Manejo de Datos Faltantes y Outliers - Día 5

Change Status

pending 40 min

Learning Objectives

1. Identificar y cuantificar datos faltantes en datasets usando técnicas sistemáticas
2. Aplicar estrategias apropiadas de imputación según el contexto y tipo de datos
3. Detectar outliers usando métodos estadísticos y visuales

Ejercicio: Pipeline completo de manejo de datos faltantes y outliers

Ejercicio práctico para aplicar los conceptos aprendidos.

Crear dataset con problemas realistas:

```
import pandas as pd
import numpy as np
from scipy import stats

# Crear dataset con missing values y outliers
np.random.seed(42)
n = 1000

datos = pd.DataFrame({
    'id': range(1, n+1),
    'edad': np.random.normal(35, 10, n).clip(18, 80),  # Normal con límites
    'salario': np.random.lognormal(10, 0.5, n),  # Distribución log-normal
    'horas_trabajo': np.random.normal(40, 5, n).clip(20, 60),
    'satisfaccion': np.random.randint(1, 6, n),
    'departamento': np.random.choice(['IT', 'Ventas', 'Marketing', 'HR'], n)
})

# Introducir missing values
```

```
mask_missing = np.random.random(n) < 0.1  # 10% missing
datos.loc[mask_missing, 'salario'] = np.nan

mask_missing_horas = np.random.random(n) < 0.05  # 5% missing
datos.loc[mask_missing_horas, 'horas_trabajo'] = np.nan

# Introducir outliers
outlier_indices = np.random.choice(n, 20, replace=False)
datos.loc[outlier_indices[:10], 'salario'] = datos.loc[outlier_indices[:10], 'salario'] * 10  # Salarios extremos altos
datos.loc[outlier_indices[10:], 'horas_trabajo'] = np.random.choice([80, 90, 100], 10)  # Horas imposibles

print(f"Dataset creado: {datos.shape}")
print(f"Valores faltantes por columna:\n{datos.isnull().sum()}")
```

Analizar datos faltantes:

```
# Análisis detallado de missing values
print("Porcentaje de datos faltantes:")
print((datos.isnull().sum() / len(datos) * 100).round(2))

# Patrón de missing values
import missingno as msno
# msno.matrix(datos)  # Visualización (requiere instalar missingno)

# Análisis por departamento
print("\nMissing values por departamento:")
print(datos.groupby('departamento').apply(lambda x: x.isnull().sum()))
```

Imputación de valores faltantes:

```
# Imputación por media para horas_trabajo
media_horas = datos['horas_trabajo'].mean()
datos['horas_trabajo'] = datos['horas_trabajo'].fillna(media_horas)

# Imputación por mediana para salario (más robusto a outliers)
mediana_salario = datos['salario'].median()
datos['salario'] = datos['salario'].fillna(mediana_salario)

# Verificar que no queden missing values
print(f"\nValores faltantes después de imputación: {datos.isnull().sum().sum()}")
```

Detección de outliers:

```
# Función para detectar outliers usando IQR
def detectar_outliers_iqr(data, columna):
    Q1 = data[columna].quantile(0.25)
    Q3 = data[columna].quantile(0.75)
    IQR = Q3 - Q1
    limite_inferior = Q1 - 1.5 * IQR
    limite_superior = Q3 + 1.5 * IQR
    return (data[columna] < limite_inferior) | (data[columna] > limite_superior)

# Detectar outliers en salario y horas
outliers_salario = detectar_outliers_iqr(datos, 'salario')
outliers_horas = detectar_outliers_iqr(datos, 'horas_trabajo')

print(f"\nOutliers detectados:")
print(f"Salario: {outliers_salario.sum()} ({outliers_salario.mean()*100:.1f}%)")
print(f"Horas trabajo: {outliers_horas.sum()} ({outliers_horas.mean()*100:.1f}%)")
```

Manejo de outliers:

```
# Para horas_trabajo: cap at reasonable maximum
max_horas_normales = 60
datos.loc[datos['horas_trabajo'] > max_horas_normales, 'horas_trabajo'] = max_horas_normales

# Para salario: transformar usando log (más robusto)
datos['salario_log'] = np.log1p(datos['salario'])

# Comparar estadísticas antes y después
print(f"\nEstadísticas de salario original:")
print(datos['salario'].describe().round(2))

print(f"\nEstadísticas de salario transformado (log):")
print(datos['salario_log'].describe().round(2))

# Verificar reducción de outliers
outliers_salario_log = detectar_outliers_iqr(datos, 'salario_log')
print(f"\nOutliers en salario log-transformado: {outliers_salario_log.sum()}")
```

Verificación: Compara las estadísticas y distribuciones antes y después del procesamiento para confirmar que los datos están más limpios y apropiados para análisis.

Requerimientos:

Python con Pandas y NumPy

Opcional: scipy para estadísticas avanzadas
Dataset con missing values y outliers para practicar
Conocimiento de operaciones básicas de Pandas