Documentación de la Arquitectura y Modelo de Datos

Elaborado por:

Jeisson Estevens Araque Ramírez.

Docente:

Andrés Felipe Callejas Jaramillo

Institución Universitaria Digital de Antioquia

Ingeniería de Software y Datos

Infraestructura y arquitectura para Big Data

2025

Contenido

Introducción

Este documento presenta la arquitectura y el modelo de datos implementados en el proyecto integrador de Big Data. El proyecto se enfoca en la creación de un pipeline de datos que abarca desde la ingesta de información de mercados de criptomonedas hasta su procesamiento y enriquecimiento para análisis posteriores.

La solución propuesta simula un entorno de nube para Big Data, implementando prácticas y patrones comunes en la industria, adaptados a un contexto de desarrollo que permite la experimentación y aprendizaje de conceptos fundamentales en el procesamiento de datos a gran escala.

Este documento detalla la estructura técnica del sistema, los flujos de información, las decisiones de diseño y las tecnologías seleccionadas, proporcionando una visión completa de la arquitectura implementada.

Descripción General de la Arquitectura

Visión Global

El proyecto implementa una arquitectura de procesamiento de datos que simula un entorno de Big Data en la nube, siguiendo un flujo de trabajo secuencial que abarca tres fases principales:

1. **Ingesta de datos**: Extracción de información desde la API de MercadoBitcoin y almacenamiento en múltiples formatos (JSON, SQLite, CSV).
2. **Preprocesamiento**: Limpieza de datos mediante la identificación y corrección de valores nulos, inconsistencias y outliers.
3. **Enriquecimiento**: Integración con fuentes externas de datos para añadir contexto y valor analítico a la información procesada.

Esta arquitectura simula los componentes clave de un entorno de Big Data en la nube, implementando patrones comunes de ETL (Extract, Transform, Load) adaptados a un contexto local.

Componentes Principales

Base de Datos Analítica (SQLite)

La solución utiliza SQLite como motor de base de datos, lo que permite:

- Almacenamiento estructurado y relacional de los datos procesados
- Persistencia de información entre las distintas fases del procesamiento
- Capacidad de realizar consultas SQL para análisis posteriores
- Portabilidad y bajo consumo de recursos

La base de datos incluye tablas para almacenar tanto los datos crudos de la API como los datos procesados y enriquecidos.

Scripts de Procesamiento

El sistema implementa tres scripts principales correspondientes a cada fase del pipeline:

1. **Script de Ingesta** (ingestion.py):

   Función: Extracción y almacenamiento inicial de datos

   Características clave:

   - Implementa peticiones HTTP a la API de MercadoBitcoin
   - Parsea y estructura las respuestas JSON
   - Almacena datos en múltiples formatos (JSON, SQLite, CSV)
   - Incluye mecanismos de auditoría para validar integridad

   Métodos principales: obtener_datos_api(), guardar_datos(), guardar_db(), guardar_csv(), validar_auditoria()

2. **Script de Limpieza** (script de preprocesamiento):

   Función: Transformación y normalización de datos

   Características clave:

   - Carga datos desde CSV generado en la fase de ingesta
   - Identifica y corrige valores nulos o inconsistentes
   - Maneja valores extremos mediante filtrado
   - Normaliza tipos de datos, especialmente fechas

   Métodos principales: generar_datos_de_prueba(), limpiar_datos(), cargar_datos_csv(), guardar_csv()

3. **Script de Enriquecimiento** (último script del flujo):

   Función: Integración con fuentes externas y validación final

   Características clave:

   - Combina datos limpios con información adicional de mercado
   - Normaliza formatos para garantizar integrabilidad
   - Implementa auditoría del proceso de enriquecimiento
   - Genera el dataset final para análisis

   Métodos principales: enrich_data(), parse_date(), auditoria_enrich()

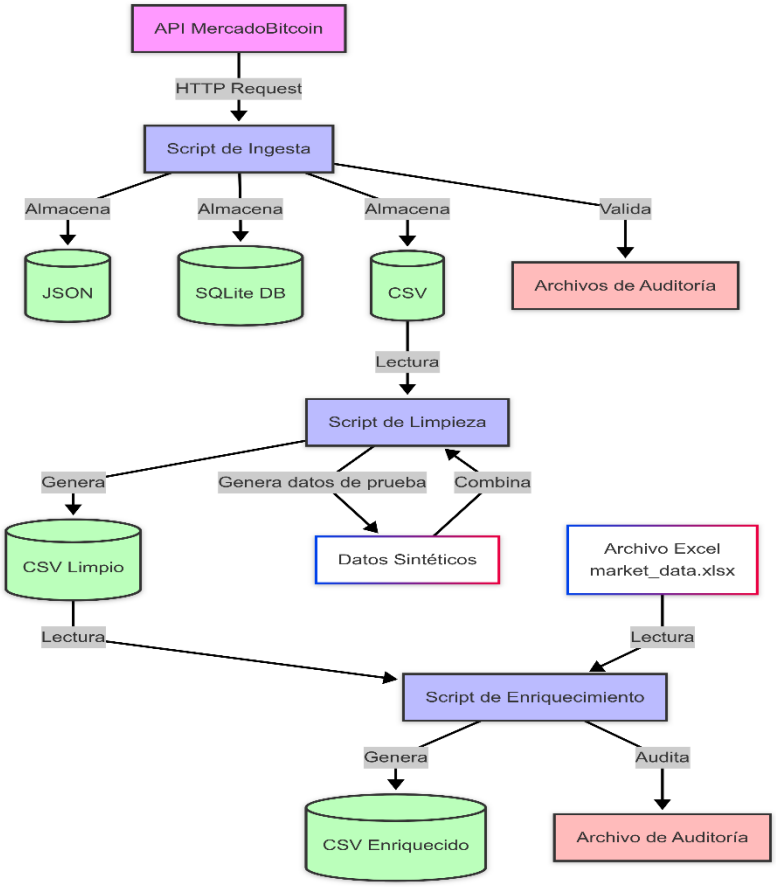
Mecanismo de automatización

La ejecución del pipeline se ha automatizado utilizando GitHub Actions como solución de orquestación. El flujo de trabajo se dispara automáticamente al detectarse un commit en la rama principal del repositorio.

Diagramas de Arquitectura

Diagrama de Flujo de Datos

El siguiente diagrama ilustra el flujo completo de datos a través del sistema, desde la fuente original hasta los datos finales enriquecidos:

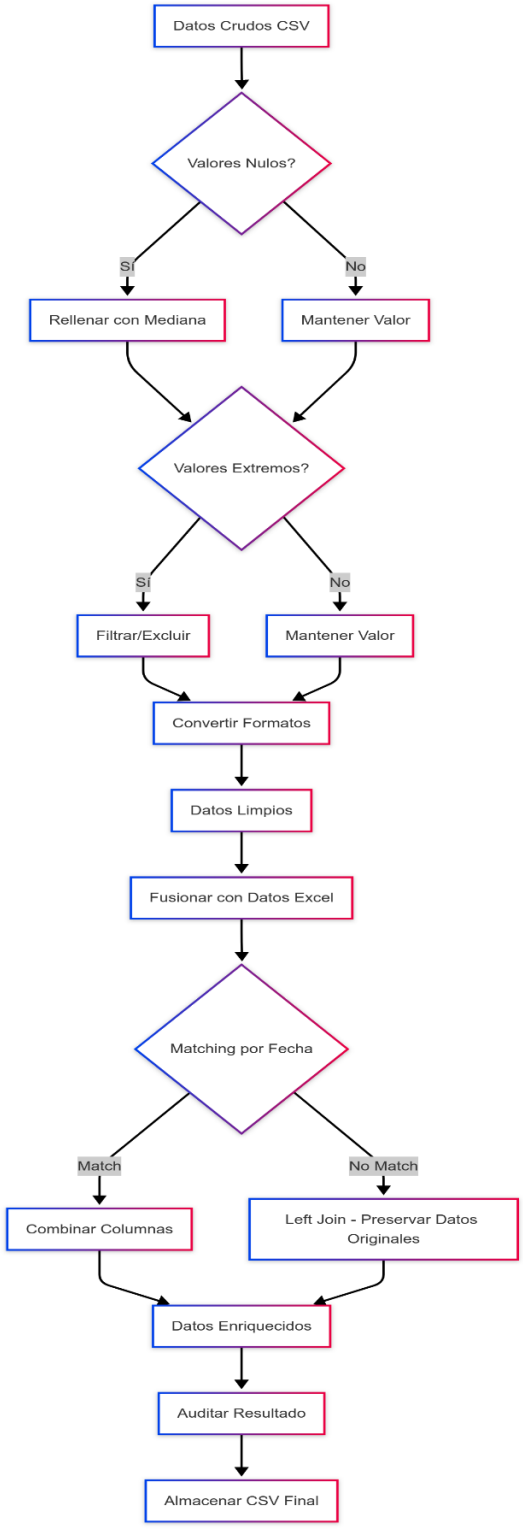


El diagrama anterior ilustra de forma visual la secuencia de operaciones y el flujo de información dentro del sistema. En él se destacan las fuentes de datos externas, incluyendo tanto la integración con APIs como la lectura de archivos Excel; los componentes de procesamiento, representados por los scripts encargados de la transformación de datos; los artefactos generados, que son los resultados del procesamiento en formatos como archivos JSON y CSV, así como las actualizaciones en la base de datos (DB); y finalmente, los mecanismos de validación y auditoría, que son los procesos implementados para asegurar la calidad y el seguimiento de las operaciones.

Diagrama de Procesos

El siguiente diagrama detalla los procesos específicos de preprocesamiento y enriquecimiento, mostrando las transformaciones aplicadas a los datos:

Este diagrama ilustra las decisiones lógicas que se toman durante el proceso de limpieza de datos, el tratamiento específico aplicado a distintos tipos de inconsistencias encontradas, el proceso de integración de datos provenientes de fuentes externas, y las validaciones finales que se realizan al concluir el proceso.


Datos Crudos CSV
Valores Nulos?
Si
No
Rellenar con Mediana
Mantener Valor
Valores Extremos?
Si
No
Filtrar/Excluir
Mantener Valor
Convertir Formatos
Datos Limpios
Fusionar con Datos Excel
Matching por Fecha
Match
No Match
Combinar Columnas
Left Join - Preservar Datos Originales
Datos Enriquecidos
Auditar Resultado
Almacenar CSV Final

Modelo de Datos

Definición del Esquema

El modelo de datos evoluciona a través de las distintas fases del procesamiento, reflejando las transformaciones aplicadas a la información:

1. Tabla de Datos Crudos (SQLite)

Tabla: mercado_bitcoin

Campo	Tipo	Descripción	Restricciones
Id	INTEGER	Identificador único de registro	Clave primaria, autoincremental
High	REAL	Precio más alto del periodo	Puede ser NULL
Low	REAL	Precio más bajo del periodo	Puede ser NULL
Vol	REAL	Volumen de transacciones	Puede ser NULL
Last	REAL	Último precio negociado	Puede ser NULL
Buy	REAL	Precio de compra actual	Puede ser NULL
Sell	REAL	Precio de venta actual	Puede ser NULL
date	INTEGER	Fecha en formato timestamp	Puede ser NULL

Esta tabla almacena los datos crudos tal como se reciben de la API, sin transformaciones significativas más allá de la estructuración en el esquema relacional.

Datos Limpios (CSV)

Archivo: cleaned_data.csv

Campo	**Tipo**	**Descripción**	**Transformaciones aplicadas**
High	Float	Precio más alto	Valores NULL reemplazados con mediana
Low	Float	Precio más bajo	Valores NULL reemplazados con mediana
Vol	Float	Volumen	Outliers filtrados (<1000), NULL reemplazados
Last	Float	Ultimo precio	Valores negativos filtrados, NULL reemplazados
Buy	Float	Precio de compra	NULL reemplazados con mediana
Sell	Float	Precio de venta	NULL reemplazados con mediana
Open	Float	Precio de apertura	NULL reemplazados con mediana
Date	Datetime	Fecha convertida	Convertida de timestamp a formato datetime
Pair	String	Par de trading	Valores NULL filtrados

Este modelo intermedio refleja la aplicación de técnicas de limpieza y normalización, preparando los datos para el análisis y el enriquecimiento.

Datos Enriquecidos (CSV)

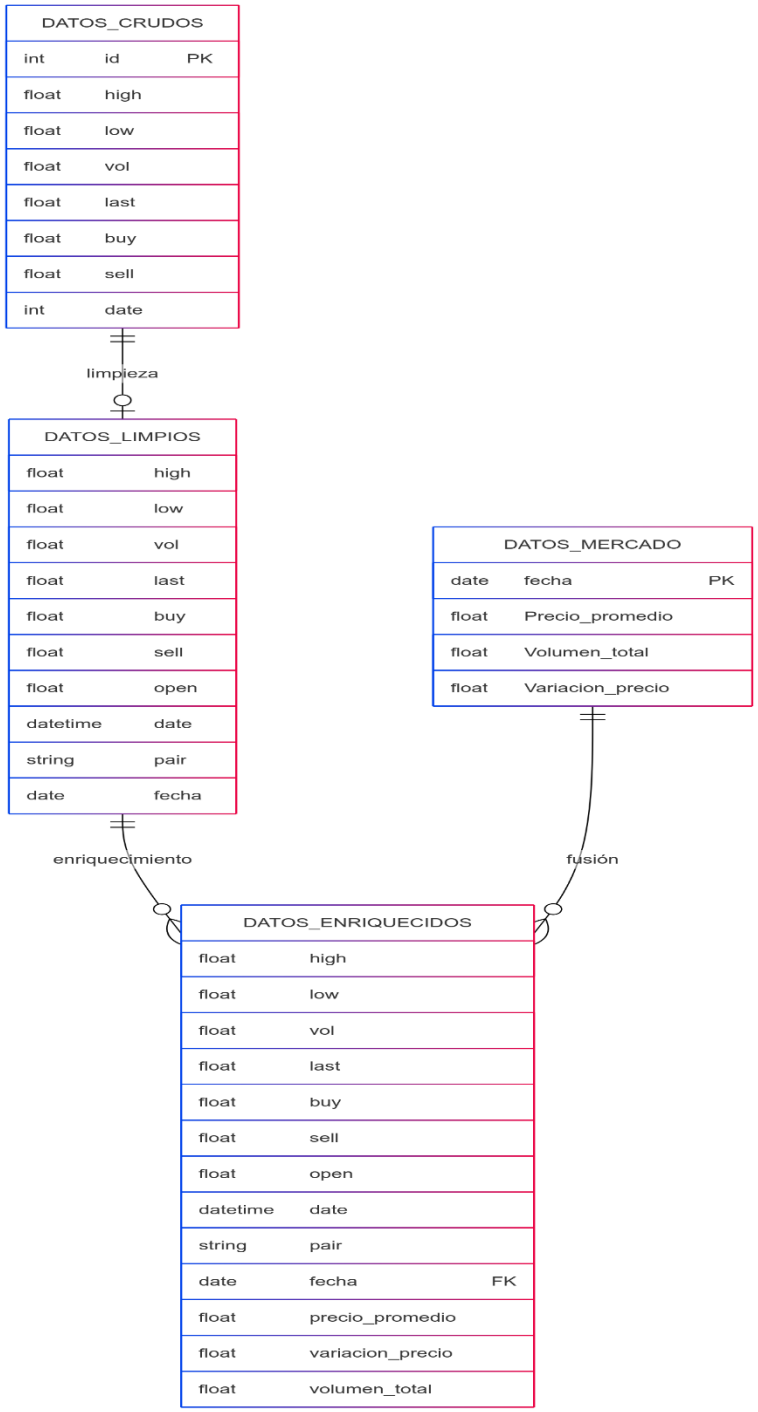
Archivo: enriched_data.csv

Campo	Tipo	Descripción	Origen
High	Float	Precio más alto	cleaned_data.csv
Low	Float	Precio más bajo	cleaned_data.csv
Vol	Float	Volumen	cleaned_data.csv
Last	Float	Ultimo precio	cleaned_data.csv
Buy	Float	Precio de compra	cleaned_data.csv
Sell	Float	Precio de venta	cleaned_data.csv
Open	Float	Precio de apertura	cleaned_data.csv
Date	Datetime	Fecha en forma datetime	cleaned_data.csv
Pair	String	Par de trading	cleaned_data.csv
Fecha	Date	Fecha para matching	Script enrichment.py
Precio_Promedio	Float	Precio promedio	market_data.xlsx
Volumen_Total	Float	Volumen total	market_data.xlsx
Variacion_Precio	Float	Indicador de variación	market_data.xlsx

Este modelo final integra datos operativos del mercado de criptomonedas con información contextual macroeconómica, permitiendo análisis multidimensionales.

Diagrama del Modelo de Datos

El siguiente diagrama entidad-relación (ER) ilustra la estructura y relaciones del modelo de datos:



Este diagrama muestra las entidades principales que componen el modelo de datos, los atributos específicos asociados a cada una de estas entidades, las relaciones lógicas que existen entre ellas y el flujo detallado de transformación de los datos.

Justificación.

La justificación detrás del diseño de este modelo se fundamenta en una serie de principios y objetivos clave. En primer lugar, se busca una progresión lógica de la información, donde el modelo evoluciona secuencialmente a través de fases de datos crudos (preservando la información original), datos limpios (normalizados y con inconsistencias corregidas) y datos enriquecidos (integrados con fuentes contextuales), asegurando así una trazabilidad completa del proceso y la capacidad de auditar y reconstruir cualquier transformación aplicada. En segundo lugar, la estructura final del modelo está optimizada para facilitar diversos tipos de análisis, incluyendo el análisis temporal mediante la normalización de fechas, el análisis correlacional a través de la integración de variables macroeconómicas, el análisis técnico con la preservación de indicadores clave de trading, y el análisis fundamental mediante la incorporación de factores externos que influyen en el mercado. En tercer lugar, el proceso de limpieza aplica principios de normalización de forma incremental, abordando la eliminación de valores nulos mediante sustituciones estadísticamente representativas, la estandarización de tipos de datos para garantizar la consistencia, y el filtrado de anomalías para evitar la distorsión del análisis. En cuarto lugar, el modelo utiliza la fecha como dimensión principal para una integración natural de los datos, lo que implica la conversión a formatos estándar, la extracción de componentes relevantes y el mantenimiento de una granularidad diaria adecuada para el análisis macroeconómico. Finalmente, el diseño del modelo prioriza la extensibilidad y la escalabilidad, presentando una estructura abierta con capacidad para incorporar nuevas dimensiones o métricas, independencia entre componentes para permitir la evolución individual de cada entidad, y compatibilidad con diversas herramientas analíticas gracias al uso de formatos estándar. Esta estructura integral facilita tanto el análisis operativo del comportamiento del precio de BTC como análisis más sofisticados que relacionan el activo con variables macroeconómicas y de mercado, lo que amplía significativamente el valor analítico de la información.

Justificación de Herramientas y Tecnologías

Elección de Herramientas

La selección de tecnologías para el proyecto se realizó considerando el balance entre capacidades técnicas, facilidad de implementación y relevancia en el ecosistema actual de Big Data.

Python como Lenguaje Principal

Justificación:

- **Versatilidad**: Capacidad para manejar desde la ingesta de datos hasta el análisis avanzado
- **Ecosistema de datos**: Amplio conjunto de bibliotecas especializadas para ciencia de datos
- **Comunidad activa**: Extensa documentación y soporte de la comunidad
- **Integración**: Compatibilidad con diversas fuentes de datos, APIs y servicios cloud
- **Curva de aprendizaje**: Sintaxis legible que facilita el desarrollo y mantenimiento

Contribución a la escalabilidad: Python permite la migración natural hacia frameworks de procesamiento distribuido como PySpark, manteniendo gran parte del código y los conceptos implementados.

Contribución a la eficiencia: Las bibliotecas optimizadas como NumPy y Pandas implementan operaciones vectorizadas que aprovechan eficientemente los recursos computacionales.

Contribución a la mantenibilidad: La legibilidad del código y la estructura modular facilitan la documentación, testing y extensión del sistema.

SQLite como Base de Datos

Justificación:

- **Ligereza**: No requiere instalación de servidor o configuración compleja
- **Portabilidad**: La base de datos completa se almacena en un único archivo
- **Compatibilidad SQL**: Soporte para el estándar SQL, facilitando la transición a sistemas más robustos

- **Sin dependencias externas**: Funciona sin infraestructura adicional
- **Rendimiento adecuado**: Eficiente para volúmenes moderados de datos

Contribución a la escalabilidad: Aunque limitado para Big Data real, proporciona una base conceptual sólida que facilita la migración posterior a sistemas como PostgreSQL, MySQL o soluciones cloud.

Contribución a la eficiencia: El modelo file-based elimina la sobrecarga de comunicación cliente-servidor, optimizando el rendimiento en entornos de desarrollo.

Contribución a la mantenibilidad: La simplicidad operativa reduce la complejidad del sistema y los puntos de fallo potenciales.

Pandas para Manipulación de Datos
Justificación:

- **Estructuras optimizadas**: DataFrame y Series proporcionan abstracciones eficientes
- **Funcionalidad integrada**: Amplio conjunto de operaciones para limpieza y transformación
- **Integración con diversos formatos**: Soporte nativo para CSV, Excel, SQL, JSON
- **Análisis estadístico**: Funciones integradas para estadísticas descriptivas
- **Manejo de fechas**: Capacidades avanzadas para series temporales

Contribución a la escalabilidad: Conceptos aplicables a frameworks distribuidos como Dask o PySpark.

Requests para Integración con APIs
Justificación:

- **API intuitiva**: Simplicidad en la implementación de llamadas HTTP
- **Gestión robusta de errores**: Manejo consistente de excepciones y códigos de estado
- **Soporte completo HTTP**: Implementación de todos los métodos y características relevantes

Contribución a la escalabilidad: Abstracción que permite reemplazar la implementación subyacente sin cambiar la interfaz.

Contribución a la eficiencia: Implementación optimizada con soporte para conexiones persistentes.

Contribución a la mantenibilidad: Separación clara entre la lógica de negocio y los detalles de comunicación HTTP.

Simulación del Entorno Cloud

El proyecto implementa una simulación conceptual de un entorno de Big Data en la nube, aplicando los siguientes enfoques:

1. Abstracción de Almacenamiento

- **Estructura de directorios**: Organización en carpetas que simulan buckets de almacenamiento

Copiar

```
src/
├── static/
│   ├── db/          # Almacenamiento de bases de datos
│   ├── csv/         # Almacenamiento de archivos tabulares
│   ├── auditoria/   # Logs y reportes de validación
│   └── enriched_data/ # Datos finales procesados
```

- **Referencia por rutas relativas**: Acceso a recursos mediante paths que abstraen la ubicación física
- **Separación de datos crudos y procesados**: Aplicación del principio de inmutabilidad de datos original

2. Procesamiento por Etapas

- **Componentes independientes**: Cada script representa un servicio discreto con responsabilidad única
- **Comunicación basada en archivos**: Simulación de transferencia entre servicios mediante almacenamiento compartido
- **Validación entre etapas**: Implementación de checkpoints que verifican la integridad antes de avanzar

3. Logs y Auditoría

- **Archivos de auditoría estructurados**: Generación de reportes en formato JSON dentro de archivos de texto
- **Métricas de procesamiento**: Registro de estadísticas como recuentos de registros y columnas
- **Validación de resultados**: Verificación de la consistencia entre datos de entrada y salida

4. Independencia de Componentes

- **Interfaces definidas**: Cada componente espera inputs específicos y genera outputs bien definidos
- **Configuración flexible**: Parametrización de rutas y nombres de archivo
- **Desacoplamiento**: Minimización de dependencias directas entre componentes

Esta estructura simula conceptualmente servicios cloud como:

- **Almacenamiento**: Equivalente conceptual a Amazon S3 o Google Cloud Storage
- **Procesamiento**: Análogo a funciones Lambda o Cloud Functions
- **Base de datos**: Versión simplificada de servicios como RDS o Cloud SQL
- **Monitoreo**: Representación básica de servicios como CloudWatch o Cloud Monitoring

Flujo de Datos y Automatización

Explicación del Flujo

Fase 1: Ingesta de Datos

1. **Extracción desde API**

El script ingestion.py inicia una solicitud HTTP a la API de MercadoBitcoin

Se especifica el par de trading (BTC) y el método (ticker)

La API devuelve información actualizada del mercado en formato JSON

2. **Procesamiento Inicial**

Se parsea la respuesta JSON y se almacena en la variable datos

Se verifica la integridad de la respuesta con validaciones básicas

Se muestra la respuesta formateada para verificación visual

3. **Almacenamiento Multiformato**

JSON: Se guarda la respuesta completa sin modificar en src/static/db/ingestion.json

SQLite: Se extrae la información relevante (ticker) y se almacena en una tabla relacional

CSV: Se convierte a formato tabular para facilitar el procesamiento posterior

4. **Validación y Auditoría**

Se compara la estructura de datos en memoria con los archivos generados

Se verifica la cantidad de registros y columnas

Se comprueba la existencia de los archivos generados

Se crea un reporte de auditoría en src/static/auditoria/ingestion.txt

El siguiente fragmento de código muestra el núcleo de la fase de ingesta:

```
ingestion = Ingestion()
parametros = {"coin":"BTC","method":"ticker"}
url = "https://www.mercadobitcoin.net/api"
datos = ingestion.obtener_datos_api(url=url, params=parametros)
if len(datos)>0:
    print(json.dumps(datos,indent=4))
else:
    print("no se obtubo la consulta")
ingestion.guardar_datos(datos=datos,nombre_archivo="ingestion")
ingestion.guardar_db(datos=datos,nombre_archivo="ingestion")
ingestion.guardar_csv(datos=datos,nombre_archivo="ingestion")
ingestion.validar_auditoria(datos=datos,nombre_archivo="ingestion")
```

Fase 2: Limpieza de Datos

1. **Preparación de Datos de Prueba**

Se generan datos sintéticos con inconsistencias intencionadas para probar la robustez del proceso

Se introducen valores nulos en posiciones específicas

Se agregan valores extremos y erróneos para verificar el manejo de anomalías

2. **Carga de Datos Existentes**

Se lee el CSV generado en la fase de ingesta

Se verifica la existencia del archivo antes de proceder

3. **Combinación de Datos**

Se concatenan los datos existentes con los sintéticos

Se crea un conjunto de prueba más completo y desafiante

4. **Proceso de Limpieza**

Manejo de valores vacíos: Sustitución de cadenas vacías por NaN

Eliminación de filas incompletas: Descarte de registros sin par de trading

Conversión de tipos: Transformación a tipos numéricos adecuados

Filtrado de anomalías: Eliminación de volúmenes extremos y valores negativos

Imputación de valores: Relleno de nulos con estadísticos (medianas)

Normalización de fechas: Conversión de timestamps a formato datetime

5. **Almacenamiento de Resultados**

Se guarda el dataset limpio en src/static/csv/cleaned_data.csv

El núcleo del proceso de limpieza se implementa en la función:

```
def limpiar_datos(df):
    # 1Reemplazar valores vacíos o nulos en cada columna
    df.replace("", np.nan, inplace=True)
    df.dropna(subset=["pair"], inplace=True)

    # 2Eliminar filas con errores en datos numéricos
    # Convertir a numérico, forzando errores a NaN
    cols_numericas = ["high", "low", "vol", "last", "buy", "sell",
"open", "date"]
    for col in cols_numericas:
        df[col] = pd.to_numeric(df[col], errors="coerce")

    # 3Manejar valores extremos
    df = df[df["vol"] < 1000]
    df = df[df["last"] >= 0]

    # 4Rellenar valores nulos con la media (o mediana si hay sesgo)
    df.fillna({
        "high": df["high"].median(),
```

```
        "low": df["low"].median(),
        "vol": df["vol"].median(),
        "last": df["last"].median(),
        "buy": df["buy"].median(),
        "sell": df["sell"].median(),
        "open": df["open"].median(),
        "date": df["date"].median()
    }, inplace=True)

    # 5Corregir formato de fechas
    df["date"] = pd.to_datetime(df["date"], unit="s", errors="coerce")

    return df
```

Fase 3: Enriquecimiento

1. **Carga de Fuentes de Datos**

Se lee el CSV limpio generado en la fase anterior

Se carga información adicional desde un archivo Excel externo

2. **Preparación para Integración**

Se normalizan los formatos de fecha en ambos conjuntos

Se extrae el componente de fecha para usarlo como clave de integración

3. **Fusión de Datos**

Se realiza un "left join" usando la fecha como clave

Se preservan todos los registros originales incluso sin correspondencia

4. **Almacenamiento del Resultado Final**

Se guarda el dataset enriquecido en src/static/enriched_data/enriched_data.csv

5. **Auditoría del Proceso**

Se verifica la existencia del archivo generado

Se comparan las estructuras antes y después del enriquecimiento

Se identifican nuevas columnas añadidas

Se analizan valores faltantes

Se identifican posibles duplicados

Se genera un reporte en src/static/auditoria/auditoria_enrich.txt

El código principal de enriquecimiento:

```
def enrich_data():

    df_clean = pd.read_csv("src/static/csv/cleaned_data.csv")
    df_new_data = pd.read_excel("src/market_data.xlsx")

    def parse_date(df, date_column):
        df[date_column] = pd.to_datetime(df[date_column])
        df["fecha"] = df[date_column].dt.date
        return df


    parse_date(df_clean, "date")
    parse_date(df_new_data, "fecha")


    df_merged = pd.merge(df_clean, df_new_data, on="fecha", how="left")

    df_merged.to_csv("src/static/enriched_data/enriched_data.csv",
index=False)

    print("Archivo guardado en
src/static/enriched_data/enriched_data.csv")
    print("Data enriquecida.")
```

Conclusiones y recomendaciones

El pipeline de datos implementado ofrece notables ventajas gracias a su arquitectura modular que separa claramente las fases de ingesta, limpieza y enriquecimiento, facilitando el mantenimiento y la extensibilidad del sistema. La flexibilidad para almacenar datos en múltiples formatos (JSON, SQLite, CSV) proporciona opciones adaptables a diferentes necesidades analíticas, mientras que los mecanismos integrados de auditoría y validación garantizan la trazabilidad y confiabilidad de los datos procesados. Sin embargo, el sistema presenta limitaciones importantes como el acoplamiento entre componentes mediante rutas fijas, la ejecución secuencial que podría generar cuellos de botella con volúmenes mayores, y un esquema relativamente rígido que podría complicar la adaptación a cambios en las fuentes de datos. Estas características hacen que la solución sea adecuada para el entorno de simulación local actual, pero requeriría modificaciones sustanciales para escalar eficientemente en un contexto de producción en la nube.

Recomendaciones

Para mejorar el sistema actual y facilitar su migración a la nube, recomendamos implementar orquestación de flujos con herramientas como Airflow para gestionar dependencias entre tareas, contenerizar los componentes con Docker para mayor portabilidad, y separar la configuración del código base. Al migrar a la nube, considere utilizar arquitectura serverless para el procesamiento intermitente de datos, servicios administrados para almacenamiento como S3 o Cloud Storage, y soluciones ETL gestionadas que reduzcan el mantenimiento manual. Es importante también implementar monitoreo automatizado con alertas, mejorar la seguridad mediante IAM y cifrado, y desarrollar un sistema de catalogación de datos que documente el linaje y facilite el gobierno de datos. Estas mejoras transformarían su simulación local en una solución cloud escalable y resiliente, manteniendo la modularidad y trazabilidad que ya caracterizan su diseño actual.