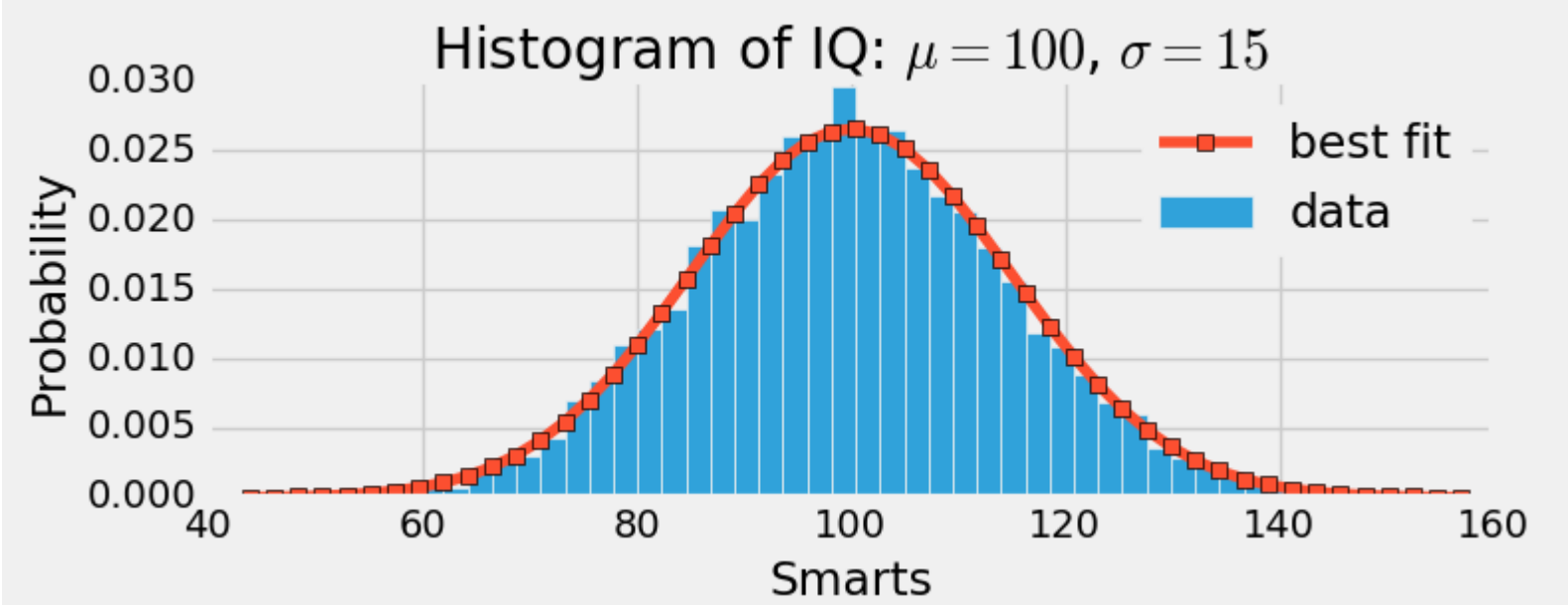
Libreria de Gràficos

matplotlib



Repàs de Numpy

Valores consecutivos

https://numpy.org/doc/stable/reference/generated/numpy.arange.html

```
numpy.arange(start=0, stop, step = 1)
```

```
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
>>> np.arange(3,7)
array([3, 4, 5, 6])
>>> np.arange(3,7,2)
array([3, 5])
```

Valores Random

```
np.random.rand(10,2) -> Tamaños de la dimensiones de la matriz
np.random.randint(low, high, size)-> Vector de enteros
np.random.uniform(low, high, size)-> Vector de floats
```

```
print ("Valores random en dimensiones (sized1, sized2, sizedn)")
print (np.random.rand(2, 3, 4))
print ("Lo mismo pero valores de distribución normal")
print(np.random.randn(2, 3, 4))

print ("Enteros random : low (incl), high (excl), size")
low = 2
high = 4
size=10
print(np.random.randint(low, high, size))


size=10
print ("Retorna un número de floats entre [0.0, 1.0).")
print (np.random.random([size]))
print ("Retorna un número de floats entre low, to high, size")
print (np.random.uniform(2,80,10))
print ("Genera ejemplos randoms tomados de un array : a[, size,
replace, p]")
print (np.random.choice([1,2,3,4,5,6],4))
```

- Matplotlib es una biblioteca
  del lenguaje de programación Python
  con su extensión matemática NumPy
  que sirve para la generación de gráficos
  a partir de datos contenidos en listas o arrays.

- Proporciona una API, pylab, diseñada para recordar a la de MATLAB.

https://matplotlib.org/3.1.1/tutorials/index.html

https://matplotlib.org/tutorials/introductory/pyplot.html

MATLAB

MATLAB (abreviatura de ***MAT****rix* ***LAB****oratory*, "laboratorio de matrices") es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux .

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las *cajas de herramientas* (*toolboxes*); y las de Simulink con los *paquetes de bloques* (*blocksets*).

Es un software muy usado en universidades y centros de investigación y desarrollo. En los últimos años ha aumentado el número de prestaciones, como la de programar directamente procesadores digitales de señal o crear código VHDL.

En 2004, se estimaba que MATLAB era empleado por más de un millón de personas en ámbitos académicos y empresariales.[1]

Gráficos e interfaces gráficas

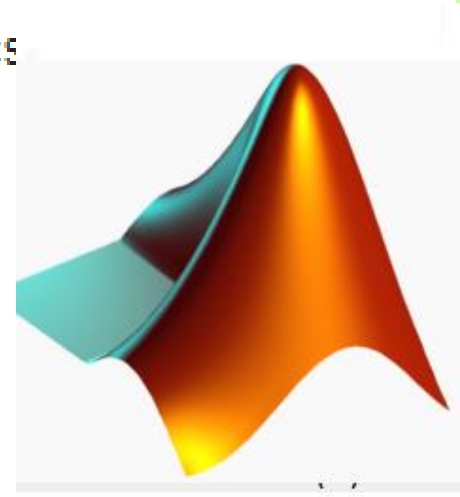
MATLAB provee funciones y herramientas para visualizar datos en 2D y 3D.

MATLAB

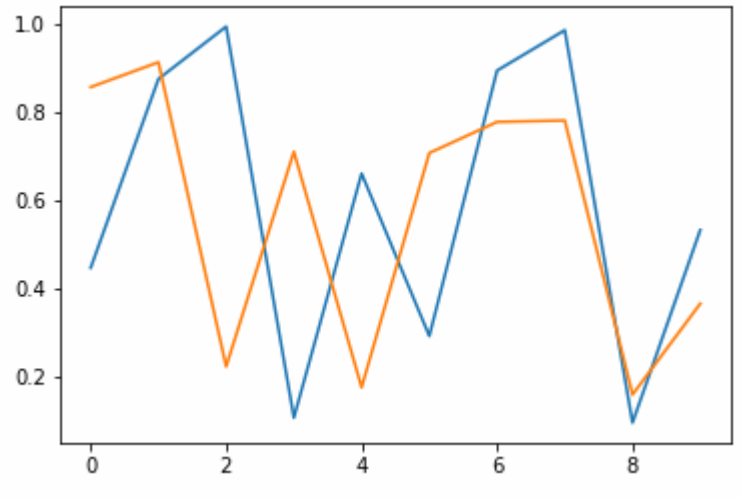
- Parallel Computing
- Math, Statistics, and Optimization
- Control Systems
- Signal Processing and Communication
- Image Processing and Computer Vision
- Test and Measurement
- Computational Finance
- Computational Biology
- Code Generation and Verification
- Application Deployment
- Database Connectivity and Reporting
- MATLAB Report Generator
- Text Analytics Toolbox™

Simulink

- Event-Based Modeling
- Physical Modeling
- Control Systems
- Signal Processing and Communications
- Code Generation
- Real-Time Simulation and Testing
- Verification, Validation, and Test
- Simulation Graphics



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

plt.plot(np.random.rand(10))
plt.plot(np.random.rand(10))
plt.show()
```

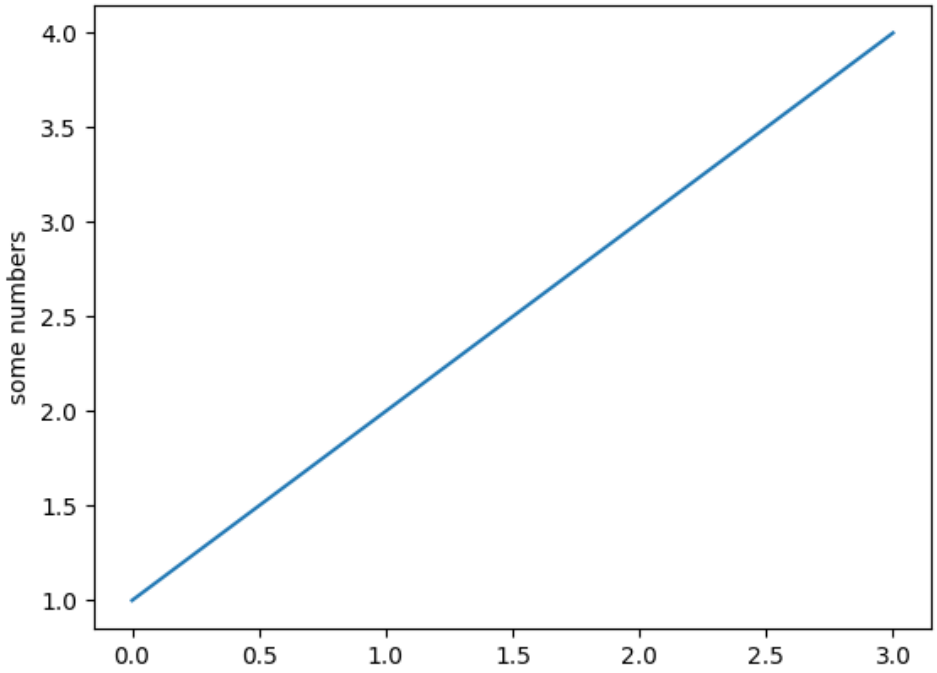


matplotlib.pyplot es una colección de funciones que hacen que matplotlib funcione como MATLAB.
Cada función pyplot hace algún cambio en una figura: por ejemplo,

- crear una figura,
- crear un área de trazado en una figura,
- trazar líneas en un área de trazado,
- decorar la trama con etiquetas, etc.

La generación de visualizaciones con pyplot es muy rápida:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```

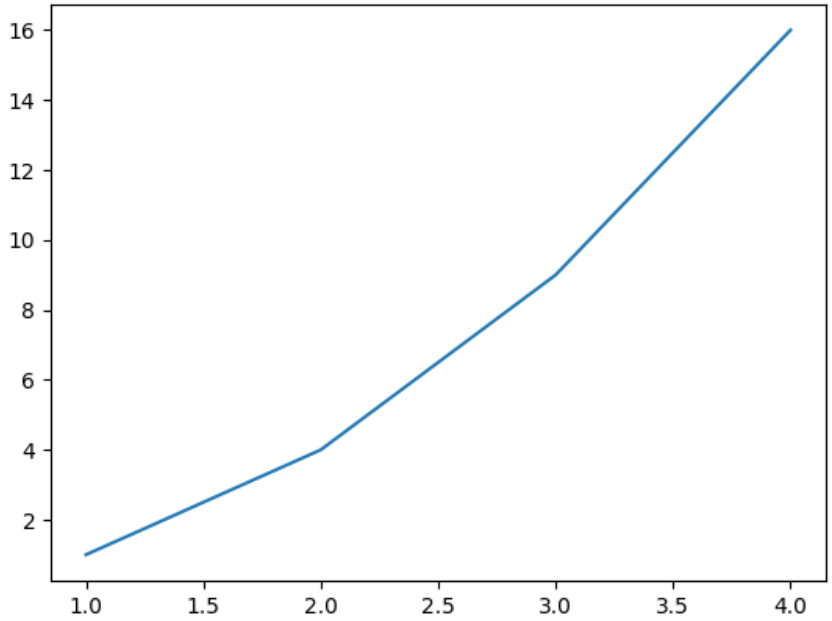


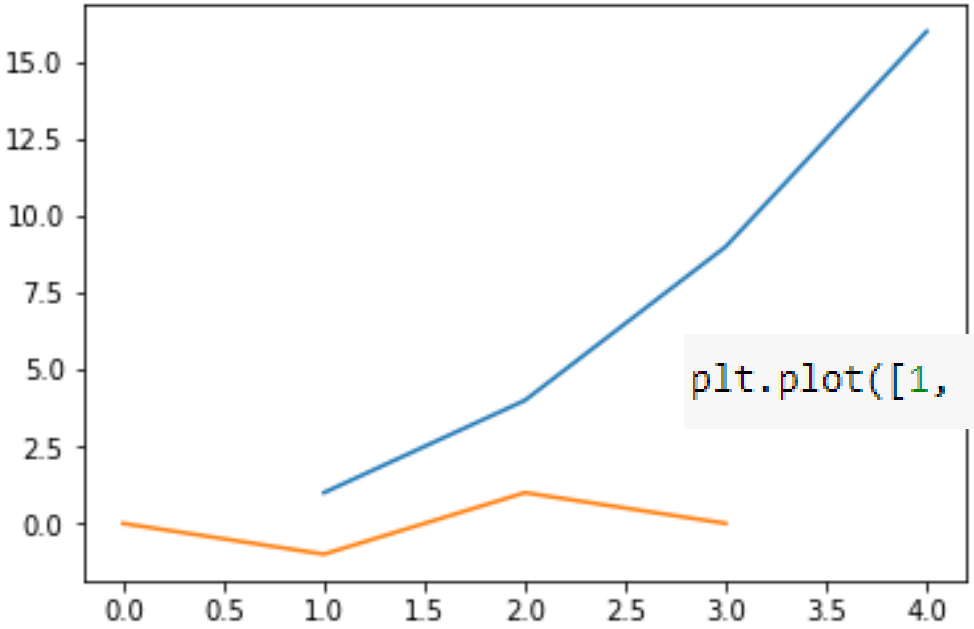
- Quizás se esté preguntando por qué el eje x varía de 0-3 y el eje y de 1-4?
- Si proporciona una sola lista o matriz al plot(), el comando, matplotlib asume que es una secuencia de valores y y genera automáticamente los valores
- Dado que los rangos de python comienzan con 0, el vector x predeterminado tiene la misma longitud que y, pero comienza con 0. Por lo tanto, los datos de x son [0,1,2,3].

`plot()` es un comando versátil, y tomará un número arbitrario de argumentos.

Por ejemplo, para trazar x y, puede emitir el comando:

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```

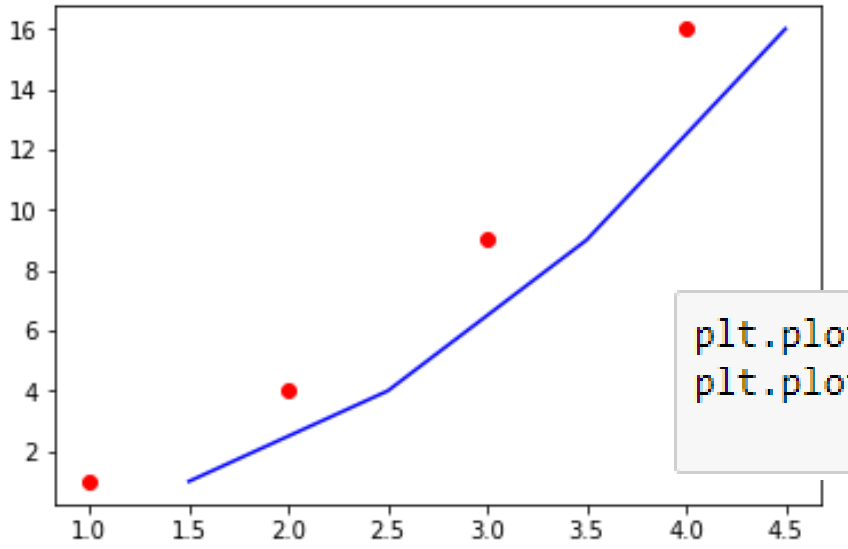





```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], [0, -1, 1, 0])
```

Estilo de la trama

- Hay un tercer argumento opcional que es la cadena de formato que indica el color y el tipo de línea de la gráfica.
- Las letras y los símbolos de la cadena de formato son de MATLAB, se concatena una cadena de color con una cadena de estilo de línea.
- La cadena de formato predeterminada es 'b-', que es una línea azul continua.
- Por ejemplo, para trazar lo anterior con círculos rojos, se escribiria “ro”

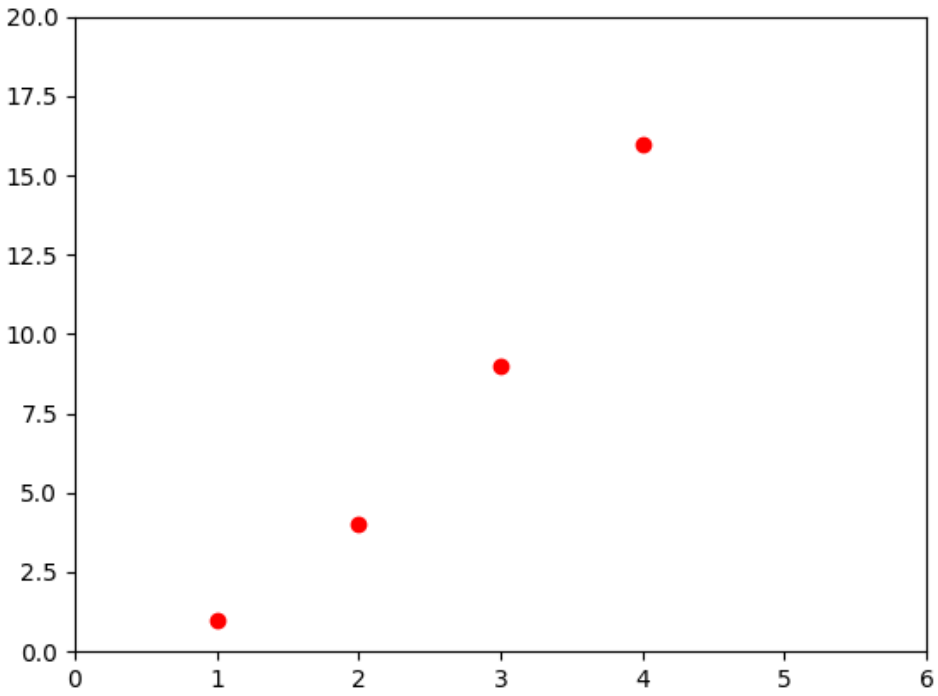



```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], "ro")
plt.plot([1.5, 2.5, 3.5, 4.5], [1, 4, 9, 16], "-b")
```

Consulte la documentación de plot() para obtener una lista completa de estilos de línea y las cadenas de formato.

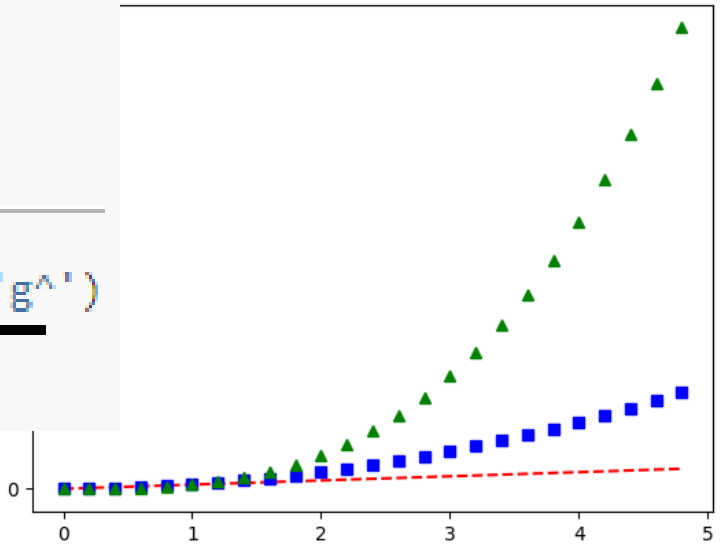
El comando axis() define la ventana gráfica según los parámetros [xmin, xmax, ymin, ymax]

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



Si matplotlib se limitara a trabajar con listas, sería bastante inútil para el procesamiento numérico. En general, utilizará matrices numpy . De hecho, todas las secuencias se convierten internamente en matrices numpy. El siguiente ejemplo ilustra un trazado de varias líneas con diferentes estilos de formato en un comando utilizando matrices.

```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

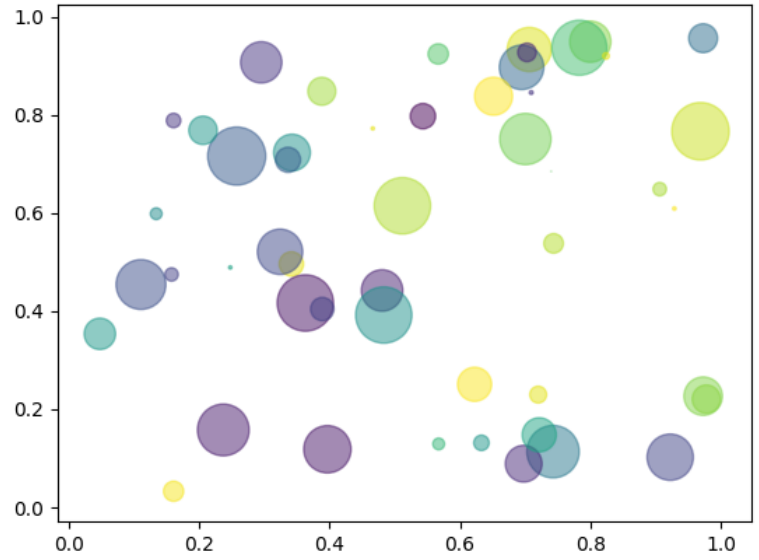



```
import matplotlib.pyplot as plt

t = np.arange(0.0, 5.0, 0.2)
plt.plot(t,t,'r--', t,t**2, 'bs', t,t**3,'g^')
plt.show()
```

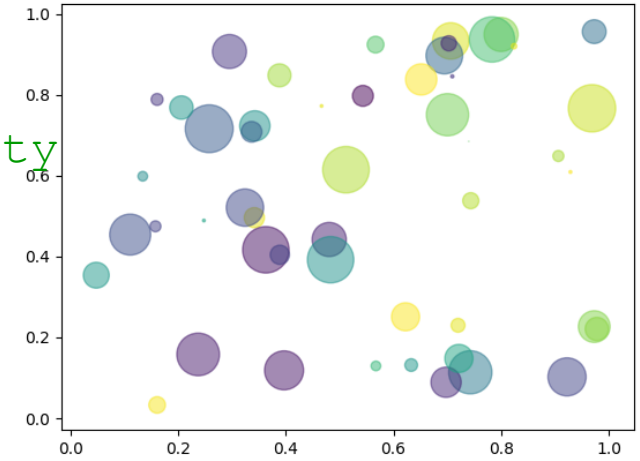
Gráficos de burbujas

```
pyplot.scatter( x , y ,                 # Posiciones de datos en forma de array
                tamayo ,                # Tamaño del punto en forma  escalar o en array
                color  ,                # Un color, o secuencia de colores, o matriz RGB
                marcador ,              # Estilo del marcador, por defecto "o"
                color_map  ,            # mapa de colores
                norm_color ,            # normalizar la luminancia del color 0-1
                vmin  ,                 # máximo y mínimo de la matriz de colores
                vmax  ,
                alfa  ,                 # transparencia 0=transparente  1=opaco
                ancho_de_linea ,        # línea de los bordes del marcador
                verts  ,
                edgecolors ,            # Color de las esquinas
                '*'  ,
                data ,
                '** kwargs' )
```

```
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
```



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

# Fixing random state for reproducibility
np.random.seed(19680801)


N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



https://matplotlib.org/3.1.1/gallery/shapes_and_collections/scatter.html#sphx-glr-gallery-shapes-and-collections-scatter-py

Gráficos de burbujas

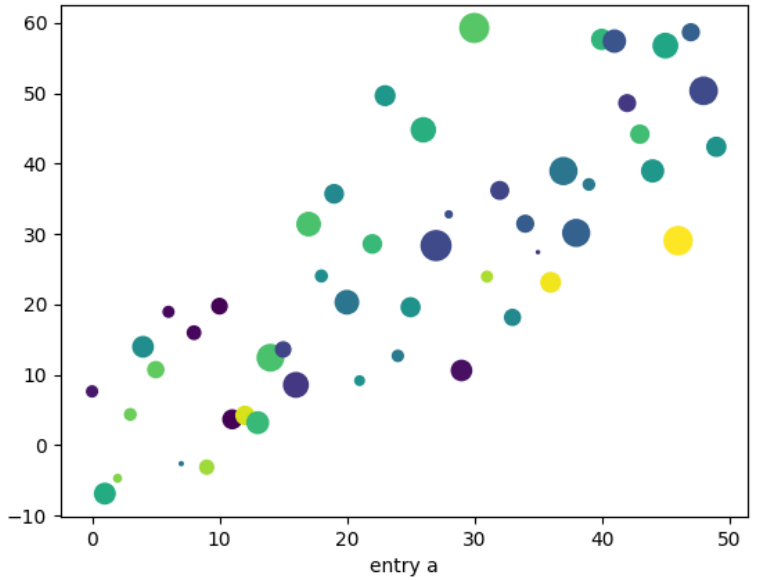
Nota

Además de los argumentos descritos anteriormente, esta función puede tomar un argumento de palabra clave de **datos** . Si se proporciona dicho argumento de **datos** , los siguientes argumentos son reemplazados por los **datos [<arg>]** :

- Todos los argumentos con los siguientes nombres: 'c', 'color', 'edgecolors', 'facecolor', 'facecolors', 'linewidths', 's', 'x', 'y'.

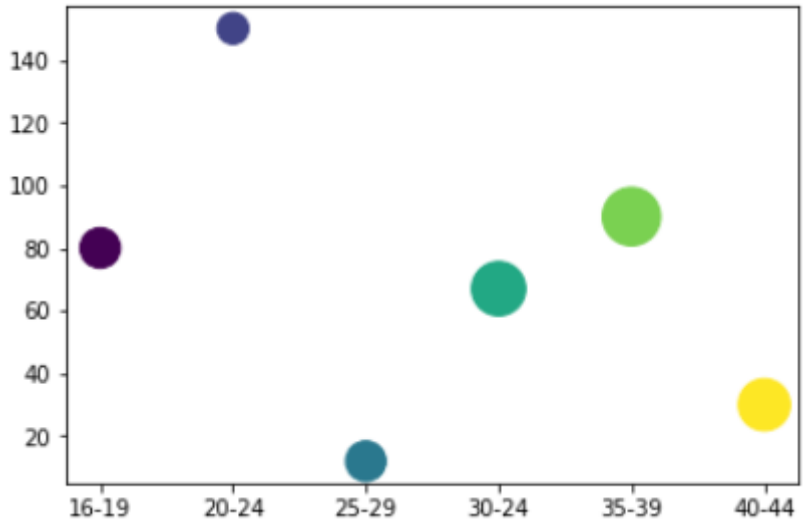
Los objetos pasados como **datos** deben admitir el acceso a elementos (`data[<arg>]`) y la prueba de membresía ( ).`<arg> in data`

```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```



Compras de móviles por franja de edad en unidades y en valor total. (datos no reales)

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

data = {'a': np.arange(1)}
data ['a'] = ['16-19', '20-24', '25-29', '30-24', '35-39','40-44']
data ['unidades'] = [80, 150, 12, 67, 90, 30]
data ['valor'] = [335 , 203, 334 , 604, 700, 555]
data ['c'] = [1,2,3,4,5,6]

plt.scatter('a', 'unidades', c='c', s='valor', data = data)
plt.show()
```



Práctica P02

Ejercicio 1. Estudia este código y realiza los siguientes cambios.

1. Pasa de 50 elementos random a 12
2. Imprime las matrices : a, b, c, d, antes de mostrar el gráfico.
3. Crea una documentación completa, para el ejercicio.
    1. Debe figurar el título de cada matriz y sus valores
    2. y luego la captura de pantalla del resultado.

```
data = {'a': np.arange(50),
'c': np.random.randint(0,50,50),
'd': np.random.randn(50)}

data['b'] = data ['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100
plt.scatter('a','b',c='c', s='d', data=data)
plt.xlabel = 'entry a'
plt.ylabel = 'entry b'
plt.show()
```

Práctica P03

- Genera con matplotlib un gráfico de dispersión (scatter) con los datos :
  - de temperaturas medias de 12 meses del año,
  - y volumen de lluvia mensual.

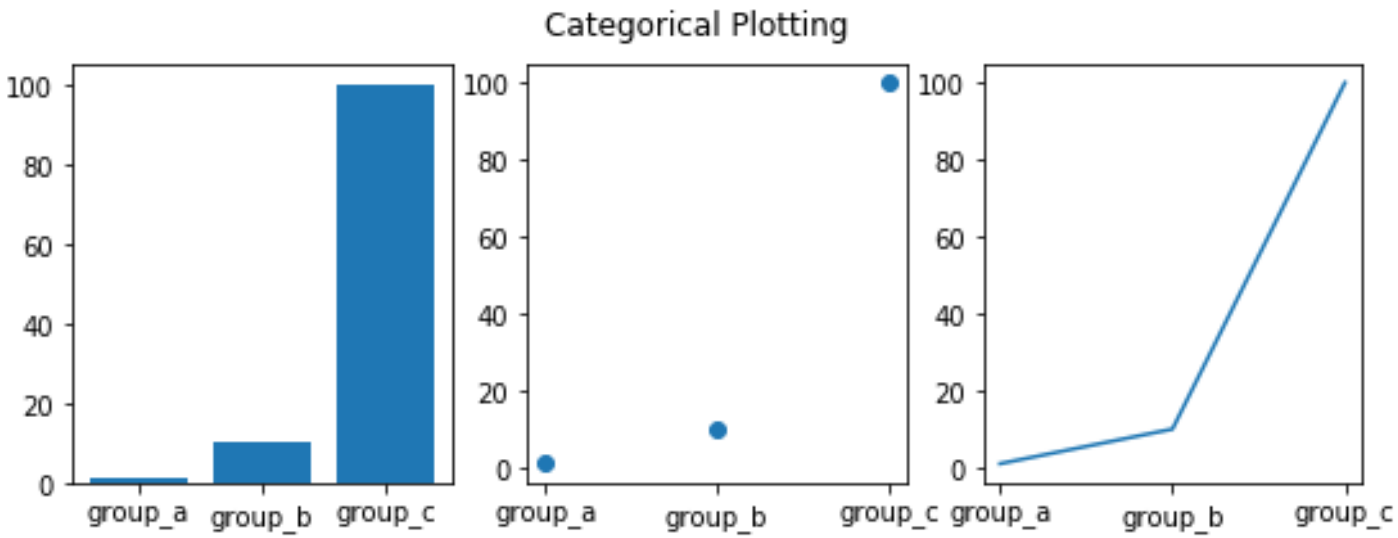
Te dan 6 meses y tienes que generar los otros 6 de forma aleatoria en un rango de valores posibles.

```
meses = [ 'Gen', 'Feb', 'Mar', 'Abr', 'Mayo', 'Junio']
temperaturas = [12.5, 12.3, 15.2, 16.4, 20.5, 22.2]
plujamm = [335, 203, 334, 604, 700, 555]
```

- Tienes que mostrar 12 valores para x = (meses) y = (temperatura media)
- Medidas de los puntos, proporcionales a el volumen de lluvia cada mes
- Colores diferentes para cada mes

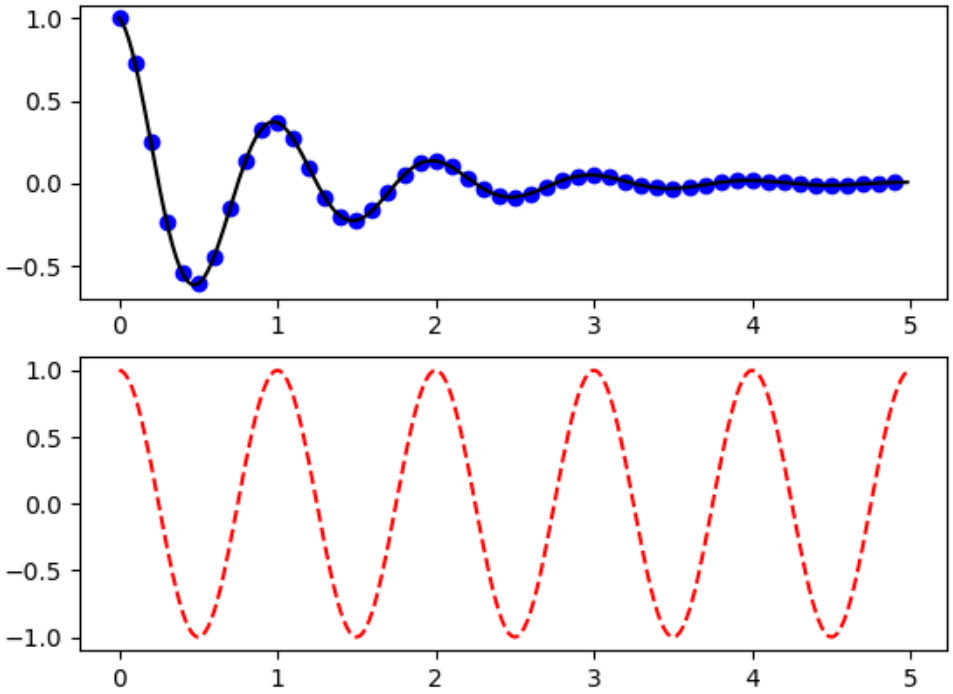
Subgráficos

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(1, figsize=(9, 3))
#------------------------------ Subgráfico en barras
plt.subplot(131)
plt.bar(names, values)
#------------------------------ Subgráfico en puntos
plt.subplot(132)
plt.scatter(names, values)
#------------------------------ Subgráfico en líneas
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



Trabajando con múltiples figuras y ejes

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```



Práctica P04

Con los conocimientos actuales, decide la mejor manera de representar esta tabla en colores diferentes por sexo.

	16 a 19 años	20 a 24 años	25 a 29 años	30 a 34 años	35 a 39 años	40 a 44 años
	2018	2018	2018	2018	2018	2018
Ambos sexos						
Estudios primarios incompletos	39,90	35,20	51,70	62,50	60,40	66,40
Hombres						
Estudios primarios incompletos	48,70	45,30	66,30	76,10	75,50	75,40
Mujeres						
Estudios primarios incompletos	23,70	24,40	33,80	45,80	45,10	54,80

Control de propiedades de línea

Las líneas tienen muchos atributos que puede configurar: ancho de línea, estilo de guión, suavizado, etc. ver `matplotlib.lines.Line2D`. Hay varias maneras de establecer propiedades de línea

- Utilice args de palabras clave:

```
plt.plot(x, y, linewidth=2.0)
```

- Utilice los métodos setter de una `Line2D`instancia. `plot`devuelve una lista de `Line2D`objetos; por ejemplo, . En el código a continuación, supondremos que solo tenemos una línea para que la lista devuelta sea de longitud 1. Usamos el desempaquetado de tuplas para obtener el primer elemento de esa lista:`line1, line2 = plot(x1, y1, x2, y2)line,`

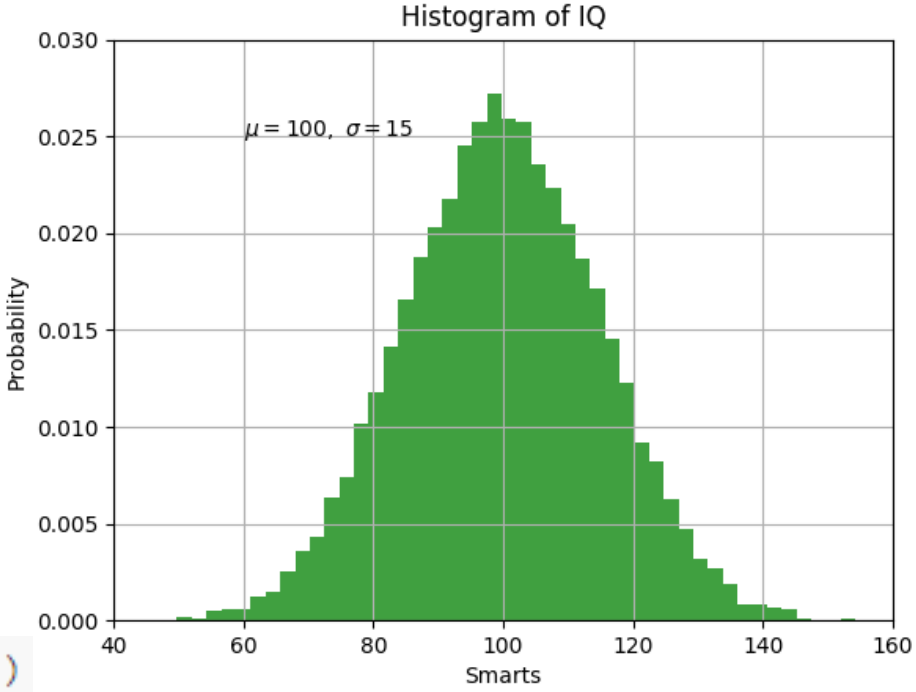
```
line, = plt.plot(x, y, '-')
line.set_antialiased(False) # turn off antialising
```

- Usa `setp()` para establecer varias propiedades en una lista de líneas. estilo MATLAB

  Puede usar argumentos de palabras clave de python o pares de cadena / valor de estilo MATLAB:

```
lines = plt.plot(x1, y1, x2, y2)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)
# or MATLAB style string value pairs
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



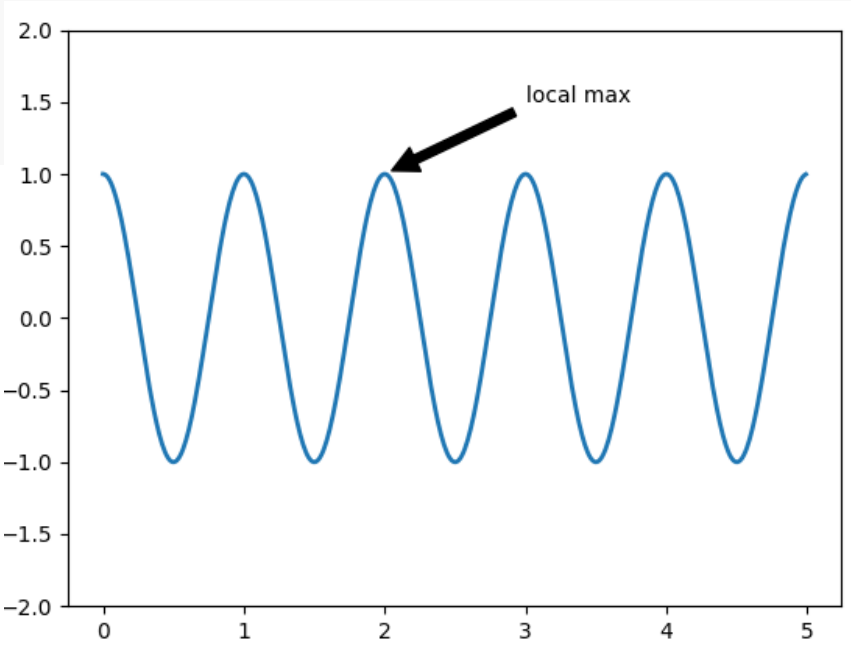
El comando text() se puede usar para agregar texto en una ubicación arbitraria, y el xlabel(), ylabel() y title() se usa para agregar texto en las ubicaciones indicadas

```
t = plt.xlabel('my data', fontsize=14, color='red')
```

Anotando texto

Los usos del `text()`comando básico anterior colocan el texto en una posición arbitraria en los ejes. Un uso común del texto es anotar algunas características de la trama, y el `annotate()`método proporciona una funcionalidad de ayuda para facilitar las anotaciones. En una anotación, hay dos puntos a considerar: la ubicación que se anota representada por el argumento `xy`y la ubicación del texto `xytext`. Ambos de estos argumentos son `(x,y)`tuplas.

```
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )

plt.ylim(-2, 2)
plt.show()
```



En este ejemplo básico, tanto la xy(punta de la flecha) como las xytext ubicaciones (ubicación del texto) están en coordenadas de datos. Hay una variedad de otros sistemas de coordenadas que puede elegir: vea Anotación básica

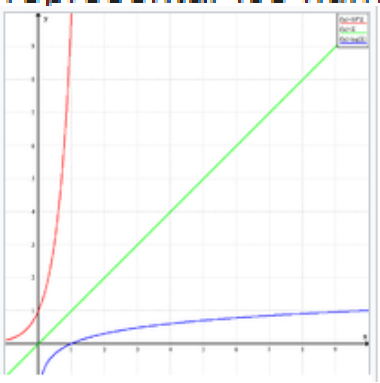
y Anotación avanzada para más detalles.

Escala logarítmica

Una **escala logarítmica** es una escala de medida que utiliza el logaritmo de una cantidad física en lugar de la propia cantidad.

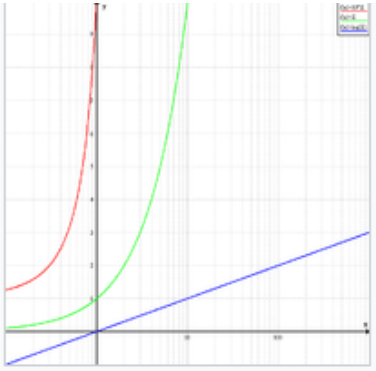
Un ejemplo sencillo de escala logarítmica muestra divisiones igualmente espaciadas en el eje vertical de un gráfico marcadas con 1, 10, 100, 1000, ... en vez de 0, 1, 2, 3, ...

La presentación de datos en una escala logarítmica puede ser útil cuando los datos cubren una amplia gama de valores - el logaritmo los reduce a un rango más manejable. Algunos de nuestros sentidos funcionan de manera logarítmica (ley de Weber-Fechner), lo que hace especialmente apropiadas a las escalas logarítmicas para representar estas cantidades. En particular, nuestro sentido del oído percibe cocientes iguales de frecuencias como diferencias iguales en el tono. Además, los estudios en niños pequeños y en tribus aisladas han demostrado que las escalas logarítmicas pueden ser la manera más natural de representar los números por parte de los seres humanos.[1]



Gráfica lineal-lineal
Las funciones representadas son:
$y = x$, $y = 10^x$, $y = \log(x)$.

Gráfica lineal-logarítmica (semilogarítmica).
$y = x$, $y = 10^x$, $y = \log(x)$.



Logarítmicos y otros ejes no lineales

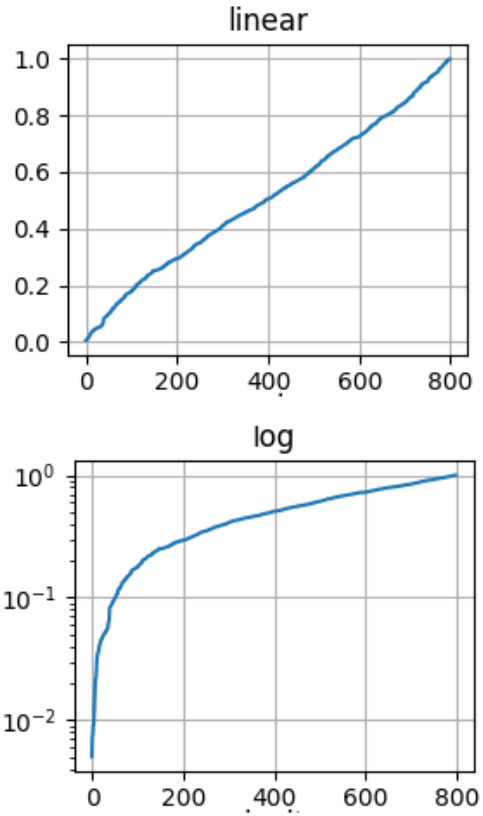
`matplotlib.pyplot` no solo admite escalas de ejes lineales, sino también escalas logarítmicas y logit. Esto se usa comúnmente si los datos abarcan muchos órdenes de magnitud. Cambiar la escala de un eje es fácil:

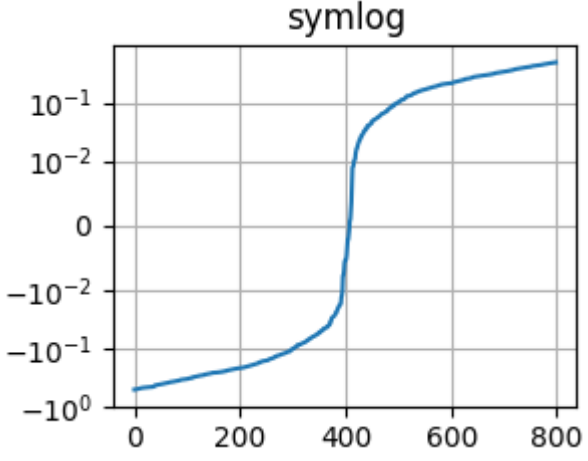
```
plt.xscale ('log')
```

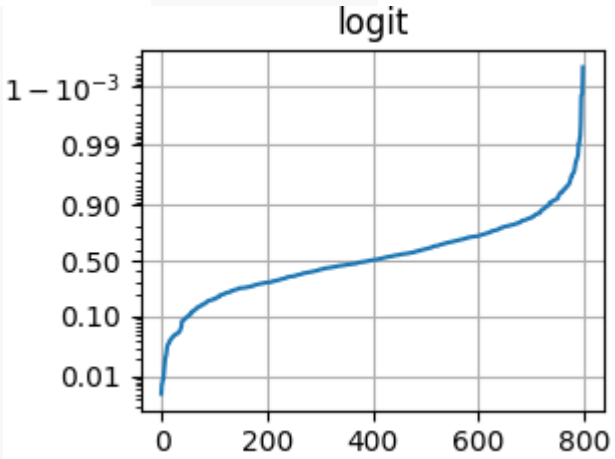
https://matplotlib.org/tutorials/introductory/pyplot.html

A continuación se muestra un ejemplo de cuatro gráficos con los mismos datos y diferentes escalas para el eje y.

```python
from matplotlib.ticker import NullFormatter  # useful for `logit` scale

# Fixing random state for reproducibility
np.random.seed(19680801)

# make up some data in the interval ]0, 1[
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))

# plot with various axes scales
plt.figure(1)

# linear
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)
```








```
# symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthreshy=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
# Format the minor tick labels of the y-axis into empty strings with
# `NullFormatter`, to avoid cumbering the axis with too many labels.
plt.gca().yaxis.set_minor_formatter(NullFormatter())
# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
                    wspace=0.35)

plt.show()
```