Para mejorar la velocidad de procesamiento de un paquete de datos de 28 MB, hay varios enfoques que puedes aplicar dependiendo del tipo de datos y de las operaciones que necesitas realizar. Aquí tienes algunas estrategias generales:

1. Optimización de la Lectura de Datos

- Elegir el formato adecuado: Si estás leyendo desde un archivo, formatos como `parquet` o `feather` son más eficientes que `CSV` o `Excel`. Estos formatos binarios están optimizados para carga rápida y ocupan menos espacio.

- Cargar solo las columnas necesarias: En `pandas`, especifica las columnas que necesitas con el parámetro `usecols` para reducir la memoria utilizada.

- Trabajar en fragmentos (chunks): Si el archivo es muy grande y no necesitas todo el contenido en memoria, usa el parámetro `chunksize` de `pandas` para procesar fragmentos del archivo.

```python

Ejemplo de carga de datos en chunks

chunks = pd.read\_csv('data.csv', chunksize=5000) Procesar de 5000 filas a la vez

for chunk in chunks:

Procesar cada chunk

process(chunk)

```

2. Optimización de la Memoria del `DataFrame`

- Reducir tipos de datos: Cambia los tipos de datos de las columnas para ahorrar memoria. Por ejemplo, usa `float32` en lugar de `float64` o convierte las columnas categóricas a tipos `category`.

```python

Convertir tipo de columna para ahorrar memoria

df['column'] = df['column'].astype('float32')

```

3. Paralelizar el Procesamiento

- Usar procesamiento en paralelo: Si estás realizando operaciones pesadas de cálculo, puedes usar `Dask`, `modin` o `pandas` en combinación con `multiprocessing`. `Dask` o `modin` son bibliotecas diseñadas para trabajar con datos grandes en paralelo y son compatibles con la sintaxis de `pandas`.

```python

import dask.dataframe as dd

df = dd.read\_csv('data.csv')

Procesamiento con Dask

df = df.groupby('column').mean().compute()

```

4. Filtrar y Limpiar Datos al Comienzo

Filtra y elimina filas o columnas innecesarias al inicio para reducir el tamaño del `DataFrame` en memoria. Esto permite que las operaciones posteriores sean más rápidas.

5. Indexación y Muestreo

- Indexación: Establecer índices en las columnas utilizadas con frecuencia en búsquedas puede acelerar la recuperación de datos.

- Muestreo: Si solo necesitas una estimación, considera trabajar con una muestra de los datos en lugar del `DataFrame` completo.

6. Optimización de Funciones Aplicadas

- Aplicar funciones vectorizadas de `pandas` o `numpy`: Son más rápidas que las funciones `apply()`.

- Optimización de funciones personalizadas: Las funciones dentro de `apply` pueden ralentizar mucho el procesamiento si no están optimizadas.

Solamente aplicando la primera técnica (excepto los chunks), el procesamiento completo se demora entre 31 y 34 segundos para un archivo con 243229 tuplas que pesa 28 mb

Aplicando la optimización de memoria del dataframe (segunda técnica), no hay una gran diferencia, entre 31 y 34 segundos