Ques 1.

Soln. In this ques we have done the data processing on the given data set , to clean and prepare the data that can be used for making the proper index where we have performed the various functions i.e. -:

1. **Lowercase the text:-** lowercasing involves converting all letters in the text to lowercase, to ensure uniformity and consistency in the text data.

2. **Perform tokenization: -** Tokenization is the process of breaking down a text into individual words or token, this process allows us to examine the text at a more granular level, and this process is done with the help of the NTLK library.

3. **Remove stop words: -** Stop words are common words that usually don't contribute much to the meaning of a sentence. Eg "the", "and", "is"etc. This process helps to reduce the dimensionality of the data and focus more on the meaningful words.

4. **Remove punctuations: -** Punctuation removal involves eliminating punctuation marks from the text. Punctuation typically doesn't carry much semantic meaning and can be easily removed.

5. **Remove Blank space tokens:** - Removing blank space tokens involves eliminating any remaining empty tokens that might have been created during previous processing steps. This ensures that only meaningful tokens are retained.

Ques 2.

Soln.

About Unigram index - A unigram inverted index is a data structure that maps each unique term (unigram) to the set of document IDs in which that term appears. The inverted index is constructed after preprocessing the dataset. The process involves iterating through each document, tokenizing it, and creating an inverted index.

Creating the Unigram Index -:

1. Giving each file a doc_id, so that each file can be uniquely identified
2. We use a dictionary where keys are tokens, and values are sets or lists of document IDs.
3. Using the for we will iterate through all the file and in the each file we will iterate through all the tokens and built a unigram index using a dictionary in the python
4. Update the inverted index for each document by adding the document ID to the corresponding token's entry.

Searching a key in the Unigram Index -:

In this step user will input how many query does it want to check into the system , and on that query which operation he wants to perform i.e. "and", "or", "andnot", "ornot"etc.

The above Boolean expression have there own meaning

1. **And -:** Will only print the docid in which both the word in the query are present
2. **Or -:** Will only print the docid where either word in the query are present
3. **Andnot -:** will only print the docid where word_1 is present and word_2 is absent
4. **Ornot -:** will only print the docid where word1 is present and word2 is absent

Saving the index using Pickle library

The **pickle** module in Python is commonly used for serializing and deserializing Python objects, allowing you to save objects to a file and load them back into memory.

It is mainly use in python to :-

1. Persistence
2. Efficient Storage
3. Data Sharing
4. Reduced Computation overhead

Therefore, due to above reason we save the unigram index using the module pickle

To save the index using the pickle module, this is the given below code :-

```
def save_index_to_pickle(index, filename):
    with open(filename, 'wb') as file:
    pickle.dump(index, file)
```

To load the index from the pickle file using the pickle module, this is the given below code :-

```
def load_index_from_pickle(filename):
    with open(filename, 'rb') as file:
    return pickle.load(file)
```

Note – Here filename consist of the name of the filename to which we want to sve the index , filename with the extension of .pkl

Ques 3. Creation the positional index with the help of the same above pre-processed data

Soln. A positional index is a data structure used in information retrieval systems, particularly in search engines, to keep track of the positions of terms within documents. Unlike a simple inverted index that only records the presence of terms in documents, a positional index stores additional information about where terms occur in the documents. This added information enables more sophisticated and accurate retrieval of relevant documents.

Positional index is created using the dictionary where mapping from terms to a list of pairs, where each pair consists of a document ID and the positions of the term in that document id done.

We use the positional index to check the result of the phrase query.

Phrase query is a type of search query used in information retrieval systems to find documents that contain an exact sequence of terms in a specified order. Unlike simple keyword queries, which look for the presence of individual terms, a phrase query requires the terms to appear consecutively and with the specified order within the document.

Searching a phrase query in the Positional index:-

Searching for a phrase query in a positional index is a specialized operation that ensures the precision of results by considering not only the presence but also the order of terms within documents. The design of the positional index and the efficiency of the search algorithm significantly impact the performance of phrase query searches.
We mainly use the phrase query where we have to words that are adjacent to each other or they must be appear together so therefore we use the positional index because positional index is having the position of the words due to which we matches the position of the preceding term and the succeeding sum, i.e. if the position of the succeeding terms is 1 greater than the proceeding term in the same

doc_id then we include doc , in this way we check for all the data file that are present in dataset , if the query is not present in any of the datafile then we return an empty set

Code:- Searching in the positional index

```
def search_phrase(index, terms):
    if len(terms) < 2:
        raise ValueError("Phrase queries must contain at least two  terms.")
    result_set = set(index[terms[0]])
    for term in terms:
        term_positions = index[term]
        result_set.intersection_update(set(term_positions.keys()))
    result_copy = set(result_set)
    lst=[]
    for doc_id in result_copy:

      positions = [set(index[term][doc_id]) for term in terms]
      positions_1 = positions[0]
      positions_2 = positions[1]
      if any(position2 - 1 == position1 for position1 in positions_1      for position2 in positions_2):
          lst.append(doc_id)
    return lst
```

Result after searching for a phrase query will be:-
If the match found – then return the set having the doc_id that contain the phrase query.
If the match not found within any document in the given dataset then straight way return empty dataset

Saving the positional index in the pickle file so that after it can be loaded from the pickle whenever needed without, to save the time needed the positional index again

```
import pickle       #importing the pickle module
pickle_filename = "index.pkl"       #file in which index will be stored
```

Code for saving the positional index in the “index.pkl” file

```
 def save_index_to_pickle(index, filename):
    with open(filename, 'wb') as file:
        pickle.dump(index, file)
```

Code for loading the positional index form the “index.pkl” file to the dictionary name “index”

```
def load_index_from_pickle(filename):
    with open(filename, 'rb') as file:
        return pickle.load(file)
```

Assumptions that I have taken in the above code:-

1. **Assumptions that are taking while data preprocessing: -**

- Data is in one language.
- Treating text as case-insensitive to simplify matching and ensure consistency.
- Assuming standard tokenization rules
- Using a predefined list of stop words for removal.
- Treating numeric values as non-textual entities and not applying specific natural language processing to them.

2. **Assumption taken in unigram index: -**
    - Assuming a one-to-one mapping between document IDs and documents.
    - Assuming having enough main memory space
    - Assuming empty set for the word if it is not present in any given file of the given dataset
    - Assuming a standard query language or format for simplicity in query processing.

3. **Assumption taken in the positional index: -**
    - Assuming a consistent and standard tokenization approach that captures meaningful terms.
    - Storing precise positional information to support phrase queries and proximity searches.
    - Treating all terms equally without considering term weighting schemes
    - Assuming that document IDs are assigned sequentially and there are no gaps or missing document IDs.

Result for the given assignment –

1. **Data preprocessing on the any 5 files given in the dataset**

Output: -

```
file982.txt
i love how this baby sounds! it has a nice full loud sound. tones are just right! its better looking t
['i', 'love', 'how', 'this', 'baby', 'sounds', '!', 'it', 'has', 'a', 'nice', 'full', 'loud', 'sound',
['love', 'baby', 'sounds', '!', 'nice', 'full', 'loud', 'sound', '.', 'tones', 'right', '!', 'better',
[' love', ' baby', ' sounds', ' ', ' nice', ' full', ' loud', ' sound', ' ', ' tones', ' right', ' ',
[' love', ' baby', ' sounds', ' nice', ' full', ' loud', ' sound', ' tones', ' right', ' better', ' lo
file924.txt
i though this was gonna be three green packs separate, but came in one green pack. it has all 3 sets b
['i', 'though', 'this', 'was', 'gonna', 'be', 'three', 'green', 'packs', 'separate', ',', 'but', 'came
['though', 'gonna', 'three', 'green', 'packs', 'separate', ',', 'came', 'one', 'green', 'pack', '.', '
[' though', ' gonna', ' three', ' green', ' packs', ' separate', ' ', ' came', ' one', ' green', ' pac
[' though', ' gonna', ' three', ' green', ' packs', ' separate', ' came', ' one', ' green', ' pack', '
file405.txt
this guitar is far more beautiful in person. the red to gold tones blend so wonderfully and it has a b
['this', 'guitar', 'is', 'far', 'more', 'beautiful', 'in', 'person', '.', 'the', 'red', 'to', 'gold',
['guitar', 'far', 'beautiful', 'person', '.', 'red', 'gold', 'tones', 'blend', 'wonderfully', 'barely'
[' guitar', ' far', ' beautiful', ' person', ' ', ' red', ' gold', ' tones', ' blend', ' wonderfully',
[' guitar', ' far', ' beautiful', ' person', ' red', ' gold', ' tones', ' blend', ' wonderfully', ' ba
file938.txt
i find this unit adequate, i use it with my macbook as part of a portable 'winter' studio. i also own
['i', 'find', 'this', 'unit', 'adequate', ',', 'i', 'use', 'it', 'with', 'my', 'macbook', 'as', 'part'
['find', 'unit', 'adequate', ',', 'use', 'macbook', 'part', 'portable', "'", 'winter', "'", 'studio',
[' find', ' unit', ' adequate', ' ', ' use', ' macbook', ' part', ' portable', ' ', ' winter', ' ', '
[' find', ' unit', ' adequate', ' use', ' macbook', ' part', ' portable', ' winter', ' studio', ' also
file970.txt
```

2. **Creating a unigram index , Screenshot for some of the tokens**

Output: -

```
Term: concept, Document IDs: {96}
Term: concern, Document IDs: {873, 764, 767}
Term: concerned, Document IDs: {328, 524, 48, 51, 152, 61}
Term: concerning, Document IDs: {835}
Term: concerns, Document IDs: {483}
Term: concert, Document IDs: {120, 465, 138, 621}
Term: conclude, Document IDs: {108}
Term: concorde, Document IDs: {690}
Term: condenser, Document IDs: {512, 833, 228, 133, 872, 308, 631, 637}
Term: conderser, Document IDs: {146}
Term: condition, Document IDs: {65, 363, 660, 505, 698}
Term: conditioner, Document IDs: {306}
Term: conditions, Document IDs: {130, 359}
Term: cone, Document IDs: {855}
Term: confidence, Document IDs: {33, 397, 150}
Term: confident, Document IDs: {111, 788, 596, 443, 349}
Term: configuration, Document IDs: {235}
Term: configure, Document IDs: {382}
Term: configured, Document IDs: {830}
Term: configuring, Document IDs: {22, 55}
Term: confirm, Document IDs: {174}
Term: confirmation, Document IDs: {334}
Term: conform, Document IDs: {379}
Term: confuse, Document IDs: {127}
Term: confused, Document IDs: {39}
```

3. Saving the unigram index in the .pkl file and also loading the unigram index from that file

Output:-

```
Unigram index saved to unigram_index.pkl
Unigram index loaded successfuly
```

4. Finding the given in the file using operation (“and”, “or”, “andnot”, “ornot”)

Output:- The below output shows when we are entering the string “cat is a bag and is a mystery ” and want to perform “and ” and “or ” operation on it then the following given below is the output

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Enter the no of statement you want to search 1
Enter the string you want to enter :- cat is a bag and is a mystery
Enter the no of operation you want to perform:- 2
Enter the operation:-and
Enter the operation:-or
[899]
Thank you
```

5. Creating the positional index from the pre-processed files

Output:- Small instance of a positional index , how it is being created

```
Term: yoy
  Document ID: 893, Positions: [11]
Term: yr
  Document ID: 62, Positions: [5]
Term: yrs
  Document ID: 681, Positions: [9]
  Document ID: 418, Positions: [9]
Term: yuk
  Document ID: 666, Positions: [13]
Term: z
  Document ID: 673, Positions: [2, 21]
Term: zakk
  Document ID: 196, Positions: [2]
  Document ID: 360, Positions: [7]
Term: zelda
  Document ID: 935, Positions: [1, 35]
```

6. Saving the positional index into the index.pkl and loading the positional index from index.pkl

Output: -

```
Unigram index saved to index.pkl
Unigram index is loaded successfully
```

7. Searching the phrase query in the positional index

Output:-

```
The result for the query:- gig is a bag
Documents containing the phrase 'gig is a bag': [864, 3, 73, 363, 174, 942, 886, 118, 665]
```