Operaciones avanzadas en pandas: merge, concat y groupby

Este documento explica tres funciones fundamentales de pandas para el análisis y manipulación de datos: **merge**, **concat** y **groupby**.

1. Merge

La función `merge` permite combinar dos DataFrames en base a una o más columnas clave. Es útil para unir información de diferentes fuentes.

Sintaxis básica:

```
## Una Columna Clave
pd.merge(df1, df2, on='columna_clave', how='inner')

## Dos o Más
pd.merge(df1, df2, on=['columna1', 'columna2'], how='inner')
```

- `on`: columna(s) por las que se realiza la unión.
- `how`: tipo de unión (`inner`, `outer`, `left`, `right`).

Ejemplo:

Supongamos que tienes dos DataFrames con información de estudiantes y sus notas:

```
import pandas as pd

**DataFrame estudiantes:**

| id | nombre |
|----|--------|
| 1  | Ana    |
| 2  | Luis   |
| 3  | Sofía  |

**DataFrame notas:**

| id | nota |
|----|------|
| 1  | 6.5  |
| 2  | 5.8  |
| 4  | 7.0  |
```

Inner

```
resultado = pd.merge(estudiantes, notas, on='id', how='inner')

**Resultado del merge:**

| id | nombre | nota |
|----|--------|------|
| 1  | Ana    | 6.5  |
| 2  | Luis   | 5.8  |
```

Outer

```
resultado_outer = pd.merge(estudiantes, notas, on='id', how='outer')

**Resultado del merge (outer):**

| id | nombre | nota |
|----|--------|------|
| 1  | Ana    | 6.5  |
| 2  | Luis   | 5.8  |
| 3  | Sofía  | NaN  |
| 4  | NaN    | 7.0  |
```

Left

```
resultado_left = pd.merge(estudiantes, notas, on='id', how='left')

**Resultado del merge (left):**

| id | nombre | nota |
|----|--------|------|
| 1  | Ana    | 6.5  |
| 2  | Luis   | 5.8  |
| 3  | Sofía  | NaN  |
```

Right

```
resultado_right = pd.merge(estudiantes, notas, on='id', how='right')

**Resultado del merge (right):**

| id | nombre | nota |
|----|--------|------|
| 1  | Ana    | 6.5  |
```

```
| 2  | Luis   | 5.8  |
| 4  | NaN    | 7.0  |
```

El resultado contendrá solo los estudiantes que tienen nota registrada.

Merge con varias columnas clave

También puedes combinar DataFrames usando más de una columna como clave. Por ejemplo:

```
**DataFrame df1:**

| id | curso | nota |
|----|-------|------|
| 1  | A     | 6.5  |
| 2  | B     | 5.8  |
| 3  | A     | 7.0  |

**DataFrame df2:**

| id | curso | asistencia |
|----|-------|------------|
| 1  | A     | 90         |
| 2  | B     | 85         |
| 3  | B     | 88         |

resultado_multi = pd.merge(df1, df2, on=['id', 'curso'], how='inner')

**Resultado del merge con varias columnas:**

| id | curso | nota | asistencia |
|----|-------|------|------------|
| 1  | A     | 6.5  | 90         |
| 2  | B     | 5.8  | 85         |
```

Merge usando `left_on` y `right_on`

A veces las columnas clave tienen nombres diferentes en cada DataFrame. En ese caso, puedes usar los argumentos `left_on` y `right_on`:

```
**DataFrame alumnos:**
| id_alumno | nombre |
|-----------|--------|
| 1         | Ana    |
| 2         | Luis   |
| 3         | Sofía  |

**DataFrame calificaciones:**
| id_estudiante | nota |
|---------------|------|
```

```
| 1            | 6.5  |
| 2            | 5.8  |
| 4            | 7.0  |

resultado = pd.merge(alumnos, calificaciones, left_on='id_alumno',
right_on='id_estudiante', how='inner')

**Resultado del merge usando left_on y right_on:**
| id_alumno | nombre | id_estudiante | nota |
|-----------|--------|---------------|------|
| 1         | Ana    | 1             | 6.5  |
| 2         | Luis   | 2             | 5.8  |
```

2. Concat

La función `concat` permite unir DataFrames o Series de manera vertical (por filas) u horizontal (por columnas). Es útil para apilar datos o agregar nuevas variables.

Sintaxis básica:

```
pd.concat([df1, df2], axis=0)  # Por filas
pd.concat([df1, df2], axis=1)  # Por columnas
# El default es por filas (`axis=0`)
```

- `axis=0`: concatena por filas.
- `axis=1`: concatena por columnas.

Si no se especifica el parámetro `axis`, la función `concat` une los DataFrames por filas. Es decir, agrega los registros uno debajo del otro.

Por ejemplo:

```
pd.concat([df1, df2])
# Equivalente a pd.concat([df1, df2], axis=0)
```

Ejemplo: Concatencación por filas

Supón que tienes dos DataFrames con información de diferentes años:

```
**DataFrame df_2023:**

| id | nota |
|----|------|
| 1  | 6.5  |
| 2  | 5.8  |

**DataFrame df_2024:**
```

```
| id | nota |
|----|------|
| 3  | 7.0  |
| 4  | 6.2  |

concat_df = pd.concat([df_2023, df_2024], axis=0)

**Resultado del concat:**

| id | nota |
|----|------|
| 1  | 6.5  |
| 2  | 5.8  |
| 3  | 7.0  |
| 4  | 6.2  |
```

Ejemplo: Concatenación por columnas

Supón que tienes dos DataFrames con información distinta para los mismos estudiantes:

```
**DataFrame notas:**
| id | nota |
|----|------|
| 1  | 6.5  |
| 2  | 5.8  |

**DataFrame asistencia:**
| id | asistencia |
|----|------------|
| 1  | 90         |
| 2  | 85         |

concat_columnas = pd.concat([notas, asistencia], axis=1)

**Resultado del concat por columnas:**
| id | nota | id | asistencia |
|----|------|----|------------|
| 1  | 6.5  | 1  | 90         |
| 2  | 5.8  | 2  | 85         |
```

El resultado es un solo DataFrame con todos los registros.

3. GroupBy

La función `groupby` permite agrupar los datos según una o más columnas y aplicar funciones de agregación (como suma, promedio, conteo, etc.).

Sintaxis básica:

```
df.groupby('columna').agg({'otra_columna': 'funcion'})
```

- `'columna'`: columna por la que se agrupa.
- `'funcion'`: función de agregación (`sum`, `mean`, `count`, `max`, `min`, `median`, etc.).

Ejemplo: Supón que tienes un DataFrame con información de estudiantes y sus notas:

```
**DataFrame datos:**

| curso | nota |
|-------|------|
| A     | 6.5  |
| A     | 5.8  |
| B     | 7.0  |
| B     | 6.2  |

promedios = datos.groupby('curso').agg({'nota': 'mean'})

**Resultado del groupby:**

| curso | nota promedio |
|-------|---------------|
| A     | 6.15          |
| B     | 6.60          |

**Ejemplo: Suma de notas por curso**

suma_notas = datos.groupby('curso').agg({'nota': 'sum'})

**Resultado del groupby (suma):**

| curso | suma de notas |
|-------|---------------|
| A     | 12.3          |
| B     | 13.2          |
```

Ejemplo: GroupBy con múltiples columnas

Puedes agrupar por más de una columna para obtener estadísticas por combinación de valores:

```
**DataFrame datos:**
| curso | año | nota |
|-------|-----|------|
| A     | 2023| 6.5  |
| A     | 2023| 5.8  |
| B     | 2023| 7.0  |
| B     | 2024| 6.2  |
```

```
agrupado = datos.groupby(['curso', 'año']).agg({'nota': 'mean'})

**Resultado:**
| curso | año  | nota promedio |
|-------|------|---------------|
| A     | 2023 | 6.15          |
| B     | 2023 | 7.00          |
| B     | 2024 | 6.20          |
```

Ejemplo: GroupBy con `apply`

El método `apply` permite aplicar funciones personalizadas a cada grupo (Conjunto de Filas):

```
def rango_notas(grupo):
    return grupo['nota'].max() - grupo['nota'].min()

**DataFrame datos:**
| curso | año | nota |
|-------|-----|------|
| A     | 2023| 6.5  |
| A     | 2023| 5.8  |
| B     | 2023| 7.0  |
| B     | 2024| 6.2  |

rango_por_curso = datos.groupby('curso').apply(rango_notas)

**Resultado:**

|curso     | ?????
|A     0.7 | ????
|B     0.8 | ??

# Es necesario el reset_index() porque el resultado de groupby + apply devuelve
una Serie con el índice igual al grupo (en este caso 'curso').
# Si queremos convertir esa Serie en un DataFrame y que 'curso' sea una columna
normal (no el índice), usamos reset_index().
# Así podemos renombrar las columnas y trabajar con el resultado como un DataFrame
estándar.
rango_por_curso_df = rango_por_curso.reset_index()
rango_por_curso_df.columns = ['curso', 'rango_nota']

**Resultado:**
| curso | rango de notas |
|-------|----------------|
| A     | 0.7            |
| B     | 0.8            |
```

Extra: Ejemplo Solo de `aply`

```
def redondear_columna(col):
    return round(col)

notas_redondeadas = datos['nota'].apply(redondear_columna)
notas_redondeadas

**Resultado**

|6|
|6|
|7|
|6|
```

Ahora creamos un nuevo datafrmame con esa columna

```
datos_redondeados = datos.copy()  # Hacemos una copia del DataFrame original
datos_redondeados['nota'] = notas_redondeadas  # Asignamos las notas redondeadas a
la columna 'nota'
datos_redondeados

**Resultado de datos_redondeados:**

| curso | año  | nota |
|-------|------|------|
| A     | 2023 | 6    |
| A     | 2023 | 6    |
| B     | 2023 | 7    |
| B     | 2024 | 6    |
```

Extra: `explode`

La función `explode` en pandas permite transformar una columna que contiene listas o arreglos en varias filas, expandiendo cada elemento de la lista en una fila separada. Es útil cuando tienes datos anidados y quieres "desempaquetarlos" para analizarlos más fácilmente.

Sintaxis básica:

```
df.explode('columna')
```

- `'columna'`: nombre de la columna que contiene listas o arreglos.

Ejemplo:

Supón que tienes un DataFrame donde cada estudiante puede tener varias materias inscritas:

```
import pandas as pd

datos = pd.DataFrame({
    'id': [1, 2, 3],
    'nombre': ['Ana', 'Luis', 'Sofía'],
    'materias': [['Matemáticas', 'Historia'], ['Física'], ['Química', 'Arte',
'Música']]
})

datos
```

Resultado original:

id	nombre	materias
1	Ana	['Matemáticas', 'Historia']
2	Luis	['Física']
3	Sofía	['Química', 'Arte', 'Música']

Ahora aplicamos `explode` sobre la columna `materias`:

```
datos_exploded = datos.explode('materias')
datos_exploded
```

Resultado después de explode:

id	nombre	materias
1	Ana	Matemáticas
1	Ana	Historia
2	Luis	Física
3	Sofía	Química
3	Sofía	Arte
3	Sofía	Música

Cada materia ahora aparece en una fila separada, facilitando el análisis por materia.