Transformación y Limpieza de Datos - Día 3

Change Status

pending · 40 min

Learning Objectives

1. Aplicar transformaciones condicionales complejas usando funciones vectorizadas de Pandas
2. Implementar validaciones de reglas de negocio e integridad de datos
3. Crear cálculos derivados y enriquecimiento de datasets

Ejercicio: Pipeline de transformación completo con validaciones

Ejercicio práctico para aplicar los conceptos aprendidos.

Crear dataset con datos que requieren transformación:

```
import pandas as pd
import numpy as np

# Crear datos con problemas realistas
np.random.seed(42)
n = 1000

df = pd.DataFrame({
    'id_cliente': range(1, n+1),
    'edad': np.random.normal(35, 15, n).clip(18, 80).astype(int),
    'ingresos': np.random.lognormal(10, 0.8, n),
    'gastos_mensuales': np.random.normal(2000, 500, n).clip(500, 10000),
    'categoria_cliente': np.random.choice(['A', 'B', 'C', 'D'], n),
    'fecha_registro': pd.date_range('2020-01-01', periods=n, freq='D')[:n],
    'email': [f'cliente{i}@ejemplo.com' for i in range(1, n+1)],
    'telefono': [f'({np.random.randint(100, 999)}){np.random.randint(100, 999)}-{np.random.randint(1000, 9999)}' for _ in range(n)]
})

# Introducir algunos errores intencionalmente
error_indices = np.random.choice(n, 50, replace=False)
df.loc[error_indices[:20], 'edad'] = np.random.choice([-5, 150, np.nan], 20)  # Edades inválidas
```

```
df.loc[error_indices[20:35], 'ingresos'] = -1000  # Ingresos negativos
df.loc[error_indices[35:], 'gastos_mensuales'] = df.loc[error_indices[35:], 'ingresos'] * 2  # Gastos > ingresos
```

Aplicar validaciones y correcciones:

```
# Validar y corregir edades
df['edad_valida'] = df['edad'].apply(lambda x: True if 18 <= x <= 80 else False)
df.loc[~df['edad_valida'], 'edad'] = np.nan  # Marcar inválidas como NaN

# Validar ingresos (no negativos)
df.loc[df['ingresos'] < 0, 'ingresos'] = np.nan

# Validar gastos vs ingresos
df['ratio_gasto_ingreso'] = df['gastos_mensuales'] / df['ingresos']
df.loc[df['ratio_gasto_ingreso'] > 1, 'gastos_mensuales'] = df.loc[df['ratio_gasto_ingreso'] > 1, 'ingresos'] * 0.8
```

Crear transformaciones y enriquecimientos:

```
# Categorizar por edad
df['grupo_edad'] = pd.cut(df['edad'],
                          bins=[18, 25, 35, 50, 80],
                          labels=['Joven', 'Adulto_Joven', 'Adulto', 'Senior'])

# Calcular capacidad de ahorro
df['capacidad_ahorro'] = df['ingresos'] - df['gastos_mensuales']
df['ratio_ahorro'] = df['capacidad_ahorro'] / df['ingresos']

# Clasificar capacidad financiera
df['clasificacion_financiera'] = np.where(df['ratio_ahorro'] > 0.3, 'Ahorra_Mucho',
                                          np.where(df['ratio_ahorro'] > 0.1, 'Ahorra_Poco',
                                                   np.where(df['ratio_ahorro'] > 0, 'Equilibra', 'Deficit')))

# Extraer información del teléfono
df['codigo_area'] = df['telefono'].str.extract(r'\((\d{3})\)')

# Calcular antigüedad
df['antiguedad_dias'] = (pd.Timestamp.now() - df['fecha_registro']).dt.days
df['antiguedad_meses'] = df['antiguedad_dias'] // 30
```

Crear métricas agregadas por categoría:

```
# Métricas por grupo de edad
metricas_edad = df.groupby('grupo_edad').agg({
    'ingresos': ['mean', 'median', 'std'],
```

```
    'capacidad_ahorro': 'mean',
    'ratio_ahorro': 'mean'
}).round(2)

print("Métricas por grupo de edad:")
print(metricas_edad)

# Resumen de validaciones
resumen_validacion = {
    'total_registros': len(df),
    'edades_invalidas': (~df['edad_valida']).sum(),
    'ingresos_negativos_corregidos': (df['ingresos'].isna()).sum(),
    'registros_procesados': len(df)
}

print("\nResumen de validación:")
for clave, valor in resumen_validacion.items():
    print(f"{clave}: {valor}")
```

Verificación: Examina las transformaciones aplicadas y confirma que los datos cumplen con las reglas de negocio definidas.

Requerimientos:

Python con Pandas y NumPy
Dataset de ejemplo o datos para transformar
Conocimiento de operaciones básicas de Pandas