**1. Project Setup and Planning**

* **Define Objectives & Scope**
  + Clarify the main goal of your RAG system (e.g., Q&A, semantic search, content generation).
  + Decide on the domain (e.g., news, scientific papers, product reviews) based on the datasets available.
* **Set Deliverables**
  + Python code with clear documentation.
  + Detailed project report covering methodology, EDA findings, system architecture, evaluation, and conclusions.
  + A short presentation with visuals (charts, sample responses, etc.).
  + **Bonus:** Interactive demo (web interface/API).

**2. Dataset Selection**

* **Review Available Datasets**
  + Puerto Rico news articles
* **Selection Criteria**
  + **Relevance:** Does the dataset align with your intended application (e.g., Q&A, content summarization)?
  + **Quality & Size:** Is the data clean and substantial enough for retrieval tasks?
  + **Multimodal Bonus:** Consider datasets that include additional modalities (text + images/audio) if you wish to expand the scope.
* **Decision & Acquisition**
  + Choose one (or a combination) of the datasets.
  + Download and store the dataset(s) locally or on cloud storage for analysis.

**3. Exploratory Data Analysis (EDA)**

* **Initial Inspection**
  + Examine data structure, formats, and metadata.
  + Identify any missing values, duplicates, or inconsistencies.
* **Data Cleaning & Preprocessing**
  + Clean the data as necessary (e.g., removing noise, normalizing text).
  + If required, split long documents into smaller, coherent chunks.
* **Visualization & Reporting**
  + Use visualization libraries (e.g., Matplotlib, Seaborn) to generate charts showing data distributions, word frequencies, etc.
  + Document insights and challenges that might affect the retrieval and generation stages.

**4. Embedding and Storing Chunks**

**4.A Generate Document Embeddings**

* **Tool Selection**
  + Use [all-MiniLM-L6-v2](https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2) for English text.
  + **Bonus:** Experiment with OpenAI embeddings if budget allows.
* **Processing Steps**
  + Tokenize and chunk your documents.
  + Generate embeddings for each chunk using Sentence Transformers.
  + Validate that the embeddings capture semantic meaning by checking a few sample outputs.

**4.B Connect to a Vector Database**

* **Choosing the VectorDB**
  + Use [ChromaDB](https://www.trychroma.com/) for storing and retrieving embeddings.
  + **Bonus:** Consider cloud-based options like Azure AI Search or Weaviate.
* **Integration Steps**
  + **Pre-process:** Ensure each document chunk has an associated embedding.
  + **Store:** Upload the embeddings along with metadata (e.g., document IDs, original text) to ChromaDB.
  + **Retrieve:** Develop logic to perform similarity searches (e.g., cosine similarity) on stored embeddings given a query.

**4.C Integration Frameworks**

* **Optional Libraries**
  + Leverage frameworks like [LangChain](https://python.langchain.com/docs/integrations/vectorstores/chroma) or [LlamaIndex](https://gpt-index.readthedocs.io/en/latest/examples/vector_stores/ChromaIndexDemo.html) for easier integration.

**5. Connecting to a Large Language Model (LLM)**

* **LLM Selection & Setup**
  + Use the [OpenAI API](https://platform.openai.com/docs/api-reference/introduction) for generating responses.
  + Alternatively, set up a different LLM API if preferred.
* **Integration Workflow**
  + **Query Processing:** When a user submits a query, first use the vector DB to retrieve the most relevant document chunks.
  + **Prompt Construction:** Combine the retrieved documents with the query to create a context-rich prompt.
  + **Response Generation:** Pass the prompt to the LLM and retrieve the generated answer.
  + **Testing:** Iterate on the prompt structure and retrieval parameters to fine-tune the quality of responses.

**6. Evaluation**

* **Performance Testing**
  + **Manual Evaluation:** Test the system with multiple queries to understand its strengths and weaknesses.
  + **Automated Evaluation:**
    - Create a test set with queries and expected responses.
    - **Bonus:** Use an LLM as a judge to generate questions and evaluate the quality of the responses.
* **Metrics & Documentation**
  + Define success metrics (e.g., relevance, accuracy, response time).
  + Document your evaluation process and results in the project report.
  + Identify areas for improvement and possible refinements to the pipeline.

**7. Deployment (Bonus)**

* **Web Application / API Development**
  + **Backend:** Develop a RESTful API using Flask or FastAPI to handle requests and interact with your RAG system.
  + **Frontend:** Optionally, create a simple user interface using Streamlit for interactive demos.
* **Hosting**
  + Deploy the application on a cloud platform such as AWS, GCP, or Heroku.
  + Ensure proper handling of API keys and secure data transmission.
* **Demo Preparation**
  + Prepare an interactive demo to showcase during the project presentation.

**8. Documentation and Presentation**

* **Python Code Documentation**
  + Ensure the code is modular, well-commented, and follows best practices.
  + Include a README with setup instructions and usage examples.
* **Project Report**
  + Outline the project workflow from dataset selection to evaluation.
  + Include details of the EDA, system architecture, challenges, and performance metrics.
* **Presentation**
  + Develop slides that cover:
    - Introduction and project objectives.
    - Key steps and methodologies.
    - Visualizations from the EDA.
    - Examples of retrieval and generated responses.
    - Evaluation results and conclusions.
  + **Bonus:** Incorporate a live demo if possible.