2. Fuentes de datos: archivos, bases de datos y APIs.

2.1. La diversidad de las fuentes de datos.

Los datos con los que trabaja un proyecto de machine learning pueden proceder de fuentes muy diversas. Cada tipo de fuente tiene sus propias características técnicas, sus ventajas y sus limitaciones, y requiere herramientas específicas para la extracción.

Conocer las fuentes de datos más habituales y saber cómo acceder a ellas es una competencia fundamental en ciencia de datos. En un proyecto real, raramente los datos llegan en un único fichero perfectamente estructurado. Lo habitual es que haya que combinar datos de varias fuentes con formatos distintos.

Ejemplo práctico

Contexto: Una startup de movilidad lanza un modelo para predecir la demanda de patinetes por barrio y hora.

Planteamiento: Sus datos viven en cuatro sitios: un PostgreSQL con los viajes (relacional), un Excel con la flota (planificación humana), una API meteorológica (JSON en tiempo real) y eventos urbanos extraídos por web scraping del ayuntamiento (HTML).

Resultado: El modelo necesita los cuatro a la vez. Saber qué herramienta usar para cada fuente es una competencia tan importante como elegir el algoritmo.

2.2. Archivos planos: CSV, Excel y JSON.

Los **archivos planos** son la fuente de datos más sencilla y más habitual en proyectos de ciencia de datos, especialmente en fases de prototipado.

El formato **CSV** (Comma-Separated Values) es un fichero de texto donde cada fila representa una observación y los valores de cada columna se separan por comas (o por punto y coma, según la configuración regional). Es el formato más universal para intercambiar datos tabulares:

```
import pandas as pd

# Leer un CSV
df = pd.read_csv("ventas.csv", sep=";", encoding="utf-8")

# Guardar un CSV
df.to_csv("ventas_limpias.csv", index=False)
```

El formato **Excel** (`.xlsx` o `.xls`) es muy habitual en entornos empresariales. Pandas puede leer y escribir ficheros Excel directamente mediante la librería `openpyxl`:

```
df = pd.read_excel("informe.xlsx", sheet_name="Ventas")
```

El formato **JSON** (JavaScript Object Notation) es el estándar para intercambiar datos en APIs web. Tiene una estructura jerárquica de clave-valor que permite representar datos anidados, más flexible que el CSV pero más compleja de manejar:

```
import json

with open("datos.json", "r") as f:
    datos = json.load(f)

# También directamente con Pandas
df = pd.read_json("datos.json")
```

Otros formatos habituales son **Parquet** (formato columnar muy eficiente para grandes volúmenes), **HDF5** (para datos científicos de alta dimensionalidad) y **XML** (en sistemas heredados y servicios web antiguos).

Ejemplo práctico

Contexto: Una analista recibe tres ficheros para construir un dataset de clientes: ventas.csv, contactos.xlsx y comportamiento_web.json.

Planteamiento: Carga el CSV con pd.read_csv() especificando sep=";" y encoding="utf-8" (en España, los CSV vienen muchas veces con punto y coma); abre el Excel con pd.read_excel(sheet_name="Clientes") indicando la hoja exacta; y lee el JSON con pd.read_json() para los datos web anidados.

Resultado: En tres líneas tiene los tres DataFrames listos para combinar. Conocer las opciones de cada lector (separador, encoding, hoja) evita el 90 % de los errores típicos de carga.

2.3. Bases de datos relacionales.

Las **bases de datos relacionales** son el sistema de almacenamiento de datos más utilizado en entornos empresariales. Organizan los datos en tablas relacionadas entre sí mediante claves. Los sistemas de gestión de bases de datos relacionales más habituales son **PostgreSQL**, **MySQL**, **Microsoft SQL Server**, **Oracle** y **SQLite** (este último es una base de datos ligera que no requiere servidor, muy útil para desarrollo y pruebas).

Para conectarse a una base de datos desde Python se utiliza habitualmente la librería **SQLAlchemy** como capa de abstracción, combinada con un driver específico para cada sistema (psycopg2 para PostgreSQL, pymysql para MySQL, etc.):

```
from sqlalchemy import create_engine
import pandas as pd

# Conectar a una base de datos PostgreSQL
engine =
create_engine("postgresql://usuario:contraseña@servidor:5432/nombre_bd")

# Ejecutar una consulta SQL y cargar el resultado en un DataFrame
df = pd.read_sql("SELECT  FROM clientes WHERE activo = true", engine)
```

La ventaja de trabajar con bases de datos es que permiten filtrar y agregar los datos directamente en la consulta SQL antes de cargarlos en memoria, lo que es mucho más eficiente que cargar toda la tabla y filtrar después con Pandas.

Ejemplo práctico

Contexto: Un departamento de finanzas tiene 80 millones de pedidos en PostgreSQL y solo le interesan los del último trimestre por región norte.

Planteamiento: En vez de descargar toda la tabla, escribe la consulta filtrada: pd.read_sql("SELECT cliente_id, importe, fecha FROM pedidos WHERE fecha >= '2025-01-01' AND region = 'NORTE'", engine). El servidor de la base de datos filtra antes de enviar.

Resultado: Recibe 1,2 millones de filas en lugar de 80 millones. La consulta tarda 4 segundos; cargar todo y filtrar después habría reventado la memoria del portátil.

2.4. APIs web.

Una **API** (Application Programming Interface) es una interfaz que permite a un programa acceder a los datos o funcionalidades de otro sistema a través de internet. Las APIs web son una fuente de datos muy importante en ciencia de datos: muchos servicios (redes sociales, sistemas meteorológicos, plataformas financieras, sistemas de geolocalización) ofrecen sus datos a través de APIs.

El protocolo más habitual es **REST** (Representational State Transfer), que utiliza peticiones HTTP para solicitar y enviar datos en formato JSON. En Python, la librería **requests** simplifica enormemente el trabajo con APIs:

```
import requests

url = "https://api.ejemplo.com/datos"
cabeceras = {"Authorization": "Bearer MI_TOKEN"}
respuesta = requests.get(url, headers=cabeceras)

if respuesta.status_code == 200:
    datos = respuesta.json()
else:
    print(f"Error: {respuesta.status_code}")
```

Las APIs suelen requerir autenticación (mediante claves de API o tokens) y pueden tener límites en el número de peticiones por hora o por día (rate limits). Es importante gestionar estos límites en el código para evitar errores o bloqueos.

Ejemplo práctico

Contexto: Un equipo de análisis quiere enriquecer su CRM con datos de clima para correlacionar ventas con temperatura.

Planteamiento: Llama a la API de OpenWeatherMap con requests.get(), añadiendo en las cabeceras el token de autenticación y respetando el límite de 60 peticiones/minuto. Para no saltarlo, intercala time.sleep(1) entre llamadas y captura el código 429 (Too Many Requests).

Resultado: Recibe los datos meteorológicos de cada código postal en JSON y los integra en el dataset. La API se ha convertido en una fuente de datos tan fiable como un fichero local.

2.5. Web scraping

Cuando los datos no están disponibles en ningún formato estructurado ni a través de ninguna API, es posible extraerlos directamente de páginas web mediante **web scraping**: el proceso de analizar el HTML de una página y extraer los datos de interés.

En Python, las librerías más habituales para web scraping son **BeautifulSoup** (para analizar HTML) y **Selenium** (para interactuar con páginas que requieren JavaScript). El web scraping debe hacerse respetando los términos de uso del sitio web y las normas legales aplicables.

Ejemplo práctico

Contexto: Una analista de mercado quiere conocer los precios actuales de 200 productos competidores que no publican lista oficial.

Planteamiento: Escribe un script con BeautifulSoup que descarga la página de cada producto, localiza la etiqueta HTML del precio (por ejemplo) y extrae el valor. Para páginas con precios cargados por JavaScript, usa Selenium.

Resultado: Obtiene un CSV diario con los 200 precios actualizados. Antes de poner el scraper en producción comprueba los términos de uso del sitio y el archivo robots.txt para no incumplir la legalidad.

2.6. Bases de datos no relacionales (NoSQL).

Las **bases de datos NoSQL** almacenan datos en formatos distintos a las tablas relacionales: documentos JSON (MongoDB), grafos (Neo4j), clave-valor (Redis) o columnas anchas (Cassandra). Son especialmente adecuadas para datos no estructurados o semiestructurados, como registros de logs, datos de sensores o perfiles de usuarios con atributos variables.

Ejemplo práctico

Contexto: Una red social almacena perfiles donde cada usuario puede tener atributos distintos: unos rellenan biografía, otros vinculan Spotify, otros suben fotos con etiquetas.

Planteamiento: Un esquema relacional con columnas fijas obligaría a dejar muchas vacías. En su lugar guardan cada perfil como un documento JSON en MongoDB y desde Python lo leen con: list(coleccion.find({"activo": True})).

Resultado: Cada documento puede tener su propio conjunto de campos sin que la base de datos proteste. NoSQL encaja cuando la estructura es heterogénea por naturaleza.

En ciencia de datos, MongoDB es la base de datos NoSQL más habitual. La librería **pymongo** permite conectarse y consultar datos desde Python:

```
from pymongo import MongoClient
import pandas as pd

cliente = MongoClient("mongodb://localhost:27017/")
db = cliente["mi_base_de_datos"]
coleccion = db["clientes"]

datos = list(coleccion.find({"activo": True}))
df = pd.DataFrame(datos)
```

Tipo de fuente	Formato habitual	Librería Python	Caso de uso típico
Archivo plano	CSV, Excel, JSON, Parquet	Pandas, json	Prototipos, intercambio de datos
Base de datos relacional	Tablas SQL	SQLAlchemy, psycopg2	Sistemas empresariales
API REST	JSON	requests	Datos de terceros en tiempo real
Web scraping	HTML	BeautifulSoup, Selenium	Datos públicos no estructurados
Base de datos NoSQL	JSON, clave-valor, grafos	pymongo, redis-py	Datos semiestructurados o en tiempo real
Datos en streaming	JSON, Avro	Kafka, PySpark	Datos en tiempo real de sensores o logs

Actividad de relacionar 11. Fuentes de datos.

Relaciona cada fuente de datos de la columna A con la librería o herramienta de Python más adecuada para acceder a ella de la columna B.

Columna A: Fuente de datos.	Columna B: Librería o herramienta.
1. Fichero CSV almacenado localmente	A. requests
2. Base de datos PostgreSQL	B. pymongo
3. API REST que devuelve datos en JSON	C. pandas.read_csv()
4. Base de datos MongoDB	D. SQLAlchemy + psycopg2
5. Página web con datos en formato HTML	E. BeautifulSoup

3. Limpieza de datos: valores nulos, duplicados y errores.

3.1. Por qué los datos reales están sucios.

En un proyecto de machine learning real, es muy raro encontrar un conjunto de datos perfectamente limpio y listo para usar.

Los datos reales proceden de sistemas heterogéneos, de procesos manuales de introducción de datos, de sistemas heredados o de fuentes externas que no siempre tienen los mismos estándares de calidad. El resultado es que casi siempre contienen algún tipo de problema: valores ausentes, duplicados, errores tipográficos, formatos inconsistentes, valores imposibles o variables mal definidas.

La limpieza de datos no es un proceso que se pueda automatizar completamente. Requiere criterio, conocimiento del dominio y decisiones que tienen consecuencias para el análisis. Decidir cómo tratar los valores nulos, por ejemplo, puede afectar significativamente a los resultados del modelo.

Ejemplo práctico

Contexto: Un equipo recibe un dataset de 50.000 pedidos para predecir devoluciones y lo lanza directamente al modelo.

Planteamiento: El primer entrenamiento da una precisión del 99 % (sospechosamente alta). Al explorar, descubren que la columna «motivo_devolución» —que se rellena después de la devolución— estaba como variable predictora. Además, 6.000 filas son duplicados por errores del TPV.

Resultado: Quitan la columna culpable, eliminan los duplicados y vuelven a entrenar. La precisión cae al 78 %, pero ahora es real. Sin esa limpieza, el modelo en producción habría dado predicciones inservibles.

3.2. Valores nulos o ausentes.

Un **valor nulo** (también llamado valor ausente o missing value) es una observación para la que no existe ningún dato registrado en una variable concreta. En Pandas, los valores nulos se representan habitualmente como `NaN` (Not a Number) para variables numéricas y como `None` para variables de texto.

El primer paso es detectar cuántos valores nulos hay en cada columna:

```
# Ver el número de nulos por columna
df.isnull().sum()

# Ver el porcentaje de nulos por columna
df.isnull().mean()  100
```

Una vez detectados, existen varias estrategias para manejar los valores nulos:

Eliminar las filas o columnas con nulos. Es la opción más sencilla, pero puede suponer una pérdida de información importante si hay muchos nulos o si los nulos no son aleatorios (es decir, si los datos ausentes tienen algún patrón relacionado con la variable objetivo).

```
# Eliminar filas con al menos un valor nulo
df_limpio = df.dropna()

# Eliminar columnas con más del 50 % de nulos
df_limpio = df.dropna(axis=1, thresh=int(0.5  len(df)))
```

Imputar los valores nulos. La imputación consiste en rellenar los valores nulos con un valor calculado o estimado. Las estrategias más habituales son:

- Imputar con la media o la mediana (para variables numéricas).
- Imputar con la moda (para variables categóricas).
- Imputar con el valor anterior o posterior (para series temporales, donde tiene sentido la continuidad).
- Imputar con un valor fijo (por ejemplo, 0 o «Desconocido»).
- Usar un modelo predictivo para estimar el valor más probable (imputación avanzada).

```
# Imputar con la mediana
df["edad"].fillna(df["edad"].median(), inplace=True)

# Imputar con la moda
df["ciudad"].fillna(df["ciudad"].mode()[0], inplace=True)

# Scikit-learn también ofrece imputadores más sofisticados
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="mean")
df_imputado = imputer.fit_transform(df[["edad", "ingresos"]])
```

La elección entre eliminar o imputar depende del contexto. Como regla general, si una columna tiene más del 40-50 % de valores nulos, suele ser mejor eliminarla. Si tiene pocos nulos y son aleatorios, la imputación suele ser la mejor opción.

3.3. Valores duplicados.

Un **registro duplicado** es una fila que aparece más de una vez en el conjunto de datos con los mismos valores (o valores muy similares) en todas sus columnas. Los duplicados inflan artificialmente el tamaño del dataset y pueden sesgar el modelo si una misma observación se cuenta varias veces en el entrenamiento.

Detectar y eliminar duplicados en Pandas es sencillo:

```
# Detectar cuántas filas están duplicadas
df.duplicated().sum()

# Ver las filas duplicadas
df[df.duplicated()]
```

```
# Eliminar duplicados (conservar la primera aparición)
df_limpio = df.drop_duplicates()

# Eliminar duplicados basados en columnas específicas
df_limpio = df.drop_duplicates(subset=["id_cliente", "fecha_pedido"])
```

Es importante definir bien qué significa «duplicado» en el contexto del problema. A veces dos filas tienen el mismo ID de cliente, pero corresponden a pedidos distintos: no son duplicados. Otras veces, dos filas tienen el mismo contenido porque el mismo registro se importó dos veces: son duplicados reales.

3.4. Errores tipográficos y valores inconsistentes.

Los errores tipográficos son especialmente habituales en variables de texto introducidas manualmente. Por ejemplo, la misma ciudad puede aparecer escrita como «Madrid», «madrid», «MADRID» o «Madrd». Para un algoritmo, estas son cuatro categorías distintas, cuando en realidad deberían ser una sola.

Las técnicas habituales para detectar y corregir este tipo de errores son:

- Estandarización de mayúsculas y minúsculas:

```
df["ciudad"] = df["ciudad"].str.strip().str.title()
# "madrid" → "Madrid", " MADRID " → "Madrid"
```

- Eliminar espacios en blanco adicionales:

```
df["ciudad"] = df["ciudad"].str.strip()
```

- Corrección de valores específicos mediante diccionarios de mapeo:

```
correcciones = {"Madrd": "Madrid", "Barcelna": "Barcelona", "Sevlla":
"Sevilla"}
df["ciudad"] = df["ciudad"].replace(correcciones)
```

- Fuzzy matching: **técnica que compara cadenas de texto por similitud (no por igualdad exacta) y agrupa valores que probablemente representan lo mismo. La librería** fuzzywuzzy **(o su sucesor** rapidfuzz) implementa estas técnicas en Python.

Ejemplo práctico

Contexto: Una analista recibe una tabla con la columna «ciudad» donde Madrid aparece como «Madrid», «madrid», «MADRID», «Madrd», « Madrid » y «Madrid (España)».

Planteamiento: Aplica primero df["ciudad"] = df["ciudad"].str.strip().str.title() para uniformar mayúsculas y espacios. Después define un diccionario de correcciones {"Madrd": "Madrid", "Madrid (España)": "Madrid"} y lo aplica con replace.

Resultado: De seis variantes pasa a una sola «Madrid». El modelo, que antes habría tratado cada forma como una categoría distinta, ahora ve correctamente una única ciudad.

3.5. Valores imposibles y outliers.

Los **valores imposibles** son valores que no pueden ser correctos desde el punto de vista del dominio: una edad de -5 años, una temperatura de 500 °C en un entorno doméstico o un porcentaje de descuento del 150 %. Este tipo de errores suelen deberse a errores de introducción de datos o a problemas en la integración de fuentes.

La detección de estos valores requiere conocer el dominio: ¿cuáles son los rangos válidos para cada variable? Una vez definidos, es posible filtrarlos y tratarlos como valores nulos:

```
# Eliminar edades fuera del rango válido
df = df[df["edad"].between(0, 120)]

# Reemplazar valores imposibles por NaN
df.loc[df["descuento"] > 100, "descuento"] = None
```

Los **outliers** (valores atípicos) no son necesariamente errores: pueden ser observaciones genuinas pero inusuales. Como se vio en la Unidad 1, el rango intercuartílico es el criterio más habitual para detectarlos. La decisión de eliminarlos o conservarlos depende del contexto y del tipo de modelo.

Actividad de relacionar 12. Problemas de calidad de datos

Relaciona cada problema de la columna A con la técnica de limpieza más adecuada de la columna B.

Columna A: Problema.	Columna B: Técnica de limpieza.
1. Una columna numérica con el 20 % de valores ausentes	A. drop_duplicates()
2. La misma ciudad escrita como "madrid" y "Madrid"	B. Eliminar filas o columnas con dropna()
3. Filas idénticas cargadas dos veces al importar el fichero	C. Regla de tres sigmas o IQR
4. Una columna con el 80 % de valores nulos	D. fillna() con la mediana o la moda
5. Un valor de edad de 350 años	E. str.title() o replace() con diccionario

4. Transformación y normalización de datos.

4.1. Por qué es necesario transformar los datos.

Una vez limpios, los datos necesitan en muchos casos ser transformados antes de ser introducidos en un modelo de machine learning. La razón es que muchos algoritmos tienen supuestos o requisitos sobre la forma en que deben estar los datos: que estén en la misma escala, que sigan una distribución aproximadamente normal, que las variables categóricas estén codificadas como números o que no existan variables redundantes.

La fase de transformación convierte los datos limpios en datos aptos para el modelo. Es también el momento en que se puede crear nuevas variables a partir de las existentes, una técnica conocida como **ingeniería de características** o feature engineering.

Ejemplo práctico

Contexto: Una analista quiere entrenar una red neuronal con dos variables: edad (rango 18-80) e ingresos (rango 12.000-200.000 €).

Planteamiento: Lanza el entrenamiento sin transformar nada. El modelo converge mal: los pesos asociados a «ingresos» dominan el aprendizaje porque su escala es 1.000 veces mayor que la de «edad». Aplica StandardScaler para llevar ambas a media 0 y desviación 1.

Resultado: Tras la estandarización, el modelo converge en la mitad de épocas y mejora 8 puntos de F1. La transformación no era cosmética: era condición para que el algoritmo funcionara.

4.2. Normalización y estandarización.

La **normalización** y la **estandarización** son dos técnicas para escalar variables numéricas. Son necesarias porque muchos algoritmos —como la regresión logística, las redes neuronales, el algoritmo k-NN o las máquinas de vectores soporte— son sensibles a la escala de las variables: si una variable va de 0 a 1 y otra va de 0 a 1.000.000, el algoritmo tenderá a dar mucha más importancia a la segunda, aunque no sea más relevante.

La **normalización min-max** transforma los valores de una variable para que queden en el rango [0, 1]:

```
x_normalizado = (x - x_min) / (x_max - x_min)
```

La **estandarización z-score** transforma los valores para que tengan media 0 y desviación típica 1:

```
x_estandarizado = (x - media) / desviación_típica
```

En Scikit-learn, estas transformaciones se implementan con `MinMaxScaler` y `StandardScaler`:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler_minmax = MinMaxScaler()
df_normalizado = scaler_minmax.fit_transform(df[["edad", "ingresos"]])

scaler_std = StandardScaler()
df_estandarizado = scaler_std.fit_transform(df[["edad", "ingresos"]])
```

Un aspecto crítico es que el escalado debe ajustarse (fit) solo sobre los datos de entrenamiento y aplicarse (transform) tanto a los datos de entrenamiento como a los de test. Si se incluyen los datos de test en el ajuste del escalado, se produce una **fuga de datos** (data leakage), que da lugar a métricas de evaluación artificialmente optimistas.

Técnica	Fórmula	Rango resultante	Cuándo usarla
Normalización min-max	(x - valor mínimo) / (valor máximo - valor mínimo)	Entre 0 y 1	Cuando se quiere transformar los datos a una misma escala y no hay valores atípicos extremos.
Estandarización z-score	(x - media) / desviación típica	Media 0 y desviación típica 1	Cuando las variables tienen escalas diferentes y se quiere comparar los datos respecto a su media.
Escalado robusto	(x - mediana) / IQR	Sin rango fijo	Cuando hay valores atípicos u outliers, porque usa la mediana y el rango intercuartílico.
Normalización L2	vector original / norma L2 del vector	Norma unitaria	Para datos de texto o vectores de características, especialmente cuando interesa comparar la similitud o la dirección entre vectores.

4.3. Transformaciones de distribución.

Cuando una variable tiene un sesgo muy pronunciado, puede ser útil aplicarle una transformación matemática para acercar su distribución a la normal. Las más habituales son:

- Transformación logarítmica: muy efectiva para variables con sesgo positivo (cola derecha), como ingresos o precios. `log(x)` solo es válida para valores positivos.
- Raíz cuadrada: suaviza el sesgo de forma menos agresiva que el logaritmo.
- Transformación de Box-Cox: familia paramétrica de transformaciones que busca automáticamente el exponente que mejor normaliza la distribución. Solo válida para valores positivos.
- Transformación de Yeo-Johnson: similar a Box-Cox pero también válida para valores cero o negativos.

```
import numpy as np
from scipy import stats

# Transformación logarítmica
df["ingresos_log"] = np.log1p(df["ingresos"])  # log(1+x) para evitar
log(0)

# Box-Cox (solo valores positivos)
df["precio_bc"], lambda_bc = stats.boxcox(df["precio"])
```

4.4. Codificación de variables categóricas.

Como se vio en la Unidad 1, los algoritmos de machine learning requieren que todas las variables sean numéricas. La codificación de variables categóricas es una de las transformaciones más habituales.

Las técnicas principales son:

- Label encoding: asigna un número entero a cada categoría. Adecuado para variables ordinales. Para variables nominales puede introducir un orden artificial.
- One-hot encoding: crea una columna binaria por cada categoría. Adecuado para variables nominales con pocas categorías.
- Ordinal encoding: similar al label encoding pero respetando el orden natural de las categorías ordinales.
- Target encoding: sustituye cada categoría por la media de la variable objetivo para esa categoría. Muy potente pero puede introducir fuga de datos si no se aplica correctamente.
- Frecuency encoding: sustituye cada categoría por su frecuencia relativa en el dataset.

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

# One-hot encoding con Pandas
df_encoded = pd.get_dummies(df, columns=["ciudad", "categoria"])

# Label encoding con Scikit-learn
le = LabelEncoder()
df["ciudad_encoded"] = le.fit_transform(df["ciudad"])
```

Cuando una variable categórica tiene muchas categorías (alta cardinalidad), el one-hot encoding genera demasiadas columnas. En ese caso, el target encoding o el frecuency encoding son alternativas más eficientes.

Ejemplo práctico

Contexto: Una analista trabaja con dos variables categóricas en un modelo de scoring: «nivel_educativo» (4 categorías ordenadas) y «provincia» (50 categorías sin orden).

Planteamiento: Para «nivel_educativo» aplica OrdinalEncoder con el orden ESO < Bachillerato < Grado < Máster. Para «provincia», como tiene muchas categorías, sustituye one-hot por target encoding: cada provincia se sustituye por la tasa media de impago histórica.

Resultado: El modelo aprende que el orden educativo importa y que la geografía aporta una señal de riesgo continua, sin disparar la dimensionalidad. La codificación elegida cambia radicalmente el comportamiento del algoritmo.

4.5. Ingeniería de características (feature engineering).

La **ingeniería de características** es el proceso de crear nuevas variables a partir de las existentes para mejorar el rendimiento del modelo. Es una de las habilidades más valiosas en ciencia de datos porque permite codificar conocimiento del dominio en el formato que mejor aprovecha el algoritmo.

Algunos ejemplos habituales de ingeniería de características:

- Variables de fecha: a partir de una fecha, se pueden extraer el año, el mes, el día de la semana, si es festivo, si es fin de semana o la diferencia respecto a otra fecha.

```
df["fecha"] = pd.to_datetime(df["fecha"])
df["mes"] = df["fecha"].dt.month
df["dia_semana"] = df["fecha"].dt.dayofweek
df["es_fin_de_semana"] = df["dia_semana"].isin([5, 6]).astype(int)
```

- Interacciones entre variables: multiplicar o dividir dos variables puede capturar relaciones no lineales que un modelo lineal no detectaría por sí solo. Por ejemplo, el ratio entre ingresos y gastos puede ser más informativo que las dos variables por separado.
- Variables de texto: a partir de una cadena de texto se puede extraer su longitud, el número de palabras, si contiene ciertas palabras clave o aplicar técnicas de procesamiento de lenguaje natural (PLN).
- Variables de agrupación: calcular estadísticas (media, máximo, mínimo) de una variable numérica agrupada por una categórica. Por ejemplo, el importe medio de compra de cada cliente en el histórico.

4.6. Selección de características (feature selection).

No todas las variables disponibles aportan información útil al modelo. Incluir variables irrelevantes o redundantes puede empeorar el rendimiento, aumentar el tiempo de entrenamiento y dificultar la interpretación.

La **selección de características** es el proceso de identificar y conservar solo las variables más relevantes para el problema. Los métodos más habituales son:

- Métodos de filtro: evalúan la relevancia de cada variable de forma independiente, usando métricas estadísticas como la correlación con la variable objetivo, el test chi-cuadrado (para variables categóricas) o la varianza.
- Métodos de envoltura (wrapper): evalúan subconjuntos de variables entrenando y evaluando el modelo para cada subconjunto. Son más precisos, pero computacionalmente costosos.
- Métodos de incrustación (embedded): la selección de variables forma parte del propio proceso de entrenamiento del modelo. Los árboles de decisión y los modelos de regularización (Lasso) son ejemplos de este enfoque.

```
from sklearn.feature_selection import SelectKBest, f_classif

# Seleccionar las 10 mejores variables por test F
selector = SelectKBest(score_func=f_classif, k=10)
X_seleccionado = selector.fit_transform(X, y)
```

Actividad de relacionar 13. Transformaciones de datos.

Relaciona cada situación de la columna A con la transformación más adecuada de la columna B.

Columna A: Situación.	Columna B: Transformación.
1. Una variable numérica con sesgo positivo muy marcado	A. One-hot encoding
2. Una variable categórica nominal con 5 categorías	B. Estandarización z-score
3. Dos variables en escalas muy distintas (0-1 y 0-1.000.000) para un modelo de red neuronal	C. Transformación logarítmica
4. Extraer el día de la semana de una columna de fechas	D. Ingeniería de características con .dt.dayofweek
5. Una variable ordinal con categorías "bajo", "medio" y "alto"	E. Ordinal encoding

5. Integración de datos procedentes de distintas fuentes.

5.1. El reto de combinar datos heterogéneos.

En proyectos reales de machine learning, los datos raramente provienen de una única fuente perfectamente estructurada. Lo habitual es tener que combinar datos de varias fuentes: una base de datos de clientes, un sistema de facturación, un fichero de encuestas de satisfacción y quizás datos de comportamiento web extraídos de una API.

La **integración de datos** es el proceso de combinar estas fuentes en un único conjunto de datos coherente y consistente. Es una de las etapas más complejas del proceso ETL porque requiere resolver varios problemas simultáneamente: claves de unión distintas, formatos inconsistentes entre fuentes, duplicidades, distintos niveles de granularidad y posibles contradicciones entre fuentes.

Ejemplo práctico

Contexto: Un proyecto necesita combinar datos de CRM (10.000 clientes), facturación (45.000 pedidos), encuestas NPS (2.300 respuestas) y comportamiento web (350.000 sesiones) para predecir abandono.

Planteamiento: El equipo dedica las primeras dos semanas no a entrenar el modelo, sino a resolver: los IDs de cliente vienen en tres formatos distintos, una misma persona aparece como «Juan García» en CRM y «J. García» en facturación, las fechas usan tres zonas horarias.

Resultado: Hasta que la integración no está resuelta, no hay modelo posible. La complejidad del proyecto vive aquí, no en elegir el algoritmo.

5.2. Uniones (joins) entre tablas.

La operación más habitual para combinar dos fuentes de datos es el **join** (unión), que combina dos tablas basándose en una columna común. Existen cuatro tipos principales de join:

- Inner join: devuelve solo las filas que tienen correspondencia en ambas tablas. Las filas sin correspondencia se eliminan.
- Left join: devuelve todas las filas de la tabla izquierda y las filas correspondientes de la derecha. Las filas de la izquierda sin correspondencia se rellenan con nulos en las columnas de la derecha.
- Right join: análogo al left join pero conservando todas las filas de la tabla derecha.
- Full outer join: devuelve todas las filas de ambas tablas, rellenando con nulos donde no hay correspondencia.

```
import pandas as pd

# Combinar clientes con sus pedidos (inner join por defecto)
df_combinado = pd.merge(df_clientes, df_pedidos, on="id_cliente",
how="left")
```

Tipo de join	Filas que conserva	Cuándo usarlo
Inner join	Solo las que tienen correspondencia en ambas tablas	Cuando solo interesan registros comunes
Left join	Todas las de la tabla izquierda	Cuando la tabla izquierda es la principal
Right join	Todas las de la tabla derecha	Cuando la tabla derecha es la principal
Full outer join	Todas las de ambas tablas	Cuando no se quiere perder ningún registro

5.3. Concatenación de tablas.

La **concatenación** une dos tablas con la misma estructura (mismas columnas) apilando una sobre la otra. Es útil cuando se tienen datos del mismo tipo pero de períodos o fuentes distintos:

```
# Unir dos DataFrames con la misma estructura
df_total = pd.concat([df_enero, df_febrero, df_marzo], ignore_index=True)
```

5.4. Problemas frecuentes en la integración.

La integración de datos presenta varios problemas habituales que hay que anticipar:

- Claves de unión con formatos distintos. En una tabla el ID de cliente puede ser numérico y en otra puede ser un texto con formato «CLI-00123». Hay que estandarizar las claves antes de hacer el join.
- Distinta granularidad. Una tabla puede tener un registro por cliente y otra puede tener un registro por pedido. Antes de combinarlas hay que agregar la más detallada al nivel de la más general.
- Valores contradictorios. Dos fuentes pueden tener valores diferentes para el mismo campo de la misma entidad. Por ejemplo, la dirección de un cliente puede ser distinta en el sistema de ventas y en el sistema de atención al cliente. Hay que definir qué fuente tiene prioridad.
- Campos con el mismo nombre, pero significado distinto. Por ejemplo, el campo «fecha» en una tabla puede ser la fecha de alta del cliente y en otra puede ser la fecha del último pedido. Hay que renombrar los campos antes de combinar las tablas.

5.5. Agregaciones y pivotado.

Antes de combinar tablas de distinta granularidad, suele ser necesario **agregar** la tabla más detallada. Por ejemplo, para combinar una tabla de clientes con una tabla de pedidos, puede interesar calcular primero el número total de pedidos, el importe medio o la fecha del último pedido de cada cliente:

```
# Agregar pedidos por cliente
resumen_pedidos = df_pedidos.groupby("id_cliente").agg(
    num_pedidos = ("id_pedido", "count"),
    importe_total = ("importe", "sum"),
    importe_medio = ("importe", "mean"),
    ultimo_pedido = ("fecha", "max")
).reset_index()

# Combinar con la tabla de clientes
df_final = pd.merge(df_clientes, resumen_pedidos, on="id_cliente",
how="left")
```

El **pivotado** transforma filas en columnas o columnas en filas, lo que es útil cuando los datos están en formato largo (long format) y se necesitan en formato ancho (wide format) o viceversa:

```
# De formato largo a formato ancho
df_pivot = df.pivot_table(index="cliente", columns="mes", values="ventas",
aggfunc="sum")
```

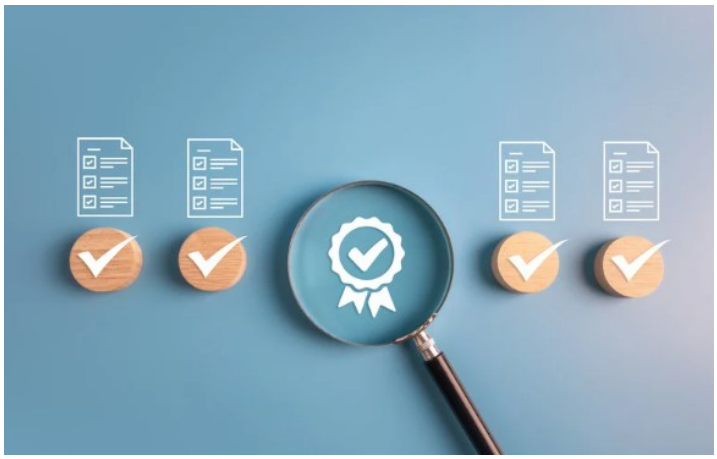
Actividad de relacionar 14. Integración de datos.

Relaciona cada situación de la columna A con la operación de integración más adecuada de la columna B.

Columna A: Situación.	Columna B: Operación.
Combinar datos de clientes con sus pedidos conservando solo los clientes que tienen pedidos	A. Concatenación (pd.concat)
Apilar los registros de ventas de enero y febrero en un único DataFrame	B. Inner join
Calcular el importe total de pedidos por cliente antes de unirlo a la tabla de clientes	C. Left join
Combinar datos de clientes con sus pedidos conservando todos los clientes, aunque no tengan pedidos	D. groupby + agg
Transformar datos en formato largo a formato ancho por meses	E. pivot_table

6. Calidad del dato y trazabilidad del proceso.

6.1. Qué es la calidad del dato.



La **calidad del dato** es el grado en que los datos satisfacen los requisitos para los que van a ser utilizados. Un dato de alta calidad es aquel que es preciso, completo, consistente, actualizado, relevante y comprensible.

En el contexto del machine learning, la calidad del dato es especialmente crítica porque los modelos aprenden directamente de los datos. Un modelo entrenado con datos de baja calidad generará predicciones poco fiables, aunque el algoritmo sea el más sofisticado disponible.

Ejemplo práctico

Contexto: Un banco entrena un modelo de scoring sobre datos extraídos de su CRM antiguo.

Planteamiento: El modelo tiene una precisión teórica del 91 %. Al ponerlo en producción, falla. Investigan: el 40 % de los códigos postales están vacíos, hay 3.000 clientes duplicados con IDs distintos, las edades de algunos llegan a 250 años y un 8 % de los pedidos apuntan a clientes inexistentes.

Resultado: El problema no es el algoritmo: es que entrena con datos que fallan en exactitud, completitud, unicidad e integridad. «Garbage in, garbage out»: ningún modelo arregla datos rotos.

Las dimensiones principales de la calidad del dato son:

Dimensión	Descripción	Ejemplo de problema
Exactitud	El dato refleja correctamente la realidad	Edad registrada como 250 años
Completitud	No hay valores ausentes relevantes	El 40 % de los registros no tienen código postal
Consistencia	El dato tiene el mismo formato y significado en todas las fuentes	La misma ciudad escrita de formas distintas
Actualidad	El dato está vigente y no está desfasado	Dirección de cliente desactualizada
Unicidad	No hay duplicados innecesarios	El mismo cliente aparece dos veces con ID distintos
Validez	El dato cumple las reglas del dominio	Un porcentaje con valor 150 %
Integridad	Las relaciones entre tablas son coherentes	Un pedido asociado a un ID de cliente inexistente

6.2. El perfil de datos (data profiling).

El **perfil de datos** es el análisis sistemático de la estructura, el contenido y la calidad de un conjunto de datos. Es el punto de partida de cualquier proceso de limpieza y una práctica imprescindible antes de iniciar un proyecto de machine learning.

Un perfil de datos incluye:

- Número de filas y columnas.
- Tipo de dato de cada columna.
- Número y porcentaje de valores nulos por columna.
- Estadísticas descriptivas (media, mediana, mínimo, máximo, cuartiles) para variables numéricas.
- Número de valores únicos y distribución de frecuencias para variables categóricas.
- Detección de outliers y valores imposibles.
- Análisis de correlación entre variables.

En Python, la librería **ydata-profiling** (anteriormente pandas-profiling) genera automáticamente un informe de perfil de datos muy completo con una sola línea de código:

```
from ydata_profiling import ProfileReport

perfil = ProfileReport(df, title="Perfil del dataset de clientes")
perfil.to_file("perfil_datos.html")
```

Este informe es muy útil al inicio de un proyecto para obtener una visión rápida del estado del conjunto de datos y priorizar los problemas a resolver.

6.3. Trazabilidad del proceso.

La **trazabilidad** es la capacidad de reconstruir el historial completo de transformaciones aplicadas a los datos: de dónde vienen, qué se ha hecho con ellos y por qué se tomaron determinadas decisiones.

La trazabilidad es importante por varias razones:

- Permite reproducir el proceso exactamente si es necesario repetirlo con datos nuevos.
- Facilita la auditoría del proceso por parte de terceros.
- Permite detectar el origen de errores cuando los resultados del modelo no son los esperados.
- Es un requisito en algunos entornos regulados (como los análisis de riesgo en banca o en sanidad).

Ejemplo práctico

Contexto: El supervisor bancario pide a una entidad demostrar cómo se construyó la variable «score_riesgo» que entró al modelo de concesión de hipotecas hace tres meses.

Planteamiento: Gracias a la trazabilidad documentada (commits de Git, comentarios en código, parámetros guardados en un archivo de configuración), el equipo reconstruye paso a paso: qué tabla origen, qué imputación, qué umbral de outliers, qué transformación final.

Resultado: Entregan el informe completo en dos días. Sin trazabilidad, habrían tenido que rehacer el proceso desde cero y arriesgar diferencias. La trazabilidad es ingeniería de seguros: invertir tiempo cuando se construye para no perderlo cuando se audita.

Las buenas prácticas para garantizar la trazabilidad incluyen:

- Documentar cada decisión de preprocesamiento en comentarios del código o en un documento de registro.
- Conservar los datos originales sin modificar y trabajar siempre sobre copias.
- Versionar el código con Git para registrar los cambios en el proceso.
- Registrar los parámetros de cada transformación (por ejemplo, los valores usados para imputar nulos o los umbrales para eliminar outliers).
- Usar pipelines de Scikit-learn para encadenar las transformaciones de forma reproducible y evitar la fuga de datos:

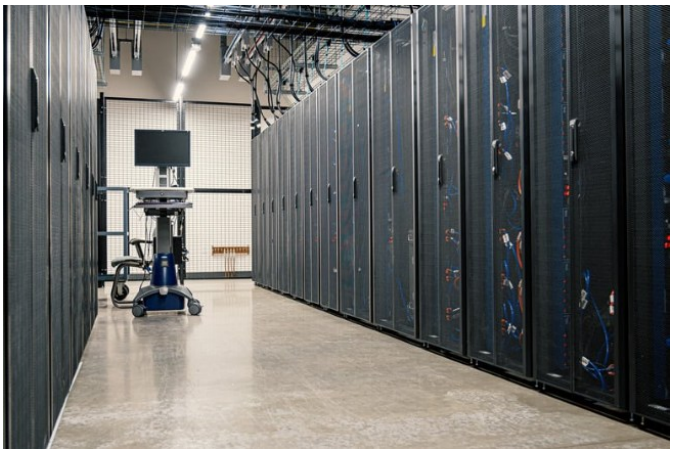
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

pipeline = Pipeline([
    ("imputar_nulos", SimpleImputer(strategy="median")),
    ("estandarizar",  StandardScaler())
])

X_train_prep = pipeline.fit_transform(X_train)
X_test_prep  = pipeline.transform(X_test)   # Solo transform, no fit
```

El uso de pipelines de Scikit-learn es una de las mejores prácticas en machine learning. Encadena todas las transformaciones en un único objeto que se puede ajustar sobre los datos de entrenamiento y aplicar de forma consistente sobre cualquier conjunto de datos nuevo, garantizando la reproducibilidad y evitando la fuga de datos.

6.4. Fuga de datos (data leakage).



La **fuga de datos** es uno de los errores más graves y más difíciles de detectar en un proyecto de machine learning. Se produce cuando información del conjunto de test (o del futuro) se filtra al proceso de entrenamiento, dando lugar a métricas de evaluación artificialmente altas que no se mantienen cuando el modelo se despliega en producción.

Los dos tipos más habituales de fuga de datos son:

- Fuga de preprocesamiento: el escalado, la imputación u otras transformaciones se ajustan sobre todo el dataset (incluyendo el test) en lugar de solo sobre el entrenamiento. Esto hace que el modelo «vea» estadísticas del conjunto de test durante el entrenamiento.
- Fuga de variables: se incluyen en el modelo variables que, en un escenario real, no estarían disponibles en el momento de hacer la predicción. Por ejemplo, incluir como variable predictora el resultado de un análisis que se realiza después del evento que se quiere predecir.

Detectar y evitar la fuga de datos requiere un conocimiento profundo del dominio del problema y de las restricciones temporales del proceso de negocio.

6.5. Gobierno del dato.

El **gobierno del dato** (data governance) es el conjunto de políticas, procesos y responsabilidades que garantizan que los datos de una organización son gestionados de forma coherente, segura y de acuerdo con la normativa aplicable.

En el contexto del machine learning, el gobierno del dato es relevante por varios motivos:

- Privacidad y protección de datos. En Europa, el Reglamento General de Protección de Datos (RGPD) impone restricciones estrictas sobre el uso de datos personales. Los modelos entrenados con datos personales deben cumplir con este reglamento.
- Sesgo algorítmico. Si los datos de entrenamiento reflejan sesgos históricos (por ejemplo, decisiones discriminatorias en la concesión de préstamos), el modelo los reproducirá. Identificar y mitigar estos sesgos es una responsabilidad técnica y ética.
- Documentación y linaje del dato. Las organizaciones necesitan saber de dónde vienen sus datos, qué transformaciones han sufrido y quién es responsable de cada conjunto de datos.

Ejemplo práctico

Contexto: Una aseguradora entrena un modelo de tarificación con datos históricos de los últimos 20 años.

Planteamiento: El modelo aprende que ciertos códigos postales tienen mayor siniestralidad. Resulta que esos códigos postales coinciden con barrios de menor renta. Sin un marco de gobierno del dato (RGPD, política antidiscriminación, auditoría de sesgos), el modelo perpetuaría una discriminación histórica.

Resultado: Implantan revisiones de sesgo trimestrales y eliminan variables proxy de origen socioeconómico. El gobierno del dato no es burocracia: es lo que evita que un modelo técnicamente correcto produzca decisiones éticamente inaceptables.

Actividad de relacionar 15. Calidad del dato y trazabilidad.

Relaciona cada concepto de la columna A con su descripción de la columna B.

Columna A: Concepto.	Columna B: Descripción.
1. Exactitud del dato	Capacidad de reconstruir el historial completo de transformaciones aplicadas a los datos
2. Fuga de datos	El dato refleja correctamente la realidad
3. Trazabilidad	Error que produce métricas de evaluación artificialmente optimistas al filtrar información del test al entrenamiento
4. Data profiling	Análisis sistemático de la estructura, el contenido y la calidad de un conjunto de datos
5. Gobierno del dato	Conjunto de políticas y procesos para gestionar los datos de forma coherente, segura y conforme a la normativa

7. Caso práctico integrador.

Una cadena de supermercados con 120 tiendas en toda España te contrata para unificar y preparar sus datos antes de implementar un sistema de predicción de demanda. La empresa almacena información en múltiples fuentes: un ERP que exporta ventas en archivos CSV diarios, una base de datos SQL con el inventario, hojas de cálculo de Excel con las promociones planificadas por el equipo de marketing, y una API meteorológica que proporciona datos climáticos por código postal. Los datos presentan inconsistencias graves: códigos de producto diferentes entre sistemas, fechas en formatos distintos, valores faltantes en el 12 % de los registros y duplicados generados por errores en los terminales de punto de venta.

Tareas a desarrollar:

1. Diseña un pipeline ETL completo que extraiga datos de las cuatro fuentes, los transforme a un esquema unificado y los cargue en un almacén de datos analítico. Define las etapas de extracción, las transformaciones necesarias en cada paso y el esquema de destino. Justifica la elección entre un proceso batch diario y un enfoque en tiempo real.
2. Desarrolla una estrategia de limpieza de datos que aborde cada uno de los problemas identificados: diseña una tabla maestra de productos para unificar los códigos, estandariza los formatos de fecha, implementa al menos tres técnicas de imputación para los valores faltantes (eliminación, media/mediana, e imputación basada en K vecinos) y compara sus efectos en la distribución de los datos.
3. Detecta y gestiona los valores atípicos en las ventas diarias. Aplica al menos dos métodos (IQR y Z-score) e identifica si los outliers corresponden a errores de registro, promociones especiales o eventos estacionales. Argumenta cuáles deben eliminarse, cuáles transformarse y cuáles conservarse.
4. Implementa la ingeniería de características (feature engineering): crea variables derivadas relevantes para la predicción de demanda, como medias móviles de ventas, indicadores de festivos, variables cíclicas para el día de la semana y el mes, y variables de interacción entre promociones y climatología.
5. Documenta el pipeline con un diagrama de flujo y describe las validaciones de calidad de datos que implementarías en cada etapa. Propón un sistema de alertas que notifique al equipo cuando la calidad de los datos de entrada caiga por debajo de umbrales aceptables.

Este ejercicio integra el diseño de pipelines ETL, la limpieza y transformación de datos procedentes de múltiples fuentes, el tratamiento de valores faltantes y atípicos, y la ingeniería de características, competencias esenciales del módulo de preparación de datos.

Propuesta de solución

Tarea 1

Se diseña un pipeline ETL batch diario con tres capas: (1) Extracción: un script Python orquestado con Apache Airflow que a las 02:00 h descarga los CSV de ventas del servidor FTP del ERP, ejecuta consultas SQL al inventario mediante SQLAlchemy, lee las hojas de Excel de promociones desde un directorio compartido con openpyxl, y consume la API meteorológica (OpenWeatherMap) con peticiones por código postal y almacenamiento en caché local para respetar los límites de tasa.

(2) Transformación: se aplica en pandas con las siguientes etapas secuenciales — validación de esquema (verificar que las columnas esperadas existen y tienen el tipo correcto), unificación de códigos de producto mediante una tabla maestra (merge por código ERP con fallback por nombre de producto con fuzzy matching al 90 %), estandarización de fechas a formato ISO 8601, imputación de valores faltantes, deduplicación y creación de variables derivadas.

(3) Carga: escritura en un data warehouse PostgreSQL con esquema estrella (tabla de hechos: ventas diarias por tienda y producto; dimensiones: producto, tienda, tiempo, promoción, meteorología).

Se opta por batch diario en lugar de tiempo real porque los datos del ERP se generan al cierre de caja (23:00 h), los modelos de predicción de demanda se ejecutan con granularidad diaria, y el coste de infraestructura streaming (Kafka + Spark Streaming) no se justifica para el volumen actual (120 tiendas × 5 000 productos × 1 registro/día = 600 000 registros diarios).

Tarea 2

Unificación de códigos de producto: se construye una tabla maestra con tres columnas (código_erp, código_inventario, nombre_normalizado) poblada inicialmente con un join exacto sobre los 4 200 productos que comparten código.

Los 380 productos sin coincidencia se emparejan mediante fuzzy matching (librería fuzzywuzzy, umbral de similitud del 90 % sobre el nombre del producto), generando 340 emparejamientos automáticos validados manualmente y 40 casos que requieren resolución por el equipo de compras.

Estandarización de fechas: el ERP usa dd/mm/yyyy, el inventario yyyy-mm-dd y las hojas de Excel fechas serializadas de Excel (número de días desde 01/01/1900); se aplica pd.to_datetime con format explícito para cada fuente y se convierte todo a datetime64[ns] en UTC.

Imputación de valores faltantes: (a) eliminación: se descartan 142 registros con más del 50 % de campos nulos (0,02 % del total, impacto negligible); (b) media/mediana: las ventas faltantes en días laborables se imputan con la mediana de ventas del mismo día de la semana y tienda en las 4 semanas anteriores (la mediana es más robusta ante promociones puntuales que la media); (c) KNN imputation (k=5, distancia ponderada): para los campos de inventario faltantes, se imputa basándose en las 5 tiendas más similares en volumen y ubicación geográfica.

Comparación: la imputación KNN preserva mejor la distribución original (test de Kolmogórov-Smirnov, p=0,87 vs. p=0,42 para la mediana), pero es computacionalmente más costosa (12 minutos vs. 8 segundos para 600 000 registros).

Tarea 3

Se aplican dos métodos de detección de outliers sobre las ventas diarias por tienda y producto: (a) Método IQR: Q1=12 uds, Q3=45 uds, IQR=33, límite inferior = 12 − 1,5×33 = −37,5 (se trunca a 0), límite superior = 45 + 1,5×33 = 94,5.

Se identifican 2 340 registros por encima del límite superior (3,9 % del total). (b) Z-score robusto (basado en MAD): mediana = 28 uds, MAD = 15,2, umbral = |z| > 3.

Se identifican 1 890 registros (3,15 %).

El análisis cruzado con el calendario de promociones y eventos revela tres categorías: errores de registro (210 registros con ventas > 500 uds en tiendas pequeñas donde el stock máximo es 200 — se corrigen consultando los tickets reales en el ERP), promociones especiales (1 420 registros que coinciden con fechas de ofertas del 2×1 o Black Friday — se conservan tal cual y se etiquetan con la variable binaria "es_promocion=1"), y eventos estacionales (710 registros de productos de temporada como turrones en diciembre o helados en agosto — se conservan y se incorpora la variable "semana_del_año" como feature cíclica).

Decisión: se eliminan los 210 errores confirmados, se conservan los 2 130 restantes con sus etiquetas contextuales.

Tarea 4

Variables derivadas creadas: (1) Media móvil de ventas a 7 y 28 días por producto y tienda, calculada con pandas rolling(window=7).mean() — captura la tendencia reciente y la estacionalidad mensual.

(2) Indicador de festivos: variable binaria cruzando la fecha con un calendario de festivos nacionales y autonómicos (librería holidays de Python), más una variable "días_hasta_próximo_festivo" que captura el efecto de acumulación de compras previas.

(3) Variables cíclicas: el día de la semana se transforma con sin(2π×día/7) y cos(2π×día/7) en lugar de one-hot encoding, preservando la proximidad cíclica (el domingo está cerca del lunes, no al extremo opuesto de un vector de 7 dimensiones).

El mes se transforma análogamente con período 12.

(4) Variables de interacción promoción-climatología: "promocion × lluvia" (binaria: hubo promoción y llovió más de 5 mm ese día), que captura el efecto de reducción de afluencia a tiendas físicas durante promociones en días lluviosos, y "temperatura_desviación" (diferencia respecto a la media histórica del mes), que correlaciona con cambios en la demanda de productos estacionales.

Tarea 5

El diagrama de flujo del pipeline se estructura en cinco bloques con validaciones intermedias: Bloque 1 (Extracción) → Validación V1: comprobar que todos los archivos CSV se han descargado (120 archivos, uno por tienda), que la API meteorológica ha respondido para los 120 códigos postales y que la conexión SQL al inventario devuelve datos actualizados (fecha_última_actualización = ayer).

Bloque 2 (Validación de esquema) → V2: verificar tipos de datos, rango de valores aceptables (ventas ≥ 0, precio > 0, fecha dentro del mes actual) y completitud (porcentaje de nulos < 15 % por columna).

Bloque 3 (Limpieza y transformación) → V3: confirmar que la deduplicación reduce el conteo en menos del 2 % (si supera ese umbral, posible error en la extracción), que la imputación no altera la media en más de un 5 % y que las distribuciones post-limpieza pasan el test KS respecto a las del mes anterior.

Bloque 4 (Feature engineering) → V4: verificar que no hay valores NaN en las variables derivadas y que las correlaciones con la variable objetivo se mantienen en el rango esperado. Bloque 5 (Carga) → V5: confirmar que el número de registros cargados coincide con el esperado (±1 %).

Sistema de alertas: se implementa con Airflow + Slack.

Alertas críticas (bloquean el pipeline): fallo de extracción, caída de completitud > 20 %, nulos en campos clave.

Alertas de advertencia (el pipeline continúa, pero notifica): completitud entre 15 % y 20 %, desviación de la distribución > 10 %, aumento anómalo de duplicados.

Cada alerta incluye el bloque afectado, la métrica fuera de rango y un enlace al log detallado.

8. Resumen.

Esta unidad ha cubierto el proceso completo de preparación de datos, desde la extracción de las fuentes originales hasta la obtención de un dataset limpio, transformado e integrado, listo para ser utilizado en el entrenamiento de modelos de machine learning.

El proceso ETL —extracción, transformación y carga— es la columna vertebral de cualquier pipeline de datos. La extracción obtiene los datos desde fuentes heterogéneas: ficheros CSV o Excel, bases de datos relacionales o NoSQL, APIs web o páginas web mediante scraping. Cada tipo de fuente requiere herramientas específicas en Python.

La limpieza de datos es la etapa más laboriosa del proceso. Los problemas más habituales son los valores nulos (que pueden tratarse mediante eliminación o imputación), los duplicados (que se eliminan con `drop_duplicates()`), los errores tipográficos y los valores imposibles. Cada decisión de limpieza debe documentarse y justificarse.

La transformación de datos incluye la normalización o estandarización de variables numéricas, las transformaciones de distribución para reducir el sesgo, la codificación de variables categóricas (label encoding, one-hot encoding, target encoding) y la ingeniería de características, que permite crear nuevas variables que mejoren el rendimiento del modelo.

La integración de datos combina fuentes heterogéneas en un único conjunto coherente mediante joins, concatenaciones y agregaciones. Los problemas más habituales son las claves de unión con formatos distintos, la distinta granularidad entre fuentes y los valores contradictorios.

La calidad del dato se mide a través de dimensiones como la exactitud, la completitud, la consistencia, la unicidad, la validez y la integridad. El data profiling permite obtener una visión rápida del estado de los datos al inicio del proyecto. La trazabilidad garantiza que el proceso es reproducible y auditable.

La fuga de datos es uno de los errores más graves en machine learning. Para evitarla, las transformaciones deben ajustarse solo sobre los datos de entrenamiento, y los pipelines de Scikit-learn son la herramienta más adecuada para garantizar esta práctica.

Los conocimientos de esta unidad son imprescindibles para cualquier proyecto de machine learning real. En la siguiente unidad se utilizarán estos datos preparados para entrenar los primeros modelos de aprendizaje supervisado, no supervisado y semisupervisado.

9. Actividades de autoevaluación.

Instrucciones: Lee cada situación e identifica el concepto o técnica que aparece.

Actividad 1

Un analista recibe un dataset con registros de transacciones bancarias. Al inspeccionar el dataset, encuentra que la columna «importe» tiene un 15 % de valores nulos. Decide rellenar esos valores con la mediana de la columna para no perder información.

Identifica qué técnica está aplicando el analista y justifica brevemente por qué usa la mediana en lugar de la media.

Actividad

Un ingeniero de datos combina dos tablas: una con datos de clientes y otra con datos de pedidos. Quiere conservar todos los clientes en el resultado final, aunque algunos no tengan ningún pedido registrado. Los clientes sin pedidos aparecerán con valores nulos en las columnas de pedidos.

Identifica qué tipo de join está aplicando y por qué es el más adecuado para esta situación.

Actividad 3

Un modelo de machine learning para predecir si un cliente va a cancelar su suscripción obtiene una precisión del 99 % durante la evaluación, pero cuando se despliega en producción, su rendimiento cae al 65 %. Al investigar la causa, se descubre que una de las variables incluidas en el modelo era la fecha de cancelación efectiva, información que no estaría disponible en el momento real de hacer la predicción.

Identifica qué error se ha cometido en el proceso de preparación de datos.

Actividad 4

Una empresa tiene datos de clientes en tres sistemas distintos: el CRM, el sistema de facturación y la plataforma de atención al cliente. Cada sistema tiene su propia base de datos con su propio identificador de cliente. Antes de combinar las tres fuentes, hay que estandarizar los identificadores y resolver las inconsistencias entre campos que existen en más de una fuente.

Identifica en qué fase del proceso ETL se encuentra este trabajo y qué problemas habituales de integración aparecen en la situación descrita.

Actividad 5

Un científico de datos aplica un `StandardScaler` sobre todo el dataset —entrenamiento y test juntos— antes de dividirlo. Otro miembro del equipo le advierte de que esto puede dar lugar a métricas de evaluación poco fiables.

Identifica qué error está cometiendo y cómo debería corregirlo usando un pipeline de Scikit-learn.

10. Verdadero o falso.

Indica si cada afirmación es verdadera (V) o falsa (F).

- El proceso ETL consiste en extraer, transformar y cargar datos desde sus fuentes originales hasta el destino de análisis. ( )
- La imputación con la mediana es más robusta que la imputación con la media cuando existen outliers en la columna. ( )
- El one-hot encoding es la técnica más adecuada para codificar variables ordinales porque respeta el orden natural de las categorías. ( )
- Un inner join conserva todas las filas de ambas tablas, rellenando con nulos donde no hay correspondencia. ( )
- La fuga de datos se produce cuando información del conjunto de test influye en el proceso de entrenamiento del modelo. ( )
- El data profiling es un análisis sistemático de la estructura, el contenido y la calidad de un conjunto de datos. ( )
- Ajustar el StandardScaler sobre los datos de test además de sobre los de entrenamiento es una buena práctica que mejora la consistencia del escalado. ( )
- La normalización min-max transforma los valores de una variable para que tengan media 0 y desviación típica 1. ( )
- Los pipelines de Scikit-learn permiten encadenar transformaciones de forma reproducible y evitar la fuga de datos. ( )
- En el proceso ETL, la fase de transformación es siempre la más sencilla y rápida de implementar. ( )

11. Evaluación tipo test.

1. ¿Qué significan las siglas ETL en el contexto del procesamiento de datos?

a) Extracción, Traducción y Limpieza.

b) Extracción, Transformación y Carga.

c) Evaluación, Test y Log.

d) Exploración, Transformación y Limpieza.

2. ¿Cuál de las siguientes estrategias es más adecuada para tratar los valores nulos de una columna numérica con muchos valores atípicos?

a) Imputar con la media aritmética.

b) Eliminar toda la columna.

c) Imputar con la mediana.

d) Sustituir por cero.

3. ¿Qué tipo de join conserva todas las filas de la tabla izquierda, rellenando con nulos las columnas de la tabla derecha cuando no hay correspondencia?

a) Inner join.

b) Right join.

c) Full outer join.

d) Left join.

4. ¿Qué hace la transformación min-max a los datos?

a) Les aplica una transformación logarítmica para reducir el sesgo.

b) Los escala para que tengan media 0 y desviación típica 1.

c) Los escala para que queden en el rango [0, 1].

d) Los agrupa en intervalos de igual anchura.

5. Un analista descubre que su conjunto de datos tiene registros con la misma ciudad escrita como «Sevilla» y «sevilla». ¿Qué técnica es la más adecuada para corregir este problema?

a) Eliminar las filas con el valor en minúsculas.

b) Imputar los valores con la moda.

c) Aplicar `str.strip().str.title()` para estandarizar el formato.

d) Aplicar una transformación logarítmica.

6. ¿Cuál de los siguientes es un ejemplo de fuga de datos (data leakage)?

a) Entrenar el modelo con más datos de los necesarios.

b) Incluir en el modelo una variable que solo está disponible después del evento que se quiere predecir.

c) Normalizar las variables antes de entrenar el modelo.

d) Usar validación cruzada para evaluar el modelo.

7. ¿Para qué se utiliza un pipeline en Scikit-learn?

a) Para visualizar el flujo de datos en un gráfico.

b) Para ejecutar consultas SQL sobre un DataFrame de Pandas.

c) Para encadenar transformaciones y un modelo en un único objeto reproducible que evita la fuga de datos.

d) Para descargar datos desde una API REST.

8. ¿Cuál de las siguientes librerías de Python se utiliza habitualmente para conectarse a bases de datos relacionales?

a) BeautifulSoup.

b) SQLAlchemy.

c) pymongo.

d) requests.

9. ¿Qué dimensión de calidad del dato se refiere a que no existan registros duplicados innecesarios en el conjunto de datos?

a) Exactitud.

b) Completitud.

c) Unicidad.

d) Validez.

10. ¿Cuál de las siguientes afirmaciones sobre la ingeniería de características es correcta?

a) Consiste en eliminar las variables que no siguen una distribución normal.

b) Es el proceso de crear nuevas variables a partir de las existentes para mejorar el rendimiento del modelo.

c) Se aplica únicamente a variables de texto y nunca a variables numéricas.

d) Su objetivo principal es reducir el número de observaciones del dataset.

Modelos supervisados, no supervisados y semi.

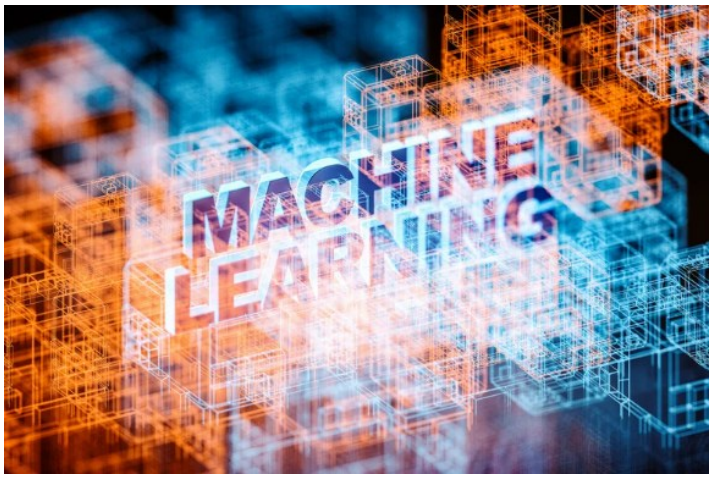
Una vez que los datos están limpios, transformados e integrados, llega el momento de construir el modelo. Esta es la unidad central del manual: aquí se introduce el aprendizaje automático como disciplina, se explican los tres grandes paradigmas de aprendizaje —supervisado, no supervisado y semisupervisado— y se describen los algoritmos más importantes de cada uno.

No todos los problemas de machine learning son iguales. A veces se quiere predecir un valor numérico, como el precio de una casa. Otras veces se quiere clasificar una observación en una categoría, como decidir si un correo es spam o no. En otros casos, no se tienen etiquetas y se quiere descubrir estructura oculta en los datos, como identificar grupos de clientes con comportamientos similares. Y en ocasiones se dispone de una mezcla de datos etiquetados y no etiquetados.

Cada tipo de problema requiere un tipo de modelo distinto. Esta unidad enseña a reconocer el tipo de problema, a elegir el modelo adecuado, a entender cómo funcionan los algoritmos más habituales y a evaluar si el modelo está funcionando correctamente.

1. Introducción al aprendizaje automático.

1.1. Qué es el aprendizaje automático.



El **aprendizaje automático** (en inglés, machine learning) es la rama de la inteligencia artificial que estudia cómo hacer que los sistemas informáticos aprendan a realizar tareas a partir de datos, sin ser programados explícitamente para cada tarea concreta.

La diferencia entre la programación tradicional y el machine learning es fundamental. En la programación tradicional, el programador define las reglas explícitamente: «si el importe de la transacción supera 10.000 euros Y el cliente no tiene historial previo de operaciones de ese tamaño, marcar como sospechosa». En machine learning, el sistema aprende esas reglas a partir de miles o millones de ejemplos de transacciones pasadas, sin que nadie las haya escrito manualmente.

Esta diferencia hace que el machine learning sea especialmente valioso en problemas donde:

- Las reglas son demasiado complejas para escribirlas a mano.
- Las reglas cambian con el tiempo y hay que actualizarlas continuamente.
- Los patrones están ocultos en grandes volúmenes de datos que un humano no podría procesar.
- La velocidad de respuesta requerida no permite intervención humana.

Ejemplo práctico

Contexto: Una pasarela de pagos procesa 5.000 transacciones por segundo y debe decidir en menos de 100 milisegundos si cada una es fraudulenta.

Planteamiento: Programar reglas a mano («si importe > 10.000 € y país distinto al habitual, bloquear») cubre solo casos obvios; los defraudadores aprenden a esquivarlas en semanas. En su lugar entrenan un modelo con 6 meses de transacciones etiquetadas como fraude/no fraude para que aprenda los patrones por sí mismo.

Resultado: El modelo detecta combinaciones sutiles (importe medio, hora del día, dispositivo, secuencia de compras) que ninguna regla manual habría capturado. Y se reentrena cada semana para seguir el ritmo de los defraudadores.