Introducción a la Computación Científica con Python
Clase 4:
Matplotlib – Gráficas en Python

Diego Passarella **Víctor Viana**

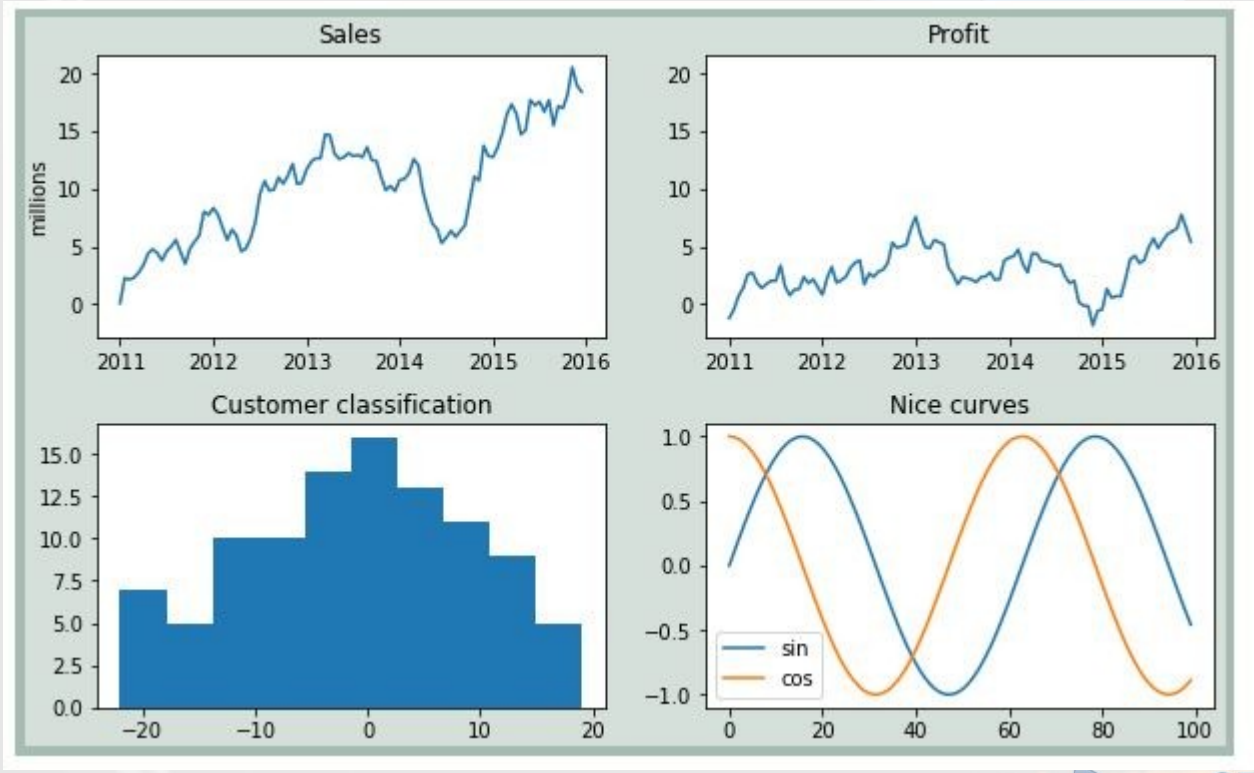
EDUCACION PERMANENTE Universidad de la República
SEDE TACUAREMBÓ CENUR NORESTE UNIVERSIDAD DE LA REPÚBLICA URUGUAY

¿Qué es matplotlib?

- Es la librería de visualización más popular de Python
- El conocimiento de esta librería sigue siendo fundamental en cualquier proyecto de Data Science.
- Matplotlib y sus dependencias están disponibles como paquetes para las distribuciones de MacOS, Windows y Linux:

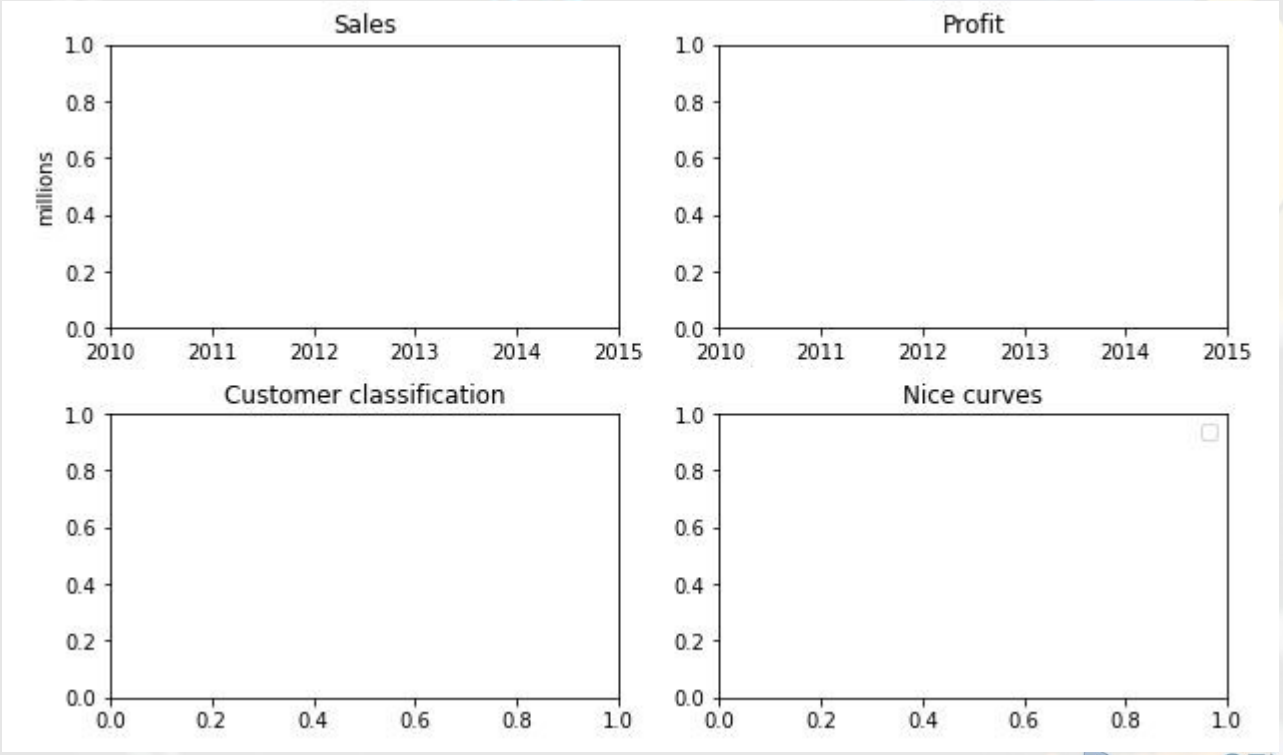
```
python -m pip install -U pip

python -m pip install -U matplotlib
```

Nomenclatura

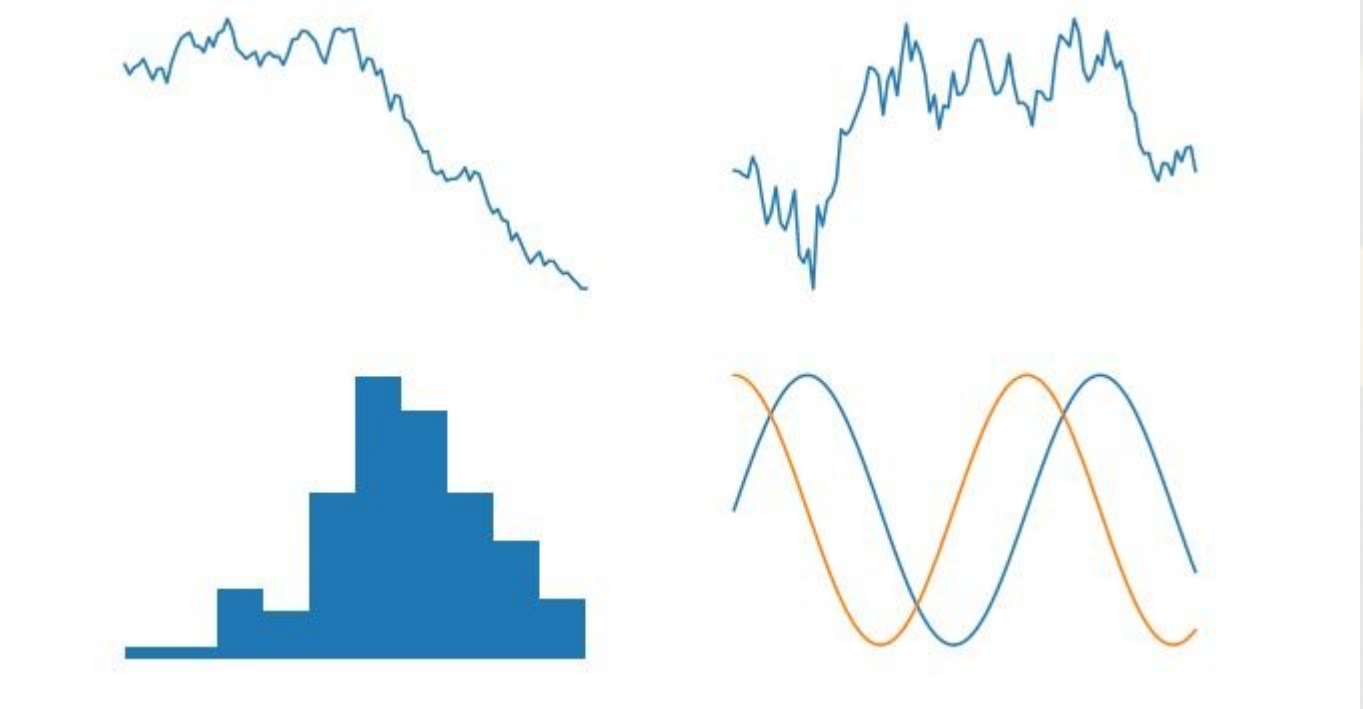


Nomenclatura - figure

Nomenclatura - axes

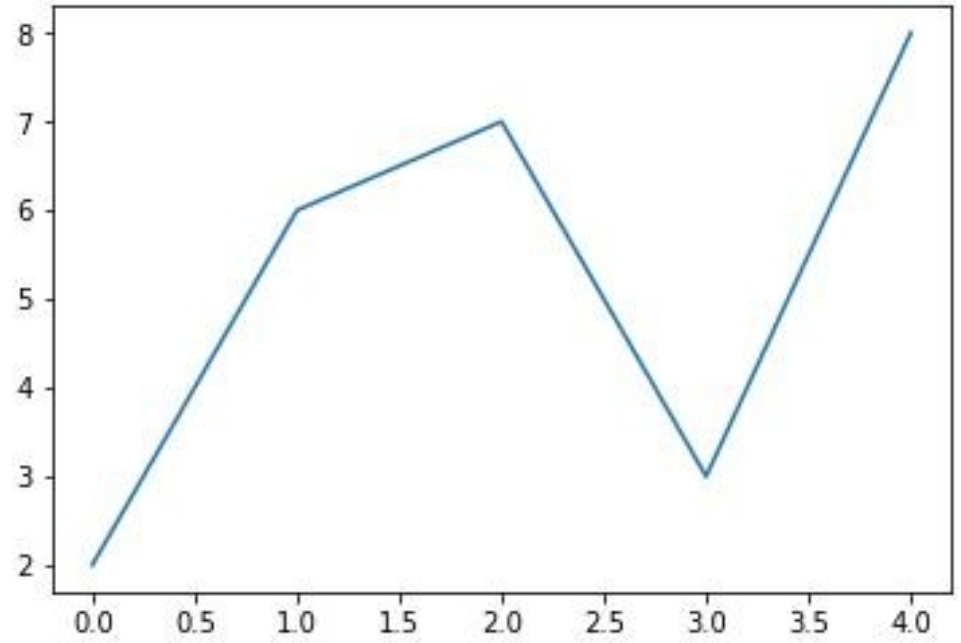


Nomenclatura - plot



La función plot

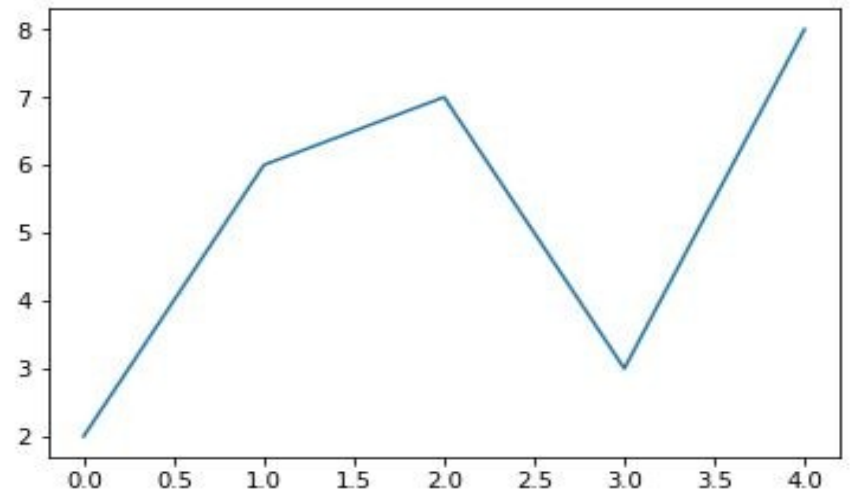
```
plt.plot([2, 6, 7, 3, 8])
plt.show()
```



La función subplots

Crea una figura y uno o más conjuntos de ejes (por defecto se crea uno solo)

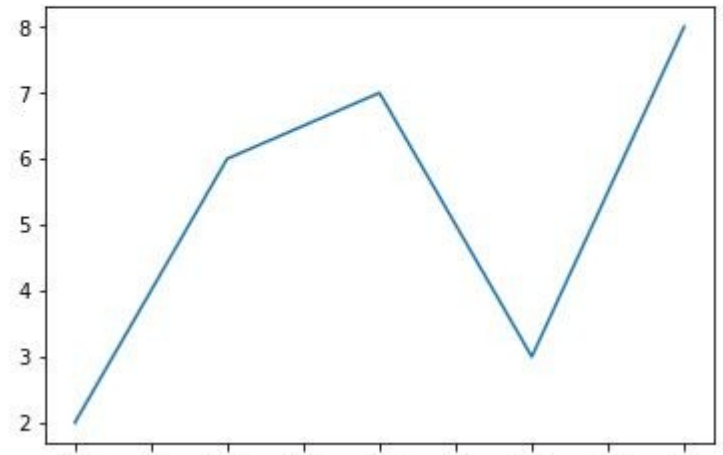
```
fig, ax = plt.subplots()
ax.plot([2, 6, 7, 3, 8])
plt.show()
```



Las funciones figure y axes

- **figure** crea de forma explícita una figura
- **axes** crea de forma explícita un conjunto de ejes en la última figura que se haya creado o referenciado

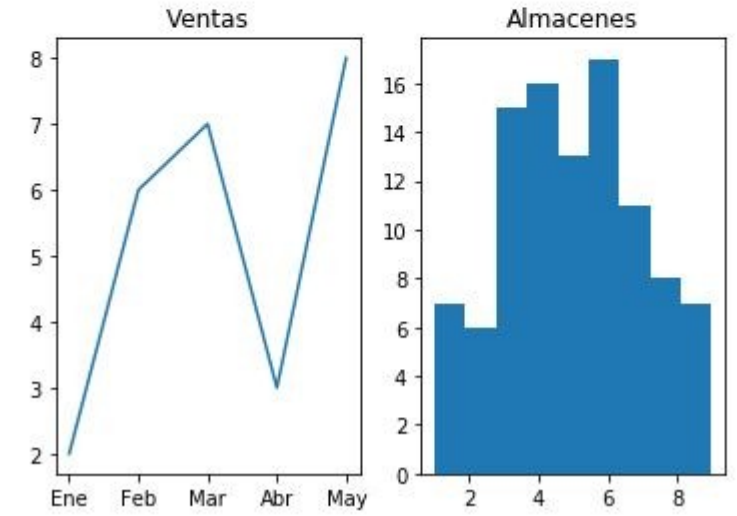
```
fig = plt.figure()
ax = plt.axes()
ax.plot([2, 6, 7, 3, 8])
plt.show()
```



Interfaces de programación

Interfaz estilo "MATLAB"

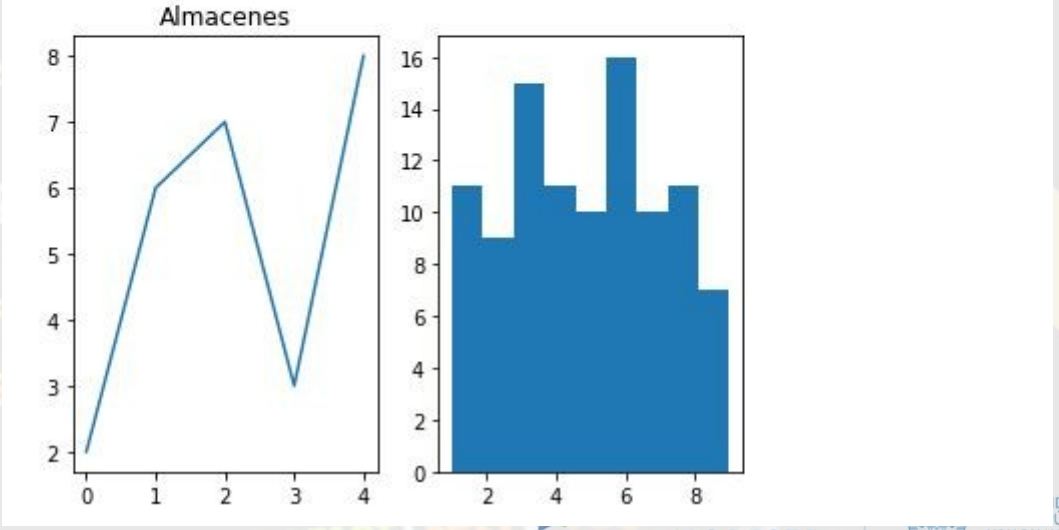
```
plt.figure()
plt.subplot(1, 2, 1)
plt.plot([2, 6, 7, 3, 8])
plt.title("Ventas")
plt.xticks(range(0,5), ["Ene", "Feb", "Mar", "Abr", "May"])
plt.subplot(1, 2, 2)
plt.hist(np.random.randint(1, 10, 100), bins = 9)
plt.title("Almacenes")
plt.show()
```



Interfaces de programación

Interfaz orientada a objetos (OO)

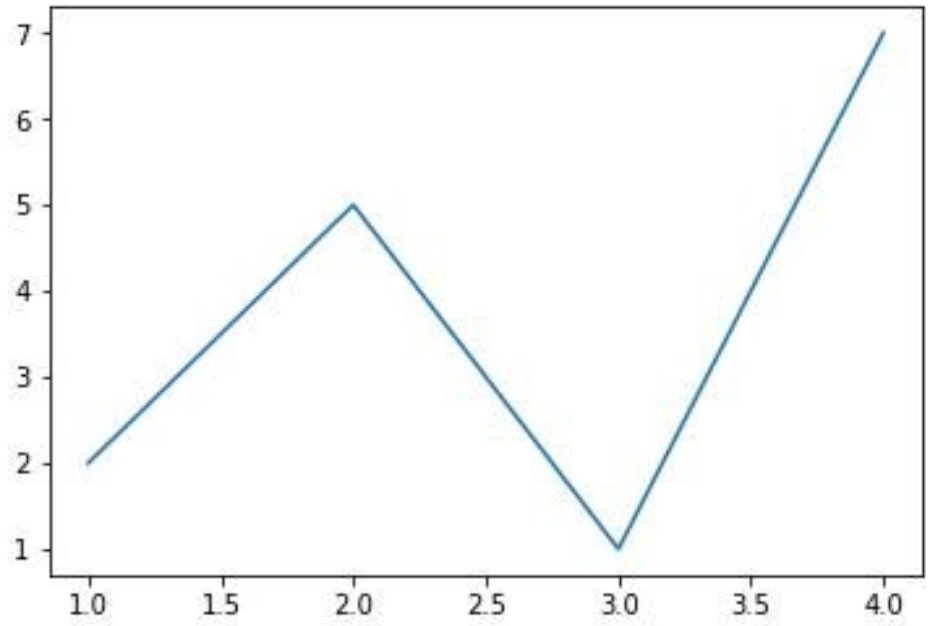
```
fig, ax = plt.subplots(1, 2)
ax[0].plot([2, 6, 7, 3, 8])
ax[0].set_title("Ventas")
ax[0].set_xticks(range(0,5), ["Ene", "Feb", "Mar", "Abr", "May"])
ax[1].hist(np.random.randint(1, 10, 100), bins = 9)
ax[0].set_title("Almacenes")
plt.show()
```



La función plot

La función plot recibe un conjunto de valores x e y, y los muestra en el plano definido por los ejes como puntos unidos por líneas

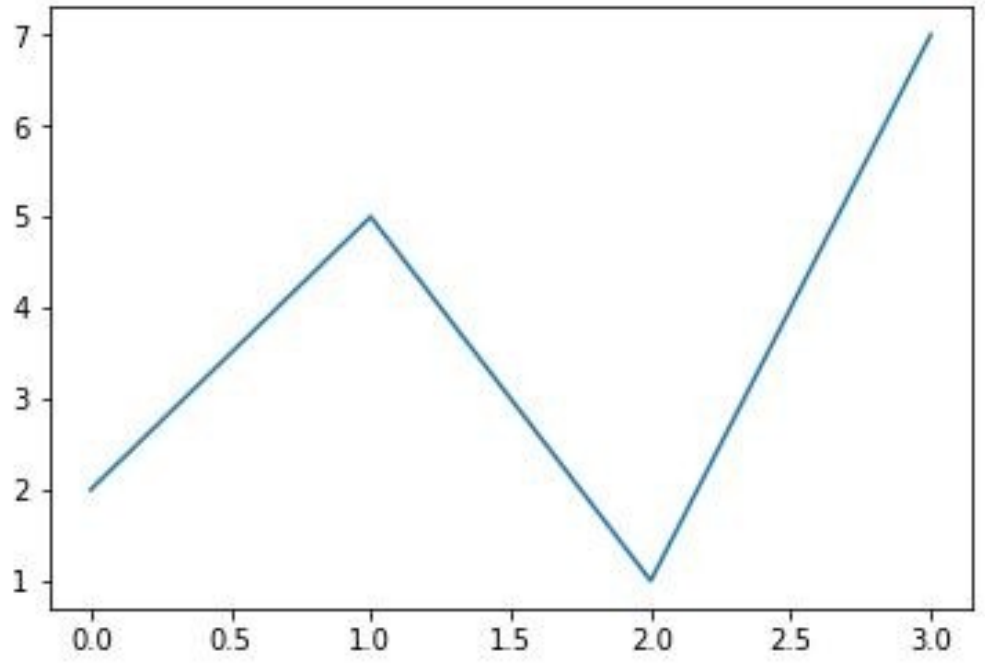
```
x = [1, 2, 3, 4]
y = [2, 5, 1, 7]
plt.plot(x, y)
plt.show()
```



La función plot

Si no se indica el argumento x, se asigna un conjunto de valores por defecto formado por números enteros desde 0 hasta n-1, siendo n el número de puntos a mostrar (es decir, la longitud de y)

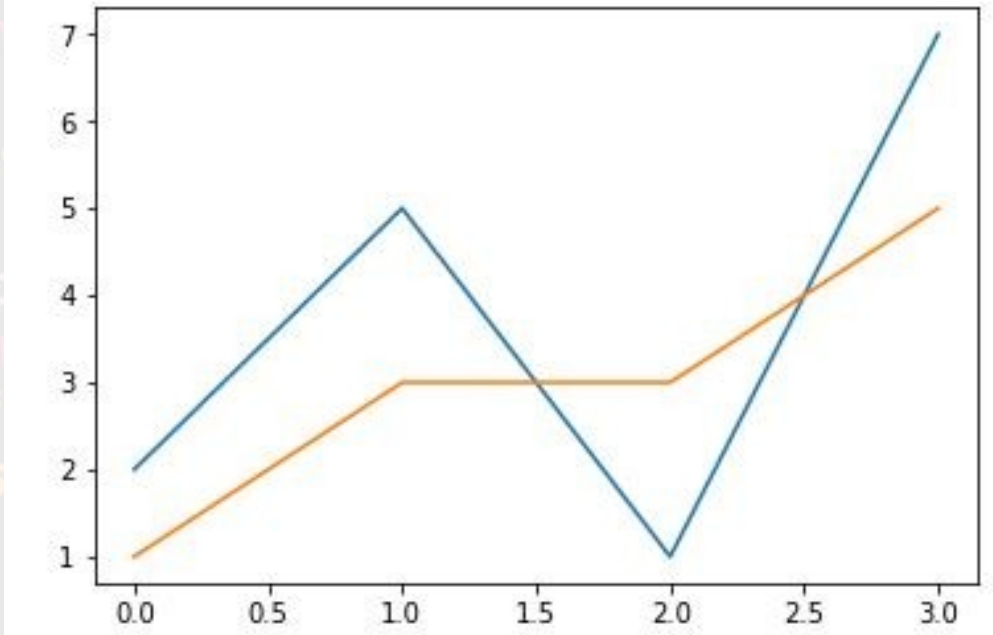
```
y = [2, 5, 1, 7]
plt.plot(y)
plt.show()
```



La función plot

Si ejecutamos dos o más veces la función plot antes de ejecutar la función show, todas las gráficas se mostrarán en el mismo conjunto de ejes:

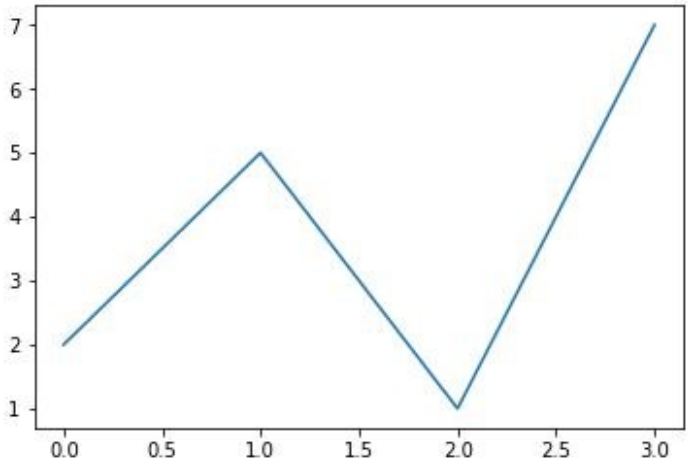
```
plt.plot([2, 5, 1, 7])
plt.plot([1, 3, 3, 5])
plt.show()
```

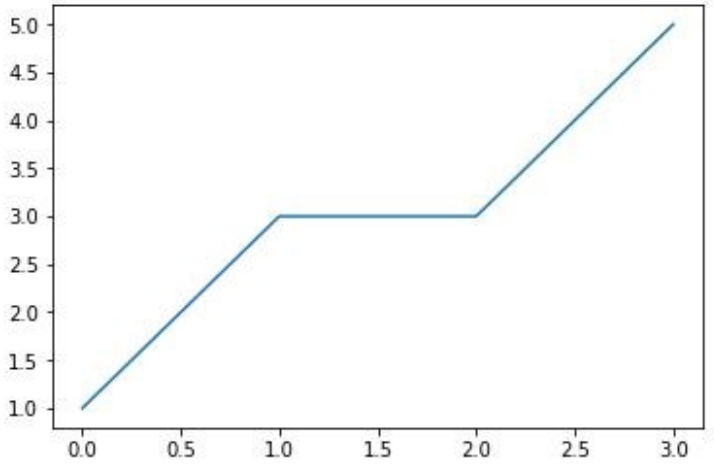


La función plot

El conjunto de ejes y la figura dentro de la cual se muestran son creados automáticamente con la primera ejecución de plot, y se mantendrá dicha figura activa hasta que se muestren las gráficas con la función show. Si, posteriormente, volvemos a ejecutar la función plot, se creará una nueva figura y un nuevo conjunto de ejes.

```
plt.plot([2, 5, 1, 7])
plt.show()
plt.plot([1, 3, 3, 5])
plt.show()
```

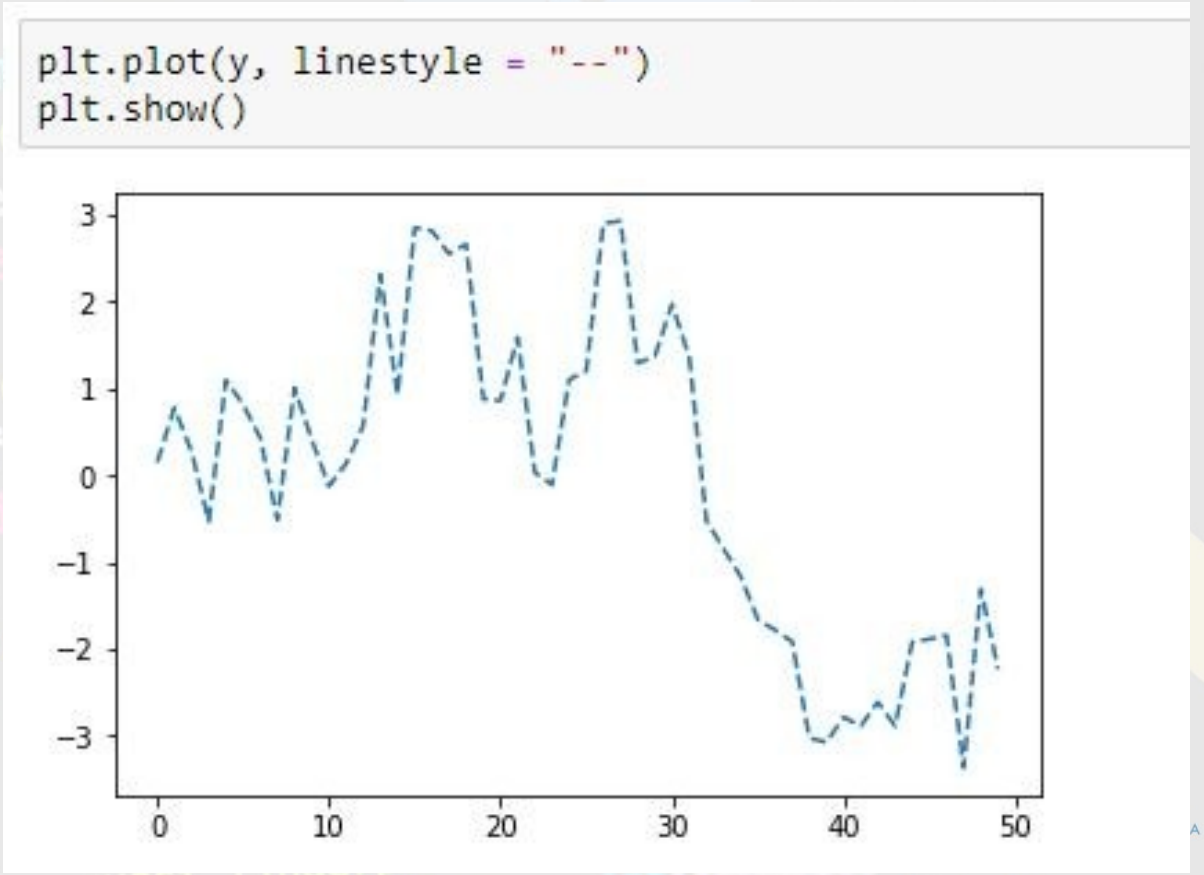




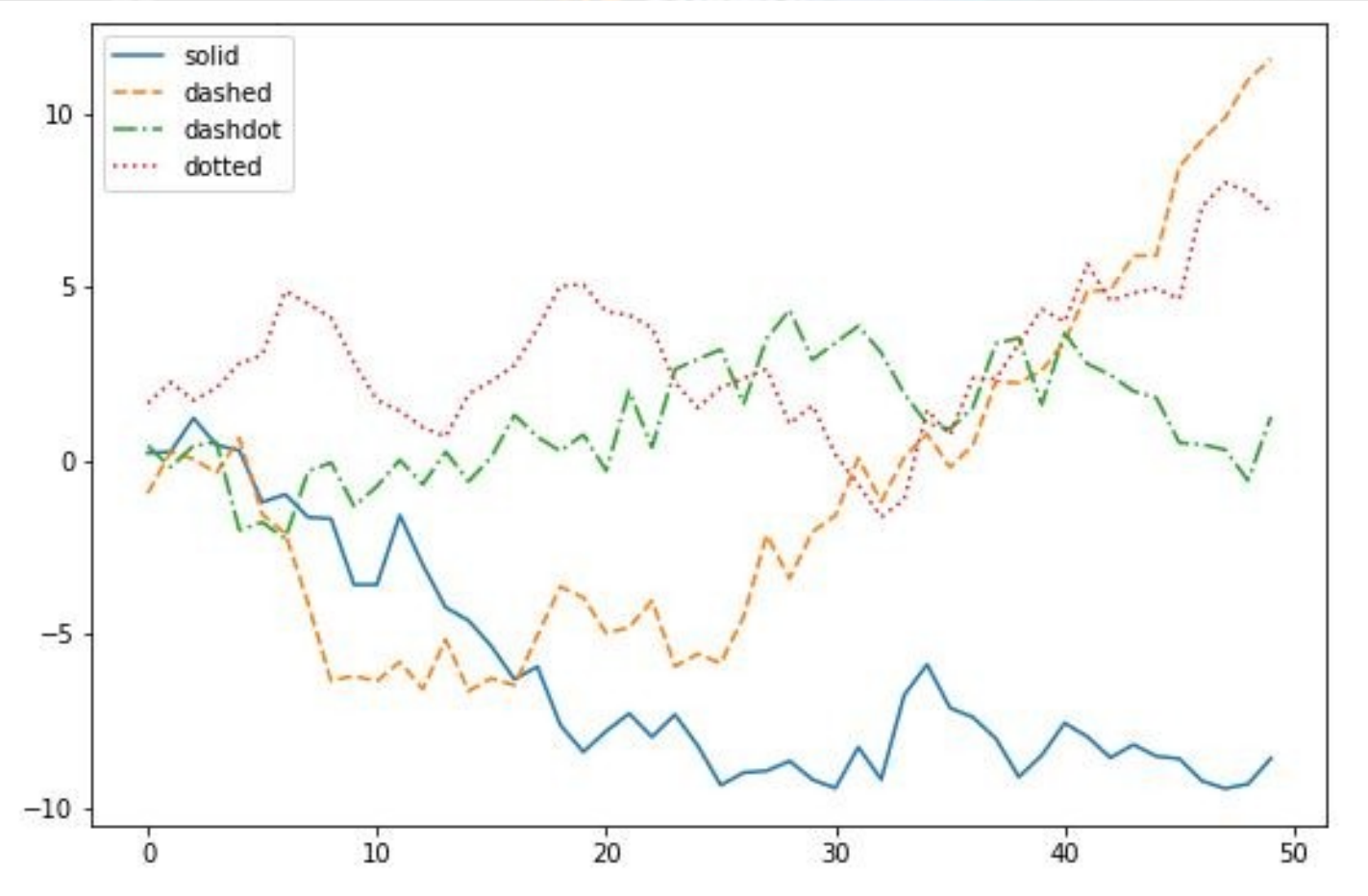
Estilo y ancho de línea

linestyle: permite especificar el estilo de línea.

```
plt.plot(y, linestyle = "--")
plt.show()
```



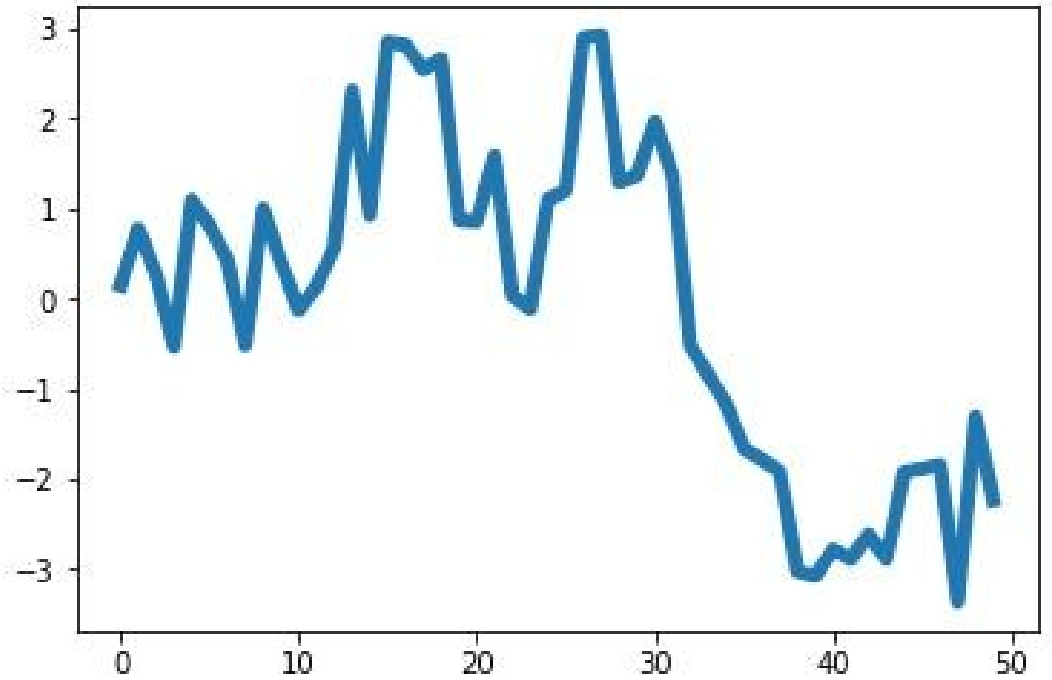
Estilo y ancho de línea



Estilo y ancho de línea

Linewidth: nos permite controlar el ancho de las líneas generadas

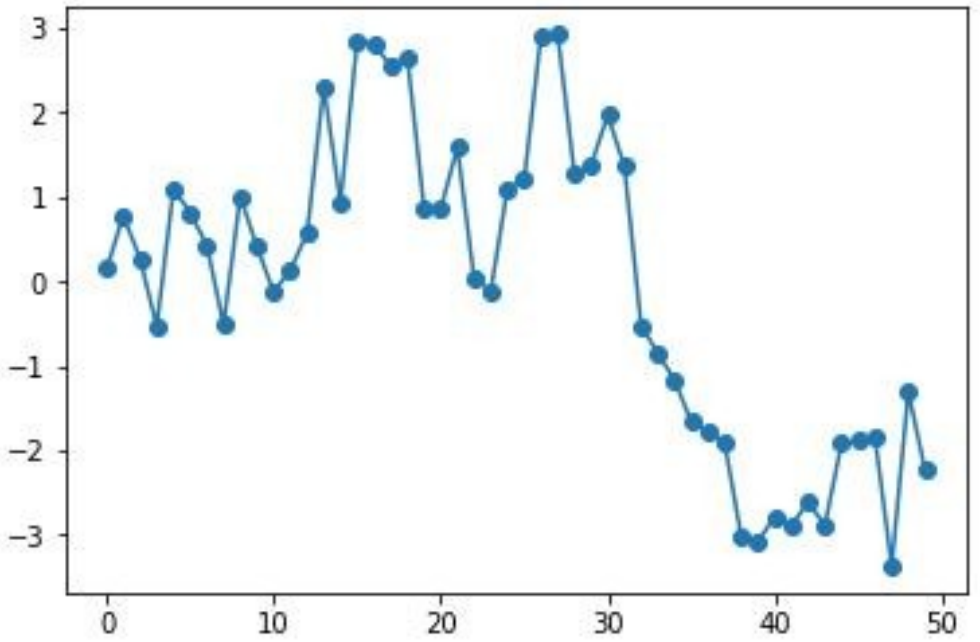
```
plt.plot(y, linewidth = 5)
plt.show()
```



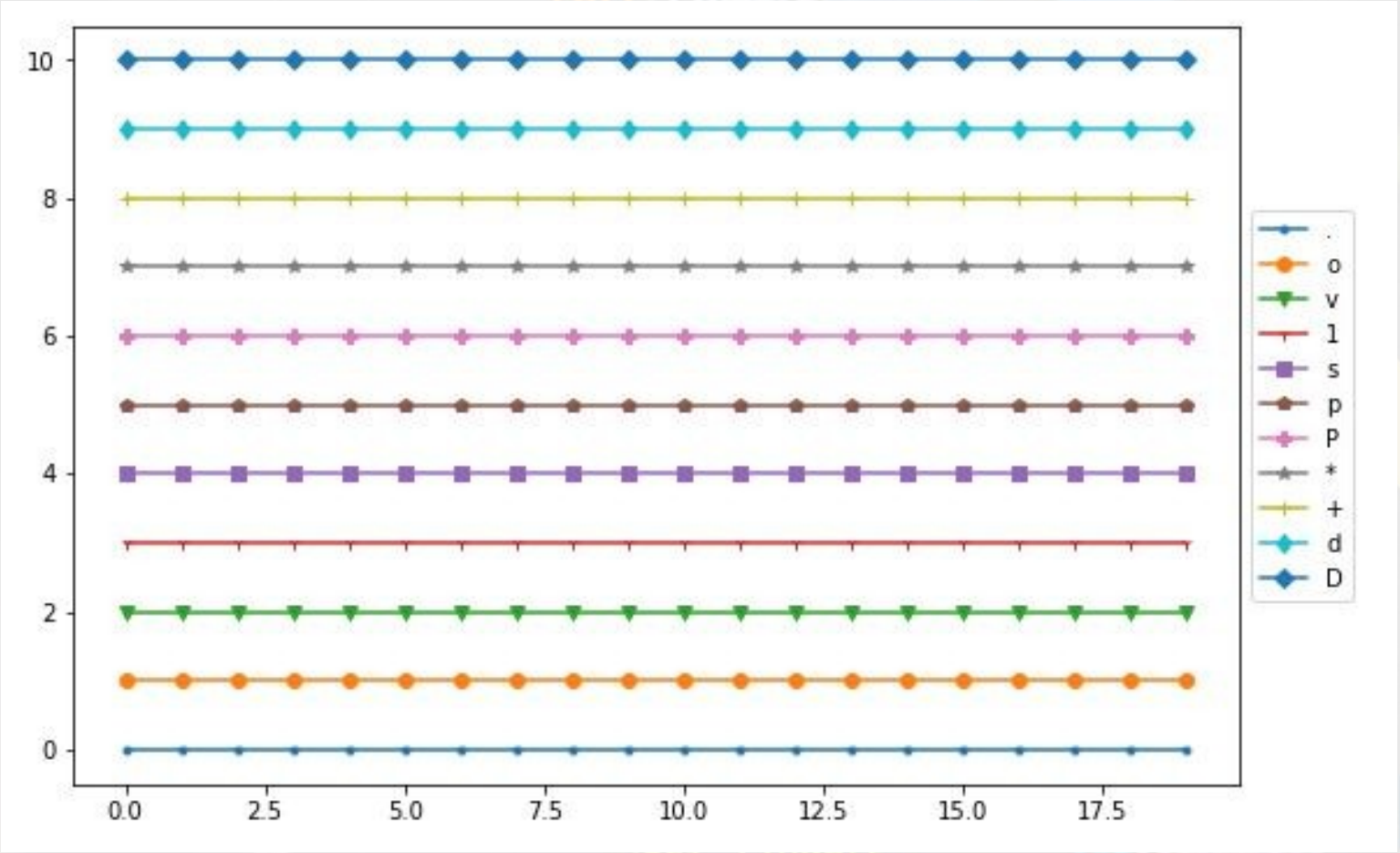
Marcadores y colores

Es posible mostrar encima de cada uno de esos puntos un símbolo (un marcador) con el parámetro **marker**

```
plt.plot(y, marker = "o")
plt.show()
```

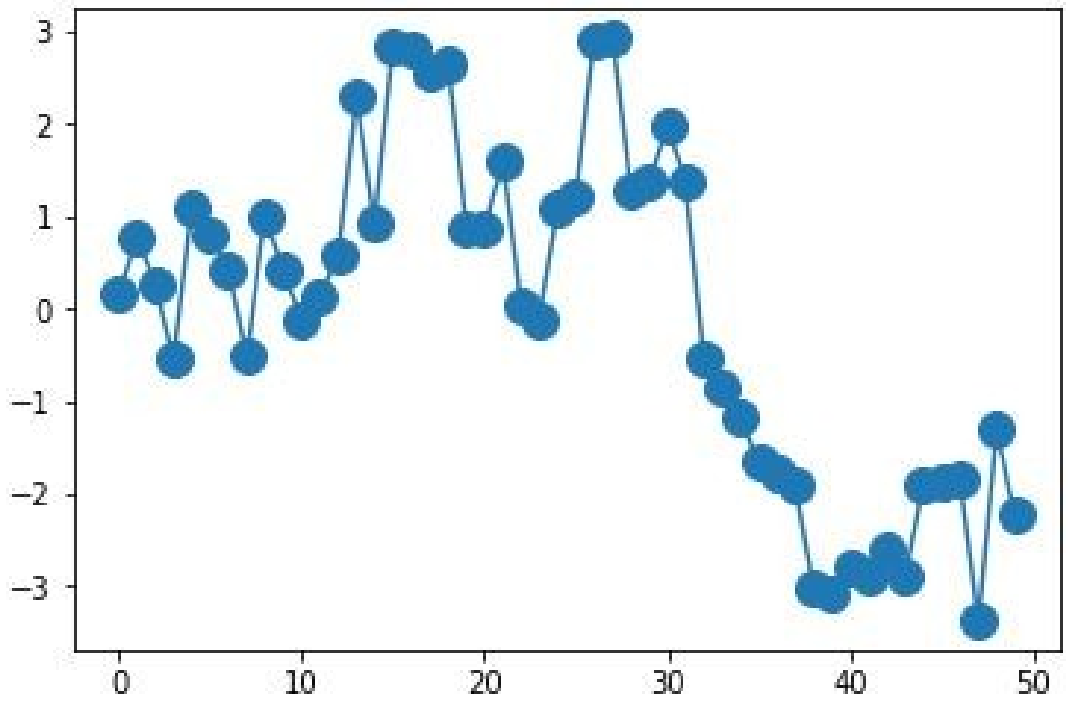


Marcadores y colores



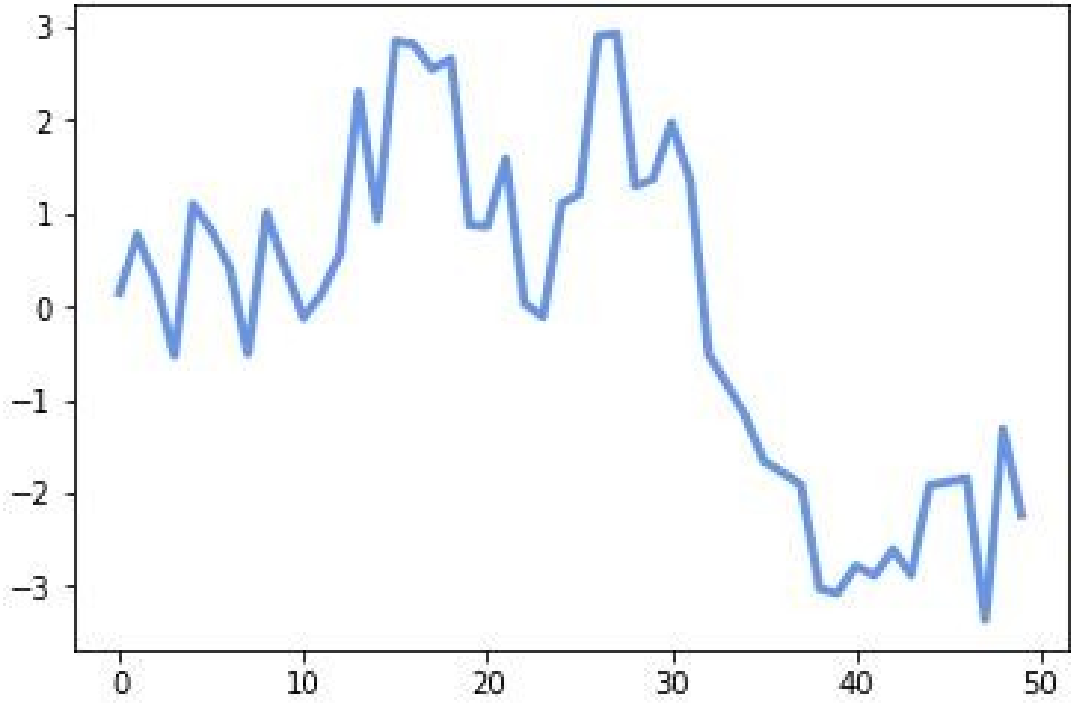
Marcadores y colores - markersize

```
plt.plot(y, marker = "o", markersize = 12)
plt.show()
```



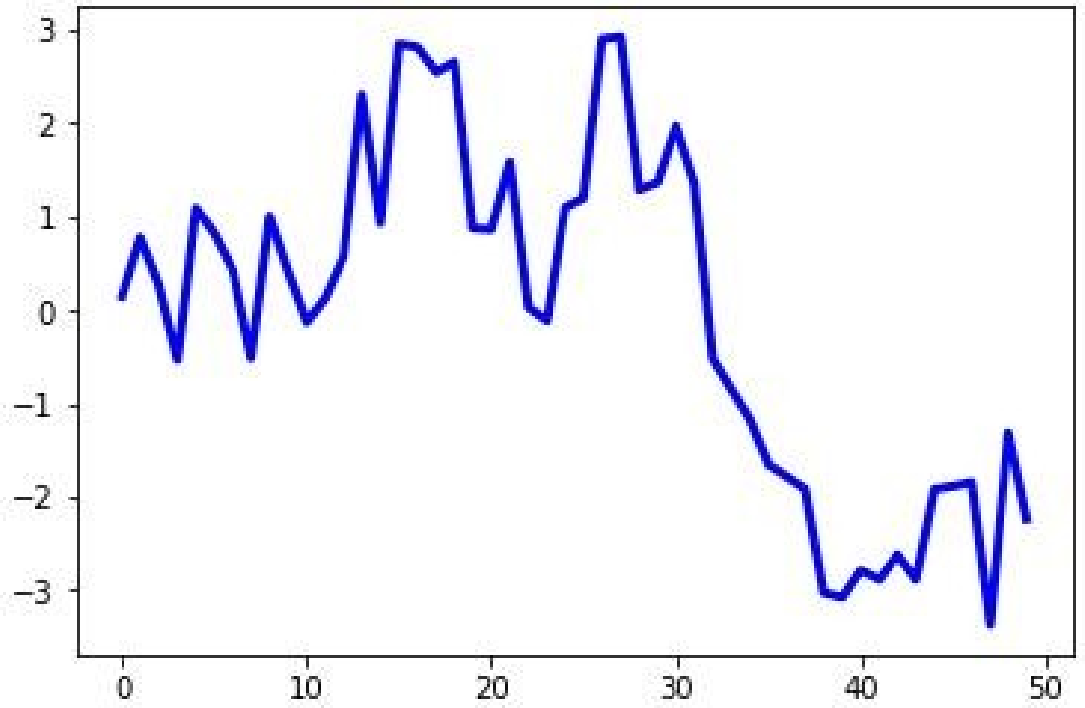
Marcadores y colores - color

```
plt.plot(y, color = "CornflowerBlue", linewidth = 3)
plt.show()
```



Marcadores y colores - color

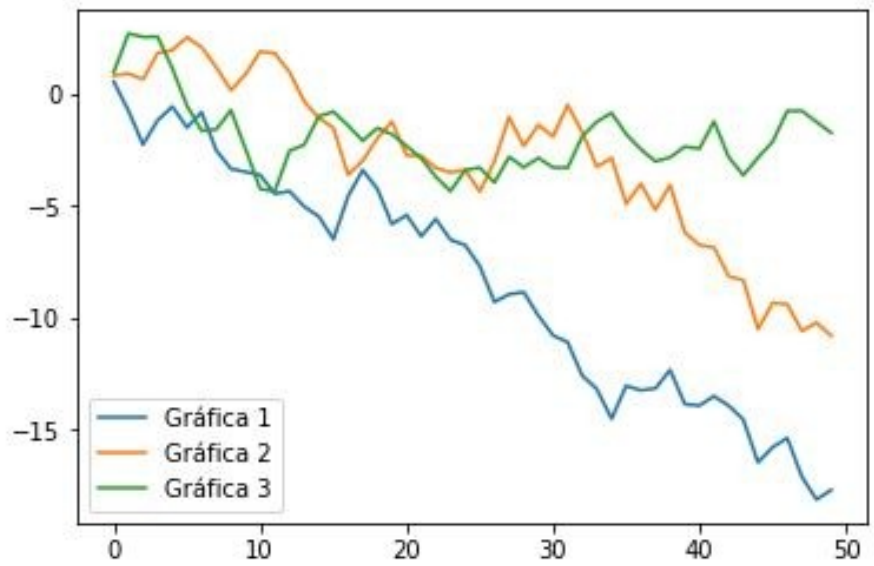
```
plt.plot(y, color = "b", linewidth = 3)
plt.show()
```



Etiquetas

Parámetro label asigna a una gráfica una etiqueta que será mostrada en la leyenda (si activamos ésta)

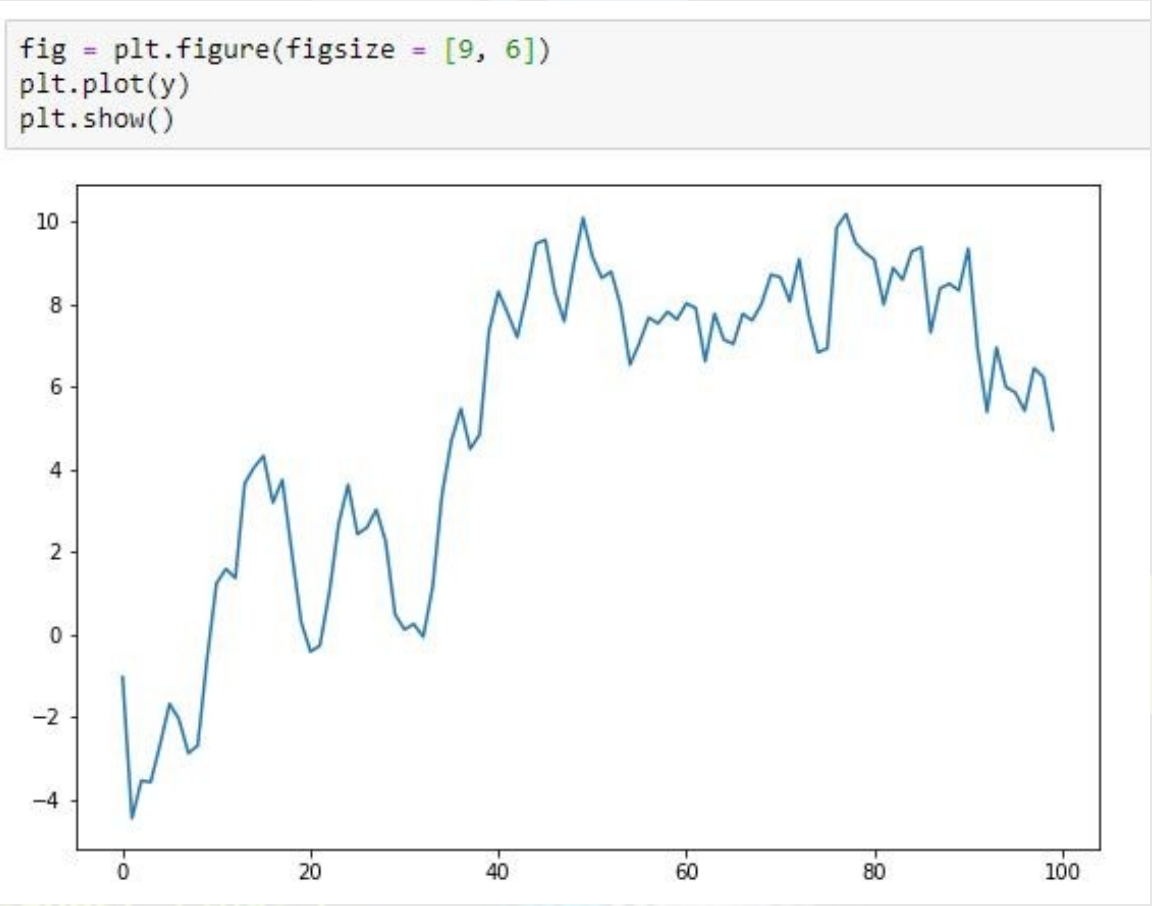
```
plt.plot(np.random.randn(50).cumsum(), label = "Gráfica 1")
plt.plot(np.random.randn(50).cumsum(), label = "Gráfica 2")
plt.plot(np.random.randn(50).cumsum(), label = "Gráfica 3")
plt.legend()
plt.show()
```



Tamaño y color de fondo

figsize: que nos permite especificar el tamaño de la figura en pulgadas, indicando en primer lugar el ancho y, a continuación, el alto.

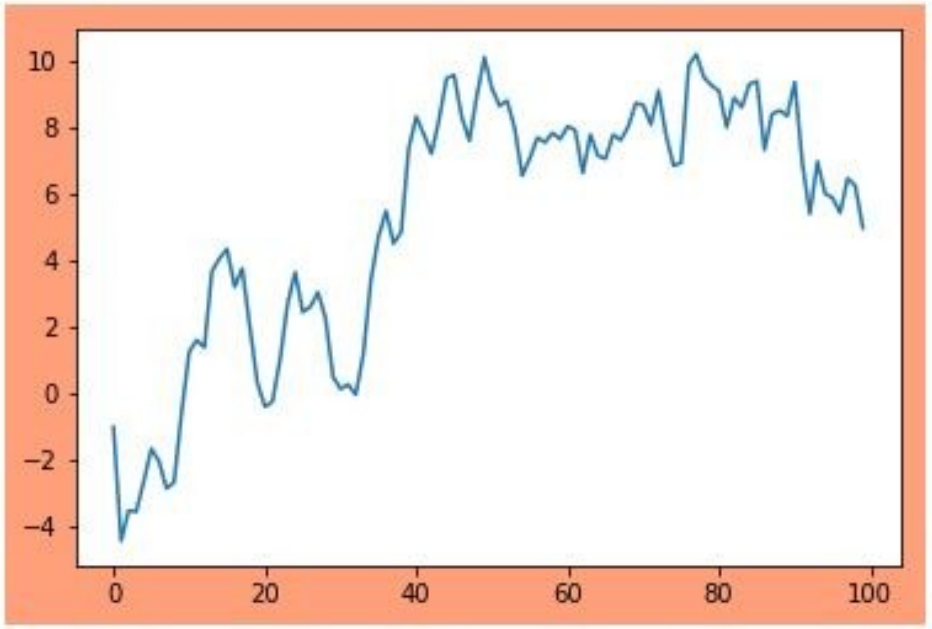
```
fig = plt.figure(figsize = [9, 6])
plt.plot(y)
plt.show()
```



Tamaño y color de fondo

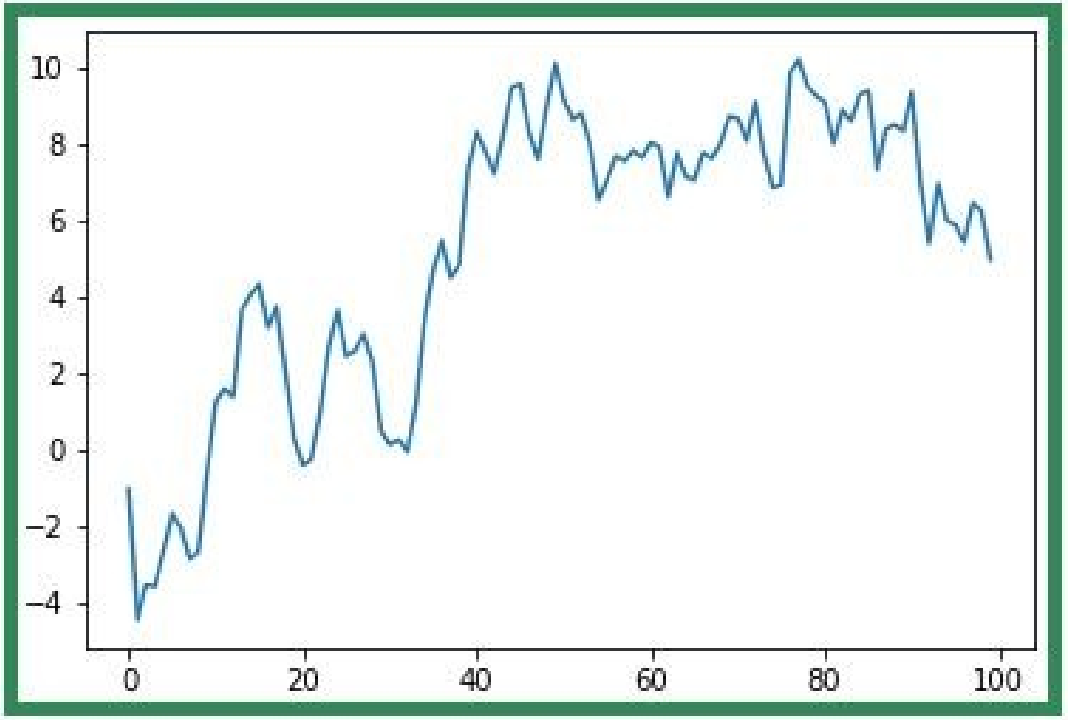
facecolor controla el color del fondo de la figura.

```
fig = plt.figure(facecolor = "LightSalmon")
plt.plot(y)
plt.show()
```



Borde

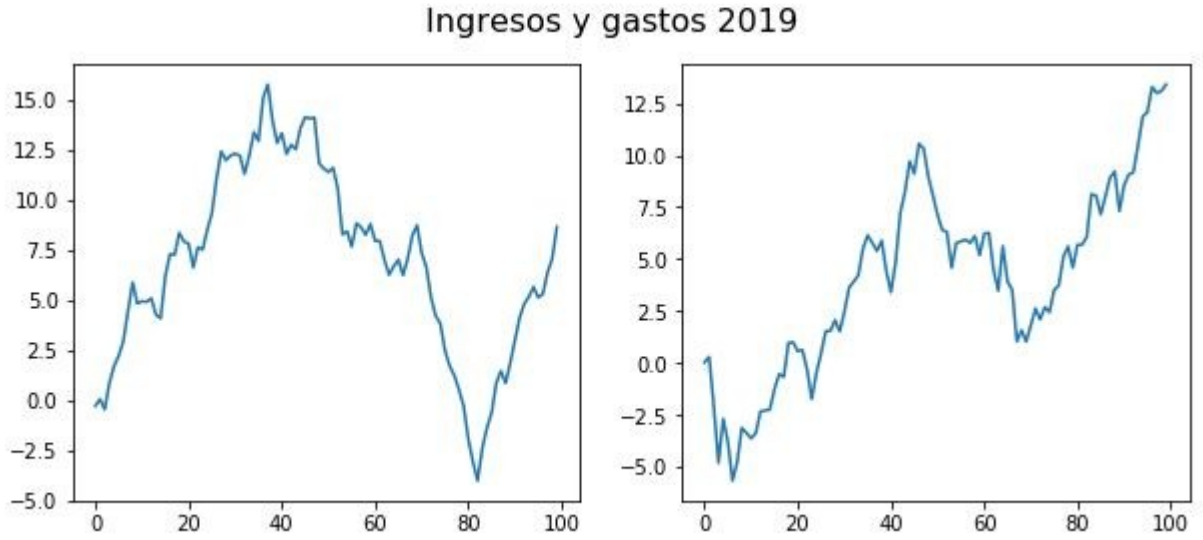
```
fig = plt.figure(edgecolor = "#2E8B57", linewidth = 10)
plt.plot(y)
plt.show()
```



Título

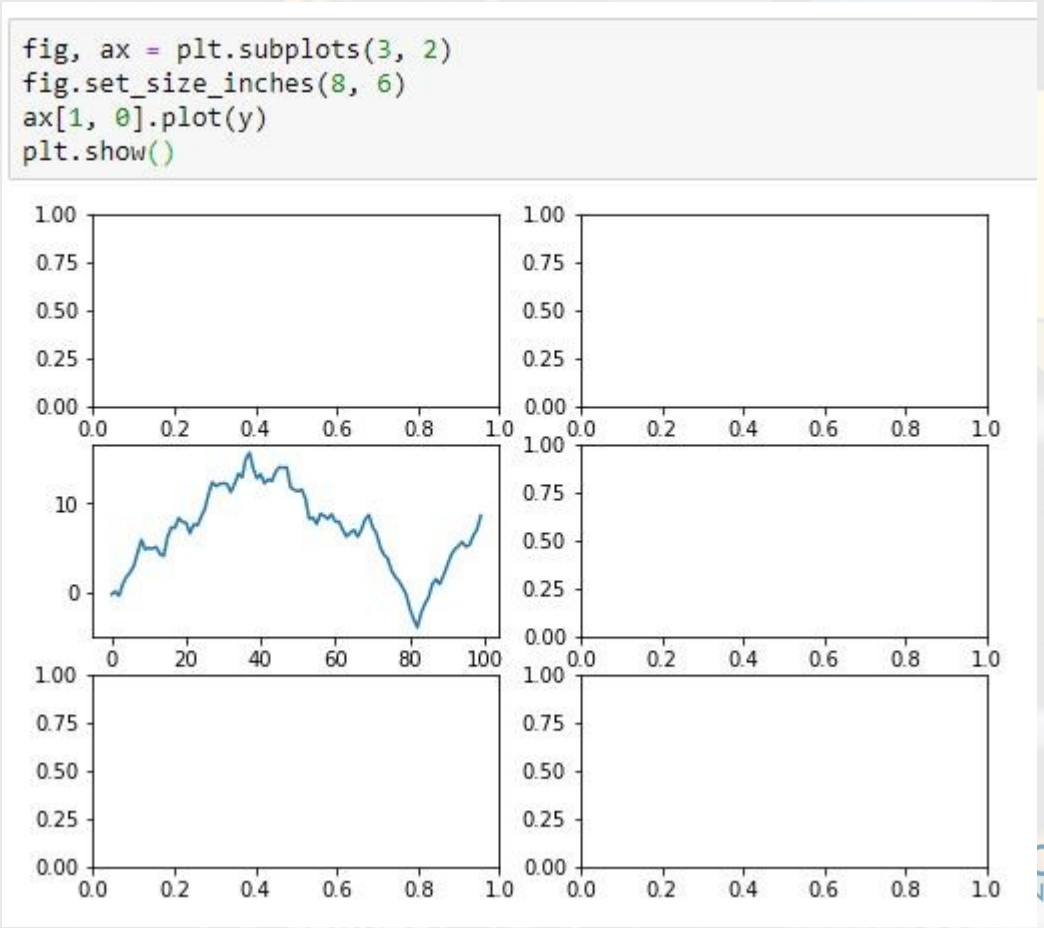
```
y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()
```

```
fig, ax = plt.subplots(1, 2)
fig.set_size_inches(10, 4)
plt.suptitle('Ingresos y gastos 2019', fontsize=16)
ax[0].plot(y1)
ax[1].plot(y2)
plt.show()
```



La función subplots

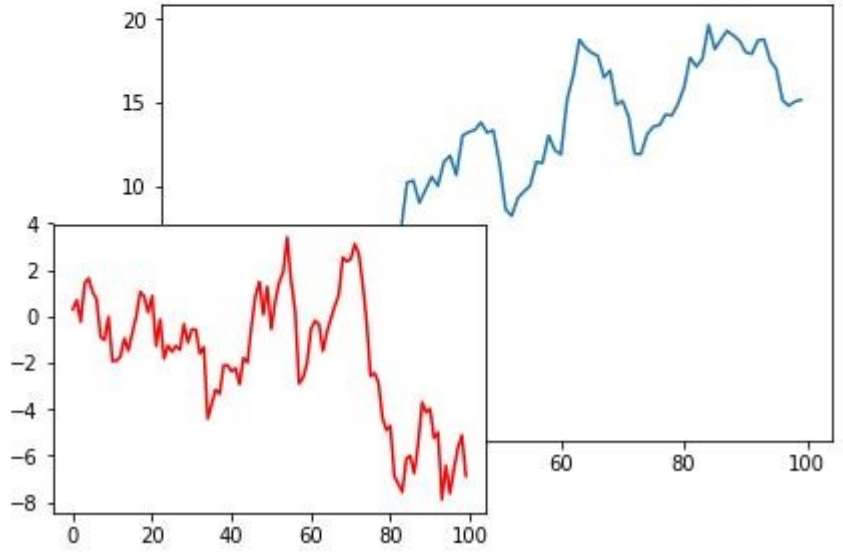
```
fig, ax = plt.subplots(3, 2)
fig.set_size_inches(8, 6)
ax[1, 0].plot(y)
plt.show()
```



La función axes

```
y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()
```

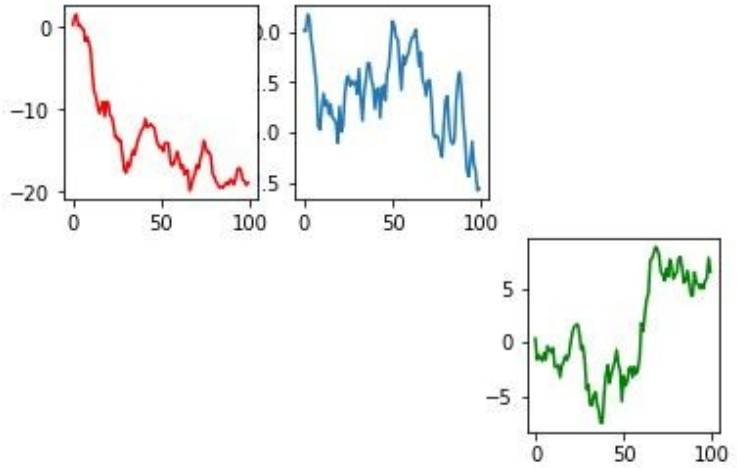
```
fig = plt.figure()

ax1 = plt.axes()
ax1.plot(y1)

ax2 = plt.axes([0.0, 0.0, 0.5, 0.5])
ax2.plot(y2, color = "red")

plt.show()
```



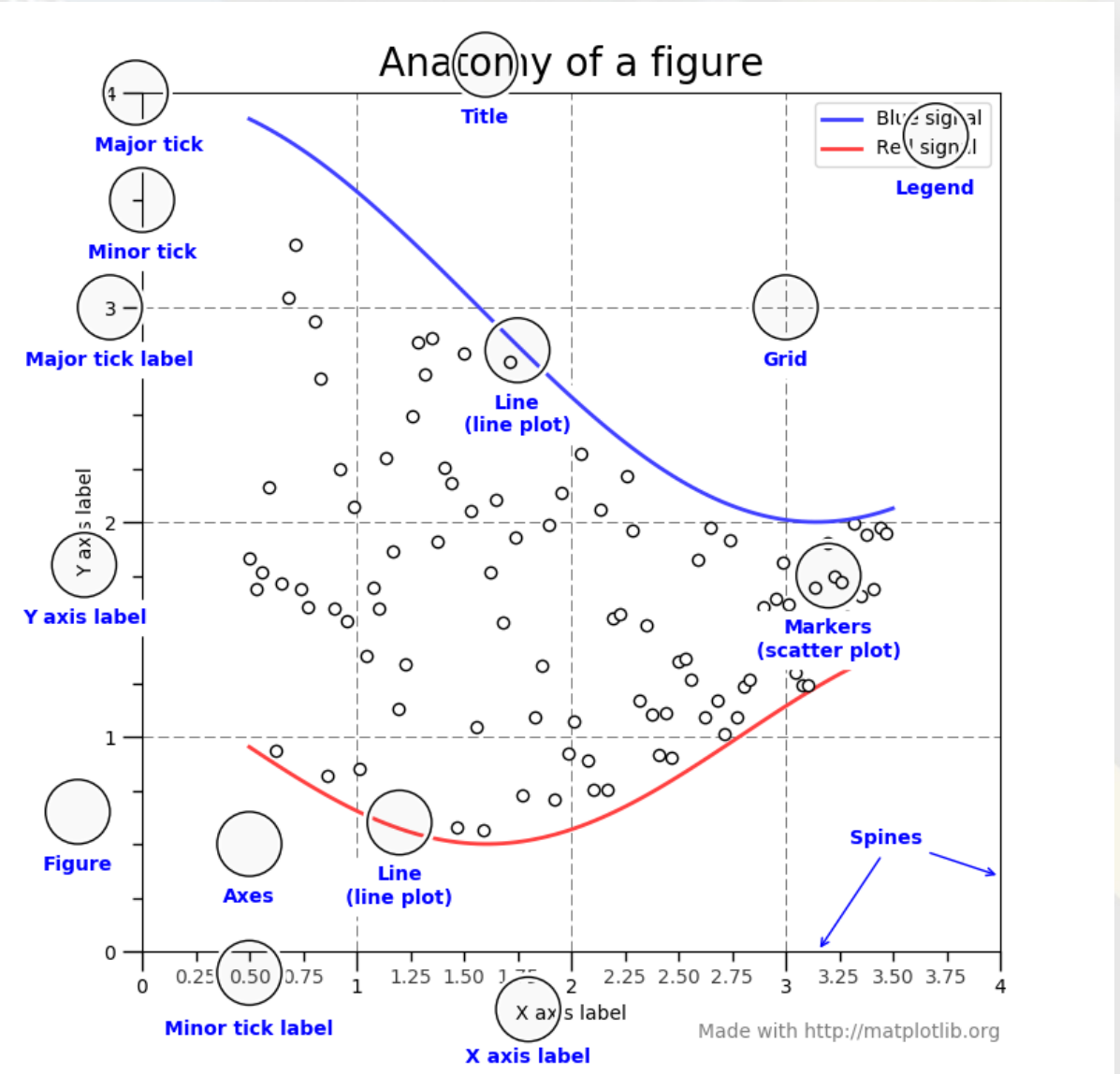
La función add_subplot

```
y1 = np.random.randn(100).cumsum()
y2 = np.random.randn(100).cumsum()
y3 = np.random.randn(100).cumsum()
```

```
fig = plt.figure()
fig.add_subplot(2, 3, 2)
plt.plot(y1)
fig.add_subplot(2, 3, 1)
plt.plot(y2, color = "red")
fig.add_subplot(2, 3, 6)
plt.plot(y3, color = "green")
plt.show()
```



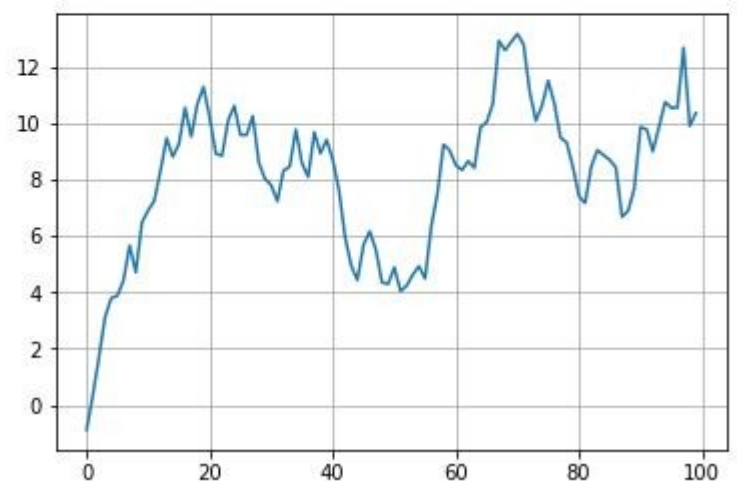
https://matplotlib.org/tutorials/introductory/usage.html#parts-of-a-figure



Grid

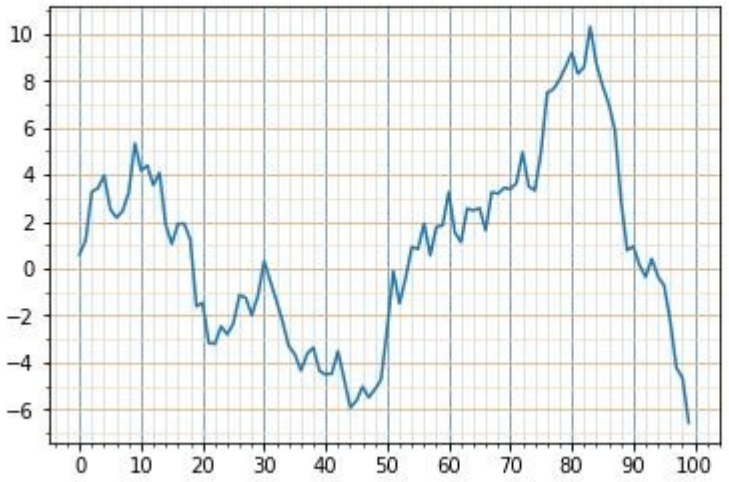
```
y = np.random.randn(100).cumsum()
```

```
fig, ax = plt.subplots()
ax.plot(y)
ax.grid()
plt.show()
```



Grid

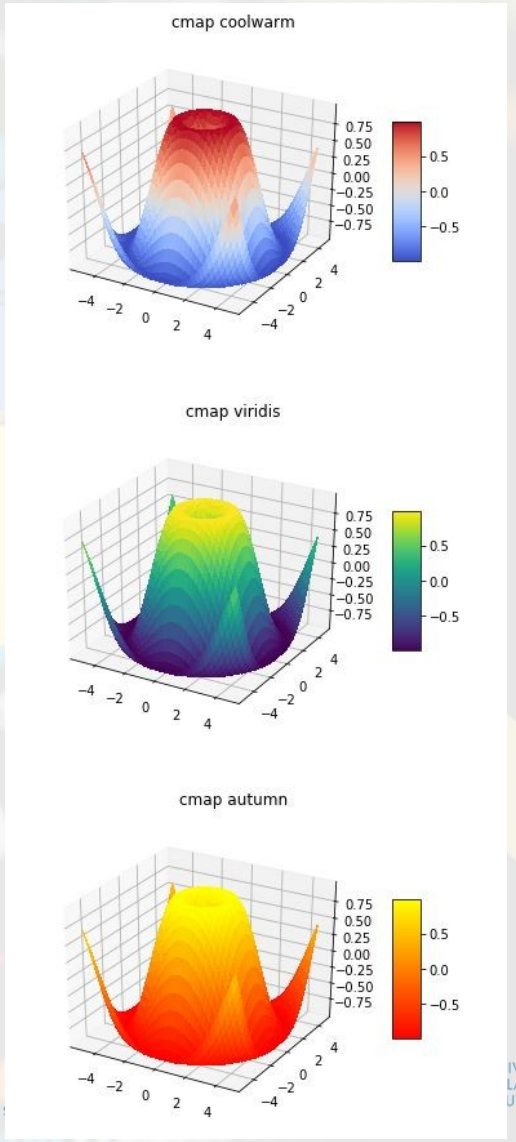
```
fig, ax = plt.subplots()
ax.plot(y)

# Eje x
ax.xaxis.set_major_locator(MultipleLocator(10))
ax.xaxis.set_minor_locator(MultipleLocator(2))
# Eje y
ax.yaxis.set_major_locator(MultipleLocator(2))
ax.yaxis.set_minor_locator(MultipleLocator(1))

ax.grid(which = "major", axis = "x", color = "SteelBlue")
ax.grid(which = "minor", axis = "x", color = "LightSteelBlue", alpha = 0.5)

ax.grid(which = "major", axis = "y", color = "Chocolate")
ax.grid(which = "minor", axis = "y", color = "Wheat", alpha = 0.8)

plt.show()
```



Mapas de color

- **matplotlib** ofrece un conjunto de mapas de color predefinidos.
- Éstos son diccionarios de colores que "mapean" los valores representados en nuestras gráficas a otros conjunto de colores.
- La idea detrás de los mapas de colores es encontrar una buena representación en espacios tridimensionales, colores capaces de transmitir la sensación de profundidad que los colores lisos no transmiten correctamente.
- La elección correcta de un mapa de color puede cambiar radicalmente la forma en la que percibimos una gráfica

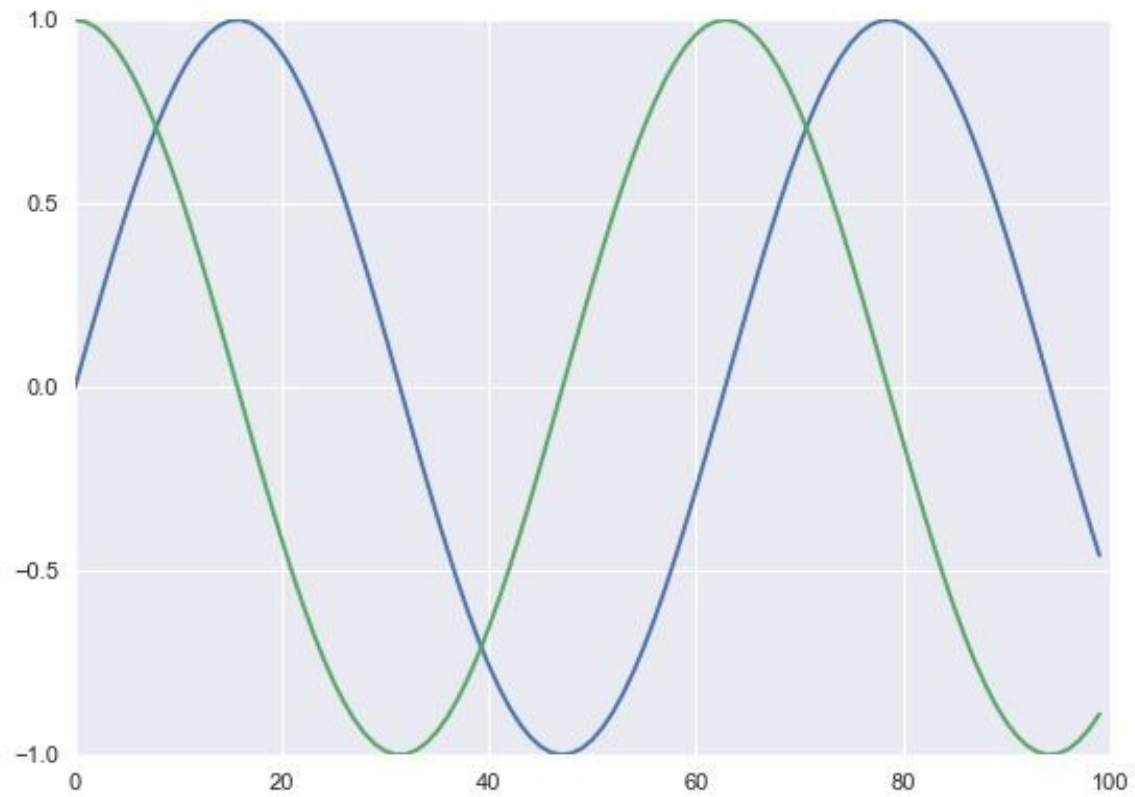
https://matplotlib.org/tutorials/colors/colormaps.html



Gráficos en 2D Lineas

```
sin = np.sin(np.arange(0, 10, 0.1))
cos = np.cos(np.arange(0, 10, 0.1))
```

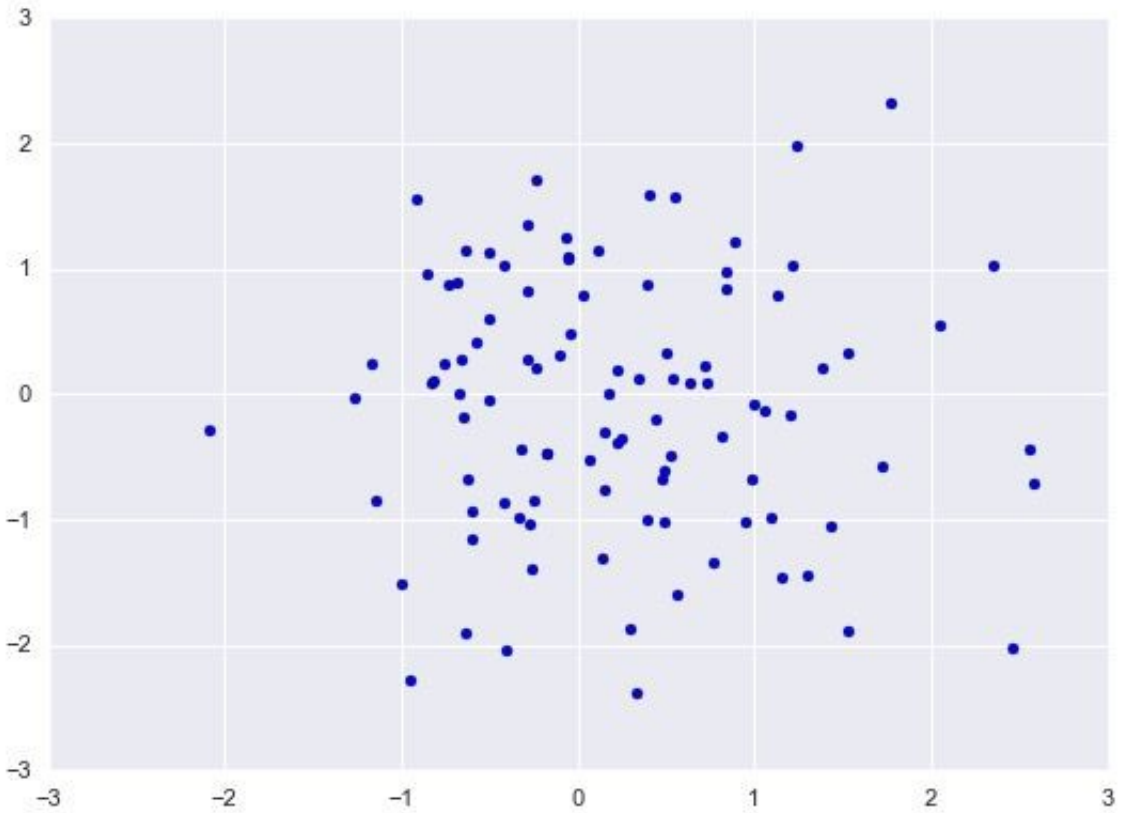
```
fig, ax = plt.subplots()
ax.plot(sin)
ax.plot(cos)
plt.show()
```



Gráficos en 2D
Dispersión

```
x = np.random.randn(100)
y = np.random.randn(100)
```

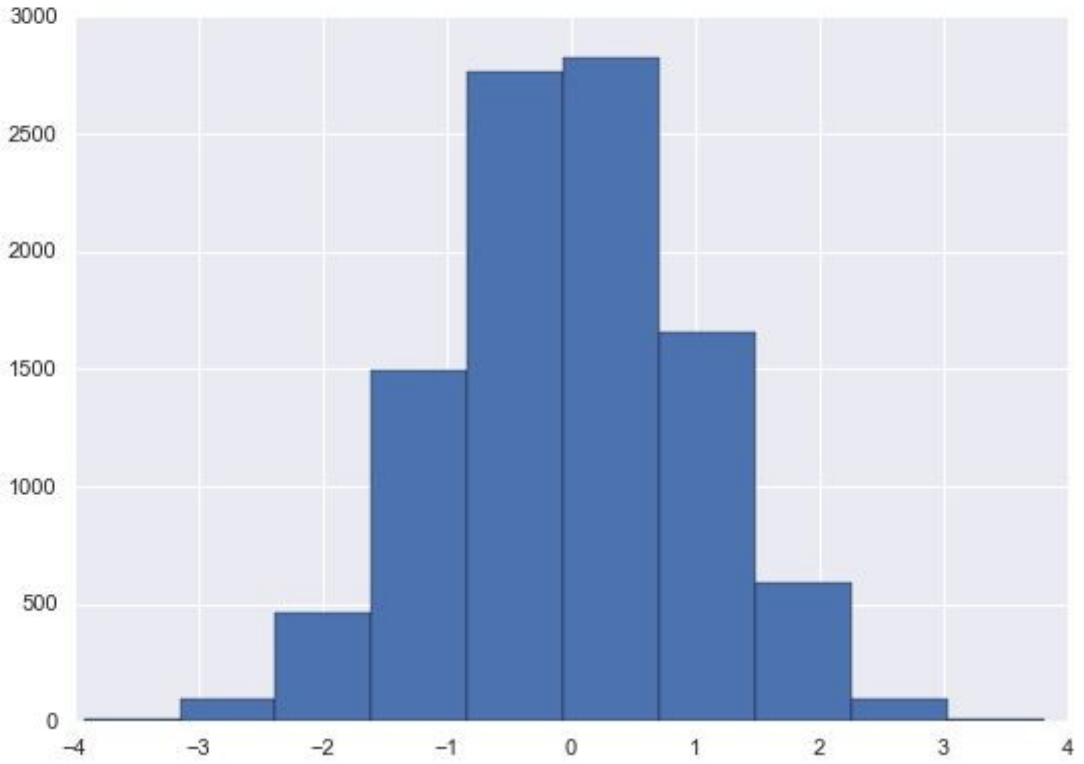
```
fig, ax = plt.subplots()
ax.scatter(x, y)
plt.show()
```



Gráficos en 2D
Historgramas

```
y = np.random.randn(10000)
```

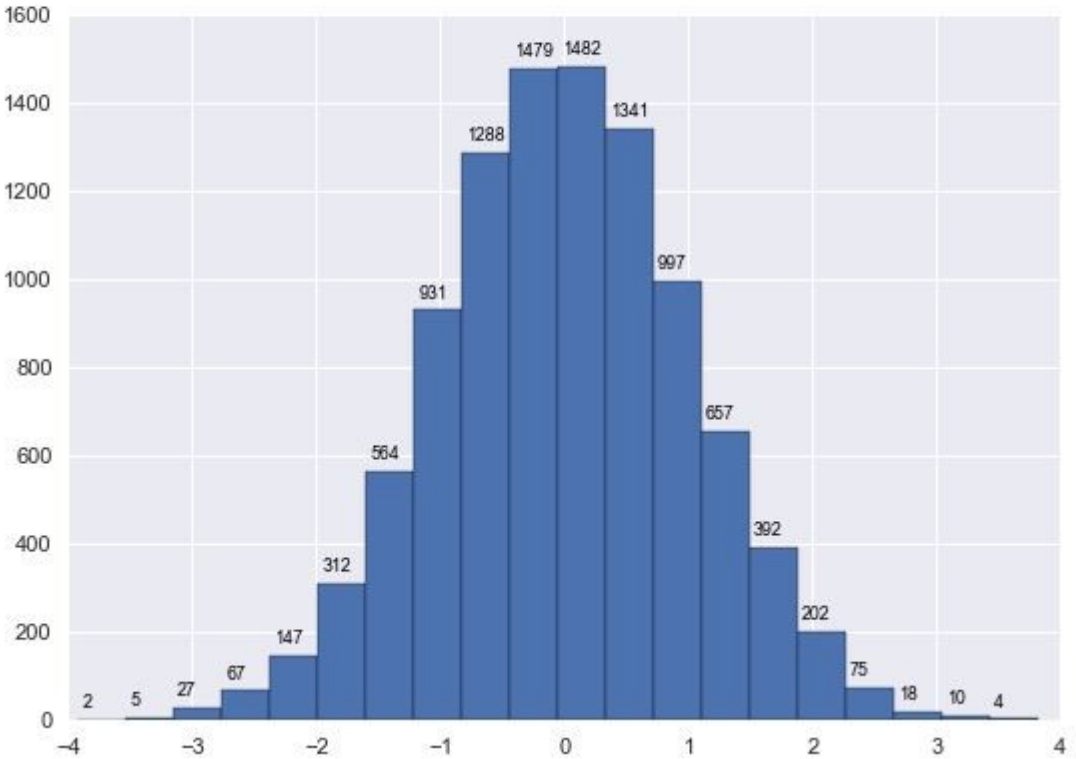
```
fig, ax = plt.subplots()
ax.hist(y)
plt.show()
```



Gráficos en 2D
Historgramas con etiquetas

- Tras crear el histograma y recoger el resultado en las variables v, m y g, recorremos con un for cada uno de los rectángulos referenciados en g.
- Para cada uno de ellos extraemos su posición x y su altura utilizando los métodos get_x y get_height.
- Por último, mostramos el valor del bin (contenido en v[i]) en la posición (posx, posy)

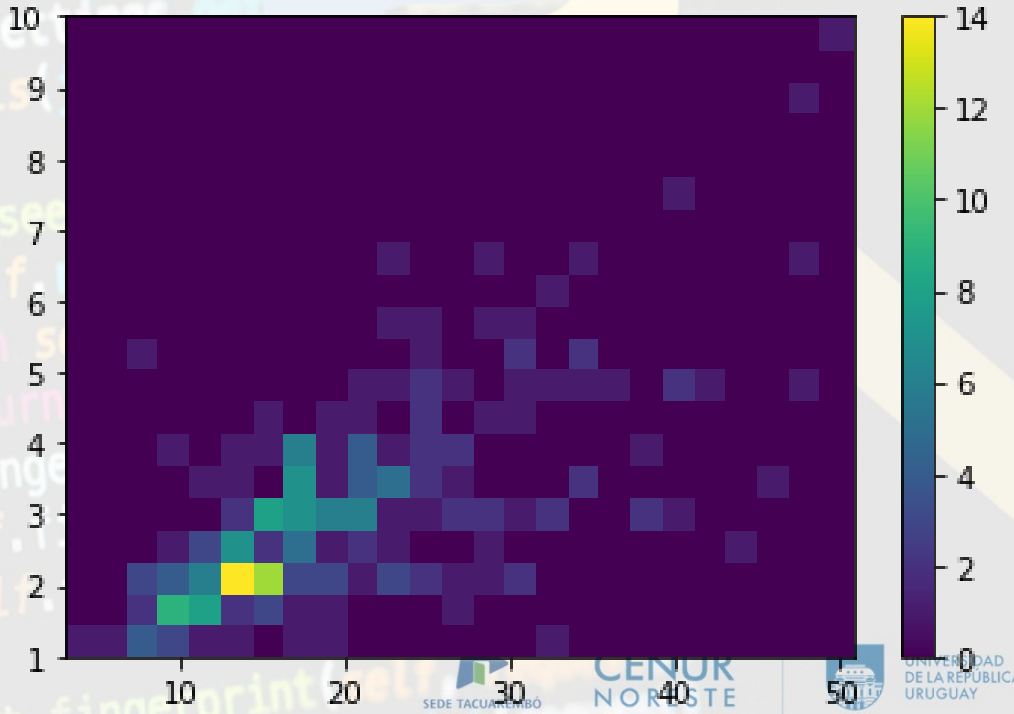
```
fig, ax = plt.subplots()
v, m, g = ax.hist(y, bins = 20)
for i, rect in enumerate(g):
    posx = rect.get_x()
    posy = rect.get_height()
    ax.text(posx + 0.03, posy + 30, int(v[i]), color='black', fontsize = 8
plt.show()
```



Gráficos en 2D
Histogramas de dos variables

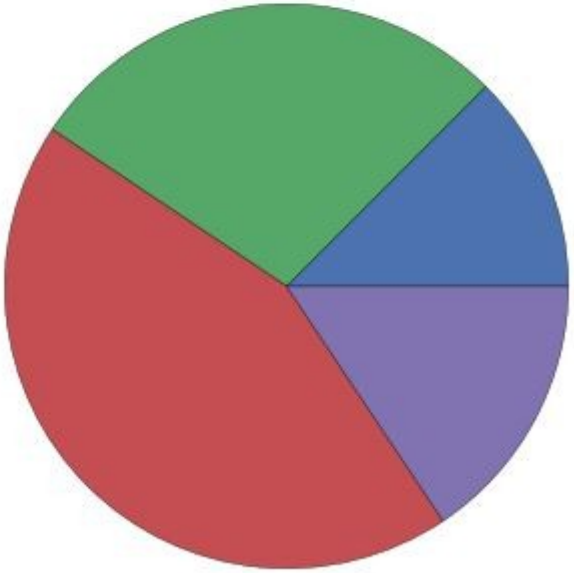
```
plt.hist2d(x, y, bins = (25, 20));
plt.colorbar();
```

- Es posible modificar el número de áreas en las que dividir el plano utilizando el parámetro **bins**.
- Si éste es un número, se dividirá tanto el eje x como el eje y en tantos bloque como indique.

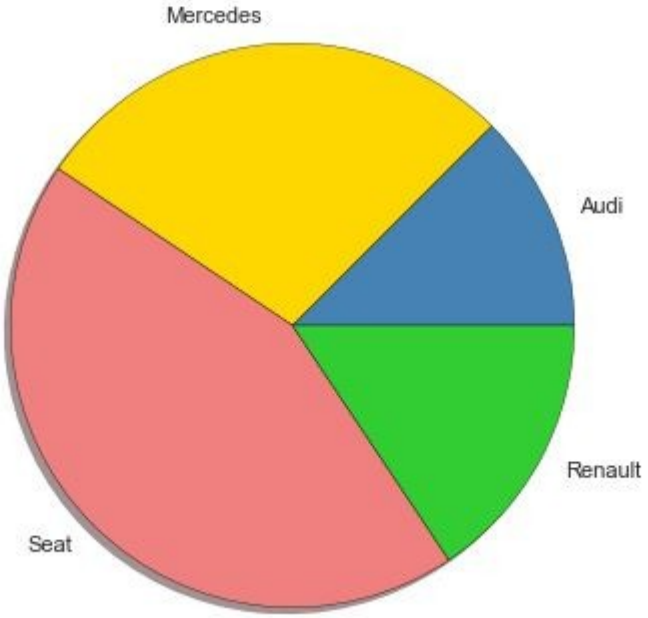


Gráficos en 2D Circulares

```
data = [4, 9, 14, 5]
fig, ax = plt.subplots()
g = ax.pie(data)
plt.show()
```



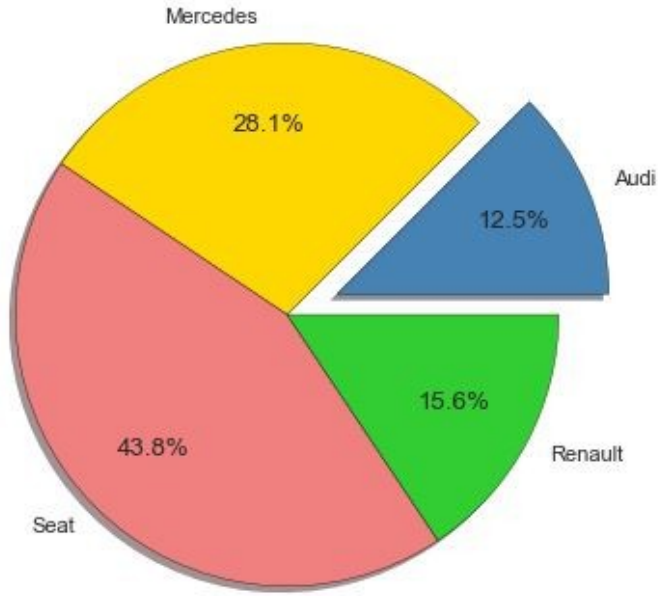
```
data = [4, 9, 14, 5]
cars = ["Audi", "Mercedes", "Seat", "Renault"]
fig, ax = plt.subplots()
g = ax.pie(
    data,
    labels = cars,
    colors = ["SteelBlue", "Gold", "LightCoral", "LimeGreen"],
    shadow = True
)
plt.show()
```



Gráficos en 2D Circulares

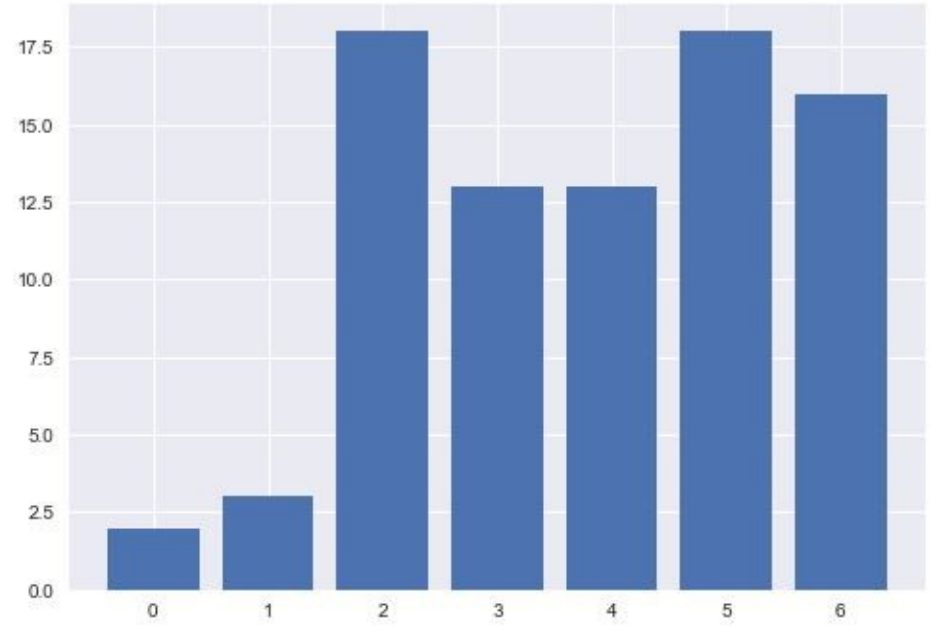
- La distancia de las etiquetas al centro del gráfico se controla mediante el parámetro **labeldistance**
- Es posible separar uno o varios de los sectores con el parámetro **explode**
- Podemos mostrar los porcentajes representados por cada sector utilizando el parámetro **autopct**
- El parámetro **pctdistance** controla la distancia del número mostrado al centro del gráfico

```
data = [4, 9, 14, 5]
cars = ["Audi", "Mercedes", "Seat", "Renault"]
fig, ax = plt.subplots()
g = ax.pie(
    data,
    labels = cars,
    labeldistance = 1.1,
    colors = ["SteelBlue", "Gold", "LightCoral", "LimeGreen"],
    shadow = True,
    explode = (0.2, 0, 0, 0),
    autopct = '%1.1f%%',
    pctdistance = 0.7
)
plt.show()
```

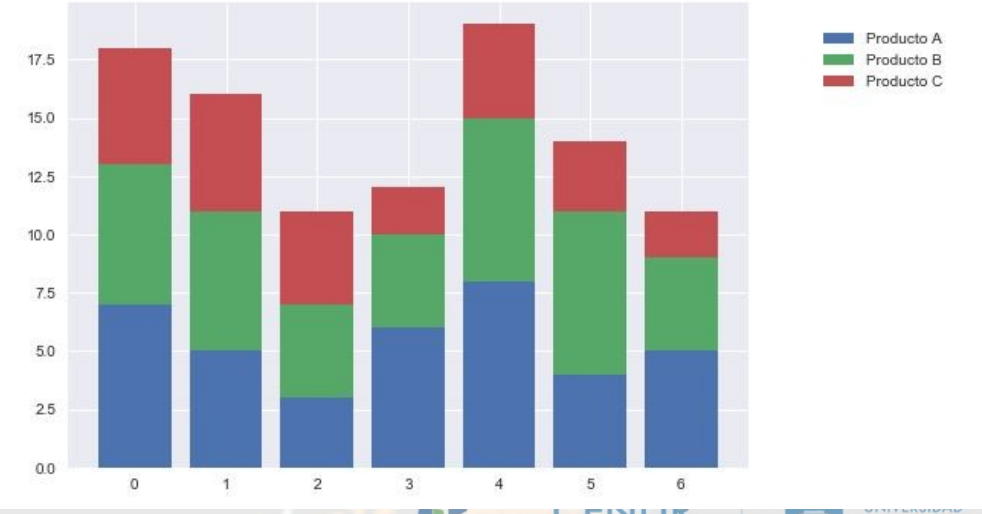


Gráficos en 2D Barras

```
x = np.arange(7)
y = np.random.randint(1, 20, 7)
fig, ax = plt.subplots()
ax.bar(x, y)
plt.show()
```

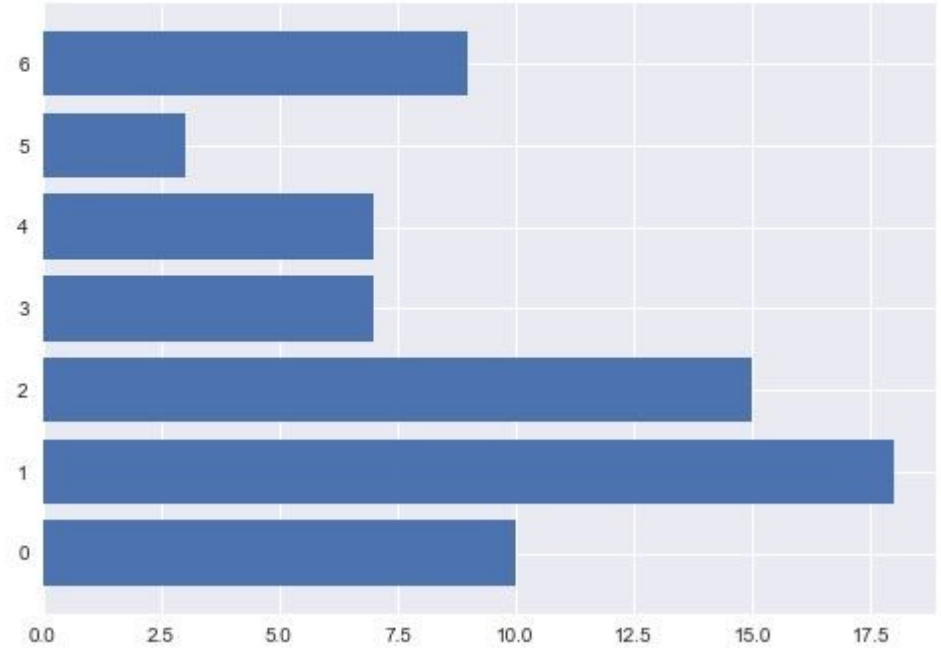



```
x = np.arange(7)
y1 = np.random.randint(1, 10, 7)
y2 = np.random.randint(1, 10, 7)
y3 = np.random.randint(1, 10, 7)
fig, ax = plt.subplots()
ax.bar(x, y1, label = "Producto A")
ax.bar(x, y2, bottom = y1, label = "Producto B")
ax.bar(x, y3, bottom = y1 + y2, label = "Producto C")
ax.legend(loc = (1.1, 0.8))
plt.show()
```

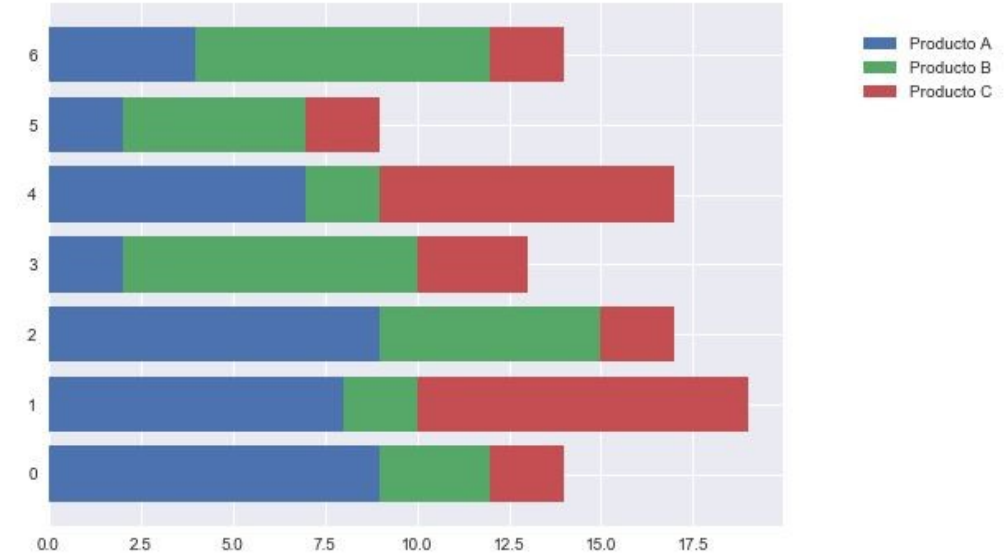


Gráficos en 2D Barras

```
y = np.arange(7)
x = np.random.randint(1, 20, 7)
fig, ax = plt.subplots()
ax.barh(y, x)
plt.show()
```




```
y = np.arange(7)
x1 = np.random.randint(1, 10, 7)
x2 = np.random.randint(1, 10, 7)
x3 = np.random.randint(1, 10, 7)
fig, ax = plt.subplots()
ax.barh(y, x1, label = "Producto A")
ax.barh(y, x2, left = x1, label = "Producto B")
ax.barh(y, x3, left = x1 + x2, label = "Producto C")
ax.legend(loc = (1.1, 0.8))
plt.show()
```

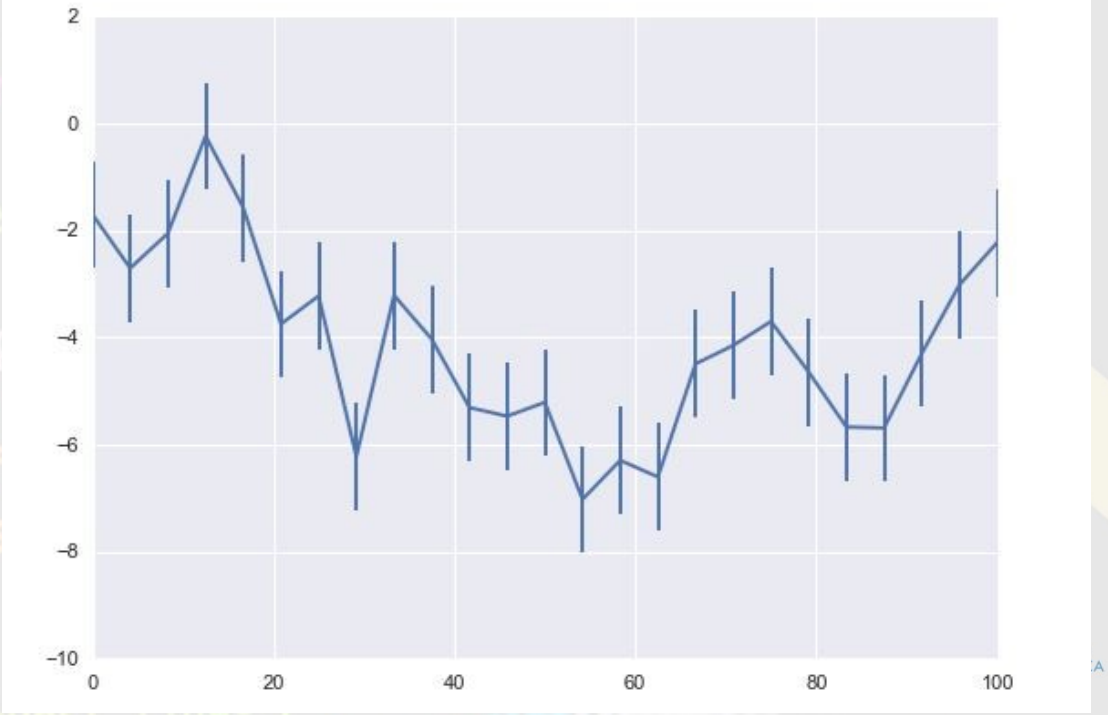


Gráficos en 2D - Barras de Error

yerr margen de error

```
x = np.linspace(0, 100, 25)
y = np.random.randn(25).cumsum()
```

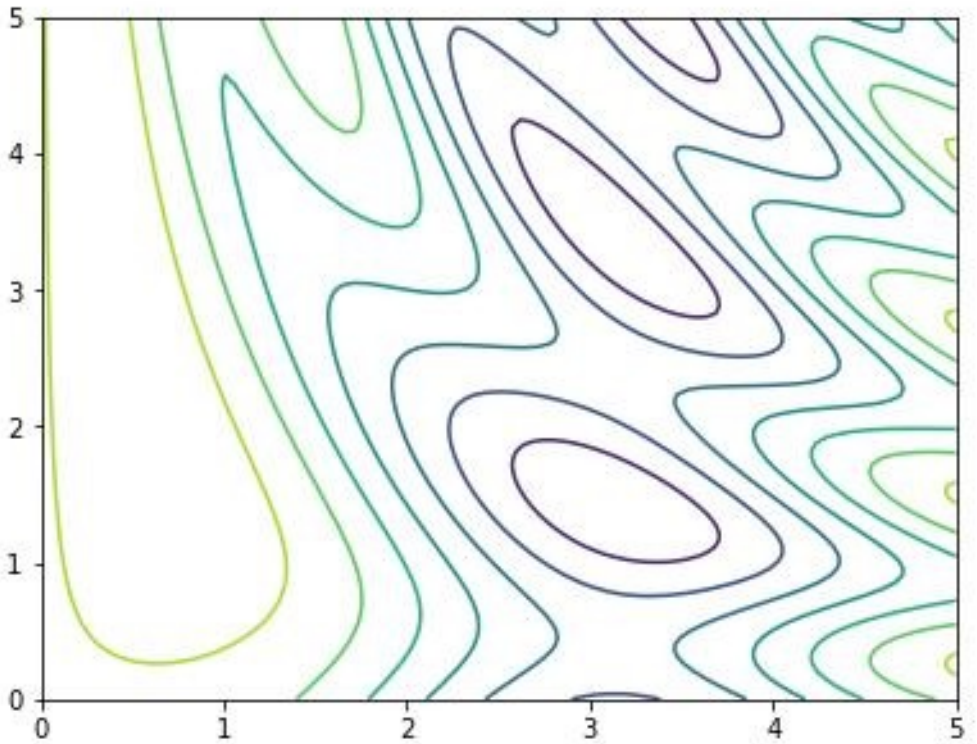
```
fig, ax = plt.subplots()
ax.errorbar(x, y, yerr = 1)
plt.show()
```



Gráficos en 2D – Curvas de nivel

```
def f(x, y):
    return np.sin(x) ** 2 +
np.cos(5 + x * y) + 2 * np.cos(x)

x = np.linspace(0, 5, 100)
y = np.linspace(0, 5, 100)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

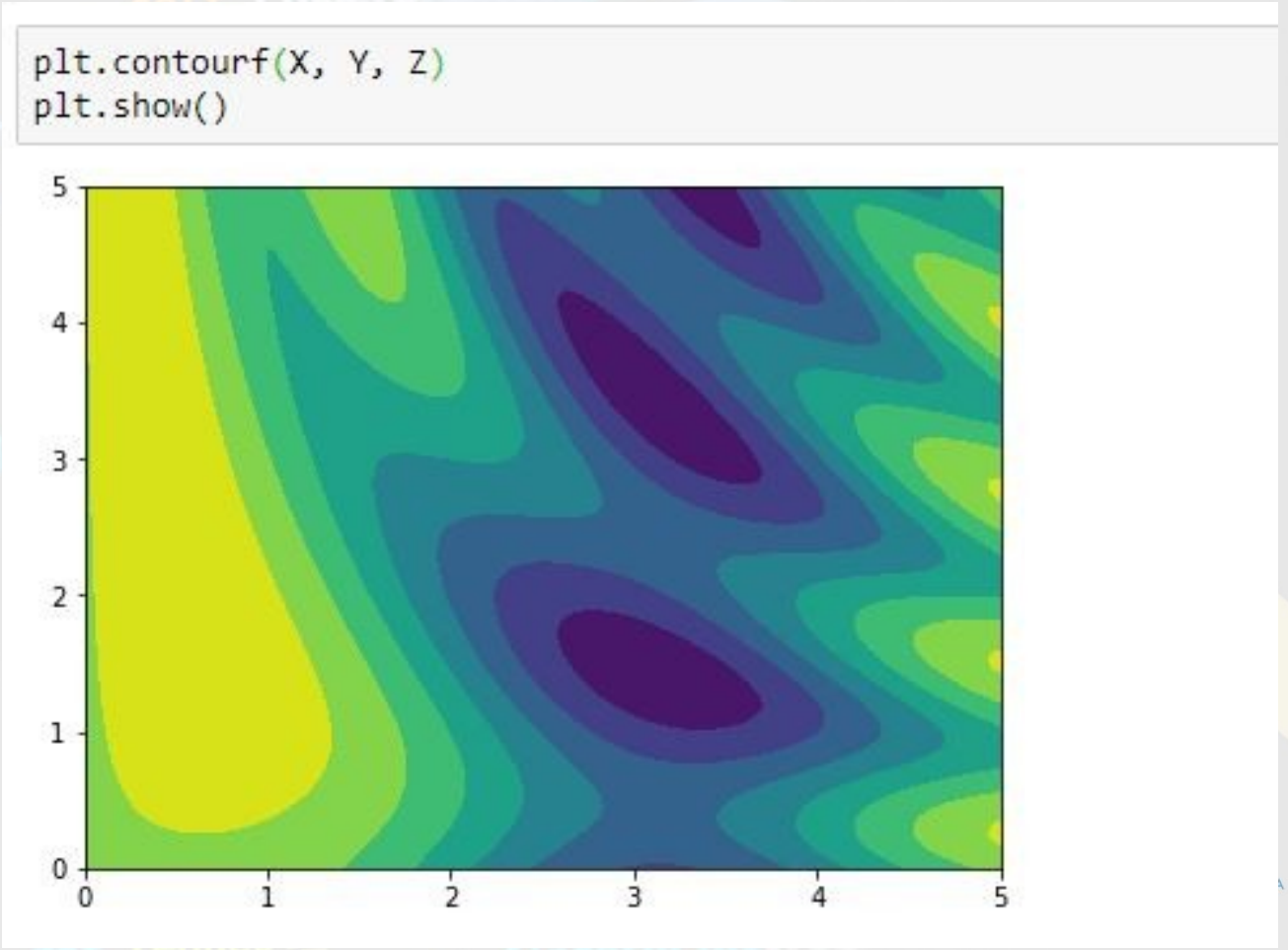
```
plt.contour(X, Y, Z)
plt.show()
```



Gráficos en 2D – Curvas de nivel

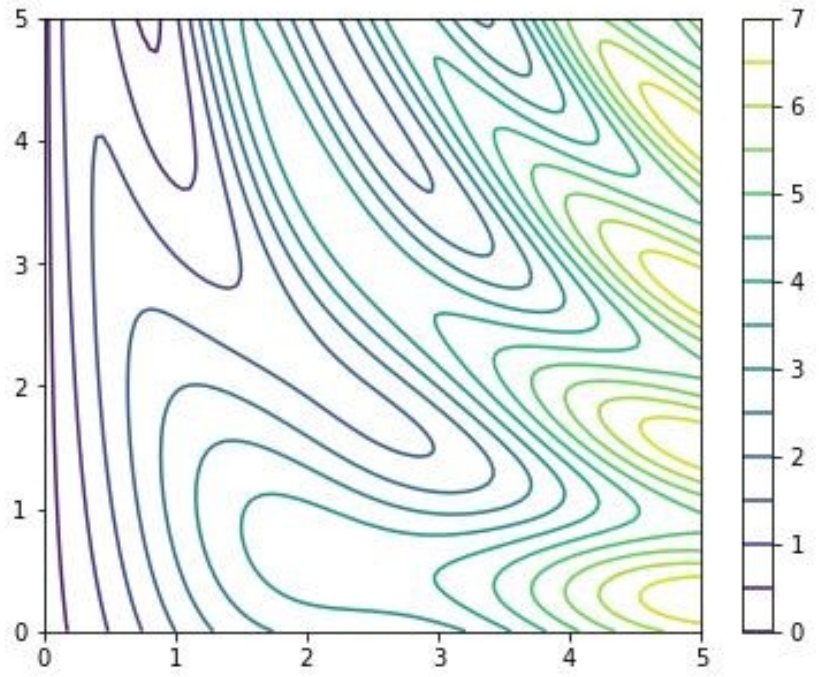
```
def f(x, y):
    return np.sin(x) ** 2 +
np.cos(5 + x * y) + 2 * np.cos(x)

x = np.linspace(0, 5, 100)
y = np.linspace(0, 5, 100)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

```
plt.contourf(X, Y, Z)
plt.show()
```

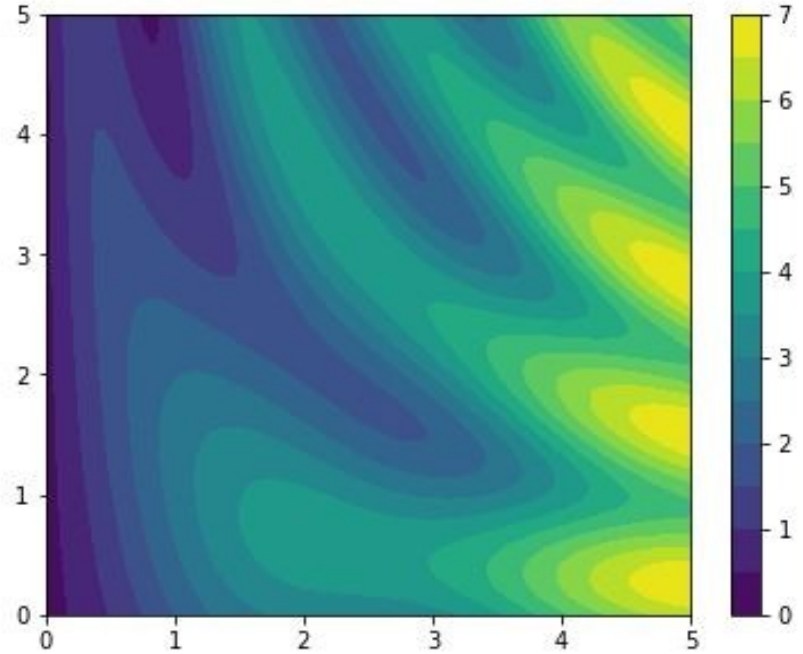


Gráficos en 2D – Curvas de nivel

```
plt.contour(X, Y, Z, levels = 15)
plt.colorbar()
plt.show()
```




```
fig, ax = plt.subplots()
g = ax.contourf(X, Y, Z, levels = 15)
fig.colorbar(g)
plt.show()
```

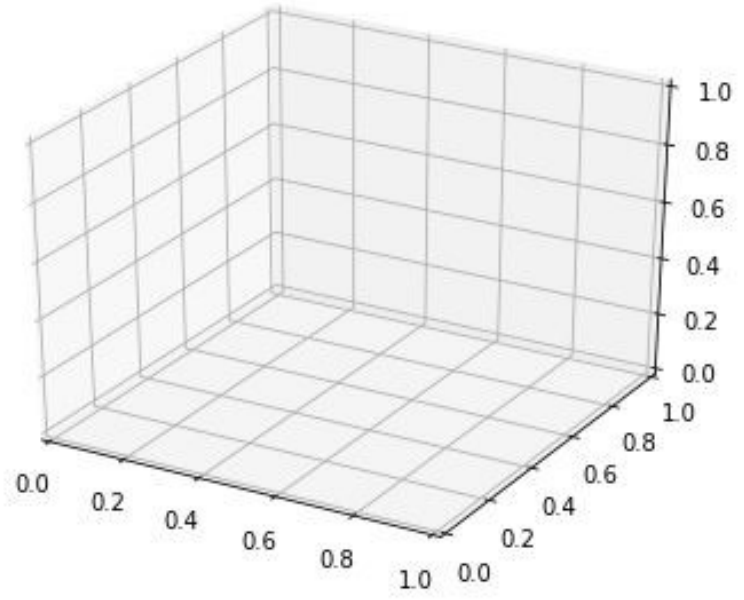


Gráficos en 3D

Para habilitar la creación de este tipo de gráficos basta importar el objeto Axes3D de la sublibrería mpl_toolkits.mplot3d, y generar un conjunto de ejes con el método gca asociado a la figura indicando que la proyección es de tipo "3d"

```
from mpl_toolkits.mplot3d import Axes3D
```

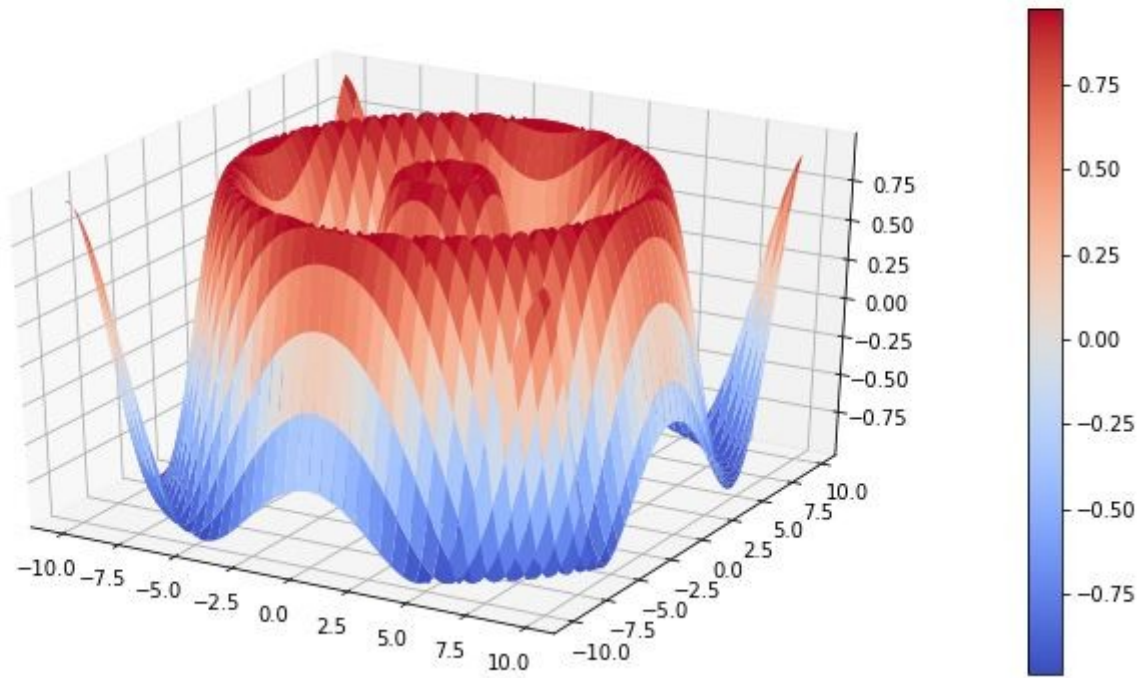
```
fig = plt.figure()
ax = fig.gca(projection='3d')
plt.show()
```



Gráficos 3D - Superficies

```
X = np.arange(-10, 10, 0.25)
Y = np.arange(-10, 10, 0.25)
X, Y = np.meshgrid(X, Y)
Z = np.sin(np.sqrt(X**2 + Y**2))
```

```
fig = plt.figure(figsize = (12, 6))
ax = fig.gca(projection='3d')
surface = ax.plot_surface(X, Y, Z, cmap = "coolwarm")
fig.colorbar(surface)
plt.show()
```

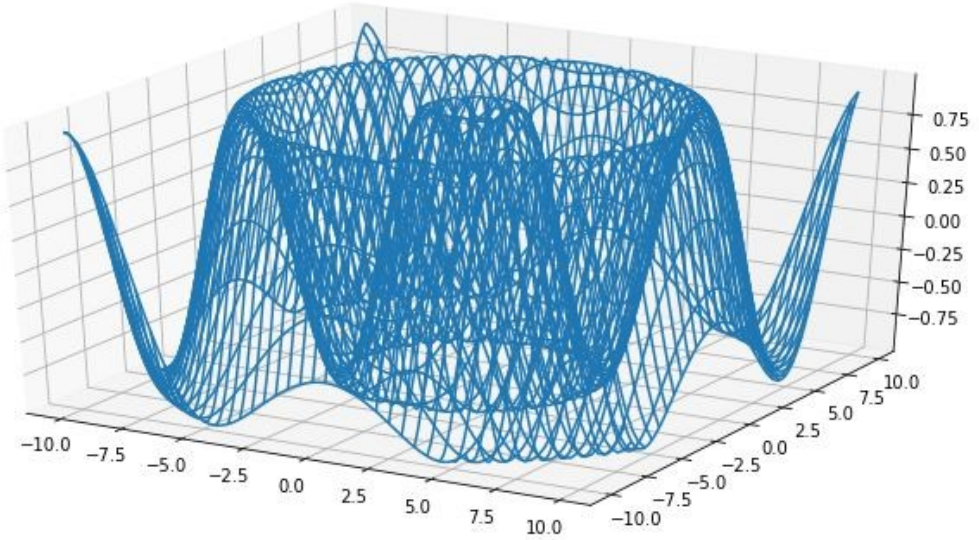


Gráficos 3D - Wireframes

La generación de "wireframes" o gráficos de mallas es extremadamente simple una vez que sabemos generar una superficie. El método es plot_wireframe y recibe como primeros tres argumentos las matrices X, Y y Z que hemos visto para la función plot_surface.

```
X = np.arange(-10, 10, 0.25)
Y = np.arange(-10, 10, 0.25)
X, Y = np.meshgrid(X, Y)
Z = np.sin(np.sqrt(X**2 + Y**2))
```

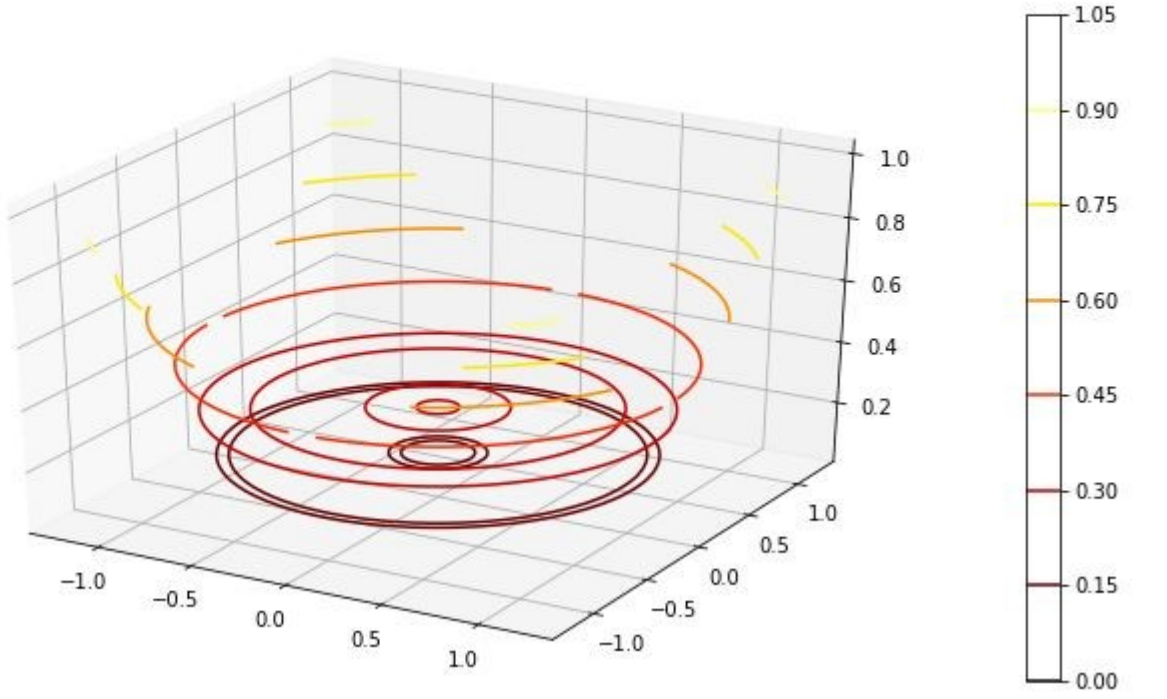
```
fig = plt.figure(figsize = (12, 6))
ax = fig.gca(projection='3d')
surface = ax.plot_wireframe(X, Y, Z)
plt.show()
```



Gráficos 3D - Curvas de nivel

```
fig = plt.figure(figsize = (12, 6))
ax = fig.gca(projection='3d')
surface = ax.contour3D(X, Y, Z, cmap = "hot")
fig.colorbar(surface)
plt.show()
```

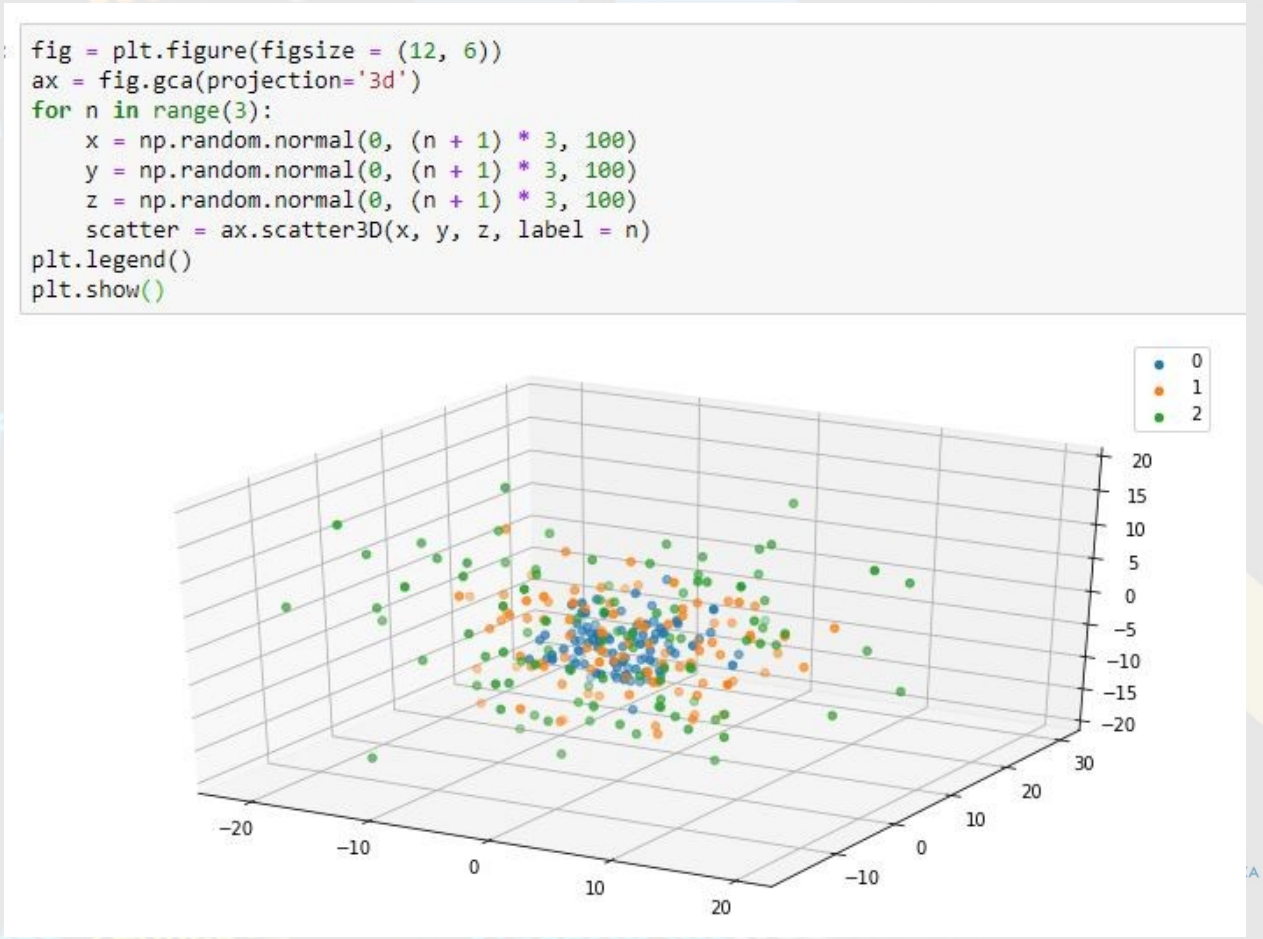
```
X = np.linspace(-1.2, 1.2, 100)
Y = np.linspace(-1.2, 1.2, 100)
X, Y = np.meshgrid(X, Y)
Z = np.abs(0.4 ** 2 - (0.6 - (X ** 2 + Y
** 2) ** 0.5) ** 2) **
```



Gráficos 3D – Dispersión

```
X = np.linspace(-1.2, 1.2, 100)
Y = np.linspace(-1.2, 1.2, 100)
X, Y = np.meshgrid(X, Y)
Z = np.abs(0.4 ** 2 - (0.6 - (X ** 2 +
** 2) ** 0.5) ** 2) **
```

```
fig = plt.figure(figsize = (12, 6))
ax = fig.gca(projection='3d')
for n in range(3):
    x = np.random.normal(0, (n + 1) * 3, 100)
    y = np.random.normal(0, (n + 1) * 3, 100)
    z = np.random.normal(0, (n + 1) * 3, 100)
    scatter = ax.scatter3D(x, y, z, label = n)
plt.legend()
plt.show()
```



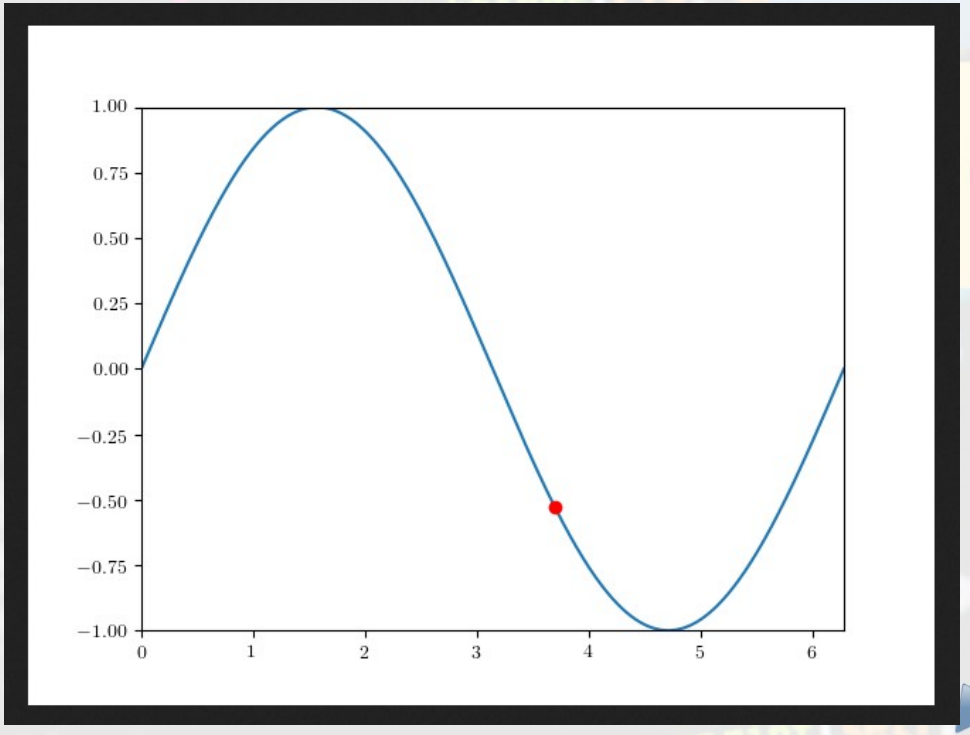
Animaciones

- El paquete matplotlib.animation ofrece algunas clases para crear animaciones.
- FuncAnimation crea animaciones llamando repetidamente a una función.
- Ejemplo: animate() que cambia las coordenadas de un punto en el gráfico de una función sinusoidal.

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

rango = 2*np.pi

fig, ax = plt.subplots()

t = np.arange(0.0, rango, 0.001)
s = np.sin(t)
l = plt.plot(t, s)

ax = plt.axis([0,rango,-1,1])

puntoRojo, = plt.plot([0], [np.sin(0)], 'ro')

def animate(i):
    puntoRojo.set_data(i, np.sin(i))
    return puntoRojo,

# cre animacion usando la funcion animate()
myAnimation = animation.FuncAnimation(fig, animate, frames=np.arange(0.0, rango, 0.1), \
                                      interval=10, blit=True, repeat=True)
# guarda la animacion en 30 frames por segundo
#myAnimation.save('myAnimation.gif', writer='imagemagick', fps=30)

plt.show()
```

Guarda la animación en gif

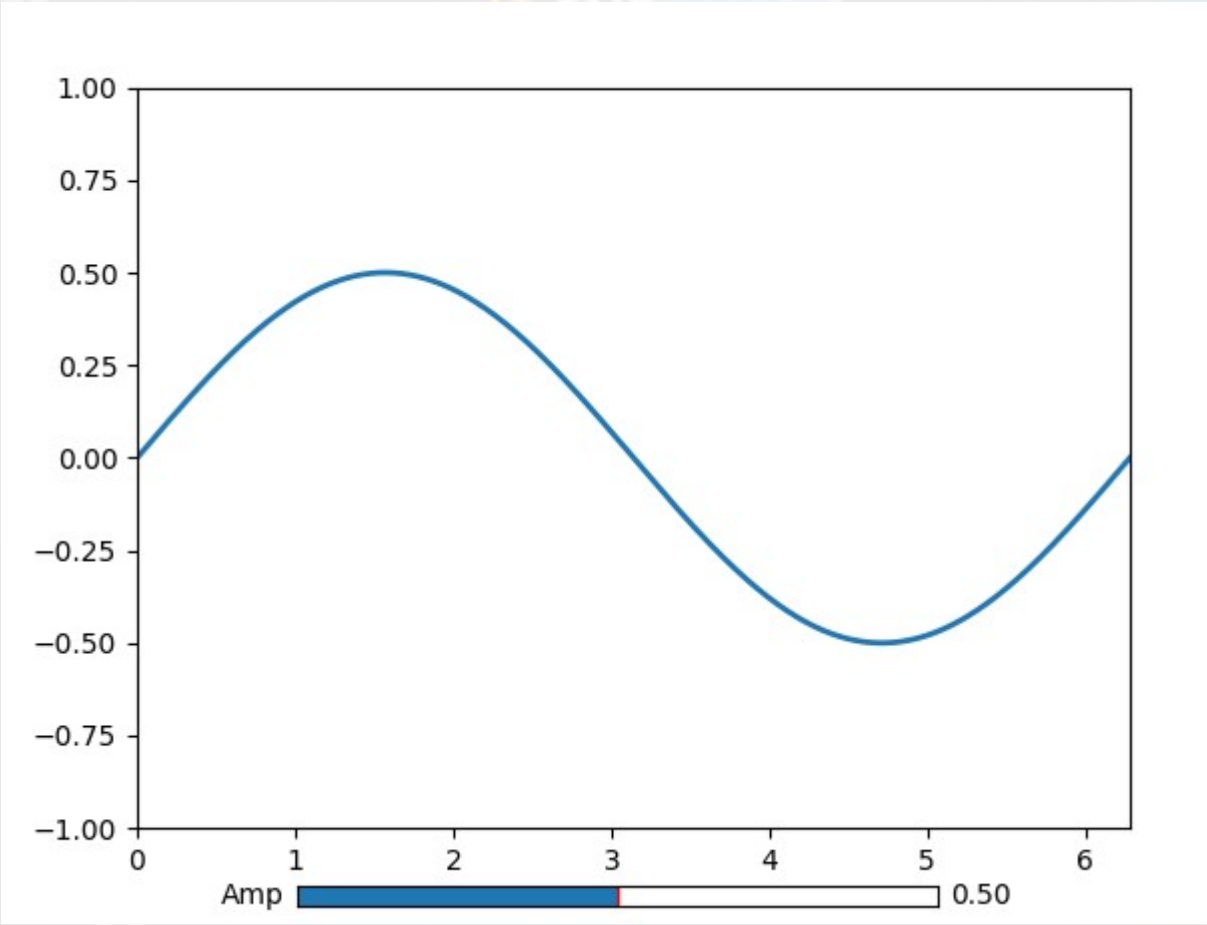
- **save(): método para guardar una Animation de objetos**



Controles interactivos con matplotlib.widgets

- Matplotlib ofrece widgets neutros de GUI. Los widgets requieren un objeto matplotlib.axes.Axes
- La función de actualización es activada por el evento on_changed() del control deslizante

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.widgets import Slider

f = 2*np.pi

fig, ax = plt.subplots()

t = np.arange(0.0, f, 0.001)
initial_amp = .5
s = initial_amp*np.sin(t)
l, = plt.plot(t, s, lw=2)

ax = plt.axis([0,f,-1,1])

axamp = plt.axes([0.25, .03, 0.50, 0.02])
# Slider
samp = Slider(axamp, 'Amp', 0, 1, valinit=initial_amp)

def update(val):
    # amp is the current value of the slider
    amp = samp.val
    # update curve
    l.set_ydata(amp*np.sin(t))
    # redraw canvas while idle
    fig.canvas.draw_idle()

# call update function on slider value change
samp.on_changed(update)

plt.show()
```



Insertando fórmulas TeX

- Las fórmulas TeX se pueden insertar en la gráfica usando la función rc

```
import matplotlib.pyplot as plt

plt.rc(usetex = True)
```

Insertando fórmulas TeX

- Las fórmulas TeX se pueden insertar en la gráfica accediendo a los rcParams

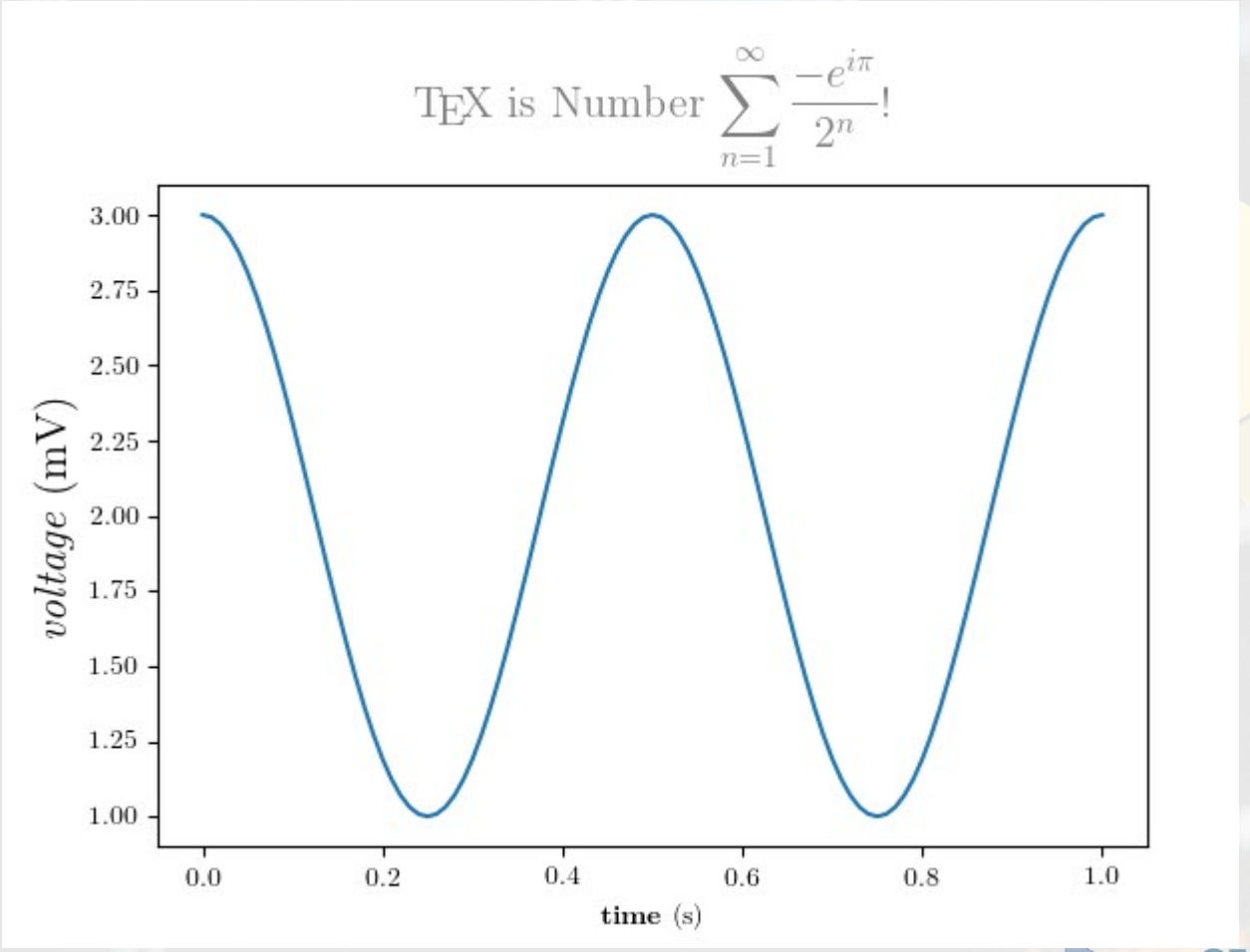
```
import matplotlib.pyplot as plt

params = {'tex.usetex': True}

plt.rcParams.update(params)
```

Insertando fórmulas TeX

- TeX utiliza la barra invertida \ para comandos y símbolos, que puede entrar en conflicto con caracteres especiales en las cadenas de Python.
- Para utilizar barras diagonales literales en una cadena de Python, deben ser evadidas o incorporadas en una cadena en bruto:

```
plt.xlabel('\\alpha')
plt.xlabel(r'\alpha')
```

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


# Example data
t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(4 * np.pi * t) + 2

plt.rc('text', usetex=True)
plt.rc('font', family='serif')

plt.plot(t, s)

plt.xlabel(r'\textbf{time} (s)')
plt.ylabel(r'\textit{voltage} (mV)',fontsize=16)
plt.title(r"\TeX\ is Number "
          r"$\displaystyle\sum_{n=1}^\infty\frac{-e^{i\pi}}{2^n}$!",
          fontsize=16, color='gray')
# Make room for the ridiculously large title.
plt.subplots_adjust(top=0.8)

plt.savefig('tex_demo')
plt.show()
```


TEX is Number $\sum_{n=1}^{\infty} \frac{-e^{i\pi}}{2^n}!$
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
voltage (mV)
0.0
0.2
0.4
0.6
0.8
1.0
time (s)