# F25 - EA4. Documentación de la Arquitectura y Modelo de Datos


**Profesor**

Andrés Felipe Callejas Jaramillo

**Estudiantes**

Davinson Stiven Rincón Campos

Estefanía Jiménez Tabares



**Institución Universitaria Digital De Antioquia.**

**Infraestructura y arquitectura para Big Data**

**PREICA2501B010112**

**06/04/2025**

**Medellín-Ant.**

# Documentación de la Arquitectura y Modelo de Datos

## Introducción

El crecimiento exponencial de los datos ha impulsado el desarrollo de nuevas arquitecturas orientadas a su recolección, procesamiento y análisis eficiente. En este contexto, el presente documento describe detalladamente la arquitectura desarrollada para un proyecto integrador de Big Data en un entorno simulado de nube. A lo largo del documento se presentan las fases fundamentales del procesamiento de datos: ingesta, preprocesamiento y enriquecimiento, además de detallar el modelo de datos resultante. Esta documentación busca servir como guía técnica y metodológica para futuras implementaciones en contextos similares o en ambientes reales de computación en la nube.

### 1. Descripción General de la Arquitectura

#### *Visión Global*

El proyecto integrador de Big Data se desarrolla en un entorno simulado de nube, emulando un pipeline de procesamiento de datos típico en soluciones analíticas modernas. La arquitectura está diseñada en fases secuenciales: ingesta, preprocesamiento y enriquecimiento, las cuales interactúan entre sí a través de una base de datos analítica central (SQLite), y están orquestadas por mecanismos de automatización (GitHub Actions) que aseguran la ejecución continua y trazable del flujo de trabajo.

- Ingesta de Datos: Se realiza mediante la conexión a una API pública desde donde se extrae información estructurada en formato JSON. Esta información es transformada y almacenada localmente en una base de datos SQLite.

- Preprocesamiento: Consiste en aplicar técnicas de limpieza y simulación de ruido sobre los datos originales, preparando así el dataset para posteriores análisis o enriquecimientos. Esta fase se encarga de corregir inconsistencias y estructurar los datos de forma estandarizada.

- Enriquecimiento: Incorpora datos adicionales provenientes de fuentes complementarias (como archivos CSV) y combina esta nueva información con los

registros ya existentes, generando así un conjunto de datos más completo y robusto.

Cada una de estas fases es autónoma y modular, lo que permite que el flujo de datos pueda escalar, adaptarse y ejecutarse de forma automática en entornos controlados o simulados.

### *Componentes Principales*

#### Base de Datos Analítica – SQLite

La base de datos central del proyecto es un archivo SQLite (ingestion.db), elegido por su facilidad de integración con Python, bajo peso y portabilidad. Funciona como repositorio analítico local que concentra la información extraída desde el API, posteriormente limpiada y enriquecida.

#### Scripts de Procesamiento

Los procesos están implementados en Python, dividiéndose en distintos scripts localizados en el directorio src/:

- ingestion.py: conecta al API, descarga los datos, los transforma y los almacena en SQLite. Además, genera archivos de evidencia (ingestion.xlsx, ingestion.txt).

- ensuciar_datos.py: simula errores, inconsistencias y ruido en los datos originales para representar condiciones del mundo real.

- enrichment.py: integra información adicional desde archivos externos (por ejemplo, CSV con datos complementarios como idiomas, regiones, etc.).

- simulacion_procesamiento.py: representa el ciclo completo de integración y transformación en pruebas.

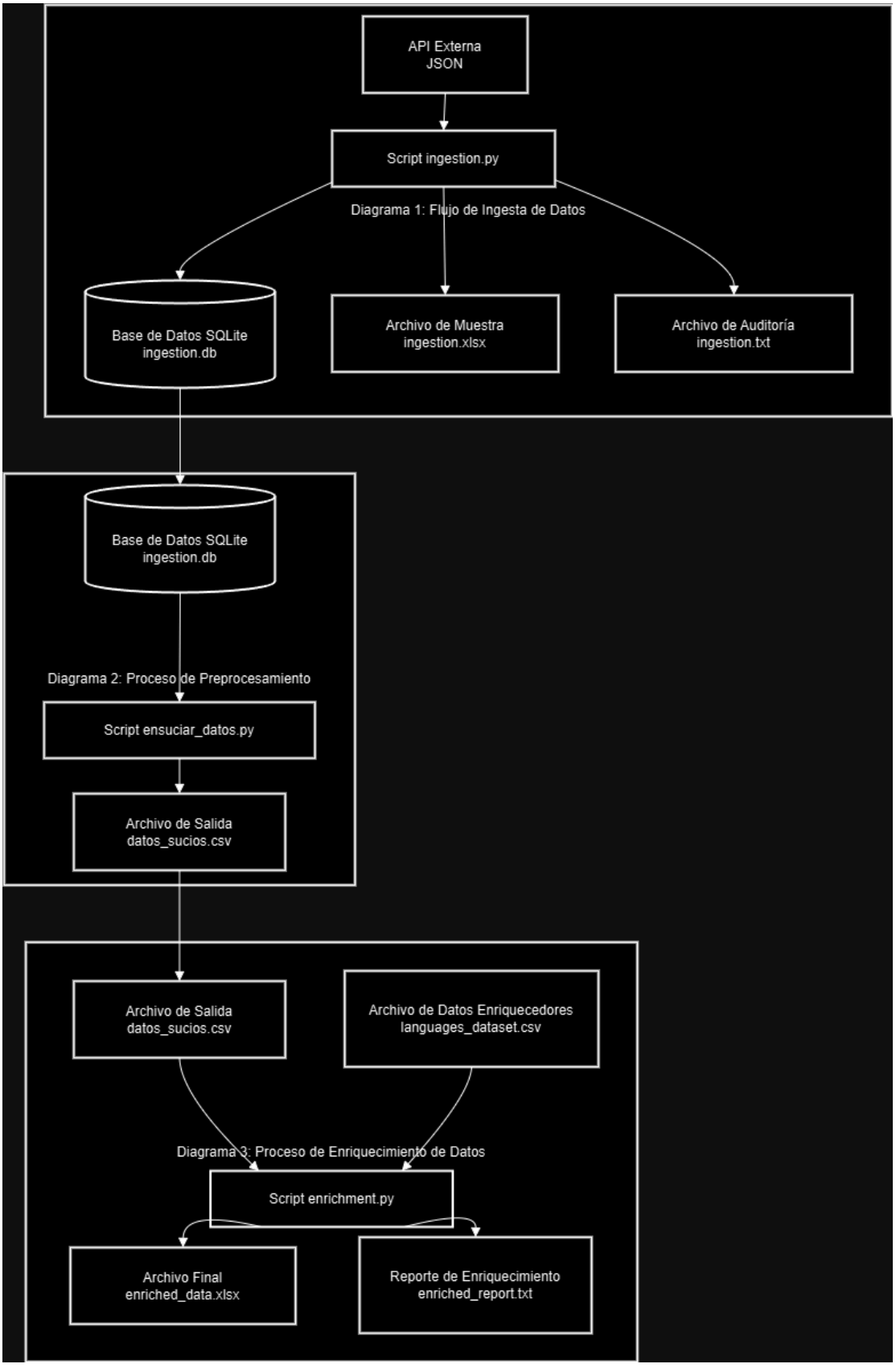
#### Mecanismo de Automatización – GitHub Actions

El proyecto está alojado en GitHub, y cuenta con un archivo de workflow YAML (.github/workflows/bigdata.yml) que permite ejecutar automáticamente todo el flujo de procesamiento cada vez que se hace un push o de forma programada.

Este mecanismo garantiza que:

- Los scripts se ejecutan en orden correcto.
- La base de datos se actualiza.
- Se generan los archivos de auditoría y muestra.
- Todos los artefactos se almacenan en la estructura esperada (static/db, static/xlsx, static/auditoria).

## 2. Diagramas de Arquitectura



## 3. Modelo de Datos

### *Definición del Esquema:*

El modelo de datos propuesto está centrado en una base de datos analítica implementada en SQLite, compuesta por una tabla principal y conjuntos de datos complementarios generados durante las fases de preprocesamiento y enriquecimiento. Este modelo sigue un enfoque estructurado para facilitar la trazabilidad, auditoría, análisis y enriquecimiento de la información.

### *Tablas Involucradas*

#### A. Tabla principal: registros_api

Esta tabla almacena los datos directamente extraídos del API. La estructura puede variar según el origen de los datos, pero típicamente incluye:

| Campo | Tipo de Dato | Descripción |
|---|---|---|
| `id` | INTEGER | Clave primaria autoincremental |
| `nombre` | TEXT | Nombre o identificador principal del registro |
| `pais` | TEXT | País asociado al registro |
| `valor` | REAL | Valor numérico representativo (ej. puntuación) |
| `fecha` | TEXT | Fecha de obtención o referencia (ISO 8601) |

#### B. Tabla simulada: `registros_sucios` (derivado de `datos_sucios.csv`)

Esta tabla (representada como CSV) contiene los datos originales con errores simulados, usada para procesos de validación y limpieza. Tiene los mismos campos que registros_api, pero puede incluir:

- Campos con valores nulos

- Registros duplicados

- Formatos incorrectos

**C. Tabla de enriquecimiento: registros_enriquecidos**

Resultado de la fusión de registros_sucios con un dataset externo (languages_dataset.csv), añade nuevos campos para análisis:

| Campo | Tipo de Dato | Descripción |
|---|---|---|
| `nombre` | TEXT | Identificador original |
| `pais` | TEXT | País de origen |
| `valor` | REAL | Valor original |
| `idioma_oficial` | TEXT | Idioma oficial del país (desde dataset externo) |
| `region` | TEXT | Región geográfica del país |

Esta tabla no está en SQLite; se genera como enriched_data.xlsx pero su lógica relacional se puede representar como tabla integrada.

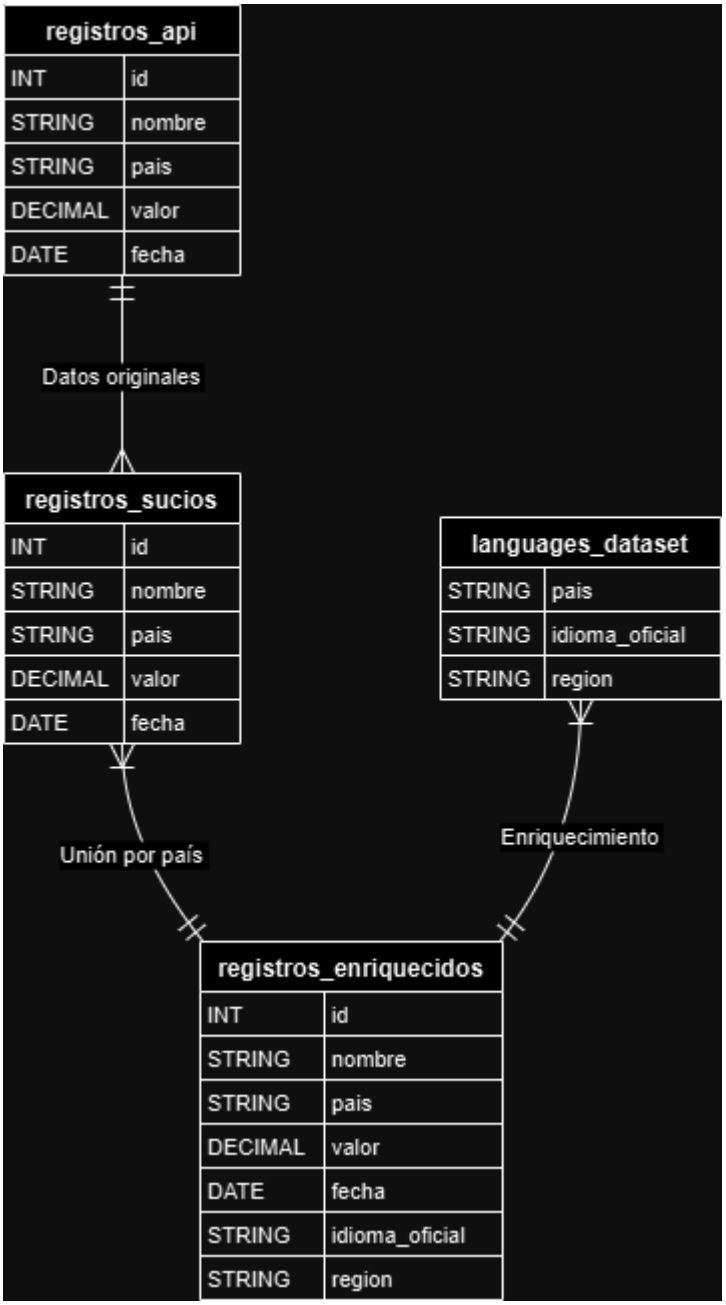
***Relaciones entre Tablas***

Aunque el proyecto no implementa relaciones directas entre múltiples tablas en la base de datos (por mantenerse simple en SQLite), conceptualmente el modelo considera:

- registros_enriquecidos es una extensión de registros_sucios, unidos por el campo pais, que actúa como clave foránea lógica para obtener información de idioma y región.

- El campo nombre puede ser considerado como clave candidata en análisis posteriores.

**Diagrama de datos:**




**Justificación del Modelo**

Este modelo de datos fue diseñado con base en los siguientes principios:

- ***Simplicidad y flexibilidad****:* Al usar una estructura tabular sencilla y separada por fases (ingesta, preprocesamiento, enriquecimiento), se facilita la comprensión, trazabilidad y mantenimiento del sistema.

- ***Modularidad****:* Cada tabla representa una fase distinta del flujo de datos, lo que permite detectar y corregir errores sin afectar el resto del sistema.

- ***Escalabilidad lógica***: Aunque SQLite es una base de datos local, el esquema puede migrarse fácilmente a un sistema distribuido (como PostgreSQL o BigQuery) en un entorno de nube real.

- ***Soporte para auditoría y análisis****:* La separación entre datos crudos, sucios y enriquecidos permite aplicar análisis comparativos, validaciones y enriquecimientos sin perder la integridad del dataset original.

## 4. Justificación de Herramientas y Tecnologías

### *Elección de Herramientas*

La arquitectura del proyecto integrador de Big Data fue diseñada para simular un entorno de procesamiento en la nube utilizando herramientas ampliamente utilizadas en la industria. La elección de estas herramientas se realizó con base en criterios de **eficiencia, escalabilidad, automatización y mantenibilidad**.

### *SQLite*

- **Motivo de elección:** Es una base de datos relacional ligera, de fácil integración con Python, sin necesidad de instalar un servidor de bases de datos adicional.

- **Contribución al proyecto:**

    - Permite simular un **data warehouse** local donde se centralizan los datos procesados.

    - Facilita operaciones SQL estándar para validar y consultar datos durante las etapas de ingesta y enriquecimiento.

    - Es ideal para entornos educativos y prototipos donde la simplicidad y portabilidad son clave.

### *Pandas*

- **Motivo de elección:** Biblioteca fundamental para la manipulación y análisis de datos estructurados en Python.

- **Contribución al proyecto:**

    - Se utilizó en todas las fases del proyecto: para cargar datos desde el API, limpiar registros sucios, realizar transformaciones y generar archivos de evidencia (.csv, .xlsx).

- Su compatibilidad con SQLite permite leer y escribir directamente en la base de datos.

- Brinda flexibilidad y rapidez para pruebas y validaciones, siendo ideal para un entorno de desarrollo iterativo.

***PySpark (simulado)***

- **Motivo de elección:** PySpark es una tecnología usada en entornos de Big Data reales para procesamiento distribuido a gran escala.

- **Contribución al proyecto:**

  - Aunque no se ejecutó en un clúster real, el uso de estructuras similares (DataFrames, transformaciones, joins) permite **simular la lógica de un entorno Spark** en local.

  - La separación por scripts (simulacion_procesamiento.py) y uso de archivos como entrada/salida reflejan cómo funcionaría el proceso en una arquitectura basada en Spark o Databricks.

***GitHub Actions***

- **Motivo de elección:** Plataforma de CI/CD que permite automatizar flujos de trabajo directamente desde el repositorio.

- **Contribución al proyecto:**

  - Permite ejecutar automáticamente el proceso de **ingesta de datos, actualización de la base de datos y generación de evidencias**.

  - Simula un entorno cloud donde un job se activa en función de un trigger (push al repositorio).

- Mejora la mantenibilidad y repetibilidad del proceso, permitiendo que cualquier cambio en el código dispare automáticamente la actualización de los datos.

### Simulación del Entorno Cloud

El entorno cloud fue simulado de forma efectiva utilizando herramientas locales y workflows automatizados que replican los componentes de una arquitectura de Big Data en la nube:

| Componente Simulado | Herramienta | Equivalente en Nube |
|---|---|---|
| API externa | `requests` en Python | API REST pública |
| Ingesta y almacenamiento | `sqlite3` + `pandas` | Data ingestion + Data Lake / Warehouse |
| Procesamiento batch | Scripts Python (`ensuciar_datos.py`, `simulacion_procesamiento.py`) | Spark Jobs o Glue ETL |
| Enriquecimiento | `pandas.merge` con dataset externo | Lookup o Join en BigQuery/Spark |
| Automatización CI/CD | GutHub Actoins | Orquestación con Airflow, Azure Data Factory o CI pipelines |

Cada fase fue implementada en un script individual y organizada dentro de carpetas (src/static/) que simulan rutas de almacenamiento jerárquico como en servicios tipo S3 o HDFS.

Además:

- La ejecución de scripts mediante GitHub Actions simula el **despliegue y operación continua en una nube pública**, donde los procesos se ejecutan sin intervención manual.

- La **generación de archivos persistentes** (ingestion.db, ingestion.xlsx, ingestion.txt) refleja cómo se estructuran los outputs en un sistema de análisis o monitoreo basado en cloud.

## 5. Flujo de Datos y Automatización

### *Explicación del Flujo de Datos*

El flujo de datos del proyecto fue diseñado para replicar el ciclo completo de un pipeline de procesamiento de datos en un entorno simulado de nube. Este flujo sigue una secuencia de etapas bien definidas que garantizan la calidad, consistencia y trazabilidad de los datos desde su origen (el API) hasta su integración final en un modelo de datos consolidado.

---

### *1. Ingesta de Datos desde el API*

- **Script responsable:** ingestion.py
- **Herramienta:** requests, pandas, sqlite3
- **Descripción:**
    - Se realiza una conexión a una API pública mediante una solicitud HTTP.
    - La respuesta se procesa y estructura en un DataFrame de Pandas.
    - Los datos se insertan en una base de datos SQLite (ingestion.db), donde se almacena la tabla principal.
    - Se genera una muestra representativa en Excel (ingestion.xlsx) y un archivo de auditoría (ingestion.txt) para validar que la ingesta fue exitosa.

---

### *2. Ensuciamiento Simulado*

- **Script responsable:** ensuciar_datos.py
- **Herramienta:** pandas
- **Descripción:**
    - A partir de los datos almacenados en la base de datos, se simulan errores comunes (valores nulos, duplicados, inconsistencias).

- Esta etapa prepara los datos para ser tratados en el proceso de preprocesamiento, imitando lo que ocurre en escenarios reales donde los datos no siempre llegan limpios.

### *3. Preprocesamiento de Datos*

- **Script responsable:** simulacion_procesamiento.py
- **Herramienta:** pandas
- **Descripción:**
  - Se eliminan duplicados y registros nulos.
  - Se realizan transformaciones básicas como cambio de tipos de datos, normalización o estandarización de campos.
  - Se crean nuevas variables si es necesario, como campos categóricos derivados de datos originales.

### *4. Enriquecimiento de Datos*

- **Script responsable:** enrichment.py
- **Herramienta:** pandas, os, openpyxl
- **Descripción:**
  - Se integra un dataset adicional (languages_dataset.csv) como fuente externa.
  - Mediante merge o join, se agregan columnas nuevas al dataset principal (por ejemplo, idioma, país, grupo, etc.).
  - Se genera una versión enriquecida de los datos, lista para ser analizada o modelada.

### *5. Construcción del Modelo de Datos*

- **Estructura generada:** ingestion.db (Base de datos SQLite)
- **Descripción:**

- Todas las etapas anteriores se reflejan en el modelo de datos final almacenado en SQLite.

- Las tablas reflejan datos originales, transformados y enriquecidos.

- La estructura del modelo de datos está diseñada para soportar consultas analíticas y posteriores fases como visualización o modelado predictivo.

**Automatización con GitHub Actions**

- ***Workflow:*** .github/workflows/bigdata.yml

- ***Descripción:***

    - Se configura un flujo automatizado que se activa ante cambios en el repositorio.

    - Este workflow ejecuta:

        - La lectura del API (ingestion.py).

        - La limpieza y simulación de errores (ensuciar_datos.py).

        - El preprocesamiento y enriquecimiento (simulacion_procesamiento.py, enrichment.py).

        - La creación/actualización de la base de datos.

        - La generación de evidencias (archivos de Excel y auditoría).

    - Como resultado, se mantiene la integridad del pipeline y se asegura que todos los cambios se reflejen automáticamente, simulando una arquitectura escalable en la nube.

## 6. Conclusiones y Recomendaciones

### *Conclusiones*

El desarrollo de este proyecto permitió implementar un pipeline de procesamiento de datos completo, desde la **ingesta automatizada** hasta la **integración de datos enriquecidos**, simulando un entorno de Big Data en la nube. A lo largo del proceso se observaron varios beneficios clave:

- **Modularidad y claridad en las etapas del flujo de datos:** Cada fase (ingesta, preprocesamiento, enriquecimiento) está claramente separada, lo que facilita su mantenimiento y actualización.

- **Uso eficiente de herramientas ligeras y potentes:** Herramientas como Pandas, SQLite y GitHub Actions fueron suficientes para simular un entorno cloud automatizado, demostrando que no siempre se requieren tecnologías complejas para validar un flujo funcional.

- **Trazabilidad y generación de evidencia:** La creación de archivos de auditoría y muestras garantiza transparencia en cada paso del flujo, lo cual es esencial para proyectos de análisis de datos.

- **Automatización robusta:** La integración de GitHub Actions permitió demostrar cómo se puede lograr una ejecución automática del proceso, facilitando la reproducibilidad y continuidad del pipeline.

### *Limitaciones*

- **Escalabilidad limitada:** Aunque SQLite es ligero y fácil de usar, no está diseñado para entornos de producción con grandes volúmenes de datos o múltiples usuarios concurrentes.

- **Procesamiento local:** El uso de scripts ejecutados localmente no simula totalmente la distribución y paralelización de tareas que ofrecen entornos reales de Big Data como Apache Spark en clusters o servicios gestionados en la nube.

- **Fuentes de datos acotadas:** La simulación se basó en una API específica y un dataset complementario. En un entorno real, sería necesario integrar múltiples fuentes heterogéneas, incluyendo flujos en tiempo real.

***Recomendaciones***

1. **Migración a una base de datos más robusta:** Para entornos reales, se recomienda usar PostgreSQL, Amazon RDS, Google BigQuery o bases diseñadas para analítica como Snowflake o Redshift.

2. **Orquestación avanzada:** Implementar herramientas como Apache Airflow o Prefect permitiría tener un mayor control sobre el scheduling, la dependencia entre tareas y el monitoreo de procesos.

3. **Contenerización y despliegue en la nube:** Utilizar Docker para encapsular los scripts y desplegarlos en servicios como AWS Lambda, Google Cloud Functions o contenedores en Kubernetes mejoraría la portabilidad y escalabilidad del proyecto.

4. **Procesamiento distribuido:** Para manejar grandes volúmenes de datos, se recomienda escalar a PySpark, Dask o frameworks similares que permitan paralelizar el procesamiento.

5. **Visualización y explotación analítica:** Como complemento al modelo de datos enriquecido, se sugiere implementar dashboards interactivos con Power BI, Tableau o herramientas open source como Apache Superset.

***Bibliografía***

https://github.com/DavsRC/DavinsonRincon_EstefaniaJimenez