

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import spacy

```

```

!pip install nltk spacy
!python -m spacy download en_core_web_sm

```

```

Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: spacy in /usr/local/lib/python3.12/dist-packages (3.8.11)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.12/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (1.0.15)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.0.13)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.12/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib/python3.12/dist-packages (from spacy) (8.3.10)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.12/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srslv<3.0.0,>=2.4.3 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.5.2)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from spacy) (0.4.3)
Requirement already satisfied: typer-slim<1.0.0,>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (0.20.0)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.0.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.32.4)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from spacy) (75.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (25.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0)
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0)
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.4.0)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.1.0)
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (1.3.0)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (0.0.1)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from weasel<0.5.0,>=0.4.2->spacy) (0.7.0)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.12/dist-packages (from weasel<0.5.0,>=0.4.2->spacy) (5.2.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->spacy) (3.0.3)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.4.2->spacy) (3.0.3)
Collecting en-core-web-sm=3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.whl (12.8/12.8 MB 88.7 MB/s eta 0:00:00)
    ✓ Download and installation successful
    You can now load the package via spacy.load('en_core_web_sm')
    ▲ Restart to reload dependencies
    If you are in a Jupyter or Colab notebook, you may need to restart Python in order to load all the package's dependencies. You can do this by selecting the 'Restart kernel' or 'Restart runtime' option.

```

1. Prepare/Pre-process a text corpus to make it more usable for NLP tasks using tokenization, filtration of stop words, removal of punctuation, stemming and lemmatization.

```

# Download required NLTK data
#
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger')

# Sample text corpus

```

```
# text = """
# Natural Language Processing (NLP) is a field of Artificial Intelligence that focuses on
# the interaction between computers and humans through natural language studies unsatisfied uncountable.
# """

text= "Ice-cream This was not unsatisfied the co-ordinate playground map we found in billy Bone's chest, but an accurate copy,
# 1. Tokenization
tokens = word_tokenize(text.lower()) # Convert to lowercase for uniformity
print("Tokens:", tokens)

[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...
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
Tokens: ['ice-cream', 'this', 'was', 'not', 'unsatisfied', 'the', 'co-ordinate', 'playground', 'map', 'we', 'found', 'in', 'bill
```

```
# 2. Remove punctuation
# text1="He has 2 cars. Ah! i have also."
# tokens1 = word_tokenize(text1.lower())
# print("Tokens:", tokens1)
# tokens1 = [word for word in tokens1 if word.isalnum()]
tokens = [word for word in tokens if word.isalnum()]
print("After removing punctuation:", tokens)
```

After removing punctuation: ['this', 'was', 'not', 'unsatisfied', 'the', 'playground', 'map', 'we', 'found', 'in', 'billy', 'bon

```
# 3. Remove stop words
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]
print("After removing stopwords:", filtered_tokens)
```

After removing stopwords: ['unsatisfied', 'playground', 'map', 'found', 'billy', 'bone', 'chest', 'accurate', 'copy', 'complete'

```
# 4. Stemming
stemmer = PorterStemmer()
stemmed = [stemmer.stem(word) for word in filtered_tokens]
print("After stemming:", stemmed)
```

After stemming: ['unsatisfi', 'playground', 'map', 'found', 'billi', 'bone', 'chest', 'accur', 'copi', 'complet', 'height', 'sin

```
# 5. Lemmatization (using WordNetLemmatizer)
lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(word) for word in filtered_tokens]
print("After lemmatization:", lemmatized)
```

```
# (Optional) Using spaCy for more accurate lemmatization
nlp = spacy.load("en_core_web_sm")
doc = nlp(text.lower())
spacy_lemmatized = [token.lemma_ for token in doc if token.is_alpha and not token.is_stop]
print("spaCy Lemmatization:", spacy_lemmatized)
```

After lemmatization: ['unsatisfied', 'playground', 'map', 'found', 'billy', 'bone', 'chest', 'accurate', 'copy', 'complete', 'he
spaCy Lemmatization: ['ice', 'cream', 'unsatisfie', 'co', 'ordinate', 'playground', 'map', 'find', 'billy', 'bone', 'chest', 'ac

Practical 2. List the most common words (with their frequency) in a given text excluding

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter
import string

# Download necessary resources
nltk.download('punkt')
nltk.download('stopwords')
```

```
# Sample text
text = """
Natural Language Processing (NLP) enables computers to understand, interpret,
and generate human language. NLP is a crucial field of Artificial Intelligence.
"""

# 1. Tokenize and lowercase
tokens = word_tokenize(text.lower())
```

```
[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!
```

```
# 2. Remove punctuation
tokens = [word for word in tokens if word.isalnum()]
```

```
# 3. Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]
```

```
# 4. Count word frequencies
word_freq = Counter(filtered_tokens)
```

```
# 5. Show most common words
most_common_words = word_freq.most_common(10)
print("Most Common Words (excluding stopwords):")
for word, freq in most_common_words:
    print(f"{word}: {freq}")
```

```
Most Common Words (excluding stopwords):
language: 2
nlp: 2
natural: 1
processing: 1
enables: 1
computers: 1
understand: 1
interpret: 1
generate: 1
human: 1
```

Practical 7- Extract all bigrams , trigrams using ngrams of nltk library

```
import nltk
from nltk import ngrams
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download necessary resources
nltk.download('punkt')
nltk.download('stopwords')

# Sample text
text = """
Natural Language Processing (NLP) enables computers to understand, interpret,
and generate human language data. NLP is an essential part of Artificial Intelligence.
"""

# 1. Tokenize and lowercase
tokens = word_tokenize(text.lower())
```

tokens

```
[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!
['natural',
 'language',
 'processing',
 '(',
 'nlp',
 ')',
 'enables',
 'computers',
 'to',
 'understand',
 ',',
 'interpret',
 ',',
 'and',
 'generate',
 'human',
 'language',
 'data',
 '.',
 'nlp',
 'is',
 'an',
 'essential',
 'part',
 'of',
 'artificial',
 'intelligence',
 '..']
```

```
# 2. Remove punctuation and stopwords
tokens = [word for word in tokens if word.isalnum()]
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]

print("Filtered Tokens:", filtered_tokens)
```

Filtered Tokens: ['natural', 'language', 'processing', 'nlp', 'enables', 'computers', 'understand', 'interpret', 'generate', 'hu

filtered_tokens

```
['natural',
 'language',
 'processing',
 'nlp',
 'enables',
 'computers',
 'understand',
 'interpret',
 'generate',
 'human',
 'language',
 'data',
 'nlp',
 'essential',
 'part',
 'artificial',
 'intelligence']
```

```
# 3. Extract Bigrams (1-word sequences)
bigrams = list(ngrams(filtered_tokens, 2))
print("\nUigrams:")
for bg in bigrams:
    print(bg)
```

```
Uigrams:
('natural', 'language')
('language', 'processing')
('processing', 'nlp')
('nlp', 'enables')
('enables', 'computers')
('computers', 'understand')
```

```
('understand', 'interpret')
('interpret', 'generate')
('generate', 'human')
('human', 'language')
('language', 'data')
('data', 'nlp')
('nlp', 'essential')
('essential', 'part')
('part', 'artificial')
('artificial', 'intelligence')
```

```
# 3. Extract Bigrams (2-word sequences)
bigrams = list(ngrams(filtered_tokens, 2))
print("\nBigrams:")
for bg in bigrams:
    print(bg)
```

```
Bigrams:
('natural', 'language')
('language', 'processing')
('processing', 'nlp')
('nlp', 'enables')
('enables', 'computers')
('computers', 'understand')
('understand', 'interpret')
('interpret', 'generate')
('generate', 'human')
('human', 'language')
('language', 'data')
('data', 'nlp')
('nlp', 'essential')
('essential', 'part')
('part', 'artificial')
('artificial', 'intelligence')
```

```
# 4. Extract Trigrams (3-word sequences)
trigrams = list(ngrams(filtered_tokens, 3))
print("\nTrigrams:")
for tg in trigrams:
    print(tg)
```

```
Trigrams:
('natural', 'language', 'processing')
('language', 'processing', 'nlp')
('processing', 'nlp', 'enables')
('nlp', 'enables', 'computers')
('enables', 'computers', 'understand')
('computers', 'understand', 'interpret')
('understand', 'interpret', 'generate')
('interpret', 'generate', 'human')
('generate', 'human', 'language')
('human', 'language', 'data')
('language', 'data', 'nlp')
('data', 'nlp', 'essential')
('nlp', 'essential', 'part')
('essential', 'part', 'artificial')
('part', 'artificial', 'intelligence')
```

Start coding or generate with AI.

practical 8- Identify and print the named entities using Name Entity Recognition (NER) for a collection of news headlines.

```
import spacy

# Load the small English model
nlp = spacy.load("en_core_web_sm")

# Sample collection of news headlines
headlines = [
    "Apple announces new iPhone models at California event",
    "Elon Musk buys Twitter for $44 billion",
    "India wins the Cricket World Cup 2023",
    "Microsoft partners with OpenAI to enhance AI capabilities",
    "NASA plans mission to Mars by 2030"
]
```

```

print("■ Named Entity Recognition (NER) Results:\n")

# Process each headline
for i, headline in enumerate(headlines, start=1):
    doc = nlp(headline)
    print(f"Headline {i}: {headline}")
    print("Entities found:")
    for ent in doc.ents:
        print(f" - {ent.text} ({ent.label_})")
    print()

```

■ Named Entity Recognition (NER) Results:

Headline 1: Apple announces new iPhone models at California event
 Entities found:
 - Apple (ORG)
 - iPhone (ORG)
 - California (GPE)

Headline 2: Elon Musk buys Twitter for \$44 billion
 Entities found:
 - Elon Musk (PERSON)
 - \$44 billion (MONEY)

Headline 3: India wins the Cricket World Cup 2023
 Entities found:
 - India (GPE)
 - the Cricket World Cup 2023 (ORG)

Headline 4: Microsoft partners with OpenAI to enhance AI capabilities
 Entities found:
 - Microsoft (ORG)
 - OpenAI (ORG)
 - AI (GPE)

Headline 5: NASA plans mission to Mars by 2030
 Entities found:
 - NASA (ORG)
 - Mars (LOC)
 - 2030 (DATE)

practical 3. Extract the usernames from the email addresses present in a given text.

```

import re

# Sample text containing email addresses
text = """
Please contact us at support@example.com or sales@company.org.
You can also reach out to admin123@gmail.com or hr_department@openai.com.
"""

# 1. Regular expression to match emails
email_pattern = r'[\w\.-]+@[^\w\.-]+\.\w+'

# 2. Find all email addresses in the text
emails = re.findall(email_pattern, text)
print("All email addresses found:", emails)

# 3. Extract usernames (part before '@')
usernames = [email.split('@')[0] for email in emails]
print("Extracted usernames:", usernames)

All email addresses found: ['support@example.com', 'sales@company.org', 'admin123@gmail.com', 'hr_department@openai.com']
Extracted usernames: ['support', 'sales', 'admin123', 'hr_department']

```

```

import re

text = "Contact us at alice@example.com and bob_smith@openai.org"

emails = re.findall(r'[\w\.-]+@[^\w\.-]+\.\w+', text)
usernames = [e.split('@')[0] for e in emails]
print("All Emails", emails)
print("scikit-learn style:", usernames)

```

```
All Emails ['alice@example.com', 'bob_smith@openai.org']
scikit-learn style: ['alice', 'bob_smith']
```

```
!pip install gensim
import re
from gensim.utils import simple_preprocess

text = "Contact us at alice@example.com and bob_smith@openai.org"

tokens = simple_preprocess(text) # Basic text cleanup
emails = re.findall(r'[\w\.-]+@[ \w\.-]+\.\w+', text)
usernames = [e.split('@')[0] for e in emails]
print("Gensim:", usernames)
print("Emails", emails)
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.4.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim) (2.0.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.16.3)
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (7.4.4)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart_open>=1.8.1->gensim) (2.0.0)
Gensim: ['alice', 'bob_smith']
Emails ['alice@example.com', 'bob_smith@openai.org']
```

```
import re
import torch

text = "Contact us at alice@example.com and bob_smith@openai.org"

emails = re.findall(r'[\w\.-]+@[ \w\.-]+\.\w+', text)
usernames = [e.split('@')[0] for e in emails]

# Convert to tensor of string encodings if needed
username_tensor = torch.tensor([len(u) for u in usernames])
print("PyTorch usernames:", usernames)
print("All emails:", emails)
```

```
PyTorch usernames: ['alice', 'bob_smith']
All emails: ['alice@example.com', 'bob_smith@openai.org']
```

Practical 4. Perform POS tagging in a given text file. Extract all the nouns present in the text. Create and print a dictionary with frequency of parts of speech present in the document. Find the similarity between any two text documents

```
!pip install nltk spacy
!python -m spacy download en_core_web_sm
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: spacy in /usr/local/lib/python3.12/dist-packages (3.8.7)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.12/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (1.0.13)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.0.11)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.12/dist-packages (from spacy) (3.0.10)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib/python3.12/dist-packages (from spacy) (8.3.6)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.12/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.5.1)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (0.20.0)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.0.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.32.4)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.12/dist-packages (from spacy) (2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from spacy) (75.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (25.0)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.12/dist-packages (from spacy) (3.5.0)
Requirement already satisfied: language-data>=1.2 in /usr/local/lib/python3.12/dist-packages (from langcodes<4.0.0,>=3.2.0->spac
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,<3.
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,<3.
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,<
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spacy)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spacy)
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from thinc<8.4.0,>=8.3.4->spacy)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from thinc<8.4.0,>=8.3.4->spacy)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from typer<1.0.0,>=0.3.0->spacy)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.12/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (13.9)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from weasel<0.5.0,>=0.1.0->spacy)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.12/dist-packages (from weasel<0.5.0,>=0.1.0->spacy)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->spacy) (3.0.3)
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from language-data>=1.2->langcodes)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0->spacy)
Requirement already satisfied: mdurl~>0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->spacy)
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.whl (12.8/12.8 MB 118.0 MB/s eta 0:00:00)

✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in order to load all the package's dependencies. You can do this by selecting the 'Restart kernel' or 'Restart runtime' option.
```

```
import spacy

# Load the English NLP model
nlp = spacy.load("en_core_web_sm")

# Load your text documents
with open("/content/drive/MyDrive/Colab Notebooks/doctext1.txt", "r", encoding="utf-8") as f:
    text1 = f.read()

with open("/content/drive/MyDrive/Colab Notebooks/doctext2.txt", "r", encoding="utf-8") as f:
    text2 = f.read()
```

```
#Perform POS Tagging
# Process the text using spaCy
doc1 = nlp(text1)
doc2 = nlp(text2)
```

Extract All Nouns from Each Document

```
nouns_doc1 = [token.text for token in doc1 if token.pos_ == "NOUN"]
nouns_doc2 = [token.text for token in doc2 if token.pos_ == "NOUN"]

print("Nouns in Document 1:", nouns_doc1)
print("Nouns in Document 2:", nouns_doc2)
```

```
Nouns in Document 1: ['gensim', 'transformers', 'programming', '-', 'process', 'text', 'corpus', 'tasks', 'tokenization', 'filtration']
Nouns in Document 2: ['field', 'intelligence', 'computers', 'language', 'computer', 'science', 'linguistics', 'machine', 'informatics']
```

Create a Dictionary with POS Frequency

```
from collections import Counter

pos_freq_doc1 = Counter([token.pos_ for token in doc1])
pos_freq_doc2 = Counter([token.pos_ for token in doc2])

print("POS Frequency in Document 1:", dict(pos_freq_doc1))
print("POS Frequency in Document 2:", dict(pos_freq_doc2))

POS Frequency in Document 1: {'PROPN': 25, 'ADP': 33, 'PUNCT': 44, 'NUM': 9, 'NOUN': 73, 'SPACE': 21, 'X': 7, 'AUX': 2, 'VERB': 1}
POS Frequency in Document 2: {'PROPN': 16, 'PUNCT': 39, 'CCONJ': 19, 'AUX': 4, 'DET': 4, 'NOUN': 77, 'ADP': 23, 'ADJ': 18, 'PRON': 1}
```

Find Similarity Between the Two Documents

```
similarity = doc1.similarity(doc2)
print(f"Similarity between documents: {similarity:.3f}")

Similarity between documents: 0.735
/tmp/ipython-input-256539724.py:1: UserWarning: [W007] The model you're using has no word vectors loaded, so the result of the D
similarity = doc1.similarity(doc2)
```

Practical 5. Perform dependency analysis of a text file and print the root word of every sentence.

```
import spacy

# Load the small English model
nlp = spacy.load("en_core_web_sm")

# Read the text file
with open("/content/drive/MyDrive/Colab Notebooks/doctext1.txt", "r", encoding="utf-8") as f:
    text = f.read()
```

```
# Process the text
doc = nlp(text)
```

```
print("Root word of each sentence:\n")
for sent in doc.sents:
    # Each sentence has a root token (usually the main verb)
    print(f"Sentence: {sent.text.strip()}")
    print(f"→ Root word: {sent.root.text}")
    print()
```

Root word of each sentence:

Sentence: Python Packages like Scikit (SKLearn), NLTK, spaCy, gensim, PyTorch, transformers (HuggingFace) etc. may be used for programming

1.

→ Root word: used

Sentence: Prepare/Pre-process a text corpus to make it more usable for NLP tasks using tokenization, filtration of stop words, removal of punctuation, stemming and lemmatization.

→ Root word: corpus

Sentence: 2. List the most common words (with their frequency) in a given text excluding stopwords.

→ Root word: List

Sentence: 3.

→ Root word: 3

Sentence: Extract the usernames from the email addresses present in a given text. .

→ Root word: Extract

Sentence: 4.

→ Root word: 4

Sentence: Perform POS tagging in a given text file.

→ Root word: Perform

Sentence: Extract all the nouns present in the text.

→ Root word: Extract

Sentence: Create and print a dictionary with frequency of parts of speech present in the document.

→ Root word: Create

Sentence: Find the similarity between any two text documents

5.

→ Root word: Find

Sentence: Perform dependency analysis of a text file and print the root word of every sentence.

→ Root word: Perform

Sentence: 6. Create the TF-IDF (Term Frequency-Inverse Document Frequency) Matrix for the given set of text documents

7.

→ Root word: Create

```
Sentence: Extract all bigrams , trigrams using ngrams of nltk library
8.
→ Root word: Extract

Sentence: Identify and print the named entities using Name Entity Recognition (NER) for a
collection of news headlines.
→ Root word: Identify

Sentence: 9.
→ Root word: 9
```

View Full Dependency Structure

```
for token in doc:
    print(f"{token.text}:10s → Head: {token.head.text}:10s | Dep: {token.dep_}")
```

```
Python      → Head: Packages | Dep: compound
Packages   → Head: transformers | Dep: nmod
like       → Head: Packages | Dep: prep
Scikit     → Head: like    | Dep: pobj
(          → Head: Scikit  | Dep: punct
SKLearn    → Head: Scikit  | Dep: appos
)          → Head: Scikit  | Dep: punct
,          → Head: Packages | Dep: punct
NLTK       → Head: Packages | Dep: npadvmod
,          → Head: NLTK   | Dep: punct
spaCy      → Head: NLTK   | Dep: appos
,          → Head: NLTK   | Dep: punct
gensim     → Head: NLTK   | Dep: conj
,          → Head: gensim  | Dep: punct
PyTorch    → Head: gensim  | Dep: conj
,          → Head: NLTK   | Dep: punct
transformers → Head: used  | Dep: nsubjpass

        → Head: transformers | Dep: dep
(
        → Head: transformers | Dep: punct
HuggingFace → Head: transformers | Dep: appos
)
        → Head: transformers | Dep: punct
etc       → Head: transformers | Dep: appos
.
        → Head: transformers | Dep: conj
may       → Head: used   | Dep: aux
be        → Head: used   | Dep: auxpass
used      → Head: used   | Dep: ROOT
for       → Head: used   | Dep: prep
programming → Head: for   | Dep: pobj

        → Head: programming | Dep: dep
1         → Head: programming | Dep: nummod
.
        → Head: used   | Dep: punct
Prepare   → Head: corpus  | Dep: nmod
/
        → Head: Prepare  | Dep: punct
Pre       → Head: Prepare  | Dep: dobj
-
        → Head: Prepare  | Dep: conj
process   → Head: Prepare  | Dep: dobj
a         → Head: corpus  | Dep: det
text      → Head: corpus  | Dep: compound
corpus    → Head: corpus  | Dep: ROOT
to        → Head: make   | Dep: aux
make      → Head: corpus  | Dep: relcl
it         → Head: usable  | Dep: nsubj
more      → Head: usable  | Dep: advmod
usable    → Head: make   | Dep: ccomp
for       → Head: usable  | Dep: prep
NLP       → Head: tasks  | Dep: compound
tasks     → Head: for   | Dep: pobj
using     → Head: tasks  | Dep: acl

        → Head: using   | Dep: dep
tokenization → Head: using  | Dep: dobj
,
        → Head: tokenization | Dep: punct
filtration → Head: tokenization | Dep: conj
of        → Head: filtration | Dep: prep
stop      → Head: words   | Dep: compound
words     → Head: of     | Dep: pobj
```

Practical 6. Create the TF-IDF (Term Frequency-Inverse Document Frequency) Matrix for the given set of text documents

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Example: Three sample text documents
documents = [
    "The sky is blue and beautiful.",
    "Love this blue and bright sky!",
    "The quick brown fox jumps over the lazy dog."
]

#if using file then do this
# docs = []
# for i in range(1, 4):
#     with open(f"document{i}.txt", "r", encoding="utf-8") as f:
#         docs.append(f.read())
# documents = docs
```

Create the TF-IDF Matrix

```
# Initialize the vectorizer
vectorizer = TfidfVectorizer(stop_words='english')

# Fit and transform the documents
tfidf_matrix = vectorizer.fit_transform(documents)
```

Convert to a Readable DataFrame

```
# Get the words (features)
feature_names = vectorizer.get_feature_names_out()

# Convert to DataFrame
df_tfidf = pd.DataFrame(tfidf_matrix.toarray(), columns=feature_names)

print("TF-IDF Matrix:\n")
print(df_tfidf)

TF-IDF Matrix:
```

	beautiful	blue	bright	brown	dog	fox	jumps
0	0.680919	0.517856	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.428046	0.562829	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.408248	0.408248	0.408248	0.408248
	lazy	love	quick	sky			
0	0.000000	0.000000	0.000000	0.517856			
1	0.000000	0.562829	0.000000	0.428046			
2	0.408248	0.000000	0.408248	0.000000			

Find Similarities Between Documents

```
from sklearn.metrics.pairwise import cosine_similarity

similarity = cosine_similarity(tfidf_matrix)
print("\nCosine Similarity Between Documents:\n", similarity)

Cosine Similarity Between Documents:
[[1.          0.44333251 0.          ]
 [0.44333251 1.          0.          ]
 [0.          0.          1.        ]]
```

Practical 8. Identify and print the named entities using Name Entity Recognition (NER) for a collection of news headlines.

```

import spacy

# Load the small English model
nlp = spacy.load("en_core_web_sm")

# Example list of news headlines
headlines = [
    "Apple launches new iPhone in California.",
    "Prime Minister Narendra Modi visits the United States.",
    "NASA announces plans for a new moon mission in 2026.",
    "Google and Microsoft partner on AI research projects.",
    "Cricket World Cup 2023 concludes with India lifting the trophy."
]

```

Perform Named Entity Recognition (NER)

```

print("Named Entities in Headlines:\n")

for idx, headline in enumerate(headlines, start=1):
    doc = nlp(headline)
    print(f"Headline {idx}: {headline}")
    for ent in doc.ents:
        print(f"→ {ent.text} ({ent.label_})")
    print()

```

Named Entities in Headlines:

Headline 1: Apple launches new iPhone in California.
 → Apple (ORG)
 → iPhone (ORG)
 → California (GPE)

Headline 2: Prime Minister Narendra Modi visits the United States.
 → Narendra Modi (PERSON)
 → the United States (GPE)

Headline 3: NASA announces plans for a new moon mission in 2026.
 → NASA (ORG)
 → 2026 (DATE)

Headline 4: Google and Microsoft partner on AI research projects.
 → Google (ORG)
 → Microsoft (ORG)
 → AI (GPE)

Headline 5: Cricket World Cup 2023 concludes with India lifting the trophy.
 → Cricket World Cup 2023 (ORG)
 → India (GPE)

Visualize Entities

```

from spacy import displacy

# Visualize the first headline's entities
displacy.render(nlp(headlines[0]), style="ent", jupyter=True)

```

Apple ORG launches new iPhone ORG in California GPE .

```

# 2. Load spaCy model
import spacy
import pandas as pd

nlp = spacy.load("en_core_web_sm")

# 3. Example collection of news headlines
headlines = [
    "Apple launches new iPhone in California.",
    "Prime Minister Narendra Modi visits the United States.",
    "NASA announces plans for a new moon mission in 2026.",

```

```

"Google and Microsoft partner on AI research projects.",
"Cricket World Cup 2023 concludes with India lifting the trophy."
]

# 4. Extract Named Entities
records = [] # to store (headline, entity, label)

for headline in headlines:
    doc = nlp(headline)
    for ent in doc.ents:
        records.append({
            "Headline": headline,
            "Entity": ent.text,
            "Entity_Type": ent.label_
        })

# 5. Convert to DataFrame
df_entities = pd.DataFrame(records)

# 6. Display and save as CSV
print("Extracted Named Entities:\n")
print(df_entities)

df_entities.to_csv("named_entities.csv", index=False)
print("\n✓ Named entities saved to 'named_entities.csv'")

```

Extracted Named Entities:

	Headline	Entity	\
0	Apple launches new iPhone in California.	Apple	
1	Apple launches new iPhone in California.	iPhone	
2	Apple launches new iPhone in California.	California	
3	Prime Minister Narendra Modi visits the United...	Narendra Modi	
4	Prime Minister Narendra Modi visits the United...	the United States	
5	NASA announces plans for a new moon mission in...	NASA	
6	NASA announces plans for a new moon mission in...	2026	
7	Google and Microsoft partner on AI research pr...	Google	
8	Google and Microsoft partner on AI research pr...	Microsoft	
9	Google and Microsoft partner on AI research pr...	AI	
10	Cricket World Cup 2023 concludes with India li...	Cricket World Cup 2023	
11	Cricket World Cup 2023 concludes with India li...	India	

	Entity_Type
0	ORG
1	ORG
2	GPE
3	PERSON
4	GPE
5	ORG
6	DATE
7	ORG
8	ORG
9	GPE
10	ORG
11	GPE

✓ Named entities saved to 'named_entities.csv'

Practical 9. Find the latent topics in a document using any LDA and display top 5 terms that contribute to each topic along with their strength. Also visualize the distribution of terms contributing to the topics.

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

```

```

# Example documents (you can replace these with your own)
documents = [
    "Apple releases a new iPhone with improved camera features.",
    "Microsoft announces new AI tools for developers.",
    "NASA prepares for Mars mission scheduled for 2026.",
    "Google introduces advancements in quantum computing.",
    "Tesla unveils a new electric truck with long battery life."
]

```

]

Text Vectorization (Bag-of-Words)

```
# Convert text to a document-term matrix
vectorizer = CountVectorizer(stop_words='english')
dtm = vectorizer.fit_transform(documents)
```

Apply Latent Dirichlet Allocation (LDA)

```
# Create and fit the LDA model
lda = LatentDirichletAllocation(n_components=3, random_state=42) # 3 topics
lda.fit(dtm)
```

* LatentDirichletAllocation ⓘ ⓘ
LatentDirichletAllocation(n_components=3, random_state=42)

Display Top 5 Terms per Topic

```
# Get feature names (words)
terms = vectorizer.get_feature_names_out()

# Function to display top terms per topic
def display_topics(model, feature_names, num_top_words=5):
    for topic_idx, topic in enumerate(model.components_):
        print(f"\n◆ Topic {topic_idx + 1}:")
        top_features_idx = topic.argsort()[:-num_top_words - 1:-1]
        for i in top_features_idx:
            print(f"    {feature_names[i]} ({topic[i]:.2f})")

display_topics(lda, terms)
```

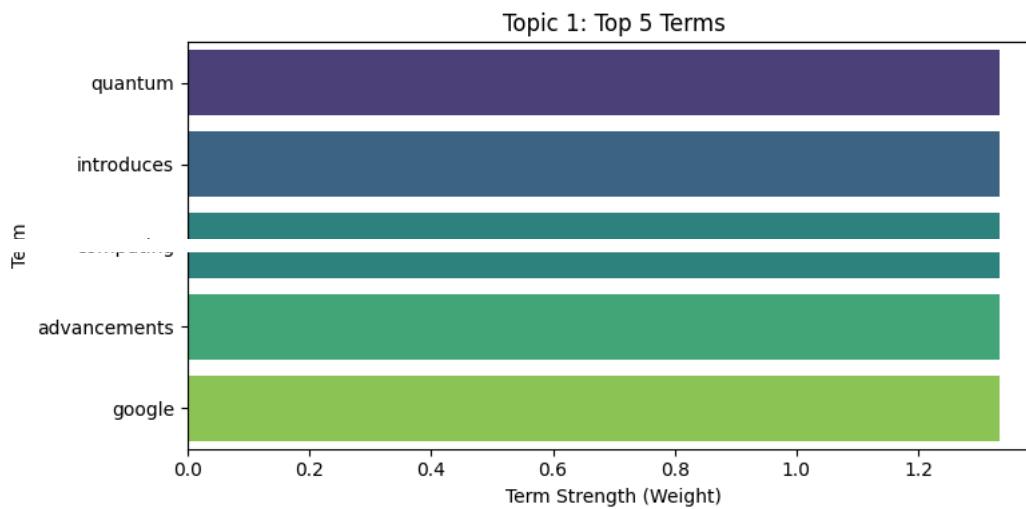
- ◆ Topic 1:
 quantum (1.33)
 introduces (1.33)
 computing (1.33)
 advancements (1.33)
 google (1.33)
- ◆ Topic 2:
 scheduled (1.33)
 prepares (1.33)
 mission (1.33)
 mars (1.33)
 nasa (1.33)
- ◆ Topic 3:
 new (3.33)
 unveils (1.33)
 tesla (1.33)
 truck (1.33)
 long (1.33)

```
# Plot term strengths
num_top_words = 5
for topic_idx, topic in enumerate(lda.components_):
    top_features_idx = topic.argsort()[:-num_top_words - 1:-1]
    top_features = [terms[i] for i in top_features_idx]
    top_weights = topic[top_features_idx]

    plt.figure(figsize=(8, 4))
    sns.barplot(x=top_weights, y=top_features, palette="viridis")
    plt.title(f"Topic {topic_idx + 1}: Top {num_top_words} Terms")
    plt.xlabel("Term Strength (Weight)")
    plt.ylabel("Term")
    plt.tight_layout()
```

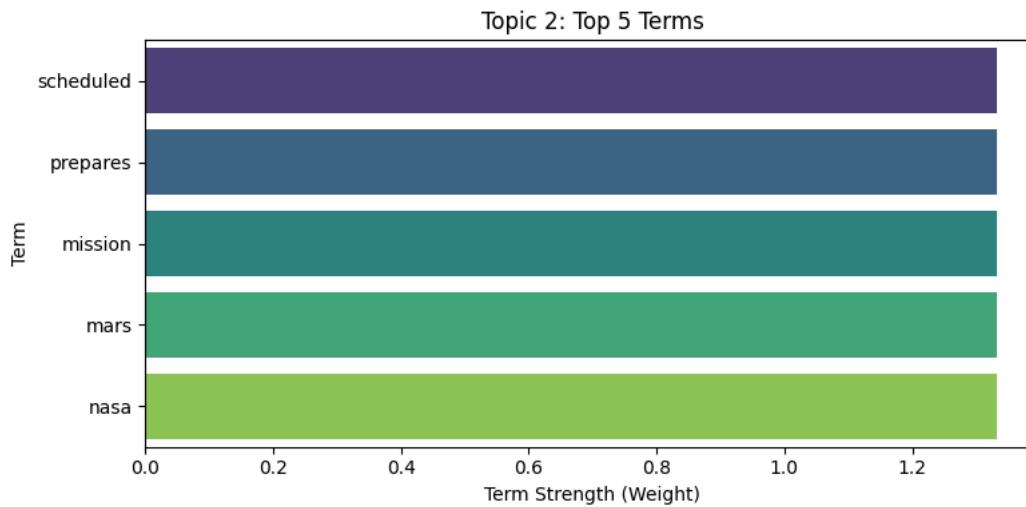
```
/tmp/ipython-input-4057204905.py:9: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and s
sns.barplot(x=top_weights, y=top_features, palette="viridis")
```



```
/tmp/ipython-input-4057204905.py:9: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and se
sns.barplot(x=top_weights, y=top_features, palette="viridis")
```



```
/tmp/ipython-input-4057204905.py:9: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and se
```

```
sns.barplot(x=top_weights, y=top_features, palette="viridis")
```

Topic 3: Top 5 Terms

