
# MANEJO DE VALORES PERDIDOS Y OUTLIERS EN DATA SCIENCE CON PYTHON

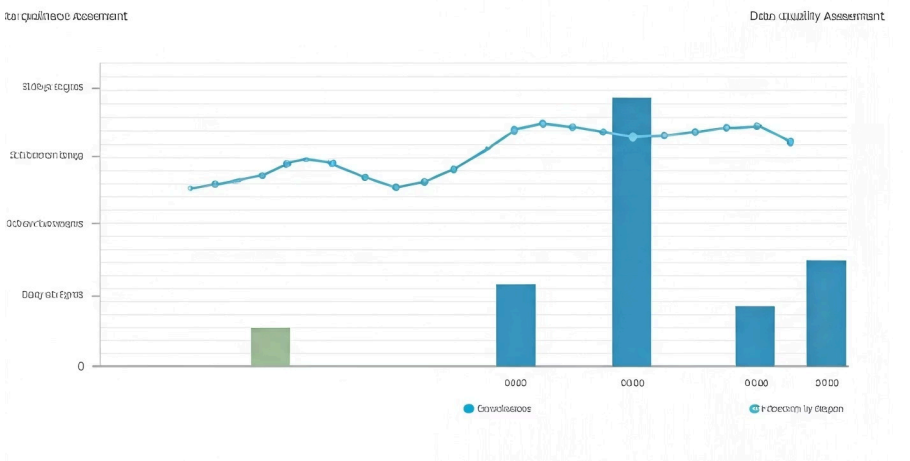
En todo proceso de análisis de datos, la calidad del dataset es un factor determinante. Datasets reales, provenientes de registros industriales, transacciones financieras, sensores o plataformas web, rara vez se presentan limpios y completos. Es común que contengan valores faltantes (missing values), inconsistencias y valores atípicos (outliers), todos los cuales pueden sesgar, distorsionar o incluso inutilizar los resultados de los análisis y modelos predictivos.

**por Kibernum Capacitación S.A.**

# Importancia de la calidad de los datos

En todo proceso de análisis de datos, la calidad del dataset es un factor determinante. Datasets reales, provenientes de registros industriales, transacciones financieras, sensores o plataformas web, rara vez se presentan limpios y completos.

Es común que contengan valores faltantes (missing values), inconsistencias y valores atípicos (outliers), todos los cuales pueden sesgar, distorsionar o incluso inutilizar los resultados de los análisis y modelos predictivos. Por ello, el manejo sistemático de valores perdidos y outliers es una etapa crítica en cualquier pipeline de ingeniería de datos, ciencia de datos o machine learning.



En la industria, la identificación y el tratamiento adecuado de estos problemas permiten evitar errores de interpretación, mejorar la robustez de los modelos y reducir riesgos en la toma de decisiones automatizadas o asistidas.

# ¿Qué es un valor perdido?

## Definición

Un valor perdido (missing value) es cualquier dato que, por diversas razones, no se encuentra presente en una observación o registro. La ausencia puede ser explícita (por ejemplo, una celda vacía en un archivo Excel) o implícita (por ejemplo, el uso de un marcador como NA, NULL, NaN, ?, o incluso un "-999" usado en ciertas industrias como "placeholder").

## Representación

Los valores perdidos pueden aparecer como celdas vacías, símbolos especiales (NA, NULL, NaN), o incluso valores numéricos específicos que actúan como marcadores (-999, 0, etc.) según las convenciones de cada industria o sistema.

# Causas típicas de valores perdidos

## Errores de captura

Fallos en dispositivos de entrada o errores humanos que impiden el registro correcto de la información.

## Falta de aplicabilidad

Campos que no corresponden a todos los casos (por ejemplo, "segundo apellido" en una base internacional).

## Pérdida de información

Deterioro, corrupción de archivos o transmisión incompleta que resulta en datos no disponibles.

## Política de empresa

Decisión deliberada de no recolectar ciertos datos por razones estratégicas o de privacidad.

# Ejemplo real de valores perdidos

Supongamos un dataset de clientes bancarios:

| ID | Nombre | Edad | Ingreso | Ciudad | Estado Civil |
|---|---|---|---|---|---|
| 1 | Ana | 32 | 2500 | Madrid | Soltera |
| 2 | Luis | | 3200 | | Casado |
| 3 | Marta | 28 | | Sevilla | |
| 4 | Pedro | 44 | 4100 | Málaga | Casado |

En este ejemplo, hay valores faltantes en "Edad", "Ingreso", "Ciudad" y "Estado Civil".

# Identificación de valores perdidos

Detectar valores perdidos es un paso fundamental antes de cualquier análisis, porque muchas funciones estadísticas y de aprendizaje automático no admiten valores nulos y pueden arrojar errores o resultados incorrectos.

Pandas es la librería estándar para manipulación de datos tabulares en Python. Permite identificar valores nulos mediante los métodos .isnull() y .isna(), que retornan un DataFrame de valores booleanos donde True indica la presencia de un valor faltante.

```
import pandas as pd

# Supongamos un DataFrame
data = {
    'Nombre': ['Ana', 'Luis', 'Marta', 'Pedro'],
    'Edad': [32, None, 28, 44],
    'Ingreso': [2500, 3200, None, 4100],
    'Ciudad': ['Madrid', None, 'Sevilla', 'Málaga'],
    'Estado Civil': ['Soltera', 'Casado', None, 'Casado']
}
df = pd.DataFrame(data)

# Identificar valores perdidos
print(df.isnull())
```

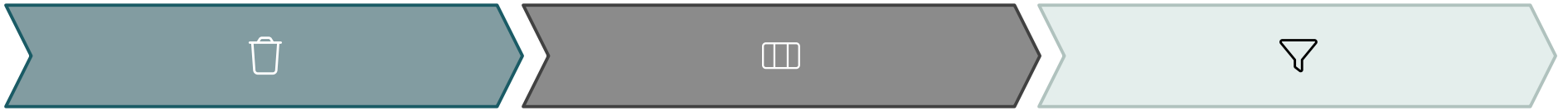
Para obtener el conteo de valores perdidos por columna:

```
print(df.isnull().sum())
```

En la práctica, muchas veces es necesario detectar tanto los None como otros marcadores especiales definidos por el usuario o la industria ('-999', 'NA', etc.), para lo cual read_csv() permite definir na_values.

# Filtrado de la data perdida

El filtrado de valores perdidos implica decidir cómo tratar las filas o columnas donde faltan datos: eliminarlas total o parcialmente, o continuar con su imputación. La decisión depende del contexto del problema, la cantidad de datos perdidos y el impacto en los análisis posteriores.



## Eliminación de filas

```
df_sin_na = df.dropna()
```

Elimina todas las filas que tengan al menos un valor faltante.

## Eliminación de columnas

```
umbral = 0.3
df_filtrado = df.dropna(axis=1,
thresh=int((1-umbral)*len(df)))
```

Elimina columnas si más del 30% de sus valores están perdidos.

## Eliminación condicional

```
df.dropna(how='all')   # Solo si toda
la fila está vacía
df.dropna(how='any')   # Si alguna
columna está vacía
```

**Consideraciones prácticas:**

- Ventaja: Sencillez, no introduce suposiciones adicionales.
- Desventaja: Si hay muchos valores perdidos, puedes perder demasiada información, generando sesgos.

# Imputación de datos

Cuando eliminar datos no es viable, la alternativa es la imputación, es decir, reemplazar los valores faltantes por estimaciones razonables. La imputación puede ser tan simple como usar la media o mediana, o tan sofisticada como emplear modelos de machine learning.

## Estrategias clásicas de imputación

**1. Imputación por media, mediana o moda:**

Para variables numéricas continuas, se puede usar la media o la mediana:

```
# Imputar con la media
df['Edad'] = df['Edad'].fillna(df['Edad'].mean())

# Imputar con la mediana
df['Ingreso'] = df['Ingreso'].fillna(df['Ingreso'].median())
```

Para variables categóricas, la moda:

```
df['Ciudad'] = df['Ciudad'].fillna(df['Ciudad'].mode()[0])
```

**Ventajas:** Fácil de implementar, rápido, no requiere modelos complejos.

**Desventajas:** Puede subestimar la varianza, introducir sesgos si los datos no son MCAR (Missing Completely At Random).

**2. Imputación condicional:**

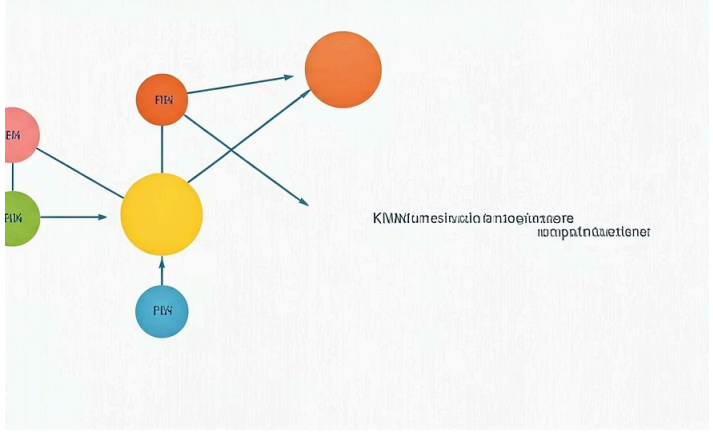
A veces, es preferible imputar usando la media o moda dentro de subgrupos (por ejemplo, por "Estado Civil" o "Ciudad"):

```
df['Edad'] = df.groupby('Ciudad')['Edad'].transform(lambda
x: x.fillna(x.mean()))
```

# Imputación mediante modelos

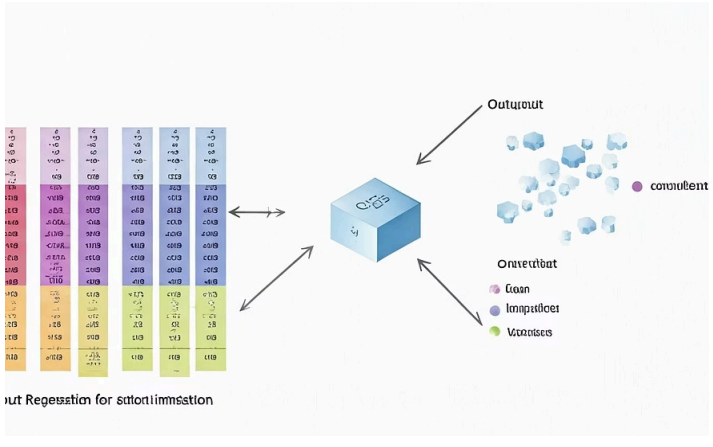
Cuando la relación entre las variables es compleja, se puede usar modelos de machine learning para predecir los valores faltantes (regresión, clasificación, KNN):

```
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=3)
df[['Edad', 'Ingreso']] = imputer.fit_transform(df[['Edad', 'Ingreso']])
```



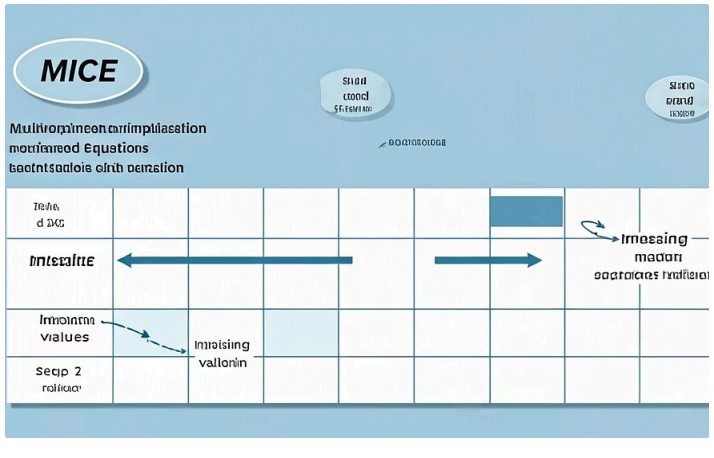
## KNN Imputer

Utiliza los K vecinos más cercanos para estimar el valor faltante basándose en la similitud entre observaciones.




## Regresión múltiple

Entrena un modelo para predecir el valor faltante usando las otras variables como predictores.




## MICE

Multiple Imputation by Chained Equations: Técnica avanzada que imputa múltiples veces para capturar la incertidumbre del proceso.

# Imputación de valores cualitativos

Las variables cualitativas (categóricas), como "Estado Civil" o "Ciudad", requieren estrategias distintas, pues no se pueden promediar.

## Imputación por moda

Es la estrategia más común.

```
df['Estado Civil'] = df['Estado Civil'].fillna(df['Estado Civil'].mode()[0])
```

## Creación de nueva categoría

Se puede introducir una categoría especial como "Desconocido" o "Sin especificar".

```
df['Ciudad'] = df['Ciudad'].fillna('Desconocido')
```

## Imputación basada en similitud

Usar algoritmos como KNN o análisis de correspondencias para imputar categorías considerando variables relacionadas.

## Modelos predictivos

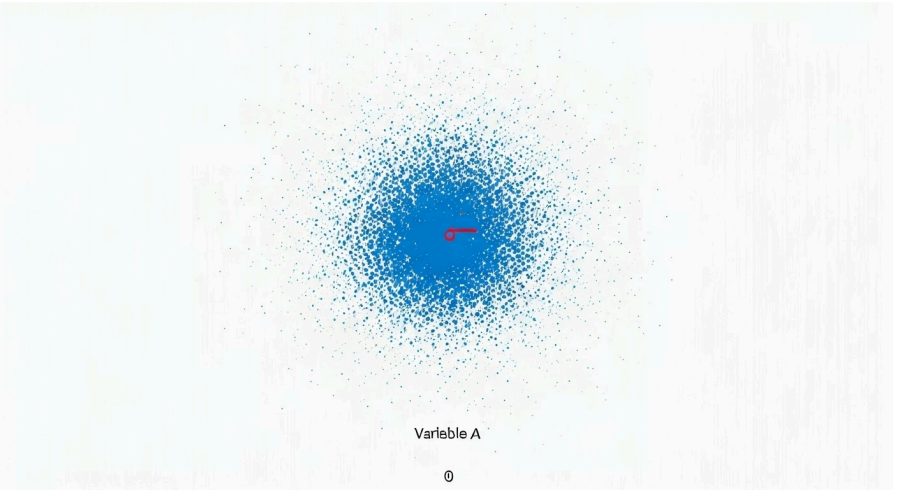
Cuando las categorías tienen estructura ordinal o dependencias, usar modelos como árboles de decisión.

# ¿Qué es un outlier?

Un outlier o valor atípico es una observación cuyo valor se desvía significativamente de la tendencia general de los datos. Los outliers pueden surgir por errores de medición, fraude, procesos excepcionales, o pueden ser comportamientos reales pero raros.

## Ejemplo típico

En un dataset de ingresos anuales de empleados, donde la mayoría gana entre 20,000 y 60,000 dólares, un ingreso reportado de 500,000 puede ser considerado un outlier.




## Tipos de outliers

- Univariantes: Valores extremos en una sola variable.
- Multivariantes: Combinaciones raras de valores en varias variables.
- Contextuales: Valores que son atípicos bajo ciertas condiciones (por ejemplo, temperatura inusualmente alta solo en una estación).

# Detección y filtrado de outliers

Detectar y tratar outliers es clave para evitar que distorsionen análisis estadísticos, especialmente medidas como la media o la varianza.

## Métodos estadísticos simples

Regla de 1.5 x IQR (Interquartile Range): Un valor es considerado outlier si está fuera del rango: [Q1 - 1.5 * IQR, Q3 + 1.5 * IQR], donde IQR = Q3 - Q1.

```
Q1 = df['Ingreso'].quantile(0.25)
Q3 = df['Ingreso'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['Ingreso'] < Q1 - 1.5 * IQR) | (df['Ingreso'] > Q3 + 1.5 * IQR)]
```

## Z-score

Un dato es outlier si su puntaje z (desviaciones estándar respecto a la media) es mayor que 3 o menor que -3.

```
from scipy import stats
import numpy as np

z_scores = np.abs(stats.zscore(df['Ingreso'].dropna()))
outliers = df[z_scores > 3]
```

## Visualización

Boxplots: Permiten visualizar rápidamente valores atípicos.

Scatter plots: Para ver relaciones entre dos variables y detectar outliers multivariantes.

## Métodos avanzados:

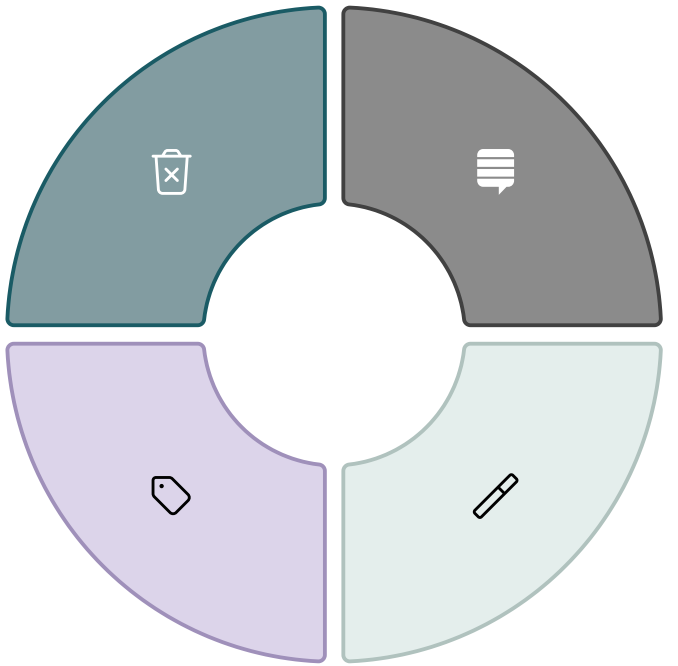
Isolation Forest, DBSCAN, LOF (Local Outlier Factor): Algoritmos de machine learning que detectan outliers en datasets complejos y multidimensionales.

# Filtrado y tratamiento de outliers

## Eliminación

Quitar los outliers si se consideran errores o no son relevantes para el análisis.

```
# Eliminar outliers según IQR
df = df[(df['Temperatura'] >= Q1 - 1.5 * IQR) &
(df['Temperatura'] <= Q3 + 1.5 * IQR)]
```



## Transformación

Aplicar transformaciones matemáticas para reducir el impacto de los valores extremos.

- Transformación logarítmica
- Winsorización (recortar valores extremos)
- Escalado robusto

## Etiquetado

Mantener los outliers pero marcarlos con una variable indicadora para análisis específicos.

## Imputación

Reemplazar outliers por valores más razonables, como la mediana o percentiles específicos.

La elección del método depende del contexto del problema, la naturaleza de los datos y el objetivo del análisis.

# Ejemplo integrado: Pipeline de tratamiento

Supongamos que tenemos un dataset de sensores industriales con variables numéricas y categóricas, donde se presentan ambos problemas.

## Identificación de valores perdidos y outliers

```
import pandas as pd
from sklearn.impute import
SimpleImputer
import numpy as np

# Cargar dataset
df = pd.read_csv('sensores.csv')

# Identificar valores perdidos
print(df.isnull().sum())

# Identificar outliers en la variable
'Temperatura'
Q1 =
df['Temperatura'].quantile(0.25)
Q3 =
df['Temperatura'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['Temperatura'] <
Q1 - 1.5 * IQR) |
          (df['Temperatura'] > Q3 +
1.5 * IQR)]
```

## Filtrado de datos perdidos y outliers

```
# Eliminar filas con más de 2
valores nulos
df = df[df.isnull().sum(axis=1) <= 2]

# Eliminar outliers
df = df[(df['Temperatura'] >= Q1 -
1.5 * IQR) &
      (df['Temperatura'] <= Q3 + 1.5
* IQR)]
```

## Imputación

```
# Imputar valores numéricos con
la mediana
imputer =
SimpleImputer(strategy='median')
df['Temperatura'] =
imputer.fit_transform(df[['Temper
atura']])

# Imputar valores categóricos con
la moda
imputer_cat =
SimpleImputer(strategy='most_fre
quent')
df['Estado'] =
imputer_cat.fit_transform(df[['Esta
do']])
```

# Actividades y desafíos comunes en la práctica

### 1 Evaluar el mecanismo de generación de los valores perdidos

¿Son MCAR (completamente aleatorios), MAR (aleatorios condicionados a otra variable), o NMAR (no aleatorios)? Esto afecta la estrategia de imputación.

### 2 Analizar el impacto de eliminar datos

Si eliminas muchas filas, ¿el dataset pierde representatividad?

### 3 Diseñar un pipeline reproducible

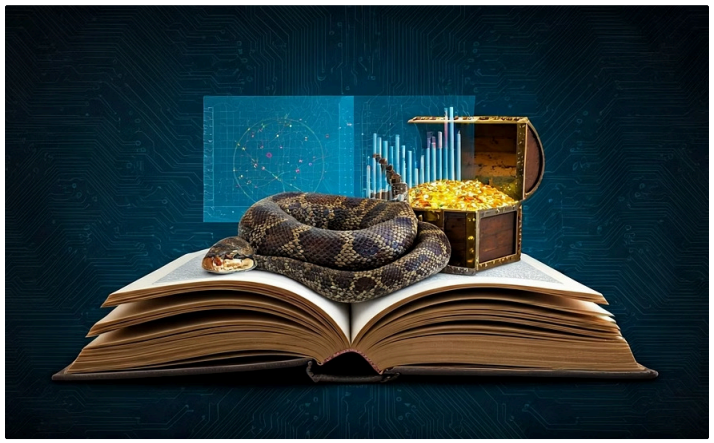
Usar funciones y scripts parametrizados para garantizar que el tratamiento de valores perdidos y outliers sea consistente en producción.

## Conexión con otras áreas

El manejo de valores perdidos y outliers es transversal en ciencia de datos, ingeniería de datos, machine learning y estadística aplicada. Por ejemplo:

- Machine learning: Muchos algoritmos no aceptan valores nulos; algunos, como los árboles de decisión, toleran outliers mejor que otros, como la regresión lineal.
- Visualización: Detectar valores extremos en gráficos ayuda a decidir si son errores o patrones interesantes.
- Big Data: En datasets masivos, el impacto de los valores perdidos y outliers puede ser más sutil, pero igual de relevante.
- Producción e integración de datos: El pipeline de datos debe incluir rutinas automáticas para el manejo de estos casos, antes de entrenar modelos o desplegar dashboards.

# Recursos y referencias recomendadas



## Python for Data Analysis

Wes McKinney

## Practical Statistics for Data Scientists

Peter Bruce & Andrew Bruce


## Hands-On Machine Learning

Aurélien Géron

- Documentación oficial de Pandas y Scikit-learn, **https://scikit-learn.org/**
- Libro: Mathematics for Machine Learning, Deisenroth, Faisal, Ong
- Libro: Introduction to Linear Algebra, Gilbert Strang

# Técnicas avanzadas de imputación

El campo de la imputación ha evolucionado más allá de los enfoques tradicionales como la media, la mediana o la moda. En la práctica profesional y científica, los siguientes métodos permiten abordar escenarios donde la estructura de los datos y el mecanismo de pérdida es más complejo.

## Imputación multivariante: MICE

MICE (Multiple Imputation by Chained Equations) es una técnica estadística que permite imputar valores faltantes considerando la correlación entre múltiples variables. La idea es iterar secuencialmente sobre las variables con valores perdidos, utilizando modelos de regresión para predecir e imputar cada columna en función de las otras, repitiendo el proceso varias veces para capturar la incertidumbre.

## Modelos supervisados de imputación

Utilizar modelos de machine learning para predecir valores faltantes a partir de las variables presentes es una técnica poderosa, especialmente cuando existen relaciones complejas no capturadas por medidas simples.

## Imputación con modelos generativos y deep learning

En contextos de Big Data, imágenes o series temporales complejas, se utilizan modelos más sofisticados, como autoencoders (redes neuronales) y modelos generativos bayesianos.

## Ventajas de MICE:

- Genera múltiples datasets imputados, permitiendo estimar la variabilidad introducida por la imputación.
- Utiliza toda la información disponible y considera relaciones entre variables.

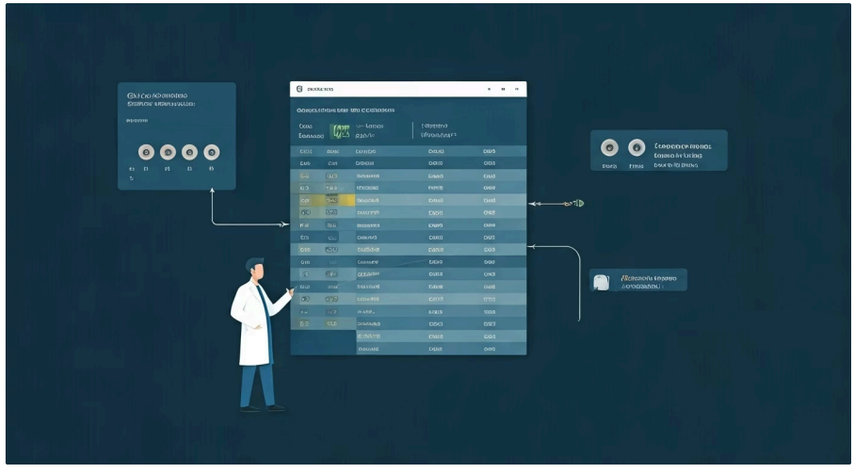
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
import pandas as pd

# Simulación de datos
df = pd.DataFrame({
    'edad': [23, 31, None, 35, 44],
    'ingresos': [2500, 3400, 3200, None, 4100],
    'gasto': [800, 1200, 1100, 1150, None]
})

imputer = IterativeImputer(random_state=0)
df_imputado = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

# Imputación basada en aprendizaje automático

## Modelos supervisados de imputación

Utilizar modelos de machine learning para predecir valores faltantes a partir de las variables presentes es una técnica poderosa, especialmente cuando existen relaciones complejas no capturadas por medidas simples.

- **Regresión lineal/múltiple:** Para valores numéricos, entrenar un modelo de regresión usando los registros completos para predecir el valor ausente.
- **Árboles de decisión:** Útiles para variables numéricas o categóricas, capturan relaciones no lineales.
- **K-Nearest Neighbors (KNN):** Imputa un valor usando el promedio (o moda) de los vecinos más cercanos (según la distancia en el espacio de variables observadas).



## Ejemplo de imputación con KNN:

```
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5)
df_imputado = pd.DataFrame(
    imputer.fit_transform(df),
    columns=df.columns
)
```

**Ventajas:**

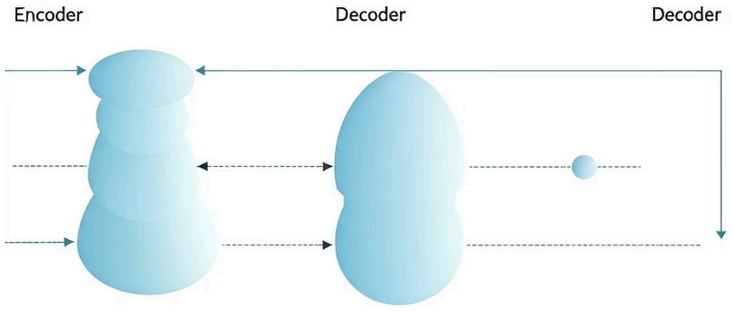
- Puede imputar variables numéricas y categóricas (tras codificación).
- Aprovecha patrones complejos y relaciones multidimensionales.

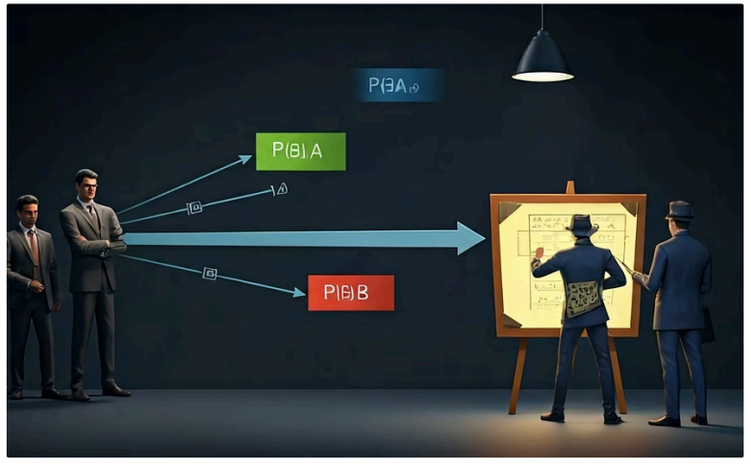
**Desventajas:**

- Puede ser computacionalmente costoso en datasets grandes.
- Puede requerir preprocesamiento adicional, como normalización.

# Imputación con modelos generativos y deep learning

En contextos de Big Data, imágenes o series temporales complejas, se utilizan modelos más sofisticados, como:




## Autoencoders (Redes neuronales)

Se entrenan para reconstruir las entradas, aprendiendo representaciones latentes. Una vez entrenados, pueden rellenar valores ausentes generando reconstrucciones plausibles.

## Modelos generativos bayesianos

Permiten inferir distribuciones sobre los datos y muestrear valores plausibles para los faltantes.

## Ejemplo conceptual con autoencoders:

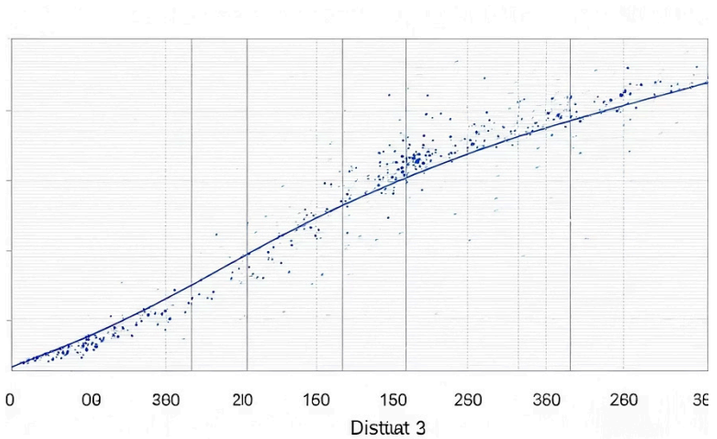
En el caso de series temporales médicas, un autoencoder entrenado puede imputar valores de presión sanguínea ausentes en registros históricos, aprovechando la dinámica temporal y la correlación con otras variables fisiológicas.

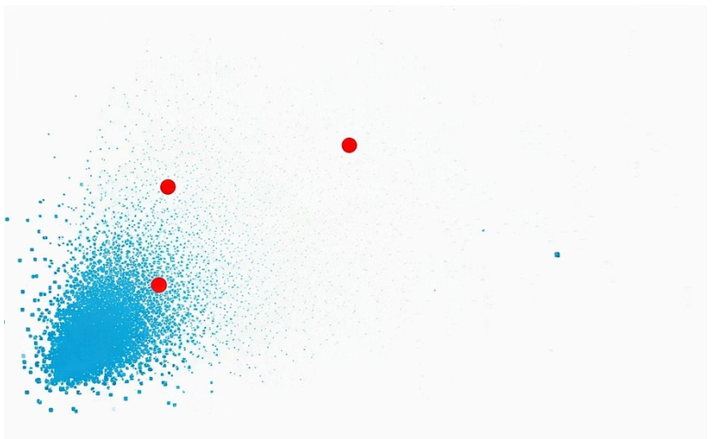
Estos métodos son especialmente útiles cuando:

- Los datos tienen estructura compleja (imágenes, audio, texto)
- Existen dependencias temporales o espaciales
- Las relaciones entre variables son altamente no lineales
- Se requiere preservar la distribución original de los datos

# Visualización de outliers

La visualización es una de las herramientas más efectivas para comprender la presencia, naturaleza y posible origen de los outliers. Los gráficos permiten tanto la identificación visual de valores atípicos como la comunicación de estos hallazgos a equipos no técnicos.



## Boxplots (diagramas de caja y bigotes)

El boxplot es el gráfico más clásico para detectar outliers univariantes. Representa la mediana, los cuartiles (Q1 y Q3), y visualiza explícitamente los puntos que se consideran atípicos (usualmente, los que están más allá de 1.5 x IQR).

```
import matplotlib.pyplot as plt

plt.boxplot(df['ingresos'].dropna())
plt.title('Boxplot de Ingresos')
plt.ylabel('Ingresos')
plt.show()
```

## Diagramas de dispersión (scatter plots)

Para relaciones bivariadas, permiten observar puntos que se apartan de la nube principal. También son útiles para detectar outliers multivariantes.

```
plt.scatter(df['edad'], df['ingresos'])
plt.title('Ingresos vs Edad')
plt.xlabel('Edad')
plt.ylabel('Ingresos')
plt.show()
```

## Histogramas y density plots

Visualizan la distribución de una variable y ayudan a detectar valores extremos que no siguen el patrón general.

```
df['ingresos'].hist(bins=20)
plt.title('Histograma de Ingresos')
plt.show()
```

# Visualización multivariante y técnicas avanzadas

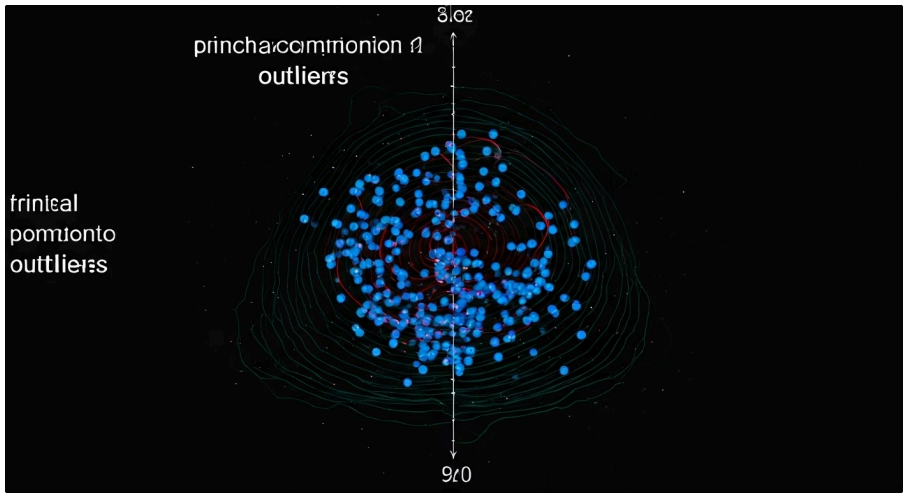
## PCA (Análisis de Componentes Principales)

Proyecta datos multidimensionales en 2D/3D y permite visualizar outliers que no serían detectables univariadamente.

## t-SNE/UMAP

Técnicas de reducción de dimensionalidad para detectar agrupaciones y outliers complejos.

**Ejemplo conceptual:**
En datasets de fraudes bancarios, se utiliza t-SNE para proyectar transacciones y visualizar operaciones sospechosas alejadas del clúster normal de comportamiento.




## Visualización interactiva

Librerías como plotly o bokeh permiten construir dashboards interactivos para explorar outliers, filtrar registros, y vincular la visualización con la tabla de datos real.

Estas técnicas avanzadas son especialmente útiles cuando:

- Los datos tienen alta dimensionalidad
- Los outliers no son evidentes en visualizaciones univariantes
- Se necesita explorar interactivamente grandes volúmenes de datos
- Se requiere comunicar patrones complejos a stakeholders no técnicos

# Pipelines automatizados en producción

El tratamiento de valores perdidos y outliers debe formar parte de un pipeline reproducible y automatizable, especialmente en contextos de integración continua (CI), despliegue de modelos o actualizaciones de datos periódicas.

### Carga y validación de datos

Leer datos, validar esquema, tipos y presencia de variables clave.

### Detección de valores perdidos y outliers

Utilizar funciones automatizadas para análisis exploratorio.

### Filtrado y limpieza automática

Aplicar reglas de eliminación condicional (porcentaje de nulos, outliers extremos).

### Imputación parametrizada

Definir estrategias de imputación reproducibles (con seeds, registro de hiperparámetros).

### Registro y monitoreo

Loggear la cantidad de registros afectados, imputaciones realizadas y outliers eliminados.

### Exportación y versionado

Guardar los datasets transformados para auditoría o uso posterior.

## Ejemplo de pipeline con Scikit-learn y Pandas

Scikit-learn ofrece la clase Pipeline que permite encadenar procesos de preprocesamiento e imputación.

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

pipeline = Pipeline([
    ('imputacion', SimpleImputer(strategy='median')),
    ('escalado', StandardScaler())
])

df['ingresos_procesados'] = pipeline.fit_transform(df[['ingresos']])
```

# Implementación en producción: buenas prácticas

## Testing y validación continua

Automatizar tests para asegurar que el pipeline maneja correctamente casos límite (muchos nulos, aparición de nuevas categorías, valores extremos inesperados).

## Configuración externa

Definir los parámetros (umbral de outliers, estrategias de imputación) en archivos de configuración para facilitar ajustes sin modificar el código.

## Alertas y monitoreo

Incorporar sistemas de alerta si se detecta un incremento inusual en la proporción de nulos o outliers, lo que puede indicar fallas en origen o fraude.

## Versionado de datos y modelos

Usar sistemas como DVC (Data Version Control) o MLflow para registrar qué versión de los datos y del pipeline produjo cada resultado/modelo.

# Ejemplo conceptual de pipeline end-to-end

Supongamos un sistema de monitoreo de sensores industriales, donde diariamente se descargan archivos de registros con potenciales valores perdidos y outliers.

### 1 Ingesta de datos

El pipeline ingiere automáticamente los archivos de sensores, valida el esquema y los tipos de datos.

### 2 Análisis de calidad

Se ejecuta un análisis de calidad, calculando la proporción de valores perdidos y reportando los sensores con mayor cantidad de datos faltantes.

### 3 Limpieza de datos

El pipeline elimina registros con más de 50% de nulos y outliers extremos según IQR.

### 4 Imputación

Imputa valores restantes usando KNN para numéricos y la moda para categóricos.

### 5 Exportación y registro

El resultado se exporta a una base de datos y se almacena un log de todas las transformaciones realizadas.

## Código simplificado de un pipeline:

```
import pandas as pd
from sklearn.impute import KNNImputer, SimpleImputer

def preprocesar_sensores(filepath):
    df = pd.read_csv(filepath)
    # Eliminar registros con más de 50% nulos
    df = df[df.isnull().mean(axis=1) < 0.5]

    # Filtrado de outliers
    for col in ['temperatura', 'presion']:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        df = df[(df[col] >= Q1 - 1.5 * IQR) & (df[col] <= Q3 + 1.5 * IQR)]

    # Imputación
    num_cols = ['temperatura', 'presion']
    cat_cols = ['estado']
    df[num_cols] = KNNImputer(n_neighbors=3).fit_transform(df[num_cols])
    df[cat_cols] = SimpleImputer(strategy='most_frequent').fit_transform(df[cat_cols])

    return df
```

# Conexión final y reflexión

## Hacia sistemas confiables y escalables

El desarrollo de pipelines automáticos para la gestión de valores perdidos y outliers es un pilar de la ingeniería de datos moderna. Permite construir sistemas confiables, auditables y escalables, facilitando la integración entre equipos de datos, operaciones y negocio.

Invertir en la implementación, monitoreo y mejora continua de estos procesos garantiza que los modelos y análisis se realicen siempre sobre datos de calidad, minimizando riesgos y maximizando el valor de la información.

## Reflexión final

La correcta identificación y tratamiento de valores perdidos y outliers no solo es un paso técnico, sino una práctica esencial para la calidad y confiabilidad de los sistemas analíticos. Invertir tiempo en este proceso, usando las herramientas que ofrece Python y sus librerías, es clave para desarrollar soluciones robustas, escalables y alineadas con los objetivos del negocio o la investigación científica.

Documentación Scikit-learn

Más recursos