



# TÉCNICAS DE PROGRAMACIÓN

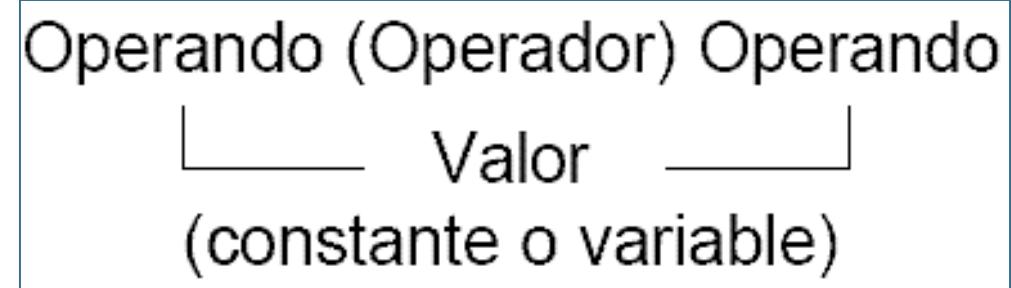
PROF. SIMÓN POLIZZI



# INTRODUCCIÓN AL LENGUAJE PYTHON

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

- Concepto de Operador: es un símbolo especial o palabra reservada que indica una operación específica que debe realizar el intérprete sobre uno o más valores (operando). Los operadores son instrucciones que el intérprete traduce en una acción concreta, es decir, permiten realizar cálculos, comparaciones, asignaciones, operaciones lógicas o manipulaciones de bits.
- Concepto de Operando: es el valor, constante, variable o expresión sobre la cual actúa un operador. Los operandos pueden ser números, cadenas, booleanos, listas, variables, etc.
- Relación: El operador define qué acción se realiza. Los operandos son los datos o elementos sobre los que actúa esa acción específica.



# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

## Operadores Aritméticos

Operador	Descripción	Ejemplo	Resultado
+	Suma	$5 + 3$	8
-	Resta	$5 - 3$	2
*	Multiplicación	$5 * 3$	15
/	División (flotante)	$5 / 2$	2.5
//	División entera (floor)	$5 // 2$	2
%	Módulo (resto)	$5 \% 2$	1
**	Potencia	$2 ** 3$	8

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

## Operadores Relacionales

Operador	Descripción	Ejemplo	Resultado
<code>==</code>	Igual a	<code>5 == 5</code>	True
<code>!=</code>	Distinto de	<code>5 != 3</code>	True
<code>&gt;</code>	Mayor que	<code>5 &gt; 3</code>	True
<code>&lt;</code>	Menor que	<code>5 &lt; 3</code>	False
<code>&gt;=</code>	Mayor o igual que	<code>5 &gt;= 5</code>	True
<code>&lt;=</code>	Menor o igual que	<code>3 &lt;= 5</code>	True

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

## Operadores Lógicos

Operador	Descripción	Ejemplo	Resultado
<b>and</b>	Verdadero si ambas son verdaderas	True and False	False
<b>or</b>	Verdadero si al menos una es verdadera	True or False	True
<b>not</b>	Niega la condición	not True	False

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

## Operadores de Asignación

Operador	Descripción	Ejemplo	Equivalente
=	Asigna valor	x = 5	x = 5
+=	Suma y asigna	x += 3	x = x + 3
-=	Resta y asigna	x -= 3	x = x - 3
*=	Multiplica y asigna	x *= 3	x = x * 3
/=	Divide y asigna	x /= 3	x = x / 3
//=	División entera y asigna	x //= 3	x = x // 3
%=	Módulo y asigna	x %= 3	x = x % 3
**=	Potencia y asigna	x **= 3	x = x ** 3

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

## Operadores de Identidad y Pertenencia

Operador	Descripción	Ejemplo	Resultado
<b>is</b>	Verdadero si apuntan al mismo objeto	x is y	True / False
<b>is not</b>	Verdadero si no apuntan al mismo objeto	x is not y	True / False
Operador	Descripción	Ejemplo	Resultado
<b>in</b>	Verdadero si está en la secuencia	'a' in 'hola'	True
<b>not in</b>	Verdadero si no está en la secuencia	'z' not in 'hola'	True

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

## Definición de Tipos de Operadores

**Aritméticos:** Realizan operaciones matemáticas básicas.

**Comparación (Relacionales):** Comparan valores y devuelven “True” o “False”.

**Lógicos:** Combinan expresiones booleanas.

**Asignación:** Asignan valores a variables, con posibilidad de operar sobre ellas.

**Identidad:** Verifican si dos objetos son el mismo en memoria.

**Pertenencia:** Verifican si un valor está en una secuencia.

**Bit a bit:** Operan a nivel de bits.

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

- **Definición de Estructuras de Control:** En programación, una **estructura de control** es un mecanismo que determina el **flujo de ejecución** de las instrucciones de un programa. En otras palabras, permiten decidir **qué instrucciones se ejecutan, cuántas veces y en qué orden**, en función de condiciones lógicas o repeticiones. Sin estructuras de control, un programa sería una secuencia lineal de instrucciones que se ejecutan de arriba hacia abajo sin posibilidad de tomar decisiones ni repetir procesos. En Python, las estructuras de control están directamente relacionadas con la **indicación de bloques de código** (definidos por indentación).
- **Tipos de Estructuras de Control de Programa:**
  - **Estructuras condicionales (de decisión):** Permiten ejecutar un bloque de código solo si se cumple una condición lógica. Se utilizan cuando el flujo del programa debe **tomar decisiones** (una o múltiples): `if - if...else... - if...elif...else...`
  - **Estructuras repetitivas (bucles o iterativas):** Permiten ejecutar un bloque de código de manera **repetitiva**, hasta que se cumpla o deje de cumplirse una condición: `while` → repite mientras la condición sea verdadera. – `for` → recorre una secuencia (lista, tupla, string, rango, etc.).
  - **Estructuras de control de salto:** Alteran el flujo normal dentro de un bucle o una estructura condicional. `break` → interrumpe la ejecución del bucle actual. `continue` → salta a la siguiente iteración del bucle. `pass` → no realiza ninguna acción (se usa como marcador de lugar).

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

- **Programa secuencial:** es aquel en el que las instrucciones se ejecutan en un **orden lineal y predecible**, una tras otra, sin que existan bifurcaciones, repeticiones o estructuras de control complejas. El flujo de ejecución sigue estrictamente la secuencia en la que fueron escritas las instrucciones. Se caracteriza por ser **simple y directo**, adecuado para algoritmos básicos o tareas que no requieren decisiones condicionales. En un programa **secuencial**, las instrucciones se ejecutan de forma **lineal**.
- **Programa anidado:** un **programa anidado** es aquel que incluye estructuras de control (condicionales o repetitivas)  **contenidas dentro de otras estructuras de control**. Este tipo de programa permite resolver **problemas más complejos**, donde se requiere tomar decisiones jerárquicas, validar múltiples condiciones o realizar iteraciones dentro de otras iteraciones. Se caracteriza por su **organización jerárquica** del flujo de ejecución, lo que incrementa la capacidad de modelar situaciones más avanzadas. En un programa **anidado**, el flujo es **jerárquico**, ya que una estructura de control puede estar dentro de otra.

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

- El acoplamiento en programación se refiere al **grado de dependencia que existe entre diferentes módulos, funciones o componentes de un sistema de software**. Si un módulo **necesita conocer muchos detalles internos** de otro para funcionar, se dice que existe un **alto acoplamiento**. Si un módulo **interactúa con otro únicamente a través de interfaces claras y mínimas dependencias**, se dice que hay un **bajo acoplamiento**. El acoplamiento es una **métrica de diseño de software** que mide el nivel de **interdependencia** entre módulos de un sistema. Un bajo acoplamiento es deseable porque promueve independencia, modularidad y mantenibilidad, mientras que un alto acoplamiento incrementa la complejidad y la propagación de errores.
- **Características**
  - **Alto acoplamiento:**
    - Los módulos están fuertemente interconectados.
    - Un cambio en un módulo puede afectar a muchos otros.
    - Dificulta la **mantenibilidad, escalabilidad y reutilización** del software.
  - **Bajo acoplamiento:**
    - Los módulos son más **independientes**.
    - Un cambio en un módulo no repercute significativamente en los demás.
    - Favorece la **modularidad, reutilización y facilidad de pruebas**.

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

- **Variable Local:** es aquella que se **declara dentro de una función**. Su **ámbito (scope)** está limitado únicamente al bloque donde fue creada. No puede ser utilizada fuera de esa función, ya que **deja de existir** una vez que la función finaliza su ejecución.
- Una **variable local**, se limita al **ámbito interno de la función o bloque donde se declara**, lo cual favorece la **modularidad**, el **encapsulamiento de datos** y la **independencia funcional**. Estas características son deseables en el diseño de software, ya que promueven programas más claros, seguros y fáciles de probar.
- **Variable Global:** es aquella que se **declara en el cuerpo principal del programa o módulo**, fuera de cualquier función o clase. Su **ámbito (scope)** es todo el archivo donde se declara, por lo que puede ser accedida desde cualquier función. Sin embargo, si dentro de una función se desea modificar una variable global, es necesario declararla explícitamente con la palabra reservada *global*.
- Una **variable global** posee un **ámbito de visibilidad amplio**, ya que puede ser accedida desde distintas partes del programa. Sin embargo, su uso excesivo puede incrementar el **grado de acoplamiento** entre módulos, dificultando la mantenibilidad y aumentando la posibilidad de efectos *colaterales* no deseados. Múltiples funciones dependen de un mismo estado compartido de una variable específica y pueden alterarlo sin control.
- Un **efecto colateral no deseado (side effect)** es una **alteración involuntaria (implícita) del estado del programa**, producto de una instrucción o función, que genera comportamientos inesperados y complica el mantenimiento del software. Aparece cuando esa modificación: **No es evidente** en la lógica del programa, **No fue planificada** por el programador, **Genera errores inesperados** en otras partes del sistema.

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

Característica	Variable Global	Variable Local
Dónde se declara	<i>En el cuerpo principal del programa</i>	<i>Dentro de una función</i>
Ámbito (scope)	<i>Todo el archivo o módulo</i>	<i>Solo la función donde se define</i>
Tiempo de vida	<i>Mientras dure la ejecución del programa</i>	<i>Solo mientras la función esté activa</i>
Modificación en función	<i>Requiere global</i>	<i>Libre dentro de la función</i>

# TÉCNICAS DE PROGRAMACIÓN – INTRODUCCIÓN A PYTHON

- El **casteo** o **casting** es el proceso de **conversión** de un dato de un tipo a otro, con el objetivo de que pueda ser interpretado, manipulado o utilizado en un contexto específico dentro de un programa. En Python y en otros lenguajes, puede presentarse en dos formas:
- **Casteo explícito:** realizado de manera intencional por el programador mediante funciones o mecanismos de conversión (ej.: `int("10")` para transformar una cadena en un número entero).
- **Casteo implícito (type coercion):** llevado a cabo automáticamente por el lenguaje cuando dos tipos de datos diferentes participan en una operación (ej.: la suma entre un `int` y un `float` que devuelve un `float`). El programador no necesita hacerlo manualmente: el lenguaje se encarga implícitamente de **ajustar los tipos para evitar pérdida de información**, promoviendo al tipo más general o preciso.
- El **casteo** constituye una herramienta **fundamental** en la **tipificación de datos**, ya que permite controlar y garantizar la coherencia en las operaciones, optimizar el uso de recursos y evitar errores de incompatibilidad entre tipos.



# ESTRUCTURAS DE DATOS

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

- **Técnica de Slicing (rebanado)**: es una operación de subindexado declarativo para manipular fácilmente y eficientemente el acceso sobre **secuencias indexadas** (por ejemplo, listas, tuplas, cadenas, rangos y arrays) que permite obtener una subsecuencia especificando explícitamente:  $\text{secuencia}[\text{inicio} : \text{fin} : \text{paso}]$ . Donde: **inicio**: índice de comienzo (inclusive). **fin**: índice de término (excluyente). **paso**: intervalo de selección (stride).
- Formalmente, el **slicing** construye un nuevo objeto del mismo tipo y con los mismos datos almacenados que la secuencia original (cuando es mutable, como una lista, la copia es shallow copy – “copia superficial”), conteniendo los elementos cuyos índices cumplen:

$i = \text{inicio} + k \cdot \text{paso} \quad \text{con } k \in \mathbb{Z}, i < \text{fin}$

- **Ventajas**: No son necesarios bucles explícitos. El **slicing** está implementado en lenguaje **C** (en CPython), por lo que es más eficiente que recorrer y construir manualmente nuevas secuencias en Python puro. Generalidad sobre estructuras de datos y tipos de datos. Soporte y manejo para índices negativos. Creación de copias superficiales.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

## □ Una estructura de datos estática es aquella en la cual:

- **El tamaño de memoria se define en tiempo de compilación (o al momento de creación).** Antes de ejecutar el programa, cuando el compilador traduce el código fuente a código máquina, **ya debe saber cuánta memoria reservar** para la estructura de datos.
- **El espacio reservado no puede modificarse durante la ejecución del programa.**
- **El acceso a los elementos se realiza mediante índices (subíndices) o posiciones fijas.**
- Son eficientes en cuanto a acceso (tiempo constante  $O(1)$  para acceder por índice – acceso directo), pero poco flexibles porque **no permiten crecer ni decrecer dinámicamente**.

## □ Características principales:

- **Memoria contigua:** los elementos se guardan en posiciones de memoria consecutivas.
- **Acceso directo:** se puede acceder a cualquier elemento de manera inmediata conociendo su índice.
- **Rigidez en tamaño:** el número de elementos totales debe conocerse o definirse al inicio.
- **Eficiencia en tiempo:** rápidos para acceder y recorrer, pero poco adaptables a cambios de tamaño.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

## Concepto técnico de Vector:

- Un **vector informático** es una **estructura de datos estática y lineal** que:
- **Almacena un conjunto finito de elementos** del mismo tipo de datos (enteros, reales, caracteres, etc.).
- **Guarda sus elementos en posiciones de memoria contiguas**, lo que permite un **acceso directo e inmediato** a cada componente mediante un **índice o subíndice**.
- El **tamaño (número de elementos)** se define al momento de su creación y permanece fijo durante toda la ejecución del programa.
- Se considera un **caso particular de arreglo unidimensional**.
- Características principales:
  - **Homogeneidad** → todos los elementos son del mismo tipo.
  - **Acceso por índice** → el primer elemento suele estar en la posición 0 o 1 según el lenguaje.
  - **Estático** → tamaño definido al inicio, no cambia en ejecución.
  - **Eficiencia** → el acceso a cualquier posición es en tiempo constante  $O(1)$  porque se calcula la dirección de memoria como:

$$\text{Dirección}(A[i]) = \text{DirecciónBase}(A) + i \times \text{TamañoDelElemento} \cdot \text{Dirección}(A[0])$$

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

## □ Concepto técnico de Vector:

$$\text{Dirección}(A[i]) = \text{DirecciónBase}(A) + i \times \text{TamañoDelElemento} \text{Dirección}(A[i])$$

- Donde:
  - **DirecciónBase(A)** → la posición de memoria del primer elemento del vector declarado como A.
  - **i** → el índice del elemento (empezando en posición = 0 en la mayoría de lenguajes).
  - **TamañoDelElemento** → cuántos bytes ocupa cada dato almacenado en una posición específica del vector (ej: 4 bytes para un entero). Cada elemento ocupa un cierto número fijo de bytes
- **Ejemplo** → Para encontrar cualquier elemento  $A[i]$ , no hace falta recorrer desde el inicio. **Supongamos:** Un vector de enteros de longitud = 5. Cada entero ocupa 4 bytes. La dirección base es 1000 (posición del primer elemento  $A[0]$ ). Cálculo de direcciones:  $A[0] \rightarrow 1000 + (0 \times 4) = 1000$ ;  $A[1] \rightarrow 1000 + (1 \times 4) = 1004$ ;  $A[2] \rightarrow 1000 + (2 \times 4) = 1008$ ;  $A[3] \rightarrow 1000 + (3 \times 4) = 1012$ ;  $A[4] \rightarrow 1000 + (4 \times 4) = 1016$ .
- El **compilador** sabe que  $A[2]$  está exactamente en la dirección 1008 por la operación aritmética computada anteriormente → Entonces puede saltar directo ahí, sin recorrer  $A[0]$  y  $A[1]$ .
- En resumen: Un **vector** es un **arreglo unidimensional estático, homogéneo y de acceso directo**, usado para **representar colecciones** de datos ordenados.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

## Partes de un vector:

- ✓ **Nombre del vector** → identificador simbólico usado para referirse al arreglo (ej: A).
- ✓ **Dirección base** → la posición de memoria del primer elemento (A[0]).
- ✓ **Tamaño o longitud** → número fijo de elementos que puede almacenar.
- ✓ **Tipo de datos** → todos los elementos son homogéneos (ej: enteros, reales, caracteres, cadenas, etc.). Un tipo de dato por vector.
- ✓ **Índice o subíndice** → entero que indica la posición de cada elemento (normalmente un rango desde 0 hasta n-1).
- ✓ **Elementos almacenados** → los datos en sí, ubicados en posiciones de memoria contiguas.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

## □ Concepto técnico de Matriz:

- Una **matriz** es una **estructura de datos estática y de tipo homogéneo**, organizada generalmente, en dos dimensiones (filas y columnas), donde cada elemento puede identificarse de manera unívoca a través de un **par de índices**  $(i, j)$ , donde fundamentalmente,  $i = \text{row}$  y  $j = \text{column}$ .

## Características principales

- **Homogeneidad:** todos los elementos de la matriz son del mismo tipo de dato.
- **Dimensionalidad:** es una extensión del vector (1D) a 2D o más (matriz bidimensional, tridimensional, etc.).
  - Matriz 2D → tabla de  $m \times n$ . Matriz 3D → colección de tablas.
- **Almacenamiento en memoria:** aunque se la piense en dos dimensiones, en memoria se guarda en una **región o bloque contiguo lineal**, siguiendo un **orden**: **Row-major order**: primero se almacenan todos los elementos de la **primera fila**, luego todos los de la **segunda fila**, y así sucesivamente. **Column-major order**: primero se almacenan todos los elementos de la **primera columna**, luego todos los de la **segunda columna**, y así sucesivamente.
- **Acceso directo (aleatorio):** el elemento en la posición  $(i, j)$  puede calcularse matemáticamente mediante una fórmula de direccionamiento, sin recorrer toda la estructura.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

- **Acceso a elementos de una Matriz:**

- Matriz  $3 \times 4$  (3 filas, 4 columnas):

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Donde:

- Tamaño de cada elemento:  $s=4$  bytes (int type)
- Número de filas:  $m=3$
- Número de columnas:  $n=4$
- Dirección base en memoria:  $\text{DIR}(A[0][0])=1000$
- Queremos **acceder** a  $A[2][1]$  (fila 2, columna 1, valor “10”).

- **Fórmula Row-major order (por filas):**

- **Sustituimos valores:**  $i=2, j=1, n=4, s=4, \text{DIR}(A[0][0])=1000$

- $\text{DIR}(A[2][1]) = 1000 + (2 \times 4 + 1) \times 4 = 1000 + (8 + 1) \times 4 = 1000 + 9 \times 4 = 1000 + 36 \rightarrow \text{DIR}(A[2][1]) = 1036$

$$\text{DIR}(A[i][j]) = \text{DIR}(A[0][0]) + (i \times n + j) \times s$$

- **Fórmula Column-major order (por columnas):**

- **Sustituimos valores:**  $i=2, j=1, m=3, s=4, \text{DIR}(A[0][0])=1000$

- $\text{DIR}(A[2][1]) = 1000 + (1 \times 3 + 2) \times 4 = 1000 + (3 + 2) \times 4 = 1000 + 5 \times 4 = 1000 + 20 \rightarrow \text{DIR}(A[2][1]) = 1020$

$$\text{DIR}(A[i][j]) = \text{DIR}(A[0][0]) + (j \times m + i) \times s$$

- En resumen, la dirección cambia porque la memoria es **lineal (1D)** y la matriz es **2D**. Según la convención de almacenamiento (por filas o por columnas), los índices  $(i, j)$  se transforman en offsets distintos. Un **Offset** es el **desplazamiento** en bytes desde el inicio de una estructura hasta el elemento deseado.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

## □ Concepto técnico de Matriz:

$$A = [a_{ij}] \quad \text{con} \quad 0 \leq i < m, 0 \leq j < n$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix}$$

### Donde:

- $A$  = nombre propio de la matriz.
- $m$  = número total de filas.
- $n$  = número total de columnas.
- $a_{ij}$  = elemento en la fila  $i$ , columna  $j$ . Representan juntos los valores numéricos que indican la posición exacta de un elemento dentro de la matriz. Utilizan indexación 0-based.
- *Base Address*: es la posición en memoria donde comienza la matriz. La dirección de  $A[0][0]$  es el punto de referencia para calcular offsets.
- *dimensión ( $m \times n$ )*: determinan el tamaño lógico de la matriz. Es la cantidad total de celdas o espacios de almacenaje.
- *Data Type*: es el tipo de dato que se guarda en una celda específica.
- *Element Size*: según el tipo, se determina cuántos bytes ocupa cada elemento.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

## □ Concepto técnico de Matriz:

- Sea  $K$  un cuerpo numérico del conjunto de valores pertenecientes a  $N$ , y  $M$  un conjunto de matrices. Una matriz  $A$  es **cuadrada** si  $A \in K^{n \times n} = M_n(K)$  para algún  $n \in N$  (es decir, el número de filas y de columnas coincide:  $m = n$ ); en caso contrario,  $A \in K^{m \times n}$  para algún  $m, n \in N$  con  $m \neq n$ , y se denomina **rectangular o no cuadrada**.
- El **aplanamiento** (en inglés, *flattening*) es una operación que transforma una **matriz bidimensional**  $A_{m \times n}$  en un **vector unidimensional**  $v_{mn \times 1}$ , preservando el orden de los elementos según una convención de almacenamiento (*row-major* o *column-major*). El resultado del aplanamiento es una **estructura lineal de tamaño**  $m \times n$ . Cada elemento  $a_{(i,j)}$  de la matriz pasa a una posición  $k$  del vector, calculada como:  $k = i \times n + j$  para **almacenamiento por filas (row-major)**, o bien,  $k = j \times m + i$  para **almacenamiento por columnas (column-major)**.
- Formalmente:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \xrightarrow{\text{flatten}} v = [a_{11}, a_{12}, \dots, a_{1n}, a_{21}, a_{22}, \dots, a_{mn}]$$

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

- Cálculo para saber cuánto ocupa un Vector y una Matriz en memoria (en tiempo de ejecución):

$$T = N \times s$$

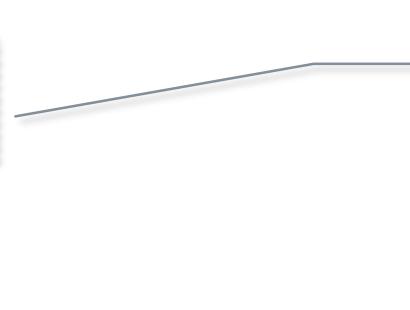


$T$  = tamaño total en Bytes de la estructura de datos (vector estático).

$N$  = cantidad de espacios y celdas de memoria disponibles en total.

$s$  = a partir de su Data Type se fija su Element Size para todos sus elementos.

$$T = (m \times n) \times s$$



$T$  = tamaño total en Bytes de la estructura de datos (matriz estática).

$m$  = cantidad total de filas de la matriz (cada fila es un vector).

$n$  = cantidad total de columnas de la matriz (cada columna es un vector).

$s$  = a partir de su Data Type se fija su Element Size para todos sus elementos.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS ESTÁTICAS

## □ Concepto técnico de Tupla:

- Es una **estructura de datos lineal y ordenada** que permite almacenar un **conjunto de elementos heterogéneos o homogéneos**, accesibles mediante índices enteros, donde:  
$$\text{tupla} = (e_1, e_2, \dots, e_n), \quad e_i \in \text{tipo de dato}, \quad i \in \mathbb{N}, 1 \leq i \leq n$$
- **Características principales → Inmutable:** Una vez creada, no se pueden modificar, agregar ni eliminar elementos. Esto garantiza integridad de datos y permite su uso como clave en diccionarios. Evitan modificaciones accidentales. **Ordenada:** Los elementos mantienen el orden de inserción, y se accede a ellos mediante índices ( $0, 1, \dots, n-1$ ). **Heterogénea:** Puede contener elementos de distintos tipos: enteros, flotantes, caracteres, booleanos, cadenas, listas, incluso otras tuplas. **Acceso por índice:** Los elementos se pueden recuperar usando su índice correspondiente. **Eficiencia en memoria:** Al ser inmutable, las tuplas suelen ocupar menos espacio que las listas equivalentes. **Anidamiento:** Una tupla puede contener otras tuplas (o listas), formando estructuras multidimensionales como matrices. **Iterables:** Se pueden recorrer con bucles for, comprehensions, y algunas funciones nativas. **Soporte para slicing:** Se pueden extraer sub-tuplas mediante la notación de corte. **Compatibilidad:** Pueden combinarse fácilmente con otro tipo de estructuras lógicas, como: listas, diccionarios, arrays y otras tuplas.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS DINÁMICAS

- **Concepto técnico de Lista:** Una **lista** es una **estructura de datos secuencial, dinámica, ordenada y mutable** que permite almacenar un conjunto finito de elementos **heterogéneos** pertenecientes a un **conjunto** determinado, accesibles mediante índices enteros. Constituyen uno de los **tipos de datos fundamentales** y son implementadas internamente como **vectores dinámicos contiguos en memoria**, con capacidad de redimensionamiento automático. Puede considerarse una **colección indexada de referencias a objetos**, donde cada posición (índice) apunta a una dirección de memoria en la que se encuentra el valor correspondiente. El primer elemento tiene índice 0, y el último índice es  $n-1$ , siendo  $n$  la longitud total (size) de la lista. Definición **formal**:

$$L = \langle a_1, a_2, \dots, a_n \rangle, \quad a_i \in D, \quad n \in \mathbb{N}, \quad f_L(i) = a_i, \quad 1 \leq i \leq n$$

- Una **estructura de datos es dinámica** cuando su tamaño puede cambiar durante la ejecución del programa. No tiene un tamaño fijo predefinido. Se realizan operaciones sobre la misma sin necesidad de declarar nuevamente la lista ni crear una nueva estructura. El intérprete puede **reasignar memoria** a medida que se **insertan o eliminan** elementos. Esto se logra mediante **redimensionamiento dinámico**, normalmente con estrategias como: 1) una lista dinámica cuando se crea o se expande, el sistema no reserva memoria solo para los elementos actuales, sino también para algunos futuros. Esto evita que tenga que pedir memoria al sistema operativo cada vez que agregás un elemento — lo cual sería muy costoso en tiempo. 2) cuando la lista se llena completamente (es decir, la cantidad de elementos = capacidad disponible), el sistema duplica o aumenta significativamente la capacidad total. Este comportamiento se llama **estrategia de crecimiento amortizado**. Aunque una expansión individual cuesta mucho, el **promedio de costo por inserción** se mantiene bajo ( $O(1)$  amortizado), ya que las expansiones no ocurren con cada inserción, sino ocasionalmente. 3) cuando la longitud real es mucho menor que la capacidad reservada, se libera parte del espacio o se reasigna un bloque más pequeño. Esto mantiene el uso de memoria **proporcional al tamaño real** de los datos.

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS DINÁMICAS

## □ Concepto técnico de Lista:

- Desde el punto de vista de la **lógica de primer orden**, una lista puede representarse como una relación binaria  $R(i, a_i)$  donde:

$$R = \{(1, a_1), (2, a_2), \dots, (n, a_n)\}$$

- Puede verse como una **relación matemática** entre **índices y valores**.
- Para todo **par de elementos** de la lista, si el índice  $i$  es **menor** que  $j$ , entonces el elemento  $a_i$  aparece **antes** que  $a_j$  en la lista. El **orden** de los **índices** determina el orden de los elementos.
- Para cada índice  $i$  del conjunto  $\{1, \dots, n\}$ , **existe un único elemento**  $a_i$  del dominio  $D$  tal que  $R(i, a_i)$  es **verdadero** (true). Cada posición de la lista tiene **exactamente un valor asociado**.

$$\forall (i, a_i), (j, a_j) \in R : i < j \Rightarrow a_i \text{ precede a } a_j$$

$$\forall i \in \{1, 2, \dots, n\} : \exists! a_i \in D \text{ tal que } R(i, a_i)$$

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS DINÁMICAS

## Concepto técnico de Lista: Características fundamentales

- **Mutabilidad:** Permiten **modificar, agregar o eliminar** elementos en tiempo de ejecución sin necesidad de crear un nuevo objeto. Esta propiedad las hace ideales para representar colecciones de datos que **varían** en tamaño o contenido.
- **Orden y acceso por índice:** Los elementos mantienen el **orden de inserción**, y pueden ser accedidos, modificados o recorridos mediante **índices enteros**.
- **Heterogeneidad:** Pueden contener objetos de **tipos de datos distintos** en una misma estructura: enteros, flotantes, cadenas, booleanos, listas anidadas, etc.
- **Redimensionamiento dinámico:** Las listas se **expanden o contraen automáticamente** cuando se insertan o eliminan elementos. Este comportamiento se logra gracias a una **gestión interna de capacidad y reserva de memoria**.
- **Iterabilidad y compatibilidad funcional:** Son **iterables**, por lo que pueden recorrerse con **bucles normales, comprensiones** (*list comprehensions*), y distintos tipos de funciones nativas del lenguaje.
- **Anidamiento y estructuras complejas:** Pueden contener otras listas almacenadas, formando **estructuras multidimensionales**, como matrices (*tablas*). También, es compatible con otros tipos de estructuras de datos, como pueden ser: arrays, tuplas y diccionarios.
- **Eficiencia en tiempo de acceso:** El acceso por índice tiene una **complejidad  $O(1)$  (tiempo constante)**, ya que las posiciones están mapeadas directamente en memoria. Sin embargo, operaciones como inserciones o eliminaciones en el medio de la lista pueden implicar **reajustes y desplazamientos** de elementos, lo que eleva su complejidad a  **$O(n)$** .
- **Amplitud de métodos integrados:** Al ser una estructura de datos pero a su vez, tratada internamente como un objeto, se pueden invocar y utilizar distintos métodos (funciones internas al objeto) para manipular y realizar de forma más rápida y accesible varios tipos de operaciones (sobre la estructura).

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS DINÁMICAS

□ **Concepto técnico de Diccionario:** Un **diccionario** (también llamado **mapa**, *mapping*, o *associative array*) es una **estructura de datos abstracta** que almacena **pares de asociación** del tipo:  $(k, v)$  donde  $k \in K$  es una clave única y  $v \in V$  es el valor asociado.

- Formalmente, un diccionario implementa una **función parcial**:  $D : K \rightarrow V$
- tal que:  $\forall k_1, k_2 \in K, D(k_1) = D(k_2) \Rightarrow k_1 = k_2$  **no existen dos claves iguales**; cada clave identifica de forma **única** a su valor asociado.
- La fórmula indica que **no pueden existir dos claves distintas que apunten al mismo valor mediante la función  $D$** . En otras palabras,  $D$  (el diccionario) se comporta como una **función inyectiva** respecto de sus claves: cada clave identifica **únicamente** a su valor asociado. Matemáticamente la definición más estricta asegura **unicidad de clave → valor**, no necesariamente **unicidad de valor → clave**.
- Podemos agregar que:  $\forall k \in K, D(k)$  está definida solo si  $k \in \text{dom}(D)$   $\text{dom}(D) = \{k \in K \mid \exists v \in V, (k, v) \in D\}$

# TÉCNICAS DE PROGRAMACIÓN – ESTRUCTURAS DINÁMICAS

## Concepto técnico de Diccionario: Características fundamentales

- **Asociación clave–valor:** Cada elemento del diccionario se representa como un **par (clave, valor)**. La **clave** sirve como identificador único, y el **valor** es el dato asociado.
- **Unicidad de claves:** No pueden existir dos elementos con la misma clave. Garantiza **integridad y acceso inequívoco** a los valores.
- **Acceso eficiente por clave:** Permite recuperar, modificar o eliminar un valor usando directamente su clave, sin necesidad de recorrer todos los elementos. En implementaciones típicas (tablas hash), el acceso promedio es **O(1)**.
- **Mutabilidad / dinamismo:** Los diccionarios permiten **insertar, eliminar y actualizar** pares de manera dinámica en tiempo de ejecución. Su tamaño no está fijado en el momento de creación.
- **No indexado por posición:** A diferencia de listas o arreglos, no se accede a los elementos mediante índices numéricos, sino mediante **claves arbitrarias**. Esto permite usar cadenas, números u otros objetos hashables como identificadores.
- **Flexibilidad de tipos de datos:** Las claves deben ser **inmutables y únicas** (enteros, cadenas, tuplas, etc.). Los valores pueden ser de **cualquier tipo**, incluyendo listas, tuplas, otros diccionarios u objetos complejos.
- **Possible desorden:** En muchas implementaciones, el orden de los elementos **no está garantizado**; el diccionario se organiza para maximizar eficiencia. El orden de inserción se preserva, pero esto es una **propiedad de implementación**, no un requisito matemático.
- **Implementación eficiente en memoria:** Las implementaciones típicas usan **tablas hash** o **árboles balanceados**, optimizando el acceso, la búsqueda y la inserción. Esto permite manejar grandes volúmenes de datos con costos computacionales predecibles.
- **En resumen, un diccionario es una estructura de datos dinámica, no indexada por posición, basada en pares clave–valor, donde cada clave es única, el acceso es directo y eficiente, y los valores pueden ser de cualquier tipo.**



# INTRODUCCIÓN AL MACHINE LEARNING

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

## Conceptos Básicos de ML:

- **Datos:** son la materia prima del *Machine Learning*. Son observaciones o registros que describen situaciones, comportamientos o características del mundo real. Cada dato se compone de **atributos o variables** (también llamados *features*).
- **Conjunto de Datos (Dataset):** es una **colección estructurada de observaciones, registros o instancias** que representan ejemplos del fenómeno que un modelo de *ML* intenta aprender. Cada observación se compone de un conjunto de **variables o atributos (features)** que describen sus características, y opcionalmente de una **etiqueta o variable objetivo (target)** que indica el resultado asociado a esa observación específica.
- **Variable Dependiente e Independiente: Variables Independientes:** Son los factores o características más **relevantes** que explican o influyen en el resultado o el fenómeno estudiado. **Variable Dependiente:** Es la **salida (output)** o el valor que queremos **predecir o explicar**. El **modelo** busca una función matemática que **relacione** ambos tipos de variables.
- **Modelo:** es una **representación matemática, estadística o computacional** que **aprende** una **función  $f$**  capaz de **aproximar la relación existente entre un conjunto de variables de entrada y una variable de salida** a partir de los datos observados. **Identifica** y modela **patrones y relaciones** entre las variables con el fin de realizar **predicciones y clasificaciones** sobre **nuevos datos**.
- **Entrenamiento (Training):** es la fase donde el modelo *aprende los patrones* de los datos de **entrenamiento**. Durante este proceso, el algoritmo ajusta () **parámetros internos** (pesos, coeficientes, etc.) para **reducir** los errores entre sus predicciones y los valores reales.

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

## Conceptos Básicos de ML:

- **Predicción:** una vez entrenado, el modelo puede recibir **nuevos datos** (que nunca ha visto) y generar **una salida o respuesta estimada para llevar a cabo la predicción de un valor de tipo numérico continuo**. Ese proceso se llama **predicción o inferencia**. El modelo no "sabe" la respuesta correcta, pero usa lo que aprendió para estimarla.
- **Clasificación:** consiste en **predecir (realizar una clasificación)** **una categoría o clase** a la que pertenece un determinado dato, basándose en ejemplos anteriores. El modelo **aprende a partir de datos históricos** (que ya están etiquetados con su clase correspondiente) y luego puede **asignar una etiqueta (tipo de dato categórico)** a **nuevos datos** que nunca ha visto antes.
- **Error o Pérdida (Loss/Error):** mide qué tan diferente es la predicción del modelo respecto al valor real. Durante el entrenamiento, el algoritmo intenta **minimizar una función de pérdida (loss function)**. El **objetivo** del algoritmo es encontrar los parámetros que **minimicen la suma o porcentaje total de errores**.
- **Aprendizaje Supervisado:** aquí los datos incluyen **entradas (X)** y **salidas conocidas (Y)**, es decir, el modelo sabe cuál es la respuesta correcta durante el entrenamiento. El **objetivo** es que el modelo aprenda la relación entre X y Y, para luego predecir Y cuando solo tenga X.

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

## Conceptos Básicos de ML:

- **Aprendizaje No Supervisado:** en este tipo de aprendizaje, los datos **no tienen etiquetas ni salidas conocidas**. El modelo no sabe la respuesta “correcta”. El objetivo principal de este enfoque es **identificar patrones, dependencias, estructuras subyacentes o distribuciones ocultas** dentro de los datos, sin intervención explícita ni conocimiento previo de las categorías o resultados esperados.
- **Variable cualitativa:** es aquella que **describe una característica o categoría, no numérica**. Sirve para **clasificar o identificar** a los datos según atributos o cualidades. Representa **categorías o clases**. Debe convertirse en números usando técnicas como “One-Hot Encoding” o “Label Encoding”.
- **Variable cuantitativa:** es aquella que **se expresa con números y permite realizar operaciones matemáticas**. Representa una **cantidad medible**. Representa **medidas o cantidades reales**. Se usa directamente en el modelo (*sin transformación/normalización*).

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

## Conceptos Básicos de ML:

- **Tipos de variables → Variables categóricas:** Representan **categorías o grupos**. No tiene sentido realizar operaciones matemáticas sobre las mismas. **Subtipos: Nominales:** No tienen orden intrínseco. Solo identifican la categoría a la que pertenece un objeto o individuo. *Ejemplo:* color de ojos, grupo sanguíneo. **Ordinales:** Tienen un **orden lógico**, pero la distancia entre categorías no necesariamente es igual. *Ejemplo:* nivel de dolor (Leve, Moderado, Severo). **Variables numéricas (cuantitativas):** Representan **valores numéricos** sobre los cuales se pueden realizar operaciones matemáticas. **Subtipos: Continuas:** Pueden tomar **cualquier valor dentro de un intervalo**, incluyendo decimales. *Ejemplo:* presión arterial, nivel de glucosa, peso. **Discretas:** Solo toman **valores enteros específicos**, generalmente contables. *Ejemplo:* número de visitas al médico, número de hospitalizaciones, etc.
- **Sobreentrenamiento (Overfitting):** ocurre cuando un modelo de aprendizaje supervisado **aprende con excesiva fidelidad las particularidades del conjunto de entrenamiento**, incluyendo el **ruido estadístico y las fluctuaciones aleatorias**, en lugar de aprender los **patrones subyacentes de la distribución de los datos**. Esto provoca que la función ajustada  $f^*(x)$  tenga un **error de generalización elevado** sobre datos no vistos, aunque el error de entrenamiento sea muy bajo. **Causas principales:** **Modelo demasiado complejo** para la cantidad de datos disponible, **Cantidad insuficiente de datos de entrenamiento** en relación con la dimensionalidad del problema, **Exceso de variables irrelevantes o ruido** sin regularización.
- **Subentrenamiento (Underfitting):** El subentrenamiento ocurre cuando el modelo **no captura adecuadamente la estructura de los datos**, es decir, la función aprendida  $f^*(x)$  **no se ajusta a los patrones reales presentes en la distribución de los datos**. Esto provoca **alto error tanto en entrenamiento como en prueba**, reflejando **alta sesgo** y baja complejidad del modelo. **Causas principales:** Modelo demasiado simple para la complejidad de la relación entre variables, Falta de características relevantes (variables predictoras insuficientes), Subajuste de hiperparámetros, regularización excesiva o poco entrenamiento.

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

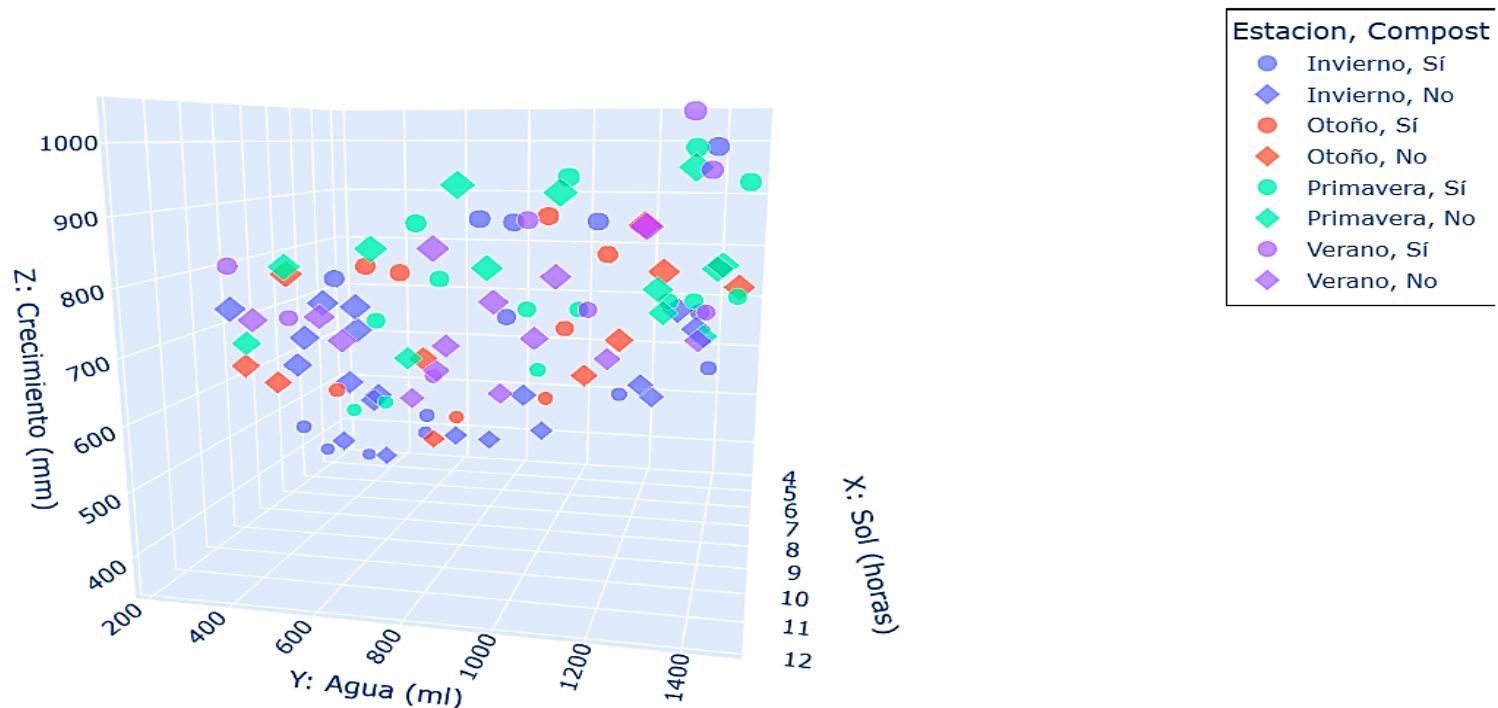
- **Regresión Lineal (Multiple Linear Regression)** ⇒ Su objetivo es **modelar la relación matemática entre una variable dependiente continua (Y) y una o más variables independientes (X)** mediante una **función lineal** para posteriormente llevar a cabo **estimaciones** sobre **nuevos datos** una vez encontrados y **calculados** los **coeficientes** correctamente.
- Formalmente, buscamos una función:
- donde:
$$f(x) = \hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$
- $\hat{y}$ : es la estimación numérica que el modelo produce para la variable dependiente Y a partir de los valores de entrada X. Representa la respuesta esperada o estimada del modelo, no el valor real observado.
- $\beta_0$ : término independiente o intercepto (valor de Y cuando todas las  $X_i = 0$ ). Geométricamente, representa el punto donde la recta (o el plano, en modelos multivariados) corta el eje Y.
- $\beta_i$ : coeficientes o parámetros del modelo (indican cuánto cambia Y cuando  $X_i$  cambia una unidad). Cada  $\beta_i$  cuantifica la influencia o peso de la variable independiente  $X_i$  sobre la variable dependiente Y. Matemáticamente, indica el cambio promedio en Y cuando  $X_i$  aumenta una unidad, manteniendo las demás variables constantes.
- $x_i$ : variables independientes o predictoras. Son las características, factores o entradas que el modelo utiliza para explicar o predecir la variable dependiente. Cada observación en el conjunto de datos tiene un valor para cada  $x_i$ .
- $p$ : número total de variables independientes incluidas en el modelo. Determina la dimensión del espacio de características (es decir, el número de ejes en los que se representa el modelo). Si el modelo usa una sola variable ( $p = 1$ ), la regresión se representa como una línea. Si usa dos o más ( $p \geq 2$ ), se representa como un plano o hiperplano en un espacio de múltiples dimensiones.

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

- **Ejemplo práctico →** Se presenta un **problema de regresión lineal múltiple**: predecir el **crecimiento** de la planta de “*albahaca*” en función de varias variables: **Sol** (“horas de luz por día (hs./d)”, discreta), **Agua** (“cantidad de riego (mililitros: ml.)”, continuo), **Estación del año** (“*invierno*”, “*primavera*”, “*verano*”, “*otoño*”, categórica), y **Compost** (“sí” / “no”, categórica). Se desea **estimar** el valor del crecimiento de dicha planta en *milímetros (mm.)*.
- **Observación:** para un conjunto finito de variables independientes donde  $p \geq 2$ , se pueden utilizar algunas técnicas para representar el gráfico correspondiente, por ejemplo en este caso, 5 dimensiones en 3D. **Aplicación → Gráfico 3D** para las variables continuas: {Eje X: “*Sol*”, Eje Y: “*Agua*”, Eje Z: “*Crecimiento predicho*”} | Para las variables categóricas utilizamos el gráfico anterior pero modificando sus puntos específicos aplicando: {**Color del punto**: “*Estación*”, **Forma de los puntos**: “*Compost*”}.
- **Objetivo → Estimar el valor numérico (en mm.) del crecimiento** de la albahaca para cualquier combinación de variables que se den como input. Por ejemplo: “Si la planta recibe 8 hs. de sol al día en promedio, 800 ml. de agua (semanalmente), estamos en otoño y tiene compost, ¿cuánto crecerá en mm.?”. **Cuantificar la influencia de cada variable** sobre el crecimiento: Saber cuánto aporta el sol, el agua, la estación o el compost al crecimiento. Esto se logra mediante el **intercepto  $\beta_0$**  y los **coeficientes  $\beta_1, \beta_2, \beta_3, \beta_4$**  del modelo de regresión. **Identificar patrones y relaciones**: Por ejemplo, detectar si el crecimiento es mayor en primavera que en invierno, o si el compost tiene un efecto significativo. **Hacer predicciones para planificar el cultivo**: Optimizar riego, luz y uso de compost según la estación para maximizar el crecimiento de la planta.

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

Crecimiento de Albahaca según Sol, Agua, Estación y Compost



# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

- **Regresión Logística (Multiple Logistic Regression)** → La regresión logística múltiple es un **modelo estadístico supervisado** utilizado para **predecir la probabilidad de ocurrencia de un evento binario o categórico** ( $Y=0$  o  $Y=1$ ) en función de **varias variables independientes o predictoras** ( $X_1, X_2, \dots, X_n$ ), que pueden ser continuas o categóricas.
- La relación entre las variables predictoras y la probabilidad de éxito se modela mediante la **función logística o sigmoide**:

$$P(Y = 1 | X_1, X_2, \dots, X_n) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n)}}$$

■ donde:

$$P(Y = 1 | X_1, \dots, X_n) = \frac{1}{1 + e^{-(\beta_0 + \sum_{i=1}^n \beta_i X_i)}}$$

- $P(Y=1 | X)$  = representa la probabilidad de que ocurra el evento de interés dado un conjunto de variables independientes ( $X_1, X_2, \dots, X_n$ ). Se expresa como un número entre 0 y 1.
- $\beta_0$  = intercepto o término independiente. Es el valor del logaritmo de las probabilidades cuando todas las variables  $X_i$  son cero. Es la base de referencia del modelo. Representa la probabilidad inicial o base antes de considerar los efectos de las variables independientes.
- $\beta_i$  = coeficientes de cada variable. Cada  $\beta_i$  cuantifica el efecto de la variable  $X_i$  sobre el logaritmo de las probabilidades (log-odds). Mide cómo cambia la probabilidad del evento cuando  $X_i$  aumenta en 1 unidad, manteniendo las demás variables constantes.
- $e$  = es la base de los logaritmos naturales ( $\approx 2.718$ ). Se usa en la función logística para transformar los log-odds lineales en probabilidades interpretables entre 0 y 1. Gracias a este término, la función sigmoide siempre produce un valor adecuado para probabilidades. Determina la curvatura de la función.

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

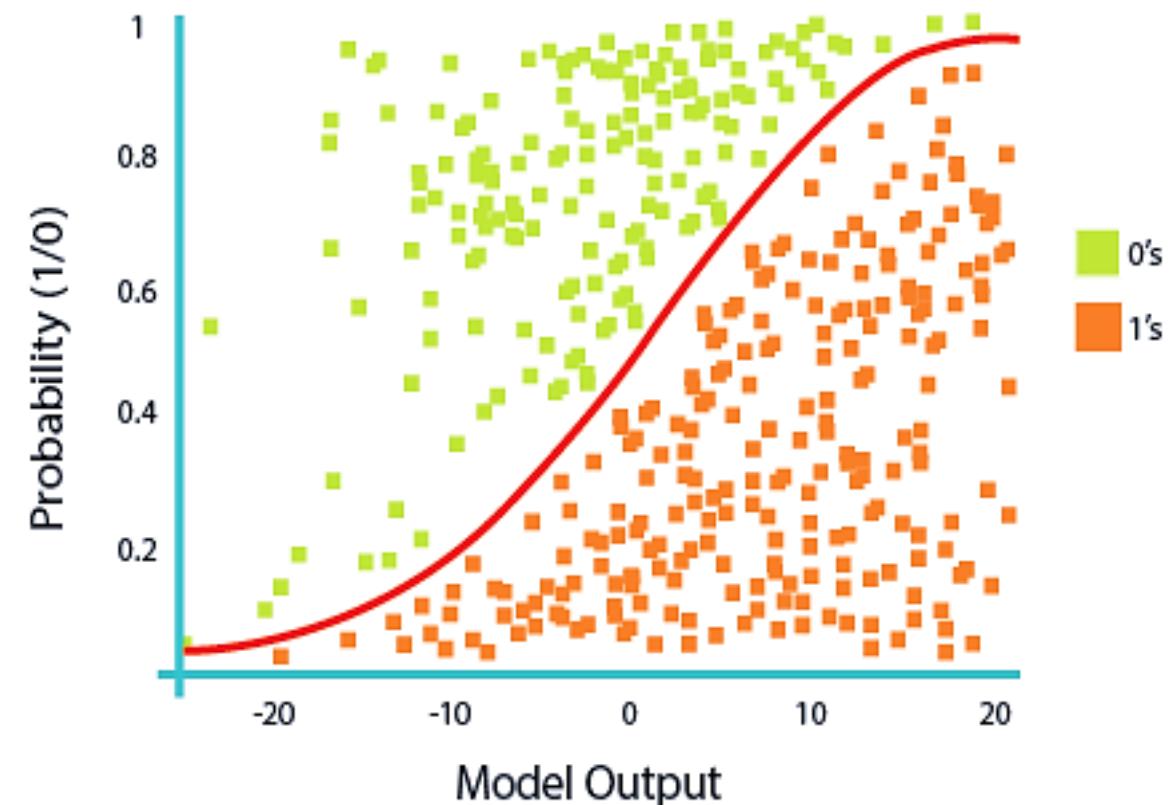
- **Umbral (Threshold)** → Es el **valor límite** que se utiliza para decidir a qué **clase** pertenece una observación según su probabilidad predicha. En regresión logística, el modelo devuelve una probabilidad  $P(Y=1|X) \in [0, 1]$ . Si esa probabilidad es **mayor o igual al umbral**, se clasifica como **1 (evento ocurre)**; si es menor, como **0 (evento no ocurre)**. Por defecto, el umbral suele ser **0.5**, aunque puede ajustarse según el contexto (*por ejemplo, en salud, para priorizar sensibilidad o especificidad*).

$$\hat{Y} = \begin{cases} 1, & \text{si } P(Y = 1|X) \geq \text{umbral} \\ 0, & \text{si } P(Y = 1|X) < \text{umbral} \end{cases}$$

- **Función Sigmoide** → La **función sigmoide** es una función matemática utilizada en la **regresión logística** para transformar una combinación lineal de variables (*que puede tomar cualquier valor real*) en una **probabilidad comprendida entre 0 y 1**. Tiene forma de **S** (sigmoide). Cuando la entrada es **muy negativa** → la salida se acerca a **0** (evento muy improbable). Cuando la entrada es muy positiva → la salida se acerca a **1** (evento muy probable). El **punto central** (donde la probabilidad = 0.5) se produce cuando el valor lineal es 0.

# PRÁCTICA PROFESIONAL – INTRO. AL MACHINE LEARNING

El **modelo** estima una **probabilidad** de pertenecer a la **clase positiva** (normalmente representada como 1). La clase opuesta se llama **clase negativa** (representada como 0). Cada variable independiente (predictora) tiene un **coeficiente  $\beta$  (beta)**. Ese coeficiente indica **cómo cambia el logaritmo de la probabilidad** de pertenecer a la clase positiva. Si  $\beta$  es **positivo**, la variable **aumenta la probabilidad** de que el resultado sea 1 (clase positiva). Si  $\beta$  es **negativo**, la variable **disminuye la probabilidad** de que el resultado sea 1 (clase positiva).





# INTRODUCCIÓN AL MACHINE LEARNING - MÉTRICAS

# PRÁCTICA PROFESIONAL – INTRO. AL ML - MÉTRICAS

- **Métrica:** es una función matemática que permite **cuantificar objetivamente el desempeño de un modelo** comparando sus **predicciones** con los **valores reales** observados. Una métrica **transforma** el comportamiento del modelo sobre un conjunto de datos en un número que resume qué tan “bueno” o “malo” es. Es una **función de evaluación** que **formaliza** el concepto de **calidad predictiva, cuantificando la discrepancia** entre el comportamiento del modelo y la realidad observada, de acuerdo con los objetivos, restricciones y costos implícitos del problema.
- **Representación lógico matemática de una métrica:**
  - $f_\theta(x) = \hat{y}$ : “*Dado  $x$ , creo que el resultado es  $\hat{y}$* ”.  $f_\theta$ : pertenece a una familia de funciones. No es una función fija. Se debe ajustar a partir de entrenamientos sucesivos, calculando así sus parámetros parciales.
  - $M(y, f_\theta(x)) \in \mathbb{R}$ : “*Dado lo que el modelo dijo y lo que era verdad, ¿qué tan bueno fue el modelo?*”. Es una función que juzga a otra función.
- **Interpretación:**
  - Es el proceso mediante el cual el valor numérico producido por una función de evaluación se traduce en un juicio significativo sobre el comportamiento del modelo respecto del problema que se intenta resolver. Es asignar significado operativo al escalar  $M(y, \hat{y})$  comprendiendo qué noción de calidad modela.

# PRÁCTICA PROFESIONAL – INTRO. AL ML - MÉTRICAS

## ❑ Métricas para Logistic Regression:

- **FI SCORE:** es una medida cuantitativa y escalar de qué tan bien el modelo logra clasificar en 0 y 1 después de aplicar un umbral. Evalúa el equilibrio entre: detectar correctamente los casos positivos (recall) y no etiquetar como positivos a los que no lo son (precisión). Cómo juzgar resultados:  $FI = 1$ : “Perfecto” |  $FI \geq 0.85$ : “Muy Bueno” |  $FI$  entre 0.70 – 0.84: “Bueno” |  $FI$  entre 0.55 – 0.69: “Moderado” |  $FI < 0.55$ : “Malo” |  $FI = 0$ : “Inútil”. Responde a la pregunta: “*¿Qué tan confiable es la decisión binaria que toma mi modelo?*”.
- **LOG LOSS:** mide qué tan buenas y realistas son las probabilidades que produce la regresión logística. Evalúa si el modelo: asigna probabilidades altas a lo que realmente ocurre y bajas a lo que no ocurre. Tiene un valor Bajo cuando las probabilidades del modelo coinciden con lo que realmente pasa. Cuanto más Bajo es el valor, de mejor calidad son las probabilidades.  $\text{Log Loss} = 0$ : “Modelo perfecto”. No tiene techo o límite superior. Esta métrica penaliza la “confianza equivocada” computada por el modelo. Cómo juzgar resultados:  $\text{Log Loss} < 0.20$ : “Excelente” |  $\text{Log Loss}$  entre 0.20 – 0.40: “Bueno” |  $\text{Log Loss}$  entre 0.40 – 0.60: “Moderado” |  $\text{Log Loss} > 0.60$ : “Malo”. Responde a la pregunta: “*¿Qué tan creíbles son las probabilidades que estima mi modelo?*”.

# PRÁCTICA PROFESIONAL – INTRO. AL ML - MÉTRICAS

## Métricas para Lineal Regression:

- **R:** medida de la fuerza y dirección de la relación lineal entre valores reales y predichos; indica qué tan alineadas están las predicciones con la tendencia de los datos. Responde: “*¿Qué tan alineadas están las predicciones con los valores reales?*”. Análisis: R = 1: “relación lineal perfecta positiva” | R = 0: “no hay relación lineal” | R = -1: “relación lineal perfecta negativa” | R ≥ 0.85: “Relación muy fuerte” | R esta entre 0.70 – 0.84: “Fuerte” | R esta entre 0.50 – 0.69: “Moderada” | R esta entre 0.30 – 0.49: “Débil” | R < 0.30: “Muy débil”.
- **R<sup>2</sup>:** proporción de la variabilidad de la variable dependiente (Y) que el modelo logra explicar respecto a usar solo el promedio (Media). Variabilidad es qué tan diferentes son los valores de Y entre sí. La métrica trata de explicar qué parte de esas diferencias el modelo logra justificar usando las X (variable independiente). Responde: “*¿Qué porcentaje del comportamiento de Y entiende mi modelo?*”. R<sup>2</sup> ≥ 0.80: “Excelente” | R<sup>2</sup> esta entre 0.60 – 0.79: “Bueno” | R<sup>2</sup> esta entre 0.40 – 0.59: “Moderado” | R<sup>2</sup> esta entre 0.20 – 0.39: “Bajo” | R<sup>2</sup> < 0.20: “Muy Bajo”. Convertir a Valor Porcentual: R%<sup>2</sup> = (R<sup>2</sup> × 100).
- **RMSE — Root Mean Squared Error:** es el error promedio del modelo medido en las mismas unidades que la variable Y. Para cada caso, aplica diferencia entre lo real y lo predicho. Tamaño promedio de las diferencias entre valores reales y predichos por el modelo. Responde: “*En promedio, ¿cuánto me equivoco al predecir Y?*”. Análisis: RMSE < 10%: “Muy Bueno” | RMSE esta entre 10 – 20%: “Aceptable” | RMSE > 20%: “Pobre”.



# PRÁCTICA PROFESIONAL

# PRÁCTICA PROFESIONAL – PRÁCTICA PROFESIONAL

- Definición del Contexto de la Práctica Profesionalizante:
  - ✓ Temática general → Es el **ámbito o área de aplicación** donde se enmarca el proyecto. Describe **el entorno, los actores involucrados y la relevancia del tema**.
  - ✓ Problema específico → Es la **situación concreta que se busca resolver** dentro del **contexto**. Debe ser **medible, delimitado y abordable** con herramientas de Ciencia de Datos o IA.
  - Requisitos de Desarrollo → • **Lenguaje**: Python ≥ 3.12.3 • **IDE**: Visual Studio Code (configurar el *Python interpreter* al entorno con la versión requerida). **Nota**: saber versión actual instalada del intérprete Python: → 1) CMD: `python --version` O 2) `python -c "import sys; print(sys.version)"`.
  - **Flujo de trabajo** → **pandas**: para manipular, limpiar y analizar datos tabulares (*DataFrames*). **numpy**: cálculo y operaciones numéricas y manejo eficiente de arrays multidimensionales. **matplotlib**: biblioteca principal de visualización de datos y gráficos 2D/3D. **scikit-learn**: es una de las librerías más completas para machine learning en Python (aplicación de los algoritmos fundamentales de ML).
  - **Framework de trabajo**: → **Tkinter** es el **framework** estándar de Python para crear interfaces gráficas de usuario (GUI). Es liviano, está incluido por defecto en Python y resulta ideal para proyectos pequeños y medianos que requieren una interfaz sencilla y funcional.

# PRÁCTICA PROFESIONAL – PRÁCTICA PROFESIONAL

- **Partes del Proyecto de Prácticas Profesionales** (“Acercamiento al Campo Laboral I”):
  - **Portada con:** título del proyecto – integrantes (grupo) – nombre de asignatura – año lectivo - año de carrera – docente responsable.
  - **Fundamentación|problema/s específico/s a resolver:** redactar el o los problemas que se desean resolver u optimizar a partir de un caso u observación de un contexto del mundo real.
  - **Objetivo/s:** definir el propósito principal del desarrollo e implementación del proyecto, es decir, qué se busca lograr mediante su creación. Los objetivos deben reflejar la meta general y las metas específicas.
  - **Fuentes de Datos y Datasets:** de dónde provienen los datos que se utilizaron en el proyecto y cómo fueron tratados o creados antes del análisis. En esta sección se deberá realizar el análisis estadístico sobre el conjunto de datos seleccionado previamente.
  - **Algoritmos y justificación:** qué algoritmo usaste, por qué lo elegiste y cómo contribuye a resolver el problema planteado.
  - **Resultados esperados:** no muestra resultados reales todavía, sino que describe qué se espera lograr al finalizar el desarrollo del proyecto, tanto desde el punto de vista técnico como práctico.
  - **Conclusión y Proyección:** se resumen los principales hallazgos y logros del proyecto. Es decir, qué se consiguió, qué se aprendió y cómo responde el modelo al problema planteado.
  - **Referencias y Fuentes:** herramientas utilizadas: lenguaje, versión, APIs, Frameworks, Librerías, etc. – Fuentes de información: Datasets históricos relacionados directamente e indirectamente con el problema, APIs, documentación técnica relacionada a las tecnologías utilizadas en la parte de “Herramientas”.
  - **Nota →** Entregar documentación en formato de archivo .PDF, incluyendo Portada, índice (después de la portada) y Fuentes bibliográficas o de recursos (al final del proyecto).

# PRÁCTICA PROFESIONAL – PRÁCTICA PROFESIONAL

- Diferencia entre Librería y Framework:
  - Una **librería** es un conjunto de **funciones, clases y módulos** que vos **llamás, importas e incluís cuando las necesitás**. Vos **controlás el flujo del programa**, y la librería solo te da herramientas para utilizarlas en la aplicación de algoritmos que resuelven un problema específico dentro del programa. La **librería está a tu servicio**, vos le decís cuándo y cómo usarla. Ofrecen herramientas **puntuales**. Suelen ser más **simples y específicas**.
  - Un **framework** es una estructura **más completa y compleja** que te da un “esqueleto” para construir una aplicación. Con un framework, **vos adaptás tu código a sus reglas y flujo**, no al revés. Suele ser más **completo y estructurado**. Su función principal es **construir** toda una aplicación o **sistema integramente**.
- Importar Dependencias → Una vez dentro de VS Code: **Presiona “Ctrl + Shift + P”** → Escribe “**Python: Select Interpreter**” y selecciona la versión de Python que quieras utilizar. **Abre la terminal** integrada en VS Code: “**Ctrl + ñ**” → **Escribir** en la terminal lo siguiente: “**pip install pandas numpy matplotlib scikit-learn**”.
- **Aclaraciones:** **tkinter** viene incluido con Python, así que no hace falta instalarlo. **mpl\_toolkits.mplot3d** viene con **matplotlib**, así que tampoco necesita instalación adicional.

# PRÁCTICA PROFESIONAL – PRÁCTICA PROFESIONAL

- **Otros Datos:**

- Un **gestor de dependencias** (o *package manager*) es una herramienta que se encarga de: **Descargar, instalar, actualizar y eliminar librerías o paquetes** de software. Asegurar que tu proyecto tenga todas las **dependencias** (otros paquetes que necesita para funcionar correctamente). Llevar un **control de versiones**, evitando conflictos entre librerías. Un gestor de dependencias te evita tener que buscar, descargar e instalar manualmente los módulos que usa tu programa.
- **Dependencia** → es cualquier **librería externa** que tu programa necesita para funcionar correctamente. Existen **dependencias críticas y no críticas**.
- Requirements → Si ya **instalaste las librerías necesarias** para tu proyecto (por ejemplo: *pandas*, *numpy*, *matplotlib*, *scikit-learn*, etc.), podés **crear el archivo requirements.txt** automáticamente desde la consola: “**pip freeze > requirements.txt**” → “*pip freeze*” lista todas las librerías instaladas en el entorno actual con sus versiones exactas. El símbolo “>” redirige esa salida a un archivo llamado “*requirements.txt*”.
- ¿Qué es “**pip**”? → **pip** significa “*Pip Installs Packages*”. Es la **herramienta oficial de Python** para **instalar y administrar paquetes** (librerías o frameworks).



# INTRODUCCIÓN A LA ESTADÍSTICA

# PRÁCTICA PROFESIONAL – INTRODUCCIÓN A LA ESTADÍSTICA

## ○ Algunos Conceptos Importantes:

- ✓ **Media** → Es el **valor promedio** de los datos. Representa el **punto de equilibrio** del conjunto de datos. Da una idea del **valor típico** del conjunto. **Se ve muy afectada por valores extremos** (*outliers*). Sirve para describir el **nivel general** del fenómeno que estás analizando. "*En promedio, ¿cuánto vale el dato típico?*" → Si la media aumenta o disminuye en el tiempo → indica **tendencias**.
- ✓ **Mediana** → Es el **valor central** del conjunto de datos una vez ordenado de manera creciente. Es un indicador del **valor típico**, pero **no se afecta por valores extremos**. Cuando la media y la mediana son muy diferentes (*datos desbalanceados*), significa que **hay asimetría** en los datos. La mediana refleja mejor la realidad. "*¿Cuál es el valor central, el que se ubica en el medio de todos los datos?*".
- ✓ **Rango** → Es la diferencia entre el valor **máximo** y el **mínimo**. Indica **cuán extendidos** están los valores. Te dice **cuánto se dispersan los datos**, pero **solo considera 2 valores u observaciones**, puede que no refleje alta fidelidad. Si hay un solo dato extremo, el rango crece mucho. Se usa como primera medida de **dispersión rápida**, no como análisis profundo. "*¿Cuál es la distancia total entre el menor y el mayor valor observado?*".
- ✓ **Varianza** → Mide cuán **dispersos** están los datos respecto a la media. **Varianza alta**: los datos están muy esparcidos, poco consistentes. **Varianza baja**: los datos están agrupados, más consistentes (*los datos están concentrados cerca del promedio*). Mide **variabilidad** en términos conceptuales, pero no en la misma unidad de los datos originales. "*En promedio, ¿qué tan lejos están los valores de la media?*".

# PRÁCTICA PROFESIONAL – INTRODUCCIÓN A LA ESTADÍSTICA

- Algunos Conceptos Importantes:
- ✓ Desviación Estándar → Es la medida de **dispersión más utilizada** y se usa como **medida práctica**. Está en la **misma unidad** que los datos. Representa la **dispersión promedio** en la misma unidad que los datos. **Alta desviación estándar**: mucha variabilidad. **Baja desviación estándar**: valores parecidos entre sí. Cuanto **mayor** es la desviación estándar, **menos consistentes** son los datos. "En promedio, ¿cuánto se alejan los datos del valor promedio?".
- ✓ Coeficiente de Variación (CV) → Sirve para comparar **variabilidad relativa** entre conjuntos con **unidades distintas o escalas diferentes**. Se interpreta como **porcentaje de variación respecto al promedio**. CV <10%: Muy baja variabilidad (*datos muy consistentes*). CV<10%–20%:Variabilidad moderada. CV>20%:Alta variabilidad (*datos dispersos*). "¿Cuánta variabilidad tengo en relación al promedio?".
- ✓ Cuartiles → Son valores que **dividen un conjunto de datos ordenados de forma creciente en 4 partes iguales**, donde cada parte contiene el **25%** de los datos. Se representan como: **Q1**: Primer cuartil (25%). **Q2**: Segundo cuartil (50%) (*es la mediana*). **Q3**:Tercer cuartil (75%). Podemos definir el **recorrido intercuartílico (IQR)**: Mide la **dispersión del 50% central** de los datos, ignorando extremos.  $IQR=(Q3-Q1)$ . Se utilizan para: **Describir cómo se distribuyen los datos**, especialmente cuando no son simétricos. **Detectar valores atípicos (outliers)**. **Entender la dispersión sin verse afectados por valores extremos**, a diferencia del rango total. **Q1**: "¿Cuál es el valor por debajo del cual se encuentra el 25% de los datos?". **Q2 (Mediana)**: "¿Cuál es el valor central, donde se encuentra el 50% de los datos?". **Q3**: "¿Cuál es el valor por debajo del cual se encuentra el 75% de los datos?". **IQR**: "¿Cuál es el rango en el que se encuentra el 50% central de los datos, ignorando los extremos?". **Q1 Bajo**: Muchos datos se concentran en valores **pequeños**. **Q1 Alto**: El 25% inferior ya está en valores relativamente altos. **Q2 Bajo**: La mayoría de los datos tienden a estar en valores **pequeños**. **Q2 Alto**: La mayoría de los datos se ubican en valores **altos**. **Q3 Bajo**: Incluso la mayoría de los valores **no llega a niveles altos**. **Q3 Alto**: Una gran parte de los datos está en valores altos. **IQR Bajo**: Los datos están **apretados** entre sí (*poca variación y estabilidad*). **IQR Alto**: Los datos centrales están **muy dispersos** (*existe diversidad y heterogeneidad*).

# PRÁCTICA PROFESIONAL – INTRODUCCIÓN A LA ESTADÍSTICA

- Algunos Conceptos Importantes:
- ✓ **Outlier** → Un **outlier** (*valor atípico*) es un valor que **se aleja mucho del resto** de los datos. No sigue el **patrón general** del conjunto.  
**Puede ser:** Un **valor muy alto** comparado con los demás. Un **valor muy bajo** comparado con los demás. **¿Por qué existen los outliers?** Pueden aparecer por: Errores de medición, Datos excepcionales reales, Cambios o casos muy especiales. El **objetivo** es saber si esos valores: **Deben corregirse** (si son errores). **Deben eliminarse** (si distorsionan el análisis). **Deben analizarse aparte** (si representan casos especiales importantes). “*¿Este dato pertenece realmente al comportamiento del grupo, o es una excepción?*”. *¿Este valor es representativo o no?.* *¿Debo incluirlo en el análisis o tratarlo por separado?.* *¿Afecta la media, la varianza o las conclusiones finales?*”.
- ✓ **Detección de Outliers** → Límite Inferior= $Q_1 - (1,5 \cdot IQR)$ . Límite Superior= $Q_3 + (1,5 \cdot IQR)$ . **Regla** → Un valor es outlier si:  $\{x < L_I\} \text{ OR } \{x > L_S\}$ . **Muchos** outliers: Los datos son **muy variados**, existe **heterogeneidad**. **Pocos** outliers: El grupo es **consistente y estable**. **Un** outlier muy extremo: Puede distorsionar la media y desviación estándar.

$$L_{\inf} = Q_1 - 1.5 \cdot IQR$$

$$L_{\sup} = Q_3 + 1.5 \cdot IQR$$

$x \in \text{Outliers}$  si y solo si  $(x < L_{\inf}) \vee (x > L_{\sup})$

$$x > Q_3 + 1.5 \cdot (Q_3 - Q_1)$$

$$x < Q_1 - 1.5 \cdot (Q_3 - Q_1)$$

$x$  es outlier  $\iff x \notin [Q_1 - 1.5 \cdot IQR, Q_3 + 1.5 \cdot IQR]$

# PRÁCTICA PROFESIONAL – INTRODUCCIÓN A LA ESTADÍSTICA

- Algunos Conceptos Importantes:
- ✓ **Dato Típico (Typical Data Point)** → un **dato típico** es una instancia que **representa adecuadamente la distribución estadística general del conjunto de datos**. Es decir, presenta valores coherentes, esperables y frecuentes según el patrón global. **Características técnicas:** Se ubica cerca de la **media** o en zonas de **alta densidad** de datos. No contiene valores extremos. Es consistente con la estructura o correlaciones del dataset. Contribuye a entrenamientos estables, ya que el modelo lo interpreta como parte del “comportamiento normal”.
- ✓ **Ruido (Noise)** → se refiere a variaciones aleatorias o errores en los datos que **no aportan información relevante** y que pueden degradar el rendimiento del modelo. **Tipos de ruido:** Ruido aleatorio (*random noise*) → son **variaciones impredecibles** en los datos, como si el valor “temblara”. Ocurre por imprecisiones de sensores, errores humanos al medir o simples fluctuaciones naturales. No sigue ningún patrón; es azar puro. Ruido sistemático (*systematic noise / bias*) → **errores constantes** generados por instrumentos o procesos sesgados. Es un problema en la forma en que se mide, registra o procesa la información. Proviene del sistema de medición, no del azar. Es una anomalía constante y afecta a la totalidad del conjunto de datos. Ruido de etiquetado (*label noise*) → **errores en las etiquetas de entrenamiento en tareas supervisadas**: algunas observaciones (*clases o valores objetivo*) están mal etiquetadas. Este tipo de ruido es muy dañino porque el modelo aprende ejemplos “mal enseñados”.
- ✓ **Sesgo (Bias)** → es un **error sistemático** en un modelo de machine learning: no es un error aleatorio, sino una **tendencia del modelo a equivocarse siempre hacia el mismo lado**. Aparece cuando un modelo aprende una **versión demasiado simplificada o incorrecta de la realidad**. Es como si el modelo mirara el mundo con anteojos distorsionados. **Posibles causas:** No alcanza para capturar la complejidad del problema (*por ejemplo, mala elección del algoritmo o modelo de ML*). El modelo aprende mal porque los datos están incompletos o desbalanceados (*datos poco representativos en el estudio del problema*). El modelo supone cosas que no son reales (*supuestos incorrectos*).