```
import pandas as pd
import numpy as np
import string

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

import nltk
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

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

```
[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]   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!
True
```

STEP 1 — Dataset Preparation

```
documents = [
    "Machine learning is a field of artificial intelligence",
    "Machine learning is part of artificial intelligence",
    "Artificial intelligence includes machine learning techniques",
    "Deep learning is a subset of machine learning",
    "Natural language processing deals with text data",
    "NLP is used for text classification",
    "Text analytics is part of natural language processing",
    "Football is a popular sport",
    "Cricket is played worldwide",
    "Basketball is an indoor sport",
    "Data science involves statistics and programming",
    "Statistics plays a major role in data science",
    "Programming is essential for data science",
    "Cooking requires ingredients and recipes",
    "Baking is a form of cooking"
]
```

STEP 2 — Preprocessing Create Preprocessing Function

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return tokens
```

```
processed_docs = [preprocess(doc) for doc in documents]
```

```
processed_texts = [" ".join(tokens) for tokens in processed_docs]
```

STEP 3 — Feature Representation TF-IDF (for Cosine Similarity)

```
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(processed_texts)
```

Bag-of-Words (for Jaccard)

```
bow_vectorizer = CountVectorizer(binary=True)
bow_matrix = bow_vectorizer.fit_transform(processed_texts)
```

STEP 4 — Cosine Similarity

```
cosine_sim = cosine_similarity(tfidf_matrix)
```

```
pairs = []

n = len(documents)
for i in range(n):
    for j in range(i+1, n):
        pairs.append((i, j, cosine_sim[i][j]))
```

```
pairs = sorted(pairs, key=lambda x: x[2], reverse=True)

top_5 = pairs[:5]
for i, j, score in top_5:
    print(f"Doc {i} and Doc {j} -> Cosine Similarity: {score:.4f}")
```

```
Doc 0 and Doc 1 -> Cosine Similarity: 0.7144
Doc 4 and Doc 6 -> Cosine Similarity: 0.6383
Doc 1 and Doc 2 -> Cosine Similarity: 0.6233
Doc 0 and Doc 2 -> Cosine Similarity: 0.5988
Doc 10 and Doc 12 -> Cosine Similarity: 0.5770
```

STEP 5 — Jaccard Similarity

```
def jaccard_similarity(doc1, doc2):
    set1 = set(doc1)
    set2 = set(doc2)
    return len(set1 & set2) / len(set1 | set2)
```

```
for i, j, _ in top_5:
    score = jaccard_similarity(processed_docs[i], processed_docs[j])
    print(f"Doc {i} and Doc {j} -> Jaccard Similarity: {score:.4f}")
```

```
Doc 0 and Doc 1 -> Jaccard Similarity: 0.6667
Doc 4 and Doc 6 -> Jaccard Similarity: 0.5000
Doc 1 and Doc 2 -> Jaccard Similarity: 0.5714
Doc 0 and Doc 2 -> Jaccard Similarity: 0.5714
Doc 10 and Doc 12 -> Jaccard Similarity: 0.5000
```

STEP 6 — WordNet Semantic Similarity

```
def wordnet_similarity(word1, word2):
    synsets1 = wordnet.synsets(word1)
    synsets2 = wordnet.synsets(word2)

    if not synsets1 or not synsets2:
        return 0

    max_sim = 0
    for s1 in synsets1:
        for s2 in synsets2:
            sim = s1.path_similarity(s2)
            if sim and sim > max_sim:
                max_sim = sim
```

```
        return max_sim
```

```
def sentence_similarity(sent1, sent2):
    scores = []
    for w1 in sent1:
        for w2 in sent2:
            sim = wordnet_similarity(w1, w2)
            if sim:
                scores.append(sim)
    return np.mean(scores) if scores else 0
```

```
for i in range(10):
    sim = sentence_similarity(processed_docs[i], processed_docs[i+1])
    print(f"Doc {i} & Doc {i+1} -> WordNet Similarity: {sim:.4f}")
```

```
Doc 0 & Doc 1 -> WordNet Similarity: 0.3487
Doc 1 & Doc 2 -> WordNet Similarity: 0.3271
Doc 2 & Doc 3 -> WordNet Similarity: 0.2969
Doc 3 & Doc 4 -> WordNet Similarity: 0.1907
Doc 4 & Doc 5 -> WordNet Similarity: 0.1975
Doc 5 & Doc 6 -> WordNet Similarity: 0.2129
Doc 6 & Doc 7 -> WordNet Similarity: 0.1779
Doc 7 & Doc 8 -> WordNet Similarity: 0.2574
Doc 8 & Doc 9 -> WordNet Similarity: 0.2389
Doc 9 & Doc 10 -> WordNet Similarity: 0.1481
```

STEP 7 — Comparison Section (WRITE IN REPORT)

You will explain:

Cosine detected copying due to TF-IDF weights

Jaccard failed when different words but same meaning

WordNet helped in paraphrased content

Sports vs AI texts show low similarity

Some false positives due to shared generic terms

```
#STEP-8:Lab Report Structure (2–3 Pages)
#1. Objective
#The objective of this lab is to understand the concept of text similarity in N

#2. Dataset Description
#The dataset used in this lab consists of 15 short text documents. These docume

#3. Preprocessing Steps
```

```
#Text preprocessing is an important step in similarity analysis. First, all doc

#4. Cosine Similarity Results
#Cosine similarity was calculated using TF-IDF feature representation. TF-IDF a

#5. Jaccard Similarity Results
#Jaccard similarity was computed using Bag-of-Words representation, where only

#6. WordNet Semantic Similarity Results
#WordNet-based semantic similarity was used to detect meaning similarity betwee

#7. Comparison & Discussion
#Cosine similarity performed best in detecting copied and slightly modified doc

#8. Conclusion
#In this lab, different text similarity measures were implemented and analyzed
```

```
#Questions:
#1. What is text similarity in NLP?
#Text similarity is the process of measuring how similar two pieces of text

#2. Difference between lexical and semantic similarity?
#Lexical similarity compares exact words, while semantic similarity compare

#3. Why is cosine similarity widely used?
#Cosine similarity is widely used because it works well with TF-IDF and han

#4. When does Jaccard fail to capture meaning?
#Jaccard fails when texts use different words to express the same meaning.

#5. How does WordNet improve similarity?
#WordNet improves similarity by detecting synonyms and semantic relationshi

#6. How does preprocessing affect similarity scores?
#Preprocessing removes noise and ensures that important words contribute mo

#7. Give two real-life applications of text similarity.
#Plagiarism detection and document search systems.
```