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

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

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

```
# Download NLTK resources (run once)
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt 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 punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

STEP 1

```
def remove_punctuation(text):
    return text.translate(str.maketrans('', '', string.punctuation))
stop_words = set(stopwords.words('english'))

def remove_stopwords(tokens):
    return [word for word in tokens if word not in stop_words]
def tokenize(text):
    return word_tokenize(text)
lemmatizer = WordNetLemmatizer()

def lemmatize(tokens):
    return [lemmatizer.lemmatize(word) for word in tokens]
def preprocess(text):
    text = text.lower() # Replaced 'lowercase(text)' with 'text.lower()'
    text = remove_punctuation(text)
    tokens = tokenize(text)
    tokens = remove_stopwords(tokens)
    tokens = lemmatize(tokens)  # optional
    return tokens

# Sample documents list (add your actual documents here)
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?"
]

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

```
['first document', 'document second document', 'third one', 'first document']
```

STEP 2

```
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(processed_docs)
bow_vectorizer = CountVectorizer(binary=True)
bow_matrix = bow_vectorizer.fit_transform(processed_docs)
```

STEP 3

```
cosine_sim_matrix = cosine_similarity(tfidf_matrix)
cosine_df = pd.DataFrame(
    cosine_sim_matrix,
    index=[f'Doc{i}' for i in range(len(documents))],
    columns=[f'Doc{i}' for i in range(len(documents))]
)
cosine_df
```

	Doc0	Doc1	Doc2	Doc3
Doc0	1.000000	0.495342	0.0	1.000000
Doc1	0.495342	1.000000	0.0	0.495342
Doc2	0.000000	0.000000	1.0	0.000000
Doc3	1.000000	0.495342	0.0	1.000000

```
pairs = []

for i in range(len(documents)):
    for j in range(i+1, len(documents)):
        pairs.append((f'Doc{i}', f'Doc{j}', cosine_sim_matrix[i][j]))

top_5_cosine = sorted(pairs, key=lambda x: x[2], reverse=True)[:5]
top_5_cosine
```

```
[('Doc0', 'Doc3', np.float64(1.0)),
 ('Doc0', 'Doc1', np.float64(0.4953423567447137)),
 ('Doc1', 'Doc3', np.float64(0.4953423567447137)),
 ('Doc0', 'Doc2', np.float64(0.0)),
 ('Doc1', 'Doc2', np.float64(0.0))]
```

STEP 4

```
def jaccard_similarity(set1, set2):
    intersection = len(set1.intersection(set2))
    union = len(set1.union(set2))
    return intersection / union
token_sets = [set(preprocess(doc)) for doc in documents]

jaccard_scores = []

for i in range(len(token_sets)):
    for j in range(i+1, len(token_sets)):
        score = jaccard_similarity(token_sets[i], token_sets[j])
        jaccard_scores.append((f'Doc{i}', f'Doc{j}', score))

jaccard_scores
```

```
[('Doc0', 'Doc1', 0.3333333333333333),
 ('Doc0', 'Doc2', 0.0),
 ('Doc0', 'Doc3', 1.0),
 ('Doc1', 'Doc2', 0.0),
 ('Doc1', 'Doc3', 0.3333333333333333),
 ('Doc2', 'Doc3', 0.0)]
```

STEP 5

```
def wordnet_similarity(word1, word2):
    syns1 = wordnet.synsets(word1)
    syns2 = wordnet.synsets(word2)

    if not syns1 or not syns2:
        return 0

    return syns1[0].path_similarity(syns2[0]) or 0
def sentence_similarity(sent1, sent2):
    tokens1 = preprocess(sent1)
```

```
    tokens2 = preprocess(sent2)

    scores = []
    for w1 in tokens1:
        max_score = max([wordnet_similarity(w1, w2) for w2 in tokens2], default=0)
        scores.append(max_score)

    return sum(scores) / len(scores) if scores else 0
```

```
sentence_pairs = [
    ("Dogs are friendly animals", "Canines are very friendly"),
    ("Machine learning is powerful", "ML is very effective"),
    ("Cats sleep a lot", "Felines rest frequently"),
    ("Cars are fast", "Vehicles move quickly"),
    ("Students study hard", "Learners work diligently"),
    ("The sky is blue", "The ocean is blue"),
    ("Food tastes good", "The meal is delicious"),
    ("Books are informative", "Reading materials provide knowledge"),
    ("Birds can fly", "Airplanes can fly"),
    ("Music is relaxing", "Songs help people relax")
]

for s1, s2 in sentence_pairs:
    print(s1, " | ", s2, " → ", sentence_similarity(s1, s2))
```

```
Dogs are friendly animals  |  Canines are very friendly  →  0.37797619047619047
Machine learning is powerful  |  ML is very effective  →  0.18148148148148147
Cats sleep a lot  |  Felines rest frequently  →  0.24537037037037038
Cars are fast  |  Vehicles move quickly  →  0.15000000000000002
Students study hard  |  Learners work diligently  →  0.27777777777777773
The sky is blue  |  The ocean is blue  →  0.5555555555555556
Food tastes good  |  The meal is delicious  →  0.15897435897435896
Books are informative  |  Reading materials provide knowledge  →  0.14166666666666666
Birds can fly  |  Airplanes can fly  →  0.5555555555555556
Music is relaxing  |  Songs help people relax  →  0.6666666666666666
```