The shared code implements a **Retrieval-Augmented Generation (RAG) system** using **Google Gemini** to answer user queries based on uploaded documents. It allows users to upload documents, index them for efficient retrieval, and ask questions to get relevant responses.

**1. Setting Up the Environment**

* The script imports necessary libraries like **Streamlit** (for UI), **FAISS** (for vector search), **SentenceTransformer** (for embeddings), **PyPDF2** and **docx** (for document parsing).
* The **Google Gemini API** is configured for AI-powered responses.

**2. Initializing FAISS for Document Retrieval**

* The **SentenceTransformer model ('all-MiniLM-L6-v2')** converts text into numerical vectors (embeddings). Embeddings is the process of converting text into numerical representation for machine/model to understand
* A **FAISS index** (index.faiss) is created or loaded if it already exists.
* Documents are stored in a list for reference.

**3. Processing Uploaded Documents**

* Users can upload **TXT, PDF, or DOCX** files.
* Depending on the file type, the content is extracted using relevant libraries.
* The document is then embedded and stored in the FAISS index for later retrieval.

**4. Querying the System**

* Users enter a query in the Streamlit interface.
* The system retrieves **top 3 most relevant documents** from the FAISS index.
* If relevant documents exist, they are passed along with the user’s query to the **Gemini model**.
* The model generates an answer based on the retrieved content and provides a structured response with:
  + **Bullet points**
  + **Headings**
  + **Visualizations (if applicable)**

**5. Streamlit UI Components**

* A **file uploader** to add documents.
* A **text input field** for user queries.
* A **button to fetch answers**, displaying results retrieved by FAISS and generated by Gemini.