Exploración Completa y Explicada de la Librería Pandas

Victor David Niño Vivas

26/02/2026

1. Introducción

Pandas es una biblioteca de Python diseñada para manipular y analizar datos estructurados. Trabaja principalmente con dos estructuras: Series (una dimensión) y DataFrame (dos dimensiones).

Su ventaja principal es que realiza operaciones vectorizadas, es decir, procesa columnas completas sin necesidad de usar ciclos manuales.

2. Creación de Series y DataFrame

Antes de trabajar con datos, primero debemos crear las estructuras principales de Pandas. Para eso normalmente se importa la librería con:

```
import pandas as pd
```

2.1. Creación de una Series

Una Series se crea usando la función `pd.Series()`. Internamente, Pandas toma los datos que le damos, los convierte en un arreglo tipo NumPy y les asigna un índice.

Desde una lista

```
s = pd.Series([10, 20, 30, 40])
```

Aquí:

- Los valores son 10, 20, 30 y 40.
- Como no se indicó índice, Pandas crea uno automáticamente empezando desde 0.
- El tipo de dato se detecta automáticamente (en este caso entero).

Con índice personalizado

```
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
```

Aquí Pandas:

- Asocia cada valor con una etiqueta específica.
- Permite acceder a los datos usando esas etiquetas.

Desde un diccionario

```
s = pd.Series({"a": 10, "b": 20, "c": 30})
```

En este caso:

- Las claves del diccionario se convierten en el índice.
- Los valores del diccionario se convierten en los datos.

2.2. Creación de un DataFrame

Un DataFrame se crea con `pd.DataFrame()`. Internamente organiza los datos en forma de tabla, donde cada columna es una Series.

Desde un diccionario de listas

```
df = pd.DataFrame({
    "Nombre": ["Ana", "Luis", "Pedro"],
    "Edad": [23, 30, 28]
})
```

Aquí:

- Cada clave del diccionario se convierte en una columna.
- Cada lista debe tener la misma cantidad de elementos.
- Pandas asigna automáticamente un índice numérico.

Desde una lista de diccionarios

```
df = pd.DataFrame([
    {"Nombre": "Ana", "Edad": 23},
    {"Nombre": "Luis", "Edad": 30}
])
```

Aquí:

- Cada diccionario representa una fila.
- Las claves se convierten en columnas.
- Si falta una clave en alguna fila, Pandas coloca un valor nulo (NaN).

Desde un archivo CSV

```
df = pd.read_csv("archivo.csv")
```

Aquí Pandas:

- Lee el archivo.
- Detecta separadores.
- Interpreta los tipos de datos automáticamente.
- Crea el DataFrame listo para trabajar.

2.3. Series

Una Series es un arreglo unidimensional con índices.

Propiedades

- `index`: devuelve el objeto que contiene las etiquetas asociadas a cada valor.
- `values`: retorna los datos en formato array de NumPy.
- `dtype`: indica el tipo de dato almacenado (int64, float64, object, etc.).
- `shape`: devuelve una tupla indicando la dimensión.
- `size`: número total de elementos.
- `name`: nombre asignado a la serie.

Funciones importantes

- `head(n)`: devuelve los primeros n elementos. Si no se especifica n, devuelve 5 por defecto.
- `tail(n)`: devuelve los últimos n elementos.
- `describe()`: calcula estadísticas básicas automáticamente como media, desviación estándar, mínimo, máximo y percentiles. Solo aplica a datos numéricos.
- `mean()`: calcula el promedio sumando todos los valores y dividiendo entre la cantidad de datos válidos. Ignora valores NaN por defecto.
- `median()`: ordena los datos y devuelve el valor central.
- `mode()`: devuelve el valor que más se repite.
- `sum()`: suma todos los valores numéricos. Ignora valores NaN automáticamente.
- `min()`: devuelve el valor más pequeño comparando todos los elementos.
- `max()`: devuelve el valor más grande.
- `unique()`: devuelve un array con los valores distintos sin repetir.
- `value_counts()`: cuenta cuántas veces aparece cada valor y devuelve una Series ordenada de mayor a menor frecuencia.
- `sort_values()`: ordena los datos de menor a mayor (o mayor a menor si se usa ascending=False).
- `astype(tipo)`: convierte los datos al tipo especificado.
- `copy()`: crea una copia independiente en memoria.

3. DataFrame

Un DataFrame es una estructura bidimensional compuesta por múltiples Series alineadas por índice.

3.1. Propiedades

- `columns`: devuelve los nombres de columnas.
- `index`: devuelve los índices de filas.
- `dtypes`: muestra el tipo de dato de cada columna.
- `shape`: devuelve (filas, columnas).
- `size`: total de celdas.
- `values`: convierte el DataFrame en un array de NumPy.
- `T`: transpone filas por columnas.

3.2. Funciones básicas

- `info()`: muestra resumen general incluyendo cantidad de valores no nulos por columna y uso de memoria.
- `describe()`: genera estadísticas descriptivas por columna numérica.
- `sample(n)`: devuelve n filas aleatorias.
- `sort_values(by=columna)`: ordena el DataFrame según una o varias columnas.
- `sort_index()`: ordena según el índice.
- `drop(labels, axis)`: elimina filas o columnas.
- `rename(columns={})`: cambia nombres de columnas sin modificar los datos.
- `set_index(columna)`: convierte una columna en el nuevo índice.
- `reset_index()`: devuelve el índice a su forma original numérica.

4. Selección y Filtrado

- `loc[filas, columnas]`: selecciona usando etiquetas.
- `iloc[filas, columnas]`: selecciona usando posiciones numéricas.
- Filtros booleanos: permiten seleccionar filas que cumplan condiciones.
- `query()`: permite filtrar usando una expresión en formato texto.

5. Agrupación y Organización

- `groupby(columna)`: divide los datos en grupos según valores únicos y permite aplicar funciones como sum, mean o count a cada grupo.
- `agg()`: permite aplicar múltiples funciones estadísticas al mismo tiempo.
- `pivot()`: reorganiza la tabla cambiando filas por columnas según valores únicos.
- `pivot_table()`: similar a pivot pero permite aplicar funciones de agregación.
- `melt()`: transforma columnas en filas, útil para normalizar datos.
- `stack()` y `unstack()`: reorganizan niveles de índice en estructuras jerárquicas.

6. Unión de Datos

- `merge()`: combina dos DataFrames usando una columna común.
- `join()`: combina usando el índice por defecto.
- `concat()`: une DataFrames verticalmente o horizontalmente

7. Limpieza de Datos

- `isnull()`: devuelve True donde hay valores faltantes.
- `dropna()`: elimina filas o columnas con valores nulos.
- `fillna(valor)`: reemplaza valores nulos por uno específico.
- `replace()`: sustituye valores específicos por otros.
- `duplicated()`: detecta filas repetidas.
- `drop_duplicates()`: elimina filas duplicadas.

8. Operaciones con Texto

Estas funciones solo funcionan si la columna es tipo string.

- `str.lower()`: convierte todo el texto a minúsculas.
- `str.upper()`: convierte a mayúsculas.
- `str.contains("texto")`: devuelve True si el texto contiene la palabra indicada.
- `str.replace(a,b)`: reemplaza texto a por b.
- `str.len()`: devuelve la longitud de cada cadena.
- `str.strip()`: elimina espacios al inicio y final.

9. Trabajo con Fechas

- `to_datetime()`: convierte datos tipo texto en objetos fecha.
- `dt.year, dt.month, dt.day`: extraen partes específicas de una fecha.
- `resample()`: agrupa datos por intervalos de tiempo (día, mes, año).
- `rolling()`: aplica cálculos sobre ventanas móviles.
- `shift()`: desplaza los datos hacia adelante o atrás en el tiempo.

10. Funciones Estadísticas

- `std()`: calcula la desviación estándar.
- `var()`: calcula la varianza.
- `corr()`: mide la relación lineal entre columnas numéricas.
- `cov()`: calcula la covarianza.
- `count()`: cuenta valores no nulos.
- `nunique()`: cuenta valores distintos.

11. Funciones Avanzadas

- `apply()`: aplica una función personalizada a una columna o fila.
- `map()`: reemplaza o transforma valores individuales.
- `clip()`: limita los valores a un rango mínimo y máximo.
- `rank()`: asigna una posición según el orden de los valores.
- `cumsum()`, `cumprod()`, `cummax()`, `cummin()`: realizan operaciones acumulativas progresivas.

12. Conclusión

Pandas permite manipular datos de forma estructurada, eficiente y rápida. Sus funciones trabajan sobre columnas completas, optimizando el procesamiento. Es una herramienta fundamental para cualquier análisis de datos profesional.