```
import os
import nltk
#nltk.download()

#import nltk.corpus

# we will see what is mean by corpora and what all are availabel in
nltk python library
#print(os.listdir(nltk.data.find('corpora')))

#you get a lot of file , some of have some textual document, different
function associated with that function , stopwords, differenent type
of function
#for our example i will lets take consideration as brown & we will
understand what exactly nlp can do

#from nltk.corpus import brown
#brown.words()

#nltk.corpus.brown.fileids()

#nltk.corpus.gutenberg

#nltk.corpus.gutenberg.fileids()

# you can also create your own words

AI = '''Artificial Intelligence refers to the intelligence of
machines. This is in contrast to the natural intelligence of
humans and animals. With Artificial Intelligence, machines perform
functions such as learning, planning, reasoning and
problem-solving. Most noteworthy, Artificial Intelligence is the
simulation of human intelligence by machines.
It is probably the fastest-growing development in the World of
technology and innovation. Furthermore, many experts believe
AI could solve major challenges and crisis situations.'''

AI
```

```
'Artificial Intelligence refers to the intelligence of machines. This
is in contrast to the natural intelligence of \nhumans and animals.
With Artificial Intelligence, machines perform functions such as
learning, planning, reasoning and \nproblem-solving. Most noteworthy,
Artificial Intelligence is the simulation of human intelligence by
machines. \nIt is probably the fastest-growing development in the
World of technology and innovation. Furthermore, many experts believe\
nAI could solve major challenges and crisis situations.'
```

```
type(AI)
```

```
str
```

```python
from nltk.tokenize import word_tokenize

AI_tokens = word_tokenize(AI)
AI_tokens
```

```
['Artificial',
 'Intelligence',
 'refers',
 'to',
 'the',
 'intelligence',
 'of',
 'machines',
 '.',
 'This',
 'is',
 'in',
 'contrast',
 'to',
 'the',
 'natural',
 'intelligence',
 'of',
 'humans',
 'and',
 'animals',
 '.',
 'With',
 'Artificial',
 'Intelligence',
 ',',
 'machines',
 'perform',
 'functions',
 'such',
 'as',
 'learning',
 ',',
 'planning',
 ',',
 'reasoning',
 'and',
 'problem-solving',
 '.',
 'Most',
 'noteworthy',
 ',',
 'Artificial',
 'Intelligence',
 'is',
 'the',
 'simulation',
 'of',
 'human',
 'intelligence',
 'by',
 'machines',
 '.',
 'It',
 'is',
 'probably',
 'the',
 'fastest-growing',
 'development',
 'in',
 'the',
 'World',
 'of',
 'technology',
 'and',
 'innovation',
 '.',
 'Furthermore',
 ',',
 'many',
 'experts',
 'believe',
 'AI',
 'could',
 'solve',
 'major',
 'challenges',
 'and',
 'crisis',
 'situations',
 '.']
```

```python
len(AI_tokens)
```

```
81
```

```python
from nltk.tokenize import sent_tokenize
```

```python
AI_sent = sent_tokenize(AI)
AI_sent
```

```
['Artificial Intelligence refers to the intelligence of machines.',
 'This is in contrast to the natural intelligence of \nhumans and
animals.',
 'With Artificial Intelligence, machines perform functions such as
learning, planning, reasoning and \nproblem-solving.',
 'Most noteworthy, Artificial Intelligence is the simulation of human
intelligence by machines.',
 'It is probably the fastest-growing development in the World of
technology and innovation.',
 'Furthermore, many experts believe\nAI could solve major challenges
and crisis situations.']
```

```
len(AI_sent)
```

```
6
```

```
AI
```

```
'Artificial Intelligence refers to the intelligence of machines. This
is in contrast to the natural intelligence of \nhumans and animals.
With Artificial Intelligence, machines perform functions such as
learning, planning, reasoning and \nproblem-solving. Most noteworthy,
Artificial Intelligence is the simulation of human intelligence by
machines. \nIt is probably the fastest-growing development in the
World of technology and innovation. Furthermore, many experts believe\
nAI could solve major challenges and crisis situations.'
```

```
from nltk.tokenize import blankline_tokenize # GiVE YOU HOW MANY
PARAGRAPH
AI_blank = blankline_tokenize(AI)
AI_blank
#AI_blank
```

```
['Artificial Intelligence refers to the intelligence of machines. This
is in contrast to the natural intelligence of \nhumans and animals.
With Artificial Intelligence, machines perform functions such as
learning, planning, reasoning and \nproblem-solving. Most noteworthy,
Artificial Intelligence is the simulation of human intelligence by
machines. \nIt is probably the fastest-growing development in the
World of technology and innovation. Furthermore, many experts believe\
nAI could solve major challenges and crisis situations.']
```

```
len(AI_blank)
```

```
1
```

```
# NEXT WE WILL SEE HOW WE WILL USE UNI-GRAM,BI-GRAM,TRI-GRAM USING
NLTK
```

```
from nltk.util import bigrams,trigrams,ngrams
```

```
string = 'the best and most beautifull thing in the world cannot be
seen or even touched,they must be felt with heart'
quotes_tokens = nltk.word_tokenize(string)
```

```
quotes_tokens
```

```
['the',
 'best',
 'and',
 'most',
 'beautifull',
 'thing',
 'in',
 'the',
 'world',
 'can',
 'not',
 'be',
 'seen',
 'or',
 'even',
 'touched',
 ',',
 'they',
 'must',
 'be',
 'felt',
 'with',
 'heart']
```

```
len(quotes_tokens)
```

```
23
```

```
quotes_bigrams = list(nltk.bigrams(quotes_tokens))
quotes_bigrams
```

```
[('the', 'best'),
 ('best', 'and'),
 ('and', 'most'),
 ('most', 'beautifull'),
 ('beautifull', 'thing'),
 ('thing', 'in'),
 ('in', 'the'),
 ('the', 'world'),
 ('world', 'can'),
 ('can', 'not'),
 ('not', 'be'),
 ('be', 'seen'),
 ('seen', 'or'),
 ('or', 'even'),
 ('even', 'touched'),
 ('touched', ','),
 (',', 'they'),
 ('they', 'must'),
 ('must', 'be'),
 ('be', 'felt'),
 ('felt', 'with'),
 ('with', 'heart')]
```

```
quotes_tokens
```

```
['the',
 'best',
 'and',
 'most',
 'beautifull',
 'thing',
 'in',
 'the',
 'world',
 'can',
 'not',
 'be',
 'seen',
 'or',
 'even',
 'touched',
 ',',
 'they',
 'must',
 'be',
 'felt',
 'with',
 'heart']
```

```
quotes_trigrams = list(nltk.trigrams(quotes_tokens))
quotes_trigrams
```

```
[('the', 'best', 'and'),
 ('best', 'and', 'most'),
 ('and', 'most', 'beautifull'),
 ('most', 'beautifull', 'thing'),
 ('beautifull', 'thing', 'in'),
 ('thing', 'in', 'the'),
 ('in', 'the', 'world'),
 ('the', 'world', 'can'),
 ('world', 'can', 'not'),
 ('can', 'not', 'be'),
 ('not', 'be', 'seen'),
 ('be', 'seen', 'or'),
 ('seen', 'or', 'even'),
 ('or', 'even', 'touched'),
 ('even', 'touched', ','),
 ('touched', ',', 'they'),
 (',', 'they', 'must'),
 ('they', 'must', 'be'),
 ('must', 'be', 'felt'),
 ('be', 'felt', 'with'),
 ('felt', 'with', 'heart')]
```

```
quotes_trigrams = list(nltk.ngrams(quotes_tokens))
quotes_trigrams
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-a46038d19d4e> in <module>
----> 1 quotes_trigrams = list(nltk.ngrams(quotes_tokens))
      2 quotes_trigrams

TypeError: ngrams() missing 1 required positional argument: 'n'
```

```
quotes_ngrams = list(nltk.ngrams(quotes_tokens, 4))
quotes_ngrams

#it has given n-gram of length 4

len(quotes_tokens)
```

```
23
```

```
quotes_ngrams_1 = list(nltk.ngrams(quotes_tokens, 5))
quotes_ngrams_1
```

```
[('the', 'best', 'and', 'most', 'beautifull'),
 ('best', 'and', 'most', 'beautifull', 'thing'),
 ('and', 'most', 'beautifull', 'thing', 'in'),
 ('most', 'beautifull', 'thing', 'in', 'the'),
 ('beautifull', 'thing', 'in', 'the', 'world'),
 ('thing', 'in', 'the', 'world', 'can'),
 ('in', 'the', 'world', 'can', 'not'),
 ('the', 'world', 'can', 'not', 'be'),
 ('world', 'can', 'not', 'be', 'seen'),
 ('can', 'not', 'be', 'seen', 'or'),
 ('not', 'be', 'seen', 'or', 'even'),
 ('be', 'seen', 'or', 'even', 'touched'),
 ('seen', 'or', 'even', 'touched', ','),
 ('or', 'even', 'touched', ',', 'they'),
 ('even', 'touched', ',', 'they', 'must'),
 ('touched', ',', 'they', 'must', 'be'),
 (',', 'they', 'must', 'be', 'felt'),
 ('they', 'must', 'be', 'felt', 'with'),
 ('must', 'be', 'felt', 'with', 'heart')]
```

```
quotes_ngrams = list(nltk.ngrams(quotes_tokens, 9))
quotes_ngrams
```

```
[('the', 'best', 'and', 'most', 'beautifull', 'thing', 'in', 'the',
'world'),
 ('best', 'and', 'most', 'beautifull', 'thing', 'in', 'the', 'world',
'can'),
 ('and', 'most', 'beautifull', 'thing', 'in', 'the', 'world', 'can',
'not'),
 ('most', 'beautifull', 'thing', 'in', 'the', 'world', 'can', 'not',
'be'),
 ('beautifull', 'thing', 'in', 'the', 'world', 'can', 'not', 'be',
'seen'),
 ('thing', 'in', 'the', 'world', 'can', 'not', 'be', 'seen', 'or'),
 ('in', 'the', 'world', 'can', 'not', 'be', 'seen', 'or', 'even'),
 ('the', 'world', 'can', 'not', 'be', 'seen', 'or', 'even',
'touched'),
 ('world', 'can', 'not', 'be', 'seen', 'or', 'even', 'touched', ','),
 ('can', 'not', 'be', 'seen', 'or', 'even', 'touched', ',', 'they'),
 ('not', 'be', 'seen', 'or', 'even', 'touched', ',', 'they', 'must'),
 ('be', 'seen', 'or', 'even', 'touched', ',', 'they', 'must', 'be'),
 ('seen', 'or', 'even', 'touched', ',', 'they', 'must', 'be', 'felt'),
 ('or', 'even', 'touched', ',', 'they', 'must', 'be', 'felt', 'with'),
 ('even', 'touched', ',', 'they', 'must', 'be', 'felt', 'with',
'heart')]
```

```
# Next we need to make some changes in tokens and that is called as
stemming, stemming will gives you root form of an word
# also we will see some root form of the word & limitation of the word

#porter-stemmer
from nltk.stem import PorterStemmer
pst = PorterStemmer()

pst.stem('having') #stem will gives you the root form of the word
```

```
'have'
```

```
pst.stem('affection')
```

```
'affect'
```

```
pst.stem('playing')
```

```
'play'
```

```
pst.stem('give')
```

```
'give'
```

```
words_to_stem=['give','giving','given','gave']
for words in words_to_stem:
    print(words+  ':' + pst.stem(words))
```

```
give:give
giving:give
given:given
gave:gave
```

```
pst.stem('playing')
```

```
'play'
```

```
words_to_stem=['give','giving','given','gave','thinking', 'loving',
'final', 'finalized', 'finally']
# i am giving these different words to stem, using porter stemmer we
get the output

for words in words_to_stem:
    print(words+ ':' +pst.stem(words))

#in porterstemmer removes ing and replaces with e
```

```
give:give
giving:give
given:given
gave:gave
thinking:think
loving:love
final:final
finalized:final
finally:final
```

```
#another stemmer known as lencastemmer stemmer and lets see what the
different we will get hear
#stem the same thing using lencastemmer

from nltk.stem import LancasterStemmer
lst = LancasterStemmer()
for words in words_to_stem:
    print(words + ':' + lst.stem(words))

# lancasterstemmer is more aggresive then the porterstemmer
```

```
give:giv
giving:giv
given:giv
gave:gav
thinking:think
loving:lov
final:fin
```

```
finalized:fin
finally:fin

words_to_stem=['give','giving','given','gave','thinking', 'loving',
'final', 'finalized', 'finally']
# i am giving these different words to stem, using porter stemmer we
get the output

for words in words_to_stem:
    print(words+ ':' +pst.stem(words))

give:give
giving:give
given:given
gave:gave
thinking:think
loving:love
final:final
finalized:final
finally:final

#we have another stemmer called as snowball stemmer lets see about
this snowball stemmer

from nltk.stem import SnowballStemmer
sbst = SnowballStemmer('english')
for words in words_to_stem:
    print(words+ ':' +sbst.stem(words))

#snowball stemmer is same as portstemmer
#different type of stemmer used based on different type of task
#if you want to see how many type of giv has occured then we will see
the lancaster stemmer

give:give
giving:give
given:given
gave:gave
thinking:think
loving:love
final:final
finalized:final
finally:final

#sometime stemming does not work & lets say e.g - fish,fishes &
fishing all of them belongs to root word fish,
#one hand stemming will cut the end & lemmatization will take into the
morphological analysis of the word

from nltk.stem import wordnet
from nltk.stem import WordNetLemmatizer
```

```
word_lem = WordNetLemmatizer()

#Hear we are going to wordnet dictionary & we are going to import the
wordnet lematizer

words_to_stem

['give',
 'giving',
 'given',
 'gave',
 'thinking',
 'loving',
 'final',
 'finalized',
 'finally']

#word_lem.lemmatize('corpora') #we get output as corpus

#refers to a collection of texts. Such collections may be formed of a
single language of texts, or can span multiple languages -- there are
numerous reasons for which multilingual corpora (the plural of corpus)
may be useful

for words in words_to_stem:
    print(words+ ':' +word_lem.lemmatize(words))

give:give
giving:giving
given:given
gave:gave
thinking:thinking
loving:loving
final:final
finalized:finalized
finally:finally

pst.stem('final')

'final'

lst.stem('finally')

'fin'

sbst.stem('finalized')

'final'

lst.stem('final')

'fin'
```

```
lst.stem('finalized')
```

```
'fin'
```

```
# there is other concept called POS (part of speech) which deals with
subject, noun, pronoun but before of this lets go with other concept
called STOPWORDS
# STOPWORDS = i, is, as,at, on, about & nltk has their own list of
stopewords

from nltk.corpus import stopwords

stopwords.words('english')
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her',
 'hers',
 'herself',
 'it',
 "it's",
 'its',
 'itself',
 'they',
 'them',
 'their',
 'theirs',
 'themselves',
 'what',
 'which',
 'who',
 'whom',
 'this',
 'that',
 "that'll",
 'these',
 'those',
 'am',
 'is',
 'are',
 'was',
 'were',
 'be',
 'been',
 'being',
 'have',
 'has',
 'had',
 'having',
 'do',
 'does',
 'did',
 'doing',
 'a',
 'an',
 'the',
 'and',
 'but',
 'if',
 'or',
 'because',
 'as',
 'until',
 'while',
 'of',
 'at',
 'by',
 'for',
 'with',
 'about',
 'against',
 'between',
 'into',
 'through',
 'during',
 'before',
 'after',
 'above',
 'below',
 'to',
 'from',
 'up',
 'down',
 'in',
 'out',
 'on',
 'off',
 'over',
 'under',
 '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',
 "don't",
 'should',
 "should've",
 'now',
 'd',
 'll',
 'm',
 'o',
 're',
 've',
 'y',
 'ain',
 'aren',
 "aren't",
 'couldn',
 "couldn't",
 'didn',
 "didn't",
 'doesn',
 "doesn't",
 'hadn',
 "hadn't",
 'hasn',
 "hasn't",
 'haven',
 "haven't",
 'isn',
 "isn't",
 'ma',
 'mightn',
 "mightn't",
 'mustn',
 "mustn't",
 'needn',
 "needn't",
 'shan',
 "shan't",
 'shouldn',
 "shouldn't",
 'wasn',
 "wasn't",
 'weren',
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

```
len(stopwords.words('english'))
```

```
179
```

```python
stopwords.words('spanish')
```

```
['de',
 'la',
 'que',
 'el',
 'en',
 'y',
 'a',
 'los',
 'del',
 'se',
 'las',
 'por',
 'un',
 'para',
 'con',
 'no',
 'una',
 'su',
 'al',
 'lo',
 'como',
 'más',
 'pero',
 'sus',
 'le',
 'ya',
 'o',
 'este',
 'sí',
 'porque',
 'esta',
 'entre',
 'cuando',
 'muy',
 'sin',
 'sobre',
 'también',
 'me',
 'hasta',
 'hay',
 'donde',
 'quien',
 'desde',
 'todo',
 'nos',
 'durante',
 'todos',
 'uno',
 'les',
 'ni',
 'contra',
 'otros',
 'ese',
 'eso',
 'ante',
 'ellos',
 'e',
 'esto',
 'mí',
 'antes',
 'algunos',
 'qué',
 'unos',
 'yo',
 'otro',
 'otras',
 'otra',
 'él',
 'tanto',
 'esa',
 'estos',
 'mucho',
 'quienes',
 'nada',
 'muchos',
 'cual',
 'poco',
 'ella',
 'estar',
 'estas',
 'algunas',
 'algo',
 'nosotros',
 'mi',
 'mis',
 'tú',
 'te',
 'ti',
 'tu',
 'tus',
 'ellas',
 'nosotras',
 'vosotros',
 'vosotras',
 'os',
 'mío',
 'mía',
 'míos',
 'mías',
 'tuyo',
 'tuya',
 'tuyos',
 'tuyas',
 'suyo',
 'suya',
 'suyos',
 'suyas',
 'nuestro',
 'nuestra',
 'nuestros',
 'nuestras',
 'vuestro',
 'vuestra',
 'vuestros',
 'vuestras',
 'esos',
 'esas',
 'estoy',
 'estás',
 'está',
 'estamos',
 'estáis',
 'están',
 'esté',
 'estés',
 'estemos',
 'estéis',
 'estén',
 'estaré',
 'estarás',
 'estará',
 'estaremos',
 'estaréis',
 'estarán',
 'estaría',
 'estarías',
 'estaríamos',
 'estaríais',
 'estarían',
 'estaba',
 'estabas',
 'estábamos',
 'estabais',
 'estaban',
 'estuve',
 'estuviste',
 'estuvo',
 'estuvimos',
 'estuvisteis',
 'estuvieron',
 'estuviera',
 'estuvieras',
 'estuviéramos',
 'estuvierais',
 'estuvieran',
 'estuviese',
 'estuvieses',
 'estuviésemos',
 'estuvieseis',
 'estuviesen',
 'estando',
 'estado',
 'estada',
 'estados',
 'estadas',
 'estad',
 'he',
 'has',
 'ha',
 'hemos',
 'habéis',
 'han',
 'haya',
 'hayas',
 'hayamos',
 'hayáis',
 'hayan',
 'habré',
 'habrás',
 'habrá',
 'habremos',
 'habréis',
 'habrán',
 'habría',
 'habrías',
 'habríamos',
 'habríais',
 'habrían',
 'había',
 'habías',
 'habíamos',
 'habíais',
 'habían',
 'hube',
 'hubiste',
 'hubo',
 'hubimos',
 'hubisteis',
 'hubieron',
 'hubiera',
 'hubieras',
 'hubiéramos',
 'hubierais',
 'hubieran',
 'hubiese',
 'hubieses',
 'hubiésemos',
 'hubieseis',
 'hubiesen',
 'habiendo',
 'habido',
 'habida',
 'habidos',
 'habidas',
 'soy',
 'eres',
 'es',
 'somos',
 'sois',
 'son',
 'sea',
 'seas',
 'seamos',
 'seáis',
 'sean',
 'seré',
 'serás',
 'será',
 'seremos',
 'seréis',
 'serán',
 'sería',
 'serías',
 'seríamos',
 'seríais',
 'serían',
 'era',
 'eras',
 'éramos',
 'erais',
 'eran',
 'fui',
 'fuiste',
 'fue',
 'fuimos',
 'fuisteis',
 'fueron',
 'fuera',
 'fueras',
 'fuéramos',
 'fuerais',
 'fueran',
 'fuese',
 'fueses',
 'fuésemos',
 'fueseis',
 'fuesen',
 'sintiendo',
 'sentido',
 'sentida',
 'sentidos',
 'sentidas',
 'siente',
 'sentid',
 'tengo',
 'tienes',
 'tiene',
 'tenemos',
 'tenéis',
 'tienen',
 'tenga',
 'tengas',
 'tengamos',
 'tengáis',
 'tengan',
 'tendré',
 'tendrás',
 'tendrá',
 'tendremos',
 'tendréis',
 'tendrán',
 'tendría',
 'tendrías',
 'tendríamos',
 'tendríais',
 'tendrían',
 'tenía',
 'tenías',
 'teníamos',
 'teníais',
 'tenían',
 'tuve',
 'tuviste',
 'tuvo',
 'tuvimos',
 'tuvisteis',
 'tuvieron',
 'tuviera',
 'tuvieras',
 'tuviéramos',
 'tuvierais',
 'tuvieran',
 'tuviese',
 'tuvieses',
 'tuviésemos',
 'tuvieseis',
 'tuviesen',
 'teniendo',
 'tenido',
 'tenida',
 'tenidos',
 'tenidas',
 'tened']
```

```
len(stopwords.words('spanish'))
```

```
313
```

```
stopwords.words('french')
```

```
['au',
 'aux',
 'avec',
 'ce',
 'ces',
 'dans',
 'de',
 'des',
 'du',
 'elle',
 'en',
 'et',
 'eux',
 'il',
 'ils',
 'je',
 'la',
 'le',
 'les',
 'leur',
 'lui',
 'ma',
 'mais',
 'me',
 'même',
 'mes',
 'moi',
 'mon',
 'ne',
 'nos',
 'notre',
 'nous',
 'on',
 'ou',
 'par',
 'pas',
 'pour',
 'qu',
 'que',
 'qui',
 'sa',
 'se',
 'ses',
 'son',
 'sur',
 'ta',
 'te',
 'tes',
 'toi',
 'ton',
 'tu',
 'un',
 'une',
 'vos',
 'votre',
 'vous',
 'c',
 'd',
 'j',
 'l',
 'à',
 'm',
 'n',
 's',
 't',
 'y',
 'été',
 'étée',
 'étées',
 'étés',
 'étant',
 'étante',
 'étants',
 'étantes',
 'suis',
 'es',
 'est',
 'sommes',
 'êtes',
 'sont',
 'serai',
 'seras',
 'sera',
 'serons',
 'serez',
 'seront',
 'serais',
 'serait',
 'serions',
 'seriez',
 'seraient',
 'étais',
 'était',
 'étions',
 'étiez',
 'étaient',
 'fus',
 'fut',
 'fûmes',
 'fûtes',
 'furent',
 'sois',
 'soit',
 'soyons',
 'soyez',
 'soient',
 'fusse',
 'fusses',
 'fût',
 'fussions',
 'fussiez',
 'fussent',
 'ayant',
 'ayante',
 'ayantes',
 'ayants',
 'eu',
 'eue',
 'eues',
 'eus',
 'ai',
 'as',
 'avons',
 'avez',
 'ont',
 'aurai',
 'auras',
 'aura',
 'aurons',
 'aurez',
 'auront',
 'aurais',
 'aurait',
 'aurions',
 'auriez',
 'auraient',
 'avais',
 'avait',
 'avions',
 'aviez',
 'avaient',
 'eut',
 'eûmes',
 'eûtes',
 'eurent',
 'aie',
 'aies',
 'ait',
 'ayons',
 'ayez',
 'aient',
 'eusse',
 'eusses',
 'eût',
 'eussions',
 'eussiez',
 'eussent']
```

```
len(stopwords.words('french'))
```

```
157
```

```
stopwords.words('german')
```

```
['aber',
 'alle',
 'allem',
 'allen',
 'aller',
 'alles',
 'als',
 'also',
 'am',
 'an',
 'ander',
 'andere',
 'anderem',
 'anderen',
 'anderer',
 'anderes',
 'anderm',
 'andern',
 'anderr',
 'anders',
 'auch',
 'auf',
 'aus',
 'bei',
 'bin',
 'bis',
 'bist',
 'da',
 'damit',
 'dann',
 'der',
 'den',
 'des',
 'dem',
 'die',
 'das',
 'dass',
 'daß',
 'derselbe',
 'derselben',
 'denselben',
 'desselben',
 'demselben',
 'dieselbe',
 'dieselben',
 'dasselbe',
 'dazu',
 'dein',
 'deine',
 'deinem',
 'deinen',
 'deiner',
 'deines',
 'denn',
 'derer',
 'dessen',
 'dich',
 'dir',
 'du',
 'dies',
 'diese',
 'diesem',
 'diesen',
 'dieser',
 'dieses',
 'doch',
 'dort',
 'durch',
 'ein',
 'eine',
 'einem',
 'einen',
 'einer',
 'eines',
 'einig',
 'einige',
 'einigem',
 'einigen',
 'einiger',
 'einiges',
 'einmal',
 'er',
 'ihn',
 'ihm',
 'es',
 'etwas',
 'euer',
 'eure',
 'eurem',
 'euren',
 'eurer',
 'eures',
 'für',
 'gegen',
 'gewesen',
 'hab',
 'habe',
 'haben',
 'hat',
 'hatte',
 'hatten',
 'hier',
 'hin',
 'hinter',
 'ich',
 'mich',
 'mir',
 'ihr',
 'ihre',
 'ihrem',
 'ihren',
 'ihrer',
 'ihres',
 'euch',
 'im',
 'in',
 'indem',
 'ins',
 'ist',
 'jede',
 'jedem',
 'jeden',
 'jeder',
 'jedes',
 'jene',
 'jenem',
 'jenen',
 'jener',
 'jenes',
 'jetzt',
 'kann',
 'kein',
 'keine',
 'keinem',
 'keinen',
 'keiner',
 'keines',
 'können',
 'könnte',
 'machen',
 'man',
 'manche',
 'manchem',
 'manchen',
 'mancher',
 'manches',
 'mein',
 'meine',
 'meinem',
 'meinen',
 'meiner',
 'meines',
 'mit',
 'muss',
 'musste',
 'nach',
 'nicht',
 'nichts',
 'noch',
 'nun',
 'nur',
 'ob',
 'oder',
 'ohne',
 'sehr',
 'sein',
 'seine',
 'seinem',
 'seinen',
 'seiner',
 'seines',
 'selbst',
 'sich',
 'sie',
 'ihnen',
 'sind',
 'so',
 'solche',
 'solchem',
 'solchen',
 'solcher',
 'solches',
 'soll',
 'sollte',
 'sondern',
 'sonst',
 'über',
 'um',
 'und',
 'uns',
 'unsere',
 'unserem',
 'unseren',
 'unser',
 'unseres',
 'unter',
 'viel',
 'vom',
 'von',
 'vor',
 'während',
 'war',
 'waren',
 'warst',
 'was',
 'weg',
 'weil',
 'weiter',
 'welche',
 'welchem',
 'welchen',
 'welcher',
 'welches',
 'wenn',
 'werde',
 'werden',
 'wie',
 'wieder',
 'will',
 'wir',
 'wird',
 'wirst',
 'wo',
 'wollen',
 'wollte',
 'würde',
 'würden',
 'zu',
 'zum',
 'zur',
 'zwar',
 'zwischen']
```

```
len(stopwords.words('german'))
```

```
232
```

```
stopwords.words('hindi') # research phase
```

```
-----------------------------------------------------------------------
-----
OSError                                   Traceback (most recent call
last)
<ipython-input-56-df50742b7e1b> in <module>
----> 1 stopwords.words('hindi') # research phase

~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
words(self, fileids, ignore_lines_startswith)
     23         return [
     24             line
---> 25             for line in line_tokenize(self.raw(fileids))
     26             if not line.startswith(ignore_lines_startswith)
     27         ]
```

```
~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
raw(self, fileids)
     32         elif isinstance(fileids, string_types):
     33             fileids = [fileids]
---> 34         return concat([self.open(f).read() for f in fileids])
     35
     36

~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
<listcomp>(.0)
     32         elif isinstance(fileids, string_types):
     33             fileids = [fileids]
---> 34         return concat([self.open(f).read() for f in fileids])
     35
     36

~\anaconda3\lib\site-packages\nltk\corpus\reader\api.py in open(self,
file)
    211         """
    212         encoding = self.encoding(file)
--> 213         stream = self._root.join(file).open(encoding)
    214         return stream
    215

~\anaconda3\lib\site-packages\nltk\data.py in join(self, fileid)
    353     def join(self, fileid):
    354         _path = os.path.join(self._path, fileid)
--> 355         return FileSystemPathPointer(_path)
    356
    357     def __repr__(self):

~\anaconda3\lib\site-packages\nltk\compat.py in _decorator(*args,
**kwargs)
    226     def _decorator(*args, **kwargs):
    227         args = (args[0], add_py3_data(args[1])) + args[2:]
--> 228         return init_func(*args, **kwargs)
    229
    230     return wraps(init_func)(_decorator)

~\anaconda3\lib\site-packages\nltk\data.py in __init__(self, _path)
    331         _path = os.path.abspath(_path)
    332         if not os.path.exists(_path):
--> 333             raise IOError('No such file or directory: %r' %
_path)
    334         self._path = _path
    335

OSError: No such file or directory: 'C:\\Users\\kdata\\AppData\\
Roaming\\nltk_data\\corpora\\stopwords\\hindi'
```

```
stopwords.words('marathi')

----------------------------------------------------------------------
-----
OSError                                   Traceback (most recent call
last)
<ipython-input-57-4f60cd586bb5> in <module>
----> 1 stopwords.words('marathi')

~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
words(self, fileids, ignore_lines_startswith)
     23         return [
     24             line
---> 25             for line in line_tokenize(self.raw(fileids))
     26             if not line.startswith(ignore_lines_startswith)
     27         ]

~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
raw(self, fileids)
     32         elif isinstance(fileids, string_types):
     33             fileids = [fileids]
---> 34         return concat([self.open(f).read() for f in fileids])
     35 
     36 

~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
<listcomp>(.0)
     32         elif isinstance(fileids, string_types):
     33             fileids = [fileids]
---> 34         return concat([self.open(f).read() for f in fileids])
     35 
     36 

~\anaconda3\lib\site-packages\nltk\corpus\reader\api.py in open(self,
file)
    211         """
    212         encoding = self.encoding(file)
--> 213         stream = self._root.join(file).open(encoding)
    214         return stream
    215 

~\anaconda3\lib\site-packages\nltk\data.py in join(self, fileid)
    353     def join(self, fileid):
    354         _path = os.path.join(self._path, fileid)
--> 355         return FileSystemPathPointer(_path)
    356 
    357     def __repr__(self):

~\anaconda3\lib\site-packages\nltk\compat.py in _decorator(*args,
**kwargs)
```

```
    226     def _decorator(*args, **kwargs):
    227         args = (args[0], add_py3_data(args[1])) + args[2:]
--> 228         return init_func(*args, **kwargs)
    229 
    230     return wraps(init_func)(_decorator)

~\anaconda3\lib\site-packages\nltk\data.py in __init__(self, _path)
    331         _path = os.path.abspath(_path)
    332         if not os.path.exists(_path):
--> 333             raise IOError('No such file or directory: %r' %
_path)
    334         self._path = _path
    335 

OSError: No such file or directory: 'C:\\Users\\kdata\\AppData\\
Roaming\\nltk_data\\corpora\\stopwords\\marathi'
```

```
stopwords.words('telugu')
```

```
---------------------------------------------------------------------------
-----
OSError                                   Traceback (most recent call
last)
<ipython-input-58-d1b3db3d8785> in <module>
----> 1 stopwords.words('telugu')

~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
words(self, fileids, ignore_lines_startswith)
     23         return [
     24             line
---> 25             for line in line_tokenize(self.raw(fileids))
     26             if not line.startswith(ignore_lines_startswith)
     27         ]

~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
raw(self, fileids)
     32         elif isinstance(fileids, string_types):
     33             fileids = [fileids]
---> 34         return concat([self.open(f).read() for f in fileids])
     35 
     36 

~\anaconda3\lib\site-packages\nltk\corpus\reader\wordlist.py in
<listcomp>(.0)
     32         elif isinstance(fileids, string_types):
     33             fileids = [fileids]
---> 34         return concat([self.open(f).read() for f in fileids])
     35 
     36 
```

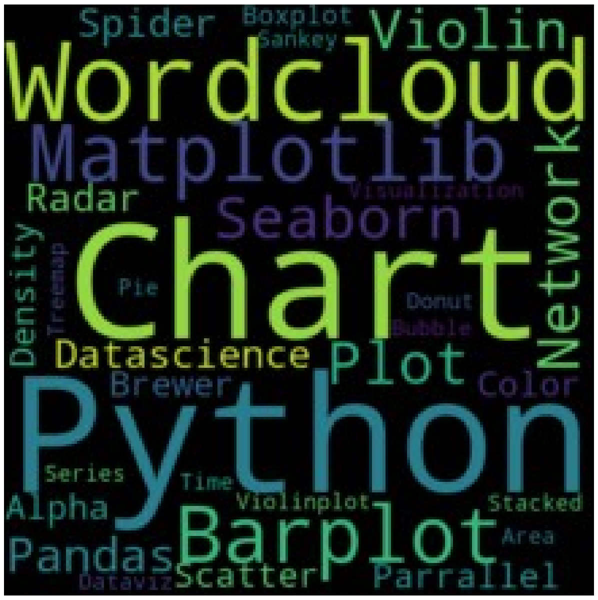
```
~\anaconda3\lib\site-packages\nltk\corpus\reader\api.py in open(self,
file)
    211         """
    212         encoding = self.encoding(file)
--> 213         stream = self._root.join(file).open(encoding)
    214         return stream
    215

~\anaconda3\lib\site-packages\nltk\data.py in join(self, fileid)
    353     def join(self, fileid):
    354         _path = os.path.join(self._path, fileid)
--> 355         return FileSystemPathPointer(_path)
    356
    357     def __repr__(self):

~\anaconda3\lib\site-packages\nltk\compat.py in _decorator(*args,
**kwargs)
    226     def _decorator(*args, **kwargs):
    227         args = (args[0], add_py3_data(args[1])) + args[2:]
--> 228         return init_func(*args, **kwargs)
    229
    230     return wraps(init_func)(_decorator)

~\anaconda3\lib\site-packages\nltk\data.py in __init__(self, _path)
    331         _path = os.path.abspath(_path)
    332         if not os.path.exists(_path):
--> 333             raise IOError('No such file or directory: %r' %
_path)
    334         self._path = _path
    335

OSError: No such file or directory: 'C:\\Users\\kdata\\AppData\\
Roaming\\nltk_data\\corpora\\stopwords\\telugu'
```

```python
# first we need to compile from re module to create string that
matched any digits or special character
import re
punctuation = re.compile(r'[-.?!,:;()|0-9]')

#now i am going to create to empty list and append the word without
any punctuation & naming this as a post punctuation

punctuation
```

```
re.compile(r'[-.?!,:;()|0-9]', re.UNICODE)
```

```python
AI

AI_tokens

len(AI_tokens)
```

```
# we will see how to work in POS using NLTK library

sent = 'kathy is a natural when it comes to drawing'
sent_tokens = word_tokenize(sent)
sent_tokens

# first we will tokenize usning word_tokenize & then we will use
pos_tag on all of the tokens

['kathy', 'is', 'a', 'natural', 'when', 'it', 'comes', 'to',
'drawing']

for token in sent_tokens:
    print(nltk.pos_tag([token]))

[('kathy', 'NN')]
[('is', 'VBZ')]
[('a', 'DT')]
[('natural', 'JJ')]
[('when', 'WRB')]
[('it', 'PRP')]
[('comes', 'VBZ')]
[('to', 'TO')]
[('drawing', 'VBG')]

sent2 = 'john is eating a delicious cake'
sent2_tokens = word_tokenize(sent2)

for token in sent2_tokens:
    print(nltk.pos_tag([token]))

[('john', 'NN')]
[('is', 'VBZ')]
[('eating', 'VBG')]
[('a', 'DT')]
[('delicious', 'JJ')]
[('cake', 'NN')]

# Another concept of POS is called NER ( NAMED ENTITIY RECOGNITION ),
NER is the process of detecting name such as movie, moneytary
value,organiztion, location, quantities & person
# there are 3 phases of NER - ( 1ST PHASE IS - NOUN PHRASE EXTRACTION
OR NOUN PHASE IDENTIFICATION - This step deals with extract all the
noun phrases from text using dependencies parsing and pos tagging
# 2nd step we have phrase classification - this is the classification
where all the extracted nouns & phrase are classified into category
such as location,names and much more
# some times entity are misclassification
# so if you are use NER in python then you need to import NER_CHUNK
from nltk library
from nltk import ne_chunk

NE_sent = 'The US president stays in the WHITEHOUSE '

NE_tokens = word_tokenize(NE_sent)

#after tokenize need to add the pos tags
NE_tokens

['The', 'US', 'president', 'stays', 'in', 'the', 'WHITEHOUSE']

NE_tags = nltk.pos_tag(NE_tokens)
NE_tags

[('The', 'DT'),
 ('US', 'NNP'),
 ('president', 'NN'),
 ('stays', 'NNS'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('WHITEHOUSE', 'NNP')]

#we are passin the NE_NER into ne_chunks function and lets see the
outputs

NE_NER = ne_chunk(NE_tags)
print(NE_NER)

(S
  The/DT
  (GSP US/NNP)
  president/NN
  stays/NNS
  in/IN
  the/DT
  (ORGANIZATION WHITEHOUSE/NNP))

new = 'the big cat ate the little mouse who was after fresh cheese'
new_tokens = nltk.pos_tag(word_tokenize(new))
new_tokens

# tokenize done and lets add the pos tags also

[('the', 'DT'),
 ('big', 'JJ'),
 ('cat', 'NN'),
 ('ate', 'VBD'),
 ('the', 'DT'),
 ('little', 'JJ'),
 ('mouse', 'NN'),
 ('who', 'WP'),
 ('was', 'VBD'),
 ('after', 'IN'),
 ('fresh', 'JJ'),
 ('cheese', 'NN')]

# Libraries
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Create a list of word
text=("Python Python Python Matplotlib Matplotlib Seaborn Network Plot
Violin Chart Pandas Datascience Wordcloud Spider Radar Parrallel Alpha
Color Brewer Density Scatter Barplot Barplot Boxplot Violinplot
Treemap Stacked Area Chart Chart Visualization Dataviz Donut Pie Time-
Series Wordcloud Wordcloud Sankey Bubble")

text

'Python Python Python Matplotlib Matplotlib Seaborn Network Plot
Violin Chart Pandas Datascience Wordcloud Spider Radar Parrallel Alpha
Color Brewer Density Scatter Barplot Barplot Boxplot Violinplot
Treemap Stacked Area Chart Chart Visualization Dataviz Donut Pie Time-
Series Wordcloud Wordcloud Sankey Bubble'

# Create the wordcloud object
wordcloud = WordCloud(width=480, height=480, margin=0).generate(text)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```


Spider
Boxplot
Sankey
Violin
Wordcloud
Matplotlib
Network
Radar
Visualization
Seaborn
Density
Treemap
Chart
Pie
Donut
Bubble
Datascience
Plot
Brewer
Color
Python
Series
Time
Alpha
Violinplot
Stacked
Area
Pandas
Barplot
Dataviz
Scatter
Parrallel