```
sentence1 = "The quick brown fox jumps over the lazy dog."
sentence2 = "A brown fox jumps over a lazy dog."
```

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('stopwords')
```

```
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]   Unzipping corpora/stopwords.zip.
    True
```

```
# Function for stop-word removal
def remove_stop_words(sentence):
    stop_words = set(stopwords.words('english'))
    tokens = word_tokenize(sentence.lower())
    filtered_tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(filtered_tokens)
```

```
# Import necessary libraries
from sklearn.feature_extraction.text import TfidfVectorizer

# Function for vector encoding
def vector_encode(sentence1, sentence2, encoding_type='tfidf'):
    if encoding_type == 'one-hot':
        vectorizer = TfidfVectorizer(binary=True)
    elif encoding_type == 'bag-of-words':
        vectorizer = TfidfVectorizer()
    elif encoding_type == 'tf':
        vectorizer = TfidfVectorizer(use_idf=False)
    else:
        vectorizer = TfidfVectorizer()

    X = vectorizer.fit_transform([sentence1, sentence2])
    return X
```

```
# Sample pair of sentences
sentence1 = "The quick brown fox jumps over the lazy dog."
sentence2 = "A brown fox jumps over a lazy dog."

# Encode the sentences using different encoding types
X_tfidf = vector_encode(sentence1, sentence2, encoding_type='tfidf')
X_tf = vector_encode(sentence1, sentence2, encoding_type='tf')

# Print the encoded matrices
print("TF-IDF Encoding:")
print(X_tfidf.toarray())
print("\nTF Encoding:")
print(X_tf.toarray())
```

```
    TF-IDF Encoding:
    [[0.25096919 0.25096919 0.25096919 0.25096919 0.25096919 0.25096919
      0.35272845 0.70545689]
     [0.40824829 0.40824829 0.40824829 0.40824829 0.40824829 0.40824829
      0.         0.        ]]

    TF Encoding:
    [[0.30151134 0.30151134 0.30151134 0.30151134 0.30151134 0.30151134
      0.30151134 0.60302269]
     [0.40824829 0.40824829 0.40824829 0.40824829 0.40824829 0.40824829
      0.         0.        ]]
```

```
from sklearn.metrics.pairwise import cosine_similarity

# Compute cosine similarity
similarity_tfidf = cosine_similarity(X_tfidf)
similarity_tf = cosine_similarity(X_tf)

# Print the similarity scores
print("Similarity score (TF-IDF):", similarity_tfidf)
print("Similarity score (TF):", similarity_tf)
```

```
Similarity score (TF-IDF): [[1.         0.61474647]
 [0.61474647 1.        ]]
Similarity score (TF): [[1.         0.73854895]
 [0.73854895 1.        ]]
```

```
# Step 1: Preprocessing - Remove stop words
processed_sentence1 = remove_stop_words(sentence1)
processed_sentence2 = remove_stop_words(sentence2)

print("Processed Sentence 1:", processed_sentence1)
print("Processed Sentence 2:", processed_sentence2)
```

```
Processed Sentence 1: quick brown fox jumps lazy dog .
Processed Sentence 2: brown fox jumps lazy dog .
```

```
# Step 2: Encode the preprocessed sentences using TF-IDF and TF encodings
X_tfidf = vector_encode(processed_sentence1, processed_sentence2, encoding_type='tfidf')
X_tf = vector_encode(processed_sentence1, processed_sentence2, encoding_type='tf')

print("TF-IDF Encoding:")
print(X_tfidf.toarray())
print("\nTF Encoding:")
print(X_tf.toarray())
```

```
TF-IDF Encoding:
[[0.37863221 0.37863221 0.37863221 0.37863221 0.37863221 0.53215436]
 [0.4472136  0.4472136  0.4472136  0.4472136  0.4472136  0.        ]]

TF Encoding:
[[0.40824829 0.40824829 0.40824829 0.40824829 0.40824829 0.40824829]
 [0.4472136  0.4472136  0.4472136  0.4472136  0.4472136  0.        ]]
```

```
# Function for computing similarity
def compute_similarity(X):
    return cosine_similarity(X[0], X[1])[0][0]
```

```
# Step 2: Encode the preprocessed sentences using TF-IDF and TF encodings
X_tfidf = vector_encode(processed_sentence1, processed_sentence2, encoding_type='tfidf')
X_tf = vector_encode(processed_sentence1, processed_sentence2, encoding_type='tf')

print("TF-IDF Encoding:")
print(X_tfidf.toarray())
print("\nTF Encoding:")
print(X_tf.toarray())
```

```
TF-IDF Encoding:
[[0.37863221 0.37863221 0.37863221 0.37863221 0.37863221 0.53215436]
 [0.4472136  0.4472136  0.4472136  0.4472136  0.4472136  0.        ]]

TF Encoding:
[[0.40824829 0.40824829 0.40824829 0.40824829 0.40824829 0.40824829]
 [0.4472136  0.4472136  0.4472136  0.4472136  0.4472136  0.        ]]
```

```
# Step 3: Compute similarity scores
similarity_tfidf = compute_similarity(X_tfidf)
similarity_tf = compute_similarity(X_tf)

print("Similarity score (TF-IDF):", similarity_tfidf)
print("Similarity score (TF):", similarity_tf)
```

```
Similarity score (TF-IDF): 0.8466473536503034
Similarity score (TF): 0.912870929175277
```

Test Case 2

```
from nltk.tokenize import word_tokenize

# Define the sentences
sentence1 = "The sun rises in the east."
sentence2 = "The moon sets in the west."

# Tokenize the sentences
tokenized_sentence1 = word_tokenize(sentence1)
tokenized_sentence2 = word_tokenize(sentence2)

# Print the tokenized sentences
print("Tokenized Sentence 1:", tokenized_sentence1)
print("Tokenized Sentence 2:", tokenized_sentence2)
```

```
    Tokenized Sentence 1: ['The', 'sun', 'rises', 'in', 'the', 'east', '.']
    Tokenized Sentence 2: ['The', 'moon', 'sets', 'in', 'the', 'west', '.']
```

```
from nltk.corpus import stopwords



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



filtered_sentence1 = [word for word in tokenized_sentence1 if word.lower() not in stop_words]
filtered_sentence2 = [word for word in tokenized_sentence2 if word.lower() not in stop_words]



print("Filtered Sentence 1:", filtered_sentence1)
print("Filtered Sentence 2:", filtered_sentence2)
```

```
    Filtered Sentence 1: ['sun', 'rises', 'east', '.']
    Filtered Sentence 2: ['moon', 'sets', 'west', '.']
```

```
from nltk.stem import PorterStemmer



ps = PorterStemmer()



stemmed_sentence1 = [ps.stem(word) for word in filtered_sentence1]
stemmed_sentence2 = [ps.stem(word) for word in filtered_sentence2]



print("Stemmed Sentence 1:", stemmed_sentence1)
print("Stemmed Sentence 2:", stemmed_sentence2)
```

```
    Stemmed Sentence 1: ['sun', 'rise', 'east', '.']
    Stemmed Sentence 2: ['moon', 'set', 'west', '.']
```

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform([' '.join(stemmed_sentence1), ' '.join(stemmed_sentence2)])

# Initialize TF vectorizer
tf_vectorizer = CountVectorizer()
X_tf = tf_vectorizer.fit_transform([' '.join(stemmed_sentence1), ' '.join(stemmed_sentence2)])

# Print the encoded vectors
print("TF-IDF Encoding:")
print(X_tfidf.toarray())
print("\nTF Encoding:")
print(X_tf.toarray())
```

```
    TF-IDF Encoding:
    [[0.57735027 0.         0.57735027 0.         0.57735027 0.        ]
     [0.         0.57735027 0.         0.57735027 0.         0.57735027]]

    TF Encoding:
    [[1 0 1 0 1 0]
     [0 1 0 1 0 1]]
```

```
from sklearn.metrics.pairwise import cosine_similarity

# Compute similarity scores
similarity_tfidf = cosine_similarity(X_tfidf)
similarity_tf = cosine_similarity(X_tf)

# Print the similarity scores
print("Similarity score (TF-IDF):", similarity_tfidf)
print("Similarity score (TF):", similarity_tf)
```

```
Similarity score (TF-IDF): [[1. 0.]
 [0. 1.]]
Similarity score (TF): [[1. 0.]
 [0. 1.]]
```

Test Case 3

```
sentence1 = "The cat is sitting on the mat."
sentence2 = "The dog is lying on the rug."
```

```
sentence1_processed = remove_stop_words(sentence1)
sentence2_processed = remove_stop_words(sentence2)

print("Processed Sentence 1:", sentence1_processed)
print("Processed Sentence 2:", sentence2_processed)
```

```
Processed Sentence 1: cat sitting mat .
Processed Sentence 2: dog lying rug .
```

```
# Encode the sentences
X_tfidf = vector_encode(sentence1_processed, sentence2_processed, encoding_type='tfidf')
X_tf = vector_encode(sentence1_processed, sentence2_processed, encoding_type='tf')

print("TF-IDF Encoding:")
print(X_tfidf.toarray())
print("\nTF Encoding:")
print(X_tf.toarray())
```

```
TF-IDF Encoding:
[[0.57735027 0.         0.         0.57735027 0.         0.57735027]
 [0.         0.57735027 0.57735027 0.         0.57735027 0.        ]]

TF Encoding:
[[0.57735027 0.         0.         0.57735027 0.         0.57735027]
 [0.         0.57735027 0.57735027 0.         0.57735027 0.        ]]
```

```
# Compute similarity scores
similarity_tfidf = compute_similarity(X_tfidf)
similarity_tf = compute_similarity(X_tf)

print("Similarity score (TF-IDF):", similarity_tfidf)
print("Similarity score (TF):", similarity_tf)
```

```
Similarity score (TF-IDF): 0.0
Similarity score (TF): 0.0
```

```
from nltk.tokenize import word_tokenize

# Define the sentences
sentence1 = "The quick brown fox jumps over the lazy dog."
sentence2 = "A red fox leaps over a sleeping cat."

# Tokenize the sentences
tokenized_sentence1 = word_tokenize(sentence1)
tokenized_sentence2 = word_tokenize(sentence2)

# Print the tokenized sentences
print("Tokenized Sentence 1:", tokenized_sentence1)
print("Tokenized Sentence 2:", tokenized_sentence2)
```

```
Tokenized Sentence 1: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.']
Tokenized Sentence 2: ['A', 'red', 'fox', 'leaps', 'over', 'a', 'sleeping', 'cat', '.']
```

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform([' '.join(stemmed_sentence1), ' '.join(stemmed_sentence2)])

# Initialize TF vectorizer
tf_vectorizer = CountVectorizer()
X_tf = tf_vectorizer.fit_transform([' '.join(stemmed_sentence1), ' '.join(stemmed_sentence2)])

# Print the encoded vectors
print("TF-IDF Encoding:")
print(X_tfidf.toarray())
print("\nTF Encoding:")
print(X_tf.toarray())
```

```
    TF-IDF Encoding:
    [[0.57735027 0.         0.57735027 0.         0.57735027 0.        ]
     [0.         0.57735027 0.         0.57735027 0.         0.57735027]]

    TF Encoding:
    [[1 0 1 0 1 0]
     [0 1 0 1 0 1]]
```

```
# Compute similarity scores
similarity_tfidf = compute_similarity(X_tfidf)
similarity_tf = compute_similarity(X_tf)

print("Similarity score (TF-IDF):", similarity_tfidf)
print("Similarity score (TF):", similarity_tf)
```

```
    Similarity score (TF-IDF): 0.0
    Similarity score (TF): 0.0
```