Methodologies-

Q1(i)

To convert the files given in the dataset to XML files, we added **<?xml version="1.0" encoding="UTF-8"?>** as the first line of every file. This was done so that we could use Beautiful Soup on the files.

Changed doctype to make it readable in beautiful soup

```
'''
#open and add a line at the starting of the file
def add_line(filename):
    with open(filename, 'r+') as f:
        content = f.read()
        f.seek(0, 0)
        f.write('<?xml version="1.0" encoding="UTF-8"?>' + '\n' + content)
        f.close()

#add a line at the starting of all the files
for filename in os.listdir():
    add_line(filename)
'''
```

After converting the files into XML files, we used Beautiful Soup Library to extract the desired content and save it in the same files.

Q1(ii)

We used the nltk library to preprocess our XML files which we received from the previous part. We made a function which would take a file as input and first lowercase every character. Then it will create tokens for each word. Here we assumed the hyphen separated words as a single word. Then we remove the stopwords, and after that we remove punctuations. Then blank spaces are removed. Also, we store the order of every token in a list if it occurs more than once in a sentence and then remove the duplicates. This is to make sure that order of duplicate words remains preserved.

```
# function with subfunctions to Lowercase the text Perform tokenization, Remove stopwords, Remove punctuations, Remove blank space tokens using filnam
def preprocess(filename):
    # open the file and read the text
    with open(filename, 'r') as f:
        text = f.read()
    # Lowercase the text
    text = text.lower()

    # Perform tokenization
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # remove punctuations but not hyphen separated words
    tokens = [w for w in tokens if w.isalpha() or '-' in w]
    # Remove blank space tokens
    tokens = [w for w in tokens if w.strip()]
    # Remove tokens that occur twice, preserve the order of tokens in list
    tokens = list(dict.fromkeys(tokens))
    return tokens
```

```
['My', 'name', 'is', 'Uttkasrs-singh']
```

Then we make a dictionary to store the tokens and corresponding to the tokens store the list of document id in which those words are present and also store the count of each token at the end of the lists.

```
db = {}

# make a dictionary of token conatining the list of files and size of list it is stored in and store the dictionary in db
def make_dict(tokens, filename):
    for token in tokens:
        if token in db:
            db[token][0].append(filename)
            db[token][1] += 1
        else:
            db[token] = [[filename], 1]
```

Then we saved the dictionary in a pickle file and also converted the pickle file to a text file to improve the readability of the file.

Q2

Firstly we load the dictionary made in the previous part. We also made a function which will give us the list of all documents in which the word exists (from the dictionary).

```
#load the invertedindex data
with open('invertedindex.pickle', 'rb') as handle:
    invertedindex = pickle.load(handle)

def get_posting_list(term):
    if term in invertedindex:
        return invertedindex[term][0]
    else:
        return []

def get_posting_listsize(term):
    if term in invertedindex:
        return invertedindex[term][1]

os.chdir('../CSE508_Winter2023_Dataset')
all_docs = os.listdir()
```

We made an AND query which will first get a list of documents of each term on which we want to run the AND operator. Then we call the AND helper function in which we extract the posting list for both the terms and then check whether both the terms exist in the same file or not. If they are, then the filename is appended to the result.

We made an AND NOT query which will first get a list of documents of each term on which we want to run the AND NOT operator. We call the AND NOT helper function in which the posting list for both the terms is extracted and we perform the AND operation for all the files

We made an OR query which will first get a list of documents of each term on which we want to run the OR operator. We call the OR helper function in which the posting list for both the terms is extracted, and if either of the terms is present in the files, then that filename is appended to the result.

We made an OR NOT query which will first get a list of documents of each term on which we want to run the OR NOT operator. We call the OR NOT helper function in which the posting list for both the terms is extracted and we perform the OR operation for all the files in which the first term is present and the files in which the second term is not present. The resulting filenames are appended to the result.

For Input-Output, we take input n which is number of queries and the next 2*n lines contain the queries that need to be processed. Then we perform the preprocessing steps from Q1 on the input sequence.

```
for i in range(n):
    query = phrase[i]
    # query = query.split()
    query = query.lower()
    # Perform tokenization
    tokens = word_tokenize(query)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # remove punctuations but not hyphen separated words
    tokens = [w for w in tokens if w.isalpha() or '-' in w]
    # Remove blank space tokens
    tokens = [w for w in tokens if w.strip()]
    query = set(tokens)
    op = ops[i]
    op = op.split(',')
```

Then we perform the desired operations as asked in the query and display the output.

Q3

For Bigram Inverted Index

We preprocess the files and then create a dictionary for creating a bigram inverted index. We store tokens as a pair and corresponding to them we store the list of document id in which they are present. Then we saved the dictionary in a pickle file and converted the pickle file to a text file to improve the readability of the file.

For Positional Index

We made a dictionary in which token was stored as key and corresponding to each key we stored a dictionary in which documents were stored as key and corresponding to each document we stored a list in which position of each token was stored. Then we saved the dictionary in a pickle file and converted the pickle file to a text file to improve the readability of the file.

Assumption-

Included hyphen separated word as a single word

For Example- `['My', 'name', 'is', 'Uttkasrs-singh']`

We did not consider all the words in the positional index. We considered those words which we received from the preprocessing data from question 1.

Result-

The Bigram Index will give False Positive in some cases as if it finds two words adjacently irrespective of their positions then it will return true.
Unlike Bigram Index, the Positional Index will never give False Positive as it always considers the positions of the words as their positions are stored in the index.