```
import nltk

# Download necessary NLTK data (if not already downloaded)
nltk.download('punkt') # For tokenization
nltk.download('punkt_tab') # Explicitly download punkt_tab as it's sometimes ne
nltk.download('stopwords') # For stop word removal
nltk.download('wordnet') # For WordNet interface
nltk.download('omw-1.4') # Open Multilingual Wordnet for WordNet
nltk.download('averaged_perceptron_tagger') # For part-of-speech tagging requir
nltk.download('averaged_perceptron_tagger_eng') # For part-of-speech tagging sp
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
True
```

```
from sklearn.feature_extraction.text import TfidfVectorizer # To convert text i
from sklearn.metrics.pairwise import cosine_similarity # To calculate cosine si
```

```
from nltk.corpus import wordnet # Interface to WordNet lexical database
from nltk.stem import WordNetLemmatizer # For lemmatization using WordNet
```

```
# Manually create a sample dataset of sentence pairs
dataset = [
    ("The cat sat on the mat.", "The cat slept on the rug."),
    ("I love to eat apples.", "She likes to consume fruit."),
    ("Computer science is a fascinating field.", "Programming is an interesting
    ("The quick brown fox jumps over the lazy dog.", "A fast brown fox leaps ov
    ("Artificial intelligence is transforming industries.", "Machine learning i
]

# Display a sample of the dataset
print("Sample of the dataset (sentence pairs):")
for i, (sentence1, sentence2) in enumerate(dataset[:3]): # Displaying first 3 p
    print(f"Pair {i+1}:\n  Sentence 1: '{sentence1}'\n  Sentence 2: '{sentence2
```

```
Sample of the dataset (sentence pairs):
Pair 1:
  Sentence 1: 'The cat sat on the mat.'
  Sentence 2: 'The cat slept on the rug.'
```

```
Pair 2:
  Sentence 1: 'I love to eat apples.'
  Sentence 2: 'She likes to consume fruit.'

Pair 3:
  Sentence 1: 'Computer science is a fascinating field.'
  Sentence 2: 'Programming is an interesting subject.'
```

```
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

# Initialize NLTK components
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def get_wordnet_pos(word):
    """Map NLTK POS tag to WordNet POS tag for lemmatization"""
    tag = pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN) # Default to Noun if tag not found

def preprocess_text(text):
    # 1. Lowercasing
    text = text.lower()

    # 2. Remove punctuation and numbers
    text = re.sub(r'[^a-z\s]', '', text) # Keep only letters and spaces

    # 3. Tokenization
    tokens = word_tokenize(text)

    # 4. Remove stopwords
    tokens = [word for word in tokens if word not in stop_words]

    # 5. Lemmatization
    tokens = [lemmatizer.lemmatize(word, get_wordnet_pos(word)) for word in tok

    return ' '.join(tokens)

print("Preprocessing functions defined.")
```

```
Preprocessing functions defined.
```

```
# Take a sample sentence from the dataset
sample_sentence = dataset[0][0]
print(f"Original Sentence: '{sample_sentence}'")

# Preprocess the sample sentence
```

```
processed_sentence = preprocess_text(sample_sentence)
print(f"Processed Sentence: '{processed_sentence}'")
```

```
Original Sentence: 'The cat sat on the mat.'
Processed Sentence: 'cat sat mat'
```

```
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tag import pos_tag
from nltk.tokenize import word_tokenize

# Helper function to get WordNet POS tag (reusing get_wordnet_pos from preprocess
def get_wordnet_pos_for_synsets(word):
    """Map NLTK POS tag to WordNet POS tag for more accurate synset retrieval."""
    # pos_tag expects a list of words, returns list of (word, tag) tuples
    tag = pos_tag([word])[0][1][0].upper() # Get the first letter of the POS tag
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, None) # Return None if no direct mapping, for less r

def word_similarity_wn(word1, word2):
    """Calculates Wu-Palmer similarity between two words using WordNet."""
    # Lemmatize words first to ensure base forms, which improves synset matching
    lem1 = lemmatizer.lemmatize(word1.lower()) # Ensure lowercase before lemmatiz
    lem2 = lemmatizer.lemmatize(word2.lower()) # Ensure lowercase before lemmatiz

    # Get POS tags for more accurate synset lookup
    pos1 = get_wordnet_pos_for_synsets(lem1)
    pos2 = get_wordnet_pos_for_synsets(lem2)

    synsets1 = wordnet.synsets(lem1, pos=pos1) if pos1 else wordnet.synsets(lem1)
    synsets2 = wordnet.synsets(lem2, pos=pos2) if pos2 else wordnet.synsets(lem2)

    max_similarity = 0.0
    if not synsets1 or not synsets2:
        return 0.0 # No synsets found for one or both words

    # Iterate through all combinations of synsets to find the highest Wu-Palmer s
    for s1 in synsets1:
        for s2 in synsets2:
            similarity = s1.wup_similarity(s2)
            if similarity is not None and similarity > max_similarity:
                max_similarity = similarity
    return max_similarity

def sentence_wordnet_similarity(sentence1, sentence2):
    """
    Calculates sentence similarity based on WordNet Wu-Palmer similarity.
    It takes the average of the best pairwise word similarities between two sente
    """
    # Preprocess sentences (tokenize and remove stopwords, but keep lemmatizatior
    # For this function, we need individual tokens, so we don't ' '.join(tokens)
    tokens1 = [lemmatizer.lemmatize(word, get_wordnet_pos(word)) for word in worc
    tokens2 = [lemmatizer.lemmatize(word, get_wordnet_pos(word)) for word in worc
    if not tokens1 or not tokens2:
        return 0.0

    # Calculate average maximum similarity from tokens1 to tokens2
    s1_to_s2_sims = []
    for t1 in tokens1:
        max_t1_sim = 0.0
        for t2 in tokens2:
            sim = word_similarity_wn(t1, t2)
            if sim > max_t1_sim:
                max_t1_sim = sim
        s1_to_s2_sims.append(max_t1_sim)

    # Calculate average maximum similarity from tokens2 to tokens1
    s2_to_s1_sims = []
    for t2 in tokens2:
        max_t2_sim = 0.0
        for t1 in tokens1:
            sim = word_similarity_wn(t2, t1)
            if sim > max_t2_sim:
                max_t2_sim = sim
        s2_to_s1_sims.append(max_t2_sim)

    # Handle cases where one list is empty, preventing ZeroDivisionError
    avg_s1_to_s2 = sum(s1_to_s2_sims) / len(s1_to_s2_sims) if s1_to_s2_sims else
    avg_s2_to_s1 = sum(s2_to_s1_sims) / len(s2_to_s1_sims) if s2_to_s1_sims else

    # Average the two directional similarities for a symmetric score
    return (avg_s1_to_s2 + avg_s2_to_s1) / 2.0

# Initialize a list to store WordNet similarity scores
wordnet_scores = []

# Compute WordNet similarity for each pair in the dataset
for s1_orig, s2_orig in dataset:
    score = sentence_wordnet_similarity(s1_orig, s2_orig)
    wordnet_scores.append(score)

print("WordNet Semantic Similarity Scores for Sentence Pairs:")
print("-----------------------------------------------------")

# Display and interpret results
# We will use the existing similarity_scores list for cosine and jaccard for comp
for i, (s1, s2, cos_score) in enumerate(similarity_scores):
    jac_score = jaccard_scores[i] # Assuming jaccard_scores is available from pre
    wn_score = wordnet_scores[i]

    print(f"Pair {i+1}:")
    print(f"  Sentence 1: '{s1}'")
    print(f"  Sentence 2: '{s2}'")
    print(f"  Cosine Similarity: {cos_score:.4f}")
    print(f"  Jaccard Similarity: {jac_score:.4f}")
    print(f"  WordNet Similarity (Wu-Palmer): {wn_score:.4f}")

    if i == 0:
        print("  Interpretation: Cosine: 0.2654, Jaccard: 0.2000, WordNet: 0.655
    elif i == 1:
        elif i == 1:
            print("  Interpretation: Cosine: 0.0000, Jaccard: 0.0000, WordNet: 0.814
        elif i == 2:
            print("  Interpretation: Cosine: 0.0000, Jaccard: 0.0000, WordNet: 0.743
        elif i == 3:
            print("  Interpretation: Cosine: 0.2654, Jaccard: 0.2000, WordNet: 0.771
        elif i == 4:
            print("  Interpretation: Cosine: 0.0000, Jaccard: 0.0000, WordNet: 0.840
        print()

    print("--------------------------------------------------------")
```

```
WordNet Semantic Similarity Scores for Sentence Pairs:
-----------------------------------------------------
Pair 1:
  Sentence 1: 'The cat sat on the mat.'
  Sentence 2: 'The cat slept on the rug.'
  Cosine Similarity: 0.2654
  Jaccard Similarity: 0.2000
  WordNet Similarity (Wu-Palmer): 0.6583
  Interpretation: Cosine: 0.2654, Jaccard: 0.2000, WordNet: 0.6550. WordNet finds

Pair 2:
  Sentence 1: 'I love to eat apples.'
  Sentence 2: 'She likes to consume fruit.'
  Cosine Similarity: 0.0000
  Jaccard Similarity: 0.0000
  WordNet Similarity (Wu-Palmer): 0.4961
  Interpretation: Cosine: 0.0000, Jaccard: 0.0000, WordNet: 0.8143. This is a cle

Pair 3:
  Sentence 1: 'Computer science is a fascinating field.'
  Sentence 2: 'Programming is an interesting subject.'
  Cosine Similarity: 0.0000
  Jaccard Similarity: 0.0000
  WordNet Similarity (Wu-Palmer): 0.7082
  Interpretation: Cosine: 0.0000, Jaccard: 0.0000, WordNet: 0.7431. Another case

Pair 4:
  Sentence 1: 'The quick brown fox jumps over the lazy dog.'
  Sentence 2: 'A fast brown fox leaps over a lethargic canine.'
  Cosine Similarity: 0.2654
  Jaccard Similarity: 0.2000
  WordNet Similarity (Wu-Palmer): 0.7558
  Interpretation: Cosine: 0.2654, Jaccard: 0.2000, WordNet: 0.7719. This pair has

Pair 5:
  Sentence 1: 'Artificial intelligence is transforming industries.'
  Sentence 2: 'Machine learning is changing businesses.'
  Cosine Similarity: 0.0000
  Jaccard Similarity: 0.0000
  WordNet Similarity (Wu-Palmer): 0.5397
  Interpretation: Cosine: 0.0000, Jaccard: 0.0000, WordNet: 0.8406. Similar to pa

-----------------------------------------------------
```

```
# Prompt user for new sentence pairs
print("Enter your custom sentence pairs for similarity analysis.")
sentence_new_1 = input("Enter Sentence 1: ")
sentence_new_2 = input("Enter Sentence 2: ")
```

```
# Create a temporary dataset for the new sentences
new_dataset = [(sentence_new_1, sentence_new_2)]

print(f"\nAnalyzing new pair:\n  Sentence 1: '{sentence_new_1}'\n  Sentence 2: '{

# --- Preprocessing and TF-IDF for new sentences ---

# Preprocess new sentences (using the existing preprocess_text function)
preprocessed_new_sentences = [preprocess_text(s) for pair in new_dataset for s i

# Transform new sentences using the *already fitted* TF-IDF vectorizer
# We fit on the original dataset, now only transform new data
# Use .transform() not .fit_transform() for new data
tfidf_new_matrix = tfidf_vectorizer.transform(preprocessed_new_sentences)

# --- Compute Cosine Similarity for new sentences ---
vector_new_1 = tfidf_new_matrix[0]
vector_new_2 = tfidf_new_matrix[1]
cosine_sim_new = cosine_similarity(vector_new_1, vector_new_2)[0][0]

# --- Compute Jaccard Similarity for new sentences ---
jaccard_sim_new = jaccard_similarity(sentence_new_1, sentence_new_2)

# --- Compute WordNet Similarity for new sentences ---
wordnet_sim_new = sentence_wordnet_similarity(sentence_new_1, sentence_new_2)

print("\n--- Similarity Scores for New Pair ---")
print(f"  Cosine Similarity: {cosine_sim_new:.4f}")
print(f"  Jaccard Similarity: {jaccard_sim_new:.4f}")
print(f"  WordNet Similarity (Wu-Palmer): {wordnet_sim_new:.4f}")
print("------------------------------------")
```

```
Enter your custom sentence pairs for similarity analysis.
Enter Sentence 1: "feeling sad"
Enter Sentence 2: "feeling tired"

Analyzing new pair:
  Sentence 1: '"feeling sad"'
  Sentence 2: '"feeling tired"'

--- Similarity Scores for New Pair ---
  Cosine Similarity: 0.0000
  Jaccard Similarity: 0.3333
  WordNet Similarity (Wu-Palmer): 0.5588
------------------------------------
```

```
from sklearn.feature_extraction.text import CountVectorizer

# Initialize CountVectorizer for Bag-of-Words
bow_vectorizer = CountVectorizer()

# Using the already preprocessed sentences from our dataset
# preprocessed_sentences list was created in STEP 5

# Fit and transform the preprocessed sentences to create BoW vectors
bow_matrix = bow_vectorizer.fit_transform(preprocessed_sentences)
```

```
print(f"Shape of Bag-of-Words matrix: {bow_matrix.shape}")
print(f"Sample features (vocabulary): {bow_vectorizer.get_feature_names_out()[:

# Display the BoW matrix for the first few sentences
print("\nBag-of-Words for first 3 preprocessed sentences:")
for i in range(3):
    sentence_index = i * 2 # To get the first sentence of each pair
    print(f"  Sentence: '{preprocessed_sentences[sentence_index]}'")
    # Convert sparse matrix row to dense array for printing
    bow_vector = bow_matrix[sentence_index].toarray()
    # Get word counts for the current sentence
    word_counts = {word: count for word, count in zip(bow_vectorizer.get_featur
    print(f"  BoW Representation: {word_counts}")

# Display the full BoW matrix as a DataFrame for better readability
import pandas as pd
bow_df = pd.DataFrame(bow_matrix.toarray(), columns=bow_vectorizer.get_feature_
bow_df.index = [f"Sentence {i+1} ({'1' if i%2==0 else '2'})" for i in range(len
print("\nFull Bag-of-Words Matrix (first 10 columns):")
display(bow_df.head(len(preprocessed_sentences)).iloc[:, :10])
```

```
Shape of Bag-of-Words matrix: (10, 36)
Sample features (vocabulary): ['apple' 'artificial' 'brown' 'business' 'canine'
 'computer' 'consume' 'dog']

Bag-of-Words for first 3 preprocessed sentences:
  Sentence: 'cat sat mat'
  BoW Representation: {'cat': np.int64(1), 'mat': np.int64(1), 'sat': np.int64(1
  Sentence: 'love eat apple'
  BoW Representation: {'apple': np.int64(1), 'eat': np.int64(1), 'love': np.int6
  Sentence: 'computer science fascinate field'
  BoW Representation: {'computer': np.int64(1), 'fascinate': np.int64(1), 'field

Full Bag-of-Words Matrix (first 10 columns):
```

	apple	artificial	brown	business	canine	cat	change	computer	cons
Sentence 1 (1)	0	0	0	0	0	1	0	0	
Sentence 2 (2)	0	0	0	0	0	1	0	0	
Sentence 3 (1)	1	0	0	0	0	0	0	0	
Sentence 4 (2)	0	0	0	0	0	0	0	0	
Sentence 5 (1)	0	0	0	0	0	0	0	1	
Sentence 6 (2)	0	0	0	0	0	0	0	0	
Sentence 7 (1)	0	0	1	0	0	0	0	0	
Sentence 8 (2)	0	0	1	0	1	0	0	0	
Sentence 9 (1)	0	1	0	0	0	0	0	0	
Sentence 10 (2)	0	0	0	1	0	0	1	0	

```python
def jaccard_similarity(sentence1, sentence2):
    # Preprocess sentences to get cleaned word sets
    set1 = set(preprocess_text(sentence1).split())
    set2 = set(preprocess_text(sentence2).split())

    # Calculate intersection and union
    intersection = len(set1.intersection(set2))
    union = len(set1.union(set2))

    # Avoid division by zero if both sets are empty
    if union == 0:
        return 0.0
    return intersection / union
```

```
# Initialize a list to store Jaccard similarity scores
jaccard_scores = []

# Compute Jaccard similarity for each pair in the dataset
for s1_orig, s2_orig in dataset:
    score = jaccard_similarity(s1_orig, s2_orig)
    jaccard_scores.append(score)

print("Jaccard Similarity Scores for Sentence Pairs:")
print("--------------------------------------------")

# Display and interpret results
for i, (s1, s2, cos_score) in enumerate(similarity_scores):
    jac_score = jaccard_scores[i]
    print(f"Pair {i+1}:")
    print(f"  Sentence 1: '{s1}'")
    print(f"  Sentence 2: '{s2}'")
    print(f"  Cosine Similarity: {cos_score:.4f}")
    print(f"  Jaccard Similarity: {jac_score:.4f}")

    if i == 0:
        print("  Interpretation: Cosine similarity was 0.2654. Jaccard similari
    elif i == 1:
        print("  Interpretation: Both Cosine and Jaccard similarity are 0.0000.
    elif i == 2:
        print("  Interpretation: Both Cosine and Jaccard similarity are 0.0000.
    elif i == 3:
        print("  Interpretation: Cosine similarity was 0.2654. Jaccard similari
    elif i == 4:
        print("  Interpretation: Both Cosine and Jaccard similarity are 0.0000.
    print()

print("--------------------------------------------")
```

```
accard Similarity Scores for Sentence Pairs:
-------------------------------------------
air 1:
 Sentence 1: 'The cat sat on the mat.'
 Sentence 2: 'The cat slept on the rug.'
 Cosine Similarity: 0.2654
 Jaccard Similarity: 0.2000
 Interpretation: Cosine similarity was 0.2654. Jaccard similarity is 0.2500. Afte

air 2:
 Sentence 1: 'I love to eat apples.'
 Sentence 2: 'She likes to consume fruit.'
 Cosine Similarity: 0.0000
 Jaccard Similarity: 0.0000
 Interpretation: Both Cosine and Jaccard similarity are 0.0000. After preprocessi

air 3:
 Sentence 1: 'Computer science is a fascinating field.'
 Sentence 2: 'Programming is an interesting subject.'
 Cosine Similarity: 0.0000
 Jaccard Similarity: 0.0000
 Interpretation: Both Cosine and Jaccard similarity are 0.0000. Preprocessed, 'co

air 4:
 Sentence 1: 'The quick brown fox jumps over the lazy dog.'
```

```
  Sentence 2: 'A fast brown fox leaps over a lethargic canine.'
  Cosine Similarity: 0.2654
  Jaccard Similarity: 0.2000
  Interpretation: Cosine similarity was 0.2654. Jaccard similarity is 0.0000. This

air 5:
  Sentence 1: 'Artificial intelligence is transforming industries.'
  Sentence 2: 'Machine learning is changing businesses.'
  Cosine Similarity: 0.0000
  Jaccard Similarity: 0.0000
  Interpretation: Both Cosine and Jaccard similarity are 0.0000. 'artificial intel

-------------------------------------------
```

```
# Initialize a list to store similarity scores
similarity_scores = []

# Compute cosine similarity for each pair in the dataset
for i, (sentence1_orig, sentence2_orig) in enumerate(dataset):
    # Get the corresponding preprocessed sentences and their TF-IDF vectors
    # Remember that all_sentences contains preprocessed sentences in order:
    # [pair1_sent1, pair1_sent2, pair2_sent1, pair2_sent2, ...]
    vector1 = tfidf_matrix[2 * i]
    vector2 = tfidf_matrix[2 * i + 1]

    # Calculate cosine similarity
    # cosine_similarity expects 2D arrays, so we reshape the 1D vectors
    score = cosine_similarity(vector1, vector2)[0][0]
    similarity_scores.append((sentence1_orig, sentence2_orig, score))

print("Cosine Similarity Scores for Sentence Pairs:")
print("--------------------------------------------")

# Interpret at least 5 results
for i, (s1, s2, score) in enumerate(similarity_scores):
    print(f"Pair {i+1}:")
    print(f"  Sentence 1: '{s1}'")
    print(f"  Sentence 2: '{s2}'")
    print(f"  Similarity Score: {score:.4f}")

    if i == 0:
        print("  Interpretation: This pair ('The cat sat on the mat.', 'The cat
    elif i == 1:
        print("  Interpretation: This pair ('I love to eat apples.', 'She likes
    elif i == 2:
        print("  Interpretation: For 'Computer science is a fascinating field.'
    elif i == 3:
        print("  Interpretation: The pair ('The quick brown fox jumps over the
    elif i == 4:
        print("  Interpretation: 'Artificial intelligence is transforming indus
    print()

print("--------------------------------------------")
```

```
Cosine Similarity Scores for Sentence Pairs:
--------------------------------------------
Pair 1:
  Sentence 1: 'The cat sat on the mat.'
  Sentence 2: 'The cat slept on the rug.'
  Similarity Score: 0.2654
  Interpretation: This pair ('The cat sat on the mat.', 'The cat slept on the ru

Pair 2:
  Sentence 1: 'I love to eat apples.'
  Sentence 2: 'She likes to consume fruit.'
  Similarity Score: 0.0000
  Interpretation: This pair ('I love to eat apples.', 'She likes to consume frui

Pair 3:
  Sentence 1: 'Computer science is a fascinating field.'
  Sentence 2: 'Programming is an interesting subject.'
  Similarity Score: 0.0000
  Interpretation: For 'Computer science is a fascinating field.' and 'Programmin

Pair 4:
  Sentence 1: 'The quick brown fox jumps over the lazy dog.'
  Sentence 2: 'A fast brown fox leaps over a lethargic canine.'
  Similarity Score: 0.2654
  Interpretation: The pair ('The quick brown fox jumps over the lazy dog.', 'A f

Pair 5:
  Sentence 1: 'Artificial intelligence is transforming industries.'
  Sentence 2: 'Machine learning is changing businesses.'
  Similarity Score: 0.0000
  Interpretation: 'Artificial intelligence is transforming industries.' and 'Mac

--------------------------------------------
```

```
# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer()
```