a) Introducción

Matplotlib es una biblioteca para crear gráficas, tablas y figuras. También proporciona funciones para personalizar estas gráficas, modificando lo colores, etiquetas, etc.

Antes de utilizar matplotlib, debemos instalarlo con `pip install matplotlib` en la línea de comandos (si estamos trabajando en un entorno local, y siempre creando un entorno virtual para nuestro proyecto), y luego importarla en nuestro código:

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

pyplot es el módulo que dibuja las gráficas.

plt es el alias habitual para importar este módulo.

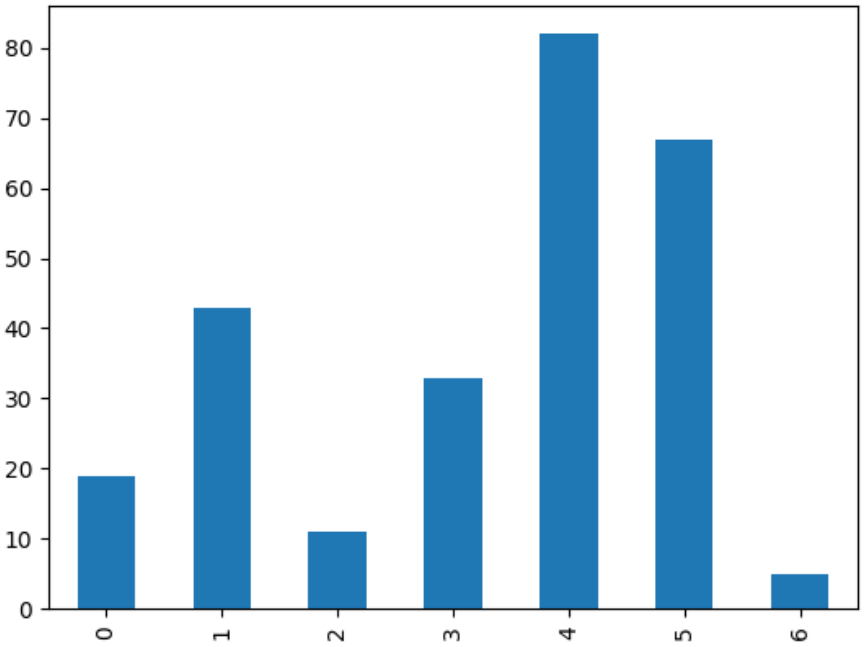
Matplotlib es el complemento ideal para trabajar con Pandas.

Veamos un ejemplo (debemos recordar importar pandas):

```
s=pd.Series([19,43,11,33,82,67,5])
s.plot(kind='bar')
plt.savefig('plot.png')
```

La función `plot()` crea la gráfica a partir de una serie o dataframe de pandas.

Como no lo especificamos de otra forma, los datos se muestran en el eje y, mientras que en el eje x se muestran los índices de los datos.



`plt.savefig('plot.png')` graba un archivo con la imagen.

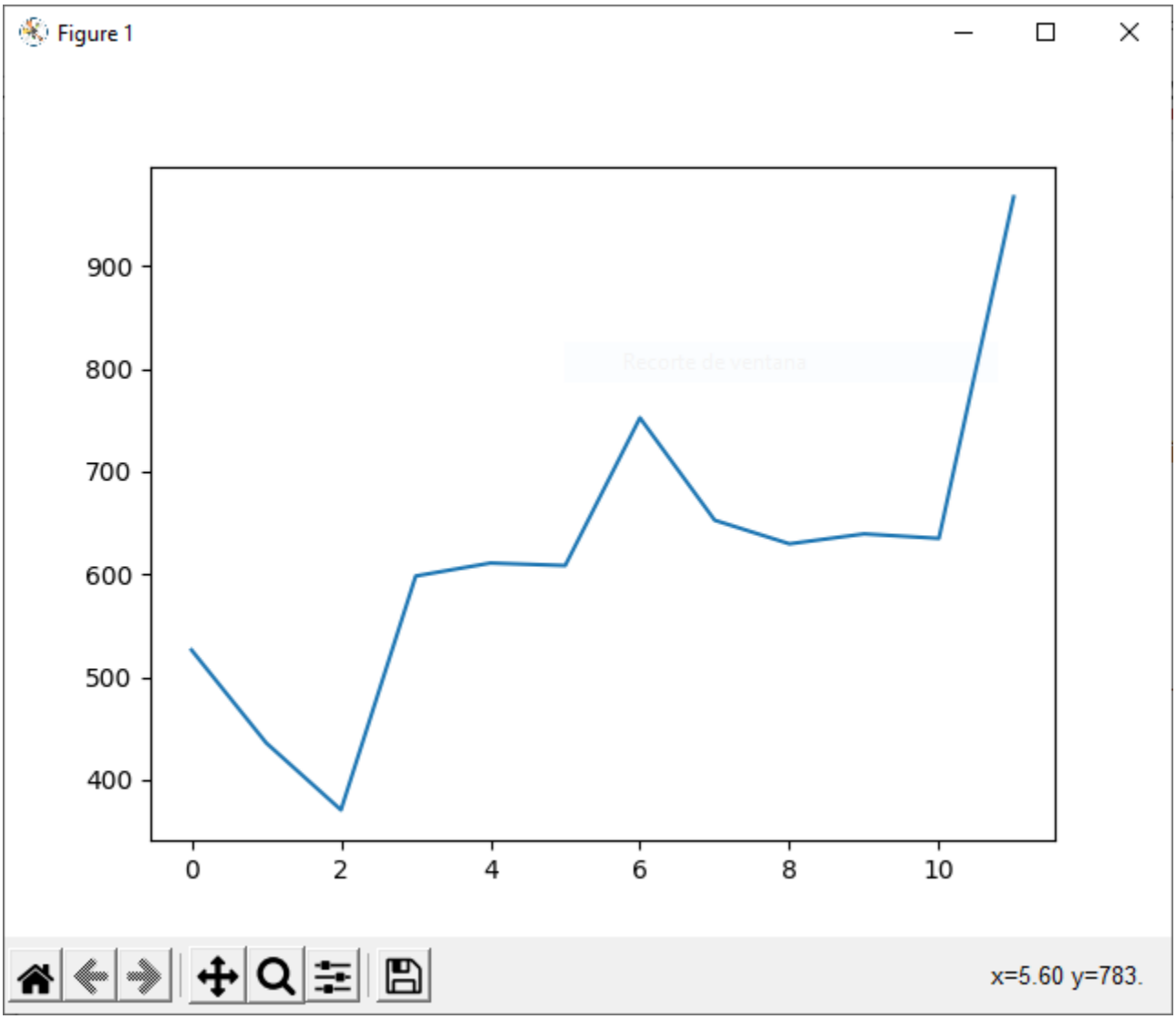
.b) Gráfica de líneas

Matplotlib soporta diferentes tipos de gráficas. La más básica es la gráfica de líneas.

Tomemos como ejemplo un dataframe extraído del archivo *'balance_2021.csv'* y grafiquemos las ventas del año. Podrás encontrar el archivo en el material de la plataforma.

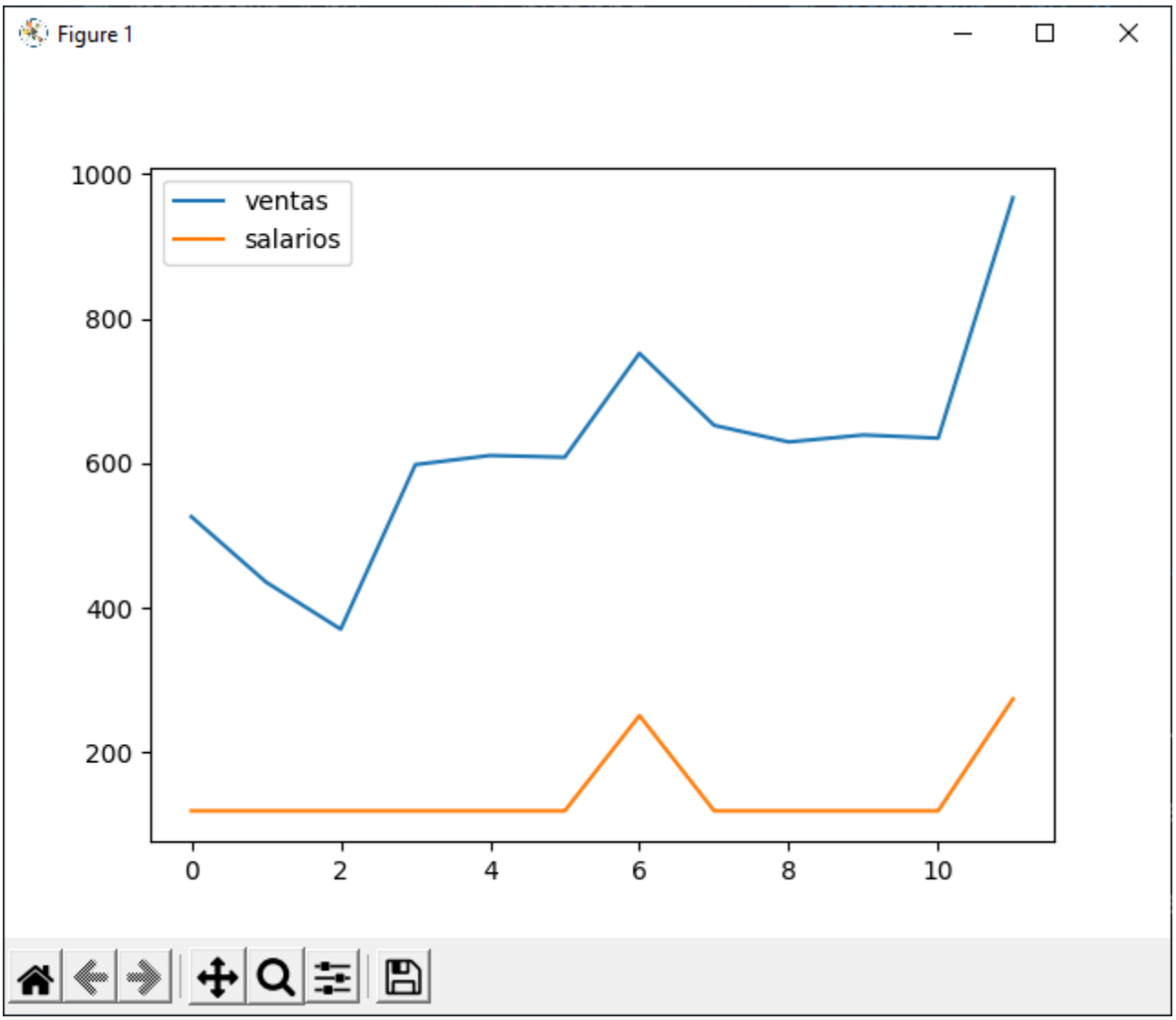
```
df = pd.read_csv('balance_2021.csv')
df['ventas'].plot()
plt.show()
```

Si no deseamos grabar la gráfica en un archivo, y sólo verlo, podemos usar la función **show()**.



También podemos graficar múltiples líneas en la misma gráfica. Probemos mostrar los ventas y salarios al mismo tiempo:

```
df['ventas','salarios'].plot()
```



Matplotlib agrega título y color automáticamente a las líneas que representan los datos de cada columna.

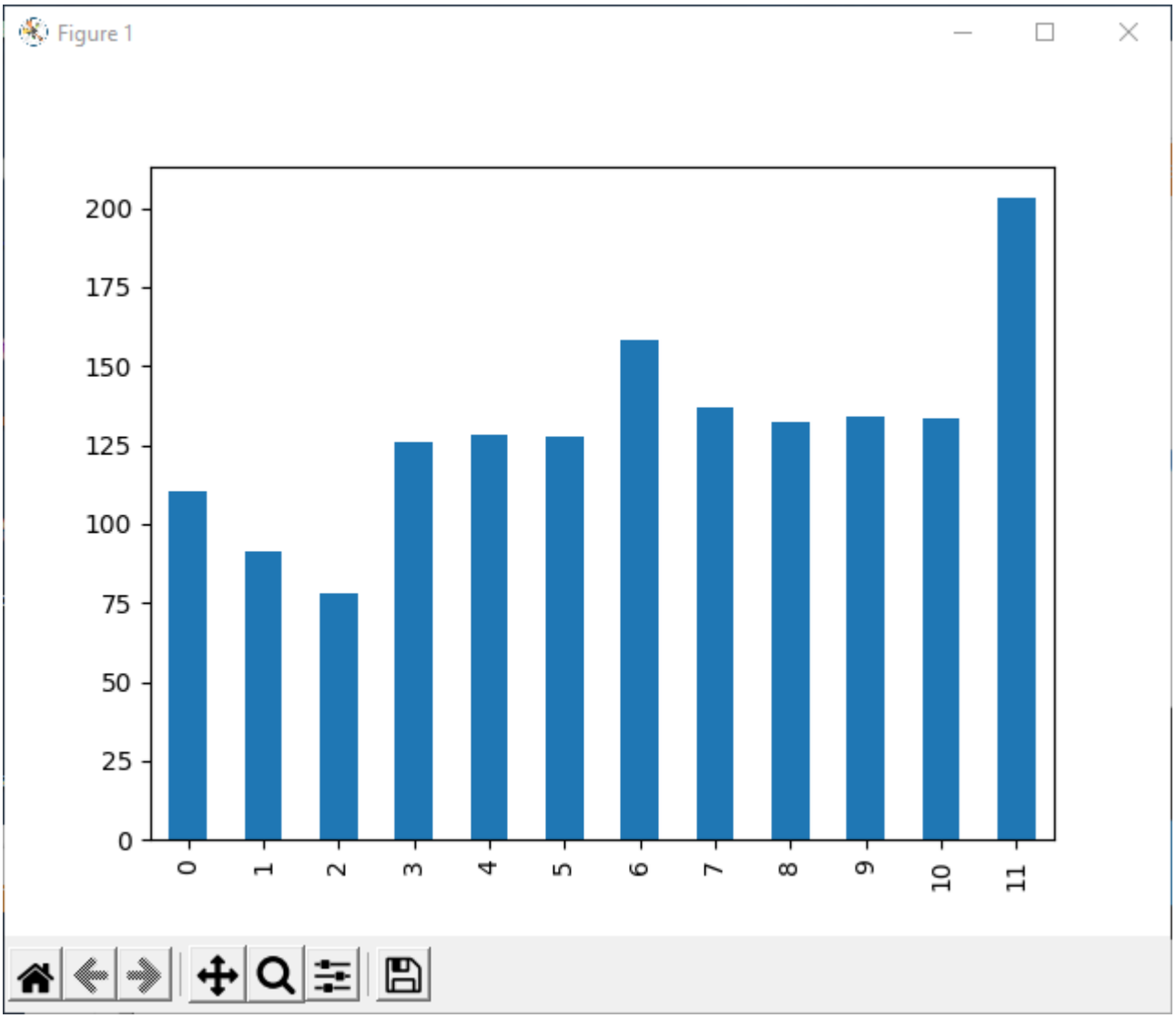
.b) Gráfica de barras

La función **plot()** admite argumentos en los que podemos especificar el tipo de gráfico que queremos obtener.

Para crear una gráfico de barras, utilizamos el atributo **kind** con el valor **'bar'**.

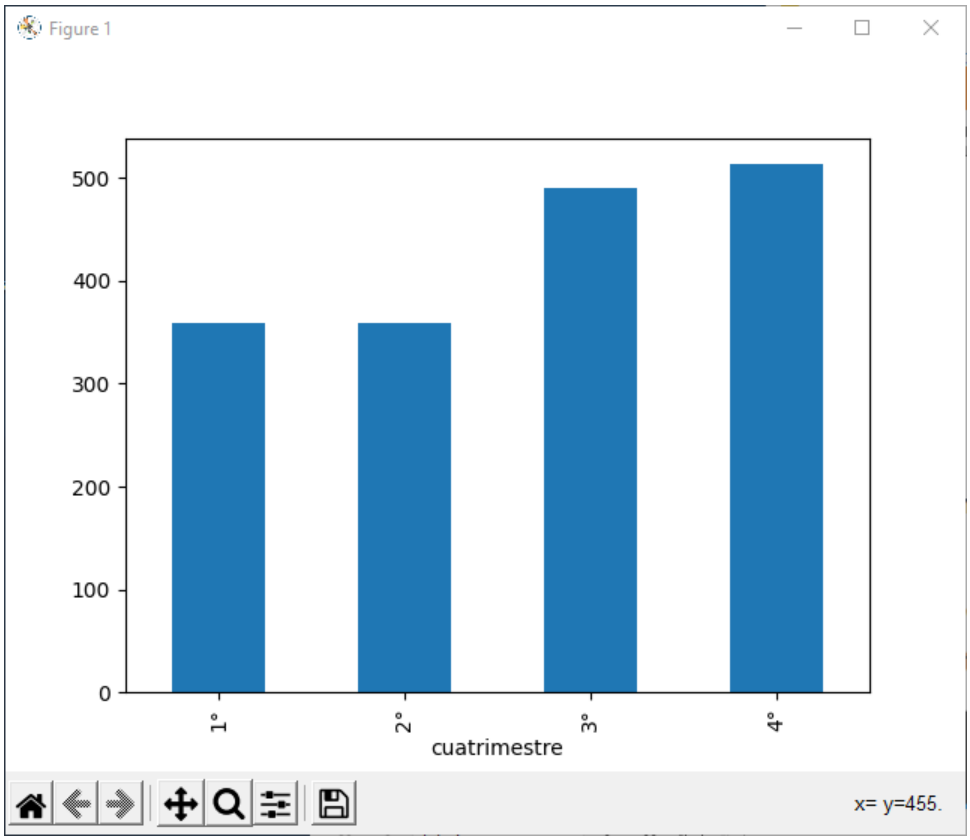
Vamos a obtener una gráfica con los impuestos pagados en 2021 a partir del csv del ejemplo anterior.

```
df['impuestos'].plot(kind='bar')
```



Podemos combinar matplotlib con las funciones de pandas. Veamos los salarios por cuatrimestre:

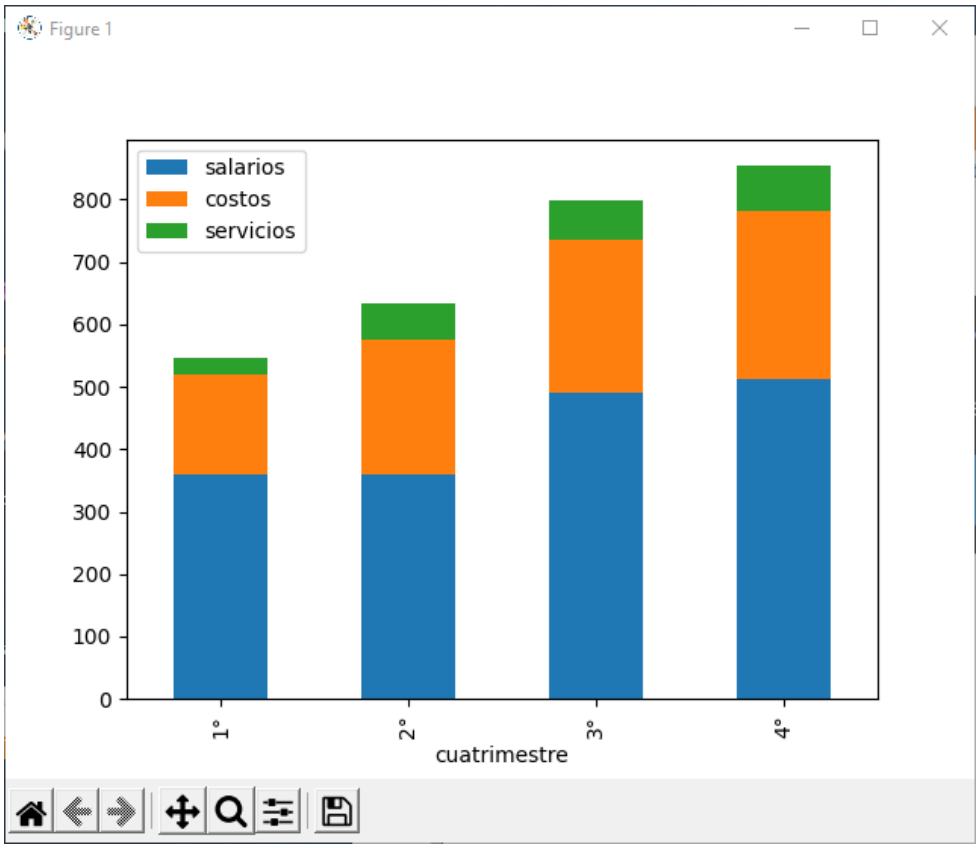
```
(df.groupby('cuatrimestre')['salarios'].sum()).plot(kind='bar')
```



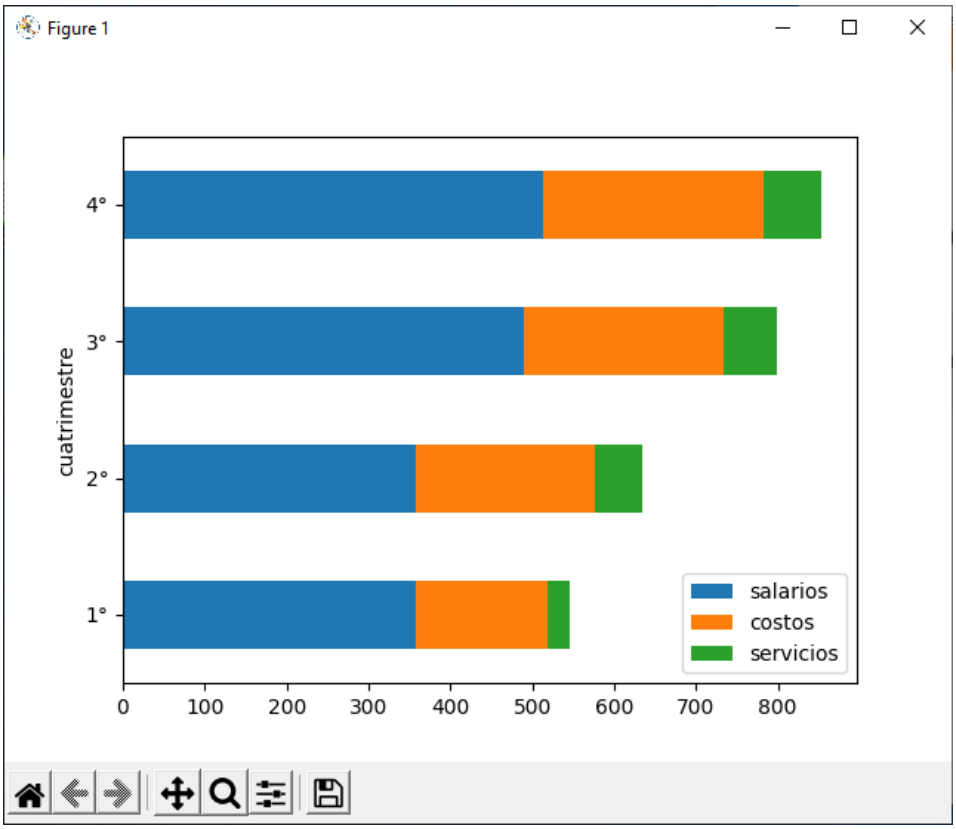
Primero agrupamos los salarios por cuatrimestre, y calculamos la suma en cada uno de los períodos.

También podemos dibujar múltiples columnas. Si utilizamos la propiedad **stacked,** obtendremos las barras apiladas.

```
df = df.groupby('cuatrimestre')[['salarios','costos','servicios']].sum()
df.plot(kind='bar', stacked=True)
```



Si modificamos el atributo **kind** a **‘barh’**, obtenemos barras horizontales.



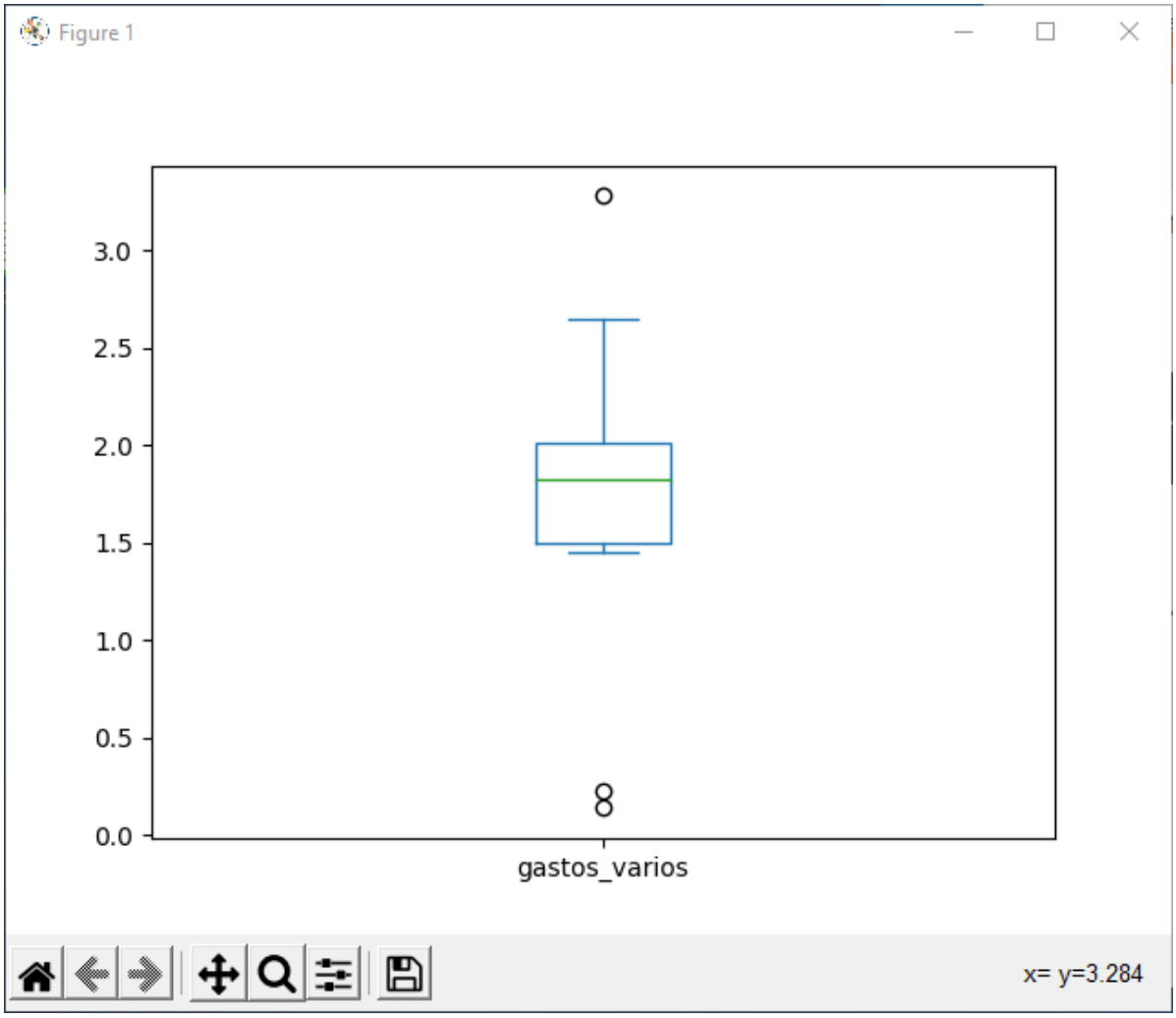
.c) Gráfica de caja y bigotes

Nos permite visualizar la distribución de los valores en una columna. Presenta un resumen estadístico de nuestro dataframe. Específicamente nos grafica el resultado de la función **describe()** de pandas.

Por ejemplo, la gráfica de los costos en nuestro dataframe de balance.

```
print(df['gastos_varios'].describe())
df['gastos_varios'].plot(kind='box')
```

```
count    12.000000
mean      1.703333
std       0.875093
min       0.140000
25%       1.502500
50%       1.825000
75%       2.010000
max       3.280000
Name: gastos_varios, dtype: float64
```



- La línea verde representa la mediana.
- La caja representa el rango intercuartil.

- Las líneas verticales que salen de la caja, abarcan los datos mínimos a máximo, excluyendo los outliers.
- Los círculos muestran los outliers o datos anómalos.

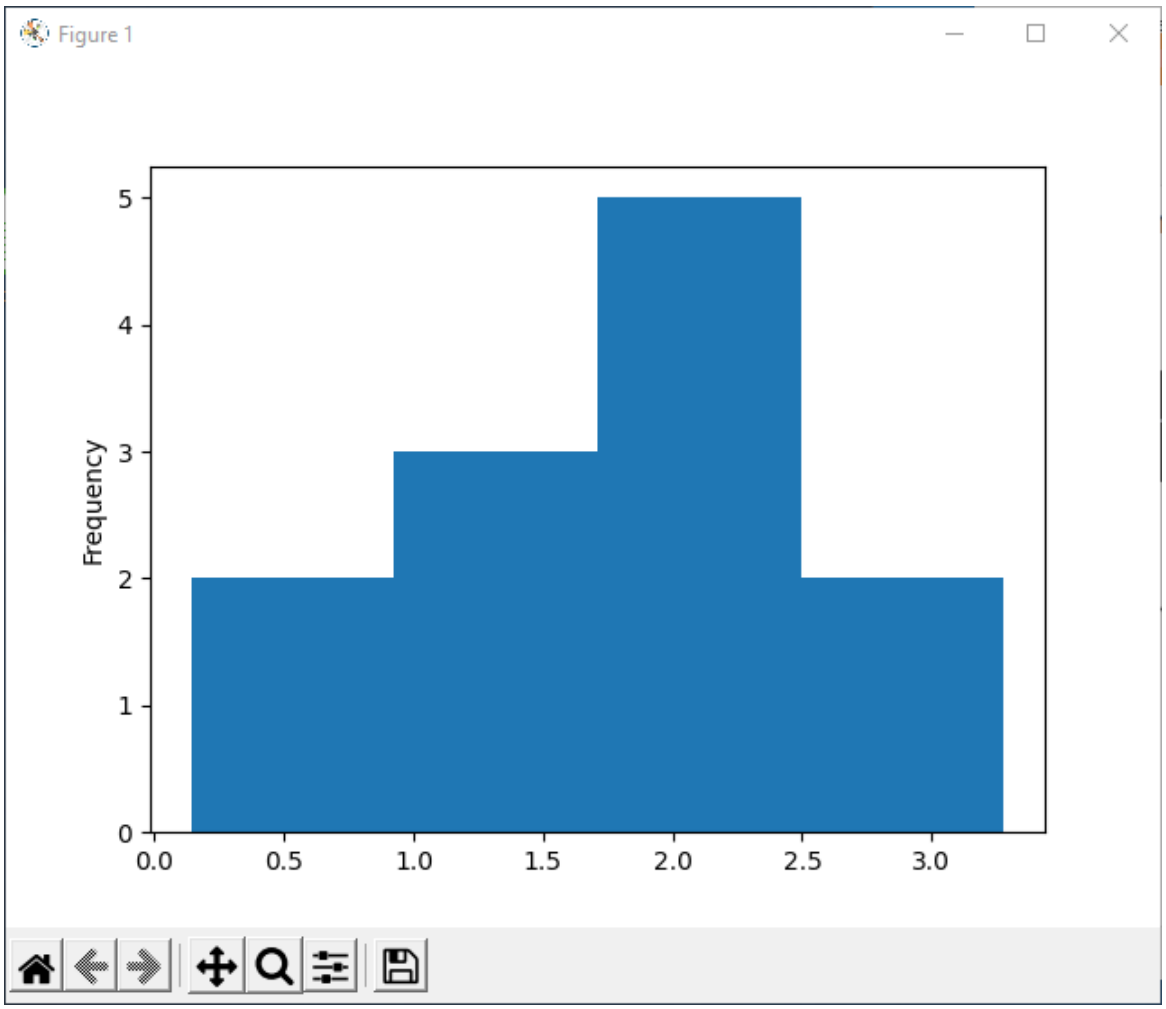
.d) Histograma

Así como los diagramas de caja y bigotes, los histogramas también muestran la distribución de los datos.

Visualmente son similares a los gráficos de barras, pero al representar frecuencia en un grupo o rango de datos, no hay espacios entre las barras.

Obtengamos los datos de los gastos varios, divididos en 4 columnas. Especificamos el atributo **kind** como **'hist'** y el número de columnas con el atributo **bins**:

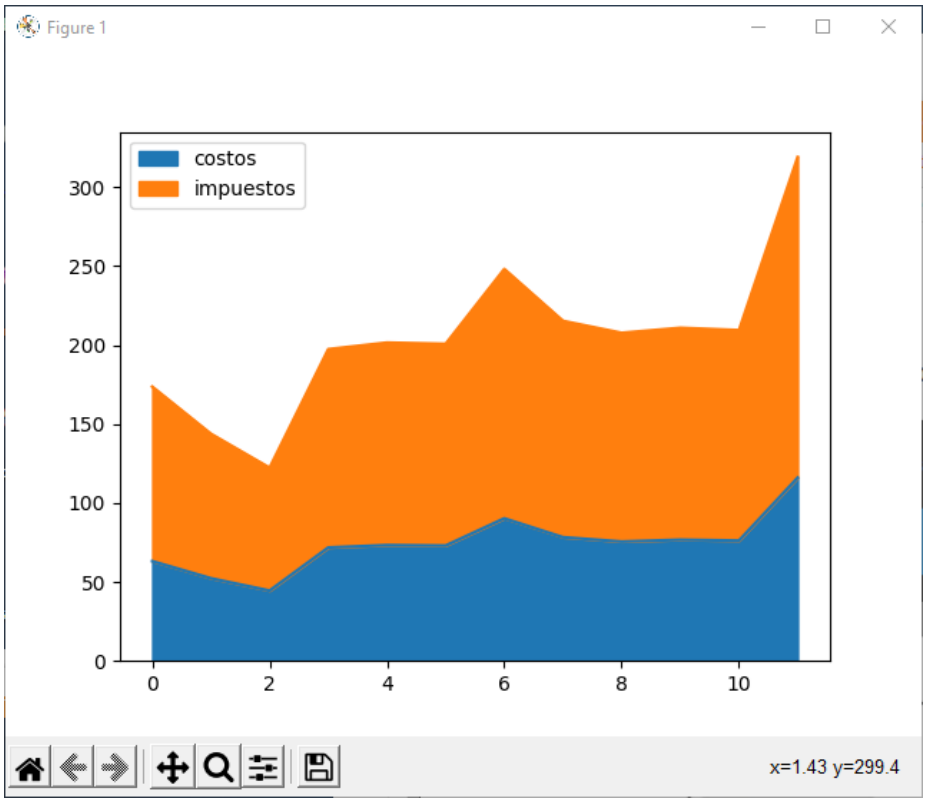
```
df['gastos_varios'].plot(kind='hist',bins=4     )
```



.e) Gráfica de área

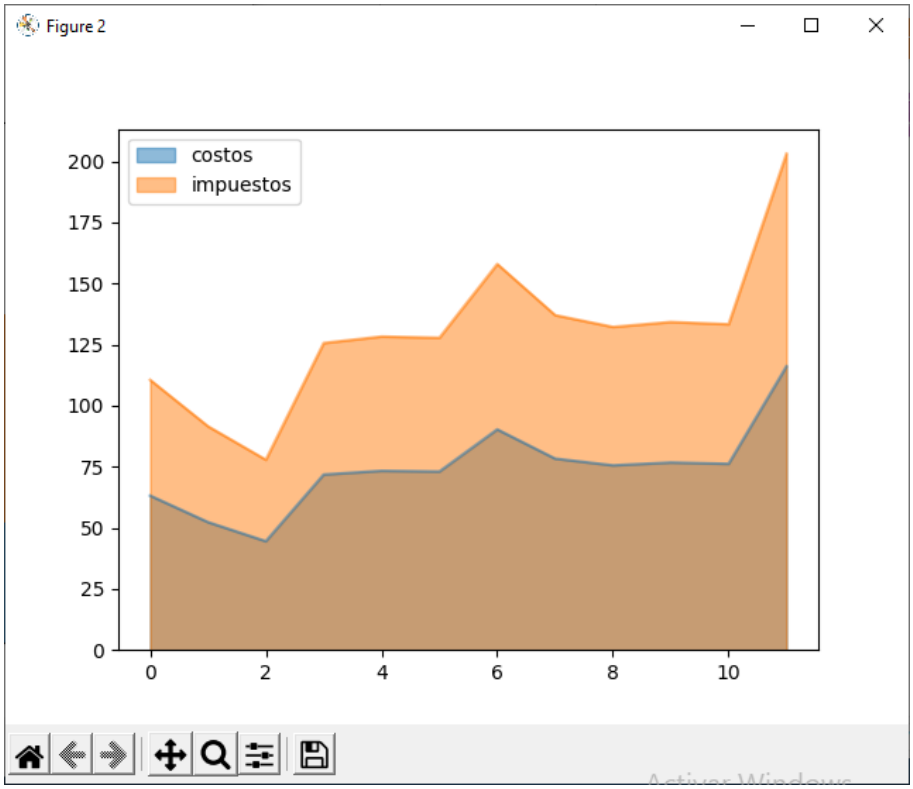
El atributo **kind='area'** crea una gráfica de área.

```
df[['costos','impuestos']].plot(kind='area')
```



Las gráficas de área se muestran apiladas por defecto, por lo que debemos explicitar si no lo deseamos de esa manera con el atributo **stacked = False**.

```
df[['costos','impuestos']].plot(kind='area', stacked = False)
```

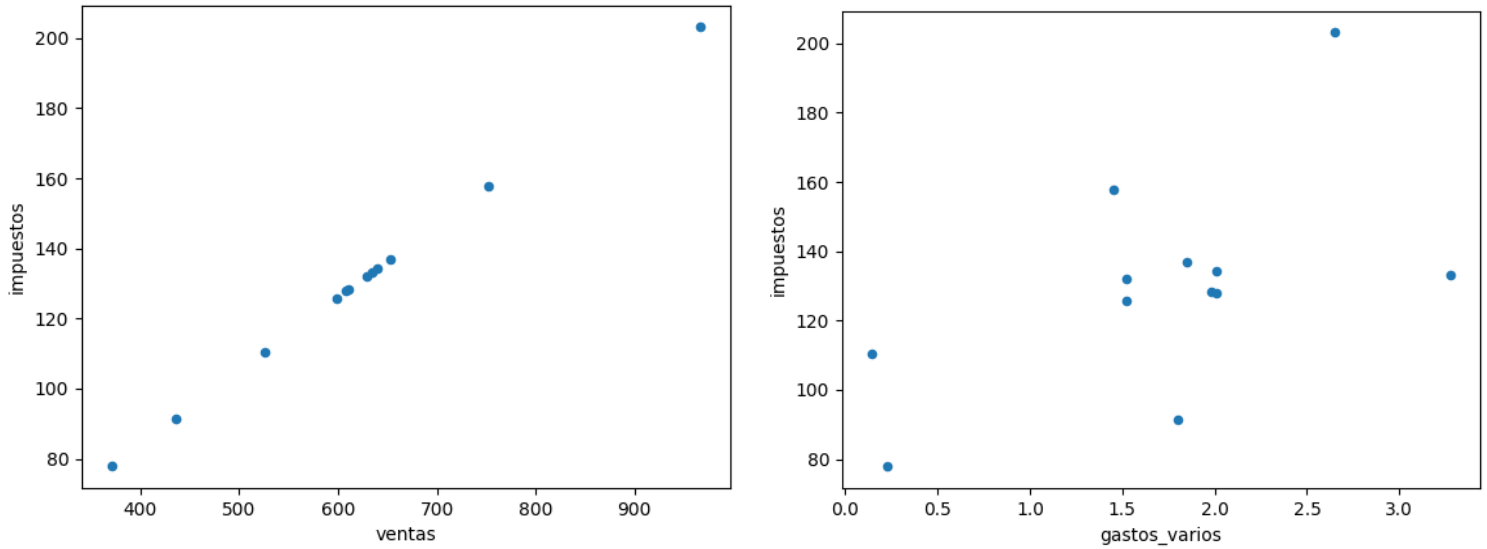


.f) Diagrama de dispersión

El diagrama de dispersión nos muestra la relación entre dos variables.

Por ejemplo, podemos ver cómo están relacionados los impuestos con las ventas y con los gastos varios, creando dos gráficas diferentes con **kind='scatter'** y especificando cómo queremos ubicar los ejes:

```
df[['ventas','impuestos']].plot(kind='scatter', x='ventas', y='impuestos')
df[['gastos_varios','impuestos']].plot(kind='scatter', x='gastos_varios', y='impuestos')
```



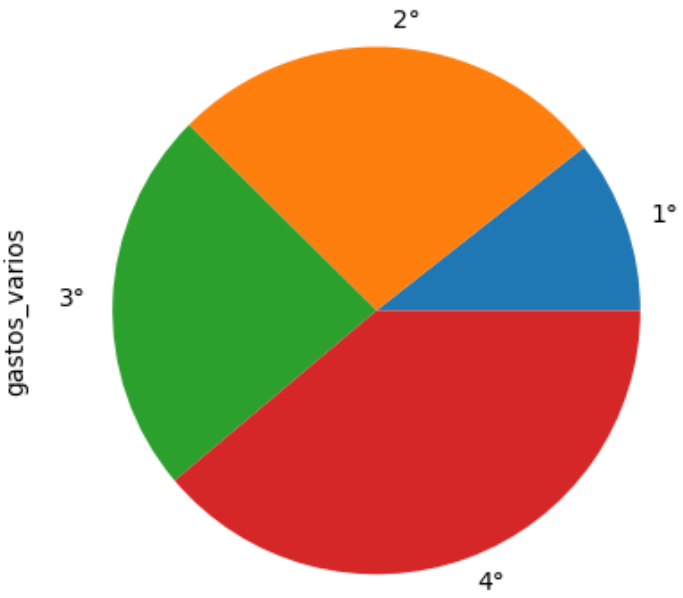
Ambas gráficas muestran 12 puntos, 1 por cada dato (mes).

.g) Pie chart

Creamos gráficos de torta con el atributo **kind = 'pie'**.

Veamos el promedio de gastos varios por cuatrimestre:

```
df.groupby('cuatrimestre')['gastos_varios'].mean().plot(kind='pie')
```



Generalmente utilizamos pie charts para mostrar porcentajes o proporcionalidad entre los datos.

Se recomienda usarlos cuando tenemos hasta un máximo de 6 categorías.

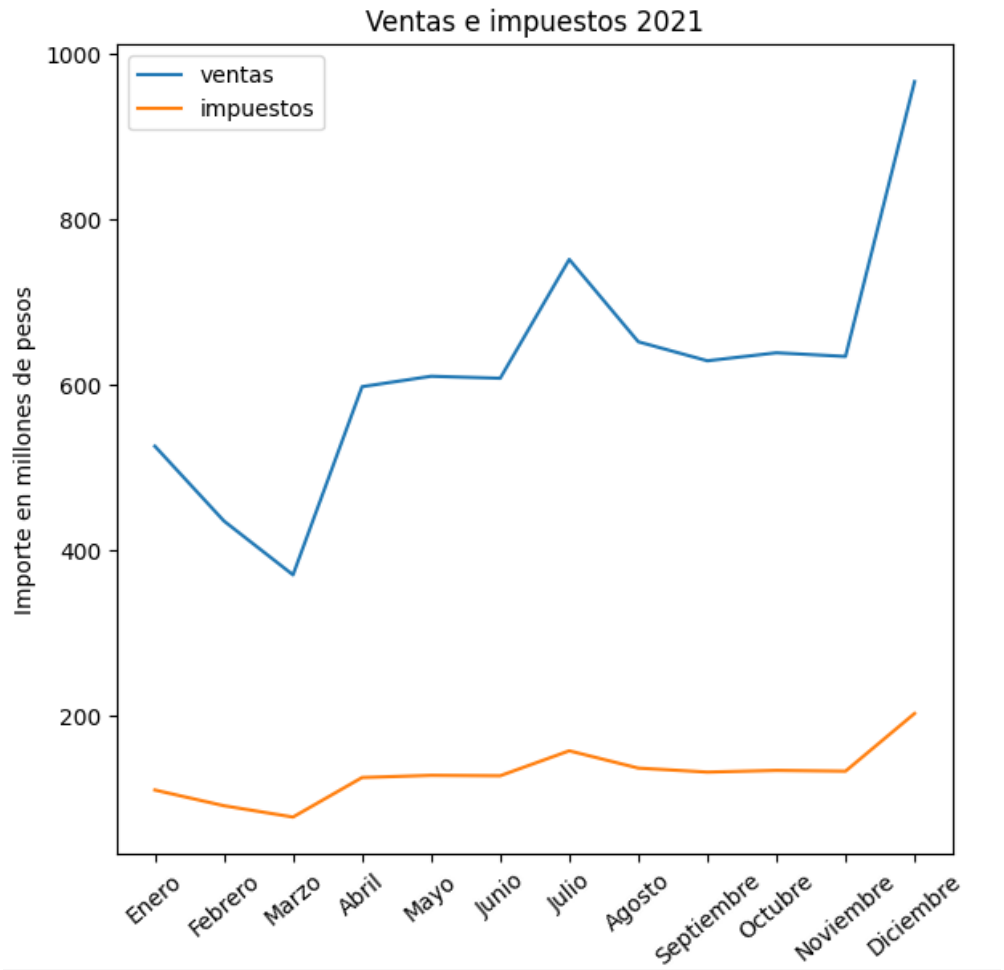
.h) Formato

Matplotlib facilita una serie de argumentos para personalizar nuestras gráficas.

- **title=cadena_de_texto** le pone un título a la gráfica
- **legend=True|False:** especifica si deseamos mostrar el recuadro con la especificación de colores para las líneas.
- **xlabel(valor) / ylabel(valor)** especifican las etiquetas para los ejes correspondientes.
- **xticks(posiciones,valores)** permite cambiar los valores sobre el eje x. Si el texto a mostrar es muy ancho, podemos aplicarle una rotación con el argumento **rotation=x** (en grados). Por defecto se utilizan los índices de los datos de ese eje.

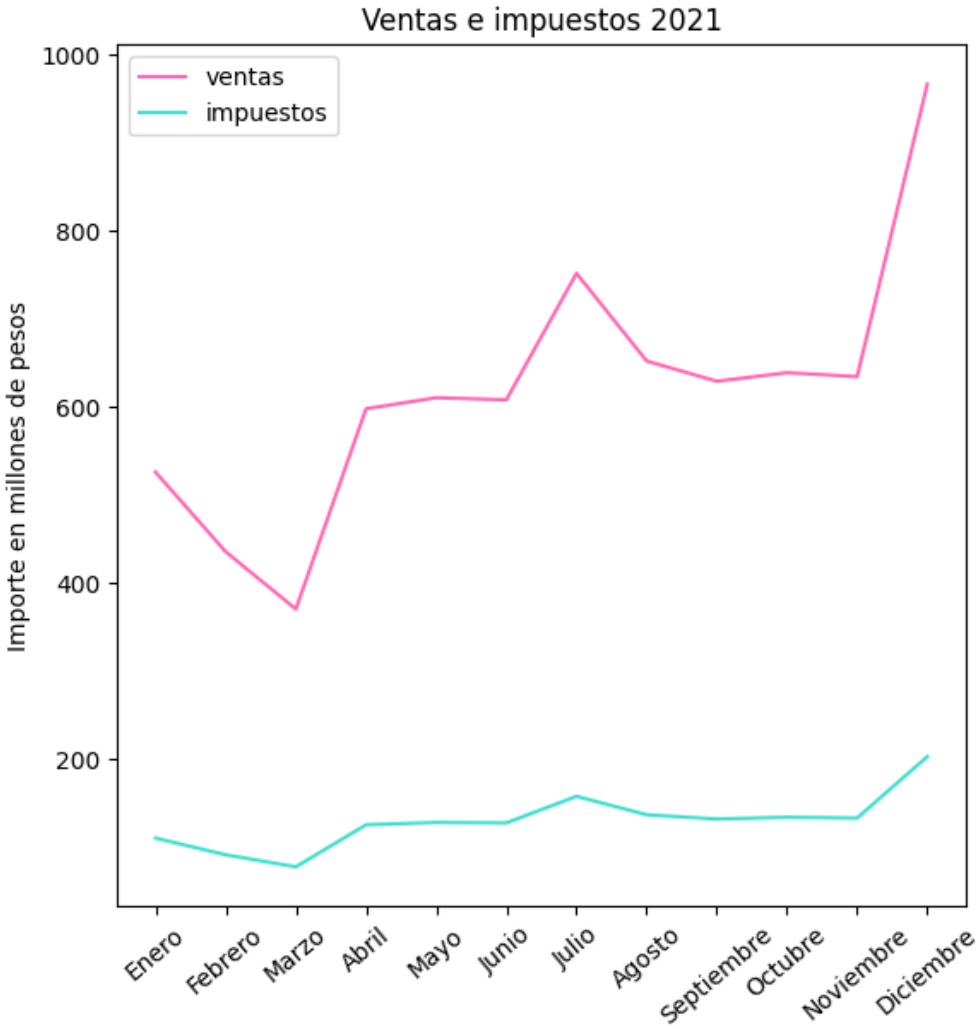
Veamos un primer ejemplo con estas personalizaciones:

```
df[['ventas','impuestos']].plot(kind='line',legend=True,    title='Ventas e impuestos 2021')

posiciones = np.arange(0,12)

plt.xticks(posiciones, df['mes'], rotation=40)

plt.xlabel('Mes')

plt.ylabel('Importe en millones de pesos')
```



También podemos cambiar los colores con el atributo color. Acepta valores tipo string (el nombre del color), un carácter (para los colores básicos), hexadecimal, hexadecimal abreviado, RGB, RGBA, etc.

Por ejemplo, agregando `color=['hotpink','turquoise']` dentro de los atributos de **plot()**, obtenemos:



Todos estos atributos aplican a la mayoría de los tipos de gráficas que hemos visto.

En la siguiente tabla obtenida de la documentación de matplotlib pueden ver distintos colores y tipos de línea:

character	color
‘b’	blue
‘g’	green
‘r’	red
‘c’	cyan
‘m’	magenta
‘y’	yellow
‘k’	black
‘w’	white

character	description
`'-'`	solid line style
`'--'`	dashed line style
`'-.'`	dash-dot line style
`':'`	dotted line style
`'.'`	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

1 Introducción

Seaborn es una biblioteca de visualización de datos de Python basada y que complementa a matplotlib . Esto significa que es un complemento, no un reemplazo.

Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.

Algunas de sus características son:

- temas predefinidos
- visualización de datos de una o dos variables
- manejo de gráficos de serie temporal
- integra perfectamente estructuras de datos de NumPy y PanDas

Para poder utilizarla en un entorno local, debemos instalarla previamente con `pip install seaborn`. Debemos recordar crear un entorno virtual para nuestro proyecto. Luego podremos importarla en nuestro código:

```
import seaborn as sns
```

sns es el alias que habitualmente utilizamos al importar seaborn.

a) Métodos de matplotlib que se reutilizan en seaborn

```
plt.figure()
plt.title()
plt.legend()
plt.show()
```

b) Configuración de estilos

La librería seaborn permite personalizar el estilo del gráfico, la paleta de colores que se utiliza, la fuente y el tamaño de los diferentes integrantes de la visualización, así como definir un contexto en el que se va a utilizar el gráfico para adaptar el tamaño.

```
sns.set_style()
sns.set_palette()
sns.set_context()
```

2 Explorando Seaborn

Seaborn aporta diversos datasets para explorar sus funciones. Una de ellas es ‘iris’. Lo utilizaremos para ejemplificar esta unidad:

```
datos = sns.load_dataset( ‘iris’ )
print(datos.head())
```

```
   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

Podemos explorar los diversos datasets que seaborn ofrecen con la función **get_dataset_names()**.

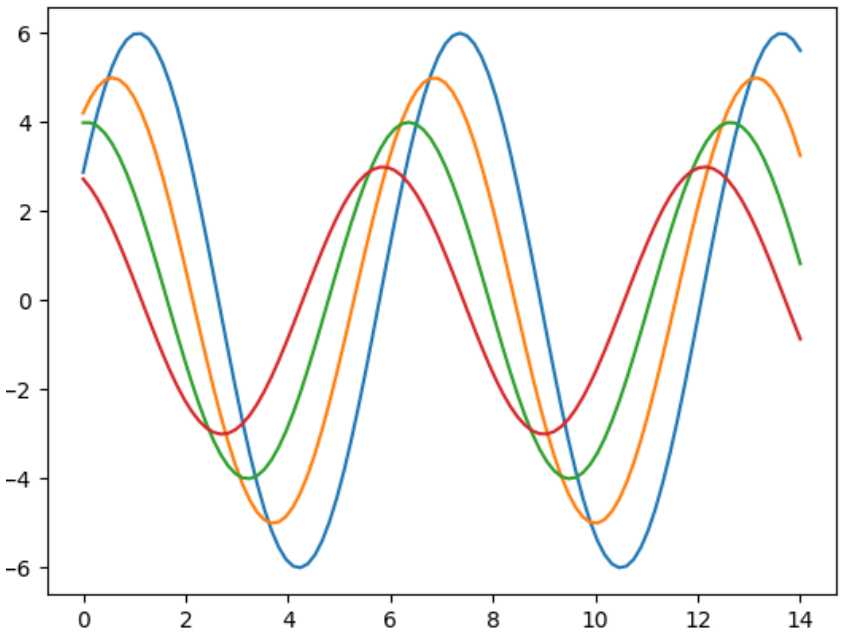
```
print(sns.get_dataset_names())
```

```
['anagrams', 'anscombe', 'attention', 'brain_networks', 'car_crashes',
'diamonds', 'dots', 'exercise', 'flights', 'fmri', 'gammas', 'geyser', 'iris',
'mpg', 'penguins', 'planets', 'taxis', 'tips', 'titanic']
```

3 Matplotlib vs Seaborn

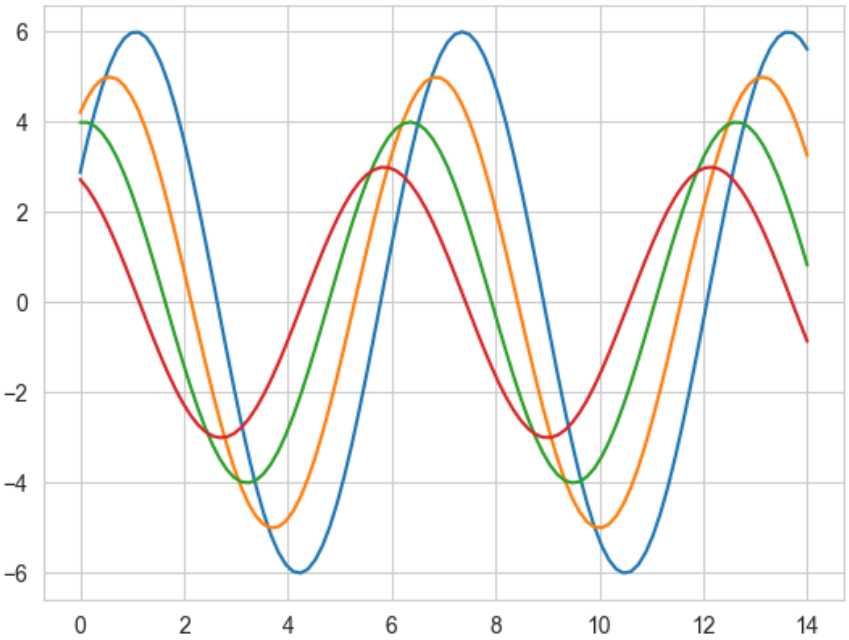
Probemos visualizar la función trigonométrica seno con ambas librerías.

```
# MATPLOTLIB
x = np.linspace(0, 14, 100)
for i in range(1, 5):
    plt.plot(x, np.sin(x + i * .5) * (7 - i) * 1)
plt.show()
```



Para aplicar los temas de seaborn, basta con agregar previamente la línea:

```
sns.set_style('whitegrid')
# opciones:darkgrid, whitegrid, dark, white, ticks
```



4 Personalización

a) Estilo

Podemos pasar un diccionario de parámetros para modificar los valores por defecto de la función **set_style()**.

Podemos explorar estas opciones y sus valores por defecto por medio de la función axes_styles().

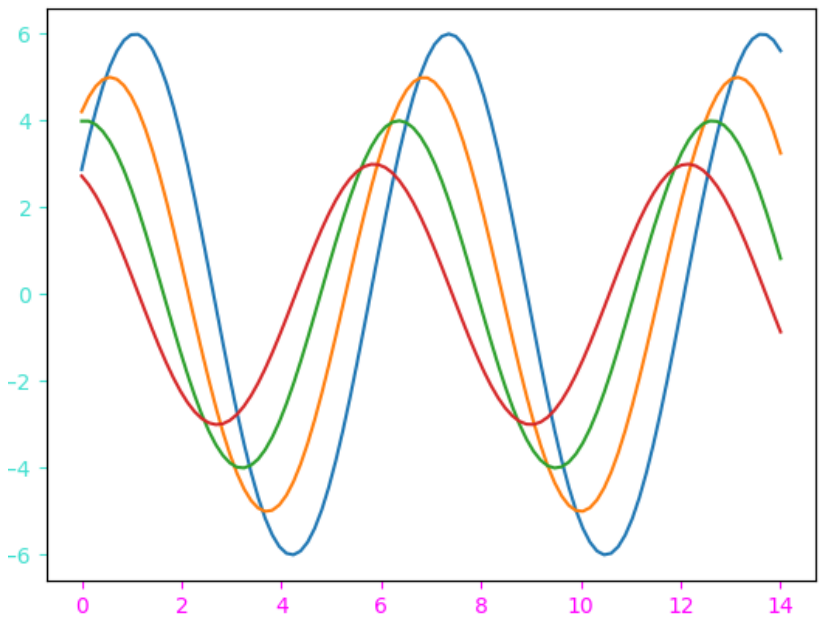
```
print(sns.axes_style())
```

```
{
'axes.facecolor': 'white',
'axes.edgecolor': '.8',
'axes.grid': True,
'axes.axisbelow': True,
'axes.labelcolor': '.15',
'figure.facecolor': 'white',
'grid.color': '.8',
'grid.linestyle': '-',
'text.color': '.15',
'xtick.color': '.15',
'ytick.color': '.15',
'xtick.direction': 'out',
```

```
'ytick.direction': 'out',
'lines.solid_capstyle': <CapStyle.round: 'round'>,
'patch.edgecolor': 'w',
'patch.force_edgecolor': True,
'image.cmap': 'rocket',
'font.family': ['sans-serif'],
'font.sans-serif': ['Arial', 'DejaVu Sans', 'Liberation Sans', 'Bitstream Vera Sans',
'sans-serif'], 'xtick.bottom': False,
'xtick.top': False,
'ytick.left': False,
'ytick.right': False,
'axes.spines.left': True,
'axes.spines.bottom': True,
'axes.spines.right': True,
'axes.spines.top': True
}
```

Entonces, para variar los colores del texto en las escalas, bastaría con agregar el diccionario de opciones:

```
sns.set_style({'xtick.color':'magenta','ytick.color':'turquoise'})
```

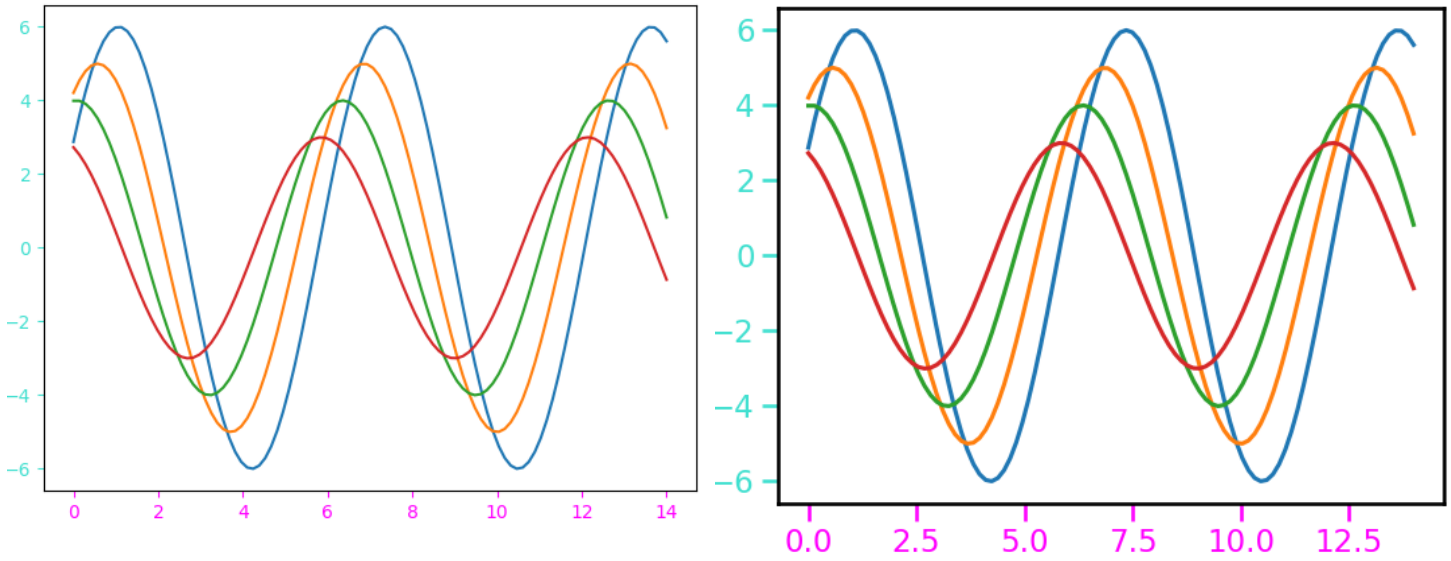


b) Escala

Utilizando la función **set_context()** podemos utilizar cualquiera de las siguientes plantillas: paper, notebook, talk, poster. El valor por defecto es notebook.

Comparemos la visualización anterior con el modo 'talk':

```
sns.set_context('talk')
```



c) Colores

Seaborn provee varias funciones para modificar los colores de nuestras gráficas. Veremos un par de ejemplos.

Podrás encontrar más información sobre los colores y Seaborn en los enlaces externos proporcionados en esta unidad.

La paleta de colores de seaborn consta de 10 matices diferentes, ordenados de la misma manera que la paleta de matplotlib, de la que tiene 6 variaciones:

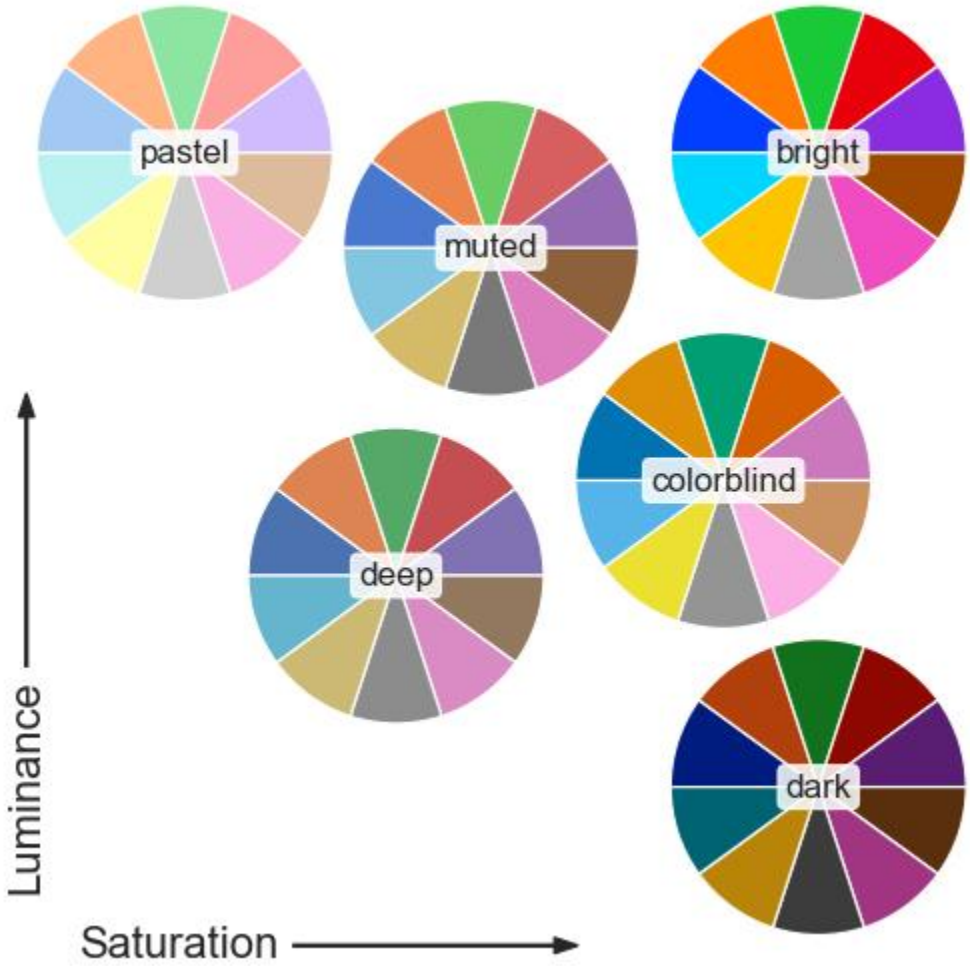
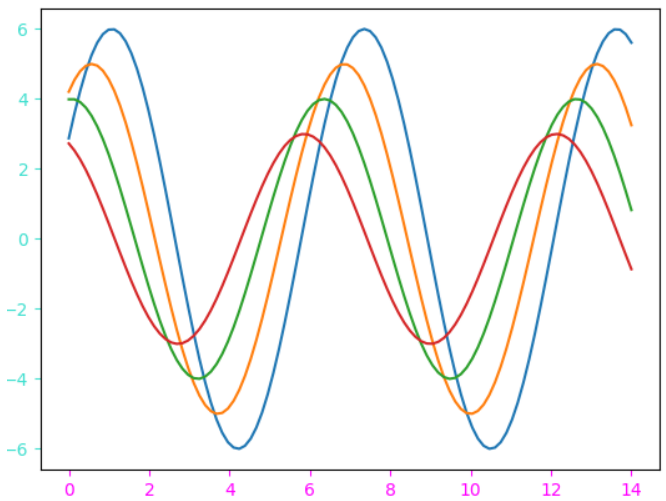
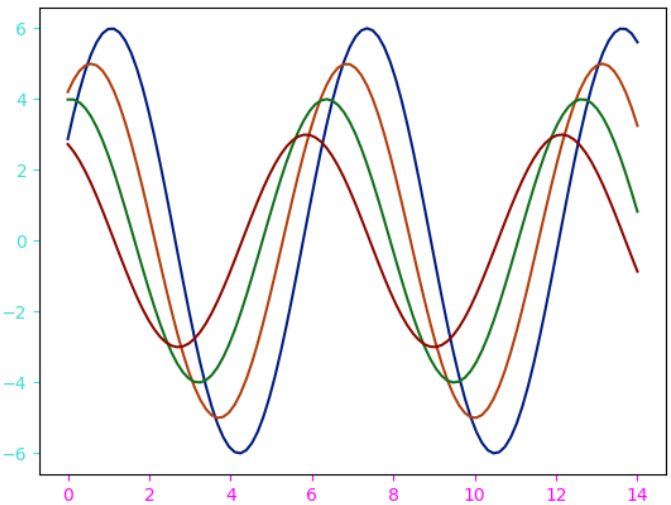


Imagen: https://seaborn.pydata.org/tutorial/color_palettes.html

Para aplicar cualquiera de estas variaciones a nuestras gráficas, utilizamos, por ejemplo:

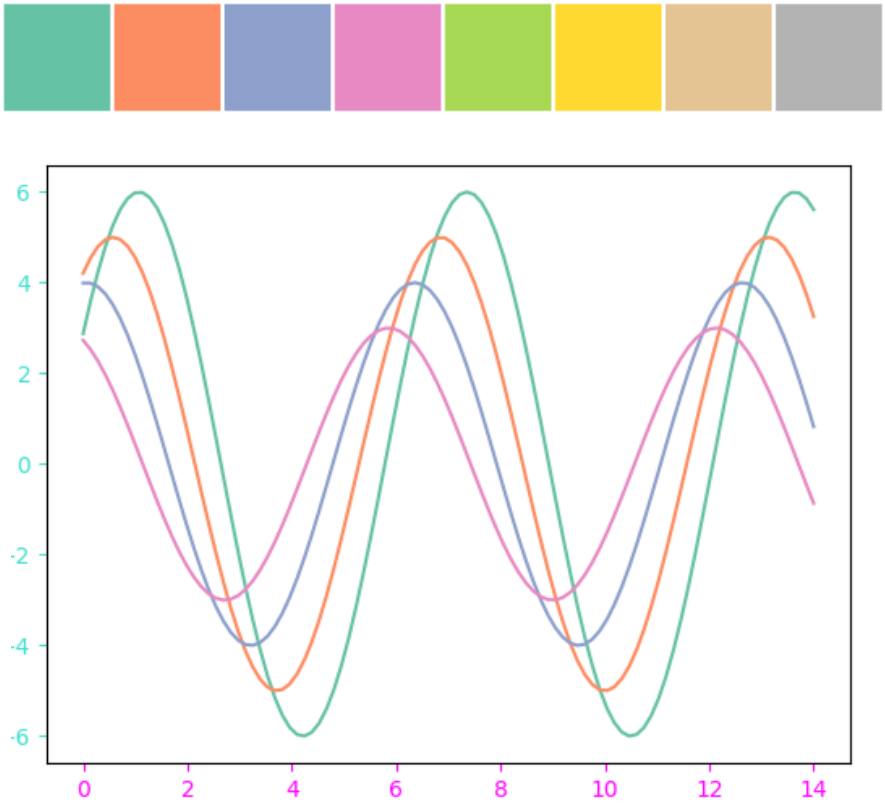
```
sns.set_palette('dark')
```



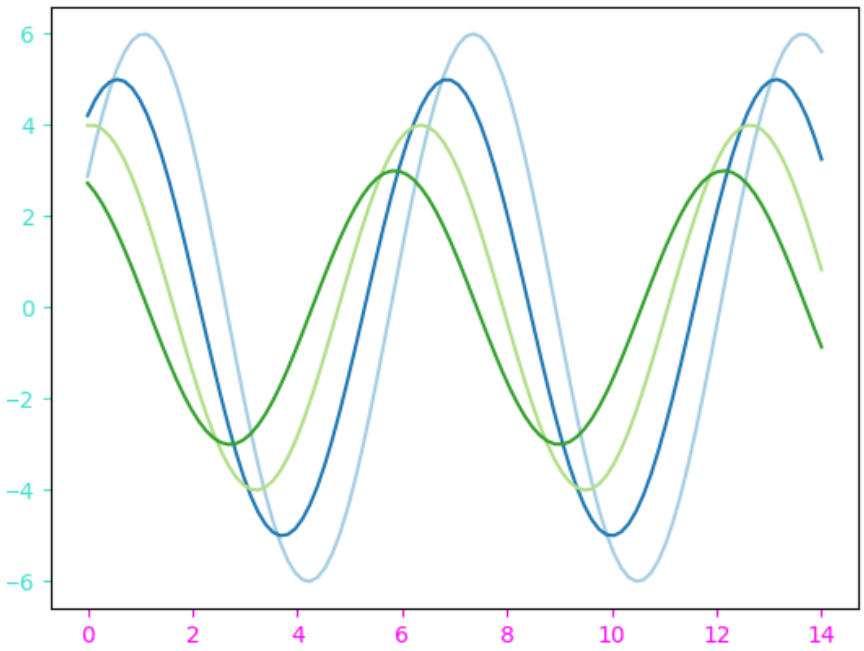


También tenemos disponibles una serie de paletas de colores:

sns.set_palette('Set2')



sns.set_palette('Paired')


6
4
2
0
−2
−4
−6
0
2
4
6
8
10
12
14

5 Diagrama de dispersión

Es la mejor manera de visualizar la distribución en dos variables numéricas representadas en una gráfica de dos dimensiones (ejes x/y)

Sintaxis:

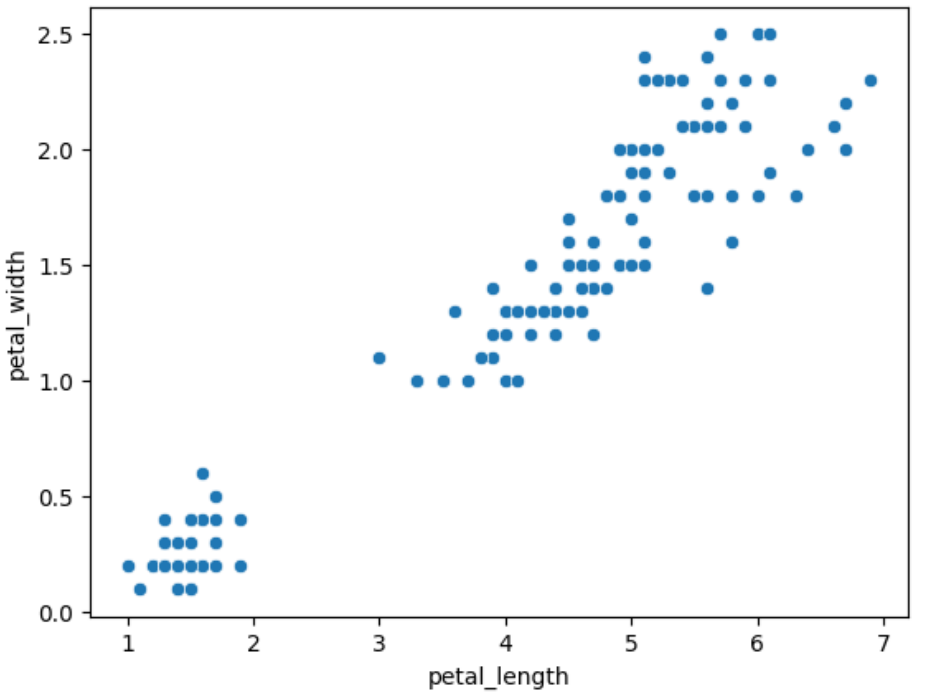
```
seaborn.scatterplot()
```

Parámetros:

- x: variable a comparar
- y: la otra variable
- data: serie, vector o lista

Ejemplo:

```
sns.scatterplot(x='petal_length', y='petal_width', data = datos)
```



6 Histogramas

Importaremos el dataset 'iris' de Seaborn para este capítulo.

Los histogramas representan la distribución de los datos por medio de barras que abarcar rangos de estos datos.

Sintaxis:

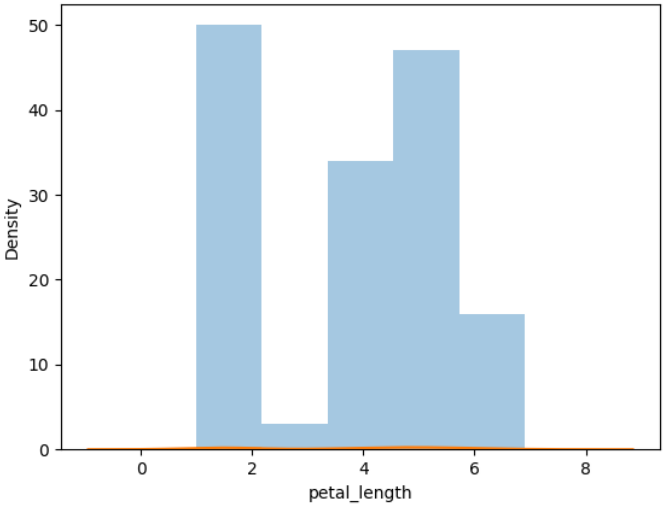
```
seaborn.distplot()
```

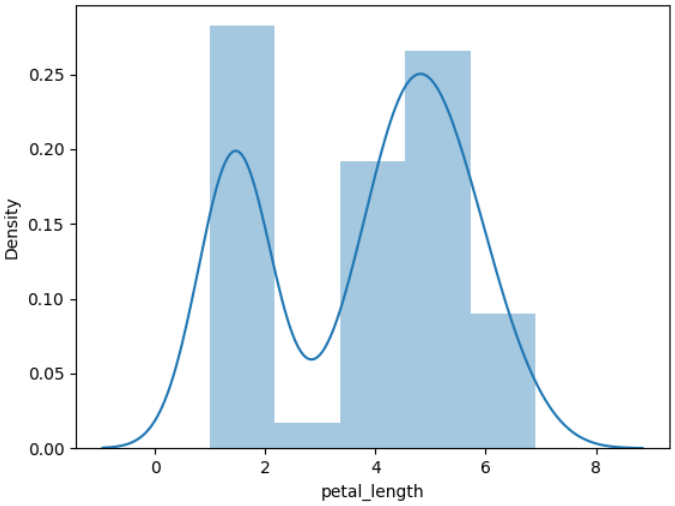
Parámetros:

- data: serie, vector o lista
- bins: entero (cantidad de fragmentos)
- hist: booleano (muestra las barras. Default: True)
- kde: booleano (muestra las líneas. Default: True)

Ejemplos:

```
sns.distplot(datos['petal_length'], kde=False)
sns.distplot(datos['petal_length'], kde=True)
```





a) Histograma de dos variables

Se utiliza para determinar la relación entre dos variables (cómo se comporta una respecto de la otra)

Para ello utilizamos la función joinplot(), que crea una figura en varios paneles, que muestra la distribución de cada variable por separado, y al mismo tiempo la relación entre ambas.

Sintaxis:

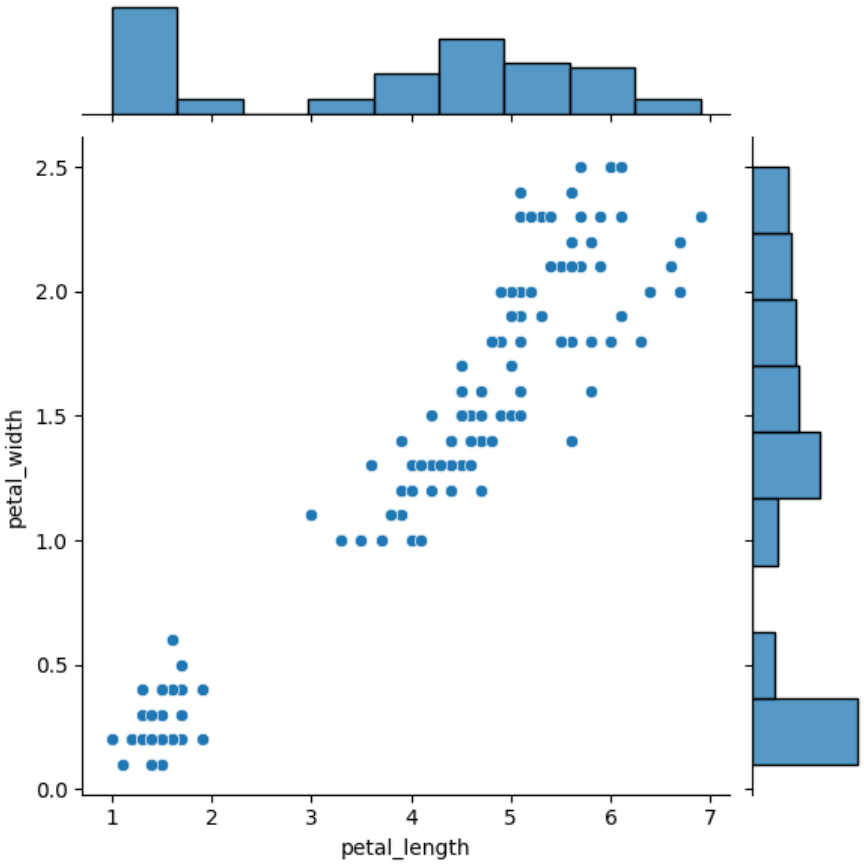
```
seaborn.jointplot()
```

Parámetros:

- x: variable a comparar
- y: la otra variables
- data: serie, vector o lista

Ejemplo:

```
sns.jointplot(x='petal_length', y='petal_width', data = datos)
```



7 Diagramas de barras

Nos ofrece una estimación de los valores de tendencia central, resumiendo la distribución de los datos. Promedio y mediana son las técnicas más utilizadas para estimar al tendencia central de la distribución.

a) Barplot

Muestra la relación entre una variable categórica y una variable continua. Los datos están representados en barras rectangulares, y la longitud de la barra representa la proporción de los datos en esa categoría.

Sintaxis:

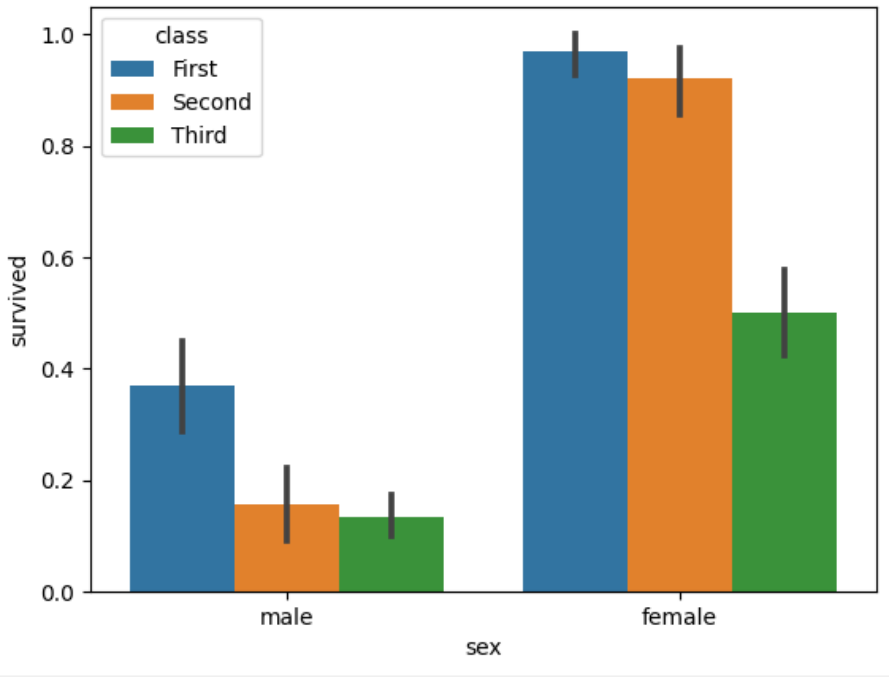
```
seaborn.barplot()
```

Parámetros:

- x: variable categórica
- y: variable continua
- hue: segunda variable categórica para desglose (opcional)
- data: serie, vector o lista

Ejemplo:

```
# dataframe creado a partir de titanic.csv
sns.barplot(x='sex', y='survived', hue='class', data = datos)
```



Como podemos observar, se nos permite desglosar los datos por una segunda variable categórica con el atributo ‘hue’.

b) Countplot

Es un caso especial de gráfico de barras, en que la variable numérica es la cantidad de ejemplares observados en cada categoría

Sintaxis:

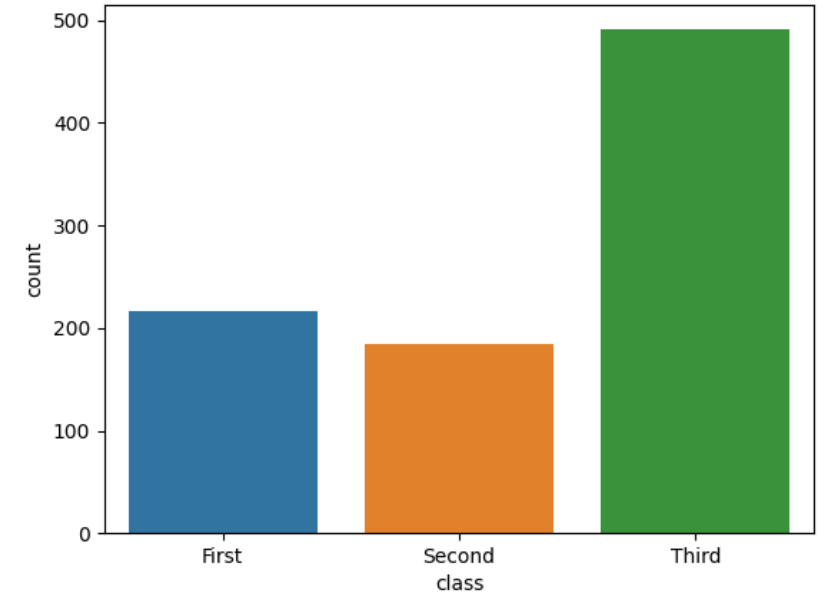
```
seaborn.countplot()
```

Parámetros:

- x: variable categórica
- data: serie, vector o lista

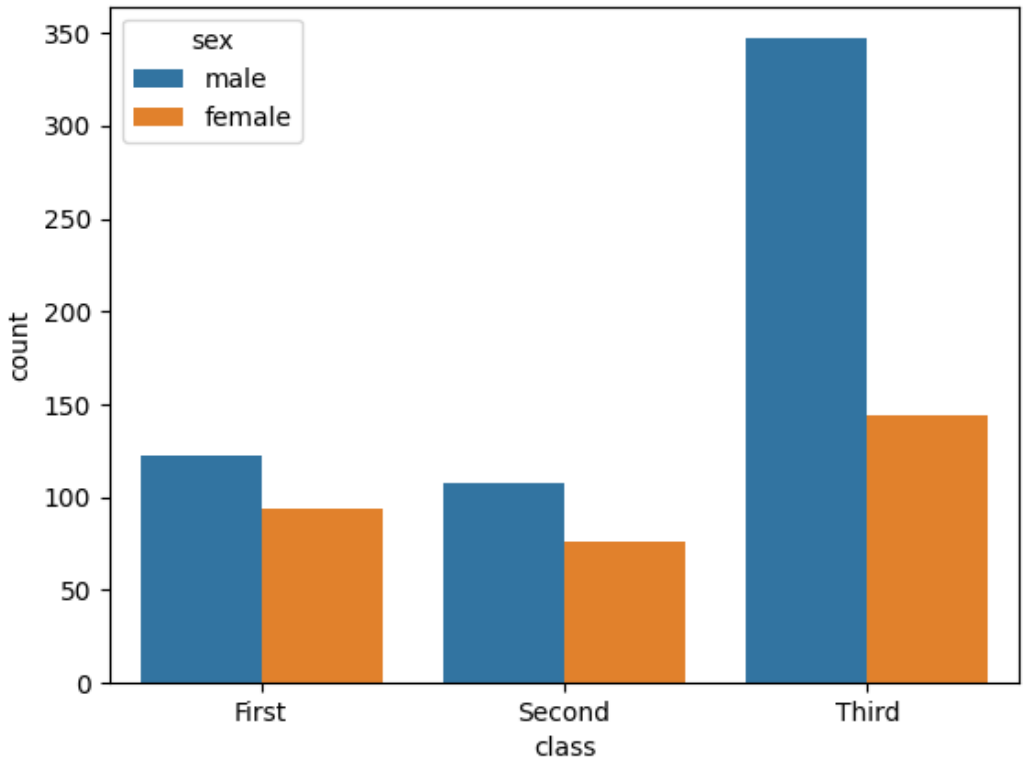
Ejemplo:

```
# dataframe creado a partir de titanic.csv      sns.countplot(x='class', data = datos)
```



También podemos aplicar desglose por una segunda variable categórica:

```
sns.countplot(x='class', hue='sex',data = datos)
```



8 Diagrama de cajas

También llamado diagrama de caja y bigotes (box-and-whisker), permite la visualización de la distribución de datos a través de los cuartiles. Es una excelente forma de resumir los datos estadísticos. Los outliers quedan señalados en el diagrama como puntos individuales.

Sintaxis:

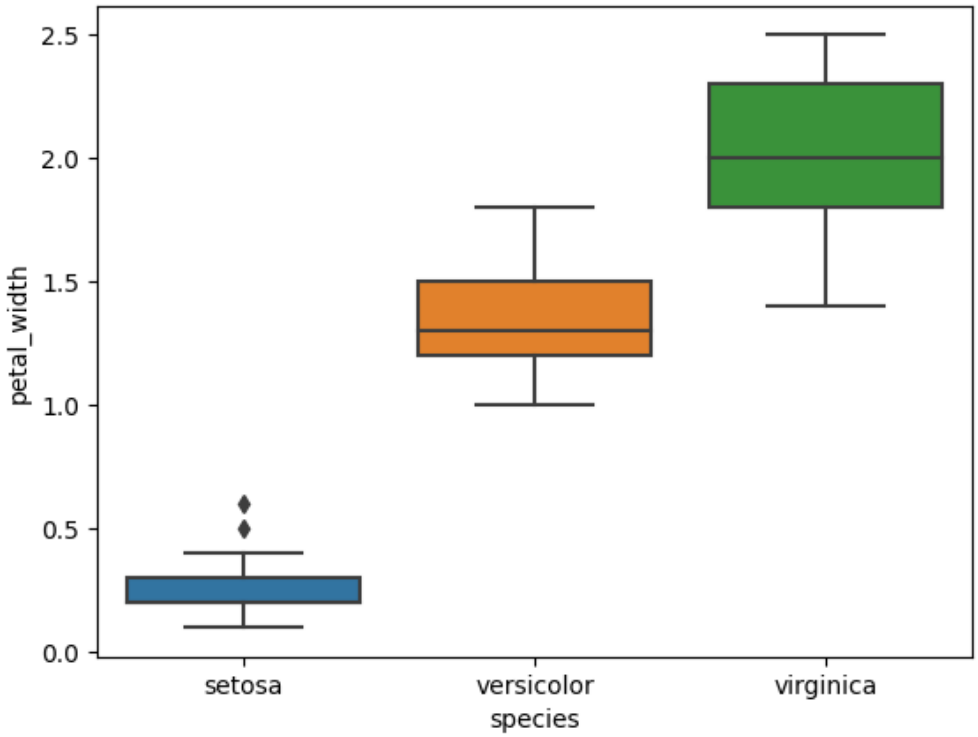
```
seaborn.boxplot()
```

Parámetros:

- x: variable categórica
- y: variable numérica
- data: serie, vector o lista

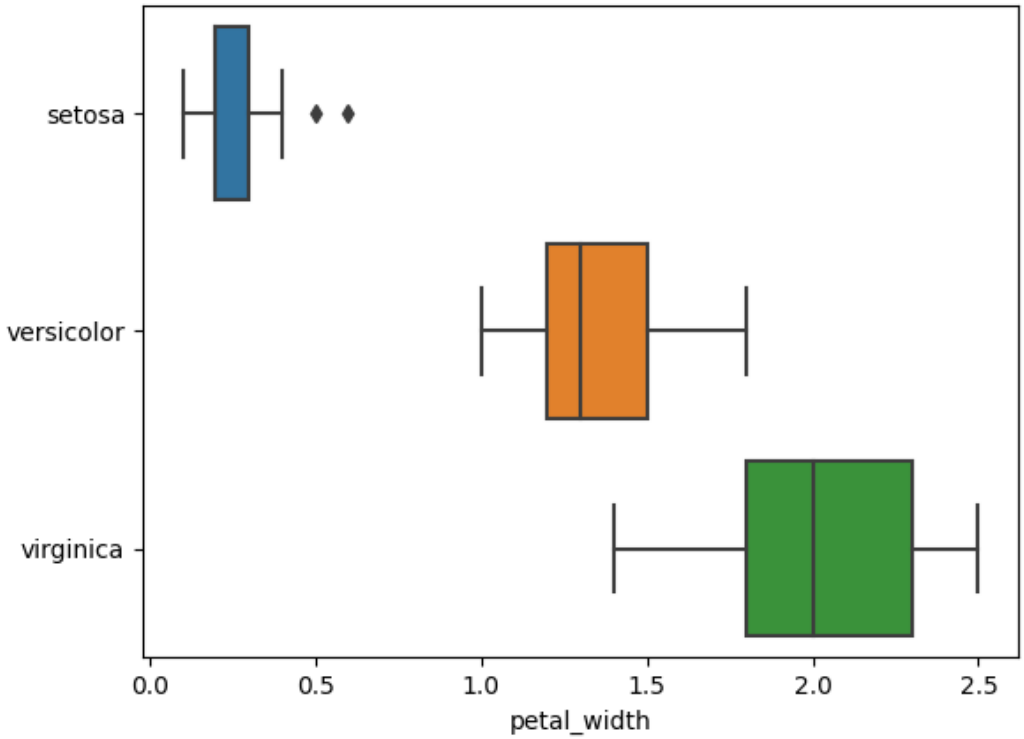
Ejemplo:

```
sns.boxplot(x='species', y='petal_width', data = datos)
```



Si nos resulta más cómodo visualizar la variable numérica en el eje horizontal, podemos invertir los ejes, e indicar a seaborn que nos modifique la gráfica con el atributo “orient”:

```
sns.boxplot(y='species', x='petal_width', data = datos, orient='h')
```


setosa
versicolor
virginica
0.0
0.5
1.0
1.5
2.0
2.5
petal_width

Enlaces externos

Documentación oficial Matplotlib: https://matplotlib.org/
Documentación oficial Seaborn: https://seaborn.pydata.org/
Paletas de colores en Seaborn: https://seaborn.pydata.org/tutorial/color_palettes.html