Q1 PreProcessing

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import os

# Download necessary NLTK datasets
nltk.download('punkt')
nltk.download('stopwords')

def preprocess_text(content):
    # Lowercase the text
    content = content.lower()

    # Tokenize the text
    tokens = word_tokenize(content)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Remove punctuation
    tokens = [token for token in tokens if token not in
string.punctuation]

    # Remove blank space tokens
    tokens = [token for token in tokens if token.strip()]

    # Rejoin tokens into a string
    preprocessed_content = ' '.join(tokens)

    return preprocessed_content

# Assuming we have a list of file paths for the dataset
dataset_directory = '/content/drive/MyDrive/text_files'  # Update this
path
file_paths = [os.path.join(dataset_directory, f) for f in
os.listdir(dataset_directory) if f.endswith('.txt')][:5]  # Only take
5 sample files

for file_path in file_paths:
    # Read the content of each file
    with open(file_path, 'r', encoding='utf-8') as file:
        content = file.read()

    print(f"Original content of {os.path.basename(file_path)}:\
```

```
n{content[:500]}\n")  # Print first 500 characters for brevity

    # Preprocess the content
    preprocessed_content = preprocess_text(content)

    print(f"Preprocessed content of {os.path.basename(file_path)}:\
n{preprocessed_content[:500]}\n")  # Print first 500 characters for
brevity

    # Save the preprocessed content back to a new file
    preprocessed_file_path = os.path.join(dataset_directory,
'preprocessed_' + os.path.basename(file_path))
    with open(preprocessed_file_path, 'w', encoding='utf-8') as file:
        file.write(preprocessed_content)
```

```
Original content of file511.txt:
I loved using this while producing videos with my Canon EOS 6D, but it
didn't last long.  When I inserted the battery into the receiver, one
of the battery contacts broke off.  Here are some pictures of the
receiver mounted on my EOS 6D and a picture showing the broken contact
wire.  The battery holder has a bad design because the contacts are
weak wire loops and the battery fits very snug and puts a lot of
pressure on these weak contacts.  Samson should update the battery
holder design.  I stil

Preprocessed content of file511.txt:
loved using producing videos canon eos 6d n't last long inserted
battery receiver one battery contacts broke pictures receiver mounted
eos 6d picture showing broken contact wire battery holder bad design
contacts weak wire loops battery fits snug puts lot pressure weak
contacts samson update battery holder design still recommend wonderful
wireless system something aware

Original content of file209.txt:
Love it!!! Would buy again!

Preprocessed content of file209.txt:
love would buy

Original content of file657.txt:
This is for my son in his rehearsal room and they are perfect! Great
quality, arrived within a few days and the price you cant beat!

Preprocessed content of file657.txt:
son rehearsal room perfect great quality arrived within days price
cant beat

Original content of file172.txt:
These stands are excellent. They are well built and easy to assemble.
they come with an Allen wrench and that is the only tool needed. The
```

```
stand has a nice design when it comes to the base. There are 3 soft rubber pads that come on the stand. They also give you 3 pointed metal spikes that you can put on, which is perfect if placing on a rug. The spikes are adjustable so you can get set for the rug height and get the stand perfectly level. I put a pair of JBL monitor speakers (LSR305) on this sta

Preprocessed content of file172.txt:
stands excellent well built easy assemble come allen wrench tool needed stand nice design comes base 3 soft rubber pads come stand also give 3 pointed metal spikes put perfect placing rug spikes adjustable get set rug height get stand perfectly level put pair jbl monitor speakers lsr305 stand yamaha piano dgx-650b see attached pictures stands look like made go piano quality metal good despite reviews pins allow lock stand certain height n't worry lowering comes wire guides run cables tight stand

Original content of file366.txt:
Overall I'm happy with the stand.  I admit I was scratching my head a bit when I was putting it together.  There's more pieces to this stand than others I've had.  Adjusting it is a little bit of a pain because there are 4 different areas that you can loosen and change around.  If you travel and gig a lot, you may want to go with the higher model, the adjustments look a bit easier.  However once all setup, it's sturdy and well built.

FYI the stand shown currently is not the same design as the o

Preprocessed content of file366.txt:
overall 'm happy stand admit scratching head bit putting together 's pieces stand others 've adjusting little bit pain 4 different areas loosen change around travel gig lot may want go higher model adjustments look bit easier however setup 's sturdy well built fyi stand shown currently design one received uploaded picture one received order page separate arm piece inserts main stand


[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

#**Q2** Inverted_Index

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import os
import pickle
from tqdm.notebook import tqdm
```

```
import string
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from collections import defaultdict
from functools import reduce
import copy

nltk.download("punkt")
nltk.download("stopwords")

class TextIndexer:
    def __init__(self):
        self.index = {}

    def removeSpecialChars(self, text):
        return ''.join(c for c in text if c.isalnum() and not
c.isdigit() and c not in string.punctuation)

    def processText(self, text):
        filteredWords = set(stopwords.words('english'))
        text = text.lower()
        tokens = word_tokenize(text)
        tokens = [i for i in tokens if i not in filteredWords]
        tokens = [self.removeSpecialChars(x) for x in tokens]
        return tokens

    def addToIndex(self, content, contentID):
        words = self.processText(content)
        for position, word in enumerate(words):
            if word in self.index:
                self.index[word][0] += 1
                if contentID in self.index[word][1]:
                    self.index[word][1][contentID].append(position)
                else:
                    self.index[word][1][contentID] = [position]
            else:
                self.index[word] = [1, {contentID: [position]}]

    def storeIndex(self, filename='index_output.pickle'):
        with open(filename, 'wb') as file:
            pickle.dump(self.index, file)

class SearchQuery:
    def __init__(self, index_file='index_output.pickle',
mapping_file='file_mapping.pickle'):
        with open(index_file, 'rb') as file:
            self.index = pickle.load(file)
        self.index = defaultdict(lambda: [], self.index)
        with open(mapping_file, 'rb') as file:
            self.mapping = pickle.load(file)
```

```
    def removeSpecialChars(self, text):
        return ''.join(c for c in text if c.isalnum() and not
c.isdigit() and c not in string.punctuation)

    def processText(self, text):
        filteredWords = set(stopwords.words('english'))
        text = text.lower()
        tokens = word_tokenize(text)
        tokens = [i for i in tokens if i not in filteredWords]
        tokens = [self.removeSpecialChars(x) for x in tokens]
        tokens = [x for x in tokens if len(x) > 1]
        return tokens

    def listIntersection(self, lists):
        if not lists:
            return []
        lists.sort(key=len)
        return list(reduce(lambda x, y: set(x) & set(y), lists))

    def getTermPostings(self, terms):
        return [[ [docID, self.index[term][1][docID]] for docID in
self.index[term][1]] for term in terms]

    def extractDocIDs(self, postings):
        return [[item[0] for item in term] for term in postings]

    def query(self, queryText):
        terms = self.processText(queryText)
        if not self.index.keys():
            return []
        for term in terms:
            if term not in self.index:
                return []
        if len(terms) == 1:
            docIDs = list(self.index[terms[0]][1].keys())
        else:
            postings = self.getTermPostings(terms)
            docs = self.extractDocIDs(postings)
            docs = self.listIntersection(docs)
            for i in range(len(postings)):
                postings[i] = [x for x in postings[i] if x[0] in docs]
            postings = copy.deepcopy(postings)
            for i in range(len(postings)):
                for j in range(len(postings[i])):
                    postings[i][j][1] = [pos - i for pos in
postings[i][j][1]]
            docIDs = []
            for i in range(len(postings[0])):
                commonPos = self.listIntersection([x[i][1] for x in
```

```
postings])
                if commonPos:
                    docIDs.append(postings[0][i][0])
        docIDs = list(map(int, docIDs))
        return docIDs

    def displayFiles(self, docIDs):
        files = sorted([(docID, self.mapping[docID]) for docID in
docIDs])
        return files

def main():
    indexer = TextIndexer()
    fileList = []
    dataset_directory = '/content/drive/MyDrive/text_files'
    fileList = [os.path.join(dataset_directory, f) for f in
os.listdir(dataset_directory) if f.endswith('.txt')]

    mapping = {}

    for i, path in enumerate(fileList):
        try:
            content = open(path, encoding="utf8").read().replace('\n',
' ')
        except Exception:
            content = open(path,
encoding="unicode_escape").read().replace('\n', ' ')
        indexer.addToIndex(content, i)
        mapping[i] = path

    indexer.storeIndex()
    with open('file_mapping.pickle', 'wb') as file:
        pickle.dump(mapping, file)

    queryEngine = SearchQuery()

    num_queries = int(input("Enter number of queries: "))
    for i in range(num_queries):
        queryText = input(f"Enter query {i+1}: ")
        results = queryEngine.query(queryText)
        files = queryEngine.displayFiles(results)
        print(f"Number of documents retrieved for query {i+1} using
positional index: {len(results)}")
        if files:
            print(f"Names of documents retrieved for query {i+1} using
positional index: {', '.join([f[1] for f in files])}")
        else:
            print("No documents retrieved")
```

```
if __name__ == "__main__":
    main()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Enter number of queries: 2
Enter query 1: Car bag in a canister
Number of documents retrieved for query 1 using positional index: 0
No documents retrieved
Enter query 2:  Coffee brewing techniques in cookbook
Number of documents retrieved for query 2 using positional index: 0
No documents retrieved
```

#**Q3** Positional_Index

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import os
import pickle
from tqdm import tqdm
import string
from collections import defaultdict
from functools import reduce
import copy

# Ensure NLTK resources have been downloaded
nltk.download("punkt", quiet=True)
nltk.download("stopwords", quiet=True)

class IndexCreator:
    def __init__(self):
        self.indices = {}

    def cleanText(self, content):
        return ''.join(c for c in content if c.isalnum() and not
c.isdigit() and c not in string.punctuation)

    def processContent(self, content):
        ignoreWords = set(stopwords.words('english'))
        content = content.lower()
        tokens = word_tokenize(content)
        tokens = [i for i in tokens if i not in ignoreWords]
        tokens = [self.cleanText(x) for x in tokens]
        return tokens

    def buildIndex(self, text, docID):
```

```
        words = self.processContent(text)
        for pos, term in enumerate(words):
            if term in self.indices:
                self.indices[term][0] += 1
                if docID in self.indices[term][1]:
                    self.indices[term][1][docID].append(pos)
                else:
                    self.indices[term][1][docID] = [pos]
            else:
                self.indices[term] = [1, {docID: [pos]}]

    def saveIndices(self, filename='indices.pkl'):
        with open(filename, 'wb') as f:
            pickle.dump(self.indices, f)

class QueryProcessor:
    def __init__(self, indexFile='indices.pkl',
docMappingFile='docMapping.pkl'):
        with open(indexFile, 'rb') as f:
            self.indices = pickle.load(f)
        with open(docMappingFile, 'rb') as f:
            self.docMapping = pickle.load(f)

    def search(self, queryTerms, logicOps):
        docSets = []
        for term in queryTerms:
            if term in self.indices:
                docIDs = set(self.indices[term][1].keys())  # Get
document IDs containing the term
                docSets.append(docIDs)
            else:
                return []  # If any term is not found, return an empty
list

        # Find the intersection of all document ID sets to get IDs
containing all terms
        if docSets:
            resultIDs = set.intersection(*docSets)
            return list(resultIDs)
        else:
            return []

    def displayResults(self, docIDs):
        if docIDs is None:  # Handle the case where docIDs is None
            return []
        docNames = [self.docMapping[str(id)] for id in docIDs]  #
Ensure IDs are converted to strings if necessary
        return docNames
```

```
def main():
    dataset_directory = '/content/drive/MyDrive/text_files'  # Adjust
this path to your dataset directory
    fileList = [os.path.join(dataset_directory, f) for f in
os.listdir(dataset_directory) if f.endswith('.txt')]
    docMapping = {i: name for i, name in enumerate(fileList)}

    ic = IndexCreator()
    for docID, filePath in enumerate(fileList):
        with open(filePath, 'r', encoding='utf-8') as file:
            content = file.read()
        ic.buildIndex(content, docID)
    ic.saveIndices()

    # Save the document mapping
    with open('docMapping.pkl', 'wb') as f:
        pickle.dump(docMapping, f)

    qp = QueryProcessor()

    numQueries = int(input("Enter the number of queries: "))
    for i in range(numQueries):
        query = input(f"Enter query {i+1}: ")
        logic = input("Enter logic operators: ")
        queryTerms = query.split(', ')
        logicOps = logic.split(', ')
        docIDs = qp.search(queryTerms, logicOps) # You need to
implement the search logic based on queryTerms and logicOps
        print(f"Query {i+1}: {' '.join(queryTerms)}")
        # Assume docIDs is defined after implementing search logic
        docNames = qp.displayResults(docIDs)
        print(f"Names of the documents retrieved for query {i+1}: {',
'.join(docNames)}")
        # Placeholder for search logic demonstration
        print("Search logic to be implemented")

if __name__ == "__main__":
    main()
```

```
Enter the number of queries: 2
Enter query 1: Car bag in a canister
Enter logic operators: OR, AND NOT
Query 1: Car bag in a canister
Names of the documents retrieved for query 1:
Search logic to be implemented
Enter query 2:  Coffee brewing techniques in cookbook
Enter logic operators: AND, OR NOT,OR
Query 2:  Coffee brewing techniques in cookbook
Names of the documents retrieved for query 2:
Search logic to be implemented
```