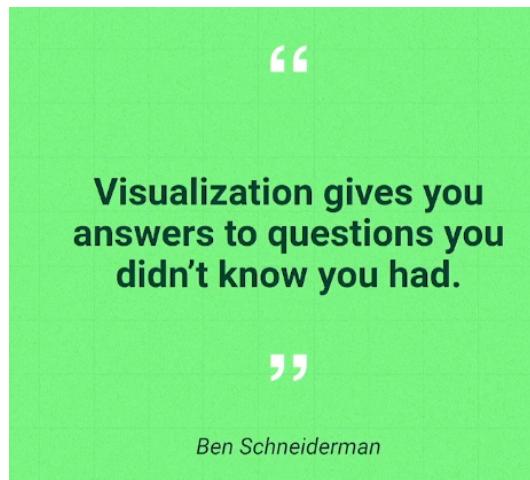
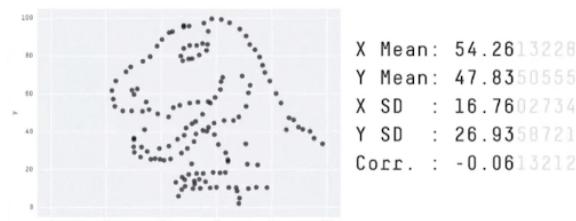
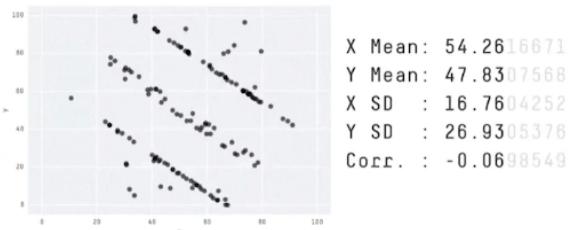
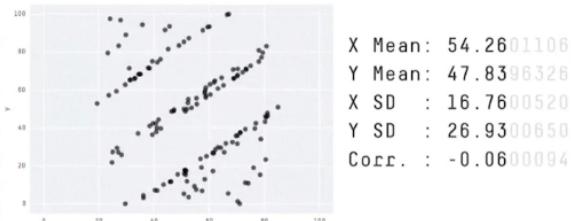
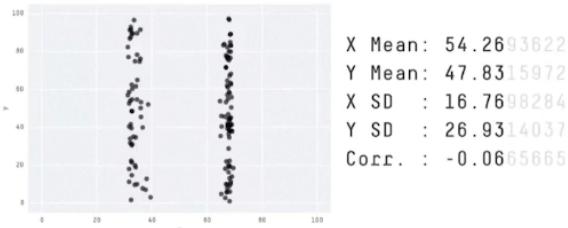
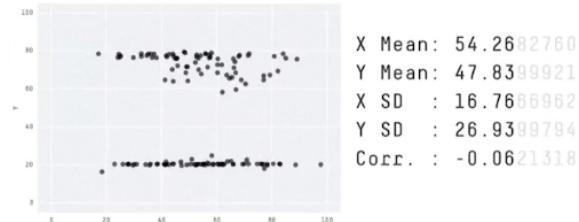
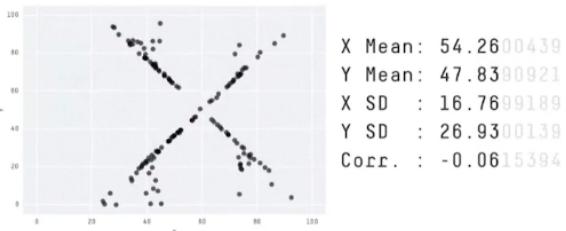


■ Matplotlib ■

¿Por qué es importante la visualización de datos?

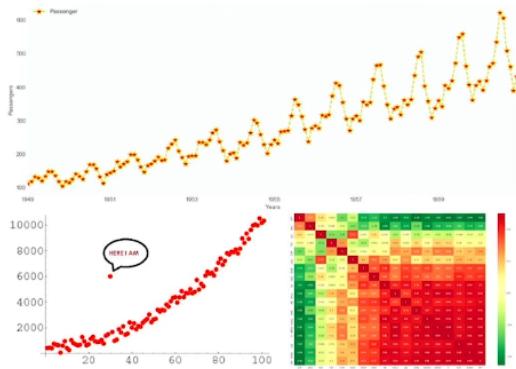


"La visualización nos da respuestas a preguntas que no sabíamos que teníamos"



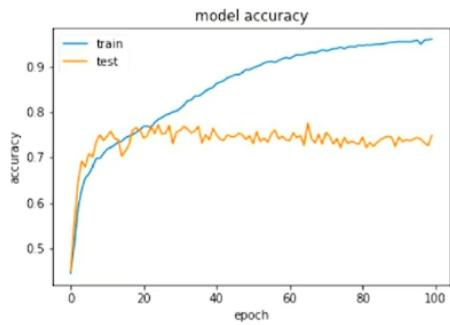
Para diferentes distribuciones de puntos podemos obtener los mismos estadísticos

Hallazgos en nuestros datos



Encontrar tendencias
Comportamiento en los datos
Datos atípicos

Mayor claridad en nuestro código



Orientado al ML e IA

Ver las líneas de entrenamiento de cualquier algoritmo

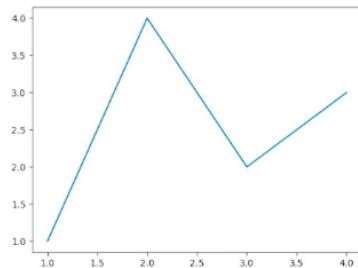
Matplotlib

- Escrita por John D. Hunter.
- Creada en 2003.
- Emula comandos de MATLAB.
- Usa NumPy.
- Escrita en Python.



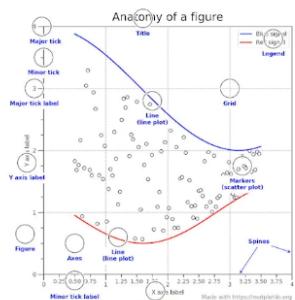
Simple

```
fig, ax = plt.subplots() # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3]); # Plot some data on the axes.
```



Gráficos sencillos de realizar

Personalizable



Diferentes atributos de podemos modificar

Pyplot

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

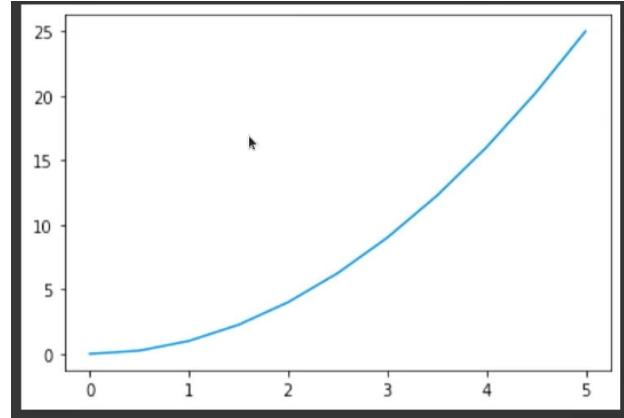
Librerías base para trabajar

```
1 x = np.linspace(0,5,11)  
2 y = x**2
```

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])  
array([ 0. , 0.25, 1. , 2.25, 4. , 6.25, 9. , 12.25, 16. ,  
       20.25, 25. ])
```

Para realizar una visualización

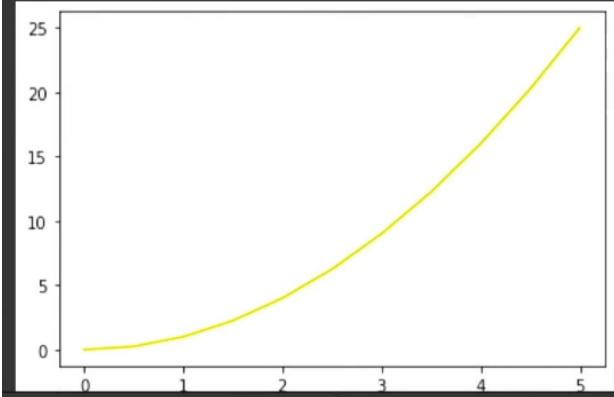
```
1 plt.plot(x,y)  
2 plt.show()
```



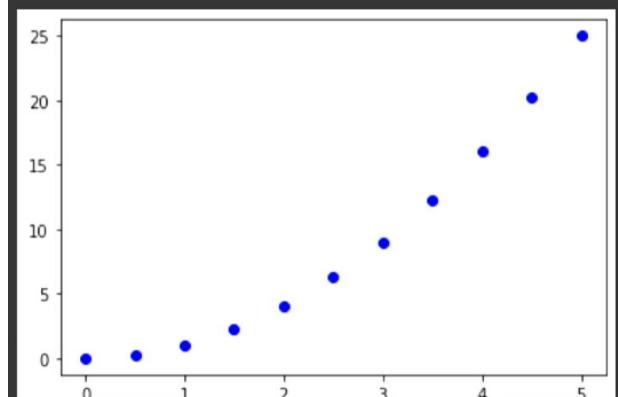
Podemos definir el color de manera sencilla

Además del color, indicar los 'markets' o marcadores

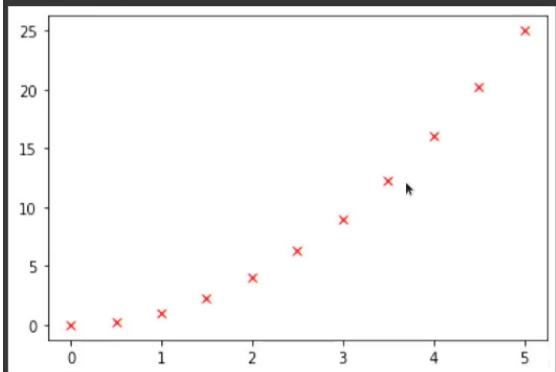
```
1 plt.plot(x,y, 'y')  
2 plt.show()
```



```
1 plt.plot(x,y, 'bo')  
2 plt.show()
```

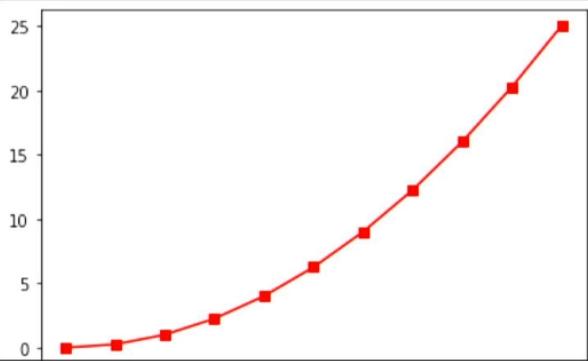


```
1 plt.plot(x,y, 'rx')  
2 plt.show()
```

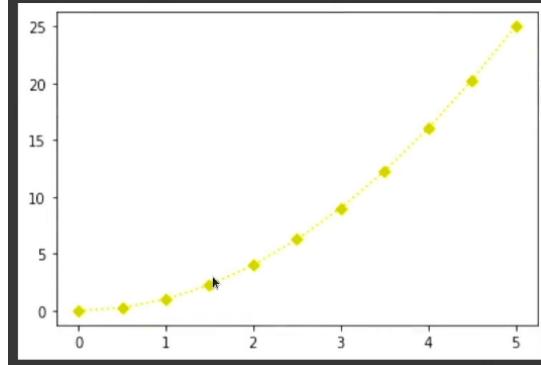


Otro ejemplo con color rojo 'r' y con marcadores 'x'

```
1 plt.plot(x,y,'rs-')
2 plt.show()
```

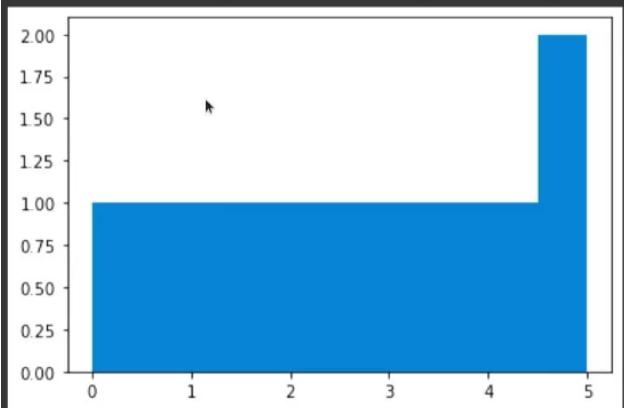


```
1 plt.plot(x,y,'yD-')
2 plt.show()
```

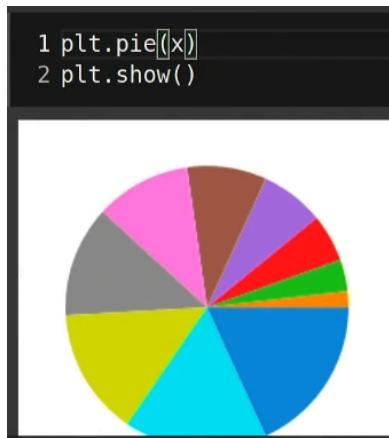


Podemos realizar histogramas que necesitan de una sola variable

```
1 plt.hist([x])
2 plt.show()
```

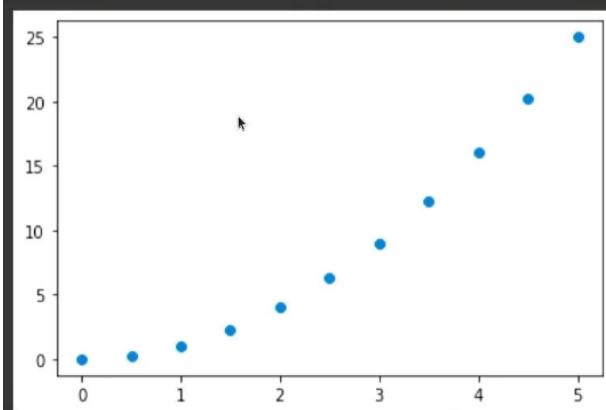


O gráficas de torta o 'pie'



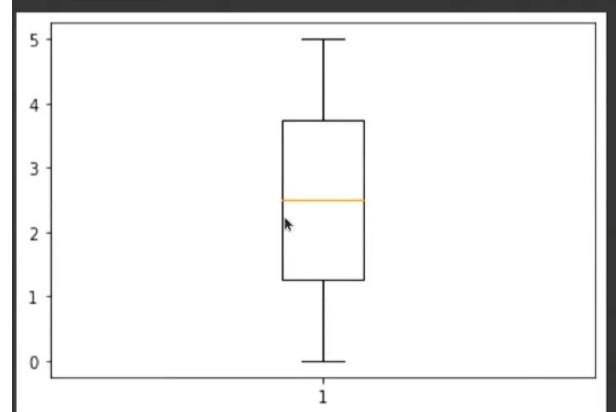
O ver también una relación de unos valores respecto a los otros

```
1 plt.scatter(x,y)
2 plt.show()
```



O una distribución de los datos

```
1 plt.boxplot(x)
2 plt.show()
```



Subplot

Obtener más de un gráfico o visualización

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

```
x = np.linspace(0,5,11)  
y = x**2
```

Trabaja con tres parámetros: Núm de filas , Núm de columnas , Índice de selección

```
plt.subplot(1,2,1)  
plt.plot(x,y)  
plt.subplot(1,2,2)  
plt.hist(y)  
plt.show()
```

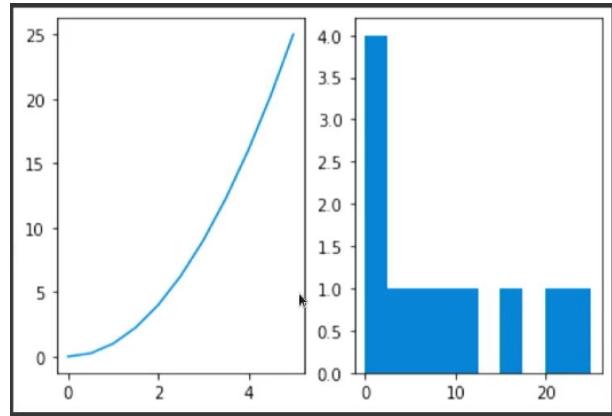
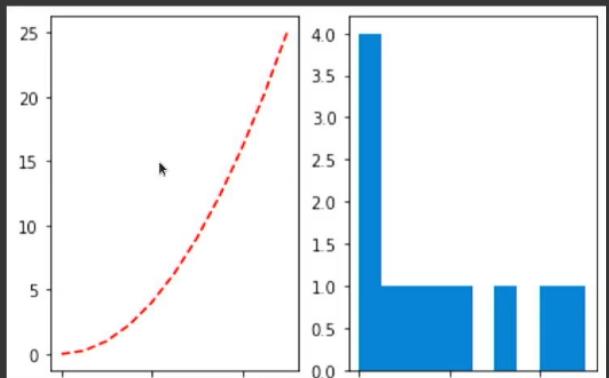
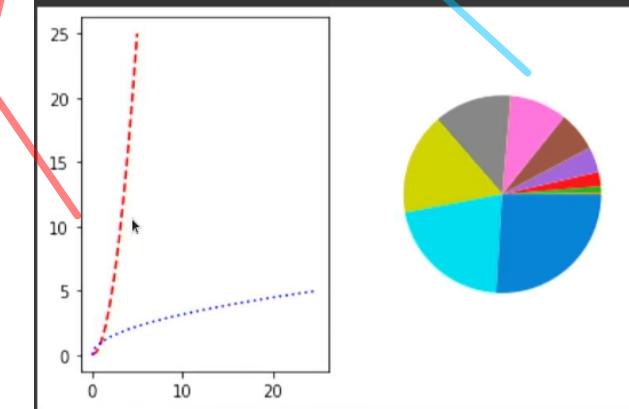


Gráfico con una fila y dos columnas

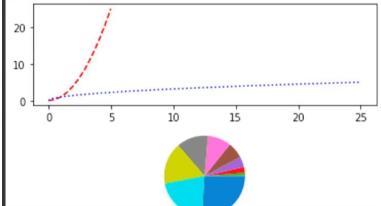
```
1 plt.subplot(1,2,1)  
2 plt.plot([x,y,'r--'])  
3 plt.subplot(1,2,2)  
4 plt.hist(y)  
5 plt.show()  
6
```



```
1 plt.subplot(1,2,1)  
2 plt.plot(x,y,'r--')  
3 plt.plot(y,x,'b:')  
4 plt.subplot(1,2,2)  
5 plt.pie(y)  
6 plt.show()
```



```
1 plt.subplot(2,1,1)  
2 plt.plot(x,y,'r--')  
3 plt.plot(y,x,'b:')  
4 plt.subplot(2,1,2)  
5 plt.pie(y)  
6 plt.show()
```



Método orientado a objetos

¿Para qué sirven?

Pyplot

- Rápido
- Fácil
- Una sola figura

Object Oriented

- Mayor personalización.
- Más amigable a múltiples diagramas.
- Más código.

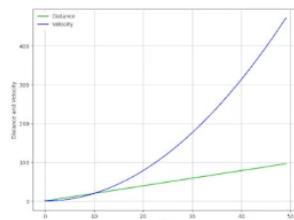
```
1 import matplotlib.pyplot as plt  
2 import numpy as np
```

```
1 x = np.linspace(0,5,11)  
2 y = x**2
```

Ejemplo usando Pyplot

Pyplot

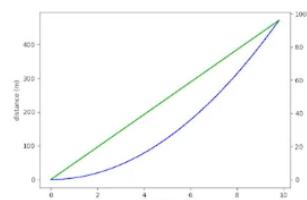
```
plt.figure(figsize=(9,7),  
dpi=100)  
plt.plot(time, velocity,'g-')  
plt.plot(time, distance,'b-')  
plt.ylabel("Distance and  
Velocity")  
plt.xlabel("Time")  
plt.grid(True)
```



Ejemplo usando orientado a objetos

Object oriented

```
fig, ax1 = plt.subplots()  
ax1.set_ylabel("distance (m)")  
ax1.set_xlabel("time")  
ax1.plot(time, distance, "blue")  
ax2 = ax1.twinx()  
ax2.set_ylabel("velocity (m/s)")  
ax2.set_xlabel("time")  
ax2.plot(time, velocity, "green")  
fig.set_size_inches(7.5)  
fig.set_dpi(100)
```



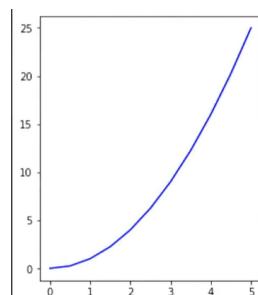
Definimos la posición del axis en el eje x y en la posición en el eje y
Además de el largo del eje x y el largo del eje y

```
fig = plt.figure()  
axes = fig.add_axes([0.1,0.1,0.5,0.9])  
axes.plot(x,y,'b')  
fig.show()
```

Creamos un lienzo (o figura) donde quedara nuestras gráficas

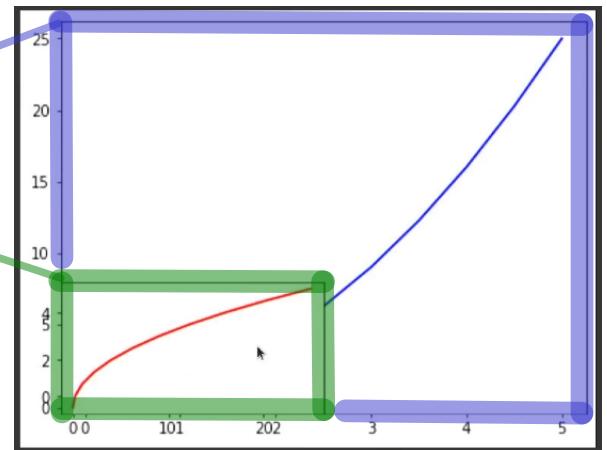
Las gráficas que pintemos serán llamadas axes

Gráficamos como lo habíamos hecho previamente

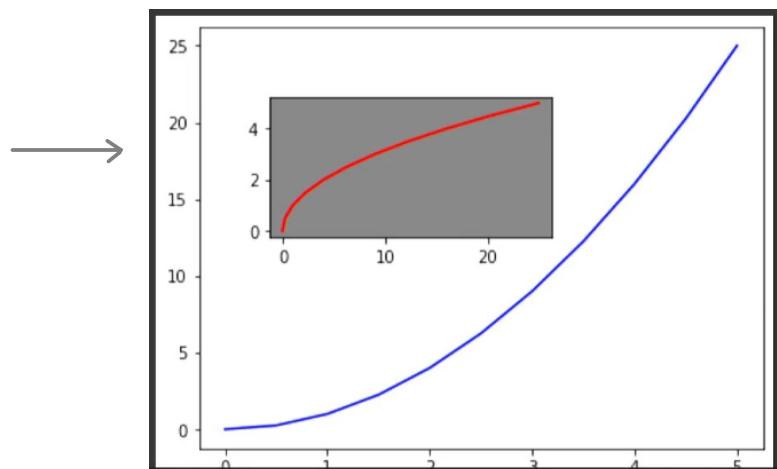


En esta ocasión definimos dos axes

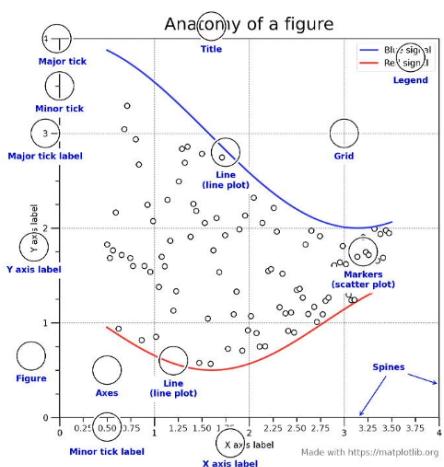
```
fig = plt.figure()  
axes = fig.add_axes([0.1,0.1,0.8,0.9])  
axes2 = fig.add_axes([0.1,0.1,0.4,0.3])  
  
axes.plot(x,y, 'b')  
axes2.plot(y,x, 'r')  
  
fig.show()
```



```
fig = plt.figure()  
axes = fig.add_axes([0.1,0.1,0.8,0.9])  
axes2 = fig.add_axes([0.2,0.55,0.4,0.3])  
  
axes.plot(x,y, 'b')  
axes2.plot(y,x, 'r')  
axes2.set_facecolor(['gray'])  
fig.show()
```

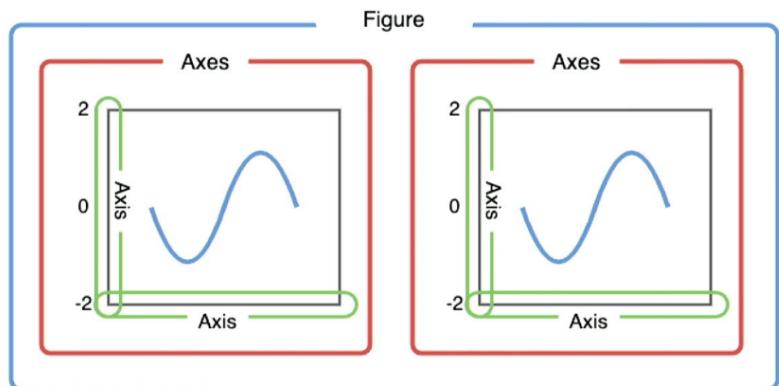


Personalizable



Los diferentes parámetros que podemos realizar a los objetos que definimos en la figura (o lienzo).

Estructura



Un objeto define una figura (o lienzo), en el cual puedo introducir gráficas, a cada una de estas gráficas las voy a llamar axes, y estos axes pueden a su vez tener diferentes ejes (x o y, o tridimensional).

Subplots

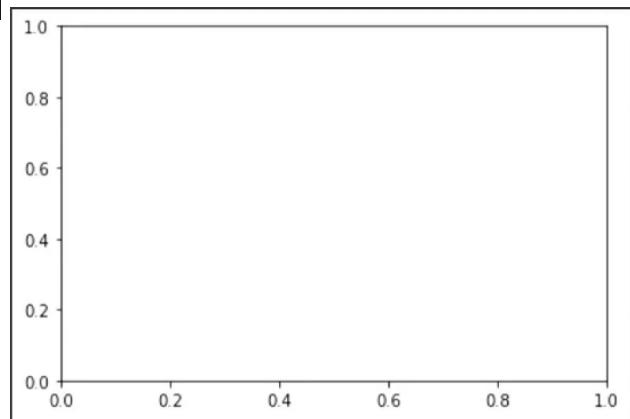
Anteriormente trabajamos fue con Subplot, sin 's' al final

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

```
x = np.linspace(0,5,11)  
y = np.sin(x)
```

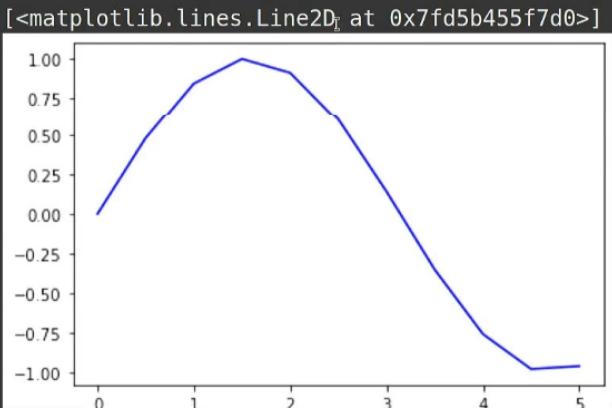
Un lienzo con un gráfico adentro

```
fig, axes = plt.subplots()
```



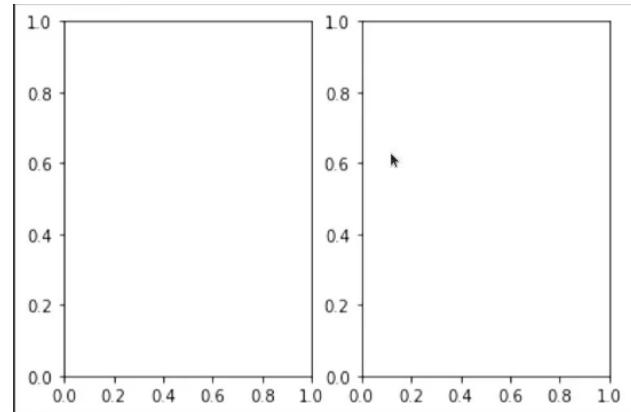
Realizando dibujos sobre la gráfica

```
1 fig, axes = plt.subplots()  
2 axes.plot(x,y, 'b')
```



Un lienzo con 2 gráficos al interior

```
fig, axes = plt.subplots(nrows=1, ncols=2)
```



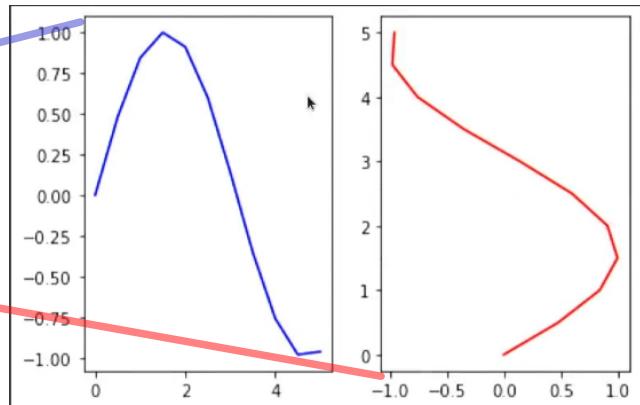
ndarray: axes
ndarray with shape (2,)

Un array al que se puede ingresar

```
fig, axes = plt.subplots(nrows=1, ncols=2)  
axes[0].plot(x,y, 'b')  
axes[1].plot(y,x, 'r')
```

Ingresamos a los axes como un array
y especificamos la gráfica que deseamos

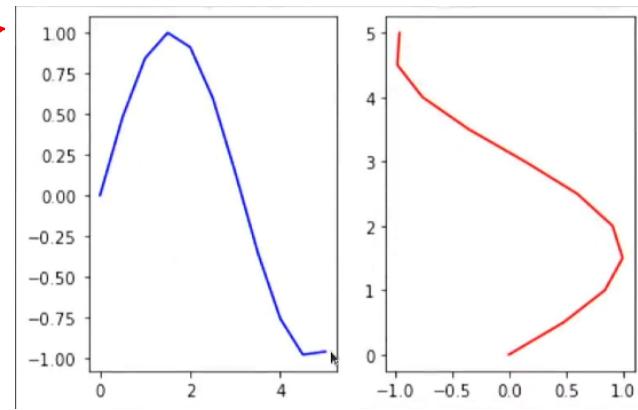
Objetos independientes a los que
accedemos por medio de un índice



Ya sabiendo que el objeto que devuelve es un array, podemos configurar previamente el nombres del axes para trabajar a conveniencia

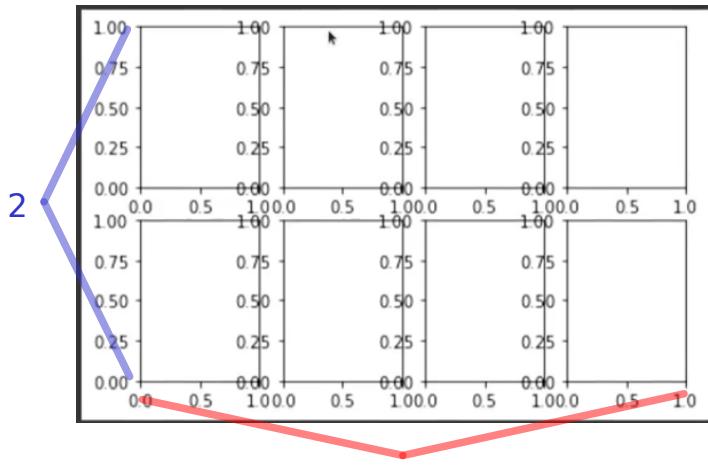
```
fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2)
ax1.plot(x,y, 'b')
ax2.plot(y,x, 'r')
```

Definimos una tupla que será donde se almacene los axes



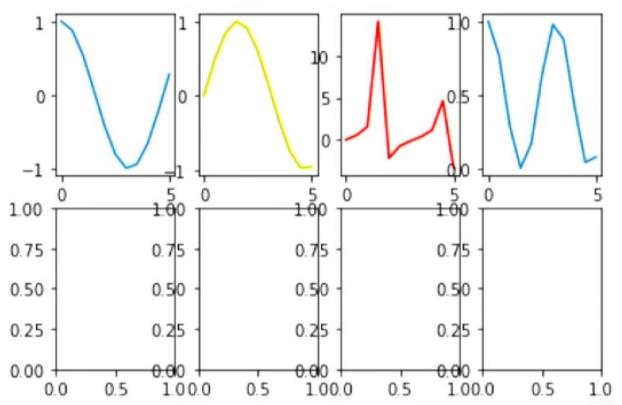
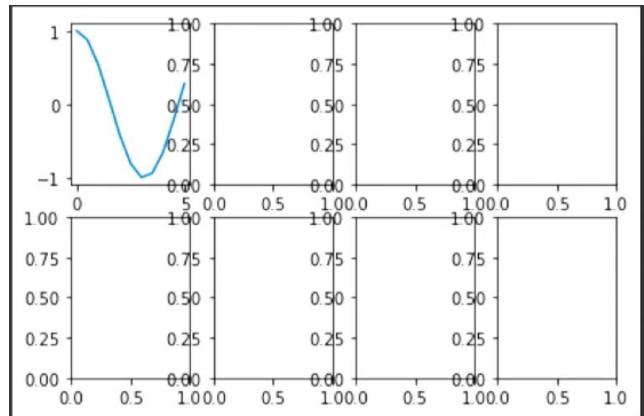
Podemos generar figuras de forma matricial

```
fig, axes = plt.subplots(2, 4)
```



```
fig, axes = plt.subplots(2, 4)
axes[0,0].plot(x,np.cos(x))
axes[0,1].plot(x,np.sin(x), 'y')
axes[0,2].plot(x,np.tan(x), 'r')
axes[0,3].plot(x,np.cos(x)**2)
```

```
fig, axes = plt.subplots(2, 4)
axes[0,0].plot(x,np.cos(x))
```

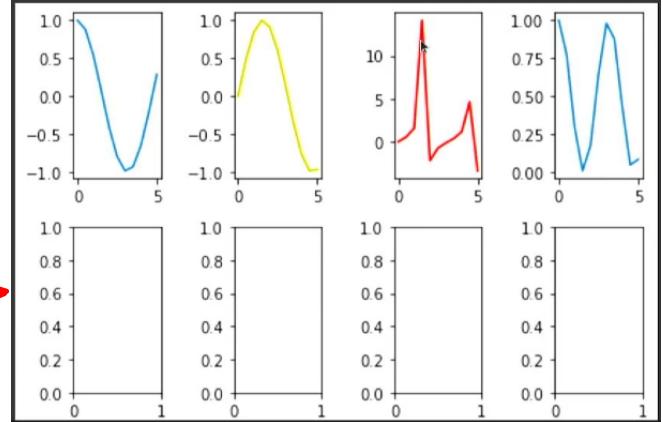


```
fig, ((ax1,ax2,ax3,ax4),(ax5,ax6,ax7,ax8)) = plt.subplots(2, 4)
ax1.plot(x,np.cos(x))
ax2.plot(x,np.sin(x), 'y')
ax3.plot(x,np.tan(x), 'r')
ax4.plot(x,np.cos(x)**2)
```

En esta ocasión definimos previamente los nombres de los axes

```
fig, ((ax1,ax2,ax3,ax4),(ax5,ax6,ax7,ax8)) = plt.subplots(2, 4)
ax1.plot(x,np.cos(x))
ax2.plot(x,np.sin(x), 'y')
ax3.plot(x,np.tan(x), 'r')
ax4.plot(x,np.cos(x)**2)
fig.tight_layout()
```

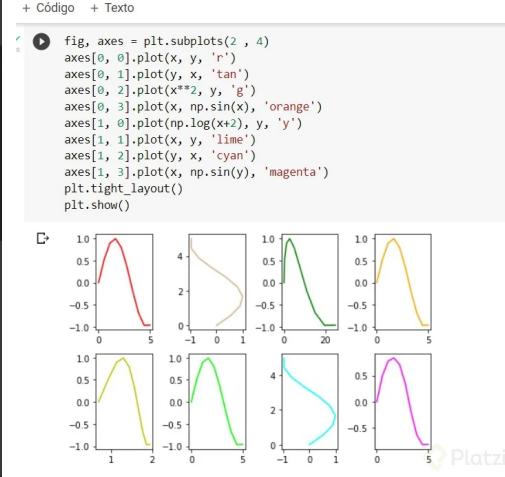
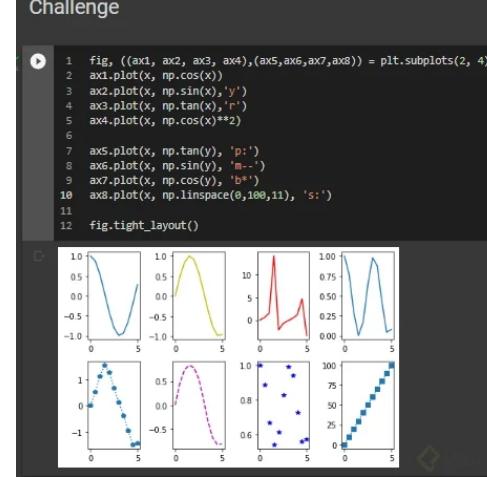
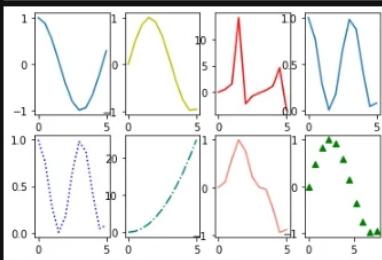
Evitamos que se sobrepongan las etiquetas



```
[72]: fig, axes = plt.subplots(2, 4)
axes[0,0].plot(x,np.cos(x))
axes[0,1].plot(x,np.sin(x), 'y')
axes[0,2].plot(x,np.tan(x), 'r')
axes[0,3].plot(x,np.cos(x)**2)

axes[1,0].plot(x,np.cos(x)**2,'b', linestyle=":")
axes[1,1].plot(x*x**2,'teal', linestyle=".-")
axes[1,2].plot(x,y***3,'salmon')
axes[1,3].plot(x**1.5,y,'g');


```



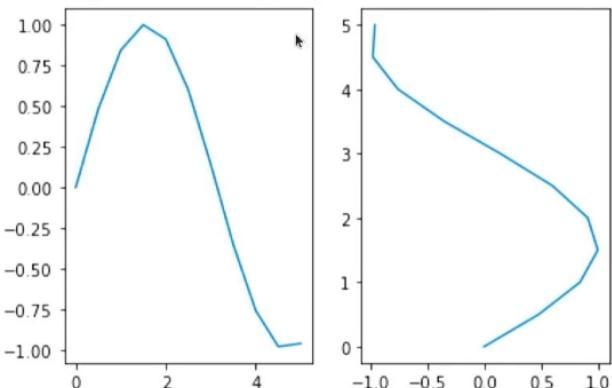
Leyendas, etiquetas, títulos, tamaño

Darle contexto y agregar valor a nuestros gráficos, adicionando títulos, leyendas, labels, modificar tamaños entre otros

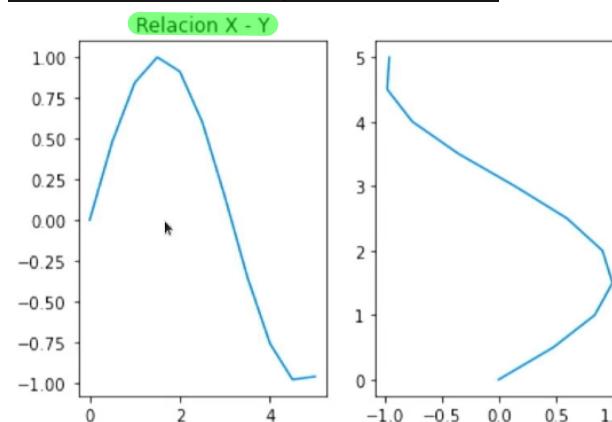
```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.linspace(0,5,11)
y = np.sin(x)
```

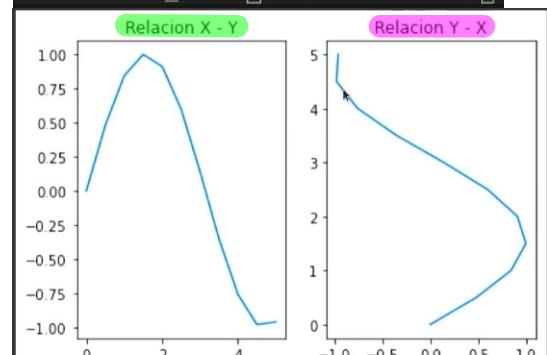
```
fig, axes = plt.subplots(1,2)
axes[0].plot(x,y)
axes[1].plot(y,x)
```



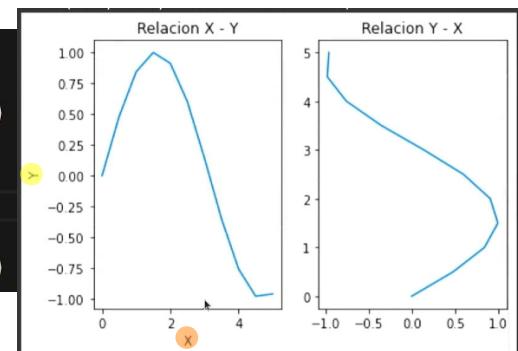
```
fig, axes = plt.subplots(1,2)
axes[0].plot(x,y)
axes[0].set_title(['Relacion X - Y'])
axes[1].plot(y,x)
```



```
fig, axes = plt.subplots(1,2)
axes[0].plot(x,y)
axes[0].set_title('Relacion X - Y')
axes[1].plot(y,x)
axes[1].set_title(['Relacion Y - X'])
```



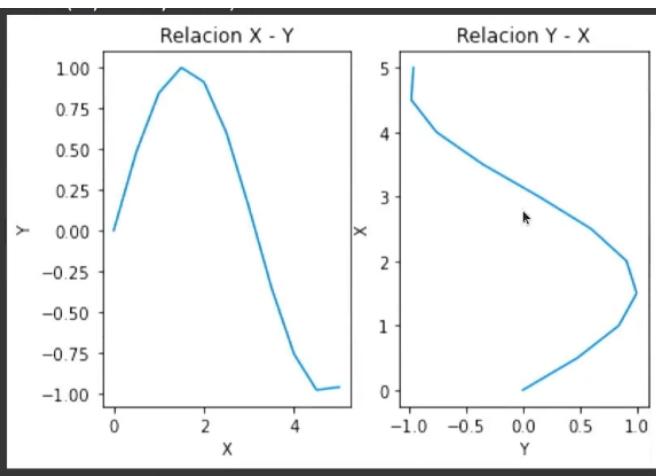
```
fig, axes = plt.subplots(1,2)
axes[0].plot(x,y)
axes[0].set_title('Relacion X - Y')
axes[0].set_xlabel('X')
axes[0].set_ylabel('Y')
axes[1].plot(y,x)
axes[1].set_title('Relacion Y - X')
```



```

fig, axes = plt.subplots(1,2)
axes[0].plot(x,y)
axes[0].set_title('Relacion X - Y')
axes[0].set_xlabel('X')
axes[0].set_ylabel('Y')
axes[1].plot(y,x)
axes[1].set_title('Relacion Y - X')
axes[1].set_xlabel('Y')
axes[1].set_ylabel(['X'])

```



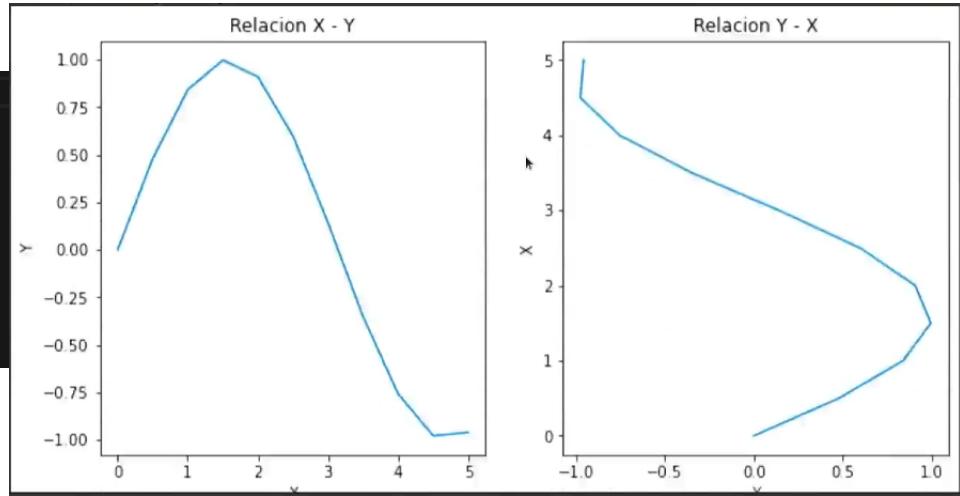
Un ejemplo con todas las labels en las dos gráficas

```

fig, axes = plt.subplots(1,2, figsize=[10,5])
axes[0].plot(x,y)
axes[0].set_title('Relacion X - Y')
axes[0].set_xlabel('X')
axes[0].set_ylabel('Y')
axes[1].plot(y,x)
axes[1].set_title('Relacion Y - X')
axes[1].set_xlabel('Y')
axes[1].set_ylabel('X')

```

Podemos modificar el tamaño del lienzo (o figura)



```

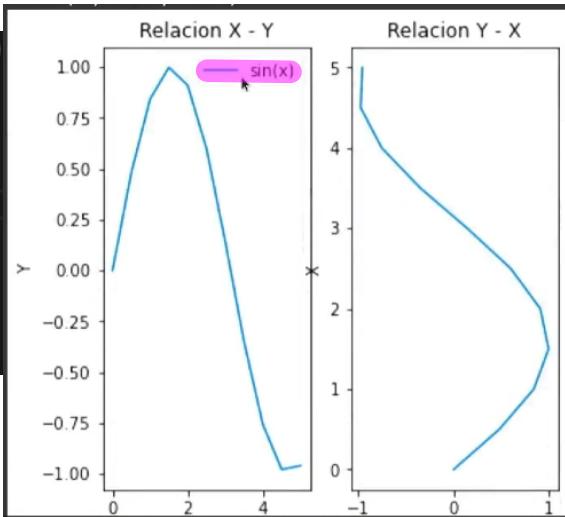
fig, axes = plt.subplots(1,2, figsize=(5,5))
axes[0].plot(x,y, label="sin(x)")
axes[0].set_title('Relacion X - Y')
axes[0].set_xlabel('X')
axes[0].set_ylabel('Y')
axes[0].legend()

axes[1].plot(y,x)
axes[1].set_title('Relacion Y - X')
axes[1].set_xlabel('Y')
axes[1].set_ylabel('X')

```

Con '\$' se introduce la leyenda en una notación matemática (LaTeX!?)

```
axes[0].plot(x,y, label="$\sin(x)$")
```



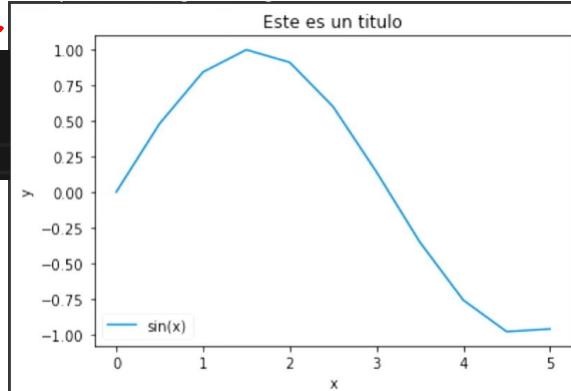
Podemos agregar leyendas a las gráficas, y además debemos indicar que las mostraremos

```

plt.plot(x,y,label='sin(x)')
plt.title('Este es un titulo')
plt.xlabel('x')
plt.ylabel(['y'])
plt.legend(loc='lower left')

```

Definimos los distintos parámetros de las leyendas



También podemos aplicar estos valores al plt.suptitle, (sin trabajar la orientación a objetos)

```
plt.legend(bbox_to_anchor=[0.5, 0.2])
```

Podemos definir específicamente la posición de la leyenda

Colores y estilos

Como customizar de mejor manera las gráficas a nivel de colores y estilos

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

```
print(plt.style.available)
```

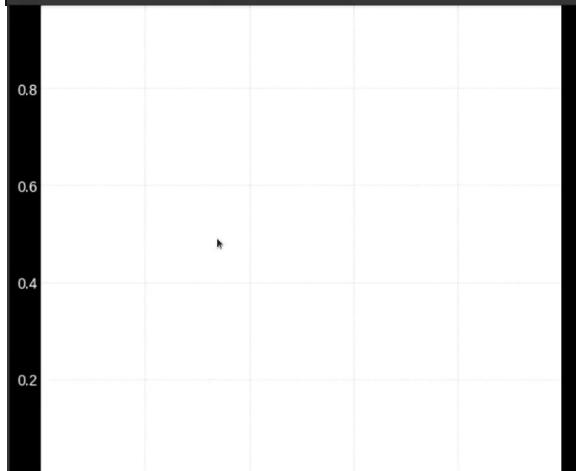
→ ['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', ...]

Ver cuales son los estilos con los que se puede trabajar por default, estilos predefinidos con ciertas gamas de colores, cierres grillas y otras opciones

Escogemos un estilo de los que predeterminados (podemos definirlo en una celda aparte)

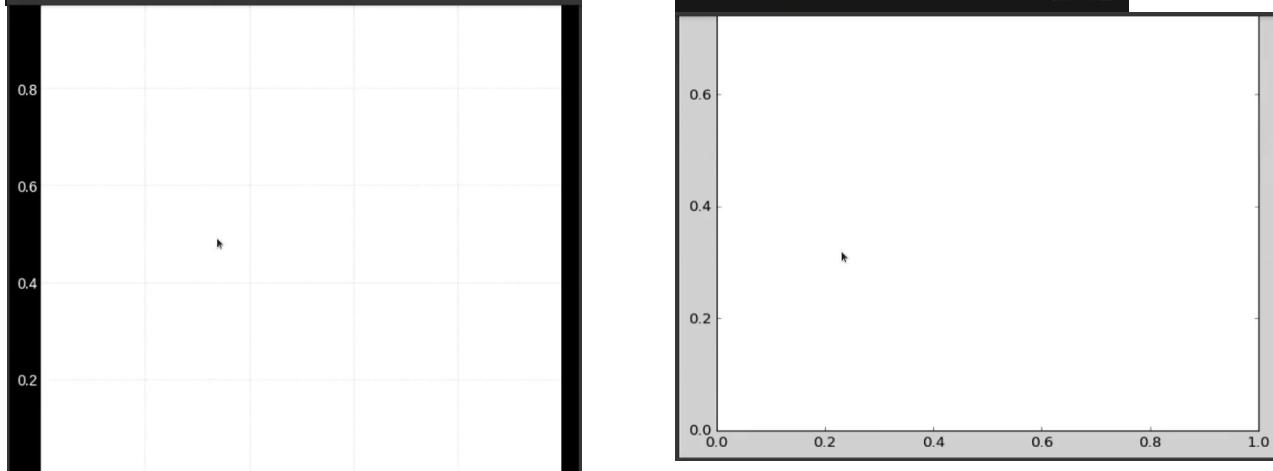
```
plt.style.use('bmh')
```

```
fig, ax = plt.subplots(figsize=(8,8))
```



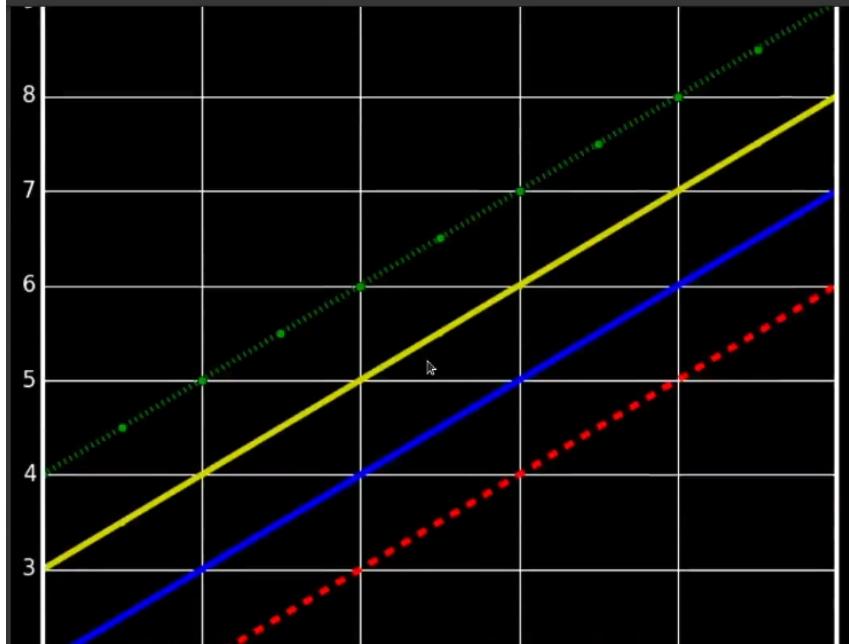
```
plt.style.use('classic')
```

```
fig, ax = plt.subplots(figsize=(8,8))
```



→

```
plt.style.use('dark_background')  
fig, ax = plt.subplots(figsize=(8,8))  
  
ax.plot(x,x+1,'r--')  
ax.plot(x,x+2,'b-')  
ax.plot(x,x+3,'y.-')  
ax.plot([x,x+4], 'go:')
```



Un ejemplo de un estilo con diferentes gráficas

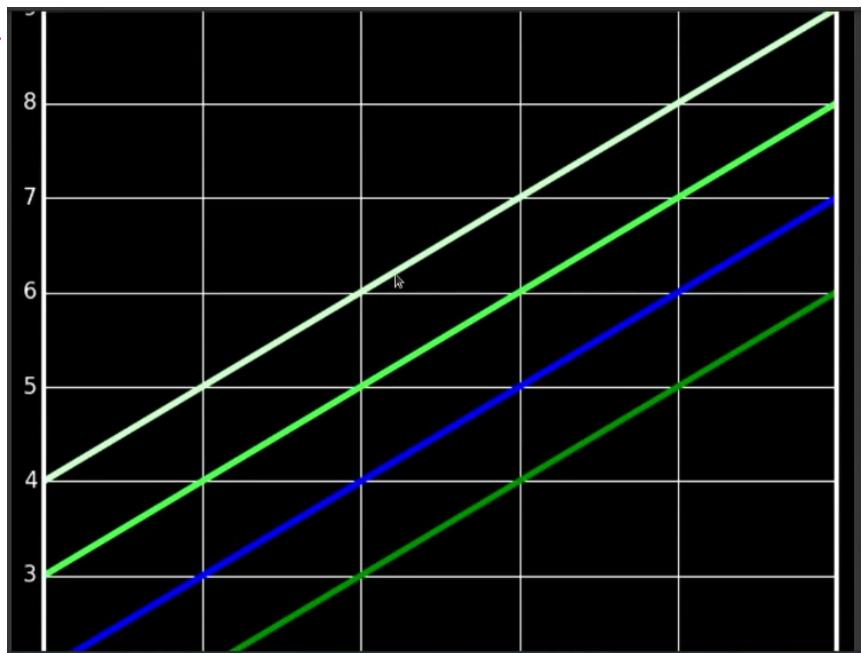
```
fig, ax = plt.subplots(figsize=(8,8))

ax.plot(x,x+1,color = 'green')
ax.plot(x,x+2,color = 'blue')
ax.plot(x,x+3,color = '#66FF66')
ax.plot(x,x+4,color = '#CCFFCC')
```

Podemos definir colores usando la notación **RGB**

```
ax.plot(x,x+1,color = 'green', alpha=0.5)
```

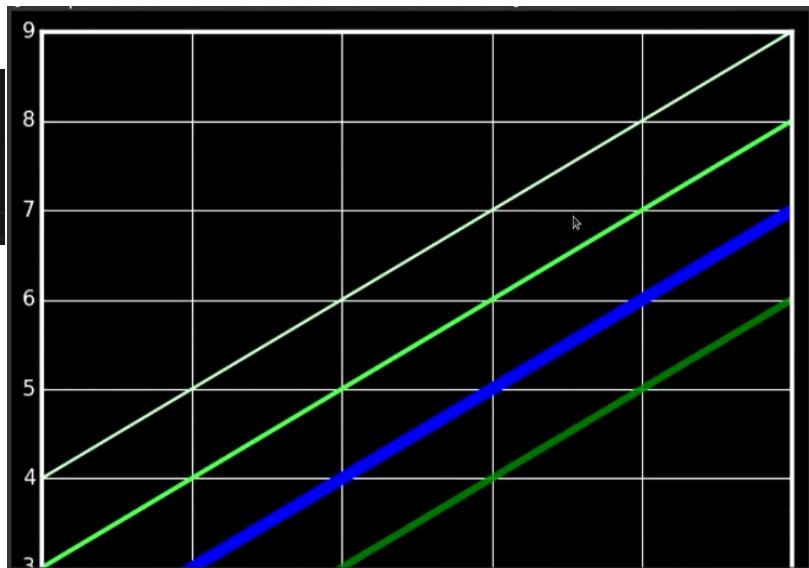
Definimos las transparencia



```
fig, ax = plt.subplots(figsize=(8,8))

ax.plot(x,x+1,color = 'green', alpha=0.8, linewidth=5)
ax.plot(x,x+2,color = 'blue', linewidth=8)
ax.plot(x,x+3,color = '#66FF66', linewidth=3)
ax.plot(x,x+4,color = '#CCFFCC', linewidth=2)
```

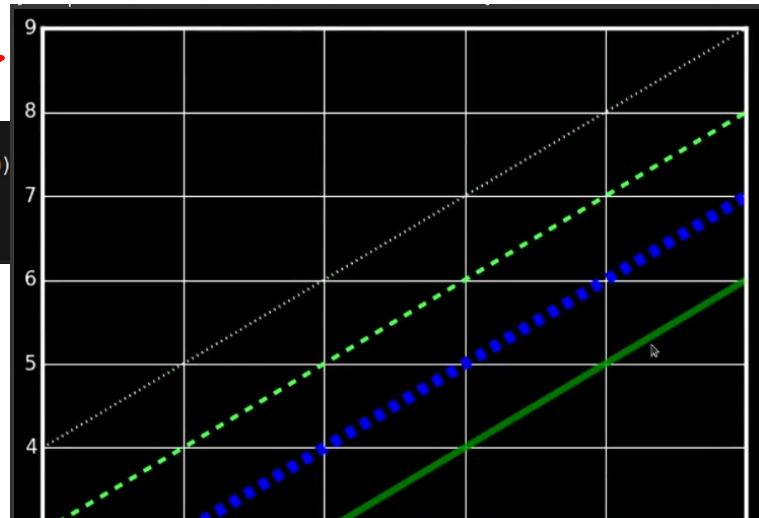
Podemos definir el grosor de las líneas



```
fig, ax = plt.subplots(figsize=(8,8))
```

```
ax.plot(x,x+1,color = 'green', alpha=0.8, linewidth=5, linestyle='--')
ax.plot(x,x+2,color = 'blue', linewidth=8, linestyle='--')
ax.plot(x,x+3,color = '#66FF66', linewidth=3, linestyle='dashed')
ax.plot(x,x+4,color = '#CCFFCC', linewidth=2, linestyle=':')
```

Otra manera de especificar el **estilo de las líneas**



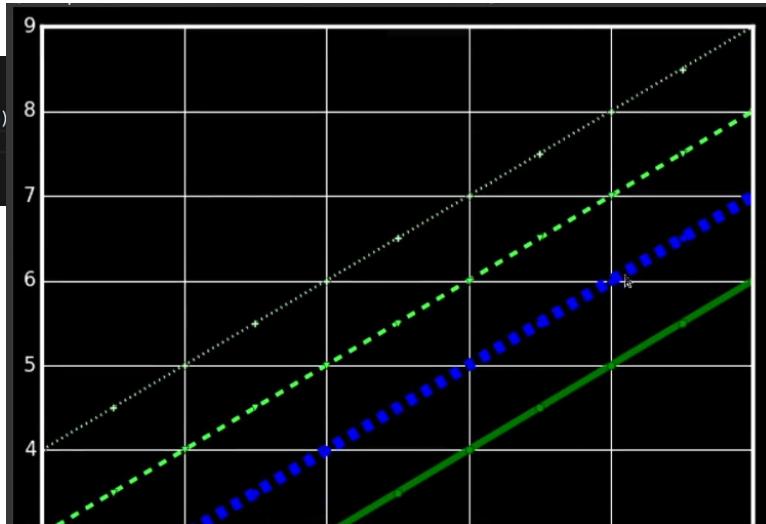
```

fig, ax = plt.subplots(figsize=(8,8))

ax.plot(x,x+1,color = 'green', alpha=0.8, linewidth=5, linestyle='--',marker='o')
ax.plot(x,x+2,color = 'blue', linewidth=8 , linestyle='--',marker='8')
ax.plot(x,x+3,color = '#66FF66', linewidth=3 ,linestyle='dashed',marker='v')
ax.plot(x,x+4,color = '#CCFFCC', linewidth=2 ,linestyle=':',marker='P')

```

Agregarle marcadores a las líneas



```

ax.plot(x,x+1,color = 'green', alpha=0.8, linewidth=5, linestyle='--',marker='o'
,markersize=10, markerfacecolor="#CCFFCC")

```

También podemos definir el tamaño y color de los marcadores o 'marker'

```

for i in plt.style.available:
    plt.style.use(i)
    fig, ax = plt.subplots(figsize=(5,5))
    ax.plot(x, y)

```

Código para ver todos los estilos disponibles

```

for x in plt.style.available:
    plt.style.use(x)
    fig, ax =
    plt.subplots(figsize=(2,2))
    ax.set_title(x, size=20)

```

Bar Plot

Las gráficas de barras nos permiten graficar variables categoricas

```

import matplotlib.pyplot as plt
import numpy as np

```

```

country = ['INDIA', 'JAPAN', 'MEXICO', 'COLOMBIA', 'GERMANY']
population = [1000,800,900,1000,300]

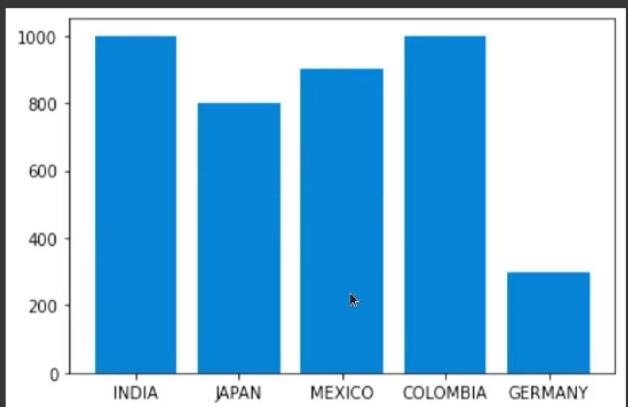
```

En el eje x se manejan variables categoricas

```

plt.bar(country,population)
plt.show()

```

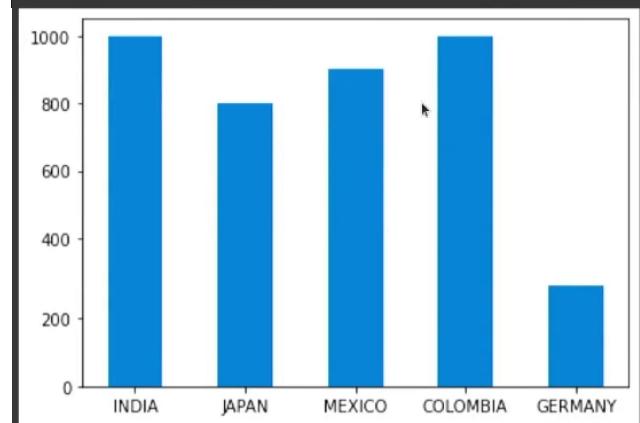


```

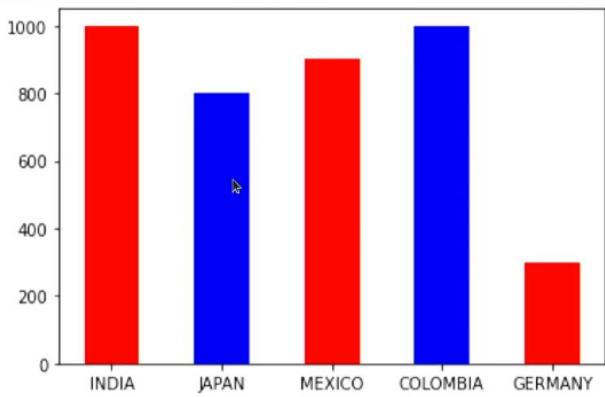
plt.bar(country,population, width=0.5)
plt.show()

```

La longitud de la base de las barras

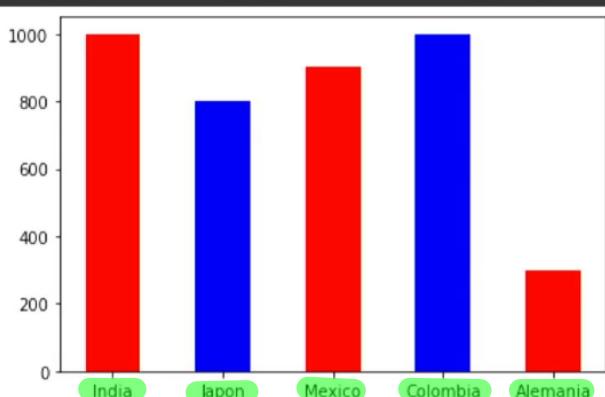


```
plt.bar(country,population,width=0.5,color=['red','blue'])
plt.show()
```



Podemos definir los colores de las barras, e inclusive indicar varios colores para que estos sean alternados entre las barras

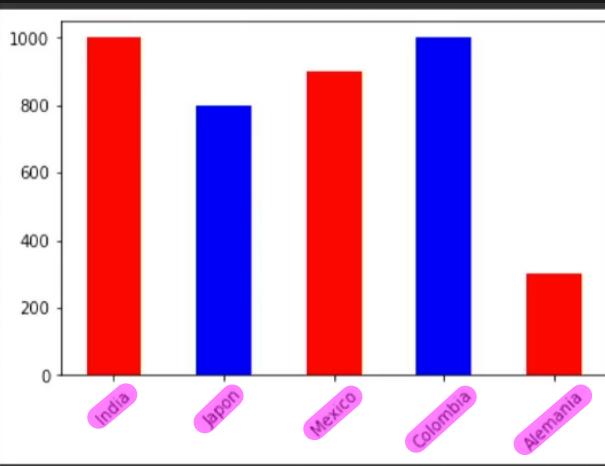
```
plt.bar(country,population,width=0.5,color=['red','blue'])
plt.xticks(np.arange(5),['India','Japon','Mexico','Colombia','Alemania'])
plt.show()
```



Creamos un arreglo y posteriormente indicamos los valores que deben ir en el arreglo (array)

`plt.xticks()` Configuraciones del eje x

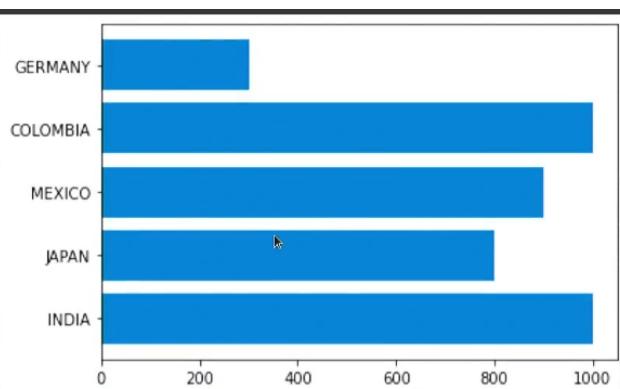
```
plt.bar(country,population,width=0.5,color=['red','blue'])
plt.xticks([np.arange(5),('India','Japon','Mexico','Colombia','Alemania')],rotation=45)
plt.show()
```



Para cuando los valores se encuentran muy amontonados, podemos especificar una rotación en las etiquetas del eje x

```
plt.bart(country,population)
plt.show()
```

Podemos generar un gráfico de barras de manera horizontal



Crear otro tipo de gráficas

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

Existen otros tipos de gráfico que matplotlib nos proporciona

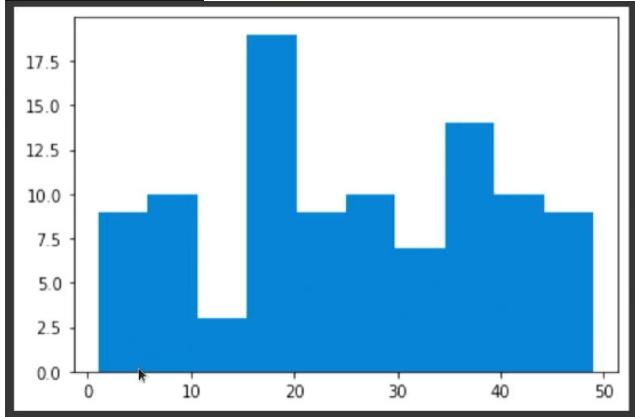
```
data = np.random.randint(1,50,100)
```

```
array([23, 26, 18, 28, 35, 11, 34, 16, 17, 48, 34, 5, 19, 15, 2, 29, 30,  
       44, 37, 10, 9, 27, 45, 18, 17, 8, 47, 17, 25, 7, 16, 25, 20, 36,  
       17, 19, 26, 48, 41, 1, 11, 5, 21, 9, 35, 35, 49, 44, 24, 44, 40,  
       4, 30, 35, 27, 35, 42, 20, 42, 38, 40, 2, 10, 26, 47, 24, 22, 38,  
       30, 18, 47, 36, 23, 6, 10, 48, 25, 4, 6, 35, 16, 30, 23, 37, 16,  
       39, 8, 33, 21, 1, 2, 19, 19, 46, 44, 24, 39, 19, 18, 44])
```

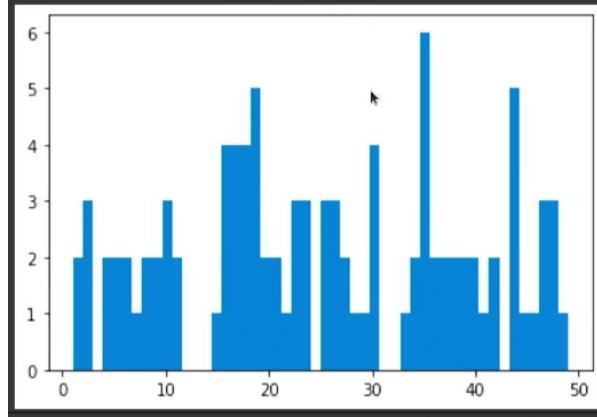
Histograma

Con el **histograma** podemos ver la frecuencia de los datos, tenemos la opción de definir la cantidad de barras que estarán presentes en el histograma.

```
plt.hist(data)  
plt.show()
```

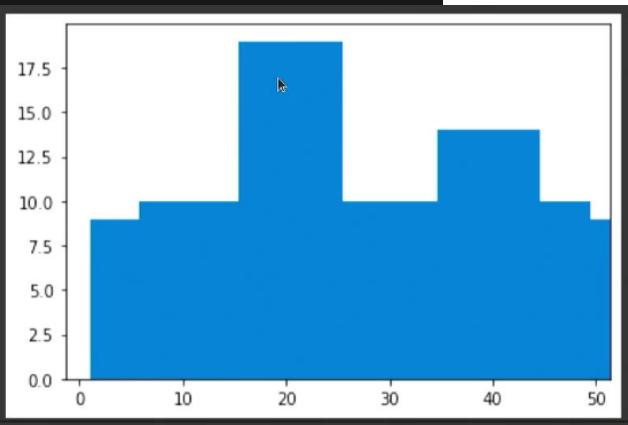


```
plt.hist(data,bins=50)  
plt.show()
```

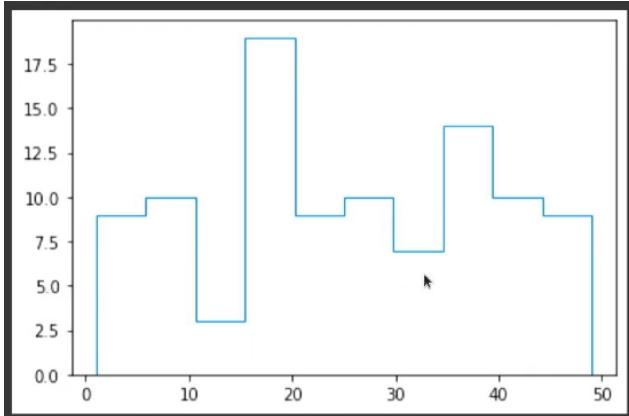


El valor por defecto para bins es 10 (bins=10)

```
plt.hist(data,bins=10,width=10)  
plt.show()
```



```
plt.hist(data,bins=10,histtype='step')  
plt.show()
```

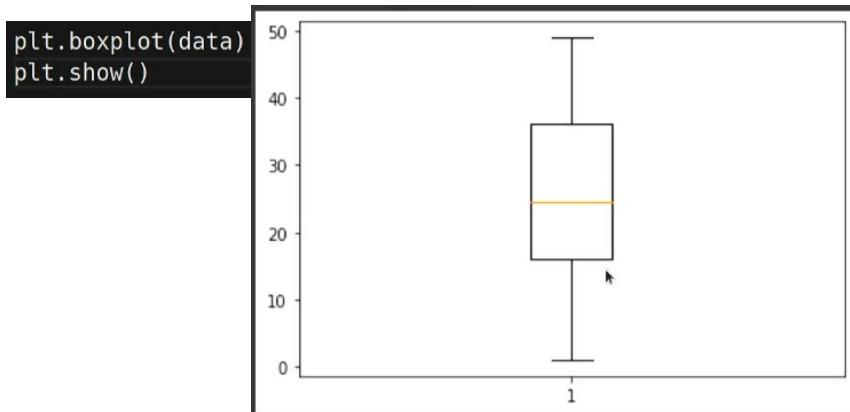


Podemos definir el **tamaño de las barras**

Podemos definir el **estilo de las barras**

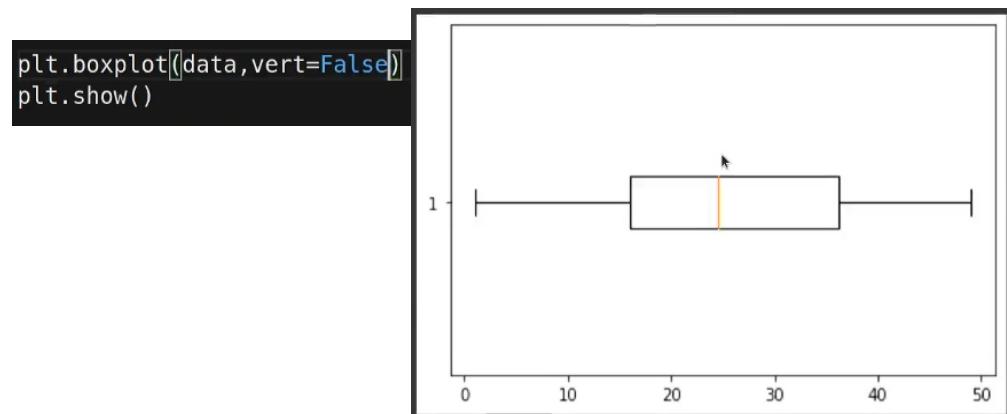
Boxplot

También conocida como gráfica de caja y bigotes

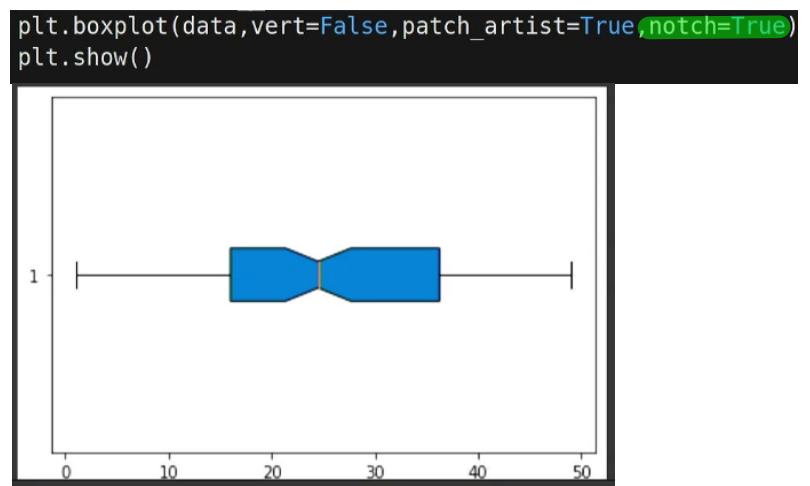
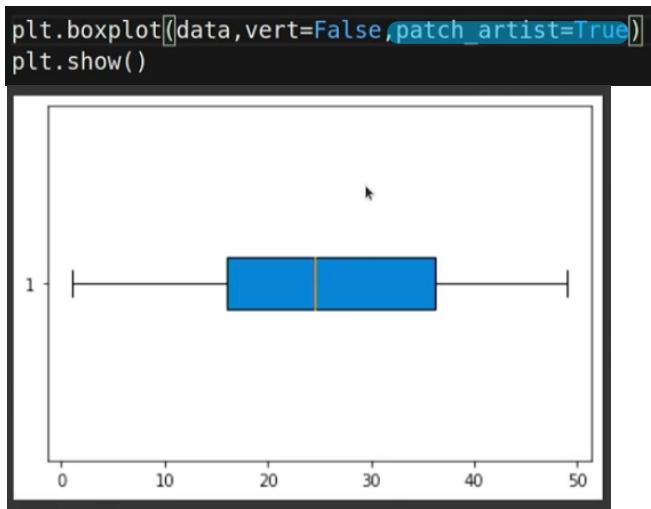


Nos enseña el rango intercuartil y donde se ubica el percentil 25, la mediana, y el percentil 75

Util para identificar la distribución de los datos

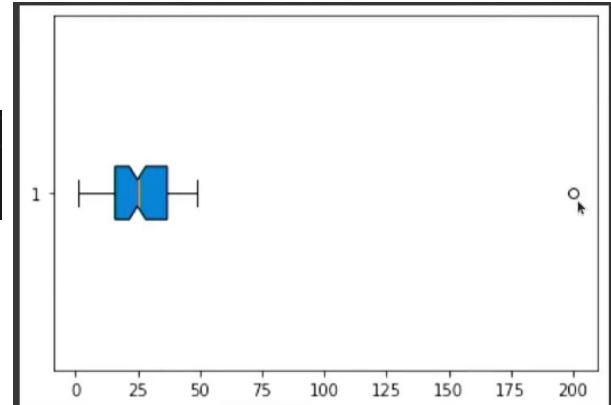


Podemos especificar que los no lo queremos de manera vertical, si no horizontal



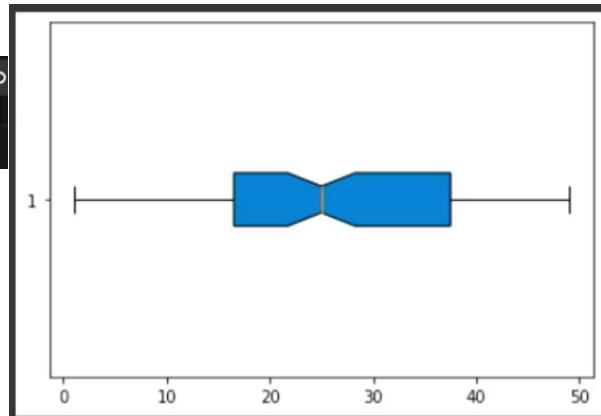
```
data = np.append([data,200])
plt.boxplot(data,vert=False,patch_artist=True,notch=True)
plt.show()
```

Agregamos un outlier para ver como se comporta la gráfica



Podemos escoger no mostras los outlayer en las gráfica

```
data = np.append(data,200)  
plt.boxplot([data,vert=False,patch_artist=True,notch=True,showfliers=False])  
plt.show()
```



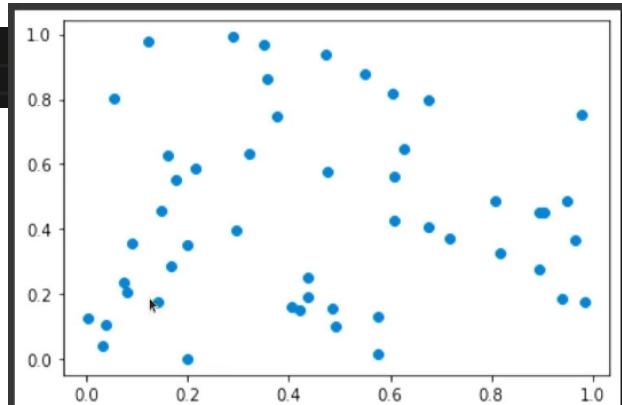
Scatter

Ideal para ver la relación entre dos variables distintas

```
N = 50  
x = np.random.rand(N)  
y = np.random.rand(N)  
area = (30 * np.random.rand(N)) **2  
colors = np.random.rand(N)
```

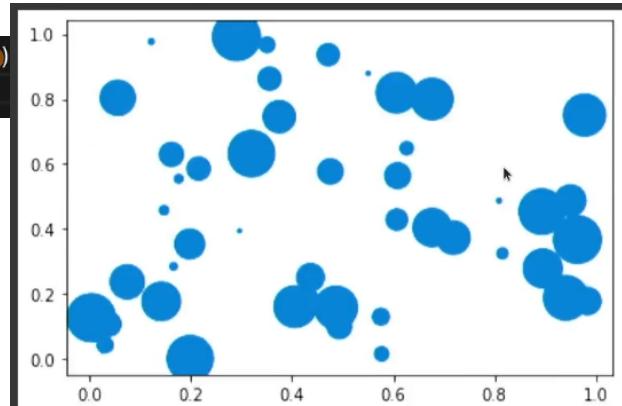
Ideal para verificar si hay correlación entre las variables

```
plt.scatter(x,y)  
plt.show()
```



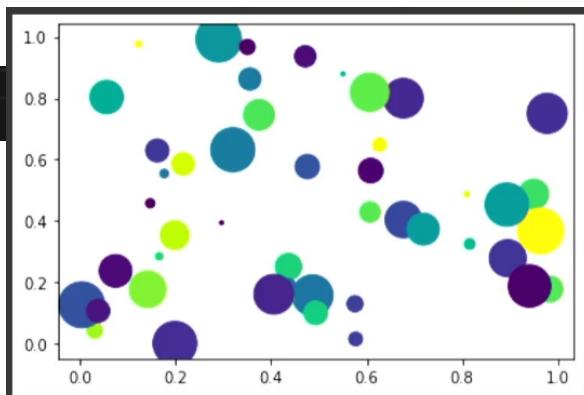
Gráficamos los datos aleatorios generados previamente

```
plt.scatter(x,y,s=area)  
plt.show()
```



Podemos asignar diferentes tamaños

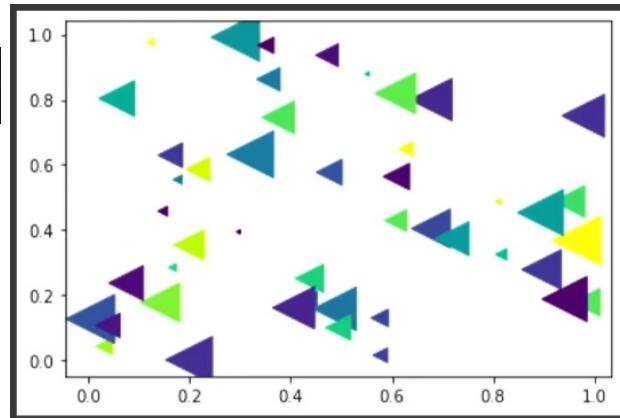
```
plt.scatter(x,y,s=area,c=colors)  
plt.show()
```



Agregar diferentes colores

```
plt.scatter([x,y,s=area,c=colors,marker='<'])  
plt.show()
```

También podemos modificar los marker



```
plt.scatter([x,y,s=area,c=colors,marker='o',alpha=0.5])  
plt.show()
```

Podemos definir una transparencia para ver de mejor manera las figuras sobre-puestas

