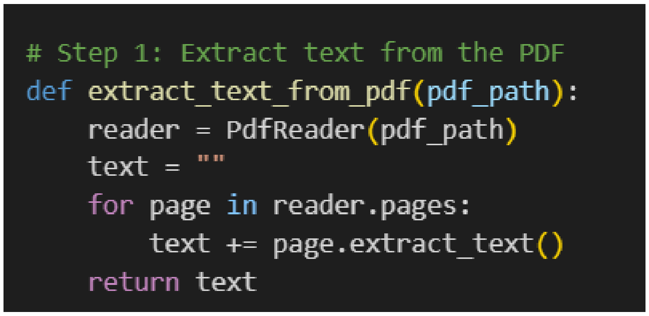
Malhar Sham Ghogare

ghogare.m@northeastern.edu

Shakespeare QA Bot Using Hugging Face Transformers

1. Data Processing

Code:



```
# Step 1: Extract text from the PDF
def extract_text_from_pdf(pdf_path):
    reader = PdfReader(pdf_path)
    text = ""
    for page in reader.pages:
        text += page.extract_text()
    return text
```

Purpose:

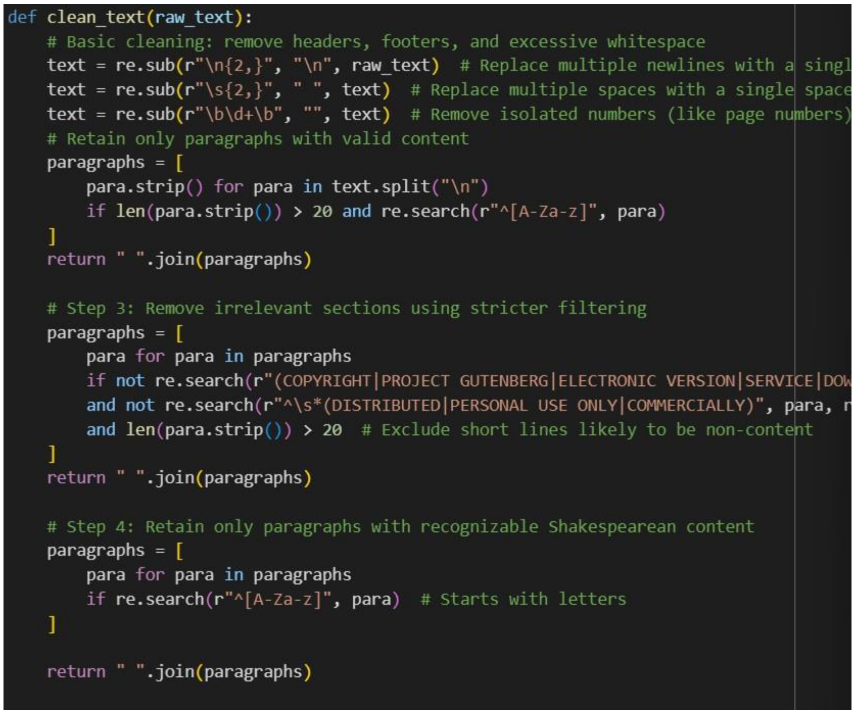
- This function extracts raw text from a PDF file.
- It reads each page of the PDF using the PdfReader from the PyPDF2 library and concatenates the text into a single string.

Usage:

- Provide the path to the PDF file as an argument to this function.

1.2. Cleaning Extracted Text

Code:



```
def clean_text(raw_text):
    # Basic cleaning: remove headers, footers, and excessive whitespace
    text = re.sub(r"\n{2,}", "\n", raw_text)  # Replace multiple newlines with a singl
    text = re.sub(r"\s{2,}", " ", text)  # Replace multiple spaces with a single space
    text = re.sub(r"\b\d+\b", "", text)  # Remove isolated numbers (like page numbers)
    # Retain only paragraphs with valid content
    paragraphs = [
        para.strip() for para in text.split("\n")
        if len(para.strip()) > 20 and re.search(r"^[A-Za-z]", para)
    ]
    return " ".join(paragraphs)

    # Step 3: Remove irrelevant sections using stricter filtering
    paragraphs = [
        para for para in paragraphs
        if not re.search(r"(COPYRIGHT|PROJECT GUTENBERG|ELECTRONIC VERSION|SERVICE|DOW
        and not re.search(r"^\s*(DISTRIBUTED|PERSONAL USE ONLY|COMMERCIALLY)", para, r
        and len(para.strip()) > 20  # Exclude short lines likely to be non-content
    ]
    return " ".join(paragraphs)

    # Step 4: Retain only paragraphs with recognizable Shakespearean content
    paragraphs = [
        para for para in paragraphs
        if re.search(r"^[A-Za-z]", para)  # Starts with letters
    ]

    return " ".join(paragraphs)
```

Output:

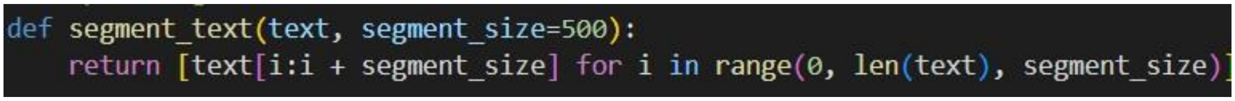
- Returns cleaned text as a single string, ready for further processing.

Usage:

- Pass the raw_text (output from extract_text_from_pdf) to this function.

1.3. Segmenting the Text

Code:



```python
def segment_text(text, segment_size=500):
    return [text[i:i + segment_size] for i in range(0, len(text), segment_size)]
```

Purpose:

- Divides the cleaned text into smaller, fixed-sized chunks for efficient processing.
- The default segment size is 500 characters.

Output:

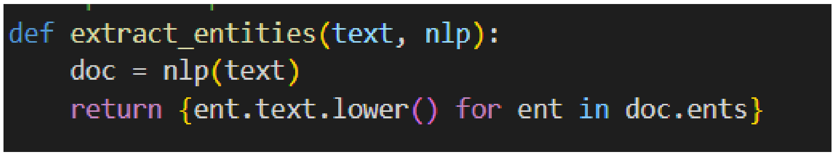
- A list of text segments.

Usage:

- Pass the cleaned_text to this function.

1.4. Extracting Named Entities

Code:



```python
def extract_entities(text, nlp):
    doc = nlp(text)
    return {ent.text.lower() for ent in doc.ents}
```

Purpose:

- Uses spaCy's Named Entity Recognition (NER) to identify and extract named entities (e.g., people, places, dates) from a text segment.

Output:

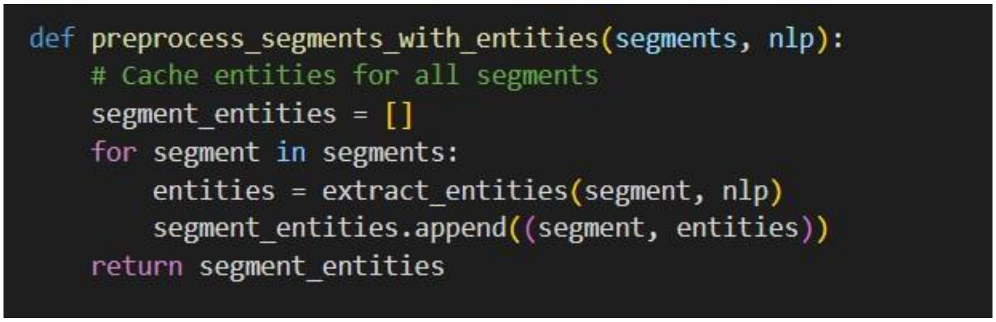
- A set of named entities extracted from the text.

Usage:

- Load spaCy's English model and pass it along with a text segment to this function.

1.5. Preprocessing Segments with Named Entities

Code:



```
def preprocess_segments_with_entities(segments, nlp):
    # Cache entities for all segments
    segment_entities = []
    for segment in segments:
        entities = extract_entities(segment, nlp)
        segment_entities.append((segment, entities))
    return segment_entities
```

Purpose:

- Extracts and caches named entities for each text segment.
- Stores the segment and its corresponding entities as a tuple for later use.

Output:

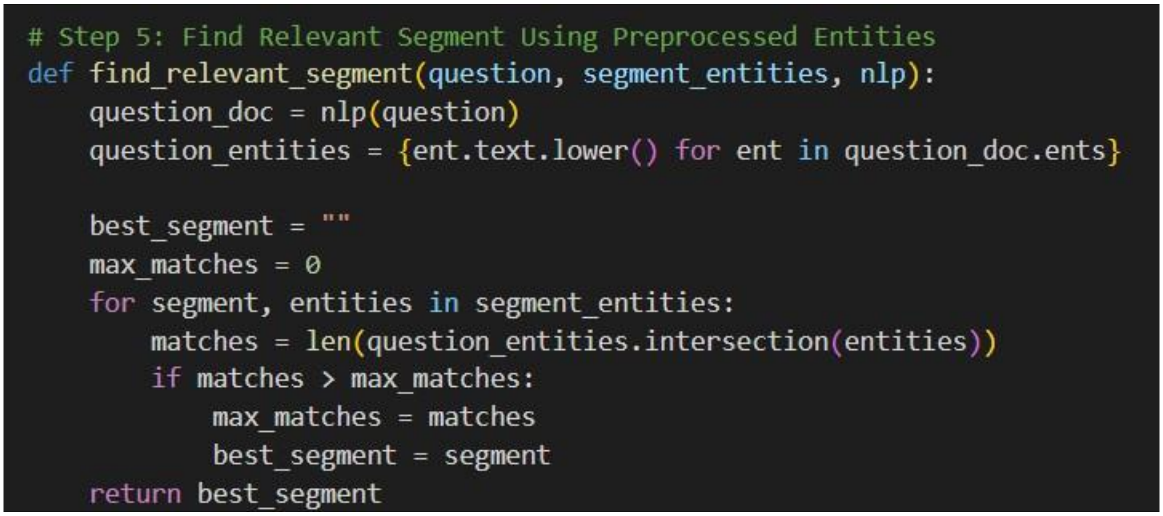
- A list of tuples, where each tuple contains:
    1. A text segment.
    2. A set of named entities extracted from the segment.

Usage:

- Pass the segmented text and spaCy model to this function.

1.6. Finding Relevant Segments

Code:



```python
# Step 5: Find Relevant Segment Using Preprocessed Entities
def find_relevant_segment(question, segment_entities, nlp):
    question_doc = nlp(question)
    question_entities = {ent.text.lower() for ent in question_doc.ents}

    best_segment = ""
    max_matches = 0
    for segment, entities in segment_entities:
        matches = len(question_entities.intersection(entities))
        if matches > max_matches:
            max_matches = matches
            best_segment = segment
    return best_segment
```

Purpose:

- Matches named entities in the user's question with those in the preprocessed segments.
- Selects the segment with the highest number of matching entities as the most relevant context.

Output:

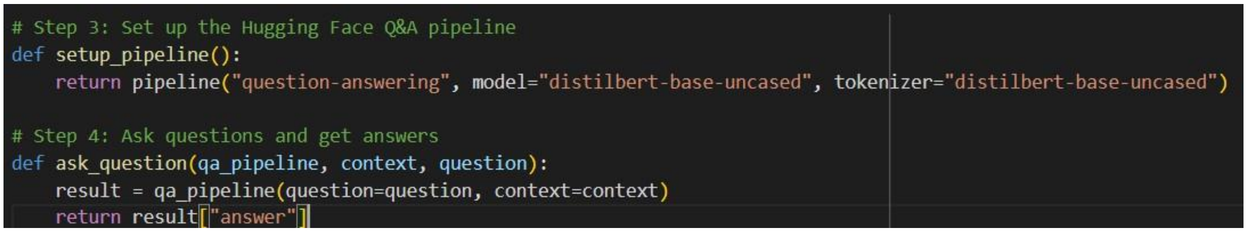
- The most relevant text segment.

Usage:

- Pass the question, preprocessed segments (segment_entities), and spaCy model to this function.

2. Model Selection and Setup

Code:



```python
# Step 3: Set up the Hugging Face Q&A pipeline
def setup_pipeline():
    return pipeline("question-answering", model="distilbert-base-uncased", tokenizer="distilbert-base-uncased")

# Step 4: Ask questions and get answers
def ask_question(qa_pipeline, context, question):
    result = qa_pipeline(question=question, context=context)
    return result["answer"]
```

Purpose:

- Initializes the Hugging Face Question-Answering pipeline using the pre-trained distilbert-base-uncased model.

Model Description:

This Question and Answer bot leverages **DistilBERT**, a smaller, faster, and lighter variant of the BERT (Bidirectional Encoder Representations from Transformers) model, fine-tuned for question-answering tasks.

Key Features of the Model:

- **Transformer Architecture**:
  - DistilBERT is based on the Transformer architecture, which excels in understanding contextual relationships between words and phrases in text.
- **Pretrained on Large Text Datasets**:
  - The model has been pre-trained on extensive corpora (e.g., Wikipedia and BookCorpus), making it capable of understanding a wide variety of natural language queries.
- **Fine-Tuned for Question Answering**:
  - Fine-tuning on datasets like SQuAD (Stanford Question Answering Dataset) enables the model to extract precise answers from a given context.

Why Use DistilBERT?

- **Efficiency**: DistilBERT is 60% smaller than BERT while retaining 97% of its language understanding capabilities. This makes it ideal for real-time applications like this bot.
- **Speed**: It runs faster than standard BERT models, making the QCA experience more seamless for users.
- **Accuracy**: Its fine-tuning on question-answering tasks ensures high-quality answers when provided with a relevant context.

How It Works in This Bot:

1. **Input Question**:
   - The user enters a question about Shakespeare’s works.

2. **Context Extraction**:
    - The bot first identifies the most relevant text segment from Shakespeare’s corpus using Named Entity Recognition (NER) and entity matching techniques.
3. **Answer Generation**:
    - The question and extracted context are passed to the DistilBERT model, which generates a concise and accurate answer.

Output:

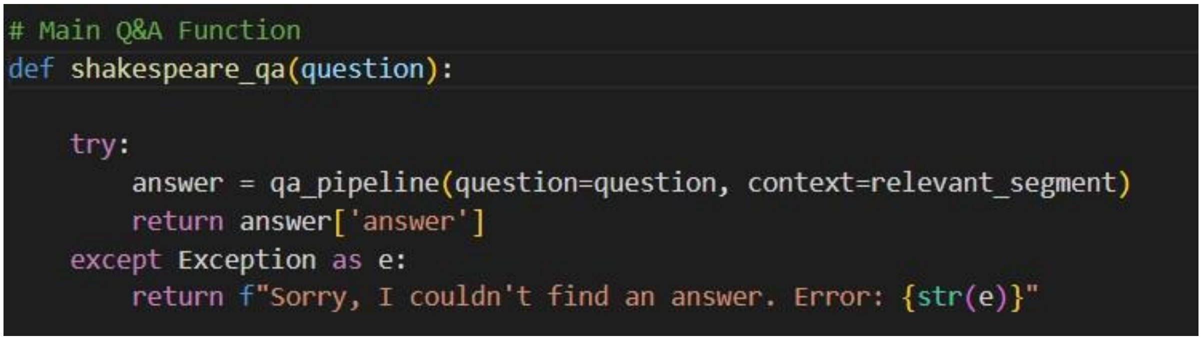
- A pipeline object for question-answering tasks.

Usage:

- Call this function to set up the pipeline.
- Example:

3. Design Q A system

Code:



```
# Main Q&A Function
def shakespeare_qa(question):

    try:
        answer = qa_pipeline(question=question, context=relevant_segment)
        return answer['answer']
    except Exception as e:
        return f"Sorry, I couldn't find an answer. Error: {str(e)}"
```

Purpose:

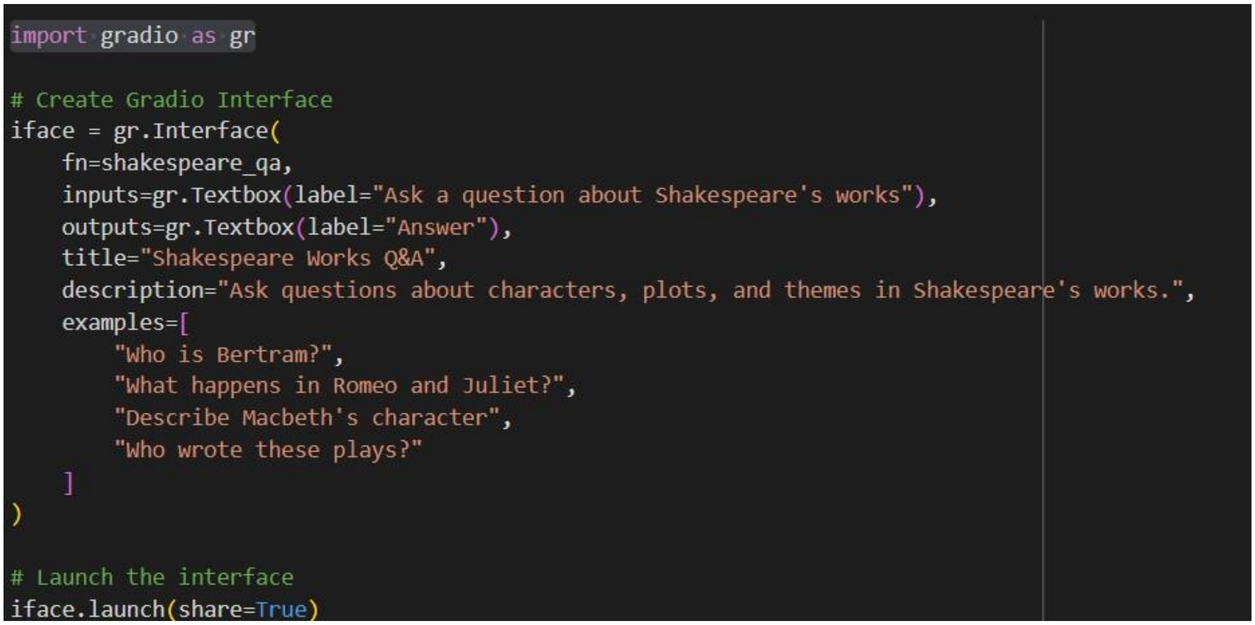
- Passes the user’s question and the relevant context to the Hugging Face pipeline.
- Retrieves and returns the answer from the pipeline’s output.

Usage:

- Provide the pipeline object, a text segment (context), and a question to this function.

4. Implementing User Interface

Code:



```
import gradio as gr

# Create Gradio Interface
iface = gr.Interface(
    fn=shakespeare_qa,
    inputs=gr.Textbox(label="Ask a question about Shakespeare's works"),
    outputs=gr.Textbox(label="Answer"),
    title="Shakespeare Works Q&A",
    description="Ask questions about characters, plots, and themes in Shakespeare's works.",
    examples=[
        "Who is Bertram?",
        "What happens in Romeo and Juliet?",
        "Describe Macbeth's character",
        "Who wrote these plays?"
    ]
)

# Launch the interface
iface.launch(share=True)
```

Purpose:

- This code sets up an interactive web interface for the QCA system using **Gradio**.
- Users can input questions related to Shakespeare's works and receive answers in real time.

Usage:

- **gr.Interface:**
  - **fn:** Specifies the function to handle user queries. In this case, shakespeare_qa is the function that processes questions and retrieves answers.
  - **inputs:** Defines the input field for the interface. Here, it's a text box labeled *"Ask a question about Shakespeare's works"*.
  - **outputs:** Defines the output field for the interface. Here, it's another text box labeled *"Answer"*.

    - **title**: Sets the title of the interface, displayed at the top of the page.
    - **description**: Provides additional details about the interface's functionality.
    - **examples**: Displays pre-defined example questions for users to try.
- **Launching the Interface**:
    - **iface.launch(share=True)**:
        - Starts the Gradio web interface.
        - The share=True option generates a public link, allowing anyone to access the interface via a web browser.

How to Use the Web Interface:

1. **Launching the Interface**:
    - Run the code block in your Python environment.
    - Gradio will generate a local URL (e.g., http://127.0.0.1:7860) and a public URL if share=True is used (e.g., https://1234abcd.gradio.app).
2. **Accessing the Interface**:
    - Open the public URL in a web browser.
    - The interface will display the following components:
        - **Title**: *"Shakespeare Works Q&A"*
        - **Description**: *"Ask questions about characters, plots, and themes in Shakespeare's works."*
        - **Input Textbox**: A field to type your question.
        - **Output Textbox**: A field to display the answer.
        - **Examples:** Predefined example questions to help users get started.
3. **Asking Questions**:
    - Type a question into the input text box (e.g., *"Who is Bertram?"*).
    - Click the *"Submit"* button (or press Enter).

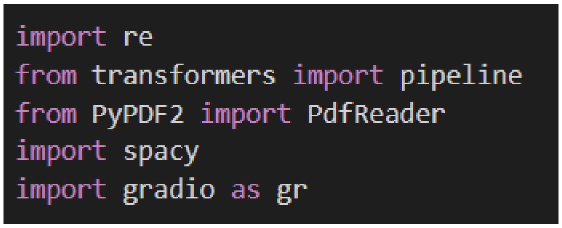
   - The interface will process the question using the QCA pipeline and display the answer in the output text box.

4. **Using Predefined Examples**:
   - Click on any example question (e.g., *"What happens in Romeo and Juliet?"*).
   - The interface will automatically populate the input box with the selected question.

Instructions to Set Up and Execute the Code:

Prerequisites

1. Install the required libraries:



```
import re
from transformers import pipeline
from PyPDF2 import PdfReader
import spacy
import gradio as gr
```

Steps to Run the Code

1. **Extract and Clean Text**:
   - Use extract_text_from_pdf to read the PDF.
   - Clean the text using clean_text.
2. **Preprocess Segments**:
   - Segment the text using segment_text.
   - Extract and cache entities with preprocess_segments_with_entities.
3. **Set Up the Pipeline**:
   - Initialize the QCA pipeline with setup_pipeline.
4. **Ask Questions**:
   - Find the relevant segment using find_relevant_segment.
   - Retrieve the answer using ask_question.

5. **Retrieve Answer:**

   - Use the ask_question function to pass the question and the relevant segment to the QCA pipeline.Clean the text using clean_text.
   - The pipeline will extract and return the answer.

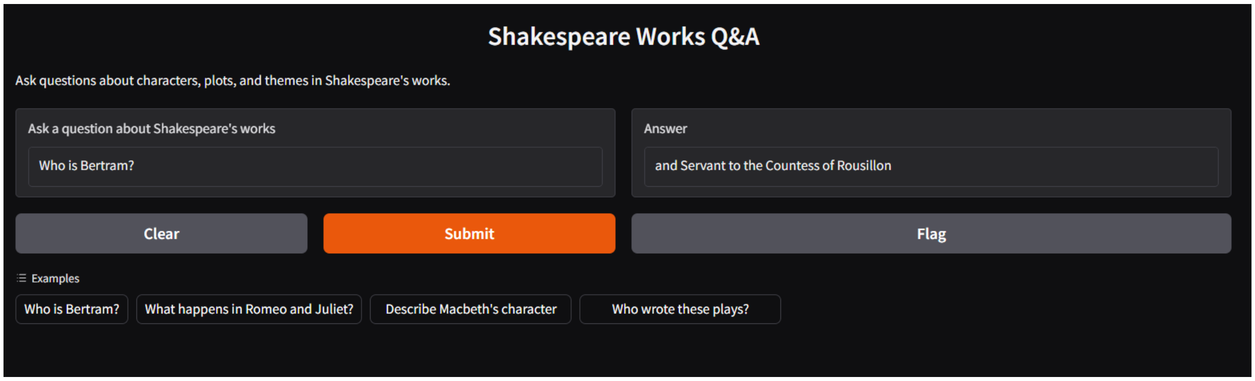
6. **Activate User Interface:**

Set Up Gradio Interface:

- The Gradio code (as shown in the image) sets up a user-friendly web interface.
- **Key Features:**
  - **fn=shakespeare_qa:** Connects the question-answering function (shakespeare_qa) to the interface.
  - **Input:** A text box labeled *"Ask a question about Shakespeare's works"*.
  - **Output:** A text box labeled *"Answer"* for displaying the generated answers.
  - **Examples:** Predefined sample questions like:
    - *"Who is Bertram?"*
    - *"What happens in Romeo and Juliet?"*
    - *"Describe Macbeth's character"*
    - *"Who wrote these plays?"*

Launch the Interface:

- Use the iface.launch(share=True) function to start the Gradio interface.
- This generates a public URL that can be shared for accessing the QCA bot



Shakespeare Works Q&A

Ask questions about characters, plots, and themes in Shakespeare's works.

Ask a question about Shakespeare's works: Who is Bertram?

Answer: and Servant to the Countess of Rousillon

Clear
Submit
Flag
Examples
Who is Bertram?
What happens in Romeo and Juliet?
Describe Macbeth's character
Who wrote these plays?