

Asociación Española de Programadores Informáticos (AEPI)

# CURSO PROFESIONAL DE INTELIGENCIA ARTIFICIAL

Epifanio Suárez Martínez - @episuarez  
epifanio.suarez@asociacionaepi.es

# Fundamentos de IA

Epifanio Suárez Martínez



## Definición

En ciencias de la computación, una máquina «inteligente» ideal es un agente flexible que percibe su entorno y lleva a cabo acciones que maximicen sus posibilidades de éxito en algún objetivo o tarea.

Aunque todavía estamos lejos de ese destino, si que hemos llegado a desarrollar Inteligencias Artificiales más efectivas que los humanos en muchos campos.

Las capacidades inherentes de la IA.

### Autonomía

Capacidad de realizar tareas en entornos complejos sin la guía constante de un usuario.

### Adaptabilidad

Capacidad de mejorar el desempeño aprendiendo a base de la experiencia.

# Tipos de IA

## Fuerte vs Débil

Una IA fuerte sería una inteligencia genuina y consciente de si misma. Y una débil equivale a una que simula comportamientos inteligentes.

## General vs Limitada

La IA limitada permite realizar una actividad concreta. Y la IA general permite a las máquinas realizar cualquier tarea intelectual.

Por lo tanto a lo que aspiramos es a generar IA fuertes y generales, aunque a día de hoy no tenemos capacidad de ello.

# General vs Limitada

LIMITADA	GENERAL
Una IA enfocada a una tarea específica.	No se ha llegado a desarrollar por completo y se duda de su posibilidad
Entrenamiento y capacitación	Manejo de multiples tareas cognitivas con poca supervisión.
Correlacion de preguntas o asignaciones especificas de una tarea	Aprender, generalizar, aplicar conocimientos, ...
Sin capacidad de pensar	Tiene sentido común y creatividad, e incluso puede expresar emociones.
Reconocimiento de imágenes, texto predictivo, traducciones, ...	Tiene que pasar el test de Turing

# Test de Turing

Alan turing [https://es.wikipedia.org/wiki/Alan\\_Turing](https://es.wikipedia.org/wiki/Alan_Turing), desarrolló la prueba de Turing en 1950.

El Test de Turing trata de evaluar si el comportamiento de una maquina se puede distinguir o no de la de un humano.



<https://www.youtube.com/watch?v=3wLqsRLvV-c>

# Inteligencia artificial

No hay que confundir los subcampos de la Inteligencia artificial, aunque algunos sean difíciles de diferenciar o se mezclen para conseguir mejores logros.

## Aprendizaje automático

Proporciona a las computadoras la capacidad de aprender, sin ser programadas explícitamente.

## Aprendizaje profundo

El aprendizaje profundo es parte de un conjunto más amplio de métodos de aprendizaje automático basados en asimilar representaciones de datos.

## Ciencia de datos

Es un campo interdisciplinario que involucra métodos científicos, procesos y sistemas para extraer conocimiento o un mejor entendimiento de datos en sus diferentes formas, ya sea estructurados o no estructurados.

## Robotica

Se ocupa del diseño, construcción, operación, estructura, manufactura y aplicación de los robots.

# Inteligencia artificial

INTELIGENCIA ARTIFICIAL	APRENDIZAJE AUTOMÁTICO	APRENDIZAJE PROFUNDO
Es una tecnología que se utiliza para crear máquinas inteligentes que pueden imitar el comportamiento humano.	Es un subconjunto de IA que aprende de datos y experiencias pasadas.	Es el subconjunto del aprendizaje automático y la inteligencia artificial que se inspira en las células del cerebro humano, llamadas neuronas, e imita el funcionamiento del cerebro humano.
La IA puede tratar datos estructurados, no estructurados y semiestructurados.	El ML puede lidiar con datos estructurados y semiestructurados.	El DL se ocupa de datos estructurados y no estructurados.
En general, requiere una gran cantidad de datos para funcionar.	Puede funcionar con menos cantidad de datos en comparación con el aprendizaje profundo.	Requiere una gran cantidad de datos en comparación con el aprendizaje automático.
El objetivo de la IA es permitir que la máquina piense y actúe sin ninguna intervención humana.	Su objetivo es permitir que una máquina aprenda de experiencias pasadas.	Su objetivo es resolver los problemas complejos como lo hace el cerebro humano, utilizando para ello varios algoritmos que imitan su funcionamiento.



# Robotica

Que la robotica sea controlada por la IA no quiere decir que no sea un subcampo diferente.

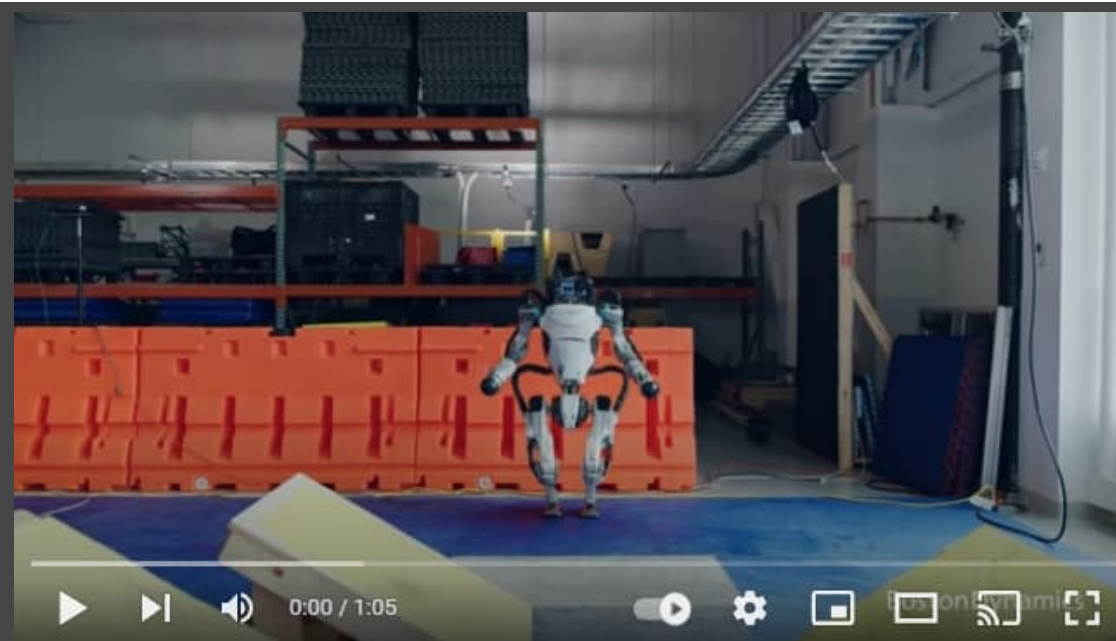
Aqui tenemos 3 ejemplos de las aplicaciones de la robotica.

FABRICA TESLA



[https://youtu.be/qqTx\\_eY0T2M?t=27](https://youtu.be/qqTx_eY0T2M?t=27)

BOSTON DYNAMICS



<https://youtu.be/tF4DML7FIWk>

LUCHA



[https://youtu.be/Jbpq\\_s5SYe0](https://youtu.be/Jbpq_s5SYe0)

# Probabilidad y estadística

Epifanio Suárez Martínez

# Probabilidad y estadística

El aprendizaje automático se basa en probabilidad y estadística.

Por ello aunque podamos usar herramientas para facilitarnos la obtención de resultados es vital comprender algunos conceptos.

Vamos a repasar algunos conceptos, aunque en nuestro trabajo trataremos con programación, si quereis profundizar en ello, tendreis que enfocar estos dos puntos.

# Probabilidad

La teoría de probabilidad proporciona un marco para razonar sobre la probabilidad de eventos.

## Terminología

- ❑ Experimento: Procedimiento que produce un posible conjunto de resultados. Lanzar repetidamente una moneda al aire.
- ❑ Espacio muestral  $S$ : Conjunto de posibles resultados de un experimento. Lanzamiento de un dado  $S = [1,2,3,4,5,6]$
- ❑ Evento  $E$ : Conjunto de resultados de un experimento. Por ejemplo, el caso concreto de una tirada tenga como resultado un 2.



# Probabilidad

## Probabilidad de un resultado $s$ . $P(s)$

Número que satisface dos propiedades:

□ Para cada resultado  $s$ ,  $0 \leq P(s) \leq 1$

□  $\sum P(s) = 1$

## Probabilidad de un evento $E$

Suma de las probabilidades de los resultados del experimento:  $p(E) = \sum_{s \in E} P(s)$

## Variable aleatoria $V$

Función numérica sobre los resultados de un espacio de probabilidad

## Valor esperado de una variable aleatoria $V$

$E(V) = \sum_{s \in E} P(s) * V(s)$

# Probabilidad

## Eventos independientes

A y B son independientes si:

$$\square P(A \cap B) = P(A) P(B)$$

$$\square P(A | B) = P(A)$$

$$\square P(B | A) = P(B)$$

## Probabilidad condicionada

$$P(A | B) = P(A, B) / P(B)$$

## Teorema de Bayes

$$P(A | B) = P(B | A) P(A) / P(B)$$

## Probabilidad conjunta

$$P(A, B) = P(B | A) P(A)$$

## Probabilidad marginal

$$P(A)$$

# Probabilidad

## Función de densidad de probabilidad (PDF)

Función que describe la probabilidad relativa según la cual una variable aleatoria tomará determinado valor:

$$p_X(x) = P(X = x)$$

## Función de densidad acumulativa (CDF)

Función que describe la probabilidad de que una variable aleatoria sea menor o igual que  $x$ :

$$F_X(x) = P(X \leq x)$$

## Resumen

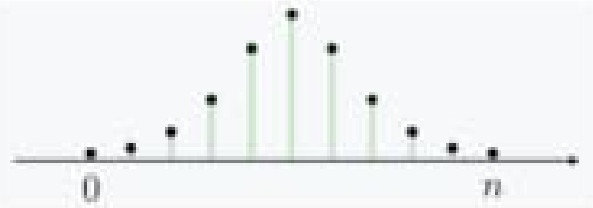
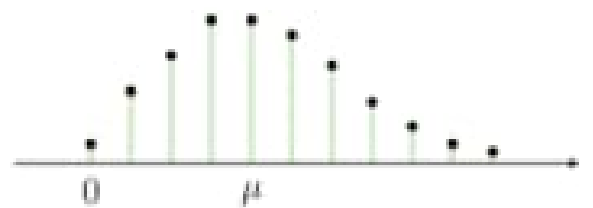
La función de densidad de probabilidad y la función de densidad acumulativa de una variable aleatoria dada contienen exactamente la misma información

# Probabilidad

## Distribuciones discretas

Las principales son las siguientes:

- ❑ Binomial, que cuenta el número de éxitos en una secuencia de  $n$  ensayos de Bernoulli independientes entre sí con una probabilidad fija  $p$  de ocurrencia de éxito entre ellos.
- ❑ Poisson, que expresa, a partir de una frecuencia de ocurrencia media, la probabilidad de que ocurra un determinado número de eventos durante un cierto periodo de tiempo.

Distribution	$P(X = x)$	$\psi(\omega)$	$E[X]$	$\text{Var}(X)$	Illustration
$X \sim \mathcal{B}(n, p)$	$\binom{n}{x} p^x q^{n-x}$	$(pe^{i\omega} + q)^n$	$np$	$npq$	
$X \sim \text{Po}(\mu)$	$\frac{\mu^x}{x!} e^{-\mu}$	$e^{\mu(e^{i\omega} - 1)}$	$\mu$	$\mu$	


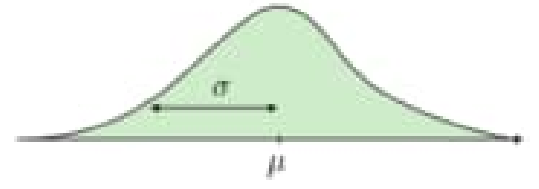
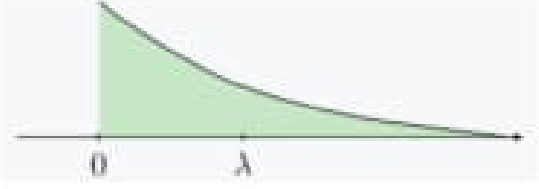


# Probabilidad

## Distribuciones continuas

Las principales son las siguientes:

- ❑ Uniforme, que es una familia de distribuciones de probabilidad para variables aleatorias continuas, tales que para cada miembro de la familia, todos los intervalos de igual longitud en la distribución en su rango son igualmente probables.
- ❑ Normal, cuya función de densidad tiene una gráfica con forma acampanada y es simétrica respecto de un determinado parámetro estadístico. Esta curva se conoce como campana de Gauss.
- ❑ Exponencial, que se utiliza para modelar tiempos de espera para la ocurrencia de un cierto evento.

Distribution	$f(x)$	$\psi(\omega)$	$E[X]$	$\text{Var}(X)$	Illustration
$X \sim \mathcal{U}(a, b)$	$\frac{1}{b-a}$	$\frac{e^{i\omega b} - e^{i\omega a}}{(b-a)i\omega}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$	
$X \sim \mathcal{N}(\mu, \sigma)$	$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	$e^{i\omega\mu - \frac{1}{2}\omega^2\sigma^2}$	$\mu$	$\sigma^2$	
$X \sim \text{Exp}(\lambda)$	$\lambda e^{-\lambda x}$	$\frac{1}{1 - \frac{i\omega}{\lambda}}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$	

# Estadística descriptiva

La estadística descriptiva proporciona técnicas para representar datos, resumirlos con una medida de posición y analizar lo concentrados que están alrededor de dicha medida de posición mediante una medida de dispersión.

## Medidas de posición

### Media aritmética

Suma de todos los valores observados dividida por el número total de ellos. Es útil para caracterizar distribuciones simétricas sin valores atípicos.

### Mediana

Es exactamente el valor que se sitúa en el medio de un conjunto de datos ordenados. Es muy útil para distribuciones de datos sesgadas o para conjuntos de datos con valores atípicos.

### Moda

Es el valor más frecuente de un conjunto de datos.

# Estadística descriptiva

## Medidas de dispersión

### Varianza

Se puede definir como la esperanza del cuadrado de la desviación de una variable con respecto a su media. Dicho con otras palabras, es la media aritmética de las desviaciones de los valores de una variable con respecto a su media.

### Desviación típica

La varianza tiene un problema, y es que está expresada en unidades al cuadrado. Esto puede provocar una falsa imagen de la dispersión de la distribución, ya que no es lo mismo decir que la dispersión en torno a la estatura media es de 16 cm, que decir que es de 4 cm. Por este motivo, suele utilizarse como medida de dispersión la raíz cuadrada de la varianza, que es la desviación típica.

### Análisis de correlaciones

Un coeficiente de correlación es un estadístico que mide el grado en que una variable Y es función de otra variable X, y viceversa. Los valores de correlación se encuentran entre -1 (correlación negativa) y 1 (correlación positiva). Por su parte, el 0 significa que no existe ningún tipo de correlación.

# Aprendizaje automático

Epifanio Suárez Martínez



# Tipos de aprendizajes

## Aprendizaje supervisado

En el aprendizaje supervisado, se entrena un modelo a partir de datos de entrada y sus etiquetas correspondientes.

## Aprendizaje semisupervisado

El aprendizaje semisupervisado consiste en entrenar un modelo sobre datos en los que algunos ejemplos tienen etiquetas, mientras que otros no.

## Aprendizaje no supervisado

En el aprendizaje no supervisado, se entrena un modelo para encontrar patrones en un conjunto de datos que no tienen etiquetas asociadas.

## ¿Qué resuelven?

En Aprendizaje automatico nos permite resolver dos principales problemas especificos.

REGRESIÓN	CLASIFICACIÓN
¿Cuál es la probabilidad de un cliente compre un producto?	¿Este correo es SPAM?
¿Cuál será el precio del pretoleo?	¿Esta imagen es de un perro o un gato?

Sin embargo el problema es el punto de vista y el agrupamiento, que puede hacer que tomemos con buenos datos, malas decisiones.

# Ciclo del aprendizaje automático

En el tiempo que le dedicamos a crear una solución de aprendizaje automático, la preparación de los datos es donde más tiempo gastamos. Y recuerdo que los procesos son iterativos.

- ☐ Recolección de datos
- ☐ Preprocesamiento
- ☐ Entrenamiento
- ☐ Pruebas
- ☐ Despliegue

# Aprendizaje por refuerzo

En el **aprendizaje por refuerzo** se trata de tres partes, un entorno, un agente y un intérprete.

El agente va a decidir que acciones va a realizar en el entorno, y dependiendo de sus acciones, el intérprete le va a recompensar de forma positiva, negativa o ambos.

Quizás el peor problema que podemos tener en estos algoritmos es que no haya un progreso en el aprendizaje.

Para evitarlo, tenemos que evaluar el aprendizaje para iterar sobre el proceso hasta que tengamos un resultado optimo.

Este aprendizaje puede caer en muchos problemas, pero tiene una gran capacidad de generalización y adaptación a nuevas situaciones, por lo que en muchas ocasiones suele ser una gran opción.



# Conceptos

Cada uno de los elementos de nuestro aprendizaje por refuerzo debe de contar con unos elementos.

## Agente

- ☐ Lista de acciones
- ☐ Cálculo del siguiente paso
- ☐ Actualizar la política

## Entorno

- ☐ Variables para conocer y medir los resultados
- ☐ Opción para reiniciar nuestro entorno
- ☐ La posibilidad de visualizar un entrenamiento

## Aprendizaje por asociación

El **aprendizaje por asociación** nos ayudan a descubrir todas las relaciones, por ejemplo entre una lista enorme de artículos de una base de datos.

Las reglas no se extraen de un las elecciones del usuario. Si no, del conjunto de elementos de cada transacción.

No se estudia los resultados de un usuario. En este caso hablamos de todos los elementos con unos identificadores únicos y como un grupo.

Esto es muy útil para recomendar páginas, decidir donde colocar un anuncio o un producto en una tienda, o simplemente recomendar una canción.

# Numpy

Epifanio Suárez Martínez

# Numpy

**Numpy** se define como el paquete fundamental de la informática científica.

## Características

- ☐ Arrays de n dimensiones
- ☐ Funciones especializadas de Matemáticas
- ☐ Soporta distintos entornos
- ☐ Un buen rendimiento debido a que está optimizado el código desde C.
- ☐ Fácil de usar
- ☐ Software libre

**Documentación** <https://numpy.org/doc/stable/>

## CheatSheet

[https://s3.amazonaws.com/assets.datacamp.com/blog\\_assets/Numpy\\_Python\\_Cheat\\_Sheet.pdf](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf)

# Arrays

Los **arrays** de **numpy** pueden tener tantas dimensiones como necesitemos.

Aquí perdemos las ventajas de python respecto al tipado flexible. Es necesario tener claro que tipo necesitamos para poder trabajar.

TIPO	DESCRIPCIÓN
bool	True / False
int8	Byte (-128 a 127)
int16	Integer (-32768 a 32767)
int32	Integer (-2147483648 a 2147483647)
int64	Integer (-9223372036854775808 a 9223372036854775807)
float16	5 bit para la parte entera, 10 para la parte decimal y 1 bit para el signo
float32	8 bit para la parte entera, 23 para la parte decimal y 1 bit para el signo
float64	11 bit para la parte entera, 52 para la parte decimal y 1 bit para el signo
complex64	Número complejo, representado por dos flotantes de 32 bits
complex128	Número complejo, representado por dos flotantes de 64 bits

Más información en <https://numpy.org/doc/stable/user/basics.types.html>

## Funciones universales

**Numpy** tiene lo que define como **funciones universales**, funciones que operan sobre **ndarrays** de forma elemento por elemento.

Tenemos desde funciones **Matemáticas**, **Trigonométricas**, **Manipulación de bits**, **Comparativas** y **Flotantes**.

Son funciones muy útiles y podeis encontrar una buena lista en <https://numpy.org/doc/stable/reference/ufuncs.html#available-ufuncs>



# Arrays

En Numpy cada dimensión se denomina **axis**, el número de dimensiones **rank**, la lista de dimensiones **shape** y el número total de elementos **size**.

```
miArray = numpy.zeros((2, 4));

print(miArray);

"""
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])
"""

print(miArray.shape); # (2, 4)
print(miArray.ndim); # 2
print(miArray.size); # 8
```

# Arrays

## Crear un array con ceros

```
numpy.zeros((2, 3, 4));
```

```
array([[[[0., 0., 0., 0.],  
         [0., 0., 0., 0.],  
         [0., 0., 0., 0.]],  
       [[0., 0., 0., 0.],  
         [0., 0., 0., 0.],  
         [0., 0., 0., 0.]])])
```

# Arrays

## Crear un array con unos

```
numpy.ones((2, 3, 4));
```

```
array([[[1., 1., 1., 1.],  
        [1., 1., 1., 1.],  
        [1., 1., 1., 1.]],  
       [[1., 1., 1., 1.],  
        [1., 1., 1., 1.],  
        [1., 1., 1., 1.]])
```

# Arrays

## Crear un array con un numero

```
numpy.full((2, 3, 4), 8);
```

```
array([[[8, 8, 8, 8],  
        [8, 8, 8, 8],  
        [8, 8, 8, 8]],  
       [[8, 8, 8, 8],  
        [8, 8, 8, 8],  
        [8, 8, 8, 8]]])
```

# Arrays

## Crear un array vacios

¡Cuidado! Al ser valores vacios pueden obtener cualquier valor.

```
numpy.empty((2, 3, 9));
```

```
array([[[ 2.31584178e+077, -1.49457241e-154,  2.24704384e-314,
          2.24703961e-314,  2.24703803e-314,  2.24059351e-314,
          2.24704094e-314,  2.24704777e-314,  2.24703958e-314],
        [ 2.24703739e-314,  2.24703980e-314,  2.24703344e-314,
          2.24704170e-314,  2.24704720e-314,  2.24704043e-314,
          2.26943022e-314,  2.26943028e-314,  2.26857648e-314],
        [ 2.26864577e-314,  2.24703942e-314,  2.24703945e-314,
          2.24703983e-314,  2.24704008e-314,  2.24704011e-314,
          2.24704736e-314,  2.24704745e-314,  2.24704739e-314]],
       [[ 2.24704764e-314,  2.24704087e-314,  2.26859791e-314,
          2.26859772e-314,  2.26859855e-314,  2.26859779e-314,
          2.26859863e-314,  2.31584178e+077,  2.31584178e+077],
        [ 2.24076228e-314,  2.24133656e-314,  0.00000000e+000,
          0.00000000e+000,  0.00000000e+000,  4.94065646e-324,
          4.94065646e-324,  0.00000000e+000,  0.00000000e+000],
        [ 2.26958240e-314,  0.00000000e+000,  0.00000000e+000,
          0.00000000e+000,  2.26958233e-314,  2.26958233e-314,
          0.00000000e+000, -1.29073793e-231,  3.75914434e-308]]])
```

# Arrays

## Crear un array con la clase

```
miArray = numpy.array([[1, 2, 3], [4, 5, 6]]);  
print(miArray);  
  
"""  
array([[1, 2, 3],  
       [4, 5, 6]])  
"""  
  
print(miArray.shape); # (2, 3)
```



# Arrays

## Crear un array espaciado linealmente

Podemos crear un array considerando el mínimo, máximo y la cantidad total.

```
miArray = numpy.linspace(0, 6, 10);  
print(miArray);
```

```
"""
```

```
[0.          0.66666667  1.33333333  2.          2.66666667  3.33333333  
 4.          4.66666667  5.33333333  6.          ]
```

```
"""
```

# Arrays

## Crear un array aleatorio

```
miArray = numpy.random.rand(2, 3, 4);
print(miArray);

"""
array([[ [0.03378783, 0.51370292, 0.0448622 , 0.59146622],
        [0.65020737, 0.98552739, 0.02602994, 0.23295317],
        [0.61644088, 0.74966376, 0.98919117, 0.10470065] ],
       [ [0.43496062, 0.5189665 , 0.6438961 , 0.75687474],
        [0.76658059, 0.27930081, 0.70681049, 0.74179797],
        [0.39368796, 0.2892813 , 0.85135049, 0.84055115] ] ])
```

# Arrays

## Crear un array a través de funciones

Podemos crear arrays a través de funciones.

```
def calculo(x, y):  
    return x + 2 * y;  
  
miArray = numpy.fromfunction(calculo, (3, 5));  
print(miArray);  
  
"""  
array([[ 0.,  2.,  4.,  6.,  8.],  
       [ 1.,  3.,  5.,  7.,  9.],  
       [ 2.,  4.,  6.,  8., 10.]])  
"""
```

# Arrays

## Acceso a la información de un array unidimensional

```
miArray = numpy.array([1, 3, 5, 7, 9, 11]);  
print(miArray.shape); # (6,)  
print(miArray); # [ 1  3  5  7  9 11]  
print(miArray[4]); # 9  
print(miArray[2:4]); # array([5, 7])  
print(miArray[0::3]); # array([1, 7])
```

# Arrays

## Acceso a la información de un array bidimensional

```
miArray = numpy.array([[1, 2, 3, 4], [5, 6, 7, 8]]);  
print(miArray.shape); # (2, 4)  
print(miArray); # [[1 2 3 4] [5 6 7 8]]  
print(miArray[0, 3]); # 4  
print(miArray[1, :]); # array([5, 6, 7, 8])  
print(miArray[0:2, 2]); # array([3, 7])
```

# Arrays

## Crear un array

```
mi_array = numpy.array([1, 2, 3]);
```

## Agregar varios elementos a un array

```
mi_array = numpy.array([1, 2, 3]);  
nuevo_array = numpy.append(mi_array, [11, 12, 13]);
```

## Agregar varios elementos en otra dimensión

```
mi_array1 = numpy.array([ [1, 2, 3], [4, 5, 6] ]);  
mi_array2 = numpy.array([ [10, 20, 30], [40, 50, 60] ]);  
nuevo_array = numpy.append(mi_array1, mi_array2, axis = 1);
```



# Arrays

## Insertar un valor en una posición

```
mi_array = numpy.array([1, 2, 3]);  
nuevo_array = numpy.insert(mi_array, 1, 22); # [ 1 22  2  3]
```

## Insertar una fila

```
mi_array = numpy.array([ [1, 2, 3], [4, 5, 6] ]);  
nuevo_array = numpy.append(mi_array, [[10, 11, 12]], axis = 0);
```

```
"""  
[[ 1  2  3]  
 [ 4  5  6]  
 [10 11 12]]  
"""
```

## Filtrar datos

```
mi_array = numpy.array([1, 2, 3]);  
array_filtrado = mi_array[mi_array > 1]; # [2 3]
```

# Arrays

## Modificar arrays

```
mi_array = numpy.arange(15);  
print(mi_array.shape); # (15,)  
print(mi_array); # [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

## Cambiar las dimensiones

```
mi_array.shape = (3, 5);  
print(mi_array.shape); # (3, 5)  
print(mi_array);
```

```
"""
```

```
[[ 0  1  2  3  4]  
 [ 5  6  7  8  9]  
 [10 11 12 13 14]]"""
```

Tambien puedes cambiar el tamaño con `reshape(3, 5)`, la diferencia entre uno y otro, es que si al transformar el tamaño surge un problema, `reshape` deja el estado anterior, y `shape` arroja una excepcion.

## Convertir un array en unidimensional

```
print(array1.ravel()); # [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

# Arrays

## Eliminar un valor

```
mi_array1 = numpy.array([1, 2, 3]);  
nuevo_array = numpy.delete(mi_array1, 1, axis = 0);
```

## Eliminar una fila

```
mi_array1 = numpy.array([ [1, 2, 3], [4, 5, 6] ]);  
nuevo_array = numpy.delete(mi_array1, 1, axis = 0);
```

# Arrays

## Buscar en un array

```
mi_array1 = numpy.array([1, 2, 3]);  
print(np.where(mi_array1 == 2));
```

## Obtener parte de un array

```
mi_array1 = numpy.array([1, 2, 3]);  
print(mi_array1[1:2]);
```

# Arrays

## Aplicar una función a un array

```
incremento = lambda x: x + 3;  
mi_array1 = numpy.array([1, 2, 3, 4]);  
print(incremento(mi_array1));
```

# Arrays

## Normalizar un array (Valores entre -1 y 1)

```
import numpy as np

mi_array1 = np.array([200, 300, 500, 800, 100, 10, 1000]);
maximo = mi_array1.max();
minimo = mi_array1.min();

valores_normalizados = (mi_array1 - minimo) / (maximo - minimo);
print(valores_normalizados);
```



# Pandas

Epifanio Suárez Martínez

# Pandas

**Pandas** es una herramienta para el análisis y manipulación de datos.

Pandas tiene dos principales elementos para manejar datos, las series y los dataframe. Las series son como array unidimensionales, y los dataframes son como array bidimensionales.

## Características

- ❑ Herramientas para leer CSV, Excel, Bases de datos y HDF.
- ❑ Alto manejo de **DataFrames**.
- ❑ Un buen rendimiento debido a que está optimizado el código desde C.

**Documentación** <https://pandas.pydata.org/docs/> ¡Más de **3000** páginas el pdf!

**CheatSheet** [https://pandas.pydata.org/Pandas\\_Cheat\\_Sheet.pdf](https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf)

# Pandas

## Series

Es una estructura de datos de tamaño mutable unidimensional.

Se componen de indice y una columna con sus valores.

INDICE	ANIMAL
0	perro
1	pez
2	pajaro

# Pandas

## Generar una Serie

```
serie = pandas.Series([2, 4, 6, 8, 10]);  
print(serie);
```

```
"""  
0      2  
1      4  
2      6  
3      8  
4     10  
dtype: int64  
"""
```

# Pandas

## Generar una Serie con un diccionario

```
altura = {"Santiago": 187, "Pedro": 178, "Julia": 170, "Ana": 165};  
serie = pandas.Series(altura);  
print(serie);
```

```
"""  
Santiago      187  
Pedro         178  
Julia         170  
Ana           165  
dtype: int64  
"""
```

# Pandas

## Generar una Serie con un diccionario

```
altura = {"Santiago": 187, "Pedro": 178, "Julia": 170, "Ana": 165};  
serie = pandas.Series(altura, index=["Pedro", "Julia"]);  
print(serie);
```

```
"""
```

```
Pedro      178
```

```
Julia      170
```

```
dtype: int64
```

```
"""
```

# Pandas

## DataFrame

Es una estructura de datos tabulares de tamaño mutable bidimensional con ejes etiquetados.

Se componen de indice, filas y columnas, con sus valores.

INDICE	ANIMAL	PATAS	ALAS
0	perro	4	0
1	pez	0	0
2	pajaro	0	2



## Generar un DataFrame

De una forma fácil podemos usar **diccionarios** de **Python** para crear cualquier **DataFrame** e incluso exportarlo a **CSV** u otro formato.

```
manzanas = [10, 20, 45, 30, 25]
peras = [5, 3, 5, 7, 9]

diccionario = {
    "Manzanas": manzanas,
    "Peras": peras
}

dataframe = pandas.DataFrame(diccionario);
```

# Manejo de DataFrames

```
# Obtener todos los datos  
dataframe;
```

```
# Obtener una columna  
dataframe["Nombre"];  
dataframe.nombre;
```

```
# Obtener el tipo  
dataframe.dtypes;
```

```
# Obtener el tipo de una columna  
dataframe["Nombre"].dtypes;
```

```
# Obtener los 5 primeros registros  
dataframe.head();
```

```
# Obtener los 5 últimos registros  
dataframe.tail();
```

# Manejo de DataFrames

```
# Obtener estadísticas descriptivas  
dataframe.describe();
```

```
# Obtener los valores máximos y mínimos  
dataframe.max();  
dataframe.min();
```

```
# Obtener el tipo  
dataframe.mean();  
dataframe.median();
```

```
# Obtener la desviación típica o estándar  
dataframe.std();
```

```
# Obtener una muestra aleatoria  
dataframe.sample();
```

## Conjunto de datos

Un **conjunto de datos** nos permite dividir los datos en grupos según unos criterios y calcular estadística al grupo.

```
conjunto_datos = dataframe.groupby("nombre");  
conjunto_datos.mean();  
  
medias_conjunto_datos = dataframe.groupby("nombre").mean();
```

# Filtrado de datos

Podemos filtrar datos tanto de un **dataframe** como de un **conjunto de datos**.

```
nuevo_dataframe = dataframe[dataframe["edad"] > 18];  
nuevo_dataframe = dataframe[dataframe["nombre"] == "pepe"];  
nuevo_dataframe = dataframe.query("edad > 17")
```

## Valores extremos

Si tenemos valores extremos podemos filtrarlos de muchas maneras, aunque una opción recomendada es jugar con **quantile**.

```
dataframe[dataframe["edad"] < trabajadores["edad"].quantile()];
```

# Separando los datos

Ya hemos visto como obtener los datos de una columna o varias columnas.

```
dataframe.peras;  
dataframe["Peras"];  
  
dataframe[["Manzanas", "Peras"]];
```

También podemos obtener un rango de filas, lo mismo que haríamos en **Python** con las listas.

```
dataframe[10:20];
```

E incluso podemos separar los datos con un rango de filas y n columnas.

```
dataframe.loc[10, 20, ["nombre", "edad"]];
```

O hacer lo mismo, pero a través de las posiciones de las columnas.

```
dataframe.iloc[10, 20, [0, 3]];
```

## Más usos de iloc

```
dataframe.iloc[0] # Primera fila de un conjunto de datos  
dataframe.iloc[i] # (i + 1) fila  
dataframe.iloc[-1] # Última fila
```

```
dataframe.iloc[:, 0] # Primera columna  
dataframe.iloc[:, -1] # Última columna
```

```
dataframe.iloc[0: 7] #Primeras 7 filas  
dataframe.iloc[:, 0: 2] #Primeras 2 columnas  
dataframe.iloc[1: 3, 0: 2] #Segunda a tercera fila y primeras 2 columnas  
dataframe.iloc[[0,5], [1,3]] # Primera y sexta fila y segunda y cuarta columna
```

# Ordenar

Podemos ordenar los datos de un **DataFrame**. Siempre se ordena por defecto de forma ascendente.

```
nuevo_dataframe = dataframe.sort_values(by="nombre");  
nuevo_dataframe = dataframe.sort_values(by=["precio", "estrellas"], ascending=[False, False]);
```



## Valores faltantes

Para tratar los valores faltantes podemos resumir una sería de reglas.

- ❑ Al sumar los datos, los valores faltantes se tratarán como cero.
- ❑ Si faltan todos los valores, la suma será igual a **NaN**
- ❑ Los métodos **cumsum** y **cumprod** ignoran los valores perdidos, pero los conservan en las matrices resultantes
- ❑ Los valores faltantes en el método **GroupBy** están excluidos.
- ❑ Muchos **métodos de estadística** descriptiva tienen **skipna** como opción para controlar si se deben excluir los datos faltantes.
- ❑ Este valor se establece en **True** por defecto.

# Valores faltantes

```
# Elimina los valores faltantes  
dataframe.dropna();
```

```
# Aparte de eliminar, muestra observaciones donde hay valores nulos  
dataframe.dropna(how="any");
```

```
# Elimina toda la columna si faltan todos los valores  
dataframe.dropna(axis=1, how="all");
```

```
# Elimina las filas que contienen al menos 5 valores faltantes  
dataframe.dropna(thresh=5);
```

```
# Sustituye los valores que faltan por ceros  
dataframe.fillna();
```

```
# Devuelve True si falta el valor  
dataframe.isnull();
```

```
# Devuelve True si no falta el valor  
dataframe.notnull();
```

# Agregación

Cuando queremos sacar varias estadísticas a la vez, podemos usar la función de **agregación** en una o más columnas.

```
dataframe["nombre"].agg(["min", "mean", "std"]);
```

## Estadística descriptiva básica

MÉTODOS	DESCRIPCIÓN
describe	Estadísticas básicas
min, max	Mínimo / Máximo
mean, median, mode	Media / Mediana / Moda
var, std	Varianza / Desviación estándar
sem	Error estándar de media
skew	Asimetría de la muestra
kurt	Curtosis

## Datos cruzados

Para obtener muestras de **datos cruzados** podemos usar **crosstab** para analizar esos datos.

```
pandas.crosstab(dataframe.Sexo, dataframe.TienenCoche);
```

De esta manera podemos conseguir buenos resultados sobre todo en datos **booleanos**.

# Operadores lógicos

En **Python** tenemos los operadores lógicos de **and**, **or**, y **not**. Sin embargo para **Pandas** para poder trabajar por elementos, por ello en vez de usar los operadores como conocemos, debemos cambiar un poco.

```
# and
dataframe["nombre"] == "pepe" & dataframe["edad"] == "60"

# or
dataframe["nombre"] == "pepe" | dataframe["edad"] == "60"

# not
~(dataframe["nombre"] == "pepe") & ~(dataframe["edad"] == "60")
```

## Combinación de DataFrames

Al igual que en cualquier combinación, podemos realizar varios tipos para juntar los datos.

- ❑ **Join:** Une los diversos datos por índice/posición.
- ❑ **Merge:** Une los diversos datos a través de una columna, y guardando aquellos que son iguales a la misma.
- ❑ **Concat:** Añade los datos a través de un eje seleccionado.
- ❑ **Append:** Añade una o más filas de un **Dataframe** a otro.

# Lectura de ficheros

# CSV

```
dataframe = pandas.read_csv("datos.csv");
```

# Excel

```
dataframe = pandas.read_excel("datos.xlsx", "Hoja1");
```

# SQL

```
conexion = sqlite3.connect('base_de_datos.db');  
dataframe = pandas.read_sql('SELECT * FROM Alumnos', conexion);
```

# DTA

```
dataframe = pandas.read_stata("datos.dta");
```

# SAS

```
dataframe = pandas.read_sas("datos.sas7bdat");
```

# HDF

```
dataframe = pandas.read_hdf("datos.h5", "df");
```

# Escritura en ficheros

# CSV

```
dataframe.to_csv("datos.csv");
```

# Excel

```
dataframe.to_excel("datos.xlsx", "Hoja1");
```

# DTA

```
dataframe.to_stata("datos.dta");
```

# HDF

```
dataframe.to_hdf("datos.h5", "df");
```



# Matplotlib

Epifanio Suárez Martínez

# Matplotlib

Nos permite representar todo tipo de gráficos, histogramas, espectros de potencia, gráficos de barras, gráficos de error, gráficos de dispersión, ...

**¡Importante!** En **Jupyter** muchas funciones o variables se visualizan automáticamente, sin embargo esto no es así siempre. A veces necesitamos hacer un **print**, incluso en el caso de **Matplotlib** llamar a **show** para que se visualice, o generar una imagen o lanzarlo en **Jupyter**.

**Documentación** <https://matplotlib.org/stable/contents.html>

**Cheat Sheet**

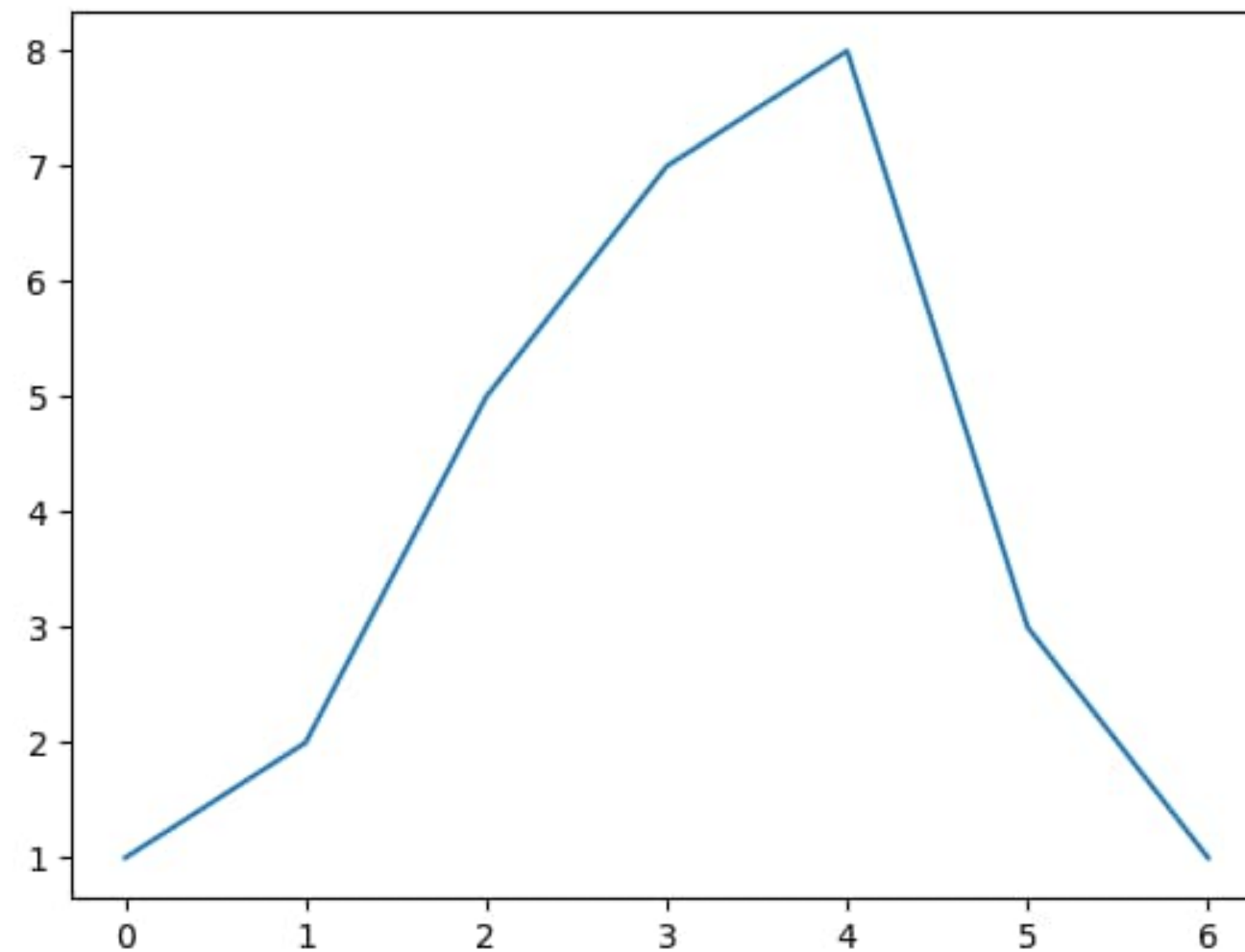
[https://s3.amazonaws.com/assets.datacamp.com/blog\\_assets/Python\\_Matplotlib\\_Cheat\\_Sheet.pdf](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Matplotlib_Cheat_Sheet.pdf)

# Gráficas

Pintar una gráfica es sencillo, solo necesitamos los datos y mostrarla.

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

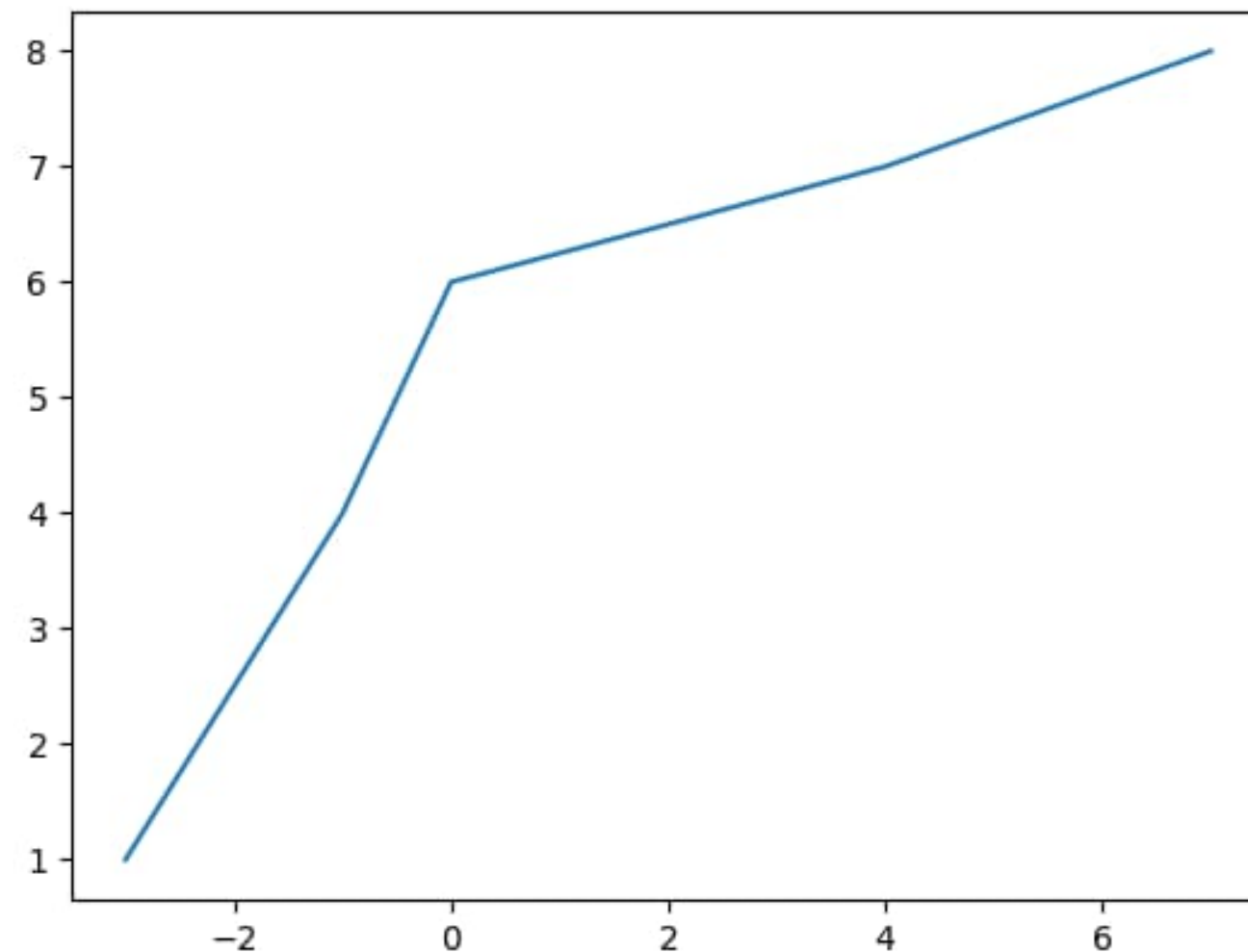
```
pyplot.plot([1, 2, 5, 7, 8, 3, 1]);  
pyplot.show();
```



# Gráficas

También podemos generar gráficas con los listas, para el eje x y el eje y.

```
import matplotlib.pyplot as pyplot  
  
pyplot.plot([-3, -1, 0, 4, 7], [1, 4, 6, 7, 8]);  
pyplot.show();
```

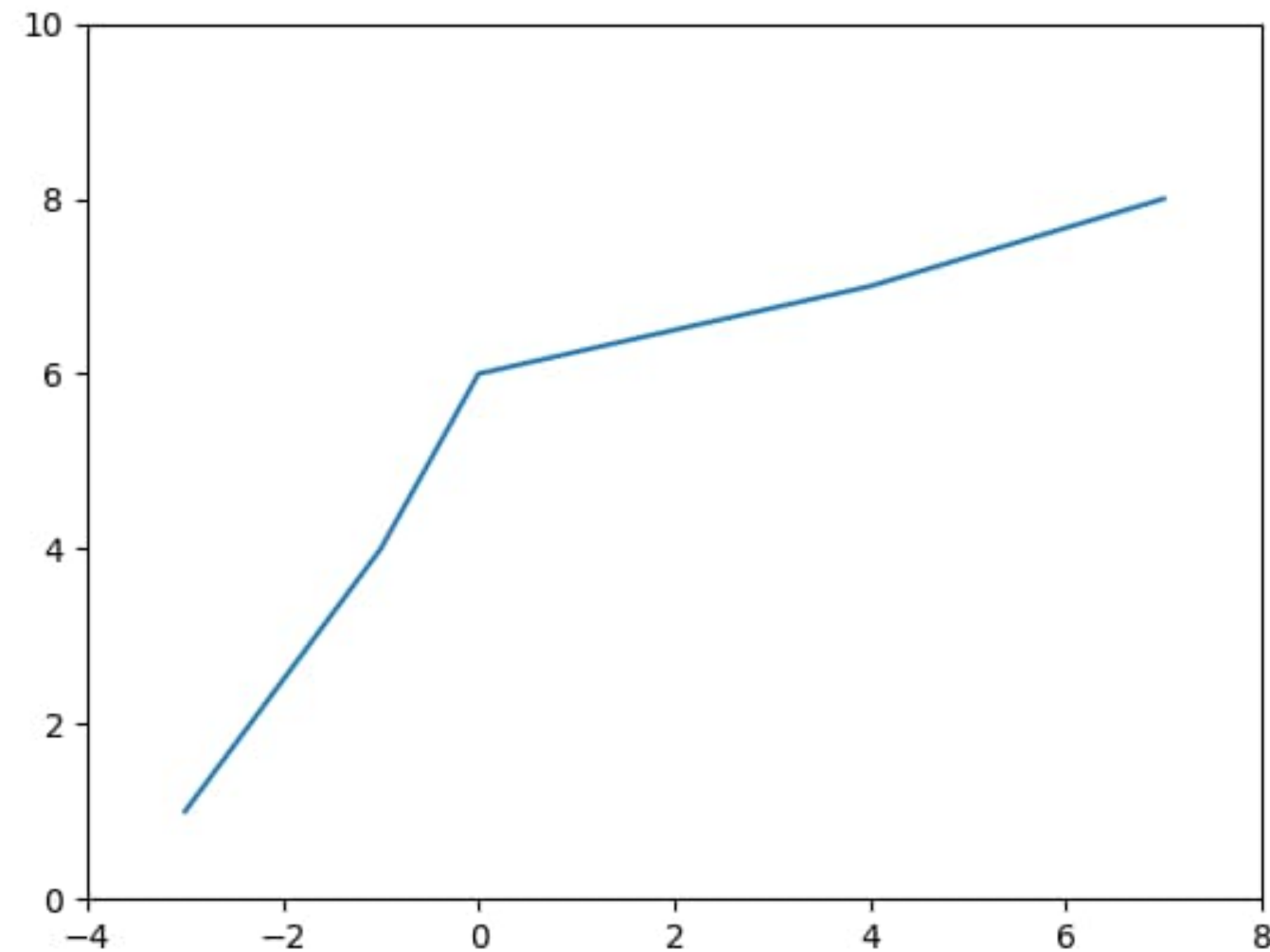


# Gráficas

Podemos modificar los límites de las gráficas para mostrar mejor la figura.

```
import matplotlib.pyplot as pyplot

pyplot.plot([-3, -1, 0, 4, 7], [1, 4, 6, 7, 8]);
pyplot.axis([-4, 8, 0, 10]);
pyplot.show();
```



# Gráficas

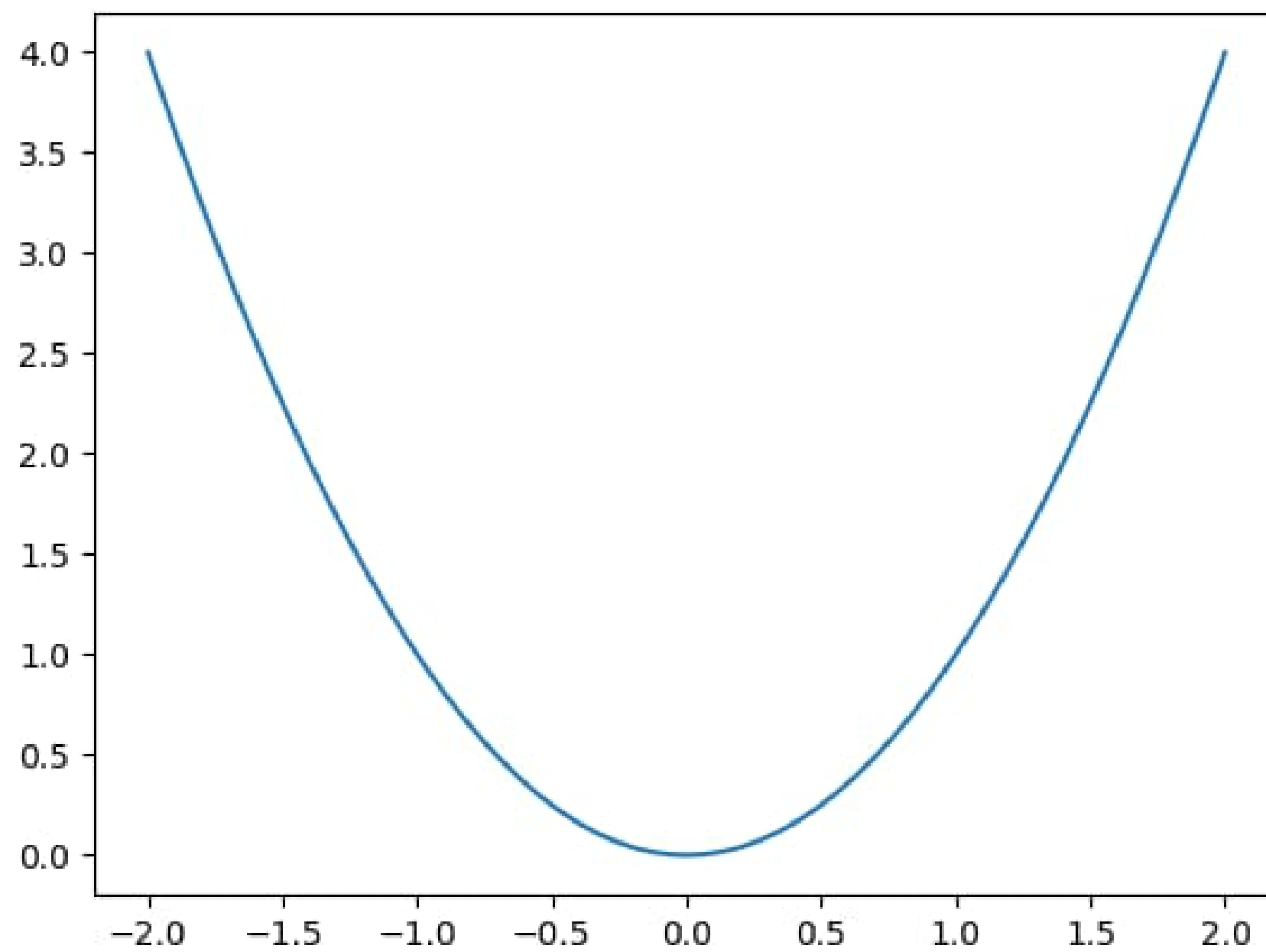
Podemos pintar una función matemática.

```
import matplotlib.pyplot as pyplot
import numpy

values_x = numpy.linspace(-2, 2, 500);
values_y = values_x ** 2;

pyplot.plot(values_x, values_y);
pyplot.show();
```

# Gráficas



# Gráficas

Podemos pintar una función matemática, pero podemos añadirle mucho más detalle.

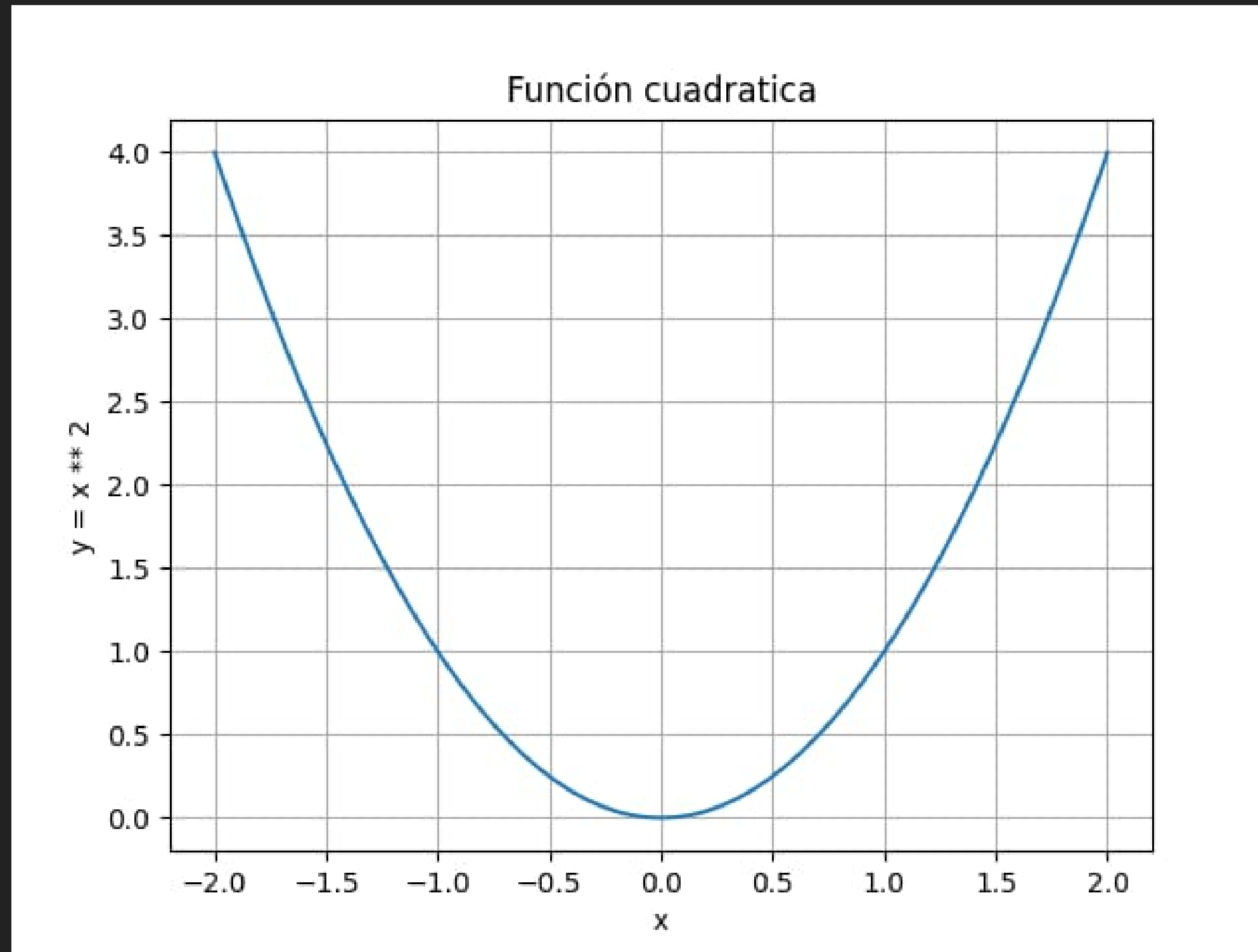
```
import matplotlib.pyplot as plt
import numpy

values_x = numpy.linspace(-2, 2, 500);
values_y = values_x ** 2;

plt.plot(values_x, values_y);
plt.title("Función cuadrática");
plt.xlabel("x");
plt.ylabel("y = x ** 2");
plt.grid(True);
plt.show();
```



# Gráficas



# Gráficas

Podemos tener varias graficas en el mismo plot, y cambiar el tipo de la linea.

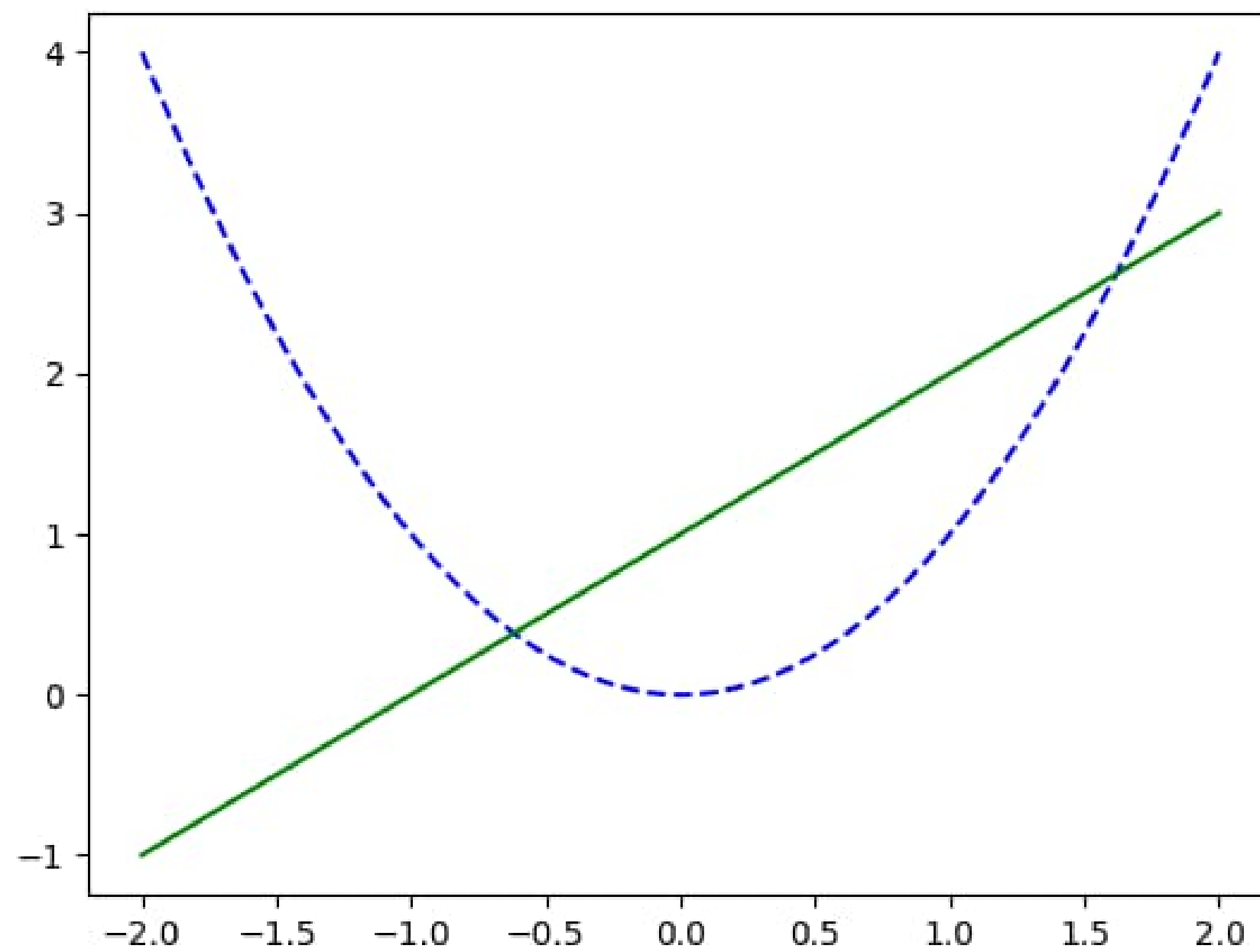
```
import matplotlib.pyplot as pyplot
import numpy

values_x = numpy.linspace(-2, 2, 500);
values_y = values_x ** 2;
values_y2 = values_x + 1;

pyplot.plot(values_x, values_y, 'b--', values_x, values_y2, 'g');

pyplot.show();
```

# Gráficas



# Gráficas

También podemos agregar la leyenda en el grafico.

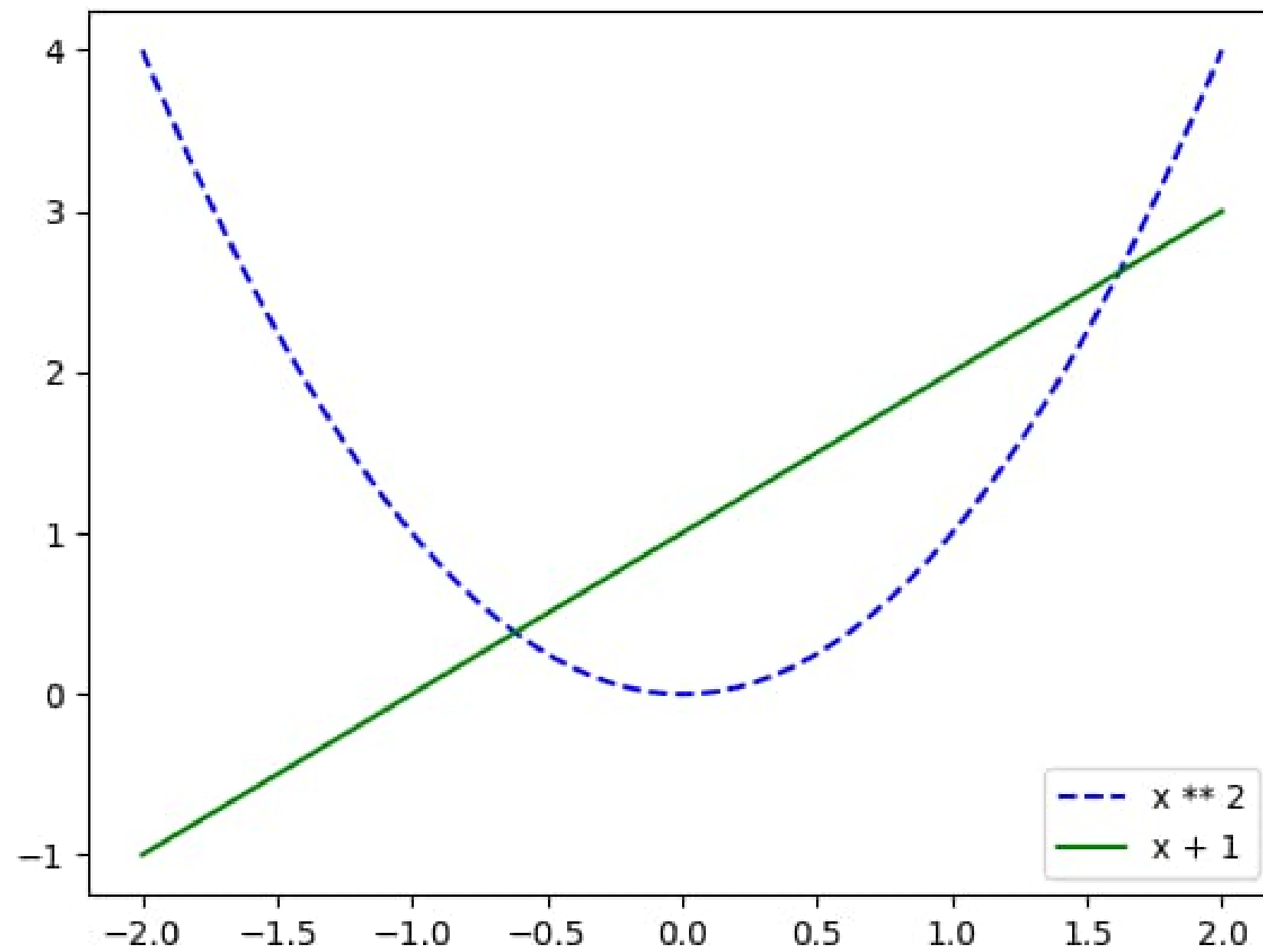
```
import matplotlib.pyplot as pyplot
import numpy

values_x = numpy.linspace(-2, 2, 500);
values_y = values_x ** 2;
values_y2 = values_x + 1;

pyplot.plot(values_x, values_y, 'b--', label="x ** 2");
pyplot.plot(values_x, values_y2, 'g', label="x + 1");
pyplot.legend(loc="best");

pyplot.show();
```

# Gráficas



# Gráficas

Tambien podemos agregar la leyenda en el grafico.

```
import matplotlib.pyplot as pyplot
import numpy

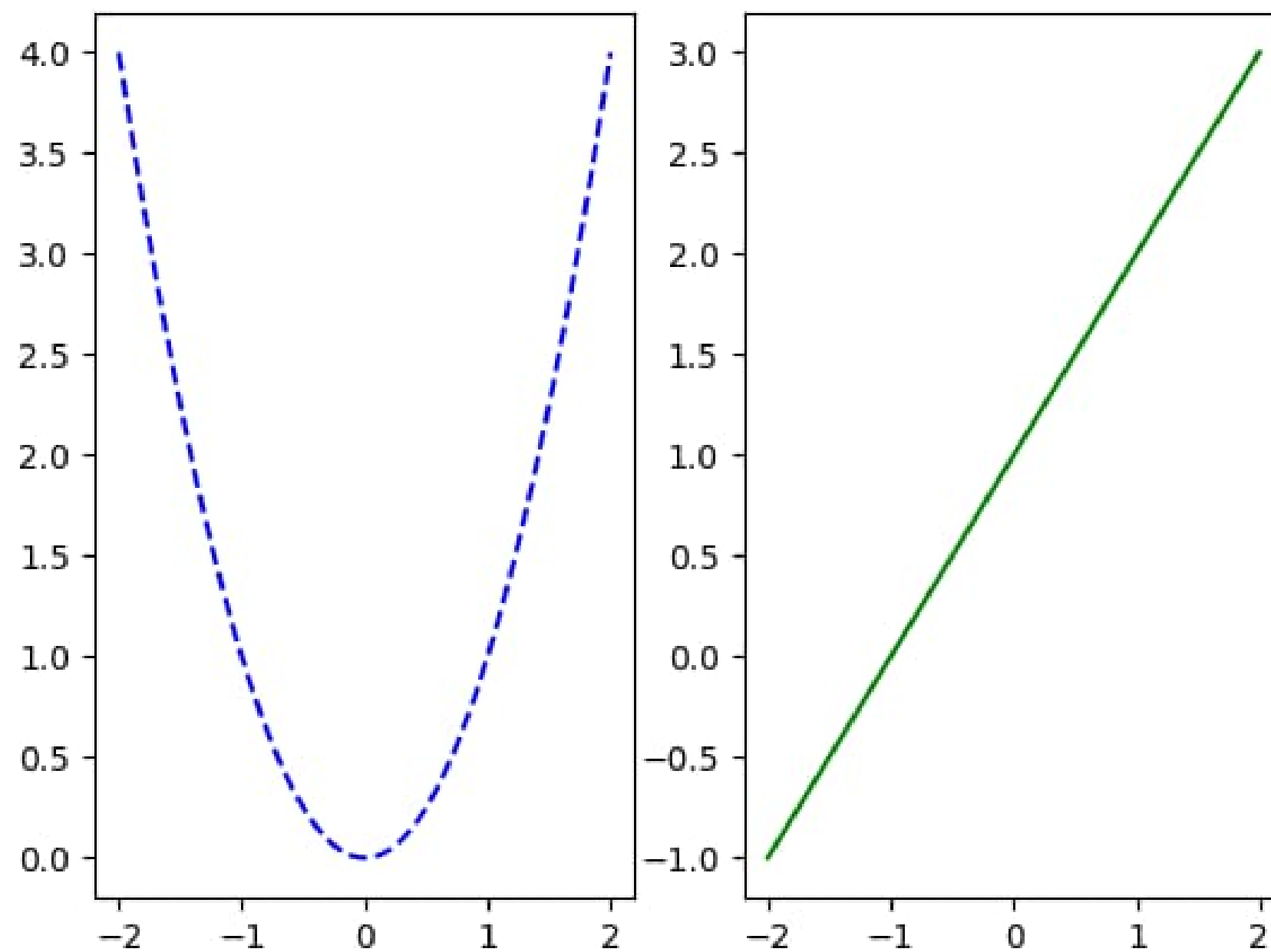
values_x = numpy.linspace(-2, 2, 500);
values_y = values_x ** 2;
values_y2 = values_x + 1;

pyplot.subplot(1, 2, 1);
pyplot.plot(values_x, values_y, 'b--');

pyplot.subplot(1, 2, 2);
pyplot.plot(values_x, values_y2, 'g');

pyplot.show();
```

# Gráficas



# Gráficas

También podemos agregar la leyenda en el gráfico.

```
import matplotlib.pyplot as plt
import numpy

values_x = numpy.linspace(-2, 2, 500);
values_y = values_x ** 2;
values_y2 = values_x + 1;

plt.figure(figsize=(14,6));

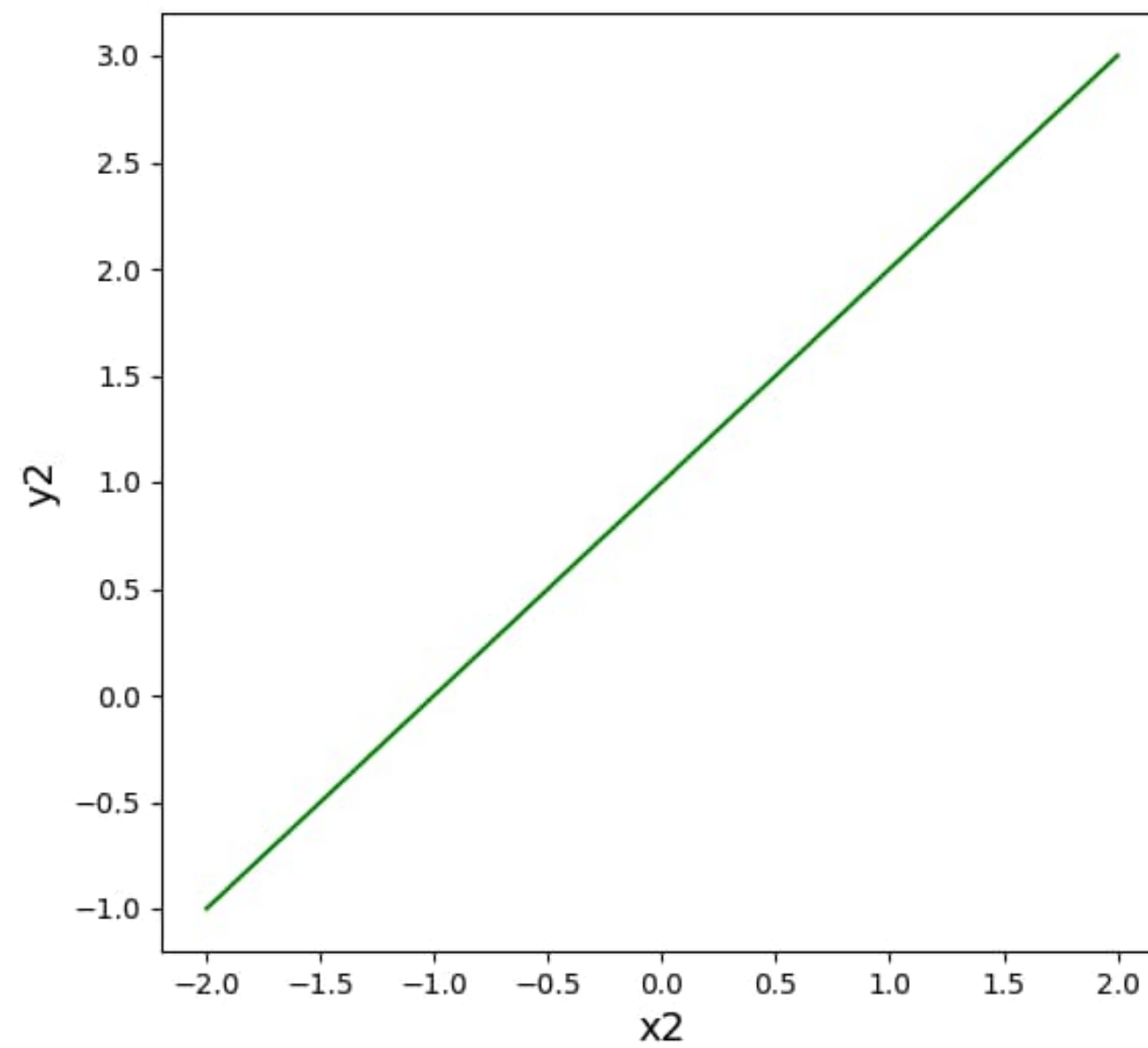
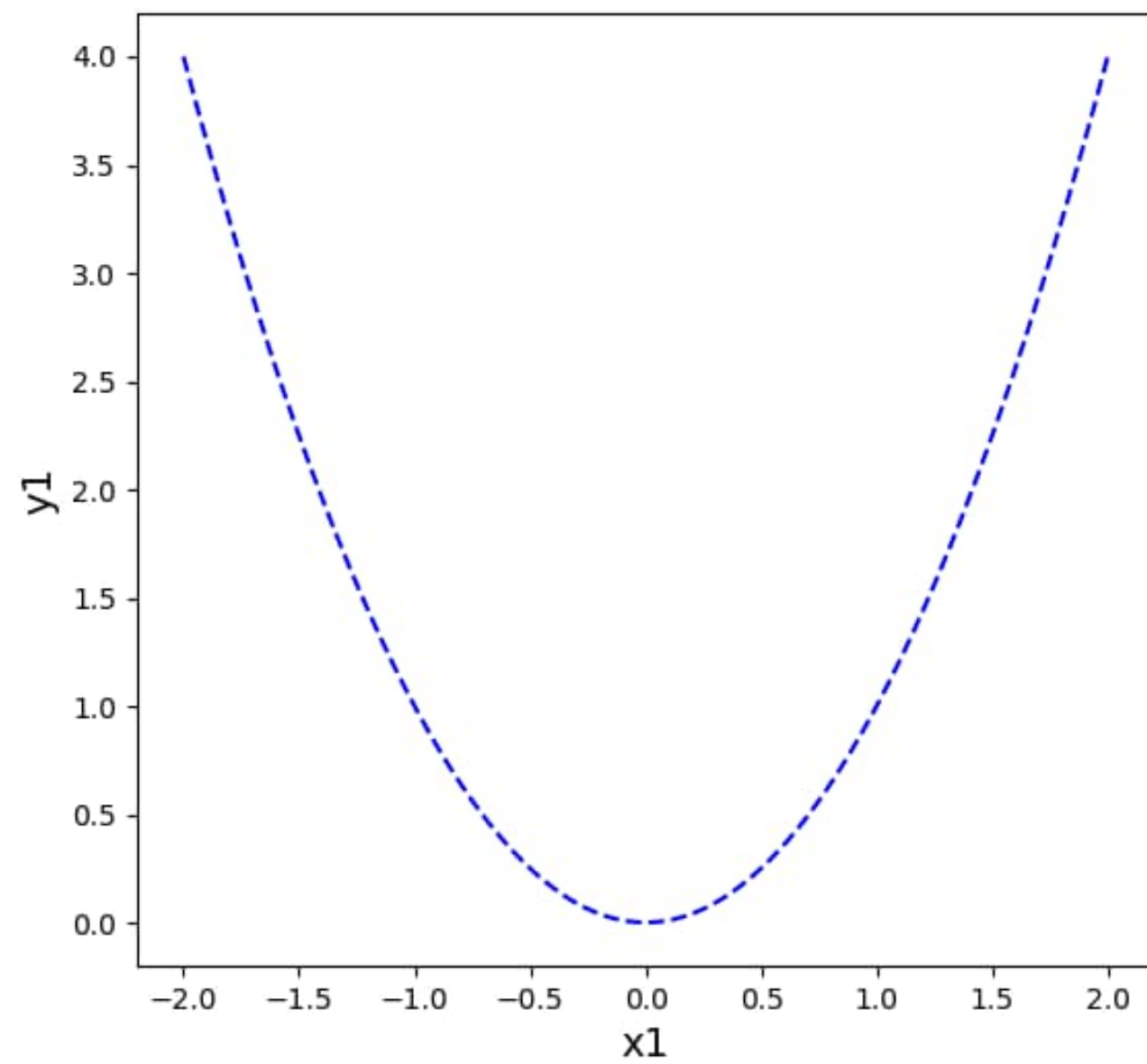
plt.subplot(1, 2, 1);
plt.plot(values_x, values_y, 'b--');
plt.xlabel("x1", fontsize=14);
plt.ylabel("y1", fontsize=14);

plt.subplot(1, 2, 2);
plt.plot(values_x, values_y2, 'g');
plt.xlabel("x2", fontsize=14);
plt.ylabel("y2", fontsize=14);

plt.show();
```



# Gráficas

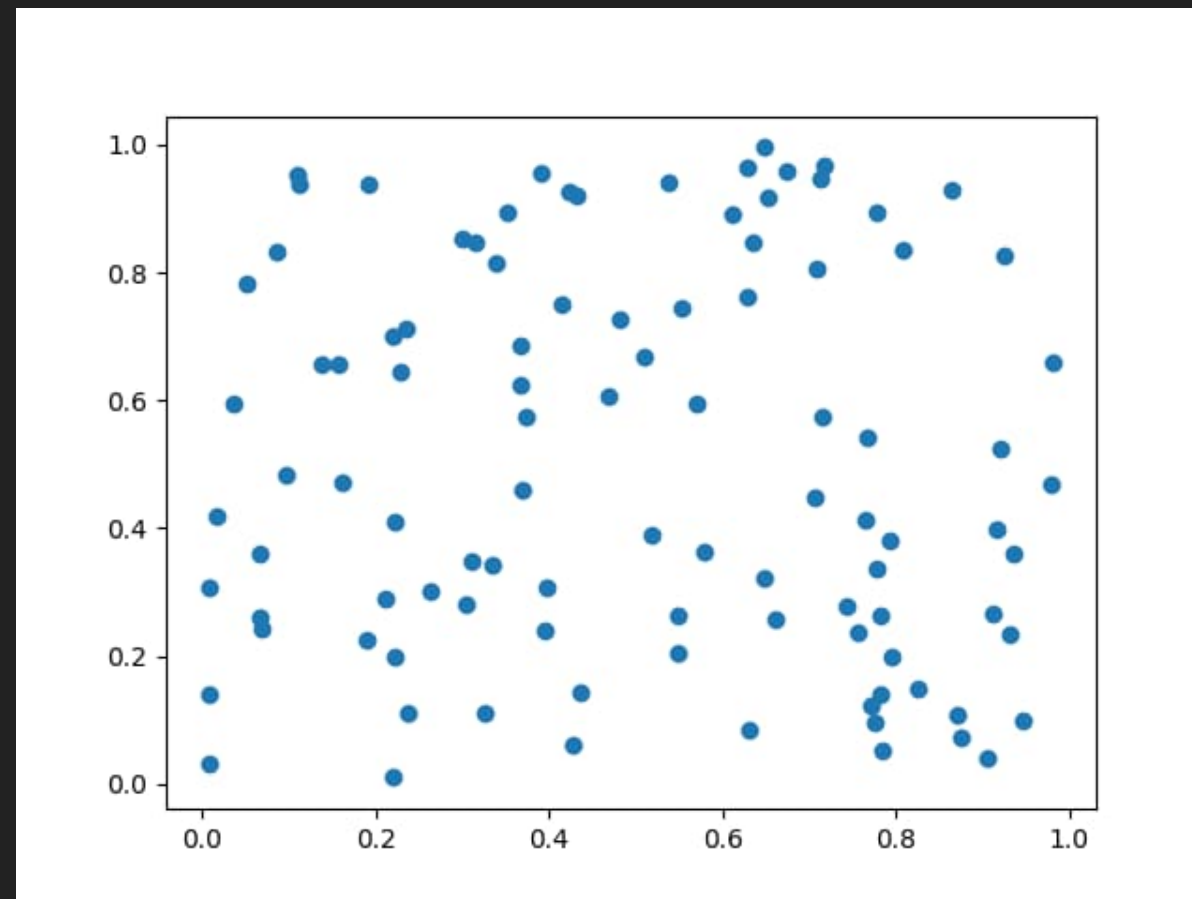


# Gráficas

Tambien podemos generar gráficas de dispersión.

```
import matplotlib.pyplot as pyplot
import numpy

value_x, value_y = numpy.random.rand(2, 100);
pyplot.scatter(value_x, value_y);
pyplot.show();
```



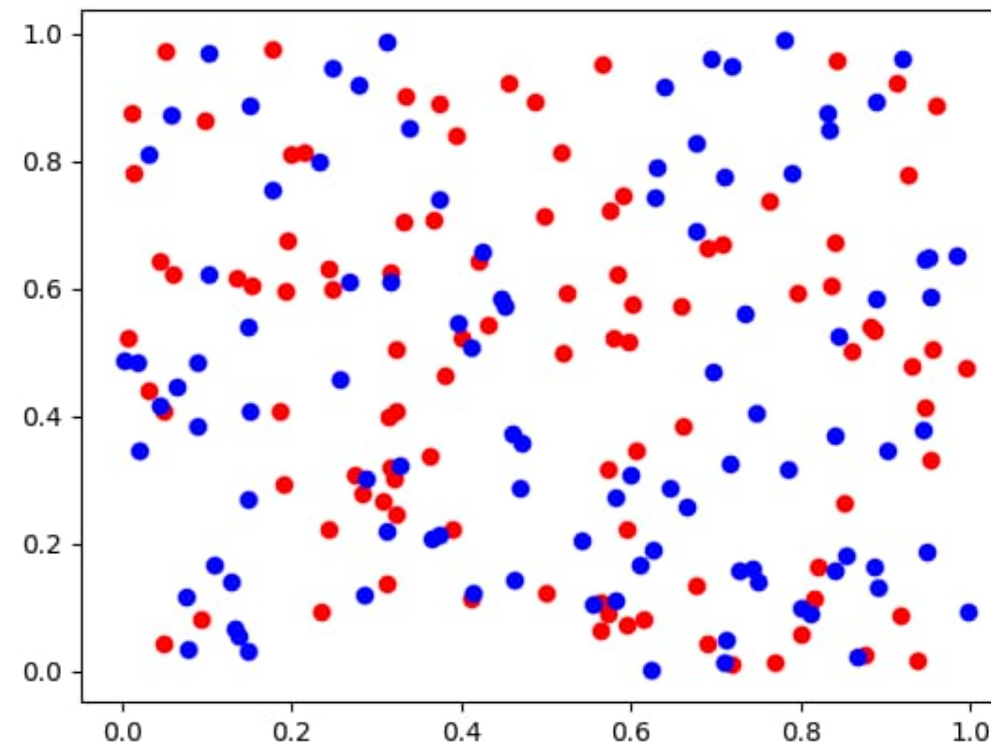
# Gráficas

Podemos separar las gráficas de dispersión aunque las mostremos en el mismo grafico.

```
import matplotlib.pyplot as pyplot
import numpy

value_x1, value_y1 = numpy.random.rand(2, 100);
value_x2, value_y2 = numpy.random.rand(2, 100);

pyplot.scatter(value_x1, value_y1, c="red");
pyplot.scatter(value_x2, value_y2, c="blue");
pyplot.show();
```



# Gráficas

También podemos crear histogramas. Recordad que los bins, es decir, el número de barras lo definimos y puede variar el contenido.

```
import matplotlib.pyplot as plt
import numpy

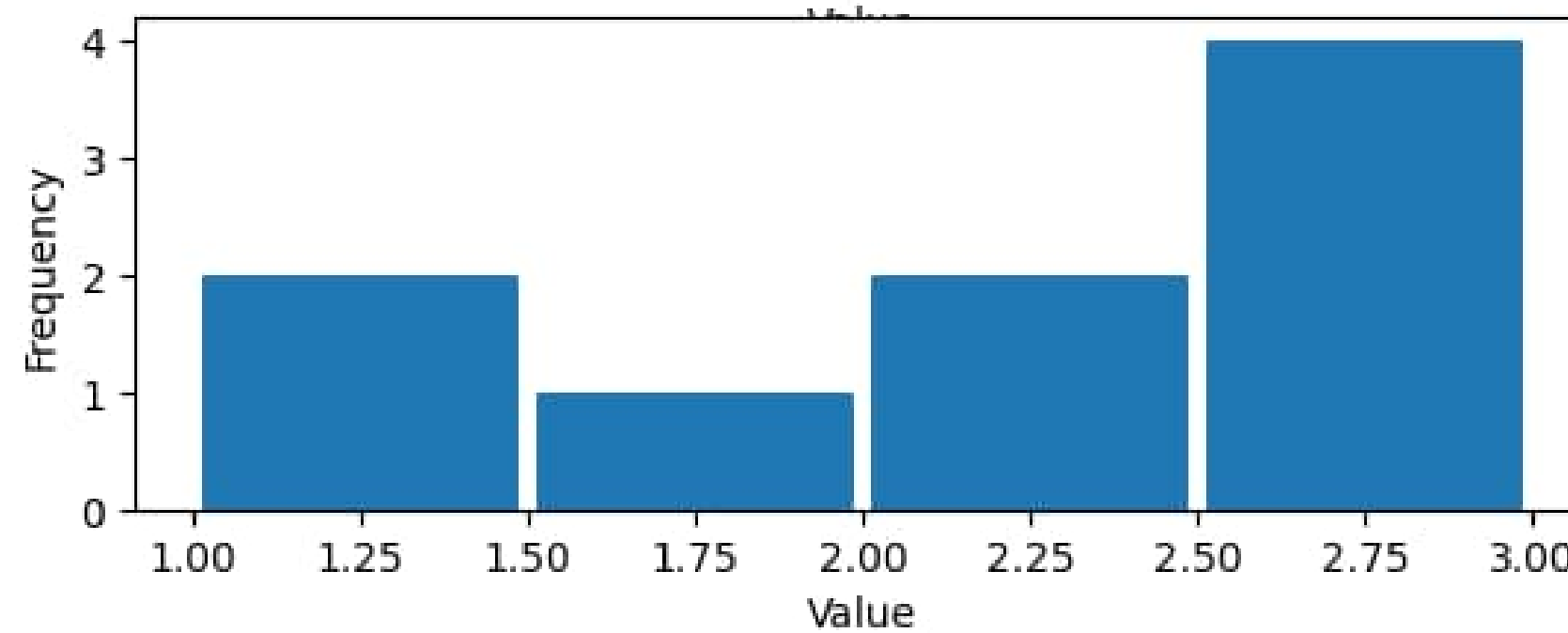
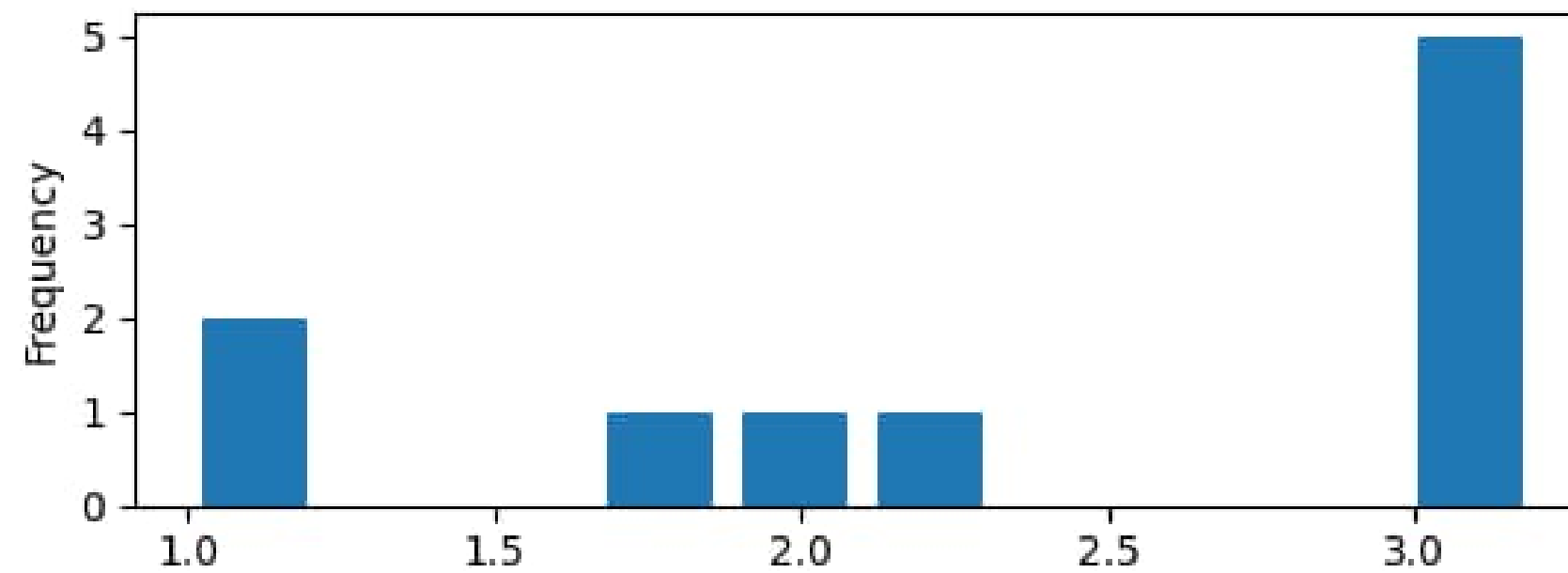
data = [1, 1.1, 1.8, 2, 2.1, 3.2, 3, 3, 3, 3];

plt.subplot(211);
plt.hist(data, bins = 10, rwidth=0.8);
plt.xlabel("Value");
plt.ylabel("Frequency");

plt.subplot(212);
plt.hist(data, bins = [1, 1.5, 2, 2.5, 3], rwidth=0.95);
plt.xlabel("Value");
plt.ylabel("Frequency");

plt.show();
```

# Gráficas



# Gráficas

Mostramos las gráficas con:

```
pyplot.show();
```

Podemos guardar una imagen con:

```
pyplot.savefig("imagen.png");
```

Cuando generamos varias gráficas seguidas es necesario cerrar el plot, porque si no, se concatenan.

```
pyplot.close();
```

# Seaborn

Epifanio Suárez Martínez

# Seaborn

Nos permite representar todo tipo de gráficos, histogramas, espectros de potencia, gráficos de barras, gráficos de error, gráficos de dispersión, ... Quizás esta herramienta está más enfocada a la visualización de datos estadísticos.

**¡Importante!** En **Jupyter** muchas funciones o variables se visualizan automáticamente, sin embargo esto no es así siempre. A veces necesitamos hacer un **print**, incluso en el caso de **Seaborn** llamar a **show** para que se visualice, o generar una imagen o lanzarlo en **Jupyter**.

**Documentación** <https://seaborn.pydata.org/introduction.html>

**Cheat Sheet**

[https://s3.amazonaws.com/assets.datacamp.com/blog\\_assets/Python\\_Seaborn\\_Cheat\\_Sheet.pdf](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Seaborn_Cheat_Sheet.pdf)

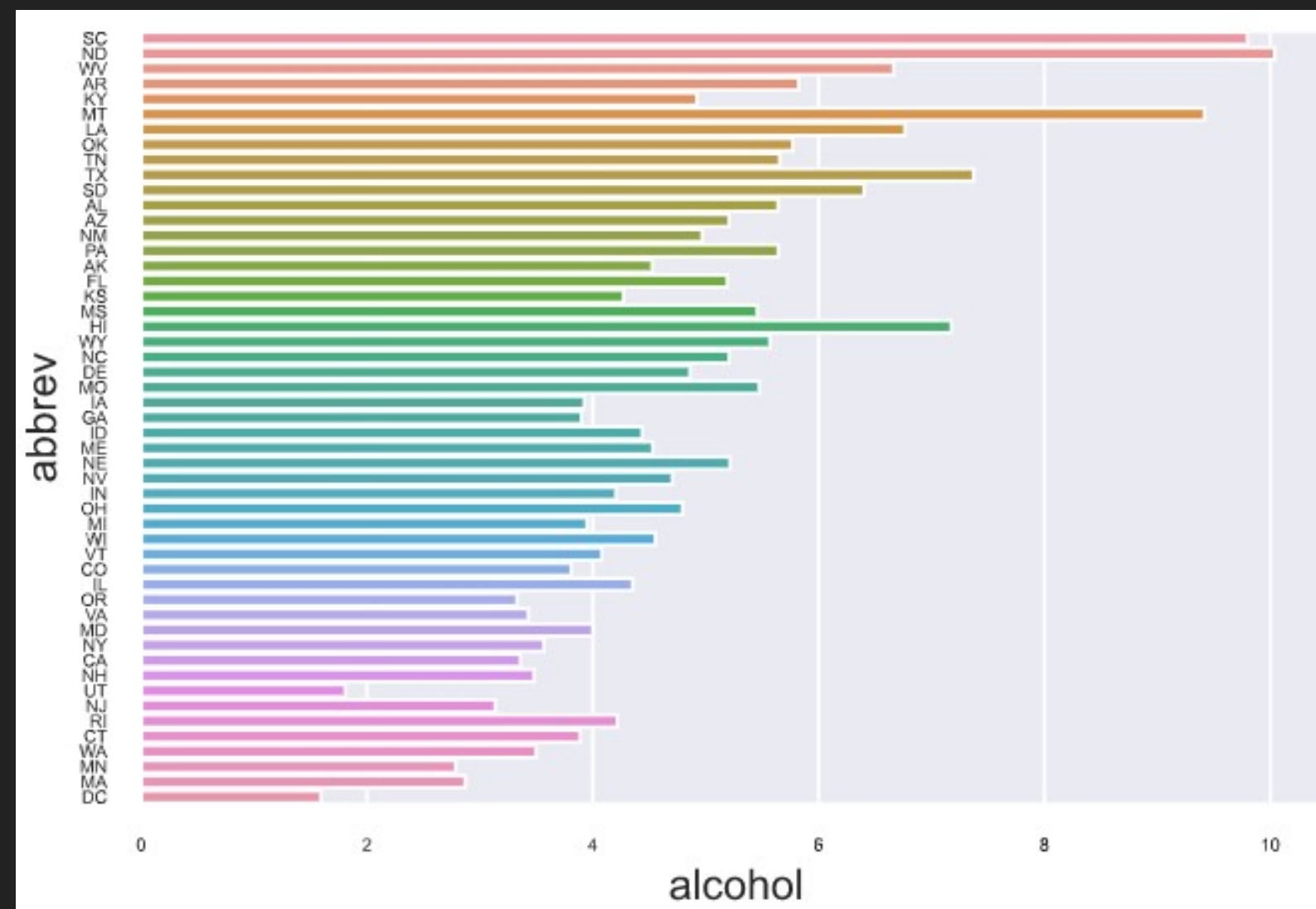


# Gráfico de barras

```
import seaborn

seaborn.set_color_codes("pastel");
crashes = seaborn.load_dataset("car_crashes").sort_values("total", ascending=False);

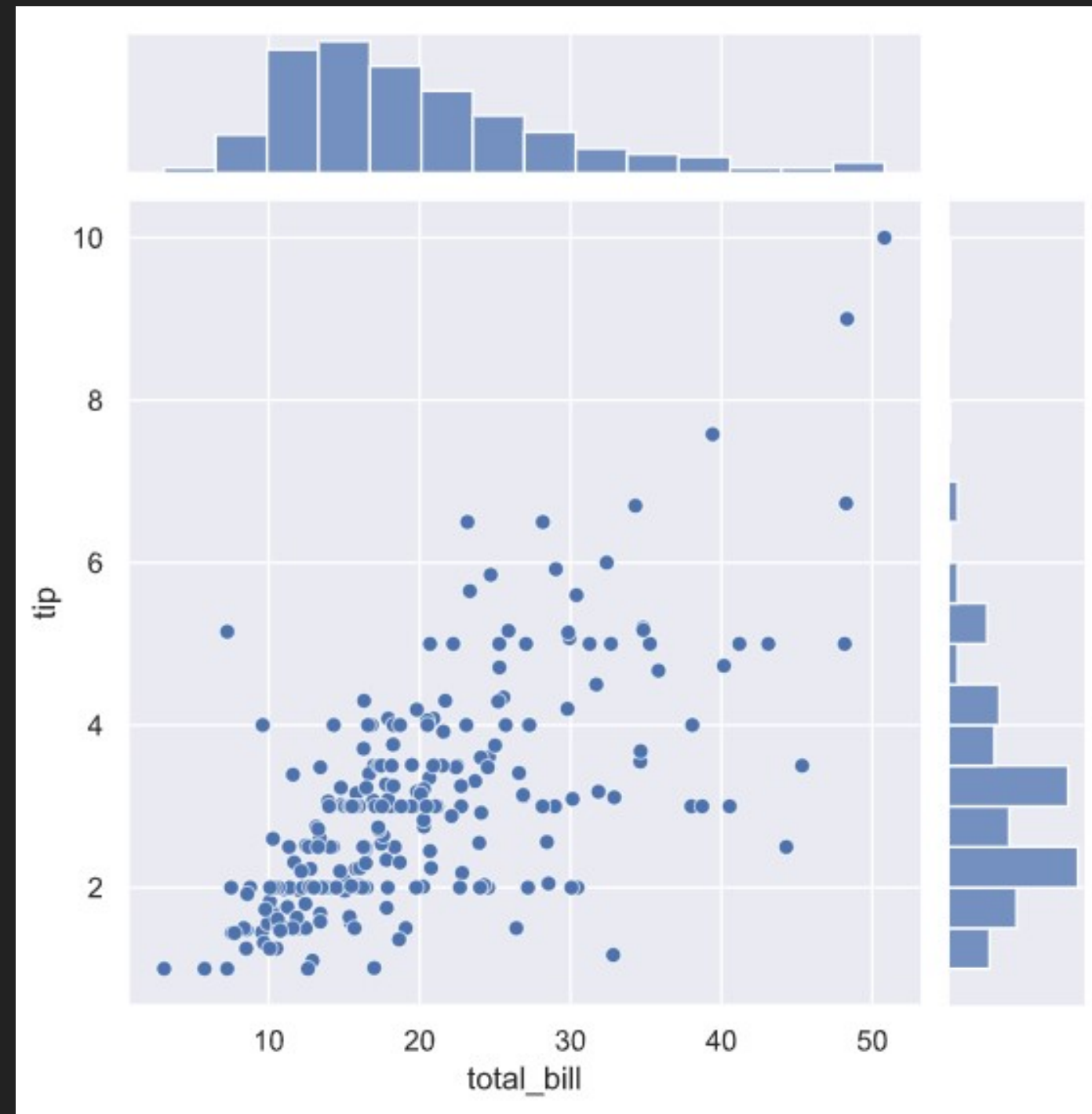
grafica = seaborn.barplot(x="alcohol", y="abbrev", data=crashes, label="Alcohol-involved");
grafica.tick_params(labelsize=5)
```



# Gráfico de dispersión

```
import seaborn
```

```
tips = seaborn.load_dataset("tips");  
seaborn.jointplot(x="total_bill", y="tip", data=tips);
```

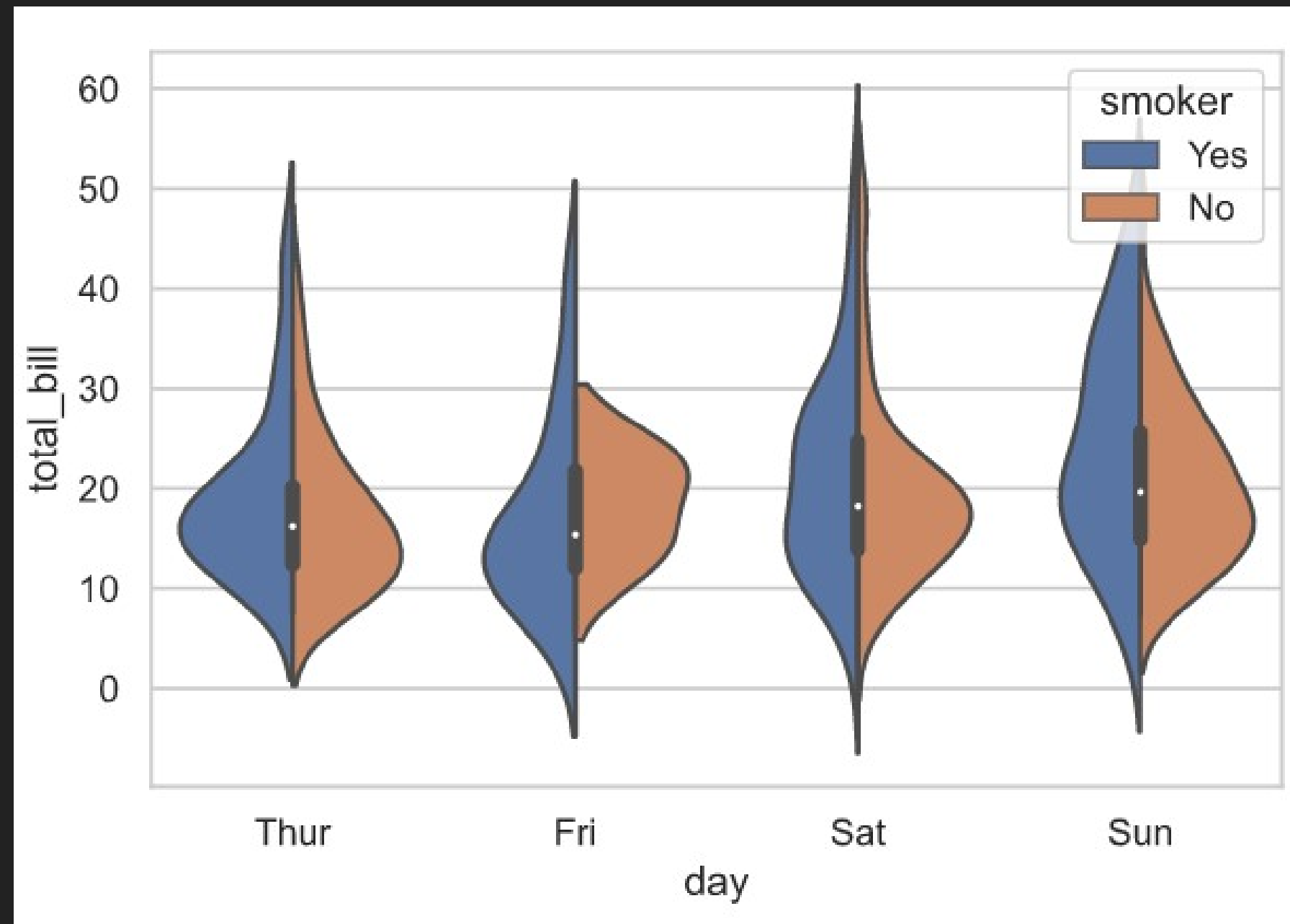


# Gráfico de violín

```
import seaborn

tips = sns.load_dataset("tips");

seaborn.violinplot(data=tips, x="day", y="total_bill", hue="smoker", split=True);
```



# Datos

Epifanio Suárez Martínez

# Datos

Los datos para la IA es lo más importante. Sin buenos datos tendremos resultados mediocres y eso con suerte.

Por lo tanto, lo más importante sobre todo en aprendizaje automatico y aprendizaje profundo, es tener unos buenos datos, y aquí la calidad es importante.

Como hemos hablando antes en las IA, el tiempo de preprocesamiento de los datos, es donde más tiempo gastamos, porque tener unos datos sin problemas, es algo utópico.

Por ello el tratamiento de los datos y su calidad es algo importante.

# Calidad de los datos

La calidad de los datos se puede asociar fundamentalmente por las siguientes características.

Nuestro objetivo es conseguir la mayor calidad de datos que nos sea posible.

- ☐ Precisión
- ☐ Consistencia
- ☐ Uniformidad
- ☐ Validez
- ☐ Completamiento

## Precisión

La precisión es el grado en el que los datos se acercan a los valores reales.

Podemos detectar los valores no válidos, pero eso no significa que sean precisos.

- ❑ Rango de valores, es muy posible que su precisión no sea suficiente.
- ❑ Datos parciales, puedo tener el país de una persona, pero desconozco realmente donde vive.
- ❑ Valores decimales, es posible que 1 o 2 decimales no sean suficientes.

# Consistencia

Ocurren problemas de consistencia cuando un conjunto de datos se contradicen entre sí.

- ❑ Podemos tener 10 años, pero no estar divorciados.
- ❑ Una misma persona con registros diferentes en distintas tablas.
- ❑ La falta de datos también es un problema de consistencia a veces.



# Uniformidad

Es el grado en que se especifican los datos utilizando una misma unidad de medida.

Hay que tener especialmente cuidado con ello, al igual que con los datos extremos, pues puede ser un sintoma de mala uniformidad.

- ☐ La fecha o el peso cambia dependiendo del país.
- ☐ Tenemos distintas monedas.
- ☐ La falta de control al recoger los datos puede provocarlo.

# Validez

Como los datos se ajustan a las reglas, como sufren restricciones por el negocio o las herramientas.

- ☐ Las restricciones con los tipos de datos, o con los rangos.
- ☐ Valores únicos, que deben ser únicos.
- ☐ Valores que deben detectarse con expresiones regulares.
- ☐ Validaciones de datos cruzados.

# Completamiento

El grado en el que conocemos todos los datos necesarios.

Puede que no tengamos todos los datos que necesitamos para desarrollar una IA.

- ☐ Puede ser que podamos completarlos.
- ☐ Puede que no se pueda obtener y tengamos que readptar el alcance o adaptarnos a ello.

# Limpieza de datos

Teniendo en cuenta todos los puntos anteriores, y buscando el objetivo de tener la mayor calidad en los datos, podemos tener un proceso iterativo para mejorar la calidad de los mismos.

- ☐ Inspección: Detectar datos incorrectos, inconsistentes, ...
- ☐ Limpieza: Ajustar los datos, eliminar las anomalías y ceñirse a los requisitos que tengamos.
- ☐ Verificación: Verificar los datos tras los cambios.

Es importante que en todo el proceso de transformación de los datos brutos quede registrado, para poder reproducirlo o detectar posibles problemas. En más de una ocasión nos va a venir muy bien.

# Inspección

La inspección es la tarea que más tiempo consume, ya que requiere que revisiones con distintos objetivos. Es un proceso que debemos tener en cuenta durante todo el proceso.

Como en otros momentos del trabajo con datos, un método muy útil es ir planteandose preguntas. E incluso cada pregunta es una buena practica acompañarlo de visualizaciones.

## Ejemplos

- ☐ ¿Qué valores faltan?
- ☐ ¿Los valores tienen formatos correctos?
- ☐ ¿Hay inconsistencias en los valores?

# Limpieza

A pesar de que la limpieza depende de los datos en si, hay varias tecnicas que podemos valorar para cada tipo de problema.

## Tratamiento de datos irrelevantes

La opción más sencilla de reducir nuestros datos, es quitar aquellos que no se necesitan, y no encajan en el contexto del problema.

Si queremos predecir un estado de salud, su número telefono, quizás no sea útil.

## Tratamiento de datos duplicados

Detectar y eliminar los datos que se repiten innecesariamente.

## Conversión de tipos

Tenemos que verificar que no tengamos problemas a la hora de manejar los datos. Por ejemplo, los datos categoricos deben ser números para poder ser modelados. Cuidado con las fechas y los números decimales y enteros.

# Limpieza

## Errores de sintaxis

Cuidado con datos que no estan escritos correctamente, e incluso en las cadenas de texto con los espacios de más.

## Pad strings

A veces podemos tener números en string para que siempre tengan el mismo número de digitos.

## Corrección de errores tipográficos

Siempre que se meta un dato y no haya una validación corremos el riesgo de que haya muchos errores. Spain, SPAIN, española, Espana, esp, Espania, ...

## Estandarización

Normalmente no tenemos todos los datos en el mismo formato estandar. E incluso a veces lo necesitamos en otro formato.

- ☐ Los textos es ideal que todos esten en mayúsculas o minúsculas.
- ☐ Los números deben tener todos la misma unidad de medida.
- ☐ Las fechas, cada pais lo registramos de una manera distinta.

# Limpieza

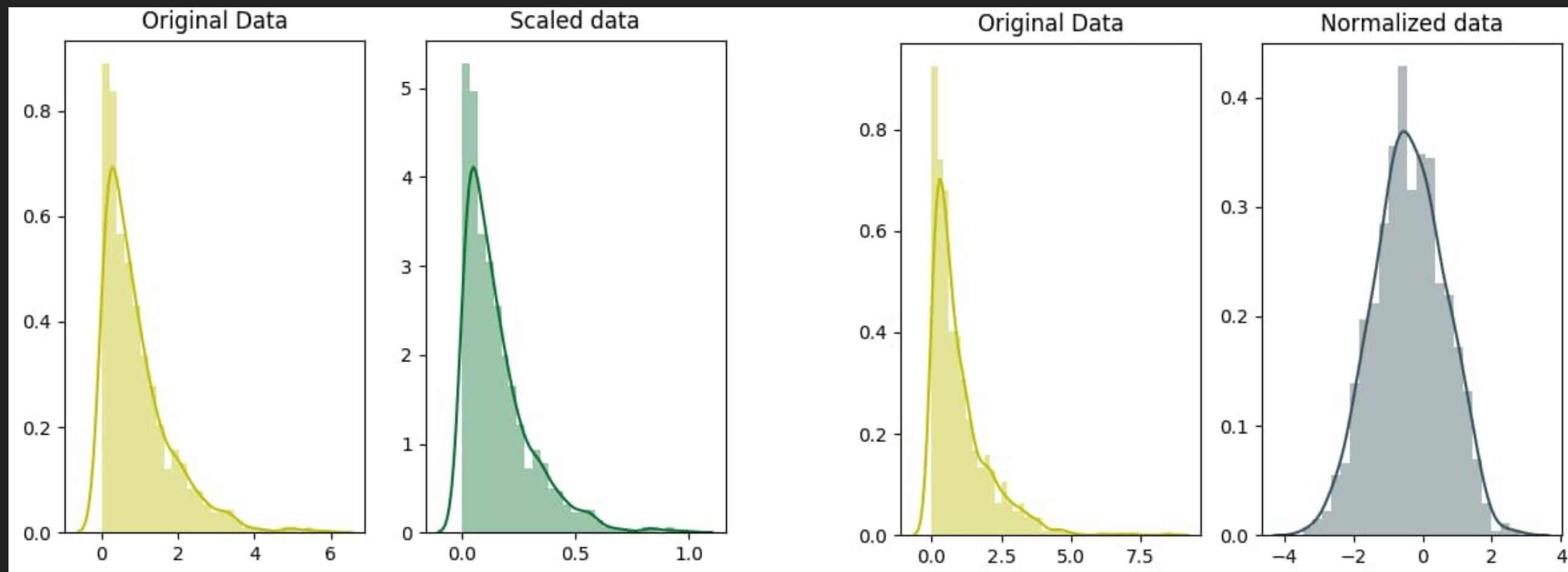
## Escalado

Escalar uno dato es ajustar los datos a una escala específica, por ejemplo, números entre 0 y 1.

De esta manera datos que no son comparables, podemos realizarlo. Lo más usado es el logaritmo, y la raíz cuadrada.

## Normalización

La normalización también es un cambio de escala en un rango de 0 y 1, aunque en este caso la intención es transformar los datos para que se distribuyan normalmente.





# Limpieza

## Tratamiento de valores faltantes

Aunque los datos faltantes son inevitables, no debemos ignorarlos, aunque el algoritmo sea capaz de procesarlo.

## Podemos realizar las siguientes acciones

- ☐ Eliminación
- ☐ Imputación
- ☐ Bandera

# Limpieza

## Eliminación

A la hora de detectar valores perdidos en una columna puede pasar dos circunstancias, que ocurran de una manera al azar, por lo que podremos eliminar las filas donde ocurra, o la otra circunstancia si es mayoría los datos faltantes, entonces podemos eliminar la columna.

## Bandera

Un problema que hay al completar los valores faltantes lleva a una pérdida de información, pues la falta es de por sí, información.

Por ello, se suele recomendar que las transformaciones de los valores se realicen en nuevas columnas.

# Limpieza

## Imputación

Esta tecnica se enfoca en calcular el valor faltante mediante otras observaciones, de tal manera que se intenta rellenar de la mejor manera posible.

- ☐ Podemos usar datos estadísticos como la media o la mediana, aunque no garantiza que los datos no esten sesgados.
- ☐ En datos distribuidos normalmente se puede obtener todos los valores que están dentro de 2 desviaciones estándar de la media. ( $\text{media} - 2 * \text{estandar}$ ) y ( $\text{media} + 2 * \text{estandar}$ );
- ☐ También nos podemos apoyar a una regresión lineal para calcular los valores faltantes.
- ☐ Y por último podemos copiar valores de otros registros similares.

## Limpieza

Un valor faltante no es lo mismo que un valor predeterminado.

En algunas prespuestas de preguntas la ausencia de datos no quiere decir que sea una respuesta de SI o NO.

Una persona que no haya respondido nada, puede significar un no recuerdo.

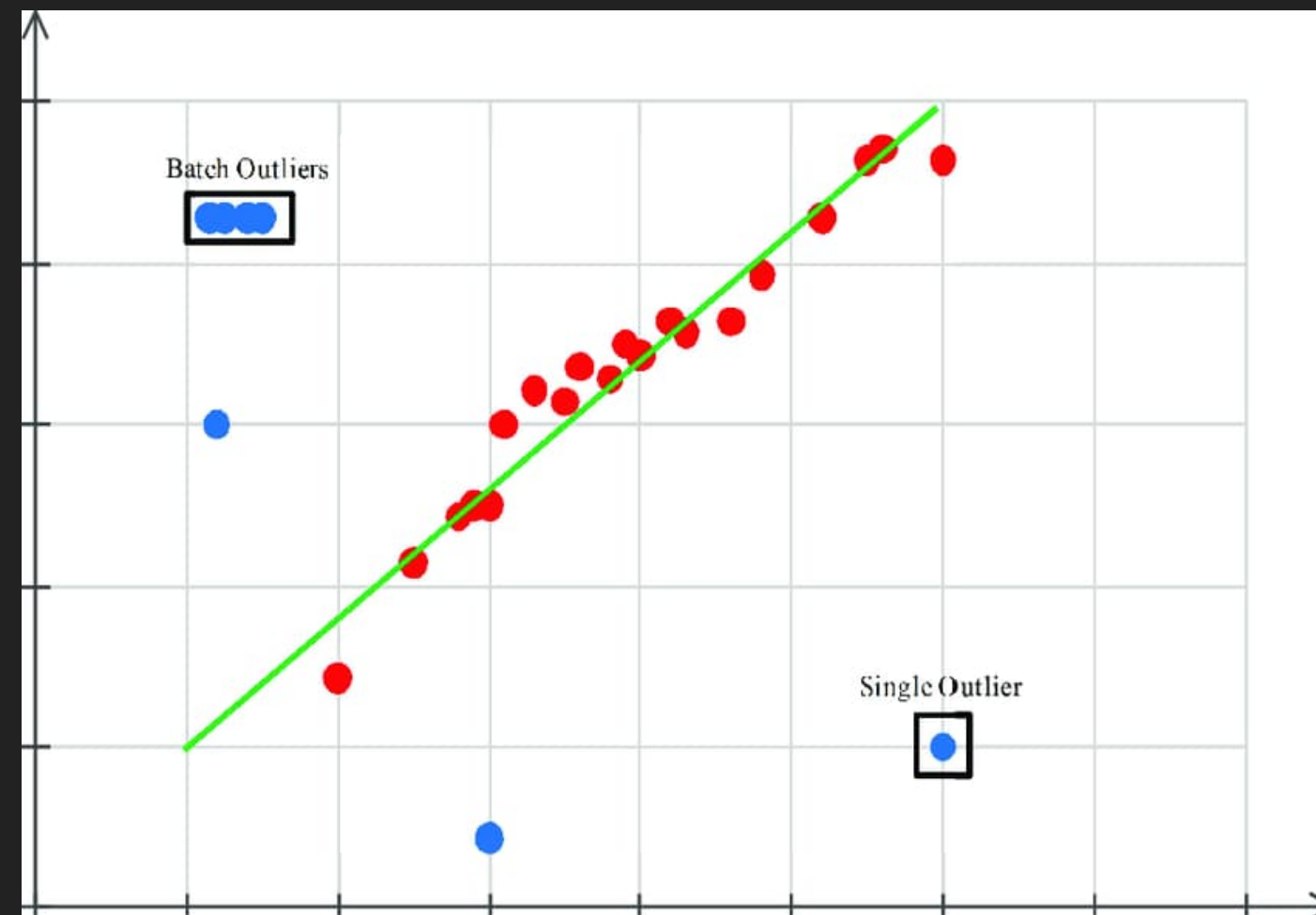
Cada dato hay que tratarlo individualmente, pues cada uno de ellos hay que enfrentarlo de manera distinta.

# Limpieza

## Tratamiento de datos extremos

Los valores extremos son diferentes en todas las observaciones. ¡Cuidado! Hay casos como los del cuarteto de anscombe [https://es.wikipedia.org/wiki/Cuarteto\\_de\\_Anscombe](https://es.wikipedia.org/wiki/Cuarteto_de_Anscombe) que tienen las mismas propiedades estadísticas, pero sus datos son distintos al ver sus gráficos.

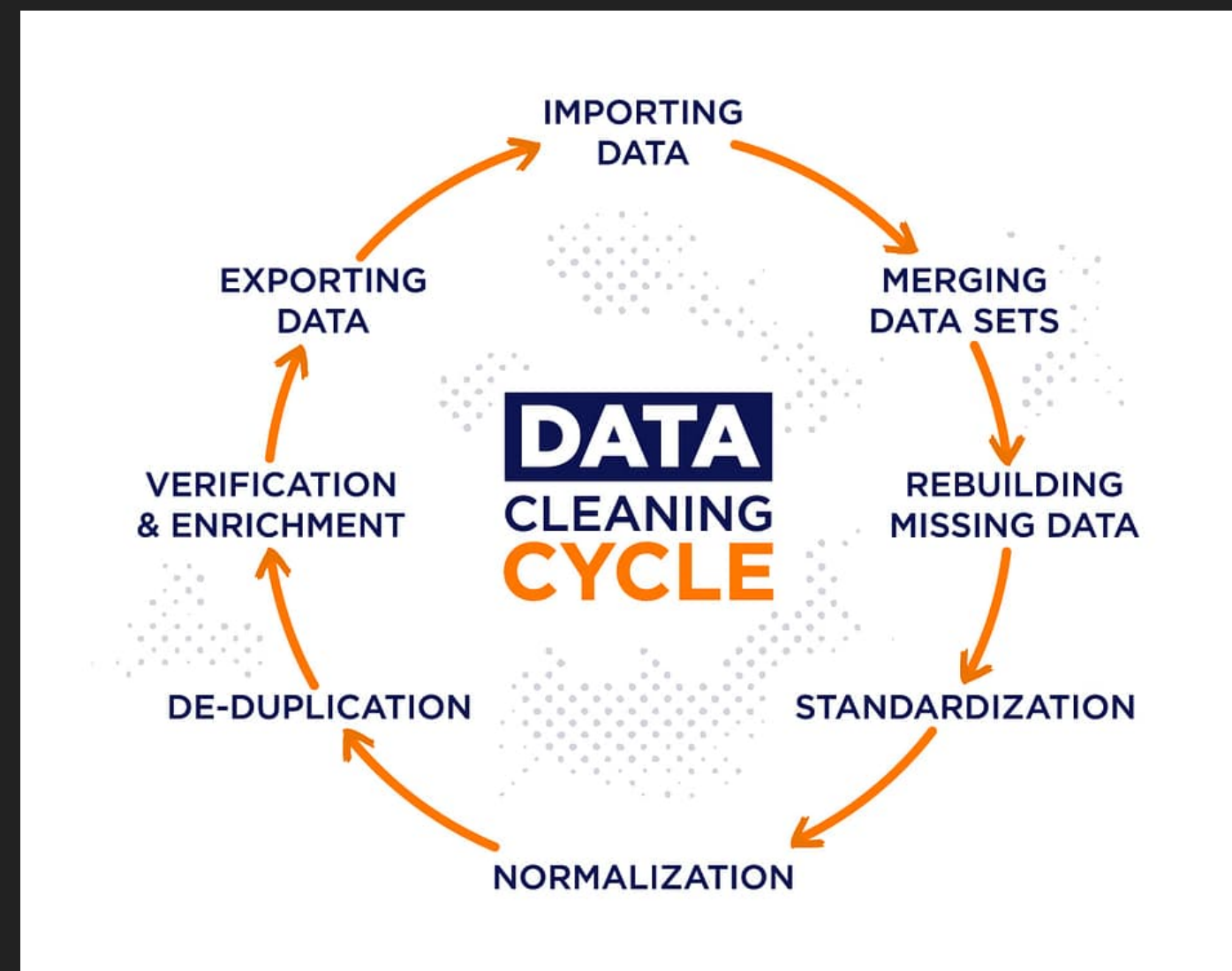
Por lo que hay que investigar antes de eliminarlos. Sobre todo con las regresiones lineales que son más sensibles.



# Limpieza

## Verificación

Tras cada modificación se trata de comprobar y verificar que todo lo que hemos tratado va por el bien camino.



# Entrenamiento de modelos

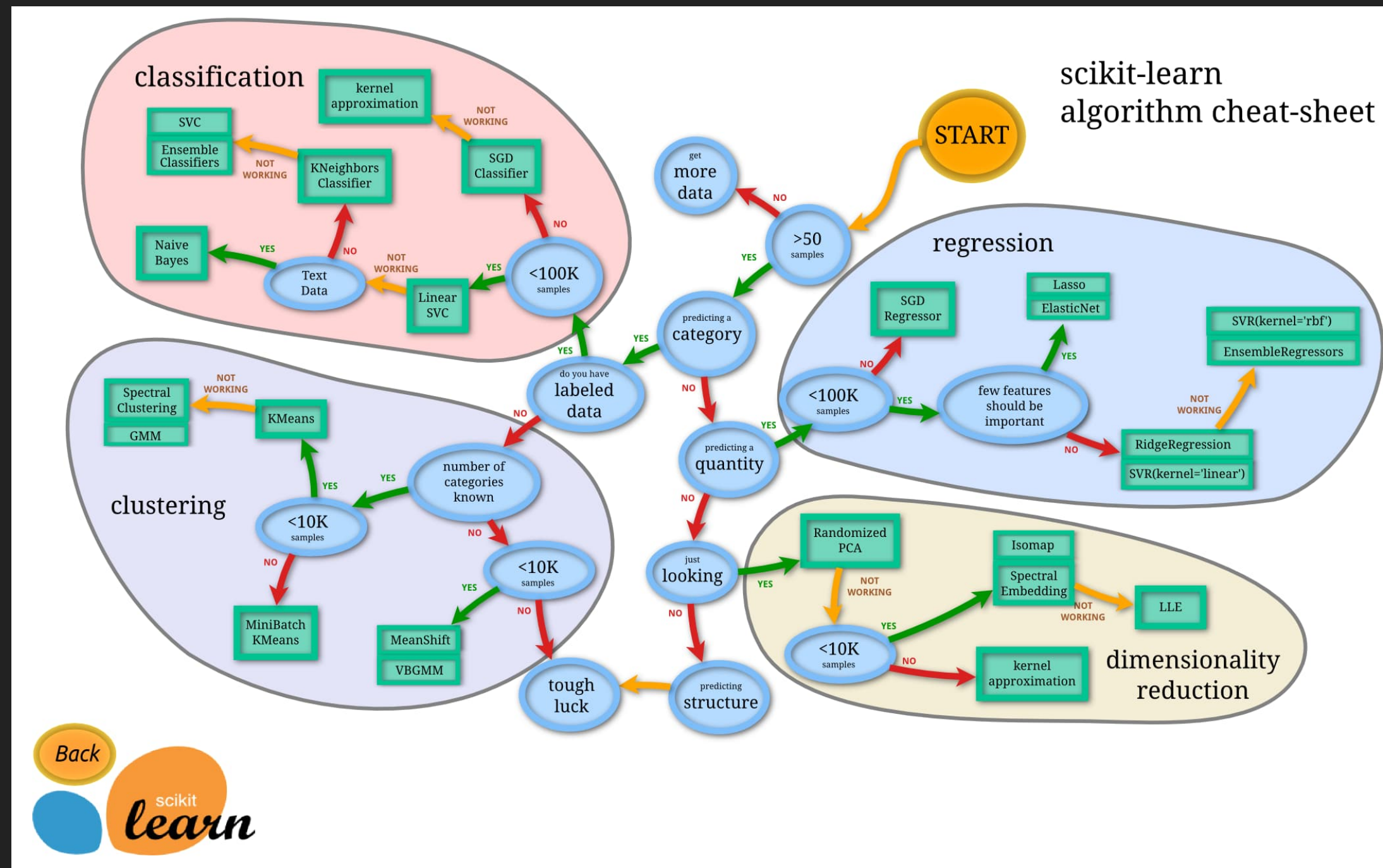
Epifanio Suárez Martínez



# Entrenamiento de modelos

Se trata de analizar que modelo es el que mejor se ajusta a nuestros datos, y de conseguir que el modelo elegido generalice de la mejor manera posible.

Una vez que tengamos el modelo entrenado podremos predecir resultados, y es aquí donde nos ayuda a resolver problemas.





## Proceso de entrenamiento

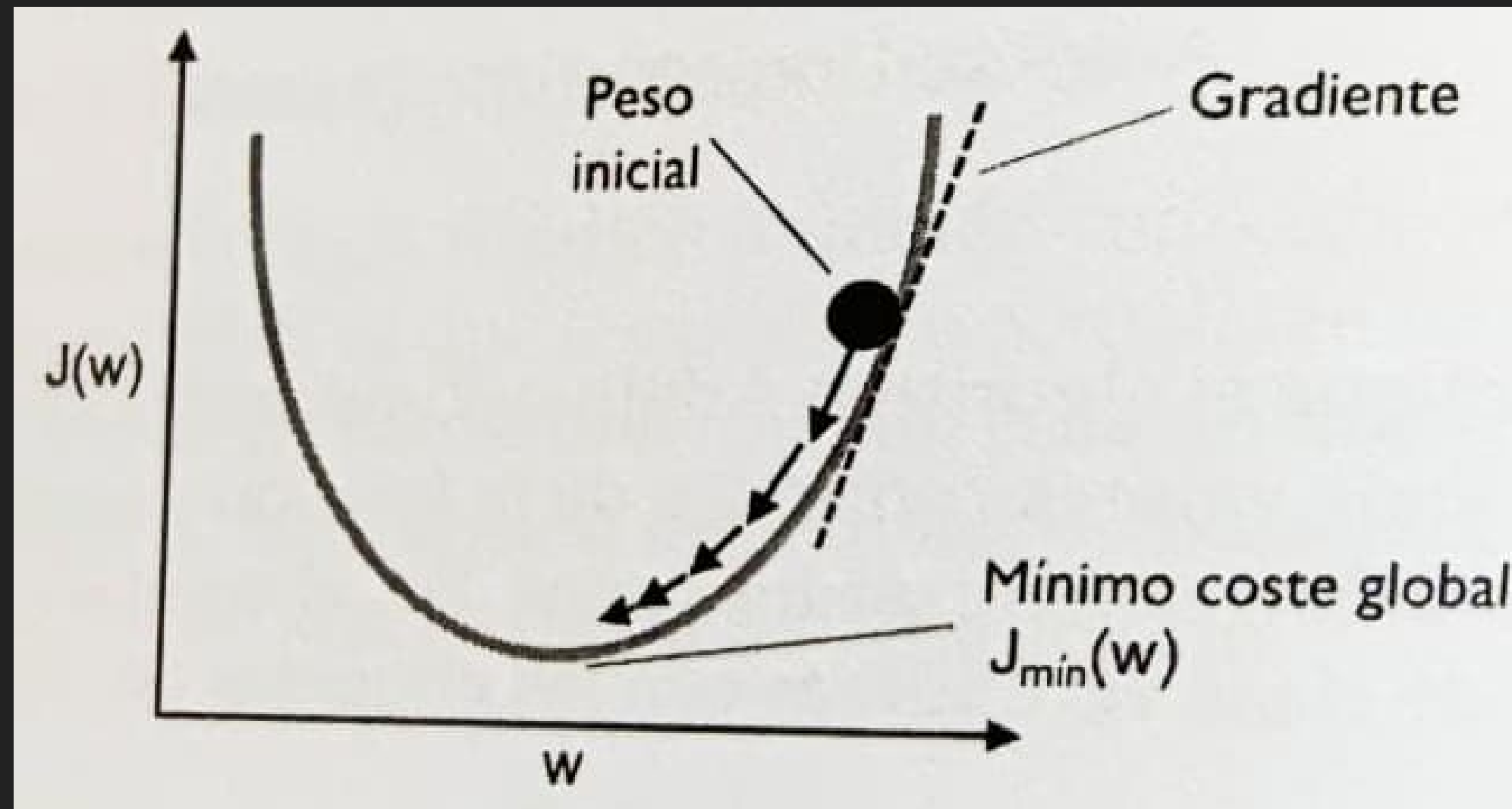
El proceso de entrenamiento como en otros procesos en IA, es un proceso iterativo. Un buen camino sería seguir estos pasos.

1. Seleccionar características y recopilar muestras de entrenamiento
2. Elegir una medición del rendimiento
3. Elegir un algoritmo
4. Evaluar el rendimiento del modelo
5. Afinar el algoritmo

# Objetivo

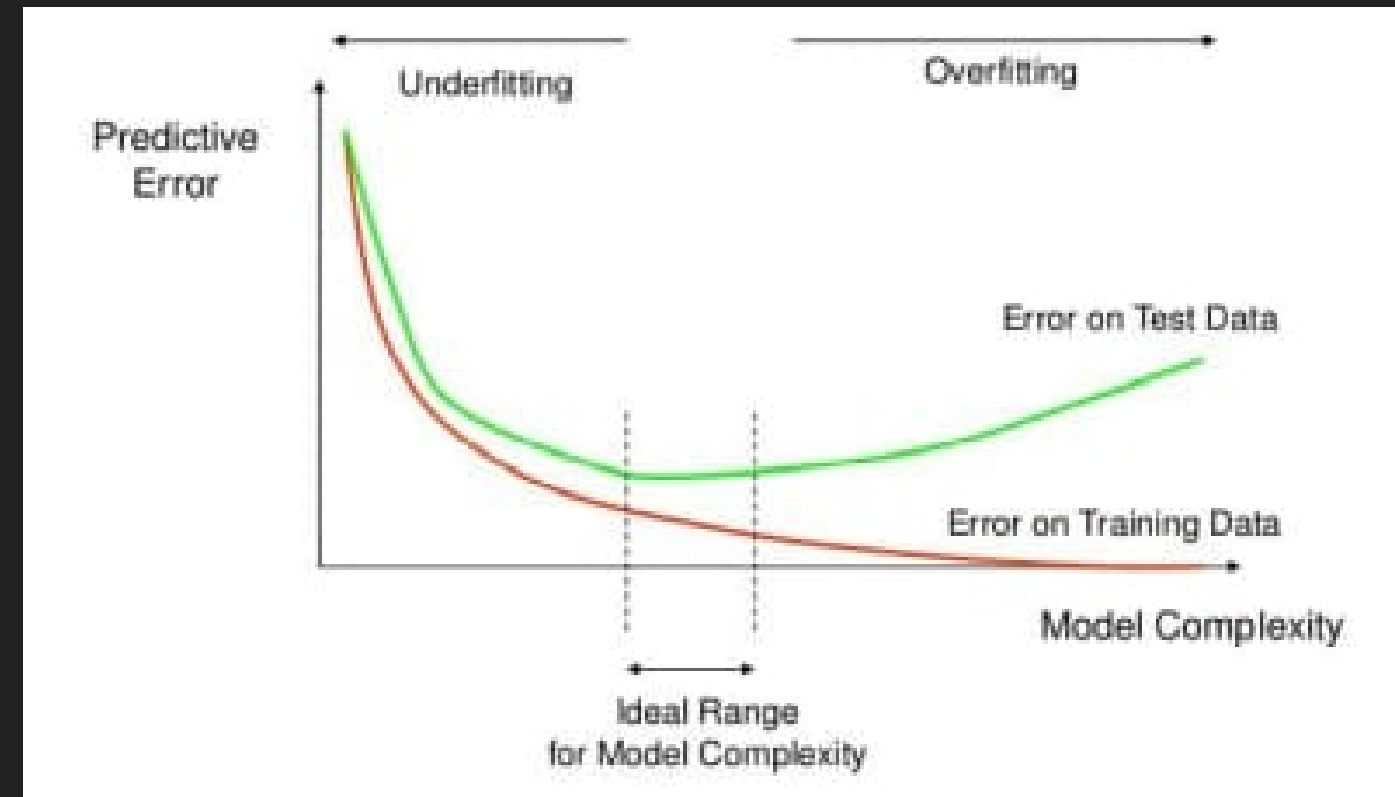
Nuestro objetivo a la hora de trabajar con cualquier algoritmo de ciencia de datos, es el de ajustar los pesos para que supongan el mínimo coste con la mayor eficacia.

¡No se trata de acertar al 100% como una máxima!



# Sobreajuste / Subajuste

Una de las principales causas de tener malos resultados es el sobreajuste o el subajuste, también conocidos como **overfitting** y **underfitting**.



## Sobreajuste

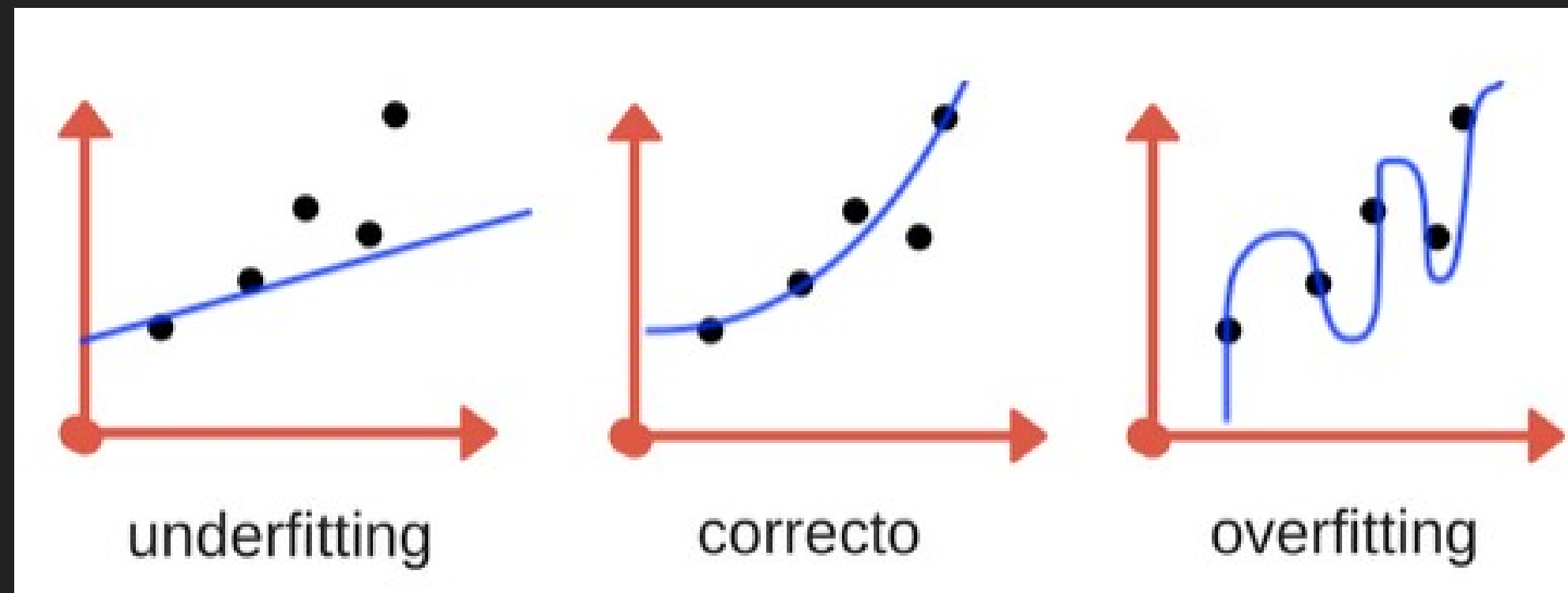
- ☐ Cuando entrenamos demasiado un modelo
- ☐ Cuando tenemos pocos datos
- ☐ Cuando hay datos anómalos
- ☐ Cuando tenemos modelos demasiados complejos

# Sobreaajuste / Subajuste

## Subajuste

- ❑ Cuando no hay suficientes parámetros o complejidad
- ❑ Cuando no se le da suficiente tiempo al algoritmo para entrenar

**El problema está al generalizar. Y la clave está en encontrar un punto medio.**



# Métricas

Las **métricas** que podemos sacar de los modelos entrenados nos ayudan a evaluar los mismos.

**No podemos sacar las mismas métricas de todos los modelos.**

## Regresión

Error medio absoluto, Error cuadrático medio, Raíz error cuadrático medio

## Clasificación

Matriz de confusión, Métrica de exactitud, Métrica de exhaustividad, Métrica de precisión, Puntuación F1

## Cluster

Curva de Elbow, Silhouette

# Regresión

## Error medio absoluto

Contra mayor sea el valor, peor es el modelo. Nunca puede ser negativo.  
0 sería un modelo perfecto.

```
error_medio_absoluto = mean_absolute_error(y_datos, y_prediccion);
```

## Documentación

## Error cuadrático medio

Contra mayor sea el valor, peor es el modelo. Nunca puede ser negativo.  
0 sería un modelo perfecto.

```
error_cuadratico_medio = mean_squared_error(y_datos, y_prediccion);
```

## Documentación

# Regresión

## Raíz error cuadrático medio

Contra mayor sea el valor, peor es el modelo. Nunca puede ser negativo.  
0 sería un modelo perfecto.

```
raiz_error_cuadratico_medio = numpy.sqrt(error_cuadratico_medio);
```

# Clasificación

## Matriz de confusión

La **matriz de confusión** es una representación de los resultados de las predicciones.

```
matriz_confusion = confusion_matrix(prediccion_y_entrenamiento, y_entrenamiento);
```

-	POSITIVO	NEGATIVO
Positivo	42 (Verdaderos positivos)	0 (Falsos positivos)
Negativo	1 (Falsos negativos)	25 (Verdaderos negativos)

## Documentación

## Métrica de exactitud

La **métrica de exactitud** nos indica el porcentaje de casos que el modelo ha acertado. No siempre es una buena opción, porque puede indicar que un modelo malo, parezca mejor de lo que es.

```
precision = accuracy_score(prediccion_y_entrenamiento, y_entrenamiento);
```

## Documentación



# Clasificación

## Métrica de exhaustividad

Nos indica la cantidad de valores que hemos acertado correctamente.

```
exhaustividad = recall_score(y_datos, y_prediccion);
```

## Documentación

## Métrica de precisión

Nos indica la precisión del modelo.

```
precision = precision_score(y_datos, y_prediccion);
```

## Documentación

## Puntuación F1

El valor F1 combina las medidas de precisión y exhaustividad. Es la media armónica entre ambas.

```
puntuacion_f1 = f1_score(y_datos, y_prediccion);
```

## Documentación

# Clasificación

## Curva de Elbow

La **curva de elbow** sirve para detectar cuando llegamos al **codo** y evitar seguir trabajando sin obtener mejoría.

```
numero_cluster = range(1, 10)

kmeans = [KMeans(n_clusters=i) for i in numero_cluster];
score = [kmeans[i].fit(x).score(x) for i in range(len(kmeans))];
```

## Silhouette

El mejor valor es un 1, el peor es un -1.

```
silhouette = silhouette_score(x, etiquetas);
```

## Documentación

# Resumen

## Regresión

Puede predecir un valor con base a unos datos de ejemplo. Por ejemplo, predecir el tiempo que hará mañana.

## Clasificación

Nos permite clasificar o etiquetar valores a partir de unos datos previamente clasificados. Por ejemplo, clasificar animales por su especie.

## Clusterización

En este caso nos permite encontrar aspectos en común para formar grupos. Por ejemplo, clasificar animales por el número de patas.

# Datos de entrenamiento, pruebas y validación

Podemos entrenar un modelo con el total de nuestros datos, pero eso no asegura que un éxito. **Seguramente sea un fracaso.**

Los primeros pasos que podemos hacer es dividir nuestros datos. Podemos dividir los mismos en los siguientes conjuntos de datos.

- ❑ Entrenamiento: Los datos para el entrenamiento del modelo
- ❑ Validación: Los datos para la selección y optimización del modelo.
- ❑ Prueba: Los datos para estimar el error del modelo seleccionado.



## ¿Cómo evitamos estos problemas?

- ❑ Elegir o probar una **cantidad mínima de muestras** tanto para entrenar como para validar. Ejemplo: **train\_test\_split**
- ❑ **Tener datos equilibrados**, es decir, tener más o menos las mismas cantidades de datos de distintos tipos.
- ❑ **Trabajar con un conjunto de datos de prueba** para poder validar los aciertos/errores y poder detectar estos problemas.
- ❑ **Iterar sobre los entrenamientos y los parámetros** para encontrar el equilibrio en el entrenamiento.
- ❑ Si tenemos **cantidad excesiva de dimensiones o de parámetros**, sin suficientes muestras podemos caer en estos problemas.

Si el modelo entrenado con los datos de entrenamiento tiene un 90% de aciertos y el test un porcentaje mucho más bajo, es un problema de **sobreajuste**. Y si solo acierta un mismo tipo de valor, es que tenemos un problema de **subajuste**.

## Consejos para mejorar el modelo

- ❑ Se trata de conseguir que la máquina generalice.
- ❑ Los datos no lo son todo, aunque sean importantes.
- ❑ El subajuste o el sobreajuste tienen muchas caras, siempre analizar los resultados.
- ❑ ¡Cuidado con las dimensiones! No siempre es bueno entrenar un modelo con 100 parámetros.
- ❑ Contrastar datos y resultados. Variar un parámetro puede cambiar el resultado.
- ❑ Una de las claves es detectar que parámetros son los importantes.
- ❑ Siempre probad de modelos más sencillos a más complejos.
- ❑ Combina distintos modelos. En ocasiones, pueden dar increíbles resultados.
- ❑ Simplificación no implica precisión.
- ❑ No todos los problemas se pueden resolver con aprendizaje automático.
- ❑ La correlación no implica causa.

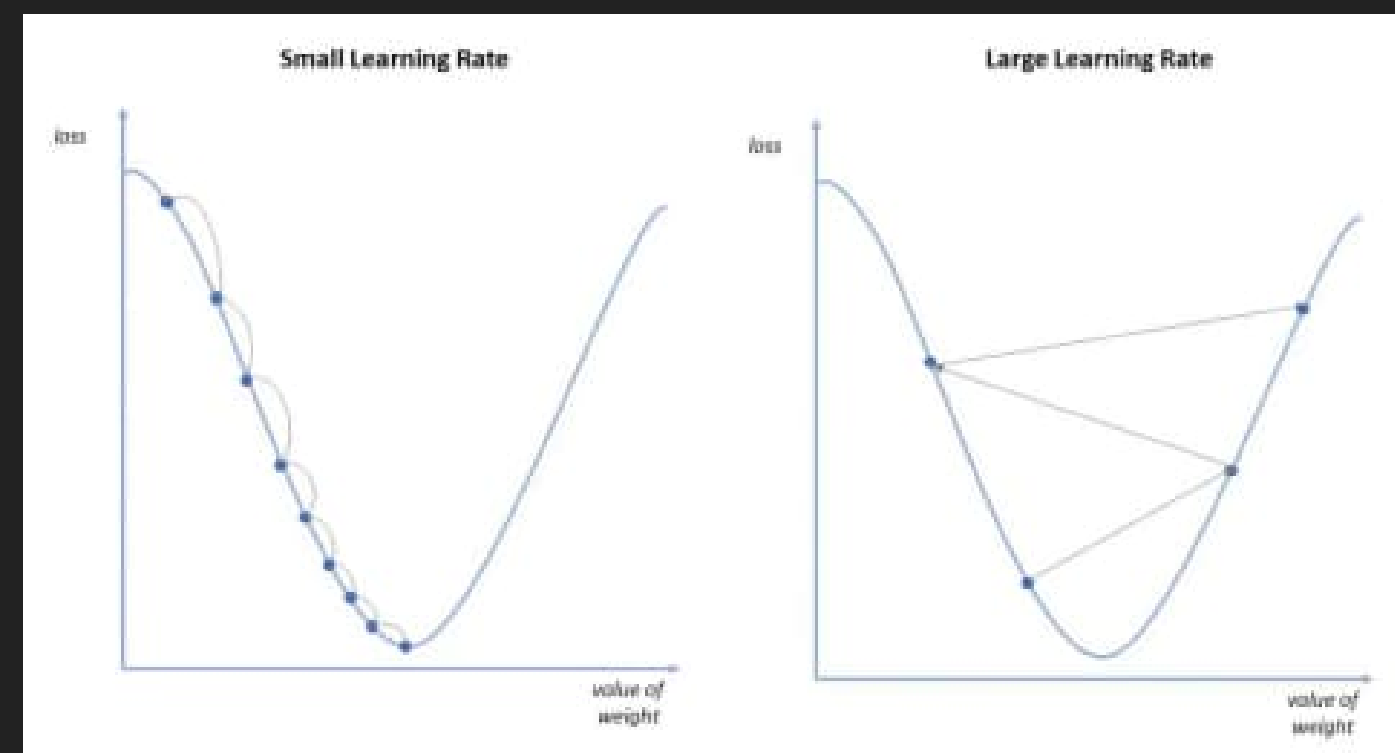
# Tasa de aprendizaje

Es el tamaño de los pasos que se toman para alcanzar el mínimo de la función de coste. También se le conoce con el nombre de tamaño de paso y suele venir representada por la letra alfa ( $\alpha$ ).

Normalmente, se trata de un valor pequeño y se evalúa y actualiza en función del comportamiento de la función de coste.

Las altas tasas de aprendizaje dan lugar a pasos grandes, pero se corre el riesgo de superar el mínimo constantemente.

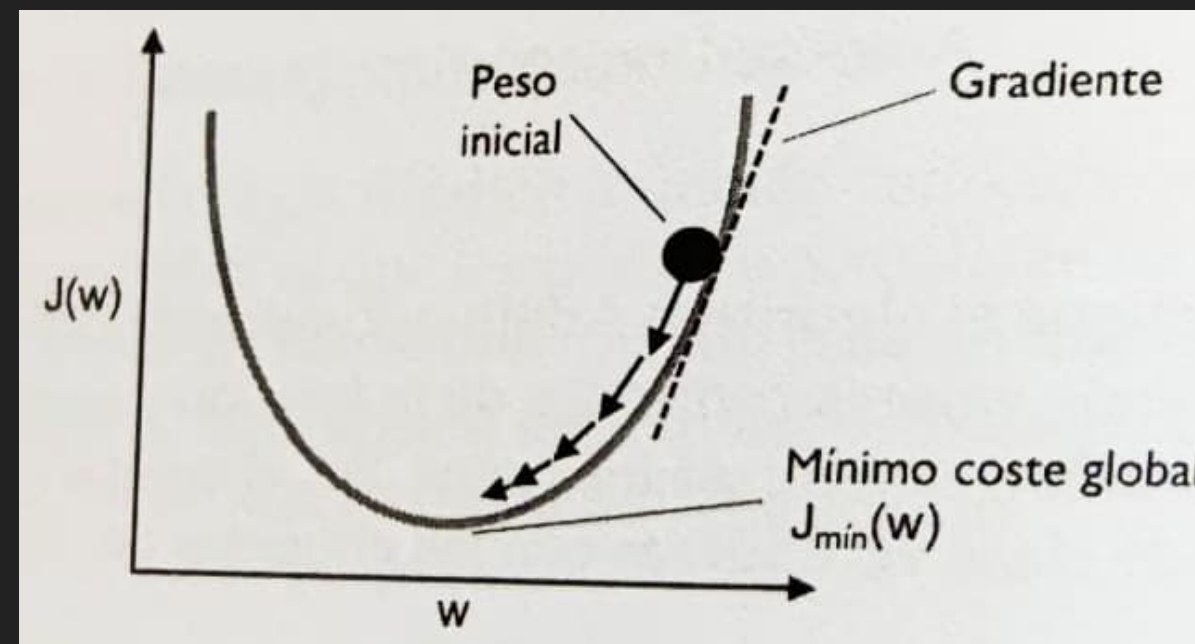
Por el contrario, una baja tasa de aprendizaje implica tamaños de pasos pequeños, lo que implica mucho más tiempo y cálculos durante el entrenamiento hasta conseguir alcanzar el mínimo.





# Descenso por gradiente

El descenso de gradiente es un algoritmo que estima numéricamente dónde una función genera sus valores más bajos.



Se trata de evaluar el apredizaje e ir ajustando los pesos en cada iteración. El descenso por gradiente actúa tomando gradientes de la función de coste, para lo cual es necesario calcular las derivadas parciales con respecto a los parámetros.

Aunque suena muy bien, tiene algunos inconvenientes, como todo en ia, hay que medir, analizar y repetir, hasta estar seguros de que vamos por el buen camino.



# Técnicas para evitar el sobreajuste

Hay algunas técnicas para evitar el sobreajuste que podemos realizar.

- ☐ Validación cruzada
- ☐ Parada anticipada
- ☐ Regularización
- ☐ Modificación de parámetros del modelo

# Validación cruzada

La validación cruzada se puede utilizar para validar los modelos cuando el conjunto de datos es reducido.

Los datos se dividen en N grupos. El algoritmo se utiliza en todos menos uno de los grupos y se prueba en el restante. Después, los grupos son rotados para que el algoritmo aprenda de todos los datos.

```
from sklearn.model_selection import KFold

kfold = KFold(n_splits=5);

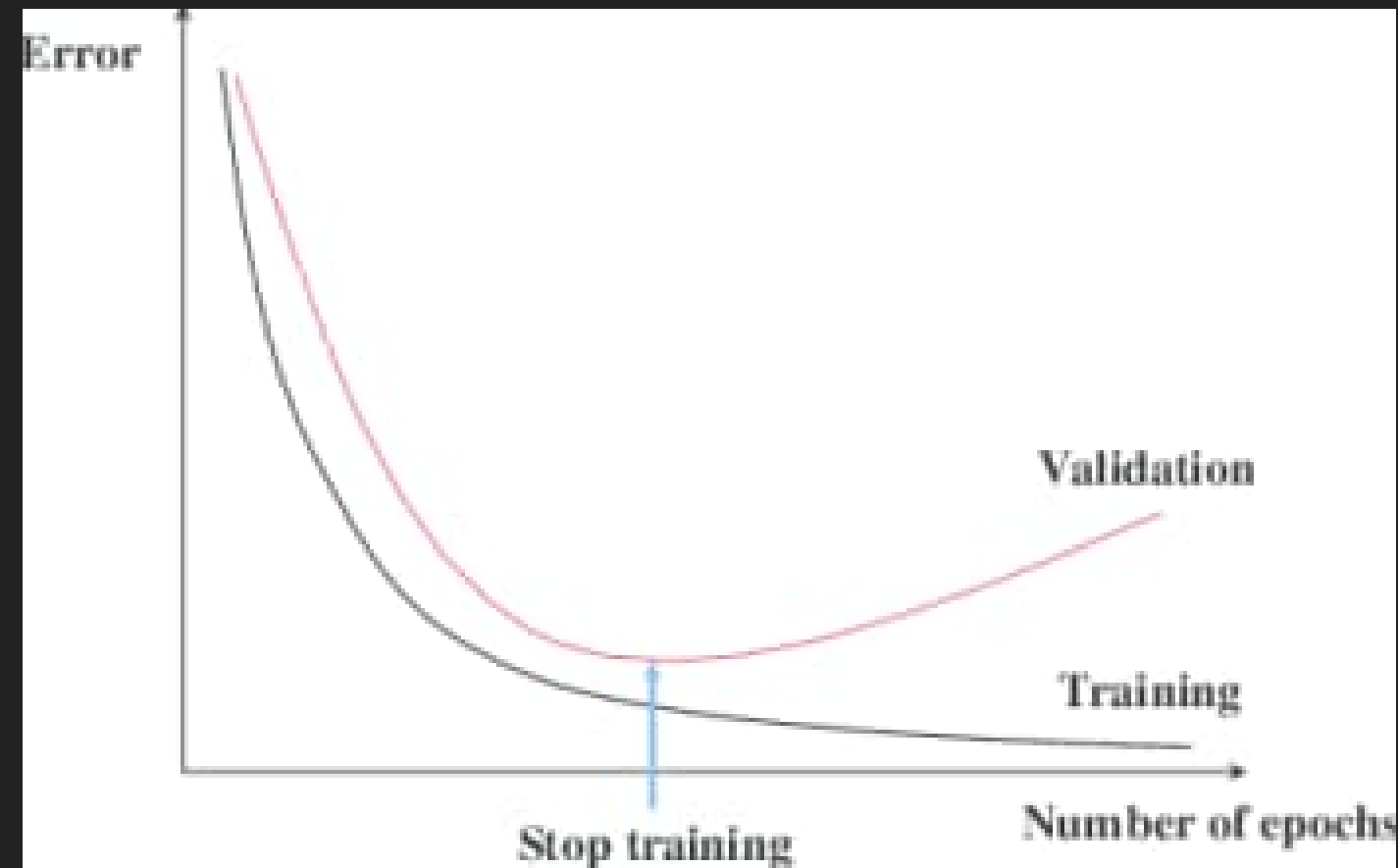
puntuaciones = cross_val_score(modelo, x_entrenamiento, y_entrenamiento, cv=kfold,
                                scoring="accuracy");

print(f"Metricas cross_validation {puntuaciones}");
print(f"Media de cross_validation {puntuaciones.mean()}");
```

	Datos A	Datos B	Datos C	Datos D
Iteración 1	Test	Entrenamiento	Entrenamiento	Entrenamiento
Iteración 2	Entrenamiento	Test	Entrenamiento	Entrenamiento
Iteración 3	Entrenamiento	Entrenamiento	Test	Entrenamiento
Iteración 4	Entrenamiento	Entrenamiento	Entrenamiento	Test

## Parada anticipada

La parada anticipada es una técnica sencilla, que a través de la validación podemos ver cuando es necesario parar en el momento óptimo.



# Regularización

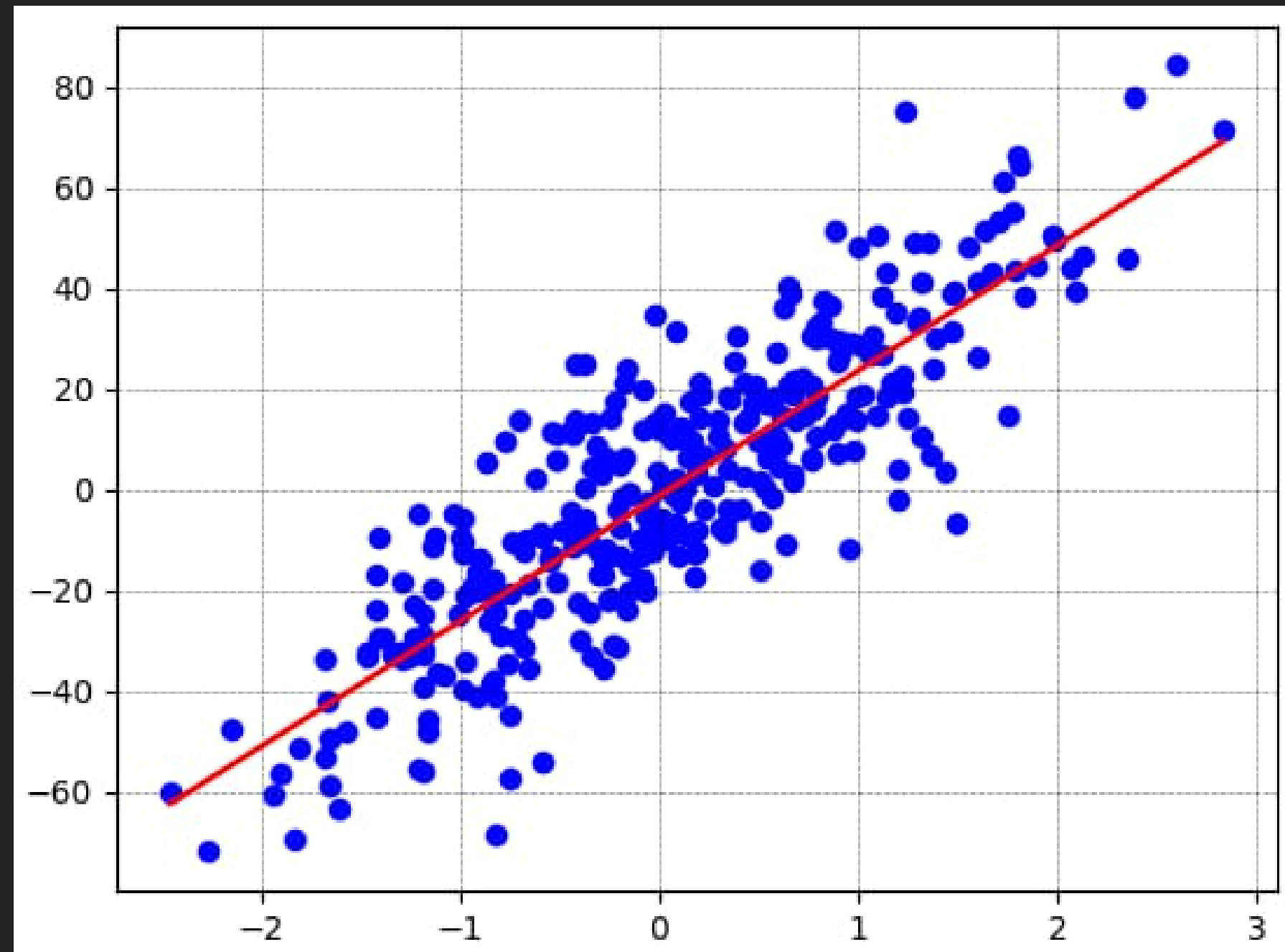
La regularización se utiliza con el fin de minimizar la complejidad del modelo y, a su vez, minimizar la función de coste. Así, se previene el sobreajuste y se consigue que el modelo generalice mejor ante datos nuevos.

Los modelos que lo permiten, tienen las siguientes regularización.

- ☐ Regularización Lasso (L1): Útil cuando algunas de las variables se consideran poco importantes.
- ☐ Regularización Ridge (L2): Útil cuando algunas de las variables estén posiblemente correlacionadas entre sí.
- ☐ Regularización ElasticNet (L1 y L2): Útil cuando hay un gran número de variables.

# Regresión lineal

Es la relación de dependencia entre una variable dependiente y otra independiente. Es un algoritmo de aprendizaje supervisado.



# Regresión lineal

Este algoritmo es un método estadístico que nos permite resumir y estudiar las relaciones entre dos variables continuas cuantitativas.

- ❑ Existe una relación lineal y aditiva, entre las variables dependientes e independientes
- ❑ No debe haber correlación entre las variables independientes.
- ❑ La variable dependiente y los términos de error deben tener una distribución normal.

# Regresión lineal

## Preparamos los datos

```
import matplotlib.pyplot as pyplot
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Obtenemos los datos de prueba
valor_x, valor_y = datasets.make_regression(n_samples=100, noise=15, n_features=1);

# Tomamos muestras de los valores totales
x_entrenamiento, x_prueba, y_entrenamiento, y_prueba =
    train_test_split(valor_x, valor_y, test_size=0.33, shuffle=True);
```

# Regresión lineal

## Entrenamiento y resultados

```
# Entrenamos el modelo
modelo = LinearRegression().fit(x_entrenamiento, y_entrenamiento);

# Realizamos la predicción
prediccion = modelo.predict(x_prueba);

# Representación
pyplot.scatter(x_prueba, y_prueba, c="blue");
pyplot.grid(color="black", alpha=.5, linestyle="--", linewidth = 0.5);
pyplot.plot(x_prueba, prediccion, c="red");
pyplot.show();

# Metricas
error_cuadratico_medio = mean_squared_error(y_prueba, prediccion);
error_absoluto_medio = mean_absolute_error(y_prueba, prediccion);
puntuacion = r2_score(y_prueba, prediccion);

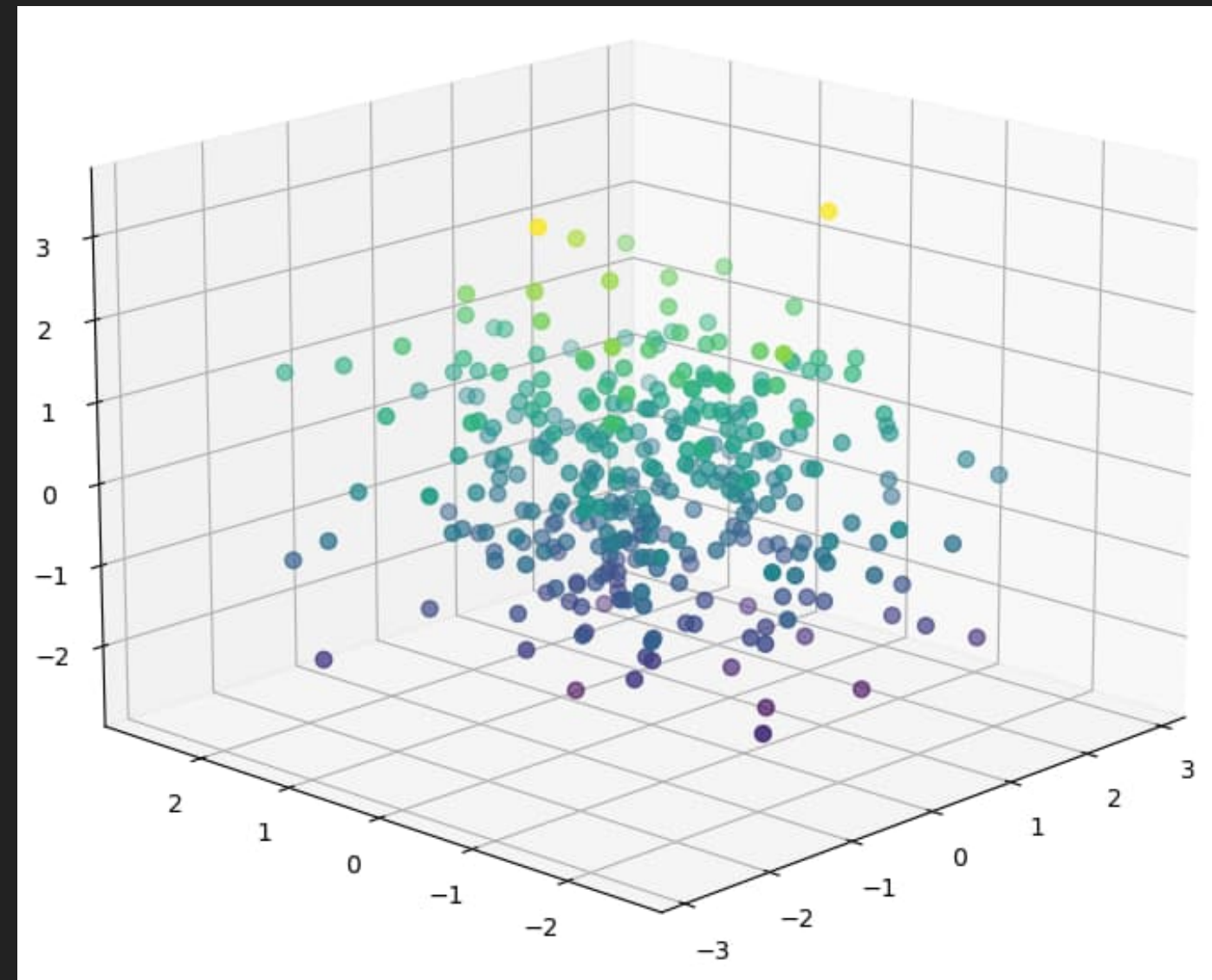
print(f"Error cuadratico medio: {error_cuadratico_medio}");
print(f"Error absoluto medio: {error_absoluto_medio}");
print(f"Puntuación: {puntuacion}");
```



# Regresión múltiple

En este caso es una ampliación en número de variables de una regresión lineal.

La visualización de datos en 3d es una desventaja. Es recomendable que la visualización se haga en 2D para evitar los errores de compresión de los datos.



# Regresión multiple

## Preparamos los datos

```
import matplotlib.pyplot as pyplot
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Obtenemos los datos de prueba
valor_x, valor_y = datasets.load_diabetes(return_X_y=True, as_frame=True);
valor_x = valor_x[["age", "sex", "bmi"]];

figure, plots = pyplot.subplots(len(valor_x.columns));
indice = 0;
```

# Regresión multiple

## Entrenamiento y resultados

```
for columna in valor_x:
    # Tomamos muestras de los valores totales
    x_entrenamiento, x_prueba, y_entrenamiento, y_prueba =
        train_test_split(valor_x[columna].values.reshape(-1, 1), valor_y,
            test_size=0.10, shuffle=True);

    # Entrenamos el modelo
    modelo = LinearRegression();
    modelo.fit(x_entrenamiento, y_entrenamiento);

    # Realizamos la predicción
    prediccion = modelo.predict(x_prueba);

    # Representación en 2D
    plots[indice].title.set_text(columna);
    plots[indice].scatter(x_prueba, y_prueba, color="red");
    plots[indice].plot(x_prueba, prediccion, color="blue");

    indice += 1;

pyplot.show();
```

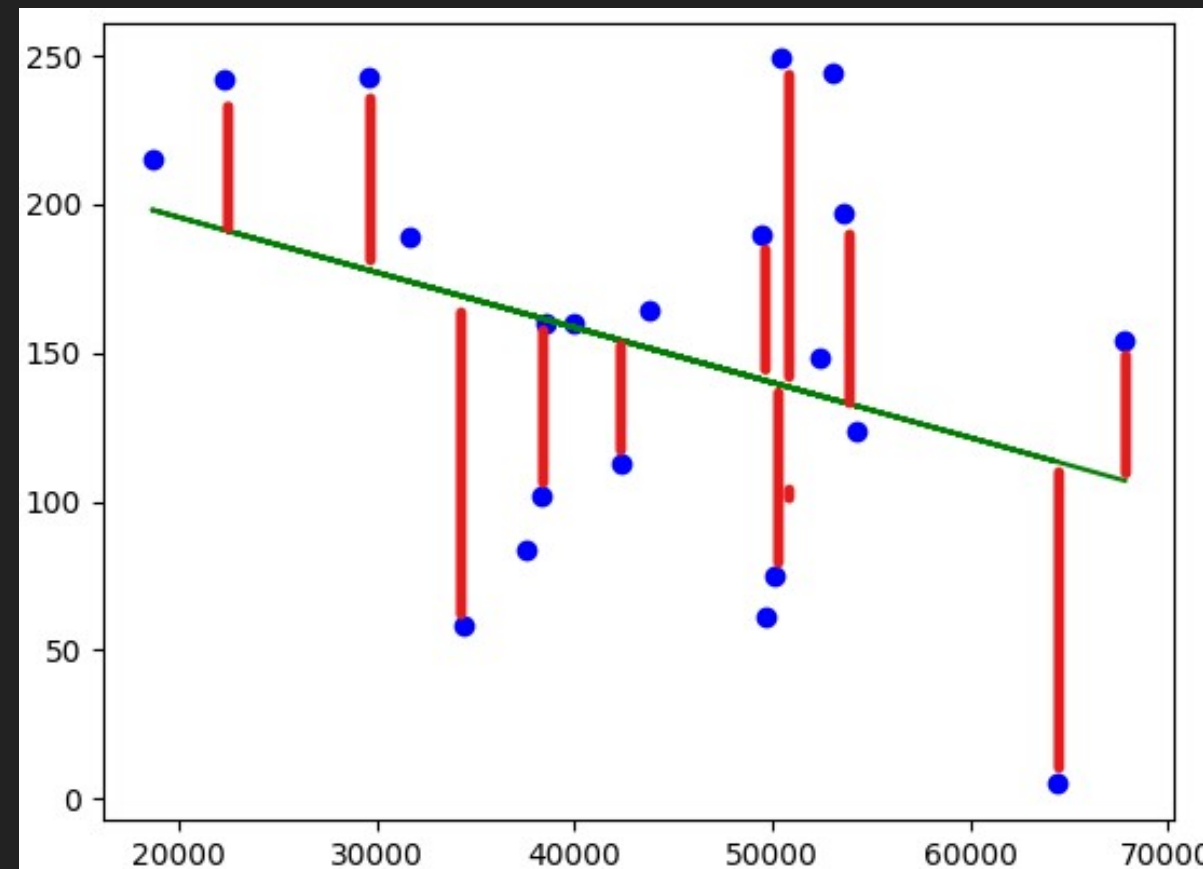
# Error cuadrático medio

El **error cuadrático medio** nos indican como de cerca o alejados estamos entre los valores reales y los estimados.

Contra más bajo mejor.

```
from sklearn.metrics import mean_squared_error
```

```
error_cuadratico_medio = mean_squared_error(y_true=x_prueba, y_pred=prediccion);  
print(f"Error cuadrático medio: {error_cuadratico_medio}");
```



## Error medio absoluto

El **error medio absoluto** es una medida de la diferencia entre dos variables continuas. Contra más bajo mejor.

```
from sklearn.metrics import mean_absolute_error

error_absoluto_medio = mean_absolute_error(y_prueba, prediccion);
print(f"Error absoluto medio: {error_absoluto_medio}");
```

# Coeficiente

El **coeficiente** nos indica la calidad del modelo para replicar los resultados y la proporción de variación de los resultados.

Un **coeficiente** de 1 sería el mejor resultado.

```
import sklearn.metrics as metrics

coeficiente = metrics.r2_score(y_prueba, prediccion);
print(f"Coeficiente: {coeficiente}");
```

Por lo que muchas veces tendremos que tener más datos para el entrenamiento para poder obtener mejores valores.

# ¿Qué es un árbol de decisión?

Los **árboles de decisión** es una representación gráfica de posibles soluciones a una decisión basada en ciertas condiciones.

Los **arbores de decisión** pueden realizar tareas de **clasificación o regresión**.

Este trabajo **no es óptimo cuando tenemos miles o millones de combinaciones**.



# Arbol de decisión

Es uno de los algoritmos más populares gracias a que se puede ver un diagrama de flujo con los resultados.

- ❑ Los árboles de decisión son fáciles de interpretar y visualizar.
- ❑ Puede capturar fácilmente patrones no lineales.
- ❑ Requiere menos preprocesamiento de datos por parte del usuario, por ejemplo, no es necesario normalizar las columnas.
- ❑ Datos sensibles al ruido, puede sobredimensionar los datos ruidosos.
- ❑ La pequeña variación en los datos puede dar lugar a un árbol de decisión diferente.



# Arbol de decisión

```
import matplotlib.pyplot as pyplot
import numpy
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report, confusion_matrix, r2_score
from sklearn.tree import DecisionTreeClassifier

# Obtenemos los datos de prueba
valor_x, valor_y = load_iris(return_X_y=True);

# Entrenamos el modelo
modelo = DecisionTreeClassifier().fit(valor_x, valor_y);

# Realizamos la predicción
prediccion = modelo.predict(valor_x);

# Visualizacion de datos del arbol
tree.plot_tree(modelo, filled=True);
pyplot.show();
```

# Arbol de decisión

```
# Metricas
puntuacion = r2_score(valor_y, prediccion);
reporte = classification_report(valor_y, prediccion);
matriz_confusion = confusion_matrix(valor_y, prediccion);

print(f"Puntuación: {puntuacion}");
print(f"Reporte: {reporte}");

# Visualizacion de la matriz de confusion
pyplot.matshow(matriz_confusion);
pyplot.title("Matriz de confusión");
pyplot.colorbar();
for (x, y), value in numpy.ndenumerate(matriz_confusion):
    pyplot.text(x, y, f"{value:.2f}", va="center", ha="center");
pyplot.show();

# Representacion de la clasificacion
figure, (plot1, plot2) = pyplot.subplots(2);
plot1.title.set_text("Valores clasificados");
plot1.scatter(valor_x[:, 2], valor_x[:, 3], c=valor_y, cmap='viridis');
plot2.title.set_text("Valores calculados");
plot2.scatter(valor_x[:, 2], valor_x[:, 3], c=prediccion, cmap='viridis');
pyplot.show();
```

# Coeficiente

El **coeficiente** nos indica la calidad del modelo para replicar los resultados y la proporción de variación de los resultados.

Un **coeficiente** de 1 sería el mejor resultado.

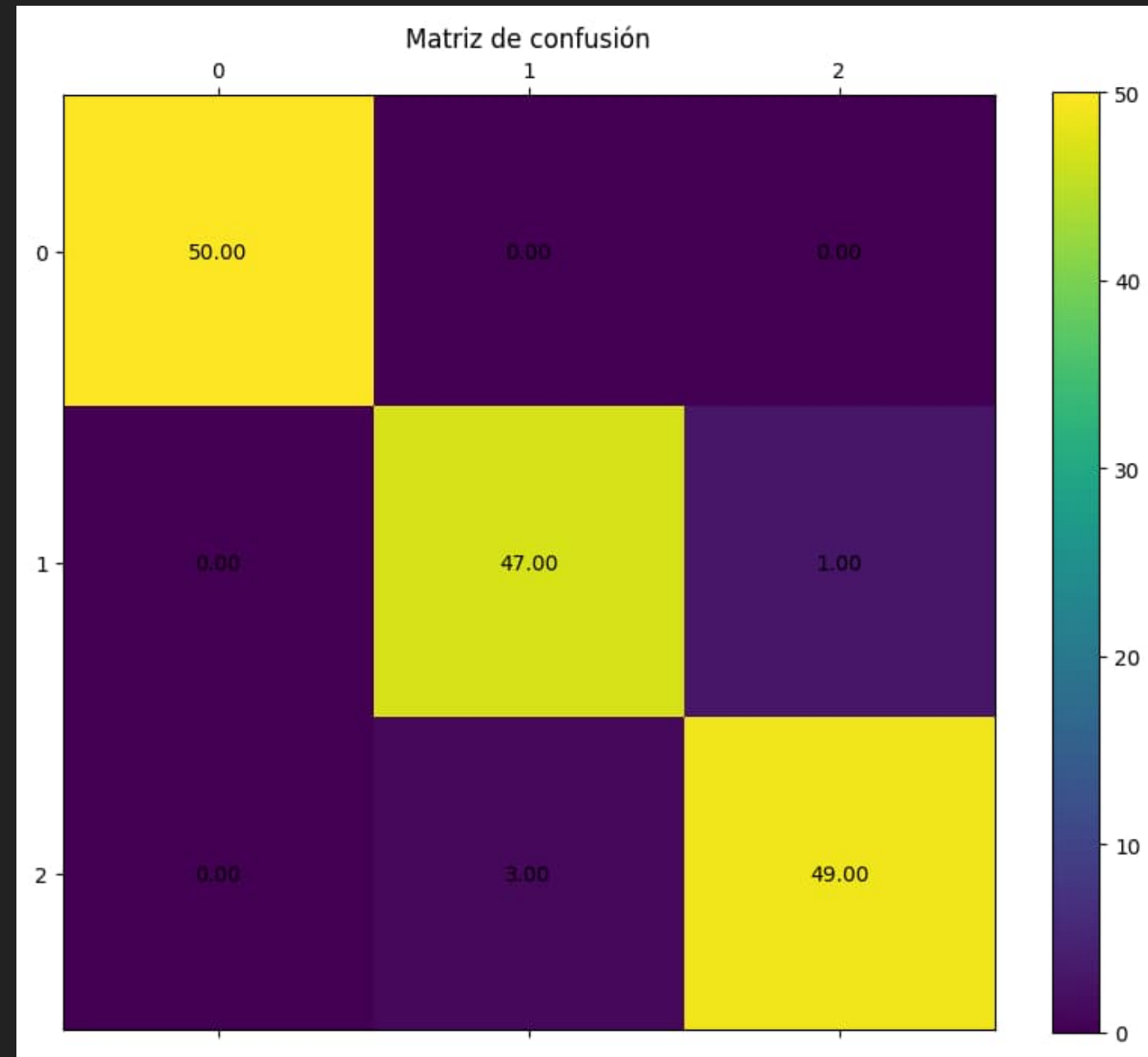
```
import sklearn.metrics as metrics

coeficiente = metrics.r2_score(y_prueba, prediccion);
print(f"Coeficiente: {coeficiente}");
```

Por lo que muchas veces tendremos que tener más datos para el entrenamiento para poder obtener mejores valores.

# Matriz de confusión

Es una herramienta que nos permite conocer el desempeño de un algoritmo.  
La filas siempre son los totales reales y las columnas, los totales predichos.



## Reporte de clasificación

- ❑ La puntuación f1, es la media armónica de precisión y recuperación.
- ❑ Las cifras de cada clase, es la precisión al detectar cada tipo.
- ❑ El soporte es el número de muestras que se encuentran en la clase.
- ❑ La precisión es precisamente, como de efectivo ha sido la clasificación en cada clase.
- ❑ Recall es la exhaustividad del modelo, sobre su capacidad de identificación.

Reporte:	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	0.98	0.94	0.96	50
2	0.94	0.98	0.96	50
accuracy			0.97	150
macro avg	0.97	0.97	0.97	150
weighted avg	0.97	0.97	0.97	150

## K-means

**K-means** es un algoritmo no supervisado de **cluster**. Se suele utilizar cuando tenemos un montón de datos sin etiquetar o clasificar. El objetivo de este algoritmo es **encontrar los grupos entre los datos** crudos.

El algoritmo trabaja iterativamente para asignar a cada punto uno de los grupos con base a sus características.

- ❑ Al ejecutar el algoritmo tendremos los **centroids** de cada grupo, que serán los puntos de cada uno de los grupos para etiquetar a las nuevas muestras.
- ❑ Etiquetas para el conjunto de datos de entrenamiento.

## Datos de entrada

Las **características** que utilizaremos para aplicar el algoritmo deberán ser valores numéricos, continuos en lo posible. En el caso de datos **categoricos** (textos) se puede convertir a número, pero no se recomienda, pues no hay distancia entre los valores.

Además es recomendable que los valores **estén normalizados o mantengan la misma escala**.

En algunos casos funcionan mejor **los datos porcentuales** en vez de los absolutos.

# K-means

```
import matplotlib.pyplot as pyplot
import numpy
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report, confusion_matrix, r2_score

# Obtenemos los datos de prueba
valor_x, valor_y = load_iris(return_X_y=True);

# Mostramos los datos
print(valor_x[:5,:]);
print(valor_x[-5:,:]);

# Entrenamos el modelo
modelo = KMeans(n_clusters=3).fit(valor_x);

# Realizamos la predicción
prediccion = modelo.predict(valor_x);
```



# K-means

```
# Metricas
puntuacion = r2_score(valor_y, prediccion);
reporte = classification_report(valor_y, prediccion);
matriz_confusion = confusion_matrix(valor_y, prediccion);

print(f"Puntuación: {puntuacion}");
print(f"Reporte: \n{reporte}");
print(f"Centroides: \n{modelo.cluster_centers_}");

# Visualizacion de la matriz de confusion
pyplot.matshow(matriz_confusion);
pyplot.title("Matriz de confusión");
pyplot.colorbar();
for (x, y), value in numpy.ndenumerate(matriz_confusion):
    pyplot.text(x, y, f"{value:.2f}", va="center", ha="center");
pyplot.show();

# Representacion de la clasificacion
figure, (plot1, plot2) = pyplot.subplots(2);
plot1.title.set_text("Valores clasificados");
plot1.scatter(valor_x[:, 2], valor_x[:, 3], c=valor_y, cmap='viridis');
plot2.title.set_text("Valores calculados");
plot2.scatter(valor_x[:, 2], valor_x[:, 3], c=prediccion, cmap='viridis');
pyplot.show();
```

# Curva de elbow

Cuando estamos entrenando cualquier modelo de aprendizaje llega un momento en el que seguir entrenando no nos aporta mucho beneficio o el beneficio es nulo.

Por ello nos podemos apoyar en la **curva de elbow** para detectar cuando llegamos al **codo** y evitar seguir trabajando sin obtener mejoría.

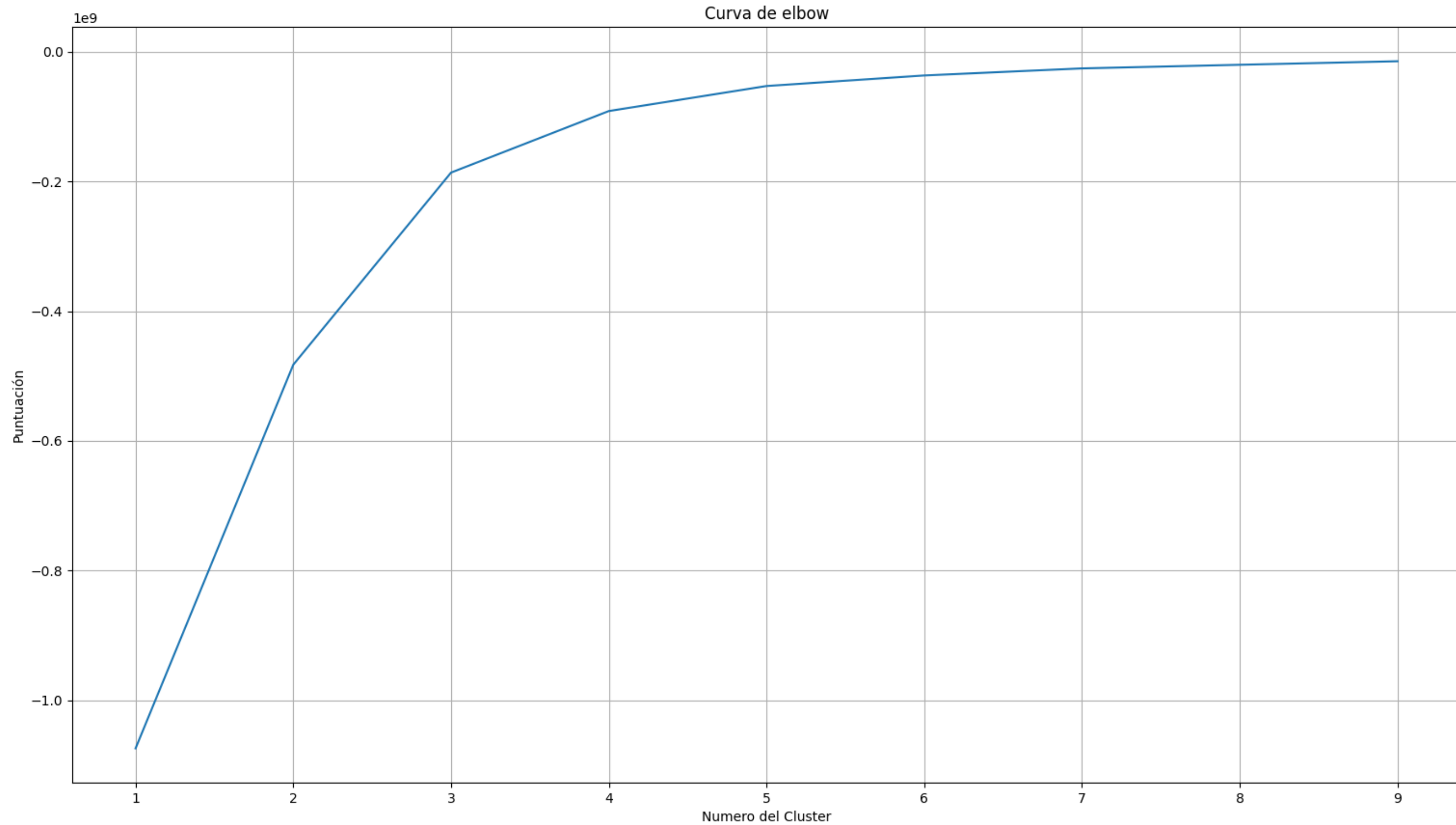
```
numero_cluster = range(1, 10)

kmeans = [KMeans(n_clusters=i) for i in numero_cluster];
score = [kmeans[i].fit(x).score(x) for i in range(len(kmeans))];

pyplot.plot(numero_cluster, score);
pyplot.xlabel("Numero del Cluster");
pyplot.ylabel("Puntuación");
pyplot.title("Curva de elbow");
pyplot.grid();
pyplot.show();

modelo = KMeans(n_clusters=5).fit(x);
```

# Curva de elbow



# Coeficiente

El **coeficiente** nos indica la calidad del modelo para replicar los resultados y la proporción de variación de los resultados.

Un **coeficiente** de 1 sería el mejor resultado.

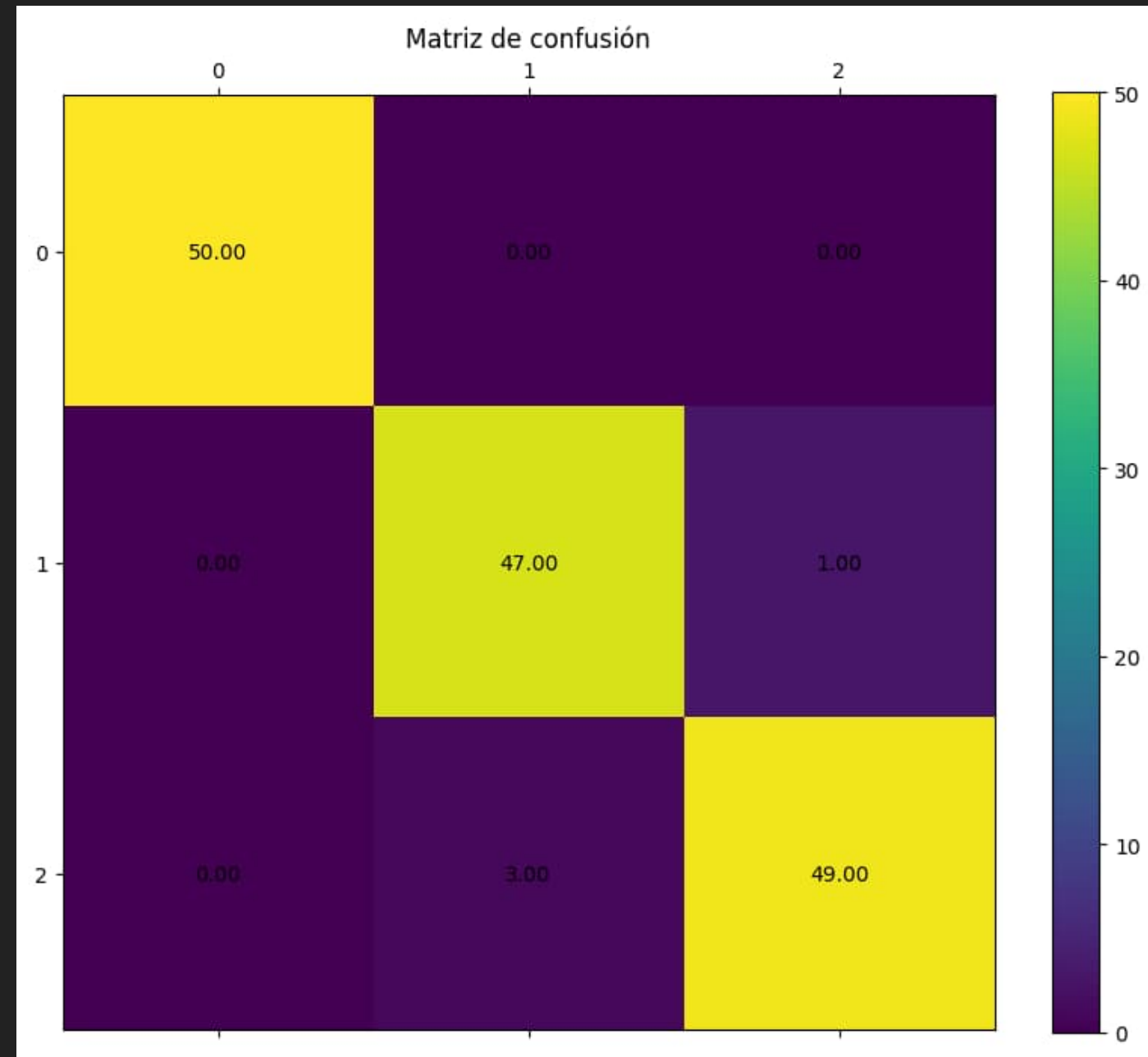
```
import sklearn.metrics as metrics

coeficiente = metrics.r2_score(y_prueba, prediccion);
print(f"Coeficiente: {coeficiente}");
```

Por lo que muchas veces tendremos que tener más datos para el entrenamiento para poder obtener mejores valores.

# Matriz de confusión

Es una herramienta que nos permite conocer el desempeño de un algoritmo.  
La filas siempre son los totales reales y las columnas, los totales predichos.



## Reporte de clasificación

- ❑ La puntuación f1, es la media armónica de precisión y recuperación.
- ❑ Las cifras de cada clase, es la precisión al detectar cada tipo.
- ❑ El soporte es el número de muestras que se encuentran en la clase.
- ❑ La precisión es precisamente, como de efectivo ha sido la clasificación en cada clase.
- ❑ Recall es la exhaustividad del modelo, sobre su capacidad de identificación.

Reporte:		precision	recall	f1-score	support
	0	1.00	1.00	1.00	50
	1	0.98	0.94	0.96	50
	2	0.94	0.98	0.96	50
accuracy				0.97	150
macro avg		0.97	0.97	0.97	150
weighted avg		0.97	0.97	0.97	150