

# Introducción al Aprendizaje de Máquina

El aprendizaje de máquina o Machine Learning (ML) es una subdisciplina de la inteligencia artificial (IA) que permite a las computadoras detectar patrones en datos y hacer predicciones o tomar decisiones sin estar explícitamente programadas para cada tarea. En lugar de seguir reglas rígidas, un modelo de ML aprende a partir de la experiencia (datos) y mejora su desempeño con el tiempo.

Una plataforma como Netflix o YouTube utiliza ML para recomendar contenido personalizado. El sistema analiza tus hábitos de consumo, los compara con otros usuarios similares, y predice qué películas o videos podrían gustarte más.

**R** por Kibernetum Capacitación



# Explorando los Conceptos de Machine Learning

## Fuentes de Datos en Machine Learning

¿Recuerdas cómo definimos una fuente de datos en el módulo anterior sobre arquitectura y modelamiento? ¿Qué importancia tiene en un flujo de Machine Learning?

## Plataformas que Aprenden

¿Has interactuado con alguna plataforma que "aprenda" de tus decisiones (como Netflix, Spotify o Amazon)? ¿Cómo crees que lo hace?

## Tipos de Predicciones

¿Cuál crees que sea la diferencia entre que un modelo prediga un número (por ejemplo, ingresos mensuales) o una categoría (por ejemplo, tipo de cliente)?

# ¿Por qué es importante el aprendizaje de máquina?

## **Automatización inteligente**

Detección automática de fraudes bancarios y otros procesos que requieren análisis de patrones complejos sin intervención humana constante.

## **Mejora de procesos**

Optimización logística en tiempo real, permitiendo a las empresas mejorar su eficiencia operativa y reducir costos.

## **Experiencia personalizada**

Recomendaciones en e-commerce y plataformas de contenido, mejorando la experiencia del usuario y aumentando la conversión.

## **Descubrimiento de patrones**

Diagnóstico médico basado en imágenes, identificando indicadores sutiles que podrían pasar desapercibidos para el ojo humano.

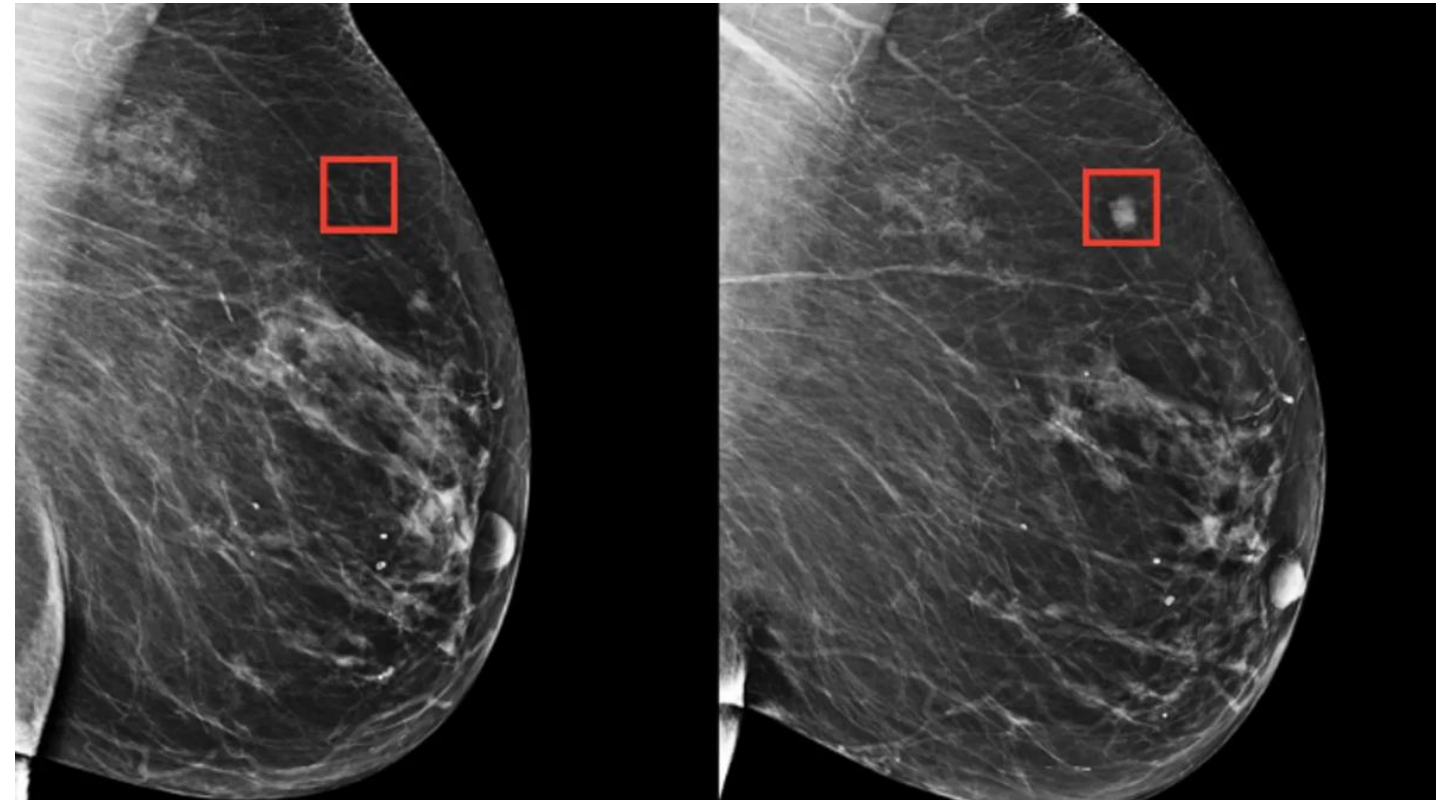




# Aplicaciones en medicina

El Machine Learning ha revolucionado el campo de la medicina, especialmente en el diagnóstico temprano. Los investigadores han desarrollado algoritmos capaces de aprender los sutiles patrones del tejido mamario que son precursores de tumores malignos, permitiendo detectar el cáncer en etapas más tempranas que los métodos tradicionales.

Estos sistemas pueden analizar miles de imágenes médicas en minutos, identificando anomalías con una precisión que complementa el trabajo de los especialistas humanos.



La detección temprana mediante algoritmos de ML puede aumentar significativamente las tasas de supervivencia, al permitir intervenciones médicas cuando el tratamiento es más efectivo.

# ¿Cómo funciona un modelo de ML?



## Entrada de datos

Recopilación de información desde diversas fuentes como sensores IoT, formularios digitales o redes sociales.



## Preprocesamiento

Transformación de datos crudos a un formato adecuado, eliminando valores nulos y normalizando variables.



## Entrenamiento

El algoritmo aprende patrones a partir de los datos procesados, ajustando sus parámetros internos.



## Evaluación

Medición del rendimiento del modelo utilizando métricas específicas según el tipo de problema.



## Predicción

Aplicación del modelo entrenado para generar resultados con nuevos datos.

## Flujo básico de un modelo de Machine Learning

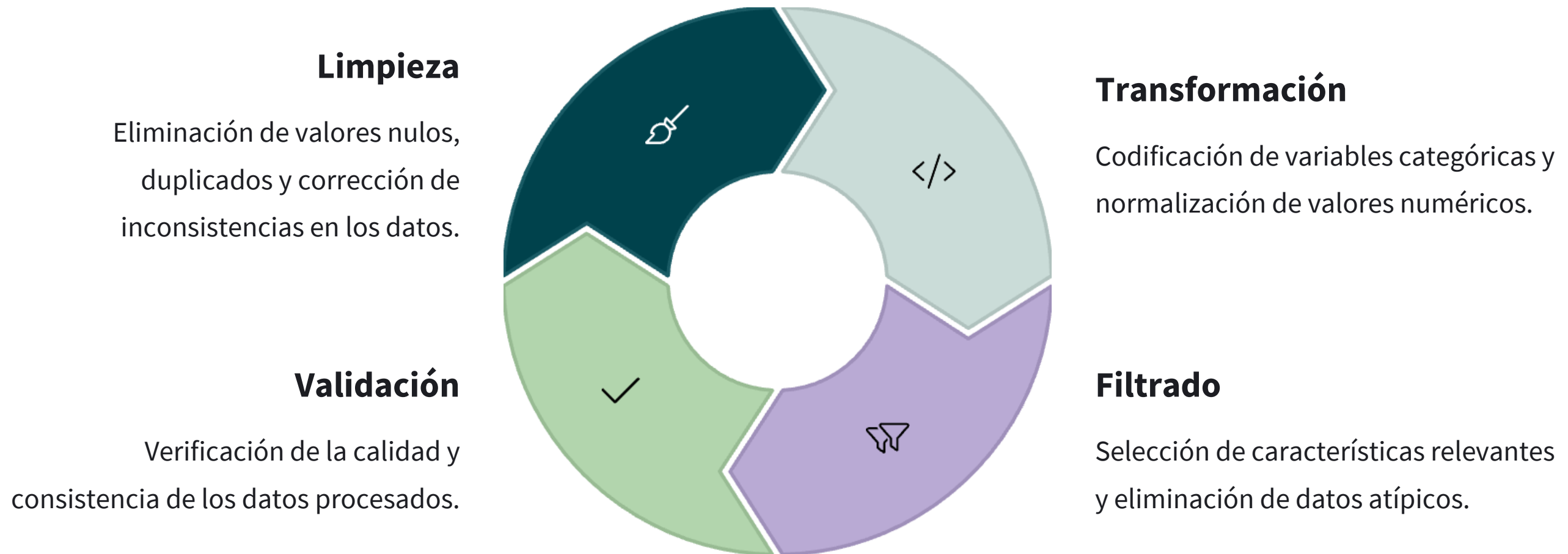


# Obtención de datos



Los datos son el insumo fundamental para cualquier sistema de análisis o decisión. Pueden originarse en diversas fuentes como sensores IoT, formularios digitales, redes sociales, registros (logs) de sistemas, aplicaciones empresariales, entre otros. Identificar correctamente estas fuentes es clave para diseñar una arquitectura eficiente y robusta.

# Preprocesamiento y limpieza de datos



Aunque parezca sorprendente, el preprocesamiento y la limpieza de datos pueden consumir hasta el 80% del tiempo total en un proyecto de ciencia de datos o Machine Learning. Este proceso es fundamental para garantizar resultados precisos y confiables.



# Entrenamiento del modelo



## Pronóstico del tiempo

Los modelos meteorológicos analizan patrones históricos y condiciones actuales para predecir el clima con días de anticipación.



## Recomendaciones de libros

Las librerías virtuales sugieren títulos basados en tus compras anteriores y preferencias de lectura.



## Análisis de criptomonedas

Aplicaciones de inversión utilizan modelos entrenados para predecir tendencias y sugerir decisiones financieras.

En esta etapa se selecciona un algoritmo (como regresión lineal, árboles de decisión o redes neuronales) y se entrena para que aprenda a reconocer patrones en los datos. Este entrenamiento permite que el modelo haga predicciones o clasificaciones basadas en ejemplos previos.



# Evaluación del modelo



## Precisión

Mide la proporción de predicciones positivas correctas, especialmente útil en problemas de clasificación binaria.



## Error cuadrático medio (MSE)

Calcula la media de los errores al cuadrado, penalizando más los errores grandes. Común en problemas de regresión.



## Área bajo la curva ROC (ROC-AUC)

Evalúa la capacidad del modelo para distinguir entre clases, independientemente del umbral de clasificación elegido.



## F1-Score

Combina precisión y exhaustividad, proporcionando una medida equilibrada para conjuntos de datos desbalanceados.



# Predicción y despliegue



## Desarrollo del modelo

Creación y refinamiento del algoritmo en entorno de laboratorio



## Conversión a API

Transformación del modelo en un servicio accesible



## Integración

Incorporación en aplicaciones o flujos de trabajo

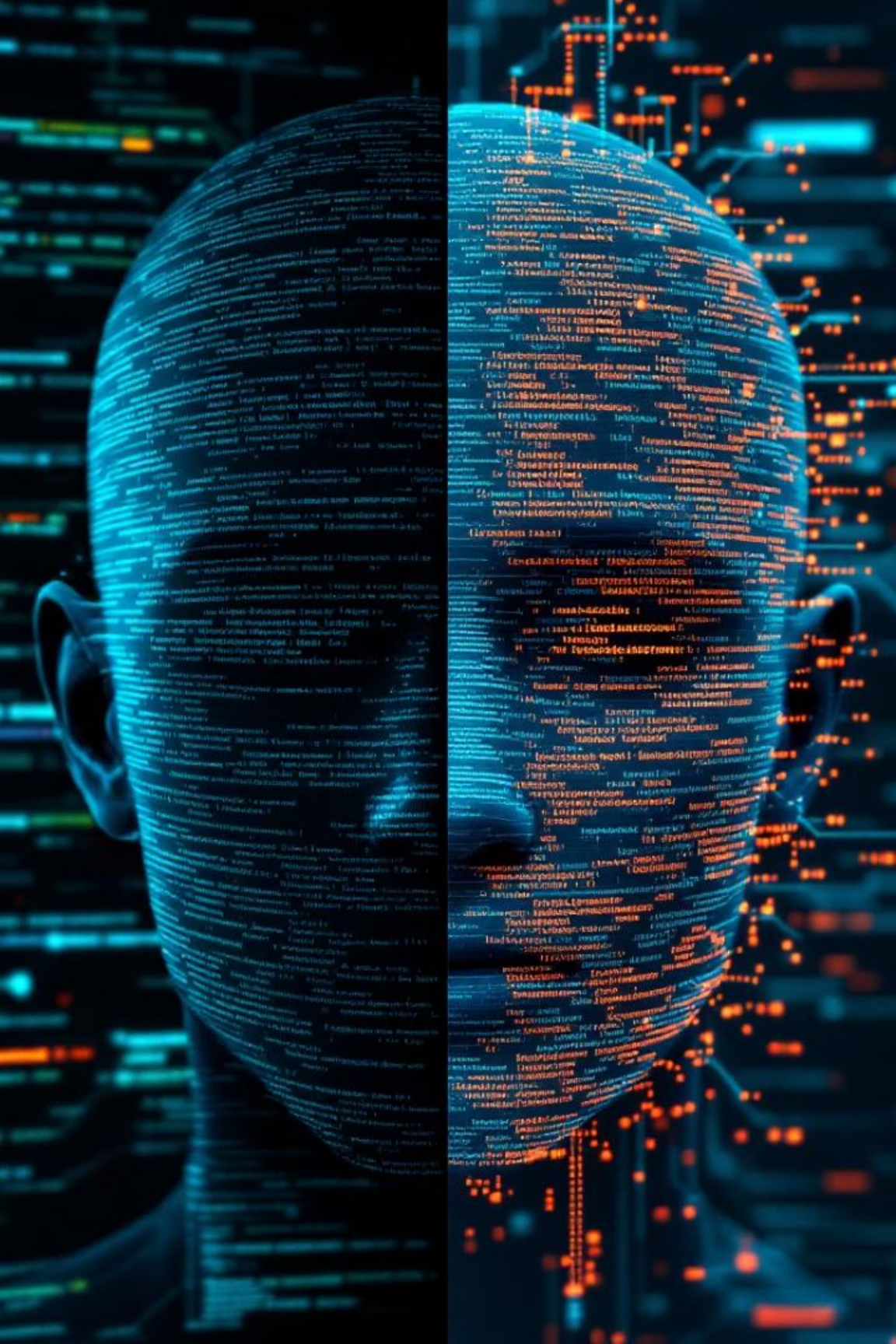


## Monitorización

Seguimiento continuo del rendimiento

En el caso de Netflix, el modelo entrenado se despliega a través de una API. Cada vez que un usuario inicia sesión, la aplicación llama a esta API, el modelo predice contenidos de interés y la plataforma muestra automáticamente esas recomendaciones en segundos.





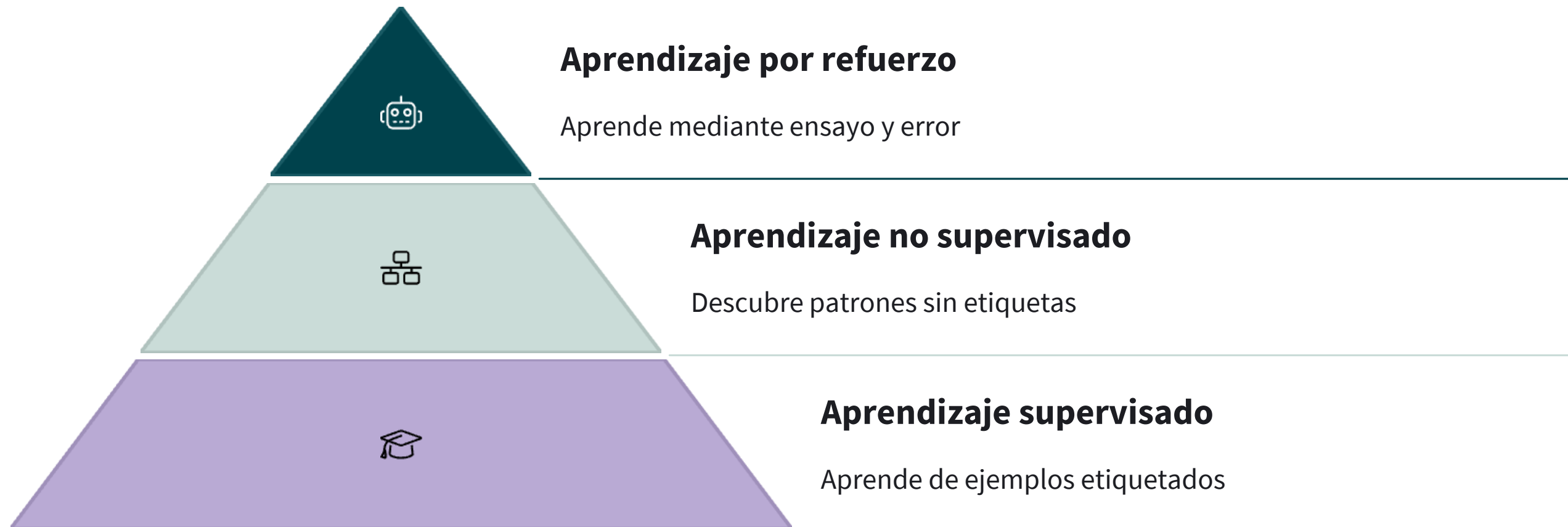
# Programación tradicional vs. Machine Learning

Característica	Programación Tradicional	Machine Learning
Entrada	Datos + Reglas explícitas	Datos + Resultados esperados
Salida	Resultado	Reglas implícitas (modelo aprendido)
Adaptabilidad	Baja	Alta (aprende y mejora)
Ejemplo	Calcular impuesto con fórmula	Predecir si un cliente se dará de baja

Para entender mejor qué hace único al Machine Learning, es útil compararlo con la programación tradicional. Aunque ambos enfoques utilizan datos como insumo, difieren profundamente en su lógica y funcionamiento. El aprendizaje automático permite que los sistemas aprendan patrones a partir de los datos, en lugar de seguir instrucciones rígidas predefinidas.



# Tipos de Aprendizaje de Máquina



Una vez comprendido qué es el aprendizaje automático y su relevancia, es fundamental conocer cómo se clasifica según la forma en que el modelo accede y aprende de los datos. Existen principalmente tres enfoques, siendo el aprendizaje supervisado y no supervisado los más comunes, mientras que el aprendizaje por refuerzo se utiliza en contextos específicos como robótica y juegos.

# Aprendizaje supervisado

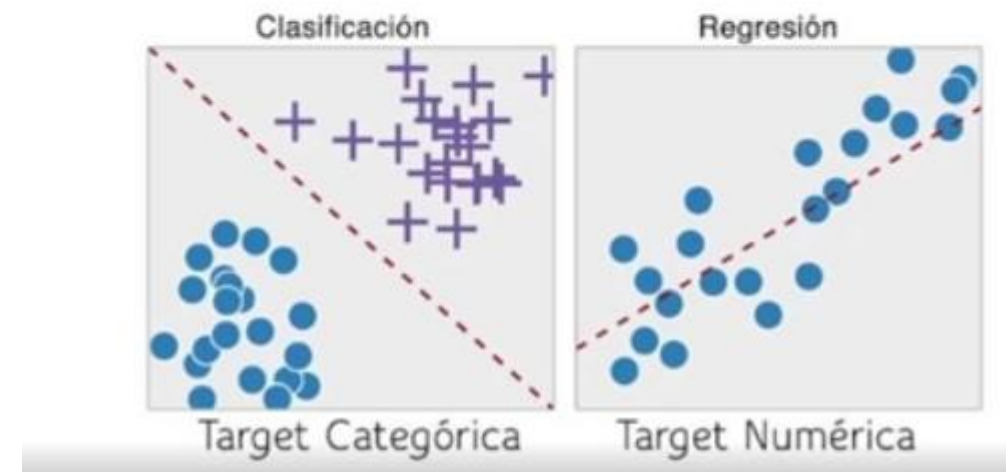
En este enfoque, el modelo aprende a partir de un conjunto de datos que incluye tanto las entradas como las salidas esperadas. Se le "supervisa" durante el entrenamiento, indicándole cuál es la respuesta correcta para cada caso.

Un ejemplo clásico es un modelo que predice si un correo es spam o no, utilizando correos etiquetados previamente como "spam" o "no spam". Las tareas más comunes en este tipo de aprendizaje son la clasificación y la regresión.

Este tipo de aprendizaje es central en el desarrollo de modelos que requieren evaluación cuantitativa, como sistemas de predicción de precios o diagnóstico médico automatizado.



Se conoce la variable target

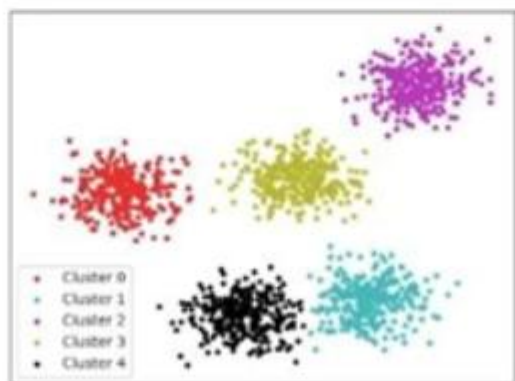


El aprendizaje supervisado requiere datos etiquetados donde cada ejemplo tiene una respuesta correcta conocida. El modelo aprende a generalizar a partir de estos ejemplos para hacer predicciones sobre nuevos datos no vistos anteriormente.

# Aprendizaje no supervisado



No se conoce la variable target



En el aprendizaje no supervisado, los datos no contienen etiquetas ni respuestas correctas. El objetivo del modelo es descubrir estructuras, patrones o agrupaciones naturales en los datos, sin intervención humana directa.

Un ejemplo común es agrupar clientes según sus hábitos de compra para personalizar campañas de marketing. Las tareas más frecuentes incluyen clustering (agrupamiento), reducción de dimensiones y detección de anomalías.

Este enfoque es muy útil en etapas exploratorias o cuando no se dispone de información categorizada, permitiendo descubrir insights que no eran evidentes inicialmente.

A diferencia del aprendizaje supervisado, en este enfoque los algoritmos trabajan con datos sin etiquetar, buscando descubrir estructuras ocultas por sí mismos.



# Comparativa de enfoques de aprendizaje

Característica	Aprendizaje Supervisado	Aprendizaje No Supervisado
Datos de entrada	Etiquetados (entrada + salida conocida)	No etiquetados (solo entrada)
Objetivo	Predecir un valor o clase	Encontrar estructura o patrones ocultos
Ejemplos típicos	Clasificación de correos, regresión de precios	Segmentación de clientes, detección de fraudes
Aplicaciones comunes	Predicción, detección de enfermedades	Agrupación, análisis exploratorio

Comprender las diferencias entre estos enfoques es fundamental para elegir la metodología adecuada según el problema a resolver y los datos disponibles. En las próximas sesiones trabajaremos con ejemplos prácticos tanto de clasificación como de regresión, que forman parte del aprendizaje supervisado.

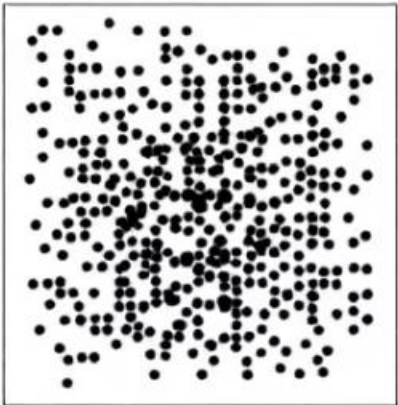
Supervised Learning



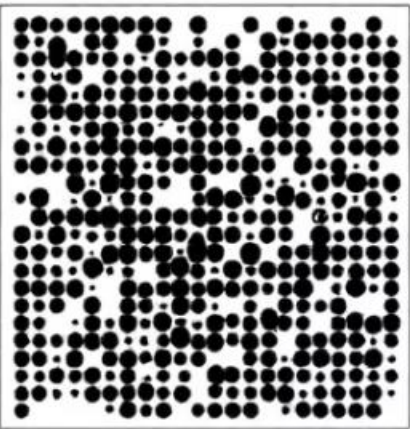
Supervised Learning



Unsupervised Learning



Unsupervised Learning



# Aprendizaje por refuerzo



## Observación del entorno

El agente percibe el estado actual del entorno en el que opera, recopilando información sobre su situación.



## Toma de decisiones

Basándose en su política actual, el agente selecciona y ejecuta una acción que modifica el entorno.



## Recepción de recompensa

El entorno proporciona una recompensa o penalización según el resultado de la acción realizada.



## Actualización de la política

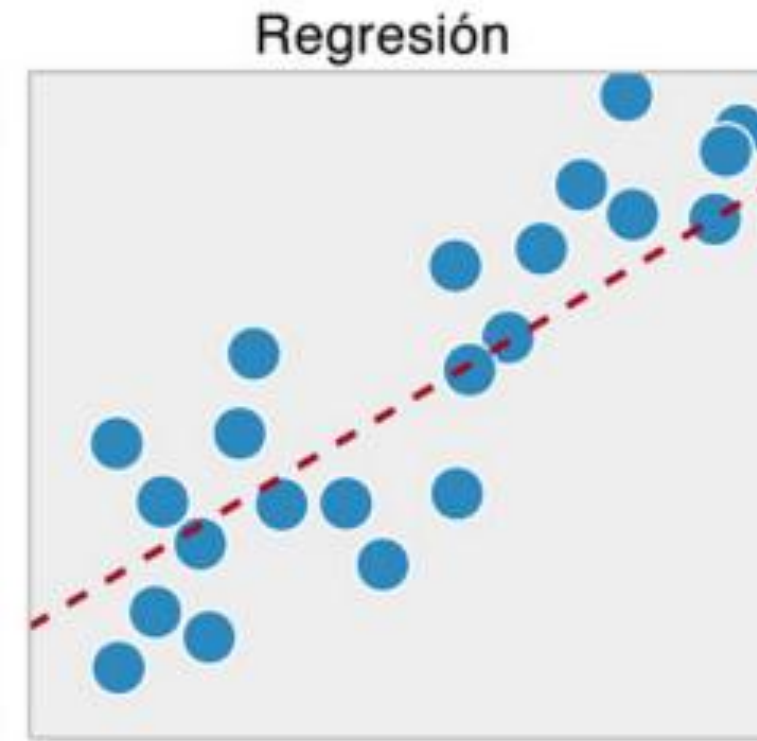
El agente ajusta su estrategia para maximizar las recompensas futuras, mejorando con cada iteración.

El aprendizaje por refuerzo es un tercer enfoque donde el modelo aprende a través de ensayo y error, recibiendo recompensas o penalizaciones en función de sus acciones. Es ampliamente utilizado en robótica, juegos y control de sistemas autónomos, como los vehículos de conducción autónoma o los robots que aprenden a caminar.

# Tareas de regresión

La regresión se utiliza cuando el objetivo es predecir un valor numérico continuo. Se aplica en situaciones donde queremos estimar cantidades como precios, temperaturas, tiempos o cualquier variable que varíe en una escala numérica.

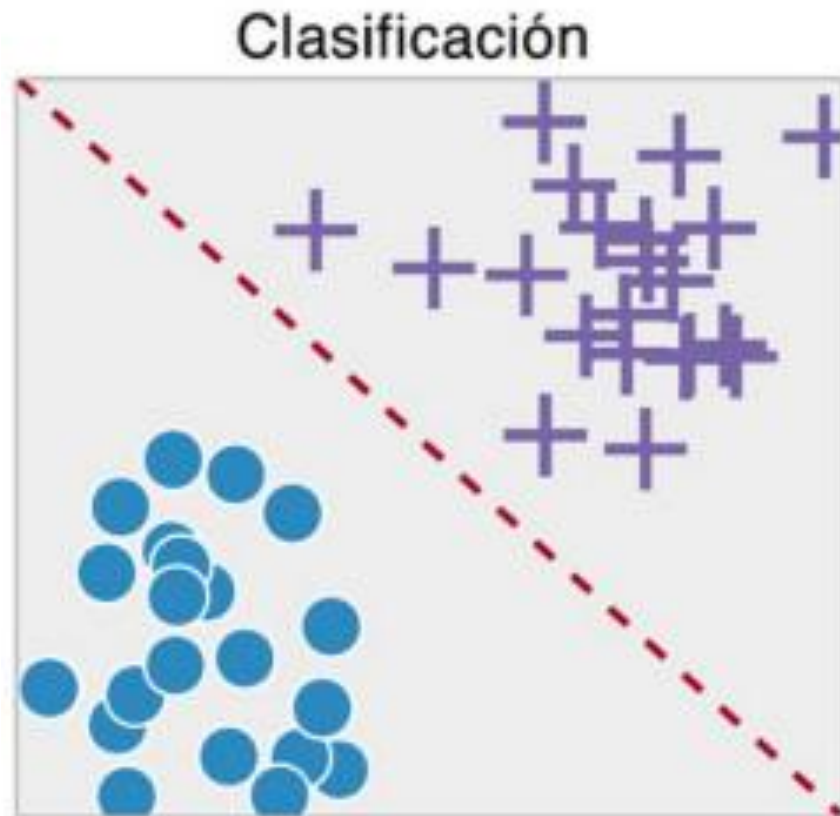
Un ejemplo típico sería predecir el precio de una casa a partir de características como los metros cuadrados, la ubicación, el número de habitaciones y baños, entre otros factores. Este tipo de tarea se aborda en profundidad en la Sesión 3, donde se exploran diferentes modelos de regresión.



Los modelos de regresión intentan encontrar la relación matemática entre las variables de entrada y la variable objetivo continua, permitiendo hacer predicciones precisas sobre nuevos datos.



# Tareas de clasificación



La clasificación se utiliza cuando el objetivo es predecir una categoría o clase. En lugar de entregar un valor numérico, el modelo selecciona una etiqueta entre un conjunto de posibles opciones.

Un ejemplo clásico es determinar si un correo electrónico es spam o no spam, basándose en su contenido, título y origen. Dependiendo del número de categorías posibles, podemos hablar de clasificación binaria (dos clases), multiclase (más de dos categorías) o multi-etiqueta (múltiples etiquetas por instancia).

Los modelos de clasificación aprenden a distinguir entre diferentes categorías, asignando nuevas observaciones a la clase más probable según los patrones aprendidos.

# Tipos de clasificación

## Clasificación binaria

Solo existen dos clases posibles, como en la detección de spam (spam/no spam) o en el diagnóstico médico (enfermo/sano). Los algoritmos están optimizados para encontrar un límite claro entre ambas categorías.

## Clasificación multiclase

Existen más de dos categorías posibles y se asigna solo una por instancia. Por ejemplo, clasificar tipos de flores (rosa, tulipán, margarita) o reconocer dígitos escritos a mano (0-9).

## Clasificación multi-etiqueta

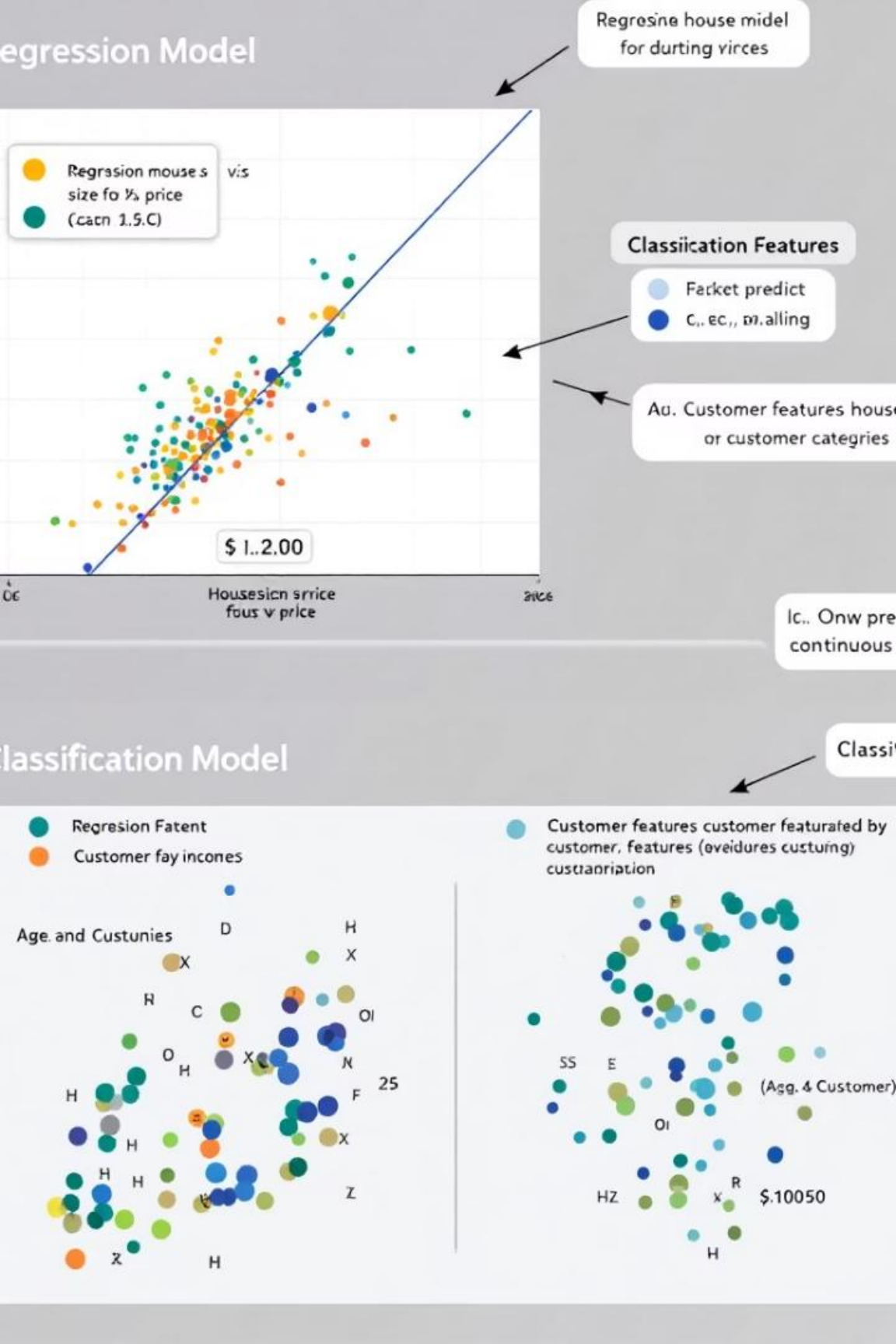
Se pueden asignar múltiples etiquetas a una misma instancia. Un caso típico es el etiquetado de imágenes, donde una foto puede clasificarse simultáneamente como "playa", "atardecer" y "vacaciones".

Cada tipo de clasificación requiere diferentes métricas y enfoques de evaluación, los cuales serán tratados en detalle durante la Sesión 4 del módulo.

# Comparativa: Regresión vs. Clasificación

Tipo de tarea	Objetivo principal	Salida esperada	Ejemplo típico
Regresión	Predecir un valor continuo	Número real (float)	Precio de una vivienda
Clasificación	Predecir una clase o categoría	Etiqueta/categoría	Tipo de cliente (fidel/nuevo)

Una buena forma de decidir qué tipo de tarea estás enfrentando es preguntarte: ¿La salida que espero es un número? → Regresión. ¿La salida que espero es una categoría? → Clasificación. En algunos casos, un mismo problema puede plantearse como regresión o clasificación, dependiendo de cómo se estructuren los datos.







# Preprocesamiento de datos

**80%**

## Tiempo dedicado

Porcentaje aproximado del tiempo total de un proyecto de ciencia de datos que se dedica al preprocesamiento.

**60%**

## Mejora de rendimiento

Aumento potencial en la precisión de un modelo gracias a un preprocesamiento adecuado.

**90%**

## Proyectos exitosos

Porcentaje de proyectos de ML donde la calidad del preprocesamiento fue factor decisivo en el éxito.

El preprocesamiento de datos es una de las etapas más importantes (y muchas veces subestimadas) en cualquier proyecto de Machine Learning. Incluso con modelos sofisticados, si los datos no han sido correctamente preparados, los resultados serán poco confiables o incluso erróneos.



	\$10,000	Customer Nam, \$0,055 Name
Data	\$1,000	Customer Names; \$5\$5,0050
All	1001	Normal Name
Puctamers	\$50,001	Customer: \$1,001 \$58
Addressdd	\$00,000	Customer \$110007

# ¿Por qué es necesario el preprocesamiento?



## Limitaciones de los modelos

Los modelos de ML no pueden trabajar directamente con datos crudos. Necesitan que la información esté limpia, estructurada y transformada numéricamente.



## Eficiencia computacional

Datos bien preparados reducen el tiempo de entrenamiento y los recursos computacionales necesarios para ejecutar los algoritmos.



## Mejora del rendimiento

Preprocesar adecuadamente puede marcar la diferencia entre un modelo mediocre y uno preciso, optimizando la capacidad predictiva.



## Reducción de errores

Elimina inconsistencias, valores atípicos y ruido que podrían llevar a conclusiones erróneas o modelos inestables.

# Codificación de variables categóricas

## Label Encoding

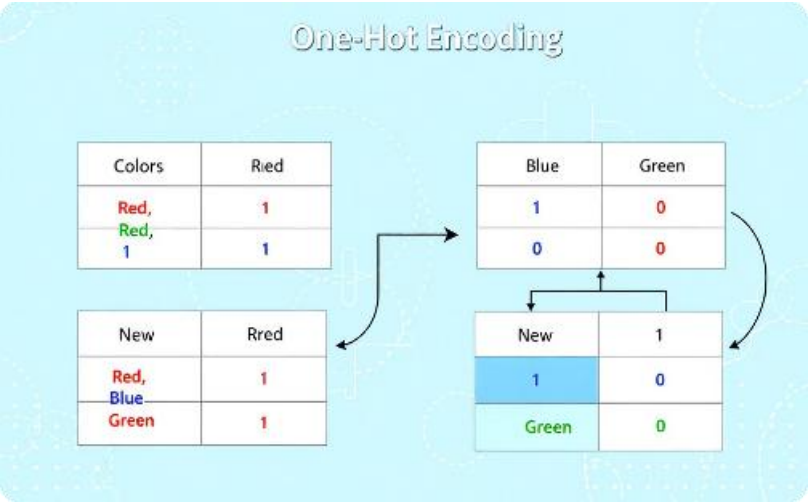
Original Labels	Encoded Labels
Red,	1
Green	2
Blue	(2)
Blue	3

## Label Encoding

Asigna un número entero a cada categoría. Útil para modelos basados en árboles de decisión, donde el orden numérico no distorsiona los resultados.

Las variables categóricas (como "rojo", "verde", "azul") deben transformarse a valores numéricos para que puedan ser utilizadas por los modelos. La elección de la técnica depende del algoritmo que se utilizará posteriormente.

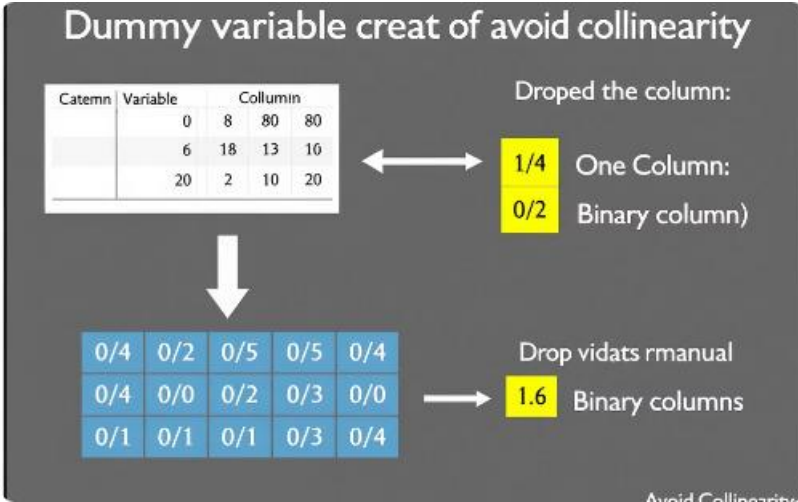
## One-Hot Encoding



## One-Hot Encoding

Crea una nueva columna binaria para cada categoría. Preferido en regresión lineal o redes neuronales, donde los modelos interpretan valores numéricos como jerárquicos.

## Dummy variable creat of avoid collinearity



## Variables Dummy

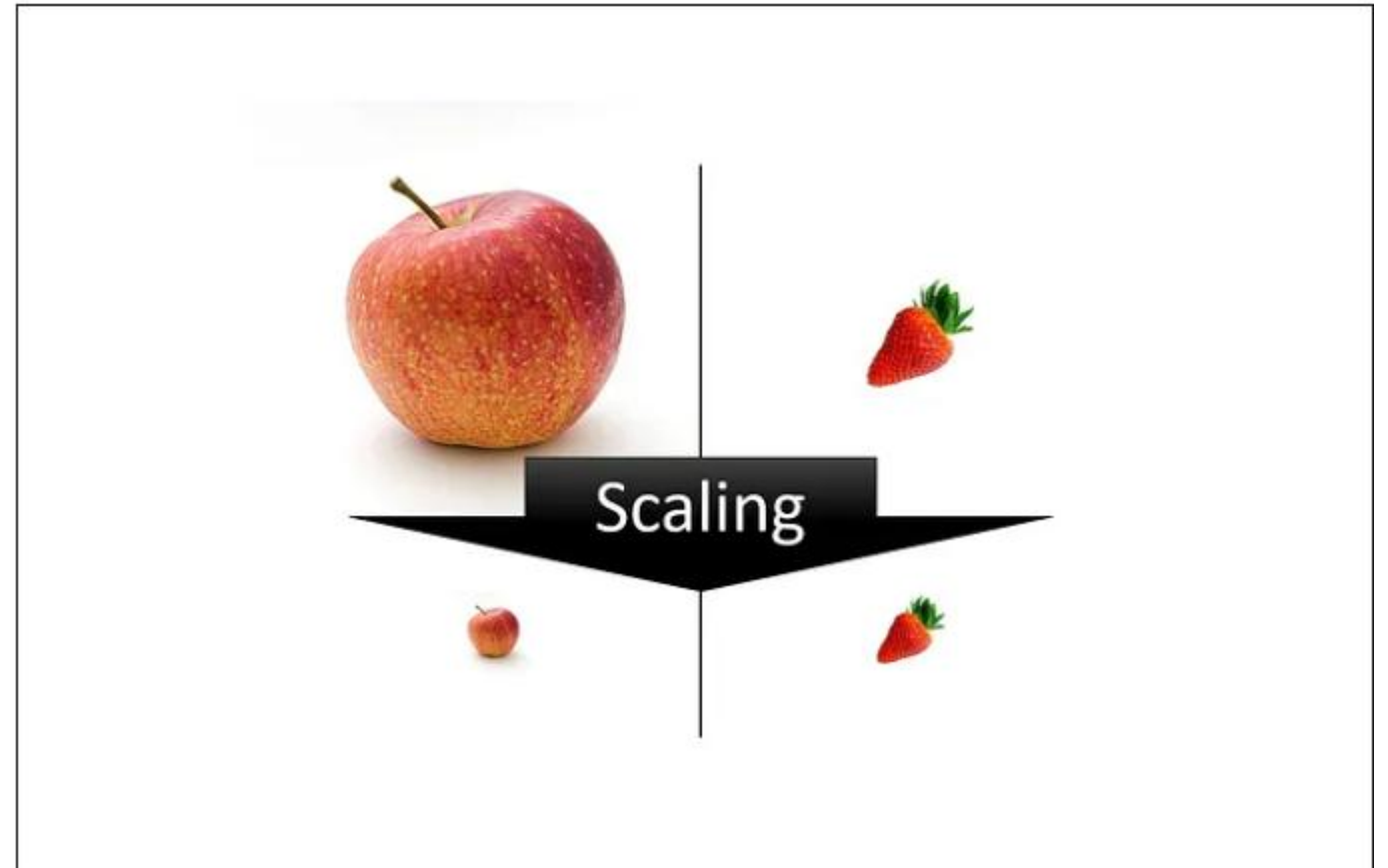
Similar al One-Hot, pero elimina una columna para evitar colinealidad. Implementado con `get_dummies(drop_first=True)` de pandas para análisis estadísticos.



# Escalado de datos

Cuando los datos numéricos están en escalas diferentes (por ejemplo, centímetros vs. grados), el modelo puede dar más importancia a una variable solo por tener números más grandes. Para evitar esto, se aplica escalado o normalización.

El escalado es especialmente importante en modelos que utilizan distancias (como KNN, clustering), que son sensibles a las magnitudes de las variables. Sin esta transformación, características con valores grandes dominarían el cálculo de similitud entre observaciones.



A la izquierda vemos una manzana y una frutilla en sus tamaños reales. Si un algoritmo comparara estas imágenes por tamaño, daría más peso a la manzana simplemente por su escala. Tras aplicar scaling, ambas frutas aparecen en proporciones comparables, permitiendo al modelo tratarlas con equidad.



# Técnicas de escalado

## MinMax Scaling

Escala los valores a un rango fijo, generalmente  $[0, 1]$ . La fórmula es:  $X_{\text{scaled}} = (X - X_{\text{min}}) / (X_{\text{max}} - X_{\text{min}})$ . Preserva la distribución original pero comprimida en el nuevo rango.

Ideal cuando conocemos los límites exactos de nuestros datos y queremos mantener las relaciones entre valores.

## Estandarización (Z-score)

Centra los datos en media 0 con desviación estándar 1. La fórmula es:  $X_{\text{scaled}} = (X - \mu) / \sigma$ . Útil cuando no conocemos la distribución o rango de los datos.

Recomendado para algoritmos que asumen distribución normal, como regresión logística o SVM.

## Robust Scaling

Similar a MinMax pero usa la mediana y rango intercuartílico en lugar de mínimo y máximo. Menos sensible a valores atípicos extremos que podrían distorsionar el escalado.

Preferible cuando los datos contienen outliers significativos que no queremos eliminar.

# Actividad Práctica Guiada: Aplicando Transformaciones básicas en el iris dataset

## Objetivo

Aplicar paso a paso las técnicas básicas de **preprocesamiento** sobre un **dataset real**, observando cómo **cambian los datos** y **reflexionando** sobre su **importancia** antes de entrenar modelos.

Antes de iniciar el ejercicio guiado, exploremos juntos esta librería de Python llamada **scikit-learn**.

## ¿Qué es scikit-learn?

**scikit-learn** (abreviado como sklearn) es una de las librerías más utilizadas en Python para construir modelos de Machine Learning. Ofrece herramientas para tareas de **preprocesamiento**, **entrenamiento de modelos**, **evaluación** y **selección de características**, entre otras.

Una de sus grandes ventajas es que permite aplicar modelos y transformaciones con pocas líneas de código, manteniendo una estructura uniforme y clara.

En este caso, utilizaremos su clase **LabelEncoder** para transformar etiquetas de texto en valores numéricos.

# Actividad Práctica Guiada: Aplicando Transformaciones básicas en el iris dataset

## Lo más útil de scikit-learn para principiantes

**scikit-learn** es una librería **todo-en-uno** para tareas comunes de **Machine Learning** en Python.

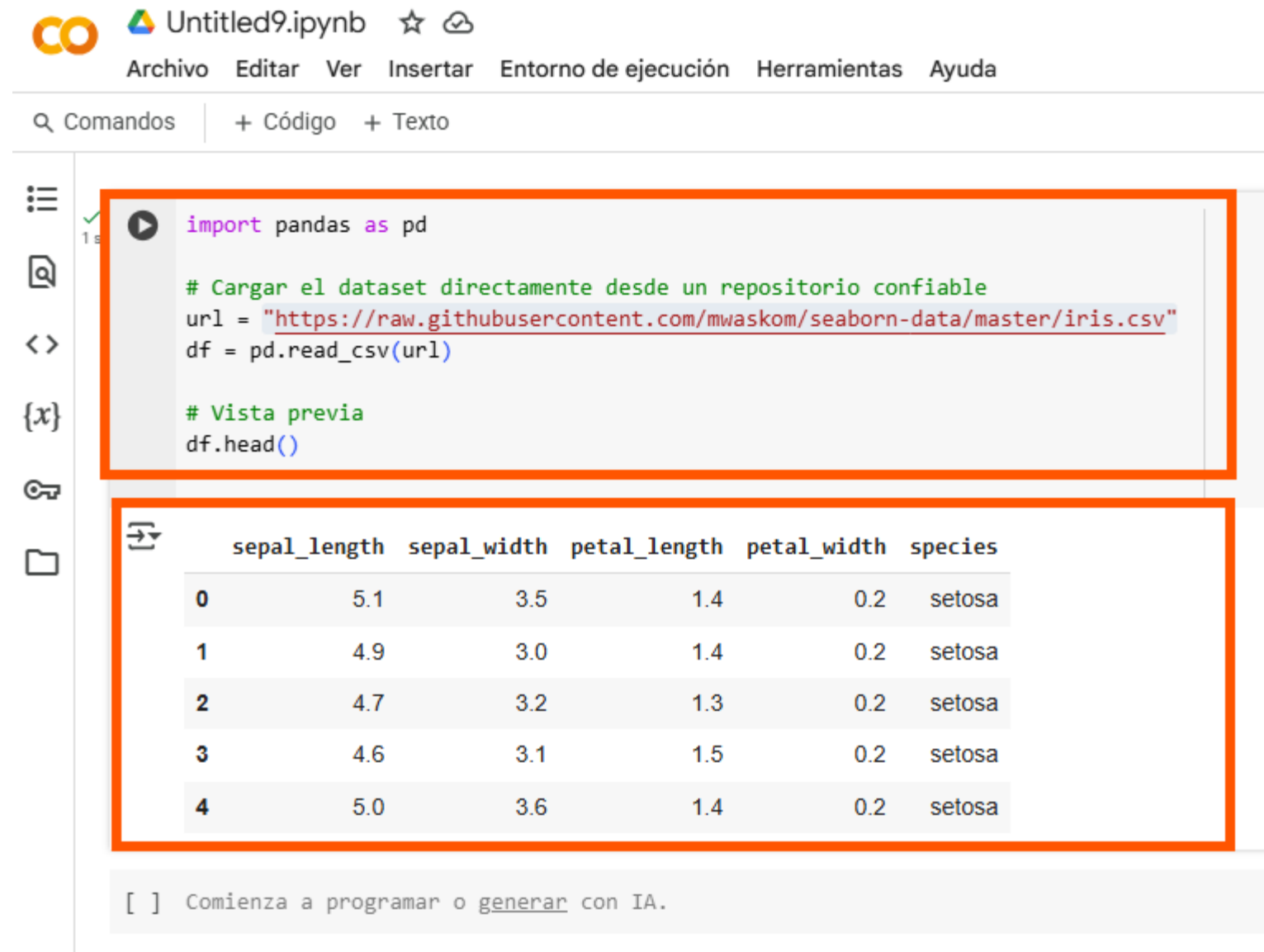
Estas son algunas de sus funcionalidades más utilizadas:

Funcionalidad	Ejemplo de uso	Clase o función
Preprocesamiento	Escalar datos, variables codificar	LabelEncoder, StandardScaler, OneHotEncoder
Modelos supervisados	Clasificación y regresión	LogisticRegression, KNeighborsClassifier, LinearRegression
Evaluación de modelos	Medir desempeño con métricas	accuracy_score, confusion_matrix, mean_squared_error
Validación cruzada	Evaluar modelos con particiones de datos	train_test_split, cross_val_score
Pipelines	Encadenar procesos en un solo flujo	Pipeline

# Actividad Práctica Guiada: Aplicando Transformaciones básicas en el iris dataset

## Paso a Paso

### 1.- Cargar el dataset



The screenshot shows a Jupyter Notebook titled "Untitled9.ipynb". The interface includes a top menu bar with options: Archivo, Editar, Ver, Insertar, Entorno de ejecución, Herramientas, and Ayuda. Below the menu is a toolbar with icons for Comandos, + Código, and + Texto. The notebook content is divided into two cells, both highlighted with orange borders.

The first cell is a code cell containing the following Python code:

```
import pandas as pd

# Cargar el dataset directamente desde un repositorio confiable
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv"
df = pd.read_csv(url)

# Vista previa
df.head()
```

The second cell is a table cell displaying the first five rows of the dataset. The table has the following structure:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

At the bottom of the notebook interface, there is a status bar that reads: [ ] Comienza a programar o generar con IA.



# Actividad Práctica Guiada: Aplicando Transformaciones básicas en el iris dataset

## Paso a Paso

2.- Codificación de la variable species: Usamos LabelEncorder para convertir las especies de flores (clases categóricas) en valores numéricos que pueden ser interpretados por un modelo de aprendizaje automático.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Cargar el dataset directamente desde un repositorio confiable
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv"
df = pd.read_csv(url)

# Vista previa
df.head()
```

```
# Creamos una instancia del codificador y aplicamos sobre la columna 'species'
encoder = LabelEncoder()
df['species_encoded'] = encoder.fit_transform(df['species'])

# Mostramos una muestra aleatoria de 5 filas
print("Muestra aleatoria de 5 filas:")
display(df.sample(n=5, random_state=42))
```

Muestra aleatoria de 5 filas:

	sepal_length	sepal_width	petal_length	petal_width	species	species_encoded
73	6.1	2.8	4.7	1.2	versicolor	1
18	5.7	3.8	1.7	0.3	setosa	0
118	7.7	2.6	6.9	2.3	virginica	2
78	6.0	2.9	4.5	1.5	versicolor	1
76	6.8	2.8	4.8	1.4	versicolor	1

```
# Mostramos un ejemplo por cada clase para verificar la codificación
print("\n Una fila representativa de cada clase:")
for especie in df['species'].unique():
    muestra = df[df['species'] == especie].sample(1)
    display(muestra[['species', 'species_encoded']])
```

Una fila representativa de cada clase:

	species	species_encoded
25	setosa	0
	species	species_encoded
76	versicolor	1
	species	species_encoded
145	virginica	2

# Actividad Práctica Guiada: Aplicando Transformaciones básicas en el iris dataset

## Paso a Paso

### 3.- Escalado de variables numéricas: Aplicamos MinMaxScaler y StandardScaler a las variables continuas

#### Importación de librerías

```
[24] # Importar escaladores desde Scikit-learn
      from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

#### Definir las variables a escalar

```
[25] # Definir las características numéricas del dataset Iris
      features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
      print("Variables a escalar:", features)
```

Variables a escalar: ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']

#### Aplicar MinMaxScaler

```
[27] # Escalado MinMax (valores entre 0 y 1)
      print("\n Aplicando MinMaxScaler (rango [0, 1])...")

      minmax = MinMaxScaler()
      df_minmax = pd.DataFrame(minmax.fit_transform(df[features]), columns=features)

      # Vista previa
      print(" Vista previa de los datos escalados con MinMax:")
      print(df_minmax.head())
```



Aplicando MinMaxScaler (rango [0, 1])...  
Vista previa de los datos escalados con MinMax:

	sepal_length	sepal_width	petal_length	petal_width
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667

#### Aplicar StandardScaler (Z-score)

```
[28] # Escalado Z-score (media 0, desviación estándar 1)
      print("\n Aplicando StandardScaler (Z-score)...")

      standard = StandardScaler()
      df_standard = pd.DataFrame(standard.fit_transform(df[features]), columns=features)

      # Vista previa
      print("Vista previa de los datos escalados con Z-score:")
      print(df_standard.head())
```



Aplicando StandardScaler (Z-score)...  
Vista previa de los datos escalados con Z-score:

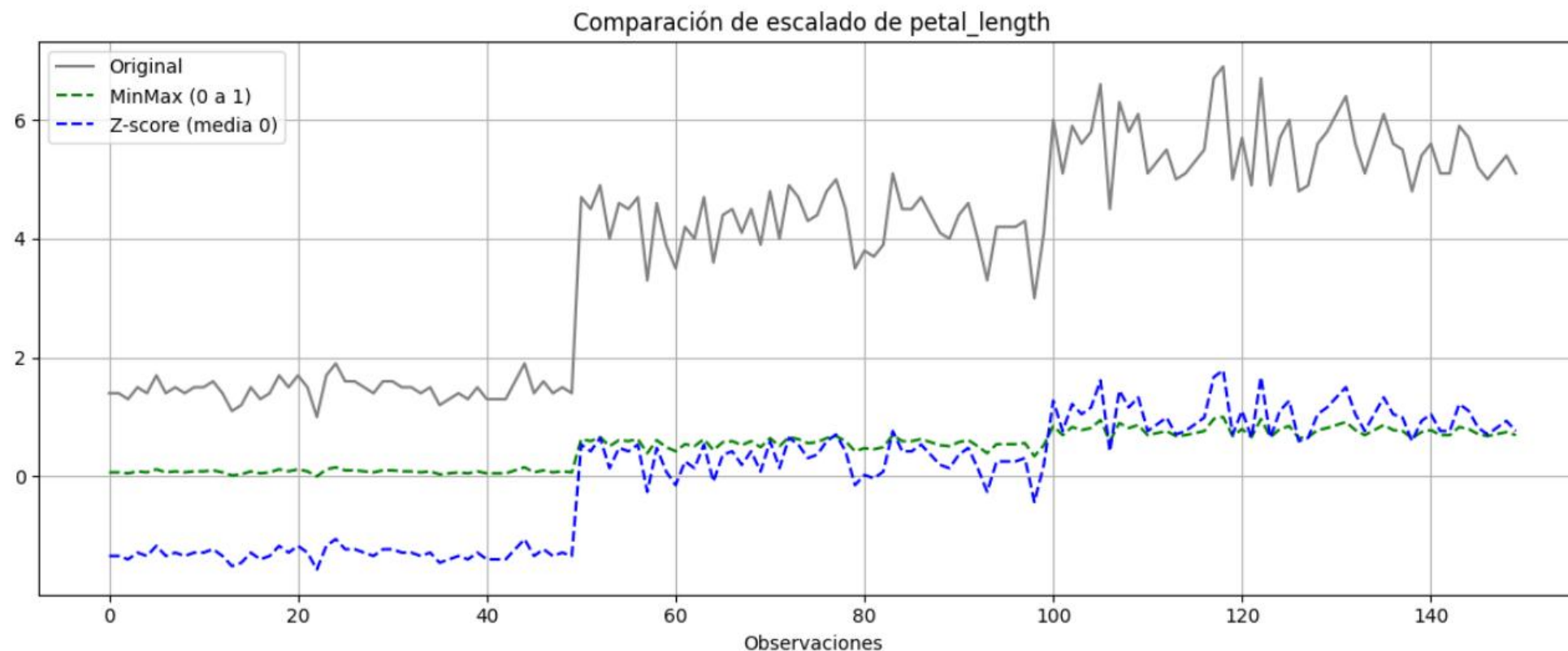
	sepal_length	sepal_width	petal_length	petal_width
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444

# Actividad Práctica Guiada: Aplicando Transformaciones básicas en el iris dataset

## Reflexionemos:

1. ¿Qué diferencias observas entre los valores originales y los escalados?
2. ¿Qué técnica preferirías si tus datos tuvieran valores extremos?
3. ¿Tiene sentido escalar la columna species\_encoded? ¿Por qué?

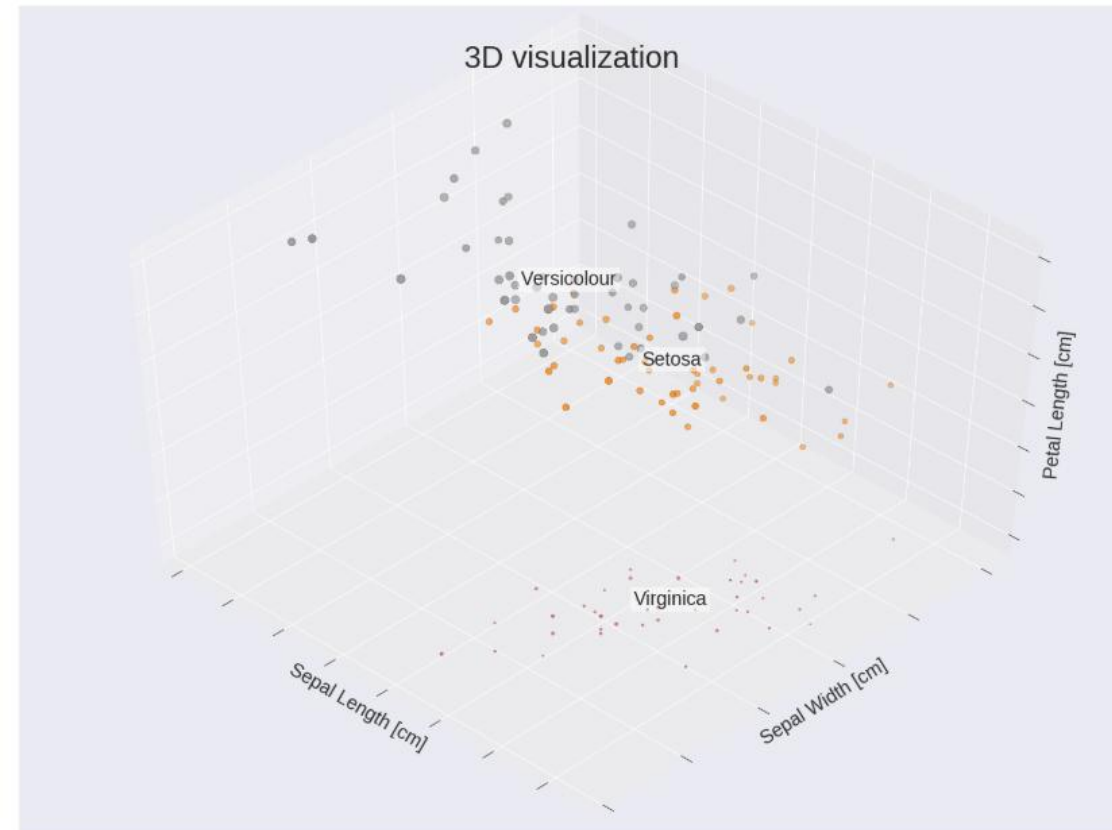
Recuerden que los modelos **no ‘piensan’**. No saben que un valor **100** puede ser simplemente una **unidad diferente**, o que **‘setosa’** no es más grande que **‘versicolor’**. Por eso el **preprocesamiento no es un paso opcional**: es la forma en que **‘preparamos el terreno’** para que los **algoritmos** trabajen con **lógica y coherencia.**”



# Concepto de Distancia en Machine Learning

Algunos algoritmos de Machine Learning, como k-Nearest Neighbors (KNN) o los métodos de clustering, necesitan una forma de "medir la cercanía" entre los datos. Para esto utilizan métricas de distancia, que permiten comparar cuánto se parecen o se diferencian dos puntos del espacio de características.

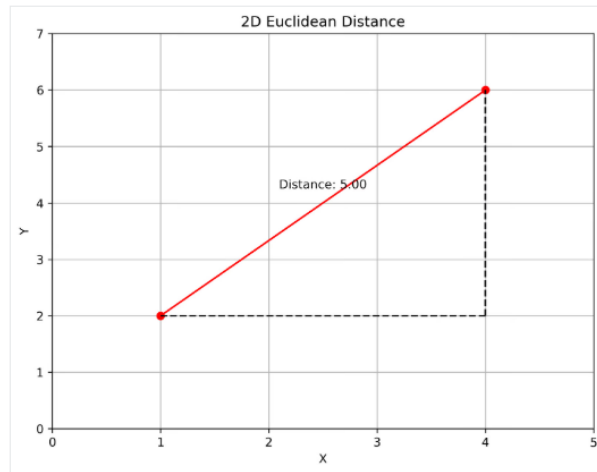
En un espacio de datos, cada observación puede verse como un punto definido por sus características numéricas. Por ejemplo, una flor del dataset Iris puede representarse como un punto en un espacio de 4 dimensiones: largo del sépalo, ancho del sépalo, largo del pétalo y ancho del pétalo.



Cada punto en el gráfico representa una flor, posicionada según sus medidas. Esta vista en 3D nos permite imaginar el espacio de características donde se agrupan las clases, y entender cómo la distancia entre puntos refleja la similitud entre observaciones.



# Tipos de distancia más comunes



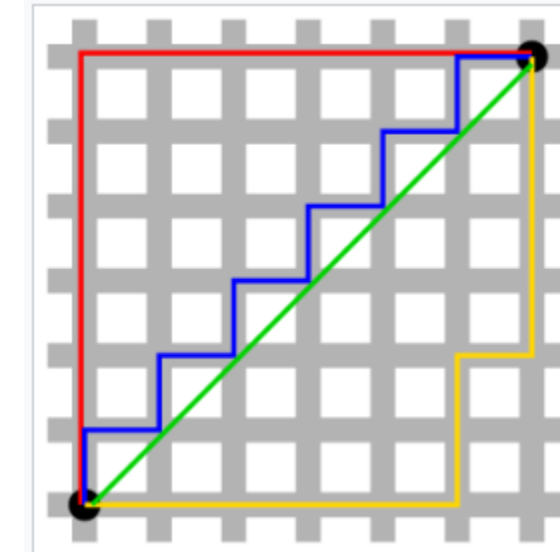
## Distancia Euclidiana

Es la distancia "en línea recta" entre dos puntos, calculada con la fórmula:  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ . Representa el camino más corto entre dos puntos en un espacio euclidiano.



## Distancia Manhattan

Suma de las diferencias absolutas entre coordenadas:  $|x_1 - x_2| + |y_1 - y_2|$ . Simula el movimiento en calles rectas tipo cuadrícula, como en una ciudad con bloques rectangulares.



## Comparación visual

La línea verde representa la distancia euclidiana (camino recto), mientras que las rutas en rojo, azul y amarillo muestran diferentes recorridos que corresponden a la distancia de Manhattan.

La elección de la métrica de distancia puede afectar significativamente el comportamiento de algoritmos como KNN o K-means, por lo que debe seleccionarse considerando la naturaleza del problema y la distribución de los datos.

# Actividad Práctica Guiada: Visualizando Distancias en 2D

## Objetivo

Simular dos puntos en un plano 2D y calcular la distancia entre ellos usando ambas métricas: Euclidiana y Manhattan.

### ✓ Celda 1: Importación de librerías

```
✓  
0s [1] # Importamos NumPy para los cálculos y Matplotlib para visualizar  
import numpy as np  
import matplotlib.pyplot as plt
```

# Actividad Práctica Guiada: Visualizando Distancias en 2D

## Objetivo

Simular dos puntos en un plano 2D y calcular la distancia entre ellos usando ambas métricas: Euclidiana y Manhattan.

### ✓ Celda 2: Definir los puntos A y B

En este paso definiremos dos puntos A y B en un plano bidimensional. Estos puntos representarán dos observaciones, y nos servirán para calcular las distancias entre ellos usando distintas métricas.

```
[9] # Definimos dos puntos en un plano 2D
    # Punto A en (2, 3) y punto B en (7, 6)
    A = np.array([2, 3])
    B = np.array([7, 6])

    print("A: ", A)
    print("B: ", B)
```

```
⇒ A: [2 3]
   B: [7 6]
```

# Actividad Práctica Guiada: Visualizando Distancias en 2D

## Objetivo

Simular dos puntos en un plano 2D y calcular la distancia entre ellos usando ambas métricas: Euclidiana y Manhattan.

### ✓ Celda 3: Cálculo de distancias

Una vez definidos los puntos, calculamos dos tipos de distancia:

- **Euclidiana:** la línea recta más corta entre A y B.
- **Manhattan:** la suma de las distancias horizontales y verticales entre A y B (como si camináramos por una ciudad en cuadrícula).

Esto nos permitirá comparar cómo varía la noción de “distancia” según la métrica utilizada.

```
[11] # Calculamos la distancia euclidiana (línea recta entre A y B)
      euclidean = np.linalg.norm(A - B)

      # Calculamos la distancia Manhattan (camino en forma de "L")
      manhattan = np.sum(np.abs(A - B))

      # Mostramos los resultados
      print("Distancia Euclidiana:", euclidean)
      print("Distancia Manhattan:", manhattan)
```

```
➡ Distancia Euclidiana: 5.830951894845301
   Distancia Manhattan: 8
```



# Actividad Práctica Guiada: Visualizando Distancias en 2D

## Objetivo

Simular dos puntos en un plano 2D y calcular la distancia entre ellos usando ambas métricas: Euclidiana y Manhattan.

### ✓ Celda 4: Visualización de las distancias

Finalmente, representamos los puntos y las dos distancias en un gráfico:

- La línea punteada muestra la **distancia euclidiana**.
- El trazo en forma de "L" representa la **distancia Manhattan**.

Esta visualización nos ayudará a entender intuitivamente la diferencia entre ambas medidas en el espacio.

```
# Creamos el gráfico para comparar ambas distancias
plt.figure(figsize=(6,6))

# Dibujamos los puntos A y B
plt.scatter(*A, color='blue', label='Punto A')
plt.scatter(*B, color='red', label='Punto B')

# Línea punteada (Euclidiana)
plt.plot([A[0], B[0]], [A[1], B[1]], 'k--', label='Euclidiana')

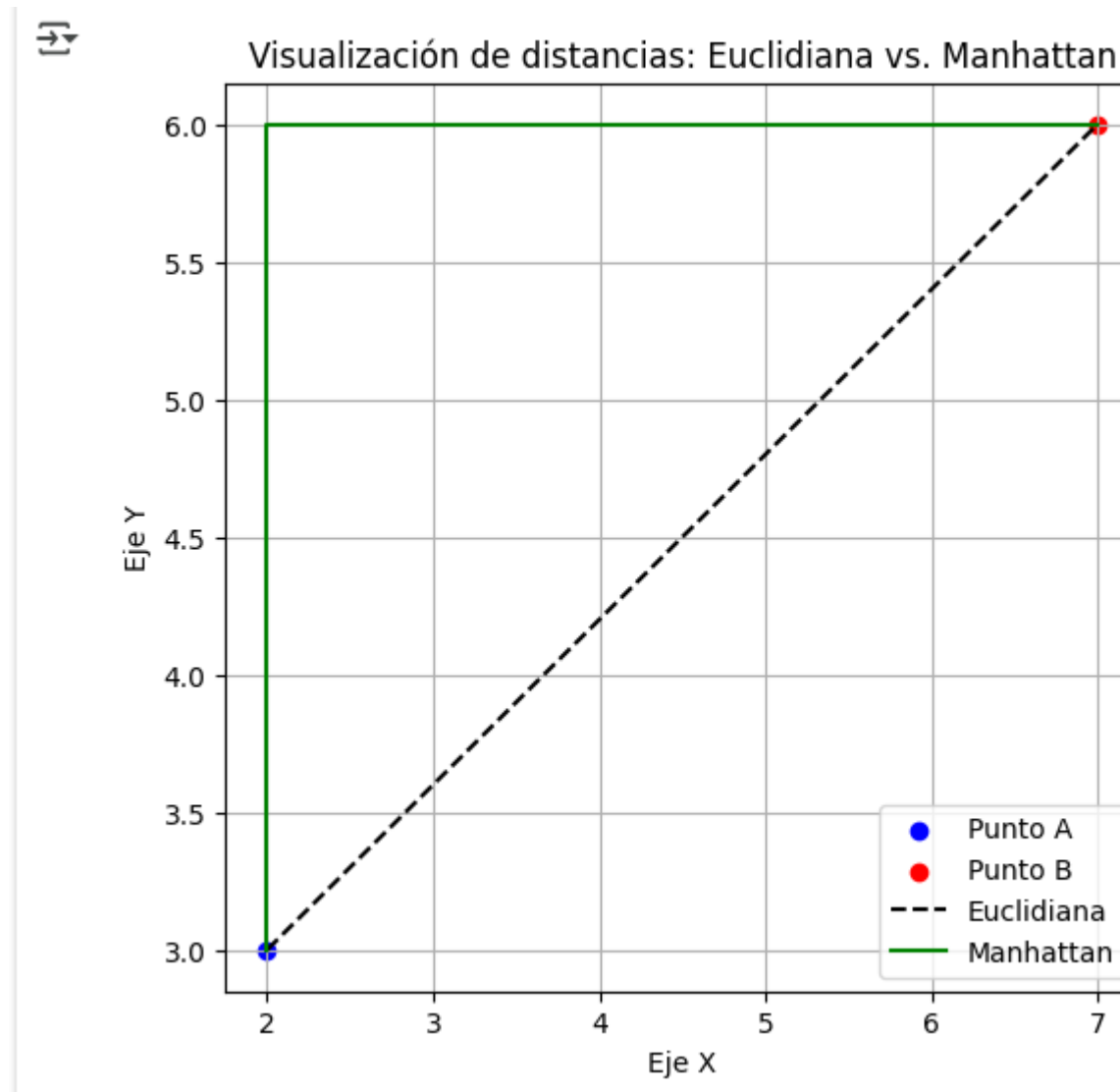
# Recorrido en forma de L (Manhattan)
plt.plot([A[0], A[0], B[0]], [A[1], B[1], B[1]], 'g-', label='Manhattan')

# Detalles del gráfico
plt.legend()
plt.grid(True)
plt.title("Visualización de distancias: Euclidiana vs. Manhattan")
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.show()
```

# Actividad Práctica Guiada: Visualizando Distancias en 2D

## Objetivo

Simular dos puntos en un plano 2D y calcular la distancia entre ellos usando ambas métricas: Euclidiana y Manhattan.



# Actividad Práctica Guiada: Visualizando Distancias en 2D

## Objetivo

Simular dos puntos en un plano 2D y calcular la distancia entre ellos usando ambas métricas: Euclidiana y Manhattan.

1. ¿Qué camino parece más corto? ¿Y cuál es más realista si tienes restricciones como calles en ángulo recto?
2. ¿Cuál distancia se usaría en un videojuego? ¿Y en un GPS de ciudad?
3. ¿Qué pasaría si X y Y tuvieran escalas diferentes (por ejemplo, cm y kilómetros)?



# Dimensiones y la Maldición de la Dimensionalidad

**1 m** Espacio 1D

Volumen necesario para cubrir una línea de 1 metro.

**1 m<sup>2</sup>** Espacio 2D

Volumen para cubrir un cuadrado de 1x1 metros.

**1 m<sup>3</sup>** Espacio 3D

Volumen para cubrir un cubo de 1x1x1 metros.

**10<sup>9</sup> m<sup>10</sup>** Espacio 10D

Volumen necesario en un espacio de 10 dimensiones.

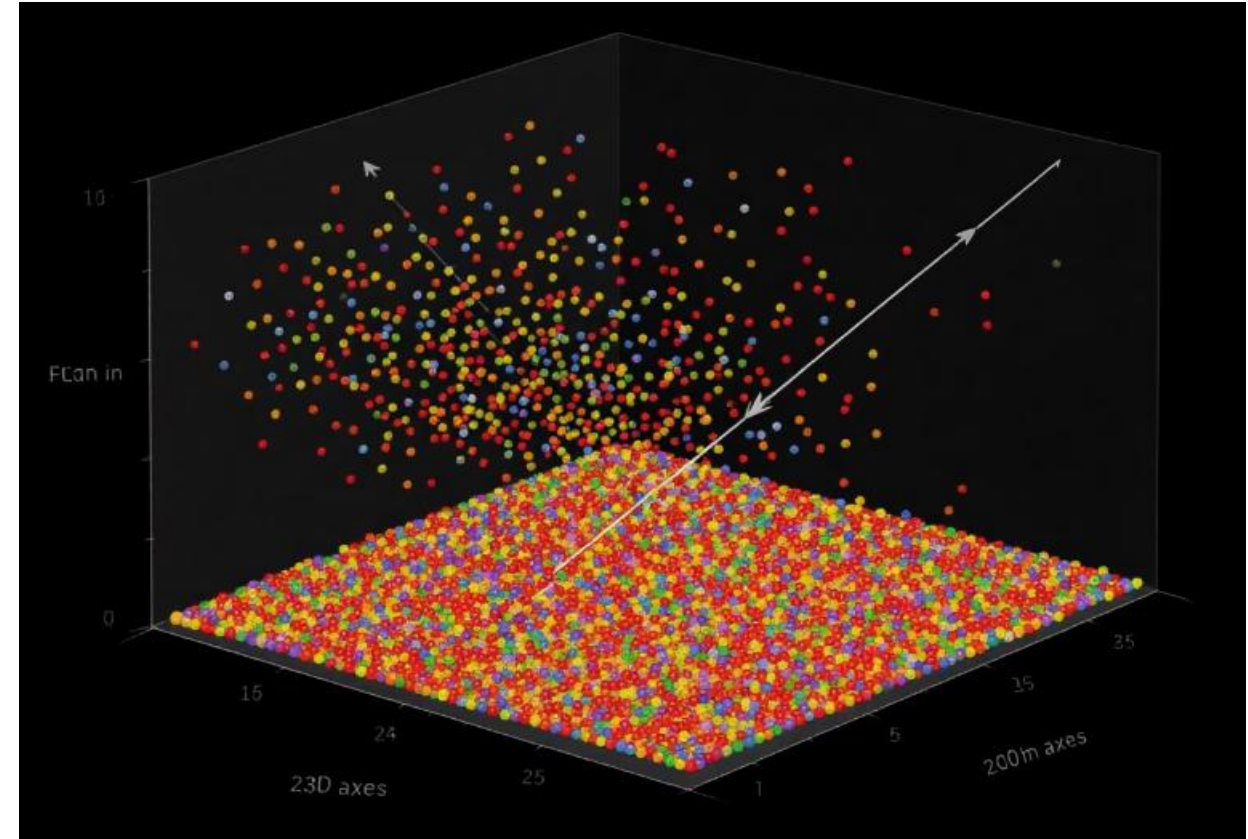
En Machine Learning, cada variable o característica de un dataset representa una dimensión en el espacio de datos. A medida que aumentan las dimensiones, los datos se dispersan en el espacio: hay más combinaciones posibles de valores, pero la densidad de datos se diluye, dificultando el aprendizaje de patrones.



# Reducción de dimensionalidad

La solución a la maldición de la dimensionalidad no es eliminar variables al azar, sino aplicar técnicas que reduzcan la dimensionalidad de forma inteligente, conservando la información relevante. Estas técnicas permiten visualizar datos complejos, mejorar el rendimiento de los modelos y reducir el ruido.

El Análisis de Componentes Principales (PCA) es una de las técnicas más utilizadas. Proyecta los datos en un nuevo espacio con menos dimensiones, manteniendo la mayor varianza posible. Es especialmente útil para exploración, visualización y preprocesamiento antes de aplicar algoritmos de clustering.



PCA transforma un conjunto de variables correlacionadas en un conjunto menor de variables no correlacionadas llamadas componentes principales, que capturan la mayor parte de la variabilidad en los datos originales.

# Actividad Práctica Guiada: Visualización con Reducción de dimensiones (PCA)

Cuando trabajamos con datos multivariantes —como en el caso del dataset Iris, que incluye medidas de largo y ancho de sépalo y pétalo— puede ser difícil identificar patrones visuales porque no podemos "ver" en más de 3 dimensiones.

Aquí es donde entra en juego **PCA (Análisis de Componentes Principales)**, una técnica de reducción de dimensionalidad que nos permite proyectar datos de alta dimensión en un espacio de 2 o 3 dimensiones, **manteniendo la mayor cantidad de información posible**.

Reducir dimensiones no solo ayuda a visualizar los datos, sino también a **detectar relaciones ocultas**, agrupar observaciones similares y **mejorar el rendimiento de algunos algoritmos de Machine Learning**.

## Objetivo:

- Aplicar la técnica de **PCA** al dataset Iris para:
- Reducir de 4 a 2 dimensiones sus variables numéricas.
- Visualizar los datos en un plano 2D.

Observar cómo diferentes especies de flores tienden a agruparse cuando proyectamos los datos sobre sus componentes principales.

Al finalizar, comprenderás cómo PCA transforma el espacio de características y cómo puede ayudarte a **entender mejor la estructura interna de tus datos**.

# Actividad Práctica Guiada: Visualización con Reducción de dimensiones (PCA)

## 1.- Importación de librerías y carga de datos

```
[1] # Paso 1: Cargar el dataset y aplicar escalado

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Cargar dataset desde URL
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv"
df = pd.read_csv(url)

# Escalar las características numéricas
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
X_scaled = StandardScaler().fit_transform(df[features])
```

Cargamos el dataset Iris y seleccionamos sus 4 características numéricas. Antes de aplicar PCA, **estandarizamos** los datos para que todas las variables tengan la misma escala. Esto es importante porque PCA calcula combinaciones lineales, y si una variable tiene valores más grandes, puede dominar el resultado injustamente.

# Actividad Práctica Guiada: Visualización con Reducción de dimensiones (PCA)

## 2.- Aplicar PCA

```
[2] # Paso 2: Aplicar PCA y almacenar resultados en un nuevo DataFrame

pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_scaled)

# Crear un nuevo DataFrame con las dos componentes principales
df_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
df_pca['species'] = df['species']
```

Aquí le indicamos a PCA que queremos reducir a 2 dimensiones, pero no elegimos manualmente qué columnas conservar.

PCA automáticamente calcula nuevas variables (PC1 y PC2) que son combinaciones de las originales, y que capturan la mayor cantidad de varianza posible.

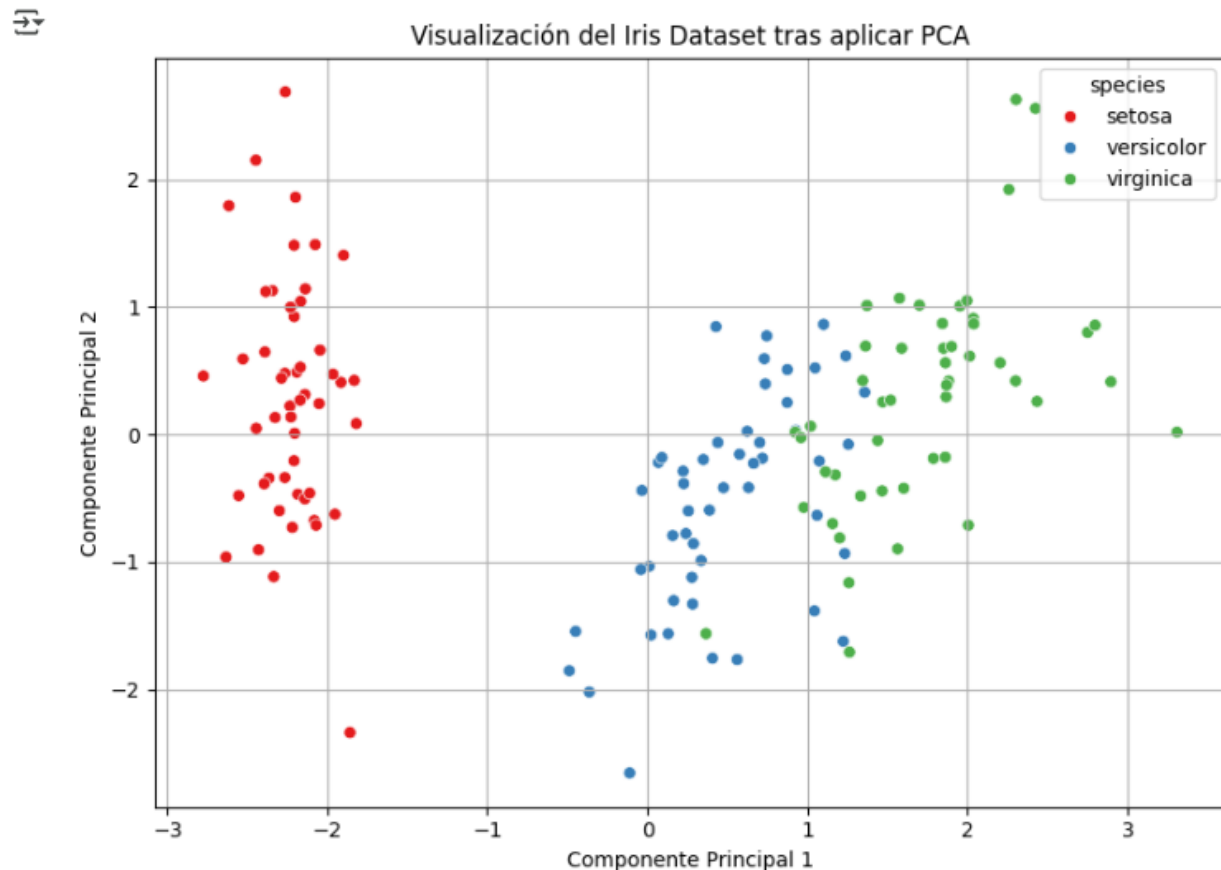
Estas nuevas columnas no son parte del dataset original, pero representan lo más importante de él.

# Actividad Práctica Guiada: Visualización con Reducción de dimensiones (PCA)

## 3.- Visualizar resultados en 2D

```
[3] # Paso 3: Visualización en 2D de los datos transformados por PCA
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='species', palette='Set1')
plt.title('Visualización del Iris Dataset tras aplicar PCA')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Cada punto en este **gráfico** es una **flor**, proyectada en un plano 2D **gracias a PCA**.

El **color** indica su **especie**.

Aquí podemos observar cómo **las clases tienden a separarse** de forma natural, lo que sugiere que la estructura del dataset es bastante clara.

Este tipo de visualización es útil para detectar patrones ocultos, explorar agrupaciones, y mejorar la intuición antes de entrenar modelos.



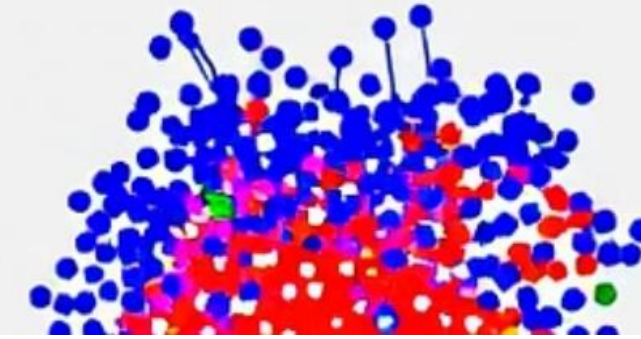
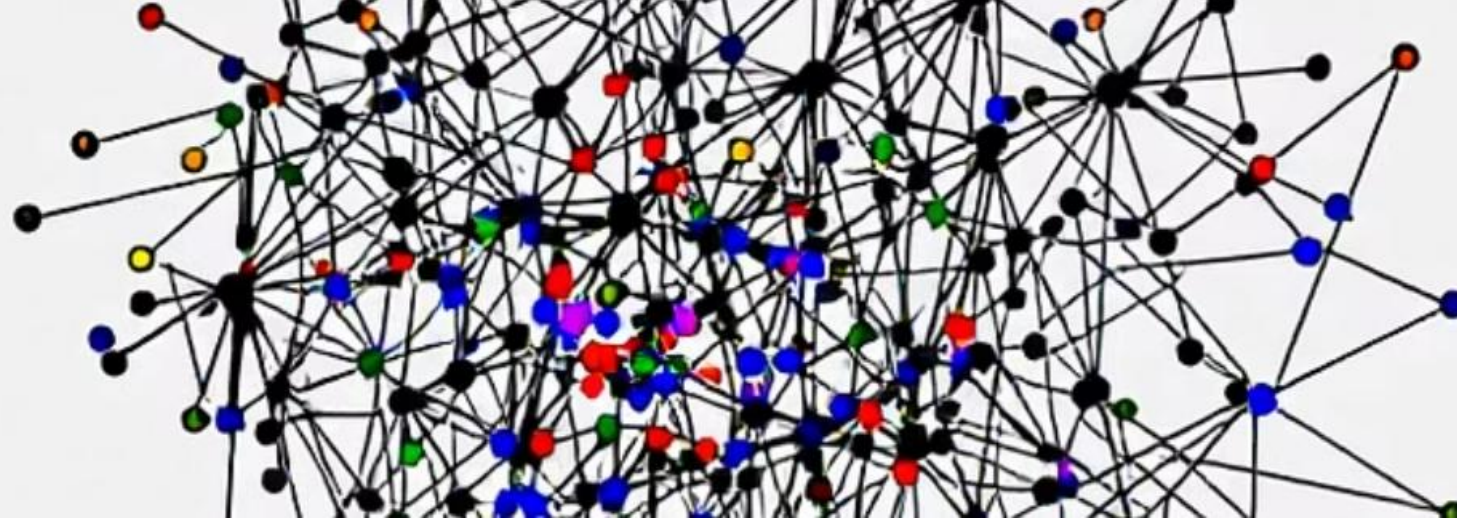
# Actividad Práctica Guiada: Visualización con Reducción de dimensiones (PCA)

## Reflexionemos:

1. ¿Perdimos mucha información al reducir de 4 dimensiones a 2?
  2. ¿Qué ventaja tiene observar los datos en 2D?
  3. ¿Qué especies parecen más fáciles de separar? ¿Cuáles se solapan?
- 
- ✓ Más variables no siempre significan más información útil.
  - ✓ La maldición de la dimensionalidad puede llevar a modelos lentos, confusos o ineficientes.
  - ✓ Usar técnicas como **PCA** o **selección de características** permite trabajar con datos más limpios y comprensibles.

# Enlaces de Interés

- [Netflix utiliza la inteligencia artificial para personalizar sus recomendaciones](#)
- [Un método basado en inteligencia artificial detecta el cáncer del futuro](#)
- [Datacleaning Limpieza de datos: definición, técnicas, importancia en Data Science](#)
- **Puedes profundizar en Evaluación de modelos de ML en este [link](#).**
- [¿Qué es Machine Learning? - Video de IBM Watson \(YouTube\)](#)
- [¿Qué es el Aprendizaje Supervisado y No Supervisado? | DotCSV](#)
- [Todo sobre el Feature Scaling](#)
- [APRENDIZAJE SUPERVISADO: K-NEAREST NEIGHBORS](#)
- [¿Qué es Clustering](#)
- [Visualización de datos de iris y clasificación KNN](#)
- [Comprender la distancia euclidiana: De la teoría a la práctica](#)



# Consideraciones Importantes en el Modelado de Datos



## Preprocesamiento Crítico

¿Qué parte del preprocesamiento te pareció más crítica antes de entrenar un modelo? ¿Por qué?



## Escalado de Datos

¿Qué efectos podría tener no escalar adecuadamente los datos en un modelo como KNN?



## Dimensionalidad

¿Crees que es mejor tener más variables o menos en un dataset? ¿Qué aprendiste sobre la maldición de la dimensionalidad?