Extracción y Almacenamiento Automatizado de Datos Financieros desde Yahoo Finance utilizando Python

Resumen

El presente trabajo describe la implementación de una solución automatizada para la extracción, transformación y almacenamiento de datos históricos financieros de la empresa MercadoLibre S.A. (MELI) desde la plataforma Yahoo Finance. Utilizando bibliotecas de Python como `requests`, `BeautifulSoup`, `pandas` y `sqlite3`, se construye una herramienta que recolecta los datos de la web, los convierte a un formato estructurado, y los almacena tanto en formato CSV como en una base de datos SQLite. Además, se incorpora un sistema de registro de eventos (logging) y se automatiza su ejecución mediante GitHub Actions. Esta solución demuestra cómo automatizar flujos de datos con técnicas reproducibles y adaptables a otros contextos similares.

Introducción

En la era del Big Data, la disponibilidad y accesibilidad de datos financieros es fundamental para el análisis cuantitativo, el modelado económico y la toma de decisiones en tiempo real. Yahoo Finance es una fuente ampliamente utilizada para acceder a información bursátil gratuita, sin embargo, no ofrece una API oficial para todos sus servicios, lo cual representa un reto técnico. Este proyecto presenta una solución automatizada desarrollada en Python para extraer datos históricos del precio de las acciones de MELI, almacenarlos localmente, y mantener registros del proceso con fines de trazabilidad. La solución también se integra con GitHub Actions para asegurar su ejecución automática ante cambios en el repositorio.

Metodología

La solución propuesta se desarrolló utilizando un enfoque modular, dividiendo el sistema en los siguientes componentes:

1. Recolección de datos (Collector): Se implementó una clase `Collector` en Python que hace una solicitud HTTP a la URL correspondiente de Yahoo Finance, utilizando `requests` con cabeceras personalizadas para simular la navegación desde un navegador. El contenido HTML se procesa con `BeautifulSoup` para extraer la tabla de datos históricos mediante el selector `data-testid="history-table"`. La tabla es convertida a un `DataFrame` de `pandas`, y se renombran las columnas para estandarizar los

nombres a español.
2. Persistencia de datos: El `DataFrame` generado se guarda como un archivo CSV y también se inserta en una base de datos SQLite con el nombre `meli_history`, permitiendo su reutilización en análisis posteriores.
3. Registro de eventos (Logger): Para asegurar la trazabilidad del proceso, se construyó una clase `Logger` personalizada que registra eventos en archivos `.log` con detalles como la clase y la función origen del mensaje.
4. Automatización con GitHub Actions: Se definió un flujo de trabajo en YAML (`workflow.yml`) que se activa automáticamente en cada `push` a la rama `main`. Este workflow instala las dependencias necesarias, ejecuta el script `main.py` y realiza un `commit` con los archivos generados o modificados.

Conclusiones

La solución desarrollada demuestra que es posible automatizar con éxito la recolección y almacenamiento de datos financieros sin depender de APIs oficiales. Al combinar herramientas de scraping, análisis de datos y automatización CI/CD, se obtiene un flujo de trabajo robusto y adaptable. Este tipo de sistema puede ser reutilizado para diferentes fuentes o activos financieros con mínimas modificaciones. Asimismo, la inclusión de un sistema de logging facilita la trazabilidad y el mantenimiento. Finalmente, GitHub Actions permite mantener la solución actualizada de forma automática, fortaleciendo su utilidad en contextos académicos o profesionales orientados al análisis financiero.