Visualización de datos

Existen varias librerías populares de visualización de datos, siendo la principal `matplotlib`

https://matplotlib.org/

`matplotlib` permite a través de la programación crear todo tipo de gráficas y personalizar cualquiera de sus aspectos.

- Permite crear gráficas de alta calidad.
- Permite interactuar con ellas (zoom, actualizar)
- Permite controlar los estilos de líneas, de marcadores, de fuentes, las propiedades de los ejes, las etiquetas
- Permite exportar la gráfica a muchos formatos de ficheros

Este control muy fino permite gráficas realmente espectaculares.

Entre las mejores, sin duda:
https://github.com/rougier/scientific-visualization-book

Un gráfica reciente llamada bump chart, Premier League
https://twitter.com/utdmaram/status/1349117512248664071?s=20

Algún ejemplos de animación:

- https://twitter.com/SilverJacket/status/1348290841073299459?s=20
- https://twitter.com/camminady/status/1350407474558464001?s=20

> *Podéis encontrar muchos ejemplos en @matplotlib*

Pero la curva de aprendizaje es muy empinada...

> *Nosotros veremos unos principios básicos, y recurriremos a algunas funciones de visualización en* `pandas` *así como a la librería* `seaborn` *ambas construidas con* `matplotlib` *pero que facilitan la exploración rápida de conjuntos de datos.*

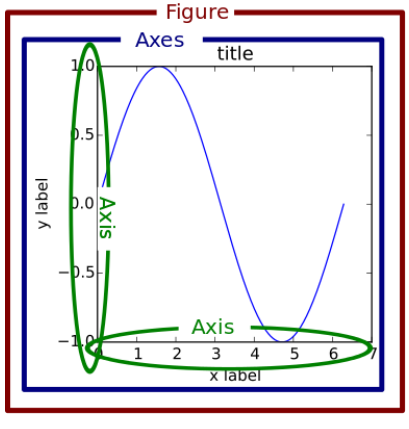
> *Un recurso muy útil: las "cheatsheets" oficiales de `matplotlib', disponibles en: https://github.com/matplotlib/cheatsheets/*

Gráficas básicas en `matplotlib`

`matplotlib` presenta una jerarquía de los distintos elementos de una gráfica:

Sus elementos principales son:

- `figure`: la figura completa que puede contener diferentes gráficas
- `axes`: lo que normalmente entendemos por gráfica.
- `axis`: los ejes de cada gráfica
- `artist`: todo lo que se ve en una figura es un `artist`, incluso `figure`, `axes`, `axis`.



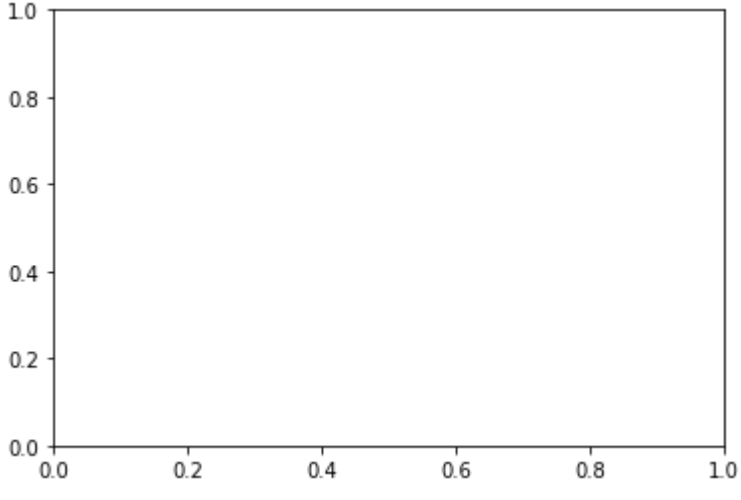
El primer paso consiste en importar pyplot del módulo `matplotlib`

```
from matplotlib import pyplot as plt
```

Empezaremos por crear el elemento `figure` y su o sus `axes` (gráficas)

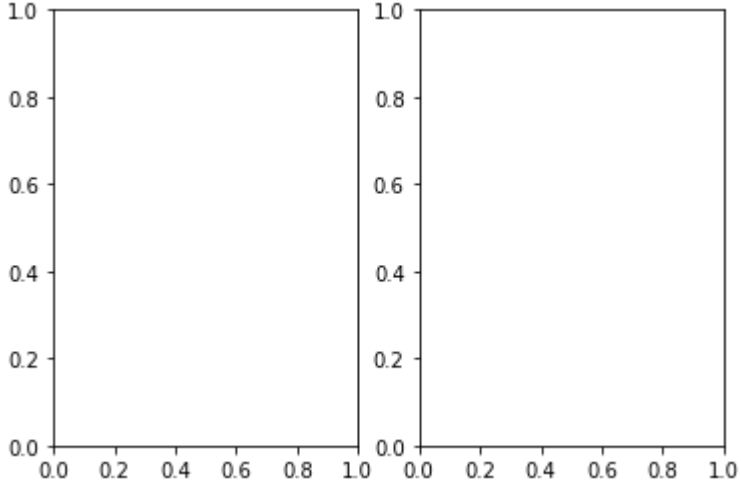
Para ello, usaremos la instrucción `subplots`

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



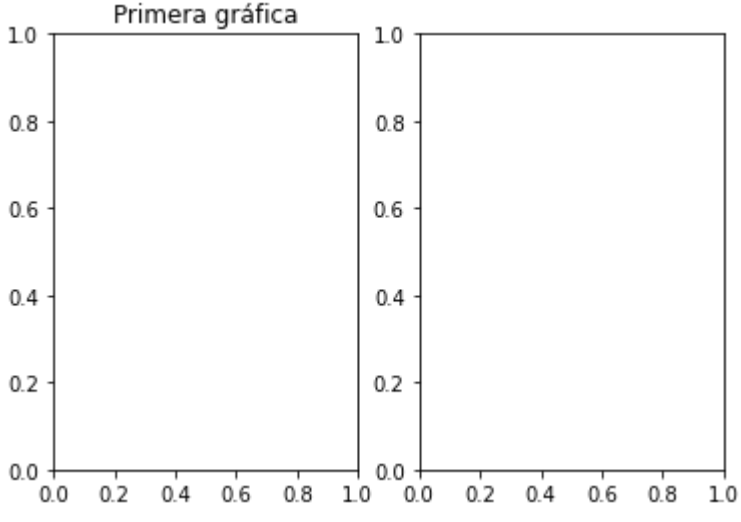
Podemos especificar una matriz de `axes` : por ejemplo una fila y dos columnas

```
fig, axs = plt.subplots(1, 2)
```



Ahora añadimos elementos al objeto `ax` o a cada componente de `axs`, `axs[0]` o `axs[1]`, aplicandoles métodos, por ejemplo, añadiendo un título:

```
fig, axs = plt.subplots(1, 2)
axs[0].set_title("Primera gráfica");
```

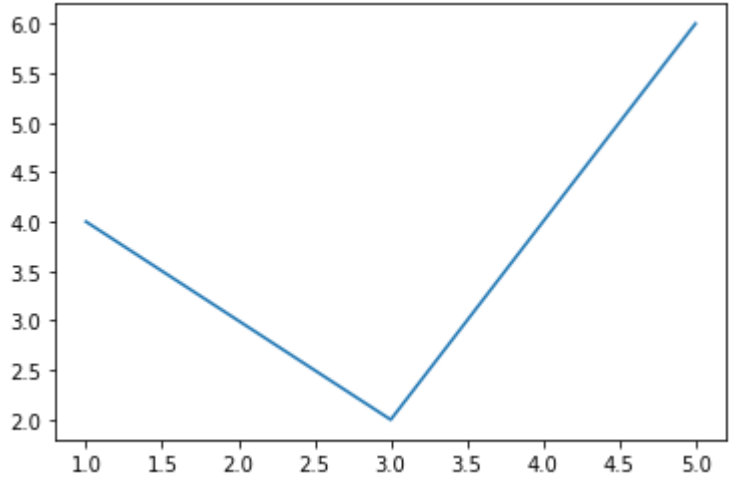


Consideremos unos datos sencillos

```
import pandas as pd
df = pd.DataFrame({
    'col1': [3, 1, 5],
    'col2': [2, 4, 6],
    'col3': ['a', 'b', 'c'],
    'col4': ['Hombre', 'Mujer', 'Hombre'],
    'col5': [100, 300, 200]
    }
)
df.sort_values('col1', inplace=True)
```

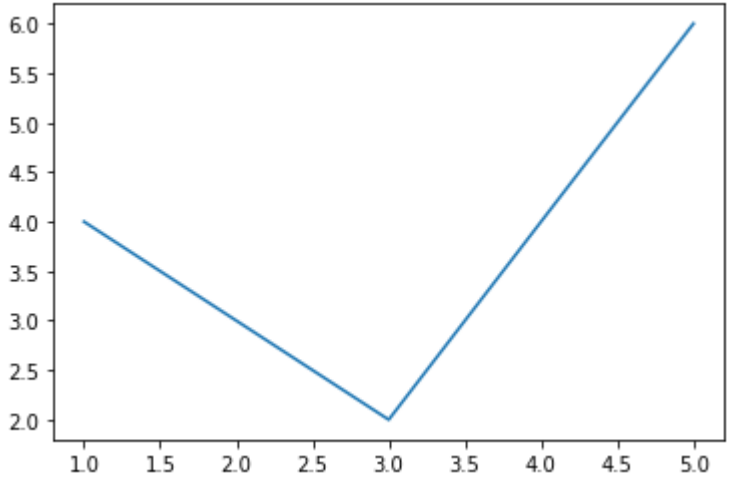
Vamos a crear un `figure` con un único `axes`, y representar `col2` en función de `col1`

```
fig, ax = plt.subplots()
ax.plot(df['col1'], df['col2']);
```



En la gráfica representada

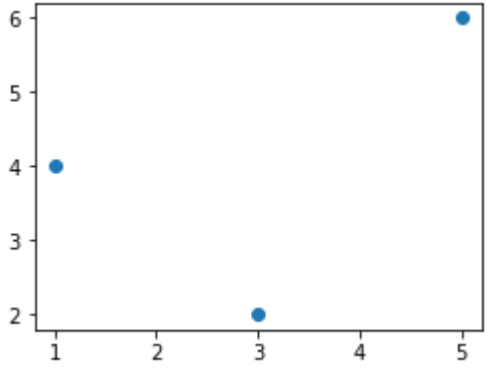
```
fig, ax = plt.subplots()
ax.plot(df['col1'], df['col2']);
```



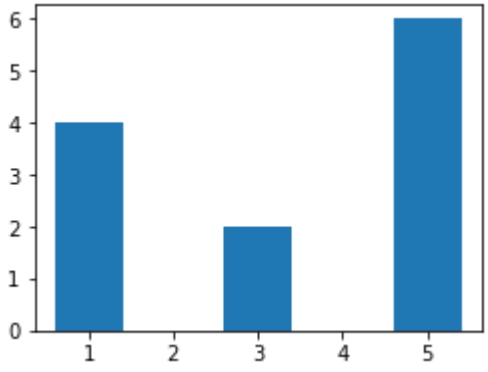
El método `plot` representa líneas que unen los puntos por orden de aparición en los vectores `df['col1']` y `df['col2']`.

Otros métodos que se aplican a un `axes` representan otros tipos de gráficas:

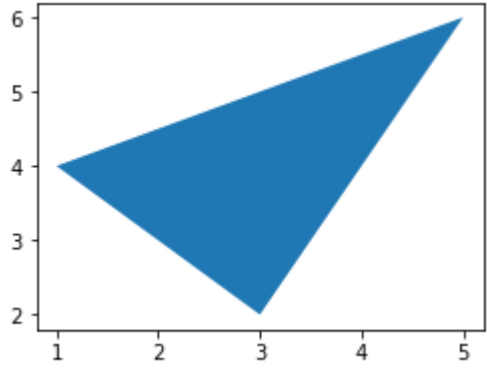
```
fig, ax = plt.subplots(figsize=(4, 3))
ax.scatter(df['col1'], df['col2']);
```



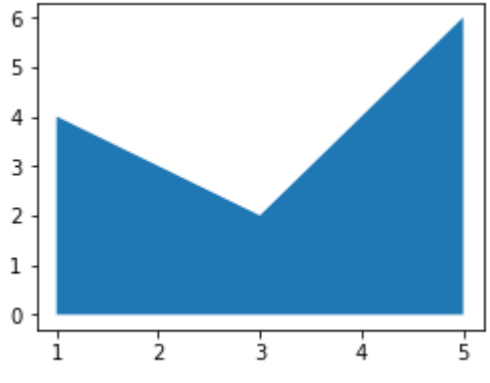

```
fig, ax = plt.subplots(figsize=(4, 3))
ax.bar(df['col1'], df['col2']);
```


6
5
4
3
2
1
0
1
2
3
4
5

```
fig, ax = plt.subplots(figsize=(4, 3))
ax.fill(df['col1'], df['col2']);
```



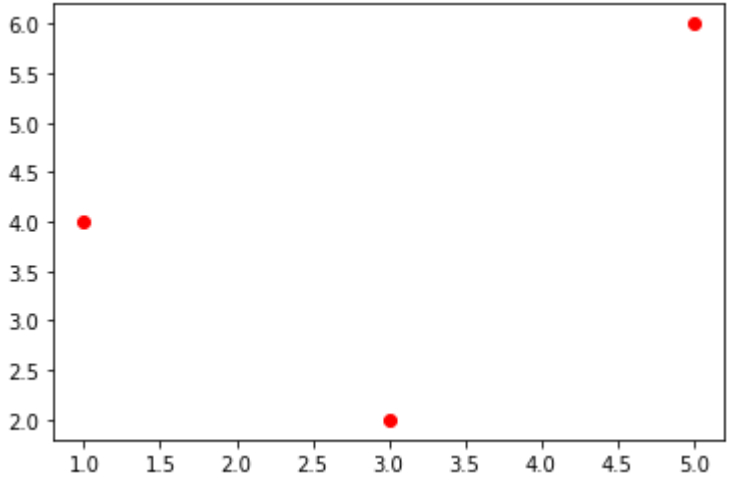

```
fig, ax = plt.subplots(figsize=(4,3))
ax.fill_between(df['col1'], df['col2']);
```



Para fijar algunos aspectos de la representación gráfica

Podemos fijar el color de los marcadores, las barras o las líneas.

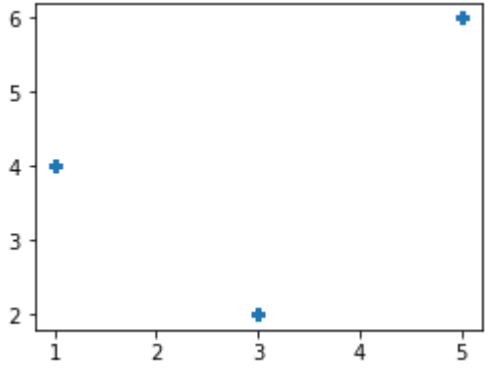
```
fig, ax = plt.subplots()
ax.scatter(df['col1'], df['col2'], color='red');
```



> *El control de los colores en `matplotlib` es muy extenso, por una parte, se puede consultar la lista de nombres de colores soportados en https://matplotlib.org/3.3.3/gallery/color/named_colors.html Por otra parte, se pueden usar "colormaps", para disponer de colores de variación uniforme. Más información más adelante.*

Podemos fijar el tipo de marcadores

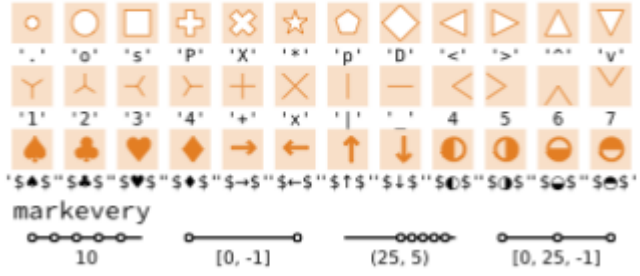
```
fig, ax = plt.subplots(figsize=(4, 3))
ax.scatter(df['col1'], df['col2'], marker='P');
```



Ver los marcadores posibles en la cheatsheet: https://github.com/matplotlib/cheatsheets/ de la que está extraida la siguiente imagen:

Markers

API

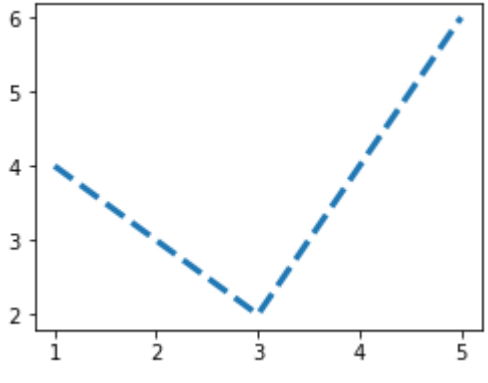


markevery

Podemos fijar el tipo de líneas y su grosor

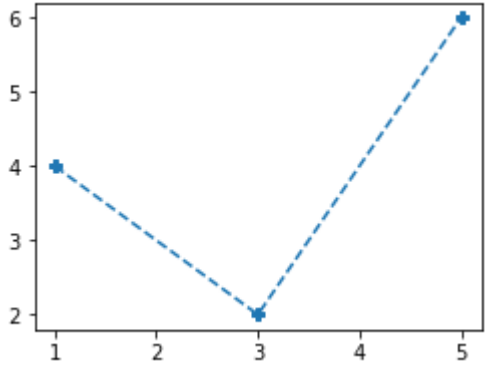
Extraido de la cheatsheet https://github.com/matplotlib/cheatsheets/:


```
fig, ax = plt.subplots(figsize=(4, 3))
ax.plot(df['col1'], df['col2'], linestyle='--', linewidth=3);
```



Podemos combinar marcadores y líneas:

```
fig, ax = plt.subplots(figsize=(4, 3))
ax.plot(df['col1'], df['col2'], linestyle='--', marker='P');
```



Todos los elementos de una figura son modificables

Para ello, existen una gran variedad de métodos que se aplican a `figure` o `axes` o a sus elementos.

Nos fijamos por ejemplo en los básicos mencionados en la cheatsheet

```
ax.grid() # Para añadir una rejilla
ax.set_xlim(vmin, vmax) # fija los límites del eje Ox, igual para set_ylim
ax.set_xlabel(label) # Establece la etiqueta del eje Ox, igual para Oy
ax.set_xticks(list) # indica en qué valores debe aparecer "ticks" en Ox, igual para Oy
ax.set_ticklabels(list) # indica las etiquetas de los "ticks" en Ox, igual en Oy
ax.set_title(title) # Indica el título de la gráfica
ax.set_tick_params(width=10) # fija diferentes parámetros por ejemplo rotation, se puede especificar axis
```

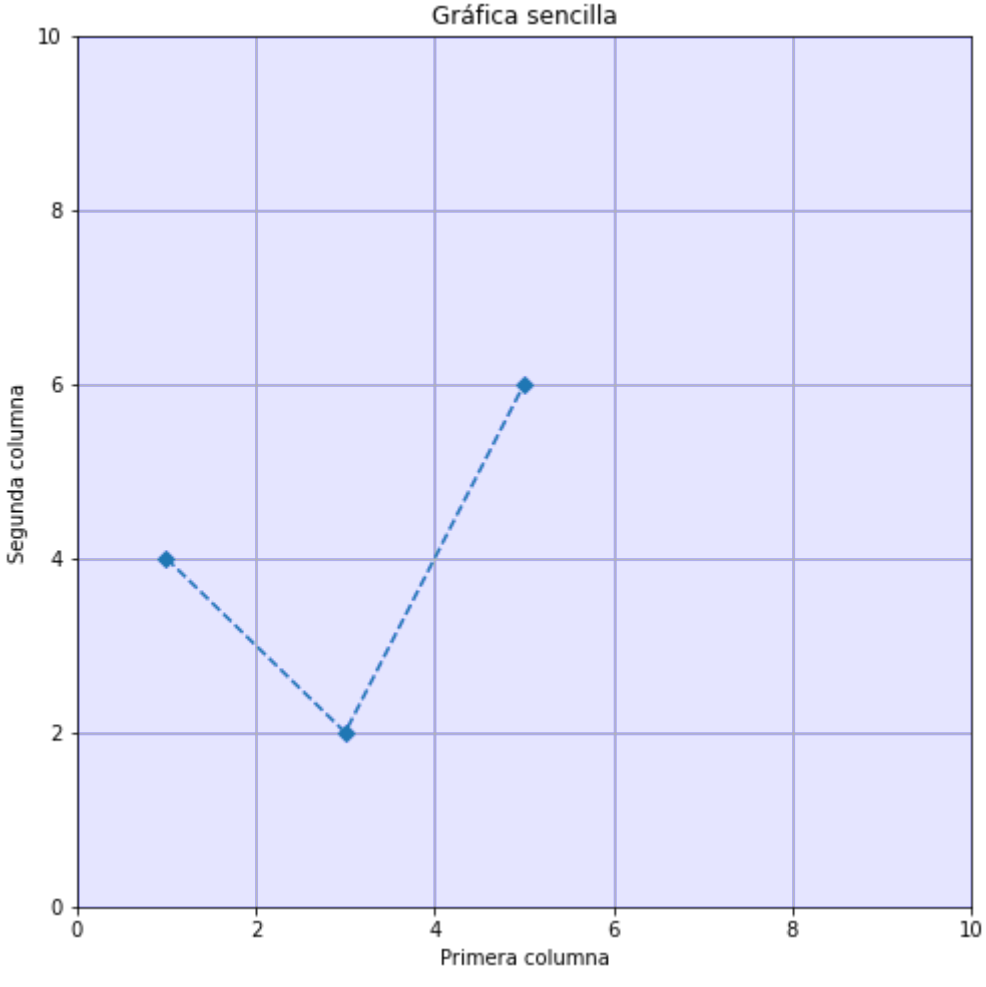
Para evitar solapamientos en las etiquetas de los ejes, los títulos, etc... podemos usar el método `tight_layout` que se aplica a un `axes` concreto o a todo un `figure`.

```
fig.tight_layout() # también podría ser ax.tight_layout()
```

El patch es un rectángulo de un elemento geométrico con color de fondo y color de borde. Tanto `figure` como `axes` tienen patch. Podemos usar los métodos `set_facecolor`, `set_edgecolor` y `set_alpha` sobre un patch:

```
fig.patch.set_facecolor('green') # El patch
```

Intentad reproducir la figura siguiente



Mapas de colores (`colormaps`) en `matplotlib`

Muy a menudo necesitamos un conjunto de colores para distinguir grupos de datos en la gráfica. `matplotlib` proporciona muchos mapas de colores diseñados para permitir visualizar de manera fidedigna la variación de una o varias cantidades.

Se puede consultar el tutorial oficial y la cheatsheet presenta varios mapas de colores:

Colormaps

API

`plt.get_cmap(name)`

Uniform

viridis
magma
plasma

Sequential

Greys
YlOrBr
Wistia

Diverging

Spectral
coolwarm
RdGy

Qualitative

tab10
tab20

Cyclic

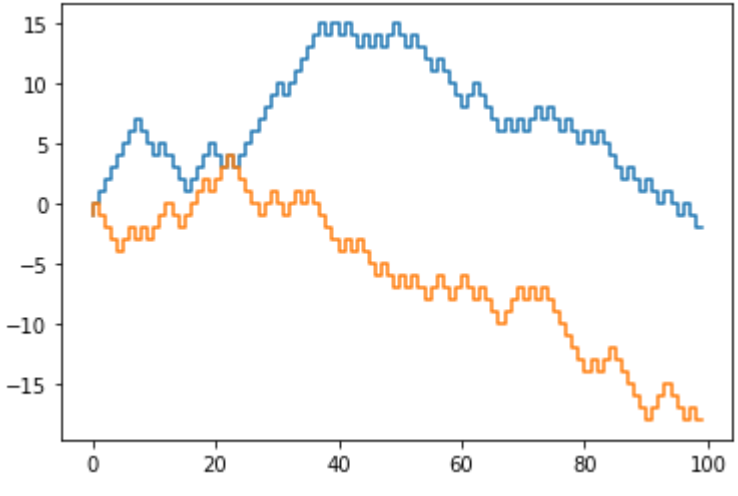
twilight

Si represento datos en una gráfica que corresponden a dos grupos diferentes, es bueno distinguirlos con colores diferentes o/y tipos de líneas diferentes, marcadores diferentes.

Por una parte, poder añadir tantos `ax.plot`, `ax.scatter` etc, como grupos:

```
import pandas as pd
import numpy as np
rng = np.random.default_rng(314)
df = pd.DataFrame({'H': 2 * rng.integers(low=0, high=2, size=100) - 1,
                   'M': 2 * rng.integers(low=0, high=2, size=100) - 1}).cumsum()
```

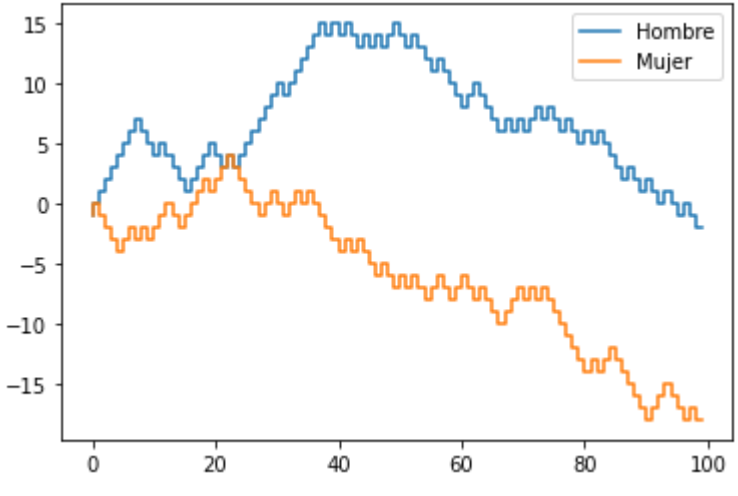
```
fig, ax = plt.subplots()
ax.step(df.index.values, df['H'])
ax.step(df.index.values, df['M']);
```



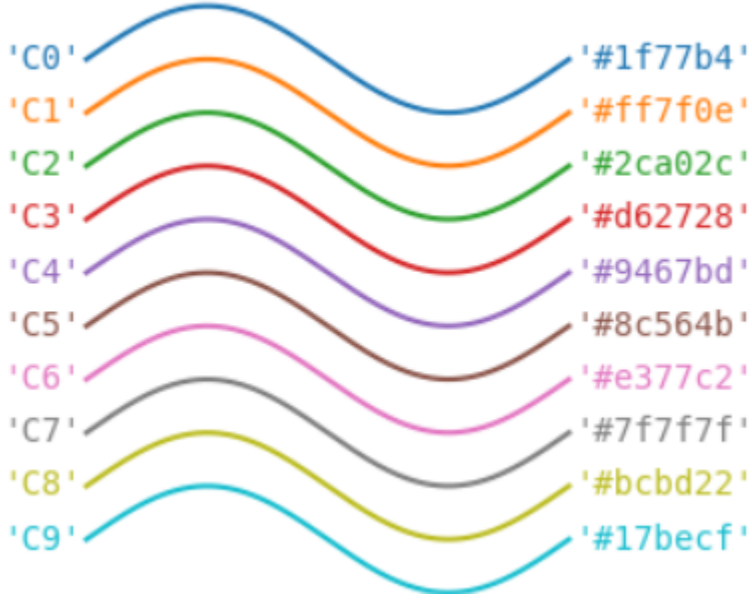
`matplotlib` se ha encargado de cambiar de color para cada `plot`.

Para que sea más informativo, puedo añadir una etiqueta a cada serie representada y generar una leyenda:

```
fig, ax = plt.subplots()
ax.step(df.index.values, df['H'], label='Hombre')
ax.step(df.index.values, df['M'], label='Mujer')
ax.legend();
```

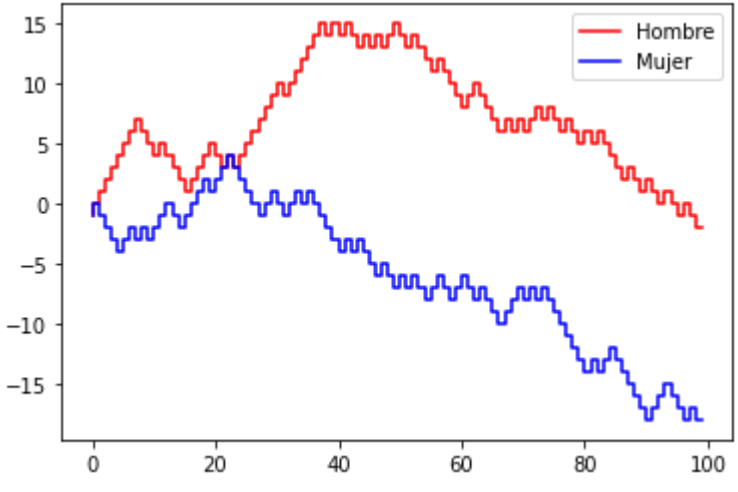


A medida que voy añadiendo elementos, `matplotlib` recorre un ciclo de colores que por defecto es el siguiente: (imagen extraida de https://matplotlib.org/3.2.1/users/dflt_style_changes.html#colors-in-default-property-cycle)

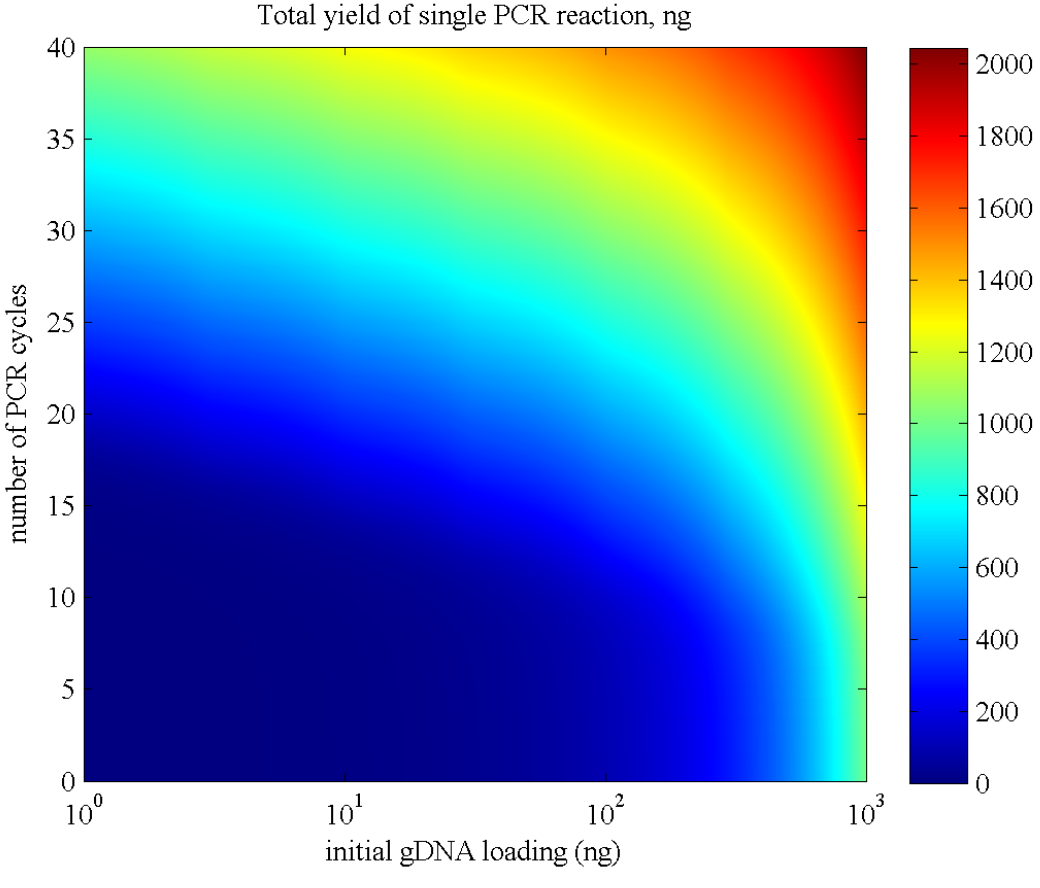


También es posible especificar el color en cada serie, por supuesto

```
fig, ax = plt.subplots()
ax.step(df.index.values, df['H'], label='Hombre', color='red')
ax.step(df.index.values, df['M'], label='Mujer', color='blue')
ax.legend();
```



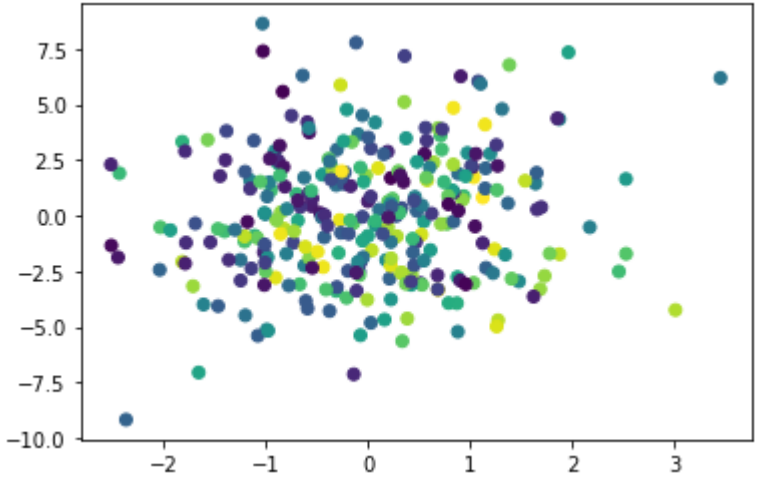
A menudo usamos el color para visualizar la variación de una variable, podéis pensar en un "heatmap" por ejemplo. (Fuente: http://www.wright.edu/~oleg.paliy/Papers/PCR_modeling/FigureS5.png)



Veamos cómo hacerlo en un scatter plot:

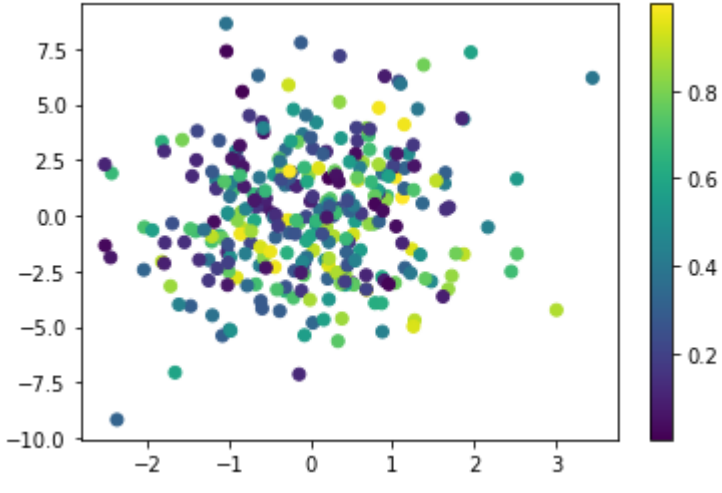
```
x = rng.standard_normal(size=300)
y = rng.standard_normal(size=300) * 3
z = rng.random(300)
```

```
fig, ax = plt.subplots()
p = ax.scatter(x, y, c=z);
```



Podemos añadir una barra de colores (`colorbar`):

```
fig, ax = plt.subplots()
p = ax.scatter(x, y, c=z)
fig.colorbar(p, ax=ax);
```



El mapa de colores que ha usado `matplotlib` por defecto es `viridis`

Ha mapeado el rango de valores que toma la variable `z` con el rango de colores de `viridis`. El mínimo de `z` está asociado con el lila oscuro, mientras que el máximo está asociado con amarillo. El color de cualquier punto en medio está interpolado.

> *Probad a cambiar el colormap, añadiendo el parámetro `cmap` en `ax.scatter`, por ejemplo, `cmap='Spectral'`*

Para usar los colores de un colormap

Hemos visto que `scatter` tenía un parámetro `cmap`, pero hay ocasiones en que queremos usar los colores de un colormap en otros contextos.

Empezamos por recuperar un colormap, por ejemplo `inferno`

```
clmap = plt.cm.get_cmap('inferno')
```

Hemos de ser conscientes de lo siguiente:

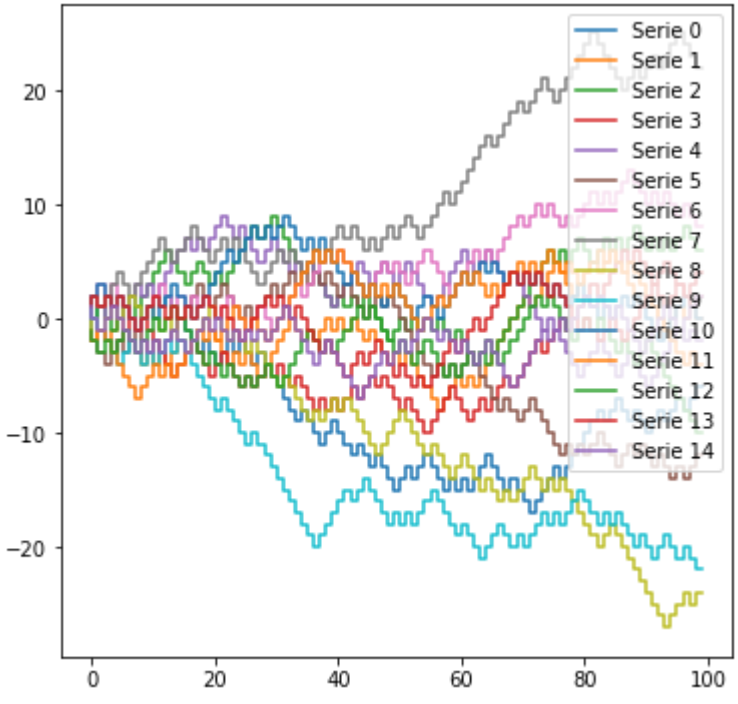
- Un colormap está basado en un número de colores. En el caso de `inferno` es 256.
- El objeto `clmap` que hemos definido realiza el mapeo se hace desde el intervalo de 0 a 1 hacia los colores del colormap.
- Si necesito un número n de colores, lo más sencillo es pasarle a `clmap` el vector de n valores repartidos uniformemente entre 0 y 1. Puedo construir este vector con `np.linspace(0, 1, n)`

Podemos usar estos colores en cualquier elemento gráfico de un `ax`. Por ejemplo, supongamos que tenemos 15 series de caminatas aleatorias que recogemos en una lista

```
series = [ np.cumsum(2 * rng.integers(low=0, high=2, size=100) - 1) for i in range(15)]
```

Vamos a representarlas, con funciones step y añadiendo una leyenda

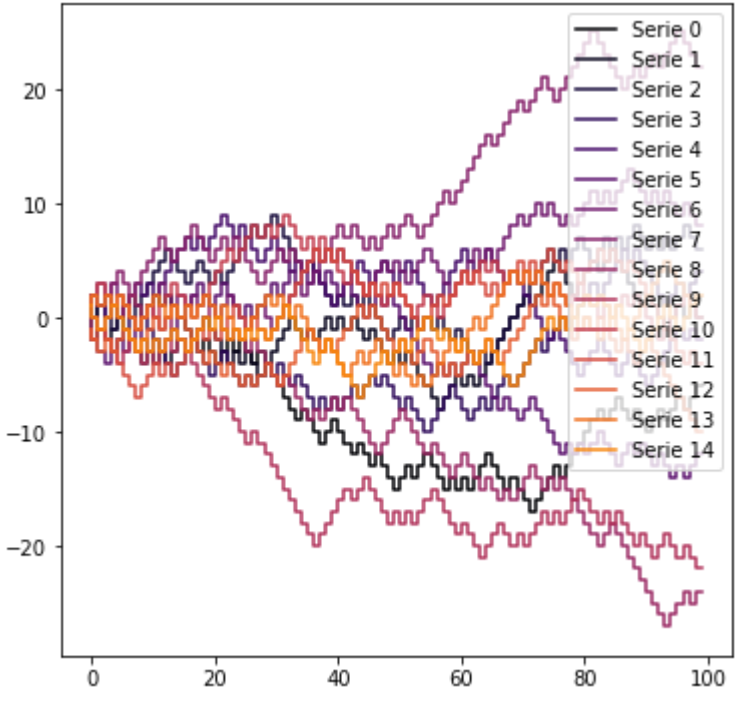
```
fig, ax = plt.subplots(figsize=(6, 6))
for i, s in enumerate(series):
    ax.step(np.arange(0, 100), s, label=f'Serie {i}')
    ax.legend()
```


Serie 0
Serie 1
Serie 2
Serie 3
Serie 4
Serie 5
Serie 6
Serie 7
Serie 8
Serie 9
Serie 10
Serie 11
Serie 12
Serie 13
Serie 14
20
10
0
−10
−20
0
20
40
60
80
100

Creamos un vector de colores, usando el colormap que hemos escogido:

```
colores = clmap(np.linspace(0, 1, 20))
```

```
fig, ax = plt.subplots(figsize=(6, 6))
for i, s in enumerate(series):
    ax.step(np.arange(0, 100), s, label=f'Serie {i}', color=colores[i])
    ax.legend()
```



En este caso, clmap está basado en los 8 primeros colores del colormap 'Spectral'.