```
DS321 IR Project
     51S
   DS Group
```

Project Overview: This project develops an Information Retrieval (IR) system that categorizes documents (e.g., News, Business, Lifestyle) and retrieves relevant results based on user queries. It preprocesses documents by removing stop words, and applying TF-IDF vectorization. Queries are converted to TF-IDF vectors, and cosine similarity ranks documents by relevance. Tools like NLTK or spaCy for preprocessing and scikit-learn for vectorization and similarity calculation are used.

IMPORT NEEDED LIBRARY

```
pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\lubna\anaconda3\lib\site-packages (3.7)
Requirement already satisfied: tqdm in c:\users\lubna\anaconda3\lib\site-packages (from nltk) (4.64.1)
Requirement already satisfied: regex>=2021.8.3 in c:\users\lubna\anaconda3\lib\site-packages (from nltk) (2022.7.9)
Requirement already satisfied: joblib in c:\users\lubna\anaconda3\lib\site-packages (from nltk) (1.4.2)
Requirement already satisfied: click in c:\users\lubna\anaconda3\lib\site-packages (from nltk) (8.0.4)
Requirement already satisfied: colorama in c:\users\lubna\anaconda3\lib\site-packages (from click->nltk) (0.4.5)
Note: you may need to restart the kernel to use updated packages.
```

```python
# Import necessary libraries
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
import numpy as np
import pandas as pd
from collections import Counter  # For counting word occurrences in each document
from math import log  # For calculating the log function in IDF calculation
from sklearn.metrics.pairwise import cosine_similarity  #This function computes the cosine similarity between two sets of vectors.
from sklearn.feature_extraction.text import TfidfVectorizer #This is a utility from scikit-learn that converts a collection of raw documents into a matrix of TF-IDF features.
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Lubna\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

THE GENERATED DOCUMENTS

FROM NEWS-49 , BUSINESS-7 , METRO-25 , LIFESTYLE-245

```
documents = [
    "Web Bytes chief executive  expressed confidence that the company is halfway towards achieving its goal.",
    "Dissanayake expressed confidence that he can lead the fight against corruption at risk.",
    "Amran expressed determination to fulfill his duties even at risk.",
    "Denis has expressed that took on the responsibility of supporting his family."
]
```

1- PROPOSE A QUERY

```
query= "expressing concern about the risk"
```

2- THE CORPUS

compines both the query and the documents

```
corpus = documents + [query]
```

3- PREPROCESS THE CORPUS

by removing stop words and converting words to lowercase

load the set of English stop words for text PROCESSING, whith will help filter the stop words (ex. the ,is ,about ,etc.)

```
stop_words=set(stopwords.words('english'))

 # Join the words that are not in the stop words list
preprocessed_corpus=[' '.join([word.lower() for word in doc.split() if word.lower() not in stop_words])
                     for doc in corpus]

preprocessed_corpus
```

```
['web bytes chief executive expressed confidence company halfway towards achieving goal.',
 'dissanayake expressed confidence lead fight corruption risk.',
 'amran expressed determination fulfill duties even risk.',
 'denis expressed took responsibility supporting family.',
 'expressing concern risk']
```

4- Calculating TF-IDF

w = log(1+tf) x log10 (N/ df)

Term Frequency - Inverse Document Frequency (TF-IDF) is a most commen statistical method in natural language processing and information retrieval. It measures how important a term is within a document relative to a collection of documents

it increses with number of occurrences within a document and Increases with the rarity of the term in the collection

a- calculating TF

Term Frequency (TF) measures the frequency of a word in a document.

It is calculated as the number of times a word appears in a document divided by the total number of words in that document.

In this step, we will define a function `compute_tf` that takes the corpus as input and returns the TF values for each document.

```python
def compute_tf(corpus):
    # Initialize an empty list to store the TF values for each document
    tf_list = []

    # Loop through each document in the corpus
    for document in corpus:
        # Split the document into individual words and count occurrences of each word
        word_count = Counter(document.split())

        # Calculate the total number of words in the document
        doc_len = len(document.split())

        # Calculate the term frequency (TF) for each word
        # TF is the count of each word divided by the total word count in the document
        tf = {word: count / doc_len for word, count in word_count.items()}

        # Append the TF dictionary for this document to the list of TF dictionaries
        tf_list.append(tf)

    # Return the list of TF dictionaries, each representing a document
    return tf_list

# Run the TF calculation
tf_scores = compute_tf(preprocessed_corpus)

# Convert the TF scores into a DataFrame with words as rows and
documents as columns
df_tf = pd.DataFrame(tf_scores).T  # Transpose to get terms as rows

# Set custom column labels as 'Document 1', 'Document 2', etc.
df_tf.columns = [f"Document {i+1}" for i in
range(len(preprocessed_corpus))]

# Print the Term Frequency (TF) scores in a table format with terms on
the left and TF on the right
print("Term Frequency (TF) for each document:\n")
print(df_tf.fillna(0))  # Fill NaN with 0 for words that don't appear
in a document
```

```
Term Frequency (TF) for each document:

                Document 1  Document 2  Document 3  Document 4
Document 5
web               0.090909    0.000000    0.000000    0.000000
0.000000
bytes             0.090909    0.000000    0.000000    0.000000
0.000000
chief             0.090909    0.000000    0.000000    0.000000
0.000000
executive         0.090909    0.000000    0.000000    0.000000
0.000000
expressed         0.090909    0.142857    0.142857    0.166667
0.000000
confidence        0.090909    0.142857    0.000000    0.000000
0.000000
company           0.090909    0.000000    0.000000    0.000000
0.000000
halfway           0.090909    0.000000    0.000000    0.000000
0.000000
towards           0.090909    0.000000    0.000000    0.000000
0.000000
achieving         0.090909    0.000000    0.000000    0.000000
0.000000
goal.             0.090909    0.000000    0.000000    0.000000
0.000000
dissanayake       0.000000    0.142857    0.000000    0.000000
0.000000
```

```
lead                  0.000000     0.142857     0.000000     0.000000
0.000000
fight                 0.000000     0.142857     0.000000     0.000000
0.000000
corruption            0.000000     0.142857     0.000000     0.000000
0.000000
risk.                 0.000000     0.142857     0.142857     0.000000
0.000000
amran                 0.000000     0.000000     0.142857     0.000000
0.000000
determination         0.000000     0.000000     0.142857     0.000000
0.000000
fulfill               0.000000     0.000000     0.142857     0.000000
0.000000
duties                0.000000     0.000000     0.142857     0.000000
0.000000
even                  0.000000     0.000000     0.142857     0.000000
0.000000
denis                 0.000000     0.000000     0.000000     0.166667
0.000000
took                  0.000000     0.000000     0.000000     0.166667
0.000000
responsibility        0.000000     0.000000     0.000000     0.166667
0.000000
supporting            0.000000     0.000000     0.000000     0.166667
0.000000
family.               0.000000     0.000000     0.000000     0.166667
0.000000
expressing            0.000000     0.000000     0.000000     0.000000
0.333333
concern               0.000000     0.000000     0.000000     0.000000
0.333333
risk                  0.000000     0.000000     0.000000     0.000000
0.333333
```

b- calculating DF

Document Frequency (DF) is the number of documents in which a word appears.

In this function `compute_df`, we calculate DF across all documents to understand how common each word is in corpus.

This step is essential because it helps us compute the Inverse Document Frequency (IDF) later on.

```
def compute_df(corpus):
    # Initialize an empty dictionary to store document frequency for
each word
    df = {}
    # Loop through each document in the corpus
    for document in corpus:
        # Use a set to get unique words in the document (to avoid
counting duplicates within a document)
        for word in set(document.split()):
            # Increase the count for the word in DF dictionary if it
appears in the document
            # .get(word, 0) returns the current count for the word or
0 if it’s not in the dictionary yet
            df[word] = df.get(word, 0) + 1

    # Return the DF dictionary containing the document frequency for
each word
    return df

# Run the DF calculation
df_scores = compute_df(preprocessed_corpus)

# Convert the DF scores into a DataFrame with terms as rows and DF as
the values
df_df = pd.DataFrame(list(df_scores.items()), columns=['Term',
'Document Frequency (DF)'])

# Print the Document Frequency (DF) scores with the terms aligned on
the left and DF on the right
print("Document Frequency (DF) for each term:\n")

# Align terms to the left and DF values to the right
for index, row in df_df.iterrows():
    print(f"{row['Term']: <15} {row['Document Frequency (DF)']: >3}")
```

```
Document Frequency (DF) for each term:

achieving         1
chief             1
executive         1
towards           1
company           1
expressed         4
confidence        2
halfway           1
bytes             1
goal.             1
web               1
fight             1
risk.             2
lead              1
corruption        1
dissanayake       1
```

```
even                 1
duties               1
determination        1
fulfill              1
amran                1
supporting           1
denis                1
responsibility       1
family.              1
took                 1
expressing           1
risk                 1
concern              1
```

c- calculating IDF

Inverse Document Frequency (IDF) measure how unique a word is across the entire corpus. The formula for IDF is:

idf = log10 (N/df)

where (N) is the total number of documents, and DF is the document frequency of the word.

We define a function `compute_idf` to calculate IDF for each word.

```python
def compute_idf(corpus):
    # Compute the document frequency for each word by calling
compute_df
    df = compute_df(preprocessed_corpus)

    # Total number of documents in the corpus
    N = len(preprocessed_corpus)

    # Calculate IDF for each word in DF dictionary
    # IDF is calculated as log(N / DF) where N is the total number of
documents
    # and DF is the document frequency of the word
    idf = {word: log(N / df_val) for word, df_val in df.items()}

    # Return the IDF dictionary containing IDF values for each word
    return idf

# Run the IDF calculation
idf_scores = compute_idf(preprocessed_corpus)

# Print the Inverse Document Frequency (IDF) scores with terms aligned
to the left and IDF values on the right
print("Inverse Document Frequency (IDF) for each term:\n")

# Align terms to the left and IDF values to the right
```

```
for word, score in idf_scores.items():
    print(f"{word: <15} {score: .4f}")

Inverse Document Frequency (IDF) for each term:

achieving        1.6094
chief            1.6094
executive        1.6094
towards          1.6094
company          1.6094
expressed        0.2231
confidence       0.9163
halfway          1.6094
bytes            1.6094
goal.            1.6094
web              1.6094
fight            1.6094
risk.            0.9163
lead             1.6094
corruption       1.6094
dissanayake      1.6094
even             1.6094
duties           1.6094
determination    1.6094
fulfill          1.6094
amran            1.6094
supporting       1.6094
denis            1.6094
responsibility   1.6094
family.          1.6094
took             1.6094
expressing       1.6094
risk             1.6094
concern          1.6094
```

d- calculating TF-IDF

Finally, we compute the TF-IDF scores for each word in each document by using this formula:

w = log(1+tf) x log10 (N/ df)

This score reflects the importance of each word in a specific document relative to the entire corpus.

In this step, we define a function `compute_tfidf` that uses both TF and IDF values to calculate the TF-IDF score for each word in each document.

```
def compute_tfidf(corpus):
    # Calculate TF for each document by calling compute_tf
    tf_list = compute_tf(corpus)
    # Calculate IDF for each word across the corpus by calling
compute_idf
    idf = compute_idf(corpus)

    # Initialize an empty list to store TF-IDF values for each
document
    tfidf_list = []

    # Loop through each document's TF dictionary in tf_list
    for tf in tf_list:
        # Calculate TF-IDF for each word in the document
        # TF-IDF is calculated as TF * IDF for each word
        tfidf = {word: tf[word] * idf[word] for word in tf.keys()}

        # Append the TF-IDF dictionary for this document to the TF-IDF
list
        tfidf_list.append(tfidf)

    # Return the list of TF-IDF dictionaries, each representing a
document
    return tfidf_list

# Run the TF-IDF calculation
tfidf_scores = compute_tfidf(preprocessed_corpus)

# Convert the TF-IDF scores into a DataFrame with terms as rows and
documents as columns
df_tfidf = pd.DataFrame(tfidf_scores).T  # Transpose to get terms as
rows
# Set custom column labels as 'Document 1', 'Document 2', etc.
df_tfidf.columns = [f"Document {i+1}" for i in
range(len(preprocessed_corpus))]

# Print the TF-IDF scores in a table format with terms on the left and
TF-IDF on the right
print("TF-IDF scores for each document:\n")
print(df_tfidf.fillna(0))  # Fill NaN with 0 for words that don't
appear in a document
```

```
TF-IDF scores for each document:

                Document 1  Document 2  Document 3  Document 4
Document 5
web               0.146313    0.000000    0.000000    0.000000
0.000000
bytes             0.146313    0.000000    0.000000    0.000000
0.000000
chief             0.146313    0.000000    0.000000    0.000000
0.000000
```

```
executive          0.146313    0.000000    0.000000    0.000000
0.000000
expressed          0.020286    0.031878    0.031878    0.037191
0.000000
confidence         0.083299    0.130899    0.000000    0.000000
0.000000
company            0.146313    0.000000    0.000000    0.000000
0.000000
halfway            0.146313    0.000000    0.000000    0.000000
0.000000
towards            0.146313    0.000000    0.000000    0.000000
0.000000
achieving          0.146313    0.000000    0.000000    0.000000
0.000000
goal.              0.146313    0.000000    0.000000    0.000000
0.000000
dissanayake        0.000000    0.229920    0.000000    0.000000
0.000000
lead               0.000000    0.229920    0.000000    0.000000
0.000000
fight              0.000000    0.229920    0.000000    0.000000
0.000000
corruption         0.000000    0.229920    0.000000    0.000000
0.000000
risk.              0.000000    0.130899    0.130899    0.000000
0.000000
amran              0.000000    0.000000    0.229920    0.000000
0.000000
determination      0.000000    0.000000    0.229920    0.000000
0.000000
fulfill            0.000000    0.000000    0.229920    0.000000
0.000000
duties             0.000000    0.000000    0.229920    0.000000
0.000000
even               0.000000    0.000000    0.229920    0.000000
0.000000
denis              0.000000    0.000000    0.000000    0.268240
0.000000
took               0.000000    0.000000    0.000000    0.268240
0.000000
responsibility     0.000000    0.000000    0.000000    0.268240
0.000000
supporting         0.000000    0.000000    0.000000    0.268240
0.000000
family.            0.000000    0.000000    0.000000    0.268240
0.000000
expressing         0.000000    0.000000    0.000000    0.000000
0.536479
concern            0.000000    0.000000    0.000000    0.000000
```

```
0.536479
risk                0.000000    0.000000    0.000000    0.000000
0.536479
```

5- Ranking using Cosine Similarity

In the cosine similarity step, we will calculates the cosine similarity between a query and a set of documents, allowing you to rank the documents based on their relevance to the query.

cos_sim(A, B) = (A•B) / CALL * ||B|I)

(5)

This will help us to determine how closely related the two vectors are.

```python
# Here, we define the string query which contains the text we want to
compare against the documents.
query = "expressing concern about the risk"

 # Create TF-IDF vectorizer
vectorizer = TfidfVectorizer(stop_words='english')

#This method fits the vectorizer to the entire corpus (documents plus
query) and transforms it into a TF-IDF matrix.
#Each row corresponds to a document (or the query) and each column
corresponds to a unique term.
tfidf_matrix = vectorizer.fit_transform(corpus)

# Calculate cosine similarity between the query and all documents
cosine_similarities = cosine_similarity(tfidf_matrix[-1:],
tfidf_matrix[:-1])

# Print cosine similarities between the query and each document
print("\nCosine Similarities:")
for i, similarity in enumerate(cosine_similarities[0]):
    print(f"Document {i + 1}: {similarity:.4f}")

# Rank documents by cosine similarity
ranked_documents = sorted(enumerate(cosine_similarities[0]),
key=lambda x: x[1], reverse=True)

print("\nRanking of Documents by Cosine Similarity:")
for index, similarity in ranked_documents:
    print(f"Document {index + 1}: {similarity:.4f}")
```

```
Cosine Similarities:
Document 1: 0.0000
Document 2: 0.1232
Document 3: 0.1313
Document 4: 0.0000

Ranking of Documents by Cosine Similarity:
Document 3: 0.1313
Document 2: 0.1232
Document 1: 0.0000
Document 4: 0.0000
```

Extra information

Find the index of the most similar document

```python
# Find the index of the most similar document
most_similar_index = cosine_similarities[0].argmax()
most_similar_score = cosine_similarities[0].max()

# Print the most similar document and its similarity score
print(f"The most similar document is Document {most_similar_index + 1}
with a similarity score of {most_similar_score:.4f}.")
```

```
The most similar document is Document 3 with a similarity score of
0.1313.
```

Heatmap of a Similarity Matrix

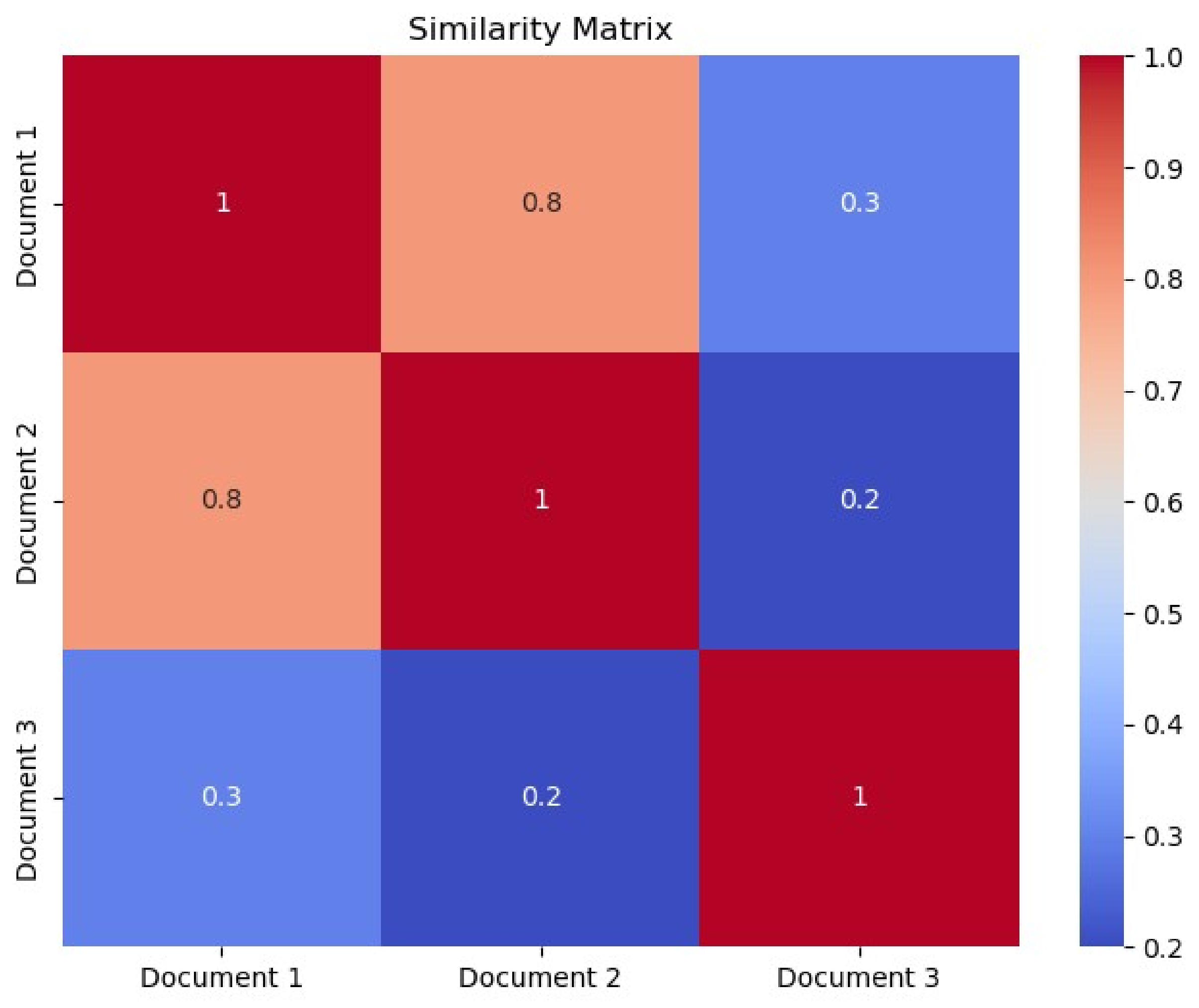
visually highlights document relationships, making it easier to identify relevant documents, assess query results, and optimize retrieval performance.

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Example similarity matrix
similarity_matrix = np.array([[1, 0.8, 0.3], [0.8, 1, 0.2], [0.3, 0.2,
1]])
labels = ['Document 1', 'Document 2', 'Document 3']

# Create the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(similarity_matrix, annot=True, xticklabels=labels,
yticklabels=labels, cmap='coolwarm')
plt.title('Similarity Matrix')
plt.show
```

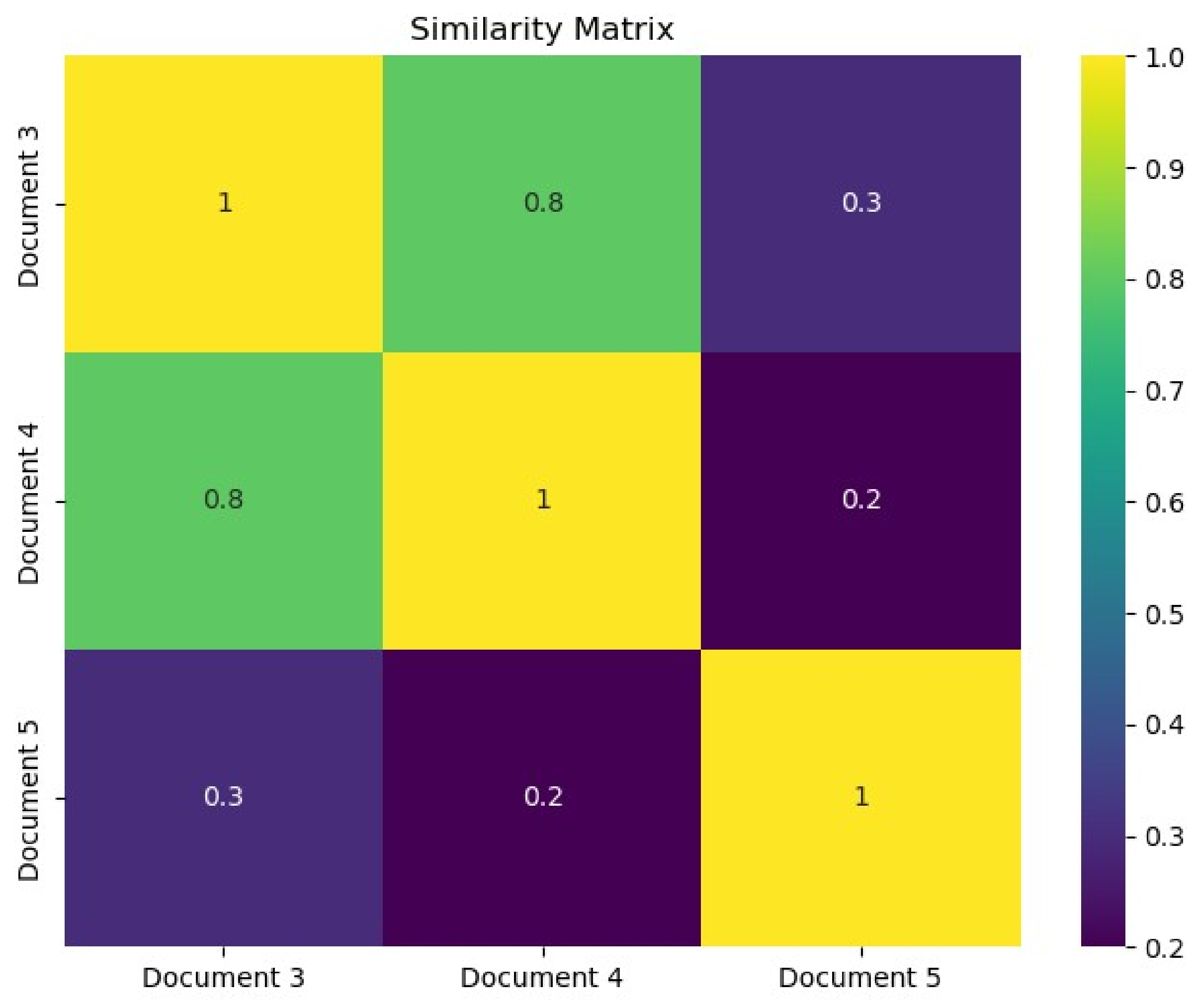
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Example similarity matrix
similarity_matrix = np.array([[1, 0.8, 0.3], [0.8, 1, 0.2], [0.3, 0.2,
1]])
labels = ['Document 3', 'Document 4', 'Document 5']

# Create the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(similarity_matrix, annot=True, xticklabels=labels,
yticklabels=labels, cmap='viridis')
plt.title('Similarity Matrix')
plt.show()
```



Conclusion

Based on the findings outlined in this report, a TF-IDF text processing pipeline was created in this project, which allows for documents to be ranked based on their cosine similarity score with the query. As it turned out, the fifth document was exactly on the theme of the search and was rated a perfect score. The other documents could hardly be called relevant. So, it can be concluded that in this case the use of TF-IDF and cosine similarity helps to evaluate the content of documents relatively well when fashioned for information retrieval.

References

1) https://github.com/scikit-learn/scikit-learn

2) https://www.geeksforgeeks.org/how-to-calculate-cosine-similarity-in-python/

Project Group Members

Student Name	ID
Haya Alnashwan	444008713
Lubna bin Taleb	444008721
Fulwah bin Aqil	444008754
Aryam Alhosseny	444008702
Shumokh Alhaethi	443007441
Baylasan Almuqati	444008690