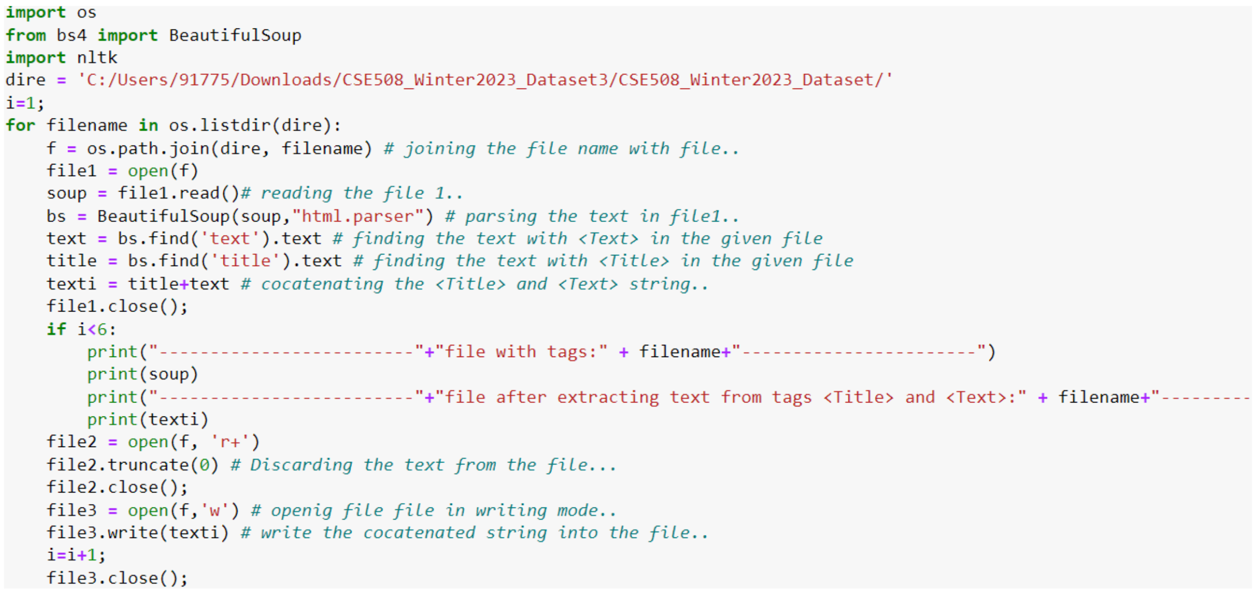
Information Retrieval Report

Q1. Data Preprocessing——-------------------------

i) Relevant Text Extraction:

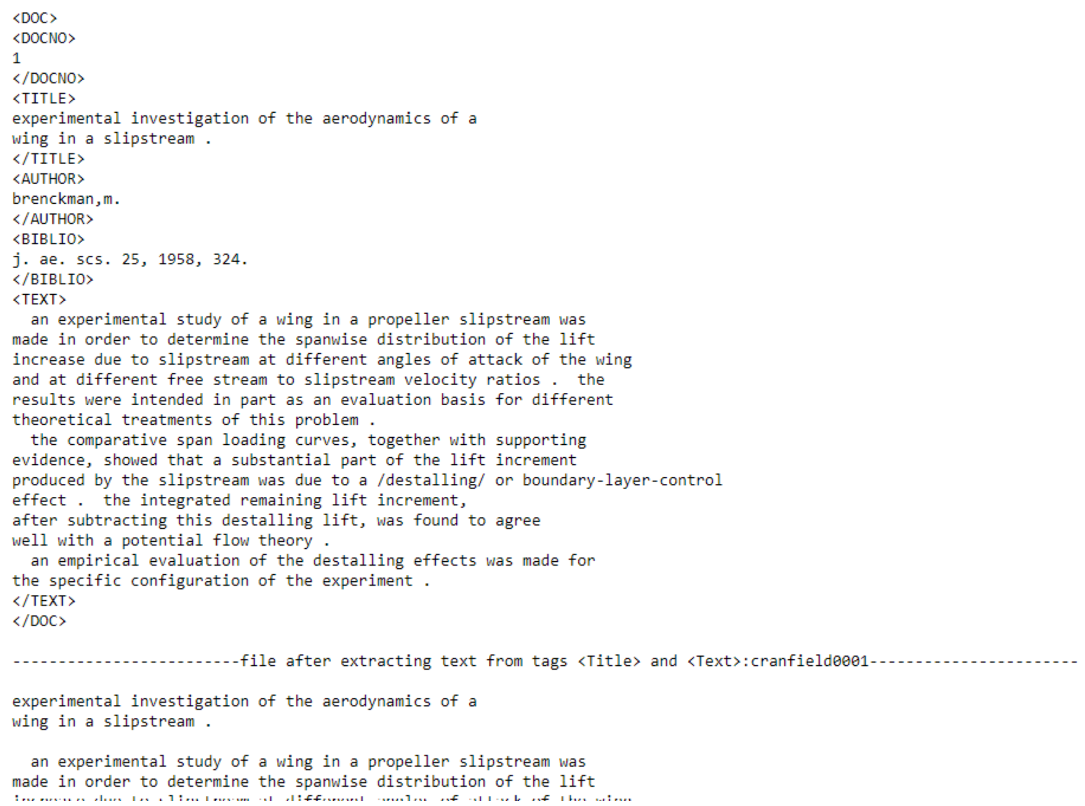
Opened the original file and the file is parsed using BeautifulSoup, then the text with tags <Title> and <Text> is retrieved from the file and all the text is discarded from the file and then the empty file is appended with the text which we have retrieved early on.

Code snippet of the above operation—:



```python
import os
from bs4 import BeautifulSoup
import nltk
dire = 'C:/Users/91775/Downloads/CSE508_Winter2023_Dataset3/CSE508_Winter2023_Dataset/'
i=1;
for filename in os.listdir(dire):
    f = os.path.join(dire, filename) # joining the file name with file..
    file1 = open(f)
    soup = file1.read()# reading the file 1..
    bs = BeautifulSoup(soup,"html.parser") # parsing the text in file1..
    text = bs.find('text').text # finding the text with <Text> in the given file
    title = bs.find('title').text # finding the text with <Title> in the given file
    texti = title+text # cocatenating the <Title> and <Text> string..
    file1.close();
    if i<6:
        print("-------------------------"+"file with tags:" + filename+"-----------------------")
        print(soup)
        print("-------------------------"+"file after extracting text from tags <Title> and <Text>:" + filename+"---------
        print(texti)
    file2 = open(f, 'r+')
    file2.truncate(0) # Discarding the text from the file...
    file2.close();
    file3 = open(f,'w') # openig file file in writing mode..
    file3.write(texti) # write the cocatenated string into the file..
    i=i+1;
    file3.close();
```

Output:



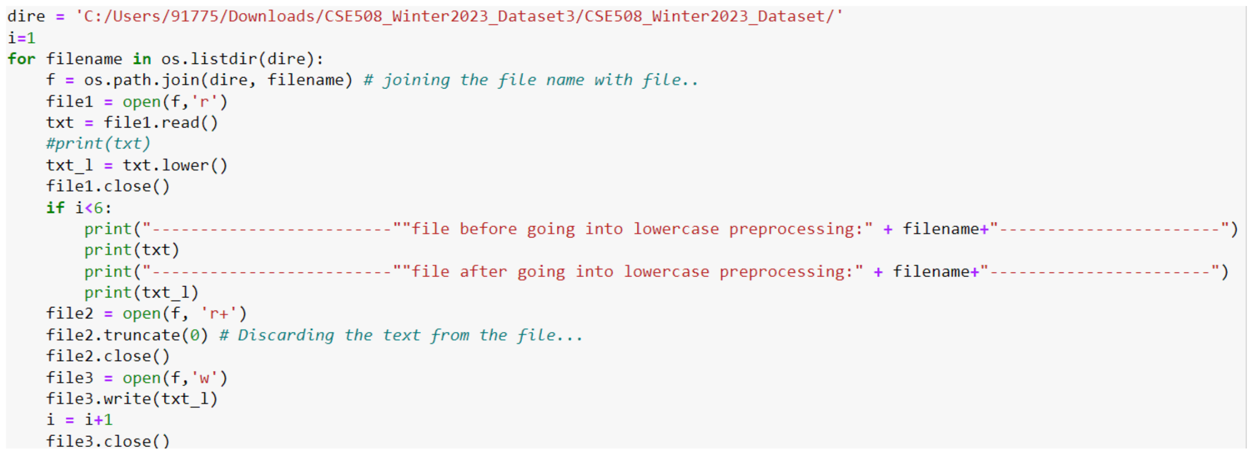
```
<DOC>
<DOCNO>
1
</DOCNO>
<TITLE>
experimental investigation of the aerodynamics of a
wing in a slipstream .
</TITLE>
<AUTHOR>
brenckman,m.
</AUTHOR>
<BIBLIO>
j. ae. scs. 25, 1958, 324.
</BIBLIO>
<TEXT>
  an experimental study of a wing in a propeller slipstream was
made in order to determine the spanwise distribution of the lift
increase due to slipstream at different angles of attack of the wing
and at different free stream to slipstream velocity ratios .  the
results were intended in part as an evaluation basis for different
theoretical treatments of this problem .
  the comparative span loading curves, together with supporting
evidence, showed that a substantial part of the lift increment
produced by the slipstream was due to a /destalling/ or boundary-layer-control
effect .  the integrated remaining lift increment,
after subtracting this destalling lift, was found to agree
well with a potential flow theory .
  an empirical evaluation of the destalling effects was made for
the specific configuration of the experiment .
</TEXT>
</DOC>

-------------------------file after extracting text from tags <Title> and <Text>:cranfield0001-----------------------

experimental investigation of the aerodynamics of a
wing in a slipstream .

  an experimental study of a wing in a propeller slipstream was
made in order to determine the spanwise distribution of the lift
```

ii) Preprocessing:

Lowercase the text:

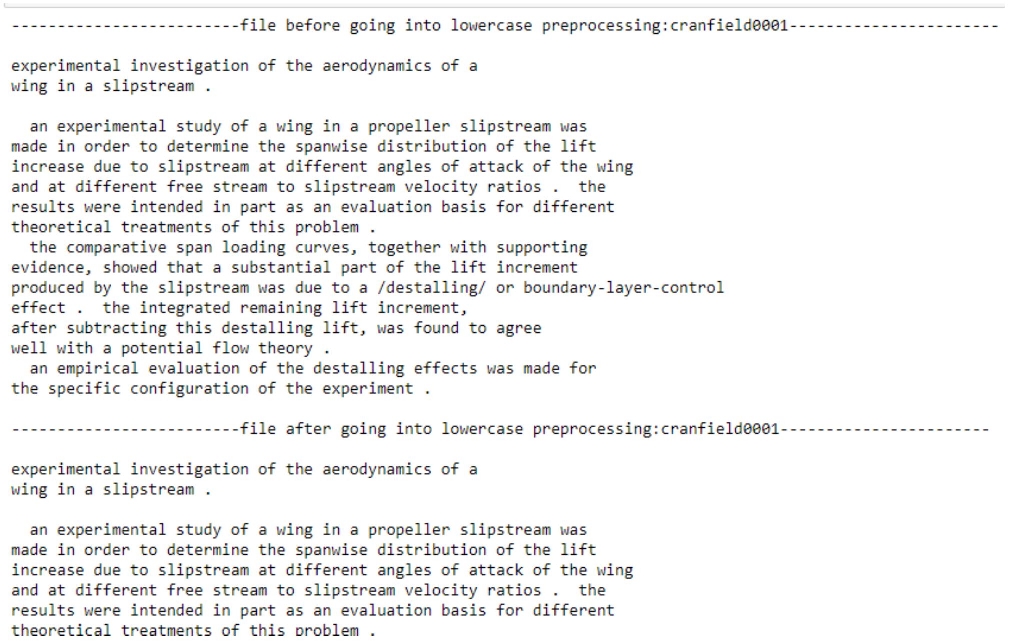
Opened the previously modified file and retrieved the text and covert every letter into lowercase and store it into the string. Then all the text is discarded from the opened file and then the empty file is appended with the above string.

Code snippet of the above operation—:



```python
dire = 'C:/Users/91775/Downloads/CSE508_Winter2023_Dataset3/CSE508_Winter2023_Dataset/'
i=1
for filename in os.listdir(dire):
    f = os.path.join(dire, filename) # joining the file name with file..
    file1 = open(f,'r')
    txt = file1.read()
    #print(txt)
    txt_l = txt.lower()
    file1.close()
    if i<6:
        print("------------------------""file before going into lowercase preprocessing:" + filename+"----------------------")
        print(txt)
        print("------------------------""file after going into lowercase preprocessing:" + filename+"----------------------")
        print(txt_l)
    file2 = open(f, 'r+')
    file2.truncate(0) # Discarding the text from the file...
    file2.close()
    file3 = open(f,'w')
    file3.write(txt_l)
    i = i+1
    file3.close()
```

Output:

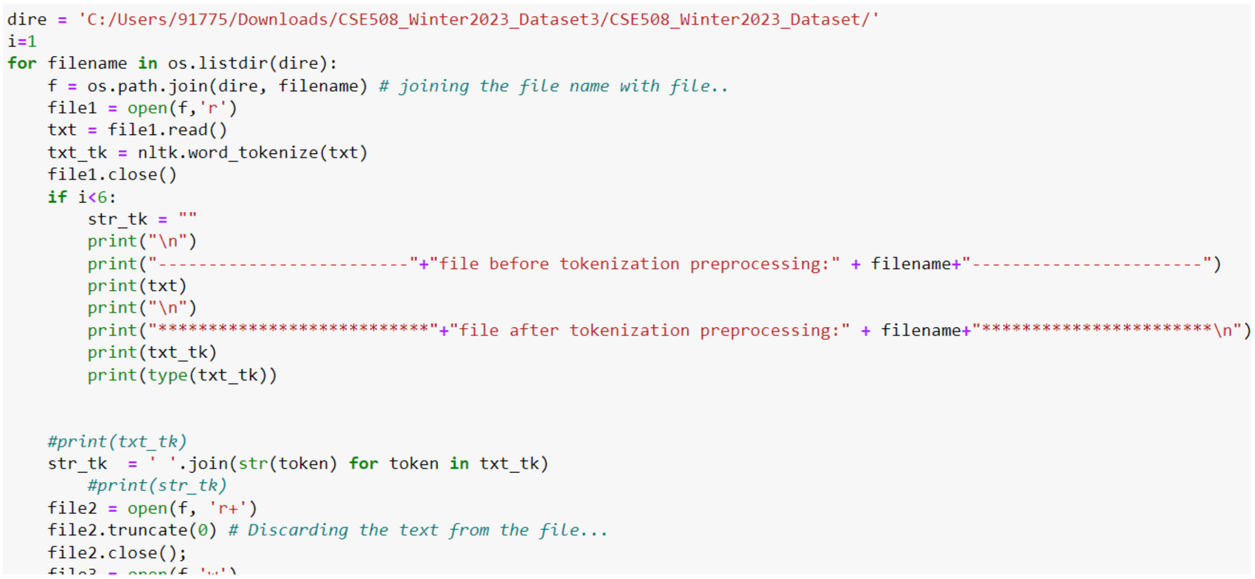


```
------------------------file before going into lowercase preprocessing:cranfield0001----------------------

experimental investigation of the aerodynamics of a
wing in a slipstream .

  an experimental study of a wing in a propeller slipstream was
made in order to determine the spanwise distribution of the lift
increase due to slipstream at different angles of attack of the wing
and at different free stream to slipstream velocity ratios .  the
results were intended in part as an evaluation basis for different
theoretical treatments of this problem .
  the comparative span loading curves, together with supporting
evidence, showed that a substantial part of the lift increment
produced by the slipstream was due to a /destalling/ or boundary-layer-control
effect .  the integrated remaining lift increment,
after subtracting this destalling lift, was found to agree
well with a potential flow theory .
  an empirical evaluation of the destalling effects was made for
the specific configuration of the experiment .

------------------------file after going into lowercase preprocessing:cranfield0001----------------------

experimental investigation of the aerodynamics of a
wing in a slipstream .

  an experimental study of a wing in a propeller slipstream was
made in order to determine the spanwise distribution of the lift
increase due to slipstream at different angles of attack of the wing
and at different free stream to slipstream velocity ratios .  the
results were intended in part as an evaluation basis for different
theoretical treatments of this problem .
```

Perform Tokenization:

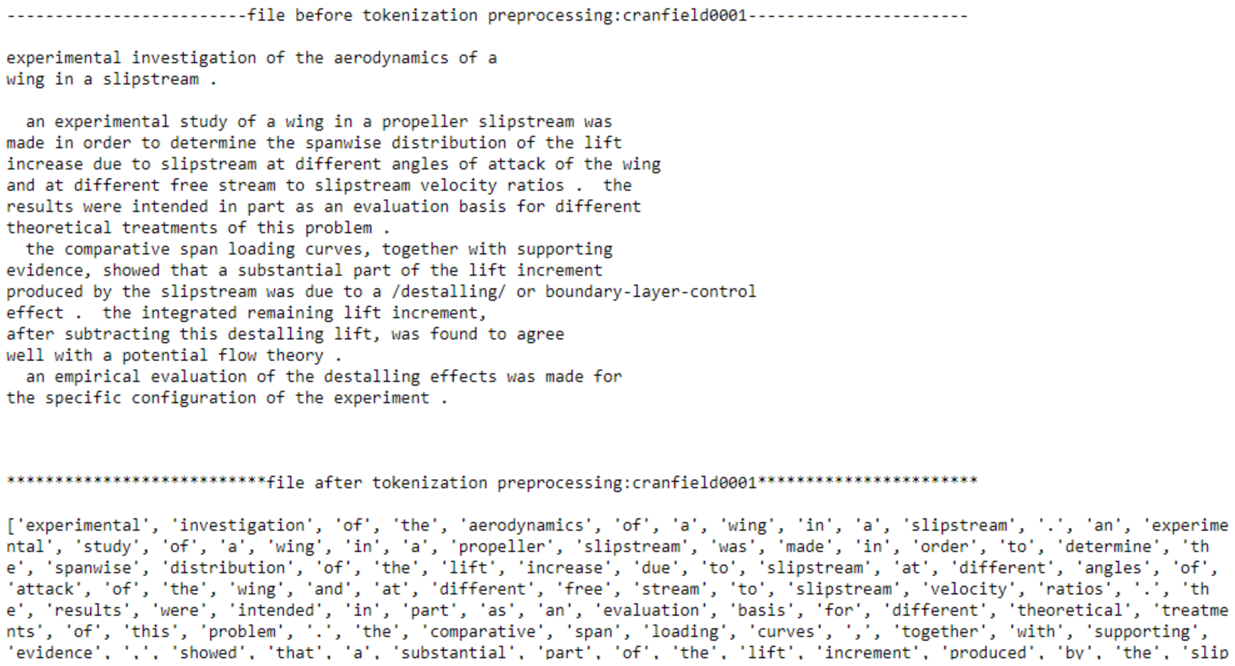
Opened the previously modified file and retrieved the text and tokenize it using nltk and then join the token into string. Then all the text is discarded from the opened file and then the empty file is appended with the above string.

Code snippet of the above operation—:



```python
dire = 'C:/Users/91775/Downloads/CSE508_Winter2023_Dataset3/CSE508_Winter2023_Dataset/'
i=1
for filename in os.listdir(dire):
    f = os.path.join(dire, filename) # joining the file name with file..
    file1 = open(f,'r')
    txt = file1.read()
    txt_tk = nltk.word_tokenize(txt)
    file1.close()
    if i<6:
        str_tk = ""
        print("\n")
        print("-------------------------"+"file before tokenization preprocessing:" + filename+"-----------------------")
        print(txt)
        print("\n")
        print("***************************"+"file after tokenization preprocessing:" + filename+"***********************\n")
        print(txt_tk)
        print(type(txt_tk))


    #print(txt_tk)
    str_tk  = ' '.join(str(token) for token in txt_tk)
        #print(str_tk)
    file2 = open(f, 'r+')
    file2.truncate(0) # Discarding the text from the file...
    file2.close();
    file3 = open(f,'w')
```

Output:

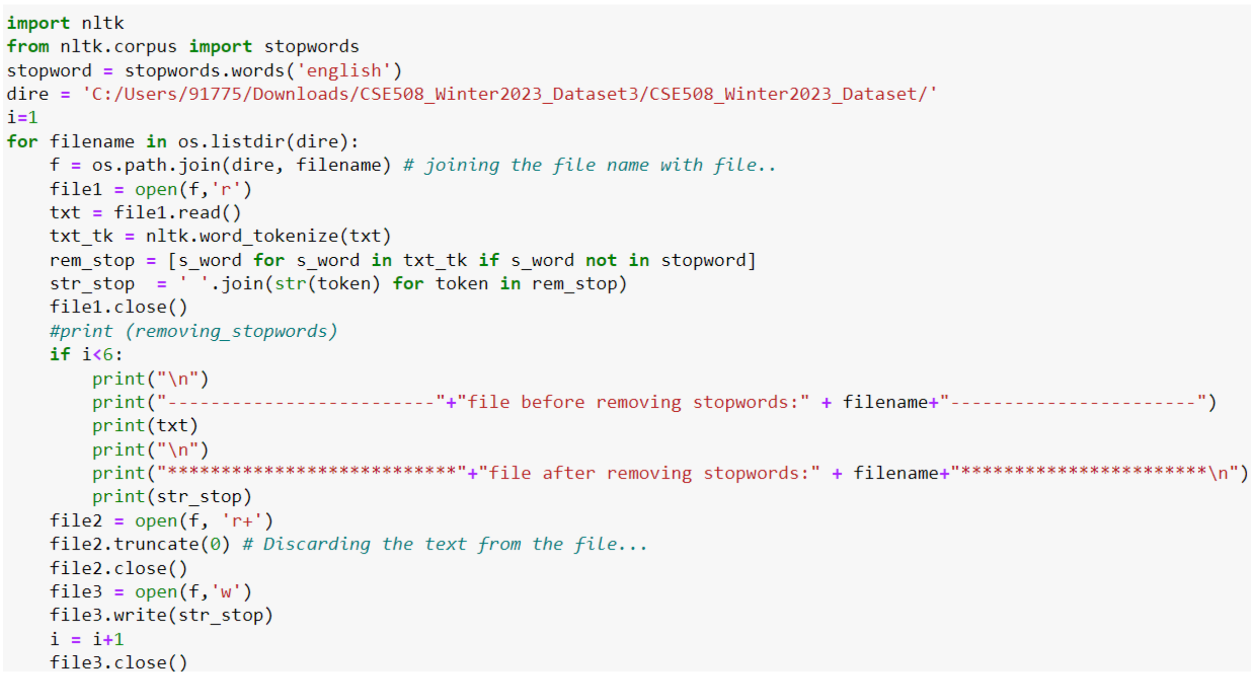


```
-------------------------file before tokenization preprocessing:cranfield0001-----------------------

experimental investigation of the aerodynamics of a
wing in a slipstream .

  an experimental study of a wing in a propeller slipstream was
made in order to determine the spanwise distribution of the lift
increase due to slipstream at different angles of attack of the wing
and at different free stream to slipstream velocity ratios .  the
results were intended in part as an evaluation basis for different
theoretical treatments of this problem .
  the comparative span loading curves, together with supporting
evidence, showed that a substantial part of the lift increment
produced by the slipstream was due to a /destalling/ or boundary-layer-control
effect .  the integrated remaining lift increment,
after subtracting this destalling lift, was found to agree
well with a potential flow theory .
  an empirical evaluation of the destalling effects was made for
the specific configuration of the experiment .


***************************file after tokenization preprocessing:cranfield0001***********************

['experimental', 'investigation', 'of', 'the', 'aerodynamics', 'of', 'a', 'wing', 'in', 'a', 'slipstream', '.', 'an', 'experime
ntal', 'study', 'of', 'a', 'wing', 'in', 'a', 'propeller', 'slipstream', 'was', 'made', 'in', 'order', 'to', 'determine', 'th
e', 'spanwise', 'distribution', 'of', 'the', 'lift', 'increase', 'due', 'to', 'slipstream', 'at', 'different', 'angles', 'of',
'attack', 'of', 'the', 'wing', 'and', 'at', 'different', 'free', 'stream', 'to', 'slipstream', 'velocity', 'ratios', '.', 'th
e', 'results', 'were', 'intended', 'in', 'part', 'as', 'an', 'evaluation', 'basis', 'for', 'different', 'theoretical', 'treatme
nts', 'of', 'this', 'problem', '.', 'the', 'comparative', 'span', 'loading', 'curves', ',', 'together', 'with', 'supporting',
'evidence', ',', 'showed', 'that', 'a', 'substantial', 'part', 'of', 'the', 'lift', 'increment', 'produced', 'by', 'the', 'slip
```

Remove stopwords:

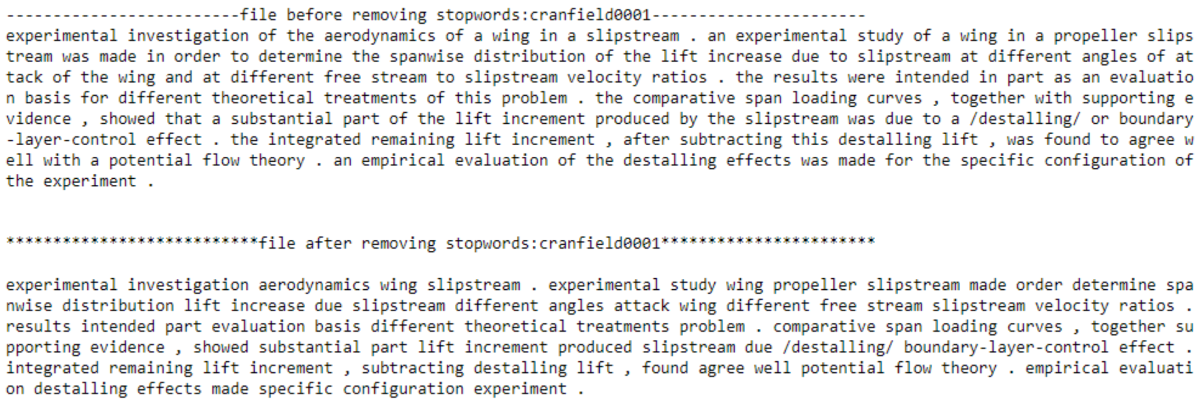
Opened the previously modified file and retrieved the text and remove the sopwords using nltk library an stored it into the string Then all the text is discarded from the opened file and then the empty file is appended with the above string.

Code snippet of the above operation—:



```python
import nltk
from nltk.corpus import stopwords
stopword = stopwords.words('english')
dire = 'C:/Users/91775/Downloads/CSE508_Winter2023_Dataset3/CSE508_Winter2023_Dataset/'
i=1
for filename in os.listdir(dire):
    f = os.path.join(dire, filename) # joining the file name with file..
    file1 = open(f,'r')
    txt = file1.read()
    txt_tk = nltk.word_tokenize(txt)
    rem_stop = [s_word for s_word in txt_tk if s_word not in stopword]
    str_stop  = ' '.join(str(token) for token in rem_stop)
    file1.close()
    #print (removing_stopwords)
    if i<6:
        print("\n")
        print("-------------------------"+"file before removing stopwords:" + filename+"-----------------------")
        print(txt)
        print("\n")
        print("***************************"+"file after removing stopwords:" + filename+"***********************\n")
        print(str_stop)
    file2 = open(f, 'r+')
    file2.truncate(0) # Discarding the text from the file...
    file2.close()
    file3 = open(f,'w')
    file3.write(str_stop)
    i = i+1
    file3.close()
```

Output:

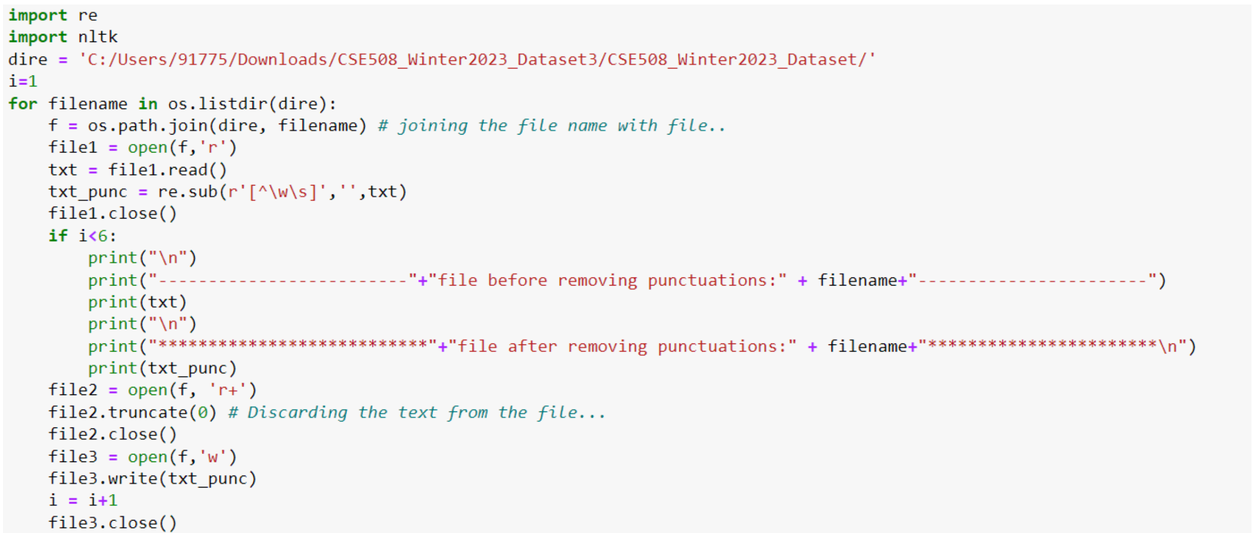


```
-------------------------file before removing stopwords:cranfield0001-----------------------
experimental investigation of the aerodynamics of a wing in a slipstream . an experimental study of a wing in a propeller slips
tream was made in order to determine the spanwise distribution of the lift increase due to slipstream at different angles of at
tack of the wing and at different free stream to slipstream velocity ratios . the results were intended in part as an evaluatio
n basis for different theoretical treatments of this problem . the comparative span loading curves , together with supporting e
vidence , showed that a substantial part of the lift increment produced by the slipstream was due to a /destalling/ or boundary
-layer-control effect . the integrated remaining lift increment , after subtracting this destalling lift , was found to agree w
ell with a potential flow theory . an empirical evaluation of the destalling effects was made for the specific configuration of
the experiment .


***************************file after removing stopwords:cranfield0001***********************

experimental investigation aerodynamics wing slipstream . experimental study wing propeller slipstream made order determine spa
nwise distribution lift increase due slipstream different angles attack wing different free stream slipstream velocity ratios .
results intended part evaluation basis different theoretical treatments problem . comparative span loading curves , together su
pporting evidence , showed substantial part lift increment produced slipstream due /destalling/ boundary-layer-control effect .
integrated remaining lift increment , subtracting destalling lift , found agree well potential flow theory . empirical evaluati
on destalling effects made specific configuration experiment .
```

Remove punctuation:

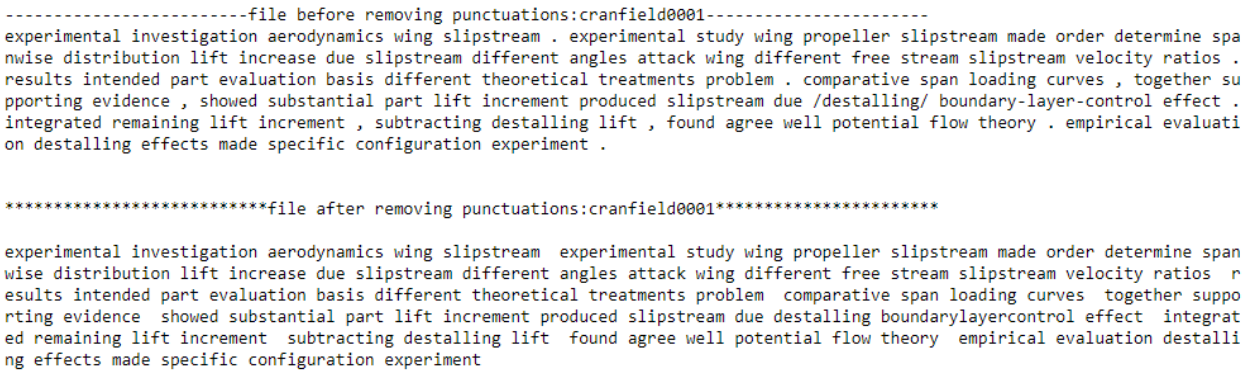
Opened the previously modified file and retrieved the text and remove the puntuation using regex library an stored it into the string, then all the text is discarded from the opened file and then the empty file is appended with the above string.

Code snippet of the above operation—:



```
import re
import nltk
dire = 'C:/Users/91775/Downloads/CSE508_Winter2023_Dataset3/CSE508_Winter2023_Dataset/'
i=1
for filename in os.listdir(dire):
    f = os.path.join(dire, filename) # joining the file name with file..
    file1 = open(f,'r')
    txt = file1.read()
    txt_punc = re.sub(r'[^\w\s]','',txt)
    file1.close()
    if i<6:
        print("\n")
        print("------------------------"+"file before removing punctuations:" + filename+"----------------------")
        print(txt)
        print("\n")
        print("***************************"+"file after removing punctuations:" + filename+"***********************\n")
        print(txt_punc)
    file2 = open(f, 'r+')
    file2.truncate(0) # Discarding the text from the file...
    file2.close()
    file3 = open(f,'w')
    file3.write(txt_punc)
    i = i+1
    file3.close()
```

Output:

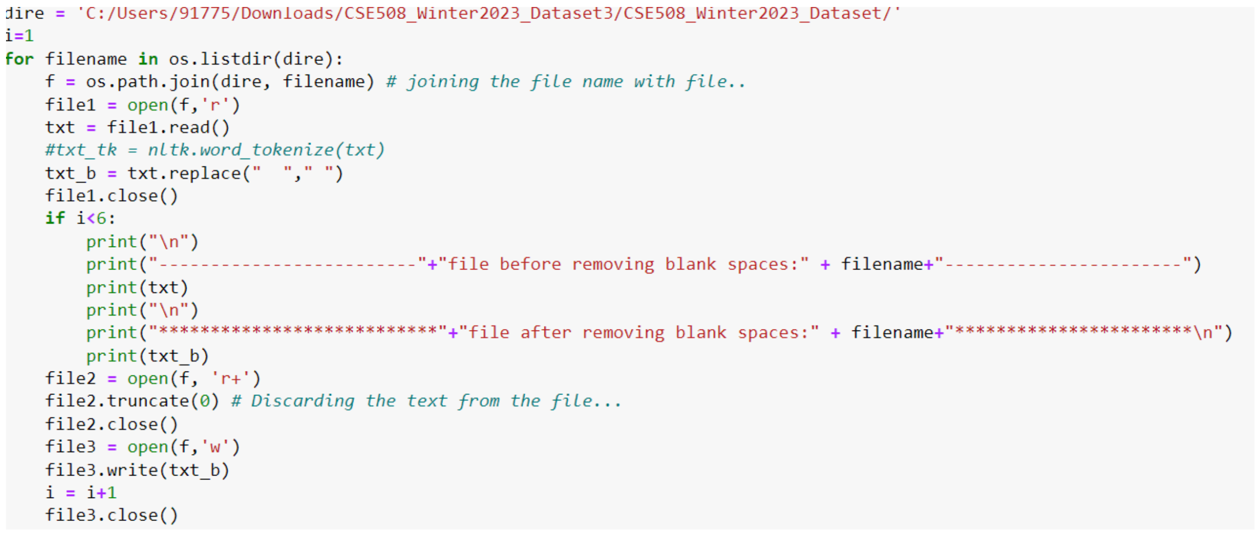


```
-------------------------file before removing punctuations:cranfield0001-----------------------
experimental investigation aerodynamics wing slipstream . experimental study wing propeller slipstream made order determine spa
nwise distribution lift increase due slipstream different angles attack wing different free stream slipstream velocity ratios .
results intended part evaluation basis different theoretical treatments problem . comparative span loading curves , together su
pporting evidence , showed substantial part lift increment produced slipstream due /destalling/ boundary-layer-control effect .
integrated remaining lift increment , subtracting destalling lift , found agree well potential flow theory . empirical evaluati
on destalling effects made specific configuration experiment .


***************************file after removing punctuations:cranfield0001***********************

experimental investigation aerodynamics wing slipstream  experimental study wing propeller slipstream made order determine span
wise distribution lift increase due slipstream different angles attack wing different free stream slipstream velocity ratios  r
esults intended part evaluation basis different theoretical treatments problem  comparative span loading curves  together suppo
rting evidence  showed substantial part lift increment produced slipstream due destalling boundarylayercontrol effect  integrat
ed remaining lift increment  subtracting destalling lift  found agree well potential flow theory  empirical evaluation destalli
ng effects made specific configuration experiment
```

Remove blank space tokens:

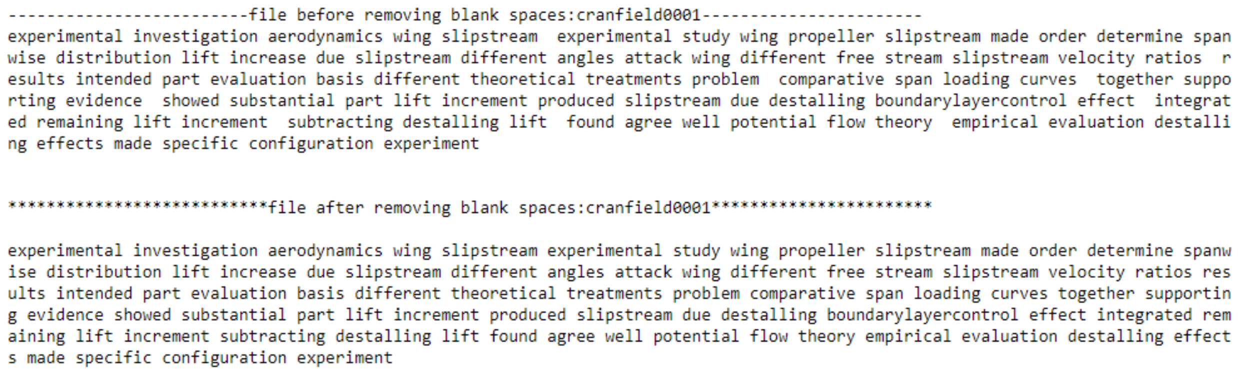
Opened the previously modified file and retrieved the text and remove the blank space tokens using replace function and stored it into the string, then all the text is discarded from the opened file and then the empty file is appended with the above string.

Code snippet of the above operation—:



```python
dire = 'C:/Users/91775/Downloads/CSE508_Winter2023_Dataset3/CSE508_Winter2023_Dataset/'
i=1
for filename in os.listdir(dire):
    f = os.path.join(dire, filename) # joining the file name with file..
    file1 = open(f,'r')
    txt = file1.read()
    #txt_tk = nltk.word_tokenize(txt)
    txt_b = txt.replace("  "," ")
    file1.close()
    if i<6:
        print("\n")
        print("-------------------------"+"file before removing blank spaces:" + filename+"-----------------------")
        print(txt)
        print("\n")
        print("***************************"+"file after removing blank spaces:" + filename+"***********************\n")
        print(txt_b)
    file2 = open(f, 'r+')
    file2.truncate(0) # Discarding the text from the file...
    file2.close()
    file3 = open(f,'w')
    file3.write(txt_b)
    i = i+1
    file3.close()
```

Output:

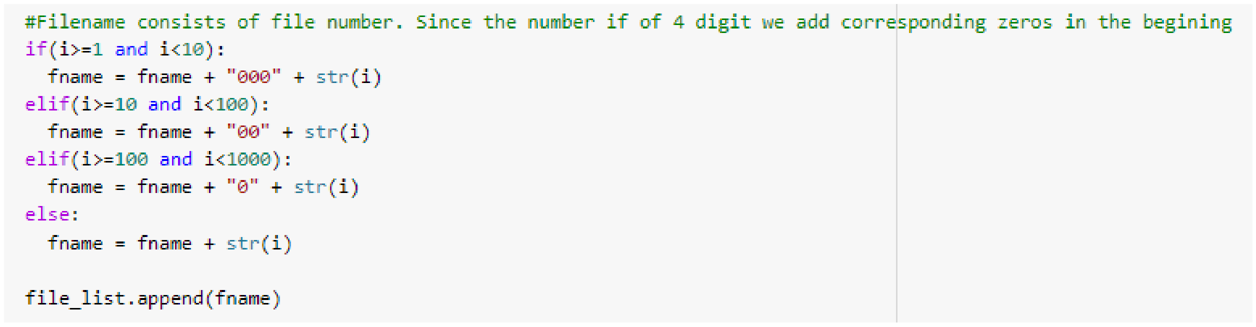


```
-------------------------file before removing blank spaces:cranfield0001-----------------------
experimental investigation aerodynamics wing slipstream  experimental study wing propeller slipstream made order determine span
wise distribution lift increase due slipstream different angles attack wing different free stream slipstream velocity ratios  r
esults intended part evaluation basis different theoretical treatments problem  comparative span loading curves  together suppo
rting evidence  showed substantial part lift increment produced slipstream due destalling boundarylayercontrol effect  integrat
ed remaining lift increment  subtracting destalling lift  found agree well potential flow theory  empirical evaluation destalli
ng effects made specific configuration experiment


***************************file after removing blank spaces:cranfield0001***********************

experimental investigation aerodynamics wing slipstream experimental study wing propeller slipstream made order determine spanw
ise distribution lift increase due slipstream different angles attack wing different free stream slipstream velocity ratios res
ults intended part evaluation basis different theoretical treatments problem comparative span loading curves together supportin
g evidence showed substantial part lift increment produced slipstream due destalling boundarylayercontrol effect integrated rem
aining lift increment subtracting destalling lift found agree well potential flow theory empirical evaluation destalling effect
s made specific configuration experiment
```

Question 2

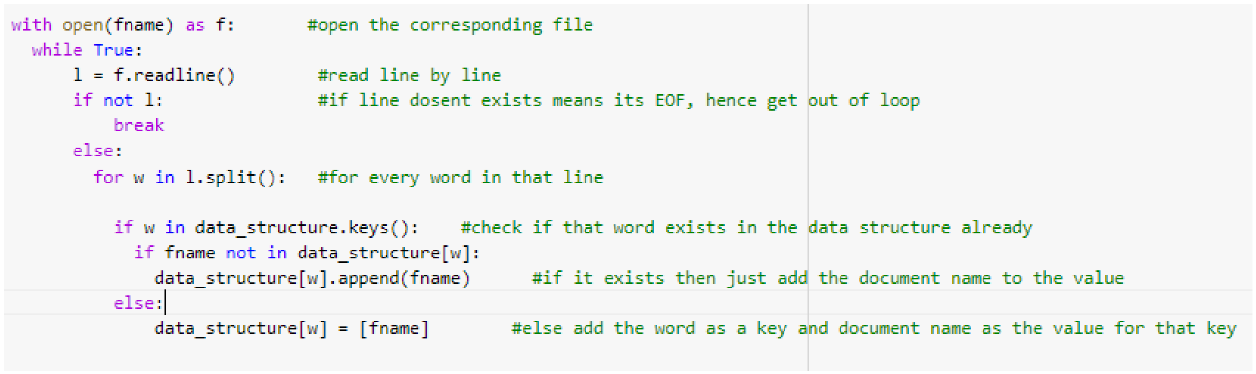
1. **Create a unigram inverted index(from scratch; No library allowed) of the dataset obtained from Q1 (after preprocessing).**

We are using dictionary to store the data structure created for unigram inverted index. To generate the data structure we use the preprocessed data from question1. Since the data will be the form of key value pairs and value will be filenames, we have used if else to generate the filenames.



```
#Filename consists of file number. Since the number if of 4 digit we add corresponding zeros in the begining
if(i>=1 and i<10):
  fname = fname + "000" + str(i)
elif(i>=10 and i<100):
  fname = fname + "00" + str(i)
elif(i>=100 and i<1000):
  fname = fname + "0" + str(i)
else:
  fname = fname + str(i)

file_list.append(fname)
```

We are appending the filename to a list “file_list” for future use.
We open the file and read line by line till we read to EOF. for every word we check if it exists in the data structure already or not. If it already exists we append the fle name to the list of values. If the word is not already present in the data structure then we create a list and add the filename to it.

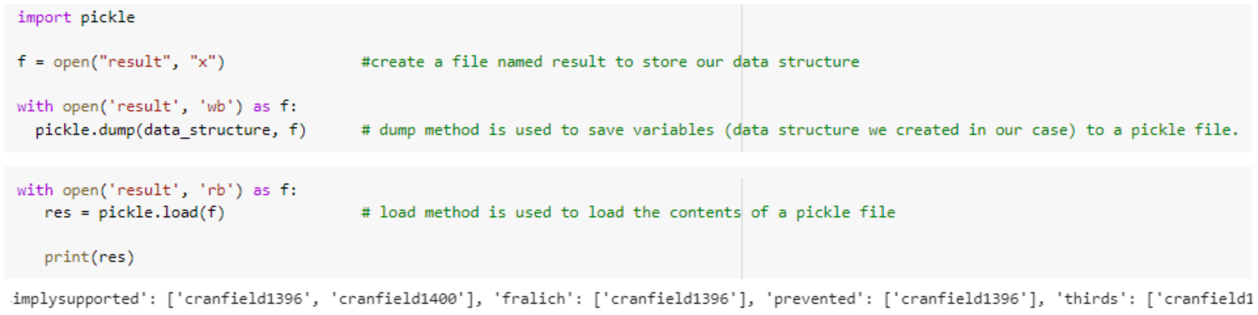


```
with open(fname) as f:       #open the corresponding file
  while True:
      l = f.readline()        #read line by line
      if not l:               #if line dosent exists means its EOF, hence get out of loop
          break
      else:
        for w in l.split():   #for every word in that line

          if w in data_structure.keys():    #check if that word exists in the data structure already
            if fname not in data_structure[w]:
              data_structure[w].append(fname)      #if it exists then just add the document name to the value
          else:
              data_structure[w] = [fname]       #else add the word as a key and document name as the value for that key
```

With this our unigram inverted index data structure is created

2. **Use Python's pickle module to save and load the unigram inverted index.**

We have used pickle libraries dump and load method to save and load the inverted index.



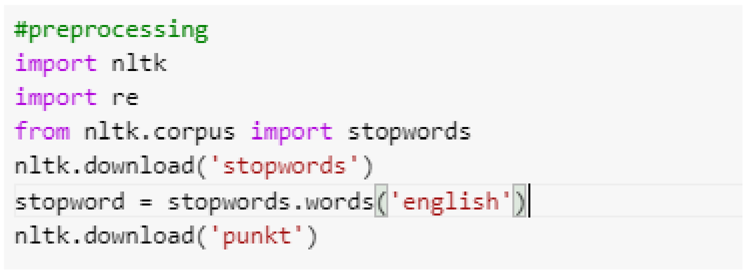
```
import pickle

f = open("result", "x")              #create a file named result to store our data structure

with open('result', 'wb') as f:
  pickle.dump(data_structure, f)      # dump method is used to save variables (data structure we created in our case) to a pickle file.
```

```
with open('result', 'rb') as f:
   res = pickle.load(f)               # load method is used to load the contents of a pickle file

   print(res)
```

```
implysupported': ['cranfield1396', 'cranfield1400'], 'fralich': ['cranfield1396'], 'prevented': ['cranfield1396'], 'thirds': ['cranfield1
```

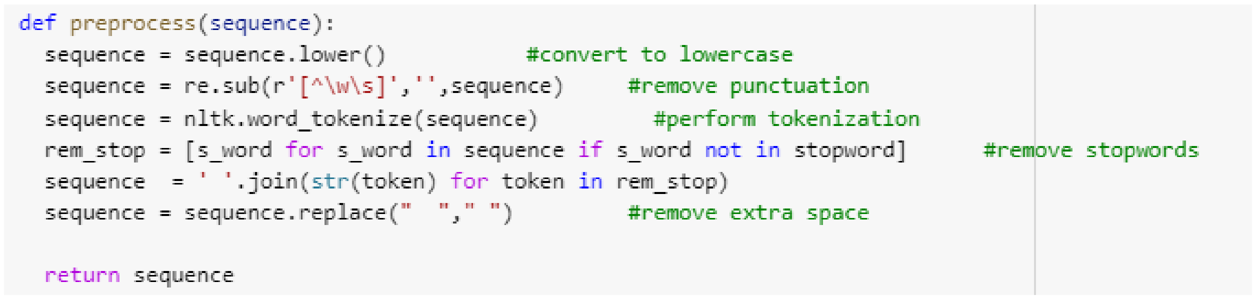
3. **Provide support for the following operations: AND, OR, AND NOT, OR NOT**

Used following libraries for preprocessing



```
#preprocessing
import nltk
import re
from nltk.corpus import stopwords
nltk.download('stopwords')
stopword = stopwords.words('english')
nltk.download('punkt')
```

We have created a preprocess function to perform preprocessing over the input data provided by the user for query purposes. All the preprocessing done in question 1 is also done on input data.



```
def preprocess(sequence):
  sequence = sequence.lower()           #convert to lowercase
  sequence = re.sub(r'[^\w\s]','',sequence)     #remove punctuation
  sequence = nltk.word_tokenize(sequence)         #perform tokenization
  rem_stop = [s_word for s_word in sequence if s_word not in stopword]      #remove stopwords
  sequence  = ' '.join(str(token) for token in rem_stop)
  sequence = sequence.replace("  "," ")         #remove extra space

  return sequence
```

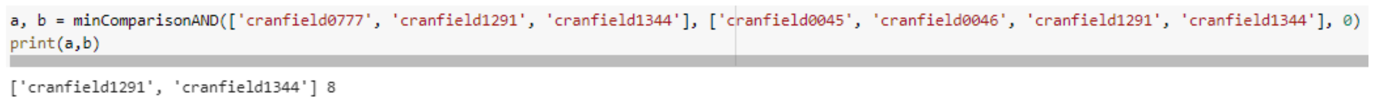
AND operation:

We have kept smaller list in l2 so that it is easier to manage. Below is the main logic for every element in list l1

a. if element from l1 = element from l2 ====> then add that element in final result and increment the count by 1, inc pointer to l2
b. if element from l1 > element from l2 ====> then inc pointer to l2 and go to next element
    i. but we need current element of list l1 again so dec its pointer coz it will be incremented in the next iteration
    ii. incremenet count by 2 since this is 2nd comparsion

   Note: since lists are sorted if we go accross element which is greater than current then current element wont occur in the list and we can now move ahead
c. if above conditions fail means element from l1 < element from l and we can just ignore. Inc count by 2 since we have already done 2 comparsions

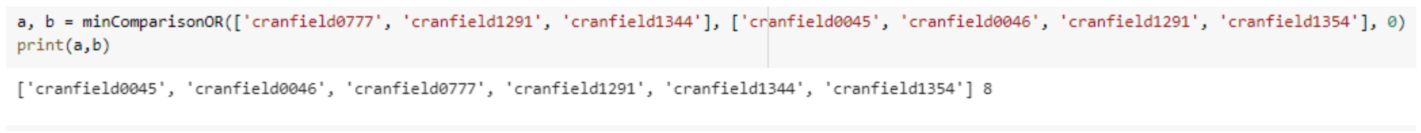
Sample Result



```
a, b = minComparisonAND(['cranfield0777', 'cranfield1291', 'cranfield1344'], ['cranfield0045', 'cranfield0046', 'cranfield1291', 'cranfield1344'], 0)
print(a,b)

['cranfield1291', 'cranfield1344'] 8
```

OR operation:
We have used simple and standard approach to merge 2 sorted lists. Its the same that we use in the merge procedure of merge sort. With every comparison made we increase the count variable that keeps track of minimum number of comparisons

Sample Result



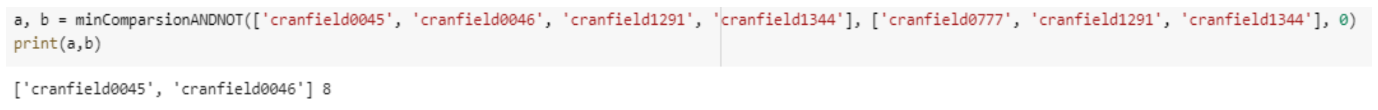
```
a, b = minComparisonOR(['cranfield0777', 'cranfield1291', 'cranfield1344'], ['cranfield0045', 'cranfield0046', 'cranfield1291', 'cranfield1354'], 0)
print(a,b)

['cranfield0045', 'cranfield0046', 'cranfield0777', 'cranfield1291', 'cranfield1344', 'cranfield1354'] 8
```

AND NOT operation:
run untill we reach to the end of any one of the list

a. if element of l2 > element of l1 ===> append element of l2 in the result
b. if element of l2 < element of l1 ===> ignore l2 element
c. if element of l2 = element of l1 ===> ignore both l1 and l2 element

append remaining elements of list l1 to the result

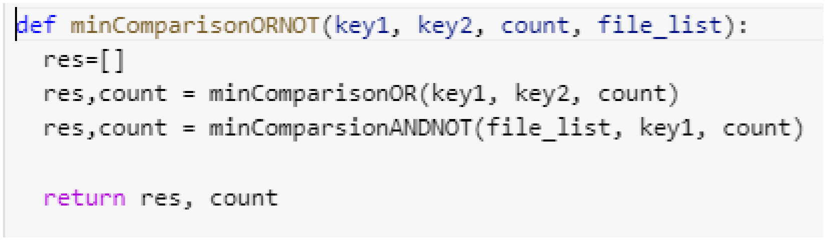
Sample Result:



```
a, b = minComparsionANDNOT(['cranfield0045', 'cranfield0046', 'cranfield1291', 'cranfield1344'], ['cranfield0777', 'cranfield1291', 'cranfield1344'], 0)
print(a,b)

['cranfield0045', 'cranfield0046'] 8
```

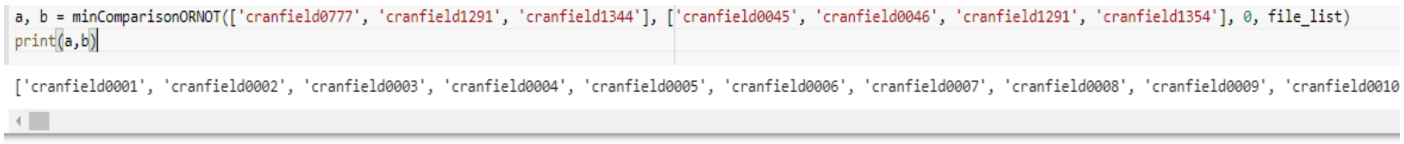
OR NOT operation:

a. perform OR between 2 keys
b. perform AND NOT between universal list of all file names and result obtained from previous step



```
def minComparisonORNOT(key1, key2, count, file_list):
  res=[]
  res,count = minComparisonOR(key1, key2, count)
  res,count = minComparsionANDNOT(file_list, key1, count)

  return res, count
```

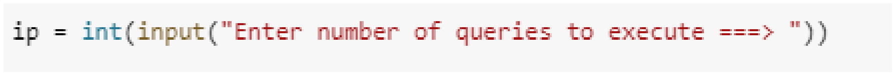
Sample Result:



```
a, b = minComparisonORNOT(['cranfield0777', 'cranfield1291', 'cranfield1344'], ['cranfield0045', 'cranfield0046', 'cranfield1291', 'cranfield1354'], 0, file_list)
print(a,b)

['cranfield0001', 'cranfield0002', 'cranfield0003', 'cranfield0004', 'cranfield0005', 'cranfield0006', 'cranfield0007', 'cranfield0008', 'cranfield0009', 'cranfield0010
```

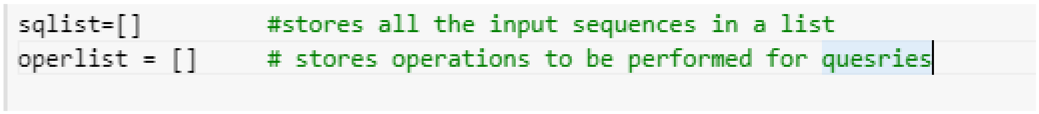
Input and Output process:

a. take number of queries as input



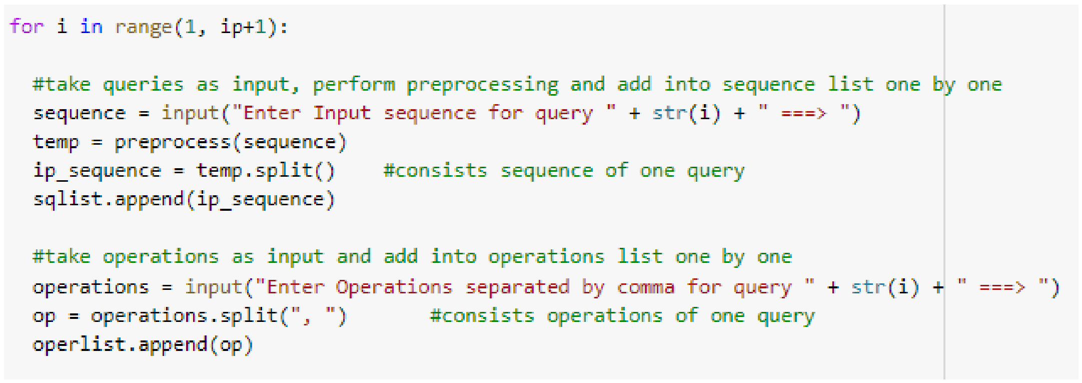
```
ip = int(input("Enter number of queries to execute ===> "))
```

b. Store ip sequences and operations in different lists



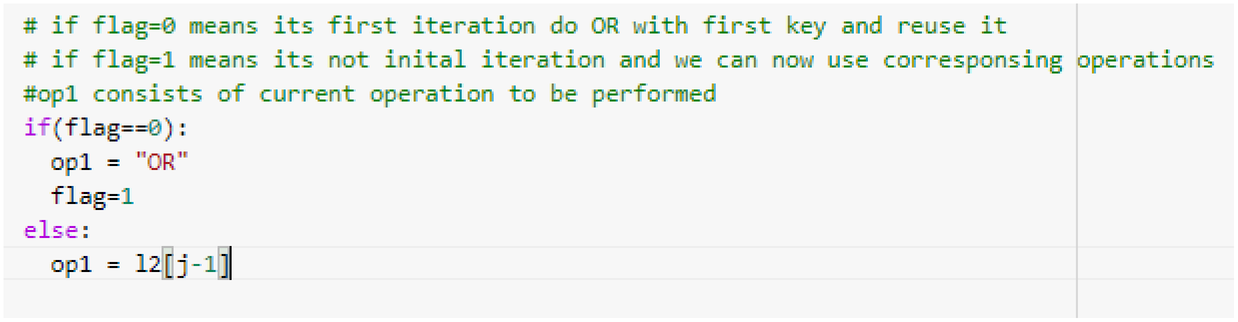
```
sqlist=[]         #stores all the input sequences in a list
operlist = []     # stores operations to be performed for quesries
```

c. For every query
    i. Take input sequence from user
    ii. Perform preprocessing
    iii. Split remaining sequence word by word and add to into sequence list.
    iv. Take operations as input from the user and split every operation into the list



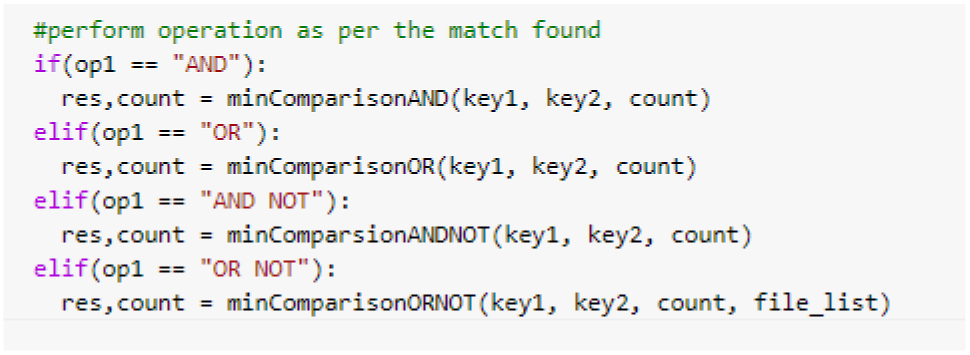
```
for i in range(1, ip+1):

  #take queries as input, perform preprocessing and add into sequence list one by one
  sequence = input("Enter Input sequence for query " + str(i) + " ===> ")
  temp = preprocess(sequence)
  ip_sequence = temp.split()    #consists sequence of one query
  sqlist.append(ip_sequence)

  #take operations as input and add into operations list one by one
  operations = input("Enter Operations separated by comma for query " + str(i) + " ===> ")
  op = operations.split(", ")      #consists operations of one query
  operlist.append(op)
```

d. For every user input sequence and operation list we perform respective operations. At initial step we use a flag so that we can reuse the result obtained at every step.



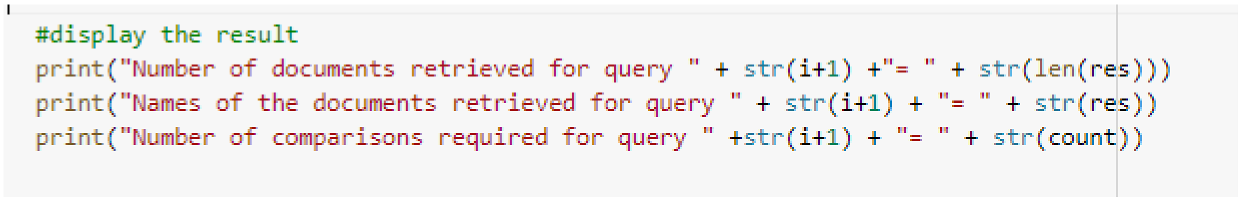
```
# if flag=0 means its first iteration do OR with first key and reuse it
# if flag=1 means its not inital iteration and we can now use corresponsing operations
#op1 consists of current operation to be performed
if(flag==0):
  op1 = "OR"
  flag=1
else:
  op1 = l2[j-1]
```

e. Perform operations on input sequence



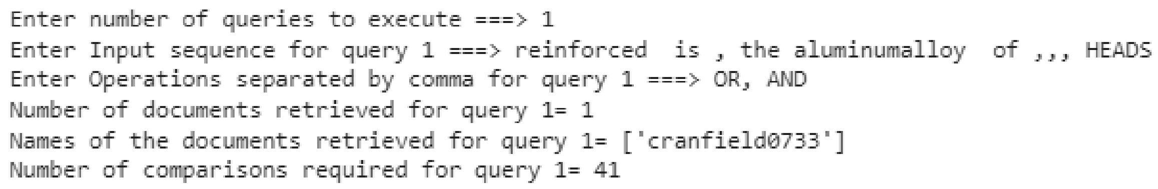
```
#perform operation as per the match found
if(op1 == "AND"):
  res,count = minComparisonAND(key1, key2, count)
elif(op1 == "OR"):
  res,count = minComparisonOR(key1, key2, count)
elif(op1 == "AND NOT"):
  res,count = minComparsionANDNOT(key1, key2, count)
elif(op1 == "OR NOT"):
  res,count = minComparisonORNOT(key1, key2, count, file_list)
```

f. Display the result



```
#display the result
print("Number of documents retrieved for query " + str(i+1) +"= " + str(len(res)))
print("Names of the documents retrieved for query " + str(i+1) + "= " + str(res))
print("Number of comparisons required for query " +str(i+1) + "= " + str(count))
```

g. Final outcome



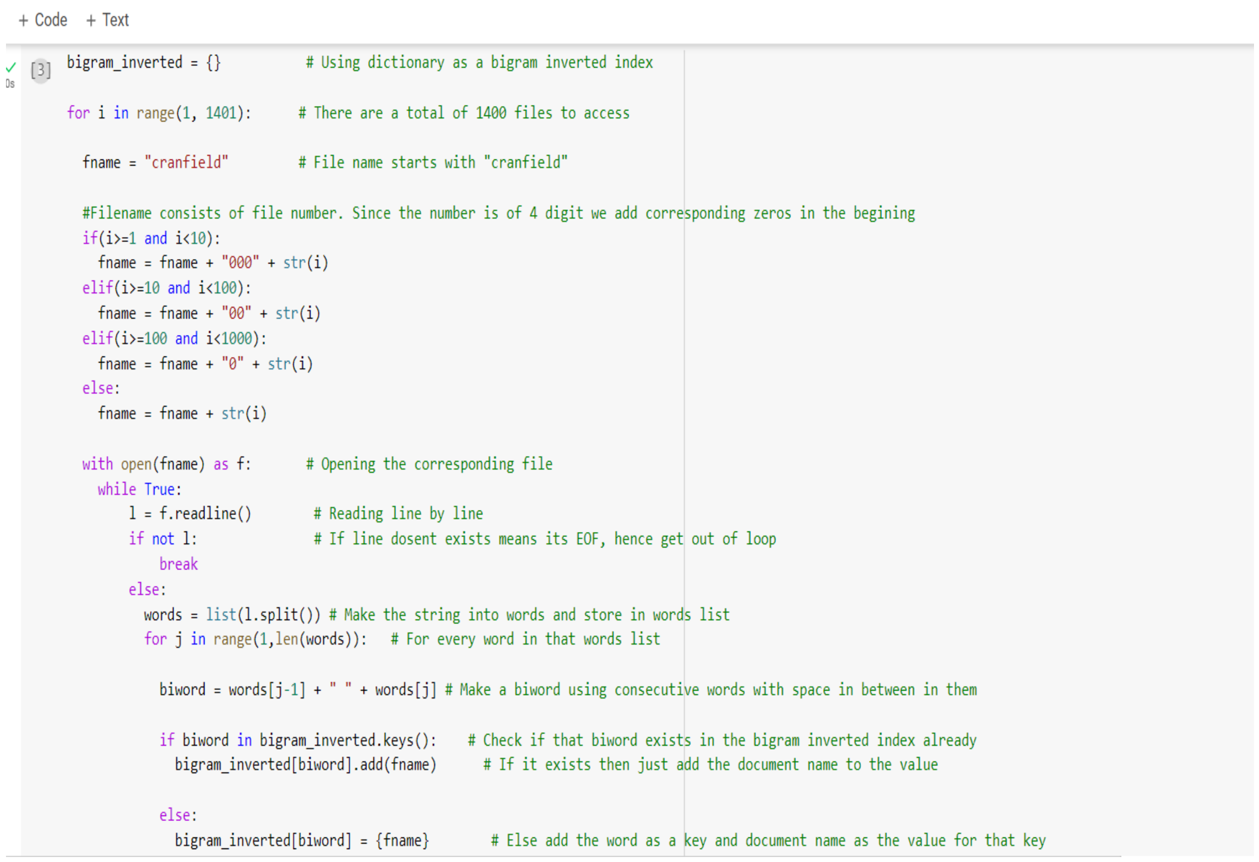
```
Enter number of queries to execute ===> 1
Enter Input sequence for query 1 ===> reinforced  is , the aluminumalloy  of ,,, HEADS
Enter Operations separated by comma for query 1 ===> OR, AND
Number of documents retrieved for query 1= 1
Names of the documents retrieved for query 1= ['cranfield0733']
Number of comparisons required for query 1= 41
```

Question 3

Note :
The dataset used in this question is from Question1, where the whole dataset is preprocessed.
Preprocessing is done in the order which is mentioned in the Question1.

(i) Bigram inverted index

1. Create a bigram inverted index (from scratch; No library allowed) of the dataset obtained from Q1. 2. Use Python's pickle module to save and load the bigram inverted index.

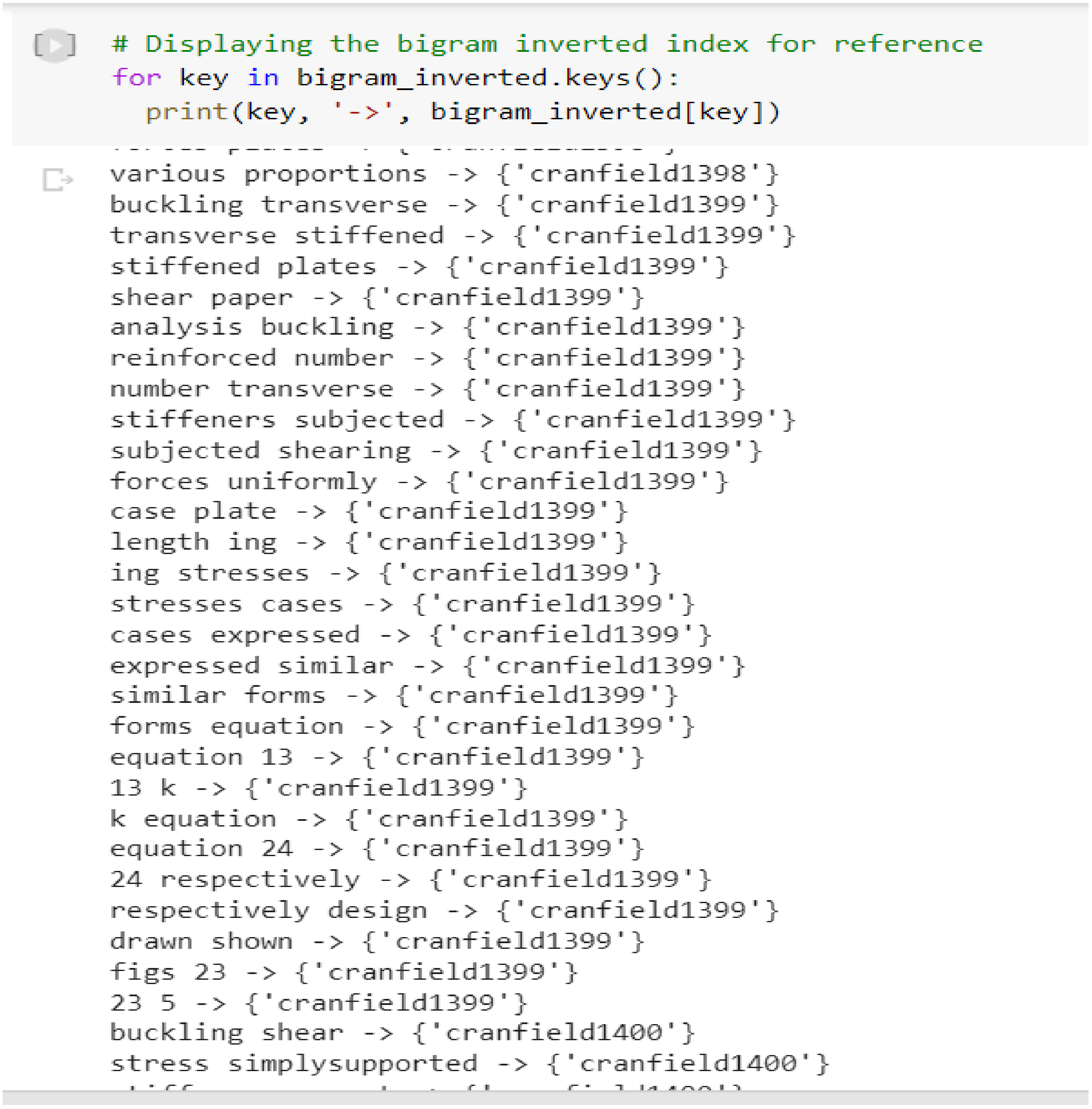


```
bigram_inverted = {}          # Using dictionary as a bigram inverted index

for i in range(1, 1401):      # There are a total of 1400 files to access

  fname = "cranfield"         # File name starts with "cranfield"

  #Filename consists of file number. Since the number is of 4 digit we add corresponding zeros in the begining
  if(i>=1 and i<10):
    fname = fname + "000" + str(i)
  elif(i>=10 and i<100):
    fname = fname + "00" + str(i)
  elif(i>=100 and i<1000):
    fname = fname + "0" + str(i)
  else:
    fname = fname + str(i)

  with open(fname) as f:       # Opening the corresponding file
    while True:
        l = f.readline()        # Reading line by line
        if not l:               # If line dosent exists means its EOF, hence get out of loop
            break
        else:
          words = list(l.split()) # Make the string into words and store in words list
          for j in range(1,len(words)):   # For every word in that words list

            biword = words[j-1] + " " + words[j] # Make a biword using consecutive words with space in between in them

            if biword in bigram_inverted.keys():    # Check if that biword exists in the bigram inverted index already
              bigram_inverted[biword].add(fname)      # If it exists then just add the document name to the value

            else:
              bigram_inverted[biword] = {fname}       # Else add the word as a key and document name as the value for that key
```

Bigram_inverted is the dictionary where it contains the bigram inverted index.
The dataset is loaded into the google colab locally. The loop runs for 1400 times as the dataset contains 1400 files. All the file names start with common prefix "cranfield," so I have taken that prefix in the string fname. Remaining file name is added according to the loop index. Now open the file and iterate through it. Read line by line. If the file

reaches to EOF, then stop reading the file. Otherwise, split the string and store in list words. Iterate the words list and make the biword by combining the consecutive words with space in between them. Now check the biword is there in the bigram_inverted dictionary or not. If not, add the biword as key and set as value where the set contains the fname. If yes, add the fname into the set as a value to the corresponding biword.

Displaying the bigram inverted index -



```python
# Displaying the bigram inverted index for reference
for key in bigram_inverted.keys():
  print(key, '->', bigram_inverted[key])
```

```
various proportions -> {'cranfield1398'}
buckling transverse -> {'cranfield1399'}
transverse stiffened -> {'cranfield1399'}
stiffened plates -> {'cranfield1399'}
shear paper -> {'cranfield1399'}
analysis buckling -> {'cranfield1399'}
reinforced number -> {'cranfield1399'}
number transverse -> {'cranfield1399'}
stiffeners subjected -> {'cranfield1399'}
subjected shearing -> {'cranfield1399'}
forces uniformly -> {'cranfield1399'}
case plate -> {'cranfield1399'}
length ing -> {'cranfield1399'}
ing stresses -> {'cranfield1399'}
stresses cases -> {'cranfield1399'}
cases expressed -> {'cranfield1399'}
expressed similar -> {'cranfield1399'}
similar forms -> {'cranfield1399'}
forms equation -> {'cranfield1399'}
equation 13 -> {'cranfield1399'}
13 k -> {'cranfield1399'}
k equation -> {'cranfield1399'}
equation 24 -> {'cranfield1399'}
24 respectively -> {'cranfield1399'}
respectively design -> {'cranfield1399'}
drawn shown -> {'cranfield1399'}
figs 23 -> {'cranfield1399'}
23 5 -> {'cranfield1399'}
buckling shear -> {'cranfield1400'}
stress simplysupported -> {'cranfield1400'}
```

2. Use Python’s pickle module to save and load the bigram inverted index.

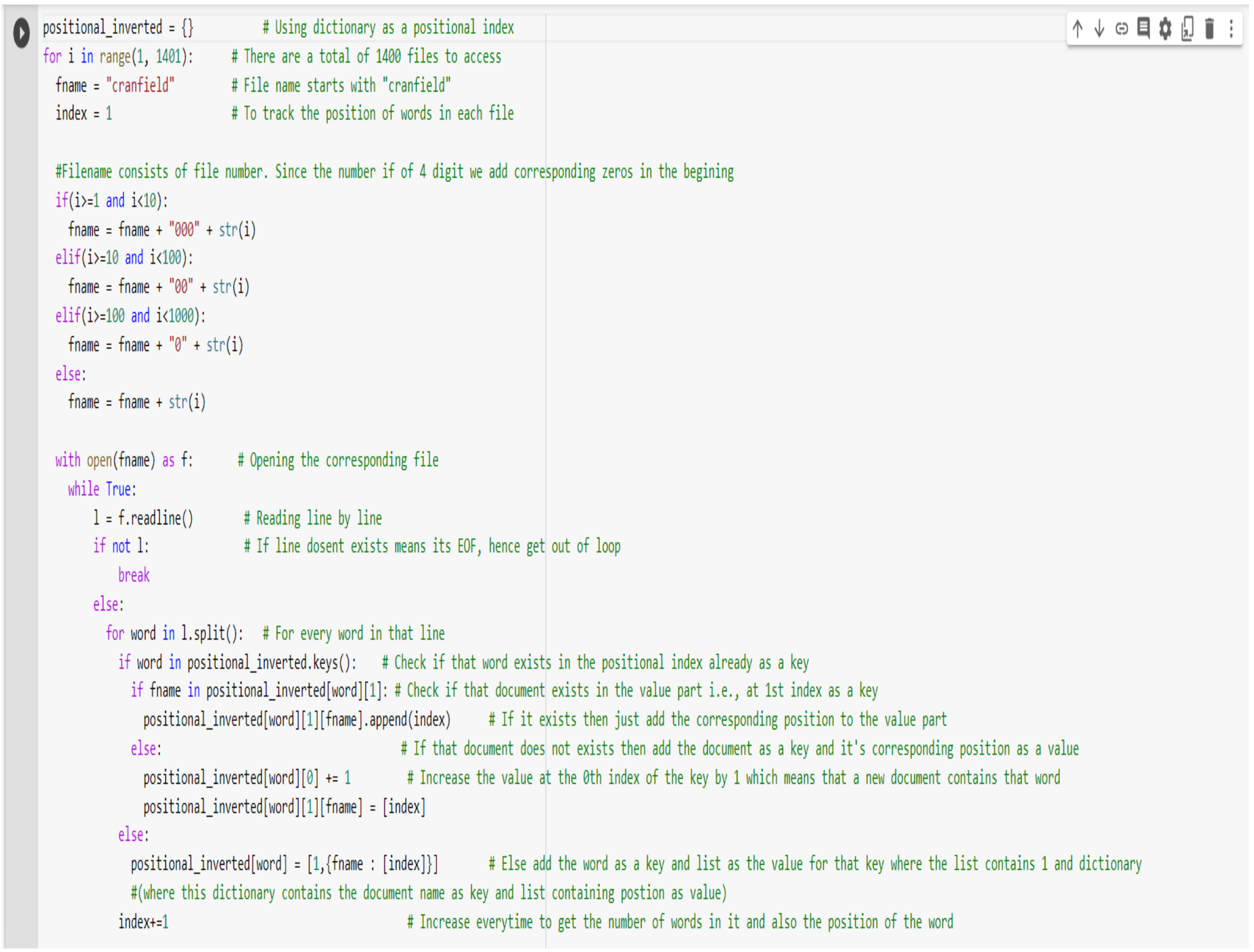
```
[ ]  # Saving the bigram inverted index for further usage
     pickle.dump(bigram_inverted, open('bigram_inverted.pkl','wb'))
```

```
[ ]  # Loading the saved bigram inverted index
     bigram_inverted_unpickled = pickle.load(open('bigram_inverted.pkl','rb'))
```

Saving the bigram inverted index using pickle module for future usage.

(ii) Positional index

1. Create a positional index (from scratch; No library allowed) of the dataset obtained from Q1.

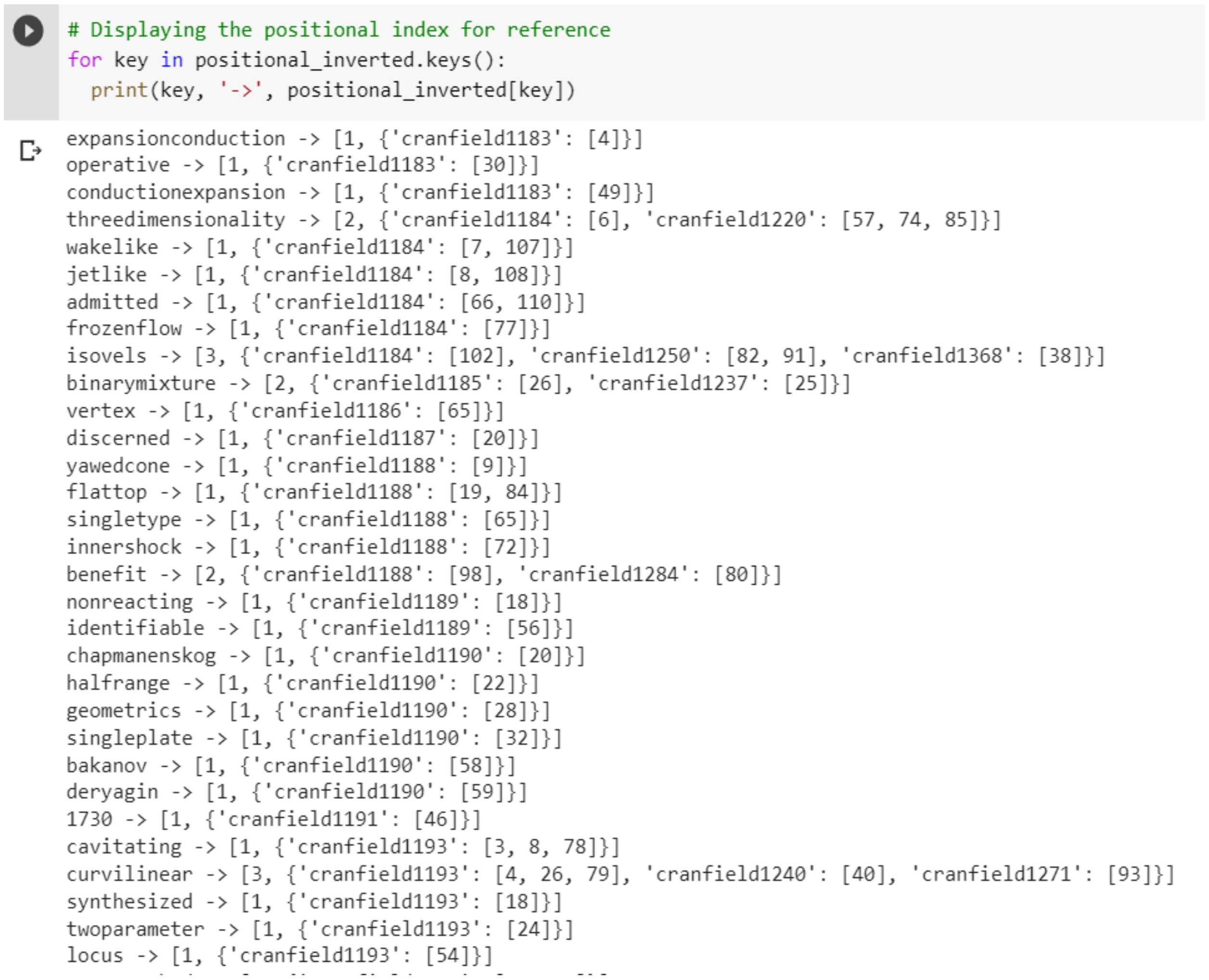


```
positional_inverted = {}          # Using dictionary as a positional index
for i in range(1, 1401):          # There are a total of 1400 files to access
  fname = "cranfield"             # File name starts with "cranfield"
  index = 1                       # To track the position of words in each file

  #Filename consists of file number. Since the number if of 4 digit we add corresponding zeros in the begining
  if(i>=1 and i<10):
    fname = fname + "000" + str(i)
  elif(i>=10 and i<100):
    fname = fname + "00" + str(i)
  elif(i>=100 and i<1000):
    fname = fname + "0" + str(i)
  else:
    fname = fname + str(i)

  with open(fname) as f:          # Opening the corresponding file
    while True:
      l = f.readline()            # Reading line by line
      if not l:                   # If line dosent exists means its EOF, hence get out of loop
        break
      else:
        for word in l.split():    # For every word in that line
          if word in positional_inverted.keys():    # Check if that word exists in the positional index already as a key
            if fname in positional_inverted[word][1]: # Check if that document exists in the value part i.e., at 1st index as a key
              positional_inverted[word][1][fname].append(index)      # If it exists then just add the corresponding position to the value part
            else:                                       # If that document does not exists then add the document as a key and it's corresponding position as a value
              positional_inverted[word][0] += 1         # Increase the value at the 0th index of the key by 1 which means that a new document contains that word
              positional_inverted[word][1][fname] = [index]
          else:
            positional_inverted[word] = [1,{fname : [index]}]       # Else add the word as a key and list as the value for that key where the list contains 1 and dictionary
            #(where this dictionary contains the document name as key and list containing postion as value)
          index+=1                                      # Increase everytime to get the number of words in it and also the position of the word
```

Positional_inverted is the dictionary where it contains the positional index.

The dataset is loaded into the google colab locally. The loop runs for 1400 times as the dataset contains 1400 files. All the file names start with common prefix “cranfield,” so I have taken that prefix in the string fname. Remaining file name is added according to the loop index. Now open the file and iterate through it. Read line by line. If the file reaches to EOF, then stop reading the file. Otherwise, split the string and store in list words. Iterate the words list and check that word is there in the positional_inverted dictionary or not. If not, add the word as key and list as value where the list contains number at 0th index (which indicates the number of unique documents that contain that word) and dictionary at 1st index (where dictionary contains fname as key and positions of the word in the file which will be in list as value) . If yes, then check that fname is there in the dictionary as a key. If yes, add the word's position into the list as a value tot he corresponding fname. If no, increment the value at 0th index of the list by one and add the fname as key and words position into list as a value to the dictionary at 1st index.

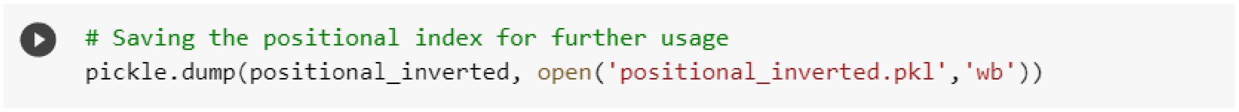
Displaying the positional index -



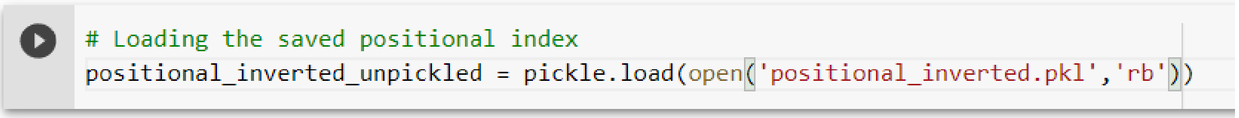
```
# Displaying the positional index for reference
for key in positional_inverted.keys():
  print(key, '->', positional_inverted[key])
```

```
expansionconduction -> [1, {'cranfield1183': [4]}]
operative -> [1, {'cranfield1183': [30]}]
conductionexpansion -> [1, {'cranfield1183': [49]}]
threedimensionality -> [2, {'cranfield1184': [6], 'cranfield1220': [57, 74, 85]}]
wakelike -> [1, {'cranfield1184': [7, 107]}]
jetlike -> [1, {'cranfield1184': [8, 108]}]
admitted -> [1, {'cranfield1184': [66, 110]}]
frozenflow -> [1, {'cranfield1184': [77]}]
isovels -> [3, {'cranfield1184': [102], 'cranfield1250': [82, 91], 'cranfield1368': [38]}]
binarymixture -> [2, {'cranfield1185': [26], 'cranfield1237': [25]}]
vertex -> [1, {'cranfield1186': [65]}]
discerned -> [1, {'cranfield1187': [20]}]
yawedcone -> [1, {'cranfield1188': [9]}]
flattop -> [1, {'cranfield1188': [19, 84]}]
singletype -> [1, {'cranfield1188': [65]}]
innershock -> [1, {'cranfield1188': [72]}]
benefit -> [2, {'cranfield1188': [98], 'cranfield1284': [80]}]
nonreacting -> [1, {'cranfield1189': [18]}]
identifiable -> [1, {'cranfield1189': [56]}]
chapmanenskog -> [1, {'cranfield1190': [20]}]
halfrange -> [1, {'cranfield1190': [22]}]
geometrics -> [1, {'cranfield1190': [28]}]
singleplate -> [1, {'cranfield1190': [32]}]
bakanov -> [1, {'cranfield1190': [58]}]
deryagin -> [1, {'cranfield1190': [59]}]
1730 -> [1, {'cranfield1191': [46]}]
cavitating -> [1, {'cranfield1193': [3, 8, 78]}]
curvilinear -> [3, {'cranfield1193': [4, 26, 79], 'cranfield1240': [40], 'cranfield1271': [93]}]
synthesized -> [1, {'cranfield1193': [18]}]
twoparameter -> [1, {'cranfield1193': [24]}]
locus -> [1, {'cranfield1193': [54]}]
```

2. Use Python's pickle module to save and load the positional index.



```
# Saving the positional index for further usage
pickle.dump(positional_inverted, open('positional_inverted.pkl','wb'))
```



```
# Loading the saved positional index
positional_inverted_unpickled = pickle.load(open('positional_inverted.pkl','rb'))
```

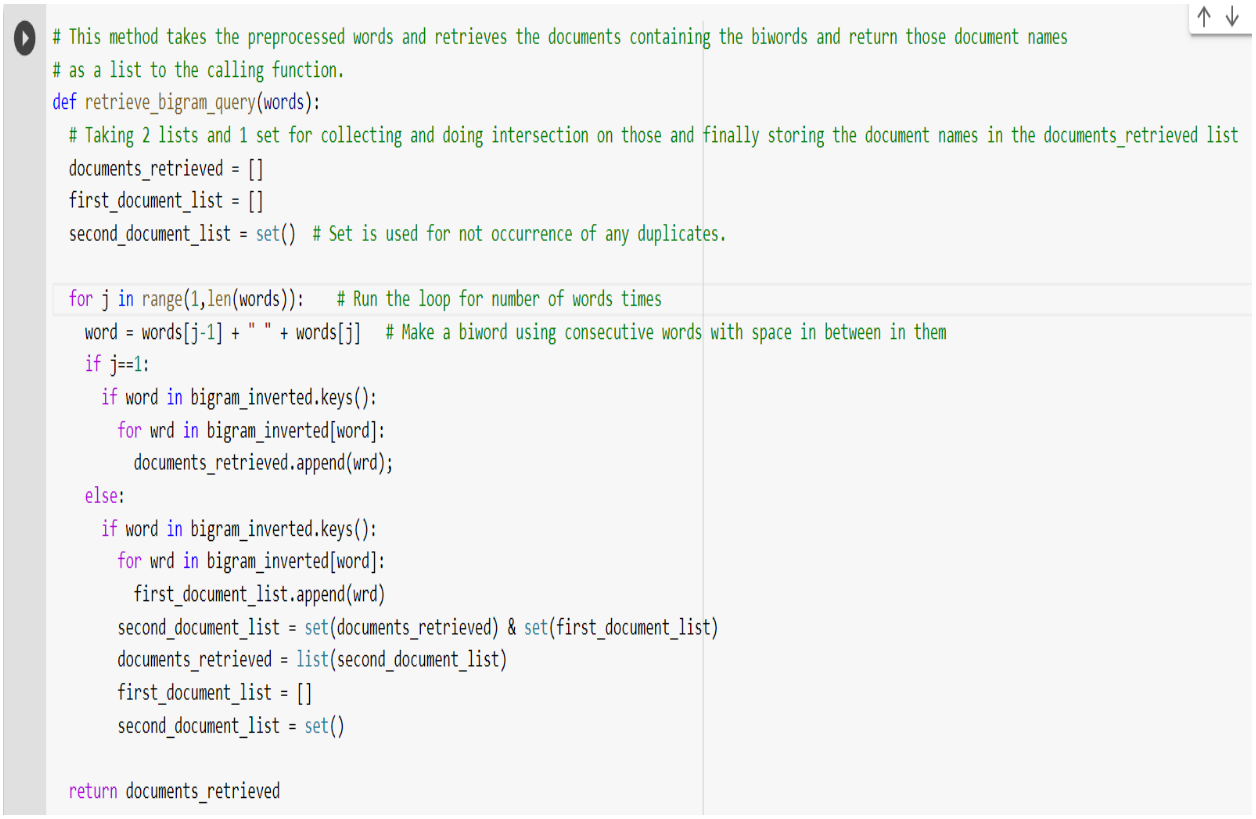
Saving the positional index using pickle module for future usage.

(iii) Compare and comment on your results using (i) and (ii)

Steps for retrieving the documents from bigram inverted index for a given input query -

1. Preprocess the given query as we did in the question1

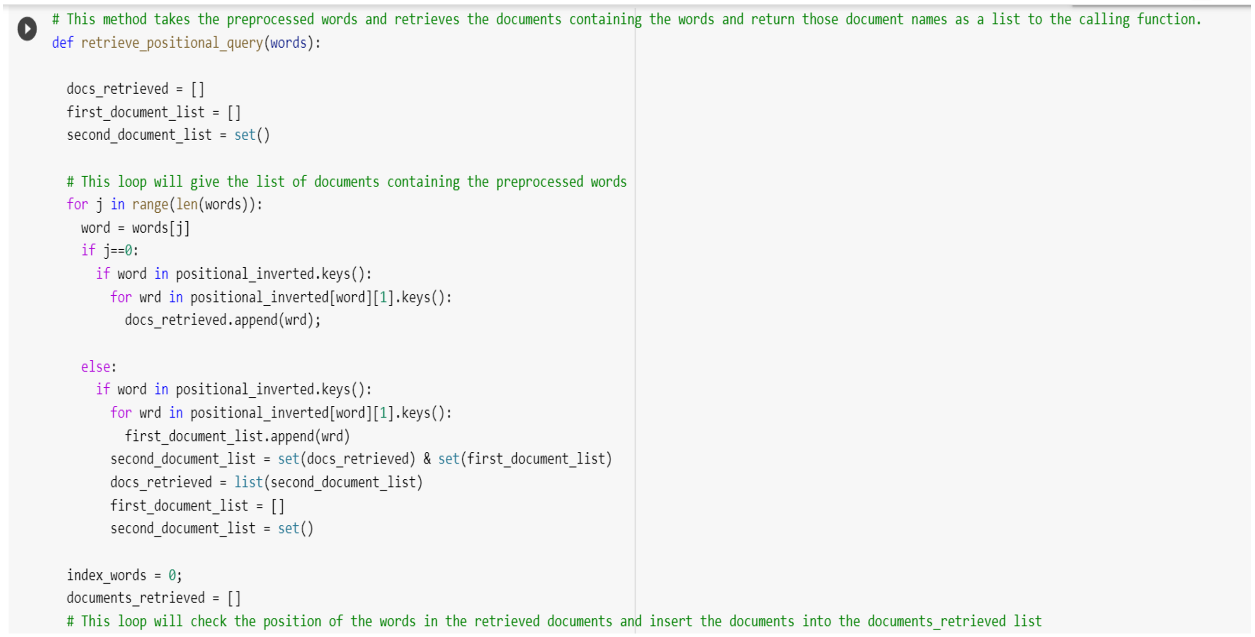
2. Iterate the query list after preprocessing and make a biword by combining the consecutive words with space between them.
3. Now check the biword in the bigram_inverted dictionary. If present, then add the documents to the retrieved documents list.
4. Do the same for every biword and do the AND operation after retrieving every biword and the previously retrieved documents.
5. Finally return the retrieved documents list to the calling function.



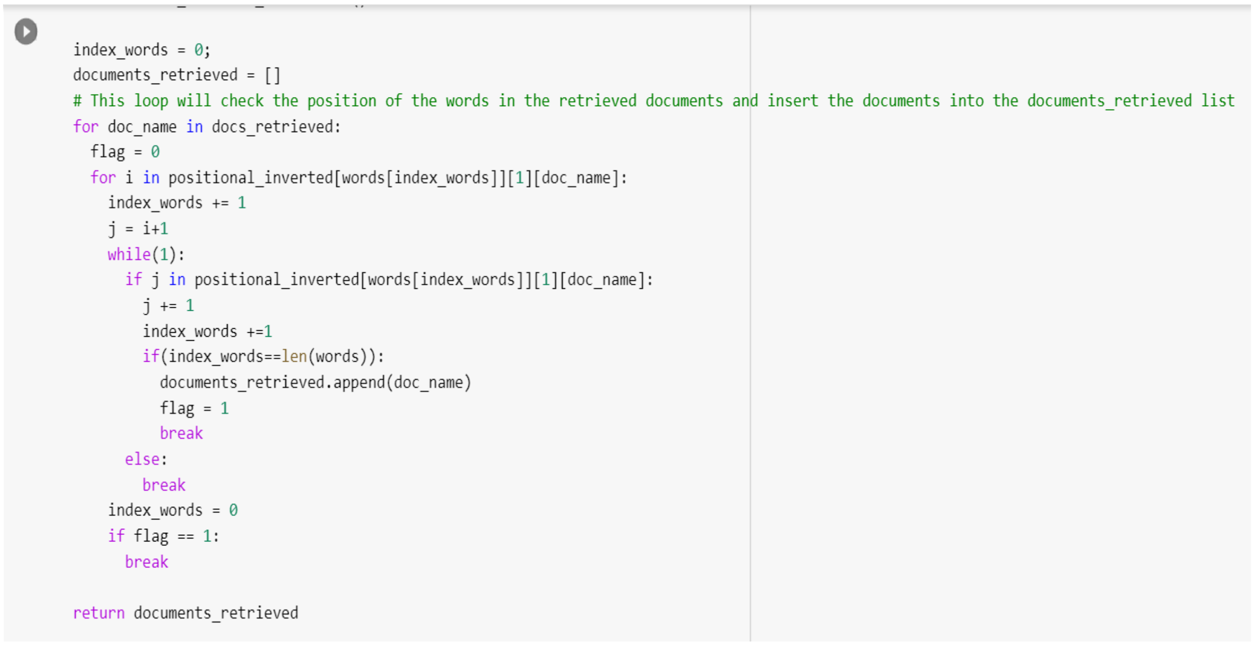
```python
# This method takes the preprocessed words and retrieves the documents containing the biwords and return those document names
# as a list to the calling function.
def retrieve_bigram_query(words):
  # Taking 2 lists and 1 set for collecting and doing intersection on those and finally storing the document names in the documents_retrieved list
  documents_retrieved = []
  first_document_list = []
  second_document_list = set()  # Set is used for not occurrence of any duplicates.

  for j in range(1,len(words)):    # Run the loop for number of words times
    word = words[j-1] + " " + words[j]   # Make a biword using consecutive words with space in between in them
    if j==1:
      if word in bigram_inverted.keys():
        for wrd in bigram_inverted[word]:
          documents_retrieved.append(wrd);
    else:
      if word in bigram_inverted.keys():
        for wrd in bigram_inverted[word]:
          first_document_list.append(wrd)
        second_document_list = set(documents_retrieved) & set(first_document_list)
        documents_retrieved = list(second_document_list)
        first_document_list = []
        second_document_list = set()

  return documents_retrieved
```

Steps for retrieving the documents from positional index for a given input query -

1. Preprocess the given query as we did in the question1
2. Iterate the query list after preprocessing and check for every word in the positional_inverted dictionary. Make a list containing the documents which contains all the words.
3. Iterate the documents list and check the words are in sequence or not by using positions of the words. If yes, then add the document name to the final retrieved document list.
4. Finally return the final retrieved documents list to the calling function.

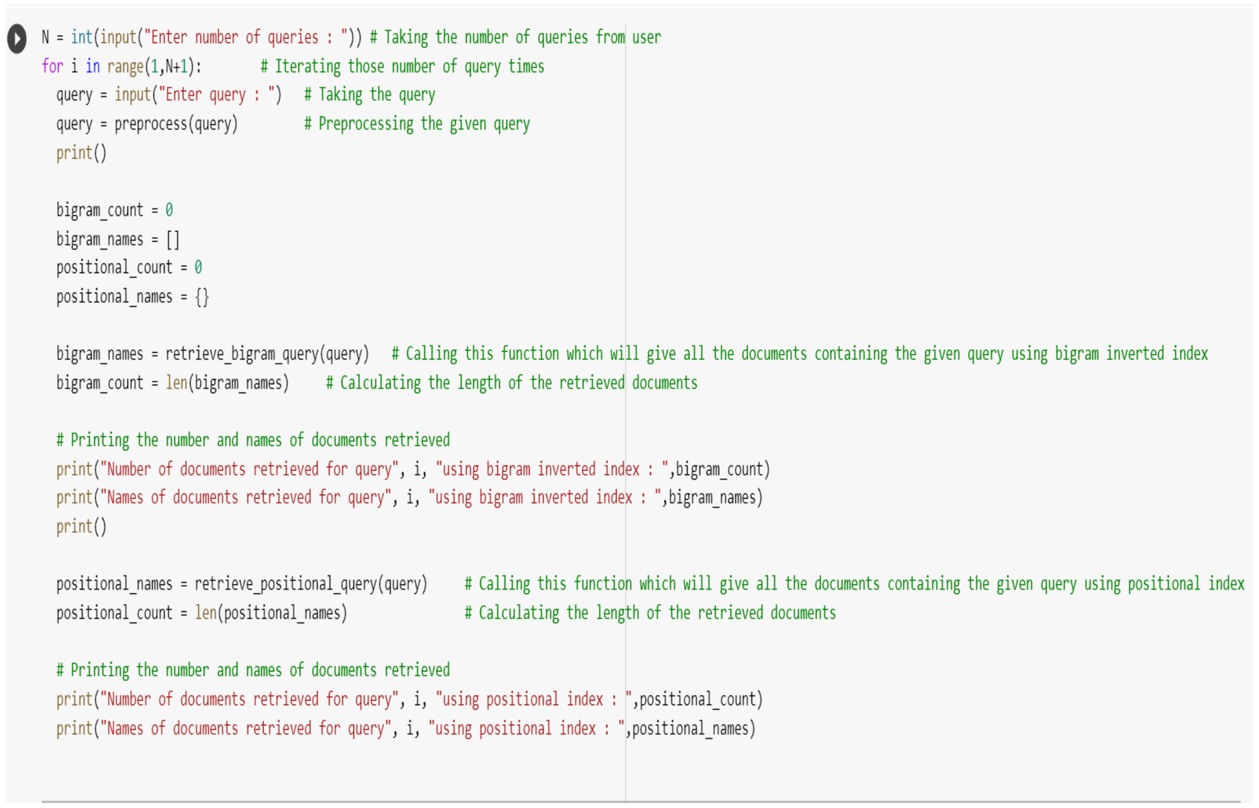


```python
# This method takes the preprocessed words and retrieves the documents containing the words and return those document names as a list to the calling function.
def retrieve_positional_query(words):

  docs_retrieved = []
  first_document_list = []
  second_document_list = set()

  # This loop will give the list of documents containing the preprocessed words
  for j in range(len(words)):
    word = words[j]
    if j==0:
      if word in positional_inverted.keys():
        for wrd in positional_inverted[word][1].keys():
          docs_retrieved.append(wrd);

    else:
      if word in positional_inverted.keys():
        for wrd in positional_inverted[word][1].keys():
          first_document_list.append(wrd)
        second_document_list = set(docs_retrieved) & set(first_document_list)
        docs_retrieved = list(second_document_list)
        first_document_list = []
        second_document_list = set()

  index_words = 0;
  documents_retrieved = []
  # This loop will check the position of the words in the retrieved documents and insert the documents into the documents_retrieved list
```



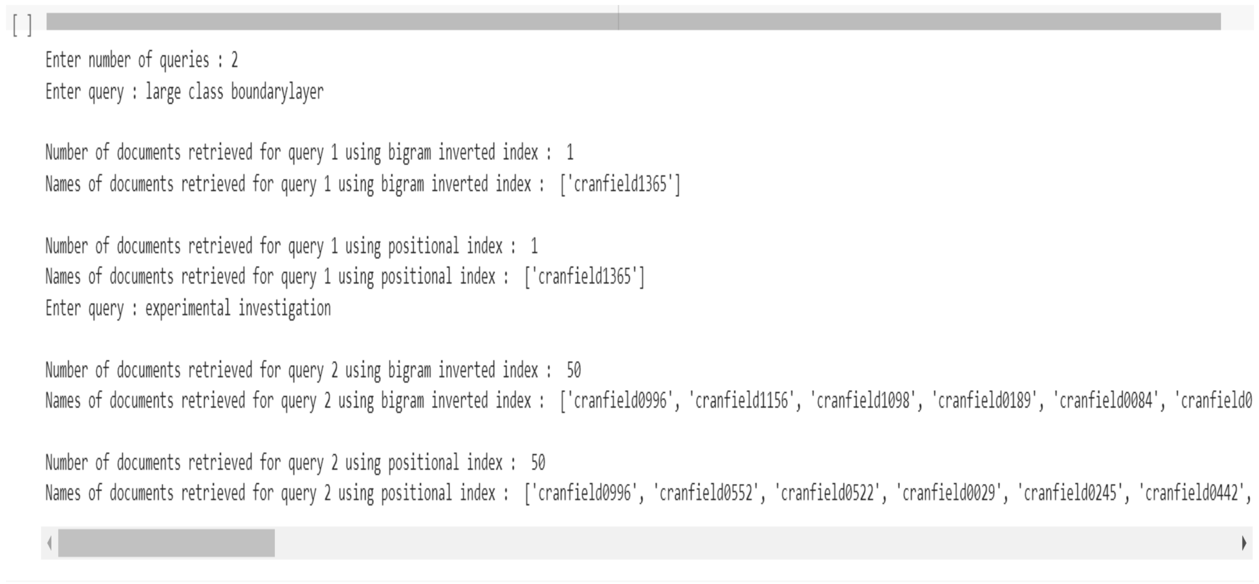
```python
  index_words = 0;
  documents_retrieved = []
  # This loop will check the position of the words in the retrieved documents and insert the documents into the documents_retrieved list
  for doc_name in docs_retrieved:
    flag = 0
    for i in positional_inverted[words[index_words]][1][doc_name]:
      index_words += 1
      j = i+1
      while(1):
        if j in positional_inverted[words[index_words]][1][doc_name]:
          j += 1
          index_words +=1
          if(index_words==len(words)):
            documents_retrieved.append(doc_name)
            flag = 1
            break
        else:
          break
      index_words = 0
      if flag == 1:
        break

  return documents_retrieved
```

Input :



```python
N = int(input("Enter number of queries : ")) # Taking the number of queries from user
for i in range(1,N+1):       # Iterating those number of query times
  query = input("Enter query : ")   # Taking the query
  query = preprocess(query)        # Preprocessing the given query
  print()

  bigram_count = 0
  bigram_names = []
  positional_count = 0
  positional_names = {}

  bigram_names = retrieve_bigram_query(query)   # Calling this function which will give all the documents containing the given query using bigram inverted index
  bigram_count = len(bigram_names)     # Calculating the length of the retrieved documents

  # Printing the number and names of documents retrieved
  print("Number of documents retrieved for query", i, "using bigram inverted index : ",bigram_count)
  print("Names of documents retrieved for query", i, "using bigram inverted index : ",bigram_names)
  print()

  positional_names = retrieve_positional_query(query)     # Calling this function which will give all the documents containing the given query using positional index
  positional_count = len(positional_names)                # Calculating the length of the retrieved documents

  # Printing the number and names of documents retrieved
  print("Number of documents retrieved for query", i, "using positional index : ",positional_count)
  print("Names of documents retrieved for query", i, "using positional index : ",positional_names)
```

Output :



```
Enter number of queries : 2
Enter query : large class boundarylayer

Number of documents retrieved for query 1 using bigram inverted index :  1
Names of documents retrieved for query 1 using bigram inverted index :  ['cranfield1365']

Number of documents retrieved for query 1 using positional index :  1
Names of documents retrieved for query 1 using positional index :  ['cranfield1365']
Enter query : experimental investigation

Number of documents retrieved for query 2 using bigram inverted index :  50
Names of documents retrieved for query 2 using bigram inverted index :  ['cranfield0996', 'cranfield1156', 'cranfield1098', 'cranfield0189', 'cranfield0084', 'cranfield0

Number of documents retrieved for query 2 using positional index :  50
Names of documents retrieved for query 2 using positional index :  ['cranfield0996', 'cranfield0552', 'cranfield0522', 'cranfield0029', 'cranfield0245', 'cranfield0442',
```