**Pipeline ETL (Extract, Transform, Load):**

1. **Extracción**:
   * **Vía PDF**:
     + Los PDFs se dividen en imágenes usando la biblioteca pdf2image para extraer contenido visual.
     + Simultáneamente, el contenido textual se extrae usando PyPDFLoader de langchain\_community, que procesa el texto de cada página del PDF.
2. **Transformación a JSON**:
   * Las imágenes generadas a partir de cada página se analizan usando un modelo de lenguaje (LLM). Este LLM convierte las imágenes en un archivo JSON estructurado que contiene información de KPIs, gráficos, tablas y otros elementos relevantes.
   * El JSON inicial se enriquece y corrige al combinarlo con el texto extraído, asegurando que no haya información perdida o malinterpretada.
   * El resultado es un archivo JSON detallado por cada página del PDF.
3. **Carga en la Base de Datos Vectorial**:
   * Los JSON resultantes se cargan en una base de datos vectorial (Qdrant).
   * El contenido de cada JSON se convierte a texto y se segmenta en "chunks" para almacenarlo en una colección separada dentro de la base de datos.
4. **Almacenamiento de Metadatos**:
   * Se extraen y almacenan metadatos relevantes como nombres de reportes y elementos identificados en los JSON.
   * Estos metadatos se guardan en una colección específica dentro de la base de datos para aplicar nuevas funcionalidades.

**Pipeline RAG (Retrieval-Augmented Generation):**

1. **Preparación de Datos y Actualización**:
   * Se verifica la necesidad de actualizar los datos de los reportes (verificación de nuevas versiones o cambios).
   * Los datos se cargan desde los directorios /app/sample\_json\_reports y /app/sample\_text\_reports usando una clase DatabaseCreator.
   * Se mantienen dos colecciones en Qdrant: una para elementos tabulares y otra para fragmentos de texto.
2. **Configuración de Qdrant y Modelos de Embeddings**:
   * Se establece un cliente para Qdrant y se conecta a las colecciones de la base de datos.
   * Se define un modelo de lenguaje como ChatOpenAI para procesar consultas y extraer información.
3. **Configuración de los Retrievers**:
   * Se configuran SelfQueryRetriever para las colecciones de datos tabulares y de texto, usando metadatos específicos.
   * Se utilizan estrategias de búsqueda, como QdrantTranslator, para ajustar la relevancia de los documentos recuperados.
4. **Ensemble Retriever y Compresión**:
   * Los retrievers de datos tabulares y de texto se combinan usando EnsembleRetriever, con la posibilidad de ajustar el peso de cada uno.
   * Se aplica un método de reordenamiento (FlashrankRerank) para priorizar documentos relevantes y ContextualCompressionRetriever para optimizar el contenido recuperado.
5. **Cadena de QA con Retrieval y LLM**:
   * Se configura una cadena de preguntas y respuestas (QA) usando el LLM y el ensemble\_retriever.
   * El usuario puede realizar consultas, y el sistema utiliza los documentos más relevantes para construir una respuesta basada en la combinación de la consulta y la información recuperada.
6. **Generación de la Respuesta**:
   * El LLM genera respuestas basadas en la consulta del usuario y la información contextual proporcionada por los documentos de la base de datos.
   * Las respuestas incluyen referencias a los documentos fuente para asegurar la transparencia y la trazabilidad de la información.

El proceso de ETL de datos sigue dos vías:

1. **Vía JSON (Sin utilizar):** Usando Selenium y mitmproxy, se obtienen datos en formato JSON, pero esta vía de momento no se sigue en el flujo final.
2. **Vía PDF (continuamos con esta):**
   * Primero, el PDF se divide en imágenes usando la biblioteca pdf2image con convert\_from\_path.
   * Paralelamente, el contenido del PDF se convierte a texto utilizando PyPDFLoader de langchain\_community.document\_loaders.

Después de esto, se sigue un proceso en el que intervienen modelos de lenguaje:

1. **Transformación a JSON:**
   * Un LLM se utiliza para analizar cada imagen generada a partir de las páginas del PDF, convirtiéndolas en un archivo JSON con una estructura definida.
   * Posteriormente, este JSON se procesa nuevamente con un LLM, combinándolo con el texto extraído del PDF, para completar y corregir cualquier error.

El resultado final es un archivo JSON por cada página del PDF, en el que se detectan y describen todos los elementos importantes: **KPIs, gráficos (Charts), tablas** y una descripción detallada de cada uno de ellos.

1. **Almacenamiento en Base de Datos Vectorial:**
   * El JSON final se carga en una base de datos vectorial.
   * Además, se transforma el contenido del JSON a texto, que luego se segmenta en "chunks" para almacenarse en otra colección de la base de datos vectorial.
2. **Almacenamiento de Metadatos:**
   * Se extraen los nombres de los reportes y de los elementos presentes en el JSON.
   * Estos nombres se almacenan en una colección de metadatos dentro de la base de datos vectorial Qdrant.

. A partir de ellos, puedo sugerir un flujo de trabajo para la parte de RAG (Retrieval-Augmented Generation) que integre los elementos mencionados en el código, como los diferentes retrievers, el uso de Qdrant, y el proceso de compresión y reordenación de documentos. Aquí tienes un esquema generalizado del flujo:

1. **Carga y Actualización de Datos:**
   * Verifica si es necesario actualizar los datos (update\_data). Si es así, procesa y carga los datos desde los directorios /app/sample\_json\_reports y /app/sample\_text\_reports utilizando DatabaseCreator tanto para los elementos tabulares como los informes de texto.
   * Se utilizan dos colecciones en Qdrant: una para elementos tabulares (table\_elements) y otra para fragmentos de texto (text\_chunks).
2. **Configuración de Qdrant y Modelos de Embeddings:**
   * Configura un cliente de Qdrant y conecta las dos colecciones a través de QdrantVectorStore.
   * Define un modelo ChatOpenAI para realizar consultas y procesamiento de datos.
3. **Configuración de los Retrievers:**
   * Crea SelfQueryRetriever para ambos tipos de datos, tabular y de texto, utilizando metadatos específicos para cada tipo.
   * Define las estrategias de consulta con QdrantTranslator y ajusta parámetros como k para la búsqueda de documentos relevantes.
4. **Ensemble Retriever y Compresión:**
   * Combina los retrievers utilizando EnsembleRetriever, con un peso ajustable para cada tipo de retriever (tabular y texto).
   * Implementa FlashrankRerank para reordenar los documentos más relevantes, aplicando ContextualCompressionRetriever para cada retriever.
5. **Cadena de QA con Retrieval y LLM:**
   * Configura una cadena de preguntas y respuestas (RetrievalQA) usando el LLM (por ejemplo, gpt-4o-mini) y el ensemble\_retriever.
   * Integra la consulta del usuario y utiliza la cadena para obtener una respuesta basada en la combinación de la consulta y los documentos relevantes recuperados de Qdrant.
6. **Generación de la Respuesta:**
   * El usuario introduce una pregunta, y el sistema utiliza el LLM junto con los documentos recuperados para generar una respuesta.
   * La respuesta incluye los documentos fuente para proporcionar contexto adicional y transparencia.