Análisis de Caso

Estructuras de dato en Python y sentencias iterativas

Fabián Sepúlveda E.

Descripción del Caso

ComEres un desarrollador de DataSolvers encargado de liderar la optimización de las estructuras de datos para el sistema de análisis financiero. Tu misión es diseñar una serie de funciones que utilicen sentencias iterativas y estructuras de datos adecuadas para procesar y analizar los datos financieros, garantizando eficiencia y precisión en los resultados. Tendrás que resolver los siguientes desafíos basándote en los conceptos estudiados.

Desarrollo

1. Análisis de la Estructura de Datos

   Ventajas:

   - **Listas**: Permiten almacenar transacciones de forma ordenada, son mutables y fáciles de recorrer con bucles. Su uso es adecuado cuando el orden importa y hay posibles valores duplicados.
   - **Diccionarios**: Útiles para clasificar ingresos por categoría; permiten acceso rápido a través de claves, agrupación eficiente y manipulación sencilla.

   Limitaciones/Mejoras:

   - Las listas pueden ser ineficientes si se busca verificar duplicados o hacer búsquedas frecuentes.
   - Los diccionarios podrían beneficiarse del uso de defaultdict para evitar condiciones if-else.
   - Para grandes volúmenes, podría considerarse el uso de collections.Counter, pandas, o incluso estructuras más avanzadas como namedtuple.

2. Optimización de Sentencias Iterativas

   Para optimizar las sentencias iterativas usé comprensiones de listas, que reduce línea de código y mejora la legibilidad. El uso de funciones que ya poseen Python, por ejemplo sum(), crea bucles más eficientes.

   - Función 1 `calcular_total_ingresos`

     ```
     def calcular_total_ingresos(self, transacciones):
         return sum(transacciones)
     ```

   - Función 2 `filtrar_ingresos_altos`

```python
def filtrar_ingresos_altos(self, transacciones, umbral):
    return [ingreso for ingreso in transacciones if ingreso > umbral]
```

3. Implementación de Pruebas para las funciones.

A continuación se presenta la función que realiza las pruebas de las funciones calcular_total_ingresos, filtrar_ingresos_altos,agrupar_por_categoria

```python
def pruebas():
    af = AnalizadorFinanciero()
    transacciones = [100, 200, 50, 400]
    categorias = ['A', 'B', 'A', 'C']
    umbral = 150

    assert af.calcular_total_ingresos(transacciones) == 750
    assert af.filtrar_ingresos_altos(transacciones, umbral) == [200, 400]
    assert af.agrupar_por_categoria(transacciones, categorias) == {'A': [100, 50], 'B': [200], 'C': [400]}
    print("Todas las pruebas pasaron.")
```

4. Aplicación de Estructuras de Datos Avanzadas

Para obtener categorías únicas rápidamente se ocupó set, ya que elimina duplicados de forma automática, tiene un acceso más rápido para la verificación de existencia (in).

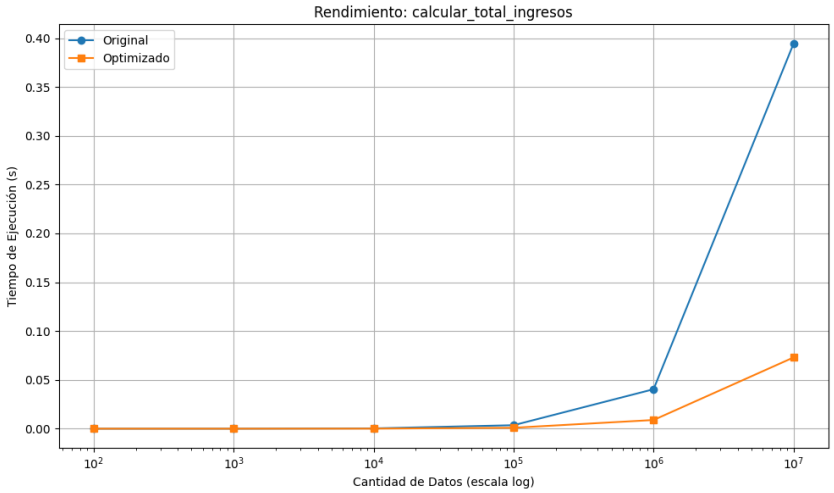
5. Refactorización del Código

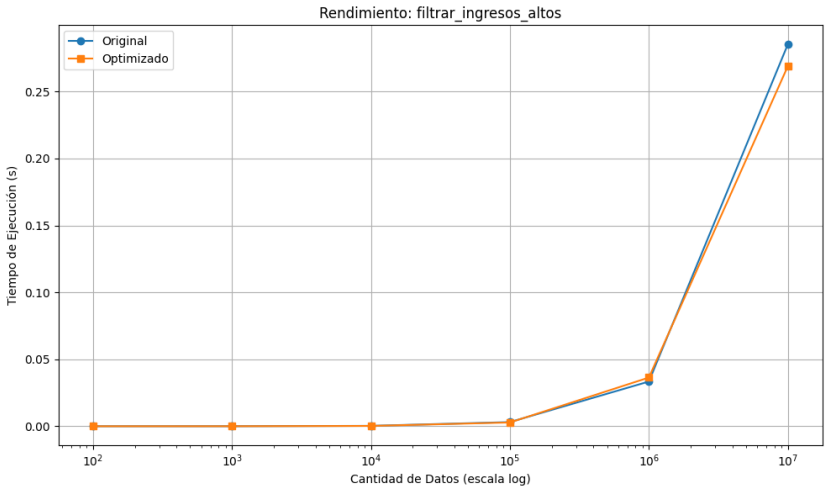
A continuación se presenta la clase AnalizadorFinanciero refactorizada:

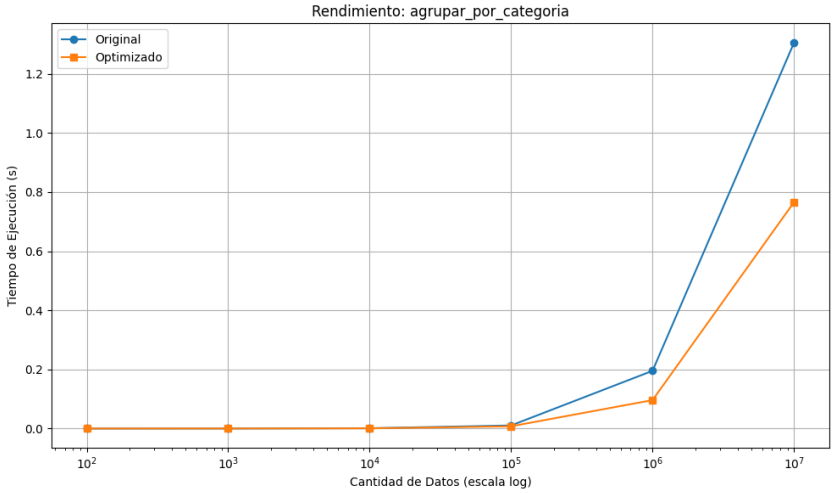
```python
class AnalizadorFinanciero:

    def calcular_total_ingresos(self, transacciones):
        return sum(transacciones)

    def filtrar_ingresos_altos(self, transacciones, umbral):
        return [ingreso for ingreso in transacciones if ingreso > umbral]

    def agrupar_por_categoria(self, transacciones, categorias):
        agrupado = defaultdict(list)
        for cat, ingreso in zip(categorias, transacciones):
            agrupado[cat].append(ingreso)
        return dict(agrupado)

    def obtener_categorias_unicas(self, categorias):
        return set(categorias)
```

6. Análisis de Rendimiento

Se analizaron las funciones calcular_total_ingresos, filtrar_ingresos_altos y agrupar_categoria en función del tiempo que se demoraron en ejecutar conjuntos de datos de los siguientes tamaños: 10,100,1.000,10.000,100.000,1.000.000 y 10.000.000 de datos. Se puede apreciar que el método optimizado mejora la performance cuando se trata de un volumen grande de datos






Rendimiento: agrupar_por_categoria
Original
Optimizado
Tiempo de Ejecución (s)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
10^2
10^3
10^4
10^5
10^6
10^7
Cantidad de Datos (escala log)