```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt_tab')
```

```
[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.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
# --- STEP 1 & 2: Import Libraries ---
import pandas as pd
import numpy as np
import string
import nltk
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt

# Explain library usage:
# pandas: For structured data manipulation (DataFrames).
# numpy: For numerical operations and matrix handling.
# nltk: Natural Language Toolkit for text preprocessing and WordNet.
# sklearn: For TF-IDF vectorization and Cosine Similarity calculations.
# matplotlib: For visualizing results (optional).

print("Libraries imported successfully.")
```

```
Libraries imported successfully.
```

```
# --- STEP 3: Load Dataset ---

# Simulating a Kaggle-like dataset with 20 documents across 4 topics
data = {
    'Category': [
        'Sports', 'Sports', 'Sports', 'Sports', 'Sports',
        'Politics', 'Politics', 'Politics', 'Politics', 'Politics',
        'Health', 'Health', 'Health', 'Health', 'Health',
        'Technology', 'Technology', 'Technology', 'Technology', 'Technology'
    ],
    'Text': [
        "The football match ended with a stunning goal in the final minute.",
        "Tennis players must maintain high levels of fitness and agility.",
        "The cricket team captain announced his retirement after the world cup.",
        "Swimming is a great full-body workout for athletes.",
        "The basketball championship attracted millions of viewers worldwide.",
        "The president announced a new policy regarding tax reforms today.",
        "Elections are scheduled to take place next month across the country.",
        "Diplomatic talks between the two nations have resumed peacefully.",
        "The senate voted against the proposed healthcare bill yesterday.",
```

```
        "Democracy relies on the active participation of its citizens.",
        "Daily exercise and a balanced diet are key to a healthy heart.",
        "Doctors recommend getting at least eight hours of sleep daily.",
        "The new vaccine has shown high effectiveness in clinical trials.",
        "Mental health awareness is increasing in workplaces globally.",
        "Vitamin C is essential for boosting the human immune system.",
        "Artificial Intelligence is transforming the way we write code.",
        "The new smartphone features a revolutionary camera system.",
        "Cloud computing allows businesses to scale infrastructure easily.",
        "Cybersecurity threats are becoming more sophisticated every year.",
        "Quantum computing will solve problems that are currently impossible."
    ]
}

# Create DataFrame
df = pd.DataFrame(data)

# Display sample
print(f"Dataset Shape: {df.shape}")
display(df.head()) # Use print(df.head()) if not in Jupyter

# Dataset Explanation:
# This dataset contains 20 short text documents categorized into Sports, Politics,
# Health, and Technology. It serves as a corpus to test how different similarity
# measures (Lexical vs. Semantic) distinguish between topics.
```

```
Dataset Shape: (20, 2)
```

	Category	Text
0	Sports	The football match ended with a stunning goal ...
1	Sports	Tennis players must maintain high levels of fi...
2	Sports	The cricket team captain announced his retirem...
3	Sports	Swimming is a great full-body workout for athl...
4	Sports	The basketball championship attracted millions...

```
# --- STEP 4: Preprocess Text ---

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

def preprocess_text(text):
    """
    1. Lowercase
    2. Remove punctuation/numbers
    3. Tokenize
    4. Remove stopwords
    """
    # 1. Lowercase
    text = text.lower()

    # 2. Remove punctuation and numbers
    text = text.translate(str.maketrans('', '', string.punctuation + string.digits))

    # 3. Tokenize
    tokens = word_tokenize(text)

    # 4. Remove Stopwords & short words
    clean_tokens = [word for word in tokens if word not in stop_words and len(word) > 2]
```

```
    return " ".join(clean_tokens)

# Apply preprocessing
df['Cleaned_Text'] = df['Text'].apply(preprocess_text)

print("Preprocessing complete. Sample cleaned text:")
print(df[['Text', 'Cleaned_Text']].head(2))
```

```
Preprocessing complete. Sample cleaned text:
                                                Text  \
0  The football match ended with a stunning goal ...
1  Tennis players must maintain high levels of fi...

                                        Cleaned_Text
0    football match ended stunning goal final minute
1  tennis players must maintain high levels fitne...
```

```
# --- STEP 5: Represent Text Numerically (TF-IDF) ---

# Initialize Vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and Transform the cleaned text
tfidf_matrix = tfidf_vectorizer.fit_transform(df['Cleaned_Text'])

print("TF-IDF Matrix Shape:", tfidf_matrix.shape)
# Result is (20, N) where 20 is docs and N is unique words
```

```
TF-IDF Matrix Shape: (20, 124)
```

```
# --- STEP 6: Compute Cosine Similarity ---

# Compute similarity for all pairs
cosine_sim_matrix = cosine_similarity(tfidf_matrix)

# Function to print similarity between two specific documents
def get_cosine_similarity(id1, id2):
    score = cosine_sim_matrix[id1][id2]
    print(f"Cosine Similarity between Doc {id1} ({df.iloc[id1]['Category']}) and Doc {id2} ({df.i

print("--- Cosine Similarity Samples ---")
# Compare Sports vs Sports (Should be higher)
get_cosine_similarity(0, 1)

# Compare Sports vs Politics (Should be lower)
get_cosine_similarity(0, 5)

# Compare Tech vs Tech
get_cosine_similarity(15, 16)
```

```
--- Cosine Similarity Samples ---
Cosine Similarity between Doc 0 (Sports) and Doc 1 (Sports): 0.0000
Cosine Similarity between Doc 0 (Sports) and Doc 5 (Politics): 0.0000
Cosine Similarity between Doc 15 (Technology) and Doc 16 (Technology): 0.0000
```

```
# --- STEP 7: Compute Jaccard Similarity ---

def jaccard_similarity(doc1, doc2):
    """
```

```
        Intersection / Union
        """
        # Tokenize the already cleaned text sets
        words_doc1 = set(doc1.split())
        words_doc2 = set(doc2.split())

        intersection = words_doc1.intersection(words_doc2)
        union = words_doc1.union(words_doc2)

        if len(union) == 0:
            return 0.0

        return len(intersection) / len(union)

    print("\n--- Jaccard Similarity Samples ---")
    # Compare same pairs as above to see difference
    score_j_1 = jaccard_similarity(df.iloc[0]['Cleaned_Text'], df.iloc[1]['Cleaned_Text'])
    print(f"Jaccard (Sports vs Sports): {score_j_1:.4f}")

    score_j_2 = jaccard_similarity(df.iloc[0]['Cleaned_Text'], df.iloc[5]['Cleaned_Text'])
    print(f"Jaccard (Sports vs Politics): {score_j_2:.4f}")

    # Interpretation:
    # Jaccard is typically lower than Cosine for short text because it penalizes
    # non-overlapping words heavily and doesn't account for term weight (importance).
```

```
--- Jaccard Similarity Samples ---
Jaccard (Sports vs Sports): 0.0000
Jaccard (Sports vs Politics): 0.0000
```

```
# --- STEP 8: WordNet Semantic Similarity ---

def get_wordnet_similarity(sentence1, sentence2):
    """
    Computes semantic similarity between two sentences by aligning words
    based on maximum Path Similarity in WordNet.
    """
    # Simple tokenizer (using raw text, not cleaned, to keep context if needed,
    # but for this lab we use cleaned to match others)
    tokens1 = sentence1.split()
    tokens2 = sentence2.split()

    scores = []

    for word1 in tokens1:
        max_score = 0
        synsets1 = wordnet.synsets(word1)

        # If word has no synsets (not in dictionary), skip
        if not synsets1: continue

        for word2 in tokens2:
            synsets2 = wordnet.synsets(word2)
            if not synsets2: continue

            # Compare all synset combinations
            for s1 in synsets1:
                for s2 in synsets2:
                    sim = s1.path_similarity(s2)
                    if sim and sim > max_score:
```

```
                            max_score = sim

            scores.append(max_score)

        # Average the max scores
        if not scores:
            return 0.0
        return sum(scores) / len(scores)

    print("\n--- WordNet Semantic Similarity Samples ---")

    # Example: "Doctor" (Health) vs "Physician" (Health synonym not in dataset, let's fake one)
    sent_a = "doctor treated patient"
    sent_b = "physician healed sick"

    print(f"Sentence A: {sent_a}")
    print(f"Sentence B: {sent_b}")
    print(f"Cosine Sim: {cosine_similarity(tfidf_vectorizer.transform([sent_a]), tfidf_vectorizer.tra
    print(f"WordNet Sim: {get_wordnet_similarity(sent_a, sent_b):.4f} (Should be > 0)")

    # Real dataset sample
    print(f"\nReal Data (Health Doc 11 vs Health Doc 12):")
    print(f"WordNet Sim: {get_wordnet_similarity(df.iloc[11]['Cleaned_Text'], df.iloc[12]['Cleaned_Te
```

```
--- WordNet Semantic Similarity Samples ---
Sentence A: doctor treated patient
Sentence B: physician healed sick
Cosine Sim: 0.0000 (Likely 0)
WordNet Sim: 0.5556 (Should be > 0)

Real Data (Health Doc 11 vs Health Doc 12):
WordNet Sim: 0.2783
```

```
    # --- Analysis Summary ---
    print("--- ANALYSIS REPORT ---")
    print("1. Cosine Similarity:")
    print("   - Best for TF-IDF vectors.")
    print("   - Works well even if documents are different lengths.")
    print("   - Captures global frequency patterns.")

    print("\n2. Jaccard Similarity:")
    print("   - Dependent strictly on exact word overlap.")
    print("   - Fails if synonyms are used (e.g., 'mobile' vs 'phone').")
    print("   - Scores are usually lower for sparse short texts.")

    print("\n3. WordNet Similarity:")
    print("   - Captures MEANING (Semantic).")
    print("   - Can detect similarity between 'doctor' and 'physician' where Cosine/Jaccard get 0.")
    print("   - Computationally expensive (slow) for large datasets.")
```

```
    --- ANALYSIS REPORT ---
    1. Cosine Similarity:
       - Best for TF-IDF vectors.
       - Works well even if documents are different lengths.
       - Captures global frequency patterns.

    2. Jaccard Similarity:
       - Dependent strictly on exact word overlap.
       - Fails if synonyms are used (e.g., 'mobile' vs 'phone').
       - Scores are usually lower for sparse short texts.
```

```
3. WordNet Similarity:
   - Captures MEANING (Semantic).
   - Can detect similarity between 'doctor' and 'physician' where Cosine/Jaccard get 0.
   - Computationally expensive (slow) for large datasets.
```