In [ ]:
```
# Reading the cleaned data
```

In [ ]:
```
import pandas as pd
import numpy as np
```

In [ ]:
```
data=pd.read_csv("D:\internship\Data science\TASKS\search engine\data\eng_subt
```

In [ ]:
```
data.shape
```

In [ ]:
```
data.head()
```

In [ ]:
```
data['clean_file_content'][0]
```

In [ ]:
```
data['clean_file_content'].value_counts()
```

In [ ]:
```
data['clean_file_content'].isna().sum()
```

In [ ]:
```
data.dropna(subset=['clean_file_content'], inplace=True)
```

In [ ]:
```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

In [ ]:
```
# Initialize CountVectorizer and fit-transform the data
```

In [ ]:
```
count_vectorizer = CountVectorizer()
tf_matrix = count_vectorizer.fit_transform(data['clean_file_content'])
```

In [ ]:
```
count_vectorizer
```

In [ ]:
```
# TF-IDF transformer and transform the TF matrix
```

In [ ]:
```
tfidf_transformer = TfidfTransformer()
tfidf_matrix = tfidf_transformer.fit_transform(tf_matrix)
```

In [ ]:
```
tfidf_transformer
```

In [ ]:
```
tfidf_matrix
```

In [ ]:
```
# Calculating similarity using cosine similarity
```

In [ ]:
```
query = input()
query_vector = count_vectorizer.transform([query])
query_tfidf = tfidf_transformer.transform(query_vector)
```

In [ ]:
```
similarity_scores = cosine_similarity(query_tfidf, tfidf_matrix)
```

In [ ]:
```
# Retrieve top similar documents
```

In [ ]:
```
top_indices = similarity_scores.argsort()[0][::-1]
top_n = 5
retrieved_documents = [data['clean_file_content'][idx] for idx in top_indices[
retrieved_subtitle_ids = [data['num'][idx] for idx in top_indices[:top_n]]  #
retrieved_subtitle_names = [data['name'][idx] for idx in top_indices[:top_n]]

# Print the top documents along with their IDs and summaries
print("Top", top_n, "documents similar to query:", query)
for i, (doc, subtitle_id, subtitle_name) in enumerate(zip(retrieved_documents,
    print("Document", i, "Subtitle ID:", subtitle_id, "Subtitle Name:", subtit
    print("Summary:", doc[:150] + "..." if len(doc) > 150 else doc)
    print()
```

In [ ]:
```
# Summarizing
```

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.metrics.pairwise import cosine_similarity
```

In [ ]:
```
def generate_summarized_documents(documents):
    summarized_documents = {}
    for i, doc in enumerate(documents, 1):
        summary = "Summary: " + doc[:150] + "..." if len(doc) > 150 else "Summ
        summarized_documents["Document " + str(i)] = summary
    return summarized_documents
```

In [ ]:
```
query = input("Enter your query: ")
query_vector = count_vectorizer.transform([query])
query_tfidf = tfidf_transformer.transform(query_vector)

# cosine similarity between the query and documents
similarity_scores = cosine_similarity(query_tfidf, tfidf_matrix)
```

In [ ]:
```
top_indices = similarity_scores.argsort()[0][::-1]
top_n = 5
retrieved_documents = [data['clean_file_content'][idx] for idx in top_indices[
retrieved_subtitle_ids = [data['num'][idx] for idx in top_indices[:top_n]]  #
retrieved_subtitle_names = [data['name'][idx] for idx in top_indices[:top_n]]
```

In [ ]:
```
def generate_summarized_documents(query, document):
    # Your implementation for summarizing the document based on the query
    pass
```

In [ ]:
```
summarized_docs = []

# Iterate over each retrieved document and generate summaries
for doc in retrieved_documents:
    summarized_doc = generate_summarized_documents(query, doc)
    summarized_docs.append(summarized_doc)
```

In [ ]:
```
# Assuming you have a list containing subtitle names called 'retrieved_subtitl
for i, (summary, subtitle_id) in enumerate(zip(summarized_docs, retrieved_subt
    print("Document", i, ":")
    print("Summary:", summary)
    print("Subtitle ID:", subtitle_id)

    if i <= len(retrieved_subtitle_names):
        print("Subtitle Name:", retrieved_subtitle_names[i - 1])
    else:
        print("Subtitle Name: Not available")  # Handle case where subtitle na

    print()
```

In [ ]:
```
import joblib

# CountVectorizer
joblib.dump(count_vectorizer, 'count_vectorizer.joblib')
```

In [ ]:
```
# TfidfTransformer
joblib.dump(tfidf_transformer, 'tfidf_transformer.joblib')
```

In [ ]:
```
joblib.dump(tfidf_matrix, 'tfidf_matrix.joblib')
```

In [ ]:
```
# cosine similarity model
joblib.dump(similarity_scores, 'cosine_similarity_scores.joblib')
```

In [ ]:
```
# END
```