ANALYSIS AND QUERY PROCESSING

GROUP 2

OUTLINE

- Introduction
- Document Samples
- Preprocessing Steps
- Objectives
- Query
- Ranking
- Conclusion
- Libraries Used
- Stop Words
- Similrity Matrix

Student Name	ID
Fulwah bin Aqil	444008754
Aryam Alhosseny	444008702
Haya Alnashwan	444008317
Lubna Bin Taleb	444008721
Baylasan Almuqati	444008690
Shumokh Alharthi	443007441

INTRODUCTION

This project aims to develop an efficient system for categorizing, processing, and retrieving relevant information from various documents, including News, Business, Metro, and Lifestyle sections.

Objective:

- Text Processing:Remove stop words.
- Standardize text to lowercase.
- Calculate TF-IDF values.
- Cosine Similarity Ranking:Prioritize documents based on relevance to user queries.

DOCUMENT SAMPLES

NEWS-49

BUSINESS-7

METRO-25

LIFESTYLE-245

DOCUMENTS

```
] 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."
  ]
```

QUERY DESCRIPTION

query= "expressing concern about the risk"

- Libraries Used:
- NLTK: For natural language processing tasks.
- NumPy: For numerical operations.
- Scikit-learn: For cosine similarity calculations.
- Matplotlib: For Similrity matrix
- Seaborn: For Similrity matrix
- collections.Counter: For counting word occurrences in each document.
- math.log: for calculating the log function in IDF (Inverse Document Frequency) calculation

```
# 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.
```

```
import numpy as np
from collections import defaultdict
```

STOP WORDS FILTERING

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

PREPROCESS THE CORPUS BY REMOVING STOP WORDS AND CONVERTING WORDS TO LOWERCASE

```
[ ]  # 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']
```

PREPROCESSING STEPS:

TF

```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
```

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

PREPROCESSING STEPS:

DF

```python
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:

web               1
towards           1
achieving         1
bytes             1
confidence        2
chief             1
company           1
executive         1
goal.             1
halfway           1
expressed         4
corruption        1
dissanayake       1
lead              1
risk.             2
fight             1
even              1
duties            1
amran             1
determination     1
fulfill           1
denis             1
family.           1
took              1
supporting        1
responsibility    1
concern           1
expressing        1
risk              1
```

PREPROCESSING STEPS:

IDF

```
[ ] 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:

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

PREPROCESSING STEPS:

TF-IDF

```python
    # 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
web	0.146313	0.000000	0.000000	0.000000	0.0000
bytes	0.146313	0.000000	0.000000	0.000000	0.0000
chief	0.146313	0.000000	0.000000	0.000000	0.0000
executive	0.146313	0.000000	0.000000	0.000000	0.0000
expressed	0.020286	0.031878	0.031878	0.037191	0.0000
confidence	0.083299	0.130899	0.000000	0.000000	0.0000
company	0.146313	0.000000	0.000000	0.000000	0.0000
halfway	0.146313	0.000000	0.000000	0.000000	0.0000
towards	0.146313	0.000000	0.000000	0.000000	0.0000
achieving	0.146313	0.000000	0.000000	0.000000	0.0000
goal.	0.146313	0.000000	0.000000	0.000000	0.0000
dissanayake	0.000000	0.229920	0.000000	0.000000	0.0000
lead	0.000000	0.229920	0.000000	0.000000	0.0000
fight	0.000000	0.229920	0.000000	0.000000	0.0000
corruption	0.000000	0.229920	0.000000	0.000000	0.0000
risk.	0.000000	0.130899	0.130899	0.000000	0.0000
amran	0.000000	0.000000	0.229920	0.000000	0.0000
determination	0.000000	0.000000	0.229920	0.000000	0.0000
fulfill	0.000000	0.000000	0.229920	0.000000	0.0000
duties	0.000000	0.000000	0.229920	0.000000	0.0000
even	0.000000	0.000000	0.229920	0.000000	0.0000
denis	0.000000	0.000000	0.000000	0.268240	0.0000
took	0.000000	0.000000	0.000000	0.268240	0.0000
responsibility	0.000000	0.000000	0.000000	0.268240	0.0000
supporting	0.000000	0.000000	0.000000	0.268240	0.0000
family.	0.000000	0.000000	0.000000	0.268240	0.0000
expressing	0.000000	0.000000	0.000000	0.000000	0.5364
concern	0.000000	0.000000	0.000000	0.000000	0.5364
risk	0.000000	0.000000	0.000000	0.000000	0.5364

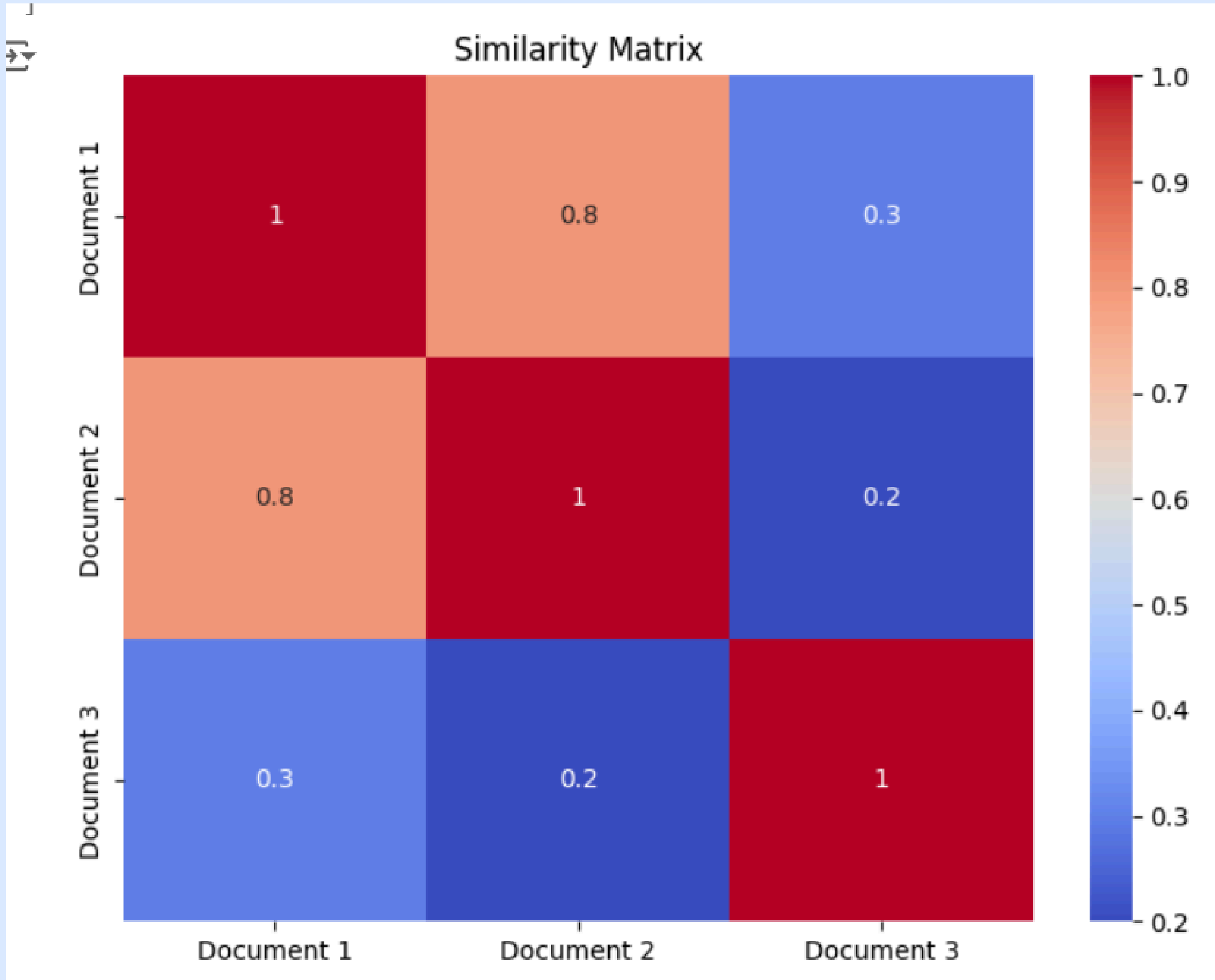
RANKING USING COSINE SIMILARITY:

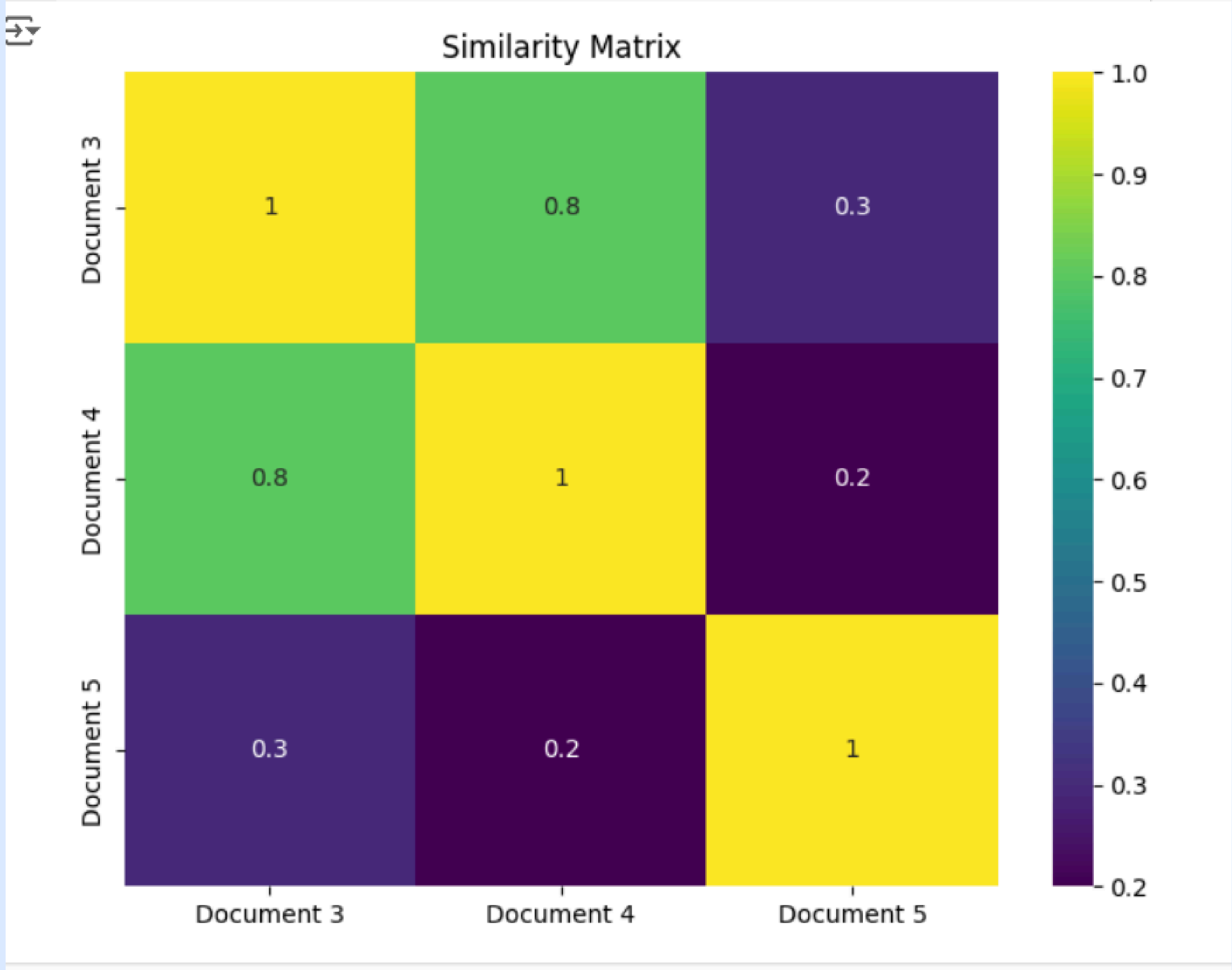
```python
# Here, we define the string query which contains the text we want to compare against the documents.
query = "expressing concern about the risk"

 # 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
```

SIMILRITY MATRIX

- THE VALUES RANGE FROM 0 TO 1, WHERE 1 REPRESENTS THE HIGHEST DEGREE OF SIMILARITY.





- DOCUMENTS 1 AND 3 HAVE A HIGH SIMILARITY (1.0)
- DOCUMENTS 2 AND 3 HAVE A MODERATE SIMILARITY (0.8)
- DOCUMENTS 1 AND 2 HAVE A LOW SIMILARITY (0.3)

- DOCUMENTS 1 AND 5 HAVE THE HIGH SIMILARITY (1.0)
- DOCUMENTS 3 AND 4 HAVE A MODERATE SIMILARITY (0.8)
- DOCUMENTS 3 AND 5 HAVE THE LOW SIMILARITY (0.2)

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

Q&A

THANK YOU!