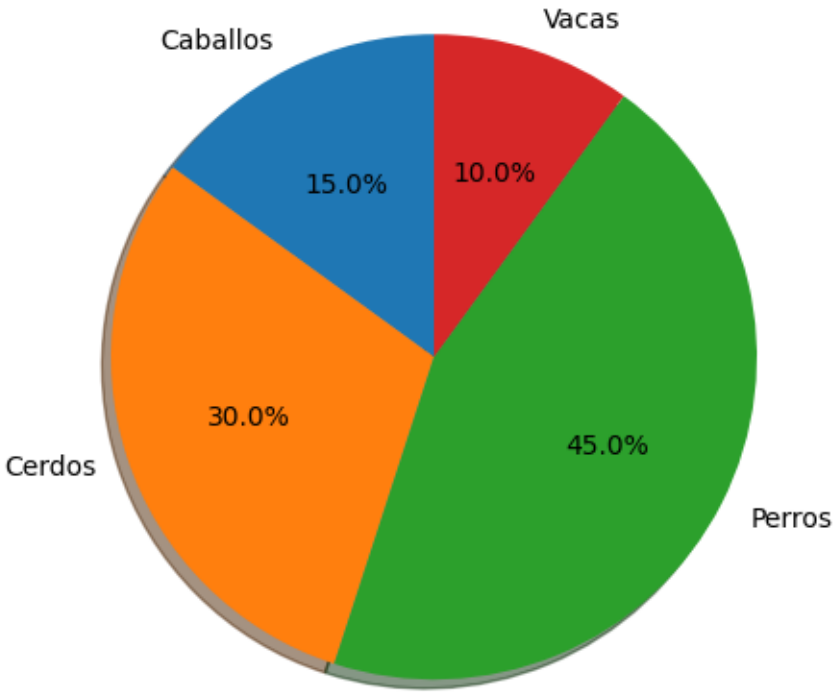


[27]:
```
#Diagramas de Violin
#Permiten ver como es la distribucion de los datos

all_data = [np.random.normal(0, std, 100) for std in range(6, 10)]

# grafico de violin, se puede activar la visualizacion de la media y de la␣
  ↪mediana
plt.violinplot(all_data, showmeans=False, showmedians=True)
plt.title('violin plot');
```



[30]:
```
# Pastel
labels = 'Caballos', 'Cerdos', 'Perros', 'Vacas'
sizes = [15, 30, 45, 10]
#explode = (0, 0.1, 0, 0)

# Los datos se definen mediante tres listas: labels contiene las etiquetas para␣
 ↪cada porción del pie,
# sizes especifica el tamaño relativo de cada porción, y explode indica qué␣
 ↪porción debe separarse ligeramente del resto
# (en este caso, la porción correspondiente a "Cerdos"). La función plt.pie()␣
 ↪genera el gráfico con los datos proporcionados,
# incluyendo etiquetas de porcentaje, sombra y un ángulo de inicio específico.

#plt.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
#        shadow=True, startangle=90)
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)

plt.axis('equal')  #La relación de aspecto igual garantiza que el círculo sea␣
 ↪homogeneo
```

```
plt.show()
```



7 Guardando las figuras

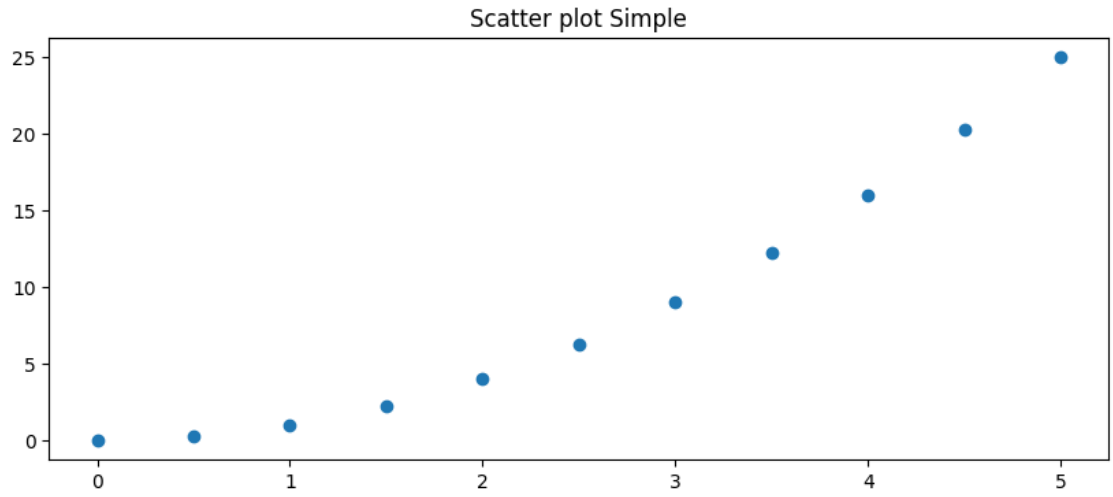
Matplotlib puede generar resultados de alta calidad en varios formatos, incluidos PNG, JPG, EPS, SVG, PGF y PDF.

-Para guardar una figura en un archivo, podemos usar el método savefig de la claseFigure:

-Lo primero es antes de crear una grafica definir la clase Figure al principio de todo la grafica, Ejemplo:

fig = plt.figure(figsize=(10,4)) plt.scatter(x, y) plt.title("Scatter plot Simple");

[31]:
```
fig = plt.figure(figsize=(10,4))
plt.scatter(x, y)
plt.title("Scatter plot Simple");
fig.savefig("figura.png")
```



[32]:
```
# Aquí también podemos especificar opcionalmente el DPI y elegir entre␣
  ↪diferentes formatos de salida (PNG, JPG, EPS, SVG, PGF y PDF):
fig.savefig("figura.pdf", dpi=50)
```

[ ]:
```

```