

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

Theory

Practice

Quiz

Evidence

 Actividades y Aprendizajes

Aprende todo sobre funciones y módulos en Python con ejemplos prácticos.

Task 1: Transformaciones Condicionales con apply() y map() (10 minutos)

Las transformaciones condicionales representan la **lógica de negocio aplicada** a los datos, convirtiendo información cruda en insights accionables mediante reglas específicas del dominio.

apply(): Transformación por Filas o Columnas

apply() en Series: Aplica una función a cada elemento individualmente.

```
# Transformación simple
df['precio_con_iva'] = df['precio_sin_iva'].apply(lambda x: x * 1.21)

# Transformación condicional
df['categoria_precio'] = df['precio'].apply(lambda x: 'Alto' if x > 100 else 'Bajo')
```

apply() en DataFrame: Opera a lo largo de filas o columnas completas.





```
# Por filas (axis=1)
df['total_venta'] = df.apply(lambda row: row['cantidad'] * row['precio_unitario'], axis=1)

# Por columnas (axis=0)
df['promedio_categoria'] = df.groupby('categoria')['precio'].apply(lambda x: x.mean())
```

Ventajas de apply():

Flexibilidad máxima: Puede contener lógica compleja

Funciones personalizadas: Ideal para transformaciones específicas de negocio

Debugging fácil: Se puede agregar print statements para depuración

map(): Mapeo Directo de Valores

map() con diccionarios: Transformación directa valor-a-valor.

```
# Mapeo de categorías
categoria_map = {'A': 'Premium', 'B': 'Estándar', 'C': 'Básico'}
df['categoria_texto'] = df['categoria_codigo'].map(categoria_map)
```

map() con funciones: Aplicación de función a cada elemento.

```
# Transformación de texto
df['nombre_limpio'] = df['nombre'].map(lambda x: x.strip().title() if isinstance(x, str) else x)
```

np.where(): Condicionales Vectorizados

Estructura básica: np.where(condicion, valor_si_true, valor_si_false)

```
import numpy as np

# Clasificación simple
df['tipo_cliente'] = np.where(df['compras_totales'] > 1000, 'Premium', 'Regular')

# Condicional anidado
df['segmento'] = np.where(df['edad'] < 25, 'Joven',
                          np.where(df['edad'] < 45, 'Adulto', 'Senior'))
```

Ventajas de np.where():

Performance superior: Operaciones vectorizadas, no loops

Sintaxis clara: Similar a Excel IF()

Manejo de arrays: Trabaja bien con datos numéricos

Task 2: Validaciones y Reglas de Negocio (10 minutos)

Las validaciones aseguran que los datos transformados cumplan con **reglas de negocio críticas** que garantizan calidad y consistencia analítica.

Validaciones Básicas de Integridad

Rangos permitidos: Verificar que valores estén dentro de límites lógicos.

```
# Edad entre 0 y 120
mask_edad_invalida = (df['edad'] < 0) | (df['edad'] > 120)
df.loc[mask_edad_invalida, 'edad'] = np.nan # Marcar como faltante

# Precios positivos
df.loc[df['precio'] < 0, 'precio'] = 0
```

Consistencia relacional: Validar que claves foráneas existan.

```
# Verificar que todos Los clientes en pedidos existan en tabla clientes
clientes_validos = df_clientes['id_cliente'].unique()
pedidos_invalidos = ~df_pedidos['id_cliente'].isin(clientes_validos)
df_pedidos.loc[pedidos_invalidos, 'id_cliente'] = np.nan
```

Validaciones de Reglas de Negocio

Lógica condicional compleja: Reglas que involucran múltiples campos.

```
# Descuento solo para clientes premium con compras > 500
mask_descuento_invalido = (df['tipo_cliente'] == 'Premium') & (df['compra_total'] <= 500) & (df['descuento_aplicado'] > 0)
df.loc[mask_descuento_invalido, 'descuento_aplicado'] = 0
```

Consistencia temporal: Validar secuencia de eventos.

```
# Fecha de pedido no puede ser futura
fecha_actual = pd.Timestamp.now()
```



```
df.loc[df['fecha_pedido'] > fecha_actual, 'fecha_pedido'] = pd.NaT
```

Estrategias de Manejo de Errores de Validación

Logging de problemas: Registrar qué validaciones fallaron.

```
errores_validacion = []

if mask_edad_invalida.any():
    errores_validacion.append(f"Edades inválidas encontradas: {mask_edad_invalida.sum()} registros")

# Generar reporte de validación
if errores_validacion:
    with open('reporte_validacion.txt', 'w') as f:
        f.write('\n'.join(errores_validacion))
```

Task 3: Enriquecimiento y Cálculos Derivados (10 minutos)

El enriquecimiento transforma datos básicos en **información analítica rica** mediante cálculos y combinaciones inteligentes.

Cálculos Financieros y Métricas

Márgenes y porcentajes: Cálculos relativos importantes para análisis.

```
# Cálculo de márgenes
df['costo_venta'] = df['precio_compra'] * 1.1 # Costo + 10% gastos
df['margen_bruto'] = (df['precio_venta'] - df['costo_venta']) / df['precio_venta']
df['margen_porcentual'] = df['margen_bruto'] * 100
```

Acumulaciones temporales: Métricas que dependen de períodos anteriores.

```
# Ventas acumuladas por mes
df['ventas_acumuladas'] = df.groupby(['año', 'mes'])['ventas'].cumsum()

# Crecimiento mes a mes
df['crecimiento_mensual'] = df.groupby('cliente')['ventas'].pct_change()
```

Enriquecimiento con Datos Externos

Joins internos durante transformación: Combinar con tablas de referencia.



Enriquecer con información geográfica

```
df_enriquecido = pd.merge(df_ventas, df_ciudades,
                           left_on='codigo_postal', right_on='cp',
                           how='left')
```

Agregar categorías basadas en rangos

```
bins = [0, 100, 500, 1000, float('inf')]
labels = ['Bajo', 'Medio', 'Alto', 'Muy Alto']
df['segmento_compra'] = pd.cut(df['total_compras'], bins=bins, labels=labels)
```

Transformaciones de Texto Avanzadas

Extracción de patrones: Usar expresiones regulares para extraer información.

```
import re
```

Extraer código de área de teléfono

```
df['codigo_area'] = df['telefono'].str.extract(r'\((\d{3})\)')
```

Clasificar emails por dominio

```
def clasificar_email(email):
    if pd.isna(email):
        return 'Desconocido'
    dominio = email.split('@')[1].lower()
    if dominio in ['gmail.com', 'yahoo.com', 'hotmail.com']:
        return 'Personal'
    elif dominio.endswith('.edu'):
        return 'Educativo'
    else:
        return 'Empresarial'
```

```
df['tipo_email'] = df['email'].apply(clasificar_email)
```

