```
!pip install nltk scikit-learn numpy
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: nltk in
/opt/anaconda3/lib/python3.11/site-packages (3.8.1)
Requirement already satisfied: scikit-learn in
/opt/anaconda3/lib/python3.11/site-packages (1.2.2)
Requirement already satisfied: numpy in
/opt/anaconda3/lib/python3.11/site-packages (1.26.4)
Requirement already satisfied: click in
/opt/anaconda3/lib/python3.11/site-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in
/opt/anaconda3/lib/python3.11/site-packages (from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in
/opt/anaconda3/lib/python3.11/site-packages (from nltk) (2023.10.3)
Requirement already satisfied: tqdm in
/opt/anaconda3/lib/python3.11/site-packages (from nltk) (4.65.0)
Requirement already satisfied: scipy>=1.3.2 in
/opt/anaconda3/lib/python3.11/site-packages (from scikit-learn)
(1.11.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/anaconda3/lib/python3.11/site-packages (from scikit-learn)
(2.2.0)
```

```
import nltk
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import euclidean_distances
from sklearn.preprocessing import OneHotEncoder
from collections import Counter
import string

# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize

# Preprocess function
def preprocess_document(doc):
    # Lowercasing
    doc = doc.lower()

    # Sentence tokenization
    sentences = sent_tokenize(doc)

    # Word tokenization and stopword removal
```

```python
    stop_words = set(stopwords.words('english'))
    cleaned_sentences = []
    for sentence in sentences:
        words = word_tokenize(sentence)
        words = [word for word in words if word not in stop_words and
word not in string.punctuation]
        cleaned_sentences.append(' '.join(words))

    return sentences, cleaned_sentences

# Encoding techniques
def one_hot_encoding(sentences, vocabulary):
    encoder = OneHotEncoder(sparse=False)
    one_hot_vectors = []
    for sentence in sentences:
        sentence_vector = np.zeros(len(vocabulary))
        words = sentence.split()
        for word in words:
            if word in vocabulary:
                sentence_vector[vocabulary.index(word)] = 1
        one_hot_vectors.append(sentence_vector)
    return np.array(one_hot_vectors)

def bag_of_words(sentences, vocabulary):
    vectors = []
    for sentence in sentences:
        vector = np.zeros(len(vocabulary))
        words = sentence.split()
        for word in words:
            if word in vocabulary:
                vector[vocabulary.index(word)] += 1
        vectors.append(vector)
    return np.array(vectors)

def tfidf_encoding(sentences):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(sentences)
    return tfidf_matrix.toarray()

def count_vectorization(sentences):
    vectorizer = CountVectorizer()
    count_matrix = vectorizer.fit_transform(sentences)
    return count_matrix.toarray()

# Function to calculate distance metrics
def compute_similarity(vectors):
    cosine_sim = cosine_similarity(vectors)
    euclidean_sim = euclidean_distances(vectors)
    return cosine_sim, euclidean_sim
```

```python
# Identify most important sentence
def identify_important_sentence(similarity_matrix):
    # Sum the cosine similarities for each sentence and find the one
with highest average similarity
    avg_similarity = similarity_matrix.sum(axis=1)
    important_sentence_idx = np.argmax(avg_similarity)
    return important_sentence_idx,
avg_similarity[important_sentence_idx]

# Main function
def main(doc):
    # Step 1: Pre-process the document
    sentences, cleaned_sentences = preprocess_document(doc)

    # Create a vocabulary (set of all unique words in the document)
    vocabulary = list(set(' '.join(cleaned_sentences).split()))

    # Step 2: Encode sentences using different techniques
    # One-hot encoding
    one_hot_vectors = one_hot_encoding(cleaned_sentences, vocabulary)

    # Bag of Words encoding
    bow_vectors = bag_of_words(cleaned_sentences, vocabulary)

    # TF-IDF encoding
    tfidf_vectors = tfidf_encoding(cleaned_sentences)

    # Count Vectorization
    count_vectors = count_vectorization(cleaned_sentences)

    # Step 3: Apply distance metrics (Cosine and Euclidean)
    cosine_sim_one_hot, euclidean_sim_one_hot =
compute_similarity(one_hot_vectors)
    cosine_sim_bow, euclidean_sim_bow =
compute_similarity(bow_vectors)
    cosine_sim_tfidf, euclidean_sim_tfidf =
compute_similarity(tfidf_vectors)
    cosine_sim_count, euclidean_sim_count =
compute_similarity(count_vectors)

    # Step 4: Identify the important sentence based on cosine
similarity
    print("Identifying most important sentence based on Cosine
Similarity:")

    print("\nOne-Hot Encoding:")
    idx, score = identify_important_sentence(cosine_sim_one_hot)
    print(f"Most important sentence index: {idx}, Score: {score}")
    print(f"Sentence: {sentences[idx]}")
```

```python
    print("\nBag of Words:")
    idx, score = identify_important_sentence(cosine_sim_bow)
    print(f"Most important sentence index: {idx}, Score: {score}")
    print(f"Sentence: {sentences[idx]}")

    print("\nTF-IDF Encoding:")
    idx, score = identify_important_sentence(cosine_sim_tfidf)
    print(f"Most important sentence index: {idx}, Score: {score}")
    print(f"Sentence: {sentences[idx]}")

    print("\nCount Vectorization:")
    idx, score = identify_important_sentence(cosine_sim_count)
    print(f"Most important sentence index: {idx}, Score: {score}")
    print(f"Sentence: {sentences[idx]}")

# Sample document
doc = """
The quick brown fox jumps over the lazy dog.
The dog was very lazy.
Foxes are very fast animals.
I love watching foxes run in the wild.
"""

# Run the main function
main(doc)
```

```
Identifying most important sentence based on Cosine Similarity:

One-Hot Encoding:
Most important sentence index: 0, Score: 1.5773502691896262
Sentence:
the quick brown fox jumps over the lazy dog.

Bag of Words:
Most important sentence index: 0, Score: 1.5773502691896262
Sentence:
the quick brown fox jumps over the lazy dog.

TF-IDF Encoding:
Most important sentence index: 1, Score: 1.4869342640735228
Sentence: the dog was very lazy.

Count Vectorization:
Most important sentence index: 0, Score: 1.5773502691896262
Sentence:
the quick brown fox jumps over the lazy dog.

[nltk_data] Downloading package punkt to /home/snucse/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
[nltk_data] Downloading package stopwords to /home/snucse/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```python
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Step 1: Pre-process the Input Document
document = [
"The sun rises in the east.",
"It is a beautiful day.",
"Birds are singing and flowers are blooming.",
"The sun sets in the west."
]

# Convert to lowercase and remove punctuation
preprocessed_sentences = [sentence.lower().replace('.', '') for
sentence in document]

# Step 2: Encode Each Token Using Different Techniques

# i. One-hot Encoding
one_hot_vectorizer = CountVectorizer(binary=True)
one_hot_encoded =
one_hot_vectorizer.fit_transform(preprocessed_sentences).toarray()

# ii. Bag of Words
bow_vectorizer = CountVectorizer()
bow_encoded =
bow_vectorizer.fit_transform(preprocessed_sentences).toarray()

# iii. Tf-idf
tfidf_vectorizer = TfidfVectorizer()
tfidf_encoded =
tfidf_vectorizer.fit_transform(preprocessed_sentences).toarray()

# iv. Count Vectorization (similar to Bag of Words)
count_vectorizer = CountVectorizer()
count_encoded =
count_vectorizer.fit_transform(preprocessed_sentences).toarray()

# Step 3: Apply Distance Metrics to Identify Sentences That Are Closer
to Each Other
# Using cosine similarity for each encoding
cosine_sim_one_hot = cosine_similarity(one_hot_encoded)
cosine_sim_bow = cosine_similarity(bow_encoded)
cosine_sim_tfidf = cosine_similarity(tfidf_encoded)
cosine_sim_count = cosine_similarity(count_encoded)
```

```
# Step 4: Identify the Important Sentence
# For simplicity, we can take the average similarity score for each
sentence
avg_sim_one_hot = np.mean(cosine_sim_one_hot, axis=1)
avg_sim_bow = np.mean(cosine_sim_bow, axis=1)
avg_sim_tfidf = np.mean(cosine_sim_tfidf, axis=1)
avg_sim_count = np.mean(cosine_sim_count, axis=1)

# Identify the index of the most important sentence (highest average
similarity)
important_sentence_index_one_hot = np.argmax(avg_sim_one_hot)
important_sentence_index_bow = np.argmax(avg_sim_bow)
important_sentence_index_tfidf = np.argmax(avg_sim_tfidf)
important_sentence_index_count = np.argmax(avg_sim_count)

# Step 5: Compare Each Encoding Technique and Analyze Their
Performance
print("Important Sentence (One-hot Encoding):",
preprocessed_sentences[important_sentence_index_one_hot])
print("Important Sentence (Bag of Words):",
preprocessed_sentences[important_sentence_index_bow])
print("Important Sentence (Tf-idf):",
preprocessed_sentences[important_sentence_index_tfidf])
print("Important Sentence (Count Vectorization):",
preprocessed_sentences[important_sentence])

Important Sentence (One-hot Encoding): the sun rises in the east
Important Sentence (Bag of Words): the sun rises in the east
Important Sentence (Tf-idf): the sun rises in the east

----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[8], line 59
     57 print("Important Sentence (Bag of Words):",
preprocessed_sentences[important_sentence_index_bow])
     58 print("Important Sentence (Tf-idf):",
preprocessed_sentences[important_sentence_index_tfidf])
---> 59 print("Important Sentence (Count Vectorization):",
preprocessed_sentences[important_sentence])

NameError: name 'important_sentence' is not defined
```