In [2]:
```
import re

def filter_regional_language(text, language_pattern):
    filtered_text = re.sub(language_pattern, '', text)
    return filtered_text

hindi_pattern = re.compile(r'[\u0900-\u097F]+')
input_sentence = input("Enter a sentence: ")
print("Input Sentence:", input_sentence)
filtered_sentence = filter_regional_language(input_sentence, hindi_pattern)
print("Filtered Sentence:", filtered_sentence)
```

```
Enter a sentence: Hello प्रसाद जावळे
Input Sentence: Hello प्रसाद जावळे
Filtered Sentence: Hello
```

In [4]:
```
#Stop word Filtraton

import re

def filter_stop_words(sentence) :
    stop_words = set([
        "i", "me", "my", "myself", "we", "our", "ours","you", "your",
        "yours", "he", "him", "his", "himself", "she", "her", "hers", "it",
        "its", "itself", "they", "them", "their", "theirs", "what",
        "which", "who", "whom", "this", "that", "these", "those", "am",
        "is", "are", "was", "were", "be", "been", "being", "have", "has",
        "had", "having", "do","through", "during", "before", "after", "above",
        "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "und
        "again", "further", "then", "once", "here", "there", "when", "where",
        "why", "how", "all", "any", "both", "each", "few", "more", "most",
        "other", "some", "such", "no", "nor", "not", "only", "own", "same",
        "so", "than", "too", "very", "s", "t", "can", "will", "just", "don",
        "should", "now"
    ])

    word_pattern = re.compile(r'\b\w+\b')
    filtered_sentence = word_pattern.sub(lambda match: match.group() if match.group
    return filtered_sentence

input_sentence = input("Enter a sentence: ")
filtered_sentence = filter_stop_words(input_sentence)
print("Filtered Sentence:", filtered_sentence)
```

```
Enter a sentence:  Despite their high frequency in language, these words carry lit
tle substantial meaning on their own. Their primary purpose is to facilitate the g
rammatical structure of sentences and paragraphs. However, when conducting text an
alysis, it is common practice to filter out these stop words to focus on the more
informative content words and phrases. This helps improve the quality and relevanc
e of the extracted information for various NLP tasks.
Filtered Sentence:  Despite  high frequency  language,  words carry little substan
tial meaning   .  primary purpose   facilitate the grammatical structure of senten
ces and paragraphs. However,  conducting text analysis,   common practice  filter
stop words  focus  the  informative content words and phrases.  helps improve the
quality and relevance of the extracted information for various NLP tasks.
```

In [3]:
```python
# Punctuation FIltration

import re

def filter_punctuation(sentence) :
    filtered_sentence = re.sub(r'[^\w\s]', '', sentence)
    return filtered_sentence

input_sentence = input("Enter a sentence: ")
filtered_sentence = filter_punctuation(input_sentence)
print("Filtered Sentence:", filtered_sentence)
```

```
Enter a sentence: Hello! Welcome world, how are you? Have a good day.
Filtered Sentence: Hello Welcome world how are you Have a good day
```

In [6]:
```python
#Email verification

import re

def is_valid_email(email) :
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email)

input_email = input("Enter an email address: ")

if is_valid_email(input_email) :
   print("Email is valid.")
else :
   print("Email is not valid.")
```

```
Enter an email address: prasad.32
Email is not valid.
```

In [5]:
```python
# Phone no validation

import re

def is_valid_phone_number(phone_number) :
    # Regular expression for basic phone number format validation
    pattern = r'^[2-9]\d{2}-\d{3}-\d{4}$'
    return re.match(pattern, phone_number)

input_phone_number = input("Enter a phone number : ")

if is_valid_phone_number(input_phone_number) :
  print("Phone number is valid.")
else :
  print("Phone number is not valid.")
```

```
Enter a phone number : 145879632a
Phone number is not valid.
```

In [7]:
```python
# Name validation

import re

def is_valid_name(name) :
```

```
    # Regular expression for name validation (letters and spaces only)
    pattern = r'^[a-zA-Z\s]+$'
    return re.match(pattern, name)

input_name = input("Enter a name : ")

if is_valid_name(input_name) :
    print("Name is valid.")
else :
    print("Name is not valid.")
```

```
Enter a name : Prasad
Name is valid.
```

In [7]:
```
input = input("Enter a sentence : ")
```

```
Enter a sentence : hello how are you
```

In [8]:
```
# Tokenization

list = []
for item in input.split(" ") :
  list.append(item)
print(list)
```

```
['hello', 'how', 'are', 'you']
```

In [1]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\pronn\anaconda3\lib\site-packages
(3.7)
Requirement already satisfied: click in c:\users\pronn\anaconda3\lib\site-packages
(from nltk) (8.0.4)
Requirement already satisfied: tqdm in c:\users\pronn\anaconda3\lib\site-packages
(from nltk) (4.64.1)
Requirement already satisfied: regex>=2021.8.3 in c:\users\pronn\anaconda3\lib\sit
e-packages (from nltk) (2022.7.9)
Requirement already satisfied: joblib in c:\users\pronn\anaconda3\lib\site-package
s (from nltk) (1.1.0)
Requirement already satisfied: colorama in c:\users\pronn\anaconda3\lib\site-packa
ges (from click->nltk) (0.4.5)
```

In [2]:
```
import re

def porter_stemmer(word) :
    # Step 1a
    if word.endswith('sses') :
        word = re.sub('sses$', 'ss', word)
    elif word.endswith('ies') :
        word = re.sub('ies$', 'i', word)
    elif word.endswith('ss') :
        word = re.sub('ss$', 'ss', word)
    else :
        word = re.sub('s$', '', word)

    # Step 1b
    if re.search(r'[aeiouy].*ed$', word) :
        word = re.sub('ed$', '', word)
        if re.search(r'[aeiouy]', word) :
            word = re.sub('ing$', '', word)

    # Step 1c
    if re.search(r'[aeiouy].*y$', word) :
        word = re.sub('y$', 'i', word)

    # Step 2
    if re.search(r'[aeiouy].*(ational|tional|enci|anci|izer|bli|alli|entli|eli|ousl
        word = re.sub(r'(ational|tional|enci|anci|izer|bli|alli|entli|eli|ousli|ous
        if re.search(r'[aeiouy].*(ational|tional|enci|anci|izer|bli|alli|entli|eli|
            word = re.sub(r'(ational|tional|enci|anci|izer|bli|alli|entli|eli|ousli

    # Step 3
    if re.search(r'[aeiouy].*(icate|ative|alize|iciti|ical|ful|ness)$', word) :
        word = re.sub(r'(icate|ative|alize|iciti|ical|ful|ness)$', '', word)
        if re.search(r'[aeiouy].*(icate|ative|alize|iciti|ical|ful|ness)$', word) :
            word = re.sub(r'(icate|ative|alize|iciti|ical|ful|ness)$', '', word)

    # Step 4
    if re.search(r'[aeiouy].*(al|ance|ence|er|ic|able|ible|ant|ement|ment|ent|ou|is
        word = re.sub(r'(al|ance|ence|er|ic|able|ible|ant|ement|ment|ent|ou|ism|ate
        if re.search(r'[aeiouy].*(al|ance|ence|er|ic|able|ible|ant|ement|ment|ent|o
            word = re.sub(r'(al|ance|ence|er|ic|able|ible|ant|ement|ment|ent|ou|ism
```

```
    # Step 5a
    if re.search(r'e$', word) :
        word = re.sub('e$', '', word)
        if re.search(r'[aeiouy].*e$', word) :
            word = re.sub('e$', '', word)

    # Step 5b
    if re.search(r'[aeiouy].*ll$', word) :
        word = re.sub('ll$', 'l', word)

    return word

words_to_stem = ["walking", "swayed", "happily", "seen", "children"]
stemmed_words = [porter_stemmer(word) for word in words_to_stem]
print(stemmed_words)
```

```
['walking', 'swai', 'happili', 'seen', 'children']
```

In [3]:

```
from nltk.stem import PorterStemmer

nltk_stemmer = PorterStemmer()

nltk_stemmed_words = [nltk_stemmer.stem(word) for word in words_to_stem]
print(nltk_stemmed_words)
```

```
C:\Users\pronn\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A N
umPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected
version 1.26.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
['walk', 'sway', 'happili', 'seen', 'children']
```

In [4]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import wordnet

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
```

Out[4]: True

Morphological Analysis

In [5]:
```
def analyze_word(word) :
    stemmer = PorterStemmer()
    stem = stemmer.stem(word)

    lemma = None
    synsets = wordnet.synsets(word)
    if synsets :
        lemma = synsets[0].lemmas()[0].name()

    plural = "plural" if lemma and stem != lemma else "singular"

    tense = "present"
    pos = None
    for synset in synsets :
        pos = synset.pos()
        if "past" in synset.name() :
            tense = "past"
            break
    return {
        "word" : word,
        "root" : stem,
        "singular/plural" : plural,
        "tense" : tense,
        "POS" : pos
    }

input_text = input("Enter a sentence: ")
tokens = word_tokenize(input_text)

print("{:<15} {:<15} {:<15} {:<15} {:<15}".format("word", "root", "singular/plural"
print("-"*75)
```

```
for token in tokens :
    analysis = analyze_word(token)
    print("{:<15} {:<15} {:<15} {:<15} {:<15}".format(
        analysis["word"],
        analysis["root"],
        analysis["singular/plural"],
        analysis["tense"],
        analysis["POS"]
    ))
```

```
Enter a sentence: Colorful skies looked pretty more importantly loved
word            root            singular/plural tense           POS
--------------------------------------------------------------------------
Colorful        color           plural          present         a
skies           sky             singular        present         v
looked          look            singular        present         v
pretty          pretti          plural          present         r
more            more            plural          present         r
importantly     importantli     plural          present         r
loved           love            singular        present         a
```

Word Generation

In [6]:

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet

def generate_word_forms(root) :
    # Singular and plural forms
    singular = root
    if root.endswith("s") or root.endswith("x") or root.endswith("z") or root.endsw
        plural = root + "es"
    elif root.endswith("y") and len(root) > 1 and root[-2] not in "aeiou" :
        plural = root[:-1] + "ies"
    else :
        plural = root + "s"

    # Comparative and superlative forms
    if root.endswith("e") :
        comparative = root + "r"
        superlative = root + "st"
    elif len(root) >= 2 and root[-1] not in "aeiou" and root[-2] not in "aeiou" :
        comparative = root + root[-1] + "er"
        superlative = root + root[-1] + "est"
    else :
        comparative = root + "er"
        superlative = root + "est"

    return {
        "singular" : singular,
        "plural" : plural,
        "comparative" : comparative,
        "superlative" : superlative
    }
```

```
input_text = input("Enter a sentence: ")
tokens = word_tokenize(input_text)

print("{:<15} {:<15} {:<15} {:<15} {:<15}".format("word", "singular", "plural", "co
print("-"*75)

for token in tokens :
    word_forms = generate_word_forms(token)
    print("{:<15} {:<15} {:<15} {:<15} {:<15}".format(
        token,
        word_forms["singular"],
        word_forms["plural"],
        word_forms["comparative"],
        word_forms["superlative"]
    ))
```

```
Enter a sentence: Short koala bears larger frank
word            singular        plural          comparative     superlative
---------------------------------------------------------------------------
Short           Short           Shorts          Shortter        Shorttest
koala           koala           koalas          koalaer         koalaest
bears           bears           bearses         bearsser        bearssest
larger          larger          largers         largerer        largerest
frank           frank           franks          frankker        frankkest
```

In [1]:
```
import nltk
import pandas as pd
import numpy as np
from collections import Counter
from tabulate import tabulate
nltk.download('punkt')
```

```
C:\Users\pronn\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this vers
of SciPy (detected version 1.26.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[1]: True

Word Frequency Count

In [2]:
```
text = input("Enter a text: ")

sentence = nltk.sent_tokenize(text)
tokens = [nltk.word_tokenize(sentence) for sentence in sentence]

print("Word  Frequency Count")
for sentence_tokens in tokens :
    word_counts = pd.value_counts(np.array(sentence_tokens))
    print(word_counts)

flat_tokens = [word.lower() for sentence_tokens in tokens for word in sentence_tokens]
unique_words = list(set(flat_tokens))

print (flat_tokens, unique_words)
```

```
Enter a text: nory was a catholic because her mother , and 's father his or had been
Word  Frequency Count
nory        1
was         1
a           1
catholic    1
because     1
her         1
mother      1
,           1
and         1
's          1
father      1
his         1
or          1
had         1
been        1
dtype: int64
['nory', 'was', 'a', 'catholic', 'because', 'her', 'mother', ',', 'and', "'s", 'father', 'his', 'or', 'had', 'been'] ['his', 'mother', 'or',
s', 'because', ',', "'s", 'nory', 'and', 'a', 'had', 'father', 'been', 'catholic', 'her']
```

Bigram Count Table

In [3]:
```
bigrams = list(nltk.bigrams(flat_tokens))
unigram_counter = Counter(flat_tokens)
matrix_size = len(unique_words)
matrix = [[0] * matrix_size for _ in range(matrix_size)]
word_to_index = {word : index for index, word in enumerate(unique_words)}

for bigram in bigrams :
    word1, word2 = bigram
    index1 = word_to_index[word1]
    index2 = word_to_index[word2]
    matrix[index1][index2] += 1

header = [''] + unique_words
matrix_with_headers = [[unique_words[i]] + matrix[i] for i in range(matrix_size)]

print("\nBigram Count Table:")
print(tabulate(matrix_with_headers, headers=header, tablefmt='grid'))
```

Bigram Count Table:

	his	mother	or	was	because	,	's	nory	and	a	had	father	been	catholic	her
his	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
mother	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
or	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
was	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
because	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
,	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
's	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
nory	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
and	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
had	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
father	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
been	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
catholic	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
her	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Bigram Probability table

In [4]:
```
bigrams = list(nltk.bigrams(flat_tokens))
unigram_counter = Counter(flat_tokens)
matrix_size = len(unique_words)
matrix = [[0] * matrix_size for _ in range(matrix_size)]
word_to_index = {word : index for index, word in enumerate(unique_words)}

for bigram in bigrams :
    word1, word2 = bigram
    index1 = word_to_index[word1]
    index2 = word_to_index[word2]
    matrix[index1][index2] += 1 / unigram_counter[word1]

header = [''] + unique_words
matrix_with_headers = [[unique_words[i]] + matrix[i] for i in range(matrix_size)]

print("\nBigram Probability Table:")
print(tabulate(matrix_with_headers, headers=header, tablefmt='grid'))
```

Bigram Probability Table:

	his	mother	or	was	because	,	's	nory	and	a	had	father	been	catholic	her
his	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
mother	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
or	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
was	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
because	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
,	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
's	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
nory	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
and	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
had	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
father	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
been	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
catholic	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Predicting next word

In [5]:
```
def predict_next_word(start_word, matrix_with_headers, word_to_index):
    if start_word in word_to_index :
        start_index = word_to_index[start_word]
        next_word_probs = matrix_with_headers[start_index][1:]
        max_prob_index = next_word_probs.index(max(next_word_probs))
        return unique_words[max_prob_index]
    else:
        return "Starting word not found in the text."

starting_word = input("Enter a starting word: ")
predicted_next_word = predict_next_word(starting_word, matrix_with_headers, word_to_index)
print("Predicted next word:", predicted_next_word)
```

```
Enter a starting word: mother
Predicted next word: ,
```

In [1]:
```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
C:\Users\pronn\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A N
umPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected
version 1.26.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

Out[1]:
```
True
```

In [2]:
```
from nltk.tokenize import word_tokenize

sentence = input("Enter a text: ")
words = word_tokenize(sentence)
```

```
Enter a text: We are living in an era where data is being generated at a significa
ntly fast pace.
```

In [3]:
```
from nltk import pos_tag

tags = pos_tag(words)
```

In [4]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Download NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Tokenize the sentence

words = word_tokenize(sentence)

# Perform part-of-speech tagging
tags = pos_tag(words)

# Print the tagged words and their tags
for word, tag in tags :
    print(f"{word} : {tag}")
```

```
We : PRP
are : VBP
living : VBG
in : IN
an : DT
era : NN
where : WRB
data : NN
is : VBZ
being : VBG
generated : VBN
at : IN
a : DT
significantly : RB
fast : JJ
pace : NN
. : .
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [5]:

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Download NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Tokenize the sentence

words = word_tokenize(sentence)

# Perform part-of-speech tagging
tags = pos_tag(words)

# Print the tagged words and their tags
for word, tag in tags :
    print(f"{word} : {tag}")
```

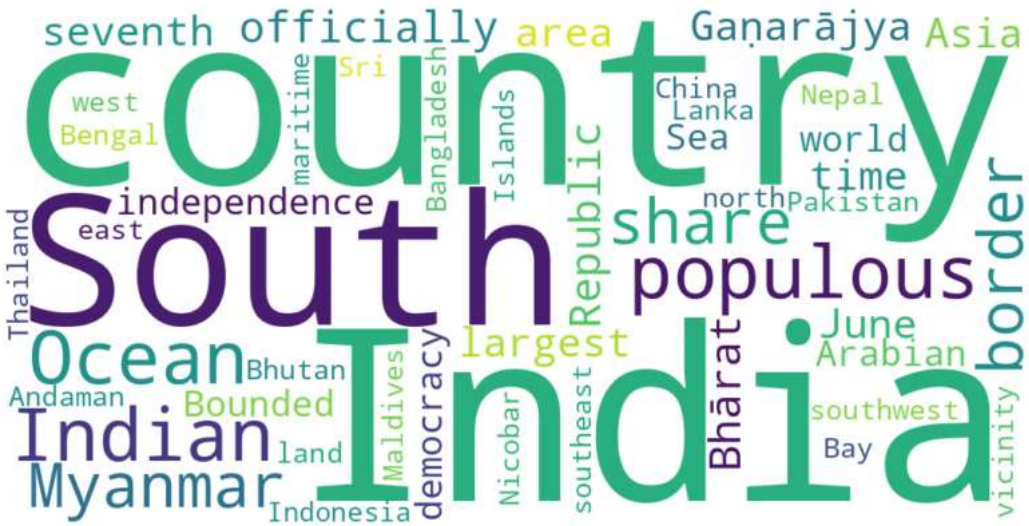
```
We : PRP
are : VBP
living : VBG
in : IN
an : DT
era : NN
where : WRB
data : NN
is : VBZ
being : VBG
generated : VBN
at : IN
a : DT
significantly : RB
fast : JJ
pace : NN
. : .
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

```
text = """
India, officially the Republic of India (Bhārat Gaṇarājya),is a country in South Asia.
It is the seventh-largest country by area; the most populous country as of June 2023; and from the time of its independence in 1947,
Bounded by the Indian Ocean on the south, the Arabian Sea on the southwest, and the Bay of Bengal on the southeast, it shares land b
In the Indian Ocean, India is in the vicinity of Sri Lanka and the Maldives; its Andaman and Nicobar Islands share a maritime border
"""
```

```
wordcloud = WordCloud(width=1000, height=500, background_color='white').generate(text)
```

```
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")  # Remove the axes
plt.show()
```



```
import spacy
nlp = spacy.load("en_core_web_sm")
```

```
text = """
India, officially the Republic of India (Bhārat Gaṇarājya),is a country in South Asia.
It is the seventh-largest country by area; the most populous country as of June 2023; and from the time of its independence in 1947,
Bounded by the Indian Ocean on the south, the Arabian Sea on the southwest, and the Bay of Bengal on the southeast, it shares land b
In the Indian Ocean, India is in the vicinity of Sri Lanka and the Maldives; its Andaman and Nicobar Islands share a maritime border
"""
```

```
label_mapping = {
    "PERSON": "Person",
    "ORG": "Organization",
    "GPE": "Geopolitical Entity",
    "DATE": "Date",
    "CARDINAL": "Cardinal Number",
    "LOC": "Location",
}

doc = nlp(text)
```

```
entities = [(ent.text, label_mapping.get(ent.label_, ent.label_)) for ent in doc.ents]

for entity, label in entities:
    print(f"Entity: {entity}, Label: {label}")
```

```
Entity: India, Label: Geopolitical Entity
Entity: the Republic of India, Label: Geopolitical Entity
Entity: South Asia, Label: Location
Entity: seventh, Label: ORDINAL
Entity: June 2023, Label: Date
Entity: 1947, Label: Date
Entity: the Indian Ocean, Label: Location
Entity: the Arabian Sea, Label: Location
Entity: the Bay of Bengal, Label: Location
Entity: Pakistan, Label: Geopolitical Entity
Entity: China, Label: Geopolitical Entity
Entity: Nepal, Label: Geopolitical Entity
Entity: Bhutan, Label: Geopolitical Entity
Entity: Bangladesh, Label: Geopolitical Entity
Entity: Myanmar, Label: Geopolitical Entity
Entity: the Indian Ocean, Label: Location
Entity: India, Label: Geopolitical Entity
Entity: Sri Lanka, Label: Geopolitical Entity
Entity: Maldives, Label: Geopolitical Entity
Entity: Andaman, Label: PRODUCT
Entity: Nicobar Islands, Label: Organization
Entity: Thailand, Label: Geopolitical Entity
Entity: Myanmar, Label: Geopolitical Entity
Entity: Indonesia, Label: Geopolitical Entity
```