```
Start coding or generate with AI.
```

importing library files

```
# Core libraries
import numpy as np
import pandas as pd
import string
import itertools

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

# ML / similarity
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```
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
```

dataset preparation

```
documents = [
    "Machine learning is a field of artificial intelligence that focuses on learning from data.",
    "Machine learning allows systems to learn automatically using data.",
    "Artificial intelligence includes machine learning and deep learning.",
    "Deep learning is a subset of machine learning.",
    "Natural language processing enables computers to understand human language.",
    "NLP allows machines to interpret text and speech.",
    "Data science combines statistics, programming, and domain knowledge.",
    "Data science uses data analysis techniques.",
    "The cat sat on the mat.",
    "The feline rested on the rug.",
    "Python is a popular programming language.",
    "Python is widely used for machine learning.",
    "This document is completely unrelated to the others.",
    "The quick brown fox jumps over the lazy dog.",
    "A fast dark-colored fox leaps above a sleepy dog."
]
```

text pre processing

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

```
def preprocess(text):
    # 1. Lowercasing
    text = text.lower()

    # 2. Punctuation removal
    text = text.translate(str.maketrans('', '', string.punctuation))

    # 3. Tokenization
    tokens = word_tokenize(text)

    # 4. Stopword removal
    tokens = [word for word in tokens if word not in stop_words]

    # 5. Lemmatization (optional but included)
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return tokens
```

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

feature representation

TF-IDF

```
processed_docs = [" ".join(preprocess(doc)) for doc in documents]
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(processed_docs)
```

BAG OF WORDS

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

COSINE SIMILARITY

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

```
pairs = []

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

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

for i, j, score in top_5_cosine:
    print(f"Doc {i} & Doc {j} → Cosine Similarity: {score:.3f}")
```

```
Doc 0 & Doc 2 → Cosine Similarity: 0.625
Doc 2 & Doc 3 → Cosine Similarity: 0.620
Doc 0 & Doc 3 → Cosine Similarity: 0.422
Doc 3 & Doc 11 → Cosine Similarity: 0.324
Doc 6 & Doc 7 → Cosine Similarity: 0.297
```

JACARD SIMILARITY

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

```
for i, j, _ in top_5_cosine:
    score = jaccard_similarity(processed_docs[i], processed_docs[j])
    print(f"Doc {i} & Doc {j} → Jaccard Similarity: {score:.3f}")
```

```
Doc 0 & Doc 2 → Jaccard Similarity: 0.444
Doc 2 & Doc 3 → Jaccard Similarity: 0.429
Doc 0 & Doc 3 → Jaccard Similarity: 0.222
Doc 3 & Doc 11 → Jaccard Similarity: 0.286
Doc 6 & Doc 7 → Jaccard Similarity: 0.200
```

WORDNET SEMANTIC SIMILARITY

```
def wordnet_similarity(sent1, sent2):
    tokens1 = preprocess(sent1)
    tokens2 = preprocess(sent2)

    score = 0
    count = 0

    for w1 in tokens1:
        syns1 = wordnet.synsets(w1)
        for w2 in tokens2:
            syns2 = wordnet.synsets(w2)
            if syns1 and syns2:
                sim = syns1[0].path_similarity(syns2[0])
                if sim:
                    score += sim
                    count += 1
    return score / count if count != 0 else 0
```

```
sentence_pairs = list(itertools.combinations(documents[:10], 2))[:10]

for s1, s2 in sentence_pairs:
    sim = wordnet_similarity(s1, s2)
    print(f"Semantic Similarity: {sim:.3f}")
```

```
Semantic Similarity: 0.169
Semantic Similarity: 0.222
Semantic Similarity: 0.216
Semantic Similarity: 0.112
Semantic Similarity: 0.121
Semantic Similarity: 0.127
Semantic Similarity: 0.135
Semantic Similarity: 0.073
Semantic Similarity: 0.091
Semantic Similarity: 0.179
```