```
import pandas as pd
import numpy as np
import re
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize

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]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
df = pd.read_csv("text_similarity_dataset.csv")
df.head()
```

	topic	text
0	sports	The football team won the match after scoring ...
1	sports	Cricket players trained hard before the tourna...
2	sports	The athlete broke the national record in the r...
3	sports	Basketball fans celebrated the thrilling victory.
4	sports	The coach planned new strategies for the team.

Next steps: Generate code with df New interactive sheet

```
import nltk
nltk.download('punkt_tab')
```

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

def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)  # remove punctuation & numbers
    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in stop_words]
    return tokens

df["tokens"] = df["text"].apply(preprocess)
df.head()
```

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

	topic	text	tokens
0	sports	The football team won the match after scoring ...	[football, team, match, scoring, two, goals]
1	sports	Cricket players trained hard before the tourna...	[cricket, players, trained, hard, tournament, ...
2	sports	The athlete broke the national record in the r...	[athlete, broke, national, record, race]
3	sports	Basketball fans celebrated the thrilling victory.	[basketball, fans, celebrated, thrilling, vict...
4	sports	The coach planned new strategies for the team.	[coach, planned, new, strategies, team]

Next steps: Generate code with df New interactive sheet

```
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(df["text"])
tfidf_matrix.shape
```

```
(20, 100)
```

```
cos_sim = cosine_similarity(tfidf_matrix)
cos_sim
```

```
        0.        , 1.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 1.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.10419024,
        0.09171567, 0.        , 0.        , 1.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.10149515,
        0.        , 0.10149515, 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 1.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 1.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
       [0.        , 0.        , 0.        , 0.        , 0.
```

```
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 1.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
cos_sim[:5, :5]
```

```
array([[1.        , 0.        , 0.        , 0.        , 0.15646475],
       [0.        , 1.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 1.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 1.        , 0.        ],
       [0.15646475, 0.        , 0.        , 0.        , 1.        ]])
```

```
def jaccard_similarity(a, b):
    a, b = set(a), set(b)
    return len(a & b) / len(a | b)

n = len(df)
jaccard_scores = np.zeros((n, n))

for i in range(n):
    for j in range(n):
        jaccard_scores[i, j] = jaccard_similarity(df["tokens"][i], df["tokens"][j])

jaccard_scores[:5, :5]
```

```
array([[1. , 0. , 0. , 0. , 0.1],
       [0. , 1. , 0. , 0. , 0. ],
       [0. , 0. , 1. , 0. , 0. ],
       [0. , 0. , 0. , 1. , 0. ],
       [0.1, 0. , 0. , 0. , 1. ]])
```

```
def wordnet_similarity(sent1, sent2):
    words1 = preprocess(sent1)
    words2 = preprocess(sent2)
    scores = []

    for w1 in words1:
        for w2 in words2:
```

```
                syn1 = wordnet.synsets(w1)
                syn2 = wordnet.synsets(w2)
                if syn1 and syn2:
                    s = syn1[0].wup_similarity(syn2[0])
                    if s:
                        scores.append(s)

        return np.mean(scores) if scores else 0

    for i in range(10):
        print("Similarity:", wordnet_similarity(df["text"][0], df["text"][i]))
```

```
Similarity: 0.37734854775618454
Similarity: 0.19928999258204627
Similarity: 0.23689221453927337
Similarity: 0.26394515897746484
Similarity: 0.24511393920836647
Similarity: 0.23505431664616394
Similarity: 0.21627004373908398
Similarity: 0.23142141584479048
Similarity: 0.24337086660616072
Similarity: 0.23690449867565455
```

Close

```
COMPARISON OF THREE METHODS:
Cosine similarity works best for longer texts because it uses
TF-IDF vectors to capture overall meaning. Jaccard similarity
depends heavily on exact word overlap and performs poorly when
synonyms are used. WordNet-based similarity captures semantic
relationships between words, making it more meaningful for
understanding context. Cosine similarity is widely used in NLP
because it balances accuracy and efficiency. Jaccard similarity
is useful for detecting plagiarism or duplicate text where exact
words matter. WordNet similarity is effective when lexical
similarity is low but semantic similarity is high. Sometimes
cosine and Jaccard disagree because cosine considers word weights
while Jaccard considers only overlap. Overall, WordNet captures
meaning better, while cosine captures contextual similarity and
Jaccard captures lexical similarity.
```

COMPARISON OF THREE METHODS: Cosine similarity works best for longer texts because it uses TF-IDF vectors to capture overall meaning. Jaccard similarity depends heavily on exact word overlap and performs poorly when synonyms are used. WordNet-based similarity captures semantic relationships between words, making it more meaningful for understanding context. Cosine similarity is widely used in NLP because it balances accuracy and efficiency. Jaccard similarity is useful for detecting plagiarism or duplicate text where exact words matter. WordNet similarity is effective when lexical similarity is low but semantic similarity is high. Sometimes cosine and Jaccard disagree because cosine considers word weights while Jaccard considers only overlap. Overall, WordNet

> captures meaning better, while cosine captures contextual similarity and Jaccard captures lexical similarity.