### Part 1: Retrieval-Augmented Generation (RAG) Model for QA Bot

**Problem Statement:**

Develop a Retrieval-Augmented Generation (RAG) model for a Question Answering (QA)

bot for a business. Use a vector database like **Pinecone DB** and a generative model like

**Cohere API** (or any other available alternative). The QA bot should be able to retrieve

relevant information from a dataset and generate coherent answers.

**Task Requirements:**

1. Implement a **RAG-based model** that can handle questions related to a provided

document or dataset.

2. Use a vector database (such as **Pinecone**) to store and retrieve document

embeddings efficiently.

3. Test the model with several queries and show how well it retrieves and generates

accurate answers from the document.

Explanation : Reference SS\_RAG\_1.ipynb(for list type document\_segments)

SS\_RAG\_3.ipynb(for pdf file)

**Step-by-step approach:**

1. **Data Ingestion and Preprocessing:**
   * Load the dataset or documents.
   * Generate embeddings for the documents using a pre-trained model like Huggigface sentence-transformers/all-MiniLM-L6-v2 transformer models.
2. **Vector Database (Pinecone DB) Setup:**
   * Initialize and configure Pinecone, which will store and index the document embeddings.
   * Insert the document embeddings into the Pinecone index for efficient retrieval.
3. **Query Processing:**
   * When a user query comes in, encode it into an embedding using the same pre-trained model.
   * Use Pinecone to retrieve the most relevant documents by finding the closest embeddings to the query.
4. **Answer Generation:**
   * Use a generative model like Cohere API to generate coherent answers based on the retrieved documents.
5. **Test the Model:**
   * We will test the model by running several queries to see how well it retrieves relevant documents and generates coherent

This RAG-based model uses a vector database (Pinecone) for efficient retrieval of document embeddings and a generative model (Cohere) to generate coherent answers. It's capable of handling questions about business documents efficiently.

### Part 2: Interactive QA Bot Interface

**Problem Statement:**

Develop an interactive interface for the QA bot from Part 1, allowing users to input queries

and retrieve answers in real time. The interface should enable users to upload documents

and ask questions based on the content of the uploaded document.

**Task Requirements:**

1. Build a simple **frontend interface** using **Streamlit** or **Gradio**, allowing users to

upload PDF documents and ask questions.

2. Integrate the backend from Part 1 to process the PDF, store document embeddings,

and provide real-time answers to user queries.

3. Ensure that the system can handle multiple queries efficiently and provide accurate,

contextually relevant responses.

4. Allow users to see the retrieved document segments alongside the generated

answer.

To build an interactive Question-Answering (QA) bot interface with real-time querying based on uploaded documents, we'll use **Streamlit** for the frontend and integrate the backend developed in Part 1.

**Explanation : Reference SS\_RAG\_2.ipynb (Using streamlit for interaction)**

1. **Frontend (Streamlit) Setup**:
   * Users will upload a PDF document.
   * Users will input queries in a text box.
   * The interface will display the query, retrieved document segments, and the generated answer.
2. **Backend Integration**:
   * Convert the PDF into text.
   * Generate embeddings for the text and store them in Pinecone.
   * Use the query input from the user to retrieve the most relevant document segments.
   * Generate the answer using a generative model like Cohere or OpenAI's GPT.
3. **Real-time Interaction**:
   * Users can make multiple queries.
   * Display the relevant document segments alongside the generated answer.

**Requirements:**

* **Libraries**:
  + Streamlit for building the frontend interface.
  + PyPDF2 to extract text from PDF files.
  + Pinecone, Cohere, or OpenAI for embeddings and text generation.

### Challenges faced :

### Understand Pinecone Api, index – creation, initialization, deletion , vectors(query), matches(relevant documents), metadata

### Understand Huggingface pretrained sentence transformer used for tokenizer and embeddings

### Understand cohere API for generating Answer of query

1. Huggingface transformer(sentence-transformers/all-MiniLM-L6-v2)used is having dimensions – 384 and metric – cosine for queries and text generation. So pinecone index should match this
2. Also cohere command-lightly model is used for text generation
3. For relevant documents, tried to use pinecone metadata text but was not able to use so used from document - matching segments id with results matches id
4. Understand streamlit and npx localtunnel port 8501. Every time command run, unique URL is generated as eg :<https://short-symbols-marry.loca.lt>, <https://six-lemons-hug-loca/lt>, etc. by clicking it give ipv4 address as password and submit to see interactive streamlit APP.











