AWAKELAB

BASECAMP

Ciencia de Datos

Análisis Exploratorio y Programación Estadística

Objetivo de la jornada

- Presentar información de un set de datos utilizando librería Matplotlib para graficar la información.

Librerías Matplotlib.

matplotlib es la librería de visualización más popular de Python. Su primera versión, creada por John D. Hunter, se remonta al año 2003, y esto puede ser considerado a la vez tanto una fortaleza como una debilidad. Es cierto que, con el tiempo, matplotlib se ha instaurado como la librería de visualización de referencia y, de hecho, otras muchas librerías han desarrollado sus herramientas de visualización sobre matplotlib: *seaborn*, *ggplot*... incluso pandas basa en matplotlib sus herramientas de visualización (los DataFrames poseen el método **plot** importado de matplotlib que permite crear y personalizar con cierta facilidad una gráfica con los datos contenidos en el DataFrame). Pero también es cierto que, con el paso de los años, matplotlib ha sido superada por las librerías mencionadas (*seaborn*, *ggplot*) y por otras (*Bokeh*, *Plotly*) que ofrecen visualizaciones más atractivas o unas interfaces más coherentes y amigables. En cualquier caso, es esa dependencia que tienen muchas librerías con respecto a matplotlib lo que hace más que conveniente el conocimiento de matplotlib: las librerías de más alto nivel diseñadas sobre ésta no siempre ofrecen toda la funcionalidad que necesitamos y, cuando queremos afinar en el diseño de una visualización, frecuentemente no nos queda más remedio que acudir directamente a matplotlib.

Otra de las fortalezas de matplotlib es su compatibilidad con diferentes sistemas operativos y entornos en los que se ejecuta, lo que ayudó a su rápida adopción por parte de la comunidad.

De lo dicho se deduce que matplotlib es una librería de bajo nivel, muy potente y extensa pero que puede resultar un tanto confusa al principio. Ofrece herramientas para la creación de visualizaciones en 2D, aunque se completa con el uso de otros add-ons que permiten la generación de gráficas 3D (mplot3d) y mapas (basemap).

La mayoría de las funciones de matplotlib están localizadas en la sublibrería *pyplot*, que suele importarse con el alias **plt**:

```
import matplotlib.pyplot as plt
```

En algún caso es necesario trabajar directamente con la librería matplotlib, que suele importarse con el alias mpl:

```
import matplotlib as mpl
```

Lógicamente, para hacer uso de cualquier función ofrecida por la librería matplotlib.pyplot, ésta deberá ser importada usando la instrucción comentada.

Componente de un gráfico

Un gráfico, está formado por diferentes partes que incluyen el área del gráfico, las series de datos, ejes, leyendas, rótulos del eje, entre otros

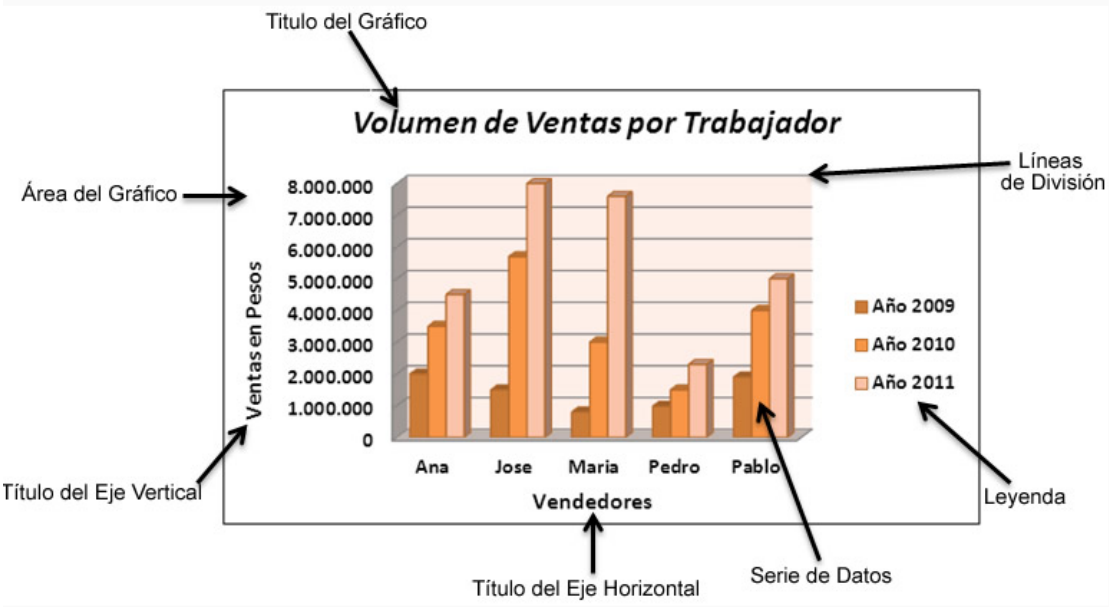

imagen:fuente propia

Área del gráfico: Esta es el área que se encuentra definida por el marco del gráfico y que incluye todas sus partes.

Título del gráfico: Texto descriptivo del gráfico que se coloca en la parte superior.

Series de datos: Son los puntos de datos relacionados entre sí trazados en un gráfico.

Cada serie de datos tiene un color exclusivo. Un gráfico puede tener una o más series de datos a excepción de los gráficos circulares que solamente pueden tener una serie de datos.

Ejes: Un eje es la línea que sirve como referencia de medida. El eje Y es conocido como el eje vertical y generalmente contiene datos. El eje X es

conocido también como el eje horizontal y suele contener las categorías del gráfico.

Líneas de división: Son líneas opcionales que extienden los valores de los ejes de manera que faciliten su lectura e interpretación.

Título de eje: Texto descriptivo que se alinea automáticamente al eje correspondiente.

Leyenda: Recuadro que ayuda a identificar los colores asignados a las series de datos.

Tipos de gráficos

Matplotlib, permite crear y personalizar los tipos de gráficos más comunes, entre ellos:

- Diagramas de dispersión o puntos
- Diagramas de barras
- Histograma
- Diagramas de caja y bigotes

y combinaciones de todos ellos.

En la siguiente galería de gráficos pueden apreciarse todos los tipos de gráficos que pueden crearse con esta librería.

Diagrama de dispersión o puntos.

scatter(x, y): Dibuja un diagrama de puntos con las coordenadas de la lista **x** en el eje X y las coordenadas de la lista **y** en el eje Y.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.scatter([1, 2, 3, 4], [1, 2, 0, 0.5])
plt.show()
```

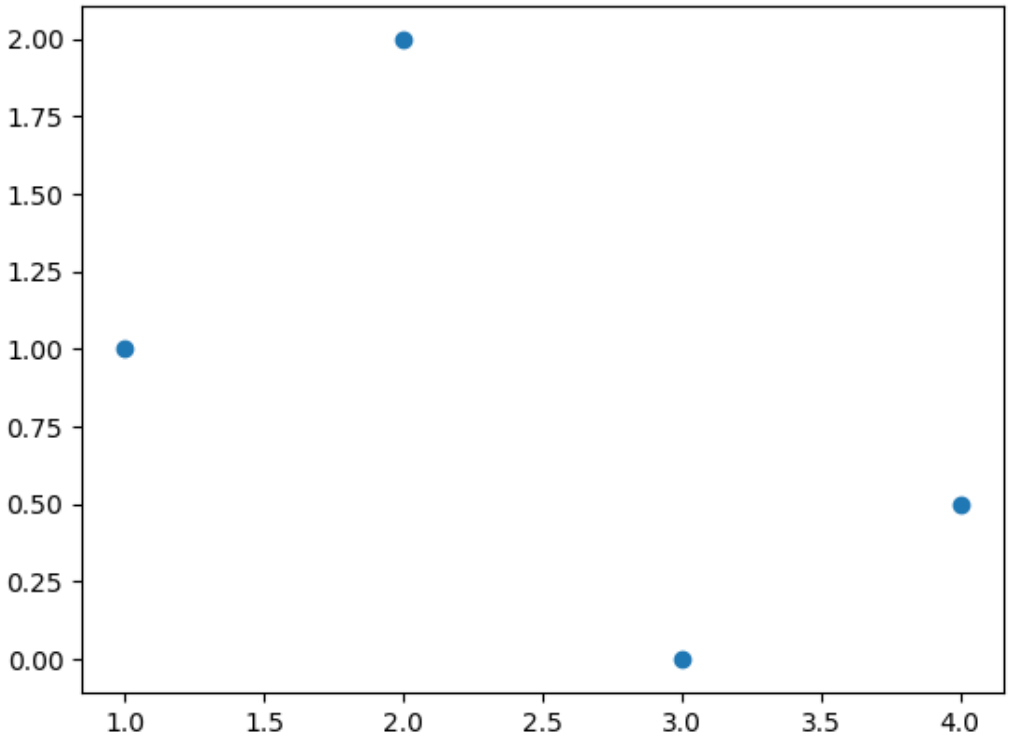

imagen:fuente propia

Diagrama de barras.

barh(x, y): Dibuja un diagrama de barras horizontales donde **x** es una lista con la posición de las barras en el eje Y, e **y** es una lista con la longitud de las barras en el eje X

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.scatter([1, 2, 3], [3, 2, 1])
plt.show()
```

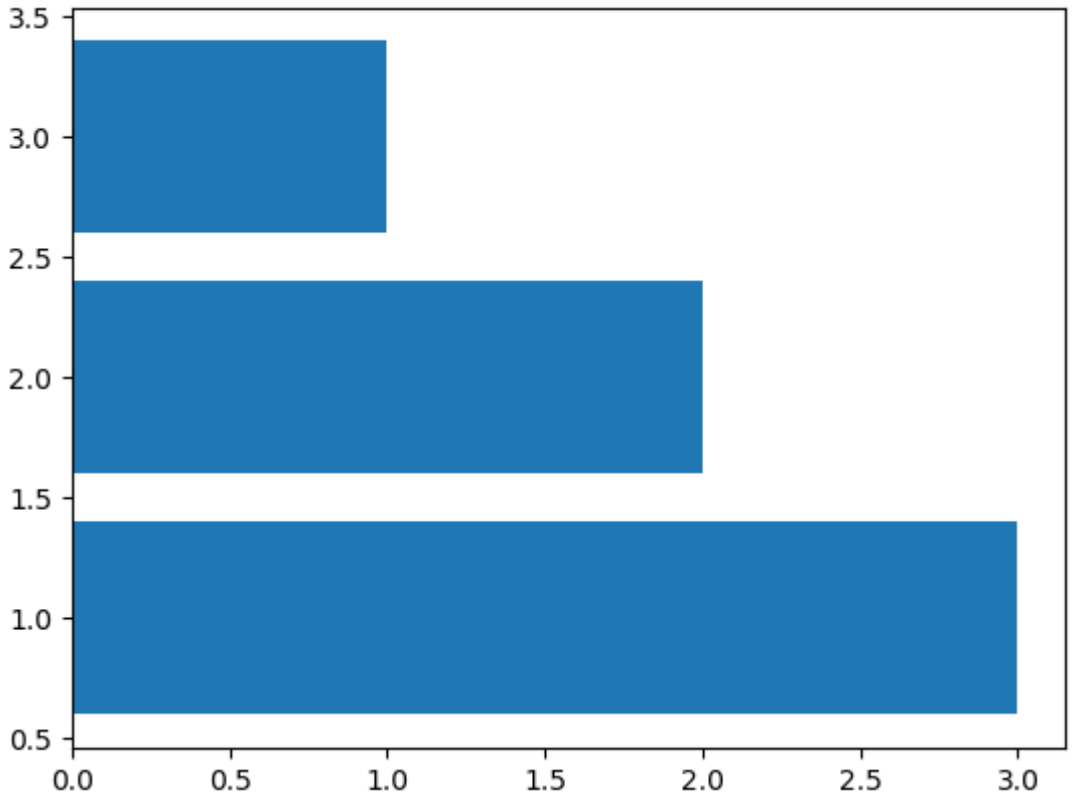

imagen:fuente propia

Histograma

hist(x, bins): Dibuja un histograma con las frecuencias resultantes de agrupar los datos de la lista **x** en las clases definidas por la lista **bins**.

```
import numpy as np
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
x = np.random.normal(5, 1.5, size = 1000)
ax.scatter(x, np.arange(0, 11))
plt.show()
```

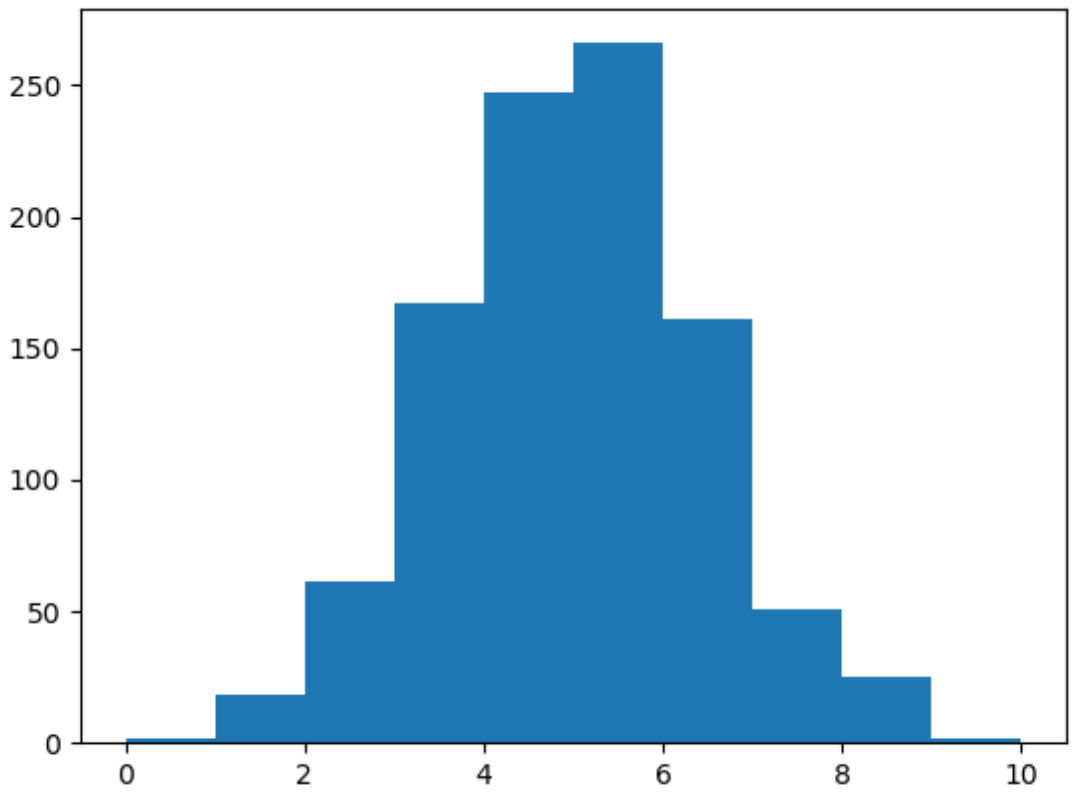

imagen:fuente propia

Diagrama de Caja y bigotes

boxplot(x): Dibuja un diagrama de caja y bigotes con los datos de la lista **x.**

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.boxplot([1, 2, 1, 2, 3, 4, 3, 3, 5, 7])
plt.show()
```

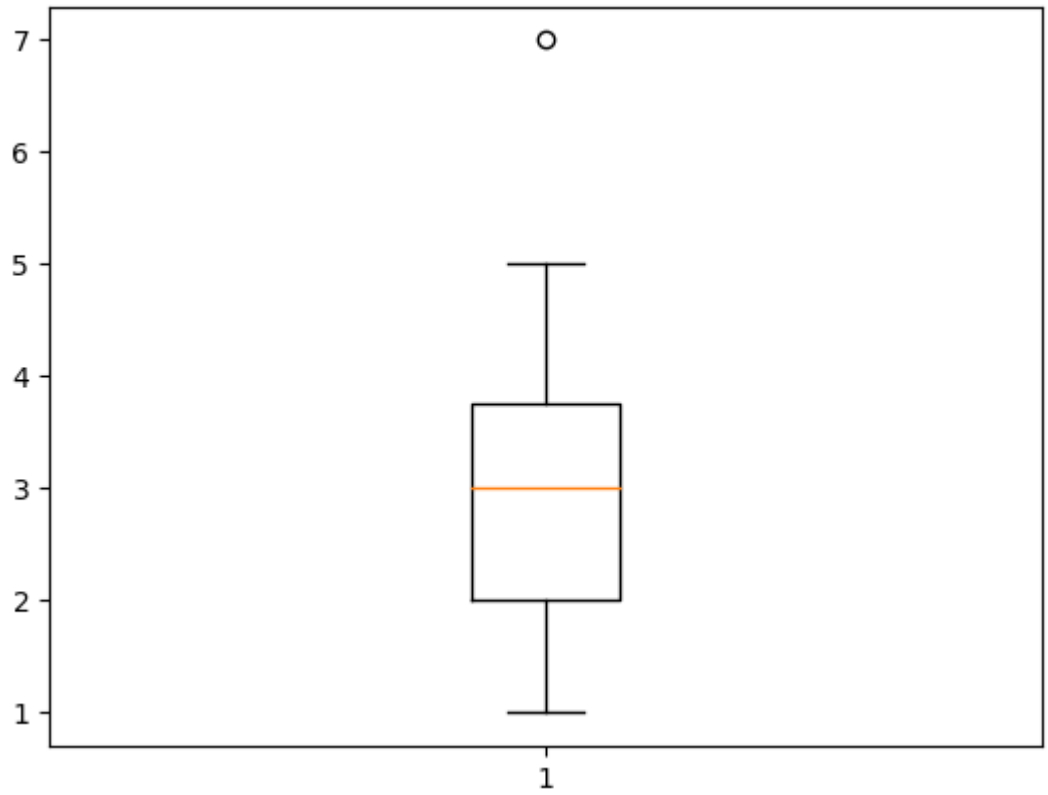

imagen:fuente propia

Figuras y subgráficos.

La figura es el contenedor dentro del cual vamos a crear una o más gráficas, cada una de ellas representada o accesible vía un conjunto de ejes. Recordemos los métodos principales para la creación de una figura y veamos cómo podemos personalizarla.

Además de la opción de usar una función como plot que genera una figura y un conjunto de ejes de forma implícita (aunque no nos permite su personalización), hemos visto ya los dos métodos principales para la creación de una figura. El primero es el uso de la función **matplotlib.pyplot.figure** que crea una nueva figura y devuelve una referencia a la misma:

```
fig = plt.figure()
```

La segunda forma vista consiste en el uso de la función **matplotlib.pyplot.subplots** que genera una figura y uno o más conjuntos de ejes (uno por defecto), devolviendo una referencia a la figura y a los conjuntos de ejes en cuestión:

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

En ambos casos, el resultado contenido en la variable fig es una referencia a un objeto de tipo **matplotlib.figure.Figure**, "contenedor" que engloba al resto de elementos involucrados en el dibujo de una gráfica.

Usando el primer método (la función **matplotlib.pyplot.figure**), es posible pasar como argumentos de la función valores que definen ciertas propiedades visuales de la figura. Generemos, en primer lugar, una figura sin personalizar (y añadamos en ella una sencilla gráfica):

```
y = np.random.randn(100).cumsum()
fig = plt.figure()
```

```
plt.plot(y)
plt.show()
```

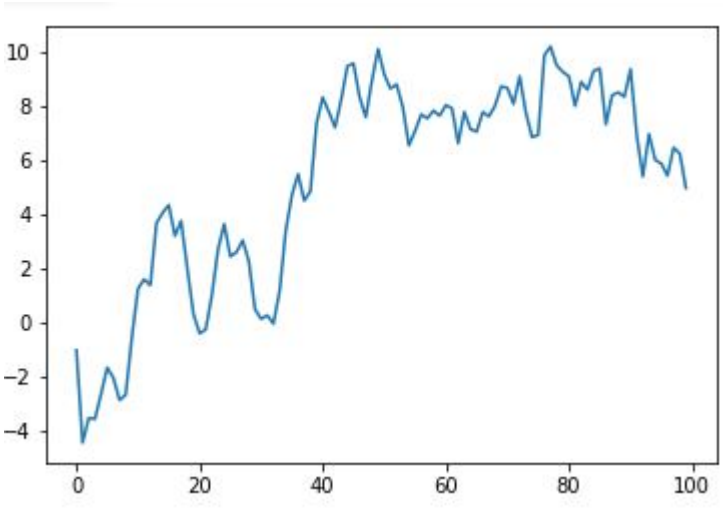

imagen:fuente propia

Posición y tamaño de los gráficos.

Para posicionar el mapa, la clase Basemap incluye los parámetros **lat_0** y **lon_0** para seleccionar la latitud (norte o sur) y longitud (este u oeste), respectivamente, del centro del mapa. Estos argumentos deben contener grados decimales. Adicionalmente, los parámetros **width** y **height** deberán contener el ancho y alto en metros del área a visualizar. Mostremos algunos ejemplos:

Madrid

Latitud: 40.4165000, longitud: -3.7025600, mapa de 1.000 km de ancho y 1.000 km de alto:

```
fig = plt.figure(figsize = (8, 8))
```

```
m = Basemap(projection = "loc"), lat_0 = 40.4165,
lon_0 = -3.70256, width = 1e6, height = 1e6,
resolution = "f")

m.shadedrelief();
```

Los Ángeles

Latitud: 34.0194, longitud: -118.411, mapa de 1.000 km de ancho y 2.000 km de alto:

```
fig = plt.figure(figsize = (5,10))

m = Basemap(projection = "loc"), lat_0 = 34.0194,
lon_0 = -118.411, width = 1e6, height = 2e6,
resolution = "f")

m.shadedrelief();
```

Colores, marcadores y estilos.

Los gráficos creados con Matplotlib son personalizables y puede cambiarse el aspecto de casi todos sus elementos. Los elementos que suelen modificarse más a menudo son:

- Colores
- Marcadores de puntos
- Estilo de líneas
- Títulos
- Ejes
- Leyenda

- Rejilla

Colores.

Para cambiar el color de los objetos se utiliza el parámetro `color = nombre-color`, donde `nombre-color` es una cadena con el nombre del color de entre los colores disponibles.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

dias = ['L', 'M', 'X', 'J', 'V', 'S', 'D']

temperaturas = {'Madrid':[28.5, 30.5, 31, 30, 28,
27.5, 30.5], 'Barcelona':[24.5, 25.5, 26.5, 25, 26.5,
24.5, 25]}

ax.plot(dias, temperaturas['Madrid'], color =
'tab:purple')

ax.plot(dias, temperaturas['Barcelona'], color =
'tab:green')

plt.show()
```

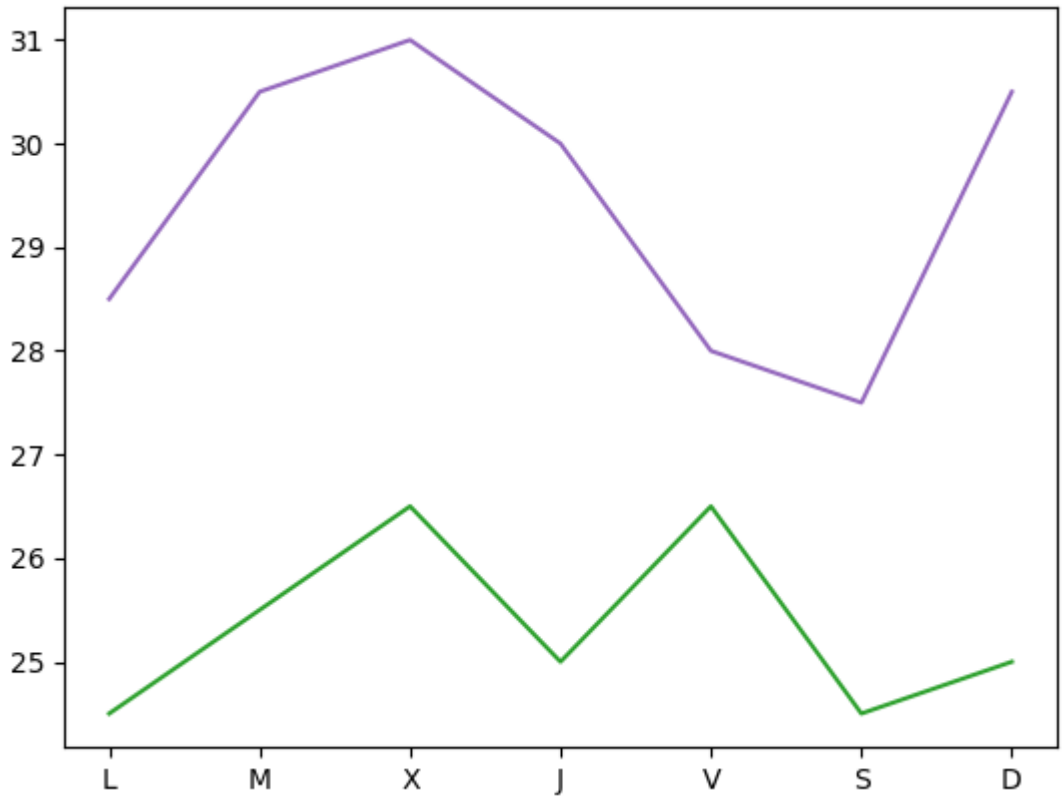

imagen:fuente propia

Marcadores.

Para cambiar la forma de los puntos marcadores se utiliza el parámetro `marker = nombre-marcador` donde `nombre-marcador` es una cadena con el nombre del marcador de entre los marcadores disponibles

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
dias = ['L', 'M', 'X', 'J', 'V', 'S', 'D']

temperaturas = {'Madrid':[28.5, 30.5, 31, 30, 28, 27.5,
30.5], 'Barcelona':[24.5, 25.5, 26.5, 25, 26.5, 24.5, 25]}
ax.plot(dias, temperaturas['Madrid'], marker = '^')
ax.plot(dias, temperaturas['Barcelona'], marker = 'o')

plt.show()
```

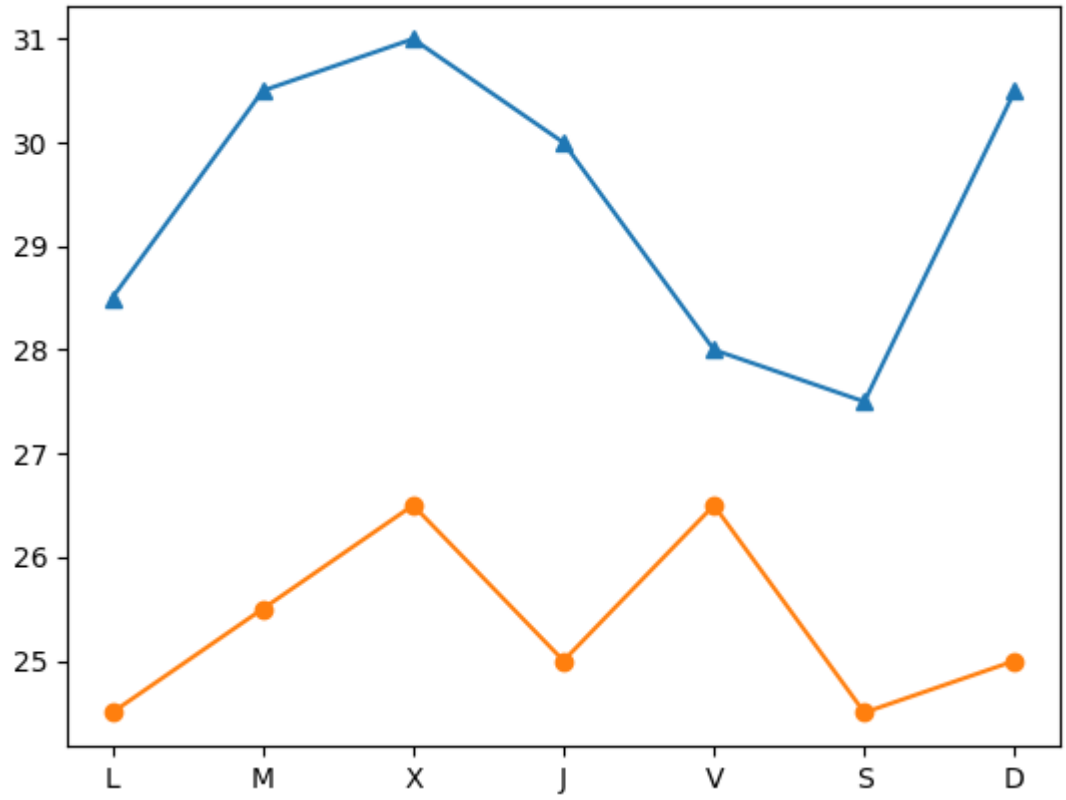

imagen:fuente propia

Líneas.

Para cambiar el estilo de las líneas se utiliza el parámetro `linestyle = nombre-estilo` donde `nombre-estilo` es una cadena con el nombre del estilo de entre los estilos disponibles

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

dias = ['L', 'M', 'X', 'J', 'V', 'S', 'D']

temperaturas = {'Madrid':[28.5, 30.5, 31, 30, 28, 27.5, 30.5],
'Barcelona':[24.5, 25.5, 26.5, 25, 26.5, 24.5, 25]}

ax.plot(dias, temperaturas['Madrid'], linestyle = 'dashed')

ax.plot(dias, temperaturas['Barcelona'], linestyle = 'dotted')
```

```
plt.show()
```

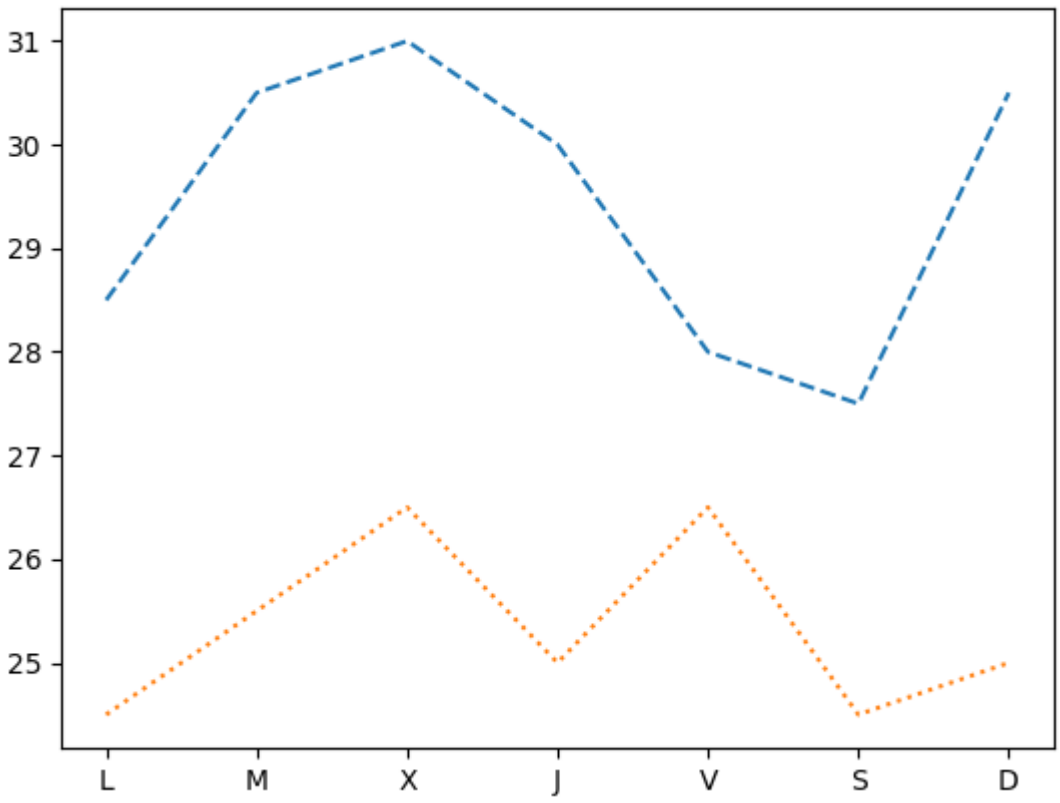

imagen:fuente propia

Títulos

Para añadir un título principal al gráfico se utiliza el siguiente método:

- `ax.set_title(titulo, loc=alineacion, fontdict=fuente)` : Añade un título con el contenido de la cadena `titulo` a los ejes `ax`. El parámetro `loc` indica la alineación del título, que puede ser `'left'` (izquierda), `'center'` (centro) o `'right'` (derecha), y el parámetro `fontdict` indica mediante un diccionario las características de la fuente (la el tamaño `fontisize`, el grosor `fontweight` o el color `color`).

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

dias = ['L', 'M', 'X', 'J', 'V', 'S', 'D']

temperaturas = {'Madrid':[28.5, 30.5, 31, 30, 28,
27.5, 30.5], 'Barcelona':[24.5, 25.5, 26.5, 25, 26.5,
24.5, 25]}

ax.plot(dias, temperaturas['Madrid'])

ax.plot(dias, temperaturas['Barcelona'])

ax.set_title('Evolución de la temperatura diaria',
loc = "left", fontdict = {'fontsize':14,
'fontweight':'bold', 'color':'tab:blue'})

plt.show()
```

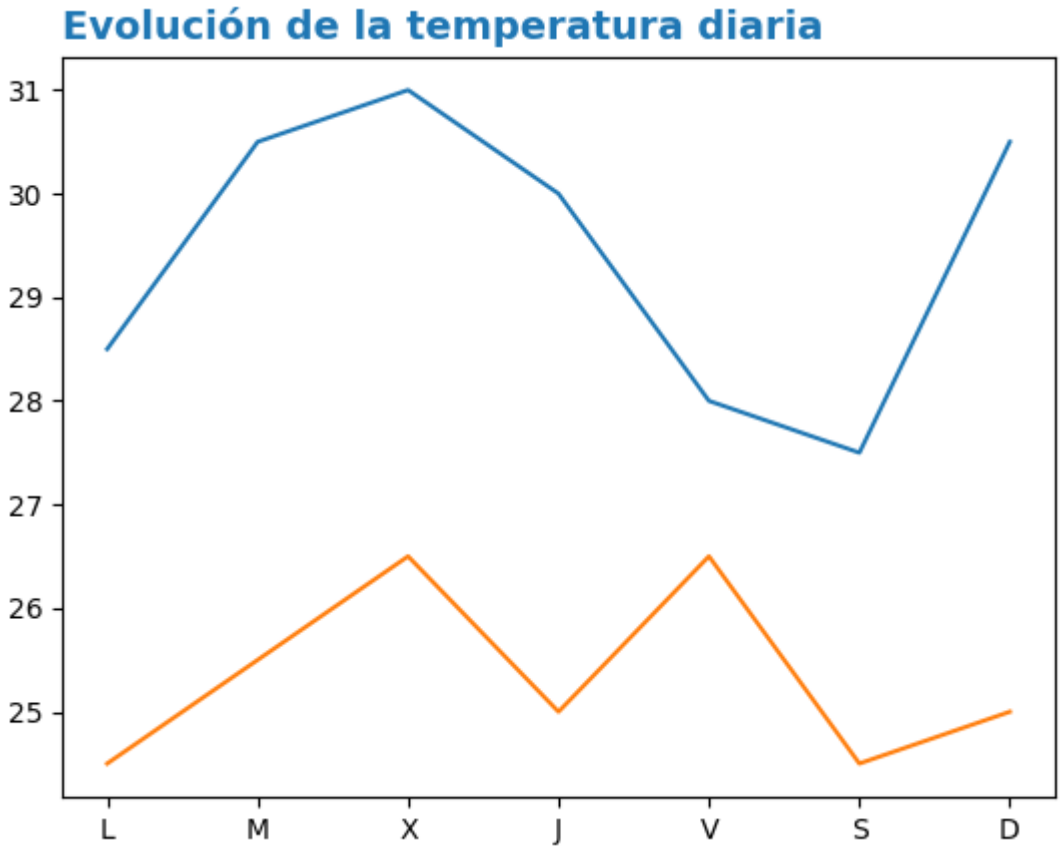

imagen:fuente propia

Ejes.

Para cambiar el aspecto de los ejes se suelen utilizar los siguientes métodos:

- `ax.set_xlabel(titulo)` : Añade un título con el contenido de la cadena `titulo` al eje x de `ax`. Se puede personalizar la alineación y la fuente con los mismos parámetros que para el título principal.
- `ax.set_ylabel(titulo)` : Añade un título con el contenido de la cadena `titulo` al eje y de `ax`. Se puede personalizar la alineación y la fuente con los mismos parámetros que para el título principal.
- `ax.set_xlim([limite-inferior, limite-superior])` : Establece los límites que se muestran en el eje x de `ax`.

- ax.set_ylim([limite-inferior, limite-superior]) : Establece los límites que se muestran en el eje y de ax.
- ax.set_xticks(marcas) : Dibuja marcas en el eje x de ax en las posiciones indicadas en la lista marcas.
- ax.set_yticks(marcas) : Dibuja marcas en el eje y de ax en las posiciones indicadas en la lista marcas.
- ax.set_xscale(escala) : Establece la escala del eje x de ax, donde el parámetro escala puede ser 'linear' (lineal) o 'log' (logarítmica).
- ax.set_yscale(escala) : Establece la escala del eje y de ax, donde el parámetro escala puede ser 'linear' (lineal) o 'log' (logarítmica).

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

dias = ['L', 'M', 'X', 'J', 'V', 'S', 'D']

temperaturas = {'Madrid':[28.5, 30.5, 31, 30, 28,
27.5, 30.5], 'Barcelona':[24.5, 25.5, 26.5, 25, 26.5,
24.5, 25]}

ax.plot(dias, temperaturas['Madrid'])

ax.plot(dias, temperaturas['Barcelona'])

ax.set_xlabel("Días", fontdict = {'fontsize':14,
'fontweight':'bold', 'color':'tab:blue'})

ax.set_ylabel("Temperatura ºC")

ax.set_ylim([20,35])

ax.set_yticks(range(20, 35))

plt.show()
```

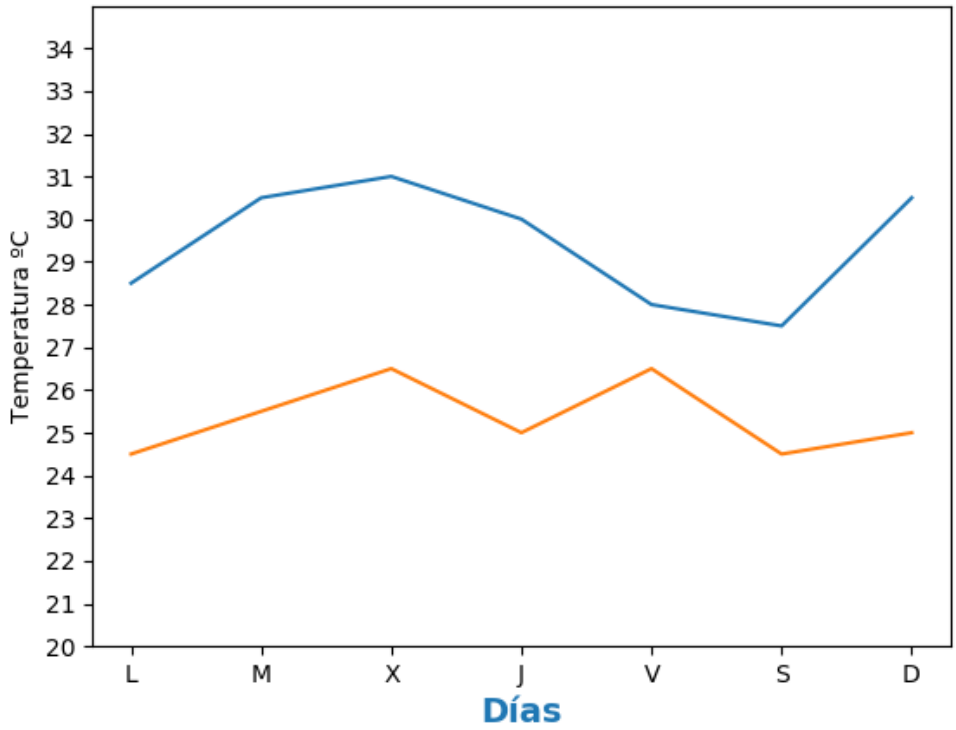

imagen:fuente propia

Leyenda

Para añadir una leyenda a un gráfico se utiliza el siguiente método:

- `ax.legend(leyendas, loc = posición)` : Dibuja un leyenda en los ejes `ax` con los nombres indicados en la lista `leyendas`. El parámetro `loc` indica la posición en la que se dibuja la leyenda y puede ser `'upper left'` (arriba izquierda), `'upper center'` (arriba centro), `'upper right'` (arriba derecha), `'center left'` (centro izquierda), `'center'` (centro), `'center right'` (centro derecha), `'lower left'` (abajo izquierda), `'lower center'` (abajo centro), `'lower right'` (abajo derecha). Se puede omitir la lista `leyendas` si se indica la leyenda de cada serie en la función que la dibuja mediante el parámetro `label`.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

dias = ['L', 'M', 'X', 'J', 'V', 'S', 'D']

temperaturas = {'Madrid':[28.5, 30.5, 31, 30, 28,
27.5, 30.5], 'Barcelona':[24.5, 25.5, 26.5, 25, 26.5,
24.5, 25]}

ax.plot(dias, temperaturas['Madrid'], label =
'Madrid')

ax.plot(dias, temperaturas['Barcelona'], label =
'Barcelona')

ax.legend(loc = 'upper right')

plt.show()
```

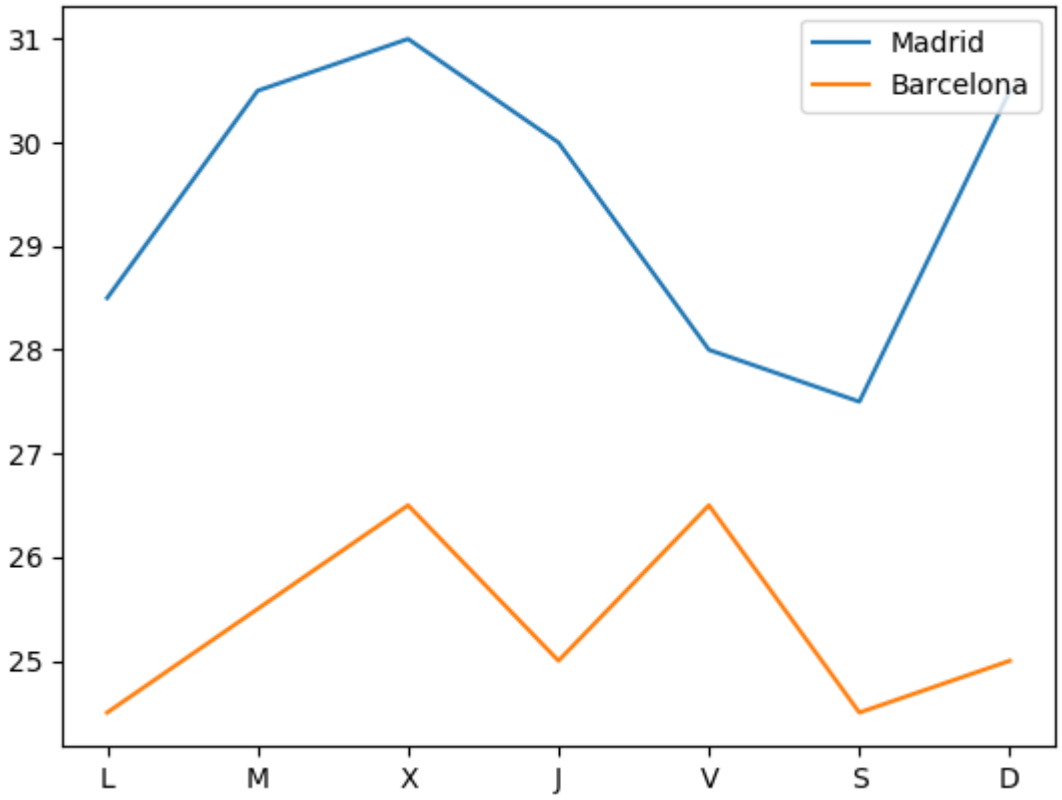

imagen:fuente propia

Rejilla

ax.grid(axis=ejes, color=color, linestyle=estilo) : Dibuja una rejilla en los ejes de ax. El parámetro axis indica los ejes sobre los que se dibuja la rejilla y puede ser 'x' (eje x), 'y' (eje y) o 'both' (ambos). Los parámetros color y linestyle establecen el color y el estilo de las líneas de la rejilla, y pueden tomar los mismos valores vistos en los apartados de colores y líneas.

```
import matplotlib.pyplot as plt

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

```
dias = ['L', 'M', 'X', 'J', 'V', 'S', 'D']

temperaturas = {'Madrid':[28.5, 30.5, 31, 30, 28,
27.5, 30.5], 'Barcelona':[24.5, 25.5, 26.5, 25, 26.5,
24.5, 25]}

ax.plot(dias, temperaturas['Madrid'])

ax.plot(dias, temperaturas['Barcelona'])

ax.grid(axis = 'y', color = 'gray', linestyle =
'dashed')

plt.show()
```

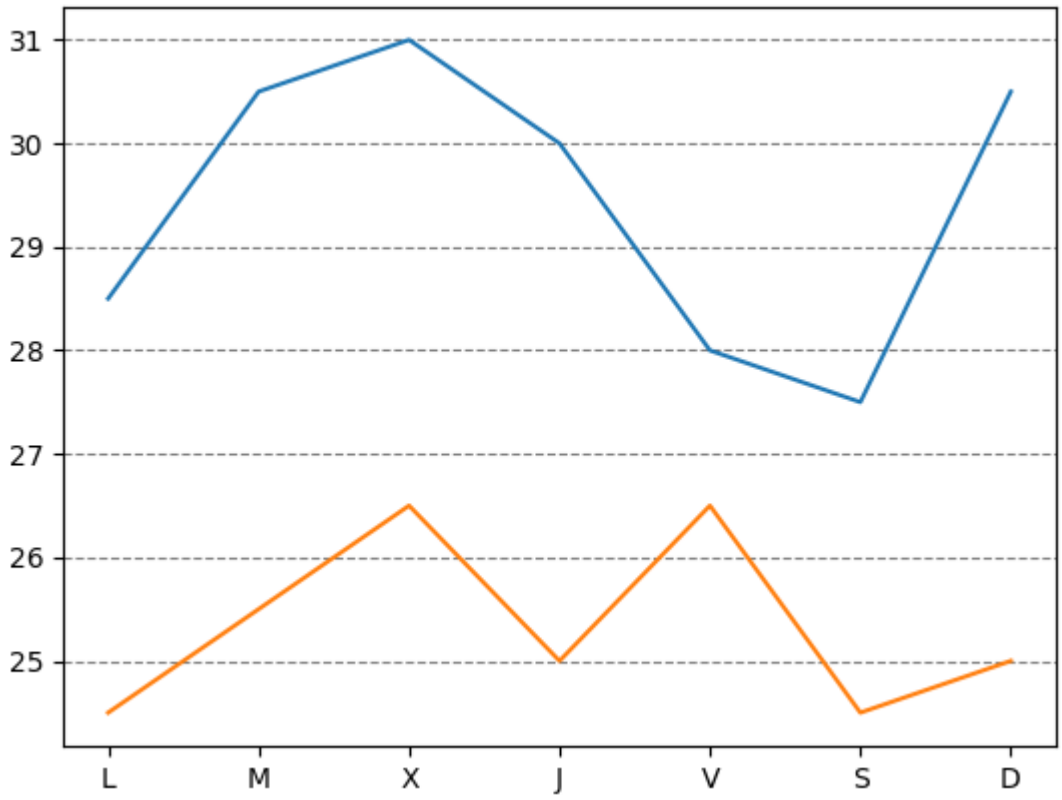

imagen:fuente propia

Ticks

Las marcas de graduación son los marcadores que indican puntos de data en los ejes. Matplotlib hasta ahora, en todos nuestros ejemplos anteriores, se ha hecho cargo automáticamente de la tarea de espaciar puntos en el eje. Los localizadores y formateadores de ticks predeterminados de Matplotlib están diseñados para ser generalmente suficientes en muchas situaciones comunes. La posición y las etiquetas de las graduaciones se pueden mencionar explícitamente para cumplir con requisitos específicos.

La función **xticks ()** y **yticks ()** toma un objeto de lista como argumento. Los elementos de la lista indican las posiciones en la acción correspondiente donde se mostrarán las marcas de verificación.

ax.set_xticks ([2,4,6,8,10])

Este método marcará puntos de data en posiciones dadas con graduaciones.

Asimismo, las etiquetas correspondientes a las marcas de graduación se pueden definir mediante **set_xlabels ()** y **set_ylabels ()** respectivamente.

ax.set_xlabels (['dos ', 'cuatro ', 'seis ', 'ocho ', 'diez '])

Esto mostrará las etiquetas de texto debajo de los marcadores en el eje x.

El siguiente ejemplo muestra el uso de marcas y etiquetas.

```
import matplotlib.pyplot as plt

import numpy as np import

math x = np.arange (0, math.pi * 2, 0.05)

fig = plt.figure ()

ax = fig.add_axes ([0.1, 0.1, 0.8, 0.8])
```

```
# ejes principales
y = np.sin(x)
ax.plot (x, y)
ax.set_xlabel('angle')
ax.set_title('sine')
ax.set_xticks([0, 2, 4, 6])
ax.set_xticklabels(['zero', 'two', 'four', 'six'])
ax .se t_yticks ([-1, 0, 1])
plt.show()
```

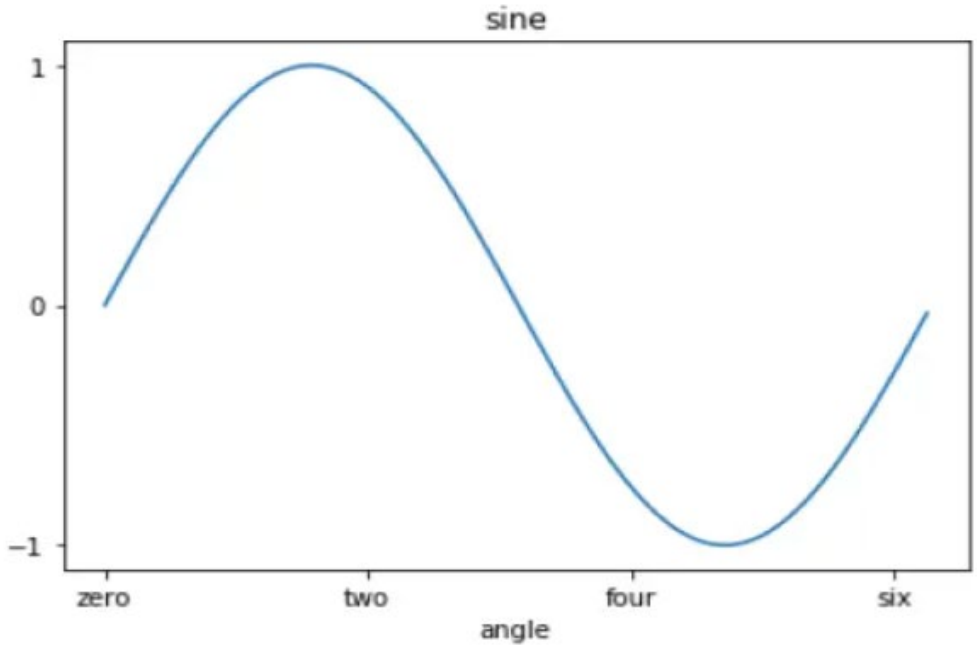

imagen:fuente propia

Límites de los ejes.

Para establecer los límites solo para el eje X, podríamos usar los métodos xlim() y set_xlim(). De manera similar para establecer los límites para el eje Y, podríamos usar los métodos ylim() y set_ylim(). También podemos usar el método axis() que puede controlar el rango de ambos ejes.

xlim() y ylim() para establecer límites de ejes en Matplotlib

matplotlib.pyplot.xlim() y matplotlib.pyplot.ylim() se puede usar para establecer u obtener límites para el eje X y el eje Y, respectivamente. Si pasamos argumentos en estos métodos, establecen los límites para los ejes respectivos y si no pasamos ningún argumento, obtenemos un rango de los ejes respectivos

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

x = np.linspace(0, 10, 500)
y = np.sin(2 * np.pi *x) + 1

fig = plt.figure(figsize = (8,6))
plt.plot(x, y)
plt.title("Fijando rangos de ejes", fontsize = 25)
plt.xlabel("x", fontsize = 18)
plt.xlabel("1 + sin(x)", fontsize = 18)
plt.ylim(4, 8)
plt.ylim(-0.5, 2.5)
plt.show()
```

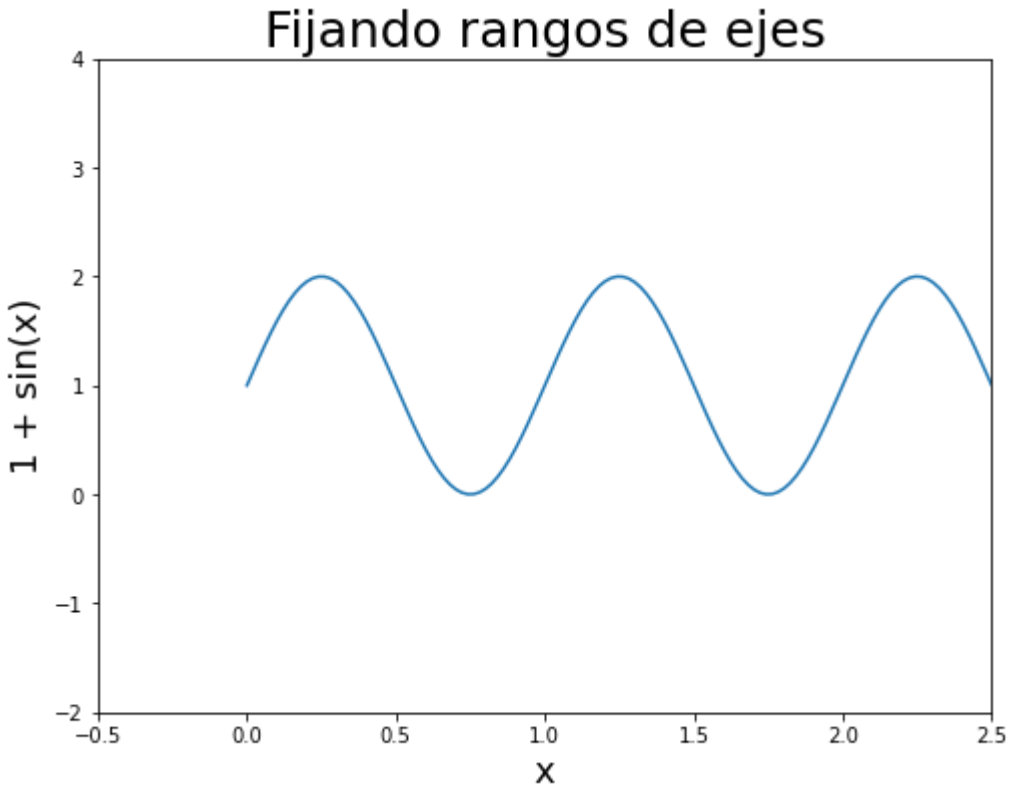

imagen:fuente propia

Esto establece el rango del eje X de `4` a `8` mientras que el del eje Y de `-0.5` a `2.5`.

```
import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(0, 10, 500)

y = np.sin(2 * np.pi *x) + 1

fig = plt.figure(figsize = (8,6))

plt.plot(x, y)

plt.title("Sin fijar ejes", fontsize = 25)

plt.xlabel("x", fontsize = 18)
```

```
plt.ylabel("1 + sin(x)", fontsize = 18)

plt.show()
```

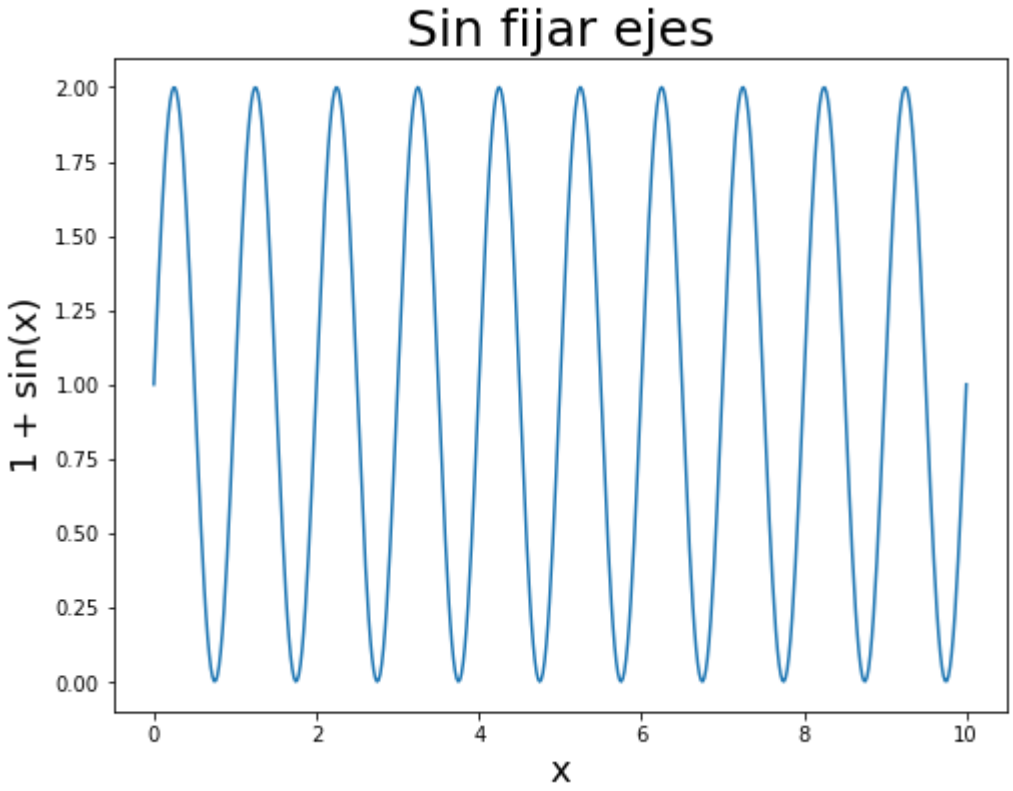

imagen:fuente propia

Como se ve en la figura de salida, si no utilizáramos las funciones xlim() y ylim(), obtendríamos una gráfica con el rango completo de ejes que es el eje X que va de 0 a 10 mientras que el eje Y va de 0 a 2

Métodos set_xlim() y set_ylim() para establecer límites de eje

El matplotlib.axes.Axes.set_xlim y matplotlib.axes.Axes.set_ylim también se utilizan para establecer los límites del rango de números que se visualizan en la gráfica resultante.

```
import numpy as np

import matplotlib.pyplot as plt


X=np. linspace(0,10,500)

y=np.sin(2 * np.pi * x)+1

fig = plt. figure (figsize=(8, 6))

axes = plt. axes ()

axes.set_xlim([4, 8])

axes.set_ylim([-0.5, 2.5])

plt.plot (x, y)


plt.title("Ajuste de la gama de ejes", fontsize=25)

plt.xlabel("x",fontsize=18)

plt.ylabel("1+sin", fontsize=18)

plt.show()
```

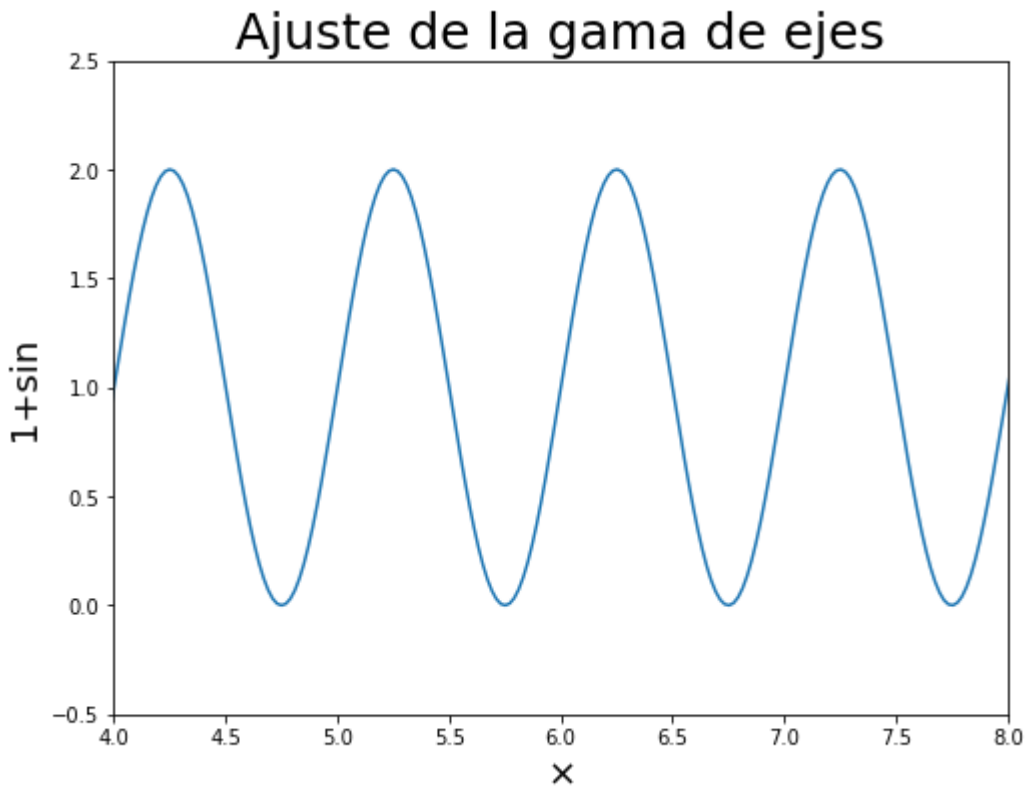

imagen:fuente propia

Método axis() para establecer límites de ejes en Matplotlib

También podríamos usar matplotlib.pyplot.axis() para establecer límites de ejes. La sintaxis es la siguiente:

```
plt.axis([xmin, xmax, ymin, ymax])
```

Este método elimina la necesidad de una función separada para controlar el eje X y el eje Y.

```
import numpy as np

import matplotlib.pyplot as plt


X=np. linspace(0,10,500)

y=np.sin(2 * np.pi * x)+1

fig = plt. figure (figsize=(8, 6))
```

```
axes = plt.axis([4, 9, -0.5, 2.5])
plt.plot (x, y)


plt.title("Ajuste de la gama de ejes", fontsize=25)
plt.xlabel("x",fontsize=18)
plt.ylabel("1+sin", fontsize=18)
plt.show()
```

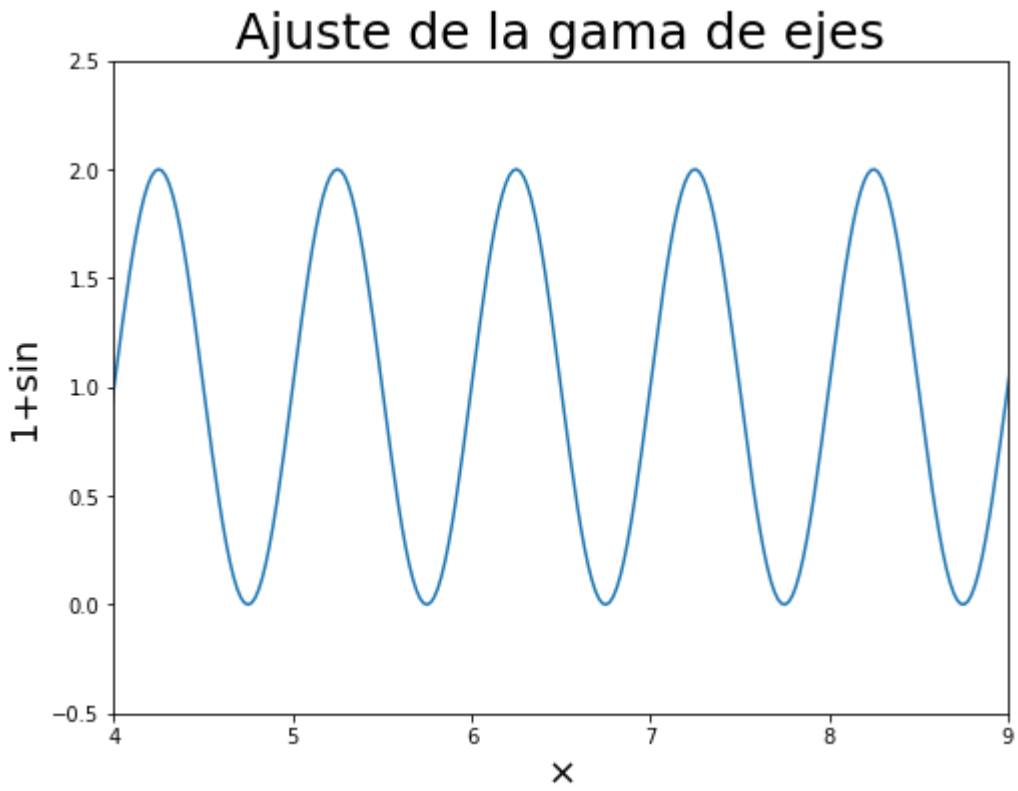

imagen:fuente propia

Anotaciones y dibujos en un gráfico

Matplotlib **text()** La función puede agregar texto en la posición especificada en el eje de coordenadas (Ejes). El texto se usa generalmente para explicar gráficos, en matplotlib **annotate()** a menudo y **text()** en conjunto con, l**as notas incluyen textos destacados (como flechas indicadoras) y texto**, como se muestra abajo. Por lo tanto, al agregar un comentario, consideramos al menos dos puntos:

- Ubicación etiquetada **xy**, Es una tupla bidimensional, con forma de (2, 3).
- Posición del texto **xytext**, Es una tupla bidimensional.

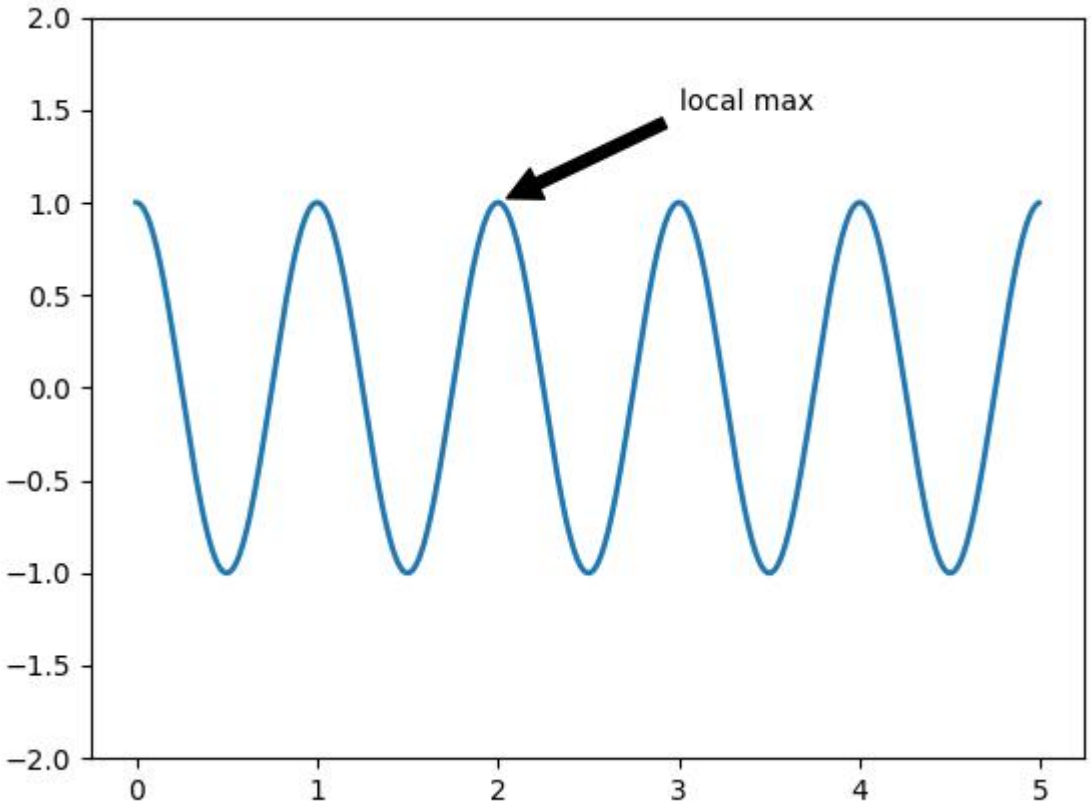

imagen:fuente propia

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


fig, ax =plt.subplots()
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2 * np.pi * t)
ax.plot(t, s, lw = 2) # lw ancho de línea


ax.annotate("local max", xy = (2, 1),
            xytext = (3, 1.5),
            arrowprops = {facecolor: 'black'})


ax.set_ylim(-2, 2)
plt.show()
```

En el ejemplo anterior, **xy** con **xytext** el sistema de coordenadas se llama coordenada de datos, que es **xy** el sistema de referencia predeterminado (los parámetros son **xycoords**), y **xytext** marco de referencia (el parámetro es **textcoords**) por defecto **xy** lo mismo. Los valores de los parámetros del sistema de coordenadas son los siguientes:

Valor del parámetro	Sistema coordinado
'figure points'	Tome la imagen como el sistema de coordenadas, el valor de la esquina inferior izquierda es bajo
'figure pixels'	Igual que el anterior, la diferencia es puntos y píxeles, y no hay diferencia real
'figure fraction'	Tomando la imagen como el sistema de coordenadas, el valor es la relación del tamaño de la imagen, (0, 0) significa sentarse, (1, 1) significa arriba a la derecha
'axes points'	Con el eje de coordenadas como sistema de referencia, ...
'axes pixels'	Con el eje de coordenadas como sistema de referencia,
'axes fraction	Con el eje de coordenadas como sistema de referencia, ...
'data'	¡Use puntos de datos en el eje de coordenadas, de uso común! Por ejemplo, la posición marcada (2, 1) en la figura anterior corresponde a un valor de coordenadas horizontales de 2 y un valor de coordenadas verticales de 1.

Por ejemplo, cambiamos el sistema de referencia del texto a **axes fraction** de, a saber:

```
ax.annotate('local max', xy = (3,1),
            xycoords = 'data', xytext = (0.8, 0.95),
            textcoords = 'axes fraction',
            arrowporps = dict{facebolor = 'black'})
```

Como se muestra en el ejemplo anterior, usando parámetros **arrowprops** para establecer el atributo de la flecha marcada, el valor de este parámetro es un diccionario (la clave del diccionario es el nombre del atributo y el valor es el valor del atributo), y los atributos relacionados se muestran en la siguiente tabla:

Nombre del Atributo	Introducción
width	Ancho de la flecha
frac	Relación de longitud de flecha
headwidth	Ancho de la punta de flecha
shrink	Relación entre la posición de la etiqueta y el texto.
**kwargs	y muchos más

Utilizando orientación a objetos para graficar

Utilizando este estilo, vamos a referenciar la figura y los ejes mediante variables y serán métodos y atributos de estas variables los que nos permitirán personalizar la visualización. Por ejemplo, el código equivalente a la figura anterior sería el siguiente:

```
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[1].set_title("Almacenes")

plt.show()
```

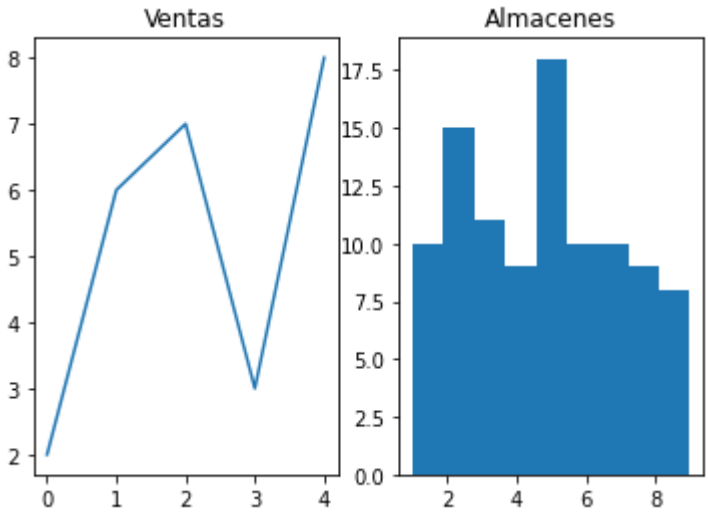

imagen:fuente propia

En este caso, las variables ***fig*** y ***ax*** nos servirán para personalizar todas las gráficas a crear en la figura. Vemos que la única función que hemos utilizado ha sido la primera, **plt.subplots**, para crear la figura y los dos conjuntos de ejes (bueno, y plt.show), y el resto de instrucciones son métodos que ejecutamos sobre dichos objetos.

La mayor parte de los métodos que nos permiten fijar un valor a un atributo comienza por "*set_*". Estos métodos suelen tener un método equivalente que comienza habitualmente por "*get_*" que sirve para leer el valor de dicho atributo.

Este estilo orientado a objetos nos da mayor control sobre las visualizaciones que creemos.

Referencias

[1] Librería Matplotlib

https://www.youtube.com/watch?v=2VeHtuqW3YY

https://programacion.net/articulo/introduccion_a_la_libreria_matplotlib_de_python_1599

[2] Matplotlib

https://matplotlib.org/

[3] Estilos.

https://interactivechaos.com/es/manual/tutorial-de-matplotlib/estilos

[4] Gráficas, programación orientada a objetos en Python y Matplotlib

https://www.youtube.com/watch?v=4AYPLe8yosY