

## SESIÓN PROCESAMIENTO Y ESCALAMIENTO DE DATOS

### CONTENIDOS:

- El concepto de preprocesamiento.
- Tareas más frecuentes de preprocesamiento.
- Codificación de Variables Categóricas
  - Label Encoding
  - One-Hot Encoding
- Variables dummy.
- Implementación con la librería scikit-learn.
- Escalamiento de datos: el concepto de distancia.
- Tipos de Distancia
  - Distancia Manhattan
  - Distancia Euclidiana
  - Distancia Minkowski
- Técnicas más usadas de Escalamiento
  - Escalador Min-Max
  - Escalador Estándar
- Implementación escalamiento con la librería scikit-learn.

### EL CONCEPTO DE PREPROCESAMIENTO.

El preprocesamiento de datos es una etapa crítica en el flujo de trabajo de la ciencia de datos y el aprendizaje automático. Consiste en la preparación y transformación de los datos en bruto para que sean adecuados para su análisis y modelado. Los datos en bruto suelen estar incompletos, contener errores, ruido o inconsistencias que pueden afectar negativamente el rendimiento de los modelos de Machine Learning. Por lo tanto, el preprocesamiento es esencial para garantizar que los datos sean precisos, consistentes y útiles para el análisis.

## ¿Por qué es importante el preprocesamiento?

### 1. Mejora la calidad de los datos:

- Los datos en bruto suelen contener errores, valores faltantes, duplicados o inconsistencias. El preprocesamiento permite identificar y corregir estos problemas, lo que resulta en un conjunto de datos más limpio y confiable.

### 2. Reduce el ruido y la redundancia:

- Los datos pueden contener información irrelevante o redundante que no contribuye al análisis. El preprocesamiento ayuda a eliminar este ruido, lo que facilita la interpretación de los datos y mejora la eficiencia del modelo.

### 3. Mejora el rendimiento del modelo:

- Los modelos de Machine Learning funcionan mejor con datos limpios y normalizados. El preprocesamiento asegura que los datos estén en un formato adecuado para ser procesados por los algoritmos, lo que mejora la precisión y la eficiencia del modelo.

## TAREAS MÁS FRECUENTES DE PREPROCESAMIENTO

El preprocesamiento de datos abarca una variedad de tareas que dependen de la naturaleza de los datos y los objetivos del análisis. A continuación, se describen las tareas más comunes:

### 1. Limpieza de Datos

La limpieza de datos es una de las tareas más importantes en el preprocesamiento. Los datos en bruto suelen contener errores, valores faltantes o inconsistencias que deben ser corregidos antes de proceder con el análisis. Algunas técnicas comunes incluyen:

- **Eliminación de valores nulos:** Si una fila tiene muchos valores faltantes, puede ser eliminada. Alternativamente, los valores nulos pueden ser imputados utilizando la media, la mediana o la moda.
- **Eliminación de duplicados:** Los registros duplicados pueden distorsionar el análisis y deben ser eliminados.

- **Corrección de inconsistencias:** Por ejemplo, si una columna de fechas tiene formatos inconsistentes (como "2023-01-01" y "01/01/2023"), deben ser estandarizados.

**Ejemplo:**

```
1 import pandas as pd
2
3 # Dataset con valores nulos y duplicados
4 data = {'Nombre': ['Alice', 'Bob', 'Charlie', None],
5         'Edad': [25, None, 30, 22],
6         'Ciudad': ['Madrid', 'Barcelona', 'Madrid', 'Madrid']}
7 df = pd.DataFrame(data)
8
9 # Eliminar filas con valores nulos
10 df_cleaned = df.dropna()
11
12 # Eliminar duplicados
13 df_cleaned = df_cleaned.drop_duplicates()
14
15 print(df_cleaned)
```

*Ilustración 1 limpieza de datos.*

## 2. Normalización

La normalización es el proceso de escalar los datos para que estén en un rango específico, como [0, 1] o [-1, 1]. Esto es especialmente útil cuando las variables tienen diferentes escalas, lo que puede afectar el rendimiento de los modelos que dependen de la distancia, como K-Nearest Neighbors (KNN) o Support Vector Machines (SVM).

**Ejemplo:**

```
1 from sklearn.preprocessing import MinMaxScaler
2 import numpy as np
3
4 # Dataset con valores en diferentes escalas
5 data = np.array([[10, 200], [20, 300], [30, 400]])
6
7 # Aplicar Min-Max Scaling
8 scaler = MinMaxScaler()
9 data_normalized = scaler.fit_transform(data)
10
11 print(data_normalized)
```

*Ilustración 2 Normalización.*

### 3. Transformación de Datos

La transformación de datos implica aplicar funciones matemáticas a los datos para mejorar su distribución o reducir el impacto de valores extremos. Algunas transformaciones comunes incluyen:

- **Transformación logarítmica:** Útil para reducir el impacto de valores extremos en datos con una distribución sesgada.
- **Transformación de raíz cuadrada:** Similar a la transformación logarítmica, pero menos agresiva.
- **Transformación de Box-Cox:** Una técnica más avanzada que busca estabilizar la varianza y normalizar la distribución.

Ejemplo:

```
1  import numpy as np
2
3  # Aplicar una transformación logarítmica
4  data = np.array([1, 10, 100, 1000])
5  data_transformed = np.log(data)
6
7  print(data_transformed)
```

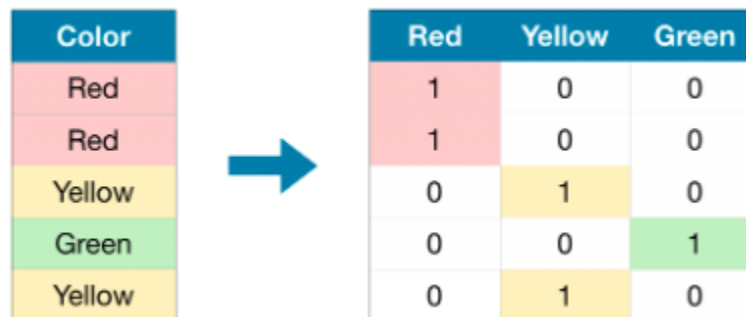
*Ilustración 3 Transformación de datos.*

### CODIFICACIÓN DE VARIABLES CATEGÓRICAS

Las variables categóricas son un tipo de dato que representa categorías o etiquetas, como colores, tamaños, géneros, niveles educativos, entre otros. A diferencia de las variables numéricas, que pueden ser procesadas directamente por los algoritmos de Machine Learning, las variables categóricas deben ser transformadas en un formato numérico para que los modelos puedan interpretarlas. Esto se debe a que la mayoría de los algoritmos de Machine Learning están diseñados para trabajar con números, no con texto o etiquetas.

La codificación de variables categóricas es un paso crucial en el preprocesamiento de datos, ya que permite que los modelos aprendan de manera efectiva a partir de estas variables. Sin embargo, es importante elegir la técnica de codificación adecuada, ya que una elección incorrecta puede introducir sesgos o relaciones artificiales en los datos.

Las dos técnicas más comunes para codificar variables categóricas son **Label Encoding** y **One-Hot Encoding**. Cada una tiene sus propias ventajas y desventajas, y su elección depende del tipo de variable categórica y del modelo que se esté utilizando.



The diagram illustrates the transformation of categorical data. On the left, a table with a 'Color' column contains five rows: 'Red', 'Red', 'Yellow', 'Green', and 'Yellow'. A blue arrow points to the right, where a new table is shown. This table has three columns: 'Red', 'Yellow', and 'Green'. Each row in the new table corresponds to a row in the original table, with a '1' in the column corresponding to the color and '0' in the others. For example, the first row (Red) has a '1' in the 'Red' column and '0' in 'Yellow' and 'Green'.

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

*Ilustración 4 Codificación de Variables Categóricas.*

### Label Encoding

El **Label Encoding** es una técnica que asigna un número único a cada categoría. Es decir, cada valor categórico se reemplaza por un número entero. Esta técnica es útil cuando las variables categóricas tienen un **orden natural** o una jerarquía intrínseca.

#### Ventajas:

- **Simplicidad:** Es fácil de implementar y no aumenta la dimensionalidad de los datos.
- **Eficiencia:** Es útil cuando las categorías tienen un orden natural, como en el caso de niveles educativos o rangos de edad.

#### Desventajas:

- **Relaciones numéricas artificiales:** Al asignar números a las categorías, el modelo puede interpretar que existe una relación numérica entre ellas. Por ejemplo, si "Pequeño" = 0, "Mediano" = 1 y "Grande" = 2, el modelo podría interpretar que "Grande" es mayor que "Pequeño", lo que no siempre es deseable.
- **No es adecuado para categorías sin orden:** Si las categorías no tienen un orden natural (por ejemplo, colores), Label Encoding puede introducir sesgos en el modelo.

**Ejemplo:** Supongamos que tenemos una variable categórica llamada "Tamaño" con los valores: ["Pequeño", "Mediano", "Grande"]. Después de aplicar Label Encoding, los valores se transformarían en: [0, 1, 2].

Para implementar este ejemplo en Python sería de la siguiente forma:

```
1  from sklearn.preprocessing import LabelEncoder
2
3  # Ejemplo de Label Encoding
4  le = LabelEncoder()
5  datos = ['Pequeño', 'Mediano', 'Grande', 'Mediano']
6  datos_transformados = le.fit_transform(datos)
7  print(datos_transformados) # Salida: [2 1 0 1]
```

*Ilustración 6 Label Encoding.*

### One-Hot Encoding

El **One-Hot Encoding** es una técnica que convierte cada categoría en una **columna binaria** (0 o 1). Es decir, para cada categoría única en la variable original, se crea una nueva columna en el dataset. Si la categoría está presente en una fila, se marca con un 1; de lo contrario, se marca con un 0.

#### Ventajas:

- **Elimina relaciones numéricas artificiales:** Al convertir cada categoría en una columna binaria, se evita que el modelo interprete relaciones numéricas entre las categorías.
- **Adecuado para categorías sin orden:** Es ideal para variables categóricas que no tienen un orden natural, como colores o tipos de productos.

#### Desventajas:

- **Aumento de la dimensionalidad:** Si la variable categórica tiene muchas categorías únicas, el número de columnas en el dataset puede aumentar significativamente, lo que puede afectar el rendimiento del modelo.

- **Dispersión de los datos:** El dataset resultante puede tener muchas columnas con valores 0, lo que puede dificultar el procesamiento en algunos casos.

**Ejemplo:** Supongamos que tenemos la misma variable categórica "Tamaño" con los valores: ["Pequeño", "Mediano", "Grande"]. Después de aplicar One-Hot Encoding, se crearían tres columnas nuevas: "Tamaño\_Pequeño", "Tamaño\_Mediano" y "Tamaño\_Grande". La transformación sería:

Tamaño	Tamaño_Pequeño	Tamaño_Mediano	Tamaño_Grande
Pequeño	1	0	0
Mediano	0	1	0
Grande	0	0	1

Si lo implementamos en Python se vería así:

```
1  from sklearn.preprocessing import OneHotEncoder
2  import numpy as np
3
4  # Ejemplo de One-Hot Encoding
5  datos = np.array([["Pequeño"], ["Mediano"], ["Grande"]])
6  ohe = OneHotEncoder()
7  datos_transformados = ohe.fit_transform(datos).toarray()
8  print(datos_transformados)
9  # Salida:
10 # [[0. 0. 1.]
11 #  [0. 1. 0.]
12 #  [1. 0. 0.]
```

*Ilustración 8 One-Hot Encoding.*

## VARIABLES DUMMY

Las **variables dummy** son una técnica de codificación utilizada para transformar variables categóricas en un formato numérico que pueda ser procesado por los algoritmos de Machine Learning. Esta técnica es similar al **One-Hot Encoding**, pero se implementa directamente en DataFrames de Pandas, lo que la hace especialmente útil para trabajar con datos tabulares. Las variables dummy convierten cada categoría única en una nueva columna binaria (0 o 1), donde un valor de 1 indica la presencia de la categoría y un valor de 0 indica su ausencia.

Las variables dummy son útiles para representar categorías sin orden natural, como colores o tipos de productos. A diferencia de **Label Encoding**, que asigna números a categorías, las variables dummy evitan relaciones numéricas artificiales, lo que las hace ideales para modelos que no deben interpretar jerarquías.

### Ventajas

1. **Elimina relaciones numéricas artificiales:** Ideal para categorías sin orden.
2. **Fácil interpretación:** Cada columna representa una categoría específica.
3. **Compatibilidad con modelos:** Funciona bien con regresión lineal, árboles de decisión y redes neuronales.

### Desventajas

1. **Aumento de dimensionalidad:** Muchas categorías únicas pueden inflar el dataset.
2. **Dispersión de datos:** Muchos valores 0 pueden dificultar el procesamiento.
3. **Redundancia:** Una columna dummy puede ser deducida de las demás.

**Ejemplo:** Para una variable "Color" con valores ["Rojo", "Azul", "Verde"], las variables dummy crearían tres columnas:

Color	Color_Rojo	Color_Azul	Color_Verde
Rojo	1	0	0



Color	Color_Rojo	Color_Azul	Color_Verde
Azul	0	1	0
Verde	0	0	1

Pandas ofrece la función **get\_dummies** para crear variables dummy, en Python se ve así:

```

1  import pandas as pd
2
3  # Crear un DataFrame con una columna categórica
4  df = pd.DataFrame({"Color": ["Rojo", "Azul", "Verde", "Azul", "Rojo"]})
5
6  # Aplicar variables dummy
7  df_dummies = pd.get_dummies(df, columns=["Color"])
8
9  print(df_dummies)
10 #Salida:
11 #   Color_Azul  Color_Rojo  Color_Verde
12 #0      False      True      False
13 #1       True      False      False
14 #2      False      False      True
15 #3       True      False      False
16 #4      False      True      False

```

*Ilustración 10 variables dummy en pandas.*

**Salida:**

	Color_Azul	Color_Rojo	Color_Verde
0	0	1	0
1	1	0	0
2	0	0	1
3	1	0	0
4	0	1	0

## IMPLEMENTACIÓN CON LA LIBRERÍA SCKIT-LEARN

Scikit-Learn es una de las librerías más populares en Python para Machine Learning, y ofrece una amplia gama de herramientas para el preprocesamiento de datos, incluyendo la transformación de variables categóricas y el escalamiento de datos. Estas herramientas son esenciales para preparar los datos antes de entrenar modelos de Machine Learning, ya que garantizan que los datos estén en un formato adecuado para ser procesados por los algoritmos.

Scikit-Learn proporciona clases como **LabelEncoder** y **OneHotEncoder** para transformar variables categóricas en valores numéricos. Estas herramientas son fáciles de usar y se integran perfectamente con el flujo de trabajo de Machine Learning.

### Actividad Práctica Guiada de implementación librería scikit-Learn:

#### 1. Importamos librerías:

- Primero, necesitamos importar las clases `LabelEncoder` y `OneHotEncoder` de `Scikit-Learn`, así como la librería `NumPy` para manejar arrays.

```
1 from sklearn.preprocessing import LabelEncoder, OneHotEncoder
2 import numpy as np
```

*Ilustración 11 importar librerías.*

#### 2. Crear un Array de Datos Categóricos

- Vamos a crear un array de datos categóricos que representan niveles: "bajo", "medio" y "alto".
- 

```
5 datos_categoricos = np.array(["bajo", "medio", "alto", "medio"])
6 print("Datos categóricos originales:\n", datos_categoricos)
7 #Datos categóricos originales:
8 # ['bajo' 'medio' 'alto' 'medio']
```

*Ilustración 12 crear array.*

### 3. Aplicar Label Encoding

- El **Label Encoding** asigna un número único a cada categoría. Vamos a utilizar la clase **LabelEncoder** para transformar los datos categóricos en valores numéricos.

```
11 le = LabelEncoder()
12 datos_numericos = le.fit_transform(datos_categoricos)
13 print("Label Encoding:", datos_numericos) # Salida: [0 1 2 1]
```

*Ilustración 13 aplicar Label Encoding.*

### 4. Aplicar One-Hot Encoding

- El **One-Hot Encoding** convierte cada categoría en una columna binaria (0 o 1). Vamos a utilizar la clase **OneHotEncoder** para realizar esta transformación.


```
15 # Nota: reshape(-1, 1) es necesario porque OneHotEncoder espera una matriz 2D
16 ohe = OneHotEncoder()
17 datos_transformados = ohe.fit_transform(datos_categoricos.reshape(-1, 1)).toarray()
18 print("One-Hot Encoding:\n", datos_transformados)
19 #Salida:
20 #One-Hot Encoding:
21 # [[0. 1. 0.]
22 #  [0. 0. 1.]
23 #  [1. 0. 0.]
24 #  [0. 0. 1.]]
```

*Ilustración 14 aplicar One-Hot Encoding.*

### 5. Salida esperada

```
Datos categóricos originales:
['bajo' 'medio' 'alto' 'medio']
Label Encoding: [1 2 0 2]
One-Hot Encoding:
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

*Ilustración 15 Salida ejercicio guiado.*



Interpretemos los resultados:

- **Label Encoding:** Los datos categóricos se han convertido en números enteros. Esta técnica es útil cuando las categorías tienen un orden natural.
- **One-Hot Encoding:** Los datos categóricos se han convertido en columnas binarias. Esta técnica es útil cuando las categorías no tienen un orden natural y se desea evitar relaciones numéricas artificiales.

## ESCALAMIENTO DE DATOS: EL CONCEPTO DE DISTANCIA

El escalamiento de datos es un proceso fundamental en el preprocesamiento de datos, especialmente cuando se utilizan modelos de Machine Learning que dependen de la **distancia** entre puntos de datos. Estos modelos, como **K-Nearest Neighbors (KNN)**, **Support Vector Machines (SVM)** y otros algoritmos basados en distancias, calculan la similitud entre observaciones utilizando métricas de distancia. Si las características no están en la misma escala, algunas pueden dominar el cálculo de la distancia, lo que lleva a resultados sesgados y a un rendimiento subóptimo del modelo.

### ¿Por qué es importante el escalamiento?

1. **Equidad en la contribución de las características:**
  - Cuando las características tienen diferentes escalas (por ejemplo, una característica varía entre 0 y 1, y otra entre 0 y 1000), la característica con valores más grandes puede dominar el cálculo de la distancia. Esto ocurre porque las distancias se calculan en función de las diferencias entre los valores de las características. El escalamiento asegura que todas las características contribuyan de manera equitativa al cálculo de la distancia.
2. **Mejora del rendimiento del modelo:**
  - Los modelos que dependen de la distancia, como KNN o SVM, funcionan mejor cuando los datos están normalizados. El escalamiento puede mejorar la precisión y la eficiencia del modelo, ya que evita que una característica domine sobre las demás.

### 3. Convergencia más rápida en algoritmos iterativos:

- En algoritmos como el descenso de gradiente, el escalamiento de datos puede acelerar la convergencia, ya que las características estarán en un rango similar. Esto es especialmente importante en modelos de regresión lineal, redes neuronales y otros algoritmos que utilizan optimización iterativa.

## TIPOS DE DISTANCIA

En el contexto de Machine Learning, la **distancia** es una medida de similitud o disimilitud entre dos puntos de datos. A continuación, se describen los tipos de distancia más comunes:

### Distancia Manhattan

También conocida como distancia L1 o distancia de la ciudad cuadrículada, la distancia Manhattan se calcula sumando las diferencias absolutas de sus coordenadas. Imagina que estás en una cuadrícula de calles como en Manhattan, donde solo puedes moverte en líneas rectas horizontales y verticales.

Para dos puntos A(x1, y1) y B(x2, y2), la fórmula de la distancia Manhattan es:

$$d_{Manhattan} = |x_1 - x_2| + |y_1 - y_2|$$

*Ilustración 16 Fórmula distancia Manhattan.*

### Distancia Euclidiana

La distancia Euclidiana, también conocida como distancia L2, es la distancia más corta entre dos puntos en un espacio Euclidiano (como una línea recta). Es la medida más común y la que usualmente asociamos con la "distancia" en el sentido cotidiano.

Para dos puntos A(x1, y1) y B(x2, y2), la fórmula de la distancia Euclidiana es:

$$d_{Euclidiana} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

*Ilustración 17 Fórmula distancia Euclidiana.*

### Distancia Minkowski

La distancia Minkowski es una generalización de las distancias Manhattan y Euclidiana, que incluye un parámetro  $p$  que determina la forma de la distancia. Cuando  $p=1$ , coincide con la distancia Manhattan, y cuando  $p=2$ , coincide con la distancia Euclidiana.

Para dos puntos  $A(x_1, y_1, \dots, z_1)$  y  $B(x_2, y_2, \dots, z_2)$ , la fórmula de la distancia Minkowski es:

$$d_{Minkowski} = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

*Ilustración 18 Fórmula distancia Minkowski.*

Dependiendo del valor de  $p$ , se pueden obtener diferentes métricas, permitiendo flexibilidad en el cálculo de distancias en diferentes contextos.

### TÉCNICAS MÁS USADAS DE ESCALAMIENTO

El escalamiento de datos es un paso crucial en el preprocesamiento de datos, especialmente cuando se utilizan modelos que dependen de la distancia, como KNN o SVM. Las técnicas de escalamiento, como **Min-Max Scaling** y **Standard Scaling**, aseguran que todas las características contribuyan de manera equitativa al cálculo de la distancia, lo que mejora el rendimiento del modelo. La elección de la técnica de escalamiento depende de la naturaleza de los datos y del modelo que se esté utilizando.

El escalamiento de datos es esencial para garantizar que todas las características estén en una escala similar. A continuación, se describen las técnicas de escalamiento más comunes:

#### Escalador Min-Max

El **escalador Min-Max** normaliza los datos a un rango específico, generalmente  $[0, 1]$ . Esta técnica es útil cuando se desea que los datos estén dentro de un rango definido.

**Fórmula:**

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

*Ilustración 19 Fórmula escalador Min-Max.*

**Ventajas:**

- Fácil de implementar.
- Mantiene la forma de la distribución original de los datos.

**Desventajas:**

- Sensible a valores atípicos (outliers), ya que estos pueden distorsionar el rango.

## 2. Escalador Estándar

El **escalador estándar** transforma los datos restando la media y dividiendo por la desviación estándar. Esto resulta en una distribución con media 0 y desviación estándar 1.

**Fórmula:**

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

*Ilustración 20 Fórmula escalador Estándar.*

Donde  $\mu$  es la media y  $\sigma$  es la desviación estándar.

**Ventajas:**

- Menos sensible a valores atípicos en comparación con Min-Max Scaling.
- Útil cuando los datos tienen una distribución normal o cuando se desea centrar los datos alrededor de 0.

**Desventajas:**

- No garantiza un rango específico para los datos.

Tabla comparativa:

Característica	Min-Max Scaling	Standard Scaling
Rango de salida	[0, 1] o un rango definido por el usuario.	No tiene un rango específico.
Sensibilidad a outliers	Muy sensible.	Menos sensible.
Distribución resultante	Mantiene la forma original de los datos.	Centra los datos en 0 con desviación estándar 1.
Uso recomendado	Cuando se necesita un rango específico.	Cuando los datos tienen una distribución normal o se desea centrarlos.

*Ilustración 21 Escalador Min-Max y Estándar.*

## IMPLEMENTACIÓN DE ESCALAMIENTO CON SCKIT-LEARN

El escalamiento es esencial cuando se trabaja con modelos que dependen de la distancia, como **K-Nearest Neighbors (KNN)**, **Support Vector Machines (SVM)**, o algoritmos que utilizan optimización iterativa, como la regresión lineal o las redes neuronales. A continuación, se muestra cómo implementar diferentes técnicas de escalamiento utilizando Scikit-Learn.

La implementación de técnicas de escalamiento con Scikit-Learn es sencilla y eficiente. Dependiendo de la naturaleza de los datos y del modelo que se esté utilizando, se puede elegir entre **MinMaxScaler**, **StandardScaler** u otro. Estas técnicas aseguran que todas las características contribuyan de manera equitativa al cálculo de la distancia, lo que mejora el rendimiento del modelo. Además, la integración de estas técnicas en un **pipeline** permite automatizar el flujo de trabajo, mejorando la eficiencia y la reproducibilidad del proceso.

A continuación, un ejercicio guiado de cómo implementar estas técnicas en un dataset:

### 1. Importación de Librerías y Creación del Dataset

Primero, importamos las librerías necesarias y creamos un dataset de ejemplo.



```
1 from sklearn.preprocessing import MinMaxScaler, StandardScaler
2 import numpy as np
3
4 # Crear un dataset de ejemplo
5 datos = np.array([[10, 200], [20, 300], [30, 400], [40, 500]])
6 print("Datos originales:\n", datos)
```

*Ilustración 22 importar librerías y dataset.*

```
Datos originales:
[[ 10 200]
 [ 20 300]
 [ 30 400]
 [ 40 500]]
```

*Ilustración 23 salida datos originales.*

## 2. Escalamiento con MinMaxScaler

El **MinMaxScaler** normaliza los datos a un rango específico, generalmente [0, 1].

```
8 # Aplicar Min-Max Scaling
9 minmax_scaler = MinMaxScaler()
10 datos_minmax = minmax_scaler.fit_transform(datos)
11 print("Min-Max Scaling:\n", datos_minmax)
```

*Ilustración 24 Min-Max Scaling.*

```
Min-Max Scaling:
[[0.         0.         ]
 [0.33333333 0.33333333]
 [0.66666667 0.66666667]
 [1.         1.         ]]
```

*Ilustración 25 Salida Min-Max Scaling.*

#### Explicación:

- La primera columna se escaló de [10, 40] a [0, 1].
- La segunda columna se escaló de [200, 500] a [0, 1].

### 3. Escalamiento con StandardScaler

El **StandardScaler** transforma los datos restando la media y dividiendo por la desviación estándar, resultando en una distribución con media 0 y desviación estándar 1.

```
13 # Aplicar Standard Scaling
14 standard_scaler = StandardScaler()
15 datos_standard = standard_scaler.fit_transform(datos)
16 print("Standard Scaling:\n", datos_standard)
```

*Ilustración 26 StandardScaler.*

```
Standard Scaling:
[[-1.34164079 -1.34164079]
 [-0.4472136  -0.4472136 ]
 [ 0.4472136   0.4472136 ]
 [ 1.34164079  1.34164079]]
```

*Ilustración 27 Salida StandardScaler.*

#### Explicación:

- La media de cada columna se centra en 0.
- La desviación estándar de cada columna es 1.