Clase 024 — Pandas: operaciones y alineación

Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.4 Operating on Data in Pandas. · Duración estimada: 60 min.

Clase 024 — Pandas: operaciones y alineación

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.4 Operating on Data in Pandas. · Duración estimada: 60 min.*

Objetivo

Que el alumno entienda cómo pandas alinea automáticamente por index en operaciones entre Series/DataFrames, cómo manejar NaN resultantes, y use apply/map para transformaciones custom (con consciencia de cuándo es lento).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Predecir el resultado de operar dos Series/DataFrames con indexes parcialmente distintos.
2. Usar fill_value en operaciones para no propagar NaN: s1.add(s2, fill_value=0).
3. Aplicar funciones con apply (lento, flexible), map (Series), applymap / df.map (elementwise).
4. Vectorizar transformaciones cuando se puede en vez de apply (10–100× más rápido).
5. Usar ufuncs NumPy sobre Series — pandas las soporta directamente y preserva el index.

Temas

#	Tema	Por qué importa
1	Alineación automática por index	Producto, suma, todo — pandas alinea, no a
2	fill_value para operaciones	Reemplaza el NaN antes de calcular.
3	apply axis=0 vs axis=1	Por columna vs por fila — costoso en filas
4	map para Series con dict	s.map({'A': 1, 'B': 2}).
5	df.map (era applymap) — elementwise	Cell-by-cell, lento.
6	Vectorización > apply	Si puedes hacerlo con ufunc, hazlo.

Definiciones y características

Alineación por index

: Operación matemática entre dos pandas (Series + Series, DF + Series, etc.) alinea por etiqueta de index. Labels que no estén en ambos → NaN.

fill_value en operaciones

: Parámetro que reemplaza NaN durante la operación: s1.add(s2, fill_value=0) trata índices ausentes como 0 en vez de propagar NaN.

apply

: Aplica función a cada fila (axis=1) o columna (axis=0) de un DataFrame, o cada elemento de una Series. Lento porque itera en Python. Usa solo si vectorización no aplica.

map (Series)

: Aplica función o dict a cada elemento. Útil para recodificación rápida: s.map({'A': 1, 'B': 2}). Más rápido que apply por su API más restringida.

df.map (antes applymap)

: Aplica función a cada celda del DataFrame (elementwise). Lento; usa solo cuando vectorización no aplica. applymap está deprecated en pandas 2.1+.

Ufunc-aware

: Pandas Series respeta ufuncs NumPy: np.log(s), np.sqrt(s) funcionan y preservan el index. Ventaja sobre convertir a array y perder labels.

Dataset / recursos

Sintético + Palmer Penguins. Sin descarga adicional.

Ejercicios

1. Suma con alineación. Dos Series con index parcialmente solapado. Súmalas (default) y con fill_value=0.

2. apply por fila. Define una función que reciba una fila de penguins y devuelva BMI = body_mass / bill_length². Aplica con axis=1.

3. Mismo cálculo vectorizado. Implementa BMI con operaciones vectorizadas. Mide ambos con %timeit.

4. map con dict. Mapea species a códigos: {'Adelie': 0, 'Chinstrap': 1, 'Gentoo': 2}.

5. ufunc NumPy preserva index. Aplica np.log a una columna; verifica que el index sigue intacto.

Homework verificable

Notebook con penguins: (a) BMI por fila con apply vs vectorizado (tabla %timeit); (b) species → código numérico con map; (c) demo de alineación con fill_value; (d) np.log sobre body_mass preservando index.

Criterio de aceptación: Vectorizado >50× más rápido que apply. Mapping y alineación correctos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
apply toma 10× más que vectorización equiv	Iteración Python por fila. Fix: piensa si
s1 + s2 produce NaN aunque los datos están	Index distinto (incluso por orden diferent
df.apply(func, axis=1) rompe con KeyError	Tu función accede row['col_x'] pero esa co
df.map(lambda x: ...) falla "object has no	Es método de DataFrame solo en pandas ≥2.1
np.log(df['x']) da error con NaN	Si NaN está presente, log da NaN. Si negat

Preguntas frecuentes

¿Cuándo apply y cuándo NO?

NO: si lo puedes hacer con df['a'] * df['b'] u operación pandas built-in (groupby, transform, etc.). SÍ: lógica

compleja por fila que no se descompone.

¿apply(axis=1) con tipos mixtos da problemas?

Sí — pandas convierte cada row a Series con dtype común. Si tienes int y str, queda object y los operadores <, > rompen. Mejor extrae columnas y opera directo.

¿Cómo paralelizo apply?

pandarallel, swifter, modin — drop-in replacements. Pero antes de paralelizar, asegúrate que tu apply no es vectorizable (el speedup es mayor).

¿s.map(dict) o s.replace(dict)?

map: mapea uno-a-uno; valores no encontrados en dict → NaN. replace: reemplaza solo los que aparecen; el resto queda igual. Elige según comportamiento deseado en valores faltantes.

¿Por qué pandas a veces es más lento que NumPy?

Overhead del index, manejo de NaN, dtype-aware. Para operaciones puramente numéricas sobre datos limpios y rectangulares, NumPy puede ser 2-5× más rápido. Pandas vale por la API, no por velocidad raw.

Referencias

- VanderPlas, cap. 3 § 3.4.
- pandas — apply, map

Siguiente clase

Clase 025 — Pandas: datos faltantes

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import numpy as np
import pandas as pd
import time
```

Archivos complementarios

- notebook.ipynb