Resumen del capítulo: Vectores y operaciones vectoriales

Creación de vectores

En Python, los conjuntos de datos a menudo se representan como listas. En matemáticas, un conjunto ordenado de datos numéricos es un **vector**, o un **vector aritmético**. Las operaciones que se puedan realizar con números, es decir, sumas, restas y multiplicaciones, también se podrán realizar con vectores. En Python, las operaciones con vectores son cientos de veces más rápidas que las operaciones con listas.

Para trabajar con vectores, volvamos a la librería **NumPy**. Convierte una lista de dos números en un vector:

```
import numpy as np

numbers1 = [2, 3] # Lista de Python
vector1 = np.array(numbers1) # Matriz de NumPy
print(vector1)
```

Los vectores se pueden crear sin una variable temporal:

```
import numpy as np
vector2 = np.array([6, 2])
print(vector2)
```

Los vectores se pueden convertir en listas:

```
numbers2 = list(vector2) # Lista a partir de vector
print(numbers2)
```

La columna de la estructura de DataFrame en pandas se convierte en un vector NumPy utilizando el atributo `values`**:**

```
import pandas as pd

data = pd.DataFrame([1, 7, 3])
print(data[0].values)
```

Utiliza la función `len()` para determinar el tamaño del vector (número de sus elementos):

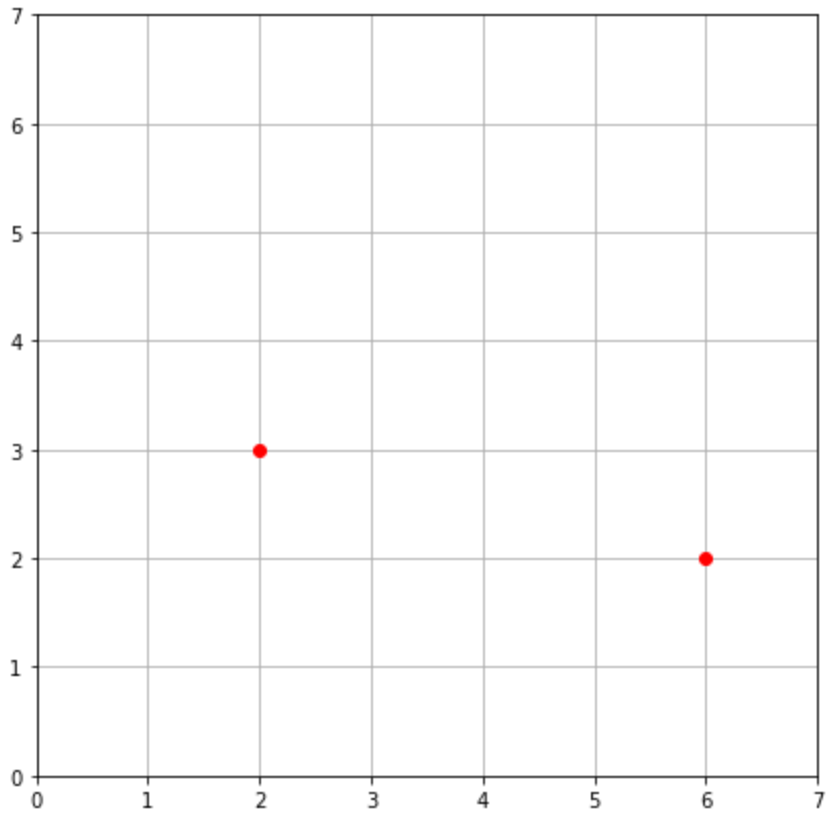
```
print(len(vector2))
```

Presentación de vectores

Tracemos un vector bidimensional. Consta de dos números. El primero es la coordenada en el eje horizontal *x* y el segundo es la coordenada en el eje vertical *y*. El vector se representa mediante un **punto** o una **flecha**, que une el origen y el punto con coordenadas (*x, y*). Cuando queremos indicar los movimientos usamos flechas. Si trabajamos con varios vectores que se encuentran en la misma línea, es mejor utilizar un punto para representar un vector.

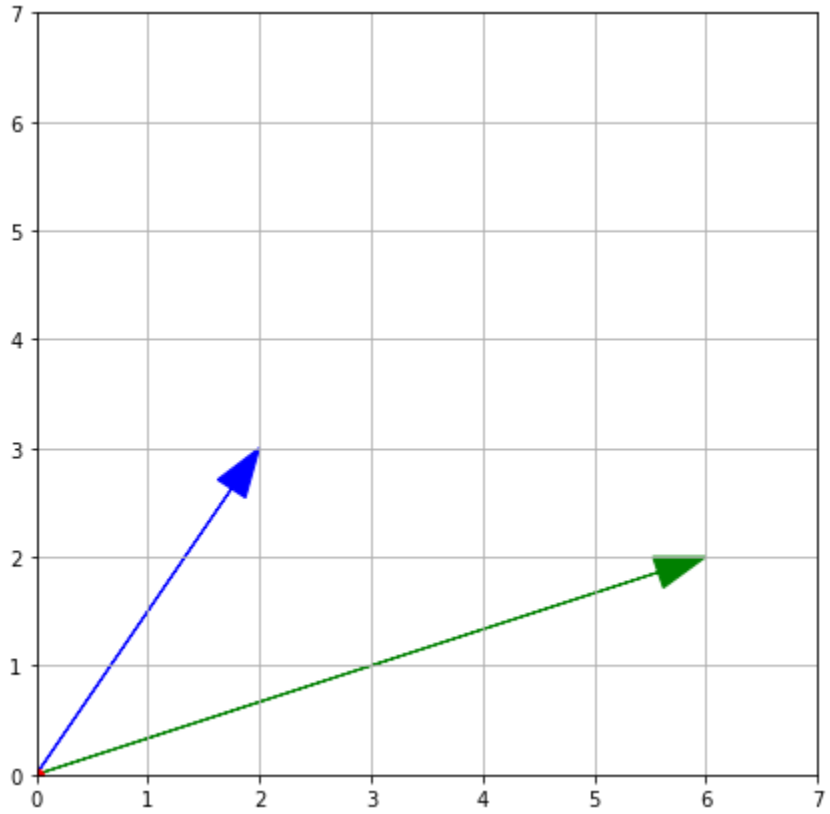
Los elementos vectoriales también se denominan coordenadas.

```
import numpy as np
import matplotlib.pyplot as plt

vector1 = np.array([2, 3])
vector2 = np.array([6, 2])

plt.figure(figsize=(7, 7))
plt.axis([0, 7, 0, 7])
# El argumento 'ro' establece el estilo del gráfico
# 'r' - rojo
# 'o' - círculo
plt.plot([vector1[0], vector2[0]], [vector1[1], vector2[1]], 'ro')
plt.grid(True)
plt.show()
```



Vamos a utilizar flechas para dibujar los mismos vectores. En lugar de `plt.plot()`, llama a `plt.arrow()`.

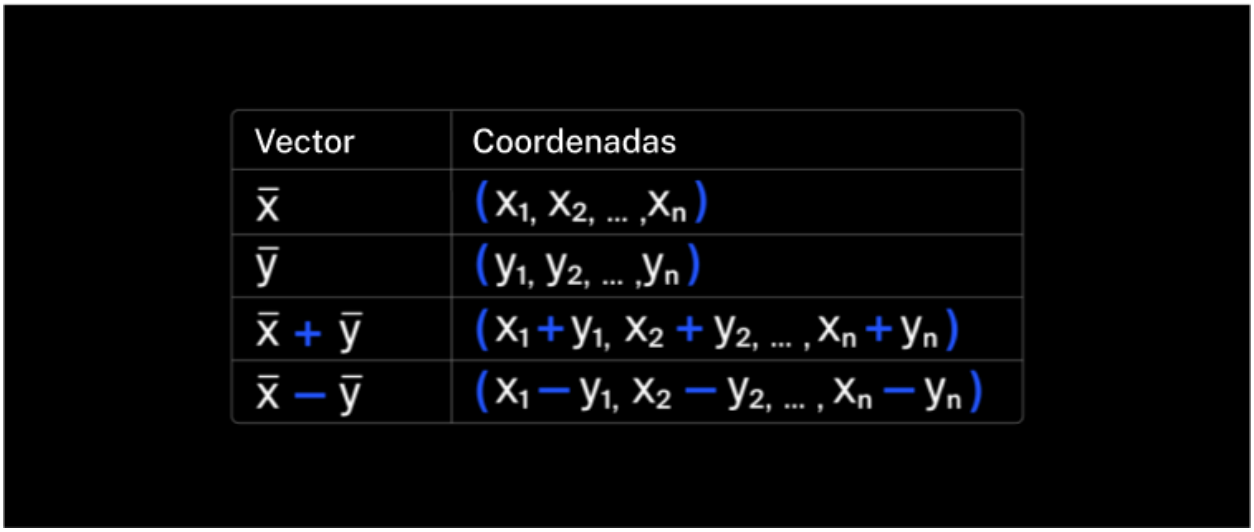
```
import numpy as np
import matplotlib.pyplot as plt

vector1 = np.array([2, 3])
vector2 = np.array([6, 2])

plt.figure(figsize=(7, 7))
plt.axis([0, 7, 0, 7])
plt.arrow(0, 0, vector1[0], vector1[1], head_width=0.3,
          length_includes_head="True", color='b')
plt.arrow(0, 0, vector2[0], vector2[1], head_width=0.3,
          length_includes_head="True", color='g')
plt.plot(0, 0, 'ro')
plt.grid(True)
plt.show()
```



Suma y resta de vectores

Los vectores del mismo tamaño tienen la misma longitud. El resultado de su suma es el vector con cada coordenada igual a la suma de las coordenadas de los vectores sumandos. La primera coordenada del vector resultante es igual a la suma de las primeras coordenadas y la segunda es la suma de las segundas coordenadas. A la hora de restar, cada coordenada del vector resultante es igual a la diferencia entre coordenadas de los vectores dados.

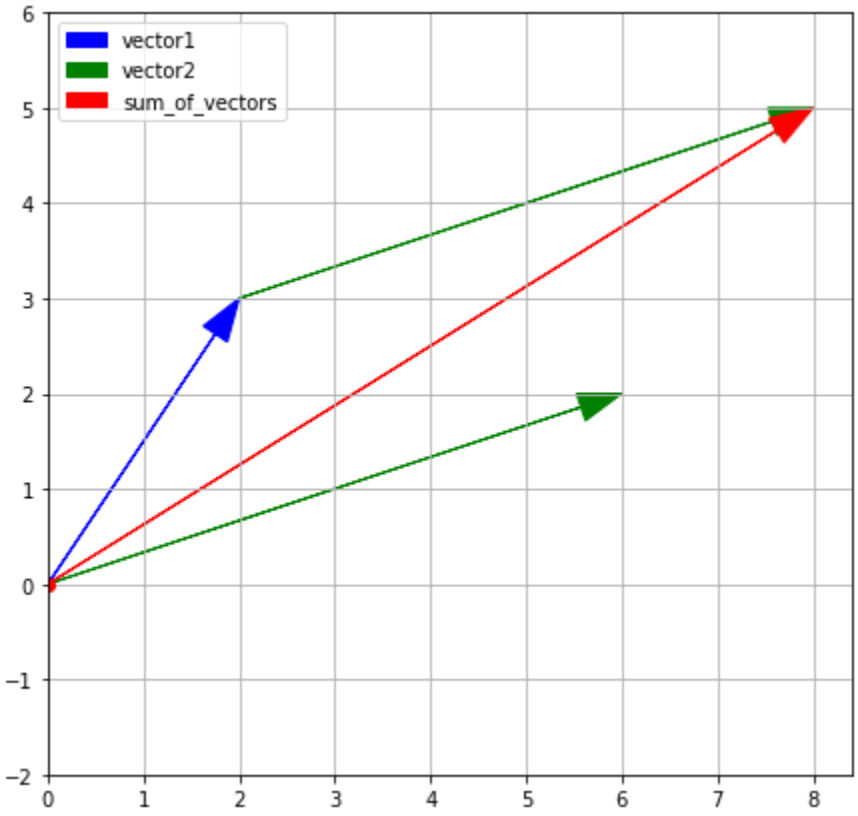
Al sumar o restar vectores, se hace la operación para cada elemento de estos.



Vector	Coordenadas
$\bar{x}$	$(x_1, x_2, \dots, x_n)$
$\bar{y}$	$(y_1, y_2, \dots, y_n)$
$\bar{x} + \bar{y}$	$(x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$
$\bar{x} - \bar{y}$	$(x_1 - y_1, x_2 - y_2, \dots, x_n - y_n)$

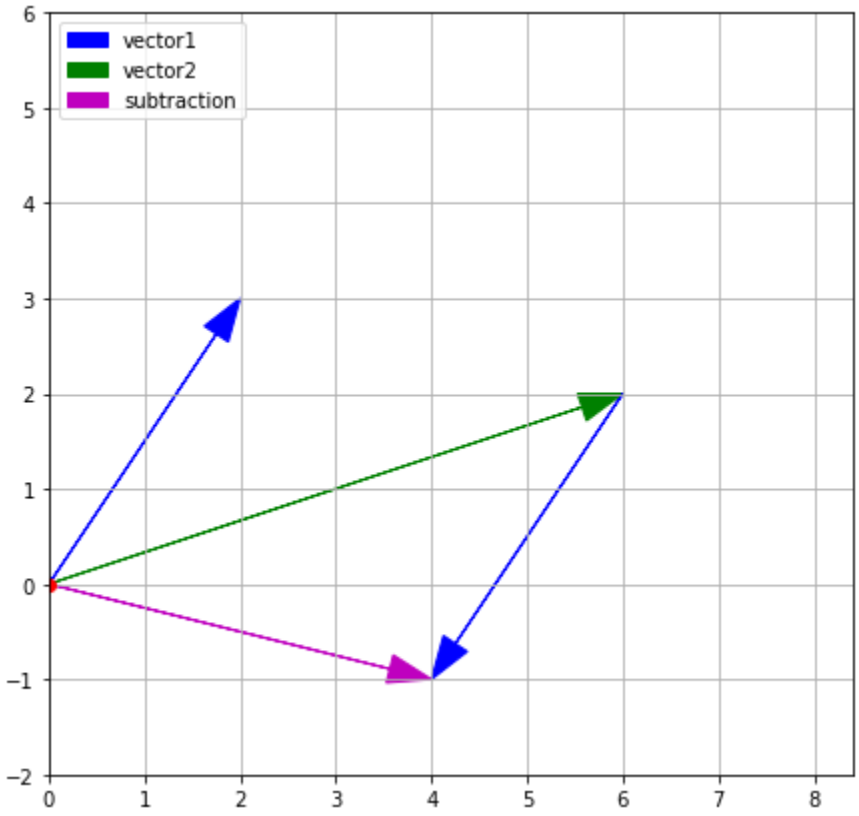
```
import numpy as np

vector1 = np.array([2, 3])
vector2 = np.array([6, 2])
sum_of_vectors = vector1 + vector2
subtraction_of_vectors = vector2 - vector1
```

Si trazamos un vector que sea igual al `vector1` verde en términos de longitud y dirección, desde el final del `vector2` azul, obtendremos el vector rojo (`sum_of_vectors`).



El triángulo obtenido en el gráfico anterior nos da el sentido geométrico de la suma de vectores. Si cada vector es un movimiento en una dirección determinada, la suma de dos vectores es el movimiento a lo largo del primer vector seguido del movimiento a lo largo del segundo.

La diferencia entre dos vectores es un paso, por ejemplo, a lo largo del `vector2`, seguido de un paso en la dirección opuesta al `vector1`.



Multiplicación de un vector por un escalar

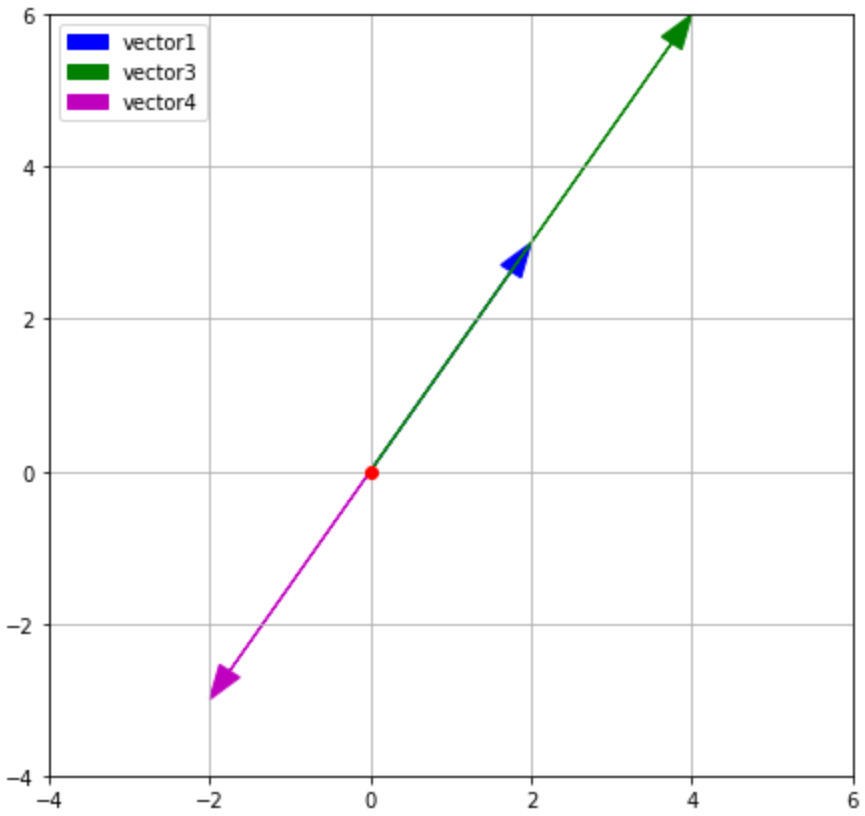
Además de la suma y la resta, los vectores también se pueden multiplicar por escalares. Cada coordenada del vector se multiplica por el mismo número:

Vector	Coordenadas
$\bar{x}$	$(x_1, x_2, \ldots, x_n)$
$k\bar{x}$	$(kx_1, kx_2, \ldots, kx_n)$

En caso de que el número sea negativo, todas las coordenadas también cambiarán de signo.

```
import numpy as np

vector1 = np.array([2, 3])
vector3 = 2 * vector1
vector4 = -1 * vector1
```

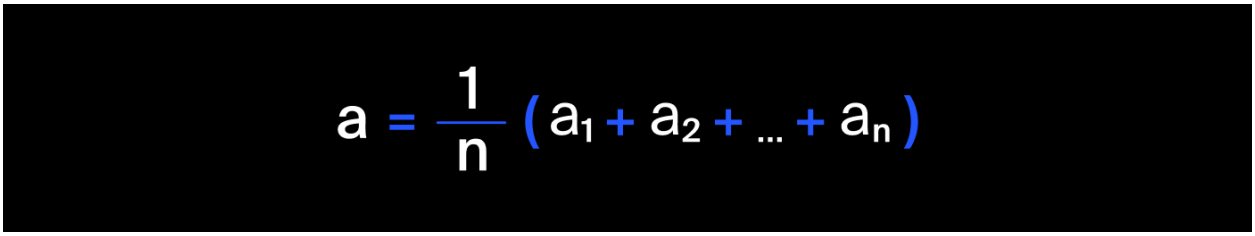
Cuando se multiplican por un número positivo, los vectores mantienen su dirección en el plano, pero las flechas cambian de longitud. Cuando se multiplican por un número negativo, los vectores cambian al sentido opuesto.



Valor medio de los vectores

Si, por ejemplo, los vectores individuales de un conjunto describen a los clientes en función de sus características, entonces el valor medio de los vectores suele describir a un cliente típico o *estadísticamente promedio*. Para el conjunto de vectores $a1$, $a2$…

an (donde *n* es el número total de vectores), el **valor medio** de los vectores es la suma de todos los vectores multiplicada por $1/n$**.**Esto da como resultado un nuevo vector *a:*

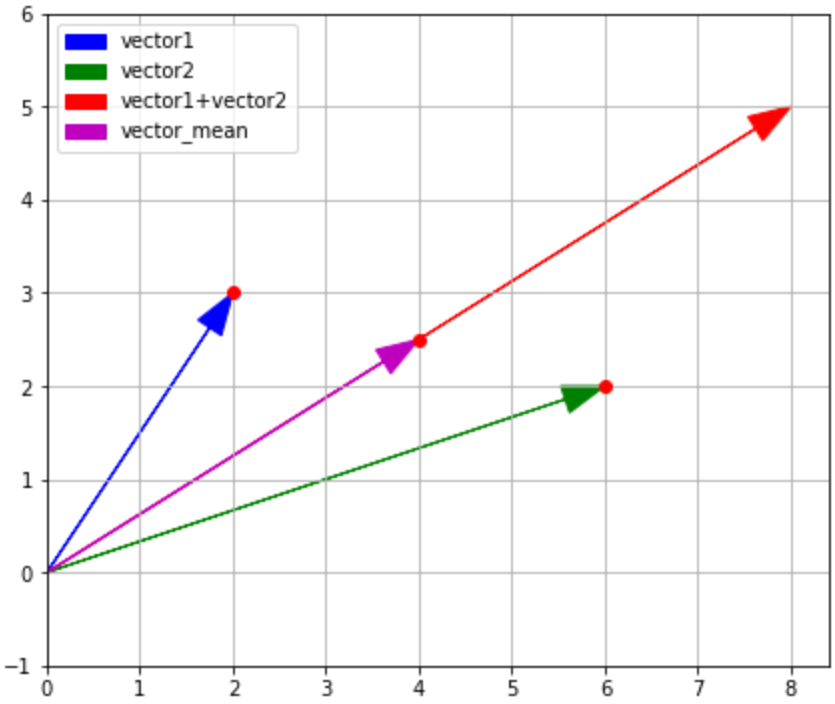


Si el conjunto está formado por un solo vector (n=1), será igual a la media: a=a1. El valor medio de dos vectores es a=0.5(a1+a2). El valor medio de un par de vectores bidimensionales es la mitad del segmento que une a1 y a2.

```
import numpy as np

vector1 = np.array([2, 3])
vector2 = np.array([6, 2])
vector_mean = .5*(vector1+vector2)
print(vector_mean)
```

La primera coordenada del nuevo vector es el valor medio de las primeras coordenadas del `vector1` y del `vector2`, y la segunda coordenada es el valor medio de las segundas coordenadas del `vector1` y del `vector2`.

Así es como dibujamos estos vectores en el plano: trazamos el vector `vector 1+vector2` y luego lo multiplicamos por 0.5.



Funciones vectorizadas

Las herramientas de NumPy nos permiten realizar varias operaciones con vectores. Si usamos la función `np.array()` después de multiplicar y dividir dos arreglos del mismo tamaño, obtendremos un nuevo vector que también tendrá el mismo tamaño:

```
import numpy as np

array1 = np.array([2, -4, 6, -8])
array2 = np.array([1, 2, 3, 4])
array_mult = array1 * array2
array_div = array1 / array2
print("Producto de dos matrices: ", array_mult)
print("Cociente de dos matrices: ", array_div)
```

Si las operaciones aritméticas se realizan sobre una matriz y un solo número, la acción se aplica a cada elemento de la matriz. Y de nuevo, se forma una matriz del mismo tamaño.

Para probar el punto, vamos a realizar sumas, restas y divisiones en una matriz con un escalar:

```
import numpy as np

array2 = np.array([1, 2, 3, 4])
array2_plus_10 = array2 + 10
array2_minus_10 = array2 - 10
array2_div_10 = array2 / 10
print("Suma: ", array2_plus_10)
print("Resta: ", array2_minus_10)
print("Cociente: ", array2_div_10)
```

El mismo principio de "elemento por elemento" se aplica a las matrices cuando tratamos con operaciones matemáticas estándar como las de exponentes o logaritmos.

Vamos a elevar una matriz a la segunda potencia:

```
import numpy as np

numbers_from_0 =  np.array([0, 1, 2, 3, 4])
squares = numbers_from_0**2
print(squares)
```

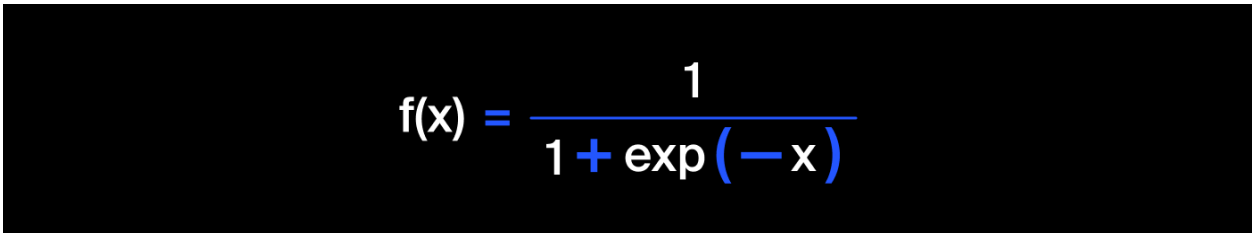
Todo esto también lo podemos hacer con listas a través de bucles, pero las operaciones con vectores en NumPy son mucho más rápidas.

Aquí hay un ejemplo: entre los elementos de la matriz `values`, el valor máximo y mínimo son un par de números, *MIN* y *MAX,* siempre que *MAX>MIN*. Para el análisis se deben convertir los datos. Cada elemento de la matriz debe convertirse a un número en el rango de 0(*MIN*) a 1(*MAX*). Esta es la fórmula de la función `min_max_scale()`:

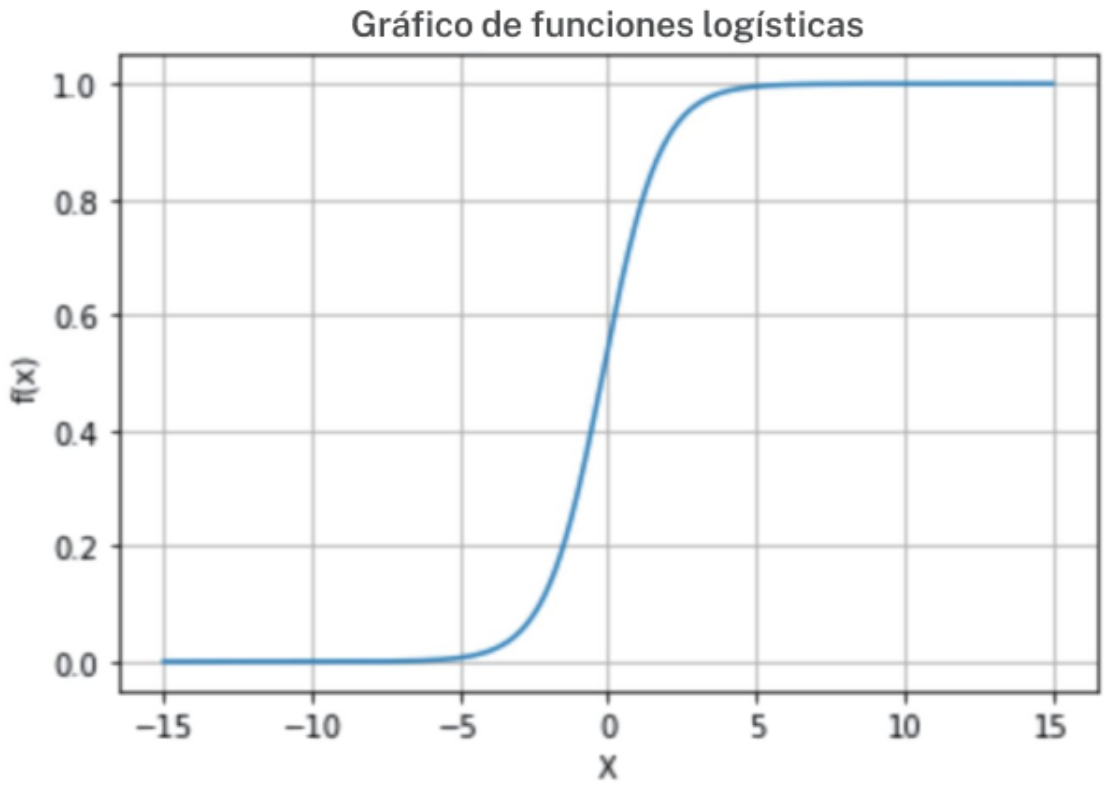
$$f(x) = \frac{x - MIN}{MAX - MIN}$$

Para aplicar esta función a todos los elementos de la matriz `values`, llama a los métodos `max()` y `min()`. Estos encontrarán sus valores máximos y mínimos. Como resultado, obtenemos una matriz de la misma longitud, pero con elementos convertidos:

```
import numpy as np
def min_max_scale(values):
    return (values - min(values)) / (max(values) - min(values))

print(min_max_scale(our_values))
```



donde **exp()** es la función exponente (del lat. *exponere*, "exponer"). Eleva *e,* el número de Euler, a la potencia del argumento. Este número recibió el nombre del gran matemático suizo Leonhard Euler y es aproximadamente igual a 2.718281828.



Realiza la transformación logística:

```
import numpy as np

def logistic_transform(values):
    return 1 / (1 + np.exp(- values))

print(logistic_transform(our_values))
```

Vectorización de métricas

Almacena un conjunto de valores reales en la variable `target` y valores pronosticados en la variable `predictions`. Ambos conjuntos son de tipo `np.array`.

Utiliza las funciones estándar de NumPy para calcular las métricas de evaluación:

- `sum()` (para encontrar la suma de los elementos de una matriz)

- `mean()` (para calcular el valor medio)

Llámalas de la siguiente manera: `<nombre de la matriz>.sum()` y `<nombre de la matriz>.mean()`.

Por ejemplo, esta es la fórmula para calcular el **error cuadrático medio** (*ECM*):

$$ECM = \frac{1}{n} \sum_{i=1}^{n} (target_i - predictions_i)^2$$

donde n es la longitud de cada matriz y $\sum$ es la suma de todas las observaciones de la muestra (i varía de 1 a n). Los elementos ordinales de los vectores *target* (objetivo) y *predictions* (predicciones) se denotan mediante *target_i* y *predictions_i*.

Escribe la fórmula utilizando `sum()`:

```
def mse1(target, predictions):
    n = target.size
    return((target - predictions)**2).sum()/n
```

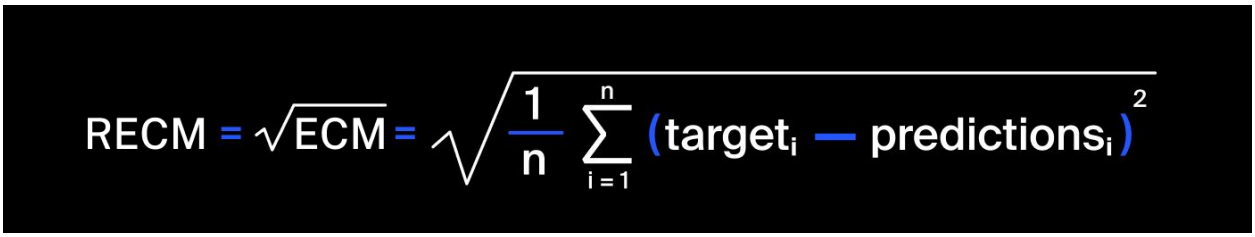
La suma de varios números dividida por su cantidad es la media de dichos números. Vamos a escribir la fórmula de *ECM* utilizando `mean()`:

```
def mse2(target, predictions):
    return((target - predictions)**2).mean()
```

Escribe la función para calcular el *EAM* utilizando `mean()`:


```
import numpy as np

def mae(target, predictions):
    return np.abs((target - predictions)).mean()

print(mae(target, predictions))
```

Las funciones vectorizadas se pueden utilizar para calcular el *RECM*. Aquí tienes la fórmula:




```
import numpy as np

def rmse(target, predictions):
    return (((target-predictions)**2).mean())**0.5

print(rmse(target, predictions))
```