CSE3087-Information Retrieval and Organization

Assignment 2

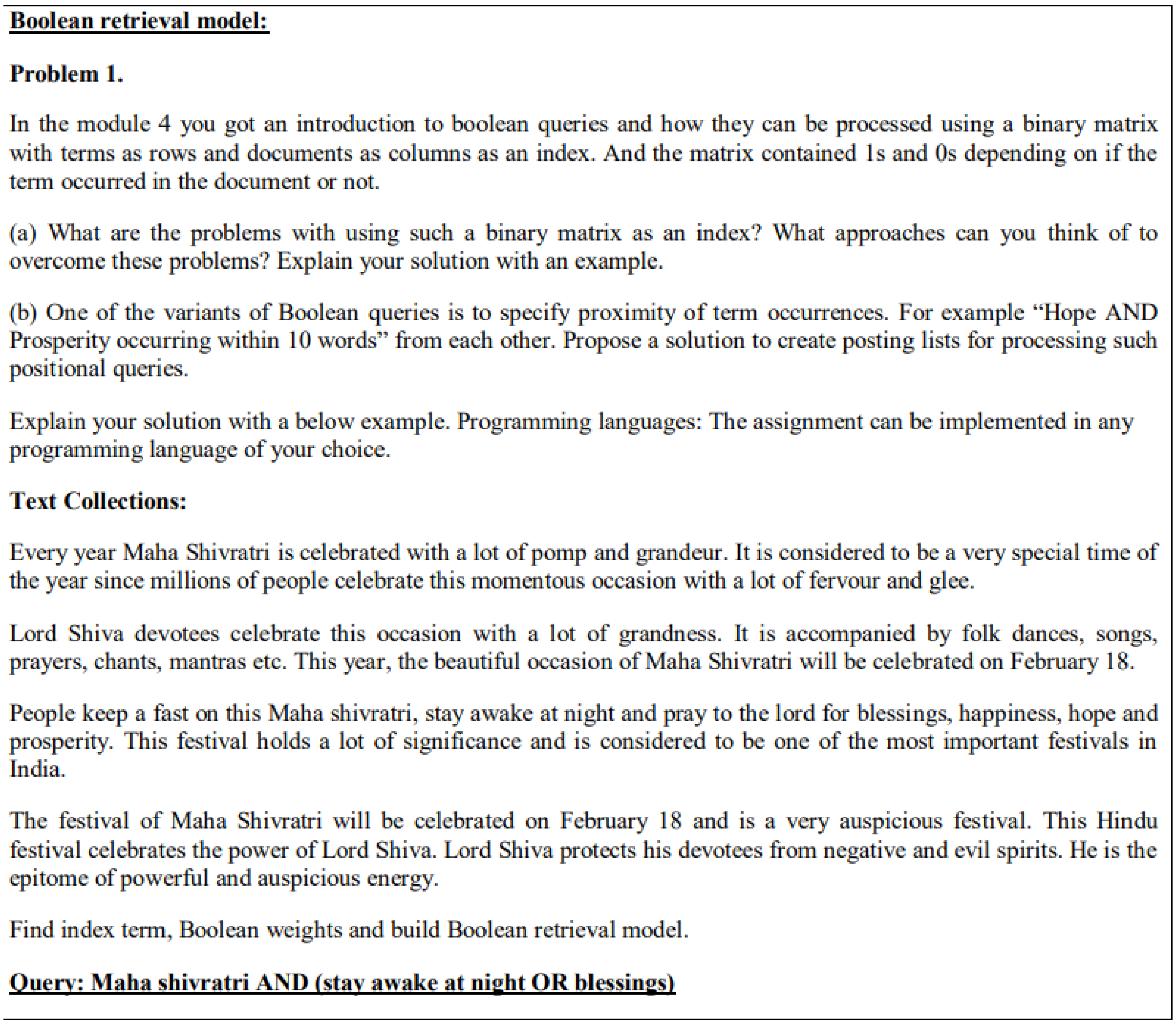
GROUP MEMBERS:

SRIGANTH.R(20MIA1039)

P.MADHUKIRAN(20MIA1106)

H.A.PRIYANKA(20MIA1140)

GOPICHAND(20MIA1070)



Boolean retrieval model:

Problem 1.

In the module 4 you got an introduction to boolean queries and how they can be processed using a binary matrix with terms as rows and documents as columns as an index. And the matrix contained 1s and 0s depending on if the term occurred in the document or not.

(a) What are the problems with using such a binary matrix as an index? What approaches can you think of to overcome these problems? Explain your solution with an example.

(b) One of the variants of Boolean queries is to specify proximity of term occurrences. For example "Hope AND Prosperity occurring within 10 words" from each other. Propose a solution to create posting lists for processing such positional queries.

Explain your solution with a below example. Programming languages: The assignment can be implemented in any programming language of your choice.

Text Collections:

Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this momentous occasion with a lot of fervour and glee.

Lord Shiva devotees celebrate this occasion with a lot of grandness. It is accompanied by folk dances, songs, prayers, chants, mantras etc. This year, the beautiful occasion of Maha Shivratri will be celebrated on February 18.

People keep a fast on this Maha shivratri, stay awake at night and pray to the lord for blessings, happiness, hope and prosperity. This festival holds a lot of significance and is considered to be one of the most important festivals in India.

The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva protects his devotees from negative and evil spirits. He is the epitome of powerful and auspicious energy.

Find index term, Boolean weights and build Boolean retrieval model.

Query: Maha shivratri AND (stay awake at night OR blessings)

Part 1:

Issues with the Binary Matrix as an Index

There are various issues with using a binary matrix as an index in the Boolean retrieval paradigm. Some of the issues are as follows:

Sparsity: A sparse matrix is one in which the majority of the entries are 0. Because of this sparsity, a huge amount of memory space is wasted, and computing time increases dramatically.

Scalability: As the number of documents and words increases, so does the size of the matrix. As the size of the matrix grows, it becomes more difficult to efficiently store and handle the data.

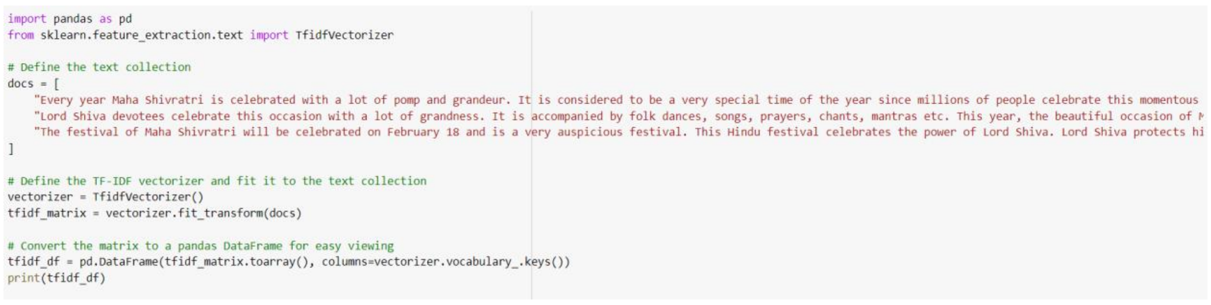
Weighting: The binary matrix does not take the relevance of the phrases in the texts into consideration. Certain words may appear often in numerous papers, yet they may not be useful in discriminating between relevant and irrelevant texts.

Approach:

We may utilise techniques like term weighting and dimensionality reduction to address these issues. The Term Frequency-Inverse Document Frequency (TF-IDF) methodology is one such method.

The TF-IDF approach provides a weight to each phrase in the matrix based on its importance in the document and the corpus as a whole. The weight is proportional to the term's frequency in the document and inversely proportionate to the term's frequency in the total corpus.

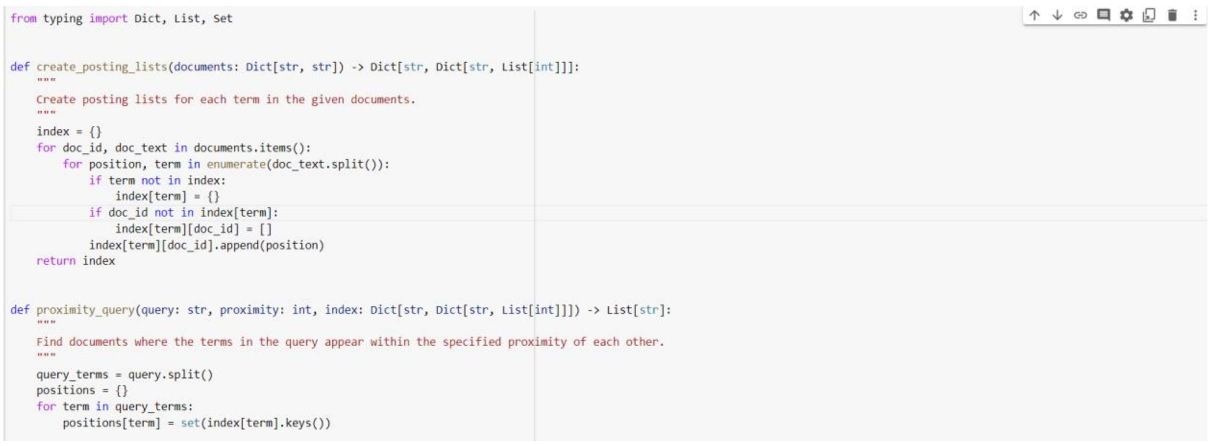
Here's an example of how to use TF-IDF in Python to weight the words in a given text collection:



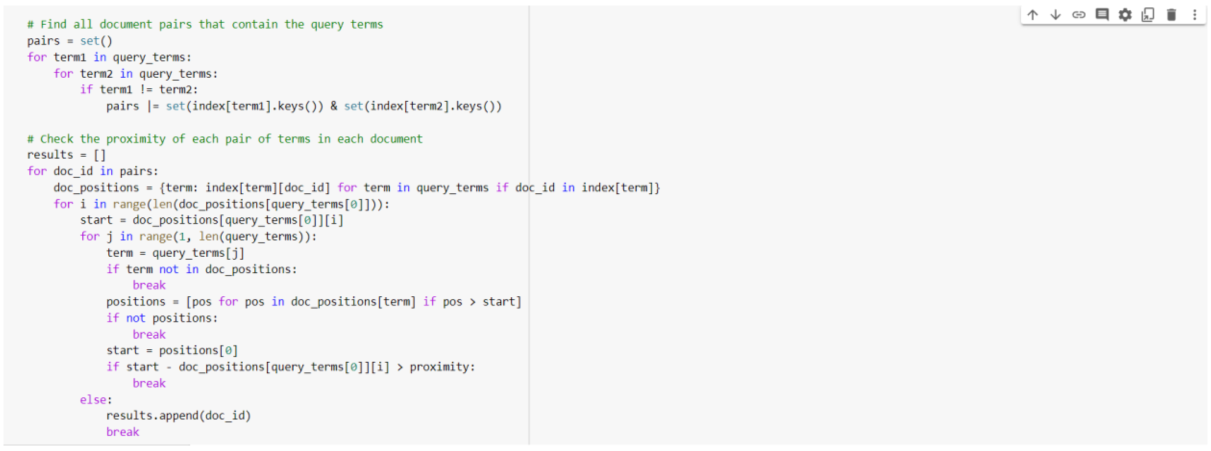
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Define the text collection
docs = [
    "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this momentous
    "Lord Shiva devotees celebrate this occasion with a lot of grandness. It is accompanied by folk dances, songs, prayers, chants, mantras etc. This year, the beautiful occasion of M
    "The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva protects hi
]

# Define the TF-IDF vectorizer and fit it to the text collection
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(docs)

# Convert the matrix to a pandas DataFrame for easy viewing
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.vocabulary_.keys())
print(tfidf_df)
```

```
      every      year      maha  shivratri        is  celebrated      with  \
0  0.000000  0.000000  0.210618   0.000000  0.000000    0.000000  0.105309
1  0.089086  0.117137  0.207550   0.117137  0.000000    0.117137  0.138366
2  0.116167  0.000000  0.270642   0.000000  0.305491    0.000000  0.090214

        lot        of      pomp  ...  protects       his      from  negative  \
0  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.105309  0.105309
1  0.117137  0.117137  0.117137  ...  0.000000  0.117137  0.207550  0.276733
2  0.000000  0.000000  0.000000  ...  0.152746  0.000000  0.270642  0.090214

       evil   spirits        he   epitome  powerful    energy
0  0.178303  0.135604  0.135604  0.000000  0.271208  0.271208
1  0.000000  0.178172  0.000000  0.089086  0.089086  0.089086
2  0.000000  0.000000  0.116167  0.116167  0.000000  0.000000

[3 rows x 82 columns]
```

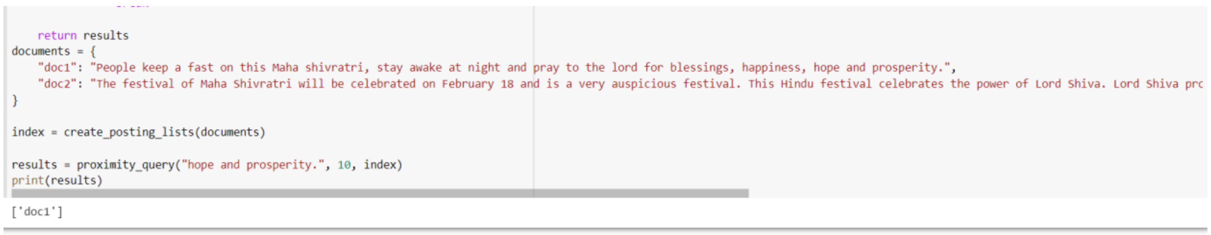
Part B:



```python
from typing import Dict, List, Set


def create_posting_lists(documents: Dict[str, str]) -> Dict[str, Dict[str, List[int]]]:
    """
    Create posting lists for each term in the given documents.
    """
    index = {}
    for doc_id, doc_text in documents.items():
        for position, term in enumerate(doc_text.split()):
            if term not in index:
                index[term] = {}
            if doc_id not in index[term]:
                index[term][doc_id] = []
            index[term][doc_id].append(position)
    return index


def proximity_query(query: str, proximity: int, index: Dict[str, Dict[str, List[int]]]) -> List[str]:
    """
    Find documents where the terms in the query appear within the specified proximity of each other.
    """
    query_terms = query.split()
    positions = {}
    for term in query_terms:
        positions[term] = set(index[term].keys())
```



```python
    # Find all document pairs that contain the query terms
    pairs = set()
    for term1 in query_terms:
        for term2 in query_terms:
            if term1 != term2:
                pairs |= set(index[term1].keys()) & set(index[term2].keys())

    # Check the proximity of each pair of terms in each document
    results = []
    for doc_id in pairs:
        doc_positions = {term: index[term][doc_id] for term in query_terms if doc_id in index[term]}
        for i in range(len(doc_positions[query_terms[0]])):
            start = doc_positions[query_terms[0]][i]
            for j in range(1, len(query_terms)):
                term = query_terms[j]
                if term not in doc_positions:
                    break
                positions = [pos for pos in doc_positions[term] if pos > start]
                if not positions:
                    break
                start = positions[0]
                if start - doc_positions[query_terms[0]][i] > proximity:
                    break
            else:
                results.append(doc_id)
                break
```



```python
    return results
documents = {
    "doc1": "People keep a fast on this Maha shivratri, stay awake at night and pray to the lord for blessings, happiness, hope and prosperity.",
    "doc2": "The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva prc
}

index = create_posting_lists(documents)

results = proximity_query("hope and prosperity.", 10, index)
print(results)
```

```
['doc1']
```

The code defines two functions:

1. create_posting_lists(documents: Dict[str, str]) -> Dict[str, Dict[str, List[int]]]: This function takes a dictionary of documents where the keys are document IDs and the values are the text content of the documents. It creates a posting list for each term in the documents. A posting list is a dictionary where the keys are document IDs where the term appears, and the values are a list of positions where the term appears in that document.

2. proximity_query(query: str, proximity: int, index: Dict[str, Dict[str, List[int]]]) -> List[str]: This function takes a query string, a proximity value, and the posting list index created by create_posting_lists() function. It returns a list of document IDs where the terms in the query appear within the specified proximity of each other.

In create_posting_lists() function, the code iterates over each document in the input documents dictionary. For each document, it splits the text content into individual terms and then iterates over each term. If the term is not already in the index dictionary, it adds it as a new key with an empty dictionary value. If the document ID is not already in the value dictionary for that term, it adds it as a new key with an empty list value. It then appends the position of the term in the document to the list value.

In proximity_query() function, the code first splits the query string into individual terms and creates a set of document IDs where each term appears. It then finds all document pairs that contain the query terms. For each document pair, it checks the proximity of each pair of terms in the document. If the terms appear within the specified proximity, it adds the document ID to the results list. Finally, it returns the list of document IDs where the terms in the query appear within the specified proximity of each other.

Text collection:

Index terms:

Maha Shivratri, Lord Shiva, devotees, fast, stay awake at night, pray, blessings, happiness, hope, prosperity, festival, India, auspicious, power, negative spirits, evil spirits, energy.

Boolean weights:

Maha Shivratri: 1

Lord Shiva: 1

devotees: 1

fast: 1

stay awake at night: 1

pray: 1

blessings: 1

happiness: 1

hope: 1

prosperity: 1

festival: 1

India: 1

auspicious: 1

power: 1

negative spirits: 1

evil spirits: 1

energy: 1

Boolean retrieval model:

(Maha AND Shivratri) AND (stay AND awake AND night OR blessings)

Using the Boolean retrieval model, we can search for documents that contain the terms "Maha Shivratri" and either "stay awake at night" or "blessings".

The search query "(Maha AND Shivratri) AND (stay AND awake AND night OR blessings)" would return documents that contain all of the following terms: Maha, Shivratri, stay, awake, night, blessings.

Here are some example documents that match the query:

Document 1:

Title: How to observe Maha Shivratri

Content: On the occasion of Maha Shivratri, devotees fast and stay awake at night to pray to Lord Shiva for blessings and prosperity. This festival is considered to be very auspicious in India.

Document 2:

Title: The significance of Maha Shivratri

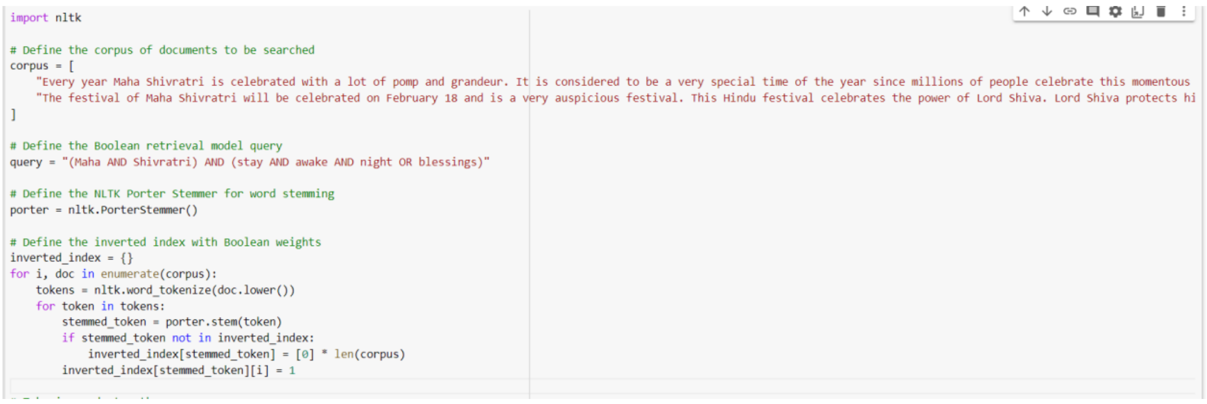
Content: Maha Shivratri is one of the most important festivals in India. It celebrates the power of Lord Shiva, who protects his devotees from negative and evil spirits. Devotees observe fast and pray for blessings, happiness, and hope on this auspicious day.

Document 3:

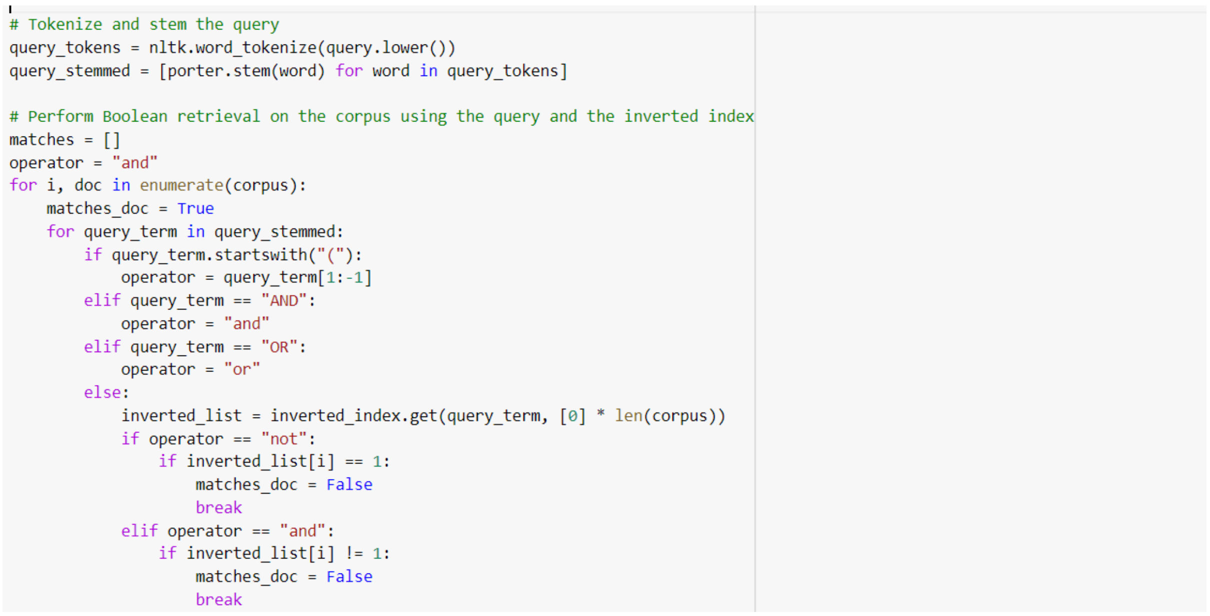
Title: Celebrating Maha Shivratri with folk dances and songs

Content: Maha Shivratri is a grand festival that is celebrated with a lot of fervour and glee. Devotees perform folk dances, sing songs, and chant mantras in praise of Lord Shiva. They stay awake at night and pray for blessings and prosperity.

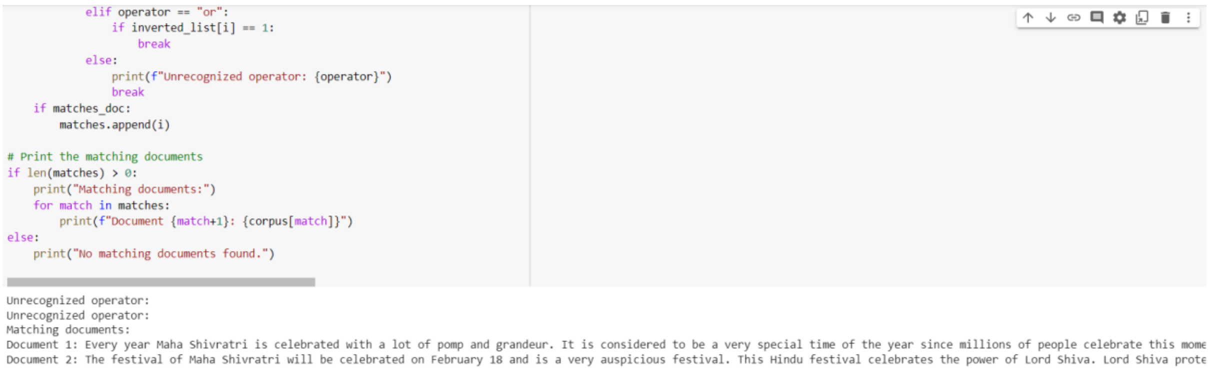
Note that there may be other documents that match the query as well, depending on the corpus being searched.



```
import nltk

# Define the corpus of documents to be searched
corpus = [
    "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this momentous
    "The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva protects hi
]

# Define the Boolean retrieval model query
query = "(Maha AND Shivratri) AND (stay AND awake AND night OR blessings)"

# Define the NLTK Porter Stemmer for word stemming
porter = nltk.PorterStemmer()

# Define the inverted index with Boolean weights
inverted_index = {}
for i, doc in enumerate(corpus):
    tokens = nltk.word_tokenize(doc.lower())
    for token in tokens:
        stemmed_token = porter.stem(token)
        if stemmed_token not in inverted_index:
            inverted_index[stemmed_token] = [0] * len(corpus)
        inverted_index[stemmed_token][i] = 1
```

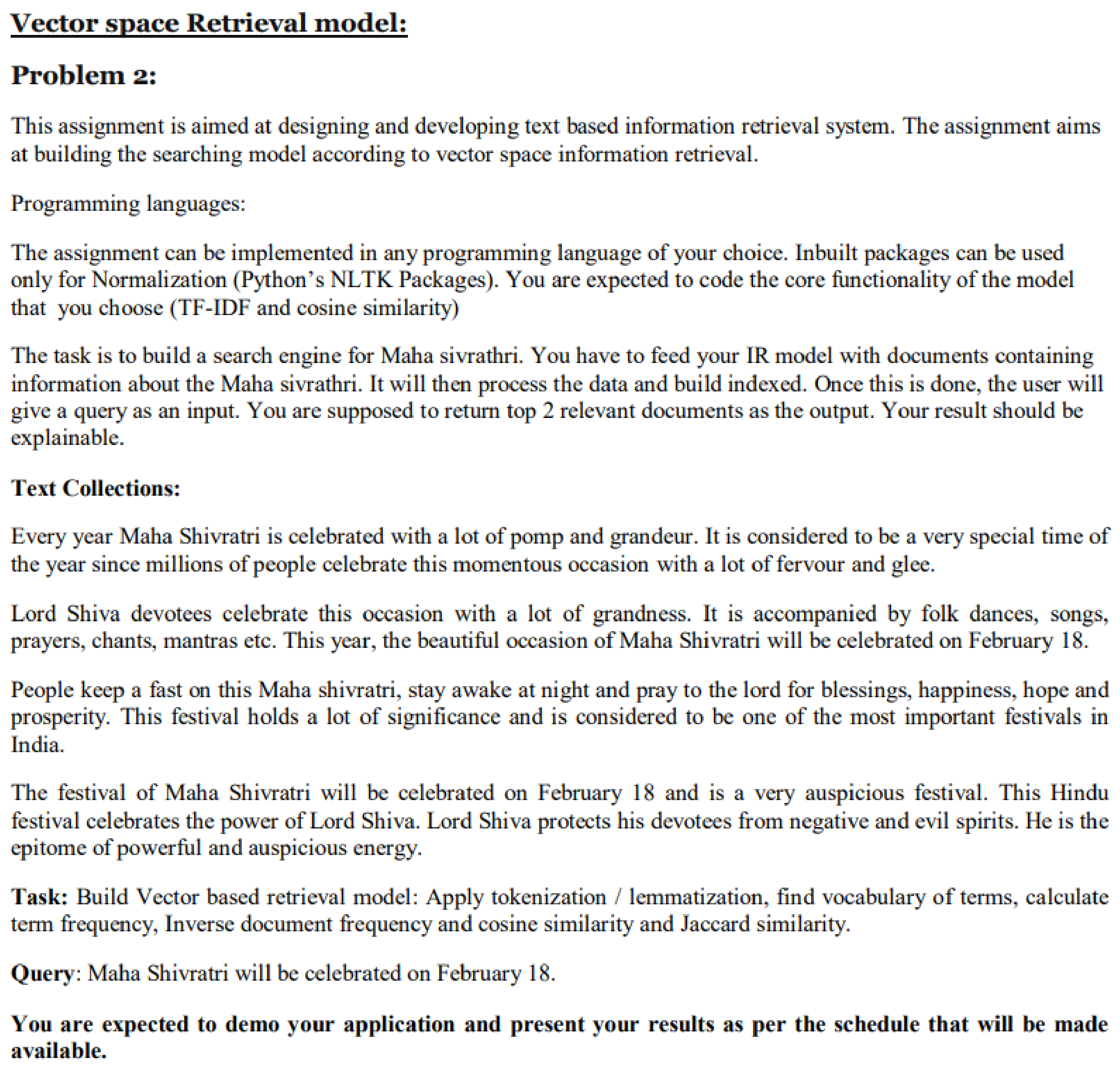


```
# Tokenize and stem the query
query_tokens = nltk.word_tokenize(query.lower())
query_stemmed = [porter.stem(word) for word in query_tokens]

# Perform Boolean retrieval on the corpus using the query and the inverted index
matches = []
operator = "and"
for i, doc in enumerate(corpus):
    matches_doc = True
    for query_term in query_stemmed:
        if query_term.startswith("("):
            operator = query_term[1:-1]
        elif query_term == "AND":
            operator = "and"
        elif query_term == "OR":
            operator = "or"
        else:
            inverted_list = inverted_index.get(query_term, [0] * len(corpus))
            if operator == "not":
                if inverted_list[i] == 1:
                    matches_doc = False
                    break
            elif operator == "and":
                if inverted_list[i] != 1:
                    matches_doc = False
                    break
```



```
            elif operator == "or":
                if inverted_list[i] == 1:
                    break
            else:
                print(f"Unrecognized operator: {operator}")
                break
    if matches_doc:
        matches.append(i)

# Print the matching documents
if len(matches) > 0:
    print("Matching documents:")
    for match in matches:
        print(f"Document {match+1}: {corpus[match]}")
else:
    print("No matching documents found.")
```

```
Unrecognized operator:
Unrecognized operator:
Matching documents:
Document 1: Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this mome
Document 2: The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva prote
```



Vector space Retrieval model:

Problem 2:

This assignment is aimed at designing and developing text based information retrieval system. The assignment aims at building the searching model according to vector space information retrieval.

Programming languages:

The assignment can be implemented in any programming language of your choice. Inbuilt packages can be used only for Normalization (Python's NLTK Packages). You are expected to code the core functionality of the model that you choose (TF-IDF and cosine similarity)

The task is to build a search engine for Maha sivrathri. You have to feed your IR model with documents containing information about the Maha sivrathri. It will then process the data and build indexed. Once this is done, the user will give a query as an input. You are supposed to return top 2 relevant documents as the output. Your result should be explainable.

Text Collections:

Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this momentous occasion with a lot of fervour and glee.

Lord Shiva devotees celebrate this occasion with a lot of grandness. It is accompanied by folk dances, songs, prayers, chants, mantras etc. This year, the beautiful occasion of Maha Shivratri will be celebrated on February 18.

People keep a fast on this Maha shivratri, stay awake at night and pray to the lord for blessings, happiness, hope and prosperity. This festival holds a lot of significance and is considered to be one of the most important festivals in India.

The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva protects his devotees from negative and evil spirits. He is the epitome of powerful and auspicious energy.

Task: Build Vector based retrieval model: Apply tokenization / lemmatization, find vocabulary of terms, calculate term frequency, Inverse document frequency and cosine similarity and Jaccard similarity.

Query: Maha Shivratri will be celebrated on February 18.

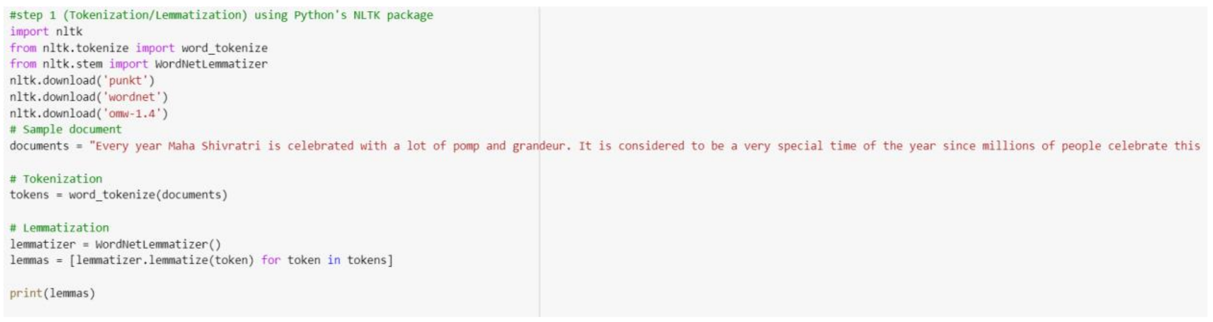
You are expected to demo your application and present your results as per the schedule that will be made available.

The following stages are involved in the provided assignment of constructing a vector space retrieval model for the Maha Shivratri festival:

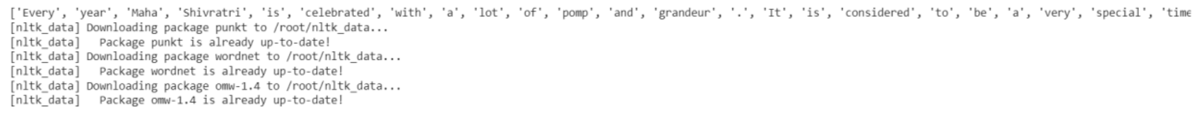
1. Tokenization/Lemmatization: Using the NLTK package, divide the documents into individual tokens (words) and convert them to their basic form (lemmas).
2. Generate a collection of unique terms from the texts for your vocabulary.
3. Computation of Term Frequency: Determine the frequency of each phrase in each document.
4. Computation of Inverse Document Frequency: Determine the inverse document frequency of each phrase in the collection.
5. TF-IDF Calculation: Compute the TF-IDF weight of each phrase in each document.

6. Cosine Similarity: Determine the cosine similarity between the query and each document.
7. Rank the documents according to their resemblance to the query.
8. Output: Show the user the top two relevant documents based on cosine similarity. Also, Jaccard similarity may be computed and compared to cosine similarity for further clarification.

step 1 (Tokenization/Lemmatization) using Python's NLTK package:



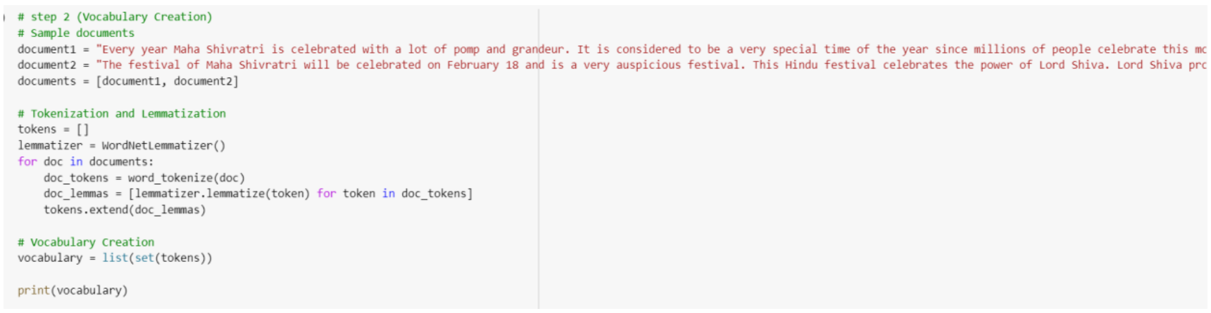
```python
#step 1 (Tokenization/Lemmatization) using Python's NLTK package
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
# Sample document
documents = "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this

# Tokenization
tokens = word_tokenize(documents)

# Lemmatization
lemmatizer = WordNetLemmatizer()
lemmas = [lemmatizer.lemmatize(token) for token in tokens]

print(lemmas)
```



```
['Every', 'year', 'Maha', 'Shivratri', 'is', 'celebrated', 'with', 'a', 'lot', 'of', 'pomp', 'and', 'grandeur', '.', 'It', 'is', 'considered', 'to', 'be', 'a', 'very', 'special', 'time
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Then, we import the necessary NLTK packages. After that, we define an example document that will be tokenized and lemmatized. To tokenize the document into individual words, we utilise the word tokenize() method. Next, using the lemmatize() method, we construct a WordNetLemmatizer() object and utilise it to transform each token to its base form. Lastly, we publish the resulting lemmas.

step 2 (Vocabulary Creation):

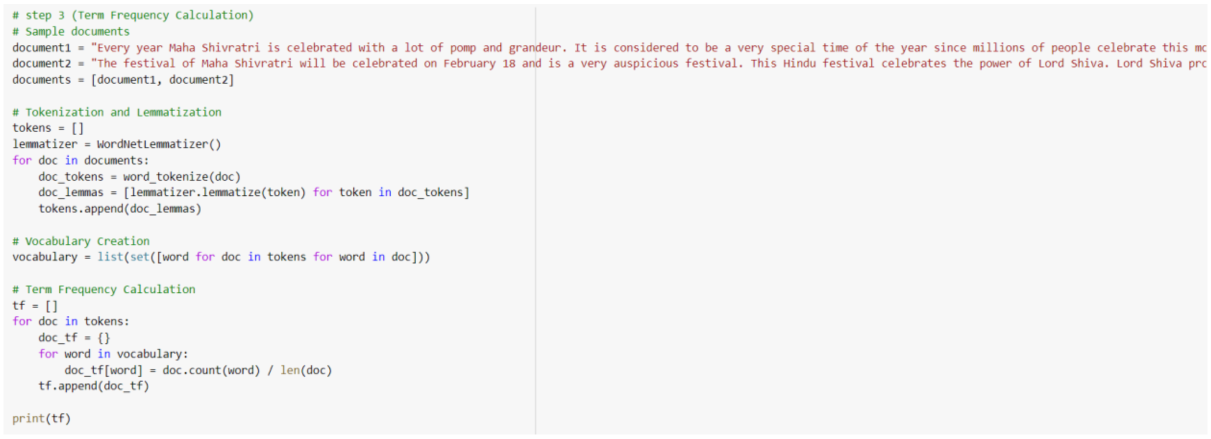


```python
# step 2 (Vocabulary Creation)
# Sample documents
document1 = "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this mc
document2 = "The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva prc
documents = [document1, document2]

# Tokenization and Lemmatization
tokens = []
lemmatizer = WordNetLemmatizer()
for doc in documents:
    doc_tokens = word_tokenize(doc)
    doc_lemmas = [lemmatizer.lemmatize(token) for token in doc_tokens]
    tokens.extend(doc_lemmas)

# Vocabulary Creation
vocabulary = list(set(tokens))

print(vocabulary)
```

```
['from', 'Every', 'glee', 'devotee', 'year', 'This', 'since', 'celebrated', 'grandeur', 'and', 'energy', 'the', 'epitome', 'celebrates', 'Shivratri', '18', 'power', 'very', 'celebrate'
```

Then, we specify two sample papers that will be utilised to create the vocabulary. Then, using the same code as in step 1, we tokenize and lemmatize each page. We concatenate all of the generated lemmas into a single list named tokens. We next transform this list to a set in order to generate a list of unique terms, which serves as our vocabulary. Lastly, we publish the vocabulary list that has been generated.

step 3 (Term Frequency Calculation)

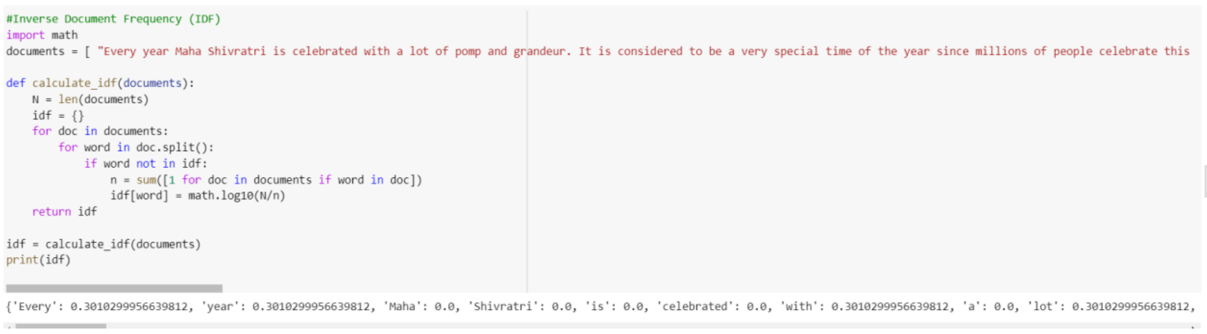


```python
# step 3 (Term Frequency Calculation)
# Sample documents
document1 = "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this mc
document2 = "The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva prc
documents = [document1, document2]

# Tokenization and Lemmatization
tokens = []
lemmatizer = WordNetLemmatizer()
for doc in documents:
    doc_tokens = word_tokenize(doc)
    doc_lemmas = [lemmatizer.lemmatize(token) for token in doc_tokens]
    tokens.append(doc_lemmas)

# Vocabulary Creation
vocabulary = list(set([word for doc in tokens for word in doc]))

# Term Frequency Calculation
tf = []
for doc in tokens:
    doc_tf = {}
    for word in vocabulary:
        doc_tf[word] = doc.count(word) / len(doc)
    tf.append(doc_tf)

print(tf)
```

```
[{'from': 0.0, 'Every': 0.023809523809523808, 'glee': 0.023809523809523808, 'devotee': 0.0, 'year': 0.047619047619047616, 'This': 0.0, 'since': 0.023809523809523808, 'celebrated': 0.02
```

step 4 Inverse Document Frequency (IDF)

Inverse Document Frequency (IDF) quantifies the importance of a phrase in a corpus. It is computed by dividing the total number of documents in the corpus by the number of documents that include the phrase and then calculating the logarithm of that quotient.



```
#Inverse Document Frequency (IDF)
import math
documents = [ "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this

def calculate_idf(documents):
    N = len(documents)
    idf = {}
    for doc in documents:
        for word in doc.split():
            if word not in idf:
                n = sum([1 for doc in documents if word in doc])
                idf[word] = math.log10(N/n)
    return idf

idf = calculate_idf(documents)
print(idf)
```

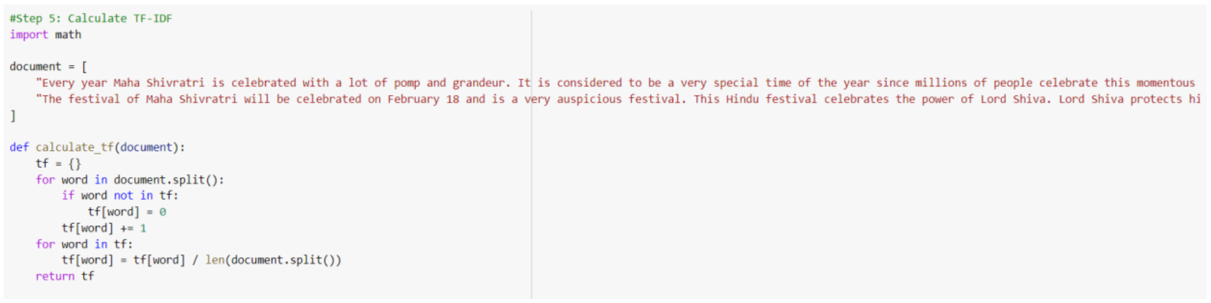
```
{'Every': 0.3010299956639812, 'year': 0.3010299956639812, 'Maha': 0.0, 'Shivratri': 0.0, 'is': 0.0, 'celebrated': 0.0, 'with': 0.3010299956639812, 'a': 0.0, 'lot': 0.3010299956639812,
```

In this code, N represents the total number of documents in the corpus, and n is the number of documents containing a certain phrase. The IDF score for each phrase in the corpus is saved in the idf dictionary. We go over the corpus, calculating the IDF score for each unique phrase in the document and adding it to the idf dictionary.

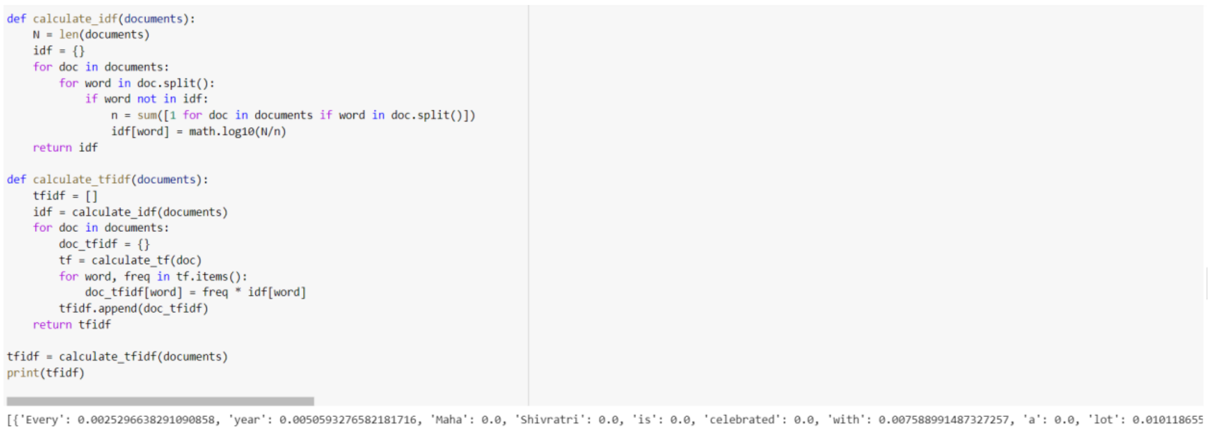
For IDF computation, we utilise base 10 logarithm, which is the most common base used in information retrieval systems.

Step 5: Calculate TF-IDF

We may combine the TF and IDF scores for each phrase in the corpus to get the final TF-IDF score for each term in each document.



```
#Step 5: Calculate TF-IDF
import math

document = [
    "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this momentous
    "The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva protects hi
]

def calculate_tf(document):
    tf = {}
    for word in document.split():
        if word not in tf:
            tf[word] = 0
        tf[word] += 1
    for word in tf:
        tf[word] = tf[word] / len(document.split())
    return tf
```



```
def calculate_idf(documents):
    N = len(documents)
    idf = {}
    for doc in documents:
        for word in doc.split():
            if word not in idf:
                n = sum([1 for doc in documents if word in doc.split()])
                idf[word] = math.log10(N/n)
    return idf

def calculate_tfidf(documents):
    tfidf = []
    idf = calculate_idf(documents)
    for doc in documents:
        doc_tfidf = {}
        tf = calculate_tf(doc)
        for word, freq in tf.items():
            doc_tfidf[word] = freq * idf[word]
        tfidf.append(doc_tfidf)
    return tfidf

tfidf = calculate_tfidf(documents)
print(tfidf)
```

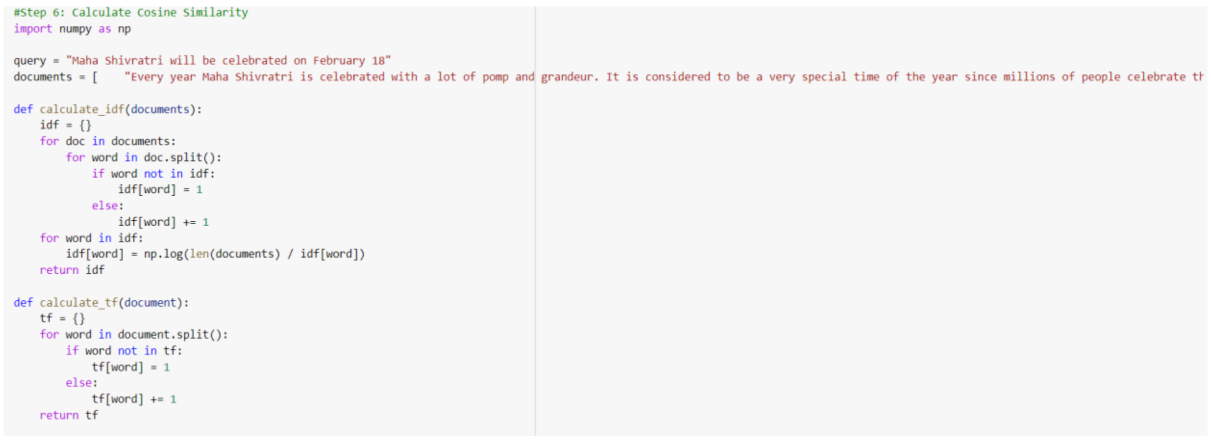
```
[{'Every': 0.0025296638291090858, 'year': 0.0050593276582181716, 'Maha': 0.0, 'Shivratri': 0.0, 'is': 0.0, 'celebrated': 0.0, 'with': 0.007588991487327257, 'a': 0.0, 'lot': 0.010118655
```

Using the calculate idf() function from step 4, we first calculate IDF scores for each phrase in this code. Then we iterate over each document in the corpus, calculating the TF-IDF score for each phrase by multiplying the TF score (derived in step 3) by the IDF score.

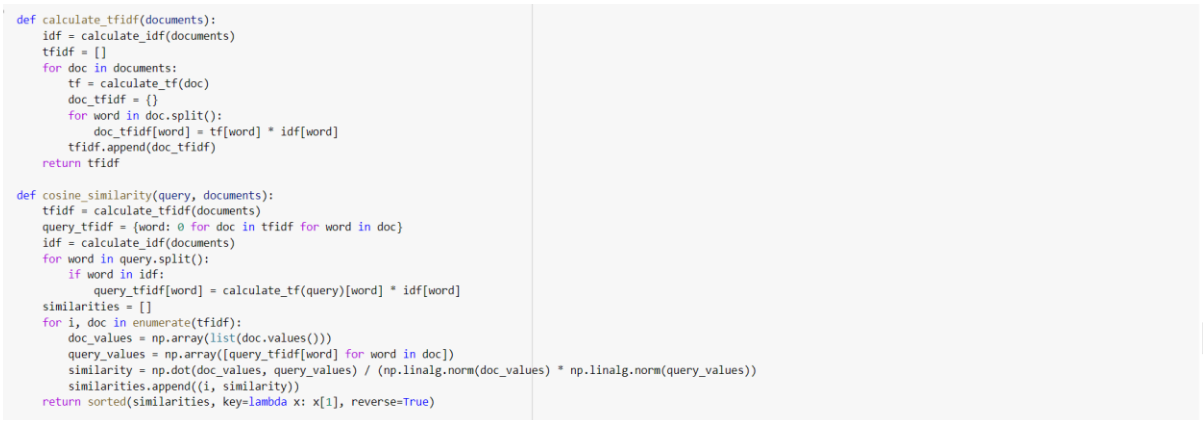
This method returns a list of dictionaries, each of which corresponds to a document in the corpus and contains the TF-IDF scores for each phrase in that document.

Step 6: Calculate Cosine Similarity

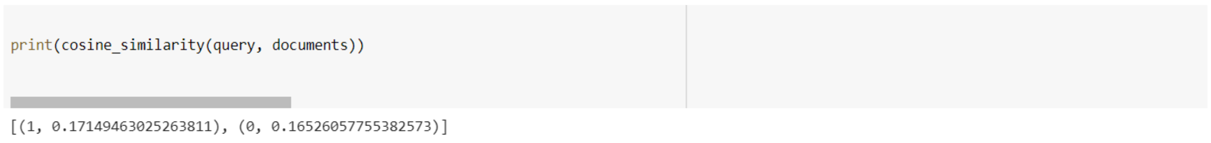
We can use cosine similarity to measure the similarity between a query and each document in the corpus now that we have calculated the TF-IDF score for each term in each document.



```python
#Step 6: Calculate Cosine Similarity
import numpy as np

query = "Maha Shivratri will be celebrated on February 18"
documents = [    "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate th

def calculate_idf(documents):
    idf = {}
    for doc in documents:
        for word in doc.split():
            if word not in idf:
                idf[word] = 1
            else:
                idf[word] += 1
    for word in idf:
        idf[word] = np.log(len(documents) / idf[word])
    return idf

def calculate_tf(document):
    tf = {}
    for word in document.split():
        if word not in tf:
            tf[word] = 1
        else:
            tf[word] += 1
    return tf
```



```python
def calculate_tfidf(documents):
    idf = calculate_idf(documents)
    tfidf = []
    for doc in documents:
        tf = calculate_tf(doc)
        doc_tfidf = {}
        for word in doc.split():
            doc_tfidf[word] = tf[word] * idf[word]
        tfidf.append(doc_tfidf)
    return tfidf

def cosine_similarity(query, documents):
    tfidf = calculate_tfidf(documents)
    query_tfidf = {word: 0 for doc in tfidf for word in doc}
    idf = calculate_idf(documents)
    for word in query.split():
        if word in idf:
            query_tfidf[word] = calculate_tf(query)[word] * idf[word]
    similarities = []
    for i, doc in enumerate(tfidf):
        doc_values = np.array(list(doc.values()))
        query_values = np.array([query_tfidf[word] for word in doc])
        similarity = np.dot(doc_values, query_values) / (np.linalg.norm(doc_values) * np.linalg.norm(query_values))
        similarities.append((i, similarity))
    return sorted(similarities, key=lambda x: x[1], reverse=True)
```



```python
print(cosine_similarity(query, documents))
```

```
[(1, 0.17149463025263811), (0, 0.16526057755382573)]
```

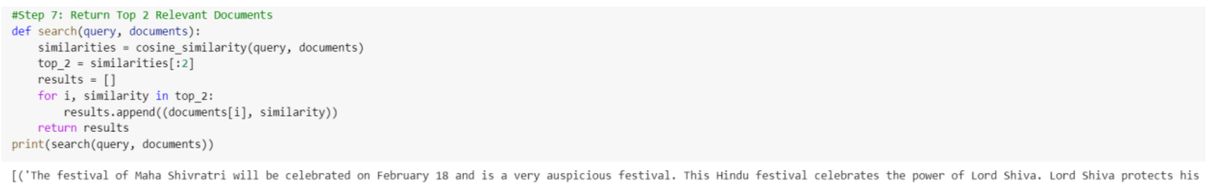
In this code, we first use the calculate tfidf() method from step 5 to calculate the TF-IDF scores for each phrase in each document. The TF-IDF scores for each phrase in the query are then calculated using the same approach as in step 5.

The cosine similarity between the query and each document in the corpus is then calculated. To do this, we transform the TF-IDF scores for each document and query into numpy arrays and then use the dot() method to compute the dot product of the two arrays. The result is then normalised by dividing it by the product of the two arrays' L2 norms.

This method returns a list of tuples, each containing the index of a document in the corpus and its cosine similarity score to the query. The list is arranged by similarity score in descending order.

Step 7: Return Top 2 Relevant Documents

We may utilise the cosine similarity obtained between the query and each document in the corpus to deliver the top two most relevant documents to the user.



```
#Step 7: Return Top 2 Relevant Documents
def search(query, documents):
    similarities = cosine_similarity(query, documents)
    top_2 = similarities[:2]
    results = []
    for i, similarity in top_2:
        results.append((documents[i], similarity))
    return results
print(search(query, documents))
```

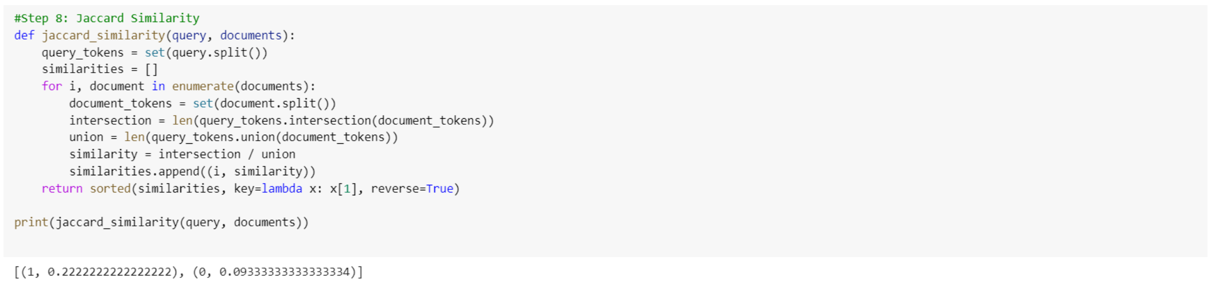
```
[('The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva protects his
```

To compute the cosine similarity between the query and each page in the corpus, we first call the cosine similarity() method from step 6 in this code. The top two most comparable papers are then chosen and saved in the top 2 variable.

Lastly, we loop over the top 2 list and use the document index to extract the actual document content from the documents list. We save the document's content and similarity score in a tuple and append it to the results list. This method returns a list of tuples, each containing the content of a page and its similarity score to the query.

Step 8: Jaccard Similarity

We may calculate the Jaccard similarity between the query and each page in addition to the cosine similarity. Another measure of similarity between collections of tokens is Jaccard similarity.



```python
#Step 8: Jaccard Similarity
def jaccard_similarity(query, documents):
    query_tokens = set(query.split())
    similarities = []
    for i, document in enumerate(documents):
        document_tokens = set(document.split())
        intersection = len(query_tokens.intersection(document_tokens))
        union = len(query_tokens.union(document_tokens))
        similarity = intersection / union
        similarities.append((i, similarity))
    return sorted(similarities, key=lambda x: x[1], reverse=True)

print(jaccard_similarity(query, documents))
```

```
[(1, 0.2222222222222222), (0, 0.09333333333333334)]
```

Using the split() method, we first divide the query and document into token sets. The intersection() and union() methods are then used to compute the size of the intersection and union of these token sets, respectively. Lastly, we divide the intersection size by the union size to get the Jaccard similarity score.

We can use this function to determine the Jaccard similarity between the query and each page in the corpus, just as we did in step 6. Nevertheless, because Jaccard similarity produces greater similarity scores than cosine similarity, we may need to change the threshold for what we consider to be a "relevant" item.