**PDF Question Answering System**

**Project Overview:**

The PDF Question Answering System is a software application designed to assist users in extracting relevant answers from PDF documents based on their questions. It provides a user-friendly graphical user interface (GUI) that allows users to input their questions, load PDF files or folders, and retrieve the most relevant answers from the documents. It also allows user to search the keywords/questions on internet via Search on Wikipedia & Search on Google buttons.

**Functionality:**

* PDF Loading: The system allows users to load a single PDF file or an entire folder containing multiple PDF files. It utilizes the PyPDF2 library to extract text from the PDFs, making them searchable for answering questions.
* Text Pre-processing: The extracted text undergoes pre-processing to facilitate efficient question answering. Natural language processing techniques from the NLTK library are employed to tokenize the text into sentences and words. Lemmatization is also applied to reduce words to their base form for better matching.
* Question Answering: Users can input their questions into the system. The system matches the questions with the pre-processed sentences from the PDF documents based on word overlap. It ranks and retrieves the top relevant sentences that contain matching words, serving as the answers to the questions.
* Summarize\_PDF: The summarize\_pdf() function extracts text from a PDF file and utilizes the summarizer object to generate a summary of the text. The summarize\_pdf\_button\_click() function prompts the user to select a PDF file using a file dialog. If a file is selected, the summarize\_pdf() function is called to summarize the PDF content. The summarized text is then displayed in the answer\_text widget. If an exception occurs during the summarization process, an error message can be displayed using a message box. You can customize the error message format or handle specific types of exceptions according to your requirements.
* Error Handling: It is recommended to add error handling for cases where the selected file is not a PDF or if there is an error during the summarization process. You can use try-except blocks to catch any exceptions and display appropriate error messages to the user.
* GUI: The system provides a user-friendly GUI implemented using the tkinter library. Users can interact with the system through various GUI components, including buttons, entry fields, and text boxes. The GUI facilitates easy file selection, question input, answer display, and result clearing.
* Additional Modules: Make sure to import the necessary modules, such as messagebox, to handle any additional exceptions specific to your use case.

**Usage:**

1. Users can select a single PDF file or an entire folder containing multiple PDF files.
2. Users input their questions into the system through the GUI.
3. The system processes the questions and matches them with the relevant sentences from the pre-processed PDF documents.
4. The top relevant sentences are displayed as answers to the questions in the GUI.
5. Users have the option to clear the displayed answers and input a new question for processing.

The PDF Question Answering System provides a convenient and efficient way for users to extract answers from PDF documents based on their questions. It can be used for various purposes, such as research, information retrieval, and document analysis.

**Installation and Dependencies:**

To set up and run the PDF Question Answering System, you need to install the required dependencies and follow the installation steps below:

\*\*Dependencies:\*\*

1. PyPDF2: A library for extracting text from PDF files.

2. NLTK (Natural Language Toolkit): A library for natural language processing tasks.

\*\*Installation Steps:\*\*

1. Install Python: Make sure you have Python installed on your system. The PDF Question Answering System is compatible with Python 3.

2. Install PyPDF2:

- Open a command prompt or terminal.

- Run the following command to install PyPDF2 using pip:

```

pip install PyPDF2

```

3. Install NLTK:

- Open a command prompt or terminal.

- Run the following command to install NLTK using pip:

```

pip install nltk

```

4. Download NLTK Data:

- After installing NLTK, you need to download additional data required for certain NLTK functionalities.

- Open a Python interactive shell or a Python script and run the following commands:

```

import nltk

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('averaged\_perceptron\_tagger')

```

This will download the necessary data for tokenization, lemmatization, and part-of-speech tagging.

5. Add Logo Image:

- Place your logo image file in the appropriate directory.

- Update the `watermark\_image` variable in the code to specify the correct path to your logo image file.

6. Run the Application:

- Save the provided code in a Python file (e.g., `pdf\_question\_answering.py`).

- Run the Python file using the following command:

```

python pdf\_question\_answering.py

```

- The GUI window for the PDF Question Answering System will open.

Note : Make sure you have the necessary permissions to access and read the PDF files or folders you intend to use with the system.

By following these steps, you will be able to install the required dependencies and run the PDF Question Answering System on your system.

**Code Explanation:**

**Importing Required Libraries:**

The code begins by importing necessary libraries and modules. os is used for interacting with the operating system. PyPDF2 is used for reading and extracting text from PDF files.

nltk is the Natural Language Toolkit library, which is used for various natural language processing tasks. sent\_tokenize and word\_tokenize are functions from NLTK used for tokenizing text into sentences and words, respectively. pos\_tag is a function from NLTK used for part-of-speech tagging. WordNetLemmatizer is a class from NLTK used for lemmatization.

PunktSentenceTokenizer is a class from NLTK used for sentence tokenization. string is a module providing common string operations. Tk, Label, Entry, Button, Text, Scrollbar, END, filedialog, Style, PhotoImage, and Font are classes and functions from the tkinter library used for creating a graphical user interface (GUI) for the application.

**PDF Loading Function:**

The load\_pdf function takes a file\_path as input and returns the extracted text from the PDF file. It uses the open function to open the PDF file in read-binary mode ('rb'). It creates a PdfReader object from PyPDF2 to read the PDF file. It initializes an empty string text to store the extracted text. It iterates through each page in the PDF using a for loop and extracts the text from each page using the extract\_text() method. The extracted text from all pages is concatenated and stored in the text variable. Finally, it returns the extracted text.

**Text Pre-processing Function:**

The preprocess\_text function takes the extracted text from the PDF as input and returns pre-processed sentences. It uses the PunktSentenceTokenizer class from NLTK to tokenize the text into sentences. The sentences are stored in the sentences variable. It uses the WordNetLemmatizer class from NLTK for lemmatization. It iterates through each sentence and tokenizes it into words using the word\_tokenize function from NLTK. The words are lemmatized, converted to lowercase, and filtered to remove non-alphanumeric characters and empty spaces. The pre-processed sentences are stored in the preprocessed\_sentences list. Finally, it returns the list of pre-processed sentences.

**Answering Questions Function:**

The answer\_questions function takes pre-processed sentences, a question string, and an optional parameter num\_sentences (default is 3) as input and returns the top relevant sentences as answers. It tokenizes the question into words, lemmatizes them, and stores them in question\_tokens. It initializes an empty list sentences to store sentences that have overlapping words with the question. It iterates through each pre-processed sentence and calculates the overlap between the sentence and the question by finding the common words. Sentences with a non-zero overlap are added to the sentences list along with the overlap count. The sentences list is sorted in descending order based on the overlap count. The top num\_sentences sentences are extracted from the sorted list and stored in the top\_sentences list. Finally, it returns the top\_sentences.

**Processing Question Function:**

The process\_question function is called when the "Process Question" button is clicked in the GUI. It retrieves the question entered in the GUI by accessing the question\_entry widget.

It calls the answer\_questions function with the preprocessed\_sentences and the entered question as arguments to get the relevant sentences. If no relevant sentences are found, it displays a message in the answer\_text widget indicating that no answer was found. If relevant sentences are found, it displays the question and the top sentences in the answer\_text widget. The question is displayed in blue color using a tag named "question", and the answers are displayed in green color using a tag named "answer". Finally, it enables the answer\_text widget for viewing, inserts the formatted question and answers, and disables the widget again.

**Loading PDF File Function:**

The load\_files function is called when the "SELECT PDF" button is clicked in the GUI.

It opens a file dialog to select a PDF file. If a file is selected, it tries to load the PDF file using the load\_pdf function and pre-processes the extracted text using the preprocess\_text function. The pre-processed sentences are stored in the preprocessed\_sentences variable as a global variable. If any error occurs during loading or processing the PDF file, it displays an error message in the status\_label widget. If no file is selected, it displays a message indicating that no file was selected.

**Open Web Page Function:**

This function opens a web page in the default web browser.

It takes a URL parameter, which is the URL of the web page to be opened.

The function uses the sub process. Popen method to execute the browser executable (chrome.exe in this case) and pass the URL as an argument.

**Search Google Function:**

This function performs a Google search for the user's question.

It retrieves the question from the question entry widget (presumably a text entry widget).

If no question is provided, it displays a message in the answer\_text widget to prompt the user to enter a question or keyword.

If a question is provided, it uses the search() function (not shown in the code) to perform the Google search and retrieve the top search results.

The function then updates the answer\_text widget to display the top search results as clickable links.

Each link is configured with a blue color and an underline, and clicking on a link opens the corresponding web page using the open\_web\_page() function.

**Summarize PDF Function:**

This function summarizes the content of a PDF document.

It takes two parameters: file\_path (the path to the PDF file) and max\_sentences (the maximum number of sentences for the summary).

The function uses the pdfplumber library to extract the text content from the PDF file.

It then creates a PlaintextParser object to parse the extracted text using the English tokenizer.

Next, it initializes an LsaSummarizer object to perform the summarization.

The function summarizes the text by selecting the top max\_sentences sentences and joins them into a single string.

Finally, it returns the summarized text.

Please note that the provided code assumes the presence of certain libraries and widgets (subprocess, search(), pdfplumber, PlaintextParser, Tokenizer, LsaSummarizer, pdfplumber, filedialog, simpledialog, END, Text, etc.), which are not included in the code snippet you provided. Make sure to import the required libraries and define the necessary widgets before using these functions.

**Creating GUI:**

The code creates a tkinter window for the GUI with the title "PDF Question Answering" and a size of 600x600 pixels.

The window background is set to white.

The GUI components like labels, buttons, and text widgets are not shown in the code snippet but would be present in the actual code.

The code ends with the window.mainloop() function, which starts the event loop and keeps the GUI window open until it is closed by the user.

**Usage Instructions:**

To use the PDF Question Answering project, follow these steps:

1. Ensure you have Python installed on your system. You can download and install Python from the official Python website (https://www.python.org) if it's not already installed.

2. Install the necessary dependencies. The project requires the following dependencies to be installed:

- PyPDF2: This library is used for extracting text from PDF files.

- NLTK (Natural Language Toolkit): This library is used for natural language processing tasks such as tokenization and lemmatization.

You can install these dependencies using the following command in your command prompt or terminal:

```

pip install PyPDF2 nltk

```

3. Import the required libraries and functions:

* import os
* import PyPDF2
* import nltk
* from nltk import sent\_tokenize, word\_tokenize, pos\_tag
* from nltk.stem import WordNetLemmatizer
* from nltk.tokenize import PunktSentenceTokenizer
* import string
* import wikipedia
* import subprocess
* from tkinter import Tk, Label, Entry, Button, Text, Scrollbar, END, filedialog, Frame, PhotoImage
* from tkinter.ttk import Style
* from tkinter.font import Font
* import webbrowser
* from googlesearch import search
* import requests
* from bs4 import BeautifulSoup
* import pyttsx3
* import threading
* import pdfplumber
* from sumy.parsers.plaintext import PlaintextParser
* from sumy.nlp.tokenizers import Tokenizer
* from sumy.summarizers.lsa import LsaSummarizer

4. Define the functions and methods required for the project. These include:

- `load\_pdf(file\_path)`: Function to load a PDF file and extract its text content.

- `preprocess\_text(text)`: Function to preprocess the extracted text by tokenizing and lemmatizing the sentences.

- `answer\_questions(preprocessed\_sentences, question, num\_sentences)`: Function to find relevant sentences that answer a given question based on their overlap with the question tokens.

- `process\_question()`: Function to process the entered question and display relevant answers in the GUI.

- `load\_files()`: Function to load a PDF file using a file dialog and extract its text content.

- `load\_folder()`: Function to load multiple PDF files from a selected folder and extract their text content.

- `clear\_text()`: Function to clear the answer text in the GUI.

- `clear\_status()`: Function to clear the status label in the GUI.

5. Create the GUI window using `Tk()` and configure its properties such as title, size, and background color.

6. Add the necessary GUI components such as labels, buttons, entry fields, and text widgets for file selection, question input, answer display, and status.

7. Configure event handlers for the buttons to call the corresponding functions when clicked.

8. Start the GUI event loop using `window.mainloop()` to display the GUI and handle user interactions.

9. Run the Python script that contains the project code.

10. The GUI window will appear. You can select a PDF file using the "SELECT PDF" button or select a folder containing multiple PDF files using the "SELECT Folder" button.

11. Enter your question in the provided input field.

12. Click the "Process Question" button to process the question and display the relevant answers in the answer text area.

13. If no relevant answer is found, a message will be displayed indicating that no answer was found.

14. You can click the "Clear Answer" button to clear the answer text area.

15. You can click the "Clear Status" button to clear any status messages displayed in the status label.

16. Close the GUI window when you're done using the project.

Note: Make sure to replace the file paths or modify the code as per your specific requirements, such as changing the logo image path or adjusting the GUI layout.  
  
 **Short Summary:**

**Importing Libraries:**

**os:** Provides functions for interacting with the operating system.

**PyPDF2:** Allows manipulation of PDF files.

**nltk:** Natural Language Toolkit library for text processing and analysis.

**string:** Provides various string-related functions and constants.

**wikipedia:** Allows searching and accessing Wikipedia content.

**subprocess:** Enables running system commands from within the Python script.

**tkinter:** Provides a graphical user interface (GUI) for the script.

**webbrowser:** Allows opening web pages in a web browser.

**googlesearch:** Provides functions for performing Google searches.

**requests:** Library for making HTTP requests.

**bs4:** Beautiful Soup library for web scraping.

**pyttsx3:** Enables text-to-speech functionality.

**threading:** Allows concurrent execution of multiple threads.

**pdfplumber:** Library for extracting text from PDF files.

**sumy:** Library for text summarization.

**Initializing the Text-to-Speech Engine:**

**pyttsx3.init():** Initializes the text-to-speech engine.

**Functions:**

**load\_pdf(file\_path):** Loads a PDF file and extracts the text content.

**preprocess\_text(text):** Tokenizes sentences and words, and performs lemmatization on the text.

**answer\_questions(preprocessed\_sentences, question, num\_sentences=3**): Processes a question and returns the top sentences containing relevant answers.

**open\_wikipedia\_page(url):** Opens a Wikipedia page in a web browser.

**search\_and\_retrieve\_summary(query):** Performs a web search and retrieves a summary from a specific website.

**search\_wikipedia():** Processes a question, searches Wikipedia, and displays the relevant answer.

**open\_web\_page(url):** Opens a web page in a web browser.

**search\_google():** Performs a Google search and displays the top search results.

**summarize\_pdf(file\_path, max\_sentences):** Summarizes a PDF file by extracting the specified number of sentences.

**process\_question():** Processes a question, searches for answers in the loaded PDF, and displays the relevant sentences.

**load\_files():** Opens a file dialog to select a PDF file for processing.

**load\_folder():** Opens a folder dialog to select a folder containing multiple PDF files for processing.

**clear\_text():** Clears the answer text box.

**clear\_status():** Clears the status label.

**convert\_to\_speech():** Converts the answer text to speech using the text-to-speech engine.

**speak(answer):** Uses the text-to-speech engine to speak the provided answer.

**Graphical User Interface (GUI):**

The script creates a GUI window using tkinter library, with various buttons, entry fields, labels, and text boxes.

Users can select a PDF file or a folder containing PDF files, enter a question or keyword, and perform different actions such as searching Wikipedia, searching the loaded PDF, searching Google, summarizing the PDF, etc.

The answer to the question or the search results are displayed in the text box.

There are buttons for clearing the text, clearing the status, and converting the answer to speech.

**Examples:**

**Example 1:**

Question: "What is the capital of France?"

Expected Output: The system will search for relevant sentences that contain words related to the question, such as "capital," "France," and "city." It will display the top sentences that have the highest overlap with the question tokens. The expected output could be:

```

Question: What is the capital of France?

Top Answers:

- The capital of France is Paris.

- Paris is the capital city of France.

```

**Example 2:**

Question: "Who wrote the novel 'Pride and Prejudice'?"

Expected Output: The system will search for relevant sentences that contain words related to the question, such as "wrote," "novel," "Pride and Prejudice," and "author." The expected output could be:

``` Question: Who wrote the novel 'Pride and Prejudice'?

Top Answers:

- Jane Austen is the author of the novel 'Pride and Prejudice.'

- 'Pride and Prejudice' was written by Jane Austen.

```

**Example 3:**

Question: "What is the temperature range for baking a cake?"

Expected Output: If there are no sentences in the pre-processed text that have a significant overlap with the question tokens, the system will display a message indicating that no relevant answer was found. The expected output could be:

```

Question: What is the temperature range for baking a cake?

No relevant answer found.

```

**Example 4:**

Question: "How can I improve my grammar skills?"

Expected Output: If there are multiple sentences with overlaps, the system will display the sentences in descending order of relevance. The expected output could be:

```

Question: How can I improve my grammar skills?

Top Answers:

- To improve your grammar skills, you can practice writing and reading extensively.

- Taking a grammar course or using online grammar resources can help you improve your grammar skills.

- One effective way to improve your grammar skills is to engage in conversations with native English speakers.

```

**Error Handling:**

- If an error occurs while loading the PDF file or processing its content, an error message will be displayed in the `status\_label` widget in the GUI, indicating that an error occurred.

- If no PDF file is selected or no relevant answer is found for a question, appropriate messages will be displayed to inform the user.

- Users may encounter errors if the selected file is not a valid PDF file or if the PDF file is password-protected or encrypted. In such cases, the system may fail to load the file or extract its text content. The error message displayed in the `status\_label` widget will provide information about the specific error.

- To troubleshoot these errors, users can ensure that they have selected a valid PDF file and that they have the necessary permissions to access the file. They can also check if the PDF file is password-protected and provide the correct password if prompted.

- If users encounter any other issues or errors, they can refer to the error messages displayed or seek assistance from the project developers or support channels.

**Conclusion:**

The PDF Question Answering system is a useful tool for extracting information from PDF documents and finding relevant answers to user questions. Its user-friendly graphical interface simplifies the process of selecting PDF files or folders, entering questions, and receiving answers based on the PDF content.

Utilizing the PyPDF2 library, the system efficiently extracts text from PDF files, making it searchable for question answering. The integration of the NLTK library enables effective natural language processing tasks such as tokenization and lemmatization. By pre-processing the extracted text and comparing it with the user's question, the system identifies relevant sentences that serve as potential answers.

The system's intuitive interface and intelligent processing capabilities make it a valuable resource for researchers, students, and professionals who frequently work with large amounts of textual information in PDF format. It provides a convenient way to quickly locate information within PDF documents, saving time and effort.

We value user feedback and encourage users to share their experience with the system. Any feedback, including suggestions or issues encountered, will be greatly appreciated as it helps us improve the system's functionality and enhance the overall user experience.

Thank you for using the PDF Question Answering system. We hope it proves to be a valuable tool in your information retrieval tasks.

**Appendix:**

PyPDF2 library: Documentation and resources can be found at https://pythonhosted.org/PyPDF2/

NLTK library: Documentation and resources can be found at https://www.nltk.org/

ChatPDF by OpenAI: For a more advanced and interactive question-answering experience, you can explore the ChatPDF model developed by OpenAI. It is designed for generating human-like responses to a wide range of queries, including PDF-related questions. You can find more information at <https://www.chatpdf.com/> Remember to refer to the respective documentation and resources for further details on the libraries and models used in the PDF Question Answering system.

**Acknowledgments:**

I would like to express my sincere gratitude to HPCL (Hindustan Petroleum Corporation Limited) for providing me with the opportunity to work on this project. The support and guidance from the organization have been instrumental in the successful completion of this PDF Question Answering system.

I would like to extend my heartfelt appreciation to my mentor, Sir Raja Prasad – Assistant Manager – IS Dept. for his invaluable guidance, expertise, and encouragement throughout the project. His deep knowledge and insights have been instrumental in shaping the development of the system.

I would also like to thank the developers and contributors of the PyPDF2 and NLTK libraries. These open-source libraries provided the essential tools and functionalities for handling PDF files and performing natural language processing tasks in this project.

Additionally, I would like to acknowledge the efforts of the entire team at HPCL who have provided valuable inputs and feedback during the development process.

Lastly, I would like to express my appreciation to all the individuals who have contributed to the project by testing, providing feedback, and offering suggestions for improvement.

This project would not have been possible without the support, guidance, and contributions of all the individuals and organizations mentioned above. Thank you for your invaluable support and assistance throughout this project.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*