F25 - EA4. DOCUMENTACIÓN DE LA ARQUITECTURA Y MODELO DE DATOS

ESTUDIANTES:

JEAN CARLOS PÁEZ RAMÍREZ (GRUPO: PREICA2402B010100)

JULIANA MARÍA PEÑA SUAREZ (GRUPO: PREICA2402B010100)

DOCENTE:

ANDRÉS FELIPE CALLEJAS JARAMILLO

ESPECIALISTA EN ANALÍTICA DE DATOS

CURSO:

INFRAESTRUCTURA Y ARQUITECTURA PARA BIG DATA

PROGRAMA DE INGENIERÍA EN DESARROLLO DE SOFTWARE Y DATOS

FACULTAD INGENIERÍA Y CIENCIAS AGROPECUARIAS

INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

2025

CONTENIDO

TABLA DE FIGURAS

INTRODUCCIÓN

El presente documento describe la arquitectura y el modelo de datos del proyecto integrador de Big Data, desarrollado como parte de las actividades del curso. Este proyecto tiene como objetivo implementar un flujo completo de procesamiento de datos, desde la ingesta hasta el enriquecimiento, utilizando herramientas modernas y simulando un entorno de nube.

El proyecto se centra en la extracción de datos desde un API (Kaggle), su almacenamiento en una base de datos SQLite, la limpieza y transformación de los datos, y su enriquecimiento con información adicional proveniente de múltiples fuentes. Todo el proceso está automatizado mediante GitHub Actions, lo que asegura la reproducibilidad y facilita la integración continua.

1 DESCRIPCIÓN GENERAL DE LA ARQUITECTURA

1.1 Visión Global

La arquitectura del proyecto se basa en tres fases principales:

a) Ingesta de Datos:

- Extracción de datos desde un API (Kaggle).
- Almacenamiento de los datos en una base de datos SQLite.
- Generación de evidencias complementarias, como archivos CSV y reportes de auditoría.

b) 2. Preprocesamiento y Limpieza de Datos:

- Validación, transformación y depuración de los datos extraídos.
- Eliminación de duplicados, manejo de valores nulos y corrección de tipos de datos.
- Generación de un dataset limpio y reportes de auditoría.

c) 3. Enriquecimiento de Datos:

- Integración de información adicional proveniente de múltiples fuentes y formatos (JSON, CSV, XML, etc.).
- Generación de un dataset enriquecido y reportes de auditoría.

1.2 Componentes Principales

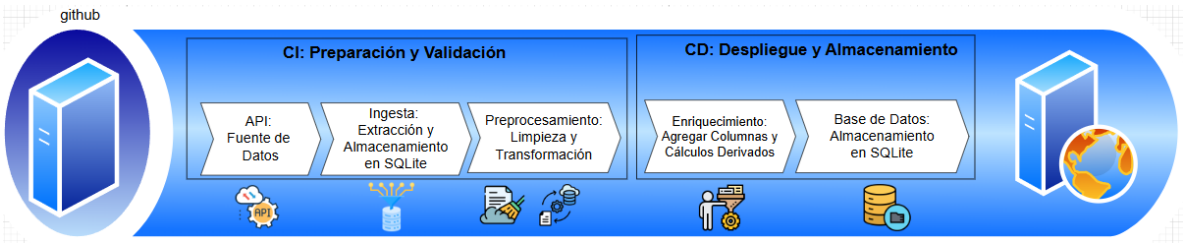
- **Base de Datos Analítica (SQLite):**
  - Almacena los datos procesados y enriquecidos, facilitando su consulta y análisis.
- **Scripts de Procesamiento:**
  - **“ingestion.py”:** Extrae los datos desde la API y los almacena en SQLite.
  - **“cleaning.py”:** Realiza la limpieza y transformación de los datos.

- **“enrichment.py”:** Enriquecer los datos con nuevas columnas y cálculos.

- **- Automatización (GitHub Actions):**
  - Ejecuta automáticamente los scripts en un flujo continuo, simulando un entorno de nube.

2 DIAGRAMAS DE ARQUITECTURA

Figura 1. Flujo de Integración y Despliegue de Datos.



2.1 Diagrama de Flujo de Datos

El flujo de datos desde la extracción de la API hasta el almacenamiento en la base de datos se ilustra en el siguiente diagrama. Este flujo está dividido en dos fases principales: **CI (Integración Continua) y CD (Despliegue Continuo)**, las cuales se describen a continuación:

2.1.1 CI: Integración Continua (Preparación y Validación)

La fase de CI se enfoca en la preparación y validación de los datos. Incluye las siguientes etapas:

- **API: Fuente de datos**

La API es el punto de partida del flujo. Aquí se obtienen los datos crudos que serán procesados en las etapas posteriores. Esta etapa representa la conexión con la fuente de datos externa.

- **Ingesta: Extracción y Almacenamiento en SQLite**

En esta etapa, los datos crudos extraídos de la API se almacenan en una base de datos SQLite. Este paso asegura que los datos estén disponibles para su procesamiento posterior.

- **Preprocesamiento: Limpieza y Transformación**

Los datos almacenados son limpiados y transformados para eliminar inconsistencias, valores nulos y errores. Este paso garantiza que los datos estén en un formato adecuado para su análisis.

2.1.2 CD: Despliegue Continuo (Entrega y Almacenamiento)

La fase de CD se enfoca en la entrega y almacenamiento de los datos procesados. Incluye las siguientes etapas:

- **Enriquecimiento: Agregar Columnas y Cálculos Derivados**

En esta etapa, se agregan nuevas columnas y se realizan cálculos derivados para enriquecer los datos. Esto incluye métricas adicionales o transformaciones que faciliten el análisis.

- **Base de Datos: Almacenamiento en SQLite**

Finalmente, los datos enriquecidos se almacenan en una base de datos SQLite, listos para ser utilizados en análisis posteriores o integrados en otros sistemas.

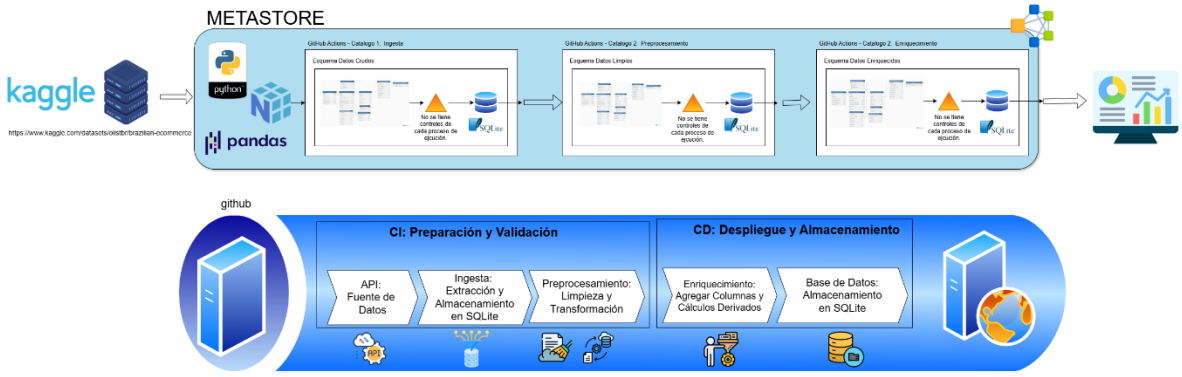
2.1.3 Automatización con GitHub Actions

Todo este flujo está automatizado mediante GitHub Actions, lo que garantiza que cada etapa se ejecute de manera eficiente, reproducible y sin intervención manual. Esto permite que el flujo sea escalable y fácil de mantener.

A continuación, se presenta el diagrama que ilustra este flujo:

2.2 Diagrama de Arquitectura

Figura 2. Arquitectura del Metastore y Pipeline de Datos.



El proyecto se basa en una infraestructura diseñada para gestionar el flujo de datos desde su extracción hasta su almacenamiento y análisis. Esta infraestructura está organizada en tres catálogos principales: Ingesta, Preprocesamiento, y Enriquecimiento, los cuales son gestionados mediante GitHub Actions como motor de automatización. A continuación, se describen las etapas y componentes clave del proyecto:

2.2.1 Kaggle como Fuente de Datos

El punto de partida del flujo de datos es Kaggle, una plataforma que proporciona el conjunto de datos utilizado en el proyecto: Brazilian E-commerce Public Dataset. Este dataset contiene información sobre pedidos, pagos, productos, vendedores y clientes, que es esencial para las etapas posteriores del pipeline.

2.2.2 Catálogo 1: Ingesta

En esta etapa, los datos crudos son extraídos desde Kaggle y almacenados en una base de datos SQLite. Este proceso es gestionado mediante GitHub Actions, que ejecuta scripts en Python utilizando librerías como:

- **Pandas:** Para la manipulación de datos.
- **NumPy:** Para cálculos numéricos.

El resultado de esta etapa es un conjunto de tablas crudas que sirven como base para las transformaciones posteriores.

2.2.3 Catálogo 2: Preprocesamiento

En esta etapa, los datos crudos son limpiados y transformados para eliminar inconsistencias, valores nulos y duplicados. Este proceso también es gestionado mediante GitHub Actions, utilizando scripts en Python con las mismas librerías mencionadas anteriormente.

El resultado de esta etapa es un conjunto de tablas limpias que están listas para ser enriquecidas. Estas tablas se almacenan nuevamente en SQLite.

2.2.4 Catálogo 3: Enriquecimiento

En la etapa de enriquecimiento, los datos limpios son transformados para agregar nuevas columnas y cálculos derivados. Por ejemplo:

- Cálculo de métricas adicionales como tasas de impuestos o rangos de peso.
- Creación de columnas derivadas para análisis más detallados.

El resultado de esta etapa es un conjunto de tablas enriquecidas que representan datos listos para análisis. Estas tablas también se almacenan en SQLite.

2.2.5 Visualización y Análisis (Futuro)

Aunque el alcance del proyecto no incluyó la etapa de visualización y análisis, se recomienda integrar herramientas de visualización como Power BI o Tableau para explorar los datos enriquecidos. Esto permitiría generar reportes y dashboards que faciliten la toma de decisiones basada en datos.

2.2.6 Recomendaciones para Mejorar el Proyecto

1) **Automatización Completa del Pipeline:**

- Actualmente, el pipeline está parcialmente automatizado mediante GitHub Actions. Se recomienda implementar un control más detallado de cada etapa para garantizar la trazabilidad y reproducibilidad de los procesos.

2) **Integración de Visualización:**
    - Incorporar herramientas como Power BI, Tableau, o incluso librerías de Python como Seaborn o Plotly para generar reportes interactivos y gráficos avanzados.
3) **Escalabilidad:**
    - Migrar el almacenamiento de datos de SQLite a una base de datos más robusta como PostgreSQL o Amazon RDS, especialmente si el volumen de datos crece en el futuro.
4) **Procesamiento Distribuido:**
    - Si el volumen de datos aumenta significativamente, considerar el uso de herramientas como Apache Spark o Dask para realizar transformaciones de datos de manera distribuida.
5) **Validación de Datos:**
    - Implementar un sistema de validación de datos en cada etapa del pipeline para garantizar la calidad de los datos procesados.
6) **Documentación y Monitoreo:**
    - Documentar cada etapa del pipeline y agregar herramientas de monitoreo para identificar posibles errores o cuellos de botella en el flujo de datos.

3 MODELO DE DATOS

3.1 Definición del Esquema

El modelo de datos resultante incluye las siguientes tablas:

3.1.1 clean_product_category_name_translation

Descripción: Contiene las traducciones de las categorías de productos.

Campos:

- **“product_category_name” (varchar):** Nombre de la categoría del producto.
- **“product_category_name_english” (varchar):** Nombre de la categoría en inglés.

3.1.2 clean_olist_sellers_dataset

Descripción: Información sobre los vendedores, incluyendo su ubicación.

Campos:

- **“seller_id” (varchar):** Identificador único del vendedor.
- **“seller_zip_code_prefix” (int):** Código postal del vendedor.
- **“seller_city” (varchar):** Ciudad del vendedor.
- **“seller_state” (varchar):** Estado del vendedor.

3.1.3 clean_olist_geolocation_dataset

Descripción: Información geográfica de ubicaciones en Brasil.

Campos:

- “**geolocation_zip_code_prefix” (int):** Código postal.
- **“geolocation_lat” (float):** Latitud de la ubicación.
- **“geolocation_lng” (float):** Longitud de la ubicación.

- **“geolocation_city” (varchar):** Ciudad de la ubicación.
- **“geolocation_state” (varchar):** Estado de la ubicación.

3.1.4 clean_olist_order_items_dataset

Descripción: Detalles de los productos incluidos en cada orden.

Campos:

- **“order_id” (varchar):** Identificador único del pedido.
- **“order_item_id” (int):** Número del ítem dentro del pedido.
- **“product_id” (varchar):** Identificador único del producto.
- **“seller_id” (varchar):** Identificador único del vendedor.
- **“shipping_limit_date” (datetime):** Fecha límite para el envío.
- **“price” (float):** Precio del producto.
- **“freight_value” (float):** Valor del flete.
- **“price_normalized” (float):** Precio normalizado del producto.

3.1.5 clean_olist_order_payments_dataset

Descripción: Detalles de los métodos de pago utilizados en las órdenes.

Campos:

- **“order_id” (varchar):** Identificador único del pedido.
- **“payment_sequential” (int):** Número secuencial del pago.
- **“payment_type” (varchar):** Tipo de pago (tarjeta, boleto, etc.).
- **“payment_installments” (int):** Número de cuotas.
- **“payment_value” (float):** Valor del pago.

3.1.6 clean_olist_order_reviews_dataset

Descripción: Contiene reseñas de los clientes sobre los pedidos realizados.

Campos:

- **“review_id” (varchar):** Identificador único de la reseña.
- **“order_id” (varchar):** Identificador único del pedido.
- **“review_score” (int):** Puntuación de la reseña (1-5).
- **“review_comment_title” (varchar):** Título del comentario.
- **“review_comment_message” (varchar):** Mensaje del comentario.
- **“review_creation_date” (datetime):** Fecha de creación de la reseña.
- **“review_answer_timestamp” (datetime):** Fecha de respuesta a la reseña.
- **“review_sentiment” (varchar):** Sentimiento asociado a la reseña.

3.1.7 enriched_olist_customers_dataset

Descripción: Información sobre los clientes, incluyendo su ubicación y segmentación.

Campos:

- “**customer_id” (varchar):** Identificador único del cliente.
- **“customer_unique_id” (varchar):** Identificador único del cliente en la plataforma.
- **“customer_zip_code_prefix” (int):** Código postal del cliente.
- **“customer_city” (varchar):** Ciudad del cliente.
- **“customer_state” (varchar):** Estado del cliente.
- **“min_purchase” (float):** Monto mínimo de compra del cliente.
- **“segment_id” (int):** Identificador del segmento del cliente.

- **“segment_name” (varchar):** Nombre del segmento del cliente.
- **“discount_rate” (float):** Tasa de descuento asociada al cliente.

3.1.8 enriched_olist_orders_dataset

Descripción: Información sobre las órdenes realizadas por los clientes.

Campos:

- **“order_id” (varchar):** Identificador único del pedido.
- **“customer_id” (varchar):** Identificador del cliente.
- **“order_status” (varchar):** Estado del pedido.
- **“order_purchase_timestamp” (datetime):** Fecha de compra.
- **“order_approved_at” (datetime):** Fecha de aprobación del pedido.
- **“order_delivered_carrier_date” (datetime):** Fecha de entrega al transportista.
- **“order_delivered_customer_date” (datetime):** Fecha de entrega al cliente.
- **“order_estimated_delivery_date” (datetime):** Fecha estimada de entrega.
- **“weight_range” (varchar):** Rango de peso del pedido.
- **“base_rate” (float):** Tarifa base del pedido.
- **“express_rate” (float):** Tarifa express del pedido.

3.1.9 enriched_olist_products_dataset

Descripción: Detalles de los productos disponibles en la plataforma.

Campos:

- - “**product_id” (varchar):** Identificador único del producto.
- **- “product_category_name” (varchar):** Categoría del producto.

- **“product_name_length” (int):** Longitud del nombre del producto.
- **“product_description_length” (int):** Longitud de la descripción del producto.
- **“product_photos_qty” (int):** Cantidad de fotos del producto.
- **“product_weight_g” (float):** Peso del producto en gramos.
- **“product_length_cm” (float):** Longitud del producto en cm.
- **“product_height_cm” (float):** Altura del producto en cm.
- **“product_width_cm” (float):** Ancho del producto en cm.
- **“category_name” (varchar):** Nombre de la categoría.
- **“tax_rate” (float):** Tasa de impuestos asociada al producto.

3.1.10 Relaciones entre las Tablas

1) clean_product_category_name_translation
   - **Claves Primarias:**
     - Ninguna clave primaria explícita.
   - **Relaciones:**
     - Relacionada con “enriched_olist_products_dataset” a través de:
       - “product_category_name” (FK) → “clean_product_category_name_translation.product_category_name”.

2) clean_olist_sellers_dataset
   - **Claves Primarias:**
     - **“seller_id” (PK):** Identificador único del vendedor.
   - **Relaciones**:
     - **Relacionada con “clean_olist_order_items_dataset” a través de:**

- “seller_id” (PK) → “clean_olist_order_items_dataset.seller_id” (FK).

❖ **Relacionada con “clean_olist_geolocation_dataset” a través de:**

- “seller_zip_code_prefix” (FK) → “clean_olist_geolocation_dataset.geolocation_zip_code_prefix”.

3) clean_olist_geolocation_dataset

➢ **Claves Primarias:**

❖ Ninguna clave primaria explícita.

➢ **Relaciones:**

❖ **Relacionada con “clean_olist_sellers_dataset” a través de:**

- “geolocation_zip_code_prefix” (PK) → “clean_olist_sellers_dataset.seller_zip_code_prefix” (FK).

❖ **Relacionada con “enriched_olist_customers_dataset” a través de:**

- “geolocation_zip_code_prefix” (PK) → “enriched_olist_customers_dataset.customer_zip_code_prefix” (FK).

4) clean_olist_order_items_dataset

➢ **Claves Primarias:**

❖ Combinación de “order_id” y “order_item_id” (PK compuesta).

➢ **Relaciones:**

❖ **Relacionada con “enriched_olist_orders_dataset” a través de:**

- “order_id” (FK) → “enriched_olist_orders_dataset.order_id” (PK).

❖ - Relacionada con “enriched_olist_products_dataset” a través de:

- “product_id” (FK) → “enriched_olist_products_dataset.product_id” (PK).

❖ - Relacionada con “clean_olist_sellers_dataset” a través de:

- “seller_id” (FK) → “clean_olist_sellers_dataset.seller_id” (PK).

5) clean_olist_order_payments_dataset

➢ **Claves Primarias:**

❖ Combinación de “order_id” y “payment_sequential” (PK compuesta).

➢ **Relaciones:**

❖ **Relacionada con “enriched_olist_orders_dataset” a través de:**

- “order_id” (FK) → “enriched_olist_orders_dataset.order_id” (PK).

6) clean_olist_order_reviews_dataset

➢ **Claves Primarias:**

❖ **“review_id” (PK):** Identificador único de la reseña.

➢ **Relaciones:**

❖ **Relacionada con “enriched_olist_orders_dataset” a través de:**

- “order_id” (FK) → “enriched_olist_orders_dataset.order_id” (PK).

7) enriched_olist_customers_dataset

- **Claves Primarias:**
  - **“customer_id” (PK):** Identificador único del cliente.
- **Relaciones:**
  - **Relacionada con “enriched_olist_orders_dataset” a través de:**
    - “customer_id” (PK) → “enriched_olist_orders_dataset.customer_id” (FK).
  - **Relacionada con “clean_olist_geolocation_dataset” a través de:**
    - “customer_zip_code_prefix” (FK) → “clean_olist_geolocation_dataset.geolocation_zip_code_prefix”.

8) enriched_olist_orders_dataset

- **Claves Primarias:**
  - **“order_id” (PK):** Identificador único del pedido.
- **Relaciones:**
  - **Relacionada con “enriched_olist_customers_dataset” a través de:**
    - “customer_id” (FK) → “enriched_olist_customers_dataset.customer_id” (PK).
  - **Relacionada con “clean_olist_order_items_dataset” a través de:**
    - “order_id” (PK) → “clean_olist_order_items_dataset.order_id” (FK).
  - **Relacionada con “clean_olist_order_payments_dataset” a través de:**
    - “order_id” (PK) → “clean_olist_order_payments_dataset.order_id” (FK).

- ❖ **Relacionada con “clean_olist_order_reviews_dataset” a través de:**
  - “order_id” (PK) → “clean_olist_order_reviews_dataset.order_id” (FK).

9) enriched_olist_products_dataset

- ➢ **Claves Primarias:**
  - ❖ **“product_id” (PK):** Identificador único del producto.
- ➢ **Relaciones:**
  - ❖ **Relacionada con “clean_product_category_name_translation” a través de:**
    - “product_category_name” (FK) → “clean_product_category_name_translation.product_category_name”.
  - ❖ **Relacionada con “clean_olist_order_items_dataset” a través de:**
    - “product_id” (PK) → “clean_olist_order_items_dataset.product_id” (FK).

3.1.11 Relaciones Clave

1) Relación entre pedidos y clientes:

- ➢ “enriched_olist_orders_dataset.customer_id” (FK) → “enriched_olist_customers_dataset.customer_id” (PK).

2) Relación entre pedidos y productos:

- ➢ “clean_olist_order_items_dataset.product_id” (FK) → “enriched_olist_products_dataset.product_id” (PK).

3) **Relación entre pedidos y pagos:**

   - “clean_olist_order_payments_dataset.order_id” (FK) → “enriched_olist_orders_dataset.order_id” (PK).

4) **Relación entre pedidos y reseñas:**

   - “clean_olist_order_reviews_dataset.order_id” (FK) → “enriched_olist_orders_dataset.order_id” (PK).

5) **Relación entre vendedores y ubicaciones:**

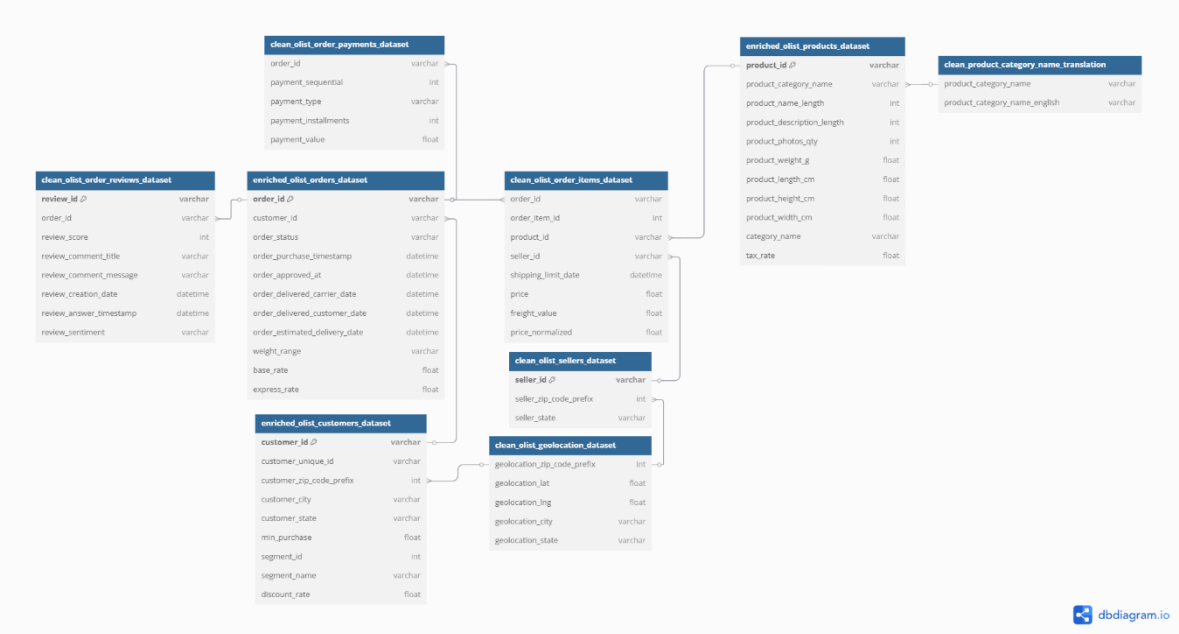
   - “clean_olist_sellers_dataset.seller_zip_code_prefix” (FK) → “clean_olist_geolocation_dataset.geolocation_zip_code_prefix” (PK).

6) **Relación entre clientes y ubicaciones:**

   - “enriched_olist_customers_dataset.customer_zip_code_prefix” (FK) → “clean_olist_geolocation_dataset.geolocation_zip_code_prefix” (PK).

3.1.12 Diagrama ER

Figura 3. Diagrama de Entidad-Relación (ER) del Proyecto.



3.1.13 Justificación del modelo de datos

El modelo de datos fue diseñado para cumplir con los siguientes objetivos principales:

3.1.13.1 Organización y Estructura Lógica

El modelo de datos se diseñó siguiendo una estructura relacional que permite organizar la información de manera lógica y eficiente. Cada tabla representa una entidad específica del negocio (clientes, pedidos, productos, vendedores, etc.), y las relaciones entre las tablas reflejan las interacciones reales entre estas entidades. Esto asegura que los datos estén bien estructurados y sean fáciles de consultar.

3.1.13.2 Normalización

El modelo sigue principios de normalización para:

- Reducir la redundancia de datos.
- Evitar inconsistencias al actualizar o eliminar registros.
- Garantizar que cada tabla almacene únicamente la información relevante a su entidad.

Por ejemplo:

- La tabla “clean_product_category_name_translation” almacena las traducciones de las categorías de productos, evitando duplicar esta información en otras tablas.
- La tabla “enriched_olist_orders_dataset” centraliza la información de los pedidos, mientras que las tablas relacionadas (“clean_olist_order_items_dataset”, “clean_olist_order_payments_dataset”, etc.) almacenan detalles específicos.

3.1.13.3 Escalabilidad

El diseño del modelo permite la integración de nuevas fuentes de datos y la expansión del sistema sin necesidad de realizar cambios significativos en la estructura existente. Por ejemplo:

- Nuevas categorías de productos pueden añadirse fácilmente a la tabla “clean_product_category_name_translation”.
- Nuevos métodos de pago pueden integrarse en la tabla “clean_olist_order_payments_dataset”.

3.1.13.4 Relaciones Claras y Consistentes

Las claves primarias y foráneas establecen relaciones claras entre las tablas, lo que facilita la integración de datos y asegura la consistencia referencial. Por ejemplo:

- La relación entre “enriched_olist_orders_dataset” y “enriched_olist_customers_dataset” asegura que cada pedido esté asociado a un cliente válido.
- La relación entre “clean_olist_order_items_dataset” y “enriched_olist_products_dataset” garantiza que cada ítem de un pedido corresponda a un producto existente.

3.1.13.5 Facilidad de Análisis

El modelo está diseñado para facilitar el análisis de datos en las etapas posteriores del proyecto. Algunas características clave incluyen:

- **Centralización de Datos:** La tabla “enriched_olist_orders_dataset” actúa como el núcleo del modelo, conectando información de clientes, productos, pagos, y reseñas. Esto permite realizar análisis integrales sobre el comportamiento de los clientes, el rendimiento de los productos y las tendencias de ventas.

- **Segmentación y Enriquecimiento:** La tabla “enriched_olist_customers_dataset” incluye información segmentada de los clientes, como su segmento y tasa de descuento, lo que facilita análisis de marketing y estrategias de fidelización.
- **Datos Geográficos:** La integración de tablas como “clean_olist_geolocation_dataset” permite realizar análisis geoespaciales, como identificar regiones con mayor volumen de ventas o tiempos de entrega más largos.

3.1.13.6 Automatización y Procesos Posteriores

El modelo de datos fue diseñado para integrarse con los scripts de procesamiento automatizados (ingesta, limpieza y enriquecimiento). Esto asegura que:

- Los datos se puedan cargar, procesar y enriquecer de manera eficiente.
- Los resultados de cada etapa se reflejen directamente en las tablas correspondientes, facilitando la trazabilidad y el control de calidad.

3.1.13.7 Adaptabilidad a Consultas Complejas

El modelo permite realizar consultas SQL complejas para responder preguntas de negocio, como:

- ¿Cuáles son los productos más vendidos por categoría?
- ¿Qué métodos de pago son más utilizados en cada región?
- ¿Qué segmentos de clientes generan mayores ingresos?

4 JUSTIFICACIÓN DE HERRAMIENTAS

4.1 Elección de Herramientas

SQLite

- Base de datos ligera y fácil de usar.
- Ideal para proyectos pequeños y medianos.
- Permite almacenar datos estructurados de manera eficiente.

Pandas

- Biblioteca de Python para la manipulación de datos.
- Facilita el preprocesamiento y enriquecimiento de datos.

GitHub Actions

- Herramienta de automatización que simula un entorno de nube.
- Permite la ejecución continua de los scripts de procesamiento.

Python

- Lenguaje de programación principal del proyecto.
- Utilizado para desarrollar los scripts de ingesta, limpieza y enriquecimiento.

4.2 Simulación del Entorno Cloud

En este proyecto, no se utilizó un entorno de nube real como AWS, Google Cloud o Azure. Sin embargo, se implementó una simulación de un entorno en la nube utilizando GitHub Actions, que permitió automatizar los procesos de ingesta, limpieza y enriquecimiento de datos de manera eficiente y reproducible. A continuación, se detalla cómo se logró esta simulación:

4.2.1 Automatización de Procesos

GitHub Actions se utilizó para automatizar la ejecución de los scripts de procesamiento (“ingestion.py”, “cleaning.py”, “enrichment.py”) en un flujo continuo. Esto simula un entorno en la nube al permitir que los procesos se ejecuten de manera remota y sin intervención manual. Cada etapa del proyecto se configuró como un paso dentro de un workflow definido en el archivo “bigdata.yml”.

4.2.2 Flujo de Trabajo Simulado

El flujo de trabajo automatizado incluye las siguientes etapas, que replican las características de un entorno en la nube:

4.2.2.1 Configuración del Entorno

- GitHub Actions configura un entorno virtual con Python 3.9 y las dependencias necesarias para ejecutar los scripts.
- Se asegura que el entorno sea limpio y reproducible en cada ejecución.

4.2.2.2 Ejecución de los Scripts

- **Ingesta de Datos:** El script “ingestion.py” extrae los datos desde la API de Kaggle y los almacena en una base de datos SQLite (“ingestion.db”).
- **Limpieza de Datos:** El script “cleaning.py” procesa los datos extraídos, eliminando duplicados, manejando valores nulos y corrigiendo inconsistencias. Los datos limpios se almacenan en una nueva base de datos SQLite (“cleaned_data.db”).
- **Enriquecimiento de Datos:** El script “enrichment.py” integra información adicional desde múltiples fuentes y genera un dataset enriquecido almacenado en “enriched_data.db”.

4.2.2.3 Gestión de Artefactos

- Los archivos generados (datasets, reportes de auditoría, bases de datos) se suben como artefactos al repositorio de GitHub, simulando el almacenamiento en un entorno de nube.

4.2.2.4 Control de Versiones

- Los cambios generados durante la ejecución de los scripts (como nuevos datasets o reportes) se versionan automáticamente mediante commits y push al repositorio remoto.

4.2.3 Reproducibilidad

La configuración de GitHub Actions asegura que el flujo de trabajo sea reproducible en cualquier momento. Esto es similar a cómo un entorno en la nube permite ejecutar procesos de manera consistente y escalable. Las principales características que garantizan la reproducibilidad incluyen:

- Definición explícita de las dependencias en el archivo “setup.py”.
- Uso de un entorno virtual limpio para cada ejecución.
- Automatización completa de los procesos mediante el archivo “bigdata.yml”.

4.2.4 Escalabilidad

Aunque GitHub Actions no es un entorno de nube real, su capacidad para ejecutar workflows en runners remotos permite simular la escalabilidad de un entorno en la nube. Por ejemplo:

- Los scripts pueden ejecutarse en paralelo o en diferentes runners para manejar grandes volúmenes de datos.
- Los artefactos generados pueden ser descargados y utilizados por otros equipos o procesos.

4.2.5 Limitaciones

Si bien GitHub Actions proporciona una simulación efectiva de un entorno en la nube, tiene algunas limitaciones en comparación con servicios reales como AWS o Google Cloud:

- **Almacenamiento:** Los artefactos generados tienen un tiempo de retención limitado (por defecto, 90 días).
- **Escalabilidad:** No es posible manejar grandes volúmenes de datos o cargas de trabajo intensivas debido a las restricciones de los runners gratuitos.
- **Integración:** No se cuenta con servicios avanzados como bases de datos distribuidas, almacenamiento en la nube o herramientas de análisis en tiempo real.

5 FLUJO DE DATOS Y AUTOMATIZACIÓN

El flujo de datos en este proyecto sigue un proceso estructurado que abarca desde la extracción inicial de los datos hasta la construcción del modelo de datos final. Este flujo está completamente automatizado mediante GitHub Actions, lo que permite simular un entorno en la nube para la ejecución de los procesos de ingesta, limpieza y enriquecimiento de datos. El código fuente y los scripts utilizados están disponibles en el repositorio oficial del proyecto: https://github.com/paez-dev/BigData2025Act1Paez_Suarez.

5.1 Ingesta de Datos

La etapa de ingesta es el punto de partida del flujo de datos. En esta fase, los datos son extraídos desde una fuente externa (API de Kaggle) y almacenados en una base de datos SQLite para su posterior procesamiento.

5.1.1 Automatización

- GitHub Actions ejecuta el script “ingestion.py” como parte del workflow definido en el archivo “bigdata.yml”.
- Se configuran las credenciales de Kaggle mediante variables de entorno seguras (“KAGGLE_USERNAME” y “KAGGLE_KEY”), asegurando la autenticidad y privacidad de la conexión.

5.1.2 Resultados

- Los datos extraídos se almacenan en una base de datos SQLite (“ingestion.db”).
- Se genera un archivo de auditoría (“ingestion.txt”) que documenta el número de registros extraídos y almacenados.

5.2 Preprocesamiento y Limpieza de Datos

En esta etapa, los datos crudos extraídos en la fase de ingesta son validados, transformados y limpiados para garantizar su calidad y consistencia.

5.2.1 Automatización

- GitHub Actions ejecuta el script “cleaning.py” después de la etapa de ingesta.
- Antes de ejecutar el script, se eliminan los archivos generados en ejecuciones anteriores para evitar conflictos o duplicados. Esto incluye bases de datos, archivos CSV, Excel, JSON, XML, HTML y TXT.

5.2.2 Resultados

- Los datos limpios se almacenan en una nueva base de datos SQLite (“cleaned_data.db”).
- Se genera un archivo de auditoría (“cleaning_report.txt”) que documenta las operaciones realizadas, como el número de registros eliminados y los valores nulos tratados.

5.3 Enriquecimiento de Datos

En esta etapa, los datos limpios son enriquecidos con información adicional proveniente de múltiples fuentes y formatos. Esto agrega valor al dataset base y lo prepara para el análisis.

5.3.1 Automatización

- GitHub Actions ejecuta el script “enrichment.py” después de la etapa de limpieza.
- Los datos adicionales se integran desde múltiples formatos, como JSON, Excel, XML y HTML, y se combinan con las tablas existentes mediante operaciones de unión (“merge”).

5.3.2 Resultados

- Los datos enriquecidos se almacenan en una nueva base de datos SQLite (“enriched_data.db”).
- Se genera un archivo de auditoría (“enriched_report.txt”) que documenta las operaciones de integración.

5.4 Manejo de Autocommits y Sincronización

El flujo incluye un manejo automatizado de commits y sincronización con el repositorio remoto, lo que asegura la trazabilidad y la actualización continua de los resultados generados.

5.4.1 Configuración de Git

- Se configuran las credenciales de Git para que los cambios sean realizados por el bot de GitHub Actions (“GitHub Actions Bot”).
- Esto asegura que los commits sean identificables y rastreables.

5.4.2 Sincronización con el Repositorio Remoto

- Antes de realizar cualquier cambio, el workflow sincroniza el repositorio local con el remoto mediante los comandos:
- **“git fetch origin main”:** Obtiene los últimos cambios del repositorio remoto.
- **“git reset --hard origin/main”:** Asegura que el repositorio local esté alineado con el remoto.
- **“git clean -fdx”:** Elimina archivos no rastreados para evitar conflictos.

5.4.3 Autocommits

- Los archivos generados (datasets, reportes de auditoría, bases de datos) se añaden al repositorio mediante el comando “git add”.

- Si hay cambios, se realiza un commit automático con el mensaje: "Actualización automática: Ingesta, limpieza de datos y evidencias del proceso de enriquecimiento de datos".
- Si no hay cambios, el workflow continúa sin errores.

5.4.4 Push Automático

- Los cambios se suben al repositorio remoto mediante “git push origin main”.
- En caso de conflictos, el workflow intenta resolverlos automáticamente con “git pull --rebase origin main”.

5.5 Verificación y Subida de Artefactos

El workflow incluye pasos para verificar y subir los artefactos generados, asegurando que los resultados estén disponibles para su análisis.

5.5.1 Verificación de Archivos

- Se verifica la existencia de los archivos generados, como bases de datos, reportes de auditoría y archivos Excel.
- Los resultados de la verificación se imprimen en la consola para facilitar el monitoreo.

5.5.2 Subida de Artefactos

- Los artefactos generados se suben al repositorio mediante el uso de “actions/upload-artifact@v4”.
- Los artefactos incluyen:
- Archivos de auditoría (“.txt”).
- Archivos CSV, Excel, JSON, XML, HTML y TXT.
- Bases de datos SQLite (“.db”).

- Los artefactos tienen una retención de 20 días, lo que asegura su disponibilidad temporal para análisis o descargas.

5.6 Beneficios del Flujo de Datos y Automatización

- **Eficiencia:** La automatización reduce el tiempo y esfuerzo manual requerido para procesar los datos.
- **Reproducibilidad:** El flujo puede ejecutarse en cualquier momento con los mismos resultados.
- **Trazabilidad:** Los autocommits y los reportes de auditoría documentan cada etapa del proceso, asegurando la transparencia y el control de calidad.
- **Escalabilidad:** La estructura del flujo permite integrar nuevas fuentes de datos y expandir el modelo de datos sin cambios significativos.

6 CONCLUSIONES

El proyecto desarrollado presenta una arquitectura sólida y bien estructurada que permite gestionar el flujo de datos desde su extracción hasta su enriquecimiento, utilizando herramientas modernas y simulando un entorno de nube mediante **GitHub Actions**. Entre los principales beneficios de la arquitectura propuesta se encuentra la automatización del flujo de trabajo, que asegura que las etapas de ingesta, limpieza y enriquecimiento de datos se ejecuten de manera eficiente, reproducible y sin intervención manual. Además, la estructura modular basada en catálogos facilita la organización y escalabilidad del pipeline, mientras que el modelo de datos relacional asegura la consistencia, evita redundancias y permite realizar consultas complejas de manera eficiente. La trazabilidad del proceso, garantizada por los reportes de auditoría y los autocommits, asegura transparencia y control de calidad en cada etapa del flujo.

Sin embargo, el proyecto también presenta algunas limitaciones. El alcance actual no incluye la etapa de visualización y análisis, lo que restringe su aplicabilidad para la toma de decisiones basada en datos. Asimismo, el uso de **SQLite**, aunque adecuado para proyectos pequeños, no es escalable para manejar grandes volúmenes de datos. La ausencia de un motor de procesamiento distribuido, como **Apache Spark**, limita la capacidad de manejar datasets más grandes o realizar transformaciones complejas. Por último, aunque GitHub Actions emula características de un entorno en la nube, no ofrece las capacidades avanzadas de servicios reales **como AWS, Google Cloud o Azure.**

7 RECOMENDACIONES

Para mejorar el proyecto y prepararlo para su implementación en un entorno real de nube, se recomienda migrar el pipeline a un servicio de nube como **AWS, Google Cloud o Azure,** lo que permitiría aprovechar características avanzadas como bases de datos distribuidas, almacenamiento escalable y procesamiento distribuido. Además, sería beneficioso reemplazar SQLite por una base de datos más robusta y escalable, como **PostgreSQL o MySQL**, para manejar mayores volúmenes de datos y permitir consultas más complejas. También se sugiere integrar herramientas de visualización como **Power BI, Tableau o librerías de Python como Plotly**, para generar dashboards interactivos y reportes que faciliten la toma de decisiones.

Asimismo, la implementación de un motor de procesamiento distribuido, como **Apache Spark o Dask**, permitiría manejar datasets más grandes y realizar transformaciones complejas de manera eficiente. Es fundamental incluir un sistema de validación de datos en cada etapa del pipeline para garantizar la calidad de los datos procesados, así como herramientas de monitoreo como **Prometheus o Grafana** para supervisar el rendimiento del pipeline y detectar posibles errores o cuellos de botella. Por último, se recomienda ampliar la automatización del pipeline para incluir notificaciones automáticas, versionado de datos y generación de vistas materializadas que aceleren análisis frecuentes.

8 BIBLIOGRAFÍA

InfoLibros.org. (2025). +20 Libros de Big Data ¡Gratis! [PDF]. Recuperado de https://infolibros.org/libros-pdf-gratis/informatica/big-data/

JimeFioni. (2025). Recursos para aprender Ciencia de Datos. Recuperado de https://github.com/JimeFioni/recursos_ciencia_datos-Books

Natayadev. (2024). Roadmap 2024 - Data engineering en español. Recuperado de https://natayadev.github.io/dataengineering-roadmap/

Ortega, J. M. (2023). Big data, machine learning y data science en Python. Barcelona: Marcombo. Recuperado de https://marcombo.com

Recursos para aprender Ciencia de Datos. (2025). Introducción a Python para ciencia de datos. Recuperado de https://github.com/camartinezbu/recursos_ciencia_datos