Problem statement:

Design and implement a prototype of an AI-powered summarization bot that can generate concise summaries of startup applications for investors' review.

LIVE BOT LINK: https://ai-startup-summarize-bot-bjh23blbykos4emwfxknfz.streamlit.app/
I have deployed the application using the streamlit cloud and Github, Please go through the above link to visit the live BOT

1. **Text extraction:**

```python
# converts the pdf file to list of paragraphs
def pdf_to_paragraphs_list(file):
    topics_list=[]
    pdf_reader = PyPDF2.PdfReader(file)
    num_pages = len(pdf_reader.pages)
    text = ""
    for page_num in range(num_pages):
        page = pdf_reader.pages[page_num]
        text = page.extract_text()
        topics_list.append(text)
    return " ".join(topics_list)
```

This function reads a PDF file, extracts the text content from each page, and returns that content as a single string with paragraphs separated by spaces. It's a useful utility function for processing PDF documents.

2. **Create api key and use it :**
   To utilize OpenAI's language models for text summarization and question-answering, you'll need to create an API key. This API key is used to authenticate and gain access to OpenAI's services. It's an essential step to connect to the OpenAI platform.

3. **openai for summarization:**

```python
# openai to generate summary
def get_startup_summary(context):
    large_language_model = OpenAI(temperature=0.9)  # model_name="text-davinci-003"
    text = '''  step 1: I have provided you a business data, first understand it clearly
                step 2: generate introduction, benificts, risks with less than 200 words
                step 3: Your answer should help investors
                content is :  '''+context
    summary_generated=large_language_model(text)
    return summary_generated
```

Python function named get_startup_summary. This function is responsible for generating a summary of startup-related content using an OpenAI language model. Here, it creates an instance of an OpenAI language model and assigns it to the

large_language_model variable. The temperature parameter is set to 0.9, which influences the randomness and creativity of the generated text. A higher value (e.g., 1.0) would result in more random text, while a lower value (e.g., 0.2) would make the text more deterministic and focused. Finally, the function returns the generated summary as a string

4. **Question and answers based on the content:**

```
def get_answers(context):
    large_language_model1 = OpenAI(temperature=0.9)  # model_name="text-davinci-003"
    text = '''  Answer only related to the above content the question is:'''+context
    answer_generated=large_language_model1(text)
    return answer_generated
```

Python function named get_answers. This function is responsible for generating answers to questions related to the provided context using an OpenAI language model. The function named get_answers takes a single argument, context. The context parameter is expected to be a string that represents the context or content to which questions will be related. Here, it creates an instance of another OpenAI language model and assigns it to the large_language_model1 variable. Similar to the previous code, the temperature parameter is set to 0.9, influencing the randomness and creativity of the generated text. This function takes a question context and a specific question, passes them to an OpenAI language model, and receives an answer related to the provided context and question. This function is particularly useful for providing responses to questions asked by users in the context of a startup application.

5. **Streamlit user interface**

Streamlit Columns:

I have used column to divide the screen into two halves, first one to display the summary and the second one for displaying queries and answers

PDF input component:

"PDF File Uploader," is used to allow users to upload PDF documents. This component is essential for the bot to extract text from the uploaded PDF startup applications. Users can upload PDF files directly through the user interface.

Text Input (Text Area):

A text input component, referred to as a "Text Input" is included to enable users to paste the content of a startup application. This input field is designed for situations where users want to input content manually rather than uploading a PDF. It provides a convenient way for users to interact with the bot.

Buttons:
Buttons are interactive elements that users can click to trigger specific actions. Buttons with labels such as "Generate Summary" and "Generate Answers." These buttons initiate the summarization and question-answering processes based on user input.

Markdown Text:
Markdown text is used to display formatted text within your Streamlit app. You've utilized markdown to create headings, subheadings, and centered text for clear and visually appealing user instructions. Markdown text provides a means to enhance the readability and structure of the interface.