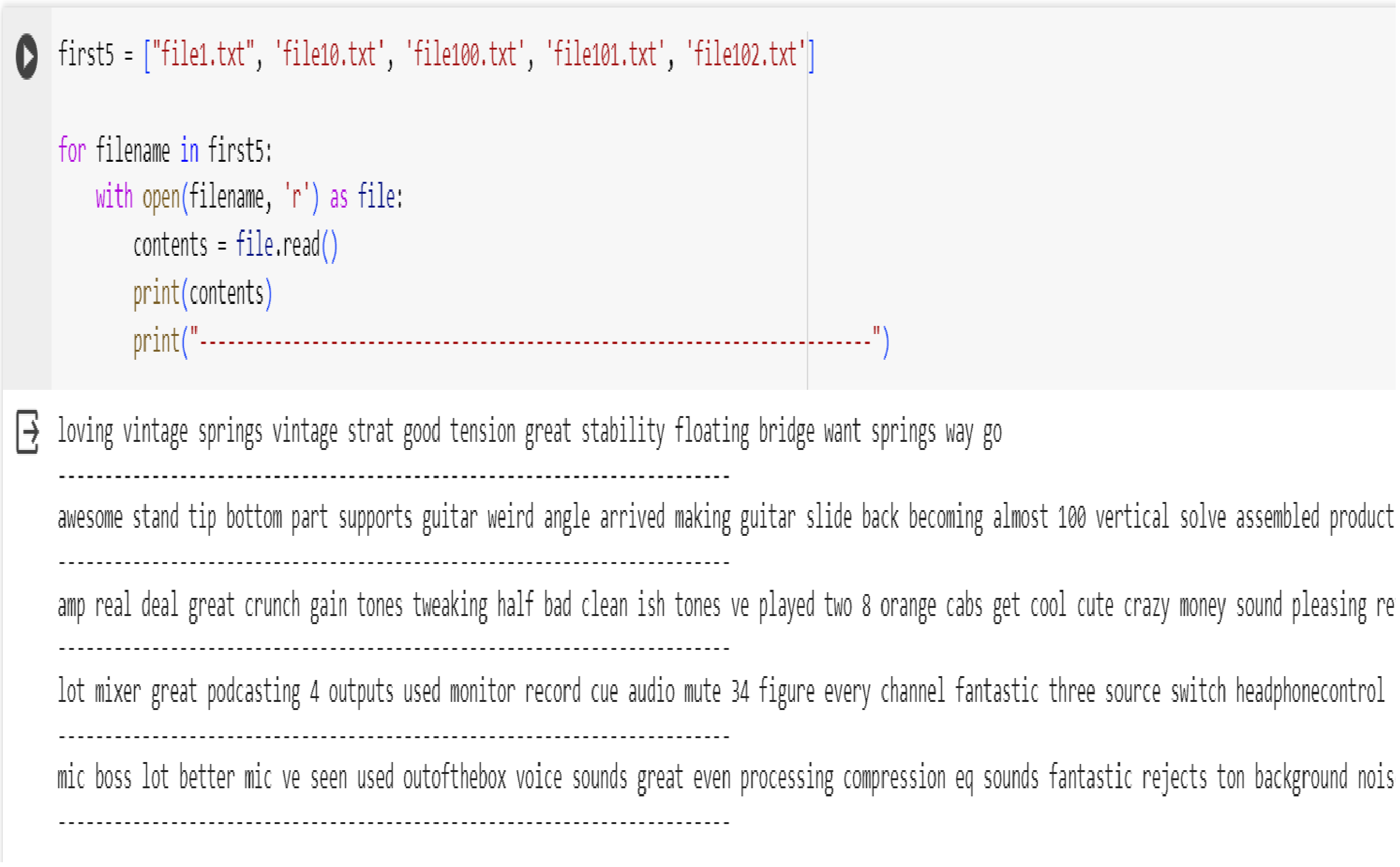
ASSIGNMENT-1 REPORT

1. DATA PREPROCESSING
    - Remove the HTML tags
      Remove the html tags related tokens using BeautifulSoup
    - Remove the lowercase
      Modify the text into lowercase using contents.lower()
    - Tokenization
      Spitting raw text into tokens using nltk.tokenize.word_tokenize()
    - Removing the stop words
      Remove stopword using nltk.corpus.stopwords.words('english')
    - Remove punctuation marks
    - Removing of blank space

File after preprocessing



```
first5 = ["file1.txt", 'file10.txt', 'file100.txt', 'file101.txt', 'file102.txt']

for filename in first5:
    with open(filename, 'r') as file:
        contents = file.read()
        print(contents)
        print("------------------------------------------------------------------------")
```

```
loving vintage springs vintage strat good tension great stability floating bridge want springs way go
------------------------------------------------------------------------
awesome stand tip bottom part supports guitar weird angle arrived making guitar slide back becoming almost 100 vertical solve assembled product
------------------------------------------------------------------------
amp real deal great crunch gain tones tweaking half bad clean ish tones ve played two 8 orange cabs get cool cute crazy money sound pleasing re
------------------------------------------------------------------------
lot mixer great podcasting 4 outputs used monitor record cue audio mute 34 figure every channel fantastic three source switch headphonecontrol
------------------------------------------------------------------------
mic boss lot better mic ve seen used outofthebox voice sounds great even processing compression eq sounds fantastic rejects ton background nois
------------------------------------------------------------------------
```

2. Unigram Inverted Index and Boolean Queries

1.Function to Extract Numbers from Filenames:

- The function **extract_number(filename)** extracts numerical values from filenames by filtering out non-digit characters and converting the remaining characters into an integer. This function is used to extract unique identifiers from filenames, presumably used to identify each document uniquely.

2.Initialization of Index:

- Initialize an empty dictionary named **index** to store the inverted index. The keys of this dictionary will be words found in the documents, and the values will be lists containing the document IDs where each word occurs.

3.Processing Each Text File:

- Iterate through each text file in the **txt_files** list.
- Open each file and read its content.
- Split the content into individual words.
- Extract the numerical ID from the filename using the **extract_number()** function.

4.Building the Inverted Index:

- For each word in the content of the file:
- Check if the word consists only of alphabetic characters (excluding punctuation, digits, etc.).
- If the word is valid:
- If the word is not already in the **index** dictionary, add it with an empty list as its value.
- Append the document ID to the list corresponding to the word.

5.Sorting Document IDs:

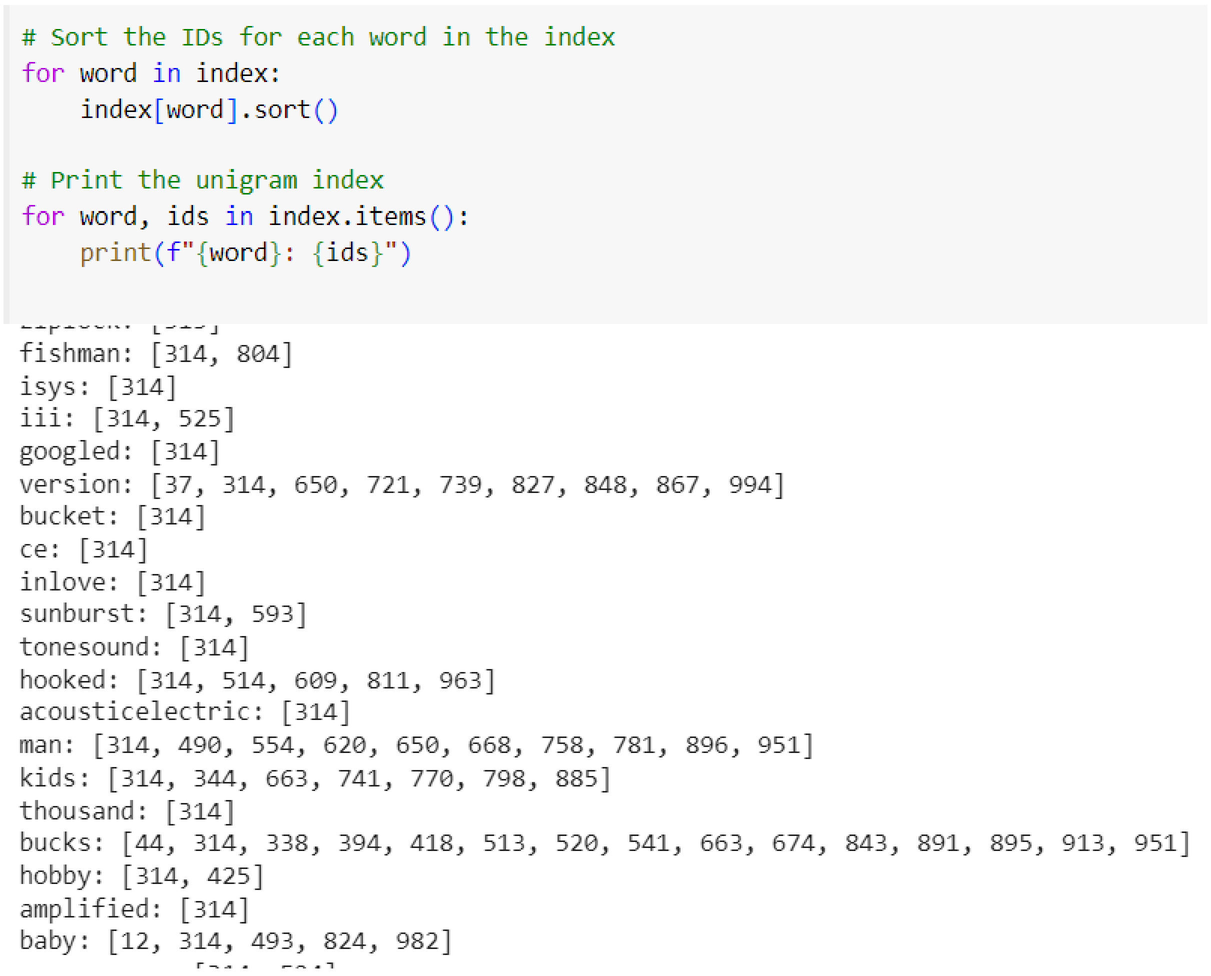
- After completing the index, sort the list of document IDs associated with each word. This step ensures that the IDs are in ascending order for easier processing or retrieval.

```python
# Function to split number and string in a filename
def extract_number(filename):
    return int(''.join(filter(str.isdigit, filename)))

# Creating the unigram index
index = {}
for file in txt_files:
    with open(file, 'r') as f:
        content = f.read()
        words = content.split()
        file_id = extract_number(file)
        for word in words:
            if word.isalpha():

              if word not in index:
                  index[word] = []
              if file_id not in index[word]:
                  index[word].append(file_id)
```

Unigram inverted index output



```python
# Sort the IDs for each word in the index
for word in index:
    index[word].sort()

# Print the unigram index
for word, ids in index.items():
    print(f"{word}: {ids}")
```

```
fishman: [314, 804]
isys: [314]
iii: [314, 525]
googled: [314]
version: [37, 314, 650, 721, 739, 827, 848, 867, 994]
bucket: [314]
ce: [314]
inlove: [314]
sunburst: [314, 593]
tonesound: [314]
hooked: [314, 514, 609, 811, 963]
acousticelectric: [314]
man: [314, 490, 554, 620, 650, 668, 758, 781, 896, 951]
kids: [314, 344, 663, 741, 770, 798, 885]
thousand: [314]
bucks: [44, 314, 338, 394, 418, 513, 520, 541, 663, 674, 843, 891, 895, 913, 951]
hobby: [314, 425]
amplified: [314]
baby: [12, 314, 493, 824, 982]
```

6. provide the support for AND,OR,OR NOT,AND NOT

```python
def AND(set1, set2):
    set_result = set1.intersection(set2)
    return set_result


def AND_NOT(set1, set2):
    set_result = set1.difference(set2)
    return set_result


def OR(set1, set2):
    set_result = set1.union(set2)
    return set_result


def OR_NOT(set1, set2):
    set_result = set1.union(set1, uni_set - set2)
    return set_result
```

7. Query Processing

1. Input

User provide the number of queries and sequence and operations.

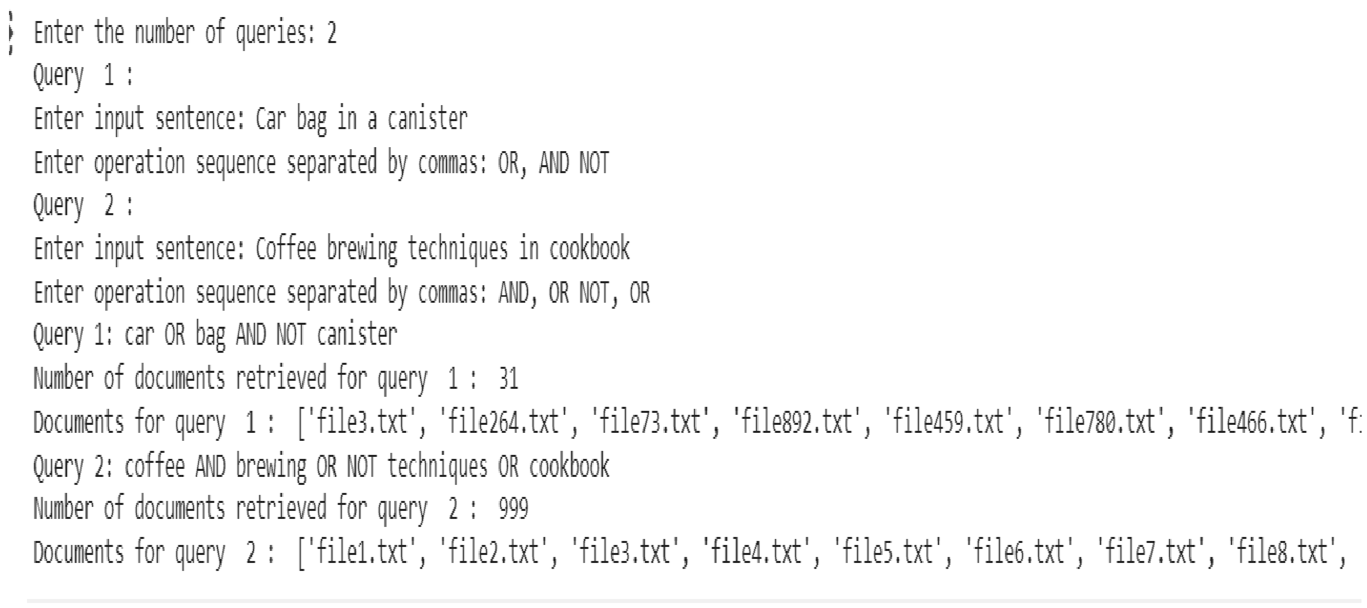
2.Preprocessing

Input sequence provided by user go through preprocessing

4. Query Execution: process_query executes the query based on operators, merging lists of document names accordingly

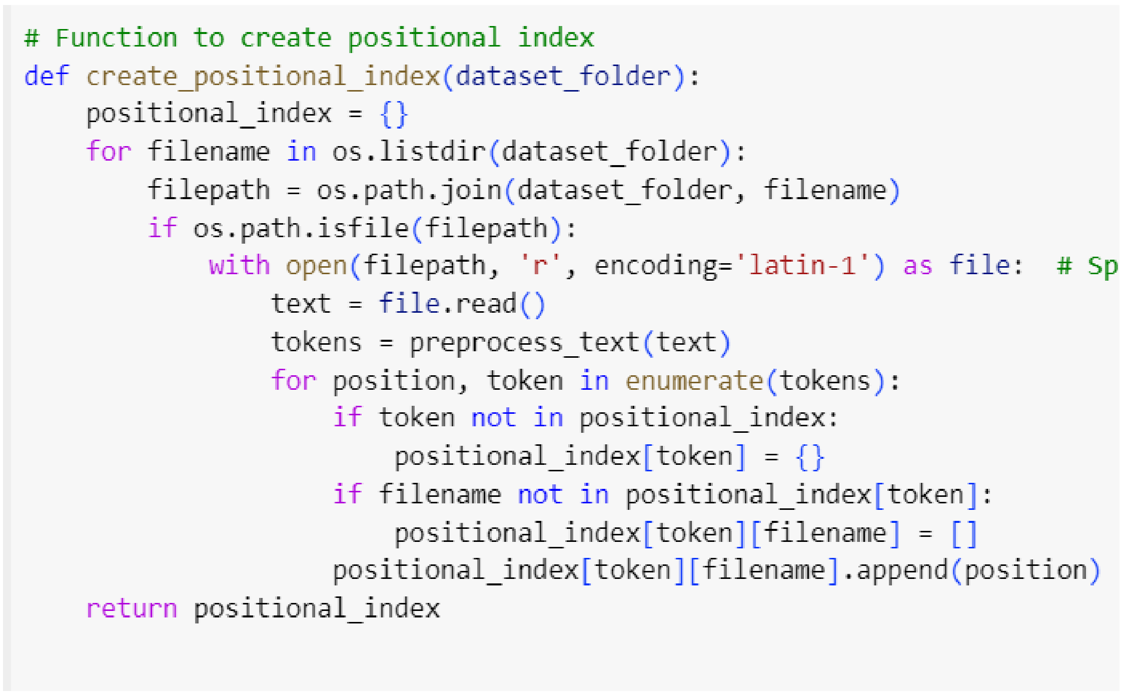
3.Output

First line will be show the Query (preprocessing sequence with Operation) and second line print the number of document retrived For query and third line print the names documents



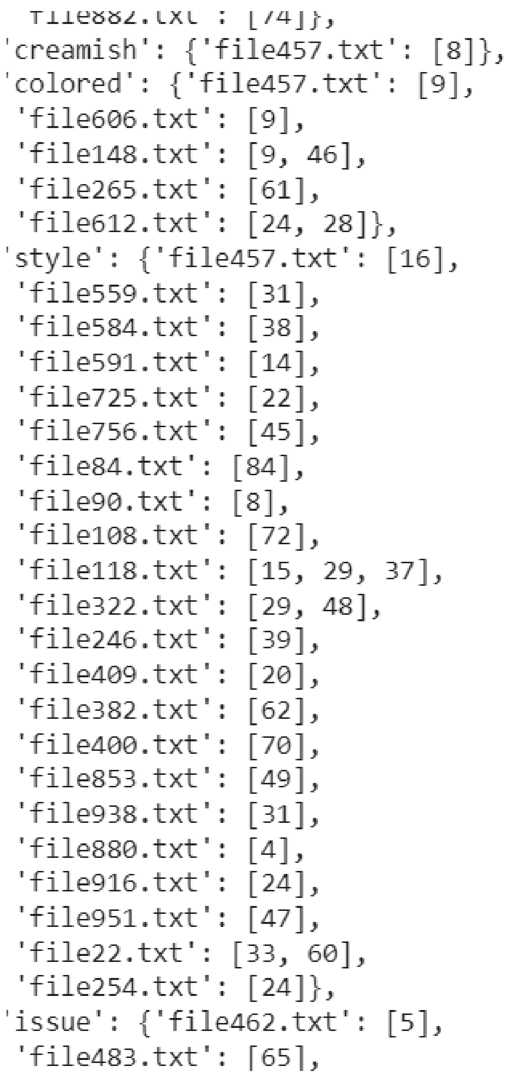
```
Enter the number of queries: 2
Query  1 :
Enter input sentence: Car bag in a canister
Enter operation sequence separated by commas: OR, AND NOT
Query  2 :
Enter input sentence: Coffee brewing techniques in cookbook
Enter operation sequence separated by commas: AND, OR NOT, OR
Query 1: car OR bag AND NOT canister
Number of documents retrieved for query  1 :  31
Documents for query  1 :  ['file3.txt', 'file264.txt', 'file73.txt', 'file892.txt', 'file459.txt', 'file780.txt', 'file466.txt', 'f:
Query 2: coffee AND brewing OR NOT techniques OR cookbook
Number of documents retrieved for query  2 :  999
Documents for query  2 :  ['file1.txt', 'file2.txt', 'file3.txt', 'file4.txt', 'file5.txt', 'file6.txt', 'file7.txt', 'file8.txt',
```

3.Positional Index and Phrase Queries



```python
# Function to create positional index
def create_positional_index(dataset_folder):
    positional_index = {}
    for filename in os.listdir(dataset_folder):
        filepath = os.path.join(dataset_folder, filename)
        if os.path.isfile(filepath):
            with open(filepath, 'r', encoding='latin-1') as file:  # Sp
                text = file.read()
                tokens = preprocess_text(text)
                for position, token in enumerate(tokens):
                    if token not in positional_index:
                        positional_index[token] = {}
                    if filename not in positional_index[token]:
                        positional_index[token][filename] = []
                    positional_index[token][filename].append(position)
    return positional_index
```

Positional index output



```
 'file882.txt': [74]},
'creamish': {'file457.txt': [8]},
'colored': {'file457.txt': [9],
 'file606.txt': [9],
 'file148.txt': [9, 46],
 'file265.txt': [61],
 'file612.txt': [24, 28]},
'style': {'file457.txt': [16],
 'file559.txt': [31],
 'file584.txt': [38],
 'file591.txt': [14],
 'file725.txt': [22],
 'file756.txt': [45],
 'file84.txt': [84],
 'file90.txt': [8],
 'file108.txt': [72],
 'file118.txt': [15, 29, 37],
 'file322.txt': [29, 48],
 'file246.txt': [39],
 'file409.txt': [20],
 'file382.txt': [62],
 'file400.txt': [70],
 'file853.txt': [49],
 'file938.txt': [31],
 'file880.txt': [4],
 'file916.txt': [24],
 'file951.txt': [47],
 'file22.txt': [33, 60],
 'file254.txt': [24]},
'issue': {'file462.txt': [5],
 'file483.txt': [65],
```

File Iteration:

- Iterate through each file in the dataset folder.

Text Processing

Read and preprocess each file's content into tokens

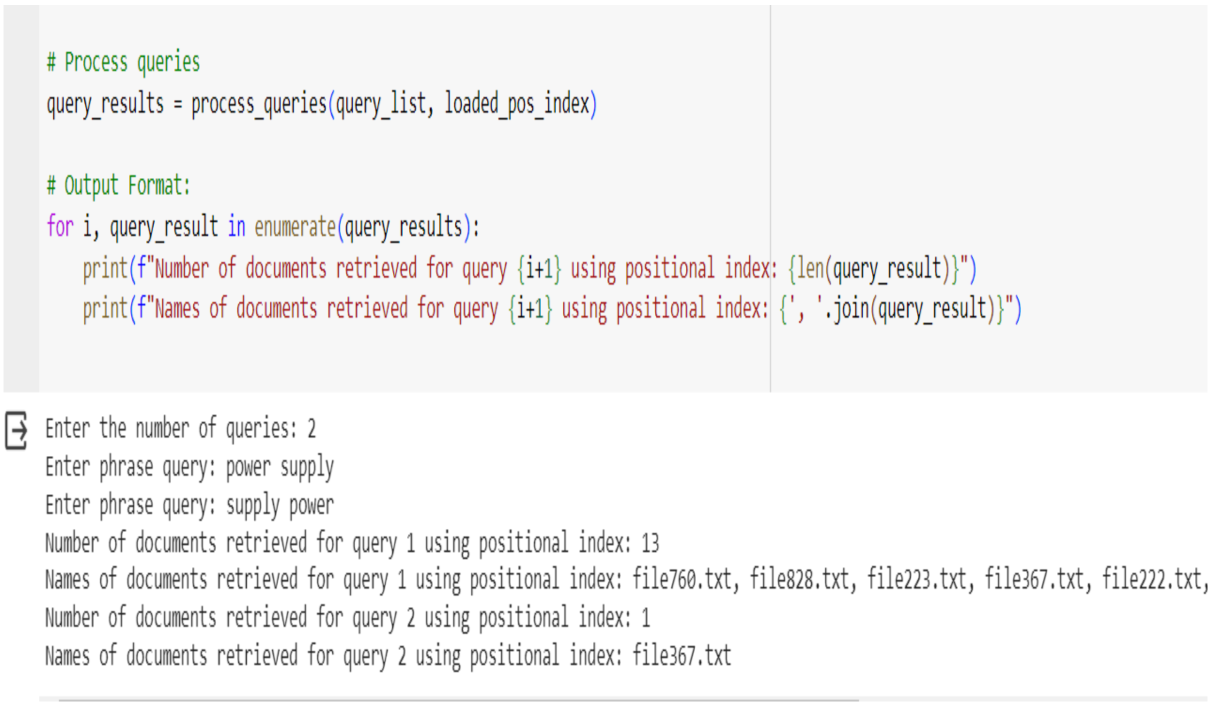
Index Construction:

- Build a positional index mapping tokens to filenames and their positions in the respective files.

Return Index:

Return the constructed positional index for efficient retrieval of terms and their positions within documents.

QUERY OUTPUT



```
# Process queries
query_results = process_queries(query_list, loaded_pos_index)

# Output Format:
for i, query_result in enumerate(query_results):
    print(f"Number of documents retrieved for query {i+1} using positional index: {len(query_result)}")
    print(f"Names of documents retrieved for query {i+1} using positional index: {', '.join(query_result)}")
```

```
Enter the number of queries: 2
Enter phrase query: power supply
Enter phrase query: supply power
Number of documents retrieved for query 1 using positional index: 13
Names of documents retrieved for query 1 using positional index: file760.txt, file828.txt, file223.txt, file367.txt, file222.txt,
Number of documents retrieved for query 2 using positional index: 1
Names of documents retrieved for query 2 using positional index: file367.txt
```