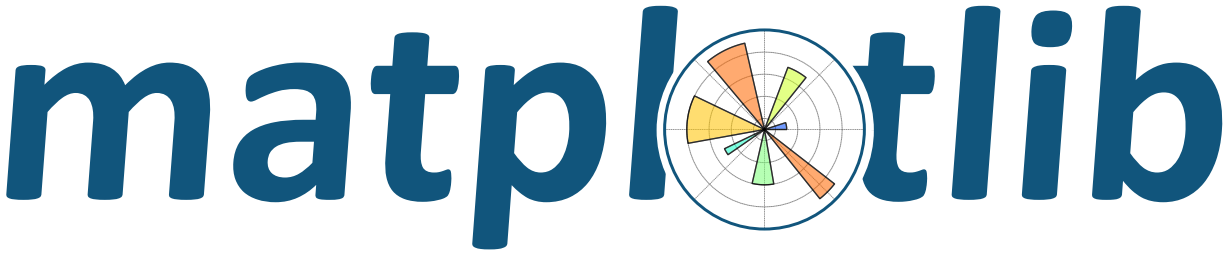
Introducción a Matplotlib

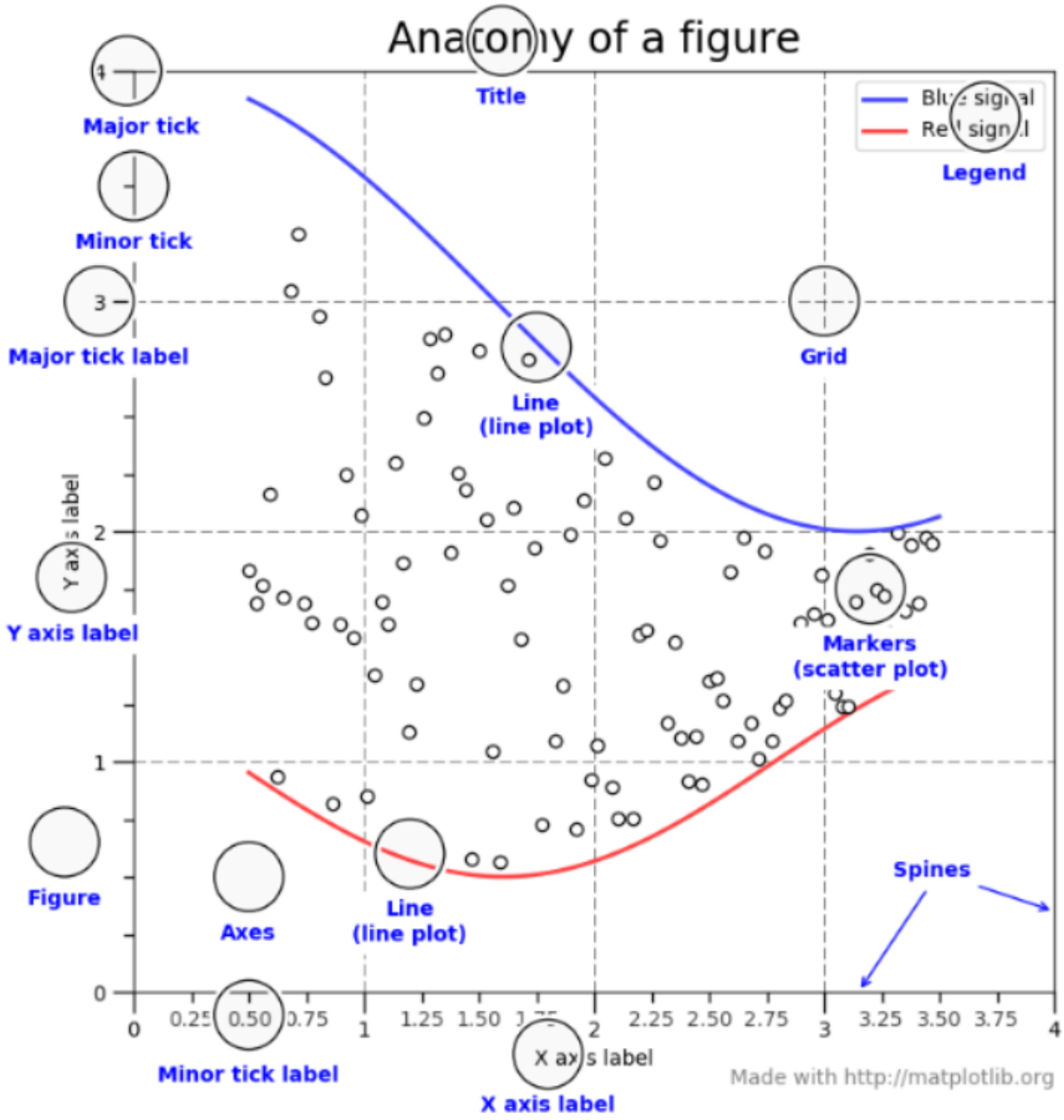


Matplotlib es básicamente una librería de visualización multiplataforma para datos que se basa en los arreglos `NumPy`. Esta librería está diseñada para funcionar en conjunto con la librería SciPy, que incluye diferentes módulos de Python utilizados para el aprendizaje automático y la ciencia de datos.

- Es el paquete de visualización de datos de Python más utilizado.
- Es una librería de visualización y estas visualizaciones nos permiten representar una gran cantidad de datos en forma visual, como gráficos y diagramas.
- Es útil para crear diagramas 2D a partir de los datos en arreglos `NumPy`.
- Está inspirada en el lenguaje de programación MATLAB y también proporciona una interfaz similar a MATLAB para gráficos.
- Esta librería se integra fácilmente con el paquete `Pandas` que se utiliza para la manipulación de datos.
- Con la combinación de `Pandas` y `Matplotlib`, la manipulación de datos se puede realizar junto con la visualización y se puede obtener información valiosa de los datos.
- Se puede usar con Jupyter Notebook, aplicaciones de servidor web y shells de IPython.

Anatomía de un gráfico en matplotlib

La figura es el contenedor de nivel superior en esta jerarquía. Es la ventana en la que se grafica todo. Puede tener múltiples figuras independientes y las figuras pueden contener múltiples Ejes(Axes/Subplot).



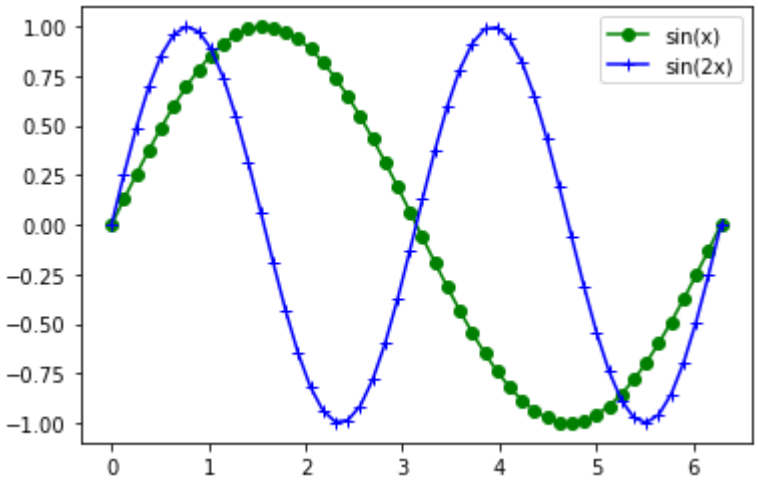
La mayoría de los gráficos se realizan sobre los Ejes(Axes). Los Ejes son efectivamente el área sobre la que graficamos los datos y cualquier marca / etiqueta / etc. asociados con ellos. Por lo general, configuramos los Ejes(Axes) con una llamada a subplot (que coloca los ejes en una cuadrícula regular), por lo que en la mayoría de los casos, Ejes(Axes) y subplot son sinónimos.

Cada Axes tiene un XAxis y un YAxis. Estos contienen las marcas, ubicaciones de las marcas, etiquetas, etc.

Importar matplotlib

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

In [14]:
```
x = np.linspace(0, 2*np.pi, 50)
y1 = np.sin(x)
y2 = np.sin(2*x)
plt.figure() # Crear una figura
plt.plot(x, y1 ,'g-o', x, y2, 'b-+')
plt.legend(['sin(x)', 'sin(2x)']);
```



Figuras

Crear una Figura

In [17]:
```
fig = plt.figure(figsize=(8,6))
plt.show()   # Solo es necesario si se va a ejecutar fuera de un notebook
```

```
<Figure size 576x432 with 0 Axes>
```

Genial, una figura en blanco! No es terriblemente útil todavía. Para cerrar una figura usar:

- close(): Cierra la figura actual
- close(num): Cierra la figura con el número `num`
- close(fig): Cierra la isntacia de figura `fig`
- close('all'): Cierra todas las figuras

In [18]:
```
plt.close()
```

Sin embargo, mientras estamos en el tema, puede controlar el tamaño de la figura a través del argumento de tamaño `figsize`, que espera una tupla de (ancho, alto) en pulgadas. Una función de utilidad realmente útil es `figaspect()`

In [19]:
```
# El doble de ancho que de alto:
fig = plt.figure(figsize=plt.figaspect(0.5))
```

```
<Figure size 576x288 with 0 Axes>
```

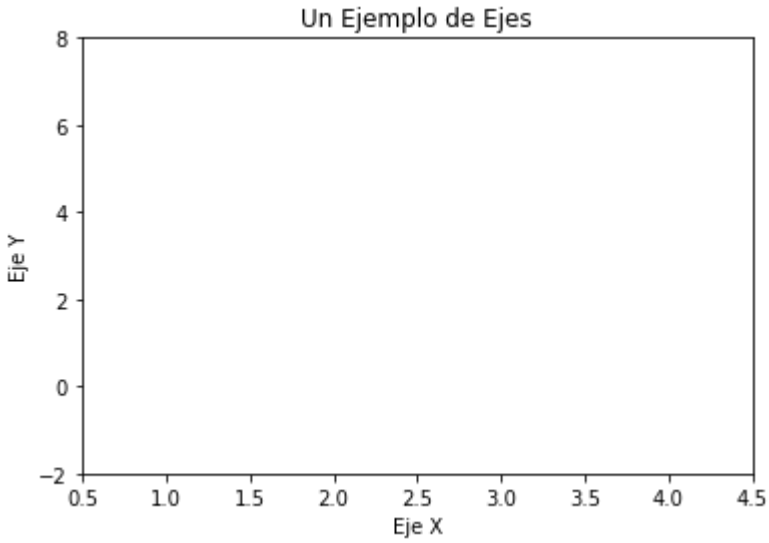
Axes (Ejes)

Todo el graficado se hace con respecto a un Axes. Un Axes se compone de objetos Axis (Eje) y muchas otras cosas. Un objeto Axes debe pertenecer a una figura (y solo una figura). La mayoría de los comandos que emitirá serán con respecto a este objeto Axes.

Normalmente, configurará una figura y luego agregará un eje a ella.

Puede usar `fig.add_axes`, pero en la mayoría de los casos, encontrará que agregar un `subplot` se ajustará perfectamente a sus necesidades. (De nuevo, un `subplot` es solo un eje en un sistema de cuadrícula).

In [20]:
```
fig = plt.figure()
ax = fig.add_subplot(111) # Explicaremos el "111" más adelante. Basicamente esta definiendo 1 fi
ax.set(xlim=[0.5, 4.5], ylim=[-2, 8], title='Un Ejemplo de Ejes',
       ylabel='Eje Y', xlabel='Eje X');
```



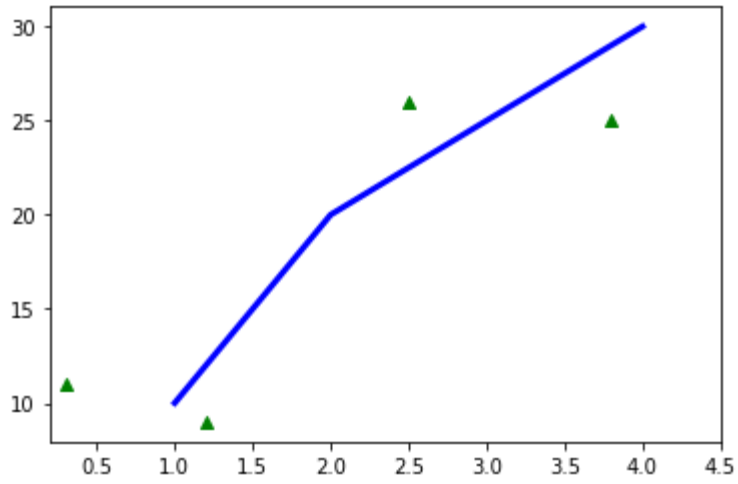
Graficar

La mayoría de los gráficos ocurren en un Ejes. Por lo tanto, si graficas algo en un eje, entonces usarás uno de sus métodos.

Hablaremos sobre los diferentes métodos de graficado en mayor profundidad en la siguiente sección. Por ahora, centrémonos en dos métodos: `plot` (gráficos de línea) y `scatter` (gráficos de puntos).

Ejemplo:

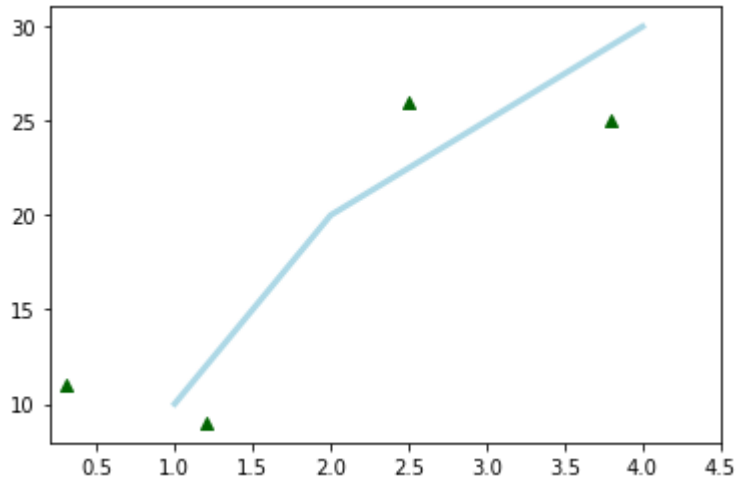
In [27]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([1, 2, 3, 4], [10, 20, 25, 30], color='blue', linewidth=3)
ax.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='green', marker='^')
ax.set_xlim(0.2, 4.5);
```



Graficado explicito (Métodos de Axes) vs. implicito (pyplot)

Curiosamente, casi todos los métodos de un objeto Axes existen como una función en el módulo pyplot (y viceversa). Por ejemplo, al llamar a `plt.xlim (1, 10)`, pyplot llama a `ax.set_xlim (1, 10)` en el Axes "actual". Aquí hay una versión equivalente del ejemplo anterior usando solo pyplot.

In [28]:
```
plt.figure()
plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3)
plt.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', marker='^')
plt.xlim(0.2, 4.5);
```



Si bien para los gráficos simples, con código cortos el enfoque implícito pyplot es mas conciso, al hacer gráficos más complicados o trabajar con códigos más grandes, es preferible referirse al Axes o la figura explícitamente.

La ventaja de mantener con claridad con que Axes estamos trabajamos se hará más obvia cuando comencemos a tener múltiples Axes en una figura.

Además, segun el Zen de Python:

"*Lo explicito es mejor que lo implicito*"

Multiples Axes

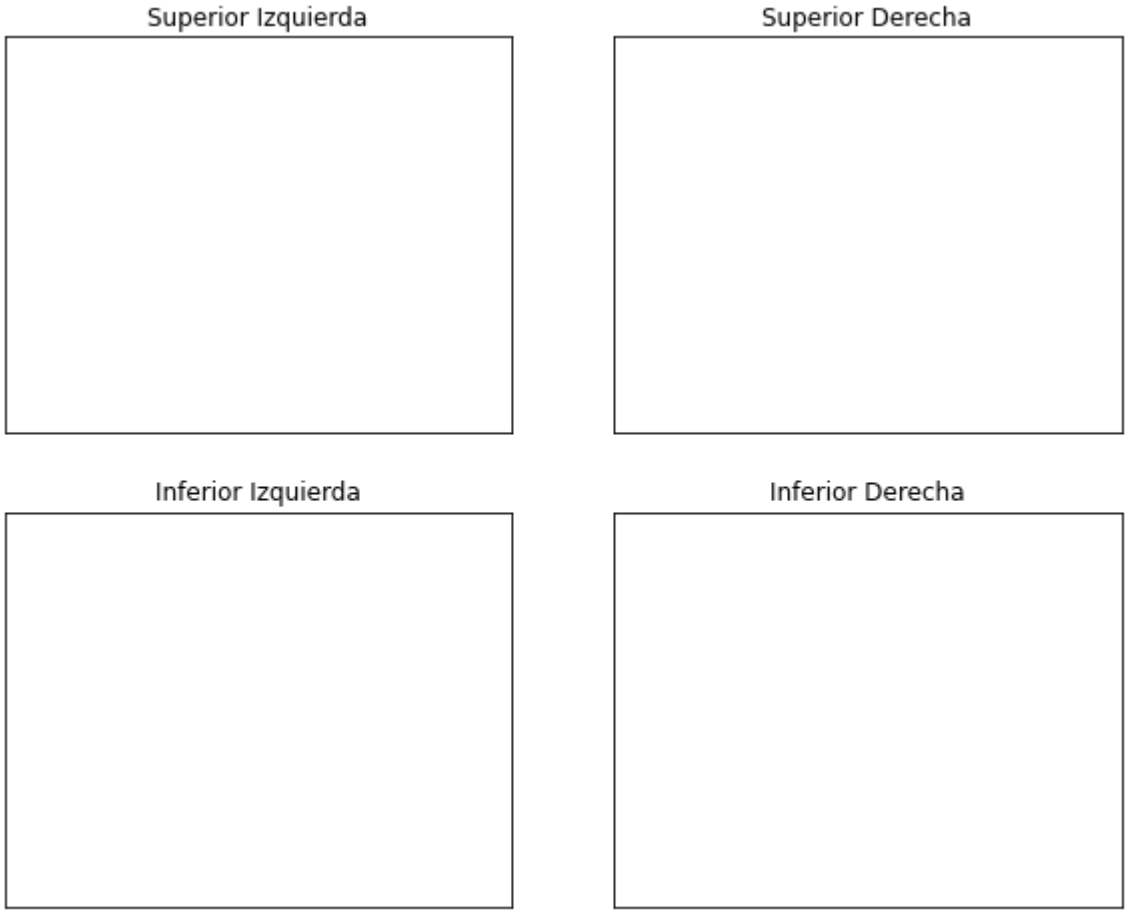
Hemos mencionado anteriormente que una figura puede tener más de un Axes en ella. Si desea que sus Axes estén en un sistema de cuadrícula regular, entonces es más fácil usar `plt.subplots(...)` para crear una figura y agregarle los Axes automáticamente.

La llamada a `plt.subplots(nrows=2, ncols=2)` crea una nueva figura y le agrega 4 subplots. El objeto `axes` que se devolvió es una matriz de objetos numpy de 2D. Cada elemento en la matriz es una de los subplots.

Por lo tanto, cuando queremos trabajar con uno de estos `axes`, podemos indexar el conjunto de `axes` y usar los métodos de ese elemento.

Por ejemplo:

In [33]:
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,8))
axes[0,0].set(title='Superior Izquierda')
axes[0,1].set(title='Superior Derecha')
axes[1,0].set(title='Inferior Izquierda')
axes[1,1].set(title='Inferior Derecha')

# Para iterar sobre todos los elementos de un arreglo multidimensional, usar el atributo `flat`
for ax in axes.flat:
    # Remove todas las marcas del eje x y del eje y
    ax.set(xticks=[], yticks=[])
```



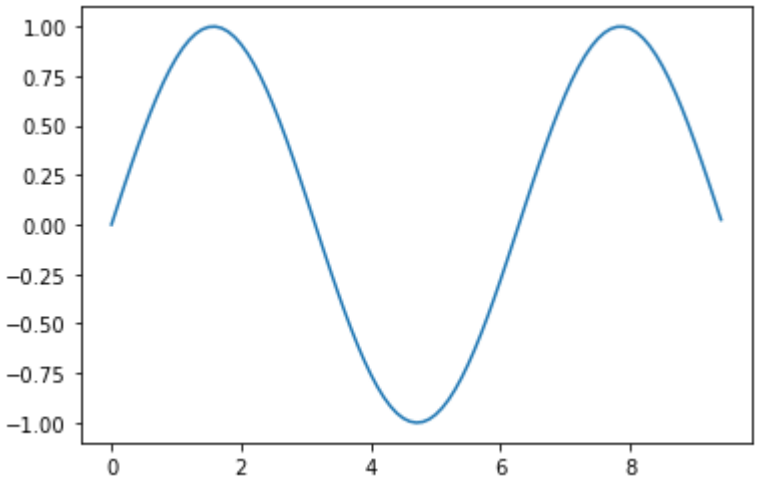
Gráficos de línea

La función mas importante de `matplotlib` es `plot`, la cual permite realizar gráficos de linea de datos en 2D.

Por ejemplo:

In [34]:
```
# Calcular las coordenadas x e y para puntos en una curva senoidal
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
# Crear una figura
plt.figure()
# Graficar los puntos usando matplotlib
plt.plot(x, y)
```

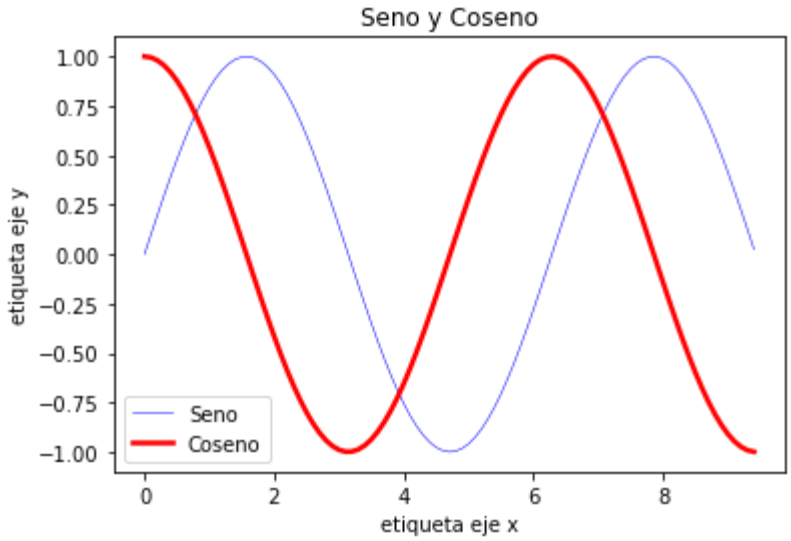
Out[34]:
```
[<matplotlib.lines.Line2D at 0x1e5d5b61790>]
```



Con solo un poco de trabajo extra, podemos trazar varias líneas a la vez y agregar etiquetas de título, leyenda y a los ejes:

In [41]:
```
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.figure()

# Graficar los puntos usando matplotlib
plt.plot(x, y_sin, color="blue", linewidth=0.5, linestyle="-")
plt.plot(x, y_cos, color="red",  linewidth=2.5, linestyle="-")
plt.xlabel('etiqueta eje x')
plt.ylabel('etiqueta eje y')
plt.title('Seno y Coseno')
plt.legend(['Seno', 'Coseno'])
```

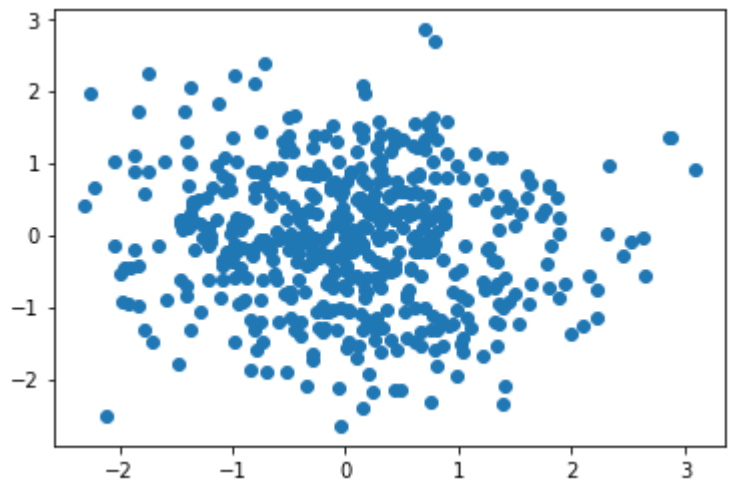
Out[41]:
```
<matplotlib.legend.Legend at 0x1e5d99fdd30>
```



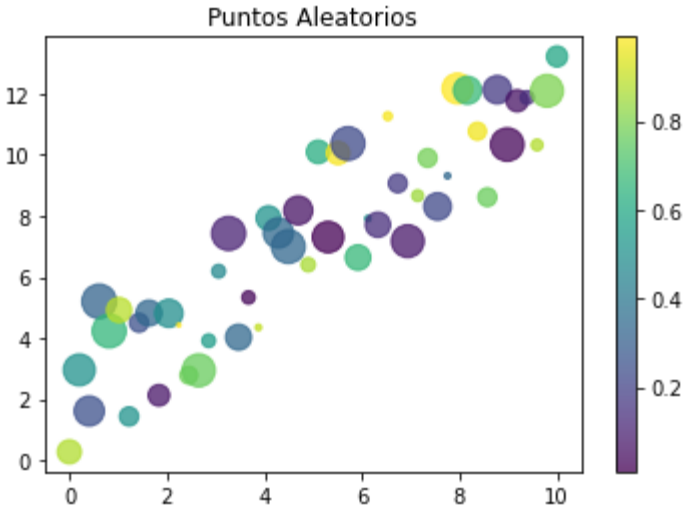
Gráficos de puntos

La función `scatter` de matplotlib nos permite realizar gráficos de puntos de datos en 2D

In [45]:
```
plt.figure()
# Graficar puntos
x = np.random.normal(size=500)
y = np.random.normal(size=500)
plt.scatter(x, y);
```



In [50]:
```
N = 50 # número de puntos
x = np.linspace(0, 10, N)
e = np.random.rand(N)*5.0 # ruido
y1 = x + e

areas = np.random.rand(N)*300

colores = np.random.rand(N)

plt.scatter(x, y1, s=areas, c=colores, alpha=0.75)

plt.colorbar()
plt.title("Puntos Aleatorios");
```



Colores

Esta es, quizás, la parte más importante del vocabulario de Matplotlib. Dado que Matplotlib es una biblioteca de graficado, los colores están asociados con todo lo que se grafica en sus figuras. Matplotlib admite un lenguaje muy sólido para especificar colores que deberían ser familiares para una gran variedad de usuarios.

Nombres de colores

Primero, los colores se pueden dar como cadenas. Para colores muy básicos, incluso se puede usar una sola letra:

- b: azul
- g: verde
- r: rojo
- c: cian
- m: magenta
- y: amarillo
- k: negro
- w: blanco

Otros nombres de colores que se permiten son los nombres de colores HTML / CSS como "burlywood" y "chartreuse". Vea la lista completa de los 147 nombres de colores.

Valores hexadecimales

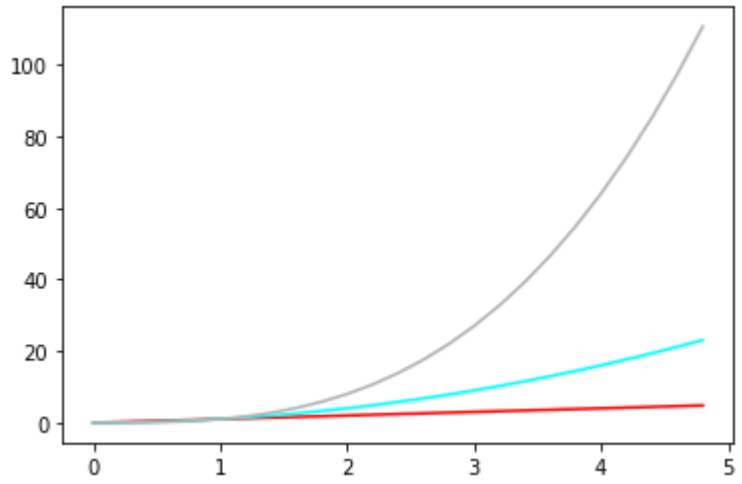
Los colores también se pueden especificar suministrando una cadena hexadecimal de HTML/CSS, como `'#0000FF'` para azul.

256 tonos de gris

Se puede dar un nivel de gris en lugar de un color al pasar una representación de cadena de un número entre 0 y 1, inclusive. `'0.0'` es negro, mientras que `'1.0'` es blanco. `'0.75'` sería un tono claro de gris.

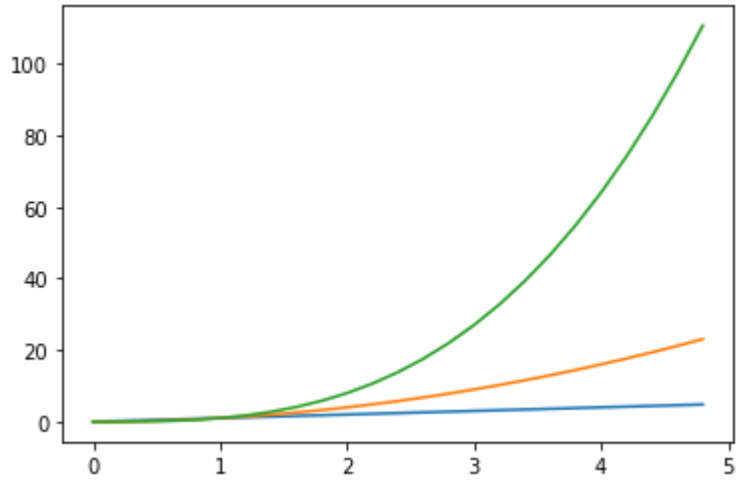
Ejemplo:

In [51]:
```
plt.figure()

t = np.arange(0.0, 5.0, 0.2)
plt.plot(t, t, 'r', t, t**2, 'cyan', t, t**3, '0.7')
plt.show()
```



En caso de no especificar los colores, matplotlib utiliza unos colores por defecto:

In [52]:
```
plt.figure()
t = np.arange(0.0, 5.0, 0.2)
plt.plot(t, t, t, t**2, t, t**3)
plt.show()
```



Marcadores

Marcadores se usan regularmente en gráficos `plot()` y `scatter()`, pero también aparecen en otros lugares. Hay un amplio conjunto de marcadores disponibles, e incluso se pueden especificar marcadores personalizados.

marcador	descripción	marcador	descripción	marcador	descripción	marcador	descripción
"."	punto	"+"	suma	","	pixel	"x"	cruz
"o"	círculo	"D"	diamante	"d"	diamante_fino		
"8"	octágono	"s"	cuadrado	"p"	pentágono	"*"	estrella
"|"	barra vertical	"_"	barra horizontal	"h"	hexágono1	"H"	hexágono2
0	tick izquierda	4	caret izquierda	"<"	triángulo izquierda	"3"	tri izquierda
1	tick derecha	5	caret derecha	">"	triángulo derecha	"4"	tri derecha
2	tick arriba	6	caret arriba	"^"	triángulo arriba	"2"	tri arriba
3	tick abajo	7	caret abajo	"v"	triángulo abajo	"1"	tri abajo
"None"	nada	`None`	valor por defecto	" "	nada	""	nada

In [55]:
```
xs, ys = np.mgrid[:4, 9:0:-1]
marcadores = [".", "+", ",", "x", "o", "D", "d", "", "8", "s", "p", "*", "|", "_", "h", "H", 0, 
           1, 5, ">", "4", 2, 6, "^", "2", 3, 7, "v", "1", "None", None, " ", ""]
descripcion = ["punto", "suma", "pixel", "cruz", "círculo", "diamante", "diamante fino", "",
             "octágono", "cuadrado", "pentágono", "estrella", "barra vertical", "barra horizonta
             "tick izquierda", "caret izquierda", "triángulo izquierda", "tri izquierda", "tick 
             "tick arriba", "caret arriba", "triángulo arriba", "tri arriba", "tick abajo", "car
             "Nada", "valor por defecto", "Nada", "Nada"]
fig, ax = plt.subplots(1, 1, figsize=(9.5, 4))
for x, y, m, d in zip(xs.T.flat, ys.T.flat, marcadores, descripcion):
    ax.scatter(x, y, marker=m, s=90)
    ax.text(x + 0.1, y - 0.1, d, size=12)
ax.set_axis_off()
plt.show()
```

Estilo de líneas

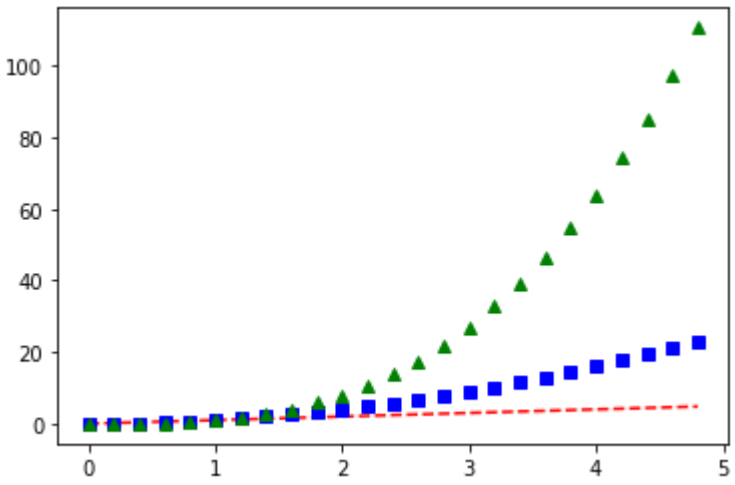
Los estilos de línea son tan comunes como los colores. Hay algunos estilos de línea predefinidos disponibles para su uso. Tenga en cuenta que hay algunas técnicas avanzadas para especificar algunos estilos de línea personalizados. Aquí es un ejemplo de un patrón de guión personalizado.

estilo línea	descripción
'-'	sólida
'--'	discontinua
'-.'	guión punto
':'	punteada
'Ninguno'	no dibujar nada
''	no dibujar nada
''	no dibujar nada

Además, no mezcle ".-" (línea con marcadores de puntos) y "-." (línea de puntos y rayas) cuando se usa la función `plot` !

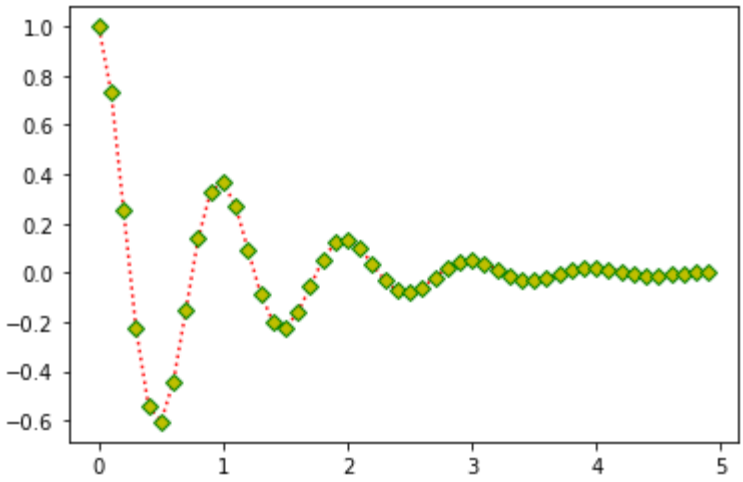
Ejemplo 1: Hacer un gráfico con rayas rojas, cuadrados azules y triángulos verdes

In [59]:
```
plt.figure()

t = np.arange(0., 5., 0.2)
# rayas rojas, cuadrados azules y triángulos verdes
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



Ejemplo 2: Hacer un gráfico que tenga una línea roja punteada, con marcadores de diamantes amarillos que tengan un borde verde

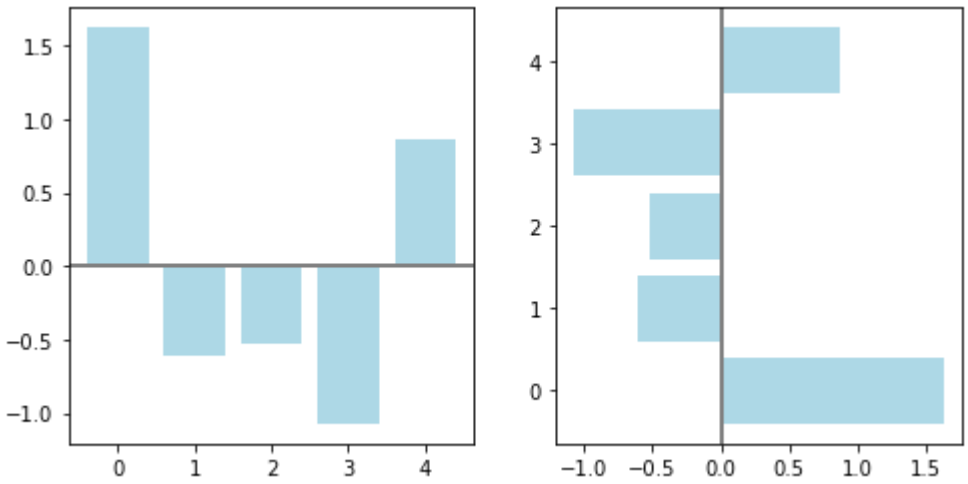
In [64]:
```
plt.figure()

t = np.arange(0.0, 5.0, 0.1)
a = np.exp(-t) * np.cos(2*np.pi*t)
plt.plot(t, a, 'r:D', markerfacecolor='y', markeredgecolor='g')
plt.show()
```



Gráficos de barras

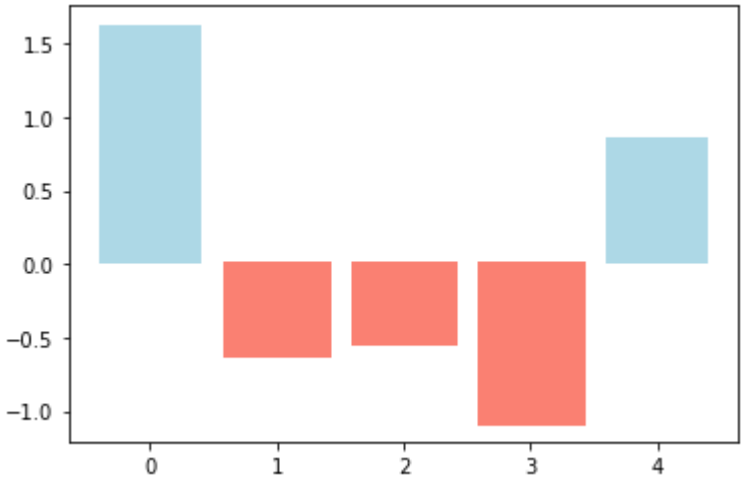
Los gráficos de barras son uno de los tipos de gráficos más comunes. El método `ax.bar(...)` de Matplotlib también puede trazar rectángulos generales, pero el valor predeterminado está optimizado para una secuencia simple de valores x, y, donde los rectángulos tienen un ancho constante. También hay `ax.barh(...)` (para horizontal), que hace una suposición de altura constante en lugar de una suposición de ancho constante.

In [69]:
```
np.random.seed(1)
x = np.arange(5)
y = np.random.randn(5)

fig, axes = plt.subplots(ncols=2, figsize=plt.figaspect(1./2))

barras_verticales = axes[0].bar(x, y, color='lightblue', align='center')
barras_horizontales = axes[1].barh(x, y, color='lightblue', align='center')

# las funciones axhline & axvline grafican una línea en todo el camino a través de los ejes
axes[0].axhline(0, color='gray', linewidth=2)
axes[1].axvline(0, color='gray', linewidth=2)

plt.show()
```



Podemos fijar el color de las barras de acuerdo a alguna condición. Por ejemplo si la barra es negativa hacer el color salmon

In [72]:
```
fig, ax = plt.subplots()
barras_verticales = ax.bar(x, y, color='lightblue', align='center')

# We could have also done this with two separate calls to `ax.bar` and numpy boolean indexing.
for barra, altura in zip(barras_verticales, y):
    if altura < 0:
        barra.set(color='salmon', linewidth=3)

plt.show()
```



Ejemplo de gráfico de barras horizontales:

In [77]:
```
plt.figure()
paises = ("Alemania", "España", "Francia", "Portugal")
posicion_y = np.arange(len(paises))
unidades = (342, 321, 192, 402)
plt.barh(posicion_y, unidades)
plt.yticks(posicion_y, paises)
plt.xlabel('Unidades vendidas')
plt.title("Ventas en Europa")
```

Out[77]:
```
Text(0.5, 1.0, 'Ventas en Europa')
```

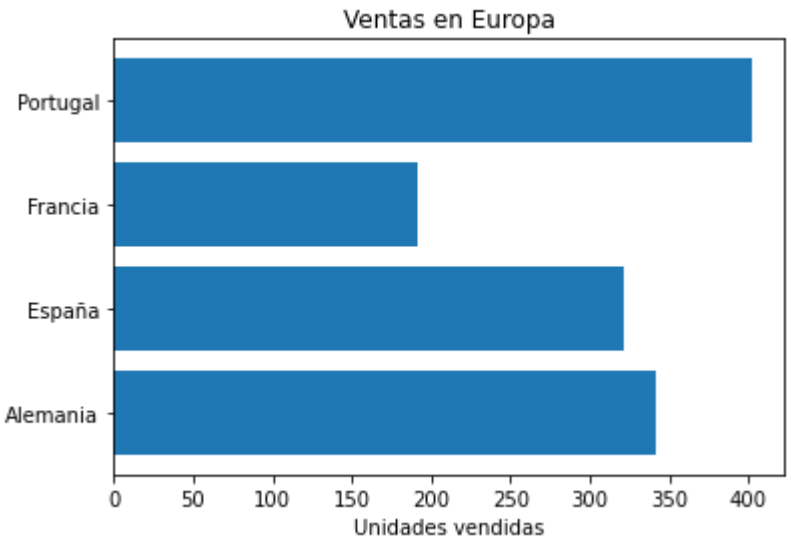
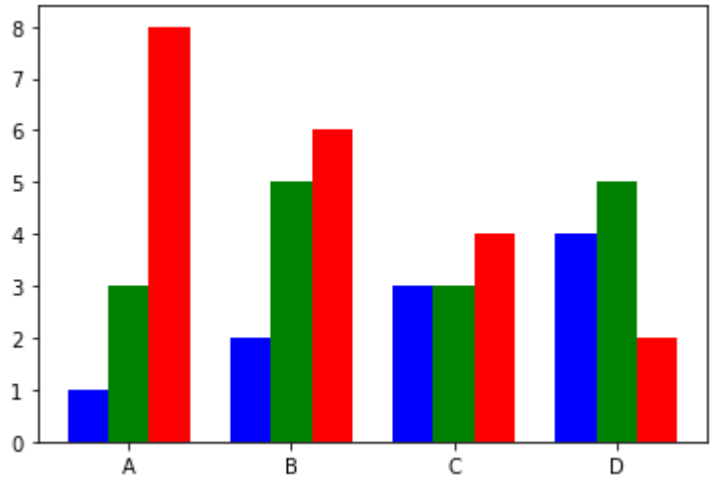


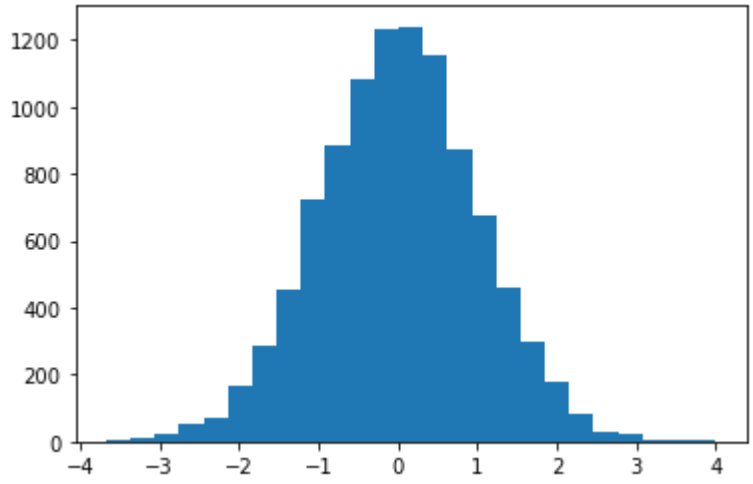
Gráfico con miltiples barras verticales:

In [78]:
```
plt.figure()

datos = [[1, 2, 3, 4], [3, 5, 3, 5], [8, 6, 4, 2]]
X = np.arange(4)
plt.bar(X + 0.00, datos[0], color = "b", width = 0.25)
plt.bar(X + 0.25, datos[1], color = "g", width = 0.25)
plt.bar(X + 0.50, datos[2], color = "r", width = 0.25)
plt.xticks(X+0.25, ["A","B","C","D"]);
```



Histogramas

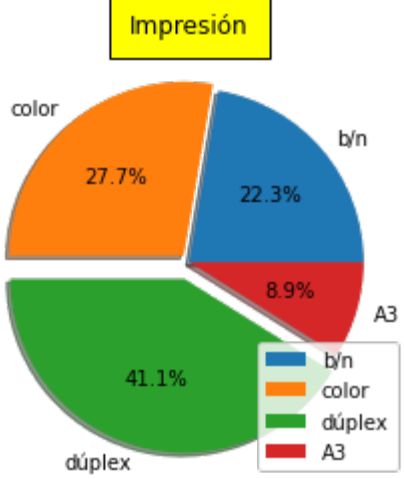
In [87]:
```
plt.figure()

x = np.random.randn(10000)  # Definimos un vector de números aleatorios de una distribución norma
plt.hist(x, bins = 25);  # Dibuja un histograma dividiendo el vector x en 20 intervalos
```



Gráficos de Torta

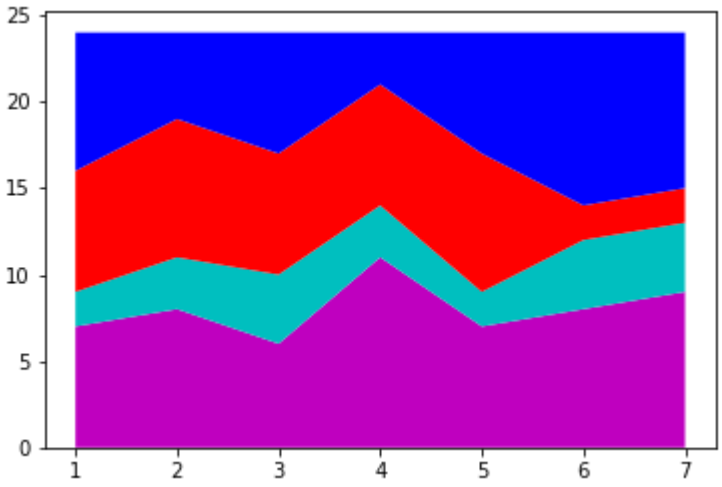
In [92]:
```
plt.figure()
impr = ["b/n", "color", "dúplex", "A3"]
vol = [25, 31, 46, 10]
expl =(0, 0.05, 0.1, 0)
plt.pie(vol, explode=expl, labels=impr, autopct='%1.1f%%', shadow=True)
plt.title("Impresión", bbox={"facecolor":"yellow", "pad":10})
plt.legend(loc='lower right')
```

Out[92]:
```
<matplotlib.legend.Legend at 0x1e5dc090d30>
```



Gráficos de Área (Apilados)

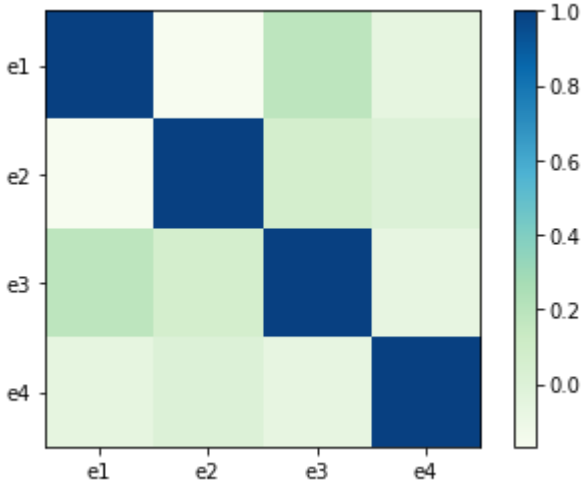
In [94]:
```
plt.figure()

dias = [1,2,3,4,5,6,7]

dormir =    [7, 8, 6, 11, 7, 8, 9]
comer =     [2, 3, 4, 3, 2, 4, 4]
trabajar =  [7, 8, 7, 7, 8, 2, 2]
jugar =     [8, 5, 7, 3, 7, 10, 9]

plt.stackplot(dias, dormir, comer, trabajar, jugar, colors=['m', 'c', 'r', 'b'])

plt.show()
```



Gráficos de Imágenes

In [95]:
```
# Crear unos datos
e1 = np.random.rand(100)
e2 = np.random.rand(100)*2
e3 = np.random.rand(100)*10
e4 = np.random.rand(100)*100
matrix_correlacion = np.corrcoef([e1, e2, e3, e4])
print(matrix_correlacion)

# Graficar la matriz de correlación como una imagen
plt.imshow(matrix_correlacion, interpolation='none', cmap='GnBu')
plt.colorbar()
var = ['e1', 'e2', 'e3', 'e4']
posicion = np.arange(len(var))
plt.yticks(posicion, var)
plt.xticks(posicion, var)
plt.show()
```

```
[[ 1.         -0.16729981  0.1864633  -0.06226559]
 [-0.16729981  1.          0.0664359   0.00425952]
 [ 0.1864633   0.0664359   1.         -0.06272994]
 [-0.06226559  0.00425952 -0.06272994  1.        ]]
```



Otras Librerías Gráficas

- **Seaborn**: es una biblioteca de visualización de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos.
- **bokeh**: es una biblioteca de visualización interactiva de Python orientada a navegadores web modernos para su presentación
- **plotly**: es una biblioteca de gráficos de Python que hace gráficos interactivos de calidad de publicación en línea

Seaborn

`Seaborn` es una librería de visualización de datos de Python basada en `matplotlib`. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.

In [100…
```
import seaborn as sns
tips = sns.load_dataset("tips")
import pandas as pd
tips.head()
```

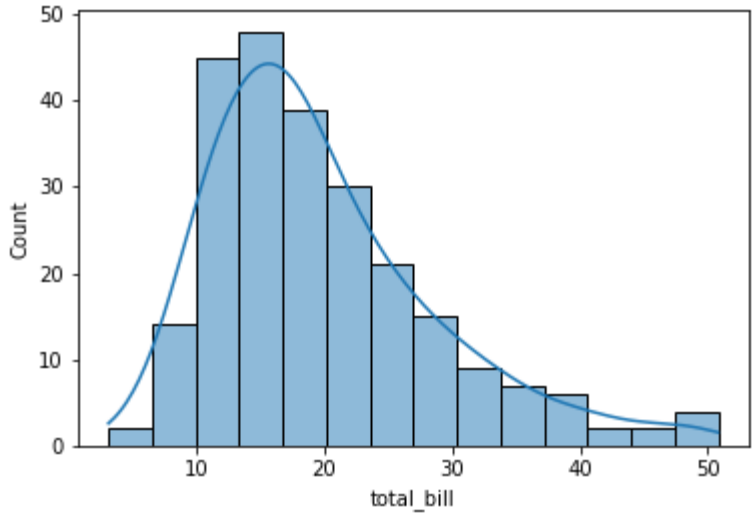
Out[100]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Los análisis estadísticos requieren conocimiento sobre la distribución de variables en su conjunto de datos. La función seaborn `displot()` admite varios enfoques para visualizar distribuciones. Estos incluyen técnicas clásicas como histogramas y enfoques computacionalmente intensivos como la estimación de la densidad del núcleo:

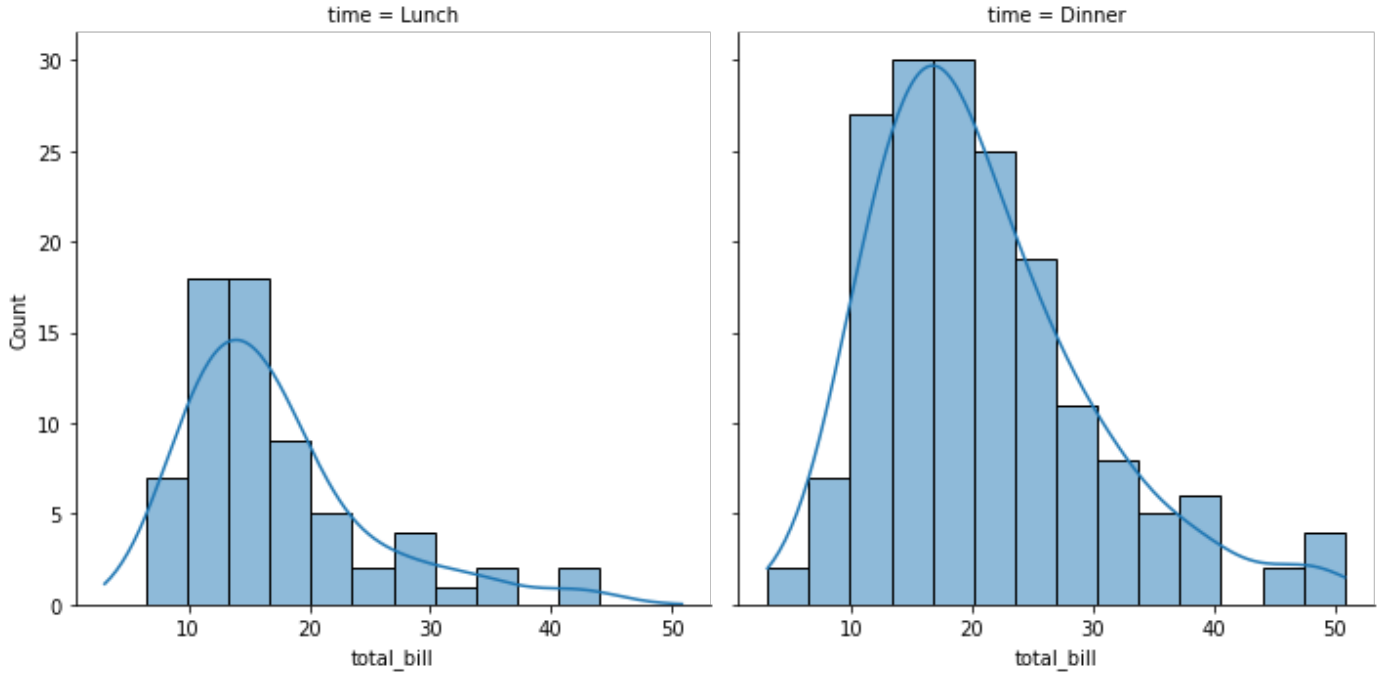
In [103…
```
sns.histplot(x='total_bill', data=tips, kde=True)
```

Out[103]:
```
<AxesSubplot:xlabel='total_bill', ylabel='Count'>
```



In [98]:
```
sns.displot(data=tips, x="total_bill", col="time", kde=True)
```

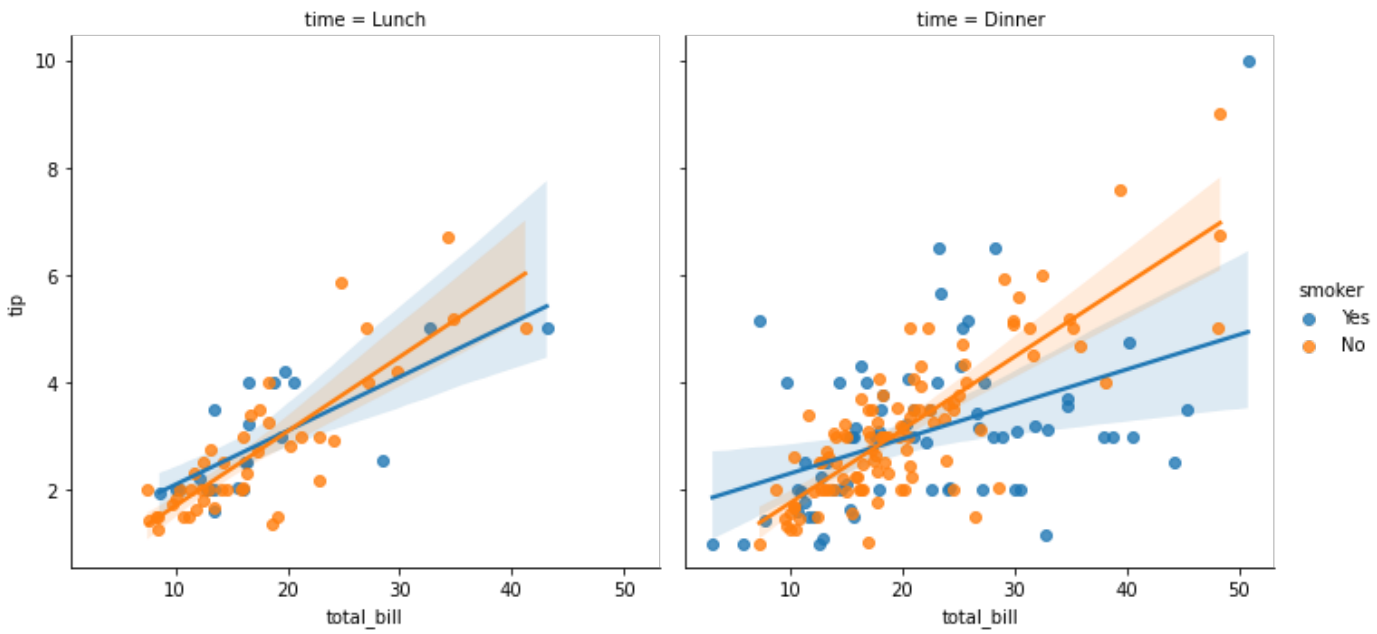
Out[98]:
```
<seaborn.axisgrid.FacetGrid at 0x1e5dc0de3a0>
```



La estimación estadística en seaborn va más allá de las estadísticas descriptivas. Por ejemplo, es posible mejorar un diagrama de dispersión al incluir un modelo de regresión lineal (y su incertidumbre) usando `lmplot()`:

In [106…
```
sns.lmplot(data=tips, x="total_bill", y="tip", col="time", hue="smoker")
```

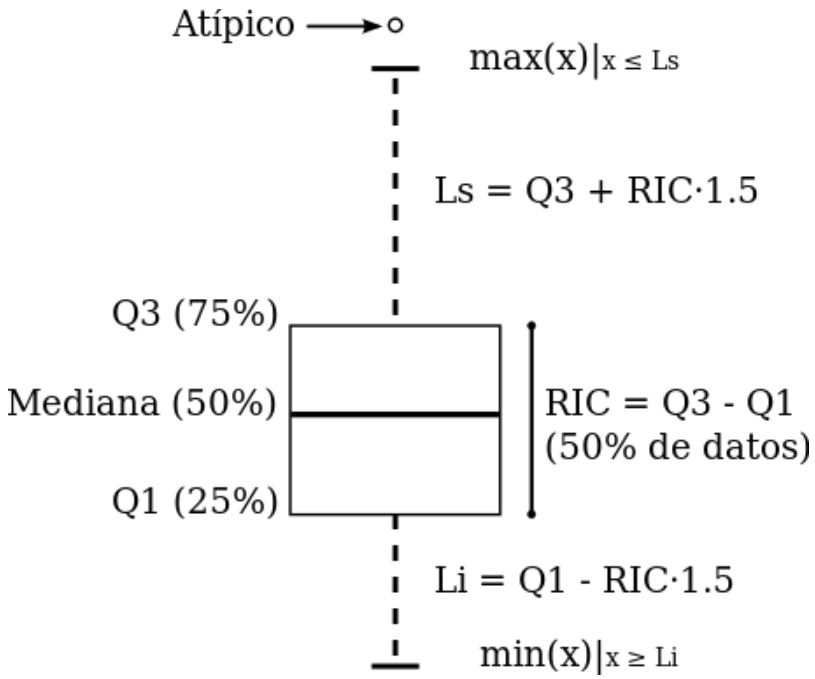
Out[106]:
```
<seaborn.axisgrid.FacetGrid at 0x1e5dfb37e80>
```



Gráficos especializados para datos categóricos

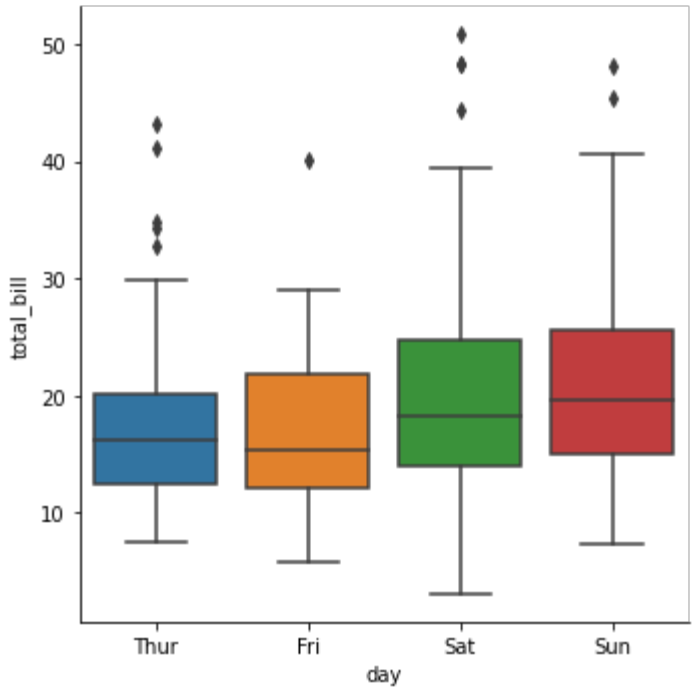
Varios tipos de gráficos especializados en seaborn están orientados hacia la visualización de datos categóricos. Se puede acceder a ellos a través de catplot().

Un diagrama de caja (también, diagrama de caja y bigotes o box plot) es un método estandarizado para representar gráficamente una serie de datos numéricos a través de sus cuartiles. De esta manera, se muestran a simple vista la mediana y los cuartiles de los datos,1 y también pueden representarse sus valores atípicos.



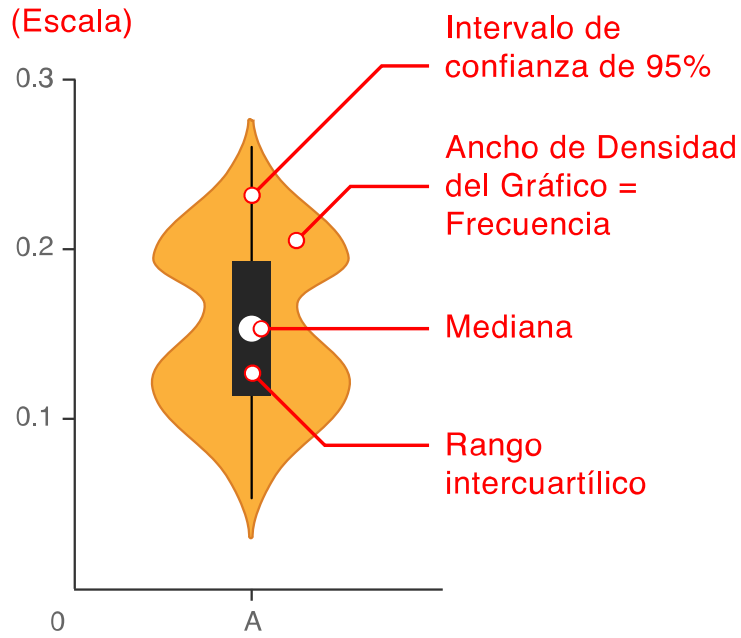
In [108…
```
sns.catplot(x="day", y="total_bill", kind="box", data=tips)
```

Out[108]:
```
<seaborn.axisgrid.FacetGrid at 0x1e5dfb9a280>
```



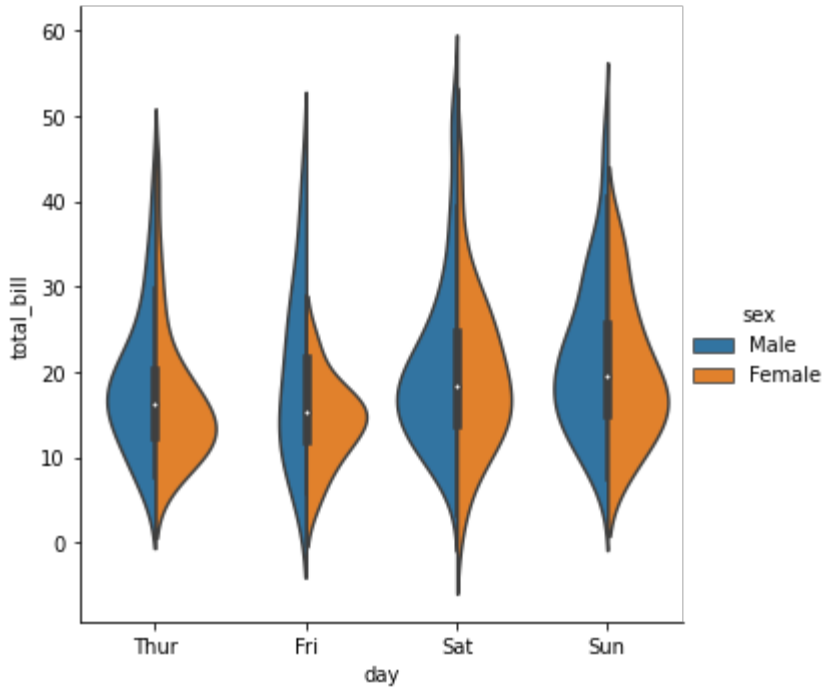
Un diagrama de violín se utiliza para visualizar la distribución de los datos y su densidad de probabilidad.

Este gráfico es una combinación de un diagrama de cajas y bigotes y un diagrama de densidad girado y colocado a cada lado, para mostrar la forma de distribución de los datos. La barra negra gruesa en el centro representa el intervalo intercuartil, la barra negra fina que se extiende desde ella, representa el 95 % de los intervalos de confianza, y el punto blanco es la mediana.



In [113…
```
sns.catplot(x="day", y="total_bill",  hue="sex", kind="violin", split=True, data=tips)
```

Out[113]:
```
<seaborn.axisgrid.FacetGrid at 0x1e5db7853a0>
```



Vistas compuestas sobre conjuntos de datos multivariantes

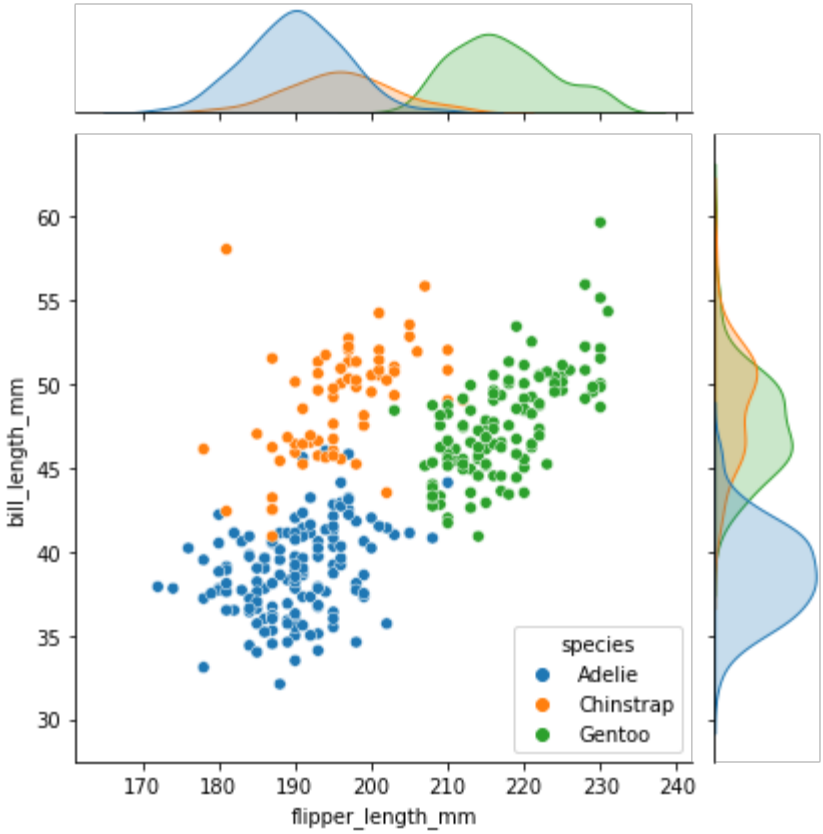
Algunas funciones de Seaborn combinan múltiples tipos de gráficos para brindar rápidamente resúmenes informativos de un conjunto de datos. `jointplot()` traza la distribución conjunta entre dos variables junto con la distribución marginal de cada variable:

In [115…

```
penguins = sns.load_dataset("penguins")
sns.jointplot(data=penguins, x="flipper_length_mm", y="bill_length_mm", hue="species")
penguins.head()
```

Out[115]:

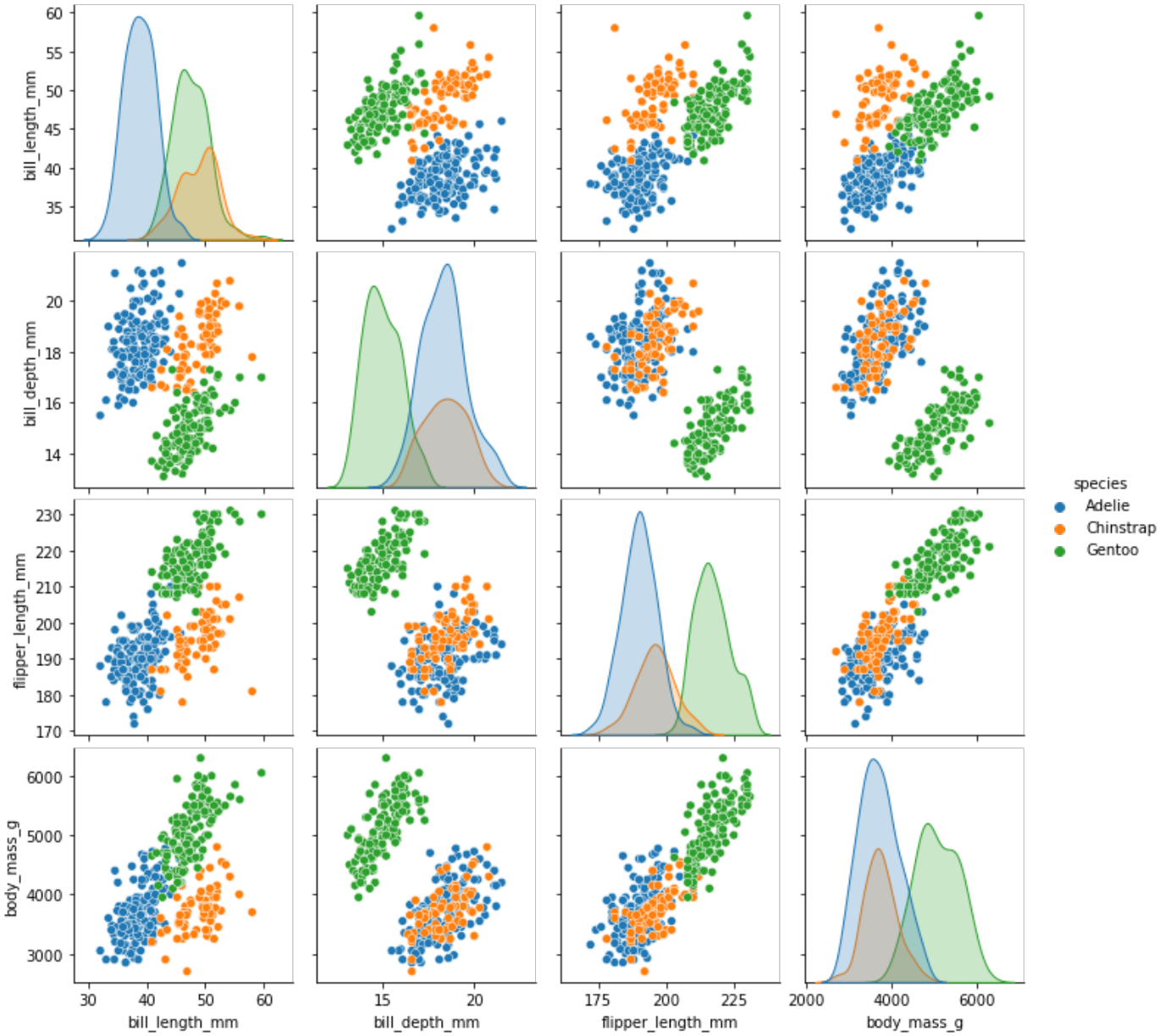
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female



`pairplot()`, tiene una visión más amplia: muestra distribuciones conjuntas y marginales para todas las relaciones por pares y para cada variable, respectivamente:

In [116…
```
sns.pairplot(data=penguins, hue="species")
```

Out[116]:
```
<seaborn.axisgrid.PairGrid at 0x1e5e15a25e0>
```



Referencias:

- Ejemplos de gráficos en Matplotlib
- Tutorial de Matplotlib de SciPy 2017
- Pyplot Tutorial de Matplotlib