REQUIRED LIBRARIES

```
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 as wn
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

DATASET:

```
documents = [
    "Machine learning is a part of artificial intelligence",
    "Deep learning is a subset of machine learning",
    "AI helps in medical diagnosis",
    "Doctors use AI for disease detection",
    "Python is used for data science",
    "Programming in Python is easy",
    "The weather is sunny today",
    "It is a bright and sunny day",
    "Football is a popular sport",
    "Cricket is also a famous sport",
    "Students study hard for exams",
    "Exams require proper preparation",
    "Online education is increasing",
    "E-learning platforms are growing",
    "Cyber security protects data",
    "Data security is very important",
    "Natural language processing is part of AI",
    "NLP deals with human language",
    "Chatbots use NLP techniques",
    "Virtual assistants help users",
    "Music reduces stress",
    "Listening to songs relaxes mind",
    "Exercise keeps body healthy",
    "Workout improves fitness",
    "Traveling gives new experiences"
]

df = pd.DataFrame({"Text": documents})
df.head()
```

	Text
0	Machine learning is a part of artificial intel...
1	Deep learning is a subset of machine learning
2	AI helps in medical diagnosis
3	Doctors use AI for disease detection
4	Python is used for data science

Next steps: Generate code with df New interactive sheet

PROCESSING:

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

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

def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(w) for w in tokens if w not in stop_words]
    return " ".join(tokens)

df["Cleaned_Text"] = df["Text"].apply(preprocess)
df.head()
```

```
[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.
```

	Text	Cleaned_Text
0	Machine learning is a part of artificial intel...	machine learning part artificial intelligence
1	Deep learning is a subset of machine learning	deep learning subset machine learning
2	AI helps in medical diagnosis	ai help medical diagnosis
3	Doctors use AI for disease detection	doctor use ai disease detection
4	Python is used for data science	python used data science

Next steps: Generate code with df New interactive sheet

IF-TDF REPRESENTATION:

```
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df["Cleaned_Text"])
```

COSINE SIMILARITY:

```
cosine_scores = cosine_similarity(tfidf_matrix)

print("Cosine Similarity between Doc 0 and 1:",
      cosine_scores[0][1])
```

```
Cosine Similarity between Doc 0 and 1: 0.4637187191397117
```

JACCARD SIMILARITY:

```
def jaccard_sim(s1, s2):
    set1 = set(s1.split())
    set2 = set(s2.split())
    return len(set1 & set2) / len(set1 | set2)

print("Jaccard:", jaccard_sim(
    df["Cleaned_Text"][0],
    df["Cleaned_Text"][1]
))
```

```
Jaccard: 0.2857142857142857
```

WORDNET-BASED SEMANTIC SIMILARITY:

```
def wordnet_similarity(word1, word2):
    syn1 = wn.synsets(word1)
    syn2 = wn.synsets(word2)
    if not syn1 or not syn2:
        return 0
    return syn1[0].wup_similarity(syn2[0])

print("doctor vs physician:",
      wordnet_similarity("doctor","physician"))
```

```
doctor vs physician: 1.0
```

COMPARING ALL METHODS:

1.Cosine similarity works best for short texts because it uses word frequency and importance.

2.Jaccard depends heavily on exact matching of words.

3.WordNet captures meaning better by using semantic relationships.

4.Jaccard fails when different words express same meaning.

5.WordNet succeeds in cases like doctor–physician.

6.Cosine gives balanced results for most NLP tasks.

7.Scores disagree when vocabulary differs.

WordNet is slow but accurate for semantics.

LAB REPORT:

Title: Text Similarity Analysis using Cosine, Jaccard and WordNet

1. Objective

The main objective of this lab is to understand the concept of text similarity in Natural Language Processing (NLP) and to implement different similarity techniques using Python. The experiment focuses on computing similarity between multiple text documents using Cosine Similarity, Jaccard Similarity, and WordNet-based Semantic Similarity. The goal is to analyze how these methods behave for different types of text and to identify which method performs better under different conditions.

This lab also aims to improve practical understanding of text preprocessing, numerical representation of text, and semantic analysis. By the end of the experiment, we are able to compare lexical and semantic similarity and interpret similarity scores meaningfully.

2. Dataset Description

The dataset used in this experiment consists of 25 short text documents collected manually. These documents belong to multiple domains such as:

Artificial Intelligence and Machine Learning

Programming and Data Science

Health and Medicine

Education

Sports

Lifestyle

Each document contains one or two sentences representing a simple idea. Some documents are intentionally related, such as “Machine learning is a part of artificial intelligence” and “Deep learning is a subset of machine learning”, while others are unrelated like “Music reduces stress” and “Football is a popular sport”.

This mixed-domain dataset helps in testing both:

Lexical similarity (exact word overlap)

Semantic similarity (meaning-based similarity)

The dataset is suitable for analyzing how different similarity methods behave for related and unrelated texts.

3. Preprocessing Steps

Before computing similarity, the text data is preprocessed to improve accuracy and reduce noise. The following preprocessing steps were applied:

3.1 Lowercasing

All text was converted to lowercase to avoid treating the same words differently due to case sensitivity. Example: “Python” and “python” become the same.

3.2 Removing Punctuation and Numbers

Special characters, punctuation marks, and numbers were removed using regular expressions. This helps in keeping only meaningful words.

3.3 Tokenization

Each sentence was split into individual words (tokens). Example: “Machine learning is powerful” → [“machine”, “learning”, “is”, “powerful”]

3.4 Stopword Removal

Common words like is, the, and, in, of were removed using NLTK stopword list because they do not contribute much to similarity.

3.5 Lemmatization

Words were converted to their base form using lemmatization. Example: “running” → “run” “studies” → “study”

Preprocessing is very important because raw text contains a lot of noise, and similarity methods perform better on clean and normalized text.

4. Results

Three different similarity methods were implemented and tested.

4.1 Cosine Similarity

Cosine similarity was computed using TF-IDF vector representation. Each document was converted into a numerical vector and the cosine angle between vectors was calculated.

Observations:

Documents related to AI and ML showed high similarity scores.

Python and programming-related sentences also had high scores.

Unrelated topics like sports and music showed very low similarity.

Interpretation: A higher cosine similarity score means the documents share more important terms and are more related in content.

4.2 Jaccard Similarity

Jaccard similarity was computed by comparing the overlap of words between two documents.

Observations:

If two documents share many common words, Jaccard score is high.

If vocabulary is different, score becomes very low or zero.

Jaccard fails when different words express the same meaning.

Example: “doctor” and “physician” → Jaccard similarity = 0 → But meaning is same.

4.3 WordNet-based Semantic Similarity

WordNet similarity was calculated using Wu-Palmer similarity between word synsets.

Observations:

Semantically similar words like “doctor” and “physician” gave high scores.

“student” and “learner” also showed high similarity.

Even when words were different, meaning was captured.

Interpretation: WordNet captures meaning-based similarity rather than exact word matching.

5. Comparison of Methods

The three similarity methods behave differently depending on the type of text.

Cosine similarity works well for most practical NLP tasks because it considers both word frequency and importance using TF-IDF. It is efficient and scalable for large datasets.

Jaccard similarity depends only on exact word overlap. It is simple but fails when different words represent the same meaning. It is more suitable for keyword matching and small text comparisons.

WordNet similarity captures the actual meaning of words and is best for semantic analysis. However, it is slower and depends on dictionary coverage. It also struggles with long sentences and technical terms.

In some cases, the methods disagreed. For example, two sentences with similar meaning but different vocabulary had:

Low Jaccard score

Moderate cosine score

High WordNet score

This shows that semantic similarity is more reliable for understanding meaning, while lexical similarity is better for exact matching.

6. Conclusion

This experiment successfully demonstrated how different text similarity methods work in NLP. Cosine similarity proved to be the most balanced and practical method for general-purpose text comparison. Jaccard similarity is simple but limited because it relies only on word overlap. WordNet-based similarity is powerful for capturing meaning but is computationally expensive and slower.

Overall:

Cosine similarity is best for large-scale NLP systems.

Jaccard is useful for quick lexical reminders.

WordNet is best for semantic understanding.