Document QA App

1. What I Worked With

I built this project to extract useful information from a healthcare-related PDF file titled 'white_paper.pdf'. The document explores the idea of high-quality primary care. I also added support for DOCX and TXT files so the app could be more flexible and handle different formats.

2. How It Works

Here's what happens behind the scenes:

- I upload a document through the web app.
- The text is cleaned using simple rules to make it more readable.
- When I ask a question, a powerful language model (from HuggingFace) finds the best answer.
- The app shows the answer, confidence score, and the source excerpt from the document.

3. How Well It Performed

The model gave confidence scores ranging from 0.48 to 0.97. It worked well for clear, fact-based questions. It had trouble with vague questions or ones that required combining ideas from different parts of the text.

4. What Could Be Better

- Sometimes the answer was too short or incomplete for complex questions.
- Long documents could exceed the token limit of the model.
- Splitting and cleaning the text wasn't always perfect.
- If no document was uploaded or a library was missing, the app would break.

5. What I Learned

- Accurate question answering is definitely possible with the right tools.
- Simpler QA pipelines often outperform complex retrieval systems when the document is well-prepped.
- Streamlit made the app quick to build, but required careful handling of user uploads and errors.

6. Why It Took Time

- I ran into errors with FAISS loading, especially when trying to reload saved indexes.

- I tested multiple QA models before settling on one that gave high confidence results.

- Streamlit deployments failed due to missing Python modules (like PyPDF2 and python-docx).

- Adapting to different document formats took time, especially getting DOCX to read cleanly.

- Debugging deployment paths, cleaning old cells, testing new pipelines, and optimizing output all took effort-but it was worth it.