## AI Copilot for Renewable Energy Data Rooms - Project Plan

This plan outlines the steps to build a prototype based on the provided challenge description.

### 1. Core Goal:

- Build an AI assistant that allows users to upload renewable energy project documents (PDFs, etc.) and ask natural language questions.
- The assistant must provide answers backed by specific references (quotes, page numbers) from the uploaded documents.

### 2. Key Features (MVP - Minimum Viable Product):

- **File Upload:** Allow users to upload multiple PDF documents.
- **Document Processing:**
  - Extract text content from PDFs.
  - Chunk the text into manageable segments.
  - Generate embeddings for each chunk.
- **Indexing:** Store chunks and their embeddings in a vector database (e.g., FAISS for simplicity initially).
- **Querying (RAG):**
  - Accept natural language questions from the user.
  - Embed the user's question.
  - Retrieve relevant document chunks from the vector database based on the question embedding.
  - Construct a prompt for an LLM, including the user's question and the retrieved chunks.
  - Call an LLM API to generate an answer based *only* on the provided context.
- **Answer Presentation:** Display the LLM's answer along with the source document name and page number for the chunks used.
- **User Interface:** Basic UI (using Streamlit) for file upload, question input, and displaying results.

### 3. Technology Stack Considerations (Based on Hints):

- **Document Parsing:** PyPDF2 or PyMuPDF (simpler) or PDFPlumber (more structured).
- **Embeddings:** Sentence Transformers (e.g., all-MiniLM-L6-v2 - runs locally) or an API like OpenAI's.
- **Vector Database:** FAISS (runs locally, good for prototypes) or cloud options like Pinecone/Weaviate.
- **LLM:** An API-based model (e.g., Google's Gemini API, OpenAI's GPT API) is

usually easiest for hackathons.

- **RAG Framework (Optional but Recommended):** LangChain or LlamaIndex can simplify the pipeline orchestration.
- **UI:** Streamlit or Gradio.

**4. Development Steps:**

- **Step 0: Setup:** Create a project environment (e.g., using venv or conda), install initial libraries.
- **Step 1: Document Loading & Chunking:** Implement functions to read PDFs, extract text, and split into chunks.
- **Step 2: Embedding & Indexing:** Implement functions to generate embeddings for chunks and store them in FAISS.
- **Step 3: RAG Core Logic:** Implement the retrieval (query FAISS) and generation (prepare prompt, call LLM API) steps. Ensure source tracking.
- **Step 4: Basic UI:** Build a simple Streamlit interface to tie everything together (upload, input, output).
- **Step 5: Testing & Refinement:** Test with sample documents and questions. Refine chunking, prompting, and retrieval strategies.

**5. Advanced Features (Post-MVP):**

- Support for other file types (Excel, Word).
- More sophisticated chunking and metadata extraction.
- Handling very large data rooms (scalability improvements).
- Checklist auto-population feature.
- Clickable links in references (if UI allows).

**Next Steps:**

- Decide on the specific libraries for parsing, embeddings, vector store, and LLM.
- Start implementing Step 1: Document Loading & Chunking.