AGRUPAMIENTO, UNIÓN Y PIVOTEO DE DATOS CON PANDAS: TÉCNICAS AVANZADAS PARA EL REACOMODO DE LA INFORMACIÓN

En la práctica profesional de la ciencia y la ingeniería de datos, el procesamiento y reorganización de la información es tan importante como su recolección. Rara vez los datos vienen listos para responder preguntas de negocio, alimentar modelos o construir dashboards útiles. Muy a menudo, los analistas y científicos de datos deben reacomodar los datos: unir tablas, agrupar registros, transformar la estructura (pivotear o despivotear), y preparar los datasets para su explotación final. Estas tareas forman parte del core de la manipulación de datos, y en Python, la librería pandas se ha convertido en el estándar de facto para abordar estos desafíos gracias a su potencia, flexibilidad y expresividad.

por Kibernum Capacitación S.A.

AGRUPAMIENTO Y PIVOTEO DE DATOS

El agrupamiento y el pivoteo permiten transformar datasets desde una forma plana ("longitudinal" o "tidy") a formas agregadas, resumidas o reestructuradas, facilitando el análisis exploratorio, la generación de reportes y el modelado avanzado.

Datos en formato plano

Estructura original con registros individuales

Agrupamiento

Segmentación y agregación por variables clave

Pivoteo

Reestructuración para análisis y visualización

AGRUPAMIENTO CON INDEXACIÓN JERÁRQUICA

En pandas, el agrupamiento con indexación jerárquica es una técnica fundamental para organizar y segmentar datos en múltiples niveles, permitiendo análisis segmentados y manipulaciones de gran detalle. Una indexación jerárquica implica que un DataFrame puede tener múltiples niveles de índice (MultiIndex), habilitando operaciones como agregaciones anidadas, filtrado eficiente y resúmenes por niveles jerárquicos.

Ejemplo:

Supongamos un DataFrame con ventas por región y año:

```
import pandas as pd

data = {
'region': ['Norte', 'Norte', 'Sur', 'Sur'],
'año': [2022, 2023, 2022, 2023],
'ventas': [150, 200, 100, 120]
}

df = pd.DataFrame(data)

# Crear un índice jerárquico
df_multi = df.set_index(['region', 'año'])
```

Esto permite acceder, agrupar y manipular datos por combinaciones específicas de región y año de forma directa y eficiente.

AGRUPAMIENTO DE DATOS CON LA FUNCIÓN GROUPBY

La función groupby es el núcleo del agrupamiento en pandas. Permite dividir el dataset en grupos basados en una o más columnas clave, aplicar funciones de agregación, transformación o filtrado sobre cada grupo, y luego recombinar los resultados.

Sintaxis general:

```
grouped = df.groupby('region')
print(grouped['ventas'].sum())
```

Para agrupamientos multinivel y agregaciones personalizadas:

```
agg = df.groupby(['region', 'año'])['ventas'].agg(['sum',
'mean', 'count'])
```

Operaciones comunes:

- Agregación: Sumar, promediar, contar valores por grupo.
- Filtrado: Seleccionar grupos que cumplen ciertas condiciones.
- Transformación: Aplicar funciones personalizadas sobre cada grupo.
- Aplicación de funciones complejas: Usar apply para operaciones más avanzadas o secuenciales.

Ejemplo avanzado:

```
df.groupby(['region', 'año']).agg(
    ventas_total=('ventas', 'sum'),
    ventas_promedio=('ventas', 'mean')
)
```

El agrupamiento permite descubrir patrones, comparar segmentos y preparar datos para modelos jerárquicos o visualizaciones agregadas.

PIVOTEO DE DATAFRAMES

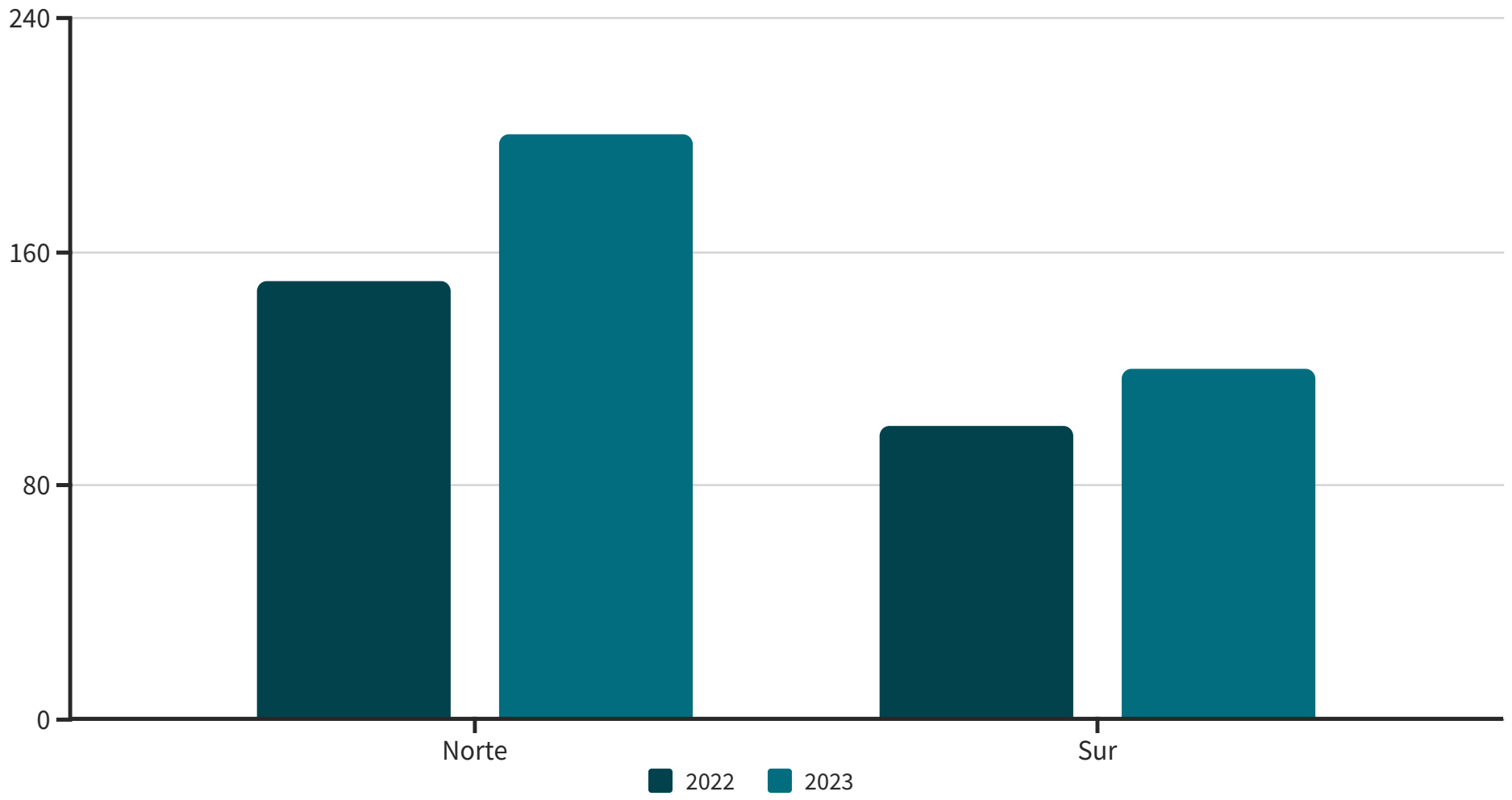
El pivoteo de DataFrames transforma datos de formato "largo" a "ancho", donde las variables de interés se convierten en columnas. Esto es fundamental para reportes, dashboards y análisis cruzados.

```
df_pivot = df.pivot(index='region', columns='año', values='ventas')
```

Cuando existen duplicados en la combinación de índices y columnas, se usa .pivot_table() y se especifica la función de agregación:

```
df_pivot = df.pivot_table(index='region', columns='año', values='ventas', aggfunc='sum')
```

El resultado es una tabla lista para visualización o análisis comparativo.

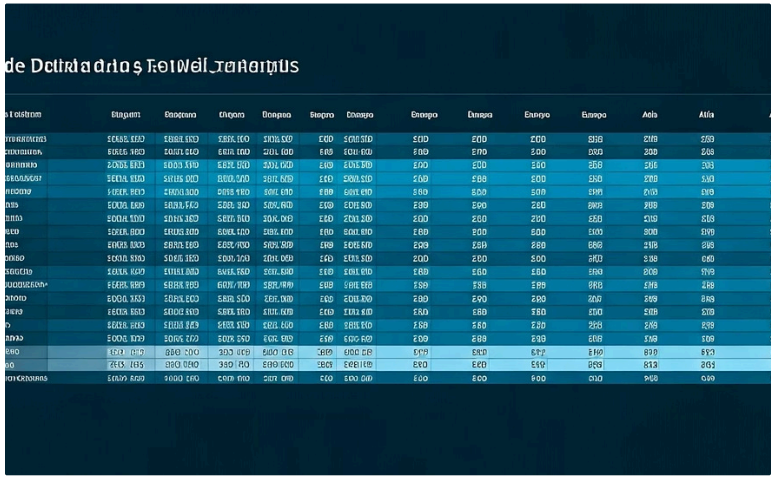


DESPIVOTEO DE DATAFRAMES (MÉTODO MELT)

El despivoteo (o unpivot) convierte una tabla ancha en formato largo, facilitando análisis, integraciones o preparación para machine learning.

```
df_long = pd.melt(df_pivot.reset_index(), id_vars='region', var_name='año', value_name='ventas')
```

Ahora cada fila es una observación única. Este formato es el esperado por la mayoría de las librerías de análisis y modelado.



Formato Ancho (Pivotado)

Cada variable tiene su propia columna, ideal para reportes y visualizaciones comparativas.

Formato Largo (Despivotado)

Cada observación tiene su propia fila, ideal para análisis estadístico y machine learning.

COMBINACIÓN Y MERGE DE DATOS

En la práctica real, los datos suelen estar dispersos en múltiples fuentes, tablas o archivos. Integrarlos correctamente es clave para un análisis robusto y exhaustivo. Las operaciones principales en pandas para combinar información son la concatenación y el merge (unión por claves).

Concatenación

Combina DataFrames apilándolos vertical u horizontalmente, sin necesidad de claves comunes.

Merge

Une DataFrames basándose en columnas clave comunes, similar a los joins en SQL.

CONCATENACIÓN DE DATAFRAMES

La concatenación de DataFrames permite apilar tablas vertical u horizontalmente, similar a un "append" o un "join" sin condición.

```
df1 = pd.DataFrame({'id': [1, 2], 'ventas': [100, 150]})
df2 = pd.DataFrame({'id': [3, 4], 'ventas': [200, 120]})

# Apilar filas
df_concat = pd.concat([df1, df2], ignore_index=True)

# Concatenar columnas
df_col_concat = pd.concat([df1, df2], axis=1)
```

Esta operación es esencial al integrar lotes periódicos, resultados de procesos en paralelo o datasets que comparten estructura.

Concatenación Vertical

id	ventas
1	100
2	150
3	200
4	120

Concatenación Horizontal

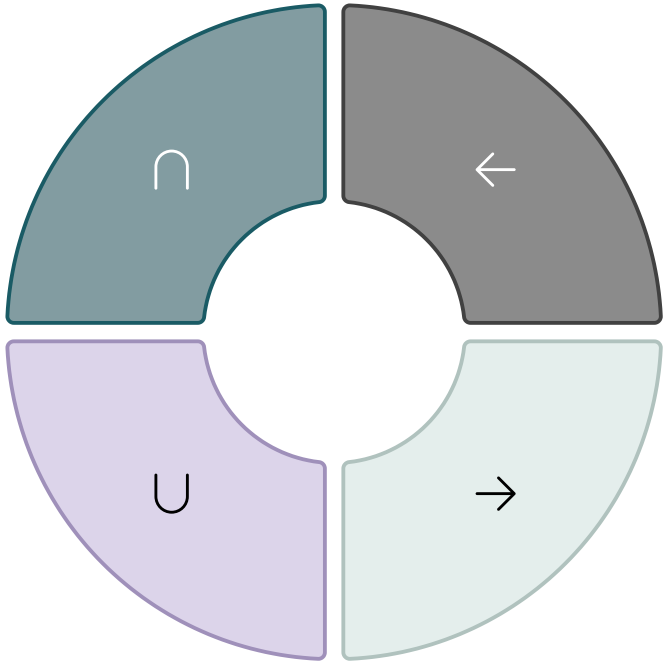
id_1	ventas_1	id_2	ventas_2
1	100	3	200
2	150	4	120

MERGE DE DATAFRAMES

El merge de DataFrames en pandas replica la lógica de los joins en bases de datos relacionales, uniendo tablas basadas en uno o más campos clave.

Inner Join

Solo filas con claves coincidentes en ambos DataFrames.



Left Join

Todas las filas del DataFrame izquierdo, añadiendo las coincidencias de la derecha.

Outer Join

Todas las filas de ambos DataFrames, uniendo donde hay coincidencias y completando con nulos donde faltan datos.

Right Join

Todas las filas del DataFrame derecho, añadiendo las coincidencias de la izquierda.

```
clientes = pd.DataFrame({'id': [1, 2, 3], 'nombre': ['Ana', 'Luis', 'Marta']})
ventas = pd.DataFrame({'id': [2, 3, 4], 'venta': [300, 400, 500]})

df_merge = pd.merge(clientes, ventas, on='id', how='inner')
```

El manejo de índices y claves es fundamental aquí (ver sección especial más abajo).

INTEGRACIÓN DE MÚLTIPLES FUENTES EN UN CASO DE NEGOCIO

En escenarios reales, los datos que se requieren para resolver un problema de negocio suelen provenir de diferentes sistemas: bases de datos transaccionales, archivos de logs, sistemas de recursos humanos, fuentes externas (como APIs o datos públicos).

Ejemplo: análisis de ventas en retail

Supongamos que un retailer necesita analizar ventas por producto, región y año. Los datos de ventas provienen de la plataforma de POS, los productos de un sistema maestro y las tiendas de una base de recursos humanos.

PIPELINE TÍPICO DE INTEGRACIÓN

Carga de datos

Desde archivos o bases de datos heterogéneas.

Normalización

De nombres, formatos y tipos de datos (por ejemplo, asegurando que los identificadores de productos sean del mismo tipo en todas las fuentes).

Merge

De ventas con productos, y luego con tiendas, usando identificadores únicos (id_producto, id_tienda).

Validación

Chequeo de filas sin correspondencia, detección de duplicados, verificación de unicidad de claves.

Agrupamiento

Y resumen de información por región y año.

Pivot

Para generar reportes anuales por región o tienda.

Despivoteo

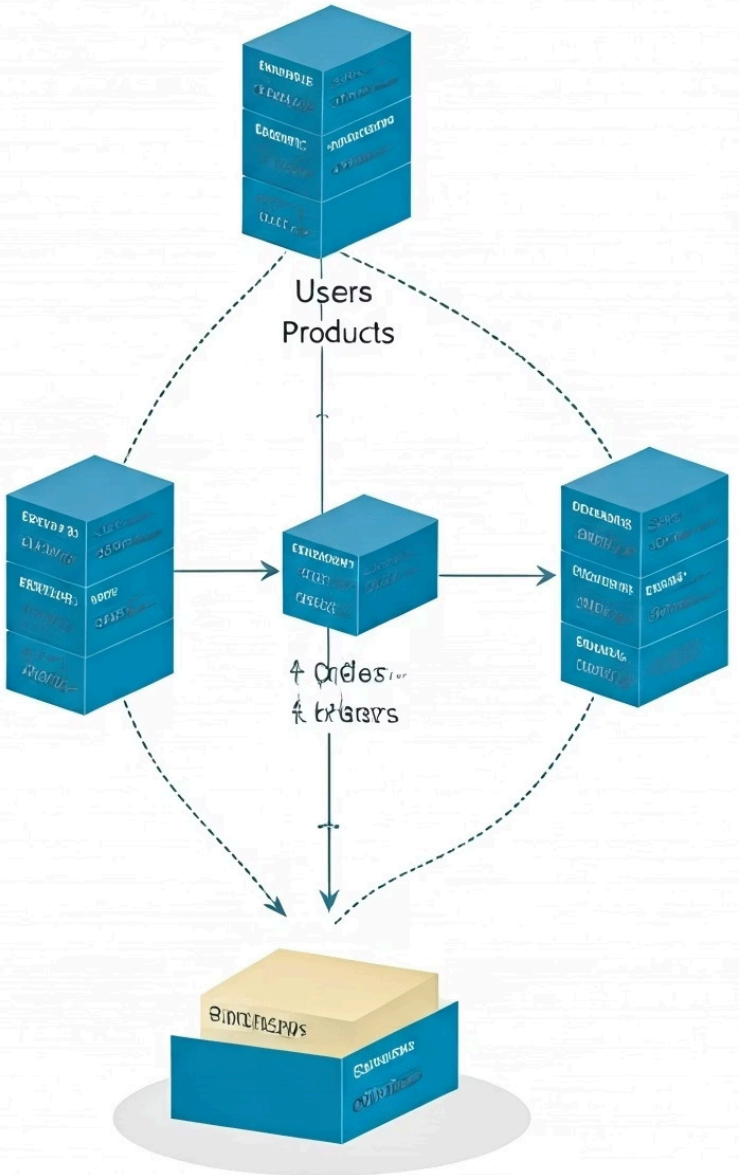
Para transformar datos y preparar análisis multivariado o modelos.

CÓDIGO ILUSTRATIVO DE INTEGRACIÓN

```
ventas_full = pd.merge(ventas, productos, on='id_producto', how='left')
ventas_full = pd.merge(ventas_full, tiendas, on='id_tienda', how='left')

# Agrupar por región y año
resumen = ventas_full.groupby(['region', 'año'])['monto'].agg(['sum', 'mean']).reset_index()

# Pivotear
pivot = resumen.pivot(index='region', columns='año', values='sum')

# Despivotear
pivot_long = pivot.reset_index().melt(id_vars='region', var_name='año', value_name='ventas_total')
```

Claves para la integración exitosa:

- Definir convenciones de nombres y tipos antes de integrar.
- Documentar la lógica de los merges y las reglas de negocio para resolver ambigüedades o duplicados.
- Registrar el porcentaje de registros que no encuentran correspondencia (por ejemplo, ventas con productos no existentes).



DESGLOSE SOBRE MANEJO DE ÍNDICES Y CLAVES EN MERGES COMPLEJOS

El correcto manejo de índices y claves es crucial para evitar errores silenciosos y asegurar integridad de los datos en operaciones de merge.

Identificación de claves únicas

- Antes de un merge, verifica que las columnas usadas como clave (on, left_on, right_on) sean únicas o, al menos, consistentes en uno de los DataFrames.
- Si las claves no son únicas, el resultado será un "producto cartesiano parcial", lo que puede inflar el número de filas inesperadamente.

```
# Validar unicidad de la clave
assert productos['id_producto'].is_unique, "id_producto no es único en productos"
```

USO DE ÍNDICES COMO CLAVES Y OPTIMIZACIÓN

Uso de índices como claves

Pandas permite mergear usando el índice (index) como clave:

```
df_merge = pd.merge(df1, df2, left_index=True,
right_index=True, how='inner')
```

Esto es útil cuando los datos ya están organizados jerárquicamente o cuando los índices representan claves naturales.

Optimización del merge

- Antes de grandes merges, convierte las claves a tipos eficientes (category para strings repetidos, int32 para identificadores numéricos).
- Usa sort=False en merge si el orden no es relevante y se busca mayor velocidad.

Gestión de claves conflictivas

Si ambas tablas tienen columnas de igual nombre que no son la clave, utiliza los parámetros suffixes para evitar sobrescritura y confusión:

```
df_merge = pd.merge(df1, df2, on='id', suffixes=('_ventas', '_productos'))
```

DETECCIÓN DE PÉRDIDAS Y DOCUMENTACIÓN

Detección de pérdidas o duplicados

- Siempre revisa el tamaño del DataFrame resultante. Si crece inesperadamente, probablemente hay un problema de claves no únicas.
- Después del merge, verifica que no haya filas duplicadas en las claves principales.

Documentación y auditoría

- Registra cómo y por qué se eligió cierto tipo de join (inner, left, etc.).
- Almacena los registros "huérfanos" (que no encontraron correspondencia) para su revisión posterior.

BUENAS PRÁCTICAS Y EFICIENCIA EN LA MANIPULACIÓN AVANZADA DE DATOS

Agrupamiento, unión y pivoteo pueden involucrar decenas de millones de registros y fuentes heterogéneas. Algunas recomendaciones esenciales son:

Optimización de memoria y procesamiento

- Usa tipos de datos compactos (categorical, int32, float32) y selecciona solo las columnas necesarias antes de combinar.
- Lee y une datos por lotes (chunksize) cuando trabajes con archivos grandes.

Trazabilidad y reproducibilidad

- Automatiza procesos en scripts versionados y centraliza parámetros en archivos de configuración.
- Define semillas fijas para operaciones aleatorias, y registra cada paso de transformación o combinación.

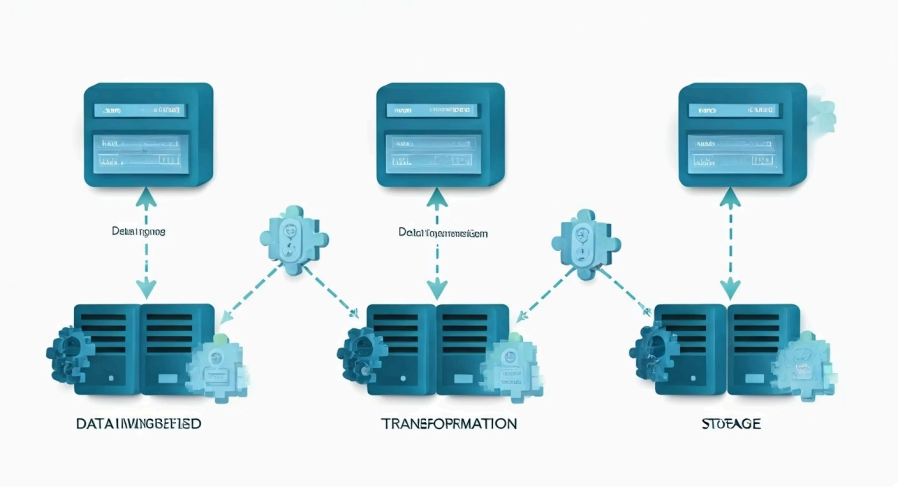
Validación y control de calidad

- Revisa el número de filas/columnas antes y después de cada merge, y valida unicidad de claves.
- Implementa validaciones automáticas y logging de errores, duplicados o pérdidas.

MODULARIDAD Y AUTOMATIZACIÓN

Modularidad y escalabilidad

- Divide operaciones complejas en funciones reutilizables.
- Evalúa frameworks distribuidos (Dask, PySpark) para grandes volúmenes.



Automatización y monitoreo

- Implementa pipelines programados (Airflow, Prefect, Luigi) para tareas recurrentes.
- Monitorea la calidad de los datos y configura alertas para duplicados, inconsistencias o registros perdidos.

RECURSOS Y REFERENCIAS

- Python for Data Analysis, Wes McKinney
- Documentación de pandas: **https://pandas.pydata.org/docs/**
- Practical Linear Algebra for Data Science, Mike X Cohen
- Cheat Sheets y tutoriales oficiales de pandas

Documentación Oficial | **Descargar Cheat Sheet**

REFLEXIÓN FINAL

El dominio de las técnicas de agrupamiento, unión y pivoteo en pandas es fundamental para transformar datos en activos analíticos listos para responder preguntas complejas y generar valor en cualquier organización. La capacidad de integrar múltiples fuentes, gestionar claves y manejar merges complejos con seguridad y eficiencia es un diferenciador clave en proyectos de datos avanzados. La correcta aplicación de estas herramientas, junto a buenas prácticas de trazabilidad y validación, asegura la calidad y confiabilidad de la información en los sistemas analíticos modernos.

RESUMEN DE TÉCNICAS CLAVE

1 Agrupamiento

Segmentación y agregación de datos por variables clave utilizando groupby, permitiendo análisis detallados por categorías.

2 Indexación Jerárquica

Organización de datos en múltiples niveles para facilitar operaciones complejas y análisis multidimensional.

3 Pivoteo

Transformación de datos de formato largo a ancho para facilitar reportes y visualizaciones comparativas.

4 Despivoteo

Conversión de datos de formato ancho a largo para análisis estadístico y preparación para machine learning.

5 Merge y Concatenación

Integración de múltiples fuentes de datos mediante uniones por claves o apilamiento de estructuras similares.

Estas técnicas, combinadas con buenas prácticas de validación, documentación y optimización, constituyen el núcleo de la manipulación avanzada de datos con pandas.