SHREE_KESHAVAN_NLP ASSIGNMENT DOCUMENTATION

Tittle: Creation of Objective Questions with Multiple Correct Answers

1. Project Introduction

Introduction:

This project is a Natural Language Processing (NLP) based application that generates Multiple-Choice Questions (MCQs) from given text data. The purpose of this project is to automate the process of This solution generates objective questions with multiple correct answers, adding complexity and depth to the assessment. The primary objective is to encourage critical thinking and perspective exploration in readers while assisting educators in creating engaging and challenging assessments for students. The provided code serves as the backbone of the project, handling all functionalities from text extraction to question generation.

Key Features:

- Uses the spaCy library, a powerful tool for advanced NLP, to parse and manipulate text data.
- Uses PyPDF2, a Python library, for extracting text from PDF files.
- Employs 'en_core_web_sm', an English language model in spaCy, for NLP tasks.
- Generates a variety of MCQs, with potential for multiple correct answers, from any given context.

2. Project Workflow

Data Collection:

Data for this project comes from PDF documents. These documents (pdf_file1, pdf_file2, pdf_file3 in the provided code) are read and processed using PyPDF2, a Python library capable of splitting, merging, and transforming PDF pages.

```
"""Importing Libraries"""

import spacy
import random
from PyPDF2 import PdfReader

# To supress warnings
import warnings
warnings.filterwarnings("ignore")
```

NLP Processing:

The extracted text undergoes NLP processing using the spaCy library. The 'en_core_web_sm' model, a small English model trained on web text, is used to analyze the extracted text.

```
"""Loading English Language Model"""

# Load English language model
nlp_model = spacy.load("en_core_web_sm")
```

Question Generation:

The program generates MCQs by identifying sentences in the text, selecting a word to be blanked out, and creating a question out of this. It then provides a set of answer options, which include the multiple correct answer and other randomly selected words from the text.

3. Function Details

get_mca_questions Function:

The get_mca_questions function is the main function that generates MCQs. It takes in the context (the text data) and the number of questions to be generated as input. The function first analyzes the text using the spaCy model create_mcq_with_multiple_correct. It then creates MCQs using the nested function generate_variety_question. The MCQs and their correct answers are stored in a dictionary and returned.

```
"""Multiple Choice Question Generating Function"""

# Define a function to generate multiple-choice questions based on the provided text context and the number of questions desired.
def get_mca_questions(text_context: str, num_of_questions: int):
    # Process the text context using the NLP model and store it in the 'doc' variable.
    doc = nlp_model(text_context)
```

generate_variety_question Function:

The generate_variety_question function is a helper function that generates a single MCQ. It selects a random sentence from the text, chooses a non-punctuated word from the sentence to be blanked out, and generates a question. The function also generates the answer options, which include the correct answer and other randomly selected words from the text. The function ensures that there is at least one correct answer and up to three additional options.

```
# Define a function to generate a variety of questions.
def generate_variety_question():
    # Select a random sentence from the processed document.
    random_sentence = random.choice(list(doc.sents))
    # Choose a random word within the sentence that is not a punctuation mark.
    random_word = random.choice([token for token in random_sentence if not token.is_punct])

    # Replace the selected word with "_____" to create a blank in the question text.
    question_text = random_sentence.text.replace(random_word.text, "_____")
    correct_answers = [random_word.text]  # List of correct answer options.

    # Create a list of other word options for the question.
    other_options = [token.text for token in doc if token.is_alpha and token.text != correct_answers[0]]
    num_correct_options = random.randint(1, 2)  # Generate 1 or 2 correct options.
    correct_answers.extend(random.sample(other_options, num_correct_options))

    num_other_options = min(4 - num_correct_options, len(other_options))
    # Randomly select additional word options to complete the multiple-choice options.
    other_options = random.sample(other_options, num_other_options)

    # Generate the multiple-choice question with correct options using the function defined earlier.
    mcq = create_mcq_with_multiple_correct(question_text, correct_answers, other_options)
    return mcq
```

Data Extraction Function:

- The `extract_text_from_pdfs` function effectively extracts text from PDF documents and consolidates it into a single text string.
- It utilizes PyPDF2 to parse PDF files, collecting text from each page and returning a cohesive text corpus.
- This function ensures accurate text retrieval from PDF sources, facilitating NLP analysis.

```
"""Data Extraction Function"""

# Define a function to extract text from PDF documents and concatenate it into a single text string.
def extract_text_from_pdfs(pdf_documents):
    extracted_text = ""
    for pdf_file in pdf_documents:
        # Initialize the PDF reader to extract text from the current PDF document.
        pdf_reader = PdfReader(pdf_file)
        for pdf_page in pdf_reader.pages:
            # Concatenate the text from each page to the 'extracted_text' string.
            extracted_text += pdf_page.extract_text()
    return extracted_text
```

4. Usage Example

Code Execution:

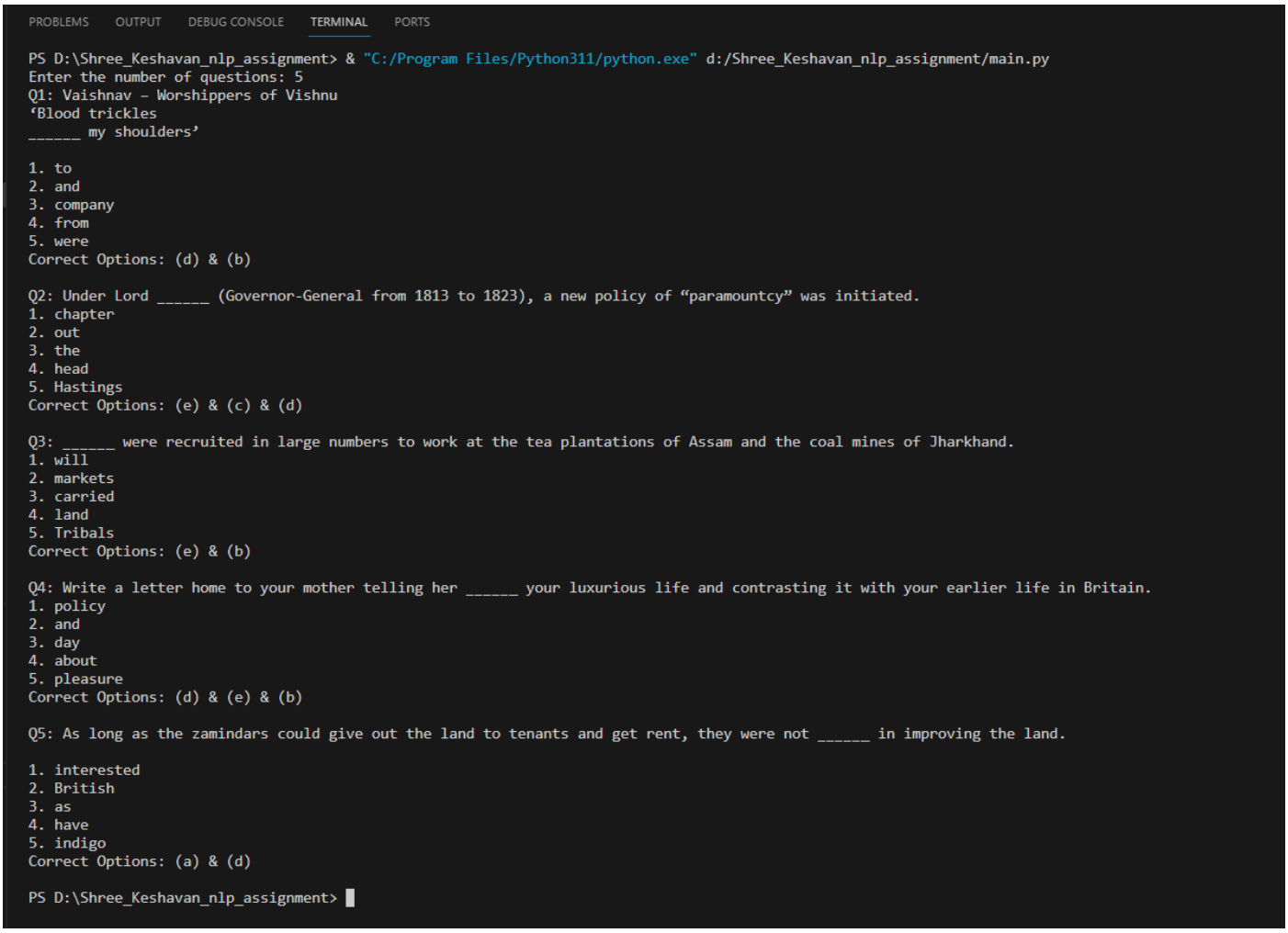
1. To execute the provided code, follow these steps:
2. Start by importing the necessary libraries (spacy, random, PyPDF2).
3. Load the English language model from spaCy.
4. Define the main functions: get_mca_questions, create_mcq_with_multiple_correct, and generate_variety_question.
5. Define the PDF documents from which you want to extract text.
6. Use the extract_text_from_pdfs function to extract text from these documents and store it in a variable.
7. Request the user to input the number of questions they want to generate.
8. Use the get_mca_questions function to generate the specified number of questions.
9. Print the generated questions.

Here is an example of how you might execute the code:

```
# Define the PDF documents to extract text from
pdf_file1 = 'chapter-2.pdf'
pdf_file2 = 'chapter-3.pdf'
pdf_file3 = 'chapter-4.pdf'

pdf_documents = [pdf_file1, pdf_file2, pdf_file3]
# Extract text from the specified PDFs and store it in 'document_text'
document_text = extract_text_from_pdfs(pdf_documents)
text_context = document_text
```

5. Results and Future Enhancements



Generated Questions:

Q1: Vaishnav – Worshippers of Vishnu 'Blood trickles _____ my shoulders'

1. to
2. and
3. company
4. from
5. were

Correct Options: (d) & (b)

Q2: Under Lord _____ (Governor-General from 1813 to 1823), a new policy of "paramountcy" was initiated.

1. chapter
2. out
3. the
4. head
5. Hastings

Correct Options: (e) & (c) & (d)

Future Enhancements (if I have more time):

While the current implementation is effective, there are several potential improvements and extensions that could be made:

Improved NLP: The system could use more advanced NLP techniques like Semantic Search, OpenAI models, VectorDB to ensure the generated questions are grammatically correct and contextually relevant.

- **Example Architecture**: Load Data --> Extract & Split into chunks --> Embed chunks --> Vector DB --> Semantic search --> Chain - PROMPT (prompt about generating MCA and templates) --> User input --> Retrieving from Vector DB --> Display MCA questions and with multiple similar answers

Customizability: Users could be given more options to customize the generated questions, such as difficulty level, question type, or topic using **Hugging face English models**.

More Document Types: The system could be extended to support text extraction from other types of documents, such as Word files or web pages.

Deployment on Streamlit and Hugging Face: In the future, I plan to deploy the project on Streamlit for user-friendly access and make it available live on Hugging Face, fostering collaboration and wider usage.