Question - 1:

Approach:

The approach involves sequentially applying several text preprocessing steps to prepare the text data for analysis or further processing. Each step modifies the text in a specific way, such as converting to lowercase, tokenizing, removing stopwords, removing punctuation, and removing blank space tokens. After each preprocessing step, the resulting text is saved into new files for subsequent tasks.

Methodologies:

> *Lowercasing the Text: Convert all text to lowercase using the lower() method.*
> *Tokenization: Tokenize the lowercase text using NLTK's word_tokenize function.*
> *Removing Stopwords: Filter out common English stopwords using NLTK's predefined list.*
> *Removing Punctuations: Eliminate punctuation marks using Python's `string.punctuation` module.*
> *Removing Blank Space Tokens: Remove any remaining tokens that consist only of whitespace characters.*

For each step, the script processes the text files one by one, saving the preprocessed content into new files after each operation. This ensures that the output of each step can be used as input for the subsequent step.

Assumptions:

- *The text files are in English.*
- *NLTK's English stopwords list is suitable for removing irrelevant words.*
- *Punctuation marks are not essential for the analysis or downstream tasks.*
- *Blank space tokens do not contribute to the meaning of the text.*

Results:

> *Lowercasing the Text: The text is converted to lowercase, preserving the original content's semantic meaning while standardizing the text for consistency.*
> *Tokenization: The lowercase text is split into individual tokens, facilitating further analysis or processing on a word level.*
> *Removing Stopwords: Common stopwords such as "the," "and," and "is" are removed, reducing noise and irrelevant information in the text.*
> *Removing Punctuations: Punctuation marks such as commas, periods, and quotation marks are eliminated, simplifying the text for analysis and improving computational efficiency.*
> *Removing Blank Space Tokens: Any remaining tokens consisting solely of whitespace characters are removed, ensuring the cleanliness of the text data.*

The original and preprocessed contents of five sample files are printed before and after each operation, demonstrating the effectiveness of each preprocessing step in cleaning and standardizing the text data for

further analysis or tasks. Additionally, each preprocessed file is saved after each preprocessing step, allowing for the use of preprocessed files in subsequent tasks or analyses.

Question-2:

Approach:

The task involves creating a Unigram Inverted Index and supporting Boolean queries based on the preprocessed text data obtained from Question 1. We'll build the index from scratch without using any libraries, implement support for various Boolean query operations, preprocess input queries, and provide the functionality to save and load the index using Python's pickle module.

Methodologies:

Creating Unigram Inverted Index:
- *Tokenize preprocessed text files and map each term to the documents it appears in.*
- *Store this mapping in a data structure, typically a dictionary where keys are terms and values are sets of document names.*

Saving and Loading Index:
- *Utilize Python's pickle module to serialize and deserialize the inverted index, enabling efficient storage and retrieval.*

Supporting Boolean Queries:
- *Implement functions for Boolean operations like AND, OR, AND NOT, and OR NOT.*
- *Use the inverted index to perform these operations efficiently, retrieving relevant document sets for each term.*

Preprocessing Input Queries:
- *Apply preprocessing steps similar to those used in Question 1 to standardize and clean the input query sequence.*

Input and Output Formats:
- *Define clear formats for input queries and output results, making it easy for users to interact with the system and understand the results.*

Assumptions:

- *The input text data has been preprocessed as per the instructions provided in Question 1.*
- *Queries are in a simple format without complex syntax or nesting.*
- *The system operates on a single machine and doesn't require distributed computing.*
- *The dataset fits into memory for efficient processing.*

Results:

- *Successfully create a Unigram Inverted Index from scratch, capturing the relationship between terms and documents in the dataset.*
- *Implement functions to perform Boolean queries efficiently using the inverted index.*

- *Support a variety of query operations, including AND, OR, AND NOT, and OR NOT, as well as generalized queries combining these operations.*
- *Provide functionality to save and load the inverted index for future use, ensuring persistence across sessions.*
- *The system preprocesses input queries and returns results in a clear and understandable format, facilitating user interaction and interpretation of results.*

Question-3:

Approach:

The task involves creating a Positional Index for the dataset obtained from Question 1 and supporting phrase queries. We'll build the positional index from scratch without using any libraries, implement support for phrase queries, and provide functionality to save and load the index using Python's pickle module.

Methodologies:

Creating Positional Index:
- *Tokenize preprocessed text files and build a positional index that maps each term to the documents it appears in along with the positions of the term in each document.*
- *Store this mapping in a data structure, typically a nested dictionary where keys are terms, and values are dictionaries with document names as keys and lists of positions as values.*

Saving and Loading Index:
- *Utilize Python's pickle module to serialize and deserialize the positional index, enabling efficient storage and retrieval.*

Supporting Phrase Queries:
- *Implement functions to handle phrase queries, which involve finding documents where a specific sequence of terms occurs consecutively.*
- *Utilize the positional index to efficiently retrieve documents satisfying the phrase queries.*

Preprocessing Input Queries:
- *Apply preprocessing steps similar to those used in Question 1 to standardize and clean the input phrase queries.*

Input and Output Formats:
- *Define clear formats for input queries and output results, making it easy for users to interact with the system and understand the results.*

Assumptions:

- *The input text data has been preprocessed as per the instructions provided in Question 1.*
- *Phrase queries consist of consecutive terms in the text without any intervening terms.*
- *The system operates on a single machine and doesn't require distributed computing.*
- *The dataset fits into memory for efficient processing.*
- *The length of the input sequence is limited to 5 queries.*

Results:

- *Successfully create a Positional Index from scratch, capturing the relationship between terms, documents, and their positions in the dataset.*
- *Implement functions to perform phrase queries efficiently using the positional index, retrieving relevant documents where the specified phrases occur.*
- *Provide functionality to save and load the positional index for future use, ensuring persistence across sessions.*
- *The system preprocesses input phrase queries and returns results in a clear and understandable format, facilitating user interaction and interpretation of results.*