Filtrado, Agrupación y Merge de Datos - Día 4

Change Status

pending · 40 min

Learning Objectives

1. Dominar técnicas avanzadas de filtrado condicional usando query() y boolean indexing
2. Aplicar operaciones de agrupación con groupby() para análisis por categorías
3. Combinar datasets de diferentes fuentes usando operaciones de merge

Actividades y Aprendizajes

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

Task 1: Filtrado Avanzado con Query y Boolean Indexing (10 minutos)

El filtrado avanzado transforma consultas simples en **análisis sofisticados** que pueden expresar lógica compleja de manera legible y eficiente.

Query(): Lenguaje Declarativo para Filtrado

Ventajas sobre boolean indexing tradicional: `df.query('precio > 100 and categoria == "Electrónica"')` es más legible que `df[(df['precio'] > 100) & (df['categoria'] == 'Electrónica')]`.

Variables externas: `df.query('precio > @precio_minimo')` permite usar variables de Python en las consultas.

Sintaxis SQL-like: `df.query('edad between 25 and 35')` usa sintaxis similar a SQL.

Columnas calculadas: `df.query('precio * cantidad > 1000')` permite filtrar por expresiones calculadas.

Boolean Indexing: Control Granular

Máscaras booleanas: `mascara = df['categoria'] == 'Electrónica'` crea arrays de True/False.

Composición de condiciones: `df[(condicion1) & (condicion2)]` combina múltiples filtros.

Selección por posición: `df.loc[mascara, ['columna1', 'columna2']]` selecciona filas y columnas específicas.

Operaciones en lugar: `df.loc[mascara, 'columna'] = nuevo_valor` modifica valores condicionalmente.

Estrategias de Filtrado para Análisis Eficiente

Filtrado temprano: Aplicar filtros restrictivos primero para reducir el dataset antes de operaciones costosas.

Indexación inteligente: Usar `.loc[]` para evitar SettingWithCopyWarning.

Query optimization: Pandas optimiza automáticamente las consultas, pero el orden importa para performance.

Task 2: GroupBy: Análisis por Categorías (10 minutos)

GroupBy representa el **pilar fundamental** del análisis de datos, permitiendo transformar datos individuales en **insights colectivos** por grupos lógicos.

El Patrón Split-Apply-Combine

Split: Dividir los datos en grupos basados en una o más columnas.

Apply: Aplicar una función (agregación, transformación, filtrado) a cada grupo.

Combine: Reunir los resultados en una estructura coherente.

Operaciones de Agregación por Grupo

Funciones básicas: `df.groupby('categoria')['ventas'].sum()` - totales por categoría.

Múltiples agregaciones: `df.groupby('categoria')['ventas'].agg(['sum', 'mean', 'count'])`.

Diferentes funciones por columna: `df.groupby('categoria').agg({'ventas': 'sum', 'clientes': 'count'})`.

Funciones personalizadas: `df.groupby('categoria')['ventas'].agg(lambda x: x.quantile(0.9))`.

Transformaciones y Filtrado por Grupo

Transform: `df.groupby('categoria')['ventas'].transform('mean')` añade columna con promedio del grupo.

Filter: `df.groupby('categoria').filter(lambda x: x['ventas'].sum() > 1000)` filtra grupos enteros.

Apply: `df.groupby('categoria').apply(funcion_personalizada)` aplica lógica compleja por grupo.

Índices Jerárquicos y Multi-level Grouping

Múltiples niveles: `df.groupby(['año', 'mes'])['ventas'].sum()` - análisis temporal jerárquico.

Operaciones en niveles: `df.groupby(level=0).sum()` opera en el primer nivel del índice jerárquico.

Task 3: Merge: Combinación de Datasets (10 minutos)

Merge permite **unir datos de múltiples fuentes** creando análisis más ricos y completos, similar a las joins de SQL pero con mayor flexibilidad.

Tipos de Joins en Pandas

Inner join: `pd.merge(df1, df2, on='clave')` - solo filas con correspondencias en ambos DataFrames.

Left join: `pd.merge(df1, df2, on='clave', how='left')` - todas las filas de df1, datos de df2 cuando existen.

Right join: Similar a left pero preserva df2 completamente.

Outer join: `pd.merge(df1, df2, on='clave', how='outer')` - todas las filas de ambos DataFrames.

Estrategias de Merge

Merge por columna: `pd.merge(df1, df2, left_on='id1', right_on='id2')` cuando las claves tienen nombres diferentes.

Merge por índice: `pd.merge(df1, df2, left_index=True, right_index=True)` cuando las claves están en el índice.

Merge many-to-many: Maneja automáticamente relaciones muchos-a-muchos.

Concatenación vs Merge

Concat: `pd.concat([df1, df2])` apila DataFrames vertical u horizontalmente.

Merge: Combina basado en valores de columnas, creando productos cartesianos cuando necesario.

Join: `df1.join(df2)` - merge simplificado usando índices.