Preprocesamiento y Limpieza de Datos en Ciencia de Datos

1. ¿Qué es el Preprocesamiento y la Limpieza de Datos?

El preprocesamiento de datos consiste en **preparar, limpiar y transformar** los datos antes de su análisis o uso en modelos de machine learning. Es un proceso **crítico**, porque la calidad de los datos afecta directamente la calidad de los resultados.

Limpieza es el subconjunto más básico del preprocesamiento.

¿Por qué es importante?

- Elimina errores, inconsistencias y valores extraños.
- Facilita el trabajo de los algoritmos.
- Aumenta la precisión y confiabilidad de los modelos.

Herramientas clave:

`pandas`, `numpy`, `scikit-learn`

2. Flujo de Trabajo del Preprocesamiento

2.1. Exploración inicial de los datos

Antes de limpiar, es esencial **conocer los datos**:

¿Qué columnas hay? ¿Qué tipos? ¿Valores nulos? ¿Errores o incoherencias?

```
import pandas as pd

# DataFrame de ejemplo
df = pd.DataFrame({
    'Edad': [23, 25, None, 22, 120, 25],
    'Genero': ['M', 'F', 'F', None, 'F', 'M'],
    'Ingresos': [40000, 42000, 38000, 50000, None, 42000]
})
```

```
print(df)
print(df.info())
print(df.describe(include='all'))
```

> ¿Para qué sirve esto?
>
> Detectar valores faltantes, posibles errores, tipos de datos y planificar limpieza.

2.2. Limpieza de Datos

a) Manejo de valores faltantes

- **Eliminar** filas/columnas incompletas
- **Imputar** valores faltantes (media, mediana, moda, otros)

```
from sklearn.impute import SimpleImputer

# Eliminar filas con al menos un valor nulo
df_limpio = df.dropna()
print("Filas sin nulos:\n", df_limpio)

# Imputar la columna 'Edad' con la media
imputer = SimpleImputer(strategy='mean')
df['Edad'] = imputer.fit_transform(df[['Edad']])
print("Con imputación en Edad:\n", df)
```

> ¿Por qué hacerlo?
>
> Los algoritmos de ML no funcionan con datos vacíos; la estrategia depende de la importancia y cantidad de datos perdidos.

---

b) Eliminar duplicados

```
# Añadimos un duplicado
df = pd.concat([df, df.iloc[[0]]], ignore_index=True)
print("Con duplicado:\n", df)

# Eliminar duplicados
df_sin_duplicados = df.drop_duplicates()
print("Sin duplicados:\n", df_sin_duplicados)
```

> ¿Por qué hacerlo?
>
> Los duplicados pueden sesgar análisis o aprendizaje.

---

c) Detección y manejo de valores atípicos (outliers)

Puedes hacerlo manualmente, estadísticamente, o visualizando.

```
import numpy as np

# Ejemplo manual: filtrar edades < 100
df = df_sin_duplicados
print(df)
df_filtrado = df[df['Edad'] < 100]
print("Sin outliers extremos en Edad:\n", df_filtrado)
```

Visualización:

```
import matplotlib.pyplot as plt

df_filtrado['Edad'].hist()
plt.title('Distribución de Edades')
plt.xlabel('Edad')
plt.ylabel('Frecuencia')
plt.show()

df_filtrado.boxplot(column='Ingresos')
plt.title('Boxplot de Ingresos')
plt.show()
```

> ¿Por qué hacerlo?
>
> *Outliers* extremos pueden distorsionar estadísticas o afectar modelos.

---

d) Corrección de tipos de datos

```
# Supón que Edad o Ingresos son string (simulemos el error)
df_filtrado['Edad'] = df_filtrado['Edad'].astype(str)
print(df_filtrado.dtypes)

df_filtrado['Edad'] = pd.to_numeric(df_filtrado['Edad'])
print("Tipos corregidos:\n", df_filtrado.dtypes)
```

2.3. Transformación de Datos

a) Normalización y escalado

Poner las variables numéricas en la misma escala.

Muy importante para KNN, SVM, redes neuronales, etc.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = MinMaxScaler()
df_filtrado['Ingresos_norm'] = scaler.fit_transform(df_filtrado[['Ingresos']])
print("Ingresos normalizados:\n", df_filtrado)

scaler_std = StandardScaler()
df_filtrado['Ingresos_std'] = scaler_std.fit_transform(df_filtrado[['Ingreso
s']])
print("Ingresos estandarizados:\n", df_filtrado)
```

> ¿Por qué hacerlo?
>
> Variables en distintas magnitudes afectan la interpretación y el funcionamiento de muchos modelos.

---

b) Codificación de variables categóricas

Muchos modelos no pueden trabajar con texto, sólo con números.

```
# One-hot encoding
df_onehot = pd.get_dummies(df_filtrado, columns=['Genero'])
print(df_onehot)

# Codificación por etiquetas
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
df_filtrado['Genero_num'] = encoder.fit_transform(df_filtrado['Genero'].asty
pe(str))
print("Con codificación de etiquetas:\n", df_filtrado)
```

---

c) Limpieza y transformación de texto (básico)

```
df_texto = pd.DataFrame({'comentario': [
    "¡Me encantó el producto!",
    "Muy malo... no lo recomiendo.",
    "   Buen servicio, volveré.   "
]})

df_texto['comentario_limpio'] = (
    df_texto['comentario'].str.lower()
    .str.strip()
    .str.replace(r'[^\w\s]', '', regex=True)
)
print(df_texto)
```

2.4. Ingeniería de características

Crear nuevas variables a partir de las existentes para mejorar el modelo.

```
# Agreguemos una columna de fechas para el ejemplo
df_filtrado['Fecha_ingreso'] = pd.to_datetime(['2022-01-01', '2021-05-10',
'2020-07-15', '2023-03-20'])
```

```
# Extraer el año y el mes
df_filtrado['Anio'] = df_filtrado['Fecha_ingreso'].dt.year
df_filtrado['Mes'] = df_filtrado['Fecha_ingreso'].dt.month
print(df_filtrado)
```

2.5. División del conjunto de datos

Separar en train y test para evaluar modelos.

```
from sklearn.model_selection import train_test_split

X = df_onehot.drop(['Ingresos', 'Ingresos_norm'], axis=1)
y = df_onehot['Ingresos_norm']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
tate=42)

print("X_train:\n", X_train)
print("y_train:\n", y_train)
```

2.6. Selección de características

Eliminar variables irrelevantes o redundantes.

```
from sklearn.feature_selection import SelectKBest, f_regression

# Selecciona las 2 características más relevantes respecto a 'Ingresos_nor
m'
selector = SelectKBest(score_func=f_regression, k=2)
X_new = selector.fit_transform(X_train, y_train)
print(X_new)
```

2.7. Pipeline de preprocesamiento

Organizar todos los pasos en un pipeline reproducible.

```
from sklearn.pipeline import Pipeline
```

```
pipeline = Pipeline([
    ('imputar', SimpleImputer(strategy='mean')),
    ('escalar', StandardScaler())
])
# Se usaría sobre variables numéricas, antes de modelar
```

2.8. Manejo de datos desbalanceados (opcional)

Si la variable objetivo está desbalanceada (ejemplo, clasificación), usar técnicas como SMOTE o submuestreo.

```
# from imblearn.over_sampling import SMOTE
# smote = SMOTE()
# X_res, y_res = smote.fit_resample(X_train, y_train)
```

---

3. Conclusión

El preprocesamiento y la limpieza de datos es un **proceso iterativo**:

analiza, limpia, transforma, visualiza y valida.

Repite hasta tener un conjunto de datos adecuado para modelar.

---

4. Resumen visual del flujo

1. **Explorar y visualizar datos**
2. **Limpieza**: nulos, duplicados, tipos, outliers
3. **Transformar**: escalar, codificar, text cleaning
4. **Ingeniería de variables**
5. **División train/test**
6. **Selección de variables**
7. **Pipelines y balanceo**