Módulo 6.4: Limpieza de datos

Prof. Carlos Cedeño

1. El Desafío de los Datos (Messy Data)

Rara vez recibirás un conjunto de datos perfecto. Lo más común es encontrar problemas. Analizaremos el registro de pasajeros del Titanic.

Para cargar el dataset, usaremos la librería `seaborn`, que lo incluye por defecto (si no la tienes, instálala con `pip install seaborn`).

```
import pandas as pd
import numpy as np
import seaborn as sns # Usamos seaborn para cargar el dataset fácilmente

# Cargar el dataset del Titanic
titanic = sns.load_dataset('titanic')

print(titanic.head())
```

Si observas las primeras filas, ya podemos ver los desafíos:

- **Valores Faltantes:** La columna `age` (edad) y `deck` (cubierta) tienen valores `NaN`.
- **Tipos de Datos Mixtos:** Hay columnas numéricas (`age`, `fare`), de texto (`sex`, `embark_town`) y categóricas (`pclass`, `survived`).
- **Potencial de Transformación:** La columna `survived` es `0` o `1`, pero sería más legible como "No" y "Sí". La columna `pclass` (clase del pasajero) es numérica pero en realidad representa una categoría (1ra, 2da, 3ra).

2. Manejo de Valores Faltantes (NaN)

Los valores faltantes (`NaN`) pueden arruinar nuestros cálculos. Pandas nos da un control total sobre cómo manejarlos.

a. Detectar Valores Nulos

Primero, debemos saber dónde y cuántos son.

- `.isnull()` o `.isna()`: Devuelven un DataFrame booleano (True si es nulo).
- Encadenamos `.sum()` para obtener un conteo de nulos por columna.

```
# Contar cuántos nulos hay en cada columna
print("Conteo de nulos por columna:")
print(titanic.isnull().sum())
```

Vemos que `age` tiene 177 valores faltantes y `deck` tiene 688 (la gran mayoría). `embark_town` y `embarked` también tienen un par.

b. Eliminar Filas o Columnas con Nulos (`.dropna()`)

Una opción es eliminar los datos faltantes.

- **Eliminar la columna `deck`:** Como le faltan demasiados datos, rellenarla no es fiable. La eliminaremos por completo (`axis=1`).
- **Eliminar filas con edad faltante:** Podríamos eliminar las 177 filas donde falta la edad, pero perderíamos mucha información valiosa.

```
# Hacemos una copia para trabajar de forma segura
titanic_limpio = titanic.copy()

# Eliminamos la columna 'deck' porque tiene demasiados nulos
titanic_limpio.dropna(axis=1, thresh=500, inplace=True) # thresh=500 elimina columnas con menos de 500 valores no nulos.

print("Columnas después de eliminar 'deck':\n", titanic_limpio.columns)
```

c. Rellenar Valores Nulos (`.fillna()`)

Una mejor estrategia para `age` y `embark_town` es rellenar los valores.

- **Para `age` (numérica):** Usaremos la **mediana** de edad. La mediana es más robusta que la media ante valores extremos (edades muy altas o muy bajas).
- **Para `embark_town` (categórica):** Usaremos la **moda** (el valor más frecuente), que es el puerto más común desde donde embarcaron los pasajeros.

```python
# Rellenar la edad faltante con la mediana
mediana_edad = titanic_limpio['age'].median()
titanic_limpio['age'].fillna(mediana_edad, inplace=True)

# Rellenar la ciudad de embarque con la moda
moda_puerto = titanic_limpio['embark_town'].mode()[0] # .mode() devuelve una Serie, tomamos el primer elemento [0]
titanic_limpio['embark_town'].fillna(moda_puerto, inplace=True)
titanic_limpio['embarked'].fillna(moda_puerto[0], inplace=True) # También la columna abreviada

# Verificamos que ya no quedan nulos importantes
print("\nConteo de nulos después de rellenar:")
print(titanic_limpio.isnull().sum())
```

3. Limpieza de Duplicados y Transformación de Datos

a. Manejo de Duplicados

Aunque este dataset tiene algunas filas duplicadas, en muchos casos querrás asegurarte de que no existan para no sesgar tu análisis.

- `.duplicated()` : Identifica filas duplicadas.
- `.drop_duplicates()` : Las elimina.

```
# Contamos el número de filas duplicadas antes de hacer nada
print(f"Número de filas duplicadas: {titanic_limpio.duplicated().sum()}")

# Eliminamos duplicados
titanic_limpio.drop_duplicates(inplace=True)
```

b. Aplicar Funciones a Columnas (`.apply()`)

Esta herramienta permite realizar transformaciones complejas. Creemos una nueva columna `age_group` para clasificar a los pasajeros.

```
def categorizar_edad(edad):
    if edad < 18:
        return 'Niño'
    elif 18 <= edad <= 65:
        return 'Adulto'
    else:
        return 'Adulto Mayor'

# Usamos .apply() en la columna 'age' para crear la nueva columna
titanic_limpio['age_group'] = titanic_limpio['age'].apply(categorizar_edad)

print(titanic_limpio[['age', 'age_group']].head())
```

c. Cambiar Tipos de Datos y Mapear Valores

Mejoremos la legibilidad de columnas como `survived` y `pclass`.

- **Mapear `survived`**: Usaremos `.map()` para convertir `0` y `1` a "No" y "Sí".
- **Convertir `pclass`**: Usaremos `.astype('category')` para tratarla como una categoría, no un número, lo cual es más correcto para los análisis estadísticos.

```
# Mapear 0/1 a No/Sí
titanic_limpio['survived'] = titanic_limpio['survived'].map({0: 'No', 1: 'Sí'})

# Convertir pclass a un tipo de dato categórico
titanic_limpio['pclass'] = titanic_limpio['pclass'].astype('category')

# Verifiquemos los cambios
print(titanic_limpio[['survived', 'pclass', 'age_group']].head())
print("\nNuevos tipos de datos:\n", titanic_limpio.dtypes)
```

4. Búsquedas y Filtros Avanzados

Ahora que los datos están limpios, podemos hacer preguntas más interesantes.

a. Filtrar con Múltiples Valores (`.isin()`)

Seleccionemos a todos los pasajeros de primera y segunda clase.

```
clases_altas = [1, 2]
pasajeros_clase_alta = titanic_limpio[titanic_limpio['pclass'].isin(clases_altas)]
print("Pasajeros en 1ra y 2da clase:")
print(pasajeros_clase_alta.head())
```

b. Filtrar por Cadenas de Texto (`.str`)

[cite_start]El accesor `.str` es muy potente para columnas de texto[cite: 106]. Encontremos a todas las familias cuyo apellido (extraído del nombre) contenga "William".

```
# Encontrar a pasajeros que tengan "William" en su nombre (útil para buscar familias)
pasajeros_william = titanic_limpio[titanic_limpio['who'].str.contains('man', case=False)]
print("\nPasajeros que son hombres:")
print(pasajeros_william.head())
```