01MIAR - Representación Gráfica

Luis Pilaguano

Representacion grafica

- matplotlib basico (plot, legend, decoradores y anotaciones, grabar)
- dibujo con pandas (tipos de graficos)
- intro a seaborn https://seaborn.pydata.org/tutorial.html

Referencias

- https://plotdb.com/
- https://plotnine.org/
- https://d3js.org/
- https://plot.ly/python/
- https://bokeh.org/
- https://shiny.rstudio.com/gallery/
- https://www.tableau.com/es-es

matplotlib

- Librería estándar de representación gráfica
- Low level: fácil de usar, difícil de dominar
- Control
- Gráficos y diagramas

Elementos de visualización

- Gráfico. Tipo de representación dentro de una subfigura o figura. Existen gran cantidad de tipos de gráficos: barras, líneas, tarta...
- Subfigura. Representación dentro de una figura.
- Figura. Marco donde se representan de manera conjunta una visualización. Se caracterizan por tener un título general y suele tener una descripción así como una fuente o referencia

In [3]:
```
# importar matplotlib
import matplotlib.pyplot as plt
```

In [4]:
```
import numpy as np
import pandas as pd
```

- 'plot(data)' para crear un gráfico con 'data'
- 'show()' para mostrarlo

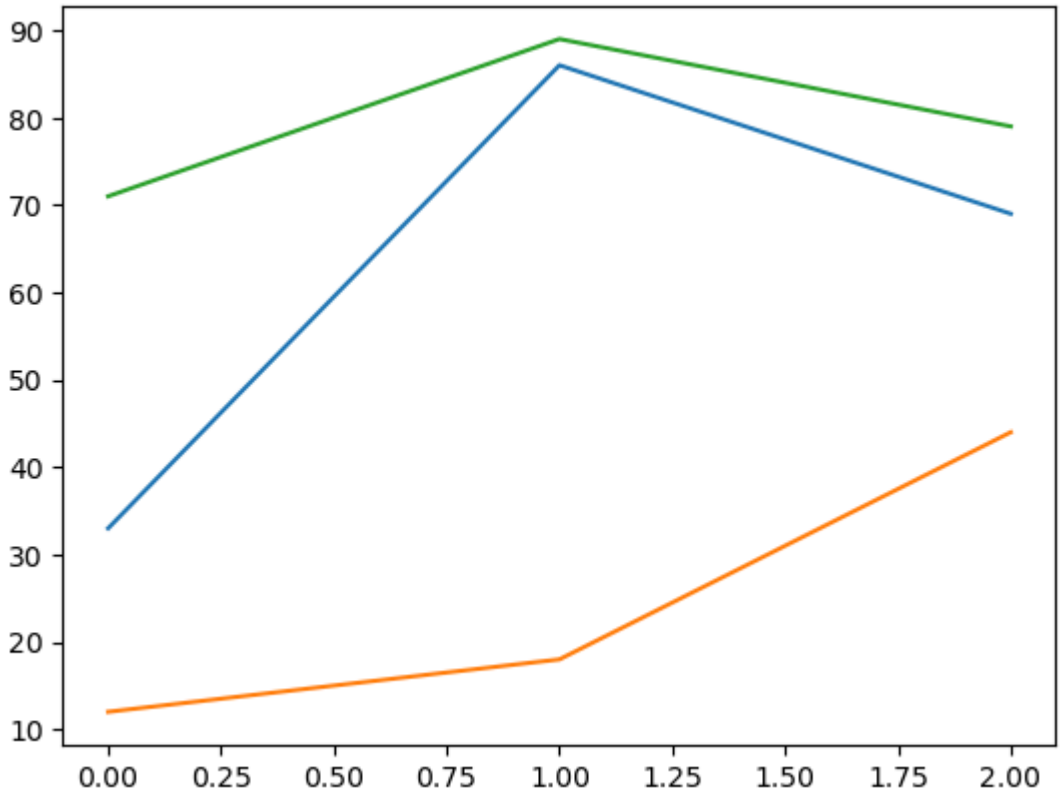
In [5]:
```
%matplotlib --list
```

```
Available matplotlib backends: ['agg', 'auto', 'cairo', 'gtk3', 'gtk3agg', 'gtk3c
airo', 'gtk4', 'gtk4agg', 'gtk4cairo', 'inline', 'macosx', 'nbagg', 'notebook',
'osx', 'pdf', 'pgf', 'ps', 'qt', 'qt5', 'qt5agg', 'qt5cairo', 'qt6', 'qtagg', 'qt
cairo', 'svg', 'template', 'tk', 'tkagg', 'tkcairo', 'webagg', 'wx', 'wx', 'wxag
g', 'wxcairo']
```

In [6]:
```
# magic command para visualizar gráficos en jupyter en modo no interactivo
%matplotlib inline
```

In [7]:
```
data = np.random.randint(100,size=(3,3))
display(data)
plt.plot(data) # creación del gráfico
plt.show() # comando que muestra los gráficos creados hasta el momento
```

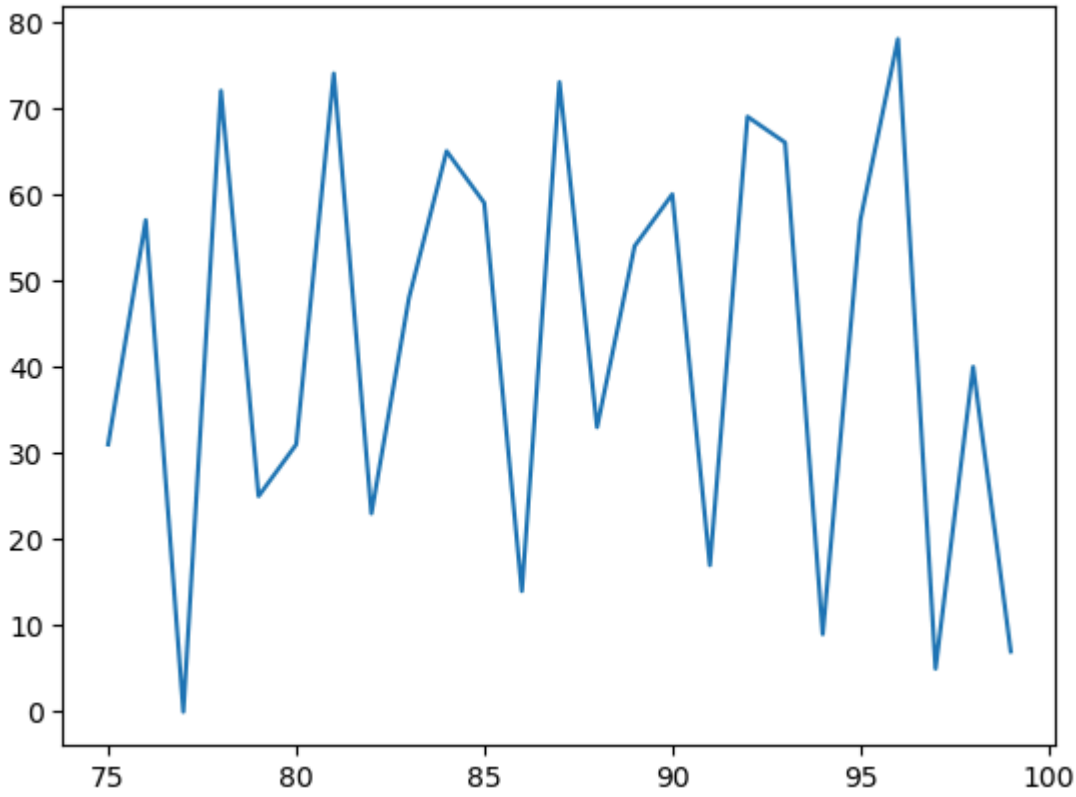
```
array([[33, 12, 71],
       [86, 18, 89],
       [69, 44, 79]], dtype=int32)
```



In [1]:
```
# activar modo interactivo
%matplotlib notebook
# plt.ioff() # evitar que se sobreescriban los gráficos
# %matplotlib inline
```

In [8]:
```
# Se pueden especificar x e y por separado
x_data = np.arange(75, 100) # dimensiones!
y_data = np.random.randint(80,size=(25))

print(x_data,y_data)

if x_data.size == y_data.size:
    plt.plot(x_data, y_data)
    plt.show()
else:
    print(f"{x_data.size} no tienen las mismas dimensiones {y_data.size}")
```

```
[75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98
 99] [31 57  0 72 25 31 74 23 48 65 59 14 73 33 54 60 17 69 66  9 57 78  5 40
  7]
```



figure, axes

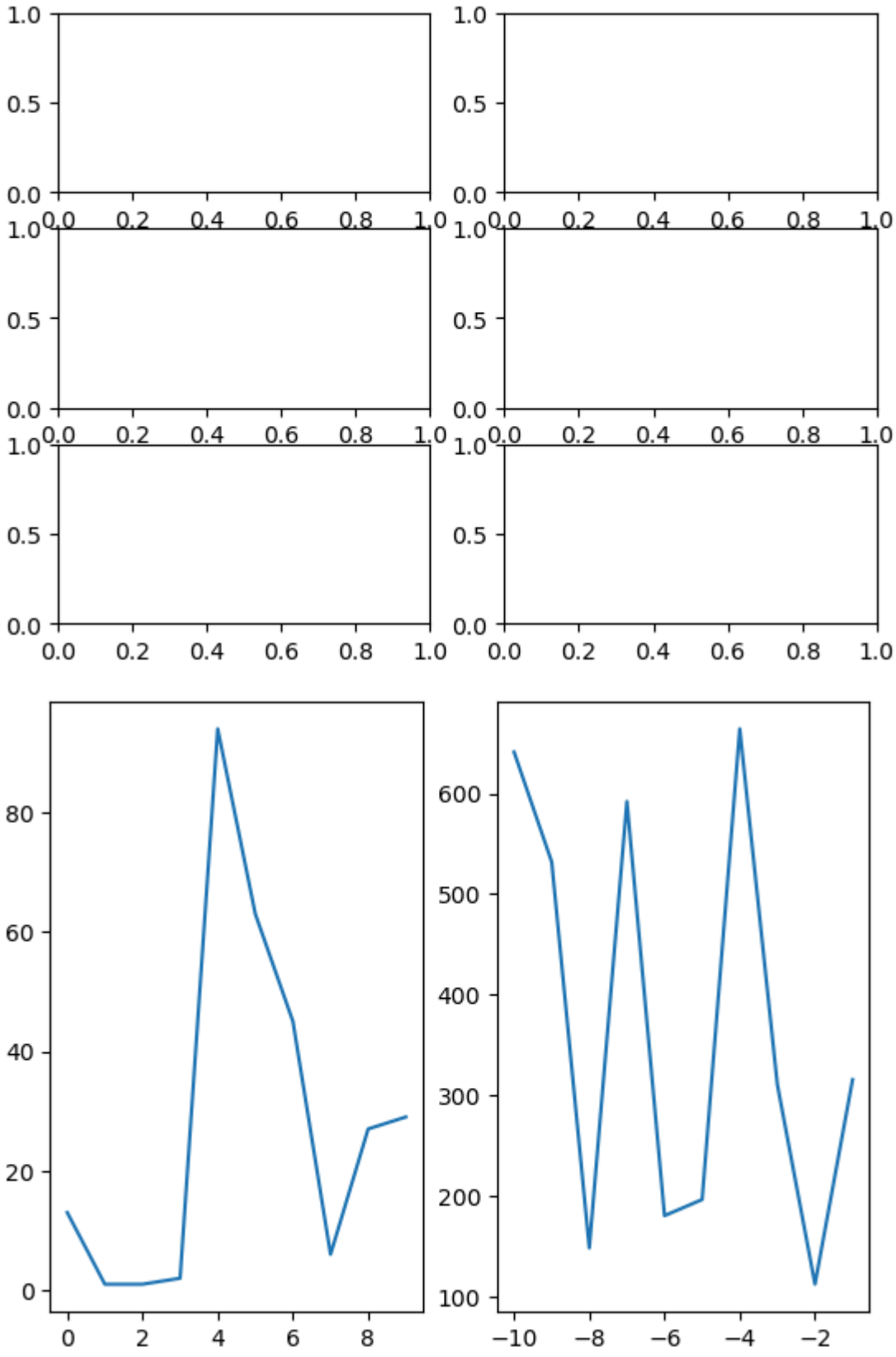
- Figure: objeto en el que residen todos los gráficos (uno o múltiples)
- Axes: cada uno de los subplots (o gráficos) dentro de Figure

In [9]:
```
fig, axes = plt.subplots(3,2) # crea una ndarray de 1x2 axes (o subplots)
print(type(axes))
print(axes.shape)
```

```
<class 'numpy.ndarray'>
(3, 2)
```

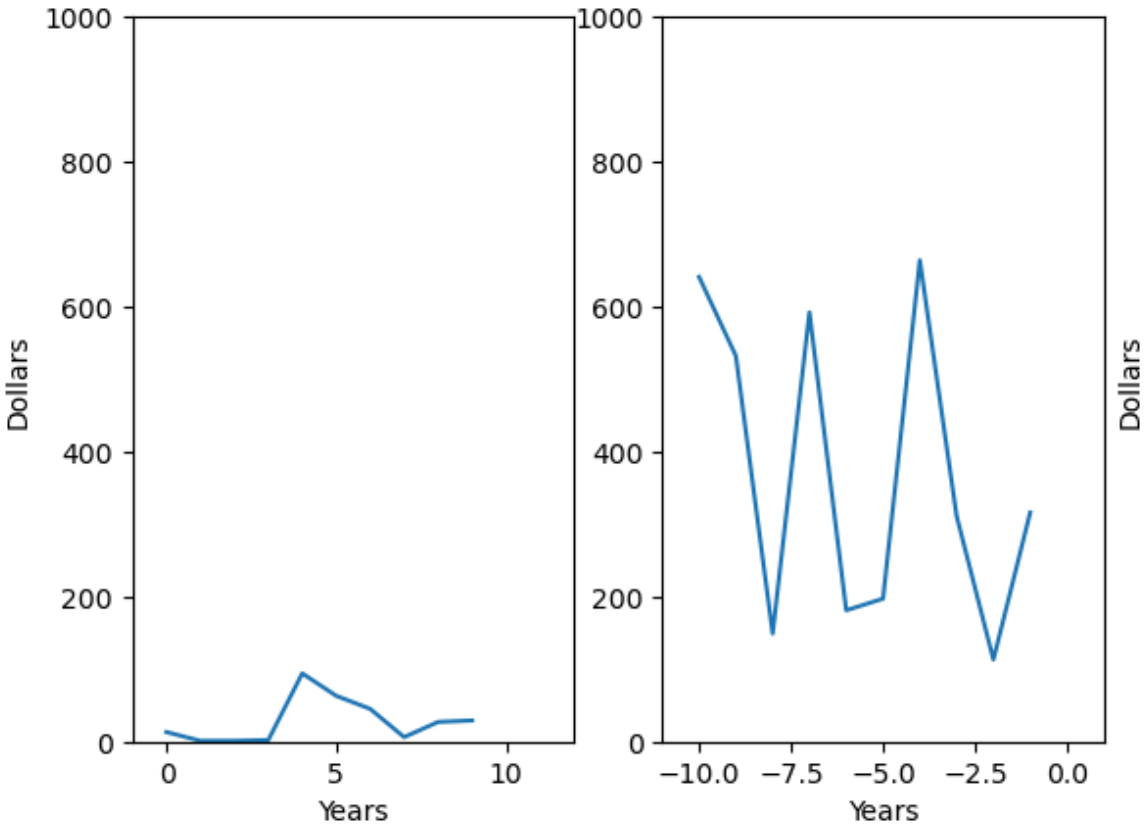
In [10]:
```
# Generar los datos
x_data1 = np.arange(0,10)
x_data2 = np.arange(-10,0)
y_data1 = np.random.randint(100,size=(10))
y_data2 = np.random.randint(1000,size=(10))
```

In [11]:
```
# Crear la figura y los subplots
fig, axes = plt.subplots(1,2)
# preparar cada subplot
axes[0].plot(x_data1,y_data1)
axes[1].plot(x_data2,y_data2)

#mostrar ambos axes
plt.show() # Recomendable usar plt y no fig.show()
```



In [ ]:
```python
# crear un nuevo eje con la x compartida
fig, ax = plt.subplots(1, 1)
x = np.arange(12)
ax.plot(x, "r")

ax2 = ax.twinx()
ax2.plot(x ** 2, "g")

plt.show()
```

In [12]:
```
# Crear la figura y los subplots
fig, axes = plt.subplots(1,2)
# preparar cada subplot
axes[0].plot(x_data1,y_data1)
axes[1].plot(x_data2,y_data2)

# tanto axes como fig pueden configurarse a través de funciones
axes[0].set_ylim([0,1000])
axes[0].set_xlim([-1,12])

axes[0].set_xlabel("Years")
axes[0].set_ylabel("Dollars")

axes[1].set_ylim([0,1000])
axes[1].set_xlim([-11,1])

axes[1].set_xlabel("Years")
axes[1].set_ylabel("Dollars", loc="center", labelpad=-210)

plt.show()
```



Más métodos

- Documentación general de la API

.plot()

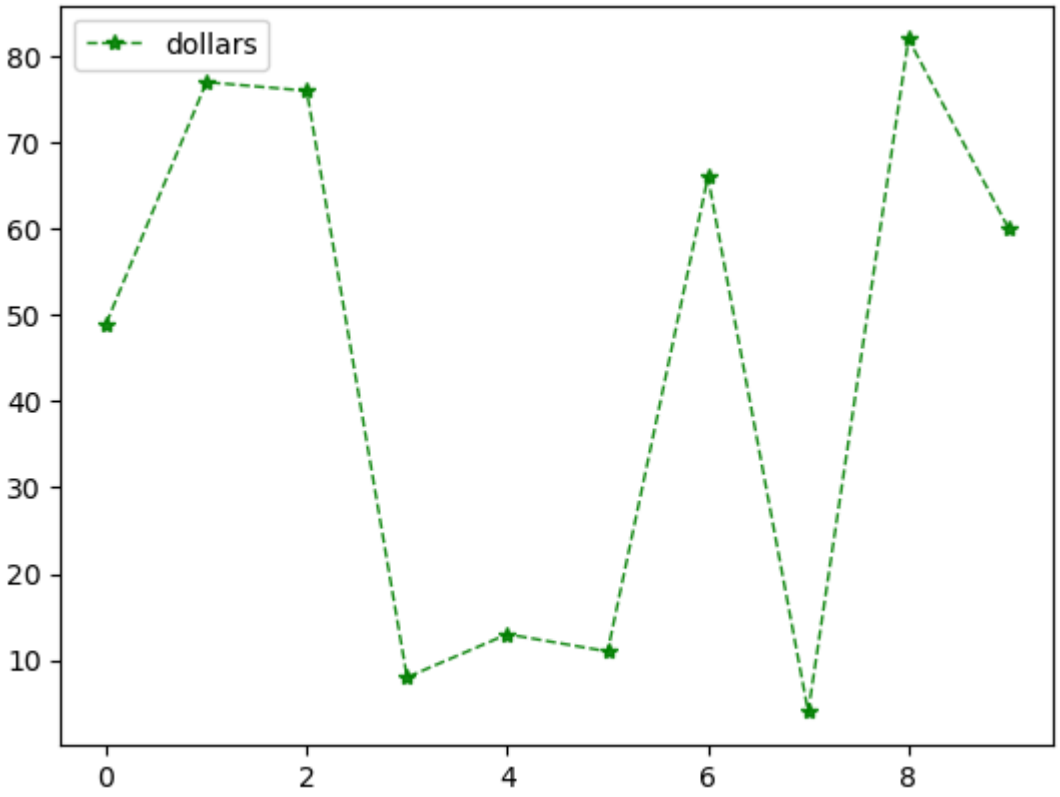
Múltiples parámetros

```
plt.plot(x, y, style, label, linewidth=1, alpha=1)
```

- 'style': cadena de texto con formato (color, tipo...) [color][marker][line]
- 'label': referencia para leyenda
- 'linewidth': brocha
- 'alpha': transparencia

Otros: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

In [13]:
```
y_data = np.random.randint(100,size=(10))
x_data = np.arange(10)

fig,axis = plt.subplots(1,1)
axis.plot(x_data,y_data,'g*--',label='dollars',linewidth=1,alpha=0.9) # creación
axis.legend(loc='best') # representar la leyenda
plt.show()
```



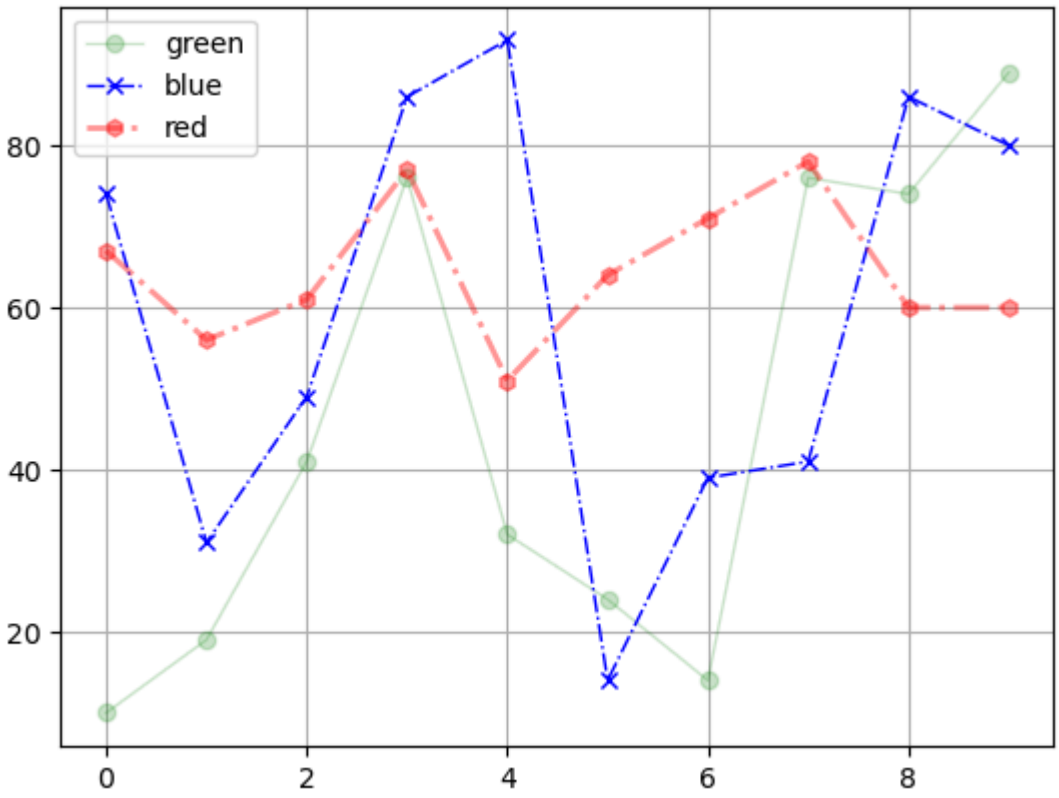
Color	Descripción
b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black
w	white

| Marcador | Descripción | |:------|:----------| | . | point marker | | , | pixel marker | | o | circle marker | | v | triangle_down marker | | ^ | triangle_up marker | | < | triangle_left marker | | > | triangle_right marker | | 1 | tri_down marker | | 2 | tri_up marker | | 3 | tri_left marker | | 4 | tri_right marker | | s | square marker | | p | pentagon marker | | * | star marker | | h | hexagon1 marker | | H | hexagon2 marker | | + | plus marker | | x | x marker | | D | diamond marker | | d | thin_diamond marker | | | vline marker | | _ | hline marker |

Línea	Descripción
-	solid line style
--	dashed line style
-.	dash-dot line style
:	dotted line style

Representar más de un gráfico en una subfigura

In [14]:
```
x_data = np.arange(10)
fig,axis = plt.subplots(1,1)
axis.plot(x_data,np.random.randint(100,size=(10)),'go-',label='green',linewidth=
axis.plot(x_data,np.random.randint(100,size=(10)),'bx-.',label='blue',linewidth=
axis.plot(x_data,np.random.randint(50,80,size=(10)),'rh-.',label='red',linewidth
axis.legend(loc='best')
plt.grid()
plt.show()
```



Decoradores

Cambian el aspecto general de la figura

```
axes.set_xticks(<lista>) #etiquetas del eje
axes.set_title('titulo') #titulo
axes.set_ylabel('label')
axes.set_frame_on(bool)
```
Más métodos

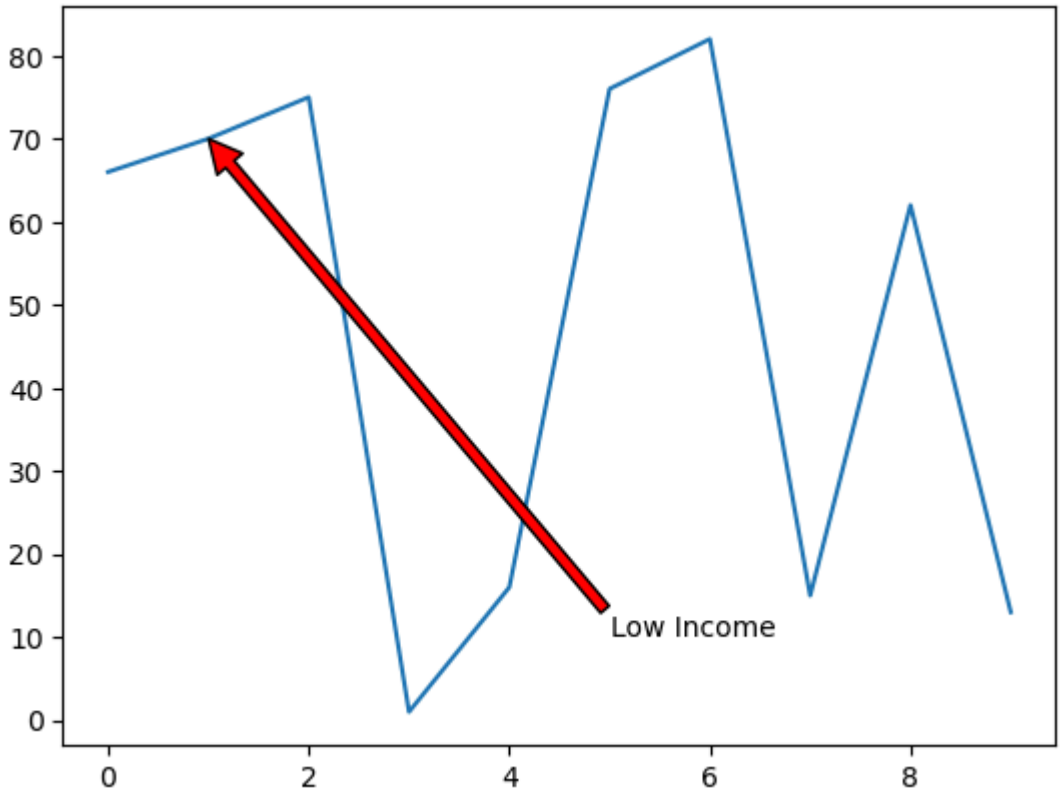
Anotaciones

Formas de añadir información sobre el gráfico

```
axes.text(x,y,'texto')
axes.annotate(texto, xy, xytext, xycoords, arrowprops)
```
Más métodos

In [15]:
```
y_data = np.random.randint(100, size=(10)) # Cuidado con las dimensiones absolut
x_data = np.arange(10)
```

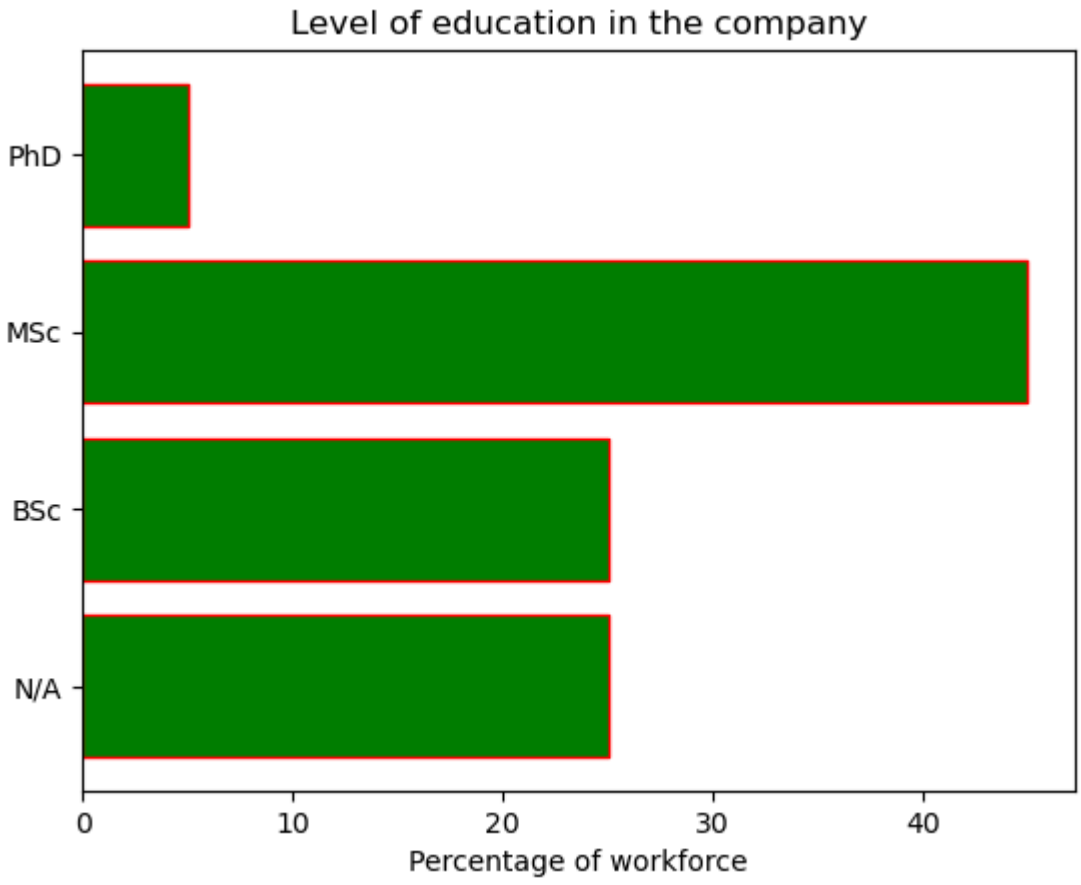
In [16]:
```
fig,axis = plt.subplots(1,1)
axis.plot(x_data,y_data)
# axis.text(x_data[1],y_data[1],'Low income') # coordenadas igual a los datos
axis.annotate('Low Income', xy=(x_data[1],y_data[1]), xytext=(5, 10), arrowprops
plt.show()
```



Tipos de gráficos

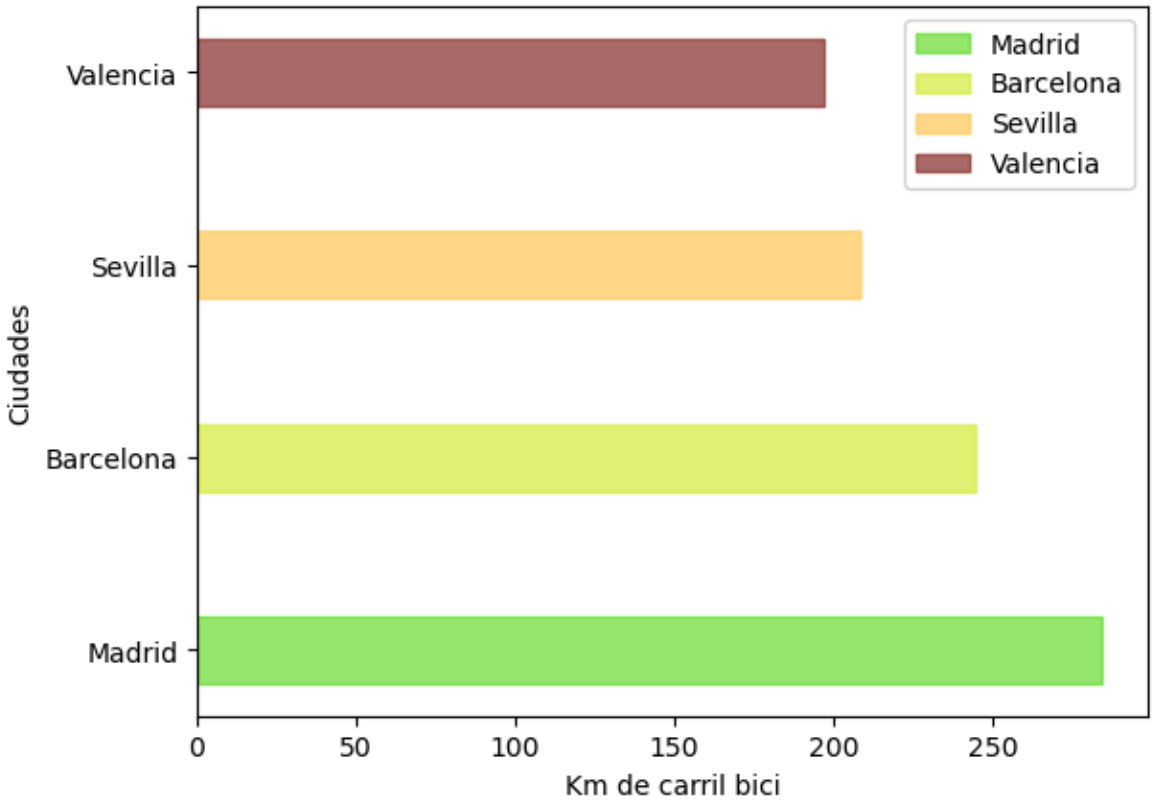
Ejemplos disponibles como referencia
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html

In [17]:
```
# Horizontal bar
fig, ax = plt.subplots()
# Example data
people = ('PhD', 'MSc', 'BSc', 'N/A')
y_pos = np.arange(len(people))
percentage = [5,45,25,25]

# print(y_pos)
ax.barh(y_pos, percentage, align='center',
        color='green', edgecolor='red')
ax.set_yticks(y_pos)
ax.set_yticklabels(people)
ax.invert_yaxis()  # labels read top-to-bottom
ax.set_xlabel('Percentage of workforce')
ax.set_title('Level of education in the company')
plt.show()
```



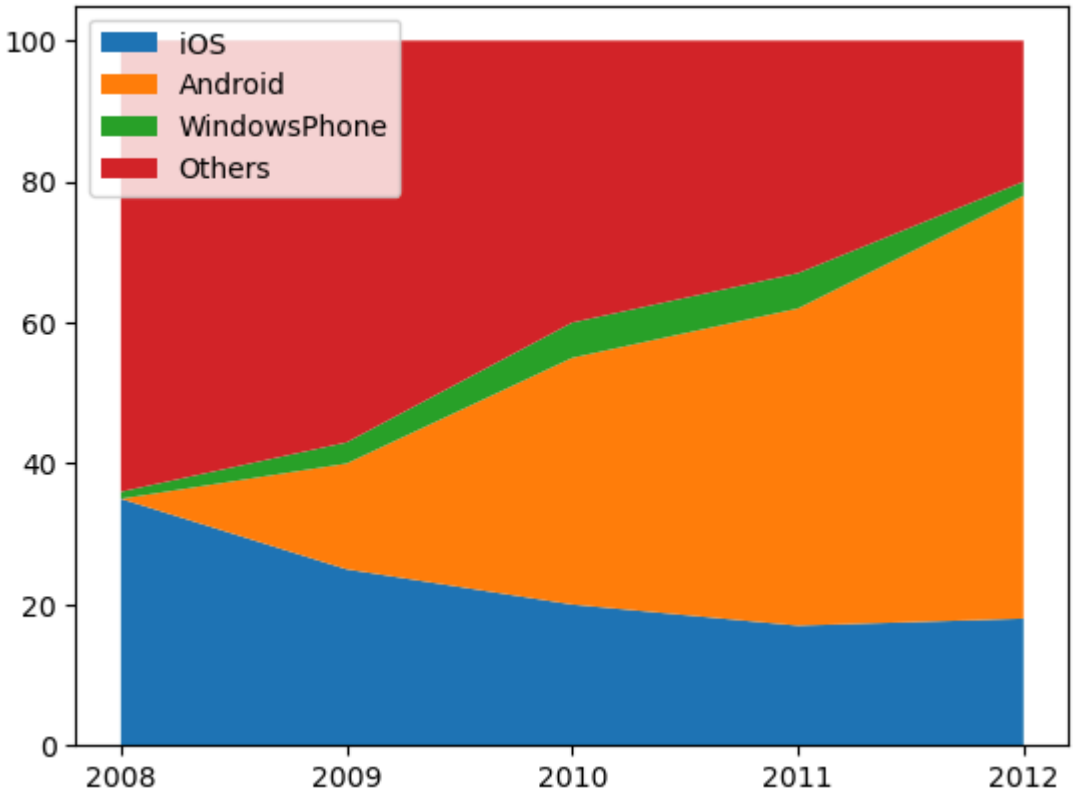
In [18]:
```
ls_count = (284487, 244560, 208493, 196764)
cities = ('Madrid', 'Barcelona', 'Sevilla', 'Valencia')
data = np.array(ls_count)/1000
y_pos = np.arange(data.size)

width = 0.35       # the width of the bars
opacity = 0.7

fig, ax = plt.subplots()
rects1 = ax.barh(y_pos, data, width, alpha=opacity)

rects1[3].set_color('#872f29')
rects1[2].set_color('#ffcb59')
rects1[1].set_color('#d6ee39')
rects1[0].set_color('#6ade30')
```

```
ax.set_yticks(y_pos)
ax.set_yticklabels(cities)

# add some text for labels, title and axes ticks
ax.set_ylabel('Ciudades')
ax.set_xlabel('Km de carril bici')

ax.legend((rects1[0], rects1[1], rects1[2], rects1[3]), cities)

plt.show()
```

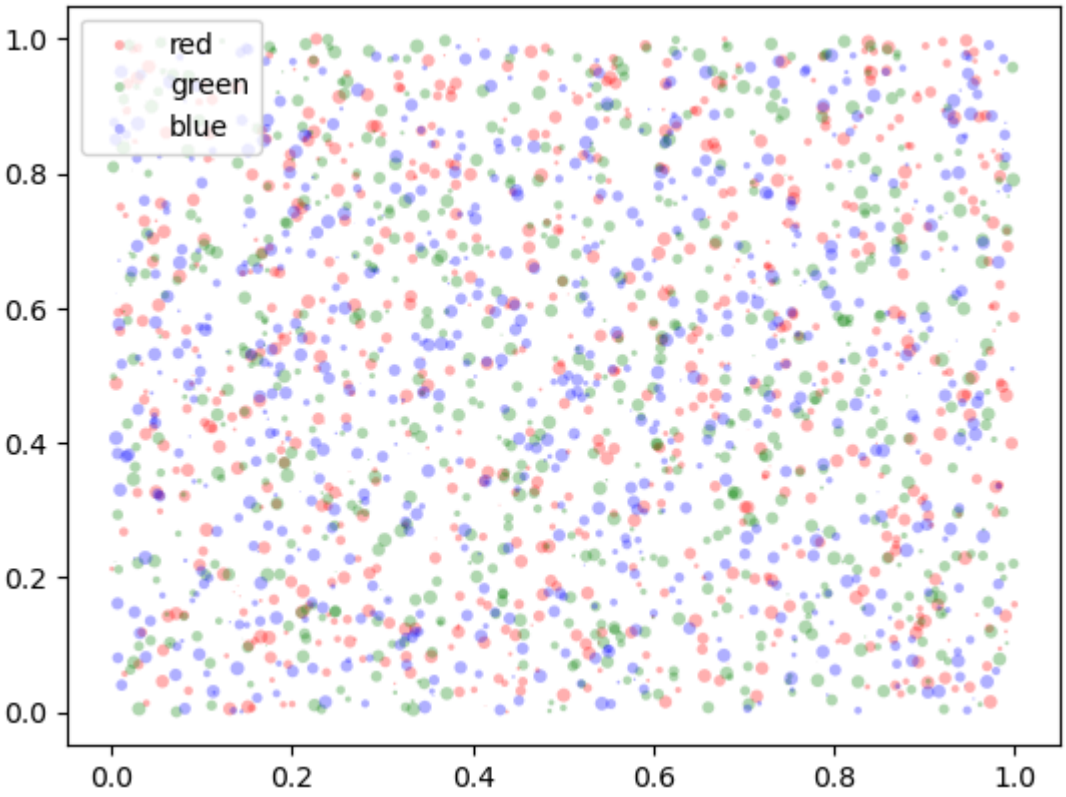


In [19]:

```
# stack plot
x = [2008, 2009, 2010, 2011, 2012]
iOS = [35, 25, 20, 17, 18]
Android = [0, 15, 35, 45, 60]
WindowsPhone = [1, 3, 5, 5, 2]
Others = [64,57,40,33,20]

# y = np.vstack([iOS, Android, WindowsPhone,Others])
# print(y)

labels = ["iOS ", "Android", "WindowsPhone", "Others"]

fig, ax = plt.subplots()
ax.stackplot(x, iOS, Android, WindowsPhone, Others,labels=labels)
# ax.stackplot(x, y, labels=labels)
ax.set_xticks(x)
ax.legend(loc='upper left')
plt.show()
```



In [20]:
```
# Scatter plot
from numpy.random import rand

fig, ax = plt.subplots()
n = 700
for color in ['red', 'green', 'blue']:
    x, y = rand(2, n)
    scale = 25.0 * rand(n)
    ax.scatter(x, y, c=color, s=scale, label=color,
               alpha=0.3, edgecolors='none')

ax.legend()
plt.show()
```



Grabar a archivo

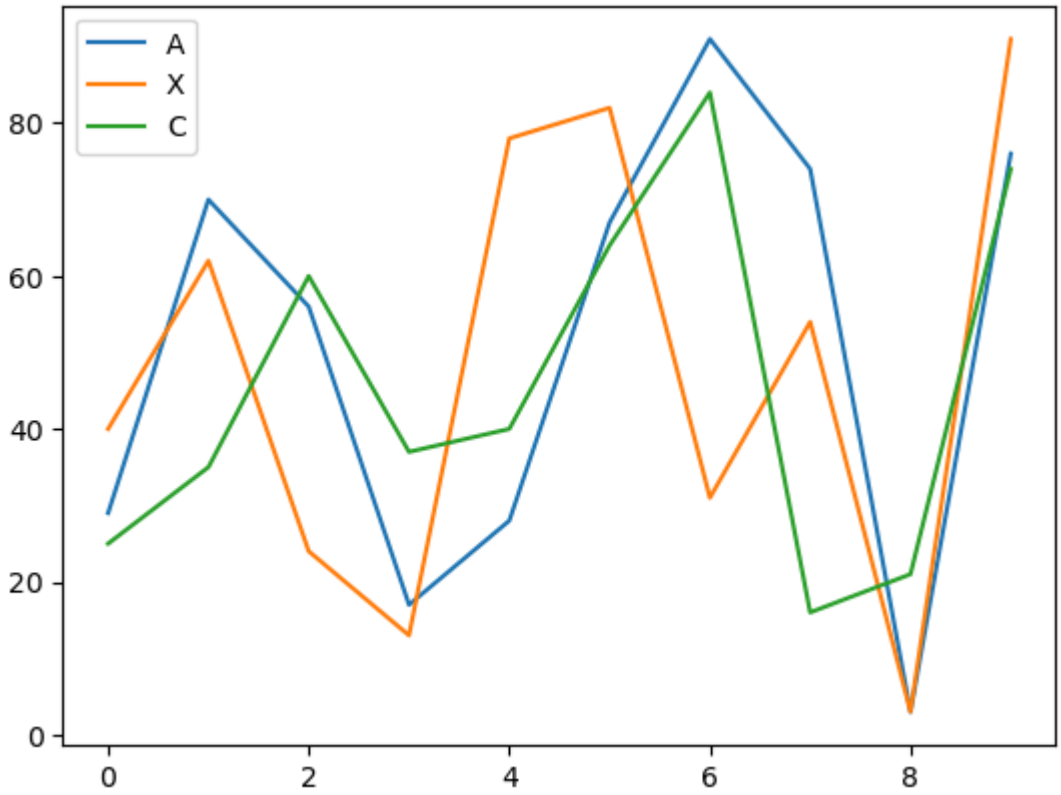
```
figure.savefig(<filename>,dpi=None, facecolor='w', edgecolor='w',
format=None, bbox_inches=None)
```

In [21]:
```
# guardar la figura (puede ser múltiples gráficos)
import os
ruta = os.path.join("res", "o_figure.png")

fig.savefig(ruta, format='png') # varios formatos aceptados ('pdf',' png', 'svg'
```

Representación gráfica en pandas

- pandas ofrece interfaz para dibujar Series y DataFrame
- Usa matplotlib internamente
- Fácil de usar

In [22]:
```
rand_matrix = np.random.randint(100,size=(10,3))
frame = pd.DataFrame(rand_matrix, columns=list('AXC'))
frame.plot()
plt.show() # aún es necesario llamar a show() para mostrar los gráficos  //TODO

display(frame) # datos en eje x, eje y, leyenda, series?
```



	A	X	C
0	29	40	25
1	70	62	35
2	56	24	60
3	17	13	37
4	28	78	40
5	67	82	64
6	91	31	84
7	74	54	16
8	3	3	21
9	76	91	74

frame.plot()

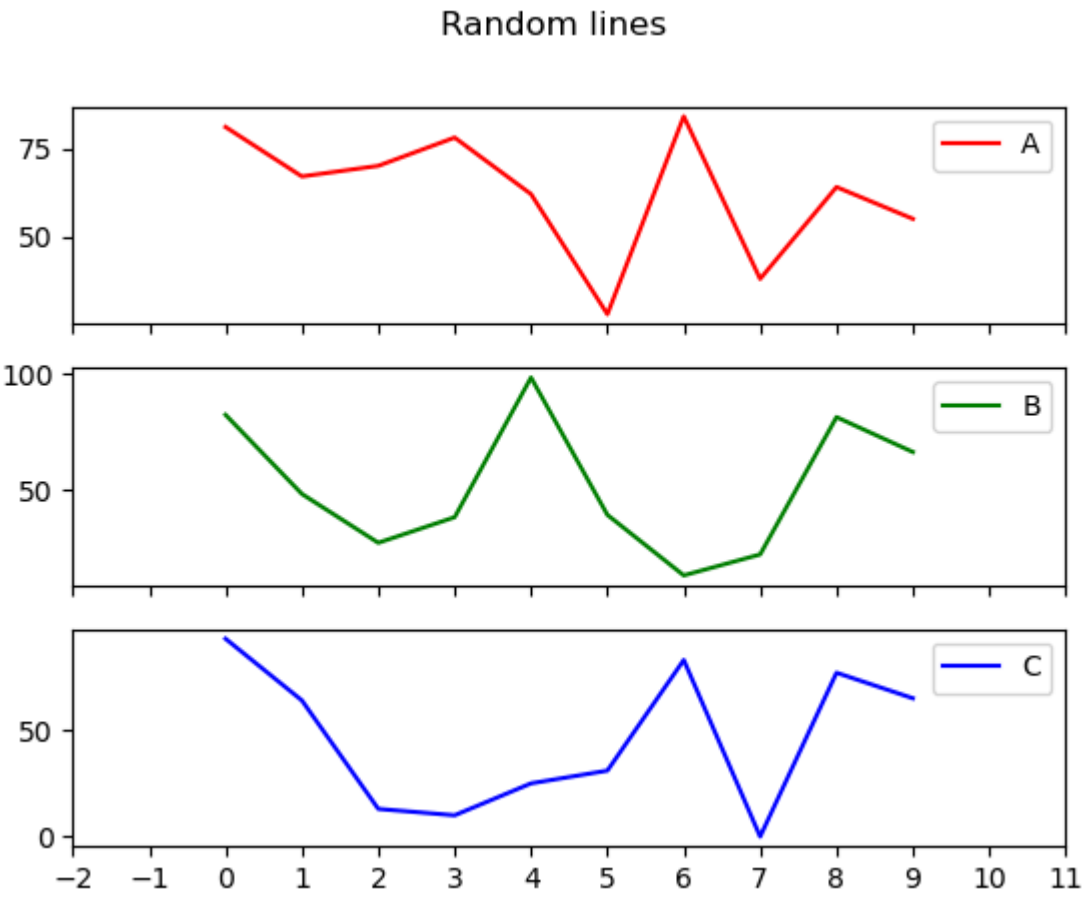
```
frame.plot(x,y,label=
<Series>,ax,style,kind,xticks,yticks,title,subplots)
```

- (solo para Series) label: referencia para la leyenda
- ax: subfigura en la que dibujar los datos
- style: estilo de la línea
- kind: tipo de gráfico ('bar', 'pie', 'hist', 'area' 'line', 'barh', 'density', 'kde').
- xticks y yticks: valores en los ejes X e Y
- title: string como título

- (solo para DataFrame) subplots: bool si se desean subfiguras separadas para cada columna.
- (solo para DataFrame) 'x' e 'y' se pueden utilizar para seleccionar columnas para cada eje

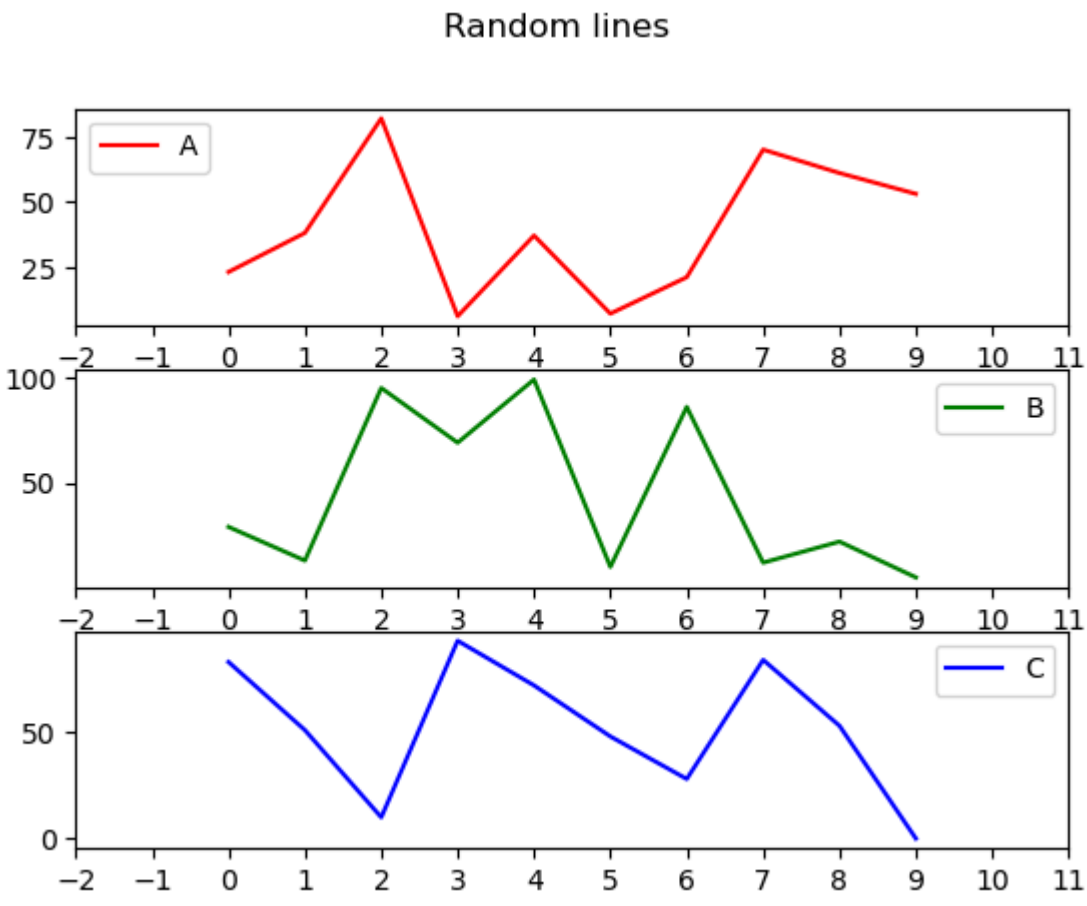
Lista completa de argumentos

In [23]:
```
rand_matrix = np.random.randint(100, size=(10,3))
frame = pd.DataFrame(rand_matrix, columns=list('ABC'))
frame.plot(style=['r-','g-','b-'], xticks=range(-2,12), title='Random lines', su
plt.show()
display(frame)
```

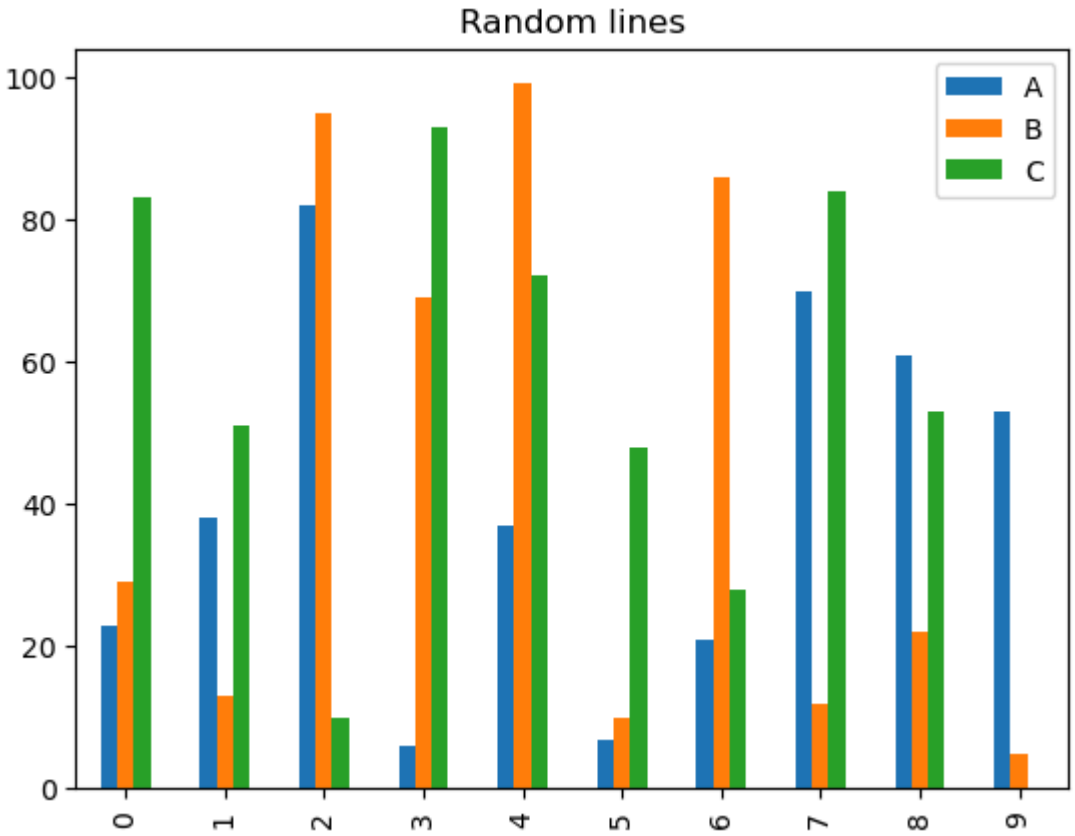


	A	B	C
0	81	82	93
1	67	48	64
2	70	27	13
3	78	38	10
4	62	98	25
5	28	39	31
6	84	13	83
7	38	22	0
8	64	81	77
9	55	66	65

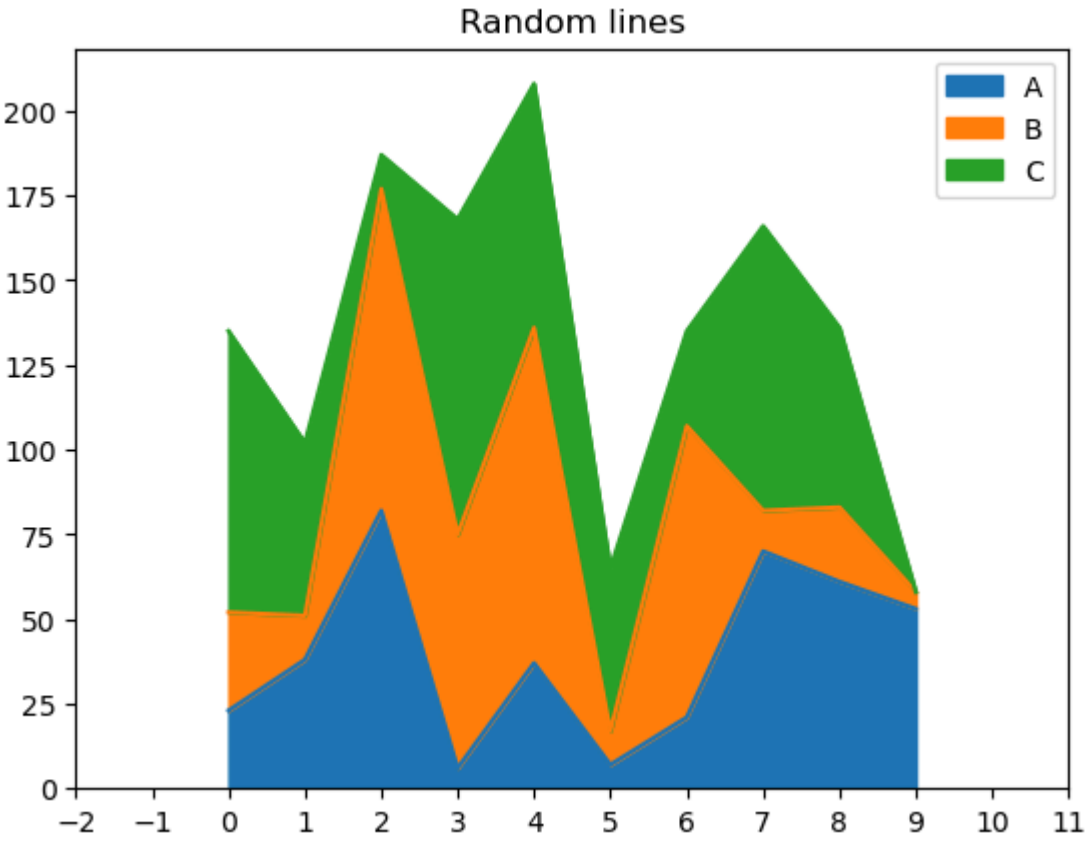
In [24]:
```
# combinar pandas plot con matplotlib
rand_matrix = np.random.randint(100, size=(10,3))
frame = pd.DataFrame(rand_matrix, columns=list('ABC'))

fig, axis = plt.subplots(3,1)
ax = frame.plot(style=['r-','g-','b-'], xticks=range(-2,12), title='Random lines

ax[0].legend(loc='upper left')

plt.show()
```



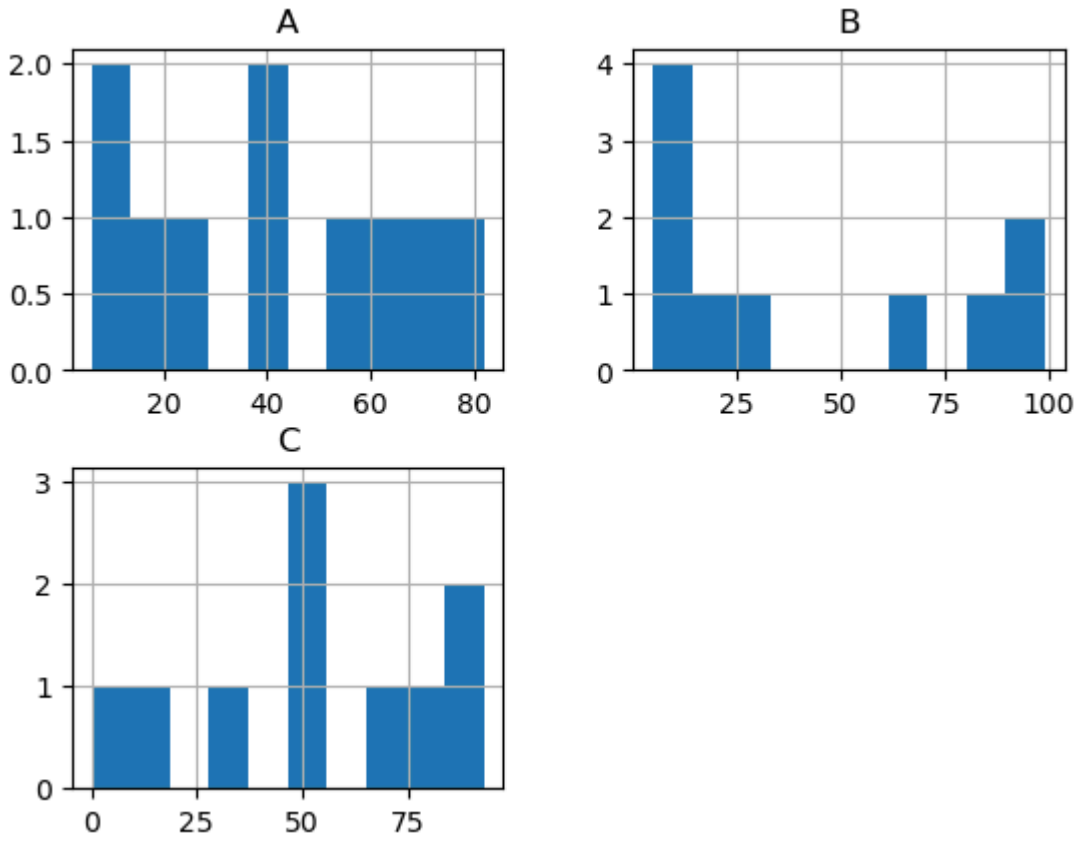
In [25]:
```
# Gráfico de barras
frame.plot(kind='bar', title='Random lines')
plt.show()
```



In [26]:
```
# Gráfico de área
frame.plot(kind='area', xticks=range(-2,12), title='Random lines')
plt.show()
```

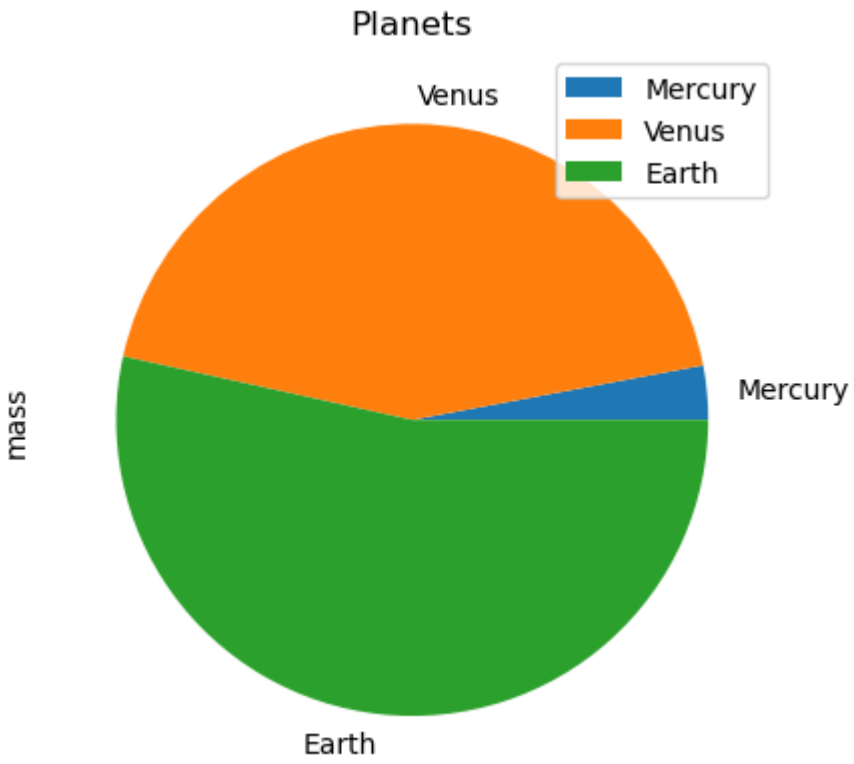


In [27]:
```
# Histogramas
frame.hist()
plt.show()
```



In [28]:
```
# Gráfico de pastel
df = pd.DataFrame({'mass': [0.330, 4.87 , 5.97],
                   'radius': [2439.7, 6051.8, 6378.1]},
                  index=['Mercury', 'Venus', 'Earth'])
display(df)
df.plot(y='mass', kind='pie', title='Planets') # y variable a usar
plt.show()
```

	mass	radius
Mercury	0.33	2439.7
Venus	4.87	6051.8
Earth	5.97	6378.1



Seaborn

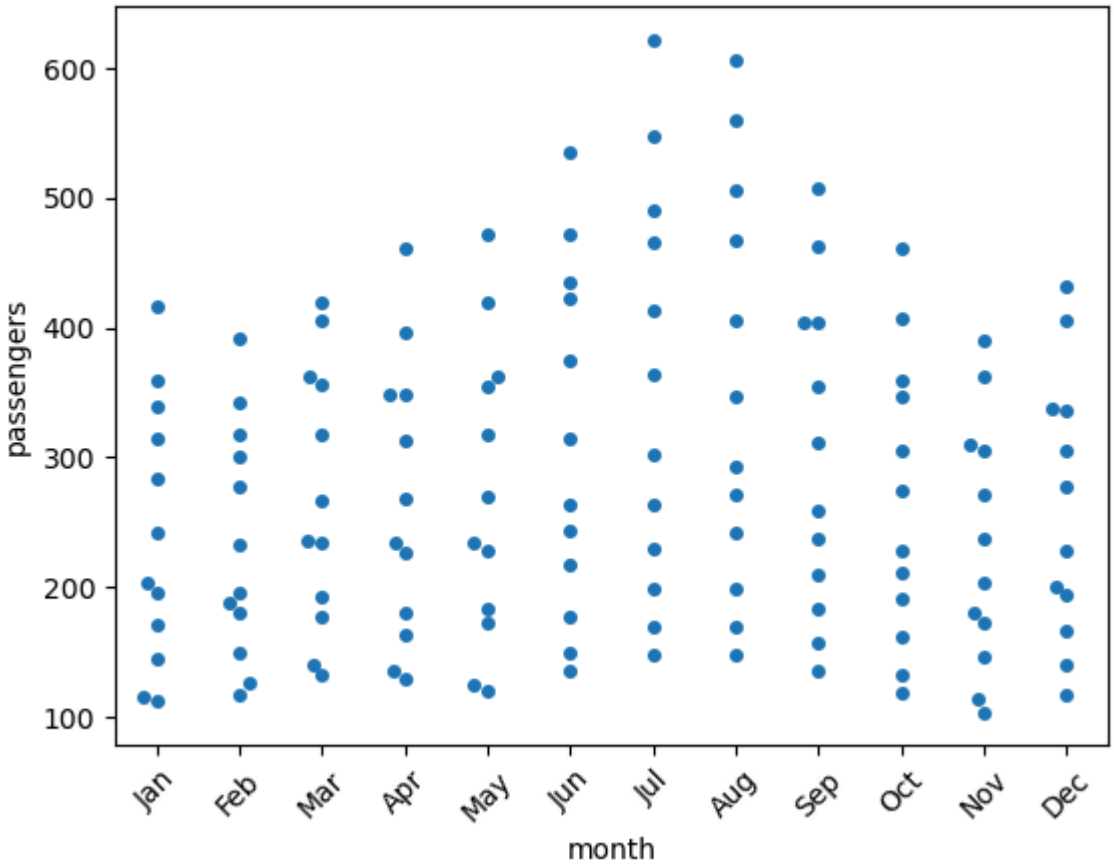
- Abstracción de matplotlib
- Facilidad de uso y personalización
- Excelente para exploración y visualización de relaciones entre variables

Viene con varios datasets predefinidos https://github.com/mwaskom/seaborn-data

Documentación oficial

In [34]:
```
import seaborn as sns
```

In [35]:
```
# Load data
flights_data = sns.load_dataset("flights")
# display(flights_data)

# Construct plot
sns.swarmplot(x="month", y="passengers", data=flights_data)
# Show plot
plt.xticks(rotation=45)
plt.show()
```



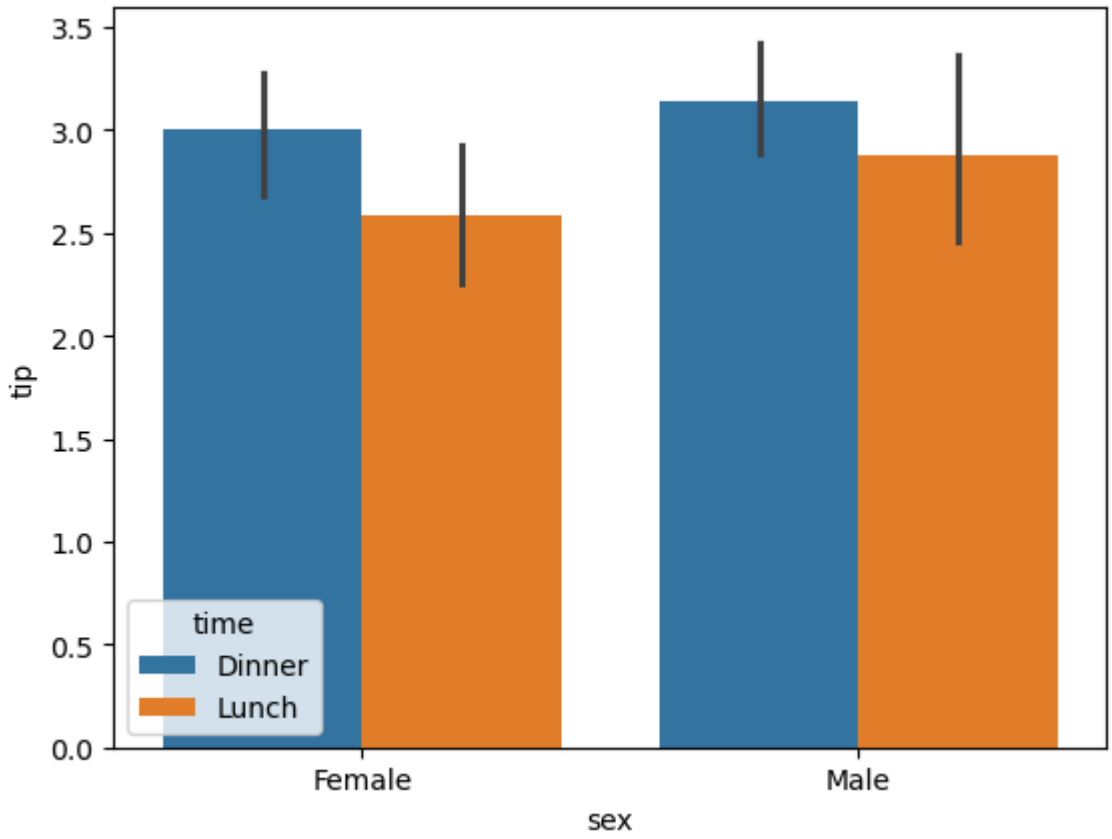
Compatible con pandas

In [36]:
```
# Load in data
tips = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/maste
tips.sample(5)
```

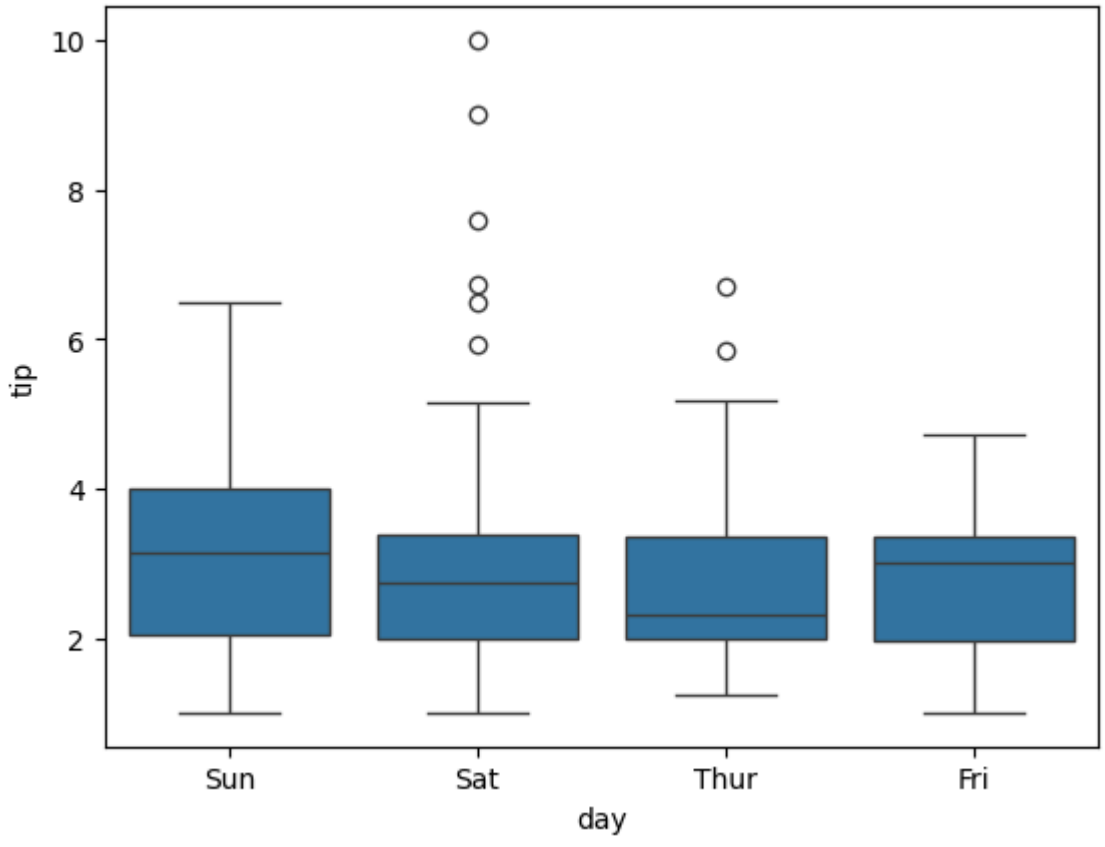
Out[36]:

	total_bill	tip	sex	smoker	day	time	size
185	20.69	5.00	Male	No	Sun	Dinner	5
116	29.93	5.07	Male	No	Sun	Dinner	4
52	34.81	5.20	Female	No	Sun	Dinner	4
42	13.94	3.06	Male	No	Sun	Dinner	2
153	24.55	2.00	Male	No	Sun	Dinner	4

In [37]:
```
sns.barplot(x='sex', y='tip', hue='time', data=tips) # palette para color (negro
plt.show()
```



In [38]:
```
# Boxplot (cuartiles)
sns.boxplot(x='day', y='tip', data=tips)
# sns.boxplot(x=tips['day'], y=tips.tip)
plt.show()
```



Relaciones entre variables

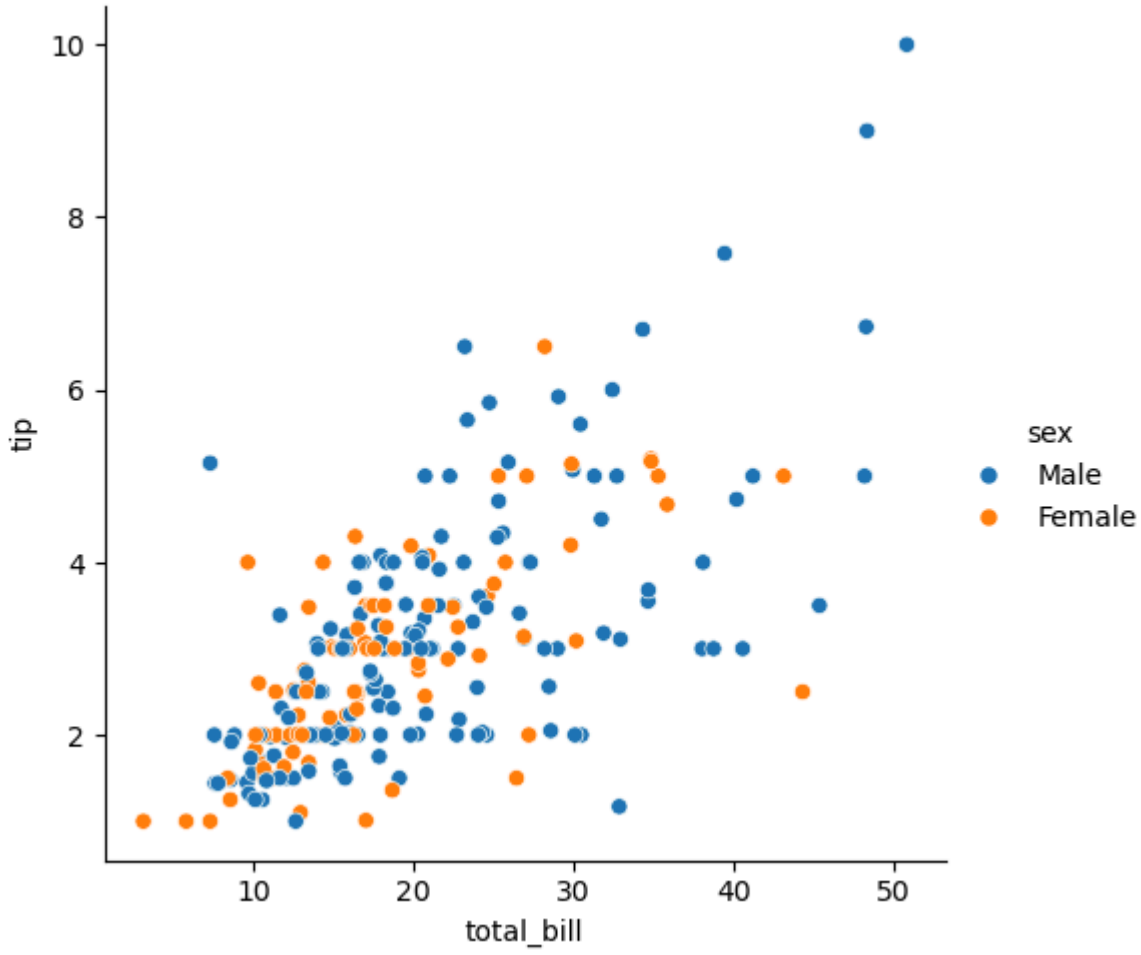
```
sns.relplot()
```

In [39]:
```
tips = sns.load_dataset("tips")
tips.sample(5)
```

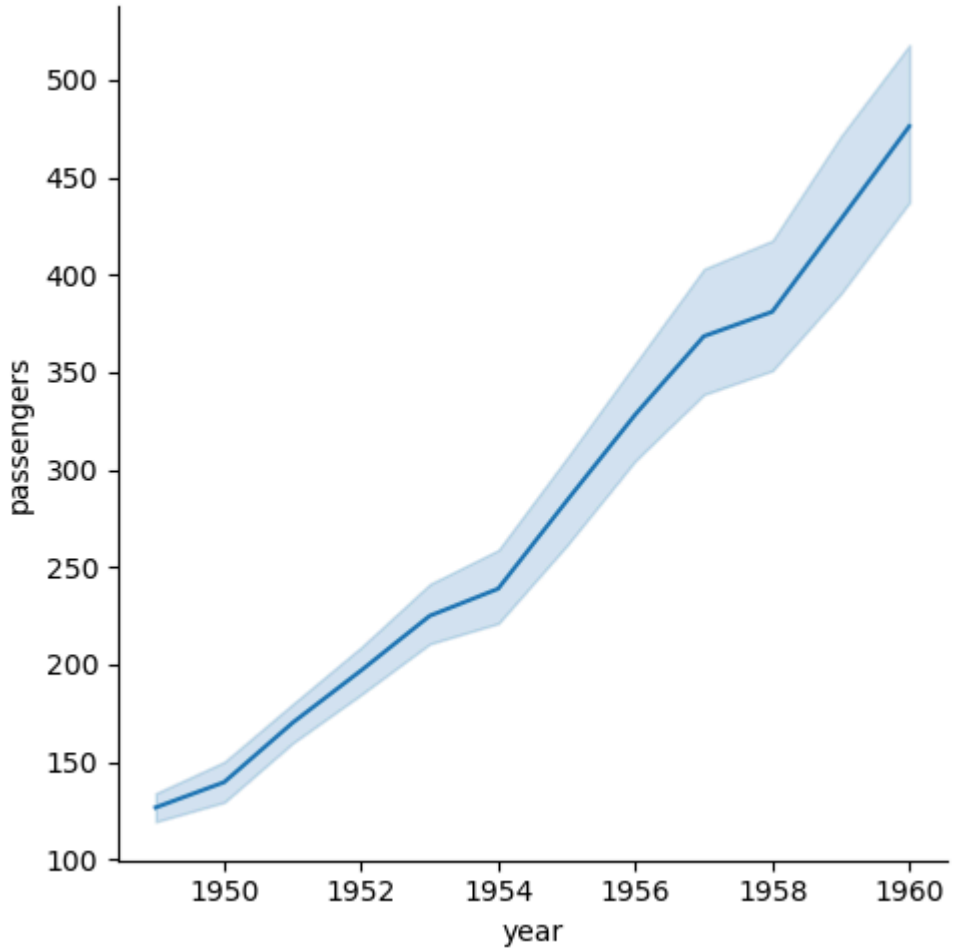
Out[39]:

	total_bill	tip	sex	smoker	day	time	size
222	8.58	1.92	Male	Yes	Fri	Lunch	1
231	15.69	3.00	Male	Yes	Sat	Dinner	3
34	17.78	3.27	Male	No	Sat	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
198	13.00	2.00	Female	Yes	Thur	Lunch	2

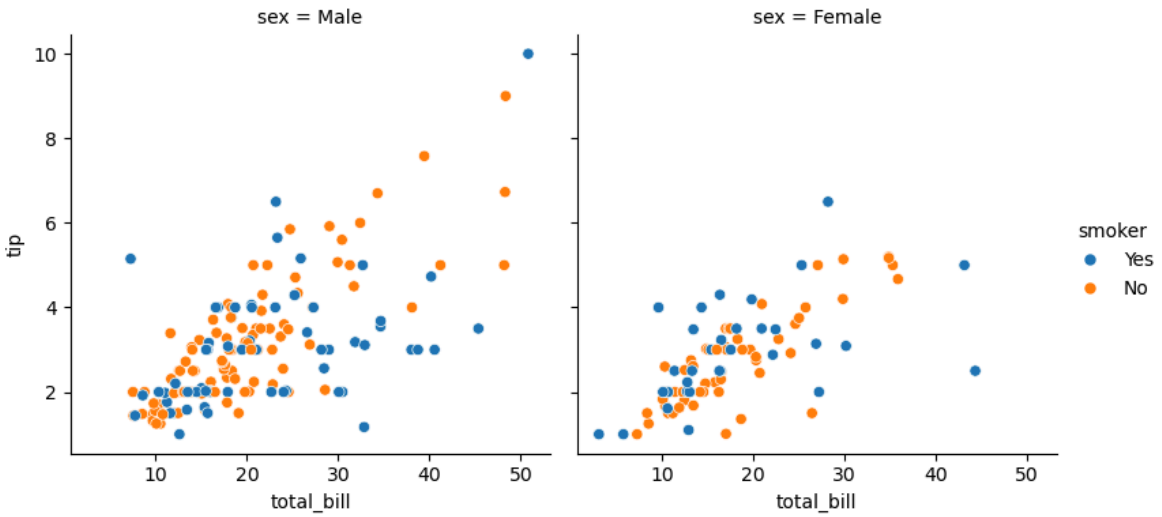
In [40]:
```
sns.relplot(x='total_bill', y='tip', hue='sex', data=tips, kind='scatter')
plt.show()
```



In [41]:
```
# representar la media y la variación de una variable en función de otra
data = sns.load_dataset('flights')
sns.relplot(x="year", y="passengers", kind="line", data=data)
plt.show()
```



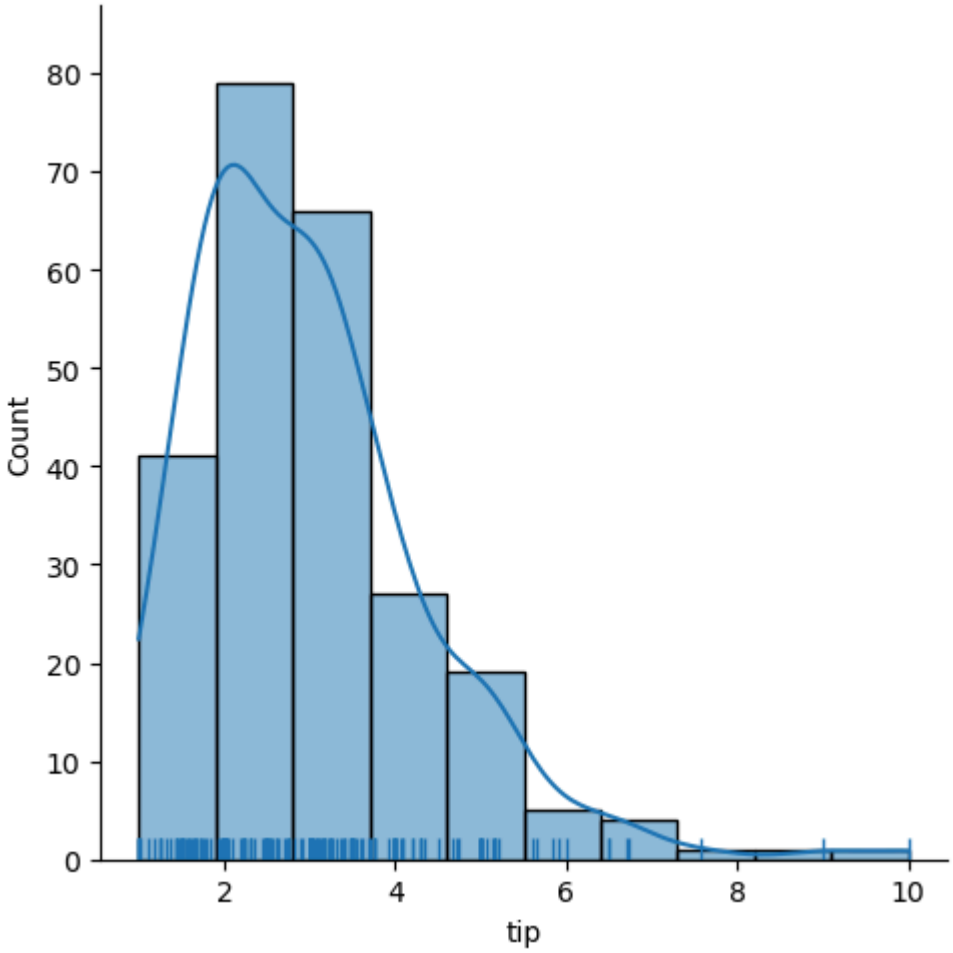
In [42]:
```
# relacionar hasta 4 variables automáticamente
sns.relplot(x="total_bill", y="tip", hue="smoker",
            col="sex", height=4,
            kind="scatter",data=tips); #kind ('line','scatter'...)
plt.show()
```



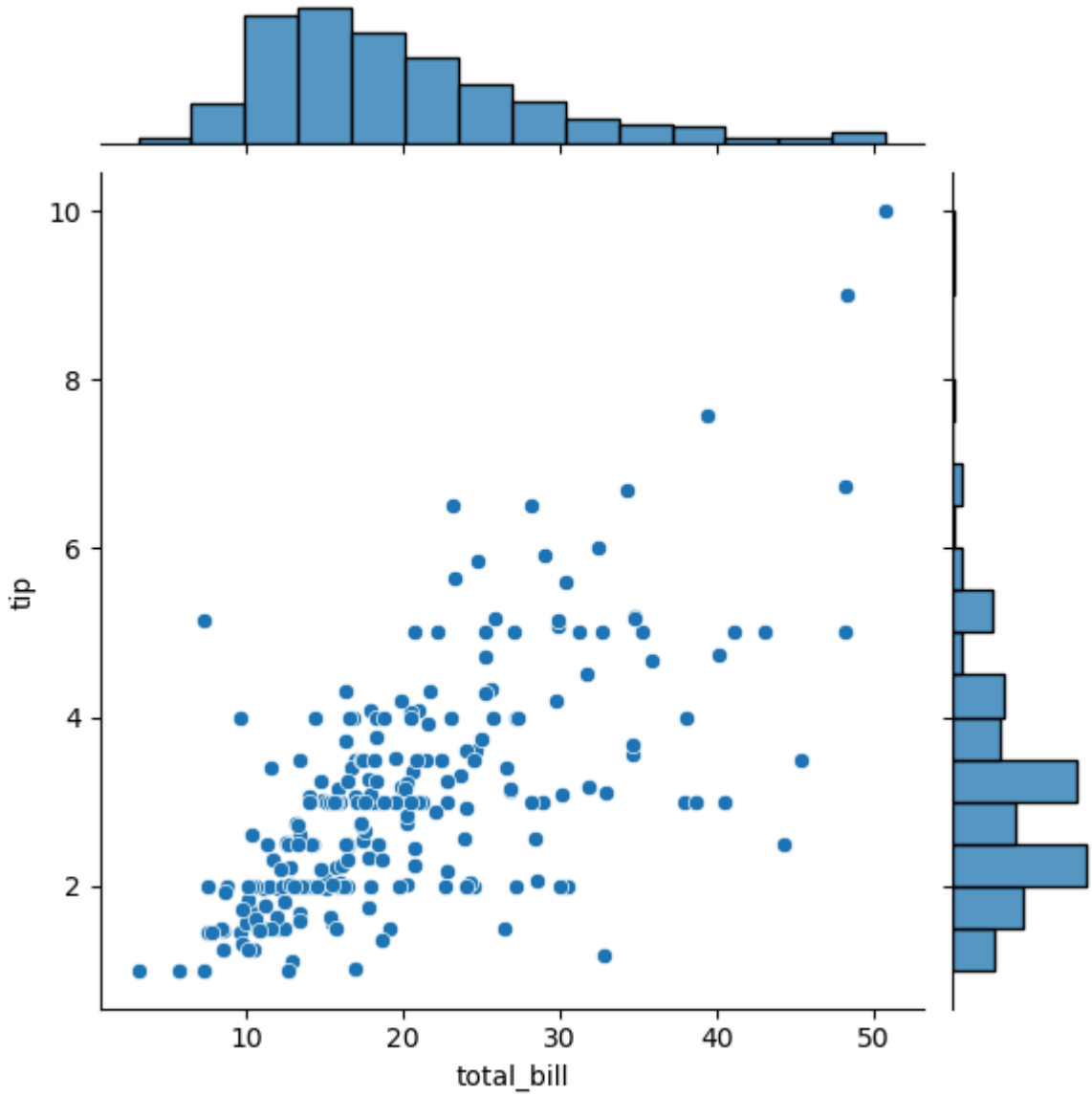
Distribución de un dataset

In [43]:
```
# histograma y density plot
# kde = gaussian kernel density estimate, linea de densidad
# bins = tamaño de los contenedores
# rug = densidad de los datos
```

```
sns.displot(tips['tip'], kde=True, bins=10, rug=True)
plt.show()
```



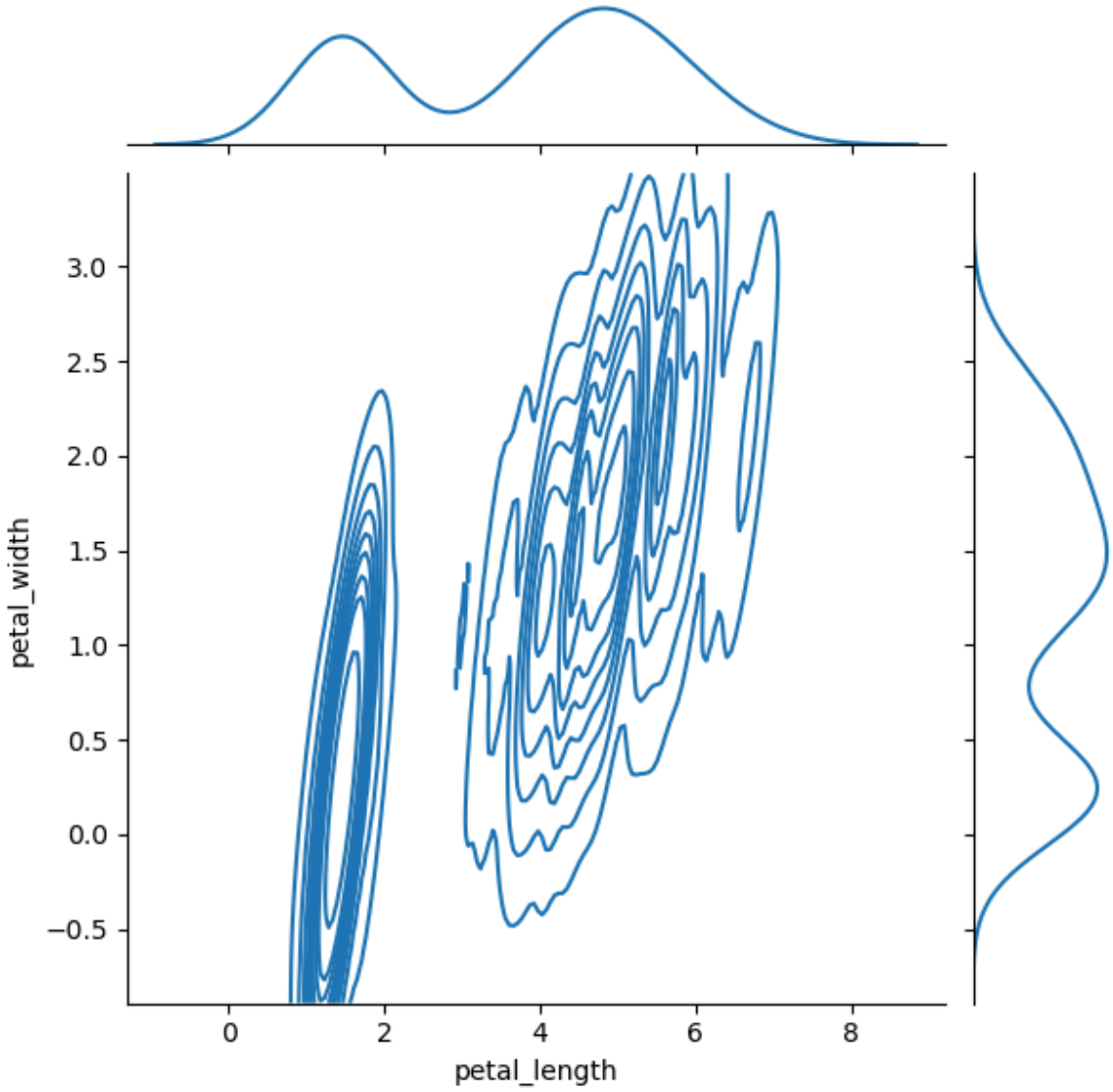
In [44]:
```
# Distribución bi-variable
sns.jointplot(x='total_bill', y='tip', data=tips)
plt.show()
```



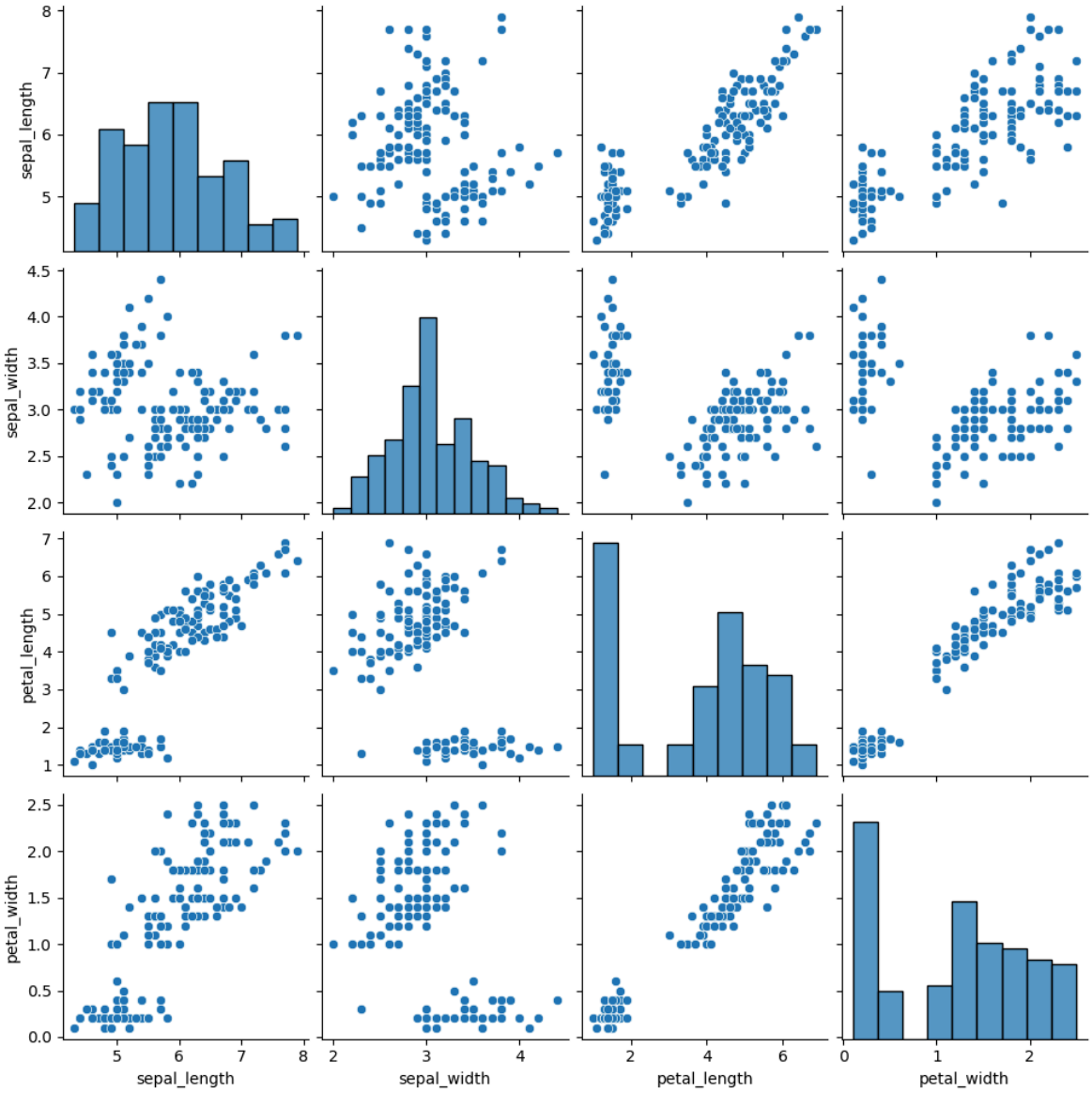
In [45]:
```
# Density plot bi-variable
iris = sns.load_dataset('iris')
display(iris.head(5))
display(iris["species"].unique())
sns.jointplot(x='petal_length', y='petal_width', kind="kde", data=iris)
plt.show()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

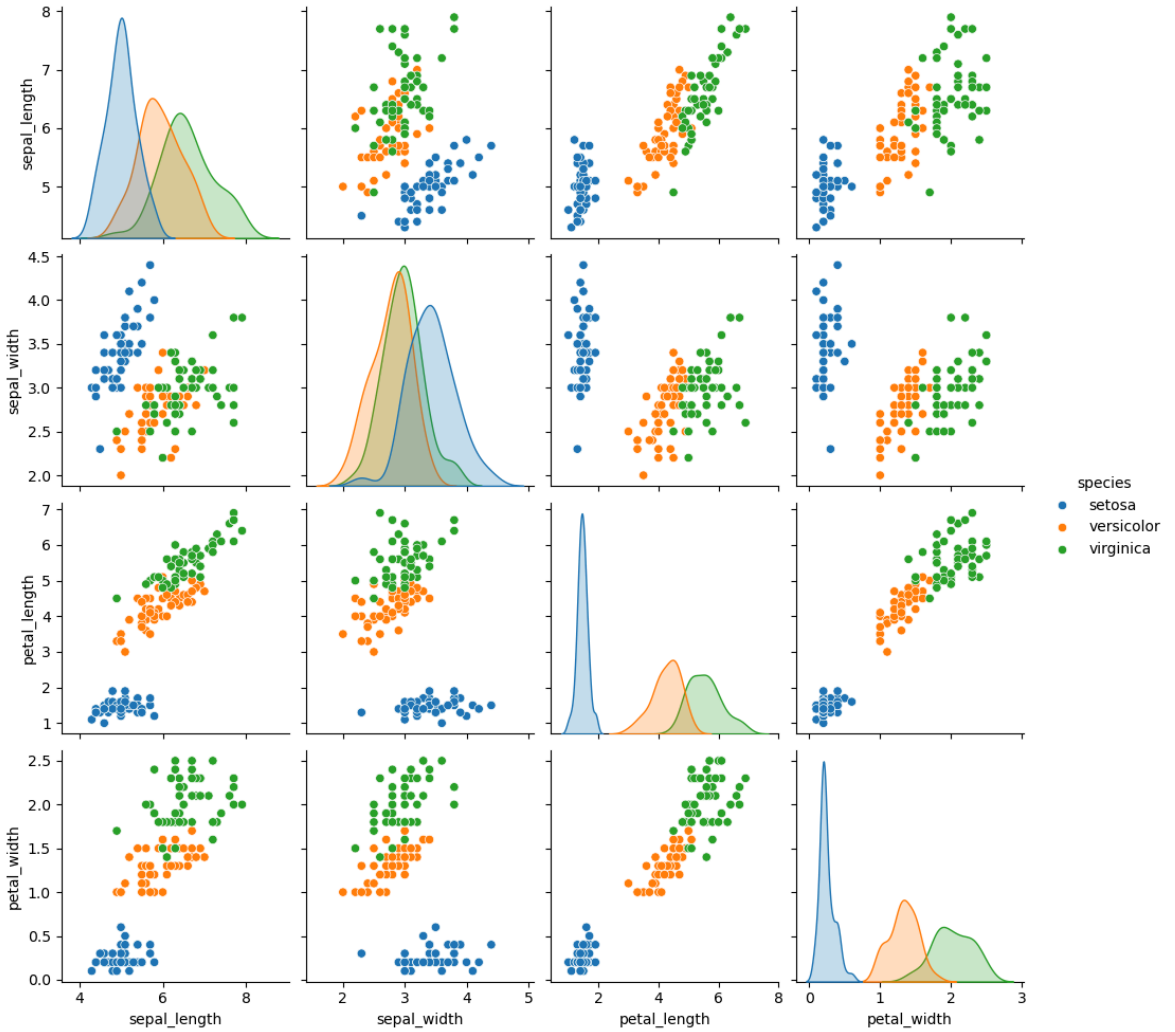
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```



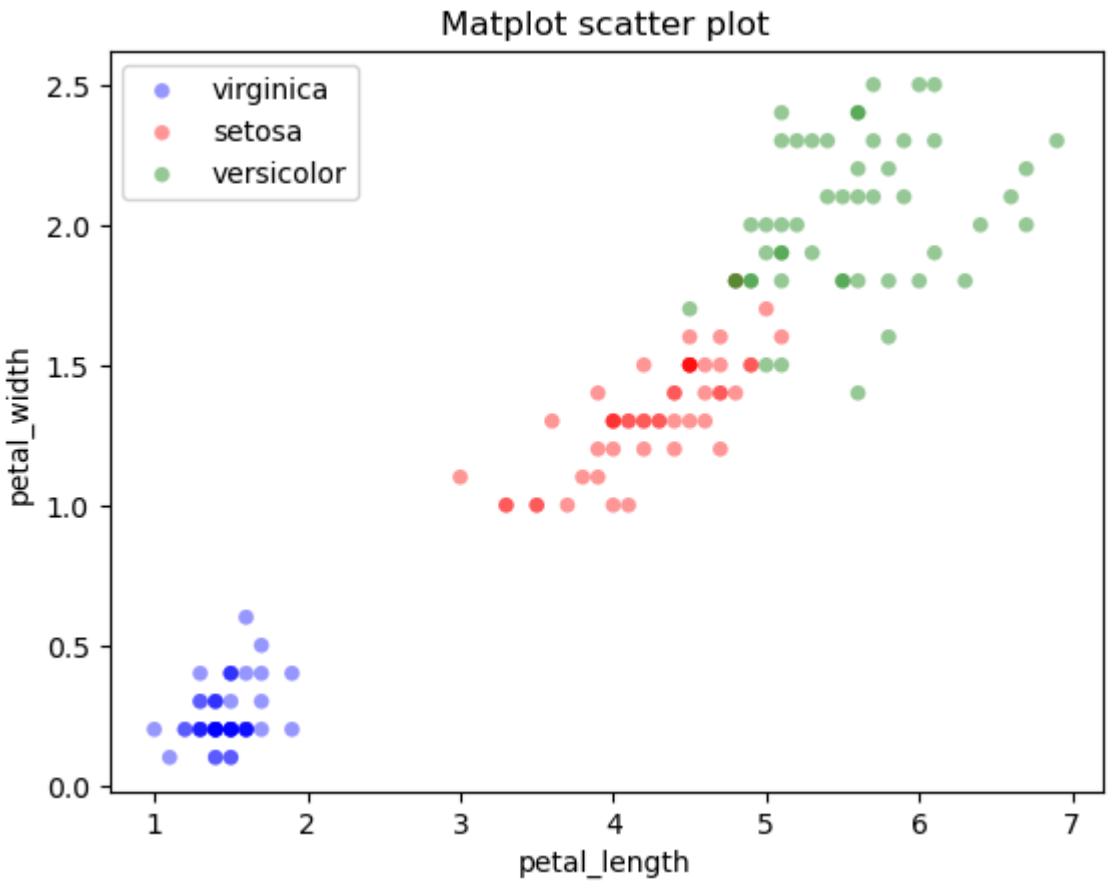
In [46]:
```
# Relación de cada pareja
sns.pairplot(iris)
plt.show()
```



In [47]:
```
# Relación de cada pareja
sns.pairplot(iris, hue = "species")
plt.show()
```



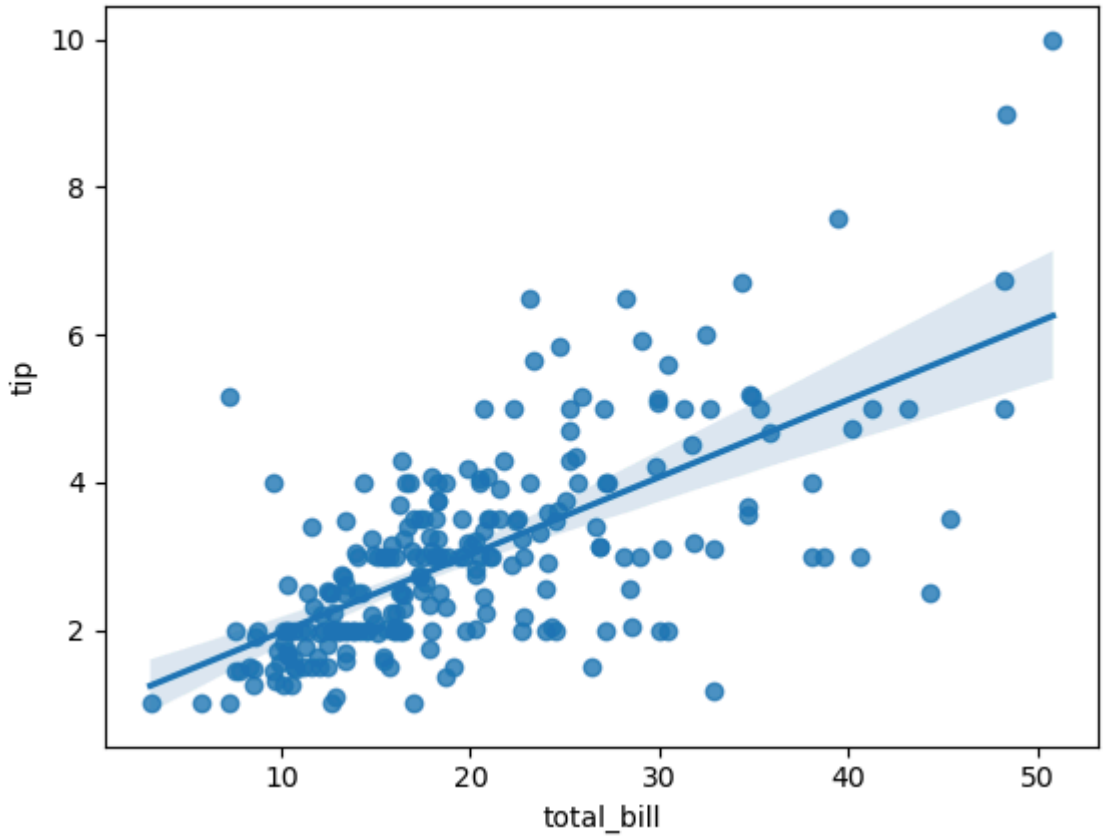
In [48]:
```
#scatter de clases, coloreando segun clase en matplotlib
iris = sns.load_dataset('iris')

# preparacion de los datos
data = []
colors = ["red", "green", "blue"]
groups = ["setosa", "versicolor", "virginica"]
columns = ["petal_length", "petal_width"]
for i in groups:
    # busqueda por clase definida
    data.append(iris[iris["species"] == i][columns].values)


#creacion de la grafica
fig = plt.figure()
ax = fig.add_subplot()
for count, d in enumerate(data):
    # separar las variables del grafico
    x, y = d[:,0], d[:,1]
    ax.scatter(x, y, alpha = 0.4, c = colors[count -1], edgecolors = 'none', s =
#etiquetas
ax.set_xlabel(columns[0])
ax.set_ylabel(columns[1])
plt.legend(loc='best')
plt.title('Matplot scatter plot')
plt.show()
```



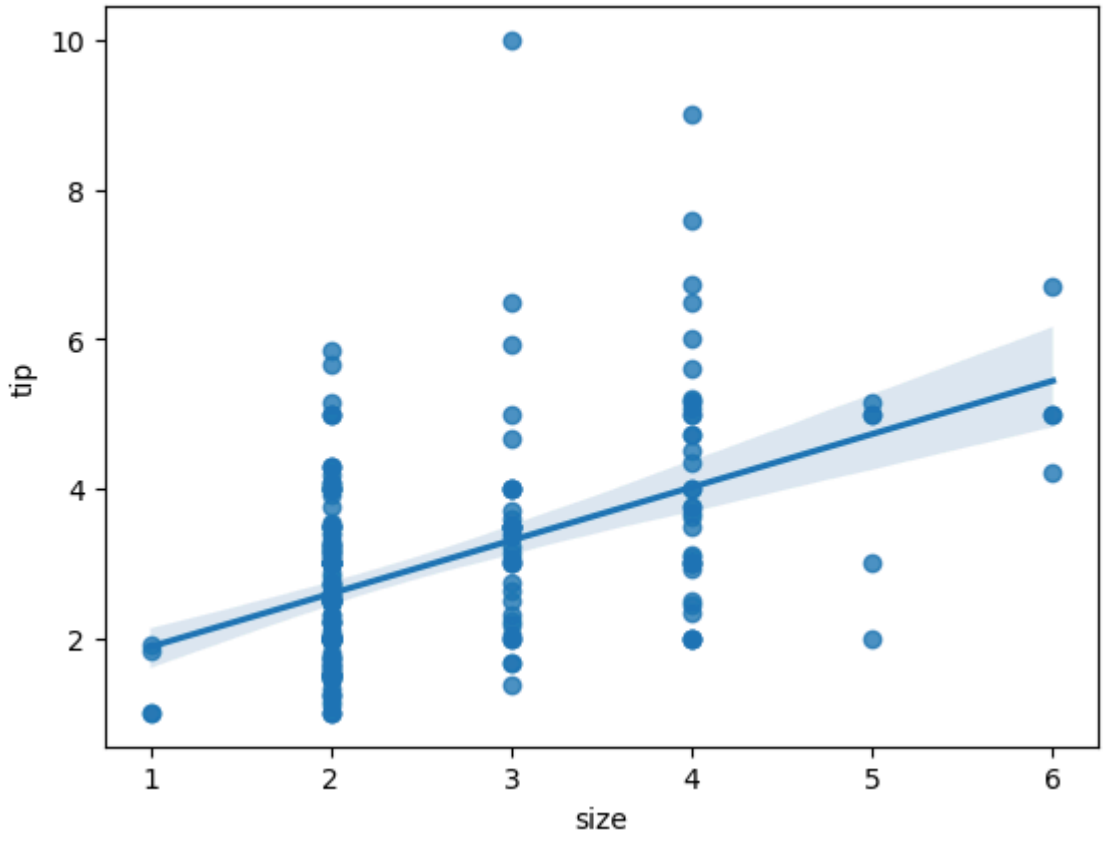
Visualización de correlaciones

- Regresión lineal con variables
- Correlaciones numéricas

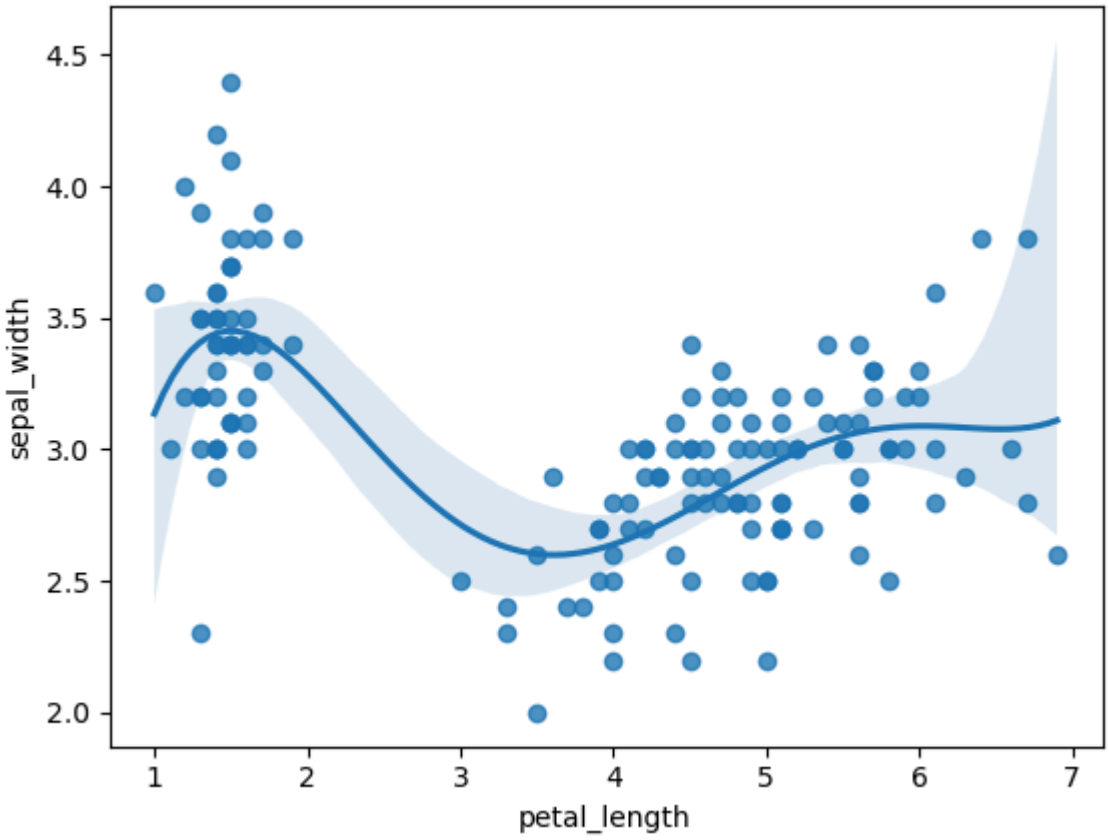
In [49]:
```
#correlación entre cuenta total y propina, recta regresion e intervalo de confia
sns.regplot(x="total_bill", y="tip", data=tips)
plt.show()
```



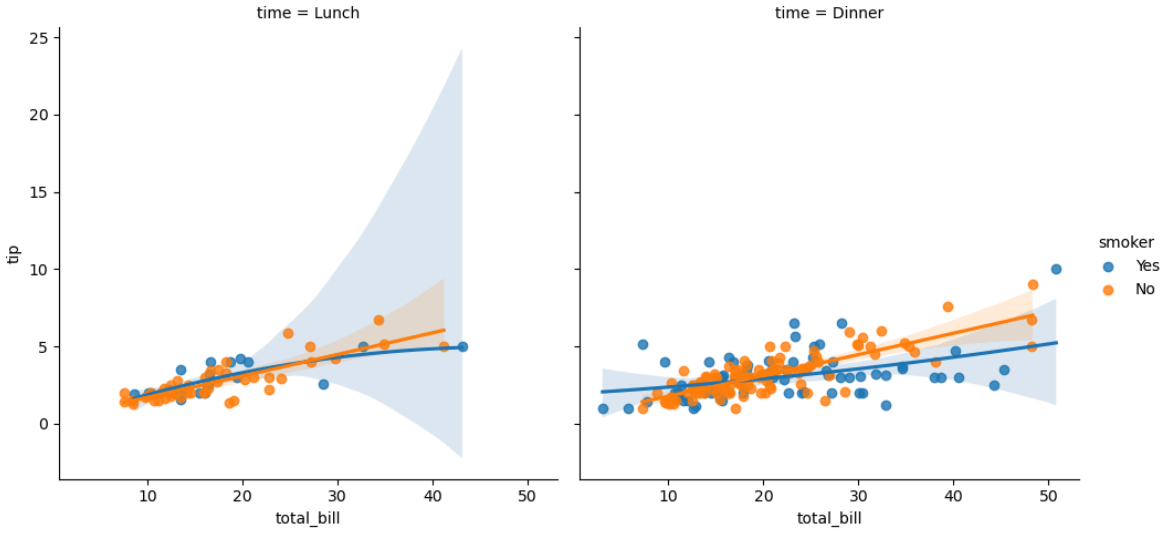
In [50]:
```
#correlación con valores categóricos (variable discreta, variable debe ser numér
sns.regplot(x="size", y="tip", data=tips)
plt.show()
```



In [51]:
```
# regresión con polinomios de más grados
sns.regplot(x="petal_length", y="sepal_width", data=iris, order=5)
plt.show()
```



In [52]:
```
# Correlaciones con varias variables
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips, col='time', order=2
plt.show()
```



Ejercicios

- Representación gráfica con matplotlib y seaborn

matplotlib

- crear una lista de años de 2000 a 2020 (eje x)
- generar datos aleatorios para el eje y (hacerlo 4 veces distintas: y0, y1, y2 e y3)

- Representar las 4 secuencias de datos aleatorios en una sola figura utilizando matplotlib

- Añade una leyenda para poder identificar cada secuencia en el gráfico
- Nombra las secuencias de la siguiente forma: "Hawaii", "San Marino", "Islas Feroe", "Guayana"

- Representar los mismos datos en 4 subfiguras (axes) distintas, parte de la misma figura

- Añade la siguiente info:
  - Título: "Datos aleatorios"
  - Leyenda
  - Nombre del eje x: "Año"
  - Nombre del eje y: "GDP"

- Añade al menos una anotación (texto y flecha) a uno de los gráficos

Seaborn

- Carga los datos de un dataset de seaborn (distinto a 'tips', 'iris' o 'flights') https://github.com/mwaskom/seaborn-data
- Si para alguno de los ejercicios no encuentras ejemplos en el dataset, busca otro para completarlo
- Recuerda utilizar la versión 'Raw' si se utiliza la URL en el método read_csv

- Representa visualmente la relación de cada pareja de atributos (variables)

- Escoge una pareja (ambos valores contínuos y numéricos) y demuestra su correlación

- Representa los valores de dos variables (x e y, numéricas) en base a otra variable (categórica)

- Crea los gráficos que demuestren lo siguiente en el dataset seleccionado:
  - La distribución de valores (media, distribución, outliers) de una variable numérica contínua
  - Compara la distribución de esa variable con otra (contínua y numérica) para representar cuantitativamente la densidad de elementos por valor en ambas variables

- Muestra un ejemplo en el dataset de una variable a la que un modelo de regresión lineal de orden > 1 sea más ajustado que uno de orden == 1

GitHub Visualizaciones

In [ ]: